myTaxiService

Design Document

Belluschi Marco 791878, Cerri Stefano 849945, Di Febbo Francesco 852389

December 4, 2015

Contents

Chapter 1

Introduction

1.1 Purpose

This document contains the complete design description of myTaxiService. This includes the architectural features of the system down through details of what operations each code module will perform and the database layout. It also shows how the use cases detailed in the RASD will be implemented in the system using this design. The primary audiences of this document are the software developers.

1.2 Scope

MyTaxiService is a taxi service for a large city. The main goals of the system are:

- simplify the access of passengers to the service
- guarantee a fair management of taxi queues

The system architecture will be a three-tier architecture: client, server application and server database. It will be created by using the MVC architectural pattern.

The system will be divided into components with respect to the principles leading to good design:

- Each individual component will be smaller in order to be easier to understand
- Coupling will be reduce where possible
- Reusability and flexibility will be increase in order to make easier future implementation

The system will have efficient algorithm in order to increase its performance; in the document will be given special attention to the sharing algorithm.

1.3 Definitions, Acronyms, Abbreviations

Definitions

- User: person that uses the service applications
- Visitor: user that has not registered nor logged in
- Registered user: user that has registered to the service
- Passenger: passenger registered to the service
- Taxi driver: taxi driver registered to the service
- System: the union of software and hardware to be developed and implemented

Acronyms

- RASD: requirements analysis and specification document
- AES: Advanced Encryption Standard
- FIFO: First In First Out
- ETA: estimated time of arrival
- API: application programming interface
- GPS: Global Positioning System
- MVC: Model View Controller
- UX: User Experience

1.4 Reference Documents

- Software Engineering 2 Project AA 2015/2016: Project Description And Rules
- Software Engineering 2 Project AA 2015/2016: Assignments 1 and 2 (RASD and DD)
- Software Engineering 2 Project AA 2015/2016: Structure of the design document

1.5 Document Structure

This document is essentially structured in seven parts:

- Introduction: it gives a description of the document and some basical information about the system design and archicture.
- Architectural Design: This is the core of the document. It gives general information about the architectural design. It also describes how the system will be divided into components and how the components communicate. It also has a description of the design pattern and architectural styles that will be used.
- Algorithm Design: it gives a description of the main algorithm that will be implemented. More focus will be given in the sharing algorithm.
- User Interface Design: it gives a description of the user interfaces of the system and the flow from one interface to another.
- Requirements Traceability: this section documents the life of a requirement and provides bi-directional traceability between various associated requirements.
- References: it gives information on the guidelines used in order to redact this document.
- Appendix: it provides informations that are not considered part of the actual DD. It includes: software and tools used, project group organization.

Chapter 2

Architectural Design

2.1 Overview

This chapter describes the software system, the relationships between software components and the relationship to actors with the system. Each component is described by a specification and an interface design. The specification is a description of its purpose, its functionality, its attributes (including dependency on other components) and the constraints under which it must operate. It also describes resources, that is, any elements used by the component which are external to the design such as physical devices and software services. The interface design is the list of the services that it provides to clients. These services are methods (procedures and functions), each carefully documented.
Each component in turn may provide its services by having an internal architectural design with its own set of subordinate components. These components may be called sub-components. The decomposition of a higher-level component into subordinate component must be explicit. The algorithm that shows how each method of the larger component is performed by these components must be explicit. Any data stored in an component must be explicitly described.

2.2 High level components and their interaction

2.2.1 myTaxiService

Type System

Node myTaxiServer

Description This is the primary entrance to the system for a user. It provides all the functionalities of myTaxiService by the interaction of software components. It provide APIs for software development.

Dependencies myTaxiDatabase, Google Maps API

Resources Glassfish 4.1.1 on Linux Ubuntu Server 14.04.3L LTE on server

Operations

- Account API (sign-up, login, personal information editing, taxi driver availability handling)
- Ride request API (standard, shared, reserve ride request)
- Notification API (notifications from system)
- Map API (city zones, geolocation services)
- Queues API (taxi queues management)

2.2.2 Web app

Type Web app

Node Client

Description The web application provide an easy way to reach myTaxiService functionalities. Through supported web browser, the passenger can request standard, shared or reserved rides.

Dependencies Account API, Ride request API, Notification API

Resources Supported web browser (see RASD for further informations)

Operations

- Sign up, login, personal informations editing
- Ride requests (only functionalities for passenger)
- Notification from service

2.2.3 Mobile app

Type Mobile app

Node Client

Description The mobile application extends web functionalities by adding taxi driver services.

Dependencies Account API, Ride request API, Notification API

Resources Supported devices (see RASD for further informations)

Operations

- Sign up, login, personal informations editing (for taxi driver, taxi driver availability handling)
- Ride request (passenger: request for ride; taxi driver: request handling)
- Notification from service

2.2.4 Account manager

Type Subsystem of myTaxiService

Node myTaxiServer

Description This component manages users connected to the service. It manages sign-up and login functionalities, profile editing and taxi driver availability handling. All data are stored on myTaxiDatabase through Data API. It provides APIs for software development.

Dependencies Data manager component

Operations

- Account API
- Sign-up for taxi drivers and passengers
- Login
- Informations editing
- Taxi driver availability handling
- Access control

2.2.5 Ride manager

Type Subsystem of myTaxiService

Node myTaxiServer

Description This component manages the incoming requests from passengers and responses from taxi drivers. It manages all types of requests, including standard, shared and reserved rides thanks to the interaction with Taxi Queues component. It takes care of messages dispatching to users by Notification component. It provides APIs for software development.

Dependencies Data manager, Map services, Taxi queues, Notification, Account manager components

Operations

- Ride request API
- Standard ride request
- Shared ride request
- Reserved ride request

2.2.6 Taxi queues

Type Subsystem of myTaxiService

Node myTaxiServer

Description this component manages the queue of available taxis for each city zone. It depends on Map services component for location services. It provides APIs for software development.

Dependencies Mpa services component

Operations

- Queues API
- Taxi queues management

2.2.7 Map services

Type Subsystem of myTaxiService

Node myTaxiServer

Description This component manages the city map, the taxi zones, the location informations provided by users (from GPS navigation systems or user input). It uses Google Maps API and location data are stored in myTaxiDatabase through Data API. It provides APIs for software development.

Dependencies Data manager component, Google Maps API

Operations

- Map API
- Taxi zones
- Location input parsing
- Places management

2.2.8 Notification

Type Subsystem of myTaxiService

Node myTaxiServer

Description This component takes care of messages dispatching to the users. It provides APIs for software development.

Dependencies Account manager component

Operations

- Notification API
- Ride handling notifications to taxi drivers
- Ride request notifications to users
- Service notifications to user

2.2.9 Data manager

Type Subsystem of myTaxiService

Node myTaxiServer

Description This component provides access to all of the data contained in the database. It provides various functions that allow entry, storage and retrieval of large quantities of information and provides ways to manage how that information is organized. It relies on myTaxiDatase through its API. It provides APIs for software development.

Dependencies myTaxiDatabase system

Operations

- Data API
- Data access
- Data presentation
- Data organization

2.2.10 myTaxiDatabase

Type System

Node myTaxiDatabase

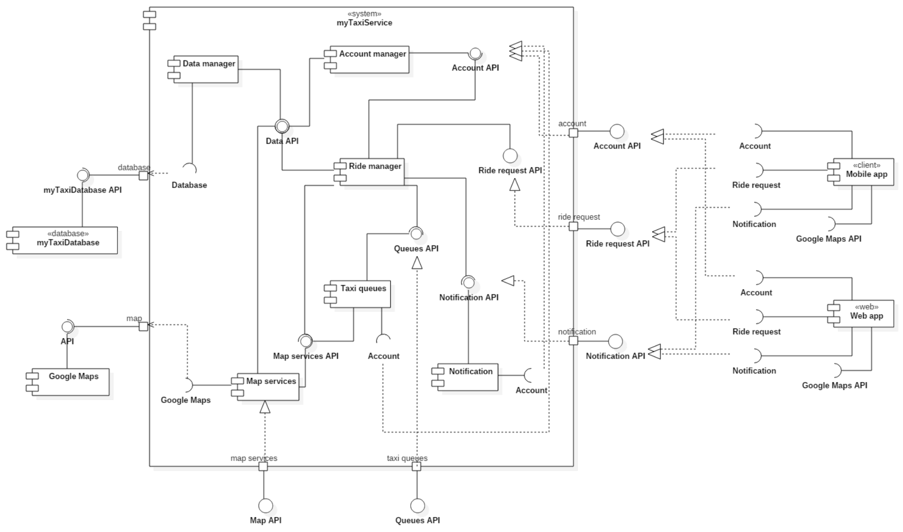
Description This system provides all data of myTaxiService. This system takes care of data storing and provisioning. The data security system prevents to unauthorized users from accessing or updating the database.

Resources MySQL 5.7 on Linux Ubuntu 14.01.3L LTE on server

Operations

- myTaxiDatabase API
- Data storare management
- Data presentation
- Security management
- Multiuser access control
- Backup and recovery management
- Data integrity management
- Data transaction management

