Salesmen (Group2 SE)

Software Design Document

author(s): Bart Maes

Abstract: This document lays down the architecture and detailed design of the Salesmen project. This project is created by group 2 of the course Software Engineering in the 2009-2010 study year at the Vrije Universiteit Brussel.

06-05-2010

History of versions

Version	Date	Author(s)	Changes
v0.1	27-11-2009	B. Maes	Draft initialization
v0.1.1	29-11-2009	B. Maes	Added Appendix A, added Technical Design
v0.1.2	29-11-2009	B. Maes	Added Appendix B, initial QA check
v0.1.3	14-12-2009	B. Maes	JBoss Seam Integration
v0.2	21-12-2009	P. Provinciael	Made the document consistent with the other documents
v0.2.1	06-04-2010	B. Maes	Update document before second iteration
v0.2.2	11-04-2010	B. Maes	Reread document, made some final changes
v0.3.0	06-05-2010	B. Maes	Added page descriptions (view), fixed various type- and spellingerrors.

Contents

Chapter 1

Introduction

1.1 Purpose

This document specifies the entire software architecture and design for the Salesmen software. These design decisions directly relate to the functionalities, performances, constraints, attributes and interfaces of the system, as they are mentioned in the Software Requirements Specification (SRS, [1]) document.

1.2 Scope

This document describes the software architecture and design for the first stable release of the Salesmen software. The intended audience of this document exclusively includes the designers, the developers and the testers of the software.
It should be considered a thorough guide for the implementation of the software. However, developers may deviate from it as they see required during the implementation phase, as long as they discuss these changes with the design manager.

1.3 Reference materials

See Appendix A.

1.4 Definitions and acronyms

CSS	Cascading Style Sheets
EJB3	Enterprise Java Beans v3
HTML	Hypertext Markup Language
XHTML	Extensible Hypertext Markup Language
MVC	Model-View-Controller, a design pattern
HQL	Hibernate Query Language
J2EE	Java 2 Platform, Enterprise Edition
JBoss	An application server
RDBMS	Relational Database Management System
SCMP	Software Configuration Management Plan
SDD	Software Design Document
SMTP	Single Mail Transfer Protocol
SRS	Software Requirements Specification
SQL	Structured Query Language

Chapter 2

System Architecture

2.1 Overview

System architecture diagram: see Appendix B, B.1

2.2 Three-Tier Architecture

Three-tier architecture is a client-server architecture in which the user interface, functional process logic, computer data storage and data access are developed and maintained as independent modules, most often on separate platforms.

Apart from the usual advantages of modular software with well defined interfaces, the three-tier architecture is intended to allow any of the three tiers to be upgraded or replaced independently as requirements or technology change. For example, a change of operating system in the presentation tier would only affect the user interface code.

Typically, the user interface runs on a desktop PC or workstation and uses a standard graphical user interface, functional process logic may consist of one or more separate modules running on a workstation or application server, and a RDBMS on a database server or mainframe contains the computer data storage logic.

Three-tier architecture has the following three tiers:

- **Presentation tier** This is the topmost level of the application. The presentation tier displays information related to such services as browsing merchandise, purchasing, and shopping cart contents. It communicates with other tiers by outputting results to the browser/client tier and all other tiers in the network.

- **Application tier (Business Logic/Logic Tier)** The logic tier is pulled out from the presentation tier and, as its own layer, it controls an applications functionality by performing detailed processing.

- **Data tier** This tier consists of Database Servers. Here information is stored and retrieved. This tier keeps data neutral and independent from application servers or business logic. Giving data its own tier also improves scalability and performance.

Benefits of using this architectural style include increased performance, flexibility, maintainability, reusability, and scalability.

2.3 Discussion of Alternative Designs

MVC At first glance, the three tiers may seem similar to the MVC (Model View Controller) concept. However, topologically they are different. A fundamental rule in a three-tier architecture is the client tier never communicates directly with the data tier; in a three-tier model all communication must pass through the middleware tier. Conceptually the three-tier architecture is linear. However, the MVC architecture is triangular: the View sends updates to the Controller, the Controller updates the Model, and the View gets updated directly from the Model.

From a historical perspective the three-tier architecture concept emerged in the 1990s from observations of distributed systems (e.g., web applications) where the client, middleware and data tiers ran on physically separate platforms. Whereas MVC comes from the previous decade and is based on observations of applications that ran on a single graphical workstation; MVC was applied to distributed applications much later in its history.

2.4 Survey of Technologies Used

2.4.1 Introduction

This section describes briefly what software and technologies will be used to implement the Salesmen software. For more information about these technologies, the reader is kindly referred to the SCMP ([2]). This document will, however, concentrate on the practical side of these applications, libraries and frameworks.

