POLITECNICO
MILANO 1863

Software Engineering 2 A.A. 2015-16

Software Design Document

for

MyTaxiService

by Francesco Picciotti (matr. 854021)
and Marco Travaglini (matr. 859186)

Table of Contents

Revision History

Date	Reason For Changes	Version
3/12/2015	*First submission of the document.*	*1.0*

1. Introduction

1.1 Purpose

This Software Design Document is made with the purpose of outlining the software architecture and design of the MyTaxiService in detail. The document will provide developers a general insight to satisfy client's needs efficiently and effectively.
Moreover the document facilitates communication and understanding of the system by providing several views of the system design.

1.2 Product Scope

The Software Design Document would demonstrate how the design will accomplish the functional and non- functional requirements captured in the Requirements Analysis and Specification Document (RASD). The document will provide a framework to the programmers through describing the high level components and architecture, sub-systems, interfaces and algorithm design. This is achieved through the use of architectural patterns, design patterns, sequence diagrams, component diagram, deployment diagram and user interfaces.

1.3 Definition, acronyms, abbreviations

- ***Request**: It indicates either ride or reservation.*
- ***Active**: This word is referred to a request, it means the request is not already finished by the driver.*
- ***Terminable**: This is a specification of an active request, if a request is terminable it states that the request is active but can be terminated. In fact in the case of a Reservation, it can be ended only later a minimum time (for example the minimum time it is the estimated time from the origin to the destination).*

1.4 Reference documents

- *Requirements Analysis Specification Document : MyTaxiService AA 2015/16.pdf*
- *IEEE St 1016 Software Design Description*
- *DD MeteoCal example.pdf*

1.5 Document Structure

I. ***Introduction**: This section deals with general description and information about the document and its content.*
II. ***Architectural Design**: In this paragraph are shown the architectural description of the software, providing details such as how the components belonging to the system, their interaction and the chosen patterns.*
III. ***Algorithm Design**: The document part in which are described the main algorithms with UML Activity Diagrams.*

IV. ***User Interface Design**: In this section are shown the software's graphic user interfaces through mockups, in this document are attached only those concerning the mobile app because in the RASD document submitted before there were only web app mockups.*

V. ***Requirements Traceability**: Basically it points out for each requirement which component ensures it.*

2. Architectural Design

2.1 Overview

In the following section it is described how the software will made of, which components will belong to it, how they will interact during the usage and what they will be exploited by the system working, even by UML diagrams such that Component, Deployment and Sequence ones. Moreover about the system there is defined either the hardware and software parts, like the chosen architectural style, the design and architectural patterns used.

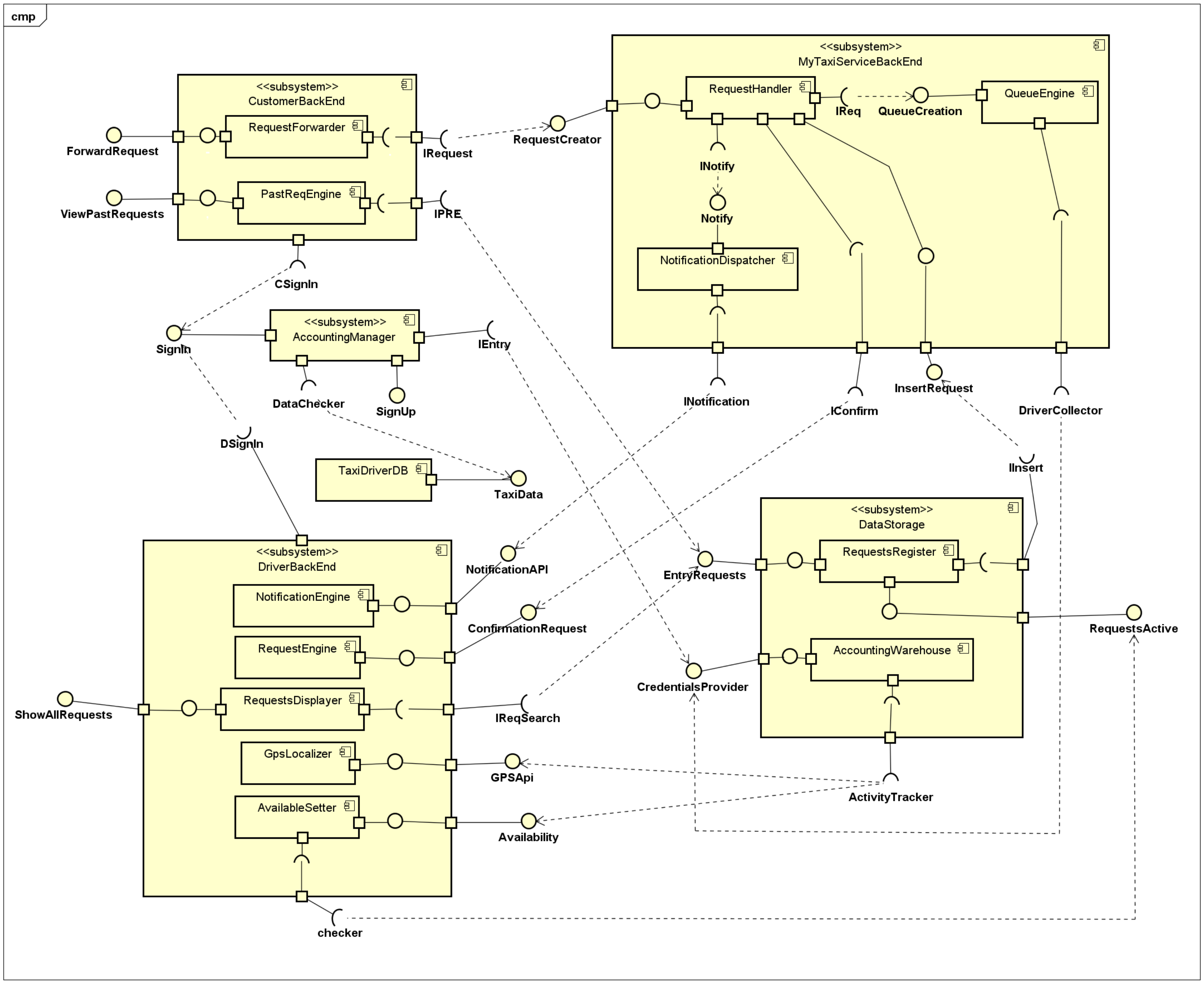
2.2 High level components and their interaction

The mainly components, in particular sub-systems, belonging to the software are the following:

- ***CustomerBackEnd**: It manages the customer's account logic, allowing him to create and forward requests, ride or reservation both, and view the past requests.*
- ***DriverBackEnd**: It contains all the logic concerning the driver's functionalities that he has got, like the possibility to confirm a reservation or a ride request, view all the requests accepted, set his availability in order to let the application forward requests and the application's permission to exploit the driver's GPS Localizer to retrieve the actual position, thus the current area to which he belongs, from it.*
- ***AccountingManager**: It takes care of the main functionalities which a guest is able to ask to the software, that are the sign up and the sign in, thus letting the guest to create an account or to login onto the service.*
- ***TaxiDriverDB**: It is an external database belonging to the city's taxi company to which the system needs to have the permission to exploit it. In fact the accounting manager has an entry on it every time that has to check the validity of the given data during the sign up procedure about the Taxi Licence and the Taxi Number.*
- ***MyTaxiServiceBackEnd**: It contains all the whole application's logic which deals with the requests management from the begin to the end, the queue used in every request in order to find out a driver who takes care of the ride or reservation request and a notification part of the system which arranges the dispatching of all the notification to a certain driver.*
- ***DataStorage**: It contains all the persistent data of the software which are about the users' data, like credentials, GPS position, availability, personal details, and all the requests done in the whole usage of the system.*

