Politecnico di Milano

A.A. 2015-2016

Software Engineering 2: "myTaxiService"

Design Document

version 1.0

Adrian Mihai Berbieru #854698

Attilio D'Onofrio #789614

04/12/2015

1 Introduction

1.1 Purpose

This document represents the Design Document of myTaxiService project. The goal of this document is twofold: to present the architectural choices in our solution with the associated tradeoffs, then to describe the system components and the way they interact each other.
This document is intended to all developer and programmer who have to implement the requirements, to system analysts who want to integrate other systems with this one, and could be used as contractual basis between the customer and the developer.

1.2 Scope

The architectural description provided concerns the functional view, module view, deployment view, data layer, business logic and the user interface.
Hence myTaxiService architecture will provide five different functionality areas:

- Users profiles: it will manage personal data and status of the different types of users. Users can be customers or taxi drivers.
- Calls: it will manage the calls customers are able to do (both long term and instant booking).
- Users login and registration: it will manage registering, logging in/out of users.
- Taxi management: it will allow drivers to manage the received calls, but also the management of taxis and queues by the system.

1.3 Definitions

- Visitor: also called unregistered user, is a person who is not registered to the system.
- Customer: also called user or client, is the person who books the ride via the web or smartphone app.
- City zone: the part of the city the taxi is assigned to. Each zone has an area of about $2km^2$.
- Availability: referred to a taxi, it says if the driver is able to take calls or not.
- Queue: the list of taxis associated to a city zone.
- Code: the identifier associated to a taxi.
- Booking: also called reservation, the act of asking for a taxi, done by a customer through the system. It may be instant or long term.
- Instant booking: also called instant reservation, it is done when a customer requests a taxi, sending his current position.
- Long term booking: also called long term reservation, it is done when a customer makes a reservation for a taxi in a specific time in the near future.

- User: whomever interacts with the system; the terms is used referring to customers and taxi drivers alike.
- Request: the message sent between client and server or between server side components.
- Active: referred to a device, it is active if it is turned on and myTaxiService application is running.

1.4 Acronyms and abbreviations

- **Gn:** n-goal.
- **DBMS:** database management system.
- **AS:** application server.

1.5 Reference documents

- **2012_project_DesignDocument_ENG.pdf** (Software Engineering 2 project of AY 2013-2014, it can be found on BeeP).

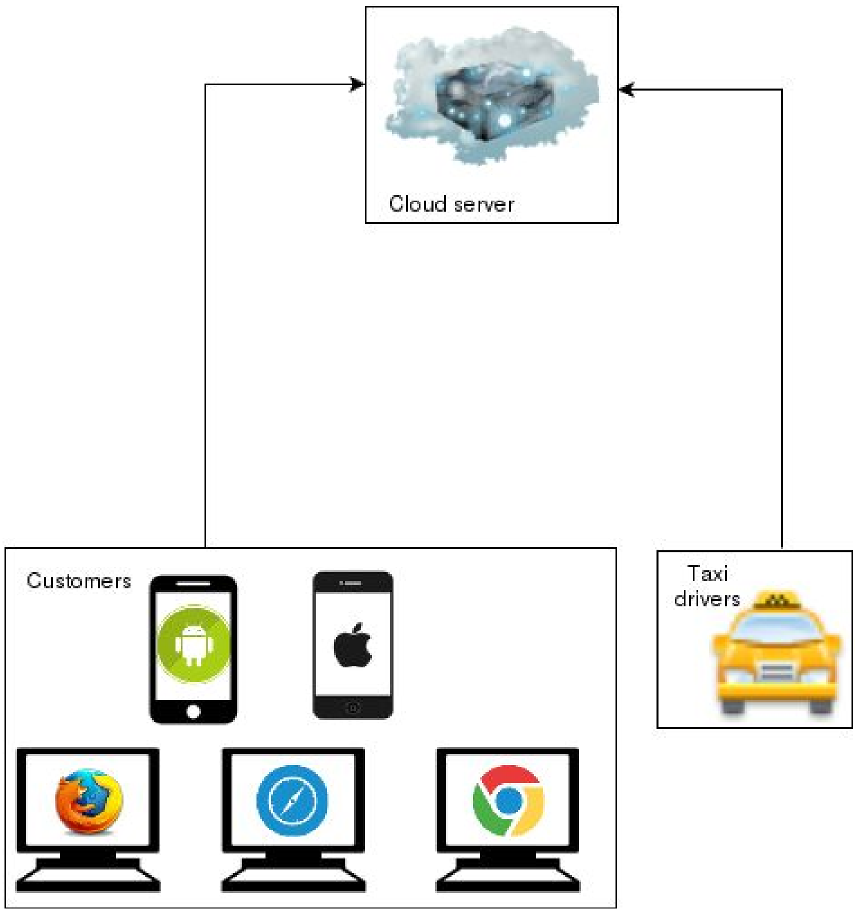
1.6 Document structure

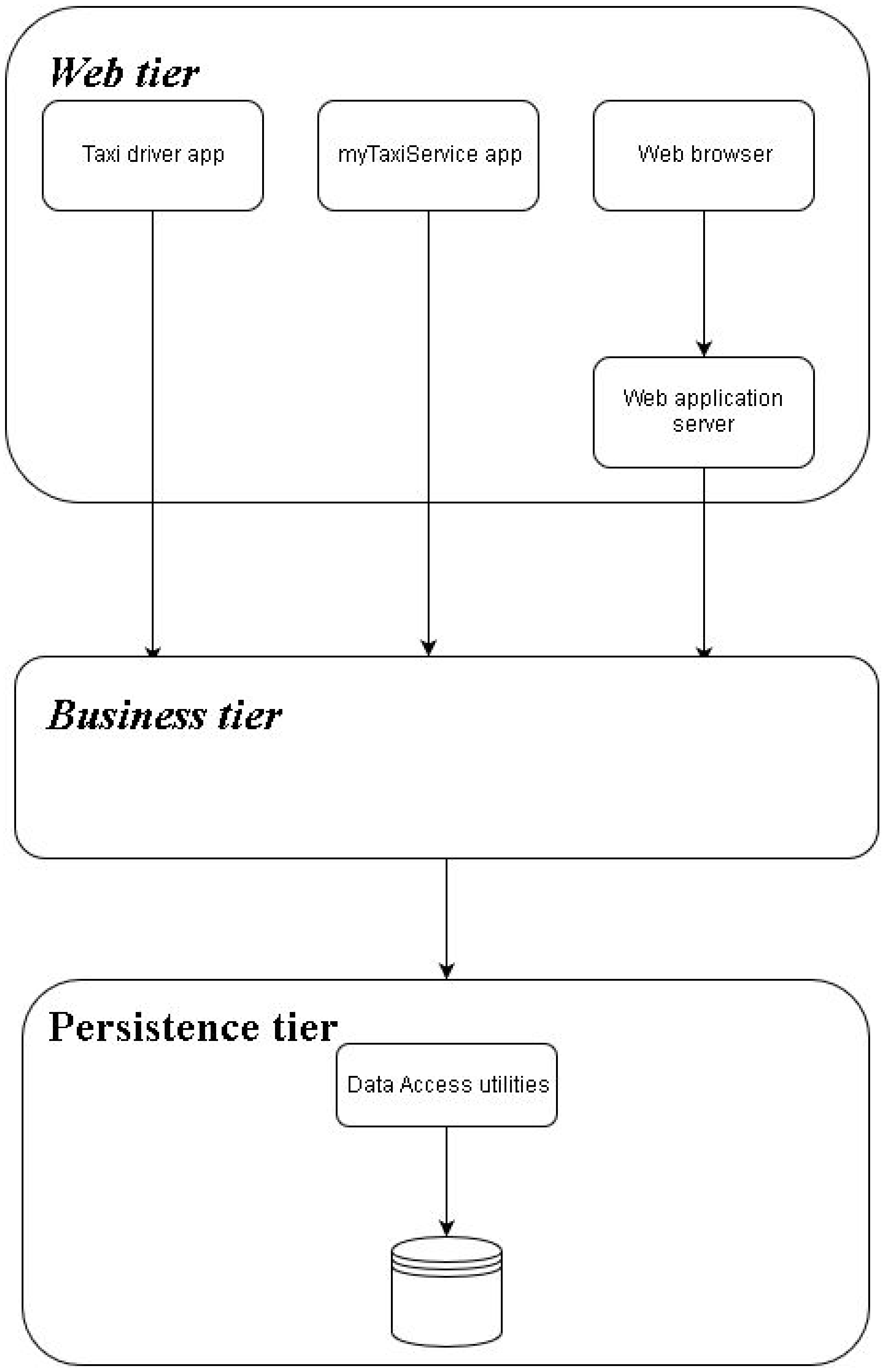
1 Introduction a presentation of the aim of the document, including the jargon used and its structure.
2 Architectural Design an initial general presentation of how the system system is design, followed by a detailed analysis of the different component the system is made up and how they communicate with each other including the technologies used to deploy the system.
3 Algorithm design a detailed explanation in natural language on the algorithms used by the system to provide functionalities.
4 User Interface Design contains UX diagrams representing "at a glance" the mockups in the RASD.
5 Requirements Traceability a mapping between the goals of the RASD and the components of the system.
6 Appendix the hours of work and the tools used to create this document.

