POLITECNICO DI MILANO

Course in Computer Science and Engineering
Department of Electronics, Informatics and Bioengineering

GuessBid Project

Design Document

Authors:
Ervin Kamberoski
Pavel Gichevski
Egzon Ademi

Supervisor:
Dr. Elisabetta Di Nitto

May 15, 2015

Contents

List of Figures

1 Introduction

The Design Document is a document to provide documentation which will be used to aid in software development by providing the details for how the software should be built. Within the Design Document we will be including textual and graphical documentation of the software design for the project including ER-diagrams, logical diagrams, sequence diagrams, collaboration models, object behavior models, and other supporting requirement information. We will try to find the right balance in putting each of these ingredients inside the document, with the aim to make the document a real reference in the development of the software, and more importantly its maintenance.

1.1 Purpose

Narrowing down what was previously said, the purpose of the Design Document is to provide a complete description of the design of a system to allow for software development to proceed with an understanding of what is to be built and how it is expected to be built. It provides detailed information about functionalities of the system by identifying objects, and interaction among them. Also, most of the steps are supported by visual diagrams, to remove any doubt that may be left unclear from the textual descriptions. Rephrased, it should serve as a reference on the description-to-code conversion process.

1.2 Scope

This document will provide detailed specifications for designing, implementing, and managing the GuessBid web application, but will not include end-user documentation for the last one, hence, it should be added as a support document to one.

1.3 Terminology

- **API.** In computer programming, an application programming interface (API) is a set of routines, protocols, and tools for building software applications.

- **Java EE.** Java Platform, Enterprise Edition is the standard in community-driven enterprise software. Java EE is developed using the Java Community Process, with contributions from industry experts, commercial and open source organizations, Java User Groups, and countless individuals. Each release integrates new features that align with industry needs, improves application portability, and increases developer productivity.

- **Tier.** In general, a tier (pronounced TEE-er ; from the medieval French tire meaning rank, as in a line of soldiers) is a row or layer in a series of similarly arranged objects. In computer programming, the parts of a program can be distributed among several tiers, each located in a different computer in a network. Such a program is said to be tiered , multitier , or multitiered .

- **JDBC.** The Java Database Connectivity API is the industry standard for database-independent connectivity between the Java programming language and a wide range of databases ? SQL databases and other tabular data sources, such as spreadsheets or flat files. The JDBC API provides a call-level API for SQL-based database access.

- **Java Beans.** JavaBeans are classes that encapsulate many objects into a single object (the bean). They are serializable, have a 0-argument constructor, and allow access to properties using getter and setter methods. The name Bean was given to encompass this standard which is meant to create reusable software components for Java.

- **Entity** An entity is a long-lived, passive element that is responsible for some meaningful chunk of information. This is not to say that entities are "data," while other design elements are "function." Entities perform behavior organized around some cohesive amount of data.

- **Boundary** A boundary element lies on the periphery of a system or subsystem, but within it. For any scenario being considered either across the whole system or within some subsystem, some boundary elements will be "front end" elements that accept input from outside of the area under design, and other elements will be "back end," managing communication to supporting elements outside of the system or subsystem..

- **Control** A control element manages the flow of interaction of the scenario. A control element could manage the end-to-end behavior of a scenario or it could manage the interactions between a subset of the elements. Behavior and business rules relating to the information relevant to the scenario should be assigned to the entities; the control elements are responsible only for the flow of the scenario.

2 Design Overview

Before going into details about our systems, we want to provide short overview of the platform JEE, which will be used for the development. The platform provides an API and runtime environment for developing and running enterprise software, including network and web services, and other large-scale, multi-tiered, scalable, reliable, and secure network applications. Java EE extends the Java Platform, Standard Edition providing an API for object-relational mapping, distributed and multi-tier architectures, and web services.

2.1 JEE architecture

For clear understanding we provide a concept of the architecture, and afterward will follow an explanation for various components.

