Fall 2009

Performance Comparison of Hibernate and EclipseLink Technologies for Mapping an Object-Oriented Model to a Relational Database

Lyubov Buleza
Regis University

Regis University
College for Professional Studies Graduate Programs
Final Project/Thesis

Disclaimer

Regis University

School for Professional Studies Graduate Programs

MScSIS Program

Graduate Programs Final Thesis

Project Advisor/ Faculty Approval Form

Student's Name: Lyubov Buleza Program MScSIS

PLEASE PRINT

Thesis Title: Performance Comparison of Hibernate and EclipseLink Technologies for Mapping an Object-Oriented Model to a Relational Database

PLEASE PRINT

Project Advisor Name Ernest Eugster

PLEASE PRINT

Project Faculty Name Donald J. Ina

PLEASE PRINT

Advisor/Faculty Declaration:

I have advised this student through the Project/Thesis Process and approve of the final document as acceptable to be submitted as fulfillment of partial completion of requirements for the MScSIS Degree Program.

Ernest Eugster — *Original Advisor Signature*

28 August 2009 — *Date*

Donald Ina — *Original Module/Class Facilitator Signature*

18 Sept. 2009 — *Date*

Degree Chair Approval if:

The student has received project approval from Faculty and has followed due process in the completion of the project and subsequent documentation.

Tarau — *Original Degree Chair/Designee Signature*

Oct. 7, 2009 — *Date*

Performance Comparison of Hibernate and EclipseLink Technologies for Mapping an Object-Oriented Model to a Relational Database

Lyubov Buleza

Regis University

School for Professional Studies

Master of Science in Software and Information Systems

Acknowledgements

Thank you to Ernest Eugster for his guidance and encouragement throughout the course of this project.

To my family, for their support, patience and care.

List of Tables

List of Figures

Figure 6.5.5 Table joins operation

Abstract

Different tiers of modern applications are built using object-oriented programming for implementing business logic and the relational database model for data storage. To solve the impendence mismatch issue that arises between the object model and relational schema, various Object-Relational Mapping (ORM) tools have been designed. In this research, the performance of two open source ORM technologies, namely Hibernate and EclipseLink, is investigated. Hibernate is a well established middleware solution while EclipseLink, which stems from Oracle's TopLink, is a new product on the persistence landscape. For the purpose of this research, a web-based application was developed and used as a test system. The performance test facilities were integrated into design of the application. The abstract layer introduced into application's architecture with the Spring Data Access Object (DAO), made the system highly modular allowing easily switching between persistence technologies with no alterations in the rest of the application code.

Chapter 1 - Introduction

As object-oriented development has assumed prominence over relational database systems in enterprise data management, object-relational mapping has become a critical task for overcoming the impedance mismatch that occurs between the object and relational models. This chapter outlines the issues of information integration in the modern heterogeneous systems, particularly in object-oriented applications using relational databases for persistent storage. It presents a review of comparative studies of different ORM technologies and identifies the lack of information about comparing leading ORM tools. This research attempts to narrow the existing information gap, as it presents a performance comparison of two ORM frameworks: Hibernate and EclipseLink. The research builds on comparative design method in the form of multiple-case study.

1.1 Statement of the Problem

As the market for databases has expanded, database users, developers and vendors have shown increasing interest in middleware tools such as ORM and its impact on business performance. Over the recent years, middleware has become the central force in addressing the ever increasing complexity of the modern diverse IT environments. Today, middleware facilitates abstraction that hides heterogeneity of the complex systems. As such, “middleware becomes a key software engineering tool” (Issarny, Caporuscio & Georgantas, 2007).

Database middleware plays the central role in the efforts of integrating and transferring data in the database applications. The middleware for database connectivity supports and extends a two-tier client-server architecture to a three-tier model or an n-tier architecture. This is indirect

database connectivity that allows moving all the required business logic to the middle tier, rather than having a more traditional fat client. A three-tier architecture consists of three components:

- the client tier presents data, receives user inputs and manages the user interface.
- The middle tier implements business rules.
- The data-server tier is used for data storage. Typically, it consists of one or more relational databases, such as Oracle and MySQL.

The key advantage to the three-tier architecture is a clear separation of the user interface control and data presentation from the application logic. It allows developing these components independently. Changes in a data server tier will not impact the clients because the clients do not access the data-server tier directly.

Another benefit is improved scalability and availability. To handle these issues, additional servers can be incorporated into the architecture at any tier. Indirect database access provides more options for implementing data security measures, such as hiding the database behind the firewall and employing an authentication and authorization scheme. Introducing an additional tier into architecture, however, impacts system's performance. But this problem can be resolved using dynamic load balancing to distribute server processes among the available servers at runtime.

Different tiers of modern applications can be built using object-oriented programming for implementing business logic and the relational database model for data storage. When combined, both models offer important advantages for implementing complex systems. Object-oriented programming provides "benefits of reusability, robustness, and maintainability" (Russel, 2008); while the relational data model can store and process large volumes of data. In addition, they both are mature technologies that still see continuous improvements.

When persisting objects to a relational database, the problem of impedance mismatch "arises between both the object model and the relational model and between the object programming language and the relational query language" (Zyl, Kourie & Boake, 2006). To close this gap, various object relational mapping tools have been developed. Object-relational mapping provides a bridge between the two models "that allows applications to access relational data in an object-oriented way" (Russel, 2008). Although other technologies, such as Java Database Connectivity (JDBC) or Open Database Connectivity (ODBC) interfaces can be used to access relational data from object-oriented programs, they do not take full advantage of the object behavior in the object-oriented paradigm. Furthermore, these mechanisms allow mixing business logic and database programming.

The programming model that requires incorporating relational concepts into object-oriented code is error-prone and time-consuming. To benefit fully from object behavior, "database-access technology should support separation of concerns, information hiding, inheritance, change detection, uniquing, and database independence" (Russel, 2008). ORM tools support these capabilities and allow a significant reduction in programming effort dedicated to persistence as well as decrease the overall cost of the project.

Today, software architects and designers have a collection of persistence solutions to choose from. Thirty six tools are stated only on the "Open Source Persistence Frameworks in Java" list in the Java-Source.net (n. d.). Yet, with so many persistence frameworks available Hibernate became the de facto standard in the industry, closely followed by Oracle's TopLink. While Hibernate is an open source solution, TopLink is a commercial product. In 2006, Oracle donated the full source code and test suits of TopLink to the Eclipse Foundation, a non-profit

organization. Supporting an Integrated Development Environment (IDE) for programming languages, such as Java, C++ and PHP, EclipseLink was developed as an open source project.

While the academic literature on middleware and its role in data connectivity is extensive, little exists on evaluating and comparing different persistence tools. Martin (2005) developed TORPEDO, the performance benchmark for ORM software. TORPEDO supports frameworks such as Container Managed Persistence (CMP), TopLink and Hibernate. Ziemniak, Sakowicz and Napieralski (2007) compared three technologies supporting ORM for J2EE platform, namely Hibernate, Java Persistent Objects (JPOX) and Java Persistence API (JPA). More recently, O'Neil (2008) compared the capabilities of Hibernate to Microsoft's new ORM technology – Entity Data Model (EDM) for .NET systems.

After an extensive review of the academic and trade journals, this researcher did not discover any publications assessing performance of such popular ORM framework tool as TopLink. Besides filling a gap in the literature, this researcher thinks the time is right for such a study. The recent launch of EclipseLink, which is built on about twelve years of TopLink experience, affords an opportunity to compare EclipseLink with Hibernate. Little research on ElipseLink exists.

1.2 Thesis Statement

This study attempts to redress the imbalance in academic literature by examining the role of database middleware tools and comparing the performance of Hibernate technology for expressing SQL queries in terms of object-oriented idiom and EclipseLink persistence framework for mapping an object-oriented model to a relational database.

1.3 Scope of the Project

The study will investigate the capabilities of Hibernate, comparing its performance characteristics with those of EclipseLink, a new technology. For this purpose, a web-based application for supporting and running the organization of football events at a given university will be developed and used as a test system. Database access in the test system will be implemented using Hibernate and EclipseLink frameworks in parallel. This will enable the researcher to compare and contrast the following characteristics of both technologies: ease-of-use, flexibility, and execution performance.

1.4 Significance of the Study

This study will contribute to the object-oriented mapping frameworks domain of knowledge. Acquired knowledge about strengths and weaknesses of Hibernate and EclipseLink technologies will assist software engineers with choosing the appropriate technology for mapping an object-oriented model to a relational database. Observations, made during a practical test, will help software developers to prevent programming mistakes and develop better quality software.

1.5 Research Methodologies

To compare the technologies, this researcher used a comparative design method that takes a form of multiple-case study. According to Bryman and Bell (2003), a comparative design entailed the study using more or less identical methods of two or more cases. When it is applied in relation to a qualitative research strategy, it takes the form of a multiple-case study. A multiple-case study occurs whenever the number of cases examined exceeds one. In this project, a multiple-case study constitutes a common research design that takes two implementation

processes as cases for comparison. The main argument in favor of the multiple-case study is that the comparison of these two cases may suggest the concepts that are relevant to the theory of the most appropriate use of Hibernate or EclipseLink technologies according to their characteristics.

1.6 Success Criteria

The success criteria for this study will be measured by the efficiency of software prototype developed that will be used as a test system. The practical test will provide a concrete basis for drawing conclusions about advantages and disadvantages of Hibernate and EclipseLink technologies.

1.7 Summary

Integration of information from different data models always presents a challenge and accounts for considerable amount of programming effort. In this chapter, the researcher described the concept of object-relational mapping. ORM technologies reside between object-oriented application and relational database to solve the problem of impedance mismatch between the two models.

To enrich the domain of knowledge of the middleware technologies that play an important role in the Software Engineering discipline, this research compares performance characteristics of Hibernate and EclipseLink – the two open source ORM tools. The research presents implementation processes of both technologies that are used as cases for comparison.

Chapter 2 - Operational and Development Environment

This chapter discusses the technologies used in the development of a web based application that serves as a test system for comparing Hibernate and EclipseLink. The stack of technologies includes Hibernate and EclipseLink ORM tools, MySQL database, Spring Framework for the Java platform, Java Server Pages (JSP), Java Server Faces (JSF), servlets, Apache Tomcat servlet container and Ant build tool.

2.1 Hibernate – Object-Relational Persistence

Hibernate, "the first popular, fully featured open source ORM solution" (Johnson, 2005), was released in 2002. Hibernate was developed in the effort to overcome problems in the existing persistence mechanisms. Persistence interfaces, such as JDBC or ODBC, allow little separation of concerns and force developers to work with relational concepts in the object-oriented environment. These mechanisms fail to support information hiding, inheritance, or change detection. Furthermore, neither provide database independence. Implementation of these interfaces require considerable amount of programming time and resources. According to Ziemniak et al. (2007), building a persistence layer based on JDBC/SQL, accounted for almost thirty percent of the application total programming effort.

With Hibernate, relational database structures are presented to the application as objects in the native object programming language. For instance, in Java, these domain classes are known as Plain Old Java Objects (POJOs), as they do not need to contain specific persistence behavior. Johnson (2005) stated that transparent persistence remains a goal, but best ORM tools come remarkably close to achieving it. Hibernate offers the level of transparency that treats

POJOs in the way that they have no notion that they are persistent objects. It provides separation of concerns by freeing domain objects from the task of managing their persistence; thus enabling isolation of business logic from persistence operations. This paradigm brings increased productivity as it eliminates the need to write verbose and error-prone persistence code.

Hibernate has a flexible architecture that supports different component stacks depending on the scope of the application. Figure 2.1.1 illustrates a high-level view of Hibernate's architecture.

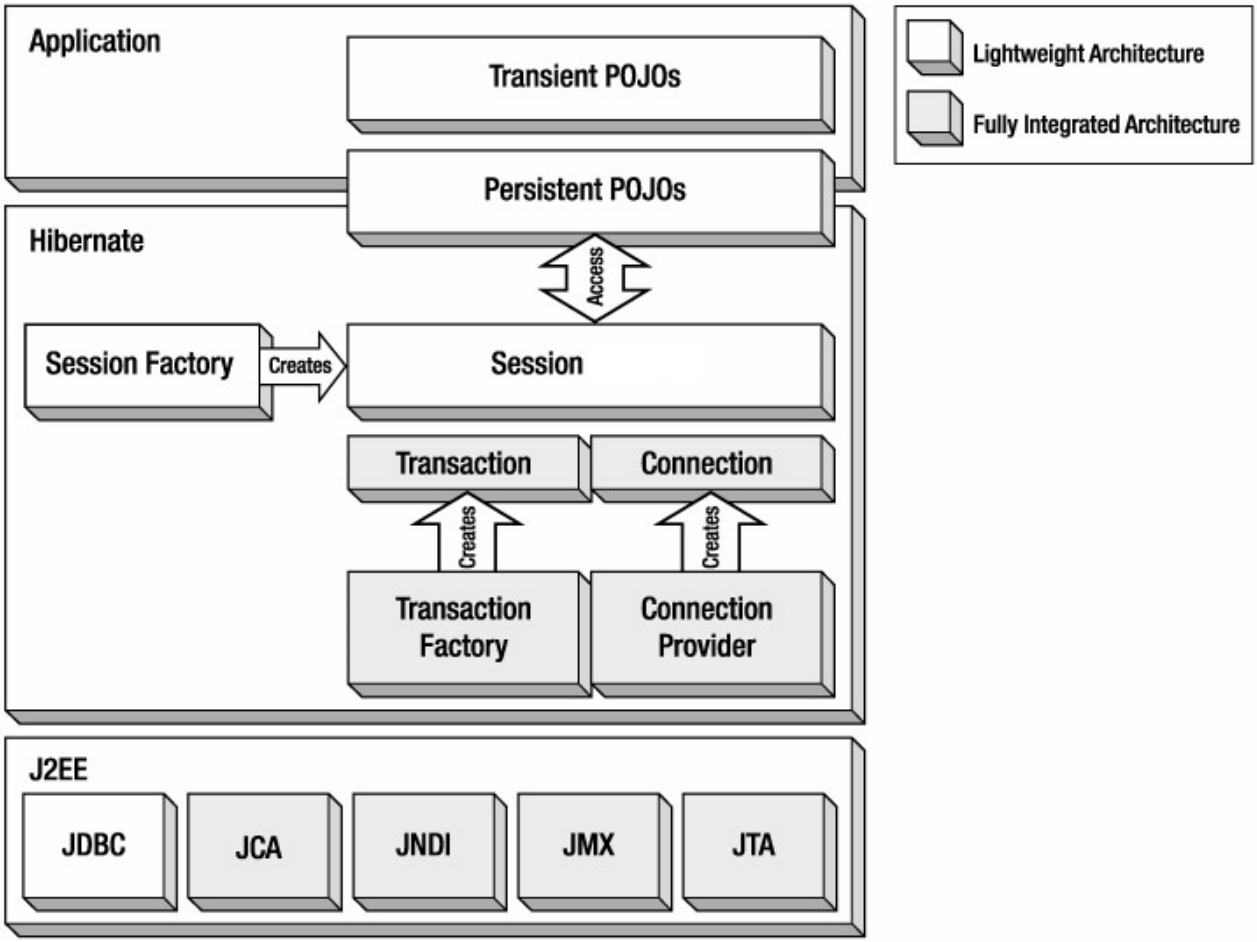

Figure 2.1.1 Hibernate's architecture (Sam-Bodden, 2006)

Sitting between J2EE database connectivity interfaces and application, Hibernate consists of various elements. The basis of integration resides in the SessionFactory; a threadsafe cache

that holds complied mappings for a particular database. Normally, an application has a single SessionFactory. Threads servicing client requests obtain a Session from the SessionFactory. A Session is a single-threaded, short-lived object representing communication between application and database. In a multiuser environment, multiple Session instances are used to isolate users from each other. Hibernate represents relational data to the application by the means of domain objects. The domain objects model classes can be implemented as JavaBeans, they are also referred to as POJOs. These objects hold persistent state and business functions, and they are associated with only one Session. Once the Session is closed, the objects are freed for use in any application layer. The objects can be in a transient or persistent state. Transient objects are not associated with a Session. The state of a transient object has not been saved to a database table and the object has no associated database identity, that is no primary key has been assigned. Upon a successful execution of the Save or Update methods, a transient object becomes persistent. Persistent object obtains a database identity and corresponds to a row in a database table.

The extended Hibernate architecture abstracts the application away from the underlying JDBC API: allowing Hibernate to manage such aspects of persistence as transaction and connection pooling. Transaction represents an atomic unit of work that is either committed or rolled back in its entirety. The concept of transaction facilitates database consistency. Hibernate's Transaction is a Transaction class, an instance of which is obtained from the Session instance. Session wraps a JDBC connection and works as a factory for Transaction. Russel (2008) noted that at any given time, there is only one active Transaction instance for a Session, but a Session can handle multiple transactions serially. The SessionFactory can include a connection pooling. ConnectionProvider produces a pool of JDBC connections, hands them out

on request and returns them to the pool when they are not needed anymore. Connection pooling promotes efficient database access. To customize the persistence layer, Hibernate can implement a number of optional extension interfaces, such as Java Naming and Directory Interface (JNDI) and Java Transaction API (JTA).

Sam-Bodden (2006) observed that the reason behind the Hibernate's success is "simplicity of its core concepts," Figure 2.1.2 illustrates an overview of how Hibernate works.

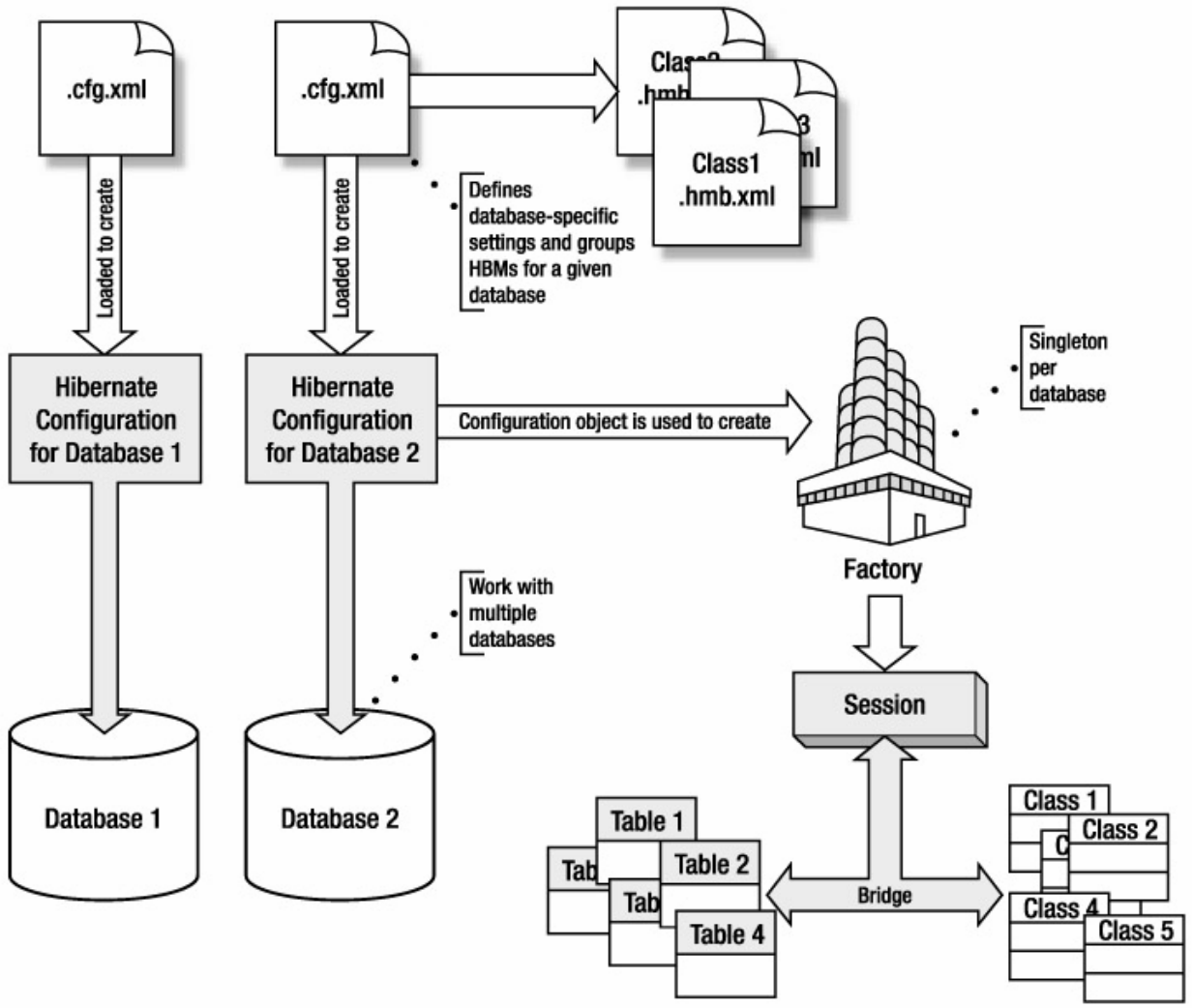

Figure 2.1.2 Hibernate in work (Sam-Bodden, 2006)

As noted, the Session is the core concept of the Hibernate persistence service. It provides a bridge for the application and the database to interact. To connect to each database used by an

application, Hibernate needs configuration information and mapping data that identifies classes mapped to a given database. Database-specific configuration files, along with corresponding class mappings are managed by the SessionFactory. The SessionFactory is employed to create Hibernate Sessions. As Sam-Bodden (2006) noted, the SessionFactory is a heavyweight object, whose creation is an expensive operation. The SessionFactory, therefore, should be created only once and made available to the application to perform persistence operations.

When dealing with the Hibernate's persistence, it is important to understand the lifecycle of the mapped object as it is implemented in Hibernate. Sam-Bodden (2006) described three possible states for a Hibernate mapped object, namely transient, persistent and detached. Figure 2.1.3 shows the states of the object and the methods for transition the object from state to state.

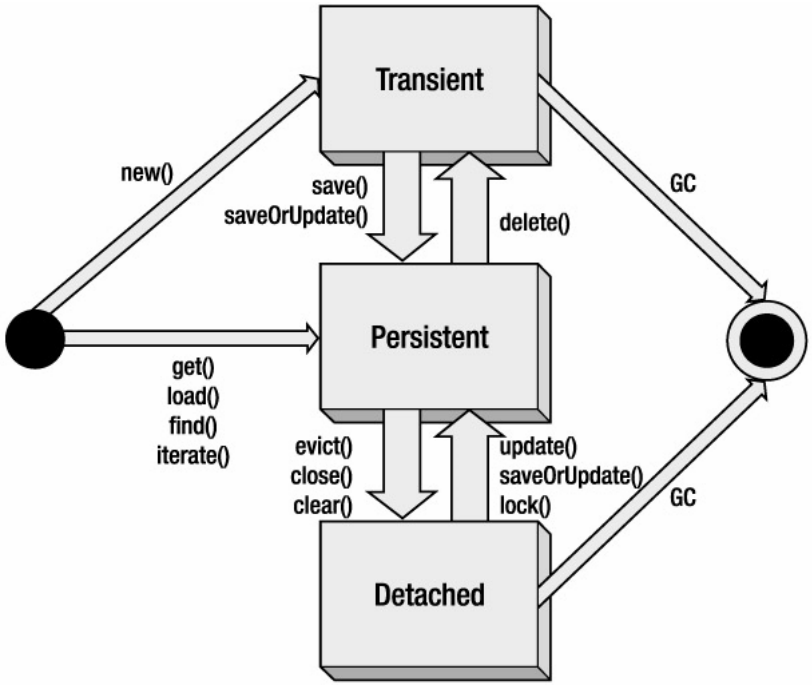

Figure 2.1.3 Persistence lifecycle (Sam-Bodden, 2006)

In the transient state, the object has no association with a database table. Transient objects are non-transactional, that is they do not participate in any of transactions associated with the Session. Hibernate Session provides methods to transform the object from one state to another.

For example, the save() or saveOrUpdate() methods transit the object from transient to persistent state, while delete() method performs the opposite operation making transient object persistent.

In the persistent state, the object has a database identity. Persistent objects are transactional, they participate in transactions associated with the Session. When transaction completes the object state is synchronized with the database.

In the detached state, the object is no longer associated with the Session. The detached object has a representation in the database, but changes to the object will not be reflected in the database, and vice versa. The object becomes detached "after a transaction completes, when the session is closed, cleared, or if the object is explicitly evicted from the Session cache" (Sam-Bodden, 2006).

Each SessionFactory is configured to work with a certain database management system by utilizing one of the Hibernate's dialects. According to Hibernate (2006), the latest version, Hibernate 3.3.2 GA, supports twenty three database dialects. Additionally, Hibernate comes with its own Hibernate Query Language (HQL) that is database-independent. These facilities allow to easily port application with Hibernate persistence from one database platform to another.

Hibernate needs to know which objects correspond to which tables. This information is placed into the mapping files. Hibernate supports two ways to express the mappings: XML mapping files and Annotations. XML mapping information is kept in the external file, while using Annotations permit to include metadata inline with the source code for the POJOs. Each approach has its own benefits and drawbacks. XML-based mapping is the older technique that allows greater flexibility when the database schema changes. However, XML mapping is more verbose than mapping with Annotations. According to Minter and Linwood (2006), Annotations feature was introduced in Java 5. Annotations-based method is more intuitive and compact than

its XML equivalent. On the negative side, using Annotations restricts application code to Java 5 environment. Additionally, Annotations are hard to maintain since business or schema changes require extensive alterations in the application code.

2.2 EclipseLink Persistence Framework

EclipseLink traces its origins to 2006 when Oracle donated the source and tests cases of its persistence framework Oracle TopLink to the Eclipse Foundation. Oracle not only made its code contribution to the open source community, but proposed “to lead a new Eclipse run-time project to provide a set of persistence services” (Oracle Corp., 2007). Today, EclipseLink offers a comprehensive persistence platform that is not limited to ORM. It also supports services such as Object-XML Binding (EclipseLink MOXy) that allows binding Java classes to XML Schemas; Service Data Objects (EclipseLink SDO) for building data object models, and Web Services for RDBMS (EclipseLink DBWS) that allow accessing RDBMS artifacts via a web service. In this research, the object-relational service that is implemented by the means of EclipseLink JPA component was studied.

EclipseLink ORM functionality is based on the JPA 2.0. Similar to Hibernate, JPA utilizes POJOs to represent persistent objects. In the persistence context, POJOs are also known as entities. Since JPA entities are Plain Old Java Objects, they do not need to extend any specific parent class or implement any specific interface. A Java class is specified as a JPA entity with the @Entity annotation. Figure 2.2.1 shows the relationships between the core components of the JPA architecture.

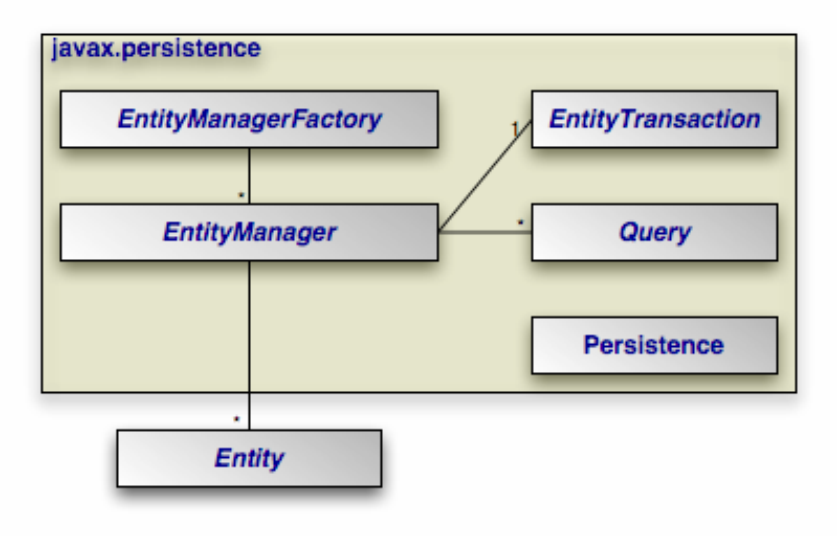

Figure 2.2.1 JPA architecture (IBM Corp., n. d.)

JPA framework builds on the following concepts:

- Entity object is a lightweight persistent domain object that represents a database record.
- EntityManager provides persistence services, including transaction management. The EntityManager is at the heart of the JPA persistence, it enables API calls to perform operations on an entity. An instance of the EntityManager represents persistence context. Kodali, Wetherbee and Zadrozny (2006) stated that persistence context permits the EntityManager to track modified, created, and removed entity instances, and to maintain entity instances in accordance with changes committed by external transactions performing concurrently with the EntityManager's own transaction.
- EntityManagerFactory creates EntityManagers. According to Oracle Corp. (2006), the configuration for each EntityManager is defined separately as a persistence unit. A unique name of each persistence unit allows application to

differentiate between the EntityManagerFactory objects. The configuration of the persistence unit is defined in a persistence.xml file.

- EntityTransaction groups the operations on persistent data into units that completely succeed or completely fail. This all-or-nothing technique supports data integrity and maintains database in a consistent state.
- Query allows to find persistent objects that meet certain criteria. To define queries, JPA supports Structured Query Language (SQL), as well as Java Persistence Query Language (JPQL). JPQL is database vendor-independent, as such it provides portability of the application.
- Persistence presents the Java class that is used to obtain EntityManagerFactory instances in a vendor-neutral way.

An EntityManager instance is associated with a persistence context. Figure 2.2.2 illustrates how the entity instance and its life cycle are managed within the persistence context. In its life, the entity may go through the new, managed, detached, and removed states.

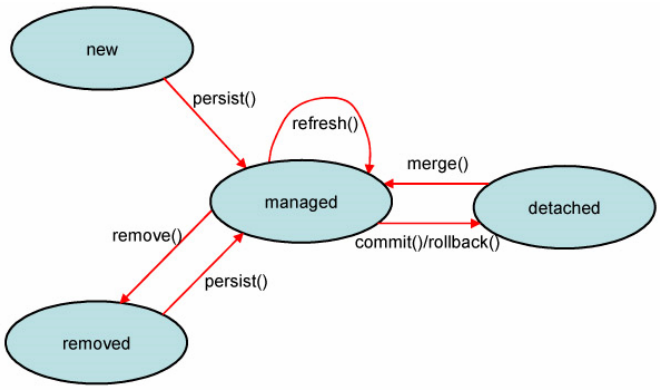

Figure 2.2.2 Entity life cycle for a persistence context (Wahli et al., 2008)

A new entity instance is created by the means of the entity's Java constructors. At the moment of construction, the entity is in the new state, it does not have yet a persistent identity as it is not associated with an EntityManager's persistence context.

This entity class turns into a persistent object after instantiating of the EntityManager and successful invocation of the persist() method. As Wahli et al. (2008) noted, when an entity is attached to the EntityManager, it moves to the managed state, and the manager tracks any changes and updates to the entity and flushes these changes to the database.

The entity remains in a managed state until the persistence context, that contains it, is closed or the entity is removed from the database. When a persistence context is closed, the entity becomes detached from the persistence context and associated EntityManager. When the entity is detached, it is unmanaged and it is not exposed to the change tracking by the EntityManager.

The entity becomes removed after invocation of its remove() method. The row corresponding to its persistent state will be removed when the context transaction commits.

JPA framework, similarly to Hibernate, supports two ways of defining the object-relational mapping of an entity: by the means of annotations or, alternatively, in an external orm.xml file. The XML mapping file has to be defined in the persistence.xml configuration file with the mapping file element.

2.3 Spring Framework

To compare Hibernate and EclipeLink, other tools are needed for the test environment. The Spring Framework is an open source framework that was developed to address complex design issues in enterprise application development. Spring offers an alternative solution to Enterprise JavaBeans (EJBs). Spring combines such concepts as Inversion of Control (IoC), dependency injection (DI), and aspect-oriented programming (AOP). Johnson et al. (2005) describe the Inversion of Control as an architectural pattern that supports an outside entity that

manages objects' dependencies and wire together related objects. In Spring, IoC is implemented by the means of a lightweight container that provides wiring for the Spring beans. To decouple class dependencies, Spring uses dependency injection that comes in two types: constructor injection and setter injection. Constructor injection provides dependencies through the Spring class constructor at instantiation of the class. Setter injection uses setter methods to inject the dependency. These dependencies are defined in an external configuration file.

Figure 2.3.1 shows the core components that comprise the Spring Framework. The Spring architecture is highly modular, it allows to implement different combinations of components.

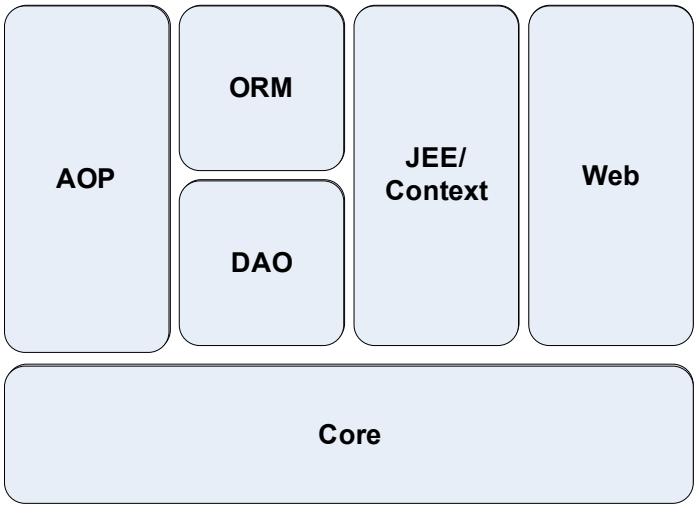

Figure 2.3.1 Spring Framework modules (Giametta, 2009)

Giametta (2009) described the six Spring modules as follows:

1. Core, responsible for governing the framework. One of the most important aspects of Spring is its utilization of IoC through DI.
2. JEE/Context, responsible for controlling of Spring objects. Additionally, it supports services such as EJBs and internationalization (i18n) messages.

3. Aspect-Oriented Programming (AOP), responsible for implementing AOP principles. In Spring, aspects are parts of the application that allow separation of concerns through modularization. For instance, Spring aspects isolate persistence logic from business logic.
4. Object-Relational Mapping (ORM), responsible for integration of ORM tools, such as Hibernate, TopLink, and JPA.
5. Data Access Object (DAO), providing abstraction layer for JDBC API. DAO allows to easily change JDBC database access, ORM technology that is managing DAO layer, or database solution without breaking the application code written in other Spring modules.
6. Web, responsible for integration features for building web components. This modules include Spring's MVC (model-view-controller) package, one of the most popular approaches to designing and developing web applications today. MVC pattern builds on the following key elements:
    - Model – the business tier's domain model with its related services. Model contains the data needed to render the View and it is populated by the Controller.
    - View – provides the user interface for displaying and manipulating the model. The View can be presented as HTML views generated with Java Server Pages (JSPs).
    - Controller – the logic that provides interaction between the Model and the View.

2.4 Java Server Pages

Java Server Pages technology is a part of J2EE. JSP is used “to create dynamically generated web pages in response to a client request” (Goncalves, 2009). JSP consists of HTML scrip with chunks of Java code embedded into it. Java code modules in the JSP are called scriplets. JSP are based on the servlet. Servlets have capabilities to accept HTTP requests from clients and create dynamic responses to those requests. Servlets and JSP can use various server-side resources, such as EJBs, databases, web services and other application components.

JSP are processed on a server in a servlet container. Goncalves (2009) described the following tasks performed by the servlet container to manage the JSP’s life cycle:

- JSP’s code is compiled into a servlet,;
- JSP is loaded and initialized;
- Clients requests are processed and dispatched to the JSP;
- The response is returned to a client as a HTML page;
- JSP is unloaded when the server shuts down.

2.5 Ajax4Jsf Framework

Java Server Faces (JSF) is a relatively new technology that was introduced to further simplify web application development. JSF is a supporting technology that can be used in conjunction with JavaBeans, JSP or servlets. Utilizing JSP or servlets for building the presentation layer limits the user interface to the capabilities of HTML. Mukhar, Zelenak, Weaver and Crume (2006) pointed to another drawback to using these technologies – the user interface components are connected to the business logic. The JSF handles this problem by decoupling the user interface and business logic by providing user interface components as

reusable objects. With JSF, there is no need to worry about the syntax of page layout as the user interface components can be simply dropped into application. Then, a custom render kit and rendering process convert the components into appropriate page layout code. The flexibility and reusability of the user interface components offered by JSF allow developer to concentrate on business logic.

AJAX (Asynchronous JavaScript and XML) is a new technique for creating interactive web applications. According to W3School.com (n. d.), AJAX supports direct communication of JavaScript with the server through the XMLHttpRequest object. This allows JavaScript to exchange data with a web server, without reloading the page. By reloading only the needed data, AJAX provides faster response to the client application.

Ajax4Jsf Framework, an open source project led by JBoss community, presents an integration of JSF with AJAX. Ajax4Jsf allows to add AJAX capabilities to the JSF. In addition to the existing JSF tags, Ajax4Jsf supports new tags that enable AJAX functionality. As Heffelfinger (2007) notes, utilizing Ajax4Jsf allows to re-render one or more components in a page rather than re-rendering the whole page. This capability makes a web application highly interactive. Ajax4jsf can be downloaded from http://labs.jboss.com/jbossajax4jsf/downloads

2.6 MySQL Relational Database

MySQL is a popular open source relational database. According to Gilmore (2008), MySQL offered a great flexibility of running on fourteen platforms. MySQL also provides different types of mechanisms for managing data, they known as storage engines. Some of the storage engines supported by MySql include MyISAM, MEMORY, InnoDB (the default on Windows), and MERGE. The recent version of MySQL includes Falcon, a high-performance

storage engine that can be employed for large-scale deployments on multiprocessing systems. A number of MySQL features are dedicated to the performance improvement. For example, the full-text indexing and searching greatly enhances the performance of mining data from text-based columns. Query caching also contributes to the speed improvements. Replication allows to spread the load among multiple servers, to increase database availability and robustness. MySQL also facilitates a number of security features, such as authentication and authorization scheme, audit of the server's user accounts, secure connection through Secure Sockets Layer (SSL) etc.

The graphically based management tools available for MySQL greatly simplify many tasks of database management. The most popular GUI solutions are MySQL Administrator and MySQL Query Browser. MySQL Administrator constitutes all aspects of database maintenance, such as user management, health monitoring, data backup and restoration, and log monitoring. MySQL Query Browser offers an efficient means for creating databases, tables, views, and stored procedures, as well as forming and executing queries and managing data.

The latest database server version, MySQL 5.4, and GUI Tools package are available for download at http://dev.mysql.com/downloads/ .

2.7 Apache Tomcat Servlet Container

To deploy a web application, a servlet container is required for executing servlets and JSP pages. Apache Tomcat is a free source servlet container that is fully compliant with the Servlet and JSP specifications published by Sun Microsystems. According to Moodie (2007), Tomcat 6 was the first container supporting the Java Server Faces 1.2 specification.

Tomcat 6.0.20 is available from http://tomcat.apache.org/download-60.cgi . The easiest way to install and run the server is by using the zipped file. The content of the Tomcat zipped file

needs to be uncompressed to a convenient directory, for example c:\tomcat. The server starts by executing the c:\tomcat\bin\startup.bat file.

2.8 Ant Build Tool

Ant is a Java-based build tool from the Apache project used to build and deploy applications. Ant is based on a build file that describes the project and the dependencies and relationships within it. Ant's build file is written in XML. The build file consists of modules associated with the discrete steps in the build process. This allows to add new entries to the build file without affecting other steps in the build process.

The binary distribution of Ant 1.7.1 is available from http://ant.apache.org/bindownload.cgi. To install Ant, the distribution is unpacked to a convenient location. For example, c:\ant. To make Ant available from any directory, it is good to add ANT_HOME environment variable and set its value to c:\ant.

2.9 Summary

This chapter showed that the development of a web-based application requires a rich set of technologies. Hibernate and EclipseLink provide powerful, high performance object/relational persistence and query services. The researcher also described the main concepts of the technologies employed for building the test application: MySQL database, Spring Framework for the Java platform, Java Server Pages (JSP), Java Server Faces (JSF), servlets, Apache Tomcat servlet container and Ant build tool. The stage is now set to compare the two ORM tools.

Chapter 3 - Requirements Description

This chapter describes the requirements for the application that is used as a test system for comparison of performance of Hibernate and EclipseLink ORM technologies.

3.1 Use Case Diagrams

Figures 3.1.1-3.1.3 illustrate the Use Case Diagrams for the application prototype for managing football events at a given university. The application is named Grasshopper.

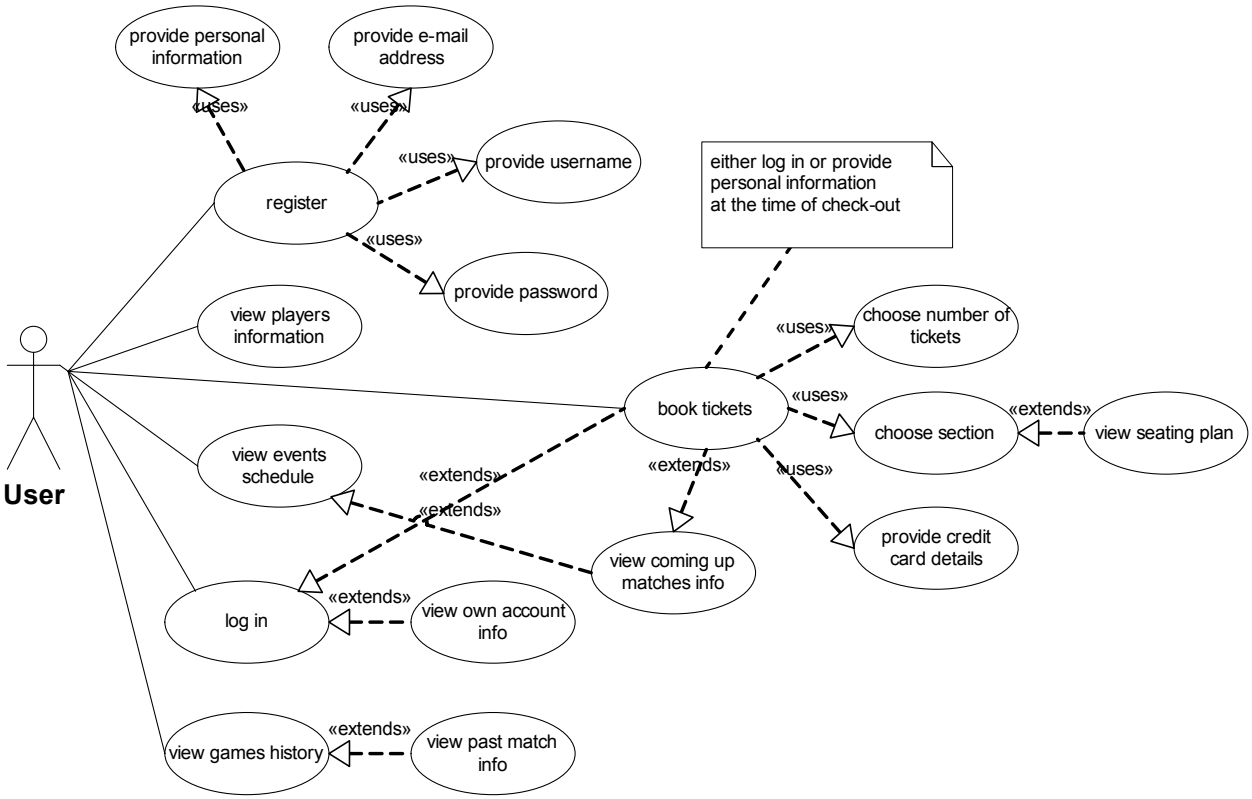

Figure 3.1 User Use Case

The front end of the Grasshopper application should provide a web interface for the users, offering a variety of functions including looking up the upcoming events and the match history, booking tickets for a particular match, and viewing players' information. The visitors of the site should be able to register for the personal accounts. An account should hold all

information needed for booking tickets, so the users do not need to re-enter information during every booking. To complete the registration, the user needs to provide personal information, e-mail address, choose a username and a password. Users can view general information about a Grasshoppers team player, view the matches schedule and look up information about the matches they are interested in. The users should also have access to the previous games history and detailed information about the past games.

The registered users should be able to book tickets for the upcoming events. To complete a booking, the user must pick a match and a section from a seating plan, select the number of tickets and provide the credit card details. Implementing a secure online payment with credit card verification is beyond the scope of this study. Therefore, it is assumed that upon the successful completing of the booking procedure, the user obtains a booking conformation number. The user then completes payment for the tickets at a booking office, providing the booking identification number.

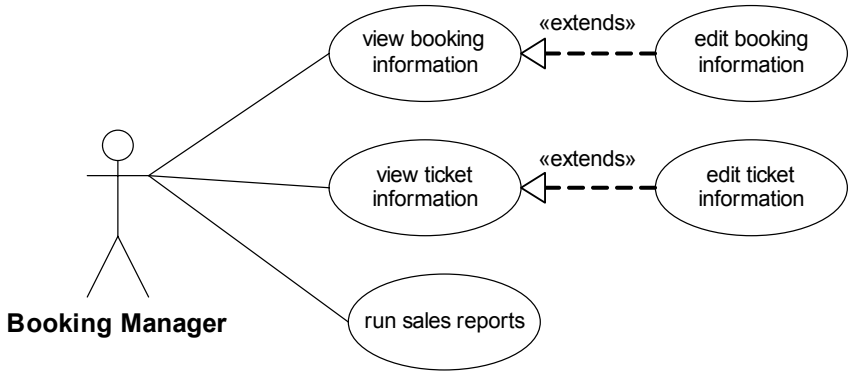

Figure 3.2 Booking Manager Use Case

The booking system should have two types of privileged users – the Booking Manager and the Team Manager. The functionally available to the Booking Manager include viewing and editing the booking information, viewing and editing ticket information and running the sales

reports. The Team Manager should be able to view and edit event information, view and edit player information, and view bookings for a particular event.

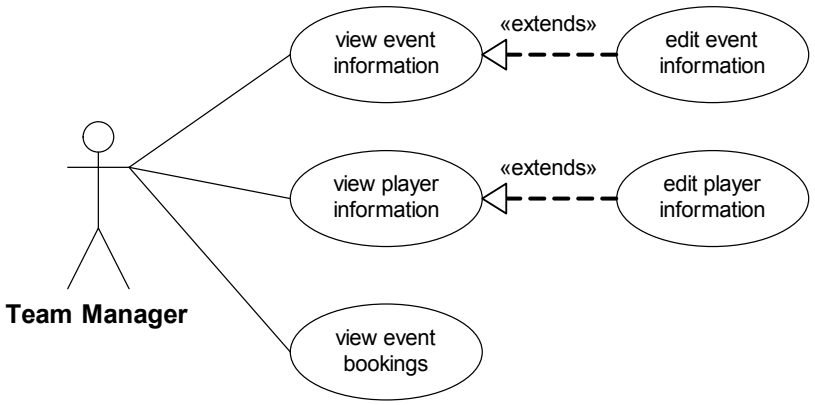

Figure 3.3 Team Manager Use Case

3.2 Design Assumptions

The following assumptions are made about the ticket booking system:

1. To confirm tickets reservation, credit card details are required. No advance credit card charges are debited. The tickets must be collected at the ticket office and paid for at the time of collection. No tickets collection will result in full charge.
2. The tickets booking is final and cannot be cancelled or changed.

Considering the fact that development of the web application is not the purpose of this research but rather the means to achieve the aim of the study, the graphic user interface will be developed only for the User actor.

3.3 Summary

This chapter presented the requirements for the Grasshopper web based application that serves as a test system for comparing Hibernate and EclipseLink. The functional requirements as

described through the Use Case diagrams make clear the parameters for the software development. Several assumptions on business rules were also identified.

Chapter 4 - Design Description

Now that the components for the test environment are known (Chapter 2) and requirements identified (Chapter3) we can address design issues. The design of the application is crucial to the success of the information system and needs to be carefully planned. This chapter covers activities focused on the design of the relational database structure, object model and architecture of the Grasshopper application.

4.1 Relational Model

Figure 4.1.1 illustrates the building blocks of the Grasshopper relational data model: entities, attributes, relationships, and constraints.

The twelve entities identified to represent the relational Grasshopper model are: User, Account, Authorities, Credit_card, Booking, Ticket, Seat, Section, Seatprice, Matches, Players, and Participation. In the relational model, each table should contain a primary key constraint to uniquely identify each record in the table. A primary key can be natural or surrogate. A natural primary key is presented by the meaningful table's attributes that can have a unique characteristic. A surrogate primary key is a sequence generated by the database. In the Grasshopper schema, the most tables have surrogate primary keys. When the relational model is translated into the object model, the surrogate keys support cleaner code than their natural counterparts.

The User table holds the personal and contact information about the users of the Grasshopper booking system. The Account table stores information about the users' accounts. A user can have one or zero accounts, this association results in one-to-one relationship between the User and the Account tables. In a relational model, a foreign key constraint is used to enforce

the relationships between the tables. Therefore, a User table has a foreign key (user_id column) into Account table. The Authorities table is designed to keep information about the users' authorization. It is linked with the Account table through the username foreign key. The username must be unique for each account. This requirement is enforced by the unique constraint. The one-to-many association between the User table and the Credit_Card table is provided by the user_id foreign key in the Credit_Card table.

The stadium is divided into sectors, information about which is placed into the Section table. The Seat table keeps the information about the seats of the Grasshopper home stadium. Each seat belongs to a specific section, but each section is comprised of multiple seats, that explains why the Section table is associated with the Seat table as a one-to-many and the Seat table has a foreign key constraint on its section_id column.

The Matches table holds information about the past and upcoming games, which is why the score and analysis columns are nullable. The information about the Grasshopper team players is stored in the Player table. Because one player can participate in many matches as well as one match needs participation of many players, the Matches and the Player tables land in the many-to-many relationship. A many-to-many relationship introduces the data redundancy into the database; therefore designing for this kind of association is not a good practice. To eliminate this problem, the Participation table is introduced. The Participation table links the Player and the Matches tables through the one-to-many relationship.

The Booking table stores information about the tickets bookings. The booking can be made for one or more tickets, but a ticket can exist without booking, which is why the one-to-many relationship between the Booking and the Ticket tables is non-identifying. As a result, in the Ticket table, the book_id, a foreign key column that references Booking table, is nullable.

The Seatprice table holds information about the ticket price. The price depends on the particular match and section combination. This puts a unique constraint on the combination of match_id and section_id columns in the Seatprice table.

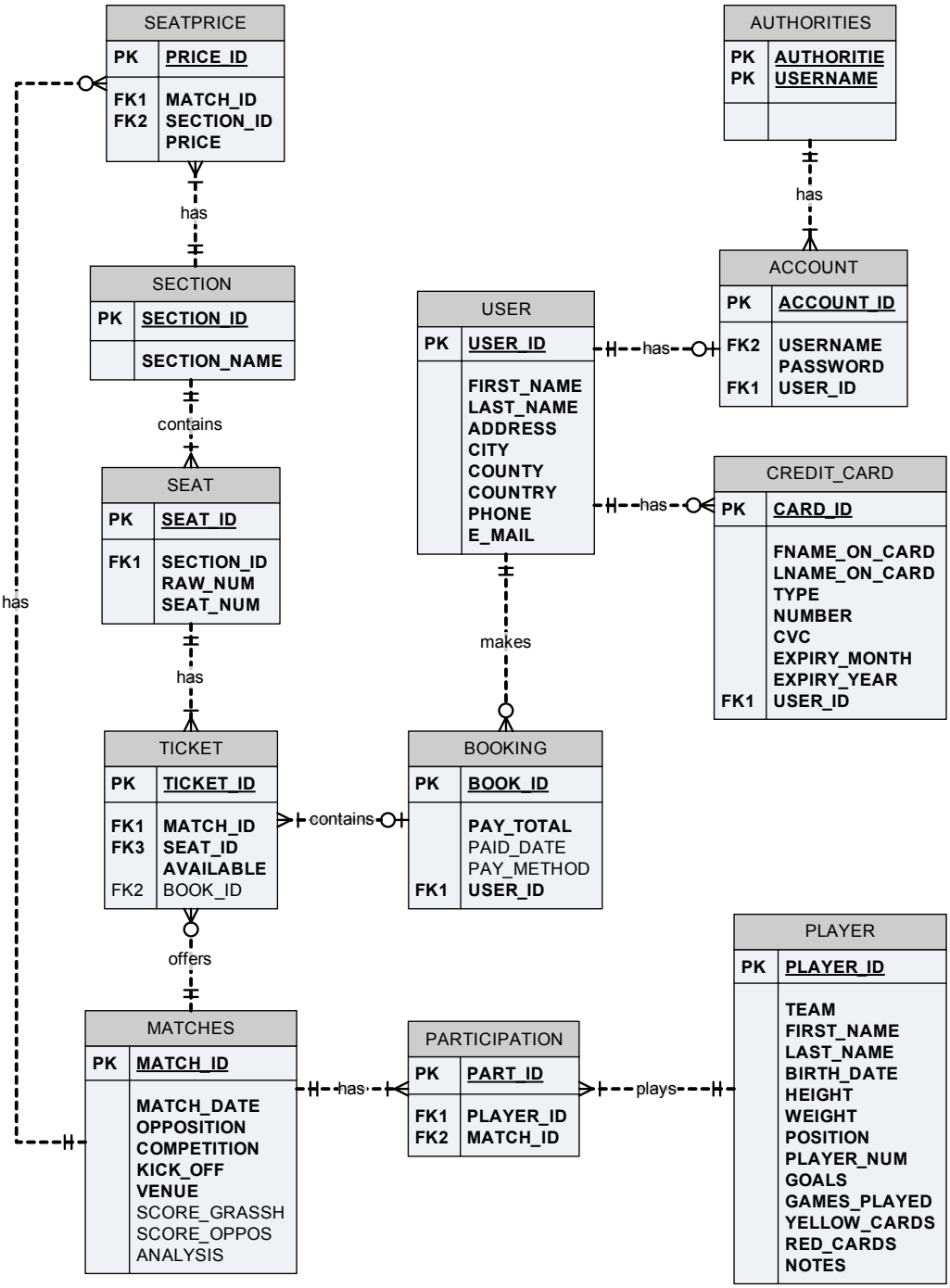

Figure 4.1.1 Entity-relationship diagram for the Grasshopper database

4.2 Object Model

Figure 4.2.1 shows the object model of the Grasshopper application, presented by the means of the class diagram. The twelve classes of the Grasshopper application correspond to the twelve tables of the Grasshopper data model. The classes are related to each other with the same associations that their relational counterparts.

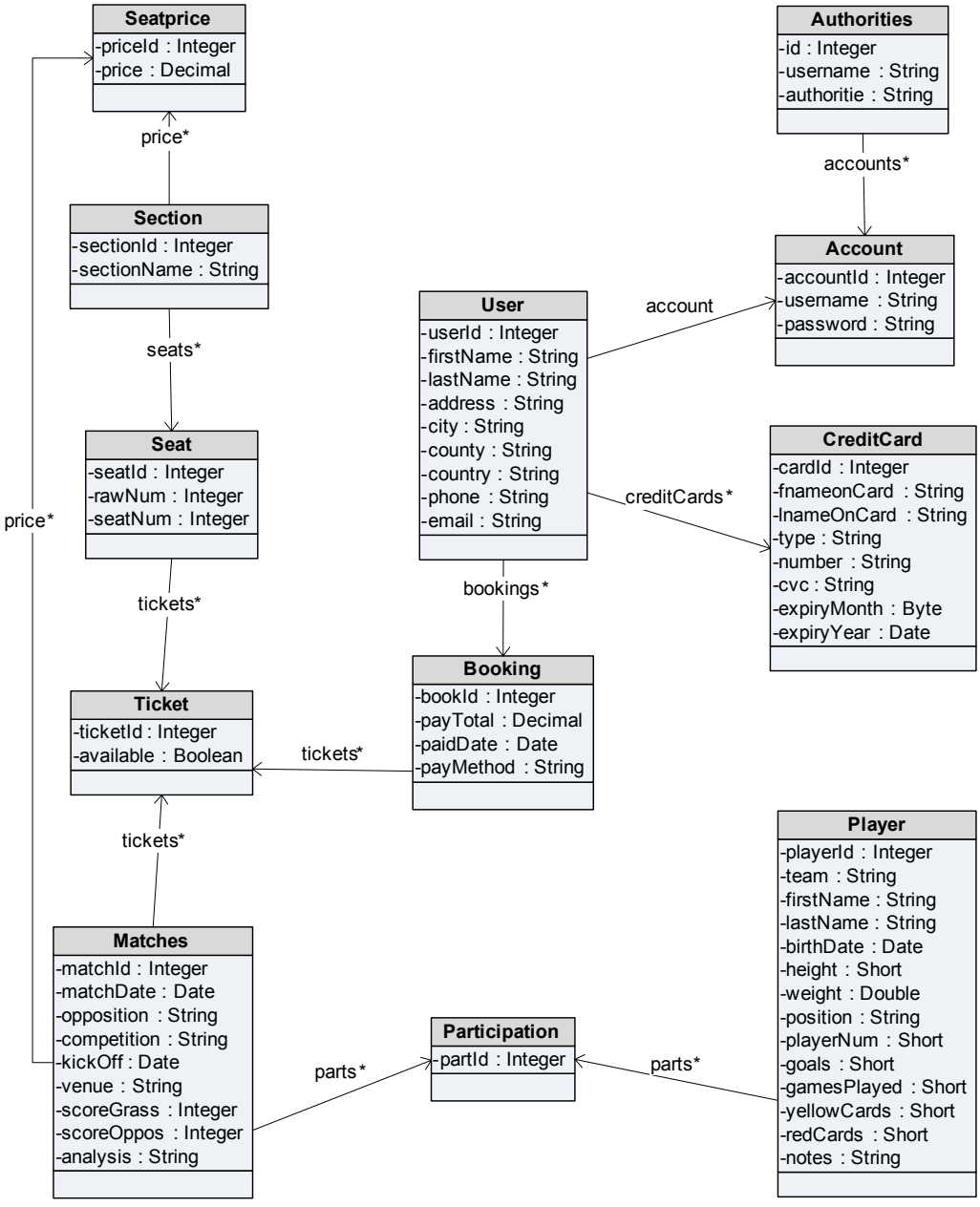

Figure 4.2 Class diagram for the Grasshopper application

4.3 Application Architecture

The Grasshopper application architecture is depicted in the Figure 4.3.1.

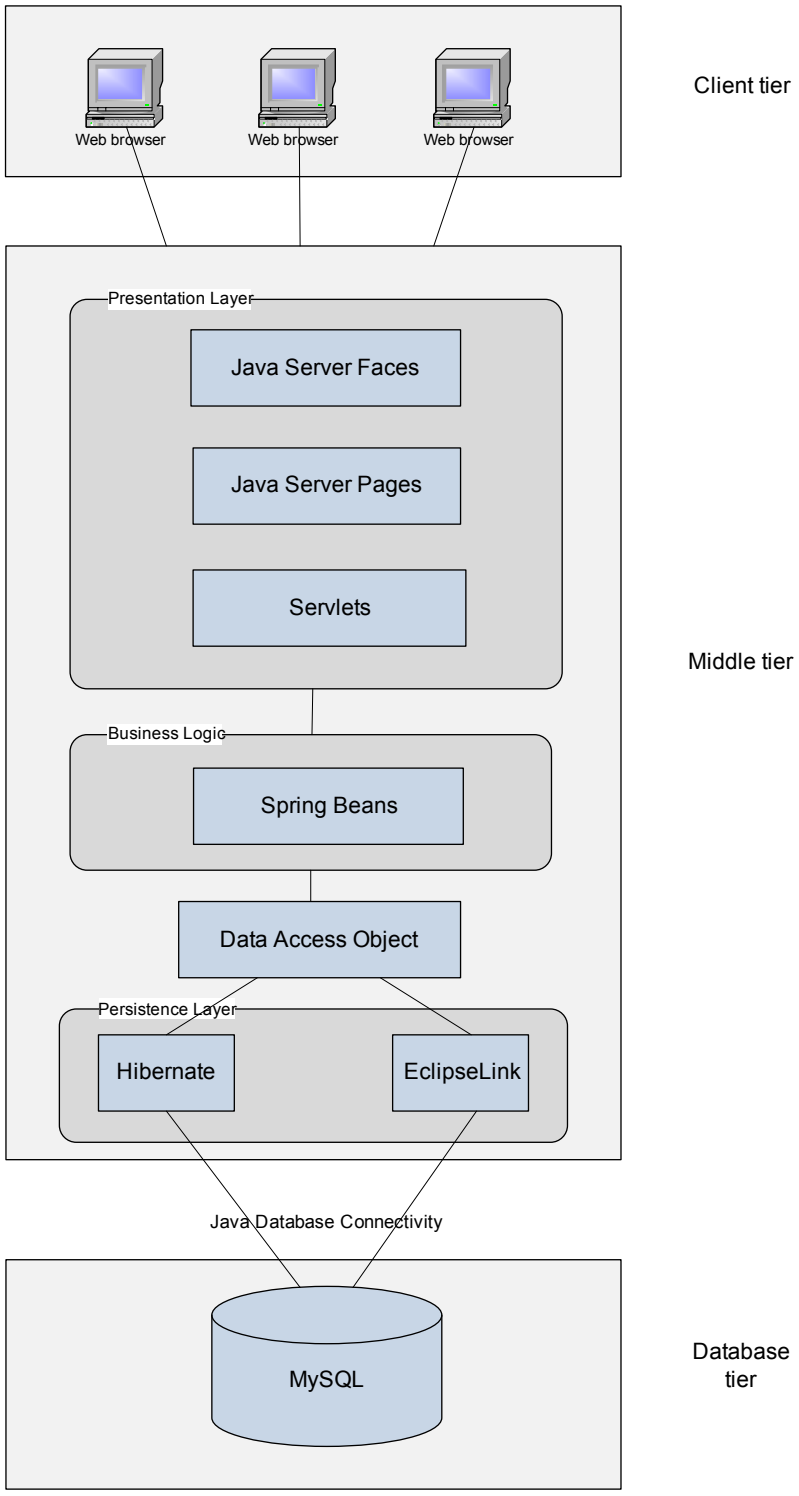

Figure 4.3.1 Architecture of the Grasshopper application

As the Figure 4.3.1 shows, the Grasshopper application is structured into a three-tier model. MySQL database is used to store data. The two versions of persistence layer are implemented with two frameworks, Hibernate and EclipseLink. Data Access Object (DAO) interface implements Spring's version of DAO pattern to connect to database. Utilizing DAO design pattern makes application highly modular and flexible. It allows changing persistence technology without alterations of other components of the application. The presentation layer builds on JSF, JSP and servlets. User-defined Spring Beans facilitate the business/persistence layer of the application and serve as an integration platform for the application's components.

