Politecnico di Milano

Master in Computer Science and Engineering

Software Engineering 2 Project:
myTaxiService

Design Document

Document Version 2

Authors:
Filippo Leporati
Danilo Fusi

Professor:
Elisabetta Di Nitto

1 TABLE OF CONTENTS

2 INTRODUCTION

2.1 PURPOSE

This document describes the general and specific architecture of the application myTaxiService. The government of Milan at optimizing its taxi service commissioned this project. In particular, it wants to: i) simplify the access of passengers to the service, and ii) guarantee a fair management of taxi queues.

This document will explain the architectural decisions and trade-offs chosen in the design process and its justifications.

2.2 SCOPE

The architectural descriptions provided concern the functional view, module view, deployment view, data layer, business logic and the user interface of the RASD. Hence, the architecture will consider the following functionalities, which myTaxiService provides:

- **Users**
  myTaxiService will manage registering, logging in/out of users.

- **Profile**
  myTaxiService will manage personal data of the different types of users. Users can be unregistered, registered and administrator.

- **Requests and reservations**
  myTaxiService will manage taxi request and reservations made by the users.

- **Taxi distribution**
  myTaxiService will manage the taxis' distribution and their optimization.

2.3 DEFINITIONS AND ACRONYMS

2.3.1 Definitions

Keyword	Definitions
Request	A passenger makes a request when he wants immediately a taxi.
Reservation	A passenger makes a reservation when he does not want taxi immediately, but he wants to select a different time or day.
Software product	System to be developed
Expert/User	A registered user

2.3.2 Acronyms and abbreviations

Acronym or abbreviation	Definitions
JEE	Java Enterprise Edition
AS	Application Server
EJB	Enterprise Java Beans
FR	Functional Requirement
RASD	Requirements Analysis and Specification Document
RDBMS	Relational Database Management System
NFR	Non-functional Requirement
QoS	Quality of Service
XHTML	Extensible HyperText Markup Language
ER	Entity Relationship
LD	Logical Design

2.4 Reference Documents

1. Analysis document: RASDv2.pdf
2. IEEE Standard for Information Technology-Systems Design- Software Design Descriptions: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5167255

2.5 Document Structure

This document specifies the architecture of myTaxiService spreading from the general into the specific. In addition, it describes the architectural decisions and trade-offs to justifies them. The design was guided by a top-down process approach and the document structure reflects this tactic.

The document is organized as follows:

Section 1, Introduction, provides a resume of the architectural descriptions.

Section 2, Architectural design, provides a general description of myTaxiService including its functionality and matters related to the overall system and its design.

Section 3, Algorithm design, focus on the definition of the most relevant algorithmic part of myTaxiService.

Section 4, User interface design, it provides an overview on how the user interface(s) of the system will look like.

Section 5, Requirements traceability, it explains how the requirements in the RASD document map into the design elements, defined in this Design Document.

Section 6, Appendixes, provides supporting information and additional material.

3 ARCHITECTURAL DESIGN

This section provides a general description of the software system including its functionality and concerns related to the overall system and its design.

3.1 OVERVIEW

3.1.1 System technologies

The myTaxiService will be designed considering the client-server 3-tier architectural style. Each tier requires specific technologies as depicted below:

Web tier

- Dynamic web pages containing XHTML, which are generated by web components.
- Web components developed with Java Server Faces technology, which is a user interface component framework for web applications.

Business Logic tier

- Java Enterprise Edition 7(JEE7) platform supports applications that provide enterprise services in the Java language.
- Enterprise Java Beans (EJB) 3.1, business components that capture the logic that solves or meets the needs of a particular business domain.
- GlassFish AS 7.1, an open-source server that provides services such as security, data services, transaction support, load balancing, management of distributed applications and supports the JEE7 platform.

Persistence tier

- MySQL Server 5.5, a RDBMS.

3.1.2 Design approach

The design approach is based on a client-server 3-tier distributed system, where each tier is described as follows:

1. **Client tier:** This tier is responsible of translating user actions and displays the output of logic operations into something the user can understand.

2. **Business Logic tier:** This tier coordinates the application, processes, operation, logical decisions and it performs calculations. It processes data between the client and persistence tiers.

3. **Persistence tier:** This tier holds the information of the system and it is in charge of storing and retrieving information from a database.

The design process followed a top-down approach, so first the team will identify the tiers and then they split them in components to analyse the inner functionalities. Hence, each component is responsible for certain functionalities and interacts with others.

3.1.3 Overall design

This subsection presents the design model of myTaxiService, specifying the basic relations between packages, use cases and users.

3.1.3.1 General Package Design

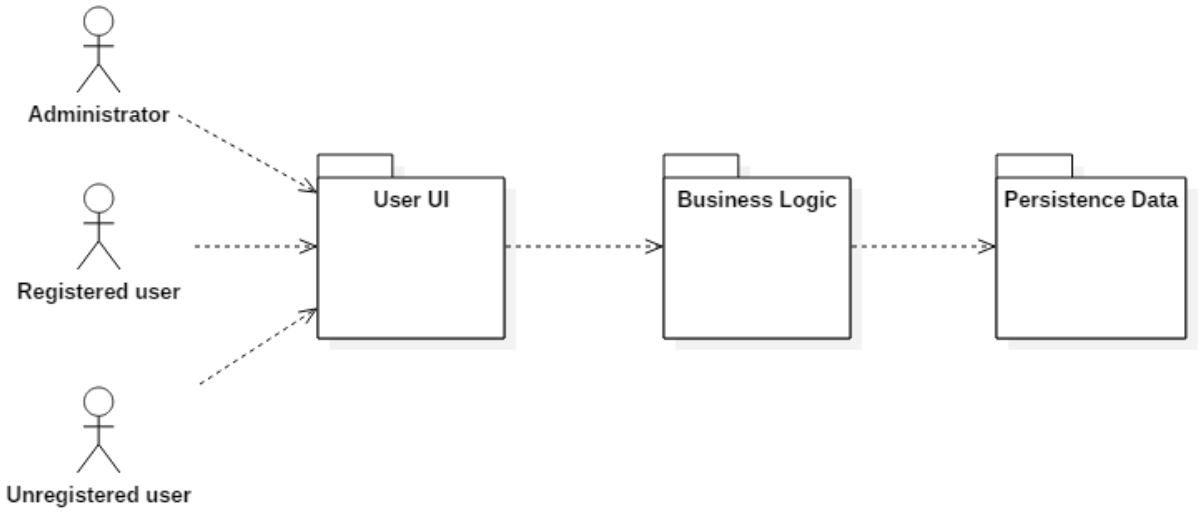
Each tier is split into components and each component must guarantee certain functionalities, which fulfil the requirements. There is a correlation between use cases and package design. In the diagram, we can identify three packages:

- **User UI Package**
  This package contains the user interfaces. It is responsible for the interaction with the user. For instance, it manages the UI request and it refers them to the Business Logic package and retrieving the data back for displaying.

- **Business Logic Package**
  This package contains the business logic components. This package is responsible for handling the User UI package request, processing the information and accessing to Persistence Package to get the data.

- **Persistence data Package**
  This package is responsible for managing the data requests from the Business Logic Package.

  Administrator, registered and unregistered users can access directly to the User UI package and submit request to accomplish their tasks.



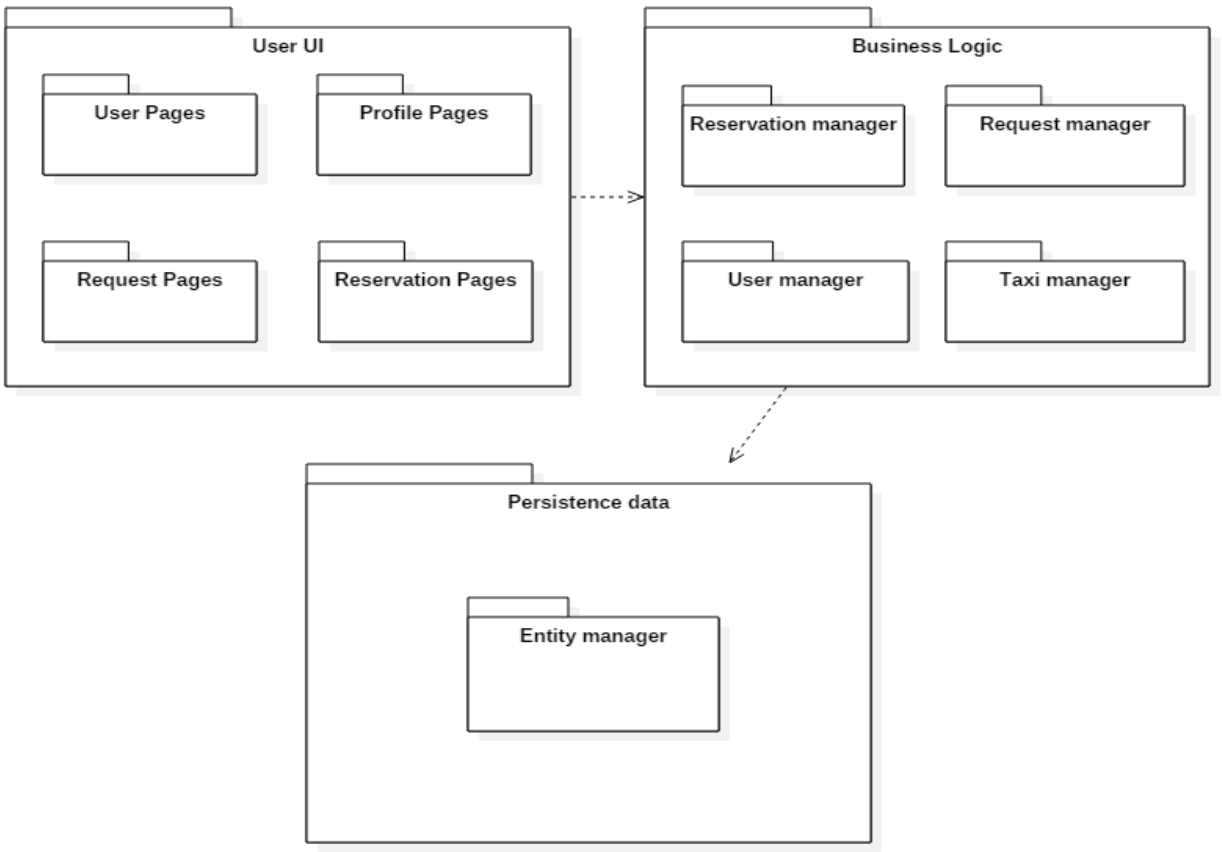
3.1.3.2 Detailed Package Design

This section will point out the package design in a more precise and detailed way:

- **User UI package**
  - User pages. This package contains the general interfaces for user pages.
  - Profile Pages. This package contains the user interfaces for personal profile.
  - Request Pages. This package contains the user interfaces for the request section.
  - Reservation Pages. This package contains the user interfaces for the reservation section.

- **Business Logic package**
  - Request manager. This package contains the logic to manage taxi requests.
  - Reservation manager. This package contains the logic to manage taxi reservations.
  - User manager. This package contains the logic to manage system's users.
  - Taxi distribution manager. This package contains the logic to manage taxi distribution.

- **Persistence data package**
  - Entity Manager. This package is responsible for managing data requests.



3.1.4 Design Considerations

This section highlights the design considerations taken into account in the myTaxiService system design. Assumptions, dependencies, general constraints and performance requirements are clearly stated.

3.1.4.1 Assumptions

Assumption	Action
The software product provides one administration by default.	The administrator credential is provided; therefore, there is no need to set it up manually.
The software product does not support more than one administrator.	Only one person can perform the administrative tasks. Support for multiple administrators is planned for future releases.

3.1.4.2 Dependencies

Dependency	Impact
Mobile device must support iOS, Android or Windows Mobile OS.	myTaxiService cannot operate in systems with different OS.
The supported browsers will be Firefox, Chrome and Internet Explorer.	myTaxiService outputs XHTML code that requires browser which support most of the web standards, otherwise the UI experience will be compromised
A JEE7 AS is required on the server side.	myTaxiService cannot operate if there is no AS that supports the JEE7 standard.

3.1.4.3 General Constraints

This subsection describes the NFRs and the QoS related to the design of the software product.

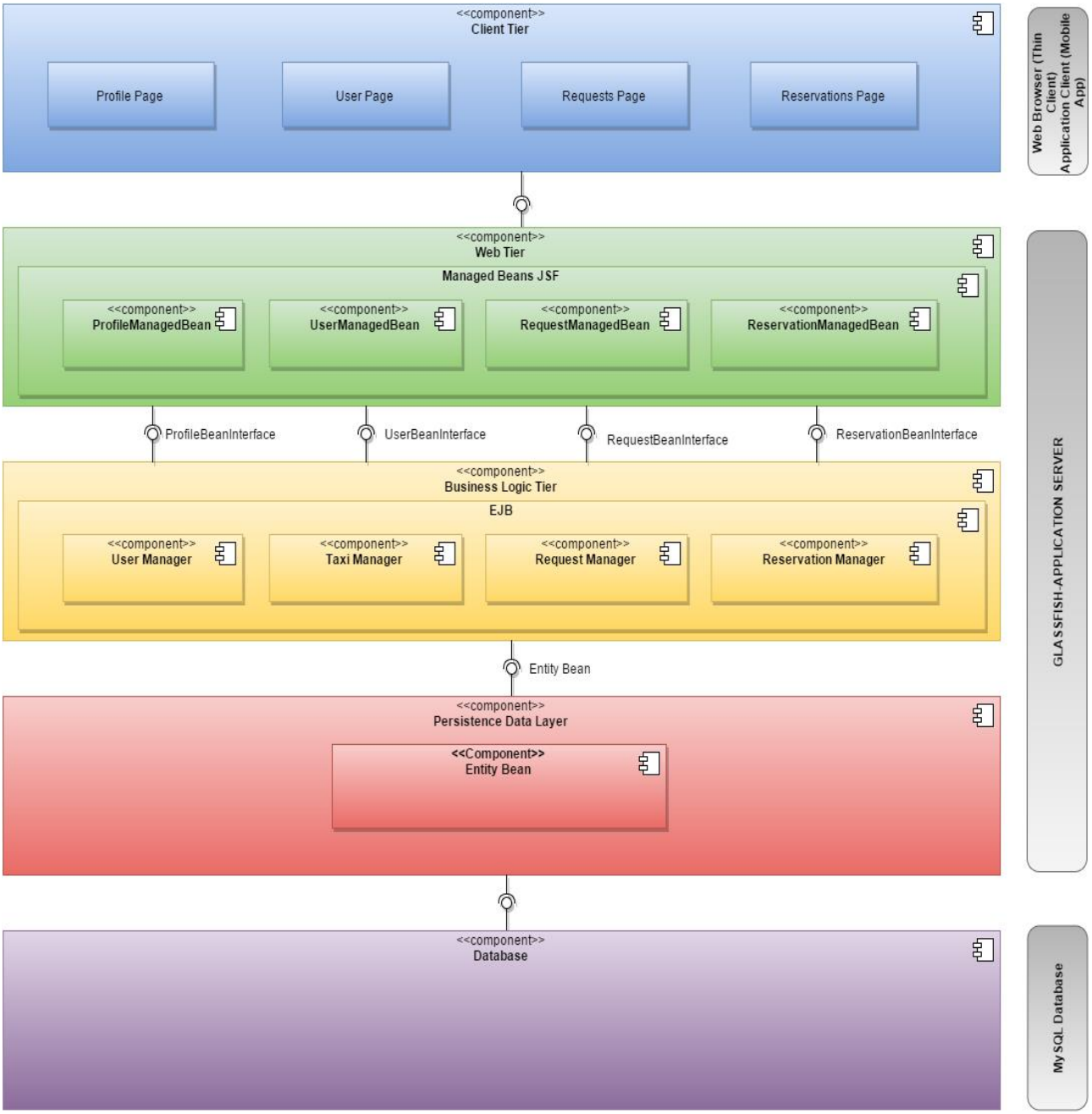
Element	Requirement
Memory	32 Gigabytes or more
Database server	MySQL
Network	Internet Access, HTTP protocol
Security	The software assures a controlled and secure connection during registration and taxi request. SSL and HTTPS are supported. All sensitive data are encrypted.
Hard disk space	1000 Gigabytes or more

3.2 HIGH LEVEL COMPONENTS AND THEIR INTERACTION

myTaxiService shall be developed by using a general 3-tier physical architecture, as mentioned in section above. Different components and subcomponents have been identified and classifies, in order to have a more clear vision of what these components are going to do. The document will also describe the architecture as it be composed by 5 logical tiers. The approach used is a typical TOP-DOWN analysis. This section gives a general description of the architecture we propose for our system and furthermore it provides a small description of each components. Detailed description is in section Detailed software design.
The next paragraph will study in deep these following points:

- Client Tier and Web Tier represent the Web component.
- Business Logic Tier represents the Business Logic component.
- Persistence Tier represents Persistence Component.
- Database represents the Data Model.

