Design Document

Rizzi Matteo (789467) & Scandella Claudio (853781)

December 4, 2015

Contents

1 Introduction

1.1 Purpose

This document explains how the application architecture should be built, allowing the developers to know how to implement it as expected. It will be given an high level components description and their distribution on the different tiers, according to their functionalities for the service, and how they interact each other through protocols.

1.2 Scope

The aim of this project is to develop the myTaxiService application, which allows users to request/reserve a taxi while on the taxi driver side it allows to manage different client calls. For a more detailed overview of the scope you can see the corresponding RASD document.

1.3 Definitions, Acronyms and Abbreviations

1.3.1 Definitions

- Layer: Software levels in which an application is divided.
- Tier: It's an hardware level on which the software layer is installed.
- Protocol: set of rules that govern every exchange of data between two entities. Protocols can exist for the transfer of files or for access to networks at all levels

1.3.2 Acronyms

- UML: Unified Modeling Language.
- GPS: Global Positioning System.
- RASD: Requirement Analysis and Specifications Document.
- DB: DataBase.
- RM: Request Manager.
- TD: Taxi Driver.
- UWAV: UsersWebAppView.
- UMAV: UsersMobileAppView.
- TDDAV: TaxiDriversDeviceAppView.
- GUI: Graphical User Interface.

1.3.3 Abbreviations

- [i,j]: i-goal and j-functional requirement.

1.4 Reference Documents

- RASD FINALE.pdf;
- DD TOC.pdf

1.5 Document Structure

The Design Document is divided in five sections:

- Introduction: it gives an initial description of the document and some informations to understand the concepts explained in the following sections;
- Architectural Design: this important section gives a detailed view and explanation of the architecture chosen upon which criteria;
- Algoritmh Design: there will be shown some of the algorithms that will be implemented in the final software application in order to ensure developers to follow design specifications;
- Requirements Traceability: in this section you can found a table that shows which functional requirement is fullfilled by which component of the system;
- References: This section contains informations about how we created this document, including the software tools used and the hours needed to redact the document for each developer.

2 Architectural Design

2.1 Overview

The architectural design section is divided into two great sub-sections: the first section is about the main components that represents our software system and how they interact with each other. It will be explained using both the text but also by Uml diagrams like component, deployment and sequence diagrams. The second section explain the architecture of our software system and all the design patterns and choices that we will use to develop the application.

2.2 High level components and their interaction

We divided our components in three types:

- Model Components
- Controller Components
- View Components

2.2.1 Model Components

Data manager This component manages all the accesses to the data(both users and taxi drivers data) that are contained in the database.

2.2.2 View Components

All this view components are connected to managers that guarantee the reception and the sending of messages and requests.

UsersWebAppView This component manages the interaction between the myTaxiService application and the user connected via Web

UsersMobileAppView This component manages the interaction between the myTaxiService application and the user connected with his mobile device

TaxiDriversDeviceAppView This component manages the interaction between the myTaxiService application and the Taxi Driver connected with his device

2.2.3 Controller Components

Dispatcher This component guarantees the distribution of notifications and messages from the system to taxi drivers and/or users. For example the dispatcher allows taxi drivers to receive notifications of incoming calls; it also allows users to receive, for example, notifications of requests or reservations with taxi code and waiting time.

Request receiver This component takes every request coming from the users and , basing on the type of the request the component has received, it will be transferred to the correct manager. For example a login request will be transferred to the Access Manager, while a reservation call will be transferred to the Request Manager.

Request Manager This component receives two main type of calls:

- Ride requests;
- Ride reservations;

It also receives other requests, but always concerning ride requests or reservations(like modification/delete of a reservation).

His goal is to deliver the calls received from the request receiver to taxi drivers with the help of the Queue Manager and the Position Manager.

In case the call is a ride reservation call the process is the same but the taxi will be allocated for the client only ten minutes before the meeting time.

Position manager The position manager is helpful because he registers every GPS position of taxi drivers and he can also give the amount of time the user must wait until the cab will arrive to his address.

Queue Manager The Queue Manager receives the GPS position of every taxi in every queue of the different zones of the city thanks to the Position Manager and he must guarantee a fair distribution of taxi in the city.

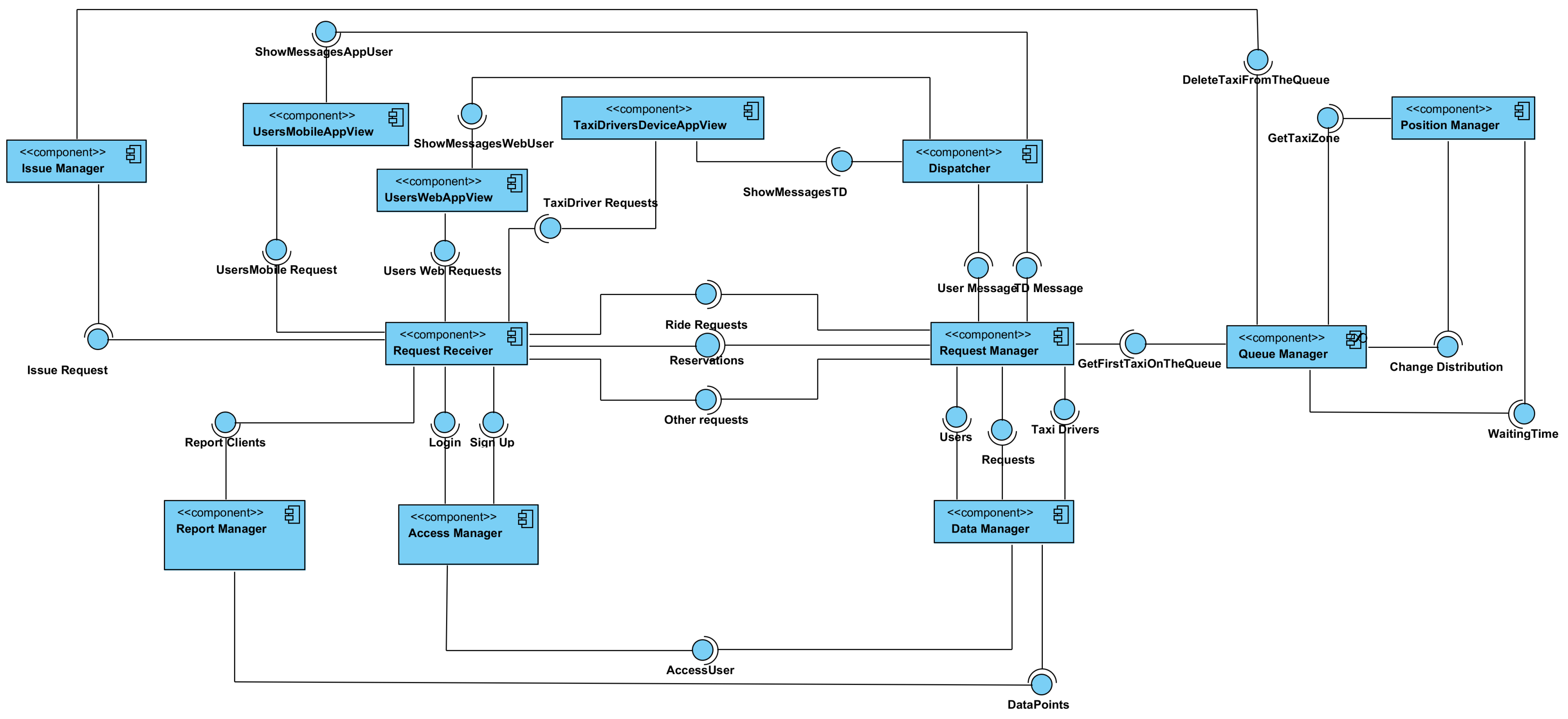
Report manager This component receives report requests sent to the request receiver from taxi drivers and he has to add one Report Point to the user if the motivation is valid.

