Car Pooling Application

Web Architecures Course Project – January 2013

Ece Mandiracioglu - 154211

1 Introduction

In the last decades the world is facing global warming problem. One of the many reasons is that as technology improves people are used to live in a life with less trouble. To ease our lives we buy cars, we turn on the AC even on a not very hot day, etc.

There are many people moving between the same places for work. They all travel with their own cars or some that don't have one may use public transportation. But imagine hordes of people travelling everyday same roads, creating lots of pollution and of course traffic jam. What if those people could meet up and share their cars?

Or imagine some random people who would like to travel with a low budget and with a good company of people which would make the journey more fun. If a person is supposed to go from A to B by car and if he or she knew there are some other people who would like to do the same way, why not to give them a ride?

So here comes into stage my solution: A carpooling portal. But this portal will be different from the others existing in the market with a slight difference. The passengers-to-be will bid for the rides. So that they could share the gas cost with the driver and still they would travel as cheap as they would like to.

The objective is, as stated in the beginning to create a platform for those who travel the same way everyday to meet and share the costs, thus pollute less the environment, and for those who would just want to travel a specific place for low cost and meet new people.

2 Requirement analysis

Any user can make searches by giving a specific date with a specific departure and arrival point. The system lists all offers on that day. Then the user can visualize the details about each offer. The user must log in in order to bid for an offer. Each offer has a deadline for bidding. After that deadline has passed no one can bid. Based on the number of free seats offered by the driver, that many bidders will be selected and all of them should pay the same price as the top bidder. For example, there are 3 free seats and say 5 people bade. After the deadline expires the top 3 bidders will be eligible to have the ride. However the other two has to pay the same as the highest bid as well.

The users can also see the other's user profiles. Only the registered ones can make a ride offer. This is done from the profile page of the user by clicking the "Add ride offer" button.

2.1 System Users ("actors")

The system actors are two kinds: registered users and unregistered users.

2.2 System Entities ("things")

The system entities are registered users, ride offers, bids and cities. The ER diagram is given under the Section 3.4.

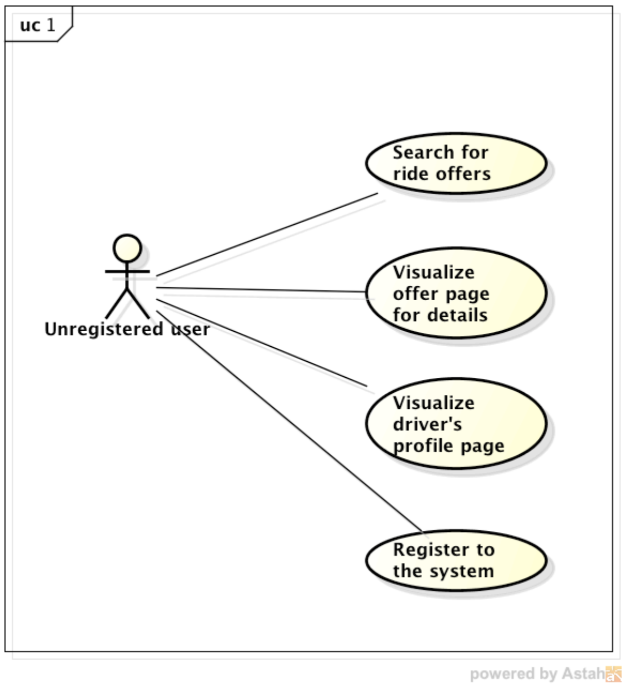
2.3 Possible User Actions

The unregistered users can search for the ride offers, visualize the driver profiles and see the details of a specific offer. However a user should be registered if she wants to bid for an offer or publish an offer.

