

**Politecnico Di Milano**

Facoltà di Ingegneria dell' Informazione

(Computer Science Engineering)

SOFTWARE ENGINEERING II PROJECT

Part I: **DD**

Prof.ssa: **Di Nitto**

**A.A. 2014/15**

Project Title: **BidWin**

Project Repository: **https://github.com/EmanueleLM/BidWin**

**Emanuele La Malfa Davide Malvestiti**

**M**atr: **841042 M**atr: **773345**

**emanuele.lamalfa@mail.polimi.it davide.malvestiti@mail.polimi.it**

**TABLE OF CONTENTS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1 / 18**

**1.System Architecture. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2**

* 1. **Brief Introduction. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2**
  2. **System Tiers. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3**
  3. **The Enterprise Java Beans. . . . . . . . . . . . . . . . . . . . . . . . . . 4**

**2.Database Definition. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5**

**2.1 Conceptual Project. . . . . . . . . . . . . . . . . . . . . . . . . . . . 5**

**2.1.1 ER Model. . . . . . . . . . . . . . . . . . . . . . . . 5**

**2.2 Logical Schema. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5**

**2.2.1 Logical ER. . . . . . . . . . . . . . . . . . . . . 5**

**2.2.2 Foreign Key. . . . . . . . . . . . . . . . . . . . 5**

**2.2.3 Tables. . . . . . . . . . . . . . . . . . . . . . . . . 7**

**3.Model. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8**

**3.1 Class Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8**

**3.2 UX Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9**

**3.2.1 Navigation Diagram . . . . . . . . . . . . 10**

**3.2.2 Analysis Diagram. . . . . . . . . . . . . . 11**

**3.2.3 Navigation + Analysis. . . . . . . . . . .12**

**3.3 Sequence Diagrams. . . . . . . . . . . . . . . . . . . . . . . . .13**

**Index Images. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18**

**1.System Architecture**

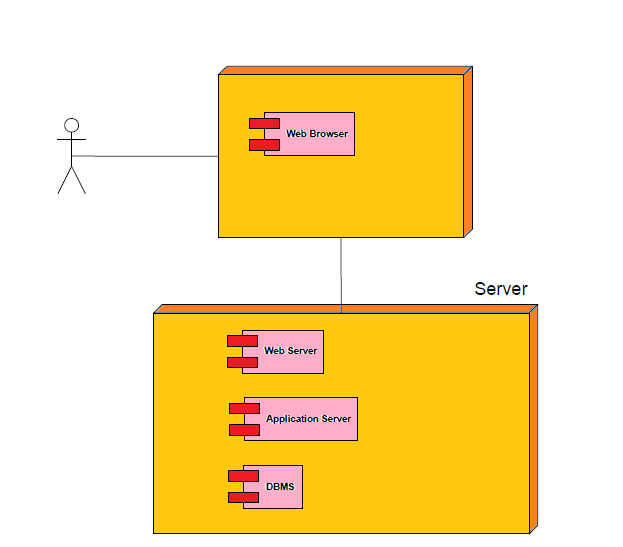
* 1. **Brief Introduction**

We develop the system by using the Java EE Platform: since the final result is a web application, we rely on the most typical structure composed by a - central - server, several clients and a database.

We will deepen these concepts in the further section, seeing how the single components are subdivided and how they collaborate in order to offer the required functionalites: moreover we present the technologies we will use to manage each of the tiers.

Since we are in a distributed scenario, we have to guarantee that our system is reliabale and robust: thus we keep the components separated as much as possible - Java EE natively supports the separation between tiers-.

By far we can present a simplified overview of our architecture:



Pic#1: appserver.png

* 1. **System Tiers**

Now we can go deeper into our system implementation, specifying what kind of technologies we use for each of the tiers.

**Application Server**

An application server is a program resident on a machine server which offers a set of services - in this case the Java EE services - which are the core of our application.

The main features are:

* Static and dinamic presentations to the final user, thanks to web-based interfaces
* Managing business logic components
* Access to the data

We use the Oracle Glassfish Server 4, which comes with a prepared community that helps solving a great variety of problems that usually arise in this distributed scenario.