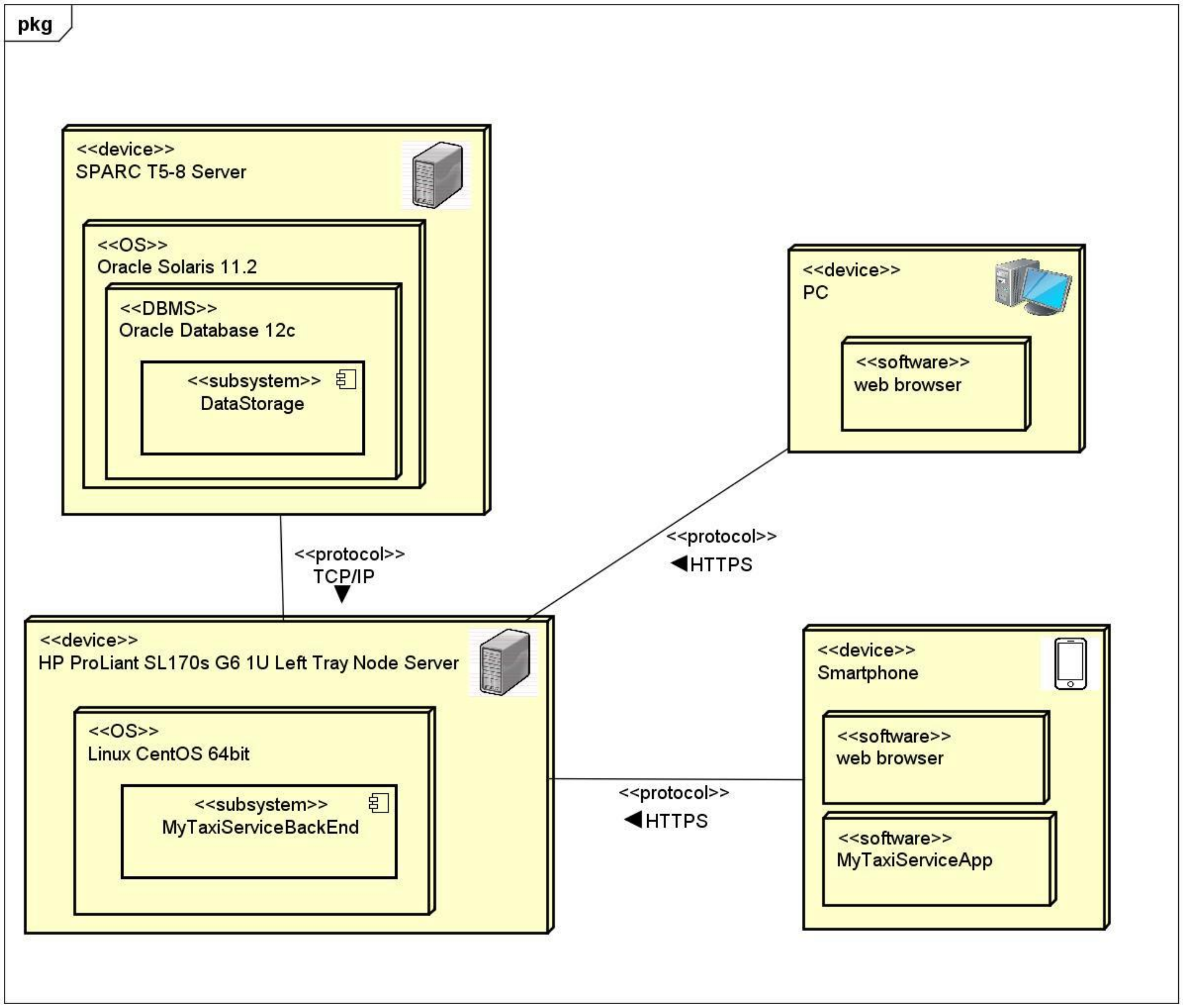
2.3 Component view

Here it's shown the whole UML component diagram describing each component and their sub-components belonging, the provided and required interfaces exposed.



2.4 Deployment view

This Deployment Diagram shows what hardware components compose the system, what software components run on each device and how the different parts are connected.

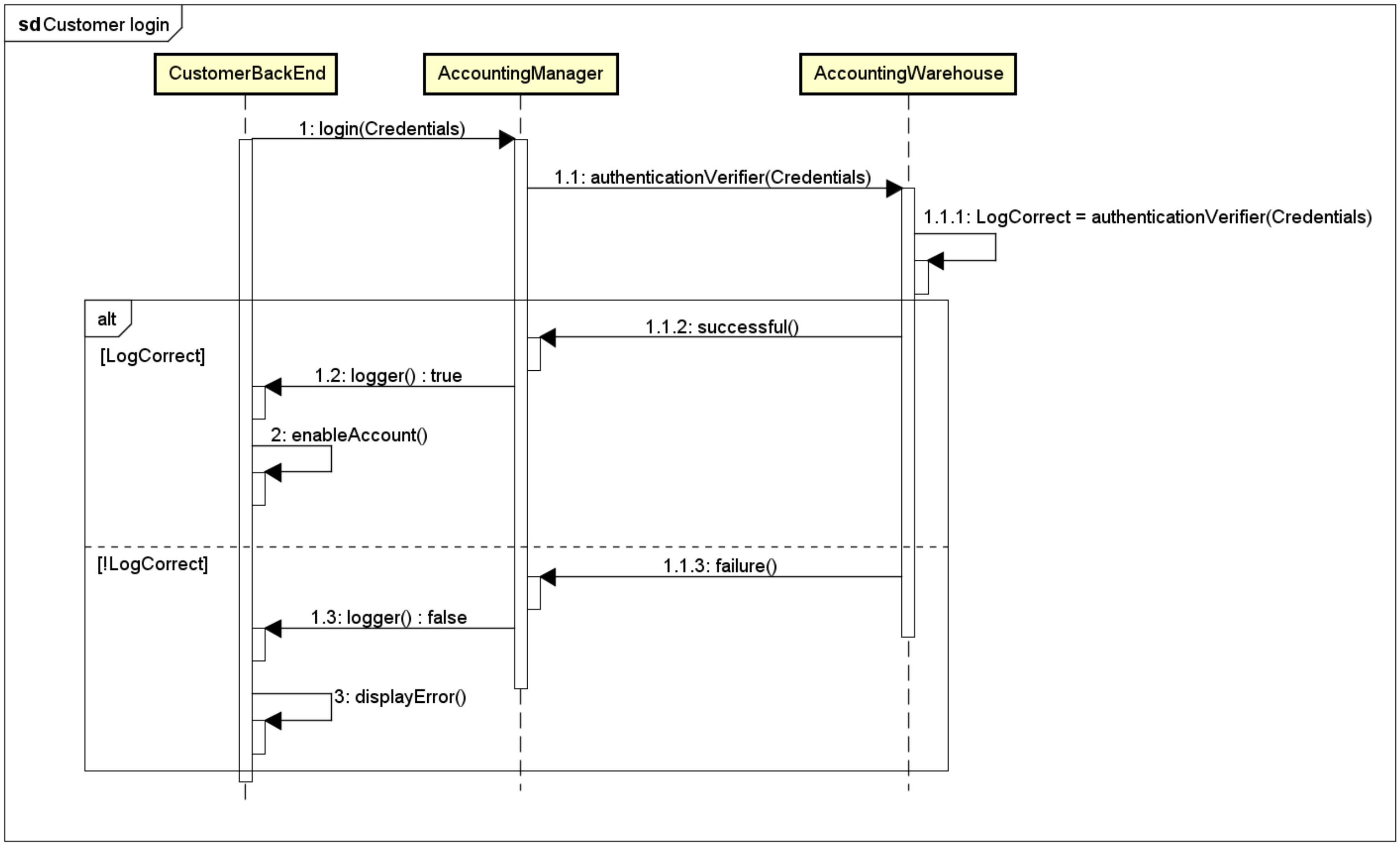


2.5 Runtime view

In this section there are shown the interaction between several components during the mainly system's functionalities through the representation with UML Sequence Diagrams for each usage.

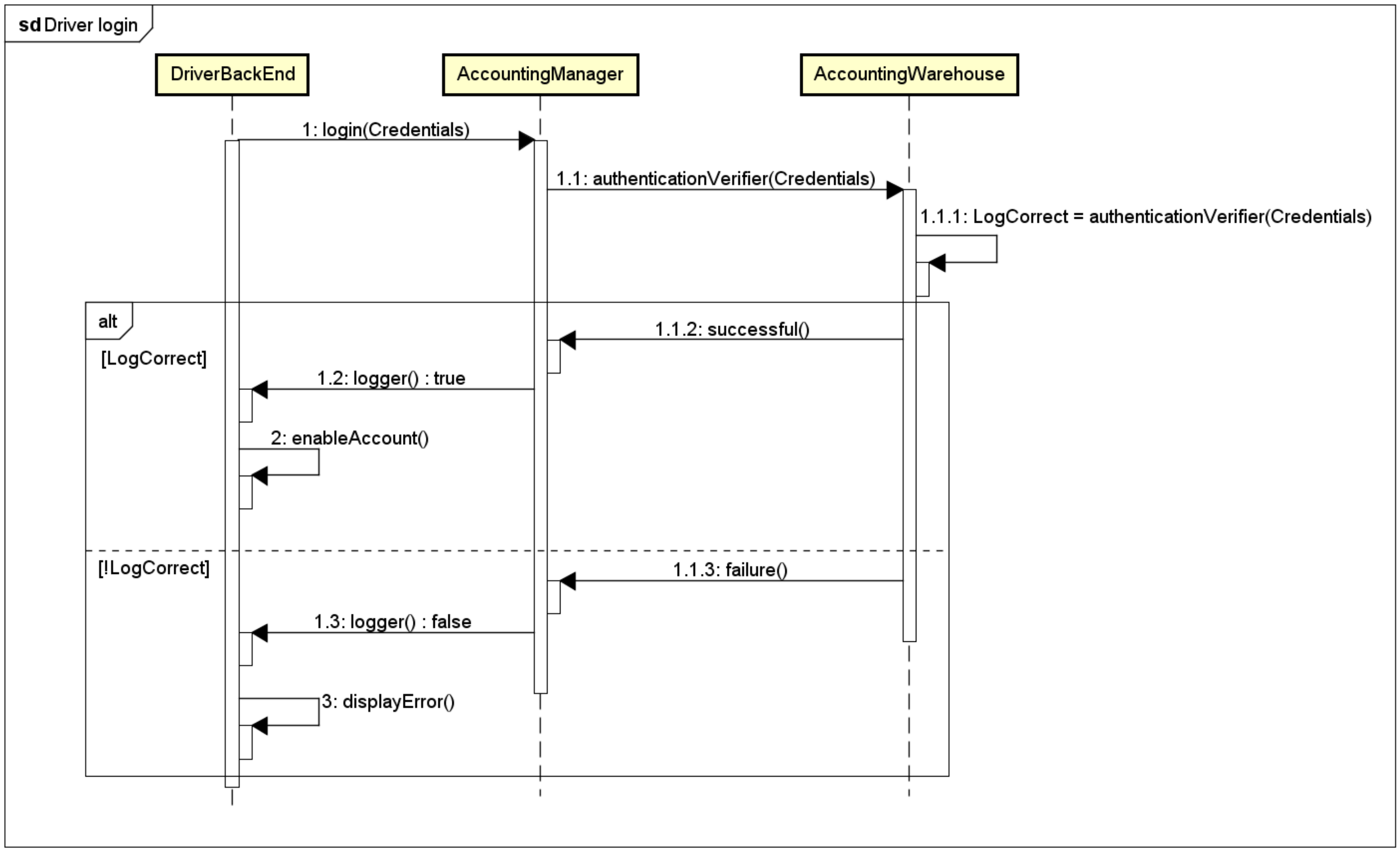
2.5.1 Customer login

On the following diagram is shown how the customer's login works and what components are involved.



