



Design Document

Software Engineering 2: “myTaxiService”

Fioratto Raffaele, Longoni Nicolò
Politecnico di Milano

A.Y. 2015/2016

Version 1.0

December 4, 2015

Contents

1	Introduction	4
1.1	Purpose	4
1.2	Scope	4
1.3	Definitions, Acronyms, Abbreviations	4
1.3.1	Acronyms	4
1.3.2	Abbreviations	5
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	6
2.3	Component View	7
2.4	Deployment View	9
2.5	Runtime View	9
2.5.1	Passenger Login	10
2.5.2	Passenger's Request	11
2.5.3	Taxi Driver Registration	12
2.5.4	Taxi Driver Availability status change	13
2.6	Component Interfaces	13
2.6.1	Passenger UI Logic Interface	13
2.6.2	Taxi Driver UI Logic Interface	14
2.6.3	System Administrator UI Logic Interface	14
2.6.4	Access Control Interface	15
2.6.5	Ride Manager Interface	15
2.6.6	Queue Manager Interface	15
2.6.7	Databases interface	16
2.6.8	Maps API	16
2.7	Selected architectural styles and patterns	16
2.8	Architectural styles	16
2.9	Design patterns	17
2.10	Other design decisions	18
3	Algorithm Design	19
3.1	Find a queue knowing the position	19
3.2	Add a taxi to a queue	19
3.3	Assign a taxi to an incoming request	20
4	User Interface Design	21
4.1	Homepage	21
4.2	Registration	22
4.3	Login	24
4.4	Recovery Password	26
4.5	Request	28
4.6	Reservation	30
4.7	Driver Interface	32
4.8	Administrator	34

4.9	UX Diagram	35
5	Requirements Traceability	36

1 Introduction

1.1 Purpose

This documentation represents the Design Document (DD). It describes completely the system in terms of components, analyzing their internal and external interactions (with the actors). It starts off with the description of the system given in the Requirements Analysis and Specification Document. In this document the system is treated like a white box, i.e. the document “opens” it and then dissects it starting with a high level overview to a more detailed representation. This document is addressed to all developers and programmers who have to implement the actual software.

1.2 Scope

The aim of the project is to create a new brand system that optimizes an existing taxi service. The system will be capable of automatizing and/or simplifying certain processes during requests or reservations of taxis. It will also guarantee a fair management of taxi queues. New passengers can sign up for service inserting some basic information in order to use service’s features as soon as possible. A passenger can request a taxi using web service or through mobile application after registration. The system will be able to localize precisely the position of the passenger, determining taxis that are available near him/her. The system will select a taxi and then it will forward the request to one of its driver. Upon confirmation, the system will notify the customer about the successful completion of the operation and the ETA of the taxi. A passenger can also reserve a taxi by specifying time and date, but they have to do it with at least two hours in advance. Cancellation is also permitted. The system will actually process the request ten minutes before the time specified during the reservation in the same way as described previously.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Acronyms

- DD: Design Document
- RASD: Requirement Analysis and Specification Document
- MVC: Model View Controller
- SOA: Service Oriented Application
- JEE: Java Enterprise Edition
- FSA: Finite State Automaton
- API: Application Programming Interface
- FSA: Finite State Automaton
- IDE: Integrated Development Environment

- URL: Uniform Resource Locator

1.3.2 Abbreviations

- [Gn]: n-goal
- [Rn]: n-functional requirement
- [Dn]: n-domain assumption
- [An]: n-assumption

1.4 Reference Documents

- myTaxiService Specification Document
- myTaxiService RASD
- DD Structure Template

1.5 Document Structure

- Section 1: Introduction, it gives a description of this document, some basic information in order to clearly understand the subsequent sections.
- Section 2: Architectural Design, it describes the software to be designed starting with a high level representation and then dissecting it providing a more detailed analysis. Diagrams are provided to better clarify this part.
- Section 3: Algorithm Design: it gives a description of the main algorithms that are implemented in the software, pseudo-code or flow diagrams are provided to describe them in a clearer way.
- Section 4: User Interface Design: starting with mockups presented in the RASD, this part further explains interactions between actors and the system, and shows how it will look like.
- Section 5: Requirements Traceability: this part explains how requirements defined in the RASD map into the design elements that we shall define in this document.
- Section 6: References: this section provides a list of external documents and materials that allowed the composition of this documentation.

2 Architectural Design

2.1 Overview

The system is constituted by a three-tier architecture, in fact it's divided into GUI, application programs and database. The first one isn't only on users' devices, since the interface stays on users' smartphones and browsers, but the logic part is located on server side. On another server lays the application part where the system manages interactions between components and elaborates inputs coming from GUI. Database storage stays on a separated server from the logic one, it contains information about taxi drivers and their cabs, in addition to passenger personal details.

2.2 High level components and their interaction

The following diagram shows the composition of the system in terms of high level components.

In the next section these components are explained and further analyzed to describe the overall system.

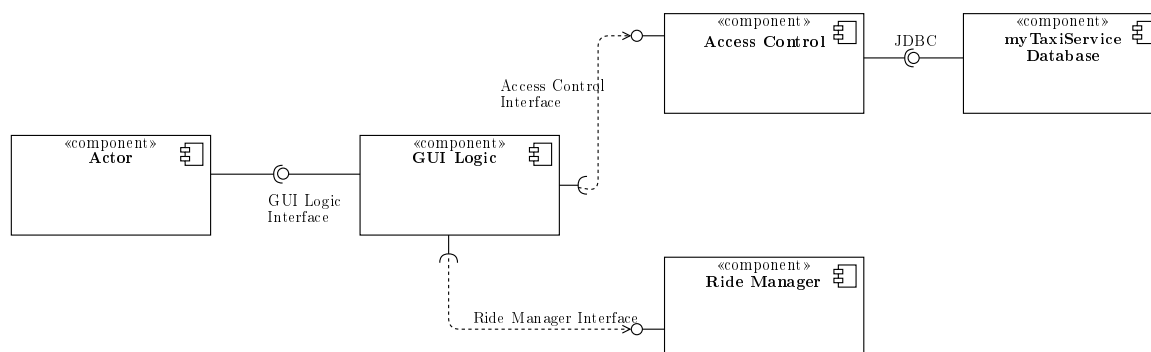


Figure 1: High level components diagram

As one can see in the diagram, the system is composed in three distinct levels:

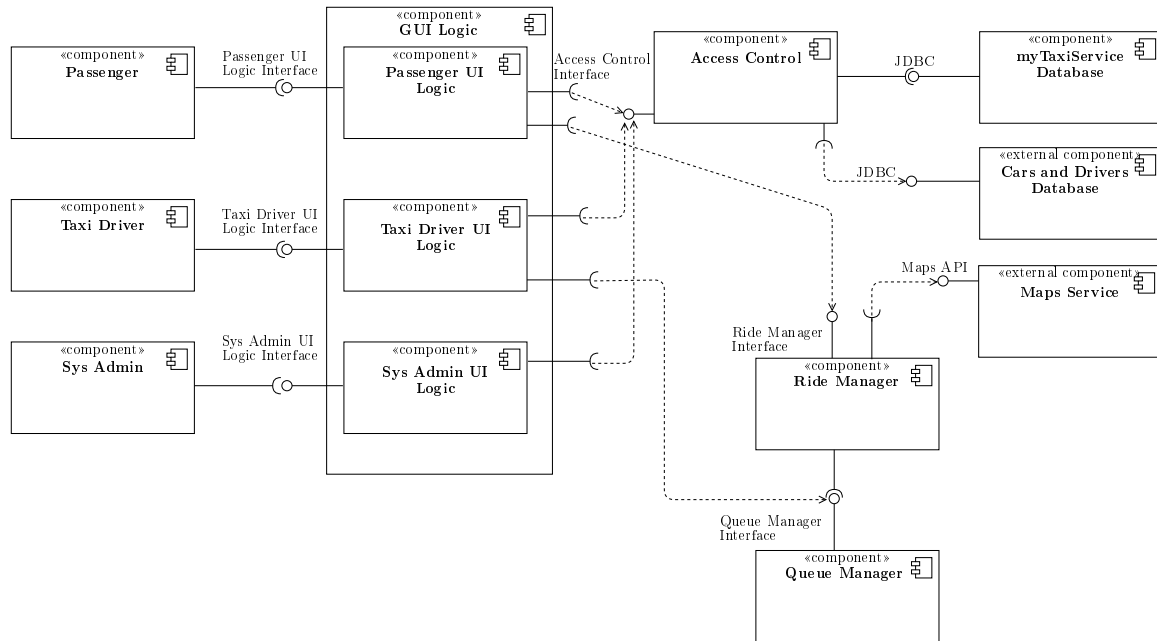
- GUI Logic
- Access Control and Ride Manager
- Database

These components can be easily mapped in the MVC architectural style (as later described in Selected architectural styles and patterns section). Each component is also mapped in a specified tier among the server architecture as briefly described in the Overview section and further explained in the Selected architectural styles and patterns section. The remaining component models all possible actors, in fact it represents physical elements (a smartphone or a computer with an appropriate software installed on them) that actually interact with the system.