4.4 Summary

This chapter described both object and relational models that are linked in the Grasshopper application. The relational model presented by the entity-relationship diagram, the class diagram describes the object model of the Grasshopper application. The architecture of the application is based on a three-tier design pattern. The next chapter will explore concrete application development and the role that Hibernate and EclipseLink play.

Chapter 5. Application Implementation

There are two approaches to creating a complete ORM domain model: POJO-driven approach and data model-driven approach. The POJO-driven approach requires starting development with an object domain model. The model-driven approach begins with building the database schema for the database system. This research takes the model-driven approach, that is, the Grasshopper database is implemented first. This chapter provides analysis of the Grasshopper application implementation, starting with the database implementation, followed by development of the domain model, building the persistence layer that intermediates between the two models, implementing application's business logic, and building the presentation layer.

5.1 Database Implementation

As noted, MySQL Server 5.4 was utilized for the Grasshopper database implementation. Appendix A contains scripts for creating Grasshopper database and its tables, and populating tables with data. The Figures 5.1.1-5.1.2 demonstrate how GUI based tools, such as MySQL Administrator and MySQL Query Browser assist the researcher with managing the database.

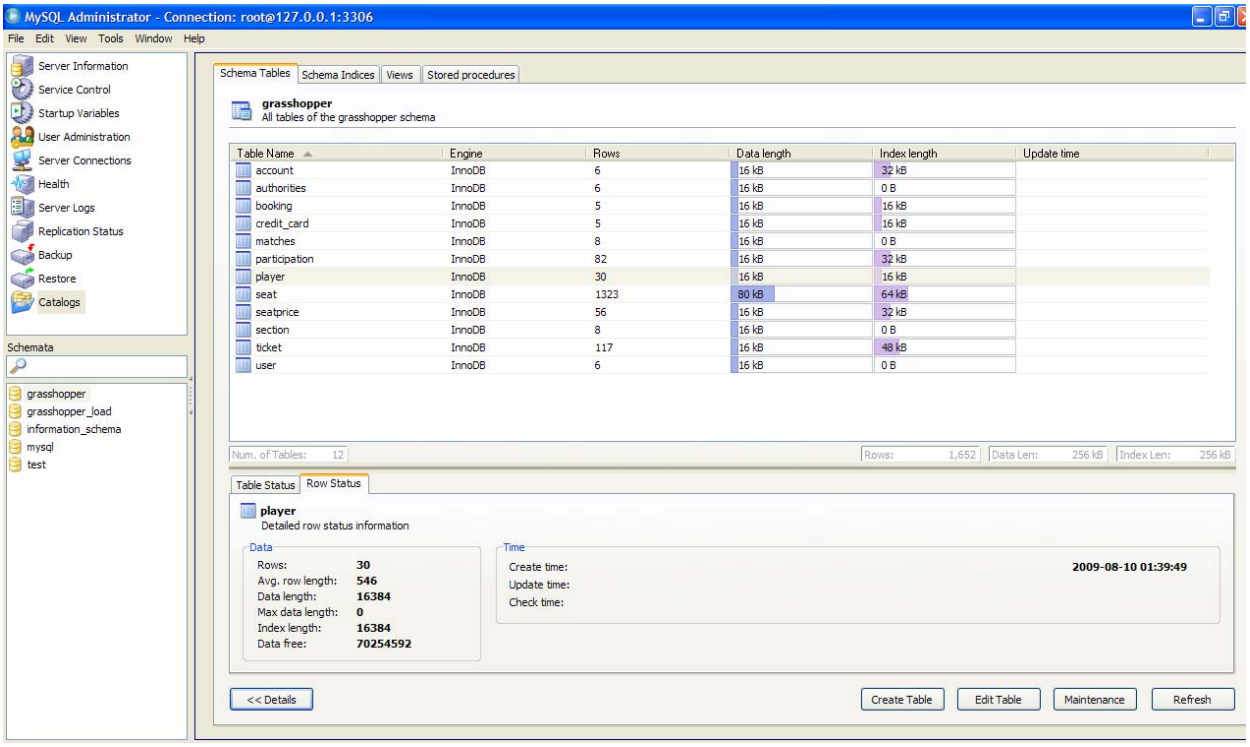

Figure 5.1.1 Grasshopper database tables

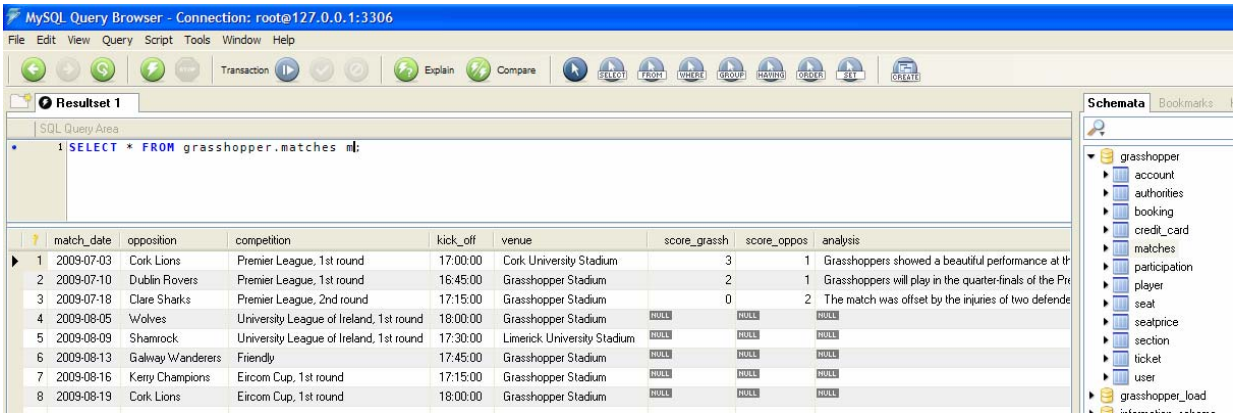

Figure 5.1.2 Managing data with MySQL Query Browser

The Grasshopper database is a production database that is connected to the back end of the Grasshopper Web site. The identical database, Grasshopper_load, is created for the purpose of performance testing of Hibernate and EclipseLink (see Appendix A for the script). Grasshopper_load is populated with the test data during the tests execution. The Figure 5.1.3 shows that the two databases are identical.

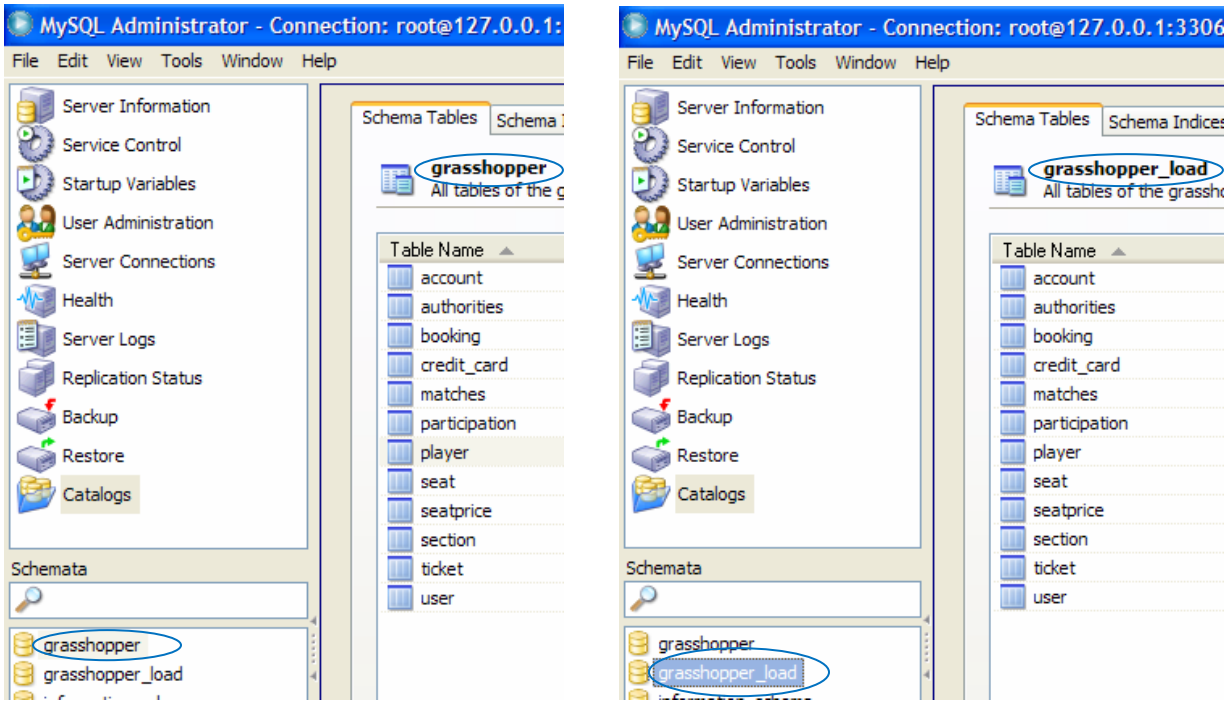

Figure 5.1.3 Grasshopper and Grasshopper_load databases

5.2 Building Domain Model Layer

Once the database schema has been created, the implementation of the domain model is at the next step of the model-driven approach. The domain model of the Grasshopper application is expressed as a set of POJOs and their interactions. As noted, POJOs are just Plain Old Java Objects, Java class definitions. Fisher and Duskis (2009) noted that a POJO conforms to the simple rules:

1. It contains a public, default constructor.
2. It has public getters for each property that is to be read and setters for each property that needs to be written.

In the Grasshopper application, the twelve POJOs identified to represent distinct entities with behavior and state within the domain model, are: Player, Matches, Participation, Seat, Section, SeatPrice, Ticket, Booking, User, Account, CreditCard, and Authorities. The POJOs

comprising the domain model correspond to the tables in the relational model. The Figure 5.2.1 illustrates how the Section POJO is linked to the Section table.

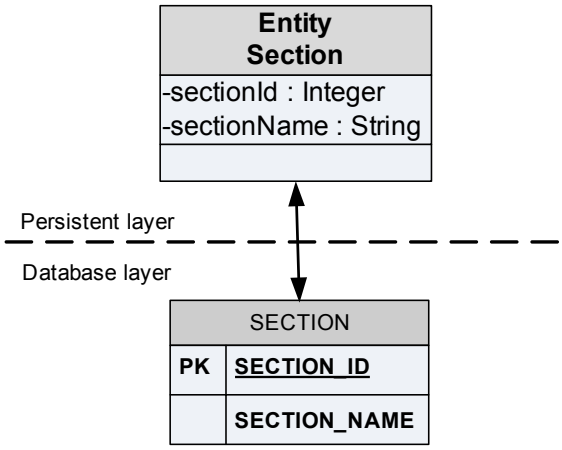

Figure 5.2.1 Mapping between the entity and the table

The following example of code shows how the simple Section POJO follows JavaBean conventions by exposing private fields via getters and setters (see Appendix B for the full source code of all POJOs):

Section.java class

```
import java.util.HashSet;
import java.util.Set;

public class Section implements java.io.Serializable {
    private static final long serialVersionUID = -329927538051457435lL;

    private Integer sectionId;
    private String sectionName;
    private Set<Seat> seats = new HashSet<Seat>(0);

    public Section() {
    }

    public Section(String sectionName) {
this.sectionName = sectionName;
    }

    public Section(String sectionName, Set<Seat> seats) {
this.sectionName = sectionName;
this.seats = seats;
    }

    public Integer getSectionId() {
```

```
return this.sectionId;
    }

    public void setSectionId(Integer sectionId) {
this.sectionId = sectionId;
    }

    …

    public Set<Seatprice> getSeatprices() {
return this.seatprices;
    }

    public void setSeatprices(Set<Seatprice> seatprices) {
this.seatprices = seatprices;
    }

}
```

5.3 The Data Access Object Layer

The more application code is abstracted away from interfacing directly with a database, the easier it is to switch to a different database platform or persistence technology. The Spring Framework enforces a modular architecture in which an application is divided into several layers. Each layer is focused on a particular task. The layer that handles persistence is referred to as a persistence layer, the layer that deals with business logic is often called the service layer while the controller layer is responsible for managing web requests. The layers are loosely coupled within the application. This separation of concerns promotes clean design.

Spring support for DAO allows “to completely abstract the datasource and ways in which it loads, saves, and manipulates the data at hand” (Fisher and Duskis, 2009). Utilizing the DAO design pattern for the Grasshopper application simplifies switching between Hibernate and EclipseLink persistence frameworks.

In the Grasshopper application, the domain layer objects have their DAO layer counterparts. Fisher and Duskis (2009) noted that the DAO methods for each persistent entity should be defined in a separate interface. This approach enables writing any number of implementations for a particular DAO interface. For example, the Grasshopper application

contains SectionDaoHibernate and SectionDaoJpa classes, both implementing SeatDao interface. This concept allows to easily switch between the two persistence implementations by adjusting the Spring configuration.

Spring provides the Generic DAO interface defining the standard methods that can be invoked on object. The following code demonstrates the GenericDao.java interface:

```
	package com.grasshopper.dao;

	import java.io.Serializable;
	import java.util.List;
	import com.grasshopper.dao.AbstractDao;

	public interface GenericDao<T extends Object, PK extends Serializable>
extends
	AbstractDao {
	    T findById(PK id, boolean lock);
	    List<T> findAll();
	    void save(T object);
	    T update(T object);
	    void delete(T object);
	    void deleteAll();
	    void flush();
	    void clear();
	    void evict(T object);
	    T saveOrUpdate(T entity);
	}
```

The GenericDAO can be extended by a concrete class, incorporating new methods specific for a particular object. For example, the SeatDao interface extends GenericDao with the getOverallSeatCount() method to count the number of seats on the stadium:

```
	package com.grasshopper.dao;

	import com.grasshopper.entity.Seat;

	public interface SeatDao extends GenericDao<Seat, Integer> {
	    Integer getOverallSeatCount();
	}
```

The Figure 5.3.1 shows the set of DAOs in the Grasshopper application (see Appendix C for the full source code of all DAO interfaces).

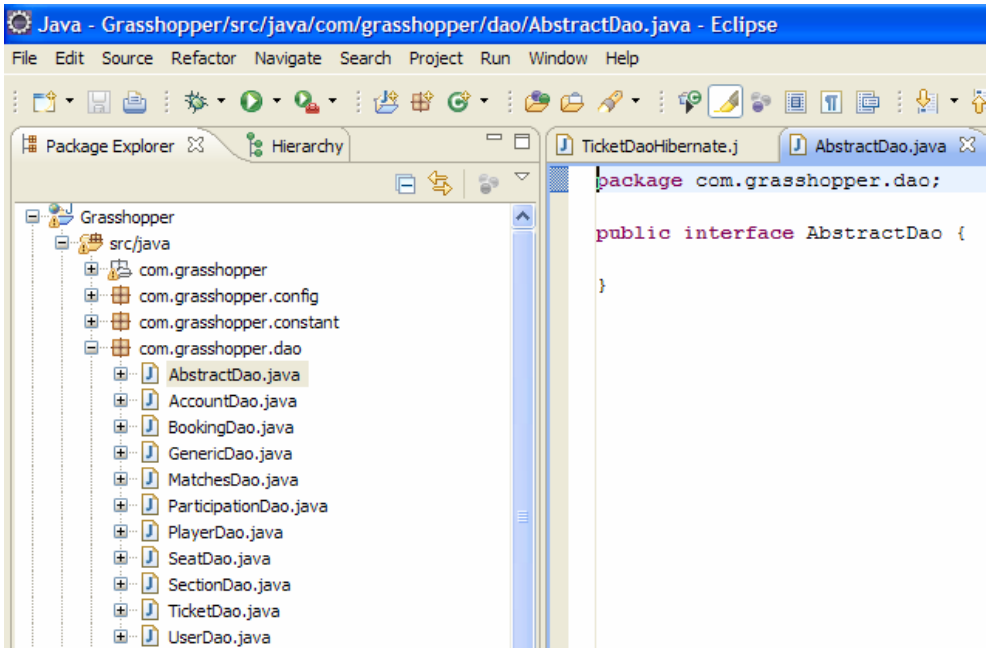

Figure 5.3.1 DAO interfaces

5.4 Object-Relational Mapping with Hibernate

5.4.1 Deploying Hibernate

Hibernate 3.3.2 was obtained from https://www.hibernate.org/ . The Hibernate distribution contains the Hibernate JAR (hibernate3.jar) with the third party dependencies, full source code and API documentation. The Hibernate JARs, that is the main Hibernate classes, are added to the application's library. The Figure 5.4.1.1 shows the list of the Grasshopper application's libraries. including JARs for Hibernate, EclipseLink, Spring and JSF.

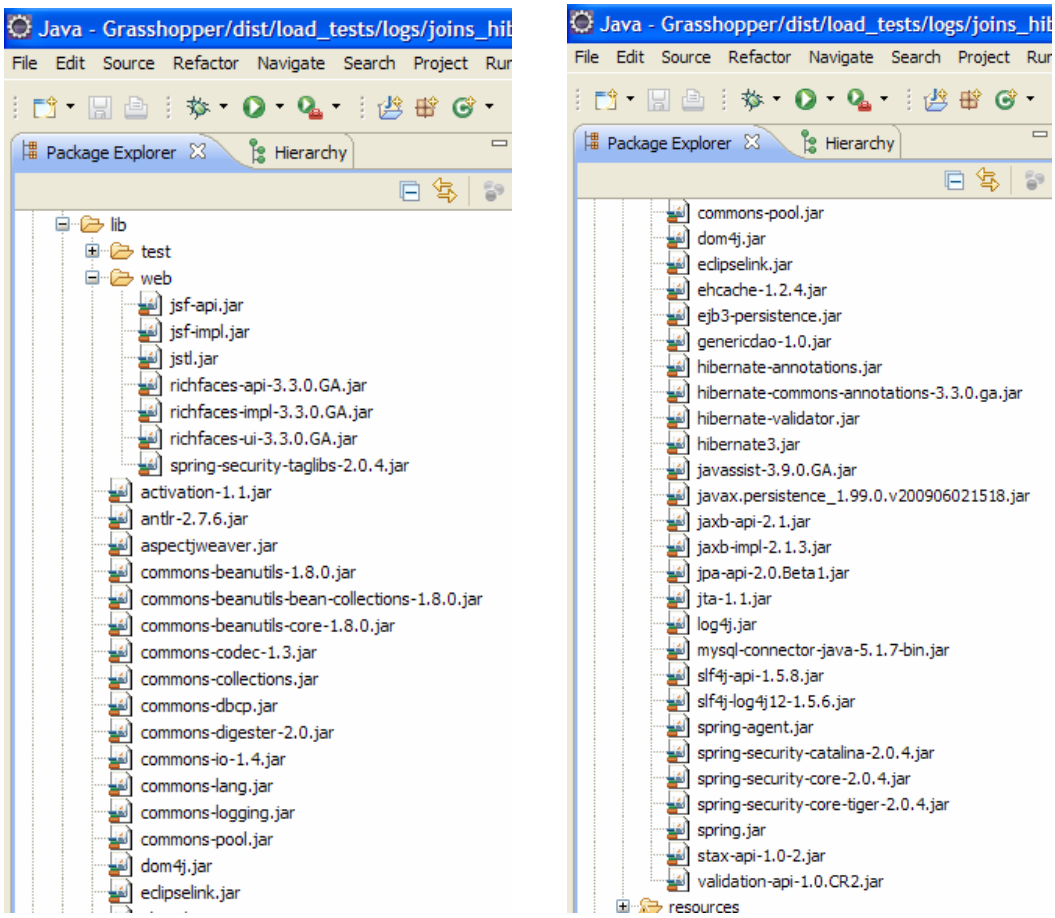

Figure 5.4.1.1 Application's libraries

5.4.2 Creating XML Mapping Files

Persistence layer with Hibernate requires mapping files that define which POJOs' properties are to be persisted in which relational tables. With Hibernate, mapping files can be designed through Annotations that go inline with the POJOs' code or as separate xml files, often referred to as hbm (HiBernate Mapping) files. As the Figure 5.4.2.1 illustrates, the Hibernate mapping in the Grasshopper application is presented with hbm files.

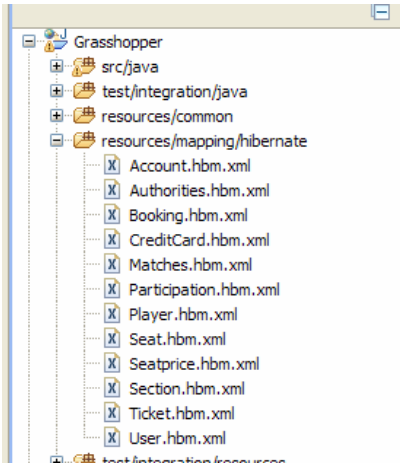

Figure 5.4.2.1 Hibernate xml mapping files

The Seat entity presents an interesting case for example of Hibernate mapping with xml. Mapping files define not only entity properties that need to be persisted by also associations between different entities. The Seat entity has both many-to-one and one-to-many relationships with the Section and Ticket entities respectively. In the relational model, this kind of relationships enforced through the foreign key constraints. The Figure 5.4.2.2 shows that the Seat entity has to be mapped to three tables: Seat, Section and Ticket.

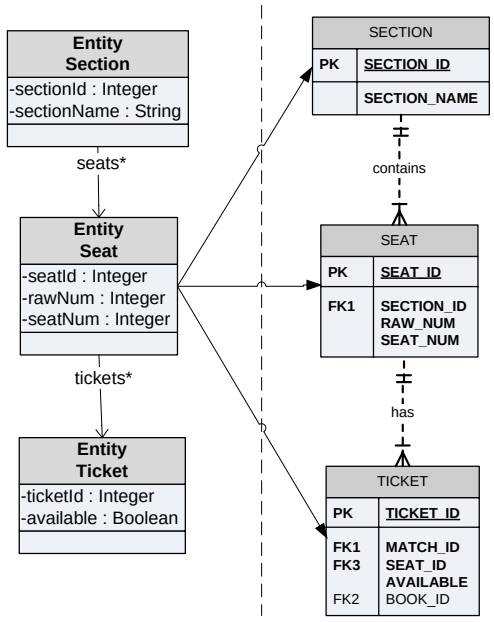

Figure 5.4.2.2 The Seat entity is mapped to three tables

The following is the code of the Seat.hbm.xml mapping file explained (see Appendix D for the full source code of all hbm files):

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.grasshopper.entity.Seat" table="seat">
        <id name="seatId" type="java.lang.Integer">
            <column name="seat_id" />
            <generator class="identity" />
        </id>
        <many-to-one name="section" class="com.grasshopper.entity.Section"
fetch="select">
            <column name="section_id" not-null="true" />
        </many-to-one>
        <property name="rawNum" type="int">
            <column name="raw_num" not-null="true" />
        </property>
        <property name="seatNum" type="int">
            <column name="seat_num" not-null="true" />
        </property>
        <set name="tickets" inverse="true" lazy="true" table="ticket"
fetch="select">
            <key>
                <column name="seat_id" not-null="true" />
            </key>
            <one-to-many class="com.grasshopper.entity.Ticket" />
        </set>
    </class>
</hibernate-mapping>
```

Seat entity is mapped to the Seat table

The name of the column in the table containing the primary key

Many-to-one relationship with the Section table, linked through the section_id column

Entity's properties to be persisted in the table's columns

One-to-many relationship with the Ticket table, enforced through the foreign key (seat_id) constraint

The root element of the mapping file is <hibernate-mapping>. The child element is <class>. In its body, it describes the relationships between Java objects and database tables. The <id> element defines the entity's primary key. Its name attribute identifies the entity's name containing the primary key. The child element <column> defines the column name that holds the primary key. The<id> element also requires a <generator> element to be specified, which defines the method to generate a new primary key for a new instance of the class. In the Seat class, the generators are defined of "identity" type. The <property> element defines a set of entity's properties to be persisted in the columns. The properties include attributes such as names, types and nullable property for both entity and column.

To enforce the many-to-one relational rule, the “many” class has a foreign key into the “one” class. The fetch attribute identifies the mode in which the element will be retrieved. In this example, the instances will be retrieved with a series of selects.

When one object has an attribute capable of containing many objects, the relationship is expressed as a one-to-many. The one-to-many association represents Collection classes, such as Set, List, or Map. The <set> element in the discussed example contains new attributes “inverse” and “lazy”. Inverse attribute, set to true value, identifies the Seat entity as the opposite navigable end of a relationship expressed in the Ticket’s mapping. To control fetching associations, Hibernate supports lazy and eager loading. The lazy fetching allows related entities to be lazily loaded when needed while the eager fetching loads data from the database when the entity is initially read. The lazy loading has performance benefits.

5.4.3 Building the Hibernate DAO Layer

The DAO layer for Hibernate requires designing of the Hibernate DAO classes. The Hibernate DAO classes define the Hibernate implementation of the DAO abstract methods that provide for the application’s persistence logic. As the Figure 5.5.3.1 shows, the Hibernate DAOs are placed into the com.grasshopper.dao.orm.hibernate package (see Appendix D for the full source code of Hibernate DAO classes).

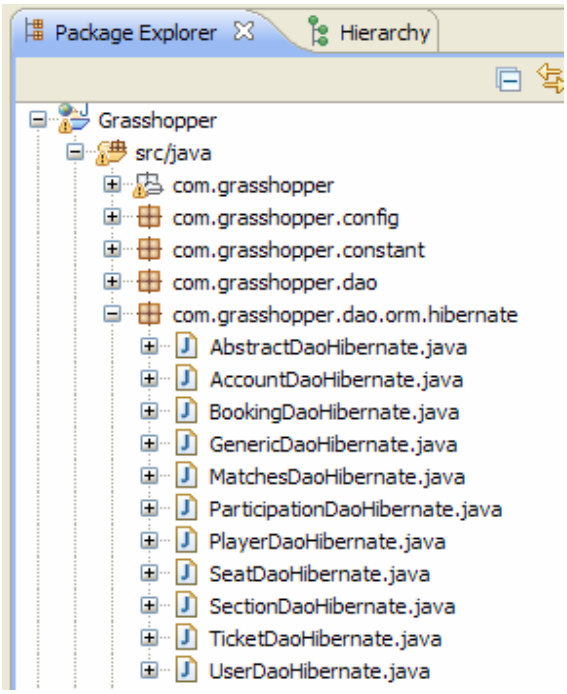

Figure 5.4.3.1 Hibernate DAO classes

The easiest way of building Hibernate DAO classes is extending Spring's HibernateDaoSupport class. This abstract class requires an instance of Hibernate SessionFactory. The SessionFactory can be injected through the Spring configuration of the Hibernate SessionFactory (this process is described in the next chapter).

When a SessionFactory is passed in to a class that extends HibernateDaoSupport, a HibernateTemplate instance is automatically created. The template pattern greatly simplifies the code and programming effort by extracting the redundant tasks into a template. The template pattern can be demonstrated with the MatchesDaoHibernate class that implements MatchesDao:

```
package com.grasshopper.dao;

public interface MatchesDao extends GenericDao<Matches, Integer> {
    int getSoldTicketsCount(Matches match);

    List<Matches> getUpcomingMatches(int matchesAmountLimit);

    List<Ticket> findAvailableTickets(Integer matchId);
}
```

```
package com.grasshopper.dao.orm.hibernate;

public class MatchesDaoHibernate extends GenericDaoHibernate<Matches,
Integer>
      implements MatchesDao {

    @SuppressWarnings("unchecked")
    public int getSoldTicketsCount(Matches match) {
      List res = getHibernateTemplate()
            .find(
                  "select count(t) from Matches m join m.tickets t where
t.available = false and m.matchId = ?",
                  match.getMatchId());
      int result;
      if (!res.isEmpty()) {
          result = ((Number) res.get(0)).intValue();
      } else {
          result = -1;
      }
      return result;
    }
…

@SuppressWarnings("unchecked")
    public List<Ticket> findAvailableTickets(Integer matchId) {
      String hql = "SELECT t FROM Ticket t JOIN t.matches m WHERE t.available
= true AND m.matchId = ?";
      return (List<Ticket>) getHibernateTemplate().find(hql, matchId);
    }
```

The code snippet above shows the three abstract methods defined in the MatchesDao interface. The templated implementation of these methods in the MatchesDaoHibernate class illustrates that untemplated code, left for developer to write, is the persistence logic itself (the text highlighted in blue). The Hibernate Template pattern allows the developer to focus on the database operation without having to worry about the details, such as beginning a transaction or catching an exception.

5.4.4 Hibernate Configuration with Spring

The integration of Hibernate with Spring is handled by the application context. The application context supports the Spring services, DAOs, Hibernate, transaction management, Hibernate XML mapping files (or annotations), and database connectivity. In the Grasshopper application, the applicationContext-hibernate.xml file holds the parameters of the application

context. This file, along with other configuration files, is placed into Grasshopper/resources/common folder, depicted in the Figure 5.4.4.1.

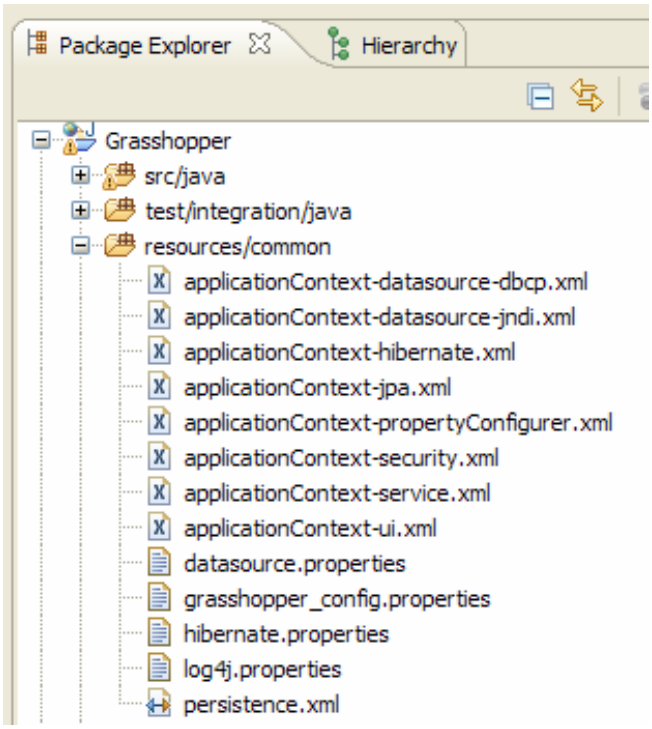

Figure 5.4.4.1 Configuration files

The applicationContext-hibernate.xml file consists of several modules. The SessionFactory module is responsible for creating a sessionFactory bean that defines all mapping files and hibernateProperties properties. The hibernateProperties property is loaded from the external hibernate.properties file. The following code snippet is taken from the sessionFactory part of the applicationContext-hibernate.xml file (see Appendix E for the full source code):

```
        <bean id="sessionFactory"

class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
        <property name="dataSource" ref="dataSource" />
        <property name="mappingResources">
                <list>
                        <value>Account.hbm.xml</value>
                        <value>Booking.hbm.xml</value>
                        …
                        <value>User.hbm.xml</value>
                                        </list>
```

```
            </property>
            <property name="hibernateProperties">
                  <props>
                        <prop
key="hibernate.dialect">${hibernate.dialect}</prop>
                        …
                        <prop
key="hibernate.hbm2ddl.auto">${hibernate.hbm2ddl.auto}</prop>
                  </props>
            </property>
      </bean>
```

The values of the hibernateProperties are defined in the hibernate.properties file:

```
hibernate.dialect=org.hibernate.dialect.MySQLDialect
hibernate.generate_statistics=true
hibernate.jdbc.batch_size=50
hibernate.show_sql=false
hibernate.format_sql=false
```

The value of hibernate.dialect field indicates that the SQL dialect is set to MySQLDialect. The value of true for statistics allows generating database statistics. The maximum batch size for updates is set to 50. The value of false defined for the show_sql element disables displaying SQL statements on console. Formatting of the SQL commands is also disabled by setting fromat_sql to false.

The next chunk of the applicationContext-hibernate.xml file is dedicated to transaction configuration. The following settings enable the use of transactions based on annotations within Spring beans:

```
      <bean id="txManagerHibernate"

      class="org.springframework.orm.hibernate3.HibernateTransactionManager">
            <property name="sessionFactory" ref="sessionFactory" />
            <property name="dataSource" ref="dataSource" />
      </bean>

      <tx:annotation-driven transaction-manager="txManagerHibernate" />
```

The configuration of annotations within beans is defined in the applicationContext-ui.xml file:

```
<context:annotation-config />

	<bean id="facesSupport" class="com.grasshopper.web.helper.FacesSupport"
/>

	<bean id="authBean" class="com.grasshopper.web.bean.AuthorizationBean"
		scope="request" />

	…

	<bean id="bookingDetails"
class="com.grasshopper.web.bean.BookingDetails"
		scope="session" />

</beans>
```

The next block of code in the applicationContext-hibernate.xml file is responsible for setting up DAOs:

```
	<bean id="abstractDaoHibernate"
class="com.grasshopper.dao.orm.hibernate.AbstractDaoHibernate"
		abstract="true">
		<property name="sessionFactory" ref="sessionFactory" />
	</bean>

	<bean id="genericDaoHibernate"
class="com.grasshopper.dao.orm.hibernate.GenericDaoHibernate"
		abstract="true" parent="abstractDaoHibernate" />

	<bean id="userDao"
class="com.grasshopper.dao.orm.hibernate.UserDaoHibernate"
		parent="genericDaoHibernate" />

	<bean id="participationDao"
class="com.grasshopper.dao.orm.hibernate.ParticipationDaoHibernate"
		parent="genericDaoHibernate" />
</beans>
```

This service includes wiring the beans. The abstractDaoHibernate bean references sessionFactory that contains database connectivity parameters and hbm files for data mapping. These settings allow injecting the instance of SessionFactory into the AbstractDaoHibernate class and, consequently, to the rest of the Hibernate DAO classes as they extend from the AbstractDaoHibernate class through the GenericDaoHibernate class.

5.5 Object-Relational Mapping with EclipseLink

5.5.1 Deploying EclipseLink

EclipseLink 1.1.2 was downloaded from http://www.eclipse.org/eclipselink/downloads/index.php . The main EclipseLink class, eclipselink.jar (see Figure 5.5.1.1), along with its dependencies are added to the Grasshopper's library, into the grasshopper/lib directory.

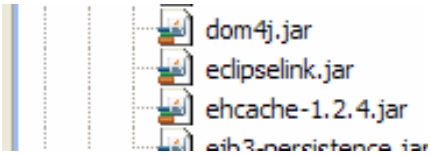

Figure 5.5.1.1 Eclipselink.jar library

5.5.2 Creating Mapping with Annotations

EclipseLink, similarly to Hibernate, supports two ways of defining the mapping metadata – using annotations or xml mapping files. To demonstrate both mapping methods in this research, the Hibernate mapping was implemented using xml files while the EclipseLink mapping was build through annotations.

Once the POJOs have been designed, they can be easily transformed into persistent entities by incorporating some annotations into the code. As much as adding an @Entity annotation and an @Id annotation to an appropriate getter makes the POJO a persistent object. The following EclipseLink JPA annotations have been used to create mapping entities for the Grasshopper application:

- @Entity, indicates that this class is an entity that should be persisted.
- @Table, specifies the name of the table in which the data will be stored.
- @Id, defines the id attribute as the primary key.

- @Generated Value, indicates that the primary key should be generated using the underlying database id utility.
- @Column, defines the properties of the column.
- @Temporal, defines a column as a DATE type.
- @OneToMany, defines a one-to-many relationship between the two objects.
- @ManyToOne, defines a many-to-one relationship. In Java, this multiplicity is described by the Collections such as List, Set and Map.
- @JoinColumn, identifies the column that joins two entities through the foreign key constraint.

The Figure 5.5.2.1 shows that the set of entities is placed into the com.grasshopper.entity package in the Grasshopper/src/java directory.

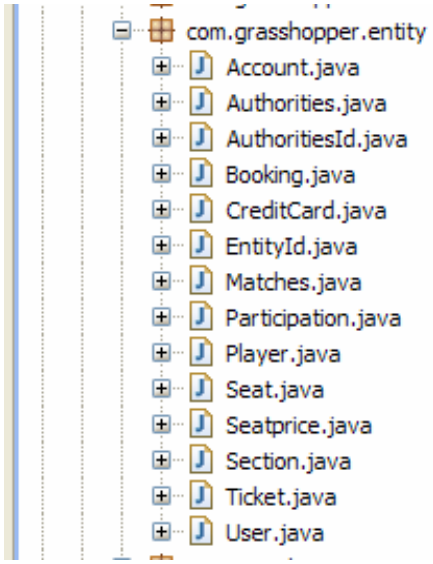

Figure 5.5.2.1 Application's entities

The diagram in the Figure 5.5.2.2 demonstrates how the object's properties and relationships are mapped to the relational tables.

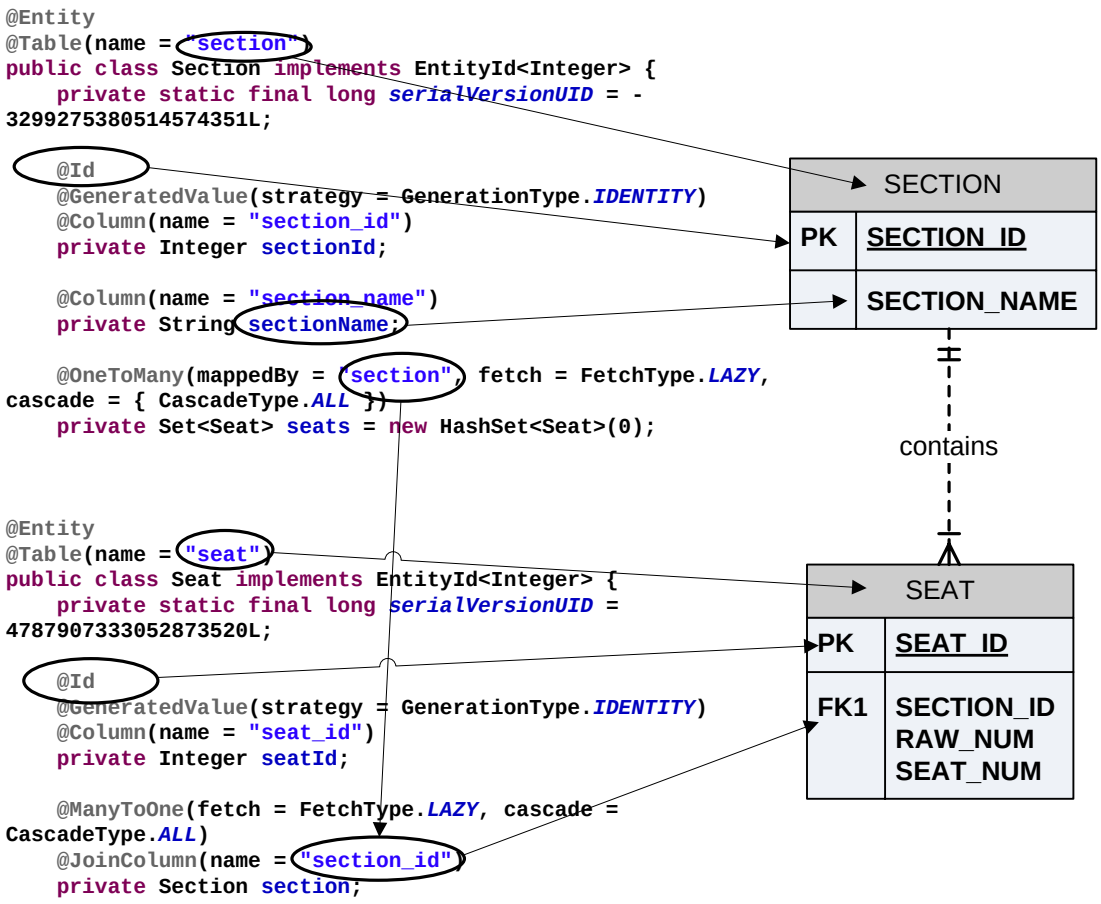

Figure 5.5.2.2 Mapping with annotations

The diagram shows the snippet of the code from Section and Seat classes (see Appendix F for the full source code of all entities). @Entity annotation in the beginning of the Seat class indicates that this entity should be persisted to the table whose name is stated in the @Table annotation. In this case, it is the seat table. The @Id attribute corresponds to the primary key, seat_id, in the Seat table. The Seat entity has many-to-one relationship with the Section entity, while Section entity is related to the Seat entity as a one-to-many. This association is handled by the @ManyToOne annotation in the Seat class and the @OneToMany annotation in the Section class. The mappedBy element in the Section entity indicates that the join column is specified at the other end of the relationship. In the Seat entity, the @JoinColumn annotation defines the

linking column as the section_id that corresponds to the section_id foreign key constraint in the Seat table.

5.5.3 Building the EclipseLink JPA DAO Layer

The DAO layer for EclipseLink JPA builds on the same principles as the DAO for Hibernate. The JPA DAO classes extend the Spring's JpaDaoSupport class, as the following code snippet shows:

```
public abstract class AbstractDaoJpa extends JpaDaoSupport implements
      AbstractDao {}
```

The set of DAO JPA classes is placed into the com.grasshopper.dao.orm.jpa package, depicted in the Figure 5.5.3.1 (see Appendix F for the full source code of EclipseLink DAO classes).

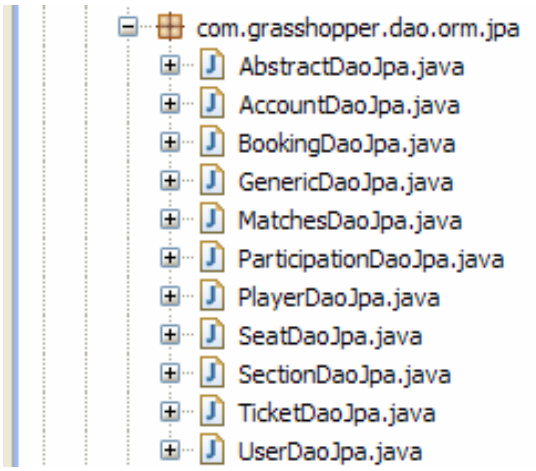

Figure 5.5.3.1 DAO JPA classes

The EntityManagerFactory is injected into the AbstractDaoJpa class by setting configuration parameters of Spring application context with JPA.

Similarly to Hibernate Template, Spring supports JPA Template. When working with EclipseLink JPA directly, it is necessary to create a new EntityManager instance from an EntityManagerFactory. The developer also needs to take care about the transactional details. The

JPA Template ensures that the EntityManager is created and the transactional rules are properly applied. The transactional requirements are specified via configuration, separating transactional details from the persistence logic.

The following code snippet, implementing the getSoldTicketCount() method with JPA Template, shows that the JPA Template works in the same fashion as the Hibernate Template (this method implemented with Hibernate Template was discussed earlier). The developer needs to write only persistence logic, the JPA Template is concerned with the rest details.

```
public class MatchesDaoJpa extends GenericDaoJpa<Matches, Integer> implements
      MatchesDao {

    @SuppressWarnings("unchecked")
    public int getSoldTicketsCount(Matches match) {
      List res = getJpaTemplate()
            .find(
                  "SELECT count(t) FROM Matches m INNER JOIN m.tickets t
where t.available = false and m.matchId = ?1 ",
                  match.getId());
      int result;
      if (!res.isEmpty()) {
          result = ((Number) res.get(0)).intValue();
      } else {
          result = -1;
      }
      return result;
    }
```

5.5.4 EclipseLink JPA Configuration with Spring

The EclipseLink integration with Spring is concerned with configuration of the persistence.xml and applicationContext-jpa.xml files. The persistence.xml file is responsible for configuring the EclipseLink JPA environment. The file consists of the several modules (see Appendix E for the full source code of persistence.xml and applicationContext-jpa.xml files).

The following module creates a persistence unit called GrasshopperPU:

```
<?xml version="1.0" encoding="UTF-8"?>

<persistence version="1.0"
      xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
      xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
      <persistence-unit name="GrasshopperPU" transaction-
type="RESOURCE_LOCAL">
```

The next chunk of code defines persistence provider and specifies the classes (mapping Entities) that will be added to the GrasshopperPU persistence unit:

```
      <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
            <class>com.grasshopper.entity.Account</class>
            <class>com.grasshopper.entity.Authorities</class>
            …
            <class>com.grasshopper.entity.User</class>
```

Finally, various database connectivity properties are defined:

```
            <properties>
                  <property name="eclipselink.jdbc.driver"
value="com.mysql.jdbc.Driver"/>
                  <property name="eclipselink.jdbc.url"
value="jdbc:mysql://localhost:3306/grasshopper"/>
                  <property name="eclipselink.ddl-generation"
value="verify"/>
                  <property name="eclipselink.jdbc.user"
value="grasshopper"/>
                  <property name="eclipselink.jdbc.password" value="gsh09"/>
                  <property name="eclipselink.logging.level" value="INFO"/>
                  <property name="eclipselink.target-database"
value="MySQL"/>
            </properties>
```

The Spring side of the integration platform is configured in the applicationContext-jpa.xml file. It handles such aspects as connections, transactions, and DAOs. The first module handles connections parameters. For example, it identifies EclipseLink JPA vendor adapter and EclipseLink JPA dialect of SQL:

```
      <bean id="jpaVendorAdapter"

      class="org.springframework.orm.jpa.vendor.EclipseLinkJpaVendorAdapter">
            <property name="databasePlatform"

      value="org.eclipse.persistence.platform.database.MySQLPlatform" />
            <property name="showSql" value="true" />
      </bean>

      <bean id="jpaDialect"
            class="org.springframework.orm.jpa.vendor.EclipseLinkJpaDialect"
```

The next block defines properties of EntityManagerFactory:

```
	<bean id="entityManagerFactory"

	class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBe
an">
		<property name="jpaVendorAdapter" ref="jpaVendorAdapter" />
		<property name="persistenceXmlLocation"
value="${persistence.xml.file}" />
		<property name="persistenceUnitName" value="GrasshopperPU" />
		<property name="jpaDialect" ref="jpaDialect" />
	</bean>
```

Furthermore, the transaction management parameters are specified. The following configuration settings define using the JPA transaction management. The annotation-driven transaction management approach makes JPA transaction manager process @Transactional annotations.

```
	<bean id="txManager"

	class="org.springframework.orm.jpa.JpaTransactionManager">

		<property name="entityManagerFactory" ref="entityManagerFactory"
/>
		<property name="dataSource" ref="dataSource" />
	</bean>

	<tx:annotation-driven transaction-manager="txManager" />
```

The next configuration fragment defines settings that allow injecting the instance of EntityManagerFactory into JPA DAOs:

```
	<bean id="abstractDaoJpa"
class="com.grasshopper.dao.orm.jpa.AbstractDaoJpa"
		abstract="true">
		<property name="entityManagerFactory" ref="entityManagerFactory"
/>
	</bean>

	<bean id="genericDaoJpa"
class="com.grasshopper.dao.orm.jpa.GenericDaoJpa"
		abstract="true" parent="abstractDaoJpa" />

	<bean id="accountDaoJpa"
class="com.grasshopper.dao.orm.jpa.AccountDaoJpa"
		parent="genericDaoJpa" />
	…
	<bean id="userDaoJpa" class="com.grasshopper.dao.orm.jpa.UserDaoJpa"
		parent="genericDaoJpa" />
```

The `<context:annotation-config />` directive makes Spring process all annotations to ensure that the correct instance of EntityManager will be instantiated and injected into the DAO.

5.6 Business Logic and Presentation Layers

In Spring framework, business logic layer is known as services, while presentation layer is referred to as controllers. The service layer provides an API through which the other layers of the application interface. The service layer defines the application's business core logic invoking one or more DAO methods to accomplish this task. Often, methods in the service layer composed of multiple DAO methods that, grouped together, execute business logic as a single unit of work. These groups of DAO methods are known as service facade methods. The facade methods implement the concept of transaction: the entire method either completes successfully or it is rolled back leaving the application in the same state as it was before the method was called.

The service classes are created using similar approach as seen in the DAO layer. The Figure 5.6.1 illustrates the file structure of the Grasshopper service layer (see Appendix G for the full source code).

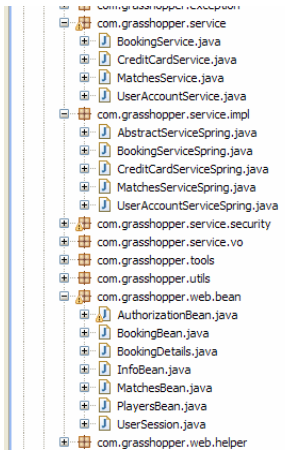

Figure 5.6.1 The service layer

The service interfaces are defined in the com.grasshopper.service package. For example, a CreaditCardService interface is designed to collect and save user's credit card data.

```
package com.grasshopper.service;

public interface CreditCardService {

    boolean saveCreditCardData(User user, CreditCard creditCard);
}
```

The interfaces are then implemented in the concrete service classes in the com.grasshopper.service.impl package. For instance, the following code snippet shows the implementation of the CreditCardService interface:

```
package com.grasshopper.service.impl;


public class CreditCardServiceSpring extends AbstractServiceSpring implements
      CreditCardService {

    @Autowired
    private CreditCardDao creditCardDao;

    @Autowired
    private UserDao userDao;

    public boolean saveCreditCardData(User user, CreditCard creditCard) {
      boolean isCreditCardAlreadyExistent = creditCardDao
            .isCreditCardRegistered(creditCard.getNumber());
      if (!isCreditCardAlreadyExistent) {
          User foundUser = userDao.findById(user.getUserId(), false);
          creditCard.setUser(foundUser);
          creditCardDao.save(creditCard);
      }
      return !isCreditCardAlreadyExistent;
    }
```

In the code above, the saveCreditCardData() method is autowired to two beans, creditCardDao and userDao. The method determines whether the data about the given credit card already exists in the database. If it is not exist, the credit card information is saved with a crediCardDao.save() method.

The Web Java Beans classes are autowired to the corresponding services. They provide user interface with data. For example, the following code snippet from the MatchesBean.java file shows how the data about the match analysis have been extracted from the database.

```
public String getAnalysis() {
      String analysis = null;
      if (matchId != null) {
          Matches match = matchesDao.findById(matchId, false);
          if (match != null) {
            analysis = match.getAnalysis();
          }
```

To present the data to the users in the form of GUI, the service classes are injected into the controllers. JSP and JSF pages display data acquired from services. For example, the JSF page, stated below, presents match analysis, obtained from the MatchesBean.java, to the user (see Appendix G for the full source code of the presentation layer).

