POLITECNICO DI MILANO
SOFTWARE ENGINEERING 2

Design Document

Mirjam Škarica

Milan, May 2015

Contents

Chapter 1

Introduction

1.1 Purpose

The purpose of this document is to provide a comprehensive description of the structure of the GuessBid system. It will state and analyze the design decisions made in order to satisfy all the requirements stated in the Requirements Analysis and Specification Document (RASD). This document is meant mainly as a guideline for developers of the software in question.

1.2 Acronyms

The following are some acronyms and their corresponding terms used throughout the document:

BCE	Boundary-Control-Entity
CSS	Cascading Style Sheets
DB	Database
DBMS	Database Management System
EIS	Enterprise Information System
ER	Entity–Relationship
HTML	Hypertext Markup Language

HTTP	Hypertext Transfer Protocol
JEE	Java Platform Enterprise Edition 1
JMS	Java Message Service
JSP	JavaServer Pages
MCV	Model View Controller
RASD	Requirements Analysis and Specification Document
UML	The Unified Modeling Language
UX	User Experience

Chapter 2

System Architecture

2.1 JEE architecture overview

Developing a system using Java Enterprise Edition (JEE) is one of the requirements imposed by the client. Having that in mind, an overview of JEE architectures is given bellow (fig 2.1).

JEE follows the distributed multi-tiered application approach which means the entire application may not reside at a single location, but is distributed. JEE is divided into four tiers:

Client tier runs on the client machine and provides a dynamic interface to the middle tier, JEE server (fig 2.1) by interacting directly with users and communicating with the aforementioned server. The client tier distinguishes two types, application client and web client. The former being a standalone desktop application, and the latter usually a web browser. GuessBid will be implemented as a web client.

Web tier runs on the JEE server and comprises of JavaSever Pages (JSP) and Java Servlets. The basic idea follows. A servlet receives an HTTP requests from the client tier and forwards the data to the business tier. After receiving a response from the business tier, dynamic web pages are generated using JSP and are sent back to the client.

Business tier runs on the JEE server and contains the application's logic. It processes data received from the client and data retrieved from the database (DB) in order to send a response back to the client. There are three types of business components in the JEE architecture:

Session Beans represent a session with a client. Being a transient object, they lose their data when the session terminates

Java Persistence Entity Beans are persistent objects and retain data even after the session. (e.g. they represent a row of data in a database table).

Message-Driven Beans are used for receiving the Java Message Service (JMS) messages asynchronously.

Enterprise Information System (EIS) tier runs on the Database Server and is responsible for storing and retrieving all persistent data.

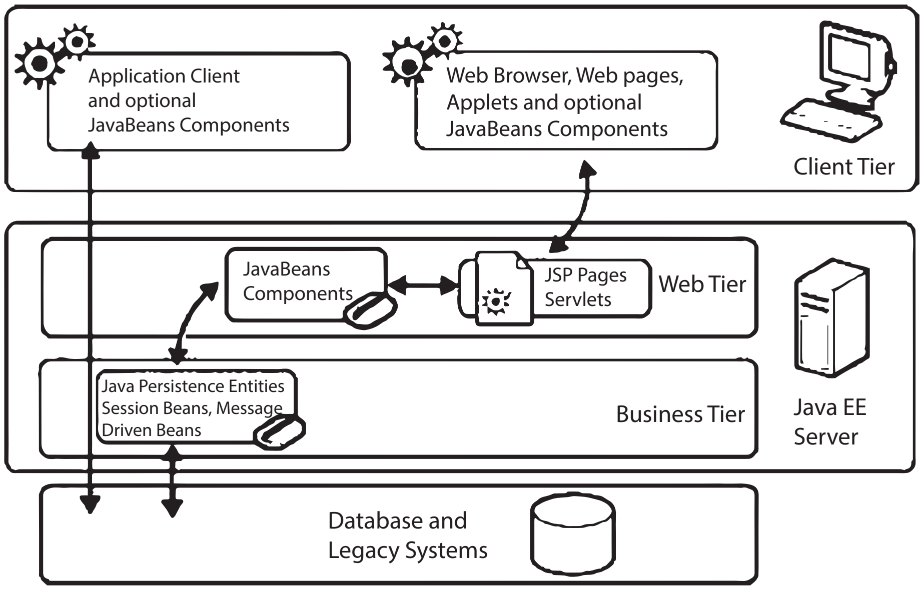

Figure 2.1:

2.2 Identifying subsystems

GuessBid's system is broken down into smaller subsystems by using a top down approach. This is done in order to distinguish logically separate components so their role in the entire system is more understandable, making their functionality easier to identify and implement. Subsystems (fig 2.2) are derived from the functional requirements stated and described in the RASD document.

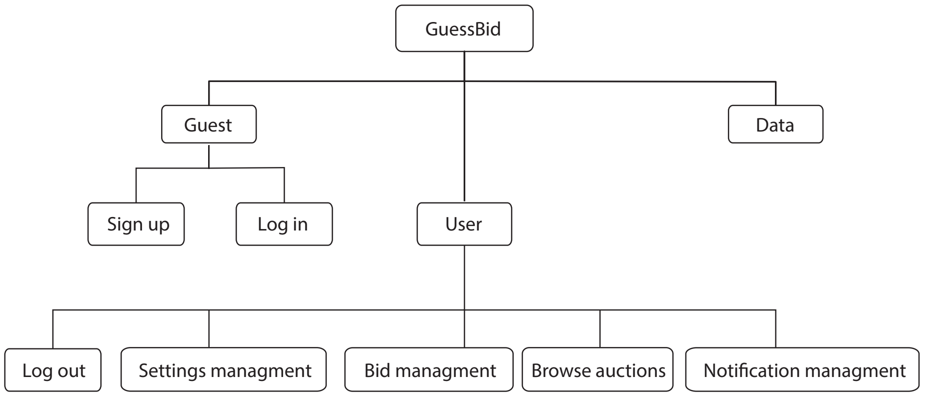

Figure 2.2:

Brief description of each subsystem's functionality is given below:

Guest subsystem allows clients access to the application:

- **Log in** allows existing users to access the application
- **Sign up** allows new clients to register and log in

User subsystem allows logged in clients (users) the full use of GuessBid's system's functionalities:

- **Log out** allows logged in users to log out
- **Browse bids** provides ability to search all active auctions
- **Bid management** allows users to create bids and check the status of own active bids
- **Notification management:** receive notifications (rank change or auction outcome)
- **Setting management:** allows users to change their settings, such as email or password.

Data subsystem is the subsystem where all the persistent data is stored (like the bid information)

Chapter 3

Persistent data management

GuessBid's system's data will be stored in a relational database. Different diagrams of the DB schema are provided in order to identify and gain a deeper understanding of the system's underlying physical structure.

3.1 Conceptual design

Conceptual view of system's data communicates a clear picture regarding the entities we want to store and relationships between. The entity-relationship diagram, fig 3.2, is derived from Class diagram stated in the RASD document. Some minor changes in the reference to the class diagram have been introduced for sake of simplicity. They will be pointed out in the short explanation of the ER diagram bellow.

Entities:

Bid stores all information regarding a placed bid

User stores all information regarding a single user

Auction stores all information about an auction

Category stores all information about a category. Even though the class diagram shows a *has-a* relationship between *Auction* and *Category*, to keep the design of the application as simple as possible, Category will be modeled as a separate entity to keep track of all possible types, but Auction will not link to it. Rather, it will have a field of type String denoting which type it is.

Notification stores all information regarding a notification. The class diagram suggests an inheritance relationship between *RankNotification* and *OutcomeNotification* and should therefore be modeled using a disjoint constraint (fig 3.1). However, since they differ minimally, instead of the constraint, a boolean attribute *is_ outcome* is added to the *Notification* entity in order to differentiate the two types of notifications.