The diagram clearly shows that there is a single macro-component that exposes a set of functionalities and methods an actor can use. This macro-component is the only

entry point available to all the actors, this choice has been made to preserve a simple architecture and also to increase the cohesion and reusability of the component itself. This component is actually composed of a bunch of sub-components as described in the next section Component View, each sub-component carries out specific functionalities for each actor defined in the RASD.

2.3 Component View



Actors It's possible to see from this diagram a generic "Actor" component is now split in three different components, one for each actor, the main reason for this decision is that each actor accesses and uses a specific interface supplied in one of the three sub-components in the GUI Logic component. Methods that comprise such interfaces are described in the Selected architectural styles and patterns section.

GUI Logic The GUI Logic component hosts the interface logic suitable for using both a web browser and a mobile application, this component has the main role of collecting all requests from external devices of all actors, and then forward them to the right back-end component.

Access Control The Access Control component manages user authentications, and their permissions, ensuring that only registered users can benefit from certain functions of the system.

Since these functionalities are similar to all actors, this component is unique, and methods in its external interface are designed so that it is aware of what type of actor is performing a specific action. The advantage of using such design is to increase flexibility and reusability without sacrificing simplicity of the overall system.

This component is also in charge of registering new passengers and taxi drivers, regarding formers it provides functionalities to generate a random password and a confirmation

E-mail or SMS containing also an activation link, while latter it checks whether some conditions are met before actually enabling a new taxi driver to authenticate in the system.

Ride Manager The Ride Manager component is in charge of managing all rides instances that are being served. This component creates, keeps track, terminates each ride managed by the system. Each instance can be seen as a FSA, and methods in component external interface implement a state transition in a automaton, this is further described in the Selected architectural styles and patterns section.

This component manages both requests and reservations, in fact a reservation can be seen as a “delayed” request, thus with a little addition in algorithms, we can achieve both functionalities with little effort, maximizing reusability of this component.

Queue Manager This component manages all queues belonging to the city map, one queue for each zone. The component strictly cooperates with the Ride Manager, but its external interface is generic enough to abstract actual implementation of policies for managing queues. This allows two components to be separated and more flexible, whereas the architecture still remains simple and easy to understand, it is possible realize different implementations of the Queue Manager, for example a generic one for most of the zones, and a specific one for certain zones, for different reasons, for instance traffic load higher than normal, or a higher or a less probability to receive a request from a client in a specific zone. These policies can be considered during the implementation phase.

The component also manages working state of Taxi drivers, it allows them to change between unavailable and available state then it removes or inserts them in an appropriate queue according to their location in the map (further described in the Algorithm Design section)

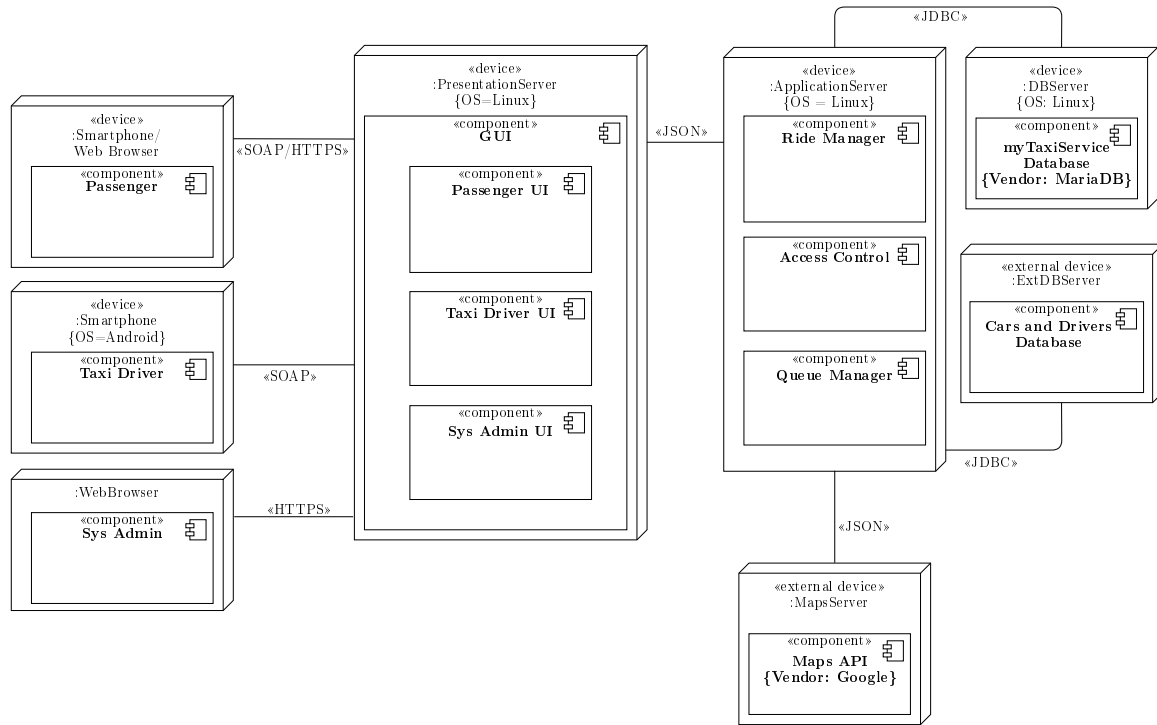
myTaxiService Database This database contains all information about registered users, both passengers and taxi drivers. Moreover it records all reservations and requests, active and terminated.

External Maps Services This external component is used to localize precisely where departure and arrival points are. Geolocalization systems are difficult to create without adequate instruments that can trace every route in the city and GPS systems. For this reason, we decided to entrust map services to an external company.

External Cars and Drivers database This database contains information about taxi drivers such as names, surnames, addresses, phone numbers, taxi driver license numbers and cab numbers. myTaxiService relies on this external data to control if all information about taxi drivers are legal.

2.4 Deployment View

The following picture is a deployment diagram for the myTaxiService system, each component showed in previous sections are mapped in a device and also communication protocols used between devices are showed.



It can be noticed that each Actor corresponds to a single device. Regarding mobile applications installed on devices, they will be developed using existing SDK frameworks which provide: already well defined interfaces and libraries to implement the rendering of the GUI, dealing with user inputs, and abstractions to communicate with the server. In database devices a MySQL implementation is used and exposed using JDBC which is comprised like a set of interfaces that are used in the Business Logic Tier and later discussed in the Component Interfaces section.

JSON is used like a mean to encapsulate data exchanged between tiers, this brings the advantage to make independent the programming language used for implementing the overall system.