```
<%@page import="com.grasshopper.web.bean.MatchesBean"%>

<rich:modalPanel id="matchPanel" width="500" height="350">
      <f:facet name="header">
            <h:outputText value="#{i18n.match_analysis}" />
      </f:facet>
      <f:facet name="controls">
            <h:panelGroup>
                  <h:graphicImage value="img/close.gif" styleClass="hidelink"
                        id="hidelinkPlayer" />
                  <rich:componentControl for="matchPanel"
attachTo="hidelinkPlayer"
                        operation="hide" event="onclick" />
            </h:panelGroup>
      </f:facet>
      <h:panelGrid id="matchInfo" columns="1" footerClass="regFooter">
            <h:outputText value="#{matchesBean.analysis}"
styleClass="infoText"/>
      </h:panelGrid>
</rich:modalPanel>
```

5.7 Application deployment

Once all layers of the application architecture have been implemented – database layer, DAO layer, persistence layer, business logic, and presentation layer – the Grasshopper application can be deployed. To create the Grasshopper deployment project, an Ant task has been

created with the Ant tool. The build.xml file provides code for creating an Ant task. As the Figure 5.7.1 illustrates, the default_clean task is defined as the default task for building the Grasshopper project. The configuration properties for building the project are defined in the build.properties file (see Appendix G for the full source of the build.xml and build.properties files). When the Ant task completes successfully, the deployment grasshopper.war file is created.

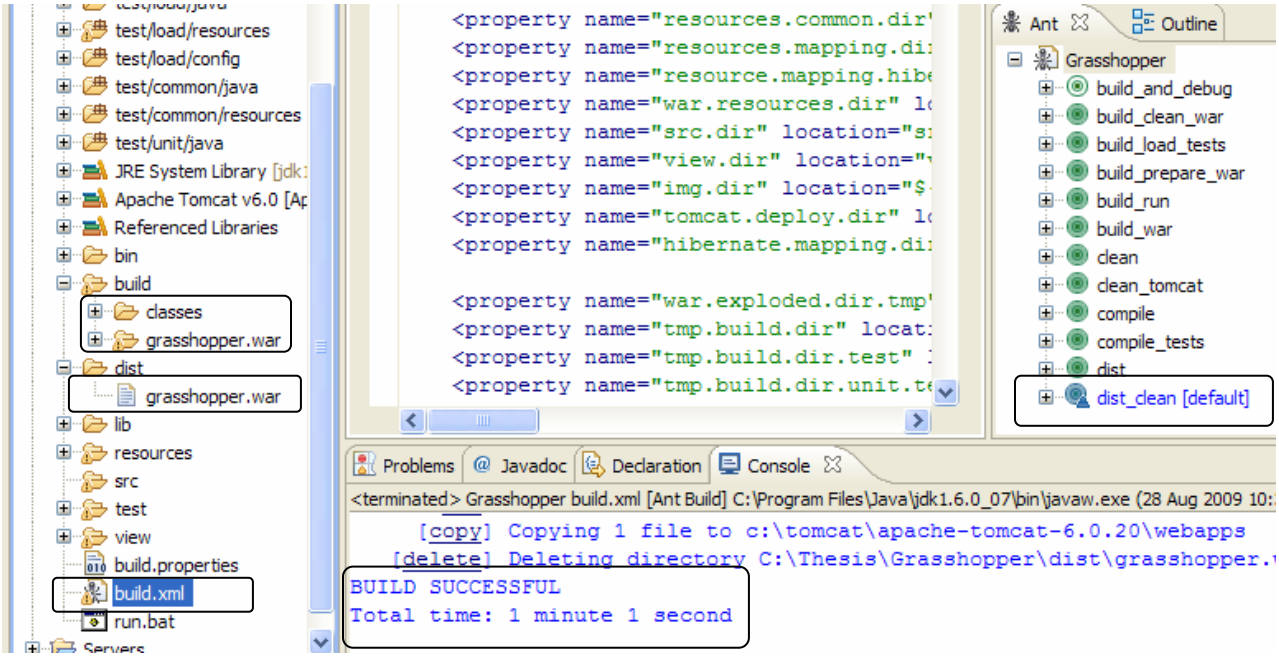

Figure 5.7.1 Building the Grasshopper deployment project with Ant task

The project was deployed in the Apache Tomcat servlet container. The Figure 5.7.2-5.7.16 walk through the pages of the Grasshopper application.

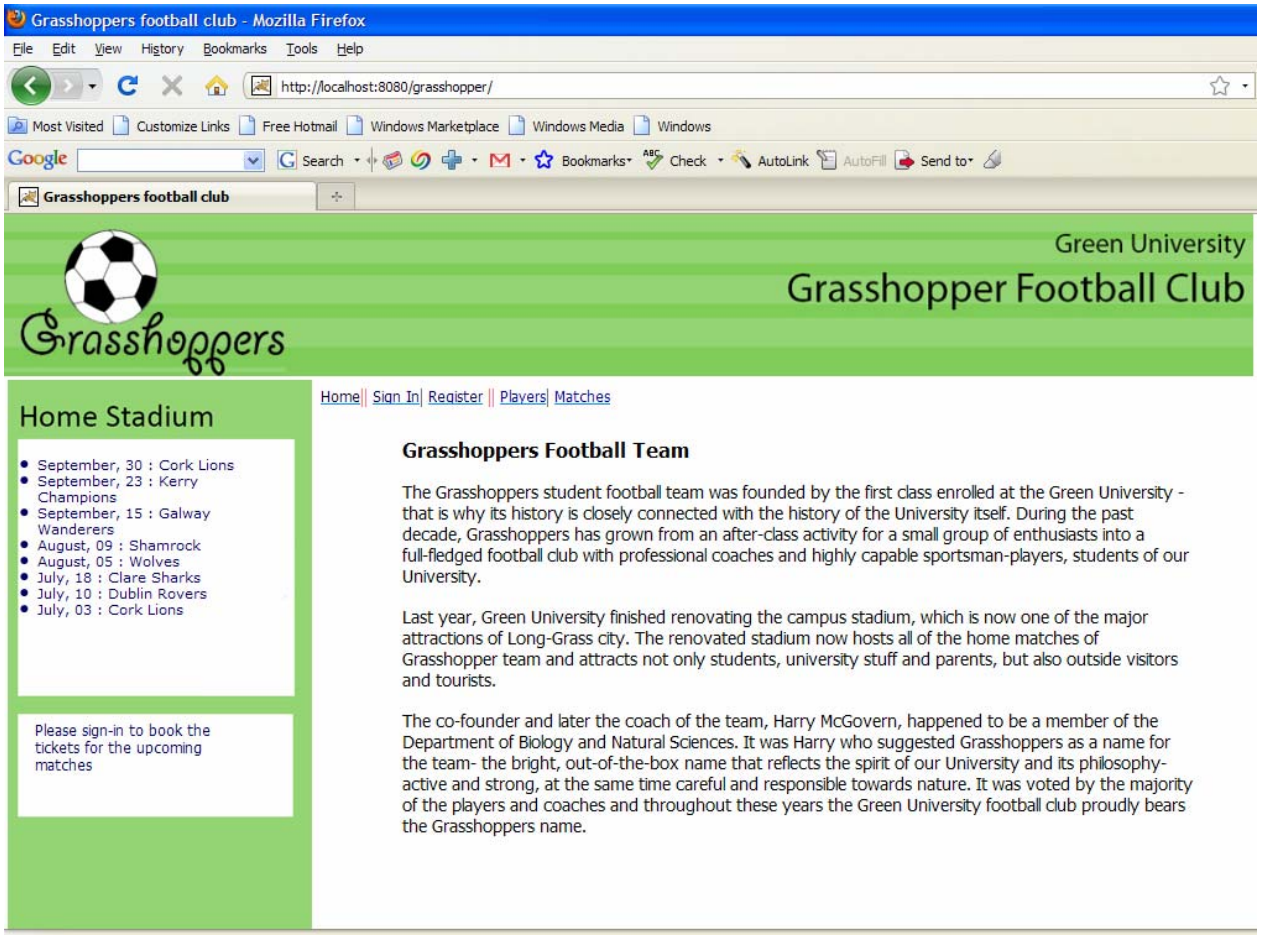

Figure 5.7.2 The main page

The main page is generated by the index.jsp file. The page contains the links to the Players and Matches web pages. Also, the links to sign into existing account, or to register for a new account are provided.

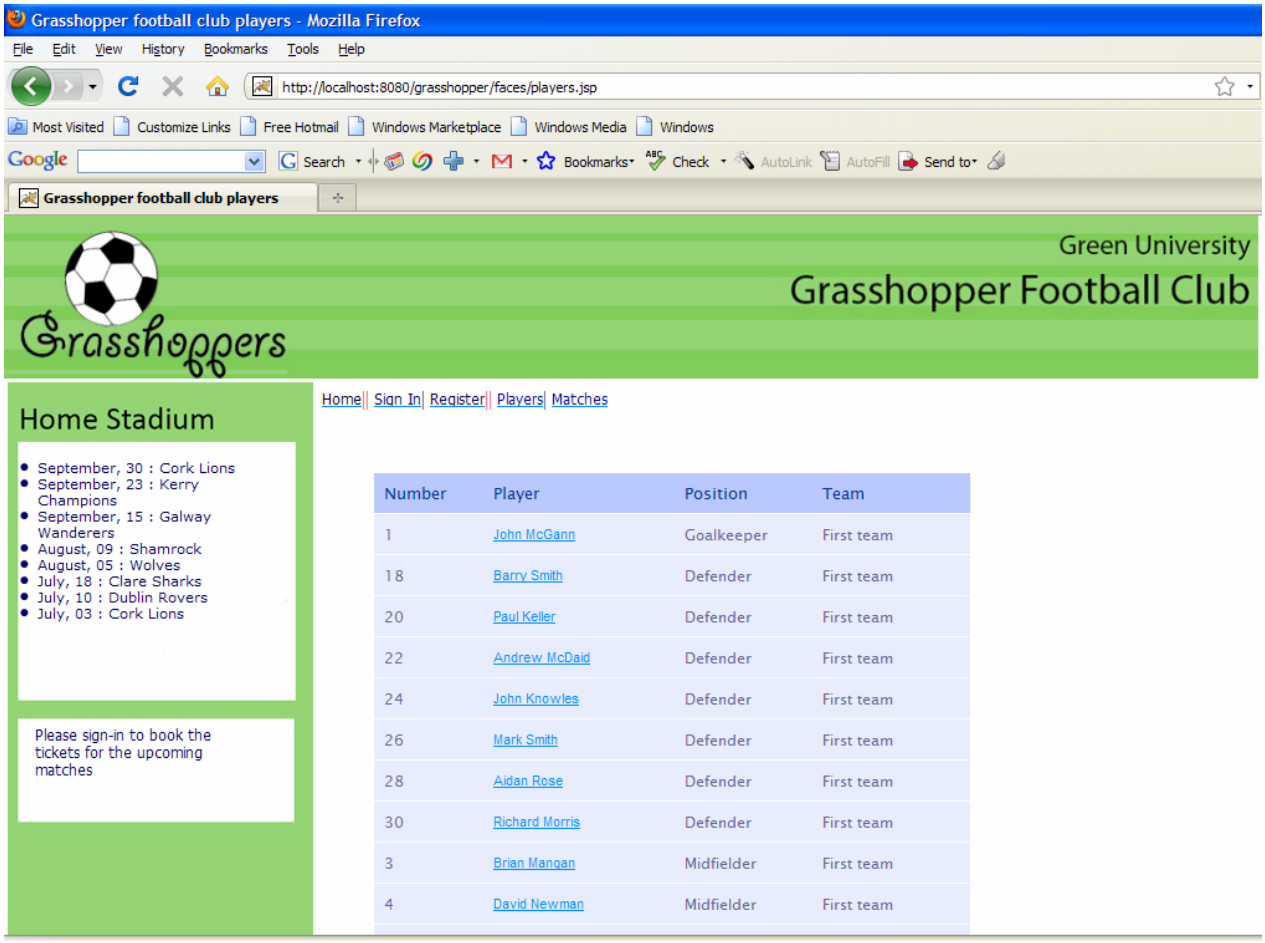

Figure 5.7.3 The Players page

The Players page is downloaded by the players.jsp file. The page presents information about the Grasshoppers team players, both first team and reserve. The names of the players are clickable links that take a user to the JSF page. The JSF page displays the full information about the selected player.

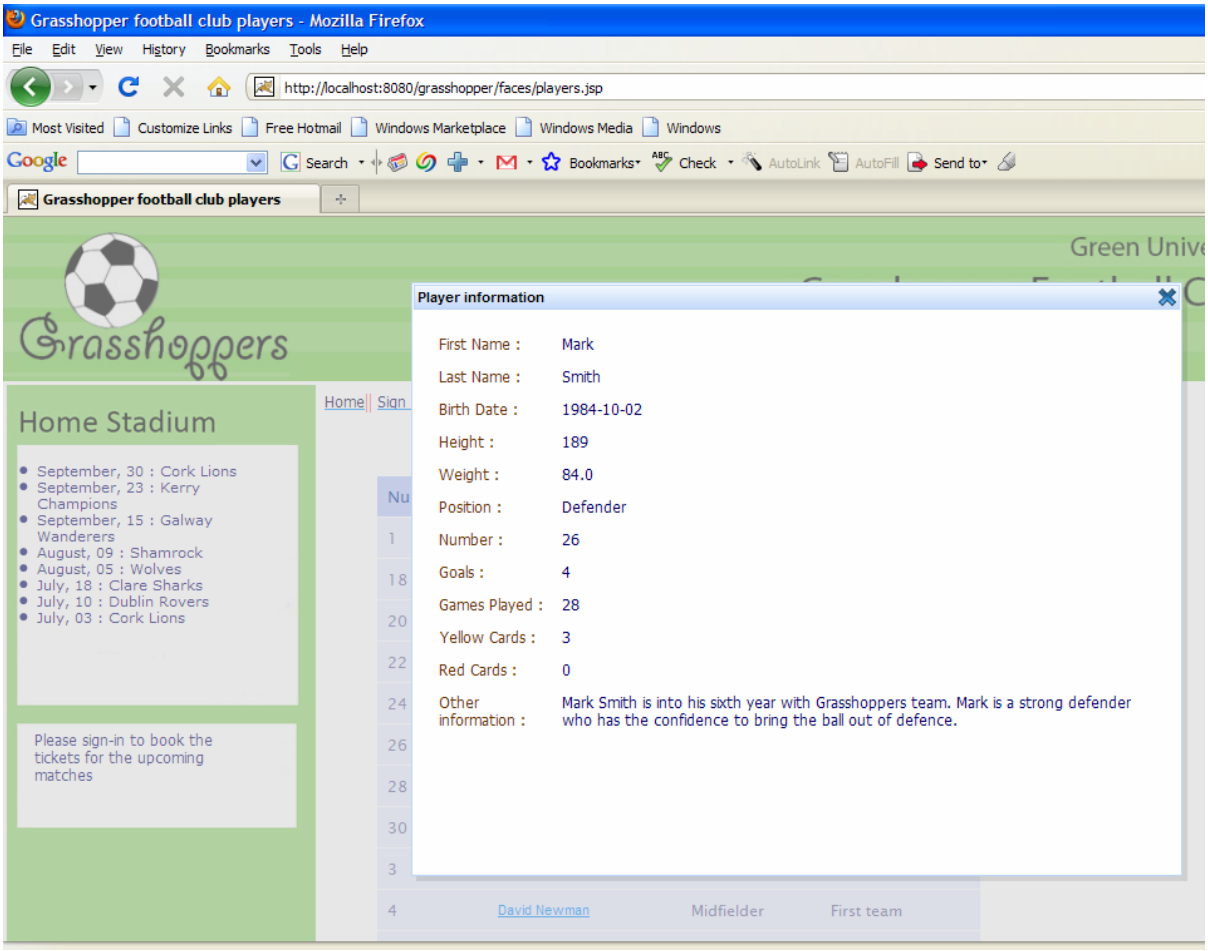

Figure 5.7.4 Player information

The information about the particular player is displayed with the JSF page. This page is generated by the player.jspf file.

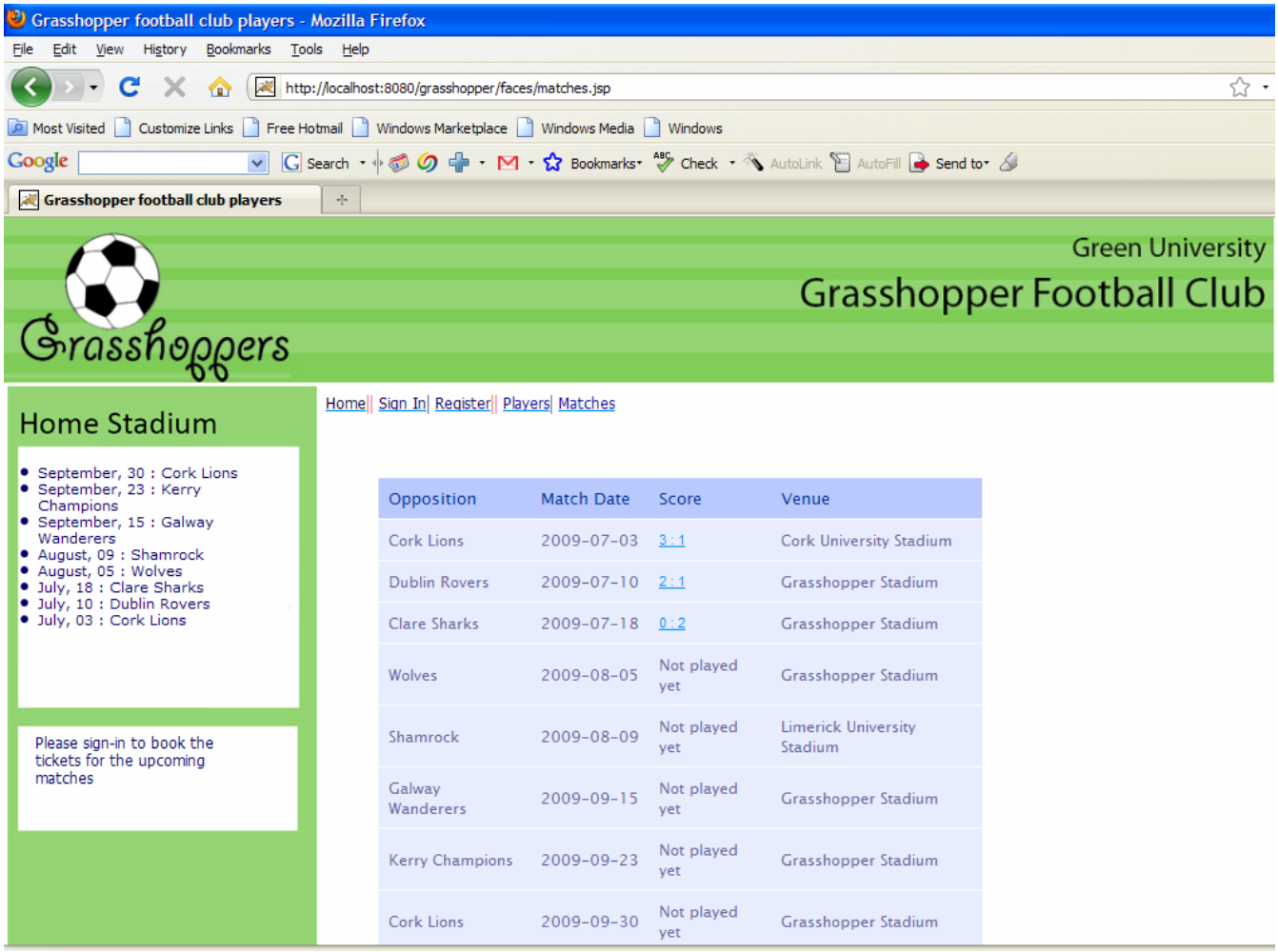

Figure 5.7.5 The Matches page

The Matches page is generated by the matches.jsp file. The page presents information about the played matches and about the future games. The analysis of a played match could be access by clicking on the score.

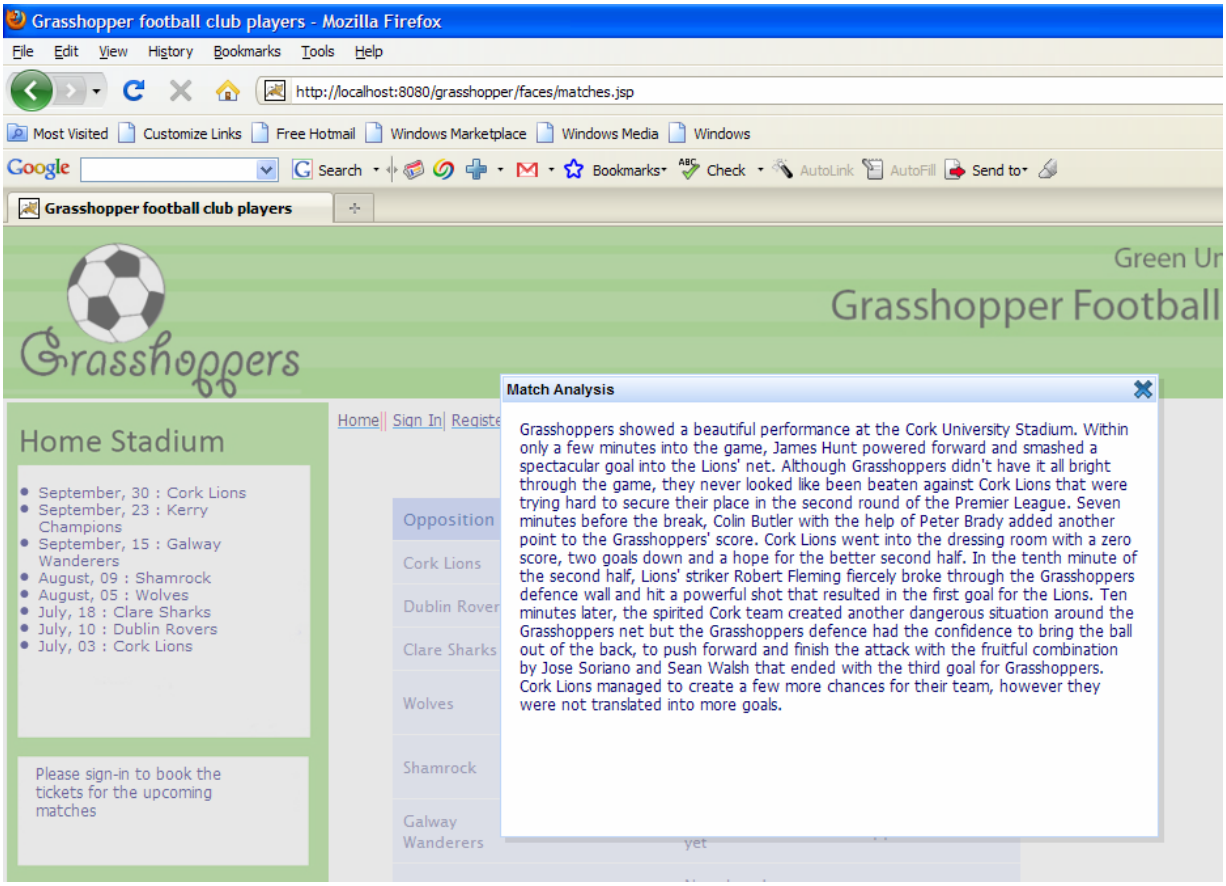

Figure 5.7.6 Match analysis

The Match Analysis JSF page is called within the Matches JSP page, by the click on the score link. The Match Analysis is generated by the matches.jspf file (the implementation process of the Match Analysis JSF was discussed earlier).

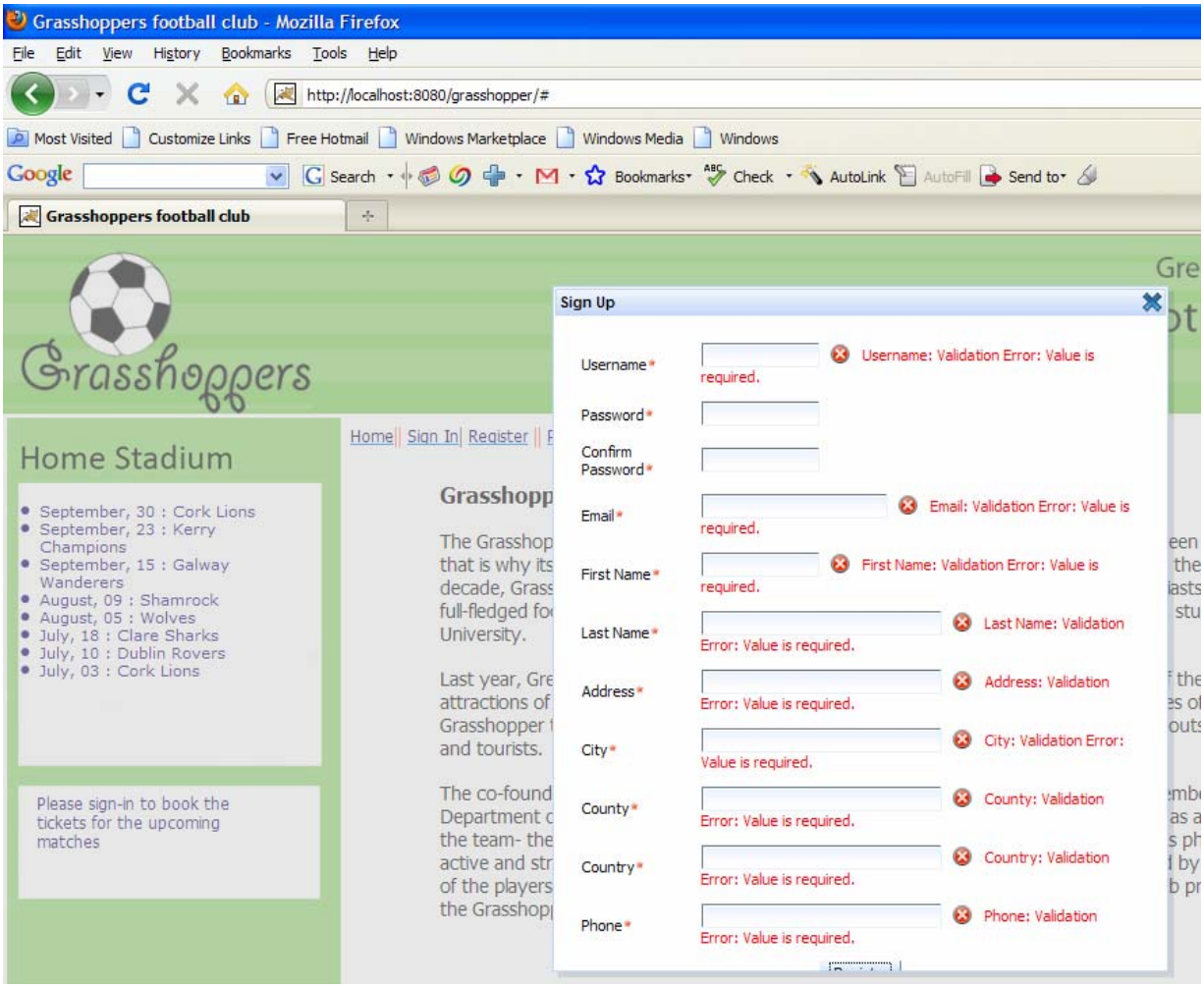

Figure 5.7.7 Validation of registration form

To register for a new account, the user has to fill in the form. The form validation functionality ensures that no user is registered if the mandatory fields are not filled in or they are filled with the data of a wrong format.

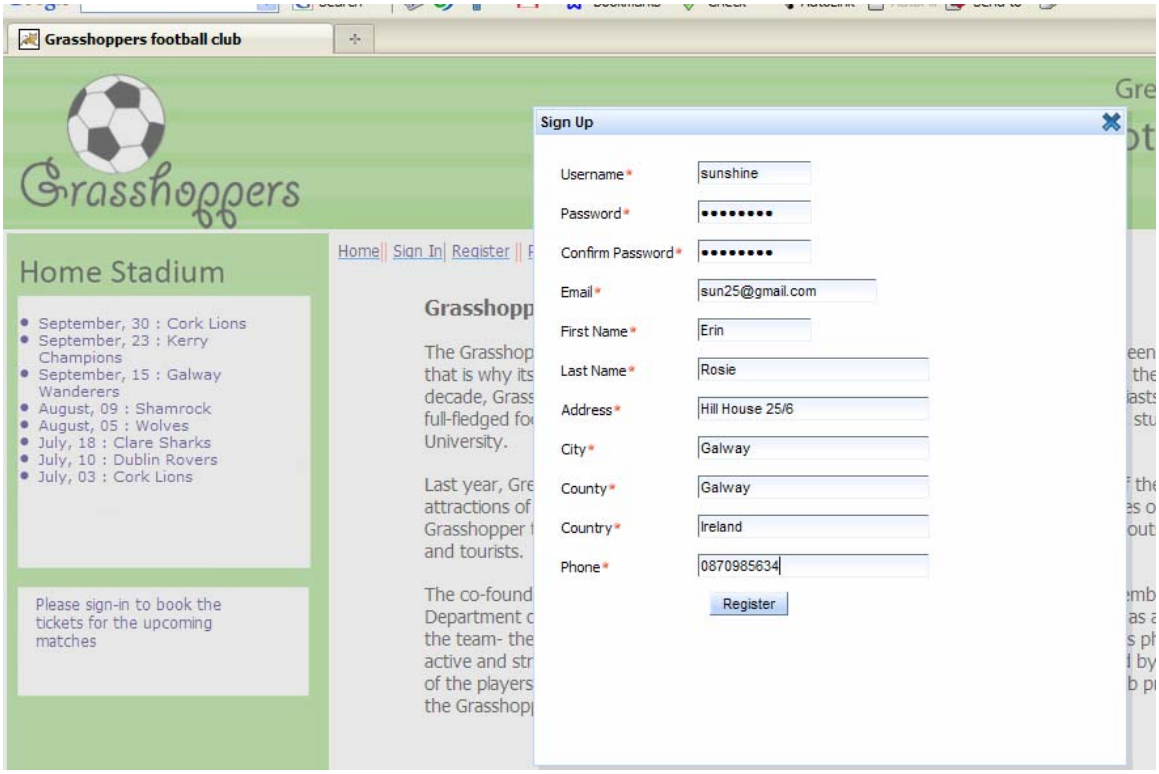

Figure 5.7.8 Registration form

Once the registration form is filled in correctly, the user hits Register button and the registration process completes.

Figure 5.7.9 Ensure unique username

Should the user select an existing username for account registration, the registration process will halt, the user will be asked to enter a unique name. Therefore, the database username Unique constraint has been supported.

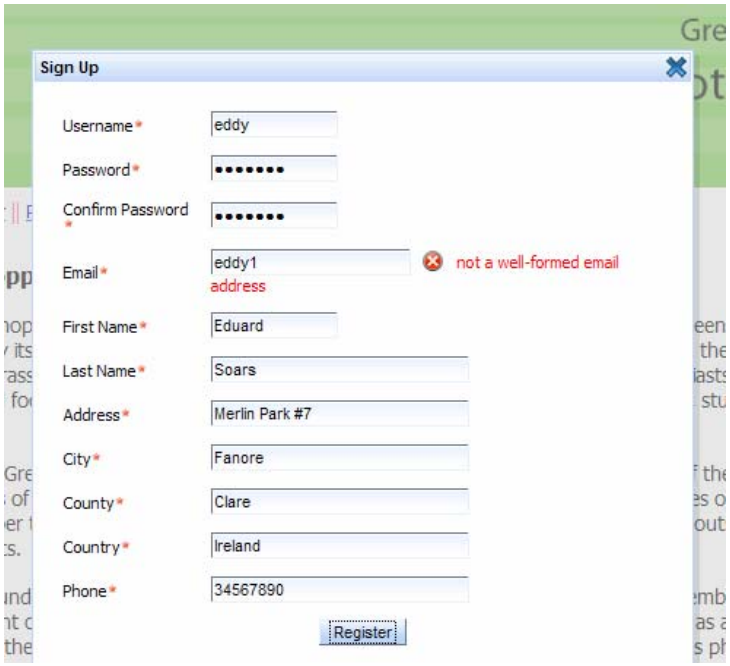

Figure 5.7.10 Attempt to register using invalid email address

Attempt to register with invalid email address will also fail. The email address field data is checked for a certain pattern.

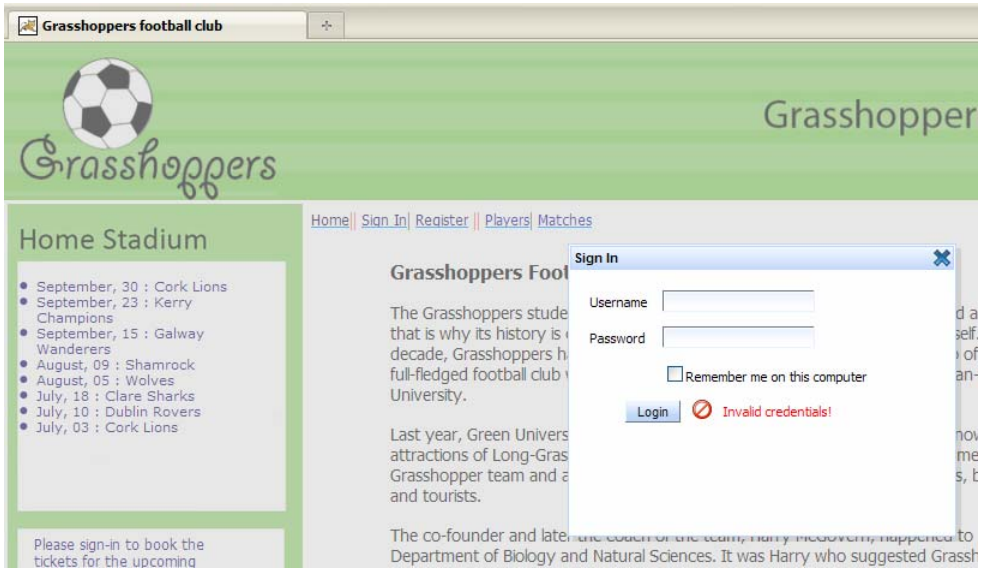

Figure 5.7.11 Attempt to sing-in using wrong credentials

If a user provides an invalid name or password in the sign-in form, the error message appears and the user is not able to sign-in. The Sign In JSF is produced by the auth.jspf file.

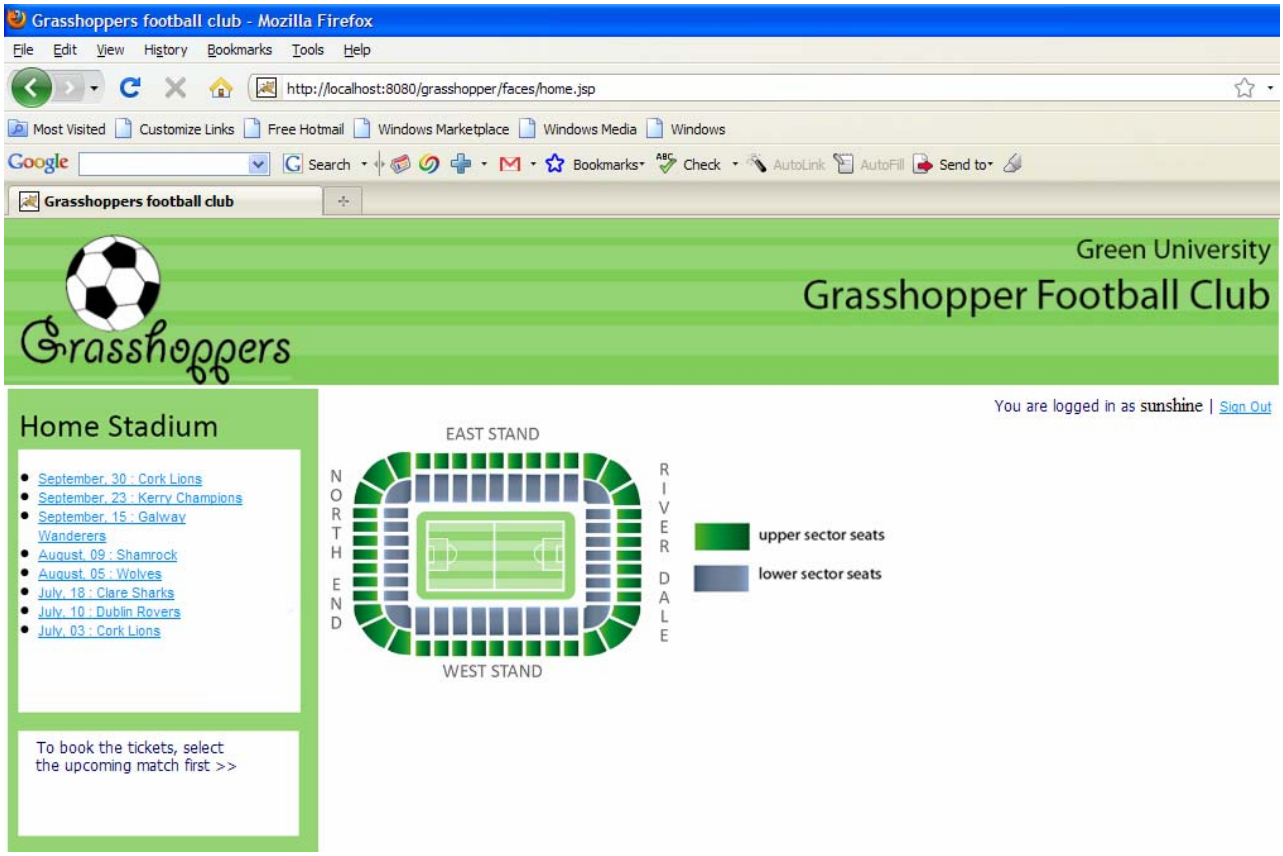

Figure 5.7.12 Successful log-in

With the correct credentials, the user signs-in into his/her account. The registered users can book tickets for the future matches. To book a ticket, the user should select an upcoming match in the left pane. This page is handled by the home.jsp file.

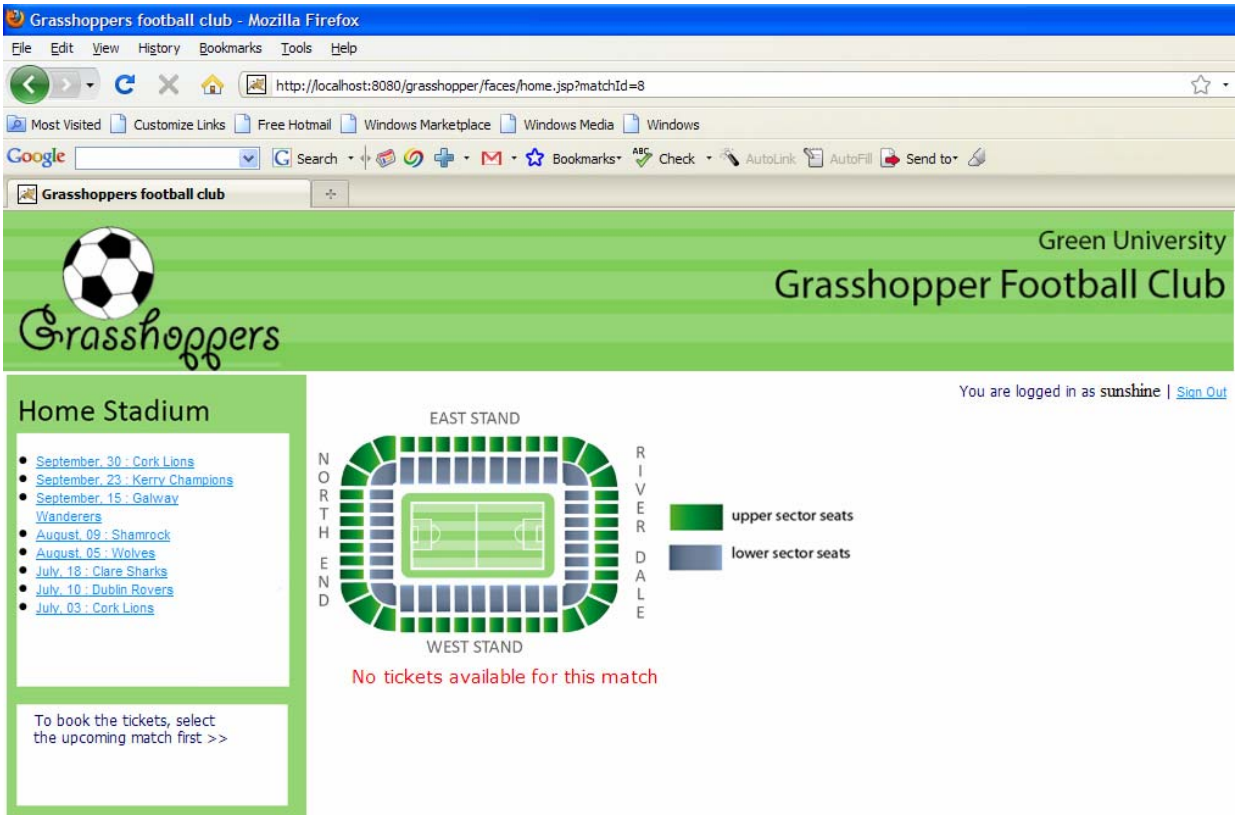

Figure 5.7.13 Attempt to book the tickets for an unavailable match

If there is no tickets available for the selected match, an according message appears.

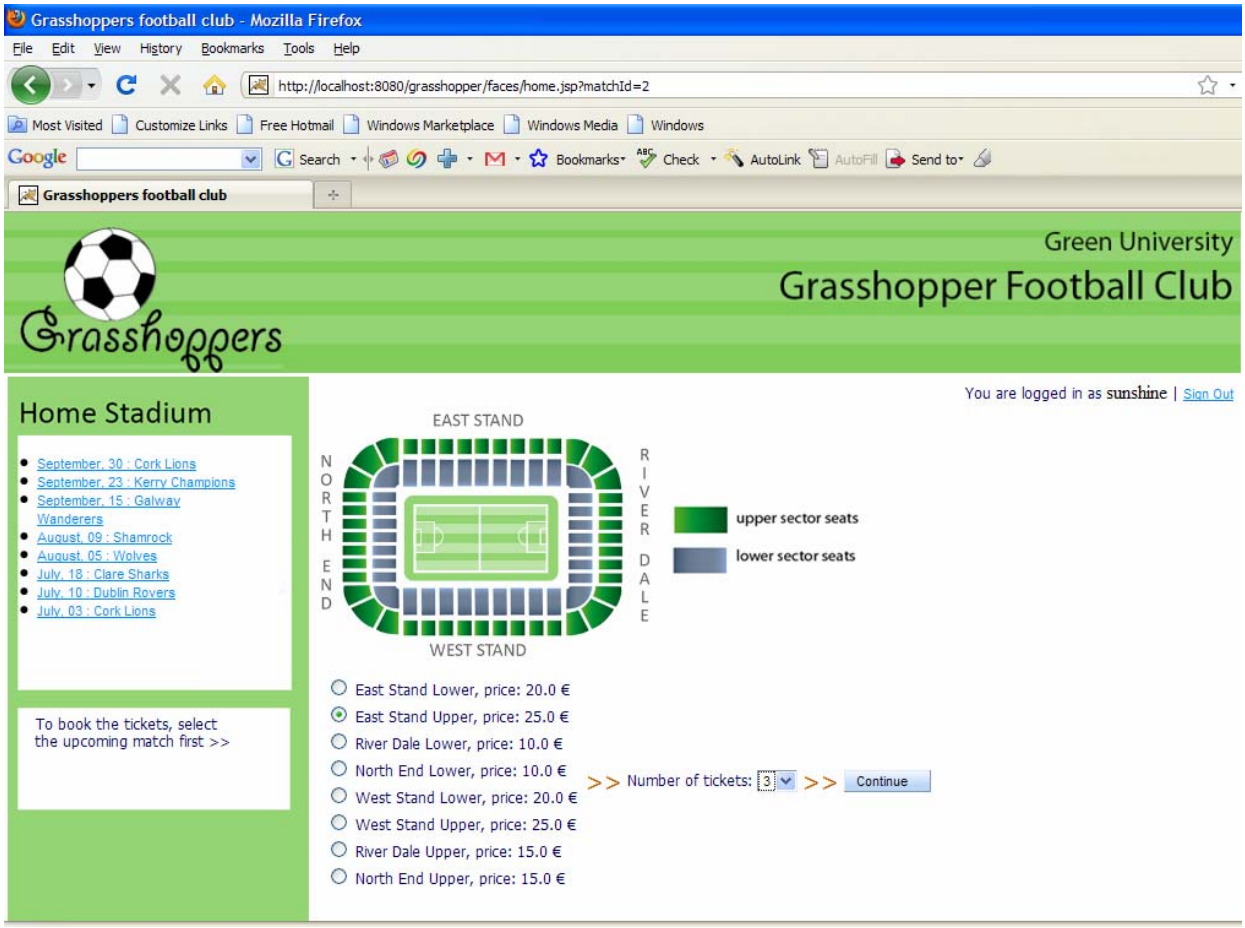

Figure 5.7.14 Booking tickets

If tickets are available for booking, the information about the seat sections and tickets price appears. The Grasshoppers stadium seating plan helps the user choose desired section. The user selects a seat section and a number of tickets and the booking process continues.

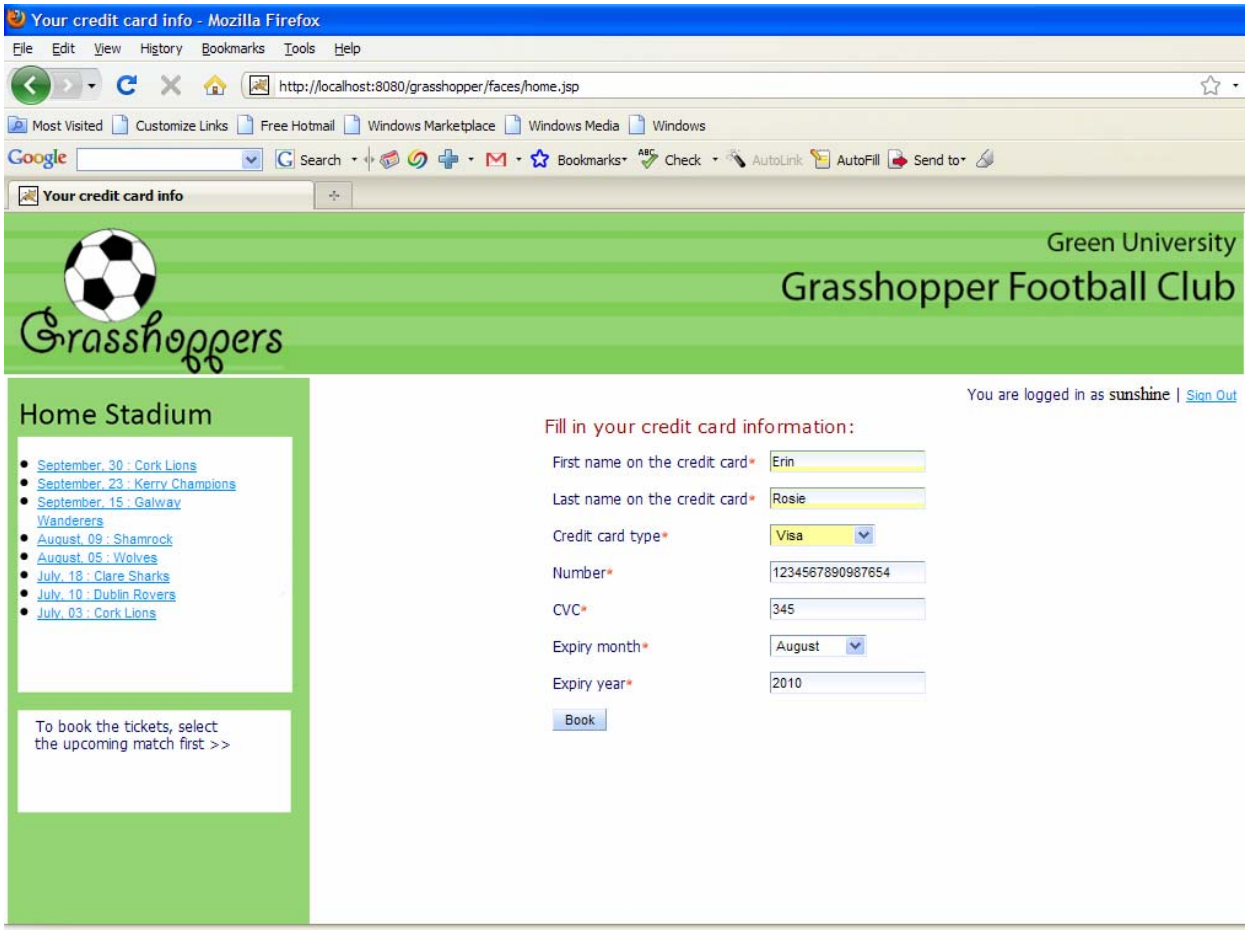

Figure 5.7.15 Credit card details

To reserve a ticket, the user has to provide credit card details. The credit card page is generated by the credit_card.jsp. The online credit card verification is beyond the scope of this study. When the credit card form is completed, the booking proceeds.

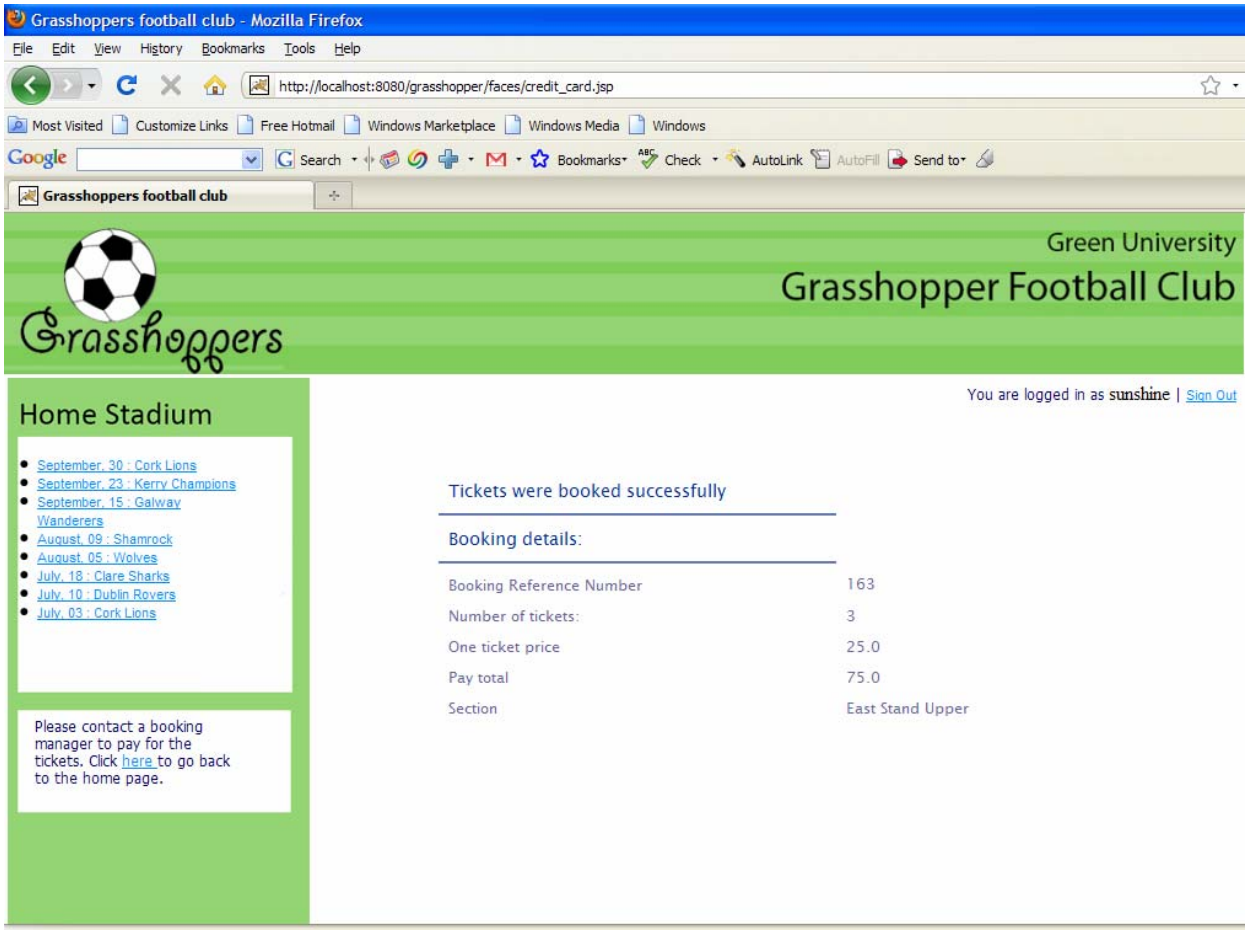

Figure 5.7.16 Successful booking

If a booking is successful, a booking summary report appears. This page is generated by the booking.jsp file.

5.8 Summary

This chapter provided an overview of the implementation processes of different technologies involved in the Grasshopper application design. A detailed insight into the process of building object-relational mapping with both Hibernate and EclipseLink technologies was presented. A layered approach for the persistence implementation, utilized with DAO pattern,

resulted in a clean code. The DAO layer introduced abstraction, which allowed decoupling of the different tiers of the system's architecture. The next chapter will demonstrate how the extension of the DAO pattern into the testing system provides abstraction from the particular persistence technology.

Chapter 6 - Performance Test

This chapter describes the core activities of the performance testing. Planning and designing stage identifies test key scenarios and establishes metrics to be collected. Implementation of test design presents an overview of the test system integrated into Grasshopper architecture. The test execution and analysis of the results conclude the chapter.

6.1 Planning and Designing Tests

The objective of the testing in this research is to compare performance characteristics of Hibernate and EclipseLink ORM technologies. The test takes the form of performance comparison of two configurations of the Grasshopper system – with Hibernate and EclipseLink.

Meier (2007) described performance testing as the type of testing that "determines or validates the speed, scalability, and/or stability characteristics of the system or application under test". This research is concerned with testing execution time of standard CRUD (create, read, update, and delete) database operations. For this purpose, a set of SQL queries is executed against Grasshopper database using both Grasshopper application configuration with Hibernate and EclipseLink. The set of queries testing CRUD operations is stated in the Table 6.1.1.

Table 6.1.1

A set of queries for testing CRUD operations

Query #	Operation	Query
Q1	Create	Insert tickets for upcoming matches
Q2	Read	Find the number of tickets sold for a particular match
Q3	Read	Find the number of seats on the stadium

Q4	Update	Reserve tickets
Q5	Delete	Delete tickets

The second set of tests is designed to measure performance of SQL queries that involve table joins. This kind of test allows discovering how effectively the two compared technologies handle table joins with different relationships – one-to-one, one-to-many, and many-to-many.

Table 6.1.2

A set of queries for testing table joins

Query #	Business transaction	SQL query	Number of tables impacted	Relationship
Q1	Browse item	Find an account that belongs to a particular user	2	1:1
Q2	Browse item	Find players participating in a particular match	3	M:N
Q3	Browse item	Find tickets that are booked under a particular booking	2	1:N

6.2 Implementation of Test Design

For the purpose of this research, a Java based testing application was developed and integrated within the Grasshopper project. Therefore, the testing environment builds on the same DAO design pattern as the Grasshopper application itself. The abstraction layer introduced by

DAO supports easily switching between the tested ORM technologies by changing testing application configuration parameters. With a little programming effort, the developed design allows switching to and testing any ORM framework, supported by Spring DAO.

The structure of the testing system is depicted in the Figure 6.2.1. The testing facility is comprised of the main testing Java files; resources needed for setting configuration of testing environment and property files that allow changing test parameters (see Appendix H for the full source code).

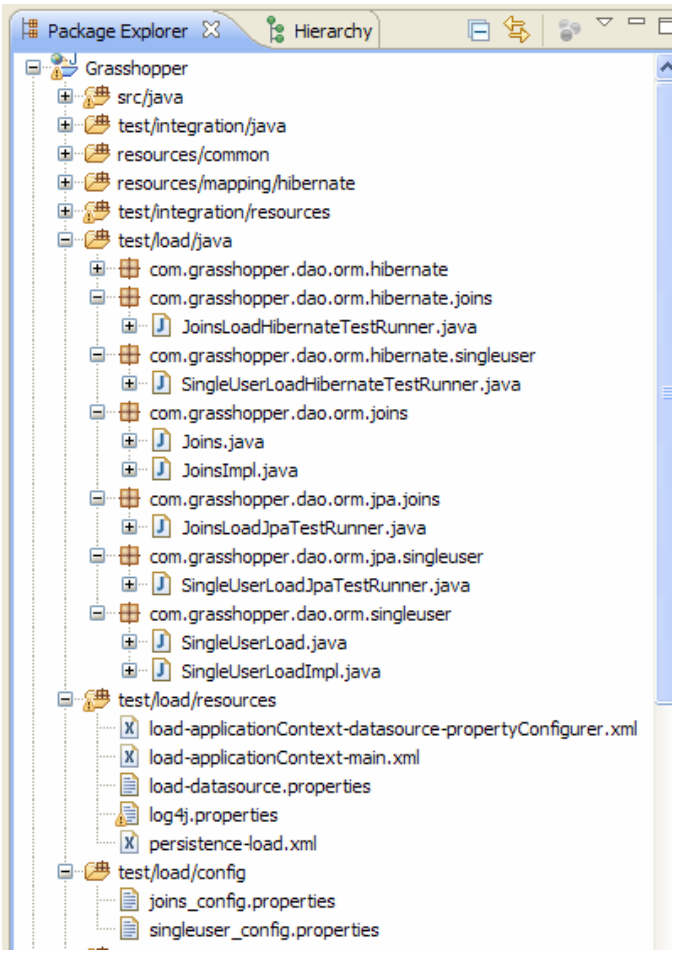

Figure 6.2.1 the structure of the test system

The testing system for the CRUD operations builds on the following strategy. The SingleUserLoad.java interface defines abstract methods needed for performing the tests. The

name of the interface indicates that the tests are performed under a single user load. The following code snippet shows an example of the methods, such as a method for deleting tickets, a method for counting sold tickets, and a method for counting seats on the stadium:

```
    /* Load tests (business logic methods) */
    Set<Ticket> insertTickets(Matches match, Set<Seat> seats);
    void reserveTickets(Set<Ticket> tickets, User user, int bookingAmount);
    void deleteAllTickets();
    void deleteTickets(Set<Ticket> tickets);
    Integer getSoldTicketsCount(Matches match);
    Integer getOverallSeatsCount();
```

Before running the tests, the test database has to be populated with the test data. For this purpose, a set of assisting methods has been defined, including a method for cleaning the database from data, a method for creating users, tickets, sections, seats and matches, as the following code demonstrates:

```
    /* Initialization methods */
    void cleanDB();
    User createTestUser();
    Set<Ticket> createTickets(Matches match, Set<Seat> seats);
    Set<Section> createSections(int amount);
    Set<Seat> createSeats(int amountOfRaws, int amountOfSeatsInRow,
          Set<Section> sections);
    Matches createMatch();
```

The SingleUserLoadImpl.java class implements SingleUserLoad interface. It implements abstract methods, defined in the SingleUserLoad interface, injecting them with the corresponding DAO methods, created and discussed earlier. Both, standard DAO methods, supported by the GenericDao, and user-defined DAO methods are utilised. For example, the getSoldTicketsCount() method is implemented with the corresponding user-defined method in the MatchesDao.java interface as follows:

```
      public Integer getSoldTicketsCount(Matches match) {
      return matchesDao.getSoldTicketsCount(match);
    }
```

The assisting cleanDB() method for cleaning the database utilizes generic deleteAll() method to delete all data from the tables:

```
public void cleanDB() {
        userDao.deleteAll();
        bookingDao.deleteAll();
        ticketDao.deleteAll();
        matchesDao.deleteAll();
        seatDao.deleteAll();
        sectionDao.deleteAll();
 }
```

Once the database operations methods have been defined, they are used to implement the test. The high view of the testing program workflow is depicted in the Figure 6.2.2.

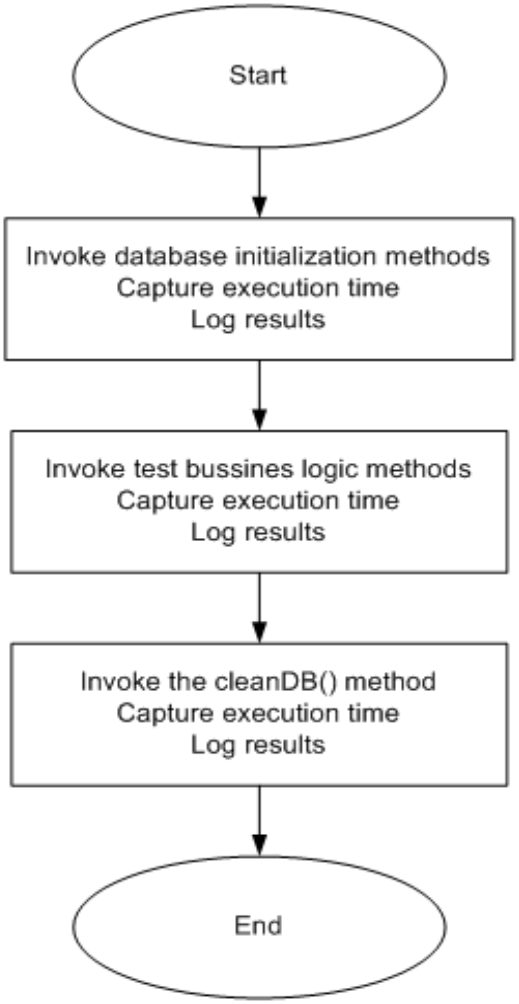

Figure 6.2.2 The test program workflow diagram

The initialization methods populate the test program with data needed to perform the test. The total execution time of database initialization is captured with the StopWatch utility, an API for timing from the Apache Sotware Foundation. The methods start(), stop(), get() and reset()

provide facilities to start the StopWatch, to stop the StopWatch, to get the time on the StopWatch, and to reset the StopWatch respectively. The results of this activity, such as the total execution time of database initialization are logged into a log file, for further analysis. The logging functionality is implemented by the means of the LogFactory, another utility by Apache Software Foundation. Once the database is ready for running the test, the methods providing for the test logic have being invoked. The execution performance of the each persistence operation has been captured and the results are logged. Additionally, the total work time, taken for performing the whole test set, is been measured. In the end, the cleanDB() method deletes all the data from the database. The time needed for the cleaning operation is logged into a log file.

The SingleUserLoadImpl.java class implements the testing functionalities by the means of DAO persistence methods, without a hint on a particular ORM technology. The test runner for a given ORM technology is utilized through the configuration settings of the Spring application context. The following code snippet from the SingleUserLoadHibernateTestRunner.java class shows how the test running program is configured for testing Hibernate:

```
public class SingleUserLoadHibernateTestRunner {

    public static void main(String[] args)throws Exception {

      ApplicationContext context = new ClassPathXmlApplicationContext(
            new String[] {
                  "test-applicationContext-hibernate-propertyConfigurer.xml",
                  "applicationContext-datasource-dbcp.xml",
                  "load-applicationContext-datasource-
propertyConfigurer.xml",
                  "applicationContext-hibernate.xml",
                  "load-applicationContext-main.xml" });
      SingleUserLoad singleUserLoad = (SingleUserLoad) context
            .getBean("singleUserLoad");
      SingleUserLoadImpl.setLoggerClass(SingleUserLoadHibernateTestRunner.cla
ss);

      singleUserLoad.executeTests();
    }
}
```

The next chunk of code, taken from the SingleUserLoadJpaTestRunner.java class demonstrates that as little alterations as changing application context configuration parameters, are needed to implement the test program for EclipseLink:

```
public class SingleUserLoadJpaTestRunner {

    public static void main(String[] args)throws Exception {

      ApplicationContext context = new ClassPathXmlApplicationContext(
            new String[] {
                  "load-applicationContext-datasource-
propertyConfigurer.xml",
                  "applicationContext-datasource-dbcp.xml",
                  "applicationContext-jpa.xml",
                  "load-applicationContext-main.xml" });
      SingleUserLoad singleUserLoad = (SingleUserLoad) context
            .getBean("singleUserLoad");
      SingleUserLoadImpl.setLoggerClass(SingleUserLoadJpaTestRunner.class);

      singleUserLoad.executeTests();
    }
}
```

The two examples stated above provide an evidence of advantages introduced by the abstraction layer in the application, utilizing the Spring DAO design pattern. Any ORM framework, supported by Spring DAO, can be implemented and tested without changing the logic of the test program.

The programs, testing table joins, were built on the same principles as the single user CRUD operation test described above. The full source code of the joins testing programs is stated in the Appendix H.

SingleUserLoadHibenateTestRunner.java, SingleUserLoadJpaTestRunner.java, JoinsLoadJpaTestRunner.java, and JoinsLoadJpaTestRunner.java classes, running the tests, can be executed either directly in the Java virtual machine environment, or functionalities programmed into the buil.xml file (see Appendix G for the source code) can be used to build the build_load_test Ant task for executing the whole test set.

6.3 Testing Environment

The tests were run on a Dell Inspiron laptop 1501 based on a AMD Turion 64 X2 Mobile Technology TL58 processor, with 1G of internal memory. The 120G SATA hard drive supported 5400 rpm speed. The operating system was MS Windows XP Home Edition. Only applications providing for the tests execution were running during the test performance. The tests were executed as the bat.files, built with Ant tool, from the Windows command line under the CMD shell. The testing environment could have cause an overhead in time measurements. However, there is no reason to suggest that the performance overhead, caused by the testing environment, influenced one persistence technology to a different degree than the other. This research is concerned with relative performance characteristics of two technologies, rather than their absolute values.

6.4 Tests Execution

The task for building tests executables were added to the build.xml file that is used for building the Grasshopper project with the Ant tool. As the Figure 6.4.1 illustrates, the target named build_load_tests, placed into build.xml file, handles building of the test project. As a result of running the build_load_tests Ant task, four executable .bat files were built for running joins tests and the CRUD operations for a single user tests for Hibernate and EclipseLink.

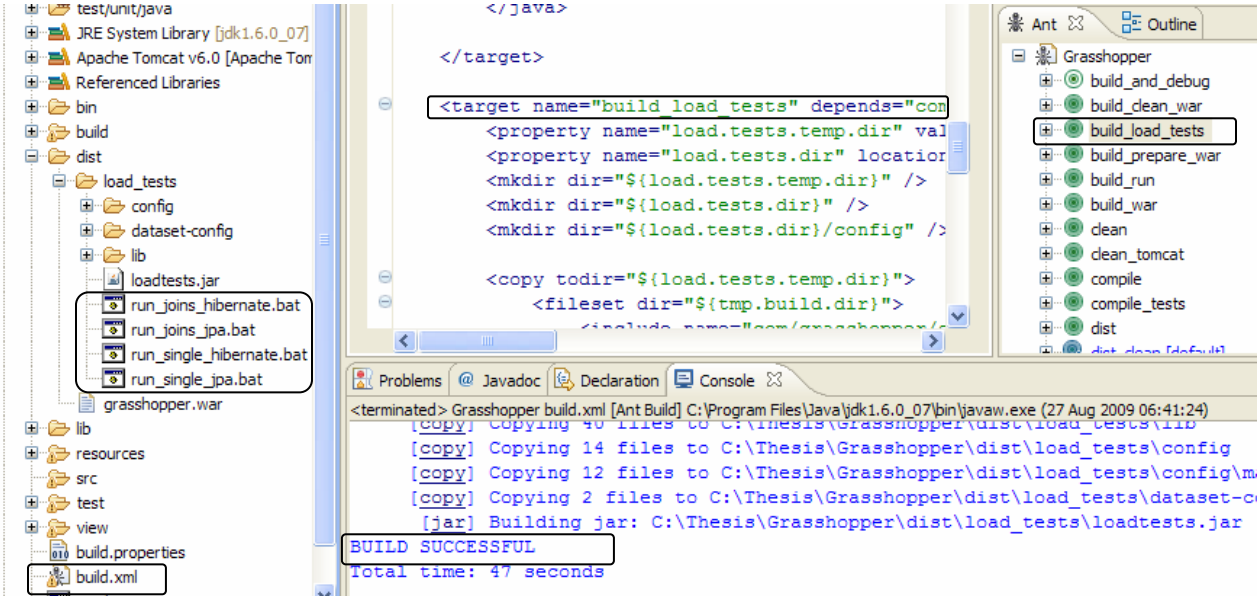

Figure 6.4.1 Building tests executables with the Ant task

Once the tests executables had been built, they were called from the Windows command line under the CMD shell. Upon their successful execution, the tests results were written into the log files (see Appendix H for the content of the log files and test parameters). Figure 6.4.1 shows the log files in the projects structure.

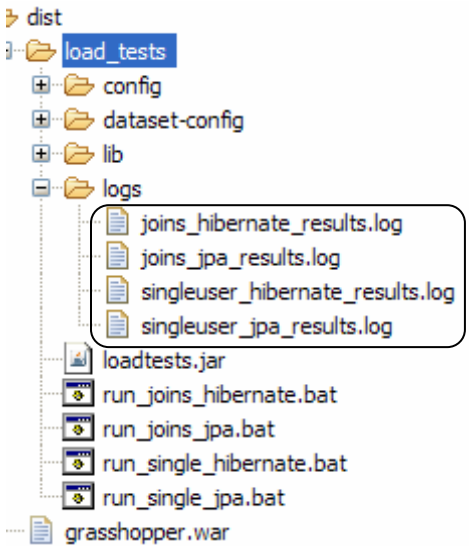

Figure 6.4.1 Tests results log files

6.5 Test Analysis

The table 6.5.1 contains the test results obtained from the singleuser_jpa_results.log singleuser_hibernate_results.log and files.

Table 6.5.1

Results of the CRUD operations test

Query #	Operation	Query	Records impacted	Execution time, ms	
				Hibernate	EclipseLink
Q1	Create	Insert tickets for upcoming matches	6000	300906	374609
Q2	Read	Find the number of tickets sold for a particular match	5000	281	844
Q3	Read	Find the number of seats on the stadium	6000	16	18
Q4	Update	Reserve tickets	5000	411781	421750
Q5	Delete	Delete tickets	6000	317531	278843
Total CRUD operations work time				1030515	1076064
Database initialization				295141	373219
Database cleaning				516	766

The Figures 6.5.1-6.5.4 show the charts of the results of the CRUD operations test.

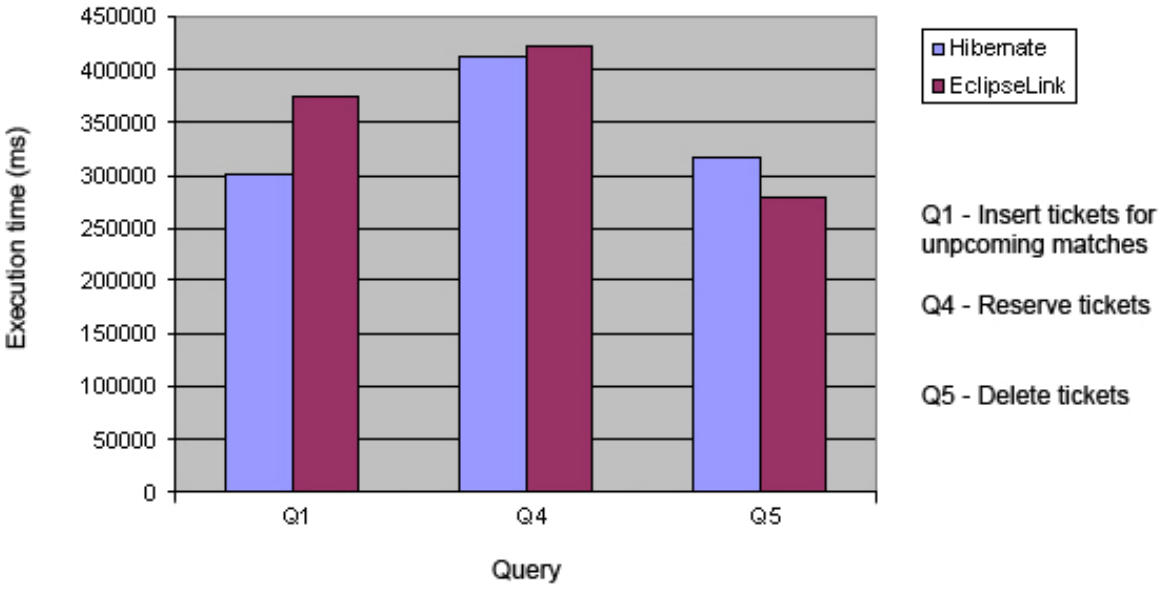

Figure 6.5.1 CRUD operations

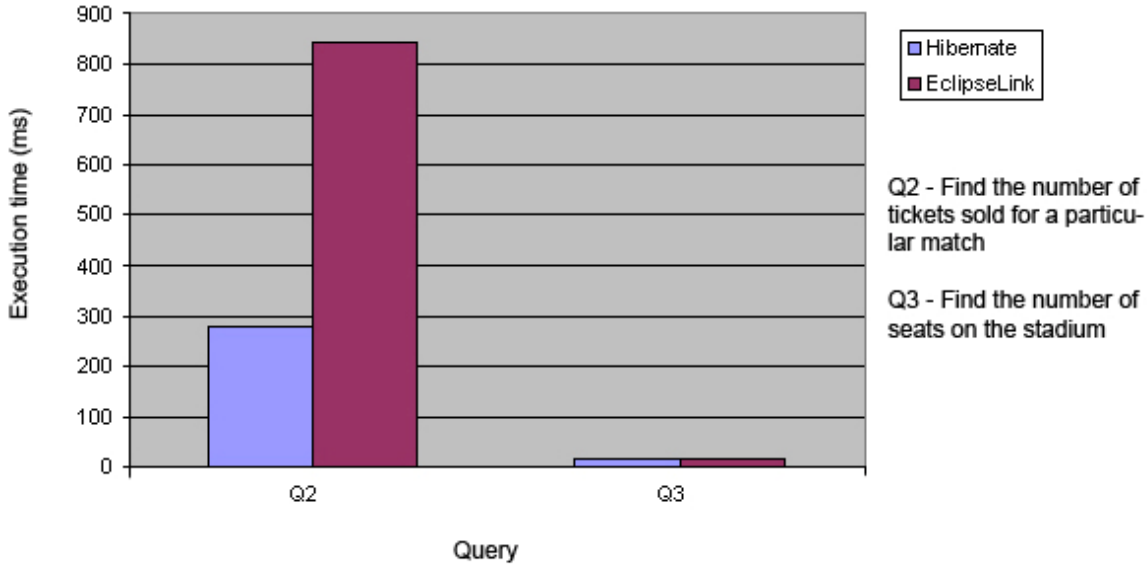

Figure 6.5.2 CRUD operations (continued)

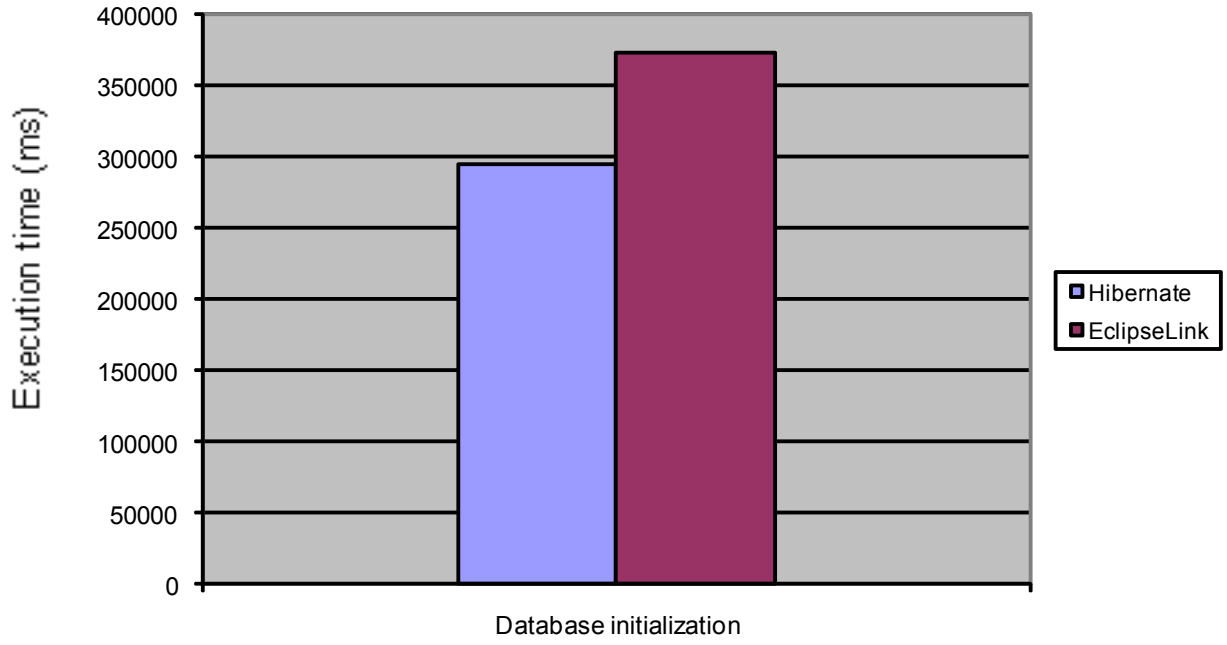

Figure 6.5.3 Database initialization for the CRUD operations

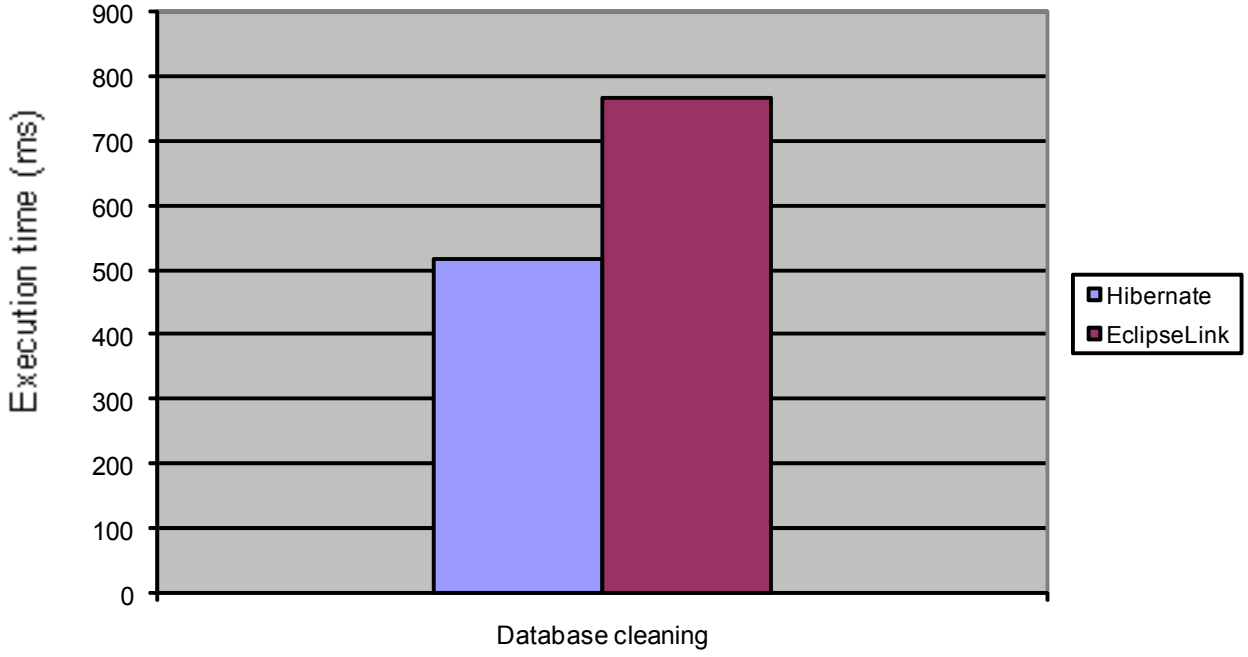

Figure 6.5.4 Database cleaning

The table 6.5.2 states the test results obtained from the joins_hibernate_results.log and joins_jpa_results.log files.

Table 6.5.2

Results for the table joins test

Query #	Business transaction	SQL query	Tables impacted	Relationship	Execution time, ms	
					Hibernate	EclipseLink
Q1	Browse item	Find an account that belongs to a particular user	2	1:1	171	343
Q2	Browse item	Find players participating in a particular match	3	M:N	47	94
Q3	Browse item	Find tickets that are booked under a particular booking	2	1:N	12	16
Total test operations work time					230	453
Database initialization					546329	617735
Database cleaning					1000	1012

The Figure 6.5.5 illustrates the charts of the results of the table joins test.

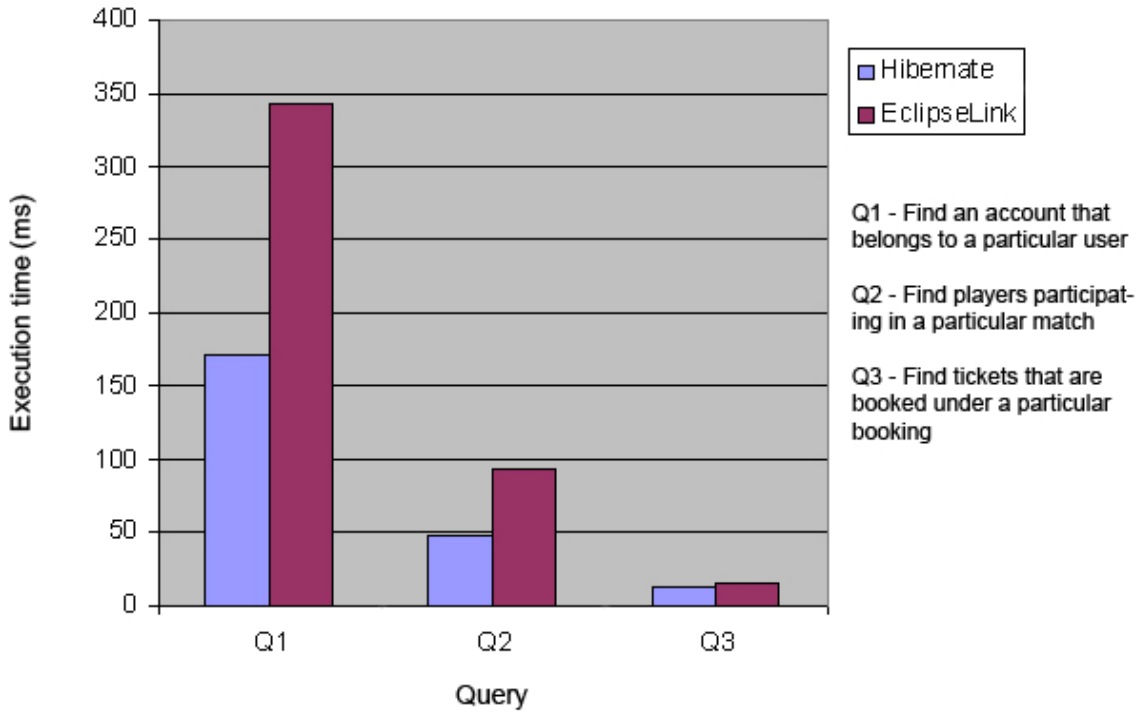

Figure 6.5.5 Table joins operation

The results of the CRUD operations test showed that Hibernate outperformed EclipseLink in the most cases. The performance of EclipseLink was better than the performance of Hibernate only in one case, that is, the execution of the delete tickets query. Hibernate, however, worked faster than EclipseLink executing the test assisting operations, such as database initialization and database cleaning.

The table joins test results demonstrated that Hibernate managed table joins more efficiently than EclipseLink. The Hibernate's execution performance was better for all types of table's relationship – one-to-one, one-to-many and many-to-many. In summary, EclipseLink produced better results than Hibernate only in one test case, the delete operation. Hibernate was faster executing the rest of the tests operations.

Conclusions

Although resource consumption was beyond the scope of this research, this researcher noticed that EclipseLink tended to require more resources than Hibernate. For example, when testing for the CRUD operations, EclipseLink test threw OutOf Memory exception once the number of reserved tickets exceeded 5000 with 6000 seats stadium. Allowing more memory for the operating of JVM did not resolve this issue. Hibernate test was successful with up to 10000 reserved tickets and 12000 seats on the stadium (see Appendix I for the results). It means that the baselines for the test parameters for a given test environment were twice lower for EclipseLink than those for Hibernate.

Throughout the project, EclipseLink was harder to work with than Hibernate. The problems arose early in the project during the technology discovery. The little information existing about EclipseLink is limited to the EclipseLink documentation, which is poor. Hibernate has a large responsive community with forums and existing body of code that can be used as example in many cases. A few existing forums for EclipseLink are filled rather with questions than answers.

Although both technologies stem from JPA, Hibernate proved to be more powerful framework offering richer API functionalities and extended set of Annotations. For example, Hibernate supports methods, such as findByExample() and findByCriteria(), that are not available in EclipseLink.

Another EclipseLink problem detected during the project development was related to the management of the DML operations. For example, with the same code structure Hibernate worked successfully with database operations while EclipseLink failed to execute certain operations, particular flush() method to synchronize the session with the state of the database.

Therefore, the researcher had to be careful to design the code that worked for both technologies. The constraints were related to EclipseLink because Hibernate was flexible and performed well under any scenarios.

Chapter 7 – Conclusions and Future Work

This study was designed to fill a gap in the literature, namely the need for performance investigation of EclipseLink, a newly emerged ORM tool, and comparison of its performance characteristics with those of Hibernate, a well established persistence framework.

Development of the web based application with the data stored in the relational database enabled performance comparison of the two ORM technologies. The developed performance testing system was integrated with the main project. The abstraction layer introduced into the application with Spring DAO made the system design highly modular and flexible. The persistence framework can be changed and tested within the application with no code alterations in the database, business logic, and presentation layers, as well as the logic of the testing system.

By using the testing system, it had been possible to compare the performance of Hibernate and EclipseLink. The test suites were developed to test execution performance of CRUD operations and table joins. It was found that the overall performance of Hibernate was better than that of EclipseLink. EclipseLink produced better result only in one test case that was a delete operation in the CRUD set of tests. Hibernate outperformed EclipseLink in the rest of the test scenarios.

During running the performance tests it came to light that EclipseLink was more resource consuming technology than Hibernate. For more comprehensive results, however, the tests should be run within a more powerful testing environment with larger sets of testing data.

Throughout the course of this research, Hibernate not only proved to be more efficient technology, but it was also easier to use when compared to EclipseLink. Some issues with EclipseLink's management of database operations surfaced during the development process. For instance, in some cases, EclipseLink failed to perform flush() operation to synchronize the

session with the state of the database. Using the same code design, Hibernate did not run into this problem. Hibernate also exposed more flexibility, offering a richer set of Annotations and API methods, such as findByExample() and findByCriteria(), which were not supported by EclipseLink. With regards to building mapping files, both technologies facilitated the same functionalities.

Finally, the implementation process with EclipseLink was slowed down by the fact that little supporting literature existed. In some cases, it was hard to find explanation of the exceptions thrown by the framework. On the other hand, Hibernate is well documented, well supported in the academic libraries and has extensive community forums.

The reason for the uncovered issues with EclipseLink could be related to the fact that this is a relatively new technology. Although it stems from the Oracle's TopLink ORM product, which has been successful in the persistence field for almost twelve years, EclipseLink got a life of its own only three years ago, and it still has a room for improvement. Hibernate, on the other hand, has been on the persistence landscape since 2002. As an open source solution, it has been continuously improving, benefiting from the large body of knowledge of its supporters.

In the future, this researcher would like to extend the capabilities of the developed system to facilitate more versatile set of queries to investigate different aspects of performance. Designing test suites for the multiple concurrent users running the queries would allow discovering the scalability of the system. The case studies of the ORM tools performance with other databases would enable more comprehensive analysis. The application can be further enhanced by developing a chart building module for the graphical presentation of the test results. The literature would benefit from investigation into the more complex cases of object-relational mapping structures, such as inheritance and composition.

References

Bryman, A. & Bell, E. (2003). *Business research methods*. Oxford University Press. Retrieved October 4, 2008, from Books24x7 database.

Eclipse Foundation (2009). EclipseLink. Retrieved June 30, from http://www.eclipse.org/eclipselink/

Fisher, P., T. & Duskis, S. (2009). Spring persistence: A running start. *FirstPress*. Retrieved July 18, 2009, from Books24x7 database.

Giametta, C. (2009). *Pro Flex on Spring.* Apress. Retrieved July 2, 2009, from Books24x7 database.

Gilmore, W., J. (2008). *Beginning PHP and MySQL: from novice to professional.* Apress. Retrieved July 12, 2009, from Books24x7 database.

Goncalves, A. (2009). *Beginning Java EE 6 platform with GlassFish3: From novice to professional.* Apress. Retrieved July 10, 2009, from Books24x7 database.

Heffelfinger, D., R. (2007). *Java EE 5 development using GlassFish Application Server.* Packt Publishing. Retrieved July 10, 2009, from Books24x7 database.

Hibernate (2004). Community Documentation. Architecture. Retrieved June 29, 2009, from http://docs.jboss.org/hibernate/stable/core/reference/en/html/architecture.html

IBM Corp. (n. d.). *Apache OpenJPA user's guide.* Retrieved June 30, 2009, from http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.ejbfep.multiplatform.doc/info/ae/ae/cejb_persistence.html

Issarny, V., Caporuscio, M. & Georgantas, N. (2007). A perspective on the future of middleware-based software engineering. *FOSE'07: 2007 Future of Software Engineering.* IEEE Computer Society. 244-258. Retrieved March 18, 2009, from ACM database.

Java_Source.net. (n. d.). Open source persistence frameworks in Java. Retrieved May 23, 2009, from http://java-source.net/open-source/persistence

Johnson, R. (2005). J2EE development frameworks. *Computers.* Vol. 38. Issue 1. 107-110. Retrieved March 12, 2009, from IEEE Xplore database.

Johnson, R. et al. (2005). *Professional Java development with the Spring Framework*. John Wiley & Sons. Retrieved July 2, 2009, from Books24x7 database.

Kodali, R., R., Wetherbee, J. & Zadrozny, P. (2006). *Beginning EJB 3 application development: From novice to professional.* Apress. Retrieved June 30, 2009, from Books24x7 database.

Martin, B. E. (2005). Uncovering database access optimizations in the middle tier with TORPEDO. *ICDE 2005: Proceedings of the 21st International Conference on Data Engineering*, 916-926. Retrieved March 12, 2009, from IEEE Xplore database.

Meier, J. D. (2007) et al. *Performance testing guidance for web applications: patterns & practices.* Microsoft Press. Retrived May, 2009, from Books24x7 database.

Minter, D. & Linwood, J. (2006). *Beginning Hibernate: Form novice to professional.* Apress. Retrieved June 29, 2009, from Book24x7 database.

Moodie, M. (2007). *Pro Apache Tomcat 6.* Apress. Retrieved July 12, 2009, from Books24x7 database.

Mukhar, K., Zelenak, C., Weaver, J., L. & Crume, J. (2006). *Beginning Java EE 5 Platform: From novice to professional.* Apress. Retrieved July 10, 2009, from Books24x7 database.

O'Neil, E., J. (2008). *Object/relational mapping 2008: Hibernate and the Entity Data Model (EDM).* SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data. 1351-1356. Retrieved October 24, 2008, from ACM database.

Oracle Corp. (2006). Oracle TopLink developer's guide. 10g (10.1.3.1.0). *Understanding the EJB 3.0 JPA entity architecture.* Retrieved June 30, 2009, from http://www.oracle.com/technology/products/ias/toplink/doc/10131/MAIN/_html/undtldev015.htm

Oracle Corp. (2007). Oracle proposes open source persistence project at Eclipse Foundation. Oracle TopLink contributed to open source community. *Oracle Press Release.* Retrieved June 29, 2009, from http://www.oracle.com/corporate/press/2007_mar/OpenSource-TopLink.html

Russel, C. (2008). Bridging the Object-Relational Divide. *ACM Queue*. *6, 3, 16-26*. Retrieved October 24, 2008, from ACM database.

Sam-Bodden, B. (2006). Beginning POJOs: novice to professional. Apress. Retrieved June 29, 2009, from Books24x7 database.

W3Schools.com (n. d.). *AJAX introduction.* Retrieved July 10, 2009, from http://www.w3schools.com/Ajax/ajax_intro.asp

Wahli, U. et al. (2008). WebSphere Application Server version 6.1 feature pack for EJB 3.0. IBM Redbooks. Retrieved June 30, 2009, from Books24x7 database.

Ziemniak, P., Sakowicz, B. & Napieralski, A. (2007). Object oriented application cooperation methods with relational database (ORM) based on J2EE technology. *CADSM'07: 9th International Conference – The Experience of Designing and Applications of CAD Systems in Microelectronics*, 327-330. Retrieved March 12, 2009, from IEEE Xplore database.

Zyl, P., V., Kourie, D., G. & Boake, A. (2006). *Comparing the performance of object databases and ORM tools.* SAICSIT '06: Proceedings of the 2006 annual research conference of the South African institute of computer scientists and information technologies on IT research in developing countries. Vol. 204. 1-11. Retrieved October 24, 2008, from ACM database.

Appendix A

The Database Scripts

Create Grasshopper database

```
CREATE DATABASE IF NOT EXISTS grasshopper
      CHARACTER SET utf8 COLLATE utf8_general_ci;
USE grasshopper;

CREATE USER 'grasshopper'@'localhost' IDENTIFIED BY 'gsh09';
GRANT ALL PRIVILEGES ON *.* TO 'grasshopper'@'localhost';

CREATE USER 'grasshopper'@'%' IDENTIFIED BY 'gsh09';
GRANT ALL PRIVILEGES ON *.* TO 'grasshopper'@'%';
```

Create Grasshopper_load database

```
CREATE DATABASE IF NOT EXISTS grasshopper_load
      CHARACTER SET utf8 COLLATE utf8_general_ci;
USE grasshopper_load;

CREATE USER 'test'@'localhost' IDENTIFIED BY 'test';
GRANT ALL PRIVILEGES ON *.* TO 'test'@'localhost';
```

Create tables for Grasshopper and Grasshopper_load databases

```
DROP TABLE IF EXISTS authorities, account, credit_card, ticket, booking,
   user, participation, player, seatprice, seat, section, matches;

CREATE TABLE user (
   user_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
   first_name VARCHAR(25) NOT NULL,
   last_name VARCHAR(25) NOT NULL,
   address VARCHAR(70) NOT NULL,
   city VARCHAR(30) NOT NULL,
   county VARCHAR(30) NOT NULL,
   country VARCHAR(25) NOT NULL,
   phone VARCHAR(15) NOT NULL,
   email VARCHAR(45) NOT NULL,
   CONSTRAINT user_pk PRIMARY KEY (user_id));

CREATE TABLE account (
   account_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
   username VARCHAR(25) NOT NULL UNIQUE,
   enabled BIT NOT NULL DEFAULT true,
   password VARCHAR(15) NOT NULL,
   user_id INT UNSIGNED NOT NULL,
   CONSTRAINT account_pk PRIMARY KEY (account_id),
   CONSTRAINT account_user_id_fk FOREIGN KEY (user_id) REFERENCES
user(user_id) ON UPDATE CASCADE ON DELETE CASCADE);
```

```
CREATE TABLE authorities (
      username VARCHAR(25) NOT NULL,
      authority VARCHAR(25) NOT NULL DEFAULT 'ROLE_USER',
      CONSTRAINT authorities_pk PRIMARY KEY (username, authority),
      CONSTRAINT username_account_fk FOREIGN KEY (username) REFERENCES
account(username) ON UPDATE CASCADE ON DELETE CASCADE);

CREATE TABLE credit_card (
   card_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
   fname_on_card VARCHAR(25) NOT NULL,
   lname_on_card VARCHAR(25) NOT NULL,
   type VARCHAR(15) NOT NULL,
   number CHAR(16) NOT NULL,
   cvc CHAR(3) NOT NULL,
   expiry_month CHAR(10) NOT NULL,
   expiry_year DATE NOT NULL,
   user_id INT UNSIGNED NOT NULL,
   CONSTRAINT credit_card_pk PRIMARY KEY (card_id),
   CONSTRAINT credit_card_user_id_fk FOREIGN KEY (user_id) REFERENCES
user(user_id) ON UPDATE CASCADE ON DELETE CASCADE);

CREATE TABLE player (
   player_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
   team ENUM ('First team', 'Reserve team') NOT NULL,
   first_name VARCHAR(25) NOT NULL,
   last_name VARCHAR(25) NOT NULL,
   birth_date DATE NOT NULL,
   height SMALLINT UNSIGNED NOT NULL,
   weight DECIMAL(4,1) NOT NULL,
   position ENUM ('Midfielder', 'Defender', 'Striker', 'Goalkeeper') NOT
NULL,
   player_num SMALLINT UNSIGNED UNIQUE NOT NULL,
   goals SMALLINT UNSIGNED NOT NULL,
   games_played SMALLINT UNSIGNED NOT NULL,
   yellow_cards SMALLINT UNSIGNED NOT NULL,
   red_cards SMALLINT UNSIGNED NOT NULL,
   notes VARCHAR(5000) NOT NULL,
   CONSTRAINT player_pk PRIMARY KEY (player_id));

CREATE TABLE matches (
   match_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
   match_date DATE NOT NULL,
   opposition VARCHAR(45) NOT NULL,
   competition VARCHAR(50) NOT NULL,
   kick_off TIME NOT NULL,
   venue VARCHAR(50) NOT NULL,
   score_grassh INT UNSIGNED,
   score_oppos INT UNSIGNED,
   analysis VARCHAR(5000),
   CONSTRAINT match_pk PRIMARY KEY (match_id));

CREATE TABLE participation (
   part_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
   player_id INT UNSIGNED NOT NULL,
   match_id INT UNSIGNED NOT NULL,
   CONSTRAINT part_pk PRIMARY KEY (part_id),
```

```
   CONSTRAINT part_player_id_fk FOREIGN KEY (player_id) REFERENCES
player(player_id),
   CONSTRAINT part_match_id_fk FOREIGN KEY (match_id) REFERENCES
matches(match_id) ON UPDATE CASCADE ON DELETE CASCADE,
   CONSTRAINT part_unique UNIQUE (player_id, match_id));

CREATE TABLE booking (
   book_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
   user_id INT UNSIGNED NOT NULL,
   pay_total DECIMAL(5,2) NOT NULL,
   paid_date DATE,
   pay_method ENUM('Cash', 'Credit Card'),
   CONSTRAINT booking_pk PRIMARY KEY (book_id),
   CONSTRAINT booking_user_id_fk FOREIGN KEY (user_id) REFERENCES
user(user_id) ON UPDATE CASCADE ON DELETE CASCADE);

CREATE TABLE section (
   section_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
   section_name VARCHAR(20) NOT NULL,
   CONSTRAINT section_pk PRIMARY KEY (section_id));

CREATE TABLE seat (
   seat_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
   section_id INT UNSIGNED NOT NULL,
   raw_num INT UNSIGNED NOT NULL,
   seat_num INT UNSIGNED NOT NULL,
   CONSTRAINT seat_pk PRIMARY KEY (seat_id),
   CONSTRAINT seat_section_id_fk FOREIGN KEY (section_id) REFERENCES
section(section_id) ON UPDATE CASCADE ON DELETE CASCADE,
   CONSTRAINT seat_seat_num_unique UNIQUE (section_id, raw_num, seat_num));

CREATE TABLE seatprice (
   seatprice_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
   match_id INT UNSIGNED NOT NULL,
   section_id INT UNSIGNED NOT NULL,
   price DECIMAL(4,2) NOT NULL,
   CONSTRAINT seatprice_pk PRIMARY KEY (seatprice_id),
   CONSTRAINT seatprice_match_id_fk FOREIGN KEY (match_id) REFERENCES
matches(match_id) ON UPDATE CASCADE ON DELETE CASCADE,
   CONSTRAINT seatprice_section_id_fk FOREIGN KEY (section_id) REFERENCES
seat(section_id) ON UPDATE CASCADE ON DELETE CASCADE,
   CONSTRAINT seatprice_unique UNIQUE (match_id, section_id));

CREATE TABLE ticket (
   ticket_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
   match_id INT UNSIGNED NOT NULL,
   seat_id INT UNSIGNED NOT NULL,
   book_id INT UNSIGNED,
   available BOOLEAN NOT NULL DEFAULT 1,
   CONSTRAINT ticket_pk PRIMARY KEY (ticket_id),
   CONSTRAINT ticket_match_id_fk FOREIGN KEY (match_id) REFERENCES
matches(match_id) ON UPDATE CASCADE ON DELETE CASCADE,
   CONSTRAINT ticket_seat_id_fk FOREIGN KEY (seat_id) REFERENCES
seat(seat_id) ON UPDATE CASCADE ON DELETE CASCADE,
   CONSTRAINT ticket_book_id_fk FOREIGN KEY (book_id) REFERENCES
booking(book_id) ON UPDATE CASCADE ON DELETE CASCADE,
   CONSTRAINT ticket_unique UNIQUE (match_id, seat_id)); COMMIT;
```

Populate Grasshopper database

```
INSERT INTO player (team, first_name, last_name, birth_date, height, weight, position,
player_num, goals, games_played, yellow_cards, red_cards, notes) VALUES
   ('First team', 'John', 'McGann', '1985-09-29', 196, 84, 'Goalkeeper', 1, 0, 27, 0, 0, 'John
McGann joined Grasshoppers team in 2005. He missed the opening games of the 2007/08 University
League of Ireland due to the shoulder injury. John returned in October 2007 in full working
order. In 2008/09 season, John performed a number of top class saves to secure the second place
for his team in the Premier League competition.'),
   ('First team', 'Brian', 'Mangan', '1985-03-16', 183, 84, 'Midfielder', 3, 6, 24, 1, 0, 'The
2009/10 season is the fourth season at Grasshoppers for Brian Mangan. Brian has a great ability
to read the game, his experience is valuable for the Grasshoppers team.'),
   ('First team', 'David', 'Newman', '1987-06-24', 179, 81.5, 'Midfielder', 4, 2, 14, 0, 0,
'David Newman first played for Grasshopper in the University League of Ireland competition in
2007/08 season. Since then, David has proven to be a fine elegant player.'),
   ('First team', 'Michael', 'Scott', '1986-11-04', 185, 83.7, 'Midfielder', 5, 6, 23, 1, 0,
'Michael Scott started his football career at Grasshoppers in the 2006/07 season. Michael is a
versatile player, who can play centre-back, right-back or in midfield.'),
   ('First team', 'Brendan', 'McKevitt', '1987-09-23', 176, 74, 'Midfielder', 8, 3, 12, 0, 0,
'Brendan McKevitt is a competitive midfielder who has been with Grasshoppers since the beginning
of the 2006/07 season.'),
   ('First team', 'Colm', 'Moloney', '1986-12-19', 184, 81, 'Midfielder', 9, 5, 21, 1, 1, 'Colm
Moloney\'s debut with Grasshoppers took place in September 2006. Colm is a talented midfielder
who can feel the game and react fast in any situation.'),
   ('First team', 'Jose', 'Soriano', '1984-05-28', 192, 85, 'Midfielder', 10, 12, 32, 2, 0, 'Jose
Soriano has played for Grasshoppers since October 2004. Jose is a fast attacking midfielder who
has contributed to many winning moments of his team.'),
   ('First team', 'Andrew', 'Collins', '1985-02-17', 177, 75, 'Midfielder', 12, 8, 27, 0,0,
'Andrew Collins joined Grasshoppers team in the beginning of the 2005/06 season. Andrew is an
athletic midfielder with quick reaction and fast pace.'),
   ('First team', 'Peter', 'Brady', '1984-11-23', 174, 73.4, 'Midfielder', 14, 5, 24, 1, 0,
'Peter Brady has been with Grasshoppers for more than five years. Peter is recognized as a player
for the big occasion, he possesses a strong ability to evaluate a situation in the heat of
battle.'),
   ('First team', 'Rory', 'Campbell', '1985-03-08', 181, 83, 'Midfielder', 16, 7, 26, 2, 0, 'Rory
Campbell joined Grasshoppers team in September 2005. Rory is an accomplished player in the middle
of the field who also likes to score goals.'),
   ('First team', 'Barry', 'Smith', '1986-07-05', 183, 84, 'Defender', 18, 4, 18, 1, 1, 'Barry
Smith made his debut at Grasshoppers in the first round of 2005/06 University League of Ireland.
Barry is an intelligent defender who has the ability to pass the ball out from the back.'),
   ('First team', 'Paul', 'Keller', '1985-04-6', 179, 77.5, 'Defender', 20, 2, 19, 2, 0, 'Paul
Keller has been on the Grasshoppers\' players list since February 2005. Paul is a fine defender
who is always happy to contribute to the team success.'),
   ('First team', 'Andrew', 'McDaid', '1984-06-18', 183, 81.4, 'Defender', 22, 3, 24, 0, 0,
'Andrew McDaid joined the team of Grasshoppers in the middle of the 2005/06 season. Andrew is an
agile defender who does not like to give a chance to the opposition to score.'),
   ('First team', 'John', 'Knowles', '1985-07-30', 178, 79.7, 'Defender', 24, 2, 18, 1, 0, 'John
Knowles is playing his third season at Grasshoppers. John is a consistent performer playing in
defence.'),
   ('First team', 'Mark', 'Smith', '1984-10-02', 189, 84, 'Defender', 26, 4, 28, 3, 0, 'Mark
Smith is into his sixth year with Grasshoppers team. Mark is a strong defender who has the
confidence to bring the ball out of defence.'),
   ('First team', 'Aidan', 'Rose', '1986-04-18', 186, 82, 'Defender', 28, 0, 12, 0, 0, 'Aidan
Rose joined Grasshoppers team in September 2005. Aidan possesses a great ability to adapt at
playing centre-back or in the heart of midfield.'),
   ('First team', 'Richard', 'Morris', '1985-09-21', 190, 84, 'Defender', 30, 1, 19, 1, 1,
'Richard Morris has been with Grasshoppers since October 2004. Richard is a tall, good in the
air, well composed defender.'),
   ('First team', 'Sean', 'Walsh', '1985-08-14', 180, 80, 'Striker', 32, 12, 23, 1, 0, 'Sean
Walsh played his first game in the Grasshoppers squad in the first round of the Premier League 0f
the 2005/06 season. Sean is a fast striker who can break up opposition defence and create
chances.'),
   ('First team', 'Colin', 'Butler', '1984-10-12', 188, 82.7, 'Striker', 34, 14, 29, 1, 1, 'Colin
Butler joined the team of Grasshoppers in the summer of 2004. Colin is a tall and quick striker
capable of spectacular goals.'),
   ('First team', 'James', 'Hunt', '1986-11-29', 184, 83, 'Striker', 36, 10, 21, 0, 0, 'James
Hunt has become an increasingly important member of Grasshoppers squad since joining the club in
the beginning of the 2005/06 season.'),
```

```
   ('First team', 'Declan', 'Burke', '1983-04-09', 186, 85, 'Striker', 38, 13, 30, 3, 0, 'Declan
Burke is one of the veterans of the Grasshoppers club, he joined the team in September of 2002.
Declan possesses a capability to push forward and score beautiful goals.'),
   ('First team', 'Patrick', 'Walsh', '1984-08-11', 178, 73.8, 'Striker', 40, 10, 27, 1, 0, 'The
2008/09 season is the fifth season at Grasshoppers for Patrick Walsh. Patrick is a good in the
air, creative striker who makes the job of the opposition defence tough.'),
   ('Reserve team', 'William', 'Downes', '1988-01-05', 173, 71, 'Midfielder', 42, 1, 4, 0, 0,
'William Downes joined the Grasshopper family in September 2007. William is a highly promising
midfielder.'),
   ('Reserve team', 'Michael', 'McGuinness', '1989-12-19', 181, 78.8, 'Defender', 44, 0, 3, 0, 0,
'Michael McGuinness is one of the youngest members of Grasshoppers club. He joined the club in
the summer of 2008, but has already performed as a talented defender.'),
   ('Reserve team', 'Paul', 'Mullin', '1988-02-18', 191, 84.5, 'Striker', 46, 3, 4, 1, 0, 'Paul
Mullin has been with Grasshoppers for two years. Paul is an energetic precise striker.'),
   ('Reserve team', 'John', 'Smith', '1988-06-21', 186, 81, 'Midfielder', 48, 1, 3, 0, 0, 'John
Smith is in his second year at the Grasshopper club. He is a fast, attacking midfielder.'),
   ('Reserve team', 'Frank', 'Thompson', '1989-02-09', 182, 78, 'Midfielder', 49, 0, 2, 0, 0,
'Frank Thompson joined the Grasshopper club in September 2008. Frank is a well composed centre-
back who can also play in the midfield.'),
   ('Reserve team', 'Liam', 'Russell', '1988-09-23', 178, 74, 'Defender', 50, 0, 4, 0, 0, 'Liam
Russell has been with Grasshoppers since the beginning of the 2007/08 season. Liam is an elegant,
technically accomplished defender.'),
   ('Reserve team', 'Peter', 'Clarke', '1989-10-30', 193, 85.6, 'Striker', 39, 2, 5, 1, 0, 'Peter
Clarke is into his second year with Grasshoppers club. He is a talented striker, a deadball
specialist.'),
   ('Reserve team', 'Ken', 'Williams', '1988-08-26', 192, 83, 'Goalkeeper', 2, 0, 3, 0, 0, 'Ken
Williams joined the team of Grasshoppers in 2007. He is a fast and brave goalkeeper who leaves a
little chance for opposition to score.');

INSERT INTO matches (match_date, opposition, competition, kick_off, venue, score_grassh,
score_oppos, analysis) VALUES
   ('2009-07-03', 'Cork Lions', 'Premier League, 1st round', '17:00', 'Cork University Stadium',
3, 1, 'Grasshoppers showed a beautiful performance at the Cork University Stadium. Within only a
few minutes into the game, James Hunt powered forward and smashed a spectacular goal into the
Lions\' net. Although Grasshoppers didn\'t have it all bright through the game, they never looked
like been beaten against Cork Lions that were trying hard to secure their place in the second
round of the Premier League. Seven minutes before the break, Colin Butler with the help of Peter
Brady added another point to the Grasshoppers\' score. Cork Lions went into the dressing room
with a zero score, two goals down and a hope for the better second half. In the tenth minute of
the second half, Lions\' striker Robert Fleming fiercely broke through the Grasshoppers defence
wall and hit a powerful shot that resulted in the first goal for the Lions. Ten minutes later,
the spirited Cork team created another dangerous situation around the Grasshoppers net but the
Grasshoppers defence had the confidence to bring the ball out of the back, to push forward and
finish the attack with the fruitful combination by Jose Soriano and Sean Walsh that ended with
the third goal for Grasshoppers. Cork Lions managed to create a few more chances for their team,
however they were not translated into more goals.'),
   ('2009-07-10', 'Dublin Rovers', 'Premier League, 1st round', '16:45', 'Grasshopper Stadium',
2, 1, 'Grasshoppers will play in the quarter-finals of the Premier League thanks to an early goal
from Declan Burke and a late one from Colin Butler. Declan opened the scoring within six minutes.
However, thirty four minutes later, the visitors exploited the gap in the Grasshoppers\' defence
formation when John Webster struck with the superb goal that equalized the game. Both teams made
sure that their fans were kept on the edge of their seats as the hit of the battle for the place
in the next round of the Premier League continued in the second half of the game. The tension
from both sides grew as the minutes ticked by. Dublin Rovers came very close to the scoring again
when Pat Corbett fired a corner kick that was passed further by Brian Walsh. Grasshoppers\'
goalkeeper John McGann, however, made an impressive save for his team. With only three minutes of
normal time left, Grasshoppers managed to finally break Rovers\' resistance when Colin Butler
score magnificent goal that took his team to the second round of the Primer League.'),
   ('2009-07-18', 'Clare Sharks', 'Premier League, 2nd round', '17:15', 'Grasshopper Stadium', 0,
2, 'The match was offset by the injuries of two defenders, Mark Smith and Paul Keller and a
striker James Hunt. It took Clare Sharks sixteen minutes to exploit the weakness in the defence
and to score against Grasshoppers. Only five minutes later, John McGann prevented a second goal
when Sharks\' striker Justin Brennan raced towards Grasshoppers\' net and hit a powerful shot
when one on one with goalkeeper who dive to his left to make an excellent save. Clare Sharks
continued to dominate and the second goal came in the thirty ninth minute. John Knowles headed
out and unchallenged Larry Smith sent a right-footed shot, the ball flu over John McGann and
bounced off the bar into the net. Grasshoppers had a bright spell in the second half when Colm
Moloney pushed forward and passed the ball to Declan Burke who made a colossal effort to score
but the ball hit against the bar. Grasshoppers created a few more chances but nothing challenging
enough to test Sharks\' goalkeeper.');
```

```
INSERT INTO matches (match_date, opposition, competition, kick_Off, venue) VALUES
   ('2009-08-05', 'Wolves', 'University League of Ireland, 1st round', '18:00', 'Grasshopper
Stadium'),
   ('2009-08-09', 'Shamrock', 'University League of Ireland, 1st round', '17:30', 'Limerick
University Stadium'),
   ('2009-08-13', 'Galway Wanderers', 'Friendly', '17:45', 'Grasshopper Stadium'),
   ('2009-08-16', 'Kerry Champions', 'Eircom Cup, 1st round', '17:15', 'Grasshopper Stadium'),
   ('2009-08-19', 'Cork Lions', 'Eircom Cup, 1st round', '18:00', 'Grasshopper Stadium');

INSERT INTO participation (player_id, match_id) VALUES
   (1, 1), (3, 1), (5, 1), (7, 1), (8, 1), (9, 1), (10, 1), (11, 1), (12, 1), (13, 1), (18, 1),
(19, 1), (20,1),
   (1, 2), (2, 2), (4, 2), (6, 2), (7, 2), (8, 2), (10, 2), (11, 2), (12, 2), (14, 2), (16, 2),
(17, 2), (19, 2), (21, 2),
   (1, 3), (2, 3), (3, 3), (4, 3), (6, 3), (8, 3), (10, 3), (12, 3), (14, 3), (15, 3), (19, 3),
(20, 3), (21, 3), (22, 3), (27, 4),
   (1, 4), (4, 4), (5, 4), (7, 4), (9, 4), (11, 4), (12, 4), (13, 4), (16, 4), (17, 4), (18, 4),
(24, 4), (26, 4),
   (1, 5), (2, 5), (3, 5), (6, 5), (8, 5), (10, 5), (12, 5), (14, 5), (15, 5), (20, 5), (22, 5),
(25, 5), (28, 5), (29,5),
   (30, 6), (3, 6), (4, 6), (5, 6), (7, 6), (9, 6), (11, 6), (13, 6), (15, 6), (19, 6), (21, 6),
(22, 6), (23, 6);

INSERT INTO section (section_name) VALUES
   ('East Stand Upper'),
   ('East Stand Lower'),
   ('West Stand Upper'),
   ('West Stand Lower'),
   ('North End Upper'),
   ('North End Lower'),
   ('River Dale Upper'),
   ('River Dale Lower');
```

An example of INSERT INTO seat VALUES command:

```
INSERT INTO seat (section_id, raw_num, seat_num) VALUES
   (1, 1, 1), (1, 1, 2), (1, 1, 3), (1, 1, 4), (1, 1, 5), (1, 1, 6), (1, 1, 7), (1, 1, 8), (1, 1,
9), (1, 1, 10), (1, 1, 11), (1, 1, 12), (1, 1, 13), (1, 1, 14), (1, 1, 15), (1, 1, 16), (1, 1,
17), (1, 1, 18), (1, 1, 19), (1, 1, 20),
   (2, 1, 1), (2, 1, 2), (2, 1, 3), (2, 1, 4), (2, 1, 5), (2, 1, 6), (2, 1, 7), (2, 1, 8), (2, 1,
9), (2, 1, 10), (2, 1, 11), (2, 1, 12), (2, 1, 13), (2, 1, 14), (2, 1, 15), (2, 1, 16), (2, 1,
17), (2, 1, 18), (2, 1, 19), (2, 1, 20),
   (3, 1, 1), (3, 1, 2), (3, 1, 3), (3, 1, 4), (3, 1, 5), (3, 1, 6), (3, 1, 7), (3, 1, 8), (3, 1,
9), (3, 1, 10), (3, 1, 11), (3, 1, 12), (3, 1, 13), (3, 1, 14), (3, 1, 15), (3, 1, 16), (3, 1,
17), (3, 1, 18), (3, 1, 19), (3, 1, 20),
   (4, 1, 1), (4, 1, 2), (4, 1, 3), (4, 1, 4), (4, 1, 5), (4, 1, 6), (4, 1, 7), (4, 1, 8), (4, 1,
9), (4, 1, 10), (4, 1, 11), (4, 1, 12), (4, 1, 13), (4, 1, 14), (4, 1, 15), (4, 1, 16), (4, 1,
17), (4, 1, 18), (4, 1, 19), (4, 1, 20),
   (5, 1, 1), (5, 1, 2), (5, 1, 3), (5, 1, 4), (5, 1, 5), (5, 1, 6), (5, 1, 7), (5, 1, 8), (5, 1,
9), (5, 1, 10), (5, 1, 11), (5, 1, 12), (5, 1, 13), (5, 1, 14), (5, 1, 15), (5, 1, 16), (5, 1,
17), (5, 1, 18), (5, 1, 19), (5, 1, 20),
   (6, 1, 1), (6, 1, 2), (6, 1, 3), (6, 1, 4), (6, 1, 5), (6, 1, 6), (6, 1, 7), (6, 1, 8), (6, 1,
9), (6, 1, 10), (6, 1, 11), (6, 1, 12), (6, 1, 13), (6, 1, 14), (6, 1, 15),
   (7, 1, 1), (7, 1, 2), (7, 1, 3), (7, 1, 4), (7, 1, 5), (7, 1, 6), (7, 1, 7), (7, 1, 8), (7, 1,
9), (7, 1, 10), (7, 1, 11), (7, 1, 12), (7, 1, 13), (7, 1, 14), (7, 1, 15), (7, 1, 16), (7, 1,
17), (7, 1, 18), (7, 1, 19), (7, 1, 20),
   (8, 1, 1), (8, 1, 2), (8, 1, 3), (8, 1, 4), (8, 1, 5), (8, 1, 6), (8, 1, 7), (8, 1, 8), (8, 1,
9), (8, 1, 10), (8, 1, 11), (8, 1, 12), (8, 1, 13), (8, 1, 14), (8, 1, 15);


INSERT INTO seatprice (match_id, section_id, price) VALUES
   (1, 1, 25), (1, 2, 20), (1, 3, 25), (1, 4, 20), (1, 5, 15), (1, 6, 10), (1, 7, 15), (1, 8,
10),
   (2, 1, 25), (2, 2, 20), (2, 3, 25), (2, 4, 20), (2, 5, 15), (2, 6, 10), (2, 7, 15), (2, 8,
10),
   (3, 1, 25), (3, 2, 20), (3, 3, 25), (3, 4, 20), (3, 5, 15), (3, 6, 10), (3, 7, 15), (3, 8,
10),
   (4, 1, 23), (4, 2, 18), (4, 3, 23), (4, 4, 18), (4, 5, 13), (4, 6, 8), (4, 7, 13), (4, 8, 8),
   (6, 1, 20), (6, 2, 15), (6, 3, 20), (6, 4, 15), (6, 5, 10), (6, 6, 5), (6, 7, 10), (6, 8, 5),
   (7, 1, 23), (7, 2, 18), (7, 3, 23), (7, 4, 18), (7, 5, 13), (7, 6, 8), (7, 7, 13), (7, 8, 8),
   (8, 1, 23), (8, 2, 18), (8, 3, 23), (8, 4, 18), (8, 5, 13), (8, 6, 8), (8, 7, 13), (8, 8, 8);
```

```
INSERT INTO user (user_id, first_name, last_name, address, city, county, country, phone, email)
VALUES
   (1,'Sandra', 'Moran', '35 Green st.', 'Kinvara', 'Galway', 'Ireland', '0868793319',
's.moran@hotmail.com'),
   (2,'Liam', 'Wilson', '37 St. Joseph Road', 'Shannon', 'Clare', 'Ireland', '0679923021',
'liam.wilson@gmail.com'),
   (3,'Martin', 'Flynn', '356/7 Hill St.', 'Limerick', 'Limerick', 'Ireland', '0872385691',
'm.flynn@eircom.net'),
   (4,'Claire', 'Fahey', '15 College Road', 'Cork', 'Cork', 'Ireland', '0853489238',
'flower23@gmail.com'),
   (5,'James', 'Curran', '29 Stephen St.', 'Lucan', 'Dublin', 'Ireland', '08734511289',
'james.c@live.com'),
   (6,'test','test','test','test','test','test','test','test');

INSERT INTO account (username, password, user_id) VALUES
   ('s.moran@hotmail.com', 'SnizhoK61kD', 1),
   ('liam.wilson@gmail.com', 'Persyk917Kr3', 2),
   ('m.flynn@eircom.net', 'osinniyVit496', 3),
   ('flower23@gmail.com', 'Mor573Bal4T', 4),
   ('james.c@live.com', 'VovK7siryI16L', 5),
   ('test','pass',6);

INSERT INTO authorities (username, authority) VALUES
   ('s.moran@hotmail.com', 'ROLE_USER'),
   ('liam.wilson@gmail.com', 'ROLE_USER'),
   ('m.flynn@eircom.net', 'ROLE_USER'),
   ('flower23@gmail.com', 'ROLE_USER'),
   ('james.c@live.com', 'ROLE_USER'),
   ('test','ROLE_USER');

INSERT INTO credit_card (fname_on_card, lname_on_card, type, number, cvc, expiry_month,
expiry_year, user_id) VALUES
   ('Sandra', 'Moran', 'Visa', '1267456789356781', '357', 10, '12', 1),
   ('Liam', 'Wilson', 'MasterCard', '9814267902238495', '582', 8, '11', 2),
   ('Martin', 'Flynn', 'MasterCard', '3567113092842833', '763', 4, '13', 3),
   ('Claire', 'Fahey', 'Visa', '2785448919238260', '874', 12, '10', 4),
   ('James', 'Curran', 'Visa', '0826771529463857', '692', 3, '10', 5);

INSERT INTO booking (pay_total, paid_date, pay_method, user_id) VALUES
   (75, '2009-07-10', 'Credit Card', 1),
   (50, '2009-07-11', 'Cash', 2),
   (40, '2009-07-12', 'Credit Card', 3),
   (100, '2009-07-14', 'Credit Card', 4),
   (45, '2009-07-14', 'Cash', 5);

INSERT INTO ticket (match_id, seat_id) VALUES
   (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (2, 6), (2, 7), (2, 8), (2,9), (2, 10), (2, 11), (2,
12), (2, 13), (2, 14), (2, 15);

INSERT INTO ticket (match_id, seat_id, available, book_id) VALUES
   (2, 36, 0, 1), (2, 37, 0, 1), (2, 38, 0, 1), (2, 42, 0, 2), (2, 43, 0, 2), (2, 210, 0, 3), (2,
211, 0, 3), (2, 423, 0, 4), (2, 424, 0, 4), (2, 425, 0, 4), (2, 426, 0, 4), (2, 427, 0, 4), (2,
915, 0, 5), (2, 916, 0, 5), (2, 917, 0, 5);

INSERT INTO ticket (match_id, seat_id) VALUES (4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (4, 6), (4,
7), (4, 8), (4, 9), (4, 10),
   (4, 100), (4, 101), (4, 102), (4, 103), (4, 104), (4, 105), (4, 106), (4, 107), (4, 108), (4,
109), (4, 110),
   (4, 180), (4, 181), (4, 182), (4, 183), (4, 184), (4, 185), (4, 186), (4, 187), (4, 188), (4,
189), (4, 190),
   (4, 260), (4, 261), (4, 262), (4, 263), (4, 264), (4, 265), (4, 266), (4, 267), (4, 268), (4,
269), (4, 270),
   (4, 345), (4, 346), (4, 347), (4, 348), (4, 349), (4, 350), (4, 351), (4, 352), (4, 353), (4,
354), (4, 355),
   (4, 400), (4, 401), (4, 402), (4, 403), (4, 404), (4, 405), (4, 406), (4, 407), (4, 408), (4,
409), (4, 410),
   (4, 450), (4, 451), (4, 452), (4, 453), (4, 454), (4, 455), (4, 456), (4, 457), (4, 458), (4,
459), (4, 460),
   (4, 528), (4, 529), (4, 530), (4, 531), (4, 532), (4, 533), (4, 534), (4, 535), (4, 536), (4,
537), (4, 538);
COMMIT;
```

Appendix B

POJOs Identified for the Grasshopper Application

Account.java class

```
package com.grasshopper.orm.hibernate.entity;

import java.util.HashSet;
import java.util.Set;

public class Account implements java.io.Serializable {
    private static final long serialVersionUID = -8899220773768056559L;

    private Integer accountId;
    private User user;
    private String username;
    private boolean enabled;
    private String password;
    private Set<Authorities> authorities = new HashSet<Authorities>(0);

    public Account() {
    }

    public Account(User user, String username, boolean enabled, String
password) {
      this.user = user;
      this.username = username;
      this.enabled = enabled;
      this.password = password;
    }

    public Account(User user, String username, boolean enabled,
          String password, Set<Authorities> authoritieses) {
      this.user = user;
      this.username = username;
      this.enabled = enabled;
      this.password = password;
      this.authorities = authoritieses;
    }

    public Integer getAccountId() {
      return this.accountId;
    }

    public void setAccountId(Integer accountId) {
      this.accountId = accountId;
    }

    public User getUser() {
      return this.user;
    }

    public void setUser(User user) {
      this.user = user;
```

```
    }

    public String getUsername() {
      return this.username;
    }

    public void setUsername(String username) {
      this.username = username;
    }

    public boolean isEnabled() {
      return this.enabled;
    }

    public void setEnabled(boolean enabled) {
      this.enabled = enabled;
    }

    public String getPassword() {
      return this.password;
    }

    public void setPassword(String password) {
      this.password = password;
    }

    public Set<Authorities> getAuthorities() {
      return this.authorities;
    }

    public void setAuthorities(Set<Authorities> authorities) {
      this.authorities = authorities;
    }
}
```

Authorities.java class

```
package com.grasshopper.orm.hibernate.entity;

public class Authorities implements java.io.Serializable {
    private static final long serialVersionUID = -5388306759485097127L;

    private AuthoritiesId id;
    private Account account;

    public Authorities() {
    }

    public Authorities(AuthoritiesId id, Account account) {
      this.id = id;
      this.account = account;
    }

    public AuthoritiesId getId() {
      return this.id;
    }
```

```
    public void setId(AuthoritiesId id) {
      this.id = id;
    }

    public Account getAccount() {
      return this.account;
    }

    public void setAccount(Account account) {
      this.account = account;
    }

    @Override
    public int hashCode() {
      final int prime = 31;
      int result = 1;
      result = prime * result + ((account == null) ? 0 : account.hashCode());
      result = prime * result + ((id == null) ? 0 : id.hashCode());
      return result;
    }

    @Override
    public boolean equals(Object obj) {
      if (this == obj)
          return true;
      if (obj == null)
          return false;
      if (getClass() != obj.getClass())
          return false;
      Authorities other = (Authorities) obj;
      if (account == null) {
          if (other.account != null)
            return false;
      } else if (!account.equals(other.account))
          return false;
      if (id == null) {
          if (other.id != null)
            return false;
      } else if (!id.equals(other.id))
          return false;
      return true;
    }
}
```

AuthoritiesId.java class

```
package com.grasshopper.orm.hibernate.entity;

public class AuthoritiesId implements java.io.Serializable {
    private static final long serialVersionUID = -5814635226588250066L;

    private String username;
    private String authority;

    public AuthoritiesId() {
    }
```

```
    public AuthoritiesId(String username, String authority) {
      this.username = username;
      this.authority = authority;
    }

    public String getUsername() {
      return this.username;
    }

    public void setUsername(String username) {
      this.username = username;
    }

    public String getAuthority() {
      return this.authority;
    }

    public void setAuthority(String authority) {
      this.authority = authority;
    }

    public boolean equals(Object other) {
      if ((this == other))
          return true;
      if ((other == null))
          return false;
      if (!(other instanceof AuthoritiesId))
          return false;
      AuthoritiesId castOther = (AuthoritiesId) other;

      return ((this.getUsername() == castOther.getUsername()) || (this
            .getUsername() != null
            && castOther.getUsername() != null && this.getUsername()
            .equals(castOther.getUsername())))
            && ((this.getAuthority() == castOther.getAuthority()) || (this
                  .getAuthority() != null
                  && castOther.getAuthority() != null && this
                  .getAuthority().equals(castOther.getAuthority())));
    }

    public int hashCode() {
      int result = 17;

      result = 37 * result
            + (getUsername() == null ? 0 : this.getUsername().hashCode());
      result = 37 * result
            + (getAuthority() == null ? 0 : this.getAuthority().hashCode());
      return result;
    }

}
```

Booking.java class

```
package com.grasshopper.orm.hibernate.entity;

import java.math.BigDecimal;
import java.util.Date;
import java.util.HashSet;
import java.util.Set;

public class Booking implements java.io.Serializable {
    private static final long serialVersionUID = -296434934172913831L;

    private Integer bookId;
    private User user;
    private BigDecimal payTotal;
    private Date paidDate;
    private String payMethod;
    private Set<Ticket> tickets = new HashSet<Ticket>(0);

    public Booking() {
    }

    public Booking(User user, BigDecimal payTotal) {
      this.user = user;
      this.payTotal = payTotal;
    }

    public Booking(User user, BigDecimal payTotal, Date paidDate,
          String payMethod, Set<Ticket> tickets) {
      this.user = user;
      this.payTotal = payTotal;
      this.paidDate = paidDate;
      this.payMethod = payMethod;
      this.tickets = tickets;
    }

    public Integer getBookId() {
      return this.bookId;
    }

    public void setBookId(Integer bookId) {
      this.bookId = bookId;
    }

    public User getUser() {
      return this.user;
    }

    public void setUser(User user) {
      this.user = user;
    }

    public BigDecimal getPayTotal() {
      return this.payTotal;
    }
```

```
    public void setPayTotal(BigDecimal payTotal) {
      this.payTotal = payTotal;
    }

    public Date getPaidDate() {
      return this.paidDate;
    }

    public void setPaidDate(Date paidDate) {
      this.paidDate = paidDate;
    }

    public String getPayMethod() {
      return this.payMethod;
    }

    public void setPayMethod(String payMethod) {
      this.payMethod = payMethod;
    }

    public Set<Ticket> getTickets() {
      return this.tickets;
    }

    public void setTickets(Set<Ticket> tickets) {
      this.tickets = tickets;
    }

}
```

CreditCard.java class

```
package com.grasshopper.orm.hibernate.entity;

import java.util.Date;

public class CreditCard implements java.io.Serializable {
    private static final long serialVersionUID = 1997259172636246802L;

    private Integer cardId;
    private User user;
    private String fnameOnCard;
    private String lnameOnCard;
    private String type;
    private String number;
    private String cvc;
    private byte expiryMonth;
    private Date expiryYear;

    public CreditCard() {
    }

    public CreditCard(User user, String fnameOnCard, String lnameOnCard,
          String type, String number, String cvc, byte expiryMonth,
          Date expiryYear) {
      this.user = user;
```