2.4.2 JBoss Seam

Seam is a powerful open source development platform for building rich Internet applications in Java. Seam integrates technologies such as Asynchronous JavaScript and XML (AJAX), JavaServer Faces (JSF), Java Persistence (JPA), Enterprise Java Beans (EJB 3.0) and Business Process Management (BPM) into a unified full-stack solution, complete with sophisticated tooling.

Seam has been designed from the ground up to eliminate complexity at both architecture and API levels. It enables developers to assemble complex web applications using simple annotated Java classes, a rich set of UI components, and very little XML. Seam's unique support for conversations and declarative state management can introduce a more sophisticated user experience while at the same time eliminating common bugs found in traditional web applications.

As a framework, it covers the 3 layers of the three tier architecture.

Layers

Presentation Layer XHTML and CSS files will cover the view of our project. The Seam framework allows the view to communicate with the business logic though Seam components. Seam uses reflection to enable easy-to-use Seam components in the view-files of the software. As a page is requested by a web browser, Seam will instantiate all mentioned components and display the values of the requested properties. Furthermore, it is easy to fill in the properties of a component by means of a web-form. The aforementioned components will be discussed in greater detail in the Seam Components section (2.4.2).

Business Layer JBoss Application Server (or JBoss AS) is a free software/open-source Java EE-based application server. Because it is Java-based, the JBoss application server operates cross-platform: usable on any operating system that Java supports. As part of the JBoss software, the Apache Tomcat servlet container will allow us to serve dynamically parsed web pages. JBoss Seam

allows us to implement the required Java Beans and Seam Entities in a fast and easy way, ensuring security and reliability.

Data Layer Two database platforms were considered: MySQL and PostgresSQL. After thorough investigation, it was decided to go with PostgreSQL. As documented in the Software Configuration Management Plan (SCMP, [2]), a thorough comparison was made between these two RDBM systems.
However, as one will discover by reading this document, this decision became less important as Seam provides a vast amount of database-related annotations, allowing to move away from a RDBMS-specific implementation. As a framework, Seam creates the described database and will also handle all select, update, insert and delete operations. With the help of these annotations, (primary) keys, default values, allowed values, relationships between tables, and more, can be obtained in a declarative manner. A more complete description of these annotations and their use can be found in section 2.4.2
Although Seam omits the need for basic select, insert and update SQL statements, it remains possible to communicate directly with the database by means of the Hibernate Query Language, HQL, which is also extended by Seam to support Seamś components and reflective evaluation.

Annotations

The Seam framework allows the use of various annotations which help to write robust and correct code. Two mayor types can be observed:

Database-related annotations These annotations allow a programmer to describe properties of tables and fields containing the properties held by Plain Old Java Objects (POJOś, also referred to as entity beans). These objects hold various properties and provides getter and setter methods to read or update these properties. The Seam framework will create a table in the database, containing a column for each property held by this class with the help of reflection, as provided by the Java programming language. By providing an entity of this class to persist-function of Seamś EntityManager, it is simply stored in this table without further ado.
The following annotations will be used to describe the database and its tables and columns:

- @Table: defines the database table name used for this POJO.
- @NotNull: forces a column entry to have some value.
- @ManyToOne,@OneToOne,@OneToMany: describes cardinality of a property

Other annotations Seam provides more annotations which allow to describe correctness of properties. Seam enforces these rules when an attempt is made to update the ṕrotectedṕroperty. Please note that Seam implicitly provides the same checks for data-types: it will never allow a string to be set to an integer property. It is important to understand that Seam will not simply throw an exception when an input error, type-based or annotation-based, occurs. Instead, it will send messages to the FacesMessages component which will, in its turn, update the view, displaying clear descriptions, informing the user of the system of input incorrectness. Also note that these messages will be defined in language files, allowing easy support for multiple languages.
The following annotations will be used most frequently, although more do exist:

- @Name: defines the name of the Seam component.
- @In: injects a component into the current code/object

Seam Components

Seam components represent a link between the business logic, provided by a plethora of Java classes and described in detail in the class diagrams, and the view, represented by XHTML files. As explained in a previous section, Seam Components are instantiated upon a page request and their values are bound to the XHTML-forms and text fields. Apart from the components defined by the programmers, Seam provides some components of its own. The following built-in components will be used frequently throughout the code:

- EntityManager: can be used to store (persist) entity beans (objects) in the database
- FacesMessages: displays (enqueued) messages in the view of the software

As Seam is a very complete framework, describing all its features would bring us beyond the scope of this document. Therefor, the reader is kindly referred to the Seam documentation [4].