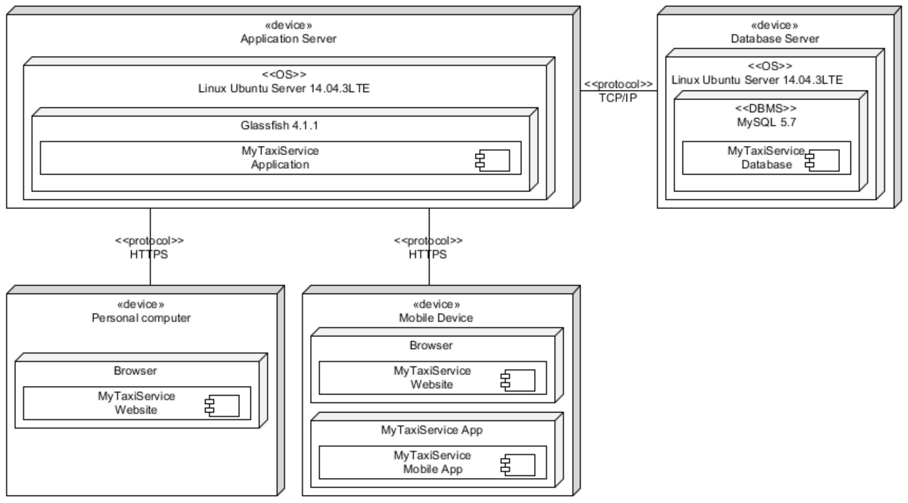
2.3 Component view



Component diagram of the entire system

2.4 Deployment view

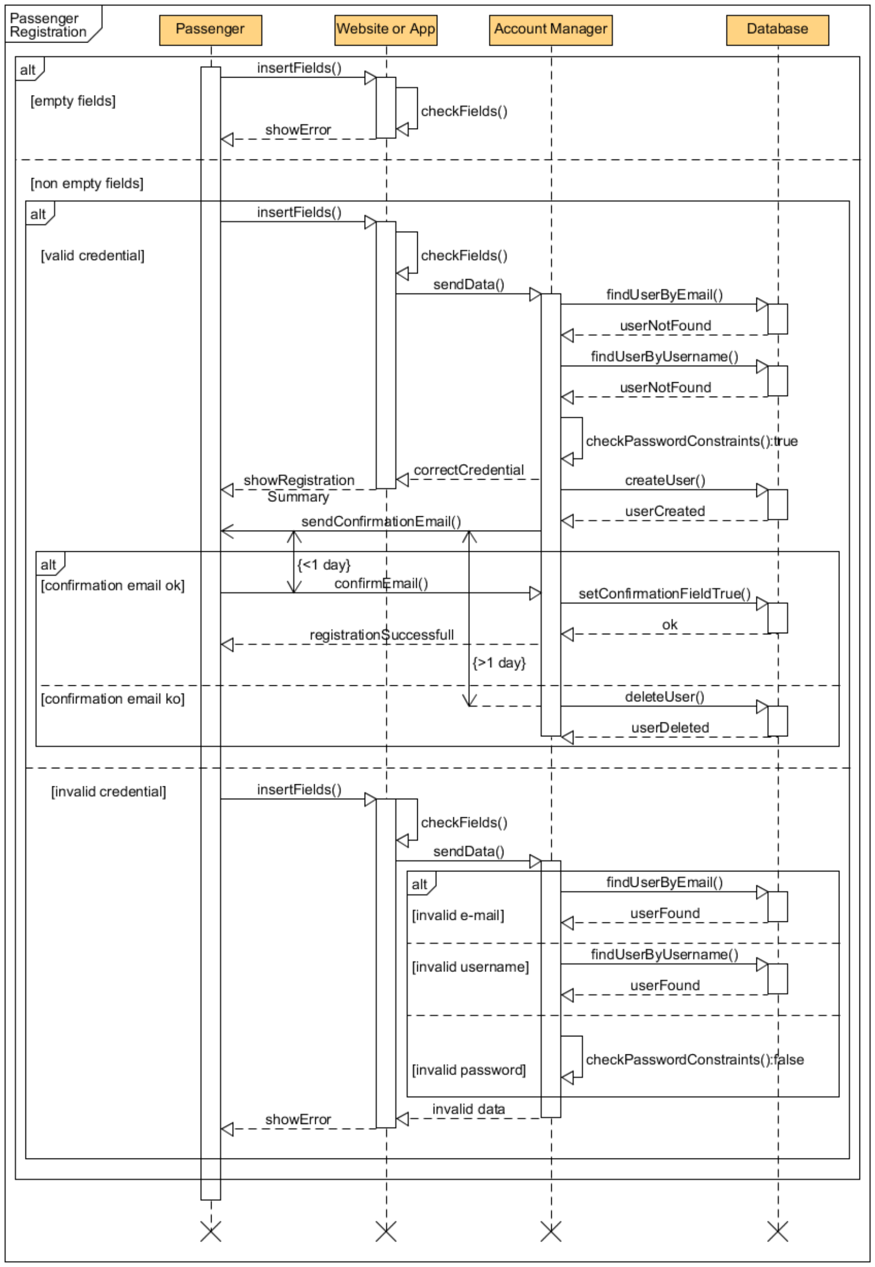
Here is the deployment diagram for MyTaxiService:

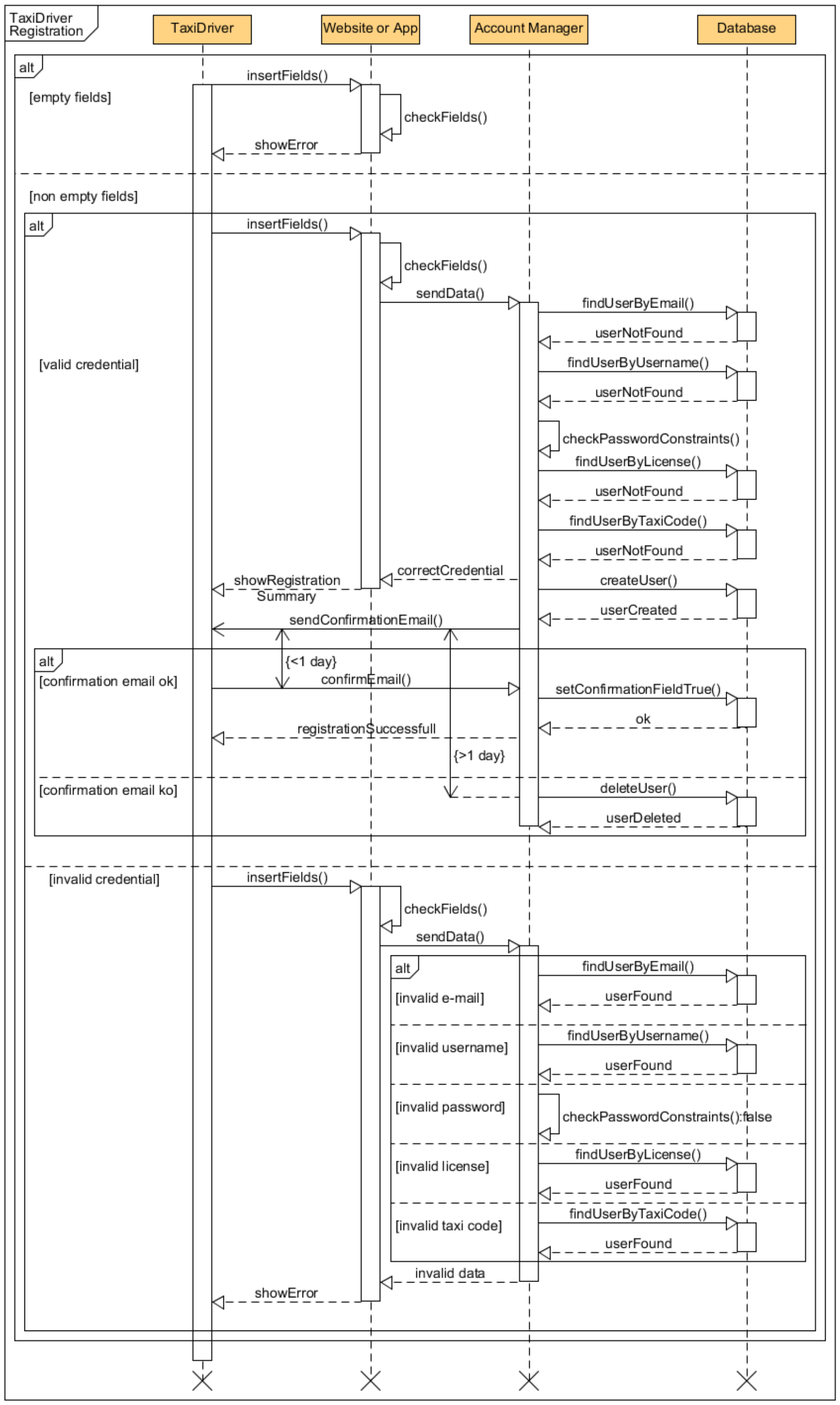


2.5 Runtime view

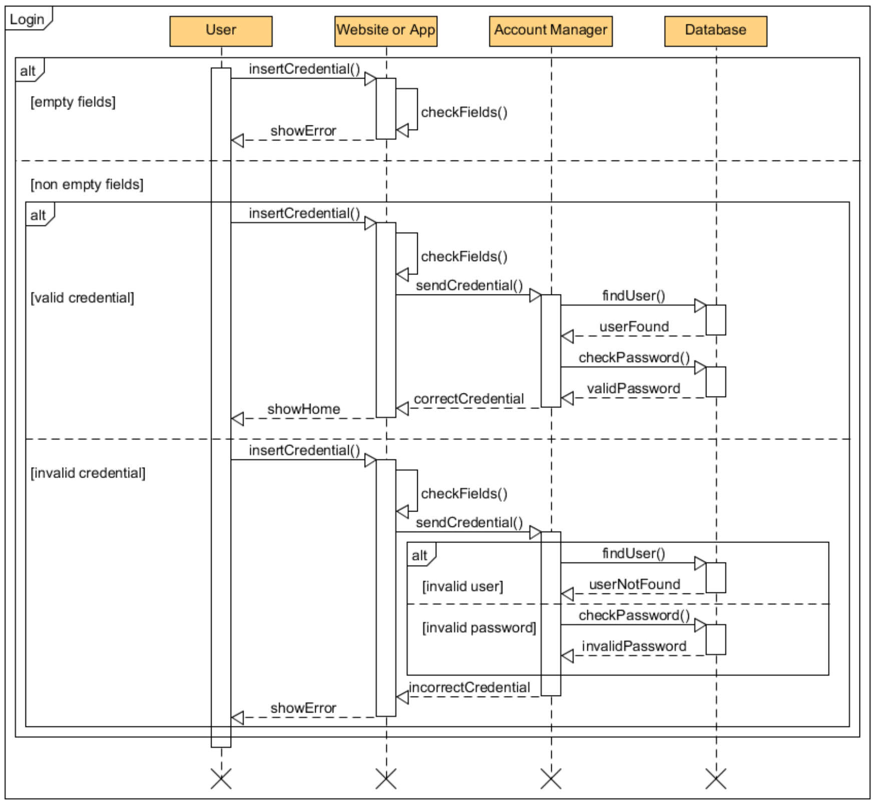
In this section will be presented the interactions between components usign some sequence diagram.