2.5.2 Driver login

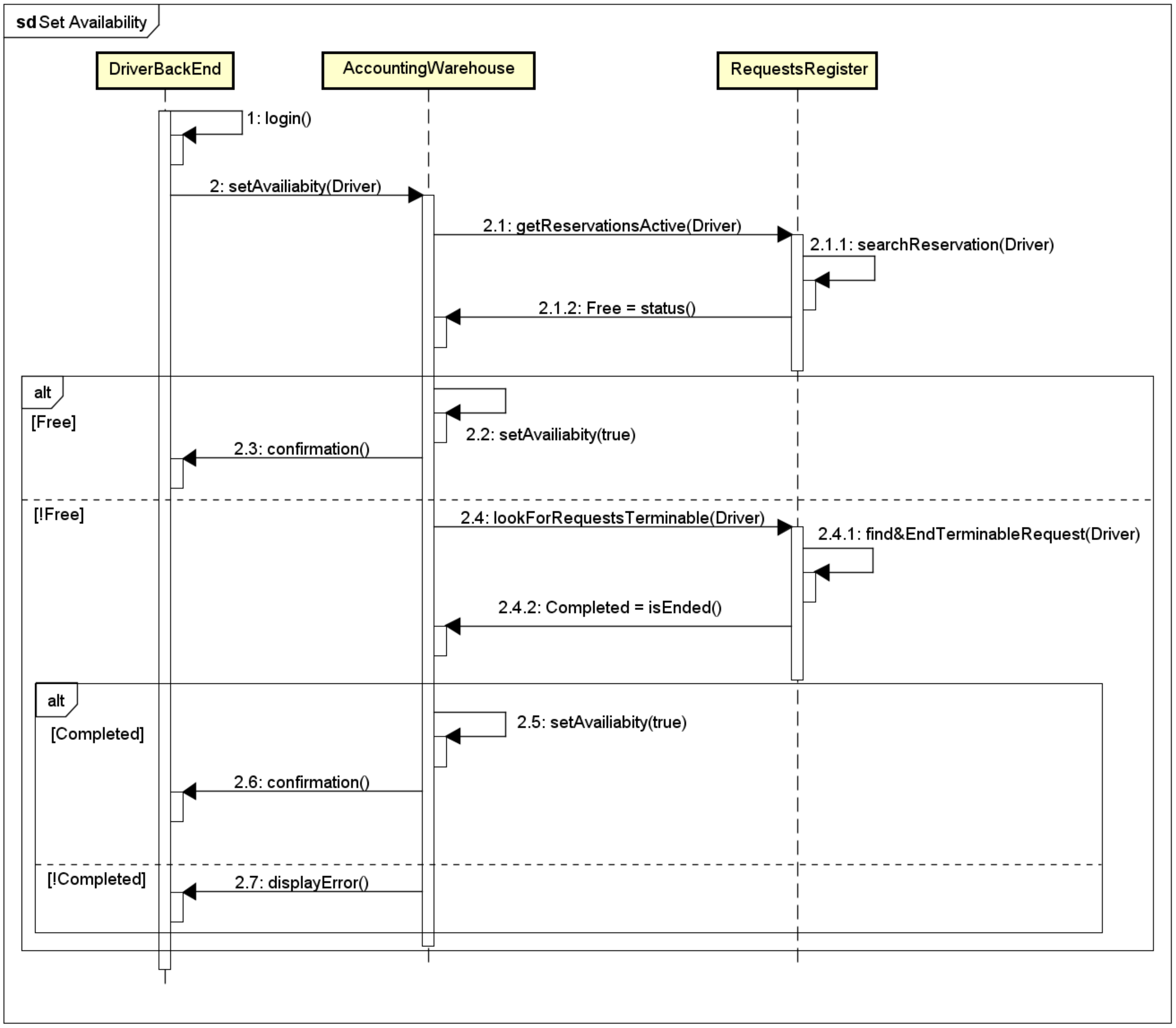
Here it is explained how the Driver can sign in and what components are interested.



2.5.3 Set availability

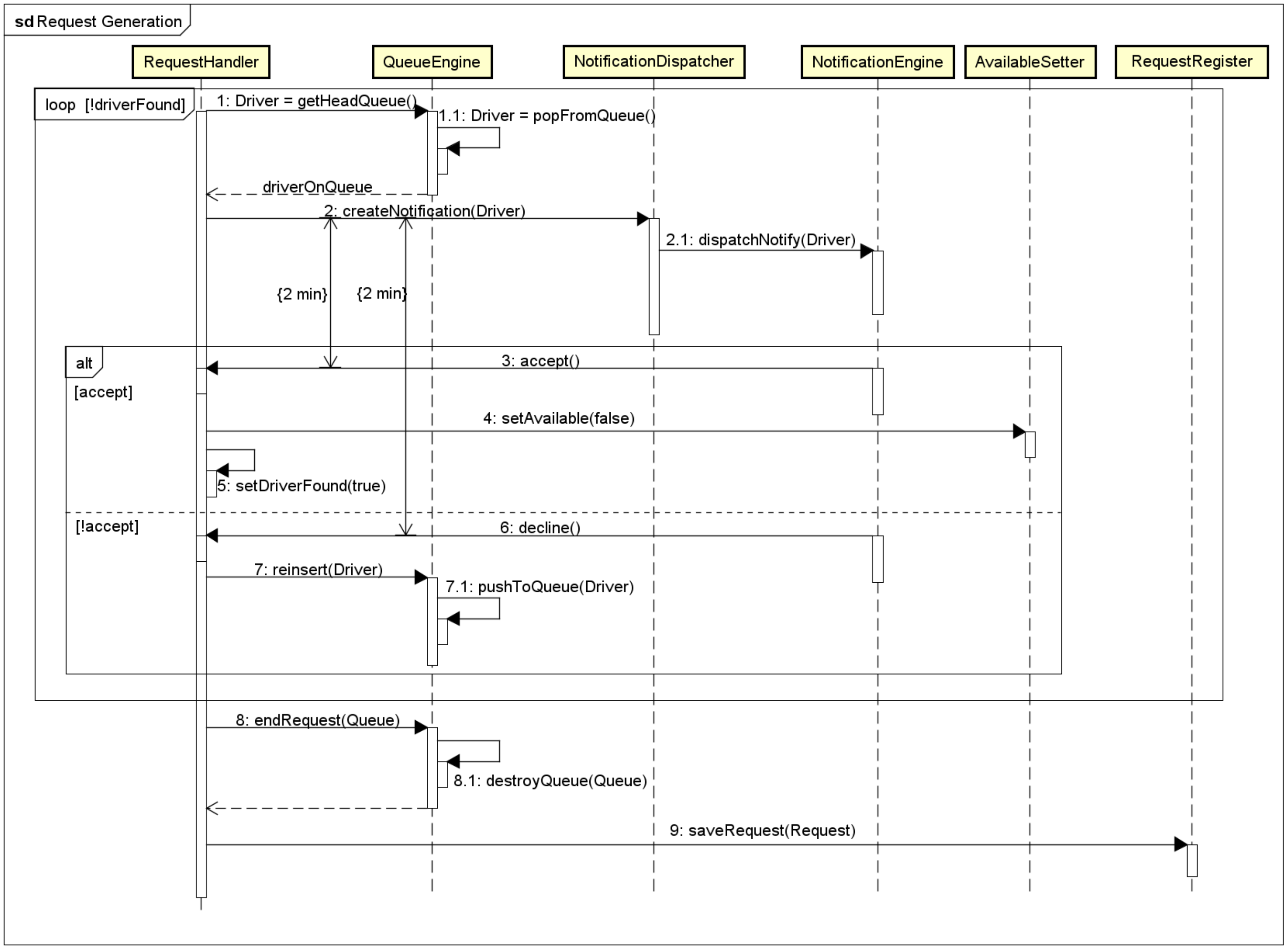
In the following Sequence Diagram is explained how the driver's availability is set. For a more readable but correct diagram (and the same in other diagrams), the login process which is necessary to update the driver's availability is only stated without explicating how it is done because of it is widely explained on the previous diagram. When a driver indicates to set himself to be available, the system first checks whether he has got an active request linked to him:

- *If not, the software confirms the correct and valid updating*
- *Otherwise, it checks if the active request is terminable; if it is the system ends that request and updates the availability status. If not, it displays an error to the driver stating the required update is not possible.*


sd Set Availability
DriverBackEnd
AccountingWarehouse
RequestsRegister
1: login()
2: setAvailiabity(Driver)
2.1: getReservationsActive(Driver)
2.1.1: searchReservation(Driver)
2.1.2: Free = status()
alt
[Free]
2.2: setAvailiabity(true)
2.3: confirmation()
[!Free]
2.4: lookForRequestsTerminable(Driver)
2.4.1: find&EndTerminableRequest(Driver)
2.4.2: Completed = isEnded()
alt
[Completed]
2.5: setAvailiabity(true)
2.6: confirmation()
[!Completed]
2.7: displayError()