Relations:

User has Notification each user can have zero or more notifications, each notification is associated with exactly one user

Notification linksTo Event each notification links to exactly one auction, but each auction can produce zero or more notifications

User creates Auction each user can create zero or more auctions, but each auction can have only one seller (creator)

User places Bid each user can place zero or more bids, each bid is placed by(associated with) exactly one user

Bid linksTo Auction each bid is linked to exactly one auction, a single auction can have zero or more bids

Bin won Auction each auction has zero or one winning bids, each bid is a winning bid for zero or one auctions

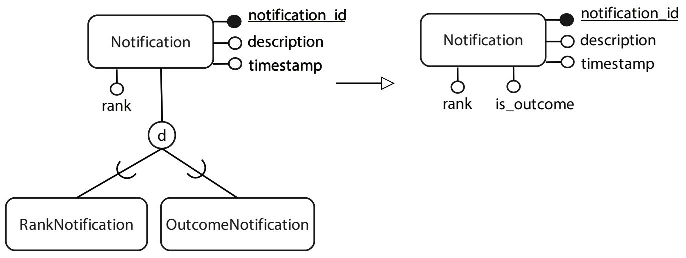

Figure 3.1:

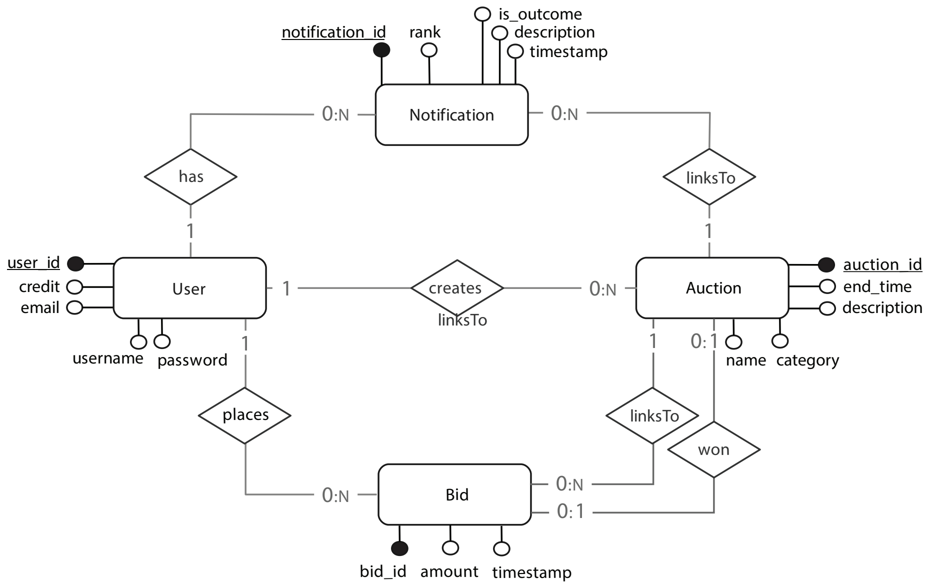

Figure 3.2:

3.2 Logical Design

The logical view (fig 3.3) of system's data is the table schema of its database. It is derived by translating the ER diagram by applying the following transformations:[1]

Entities:

Each entity turns into a table

Each attribute turns into a column in the table

The primary key of each entity becomes the primary key of the table

[1]Reference material::

- T.Haigh, teaching material `http://www.tomandmaria.com/tom/Teaching/Drexel210/translation_hints.htm`
- Connolly, Begg, Database Systems: A Practical Approach to Design, Implementation and Management

Relations:

- **1:N relation:** take the primary key of the relation on the "1" side and set it as a foreign key for the table on the "N" side
- **1:1 relation:** there is a choice which table will receive the primary key of the other as its foreign key. Usually, if the dependent entity can be determined, it receives the foreign key
- **N:N relation:** each such relation turns into a separate table and the keys of the two entities are set as the primary key

The model obtained after this process is:[2]

Table name	**Attributes**
USER	user_id, credit, email, username, password
BID	bid_id, bidder_id, auction_id, amount, timestamp
AUCTION	auction_id, seller_id, winning_bid_id, name, description, end_time, category, timestamp
NOTIFICATION	notification_id, user_id, auction_id, description, is_outcome, timestamp
CATEGORY	category_id, type

Also a decision was made to add the following views to the DB schema in order to simplify the process of querying data thus making the application more efficient:

Virtual Table name	**Attributes**
ACTIVE_AUCTIONS	auction_id, seller_id
FINISHED_AUCTIONS	winner_id, auction_id
WINNING_BID	bid_id, bidder_id, bid_auction_id

[2]Primary keys are denoted by underlining the attribute name

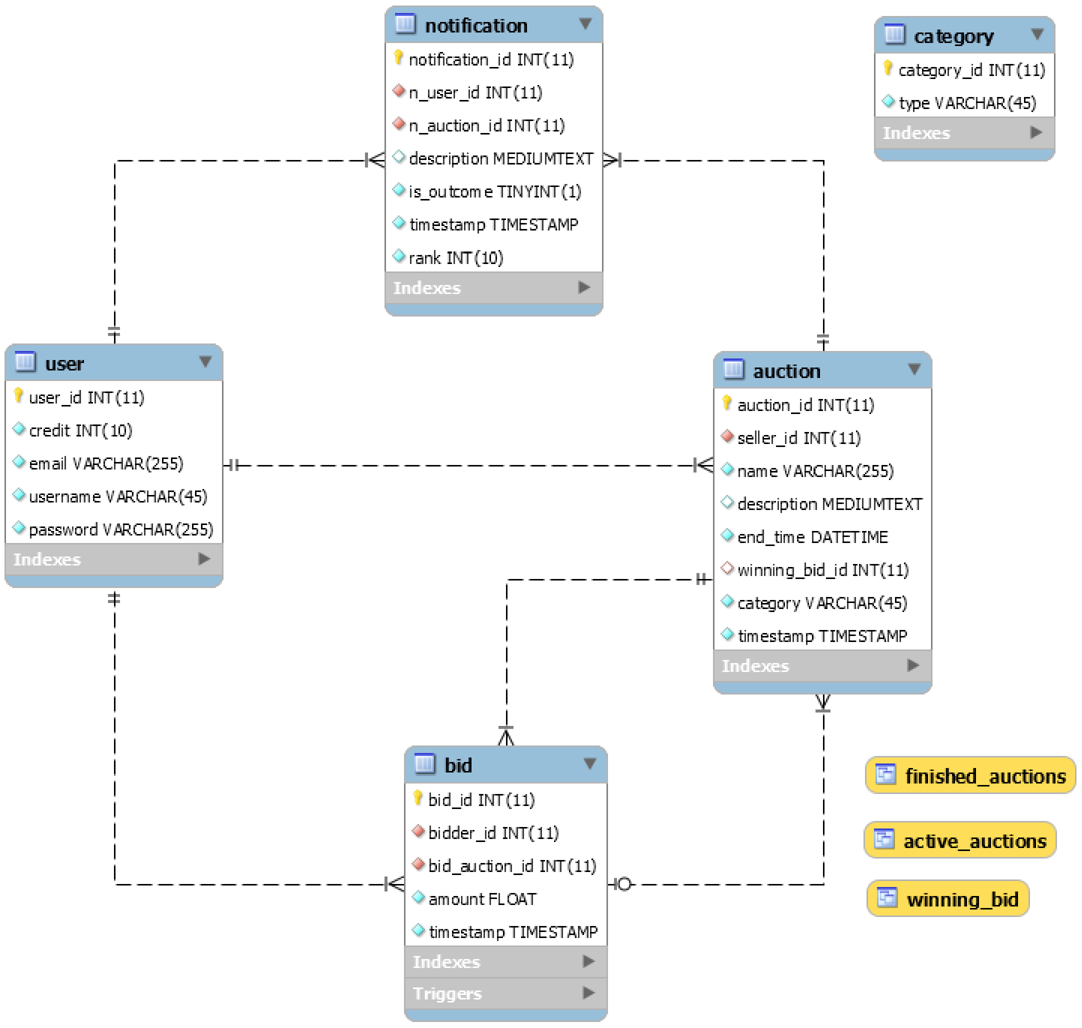
notification
notification_id INT(11)
n_user_id INT(11)
n_auction_id INT(11)
description MEDIUMTEXT
is_outcome TINYINT(1)
timestamp TIMESTAMP
rank INT(10)
Indexes
category
category_id INT(11)
type VARCHAR(45)
Indexes
user
user_id INT(11)
credit INT(10)
email VARCHAR(255)
username VARCHAR(45)
password VARCHAR(255)
Indexes
auction
auction_id INT(11)
seller_id INT(11)
name VARCHAR(255)
description MEDIUMTEXT
end_time DATETIME
winning_bid_id INT(11)
category VARCHAR(45)
timestamp TIMESTAMP
Indexes
bid
bid_id INT(11)
bidder_id INT(11)
bid_auction_id INT(11)
amount FLOAT
timestamp TIMESTAMP
Indexes
Triggers
finished_auctions
active_auctions
winning_bid

Figure 3.3:

Chapter 4

Software design

4.1 User experience

Depending on design choices, every application has a unique way of interacting with a user. This section will depict graphically GuessBid's user experience (UX). The following stereotypes are used in all diagrams:

<<**screen**>> represents a dynamically created page that contains all other elements as well as links to other pages.

<<**screen component**>> represents a modular building element with a specific purpose. It can be attached to any part of a page, thus extending it's functionality.

<<**input form**>> is a type of <<screen component>> containing input fields for user to fill out and submit it to the system for further processing.

4.1.1 Guest Home Page

The UX diagram in fig 4.1 represents GuessBid's home page. The guest homepage consists of two smaller components, specifically, two input forms for the two different types of clients previously identified in the RASD document, the SignUp form for registering a new user, and the login form for an already registered user. After filling out the appropriate form, client submits it. If the submission was successful the client is redirected to his personal home page, otherwise he is redirected back to the guest home page and is shown an appropriate error message explaining what went wrong.