**Web Application Logic**

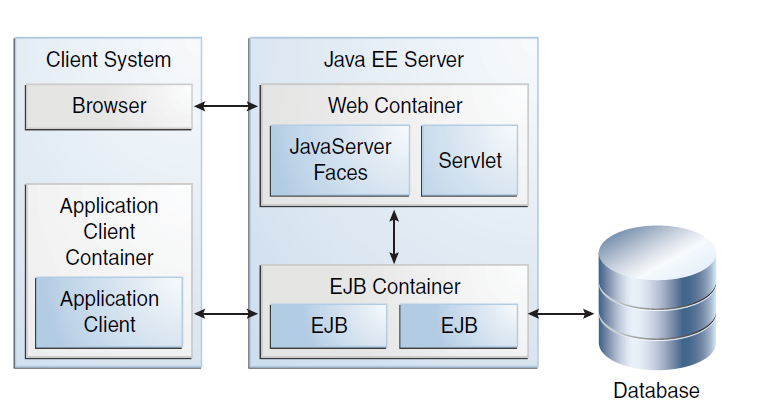
The web part is meant to allow interacting with the system in an easy way: we will shape it by the JSF - Java Server Faces -.

A JSF is composed by a Faclet and a Bean: the Faces Servlet manages the http requests as specified in the Facelets files - which are substantially markup text -. This markup includes bindings between the html elements and methods of the Beans.

Please notice that these technologies mix up java code and html markup in an MVC manner.

Historically there’s another way to approach this web application part, by the JSP method: we will not use it because since the JSF took place, it has been marked as deprecated by both the community and Oracle.

Picture #2 represent the same system as in the picture #1, but in this case it is focused on the technologies implemented in each part.



Pic#2: appserver\_tech.png

* 1. **The Enterprise Java Beans**

In the next session we present the Enterprise Java Beans - EJB -, which are inflected in three diferent ways: each of them offers a specific service which is standalone, modular and independent form the other Beans.

Please note that this does not mean that we can’t combine 2 different EJB: actually we use them togheter. In the end of the section we provide a meaningful example where we combine them.

1. **Session Beans:** they are EJB dedicated to the business logic, which carry out operations such as arithmetic calculation and so on. For example if a user needs to recharge his/her pocket, he performs the operation by filling up a form in a specific page: the calculation of the new credit count is executed by a Session Bean.
2. **Entity Beans:** they are generally used to represent the data - which is a relational one in our system, but nothing forbids from using a different kind of database. As regards our system, the will be used to interact with a lot of information: please take a loot at the ER model presented in the next session.
3. **Message Drive Beans:** they are EJB dedicated to the excahnge of asynchronous messages - usually dispatched form the system to the user(s) -. E.g. a specific Message Drive Beans will notify the user when an auction ends up by sending him/her informations about the final result - the winner, the final price etc. -.

**2.Database Definition**

**2.1 Conceptual Project**

ER diagram represent the logical structure of the database where we memorize

the information, whereas the class diagram the logical structure of the data that

we use with our application.

As regard this section we provide the data structure of the system: follows the

UX diagrams, which will help to describe the logical structure of whole the system.

As mentioned before, we use a entity relation database which is accessed by an Entity Beans, but nothing forbids from using - in the future versions - a non relational database.

**2.1.1 ER Model**

Picture #3 shows the **ER model**.

Please Notice that Auction is a weak-entity: this means that it cannot exists without the related object.

**2.2 Logical Schema**

**2.2.1 Logical ER**

Here we provide the logical schema related to the previous ER model.

Primary keys are **bold and underlined,** foreign key are *italic:* these

information are mixed up if necessary.(e.g. **BID**: user and auction are both

primary and foreign keys).

USERS(**Username**, Password, Name, Surname, Email, Rank, PaymentInformations, Birthdate, AuctionCounter, Credits, Address );

OBJECTS(**Object-id**, *Username*, ObjectName, ObjectType, Description, ImageLink);

AUCTION(**Auction-id**, *Object-id*, StartTime, EndTime);

BID(***Username***, ***Auction***, Value);

**2.2.2 Foreign Keys**

In this section we provide the foreign keys related to the previous ER schema.