Registration

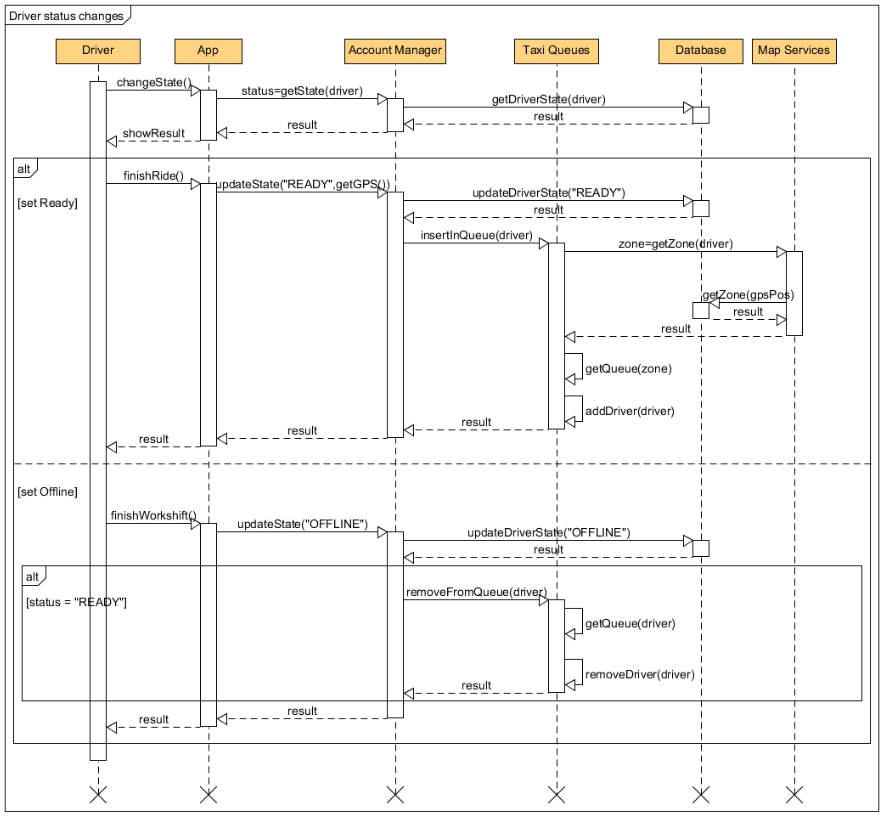



TaxiDriver Registration
TaxiDriver
Website or App
Account Manager
Database
alt
[empty fields]
insertFields()
checkFields()
showError
[non empty fields]
alt
insertFields()
checkFields()
sendData()
findUserByEmail()
userNotFound
[valid credential]
findUserByUsername()
userNotFound
checkPasswordConstraints()
findUserByLicense()
userNotFound
findUserByTaxiCode()
userNotFound
correctCredential
showRegistration Summary
createUser()
userCreated
sendConfirmationEmail()
alt
{<1 day}
[confirmation email ok]
confirmEmail()
setConfirmationFieldTrue()
ok
registrationSuccessfull
{>1 day}
[confirmation email ko]
deleteUser()
userDeleted
[invalid credential]
insertFields()
checkFields()
sendData()
alt
findUserByEmail()
[invalid e-mail]
userFound
findUserByUsername()
[invalid username]
userFound
[invalid password]
checkPasswordConstraints():false
findUserByLicense()
[invalid license]
userFound
findUserByTaxiCode()
[invalid taxi code]
userFound
invalid data
showError

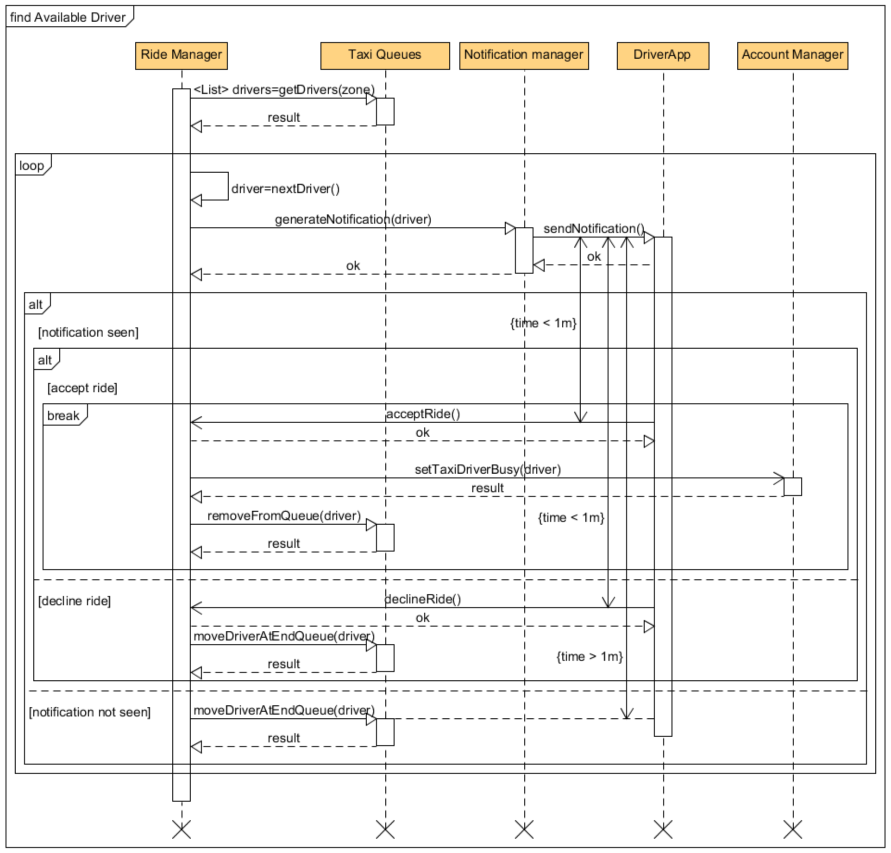
Login


Login
User
Website or App
Account Manager
Database
alt
[empty fields]
insertCredential()
checkFields()
showError
[non empty fields]
alt
[valid credential]
insertCredential()
checkFields()
sendCredential()
findUser()
userFound
checkPassword()
validPassword
correctCredential
showHome
[invalid credential]
insertCredential()
checkFields()
sendCredential()
alt
[invalid user]
findUser()
userNotFound
[invalid password]
checkPassword()
invalidPassword
incorrectCredential
showError

