



Politecnico di Milano

---

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

Computer Science and Engineering

Software Engineering 2 project

# **myTaxiService**

## **Software Design Description**

Version 1

Prof. Elisabetta Di Nitto

Andrea Autelitano Matr. 849869

Marco De Cobelli Matr. 858360

Matthew Rossi Matr. 858880



# TABLE OF CONTENTS

1	INTRODUCTION .....	4
1.1	Purpose.....	4
1.2	Scope.....	4
1.3	Definitions, Acronyms, Abbreviations.....	4
1.4	Reference Documents .....	5
1.5	Document Structure .....	5
2	ARCHITECTURAL DESIGN .....	6
2.1	Overview .....	6
2.2	High level components and their interaction .....	7
2.2.1	Components .....	7
2.2.2	Interfaces .....	7
2.3	Component view .....	10
2.3.1	Client <<UI>>.....	10
2.3.2	Taxi driver <<UI>>.....	10
2.3.3	Manage users accounts.....	11
2.3.4	Manage credit cards .....	11
2.3.5	Manage taxis .....	12
2.3.6	Manage routes .....	12
2.3.7	Manage rides allocation .....	13
2.3.8	Manage rides .....	13
2.3.9	Admin <<UI>> .....	14
2.4	Deployment view.....	15
2.5	Runtime view .....	17
2.5.1	Sign-up .....	18
2.5.2	Login .....	19
2.5.3	Password recovery .....	20
2.5.4	Create route .....	21
2.5.5	Ride allocation .....	22
2.5.6	Payment via Credit Card .....	23
2.5.7	Ride in process .....	24
2.5.8	Route cancelation .....	26
2.5.9	Route modification.....	27
2.5.10	Concluded/Pending ride details .....	28

2.6	Component interfaces.....	29
2.6.1	Client <<UI>>.....	29
2.6.2	Taxi driver <<UI>>.....	29
2.6.3	Manage users accounts.....	29
2.6.4	Manage credit cards .....	29
2.6.5	Manage taxis .....	29
2.6.6	Manage routes .....	29
2.6.7	Manage rides allocation .....	30
2.6.8	Manage rides .....	30
2.6.9	Admin <<UI>> .....	30
2.7	Selected architectural styles and patterns.....	31
2.7.1	Client-Server .....	31
2.7.2	Components and connectors .....	33
2.7.3	Cloud Computing.....	34
2.7.4	Distributed objects .....	36
2.8	Other design decisions .....	39
2.8.1	Technologies .....	39
2.8.2	External interfaces.....	39
3	ALGORITHM DESIGN .....	41
4	USER INTERFACE DESIGN.....	45
4.1	Home page.....	45
4.2	Registration page.....	46
4.3	Login page .....	48
4.4	Menu .....	49
4.5	Settings .....	50
4.6	Ride request.....	52
4.7	Shared ride .....	53
4.8	Taxi request .....	54
4.9	Accepted request .....	54
4.10	Payment.....	55
4.11	User view – Concluded rides .....	56
4.12	User view – Pending rides.....	57
4.13	Administration panel .....	58
5	REQUIREMENTS TRACEABILITY .....	59
5.1	Functions .....	59
5.2	Non-Functional Requirements .....	60

6	REFERENCES.....	61
7	APPENDIX.....	62
7.1	Used tools .....	62
7.2	Attached documents.....	62
7.2.1	Client's document .....	62
7.3	Hours of work .....	62

# 1 INTRODUCTION

## 1.1 Purpose

The Software Design Description (SDD) is a written description of a software product; it is written by a software designer in order to give a software development team overall guidance to the architecture of the software project. A SDD usually accompanies an architecture diagram with pointers to detailed feature specifications of smaller pieces of the design. Practically, the description is required to coordinate a large team under a single vision, needs to be a stable reference, and outline all parts of the software and how they will work.

## 1.2 Scope

The aim of this project is to optimize the taxi service of a large city thanks to the simplification of the access to the service for the users and the fair management of taxi queues.

The registered users will be able to request or book a taxi from either a mobile application or a web site and Taxi drivers will use the mobile application to accept the requests of the users.

The city is divided into zones of approximately  $2\text{km}^2$ , each of these with a different queue of taxi.

The system will choose the taxi from the zone that is nearest to the user that has made the request/reservation. Usually, it's their own zone.

Furthermore, when a taxi becomes available, either because its taxi driver logs into the system or the related taxi driver reports the termination of a ride, its identifier will be added at the end of the queue of the current zone in which he is located.

When the first taxi driver in a queue receives a request, they can freely choose whether to accept it or to refuse it. In the first case, a notification is sent to the client saying that one taxi driver has accepted their request, in the second case the request is sent to the second taxi driver in the queue and so on.

Moreover, there is the possibility to reserve a taxi from a certain origin to a specific destination within two hours before the ride; in this case, 10 minutes before the ride the system will allocate it to the first taxi driver in the queue.

Finally, besides the traditional interfaces, programmatic interfaces are provided in order to enable the development of additional services like the taxi sharing option. This option allows users to share the cost of a ride with other people after specifying the starting point, the destination of the ride, the number of people to carry on and the availability to share the ride.

In any case, the system automatically calculates the route of the taxi toward the destinations using Google Maps and it provides the fee for each person in the taxi and notifies the users and the Taxi driver who has accepted the ride before the departure.

## 1.3 Definitions, Acronyms, Abbreviations

- **Client:** a person that uses the application in order to request rides.
- **Taxi driver:** a person that uses their taxi in order to provide the ride.
- **User:** either a Client and a Taxi driver.
- **Guest:** a person that wants to register to the application or a user before login.
- **Route:** the ride that has been booked by a Client.
- **Ride:** the ride that has been allocated to a Taxi driver and that can be composed of a single route or, in the case of a shared ride, of different routes.
- **Zone:** a virtual square area of  $2\text{ km}^2$ . The number of zones created will cover the whole city in which the application *myTaxiService* will be used.

- **Compatibility among itineraries:** in the case of a shared ride two or more itineraries of different Clients are considered compatible when the origin zone is the same and the route is totally or partially overlapped. Compatibility can be determined using the algorithm described in chapter 3 “Algorithm design”.
- **RASD:** Requirements Analysis and Specification Document.
- **SDD:** Software Design Description.

## 1.4 Reference Documents

- Description of the problem: Assignments 1 and 2 (RASD and DD).pdf
- Requirements Analysis and Specification Document: myTaxiService\_RASD v2.0.pdf
- IEEE Std 1016-2009 IEEE Standard for Information Technology - System Design - Software Design Description
- ISO/IEC/IEEE 42010:2011 (E) Systems and software engineering – Architecture description

## 1.5 Document Structure

The contents and the organization of this document are:

- *Chapter 1: Introduction.*  
It gives a description of the document and some basic information about the software.
- *Chapter 2: Architectural design.*  
It describes all the architectural styles and the design choices that we have made during the development of the software.
- *Chapter 3: Algorithm design.*  
It contains a thorough description of the main functionalities of the software using pseudocode.
- *Chapter 4: User interface design.*  
It provides an overview on how the user interfaces of the system will look like.
- *Chapter 5: Requirements traceability.*  
It explains how the requirements we have defined in the RASD can be linked to the design elements that we have defined in this document.
- *Chapter 6: References.*  
It lists the sources used to write some parts of this document.
- *Chapter 7: Appendix.*  
It contains information about the used tools, attached documents and hours of work.

## 2 ARCHITECTURAL DESIGN

### 2.1 Overview

This is the main chapter of the Design document. It includes the description of all the choices we have made during the design process of *myTaxiService*.

The first paragraph, **High level components and their interaction**, describes the main software components on a high level of abstraction, analysing the functionalities they offer, how they interact and their interfaces.

The second paragraph, **Component view**, offers a more detailed description of the components and their interfaces: the former are decomposed in subcomponents with a lower level of abstraction, while the latter are separated depending on the component they are used to interact with.

The third paragraph, **Deployment view**, describes how the components identified in the previous paragraphs will be physically deployed on hardware.

The fourth paragraph, **Runtime view**, offers the description of the runtime behaviour of the system during the execution of its most relevant functionalities. UML Sequence diagrams are used in order to highlight the interaction among software components.

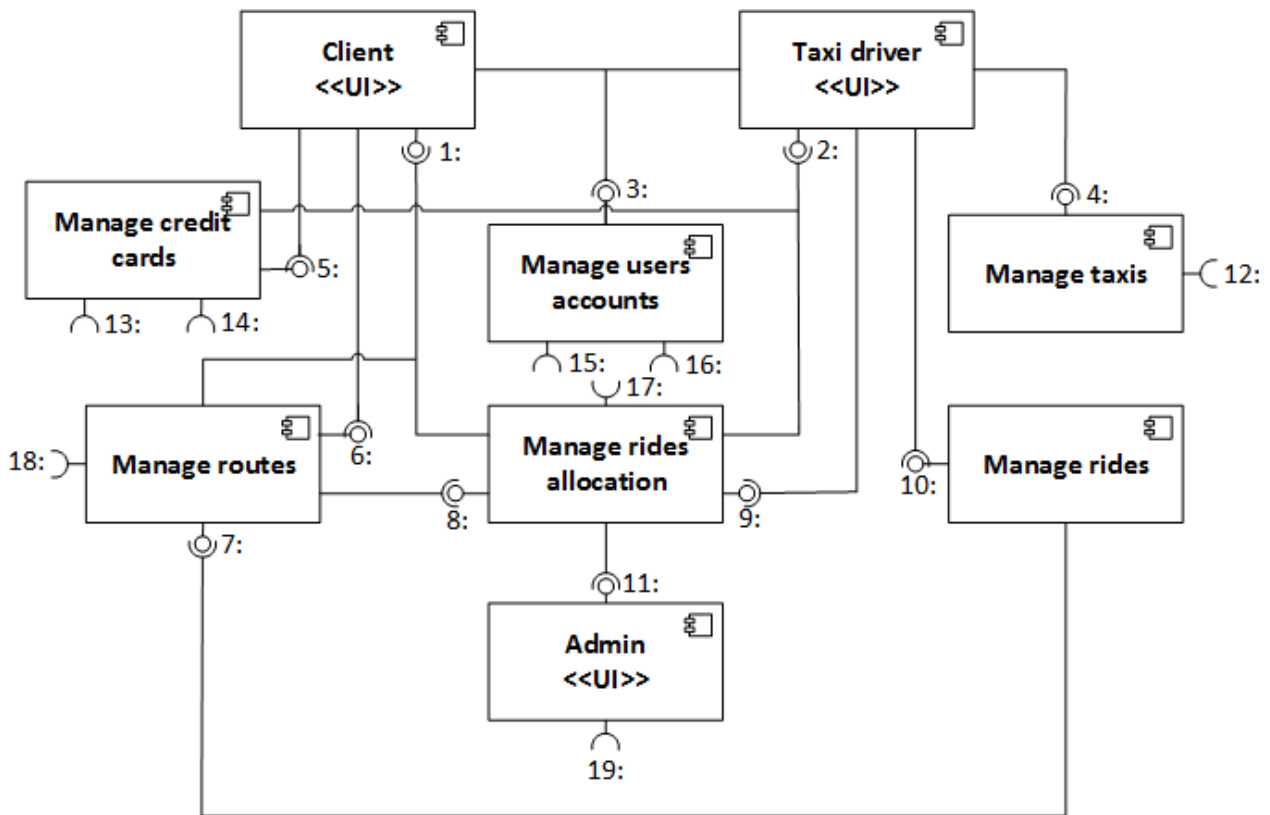
The fifth paragraph, **Component interfaces**, explains in detail the software interfaces and the parameters they will use in the implementation of the source code.

The final two paragraphs, **Selected architectural styles and patterns** and **Other design decisions**, explains the reasons that led us to the choice of the architectural styles and technologies.



## 2.2 High level components and their interaction

The following section describes the macro-components in which the software has been decomposed and the way they interact.



### 2.2.1 Components

1. **Client <<UI>>**, **Taxi driver <<UI>>** and **Admin <<UI>>** manage the presentation level of the application.
2. **Manage users accounts** handles users application access and personal information management.
3. **Manage credit cards** allows a Client to handle their credit cards and pay for the current route.
4. **Manage taxis** allows a Taxi driver to manage taxi information and to check the correctness of the inserted plate.
5. **Manage rides allocation** allocates rides to a Taxi driver choosing them from the zone queue.
6. **Manage routes** allows a Client to request, reserve, view routes and to choose the compatible shared ride they want to join.
7. **Manage rides** shows each Taxi driver their rides and handles the termination of a ride.

### 2.2.2 Interfaces

#### 1, 2: Get push notification interface

Client <<UI>> and Taxi driver <<UI>> both offer an interface to Manage rides allocation to receive push notifications about the allocation of the rides. Client <<UI>> can also be notified by Manage route about last minute changes in the route executed by a Taxi driver. Taxi driver <<UI>> can also be notified by Manage credit cards about successful payments of the current ride.

**Asynchronous:** a notification-based communication is used.

### **3: User account interface**

Client <<UI>> and Taxi driver <<UI>> use Manage users accounts' interface to manage registration, login, logout, personal info modification and password recovery functionalities.

**Synchronous**

### **4: Taxi interface**

Manage taxis offers an interface to Taxi driver <<UI>> to manage addition, visualization, modification and removal of the taxis they have added to the system.

**Synchronous**

### **5: Credit cards interface**

Manage credit cards offers an interface to Client <<UI>> to manage addition, visualization, modification and removal of the credit cards they have added to the system and the payment of the current route.

**Synchronous**

### **6: Route interface**

Manage routes offers an interface to Client <<UI>> to create, modify and delete pending routes and show all the routes they have concluded.

**Synchronous**

### **7: Compute price interface**

Manage routes offers an interface to Manage rides to recalculate the cost of each route when a ride has been stopped because of an anomaly that prevents it to be completed.

**Synchronous**

### **8: Allocate request interface**

Manage rides allocation offers an interface to Manage routes to allow a new request to be allocated.

**Asynchronous:** the two components must be decoupled on a temporal level.

### **9: Taxi driver state interface**

Manage rides allocation offers an interface to Taxi driver <<UI>> to allow a Taxi driver to change their availability state and zone position.

**Synchronous**

### **10: Ride interface**

Manage rides offers an interface to Taxi driver <<UI>> to allow a Taxi driver to examine the rides they have completed and to terminate the current ride.

**Synchronous**

### **11: Send zones queue info interface**

Admin <<UI>> offers an interface to Manage rides allocation to allow the administrator to receive custom parameters about the state of the queues in the system.

**Asynchronous:** Manage rides allocation does not need to be coupled on a temporal level to Admin<<UI>>.

### **12: Plate verification interface**

Manage taxis uses an external system to verify the existence of a car with the plate inserted during the registration phase.

**Synchronous:** the system cannot conclude the registration of the taxi driver if the plate authentication has not been completed.

**13: Credit card verification interface**

Manage credit card uses an external system to verify the correctness of credit cards data inserted by clients.

**Synchronous:** the system cannot conclude the credit card insertion if the credit card authentication has not been completed.

**14: Payment verification interface**

Manage credit card uses an external system to execute payment transactions.

**Synchronous:** the system cannot proceed until it is informed of the outcome of the payment transaction.

**15: Driver license verification interface**

Manage taxi drivers uses an external system to verify the correctness of the driver license inserted during the taxi driver registration.

**Synchronous:** the system cannot terminate the registration of a taxi driver if the driver license authentication has not been completed.

**16: Taxi license verification interface**

Manage taxi drivers uses an external system to verify the correctness of the taxi license inserted during the taxi driver registration.

**Synchronous:** the system cannot terminate the registration of the taxi driver if the taxi license authentication has not been completed.

**17: Compute ride path interface**

Manage rides allocation uses Google Maps Directions API to calculate ride cost, the estimated time of arrival of the Taxi driver and to send the Taxi driver the suggested path.

**Synchronous:** the system cannot terminate the ride allocation otherwise.

**18: Compute shared route path interface**

Manage routes uses Google Maps API to compute, if the route is shared, the compatible rides and associates the route to one of them if the Client decides to.

**Synchronous:** the system cannot terminate the identification of compatible rides and calculate price of each route otherwise.

**19: Administration interface**

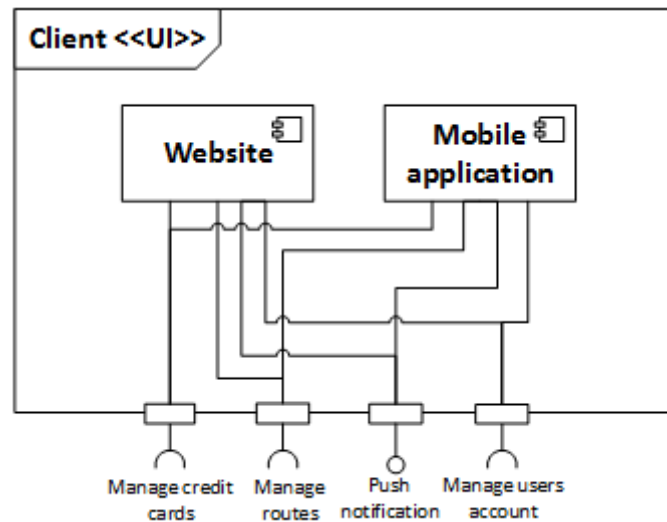
Admin <<UI>> uses Google Maps API to display on a map the zone division, taxis in queue for each zone and see the path of a selected ride.

**Asynchronous:** information received by Google Maps is not critical for the system's execution; therefore, the administration control panel can operate even if the response time from Google Maps is elevated.

## 2.3 Component view

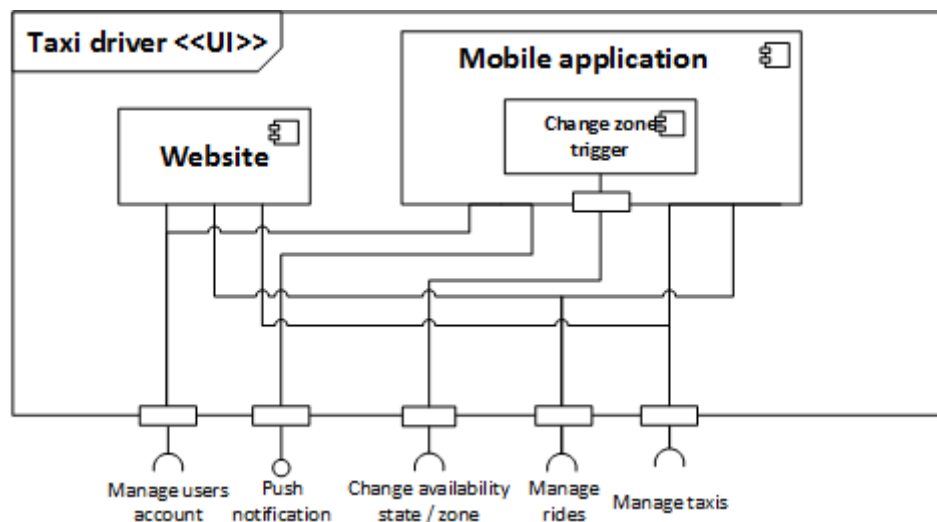
The following section describes the division in sub-components for each macro-component and the way they interact.

### 2.3.1 Client <<UI>>



**Website** and **Mobile application** offer the same functionalities to the Client but the former is accessed when using a browser and the latter using the mobile application.

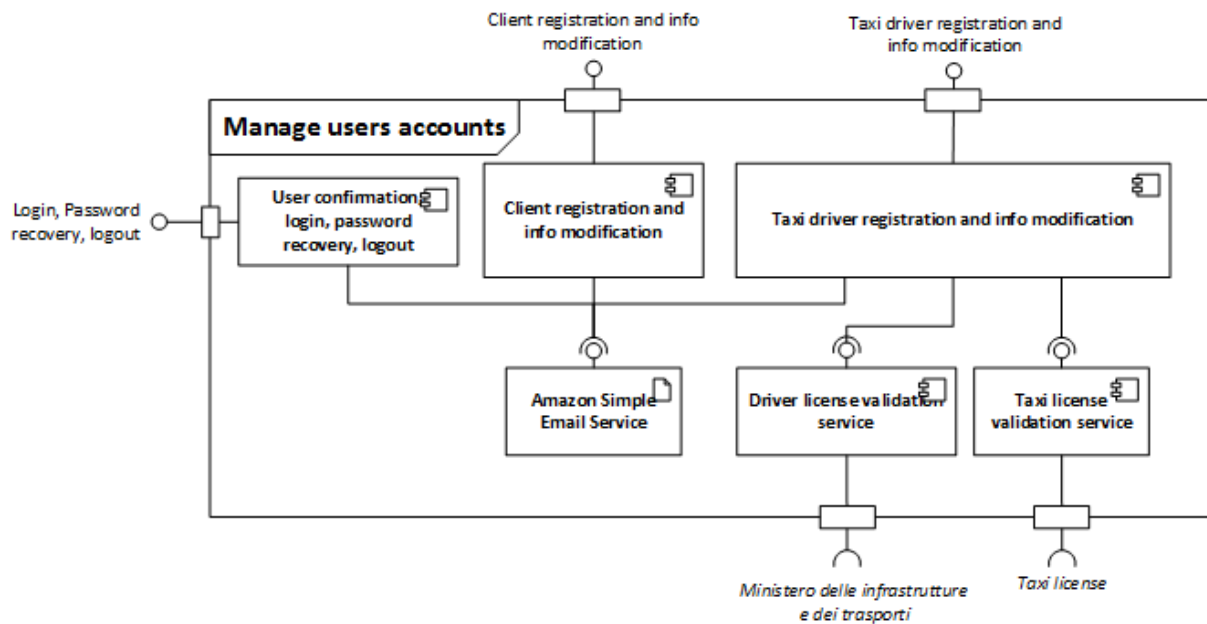
### 2.3.2 Taxi driver <<UI>>



**Website** allows a Taxi driver to manage personal and taxis information, and to see the rides they have completed.

**Mobile application** offers a Taxi driver all the **Website** functionalities and helps them in their work by allowing them to manage pending rides, change availability state and zone position.