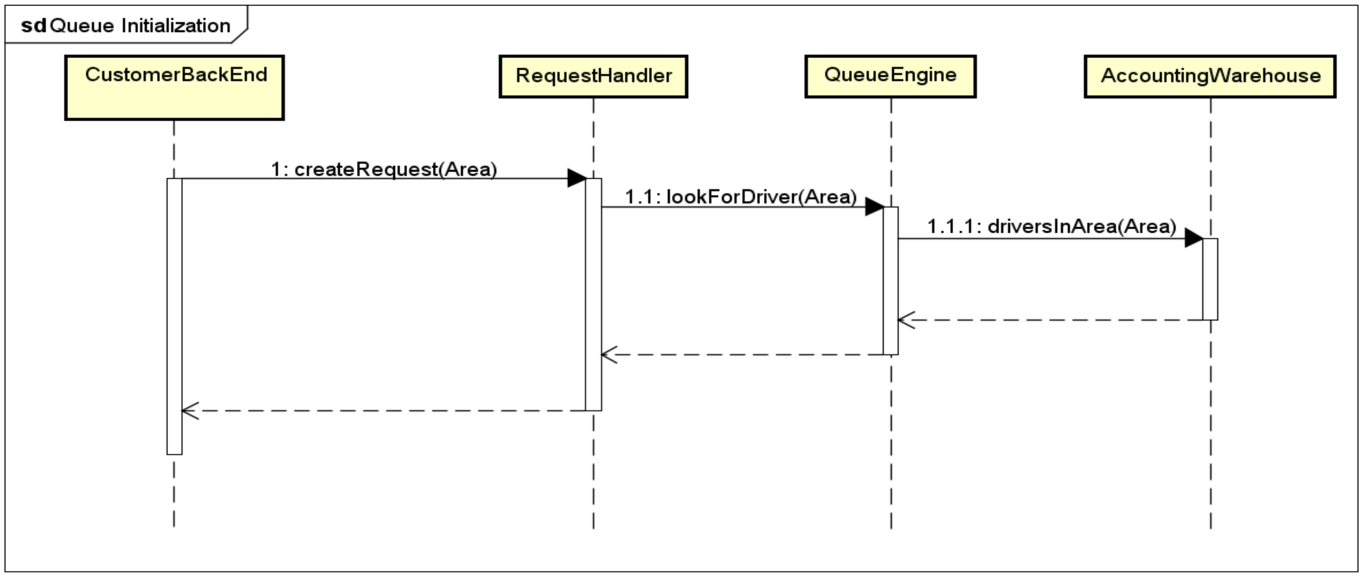
2.5.4 Request generation

Here it is described how a request is handled by the software, in particular it forwards the requests to the first driver in the queue referring to the area to which belongs, the origin, and wait for the driver reply for 2 minutes. After that the system automatically discards him and his chance to accept it, moving him to the tail of the queue and forwarding the notification to the following driver on the queue's head. Then, after looking for the driver, the system stores the request and its details into the database.



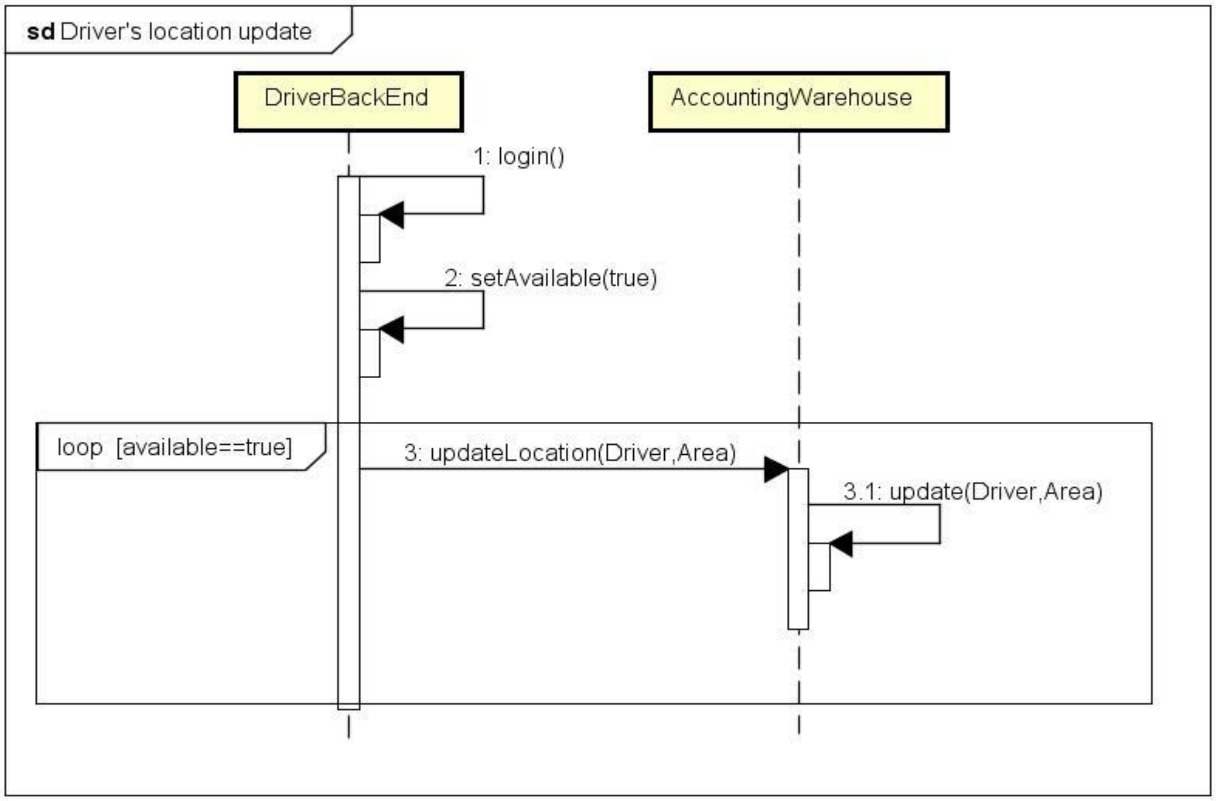
2.5.5 Queue initialization

Here it is described which components are involved for creating the queue to link to a certain request.



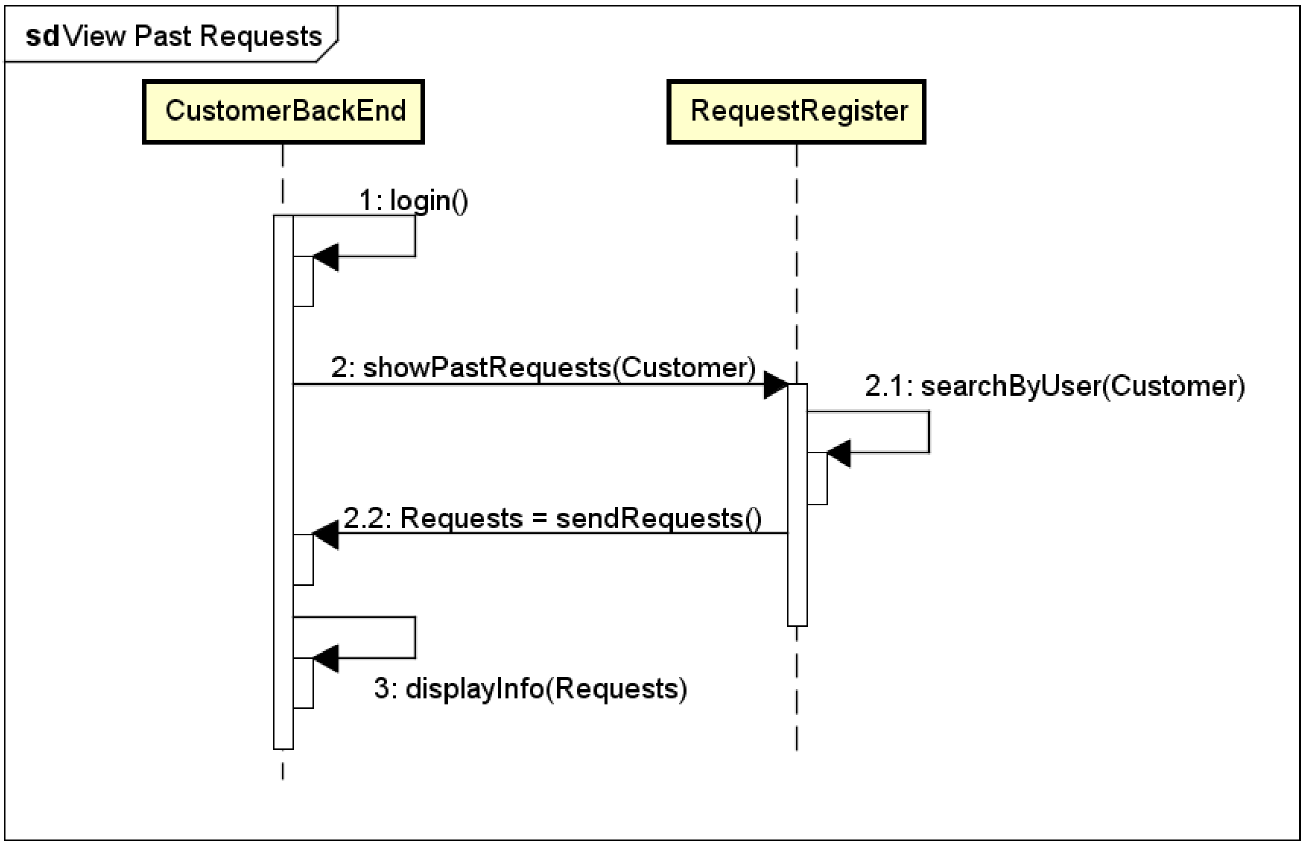
2.5.6 Driver's location update

In the following diagram it is shown how the driver's position is updated and then where it is stored.