2 Architectural Design

2.1 Overview

Our solution will use the well-known client-server architecture, since it is the simplest one able to manage in an effective way requests made by users.
We expect to have two physical tiers: the client tier and the server tier.
In the server tier there will be different machines, each one with an assigned specific task, so to have a well-balanced workload. Communications overhead among machines will be also limited.
The client side includes the customers, the taxi drivers and the call center operator. The last ones will not be taken into account in this document since their interaction is quite similar to customers'.





But before going into further detail, we want to group the functional requirements listed in the RASD according to the areas presented in the paragraph 1.2 of this document.

Users profiles

- **[G1]** To allow the visitor to register.
- **[G2]** To allow the unidentified user to log in.
- **[G6]** To allow the taxi driver to change its availability status.

Calls

- **[G3]** To allow the registered user to make an instant booking.
- **[G4]** To allow the registered user to make a reservation in a specified time.
- **[G5]** To allow the registered user to delete the reservations he made.

Users login and registration

- **[G1]** To allow the visitor to register.
- **[G2]** To allow the unidentified user to log in.
- **[G9]** To allow the taxi driver to login.

Taxi management

- **[G7]** To allow the taxi driver to receive a call.
- **[G8]** To allow the taxi driver to get GPS directions until he reaches client's position.

2.1.1 System technologies

myTaxiService will be designed considering the client-server 3-tier distributed architectural style. Each tier will take advantage of the following technologies.

Web Tier

In this tier we have all the end users' devices that is:

- personal computer where the user can make requests through a browser which in turn sends the requests to a web server on which the Web application is run,
- smartphones where the myTaxiService app is run;
- the tablets where the Taxi driver app is run.

Dynamic web pages containing XHTML, generated by web components developed with Java Server Faces technology, an interface component framework part of the Java Enterprise Edition used to build web applications.

Business Logic Tier

In this tier we can find the core of the application which handles all requests coming from customers and management of the taxi queues. To be able to do such tasks it must rely on an external component the External Map Service which offer mapping services and geolocalization. Also in this tier we can find a Plugin management component through which developers can add new plugins in order to expand the features the system offers when needed.

The Java Enterprise Edition 7 (JEE7) platform supports applications providing enterprise services in the Java language, such as the Enterprise Java Beans (EJB), business components that capture the logic of a specific domain and are in charge to meet its needs.

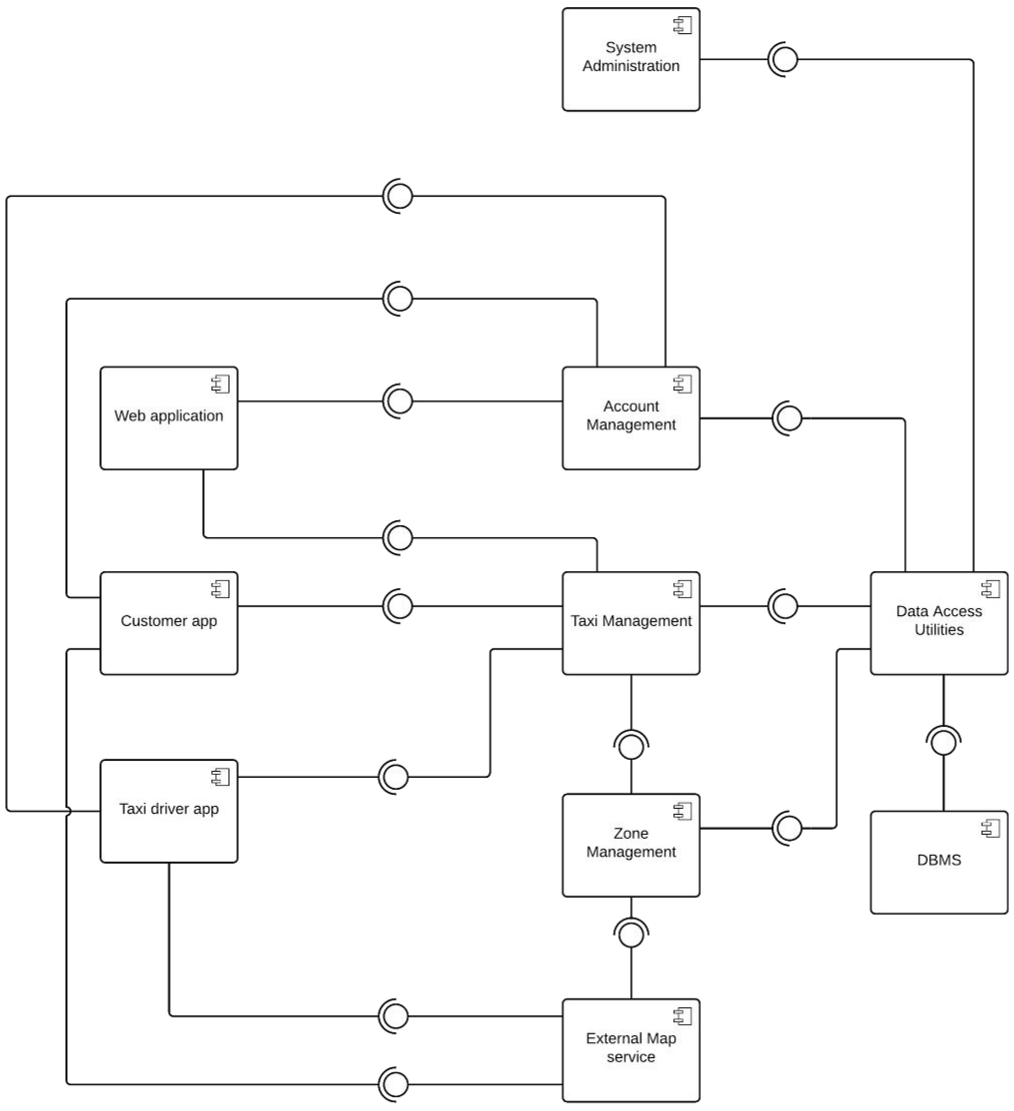
The AS chosen is GlassFish 4.1.1, providing security, transaction support, load balancing, management of distributed applications and supports the JEE7 platform.