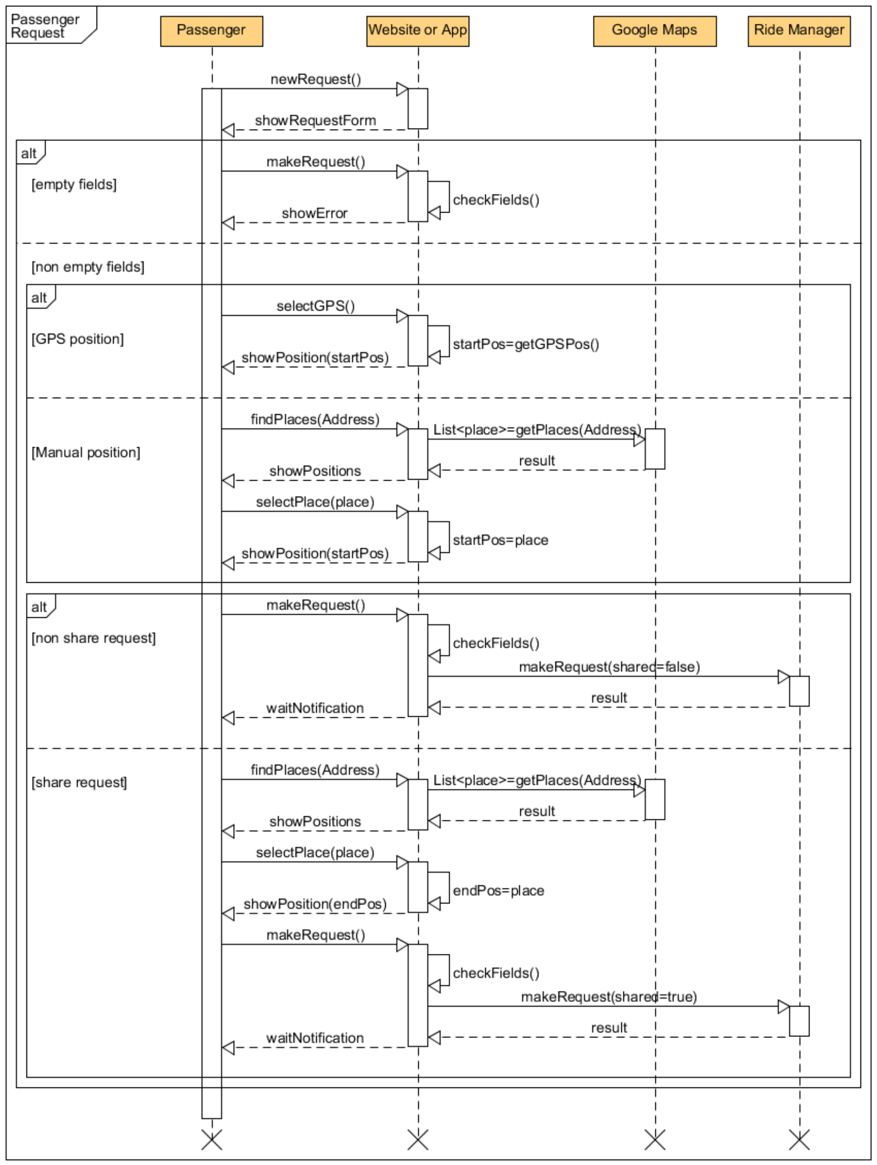
Driver Status hanges

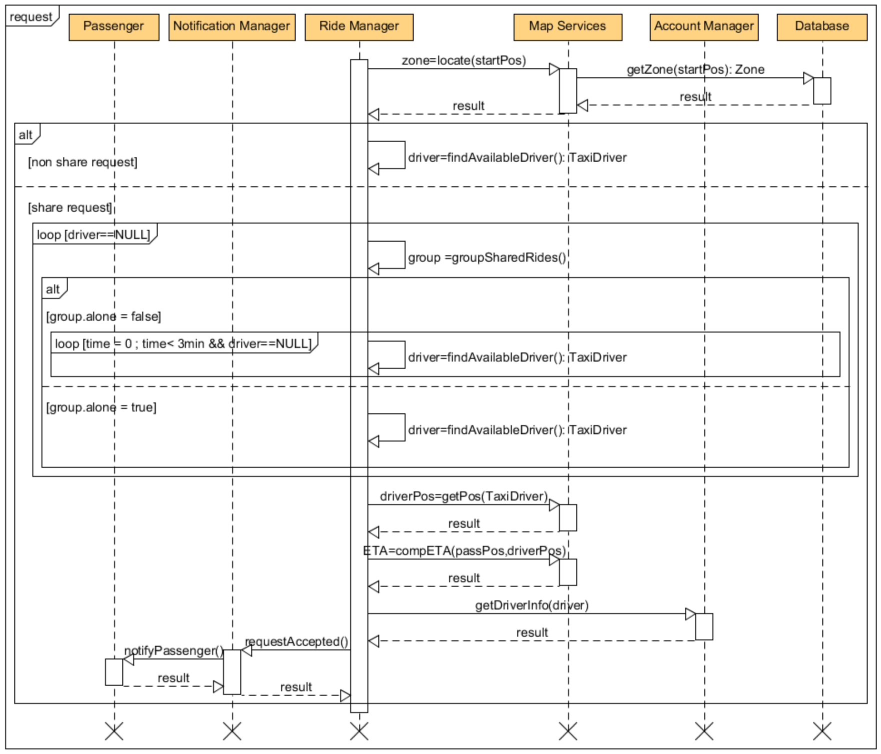


Find Available Driver

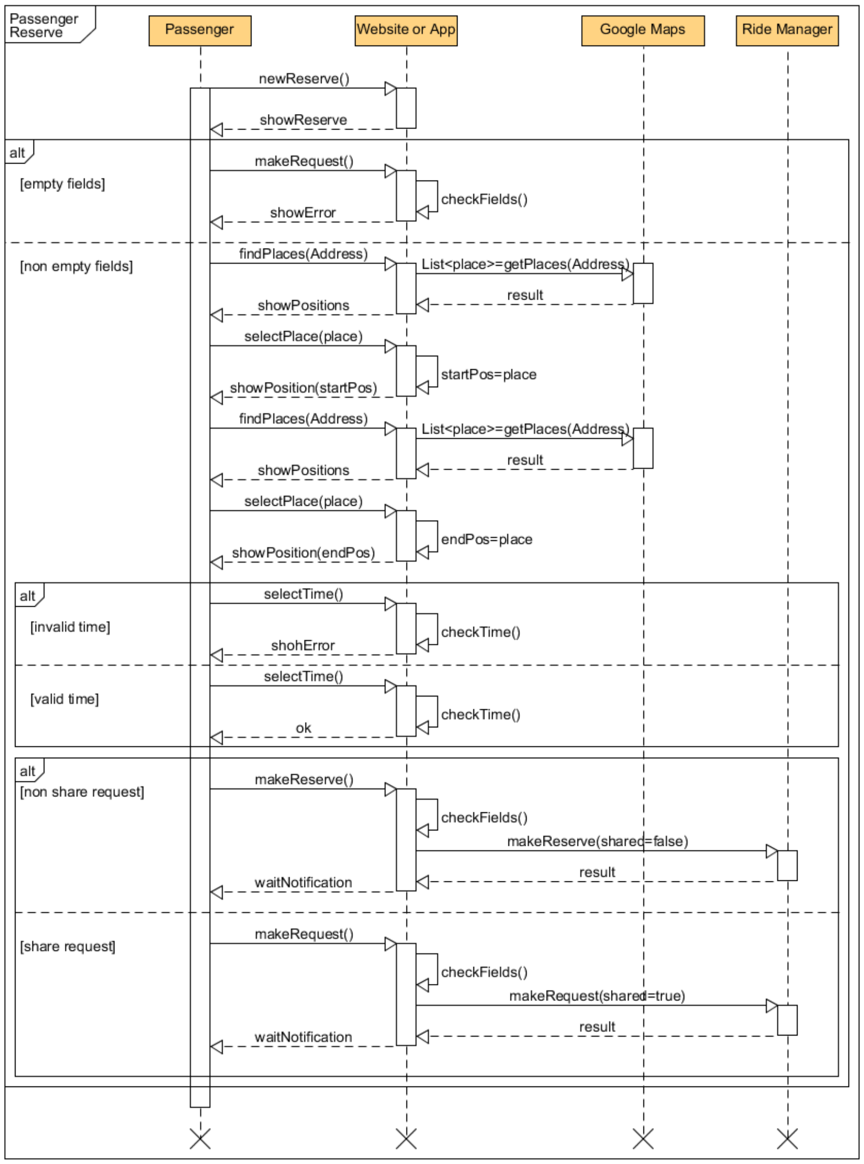


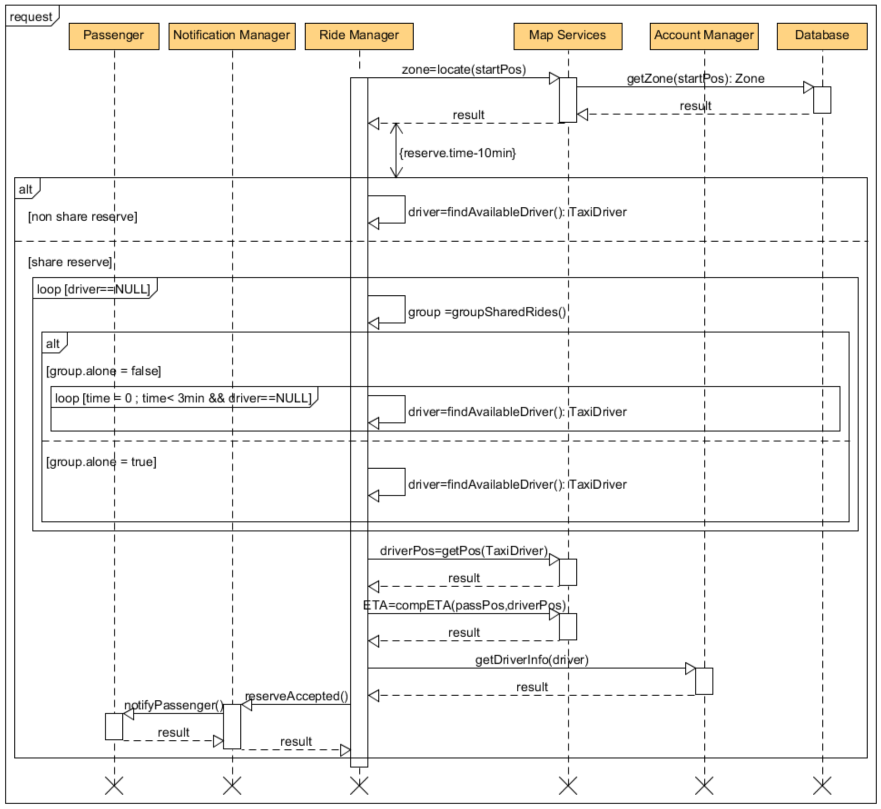
Taxi Request




request
Passenger
Notification Manager
Ride Manager
Map Services
Account Manager
Database
zone=locate(startPos)
getZone(startPos): Zone
result
result
alt
[non share request]
driver=findAvailableDriver(): TaxiDriver
[share request]
loop [driver==NULL]
group =groupSharedRides()
alt
[group.alone = false]
loop [time = 0 ; time< 3min && driver==NULL]
driver=findAvailableDriver(): TaxiDriver
[group.alone = true]
driver=findAvailableDriver(): TaxiDriver
driverPos=getPos(TaxiDriver)
result
ETA=compETA(passPos,driverPos)
result
getDriverInfo(driver)
result
requestAccepted()
notifyPassenger()
result
result

Taxi Reserve





2.6 Component interfaces

This section describes all interfaces between components, their interaction and their input/output parameters. Regarding all myTaxiServer (application) internal components, see also *RASD* and chapter 2.2, *"High level components and their interaction"* for further, detailed explanations about their functioning, dependencies, resources, operations and parameters.

2.6.1 Application↔Database Interface

Components:

- Data Manager (myTaxiServer) (Application)
- myTaxiDatabase (Database)

Communication system: JPA, JDBC APIs;

Protocols: standard TCP/IP protocols;

2.6.2 Application↔Client Interface

Components:

- myTaxiServer (Application)
- Mobile App (Client)
- Web App (Client)

Communication system: JAX-RS API (RESTful interface);

Protocols: standard HTTPS protocol;

2.6.3 Account manager

Operation		Involved Users	Input/Output Parameters	[type]
(General)		*(All)*	token errors	[/] [/]
Registration	*(Passengers)*	Visitors	username email password ...	[string] [string] [string]
	(Taxi Drivers)	Visitors	username email password license ID ...	[string] [string] [string] [string]
Login		Visitors	username/email password	[string] [string]
Email Confirmation		Passengers Taxi Drivers	password	[string]
Profile Editing		Passengers Taxi Drivers	new email new password new address ...	[string] [string] [string]
Profile Deleting		Passengers Taxi Drivers	token password	[/] [string]

2.6.4 Ride manager

Operation	**Involved Users**	**Input/Output Parameters**	[type]
(General)	*(All)*	token errors	[/] [/]
Ride Request *(Standard)*	Passengers Taxi Drivers	requesting passenger(s) start location(s) num of passengers ... shared destination(s) grouped ...	[string] [Position] [int] [boolean] [Position] [boolean]
Ride Request *(Reserved)*	Passengers Taxi Drivers	requesting passenger(s) start location(s) destination(s) num of passengers ... shared grouped ...	[string] [Position] [Position] [int] [boolean] [boolean]
Ride	Passengers Taxi Drivers	start location(s) destination(s) start date(s) end date num of passengers shared ...	[Position] [Position] [date] [date] [int] [boolean]
Request Status Update	Passengers Taxi Drivers	request status new request status	[enum] [enum]
Ride Status Update	Passengers Taxi Drivers	ride status new ride status	[enum] [enum]

2.6.5 Taxi queues

Operation	**Involved Users**	**Input/Output Parameters**	[type]
(General)	*(All)*	token errors	[/] [/]
Queues Management	Taxi Drivers	taxi driver(s) taxi zone	[Taxi Driver] [Zone]
Availability Settings	Taxi Drivers	driver status new driver status	[enum] [enum]

2.6.6 Map services

Operation	**Involved Users**	**Input/Output Parameters**	[type]
(General)	*(All)*	token errors	[/] [/]
Zones Management	Passengers Taxi Drivers	location taxi queue taxi zone	[Position] [Queue] [Zone]
Google Maps APIs	Passengers Taxi Drivers	address location	[string] [Position]

2.6.7 Notification

Operation	**Involved Users**	**Input/Output Parameters**	[type]
(General)	*(All)*	token	[/]
		errors	[/]
Ride Request Notification	Passengers	acceptor taxi driver	[string]
		driver location	[Position]
		ETA	[interval]
		grouped	[boolean]
		...	
	Taxi Drivers	requesting passenger(s)	[string]
		start location(s)	[Position]
		destination(s)	[Position]
		ETA(s)	[interval]
		estimated durations(s)	[interval]
		num of passengers	[int]
		shared	[boolean]
		grouped	[boolean]
		fee(s)	[percentage]
		...	
Ride Handling Notification	Taxi Drivers	start location(s)	[Position]
		destination(s)	[Position]
		ETA(s)	[interval]
		estimated durations(s)	[interval]
		shared	[boolean]
		grouped	[boolean]
		fee(s)	[percentage]
		...	

