SE2 Design Document

Edoardo Giacomello Mattia Fontana

December 3, 2015

Contents

1 Introduction

1.1 Purpose

1.2 Scope

1.3 Definitions, Acronyms, Abbreviations

GUI : Graphical User Interface. It is the set of graphical elements such text, buttons and images which the user can interact with.

Thin Client : It is a design style for the client in which the application running on the client contains the least business logic possible. In our case the client application will only serve as a presentation interface.

1.4 Reference Documents

- Structure of the Design Document

1.5 Document Structure

2 Architectural Design

2.1 Overview

The MyTaxiService application has been designed to run on a client-server architecture.
In particular, the general system has been divided in some sub-systems and each of those has been designed to run on a different phisical machine. Since the modularity of the system has been taken into account, eventual further design decisions that could increase or decrease the tier size of the architecture will be supported.
The sub-systems that have been identified are the following:

Client Application (1..N) : This is the user frontend of the system. It cames in form of a web-interface or a mobile web application and it has two possible realizations which are the Passenger Interface or the Taxi Driver interface. There's included also a realization that is currently designed only in a web interface form, which is the system administrator interface.

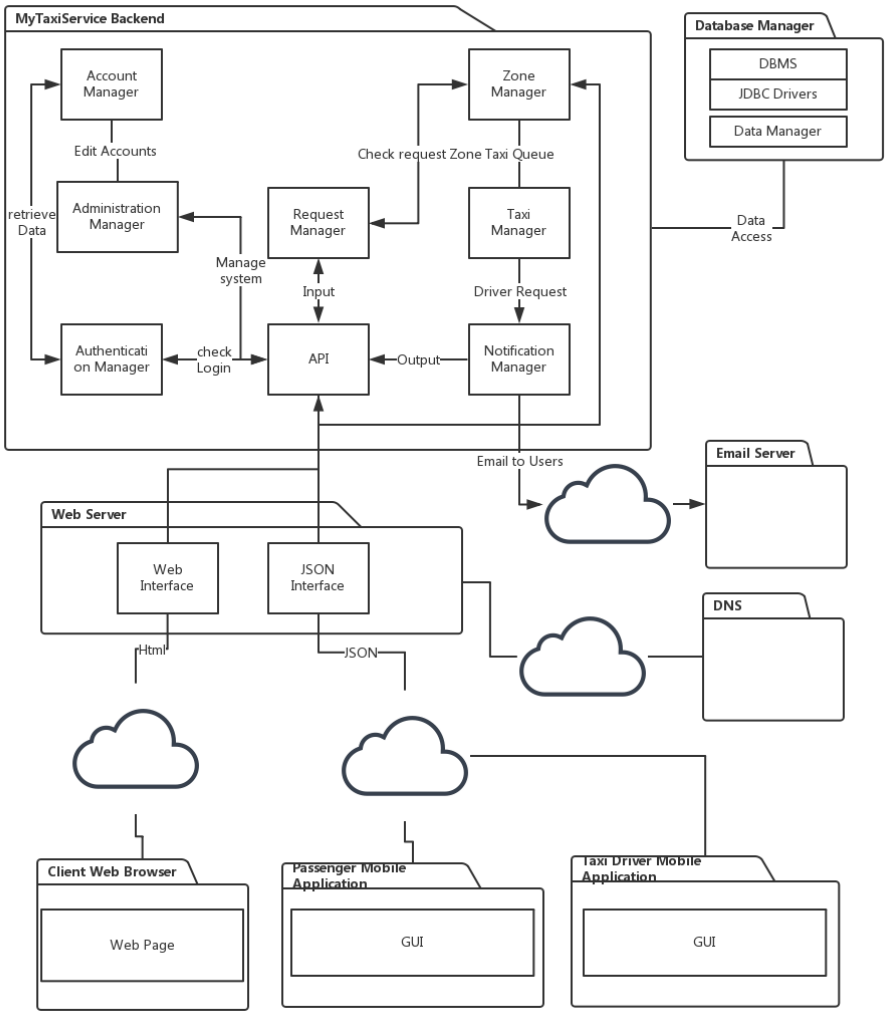
Web Server (1..N) : This is the main interface between the Backend and the Client application. The web-server could be distributed on several machines that are managed by a load-balancer, and should provide a firewall as well. The main purpose of this sub-system is that to generate the web-pages for the web-interface and forwarding/translating requests from the client and the backend.

Backend (1) : This is the sub-system in which the business logic resides.

Database (1..N) This is the sub-system in which the business data resides. It can be replicated or distributed for avoiding data corruption and undesired behaviours.

Third Party Services (N) Those sub-systems are auxiliary third-party services that the system can avail itself of for providing core services that will not be directly implemented on the system, such that email notification, DNS lookups, Map Services, etc.

2.2 High level components and their interaction



Here follows the high-level description of each component or module of the application, which is devided in a particular subsystem.

1. **Client Application**
   (a) **Website** This is the web interface the passengers and administrators can access from a web browser. It will make use of AJAX for fetching data from the server.
   (b) **GUI** These are the user interfaces for passenger or taxi drivers.
   (c) **Mobile Client** This is a client for making requests to the Server and retreiving results
2. **Web Server**
   (a) **Browsing Request Interface** This component hosts the web interface and generates web pages for displaying results to the user
   (b) **Resource Request Interface** This component collects all the requests from user web and mobile interfaces and redirect them to the server APIs.
3. **DataBase Manager**
   (a) **DBMS** This component is the actual DBMS which manages the business data.
   (b) **JDBC Drivers** Drivers that provide an interface between the DBMS and the application.
   (c) **Data Manager** This component will host all the functions useful for managing data on the server and it will offer an interface of the database for the backend. In particular, the SQL and DDL queries are stored here.
4. **Backend**
   (a) **API** This is the main interface for the backend and will provide all the system functionalities that can be accessed by other subsystems. It will parse all incoming JSON requests and activate the other components accordingly. It will also support an access for system developers, that will be regulated by an authorization token mechanism.
   (b) **Authentication Manager** This component will be in charge of managing all the authentications for the system. In particular it will manage the login functionality and the authorization token for accessing the APIs.

(c) **Account Manager** This component will manage the user accounts and the password reset service.

(d) **Request Manager** This component will host the core algorithm which manages the taxi queues and will process the requests and reservations.

(e) **Taxi Manager** This component will manage the distribution of the active taxis and the notification of incoming requests by interacting with the Notification Manager.

(f) **Notification Manager** This component will send notifications to the users, such as email for the passengers and notifications for the taxi drivers.

(g) **Zone Manager** This component will manage the zones in which the area is divided. It will also make the look-up for a location into a zone and it will offer map utility functions.

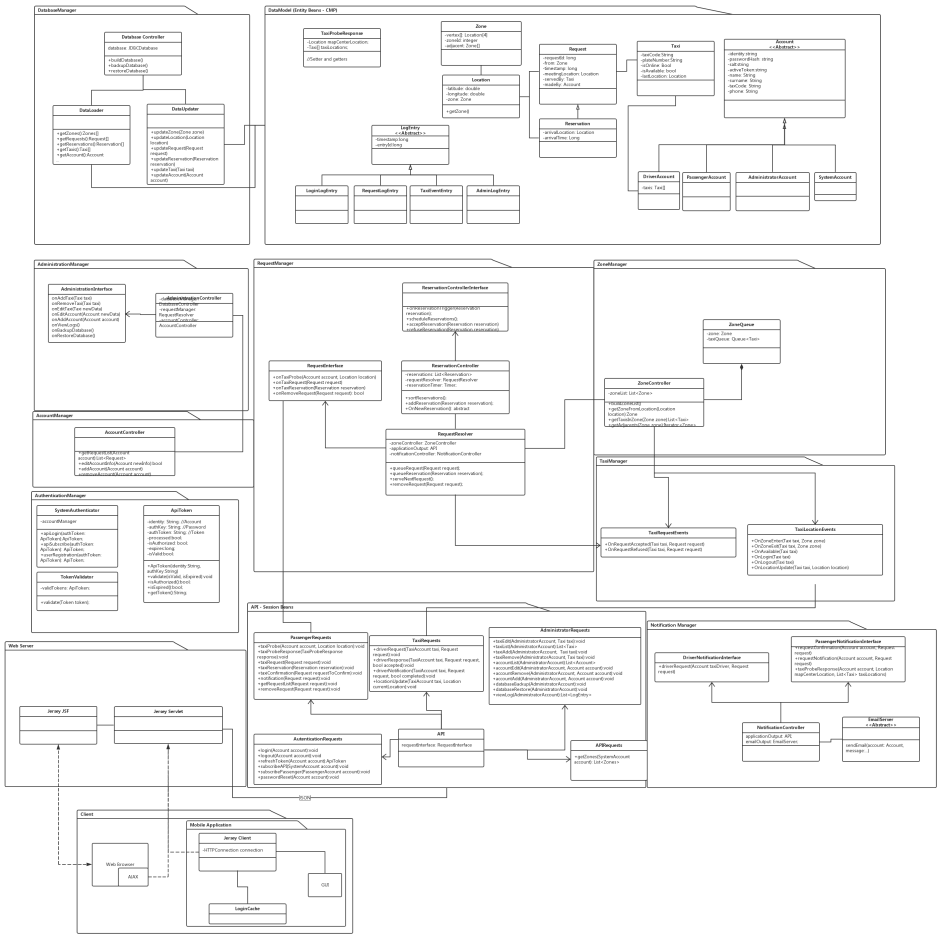
(h) **Log Manager** This component will manage all the system logs, both debugging and business inspection.