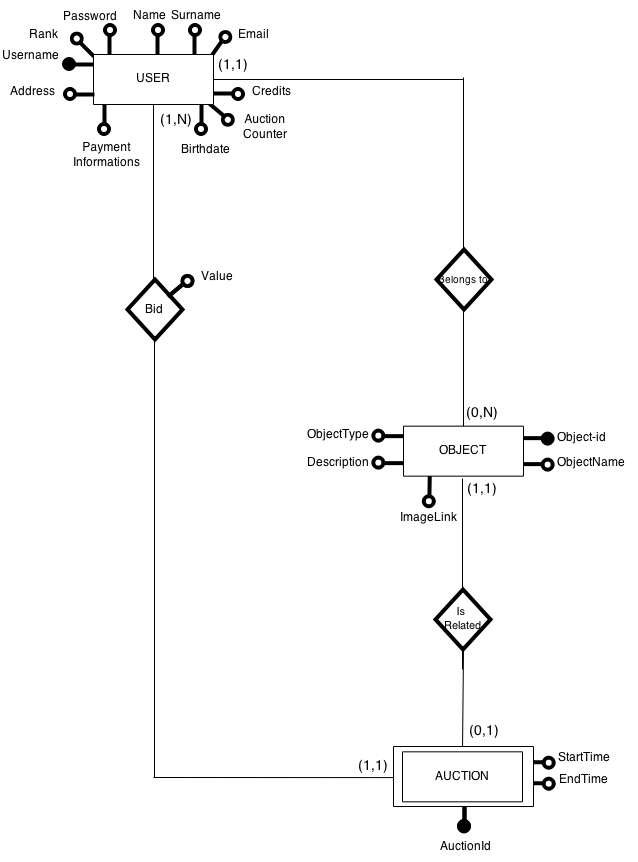
**Bid** is a relation (“connector”) which uses 2 foreign keys: that's because the relation between **Auction** and **Users** could be simply represented in this way without any meaning loss.

BID.UserName 🡪 USERS.Username

BID.Auction-id 🡪 AUCTION.Auction-id

OBJECTS.UserName 🡪 USERS.UserName

AUCTION.Object-id 🡪 OBJECTS.Object-id



Pic#3: DesignDocumentER.png

**2.2.3 Tables**

Tables are the most significant representation of our data: the information are stored

on the system in this way, and can be easily retrieved with some query language (as

SQL). This is the syntax used:

|  |
| --- |
| FieldName  Type (INTEGER, VARCHAR, DATE …)  Relation (Foreign key, Primary key, Foreign + Primary) |

**USER**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Username  VARCHAR  Primary key | Password  VARCHAR | Name  VARCHAR | Surname  VARCHAR | Email  VARCHAR | Rank  INTEGER |
| Address  VARCHAR | Payment  Information  VARCHAR | Auction Counter  INTEGER | BirthDate  TIME | Credits  INTEGER |  |

**OBJECT**:

|  |  |  |
| --- | --- | --- |
| Object-id  VARCHAR  Primary key | Username  VARCHAR  Foreign key | ObjectName  VARCHAR |
| Description  VARCHAR | ObjectType  VARCHAR | ImageLink  VARCHAR |

**AUCTION**:

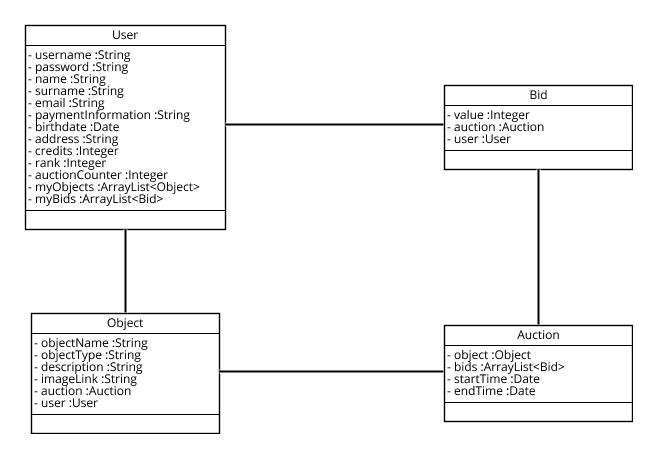
|  |  |
| --- | --- |
| AuctionId  VARCHAR  Primary key | ObjectId  VARCHAR  Foreign key |
| StartTime  TIME | EndTime  TIME |