Issue Manager This component receives the Issues messages sent to the Request Receiver from the taxi drivers and communicates with the Queue Manager in order to eliminate the broken taxi from the queue and in order to send a new taxi to the client.

Access Manager This component receives Login and Sign Up requests from the Request Receiver and he must complete the Login and the Sign Up requested by clients. He also has access to the Data Manager in order to change the state of the User, or in order to store the data of the new User into the DB.

2.3 Component view

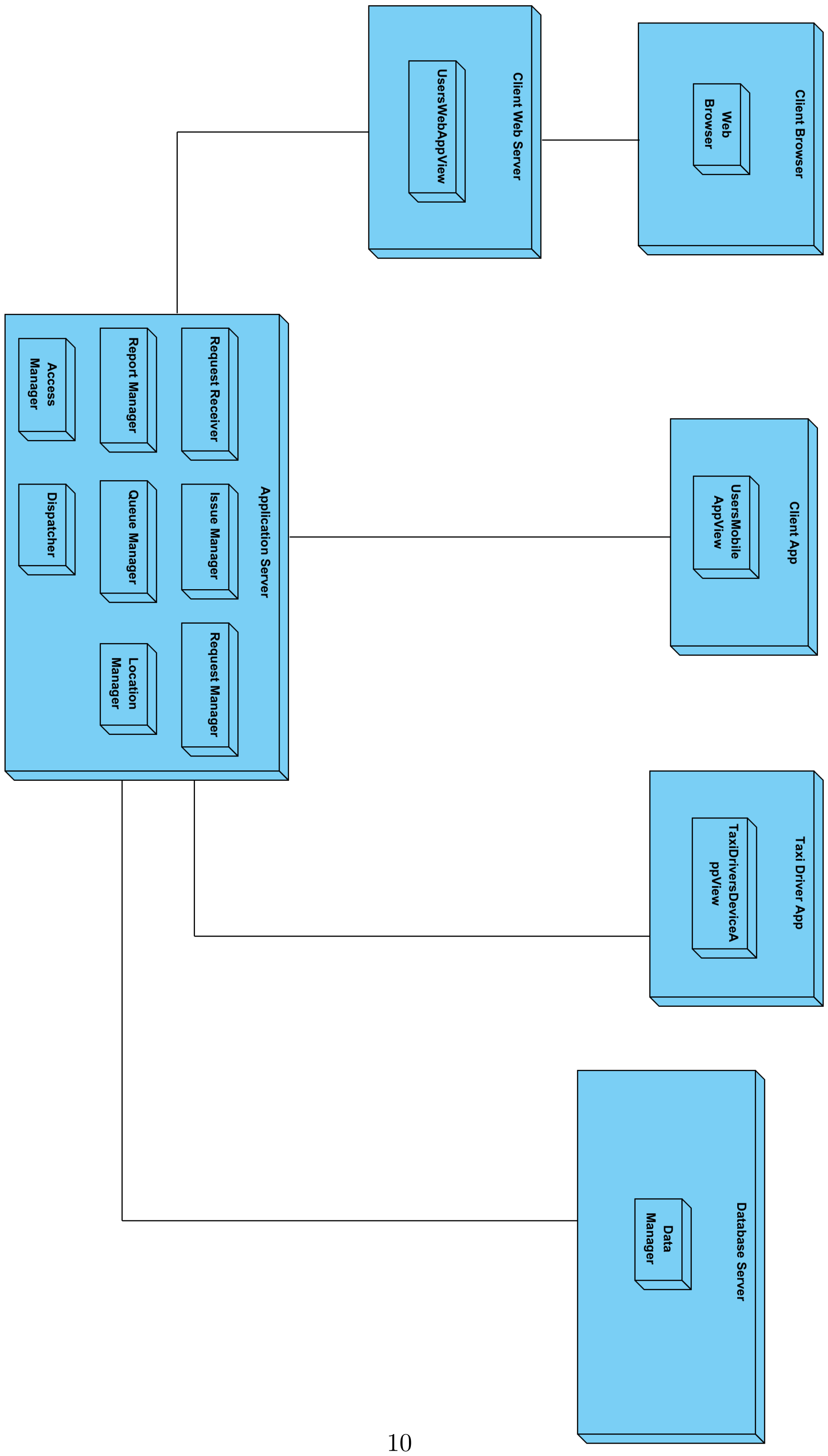
This is the Component Diagram we designed to describe the interaction between the components of our application.



<<component>> Issue Manager
<<component>> UsersMobileAppView
<<component>> UsersWebAppView
<<component>> TaxiDriversDeviceAppView
<<component>> Dispatcher
<<component>> Position Manager
<<component>> Request Receiver
<<component>> Request Manager
<<component>> Queue Manager
<<component>> Report Manager
<<component>> Access Manager
<<component>> Data Manager
ShowMessagesAppUser
ShowMessagesWebUser
DeleteTaxiFromTheQueue
GetTaxiZone
ShowMessagesTD
TaxiDriver Requests
UsersMobile Request
Users Web Requests
User Message
TD Message
Issue Request
Ride Requests
Reservations
Other requests
GetFirstTaxiOnTheQueue
Change Distribution
WaitingTime
Report Clients
Login
Sign Up
Users
Requests
Taxi Drivers
AccessUser
DataPoints