2.6.8 Data manager

See subsection 2.6.1, *"Application↔Database Interface"*.

2.7 Selected architectural styles and patterns

Various architectural and logical choices have been justified as following:

- A **3-tier architecture** has been used: client, server application and server database.

This is due to the fact that we're developing an overall light application that doesn't need a lot of computing power, especially on the client side. Therefore, it is possible to structure the system in a logically simple and easily understandable way, without having to lose much in terms of optimization. Besides, this allows to obtain a good compromise between thin client and database tiers, and a clear correspondence between tiers and layers (i.e. no layer is distributed among multiple tiers).

- For similar reasons, a **SOA (Service Oriented Architecture)** has been chosen for the communication of the application server with the front ends. This improves flexibility, through modularity and a clear documentation, and simplicity, through an higher abstraction of the components.
- The entire system is designed within the principles of **Modular programming**, focusing on assigning each functionality to a different module. This greatly improves extensibility and flexibility, and allows for an easy implementation of the APIs.
- The **Client&Server** logic is the most common, simple way to manage the communication both between client and application server, and between application server and database.
- The **MVC (Model-View-Controller)** pattern, besides being a common choice in object-oriented languages like Java, allows for a clear logical division of the various elements of the program.
- The **Adapter** pattern is largely used for portability and flexibility of the various modules.

2.8 Other design decisions

The system uses **Google Maps** to perform all the operations related to maps, i.e. map and position visualization, geolocalization (either through GPS or user input) as well as distance, route and ETA calculations. This is an easy, fast to develop solution that relies on a worldwide, well-known and well-established software.

Chapter 3

Algorithm Design

The most noteworthy algorithms used by myTaxiService are the following:

3.1 Queue management algorithm

The city is divided in a certain number of taxi zones. To each zone, a local, FIFO managed taxi queue is assigned, used by the system in order to handle all the available taxi drivers that are currently located in it [see RASD for further explanations]. Whenever certain events occur, the system launchs a *queue management algorithm* to modify the elements and/or the order of the taxi queue. Events and their consequences are described in the following table:

[Legend]

- TD_i: taxi driver, **currently** located in zone i;
- RR_i: taxi ride request, with starting point in zone i;
- Q_i: taxi queue, related to zone i;

Event	**Consequences**
TD_z logs in	TD_z is pushed in Q_z TD_z's status = "Ready"
RR_z incomes	RR_z is forwarded to TD_z on top of Q_z
TD_z on top of Q_z accepts RR_z	TD_z is removed from Q_z TD_z's status = "Busy"
TD_z on top of Q_z refuses RR_z	TD_z is removed from Q_z TD_z is pushed in Q_z repeat "RR_z incomes"
TD_z ends a ride	TD_z is put on bottom of Q_z TD_z's status = "Ready"

TD_z ends his/her workshift
or
TD_z logs out
	if (TD_z's status == "Ready")
	{TD_z is removed from Q_z}
	TD_z's status = "Offline"

3.2 Grouping algorithm

Shared rides requests can be grouped and forwarded as a single one, provided that certain requirements about the requests info are met [see RASD for further explanations]. At the time of forwarding a shared ride request, the system launchs a *grouping algorithm* to identify other ride requests eligible for grouping with the first one. In case there are some, the algorithm proceeds with the grouping; otherwise, it takes up to one minute to wait for new ones. If even after a minute no eligible requests are found, the system simply forwards the shared ride request as a single, non-shared request. The following contraints describe in detail the requirements for two (or more) ride requests to be eligible for grouping:

[Legend]

- SRR: pending shared ride request;
- $SRR_i \star SRR_j$: pending grouped ride request;
- $SRR.startLocation$: ride starting location, indicated in SRR;
- $SRR.startDate$: ride starting date and time, indicated in SRR;

- $SRR.duration$: ride estimated duration, calculated from SRR;
- $SRR.passengers$: ride total number of passengers, indicated in SRR;
- $distanceConstraint$ [meters]: variable that expresses the maximum acceptable distance between the starting locations of two different shared ride requests in order for them to be eligible for grouping ("similar starting locations");
- $dateConstraint$ [minutes]: variable that expresses the maximum acceptable time interval between the starting dates of two different shared ride requests in order for them to be eligible for grouping ("similar date and time");
- $directionConstraint$ [percentage]: variable that expresses the maximum acceptable percentage of additional duration of a ride, in case its ride request has been grouped with another one ("similar directions");
- $taxiCapacityConstraint$ [int]: variable that expresses the maxinum number of seats available in a taxi; if the sum of all passengers of two different shared ride requests is higher that this number, the two requests are not eligible for grouping ("enough seats available");

[Requirements for grouping]
$\forall i, j (i \neq j)$:

1. $|SRR_i.startLocation - SRR_j.startLocation| \leq distanceConstraint$
2. $|SRR_i.startDate - SRR_j.startDate| \leq dateConstraint$
3. $SRR_i.destination \leq (SRR_i \star SRR_j).destination(1+directionConstraint)$
4. $SRR_i.passengers + SRR_j.passengers \leq taxiCapacityConstraint$

Note: in case, at any time, there are more than two eligible shared rides requests to be grouped with each other, the algorithm groups two of them, then tries to group a third request to the first two (checking again requirement 4.), and so on. For example, let's consider the case where

- $SRR_1 \star SRR_2$
- $SRR_1 \star SRR_3$
- $SRR_2 \star SRR_3$

are all eligible groups. The algorithm groups SRR_1 and SRR_2, then tries to group $(SRR_1 \star SRR_2)$ with SRR_3. If requirement 4. is fullfilled, it creates $(SRR_1 \star SRR_2 \star SRR_3)$, otherwise forwards $(SRR_1 \star SRR_2)$ and SRR_3 separately.

3.3 Forwarding algorithm

Rides requests are forwarded in different ways, according to their type. In particular, reserved ride requests are forwarded only 10 minutes before the starting time indicated in the request, while shared ride requests need the grouping algorithm to be launched beforehand [see RASD for further explanations]. Note that this means that ride requests that are both shared *and* reserved follow both sets of rules: their design admittedly favors the certainty of the reserving to the possibility of sharing. In any case, at the time of forwarding any kind of ride request, the system launchs a *forwarding algorithm*.
Request types and their management are described in the following table:

[Legend]

- TD_i: taxi driver, **currently** located in zone i;
- RR_i: taxi ride request, with starting point in zone i;
- $RR_i.startDate$: ride starting date and time, indicated in RR_i;
- Q_i: taxi queue, related to zone i;

Incoming request type	**Forwarding**
Standard RR_z	Standard forwarding until accepted
Reserved RR_z	*(at $RR_z.startDate - 10 minutes$)* Standard forwarding until accepted
Shared RR_z	apply Grouping algorithm *(up to 1 minute)* *if non-grouped*: Standard forwarding until accepted *if grouped*: Standard forwarding for 3 minutes *if not accepted within 3 minutes*: repeat from beginning*
Reserved&Shared RR_z	*(at $RR_z.startDate - 10 minutes$)* apply Grouping algorithm *(up to 1 minute)* *if non-grouped*: Standard forwarding until accepted *if grouped*: Standard forwarding for 3 minutes *if not accepted within 3 minutes*: repeat from beginning*

(*) **Note**: when repeating the grouping algorithm, all groups of shared ride requests that have already been tried and not accepted within three minutes are discarded from the possible groupings.

3.4 Fee calculation algorithm

Whenever a taxi driver reaches the destination of one or more passengers during a shared ride, be it the last stop of the ride or not, all passengers that are going to get off the taxi must pay an equal percentage of the taxi fee, depending on the number of passengers and traveling distance provided in their respective ride requests [see RASD for further explanations]. To do so, whenever a grouped, shared ride request is accepted by some taxi driver, the system launchs a *fee calculation algorithm* to equally split the taxi fee among passenger users, in percentage.
The taxi fee calculation is described by the following formula:

[Legend]

- P: set of all passenger **users** involved in the ride;
- p: generic passenger user $\in P$;
- Fee_p: comprehensive* taxi fee, related to passenger user p;
- $AverageFee_p$: taxi fee per-person*, related to passenger user p;
- $p.numPass$: number of passengers, indicated in p's ride request;
- $p.distance$: distance to travel, calculated from starting location and destination indicated in p's ride request;

$$Fee_p = \frac{p.numPass * p.distance}{\sum\limits_{i \in P} i.numPass * i.distance}\ \%$$

$$AverageFee_p = \frac{TotalFee_p}{p.numPass}\ \%$$

(*) **Note:** Fee_p refers to the total payment of *all* passengers indicated in p's request, while $AverageFee_p$ refers to an equal splitting of Fee_p among *each* passenger in p's request.
Note that $AverageFee_p$ is simply an indication, it is not mandatory that every single passenger pays it exactly, as long as p's group pays Fee_p completely.