2.5.7 View past requests

In this sequence diagrams it is explained how all the past requests can be retrieved by a user, for driver and customer both, in order to see all the details about the request done or replied.


sd View Past Requests
CustomerBackEnd
RequestRegister
1: login()
2: showPastRequests(Customer)
2.1: searchByUser(Customer)
2.2: Requests = sendRequests()
3: displayInfo(Requests)

2.6 Component interfaces

In this paragraph are listed and shortly explained all the provided and required interfaces which are shown in the previous component diagram.

2.6.1 Request Forwarder

- *Provided Interfaces: **ForwardRequest**, it gives the chance to the customer to create a new request to forward to the system.*
- *Required Interfaces: **IRequest**, it allows the component to forward the request to the application.*

2.6.2 PastReqEngine

- *Provided Interfaces: **ViewPastRequests**, this interface allows the customer to see all the requests done with his account.*
- *Required Interfaces: **IPRE**, it provides to the component all the requests linked to a certain account and it extract it from the RequestRegister where all the requests done are stored.*

2.6.3 CustomerBackEnd

- *Provided Interfaces: NULL.*
- *Required Interfaces: **CSignIn**, it allows the customer to login correctly.*

2.6.4 AccountingManager

- *Provided Interfaces: **SignIn**, it allows customers and drivers both the login. **SignUp**, it makes possible the registration into the system for a guest who wants to create a new account.*
- *Required Interfaces: **DataChecker**, this checks whether the given driver's data, such taxi license and taxi number, are valid and exist. **IEntry**, this interface is used to check that the given credentials match with one stored, in order to successfully login into the system.*

2.6.5 TaxiDriverDB

- *Provided Interfaces: **TaxiData**, it's an entry interface into the taxi drivers' data.*
- *Required Interfaces: NULL.*

2.6.6 DriverBackEnd

- *Provided Interfaces: NULL.*
- *Required Interfaces: **DSignIn**, this interface is necessary in order to allow the drivers to login successfully.*

2.6.7 NotificationEngine

- *Provided Interfaces: **NotificationAPI**, it permits to notify the drivers about the rides or reservations which they can accept.*
- *Required Interfaces: NULL.*

2.6.8 RequestEngine

- *Provided Interfaces: **ConfirmationRequest**, allows to send a reply about a notification received.*
- *Required Interfaces: NULL.*

2.6.9 RequestsDisplayer

- *Provided Interfaces: **ShowAllRequests**, let the driver see all the requests accepted.*
- *Required Interfaces: **IReqSearch**, allows to retrieve from the DataStorage all the requests accepted by the driver.*

2.6.10 GpsLocalizer

- *Provided Interfaces: **GPSApi**, this interface permits the usage of the GPS.*
- *Required Interfaces: NULL.*

2.6.11 AvailableSetter

- *Provided Interfaces: **Availability**, let the driver to communicate whether he is available or not.*
- *Required Interfaces: **Checker**, it checks whether the driver has active reservation or not in order to update correctly the driver's state and ending a request.*

2.6.12 RequestHandler

- *Provided Interfaces: **RequestCreator**, it allows the correct request creation after a request submission. **InsertRequest** has an entry point into the DataStorage in order to store the request's data.*
- *Required Interfaces: **INotify**, permits to communicate to the NotificationDispatcher the notification sending. **IConfirm** accepts all the confirmation from drivers. **IReq** allows to create and control the queue referred to the request.*

2.6.13 QueueEngine

- *Provided Interfaces: **QueueCreation**, permits to exploit the queue, like the creation or the retrieving of drivers.*
- *Required Interfaces: **DriverCollector**, when a queue is created, this interface is used to extract from the AccountingWarehouse all the drivers available and in the same area of the origin.*

2.6.14 NotificationDispatcher

- *Provided Interfaces: **Notify**, collects all the notification that have to be dispatched to a certain driver.*
- *Required Interfaces: **INotification**, is used to send the notification to the driver's notification system.*

2.6.15 RequestRegister

- *Provided Interfaces: **EntryRequest**, is the entry point to see and extract all the requests saved in the system. **RequestsActive** shows all the requests that are already active among the RequestRegister.*
- *Required Interfaces: **IInsert** permits to add a new request into the DataStorage.*

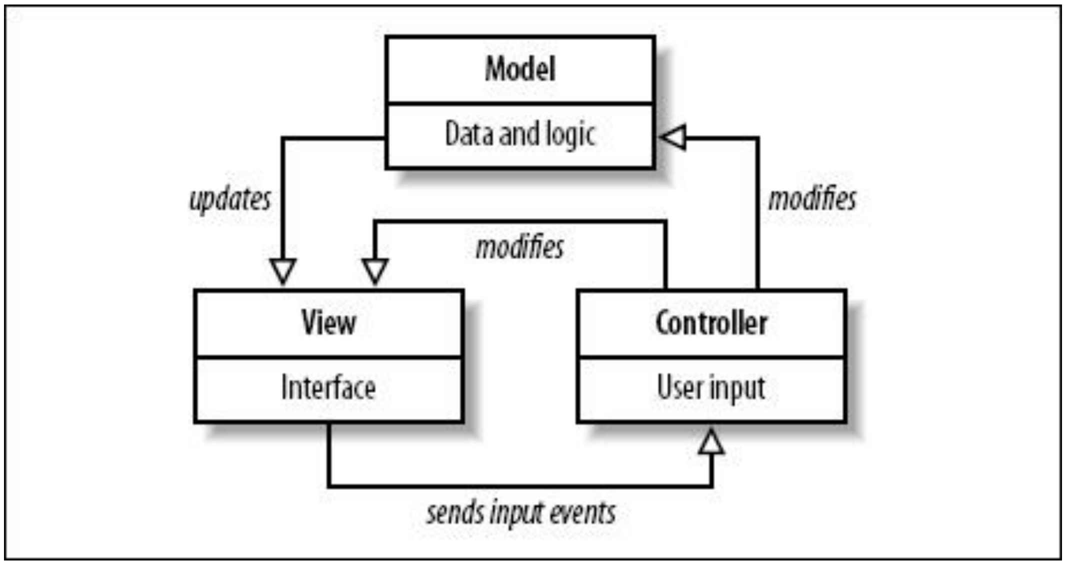
2.6.16 AccountingWarehouse

- *Provided Interfaces: **CredentialsProvider**, let the AccountingManager search among all credentials and check whether the given credential matches with one of them.*
- *Required Interfaces: **ActivityTracker** permits to update and save continuously the driver's availability and position.*

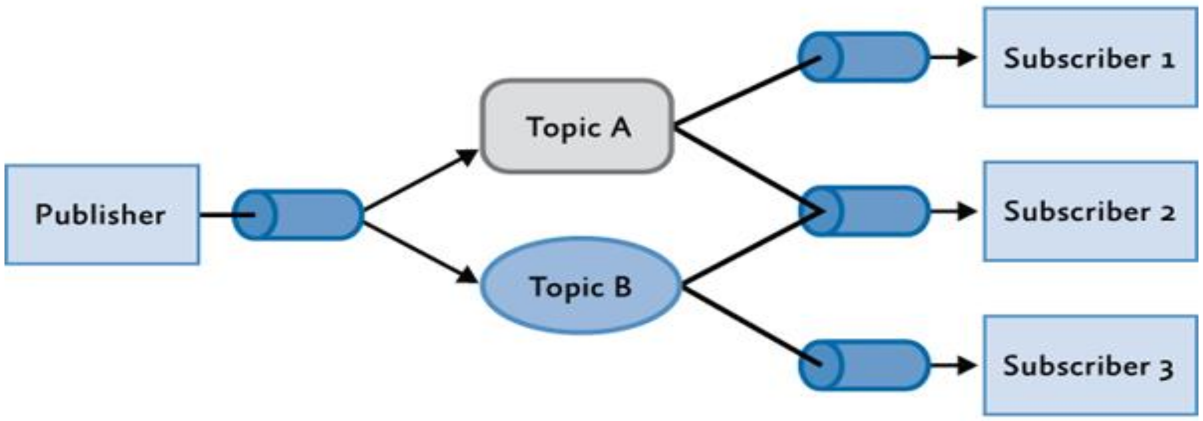
2.7 Selected architectural styles and patterns

We decided to develop the system implementing the following architectural styles and patterns:

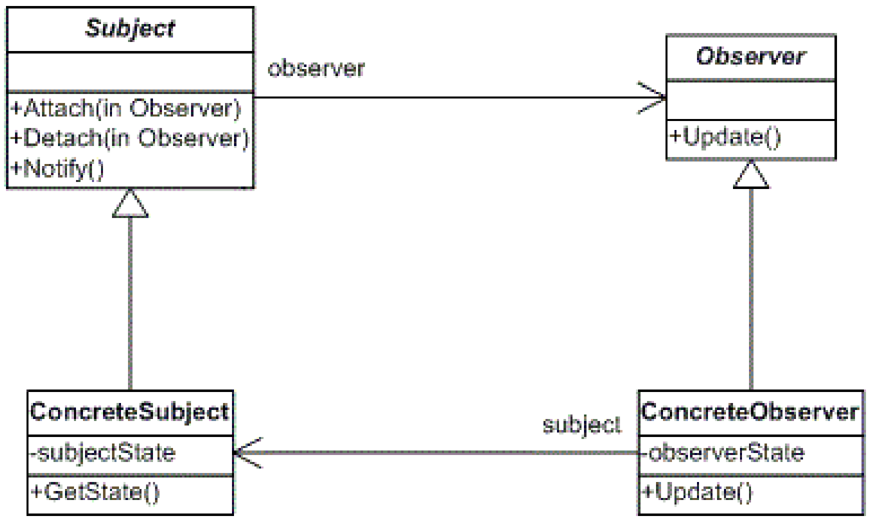
- **Model-View-Controller:** *the software application is divided into three interconnected parts. The model stores data that are retrieved according to commands from the controller and displayed in the view. The view generates an output presentation to the user based on changes in the model. The controller can send commands to the model to update the model's state. It can also send commands to its associated view to change the view's presentation of the model.*