2.5 Physical Configuration

The software is to be installed on the Wilma server at the VUB datacenter. Wilma is accessible over the Internet through web browsers and command line clients. The software relies on the availability of an SMTP server on the local LAN of the VUB, in order for us not having to include MDA (Mail Delivery Agent) functionality in the software.

System infrastructure diagram: see Appendix B, B.2

2.6 System Interface Description

The web-interface is the only interface available to users of the Salesmen software. However, depending of the access-rights provided to a given user, more options may become available. Website administrators, for example, can close auctions or ban other members while 'normal' members cannot access these facilities. A more detailed description of the access-rights provided to each group of users can be found in the SRS.
Prototypes of the web-interface can be found in Appendix C.

Chapter 3

Technical Design

3.1 Presentation layer

The presentation merely consists of static XHTML files that contain calls to Seam Components to obtain dynamically changing information. These XHTML files will use CSS to separate the static information from the actual presentation of the website. Seam uses AJAX technology, provided by RichFaces, to offer the user with a friendly and smooth interface. As discussed before, Seam Components allow easy access of information stored in JAVA objects.
Another feature of the Seam framework is built-in support of so-called language files, providing the developers an easy way to implement multiple languages.

This facilitates the following goals:

- Presentation is abstracted from (dynamically changing) information, coming from the business layer.
- Support for multiple languages, as required in the SRS.
- 'Smooth' interface with a minimum of required page refreshes, as required in the SRS.

Prototypes of the web-interface can be found in Appendix C.

3.2 Business layer

The business layer consists of the Java Components that will be able to receive and process requests, and return proper responses. From the class diagram it becomes clear that two classes will play a central role in our application: User and Auction. It should be noted that this class diagram was thoroughly updated since the last revision of this document. The previous version of the class diagram did not take the Seam framework into consideration, which let to a somewhat incorrect class diagram. The changes to this diagram can be described as follows:

- Most of the existing classes were stripped down to so-called Plain Old Java Objects (or POJOs), only containing the necessary properties and their respective getters and setters.
- Where needed, annotations were added to classes and properties, to assert correctness of data held by an entity.
- Most of the non-getter/setter methods mentioned in the previous SDD are now housed in various so-called 'Home' classes, which usually encapsulate stateful session information. These

classes are used by the various user-forms in the project and allow for extra checks on the provided input. These extra checks constitute of requirements on data (e.g. consistency) which cannot be described by annotations.

Three class diagrams are provided with this document: the first shows the relations between the different POJO classes, with the non-POJO classes encapsulated in the session-package. This view offers great insight in the relations between the different classes and resembles the previous class diagram, found in previous versions of this document. Another view expands the session package and collapses the entity package, hiding all the POJO classes. This view offers great insight in how the website processes incoming requests. The third view was added for the sake of completeness: it shows both packages expanded. However, due to the vast amount of classes in the diagram, it becomes hard to read. Because of this, this diagram will be available on the website as well ([5]).

Class diagram, see Appendix B, B.5, B.6, B.7

3.3 Data layer

The data layer consists of the database and the EJB3 standard for persistent objects. EJB3 allows, through the use of annotations in the class, to abstract away from SQL-statements as EJB3 will generate these.

The database design supports all current requirements and is designed with expansion in mind. Because the database is automatically created by the Seam framework, the Entity-Relationship diagram should be used to identify the required database annotations.

Entity-Relationship diagram: see Appendix B, B.8

Chapter 4

Software Features

4.1 Software Requirements

The design described in this document is based on the requirements as they are set in version 0.3.1 of the Software Requirements Specification (SRS) document. The design covers all of the must-have and most of the want-to-have requirements, and allows for easy expansion to support the remaining want-to-have and nice-to-have requirements.
For a complete list of these requirements, the reader is kindly referred to the SRS document (**??**).

Chapter 5

User Interface Design

5.1 Web pages tree

5.1.1 Overview

The User Interface files will be stored in the *view* folder. Although most services will be accessible or usable within 1 page, some, usually larger, services, such as *creating an auction*, are divided over several view-files. This is done for several reasons:

- Users of the system will not be overwhelmed by bombastic web-forms, possibly scaring them away from the site.
- The website can be considered more user-friendly, as a user will need to scroll fewer times down a web-page.
- In some situations it may be easier to implement browser-compatible web-pages (as required in the SRS).

5.1.2 Page Description

The following table provides an overview of the available view files and their purpose.