Chapter 4

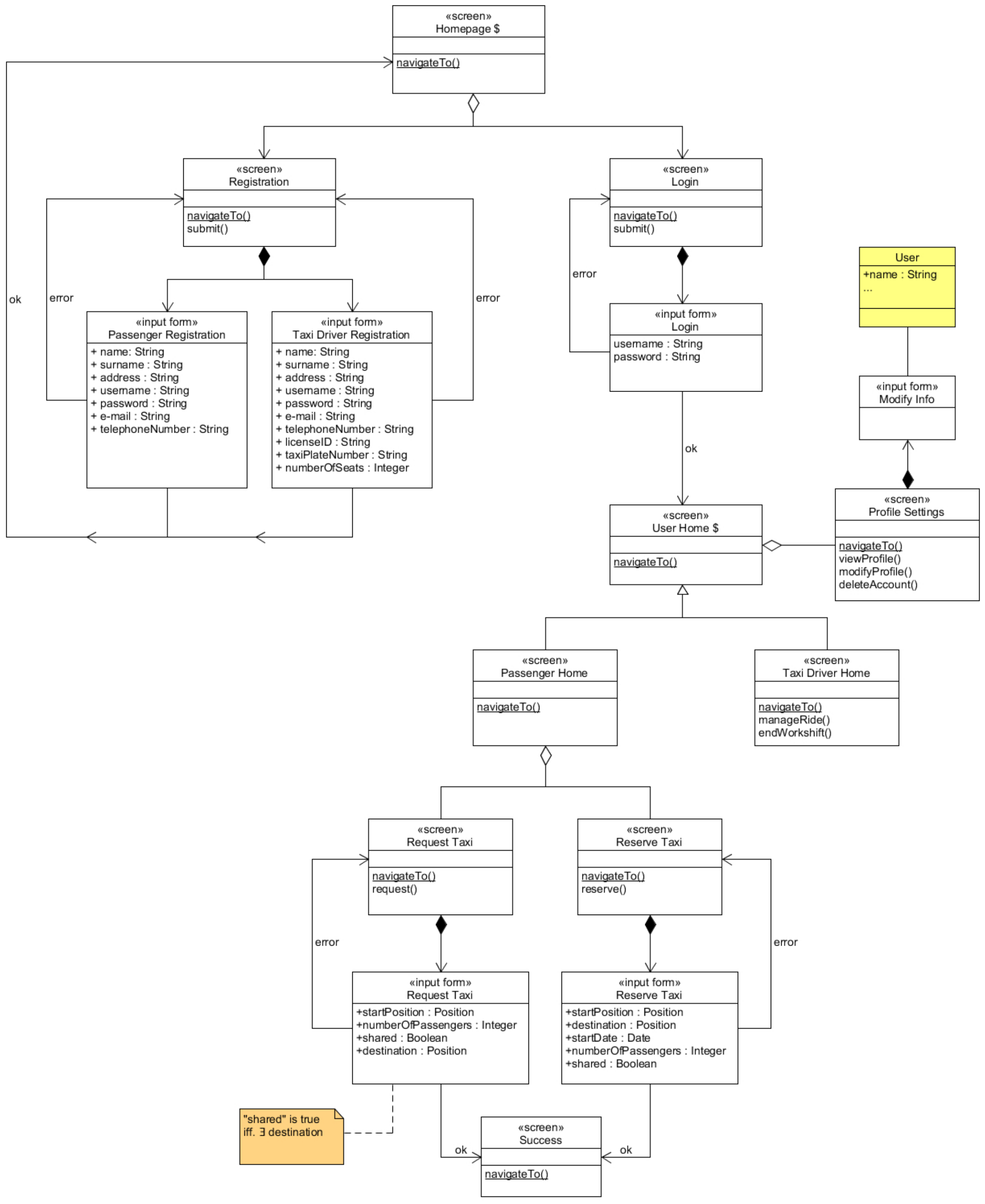
User Interface Design

4.1 Purpose

This chapter gives a description of the user interfaces of the system and the flow from one interface to another. The description includes an UX diagram and some mockups in order to understand how a user can do actions using the interface given by the system.

4.2 UX diagram

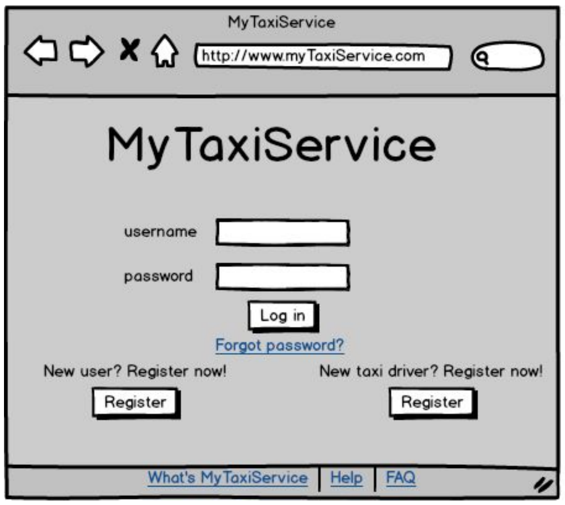
Here is the UX diagram:

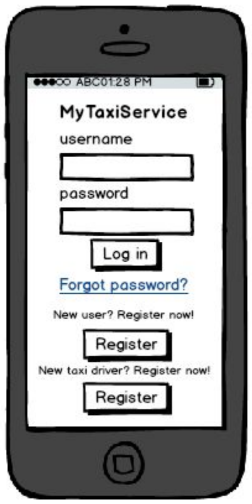

«screen» Homepage $
navigateTo()
«screen» Registration
navigateTo()
submit()
«screen» Login
navigateTo()
submit()
ok
error
error
error
«input form» Passenger Registration
+ name: String
+ surname : String
+ address : String
+ username : String
+ password : String
+ e-mail : String
+ telephoneNumber : String
«input form» Taxi Driver Registration
+ name: String
+ surname : String
+ address : String
+ username : String
+ password : String
+ e-mail : String
+ telephoneNumber : String
+ licenseID : String
+ taxiPlateNumber : String
+ numberOfSeats : Integer
«input form» Login
username : String
password : String
User
+name : String
...
«input form» Modify Info
ok
«screen» User Home $
navigateTo()
«screen» Profile Settings
navigateTo()
viewProfile()
modifyProfile()
deleteAccount()
«screen» Passenger Home
navigateTo()
«screen» Taxi Driver Home
navigateTo()
manageRide()
endWorkshift()
«screen» Request Taxi
navigateTo()
request()
«screen» Reserve Taxi
navigateTo()
reserve()
error
error
«input form» Request Taxi
+startPosition : Position
+numberOfPassengers : Integer
+shared : Boolean
+destination : Position
«input form» Reserve Taxi
+startPosition : Position
+destination : Position
+startDate : Date
+numberOfPassengers : Integer
+shared : Boolean
"shared" is true iff. ∃ destination
«screen» Success
navigateTo()
ok
ok

4.3 Mockups

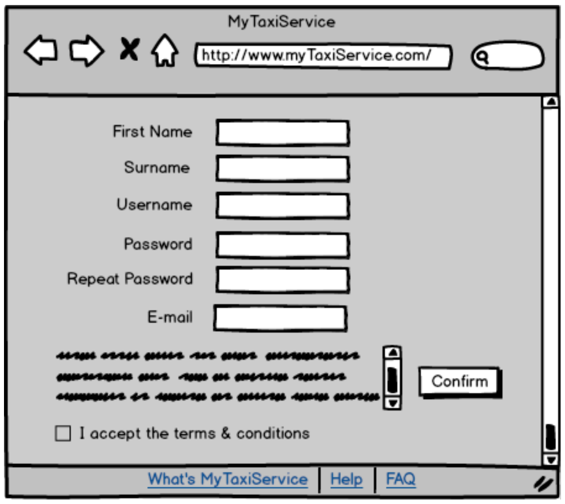
Here we will presented some mockups of MyTaxiService. Some of them referring to the RASD document Section 3.1.1 User Interface.

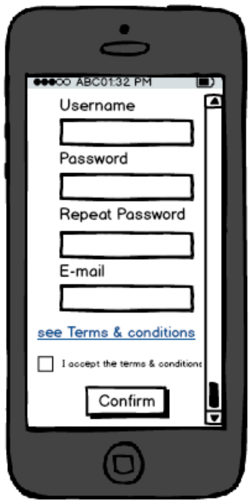
Log in: In the figure below is shown MyTaxiService's homepage



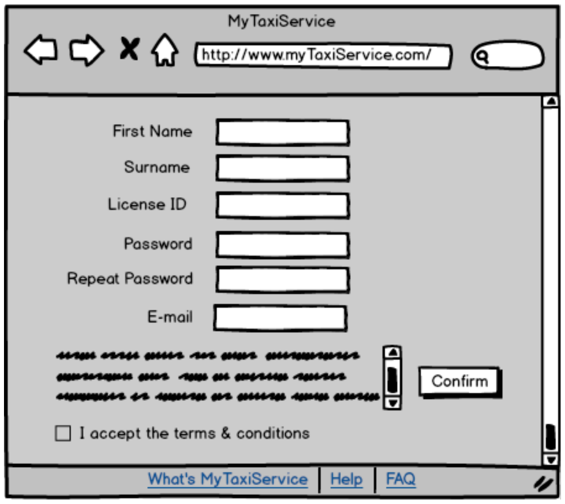


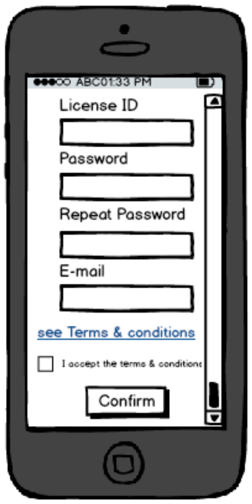
Registration passenger: View of the visitor that wants to register as a passenger



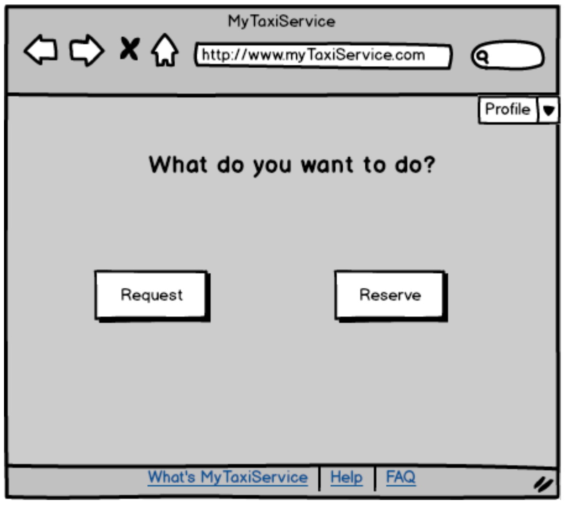


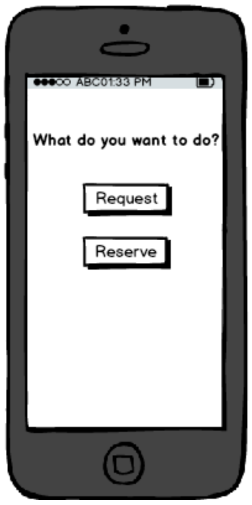
Registration taxi Driver: View of the visitor that wants to register as a taxi driver



