Politecnico di Milano

A.A. 2015-2016

Software Engineering 2: “myTaxiService”

Design **D**ocument

Davide Azzalini (855185), Fabio Azzalini (855182)

1 Introduction

1.1 Purpose

This document represents the Software Design Document (DD).

The purpose of this document is to describe how the system is organized as a set of communicating components. It provides a description of the design of the system fully enough to allow software development to proceed with an understanding of what has to be built and how it is expected to be. This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects.

While the Requirements Analysis and Specifications Document (RASD) is written for a more general audience, this document is intended for individuals directly involved in the development of myTaxiService. This includes software developers, project consultants, and team managers. This document doesn't need to be read sequentially, users are encouraged to jump to any section they find relevant.

1.2 Scope

The government of a large city aims at optimizing it's taxi service. At this stake myTaxiService will be created, a brand-new application available in two variations: web application and mobile application.

In particular, the attention will be focused on making the access to the service easier from the passengers and on guaranteeing a fairer distribution of the work load among all taxi drivers.

The application will allow passengers to request a ride and taxi drivers to decide weather or not to give the ride. Once a taxi driver has been allocated the passenger will be informed about the waiting time and the code of the incoming taxi.

Furthermore, the application will ensure a more efficient management of the service since each taxi is provided with a GPS tracker and the city is divided in zones, as a result a reduction of waiting times it is expected.

In addition to the standard taxi service (in which the costumer contacts the taxi driver in the exact moment he/she needs the service), will also be possible for the costumers to make reservations for future rides.

Important is also for the system to be developed in such a way that a future addition of some new feature can be made easily and without too much effort.

Moreover, the possibility for payments to be electronic, hence recorded, should help decrease the amount of tax evasion as all transactions can be tracked and accounted for.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- Passenger-user: is a registered passenger that can make a request or a reservation for a taxi ride.
- Driver-user: is a taxi driver registered to the service.
- Administrator: System administrator who is given specific permission for managing and controlling the system.
- Web application: is a client-server software application in which the client, user interface, runs in a web browser.
- Mobile app: is a computer program designed to run on a mobile device such as smartphones and tablet computers.
- Request: is a simple call for a taxi ride.
- Reservation: is a ride booked at least two hours before.
- Requests queue: queue associated to zone, for requests.
- Reservations queue: a unique queue for whole city, for reservations.

1.3.2 Acronyms

- RASD: Requirements Analysis and Specification Document.
- GPS: Global Positional System.
- DBMS: DataBase management system.
- DB: DataBase.
- UML: Unified Modeling Language.

- MVC: Model View Controller.
- CBSE: Component Based Software Engineering.

1.3.3 Abbreviations

- App: application.

1.4 Reference Documents

- Specification Document: myTaxiService Project AA 2015-2016.pdf
- AzzaliniAzzalini_RASD.pdf
- IEEE standard on Design Descriptions.pdf
- IEEE standard on architectural descriptions.pdf
- Template for the Design Document
- The Java EE 7 - Oracle
- Lecture slides

1.5 Overview

This document is essentially structured in six parts:

- Section 1: **Introduction**, gives a scope description and overview of everything included in this DD document. Also, the purpose for this document is described and a list of abbreviations and definitions is provided.
- Section 2: **Architectural Design**, contains all the information about how our architecture looks like. An in-depth description of the components, the way they interact and an analysis of their interfaces is given. Different types of views will be provided in order to have a better understanding of the system. In this section are also explained the architectural styles and the design patterns chosen for the implementation.
- Section 3: **Algorithm Design**, here are outlined suggested solutions to the most critical algorithms.
- Section 4: **User Interface Design**, gives an overview on how the user interfaces of our system will look like. Mockups are provided.
- Section5, **Requirements Traceability**, here are identified those architectural components that are involved in the fulfillment of the various requirements that we have specified. The purpose is to make sure that there is a complete coverage of the requirements with respect to the elements that occur in the design.

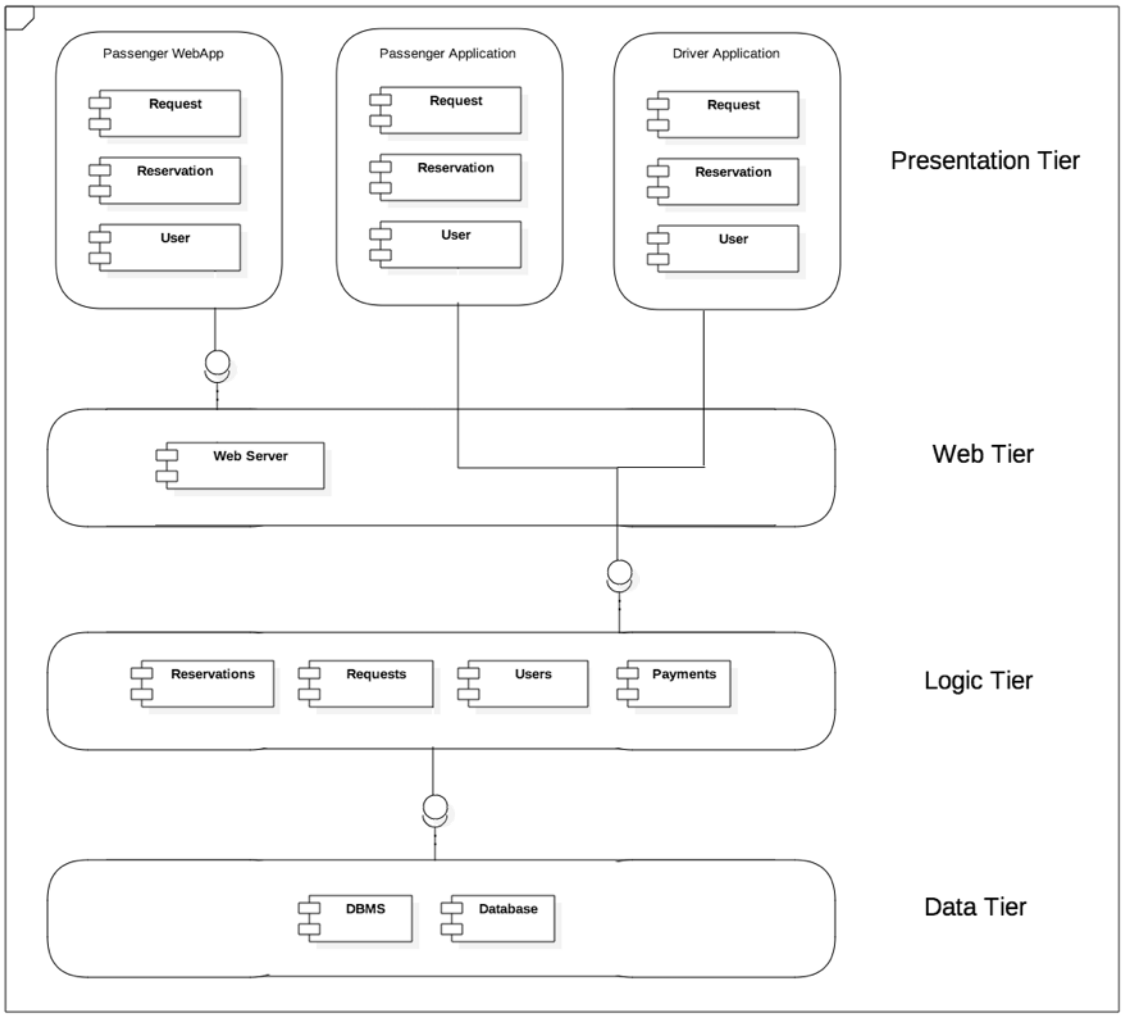
2 Architectural Design

2.1 Overview

- **Section 2.2** provides an high level view in which are identified the main components and the interaction between them.
- **Section 2.3** provides a refinement of what we had in section 2.2: the component view in which are defined the relationship between each components. Here we identify and specify the interfaces of each component.
- **Section 2.4** provides the identification of the artifact that need to be deployed to have the working system.
- **Section 2.5** provides the runtime view that shows how the system works dynamically during the execution and the way components behave in order to accomplish a certain activities.
- **Section 2.6** provides explanations about the architectural styles and design patterns chosen to be implemented.

2.2 High level components and their interaction

The chosen architectural style is a 3-tier structure.



Although the physical architecture is divided into 3 tiers, according to a logical view the system is composed of 4 tiers.

The **Presentation Tier** houses the four different versions of clients. This tier physically consist of the user devices of passengers and drivers.

All those different clients are supposed to be thin clients. This choice has been made according to the fact that the reality under assessment is mainly made up of smartphones and so the processing capabilities are somehow limited and the required volume of interactions between clients and server is very high. Thin clients basically deal only with the showing the user the information received by

the server and to forward the input. All the computation is on the server's shoulders (in this case Logic Tier).

The Passenger App is made up of 3 subsystems:

- Request
- Reservation
- Profile

The Passenger Web Page is made up of 3 subsystems:

- Request
- Reservation
- Profile

The Driver App is made up of 3 subsystems:

- Request
- Reservation
- Profile

An in-depth description of all those subsystems will be given in section 2.3.

The **Web Tier** is a sort of bridge between the Passenger Web Page and the Logic tier. It provides the browser the required web pages.

On the contrary, the two Apps don't need to communicate with the Web Tier since they can "talk" directly with the Logic Tier.

The **Logic Tier** is the "engine" of the system. Here occurs all the computation. The server is supposed to be a fat server since it has to manage a big number of users simultaneously and continuously stream realtime information. This tier also manage all the interactions with the database (Data Tier).