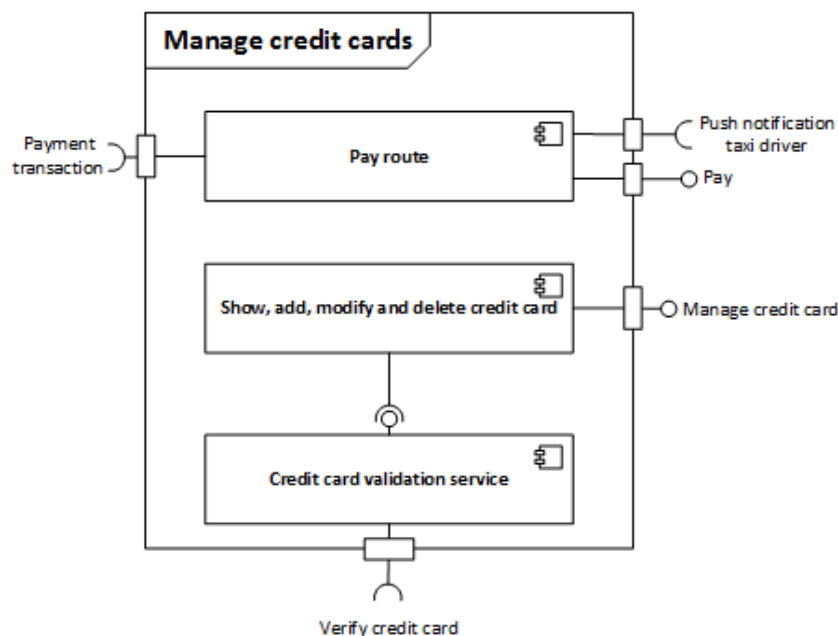
### 2.3.3 Manage users accounts



*Ministero delle infrastrutture e dei trasporti* and *Taxi license* are written in italic because they are both external systems that we have assumed to offer such a service.

**Amazon Simple Email Service** is the service Amazon offers to handle all possible email functionalities: in our case, it is used to send email during the registration phase and the password recovery process.

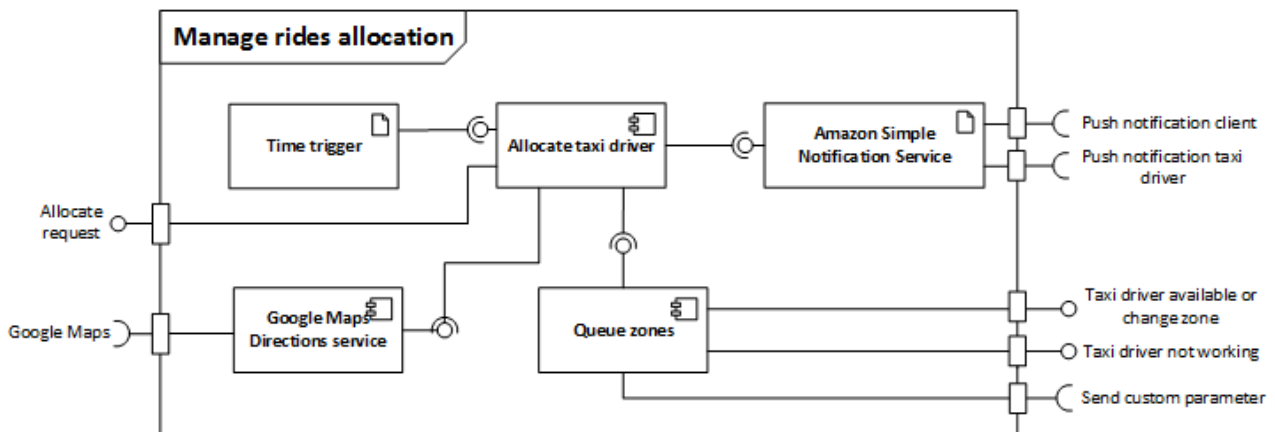
### 2.3.4 Manage credit cards



**Pay route** uses an external secure system that allows the application to handle the payment transactions.



### 2.3.7 Manage rides allocation



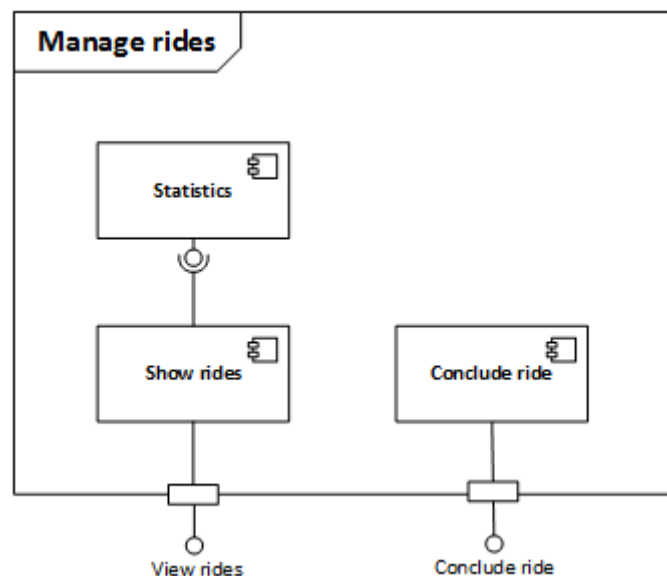
**Time trigger** is used only by reserved rides to call Allocate taxi driver 10 minutes before the ride's starting time (the time in which the Client that has asked the reservation has to be picked up by the Taxi driver). Instead, for route request Manage routes calls Allocate taxi driver during the route creation phase.

**Queue zones** handle the queues of Taxi drivers for each zone.

**Allocate taxi driver** interacts with Queue zones to retrieve the first available Taxi driver; it also sends push notification to the chosen Taxi driver and to the Client through Amazon SNS and uses **Google Maps Directions** service to estimate the time of arrival.

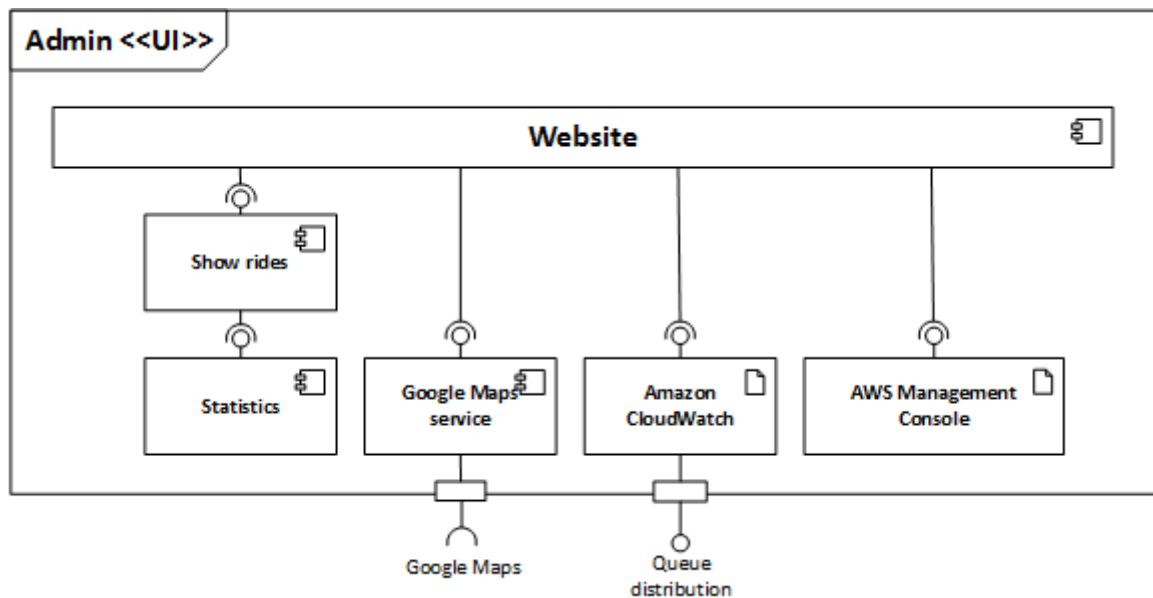
**Amazon Simple Notification Service** is a web service that coordinates and manages the delivery or sending of messages to subscribing endpoints or clients and can be used to send push notification messages.

### 2.3.8 Manage rides



**Statistics** is the component that allows a Taxi driver to analyse concluded rides in an aggregate way; the result is shown using a simple graph on the Taxi driver <<UI>>.

### 2.3.9 Admin <<UI>>



**Website** integrates all the pieces of information that can be gathered about the application in a unique interface and, using AWS Management Console, gives the Administrator the possibility to manage AWS services.

**Amazon CloudWatch** is the service used to monitor the cloud AWS resources and the applications that are executing on AWS. It is also used to keep track of parameters and to gather and monitor log files.

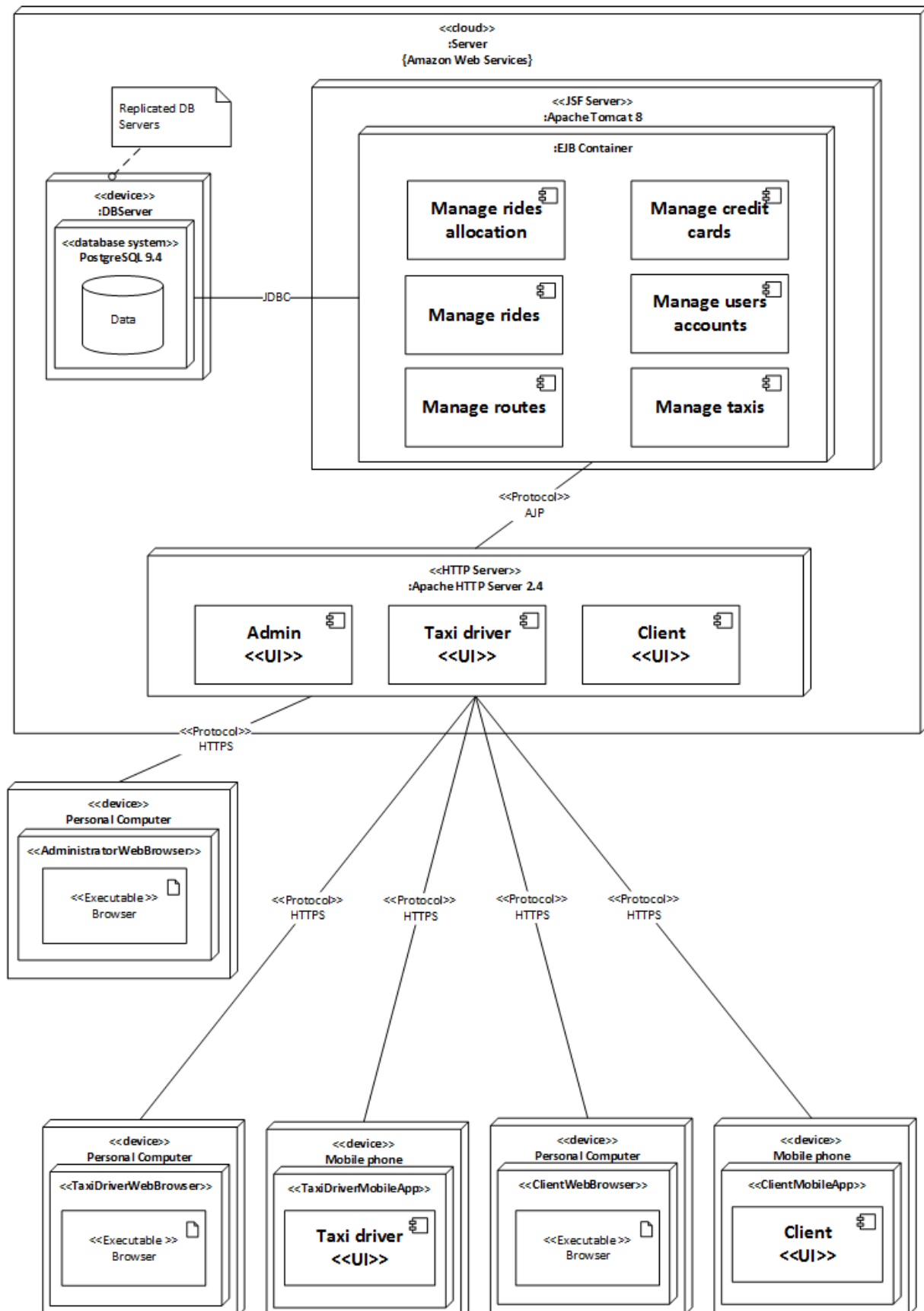
**AWS Management Console** allows the management of AWS through a simple and intuitive web-based user interface.

**Statistics** is the component that allows the Administrator to analyse concluded rides in an aggregate way; the result is shown using a simple graph.



## 2.4 Deployment view

The following section describes the physical deployment of the software components.



Two main groups of nodes can be identified: clients and the cloud server.

Clients can be of two types: either a mobile phone or a personal computer. The main difference between the two is that the user interfaces are already present on the mobile app, while they have to be downloaded from the HTTP server whenever the client or the taxi driver want to use the application via browser.

The "core" of the application is located on Amazon Web Services servers. This node can be divided into three sub-nodes: a Web Server, an Application Server and a Database server.

The Web Server is a HTTP Server: it has to accept HTTP/HTTPS request coming from clients and contains the user interfaces for the web application.

The Application Server is a JSF Server: it manages the EJB Container, where the business logic components are deployed.

The last node is a Database Server running a PostgreSQL RDBMS: this node actually represents DB Servers that are replicated in order to guarantee reliability.

## 2.5 Runtime view

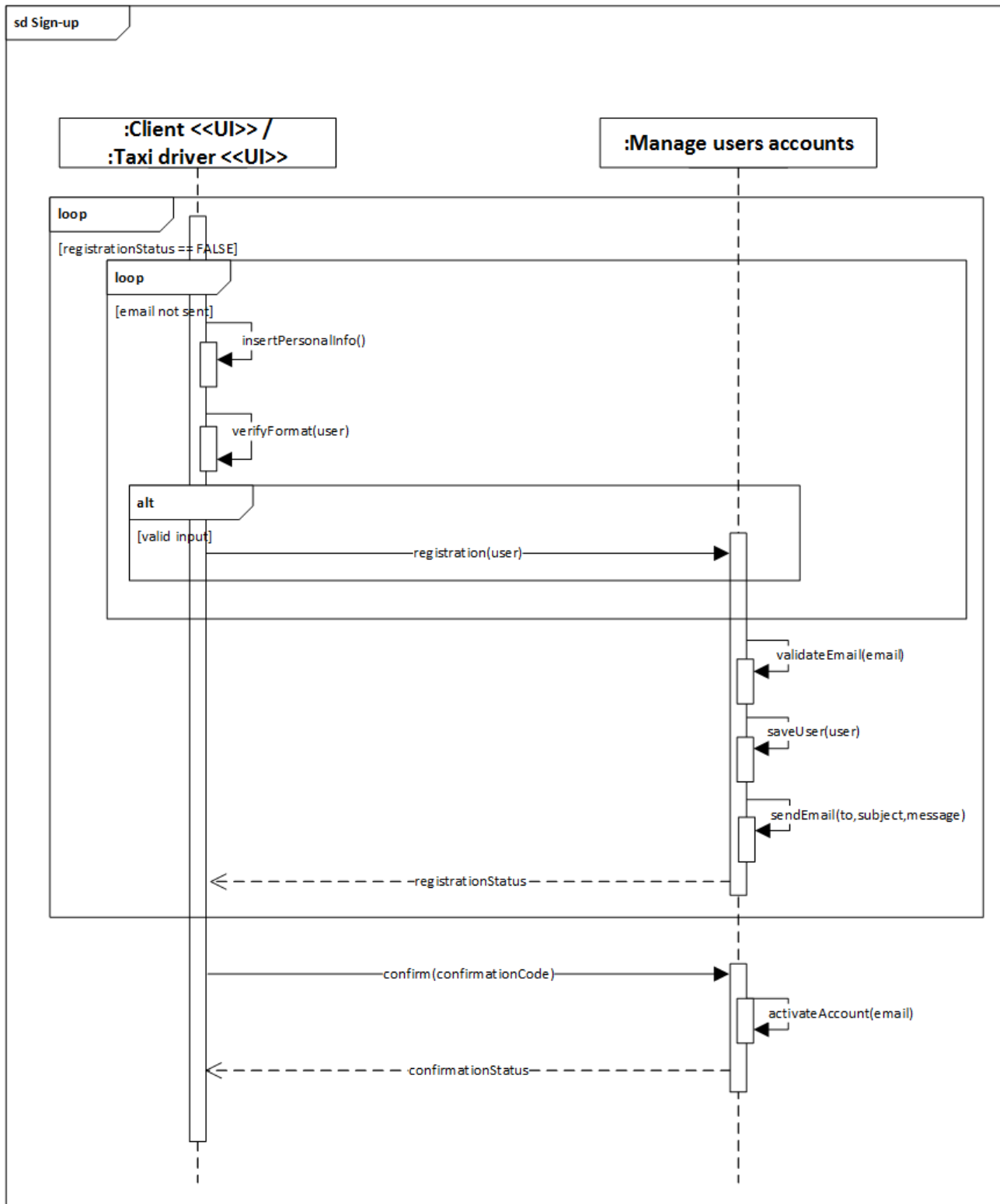
In this sections the interaction between the components described in Chapter 2.2 “High level components and their interaction” will be examined in depth using UML Sequence Diagrams. In the following table, you can find how they are linked to the ones described in Chapter 3.4 “Use cases” of the Requirements Analysis and Specification Document.

<b>SDD</b>	<b>RASD</b>
2.5.1 Sign-up	3.4.1 Sign-up
2.5.2 Login	3.4.2 Login
2.5.3 Password recovery	3.4.3 Password recovery
2.5.4 Create route 2.5.5 Ride allocation	3.4.4 Requesting a ride 3.4.5 Reserving a ride 3.4.6 Shared ride
2.5.6 Payment via Credit Card	3.4.7 Payment via Credit Card
2.5.7 Ride in process	3.4.8 Ride in process
2.5.8 Ride cancelation	3.4.9 Ride cancelation
2.5.9 Ride modification	3.4.10 Ride modification
2.5.10 Concluded/Pending ride details	3.4.11 Concluded/Pending ride details

The main difference between the two versions is that while in the RASD they describe the interaction among physical actors (e.g. a client and a taxi driver), in this document they will describe the interaction between software components: the actor “system” will be replaced by the specific pieces of software that will take part to the operation. In addition to that, we decided to separate the creation of a ride from its allocation to a specific Taxi driver.

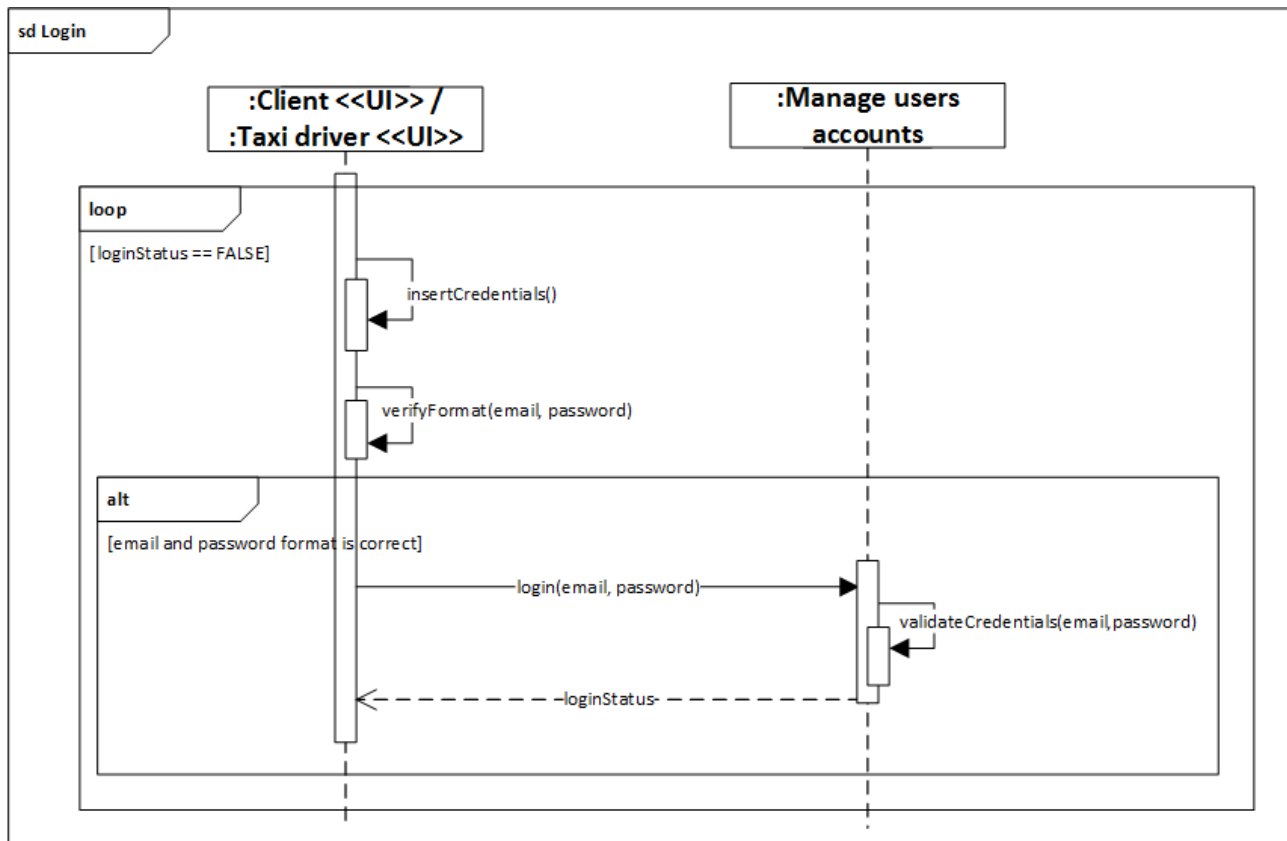
### 2.5.1 Sign-up

This sequence diagram describes the process of registration of a Guest.