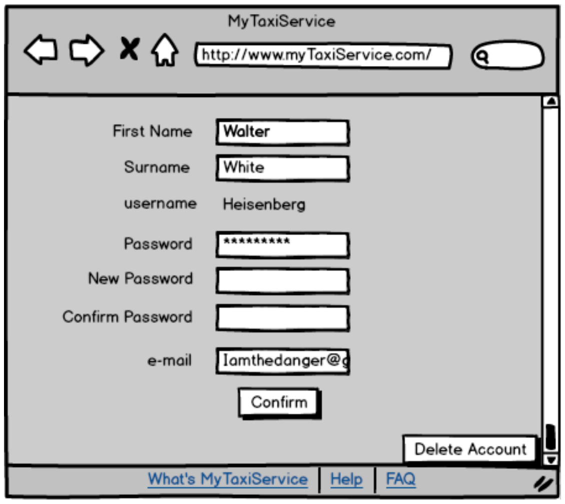


Passenger view: View of the passenger

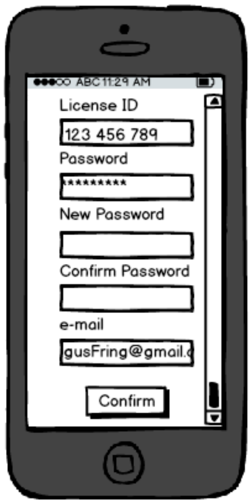




Profile: View of the profile of the user

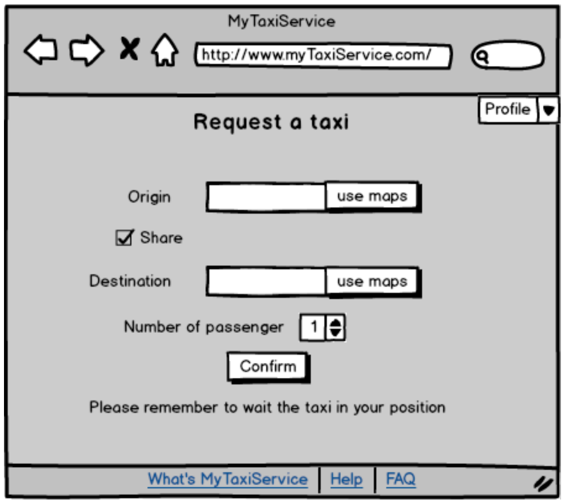


First Name	Walter
Surname	White
username	Heisenberg
Password	*********
New Password	
Confirm Password	
e-mail	Iamthedanger@g

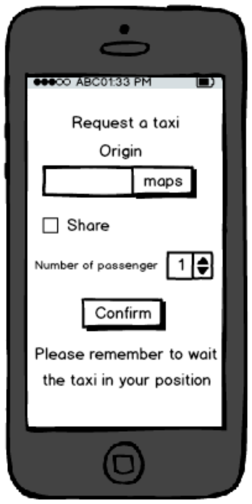


License ID
123 456 789
Password
****
New Password
Confirm Password
e-mail
gusFring@gmail.c

Request a taxi: View of the passenger when he/she requests a taxi

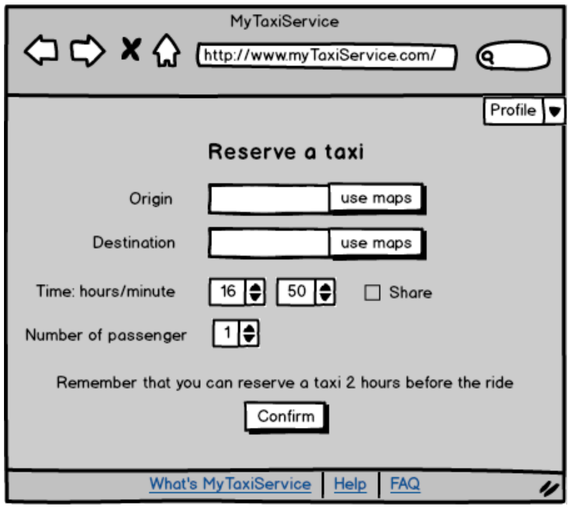


Please remember to wait the taxi in your position

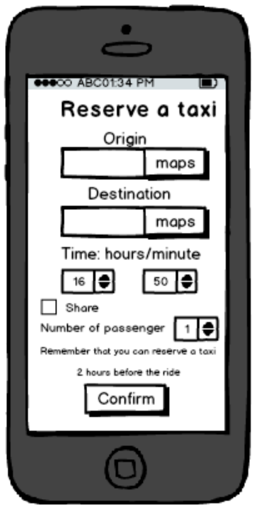


Please remember to wait the taxi in your position

Reserve a taxi: View of the passenger when he/she reserves a taxi

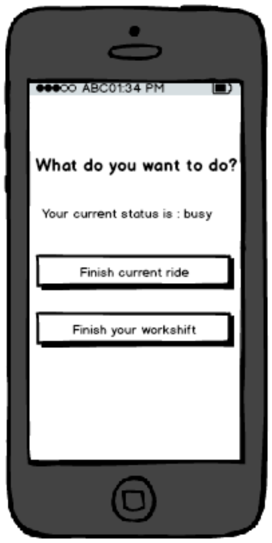


Remember that you can reserve a taxi 2 hours before the ride



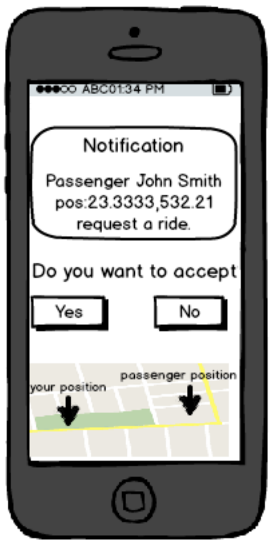
Remember that you can reserve a taxi 2 hours before the ride

Taxi driver view: View of the taxi driver



Your current status is : busy

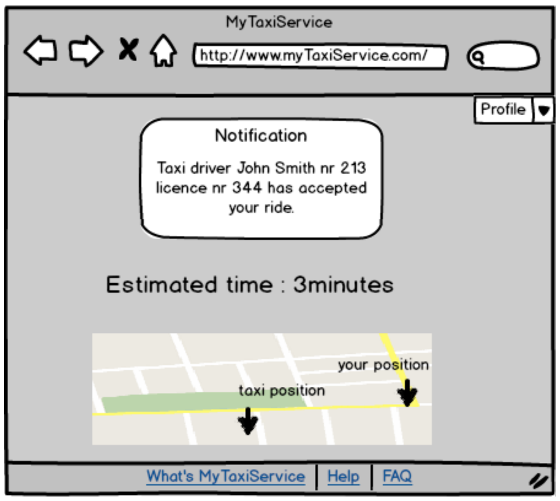
Taxi driver notification: Notification that the taxi driver, choosen by the system, sees when a passenger request a ride.


Notification

Passenger John Smith pos:23.3333,532.21 request a ride.

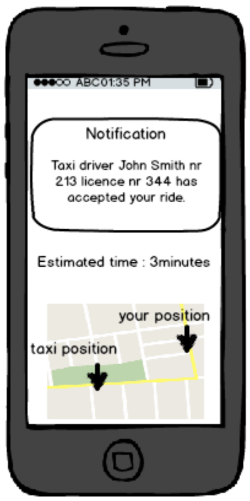
Do you want to accept

Passenger notification : Notification that the passenger see when a taxi accept the ride


Notification

Taxi driver John Smith nr 213 licence nr 344 has accepted your ride.

Estimated time : 3minutes


Notification

Taxi driver John Smith nr 213 licence nr 344 has accepted your ride.

Estimated time : 3minutes

Chapter 5

Requirements Traceability

1. **Visitors can register either as passengers or as taxi drivers**
   Account manager receives user registrations from web or mobile applications through Account API. User informations are stored by myTaxiDatabase through Data API of Data Manager.

2. **Visitors can abort the registration process at any time.**
   Account manager does not consider unfulfilled registrations. No data is stored.

3. **The link in the confirmation email must be clicked within 1 day, otherwise the registration is deleted along with the visitor's info.**
   Account manager takes care of sending the confirmation email to new user. User informations will be deleted from myTaxiDatabase if the registration has not been confirmed.

4. **Registration form requires user's info (fields). All fields must be contain valid inputs (see RASD for further informations)**
   Account manager does not consider unfulfilled registrations.

5. **Email address and username cannot be the same as ones from other myTaxiService users**
   Account manager ensures the uniqueness of users.

6. **Password must contain at least 8 characters**
   Validation is made by client apps. Further controls are made by Account manager.

7. **Password and password confirmation must match.**
   Two times password providing is requested by client apps to prevent typos.

8. **Visitors can log in either as passengers or as taxi drivers**
   Account manager allows access to service only to registered user.

9. **Visitors must fill the "username" field either with an existing username or with an existing email address in order to successfully log in**
   Account manager accepts either username or email in order to login.

10. **Visitors must fill the "password" field with the only password corresponding to the submitted username/email address in order to successfully login**
    Account manager checks the submitted password with the one associated with user.

11. **The system will ignore log in requests if at least one of the "username" and "password" fields are left blank**
    Blank fields let the client app to ignore the login.

12. **The system allows visitors to retrieve their password if they forget it, by clicking "Forgot password?"**
    Client apps provide this functionality. Account manager will take care of the recovery procedure.

13. **The system requires visitors to submit an existing email address in the "username" field in order to retrieve their password**
    To recover a password client apps require the email associated with the user.

14. **The system will take care of assigning the user a new password, when he/she states to have lost the previous one**
    Account manager generates a new password that will replace the older one.

15. **The system will take care of sending to the email address submitted by the visitor the new assigned password, when he/she states to have lost the previous one**
    Account manager sends a mail to the user with the new password.

16. **The system allows visitors to retrieve their password once a day**
    Account manager permits a password recovery once a day.

17. **The system allows standard taxi ride requests to passenger users**
    Client apps allow the user to request a standard ride. Further control will be made by Ride manager during ride request procedure.

18. **The system allows standard taxi ride requests both on the web and on the mobile application**
    Standard ride requests are allowed from either web or mobile applications.

19. **The system allows taxi ride requests if and only if the passenger accepts to give info about his/her location, either through GPS or directly writing down a valid location**

Client apps check if a location from GPS or input is valid. This is made by implementing Google Maps API.

20. **The system allows taxi ride requests if and only if the passenger can be located in some definite position of some definite taxi zone**
Ride manager checks, through Map services, if a position is in a definite tazi zone.

21. **The system uses default values for the number of passengers and sharing preferences of a ride (1 person, no sharing), unless the passenger does specify them**
Client apps let that if a preference for a ride request is untouched, default value is used.

22. **The system uses a FIFO policy to manage forwarding of pending ride requests**
Ride manager handles forwarding of pending ride requests through a FIFO policy.

23. **The system uses a FIFO policy to manage the order of taxi drivers in queues to send notifications to**
Taxi queues handles the order of taxi drivers through a FIFO policy.

24. **The system forwards a ride request to the first taxi driver in the considered zone queue if and only if he/she has a sufficient number of free seats available in his/her vehicle**
Ride manager finds the first available cab with the corresponding number of free seats from queues provided by Taxi queues.

25. **The system keeps the passenger(s) notified about the status of the ride request he/she sent**
Notification component keeps the passenger up-to-date by requests from Ride manager.

26. **Once a ride request has been accepted by some taxi driver, the system changes the request status from "Pending" to "Accepted"**
Ride request ensures the passenger to be updated about the ride status by Notification component.