**BID**:

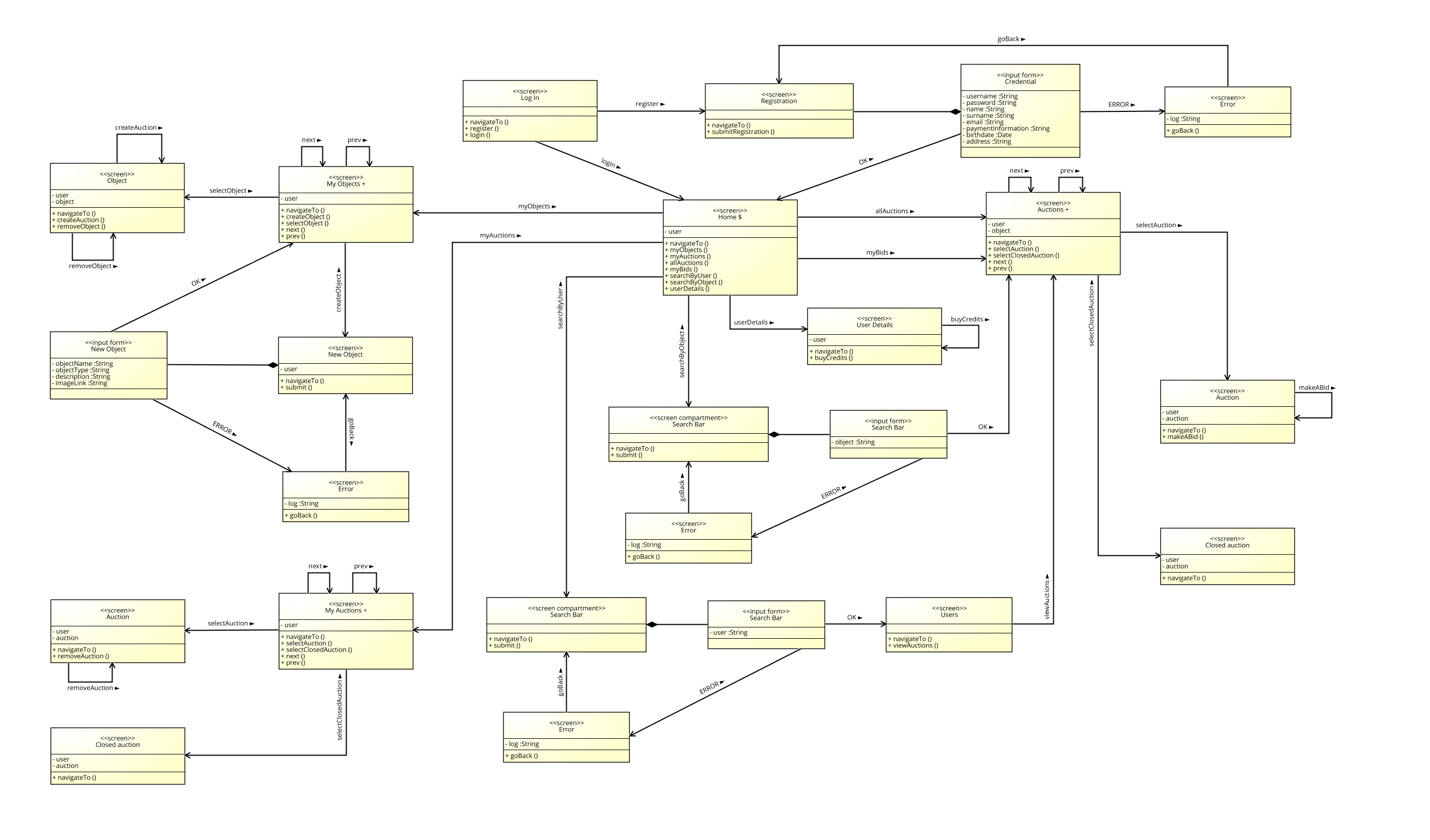
|  |  |  |
| --- | --- | --- |
| Username  VARCHAR  Primary key + Foreign key | AuctionId  VARCHAR  Primary key + Foreign key | Value  INTEGER |