5. **Third-Party Services**

(a) **DNS** This service will offer Domain Name Lookup for accessing the various subsystems.

(b) **Email Server** This service will allow the unidirectional communication between MyTaxiService and the users.

A more in-depth view of the system is presented in this diagram:

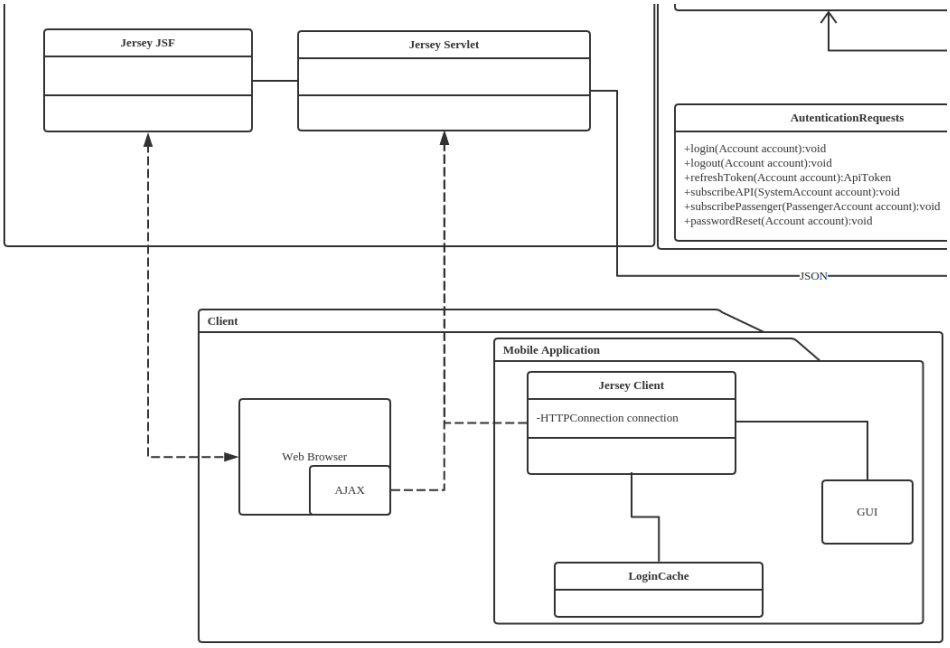


2.3 Component View

In this section a more detailed description of each component will be presented

2.3.1 Client Application

This is the client application for the passenger or the taxi manager. It consists on the following main components:



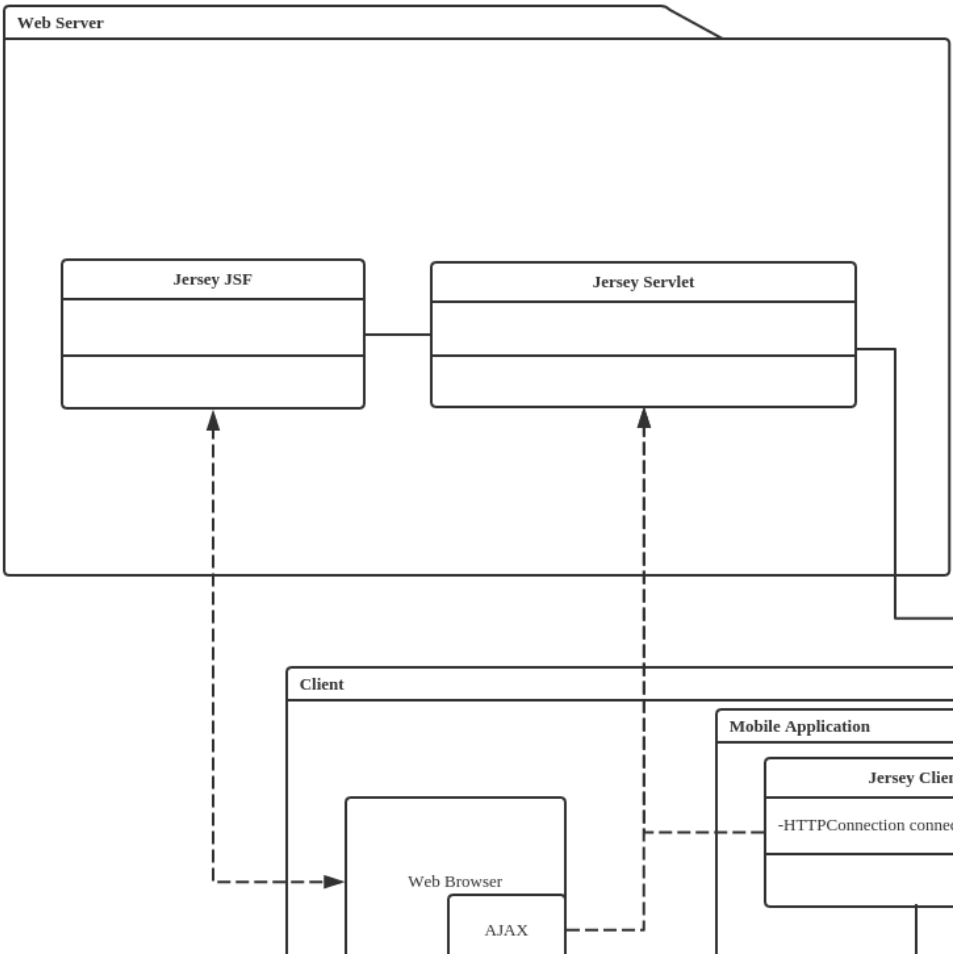
GUI It is the graphical user interface as described in the "User Interface Design" section.

RequestBuilder : It is the class that takes inputs from the GUI and builds requests for the server or, vice-versa, receives the messages from the server and display them on the GUI

LoginCache : This class will temporary store the user login information and the authorization token, for avoiding to request login data to the user as much as possible. For the implementation it will possible to rely on specific OS functions (as those offered by Android).

Client Browser This is the component for accessing the web interface.

2.3.2 Web Server



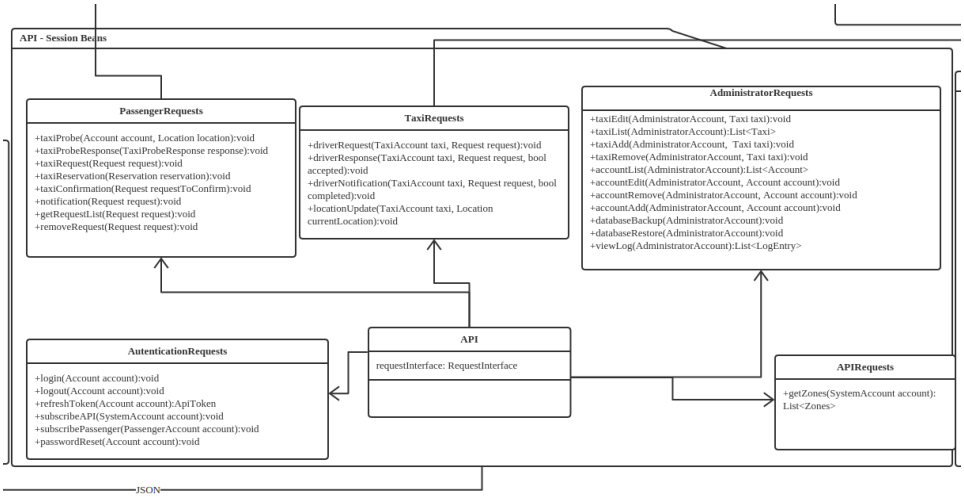
This component hosts the website along with some mechanisms for managing sessions and secure communications to the server APIs. Other than hosting the website, the purpose of this component is to avoid unauthorized requests to the server, and requesting the users a new login in the case their auth token is invalid or expired.
The authorization token is a key that is generated by the application server after a successful login and it has an expiration date, after which the client will have to login again in order to make new requests. The client must provide an active auth token along with every request it makes in order to be correctly identified by the system. This will apply both for users and systems that accesses the APIs, for further information please check the sequence diagrams section. The web browser will make use of Jersey, which is a reference implementation of JAX-RS for RESTful systems. The communications will occour using JSON, for which a jersey plugin is available.