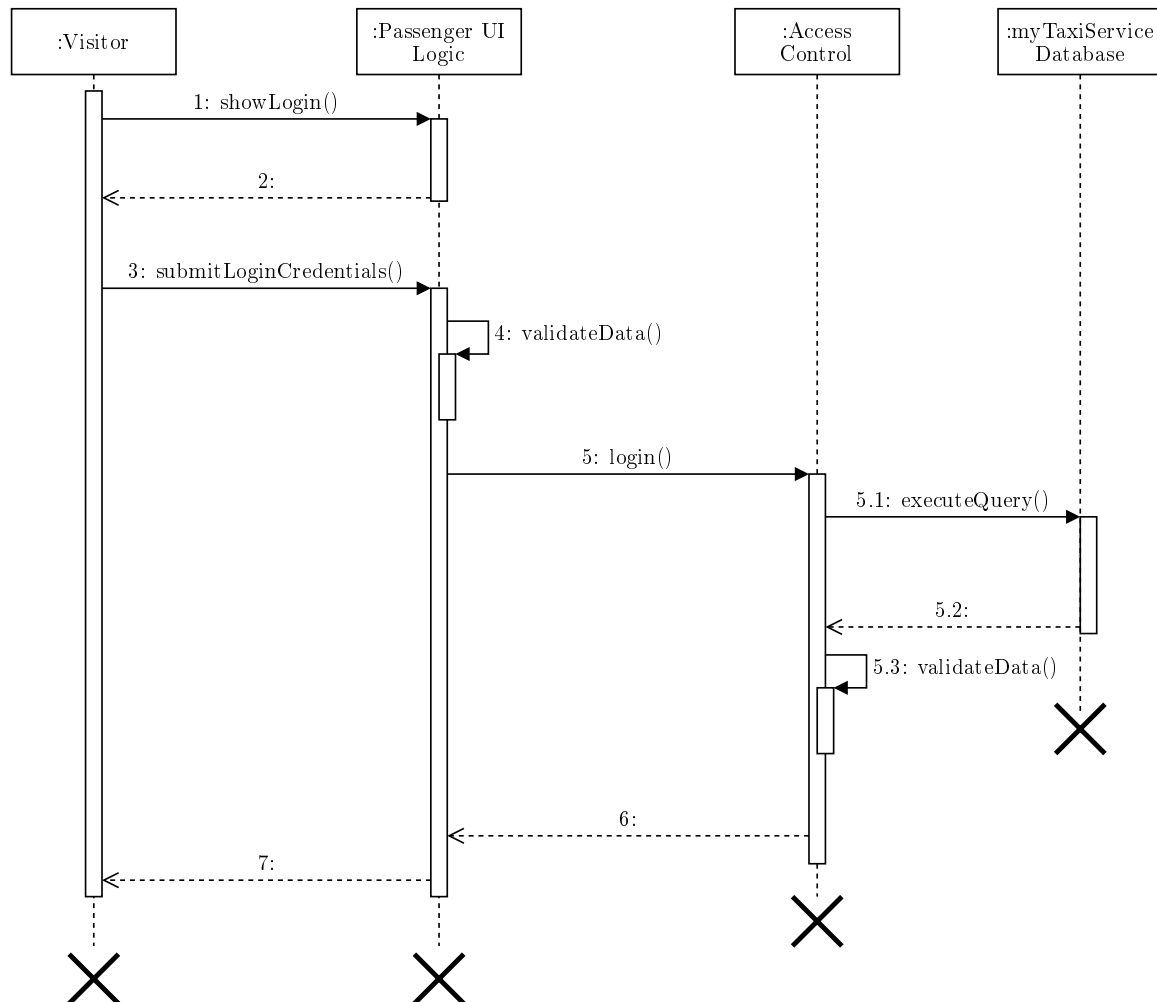
2.5 Runtime View

This section presents some sequence diagrams that explain interaction between components defined in previous sections. This will further clarify how components work in the system as a whole, and also in which order should methods be called in order to accomplish a specific task.

These diagrams don't take into account possible exceptions that can occur during the execution of a task, because of this complexity of the diagrams would increase, making more difficult their comprehension of them,. Also the notation used don't really implement a way for dealing with possible errors or exceptions.

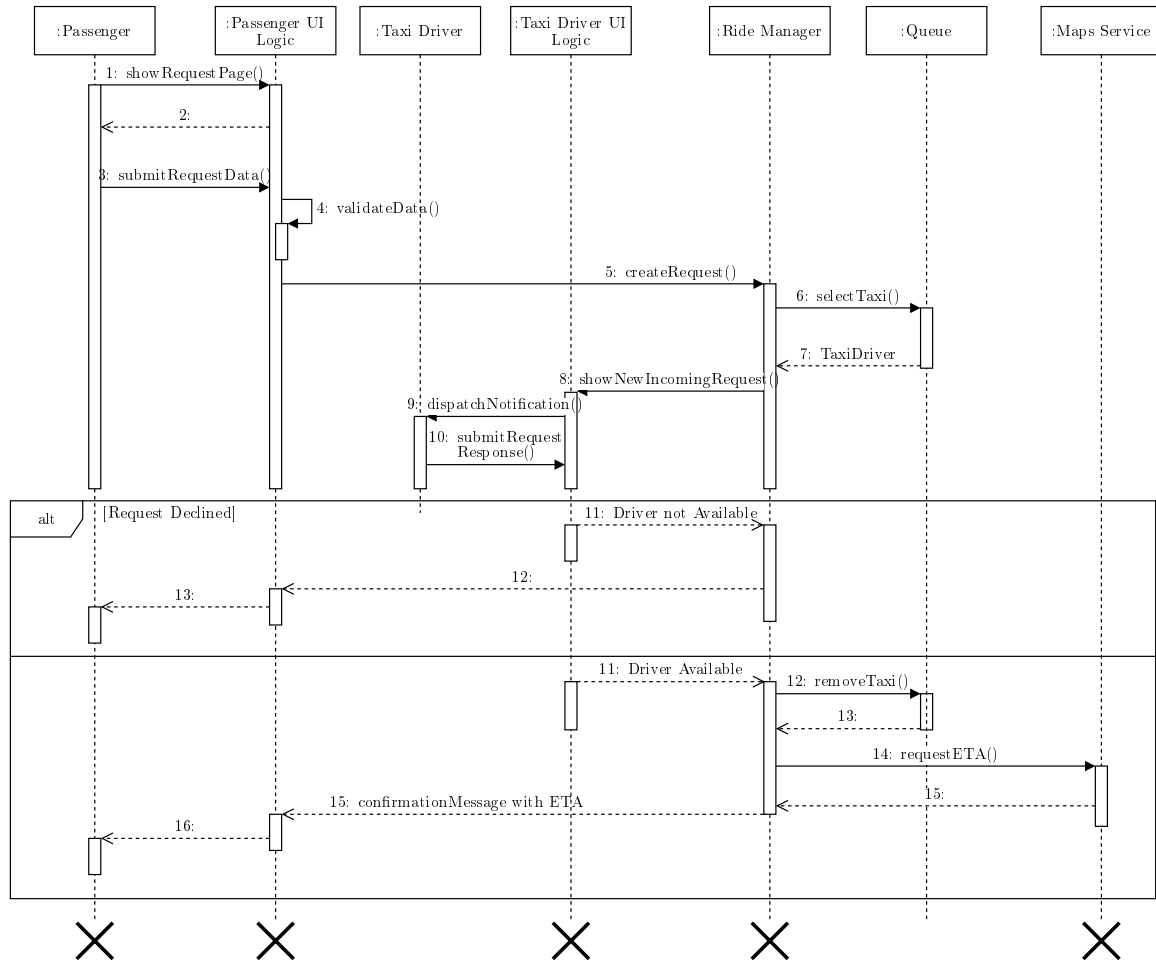
2.5.1 Passenger Login

The following describes a sequence of actions performed when a visitor who is supposed to be already registered passenger, logs in the system.