```
    this.fnameOnCard = fnameOnCard;
    this.lnameOnCard = lnameOnCard;
    this.type = type;
    this.number = number;
    this.cvc = cvc;
    this.expiryMonth = expiryMonth;
    this.expiryYear = expiryYear;
  }

  public Integer getCardId() {
    return this.cardId;
  }

  public void setCardId(Integer cardId) {
    this.cardId = cardId;
  }

  public User getUser() {
    return this.user;
  }

  public void setUser(User user) {
    this.user = user;
  }

  public String getFnameOnCard() {
    return this.fnameOnCard;
  }

  public void setFnameOnCard(String fnameOnCard) {
    this.fnameOnCard = fnameOnCard;
  }

  public String getLnameOnCard() {
    return this.lnameOnCard;
  }

  public void setLnameOnCard(String lnameOnCard) {
    this.lnameOnCard = lnameOnCard;
  }

  public String getType() {
    return this.type;
  }

  public void setType(String type) {
    this.type = type;
  }

  public String getNumber() {
    return this.number;
  }

  public void setNumber(String number) {
    this.number = number;
  }
```

```
    public String getCvc() {
      return this.cvc;
    }

    public void setCvc(String cvc) {
      this.cvc = cvc;
    }

    public byte getExpiryMonth() {
      return this.expiryMonth;
    }

    public void setExpiryMonth(byte expiryMonth) {
      this.expiryMonth = expiryMonth;
    }

    public Date getExpiryYear() {
      return this.expiryYear;
    }

    public void setExpiryYear(Date expiryYear) {
      this.expiryYear = expiryYear;
    }

}
```

Matches.java class

```
package com.grasshopper.orm.hibernate.entity;

import java.util.Date;
import java.util.HashSet;
import java.util.Set;

public class Matches implements java.io.Serializable {
    private static final long serialVersionUID = 2175906684544117259L;

    private Integer matchId;
    private Date matchDate;
    private String opposition;
    private String competition;
    private Date kickOff;
    private String venue;
    private Integer scoreGrassh;
    private Integer scoreOppos;
    private String analysis;
    private Set<Participation> participations = new
HashSet<Participation>(0);
    private Set<Seatprice> seatprices = new HashSet<Seatprice>(0);
    private Set<Ticket> tickets = new HashSet<Ticket>(0);

    public Matches() {
    }

    public Matches(Date matchDate, String opposition, String competition,
          Date kickOff, String venue) {
```

```
  this.matchDate = matchDate;
  this.opposition = opposition;
  this.competition = competition;
  this.kickOff = kickOff;
  this.venue = venue;
}

public Matches(Date matchDate, String opposition, String competition,
      Date kickOff, String venue, Integer scoreGrassh,
      Integer scoreOppos, String analysis,
      Set<Participation> participations, Set<Seatprice> seatprices,
      Set<Ticket> tickets) {
  this.matchDate = matchDate;
  this.opposition = opposition;
  this.competition = competition;
  this.kickOff = kickOff;
  this.venue = venue;
  this.scoreGrassh = scoreGrassh;
  this.scoreOppos = scoreOppos;
  this.analysis = analysis;
  this.participations = participations;
  this.seatprices = seatprices;
  this.tickets = tickets;
}

public Integer getMatchId() {
  return this.matchId;
}

public void setMatchId(Integer matchId) {
  this.matchId = matchId;
}

public Date getMatchDate() {
  return this.matchDate;
}

public void setMatchDate(Date matchDate) {
  this.matchDate = matchDate;
}

public String getOpposition() {
  return this.opposition;
}

public void setOpposition(String opposition) {
  this.opposition = opposition;
}

public String getCompetition() {
  return this.competition;
}

public void setCompetition(String competition) {
  this.competition = competition;
}
```

```
public Date getKickOff() {
  return this.kickOff;
}

public void setKickOff(Date kickOff) {
  this.kickOff = kickOff;
}

public String getVenue() {
  return this.venue;
}

public void setVenue(String venue) {
  this.venue = venue;
}

public Integer getScoreGrassh() {
  return this.scoreGrassh;
}

public void setScoreGrassh(Integer scoreGrassh) {
  this.scoreGrassh = scoreGrassh;
}

public Integer getScoreOppos() {
  return this.scoreOppos;
}

public void setScoreOppos(Integer scoreOppos) {
  this.scoreOppos = scoreOppos;
}

public String getAnalysis() {
  return this.analysis;
}

public void setAnalysis(String analysis) {
  this.analysis = analysis;
}

public Set<Participation> getParticipations() {
  return this.participations;
}

public void setParticipations(Set<Participation> participations) {
  this.participations = participations;
}

public Set<Seatprice> getSeatprices() {
  return this.seatprices;
}

public void setSeatprices(Set<Seatprice> seatprices) {
  this.seatprices = seatprices;
}

public Set<Ticket> getTickets() {
```

```
      return this.tickets;
    }

    public void setTickets(Set<Ticket> tickets) {
      this.tickets = tickets;
    }

}
```

Participation.java class

```
package com.grasshopper.orm.hibernate.entity;

public class Participation implements java.io.Serializable {
    private static final long serialVersionUID = -906474192272059687L;

    private Integer partId;
    private Matches matches;
    private Player player;

    public Participation() {
    }

    public Participation(Matches matches, Player player) {
      this.matches = matches;
      this.player = player;
    }

    public Integer getPartId() {
      return this.partId;
    }

    public void setPartId(Integer partId) {
      this.partId = partId;
    }

    public Matches getMatches() {
      return this.matches;
    }

    public void setMatches(Matches matches) {
      this.matches = matches;
    }

    public Player getPlayer() {
      return this.player;
    }

    public void setPlayer(Player player) {
      this.player = player;
    }

}
```

Player.java class

```
package com.grasshopper.orm.hibernate.entity;

import java.math.BigDecimal;
import java.util.Date;
import java.util.HashSet;
import java.util.Set;

public class Player implements java.io.Serializable {
    private static final long serialVersionUID = -255523085789394442L;

    private Integer playerId;
    private String team;
    private String firstName;
    private String lastName;
    private Date birthDate;
    private short height;
    private BigDecimal weight;
    private String position;
    private short playerNum;
    private short goals;
    private short gamesPlayed;
    private short yellowCards;
    private short redCards;
    private String notes;
    private Set<Participation> participations = new
HashSet<Participation>(0);

    public Player() {
    }

    public Player(String team, String firstName, String lastName,
          Date birthDate, short height, BigDecimal weight, String position,
          short playerNum, short goals, short gamesPlayed, short yellowCards,
          short redCards, String notes) {
      this.team = team;
      this.firstName = firstName;
      this.lastName = lastName;
      this.birthDate = birthDate;
      this.height = height;
      this.weight = weight;
      this.position = position;
      this.playerNum = playerNum;
      this.goals = goals;
      this.gamesPlayed = gamesPlayed;
      this.yellowCards = yellowCards;
      this.redCards = redCards;
      this.notes = notes;
    }

    public Player(String team, String firstName, String lastName,
          Date birthDate, short height, BigDecimal weight, String position,
          short playerNum, short goals, short gamesPlayed, short yellowCards,
          short redCards, String notes, Set<Participation> participations) {
      this.team = team;
```

```
    this.firstName = firstName;
    this.lastName = lastName;
    this.birthDate = birthDate;
    this.height = height;
    this.weight = weight;
    this.position = position;
    this.playerNum = playerNum;
    this.goals = goals;
    this.gamesPlayed = gamesPlayed;
    this.yellowCards = yellowCards;
    this.redCards = redCards;
    this.notes = notes;
    this.participations = participations;
  }

  public Integer getPlayerId() {
    return this.playerId;
  }

  public void setPlayerId(Integer playerId) {
    this.playerId = playerId;
  }

  public String getTeam() {
    return this.team;
  }

  public void setTeam(String team) {
    this.team = team;
  }

  public String getFirstName() {
    return this.firstName;
  }

  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  public String getLastName() {
    return this.lastName;
  }

  public void setLastName(String lastName) {
    this.lastName = lastName;
  }

  public Date getBirthDate() {
    return this.birthDate;
  }

  public void setBirthDate(Date birthDate) {
    this.birthDate = birthDate;
  }

  public short getHeight() {
    return this.height;
```

```
}

public void setHeight(short height) {
  this.height = height;
}

public BigDecimal getWeight() {
  return this.weight;
}

public void setWeight(BigDecimal weight) {
  this.weight = weight;
}

public String getPosition() {
  return this.position;
}

public void setPosition(String position) {
  this.position = position;
}

public short getPlayerNum() {
  return this.playerNum;
}

public void setPlayerNum(short playerNum) {
  this.playerNum = playerNum;
}

public short getGoals() {
  return this.goals;
}

public void setGoals(short goals) {
  this.goals = goals;
}

public short getGamesPlayed() {
  return this.gamesPlayed;
}

public void setGamesPlayed(short gamesPlayed) {
  this.gamesPlayed = gamesPlayed;
}

public short getYellowCards() {
  return this.yellowCards;
}

public void setYellowCards(short yellowCards) {
  this.yellowCards = yellowCards;
}

public short getRedCards() {
  return this.redCards;
}
```

```
    public void setRedCards(short redCards) {
      this.redCards = redCards;
    }

    public String getNotes() {
      return this.notes;
    }

    public void setNotes(String notes) {
      this.notes = notes;
    }

    public Set<Participation> getParticipations() {
      return this.participations;
    }

    public void setParticipations(Set<Participation> participations) {
      this.participations = participations;
    }

}
```

Seat.java class

```
package com.grasshopper.orm.hibernate.entity;

import java.util.HashSet;
import java.util.Set;

public class Seat implements java.io.Serializable {
    private static final long serialVersionUID = 4787907333052873520L;

    private Integer seatId;
    private Section section;
    private int rawNum;
    private int seatNum;
    private Set<Seatprice> seatprices = new HashSet<Seatprice>(0);
    private Set<Ticket> tickets = new HashSet<Ticket>(0);

    public Seat() {
    }

    public Seat(Section section, int rawNum, int seatNum) {
      this.section = section;
      this.rawNum = rawNum;
      this.seatNum = seatNum;
    }

    public Seat(Section section, int rawNum, int seatNum,
          Set<Seatprice> seatprices, Set<Ticket> tickets) {
      this.section = section;
      this.rawNum = rawNum;
      this.seatNum = seatNum;
      this.seatprices = seatprices;
      this.tickets = tickets;
    }
```

```
    public Integer getSeatId() {
      return this.seatId;
    }

    public void setSeatId(Integer seatId) {
      this.seatId = seatId;
    }

    public Section getSection() {
      return this.section;
    }

    public void setSection(Section section) {
      this.section = section;
    }

    public int getRawNum() {
      return this.rawNum;
    }

    public void setRawNum(int rawNum) {
      this.rawNum = rawNum;
    }

    public int getSeatNum() {
      return this.seatNum;
    }

    public void setSeatNum(int seatNum) {
      this.seatNum = seatNum;
    }

    public Set<Seatprice> getSeatprices() {
      return this.seatprices;
    }

    public void setSeatprices(Set<Seatprice> seatprices) {
      this.seatprices = seatprices;
    }

    public Set<Ticket> getTickets() {
      return this.tickets;
    }

    public void setTickets(Set<Ticket> tickets) {
      this.tickets = tickets;
    }

}
```

Seatprice.java class

```
package com.grasshopper.orm.hibernate.entity;

import java.math.BigDecimal;
```

```
public class Seatprice implements java.io.Serializable {
    private static final long serialVersionUID = -8360795903199178191L;

    private Integer seatpriceId;
    private Matches matches;
    private Seat seat;
    private BigDecimal price;

    public Seatprice() {
    }

    public Seatprice(Matches matches, Seat seat, BigDecimal price) {
      this.matches = matches;
      this.seat = seat;
      this.price = price;
    }

    public Integer getSeatpriceId() {
      return this.seatpriceId;
    }

    public void setSeatpriceId(Integer seatpriceId) {
      this.seatpriceId = seatpriceId;
    }

    public Matches getMatches() {
      return this.matches;
    }

    public void setMatches(Matches matches) {
      this.matches = matches;
    }

    public Seat getSeat() {
      return this.seat;
    }

    public void setSeat(Seat seat) {
      this.seat = seat;
    }

    public BigDecimal getPrice() {
      return this.price;
    }

    public void setPrice(BigDecimal price) {
      this.price = price;
    }

}
```

Section.java class

```
package com.grasshopper.orm.hibernate.entity;

import java.util.HashSet;
```

```
import java.util.Set;

public class Section implements java.io.Serializable {
    private static final long serialVersionUID = -3299275380514574351L;

    private Integer sectionId;
    private String sectionName;
    private Set<Seat> seats = new HashSet<Seat>(0);

    public Section() {
    }

    public Section(String sectionName) {
      this.sectionName = sectionName;
    }

    public Section(String sectionName, Set<Seat> seats) {
      this.sectionName = sectionName;
      this.seats = seats;
    }

    public Integer getSectionId() {
      return this.sectionId;
    }

    public void setSectionId(Integer sectionId) {
      this.sectionId = sectionId;
    }

    public String getSectionName() {
      return this.sectionName;
    }

    public void setSectionName(String sectionName) {
      this.sectionName = sectionName;
    }

    public Set<Seat> getSeats() {
      return this.seats;
    }

    public void setSeats(Set<Seat> seats) {
      this.seats = seats;
    }

}
```

Ticket.java class

```
package com.grasshopper.orm.hibernate.entity;

public class Ticket implements java.io.Serializable {
    private static final long serialVersionUID = -2023726346895626711L;

    private Integer ticketId;
    private Booking booking;
```

```
    private Seat seat;
    private Matches matches;
    private boolean available;

    public Ticket() {
    }

    public Ticket(Seat seat, Matches matches, boolean available) {
      this.seat = seat;
      this.matches = matches;
      this.available = available;
    }

    public Ticket(Booking booking, Seat seat, Matches matches, boolean
available) {
      this.booking = booking;
      this.seat = seat;
      this.matches = matches;
      this.available = available;
    }

    public Integer getTicketId() {
      return this.ticketId;
    }

    public void setTicketId(Integer ticketId) {
      this.ticketId = ticketId;
    }

    public Booking getBooking() {
      return this.booking;
    }

    public void setBooking(Booking booking) {
      this.booking = booking;
    }

    public Seat getSeat() {
      return this.seat;
    }

    public void setSeat(Seat seat) {
      this.seat = seat;
    }

    public Matches getMatches() {
      return this.matches;
    }

    public void setMatches(Matches matches) {
      this.matches = matches;
    }

    public boolean isAvailable() {
      return this.available;
    }
```

```
    public void setAvailable(boolean available) {
      this.available = available;
    }

}
```

User.java class

```
package com.grasshopper.orm.hibernate.entity;

import java.util.HashSet;
import java.util.Set;

import org.hibernate.validator.Email;
import org.hibernate.validator.NotEmpty;

public class User implements java.io.Serializable {
    private static final long serialVersionUID = 4292517370671262363L;

    private Integer userId;
    @NotEmpty
    private String firstName;
    @NotEmpty
    private String lastName;
    @NotEmpty
    private String address;
    @NotEmpty
    private String city;
    @NotEmpty
    private String county;
    @NotEmpty
    private String country;
    @NotEmpty
    private String phone;
    @Email
    @NotEmpty
    private String email;
    private Set<CreditCard> creditCards = new HashSet<CreditCard>(0);
    private Set<Booking> bookings = new HashSet<Booking>(0);
    private Set<Account> accounts = new HashSet<Account>(0);

    public User() {
    }

    public User(String firstName, String lastName, String address, String
city,
          String county, String country, String phone, String email) {
      this.firstName = firstName;
      this.lastName = lastName;
      this.address = address;
      this.city = city;
      this.county = county;
      this.country = country;
      this.phone = phone;
      this.email = email;
    }
```

```
    public User(String firstName, String lastName, String address, String
city,
          String county, String country, String phone, String email,
          Set<CreditCard> creditCards, Set<Booking> bookings,
          Set<Account> accounts) {
      this.firstName = firstName;
      this.lastName = lastName;
      this.address = address;
      this.city = city;
      this.county = county;
      this.country = country;
      this.phone = phone;
      this.email = email;
      this.creditCards = creditCards;
      this.bookings = bookings;
      this.accounts = accounts;
    }

    public Integer getUserId() {
      return this.userId;
    }

    public void setUserId(Integer userId) {
      this.userId = userId;
    }

    public String getFirstName() {
      return this.firstName;
    }

    public void setFirstName(String firstName) {
      this.firstName = firstName;
    }

    public String getLastName() {
      return this.lastName;
    }

    public void setLastName(String lastName) {
      this.lastName = lastName;
    }

    public String getAddress() {
      return this.address;
    }

    public void setAddress(String address) {
      this.address = address;
    }

    public String getCity() {
      return this.city;
    }

    public void setCity(String city) {
      this.city = city;
    }
```

```
    public String getCounty() {
      return this.county;
    }

    public void setCounty(String county) {
      this.county = county;
    }

    public String getCountry() {
      return this.country;
    }

    public void setCountry(String country) {
      this.country = country;
    }

    public String getPhone() {
      return this.phone;
    }

    public void setPhone(String phone) {
      this.phone = phone;
    }

    public String getEmail() {
      return this.email;
    }

    public void setEmail(String email) {
      this.email = email;
    }

    public Set<CreditCard> getCreditCards() {
      return this.creditCards;
    }

    public void setCreditCards(Set<CreditCard> creditCards) {
      this.creditCards = creditCards;
    }

    public Set<Booking> getBookings() {
      return this.bookings;
    }

    public void setBookings(Set<Booking> bookings) {
      this.bookings = bookings;
    }

    public Set<Account> getAccounts() {
      return this.accounts;
    }

    public void setAccounts(Set<Account> accounts) {
      this.accounts = accounts;
    }
}
```

Appendix C

Data Access Objects

AbstractDao.java

```
package com.grasshopper.dao;

public interface AbstractDao {

}
```

GenericDao.java

```
package com.grasshopper.dao;

import java.io.Serializable;
import java.util.List;

import com.grasshopper.dao.AbstractDao;

public interface GenericDao<T extends Object, PK extends Serializable>
extends
      AbstractDao {
    /**
     * Query one entity by primary key.
     *
     * @return list of all entities presented in DB.
     */
    T findById(PK id, boolean lock);

    /**
     * Query all entities.
     *
     * @return list of all entities presented in DB.
     */
    List<T> findAll();

    /**
     * Create new entity.
     *
     * @param entity
     *            entity to create.
     * @return primary key of created entity
     */
    void save(T object);

    /**
     * Update entity.
     *
     * @param entity
     *            entity to update.
     * @return Ne entity
     */
```

```
    T update(T object);

    /**
     * Delete entity.
     *
     * @param entity
     *            entity to delete.
     */
    void delete(T object);

    /**
     * Delete all entities.
     */
    void deleteAll();

    /**
     * Flush the session.
     */
    void flush();

    /**
     * Clear the session
     */
    void clear();

    /**
     * Evict object from the session.
     */
    void evict(T object);

    /**
     * Persist new object or update already existent in DB.
     * @param entity entity to be saved or updated
     * @return object which has been persisted to DB.
     */
    T saveOrUpdate(T entity);
}
```

AccountDao.java

```
package com.grasshopper.dao;

import java.util.Set;

import com.grasshopper.entity.Account;
import com.grasshopper.entity.User;

public interface AccountDao extends GenericDao<Account, Integer> {
    boolean checkUserNameAvailable(String username);

    Set<Account> findAccountByUserExample(User user);

    Account findAccount(String username, String password);

}
```

Matches.Dao.java

```java
package com.grasshopper.dao;

import java.util.List;

import com.grasshopper.entity.Matches;
import com.grasshopper.entity.Ticket;

public interface MatchesDao extends GenericDao<Matches, Integer> {
    int getSoldTicketsCount(Matches match);

    List<Matches> getUpcomingMatches(int matchesAmountLimit);

    List<Ticket> findAvailableTickets(Integer matchId);
}
```

SeatDao.java

```java
package com.grasshopper.dao;

import com.grasshopper.entity.Seat;

public interface SeatDao extends GenericDao<Seat, Integer> {
    Integer getOverallSeatCount();
}
```

Appendix D

Object-Relational Mapping with Hibernate

Hibernate XML Mapping Files

Account.hbm.xml file

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
      <class name="com.grasshopper.entity.Account" table="account">
            <id name="accountId" type="java.lang.Integer">
                  <column name="account_id" />
                  <generator class="identity" />
            </id>
            <many-to-one name="user"
                  class="com.grasshopper.entity.User" fetch="select"
lazy="false"
                  cascade="all">
                  <column name="user_id" not-null="true" />
            </many-to-one>
            <property name="username" type="string">
                  <column name="username" length="25" not-null="true"
unique="true" />
            </property>
            <property name="enabled" type="boolean" >
                  <column name="enabled" not-null="true" default="true"/>
            </property>
            <property name="password" type="string">
                  <column name="password" length="15" not-null="true" />
            </property>
            <set name="authorities" inverse="true" lazy="false"
table="authorities"
                  fetch="select" cascade="all">
                  <key>
                        <column name="username" not-null="true" />
                  </key>
                  <one-to-many class="com.grasshopper.entity.Authorities" />
            </set>
      </class>
</hibernate-mapping>
```

Authorities.hbm.xml file

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
```

```
<hibernate-mapping>
    <class name="com.grasshopper.entity.Authorities" table="authorities">
        <composite-id name="id" class="com.grasshopper.entity.AuthoritiesId">
            <key-property name="username" type="string">
                <column name="username" />
            </key-property>
            <key-property name="authority" type="string">
                <column name="authority" length="25" />
            </key-property>
        </composite-id>
        <many-to-one name="account" class="com.grasshopper.entity.Account"
update="false" insert="false" fetch="select"
        cascade="all">
            <column name="username" not-null="true" />
        </many-to-one>
    </class>
</hibernate-mapping>
```

Booking.hbm.xml file

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.grasshopper.entity.Booking" table="booking">
        <id name="bookId" type="java.lang.Integer">
            <column name="book_id" />
            <generator class="identity" />
        </id>
        <many-to-one name="user" class="com.grasshopper.entity.User"
fetch="select">
            <column name="user_id" not-null="true" />
        </many-to-one>
        <property name="payTotal" type="big_decimal">
            <column name="pay_total" precision="5" not-null="true" />
        </property>
        <property name="paidDate" type="date">
            <column name="paid_date" length="10" />
        </property>
        <property name="payMethod" type="string">
            <column name="pay_method" length="11" />
        </property>
        <set name="tickets" inverse="true" lazy="false" table="ticket"
fetch="select" cascade="all">
            <key>
                <column name="book_id" />
            </key>
            <one-to-many class="com.grasshopper.entity.Ticket" />
        </set>
    </class>
</hibernate-mapping>
```

CreditCard.hbm.xml file

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.grasshopper.entity.CreditCard" table="credit_card">
        <id name="cardId" type="java.lang.Integer">
            <column name="card_id" />
            <generator class="identity" />
        </id>
        <many-to-one name="user" class="com.grasshopper.entity.User"
fetch="select">
            <column name="user_id" not-null="true" />
        </many-to-one>
        <property name="fnameOnCard" type="string">
            <column name="fname_on_card" length="25" not-null="true" />
        </property>
        <property name="lnameOnCard" type="string">
            <column name="lname_on_card" length="25" not-null="true" />
        </property>
        <property name="type" type="string">
            <column name="type" length="15" not-null="true" />
        </property>
        <property name="number" type="string">
            <column name="number" length="16" not-null="true" />
        </property>
        <property name="cvc" type="string">
            <column name="cvc" length="3" not-null="true" />
        </property>
        <property name="expiryMonth" type="byte">
            <column name="expiry_month" not-null="true" />
        </property>
        <property name="expiryYear" type="date">
            <column name="expiry_year" length="0" not-null="true" />
        </property>
    </class>
</hibernate-mapping>
```

Matches.hbm.xml file

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.grasshopper.entity.Matches" table="matches">
        <id name="matchId" type="java.lang.Integer">
            <column name="match_id" />
            <generator class="identity" />
        </id>
        <property name="matchDate" type="date">
```

```
            <column name="match_date" length="10" not-null="true" />
        </property>
        <property name="opposition" type="string">
            <column name="opposition" length="45" not-null="true" />
        </property>
        <property name="competition" type="string">
            <column name="competition" length="50" not-null="true" />
        </property>
        <property name="kickOff" type="time">
            <column name="kick_off" length="8" not-null="true" />
        </property>
        <property name="venue" type="string">
            <column name="venue" length="50" not-null="true" />
        </property>
        <property name="scoreGrassh" type="java.lang.Integer">
            <column name="score_grassh" />
        </property>
        <property name="scoreOppos" type="java.lang.Integer">
            <column name="score_oppos" />
        </property>
        <property name="analysis" type="string">
            <column name="analysis" length="5000" />
        </property>
        <set name="participations" inverse="true" lazy="false"
table="participation" fetch="select">
            <key>
                <column name="match_id" not-null="true" />
            </key>
            <one-to-many class="com.grasshopper.entity.Participation" />
        </set>
        <set name="seatprices" inverse="true" lazy="true" table="seatprice"
fetch="select">
            <key>
                <column name="match_id" not-null="true" />
            </key>
            <one-to-many class="com.grasshopper.entity.Seatprice" />
        </set>
        <set name="tickets" inverse="true" lazy="true" table="ticket"
fetch="select">
            <key>
                <column name="match_id" not-null="true" />
            </key>
            <one-to-many class="com.grasshopper.entity.Ticket" />
        </set>
    </class>
</hibernate-mapping>
```

Participation.hbm.xml file

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.grasshopper.entity.Participation" table="participation">
        <id name="partId" type="java.lang.Integer">
```

```
            <column name="part_id" />
            <generator class="identity" />
        </id>
        <many-to-one name="matches" class="com.grasshopper.entity.Matches"
fetch="select">
            <column name="match_id" not-null="true" />
        </many-to-one>
        <many-to-one name="player" class="com.grasshopper.entity.Player"
fetch="select">
            <column name="player_id" not-null="true" />
        </many-to-one>
    </class>
</hibernate-mapping>
```

Player.hbm.xml file

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.grasshopper.entity.Player" table="player">
        <id name="playerId" type="java.lang.Integer">
            <column name="player_id" />
            <generator class="identity" />
        </id>
        <property name="team" type="string">
            <column name="team" length="12" not-null="true" />
        </property>
        <property name="firstName" type="string">
            <column name="first_name" length="25" not-null="true" />
        </property>
        <property name="lastName" type="string">
            <column name="last_name" length="25" not-null="true" />
        </property>
        <property name="birthDate" type="date">
            <column name="birth_date" length="10" not-null="true" />
        </property>
        <property name="height" type="short">
            <column name="height" not-null="true" />
        </property>
        <property name="weight" type="big_decimal">
            <column name="weight" precision="4" scale="1" not-null="true" />
        </property>
        <property name="position" type="string">
            <column name="position" length="10" not-null="true" />
        </property>
        <property name="playerNum" type="short">
            <column name="player_num" not-null="true" unique="true" />
        </property>
        <property name="goals" type="short">
            <column name="goals" not-null="true" />
        </property>
        <property name="gamesPlayed" type="short">
            <column name="games_played" not-null="true" />
        </property>
```

```
        <property name="yellowCards" type="short">
            <column name="yellow_cards" not-null="true" />
        </property>
        <property name="redCards" type="short">
            <column name="red_cards" not-null="true" />
        </property>
        <property name="notes" type="string">
            <column name="notes" length="5000" not-null="true" />
        </property>
        <set name="participations" inverse="true" lazy="true"
table="participation" fetch="select">
            <key>
                <column name="player_id" not-null="true" />
            </key>
            <one-to-many class="com.grasshopper.entity.Participation" />
        </set>
    </class>
</hibernate-mapping>
```

Seat.hbm.xml file

```
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.grasshopper.entity.Seat" table="seat">
        <id name="seatId" type="java.lang.Integer">
            <column name="seat_id" />
            <generator class="identity" />
        </id>
        <many-to-one name="section" class="com.grasshopper.entity.Section"
fetch="select">
            <column name="section_id" not-null="true" />
        </many-to-one>
        <property name="rawNum" type="int">
            <column name="raw_num" not-null="true" />
        </property>
        <property name="seatNum" type="int">
            <column name="seat_num" not-null="true" />
        </property>
        <set name="tickets" inverse="true" lazy="true" table="ticket"
fetch="select">
            <key>
                <column name="seat_id" not-null="true" />
            </key>
            <one-to-many class="com.grasshopper.entity.Ticket" />
        </set>
    </class>
</hibernate-mapping>
```

Seatprice.hbm.xml file

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.grasshopper.entity.Seatprice" table="seatprice">
        <id name="seatpriceId" type="java.lang.Integer">
            <column name="seatprice_id" />
```

```
            <generator class="identity" />
        </id>
        <many-to-one name="matches" class="com.grasshopper.entity.Matches"
fetch="select">
            <column name="match_id" not-null="true" />
        </many-to-one>
        <many-to-one name="section" class="com.grasshopper.entity.Section"
fetch="select">
            <column name="section_id" not-null="true" />
        </many-to-one>
        <property name="price" type="big_decimal">
            <column name="price" precision="4" not-null="true" />
        </property>
    </class>
</hibernate-mapping>
```

Section.hbm.xml file

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.grasshopper.entity.Section" table="section">
        <id name="sectionId" type="java.lang.Integer">
            <column name="section_id" />
            <generator class="identity" />
        </id>
        <property name="sectionName" type="string">
            <column name="section_name" length="20" not-null="true" />
        </property>
        <set name="seats" inverse="true" lazy="true" table="seat"
fetch="select">
            <key>
                <column name="section_id" not-null="true" />
            </key>
            <one-to-many class="com.grasshopper.entity.Seat" />
        </set>
    </class>
</hibernate-mapping>
```

Ticket.hbm.xml file

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.grasshopper.entity.Ticket" table="ticket">
        <id name="ticketId" type="java.lang.Integer">
            <column name="ticket_id" />
            <generator class="identity" />
        </id>
        <many-to-one name="booking" class="com.grasshopper.entity.Booking"
fetch="select" cascade="all">
            <column name="book_id" />
        </many-to-one>
```

```
        <many-to-one name="seat" class="com.grasshopper.entity.Seat"
fetch="select">
            <column name="seat_id" not-null="true" />
        </many-to-one>
        <many-to-one name="matches" class="com.grasshopper.entity.Matches"
fetch="select" >
            <column name="match_id" not-null="true" />
        </many-to-one>
        <property name="available" type="boolean">
            <column name="available" not-null="true" default="true"/>
        </property>
    </class>
</hibernate-mapping>
```

User.hbm.xml file

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.grasshopper.entity.User" table="user">
        <id name="userId" type="java.lang.Integer">
            <column name="user_id" />
            <generator class="identity" />
        </id>
        <property name="firstName" type="string">
            <column name="first_name" length="25" not-null="true" />
        </property>
        <property name="lastName" type="string">
            <column name="last_name" length="25" not-null="true" />
        </property>
        <property name="address" type="string">
            <column name="address" length="70" not-null="true" />
        </property>
        <property name="city" type="string">
            <column name="city" length="30" not-null="true" />
        </property>
        <property name="county" type="string">
            <column name="county" length="30" not-null="true" />
        </property>
        <property name="country" type="string">
            <column name="country" length="25" not-null="true" />
        </property>
        <property name="phone" type="string">
            <column name="phone" length="15" not-null="true" />
        </property>
        <property name="email" type="string">
            <column name="email" length="45" not-null="true" />
        </property>
        <set name="creditCards" inverse="true" lazy="true"
table="credit_card" fetch="select">
            <key>
                <column name="user_id" not-null="true" />
            </key>
            <one-to-many class="com.grasshopper.entity.CreditCard" />
```

```
        </set>
        <set name="bookings" inverse="true" lazy="true" table="booking"
fetch="select">
            <key>
                <column name="user_id" not-null="true" />
            </key>
            <one-to-many class="com.grasshopper.entity.Booking" />
        </set>
        <set name="accounts" inverse="true" lazy="false" table="account"
fetch="select" cascade="all">
            <key>
                <column name="user_id" not-null="true" />
            </key>
            <one-to-many class="com.grasshopper.entity.Account" />
        </set>
    </class>
</hibernate-mapping>
```

Hibernate DAO classes

AbstractDaoHibernate.java

```
package com.grasshopper.dao.orm.hibernate;

import org.springframework.orm.hibernate3.support.HibernateDaoSupport;

import com.grasshopper.dao.AbstractDao;

public abstract class AbstractDaoHibernate extends HibernateDaoSupport
implements
            AbstractDao {

}
```

AccountDaoHibernate.java

```
package com.grasshopper.dao.orm.hibernate;

import java.util.Collections;
import java.util.List;
import java.util.Set;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.util.CollectionUtils;

import com.grasshopper.dao.AccountDao;
import com.grasshopper.entity.Account;
import com.grasshopper.entity.User;

@Transactional
public class AccountDaoHibernate extends GenericDaoHibernate<Account,
Integer>
      implements AccountDao {
```

```
    private Log logger = LogFactory.getLog(AccountDaoHibernate.class);

    public boolean checkUserNameAvailable(String username) {
      String hql = "FROM " + Account.class.getName() + " WHERE username = ?";

      @SuppressWarnings("unchecked")
      List result = getHibernateTemplate().find(hql, username);

      return (result == null || result.isEmpty());
    }

    public Set<Account> findAccountByUserExample(User user) {
      @SuppressWarnings("unchecked")
      List<User> resLst = (List<User>) getHibernateTemplate().findByExample(
            user);
      if (!CollectionUtils.isEmpty(resLst)) {
          if (resLst.size() == 1) {
            User u = resLst.get(0);
            return u.getAccounts();
          } else {
            return Collections.<Account> emptySet();
          }
      } else {
          return Collections.<Account> emptySet();
      }
    }

    public Account findAccount(String username, String password) {
      String findHql = "from Account a WHERE a.username = ? AND a.password =
?";
      @SuppressWarnings("unchecked")
      List<Account> foundAccounts = getHibernateTemplate().find(findHql,
            new Object[] { username, password });

      Account resultAccount = null;
      if (!CollectionUtils.isEmpty(foundAccounts)) {
          resultAccount = foundAccounts.get(0);
          if (foundAccounts.size() > 1 && logger.isWarnEnabled()) {
            logger
                  .warn(foundAccounts.size()
                        + " accounts were found for username ["
                        + username
                        + "] and password ["
                        + password
                        + "]. Expected only one. Returning the first matching
account.");
          }
      }
      return resultAccount;
    }
}
```

BookingDaoHibernate.java

```
package com.grasshopper.dao.orm.hibernate;

import com.grasshopper.dao.BookingDao;
import com.grasshopper.entity.Booking;

public class BookingDaoHibernate extends GenericDaoHibernate<Booking,
Integer>
      implements BookingDao {
}
```

CreditCardDaoHibernate.java

```
package com.grasshopper.dao.orm.hibernate;

import java.util.List;

import org.springframework.util.CollectionUtils;

import com.grasshopper.dao.CreditCardDao;
import com.grasshopper.entity.CreditCard;

public class CreditCardDaoHibernate extends
      GenericDaoHibernate<CreditCard, Integer> implements CreditCardDao {

    public boolean isCreditCardRegistered(String number) {
      String hql = "FROM " + CreditCard.class.getName() + " WHERE number =
?";

      @SuppressWarnings("unchecked")
      List result = getHibernateTemplate().find(hql, number);

      return !CollectionUtils.isEmpty(result);
    }

}
```

GenericDaoHibernate.java

```
package com.grasshopper.dao.orm.hibernate;

import java.io.Serializable;
import java.lang.reflect.ParameterizedType;
import java.util.List;
import java.util.Map;

import org.hibernate.LockMode;
import org.springframework.transaction.annotation.Transactional;

import com.grasshopper.dao.GenericDao;
import com.grasshopper.entity.EntityId;
import com.grasshopper.exception.GrasshopperDaoException;
@Transactional
```

```
public abstract class GenericDaoHibernate<T extends EntityId<PK>, PK extends
Serializable>
      extends AbstractDaoHibernate implements GenericDao<T, PK> {

    private Class<T> persistentClass;

    @SuppressWarnings("unchecked")
    public GenericDaoHibernate() {
      Class clazz = getClass();
      while (!(clazz.getGenericSuperclass() instanceof ParameterizedType)) {
          clazz = clazz.getSuperclass();
      }

      this.persistentClass = (Class<T>) ((ParameterizedType) clazz
            .getGenericSuperclass()).getActualTypeArguments()[0];

    }

    public void delete(T object) {
      getHibernateTemplate().delete(object);
    }

    void test()
    {
    }

    public void deleteAll() {
      String query = "delete from " + persistentClass.getSimpleName();
      getHibernateTemplate().bulkUpdate(query);
    }

    public void evict(T object) {
      getHibernateTemplate().evict(object);
    }

    @SuppressWarnings("unchecked")
    public List<T> findAll() {
      return getHibernateTemplate().loadAll(persistentClass);
    }

    public void flush() {
      getHibernateTemplate().flush();
    }

    public void clear() {
      getHibernateTemplate().clear();
    }

    @SuppressWarnings("unchecked")
    public T findById(PK id, boolean lock) {
      T entity;
      if (lock)
          entity = (T) getSession().get(persistentClass, id,
                LockMode.UPGRADE);
      else
          entity = (T) getSession().get(persistentClass, id);
```

```
  return entity;
}

public T saveOrUpdate(T entity) {
  getSession().saveOrUpdate(entity);
  return entity;
}

static int i =0;

@SuppressWarnings("unchecked")
public void save(T object) {
  PK id = (PK) getHibernateTemplate().save(object);
  object.setId(id);
}

public T update(T object) {
  getHibernateTemplate().update(object);
  return object;
}

/**
 * Query entities by query.
 *
 * @param queryString
 *            query
 * @param params
 *            params for query
 * @throws GrasshopperDaoException
 *             if something goes wrong
 * @return list of objects.
 */
@SuppressWarnings("unchecked")
protected List findByNamedParams(String queryString,
      Map<String, Object> params) throws GrasshopperDaoException {
  String[] paramNames = params.keySet()
        .toArray(new String[params.size()]);
  Object[] paramValues = params.values().toArray();
  return getHibernateTemplate().findByNamedParam(queryString, paramNames,
        paramValues);
}

/**
 * Query entities by named query.
 *
 * @param queryString
 *            query
 * @param params
 *            params for query
 * @throws GrasshopperDaoException
 *             if something goes wrong
 * @return list of objects.
 */
@SuppressWarnings("unchecked")
protected List findByNamedQueryAndNamedParam(String queryName,
      Map<String, Object> params) throws GrasshopperDaoException {
  String[] paramNames = params.keySet()
```

```
            .toArray(new String[params.size()]);
      Object[] paramValues = params.values().toArray();
      return getHibernateTemplate().findByNamedQueryAndNamedParam(queryName,
            paramNames, paramValues);
    }
}
```

MatchesDaoHibernate.java

```
package com.grasshopper.dao.orm.hibernate;

import java.sql.SQLException;
import java.util.List;

import org.hibernate.HibernateException;
import org.hibernate.Query;
import org.hibernate.Session;
import org.springframework.orm.hibernate3.HibernateCallback;

import com.grasshopper.dao.MatchesDao;
import com.grasshopper.entity.Matches;
import com.grasshopper.entity.Ticket;

public class MatchesDaoHibernate extends GenericDaoHibernate<Matches,
Integer>
      implements MatchesDao {

    @SuppressWarnings("unchecked")
    public int getSoldTicketsCount(Matches match) {
      List res = getHibernateTemplate()
            .find(
                  "select count(t) from Matches m join m.tickets t where
t.available = false and m.matchId = ?",
                  match.getMatchId());
      int result;
      if (!res.isEmpty()) {
          result = ((Number) res.get(0)).intValue();
      } else {
          result = -1;
      }
      return result;
    }

    @SuppressWarnings("unchecked")
    public List<Matches> getUpcomingMatches(final int matchesAmountLimit) {
      final String hql = "FROM Matches m ORDER BY m.matchDate DESC";

      return (List<Matches>) getHibernateTemplate().executeFind(
            new HibernateCallback() {

                public Object doInHibernate(Session session)
                      throws HibernateException, SQLException {
                  Query query = session.createQuery(hql);
                  query.setFirstResult(0);
                  query.setMaxResults(matchesAmountLimit);
```

```
                return query.list();
              }
            });
    }

    @SuppressWarnings("unchecked")
    public List<Ticket> findAvailableTickets(Integer matchId) {
      String hql = "SELECT t FROM Ticket t JOIN t.matches m WHERE t.available
= true AND m.matchId = ?";
      return (List<Ticket>) getHibernateTemplate().find(hql, matchId);
    }

}
```

ParticipationDaoHibernate.java

```
package com.grasshopper.dao.orm.hibernate;

import com.grasshopper.dao.ParticipationDao;
import com.grasshopper.entity.Participation;

public class ParticipationDaoHibernate extends
      GenericDaoHibernate<Participation, Integer> implements ParticipationDao
{

}
```

PlayerDaoHibernate.java

```
package com.grasshopper.dao.orm.hibernate;

import com.grasshopper.dao.PlayerDao;
import com.grasshopper.entity.Player;

public class PlayerDaoHibernate extends GenericDaoHibernate<Player, Integer>
      implements PlayerDao {
}
```

SeatDaoHibernate.java

```
package com.grasshopper.dao.orm.hibernate;

import com.grasshopper.dao.SeatDao;
import com.grasshopper.entity.Seat;

public class SeatDaoHibernate extends GenericDaoHibernate<Seat, Integer>
      implements SeatDao {

    public Integer getOverallSeatCount() {
      return ((Number) (getHibernateTemplate().find(
            "select count(s) from Seat s").get(0))).intValue();
    }

}
```

SectionDaoHibernate.java

```
package com.grasshopper.dao.orm.hibernate;

import com.grasshopper.dao.SectionDao;
import com.grasshopper.entity.Section;

public class SectionDaoHibernate extends GenericDaoHibernate<Section,
Integer>
      implements SectionDao {
}
```

TicketDaoHibernate.java

```
package com.grasshopper.dao.orm.hibernate;

import com.grasshopper.dao.TicketDao;
import com.grasshopper.entity.Ticket;

public class TicketDaoHibernate extends GenericDaoHibernate<Ticket, Integer>
      implements TicketDao {
}
```

UserDaoHibernate.java

```
package com.grasshopper.dao.orm.hibernate;

import com.grasshopper.dao.UserDao;
import com.grasshopper.entity.User;
public class UserDaoHibernate extends GenericDaoHibernate<User, Integer>
      implements UserDao {
}
```

Appendix E

Application's Configuration Files

applicationContext-hibernate.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
      xmlns:tx="http://www.springframework.org/schema/tx"
      xsi:schemaLocation="
            http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
            http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
            http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

      <context:annotation-config />

      <bean id="sessionFactory"

      class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
            <property name="dataSource" ref="dataSource" />
            <property name="mappingResources">
                  <list>
                        <value>Account.hbm.xml</value>
                        <value>Booking.hbm.xml</value>
                        <value>CreditCard.hbm.xml</value>
                        <value>Matches.hbm.xml</value>
                        <value>Participation.hbm.xml</value>
                        <value>Player.hbm.xml</value>
                        <value>Seat.hbm.xml</value>
                        <value>Seatprice.hbm.xml</value>
                        <value>Section.hbm.xml</value>
                        <value>Ticket.hbm.xml</value>
                        <value>User.hbm.xml</value>
                        <value>Authorities.hbm.xml</value>
                  </list>
            </property>
            <property name="hibernateProperties">
                  <props>
                        <prop
key="hibernate.dialect">${hibernate.dialect}</prop>
                        <prop
key="hibernate.show_sql">${hibernate.show_sql}</prop>
                        <prop
key="hibernate.generate_statistics">${hibernate.generate_statistics}</prop>
                        <prop
key="hibernate.jdbc.batch_size">${hibernate.jdbc.batch_size}</prop>
                        <prop
key="hibernate.show_sql">${hibernate.show_sql}</prop>
                        <prop
key="hibernate.format_sql">${hibernate.format_sql}</prop>
```

```
                        <prop key="hibernate.query.substitutions">true 1,
false 0</prop>
                        <prop
key="hibernate.hbm2ddl.auto">${hibernate.hbm2ddl.auto}</prop>
                  </props>
            </property>
      </bean>

      <bean id="txManagerHibernate"

      class="org.springframework.orm.hibernate3.HibernateTransactionManager">
            <property name="sessionFactory" ref="sessionFactory" />
            <property name="dataSource" ref="dataSource" />
      </bean>

      <tx:annotation-driven transaction-manager="txManagerHibernate" />

      <bean id="abstractDaoHibernate"
class="com.grasshopper.dao.orm.hibernate.AbstractDaoHibernate"
            abstract="true">
            <property name="sessionFactory" ref="sessionFactory" />
      </bean>

      <bean id="genericDaoHibernate"
class="com.grasshopper.dao.orm.hibernate.GenericDaoHibernate"
            abstract="true" parent="abstractDaoHibernate" />

      <bean id="userDao"
class="com.grasshopper.dao.orm.hibernate.UserDaoHibernate"
            parent="genericDaoHibernate" />
      <bean id="accountDao"
class="com.grasshopper.dao.orm.hibernate.AccountDaoHibernate"
            parent="genericDaoHibernate" />
      <bean id="sectionDao"
class="com.grasshopper.dao.orm.hibernate.SectionDaoHibernate"
            parent="genericDaoHibernate" />
      <bean id="seatDao"
class="com.grasshopper.dao.orm.hibernate.SeatDaoHibernate"
            parent="genericDaoHibernate" />
      <bean id="matchesDao"
class="com.grasshopper.dao.orm.hibernate.MatchesDaoHibernate"
            parent="genericDaoHibernate" />
      <bean id="ticketDao"
class="com.grasshopper.dao.orm.hibernate.TicketDaoHibernate"
            parent="genericDaoHibernate" />
      <bean id="bookingDao"
class="com.grasshopper.dao.orm.hibernate.BookingDaoHibernate"
            parent="genericDaoHibernate" />
      <bean id="playerDao"
class="com.grasshopper.dao.orm.hibernate.PlayerDaoHibernate"
            parent="genericDaoHibernate" />
      <bean id="participationDao"

      class="com.grasshopper.dao.orm.hibernate.ParticipationDaoHibernate"
            parent="genericDaoHibernate" />
</beans>
```

hibernate.properties file

```
hibernate.dialect=org.hibernate.dialect.MySQLDialect
hibernate.generate_statistics=true
hibernate.jdbc.batch_size=50
hibernate.show_sql=false
hibernate.format_sql=false
hibernate.hbm2ddl.auto=verify
```

applicationContext-datasource-dbcp.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
destroy-method="close">
        <property name="driverClassName" value="${jdbc.driverClassName}"/>
        <property name="url" value="${jdbc.url}"/>
        <property name="username" value="${jdbc.username}"/>
        <property name="password" value="${jdbc.password}"/>
        <property name="initialSize" value="5"/>
        <property name="maxActive" value="10"/>
        <property name="maxIdle" value="10"/>
    </bean>

</beans>
```

applicationContext-datasource-jndi.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
            http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <bean id="dataSource"
class="org.springframework.jndi.JndiObjectFactoryBean">
        <property name="jndiName" value="java:comp/env/GrasshopperDB"/>
    </bean>

</beans>
```

applicationContext-propertyConfigurer.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

      <bean id="hibernatePropertyConfigurer"
```

```
	class="org.springframework.beans.factory.config.PropertyPlaceholderConf
igurer">
		<property name="locations">
			<list>
				<value>classpath:datasource.properties</value>
				<value>classpath:hibernate.properties</value>
			</list>
		</property>
	</bean>

</beans>
```

persistence.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0"
	xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
	<persistence-unit name="GrasshopperPU" transaction-
type="RESOURCE_LOCAL">

	<provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
		<class>com.grasshopper.entity.Account</class>
		<class>com.grasshopper.entity.Authorities</class>
		<class>com.grasshopper.entity.AuthoritiesId</class>
		<class>com.grasshopper.entity.Booking</class>
		<class>com.grasshopper.entity.CreditCard</class>
		<class>com.grasshopper.entity.Matches</class>
		<class>com.grasshopper.entity.Participation</class>
		<class>com.grasshopper.entity.Player</class>
		<class>com.grasshopper.entity.Seat</class>
		<class>com.grasshopper.entity.Seatprice</class>
		<class>com.grasshopper.entity.Section</class>
		<class>com.grasshopper.entity.Ticket</class>
		<class>com.grasshopper.entity.User</class>
		<properties>
			<property name="eclipselink.jdbc.driver"
value="com.mysql.jdbc.Driver"/>
			<property name="eclipselink.jdbc.url"
value="jdbc:mysql://localhost:3306/grasshopper"/>
			<property name="eclipselink.ddl-generation"
value="verify"/>
			<property name="eclipselink.jdbc.user"
value="grasshopper"/>
			<property name="eclipselink.jdbc.password" value="gsh09"/>
			<property name="eclipselink.logging.level" value="INFO"/>
			<property name="eclipselink.target-database"
value="MySQL"/>
		</properties>
	</persistence-unit>
</persistence>
```

applicationContext-jpa.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
      xmlns:tx="http://www.springframework.org/schema/tx"
      xsi:schemaLocation="
            http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
            http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
            http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

      <context:load-time-weaver />
      <context:component-scan base-package="com.grasshopper.entity" />
      <context:annotation-config />

      <bean id="jpaVendorAdapter"

      class="org.springframework.orm.jpa.vendor.EclipseLinkJpaVendorAdapter">
            <property name="databasePlatform"

      value="org.eclipse.persistence.platform.database.MySQLPlatform" />
            <property name="showSql" value="true" />
      </bean>

      <bean id="jpaDialect"
            class="org.springframework.orm.jpa.vendor.EclipseLinkJpaDialect"
/>

      <bean id="entityManagerFactory"

      class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBe
an">
            <property name="jpaVendorAdapter" ref="jpaVendorAdapter" />
            <property name="persistenceXmlLocation"
value="${persistence.xml.file}" />
            <property name="persistenceUnitName" value="GrasshopperPU" />
            <property name="jpaDialect" ref="jpaDialect" />
      </bean>

      <bean id="txManager"
class="org.springframework.orm.jpa.JpaTransactionManager">
            <property name="entityManagerFactory" ref="entityManagerFactory"
/>
            <property name="dataSource" ref="dataSource" />
      </bean>

      <tx:annotation-driven transaction-manager="txManager" />

      <bean id="abstractDaoJpa"
class="com.grasshopper.dao.orm.jpa.AbstractDaoJpa"
            abstract="true">
```

```
            <property name="entityManagerFactory" ref="entityManagerFactory"
/>
      </bean>

      <bean id="genericDaoJpa"
class="com.grasshopper.dao.orm.jpa.GenericDaoJpa"
            abstract="true" parent="abstractDaoJpa" />

      <bean id="accountDaoJpa"
class="com.grasshopper.dao.orm.jpa.AccountDaoJpa"
            parent="genericDaoJpa" />
      <bean id="bookingDaoJpa"
class="com.grasshopper.dao.orm.jpa.BookingDaoJpa"
            parent="genericDaoJpa" />
      <bean id="matchesDaoJpa"
class="com.grasshopper.dao.orm.jpa.MatchesDaoJpa"
            parent="genericDaoJpa" />
      <bean id="participationDaoJpa"
class="com.grasshopper.dao.orm.jpa.ParticipationDaoJpa"
            parent="genericDaoJpa" />
      <bean id="playerDaoJpa"
class="com.grasshopper.dao.orm.jpa.PlayerDaoJpa"
            parent="genericDaoJpa" />
      <bean id="seatDaoJpa" class="com.grasshopper.dao.orm.jpa.SeatDaoJpa"
            parent="genericDaoJpa" />
      <bean id="sectionDaoJpa"
class="com.grasshopper.dao.orm.jpa.SectionDaoJpa"
            parent="genericDaoJpa" />
      <bean id="ticketDaoJpa"
class="com.grasshopper.dao.orm.jpa.TicketDaoJpa"
            parent="genericDaoJpa" />
      <bean id="userDaoJpa" class="com.grasshopper.dao.orm.jpa.UserDaoJpa"
            parent="genericDaoJpa" />
</beans>
```

Appendix F

Object-Relational Mapping with EclipseLink

EclipseLink Mapping Files with Annotations

Account.java

```
package com.grasshopper.entity;

import java.util.HashSet;
import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name = "account")
public class Account implements EntityId<Integer> {
    private static final long serialVersionUID = -8899220773768056559L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "account_id")
    private Integer accountId;

    @ManyToOne(fetch = FetchType.EAGER,cascade = CascadeType.ALL)
    @JoinColumn(name = "user_id")
    private User user;

    @Column(nullable = false)
    private String username;

    @Column(nullable = false)
    private boolean enabled;

    @Column(nullable = false)
    private String password;

    @OneToMany(mappedBy = "account", fetch = FetchType.LAZY, cascade = {
CascadeType.ALL })
    private Set<Authorities> authorities = new HashSet<Authorities>(0);
```

```
    public Account(User user, String username, boolean enabled, String
password) {
      this.user = user;
      this.username = username;
      this.enabled = enabled;
      this.password = password;
    }

    public Account(User user, String username, boolean enabled,
          String password, Set<Authorities> authoritieses) {
      this.user = user;
      this.username = username;
      this.enabled = enabled;
      this.password = password;
      this.authorities = authoritieses;
    }

    public Integer getId() {
      return accountId;
    }

    public void setId(Integer id) {
      this.accountId = id;
    }

    public Account() {
    }

    public Integer getAccountId() {
      return this.accountId;
    }

    public void setAccountId(Integer accountId) {
      this.accountId = accountId;
    }

    public User getUser() {
      return this.user;
    }

    public void setUser(User user) {
      this.user = user;
    }

    public String getUsername() {
      return this.username;
    }

    public void setUsername(String username) {
      this.username = username;
    }

    public boolean isEnabled() {
      return this.enabled;
    }

    public void setEnabled(boolean enabled) {
```

```
      this.enabled = enabled;
    }

    public String getPassword() {
      return this.password;
    }

    public void setPassword(String password) {
      this.password = password;
    }

    public Set<Authorities> getAuthorities() {
      return this.authorities;
    }

    public void setAuthorities(Set<Authorities> authorities) {
      this.authorities = authorities;
    }
}
```

Authorities.java

```
package com.grasshopper.entity;

import javax.persistence.EmbeddedId;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

@Entity
@Table(name = "authorities")
public class Authorities implements EntityId<AuthoritiesId> {
    private static final long serialVersionUID = -5388306759485097127L;

    @EmbeddedId
    private AuthoritiesId id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "username", nullable = false, insertable = false,
updatable = false)
    private Account account;

    public Authorities() {
    }

    public Authorities(AuthoritiesId id, Account account) {
      this.id = id;
      this.account = account;
    }

    public AuthoritiesId getId() {
      return this.id;
    }

    public void setId(AuthoritiesId id) {
```

```
      this.id = id;
    }

    public Account getAccount() {
      return this.account;
    }

    public void setAccount(Account account) {
      this.account = account;
    }

    @Override
    public int hashCode() {
      final int prime = 31;
      int result = 1;
      result = prime * result + ((account == null) ? 0 : account.hashCode());
      result = prime * result + ((id == null) ? 0 : id.hashCode());
      return result;
    }

    @Override
    public boolean equals(Object obj) {
      if (this == obj)
          return true;
      if (obj == null)
          return false;
      if (getClass() != obj.getClass())
          return false;
      Authorities other = (Authorities) obj;
      if (account == null) {
          if (other.account != null)
            return false;
      } else if (!account.equals(other.account))
          return false;
      if (id == null) {
          if (other.id != null)
            return false;
      } else if (!id.equals(other.id))
          return false;
      return true;
    }
}
```

AuthoritiesId.java

```
package com.grasshopper.entity;

import javax.persistence.Column;
import javax.persistence.Embeddable;

@Embeddable
public class AuthoritiesId implements java.io.Serializable {
    private static final long serialVersionUID = -5814635226588250066L;

    @Column(name = "username", nullable = false)
    private String username;
```

```
    @Column(name = "authority", nullable = false)
    private String authority;

    public AuthoritiesId() {
    }

    public AuthoritiesId(String username, String authority) {
      this.username = username;
      this.authority = authority;
    }

    public String getUsername() {
      return this.username;
    }

    public void setUsername(String username) {
      this.username = username;
    }

    public String getAuthority() {
      return this.authority;
    }

    public void setAuthority(String authority) {
      this.authority = authority;
    }

    public boolean equals(Object other) {
      if ((this == other))
          return true;
      if ((other == null))
          return false;
      if (!(other instanceof AuthoritiesId))
          return false;
      AuthoritiesId castOther = (AuthoritiesId) other;

      return ((this.getUsername() == castOther.getUsername()) || (this
            .getUsername() != null
            && castOther.getUsername() != null && this.getUsername()
            .equals(castOther.getUsername())))
            && ((this.getAuthority() == castOther.getAuthority()) || (this
                  .getAuthority() != null
                  && castOther.getAuthority() != null && this
                  .getAuthority().equals(castOther.getAuthority())));
    }

    public int hashCode() {
      int result = 17;

      result = 37 * result
            + (getUsername() == null ? 0 : this.getUsername().hashCode());
      result = 37 * result
            + (getAuthority() == null ? 0 : this.getAuthority().hashCode());
      return result;
    }

}
```

Booking.java

```
package com.grasshopper.entity;

import java.math.BigDecimal;
import java.util.Date;
import java.util.HashSet;
import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

@Entity
@Table(name = "booking")
public class Booking implements EntityId<Integer> {
    private static final long serialVersionUID = -296434934172913831L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "book_id")
    private Integer bookId;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @Column(name = "pay_total", nullable = false)
    private BigDecimal payTotal;

    @Column(name = "paid_date")
    @Temporal(TemporalType.DATE)
    private Date paidDate;

    @Column(name = "pay_method")
    private String payMethod;

    @OneToMany(mappedBy = "booking", fetch = FetchType.LAZY, cascade = {
CascadeType.ALL })
    private Set<Ticket> tickets = new HashSet<Ticket>(0);

    public Booking() {
    }

    public Booking(User user, BigDecimal payTotal) {
      this.user = user;
```

```
    this.payTotal = payTotal;
  }

  public Booking(User user, BigDecimal payTotal, Date paidDate,
        String payMethod, Set<Ticket> tickets) {
    this.user = user;
    this.payTotal = payTotal;
    this.paidDate = paidDate;
    this.payMethod = payMethod;
    this.tickets = tickets;
  }

  public Integer getBookId() {
    return this.bookId;
  }

  public void setBookId(Integer bookId) {
    this.bookId = bookId;
  }

  public User getUser() {
    return this.user;
  }

  public void setUser(User user) {
    this.user = user;
  }

  public BigDecimal getPayTotal() {
    return this.payTotal;
  }

  public void setPayTotal(BigDecimal payTotal) {
    this.payTotal = payTotal;
  }

  public Date getPaidDate() {
    return this.paidDate;
  }

  public void setPaidDate(Date paidDate) {
    this.paidDate = paidDate;
  }

  public String getPayMethod() {
    return this.payMethod;
  }

  public void setPayMethod(String payMethod) {
    this.payMethod = payMethod;
  }

  public Set<Ticket> getTickets() {
    return this.tickets;
  }

  public void setTickets(Set<Ticket> tickets) {
```

```
      this.tickets = tickets;
    }

    public Integer getId() {
      return bookId;
    }

    public void setId(Integer id) {
      this.bookId = id;
    }

}
```

Booking.java

```
package com.grasshopper.entity;

import java.util.Date;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

@Entity
@Table(name = "credit_card")
public class CreditCard implements EntityId<Integer> {
    private static final long serialVersionUID = 1997259172636246802L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "card_id")
    private Integer cardId;

    @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @Column(name = "fname_on_card", nullable = false)
    private String fnameOnCard;

    @Column(name = "lname_on_card", nullable = false)
    private String lnameOnCard;

    @Column(name = "type", nullable = false)
    private String type;

    @Column(name = "number", nullable = false)
    private String number;
```

```
@Column(name = "cvc", nullable = false)
private String cvc;

@Column(name = "expiry_month", nullable = false)
private byte expiryMonth;

@Column(name = "expiry_year", nullable = false)
@Temporal(TemporalType.DATE)
private Date expiryYear;

public CreditCard() {
}

public CreditCard(User user, String fnameOnCard, String lnameOnCard,
      String type, String number, String cvc, byte expiryMonth,
      Date expiryYear) {
  this.user = user;
  this.fnameOnCard = fnameOnCard;
  this.lnameOnCard = lnameOnCard;
  this.type = type;
  this.number = number;
  this.cvc = cvc;
  this.expiryMonth = expiryMonth;
  this.expiryYear = expiryYear;
}

public Integer getCardId() {
  return this.cardId;
}

public void setCardId(Integer cardId) {
  this.cardId = cardId;
}

public User getUser() {
  return this.user;
}

public void setUser(User user) {
  this.user = user;
}

public String getFnameOnCard() {
  return this.fnameOnCard;
}

public void setFnameOnCard(String fnameOnCard) {
  this.fnameOnCard = fnameOnCard;
}

public String getLnameOnCard() {
  return this.lnameOnCard;
}

public void setLnameOnCard(String lnameOnCard) {
  this.lnameOnCard = lnameOnCard;
```

```
    }

    public String getType() {
      return this.type;
    }

    public void setType(String type) {
      this.type = type;
    }

    public String getNumber() {
      return this.number;
    }

    public void setNumber(String number) {
      this.number = number;
    }

    public String getCvc() {
      return this.cvc;
    }

    public void setCvc(String cvc) {
      this.cvc = cvc;
    }

    public byte getExpiryMonth() {
      return this.expiryMonth;
    }

    public void setExpiryMonth(byte expiryMonth) {
      this.expiryMonth = expiryMonth;
    }

    public Date getExpiryYear() {
      return this.expiryYear;
    }

    public void setExpiryYear(Date expiryYear) {
      this.expiryYear = expiryYear;
    }

    public Integer getId() {
      return cardId;
    }

    public void setId(Integer id) {
      this.cardId = id;
    }
}
```