The Server Application is made up of 4 subsystems:

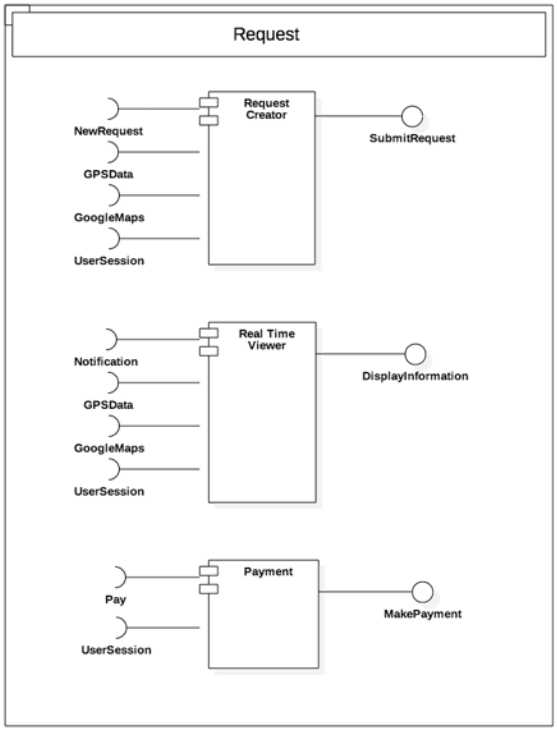
- Requests
- Reservations
- Users
- Payments

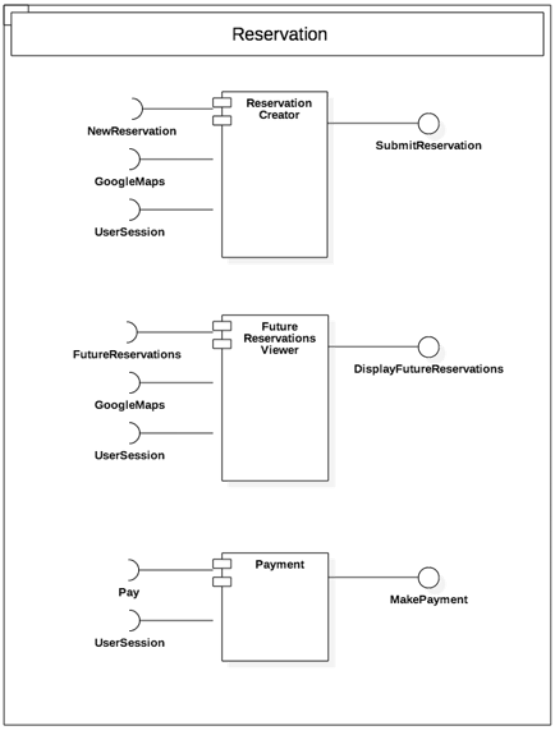
An in-depth description of this subsystem will be given in section 2.3.

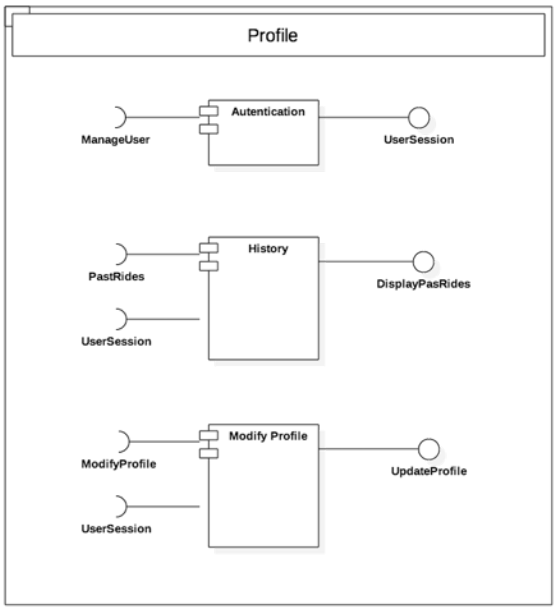
The **Data Tier** accommodates the Database where all the persistent data are stored. The access to data is provided by a DBMS that handles queries and manage security.

2.3 Component view and component interfaces

2.3.1 Client (Passenger App)







2.3.1.1 Request Creator

Subsystem: *Request*.

This component contains all the logic that the client needs to deal with the submission of a new request.

Below a list of its interfaces:

Name	**Type**	**Description**
GPSdata	Required	The Request Creator component requires GPS data in order to allow the user to select the starting point by using the current position.
GoogleMaps	Required	This interface is needed in order to give the user a first general idea about the route. Google Maps is also used to estimate the expected time.
UserSession	Required	Provided by *Authentication* (Client Passenger App). It is possible to use this component only after having successfully performed the log in.
NewRequest	Required	Provided by *Request Allocator* (Server).
SubmitRequest	Provided	This interface allows the user to submit the request.

2.3.1.2 Realtime Viewer

Subsystem: *Request*.

This component contains all the logic that the client needs in order to manage the notification to the user of all the realtime information about the ride.

Below a list of its interfaces:

Name	**Type**	**Description**
GoogleMaps	Required	This interface is needed in order to give the user a rendering of the taxi coming closer and closer and then the, once in the taxi, the current position on the route (GPS data are provided by the server).

UserSession	Required	Provided by *Authentication* (Client Passenger App). It is possible to use this component only after having successfully performed the log in.
Notification	Required	Provided by *Request Manager* (Server).
DisplayInformation	Provided	This interface shows the user all the information.

2.3.1.3 Payment

Subsystem: *Request*.

This component contains all the logic that the client needs in order to manage the payment at the end of a request.

Below a list of its interfaces:

Name	**Type**	**Description**
UserSession	Required	Provided by *Authentication* (Client Passenger App). It is possible to use this component only after having successfully performed the log in.
Pay	Required	Provided by *Payments* (Server).
MakePayment	Provided	This interface allow the user to pay.

2.3.1.4 Reservation Creator

Subsystem: *Reservation*.

This component contains all the logic that the client needs to deal with the submission of a new reservation.

Below a list of its interfaces:

Name	**Type**	**Description**
GoogleMaps	Required	This interface is needed in order to give the user a first general idea about the route. Google Maps is also used to estimate the expected time.

UserSession	Required	Provided by *Authentication* (Client Passenger App). It is possible to use this component only after having successfully performed the log in.
NewReservation	Required	Provided by *Reservation Allocator* (Server).
SubmitReservation	Provided	This interface allows the user to submit the reservation.

2.3.1.5 Future Reservations Viewer

Subsystem: *Reservation*.

This component contains all the logic that the client needs to show the passenger a list of his/her scheduled reservations.

Below a list of its interfaces:

Name	**Type**	**Description**
GoogleMaps	Required	This interface is needed in order to give the user a first general idea about the route. Google Maps is also used to estimate the expected time.
UserSession	Required	Provided by *Authentication* (Client Passenger App). It is possible to use this component only after having successfully performed the log in.
FutureReservations	Required	Provided by *Future Reservations Manager* (Server).
DisplayFutureReservations	Provided	This interface shows the passenger a list of all his/her scheduled reservations for the future.

2.3.1.6 Payment

Subsystem: *Reservation*.

This component contains all the logic that the client needs in order to manage the payment of a reservation.

Below a list of its interfaces:

Name	Type	Description
UserSession	Required	Provided by *Authentication* (Client Passenger App). It is possible to use this component only after having successfully performed the log in.
Pay	Required	Provided by *Payments* (Server).
MakePayment	Provided	This interface allow the user to pay.

2.3.1.7 Authentication

Subsystem: *Profile*.

This component contains all the logic that the client needs to log in the application.

Below a list of its interfaces:

Name	Type	Description
ManageUser	Required	Provided by *Credential Verification* (Server).
UserSession	Provided	This interface is provided to every other component in the Client since each of them require the passenger to be logged in to use it.

2.3.1.8 History

Subsystem: *Profile*.

This component contains all the logic that the client needs to show the user a list of his/her past request and reservations ordered by the most recent one to the less recent one.

Below a list of its interfaces:

Name	Type	Description
UserSession	Required	Provided by *Authentication* (Client Passenger App). It is possible to use this component only after having successfully performed the log in.
PastRides	Required	Provided by *History Manager* (Server).

DisplayPastRides	Provided	This interface shows the passenger a list of his/her past rides.

2.3.1.9 Modify Profile

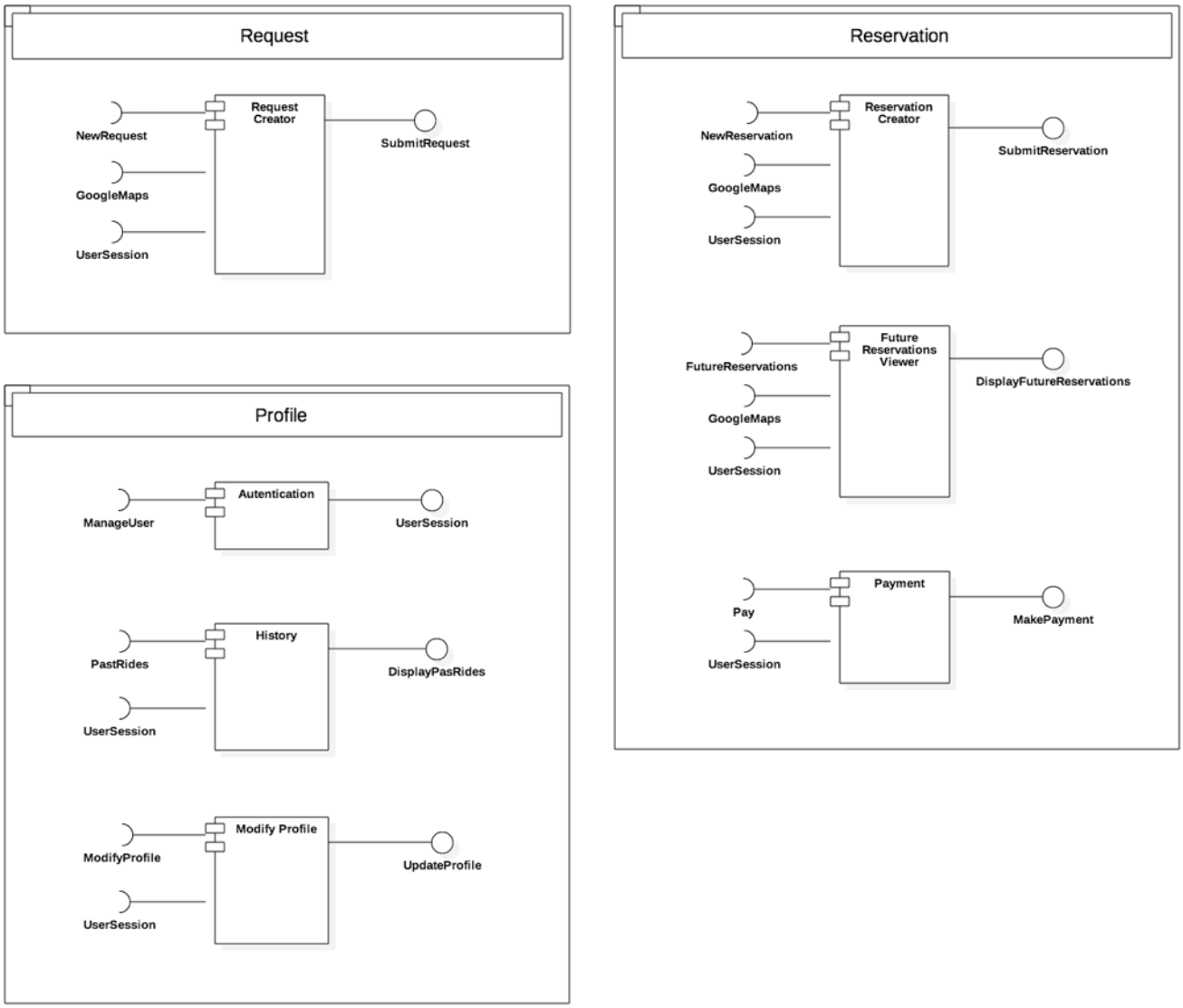
Subsystem: *Profile*.