Jersey JSF This component generates the web pages and provides a web

interface for the users.

Jersey Servlet This component manages all the incoming and outcoming requests from and to the application APIs. It both generate contents for the AJAX calls from the web browser and the mobile application.

2.3.3 API



This component is the main input/output interface of the application server and its main purpose is that to route request and responses to the right component of the application. Another important aspect that the implementation must cover is the masking of responses, that is the removal of sensible business data from the responses: i.e. when a user requests for the available taxi, he is supposed to recevive only the taxi code and the taxi location and not all the data which belongs to the Taxi class. Another example can be the masking of authentication data for a system administrator who requests the list of the accounts. /newline This class will also offer a programmatic access for system developers.

API This is the main API class and will implement serveral interfaces explained below. In the case a request is incoming it will translate it into Java messages and call a proper function of the application server. Otherwise, if a request is outcoming of the application server it will build the message for the client.

AuthenticationRequests This interface contains all the methods useful for authenticating a system or a user.

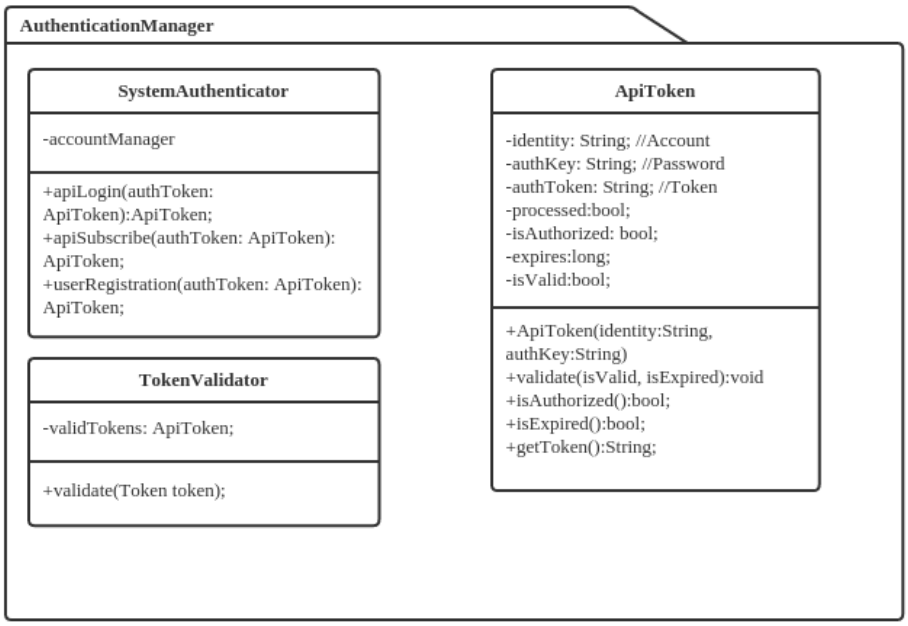
PassengerRequests This interface contains all the methods useful for managing the requests from/to the passengers.

TaxiRequests This interface contains all the methods useful for communicating with the taxi driver application

AdministrationRequests This interface contains all the methods useful for system administrators

APIRequests This interface will contain all the methods useful specifically for the system developers.

2.3.4 Authentication Manager



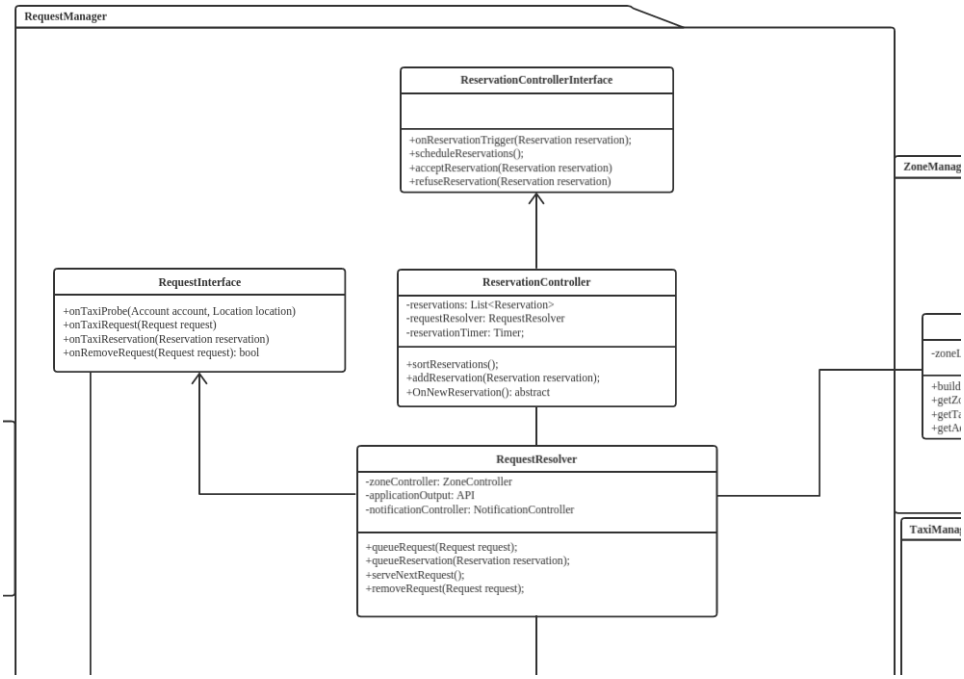
This component will check if a system or user is making a legitimate login request, and eventually generate and assign a new AuthToken.

SystemAuthenticator This class offers the main methods for logging in or registering a new user or a new dependant system.

ApiToken This class represents ad authorization token which is generated or fetched from the database. It will also offer methods for checking the token validity and permissions to a user.

TokenValidator This class offers functionalities for checking if a request is legitimate given an Auth token.

2.3.5 Request Manager



This component hosts the core of the application, that is the management of taxi, zones and requests.

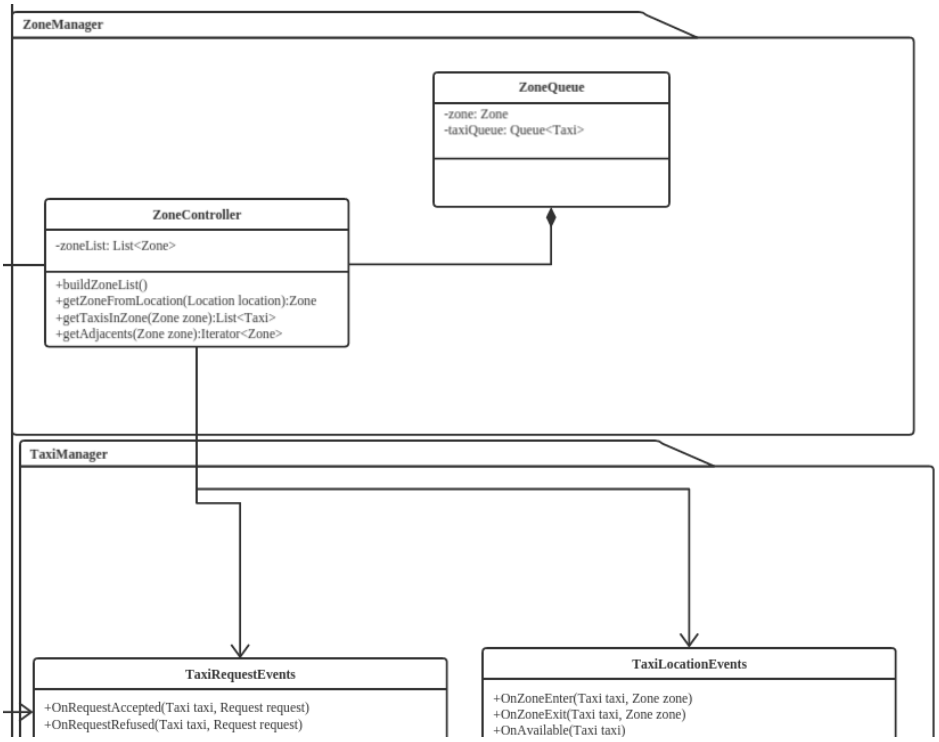
RequestResolver This class implements the core algorithm for managing incoming requests. It will run the core algorithm for managing an assigning taxi to requests

ReservationController This class will trigger pending reservations that are stored in the database at the right time. When a reservation has to be triggered, it will be pushed in the request queue in order to be processed.