EntityId.java

```
package com.grasshopper.entity;

import java.io.Serializable;
```

```
public interface EntityId<T extends Serializable> extends Serializable{

    T getId();
    void setId(T id);
}
```

Matches.java

```
package com.grasshopper.entity;

import java.util.Date;
import java.util.HashSet;
import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

@Entity
@Table(name = "matches")
public class Matches implements EntityId<Integer> {
    private static final long serialVersionUID = 2175906684544117259L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "match_id")
    private Integer matchId;

    @Column(name = "match_date", nullable = false)
    @Temporal(TemporalType.DATE)
    private Date matchDate;

    @Column(name = "opposition", nullable = false)
    private String opposition;

    @Column(name = "competition", nullable = false)
    private String competition;

    @Column(name = "kick_off", nullable = false)
    @Temporal(TemporalType.TIME)
    private Date kickOff;

    @Column(name = "venue", nullable = false)
    private String venue;

    @Column(name = "score_grassh")
    private Integer scoreGrassh;

    @Column(name = "score_oppos")
```

```
    private Integer scoreOppos;

    @Column(name = "analysis")
    private String analysis;

    @OneToMany(mappedBy = "matches", fetch = FetchType.LAZY, cascade = {
CascadeType.ALL })
    private Set<Participation> participations = new
HashSet<Participation>(0);

    @OneToMany(mappedBy = "matches", fetch = FetchType.LAZY, cascade = {
CascadeType.ALL })
    private Set<Seatprice> seatprices = new HashSet<Seatprice>(0);

    @OneToMany(mappedBy = "matches", fetch = FetchType.LAZY, cascade = {
CascadeType.ALL })
    private Set<Ticket> tickets = new HashSet<Ticket>(0);

    public Matches() {
    }

    public Matches(Date matchDate, String opposition, String competition,
          Date kickOff, String venue) {
      this.matchDate = matchDate;
      this.opposition = opposition;
      this.competition = competition;
      this.kickOff = kickOff;
      this.venue = venue;
    }

    public Matches(Date matchDate, String opposition, String competition,
          Date kickOff, String venue, Integer scoreGrassh,
          Integer scoreOppos, String analysis,
          Set<Participation> participations, Set<Seatprice> seatprices,
          Set<Ticket> tickets) {
      this.matchDate = matchDate;
      this.opposition = opposition;
      this.competition = competition;
      this.kickOff = kickOff;
      this.venue = venue;
      this.scoreGrassh = scoreGrassh;
      this.scoreOppos = scoreOppos;
      this.analysis = analysis;
      this.participations = participations;
      this.seatprices = seatprices;
      this.tickets = tickets;
    }

    public Integer getMatchId() {
      return this.matchId;
    }

    public void setMatchId(Integer matchId) {
      this.matchId = matchId;
    }

    public Date getMatchDate() {
```

```
    return this.matchDate;
  }

  public void setMatchDate(Date matchDate) {
    this.matchDate = matchDate;
  }

  public String getOpposition() {
    return this.opposition;
  }

  public void setOpposition(String opposition) {
    this.opposition = opposition;
  }

  public String getCompetition() {
    return this.competition;
  }

  public void setCompetition(String competition) {
    this.competition = competition;
  }

  public Date getKickOff() {
    return this.kickOff;
  }

  public void setKickOff(Date kickOff) {
    this.kickOff = kickOff;
  }

  public String getVenue() {
    return this.venue;
  }

  public void setVenue(String venue) {
    this.venue = venue;
  }

  public Integer getScoreGrassh() {
    return this.scoreGrassh;
  }

  public void setScoreGrassh(Integer scoreGrassh) {
    this.scoreGrassh = scoreGrassh;
  }

  public Integer getScoreOppos() {
    return this.scoreOppos;
  }

  public void setScoreOppos(Integer scoreOppos) {
    this.scoreOppos = scoreOppos;
  }

  public String getAnalysis() {
    return this.analysis;
```

```
    }

    public void setAnalysis(String analysis) {
      this.analysis = analysis;
    }

    public Set<Participation> getParticipations() {
      return this.participations;
    }

    public void setParticipations(Set<Participation> participations) {
      this.participations = participations;
    }

    public Set<Seatprice> getSeatprices() {
      return this.seatprices;
    }

    public void setSeatprices(Set<Seatprice> seatprices) {
      this.seatprices = seatprices;
    }

    public Set<Ticket> getTickets() {
      return this.tickets;
    }

    public void setTickets(Set<Ticket> tickets) {
      this.tickets = tickets;
    }

    public Integer getId() {
      return matchId;
    }

    public void setId(Integer id) {
      this.matchId = id;
    }

}
```

Participation.java

```
package com.grasshopper.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

@Entity
@Table(name = "participation")
```

```
public class Participation implements EntityId<Integer> {
    private static final long serialVersionUID = -906474192272059687L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "part_id")
    private Integer partId;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "match_id")
    private Matches matches;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "player_id")
    private Player player;

    public Participation() {
    }

    public Participation(Matches matches, Player player) {
      this.matches = matches;
      this.player = player;
    }

    public Integer getPartId() {
      return this.partId;
    }

    public void setPartId(Integer partId) {
      this.partId = partId;
    }

    public Matches getMatches() {
      return this.matches;
    }

    public void setMatches(Matches matches) {
      this.matches = matches;
    }

    public Player getPlayer() {
      return this.player;
    }

    public void setPlayer(Player player) {
      this.player = player;
    }

    public Integer getId() {
      return partId;
    }

    public void setId(Integer id) {
      this.partId = id;
    }

}
```

Player.java

```java
package com.grasshopper.entity;

import java.math.BigDecimal;
import java.util.Date;
import java.util.HashSet;
import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

@Entity
@Table(name = "player")
public class Player implements EntityId<Integer> {
    private static final long serialVersionUID = -255523085789394442L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "player_id")
    private Integer playerId;

    @Column(name = "team", nullable = false)
    private String team;

    @Column(name = "first_name", nullable = false)
    private String firstName;

    @Column(name = "last_name", nullable = false)
    private String lastName;

    @Column(name = "birth_date", nullable = false)
    @Temporal(TemporalType.DATE)
    private Date birthDate;

    @Column(name = "height", nullable = false)
    private short height;

    @Column(name = "weight", nullable = false)
    private BigDecimal weight;

    @Column(name = "position", nullable = false)
    private String position;

    @Column(name = "player_num", nullable = false)
    private short playerNum;
```

```
    @Column(name = "goals", nullable = false)
    private short goals;

    @Column(name = "games_played", nullable = false)
    private short gamesPlayed;

    @Column(name = "yellow_cards", nullable = false)
    private short yellowCards;

    @Column(name = "red_cards", nullable = false)
    private short redCards;

    @Column(name = "notes", nullable = false)
    private String notes;

    @OneToMany(mappedBy = "player", fetch = FetchType.LAZY, cascade = {
CascadeType.ALL })
    private Set<Participation> participations = new
HashSet<Participation>(0);

    public Player() {
    }

    public Player(String team, String firstName, String lastName,
          Date birthDate, short height, BigDecimal weight, String position,
          short playerNum, short goals, short gamesPlayed, short yellowCards,
          short redCards, String notes) {
      this.team = team;
      this.firstName = firstName;
      this.lastName = lastName;
      this.birthDate = birthDate;
      this.height = height;
      this.weight = weight;
      this.position = position;
      this.playerNum = playerNum;
      this.goals = goals;
      this.gamesPlayed = gamesPlayed;
      this.yellowCards = yellowCards;
      this.redCards = redCards;
      this.notes = notes;
    }

    public Player(String team, String firstName, String lastName,
          Date birthDate, short height, BigDecimal weight, String position,
          short playerNum, short goals, short gamesPlayed, short yellowCards,
          short redCards, String notes, Set<Participation> participations) {
      this.team = team;
      this.firstName = firstName;
      this.lastName = lastName;
      this.birthDate = birthDate;
      this.height = height;
      this.weight = weight;
      this.position = position;
      this.playerNum = playerNum;
      this.goals = goals;
      this.gamesPlayed = gamesPlayed;
      this.yellowCards = yellowCards;
```

```
    this.redCards = redCards;
    this.notes = notes;
    this.participations = participations;
  }

  public Integer getPlayerId() {
    return this.playerId;
  }

  public void setPlayerId(Integer playerId) {
    this.playerId = playerId;
  }

  public String getTeam() {
    return this.team;
  }

  public void setTeam(String team) {
    this.team = team;
  }

  public String getFirstName() {
    return this.firstName;
  }

  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  public String getLastName() {
    return this.lastName;
  }

  public void setLastName(String lastName) {
    this.lastName = lastName;
  }

  public Date getBirthDate() {
    return this.birthDate;
  }

  public void setBirthDate(Date birthDate) {
    this.birthDate = birthDate;
  }

  public short getHeight() {
    return this.height;
  }

  public void setHeight(short height) {
    this.height = height;
  }

  public BigDecimal getWeight() {
    return this.weight;
  }
```

```
public void setWeight(BigDecimal weight) {
  this.weight = weight;
}

public String getPosition() {
  return this.position;
}

public void setPosition(String position) {
  this.position = position;
}

public short getPlayerNum() {
  return this.playerNum;
}

public void setPlayerNum(short playerNum) {
  this.playerNum = playerNum;
}

public short getGoals() {
  return this.goals;
}

public void setGoals(short goals) {
  this.goals = goals;
}

public short getGamesPlayed() {
  return this.gamesPlayed;
}

public void setGamesPlayed(short gamesPlayed) {
  this.gamesPlayed = gamesPlayed;
}

public short getYellowCards() {
  return this.yellowCards;
}

public void setYellowCards(short yellowCards) {
  this.yellowCards = yellowCards;
}

public short getRedCards() {
  return this.redCards;
}

public void setRedCards(short redCards) {
  this.redCards = redCards;
}

public String getNotes() {
  return this.notes;
}

public void setNotes(String notes) {
```

```
      this.notes = notes;
    }

    public Set<Participation> getParticipations() {
      return this.participations;
    }

    public void setParticipations(Set<Participation> participations) {
      this.participations = participations;
    }

    public Integer getId() {
      return playerId;
    }

    public void setId(Integer id) {
      this.playerId = id;
    }

}
```

Seat.java

```
package com.grasshopper.entity;

import java.util.HashSet;
import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name = "seat")
public class Seat implements EntityId<Integer> {
    private static final long serialVersionUID = 4787907333052873520L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "seat_id")
    private Integer seatId;

    @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    @JoinColumn(name = "section_id")
    private Section section;

    @Column(name = "raw_num", nullable = false)
    private int rawNum;
```

```
    @Column(name = "seat_num", nullable = false)
    private int seatNum;

    @OneToMany(mappedBy = "seat", fetch = FetchType.LAZY, cascade = {
CascadeType.ALL })
    private Set<Ticket> tickets = new HashSet<Ticket>(0);

    public Seat() {
    }

    public Seat(Section section, int rawNum, int seatNum) {
      this.section = section;
      this.rawNum = rawNum;
      this.seatNum = seatNum;
    }

    public Seat(Section section, int rawNum, int seatNum, Set<Ticket>
tickets) {
      this.section = section;
      this.rawNum = rawNum;
      this.seatNum = seatNum;
      this.tickets = tickets;
    }

    public Integer getSeatId() {
      return this.seatId;
    }

    public void setSeatId(Integer seatId) {
      this.seatId = seatId;
    }

    public Section getSection() {
      return this.section;
    }

    public void setSection(Section section) {
      this.section = section;
    }

    public int getRawNum() {
      return this.rawNum;
    }

    public void setRawNum(int rawNum) {
      this.rawNum = rawNum;
    }

    public int getSeatNum() {
      return this.seatNum;
    }

    public void setSeatNum(int seatNum) {
      this.seatNum = seatNum;
    }

    public Set<Ticket> getTickets() {
```

```
      return this.tickets;
    }

    public void setTickets(Set<Ticket> tickets) {
      this.tickets = tickets;
    }

    public Integer getId() {
      return seatId;
    }

    public void setId(Integer id) {
      this.seatId = id;
    }

}
```

Seatprice.java

```
package com.grasshopper.entity;

import java.math.BigDecimal;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

@Entity
@Table(name = "seatprice")
public class Seatprice implements EntityId<Integer> {
    private static final long serialVersionUID = -8360795903199178191L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "seatprice_id")
    private Integer seatpriceId;

    @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    @JoinColumn(name = "match_id")
    private Matches matches;

    @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    @JoinColumn(name = "section_id")
    private Section section;

    @Column(name = "price", nullable = false)
    private BigDecimal price;

    public Seatprice() {
    }
```

```
    public Seatprice(Matches matches, Section section, BigDecimal price) {
      this.matches = matches;
      this.section = section;
      this.price = price;
    }

    public Integer getSeatpriceId() {
      return this.seatpriceId;
    }

    public void setSeatpriceId(Integer seatpriceId) {
      this.seatpriceId = seatpriceId;
    }

    public Matches getMatches() {
      return this.matches;
    }

    public void setMatches(Matches matches) {
      this.matches = matches;
    }

    public Section getSection() {
      return section;
    }

    public void setSection(Section section) {
      this.section = section;
    }

    public BigDecimal getPrice() {
      return this.price;
    }

    public void setPrice(BigDecimal price) {
      this.price = price;
    }

    public Integer getId() {
      return seatpriceId;
    }

    public void setId(Integer id) {
      this.seatpriceId = id;
    }

}
```

Section.java

```
package com.grasshopper.entity;

import java.util.HashSet;
import java.util.Set;

import javax.persistence.CascadeType;
```

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name = "section")
public class Section implements EntityId<Integer> {
    private static final long serialVersionUID = -3299275380514574351L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "section_id")
    private Integer sectionId;

    @Column(name = "section_name")
    private String sectionName;

    @OneToMany(mappedBy = "section", fetch = FetchType.LAZY, cascade = {
CascadeType.ALL })
    private Set<Seat> seats = new HashSet<Seat>(0);

    public Section() {
    }

    public Section(String sectionName) {
      this.sectionName = sectionName;
    }

    public Section(String sectionName, Set<Seat> seats) {
      this.sectionName = sectionName;
      this.seats = seats;
    }

    public Integer getSectionId() {
      return this.sectionId;
    }

    public void setSectionId(Integer sectionId) {
      this.sectionId = sectionId;
    }

    public String getSectionName() {
      return this.sectionName;
    }

    public void setSectionName(String sectionName) {
      this.sectionName = sectionName;
    }

    public Set<Seat> getSeats() {
      return this.seats;
    }
```

```
    public void setSeats(Set<Seat> seats) {
      this.seats = seats;
    }

    public Integer getId() {
      return sectionId;
    }

    public void setId(Integer id) {
      this.sectionId = id;
    }

}
```

Ticket.java

```
package com.grasshopper.entity;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

@Entity
@Table(name = "ticket")
public class Ticket implements EntityId<Integer> {
    private static final long serialVersionUID = -2023726346895626711L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ticket_id")
    private Integer ticketId;

    @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    @JoinColumn(name = "book_id")
    private Booking booking;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "seat_id")
    private Seat seat;

    @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    @JoinColumn(name = "match_id")
    private Matches matches;

    @Column(name = "available", nullable = false)
    private boolean available;

    public Ticket() {
    }
```

```
    public Ticket(Seat seat, Matches matches, boolean available) {
      this.seat = seat;
      this.matches = matches;
      this.available = available;
    }

    public Ticket(Booking booking, Seat seat, Matches matches, boolean
available) {
      this.booking = booking;
      this.seat = seat;
      this.matches = matches;
      this.available = available;
    }

    public Integer getTicketId() {
      return this.ticketId;
    }

    public void setTicketId(Integer ticketId) {
      this.ticketId = ticketId;
    }

    public Booking getBooking() {
      return this.booking;
    }

    public void setBooking(Booking booking) {
      this.booking = booking;
    }

    public Seat getSeat() {
      return this.seat;
    }

    public void setSeat(Seat seat) {
      this.seat = seat;
    }

    public Matches getMatches() {
      return this.matches;
    }

    public void setMatches(Matches matches) {
      this.matches = matches;
    }

    public boolean isAvailable() {
      return this.available;
    }

    public void setAvailable(boolean available) {
      this.available = available;
    }

    public Integer getId() {
      return ticketId;
```

```
    }

    public void setId(Integer id) {
      this.ticketId = id;
    }

}
```

User.java

```
package com.grasshopper.entity;

import java.util.HashSet;
import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;

import org.hibernate.validator.Email;
import org.hibernate.validator.NotEmpty;

@Entity
@Table(name = "user")
public class User implements EntityId<Integer> {
    private static final long serialVersionUID = 4292517370671262363L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "user_id")
    private Integer userId;

    @NotEmpty
    @Column(name = "first_name", nullable = false)
    private String firstName;

    @NotEmpty
    @Column(name = "last_name", nullable = false)
    private String lastName;

    @NotEmpty
    @Column(name = "address", nullable = false)
    private String address;

    @NotEmpty
    @Column(name = "city", nullable = false)
    private String city;

    @NotEmpty
    @Column(name = "county", nullable = false)
    private String county;
```

```
    @NotEmpty
    @Column(name = "country", nullable = false)
    private String country;

    @NotEmpty
    @Column(name = "phone", nullable = false)
    private String phone;

    @Email
    @NotEmpty
    @Column(name = "email", nullable = false)
    private String email;

    @OneToMany(mappedBy = "user", fetch = FetchType.LAZY, cascade = {
CascadeType.ALL })
    private Set<CreditCard> creditCards = new HashSet<CreditCard>(0);

    @OneToMany(mappedBy = "user", fetch = FetchType.LAZY, cascade = {
CascadeType.ALL })
    private Set<Booking> bookings = new HashSet<Booking>(0);

    @OneToMany(mappedBy = "user", fetch = FetchType.EAGER, cascade =
CascadeType.ALL)
    private Set<Account> accounts = new HashSet<Account>(0);

    public User() {
    }

    public User(String firstName, String lastName, String address, String
city,
            String county, String country, String phone, String email) {
      this.firstName = firstName;
      this.lastName = lastName;
      this.address = address;
      this.city = city;
      this.county = county;
      this.country = country;
      this.phone = phone;
      this.email = email;
    }

    public User(String firstName, String lastName, String address, String
city,
            String county, String country, String phone, String email,
            Set<CreditCard> creditCards, Set<Booking> bookings,
            Set<Account> accounts) {
      this.firstName = firstName;
      this.lastName = lastName;
      this.address = address;
      this.city = city;
      this.county = county;
      this.country = country;
      this.phone = phone;
      this.email = email;
      this.creditCards = creditCards;
      this.bookings = bookings;
```

```
    this.accounts = accounts;
}

public Integer getUserId() {
    return this.userId;
}

public void setUserId(Integer userId) {
    this.userId = userId;
}

public String getFirstName() {
    return this.firstName;
}

public void setFirstName(String firstName) {
    this.firstName = firstName;
}

public String getLastName() {
    return this.lastName;
}

public void setLastName(String lastName) {
    this.lastName = lastName;
}

public String getAddress() {
    return this.address;
}

public void setAddress(String address) {
    this.address = address;
}

public String getCity() {
    return this.city;
}

public void setCity(String city) {
    this.city = city;
}

public String getCounty() {
    return this.county;
}

public void setCounty(String county) {
    this.county = county;
}

public String getCountry() {
    return this.country;
}

public void setCountry(String country) {
    this.country = country;
```

```
    }

    public String getPhone() {
      return this.phone;
    }

    public void setPhone(String phone) {
      this.phone = phone;
    }

    public String getEmail() {
      return this.email;
    }

    public void setEmail(String email) {
      this.email = email;
    }

    public Set<CreditCard> getCreditCards() {
      return this.creditCards;
    }

    public void setCreditCards(Set<CreditCard> creditCards) {
      this.creditCards = creditCards;
    }

    public Set<Booking> getBookings() {
      return this.bookings;
    }

    public void setBookings(Set<Booking> bookings) {
      this.bookings = bookings;
    }

    public Set<Account> getAccounts() {
      return this.accounts;
    }

    public void setAccounts(Set<Account> accounts) {
      this.accounts = accounts;
    }

    public Integer getId() {
      return userId;
    }

    public void setId(Integer id) {
      this.userId = id;
    }

}
```

EclipseLink DAO classes

AbstractDaoJpa.java

```
package com.grasshopper.dao.orm.jpa;

import org.springframework.orm.jpa.support.JpaDaoSupport;

import com.grasshopper.dao.AbstractDao;

public abstract class AbstractDaoJpa extends JpaDaoSupport implements
      AbstractDao {

}
```

AccountDaoJpa.java

```
package com.grasshopper.dao.orm.jpa;

import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

import com.grasshopper.dao.AccountDao;
import com.grasshopper.entity.Account;
import com.grasshopper.entity.User;

public class AccountDaoJpa extends GenericDaoJpa<Account, Integer> implements
      AccountDao {
    public boolean checkUserNameAvailable(String username) {
      String ql = "FROM" + Account.class.getSimpleName()
            + " WHERE username = ?";
      @SuppressWarnings("unchecked")
      List result = getJpaTemplate().find(ql, username);
      return (result == null || result.isEmpty());
    }

    public Set<Account> findAccountByUserExample(User user) {
      Map<String, Object> params = new HashMap<String, Object>();
      StringBuilder ql = new StringBuilder(" SELECT a FROM ");
      ql.append(Account.class.getSimpleName()).append(
            " a INNER JOIN a.user u WHERE ");

      Integer id = user.getId();
      if (id != null) {
          String idStr = "userId";
          params.put(idStr, id);
          ql.append("u." + idStr + " = :" + idStr);
      } else {
          String firstname = user.getFirstName();
          if (firstname != null) {
            String firstnameStr = "firstName";
            params.put(firstnameStr, firstname);
            ql.append(" u." + firstnameStr + " = :" + firstnameStr);
```

```
            }
            String lastname = user.getLastName();
            if (lastname != null) {
              String lastnameStr = "lastName";
              params.put(lastnameStr, lastname);
              ql.append(" AND u." + lastnameStr + " = :" + lastnameStr);
            }
            String phone = user.getPhone();
            if (phone != null) {
              String phoneStr = "phone";
              params.put(phoneStr, phone);
              ql.append(" AND u." + phoneStr + " = :" + phoneStr);
            }
            String city = user.getCity();
            if (city != null) {
              String cityStr = "city";
              params.put(cityStr, city);
              ql.append(" AND u." + cityStr + " = :" + cityStr);
            }
            String county = user.getCounty();
            if (county != null) {
              String countyStr = "county";
              params.put(countyStr, county);
              ql.append(" AND u." + countyStr + " = :" + countyStr);
            }
            String country = user.getCountry();
            if (country != null) {
              String countryStr = "country";
              params.put(countryStr, country);
              ql.append(" AND u." + countryStr + " = :" + countryStr);
            }
        }

        @SuppressWarnings("unchecked")
        List<Account> foundUsers = (List<Account>) getJpaTemplate()
              .findByNamedParams(ql.toString(), params);

        return new HashSet<Account>(foundUsers);
    }

    public Account findAccount(String username, String password) {
        throw new UnsupportedOperationException();
    }
}
```

BookingDaoJpa.java

```
package com.grasshopper.dao.orm.jpa;

import com.grasshopper.dao.BookingDao;
import com.grasshopper.entity.Booking;

public class BookingDaoJpa extends GenericDaoJpa<Booking, Integer> implements
      BookingDao {

}
```

GenericDaoJpa.java

```
package com.grasshopper.dao.orm.jpa;

import java.io.Serializable;
import java.lang.reflect.ParameterizedType;
import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceException;
import javax.persistence.Query;

import org.springframework.orm.jpa.JpaCallback;
import org.springframework.orm.jpa.JpaTemplate;
import org.springframework.transaction.annotation.Transactional;

import com.grasshopper.dao.GenericDao;
import com.grasshopper.entity.EntityId;

@Transactional
public class GenericDaoJpa<T extends EntityId<PK>, PK extends Serializable>
      extends AbstractDaoJpa implements GenericDao<T, PK> {
    private Class<T> persistentClass;

    @SuppressWarnings("unchecked")
    public GenericDaoJpa() {
      Class clazz = getClass();
      while (!(clazz.getGenericSuperclass() instanceof ParameterizedType)) {
          clazz = clazz.getSuperclass();
      }

      this.persistentClass = (Class<T>) ((ParameterizedType) clazz
            .getGenericSuperclass()).getActualTypeArguments()[0];

    }

    void test()
    {
    }

    public void clear() {
      getJpaTemplate().getEntityManager().clear();
    }

    public void delete(T object) {
      JpaTemplate jpaTemplate = getJpaTemplate();
      jpaTemplate.remove(jpaTemplate.contains(object) ? object : jpaTemplate
            .merge(object));
    }

    public void deleteAll() {
      getJpaTemplate().execute(new JpaCallback() {
          public Object doInJpa(EntityManager em) throws PersistenceException
{
            Query query = em.createQuery("DELETE FROM "
                  + persistentClass.getSimpleName());
```

```
            return query.executeUpdate();
          }
        });
    }

    public void evict(T object) {
      throw new UnsupportedOperationException();
    }

    @SuppressWarnings("unchecked")
    public List<T> findAll() {
      return (List<T>) getJpaTemplate().find(
            "FROM " + persistentClass.getSimpleName());
    }

    public T findById(PK id, boolean lock) {
      return getJpaTemplate().find(persistentClass, id);
    }

    public void flush() {
      getJpaTemplate().flush();
    }

    public void save(T object) {
      getJpaTemplate().persist(object);
    }

    public T saveOrUpdate(T entity) {
      getJpaTemplate().persist(entity);
      return entity;
    }

    public T update(T object) {
      getJpaTemplate().merge(object);
      return object;
    }

}
```

MatchesDaoJpa.java

```
package com.grasshopper.dao.orm.jpa;

import java.util.List;

import com.grasshopper.dao.MatchesDao;
import com.grasshopper.entity.Matches;
import com.grasshopper.entity.Ticket;

public class MatchesDaoJpa extends GenericDaoJpa<Matches, Integer> implements
      MatchesDao {

    @SuppressWarnings("unchecked")
    public int getSoldTicketsCount(Matches match) {
      List res = getJpaTemplate()
            .find(
```

```
                "SELECT count(t) FROM Matches m INNER JOIN m.tickets t
where t.available = false and m.matchId = ?1 ",
                match.getId());
      int result;
      if (!res.isEmpty()) {
          result = ((Number) res.get(0)).intValue();
      } else {
          result = -1;
      }
      return result;
    }

    public List<Matches> getUpcomingMatches(int matchesAmountLimit) {
      throw new UnsupportedOperationException();
    }

    public List<Ticket> findAvailableTickets(Integer matchId) {
      throw new UnsupportedOperationException();
    }

}
```

ParticipationDaoJpa.java

```
package com.grasshopper.dao.orm.jpa;

import com.grasshopper.dao.ParticipationDao;
import com.grasshopper.entity.Participation;

public class ParticipationDaoJpa extends GenericDaoJpa<Participation,
Integer>
      implements ParticipationDao {
}
```

PlayerDaoJpa.java

```
package com.grasshopper.dao.orm.jpa;

import com.grasshopper.dao.PlayerDao;
import com.grasshopper.entity.Player;

public class PlayerDaoJpa extends GenericDaoJpa<Player, Integer> implements
      PlayerDao {

}
```

SeatDaoJpa.java

```
package com.grasshopper.dao.orm.jpa;

import com.grasshopper.dao.SeatDao;
import com.grasshopper.entity.Seat;

public class SeatDaoJpa extends GenericDaoJpa<Seat, Integer> implements
SeatDao {
```

```
    public Integer getOverallSeatCount() {
      return ((Number) (getJpaTemplate().find("select count(s) from Seat s")
            .get(0))).intValue();
    }
}
```

SectionDaoJpa.java

```
package com.grasshopper.dao.orm.jpa;

import com.grasshopper.dao.SectionDao;
import com.grasshopper.entity.Section;

public class SectionDaoJpa extends GenericDaoJpa<Section, Integer> implements
      SectionDao {

}
```

TicketDaoJpa.java

```
package com.grasshopper.dao.orm.jpa;

import com.grasshopper.dao.TicketDao;
import com.grasshopper.entity.Ticket;

public class TicketDaoJpa extends GenericDaoJpa<Ticket, Integer> implements
      TicketDao {
                                        }
```

UserDaoJpa.java

```
package com.grasshopper.dao.orm.jpa;

import com.grasshopper.dao.UserDao;
import com.grasshopper.entity.User;

public class UserDaoJpa extends GenericDaoJpa<User, Integer> implements
UserDao {

                                        }
```

Appendix G

Business Logic and Presentation Layers

Spring Services (Business Logic Layer)

BookingService.java

```
package com.grasshopper.service;

import java.util.List;

import com.grasshopper.entity.Ticket;
import com.grasshopper.service.impl.BookingServiceSpring.BookingData;
import com.grasshopper.service.vo.SectionVO;

/**
 * Service class to manage booking actions
 *
 */
public interface BookingService {

    /**
     * Get sections data including price for particular section
     *
     * @return <code>List</code> of section Value Objects
     */
    List<SectionVO> getSectionsWithPrices(Integer matchId);

    /**
     * Books tickets and return booking ID
     *
     * @param bookingData
     *            important booking data
     * @return booking ID
     */
    Integer bookTickets(BookingData bookingData);

    /**
     * Find available tickets for particular match
     *
     * @param matchId
     *            DB ID of the match
     * @return <code>List</cod> of tickets for match
     */
    List<Ticket> findAvailableTickets(Integer matchId);
}
```

CreditCardService.java

```
package com.grasshopper.service;

import com.grasshopper.entity.CreditCard;
```

```
import com.grasshopper.entity.User;

public interface CreditCardService {
    /**
     * Saves user's credit card data. If such credit card already exists do
     * nothing.
     *
     * @param user owner of credit card
     * @param creditCard
     *             credit card data value object
     * @return <code>true</code> - if the credit card record has been created
in
     *         database, <code>false</code> - if credit card record already
     *         exists in database
     */
    boolean saveCreditCardData(User user, CreditCard creditCard);
}
```

MatchesService.java

```
package com.grasshopper.service;

import java.util.List;

import com.grasshopper.entity.Matches;

public interface MatchesService {

    List<Matches> getUpcomingMatches();

}
```

UserAccountService.java

```
package com.grasshopper.service;

import com.grasshopper.entity.Account;


public interface UserAccountService {
    Account createUserAccount(Account account);

    Account createAdminAccount(Account account);

    boolean checkUserNameAvailable(String username);
}
```

AbstractServiceSpring.java

```
package com.grasshopper.service.impl;

import org.springframework.transaction.annotation.Transactional;

@Transactional(rollbackFor=Exception.class)
public abstract class AbstractServiceSpring {
```

```
}
```

BookingServiceSpring.java

```
package com.grasshopper.service.impl;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Set;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.util.CollectionUtils;

import com.grasshopper.dao.BookingDao;
import com.grasshopper.dao.MatchesDao;
import com.grasshopper.dao.TicketDao;
import com.grasshopper.dao.UserDao;
import com.grasshopper.entity.Account;
import com.grasshopper.entity.Booking;
import com.grasshopper.entity.Matches;
import com.grasshopper.entity.Seatprice;
import com.grasshopper.entity.Section;
import com.grasshopper.entity.Ticket;
import com.grasshopper.entity.User;
import com.grasshopper.service.BookingService;
import com.grasshopper.service.vo.SectionVO;

public class BookingServiceSpring extends AbstractServiceSpring implements
      BookingService {

    @Autowired
    private MatchesDao matchesDao;

    @Autowired
    private UserDao userDao;

    @Autowired
    private TicketDao ticketDao;

    @Autowired
    private BookingDao bookingDao;

    public List<SectionVO> getSectionsWithPrices(Integer matchId) {
      if (matchId == null) {
          return Collections.<SectionVO> emptyList();
      }

      Matches match = matchesDao.findById(matchId, false);
      if (match == null) {
          /* Match is not found, return empty list */
          return Collections.<SectionVO> emptyList();
      }
```

```
        Set<Seatprice> seatPricesForMatch = match.getSeatprices();

        List<SectionVO> result = new ArrayList<SectionVO>(seatPricesForMatch
              .size());
        for (Seatprice seatprice : seatPricesForMatch) {
            Number priceNum = (Number) seatprice.getPrice();
            Double price = priceNum != null ? priceNum.doubleValue() : null;

            Section section = seatprice.getSection();
            Integer sectionId = section.getSectionId();
            String sectionName = section.getSectionName();

            result.add(new SectionVO(sectionId, sectionName, price));
        }

        return result;
    }

    public List<Ticket> findAvailableTickets(Integer matchId)
    {
        return matchesDao.findAvailableTickets(matchId);
    }

    public Integer bookTickets(BookingData bookingData) {
        Account account = bookingData.getUserAccount();
        Integer matchId = bookingData.getMatchId();
        Integer ticketsAmount = bookingData.getTicketAmount();
        double totalPrice = bookingData.getTicketAmount()
              * bookingData.getTicketPrice();

        /* Attach user to ORM session */
        User user = userDao.findById(account.getUser().getUserId(), false);

        /* Create booking */
        Booking booking = new Booking();
        booking.setPayTotal(new BigDecimal(totalPrice));
        booking.setUser(user);

        bookingDao.save(booking);

        List<Ticket> tickets = matchesDao.findAvailableTickets(matchId);

        Integer bookingResult;
        if (CollectionUtils.isEmpty(tickets) || tickets.size() < ticketsAmount)
{
            /*
             * Amount of requested for booking tickets is lower than available
             * tickets amount
             */
            bookingResult = null;
        } else {
            for (int i = 0; i < ticketsAmount; i++) {
              Ticket t = tickets.get(i);
              t.setAvailable(false);
              t.setBooking(booking);

              ticketDao.update(t);
```

```
        }
        bookingResult = booking.getBookId();
    }

    return bookingResult;
}

public static class BookingData {
    private Account userAccount;
    private Integer ticketAmount;
    private Double ticketPrice;
    private Integer matchId;
    private Integer sectionId;

    public BookingData() {
    }

    public Account getUserAccount() {
        return userAccount;
    }

    public void setUserAccount(Account userAccount) {
        this.userAccount = userAccount;
    }

    public Integer getTicketAmount() {
        return ticketAmount;
    }

    public void setTicketAmount(Integer ticketAmount) {
        this.ticketAmount = ticketAmount;
    }

    public Double getTicketPrice() {
        return ticketPrice;
    }

    public void setTicketPrice(Double ticketPrice) {
        this.ticketPrice = ticketPrice;
    }

    public Integer getMatchId() {
        return matchId;
    }

    public void setMatchId(Integer matchId) {
        this.matchId = matchId;
    }

    public Integer getSectionId() {
        return sectionId;
    }

    public void setSectionId(Integer sectionId) {
        this.sectionId = sectionId;
    }
}
```

```
}
```

CreditCardSpring.java

```
package com.grasshopper.service.impl;

import org.springframework.beans.factory.annotation.Autowired;

import com.grasshopper.dao.CreditCardDao;
import com.grasshopper.dao.UserDao;
import com.grasshopper.entity.CreditCard;
import com.grasshopper.entity.User;
import com.grasshopper.service.CreditCardService;

public class CreditCardServiceSpring extends AbstractServiceSpring implements
      CreditCardService {

    @Autowired
    private CreditCardDao creditCardDao;

    @Autowired
    private UserDao userDao;

    public boolean saveCreditCardData(User user, CreditCard creditCard) {
      boolean isCreditCardAlreadyExistent = creditCardDao
            .isCreditCardRegistered(creditCard.getNumber());
      if (!isCreditCardAlreadyExistent) {
          User foundUser = userDao.findById(user.getUserId(), false);
          creditCard.setUser(foundUser);
          creditCardDao.save(creditCard);
      }
      return !isCreditCardAlreadyExistent;
    }

}
```

MatchesServiceSpring.java

```
package com.grasshopper.service.impl;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import com.grasshopper.config.GrasshopperConfig;
import com.grasshopper.dao.MatchesDao;
import com.grasshopper.entity.Matches;
import com.grasshopper.service.MatchesService;

public class MatchesServiceSpring extends AbstractServiceSpring implements
      MatchesService {
    @Autowired
    private MatchesDao matchesDao;

    public List<Matches> getUpcomingMatches() {
      /* Get from config file shown matches amount */
      Integer upcomingMatchesMaxAmount = GrasshopperConfig
```

```
            .getUpcomingMatchesMaxAmount();

      return matchesDao.getUpcomingMatches(upcomingMatchesMaxAmount);
    }

}
```

UserAccountServiceSpring.java

```
package com.grasshopper.service.impl;

import org.apache.commons.lang.Validate;
import org.springframework.beans.factory.annotation.Autowired;

import com.grasshopper.constant.Role;
import com.grasshopper.dao.AccountDao;
import com.grasshopper.entity.Account;
import com.grasshopper.entity.Authorities;
import com.grasshopper.entity.AuthoritiesId;
import com.grasshopper.service.UserAccountService;

public class UserAccountServiceSpring extends AbstractServiceSpring
implements
      UserAccountService {
    @Autowired
    private AccountDao accountDao;

    public UserAccountServiceSpring() {
    }

    public Account createUserAccount(Account account) {
      return createAccount(account, Role.ROLE_USER);
    }

    public Account createAdminAccount(Account account) {
      return createAccount(account, Role.ROLE_ADMIN);
    }

    private Account createAccount(Account account, Role role) {
      Validate.notNull(account, "Account object can't be null");
      Validate.notNull(account.getUser(), "User object can't be null");
      Validate.notNull(role, "Role can't be null");

      Authorities auth = new Authorities();
      auth.setId(new AuthoritiesId(account.getUsername(), role.toString()));

      if (!account.getAuthorities().contains(auth)) {
          account.getAuthorities().add(auth);
      }

      accountDao.save(account);

      return account;
    }

    public boolean checkUserNameAvailable(String username) {
```

```
      Validate.notEmpty(username, " User name can't be null or empty");
      return accountDao.checkUserNameAvailable(username);
    }
}
```

AutenticationManager.java

```
package com.grasshopper.service.security;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;

import javax.faces.context.FacesContext;
import javax.servlet.http.HttpServletRequest;

import org.apache.commons.lang.StringUtils;
import org.springframework.beans.BeansException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.security.Authentication;
import org.springframework.security.GrantedAuthority;
import org.springframework.security.GrantedAuthorityImpl;
import org.springframework.security.context.SecurityContext;
import org.springframework.security.context.SecurityContextHolder;
import org.springframework.security.providers.ProviderManager;
import
org.springframework.security.providers.UsernamePasswordAuthenticationToken;
import org.springframework.security.providers.dao.DaoAuthenticationProvider;
import org.springframework.security.ui.WebAuthenticationDetails;

import com.grasshopper.entity.Account;
import com.grasshopper.entity.Authorities;
import com.grasshopper.entity.AuthoritiesId;

public class AuthenticationManager {

    @Autowired
    private ProviderManager manager;

    @Autowired
    private ApplicationContext context;

    @SuppressWarnings("unchecked")
    public void authenticateUser(Account account) {
      Set<Authorities> authSet = account.getAuthorities();

      List<String> authorities = new ArrayList<String>(authSet.size());
      for (Authorities a : authSet) {
          if (a != null) {
            AuthoritiesId aId = a.getId();
            if (aId != null) {
                String role = aId.getAuthority();
                if (StringUtils.isNotEmpty(role)) {
```

```
                authorities.add(role);
              }
            }

          }
      }

      GrantedAuthority[] grantedAuthorities = new
GrantedAuthority[authorities
            .size()];
      for (int i = 0, size = authorities.size(); i < size; i++) {
          grantedAuthorities[i] = new
GrantedAuthorityImpl(authorities.get(i));
      }

      UsernamePasswordAuthenticationToken authentication = new
UsernamePasswordAuthenticationToken(
            account.getUsername(), account.getPassword(),
            grantedAuthorities);
       Map map = context

.getBeansOfType(org.springframework.security.providers.dao.DaoAuthenticationP
rovider.class);
       Set keys = map.keySet();
       Iterator itr = keys.iterator();
       Object obj = null;
       while (itr.hasNext()) {
       obj = map.get(itr.next());
       }
       DaoAuthenticationProvider daoAuthentication =
       (DaoAuthenticationProvider) obj;
       Authentication au = daoAuthentication.authenticate(authentication);

      SecurityContext securitycontext = SecurityContextHolder.getContext();
      securitycontext.setAuthentication(authentication);
    }

    public void aaa(Account a) {
      SecurityContext context = SecurityContextHolder.getContext();
      try {
          GrantedAuthority[] grantedAuthorities = new GrantedAuthority[] {
new GrantedAuthorityImpl(
                "ROLE_USER") };

          UsernamePasswordAuthenticationToken authentication = new
UsernamePasswordAuthenticationToken(
                a.getUsername(), a.getPassword(), grantedAuthorities);

          HttpServletRequest request =
            (HttpServletRequest)FacesContext
                  .getCurrentInstance()
                        .getExternalContext()
                              .getRequest();

          if (request != null)
          {
            authentication.setDetails(new WebAuthenticationDetails(request));
```

```
            }
            else
            {
              System.out.println("Requets is null");
            }

            Authentication ab = manager.authenticate(authentication);

            System.out.println("Logging with principal : "+ab.getPrincipal());
            System.out.println("And name: "+ab.getName());

            context.setAuthentication(ab);
        } catch (BeansException e) {
            e.printStackTrace();
        }
    }
                                        }
```

SectionVO.java

```
package com.grasshopper.service.vo;

public class SectionVO {
    private Integer sectionId;
    private String sectionName;
    private Double price;

    public SectionVO() {
    }

    public SectionVO(Integer sectionId, String sectionName, Double price) {
      super();
      this.sectionId = sectionId;
      this.sectionName = sectionName;
      this.price = price;
    }

    public Integer getSectionId() {
      return sectionId;
    }

    public void setSectionId(Integer sectionId) {
      this.sectionId = sectionId;
    }

    public String getSectionName() {
      return sectionName;
    }

    public void setSectionName(String sectionName) {
      this.sectionName = sectionName;
    }

    public Double getPrice() {
      return price;
    }
```

```
    public void setPrice(Double price) {
      this.price = price;
    }

}
```

Web Beans

AuthorizationBean.java

```
package com.grasshopper.web.bean;

import java.util.Map;

import javax.faces.application.FacesMessage;
import javax.faces.context.ExternalContext;
import javax.faces.context.FacesContext;

import org.apache.commons.collections.FactoryUtils;
import org.apache.commons.lang.StringUtils;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.hibernate.validator.Length;
import org.hibernate.validator.NotEmpty;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.ui.AbstractProcessingFilter;

import com.grasshopper.entity.Account;
import com.grasshopper.entity.User;
import com.grasshopper.service.UserAccountService;
import com.grasshopper.web.helper.FacesSupport;

public class AuthorizationBean {
    private Log logger = LogFactory.getLog(AuthorizationBean.class);

    @Autowired
    private FacesSupport facesSupport;

    @Autowired
    private UserAccountService userAccountService;

    private Account account;

    public Account getAccount() {
        return account;
    }

    public void setAccount(Account account) {
        this.account = account;
    }

    @NotEmpty
    @Length(min = 4, max = 16)
    // @Pattern(regex = "^[a-zA-Z\\d_]{4,12}$", message = "Invalid user
name.")
```

```
    private String username;
    @NotEmpty
    // @Length(min = 6, max = 12)
    private String password;
    private String confirmPassword;

    public AuthorizationBean() {
      User user = new User();
      account = new Account();
      account.setUser(user);
    }

    public String getUsername() {
      return username;
    }

    public void setUsername(String username) {
      // Check if username has already been registered
      if (userAccountService.checkUserNameAvailable(username)) {
          this.username = username;
      } else {
          facesSupport.addError("registerForm:userName", "username_taken");
      }
    }

    public String getPassword() {
      return password;
    }

    public void setPassword(String password) {
      this.password = password;
    }

    public String getConfirmPassword() {
      return confirmPassword;
    }

    public void setConfirmPassword(String confirmPassword) {
      this.confirmPassword = confirmPassword;
    }

    /**
     * Invoked from auth.jspf to determine if the modal login panel should be
     * displayed.
     */
    public static final void setLoginPanelVisibility() throws Exception {
      FacesContext jsf = FacesContext.getCurrentInstance();
      ExternalContext extCtxt = jsf.getExternalContext();
      if (extCtxt.getRemoteUser() == null) {
          boolean showLogin = false;
          // check for the BadCredentialsException thrown by the Spring
          // Security framework
          Map<String, Object> sessionMap = extCtxt.getSessionMap();
          Object loginError = sessionMap

.get(AbstractProcessingFilter.SPRING_SECURITY_LAST_EXCEPTION_KEY);
          if (loginError != null) {
```

```
            sessionMap
                  .put(

      AbstractProcessingFilter.SPRING_SECURITY_LAST_EXCEPTION_KEY,
                        null);
            jsf.addMessage("loginBtn", new FacesMessage(
                  FacesMessage.SEVERITY_ERROR, "Invalid credentials!",
                  null));
            showLogin = true;
          }
          FacesSupport.showWhenRendered(jsf, "loginPanel", showLogin);
      }
    }

    public void doLogin() throws Exception {
      FacesContext jsf = FacesContext.getCurrentInstance();
      jsf.getExternalContext().dispatch("/j_spring_security_check");
      jsf.responseComplete();
    }

    public synchronized String doRegister() {
      logger.info("Registration takes place...");

      if (StringUtils.isEmpty(confirmPassword)
            || !confirmPassword.equals(password)) {
          facesSupport.addError("registerForm:password",
                "password_not_confirmed");
          return null;
      }

      account.setUsername(username);
      account.setPassword(password);
      account.setEnabled(true);

      try {
          userAccountService
                .createUserAccount(account);

          return "successfullRegistration";
      } catch (Exception e) {
          if (logger.isErrorEnabled())
            logger.error(e.getMessage(), e);
          facesSupport.addError("registerForm:registerButton",
                "exception_msg");
          return null;
      }
    }
}
```

BookingBean.java

```
package com.grasshopper.web.bean;

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
```

```
import java.util.Map;

import javax.faces.model.SelectItem;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.util.CollectionUtils;

import com.grasshopper.config.GrasshopperConfig;
import com.grasshopper.dao.SectionDao;
import com.grasshopper.entity.Account;
import com.grasshopper.entity.CreditCard;
import com.grasshopper.service.BookingService;
import com.grasshopper.service.CreditCardService;
import com.grasshopper.service.impl.BookingServiceSpring.BookingData;
import com.grasshopper.service.vo.SectionVO;
import com.grasshopper.web.helper.FacesSupport;

public class BookingBean {
    public static final String BOOKING_INFO_PARAM = "booking_info";
    private static final String MATCH_ID_REQUEST_PARAM = "matchId";

    @Autowired
    private FacesSupport facesSupport;

    @Autowired
    private BookingService bookingService;

    @Autowired
    private UserSession userSession;

    @Autowired
    private BookingDetails bookingDetails;

    @Autowired
    private CreditCardService creditCardService;

    @Autowired
    private SectionDao sectionDao;

    private Integer sectionChoose;
    private Integer ticketsAmountChoose;
    private Integer matchId;

    private List<SelectItem> sectionsAndPrices;
    private Map<Integer, Double> sectionToPriceMapping;

    /* Credit card data */
    private CreditCard creditCard = new CreditCard();

    /* Booking data value object */
    private BookingData bookingData;

    /**
     * Determine whether stadium section with its prices should be shown.
     */
    public boolean isSectionsAndPricesRendered() {
      return !CollectionUtils.isEmpty(sectionsAndPrices)
```

```
        || getMatchIdFromRequest() != null;
}

public boolean isAvailableTicketsForMatch() {
  String matchId = getMatchIdFromRequest();
  if (matchId == null) {
      return true;
  } else {
      return !CollectionUtils.isEmpty(bookingService
            .findAvailableTickets(new Integer(matchId)));
  }
}

private String getMatchIdFromRequest() {
  return FacesSupport.getRequestValue(MATCH_ID_REQUEST_PARAM);
}

public List<SelectItem> getSectionsAndPrices() {
  String matchId = getMatchIdFromRequest();
  if (matchId == null) {
      if (sectionsAndPrices != null) {
        return this.sectionsAndPrices;
      } else {
        return Collections.<SelectItem> emptyList();
      }
  }

  this.matchId = new Integer(matchId);

  List<SectionVO> sectionData = bookingService
        .getSectionsWithPrices(this.matchId);

  List<SelectItem> selectItems = new ArrayList<SelectItem>(sectionData
        .size());

  this.sectionToPriceMapping = new HashMap<Integer, Double>(sectionData
        .size());

  for (SectionVO section : sectionData) {
      String sectionInfo = section.getSectionName() + ",   price: "
            + section.getPrice() + " €";

      this.sectionToPriceMapping.put(section.getSectionId(), section
            .getPrice());

      selectItems
            .add(new SelectItem(section.getSectionId(), sectionInfo));
  }

  /* Remember the sections till next new request */
  this.sectionsAndPrices = selectItems;

  return selectItems;
}

public List<SelectItem> getTicketsAmount() {
  Integer ticketsForSaleAmount = GrasshopperConfig
```

```
        .getTicketsForSaleAmount();
    List<SelectItem> selectItems = new ArrayList<SelectItem>(
          ticketsForSaleAmount);
    for (int i = 1; i <= ticketsForSaleAmount; i++) {
        selectItems.add(new SelectItem(i));
    }

    return selectItems;
}

public String fillBookingData() {
    Account userAccount = userSession.getAccount();

    Double oneTicketPrice = sectionToPriceMapping.get(this.sectionChoose);

    bookingData = new BookingData();
    bookingData.setUserAccount(userAccount);
    bookingData.setMatchId(this.matchId);
    bookingData.setSectionId(this.sectionChoose);
    bookingData.setTicketAmount(this.ticketsAmountChoose);
    bookingData.setTicketPrice(oneTicketPrice);

    return "creditCardDataPage";
}

public String fillCreditCardDataAndBookTickets() {
    /*
     * false- if amount of available tickets is lower that requested, true-
     * if booking went successfully
     */
    Integer bookingId = bookingService.bookTickets(bookingData);

    String pageOutcome = null;
    if (bookingId != null) {
        /* Add credit card data */
        creditCardService.saveCreditCardData(userSession.getAccount()
              .getUser(), creditCard);

        String sectionName = sectionDao.findById(this.sectionChoose, false)
              .getSectionName();

        bookingDetails.setBookingId(bookingId);
        bookingDetails.setSectionName(sectionName);
        bookingDetails.setOneTicketPrice(bookingData.getTicketPrice());
        bookingDetails.setTicketsAmount(this.ticketsAmountChoose);
        bookingDetails.setTotalTicketPrice(bookingData.getTicketPrice()
              * this.ticketsAmountChoose);

        pageOutcome = "successfullBooking";
    } else {
        facesSupport.addError("creditCardAndBookingForm:bookCommandButton",
              "no_available_tickets");
    }

    return pageOutcome;
}
```

```
    public Integer getSectionChoose() {
      return sectionChoose;
    }

    public void setSectionChoose(Integer sectionChoose) {
      this.sectionChoose = sectionChoose;
    }

    public Integer getTicketsAmountChoose() {
      return ticketsAmountChoose;
    }

    public void setTicketsAmountChoose(Integer ticketsAmoutChoose) {
      this.ticketsAmountChoose = ticketsAmoutChoose;
    }

    public CreditCard getCreditCard() {
      return creditCard;
    }

    public void setCreditCard(CreditCard creditCard) {
      this.creditCard = creditCard;
    }
}
```

BookingDetails.java

```
package com.grasshopper.web.bean;

public class BookingDetails {
    private Integer bookingId;
    private Double oneTicketPrice;
    private Double totalTicketPrice;
    private Integer ticketsAmount;
    private String sectionName;

    public BookingDetails() {
    }

    public Double getOneTicketPrice() {
      return oneTicketPrice;
    }

    public void setOneTicketPrice(Double oneTicketPrice) {
      this.oneTicketPrice = oneTicketPrice;
    }

    public Double getTotalTicketPrice() {
      return totalTicketPrice;
    }

    public void setTotalTicketPrice(Double totalTicketPrice) {
      this.totalTicketPrice = totalTicketPrice;
    }

    public Integer getTicketsAmount() {
      return ticketsAmount;
```

```
    }

    public void setTicketsAmount(Integer ticketsAmount) {
      this.ticketsAmount = ticketsAmount;
    }

    public String getSectionName() {
      return sectionName;
    }

    public void setSectionName(String sectionName) {
      this.sectionName = sectionName;
    }

    public Integer getBookingId() {
      return bookingId;
    }

    public void setBookingId(Integer bookingId) {
      this.bookingId = bookingId;
    }
}
```

InfoBean.Java

```
package com.grasshopper.web.bean;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import com.grasshopper.entity.Matches;
import com.grasshopper.service.MatchesService;
import com.grasshopper.utils.DateUtils;

/**
 * Data bean that provides UI prepared matches information
 */
public class InfoBean implements Serializable{
    private static final long serialVersionUID = 1L;

    @Autowired
    private MatchesService matchesService;

    /**
     * Fetches and returns UI prepared upcoming matches information. The
amount of returned upcoming matches is configurable.
     * @return <code>List</code> of upcoming matches.
     */
    public List<MatchesVO> getUpcomingMatches() {
      List<Matches> upcomingMatches = matchesService.getUpcomingMatches();

      List<MatchesVO> upcomingMatchesInfo = new ArrayList<MatchesVO>(
            upcomingMatches.size());
```

```
      for (Matches match : upcomingMatches) {
          String matchDate = DateUtils.formatDateForUI(match.getMatchDate(),
                DateUtils.DATE_FORMAT_ONLY_DATE);

          String infoString = matchDate + " : "+ match.getOpposition();
          upcomingMatchesInfo.add(new MatchesVO(match.getId(), infoString));
      }

      return upcomingMatchesInfo;
    }

    public void setUpcomingMatches() {
    }

    public static class MatchesVO{
      private Integer id;
      private String matchInfo;

      public MatchesVO(){}

      public MatchesVO(Integer id, String matchInfo) {
          super();
          this.id = id;
          this.matchInfo = matchInfo;
      }

      public Integer getId() {
          return id;
      }

      public void setId(Integer id) {
          this.id = id;
      }

      public String getMatchInfo() {
          return matchInfo;
      }

      public void setMatchInfo(String matchInfo) {
          this.matchInfo = matchInfo;
      }
    }
}
```

MatchesBean.java

```
package com.grasshopper.web.bean;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import com.grasshopper.config.GrasshopperConfig;
import com.grasshopper.dao.MatchesDao;
```

```
import com.grasshopper.entity.Matches;

public class MatchesBean implements Comparator<Matches> {
    @Autowired
    private MatchesDao matchesDao;

    private Integer matchId;

    public Integer getMatchId() {
      return matchId;
    }

    public void setMatchId(Integer matchId) {
      this.matchId = matchId;
    }

    public List<Matches> getMatches() {
      Integer shownMatchesAmount = GrasshopperConfig.getShownMatchesAmount();
      List<Matches> allMatches = matchesDao.findAll();
      // Collections.sort(allMatches, this);

      List<Matches> filteredMatches = new ArrayList<Matches>(
            shownMatchesAmount);
      for (int i = 0, size = allMatches.size(); i < shownMatchesAmount
            && i < size; i++) {
          filteredMatches.add(allMatches.get(i));
      }

      return filteredMatches;
    }

    public int compare(Matches m1, Matches m2) {
      return 0;
    }

    public String getAnalysis() {
      String analysis = null;
      if (matchId != null) {
          Matches match = matchesDao.findById(matchId, false);
          if (match != null) {
            analysis = match.getAnalysis();
          }
      }

      return analysis;
    }

}
```

PlayersBean.java

```
package com.grasshopper.web.bean;

import static com.grasshopper.entity.Player.POSITIONS_GRADATION;
import static com.grasshopper.entity.Player.TEAM_GRADATION;
```

```
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import com.grasshopper.dao.PlayerDao;
import com.grasshopper.entity.Player;
import com.grasshopper.web.helper.FacesSupport;

public class PlayersBean {
    private static final String PLAYER_ID_PARAM_NAME = "playerId";

    @Autowired
    private PlayerDao playerDao;

    private Integer playerId;

    public Integer getPlayerId() {
      return playerId;
    }

    public void setPlayerId(Integer playerId) {
      this.playerId = playerId;
    }

    private Player currPlayer;

    public List<Player> getAllPlayers() {
      List<Player> allPlayers = playerDao.findAll();
      Collections.sort(allPlayers, new Comparator<Player>() {

          public int compare(Player p1, Player p2) {
            int p1TeamIndex = TEAM_GRADATION.indexOf(p1.getTeam());
            int p2TeamIndex = TEAM_GRADATION.indexOf(p2.getTeam());

            if (p1TeamIndex < p2TeamIndex) {
                return -1;
            } else if (p1TeamIndex > p2TeamIndex) {
                return 1;
            } else {
                int p1PosIndex = POSITIONS_GRADATION.indexOf(p1
                      .getPosition());
                int p2PosIndex = POSITIONS_GRADATION.indexOf(p2
                      .getPosition());

                if (p1PosIndex < p2PosIndex) {
                  return -1;
                } else if (p1PosIndex > p2PosIndex) {
                  return 1;
                } else {
                  if (p1.getPlayerNum() > p2.getPlayerNum()) {
                      return 1;
                  } else if (p1.getPlayerNum() < p2.getPlayerNum()) {
                      return -1;
                  }
                }
```

```
            }

            return 0;
          }
      });

      return allPlayers;
    }

    public Player getPlayer() {
      if (currPlayer == null) {
          initPlayerData();
      }

      return currPlayer;
    }

    private void initPlayerData() {
      String playerId = FacesSupport.getRequestValue(PLAYER_ID_PARAM_NAME);
      if (playerId != null) {
          System.out.println("--Fetching from DB player ID");
          this.currPlayer = playerDao.findById(new Integer(playerId), false);
      }
    }
}
```

UserSession.java

```
package com.grasshopper.web.bean;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.Authentication;
import org.springframework.security.context.SecurityContextHolder;

import com.grasshopper.dao.AccountDao;
import com.grasshopper.entity.Account;

public class UserSession {
    private Log logger = LogFactory.getLog(UserSession.class);

    @Autowired
    private AccountDao accountDao;

    private Account account;

    public UserSession() {
    }

    public Account getAccount() {
      if (account == null) {
          lazyInit();
      }
      return account;
    }
```

```
    public void setAccount(Account account) {
      this.account = account;
    }

    /**
     * Lazily fetch user information
     */
    private void lazyInit() {
      Authentication aa = SecurityContextHolder.getContext()
            .getAuthentication();

      String username = aa.getName();
      String password = (String) aa.getCredentials();

      if (username != null && password != null)
      {
          account = accountDao.findAccount(username, password);

          if (logger.isInfoEnabled())
          {
            if (account != null)
                {
                  logger.info("Successfully saved in session account data for
user: "+username);
                }
                else
                {
                  logger.info("Can't find in DB account data for user:
"+username);
                }
          }
      }

    }
}
```

FacesSupport.java

```
package com.grasshopper.web.helper;

import java.text.MessageFormat;
import java.util.Locale;
import java.util.ResourceBundle;

import javax.faces.application.FacesMessage;
import javax.faces.component.UIViewRoot;
import javax.faces.context.ExternalContext;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpServletRequest;

import org.richfaces.component.html.HtmlModalPanel;

import com.grasshopper.exception.GrasshopperUIRuntimeException;

public class FacesSupport {
    private static final String bundleBaseName = "com.grasshopper.i18n_en";
```