3.2.1 Conceptual design



The above diagram represents the conceptual design of myTaxiService. This diagram does not show all the components of the designed software, but it clarifies the logical separation between the tiers. The picture shows four tiers. Therefore, the final architecture is based on n-Tier model. It is clear that the user will interact with the XHTML pages and the mobile application. Web Server implements JSF beans to manage user interaction. This web interaction is then supported by the Business tier, which holds on the information provided by the Persistence layer. The Persistence layer will handle the connection to the database and it will manage all the queries needed from the above layers.

3.2.1.1 Client Tier

The Client Tier is composed by Web Browser (Thin Client) and XHTML pages, but also by Application client (Mobile application). These kind of clients usually interact directly with the business tier, but can also open an HTTP communication channel to interact with the web tier.

3.2.1.2 Web Tier

Web Tier is composed of the web beans, which are part of JavaServer Faces MVC pattern. This tier receives the requests of the user and it displays data regarding the user requests. They interact with the beans in Business Tier, to retrieve the information. JavaServer Faces (JSF/Servlet) will be part of the Web Tier to create dynamic pages using Servlet.

3.2.1.3 Business Logic Tier

The business logic tier is composed of all the application logic; it is responsible of communication with Web Tier and Persistence Tier. Its components are the Enterprise Java Beans (EJB) of GlassFish, named as Managers, just to differentiate from the web beans.

3.2.1.4 Persistence Data Tier

The persistence tier is composed of the entity beans, which represent the entities described in the RASD document. These entities are fundamental as they represent the connection to our database. In order to satisfy the information hiding principle, they should represent a high-level object view of the database application. They will connect this layer to the Database Tier in order to perform insert, update, delete and select operations.

3.2.1.5 Database

The database is composed of the tables generated using assumptions and needs of the project.

3.2.2 System Specification

The following table displays the technologies we will use during implementation. All of the technologies are open source technologies.

Component Name	Technology
Client Tier	Web Browser and XHTML, Application Client (different program languages but JEE7 compatible)
Web Tier	JavaServer Faces/Servlet
Business Tier	JEE7 with AS GLASSFISH 4.1.1
Persistence Tier	MySQL 5.7
Database Tier	MySQL 5.7

3.2.3 Component view

In this section, we provide detailed insight into myTaxiService structure and implementation. In the following sections, we will provide more details about the general subcomponents, which are object of change in the further phase.

Three main components, which are going to be implemented in the future phases, will compose the myTaxiService project.

- Web component
- Business Logic component
- Persistence component

3.2.3.1 Web Component

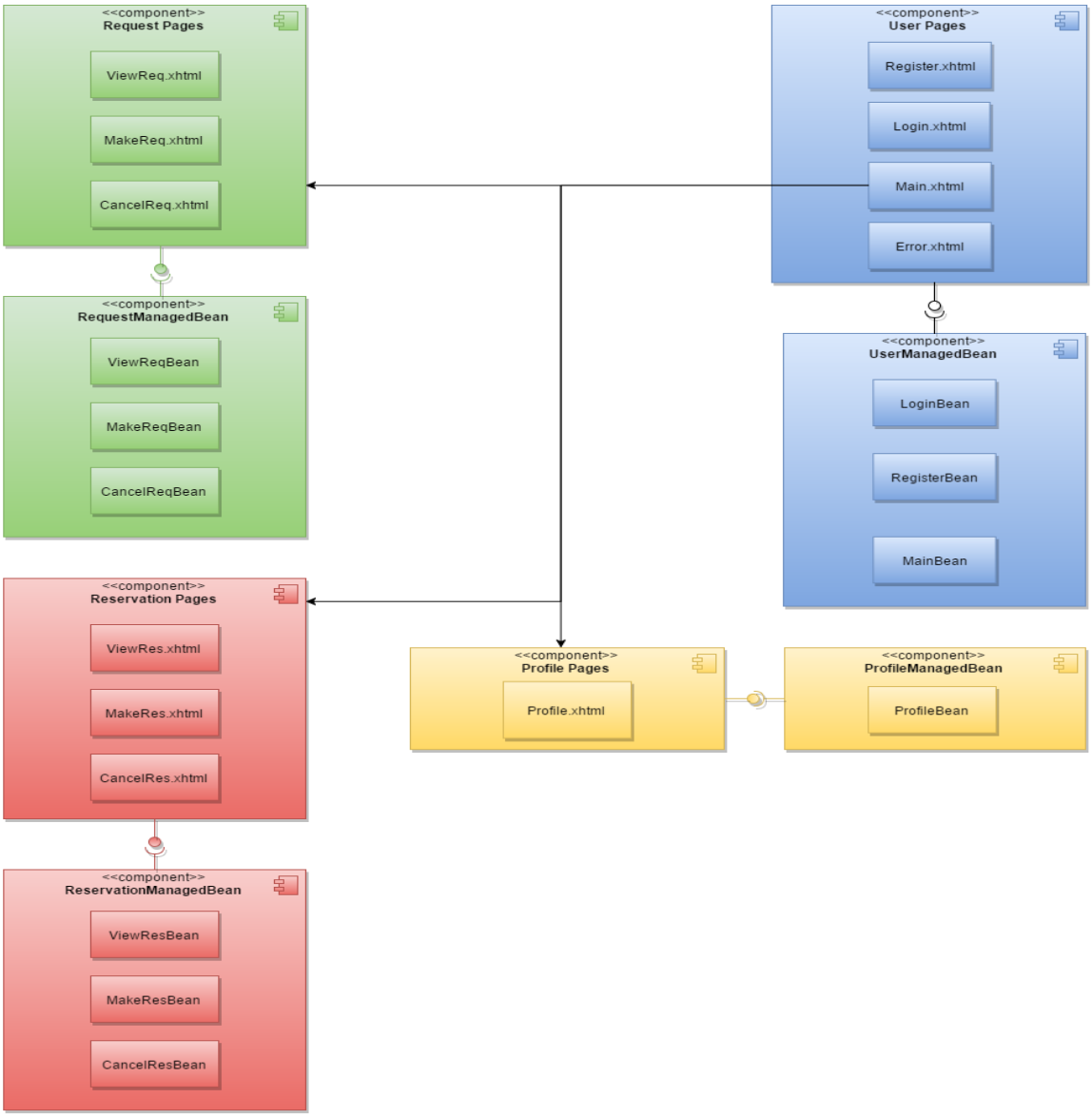
In this section we provide information about web component implementation. The following diagram shows the subcomponents and their communication.
We have identified four main subcomponents and their related Managed Beans.

- User Pages
- Profile Pages
- Request Pages
- Reservation Pages

The Web Tier uses JSF technologies and consequently Managed Beans manage user events in the pages. The related beans Managed Beans are:

- UserManagedBeans
- ProfileManagedBean
- RequestManagedBean
- ReservationManagedBean

The following component diagram shows in a detailed view the Web Tier subcomponents.



3.2.3.1.1 User Pages and managed beans

Pages and beans in this section are responsible for everything related to users.

Component Name	
Name	Login.xhtml
Description	UI for user login
Responsibility	1. Display login form

Component Name	
Name	Register.xhtml
Description	UI for user registration
Responsibility	1. Load registration form

Component Name	
Name	Main.xhtml
Description	UI for displaying main menu for a registered user

Responsibility	1. Display user main homepage 2. Display connection to Profiles, Requests, and Reservations. 3. Display admin home page 4. Display taxi driver main page
Note	Main.xhtml will detect which is the user role (admin, customer, taxi driver) and it will display the appropriate homepage.

Component Name	
Name	LoginBean
Description	Managed Bean for user login
Responsibility	1. Load login form 2. Check login parameters 3. Redirect to main.xhtml

Component Name	
Name	RegisterBean
Description	Managed Bean for user registration
Responsibility	1. Load login form 2. Check login parameters 3. Redirect to main.xhtml

Component Name	
Name	MainBean
Description	Managed Bean for displaying main menu for a registered user.
Responsibility	1. Load user main homepage 2. Load connections to Profile, Requests, and Reservations pages. 3. Load custom homepage depending on user role.

3.2.3.1.2 Profile pages and managed beans

Pages and beans in this section are responsible for everything related to profile.

Component Name	
Name	Profile.xhtml
Description	UI interface for displaying use profile information.
Responsibility	1. Display profile information.

Component Name	
Name	ProfileBean
Description	Managed Bean for displaying user profile information.
Responsibility	1. Load profile information based on user role.

3.2.3.1.3 Request pages and managed beans

Pages and bean in this section are responsible for everything related to request.

Component Name	
Name	ViewReq.xhtml
Description	UI interface for displaying user requests.
Responsibility	1. Display requests information.

Component Name	
Name	MakeReq.xhtml
Description	UI interface for making a taxi request
Responsibility	1. Display information to make a request.

Component Name	
Name	CancelReq.xhtml
Description	UI interface for cancelling a taxi request.
Responsibility	1. Display information to cancel a request.

Component Name	
Name	ViewReqBean
Description	Managed Bean for list of requests.
Responsibility	1. Load requests. 2. Load functionalities for the requests.

Component Name	
Name	MakeReqBean
Description	Manages Bean for make a request.
Responsibility	1. Load request form 2. Check request parameters 3. Redirect to main.xhtml

Component Name	
Name	CancelReqBean
Description	Manage Bean for cancel a request
Responsibility	1. Load cancellation page 2. Check cancellation requirements 3. Redirect to main.xhtml

3.2.3.1.4 Reservation pages and managed beans

Pages and beans in this section are responsible for everything related to skills.

Component Name	
Name	ViewRes.xhtml
Description	UI interface for displaying user reservations.
Responsibility	1. Display Reservations information.

Component Name	
Name	MakeRes.xhtml
Description	UI interface for making a taxi reservation
Responsibility	1. Display information to make a reservation.

Component Name	
Name	CancelRes.xhtml
Description	UI interface for cancelling a taxi reservation.
Responsibility	1. Display information to cancel a reservation.

Component Name	
Name	ViewResBean
Description	Managed Bean for list of reservations.
Responsibility	1. Load reservations. 2. Load functionalities for the reservations.

Component Name	
Name	MakeResBean
Description	Manages Bean for make a reservation.
Responsibility	1. Load reservation form 2. Check reservation parameters 3. Redirect to main.xhtml

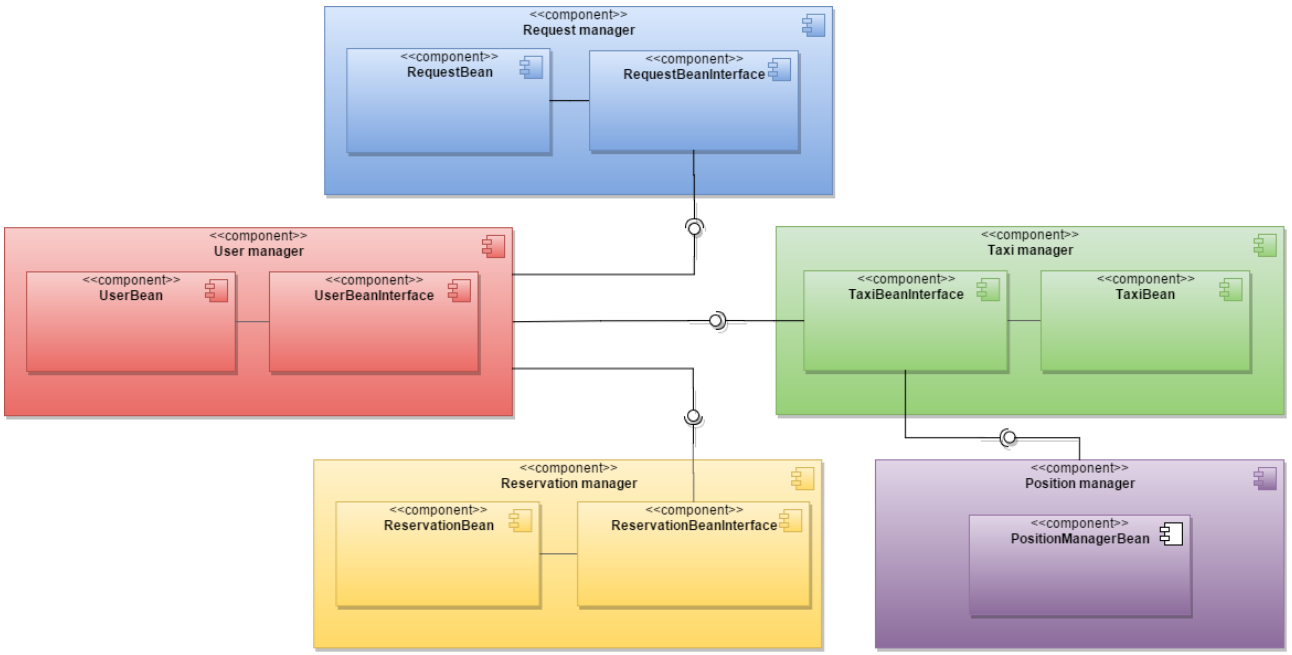
Component Name	
Name	CancelResBean
Description	Manage Bean for cancel a reservation
Responsibility	1. Load cancellation page 2. Check cancellation requirements 3. Redirect to main.xhtml

3.2.3.2 Business Logic component

In this section, we provide information for what needs to implement Business Logic components. The following general diagram shows the subcomponents and their communication. There are four main subcomponents.