RequestInterface This interface contains methods for managing the incoming requests from the users, such as creation and deletion of requests and taxi availability requests.

ReservationControllerInterface This interface contains all the methods that the reservation controller has to implement in order to manage reservations.

2.3.6 Zone Manger

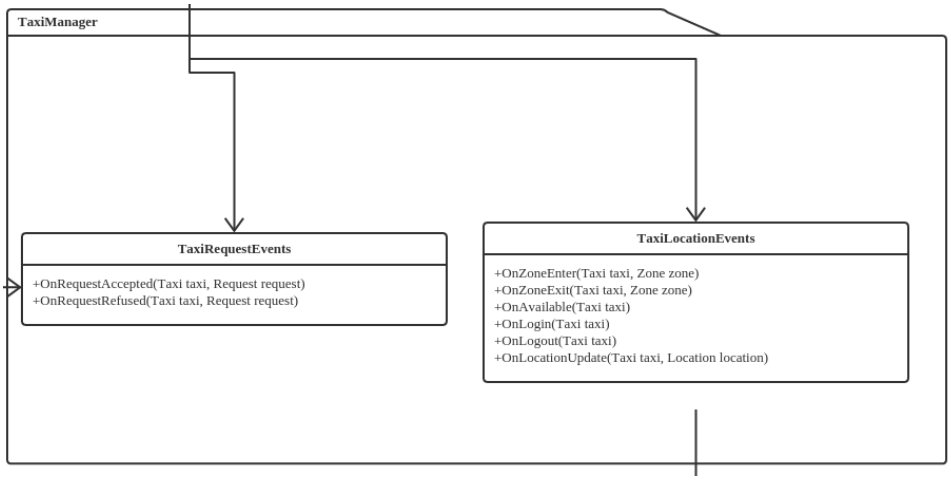


This component is a container for the definition of the zones in which the city is subdivided.

ZoneController This class will contain all the zones specified for the city and will implement functionalities for managing the tracking of taxis and accessing their state and location

ZoneQueue This is the actual taxi queue associated to a zone

2.3.7 Taxi Manager

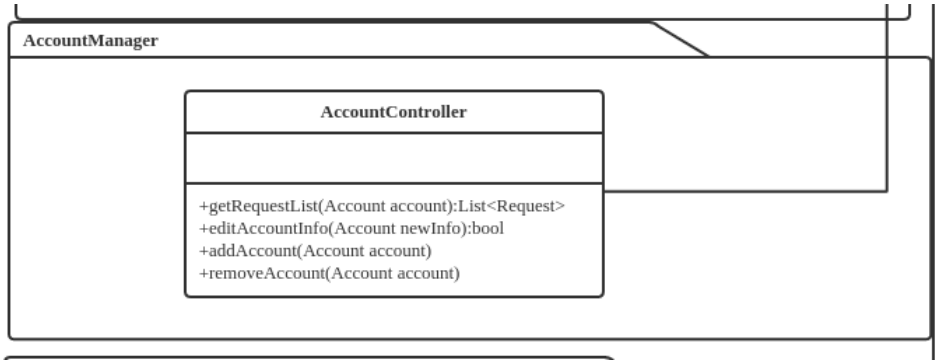


This component is a set of interfaces that will have to be implemented in order to manage the taxis

TaxiRequestEvents This interface contains methods for managing the acceptance or refusal of requests made by the taxi drivers.

TaxiLocationEvents This interface contains all the events that belongs to a taxi object, such as login/logout, moving from a zone to another, location updates and availability state changes.

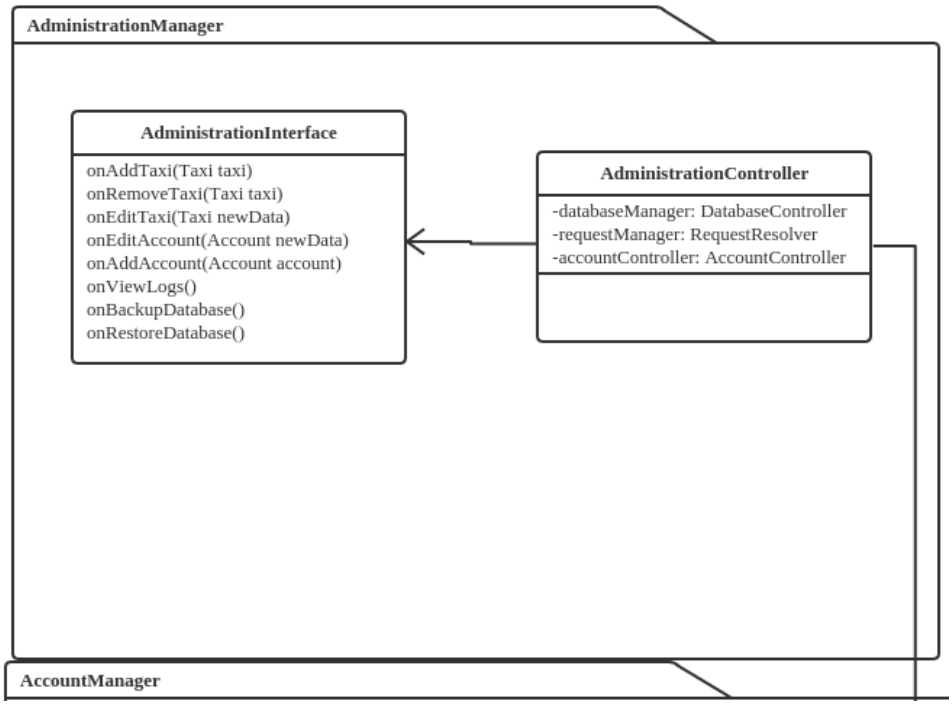
2.3.8 Account Manager



This component provides functionalities for managing and accessing stored account data

AccountController This class will provide functionalities for creating, accessing, deleting or modifying a user or system account

2.3.9 Administration Manager

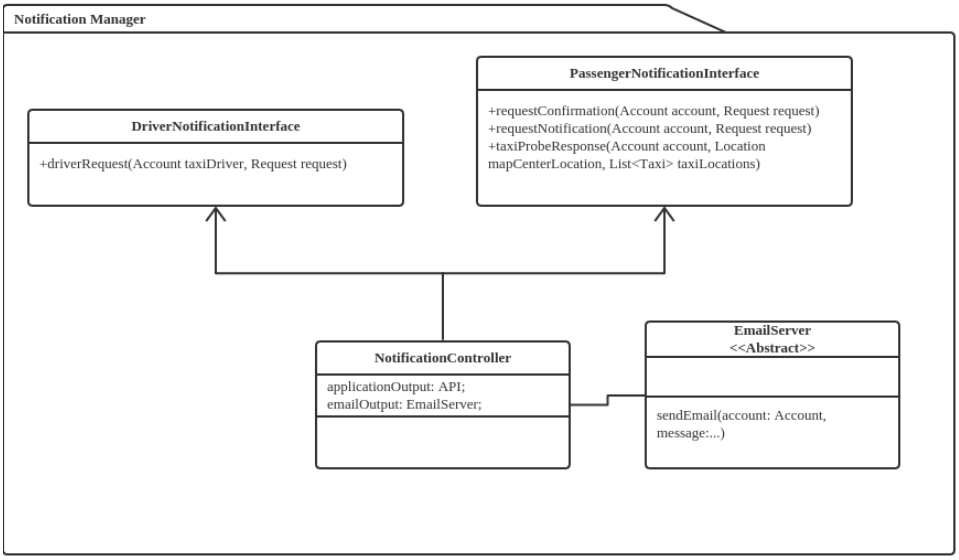


This component will offer an interface for the system administrators

AdministrationController This class will offer access to all the components that the system administrator can interact to