Pagename	Description
home.xhtml	The homepage.
auction.xhtml	Displays an auction.
auctionList.xhtml	A list of auctions.
register.xhtml	First page of the register process.
register2.xhtml	Second page of the register process.
register3.xhtml	Third page of the register process.
basicSearch.xhtml	The interface for the basic search.
advancedSearch.xhtml	The interface for the advanced search.
userAccountManagement.xhtml	Interface for users to change their account settings.
myAuctions.xhtml	Displays the auctions owned by *this* user
createAuction.xhtml	First page (step 1) of the auction-creation process.
createAuction2.xhtml	Second page (step 2) of the auction-creation process.
previewAuction.xhtml	During the auction-creation process, this page displays a preview of the new auction (step 3).
userProfile.xhtml	Displays user profile information.
userProfileManagement.xhtml	Allows for updating account information.
login.xhtml	Login form.
forgotpassword.xhtml	Page where a user can request a password reset.
error.xhtml	General error page.
activateAccount.xhtml	Displays a message informing account activation succeeded or failed.
confirmAccountEmail.xhtml	Body of the account activation/confirmation email.
accountBanned.xhtml	Displays a message to inform the used account was banned.
activateAccountEmail.xhtml	The (xhtml) body of the account activation mail.
passwordEmail.xhtml	Body of the new password (after reset) email.
requestPaymentEmail.xhtml	Body of the email requesting payment (after an auction finished).

Note that the view of our project also contains the following folders.

Foldername	Description
layout	Contains several pieces of the website that are included on (almost) every page.
img	Contains the images used throughout the website (NOT the auction-pictures).
stylesheet	Contains several (X)CSS-files.

Moreover, thanks to Seam, the view can be controlled (as in: how can one page lead to another, etc) using special but relatively simple xml files. The decision as the where and how to use these files are left to the discretion of the programmers.

5.2 User Interface

Prototypes of the web-interface can be found in Appendix C.

Appendix A

References

[1] Software Requirements Specification
`http://` [April 2010]

[2] Software Configuration Management Plan
`http://` [April 2010]

[3] Seam Documentation
`http://seamframework.org/Documentation` [April 2010]

[4] Seam documentation
`http://` [April 2010]

[5] SDD, full class diagram
`http://` [April 2010]