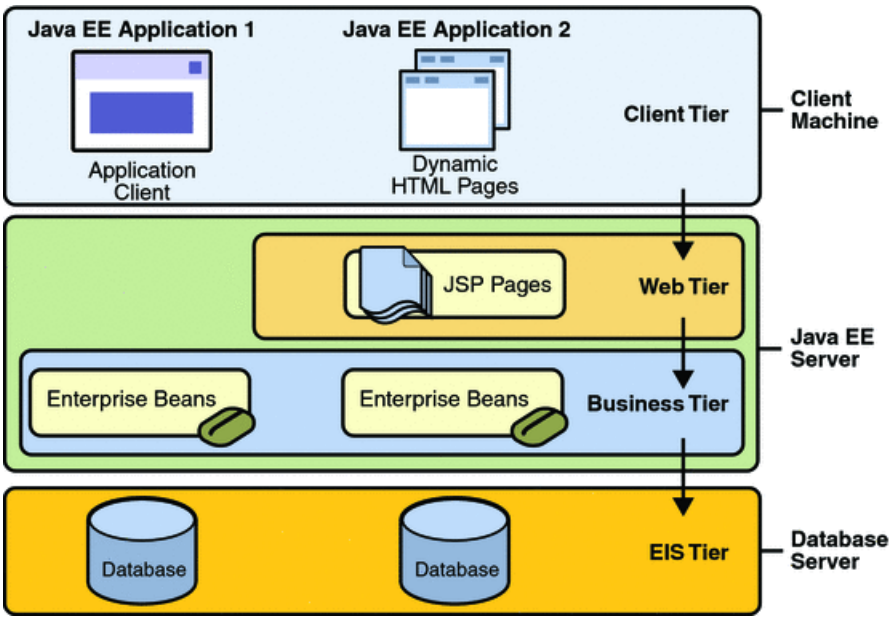

Figure 1: JEE architecure concept.

From the figure above, we can see that the architecture is based on 4-tier model, and has a layered structure, in order to separate different types of

components so that the interaction among them is managed according to the needs. This fact is useful because it reduces the amount of work needed to perform, and increases the reliability of the developed software. Let's now go into details about the four different tiers.

2.1.1 Client Tier

The client tier consists of application clients that access a Java EE server and that are usually located on a different machine from the server. The clients make requests to the server. The server processes the requests and returns a response back to the client. Many different types of applications can be Java EE clients, and they are not always, or even often Java applications. Clients can be a web browser, a standalone application, or other servers, and they run on a different machine from the Java EE server.

2.1.2 Web Tier

The web tier consists of components that handle the interaction between clients and the business tier. Its primary tasks are the following:

- Dynamically generate content in various formats for the client.
- Collect input from users of the client interface and return appropriate results from the components in the business tier.
- Control the flow of screens or pages on the client.
- Maintain the state of data for a user's session.
- Perform some basic logic and hold some data temporarily in JavaBeans components.

2.1.3 Business Tier

The business tier consists of components that provide the business logic for an application. Business logic is code that provides functionality to a particular business domain, like the financial industry, or an e-commerce site. In a properly designed enterprise application, the core functionality exists in the business tier components.

2.1.4 EIS Tier

The enterprise information systems (EIS) tier consists of database servers, enterprise resource planning systems, and other legacy data sources, like mainframes. These resources typically are located on a separate machine than the Java EE server, and are accessed by components on the business tier.

3 Data Management

The following section will provide a description of how the data will be managed in our system. We anticipate that, for storing the information we will use relational data model, which will be supported by MySQL database management system. Also, we will provide information in incremental manner, starting from conceptual model, construct the logical one, and finally get the logical table-like representation.

3.1 Conceptual Model

The conceptual diagram is visual description of the talking work we did previously. We identified the main components(entities) of the system and the interactions(relations) among them. We used UML standard to represent the concepts and some more details were added, like multiplicity, thanks to its expressive. This diagram will be the foundation stone for the whole system, in the sense that from it, we will be able to integrate more and more details. Hence we are using a sort of bottom up approach, a clear analogy with the buildings construction process.

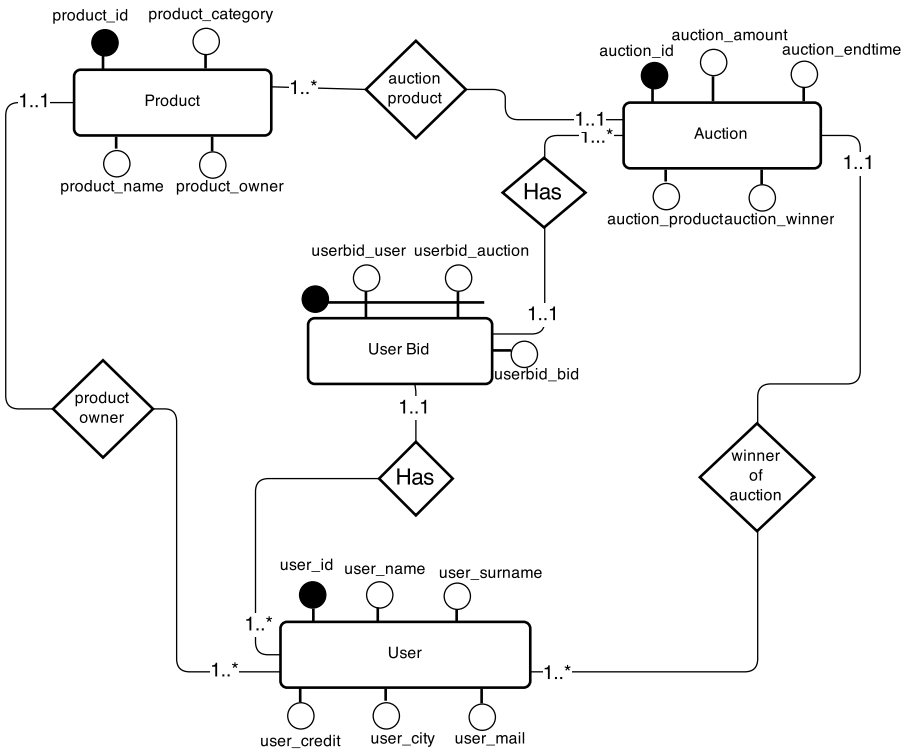

Figure 2: Conceptual model of the database.