Persistence Tier

In this tier we can find the DBMS (short for Data Base Management System) which supports the all the operations on stored data. The component will use MySQL Server 5.7, a widely used relational DBMS which support ACID distributed transactions and is runned in a DMZ zone for higher security. In this tier we can also find an intermediate Data Layer which performs queries arriving from the other components on the DBMS; this design permits a more flexible, custom and secure way in which the upper layers interact with the DBMS.

2.2 High Level Components and Their Interaction

In this section we will highlight the main components of our myTaxiService system and the relations between them. These components however are only abstract models which show the main functionalities of our system and how they interact with each other.



The Data Base Management System, shortened as **DBMS**, is a component in charge of saving and retrieving the system data in a persistent, secure and reliable way.

The **Data Access Utilities** component offers a specific interface to every other component that wants to store or retrieve data from the database. Thus it must also act as another mean of protection of the data from malicious attacks.

The **System Administration** component is responsible with some system configurations such as:

- insertion, update and deletion of vehicles and drivers;
- it also permits to redefine the boundaries and subdivisions of the city .

It will have a simple interface because its primary user will be a trained technician.

The **Account Management** component is responsible for all operations regarding user accounts:

- it has to permit visitor registration to the application;
- it has to permit to any registered user to login the application;
- it must verify email validity;
- it has to permit password retrieval in case the user has forgotten it;
- it must permit every employed taxi driver to login.

The **Taxi Management** is one of the most important component of the system because:

- it must accept all instant reservation from client and search for an available vehicle matching the ride requirements;
- it must permit the cancellation of an instant booking in the case there are no available taxis.
- it has to store all long term bookings for every registered user;
- it has to list all long term reservations made by the user;
- it has to permit the cancellation of a long term reservation;
- it has to find a suitable vehicle for every long term reservation 10 minutes prior to the scheduled departure;
- it must save all rides performed during the day as stated in the Taxi Regulation policy of the city;

The **Zone Management** is in charge of managing taxi queues:

- it has to generate a taxi queue for every zone of the city;
- it must assign a taxi to a queue by analyzing the vehicle location whenever it becomes available;
- it must constantly update the queue whenever a taxi becomes available/busy;
- it must shuffle the queue every time one of the drivers refuses a call.

The **External Map Service** is a component provided by a third party, it is used for getting the locations of clients and vehicles. It also must provide step by step navigation for taxi drivers in order for them to reach the destination.

The **Web application** and **Customer app** provide ways for clients ,after successfully registered and logged in their account, to enquiry services through a web browser in the first case and through a mobile device in the second one. The Customer app also provides autocompletion of the address the client wants to be picked up (this feature is offered by the External Map service), thus ensuring that all inputted addresses are correct.

The **Taxi driver app** provides a way for drivers to log in the application and be therefore notified about calls, accept/decline a call and change the availability status. Also this component offers to its users a step by step navigation assistance whenever they don’t know how to reach the client’s position (this feature is offered by the External Map service).

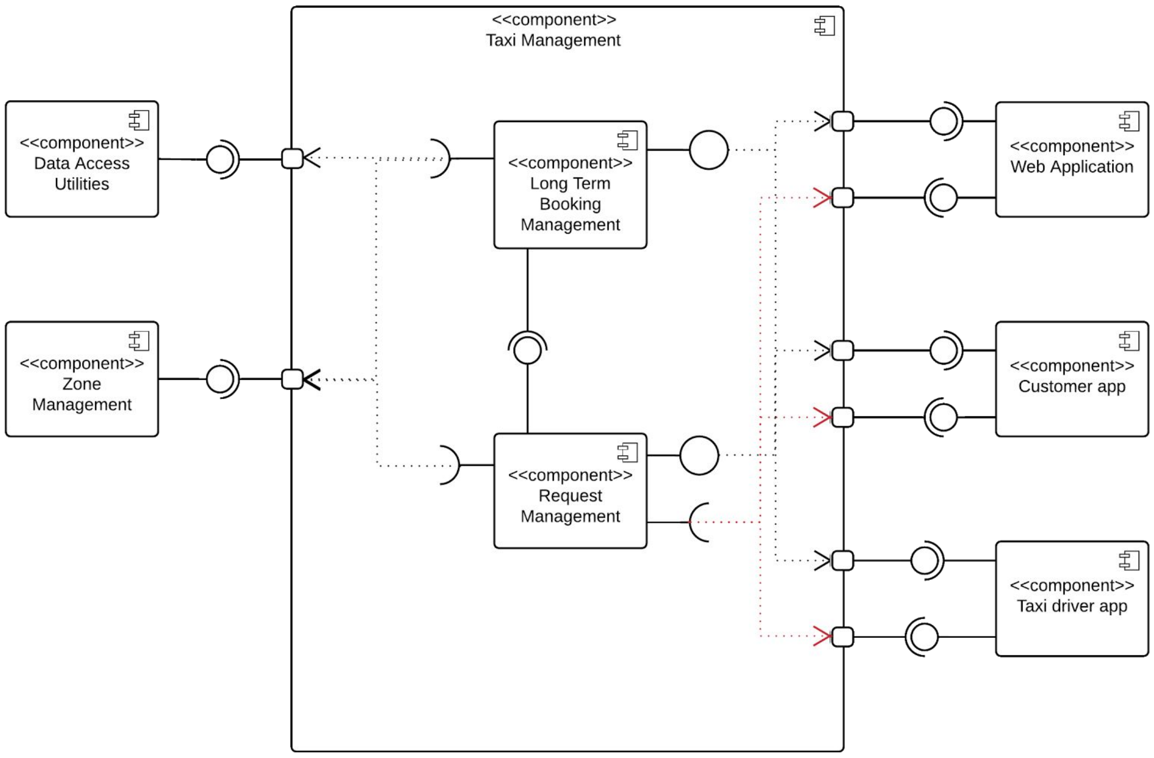
2.3 Component view

In this section we will analyze more in detail some of the main components of the system. It should be noted that we won’t go very in depth because we only want to offer a high level view of how the system is composed.