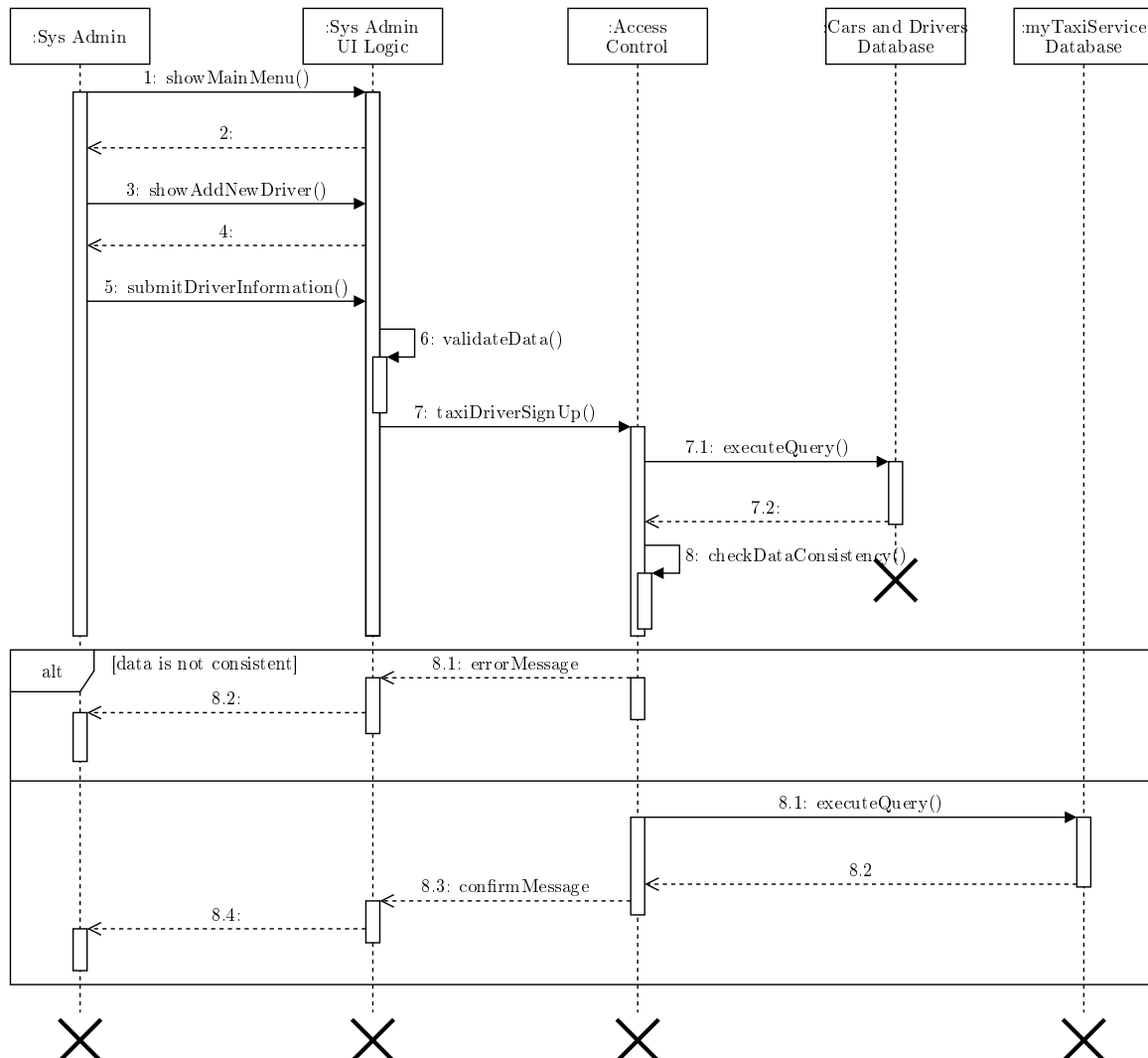
2.5.2 Passenger's Request

The following describes the sequence of actions performed when a passenger wants to perform a request.



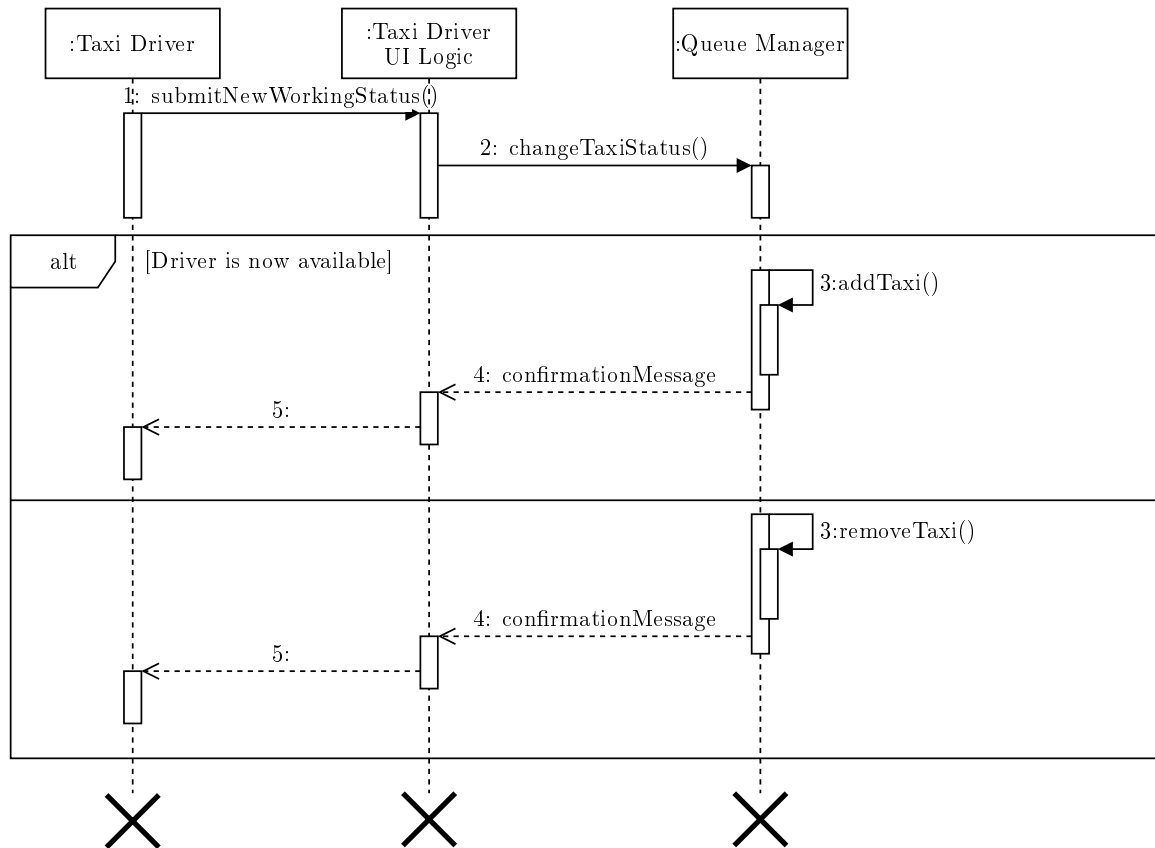
2.5.3 Taxi Driver Registration

The following describes the sequence of actions performed when a new Taxi Driver has just been hired and is registered into the system so that he/she can begin to work.



2.5.4 Taxi Driver Availability status change

The following describes the sequence of actions performed when a Taxi Driver is willing to change his/her working status.



2.6 Component Interfaces

In this section, all interfaces that connect each component in the Component View are explained and analyzed.

2.6.1 Passenger UI Logic Interface

This interface provides an entry point to the system for passengers.

The following is a list of the methods provided by this interface, they are all pretty self-explanatory:

- showHome()
- showLogin()
- showSignUp()
- showRequestPage()
- showReservationPage()

- `showRequestInformation()`
- `submitLoginCredentials()`
- `submitRegistrationCred()`
- `submitRequestData()`
- `submitReservationData()`

The interface comprised a series of methods that passengers can invoke. Each `show*()` method maps to a specific web page or screen in the mobile application, the actual invocation takes place when a passenger loads in his/her browser or in one screen of the application. Each page or screen has some sort of navigation methods so that it can call other `show*()` methods present in this interface.

The `submit*()` methods are called when a passenger wants to send data to system, for instance when he/she has just completed the filling of a form, these methods validate inputs and forward the request to the appropriate component in the system back-end, eventually they show an informative page or a screen that notify the user whether the operation is completed or not, and in the latter case why.

2.6.2 Taxi Driver UI Logic Interface

This interface provides the entry point to the system for the taxi driver. The list of the methods is the following:

- `showLogin()`
- `submitLoginCredentials()`
- `submitResponseForIncomingRequest()`
- `submitNewWorkingStatus()`
- `showNewIncomingRequest()`

Meaning and functional aspects of these methods are more or less similar to what has been explained in the previous section, one point to note is that this time, these methods are invoked only by the mobile application installed in the taxi driver's smartphone, so there is no web interface available to this specific actor.