- User Manager
- Taxi Manager
- Request Manager
- Reservation Manager

Each of these beans will be defined as an EJB Bean.



3.2.3.2.1 User manager

Component	
Name	UserBeanInterface
Description	Interface instantiated every time communication between beans is needed.
Responsibility	1. Communicate with ReservationBeanInterface, RequestBeanInterface, TaxiBeanInterface

Component	
Name	UserBean
Description	Bean in charge for all functionalities related to users.
Responsibility	1. Login user/administrator 2. Logout user/administrator 3. Register a new user 4. Load profile information for user/administrator 5. Search user

3.2.3.2.2 Reservation manager

Component	
Name	RequestBeanInterface
Description	Interface instantiated every time communication between beans is needed
Responsibility	1. Communicate with UserBeanInterface

Component	
Name	Request Bean
Description	Bean in charge for all functionalities related to taxi request.
Responsibility	1. Load list of requests 2. Add request 3. Remove request 4. Manage request 5. Search request

3.2.3.2.3 Reservation manager

Component	
Name	ReservationBeanInterface
Description	Interface instantiated every time communication between beans is needed
Responsibility	1. Communicate with UserBeanInterface

Component	
Name	Reservation Bean
Description	Bean in charge for all functionalities related to taxi reservations.
Responsibility	1. Load list of reservations 2. Add reservation 3. Remove reservation 4. Manage reservation 5. Search reservation

3.2.3.2.4 Taxi manager

Component	
Name	TaxiBeanInterface
Description	Interface instantiated every time communication between beans is needed.
Responsibility	1. Communicate with UserBeanInterface

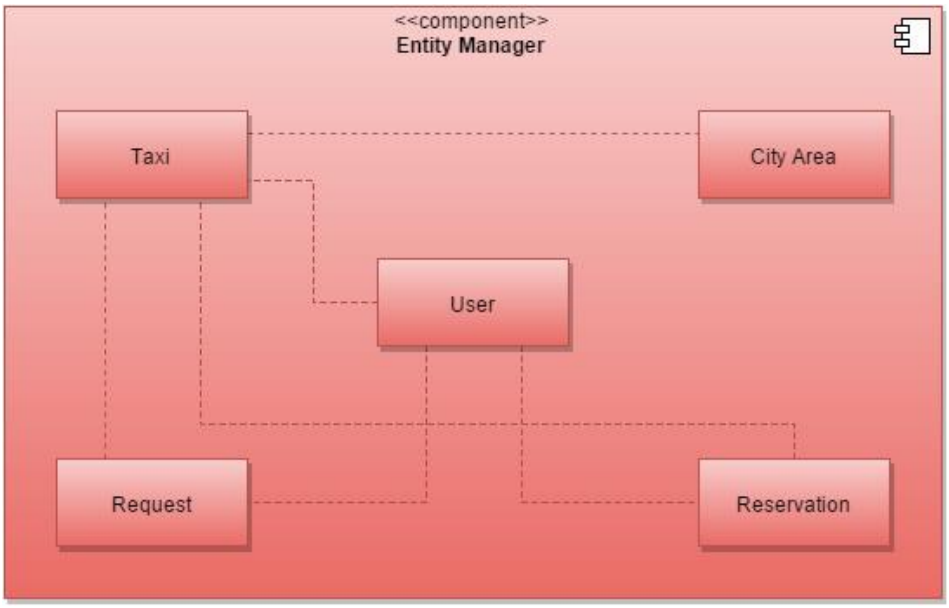
Component	
Name	Taxi Bean
Description	Bean in charge for all functionalities related to taxi management and distribution.

Responsibility	1. Add taxi 2. Remove taxi 3. Locate taxi 4. Redistribute taxi 5. Manage taxi in the city

3.2.3.2.5 Position manager

Component	
Name	Position manager bean
Description	Bean in charge for all functionalities related to manage user and taxi positions.
Responsibility	1. Update position 2. Find position 3. Manage positions 4. Receive auto-update taxi position

3.2.3.3 Persistence component



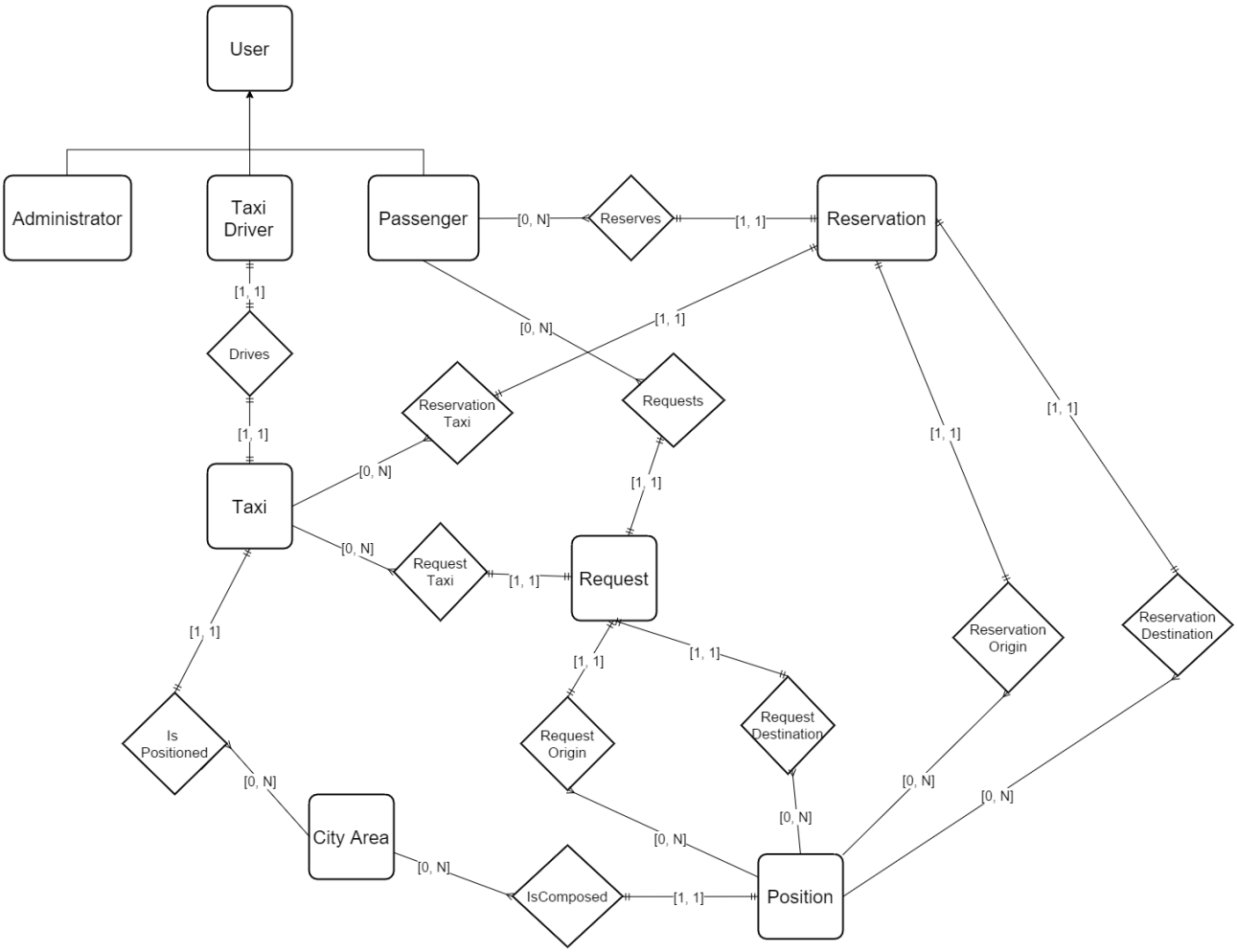
The Persistence component diagram above shows all the entities in the application. All of these entities represent a high-level object of the tables in the database, which will be used in our application. Therefore, all of the entities have inside all the attributes, which are in the associated table in the database.
In the following table, there is a short description of the entities:

Entity	**Description**
User	Represent the users in the system
Taxi	Represent the taxis in the system
Request	Represent the request in the system
Reservation	Represent the reservation in the system

3.2.4 Database Model Structure

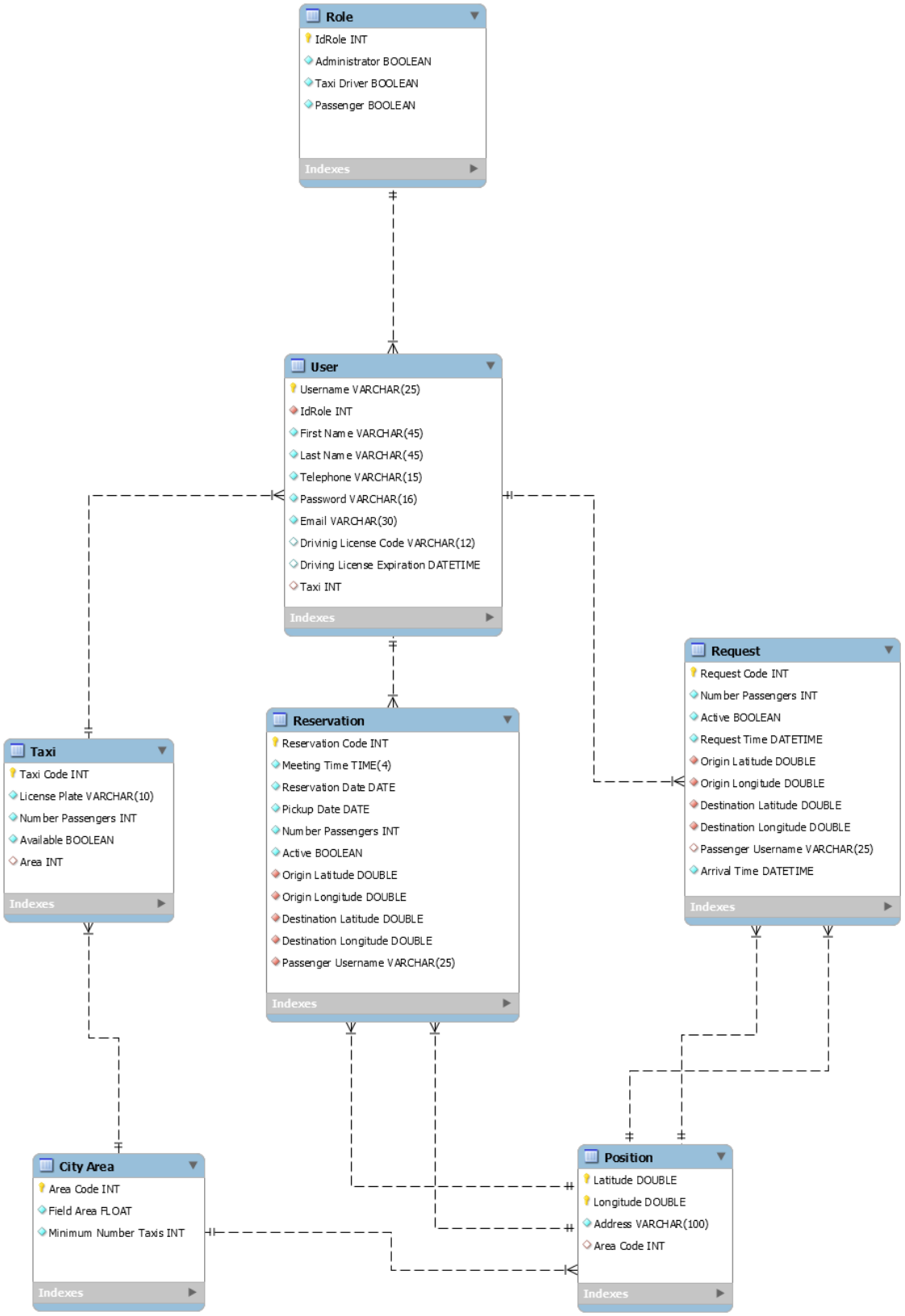
In the following section, we will provide a detail description of the database by providing its both views: the conceptual and logical design diagrams. The Entity Relationship Diagram (ER) will represent the conceptual design of the database.

3.2.4.1 Entity Relationship diagram



3.2.4.2 Conceptual and logic design

The logical design (LD) diagram represents the final implementation of the database. It is based on the previous ER diagram. The LD diagram below is generated by using the MySQL workbench tool.

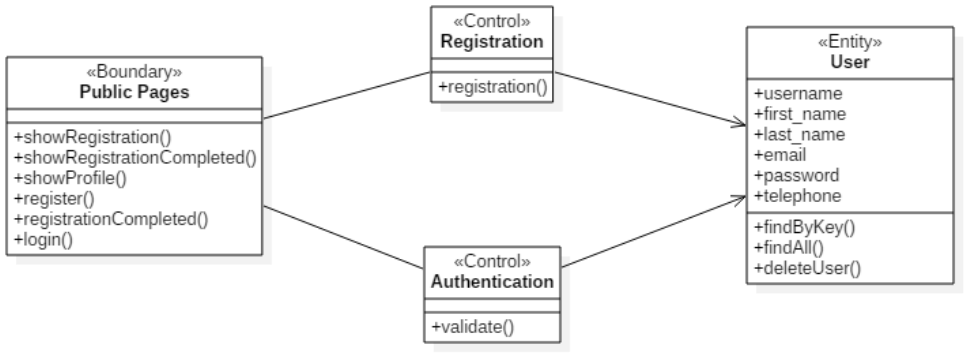


3.2.5 BCE (Boundary-Control-Entity) Model

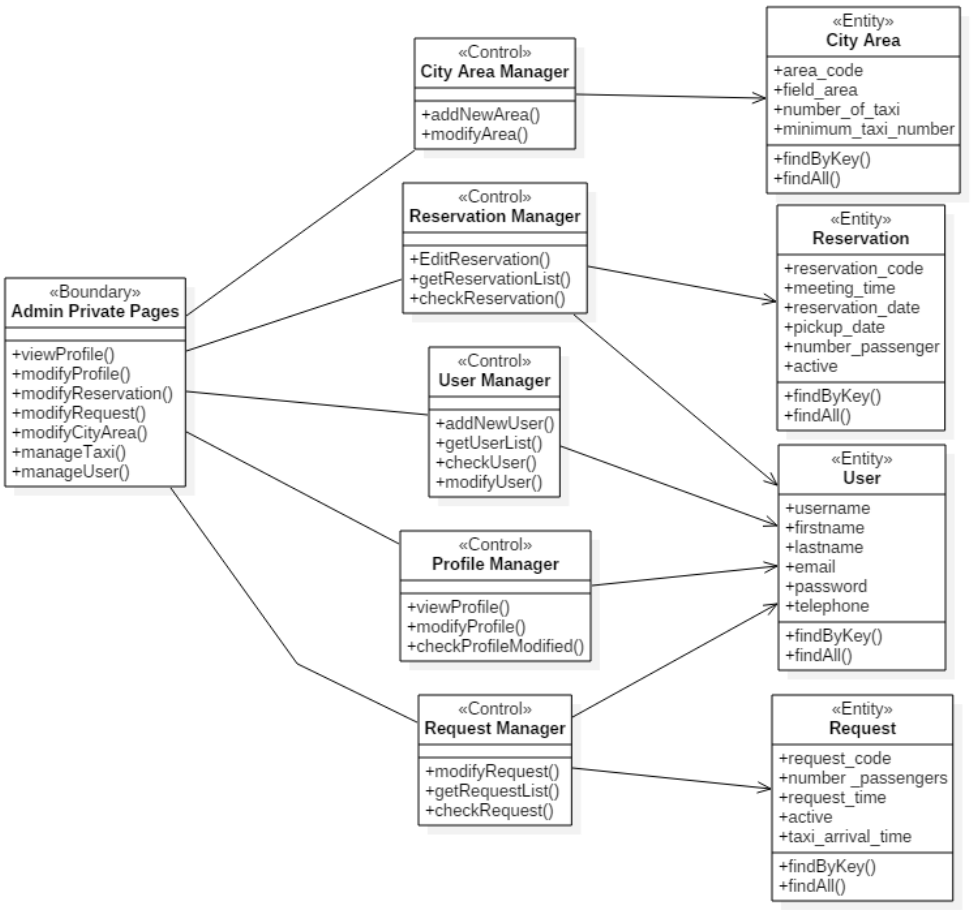
This section provide BCE model diagrams. In particular it focuses on Public Area of the unregistered user and the Private Area of registered user (Admin, taxi driver and passenger).