The conceptual diagram is pretty simple and easy to understand, but still we will provide simple facts that can be extracted from the conceptual diagram, that hold in our system , with the aim to make things clearer.

- A *User* can create many (possible zero) *auctions* and to each of them a specific product is linked. Automatically, the user is the owner of this auction.
- The *Bid* primary key is consisted of the primary keys of *User* and *Auction*.
- The Winner of each auction is only one user, and it is determined when the auction is finished.
- An *Auction* can have a single creator. This is easily managed by assigning incremental unique id's to auctions in the order they are created.
- Each *Auction* has only one *Product*.
- Each *User* in the beginning has 100 credits.

3.2 Logical Model

Before providing the actual schema, we are going to provide some of the steps that we needed to perform in the process of translating the conceptual schemata to a logical one. To make an example, a relation one-to-many creates a foreign key, for each of the many rows of one relation to point to the one of the related table. Now let's see the specific transformations we did in our case.

- Relation *productOwner* that is constructed from relations *User* and *Product* is translated by adding a foreign key attribute *product_owner* into the Product relation.
- Relation *auctionProduct* that is constructed from relations Product and Auction is translated by adding a foreign key attribute *auction_product* into the *Auction* relation.
- Relation *winnerOfAuction* that is constructed from relations User and Auction is translated by adding a foreign key attribute *auction_winner* into the Auction relation.

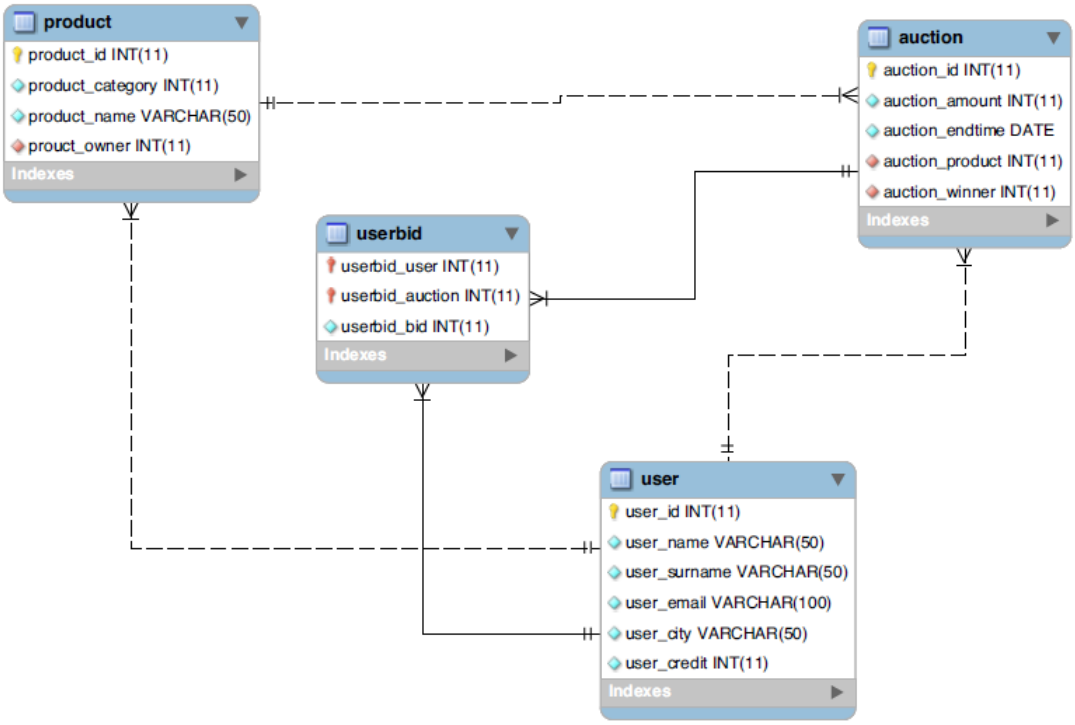

Figure 3: The translated logical model.

We finally obtain the following relations:

- **User** (user_id,user_name, user_surame, user_username, user_email, user_city, user_credit)
- **Auction** (auction_id, auction_amount, auction_endtime, *auction_product*, *auction_winner*)
- **Product** (product_id, product_name, product_category, *product_owner*)
- **Bid** (*bid_by_user*, *bid_for_auction*, bid_amount)

4 Objects Description

In this section we will try to identify the main objects of our system, as classes. For each of them we will provide the attributes and the methods they have. Attributes are not a difficult achievement, as they are directly read from the diagrams in the previous sections, while methods do require some thinking about the type of the relations the two objects in question may have.