The first component we’re going to analyze is the Taxi Management which as stated beforehand manages calls from clients and dispatches a suitable vehicle. The Taxi Management is composed of subcomponents:

- The **Long Term Booking Management** subcomponent handles every long term reservation a user makes. Once it receives a request from a customer it stores it in the database and in a special local queue. At fixed intervals it checks the local queue in order to see if any of the reservations should be serviced and thus sends them to the Request Management subcomponent. It also has the duty to list all the long term reservations a user made and the possibility to delete a reservation if it is possible.

- The **Request Management** subcomponent handles every instant booking or long term that should be serviced. It sends the coordinates of the meeting point to the Location Management, after receiving the afferent zone it searches through the queue of that particular zone and sends a notification to the taxi driver. If the driver accepts then it sends a response to client containing the taxi code and estimated waiting time and it saves the ride details into the database, if not it repeats the procedure and asks the Update Management to move the driver to the end of the queue. It must also handle the case where there are no available taxis in the zone where to pickup the client, thus the component must save the ride details in a special queue until the request can be serviced if in the meantime it receives a cancellation request it simply deletes the request from the queue.

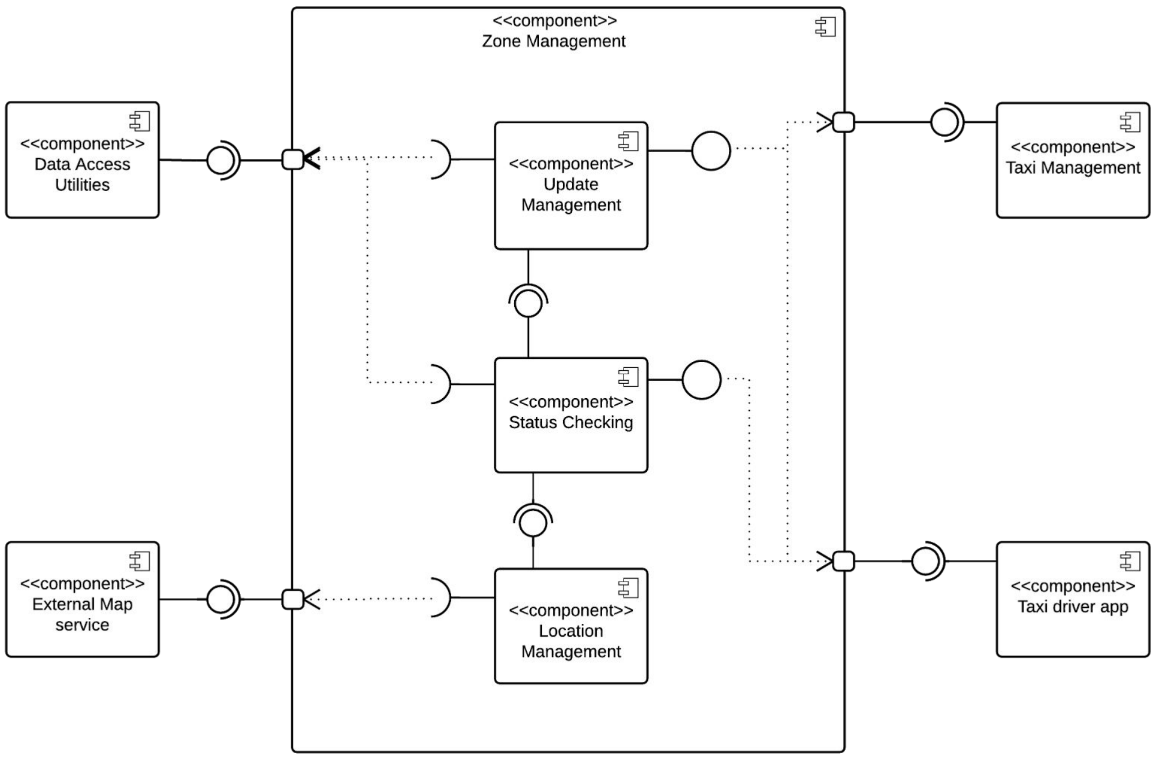

<<component>>
Taxi Management
<<component>>
Data Access
Utilities
<<component>>
Zone
Management
<<component>>
Long Term
Booking
Management
<<component>>
Request
Management
<<component>>
Web Application
<<component>>
Customer app
<<component>>
Taxi driver app

The second component to be analyzed is the Zone Management which is in charge of keeping up to date all the taxi queues. It can be divided into two subcomponents:

- The **Location Management** subcomponent is in charge with computing to which zone the coordinates of a vehicle belong to and thus add them at the end of the queue.

- The **Status Checking** subcomponent has to create a local queue of new available taxis (which is a different one from the ones containing all the available taxis for a zone) where it saves the taxi codes of the vehicles and the zone which it is located in (after sending the received coordinates to the Location Management and receiving the zone which it belongs to). After each insertion it notifies the Update Management component.

- The **Update Management** subcomponent has to update the queues of every city zone, therefore whenever it receives a notification from the Status Checking component it checks the queue of taxis that have recently become available and inserts them into the corresponding queue. It also receives notifications directly from the taxi drivers when they become busy so that it can delete them from the corresponding queue.

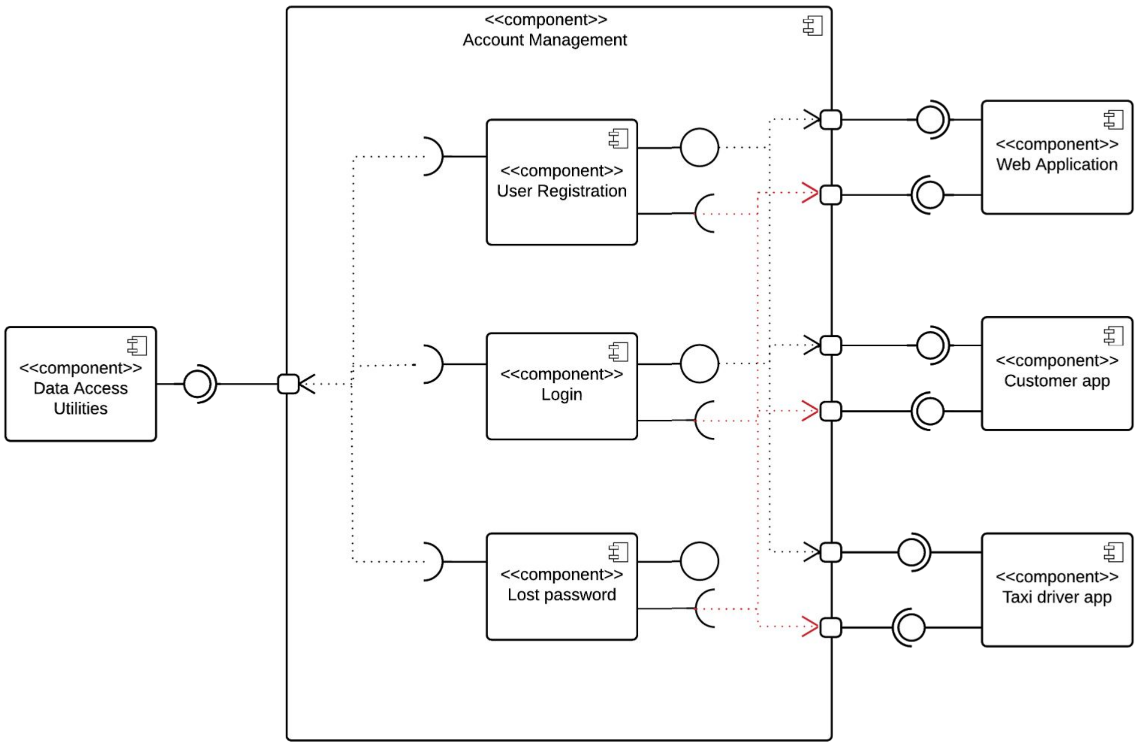


The third component to analyze is the Account Management, where it handles login and registration operations as well as validation of data. It can be decomposed into two subcomponents:

- The **User Registration** subcomponent allows Visitors to create an account in order for them to use myTaxiService services. It first validates if the inserted data has valid form(name,email and password) and saves them in an cache. It then sends a verification message to the inserted address in order to validate it. If all goes well it takes the saved data and sends it to the Data Access Utilities in order to save it.