```
    public static final void showWhenRendered(FacesContext jsf,
          String modalPanelId, boolean showWhenRendered) {
      UIViewRoot viewRoot = jsf.getViewRoot();
      if (viewRoot != null) {
          Object modalPanel = viewRoot.findComponent(modalPanelId);
          if (modalPanel != null) {
            ((HtmlModalPanel) modalPanel)
                  .setShowWhenRendered(showWhenRendered);
          }
      }
    }

    public void addInfo(String controlId, String messageKey, Object... args)
{
      addMessage(controlId, FacesMessage.SEVERITY_INFO, messageKey, args);
    }

    public void addError(String controlId, String messageKey, Object... args)
{
      addMessage(controlId, FacesMessage.SEVERITY_ERROR, messageKey, args);
    }

    public void addMessage(String controlId,
          javax.faces.application.FacesMessage.Severity sev,
          String messageKey, Object... args) {
      FacesContext jsf = FacesContext.getCurrentInstance();
      ExternalContext extCtxt = jsf.getExternalContext();
      jsf.addMessage(controlId, new FacesMessage(sev, formatMessage(
            messageKey, extCtxt.getRequestLocale(), args), null));
    }

    public String formatMessage(String messageKey, Locale locale,
          Object... args) {
      return MessageFormat.format(getMessage(messageKey, locale), args);
    }

    public String getMessage(String messageKey, Locale locale) {
      ResourceBundle i18n = getBundle(locale);
      try {
          return i18n.getString(messageKey);
      } catch (java.util.MissingResourceException mre) {
          throw new GrasshopperUIRuntimeException(mre);
      }
    }

    public ResourceBundle getBundle(Locale locale) {
      return ResourceBundle.getBundle(bundleBaseName, locale);
    }

    public static String getRequestValue(String key) {
      HttpServletRequest request = (HttpServletRequest) FacesContext
            .getCurrentInstance().getExternalContext().getRequest();
      return request.getParameter(key);
    }

    public static Object getRequestMapValue(String key) {
```

```
      return getExternalContext().getRequestMap().get(key);
    }

    public static void setRequestMapValue(String key, Object value) {
      getExternalContext().getRequestMap().put(key, value);
    }

    public static Object getSessionMapValue(String key) {
      return getExternalContext().getSessionMap().get(key);
    }

    public static void setSessionMapValue(String key, Object value) {
      getExternalContext().getSessionMap().put(key, value);
    }

    public static ExternalContext getExternalContext() {
      return FacesContext.getCurrentInstance().getExternalContext();
    }
}
```

Tools

AutoAnnotationSessionFactoryBean.java

```
package com.grasshopper.tools;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.Set;

import javax.persistence.Entity;

import org.springframework.beans.factory.config.BeanDefinition;
import
org.springframework.context.annotation.ClassPathScanningCandidateComponentPro
vider;
import org.springframework.core.type.filter.AnnotationTypeFilter;
import
org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean;
import org.springframework.util.ClassUtils;

public class AutoAnnotationSessionFactoryBean extends
AnnotationSessionFactoryBean {

    private String[] basePackages;
    private List<String> excludedClasses = new ArrayList<String>();
    private ClassLoader beanClassLoader;

    public void setBasePackage(String basePackage) {
        this.basePackages = new String[] { basePackage };
    }

    public void setBasePackages(String[] basePackages) {
        this.basePackages = basePackages;
    }
```

```
    public void setExcludedClasses(List<String> excludedClasses) {
        this.excludedClasses = excludedClasses;
    }

    public void setBeanClassLoader(ClassLoader beanClassLoader) {
        this.beanClassLoader = beanClassLoader;
    }

    public void afterPropertiesSet() throws Exception {
        Collection<Class<?>> entities = new ArrayList<Class<?>>();
        ClassPathScanningCandidateComponentProvider scanner =
createScanner();
        for (String basePackage : basePackages) {
            findEntities(scanner, entities, basePackage);
        }
        setAnnotatedClasses(entities.toArray(new Class<?>[entities.size()]));
        setAnnotatedPackages(basePackages);
        super.afterPropertiesSet();
    }

    private ClassPathScanningCandidateComponentProvider createScanner() {
        ClassPathScanningCandidateComponentProvider scanner =
                new ClassPathScanningCandidateComponentProvider(false);
        scanner.addIncludeFilter(new AnnotationTypeFilter(Entity.class));
        return scanner;
    }

    private void findEntities(ClassPathScanningCandidateComponentProvider
scanner,
            Collection<Class<?>> entities, String basePackage) {
        Set<BeanDefinition> annotatedClasses =
scanner.findCandidateComponents(basePackage);
        for (BeanDefinition bd : annotatedClasses) {
            String className = bd.getBeanClassName();
            Class<?> type = ClassUtils.resolveClassName(className,
beanClassLoader);
            if (excludedClasses == null ||
!excludedClasses.contains(type.getName())) {
                entities.add(type);
            }
        }
    }
}
```

Utilities

DateUtils.java

```
package com.grasshopper.utils;

import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
```

```
import java.util.HashMap;
import java.util.Map;
import java.util.TimeZone;

import org.apache.commons.lang.StringUtils;
import org.apache.commons.lang.Validate;

public final class DateUtils {
    public static final String DATE_FORMAT_ONLY_DATE = "MMMMM, dd";

    private static final TimeZone defaultTimezone;

    private static Map<String, DateFormat> DATE_FORMAT_MAP = new
HashMap<String, DateFormat>();
    static {
      DATE_FORMAT_MAP.put(DATE_FORMAT_ONLY_DATE, new SimpleDateFormat(
            DATE_FORMAT_ONLY_DATE));
      defaultTimezone = Calendar.getInstance().getTimeZone();
    }

    private DateUtils() {
    }

    public static TimeZone getDefaultTimeZone() {
      return defaultTimezone;
    }

    /**
     * Formats date
     *
     * @param date
     *            date to be formatted as string
     * @param pattern
     *            date format
     *
     * @return string representation of input date
     */
    public static String formatDate(Date date, String pattern) {
      Validate.notNull(date);
      String result = null;
      DateFormat df = DATE_FORMAT_MAP.get(pattern);
      if (df != null) {
          synchronized (df) {
            result = df.format(date);
          }
      } else {
          result = new SimpleDateFormat(pattern).format(date);
      }
      return result;
    }

    /**
     * Parses string representation of date.
     *
     * @param dateStr
     *            string representation of date
     * @param pattern
```

```
     *            date format to parse
     * @return standard date object
     */
    public static Date parseDate(String dateStr, String pattern)
          throws ParseException {
      if (StringUtils.isEmpty(dateStr)) {
          return null;
      }

      Date result = null;
      DateFormat df = DATE_FORMAT_MAP.get(pattern);
      if (df != null) {
          synchronized (df) {
            result = df.parse(dateStr);
          }
      } else {
          result = new SimpleDateFormat(pattern).parse(dateStr);
      }
      return result;
    }

    public static String formatDateForUI( Date date, String pattern )
    {

        return ( date == null ) ? "" : formatDate(date, pattern);
    }
}
```

PropertiesLoader.java

```
package com.grasshopper.utils;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;

import org.apache.commons.lang.Validate;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import com.grasshopper.exception.PropertiesLoaderRuntimeException;

public final class PropertiesLoader {
    private static final Log logger =
LogFactory.getLog(PropertiesLoader.class);
    private static final String SUFFIX = ".properties";

    /**
     * Load properties from CLASSPATH
     *
     * @param path
     *            path to the properties file
     * @return instance of the <code>java.util.Properties</code> class
     * @throws IOException
     *             if some problems with reading properties file occured
```

```
 */
public static Properties loadProperties(String path) {
  Validate.notEmpty(path, "null input: name");

  if (path.startsWith("/")) {
      path = path.substring(1);
  }

  if (!path.endsWith(SUFFIX)) {
      path = path + SUFFIX;
  }

  final InputStream is = PropertiesLoader.class.getClassLoader()
        .getResourceAsStream(path);
  Validate.notNull(is, "Could not load [" + path + "]"
        + " as a resource bundle");

  final Properties properties = new Properties();
  loadProperties(properties, is);

  return properties;
}

public static Properties loadProperties(String dir, String path) {
  Validate.notEmpty(path, "null input: file name");
  if (path.startsWith("/")) {
      path = path.substring(1);
  }

  if (!path.endsWith(SUFFIX)) {
      path = path + SUFFIX;
  }

  if (dir != null && !dir.endsWith("/")) {
      dir = dir + "/";
  }

  InputStream is = null;
  try {
      if (dir != null) { // Loading as external resource
        is = new FileInputStream(dir + path);
      } else { // Loading as classpath resource
        is = new FileInputStream(PropertiesLoader.class
              .getClassLoader().getResource(path).getPath());
      }
  } catch (Exception e) {
      if (logger.isWarnEnabled() == true) {
        String message = "Could not load properties from [" + dir
              + path + "].";
        message = message + " properties will be loaded from classpath";
        logger.warn(message);
      }
      return loadProperties(path);
  }

  Validate.notNull(is, "Could not load [" + dir + path + "]");
```

```
  final Properties properties = new Properties();
  loadProperties(properties, is);

  return properties;

}

/**
 * Load resource by given relative path
 *
 * @param resourcePath
 *            relative path from CLASSPATH
 * @return instance of <code>java.io.InputStream</code> class
 */
public static InputStream loadResource(final String resourcePath) {
  Validate.notEmpty(resourcePath, "null input: resource path");

  final InputStream is = PropertiesLoader.class.getClassLoader()
        .getResourceAsStream(resourcePath);
  if (is == null) {
      if (logger.isWarnEnabled()) {
        logger.warn("Could not load resource [" + resourcePath + "]");
      }
  }

  return is;
}

/**
 * Verifies whether resource(properties file as input argument is also
 * applicable) exists
 *
 * @param resourcePath
 *            relative path from CLASSPATH
 * @return true if resource exists, false otherwise
 */
public static boolean verifyResourceExist(final String resourcePath) {
  Validate.notEmpty(resourcePath, "null input: resource path");
  final InputStream is = PropertiesLoader.class.getClassLoader()
        .getResourceAsStream(resourcePath);

  boolean resourceExists = true;
  if (is == null) {
      resourceExists = false;
  }
  return resourceExists;
}

private static void loadProperties(final Properties properties,
      final InputStream is) {
  try {
      properties.load(is);
  } catch (IOException e) {
      throw new PropertiesLoaderRuntimeException(e);
  }
}
```

```
    private PropertiesLoader() {
    }

}
```

Exceptions

GrasshopperDaoException.java

```
package com.grasshopper.exception;

public class GrasshopperDaoException extends GrasshopperException {
    private static final long serialVersionUID = -8746978852477845899L;

    public GrasshopperDaoException() {
      super();
    }

    public GrasshopperDaoException(String message, Throwable cause) {
      super(message, cause);
    }

    public GrasshopperDaoException(String message) {
      super(message);
    }

    public GrasshopperDaoException(Throwable cause) {
      super(cause);
    }
}
```

GrasshopperException.java

```
package com.grasshopper.exception;

public class GrasshopperException extends Exception {
    private static final long serialVersionUID = -2228227743375144307L;

    public GrasshopperException() {
      super();
    }

    public GrasshopperException(String message, Throwable cause) {
      super(message, cause);
    }

    public GrasshopperException(String message) {
      super(message);
    }

    public GrasshopperException(Throwable cause) {
      super(cause);
    }
}
```

GrasshopperRuntimeException.java

```
package com.grasshopper.exception;

public class GrasshopperRuntimeException extends RuntimeException {
    private static final long serialVersionUID = -4305650709439961968L;

    public GrasshopperRuntimeException() {
      super();
    }

    public GrasshopperRuntimeException(String message, Throwable cause) {
      super(message, cause);
    }

    public GrasshopperRuntimeException(String message) {
      super(message);
    }

    public GrasshopperRuntimeException(Throwable cause) {
      super(cause);
    }
}
```

GrasshopperUIRuntimeException

```
package com.grasshopper.exception;

public class GrasshopperUIRuntimeException extends RuntimeException{
    private static final long serialVersionUID = 6288474201639244727L;

    public GrasshopperUIRuntimeException() {
      super();
    }

    public GrasshopperUIRuntimeException(String message, Throwable cause) {
      super(message, cause);
    }

    public GrasshopperUIRuntimeException(String message) {
      super(message);
    }

    public GrasshopperUIRuntimeException(Throwable cause) {
      super(cause);
    }
}
```

PropertiesLoaderRuntimeException

```
package com.grasshopper.exception;

public class PropertiesLoaderRuntimeException extends
GrasshopperRuntimeException{
    private static final long serialVersionUID = -1104608008459646032L;
```

```
    public PropertiesLoaderRuntimeException() {
        super();
    }

    public PropertiesLoaderRuntimeException(String message) {
        super(message);
    }

    public PropertiesLoaderRuntimeException(Throwable cause) {
        super(cause);
    }

    public PropertiesLoaderRuntimeException(String message, Throwable cause)
{
        super(message, cause);
    }
}
```

Constants

PayMethod.java

```
package com.grasshopper.constant;

public enum PayMethod {
    CASH("Cash"), CREDIT_CARD("Credit Card");
    private PayMethod(String payMethod) {
      this.payMethod = payMethod;
    }

    private String payMethod;

    public String getPayMethod() {
      return payMethod;
    }
}
package com.grasshopper.constant;
```

Position.java

```
public enum Position {

    DEFENDER("Defender"), MIDFILDER("Midfilder"), STRIKER("Striker"),
GOALKEEPER(
          "Goalkeeper");

    Position(String position) {
      this.position = position;
    }

    private String position;

    /**
     * @return the position
     */
    public String getPosition() {
```

```
      return position;
    }
}
```

Role.java

```
package com.grasshopper.constant;

public enum Role {
    ROLE_USER, ROLE_ADMIN
}
```

Team.java

```
package com.grasshopper.constant;

public enum Team {
    FIRST_TEAM("First team"), RESERVE_TEAM("Reserve team");

    Team(String team) {
      this.team = team;
    }

    private String team;

    public String getTeam() {
        return team;
    }
}
```

Grasshopper Configuration

GrasshopperConfig.java

```
package com.grasshopper.config;

import java.util.Properties;

import com.grasshopper.utils.PropertiesLoader;

public final class GrasshopperConfig {
    private static final Properties grasshopperConfigProperties =
PropertiesLoader
          .loadProperties("grasshopper_config.properties");

    private GrasshopperConfig() {
    }

    public static Integer getUpcomingMatchesMaxAmount() {
      Integer defaultResult = 5;
      Integer configResult = new Integer((String) grasshopperConfigProperties
            .getProperty("shown.upcoming.matches.amount"));
      return configResult == null ? defaultResult : configResult;
    }

    public static Integer getTicketsForSaleAmount() {
```

```
      Integer defaultAmount = 4;
      Integer configResult = new Integer((String) grasshopperConfigProperties
            .getProperty("tickets.for.sale.amount"));
      return configResult == null ? defaultAmount : configResult;
    }

    public static Integer getShownMatchesAmount() {
      Integer defaultShownMatchesAmount = 10;
      Integer configResult = new Integer(grasshopperConfigProperties
            .getProperty("shown.matches.amount"));
      return configResult == null ? defaultShownMatchesAmount : configResult;
    }
}
```

Presentation Layer

i18n_en.properties

```
#Index page
index_title=Grasshoppers football club
toolbar_separator= |
login=Sign In
logout=Sign Out
register=Register
greeting=Hello, {0}. Please, login to confirm registration
index_page__left_msg=Please sign-in to book the tickets for the upcoming
matches
players=Players
matches=Matches

#Register page
login_username=Username
login_password=Password
login_password_confirm=Confirm Password
login_email=Email
login_fullname=Full Name
login_error=Invalid credentials!
remember_me=Remember me on this computer
firstName=First Name
middleName=Middle Name
lastName=Last Name
address=Address
city=City
county=County
country=Country
phone=Phone
username_taken=This username is already registered. Please choose another
username
signup=Sign Up
password_not_confirmed=Password and password confirm values do not match!
ok=OK
help=Help
cancel=Cancel

#home.jsp page
tickets_amount=Choose number of tickets:
```

```
no_available_tickets=Booking failure: no desired number of tickets is
available for this match
no_tickets_at_all=No tickets available for this match
book_msg=To book the tickets, select the upcoming match first >>
continue=Continue

#booking.jsp page
successfull_booking=Tickets were booked successfully
booking_details=Booking details:
tickets_amount=Number of tickets:
one_ticket_price=One ticket price
total_ticket_price=Pay total
section_name=Section
click_msg=Please contact a booking manager to pay for the tickets. Click
here_msg=here
back_home_msg= to go back to the home page.
booking_id=Booking Reference Number

std_pass_err_msg=Your password is not secure enough! Please choose a mixed-
case password that is at least 8 characters long and contains at least one
digit.
exception_msg=Exception while processing your request occured. Please,
contact system administrator.

homepage=Home

team_story_indent1=The Grasshoppers student football team was founded by the
first class enrolled at the Green University - that is why its history is
closely connected with the history of the University itself. During the past
decade, Grasshoppers has grown from an after-class activity for a small group
of enthusiasts into a full-fledged football club with professional coaches
and highly capable sportsman-players, students of our University.
team_story_indent2=Last year, Green University finished renovating the campus
stadium, which is now one of the major attractions of Long-Grass city. The
renovated stadium now hosts all of the home matches of Grasshopper team and
attracts not only students, university stuff and parents, but also outside
visitors and tourists.
team_story_indent3=The co-founder and later the coach of the team, Harry
McGovern, happened to be a member of the Department of Biology and Natural
Sciences. It was Harry who suggested Grasshoppers as a name for the team- the
bright, out-of-the-box name that reflects the spirit of our University and
its philosophy- active and strong, at the same time careful and responsible
towards nature. It was voted by the majority of the players and coaches and
throughout these years the Green University football club proudly bears the
Grasshoppers name.

#credit_card.jsp
book=Book
credit_card_title=Your credit card info
first_name_on_card=First name on the credit card
last_name_on_card=Last name on the credit card
card_number=Number
cvc=CVC
expiryMonth=Expiry month
expiryYear=Expiry year
credit_card_type=Credit card type
maestro_credit_card=Maestro
```

```
master_card_credit_card=Master Card
visa_credit_card=Visa
american_express_credit_card=American Express
credit_card_prompt=Fill in your credit card information:

#players.jsp
players_title=Grasshopper football club players
player_number=Number
player=Player
position=Position
team=Team
playerTooltip=Click to get more player information
#player.jspf
player_info=Player information
birthDate=Birth Date
height=Height
weight=Weight
goals=Goals
gamesPlayed=Games Played
yellowCards=Yellow Cards
redCards=Red Cards
otherInfo=Other information


#matches.jsp
opposition=Opposition
match_date=Match Date
score=Score
venue=Venue
not_played_game_msg=Not played yet
analysis_title=Click to see match analysis
#match.jspf
match_analysis=Match Analysis
```

auth.jspf

```
<%@page import="com.grasshopper.web.bean.AuthorizationBean"%>
<rich:modalPanel id="loginPanel" width="320" height="240">
      <f:facet name="header">
            <h:outputText value="#{i18n.login}" />
      </f:facet>
      <f:facet name="controls">
            <h:panelGroup>
                  <h:graphicImage value="img/close.gif" styleClass="hidelink"
                        id="hideLoginLink" />
                  <rich:componentControl for="loginPanel"
attachTo="hideLoginLink"
                        operation="hide" event="onclick" />
            </h:panelGroup>
      </f:facet>
      <h:form id="loginForm" prependId="false">
            <h:panelGrid columns="2" columnClasses="col0,col1"
cellpadding="5"
                  footerClass="loginFooter">
                  <h:outputText value="#{i18n.login_username}" />
                  <h:inputText id="j_username" />
```

```
                  <h:outputText value="#{i18n.login_password}" />
                  <h:inputText id="j_password" />
                  <h:outputText value=" " />
                  <h:panelGroup>
                        <h:selectBooleanCheckbox
id="_spring_security_remember_me" />
                        <h:outputText value="#{i18n.remember_me}" />
                  </h:panelGroup>
                  <f:facet name="footer">
                        <h:panelGroup>
                              <h:commandButton type="submit" id="loginBtn"
                                    action="#{authBean.doLogin}"
value="Login" />
                              <rich:spacer width="5" height="1" />
                              <rich:message for="loginBtn">
                                    <f:facet name="errorMarker">
                                          <h:graphicImage
value="img/forbidden.gif" />
                                    </f:facet>
                              </rich:message>
                        </h:panelGroup>
                  </f:facet>
            </h:panelGrid>
      </h:form>
</rich:modalPanel>


<rich:modalPanel id="registerPanel" width="450" height="500">
      <f:facet name="header">
            <h:outputText value="#{i18n.signup}" />
      </f:facet>
      <f:facet name="controls">
            <h:panelGroup>
                  <h:graphicImage value="img/close.gif" styleClass="hidelink"
                        id="hidelink" />
                  <rich:componentControl for="registerPanel"
attachTo="hidelink"
                        operation="hide" event="onclick" />
            </h:panelGroup>
      </f:facet>
      <a:form id="registerForm">
            <h:panelGrid columns="1" footerClass="regFooter">
                  <h:panelGrid columns="2" columnClasses="col0,col1"
cellpadding="5">
                        <h:panelGroup>
                              <h:outputText value="#{i18n.login_username}" />
                              <h:graphicImage value="img/rqd.gif" />
                        </h:panelGroup>
                        <h:panelGroup>
                              <h:inputText label="#{i18n.login_username}"
id="userName"
                                    required="true"
value="#{authBean.username}" size="12"
                                    maxlength="16">
                                    <rich:beanValidator />
                                    <a:support event="onblur"
reRender="userName" ajaxSingle="true" />
```

```
                              </h:inputText>
                              <rich:message for="userName">
                                    <f:facet name="errorMarker">
                                          <h:graphicImage
value="img/error.gif" />
                                    </f:facet>
                              </rich:message>
                        </h:panelGroup>

                        <h:panelGroup>
                              <h:outputText value="#{i18n.login_password}" />
                              <h:graphicImage value="img/rqd.gif" />
                        </h:panelGroup>
                        <h:panelGroup>
                              <h:inputSecret label="#{i18n.login_password}"
id="password"
                                    required="true"
value="#{authBean.password}" size="12"
                                    maxlength="12">
                                    <rich:beanValidator />
                              </h:inputSecret>
                              <rich:message for="j_password">
                                    <f:facet name="errorMarker">
                                          <h:graphicImage
value="img/error.gif" />
                                    </f:facet>
                              </rich:message>
                        </h:panelGroup>

                        <h:panelGroup>
                              <h:outputText
value="#{i18n.login_password_confirm}" />
                              <h:graphicImage value="img/rqd.gif" />
                        </h:panelGroup>
                        <h:panelGroup>
                              <h:inputSecret
label="#{i18n.login_password_confirm}"
                                    id="passwordConfirm"
value="#{authBean.confirmPassword}"
                                    required="true" size="12" maxlength="12">
                              </h:inputSecret>
                        </h:panelGroup>

                        <h:panelGroup>
                              <h:outputText value="#{i18n.login_email}" />
                              <h:graphicImage value="img/rqd.gif" />
                        </h:panelGroup>
                        <h:panelGroup>
                              <h:inputText label="#{i18n.login_email}"
id="email" required="true"
                                    value="#{authBean.account.user.email}"
size="22" maxlength="60">
                                    <rich:beanValidator />
                              </h:inputText>
                              <rich:message for="email">
                                    <f:facet name="errorMarker">
```

```
                                    <h:graphicImage
value="img/error.gif" />
                                </f:facet>
                            </rich:message>
                        </h:panelGroup>

                        <h:panelGroup>
                            <h:outputText value="#{i18n.firstName}" />
                            <h:graphicImage value="img/rqd.gif" />
                        </h:panelGroup>
                        <h:panelGroup>
                            <h:inputText label="#{i18n.firstName}"
id="firstname"
                                required="true"
value="#{authBean.account.user.firstName}"
                                size="12">
                                <rich:beanValidator />
                            </h:inputText>
                            <rich:message for="firstname">
                                <f:facet name="errorMarker">
                                    <h:graphicImage
value="img/error.gif" />
                                </f:facet>
                            </rich:message>
                        </h:panelGroup>

                        <h:panelGroup>
                            <h:outputText value="#{i18n.lastName}" />
                            <h:graphicImage value="img/rqd.gif" />
                        </h:panelGroup>
                        <h:panelGroup>
                            <h:inputText label="#{i18n.lastName}"
id="lastname" required="true"
                                value="#{authBean.account.user.lastName}"
size="30">
                                <rich:beanValidator />
                            </h:inputText>
                            <rich:message for="lastname">
                                <f:facet name="errorMarker">
                                    <h:graphicImage
value="img/error.gif" />
                                </f:facet>
                            </rich:message>
                        </h:panelGroup>

                        <h:panelGroup>
                            <h:outputText value="#{i18n.address}" />
                            <h:graphicImage value="img/rqd.gif" />
                        </h:panelGroup>
                        <h:panelGroup>
                            <h:inputText label="#{i18n.address}"
id="address" required="true"
                                value="#{authBean.account.user.address}"
size="30">
                                <rich:beanValidator />
                            </h:inputText>
                            <rich:message for="address">
```

```
                                    <f:facet name="errorMarker">
                                          <h:graphicImage
value="img/error.gif" />
                                    </f:facet>
                              </rich:message>
                        </h:panelGroup>

                        <h:panelGroup>
                              <h:outputText value="#{i18n.city}" />
                              <h:graphicImage value="img/rqd.gif" />
                        </h:panelGroup>
                        <h:panelGroup>
                              <h:inputText label="#{i18n.city}" id="city"
required="true"
                                    value="#{authBean.account.user.city}"
size="30">
                                    <rich:beanValidator />
                              </h:inputText>
                              <rich:message for="city">
                                    <f:facet name="errorMarker">
                                          <h:graphicImage
value="img/error.gif" />
                                    </f:facet>
                              </rich:message>
                        </h:panelGroup>

                        <h:panelGroup>
                              <h:outputText value="#{i18n.county}" />
                              <h:graphicImage value="img/rqd.gif" />
                        </h:panelGroup>
                        <h:panelGroup>
                              <h:inputText label="#{i18n.county}" id="county"
required="true"
                                    value="#{authBean.account.user.county}"
size="30">
                                    <rich:beanValidator />
                              </h:inputText>
                              <rich:message for="county">
                                    <f:facet name="errorMarker">
                                          <h:graphicImage
value="img/error.gif" />
                                    </f:facet>
                              </rich:message>
                        </h:panelGroup>

                        <h:panelGroup>
                              <h:outputText value="#{i18n.country}" />
                              <h:graphicImage value="img/rqd.gif" />
                        </h:panelGroup>
                        <h:panelGroup>
                              <h:inputText label="#{i18n.country}"
id="country" required="true"
                                    value="#{authBean.account.user.country}"
size="30">
                                    <rich:beanValidator />
                              </h:inputText>
                              <rich:message for="country">
```

```
                                    <f:facet name="errorMarker">
                                          <h:graphicImage
value="img/error.gif" />
                                    </f:facet>
                              </rich:message>
                        </h:panelGroup>

                        <h:panelGroup>
                              <h:outputText value="#{i18n.phone}" />
                              <h:graphicImage value="img/rqd.gif" />
                        </h:panelGroup>
                        <h:panelGroup>
                              <h:inputText label="#{i18n.phone}" id="phone"
required="true"
                                    value="#{authBean.account.user.phone}"
size="30">
                                    <rich:beanValidator />
                              </h:inputText>
                              <rich:message for="phone">
                                    <f:facet name="errorMarker">
                                          <h:graphicImage
value="img/error.gif" />
                                    </f:facet>
                              </rich:message>
                        </h:panelGroup>

                  </h:panelGrid>
                  <f:facet name="footer">
                        <h:panelGroup>
                              <a:commandButton id="registerButton"
value="#{i18n.register}"
                                    action="#{authBean.doRegister}" />
                        </h:panelGroup>
                  </f:facet>
            </h:panelGrid>
            <rich:message for="registerButton">
                  <f:facet name="errorMarker">
                        <h:graphicImage value="img/error.gif" />
                  </f:facet>
            </rich:message>
      </a:form>
</rich:modalPanel>
<%
    AuthorizationBean.setLoginPanelVisibility();
                                    %>
```

booking.jsp

```
<%@page contentType="text/html;charset=UTF-8" language="java"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@taglib uri="http://richfaces.org/a4j" prefix="a"%>
<%@taglib uri="http://richfaces.org/rich" prefix="rich"%>
```

```
<%@taglib uri="http://www.springframework.org/security/tags"
      prefix="sec"%>
<html>
  <f:view>
    <head>
      <title><h:outputText value="#{i18n.index_title}" /></title>
      <a:loadStyle src="styles.css" />
    </head>
    <body>
      <%@include file="auth.jspf"%>
      <table width="80%" border="0" cellspacing="0" cellpadding="0">
        <tr valign="top">
        <td><h:graphicImage value="img/banner.jpg" styleClass="header"
        /></td>
        </tr>
      </table>
      <table width="80%">
        <tr>
        <td align="left">
            <h:graphicImage value="img/leftPane_clean.jpg" /></td>
            <rich:dataOrderedList var="match_info"
               value="#{infoBean.upcomingMatches}"
               styleClass="upcoming-matches-link"
               type="disc">
            <h:outputLink value="home.jsp">
            <h:outputText value="#{match_info.matchInfo}" />
            <f:param name="matchId" value="#{match_info.id}" />
            </h:outputLink>
            </rich:dataOrderedList>
            <h:panelGroup styleClass="leftMsg">
            <h:outputText value="#{i18n.click_msg} " styleClass="infoText"/>
            <h:outputLink value="/grasshopper/faces/home.jsp"
               styleClass="infoText">
            <h:outputText value="#{i18n.here_msg}"/>
            </h:outputLink>
            <h:outputText value=" #{i18n.back_home_msg}"
               styleClass="infoText"/>
            </h:panelGroup>
        <td valign="top">
      <table width="100%">
        <tr valign="top">
        <td align="right" valign="top">
            <h:panelGroup>
            <h:outputText value="You are logged in as "
               styleClass="infoText"/>
            <sec:authentication property="name" />
            <h:outputText value="#{i18n.toolbar_separator}   "
               styleClass="infoText"/>
            <h:outputLink
value="/grasshopper/faces/j_spring_security_logout">
            <h:outputText styleClass="toolbarLink" value="#{i18n.logout}" />
            </h:outputLink>
            </h:panelGroup></td>
        </tr>
        <tr>
      <table id="booking-info-table">
      <thead>
```

```
        <tr><th><h:outputText value="#{i18n.successfull_booking}"
        /></th></tr>
        <tr><th><h:outputText value="#{i18n.booking_details}"/></th></tr>
      </thead>
      <tbody>
        <tr>
            <td><h:outputText value="#{i18n.booking_id}"/></td>
            <td><h:outputText value="#{bookingDetails.bookingId}"/></td>
        </tr>
        <tr>
            <td><h:outputText value="#{i18n.tickets_amount}"/></td>
            <td><h:outputText value="#{bookingDetails.ticketsAmount}"/></td>
        </tr>
        <tr>
            <td><h:outputText value="#{i18n.one_ticket_price}"/></td>
            <td><h:outputText value="#{bookingDetails.oneTicketPrice}"/></td>
        </tr>
        <tr>
            <td><h:outputText value="#{i18n.total_ticket_price}"/></td>
            <td><h:outputText
            value="#{bookingDetails.totalTicketPrice}"/></td>
        </tr>
        <tr>
            <td><h:outputText value="#{i18n.section_name}"/></td>
            <td><h:outputText value="#{bookingDetails.sectionName}"/></td>
        </tr>
      </tbody>
      </table></tr>
     </table></td>

    </tr></table>

   </body>
  </f:view>
</html>
```

credit_card.jsp

```
<%@page contentType="text/html;charset=UTF-8" language="java"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@taglib uri="http://richfaces.org/a4j" prefix="a"%>
<%@taglib uri="http://richfaces.org/rich" prefix="rich"%>
<%@taglib uri="http://www.springframework.org/security/tags"
      prefix="sec"%>
<html>
<f:view>
      <head>
      <title><h:outputText value="#{i18n.credit_card_title}" /></title>
      <a:loadStyle src="styles.css" />
      </head>
      <body>
      <%@include file="auth.jspf"%>
      <table width="80%" border="0" cellspacing="0" cellpadding="0">
            <tr valign="top">
```

```
                  <td><h:graphicImage value="img/banner.jpg"
                  styleClass="header" /></td>
            </tr>
</table>
<table width="80%">
      <tr>
            <td align="left">
                  <h:graphicImage value="img/leftPane_clean.jpg" />
            </td>
            <rich:dataOrderedList var="match_info"
                  value="#{infoBean.upcomingMatches}" styleClass="upcoming-
                  matches-link"
                  type="disc">
                  <h:outputLink value="home.jsp">
                        <h:outputText value="#{match_info.matchInfo}" />
                        <f:param name="matchId" value="#{match_info.id}" />
                  </h:outputLink>
            </rich:dataOrderedList>
            <h:outputText value="#{i18n.book_msg}" styleClass="leftMsg" />
            <td valign="top">
                  <table width="100%">
                        <tr valign="top">
                              <td align="right" valign="top">
                                    <h:panelGroup>
                                          <h:outputText value="You are logged
                                          in as "
                                                styleClass="infoText" />
                                           <sec:authentication property="name"
/>

                                          <h:outputText
                                          value="#{i18n.toolbar_separator}   "
                                                styleClass="infoText" />
                                          <h:outputLink
                                          value="/grasshopper/faces/j_spring_s
                                          ecurity_logout">
                                                <h:outputText
                                          styleClass="toolbarLink"
                                          value="#{i18n.logout}" />
                                           </h:outputLink>
                                    </h:panelGroup>
                              </td>
                        </tr>
                        <tr>
                              <td>
                                    <h:form id="creditCardAndBookingForm">
                                          <h:panelGrid columns="1"
                                          footerClass="creditCardFooter">
                                                <h:outputText
                                                value="#{i18n.credit_card_prom
                                                pt}"

                                                 styleClass="creditCardPrompt"
/>

                                                <h:panelGrid columns="2"
                                                columnClasses="col0,col1"
                                                      cellpadding="5">
                                                      <h:panelGroup>
```

```
                                                <h:outputText
                                          value="#{i18n.first_name_on_ca
                                          rd}"

                                           styleClass="infoText" />
                                                <h:graphicImage
                                          value="img/rqd.gif" />
                                                </h:panelGroup>
                                                <h:panelGroup>
                                                     <h:inputText
id="fnameOnCard"

      value="#{bookingBean.creditCard.fnameOnCard}" required="true"

      maxlength="25" />


<h:message for="fnameOnCard" styleClass="errorMsg">
      <f:facet name="errorMarker">
            <h:graphicImage value="img/error.gif" />
      </f:facet>
</h:message>
</h:panelGroup>
<h:panelGroup>
 <h:outputText value="#{i18n.last_name_on_card}"
    styleClass="infoText" />
    <h:graphicImage value="img/rqd.gif" />
            </h:panelGroup>
            <h:panelGroup>
 <h:inputText id="lnameOnCard" value="#{bookingBean.creditCard.lnameOnCard}"
               required="true" maxlength="25" />
            <h:message for="lnameOnCard" styleClass="errorMsg">
 <f:facet name="errorMarker">
    <h:graphicImage value="img/error.gif" />
 </f:facet>
</h:message>
</h:panelGroup>
<h:panelGroup>
      <h:outputText value="#{i18n.credit_card_type}"
                              styleClass="infoText" />
      <h:graphicImage value="img/rqd.gif" />
</h:panelGroup>

<h:panelGroup>

      <h:selectOneMenu value="#{bookingBean.creditCard.type}">
      <f:selectItem itemValue="#{i18n.maestro_credit_card}" />
      <f:selectItem itemValue="#{i18n.master_card_credit_card}" />
      <f:selectItem itemValue="#{i18n.visa_credit_card}" />
      </h:selectOneMenu>
      </h:panelGroup>

<h:panelGroup>
      <h:outputText value="#{i18n.card_number}" styleClass="infoText" />
      <h:graphicImage value="img/rqd.gif" />
</h:panelGroup>
```

```
<h:panelGroup>
      <h:inputText id="number" value="#{bookingBean.creditCard.number}"
            required="true" maxlength="16" />
      <h:message for="number" styleClass="errorMsg">
            <f:facet name="errorMarker">
                  <h:graphicImage value="img/error.gif" />
            </f:facet>
      </h:message>
</h:panelGroup>

<h:panelGroup>
      <h:outputText value="#{i18n.cvc}" styleClass="infoText" />
      <h:graphicImage value="img/rqd.gif" />
</h:panelGroup>

<h:panelGroup>
      <h:inputText id="cvc" value="#{bookingBean.creditCard.cvc}"
            required="true" maxlength="3" />
      <h:message for="cvc" styleClass="errorMsg">
            <f:facet name="errorMarker">
            <h:graphicImage value="img/error.gif" />
            </f:facet>
      </h:message>
</h:panelGroup>

<h:panelGroup>
      <h:outputText value="#{i18n.expiryMonth}" styleClass="infoText" />
      <h:graphicImage value="img/rqd.gif" />
</h:panelGroup>

<h:panelGroup>
      <h:selectOneMenu value="#{bookingBean.creditCard.expiryMonth}">
      <f:selectItem itemLabel="January" itemValue="1" />
      <f:selectItem itemLabel="February" itemValue="2" />
      <f:selectItem itemLabel="March" itemValue="3" />
      <f:selectItem itemLabel="April" itemValue="4" />
      <f:selectItem itemLabel="May" itemValue="5" />
      <f:selectItem itemLabel="June" itemValue="6" />
      <f:selectItem itemLabel="July" itemValue="7" />
      <f:selectItem itemLabel="August" itemValue="8" />
      <f:selectItem itemLabel="September" itemValue="9" />
      <f:selectItem itemLabel="October" itemValue="10" />
      <f:selectItem itemLabel="November" itemValue="11" />
      <f:selectItem itemLabel="December" itemValue="12" />
      </h:selectOneMenu>
</h:panelGroup>

<h:panelGroup>
      <h:outputText value="#{i18n.expiryYear}" styleClass="infoText" />
      <h:graphicImage value="img/rqd.gif" />
</h:panelGroup>

<h:panelGroup>
      <h:inputText id="expiryYear"
      value="#{bookingBean.creditCard.expiryYear}"
            required="true">
      <f:convertDateTime type="date" pattern="yyyy" />
```

```
      </h:inputText>
      <h:message for="expiryYear" styleClass="errorMsg">
      <f:facet name="errorMarker">
            <h:graphicImage value="img/error.gif" />
      </f:facet>
      </h:message>
</h:panelGroup>

<h:panelGroup>
      <h:commandButton id="bookCommandButton" type="submit"

            action="#{bookingBean.fillCreditCardDataAndBookTickets}"

            value="#{i18n.book}">
      </h:commandButton>
      <br/>
      <h:message for="bookCommandButton" styleClass="errorMsg">
      <f:facet name="errorMarker">

            <h:graphicImage value="img/error.gif" />
      </f:facet>
      </h:message>
</h:panelGroup>
</h:panelGrid>
</h:panelGrid>
</h:form></td>
</tr>
</table>
</td>

</tr>
      </table>

      </body>
</f:view>
</html>
```

home.jsp

```
<%@page contentType="text/html;charset=UTF-8" language="java"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@taglib uri="http://richfaces.org/a4j" prefix="a"%>
<%@taglib uri="http://richfaces.org/rich" prefix="rich"%>
<%@taglib uri="http://www.springframework.org/security/tags"
      prefix="sec"%>
<html>
<f:view>
      <head>
      <title><h:outputText value="#{i18n.index_title}" /></title>
      <a:loadStyle src="styles.css" />
      </head>
      <body>
      <%@include file="auth.jspf"%>
      <table width="80%" border="0" cellspacing="0" cellpadding="0">
```

```
            <tr valign="top">
                  <td><h:graphicImage value="img/banner.jpg"
styleClass="header" /></td>
            </tr>
      </table>
      <table width="80%">
            <tr>
                  <td align="left"><h:graphicImage
value="img/leftPane_clean.jpg" /></td>
                  <rich:dataOrderedList var="match_info"
                        value="#{infoBean.upcomingMatches}"
                        styleClass="upcoming-matches-link" type="disc">
                        <h:outputLink value="home.jsp">
                              <h:outputText value="#{match_info.matchInfo}"
/>
                              <f:param name="matchId"
value="#{match_info.id}" />
                        </h:outputLink>
                  </rich:dataOrderedList>
                  <h:outputText value="#{i18n.book_msg}" styleClass="leftMsg"
/>
                  <td valign="top">
                  <table width="100%">
                        <tr valign="top">
                              <td align="right" valign="top"><h:panelGroup>
                                    <h:outputText value="You are logged in as
" styleClass="infoText" />
                                    <sec:authentication property="name" />
                                    <h:outputText
value="#{i18n.toolbar_separator}   "
                                          styleClass="infoText" />
                                    <h:outputLink
value="/grasshopper/faces/j_spring_security_logout">
                                          <h:outputText
styleClass="toolbarLink" value="#{i18n.logout}" />
                                    </h:outputLink>
                              </h:panelGroup></td>
                        </tr>
                        <tr>
                              <td><h:form id="bookingForm">
                                    <h:graphicImage
value="img/seatingPlan.jpg" styleClass="seatingPlan" />
<c:choose>
      <c:when test="${bookingBean.availableTicketsForMatch}">
            <h:panelGrid columns="10"
                  rendered="#{bookingBean.sectionsAndPricesRendered}">
                  <h:selectOneRadio id="sectionsOneRadio"
layout="pageDirection"
                        value="#{bookingBean.sectionChoose}"
styleClass="sectionsChoose">
                        <f:selectItems
value="#{bookingBean.sectionsAndPrices}" />
                  </h:selectOneRadio>
                  <h:outputText value=">>" styleClass="bookSeparator" />
                  <h:panelGroup>
                        <h:outputLabel id="ticketsAmountLabel"
for="ticketsAmountOneMenu"
```

```
                              value="#{i18n.tickets_amount}"
styleClass="ticketsAmount" />
                        <h:selectOneMenu id="ticketsAmountOneMenu"
                              value="#{bookingBean.ticketsAmountChoose}">
                              <f:selectItems
value="#{bookingBean.ticketsAmount}" />
                        </h:selectOneMenu>
                  </h:panelGroup>
                  <h:outputText value=">>" styleClass="bookSeparator" />
                  <h:commandButton id="bookCommandButton" type="submit"
                        action="#{bookingBean.fillBookingData}"
value="#{i18n.continue}">
                  </h:commandButton>
            </h:panelGrid>
      </c:when>
      <c:otherwise>
            <br />
            <h:outputText value="#{i18n.no_tickets_at_all}"
                  styleClass="noTickets" />
      </c:otherwise>
</c:choose>

</h:form>
</td>
</tr>
</table>
</td>

</tr>
</table>

</body>
</f:view>
</html>
```

index.html

```
<html>
<head>
<title>Welcome to Grasshopper FC site</title>
<meta content="0; url=faces/index.jsp" http-equiv="refresh">
</head>
<body onLoad="javascript:location.replace('faces/index.jsp')"></body>
</html>
```

index.jsp

```
<%@page contentType="text/html;charset=UTF-8" language="java"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@taglib uri="http://richfaces.org/a4j" prefix="a"%>
<%@taglib uri="http://richfaces.org/rich" prefix="rich"%>
<%@taglib uri="http://www.springframework.org/security/tags"
      prefix="sec"%>
<html>
<f:view>
```

```
      <head>
      <title><h:outputText value="#{i18n.index_title}" /></title>
      <a:loadStyle src="styles.css" />
      </head>
      <body>
      <%@include file="auth.jspf"%>
      <table width="80%" border="0" cellspacing="0" cellpadding="0">
            <tr valign="top">
                  <td><h:graphicImage value="img/banner.jpg"
styleClass="header" /></td>
            </tr>
      </table>
      <table>
            <tr valign="top">
                  <td align="left"><h:graphicImage
value="img/leftPane_clean.jpg"
                        styleClass="styleLeft" /></td>
                  <rich:dataOrderedList var="match_info"
                        value="#{infoBean.upcomingMatches}"
                        styleClass="upcoming-matches-text" type="disc">
                        <h:outputText value="#{match_info.matchInfo}" />
                  </rich:dataOrderedList>
                  <h:outputText value="#{i18n.index_page__left_msg}"
                        styleClass="leftMsg" />
                  <td valign="top">
                  <table width="100%">
                        <tr valign="top">
                              <td align="right" valign="top"><h:panelGroup>
                                    <c:if test="${!empty param.username}">
                                          <h:outputFormat
value="#{i18n.greeting}" styleClass="infoLabel">
                                                <f:param
value="#{userSession.account.username}" />
                                          </h:outputFormat>
                                          <h:outputText
value="#{i18n.toolbar_separator}" escape="false"
                                                styleClass="infoLabel" />
                                    </c:if>
                                    <h:outputLink
value="/grasshopper/faces/index.jsp">
                                          <h:outputText
value="#{i18n.homepage}" styleClass="homeLink"/>
                                    </h:outputLink>
                                    <h:outputText value="|| "
escape="false" styleClass="doubleSeparator"/>
                                    <h:outputLink value="#" id="loginLink">
                                          <h:outputText styleClass="authLink"
value="#{i18n.login}" />
                                          <rich:componentControl
for="loginPanel" attachTo="loginLink"
                                                operation="show"
event="onclick" />
                                    </h:outputLink>
                                    <h:outputText
value="#{i18n.toolbar_separator}" escape="false" />
                                    <h:outputLink value="#"
id="registerLink">
```

```
                                        <h:outputText styleClass="authLink"
value="#{i18n.register}" />
                                        <rich:componentControl
for="registerPanel"
                                              attachTo="registerLink"
operation="show" event="onclick" />
                                  </h:outputLink>

                            </h:panelGroup>
                            </td>
                                  <td align="right">
                                  <h:outputText value="|| "
escape="false" styleClass="doubleSeparator"/>
                                  <h:outputLink
value="/grasshopper/faces/players.jsp" >
                                        <h:outputText
value="#{i18n.players}" styleClass="pagesLink"/>
                                  </h:outputLink>
                                  <h:outputText
value="#{i18n.toolbar_separator}" escape="false" />
                                  <h:outputLink
value="/grasshopper/faces/matches.jsp" >
                                        <h:outputText
value="#{i18n.matches}" styleClass="pagesLink"/>
                                  </h:outputLink>
                            </td>
                      </tr>

                      <tr>
                            <td align="center"><h:panelGroup
styleClass="teamStory">
                                  <br />
                                  <h:outputText value="Grasshoppers
Football Team"
                                        styleClass="storyTitle" />
                                  <p><h:outputText
value="#{i18n.team_story_indent1}"
                                        styleClass="story" /></p>
                                  <p><h:outputText
value="#{i18n.team_story_indent2}"
                                        styleClass="story" /></p>
                                  <p><h:outputText
value="#{i18n.team_story_indent3}"
                                        styleClass="story" /></p>
                            </h:panelGroup>
                      </tr>
                </table>
                </td>
          </tr>
      </table>
      </body>
</f:view>
</html>
```

logout.jsp

```
<%
    try {
            session.invalidate();
    } catch (Exception ignore) {
    }
    response.sendRedirect("/grasshopper/faces/index.jsp");
    return;
                                        %>
```

match.jspf

```
<%@page import="com.grasshopper.web.bean.MatchesBean"%>

<rich:modalPanel id="matchPanel" width="500" height="350">
      <f:facet name="header">
            <h:outputText value="#{i18n.match_analysis}" />
      </f:facet>
      <f:facet name="controls">
            <h:panelGroup>
                  <h:graphicImage value="img/close.gif" styleClass="hidelink"
                        id="hidelinkPlayer" />
                  <rich:componentControl for="matchPanel"
attachTo="hidelinkPlayer"
                        operation="hide" event="onclick" />
            </h:panelGroup>
      </f:facet>
      <h:panelGrid id="matchInfo" columns="1" footerClass="regFooter">
            <h:outputText value="#{matchesBean.analysis}"
styleClass="infoText"/>
      </h:panelGrid>
</rich:modalPanel>
```

matches.jsp

```
<%@page contentType="text/html;charset=UTF-8" language="java"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@taglib uri="http://richfaces.org/a4j" prefix="a"%>
<%@taglib uri="http://richfaces.org/rich" prefix="rich"%>
<%@taglib uri="http://www.springframework.org/security/tags"
      prefix="sec"%>
<html>
<f:view>
      <head>
      <title><h:outputText value="#{i18n.players_title}" /></title>
      <a:loadStyle src="styles.css" />
      </head>
      <body>
      <%@include file="auth.jspf"%>
      <%@include file="match.jspf"%>
      <table width="80%" border="0" cellspacing="0" cellpadding="0">
```

```
            <tr valign="top">
                  <td><h:graphicImage value="img/banner.jpg"
styleClass="header" /></td>
            </tr>
      </table>
      <table>
            <tr valign="top">
                  <td align="left"><h:graphicImage
value="img/leftPane_clean.jpg"
                        styleClass="styleLeft" /></td>
                  <rich:dataOrderedList var="match_info"
                        value="#{infoBean.upcomingMatches}"
                        styleClass="upcoming-matches-text" type="disc">
                        <h:outputText value="#{match_info.matchInfo}" />
                  </rich:dataOrderedList>
                  <h:outputText value="#{i18n.index_page__left_msg}"
                        styleClass="leftMsg" />
                  <td valign="top">
                  <table width="100%">
                        <tr valign="top">
                              <td align="left" valign="top"><h:panelGroup>
                                    <h:outputLink
value="/grasshopper/faces/index.jsp">
                                          <h:outputText
value="#{i18n.homepage}" styleClass="homeLink" />
                                    </h:outputLink>
                                    <h:outputText value="|| "
escape="false"
                                          styleClass="doubleSeparator" />
                                    <h:outputLink value="#" id="loginLink">
                                          <h:outputText styleClass="authLink"
value="#{i18n.login}" />
<rich:componentControl for="loginPanel" attachTo="loginLink"
      operation="show" event="onclick" />
</h:outputLink>
<h:outputText value="#{i18n.toolbar_separator}" escape="false" />
<h:outputLink value="#" id="registerLink">
      <h:outputText styleClass="authLink" value="#{i18n.register}" />
      <rich:componentControl for="registerPanel"
            attachTo="registerLink" operation="show" event="onclick" />
</h:outputLink>
<h:outputText value="|| " escape="false" styleClass="doubleSeparator" />
<h:outputLink value="/grasshopper/faces/players.jsp">
      <h:outputText value="#{i18n.players}" styleClass="pagesLink" />
</h:outputLink>
<h:outputText value="#{i18n.toolbar_separator}" escape="false" />
<h:outputLink value="/grasshopper/faces/matches.jsp">
      <h:outputText value="#{i18n.matches}" styleClass="pagesLink" />
</h:outputLink>
</h:panelGroup>
</td>
</tr>
<tr>
      <table id="rounded-corner">
            <thead>
                  <tr>
                        <th scope="col" class="rounded-first-record">
```

```
                              <h:outputText value="#{i18n.opposition}" />
                        </th>
                        <th scope="col" class="rounded-col1">
                              <h:outputText value="#{i18n.match_date}" />
                        </th>
                        <th scope="col" class="rounded-col2">
                              <h:outputText value="#{i18n.score}" />
                        </th>
                        <th scope="col" class="rounded-col3">
                              <h:outputText value="#{i18n.venue}" />
                        </th>
                  </tr>
            </thead>
            <tbody>
                  <a:form>
                        <a:repeat value="#{matchesBean.matches}" var="match">
                              <tr>
                                    <td>
                                          <h:outputText
value="#{match.opposition}" />
                                    </td>

                                                                  <td>
      <h:outputText value="#{match.matchDate}" />
</td>
<td>
      <a:commandLink id="score" reRender="matchInfo">
            <h:outputText
                  rendered="#{match.scoreGrassh != null && match.scoreOppos
!= null}"
                  value="#{match.scoreGrassh} : #{match.scoreOppos}"
title="#{i18n.analysis_title}" />
            <a:actionparam id="matchId" name="matchId"
value="#{match.matchId}"
                  assignTo="#{matchesBean.matchId}" />
            <rich:componentControl for="matchPanel"
                  attachTo="score" operation="show" event="onclick" />
      </a:commandLink>
      <h:outputText
            rendered="#{match.scoreGrassh == null || match.scoreOppos ==
null}"
            value="#{i18n.not_played_game_msg}" />
</td>
<td>
      <h:outputText value="#{match.venue}" />
</td>
</tr>
            </a:repeat>
                              </a:form>
                                 </tbody>
                               </table>
                             </tr>
                        </table>
                  </td>
                </tr>
            </table>
       </body>
```

```
   </f:view>
                                        </html>
```

player.jspf

```
<%@page import="com.grasshopper.web.bean.PlayersBean"%>

<rich:modalPanel id="playerPanel" width="610" height="470">
      <f:facet name="header">
            <h:outputText value="#{i18n.player_info}" />
      </f:facet>
      <f:facet name="controls">
            <h:panelGroup>
                  <h:graphicImage value="img/close.gif" styleClass="hidelink"
                        id="hidelinkPlayer" />
                  <rich:componentControl for="playerPanel"
attachTo="hidelinkPlayer"
                        operation="hide" event="onclick" />
            </h:panelGroup>
      </f:facet>
      <h:panelGrid id="playerInfo" columns="1" footerClass="regFooter">
            <h:panelGrid columns="2" columnClasses="col0,col1"
cellpadding="5">
                  <h:outputText value="#{i18n.firstName} : "
styleClass="playerInfoLabel"/>
                  <h:outputText value="#{playersBean.player.firstName}"
styleClass="playerInfoVal"/>

                  <h:outputText value="#{i18n.lastName} : "
styleClass="playerInfoLabel"/>
                  <h:outputText value="#{playersBean.player.lastName}"
styleClass="playerInfoVal"/>

                  <h:outputText value="#{i18n.birthDate} : "
styleClass="playerInfoLabel"/>
                  <h:outputText value="#{playersBean.player.birthDate}"
styleClass="playerInfoVal"/>

                  <h:outputText value="#{i18n.height} : "
styleClass="playerInfoLabel"/>
                  <h:outputText value="#{playersBean.player.height}"
styleClass="playerInfoVal"/>

                  <h:outputText value="#{i18n.weight} : "
styleClass="playerInfoLabel"/>
                  <h:outputText value="#{playersBean.player.weight}"
styleClass="playerInfoVal"/>

                  <h:outputText value="#{i18n.position} : "
styleClass="playerInfoLabel"/>
                  <h:outputText value="#{playersBean.player.position}"
styleClass="playerInfoVal"/>

                  <h:outputText value="#{i18n.player_number} : "
styleClass="playerInfoLabel"/>
```

```
                  <h:outputText value="#{playersBean.player.playerNum}"
styleClass="playerInfoVal"/>

                  <h:outputText value="#{i18n.goals} : "
styleClass="playerInfoLabel"/>
                  <h:outputText value="#{playersBean.player.goals}"
styleClass="playerInfoVal"/>

                  <h:outputText value="#{i18n.gamesPlayed} : "
styleClass="playerInfoLabel"/>
                  <h:outputText value="#{playersBean.player.gamesPlayed}"
styleClass="playerInfoVal"/>

                  <h:outputText value="#{i18n.yellowCards} : "
styleClass="playerInfoLabel"/>
                  <h:outputText value="#{playersBean.player.yellowCards}"
styleClass="playerInfoVal"/>

                  <h:outputText value="#{i18n.redCards} : "
styleClass="playerInfoLabel"/>
                  <h:outputText value="#{playersBean.player.redCards}"
styleClass="playerInfoVal"/>

                  <h:outputText value="#{i18n.otherInfo} : "
styleClass="playerInfoLabel"/>
                  <h:outputText value="#{playersBean.player.notes}"
styleClass="playerInfoVal"/>
            </h:panelGrid>
      </h:panelGrid>
</rich:modalPanel>
```

players.jsp

```
<%@page contentType="text/html;charset=UTF-8" language="java"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@taglib uri="http://richfaces.org/a4j" prefix="a"%>
<%@taglib uri="http://richfaces.org/rich" prefix="rich"%>
<%@taglib uri="http://www.springframework.org/security/tags"
      prefix="sec"%>
<html>
<f:view>
      <head>
      <title><h:outputText value="#{i18n.players_title}" /></title>
      <a:loadStyle src="styles.css" />
      </head>
      <body>
      <%@include file="auth.jspf"%>
      <%@include file="player.jspf"%>
      <table width="80%" border="0" cellspacing="0" cellpadding="0">
            <tr valign="top">
                  <td><h:graphicImage value="img/banner.jpg"
styleClass="header" /></td>
            </tr>
```

```
      </table>
      <table>
            <tr valign="top">
                  <td align="left"><h:graphicImage
value="img/leftPane_clean.jpg"
                        styleClass="styleLeft" /></td>
                  <rich:dataOrderedList var="match_info"
                        value="#{infoBean.upcomingMatches}"
                        styleClass="upcoming-matches-text" type="disc">
                        <h:outputText value="#{match_info.matchInfo}" />
                  </rich:dataOrderedList>
                  <h:outputText value="#{i18n.index_page__left_msg}"
                        styleClass="leftMsg" />
                  <td valign="top">
                  <table width="100%">
                        <tr valign="top">
      <td align="left" valign="top">
            <h:panelGroup>
                  <h:outputLink value="/grasshopper/faces/index.jsp">
                        <h:outputText value="#{i18n.homepage}"
styleClass="homeLink" />
                  </h:outputLink>
                  <h:outputText value="|| " escape="false"
styleClass="doubleSeparator" />
                  <h:outputLink value="#" id="loginLink">
                        <h:outputText styleClass="authLink"
value="#{i18n.login}" />
                        <rich:componentControl for="loginPanel"
                              attachTo="loginLink" operation="show"
event="onclick" />
                  </h:outputLink>
                  <h:outputText value="#{i18n.toolbar_separator}"
escape="false" />
                  <h:outputLink value="#" id="registerLink">
                        <h:outputText styleClass="authLink"
value="#{i18n.register}" />
                        <rich:componentControl for="registerPanel"
                              attachTo="registerLink" operation="show"
event="onclick" />
                  </h:outputLink>
                  <h:outputText value="|| " escape="false"
styleClass="doubleSeparator" />
                  <h:outputLink value="/grasshopper/faces/players.jsp">
                        <h:outputText value="#{i18n.players}"
styleClass="pagesLink" />
                  </h:outputLink>
                  <h:outputText value="#{i18n.toolbar_separator}"
escape="false" />
                  <h:outputLink value="/grasshopper/faces/matches.jsp">
                        <h:outputText value="#{i18n.matches}"
styleClass="pagesLink" />
                  </h:outputLink>
            </h:panelGroup>
      </td>
</tr>
<tr>
      <table id="rounded-corner">
```

```
            <thead>
                  <tr>
                        <th scope="col" class="rounded-first-record">
                              <h:outputText value="#{i18n.player_number}" />
                        </th>
                        <th scope="col" class="rounded-col1">
                              <h:outputText value="#{i18n.player}" />
                        </th>
                        <th scope="col" class="rounded-col2">
                              <h:outputText value="#{i18n.position}" />
                        </th>
                        <th scope="col" class="rounded-col3">
                              <h:outputText

                                    value="#{i18n.team}" /></th>
                                    </tr>
                                    </thead>
                                    <tbody>
                                          <a:form>
                                                <a:repeat
value="#{playersBean.allPlayers}" var="player">
                                                      <tr>
                                                            <td><h:outputText
value="#{player.playerNum}" /></td>

      <td><a:commandLink id="firstName" reRender="playerInfo">

      <h:outputText value="#{player.firstName} #{player.lastName}"

      title="#{i18n.playerTooltip}" />

      <a:actionparam id="playerId" name="playerId"

      value="#{player.playerId}"

      assignTo="#{playersBean.playerId}" />

      <rich:componentControl for="playerPanel" attachTo="firstName"

      operation="show" event="onclick" />

      </a:commandLink></td>
                                                            <td><h:outputText
value="#{player.position}" /></td>
                                                            <td><h:outputText
value="#{player.team}" /></td>
                                                      </tr>
                                                </a:repeat>
                                          </a:form>
                                    </tbody>
                              </table>
                        </tr>
                  </table>
                  </td>
            </tr>
      </table>
      </body>
```