This component contains all the logic that the client needs to show the user his/her profile information and change them if needed.

Below a list of its interfaces:

Name	**Type**	**Description**
UserSession	Required	Provided by *Authentication* (Client Passenger App). It is possible to use this component only after having successfully performed the log in.
ModifyProfile	Required	Provided by *Account Information Manager* (Server).
Update Profile	Provided	This interface allows the user to update his/her profile information.

2.3.2 Client (Passenger Web Page)



2.3.2.1 Request Creator

Subsystem: *Request*.

This component contains all the logic that the client needs to deal with the submission of a new request.

Below a list of its interfaces:

Name	Type	Description
GoogleMaps	Required	This interface is needed in order to give the user a first general idea about the route. Google Maps is also used to estimate the expected time.
UserSession	Required	Provided by *Authentication* (Client Passenger Web Page). It is possible to use this component only after having successfully performed the log in.
NewRequest	Required	Provided by *Request Allocator* (Server).
SubmitRequest	Provided	This interface allows the user to submit the request.

2.3.2.2 Reservation Creator

Subsystem: *Reservation*.

This component contains all the logic that the client needs to deal with the submission of a new reservation.

Below a list of its interfaces:

Name	Type	Description
GoogleMaps	Required	This interface is needed in order to give the user a first general idea about the route. Google Maps is also used to estimate the expected time.
UserSession	Required	Provided by *Authentication* (Client Passenger Web Page). It is possible to use this component only after having successfully performed the log in.
NewReservation	Required	Provided by *Reservation Allocator* (Server).
SubmitReservation	Provided	This interface allows the user to submit the reservation.

2.3.2.3 Future Reservations Viewer

Subsystem: *Reservation*.

This component contains all the logic that the client needs to show the passenger a list of his/her scheduled reservations.

Below a list of its interfaces:

Name	**Type**	**Description**
GoogleMaps	Required	This interface is needed in order to give the user a first general idea about the route. Google Maps is also used to estimate the expected time.
UserSession	Required	Provided by *Authentication* (Client Passenger Web Page). It is possible to use this component only after having successfully performed the log in.
FutureReservations	Required	Provided by *Future Reservations Manager* (Server).
DisplayFutureReservations	Provided	This interface shows the passenger a list of all his/her scheduled reservations for the future.

2.3.2.4 Payment

Subsystem: *Reservation*.

This component contains all the logic that the client needs in order to manage the payment of a reservation.

Below a list of its interfaces:

Name	**Type**	**Description**
UserSession	Required	Provided by *Authentication* (Client Passenger Web Page). It is possible to use this component only after having successfully performed the log in.
Pay	Required	Provided by *Payments* (Server).
MakePayment	Provided	This interface allow the user to pay.

2.3.2.5 Authentication

Subsystem: *Profile*.

This component contains all the logic that the client needs to log in the application.

Below a list of its interfaces:

Name	Type	Description
ManageUser	Required	Provided by *Credential Verification* (Server).
UserSession	Provided	This interface is provided to every other component in the Client since each of them require the passenger to be logged in to use it.

2.3.2.6 History

Subsystem: *Profile*.

This component contains all the logic that the client needs to show the user a list of his/her past request and reservations ordered by the most recent one to the less recent one.

Below a list of its interfaces:

Name	Type	Description
UserSession	Required	Provided by *Authentication* (Client Passenger Web Page). It is possible to use this component only after having successfully performed the log in.
PastRides	Required	Provided by *History Manager* (Server).
DisplayPastRides	Provided	This interface shows the passenger a list of his/her past rides.

2.3.2.7 Modify Profile

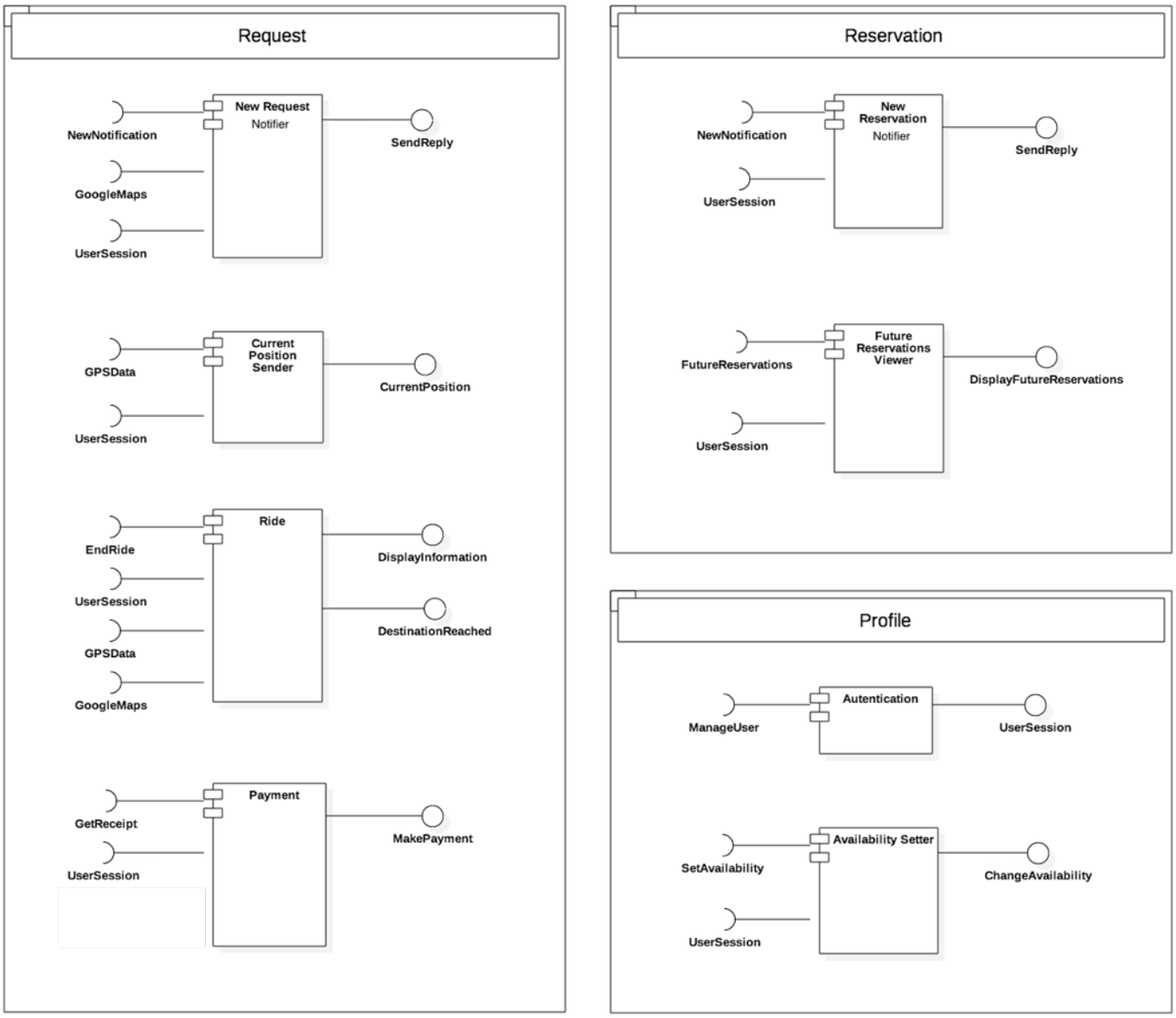
Subsystem: *Profile*.

This component contains all the logic that the client needs to show the user his/her profile information and to allow him/her to change them if needed.

Below a list of its interfaces:

Name	Type	Description
UserSession	Required	Provided by *Authentication* (Client Passenger Web Page). It is possible to use this component only after having successfully performed the log in.
ModifyProfile	Required	Provided by *Account Information Manager* (Server).
Update Profile	Provided	This interface allows the user to update his/her profile information.

2.3.3 Client (Taxi driver App)



2.3.3.1 New Request Notifier

Subsystem: *Request*.

This component contains all the logic that the client needs to deal with the notification of a new request.

Below a list of its interfaces:

Name	Type	Description
GoogleMaps	Required	This interface is needed in order to show a driver where is the passenger and the route to get to him/her. Google Maps is also used to estimate the expected time.
UserSession	Required	Provided by *Authentication* (Client taxi driver App). It is possible to use this component only after having successfully performed the log in.
NewNotification	Required	Provided by *Request Allocator* (Server).
SendReply	Provided	This interface allows the driver to inform the server if he/she is willing or not to take care of that request.

2.3.3.2 Current Position Sender

Subsystem: *Request*.

This component contains all the logic that the client needs to deal with the submission of a new request.

Below a list of its interfaces:

Name	Type	Description
GPSdata	Required	GPS data.
CurrentPosition	Provided	This interface allows the client to constantly inform the server about the current position.