AdministrationInterface This interface contains all the methods for the system administrator, such taxi management, account management, log access, database manitenance

2.3.10 Notification Manager



This component will manage all outgoing notifications for the users

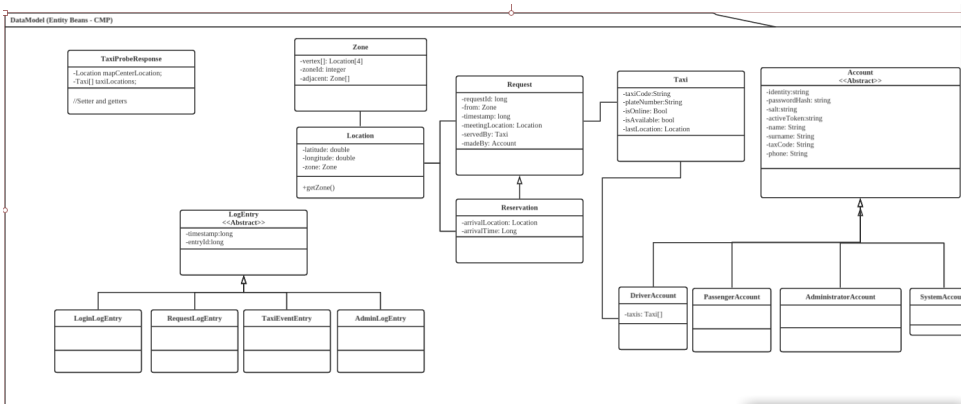
NotificationController This is the controller for managing all the application output.

DriverNotificationInterface This interface contains methods for issuing notifications to the driver application

PassengerNotificationInterface This interface contains methods for issuing notifications to the passenger application

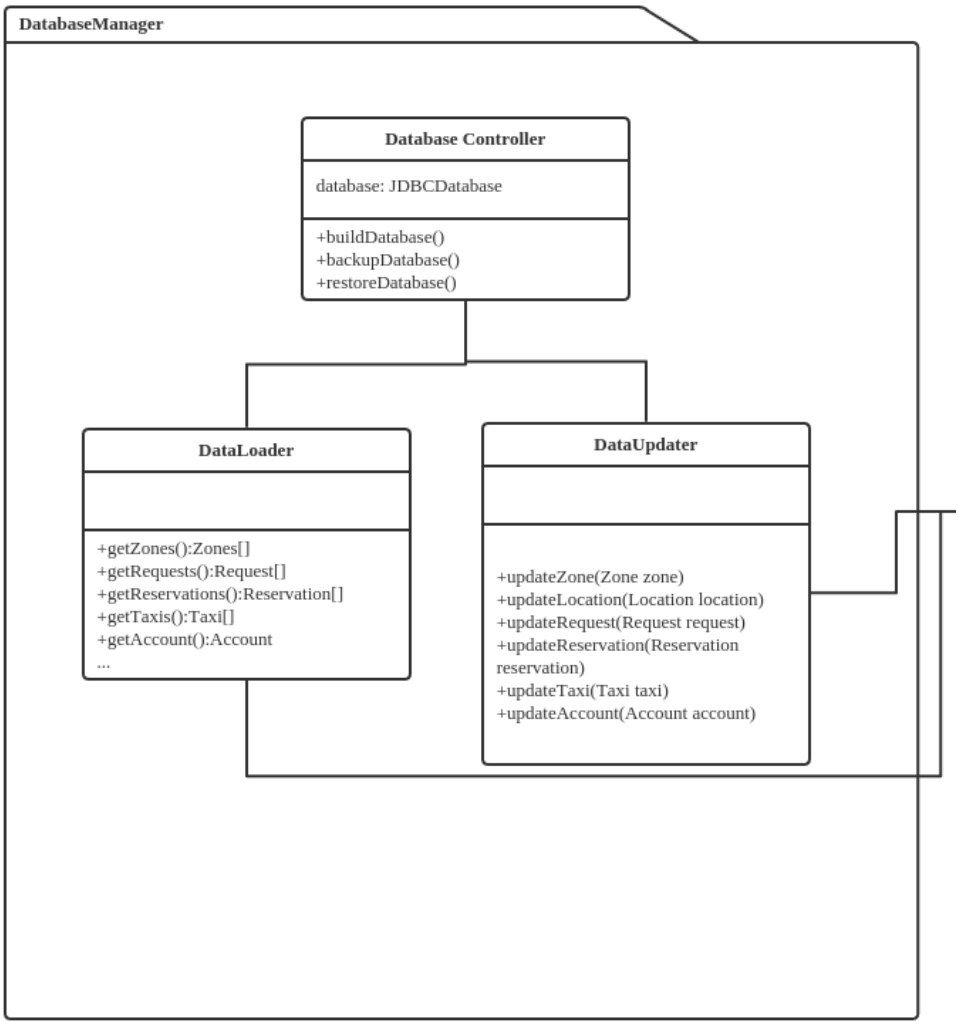
EmailServer This is an abstract object representing the email service in use with this application. The implementation will depend on further design choices.

2.3.11 Data Model



This component is the actual data model of this application. The classes specified in this component will reflect the database data schema. It will also contain some support data models such as the TaxiProbeResponse, which are not meant to be stored on the server but only generated and issued to the client application.

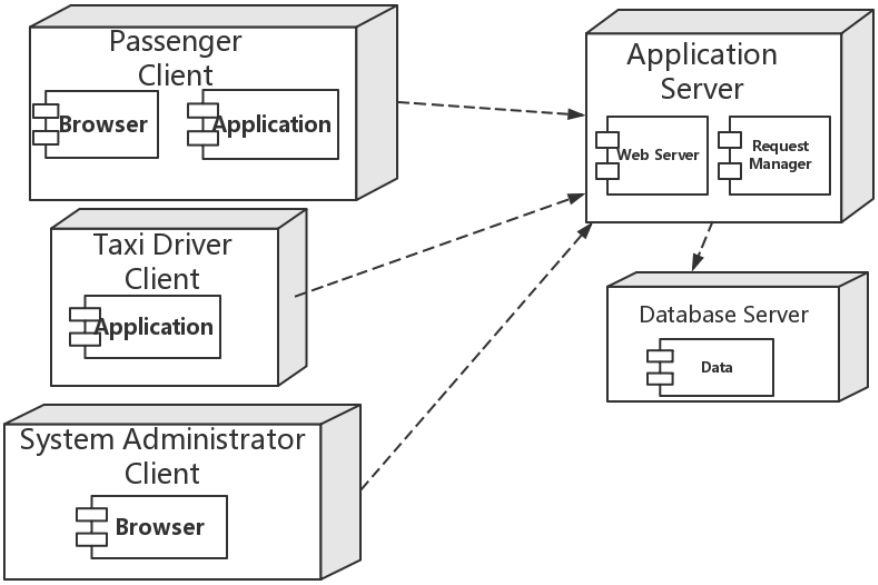
2.3.12 Database Manager



This component will manage the data access and data persistance. The implementation will be compliant to the J2EE specifications for data persistance.

2.4 Deployment View

The system is meant to be deployed in a 3-tier architecture, which consist in serveral clients (both web and mobile applications), a web-server hosted on the same machine of the application server and a database hosted on a dedicated machine.



2.5 Runtime View

2.5.1 BCE

For describe the BCE we used a Class Diagram with appropriate stereotypes <<boundary>>, <<entity>>and <<control>>. This is the function of the stereotypes:

1. boundary : represents the interface between users and system;
2. entity : represents a class that allow to use the data in a database;
3. control : controls the choise of the user.

2.5.2 BCE Passenger

It is divided in part:

1. **Login**
   (a) **CheckRegistration** This controller manages the insert of new user; it controls if the mandatory fields are complete and correctly. If the fields are complete it stores the date in the database and creates a new user.
   (b) **CheckLoginInformation** This controller manages the verification of login; it examines if the fields are filled correctly, it examines the password and the email. If the fields are correctly, the controller can load the correspondent home page.

(c) **ResetPassword** this controller manages the send of new password with email to a user if he requires a new password. It modify the user information in the database with a new password that it has generated.

2. **ControlTaxi**

   (a) **getPassengerGPSLocation** This controller manages the position of the passenger and it selects the taxi that it is close to the passenger position;

   (b) **getAddressAndZone** This controller manages the address and the zone where the passenger is.