- The **Login** subcomponent allows all users to access their accounts and from then on interact with the system. The are three types of users that can access the system: registered user, taxi driver, call centre operator; depending on the type of user different credentials can be required during the login phase.

- The **Lost Password** subcomponent permits only to registered users to regain access to their account in the case they forgot the password. The component sends to the email address inserted during the registration phase a link to a temporary page where they can change the password and thus access their account. Note that the temporary page has a lifespan of 15 minutes after which no one can change the password account.

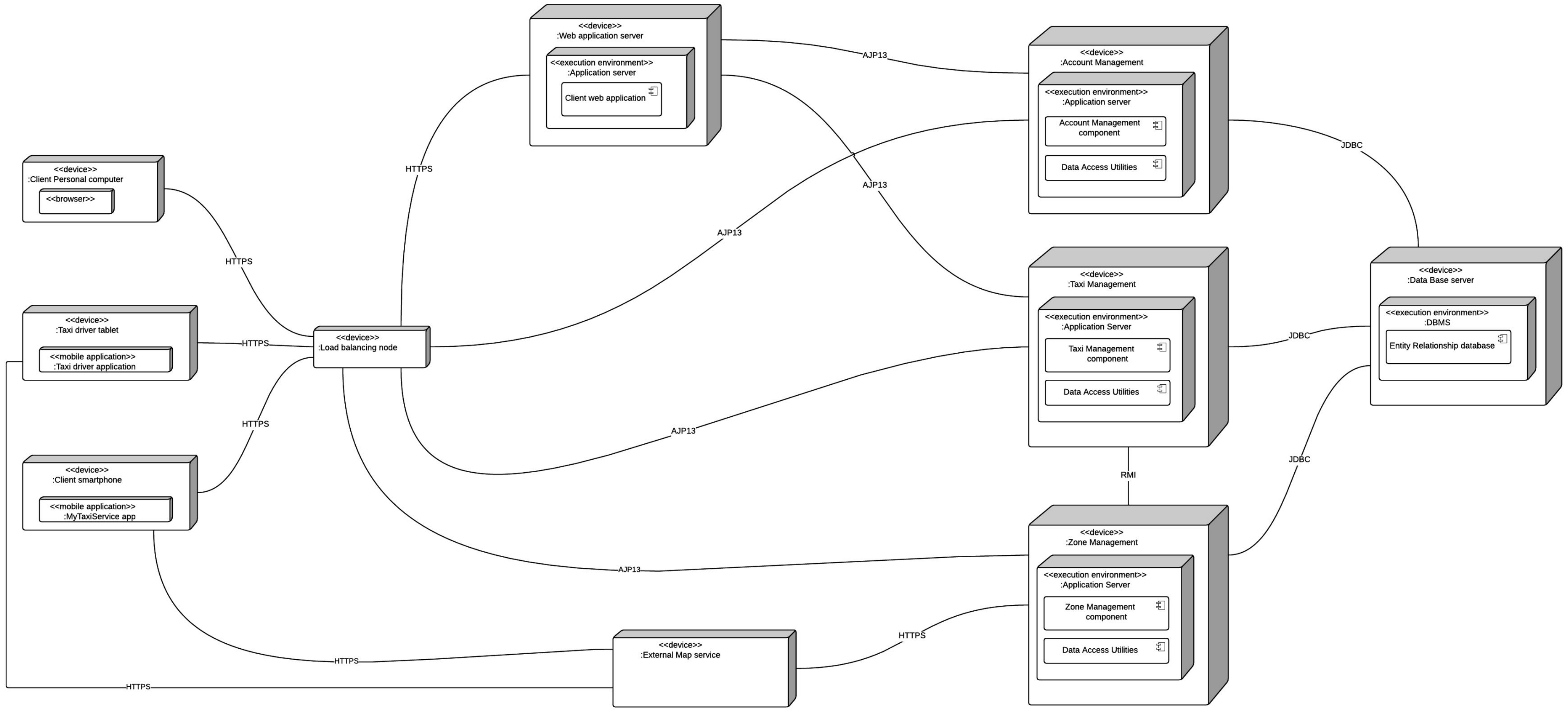


2.4 Deployment view

In this section we will illustrate how the various components explained in the previous section map into the real system. In the following deployment diagram we show how the main software components together with their corresponding hardware form the system.
This architecture was chosen accordingly to the specifications presented in the RASD document. Therefore the system must:

- offer an intuitive way of using the system services;
- reduce the implementation and future expansions costs;
- offer reliability and fast response time independently from the amount of requests in certain time.

For these reasons, and as stated in the RASD document, we will deploy the whole system on a third party cloud infrastructure offered by Google cloud as a PaaS. Therefore we will show in the diagram below only an instance of each hardware component for an easy and comprehensive understanding, while in fact there will be a large amount of them; the load balance will be handled by the third party cloud infrastructure.



<<device>>
:Client Personal computer
<<browser>>
<<device>>
:Taxi driver tablet
<<mobile application>>
:Taxi driver application
<<device>>
:Client smartphone
<<mobile application>>
:MyTaxiService app
<<device>>
:Load balancing node
<<device>>
:Web application server
<<execution environment>>
:Application server
Client web application
<<device>>
:Account Management
<<execution environment>>
:Application server
Account Management component
Data Access Utilities
<<device>>
:Taxi Management
<<execution environment>>
:Application Server
Taxi Management component
Data Access Utilities
<<device>>
:Zone Management
<<execution environment>>
:Application Server
Zone Management component
Data Access Utilities
<<device>>
:Data Base server
<<execution environment>>
:DBMS
Entity Relationship database
<<device>>
:External Map service
HTTPS
HTTPS
HTTPS
HTTPS
AJP13
AJP13
AJP13
AJP13
AJP13
JDBC
JDBC
JDBC
RMI
HTTPS
HTTPS
HTTPS

2.5 Runtime view

In this section we will show the deployed system presented in the previous section in a real world scenario, in order to see how the different components work and interact with each other. For example a real world scenario could be the following one:
Ann is a customer who has requested a cab using the mobile app that she recently installed on her smartphone, Christopher is a taxi driver who has changed his status to ‘Available’ after completing his last ride. Beatrice has just arrived in town at the Bus Station and wants to meet up with her best Diane, but she doesn’t know her way around so she calls Diane to call a taxi for her; thus Diane access her account on the Web application and makes an instant reservation and tells the details of the ride to Beatrice.
The following diagram shows a snapshot of the system running during the above presented scenario and highlights the main processes and threads that are running on the devices.