Appendix B

Diagrams

B.1 Architecture

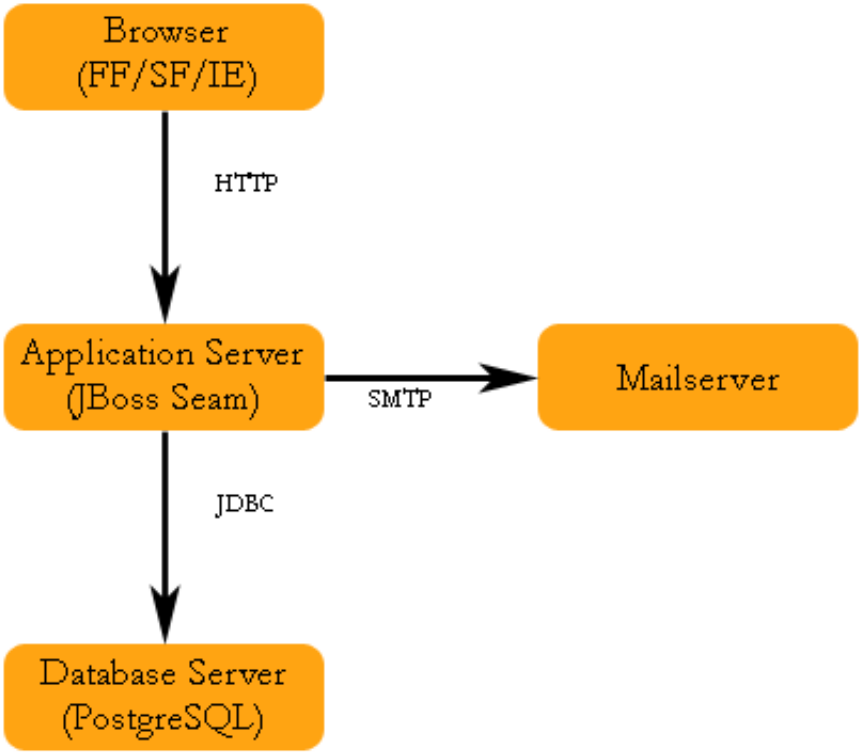

Figure B.1: Salesmen Architecture

B.2 Infrastructure

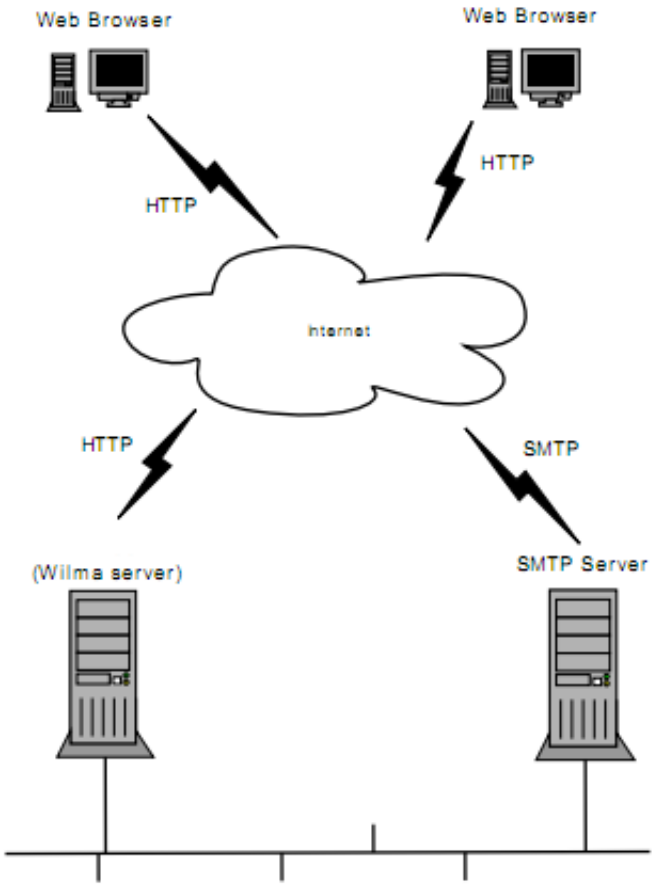

Figure B.2: Salesmen infrastructure

B.3 Modules

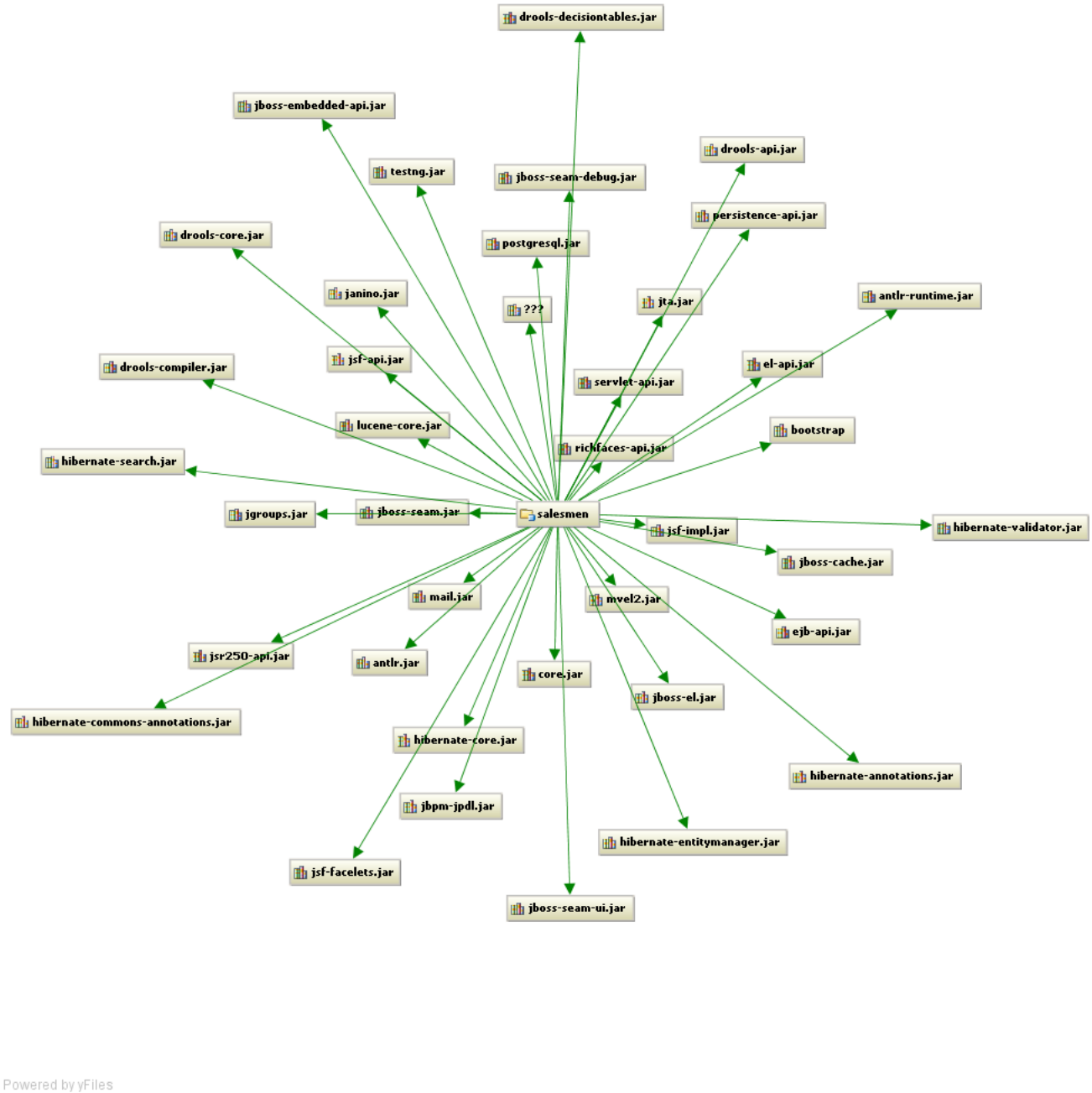

Figure B.3: Salesmen - used modules