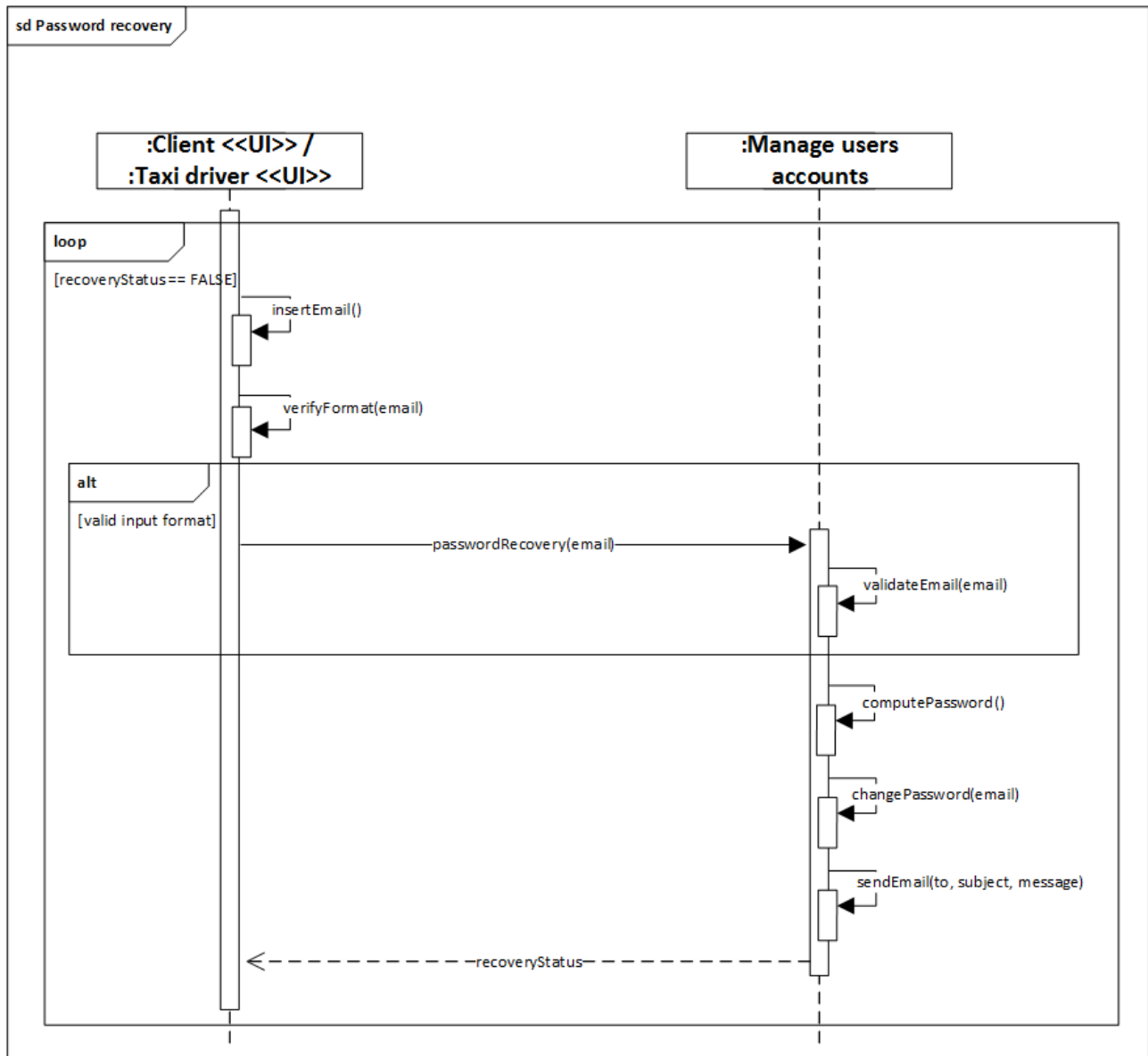
## 2.5.2 Login

This sequence diagram describes the process of login of a User.



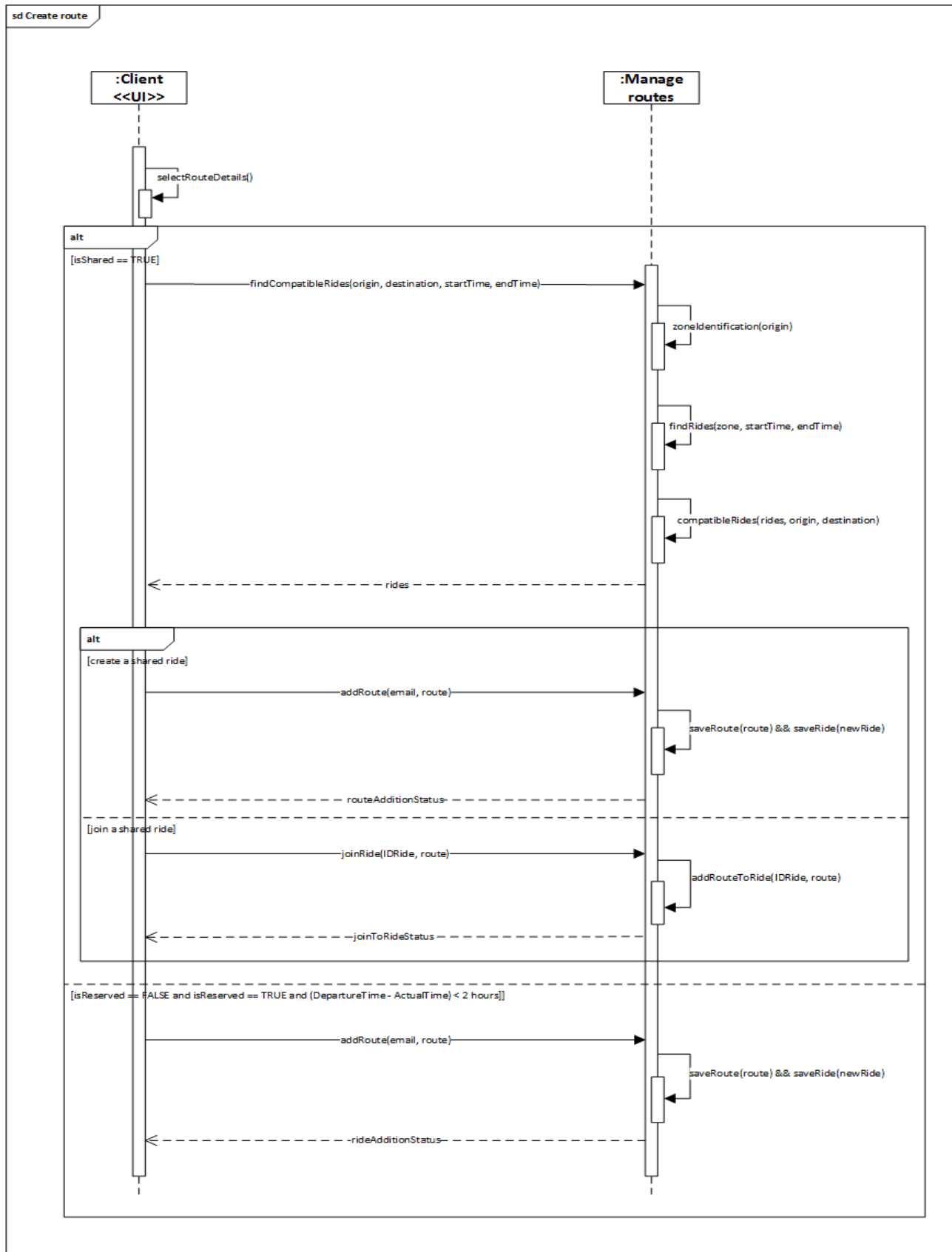
### 2.5.3 Password recovery

This sequence diagram describes the process of password recovery for a User.



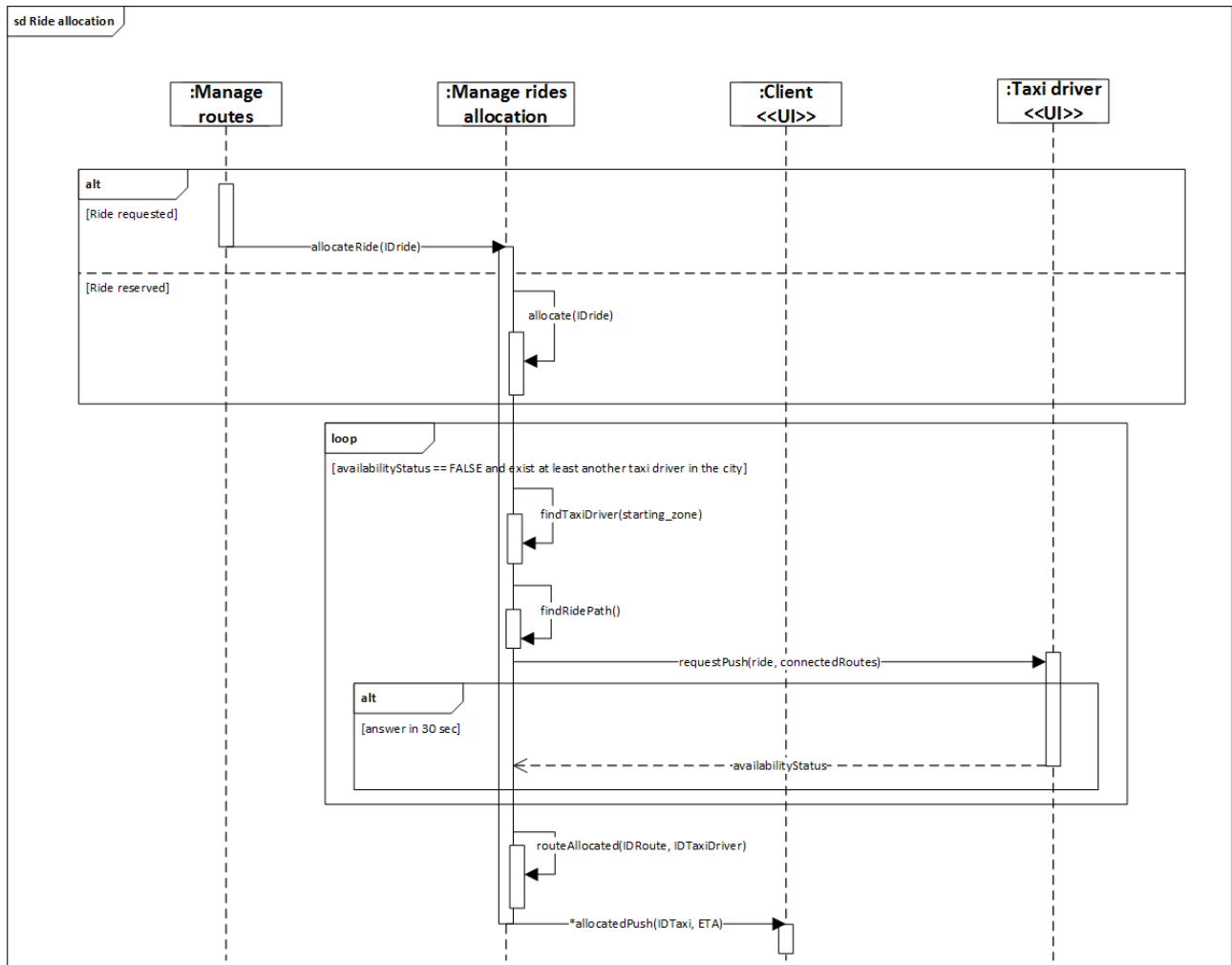
## 2.5.4 Create route

This sequence diagram describes the process of creation of a ride by a Client.



### 2.5.5 Ride allocation

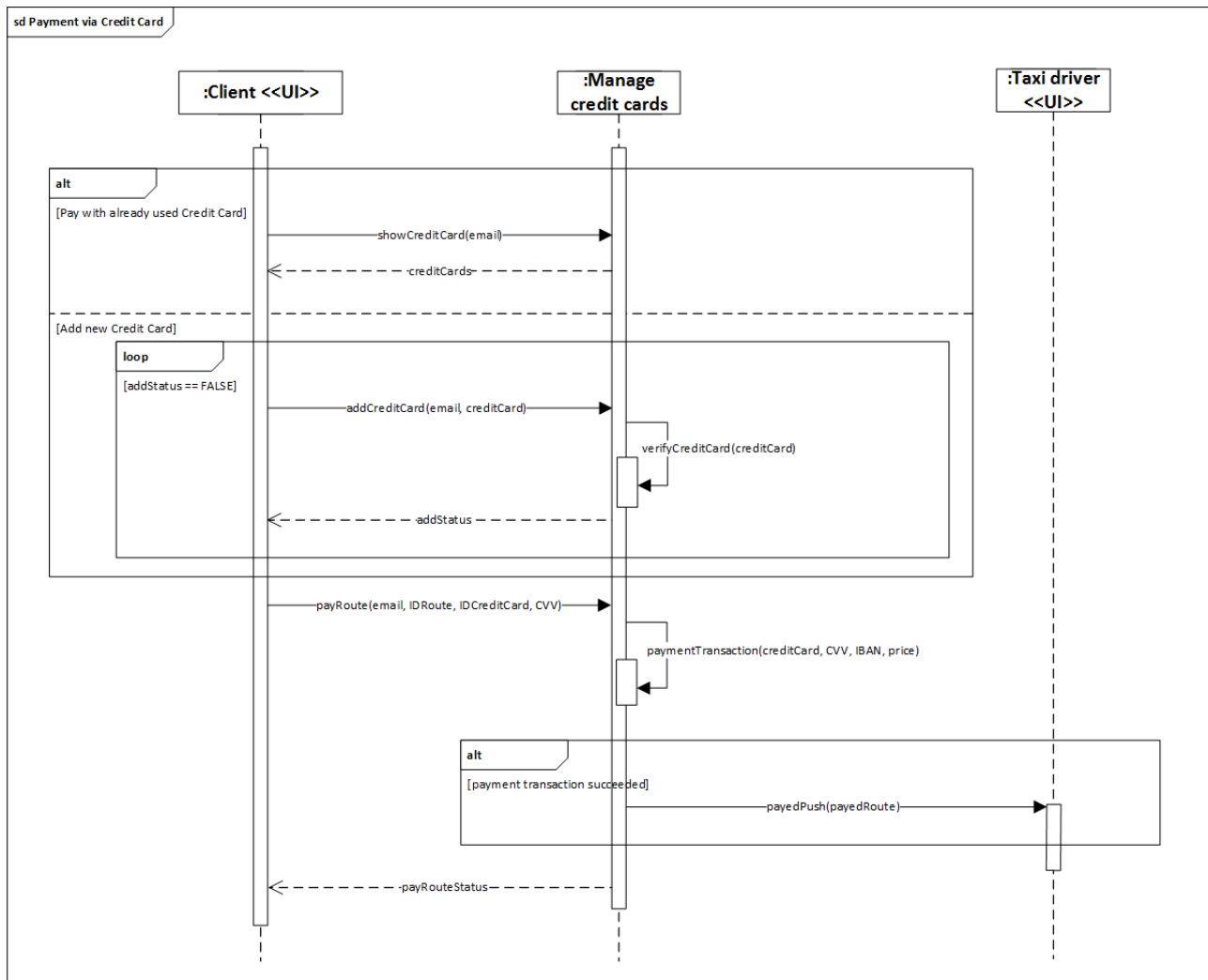
This sequence diagram describes the process of allocation of a ride to a specific Taxi driver.





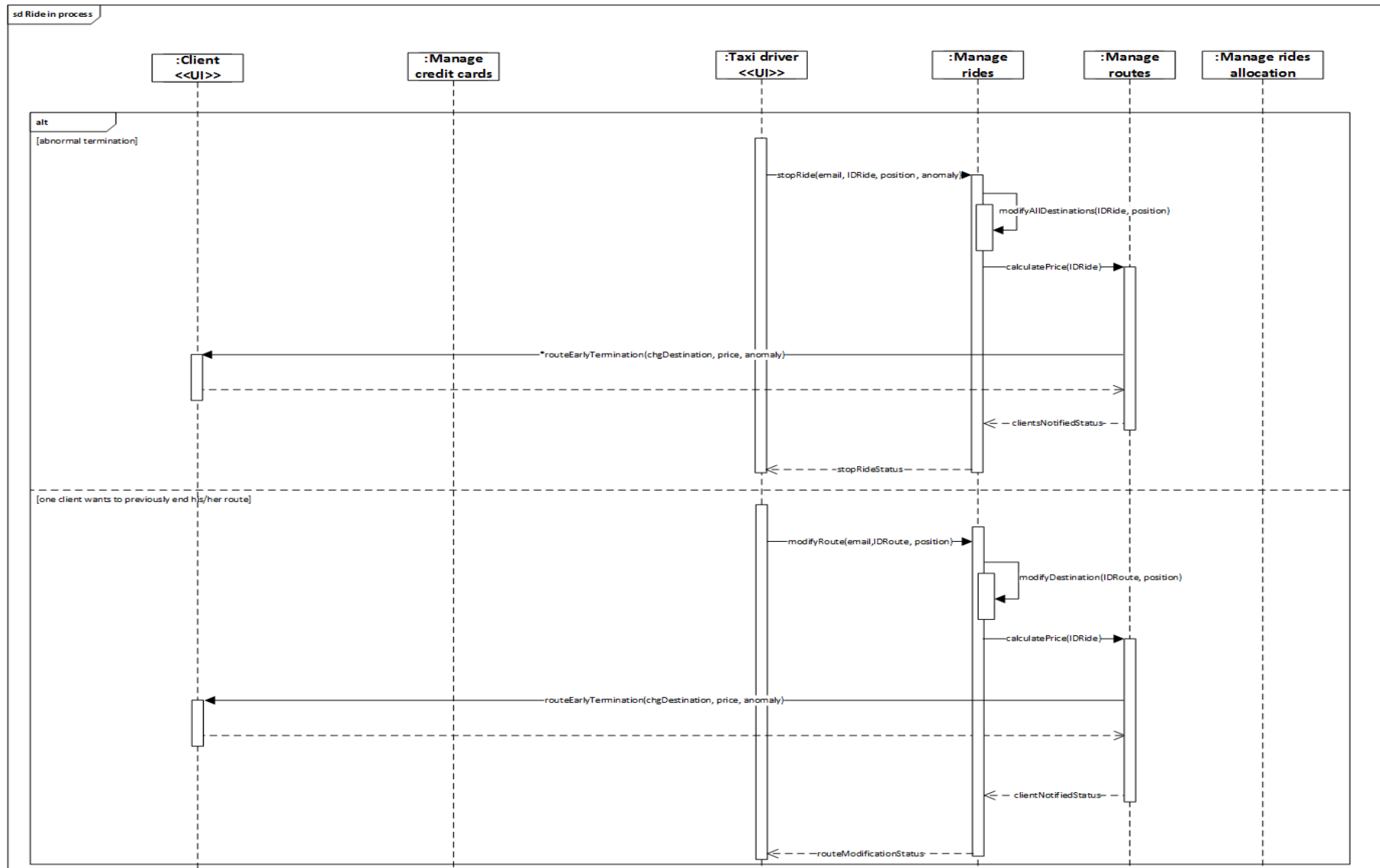
## 2.5.6 Payment via Credit Card

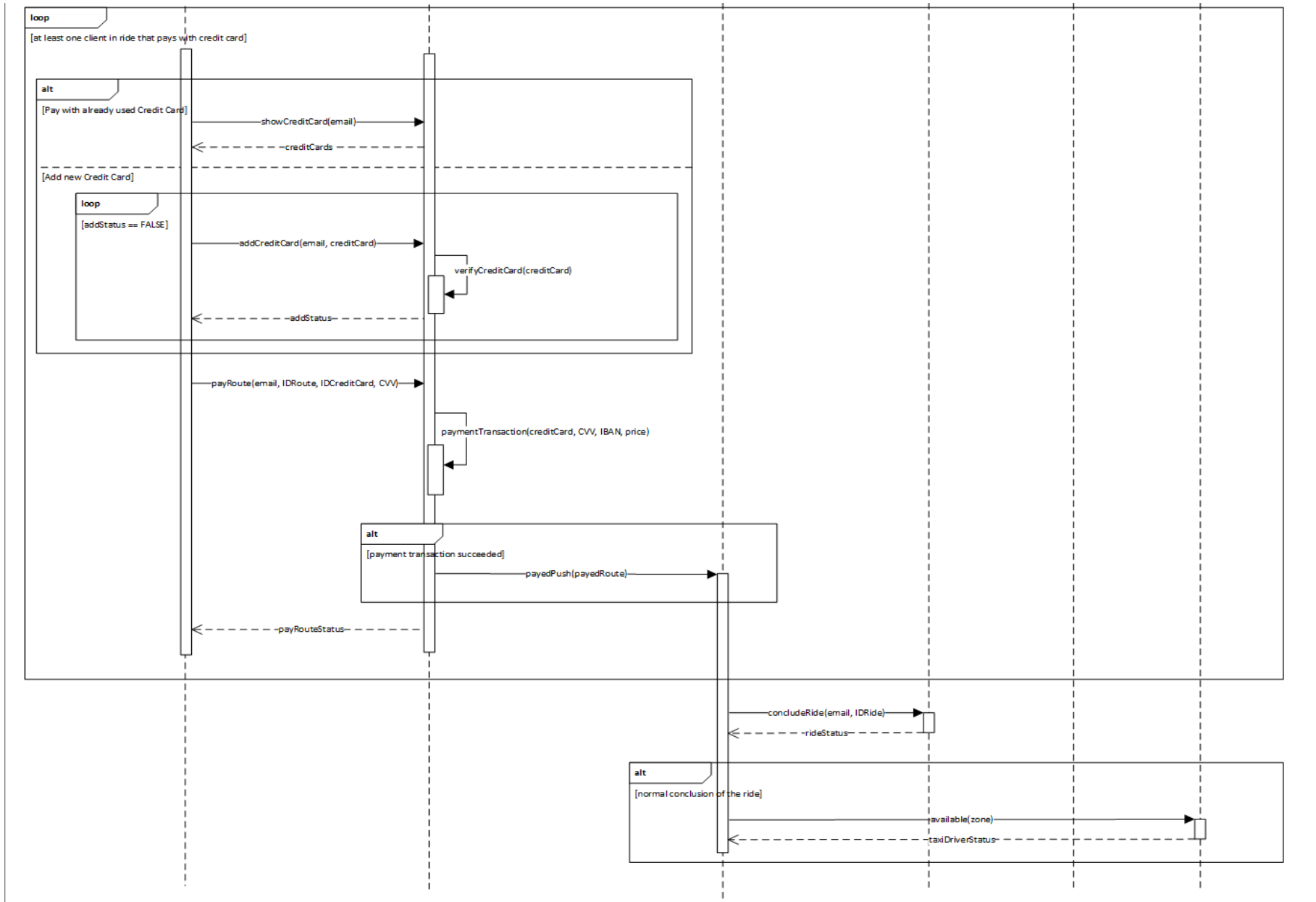
This sequence diagram describes the payment process via Credit Card by a Client.