We want to state clearly that we won't be providing any code in specific programming language. First of all because the design document should provide a general approach to the realization of the system. Next, because to understand the code would take much more than comprehending a simple sentence which says what a method does. Finally, because providing code now, will be lot less efficient in terms of time and cost resources, as we should think to implement exactly every system component, and from the other we reduce the robustness of the document itself, making it applicable for a specific platform only.

At this point of time, we have identified the following main classes: *Auction*, *User*, *Product* and *Bid*. We don't need to explain anything about them, because below we provide a full description of each of them, defining their attributes and methods. We are by no means stating that the system will have only these five classes, instead we are pointing out that those are the more important ones to describe the functions of the system.

4.1 User Class

Attribute	**Attribute description**
Name	Name of the user.
Surname	Surname of the user.
Username	A nickname that the user uses to log into the system.
Credit	Amount of the virtual credit that the user possesses.
City	The city where the user lives in.
Method Name	**Method Description**
GetUserName()	A method that will return the name of a user.
SetUserName()	A method that will set a name to a user object.
GetUserCity()	A method that returns the living city of the user.
SetUserCity()	A method that sets the city where the user lives.
GetUserSurname()	A method that will return users surname.
SetUserSurname()	A method that will set a surname to a user object.
GetUserUsername()	A method that will return users username.
SetUserUsername()	A method that sets the username of a user object.
GetUserCredit()	A method that will return the amount of credit that the user has.
SetUserCredit()	A method that will change the user virtual credit.
CreateAuction()	A method that creates a new auction owned by the current user.
BidToAuction()	A method that allows the user to make a bid to a chosen auction.

4.2 Auction Class

Attribute	**Attribute description**
Auction product	The product that is listed on the auction.
Auction end time	The ending time of the auction the owner entered.
Auction description	Product description that is listed in the auction.
Auction amount	The current bid that is on the auction.
Auction winner	After the auction ends, this attribute will hold the name of the winner.
Method Name	**Method Description**
GetOwner()	A method that will return the owner of an auction.
GetAuctionParticipants()	A method that will list participants of an auction.
GetAuctionProduct()	A method to get a product in an auction.
SetAuctionProduct()	A method to set a product in an auction.
GetProductType()	A method to set/get a product type.
GetAuctionDescription()	A method to get auction description.
SetAuctionDescription()	A method to set auction description.
GetAuctionAmount()	Method to get the current best amount of an auction.
SetAuctionAmount()	Method to set the current best amount of an auction.
GetAuctionWinner()	Method to get who the winner of a particular auction is.
SetAuctionWinner()	Method to set who the winner of a particular auction is.

4.3 Product Class

Attribute	**Attribute description**
Product Name	Attribute that holds the name of a product in an auction.
Product Owner	Owner of the auction that listed the product.
Product Category	Category on which a particular product belongs.
Method Name	**Method Description**
GetProductName()	Method to find the name of a product in an auction.
SetProductName()	Method to set the product name, when new products are inserted.
GetProductOwner()	A method to get the name of the product owner.
SetProductOwner()	A method to set the name of product owner.
GetProductCategory()	A method used to determine products category.
SetProductCategory()	A method used to assign the product to a category.
GetProductAuction()	A method that gets the auction in which the product is offered.

4.4 Bid Class

Attribute	**Attribute description**
UserBid	An attribute that holds the current bid a user placed in an auction.
Method Name	**Method Description**
GetBidUser()	A method that will return user that participated on an auction according to the bid.
GetBidAuction()	A method that will return an auction according to a bid.
GetSetBidAmount()	A method used to allow a user to place a bid on auction, or to find the amount of the bid a particular user placed.

5 Dynamic Model

In this section we will provide the dynamic model of our system. We will accomplish this by providing various diagrams to describe the interaction between objects and other identified components of the system. In the first part we will provide a series of navigation model from the user experience point of view. To be able to complete the task, we first needed to identify the various system components and classify them into stereotypes. We used $<< Screen >>$ to describe a whole static state of the web application, a $<< Window >>$ to describe something that is contained in the previous (e.g. pop up window) and $<< Form >>$ a series of fields where the user is able to input data.

In the second part we will provide a series of sequence diagrams, and in this we will be using the **Boundary-Control-Entity** of the system. There is a description for each of them in the *Terminology* section, but to remind the reader:

- **Entity.** Object representing system data, often from the domain model.
- **Boundary.** Object that interface with system actors (e.g. a user or external service). Windows, screens and menus are examples of boundaries that interface with users.
- **Control** Objects that mediate between boundaries and entities. These serve as the glue between boundary elements and entity elements, implementing the logic required to manage the various elements and their interactions.