2.6.3 System Administrator UI Logic Interface

This interface provides the entry point to the system for the administrator.
List of methods:

- `showLogin()`
- `showMainMenu()`
- `showAddNewDriver()`
- `showRemoveDriver()`
- `showModifyDriver()`

- `submitLoginCredentials()`
- `submitDriverInformation()`
- `submitDeleteRequest()`

It can notice that pattern names for these methods are the same of previous sections.

2.6.4 Access Control Interface

This interface supplies all the methods that enforce authentication and right management in the system.

List of methods:

- `login()`
- `logout()`
- `passengerSignUp()`
- `taxiDriverSignUp()`
- `passengerDelete()`
- `taxiDriverDelete()`
- `taxiDriverModifyCredentials()`
- `modifyPassengerCredentials()`

2.6.5 Ride Manager Interface

This interface contains methods to create and to manage a single ride instance, both requests and reservations.

List of methods:

- `createRequest()`
- `acceptRequest()`
- `declineRequest()`
- `createReservation()`

2.6.6 Queue Manager Interface

This interface comprises methods that manage the availability of taxis and supplies functions to add or to remove taxis.

List of methods:

- `addTaxi()`
- `removeTaxi()`
- `assignTaxi()`

- `changeTaxiStatus()`

2.6.7 Databases interface

This interface is not directly implemented in the system-to-be, instead an existing third party library (JDBC) is used, it supplies its own set of interfaces to the system that can manipulate the database.

2.6.8 Maps API

As with the previous section, also this interface is exposed to a third party component so that it can use a series of methods that are suitable in the system-to-be.

2.7 Selected architectural styles and patterns

The following section lists and describes the architectural styles that represent feasible choices during development and deployment phases. All requirements specified in the RASD are taken into account with also design choices described in this document.

2.8 Architectural styles

3-tier architecture As implicitly explained in the Overview section, this is perhaps the most important style, also because all component diagrams and also the deployment view are modelled against this architecture, from the latter diagram we can infer that tier map in this way:

- GUI Logic to Presentation tier
- Ride Manager, Access Control, Queue Manager, Maps to Logic tier
- Databases to the Data tier

Advantages coming from use of this style is that it achieves a high level of modularity and flexibility of the overall system, also each tier can be designed and dimensioned based on the computational and power needed for it, moreover, as described in non-functional requirements in the RASD, each component is isolated with the respect to others to increase security in each of them and also in the overall system. One possible drawback is the increasing complexity of interfaces to design, in fact every tier has to communicate with others, also each user interaction corresponds to many API method calls within the system, which in turn trigger some further processes of exchanged data between tiers. This can produce a loss of performances and raise response times, but this is mitigated by the fact that our interfaces are simple enough and also require little of data to process when are passed from one tier to another.

SOA This style is used by the system for both services that are offered to “Actors” and third party services that the system uses to achieve its goals. Thus the system acts both as a service provider and as a service requester.

Services provided are of course the ones that derive from goals already described in the RASD (request and reservation of taxis), plus some accessory functionalities that we also described in the RASD (login, signup, etc.).

Requested services are the Maps Service and the external car and driver database that are needed in order to actually implement services offered by this system, both are already been defined, implemented and documented, in fact this document doesn’t specify them in details but we assume it can be easily integrated in the system using the already existing documentation.

Client-Server architecture This is tightly coupled to both 3-tier and SOA styles described previously. This is one of the most used and common style.

2.9 Design patterns

MVC The Model View Controller is an architectural pattern that aims to clearly separate data (Model), the logic which manipulates data (Controller) and the way this data is presented to the user (View). As previously described this is the way our system is divided, so this can be considered as the key pattern among all the others described here. Main advantages of this pattern are: to have a clearly distinctions between modules included in the application, to increase flexibility, decoupling and reusability of each component, for instance one can replace the View while preserving the Controller and the Model. One possible drawback is the increased complexity of the overall system, especially at the design part since care must be taken when deciding which part of the system goes where.

Façade This pattern allows to users to communicate with applicative logic in an easy way, in fact they don’t have to work with various components, but only with the façade, it is up to this last one to communicate with other parts of the system.

Singleton This pattern aims to guarantee that one and only one instance of a determined object is alive in the system at runtime. For example it is used when a new registered user is created, in fact this pattern forbids to duplicate users already present in database.

State pattern This pattern can change an object behavior during runtime. Here it is used to change ride states when a request or a reservation wait to be accepted, are working progress or are terminated.

2.10 Other design decisions

As already discussed in the RASD and showed in this document, we decided that the overall logic of the service will be implemented in Java using the JEE technology. This brings the main advantage to use a single programming language and a set of libraries on the server side that are able to deal with View, Controller and also Model, providing some techniques that are capable of abstracting the database, using Object Oriented Programming Paradigms.

The View will be implemented using the Java Server Faces technology exposing methods in the GUI Logic presented in previous sections. The Controller will work tightly coupled with the rest of components again using Java and JEE, thus enabling a simpler and a coherent way to establish a communication between each tier.

Moreover, developers can use an IDE to implement actual code and to enhance features of the IDE itself, such as automated code inspection and testing (using frameworks like JUnit), to allow a more stable and bug-free code since the very first release.

3 Algorithm Design

3.1 Find a queue knowing the position

```
1 Function findQueue(loc: Location)
  Data: loc, a location on the map that can (possibly) belong to a Queue
  Result: an instance of the object Queue if the location is valid, null otherwise
2  foreach q in Map.queues() do
3    if q.containsLocation(loc) then
4      return q;
5    end
6  end
7  return null;
8 end
```

When a request or a reservation is processed, this algorithm finds the nearest queue. The function uses a location parameter passed during invocation, then it searches in which queue the location stays, returning the instance of that queue.

3.2 Add a taxi to a queue

```
1 Function addTaxiToQueue(t: TaxiDriver, loc: Location)
  Data: t, a Taxi Driver that is going to be added to the appropriate queue,
         loc: the Location of the taxi driver being processed
  Result: true if the operation successfully completes,
         false otherwise
2  q ← findQueue(loc);
3  if q ≠ null then
4    q.enqueue(t);
5    return true;
6  else
7    return false;
8  end
9
10 end
```

This function adds a taxi driver to a queue. The location parameter passed during invocation is used as a parameter for findQueue function, returning a value that is checked. If it is different from null, if so the taxi driver passed as a parameter is enqueued to the queue previously returned and the addTaxiToQueue function returns true. Any type of problem makes the function return false.

3.3 Assign a taxi to an incoming request

```
1 Function assignTaxiToRequest(loc: Location, r: Request)
  Data: loc: the Passenger requested location for the meeting point,
         r: the actual request ready to be fulfilled
  Result: true if the operation successfully completes,
           false otherwise

2  q ← findQueue(l);
3  if q ≠ null then
4    t ← q.tail();
5    if t = null then
6      adjqueues ← q.adjacents();
7      q ← adjqueues.first();
8      while q ≠ null ∧ t = null do
9        t ← q.tail();
10       q ← adjqueues.next();
11     end
12     if t = null then
13       return false;
14     end
15   end
16   if t.forwardRequest(loc, r) = true then
17     t.queue.remove(t);
18     t.queue ← null;
19     r.assignDriver(t);
20     return true;
21   else
22     return false;
23   end
24 else
25   return false;
26 end
27
28 end
```

Location and request are the two parameters passed to this function. The first one is used to find the queue where there is the location instance. Then the algorithm controls if the queue is empty, if so, a adjacent queue is searched. If all queues are empty or there aren't taxi drivers in any of them, the function returns false, otherwise the request is sent to a taxi driver, if he/she is available, the taxi driver is removed from his/her queue and the request is assigned to him/her, returning true, in other cases the function returns false.

4 User Interface Design

4.1 Homepage

In both mobile application and browser, from homepage it's possible to move to all functions: login, registration, request and reservation (these two only for users registered as passengers) and Manage work (only for users registered as taxi drivers).