## 2.5.7 Ride in process

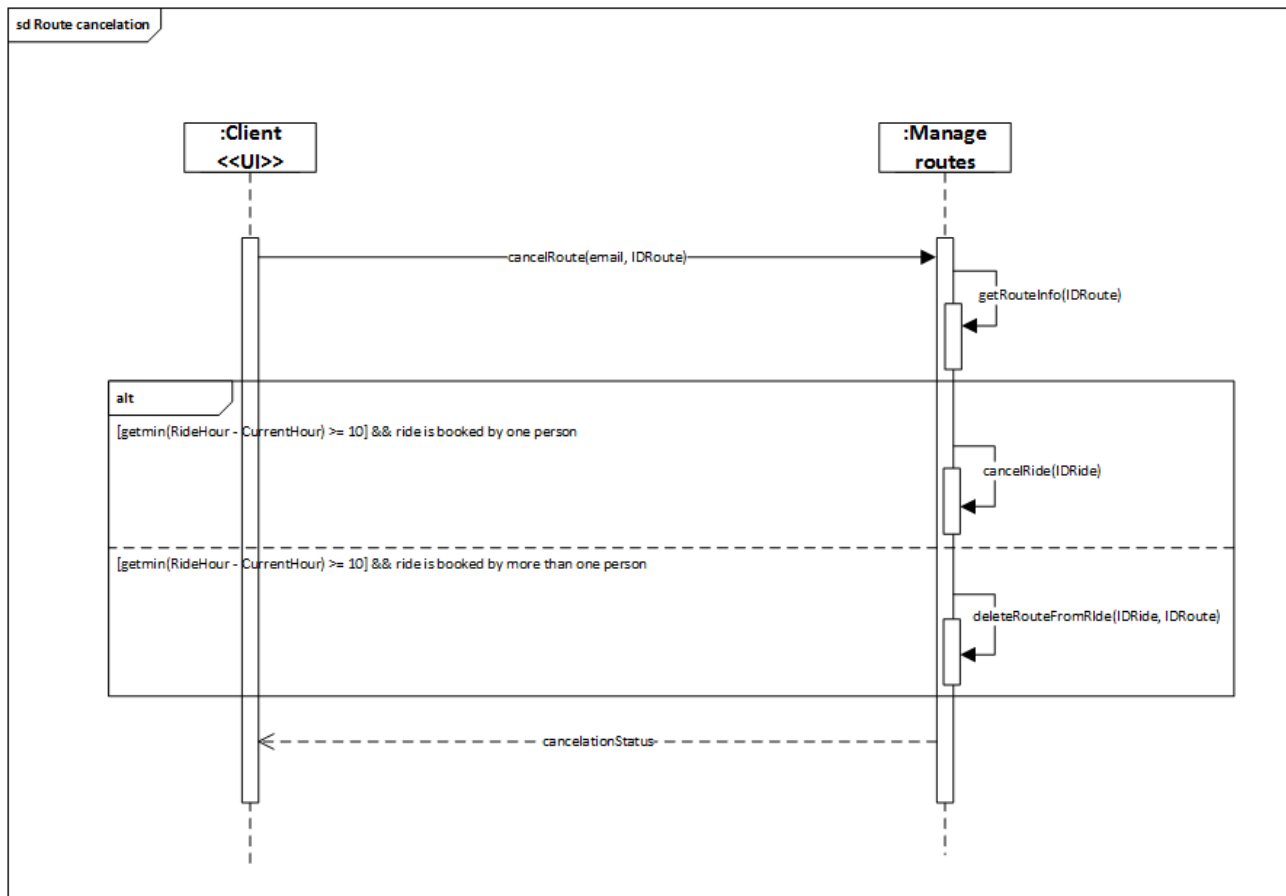
This sequence diagram describes the steps from when a ride is underway to when it ends.





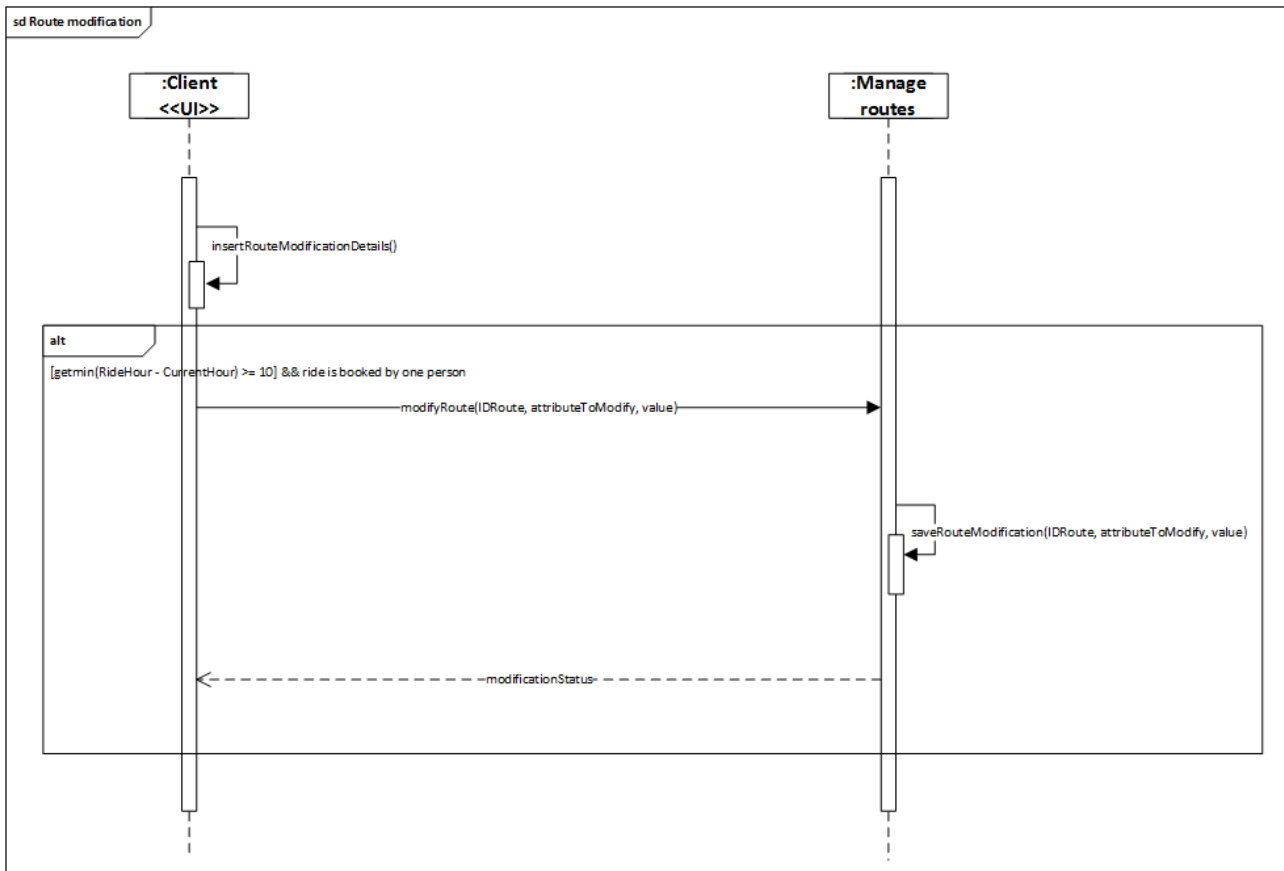
## 2.5.8 Route cancellation

This sequence diagram describes the process of cancelation of a reserved route by a Client.



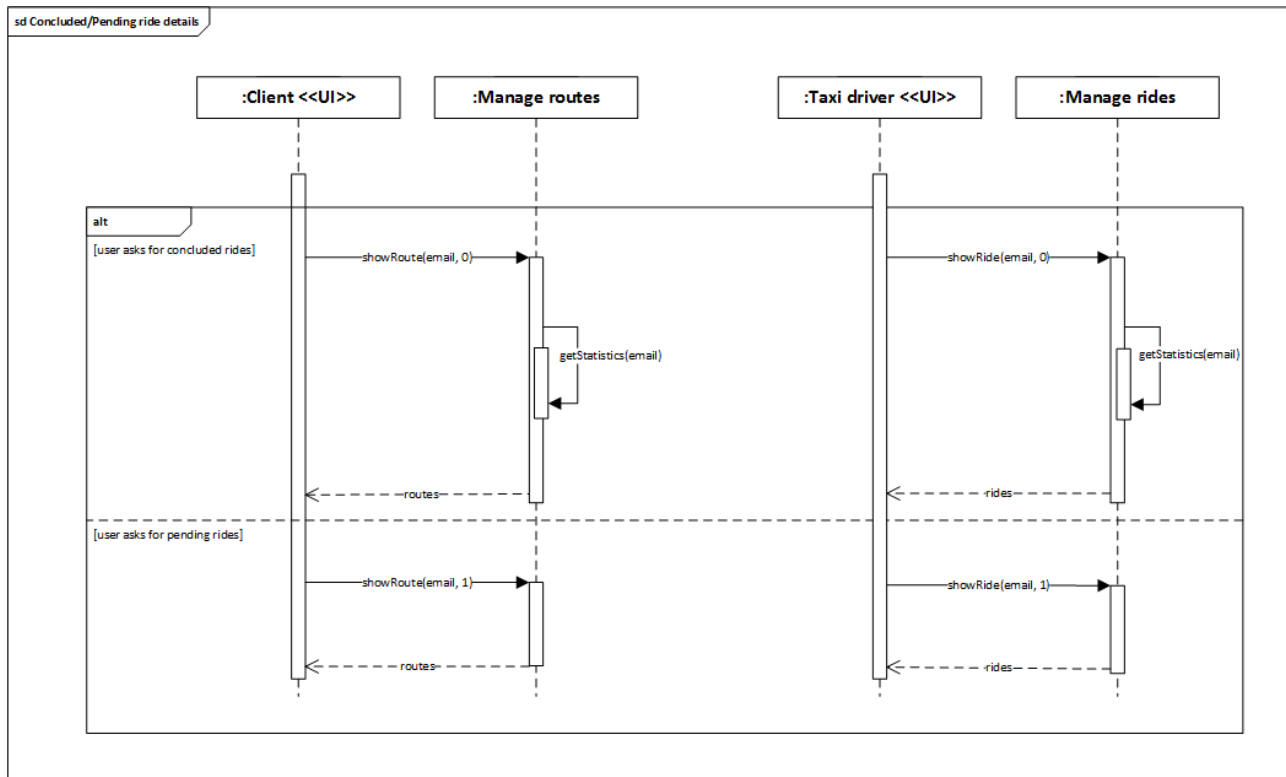
### 2.5.9 Route modification

This sequence diagram describes the process of modification of a reserved route by a Client.



## 2.5.10 Concluded/Pending ride details

This sequence diagram describes the steps to see the list of concluded and pending rides of a Taxi driver and the list of concluded and pending routes of a Client.



## 2.6 Component interfaces

In the following section we will describe each interface of each software component. We will also indicate the type and the name of the parameters they will use.

### 2.6.1 Client <<UI>>

```
void allocatedPush(string IDTaxi, int ETA)
void routeEarlyTermination(string chgDestination, float price, string anomaly)
```

### 2.6.2 Taxi driver <<UI>>

```
bool requestPush(Ride ride, Route connectedRoutes)
void payedPush(Route payedRoute)
```

### 2.6.3 Manage users accounts

```
bool confirm(string confirmationCode)
bool login(string email, string password)
bool logout()
bool passwordRecovery(string email)
bool registration(Client client)
bool modifyClient(string email, string attributeToModify, Object value)
bool registration(TaxiDriver taxiDriver)
bool modifyTaxiDriver(string email, string attributeToModify, Object value)
bool sendEmail(string to, string subject, string message)
bool validateDriverLicense(DriverLicense driverLicense)
bool validateTaxiLicense(TaxiLicense taxiLicense)
```

### 2.6.4 Manage credit cards

```
bool payRoute(string email, string IDRoute, string IDCreditCard, string CVV)
bool addCreditCard(string email, CreditCard creditCard)
bool modifyCreditCard(string email, string IDCreditCard, string attributeToModify, Object value)
bool deleteCreditCard(string email, string IDCreditCard)
CreditCards[] showCreditCard(string email)
bool validateCreditCard(CreditCard creditCard)
```

### 2.6.5 Manage taxis

```
bool addTaxi(string email, Taxi taxi)
bool modifyTaxi(string email, string IDTaxi, string attributeToModify, Object value)
bool deleteTaxi(string email, string IDTaxi)
Taxi[] showTaxi(string email)
bool validatePlate(string plate)
```

### 2.6.6 Manage routes

```
Route[] showRoute(string email)
Route[] showRoute(string email, bool pending)
bool joinRide(string email, string IDRide, Route route)
bool addRoute(string email, Route route)
bool modifyRoute(string email, string IDRoute, string attributeToModify, Object value)
bool cancelRoute(string email, string IDRoute)
```

```
int zoneIdentification(string origin)
Coordinates getCoordinates(string origin)
Ride[] findCompatibleRides(string origin, string destination, Time startTime, Time endTime)
double calculatePrice(Ride ride)
void getStatistics(string email)
```

### **2.6.7 Manage rides allocation**

```
bool allocateRide(string IDride)
bool changeWorkState(string email, int workState)
TaxiDriver findTaxiDriver(Zone zone)
Time getTimeOfArrival(string IDroute)
bool available(nZone)
```

### **2.6.8 Manage rides**

```
Ride[] showRide(string email)
Ride[] showRide(string email, bool pending)
bool stopRide(string email, string IDride, string position, string anomaly)
bool modifyRoute(string email, string IDroute, string destination)
bool concludeRide(string email, string IDride)
void getStatistics(string email)
```

### **2.6.9 Admin <<UI>>**

```
Ride[] showRide(string email)
void getStatistics(string email)
```



## 2.7 Selected architectural styles and patterns

In the design of the application *myTaxiService* we have used different styles:

1. Client-server
2. Components and connectors
3. Cloud Computing
4. Distributed objects

Below, we describe the chosen styles, focusing on the reasons for our choices and the modalities in which we have exploited them.

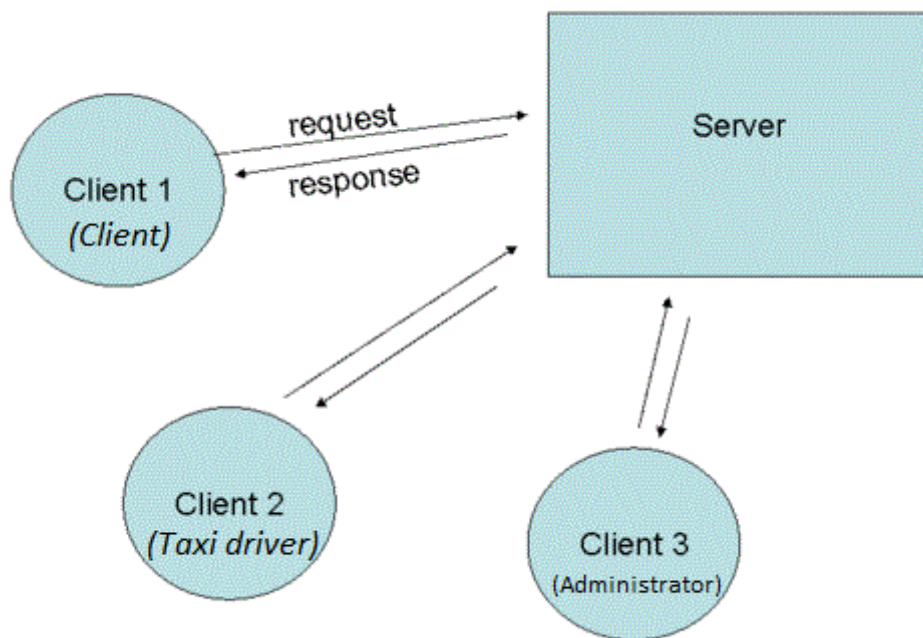
### 2.7.1 Client-Server

The main style of the application is Client-Server.

There are three types of clients:

- Client
- Taxi driver
- Administrator

The server is the central system that contains the web pages, business logic and data. We have decided to store them in the Cloud service offered by Amazon, Amazon Web Services (AWS).

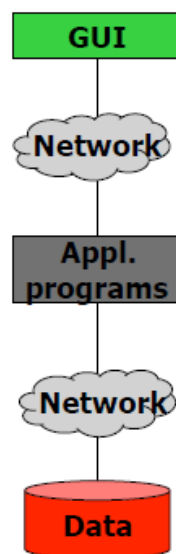


The choice of Client-Server style was quite obvious because it is intrinsic to the kind of application we are going to develop: many people use it at the same time, either from the website or the mobile app, and a central system has to manage their requests.

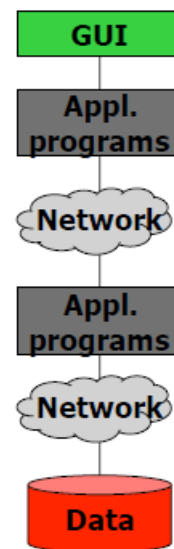
The architecture chosen is the four-tier architecture (the server tier is divided into two sub-tiers: business and web): in fact, this allows the separation of data, logic and presentation. We have applied two different variations of the schema depending on the application used. In both of them, the main business logic is located in the mid-level. The first one describes the schema when it is used the website, while the second one describes the schema when it is used the mobile application.

The two different cases, as you can also see in the pictures below, are composed by (from top to bottom):

1. GUI → Network → Application Programs → Network → Data
  - a. **GUI.** The browser used to visualize the web pages downloaded from the server.
  - b. **Network.** The network used from the user to communicate with the application programs, stored on the server on the Cloud.
  - c. **Application programs.** It contains the business logic (business tier) and the interfaces of the website (web tier) and it is stored on the server on the Cloud. In this case the user will access to both tiers.
  - d. **Network.** The network used to communicate between the application programs and the database stored on the Cloud.
  - e. **Data.** The data stored on the database on the Cloud.
2. GUI → Application Programs → Network → Application Programs → Network → Data
  - a. **GUI.** The mobile application itself.
  - b. **Application programs.** It contains the little part of the business logic of the mobile application.
  - c. **Network.** The network used from the user to communicate with the application programs, stored on the server on the Cloud.
  - d. **Application programs.** It contains the business logic (business tier) and the interfaces of the website (web tier) and it is stored on the server on the Cloud. In this case the user will access only to the business tier.
  - e. **Network.** The network used to communicate between the application programs and the database stored on the Cloud.
  - f. **Data.** The data stored on the database on the Cloud.



1. Four-tier architecture - Website  
The application programs in the image is internally divided into two tiers: Business tier and Web tier. In this case the user will access to both tiers.