2.4 Deployment view

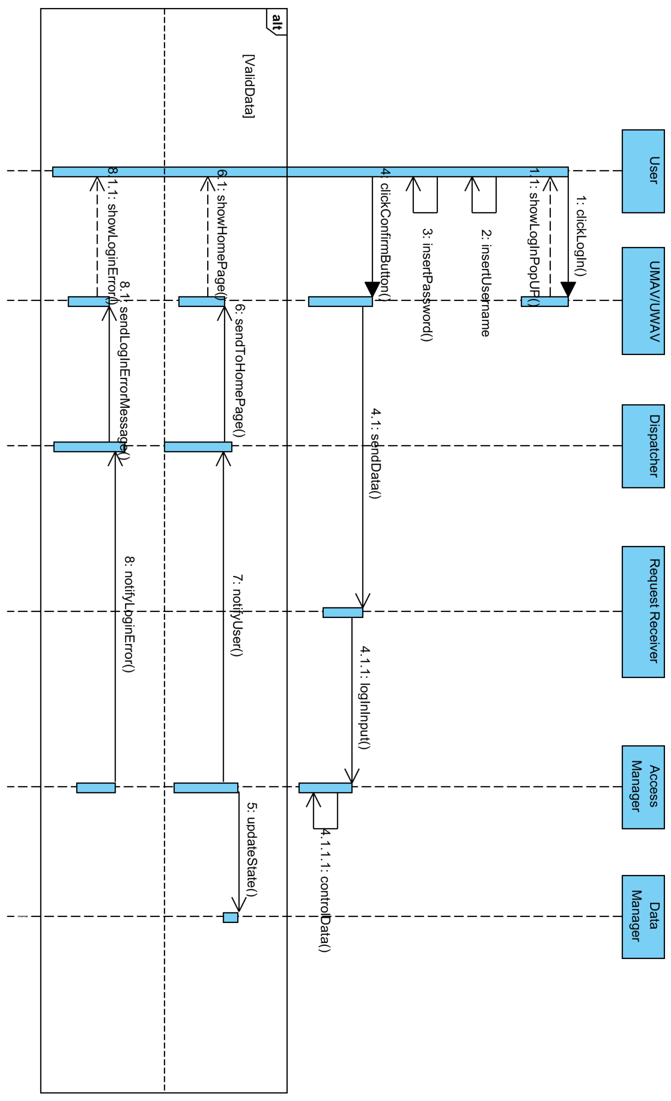
This is the deployment diagram we designed to show how hardware and software components of our application are associated.


Client Browser
Web Browser
Client Web Server
UsersWebAppView
Client App
UsersMobile AppView
Application Server
Request Receiver
Issue Manager
Request Manager
Report Manager
Queue Manager
Location Manager
Access Manager
Dispatcher
Taxi Driver App
TaxiDriversDeviceAppView
Database Server
Data Manager

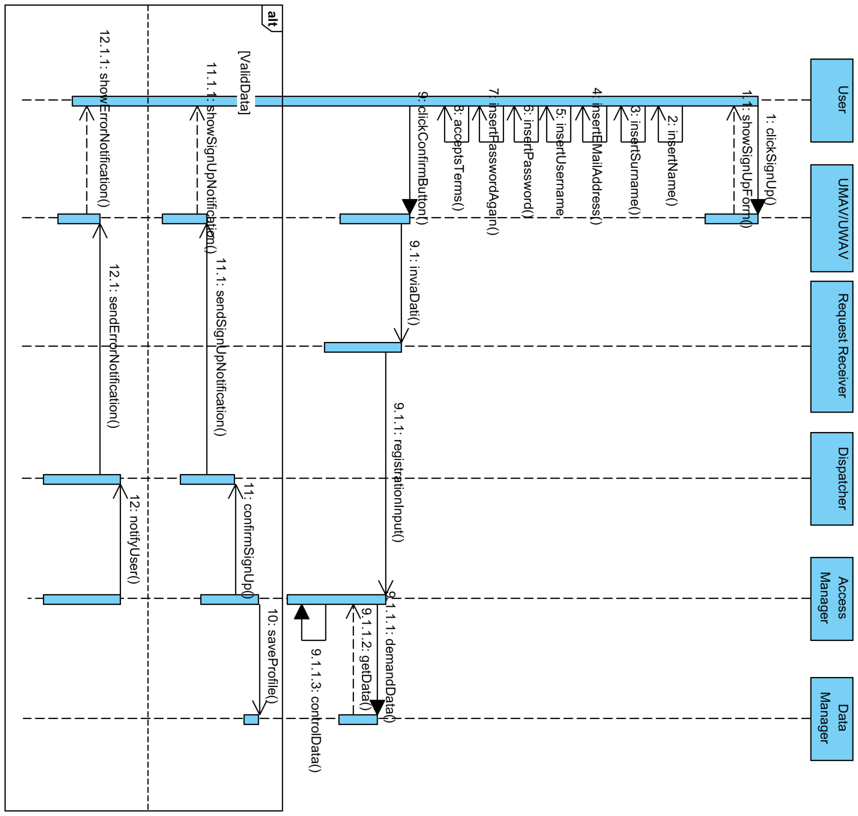
2.5 Runtime view

In this section we will show the interaction between some components using Sequence Diagrams.