B.4 Package Diagram

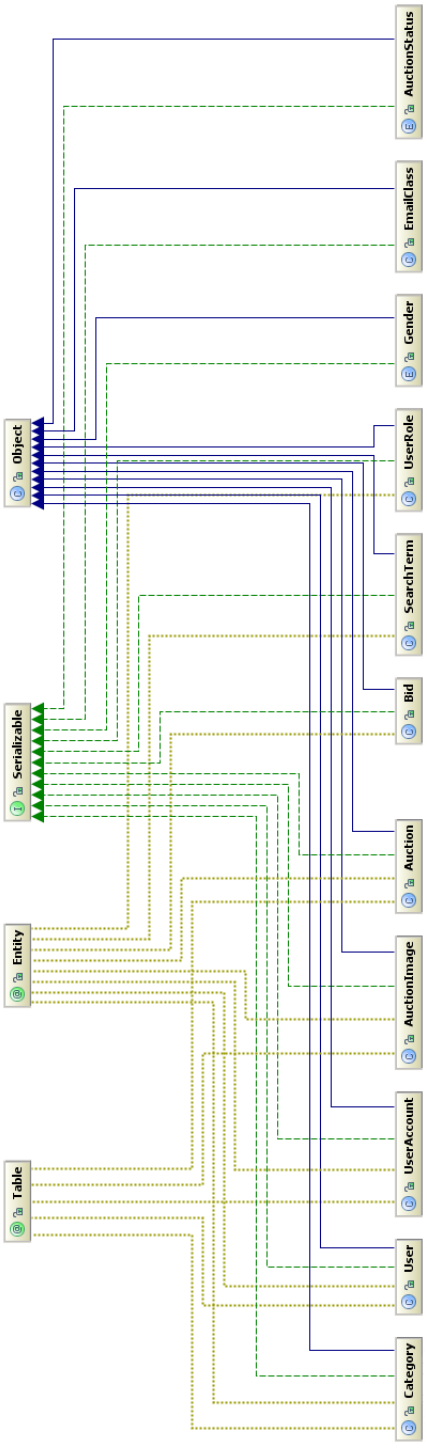

Figure B.4: Salesmen package diagram - global view

B.5 Class Diagram - Entity package

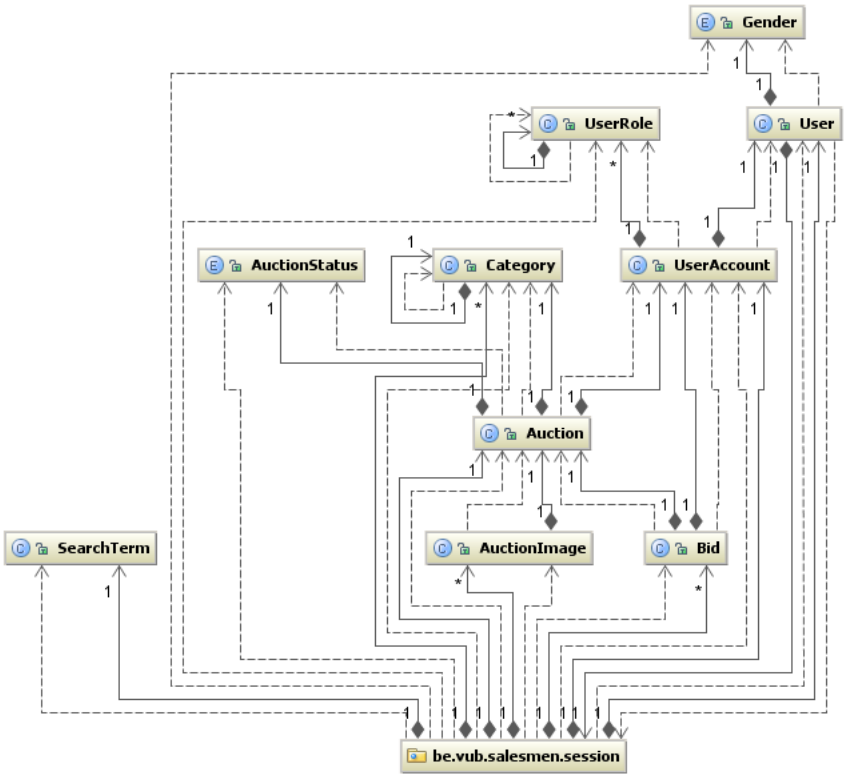

Figure B.5: Salesmen class diagram - entity package expanded, session package collapsed

