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.

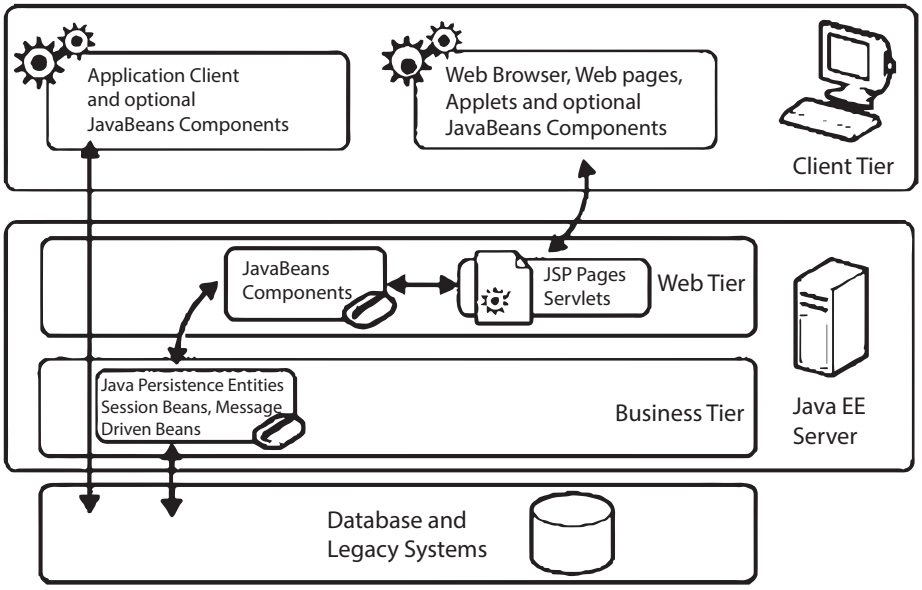

Figure 2.1:

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.

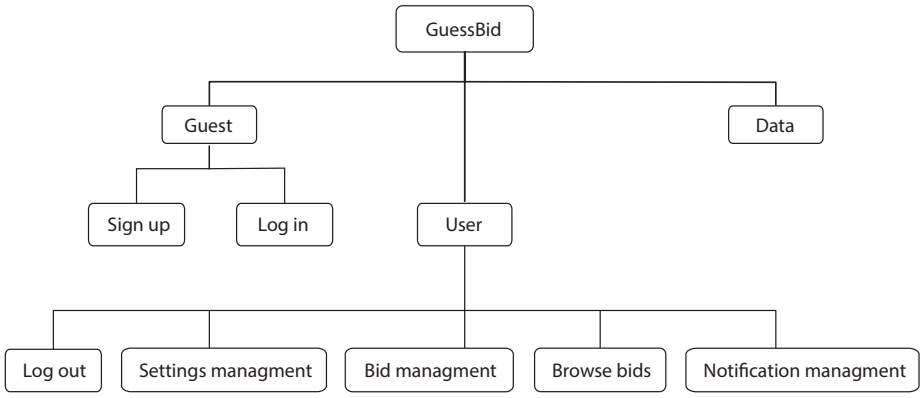

Figure 2.2:

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

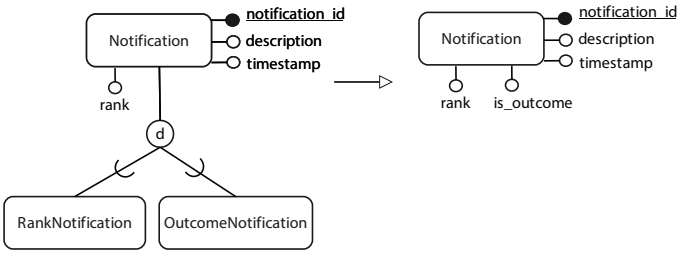

Figure 3.1:

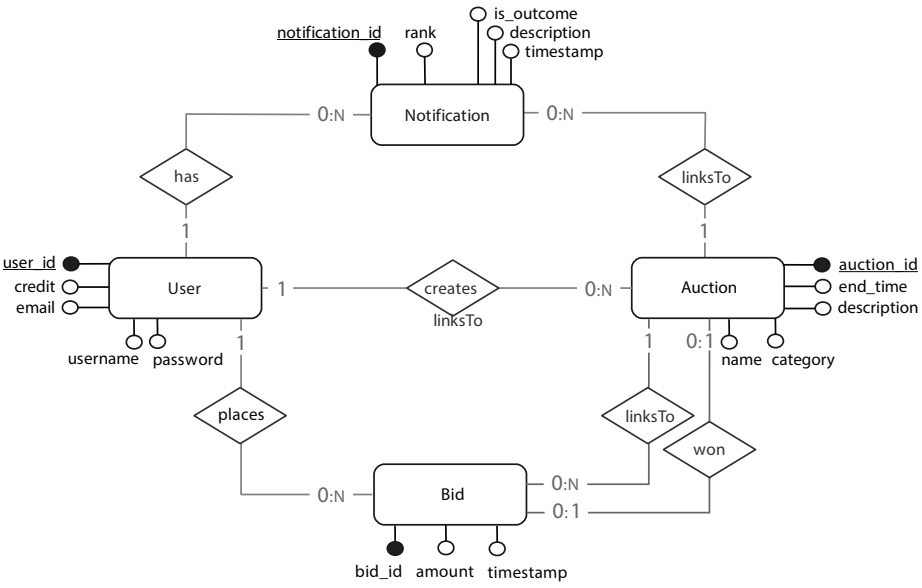

Figure 3.2:

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

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

[1]Reference material: T.Haigh, teaching material Connolly, Begg, Database Systems: A Practical Approach to Design, Implementation and Management

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, 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

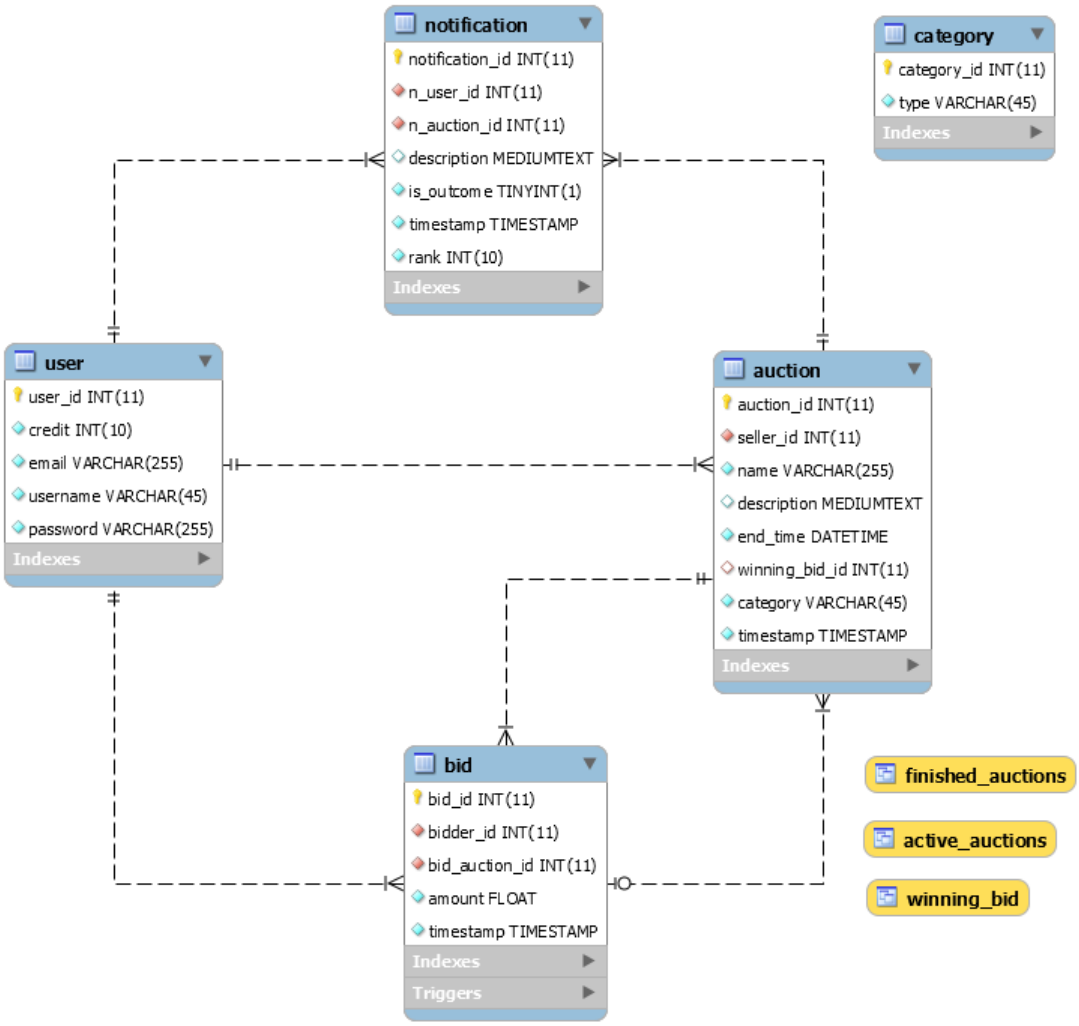
notification
notification_id INT(11)
n_user_id INT(11)
n_auction_id INT(11)
description MEDIUMTEXT
is_outcome TINYINT(1)
timestamp TIMESTAMP
rank INT(10)
Indexes
category
category_id INT(11)
type VARCHAR(45)
Indexes
user
user_id INT(11)
credit INT(10)
email VARCHAR(255)
username VARCHAR(45)
password 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:

dwdwcd p
trigger
bid-before insert - seller cannot bid on own auction

Chapter 4

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