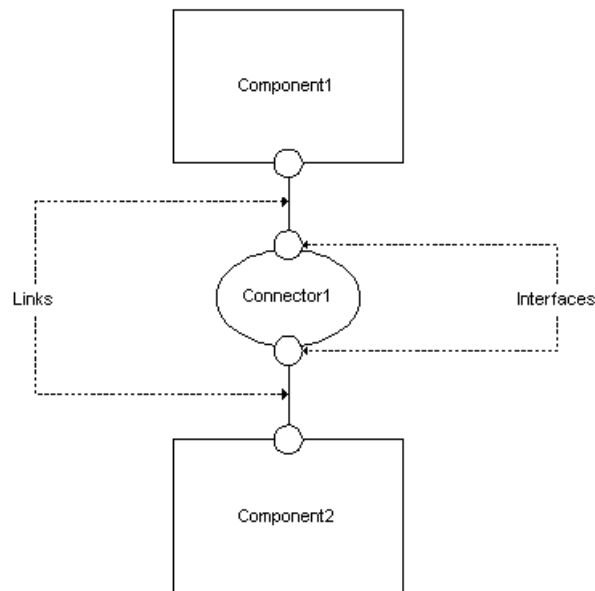
2. Four-tier architecture - Mobile app  
The application programs between the two networks in the image is internally divided into two tiers: Business tier and Web tier. In this case the user will access only to the business tier.

Another choice is to design servers as *fat* servers: in fact, they contain all the business logic. Clients can be considered *thin*: in fact, only the mobile app contains a portion of business logic, but it is not enough to consider it fat.

### 2.7.2 Components and connectors

The choice of this style is due to the organization in components of the server part of our application. It's structured in a lot of components, and each of them communicate with one or more of the others using specific interfaces.

This choice allows the programmers to separate their work per component and to save time and money for the development of the application. Moreover, it allows the realization of a scalable and extensible system: in fact, modern systems are continuously modified to fix bugs, to improve performances, to offer new services or to adapt to new laws. Dividing the software into components will make it easier to modify a single part or add new parts without having to revise the entire product.



### 2.7.3 Cloud Computing

We have chosen to use a Cloud service, in particular Amazon Web Services (AWS), to store the business logic, the interfaces of the website and data.

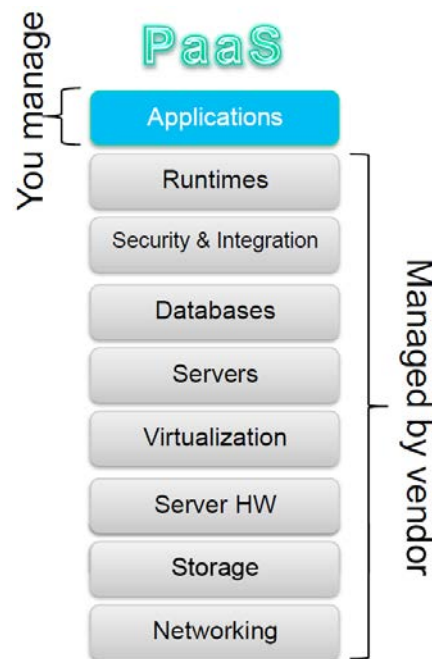
Cloud computing offers many advantages, most of which are linked to costs: in fact, they vary depending on the usage of resources and are smaller thanks to economies of scale. In addition to that, AWS adopts a pay-as-you-go system: therefore, there are no additional costs for the management and the purchase of data centers and servers.

Another advantage is the scalability of the resources, meaning that you do not need to choose the storage capacity or bandwidth every time they vary because they are dynamically modified. The cloud offers a service that is both secure and reliable: the former is granted by functions like the management of access to data, while the latter is granted by functions like *Backup and Restore* and replicated servers that may be located in different parts of the world in order to offer global availability with low latency.

Finally, it does not need the frequent updates and maintenance work that are necessary for traditional solutions, making the system faster, more agile and more stable.

The model of Cloud service chosen is **Platform as a Service (PaaS)**.

This choice was made to focus only on the development of the business logic of the application. In fact, the configuration and the organization of physical servers is managed by AWS (that avoids the problem of their sizing), as well as security and data integration, saving us time and money. It helps the achievement of non-functional requirements (e.g. reliability and efficiency of the system) in an optimal way and at the minimum cost on the market. It also permits the dynamic management of the network-band with a dynamic allocation of resources that permits, for instance, to allocate less bandwidth during the night when the taxi service is less used and more bandwidth during the rush hours.

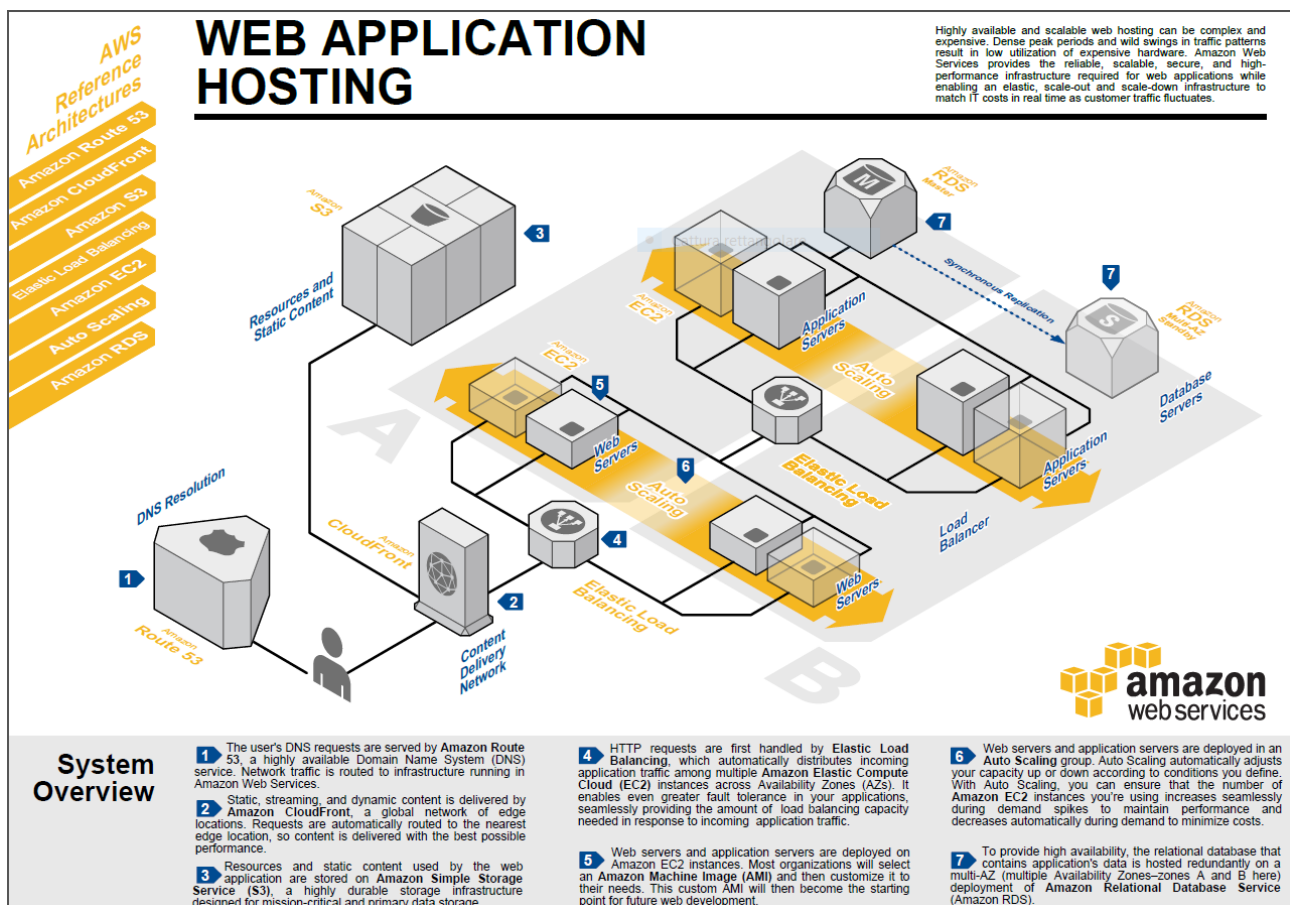


The services and products we use from AWS are the following:

- **Calculation**
  - *Amazon Elastic Compute Cloud (Amazon EC2)*. It provides resizable processing capabilities in the cloud, w.r.t. our necessities. Amazon increases or decreases rapidly the capabilities depending on the necessities of the process.

- *AWS Elastic Beanstalk*. It is used to distribute and resize application and services that we develop. By uploading the Java code, this component automatically manages the implementation. It doesn't have additional costs.
- **Database**
  - *Amazon Relational Database (Amazon RDS)*. It allows to configure, use and resize the relational database used in the cloud. It permits to have a resizable capacity at the lowest cost. The DBMS that we have chosen to use is PostgreSQL.
- **Networks**
  - *Elastic Load Balancing*. It automatically distributes the incoming applications traffic on multiple instances of Amazon EC2.
- **Management tools**
  - *Amazon CloudWatch*. It's a service for the monitoring of the cloud AWS resources and the applications that are executing on AWS. It is also used to keep track of parameters generated by the application and to gather and monitor log files.
- **Application services**
  - *Amazon Simple Email Service (Amazon SES)*. It's the email service. It allows to send transactional email, marketing messages or any other high quality contents. It follows a pay-as-you-go system, so we will only have to pay for the resources that we have used.
  - *Amazon Simple Notification Service (SNS)*. It is used to send push messages.

The following image shows the web application hosting of AWS:



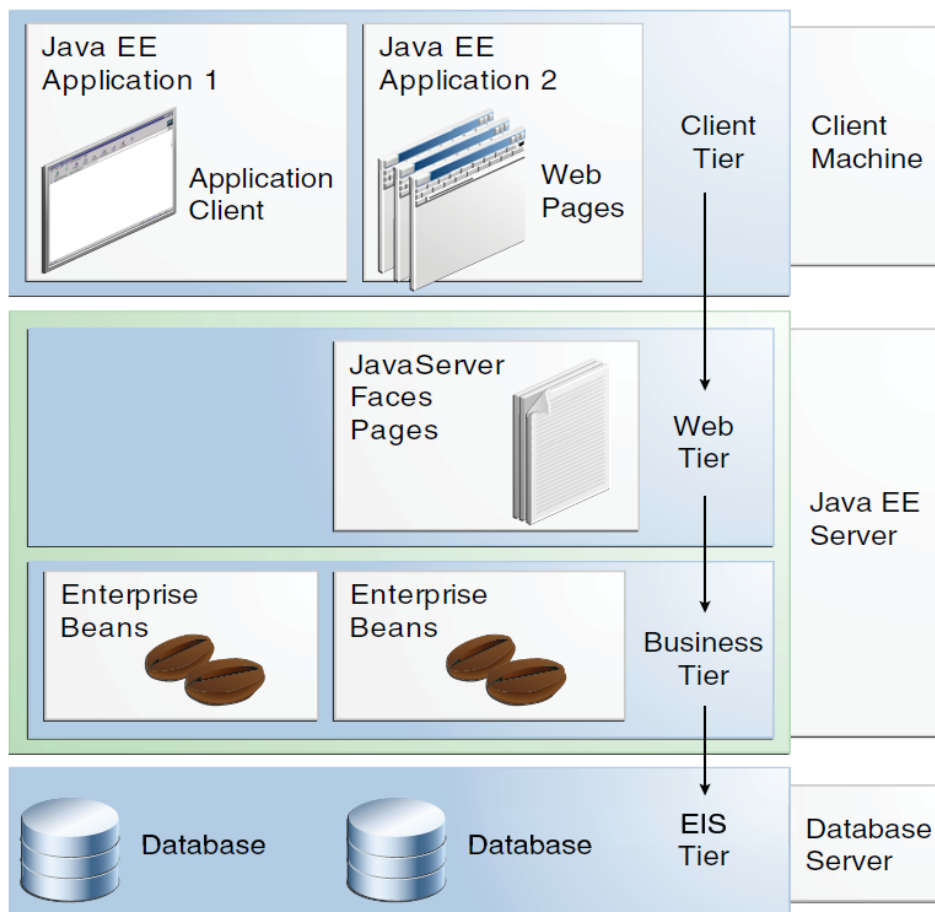
### 2.7.4 Distributed objects

This style has been chosen because of the choice of Java EE for the development of our application. This technology forces the implementation of the application for the programmers in an OO way. We think it is a good idea because Java is a consolidated standard, used worldwide and it is a programming language known by the majority of the programmers. Because of the high competition among them, it facilitates the task of finding good and suitable programmers for the development of our application in less time and saving more money.

Moreover, this choice allows to develop the whole business logic with a uniform language (Java EE) and with a proven structure for the development of an entire application from scratch.

Java EE is composed by a language, a platform (JRE) and programming tools (JDK).

It enables developers to focus on business logic and its architecture is shown in the image below.



In particular, the system we have developed is composed by the following components:

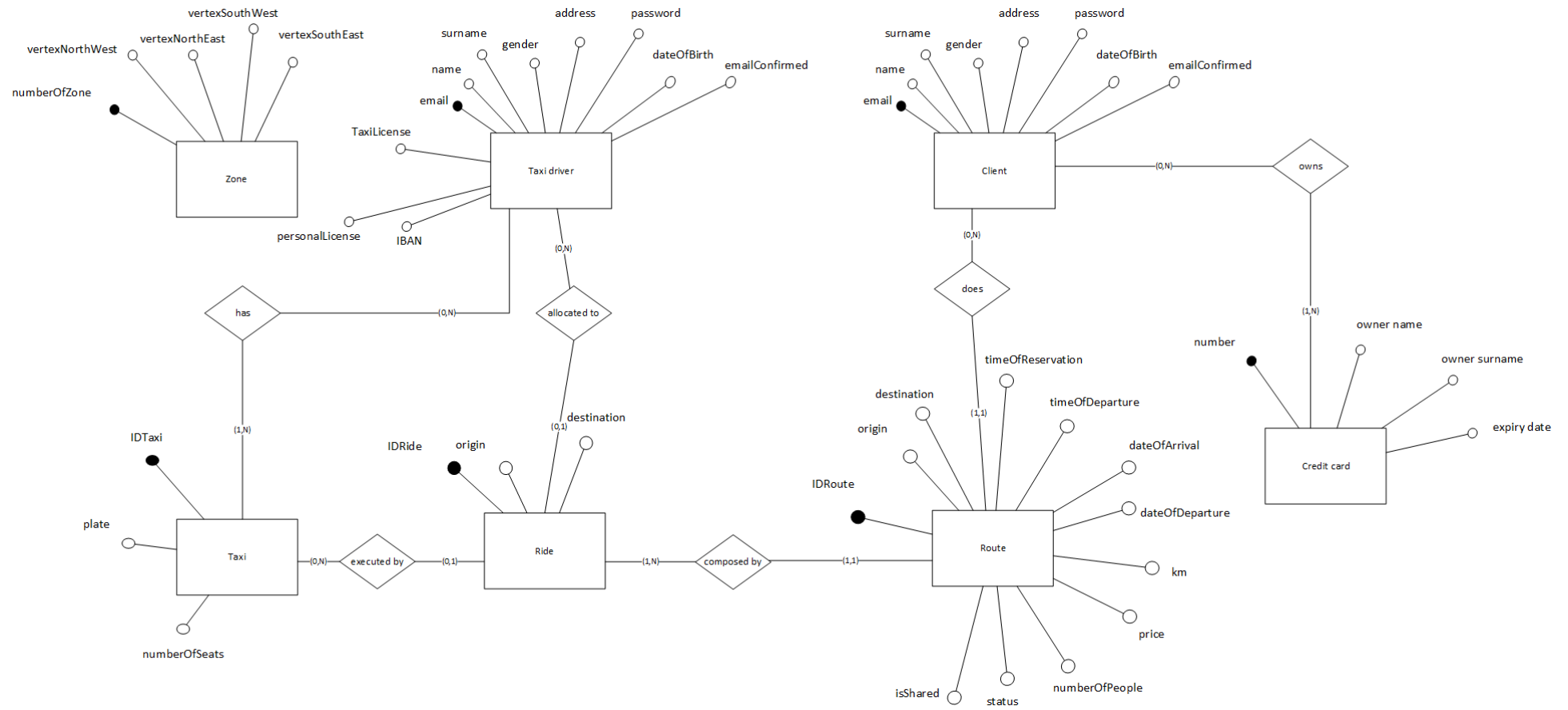
- Client components.
  - *Application clients*
- Server components.
  - *Web components*: JavaServer Faces (JSF) components, Facelets
  - *Business components*: Enterprise Java Beans (EJB), Java Persistence API (JPA) entities
  - *“glue components”*: Context and Dependency Injection (CDI) beans, JAX-WS
  - *Database*

In particular, we have decided to divide the Web tier from the Business tier according to the suggested way to host a web server in AWS since they may experience different loads.

The main beans we use for the EJB are:

- Taxi driver: JPA entity
- Client: JPA entity
- Route: JPA entity
- Ride: JPA entity
- Zone: Session bean, @Stateless
- Credit card: JPA entity
- Taxi: JPA entity

The ER diagram for the database is the following:





## 2.8 Other design decisions

The decisions taken that we have not already described in the other parts of the document are the choices of the technologies used and of the external interfaces used for the communication with other systems.

### 2.8.1 Technologies

As you can see in the paragraph 2.4 “Deployment view” we have made the following technological choices.

- The DBMS we have chosen is PostgreSQL 9.4. We made this choice because it is an open license DBMS with optimal performances and it is supported by AWS.
- The Application server we have chosen is Apache Tomcat 8. It is open source and includes a software platform for the execution of Web application developed in Java. In fact, we have chosen to use JSF and Facelets for the development of our Web components. This Application server has been chosen also because it is supported by the cloud service AWS.
- The Web server we have chosen is Apache HTTP Server 2.4. It offers control functions for the security as those realized by a proxy. This Web server has been chosen also because it is supported by the cloud service AWS.
- The main technology used to develop our application is Java EE as explained in the chapter 2.7 “Selected architectural styles and patterns”.
- The cloud service we have chosen is Amazon Web Services (AWS).

### 2.8.2 External interfaces

The communications with the already existent systems/services are:

1. Google Maps
2. A public authority (e.g. *Ministero delle infrastrutture e dei trasporti*)
3. Public authorities that manage taxi licenses and their owners
4. Credit cards' companies

In the following subparagraphs we describe how we use these systems/services.

#### 2.8.2.1 Google Maps

Google Maps is exploited by using its APIs to calculate the routes of the clients and to compute the compatibilities among different routes of different clients as explained in Chapter 3.

We have thought to use this service offered by Google because it is used worldwide, it is reliable, very efficient and has open maps. It allows us to save money because we do not have to develop such a critical component of the application.

The Google Maps Web Services used in our application are:

- **Google Maps Directions API.** Returns multi-part directions for a series of waypoints that correspond to the origin and destinations of the rides in shared rides.
- **Google Maps Distance Matrix API.** Retrieves duration and distance values based on the recommended route between start and end points. It is used to obtain the length of a single route.

- **Google Maps Geocoding API.** Used for converting addresses (like "1600 Amphitheatre Parkway, Mountain View, CA") into geographic coordinates (like latitude 37.423021 and longitude -122.083739), which we use to place markers on the map, or position the map.
- **Google Maps Geolocation API.** Returns a location and accuracy radius based on information about cell towers and Wi-Fi nodes that the mobile client can detect.

#### ***2.8.2.2 A public authority (e.g. Ministero delle infrastrutture e dei trasporti)***

The communication with this public authority is used to check if the plate inserted by the taxi drivers in the system is correct.

#### ***2.8.2.3 Public authorities that manage taxi licenses and their owners***

The communication with these entities is necessary to check the correctness of the licenses inserted by the taxi drivers at the moment of their registration to the application *myTaxiService*.

#### ***2.8.2.4 Credit cards' companies***

The communication with credit cards' companies is needed for the payment from clients to taxi drivers at the end of the rides and to check the correctness of credit cards' data when clients insert new credit cards in the system.

### 3 ALGORITHM DESIGN

#### Management of the shared rides

Since we manage shared rides in a “social” way (different clients can join a ride created by another client), the main problem with the shared rides is the choice of compatible itineraries among the routes created by different clients in order to show them to the client that wants to take part to a shared ride.

Then, assuming that at least one shared ride already exists, we describe below the core concept of the algorithm that the system executes in order to find rides that are compatible with the new one added by a client.

The main idea is based on the fact that the itineraries of our clients are calculated from Google Maps. Clearly, to do that, we use Google APIs for the communication with Google.