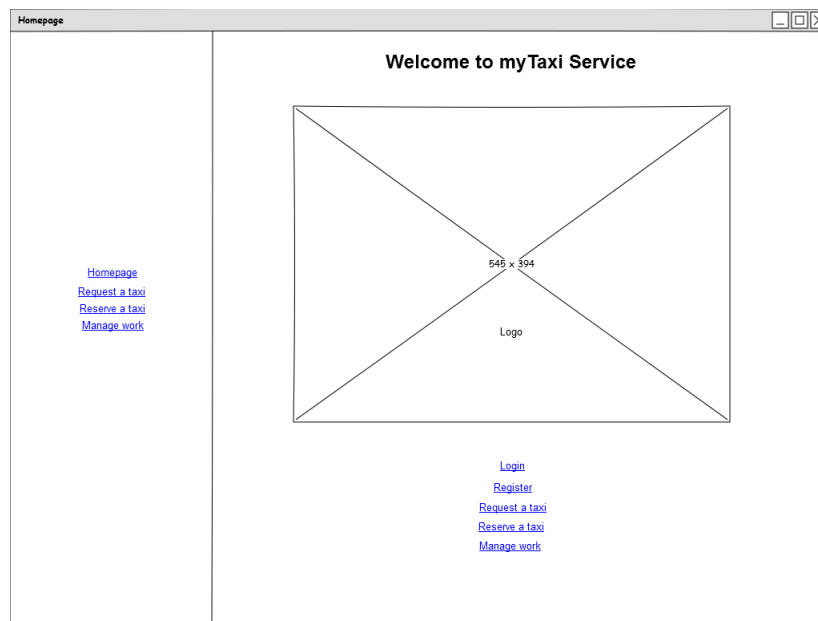


Figure 2: Homepage Web Version

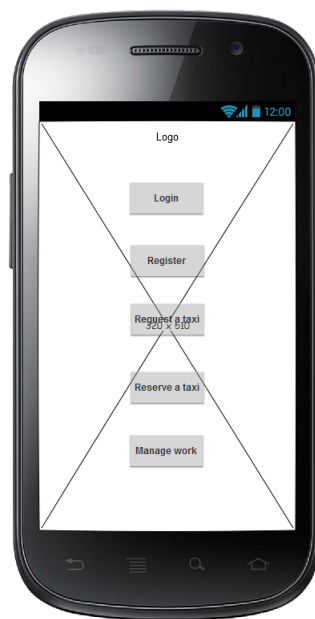


Figure 3: Homepage Mobile Version

4.2 Registration

Users can register themselves from this page filling various fields. The majority of these last ones are in common for every person who wants to register, but there is a field reserved to taxi drivers (taxi driver license number) that can be filled only choosing taxi driver, in the web registration page menu, or taxi driver radio in the mobile registration page.

Before confirming the registration, it's necessary to accept policies about privacy and use of user's personal details.

From this browser page users can go to homepage and other important pages.



The image shows a web browser window titled "Registration page". The main content area is divided into two columns. The left column contains a vertical list of links: "Homepage", "Reserve a taxi", "Reserve a taxi for another moment", and "Manage work". The right column contains a registration form with the following fields: "Name", "Surname", "E-mail", "Repeat e-mail", "Password", "Repeat Password", "Phone number", "Passenger or taxi driver?" (a dropdown menu), and "Taxi driver license number". Below these fields is a checkbox labeled "I agree to all policies" and a "Register" button. At the top of the form area, there is a placeholder for a logo, indicated by a box with the text "1000 x 100 Logo".

Figure 4: Registration Web Version

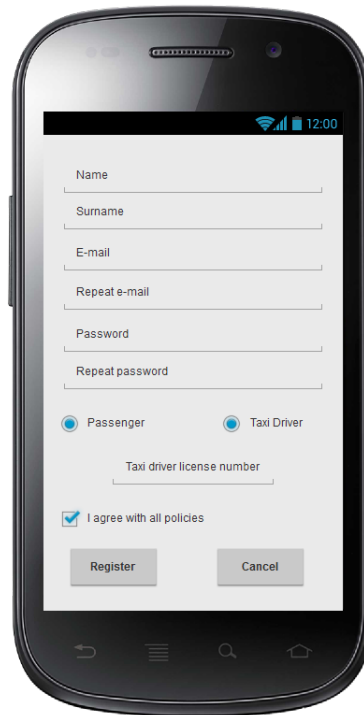


Figure 5: Registration Mobile Version

4.3 Login

From this page registered users can log in inserting either the registration email or username and the password chosen during registration.

If an user forgets his/her password, username or email, it's possible to recover them: a message on the mobile phone or an email is sent to him/her.

Also from this browser page users can go to homepage and other important pages.

The screenshot shows a web browser window titled "Login page". The browser's address bar is empty. The page layout includes a header section with a placeholder for a "1000 x 100 Logo". Below the header, the page is divided into two main columns. The left column contains a vertical list of links: "Homepage", "Request a taxi", "Reserve a taxi", and "Manage work". The right column contains the login form, which includes an "E-mail" input field, a "Password" input field, a link for "Username or password forgotten", a checkbox labeled "Stay logged in on this computer", and a "Login" button.

Figure 6: Login Web Version

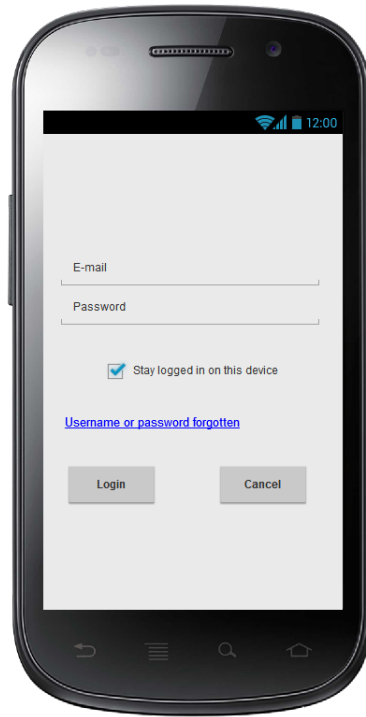


Figure 7: Login Mobile Version

4.4 Recovery Password

In the web version, registered users can recover their username, password and registration e-mail inserting following information. A SMS will send to the user who uses the recovery procedure containing all information necessary to log in.

On the contrary in mobile version, the recovery SMS is sent to the smartphone where the application is installed.

For the administrator it's not possible to use recovery procedure.

From this browser page users can go to homepage and other important pages.

The screenshot shows a web browser window titled "Recovery login page". The page layout includes a header area with a placeholder for a "1000 x 100 Logo". On the left side, there is a vertical menu with four blue hyperlinks: "Homepage", "Request a taxi", "Reserve a taxi", and "Manage work". The main content area contains four text input fields arranged in a 2x2 grid: "Name", "Surname", "Registration e-mail", and "Registration phone number". Below these fields is a grey "Confirm" button.

Figure 8: Recovery Web Version

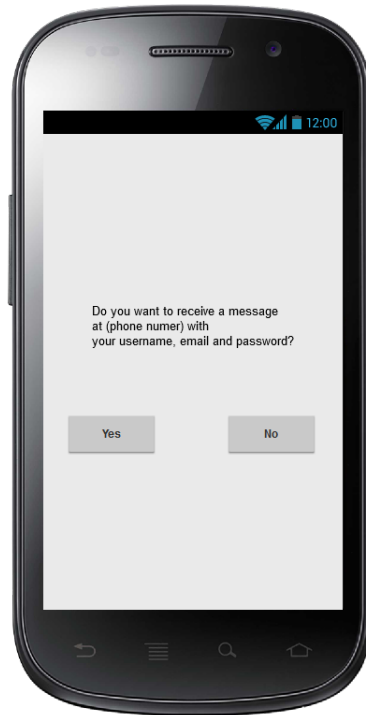


Figure 9: Recovery Mobile Version

4.5 Request

Both web page and mobile application page show how to request a taxi: in the top field registered passengers have to insert the location where they want to be picked up. This is optional, in fact registered passengers can decide to be geolocalized to be picked up where they are. The map below shows them where the inserted place is or, if they haven't inserted anything, where they are. On the bottom of the page there is another field where registered passengers have to insert the number of passengers who will take that taxi. After completing all these operations, it is necessary to push "Confirm" (for browser) or "Reserve" (for mobile application) to send the request to reserve a taxi. As usual, on the left part of the web page there are hyperlinks to reach the most important pages.