Classes identified with the stereotype << Boundary >> represent the presentation logic. Classes identified with the stereotype << Control >> represent the business logic. Classes identified with the stereotype <<Entity >> represent the information in the data system.

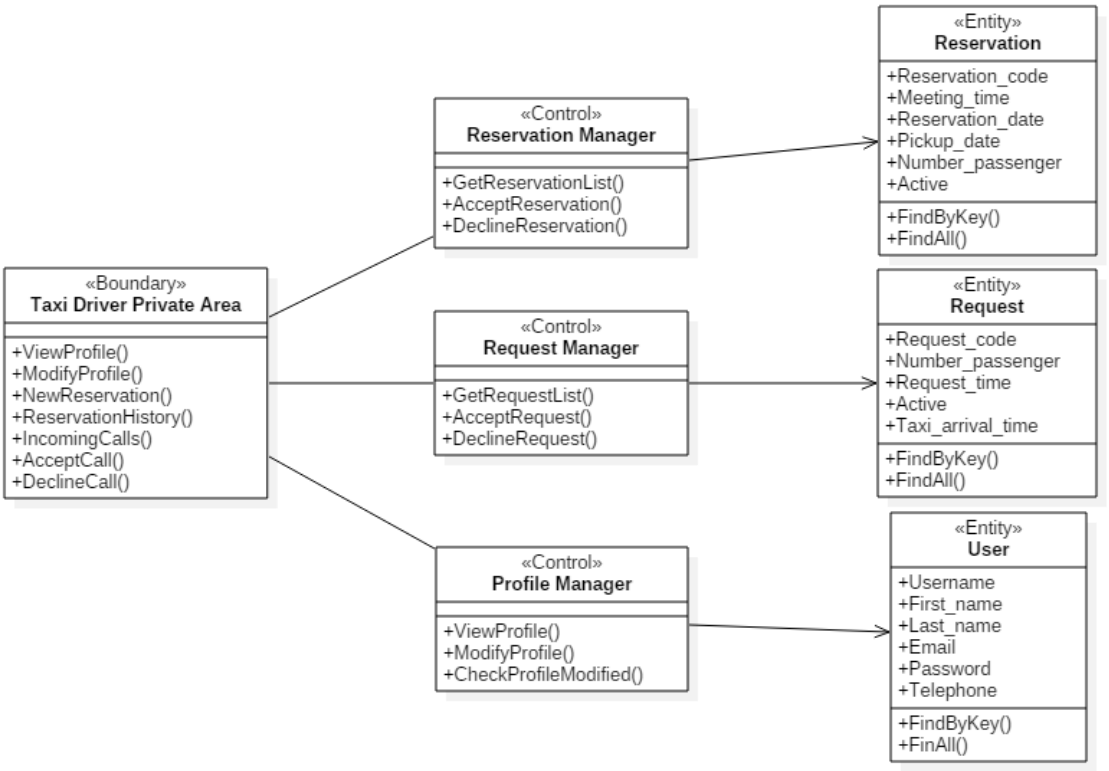
3.2.5.1 Public Area BCE



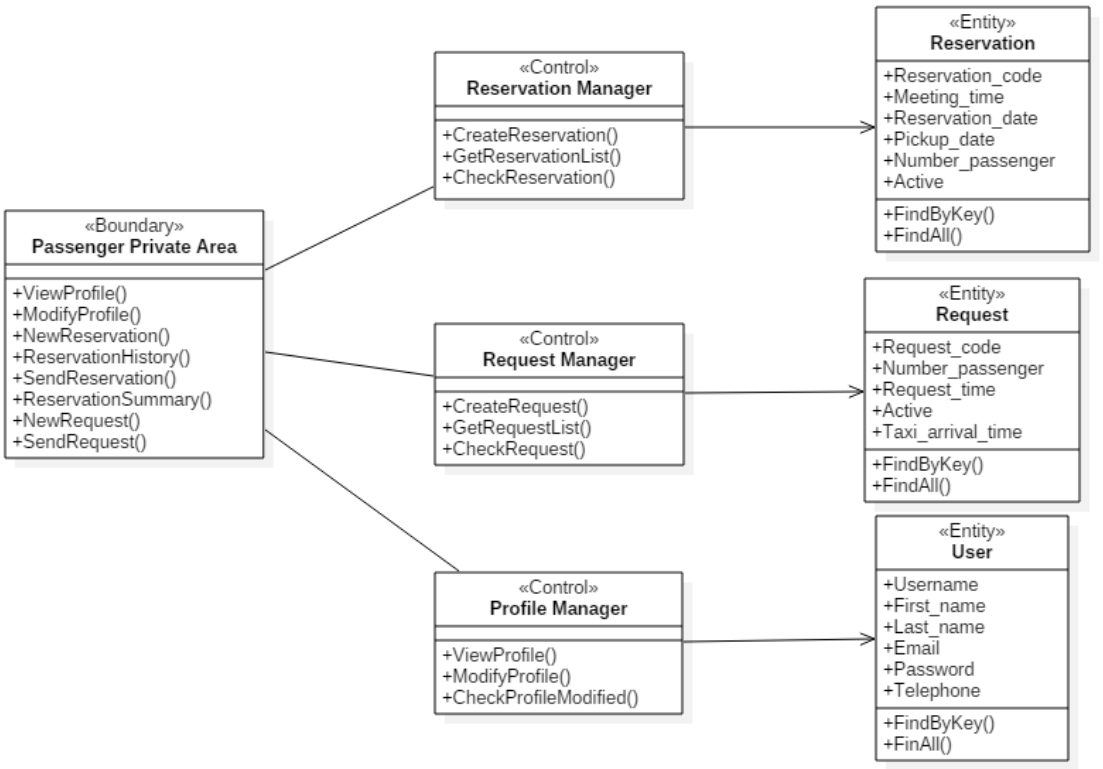
3.2.5.2 Administrator Private Area BCE



3.2.5.3 Taxi Driver Private Area BCE

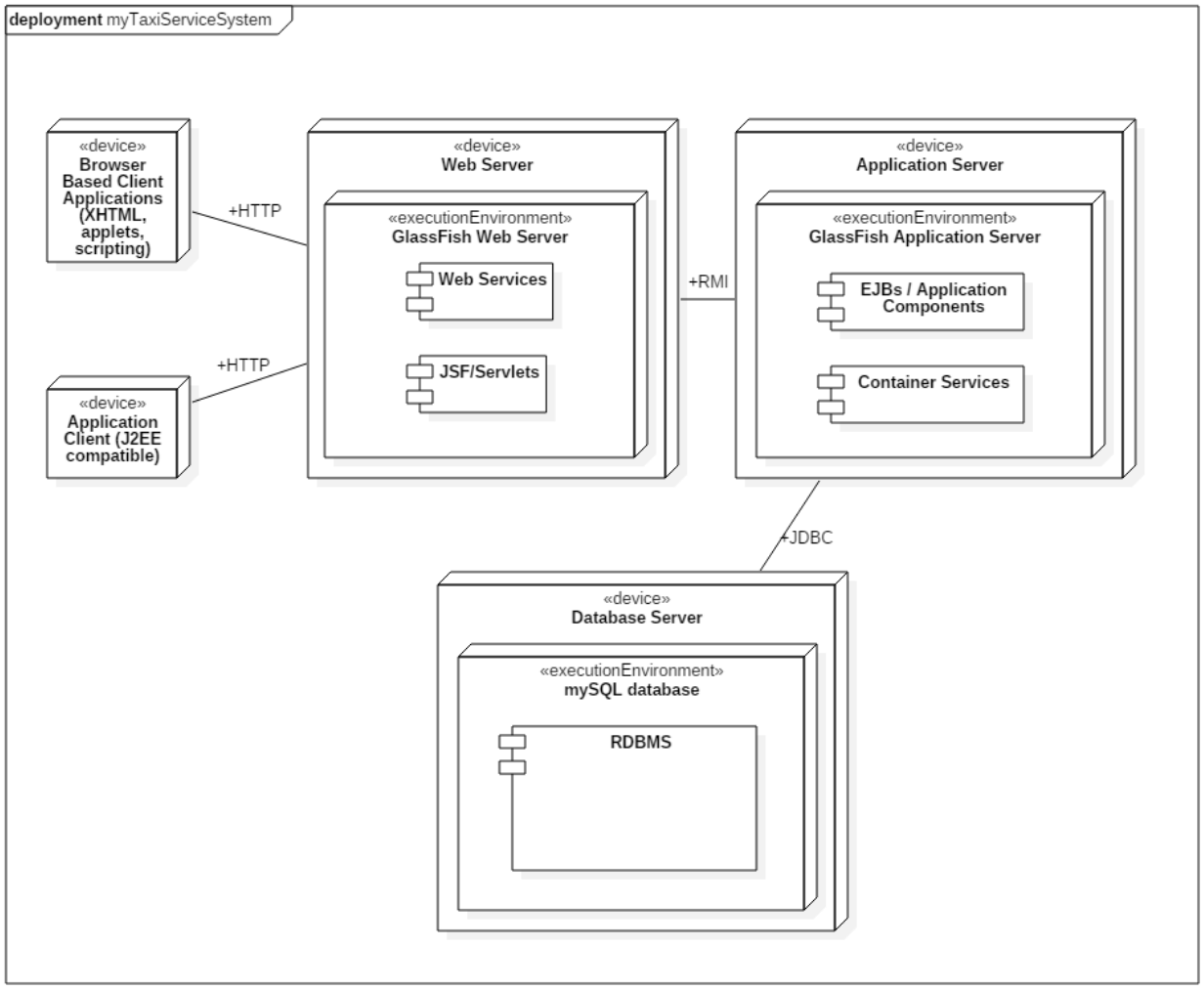


3.2.5.4 Passenger Private Area BCE



3.3 DEPLOYMENT VIEW

This section will provide a possible solution of deployment architecture.



The diagram above summarize devices and connection between the architecture components. The major features of the platform are:

- A multi-tiered application model
- A server-side component model

The client tier comprises Internet browsers, which submits HTTP request and downloads HTML pages from a Web Browser, and application clients. These last are developed with different technologies and they send HTTP request to the Web Server, but they must be compatible with the J2EE structure.

The Web tier runs a Web server to handle client requests and responds to these requests by invoking J2EE servlets or JavaServer Faces (JFPs). The server depending on the type of user request invokes Servlets. They query the business logic tier for the required information to satisfy the request and then format the information for return to the user via the server.

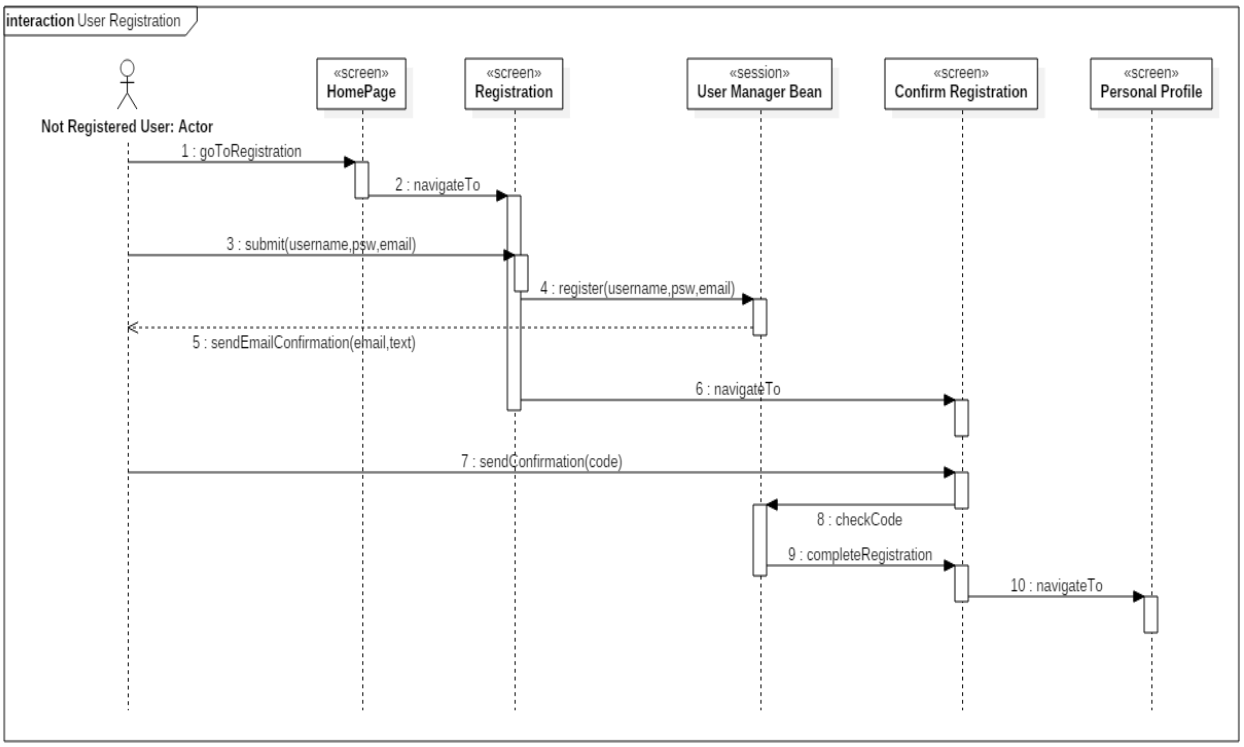
The business components of the Application Server comprise the core business logic for the application. They are realized by EJBs. EJBs receive requests from servlets in the Web tier, satisfy them usually by accessing some data sources, and return the results to the servlet. EJB components are hosted by a J2EE environment known as the EJB container.