27. **Once a ride request has been accepted by some taxi driver, the system calculates the ETA of the incoming taxi based on the distance between the taxi and the passenger(s), and the current traffic**
Ride manager asks Map services to calculate the ETA.

28. **Once a ride request has been accepted by some taxi driver, the system notifies the passenger(s) about the ETA of the incoming**

taxi
Ride manager asks Notification to send ETA to the passenger.

29. **Once a ride request has been accepted by some taxi driver, the system keeps the passenger(s) notified about the current location of the incoming taxi, showing its position on a map**
Ride manager keeps the passenger updated about taxi driver current position through Map services and Notification.

30. **Once a ride request has been accepted by some taxi driver, the system prevents the passenger(s) to make a new ride request until the taxi driver changes the status of the ride to "Completed"**
Client apps prevent user from making new requests if there's a pending one.

31. **The system allows reserved taxi ride requests to passenger users**
Client apps allow user to request a reserved ride.

32. **The system allows reserved taxi ride requests both on the web and on the mobile application**
Reserved ride requests are allowed from either web or mobile applications.

33. **The system allows reserved taxi ride requests if and only if the passenger gives definite existing positions of some definite existing taxi zones both for "Origin" and "Destination" fields**
Client apps check if a location from GPS or input is valid. This is made by implementing Google Maps API.

34. **The system allows passengers to select locations either through GPS or directly writing down a valid location**
Client apps allow user to select location either through GPS or user input.

35. **The system allows reserved taxi ride requests if and only if the passenger gives complete info about the date and the time of the meeting**
Client apps check if date and time informations are valid.

36. **The system allows reserved taxi ride requests if and only if the time of the request occurs at least two hours before the ride meeting time**
Client apps check if a reserved ride is scheduled two hours later from request.

37. **The system forwards notifications to taxi drivers about reserved taxi ride requests 10 minutes before the ride meeting time**
Ride manager ensures that a taxi driver is informed 10 minutes before the reserved ride.

38. **The system allows taxi ride sharing to passenger users**
Client apps allow user to request a shared ride.

39. **The system allows taxi ride sharing both on the web and on the mobile application**
    Shared ride requests are allowed from either web or mobile applications.

40. **The system allows both standard shared taxi ride requests and reserved shared taxi ride requests, simply ticking the "Share" checkbox while doing any kind of taxi ride request**
    Client apps allow standard and reserved ride request to be shared.

41. **Both kinds of shared ride requests require no less data than their non-shared forms (see "Standard ride request" and "Reserved ride request")**
    Client apps ensure the completeness of informations for ride requests.

42. **The system allows standard shared taxi ride requests if and only if the passenger also gives a definite existing position of some definite existing taxi zone as "Destination" (reserved taxi ride requests already require this, even if non-shared)**
    Client apps check if a location from GPS or input is valid. This is made by implementing Google Maps API.

43. **The system allows passengers to select the input for "Destination" either through GPS or directly writing down a valid location**
    Client apps allow user to select location either through GPS or user input.

44. **The system, at the time of forwarding any shared ride request, will use a specific algorithm to calculate good arrangements between different shared ride requests**
    Ride manager calculates good arrangements for shared request.

45. **The arrangement algorithm search different shared ride requests to form a single, grouped, shared ride request**
    Ride manager searches different shared ride requests to form a single, grouped, shared ride request.

46. **The arrangement algorithm considers for grouping only shared rides with close "Origin" positions and similar directions towards "Destination" positions. Similar directions means that going from the origin to the farthest destination implies passing by the other destination(s) as well**
    Ride manager uses Map services and Taxi queues components to take advantage of the algorithm.

47. **The arrangement algorithm also considers the total number of passengers in a taxi. Groups that would occupy too many seats are not eligible**
    Ride manager considers total number of passengers in a taxi.

48. **The arrangement algorithm can group even standard and reserved ride requests together, as long as all the others requirements are met**
    Ride manager can group different types of ride requests.

49. **The arrangement algorithm may take up to 1 minute to wait the reception of new shared ride requests that satisfy the grouping requirements** Ride manager may take up to 1 minute to wait the reception of new shared ride requests.

50. **The system will consider group requests as a single one when forwarding them to taxi drivers**
    Ride manager forwards a single request for grouped requests.

51. **Taxi drivers can see all details of all shared requests when receiving notification of a grouped request**
    Ride manager takes care of informing taxi driver of all details of a grouped request. Notification will dispatch the message.

52. **Taxi drivers can accept or refuse grouped requests as normal**
    Mobile app allows a taxi driver to accept or refuse a grouped request.

53. **The system will automatically split up groups of shared requests if they're continuously refused, or simply not accepted, for 3 minutes since their forwarding**
    Ride manager splits up a grouped request if it is continuously refused.

54. **The system will then proceed to recalculate possible groups with the special algorithm, but excluding the arrangements that were already tried**
    Ride manager tries different combinations of groups.

55. **The system calculates how to split taxi fees equally on all passengers on a shared ride, depending on the distance travelled by each one and the number of passengers during each part of the ride**
    Ride manager calculates how to split taxi fees equally among passengers in a grouped ride request.

56. **The system notifies the taxi driver how much of the total fee each passenger will have to pay, in percentage**
    Ride manager notifies the taxi driver of fees' percentages through Notification component.

57. **The system allows taxi drivers to receive ride request notifications on their mobile phone application and respond to them, either accepting or refusing**
    Mobile app allows taxi driver to receive ride requests and respond to them.

58. **The system notifies taxi drivers about all request notifications forwarded to them**
    Notification component takes care of all messages from service to taxi drivers.

59. **Taxi zone queues contain only taxi drivers that currently have status ”Ready”**
    Taxi queues component handles taxi driver that have ”Ready” status through Account manager.

60. **Taxi zone queues contain only taxi drivers that are currently located in that taxi zone**
    Taxi queues component manages a city zone queue with taxi drivers located in that city zone.

61. **Taxi ride notifications show all requesting passengers’ username and position on a map**
    Taxi drivers are able to see on mobile app all requesting passengers.

62. **The system gives taxi drivers one minute to accept or refuse request notifications, otherwise take it as a refusal. This is to avoid long waiting times for passengers**
    Ride manager gives to taxi drivers a window of one minute to accept or refuse a ride request.

63. **Once a ride request has been accepted by some taxi driver, the system changes his/her status to ”Busy” and he/she’s removed from his/her taxi zone queue**
    Account manager takes care of changing taxi driver status from ”Ready” to ”Busy” when he/she accepts a ride request. Taxi queues component will remove him/her from queues.

64. **The system allows taxi drivers to notify the end of the ride, when they’re doing one (i.e. when they accepted a ride request)**
    Mobile application allows taxi drivers to notify the end of ride to the system.

65. **Once a taxi driver notifies the end of a ride, the system changes his/her status to ”Ready” and he/she’s put on the bottom of his/her taxi zone queue**
    Account manager takes care of changing taxi driver status from ”Busy” to ”Ready” when he/she ends a ride. Taxi queues component will add him/her to queues.

66. **The system uses info provided by the GPS to locate taxis and decide their respective queues**
    Location data provided by mobile application are used by Ride manager and Taxi queues components for queues managing.

67. **The system automatically inserts taxi drivers in queues when their status changes to "Ready"**
    Taxi queues adds a taxi driver to a queue if he/she changes him/her status to "Ready".

68. **The system automatically removes taxi drivers from queues when their status changes to "Busy" or "Offline"**
    Taxi queues removes a taxi driver from a queue if he/she changes his/her status to "Busy" or "Offline".

69. **The status automatically changes to "Busy" when the taxi driver accepts a ride request**
    Ride manager changes a taxi driver status to "Busy" when the taxi driver accepts a ride request.

70. **The status automatically changes to "Ready" when the taxi driver notifies the end of a ride**
    Ride manager changes a taxi driver status to "Ready" when the taxi driver ends a ride.

71. **When status is "Ready", the application notifies about ride requests**
    Ride manager notifies taxi drivers, through Notification component, about ride requests.

72. **When status is "Ready", the application enables the taxi driver to accept/refuse requests**
    Mobile app allows taxi drivers to accept/refuse a request if his/her status is "Ready".

73. **When status is "Busy", the application prevents ride requests notifications**
    Ride manager does not notify taxi drivesr, through Notification component, about ride requests.

74. **When status is "Busy", the application enables the taxi driver to notify the end of the current ride**
    Mobile app allows taxi drivers to notify the end of a ride if his/her status is "Busy".

75. **Account settings are available to both passenger and taxi driver users**
    Account manager allows users to access to their profile informations.

76. **Account settings are available both on the web and the mobile application**
    Account settings are allowed from either web or mobile applications.

77. **Account settings are accessible from the start screen of both apps, through the "Profile" button**
Client apps allow users to access account settings.

78. **The system allows users to view all their profile info, submitted during registration**
Client apps allow users to view all their personal informations.

79. **The system allows users to modify all their profile info, submitted during registration, with the only exception of username**
Account manger allows users to edit personal informations with the only exception of username.

80. **Modifying the password requires to write the old one, and the new one twice; if the former password is not correct or if the two new passwords submitted do not match, the system asks for all passwords again and notifies the user**
Account manager allows users to change their passwords.

81. **Modifying the email address, the taxi license ID or the taxi code requires that the new one doesn't match with the one of another registered user**
Account manager ensures the uniqueness of email address, taxi license ID and taxi code.

82. **Modifying the email address requires confirmation through an email sent to the submitted email address**
Account manager requires email confirmation after a change of it.

83. **The system allows users to abort modifications at any time**
Client apps allow users to abort profile editing.

84. **The system allows users to delete their account: confirmation is required to proceed**
Account manager allows users to delete their accounts. Client apps provide this functionality and ask for a confirmation during this process.

Chapter 6

References

- Software Engineering 2 Project AA 2015/2016: 7 Design (parts I and II)
- Software Engineering 2 Project AA 2015/2016: 8 Architectural Styles (part I)
- Software Engineering 2 Project AA 2015/2016: 9 Architectural Styles (part II)

Appendix A

Appendix

A.1 Software and tool used

- LaTeX (http://www.latex-project.org/) : to redact and to format this document
- Balsamiq Mockups (http://balsamiq.com/products/mockups/): to create mockups
- Eclipse Luna (https://eclipse.org/luna/): to draw global use case, class diagrams, sequence diagrams and UX diagram
- StarUML (http://staruml.io/): to draw UML diagrams

A.2 Working hours

This is the time spent for redact the document

- Belluschi Marco : 24 hours
- Cerri Stefano : 24 hours
- Di Febbo Francesco : 24 hours