B.6 Class Diagram - Session package

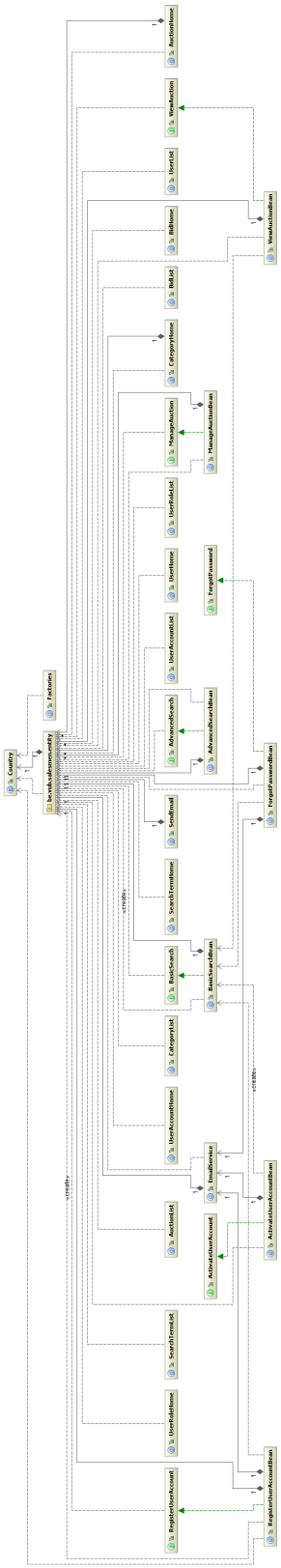

Figure B.6: Salesmen class diagram - session package expanded, entity package collapsed