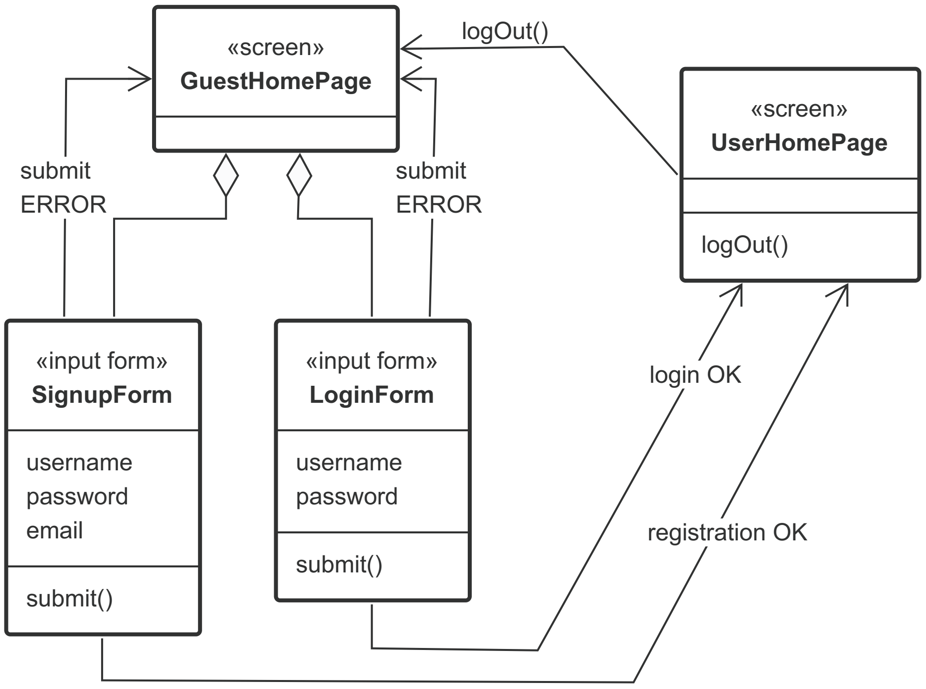

Figure 4.1:

4.1.2 User Home Page

The diagram in fig 4.2 represents user's personal home page (user being logged in is a given) from which he can access all of GuessBid's functionality like browse auctions, check notifications and manage settings. Each of these functionalities will be explained in more detail in their respective diagrams. It is important to mention that all diagrams following this one should contain the *NavigationBar* component, but it will be omitted to reduce clutter. From the *NavigationBar* user can, at any point and from any page, return to his homepage, view notifications, manage his settings and log out

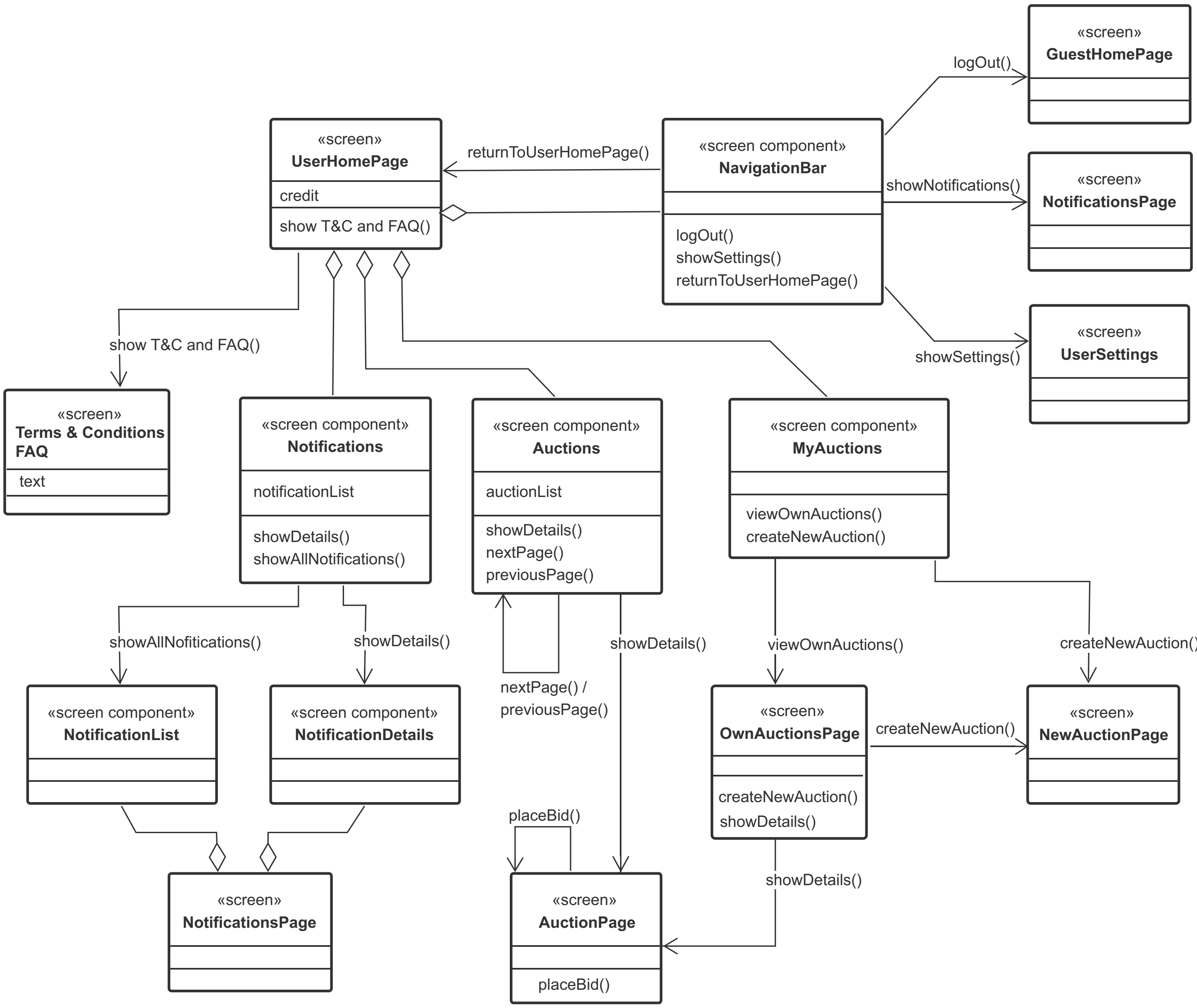
«screen»
GuestHomePage
logOut()
«screen»
UserHomePage
credit
show T&C and FAQ()
returnToUserHomePage()
«screen component»
NavigationBar
logOut()
showSettings()
returnToUserHomePage()
showNotifications()
«screen»
NotificationsPage
showSettings()
«screen»
UserSettings
show T&C and FAQ()
«screen»
Terms & Conditions
FAQ
text
«screen component»
Notifications
notificationList
showDetails()
showAllNotifications()
«screen component»
Auctions
auctionList
showDetails()
nextPage()
previousPage()
«screen component»
MyAuctions
viewOwnAuctions()
createNewAuction()
showAllNofitications()
showDetails()
nextPage() /
previousPage()
showDetails()
viewOwnAuctions()
createNewAuction()
«screen component»
NotificationList
«screen component»
NotificationDetails
«screen»
OwnAuctionsPage
createNewAuction()
showDetails()
createNewAuction()
«screen»
NewAuctionPage
placeBid()
showDetails()
«screen»
NotificationsPage
«screen»
AuctionPage
placeBid()

Figure 4.2:

4.1.3 User Settings

The diagram in fig 4.3 represents user's settings page which consists of two main components. The *UserSettings* from which he can view his username, password and email, and the *ChangeSettings* component from which he can change the two latter credentials. In order to do so, he chooses the desired change option and in the *ChangeSettings* component a corresponding input form is loaded. After filling and submitting the form, an appropriate message is displayed in the *sessionMessage* field informing the user of a successful change or asking him to fill out the form again otherwise.

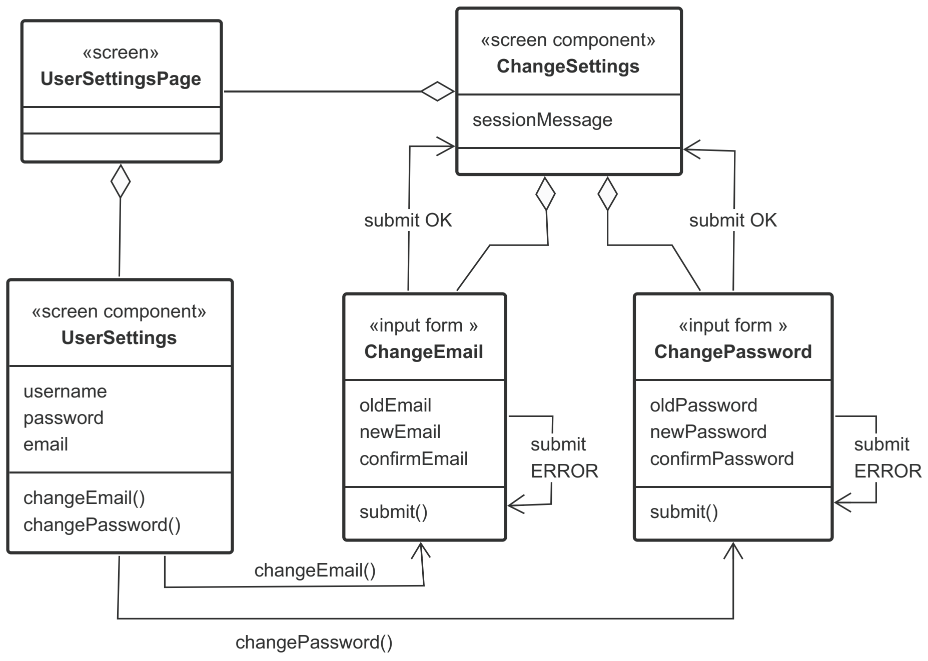

Figure 4.3:

4.1.4 Notifications Page

The diagram in fig 4.4 represents the *Notifications Page*. It consists of two main components, *NotificationList* and *NotificationDetails*. *NotificationList* lists all user's notifications which can be filtered regarding their type (rank or outcome). By selecting a specific notification its details are dynamically loaded in the *NotificationDetails* component. From the details component, user can navigate to the auction the notification is linked to.