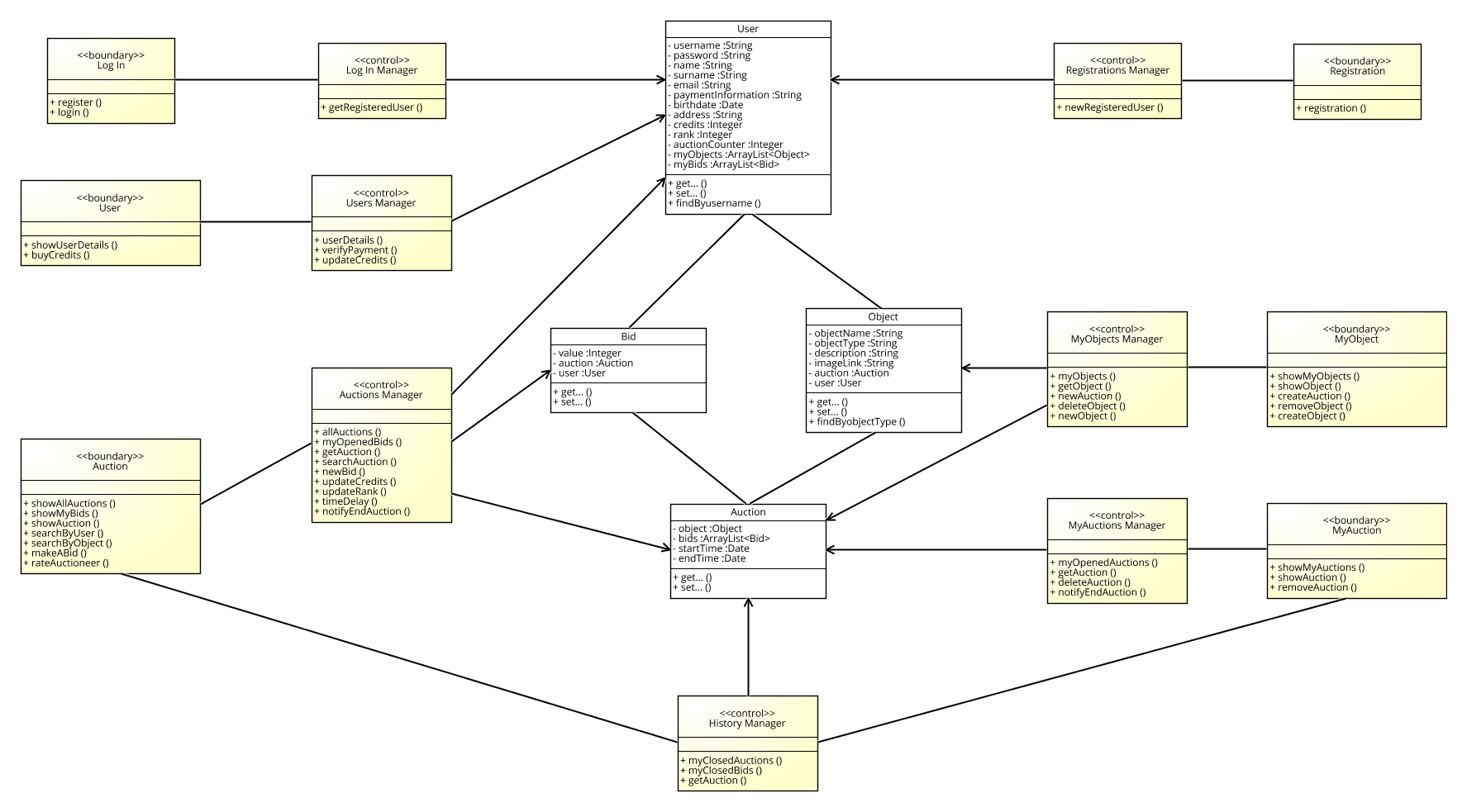
**3.Model**

**3.1 Class Diagram**

****Pic #4: ClassDiagram.png

See picture #7

****Pic #5: UxDiagram.png

****Pic #6: BCEDiagram.png

****

Pic #7: UX-BCE.png

**3.3 Sequence Diagrams**

In this section we provide several sequence diagrams which are the same presented in the RASD document: this time they refers to the BCE diagram - see the previous section -.

**#1: User Registration:** this seq. Diagram shows how a “*non-registered user”* interacts with the registration form provided by the system itself: first of all he/she has to perform a registration request, than he/she inserts a valid nickname - which will have not already been adopted by another user - a password, and a list of information such an address where the delivery pack will be sent.

If all the operations are executed correctly, the registration is successfully performed and he/she becomes a “*registered user”.*

## Picture #2 shows the use case diagram.

**#2: Bid\_auction:** this seq. Diagram shows how a “*registered user”* interacts with the *“bidding system”* functionalities provided by the system: first of all he/she needs to looks for an object - we already discussed about the search engine -, than he will choose a specific Auction from the list. Now it is up to the user to make or not an offer.

If the system accepts the offer - action time not ended and many others constraints -he will be notified whether or not he/she has won.