3 Design

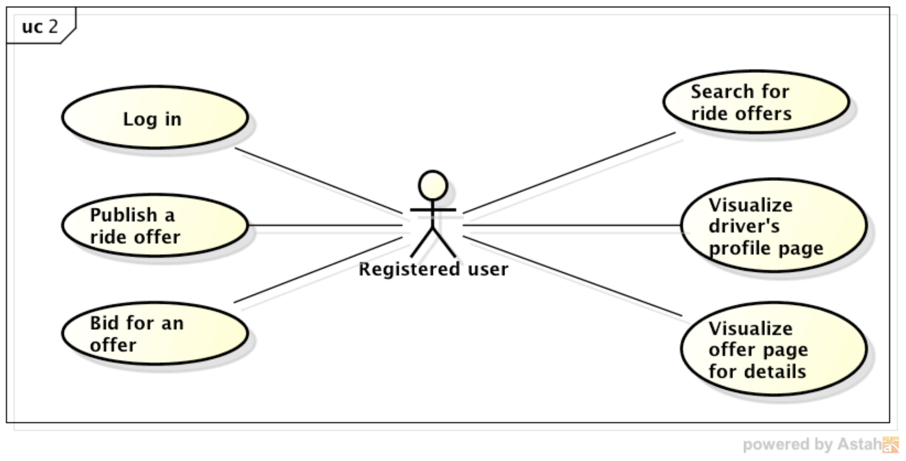
3.1 Use case diagrams

3.1.1 Unregistered User Use Case



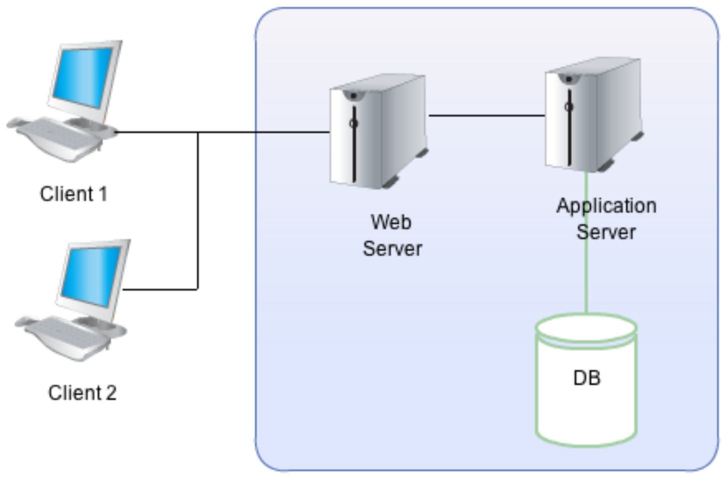
An unregistered user may, as depicted in the figure above, search for ride offers, visualize offer page for details, visualize driver's profile page and register to the system.

3.1.2 Registered User Use Case



A registered user may do whatever an unregistered user can do except from registration, since she is already registered. In addition to this she can publish a ride offer and bid for a specific offer, but for this of course she should first log in to the system.