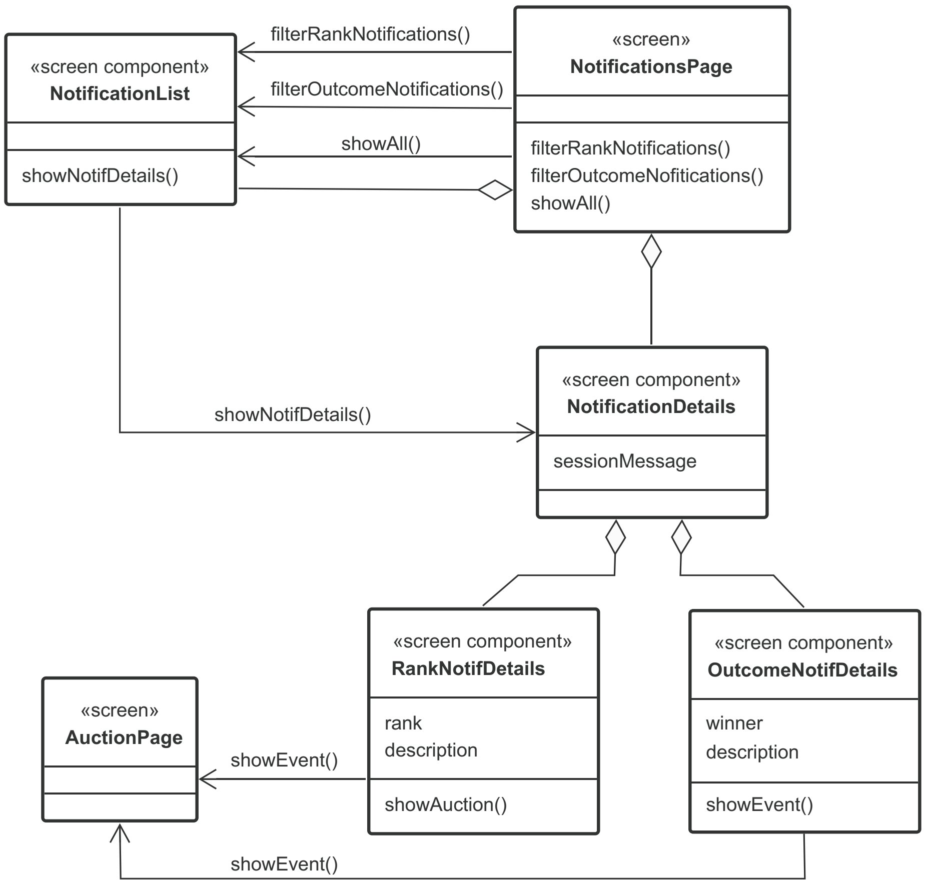

Figure 4.4:

4.1.5 Auctions Page

The diagram in fig 4.5 represents the *Auctions* component. It is a component of *UserHomePage* and it displays all active auctions in a list format. The auctions can be filtered by category, time of creation and remaining time. When clicked on a single auction, the user is redirected to *AuctionPage* containing the details of the auction in question. From that page user can place a bid by typing the desired amount int the *BidForm*, provided he is not the seller and has sufficient funds.

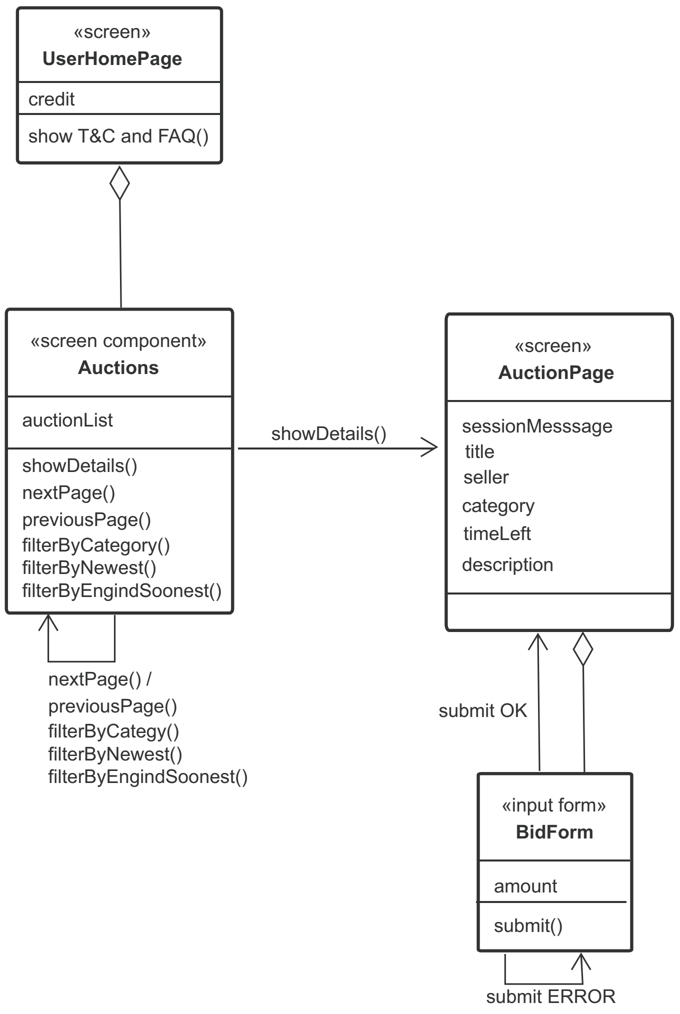

Figure 4.5:

4.1.6 Manage Auctions Page

The diagram in fig 4.6 represents the *Manage Auctions Page*. Itt displays all of users own auctions in a list format.When clicked on a single auction, the user is redirected to *AuctionPage* containing the details of the auction in question. If, on the other hand, user chooses *createNewAuction()* from the *ManageAuctionsPage* he is served a *NewAuctioPage* containing a form with all necessary fields. After user fills up the form, he submits it. If all information was valid he is redirected to the new auction's page, otherwise he is asked to fill the form again.

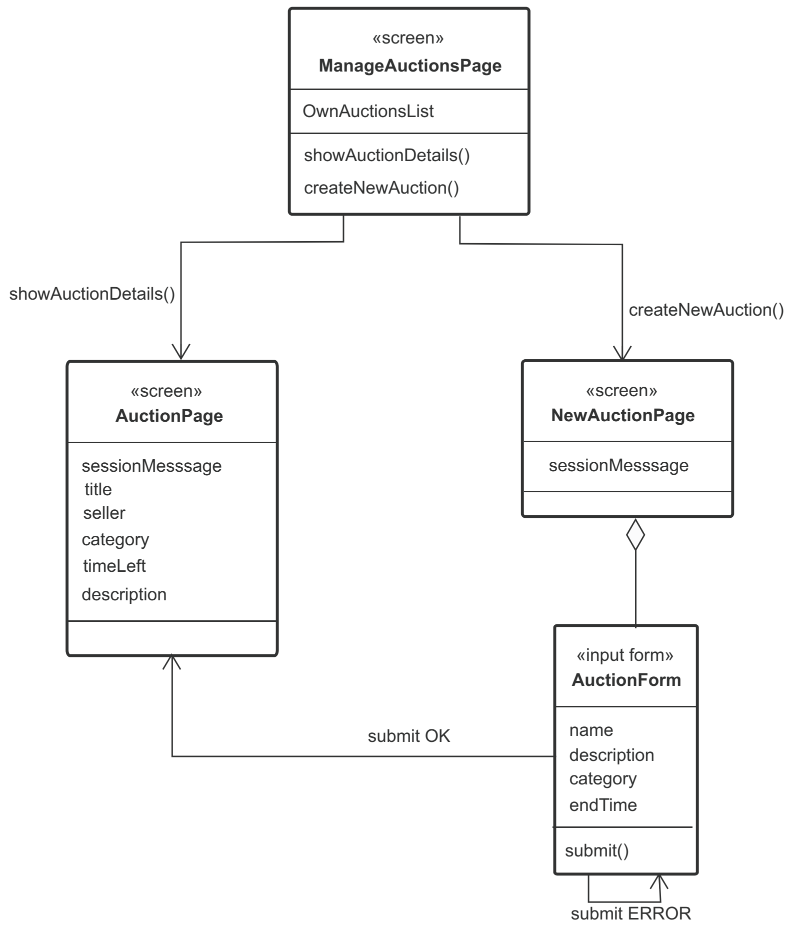

Figure 4.6:

4.2 BCE diagrams

Another informative way to model the GuessBid's system is using the Boundary-Control-Entity (BCE) Diagrams. This is done to further analyze the logic behind the application being developed. Quick explanation of the three main components is as follows:

Entity represents system's data stored in a database and it does not depend on the control or the boundary.