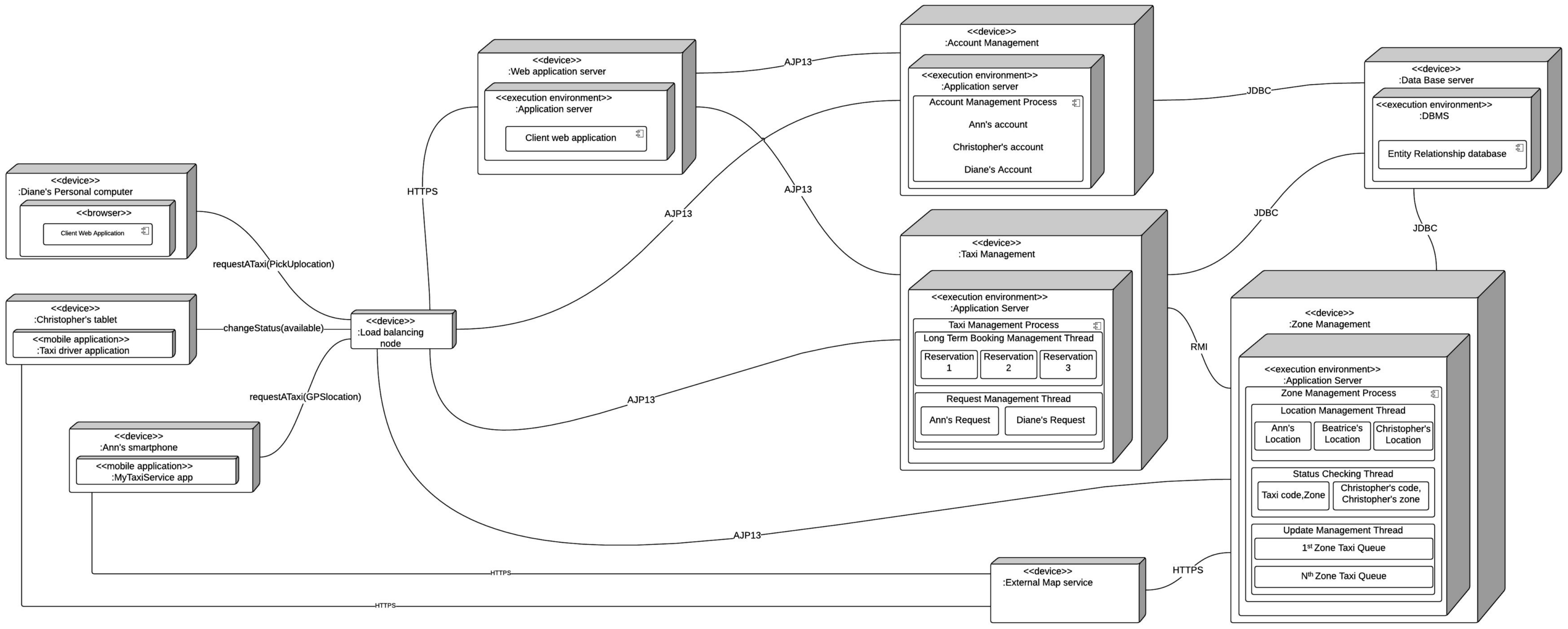
- The Taxi Management server is running a single Taxi Management Process that is responsible for handling different threads, one for every subcomponent running. As shown in the diagram the Long Term Booking Management handles a set of long term reservations; the Request Management controls different a set of requests including the two made by Ann and Diane.

- The Zone Management Server is run a single Zone Management Process that is charge of controlling the different threads in execution on the machine; in particular the Location Management is computing the zone to which a location of a user belongs to, in this case it calculates the zone where Christopher is; the Status Checking inserts Christopher corresponding queue of ‘Recent Available Taxis’(note that this queue is an internal of the Status Checking subcomponent and is not the one of the queues of available taxis for each zone) after receiving the zone to which he belongs to from Location Management; finally the Update Management checks the Status Checking queue in order to see if there are any vehicles to add to the queues of the city.

- The Account Management server is running a single Account Management Process that shows the set of all active users in our scenario; thus it shows Ann’s, Christopher’s and Diane’s accounts.

- The Web Application server are running multiple instances of Web Application which are in charge of the web presentation for each user that uses a personal computer.

- The Load Balancing node is running multiple instances of Apache Web Server in order to redirect HTTPS requests from clients to different parts of the business logic nodes.



<<device>> :Diane's Personal computer
<<browser>>
Client Web Application
<<device>> :Christopher's tablet
<<mobile application>> :Taxi driver application
<<device>> :Ann's smartphone
<<mobile application>> :MyTaxiService app
requestATaxi(PickUplocation)
changeStatus(available)
requestATaxi(GPSlocation)
<<device>> :Load balancing node
HTTPS
<<device>> :Web application server
<<execution environment>> :Application server
Client web application
AJP13
AJP13
AJP13
AJP13
AJP13
<<device>> :Account Management
<<execution environment>> :Application server
Account Management Process
Ann's account
Christopher's account
Diane's Account
<<device>> :Data Base server
<<execution environment>> :DBMS
Entity Relationship database
JDBC
JDBC
JDBC
<<device>> :Taxi Management
<<execution environment>> :Application Server
Taxi Management Process
Long Term Booking Management Thread
Reservation 1
Reservation 2
Reservation 3
Request Management Thread
Ann's Request
Diane's Request
RMI
<<device>> :Zone Management
<<execution environment>> :Application Server
Zone Management Process
Location Management Thread
Ann's Location
Beatrice's Location
Christopher's Location
Status Checking Thread
Taxi code,Zone
Christopher's code, Christopher's zone
Update Management Thread
1st Zone Taxi Queue
Nth Zone Taxi Queue
HTTPS
<<device>> :External Map service
HTTPS
HTTPS

2.6 Component interfaces

In this section we will describe how the various components interact with each other and which functionalities they offer.
The connection between the various users and the servers are made through HTTPS requests which travel through load balancing nodes offered by the third party cloud infrastructure. The load balancing node will have the duty to equally distribute the requests to the machines where our servers are running. The load balancing nodes communicate with the servers through the APJ13 protocol which is a protocol used to communicate with Web servlets in efficient and secure manner. We should also note that components of the system can communicate with each other through RMI (in our case we use it to link the Taxi Management component with the Zone Management component). Finally the components query the databases through JDBC.

Taxi Management component

Long Term Booking Management subcomponent

longTermBooking(Location departure,Location destination,Date date,Time scheduledTime,int numberPassengers, Client username)
If the bookings details are correct (that is both addresses are longer than 500m apart, the departure location is within the city boundaries and the departure date isn't two hours prior or a week after the present time) then the component saves the details of the ride in the database and in a special queue, returning a message to the client with the outcome of the operation.

deleteLongTermReservation(Reservation reservationID, Client username)
If the operation is still possible, that it is made two hours prior to the departure time, then the component will erase tuple containing that reservation from the database, returning a message to the client with the outcome of the operation.

listLongTermReservation(Client username)
If the there are any long term reservation then the component will query the database about the reservations made by the client and lists them on his/her device returns a list of all the long term reservations (if any).