3.2 Architectural considerations



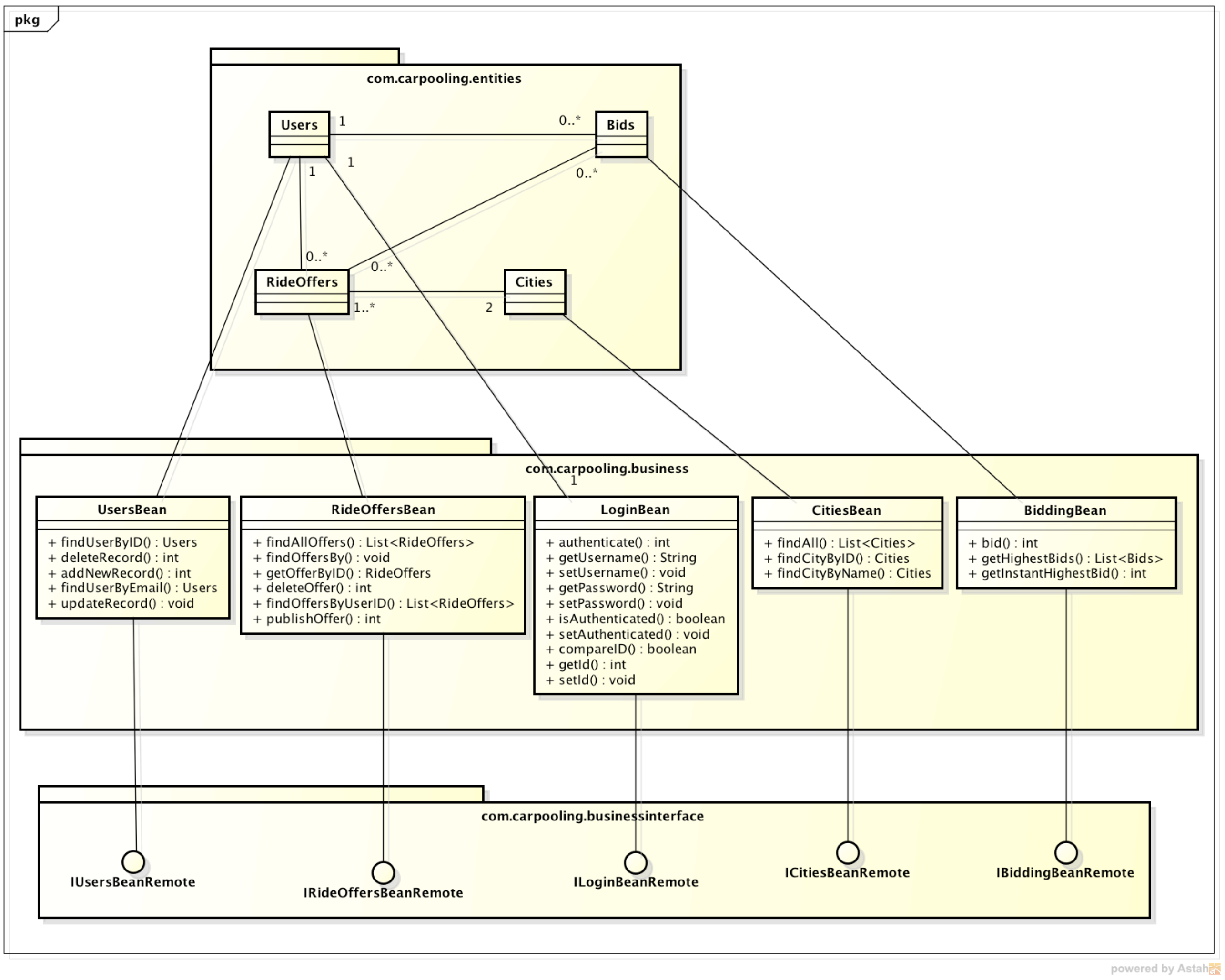
The architecture of the system consists of three parts: DB Server, Application server and Web server. The DB contains the entities of the system. The ER diagram can be found under the Section 3.4. The Application Server contains the Business-Tier of the system. All of the EJBs regarding the business logic are deployed in this server. It also contains the Entity beans, which maps with the tables in the Relational DB. This mapping is done by means of Java Persistence API.

The Web Server contains Java servlets, which has the logic for session management and works as a middle layer between the user interface and the application server. The client can access the system from the web pages, and these are controlled by the servlets.

The design pattern used in the project is Model-View-Controller since the architecture itself requires so. The View part corresponds to the web pages, the Model part refers to the entities described previously. The Controller role is shared between both the servlets and the session beans.