Boundary displays the entity data, and sends user actions (e.g. button clicks) to the control. In this case it represents <<screens>> or <<screen components>> explained in the previous chapter.

Control provides entity data to boundary, and interprets user actions such as button clicks. It depends on the boundary and the entity.

4.2.1 Guest Home Page

The Guest Home Page BCE diagram in fig 4.7 depicts the flow of events for *SignUp* and *Login* functionalities. Regarding the *SignUp* fuctionality, the user enters and submits the data. *GuestHomePage* boundary validates fields have been correctly filled and passes the data to the *SignUpController* verifies the chosen email is unique before forwarding the registration request to *UserDataController*. It creates a new user, assigns the initial credit of 100 and inserts it in the database after which the user is redirected to the *UserHomePage* boundary. On the other hand, upon *Login* form submission, *GuestHomePage* boundary first validates the entered data is in the correct format then the *LoginController* checks the existence of the supplied email in the system's database and then verifies the entered email and password match. If the information submitted was valid, the request is forwarded to UserDataController which loads all the necessary data needed, and serves the user his UserHomePage boundary. To load UserHomePage boundary *AuctionController* and *NotificationController* provide the newest auctions and notifications, respectively.

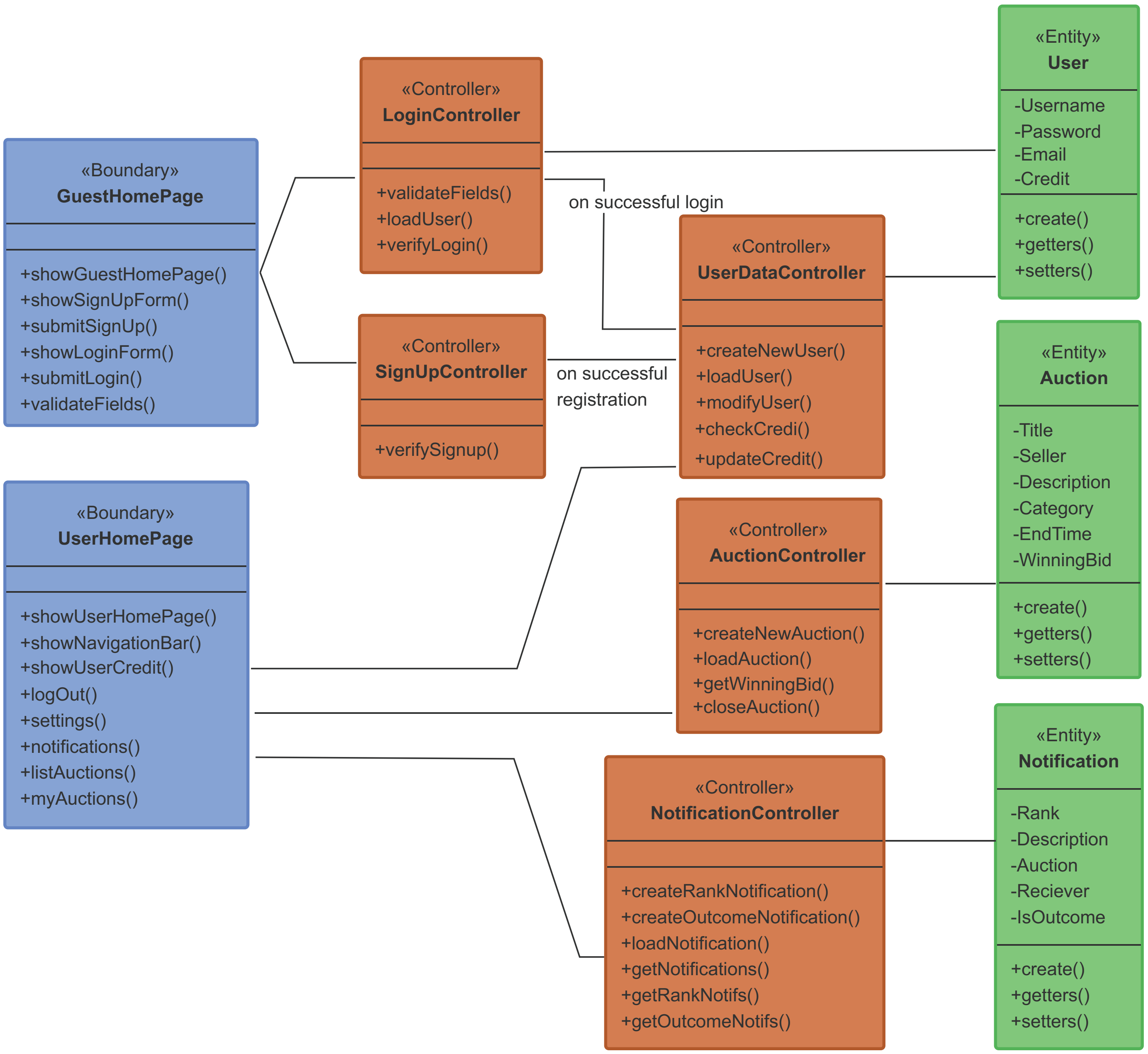
«Boundary»
GuestHomePage
+showGuestHomePage()
+showSignUpForm()
+submitSignUp()
+showLoginForm()
+submitLogin()
+validateFields()
«Controller»
LoginController
+validateFields()
+loadUser()
+verifyLogin()
on successful login
«Controller»
SignUpController
+verifySignup()
on successful registration
«Controller»
UserDataController
+createNewUser()
+loadUser()
+modifyUser()
+checkCredi()
+updateCredit()
«Entity»
User
-Username
-Password
-Email
-Credit
+create()
+getters()
+setters()
«Boundary»
UserHomePage
+showUserHomePage()
+showNavigationBar()
+showUserCredit()
+logOut()
+settings()
+notifications()
+listAuctions()
+myAuctions()
«Controller»
AuctionController
+createNewAuction()
+loadAuction()
+getWinningBid()
+closeAuction()
«Entity»
Auction
-Title
-Seller
-Description
-Category
-EndTime
-WinningBid
+create()
+getters()
+setters()
«Controller»
NotificationController
+createRankNotification()
+createOutcomeNotification()
+loadNotification()
+getNotifications()
+getRankNotifs()
+getOutcomeNotifs()
«Entity»
Notification
-Rank
-Description
-Auction
-Reciever
-IsOutcome
+create()
+getters()
+setters()

Figure 4.7:

4.2.2 Auction and Bid management

The following fig 4.8 may seem to contain a lot of concepts at one, but taking into account that making a new auction, ending it, placing a bid and receiving notifications are all intertwined actions, a decision was made to show all of them to complement their collaborative nature.

From the *AuctionList* boundary user can view a list of ongoing auctions and can choose to filer them depending on their category, time of creation, or ending time. The AuctionController handles the requests and servers the client an updated *AuctionLis*t boundary.Upon a click on a single auction in the listing, *AuctionController* loads the correct auction entity and serves user the *AuctionDetail* boundary containing the desired information. From the *AuctionDetail* boundary, user can place a bid using the *BidForm* boundary which is incorporated in the same page as the *AuctionDetail* boundary. When a new bid is submitted, *BidController* handles the request by first checking the user has enough credit via a call to *UserDataController.* If so, the *BidController* updates the DB with the new bid. After a bid has successfully been places the *NotificationController* generates rank notifications informing all bidders of that particular auction about their possible rank change.

From the *ManageAuctionsBoundry* user can choose to make a new Auction after which he is served the *AuctionForm* boundary. User fills out the form and requests to create an auction. After the boundary validates the entered data, *AuctionController* creates the auction and inserts a new record of the entity in the database. Since every auction has a user defined time limit, after the auction has ended *AuctionController* is evoked. It delegates the job of updating seller's and the winning bidder's credit to *UserDataController*, and the job of creating outcome notifications to the seller and to all bidders to *NotificationController.*

From the *NotificationList* boundary user can view a list of notifications and can choose to filer them depending on their type (outcome or rank). The *NotificationController* handles the request and servers the client an updated *NotificationList* boundary. If the user chooses to see details of a specific notification, *NotificationController* loads the correct notification entity and serves user the *NofiticationDetail* boundary containing the desired information. From the *NofiticationDetail* boundary, user to navigate to the auction in question.