   (c) **checkAvailabilityTaxi** This controller manages the taxi availability, it checks the taxi position and the passenger position for reserch the best solution. It shows to the user the taxi available.

   (d) **updateAvailabilityTaxi** This controller updates the taxi position and availablility, for a example when a taxi becomes free.

   (e) **updateMap** This controller update the position of the taxi on the map.

3. **ControlChoice**

   (a) **taxiReservation** This controller manages the passenger reservation, it controls if the information that the passenger inserts are correctly, it controls if the hours is possible, if the street exist; after it controls that the reservation is done 2 hours before the meeting time.

   (b) **taxiRequest** This controller manages the passenger request, it stores the request and send it to the first free taxi in the queue;

   (c) **updateAccount** This controller updates the passenger information and checks if the information that the passenger inserts are correctly.

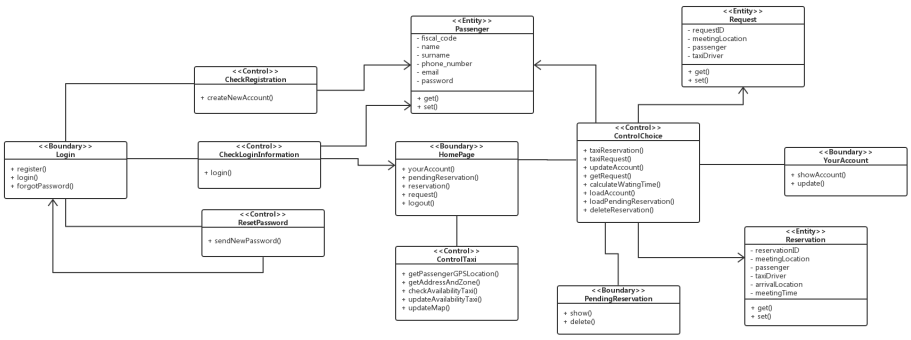
   (d) **getRequest** This controller informs the passenger about his request.

   (e) **calculateWaitingTime** This controller manages the waiting time that the passenger must wait for a taxi.

   (f) **loadAccount** This controller manages the passenger account; it shows to the passenger the information that he requires.

   (g) **loadPendingReservation** This controller manages the passenger reservation and request; it shows the reservation or the request that the passenger will see.

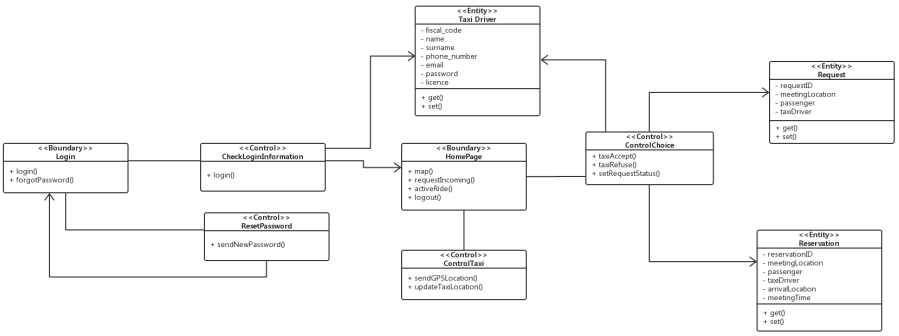
(h) **deleteReservation** This controller deletes the reservation that the passenger decides to remove; it deletes the reservation from the queue of reservations.



2.5.3 BCE Taxi Driver

It is divided in part:

1. **Login**
   (a) **CheckLoginInformation**This controller manages the verification of login; it examines if the fields are filled correctly, it examines the password and the email. If the fields are correctly, the controller can load the correspondent home page.
   (b) **ResetPassword**This controller manages the send of new password with email to a user if he requires a new password. It modify the user information in the database with a new password that it has generated.
2. **ControlTaxi**
   (a) **sendGPSLocation**This controller manages the GPS position from the taxi and it uses this position for define if the taxi can reply to a request.
   (b) **updateTaxiLocation**This controller updates the position of the taxi on the map.
3. **ControlChoice**
   (a) **taxiAccept**This controller manages the reply of the taxi driver where the reply is true; it sends a reply to the passenger about the taxi that will take him and the waiting time.
   (b) **taxiRefuse**This controller manages the reply of the taxi driver where the reply is false; it sends a request to another taxi driver;
   (c) **setRequestStatus**This controller manages the request; if the request is accepted, it delete the request from the queue of request and updates its status from free to accepted and deletes taxi from the queue of free taxi; if the request isnt accepted, it insert taxi in the queue of free taxi.



2.5.4 BCE System Administrator

It is divided in part:

1. **Login**

   (a) **CheckLoginInformation**This controller manages the verification of login; it examines if the fields are filled correctly, it examines the password and the email. If the fields are correctly, the controller can load the correspondent home page.

   (b) **ResetPassword**This controller manages the send of new password with email to a user if he requires a new password. It modify the user information in the database with a new password that it has generated.

   2)ControlTaxi :

   (a) **taxiManagement**This controller manages the taxi; it can delete, add and update the information about the taxi that the application uses. If the information insert after a update o add are wrong this controller must show what is wrong.

   (b) **accountManagement**This controller manages the account of the user, it can add, delete or update the information. If the information insert after a update o add are wrong this controller must show what is wrong.

   (c) **logs**This controller manages all information about login, request, reservation and the taxi ride.

   (d) **zone**This controller manages the different zone in the city; it can add, delete or update a zone. This controller must report if a zone is not corret or already exist.

   (e) **backup**This controller stores all information about the system; This controller must report if a backup doesnt come to a successful conclusion.

   (f) **restore**This controller restores the information stored with a previous backup. This controller must report if a restore doesnt come to a successful conclusion.

2. **ControlChoice**

   (a) **updateUserAccount**This controller manages the account of the user, it can update the information about the user.

   (b) **deleteUserAccount**This controller manages the account of the user, it can delete the information about the user or can delete the user account if the user does this request.

(c) **addUserAccount**This controller manages the account of the user, it can add the information about the user or can add new user account.

(d) **resetPasswordUser**This controller manages the send of new password with email to a user if he requires a new password. It modify the user information in the database with a new password that it has generated.

(e) **viewLogs**This controller manages all information about login, request, reservation and taxi ride and it shows the information request.

(f) **downloadLogs**This controller manages downloads of information about login, request, reservation and taxi ride.

(g) **saveData**This controller saves all data for a backup.

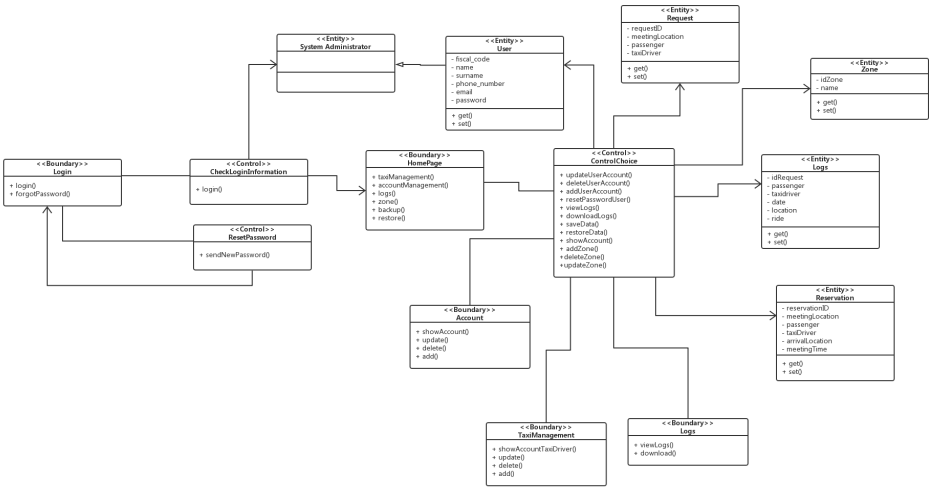
(h) **restoreData**This controller extract data from a previous backup and restart them.

(i) **showAccount**This controller manages all information about user and it shows the information request.

(j) **addZone**This controller manages the different zones of the city; it allows to add new zones in the city.

(k) **deleteZone**This controller manages the different zones of the city; it allows to delete a zone from a list of zone of the city.