```
</f:view>
                                    </html>
```

register.jsp

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
      xmlns:a4j="http://richfaces.org/a4j"
      xmlns:rich="http://richfaces.org/rich"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html" version="2.0">
      <jsp:directive.page language="java"
            contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-
8859-1" />
      <jsp:text>
            <![CDATA[ <?xml version="1.0" encoding="ISO-8859-1" ?> ]]>
      </jsp:text>
      <jsp:text>
            <![CDATA[ <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
]]>
      </jsp:text>
      <html xmlns="http://www.w3.org/1999/xhtml">
      <head>
      <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"
/>
      <title>Registration page</title>
      </head>
      <body>
      <f:view>
            <h:form>
                  <rich:panel>
                        <f:facet name="header">
                              <h:outputText value="User Info:" />
                        </f:facet>
                        <h:panelGrid columns="3">
                              <h:outputText value="Email:" />
                              <h:inputText value="#{registerBean.email}"
id="email"
                                    required="true">
                                    <f:validateLength minimum="3"
maximum="12" />
                                    <rich:ajaxValidator event="onblur" />
                              </h:inputText>
                              <rich:message for="email" />
                              <h:inputText value="#{registerBean.name}"
id="name" />
                        </h:panelGrid>
                  </rich:panel>
            </h:form>
      </f:view>
      </body>
      </html>
</jsp:root>
```

styles.css

```
html {
  overflow-y: scroll;
}

body {
  font-size: 12px;
  margin: 0px;
}

a img {
  border: none;
}

.homeLink{font: 12px tahoma,verdana,arial; text-align: left; text-decoration:
none; color:#000033; }
.authLink { font: 12px tahoma,verdana,arial; text-align: left; text-
decoration: none; color: #000066; }
.pagesLink{ font: 12px tahoma,verdana,arial; text-align: left; text-
decoration: none; color: #000099; }
.doubleSeparator{ color: red; }

.header {width: 100%}
.styleLeft{}
.rich-message-marker img { padding-left:5px; padding-right:5px; }
.rich-message-label { font: 11px tahoma,verdana,arial; color:red; padding-
left:5px; }
.rich-messages-marker img { padding-left:5px; padding-right:5px; }
.rich-messages-label { font: 11px tahoma,verdana,arial; color:red; padding-
left:5px; }
.catalogNode { font: 11px tahoma,verdana,arial; }
.col0 { font: 11px tahoma,verdana,arial; }
.col1 { font: 11px tahoma,verdana,arial; }
.col2 { font: 11px tahoma,verdana,arial; }
.loginFooter { font: 11px tahoma,verdana,arial; text-align: center; }
.regFooter { font: 11px tahoma,verdana,arial; text-align: center; }
.authHeader { font: 11px tahoma,verdana,arial; text-align: left; }
.tooltip { border-width:2px; padding:5px; }
.tooltip-text { width:350px; height:80px; cursor:arrow; border-width:2px;
text-align:center; display: table-cell; vertical-align: middle; }
.barsearch { height: 17px; width: 200px; }
.prefFooter { font: 11px tahoma,verdana,arial; text-align: left; }
.mainCanvas { vertical-align: top; width: 980px; height: 600px; }
.canvasHdr { padding: 4px; }
.rich-inplace-select-list-decoration { background-color: #ecf4fe; font: 11px
tahoma,verdana,arial; }
.infoLabel { font: 12px tahoma,verdana,arial; color:#000000; padding-
left:7px; }

.upcoming-matches-text{position: absolute; left: -1%; top: 185px; font: 11px
bolder tahoma,verdana,arial; color: #000066; max-width: 180px;}
.upcoming-matches-link{position: absolute; left: -1%; top: 185px; font: 12px
bolder tahoma,verdana,arial; max-width: 170px;}
.teamStory{position: absolute; text-align: left;padding-right:300px }

.errorMsg{ font: 11px tahoma,verdana,arial; color:red; padding-left:5px; }
```

```
.noTickets{ font: 15px bold tahoma,verdana,arial; color: red; padding-left:
30px;}

.storyTitle{ font: bold 16px tahoma,verdana,arial; color:black;}
.story{font: 14px tahoma,verdana,arial; color:black; }

.leftMsg{ font: 11.5px tahoma,verdana,arial; color: #000066; position:
absolute; left: 2%; top: 410px; max-width: 170px;}
.seatingPlan{width: 60%;}
.sectionsChoose{font: 12px tahoma,verdana,arial; color: #000066}
.ticketsAmount{ font: 12px tahoma,verdana,arial; color: #000066}
.bookSeparator{font: 20px tahoma,verdana,arial; color: #CC6600}

.infoText{font: 11.5px tahoma,verdana,arial; color: #000066}

.creditCardFooter{font: 11px tahoma,verdana,arial; text-align: left; }

.creditCardPrompt{font: 15px bold tahoma,verdana,arial; color: #990000}

.playerInfoLabel{font: 12px tahoma,verdana,arial; color: #663300}
.playerInfoVal{font: 12px tahoma,verdana,arial; color: #000066}

#booking-info-table{
font-family: "Lucida Sans Unicode", "Lucida Grande", Sans-Serif;
      font-size: 12px;
      background: #fff;
      margin: 45px;
      width: 480px;
      border-collapse: collapse;
      text-align: left;
      color: #039;
}
#booking-info-table th
{
      font-size: 14px;
      font-weight: normal;
      color: #039;
      padding: 10px 8px;
      border-bottom: 2px solid #6678b1;
}
#booking-info-table td
{
      color: #669;
      padding: 9px 8px 0px 8px;
}
#booking-info-table tbody tr:hover td
{
      color: #009;
}

#rounded-corner
{
      font-family: "Lucida Sans Unicode", "Lucida Grande", Sans-Serif;
      font-size: 12px;
      margin: 45px;
      width: 480px;
      text-align: left;
```

```
	border-collapse: collapse;
}
#rounded-corner thead th.rounded-first-record
{
	background: #b9c9fe url('table-images/left.png') left -1px no-repeat;
}
#rounded-corner thead th.rounded-col4
{
	background: #b9c9fe url('table-images/right.png') right -1px no-repeat;
}
#rounded-corner th
{
	padding: 8px;
	font-weight: normal;
	font-size: 13px;
	color: #039;
	background: #b9c9fe;
}
#rounded-corner td
{
	padding: 8px;
	background: #e8edff;
	border-top: 1px solid #fff;
	color: #669;
}
#rounded-corner tfoot td.rounded-foot-left
{
	background: #e8edff url('table-images/botleft.png') left bottom no-
repeat;
}
#rounded-corner tfoot td.rounded-foot-right
{
	background: #e8edff url('table-images/botright.png') right bottom no-
repeat;
}
#rounded-corner tbody tr:hover td
{
	background: #d0dafd;
}
```

temp.jspf

```
<div style="text-align: width: 100%;left; height: 100%;">
<table>
	<tr>
		<td><h:graphicImage value="img/banner.jpg"
styleClass="style1"/></td>
	</tr>
</table>
<table>
	<tr>
		<td valign="left"><h:graphicImage value="img/leftPane_clean.jpg"
styleClass="style1"/></td>
	</tr>
</table>
</div>
```

Build project Ant task

build.properties

```
project.name=grasshopper
project.context=grasshopper

tomcat.engine=Catalina
tomcat.host=localhost

tomcat.home=c:/tomcat/apache-tomcat-6.0.20
tomcat.deploy=${tomcat.home}/webapps
tomcat.context.dir=${tomcat.home}/conf/${tomcat.engine}/${tomcat.host}

tomcat.lib.dir = ${tomcat.home}/lib
```

build.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<project name="Grasshopper" basedir="." default="dist_clean">
      <property file="build.properties" />

      <property name="build.dir" location="build" />
      <property name="dist.dir" location="dist" />
      <property name="bin.dir" location="bin" />
      <property name="war.exploded.dir"
location="${build.dir}/${project.name}.war" />

      <property name="lib.dir" location="lib" />
      <property name="tomcat.lib.dir" location="${tomcat.lib.dir}" />
      <property name="resources.dir" location="resources" />
      <property name="resources.common.dir"
location="${resources.dir}/common" />
      <property name="resources.mapping.dir"
location="${resources.dir}/mapping" />
      <property name="resource.mapping.hibernate.dir"
location="${resources.mapping.dir}/hibernate" />
      <property name="war.resources.dir" location="${resources.dir}/web" />
      <property name="src.dir" location="src/java" />
      <property name="view.dir" location="view" />
      <property name="img.dir" location="${view.dir}/img" />
      <property name="tomcat.deploy.dir" location="${tomcat.deploy}" />
      <property name="hibernate.mapping.dir"
location="${resources.mapping.dir}/hibernate" />

      <property name="war.exploded.dir.tmp"
location="${dist.dir}/${project.name}.war.tmp" />
      <property name="tmp.build.dir" location="${build.dir}/classes" />
      <property name="tmp.build.dir.test" location="${build.dir}/classes-
test" />
      <property name="tmp.build.dir.unit.test"
location="${tmp.build.dir.test}/classes-unit-test" />
```

```
      <property name="tmp.build.dir.integration.test"
location="${tmp.build.dir.test}/classes-integration-test" />
      <property name="tmp.build.dir.load.test"
location="${tmp.build.dir.test}/classes-load-test" />

      <property name="test.dir" location="test" />
      <property name="test.load.dir" location="${test.dir}/load" />
      <property name="src.dir.common.test" location="${test.dir}/common/java"
/>
      <property name="src.dir.unit.test" location="${test.dir}/unit/java" />
      <property name="src.dir.integration.test"
location="${test.dir}/integration/java" />
      <property name="src.dir.load.test" location="${test.load.dir}/java" />

      <property name="resources.dir.unit.test"
location="${test.dir}/unit/resources" />
      <property name="resources.dir.integration.test"
location="${test.dir}/integration/resources" />
      <property name="resources.dir.load.test"
location="${test.load.dir}/resources" />
      <property name="resources.dir.common.test"
location="${test.dir}/common/resources" />


      <path id="compile.classpath">
            <fileset dir="${lib.dir}">
                  <include name="*.jar" />
                  <include name="web/*.jar" />
            </fileset>
            <fileset dir="${tomcat.lib.dir}">
                  <include name="*.jar" />
            </fileset>
      </path>

      <path id="compile.test.classpath">
            <path refid="compile.classpath" />
            <fileset dir="${lib.dir}">
                  <include name="test/*.jar" />
            </fileset>
            <pathelement location="${tmp.build.dir}" />
      </path>

      <target name="clean" description="Cleans project by removing
compilation results for all application">
            <echo message="Deleting build directories" />
            <delete dir="${build.dir}" />
            <delete dir="${dist.dir}" />
            <delete dir="${tmp.build.dir}" />
      </target>

      <target name="compile" description="Compiles source code">
            <mkdir dir="${tmp.build.dir}" />
            <javac srcdir="${src.dir}" destdir="${tmp.build.dir}"
classpathref="compile.classpath" debug="true" />
      </target>
```

```
	<target name="compile_tests" depends="compile" description="Compiles
tests source code">
		<mkdir dir="${tmp.build.dir.test}" />
		<mkdir dir="${tmp.build.dir.unit.test}" />
		<mkdir dir="${tmp.build.dir.integration.test}" />
		<mkdir dir="${tmp.build.dir.load.test}" />

		<javac srcdir="${src.dir.common.test}:${src.dir.load.test}"
destdir="${tmp.build.dir.load.test}" classpathref="compile.test.classpath"
debug="true" />
		<javac srcdir="${src.dir.unit.test}"
destdir="${tmp.build.dir.unit.test}" classpathref="compile.test.classpath"
debug="true" />
		<javac
srcdir="${src.dir.common.test}:${src.dir.integration.test}"
destdir="${tmp.build.dir.integration.test}"
classpathref="compile.test.classpath" debug="true" />

	</target>

	<target name="build_prepare_war" description="Creates structure of
exploded WAR and fill it with not compiled content">
		<!-- Configure Apache Tomcat -->
		<!-- MySQL connector should be placed in Tomcat lib directory -->
		<property name="mysql.connector.lib" value="mysql-connector-java-
5.1.7-bin.jar" />
		<copy file="${lib.dir}/${mysql.connector.lib}"
todir="${tomcat.lib.dir}" />

		<copy todir="${war.exploded.dir}">
			<fileset dir="${resources.dir}/web" />
		</copy>

		<copy todir="${war.exploded.dir}/img">
			<fileset dir="${img.dir}">
				<include name="*.*" />
			</fileset>
		</copy>

		<!-- Copy Hibernate configuration files -->
		<copy todir="${war.exploded.dir}/WEB-INF/classes">
			<fileset dir="${hibernate.mapping.dir}">
				<include name="*.hbm.xml" />
			</fileset>
		</copy>

		<copy todir="${war.exploded.dir}/WEB-INF/lib">
			<fileset dir="${lib.dir}">
				<include name="*.jar" />
				<exclude name="${mysql.connector.lib}" />
			</fileset>
			<fileset dir="${lib.dir}/web">
				<include name="*.jar" />
			</fileset>
		</copy>

		<copy todir="${war.exploded.dir}">
```

```
                  <fileset dir="${view.dir}/pages">
                        <include name="*.*" />
                  </fileset>
            </copy>
            <copy todir="${war.exploded.dir}/WEB-INF/classes">
                  <fileset dir="${resources.dir}/common">
                        <include name="*.*" />
                  </fileset>
                  <fileset dir="${src.dir}"
includes="com/grasshopper/*.properties">
                  </fileset>
            </copy>
            <copy todir="${war.exploded.dir}/img">
                  <fileset dir="${view.dir}/img">
                        <include name="*.*" />
                  </fileset>
            </copy>
      </target>

      <target name="build_war" depends="build_prepare_war,compile"
description="Compiles source code and fill exploded WAR with compiled
content">
            <!-- Copy compiled classes to their destinations in exploded WAR
-->
            <copy todir="${war.exploded.dir}/WEB-INF/classes/">
                  <fileset dir="${tmp.build.dir}" includes="com/grasshopper/"
/>
                  <fileset dir="${tmp.build.dir}" includes="demo/" />
            </copy>
      </target>

      <target name="build_clean_war" depends="clean" description="Cleans,
compiles source code and fill exploded WAR with compiled content">
            <antcall target="build_war" />
      </target>

      <target name="dist" depends="build_war" description="Creates
distributable for web version of application">
            <mkdir dir="${dist.dir}" />
            <!-- Copy exploded WAR to temporary WAR directory -->
            <copy todir="${war.exploded.dir.tmp}">
                  <fileset dir="${war.exploded.dir}" />
            </copy>

            <!-- Create packed WEB archive from an exploded one -->
            <property name="dist.file"
location="${dist.dir}/${project.name}.war" />

            <jar basedir="${war.exploded.dir}" destfile="${dist.file}" />

            <copy file="${dist.file}" todir="${tomcat.deploy.dir}" />

            <!-- Delete temporary EAR directory -->
            <delete dir="${war.exploded.dir.tmp}" />
      </target>
```

```
	<target name="clean_tomcat" description="Clean tomcat deploy
directories">
		<echo message="Cleaning Tomcat directories" />
		<delete dir="${tomcat.deploy.dir}/${project.name}" />
	</target>

	<target name="dist_clean" depends="clean,clean_tomcat">
		<antcall target="dist" />
	</target>

	<target name="build_run" depends="dist_clean" description="Build
project, deploy it and run Apache Tomcat">
		<java jar="${tomcat.home}/bin/bootstrap.jar" fork="true">
			<jvmarg value="-Dcatalina.home=${tomcat.home}" />
		</java>
	</target>

	<target name="build_and_debug">
		<java jar="${tomcat.home}/bin/bootstrap.jar" fork="true">
			<jvmarg value="-Dcatalina.home=${tomcat.home}" />
			<jvmarg value="-Xdebug" />
			<jvmarg value="-
Xrunjdwp:transport=dt_socket,address=8000,server=y,suspend=n" />
		</java>

	</target>

	<target name="build_load_tests" depends="compile_tests"
description="Builds load tests">
		<property name="load.tests.temp.dir"
value="${build.dir}/load_tests.tmp" />
		<property name="load.tests.dir" location="${dist.dir}/load_tests"
/>
		<mkdir dir="${load.tests.temp.dir}" />
		<mkdir dir="${load.tests.dir}" />
		<mkdir dir="${load.tests.dir}/config" />

		<copy todir="${load.tests.temp.dir}">
			<fileset dir="${tmp.build.dir}">
				<include name="com/grasshopper/entity/**/*.class"/>
				<include name="com/grasshopper/dao/**/*.class" />
				<include name="com/grasshopper/orm/**/*.class" />
				<include name="com/grasshopper/exception/*.class" />
				<include name="com/grasshopper/constant/**/*.class"/>
				<include name="com/grasshopper/utils/**/*.class" />
				<include name="com/grasshopper/test/**/*.class" />
				<include name="com/grasshopper/config/**/*.class" />
			</fileset>
			<fileset dir="${tmp.build.dir.load.test}">
				<include name="com/grasshopper/**/*.class" />
			</fileset>
		</copy>

		<copy todir="${load.tests.dir}">
			<fileset dir="${bin.dir}/load_tests">
				<include name="*.*" />
			</fileset>
```

```
            </copy>

            <copy todir="${load.tests.dir}/lib">
                  <fileset dir="${lib.dir}">
                        <include name="*.jar" />
                  </fileset>
            </copy>

            <copy todir="${load.tests.dir}/config">
                  <fileset dir="${resources.common.dir}">
                        <include name="*.*" />
                        <exclude name="datasource.properties" />
                        <exclude name="applicationContext-ui.xml" />
                        <exclude name="applicationContext-security.xml" />
                        <exclude name="applicationContext-datasource-
jndi.xml"/>
                        <exclude name="applicationContext-service.xml"/>
                        <exclude name="log4j*.properties"/>
                  </fileset>
                  <fileset dir="${resources.dir.load.test}">
                        <include name="*.*" />
                  </fileset>
                  <fileset dir="${load.tests.dir}/config">
                        <include name="*.*" />
                  </fileset>
                  <fileset dir="${resources.dir.common.test}">
                        <include name="*.*" />
                  </fileset>
            </copy>
            <copy todir="${load.tests.dir}/config/mapping">
                  <fileset dir="${resource.mapping.hibernate.dir}">
                        <include name="*.*" />
                  </fileset>
            </copy>
            <copy todir="${load.tests.dir}/dataset-config">
                  <fileset dir="${test.load.dir}/config">
                        <include name="*.*" />
                  </fileset>
            </copy>

            <jar basedir="${load.tests.temp.dir}"
destfile="${load.tests.dir}/loadtests.jar" />
      </target>
</project>
```

Appendix H

Testing System

Testing CRUD Operations

SingleUserLoad.java

```
package com.grasshopper.dao.orm.singleuser;

import java.util.Set;

import com.grasshopper.entity.Matches;
import com.grasshopper.entity.Seat;
import com.grasshopper.entity.Section;
import com.grasshopper.entity.Ticket;
import com.grasshopper.entity.User;

public interface SingleUserLoad {
    /* Tests runner method */
    void executeTests() throws Exception;

    /* Load tests (business logic methods) */
    Set<Ticket> insertTickets(Matches match, Set<Seat> seats);
    void reserveTickets(Set<Ticket> tickets, User user, int bookingAmount);
    void deleteAllTickets();
    void deleteTickets(Set<Ticket> tickets);
    Integer getSoldTicketsCount(Matches match);
    Integer getOverallSeatsCount();

    /* Initialization methods */
    void cleanDB();
    User createTestUser();
    Set<Ticket> createTickets(Matches match, Set<Seat> seats);
    Set<Section> createSections(int amount);
    Set<Seat> createSeats(int amountOfRaws, int amountOfSeatsInRow,
          Set<Section> sections);
    Matches createMatch();
}
```

SingleUserLoadImpl.java

```
package com.grasshopper.dao.orm.singleuser;

import java.math.BigDecimal;
import java.util.Date;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Properties;
import java.util.Set;

import org.apache.commons.lang.StringUtils;
import org.apache.commons.lang.time.StopWatch;
import org.apache.commons.logging.Log;
```

```
import org.apache.commons.logging.LogFactory;
import org.springframework.beans.factory.annotation.Autowired;

import com.grasshopper.dao.BookingDao;
import com.grasshopper.dao.MatchesDao;
import com.grasshopper.dao.SeatDao;
import com.grasshopper.dao.SectionDao;
import com.grasshopper.dao.TicketDao;
import com.grasshopper.dao.UserDao;
import com.grasshopper.entity.Account;
import com.grasshopper.entity.Booking;
import com.grasshopper.entity.Matches;
import com.grasshopper.entity.Seat;
import com.grasshopper.entity.Section;
import com.grasshopper.entity.Ticket;
import com.grasshopper.entity.User;
import com.grasshopper.utils.PropertiesLoader;

public class SingleUserLoadImpl implements SingleUserLoad {
    private static Log logger = LogFactory.getLog(SingleUserLoadImpl.class);

    public static void setLoggerClass(Class<?> loggerClass) {
      logger = LogFactory.getLog(loggerClass);
    }

    private static final String INFO_LOG_PREFIX = "-- ";
    private static final String SECTION_NAME_PREFIX = "Section_";

    private static final String CONFIG_FILE = "singleuser_config.properties";

    /*
     * Default configuration values which are used if properties
configuration
     * values can't be found
     */
    private static final int DEFAULT_NUMBER_OF_SECTIONS = 30;
    private static final int DEFAULT_NUMBER_OF_ROWS = 10;
    private static final int DEFAULT_NUMBER_OF_SEATS_IN_ROW = 20;
    private static final int DEFAULT_NUMBER_OF_BOOKINGS = 4800;

    /* Configuration properties keys */
    private static final String NUMBER_OF_SECTIONS_PROP = "sections.amount";
    private static final String NUMBER_OF_ROWS_PROP = "rows.amount";
    private static final String NUMBER_OF_SEATS_IN_ROW_PROP =
"seats.in.row.amount";
    private static final String NUMBER_OF_BOOKINGS_PROP = "bookings.amount";

    private static final Map<String, Integer> DEFAULT_MAPPING = new
HashMap<String, Integer>();
    static {
      DEFAULT_MAPPING
            .put(NUMBER_OF_SECTIONS_PROP, DEFAULT_NUMBER_OF_SECTIONS);
      DEFAULT_MAPPING.put(NUMBER_OF_ROWS_PROP, DEFAULT_NUMBER_OF_ROWS);
      DEFAULT_MAPPING.put(NUMBER_OF_SEATS_IN_ROW_PROP,
            DEFAULT_NUMBER_OF_SEATS_IN_ROW);
      DEFAULT_MAPPING
            .put(NUMBER_OF_BOOKINGS_PROP, DEFAULT_NUMBER_OF_BOOKINGS);
```

```
}

@Autowired
private SectionDao sectionDao;

@Autowired
private SeatDao seatDao;

@Autowired
private MatchesDao matchesDao;

@Autowired
private TicketDao ticketDao;

@Autowired
private UserDao userDao;

@Autowired
private BookingDao bookingDao;

public void executeTests() throws Exception {
  // Read load tests configuration
  Properties configProps = PropertiesLoader.loadProperties(CONFIG_FILE);

  StopWatch s = new StopWatch();

  // Initialize DB
  logger.info(INFO_LOG_PREFIX
        + "Initializing database(creating test dataset)");
  s.start();
  Matches match = createMatch();
  Set<Section> sections = createSections(getProp(NUMBER_OF_SECTIONS_PROP,
        configProps));
  Set<Seat> seats = createSeats(
        getProp(NUMBER_OF_ROWS_PROP, configProps), getProp(
              NUMBER_OF_SEATS_IN_ROW_PROP, configProps), sections);
  User user = createTestUser();
  s.stop();
  logger.info("Time to initialize database: " + s.getTime() + " ms");
  s.reset();

  StopWatch totalTime = new StopWatch();
  totalTime.start();

      // Insert tickets
      logger.info(INFO_LOG_PREFIX + "Inserting tickets");
      s.start();
      Set<Ticket> tickets = insertTickets(match, seats);
      s.stop();
      logger.info("Time to create " + tickets.size()
            + " tickets(with commit): " + s.getTime() + " ms");
      s.reset();

      // Book tickets
      logger.info(INFO_LOG_PREFIX + "Booking tickets");
      s.start();
```

```
            int amountOfBookings = getProp(NUMBER_OF_BOOKINGS_PROP,
configProps);
            reserveTickets(tickets, user, amountOfBookings);
            s.stop();
            logger.info("Time to reserve " + amountOfBookings
                  + " tickets(with commit): " + s.getTime() + " ms");
            s.reset();

            // Find sold tickets number
            logger.info(INFO_LOG_PREFIX + "Finding number of sold tickets");
            s.start();
            Integer amountOfSoldTickets = getSoldTicketsCount(match);
            logger.info(INFO_LOG_PREFIX + "Number of sold tickets is: "
                  + amountOfSoldTickets);
            s.stop();
            logger.info("Time to find number of sold tickets: " + s.getTime()
                  + " ms");
            s.reset();

            // Find number of seats on the stadium
            logger.info(INFO_LOG_PREFIX
                  + "Finding total number of seats on the stadium");
            s.start();
            Integer amountOfSeats = getOverallSeatsCount();
            logger.info(INFO_LOG_PREFIX + "Number of seats on the stadium: "
                  + amountOfSeats);
            s.stop();
            logger.info("Time to find total number of seats on the stadium: "
                  + s.getTime() + " ms");
            s.reset();

            // Delete tickets
            logger.info(INFO_LOG_PREFIX + "Deleting tickets");
            s.start();
            deleteTickets(tickets);
            s.stop();
            logger.info("Time to delete " + tickets.size()
                  + " tickets(with commit): " + s.getTime() + " ms");
            s.reset();

      totalTime.stop();

      // Clean DB
      logger.info("-- Cleaning database");
      s.start();
      cleanDB();
      s.stop();
      logger.info("Time to clean database: " + s.getTime() + " ms");
      logger.info("Total work time: " + totalTime.getTime() + " ms");
    }

    public void cleanDB() {
      userDao.deleteAll();
      bookingDao.deleteAll();
      ticketDao.deleteAll();
      matchesDao.deleteAll();
      seatDao.deleteAll();
```

```
      sectionDao.deleteAll();
    }

    public Integer getSoldTicketsCount(Matches match) {
      return matchesDao.getSoldTicketsCount(match);
    }

    public Integer getOverallSeatsCount() {
      return seatDao.getOverallSeatCount();
    }

    public Set<Ticket> insertTickets(Matches match, Set<Seat> seats) {
      StopWatch s = new StopWatch();
      s.start();
      Set<Ticket> tickets = createTickets(match, seats);
      s.stop();
      logger.info("Time to create " + tickets.size()
            + " tickets(without commit): " + s.getTime() + " ms");
      return tickets;
    }

    public void reserveTickets(Set<Ticket> tickets, User user, int
bookingAmount) {
      StopWatch s = new StopWatch();
      s.start();
      int count = 0;
      for (Ticket ticket : tickets) {
          if (count++ < bookingAmount) {
            ticket.setAvailable(false);

            Booking booking = new Booking();
            booking.setUser(user);
            booking.setPaidDate(new Date());
            booking.setPayMethod("Cash");
            booking.setPayTotal(new BigDecimal(100));

            bookingDao.save(booking);

            ticket.setBooking(booking);
            ticketDao.update(ticket);
          } else {
            break;
          }

      }
      s.stop();
      logger.info("Time to reserve " + bookingAmount
            + " tickets(without commit): " + s.getTime() + " ms");
    }

    public void deleteAllTickets() {
      StopWatch s = new StopWatch();
      s.start();
      ticketDao.deleteAll();
      s.stop();
      logger.info("Time to delete all tickets(without commit): "
            + s.getTime() + " ms");
```

```
}

public void deleteTickets(Set<Ticket> tickets) {
  StopWatch s = new StopWatch();
  s.start();
  for (Ticket ticket : tickets) {
      ticketDao.delete(ticket);
  }
  s.stop();
  logger.info("Time to delete " + tickets.size()
        + " tickets(without commit): " + s.getTime() + " ms");
}

public User createTestUser() {
  Account account = new Account();
  account.setPassword("password1");
  account.setUsername("username1");

  User user = new User();
  user.setFirstName("testFirstN");
  user.setLastName("testLastN");
  user.setAddress("Test Address");
  user.setCity("Test City");
  user.setCountry("Test Country");
  user.setEmail("Test Email");
  user.setCounty("Test County");
  user.setPhone("111-222-333");
  user.getAccounts().add(account);

  account.setUser(user);

  userDao.save(user);

  return user;
}

public Set<Ticket> createTickets(Matches match, Set<Seat> seats) {
  Set<Ticket> tickets = new HashSet<Ticket>(seats.size());
  for (Seat seat : seats) {
      Ticket ticket = new Ticket(seat, match, true);
      ticketDao.save(ticket);
      tickets.add(ticket);
  }

  return tickets;
}

private static int sectionId = 0;

public Set<Section> createSections(int amount) {
  Set<Section> createdSections = new HashSet<Section>(amount);
  for (int i = sectionId + 1; i <= amount; i++, sectionId++) {
      Section section = new Section(SECTION_NAME_PREFIX + i);
      sectionDao.save(section);
      createdSections.add(section);
  }
```

```
      return createdSections;
    }

    public Set<Seat> createSeats(int amountOfRaws, int amountOfSeatsInRow,
          Set<Section> sections) {
      Set<Seat> seats = new HashSet<Seat>(amountOfRaws * amountOfSeatsInRow
            * sections.size());
      for (Section section : sections) {
          for (int rawsCount = 1; rawsCount <= amountOfRaws; rawsCount++) {
            for (int seatsInRowCount = 1; seatsInRowCount <=
amountOfSeatsInRow; seatsInRowCount++) {
                Seat seat = new Seat(section, rawsCount, seatsInRowCount);
                seatDao.save(seat);
                seats.add(seat);
            }
          }
      }

      return seats;
    }

    public Matches createMatch() {
      Matches match = new Matches();
      match.setCompetition("Test competition");
      match.setAnalysis("Test Analysis");
      match.setMatchDate(new Date());
      match.setKickOff(new Date());
      match.setOpposition("Test Opposition");
      match.setVenue("Test Venue");
      match.setScoreGrassh(2);
      match.setScoreOppos(0);

      matchesDao.save(match);
      return match;
    }

    private static int getProp(String key, Properties properties) {
      String propStr = properties.getProperty(key);
      int result;
      if (StringUtils.isNotBlank(propStr) && StringUtils.isNumeric(propStr))
{
          result = Integer.valueOf(propStr);
      } else {
          logger.warn("Property [" + key
                + "] is blank or is not numeric. Returning default value.");
          result = DEFAULT_MAPPING.get(key);
      }
      return result;
    }
}
```

SingleUserLoadHibernateTestRunner.java

```
package com.grasshopper.dao.orm.hibernate.singleuser;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
```

```
import com.grasshopper.dao.orm.singleuser.SingleUserLoad;
import com.grasshopper.dao.orm.singleuser.SingleUserLoadImpl;

public class SingleUserLoadHibernateTestRunner {

    public static void main(String[] args)throws Exception {

      ApplicationContext context = new ClassPathXmlApplicationContext(
            new String[] {
                  "test-applicationContext-hibernate-propertyConfigurer.xml",
                  "applicationContext-datasource-dbcp.xml",
                  "load-applicationContext-datasource-
propertyConfigurer.xml",
                  "applicationContext-hibernate.xml",
                  "load-applicationContext-main.xml" });
      SingleUserLoad singleUserLoad = (SingleUserLoad) context
            .getBean("singleUserLoad");
      SingleUserLoadImpl.setLoggerClass(SingleUserLoadHibernateTestRunner.cla
ss);

      singleUserLoad.executeTests();
    }
}
```

SingleUserLoadJpaTestRunner.java

```
package com.grasshopper.dao.orm.jpa.singleuser;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.grasshopper.dao.orm.singleuser.SingleUserLoad;
import com.grasshopper.dao.orm.singleuser.SingleUserLoadImpl;

public class SingleUserLoadJpaTestRunner {

    public static void main(String[] args)throws Exception {

      ApplicationContext context = new ClassPathXmlApplicationContext(
            new String[] {
                  "load-applicationContext-datasource-
propertyConfigurer.xml",
                  "applicationContext-datasource-dbcp.xml",
                  "applicationContext-jpa.xml",
                  "load-applicationContext-main.xml" });
      SingleUserLoad singleUserLoad = (SingleUserLoad) context
            .getBean("singleUserLoad");
      SingleUserLoadImpl.setLoggerClass(SingleUserLoadJpaTestRunner.class);

      singleUserLoad.executeTests();
    }
}
```

Testing Table Joins

Joins.java

```
package com.grasshopper.dao.orm.joins;

import java.util.List;
import java.util.Set;

import com.grasshopper.entity.Account;
import com.grasshopper.entity.Booking;
import com.grasshopper.entity.Matches;
import com.grasshopper.entity.Participation;
import com.grasshopper.entity.Player;
import com.grasshopper.entity.Seat;
import com.grasshopper.entity.Ticket;
import com.grasshopper.entity.User;

public interface Joins {
    /* Main load tests execution method */
    void executeTests()throws Exception;

    /* Load tests (business logic methods) */
    Set<Account> findAccountByUserExample(User user);

    Set<Player> findPlayers(Matches match);

    Set<Ticket> findTicketsForTheBooking(Booking booking);

    /* Initialization methods */
    List<Ticket> createTestTickets(List<Matches> matches,
          Set<Seat> seats, int amountOfTickets);

    List<Booking> createTestBookings(List<Ticket> tickets, List<User> user);

    List<User> createTestUsers(int amountOfUsers);

    List<Player> createTestPlayers(int amountOfPlayers);

    List<Matches> createTestMatches(int amountOfMatches);

    List<Participation> createTestParticipations(List<Matches> matches,
          List<Player> players);

    void cleanDB();
}
```

JoinsImpl.java

```
package com.grasshopper.dao.orm.joins;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.HashSet;
```

```
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;

import org.apache.commons.lang.StringUtils;
import org.apache.commons.lang.math.RandomUtils;
import org.apache.commons.lang.time.StopWatch;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.transaction.annotation.Transactional;

import com.grasshopper.constant.PayMethod;
import com.grasshopper.constant.Position;
import com.grasshopper.constant.Team;
import com.grasshopper.dao.AccountDao;
import com.grasshopper.dao.BookingDao;
import com.grasshopper.dao.MatchesDao;
import com.grasshopper.dao.ParticipationDao;
import com.grasshopper.dao.PlayerDao;
import com.grasshopper.dao.TicketDao;
import com.grasshopper.dao.UserDao;
import com.grasshopper.dao.orm.singleuser.SingleUserLoad;
import com.grasshopper.entity.Account;
import com.grasshopper.entity.Booking;
import com.grasshopper.entity.Matches;
import com.grasshopper.entity.Participation;
import com.grasshopper.entity.Player;
import com.grasshopper.entity.Seat;
import com.grasshopper.entity.Section;
import com.grasshopper.entity.Ticket;
import com.grasshopper.entity.User;
import com.grasshopper.utils.PropertiesLoader;

public class JoinsImpl implements Joins {
    private static Log logger = LogFactory.getLog(JoinsImpl.class);
    private static final String INFO_LOG_PREFIX = "-- ";

    private static final String CONFIG_FILE = "joins_config.properties";

    public static void setLoggerClass(Class<?> loggerClass) {
      logger = LogFactory.getLog(loggerClass);
    }

    /*
     * Default configuration values which are used if properties
configuration
     * values can't be found
     */
    private static final int DEFAULT_NUMBER_OF_USERS = 100;
    private static final int DEFAULT_NUMBER_OF_PLAYERS = 30;
    private static final int DEFAULT_NUMBER_OF_MATCHES = 20;
    private static final int DEFAULT_NUMBER_OF_SECTIONS = 30;
    private static final int DEFAULT_NUMBER_OF_ROWS = 10;
    private static final int DEFAULT_NUMBER_OF_SEATS_IN_ROW = 20;
    private static final int DEFAULT_NUMBER_OF_TICKETS = 100;
```

```
    /* Configuration properties keys */
    private static final String NUMBER_OF_USERS_PROP = "users.amount";
    private static final String NUMBER_OF_PLAYERS_PROP = "players.amount";
    private static final String NUMBER_OF_MATCHES_PROP = "matches.amount";
    private static final String NUMBER_OF_SECTIONS_PROP = "sections.amount";
    private static final String NUMBER_OF_ROWS_PROP = "rows.amount";
    private static final String NUMBER_OF_SEATS_IN_ROW_PROP =
"seats.in.row.amount";
    private static final String NUMBER_OF_TICKETS_PROP = "tickets.amount";

    private static final Map<String, Integer> DEFAULT_MAPPING = new
HashMap<String, Integer>();
    static {
      DEFAULT_MAPPING.put(NUMBER_OF_USERS_PROP, DEFAULT_NUMBER_OF_USERS);
      DEFAULT_MAPPING.put(NUMBER_OF_PLAYERS_PROP, DEFAULT_NUMBER_OF_PLAYERS);
      DEFAULT_MAPPING.put(NUMBER_OF_MATCHES_PROP, DEFAULT_NUMBER_OF_MATCHES);
      DEFAULT_MAPPING
            .put(NUMBER_OF_SECTIONS_PROP, DEFAULT_NUMBER_OF_SECTIONS);
      DEFAULT_MAPPING.put(NUMBER_OF_ROWS_PROP, DEFAULT_NUMBER_OF_ROWS);
      DEFAULT_MAPPING.put(NUMBER_OF_SEATS_IN_ROW_PROP,
            DEFAULT_NUMBER_OF_SEATS_IN_ROW);
      DEFAULT_MAPPING.put(NUMBER_OF_TICKETS_PROP, DEFAULT_NUMBER_OF_TICKETS);
    }

    @Autowired
    private SingleUserLoad singleUserLoad;

    @Autowired
    private AccountDao accountDao;

    @Autowired
    private UserDao userDao;

    @Autowired
    private PlayerDao playerDao;

    @Autowired
    private MatchesDao matchesDao;

    @Autowired
    private ParticipationDao participationDao;

    @Autowired
    private TicketDao ticketDao;

    @Autowired
    private BookingDao bookingDao;

    public void executeTests() throws Exception {
      // Read load tests configuration
      Properties configProps = PropertiesLoader.loadProperties(CONFIG_FILE);

            logger.info(INFO_LOG_PREFIX
            + "Initializing database(creating test dataset)");
      StopWatch s = new StopWatch();
      s.start();
```

```
		List<User> users = createTestUsers(getProp(NUMBER_OF_USERS_PROP,
				configProps));

		List<Player> players = createTestPlayers(getProp(
				NUMBER_OF_PLAYERS_PROP, configProps));

		List<Matches> matches = createTestMatches(getProp(
				NUMBER_OF_MATCHES_PROP, configProps));

		createTestParticipations(matches, players);

		Set<Section> sections = singleUserLoad.createSections(getProp(
				NUMBER_OF_SECTIONS_PROP, configProps));

		Set<Seat> seats = singleUserLoad.createSeats(getProp(
				NUMBER_OF_ROWS_PROP, configProps), getProp(
				NUMBER_OF_SEATS_IN_ROW_PROP, configProps), sections);

		List<Ticket> tickets = createTestTickets(matches, seats, getProp(
				NUMBER_OF_TICKETS_PROP, configProps));

		List<Booking> bookings = createTestBookings(tickets, users);
		s.stop();
		logger.info("Time to initialize database: " + s.getTime() + " ms");
		s.reset();

		StopWatch totalTime = new StopWatch();
		totalTime.start();

			logger.info(INFO_LOG_PREFIX + "Finding a particular user account");
			s.start();
			findAccountByUserExample(users.get(RandomUtils
					.nextInt(users.size())));
			s.stop();
			logger.info("Time to find user accounts: " + s.getTime() + " ms");
			s.reset();

			logger.info(INFO_LOG_PREFIX
					+ "Finding players participating in a particular match");
			s.start();
			Set<Player> participatingPlayers = findPlayers(matches
					.get(RandomUtils.nextInt(matches.size())));
			s.stop();
			logger.info(INFO_LOG_PREFIX + " Number of participating players: "
					+ participatingPlayers.size());
			logger
					.info("Time to find players participating in a particular
match: "
							+ s.getTime() + " ms");
			s.reset();
			logger
					.info(INFO_LOG_PREFIX
							+ "Finding tickets that are booked under a particular
booking");
			s.start();
			Set<Ticket> foundTickets = findTicketsForTheBooking(bookings
					.get(RandomUtils.nextInt(bookings.size())));
```

```
            logger.info(INFO_LOG_PREFIX
                  + "Number of tickets found for a particular booking: "
                  + foundTickets.size());
            s.stop();
            logger
                  .info("Time to find tickets that are booked under a
particular booking: "
                        + s.getTime() + " ms");
            s.reset();

      totalTime.stop();

      logger.info(INFO_LOG_PREFIX + "Cleaning database");
      s.start();
      cleanDB();
      s.stop();
      logger.info("Time to clean database: " + s.getTime() + " ms");
      logger.info("Total work time: " + totalTime.getTime() + " ms");
    }

    public void cleanDB() {
      userDao.deleteAll();
      accountDao.deleteAll();
      participationDao.deleteAll();
      playerDao.deleteAll();
      matchesDao.deleteAll();
      bookingDao.deleteAll();
      ticketDao.deleteAll();
    }

    public Set<Account> findAccountByUserExample(User user) {
      Set<Account> foundAccounts = accountDao.findAccountByUserExample(user);
      logger.info(INFO_LOG_PREFIX + " Number of found user's accounts: "
            + foundAccounts.size());
      return foundAccounts;
    }

    @Transactional
    public Set<Player> findPlayers(Matches match) {
      Matches foundMatch = matchesDao.findById(match.getMatchId(), false);
      Set<Player> players = new HashSet<Player>();
      for (Participation p : foundMatch.getParticipations()) {
          players.add(p.getPlayer());
      }
      return players;
    }

    @Transactional(readOnly = true)
    public Set<Ticket> findTicketsForTheBooking(Booking booking) {
      Booking foundBooking = bookingDao.findById(booking.getBookId(), false);
      return foundBooking.getTickets();
    }

    public List<Booking> createTestBookings(List<Ticket> tickets,
          List<User> users) {
      List<Booking> createdBookings = new ArrayList<Booking>();
```

```
      for (Ticket ticket : tickets) {
          if (ticket.isAvailable()) {
            User randomUser = users.get(RandomUtils.nextInt(users.size()));

            Booking booking = new Booking();
            booking.setPaidDate(new Date());
            booking.setPayMethod(PayMethod.CASH.getPayMethod());
            booking.setPayTotal(new BigDecimal(100));
            booking.setUser(randomUser);

            bookingDao.save(booking);

            ticket.setAvailable(false);
            ticket.setBooking(booking);
            ticketDao.update(ticket);

            createdBookings.add(booking);
       } //for
          }// if

      return createdBookings;
    }

    public List<Ticket> createTestTickets(List<Matches> matches,
          Set<Seat> seats, int amountOfTickets) {
      List<Ticket> createdTickets = new ArrayList<Ticket>(amountOfTickets);
      int ticketsCount = 0;
      for (Matches match : matches) {
          for (Seat seat : seats) {
            ticketsCount++;

            Ticket ticket = new Ticket();
            ticket.setAvailable(true);

            ticket.setMatches(match);
            ticket.setSeat(seat);

            ticketDao.save(ticket);

            createdTickets.add(ticket);
          }
      }

      return createdTickets;
    }

    public List<Participation> createTestParticipations(List<Matches>
matches,
          List<Player> players) {
      List<Participation> createdParticipations = new
ArrayList<Participation>(
            matches.size() * players.size());
      for (Matches match : matches) {
          for (Player player : players) {
            Participation p = new Participation(match, player);
            participationDao.save(p);
            createdParticipations.add(p);
```

```
        }
    }

    return createdParticipations;
}

public List<Matches> createTestMatches(int amountOfMatches) {
    List<Matches> createdMatches = new ArrayList<Matches>(amountOfMatches);
    for (int i = 0; i < amountOfMatches; i++) {
        Matches match = new Matches();
        match.setCompetition("Test competition");
        match.setAnalysis("Test Analysis");
        match.setMatchDate(new Date());
        match.setKickOff(new Date());
        match.setOpposition("Test Opposition");
        match.setVenue("Test Venue");
        match.setScoreGrassh(2);
        match.setScoreOppos(0);

        matchesDao.save(match);

        createdMatches.add(match);
    }

    return createdMatches;
}

public List<Player> createTestPlayers(int amountOfPlayers) {
    List<Player> createdPlayers = new ArrayList<Player>(amountOfPlayers);
    for (int i = 0; i < amountOfPlayers; i++) {
        Player player = new Player();
        player.setTeam(Team.FIRST_TEAM.getTeam());
        player.setFirstName("testFirstN_" + i);
        player.setLastName("testLastN_" + i);
        player.setBirthDate(new Date());
        player.setHeight((short) 179);
        player.setWeight(new BigDecimal(75));
        player.setPosition(Position.DEFENDER.getPosition());
        player.setPlayerNum((short) i);
        player.setGoals((short) 20);
        player.setGamesPlayed((short) 50);
        player.setYellowCards((short) 4);
        player.setRedCards((short) 1);
        player.setNotes("Test Notes");

        playerDao.save(player);

        createdPlayers.add(player);
    }

    return createdPlayers;
}

private static int usersId = -1;

public List<User> createTestUsers(int amountOfUsers) {
    List<User> createdUsers = new ArrayList<User>();
```

```
        for (int i = usersId + 1; i < usersId + amountOfUsers; i++) {
            Account account1 = new Account();
            account1.setPassword("password_" + i);
            account1.setUsername("username_" + i);

            Account account2 = new Account();
            account2.setPassword("pass_" + i);
            account2.setUsername("user_" + i);

            User user = new User();
            user.setFirstName("testFirstN_" + i);
            user.setLastName("testLastN_" + i);
            user.setAddress("Test Address_" + i);
            user.setCity("Test City_" + i);
            user.setCountry("Test Country_" + i);
            user.setEmail("testEmail_" + i);
            user.setCounty("Test County_" + i);
            user.setPhone("111-222-333");
            account1.setUser(user);
            account2.setUser(user);
            user.getAccounts().add(account1);
            user.getAccounts().add(account2);

            userDao.save(user);

            createdUsers.add(user);
        }
        usersId += amountOfUsers;
        return createdUsers;
    }

    private static int getProp(String key, Properties properties) {
        String propStr = properties.getProperty(key);
        int result;
        if (StringUtils.isNotBlank(propStr) && StringUtils.isNumeric(propStr))
{
            result = Integer.valueOf(propStr);
        } else {
            logger.warn("Property [" + key
                    + "] is blank or is not numeric. Returning default value.");
            result = DEFAULT_MAPPING.get(key);
        }
        return result;
    }
}
```

JoinsLoadHibernateTestRunner.java

```
package com.grasshopper.dao.orm.hibernate.joins;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.grasshopper.dao.orm.joins.Joins;
import com.grasshopper.dao.orm.joins.JoinsImpl;

public class JoinsLoadHibernateTestRunner {
```

```
    public static void main(String[] args) throws Exception {
      ApplicationContext context = new ClassPathXmlApplicationContext(
            new String[] {
                  "test-applicationContext-hibernate-propertyConfigurer.xml",
                  "applicationContext-datasource-dbcp.xml",
                  "load-applicationContext-datasource-
propertyConfigurer.xml",
                  "applicationContext-hibernate.xml",
                  "load-applicationContext-main.xml" });
      Joins joins = (Joins) context.getBean("joins");
      JoinsImpl.setLoggerClass(JoinsLoadHibernateTestRunner.class);

      joins.executeTests();
    }
}
```

JoinsLoadJpaTestRunner.java

```
package com.grasshopper.dao.orm.jpa.joins;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.grasshopper.dao.orm.joins.Joins;
import com.grasshopper.dao.orm.joins.JoinsImpl;

public class JoinsLoadJpaTestRunner {

    public static void main(String[] args)throws Exception {
      ApplicationContext context = new ClassPathXmlApplicationContext(
            new String[] {
                  "load-applicationContext-datasource-
propertyConfigurer.xml",
                  "applicationContext-datasource-dbcp.xml",
                  "applicationContext-jpa.xml",
                  "load-applicationContext-main.xml" });
      Joins joins = (Joins) context.getBean("joins");
      JoinsImpl.setLoggerClass(JoinsLoadJpaTestRunner.class);

      joins.executeTests();
    }
}
```

Defining test parameters

singleuser_config.properties

```
#Number of records inserted into Section table
sections.amount=4
#Number of rows in one section
rows.amount=30
#Number of seats in one row
seats.in.row.amount=50
#Number of bookings made for a particular match (Booking table)
bookings.amount=5000
```

joins_config.properties

```
#Number of records inserted into User table
users.amount=20
#Number of records inserted into Player table
players.amount=30
#Number of records inserted into Matches table
matches.amount=5
#Number of records inserted into Section table
sections.amount=4
#Number of rows on in one section
rows.amount=10
#Number of seats in one row
seats.in.row.amount=15
#Number of tickets created for a particular match (Ticket table)
tickets.amount=100
```

Test Load Resources

load-applicationContext-datasource-propertieConfigurer.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

      <bean id="datasourcePropertyConfigurer"

      class="org.springframework.beans.factory.config.PropertyPlaceholderConf
igurer">
            <property name="location">
                  <value>classpath:load-datasource.properties</value>
            </property>
            <property name="ignoreUnresolvablePlaceholders">
                  <value>true</value>
            </property>
      </bean>

</beans>
```

load-applicationContext-main.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
      xmlns:tx="http://www.springframework.org/schema/tx"
      xsi:schemaLocation="
            http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
            http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
```

```
            http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

      <context:annotation-config />

      <bean id="singleUserLoad"
class="com.grasshopper.dao.orm.singleuser.SingleUserLoadImpl" />
      <bean id="multiUserLoad"
class="com.grasshopper.dao.orm.hibernate.multiuser.MultiUserLoadImpl"
scope="prototype"/>
      <bean id="joins" class="com.grasshopper.dao.orm.joins.JoinsImpl" />
</beans>
```

load-datasource.properties

```
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/grasshopper_load
jdbc.username=test
jdbc.password=test
persistence.xml.file=classpath:persistence-load.xml
```

log4j.properties

```
log4j.rootLogger=INFO, CONSOLE

log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
log4j.appender.CONSOLE.layout.ConversionPattern=%d{ABSOLUTE} %-5p [%c{1}]
%m%n

log4j.logger.com.grasshopper.dao.orm.hibernate.singleuser=DEBUG,
SINGLEUSER_LOAD_HIBERNATE_LOG_FILE
log4j.appender.SINGLEUSER_LOAD_HIBERNATE_LOG_FILE=org.apache.log4j.FileAppend
er
log4j.appender.SINGLEUSER_LOAD_HIBERNATE_LOG_FILE.File=logs/singleuser_hibern
ate_results.log
log4j.appender.SINGLEUSER_LOAD_HIBERNATE_LOG_FILE.layout=org.apache.log4j.Pat
ternLayout
log4j.appender.SINGLEUSER_LOAD_HIBERNATE_LOG_FILE.layout.ConversionPattern=%d
{ABSOLUTE} %-5p [%c{1}] %m%n

log4j.logger.com.grasshopper.dao.orm.hibernate.joins=DEBUG,
JOIN_QUERIES_LOAD_HIBERNATE_LOG_FILE
log4j.appender.JOIN_QUERIES_LOAD_HIBERNATE_LOG_FILE=org.apache.log4j.FileAppe
nder
log4j.appender.JOIN_QUERIES_LOAD_HIBERNATE_LOG_FILE.File=logs/joins_hibernate
_results.log
log4j.appender.JOIN_QUERIES_LOAD_HIBERNATE_LOG_FILE.layout=org.apache.log4j.P
atternLayout
log4j.appender.JOIN_QUERIES_LOAD_HIBERNATE_LOG_FILE.layout.ConversionPattern=
%d{ABSOLUTE} %-5p [%c{1}] %m%n

log4j.logger.com.grasshopper.dao.orm.jpa.singleuser=DEBUG,
SINGLEUSER_LOAD_JPA_LOG_FILE
log4j.appender.SINGLEUSER_LOAD_JPA_LOG_FILE=org.apache.log4j.FileAppender
```

```
log4j.appender.SINGLEUSER_LOAD_JPA_LOG_FILE.File=logs/singleuser_jpa_results.
log
log4j.appender.SINGLEUSER_LOAD_JPA_LOG_FILE.layout=org.apache.log4j.PatternLa
yout
log4j.appender.SINGLEUSER_LOAD_JPA_LOG_FILE.layout.ConversionPattern=%d{ABSOL
UTE} %-5p [%c{1}] %m%n

log4j.logger.com.grasshopper.dao.orm.jpa.joins=DEBUG,
JOIN_QUERIES_LOAD_JPA_LOG_FILE
log4j.appender.JOIN_QUERIES_LOAD_JPA_LOG_FILE=org.apache.log4j.FileAppender
log4j.appender.JOIN_QUERIES_LOAD_JPA_LOG_FILE.File=logs/joins_jpa_results.log
log4j.appender.JOIN_QUERIES_LOAD_JPA_LOG_FILE.layout=org.apache.log4j.Pattern
Layout
log4j.appender.JOIN_QUERIES_LOAD_JPA_LOG_FILE.layout.ConversionPattern=%d{ABS
OLUTE} %-5p [%c{1}] %m%n
```

persistence-load.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0"
      xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
      <persistence-unit name="GrasshopperPU" transaction-
type="RESOURCE_LOCAL">

      <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
            <class>com.grasshopper.entity.Account</class>
            <class>com.grasshopper.entity.Authorities</class>
            <class>com.grasshopper.entity.AuthoritiesId</class>
            <class>com.grasshopper.entity.Booking</class>
            <class>com.grasshopper.entity.CreditCard</class>
            <class>com.grasshopper.entity.Matches</class>
            <class>com.grasshopper.entity.Participation</class>
            <class>com.grasshopper.entity.Player</class>
            <class>com.grasshopper.entity.Seat</class>
            <class>com.grasshopper.entity.Seatprice</class>
            <class>com.grasshopper.entity.Section</class>
            <class>com.grasshopper.entity.Ticket</class>
            <class>com.grasshopper.entity.User</class>
            <properties>
                  <property name="eclipselink.jdbc.driver"
value="com.mysql.jdbc.Driver"/>
                  <property name="eclipselink.jdbc.url"
value="jdbc:mysql://localhost:3306/grasshopper_load"/>
                  <property name="eclipselink.ddl-generation"
value="verify"/>
                  <property name="eclipselink.jdbc.user" value="test"/>
                  <property name="eclipselink.jdbc.password" value="test"/>
                  <property name="eclipselink.logging.level" value="INFO"/>
                  <property name="eclipselink.target-database"
value="MySQL"/>
            </properties>
      </persistence-unit>
</persistence>
```

Appendix I

Test Results

singleuser-config.properties

```
#Number of records inserted into Section table
sections.amount=4
#Number of rows in one section
rows.amount=30
#Number of seats in one row
seats.in.row.amount=50
#Number of bookings made for a particular match (Booking table)
bookings.amount=5000
```

single_user_hibernate_results_log

```
14:56:50,734 INFO  [SingleUserLoadHibernateTestRunner] -- Initializing
database(creating test dataset)
15:01:45,875 INFO  [SingleUserLoadHibernateTestRunner] Time to initialize
database: 295141 ms
15:01:45,875 INFO  [SingleUserLoadHibernateTestRunner] -- Inserting tickets
15:06:46,781 INFO  [SingleUserLoadHibernateTestRunner] Time to create 6000
tickets(without commit): 300906 ms
15:06:46,781 INFO  [SingleUserLoadHibernateTestRunner] Time to create 6000
tickets(with commit): 300906 ms
15:06:46,781 INFO  [SingleUserLoadHibernateTestRunner] -- Booking tickets
15:13:38,562 INFO  [SingleUserLoadHibernateTestRunner] Time to reserve 5000
tickets(without commit): 411781 ms
15:13:38,562 INFO  [SingleUserLoadHibernateTestRunner] Time to reserve 5000
tickets(with commit): 411781 ms
15:13:38,562 INFO  [SingleUserLoadHibernateTestRunner] -- Finding number of
sold tickets
15:13:38,843 INFO  [SingleUserLoadHibernateTestRunner] -- Number of sold
tickets is: 5000
15:13:38,843 INFO  [SingleUserLoadHibernateTestRunner] Time to find number of
sold tickets: 281 ms
15:13:38,843 INFO  [SingleUserLoadHibernateTestRunner] -- Finding total
number of seats on the stadium
15:13:38,859 INFO  [SingleUserLoadHibernateTestRunner] -- Number of seats on
the stadium: 6000
15:13:38,859 INFO  [SingleUserLoadHibernateTestRunner] Time to find total
number of seats on the stadium: 16 ms
15:13:38,859 INFO  [SingleUserLoadHibernateTestRunner] -- Deleting tickets
15:18:56,390 INFO  [SingleUserLoadHibernateTestRunner] Time to delete 6000
tickets(without commit): 317531 ms
15:18:56,390 INFO  [SingleUserLoadHibernateTestRunner] Time to delete 6000
tickets(with commit): 317531 ms
15:18:56,390 INFO  [SingleUserLoadHibernateTestRunner] -- Cleaning database
15:18:56,906 INFO  [SingleUserLoadHibernateTestRunner] Time to clean
database: 516 ms
15:18:56,906 INFO  [SingleUserLoadHibernateTestRunner] Total work time:
1030515 ms
```

single_user_jpa_results_log

```
15:20:36,781 INFO  [SingleUserLoadJpaTestRunner] -- Initializing
database(creating test dataset)
15:26:50,000 INFO  [SingleUserLoadJpaTestRunner] Time to initialize database:
373219 ms
15:26:50,000 INFO  [SingleUserLoadJpaTestRunner] -- Inserting tickets
15:33:04,609 INFO  [SingleUserLoadJpaTestRunner] Time to create 6000
tickets(without commit): 374609 ms
15:33:04,609 INFO  [SingleUserLoadJpaTestRunner] Time to create 6000
tickets(with commit): 374609 ms
15:33:04,609 INFO  [SingleUserLoadJpaTestRunner] -- Booking tickets
15:40:06,359 INFO  [SingleUserLoadJpaTestRunner] Time to reserve 5000
tickets(without commit): 421750 ms
15:40:06,359 INFO  [SingleUserLoadJpaTestRunner] Time to reserve 5000
tickets(with commit): 421750 ms
15:40:06,359 INFO  [SingleUserLoadJpaTestRunner] -- Finding number of sold
tickets
15:40:07,203 INFO  [SingleUserLoadJpaTestRunner] -- Number of sold tickets
is: 5000
15:40:07,203 INFO  [SingleUserLoadJpaTestRunner] Time to find number of sold
tickets: 844 ms
15:40:07,203 INFO  [SingleUserLoadJpaTestRunner] -- Finding total number of
seats on the stadium
15:40:07,203 INFO  [SingleUserLoadJpaTestRunner] -- Number of seats on the
stadium: 6000
15:40:07,203 INFO  [SingleUserLoadJpaTestRunner] Time to find total number of
seats on the stadium: 18 ms
15:40:07,203 INFO  [SingleUserLoadJpaTestRunner] -- Deleting tickets
15:44:46,046 INFO  [SingleUserLoadJpaTestRunner] Time to delete 6000
tickets(without commit): 278843 ms
15:44:46,046 INFO  [SingleUserLoadJpaTestRunner] Time to delete 6000
tickets(with commit): 278843 ms
15:44:46,046 INFO  [SingleUserLoadJpaTestRunner] -- Cleaning database
15:44:46,812 INFO  [SingleUserLoadJpaTestRunner] Time to clean database: 766
ms
15:44:46,812 INFO  [SingleUserLoadJpaTestRunner] Total work time: 1076064 ms
```

joins-config.properties

```
#Number of records inserted into User table
users.amount=30
#Number of records inserted into Player table
players.amount=30
#Number of records inserted into Matches table
matches.amount=20
#Number of records inserted into Section table
sections.amount=4
#Number of rows on in one section
rows.amount=10
#Number of seats in one row
seats.in.row.amount=30
#Number of tickets created for a particular match (Ticket table)
```

```
tickets.amount=1000
```

joins_hibernate_results_log

```
18:05:26,171 INFO  [JoinsLoadHibernateTestRunner] -- Initializing
database(creating test dataset)
18:14:32,500 INFO  [JoinsLoadHibernateTestRunner] Time to initialize
database: 546329 ms
18:14:32,500 INFO  [JoinsLoadHibernateTestRunner] -- Finding a particular
user account
18:14:32,671 INFO  [JoinsLoadHibernateTestRunner] --  Number of found user's
accounts: 2
18:14:32,671 INFO  [JoinsLoadHibernateTestRunner] Time to find user accounts:
171 ms
18:14:32,671 INFO  [JoinsLoadHibernateTestRunner] -- Finding players
participating in a particular match
18:14:32,718 INFO  [JoinsLoadHibernateTestRunner] --  Number of participating
players: 30
18:14:32,718 INFO  [JoinsLoadHibernateTestRunner] Time to find players
participating in a particular match: 47 ms
18:14:32,718 INFO  [JoinsLoadHibernateTestRunner] -- Finding tickets that are
booked under a particular booking
18:14:32,718 INFO  [JoinsLoadHibernateTestRunner] -- Number of tickets found
for a particular booking: 1
18:14:32,718 INFO  [JoinsLoadHibernateTestRunner] Time to find tickets that
are booked under a particular booking: 12 ms
18:14:32,718 INFO  [JoinsLoadHibernateTestRunner] -- Cleaning database
18:14:33,734 INFO  [JoinsLoadHibernateTestRunner] Time to clean database:
1000 ms
18:14:33,734 INFO  [JoinsLoadHibernateTestRunner] Total work time: 230 ms
```

joins_jpa_results_log

```
18:17:19,390 INFO  [JoinsLoadJpaTestRunner] -- Initializing database(creating
test dataset)
18:27:37,125 INFO  [JoinsLoadJpaTestRunner] Time to initialize database:
617735 ms
18:27:37,125 INFO  [JoinsLoadJpaTestRunner] -- Finding a particular user
account
18:27:37,468 INFO  [JoinsLoadJpaTestRunner] --  Number of found user's
accounts: 2
18:27:37,468 INFO  [JoinsLoadJpaTestRunner] Time to find user accounts: 343
ms
18:27:37,468 INFO  [JoinsLoadJpaTestRunner] -- Finding players participating
in a particular match
18:27:37,562 INFO  [JoinsLoadJpaTestRunner] --  Number of participating
players: 30
18:27:37,562 INFO  [JoinsLoadJpaTestRunner] Time to find players
participating in a particular match: 94 ms
18:27:37,562 INFO  [JoinsLoadJpaTestRunner] -- Finding tickets that are
booked under a particular booking
18:27:37,562 INFO  [JoinsLoadJpaTestRunner] -- Number of tickets found for a
particular booking: 1
```

```
18:27:37,578 INFO  [JoinsLoadJpaTestRunner] Time to find tickets that are
booked under a particular booking: 16 ms
18:27:37,578 INFO  [JoinsLoadJpaTestRunner] -- Cleaning database
18:27:38,390 INFO  [JoinsLoadJpaTestRunner] Time to clean database: 1012 ms
18:27:38,390 INFO  [JoinsLoadJpaTestRunner] Total work time: 453 ms
```