2.3.3.3 Ride

Subsystem: *Request*.

This component contains all the logic that the client needs to deal with an ongoing request.

Below a list of its interfaces:

Name	**Type**	**Description**
GPSdata	Required	The Request Creator component requires GPS data in order to let the user know about the current position of the taxi.
GoogleMaps	Required	This interface is needed in order to show the driver the route.
UserSession	Required	Provided by *Authentication* (Client taxi driver App). It is possible to use this component only after having successfully performed the log in.
DisplayInformation	Provided	This interface allows the driver to see information about the ride on his/her smartphone.
DestinationReached	Provided	This interface allows the driver to end the ride.
EndRide	Required	Provided by *Request Manager* (Server).

2.3.3.4 Payment

Subsystem: *Request*.

This component contains all the logic that the client needs to deal with a payment.

Below a list of its interfaces:

Name	**Type**	**Description**
UserSession	Required	Provided by *Authentication* (Client taxi driver App). It is possible to use this component only after having successfully performed the log in.
PrintReceipt	Provided	This interface allows the driver to print a receipt.
GetReceipt	Required	Provided by *Payments* (Server).

2.3.3.5 New Reservation Notifier

Subsystem: *Reservation*.

This component contains all the logic that the client needs to deal with the notification of a new reservation.

Below a list of its interfaces:

Name	**Type**	**Description**
UserSession	Required	Provided by *Authentication* (Client taxi driver App). It is possible to use this component only after having successfully performed the log in.
NewNotification	Required	Provided by *Reservation Allocator* (Server).
SendReply	Provided	This interface allows the driver to inform the server if he/she is willing or not to take care of that reservation.

2.3.3.6 Future Reservation Viewer

Subsystem: *Reservation*.

This component contains all the logic that the client needs to show the driver a list of his/her scheduled reservations.

Below a list of its interfaces:

Name	**Type**	**Description**
UserSession	Required	Provided by *Authentication* (Client taxi driver App). It is possible to use this component only after having successfully performed the log in.
FutureReservations	Required	Provided by *Future Reservations Manager* (Server).
DisplayFutureReservations	Provided	This interface shows the driver a list of all his/her scheduled reservations for the future.

2.3.3.7 Authentication

Subsystem: *Profile*.

This component contains all the logic that the client needs to log in the application.

Below a list of its interfaces:

Name	**Type**	**Description**
ManageUser	Required	Provided by *Credential Verification* (Server).
UserSession	Provided	This interface is provided to every other component in the Client since each of them require the driver to be logged in to use it.

2.3.3.8 Availability Setter

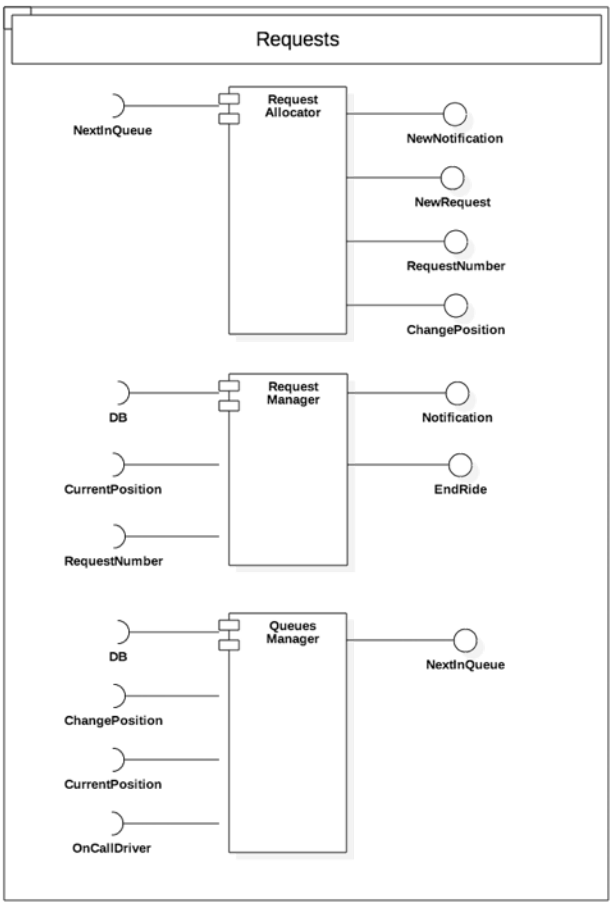
Subsystem: *Profile*.

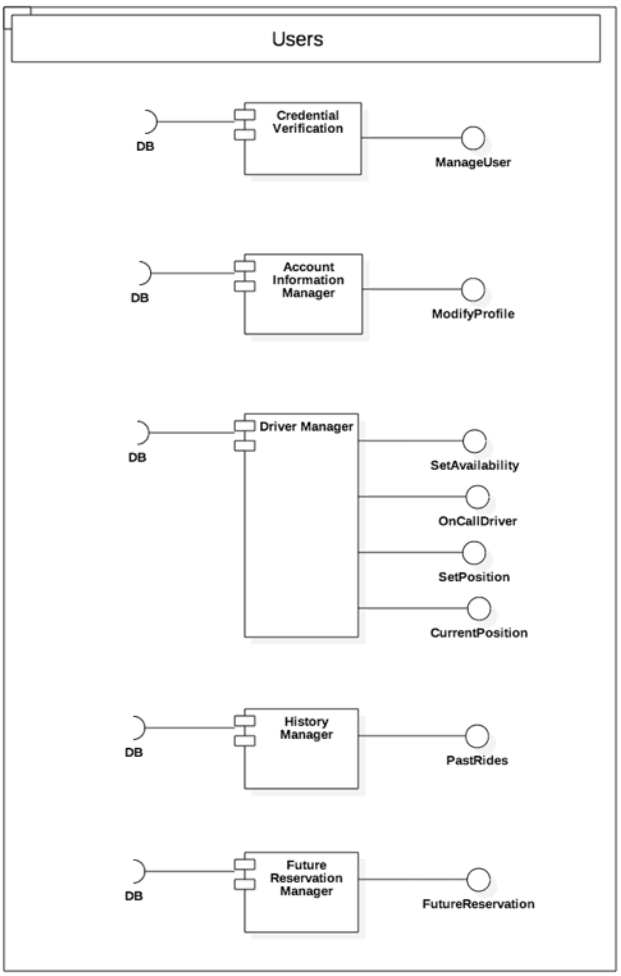
This component contains all the logic that the client needs to inform the system about his/her availability.

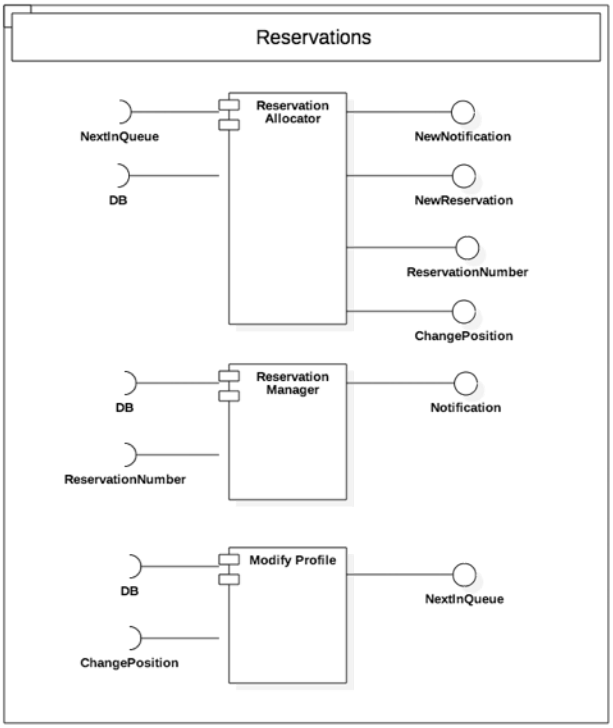
Below a list of its interfaces:

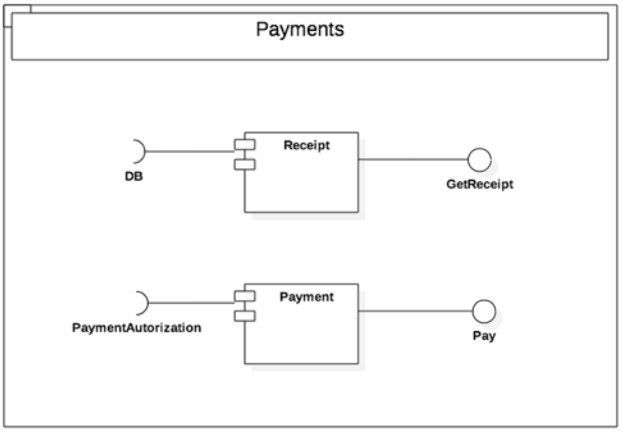
Name	**Type**	**Description**
UserSession	Required	Provided by *Authentication* (Client taxi driver App). It is possible to use this component only after having successfully performed the log in.
SetAvailability	Required	Provided by *Driver Manager* (Server).
ChangeAvailability	Provided	This interface allows the driver to change his/her availability.

2.3.4 Server


Requests
Request Allocator
NextInQueue
NewNotification
NewRequest
RequestNumber
ChangePosition
Request Manager
DB
CurrentPosition
RequestNumber
Notification
EndRide
Queues Manager
DB
ChangePosition
CurrentPosition
OnCallDriver
NextInQueue


Users
Credential Verification
DB
ManageUser
Account Information Manager
DB
ModifyProfile
Driver Manager
DB
SetAvailability
OnCallDriver
SetPosition
CurrentPosition
History Manager
DB
PastRides
Future Reservation Manager
DB
FutureReservation


Reservations
Reservation Allocator
NextInQueue
DB
NewNotification
NewReservation
ReservationNumber
ChangePosition
Reservation Manager
DB
ReservationNumber
Notification
Modify Profile
DB
ChangePosition
NextInQueue


Payments
Receipt
DB
GetReceipt
Payment
PaymentAutorization
Pay

2.3.4.1 Request Allocator

Subsystem: *Requests*.