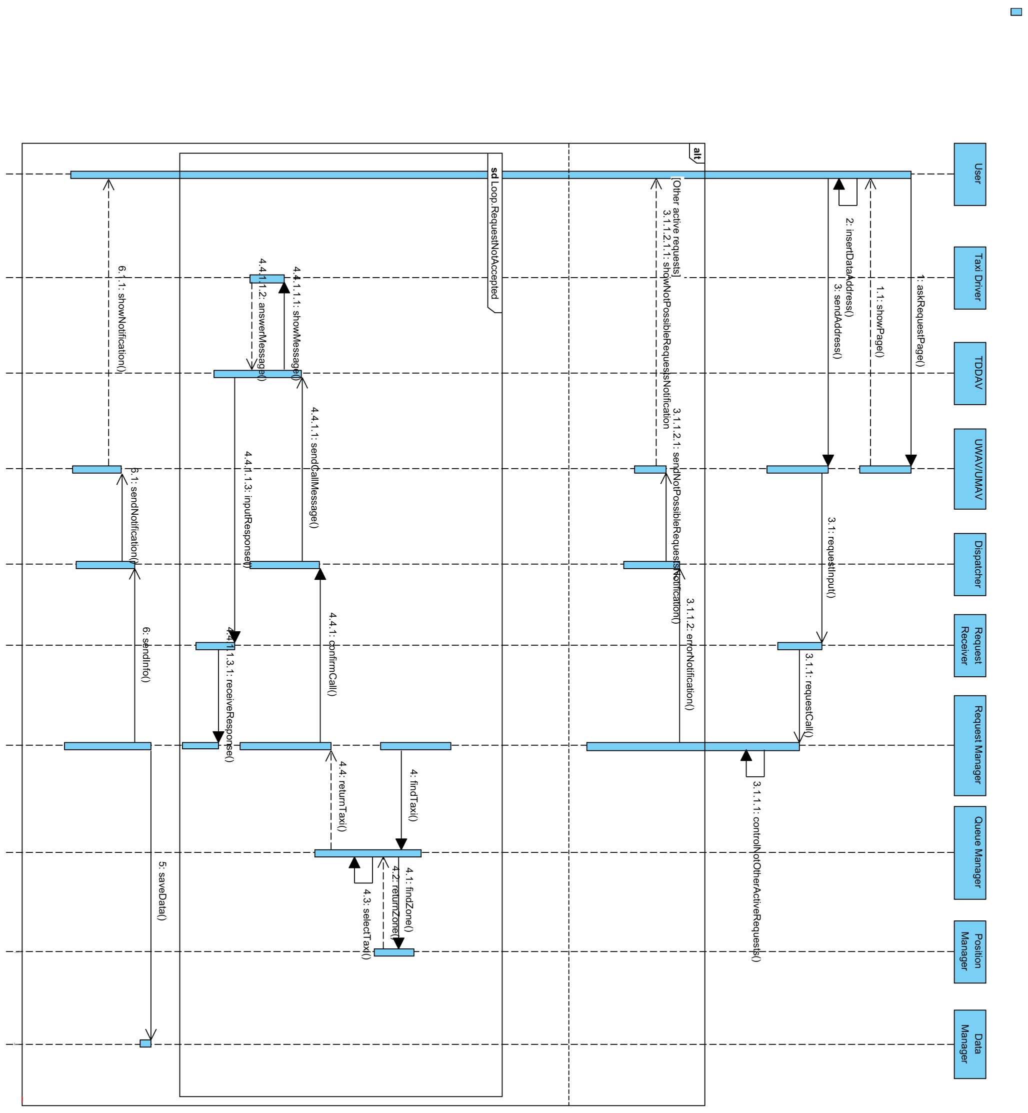
2.5.1 Login



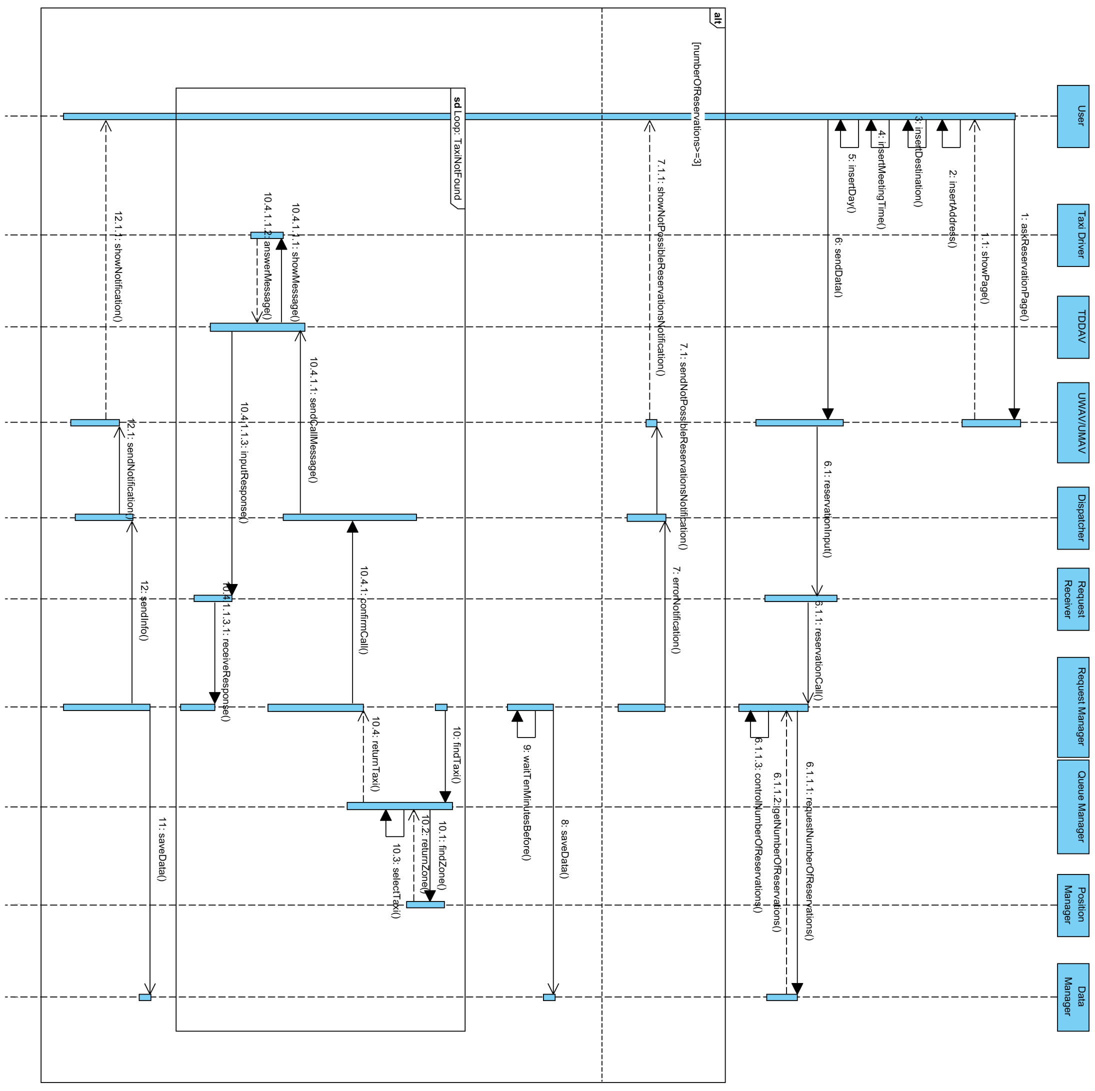
2.5.2 Registration



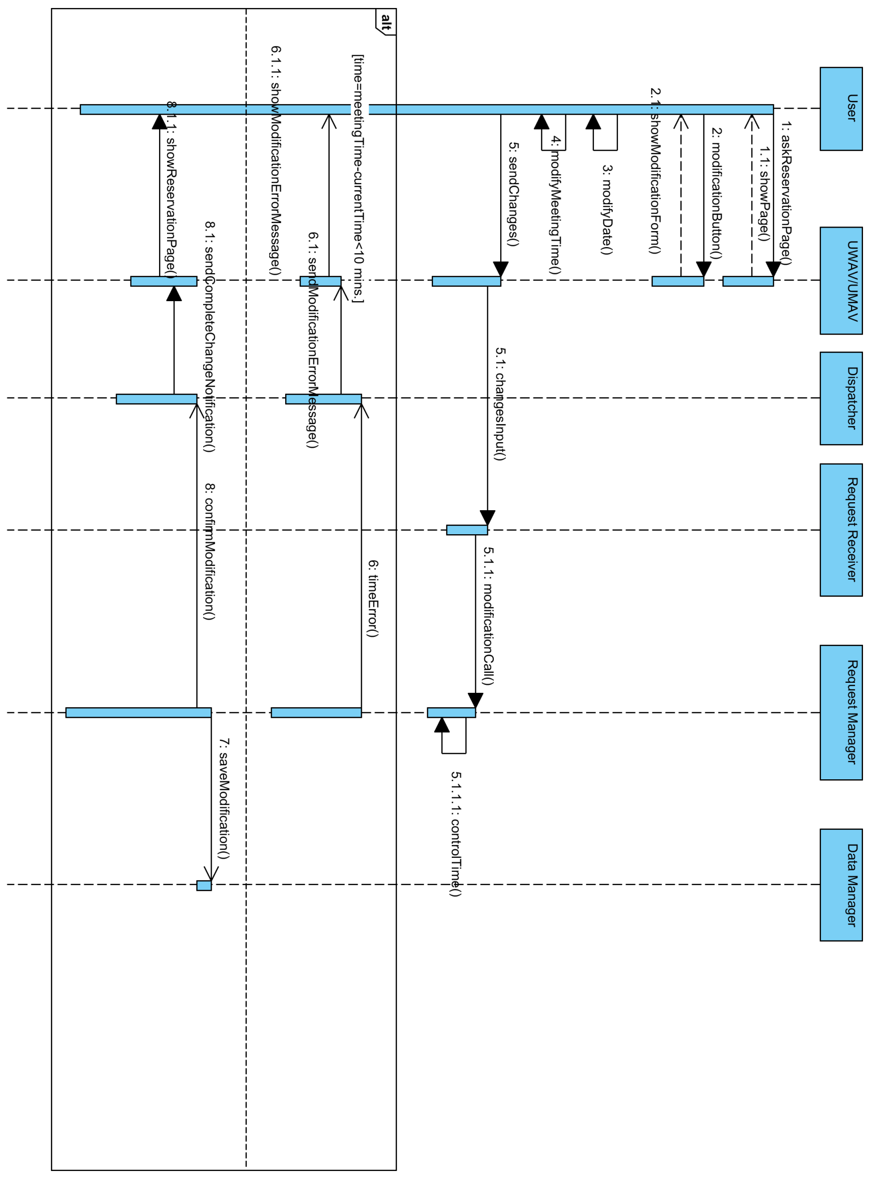
2.5.3 Ride Request



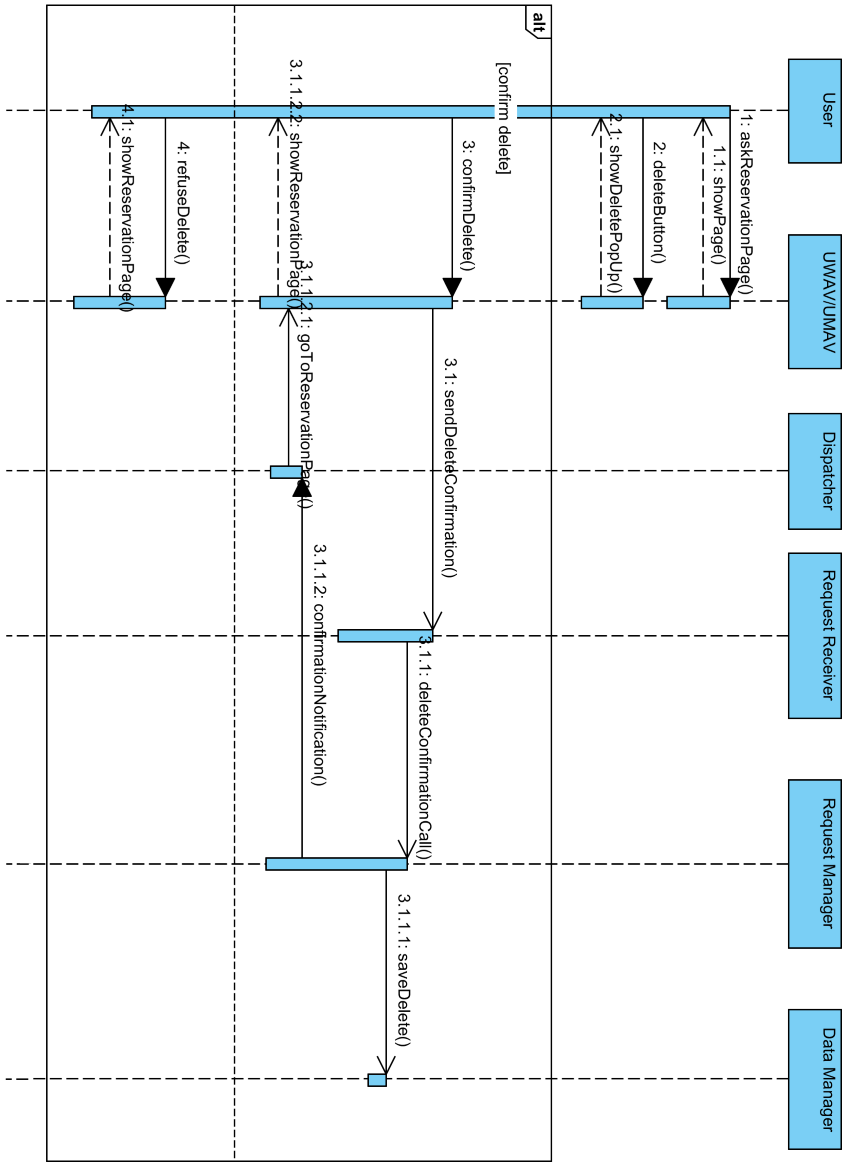
2.5.4 Reservation



2.5.5 Modify request



2.5.6 Delete request



2.6 Component interfaces

2.6.1 Model Components Interfaces

The Data Manager communicates with the request manager via three interfaces:

- Users in order to save/load data about users.
- Requests in order to save/load data about requests.
- Taxi Drivers in order to save/load data about taxi drivers.

2.6.2 Controller Components Interfaces

Issue Manager The Issue Manager communicates with the request receiver with one interface:

- Issue Requests: It receives issue calls from the Request Receiver and he has to solve the issue that has happened.

It also communicates with the Queue Manager with only one interface:

- DeleteTaxiFromTheQueue: this interface is necessary in order to delete a taxi that had any kind of problem during a ride and is no longer available to drive other clients to their destination. So the Issue Manager communicates with the Queue Manager in order to delete the Taxi from the queue of the zone where the taxi is located.

Report Manager The Report Manager communicates with the request receiver with only one interface:

- Report Clients: this interface receives reports from the request receiver, and the report manager has to control is the report is valid.