- ***Publish-Subscribe:*** *this is a messaging pattern in which senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers, but messages are published into classes, called topics, without knowledge of which subscribers there may be. This pattern is used by NotificationDispatcher to forward requests to the drivers and in general to forward asynchronous notifications to the users.*

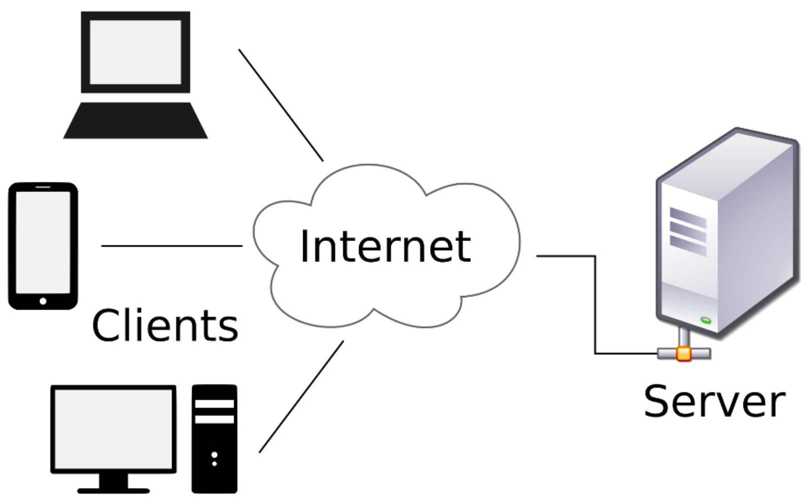


- ***Observer Pattern:*** *This pattern defines a one-to-many dependency between objects where a state change in one object results with all its dependents being notified and updated automatically. This pattern may be used in all situations where more than one display format for state information is required and where it is not necessary for the object that maintains the state information to know about the specific display formats used.*



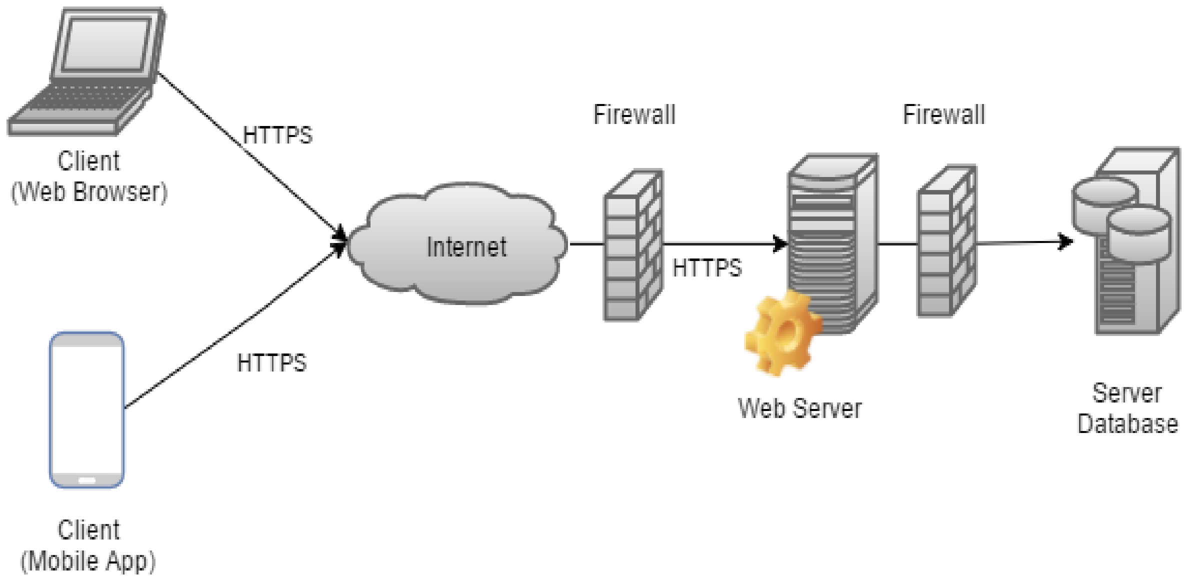
- **Client-Server:** *Client-server describes the relationship of cooperating programs in an application. The server component provides a function or service to one or many clients, which initiate requests for such services. Basically it let users create requests and send information about his state such as GPS position and availability. The pattern is used at two levels:*

  - *Communication between application server(client) and database (server).*
  - *Communication between browser/app(client) and application server (server).*



2.8 Other design decision

In this picture it's shown the hardware architecture of the system.

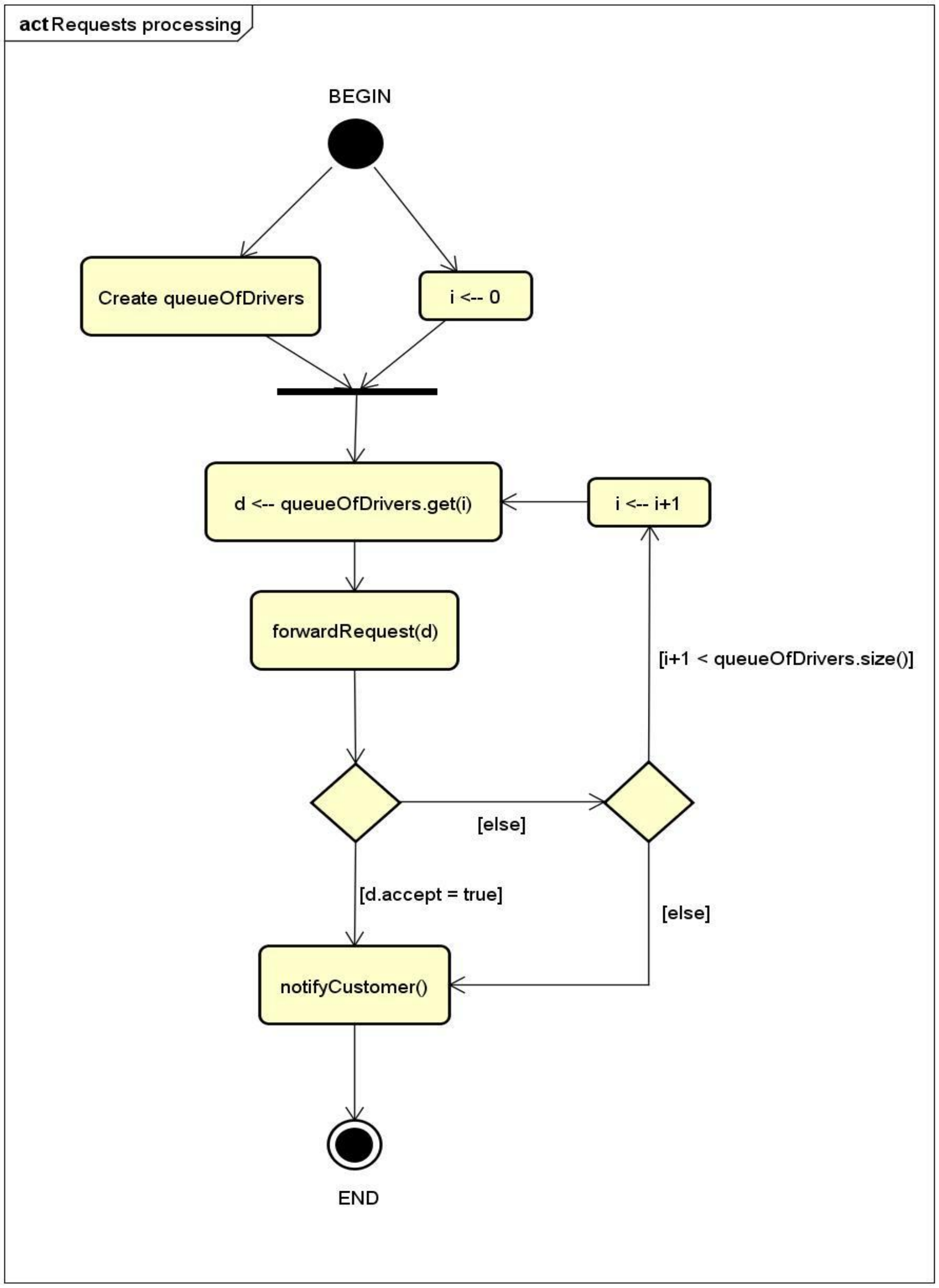


3. Algorithm Design

In this section are shown some Activity Diagrams implementing the most important functionalities of the system.