This component contains all the logic that the server needs to receive the request form the passenger, choose the taxi diver to allocate and inform him/her about the request.

Below a list of its interfaces:

Name	**Type**	**Description**
DB	Required	Access to the Database is needed.
NextInQueue	Required	Provided by *Queue Manager* (Server).
ChangePosition	Provided	This interface allows the server to change the position of the driver in the queue. Either if he/she has accepted the request or not.
NewNotification	Provided	This interface allows the server to inform the driver about a new request.
NewRequest	Provided	This interface allows the client to make a new request.
RequestNumber	Provided	This interface allow the server to create a new instance in the DB with the information about: passenger, driver and route.

2.3.4.2 Request Manager

Subsystem: *Requests*.

This component contains all the logic that the server needs to handle the request once the taxi driver has accepted. This component provide the passenger the realtime information about the ride. It is also used by the driver to inform the server the end of the ride.

Below a list of its interfaces:

Name	**Type**	**Description**

DB	Required	Access to the Database is needed.
CurrentPosition	Required	Provided by *Queue Manager* (Server).
RequestNumber	Required	Provided by *Request Allocator* (Server).
Notification	Provided	This interface allow the server to inform the passenger that a taxi driver has accepted to take care of his/her request.
EndRide	Provided	This interface allow the driver to end the ride once arrived at destination.

2.3.4.3 Queue Manager

Subsystem: *Requests*.

This component contains all the logic that the server needs to handle the queue of each zone of the city.

Below a list of its interfaces:

Name	**Type**	**Description**
DB	Required	Access to the Database is needed.
CurrentPosition	Required	Provided by *Queue Manager* (Server).
OnCallDriver	Required	Provided by *Driver Manager* (Server).
ChangePosition	Required	Provided by *Request Allocator* (Server).
NextInQueue	Provided	This interface allow the server to pick the first driver in the queue of the right zone.

2.3.4.4 Reservation Allocator

Subsystem: *Reservations*.

This component contains all the logic that the server needs to receive the reservation form the passenger, choose the taxi diver to allocate and inform him/her about the reservation.

Below a list of its interfaces:

Name	Type	Description
DB	Required	Access to the Database is needed.
NextInQueue	Required	Provided by *Queue Manager* (Server).
ChangePosition	Provided	This interface allows the server to change the position of the driver in the queue. Either if he/she has accepted the request or not.
NewNotification	Provided	This interface allows the server to inform the driver about a new reservation.
NewReservation	Provided	This interface allows the client to make a new reservation.
ReservationNumber	Provided	This interface allow the server to create a new instance in the DB with the information about: passenger, driver, date, time, origin and destination.

2.3.4.5 Reservation Manager

Subsystem: *Reservations*.

This component contains all the logic that the server needs to handle the reservation once the taxi driver has accepted.

Below a list of its interfaces:

Name	Type	Description
DB	Required	Access to the Database is needed.
ReservationNumber	Required	Provided by *Reservation Allocator* (Server).
Notification	Provided	This interface allow the server to inform the passenger that a taxi driver has accepted to take care of his/her reservation.

2.3.4.6 Queue Manager

Subsystem: *Reservations*.

This component contains all the logic that the server needs to handle the queue of the reservations.

Below a list of its interfaces:

Name	**Type**	**Description**
DB	Required	Access to the Database is needed.
ChangePosition	Required	Provided by *Reservation Allocator* (Server).
NextInQueue	Provided	This interface allow the server to pick the first driver in the queue of the reservations.

2.3.4.7 Credential Verification

Subsystem: *Users*.

This component contains all the logic that the server needs to check the user's credential during the login.

Below a list of its interfaces:

Name	**Type**	**Description**
DB	Required	Access to the Database is needed.
ManageUser	Provided	This interface allow the *Authentication* (Client) component to validate username and password during the login.

2.3.4.8 Account Info Manager

Subsystem: *Users*.

This component contains all the logic that the server needs to handle the account information of the client (both passenger and driver).

Below a list of its interfaces:

Name	Type	Description
DB	Required	Access to the Database is needed.
ModifyProfile	Provided	This interface allow the *Modify Profile* (Client) component to update the personal information modified by the client.

2.3.4.9 Driver Manager

Subsystem: *Users*.

This component contains all the logic that the server needs to handle the availability of the taxi drivers. Gives to the drivers the possibility to set their status and provides this information to all the components in the server that need it.

Below a list of its interfaces:

Name	Type	Description
DB	Required	Access to the Database is needed.
SetPosition	Provided	This interface allow the driver to inform the server about his/her position.
CurrentPosition	Provided	This interface allow the server to inform *Request Manager* (Server) about the current position of the driver taken in to consideration.
SetAvailability	Provided	This interface allow the driver to inform the server about his/her availability.
OnCallDriver	Provided	This interface allow the server to inform *Queues Manager* (Server) about the availability of the drivers.

2.3.4.10 History Manager

Subsystem: *Users*.

This component contains all the logic that the server needs to inform a passenger about his/her past rides (only request).

Below a list of its interfaces:

Name	Type	Description
DB	Required	Access to the Database is needed.
PastRides	Provided	This interface allow the *History* (Client-passenger) component to display all the past rides.

2.3.4.11 Future Reservations

Subsystem: *Users*.

This component contains all the logic that the server needs to inform a client (both passenger and driver) about his/her future reservations scheduled.

Below a list of its interfaces:

Name	Type	Description
DB	Required	Access to the Database is needed.
FutureReservations	Provided	This interface allow the *Future Reservation Viewer* (Client) component to display all the future reservations.

2.3.4.12 Receipt

Subsystem: *Users*.

This component contains all the logic that the server needs to provide the driver the capability to print the receipt.

Below a list of its interfaces:

Name	Type	Description
DB	Required	Access to the Database is needed.
GetReceipt	Provided	This interface allow the *Payment* (Client-driver) component to print the receipt.

2.3.4.13 Payment

Subsystem: *Users*.

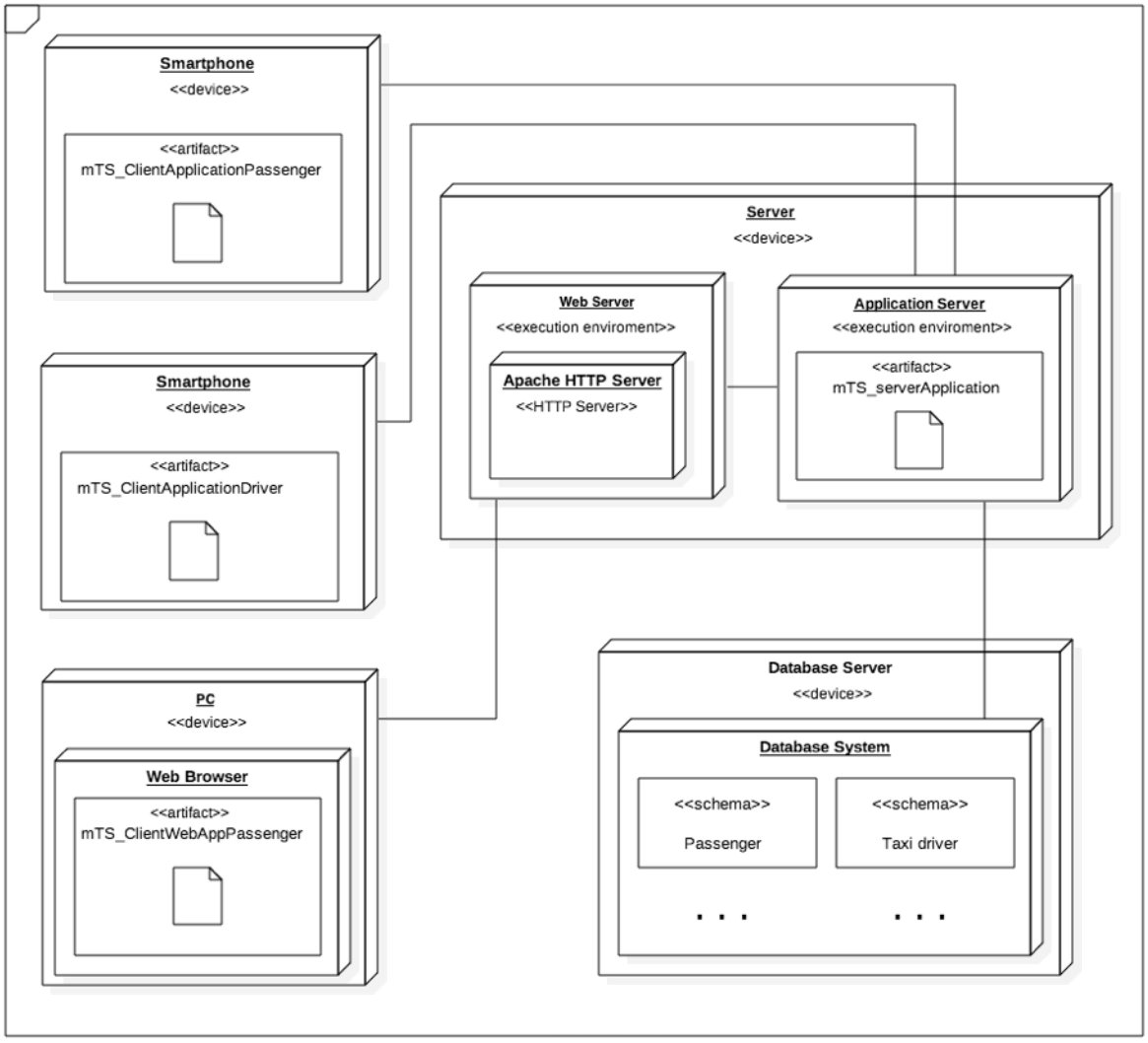
This component contains all the logic that the server needs to provide the client (both driver and passenger) all the service to perform the payment.

Below a list of its interfaces:

Name	Type	Description
PaymentAutorization	Required	Provided by the Bank.
Pay	Provided	This interface allow the *Payment* (Client) component to perform the payment.