Two different and separated machine host respectively the web server and the application server. They communicate with an implementation of Java RMI API. Developers can write remote interfaces between clients and servers and implement them using Java technology and the Java RMI APIs.

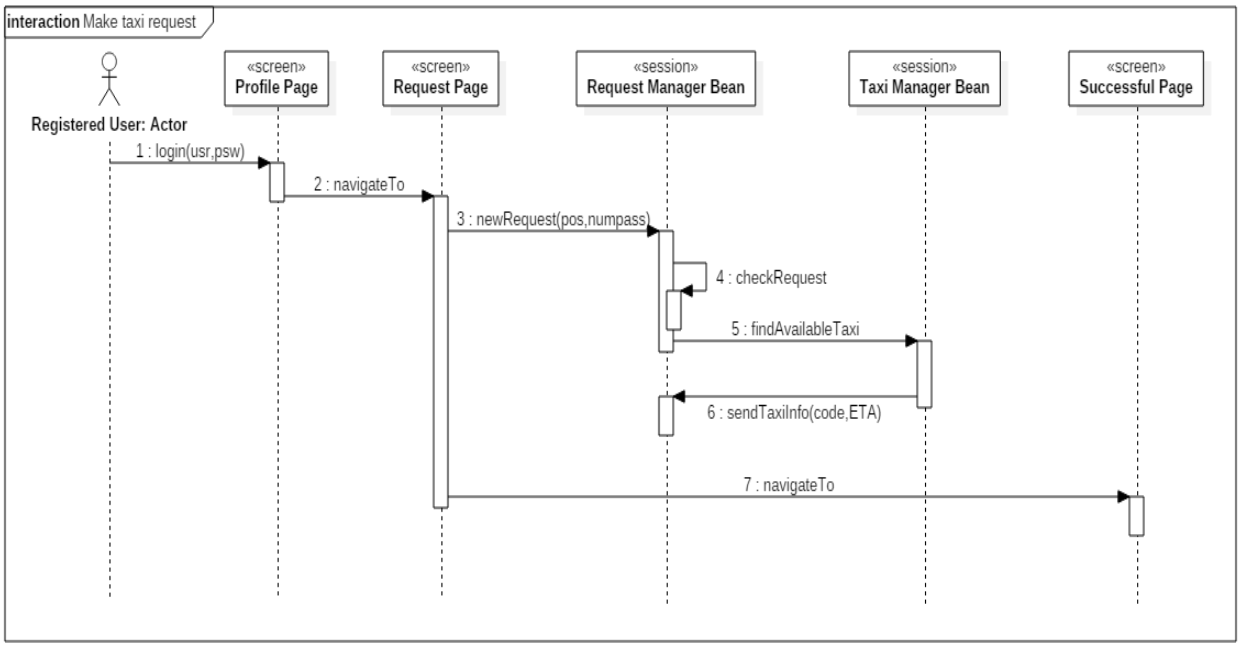
Another machine host the database tier, which consists of one or more relational databases. The application servers queries the RDBMS (Relational Database Management System) using Java Database Connectivity (JDBC).

3.4 RUNTIME VIEW

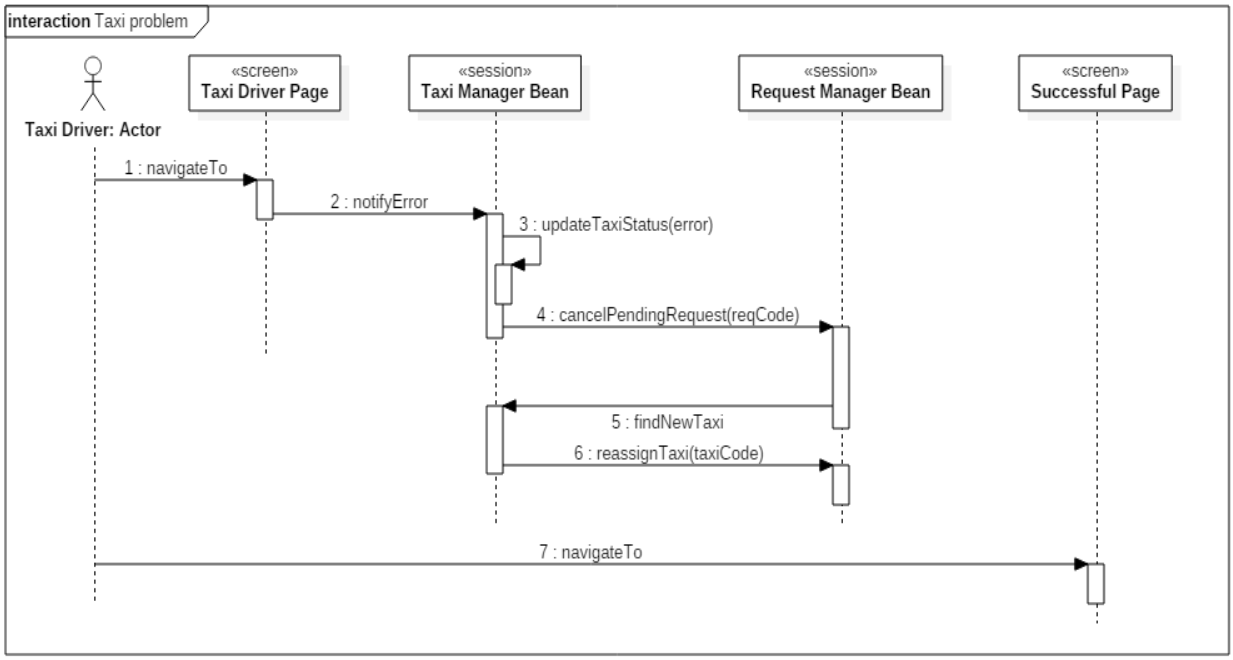
This section will provide examples of how the application will run in the future, according to the architecture the team are going to develop. The following sequence diagrams specifies how the components interact to accomplish the tasks, which are related to project scenarios and use cases.



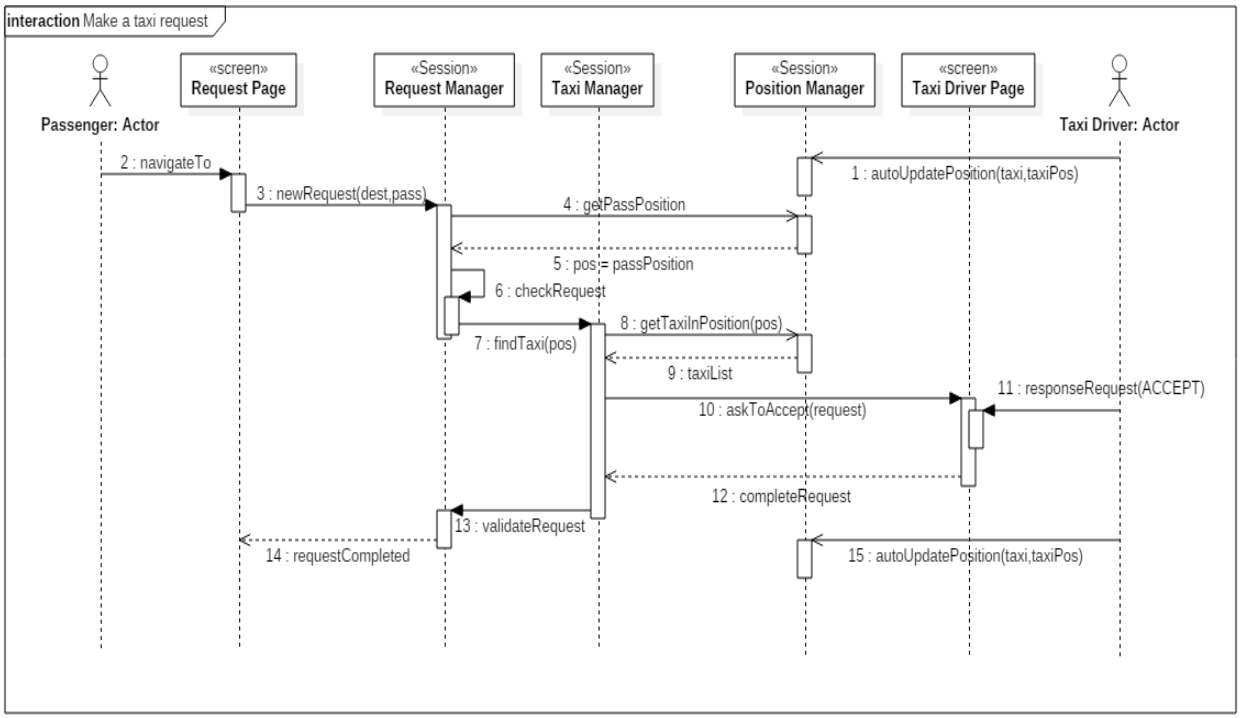
Sequence Diagram 1: it shows how the system reacts when an unregistered user want to register him/herself in the system.



Sequence Diagram 2: it shows how the system reacts when a registered user wants to make a taxi request.



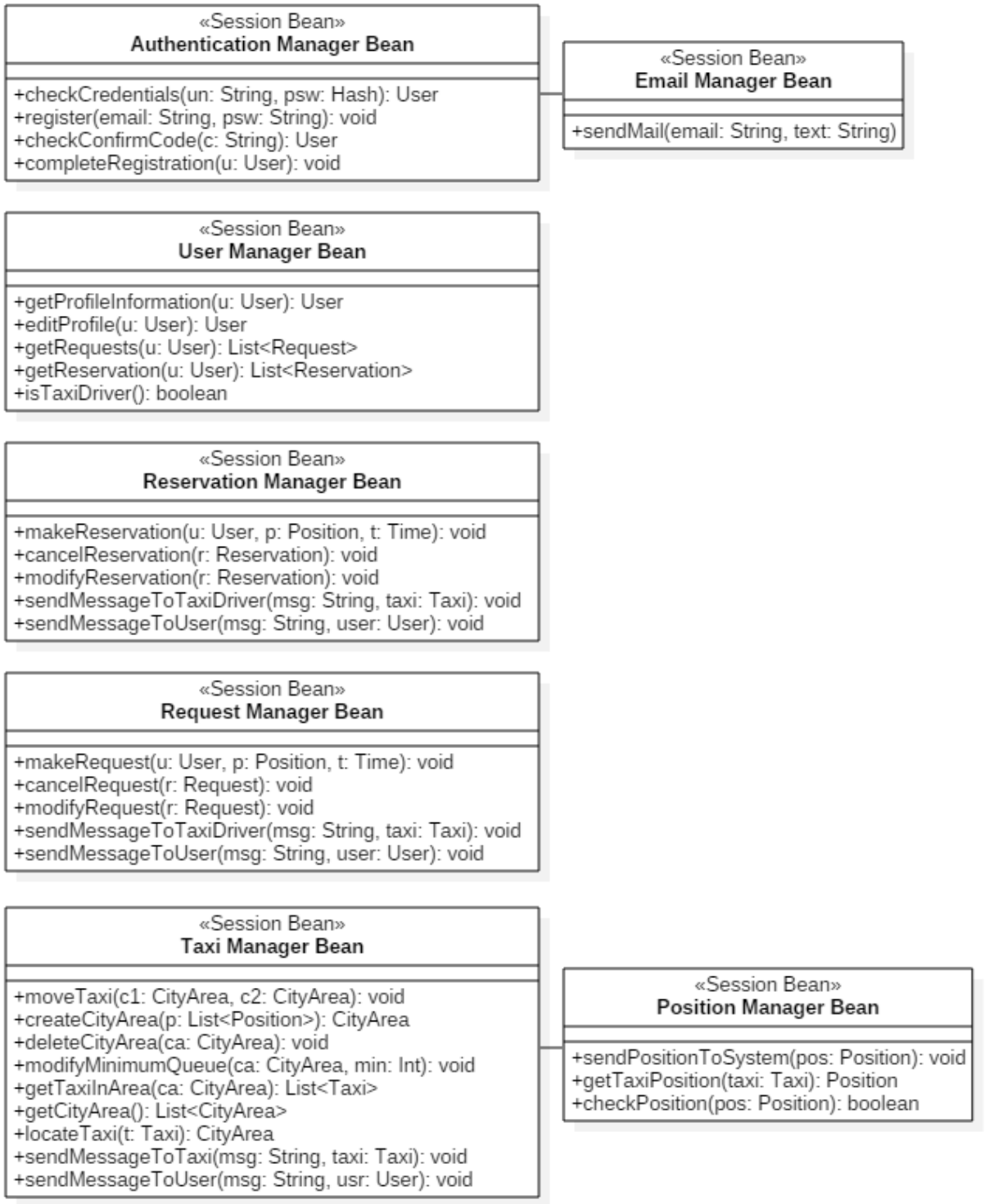
Sequence diagram 3: it shows how the system reacts when a taxi notifies a problem to the system during a request.