The report manager communicates also with the Data Manager with the:

- DataPoints interface: this interface increases the report points of the user in case the motivation of the report sent by the Taxi Driver is valid.

Access Manager The Access Manager communicates with the Request Receiver with two different interfaces:

- Login: the login interface manage the login phase of clients.
- Sign Up: it manages the sign up of clients to the application.

It also communicates with the data manager with one interface:

- AccessUser: this interface takes a login from the access manager and if the data inserted by the client are valid it changes the state of the client to logged in, or if it receives a sign up from the access manager it demands the data to the data manager in order to permit to the Access Manager to control the correctness of the data inserted. If the data are valid they are stored in the database thanks to the data manager.

Request Receiver The Request Receiver communicates with the Request Manager with three different interfaces basing on the type of the call the RM has to manage:

- Ride requests: The request receiver takes ride calls from clients and he transfers it to the request manager in order to take care of them.
- Reservations: The request receiver takes reservation calls from clients and he transfers it to the request manager in order to take care of them.
- Other Requests: The request receiver takes other type of calls(modification of a reservation, delete of a reservation) from clients and he transfers it to the request manager in order to take care of them.

Dispatcher The dispatcher communicates with the Request Manager through two interfaces:

- TD Message: that captures messages and notifications the request manager must send to Taxi Drivers ;
- User Message: that captures messages and notifications the request manager must send to Users.

The dispatcher also communicates with View Components in order to send to the view messages and notifications captured from the Request Manager, that the view will show to Users or Taxi Drivers(the show Interfaces).

Position Manager The Position Manager communicates with the Queue Manager with three interfaces:

- GetTaxiZone: this interface allows the Queue Manager to acknowledge the zone where the Taxi Driver is in order to send the first taxi available on that zone to the requiring client.
- ChangeDistribution: Controls the distribution of taxi in the city and tells taxi drivers to change zone in case of bad distribution.
- WaitingTime: It returns the waiting time basing on the position of the taxi driver.

Queue Manager The Queue Manager communicates with the Request Manager with only one interface:

- GetFirstTaxiOnTheQueue: After receiving the zone where the taxi is the Queue manager passes the first taxi of the queue to the request manager to send the taxi driver the notification.

2.6.3 View Components Interfaces

The view components (UWAV, UMAV, TDDAV) communicate with the Request Receiver in order to send it the requests from their client(at example the UMAV sends UsersMobile Requests, and the TDDAV sends Taxi Drivers requests).

2.7 Selected architectural styles and patterns

We have decided to adopt a 3-tiers architecture because it offers some important features such as performance and scalability:

- Performance because clients, called "thin clients", will take care only of view of the contents, thus the major calculations will be executed by the application server;
- Scalability because any modification on the system will be hidden to clients either they are users or taxi drivers.

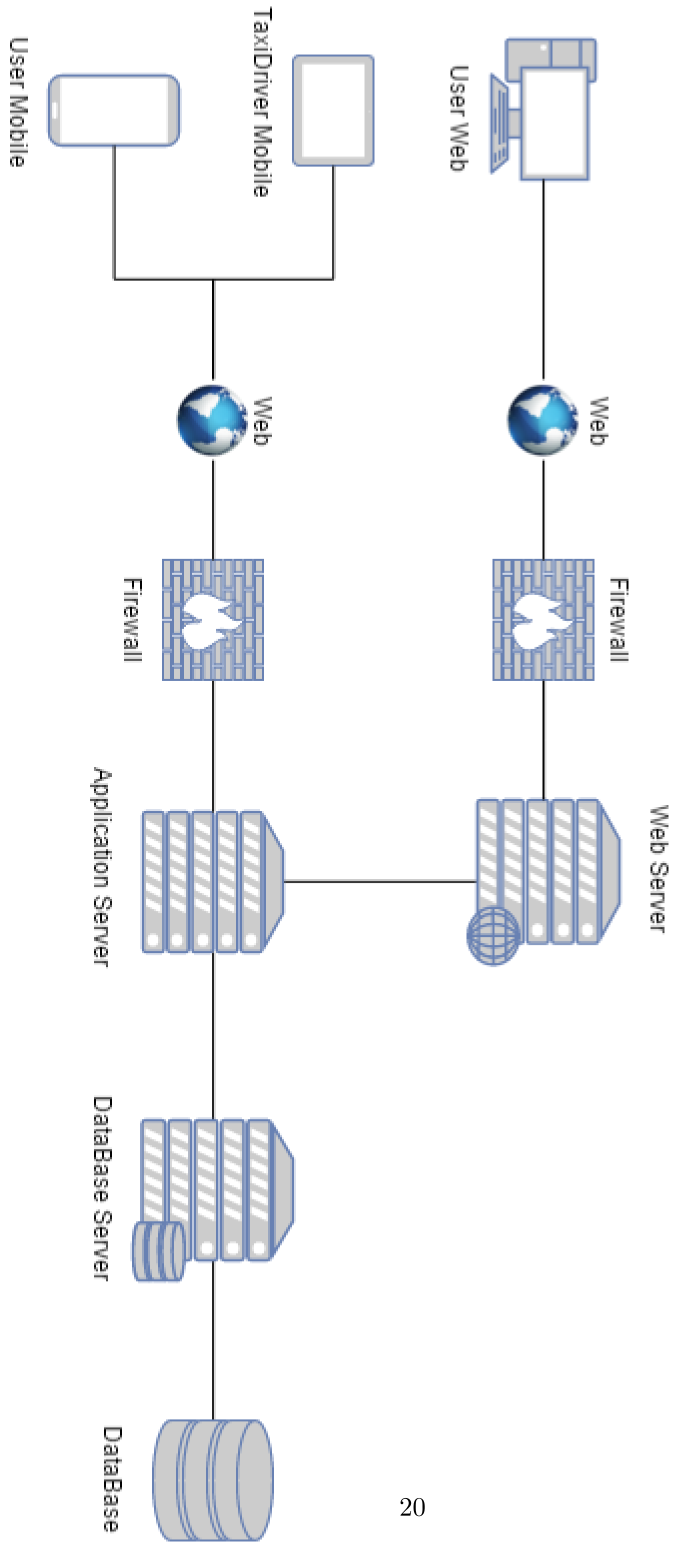
The architecture consists of three servers with different tasks:

1. A Database server to manage data on the database;
2. An Application server that is the main central computing unit;
3. A Web server used by clients through browser web.

Users and TaxiDrivers's mobile devices will have an application installed which will be used to communicate with the application server and to show GUI.

Clients that use browser web application communicates with the web server to visualize the GUI. The web server shall communicate with the application server to computing users requests. The application server can communicate with the database server whenever data need to be written or read.

All of these three ways of connecting follow the client-server paradigm indeed the server will reply only in case of a message from terminals. At the end to guarantee security in the communication and to protect data there is a firewall that protect each interface of the web server and the application server that use internet connection.