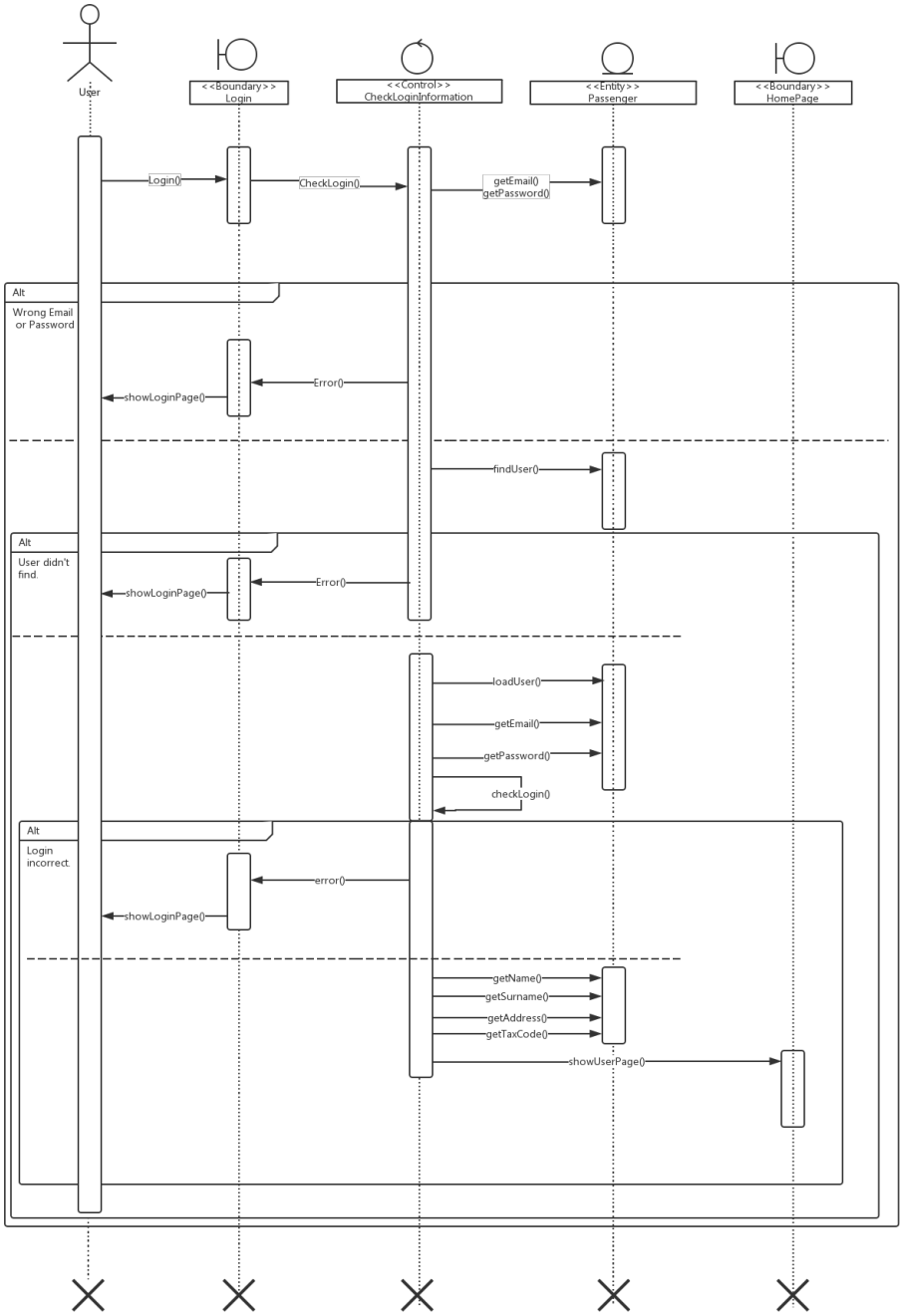
(l) **updateZone**This controller manages the different zones of the city; it allows to update the information of a zone of the city.



This diagram describe in detail that is show in the BCE diagram.

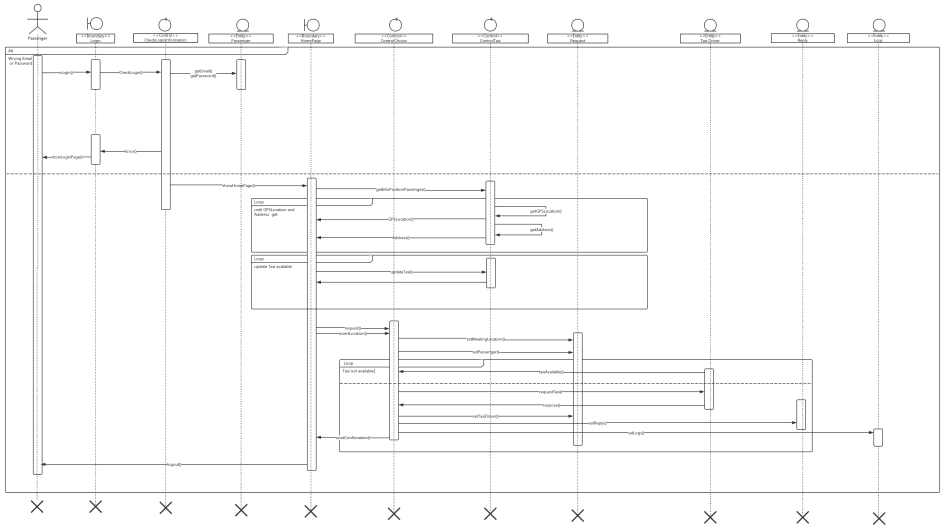
2.5.5 Sequence Diagram Login

Login can be done by all authorized User: * Passenger * Taxi Driver * System Administrator


User
<<Boundary>>
Login
<<Control>>
CheckLoginInformation
<<Entity>>
Passenger
<<Boundary>>
HomePage
Login()
CheckLogin()
getEmail()
getPassword()
Alt
Wrong Email
or Password
Error()
showLoginPage()
findUser()
Alt
User didn't
find.
Error()
showLoginPage()
loadUser()
getEmail()
getPassword()
checkLogin()
Alt
Login
incorrect.
error()
showLoginPage()
getName()
getSurname()
getAddress()
getTaxCode()
showUserPage()

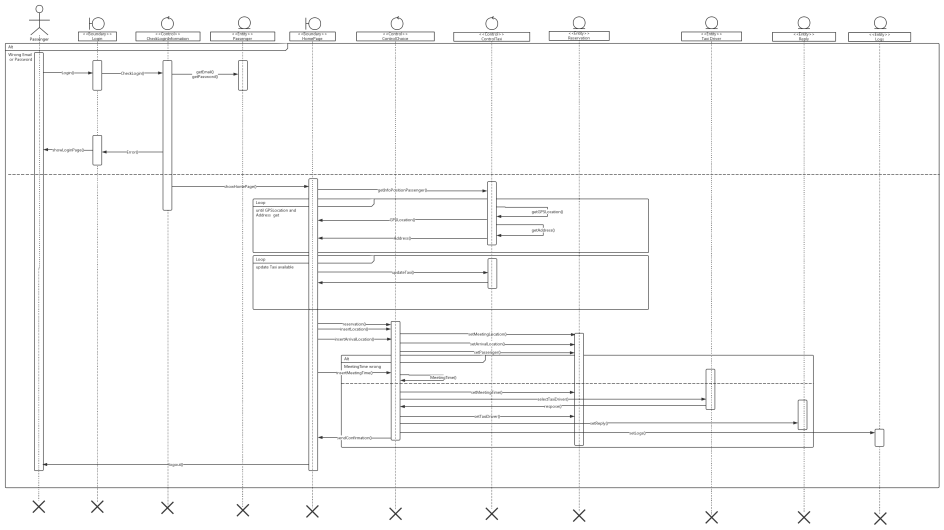
2.5.6 Sequence Diagram Request

Request can be done by all authorized Passenger.