#### Steps of the algorithm for the research of compatible rides

For each ride in the system (after a client has inserted an origin, a destination, a departure time window and has clicked on the button to search compatible rides with it), in order to be able to choose whether to join or not to one of them:

1. The system queries Google Maps with origin and destination of the routes that compose the ride (only which that has as origin the same origin zone of it and a departure time an hour within the range of time that is chosen by the client in the form for the choice of a shared ride) and obtains the length in km of each route.
2. The lengths of the routes received are ordered in an increasing way.
3. The system queries Google Maps with origin and destination of the longest route composing the ride and, with intermediate points, the destinations of the routes of the other clients ordered (point 2).
4. The length received ( $L_{tot}$ ) is compared to the length of the longest route ( $L_{longestRoute}$ ) received obtained from those calculated at point 2.

if (  $\text{abs}(L_{tot} - L_{longestRoute}) < \text{threshold}$  ) then

    Add the client to the ride

else

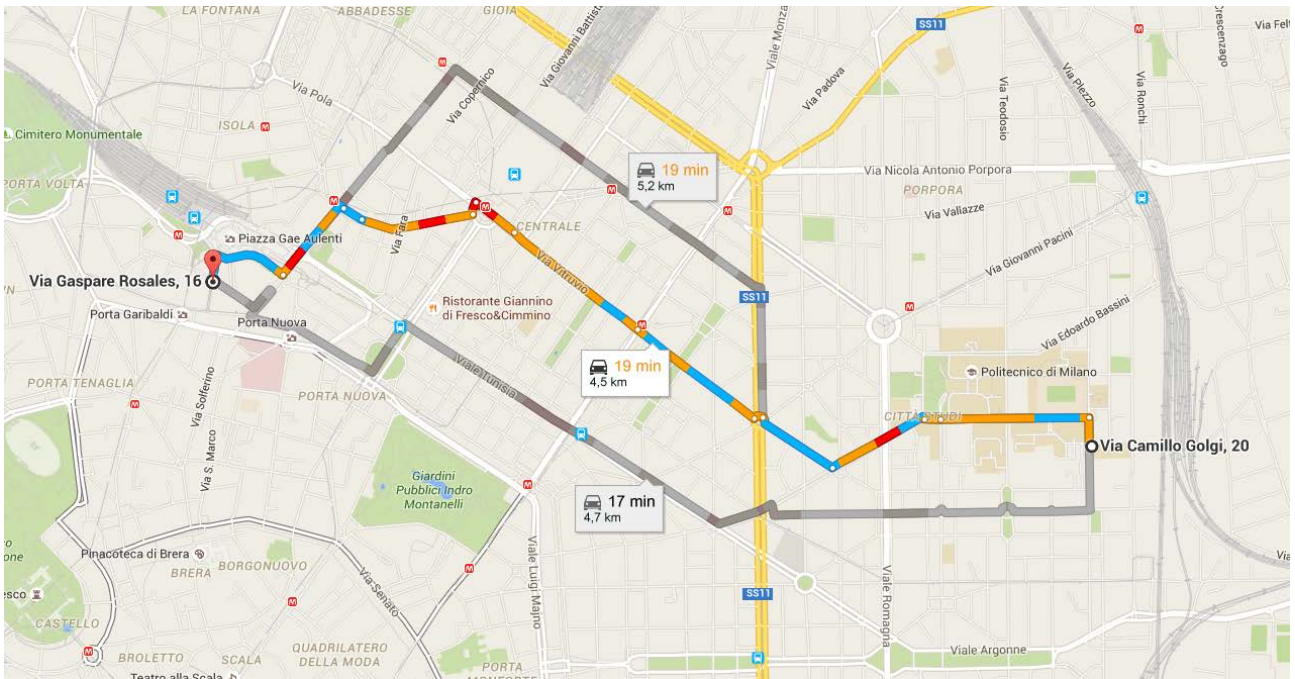
    Refuse the join

end if

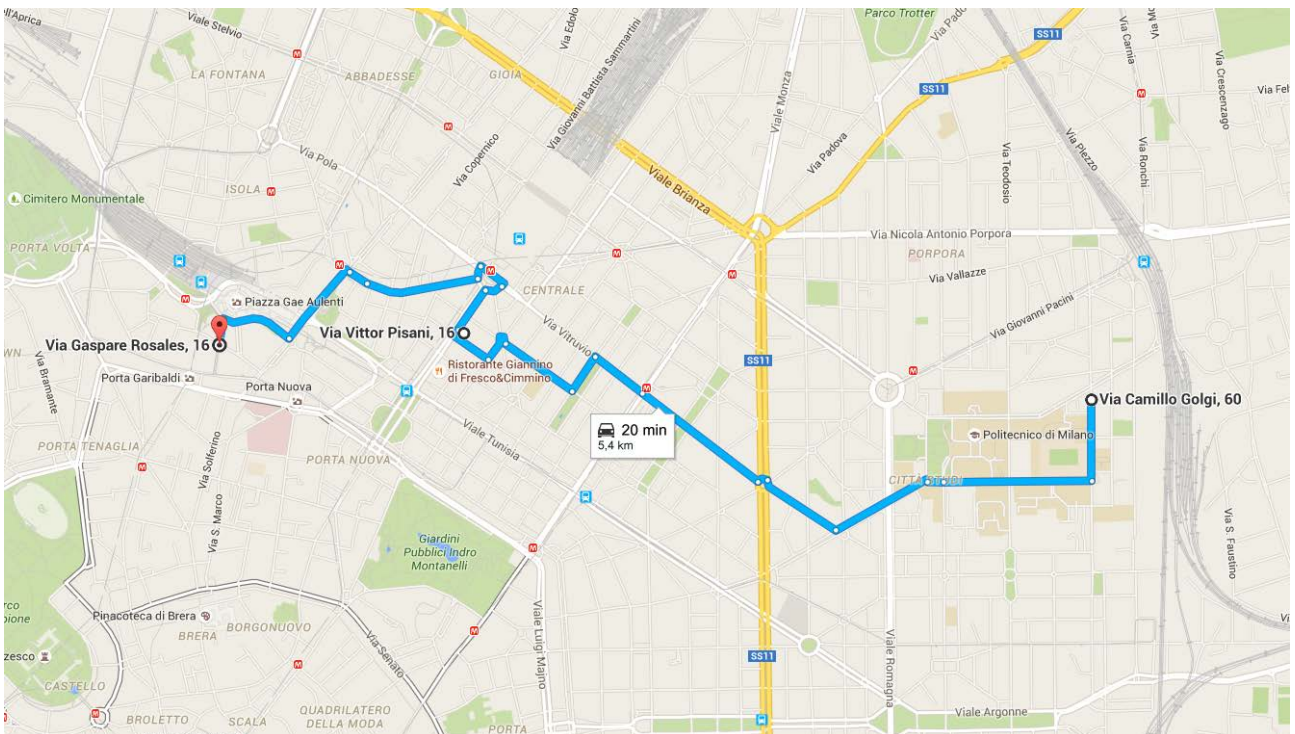
The thresholds are calculated proportionally to the number of different routes in each ride (considering the current route already joined), in this way:

- 2 routes:  
    threshold = 5% of the longest route (e.g. longest route = 50km, threshold = 2.5km)
- 3 routes:  
    threshold = 7% of the longest route (e.g. longest route = 50km, threshold = 3.5km)
- 4 routes:  
    threshold = 9% of the longest route (e.g. longest route = 50km, threshold = 4.5km)

For clearness we provide an example. This is the faster itinerary calculated by Google Maps for a ride with a single route from *Via Camillo Golgi 20, Milano* to *Via Gaspare Rosales 16, Milano*. The route has a length of 4.5 km.



When a new client starting from the same zone but from *Via Camillo Golgi 60, Milano* wants to go to *Via Vittor Pisani 16, Milano*, the system, executing the steps of the algorithm written above, find that this route is not compatible with the first route. In fact, with the addition of the new destination in the set of the routes within the ride considered (in this case was only one, now we are evaluating the addition of a second one), the total length of the ride would become 5.4 km.



As explained in the algorithm, the threshold in this case would be:

$$\text{threshold} = 5\% \text{ of } 4.5\text{km} = 225\text{m}.$$

Hence, because in this case  $L_{\text{tot}}$  (5.4 km) –  $L_{\text{longestRoute}}$  (4.5 km) = 900 m, the addition to the ride is refused. Therefore, the system will not show this ride to the client as it is not compatible with theirs.

## Choice of the correct taxi driver from the queues

When the system has to allocate a ride to a taxi driver, an algorithm is executed by the system to choose among them in the appropriate way, ensuring the fair management of the queues.

Steps:

1. The system calculates the origin zone of the ride.
2. The system considers the data structure that represents the queue corresponding to that zone.
3. The system looks for the first element (they represent taxi drivers) in the data structure considered at point 2, obviously managed with a LIFO policy (queue).

*Pseudocode*

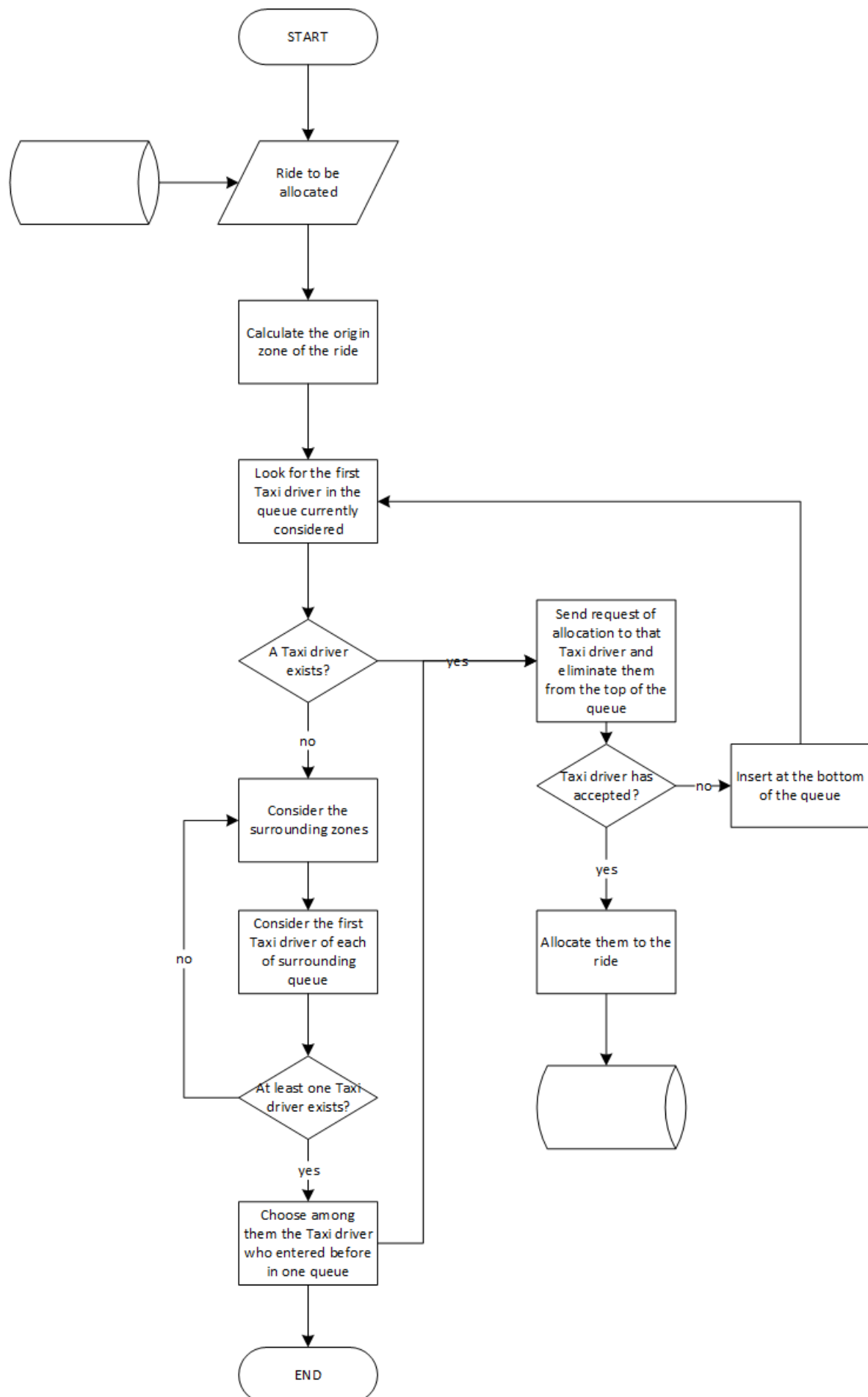
```
While ( ride not allocated )
  if ( firstTaxiDriverOfQueue(CurrentQueueConsidered) != null ) then
    Send the request of allocation of the ride to that taxi driver and eliminate them from the queue.

    While( answer not already received && timer < 30s )
      // waiting
    End while

    if ( taxi driver refuses ) then
      Insert them at the end of that queue
    else
      Allocate the ride to that Taxi driver
    end if
  else
    do
      Considering the data structure of the queues surrounding the current ones/one
      starting from the central zone and expanding to the others (as an onion).

      j=0;
      For (i=0; i<count (surrounding queues); i++)
        If (firstElementOfQueue (Queue[i]) != null ) then
          HourOfEntranceFirstTaxiDriverInTheQueue[j] =
            getHourOfEntrance (firstElementIn (Queue[i]))
          j++;
        End if
      End for

      If (j>0)
        Take the taxi driver, among those in the considered queues, that has the
        same hour of entrance in their queue:
        min ( HourOfEntranceFirstTaxiDriverInTheQueue )
        In the case two or more hours of entrance are the same choose taxi driver
        randomly among them.
      End if
      While (j==0 && other surrounding queues exist)
    End if
  End while
```

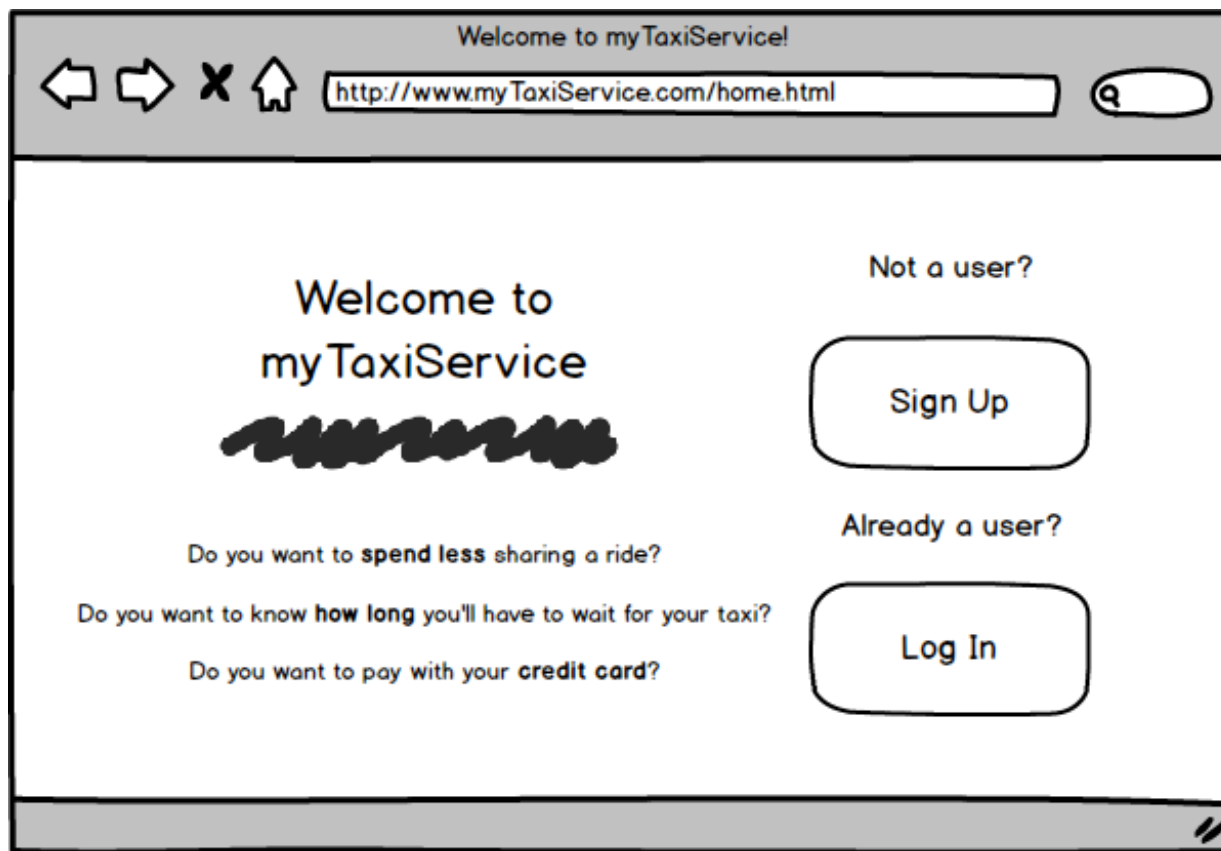


## 4 USER INTERFACE DESIGN

In this section we will provide an overview of the system's user interfaces. Some of them were already present in the paragraph 3.1 "External interfaces" of the RASD; however, we have decided to add more detailed mock-ups for user interfaces that may be different when the user is a Client or a Taxi driver or that require a more detailed explanation (e.g. the shared rides interface or the payment interface).

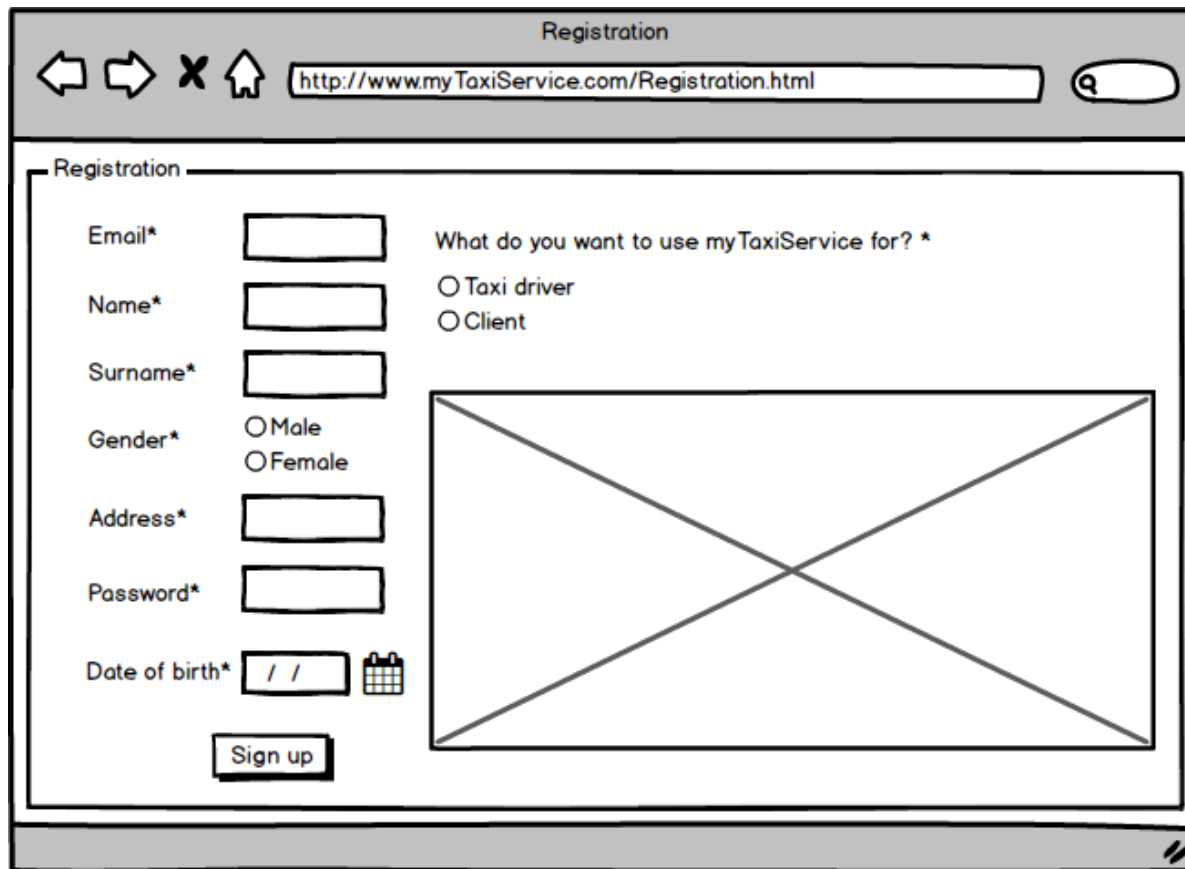
### 4.1 Home page

The following images show the home pages available for a Guest from the web site or from the mobile application.



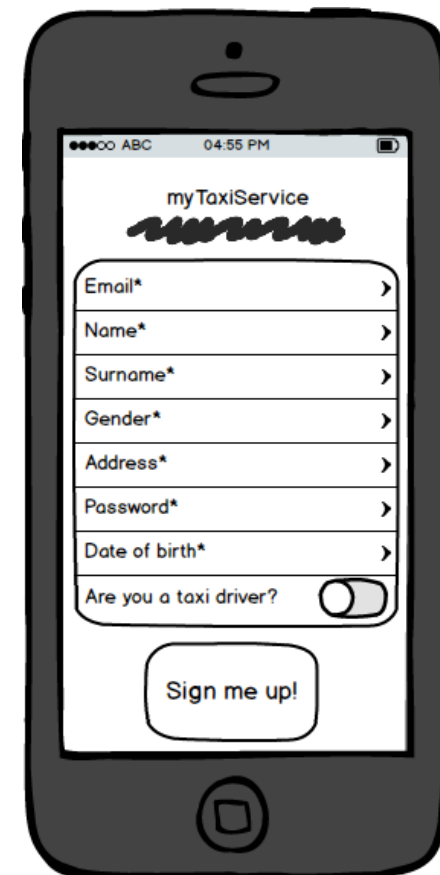
## 4.2 Registration page

The following images show the registration pages available for a Guest from the web site or from the mobile application; if they decide to register as taxi driver, they must insert additional information.