5.1 Navigation Models

5.1.1 Log In

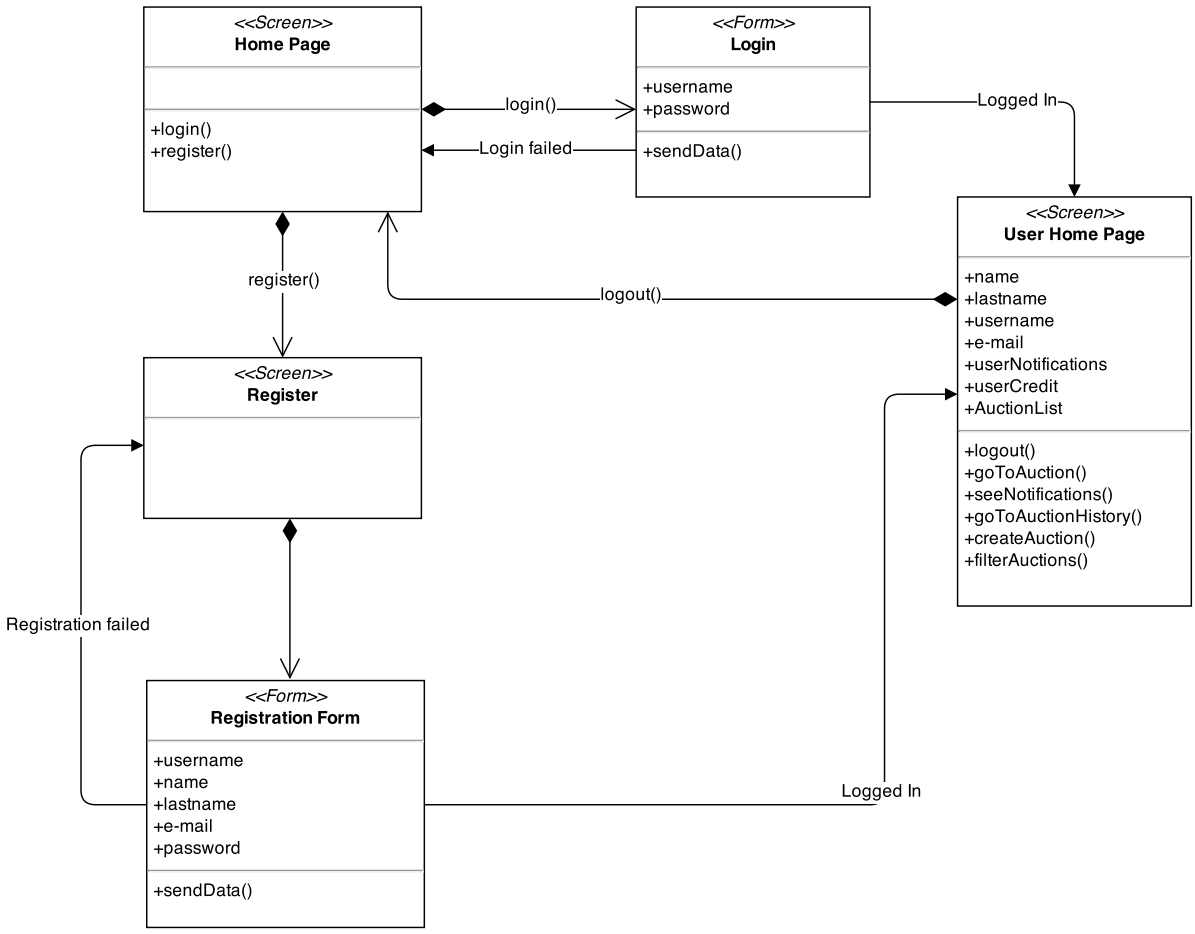

Figure 4: Navigation model for login.