2.4 Deployment view

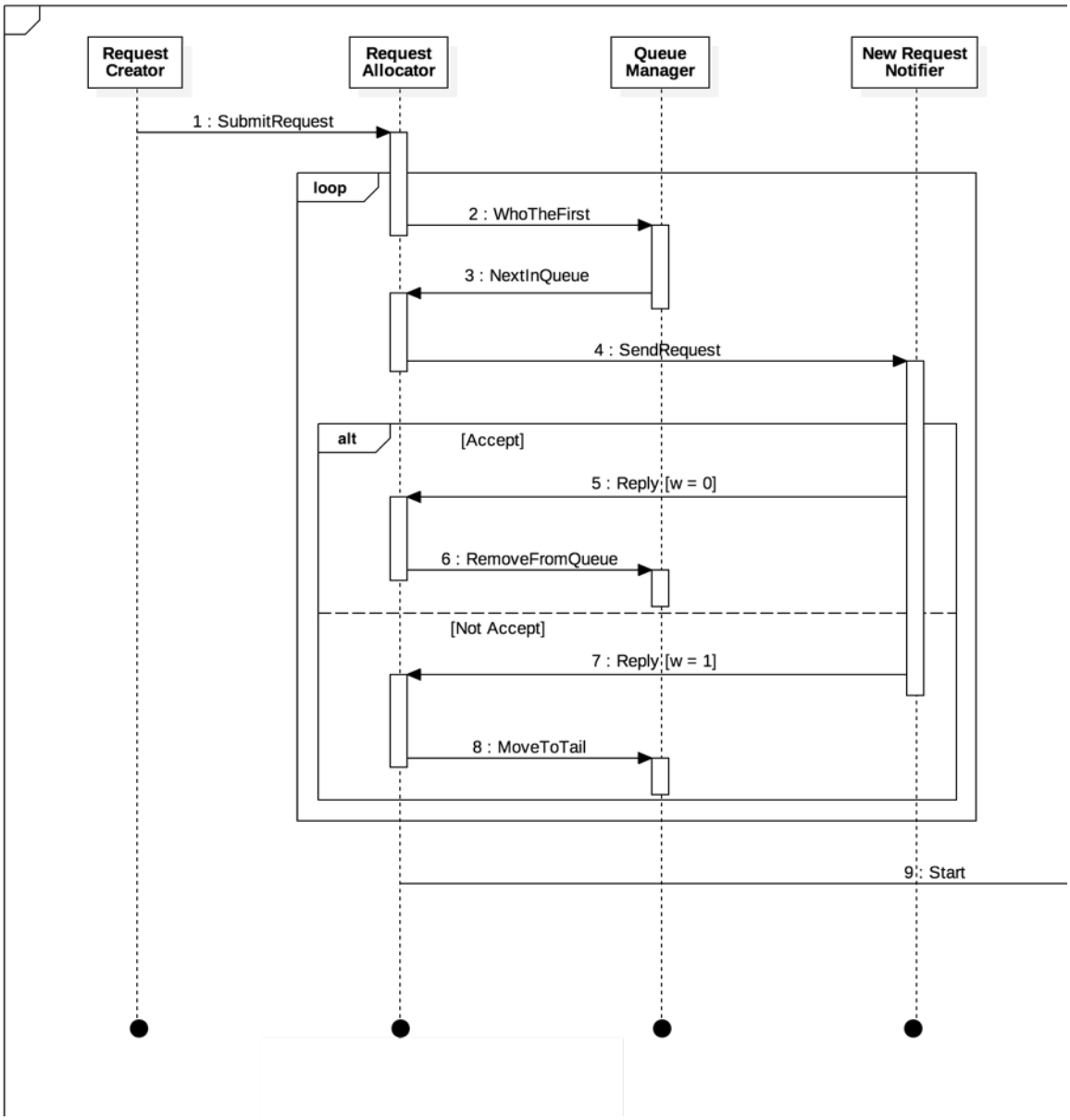
Below the deployment diagram showing how the different logical tiers are distributed among the physical devices.

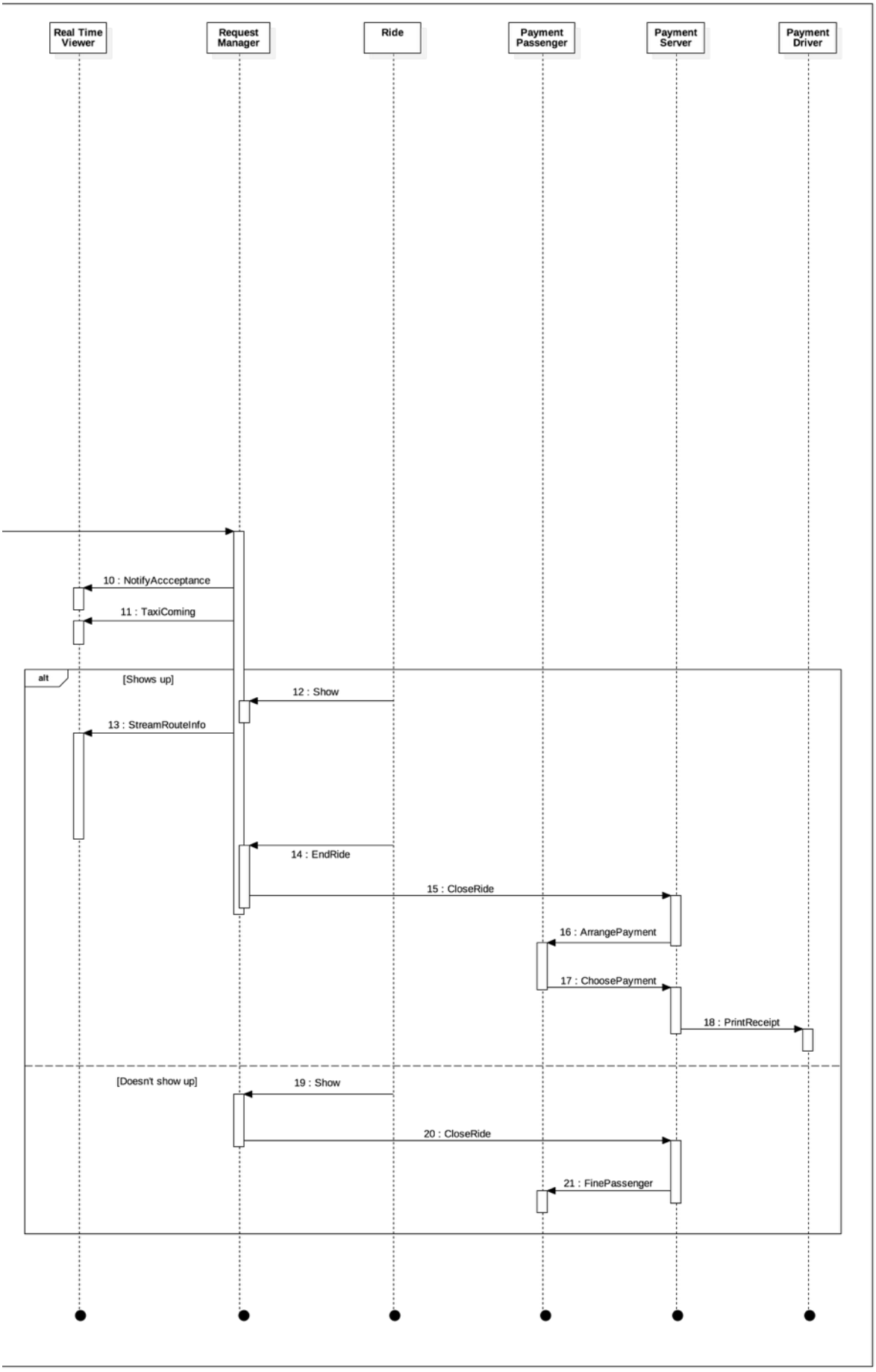


2.5 Runtime view

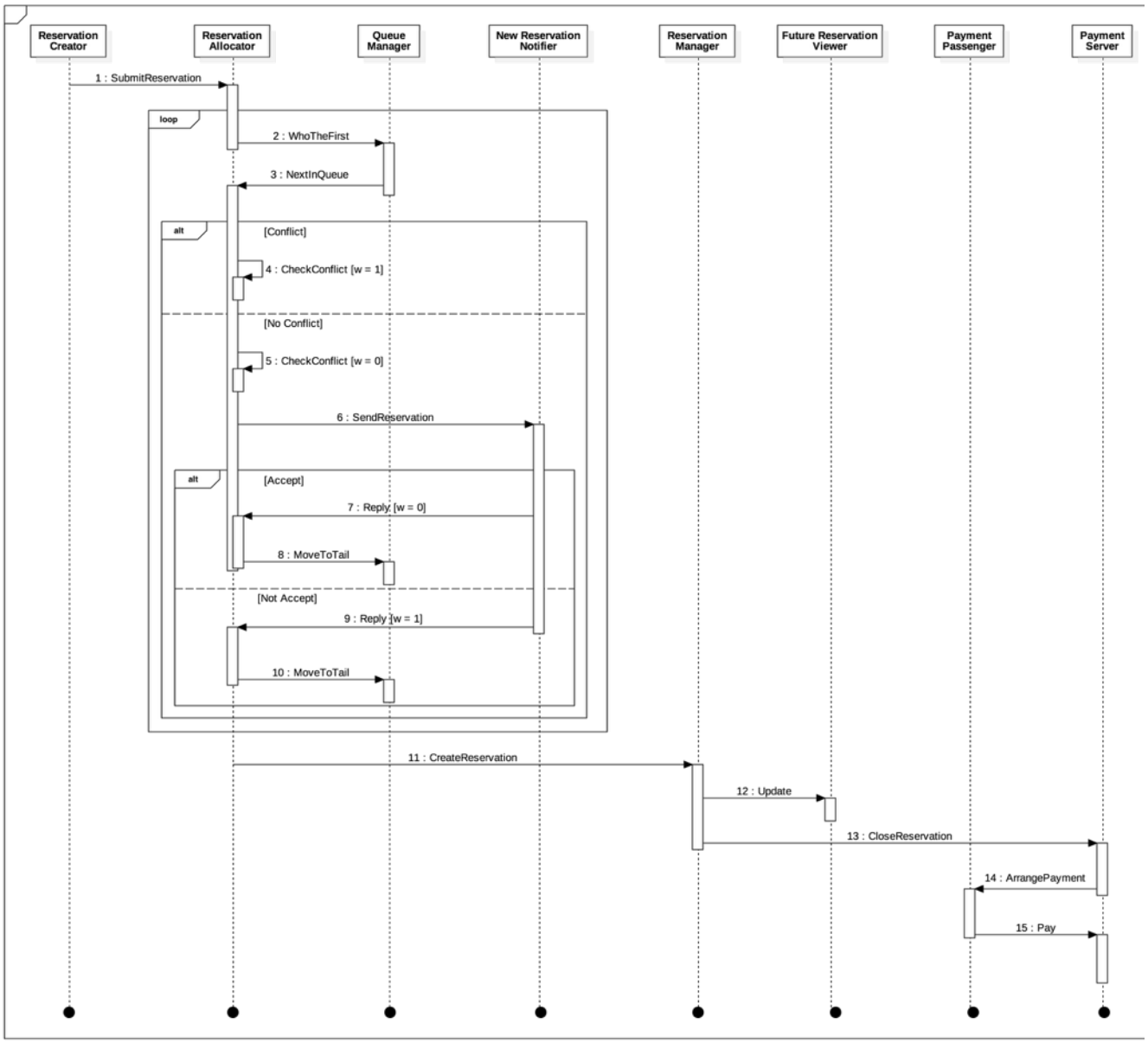
Below are shown the sequence diagrams of a request management and of a reservation management.