The image shows a web browser window titled "Registration" with the URL <http://www.myTaxiService.com/Registration.html>. The registration form includes the following fields and options:

- Email\* (text input)
- Name\* (text input)
- Surname\* (text input)
- Gender\* (radio buttons for Male and Female)
- Address\* (text input)
- Password\* (text input)
- Date of birth\* (text input with slashes and a calendar icon)
- What do you want to use myTaxiService for? \* (radio buttons for Taxi driver and Client)
- A large rectangular area with a diagonal cross, likely a placeholder for a profile picture or additional information.
- A "Sign up" button at the bottom left.



The image shows a mobile application interface for "myTaxiService". The registration form includes the following fields and options:

- Email\* (text input with a right arrow)
- Name\* (text input with a right arrow)
- Surname\* (text input with a right arrow)
- Gender\* (text input with a right arrow)
- Address\* (text input with a right arrow)
- Password\* (text input with a right arrow)
- Date of birth\* (text input with a right arrow)
- Are you a taxi driver? (toggle switch)
- A "Sign me up!" button at the bottom.



Registration

http://www.myTaxiService.com/Registration.html

Registration

Email\*

Name\*

Surname\*

Gender\* ☐ Male ☐ Female

Address\*

Password\*

Date of birth\*  /

What do you want to use myTaxiService for? \*

☒ Taxi driver ☐ Client

Driving license\*

Taxi license\*

IBAN\*

Sign up

myTaxiService

Insert your DRIVING license\*

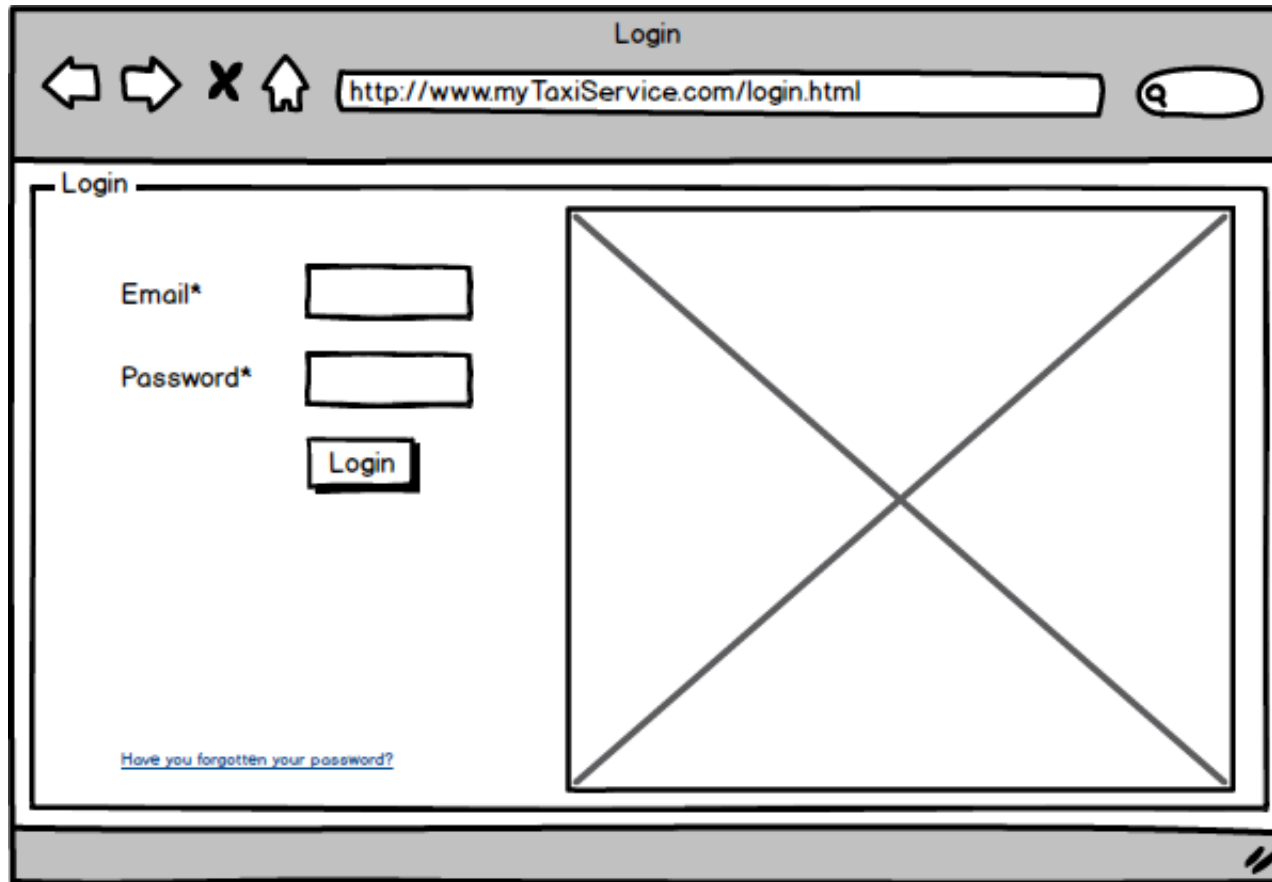
Insert your TAXI license\*

Insert your IBAN\*

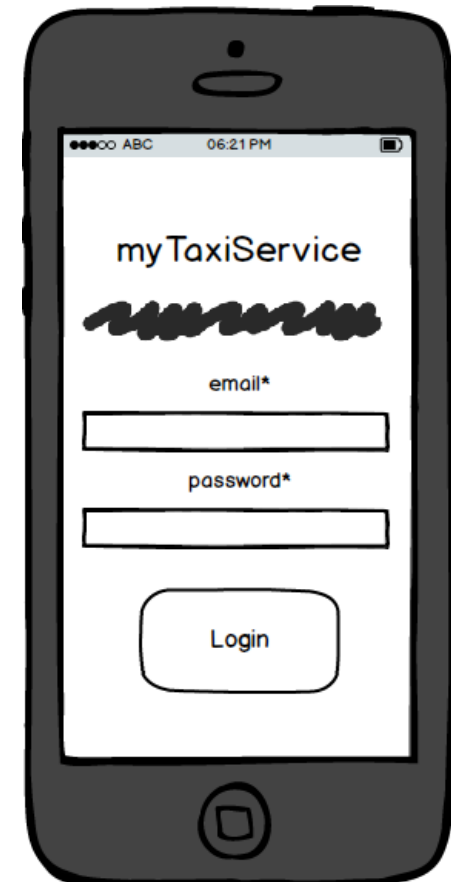
Confirm

### 4.3 Login page

The following images show the login pages available for a Guest from the web site or from the mobile application.



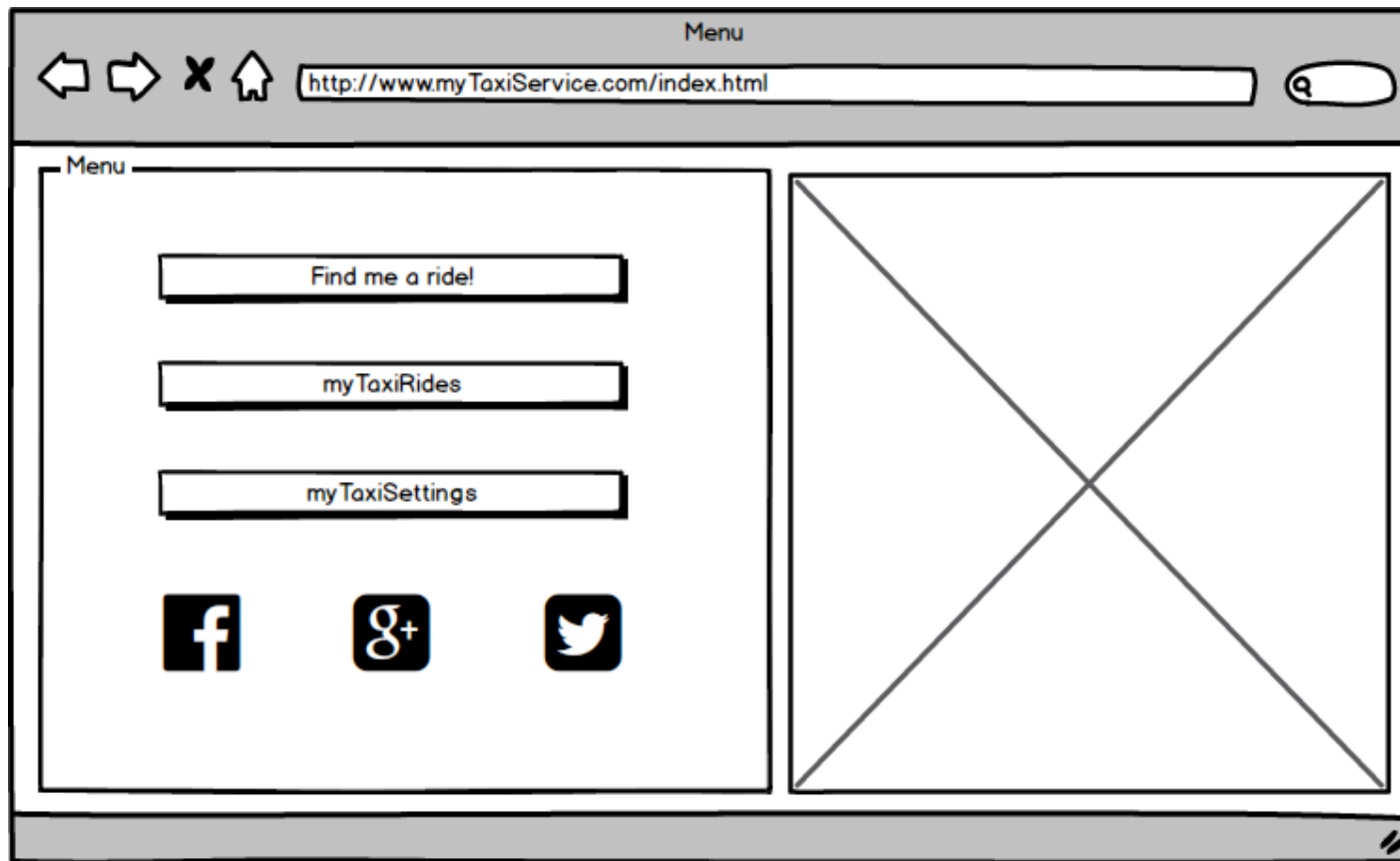
A hand-drawn illustration of a web browser window. The title bar says "Login". The address bar contains the URL "http://www.myTaxiService.com/login.html". The main content area has a "Login" heading. Below it are two input fields: "Email\*" and "Password\*", each followed by a rectangular input box. Below the "Password\*" field is a "Login" button. To the right of these fields is a large square area with a large 'X' drawn across it. At the bottom left, there is a link that says "Have you forgotten your password?".



A hand-drawn illustration of a mobile application interface. The status bar at the top shows "ABC" and "06:21 PM". The app title "myTaxiService" is at the top. Below it is a decorative wavy line. Then there are two input fields: "email\*" and "password\*", each followed by a rectangular input box. Below the "password\*" field is a "Login" button.

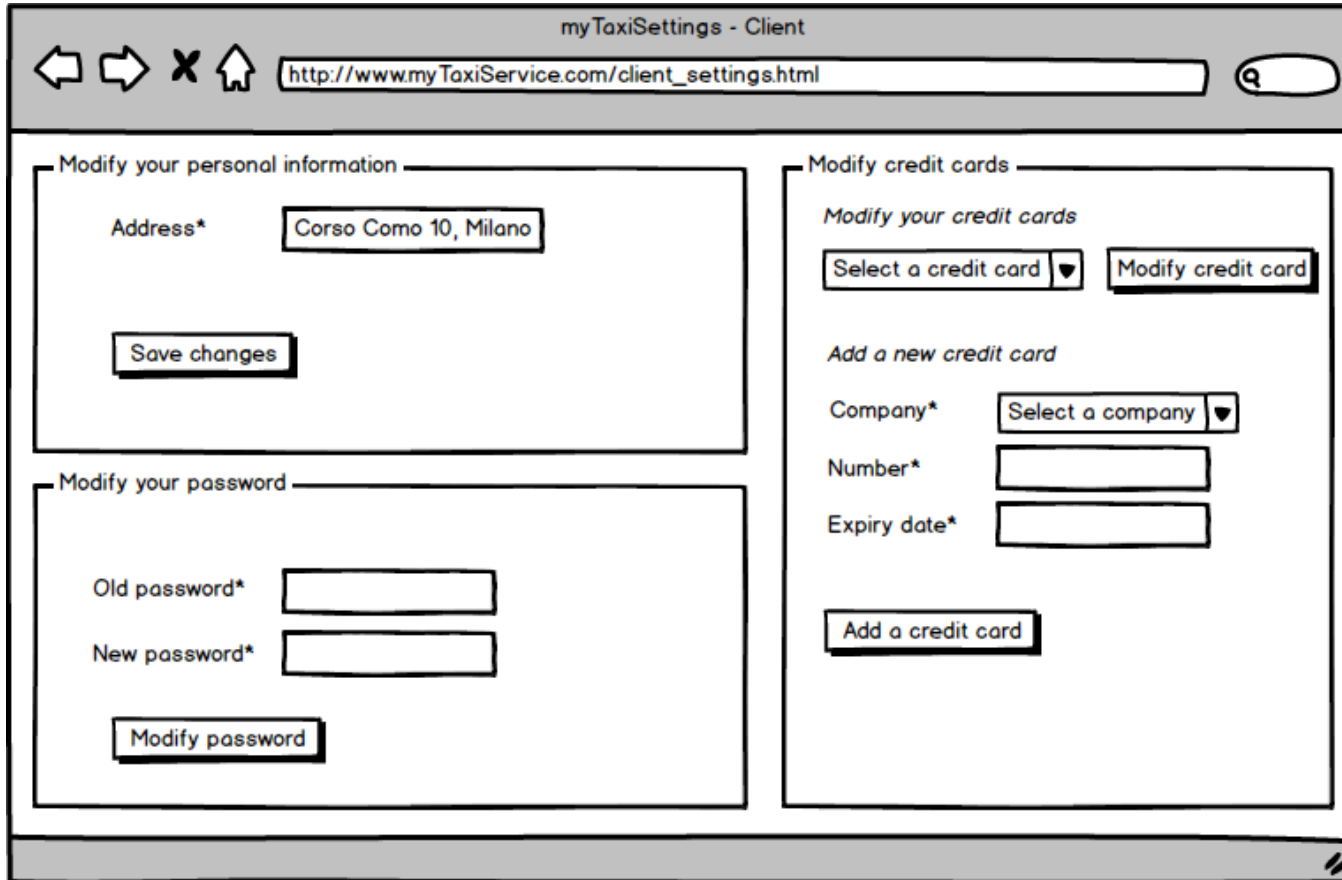
## 4.4 Menu

The following images show the menu page from which the Client can chose whether to find a new ride, see their pending and concluded rides or modify their settings.



## 4.5 Settings

The following images show the settings that can be modified by clients and taxi drivers.



The image shows a web browser window titled "myTaxiSettings - Client". The address bar displays "http://www.myTaxiService.com/client\_settings.html". The page is divided into two main sections: "Modify your personal information" and "Modify credit cards".

**Modify your personal information**

Address\*

**Modify your password**

Old password\*

New password\*

**Modify credit cards**

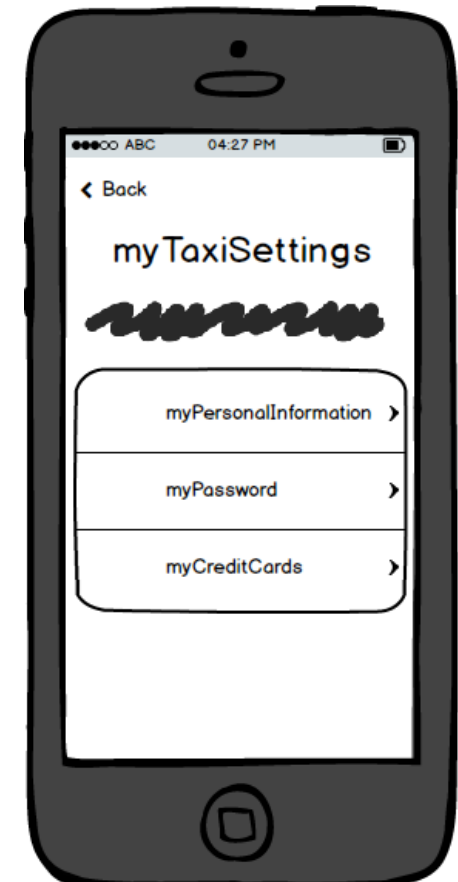
*Modify your credit cards*

*Add a new credit card*

Company\*

Number\*

Expiry date\*



myTaxiSettings - Taxi driver

http://www.myTaxiService.com/taxidriver\_settings.html

---

### Modify personal information

IBAN\*

Driving license\*

Address\*

### Modify your password

Old password\*

New password\*

### Modify/add taxis

*Modify your taxis*

*Add a taxi*

Plate\*

Total seats\*

Brand


Model

Photo

ABC 04:27 PM

< Back

## myTaxiSettings



- myPersonalInformation >
- myPassword >
- myPaymentsData >
- my Taxis >

## 4.6 Ride request

The following images show the pages that allow a Client to request a ride right now or to book a new one.

The image shows a web browser window titled "Client request". The address bar contains the URL "http://www.myTaxiService.com/clientRequest.html". The main content area is titled "Taxi request" and contains the following form elements:

- Start address\***: A text input field.
- Destination address\***: A text input field.
- Number of people\***: A numeric input field with a spinner.
- ☐ **Taxi sharing**: A checkbox.
- Departure time\***: A section containing:
  - Date**: A date input field with a calendar icon.
  - h**: A numeric input field for hours.
  - min**: A numeric input field for minutes.
- Create a new ride**: A button.

To the right of the form is a map showing a grid of streets. A green shaded area represents the start location, and a yellow line represents the route to the destination.

The image shows a mobile app interface for "myTaxiService". The status bar at the top shows "ABC" and "03:59 PM". The app has a logo and a list of options:

- Select origin of the ride >
- Select destination of the ride >
- Select time >
- Select number of people >
- Shared ride? ☐

At the bottom is a button labeled "Send request!".

## 4.7 Shared ride

The following images show the pages that allow a Client to join to a shared ride or create a new one.

Client request

http://www.myTaxiService.com/clientRequest.html

---

**Taxi request**

Start address\*

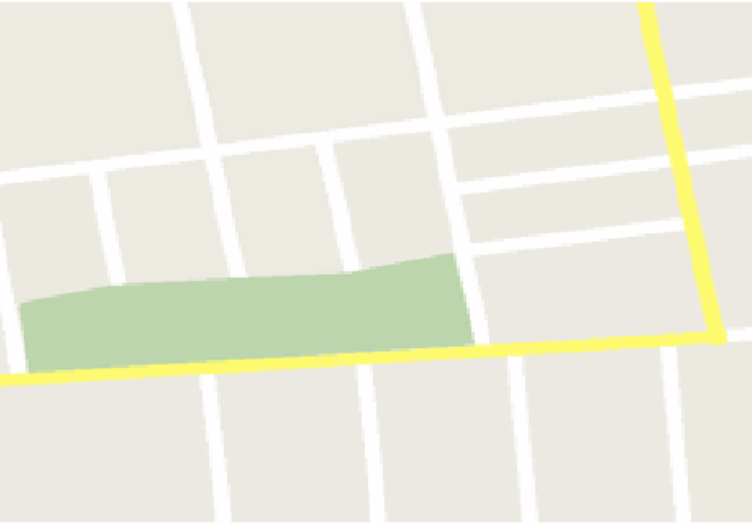
Destination address\*

Number of people\*

☒ Taxi sharing

Departure time\*

Date  h  min



**Shared rides - Availability**

17 17.30 19 24

Date	Time of departure	Price if you join	Current n. of people	Km	ID Taxi	Already allocated	
20/11/2015	17.45	13€	1	28	24366	<input checked="" type="checkbox"/>	<input type="button" value="Join"/>
20/11/2015	18.30	6€	2	14	-	<input type="checkbox"/>	<input type="button" value="Join"/>

ABC 11:44 PM

< back myTaxiService

select time slot

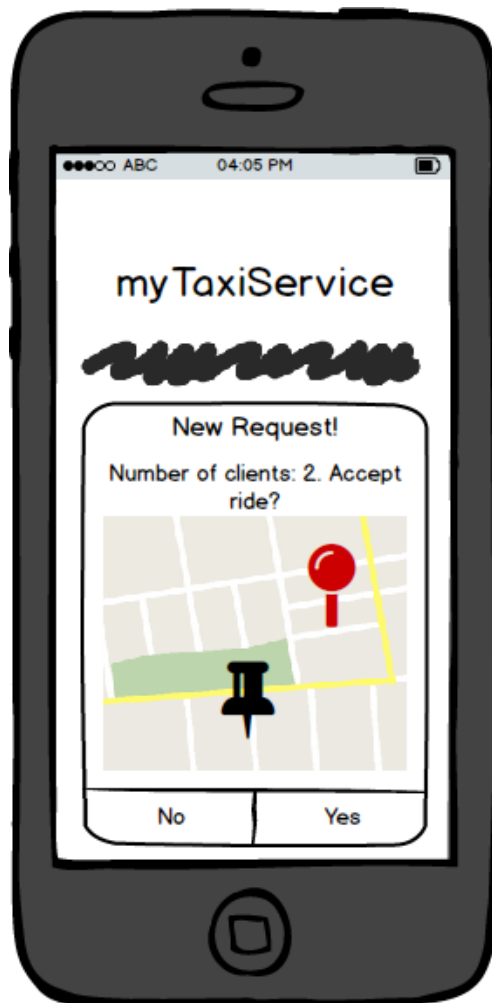
17 17.30 19 24

Date: 20/11/2015  
Time of departure: 17.45  
Price if you join: 13€  
Km: 28  
Number of people: 1  
Taxi ID: 24366  
Allocated

Date: 20/11/2015  
Time of departure: 18.30  
Price if you join: 6€  
Km: 14  
Number of people: 2  
Taxi ID: -  
Not Allocated

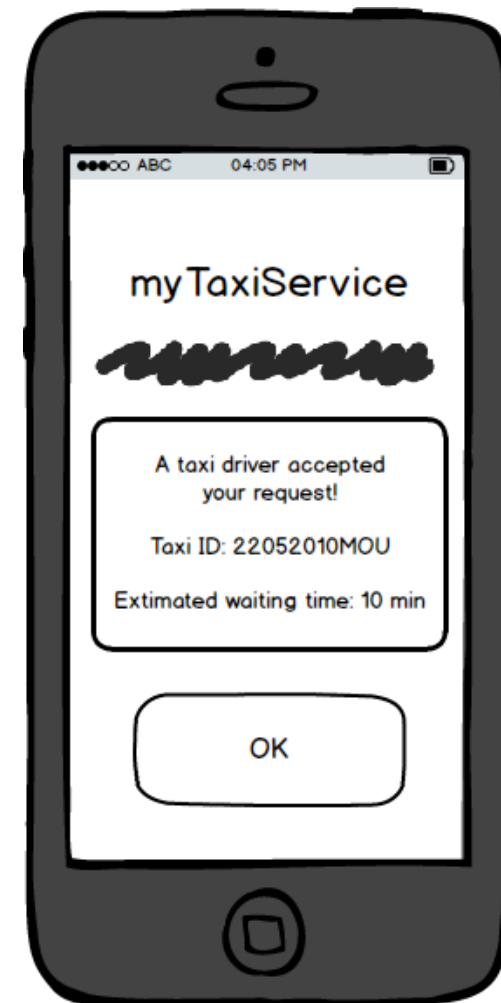
#### 4.8 Taxi request

The following image shows the notification received by the Taxi driver allowing them to accept or refuse a ride.