5.1.2 Auction Management

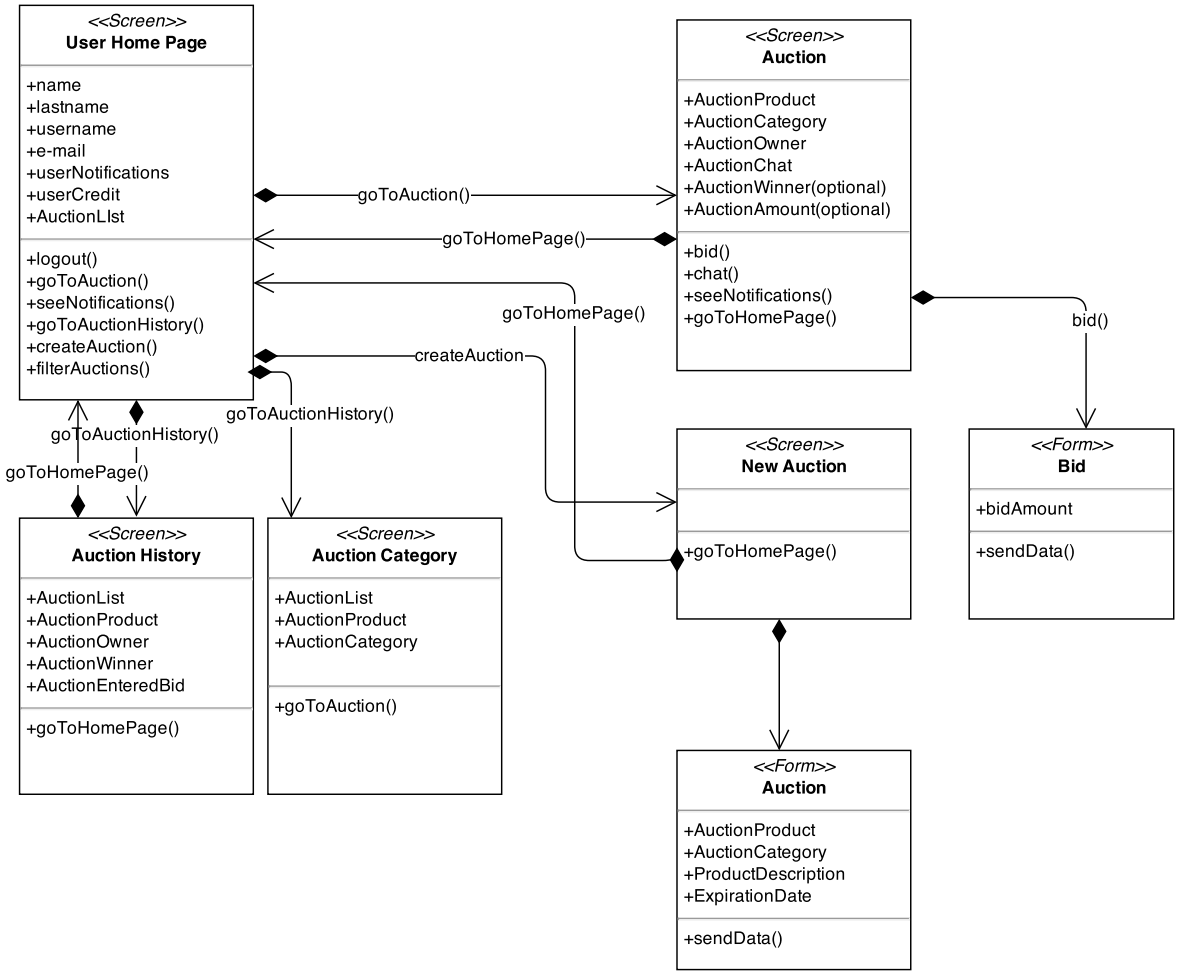

Figure 5: Auction management sequence diagram.

5.1.3 Auciton Chat

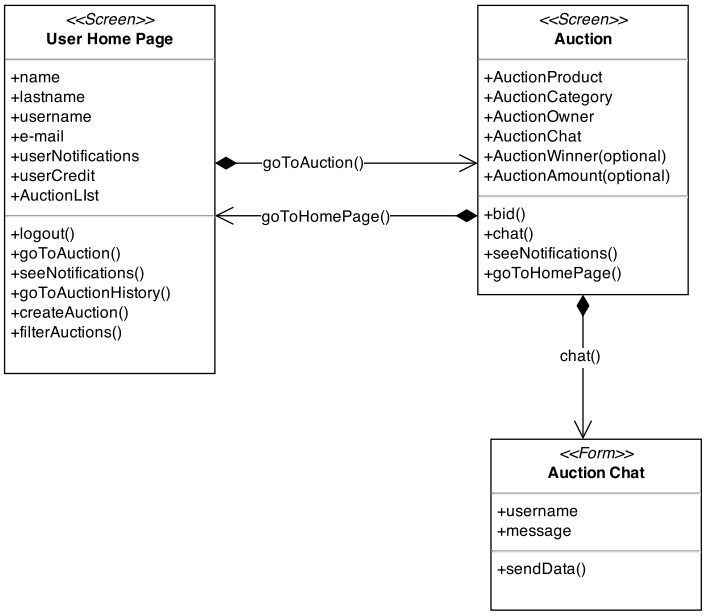

Figure 6: Navigation model for chatting in auction.