3.3 Object Design (UML Object model)

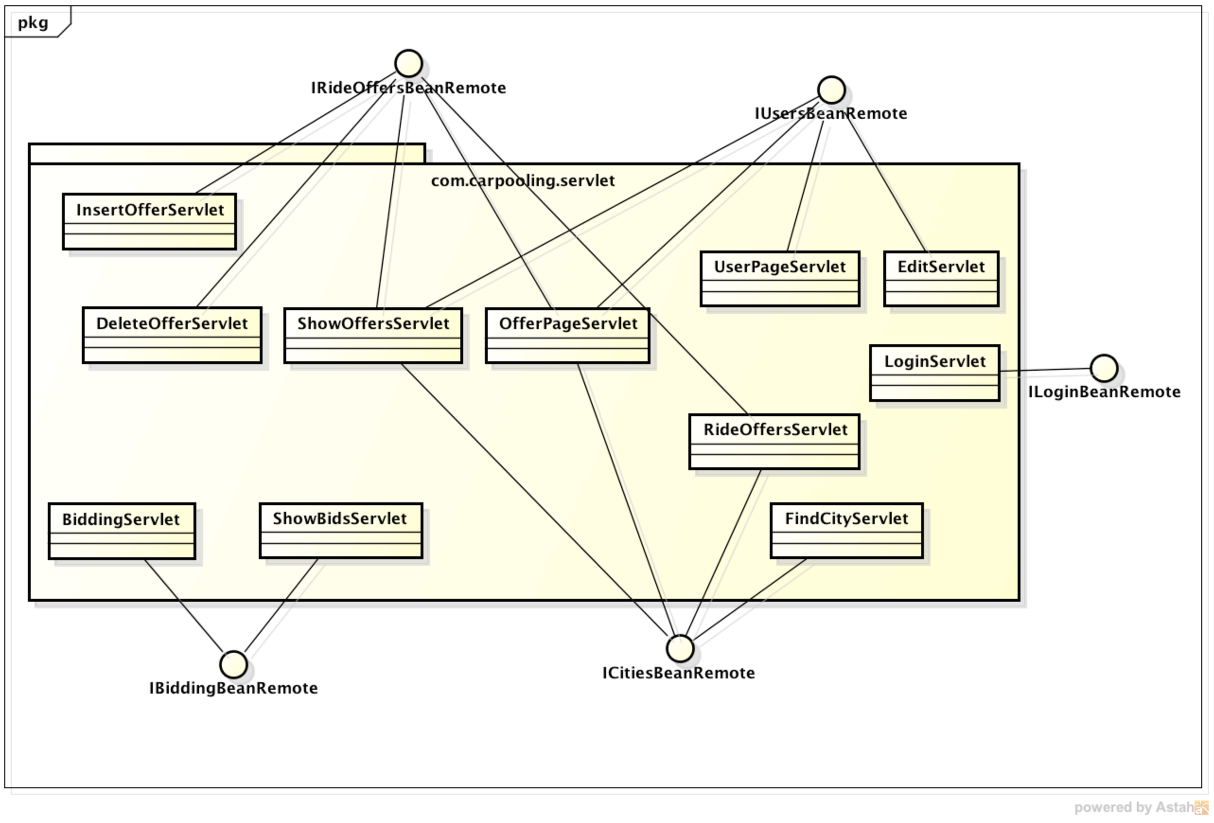
3.3.1 Business-Tier Class Diagram



In the figure above, the class diagram for the business-tier is depicted and the classes are shown with their methods. This consists of Enterprise Java Beans, which are session and entity beans. The beans can be accesses via the remote interfaces using JNDI lookup.

All beans in com.carpooling.business package are stateless session beans except from the LoginBean. This stateful session bean is responsible from authenticating the user.

3.3.2 Web-Tier Class Diagram



In the figure above, the associations between the servlets in the web tier and the remote EJB interfaces are shown. Each servlet, as can be seen, makes call to a different bean interface based on its functionality. The explanation about each servlet is given as below:

BiddingServlet: This servlet inserts new bid for an offer based on the deadline for bidding. If the user is not logged in it shows a message. If the user bids less then the latest bid, then informs the user to bid higher than that amount. On bidding, the driver will be able to see the bid if the page is open without refreshing. The page makes asynchronous auto-load. On the other hand, the bidder sees a success message.

ShowBidsServlet: This servlet is responsible for printing bids of a specific offer to the offer page. It is made an AJAX call from javascript every 1s and on a new bid is inserted it will be seen immediately.

InsertOfferServlet: This servlet is responsible for inserting new offer from a user's profile page.

DeleteOfferServlet: This servlet is responsible for deleting an offer from a user's profile page.

ShowOffersServlet: This servlet is responsible for printing offers of a specific user to the profile page. It is made an AJAX call from javascript every 1s and on every new offer inserted it will be seen immediately. It does not print offers of the user if it is not the profile page of the logged in user. Inserting new offer or editing the user details functionalities are not activated also if the user is not logged in.

OfferPageServlet: This is the servlet, which is responsible in printing details of a specific offer on offer page.

RideOffersServlet: This servlet is called when a ride offers search is made given date, departure and arrival points in the home page.