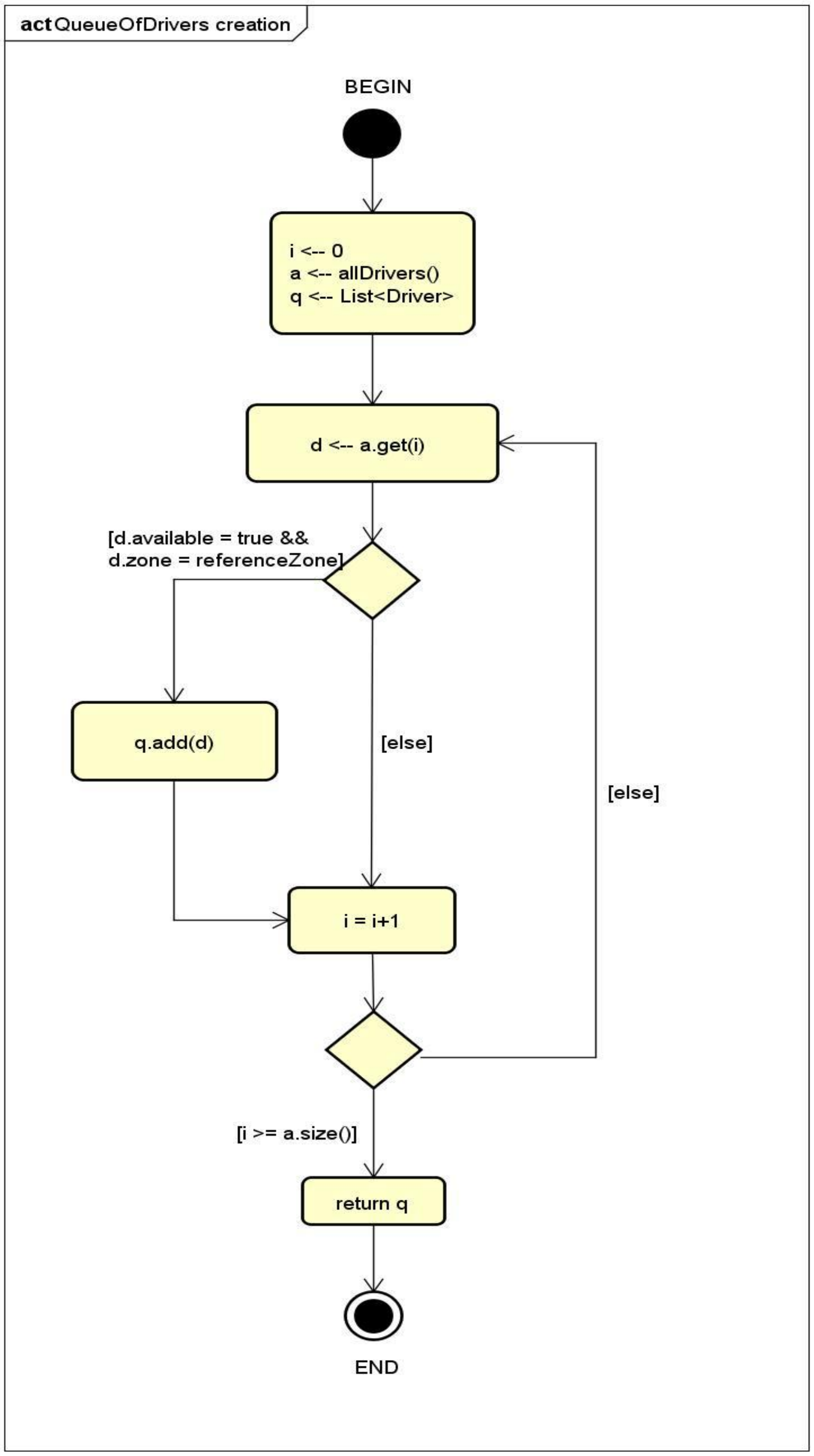
3.1 Request processing

This Activity Diagram illustrates the sequence of activities carried out by the system to forward both ride and reservation applications to drivers in queue and to notify customers about state of their requests.



3.2 QueueOfDrivers creation

This Activity Diagram illustrates how the queue of drivers, which is related to a request, is created.

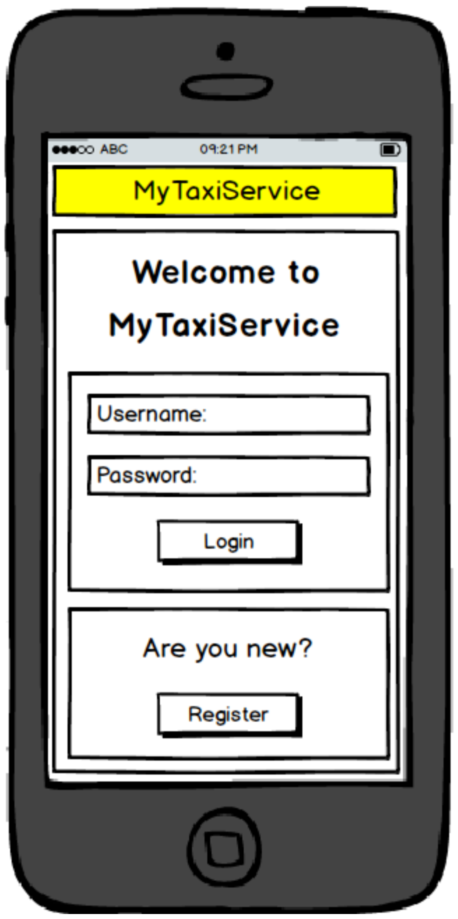


4. User Interface Design

In this section are displayed some mockups, that have not already been inserted in the RASD, representing the system from the user side; in particular the following mockups are related to the mobile application.

4.1 Homepage – Mobile App

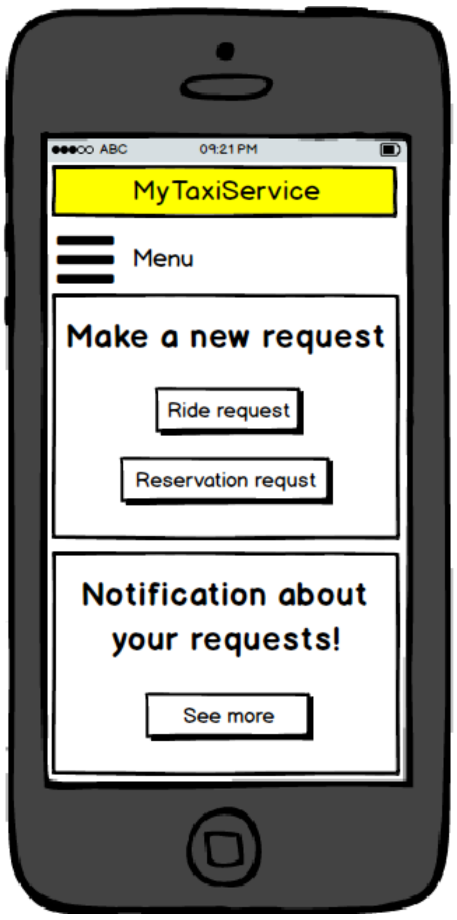
This mockup shows the homepage of the mobile app in which visitor can only sign up or sign in.



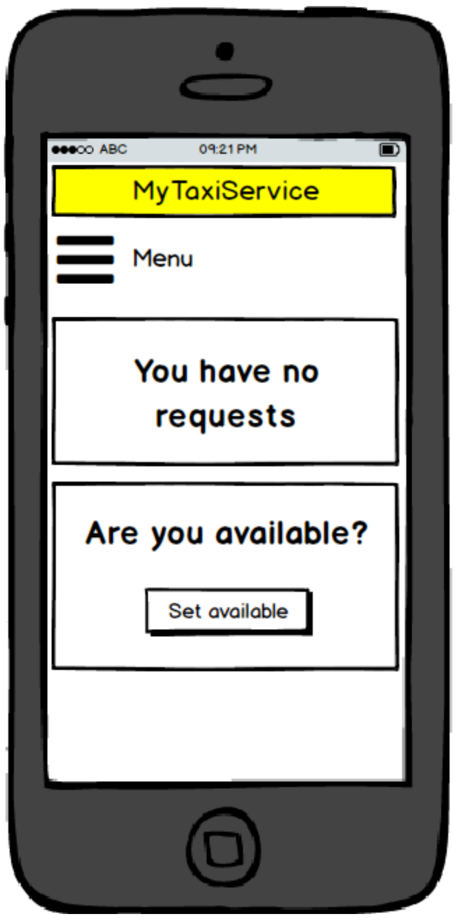
4.2 Dashboard – Mobile App (4.2.1 Customer – 4.2.2 Driver)

The first mockup (4.2.1) shows the mobile app dashboard of the customer in which he can access the menu, go to the form to request a ride or a reservation and go to the page with details about a request made before. The second mockup (4.2.2) shows the mobile app dashboard of the driver in case he has no request; in this case he can only access the menu and set his availability.

4.2.1



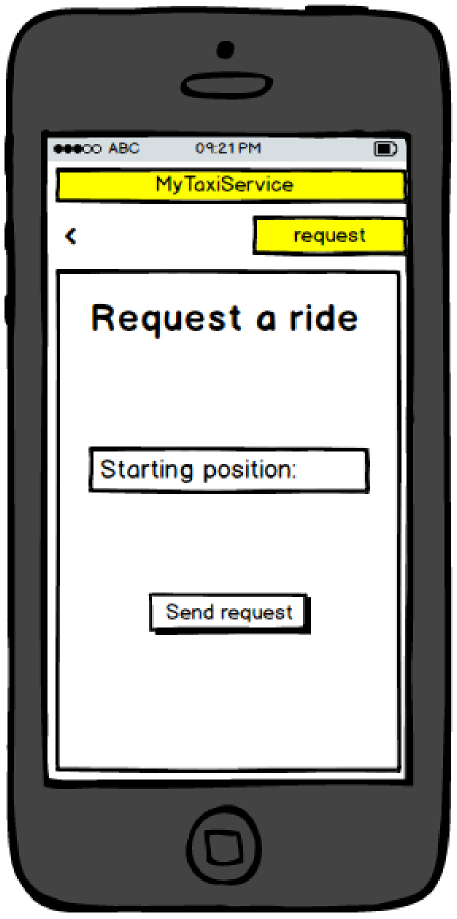
4.2.2



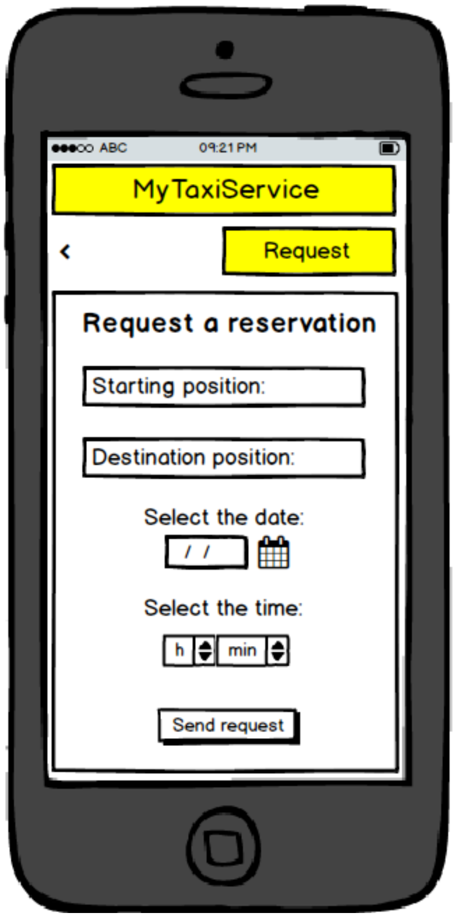
4.3 Request a ride (4.3.1) or a reservation (4.3.2)

These mockup shows the mobile app form to be completed by the customer for request a ride (4.3.1) or a reservation (4.3.2).