5.1.4 Notification System

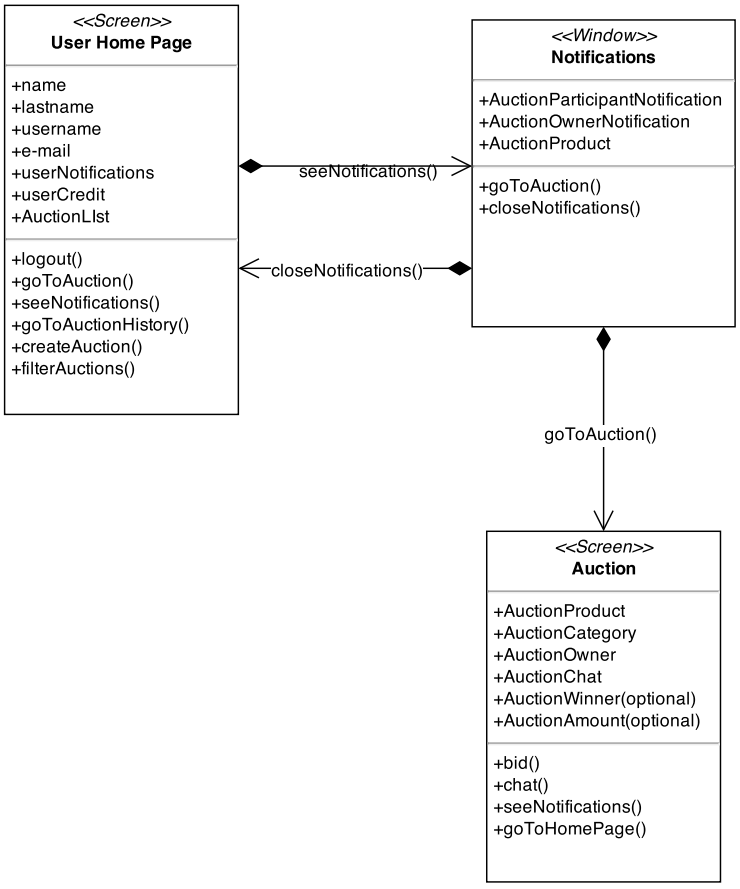

Figure 7: Navigation model for seeing notifications.

5.2 Sequence Diagrams

5.2.1 User Login

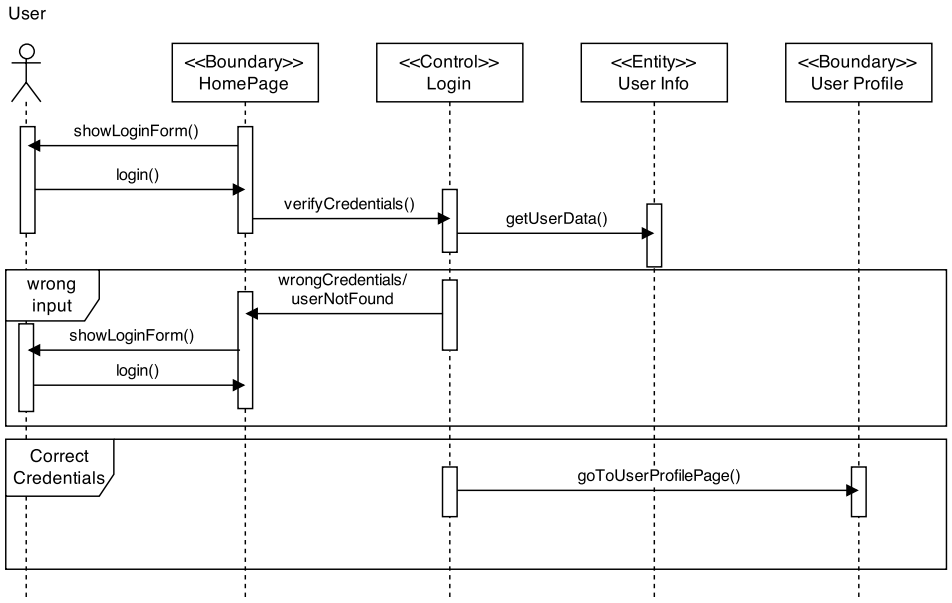

Figure 8: Login sequence diagram for the user.