Request Management subcomponent

instantBooking(Location pickUp,int numberPassengers, Client username)
If the bookings details are correct then the component initiates the procedure of creating a new ride by enquiring the Zone Management of the corresponding zone and searches through the queue of taxis of that particular zone and sends a notification to the first vehicle that fits the ride requirements. If the driver accepts then it sends a message about the ride details to the client and saves the ride into the database. If the driver refuses then it searches for a new cab and invokes the *moveToEnd* of the Update Management.

deleteInstantBooking(Client username)
If there are no available taxis in the zone where the client is then the component will send a notification letting him/her know that the request cannot be serviced immediately and will have to wait until a cab becomes available. Thus the system puts the ride details in special queue and waits for a cab to become available. In the meantime if the component receives a cancellation request from the client it removes him from the queue and sends a confirmation of the cancellation of the request.

<u>Zone Management component</u>

Location Management subcomponent

computeZone(Location address)
If the received address is a valid one, that it is in the city, then it returns the zone to which the address belongs to.

Status Checking subcomponent

changeAvailability(Taxi taxiCode, String newStatus)
If the action is possible, that is the newStatus is different from the present one and it contains the string "Available", then it inserts it into a special queue of new available taxis and notifies the Update Management about the change.

Update Management subcomponent

changeAvailability(Taxi taxiCode)
If the action is possible, that is the newStatus is different from the present one and it contains the string "Busy", then it deletes the taxi from the queue of available taxis.

moveToEnd(Taxi TaxiCode)
If the action is possible it sends the vehicle with that taxi code to the end of the queue.

<u>Account Management component</u>

User Registration subcomponent

registerAccount(String name, String email, Password userPass)
If the inserted data is valid and there isn't any other account with the same email address it creates a temporary account and sends a confirmation email to address written during the registration form.

confirmEmail(String email)
It then stores permanently the account into the database and prompts the user to the login page.

Login subcomponent

login(String username, Password userPass)
If the credentials are correct it prompts the user to his personal homepage.

Lost Password subcomponent

recoverAccount(String email)
If the email written is contained in the database it sends an email containing a link to a page where the user can change the password.

rewritePassword(String email, Password newPass)
After the user has clicked on the link provided by the recoverAccount and after setting the new password, it overwrites the password attribute with the new value of the tuple corresponding to that email.

<u>Taxi driver app component</u>

becomeAvailable(Taxi taxiCode,String available)
Sends a request to Status Checking that the taxi is now available to take new calls.

becomeBusy(Taxi taxiCode, String busy)
Sends a request to Update Management that the taxi is not available to take any new calls.

acceptCall(Taxi taxiCode, Ride acceptedRide)
It sends a message to the Request Management that it accepts the call and is asked if he/she wants to use the navigation assistance.

declineCall(Taxi taxiCode, Ride declinedRide)

It sends a message to the Request Management that it declines the call.

getStepByStepNavigation(Location startingPoint, Location destination)
It invokes the navigation assistance from a starting point to a destination offered by the External Map service.

2.7 Selected architectural styles and patterns

In this section we're going to highlight the main architectural styles and patterns that we've decided to adopt that are best in line with the functional and nonfunctional requirements presented in the RASD document. The main styles and patterns used are:

- ***Client-Server***.The type of problem the application should address is the synchronisation of the clients with the personnel who offer services. In fact the vast majority of the user that will want to use the application will want a fast, easy and reliable way of getting in contact with the taxis; therefore it seems only natural to choose this architecture style in order to guarantee these qualities while at the same time reduce the implementation complexity and cost deploying the system in a relatively short time. It is also worth noting that the client-server style ensure that all user have up to date information because it will sufficient to update only one location in order to propagate it to everyone.

- ***Thin Client***.We've decided to adopt the thin client paradigm in order to keep a higher degree of security because all the data will be stored in a central location protected by different layers from the outside world; facilitate the update of the various components, this way we make sure that every users uses up to date data without the concern to stimulate the users to update their application; and finally add new features to our system without concerning ourselves the port these new features to the majority of devices the end users will use.

- ***Model-View-Controller/MVC***. The MVC design pattern is the most common and convenient pattern used in OO (object oriented architectures) which divides into three separate components our application thus enabling us to modify only one part at a time without the need to refactoring the whole system.

- ***Plugins style***. The plugin style is used to add new functionalities to our application in the future. It is the easiest and most effective way to add different features to the system without the hassle of refactoring the application per se. This way we can add new features when time comes and can develop and test these new features separately from the core of the application and simply add them to the system.

- ***N-Tier physical architecture***. It allows us to run the various components on different devices that can be found in locations far away from each other. It also increases the security and the strength of the system with respect to external attacks and machine failure.Furthermore this architecture style gives us the opportunity to scale the number of machines in each tier accordingly to the load of requests.

3 Algorithm design

In this section we explain in detail which are the algorithms running behind the most important functionalities.
Let's see what happens "behind the curtains" when a customer makes an instant reservation.

3.1 Instant Booking

The system receives a request with the signature InstantBooking(Data). Data is a parameter containing, among other identifiers useless to our scope, the address of the customer doing the request. The address is extracted from the request, then a specific method will bind it to the city zone it belongs to. The binding occurs by searching on a table in which, for each address, there is the relative city zone. The correctness of the address, the belonging of the address to the city, is checked if the address appears in the table. Otherwise, the systems warns the user that the address is wrong (see 3.1.1.8 in the RASD).
Now the system knows the city zone the call is coming from and needs the set of taxis that are currently there.
So now starts a polling on the taxi drivers: a message is sent to all the active devices.
If the active device is available, then it replies with a ping. Otherwise, it does nothing.
The device's ping contains, among other data, the address of the current position and the relative driver's ID.
The system now gathers all the replies, extracting from the address the city zone (by using the same method invoked before).
There will be an association between driver ID and city zone. Couples are grouped based on the city zone. Within each city zone, drivers are put in a queue (the initial position depends on the speed of response of the ping). Queue is a data structure that uses the FIFO (first in, first out) policy.
This process aims to get a "fresh" queues of taxis, for example after the reboot of the system; anyway the queues are updated much more frequently (see the paragraph 5.3 for further informations).
The first driver of the queue relative to the customer's city zone is able now to accept or decline the call. If the taxi driver declines, the system will move his taxi to the last position of the queue, then it will forward the request to the second taxi in the queue, which now is the first one, and so on.
If the taxi driver accepts the call, the system will move his taxi to the last position of the queue.
From now on, the following requests will be handled with the already existing list of queues.
The case of empty queue of taxis (no taxis available) is worth mentioning. In that case, the request done by the customer is stored in a queue and he is served as soon as a taxi becomes available. Customer is able to delete his request anytime.

3.2 Long Term Booking

We want to extend the procedure presented above to the long term booking. Actually, the process is pretty much the same, except for the fact that the request by the customer doesn't happen in real time. Its request is stored in the system and will be sent to the first taxi driver in queue ten minutes before the scheduled time. The process of taxi selection is the same as above.