#### 4.9 Accepted request

The following image shows the notification received by the Client after a Taxi driver has accepted their taxi request.





## 4.10 Payment

The following images show the pages that allow a Client to choose the way they want to pay a certain ride: they can either choose a Credit Card that they had already inserted or to add a new one.

The image shows a web browser window titled "Ride payment". The address bar displays "http://www.myTaxiService.com/pay.html". The page content is divided into two sections:

- Pay with a credit card already inserted:** This section contains a dropdown menu labeled "Select a credit card" and a "Pay!" button.
- Pay with a new credit card:** This section contains several input fields, each with an asterisk indicating it is required:
  - Company\*: A dropdown menu labeled "Select a company".
  - Name\*: A text input field.
  - Surname\*: A text input field.
  - Number\*: A text input field.
  - Expiry date\*: A text input field.
  - CVV\*: A text input field.Below these fields is a "Pay! (and add the credit card)" button.

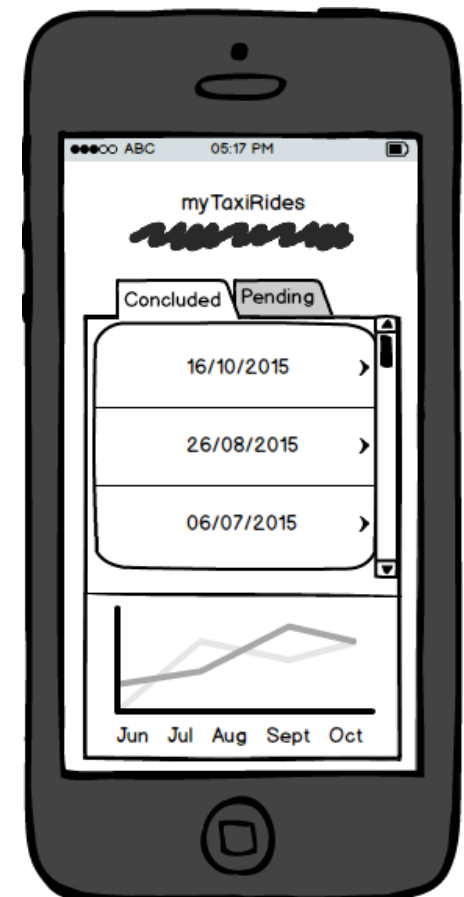
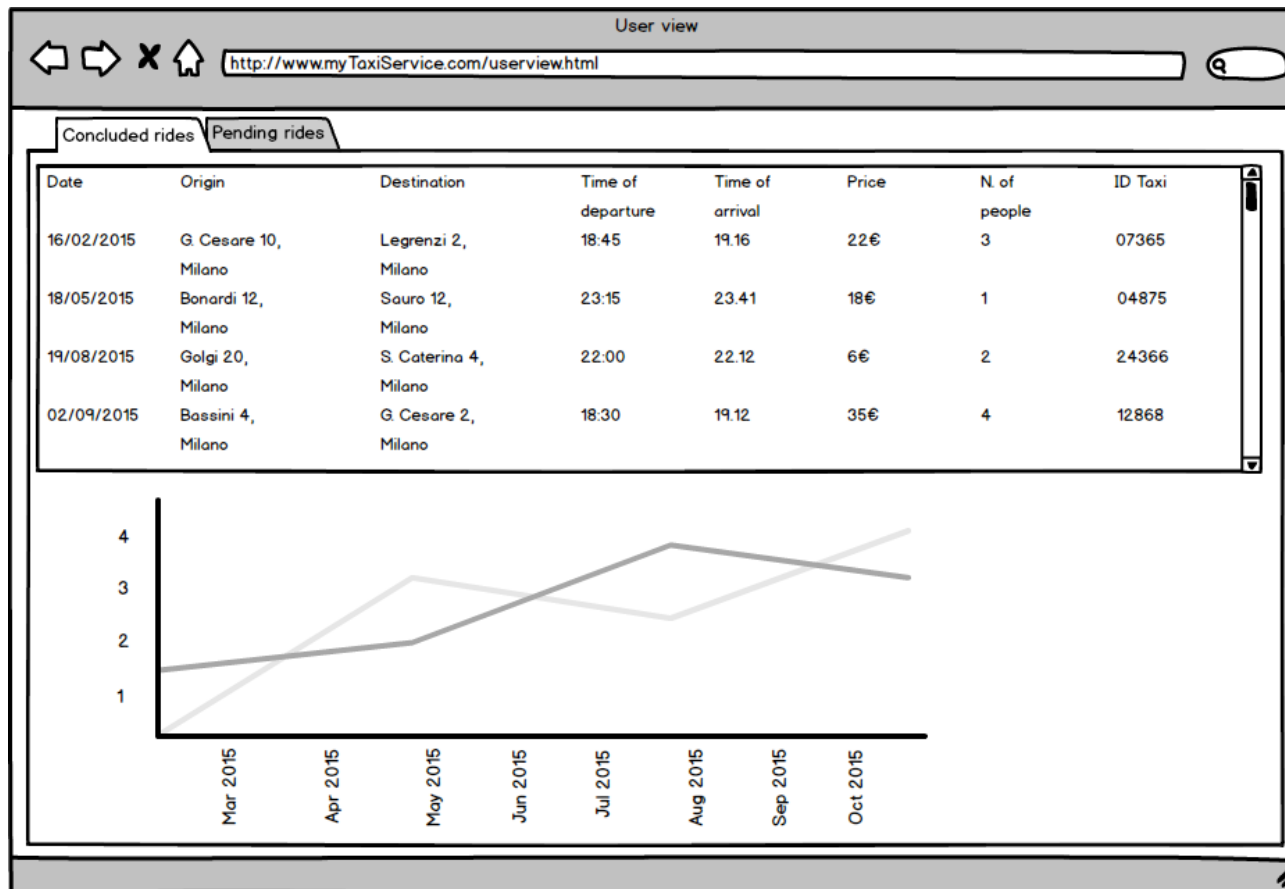
The image shows a mobile app interface for "myTaxiService". The status bar at the top shows "ABC" and "04:27 PM". The app's logo is at the top. Below the logo, there are two main sections:

- Choose an old Credit Card:** This section displays a card number "348904785485370".
- Add a new Credit Card:** This section includes a "Select company" label and three radio button options with logos: MasterCard, VISA, and PayPal.

At the bottom of the screen is a "GO" button.

#### 4.11 User view – Concluded rides

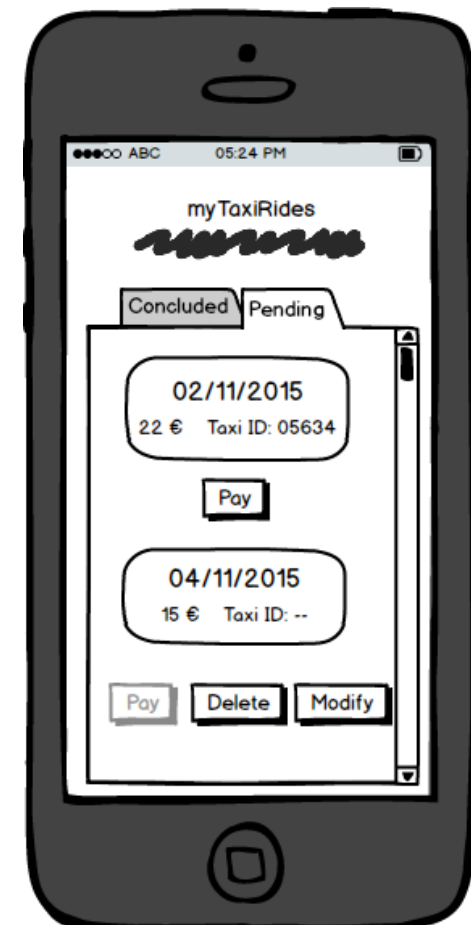
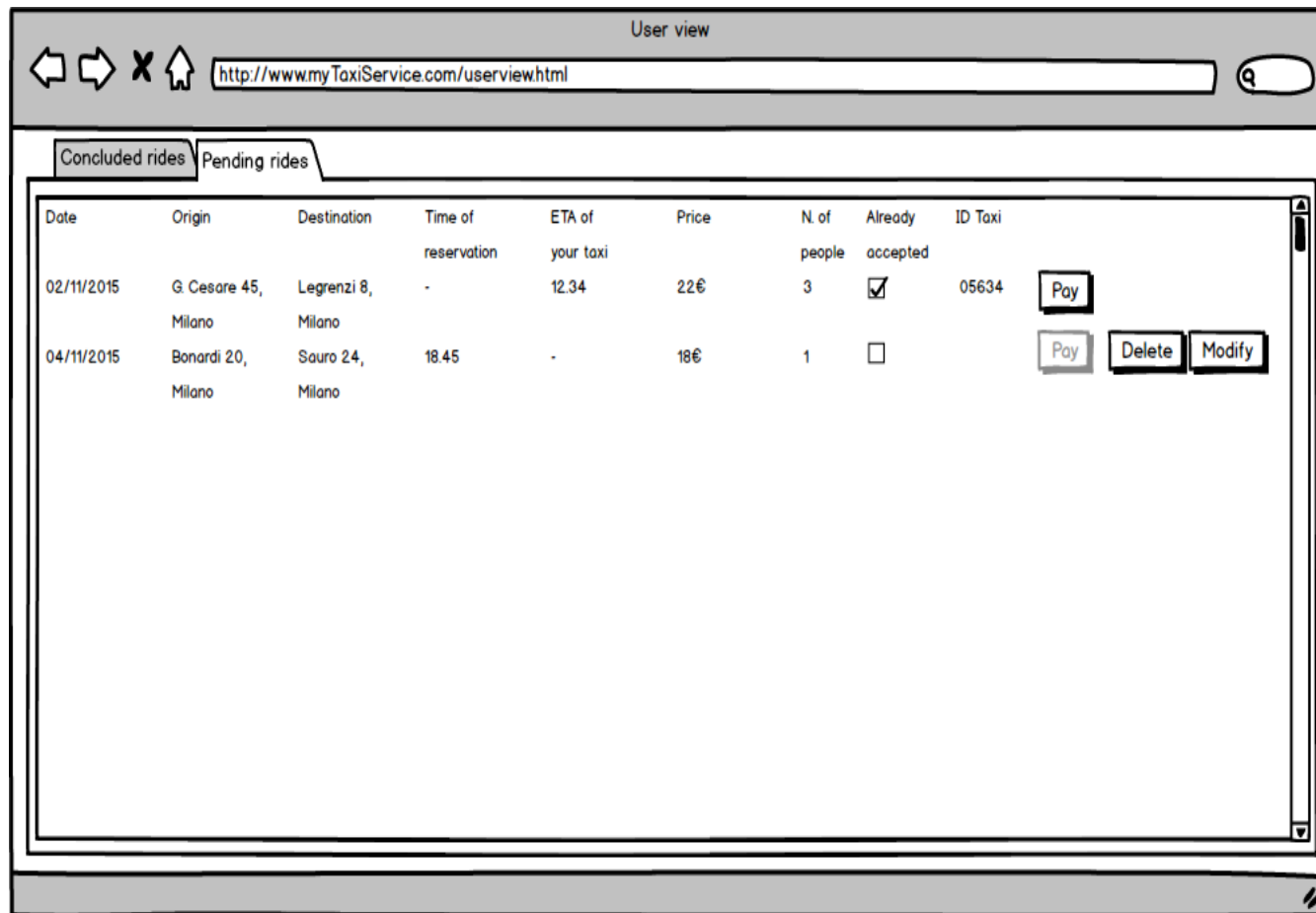
The following images show the pages that allow a user to see the list of their concluded rides with all the related information and the possibility to see a graph with the number of their concluded rides grouped by a certain period of time.



#### 4.12 User view – Pending rides

The following images show the pages that allow a Client to see the list of their pending rides with all the related information and the possibility to delete, modify and pay with a credit card.

In the example below you can see the buttons Delete and Modify only related to the booked ride because the others cannot be modified or cancelled. Moreover, the button Pay is disabled for the ride booked because has not been already accepted by a Taxi driver.



### 4.13 Administration panel

The following image shows the Administration panel (available only for the administrators of the system) with data grouped by type of information.

The screenshot displays the Administration panel of a taxi service. The browser address bar shows the URL <https://www.myTaxiService.com/administration.html>. The main content area is divided into three sections:

- Trip History Table:** A table with 8 columns: Date, Origin, Destination, Time of departure, Time of arrival, Price, N. of people, and See on the map. It contains 4 rows of trip data.
- Map Section:** Four map tiles showing taxi availability in different areas of Milan. The availability counts are 12, 3, 4, and 10.
- Errors Log:** A table with 4 columns: IDError, IDUser, OS, and Description. It contains 2 rows of error data.

Date	Origin	Destination	Time of departure	Time of arrival	Price	N. of people	See on the map
16/02/2015	G. Cesare 10, Milano	Legrenzi 2, Milano	18:45	19:16	22€	3	<input type="checkbox"/>
18/05/2015	Bonardi 12, Milano	Sauro 12, Milano	23:15	23:41	18€	1	<input type="checkbox"/>
19/08/2015	Golgi 20, Milano	S. Caterina 4, Milano	22:00	22:12	6€	2	<input type="checkbox"/>
02/09/2015	Bassini 4, Milano	G. Cesare 2, Milano	18:30	19:12	35€	4	<input type="checkbox"/>

IDError	IDUser	OS	Description
1	523454	iOS 9.1	App crashed
2	536474	iOS 8	Transact. failed

## 5 REQUIREMENTS TRACEABILITY

In this chapter all the functions listed and described in the RASD, both functional and non-functional, are linked to the ones described in this document.

### 5.1 Functions

RASD	SDD
[PF00] Client registration	Client <<UI>>, Manage users accounts
[PF01] Taxi driver registration	Taxi driver <<UI>>, Manage users accounts
[PF02] Login	Client <<UI>>, Taxi driver <<UI>>, Manage users accounts
[PF03] Password recovery	Client <<UI>>, Taxi driver <<UI>>, Manage users accounts
[PF04] Personal information management	Client <<UI>>, Taxi driver <<UI>>, Manage users accounts
[PF05] Add payment option	Client <<UI>>, Manage credit cards
[PF06] Delete payment option	Client <<UI>>, Manage credit cards
[PF07] Add new taxi	Taxi driver <<UI>>, Manage taxis
[PF08] Taxi driver working state and taxi choice	Taxi driver <<UI>>, Manage rides allocation
[PF09] Request taxi	Client <<UI>>, Manage routes
[PF10] Reservation of a taxi	Client <<UI>>, Manage routes
[PF11] Zone identification	Manage routes, Taxi driver <<UI>>
[PF12] Queue handling	Manage rides allocation (Queue zones)
[PF13] Taxi driver notification	Manage rides allocation (Amazon SNS), Taxi driver <<UI>>
[PF14] Taxi driver answer notification	Taxi driver <<UI>>
[PF15] Compute waiting time	Manage routes
[PF16] Compute ride cost	Manage routes
[PF17] Client notification	Manage rides allocation (Amazon SNS), Client <<UI>>
[PF18] Ride payment	Client <<UI>>, Manage credit cards
[PF19] Conclude ride	Taxi driver <<UI>>, Manage rides
[PF20] List of rides	Client <<UI>>, Taxi driver <<UI>>, Manage routes, Manage rides
[PF21] Modify reservation	Client <<UI>>, Manage routes
[PF22] Cancel reservation	Client <<UI>>, Manage routes
[PF23] Ride cancelation	Taxi driver <<UI>>, Manage rides
[PF24] Ride interruption	Taxi driver <<UI>>, Manage rides
[PF25] Administration panel	Admin <<UI>>
[PF26] Tutorial	Client <<UI>>, Taxi driver <<UI>>
[PF27] Zone change	Taxi driver <<UI>>, Manage rides allocation

## 5.2 Non-Functional Requirements

<b>RASD</b>	<b>SDD</b>
Functional suitability	The functions are satisfied by the implementation of the classes contained in the components described in Chapter 2.2, 2.3.
Performance efficiency	The choice of AWS guarantees, by means of a dynamic allocation of the resources, the satisfaction of the performance efficiency requirements.
Compatibility	It is satisfied by a correct implementation of the components described in Chapter 2.2, 2.3 that communicate with the outside, (Manage routes, Manage users accounts, Admin <<UI>>).
Usability	It is satisfied by interfaces described in Chapter 4.
Reliability	It is satisfied by a correct configuration of AWS.
Security	It is guaranteed by AWS and by keeping track of a log in the system.
Maintainability	The satisfaction is satisfied by means of a good programming of the components described in Chapter 2.2, 2.3.
Portability	It is guaranteed in both the website and mobile application: the website pages described in Chapter 4 can be used in almost all browsers, while the mobile applications can be used by all the people that have a phone with a certain operating system.

## 6 REFERENCES

- Agile Modelling website: <http://www.agilemodeling.com/essays/umlDiagrams.htm>
- Amazon Web Services documentation: <https://aws.amazon.com/>
- “UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition)”, by Martin Fowler

## 7 APPENDIX

### 7.1 Used tools

- Microsoft Word 2013 to write this document
- Microsoft Visio 2013 to create UML diagrams
- Balsamiq Mockup 3 to create mock-ups of the mobile application and of the web site

### 7.2 Attached documents

#### 7.2.1 Client's document

##### **The problem: myTaxiService**

##### **Part I**

The government of a large city aims at optimizing its taxi service. In particular, it wants to: i) simplify the access of Clients to the service, and ii) guarantee a fair management of taxi queues.

Clients can request a taxi either through a web application or a mobile app. The system answers to the request by informing the Client about the code of the incoming taxi and the waiting time.

Taxi drivers use a mobile application to inform the system about their availability and to confirm that they are going to take care of a certain call.

The system guarantees a fair management of taxi queues. In particular, the city is divided in taxi zones (approximately 2 km<sup>2</sup> each). Each zone is associated to a queue of taxis. The system automatically computes the distribution of taxis in the various zones based on the GPS information it receives from each taxi. When a taxi is available, its identifier is stored in the queue of taxis in the corresponding zone.

When a request arrives from a certain zone, the system forwards it to the first taxi queuing in that zone. If the taxi confirms, then the system will send a confirmation to the Client. If not, then the system will forward the request to the second in the queue and will, at the same time, move the first taxi in the last position in the queue.

Besides the specific user interfaces for Clients and Taxi drivers, the system offers also programmatic interfaces to enable the development of additional services (e.g., taxi sharing) on top of the basic one.

##### **Part II**

A user can reserve a taxi by specifying the origin and the destination of the ride. The reservation has to occur at least two hours before the ride. In this case, the system confirms the reservation to the user and allocates a taxi to the request 10 minutes before the meeting time with the user.

##### **Part III**

A user can enable the taxi sharing option. This means that he/she is ready to share a taxi with others if possible, thus sharing the cost of the ride. In this case the user is required to specify the destination of all rides which he/she wants to share with others. If others are willing to start a shared ride from the same zone going in the same direction, then the system arranges the route for the Taxi driver, defines the fee for all persons sharing the taxi and informs the Clients and the Taxi driver.

### 7.3 Hours of work

Andrea Autelitano has worked on this document for 66 hours.

Marco De Cobelli has worked on this document for 49 hours.

Matthew Rossi has worked on this document for 66 hours.