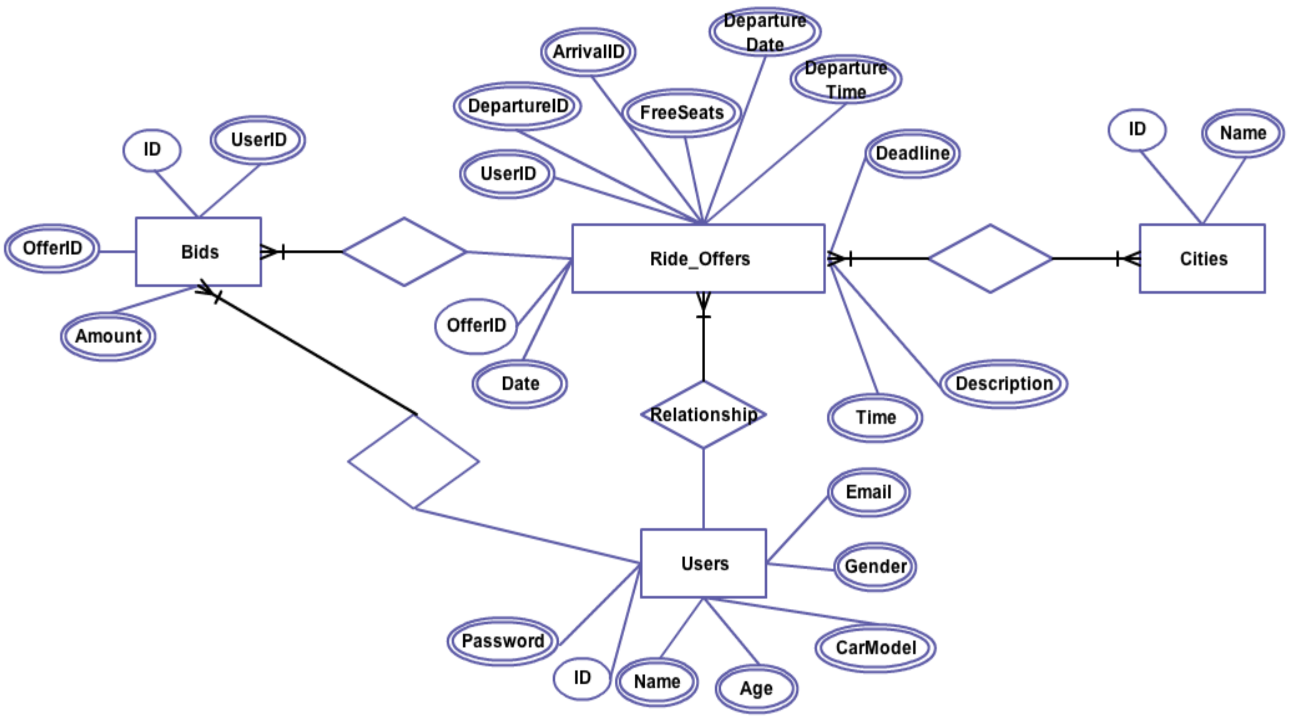
FindCityServlet: This servlet publishes a JSON array of cities based on the user input as she types in letters one by one. This is made an AJAX call within a javascript method.

UserPageServlet: This is the servlet, which is responsible in printing details of a specific user on profile page

EditServlet: Used for editing a user's details.

LoginServlet: Used for authenticating a user based on username and password.

3.4 DB organization



The ER diagram of the system is depicted in the figure above. A ride offer may have 1 or more bids whereas each submitted bid has a unique id and corresponds to only one ride offer. A ride offer can be made only one user and a user may have zero or more ride offers. Each ride offer has a departure city and an arrival city.

Bids table references Ride_Offers table with OfferID foreign key. Ride_Offers table references Users table with UserID foreign key. Bids table references Users table with UserID foreign key.

4 Implementation and deployment

4.1 State management

When a user is logged in to the system, an instance of a stateful session bean is created for that client and username and password is kept in this bean. This bean is called LoginBean. When the user logs out, the bean is destroyed.

4.2 Authentication

Authentication is realized using a servlet filter. This filter manages the login and logout of a client. If an unauthenticated (not registered or logged in) user tries to access a restricted functionality or a webpage (e.g. bidding) then it redirects to the login page. The servlet filter is declared in the web.xml file of the web-tier.

4.3 Transactional behavior

The transactions are started declaratively, i.e. controlled by the beans. Only programmatic transaction is done in the BiddingBean. When a user bids for an offer, bid() method inside the BiddingBean starts a transaction so that bidding the same amount at the same time by many clients would be prevented.

4.4 Technologies used

The project is developed on Eclipse Indigo Java EE IDE. The business logic is deployed on JBoss AS 7 (application server) and the web application logic is deployed on Apache Tomcat v7.0.34. The relational database is created on MySQL Server.

For the business logic Enterprise Java Beans are used.

For the web interface jQuery and jQueryUI API's are used together with custom CSS and javascript methods.