3.3 Queue Management

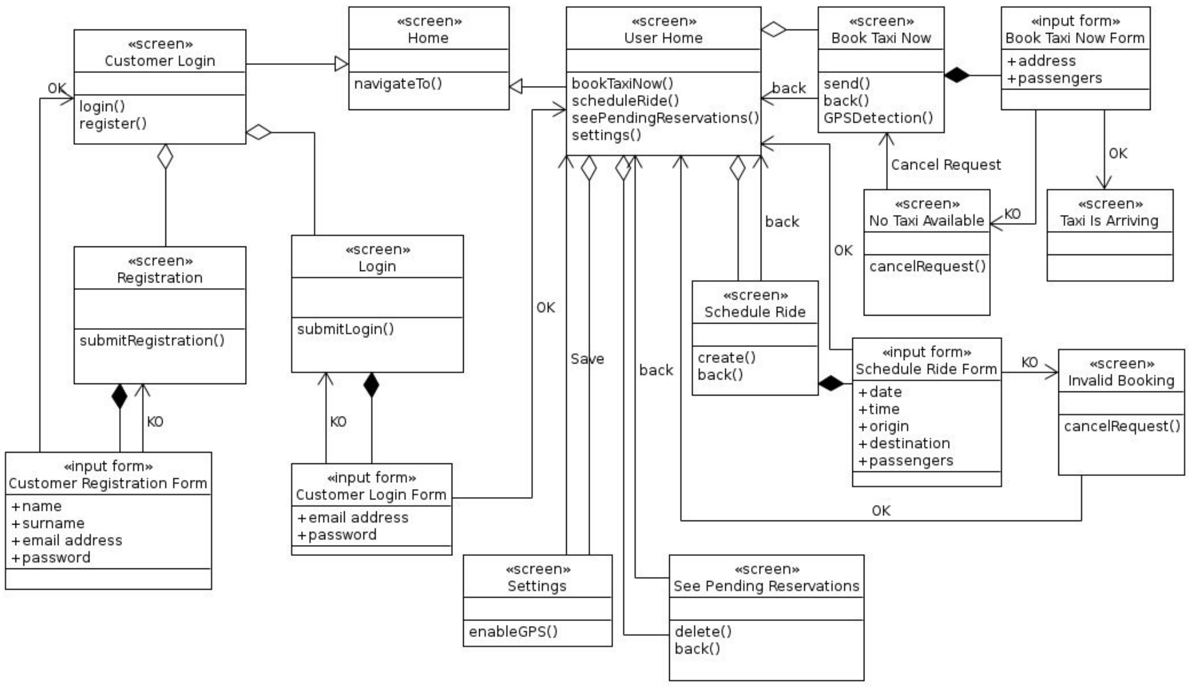
While the algorithm to create queues from scratch will be run in a few cases, 99.9% of the times the queues will be updated according to taxi driver's actions. The actions that will update the queue are the following one.

1. Disconnection (myTaxiService application crash or loss of connection): the system detects the loss of connection when it does not receive a ping from the active device in less than 120 seconds. The taxi ID is removed from the queue.
2. Connection: the taxi driver device sends its position, then the system extracts the address and puts the cab in the last position of the specific zone's queue.
3. Change of availability:
    a. Available to unavailable: the taxi is removed from the queue.
    b. Unavailable to available: same of connection.

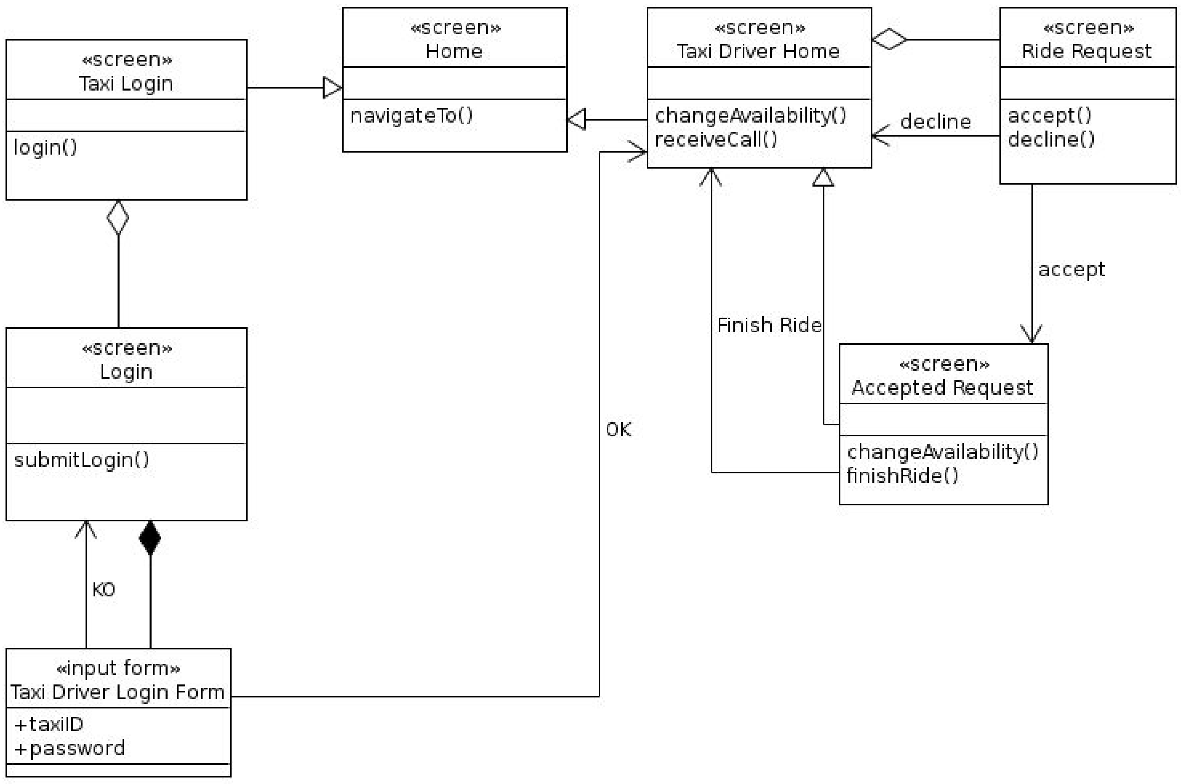
4 User Interface Design

The following diagrams represent a compact view of the user interfaces presented in detail in the RASD, section 3.1.

4.1 Customer UX



4.2 Taxi Driver UX



5 Requirements Traceability

Goal	Description	Component	Subcomponent
1	Visitors can register	Account Management	User Registration
2	Unidentified users can login	Account Management	Login
3	Users can make instant bookings	Taxi Management	Request Management
4	Users can make long term bookings	Taxi Management	Long Term Booking Management
5	Users can delete long term bookings	Taxi Management	Long Term Booking Management
6	Taxi drivers can change their status	Zone Management	Update Management
7	Taxi drivers can accept or decline a call	Taxi Management	Request Management
8	Taxi drivers can use GPS to reach the customer	External Map Service	Google Maps API
9	Taxi drivers can login	Account Management	Login

For further details about components and subcomponents, read the section 2.3.

6 Appendix

6.1 Hours spent

- Adrian Mihai Berbieru: ~20 hours
- Attilio D’Onofrio: ~20 hours

6.2 Software used

- Google Docs (http://www.drive.google.com): to redact and to format this document.
- Draw.io (http://www.draw.io/): to create the diagrams;