4.3.1



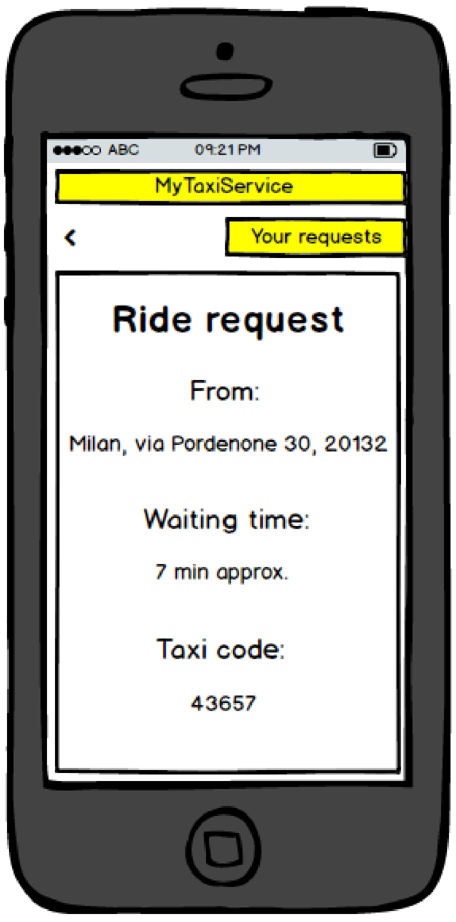
4.3.2



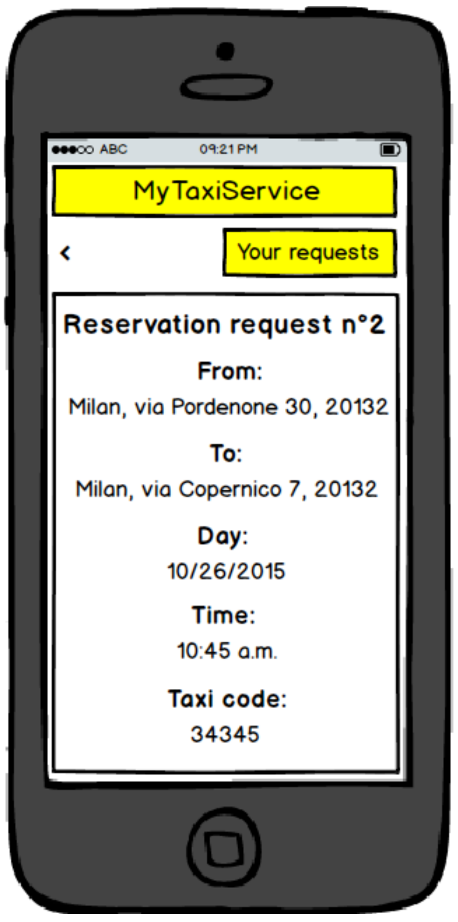
4.4 Ride's details (4.4.1) and reservation's details (4.3.2)

The first (4.4.1) mockup shows the mobile app page in which a customer can see details, such as code of taxi and waiting time of a ride. The second (4.4.2) mockup shows the mobile app page in which a customer can see details such as taxi code, date and time of a reservation that he has previously requested.

4.4.1



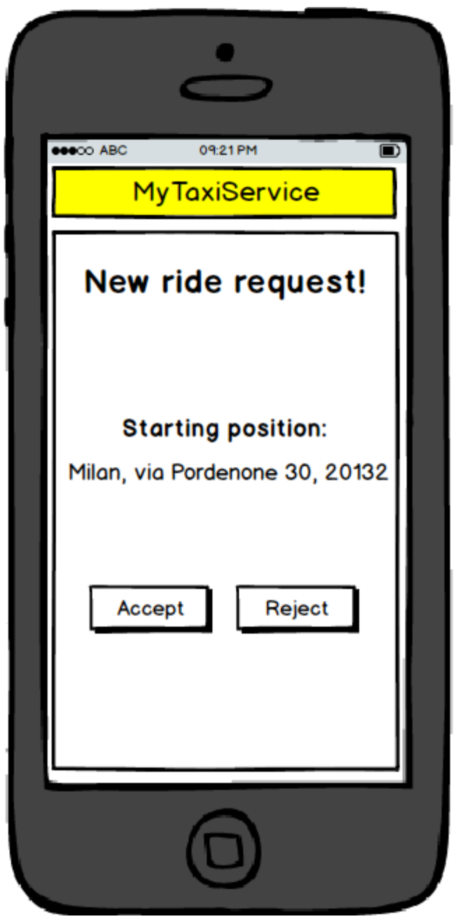
4.4.2



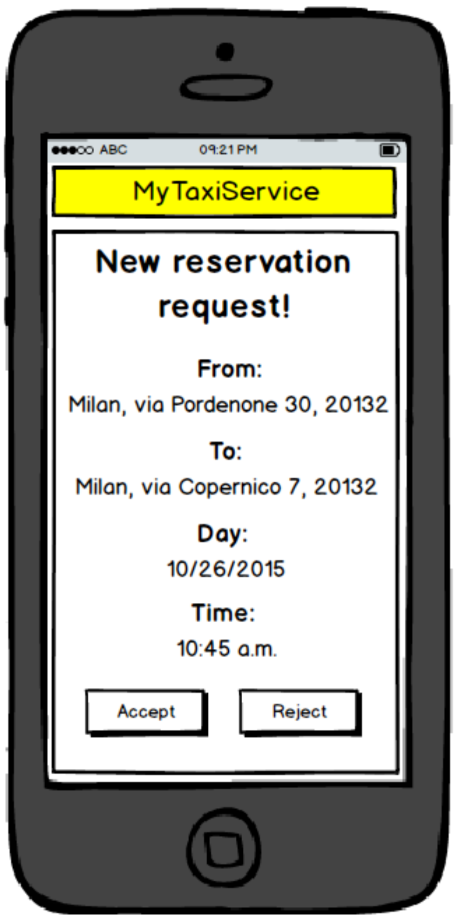
4.5 New ride request (4.5.1) and new reservation request (4.5.2)

These mockups show the mobile app page in which a driver can see details of a ride (4.5.1) or a reservation (4.5.2) that have been forwarded to him. He can accept or reject the requests.

4.5.1



4.5.2



5. Requirements Traceability

5.1 Sign up as Customer

Requirements:

[R1] CF must be in compliance to the inserted data
[R2] Visitor can only sign up or sign in

Requirement	*Design Elements*
R1	*AccountingManager*
R2	

5.2 Sign up as Driver

Requirements:

[R1] CF must be in compliance to the inserted data
[R2] Visitor can only sign up or sign in
[R3] Taxi code must be associated to the Visitor's taxi license
[R4] Taxi license must belong to the Visitor

Requirement	**Design Elements**
R1	*AccountingManager*
R2	
R3	*AccountingManager, TaxiDriverDB*
R4	

5.3 Login

Requirements:

[R1] Credentials must be correct
[R2] Visitor must not be already logged

Requirement	**Design Elements**
R1	*AccountingManager*
R2	

5.4 Make a ride

Requirements:

[R1] Customer must insert a position covered by the service
[R2] Customer must complete the form and send the request

Requirement	**Design Elements**
R1	*RequestForwarder, RequestHandler*
R2	*RequestForwarder*

5.5 Make a reservation

Requirements:

[R1] Customer must insert a position covered by the service
[R2] Customer must complete the form and send the request
[R3] Request must be forwarded to the 1st Driver in queue

Requirement	**Design Elements**
R1	*RequestForwarder, RequestHandler*
R2	*RequestForwarder*
R3	*RequestForwarder, RequestHandler, NotificationDispatcher, QueueEngine*

5.6 Set available

Requirements:

[R1] Driver must not be associated to another reservation in the same time

Requirement	**Design Elements**
R1	*AvailableSetter, RequestRegister*

5.7 Respond to a request

Requirements:

[R1] Driver must respond to a request

Requirement	**Design Elements**
R1	*RequestEngine, RequestHandler*

6. Appendix

6.1 Tools

- *Microsoft Word: Used for create the following document.*
- *Astah: For generating UML diagrams such as Deployment, Component, Sequence and Activity.*
- *Microsoft OneDrive: Used for sharing files among the group.*
- *Balsamiq Mockups 3: For generating mockups.*

6.2 Hours of Work

- *Francesco: 27 hours.*
- *Marco: 27 hours.*