The screenshot shows a web browser window titled "Reservation page". The page layout includes a header bar with a "Logo" placeholder (1000 x 100). On the left side, there is a vertical menu with four blue hyperlinks: "Homepage", "Request a taxi", "Reserve a taxi", and "Manage work". The main content area on the right starts with a "Welcome (name and surname)" greeting. Below this is an "Address" section with a text input field labeled "Departure address" and an "Enter" button. Underneath the address field is a map placeholder (470 x 328) with the text "Map with actual position or the one inserted below". At the bottom of the main area is a "Number of people:" label followed by a text input field labeled "Number". A "Confirm" button is positioned at the very bottom center of the page.

Figure 10: Request Web Version



Figure 11: Request Mobile Version

4.6 Reservation

As before, also for reservation, registered passengers have to be geolocalized or they have to insert the location where they want to be picked up and it is mandatory to select the number of passengers. Plus, to make a reservation, it's necessary to select hour and date when passengers want to take a taxi.

Also from this browser page users can go to homepage and other important pages.

The screenshot shows a web browser window with the title "Reservation in a different moment page". The browser's address bar is empty. The page layout includes a sidebar on the left with the following links: [Homepage](#), [Request a taxi](#), [Reserve a taxi](#), and [Manage work](#). The main content area has a header with a "Welcome (name and surname)" message. Below this is a "Departure address" input field. A large map placeholder is shown with a diagonal cross and the text "470 x 328" and "Departure position". Below the map is a "Number of people:" label followed by a "Number" input field. At the bottom, there are date and time selection fields: "hh" and "mm" for time, and "dd", "mm", and "yyyy" for date, each with a dropdown arrow. A "Reserve" button is located at the bottom center.

Figure 12: Reservation Web Version



Figure 13: Reservation Mobile Version

4.7 Driver Interface

This page is accessible only to registered taxi drivers, here they can manage incoming requests and reservation. Using “Previous” and “Next” buttons, taxi drivers can select available requests and reservations, accepting or refusing them clicking on specific buttons. For every request and reservation, there is a map that shows to taxi drivers where passengers want to be picked up.

As usual, on the left part of the web page there are hyperlinks to reach the most important pages.

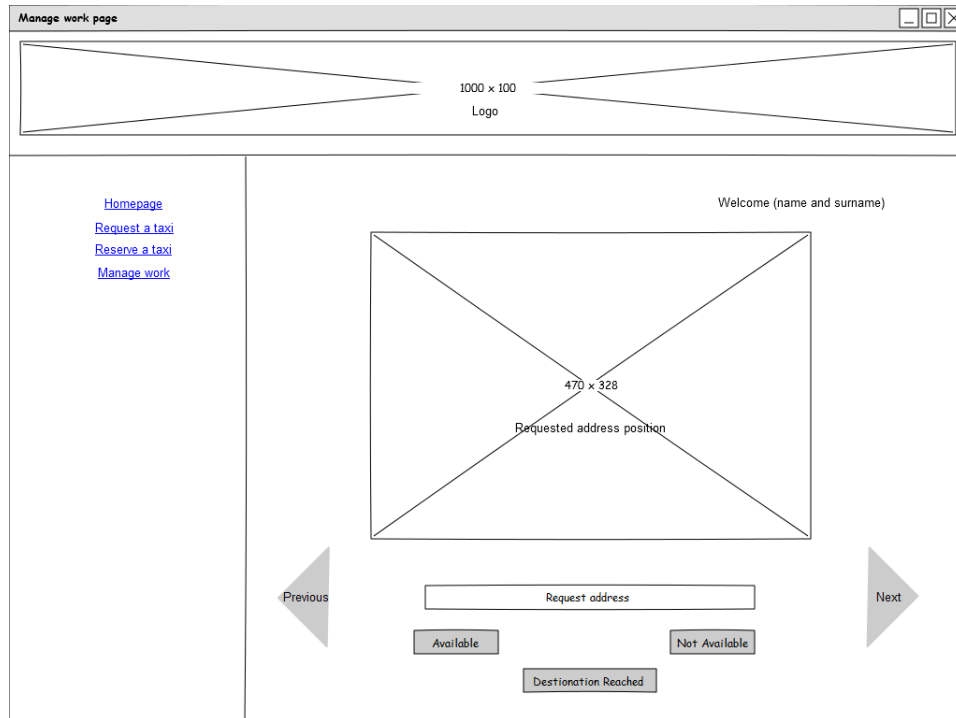


Figure 14: Driver Interface Web Version

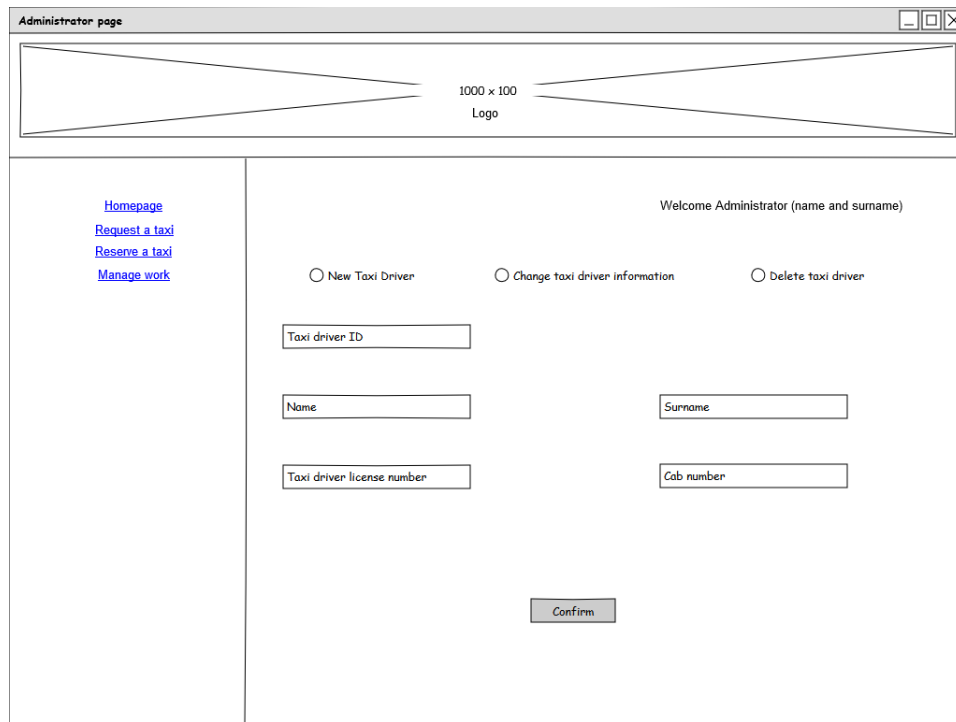


Figure 15: Driver Interface Mobile Version

4.8 Administrator

This page is accessible only to the administrator. He/she has to validate, modify, delete taxi driver profiles in case drivers respectively register themselves, change information about them or resign.

Hyperlinks work exactly like in other cases, but to use web and mobile functions it's necessary to log in as passengers or taxi driver. This page is accessible only inserting its URL, it's not possible to reach it through web site.



The screenshot shows a web browser window titled "Administrator page". The main content area is divided into two sections. The top section is a header bar with a placeholder for a "1000 x 100 Logo". Below the header, the main content area is split into a left sidebar and a right main panel. The sidebar contains four blue hyperlinks: "Homepage", "Request a taxi", "Reserve a taxi", and "Manage work". The main panel displays a "Welcome Administrator (name and surname)" message. Below the welcome message, there are three radio buttons for selecting an action: "New Taxi Driver", "Change taxi driver information", and "Delete taxi driver". Under the "Change taxi driver information" option, there are four text input fields: "Taxi driver ID", "Name", "Surname", "Taxi driver license number", and "Cab number". A "Confirm" button is located at the bottom center of the main panel.