Request:




Real Time Viewer
Request Manager
Ride
Payment Passenger
Payment Server
Payment Driver
10 : NotifyAccceptance
11 : TaxiComing
alt
[Shows up]
12 : Show
13 : StreamRouteInfo
14 : EndRide
15 : CloseRide
16 : ArrangePayment
17 : ChoosePayment
18 : PrintReceipt
[Doesn't show up]
19 : Show
20 : CloseRide
21 : FinePassenger

Reservation:



2.6 Selected architectural styles and patterns

In this section it is presented a list of the selected architectural styles and design patterns.

2.6.1 Client/Server

The strongly distributed nature of our system meets in the client/server style its most meaningful representation.

A client/server approach is used in the connection of the Web Application with the Web Server, the Apps with the Application Server, the Web Server with the Application Server and the Application Server with the Database.

2.6.2 MVC

The Model View Controller design pattern is used in the design of all user application. This design pattern consist in the separation of the three components form which take the name from.

The main advantages are:

- Reusability of the software.
- The modification of one logic does not affect other logic.
- Easy to maintain.
- Parallel development is possible and the productivity result increased

2.6.3 Observer

This design pattern is used to handle interaction between the components of the Logic Tier. By the implementation of this design pattern a more efficient handling of notifications and updates is possible.

This is a very useful design pattern in CBSE since ti allows to keep separate and independent role and responsibilities of each component.

2.6.4 Factory Method, Singleton and other design patterns

Other common and useful design pattern, such as Factory method and Singleton, are supposed to be used by the developing team during the implementation.

3 Algorithm design

In this section are suggested two algorithms for the management of a request and a reservation.
The choice of specifying the algorithms in Java programming language is not an implementation constrain but just the way chosen for that purpose in this design document.

3.1 Handling of a request

```
//each request has a unique progressive ID provided by the
method getRequestID()
int i = getRequestID();
Request r = new Request(i, p, origin, destination, nPersons);
//the time interval between the submission and the finding of
a willing taxi driver is called “pending”
r.setStatus(“pending”);
Driver d = null;
boolean driverOK = false;
//looking for a driver
while(!driverOK){
     boolean b = false;
     //here we check that the first driver in the queue has
actually enough time before his/her first next scheduled
reservation to perform this request, otherwise the driver is
removed from the queue and the same check is run on the new
first driver, and so on…
     while(!b){
          d = zone.getQueue().first();
          if(d.enoughTime(origin, destination) == true)
               b = true;
          else
               zone.getQueue().remove(d);
     }
     //once a suitable driver is found the system contact
him/her
     d.sendNotification(origin, destination, nPersons);
     //the driver has one minute to reply
     wait(1000);
     //if the he/she accept the driver is considered found
     if(d.getReply(i) == true){
          zone.getQueue().remove(d);
          driverOK = true;
     }
     //otherwise this driver is pushed to the tail of the
queue and the research for a candidate restart
     else
```

```
          zone.getQueue().moveToTail(d);
}
r.setDriver(d);
//the time interval in which the driver reaches the passenger
is called “accepted”
r.setStatus(“accepted”);
d.setAvailability(false);
//the passenger is notified with the information about the
driver and the estimated time
p.notifyInfo(d);
//as soon as the driver reaches the starting point he/she
informs the system if the passenger showed up or not. If the
passenger has respected his/her commitment the status becomes
“origin”, otherwise “noPassenger”
while(!r.getStatus.equals(“origin”) || !
r.getStatus.equals(“noPassenger”)){
          //during the time needed for the driver to reach
the passenger realtime information are sent to the passenger
          p.sendRealtimeInfo(r);
}
//if the passenger showed up
if(!r.getStatus.equals(“noPassenger”)){
     //the status changes to “traveling”
     r.setStatus(“traveling”);
     while(!r.getStatus.equals(“destination”)){
          //during the time needed for the taxi to reach the
destination realtime information are sent to the passenger
          p.sendRealtimeInfo(r);
     }
     //the following method takes care of the whole payment
procedure
     r.arrangePayment();
     //as soon as the passenger pays the status becomes
“paid” and the ride is considered to be ended
     //the driver is moved to the tail of the queue of the
zone in which the ride has ended

getZone(d.getCurrentPosition()).getQueue().moveToTail(d);
}
//if the passenger didn’t show up
else{
     //the passenger is fined
     p.finePassenger();
     //and moved to the tail of the queue (even if it’s not
his/her fault)
     zone.getQueue().moveToTail(d);
}
```

3.2 Handling of a reservation

```
//each reservation has a unique progressive ID provided by
the method getReservationID()
int i = getReservationID();
Reservation r = new Request(i, p, date, time, origin,
destination, nPersons);
//the time interval between the submission and the finding of
a willing taxi driver is called “pending”
r.setStatus(“pending”);
Driver d = null;
boolean driverOK = false;
//looking for a driver
while(!driverOK){
     boolean b = false;
     //here we check that this reservation actually fits
among the already scheduled reservations of the first driver
in the queue , otherwise the driver maintains his/her
position in the queue and the same check is run on the second
driver, and so on…
     for(int x = 1; x <= reservationsQueue.length() && !b; x+
+){
          d = reservationsQueue.getDriver(x);
          if(d.enoughTime(date, time, origin, destination) ==
true){
               reservationsQueue.moveToTail(d);
               b = true;
          }
     }
     //once a suitable driver is found the system contact
him/her
     d.sendNotiication(date, time, origin, destination,
nPersons);
     //the driver has five minute to reply
     wait(5000);
     //if the he/she accept the driver is considered found
     if(d.getReply(i) == true){
          reservationsQueue.moveToTail(d);
          driverOK = true;
     }
     //otherwise that driver is pushed to the tail of the
queue and the research for a candidate restart
     else
          reservationsQueue.moveToTail(d);
}
r.setDriver(d);
//the status becomes “accepted”
r.setStatus(“accepted”);
//the following method takes care of the whole payment
procedure
```

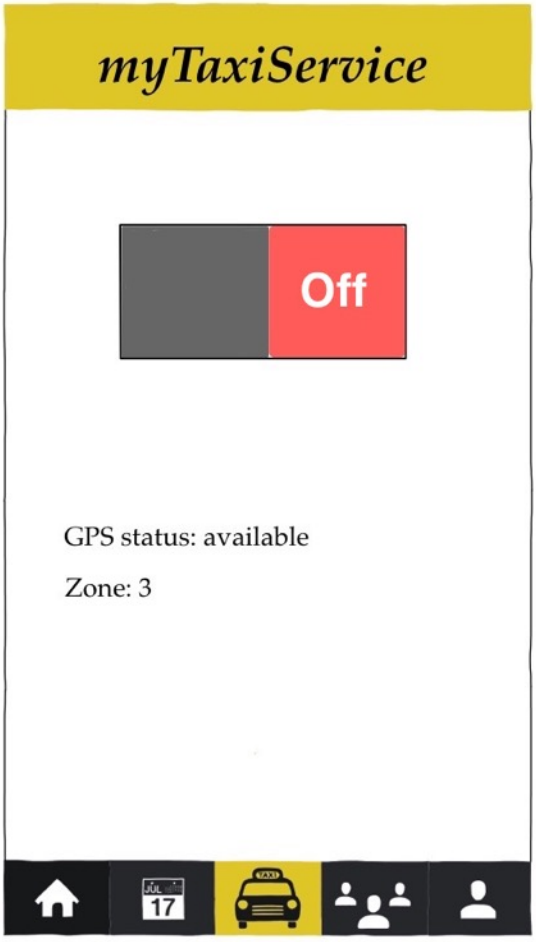
```
r.arrangePayment();
//as soon as the passenger pays the status becomes “paid” and
the reservation is considered to be successfully booked
```

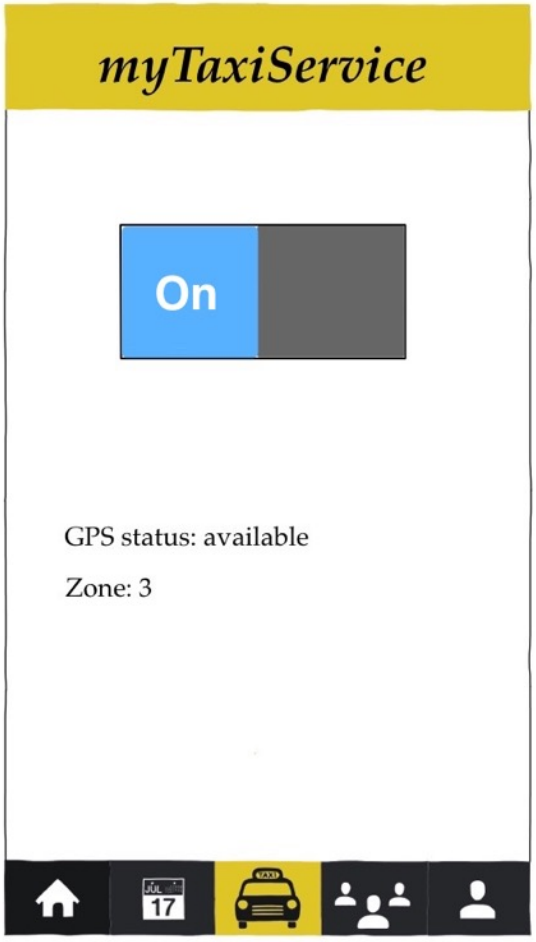
4 User Interface design

In this section are presented the mockups of the Driver App that weren't included in the RASD.