User Web
TaxiDriver Mobile
User Mobile
Web
Web
Firewall
Firewall
Web Server
Application Server
DataBase Server
DataBase

3 Algoritmh Design

3.1 Queue Management Algoritmh

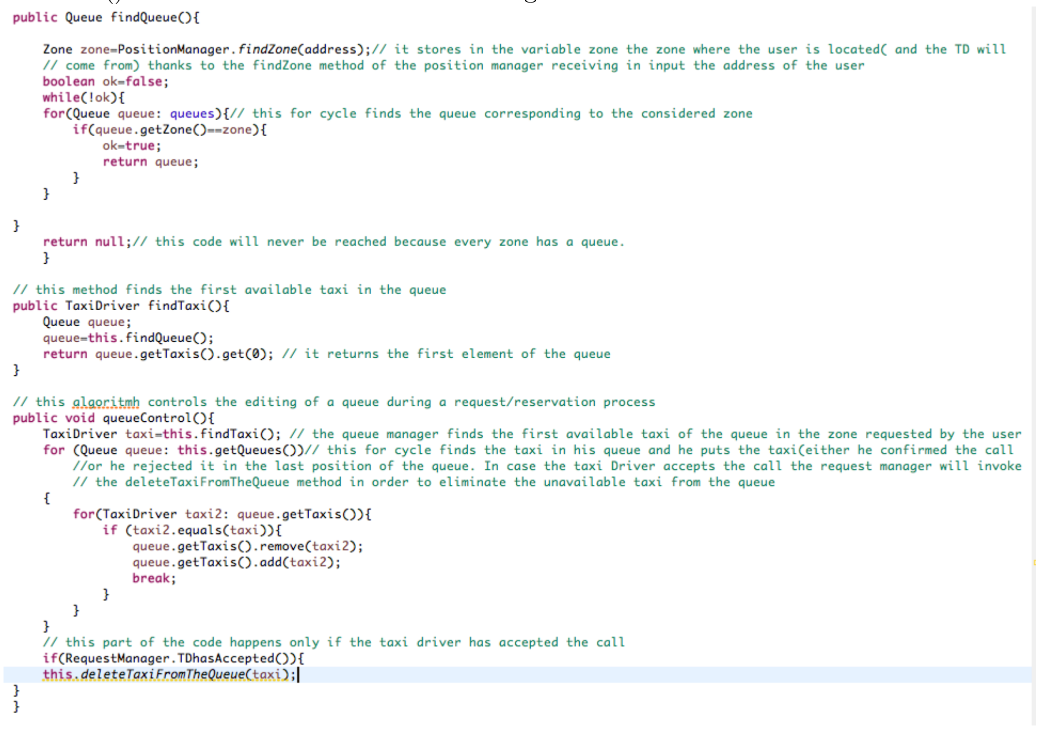
This algoritmh that is represented by the queueControl() method has the objective to find a taxi for a request/reservation call arriving from the request manager. So the queue manager finds the first taxi available in the zone requested by the client by using, one after the other the findTaxi(), findQueue(), and findZone() methods. The last one is a Position Manager method and it receives in input the address of the user and it expresses the address as a zone of the city. So, when the taxi has been found by the Queue Manager, the taxi is transferred to the last position of the queue. Also, if the Taxi Driver, once the request manager has sent him this call, will accept the call he will be removed from the queue.

Informations on the methods findQueue() is a method of the Queue Manager class.

findTaxi() is a method of the Queue Manager class.

queueControl() is a method of the Queue Manager class.

findZone() is a method of the Position Manager class.



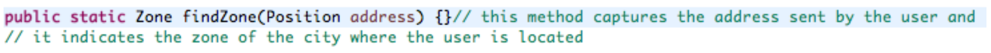
```java
public Queue findQueue(){

    Zone zone=PositionManager.findZone(address);// it stores in the variable zone the zone where the user is located( and the TD will
    // come from) thanks to the findZone method of the position manager receiving in input the address of the user
    boolean ok=false;
    while(!ok){
    for(Queue queue: queues){// this for cycle finds the queue corresponding to the considered zone
        if(queue.getZone()==zone){
            ok=true;
            return queue;
        }
    }

}
    return null;// this code will never be reached because every zone has a queue.
    }

// this method finds the first available taxi in the queue
public TaxiDriver findTaxi(){
    Queue queue;
    queue=this.findQueue();
    return queue.getTaxis().get(0); // it returns the first element of the queue
}

// this algoritmh controls the editing of a queue during a request/reservation process
public void queueControl(){
    TaxiDriver taxi=this.findTaxi(); // the queue manager finds the first available taxi of the queue in the zone requested by the user
    for (Queue queue: this.getQueues())// this for cycle finds the taxi in his queue and he puts the taxi(either he confirmed the call
        //or he rejected it in the last position of the queue. In case the taxi Driver accepts the call the request manager will invoke
        // the deleteTaxiFromTheQueue method in order to eliminate the unavailable taxi from the queue
    {
        for(TaxiDriver taxi2: queue.getTaxis()){
            if (taxi2.equals(taxi)){
                queue.getTaxis().remove(taxi2);
                queue.getTaxis().add(taxi2);
                break;
            }
        }
    }
    // this part of the code happens only if the taxi driver has accepted the call
    if(RequestManager.TDhasAccepted()){
    this.deleteTaxiFromTheQueue(taxi);
}
}
```



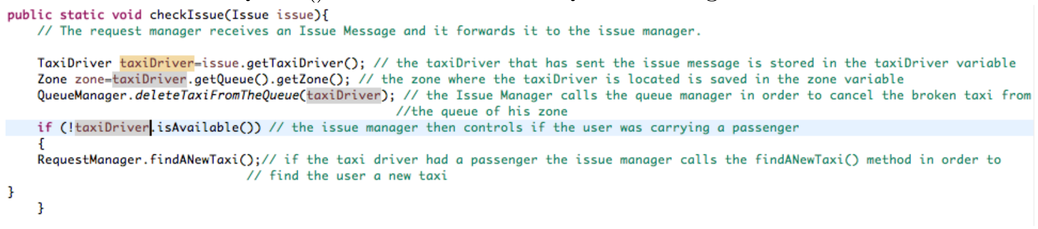
```java
public static Zone findZone(Position address) {}// this method captures the address sent by the user and
// it indicates the zone of the city where the user is located
```

3.2 Issue Management Algoritmh

After receiving a message from the request receiver the issue manager must work on this issue in order to solve it. In this case in order to simple things we call issue something that will not give the taxi driver to use his cab anymore. So the Issue Manager must delete the taxi from the queue of the zone where the taxi is stored and if the taxi driver wasn't available at the moment of the issue, the issue manager will send a request to the Request Manager, in order to find a new taxi to his passenger.