Sequence diagram 4: it shows how the system reacts when a user want to make a taxi request.
[Note: the taxi periodically auto-update its position]

3.5 COMPONENT INTERFACES

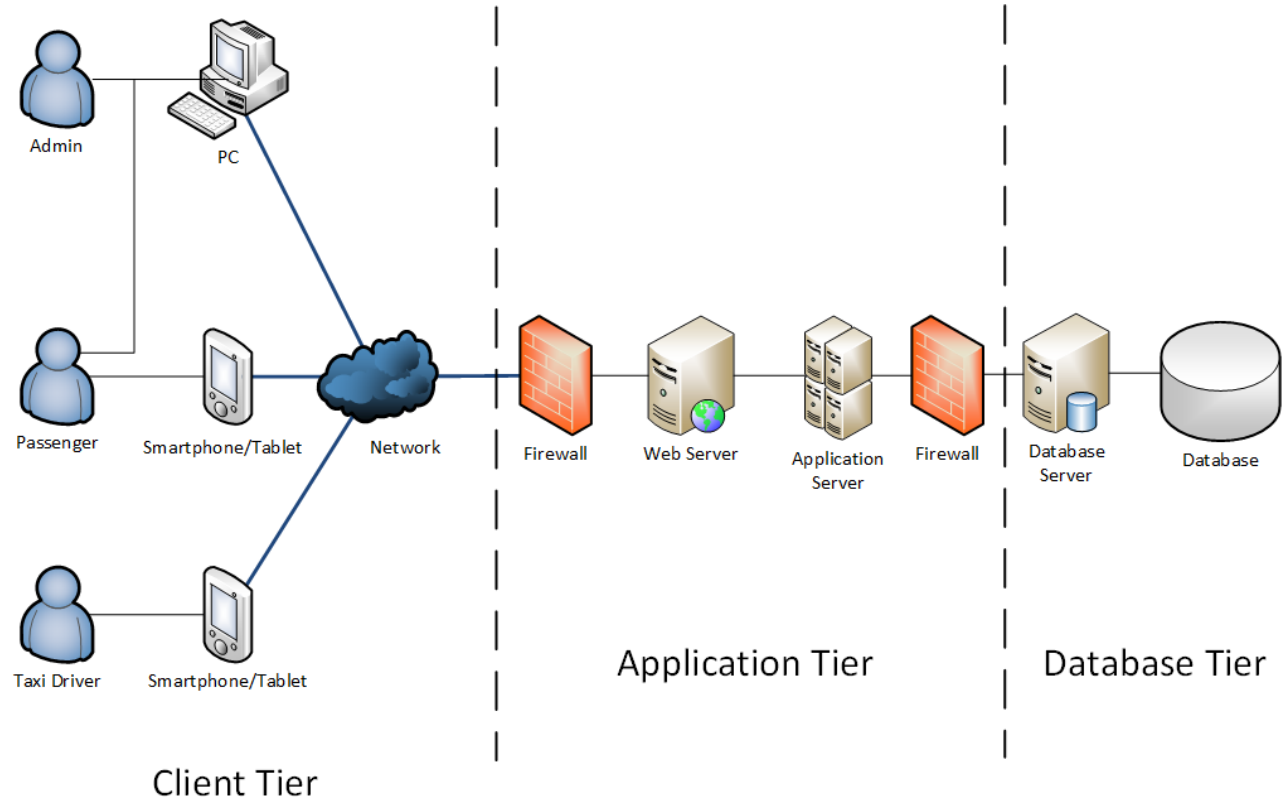
This section will describe the interfaces of the project components. The EJBs are extremely important components in the application logic. The following class diagram will points out the methods that each EJB interfaces provide to the system. The diagram contains just the most important methods, which mainly define the EJB. Of course during their implementation, other operation can be added to have a more efficient and precise system. For completeness, in the picture an Authentication Bean and an Email Manager Bean have been added. They are used to manage user registration and confirmation.



3.6 SELECTED ARCHITECTURAL STYLES AND PATTERNS

3.6.1 Multi-tiered architecture

This section explains how patterns and styles are used, why and how.
The picture below describes the general structure of the project. The architecture has been developed as a multi-tier application.



This possible multi-tiered enterprise architecture has the following major components:

- A Web browser client may or an application client, such as a smartphone or a tablet.
- An HTTP server is the Web server known to the public.
- Web containers host presentation and business logic components.
- Application containers, which host business logic components
- Relational Database Managements Systems (RDBMS) and databases, which host data and data logic.

The role of each tier and component is as follows.

Client tier. The client tier comprises an Internet browser that submits HTTP requests and downloads HTML pages from a Web server. In an application not deployed using a browser, standalone Java clients or applets can be used; these communicate directly with the business component tier.

Web tier. The Web tier runs a Web server to handle client requests and responds to these requests by invoking J2EE servlets or JavaServer Faces (JSFs). The server, depending on the type of user request, invokes Servlets. They query the business logic tier for the required information to satisfy the request and then

format the information for return to the user via the server. The code is invoked by the JSF mechanism and takes responsibility for formatting the dynamic portion of the page.

Business component tier. The business components comprise the core business logic for the application. They are realized by EJBs. EJBs receive requests from servlets in the Web tier, satisfy them by accessing some data sources, and return the results to the servlet. EJB components are hosted by a J2EE environment known as the EJB container, which supplies a number of services to the EJBs it hosts including transaction and life-cycle management, state management, security, multi-threading, and resource pooling. The Business Application servers uses EJB and session bean in order to fulfil its tasks.

Session beans contain business logic and provide services for clients. The two types of session bean are stateless and stateful. The application uses both the two types of session bean.

- A stateless session bean does not keep any state information of any client. A client will get a reference to a state less session bean in a container and can use it to make many calls on an instance of the bean. However, between each successive service invocation, a client is not guaranteed to bind to any particular stateless session bean instance. The EJB container is delegated to call to stateless session beans as needed, so the client can never be certain which bean instance it will actually talk to.
In this project, these are the most common bean used in the application, for instance in displaying profile information and manage taxi distribution.

- A stateful session bean can maintain state information about the conversation. Once a client gets a reference to a stateful session bean, subsequent calls to the bean using this reference are guaranteed to go to the same bean instance. The container creates a new, dedicated stateful session bean for each client that creates a bean instance. Therefore, clients can store any state information they wish in the bean and can be assured that it will still be there the next time they access that bean. EJB containers assume responsibility for managing the life cycle of stateful session beans.
In the project, this kind of beans are used during the registration of a new user or during the requests/reservations made by a passenger. In fact, during the creation of a request it is necessary to maintain a stateful connection between the user and the system, in order to display and refresh correctly all the information.

The business tier, in order to interface itself with the database tier and the database server, uses the typical J2EE components: the Entity Beans.

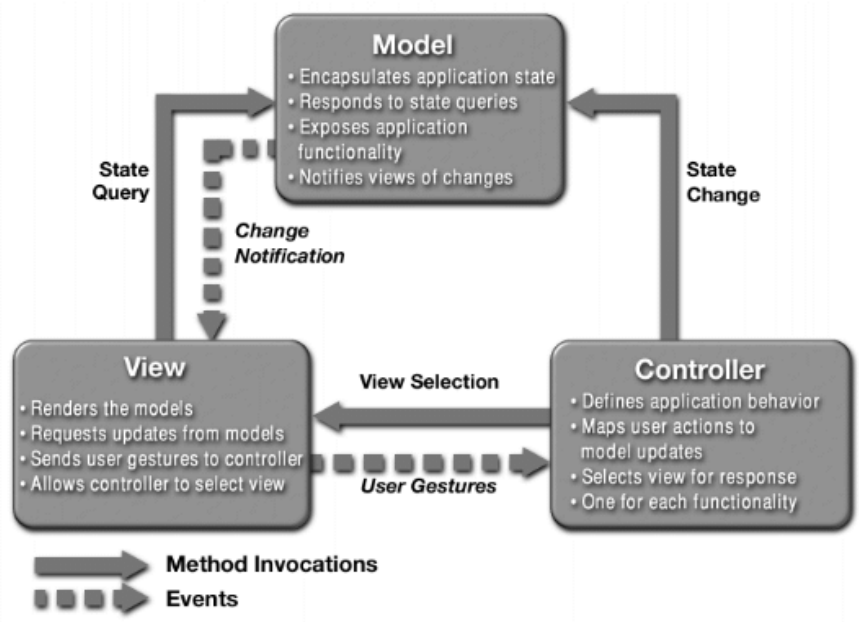
Entity beans are used for representing business data objects. In an entity bean, the data members map directly to some data items stored in the associated database. Entity beans are accessed by a session bean that provides business-level client services. The data, which the bean represents, are mapped automatically to the associated persistent data store (e.g. the database) by the container. The container is responsible for loading the data to the bean instance and writing changes back to the persistent data storage at appropriate times, such as the start and end of a transaction.

Database tier. This consists of one or more databases, which EJBs must query to process requests. JDBC drivers are used for databases, which are Relational Database Management Systems (RDBMS).

This table below resumes the pros and cons analysed, which this kind of structural implementation shows.

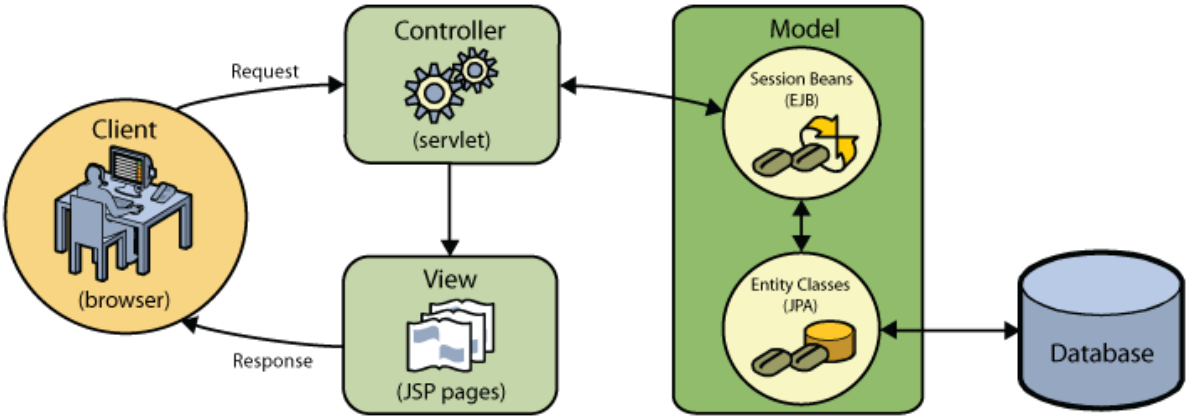
Advantages	Disadvantages
Development issues: • Complex application rules easy to implement in application server. • Business logic disjoint from database server and client, which improves performance. • Changes to business logic automatically enforced by server – changes require only new application server software to be installed. • Application server logic is portable to other database server platforms. • Greater degree of flexibility • Security can be defined for each services, and at every levels	**Development issues:** • A complex structure to develop, harder to be built. • It is more difficult to set up and maintain.
Performance: • Superior performance for medium to high volume environments. • Increased performance, as tasks are shared between servers	**Performance:** • The physical separation of application servers, containing business logic functions, and database servers, containing databases, may moderately affect performance

3.6.2 MVC (Model-View-Controller) Pattern



The architecture described in the previous paragraph implements this MVC model, in which the components have three different tasks.

- ***Model***: Represents the business data that govern access to and modification of the data. The model notifies views when it changes and lets the view query the model about its state. It also lets the controller access application functionality encapsulated by the model.
- ***View***: The view renders the contents of a model. It gets data from the model and specifies how that data should be presented. It updates data presentation when the model changes. A view also forwards user input to a controller.
- ***Controller***: The controller defines application behaviour. It dispatches user requests and selects views for presentation. It interprets user inputs and maps them into actions to be performed by the model. A controller selects the next view to display based on the user interactions and the outcome of the model operations.



From the project's point of view, the *View* will corresponds to the presentation/client level, whereas *Model* and *Controller* will be part of application logic.

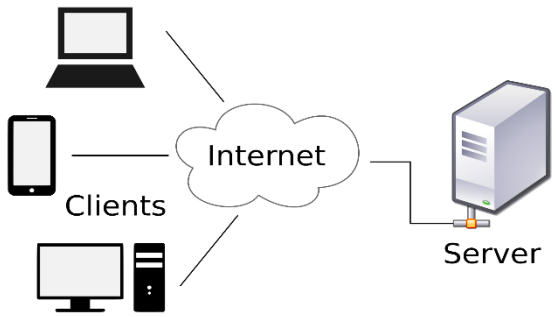
The MVC design pattern provides numerous benefits:

Separation of design concerns: Because of the separation of presentation, control and data persistence, the application becomes more flexible; modifications to one component have minimal impact on other components. For example, you can create new views without needing to rewrite the model.

More easily maintainable and extensible: Good structure can reduce code complexity. As such, code duplication is minimized.

Promotes division of labour: Developers with different skill sets are able to focus on their core skills and collaborate through clearly defined interface.

3.6.3 Client-Server architectural style



The myTaxiService application will be developed according to the well-known Client-Server paradigm. The user and their devices are the client, which connect to the server in order to use the application services. Other typology of architecture has been analysed to fulfill the non-functional requirements, but the Client-Server style has been selected as the best one.
We summarize the advantages and disadvantages that this kind of approch carries out.

Advantages:

1. **Centralization :** Unlike P2P, where there is no central administration, here in this architecture there is a centralized control. Servers help in administering the whole set-up. Access rights and resource allocation is done by servers.
2. **Proper Management :** All the files are stored at the same place. In this way, management of files becomes easy. Also it becomes easier to find files.
3. **Backup and Recovery possible :** As all the data is stored on server its easy to make a back-up of it. Also, in case of some break-down if data is lost, it can be recovered easily and efficiently. While in peer computing we have to take back-up at every workstation.
4. **Upgradation and Scalability in Client-server setup :** Changes can be made easily by just upgrading the server. Also new resources and systems can be added by making necessary changes in server.
5. **Accessibility :** From various platforms in the network, server can be accessed remotely.
6. As new information is uploaded in database, each client need not have its own storage capacities increased. All the changes are made only in central computer on which server database exists.
7. **Security :** Rules defining security and access rights can be defined at the time of set-up of server.
8. Servers can play different roles for different clients.

Disadvantages:

1. **Congestion in Network**: too many requests from the clients may lead to congestion. Overload can lead to breaking-down of servers.
2. Client-Server architecture is not as robust as other kind of network(i.e. P2P) and if the server fails, the whole network goes down.
3. **Cost** : It is expensive to install and manage this type of computing, expecially in case it is necessary to expand the architecture.
4. Professional IT people have to maintain the servers and other technical details of network.