Administrator page

1000 x 100
Logo

[Homepage](#)
[Request a taxi](#)
[Reserve a taxi](#)
[Manage work](#)

Welcome Administrator (name and surname)

☐ New Taxi Driver ☐ Change taxi driver information ☐ Delete taxi driver

Taxi driver ID

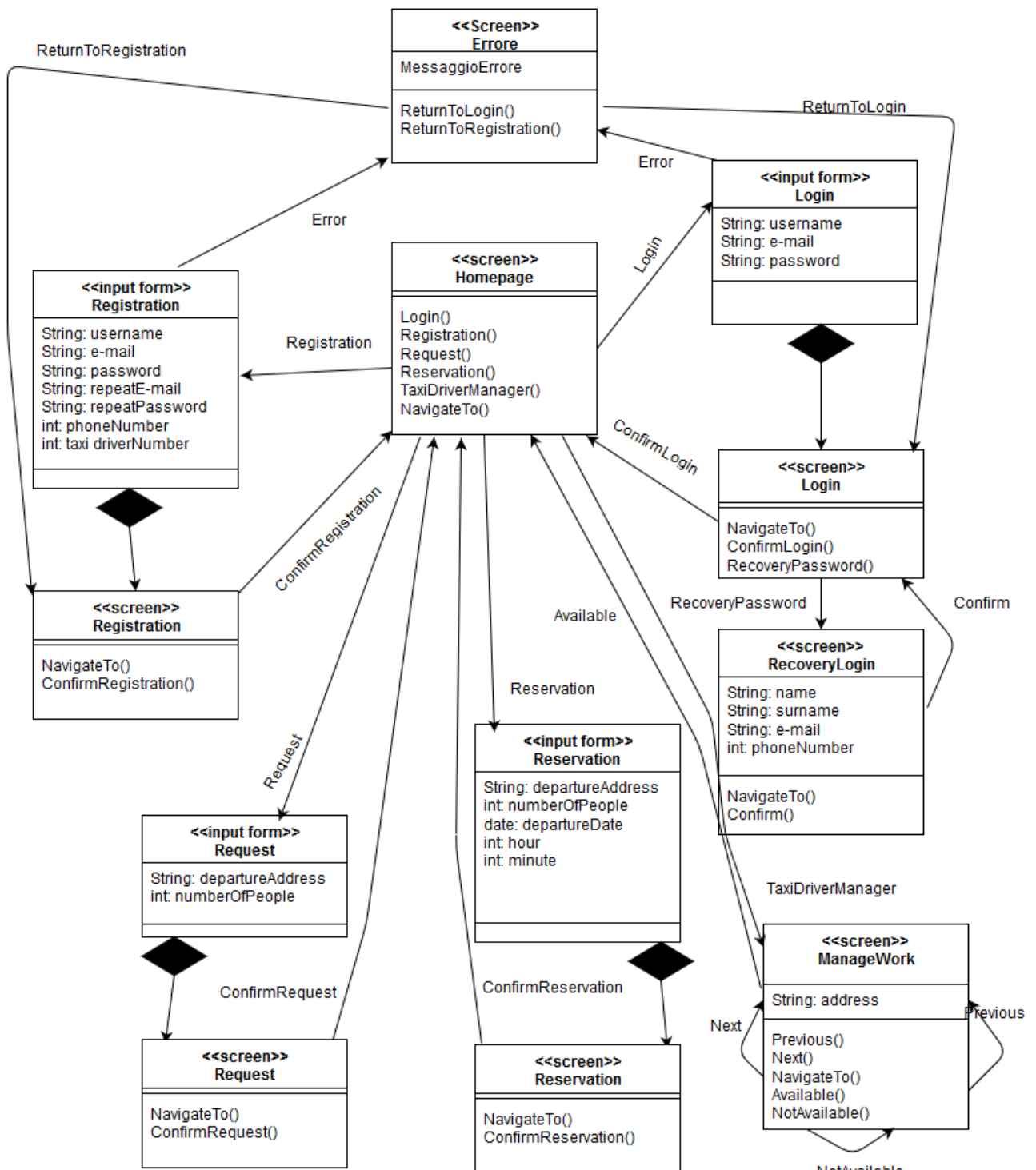
Name Surname

Taxi driver license number Cab number

Confirm

Figure 16: Administrator Web Page

4.9 UX Diagram



This diagram represents paths a user can go through. As it is shown, after a operation is successfully concluded, users are directed to homepage. Furthermore there a screen in case of mistaken information inserted in login and registration pages.

Administrator page doesn't appear because it's not a normal page accessible to people, but it can be used only by one authorized person.

5 Requirements Traceability

The following table shows how it is the mapping between the goals and their requirements with the design decisions described and analyzed in this document.

Requirements	Design Decisions
<ul style="list-style-type: none">• [G1] Allow visitors to sign up for the service<ul style="list-style-type: none">– [R1] The system requests an e-mail or a telephone number during the registration– [R2] The system rejects an e-mail or a telephone number already used by another registered passenger– [R3] After the completion of registration process, the system must send a confirmation to newly registered passengers– [R4] The system checks if e-mails or telephone numbers are well formed– [R5] The system considers passengers officially registered if and only if passengers either open an activation link in a confirmation e-mail, or insert a code received through a SMS in mobile application or in web service.	<ul style="list-style-type: none">• Access control component and interface• Passenger UI Logic component and interface• myTaxiService Database component and interface

<ul style="list-style-type: none"> • [G2] Allow registered passengers to request a taxi <ul style="list-style-type: none"> – [R1] The system must disallow visitors to perform such operation – [R2] The system has to locate passengers using GPS in his/her smartphone or through browser. – [R3] After selecting the most appropriate taxi (explained in goal [G4]), the system must inform drivers that a request is incoming. – [R4] Upon confirmation by the driver about the request of availability, the system has to inform the passenger about a positive outcome,if it occurs, and provide additional information, such as ETA – [R5] In case of unavailability of a contacted driver, the system has to pick another taxi from the most appropriate queue and to repeat what described in requirement [R3] – [R6] The system must proceed to the cancellation of the request if and only if the user requested to do it and the condition in [D1] has not met yet. • [G3] Allow registered passengers to reserve a taxi <ul style="list-style-type: none"> – [R1] The system must disallow visitors to perform such operation. – [R2] The system must check if time and the date specified is at least past two hours with the respect of present time. – [R3] The system must disallow the reservation of a taxi with an invalid meeting point. – [R4] The system must disallow the reservation of a taxi with a meeting point outside the area served by the service. – [R5] The system shall allocate a timer that will timeouts ten minutes before the time and date specified, triggering the event of allocating a taxi for this reservation. 	<ul style="list-style-type: none"> • Access control component and interface • Ride manager component and interface • Passenger UI Logic component and interface • Maps Service external component and interface
---	---

<ul style="list-style-type: none"> • [G4] Guarantee a fair management of the taxi queue <ul style="list-style-type: none"> – [R1] The system must select the first free taxi in the queue corresponding to the zone in which the meeting point is. – [R2] The system has to manage the queue using a predefined policy. – [R3] If there are no available taxis in a queue, the system scans queues of adjacent zones. – [R4] The system must remove a taxi from the queue if and only if a passenger has confirmed the intention of taking that taxi. – [R5] Queues contain only free taxis. – [R6] The system must not select taxis that belong to drivers who are spending their breaks. 	<ul style="list-style-type: none"> • Queue manager component and interface
<ul style="list-style-type: none"> • [G5] Allow taxi drivers to manage their workday <ul style="list-style-type: none"> – [R1] The system must disallow non-taxi drivers to perform such operations. – [R2] The system must not allow to start a workday twice or more in the same day. – [R3] The system must not allow to end a workday before a workday is started. – [R4] The system must enforce the driver to take some breaks during the day. 	<ul style="list-style-type: none"> • Access control component and interface • Taxi Driver UI Logic component and interface • Queue manager component and interface
<ul style="list-style-type: none"> • [G6] Allow the system administrator to update the information about drivers and taxis <ul style="list-style-type: none"> – [R1] The system must disallow non-system administrator to perform such operations. – [R2] The system must disallow the insertion of inconsistent data (for example drivers with invalid license or without a car). – [R3] The system must disallow the insertion of already present data. 	<ul style="list-style-type: none"> • Access control component and interface • myTaxiService Database component and interface • Cars and Driver external component and database

Tools Used

- Basic MiKTeX 2.9.5721 64-bit - <http://miktex.org/>
- Texmaker 4.5 - <http://www.xmlmath.net/texmaker/index.html>
- draw.io - <https://www.draw.io/>
- Pencil 2.0.5 - <http://pencil.evolus.vn>

Hours spent to write this document

- Raffaele Fioratto: 40 hours
- Nicolò Longoni: 37 hours