B.7 Class Diagram - All packages

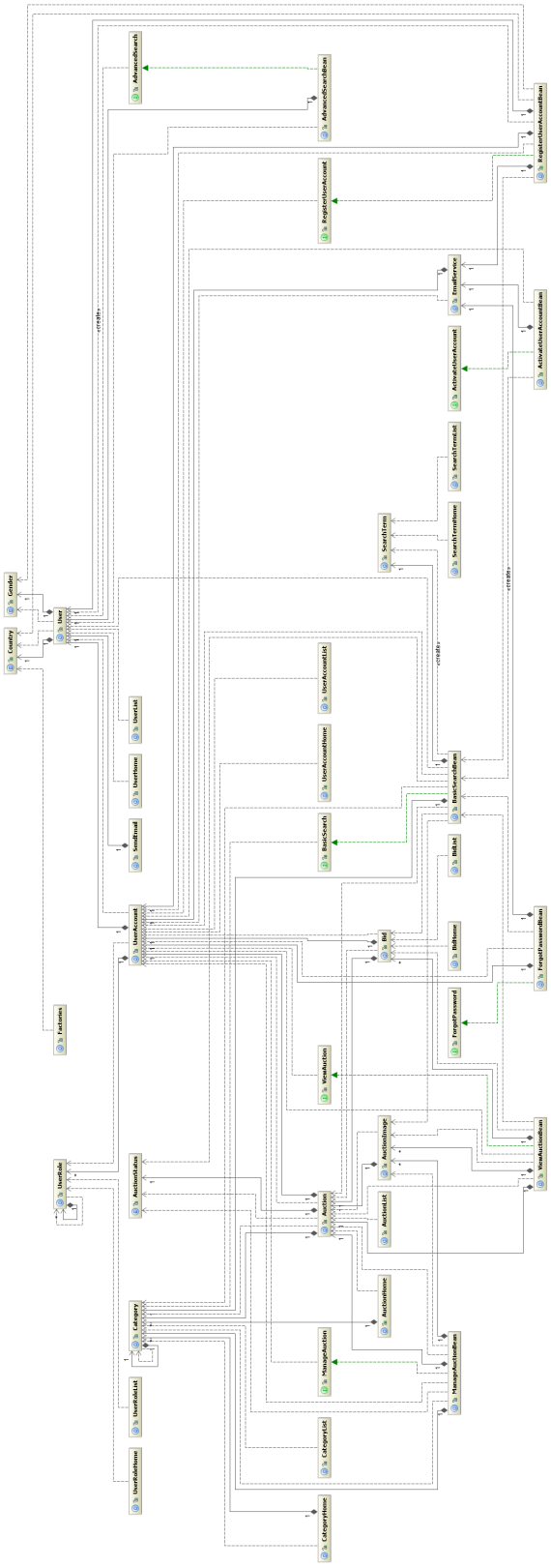

Figure B.7: Salesmen class diagram - session and entity packages expanded

B.8 Database diagram

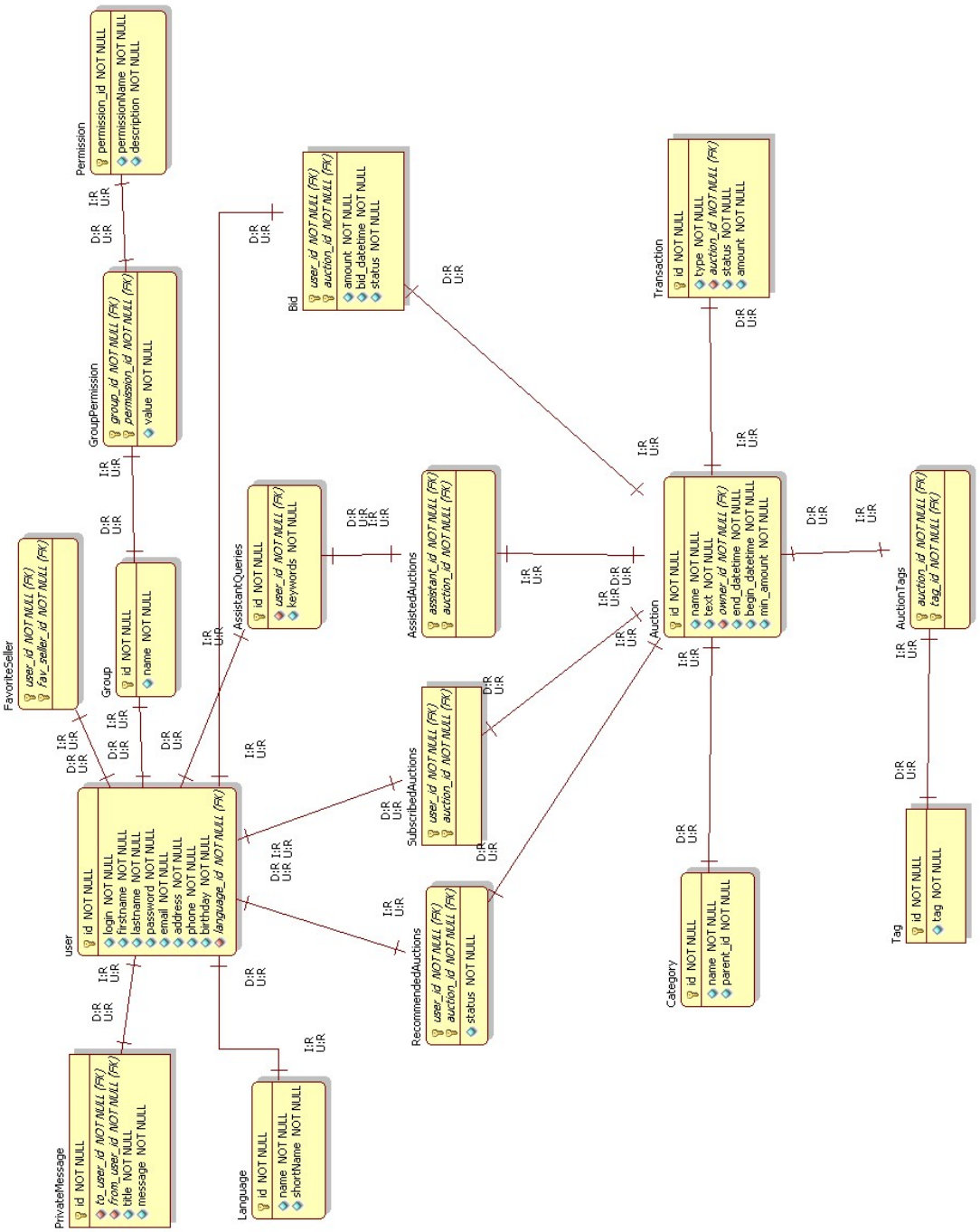

Figure B.8: Salesmen database layout

Appendix C

Website Prototypes

C.1 Auction list

Figure C.1: auctionlist.seam

C.2 Auction detail

Figure C.2: auction.seam

C.3 Profile - Auctions

Figure C.3: *.seam - auctions

C.4 Profile - Buyers assistant

Figure C.4: *.seam - buyers assistant

C.5 Profile - Personal information

Figure C.5: *.seam - personal information

C.6 Profile - Account information

Figure C.6: *.seam - website information