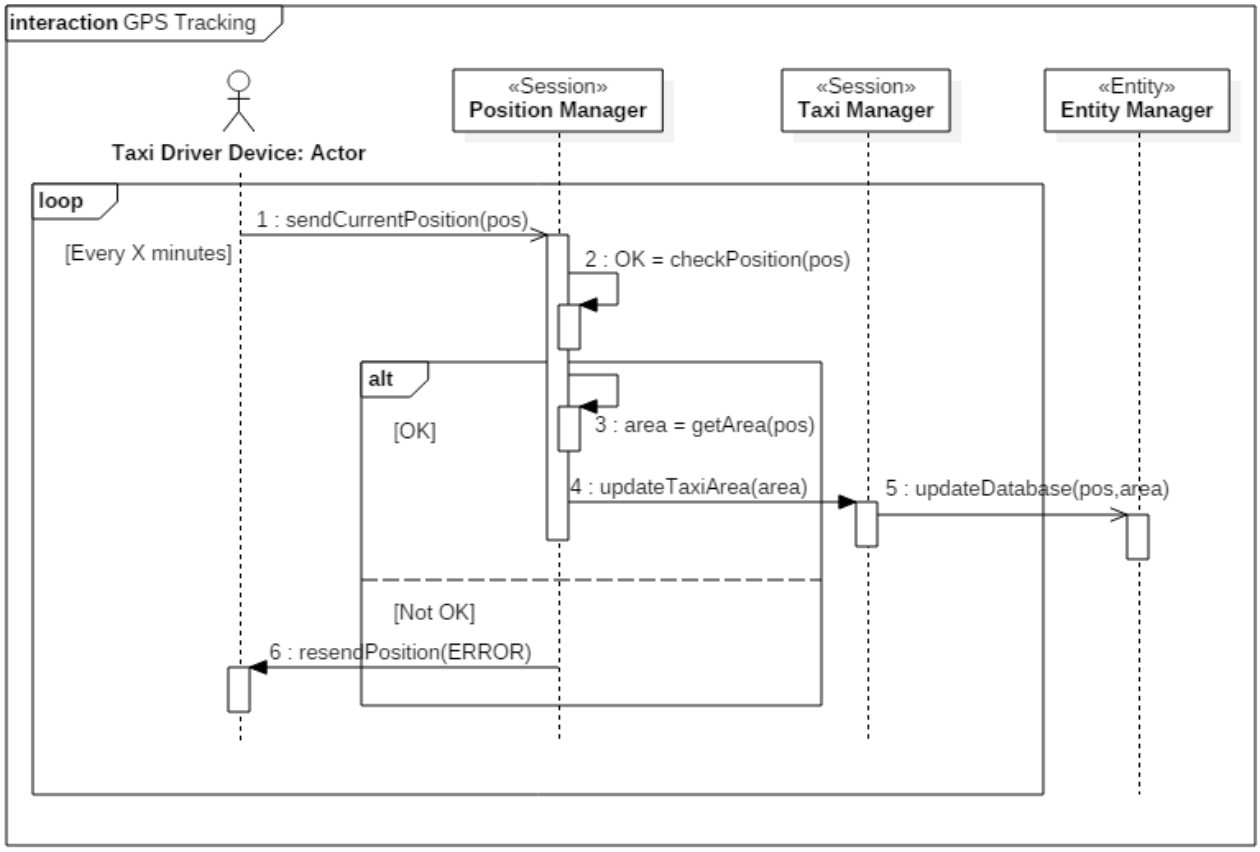
3.7 OTHER DESIGN DECISIONS

3.7.1 GPS Tracking and position management

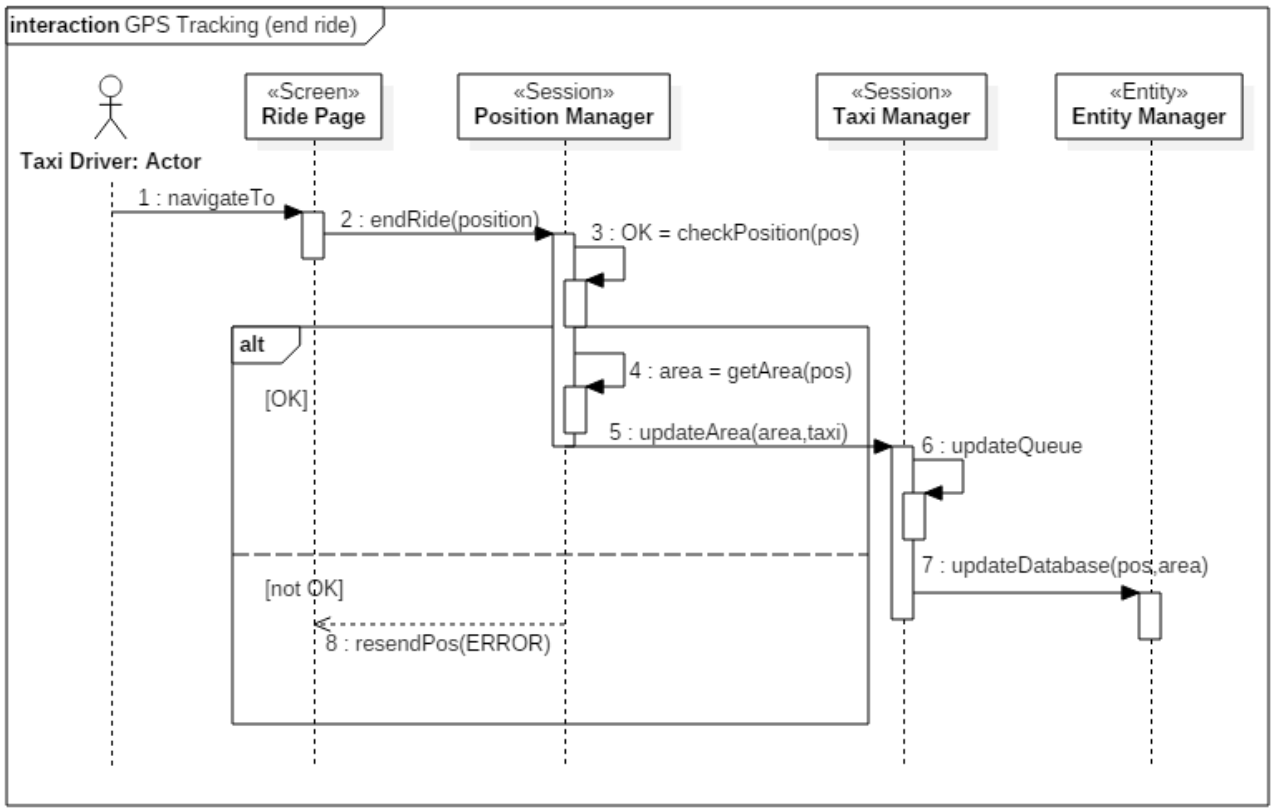
This section deals with the system's part dedicated to the management of the taxi position in the city. The following sequence diagrams show the interaction between the different components in the system.

The idea approached to update periodically the position is the following.

1. The taxi driver device sends periodically its position, using the integrated GPS in the mobile phone to the server.
2. The server receives the position, validate it and extract the city area related to that position. If an error occurs, it asks for the position again. The server will use a buffer to manage all the incoming position message and it manages them in an asynchronous way.
3. The Entity Manager in the application server will take care of all these incoming messages, in order to update the current position and areas of the taxis in the database. This can be done asynchronously as well.



An important scenario to be considered in the position management is the following.
When the taxi driver finish its ride with a passenger, he/she informs the system of his/her position using a dedicated button. The system uses this information to update the data of the city area and the queue, in which the taxi has arrived. These update operations should be forced with respect to the periodically ones. This is necessary because the queue of the areas must be always update to perform correctly the users' requests.



4 ALGORITHM DESIGN

This section describes the main algorithms involved in the design of myTaxiService application through pseudocode.

The main algorithms that we describe in this section are about:

- Updating the area associated with a taxi.
- Sending the request to the correct taxi driver.
- Reallocation of the taxis.

4.1 UPDATING THE AREA ASSOCIATED WITH A TAXI

This algorithm associates the correct area in according to the current position of a taxi driver.

When a taxi becomes available, the system calls a function in order to insert the taxi in the correct area, according to its position. The parameters of the function are:

- The taxi involved in the updating of the area: taxi
- The collection of all the areas: area_Col
- The current position of the taxi: current_Position

4.1.1 Pseudocode

```
1.  BEGIN
2.  IF TAXI IS NOT AVALAIBLE THEN
3.          TERMINATE
4.  AREA = GET AREA THAT CONTAINS CURRENT_POSITION FROM AREACOL
5.  CHOOSED_AREA = AREA
6.  DIFFERENCE = AREA.MINIMUM_NUMBER_OF_TAXIS - NUMBER OF TAXIS AVAILABLE IN
    AREA
7.  FOR EACH AREA2 IN AREA_COL.ADJACENT(AREA)     /*ADJACENT ZONES AROUND AREA */
8.          DIFFERENCE2 = AREA2.MINIMUM_NUMBER_OF_TAXIS - NUMBER OF TAXIS
    AVAILABLE IN AREA2
9.          IF DIFFERENCE2 > DIFFERENCE THEN
10.                 CHOOSED_AREA = AREA2
11.                 DIFFERENCE = DIFFERENCE2
12. QUEUE = GET TAXIS'QUEUE OF CHOOSED_AREA
13. INSERT TAXI IN THE LAST POSITION OF QUEUE
14. INFORM THE TAXI DRIVER OF TAXI OF THE NEW AREA ASSIGNED
15. END
```

4.2 SENDING THE REQUEST TO THE CORRECT TAXI DRIVER

This algorithm try to find a taxi for a passenger who have sent a request.

When a passenger uses the application to find a taxi and sent a request, the system receives it and calls a special function. The parameters of the function are:

- The request of the passenger that contains also the origin and the number of passengers: request
- The passenger who have requested the taxi: passenger

- The collection of all the areas: area_Col

4.2.1 Pseudocode

1. *BEGIN*
2. FOR EACH **AREA** IN **ADJACENT_AREAS** = **AREACOL.ADJACENT(REQUEST.POSITION)**
3. **QUEUE** = GET TAXIS' QUEUE OF **AREA**
4. FOR EACH **TAXI** IN **QUEUE**
5. **TAXI** = GET THE FIRST TAXI IN **QUEUE**
6. **ANSWER** = **NEGATIVE**
7. IF **REQUEST.NUM_OF_PASSENGERS** <= **TAXI.NUM_OF_PASSENGERS**
8. NEXT **TAXI** IN **QUEUE**
9. IF **TAXI** IS **AVAILABLE**
10. SEND **REQUEST** TO **TAXI** AND WAIT FOR THE ANSWER
11. **ANSWER = TAXI.ANSWER**
12. IF **ANSWER** IS **NEGATIVE** THEN
13. INSERT **TAXI** IN THE LAST POSITION OF **QUEUE**
14. NEXT **TAXI** IN **QUEUE**
15. IF **ANSWER** IS **POSITIVE** THEN
16. INFORM **PASSENGER**
17. EXTRACT **TAXI** FROM THE **QUEUE**
18. **TAXI** BECOMES **UNAVAILABLE**
19. TERMINATE LOOP
20. NEXT **AREA** IN **ADJACENT_AREAS**
21. INFORM **PASSENGER** THAT THERE AREN'T TAXIS AVAILABLE NEARBY
22. *END*

IMPORTANT OBSERVATION: This algorithm simply search a taxi in adjacent areas with respect to passenger position. This algorithm can be improved and used also recursively. In fact it could be possible to visit other areas with different distances from the passenger. More precisely, it is possible to perform a breadth-first visit of the areas, starting from the passenger area, in order to find a taxi.

4.3 REALLOCATION OF THE TAXIS

This algorithm is executed by the system when an area reached the minimum number of taxis available in order to find a taxi available to move in that area.

The parameters of the function are:

- The collection of all the areas with their taxis' queue: area_Col
- The area with a low number of taxis: area
- The maximum distance of areas from the area with a low number of taxis: distance

4.3.1 Pseudocode

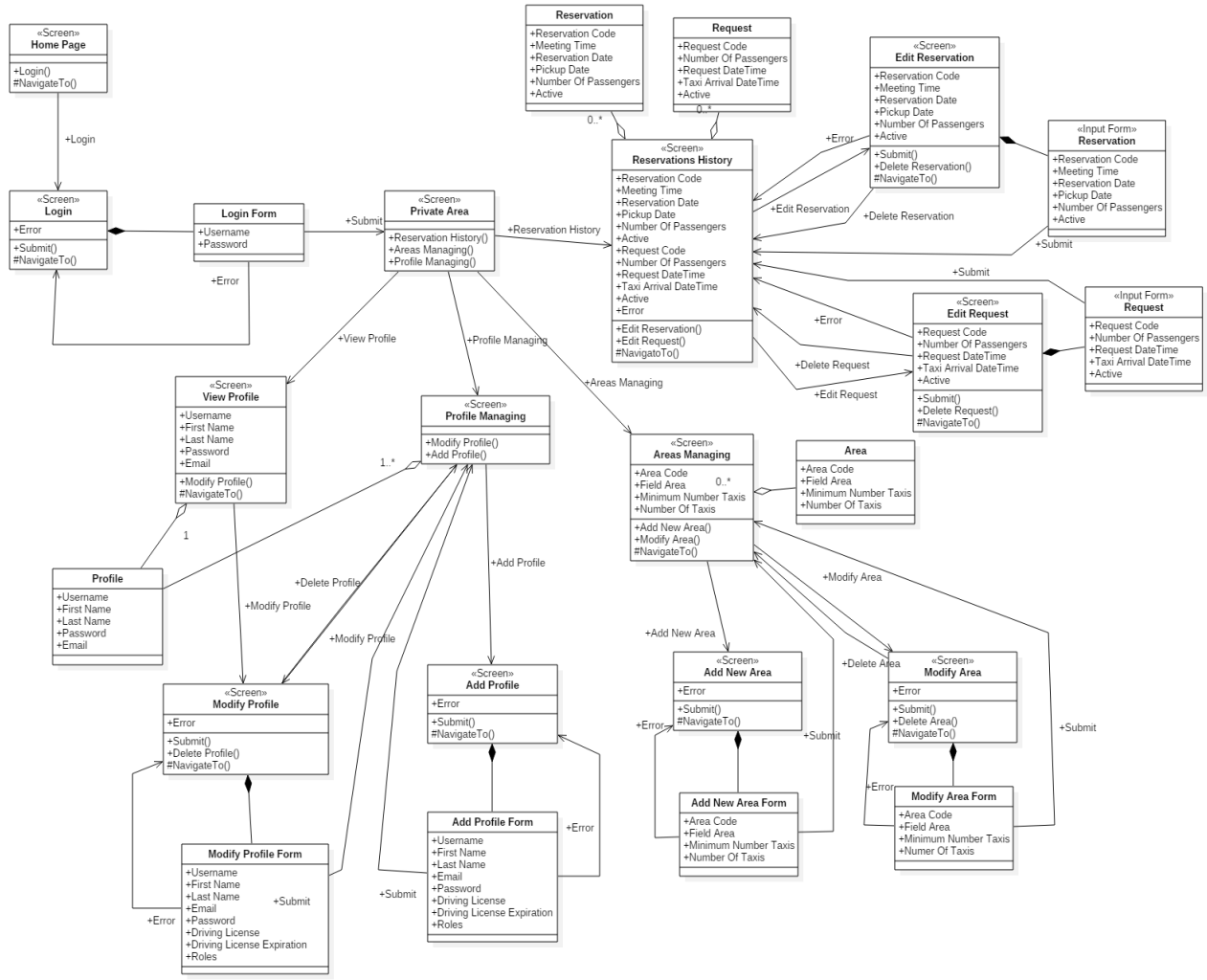
The approach used in this algorithm is to execute a breadth-first visit of the data structure, which contains all the areas.