We describe the winning scenario, which includes the notification of the winning, the shipment of the object and the Auctioneer evaluation.

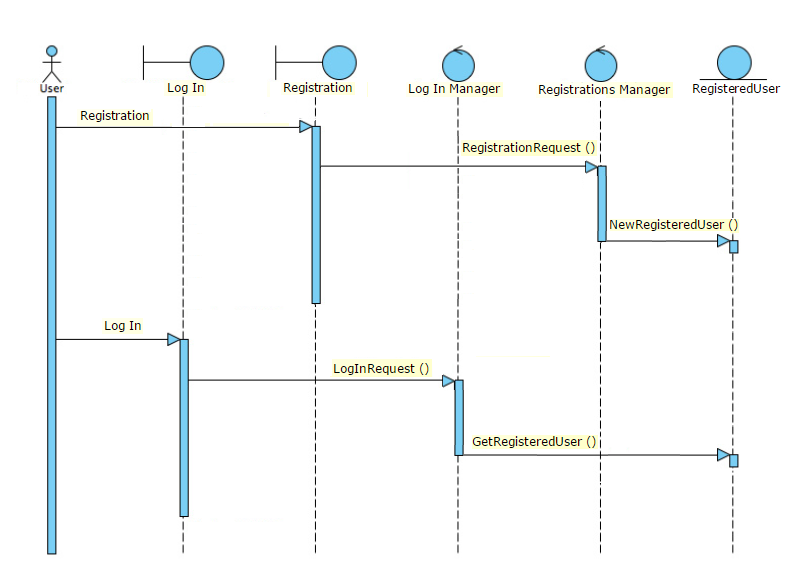
## Picture #4 shows the use case diagram.

**#3: Create Auction:** this seq. Diagram shows how a “*registered user”* interacts with the *“auction creation”* functionality provided by the system: first of all he/she has to choose an object, than he will get the object details and a menu where he can create the Auction by specifying the duration.

## Picture #4 shows the use case diagram.

**#4: Recharge\_pocket:** this seq. Diagram shows how a “*registered user”* interacts with the *“pocket”* functionalities provided by the system: first of all he/she needs to navigate to the user profile, than he/she will insert some values into an input form: these values are the method that he/she would like to use to buy credits, how much of them he/she wants to purchase and some other stuff related to the security side - e.g. the credit card number and its secure code -.

## Picture #5 shows the use case diagram.

****