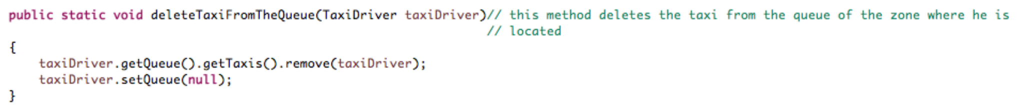
Informations on the methods checkIssue() is a method of the Issue Manager class.

deleteTaxiFromTheQueue() is a method of the Queue Manager class.



```
public static void checkIssue(Issue issue){
    // The request manager receives an Issue Message and it forwards it to the issue manager.

    TaxiDriver taxiDriver=issue.getTaxiDriver(); // the taxiDriver that has sent the issue message is stored in the taxiDriver variable
    Zone zone=taxiDriver.getQueue().getZone(); // the zone where the taxiDriver is located is saved in the zone variable
    QueueManager.deleteTaxiFromTheQueue(taxiDriver); // the Issue Manager calls the queue manager in order to cancel the broken taxi from
                                                    //the queue of his zone
    if (!taxiDriver.isAvailable()) // the issue manager then controls if the user was carrying a passenger
    {
    RequestManager.findANewTaxi();// if the taxi driver had a passenger the issue manager calls the findANewTaxi() method in order to
                              // find the user a new taxi
}
    }
```



```
public static void deleteTaxiFromTheQueue(TaxiDriver taxiDriver)// this method deletes the taxi from the queue of the zone where he is
                                                                // located
{
    taxiDriver.getQueue().getTaxis().remove(taxiDriver);
    taxiDriver.setQueue(null);
}
```

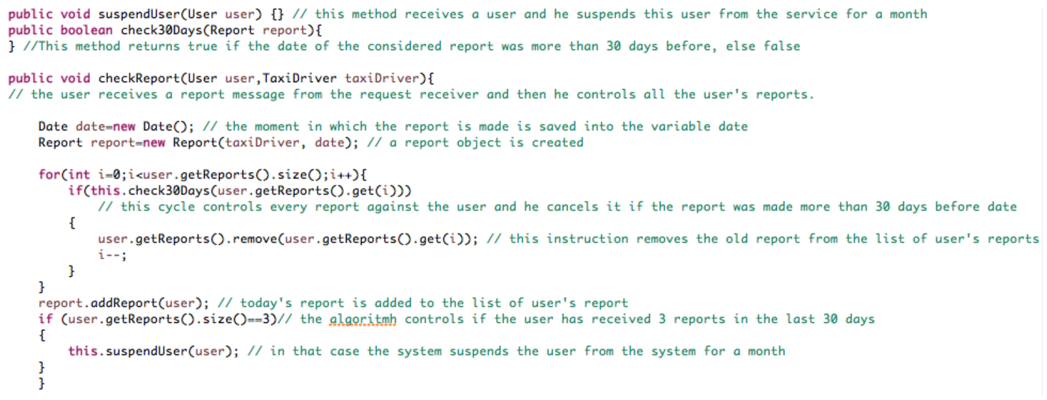
3.3 Report Management Algoritmh

This algoritmh shows what happens after the Report Manager has received a report message from the request receiver sent by a TD. First, a report object is created by taking as attributes the taxiDriver that has sent the report and the date the report has been send. Then the Report Manager controls every report received by the user and he cancels all the Report Points that lasted for at least 30 days. Then he add the report point to the user for the recent report by adding the report to the list of those received by the user. Then the method controls if the number of reports in the last 30 days(the size of the list) is equal to 3. In this case the suspendUser() method is called in order to suspend the user from the service for a month.

Informations on the methods suspendUser() is a method of the Report Manager class.

check30Days() is a method of the Report Manager class.

checkReport() is a method of the Report Manager class.



```
public void suspendUser(User user) {} // this method receives a user and he suspends this user from the service for a month
public boolean check30Days(Report report){
} //This method returns true if the date of the considered report was more than 30 days before, else false

public void checkReport(User user,TaxiDriver taxiDriver){
// the user receives a report message from the request receiver and then he controls all the user's reports.

    Date date=new Date(); // the moment in which the report is made is saved into the variable date
    Report report=new Report(taxiDriver, date); // a report object is created

    for(int i=0;i<user.getReports().size();i++){
        if(this.check30Days(user.getReports().get(i)))
            // this cycle controls every report against the user and he cancels it if the report was made more than 30 days before date
        {
            user.getReports().remove(user.getReports().get(i)); // this instruction removes the old report from the list of user's reports
            i--;
        }
    }
    report.addReport(user); // today's report is added to the list of user's report
    if (user.getReports().size()==3)// the algoritmh controls if the user has received 3 reports in the last 30 days
    {
        this.suspendUser(user); // in that case the system suspends the user from the system for a month
    }
    }
```

4 User Interfaces

No User Interfaces will be described in this document. If you want to check the User interfaces please refer to the RASD 1.0 document(or following versions if available) in the Deliveries folder in our GitLab repository.

5 Requirements traceability

Component	Requirement
Access Manager	[1,1],[1,2] [2,1],[2,3],[2,4],[3,1],[4,1],[5,1], [6,1],[7,1],[8,1],[9,1],[10,1],[11,1]
Request Receiver	[5,3],[6,3],[8,4],[11,4]
Dispatcher	[7,2],[10,3],[11,6]
Queue Manager	[10,3]
Position Manager	[8,2],[9,2],[10,2]
Request Manager	[3,4],[4,3],[4,4],[5,2],[5,3],[6,3],[11,5],[11,2]
Data Manager	[1,1],[1,2],[2,1],[3,1],[4,1],[5,1],[6,1],[7,1],[8,1],[9,1], [10,1],[11,1],[2,3],[2,4],[3,4],[5,2],[4,3],[5,3],[5,4],[11,5],[11,2]
UWAV/UMAV	[1,4],[2,2],[2,4],[5,5],[3,2],[3,3],[4,2][11,3],[11,4]
TDDAV	[2,2],[6,2],[7,2],[7,3],[7,4],[8,4],[8,5],[9,3],[9,4],[9,5]

6 References

6.1 Software Tools Used

- Draw.io (https://www.draw.io): to draw the architecture of our application.
- Word Online (https://office.live.com/start/Word.aspx?omkt=it-IT): to write the paragraphs of this document.
- LyX (http://www.lyx.org/): to redact this document.
- Visual Paradigm (http://www.visual-paradigm.com): to draw Component Diagrams, Deployment Diagrams and Sequence Diagrams.
- Eclipse (https://eclipse.org/): to write algorithms.

6.2 Hours of work

This is the time spent by each developer to redact this document:

- Matteo Rizzi: 27 hours;
- Claudio Scandella: 27 hours.