1. *BEGIN*
2. **CHOOSED_AREA =** NULL
3. **DIFFERENCE = -∞**
4. FOR EACH **AREA2** IN **AREA_COL** NEAREST TO **AREA** WITH A DISTANCE LESS OR EQUAL TO **DISTANCE**
5. **DIFFERENCE2** = NUMBER OF TAXIS AVAILABLE IN **AREA2** – **AREA2**.**MINIMUM_NUMBER_OF_TAXIS**
6. IF **DIFFERENCE2** > **DIFFERENCE** THEN

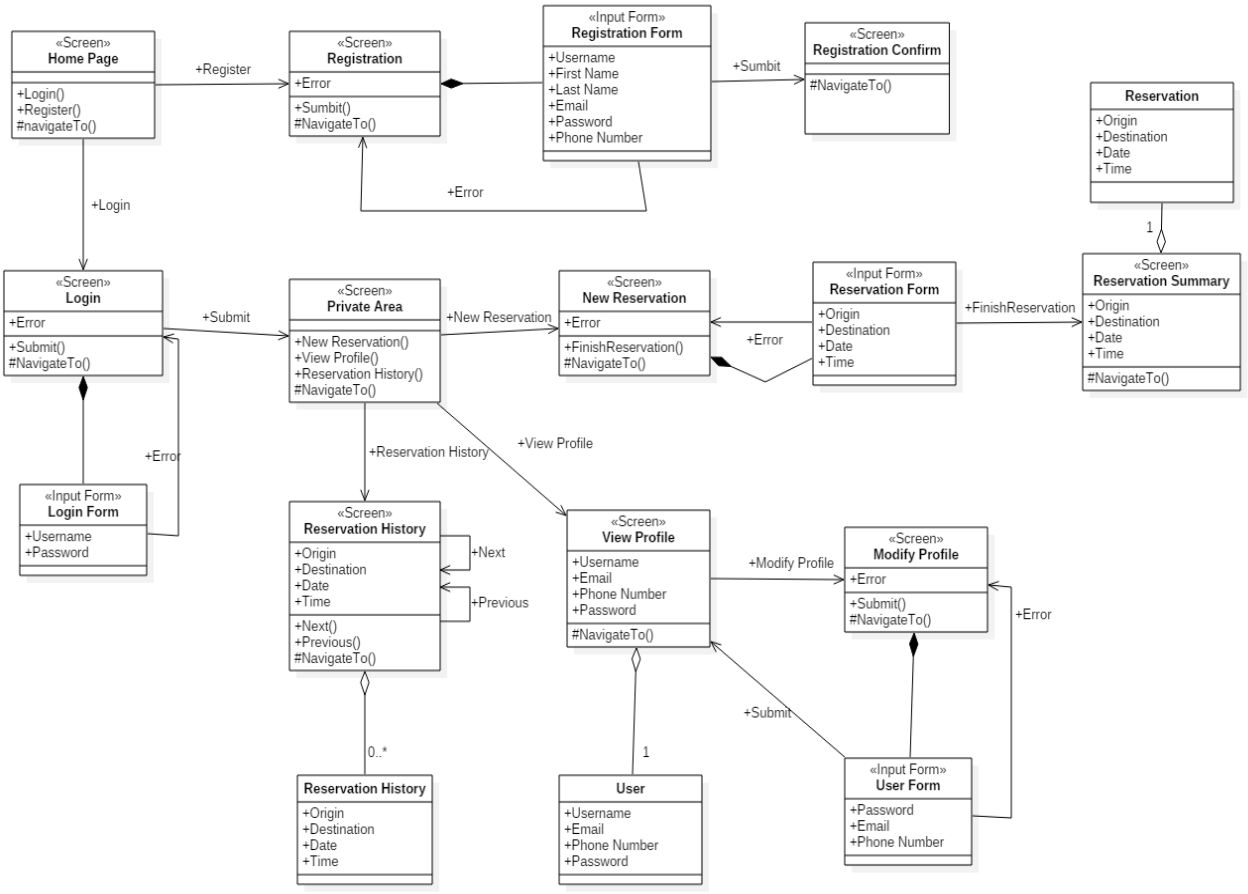
7. **CHOOSED_AREA** = AREA2
8. **DIFFERENCE** = **DIFFERENCE2**
9. IF **DIFFERENCE** > 0 THEN
10. **TAXI** = REMOVE THE LAST AVAILABLE TAXI IN **CHOOSED_AREA**.**TAXIS_QUEUE**
11. INSERT **TAXI** IN THE LAST POSITION OF **AREA**.**TAXIS_QUEUE**
12. INFORM **TAXI** OF THE NEW AREA ASSIGNED
13. *END*

5 USER INTERFACE DESIGN

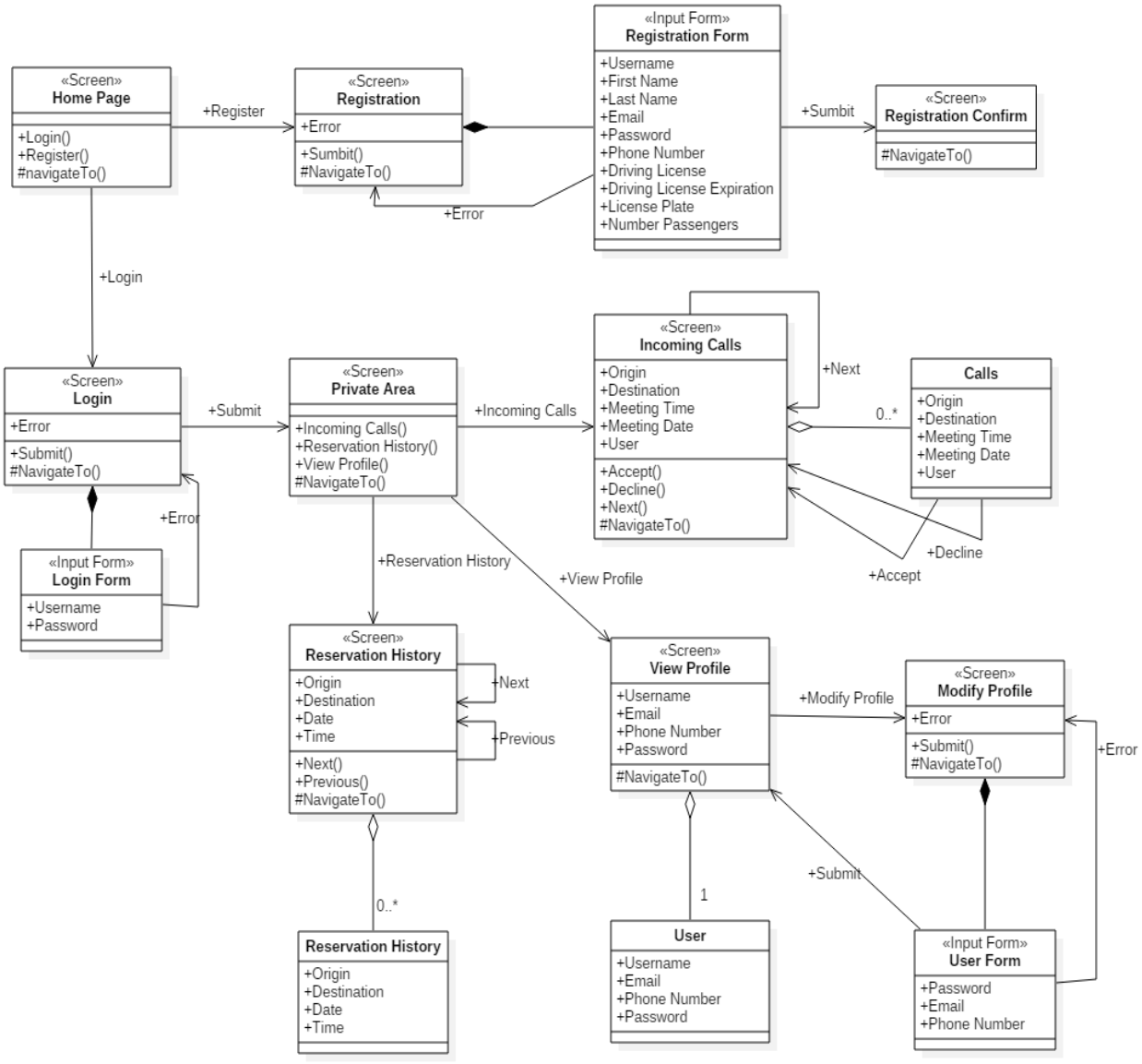
This section describes the user interfaces of the Web application that will be implemented in the client side. The notation used to describe the UX is based on UML class diagram. In order to be clearer, the following UX diagram are separated by user's role and functionalities. Moreover, they do not present in details certain screens of the application, which are considered secondary. The three diagram correspond respectively to administrator UX, passenger UX and taxi driver UX.



UX diagram 1: Administrator user interface



UX diagram 2: Passenger user interface



UX diagram 3: Taxi driver user interface

6 REQUIREMENTS TRACEABILITY

6.1 PACKAGES AND FUNCTIONALITIES RELATIONS

This section recap all the functionalities highlighted in RASD document and maps them to the project's packages.

6.1.1 Functionalities Recap

Here are listed all the functional requirements pointed out in RASD document.

6.1.1.1 Managing profiles

Functional requirements

[FR1]. View personal information
[FR2]. Modify personal information
[FR3]. Manage personal reservation/request

6.1.1.2 Managing users

Functional requirements

[FR4]. Register to the system
[FR5]. Login
[FR6]. Logout
[FR7]. Modify password
[FR8]. Recover password
[FR9]. Delete an user from the system
[FR10]. Support, at the same time, a user with different roles (admin, taxi driver and passenger).All the registered user will log in with his/her username and password. Depending on the roles the user has in the system, the application displays a menu to select in which mode he/she wants to use myTaxiService.

Non-functional requirements

a. User password must be stored securely, with certified cryptography.
b. User password must be at least 8 characters and it must be changed every 3 months.
c. System must support a high number of users (at least 500.000).

6.1.1.3 Managing taxis request and reservation

Functional requirements

1. Users must log in order to use the application.
2. All users' devices must connect to the Network in order to use the application.
3. Passenger can request a taxi through either a web application or a mobile app.
4. The system answer to the request informing the passenger about the code of the incoming taxi and the waiting time.
5. Taxi drivers inform the system about their availability and confirm that they are going to take care of a certain call.
6. A user can reserve a taxi by specifying the origin and the destination of the ride.

[FR6.1]. The reservation has to occur at least two hours before the ride.

[FR6.2]. In the case of correct reservation, the system confirms the reservation to the user and allocates a taxi to the request 10 minutes before the meeting time with the user.

[FR6.3]. In the case the user wants to cancel the reservation, he must do it two hours before the meeting time.

[FR6.4]. The system will treat a reservation as a future request, so it simply calls taxi before the reservation appointment in the preselected zone.

7. If a user reserves or require a taxi and he does not use it for more than 3 times, the administrator block his account for a month.

Non-functional requirements

a. The system must support tens of request arriving in seconds.

b. The system must be responsive(maximum 1-3 seconds to react to user's input)

c. The system must be user-friendly. A passenger must be able to make a request in at maximum 10 seconds.

6.1.1.4 Managing taxi distribution and reallocation

Functional Requirements

8. The system guarantees a fair management of taxi queues.

9. The service needs to put in relationship customer and driver finding the closest driver.

10. The system must localized (using GPS or Internet connection) the user's device in order to manage the distribution of the taxis.

11. The city should be divided in taxi zones (approximately two km^2 each, with a minimum of one km^2 and a maximum of three km^2). Taxis send periodically their position to the system.

[FR11.1]. Each zone is associated to a queue of taxis.

[FR11.2]. It is possible to cancel just the most external areas, but before the administrator must move all the taxis in another zone.

[FR11.3]. The system automatically computes the distribution of the taxis in the various zones based on the localization information it receives from the taxi.

[FR11.4]. When a taxi is available, its identifier is stored in the queue of taxis in the corresponding zone.

[FR11.5]. When a request arrives from a certain zone, the system forwards it to the first taxi queuing in that zone.

[FR11.6]. Every zone should have a minimum number of taxis, according to certain decisional parameters (density of people, number of possible calling, exposition building…).

[FR11.7]. The sum of all minimum queues values and the number of zones must be the same as the number of taxi of the service (available and not).

12. If the taxi confirms the request, then the system will send a confirmation to the passenger.

13. If the taxi does not confirm the request, then the system will forward it to the second in the queue and, at the same time, it will move the first taxi in the last position in the queue.

14. If a certain zone reach the minimum number of taxis, then the system explore the adjacent zones to find the queue with the largest number of taxis available and move one of them to that zone (it will put at the end of the new queue). This algorithm is applied recursively.

[FR14.1]. In the case all queues in the adjacent zones are below the minimum threshold, the system does nothing and wait until a taxi becomes free.

15. If a customer deletes a request before the taxi arrives or a taxi is arrived to destination, the system stops the taxi driver and put him in the queue of the current zone or the adjacent ones. The system adds the taxi to the queue in which the difference between the minimum number of taxi and the current number of them is the biggest (if there are more possibilities, the choice is random).
16. A customer can cancel the taxi request only if it is elapsed less time than the half of arrival time given by the taxi driver. For instance if the arrival time indicated is four minutes, it is possible to cancel the request within two minutes from the demand.
17. The customer who requires a taxi can specify the number of passengers with him. So that the system can choose the correct car, if available. If it is not possible to fulfil the request, the system will advise the customer.
18. If no taxi in a certain zone are available, i.e. all refuses the request, the system tells the customer that there are not free taxi in his area so it will take more time to have a cab. The customer can decline or accept. If he accepts, a taxi in adjacent areas is used, until the system can find a taxi. After that, the customer can accept or decline the choice.
19. In the rare case, no taxis are available in whole city, i.e. all the taxis refuse a request, the system tells to the customer that he has to try in another moment to require a taxi, because no one is free.
20. The system administrator decides the minimum number of taxi in each area, depending on how much taxis are request for that day in a certain zone. For instance, if it will be a football match in the evening, the admin will increase the minimum number of taxi in the areas near the stadium, so that a certain amount of cars will move towards that zone.

6.1.2 Detailed packages and requirements

As we have listed all the functional requirements, we can assign them within specific component in the package diagram as follows:

- **User UI package**
  The Client Tier and the Web Tier implement these packages.
  - User pages. This package implements: **[FR4] – [FR10].**
  - Profile Pages. This package implements: **[FR1] – [FR3].**
  - Request Pages. This package implements: **[FR3].**
  - Reservation Pages. This package implements: **[FR3].**

- **Business Logic package**
  The Business Logic tier implements these packages.
  - Request manager. This package implements: **[FR11] – [FR15], [FR17].**
  - Reservation manager. This package implements: **[FR11] – [FR17].**
  - User manager. This package implements: **[FR4] – [FR10].**
  - Taxi distribution manager. This package implements: **[FR18] – [FR30].**

- **Persistence data package**
  The Database Tier implements these packages.
  - Entity Manager. This package implements: **[FR1]** – **[FR30].**

6.2 REQUIREMENTS TRACEABILITY MATRIX

This table below maps all the functional requirements listed in the previous RASD document and the functionalities, i.e. the session beans to develop, which the system offers.

Requirements traceability matrix		**Functional requirements**																													
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Functionalities implemented	User Manager Bean	X	X	X	X	X	X	X	X	X	X	X	X					X													
	Request Manager Bean			X								X	X	X	X	X	X	X				X	X			X	X	X	X	X	
	Reservation Manager Bean			X								X	X	X	X	X	X	X				X	X			X	X	X	X	X	
	Taxi Manager Bean														X	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X

7 APPENDIX

7.1 DESIGN DOCUMENT ADJUSTMENTS

List of modification done in this Document Design version 2.

1) Gantt chart moved in Rasd document version 3.
2) Section “Other design decision” modified. Now it describes the management of GPS tracking and taxi position in the system.
3) In the “Runtime view” a sequence diagram, which describes the creation of a new request, has been added.

.