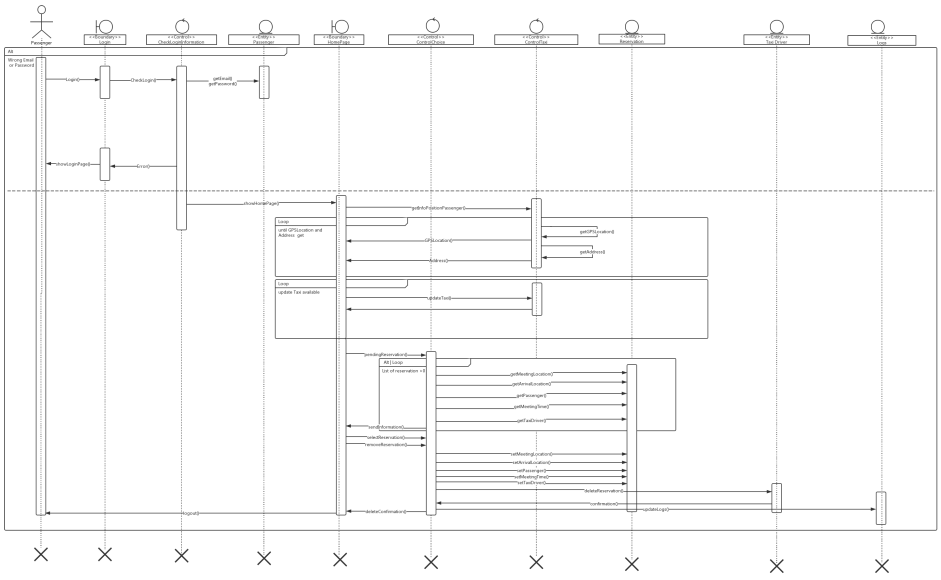
2.5.7 Sequence Diagram Reservation

Reservation can be done by all authorized Passenger.



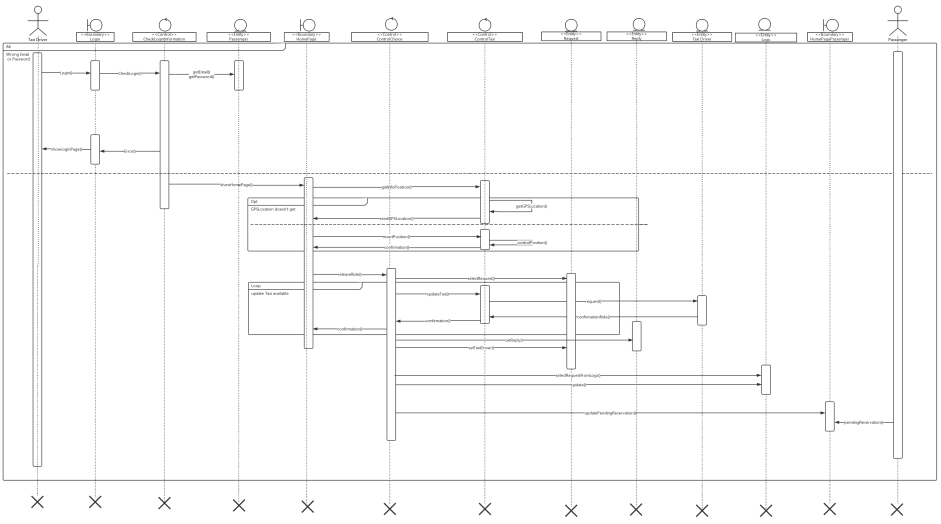
2.5.8 Sequence Diagram Reservation Delete

Delete a reservation can be done by all authorized Passenger.



2.5.9 Sequence Diagram Taxi Request Release

Release a request can be done by a Taxi Driver that can go to the location of the request that he received.



2.6 Component Interfaces

2.6.1 User Experience

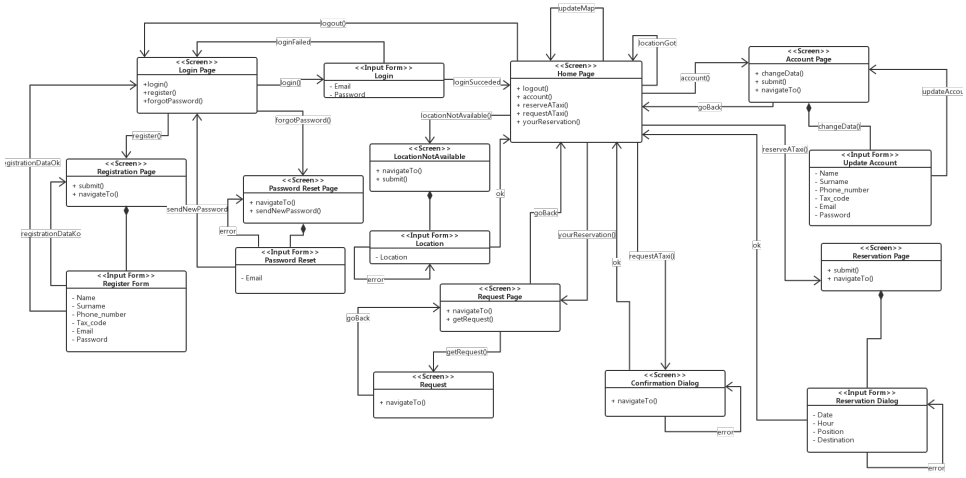
For describing the User Experience (UX) we used a Class Diagram with appropriate stereotypes <<screen>>and <<input form>>While <<screen>>represents

pages, <<input form>>represents input fields that can be complete with by user with the information that the form require.

2.6.2 UXPassenger

We can see in the Diagram the possible action that the passenger can do when he uses the application. This the most important pages:

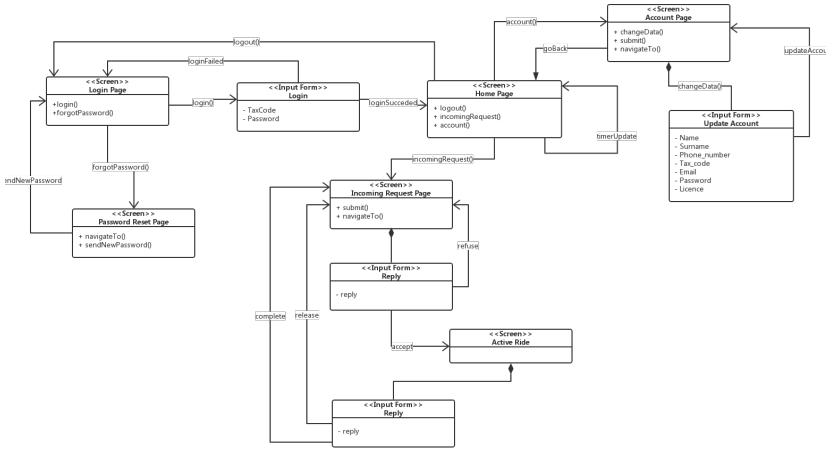
1. **LoginPage** It is the first page that the user see when start the application; in this page the user can do:
   (a) **register** User can insert his date and so can register to the system and can use the application;
   (b) **login** User can insert password and email to enter in his private page;
   (c) **forgotPassword** User can request a new password because he forgot his.
2. **HomePage** It is the private page where the user can do request and reservation; in this page the user can do :
   (a) **account** The user can see his private information and can modify them;
   (b) **reserveATaxi** The user can compile a form with the information about time, location and destination of the ride request;
   (c) **requestATaxi** The user can see the time of wait for a taxi and if his request is accept or no;
   (d) **yourReservation** The user can see all his request and reservation that he do with the application;
   (e) **logout** The user can exit from the application.



2.6.3 UXTaxiDriver

We can see in the Diagram the possible action that the taxi driver can do when he uses the application. This the most important pages:

1. **LoginPage** It is the first page that the taxi driver see when start the application; in this page the user can do:
   (a) **forgotPassword** Taxi driver can request a new password because he forgot his;
   (b) **login** Taxi driver can insert password and taxCode to enter in the application.
2. **HomePage** It is the private page where the taxi driver can accept or refuse the request that arrive from the passenger; in this page the taxi driver can do :
   (a) **logout** The taxi driver can exit from the application;
   (b) **incomingRequest** The taxi driver see the request and he decides to accept or refuse it, if he accepts the request he will see the screen with the information or the ride.
   (c) **account** The taxi driver can see or modify your private information.



2.7 Architectural Styles and Patterns

The following design patterns have driven the design process of this project:

- MVC: Model-View-Controller design pattern. This pattern separates the business data, the user interface (or the interface between systems) and the core modules that runs the business logic.
- Thin-Client: The application client is used only for the data presentation and input, therefore it will contain the least business logic possible.

2.8 Other Design Decisions

3 Algorithm Design

4 User Interfce Design

5 Requirements Traceability

6 References

- MyTaxiService Requirement Analysis and Specification Document

7 Tools and Document Information