5.2.2 Create Auction

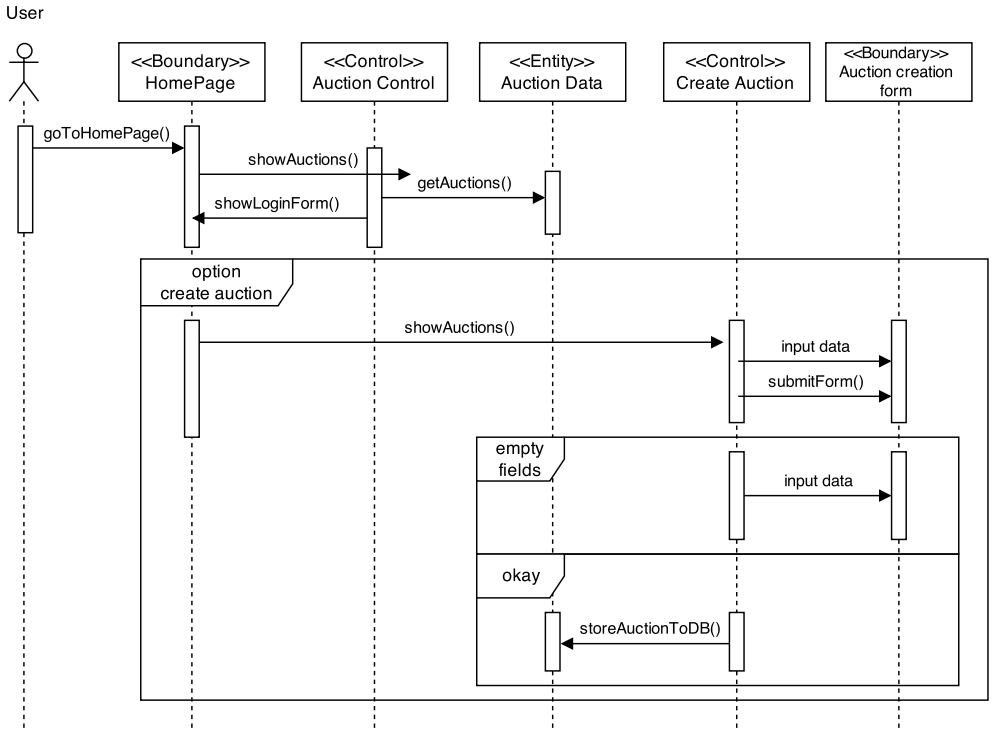

Figure 9: Auction creation sequence diagram.

5.2.3 Auction Bid

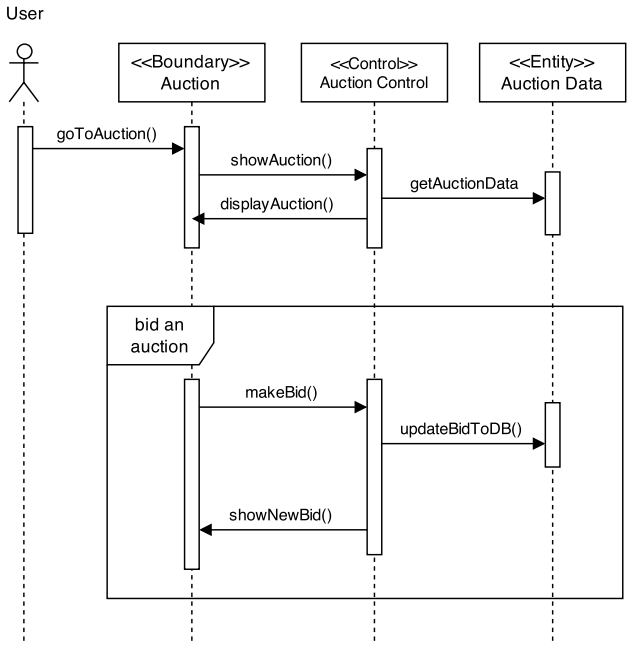

Figure 10: Sequence diagram bidding to an auction.

5.2.4 Notification System

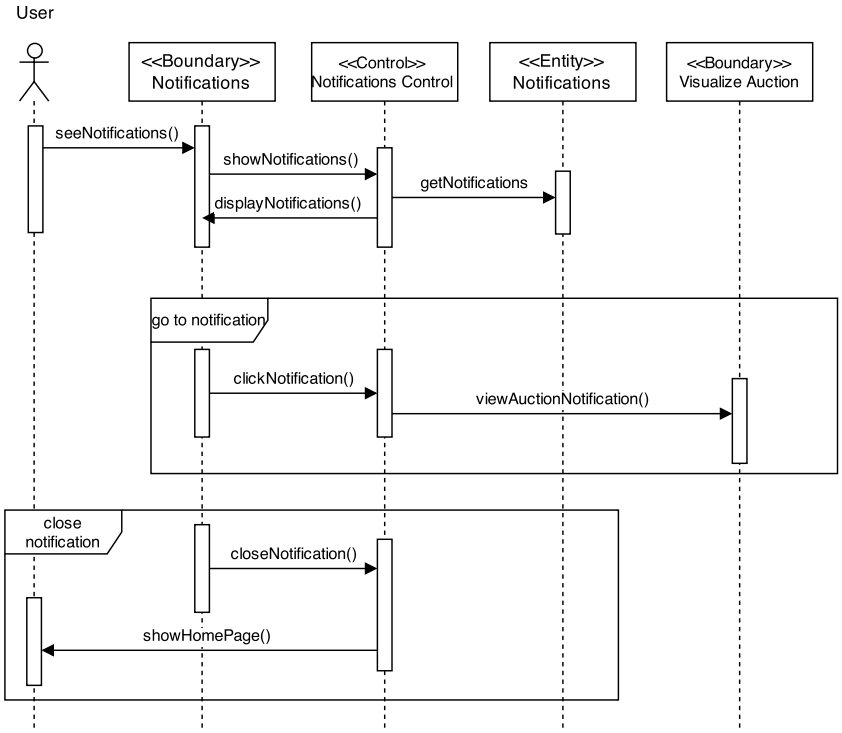

Figure 11: Sequence diagram for auction notifications.

6 Used Tools

1. LaTeX
2. Visio 2013
3. MySQL Workbench
4. www.draw.io