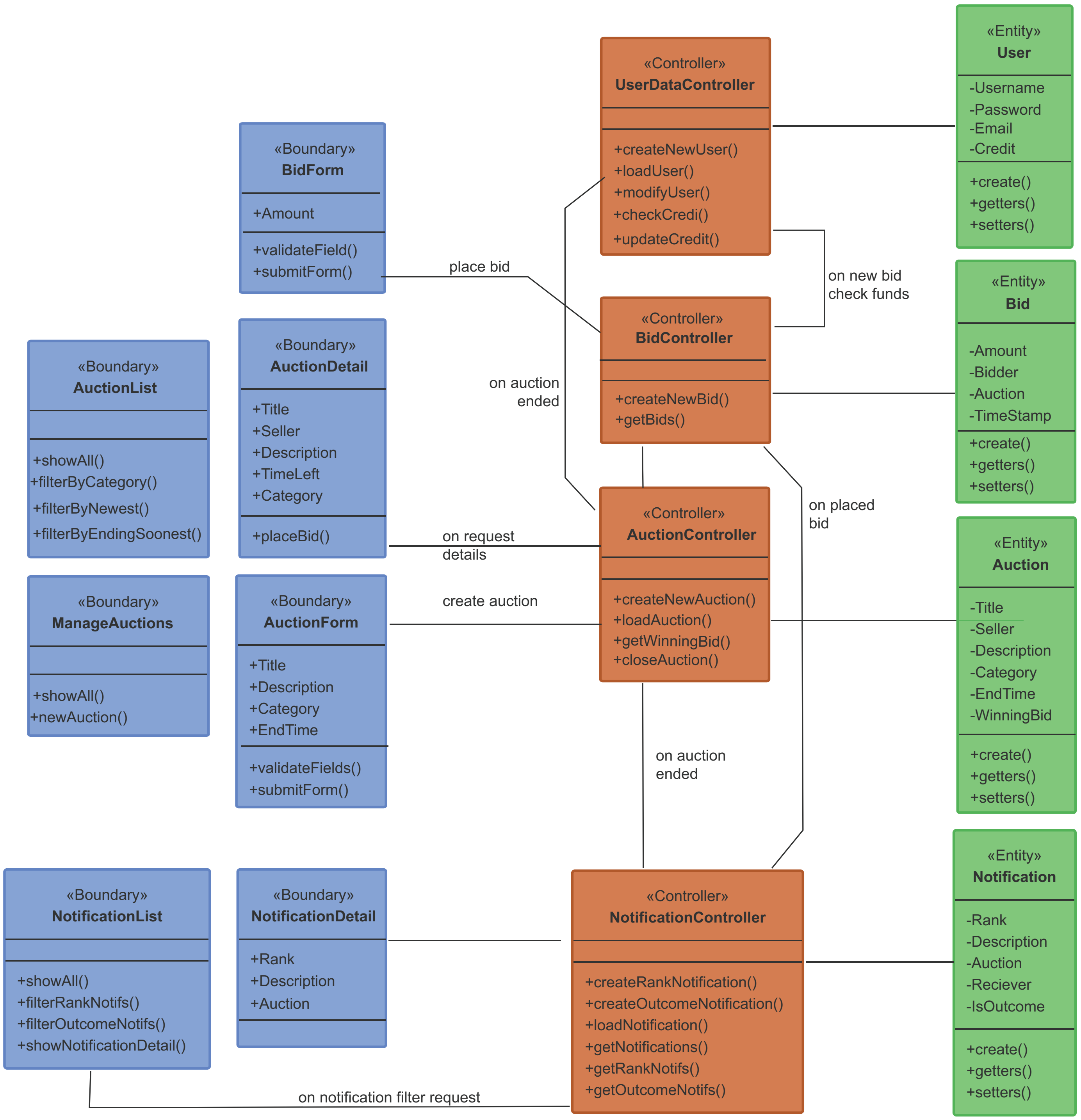
«Controller»
UserDataController
+createNewUser()
+loadUser()
+modifyUser()
+checkCredi()
+updateCredit()
«Entity»
User
-Username
-Password
-Email
-Credit
+create()
+getters()
+setters()
«Boundary»
BidForm
+Amount
+validateField()
+submitForm()
place bid
on new bid
check funds
«Controller»
BidController
+createNewBid()
+getBids()
«Entity»
Bid
-Amount
-Bidder
-Auction
-TimeStamp
+create()
+getters()
+setters()
«Boundary»
AuctionList
+showAll()
+filterByCategory()
+filterByNewest()
+filterByEndingSoonest()
«Boundary»
AuctionDetail
+Title
+Seller
+Description
+TimeLeft
+Category
+placeBid()
on auction
ended
on placed
bid
on request
details
«Controller»
AuctionController
+createNewAuction()
+loadAuction()
+getWinningBid()
+closeAuction()
«Entity»
Auction
-Title
-Seller
-Description
-Category
-EndTime
-WinningBid
+create()
+getters()
+setters()
«Boundary»
ManageAuctions
+showAll()
+newAuction()
«Boundary»
AuctionForm
+Title
+Description
+Category
+EndTime
+validateFields()
+submitForm()
create auction
on auction
ended
«Boundary»
NotificationList
+showAll()
+filterRankNotifs()
+filterOutcomeNotifs()
+showNotificationDetail()
«Boundary»
NotificationDetail
+Rank
+Description
+Auction
«Controller»
NotificationController
+createRankNotification()
+createOutcomeNotification()
+loadNotification()
+getNotifications()
+getRankNotifs()
+getOutcomeNotifs()
«Entity»
Notification
-Rank
-Description
-Auction
-Reciever
-IsOutcome
+create()
+getters()
+setters()
on notification filter request

Figure 4.8:

4.2.3 User Settings

5.2 User Settings Fig 4.9presents the BCE diagram for managing user settings. GuessBid's system gives users the liberty to change only their passwords and emails. The sequence of events is similar for both cases, so only the change password functionality will be described. After user navigates to the *ChangePasswordBoundry*, he enters and submits the required information. Using client-side validation approach, the boundary itself checks if the entered data is in a valid format before sending the request to *UserDataContoller*. The controller then modifies the table entry in the database for that user, and shows the user his *UserSetting* boundary with the changes reflected.

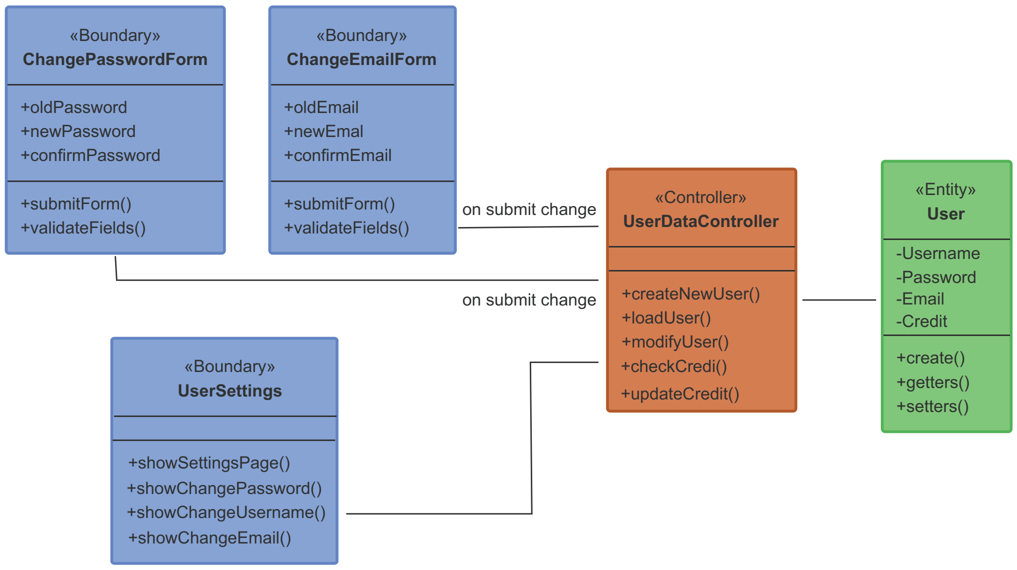

Figure 4.9:

4.3 Sequence diagrams

4.3.1 User Registration

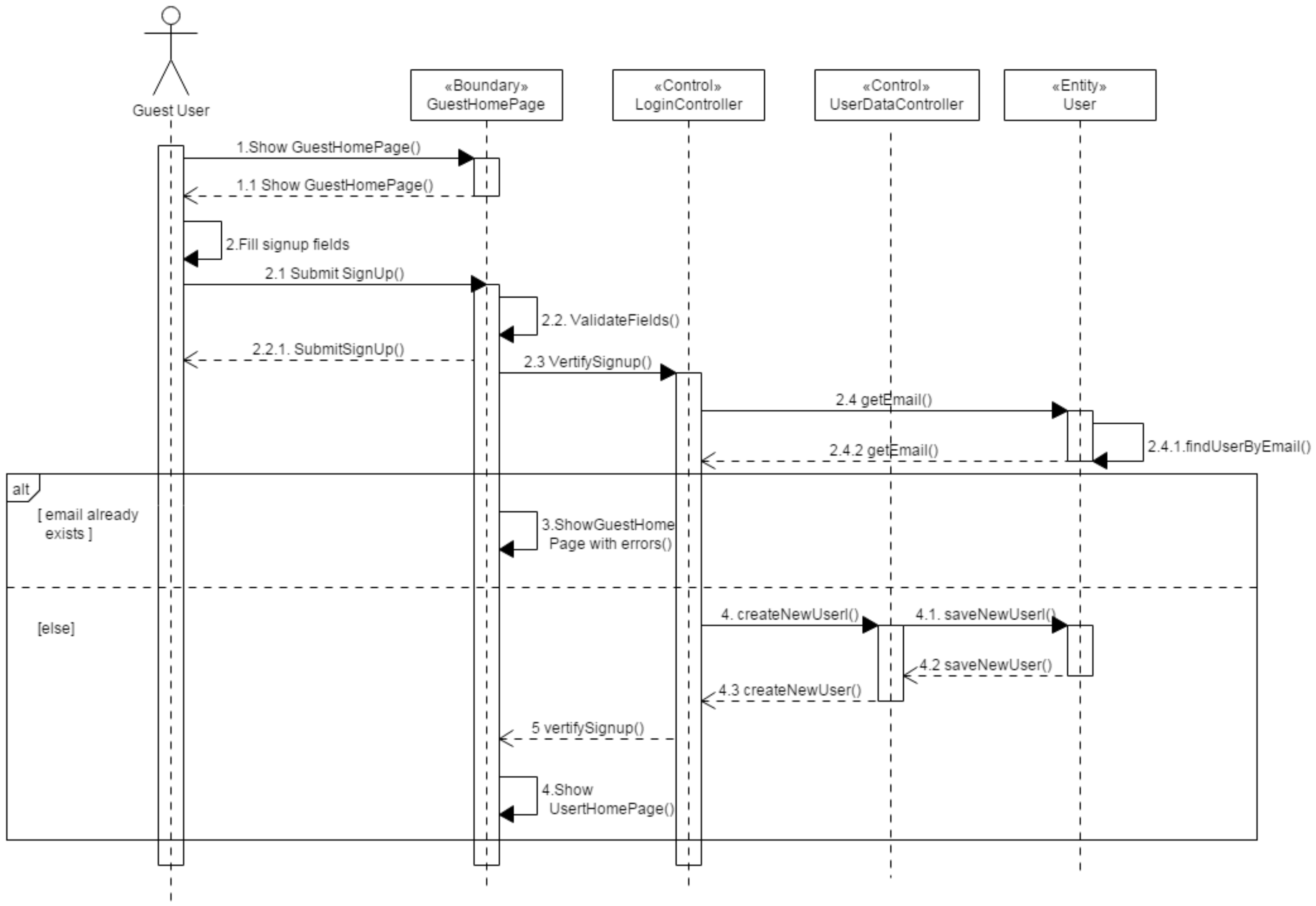

Figure 4.10:

4.3.2 User Log in

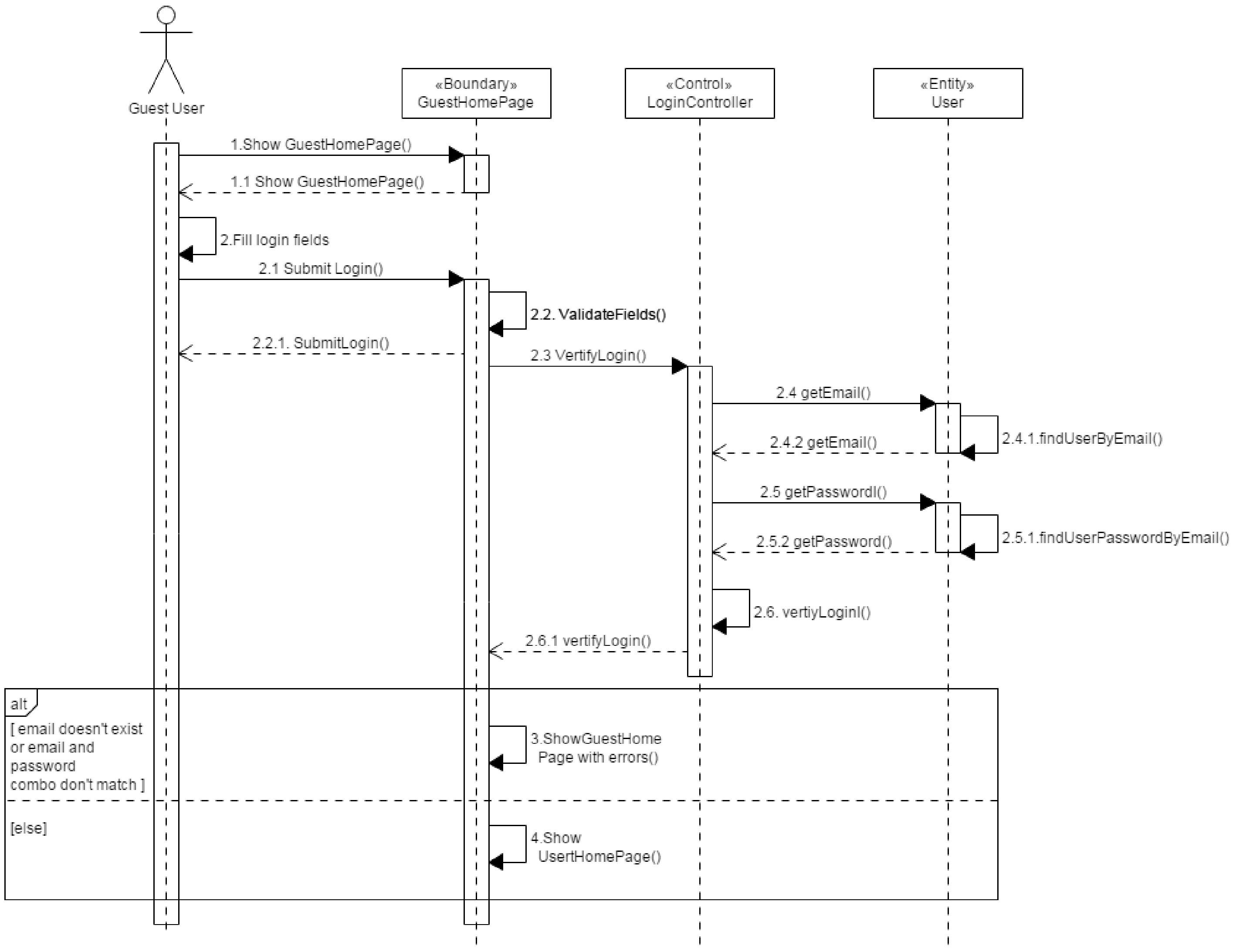

Figure 4.11:

4.3.3 Change Email

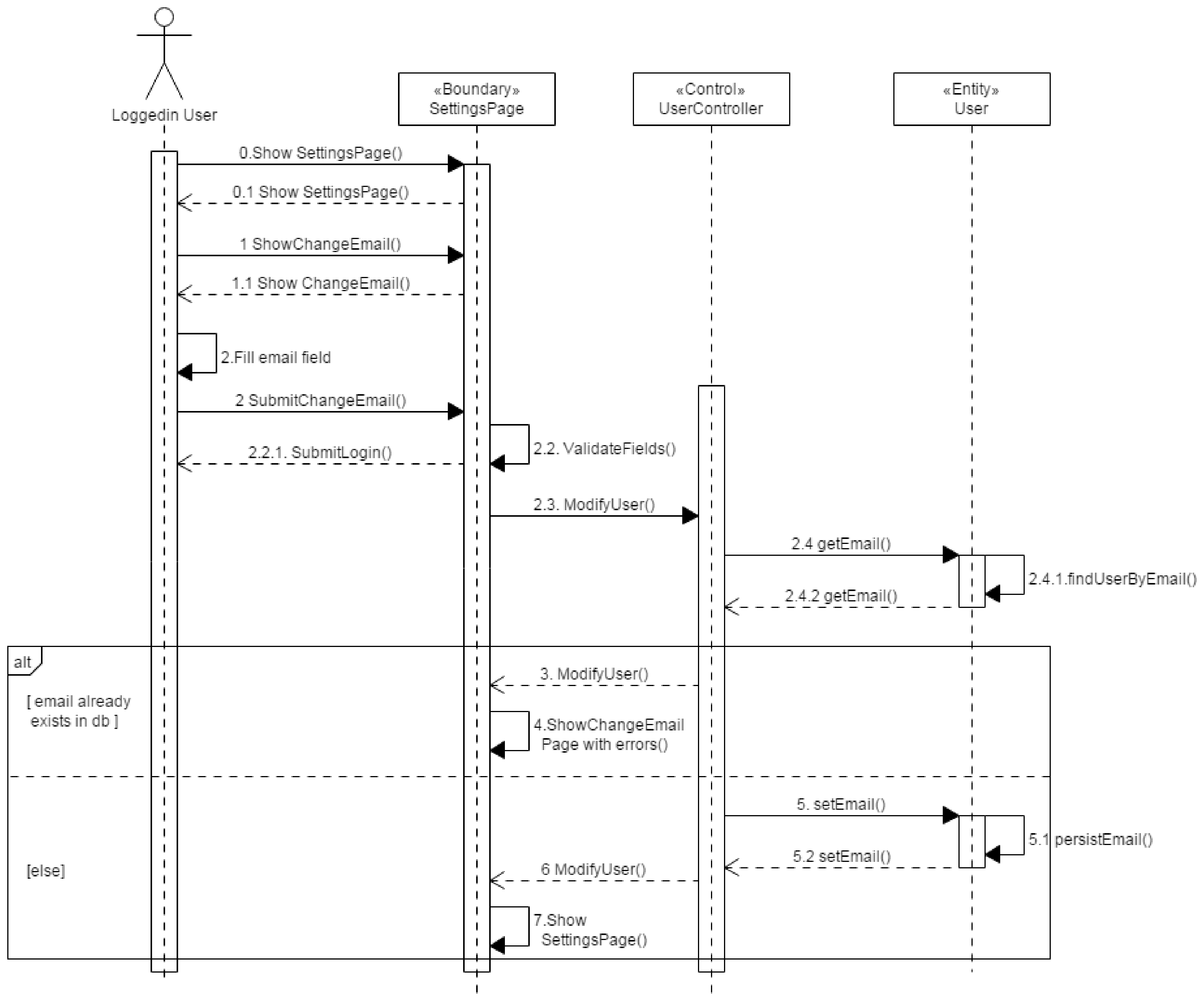

Figure 4.12:

4.3.4 Create Auction

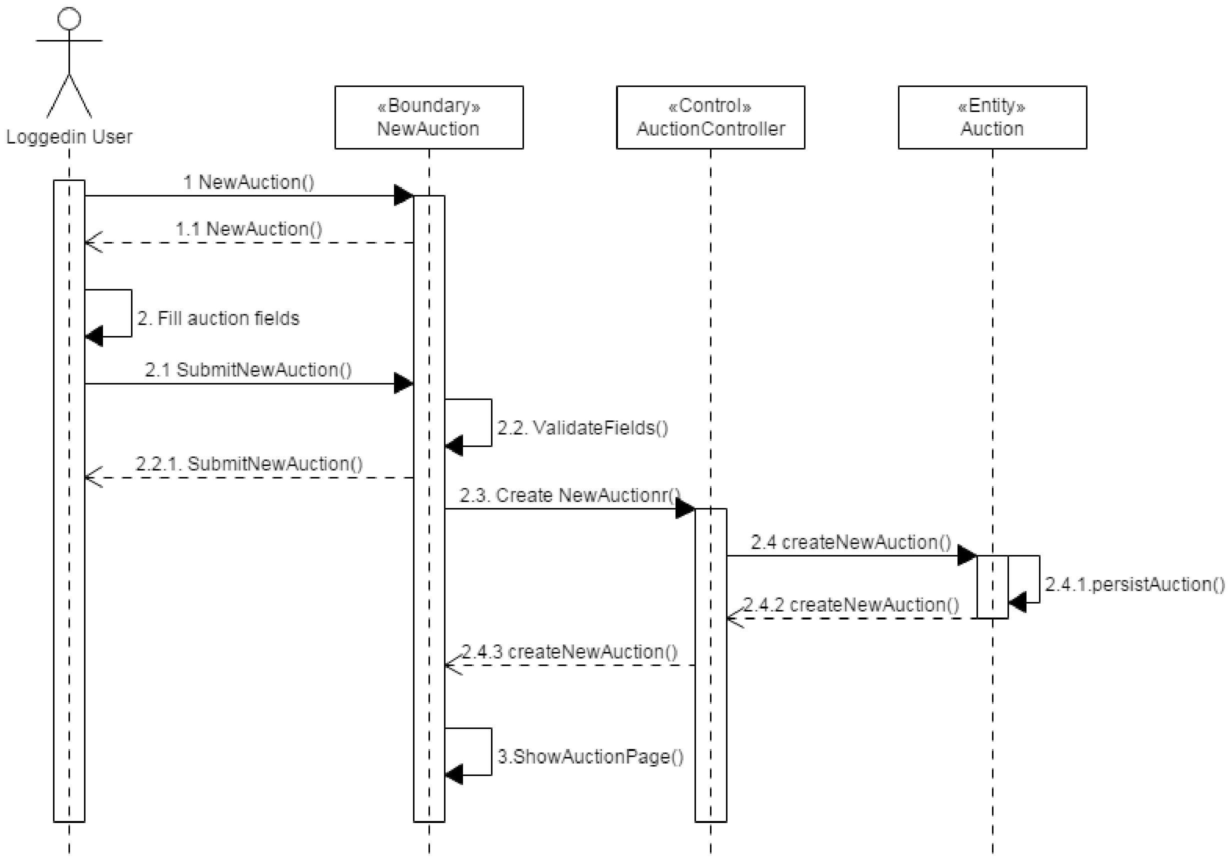

Figure 4.13:

4.3.5 Place Bid

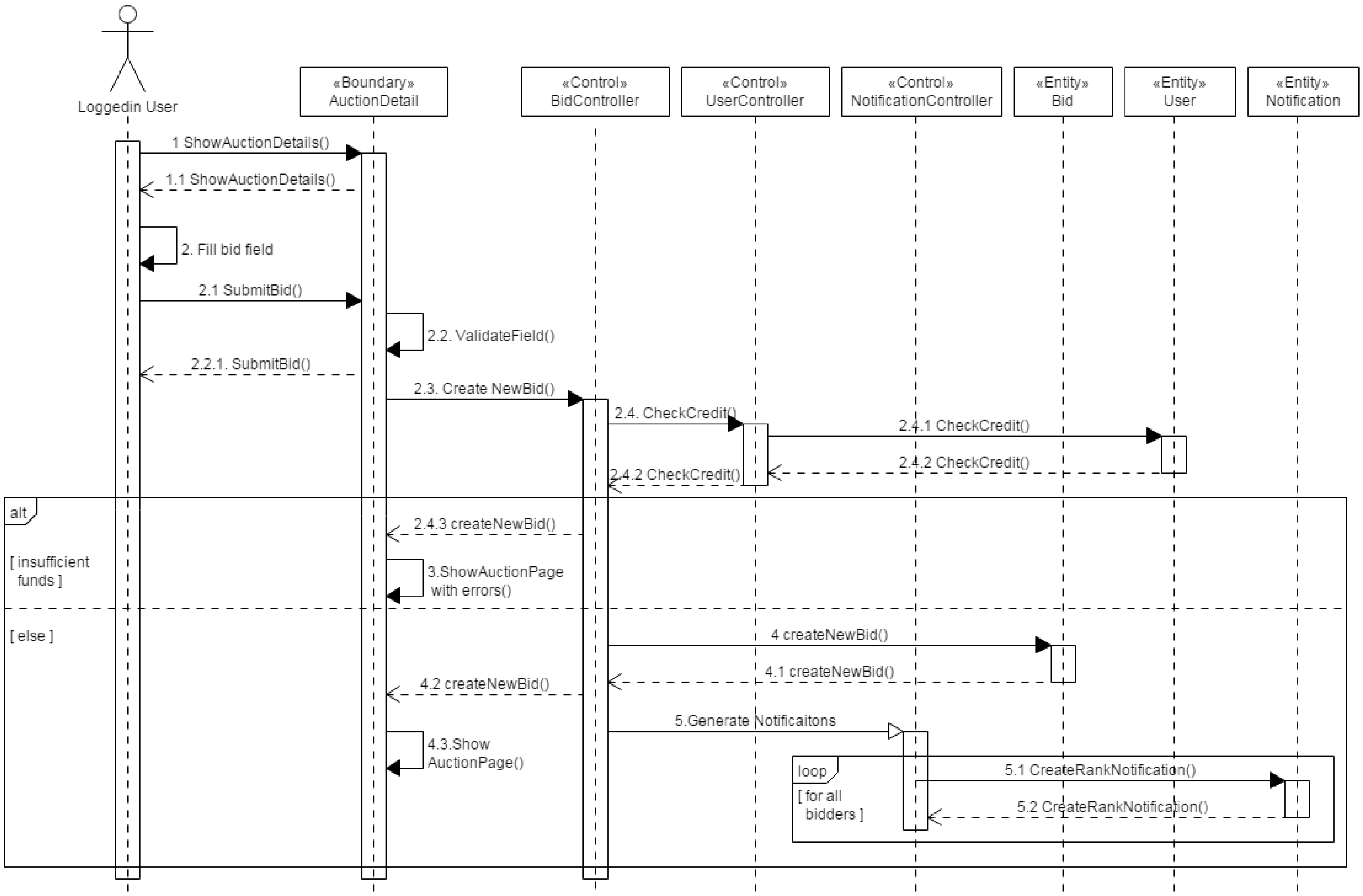

Figure 4.14:

Chapter 5

Tools used:

- Apache OpenOffice Draw 4.1.1 to create Entity-Relationship models
- Sketchboard.io to create User Experience and the BCE diagrams
- MySQL Workbench to create the logical database model
- Adobe Illustrator
- Umlet for Sequence diagrams

Chapter 6

Changes made to RASD:

In the class diagram

- names of *AuctionRankNotification* and *ActionEndNotification*, are changed to *RankNotification* and *OutcomeNotification*, respectively.
- *isRead()* method is removed, while fileds *rank, description,* and *timestamp* are added to *Notification*
- *Category Class* was added
- update an assumption regarding deposit of credit under section 2.7