4.1 Home

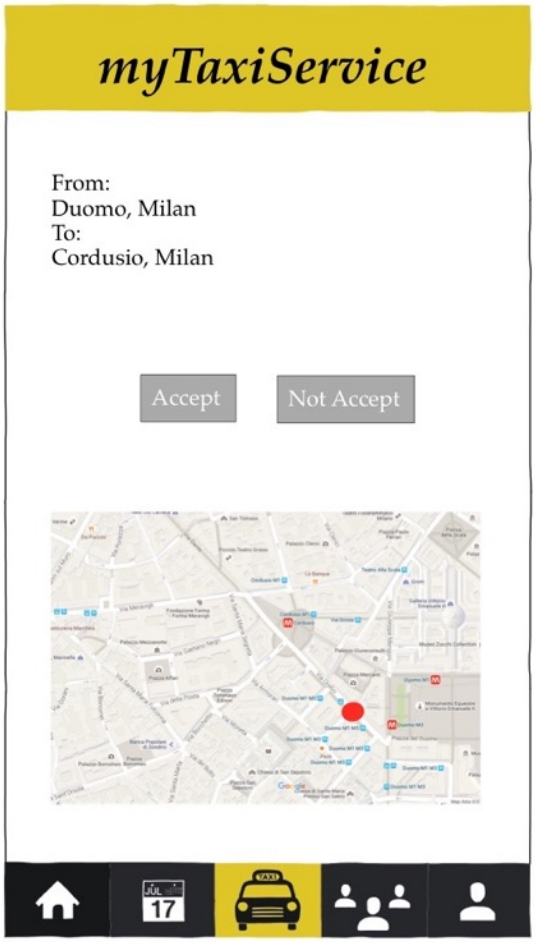
In this page it is possible to set the availability:





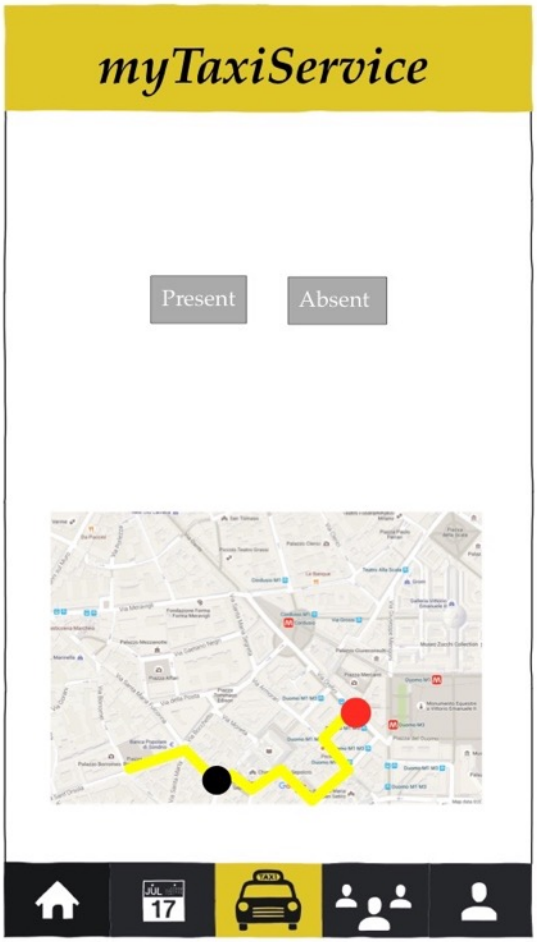
4.2 Acceptance of a request

In this page it is possible to accept a request. Information about the request are shown



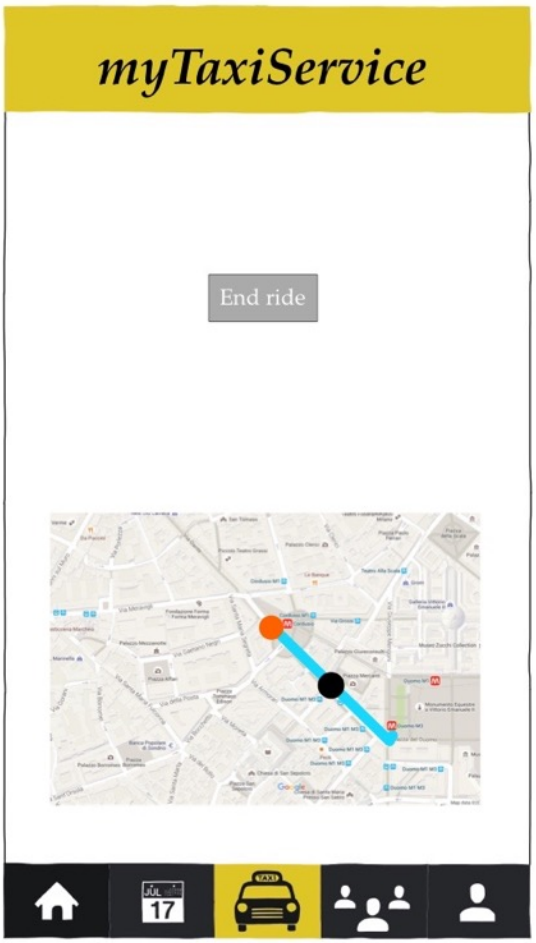
4.3 Reaching the passenger

In this page it is shown the route to reach the passenger. Once in the pick-up location it is possible to inform the system if the passenger is present or not.



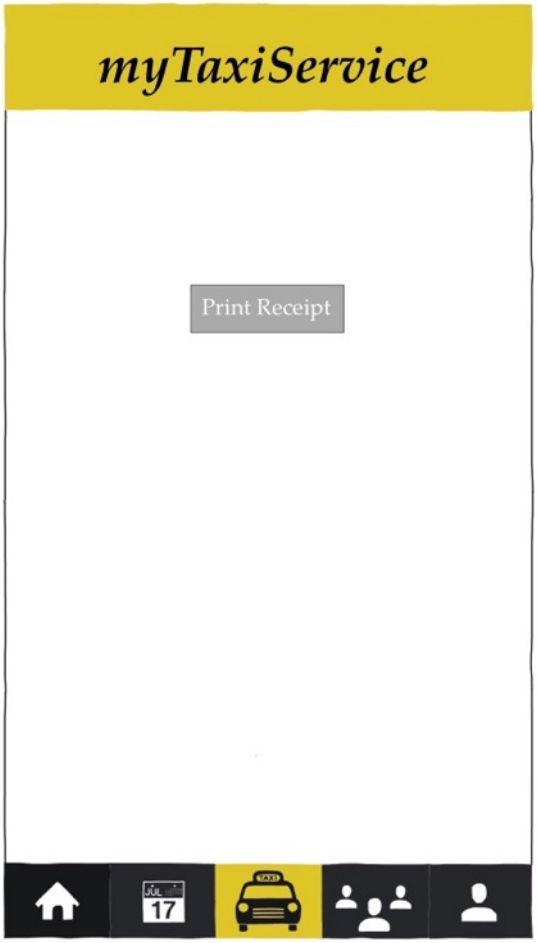
4.4 Travel

In this page it is shown the route to the destination. Once the destination has been reached it is possible to inform the system about that.



4.5 Receipt

In this page it is possible to print the receipt.

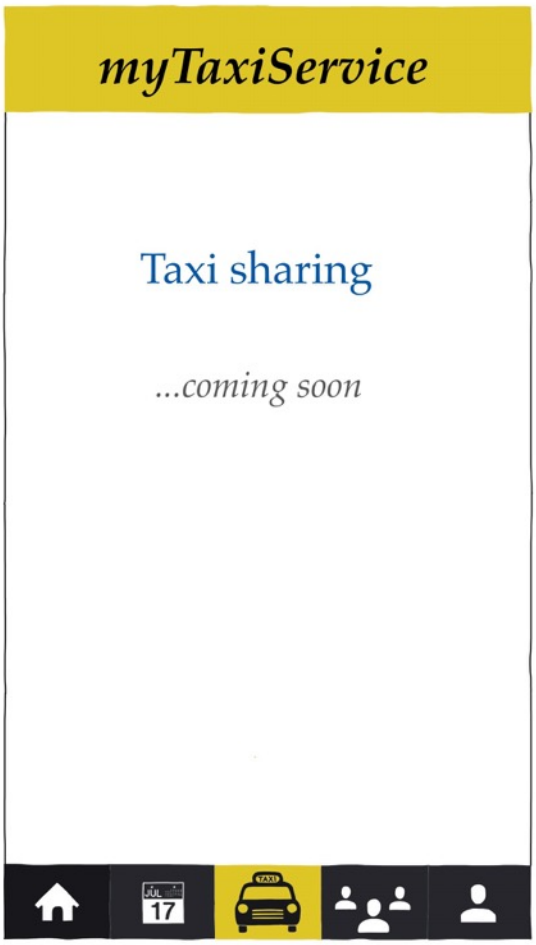


4.6 Reservations

In this page it is possible to accept or decline a reservation and to view the reservations scheduled for the future.

4.7 Taxi sharing

This feature will be implemented in the second release of the application.



5 Requirements traceability

Since during the identification of the components we have started from the Functional Requirements all of those can be considered to be met.

6 References

Tools used:
- StarUML
- Pages

7 Appendix

Hours of work:
- Azzalini Davide 50
- Azzalini Fabio 50