Politecnico di Milano
Academic Year 2015/2016
Software Engineering 2: "myTaxiService"
Design Document

Massimo Schiavo, Marco Edoardo Cittar

December 4, 2015

Contents

1 Introduction **4**
- 1.1 Purpose . . . 4
- 1.2 Scope . . . 4
- 1.3 Acronyms, abbreviations . . . 4
  - 1.3.1 Acronyms . . . 4
  - 1.3.2 Abbreviations . . . 4
- 1.4 Reference documents . . . 4
- 1.5 Document structure . . . 4

2 Architectural design **6**
- 2.1 Overview . . . 6
- 2.2 High level components and their interaction . . . 6
- 2.3 Component view . . . 8
- 2.4 Deployment view . . . 9
- 2.5 Runtime view . . . 10
  - 2.5.1 Registration . . . 10
  - 2.5.2 Log in . . . 11
  - 2.5.3 Queue management . . . 12
  - 2.5.4 Simple request . . . 13
  - 2.5.5 Detailed request . . . 14
  - 2.5.6 View free taxis . . . 15
  - 2.5.7 Be available . . . 16
- 2.6 Component interfaces . . . 17
- 2.7 Selected architectural styles and patterns . . . 18
- 2.8 Other design decisions . . . 21

3 Algorithm design **23**
- 3.1 Registration . . . 23
- 3.2 Log in . . . 23
- 3.3 Simple request . . . 23
- 3.4 Detailed request . . . 23
- 3.5 View free taxis . . . 24
- 3.6 Be available . . . 24

4 User interface design **25**
- 4.1 Log in page . . . 25
- 4.2 Registration page . . . 26
- 4.3 Confirmation page . . . 27
- 4.4 User page . . . 28
- 4.5 Simple Request . . . 29
- 4.6 Detailed Request . . . 30
- 4.7 Request confirmation page . . . 31
- 4.8 Available taxis page . . . 32
- 4.9 User notifications page . . . 33

4.10 Driver page 34
4.11 Requests page 35

5 Requirements traceability **36**
5.1 Guests requirements 36
5.2 User requirements 36
5.3 Driver requirements 36
5.4 System requirements 37

6 References **37**
6.1 Software and tools used 37
6.2 Hours of work 37

1 Introduction

1.1 Purpose

This software design specification is made with the purpose of outlining the software architecture and design of myTaxiService application system in detail. The documents will provide developers an insight in meeting client's needs efficiently and effectively. Moreover the document simplifies communication and understanding of the system by providing several views of the system design.

1.2 Scope

The software design document would demonstrate how the design will accomplish the functional and non-functional requirements defined in the RASD. The document will provide a framework to the programmers by describing the high level components and architecture, sub-systems, interfaces and algorithm design. This is achieved through the use of architectural patterns, sequence diagrams and user interfaces.

1.3 Acronyms, abbreviations

1.3.1 Acronyms

- DD: Design Document
- RASD: Requirements Analysis and Specification Document
- RDBMS: Relational Database Management System
- SQL: Structured Query Language

1.3.2 Abbreviations

- myTaxi: it obviously stands for the application name "myTaxiService"

1.4 Reference documents

- RASD of the application (RASD_myTaxiService_Schiavo_Cittar.pdf)
- The assignment document (Assignments 1 and 2 (RASD and DD).pdf)
- Reference to the structure of the DD (Structure of the design document.pdf)

1.5 Document structure

The document is structured in five parts:

- Section 1: Introduction, briefly describes what the document is about.

- Section 2: Architectural design, describes the architecture the application should have and the choices we made about styles and patterns. In particular, it contains the components and how they interact between them, sequence diagrams that show how the application works and description of the components interfaces.
- Section 3: Algorithm design, contains the explanation of how the most important algorithms work.
- Section 4: User interface design, contains mockups of how the application should look like.
- Section 5: Requirements traceability, shows how the requirements we defined in the RASD are fulfilled with the design described in the document.

2 Architectural design

2.1 Overview

The next chapter of the document describes architectural design of myTaxi system: the high level components and their interactions, suitable architectural patterns, physical arrangement of components and design decisions applied to the whole system.

2.2 High level components and their interaction

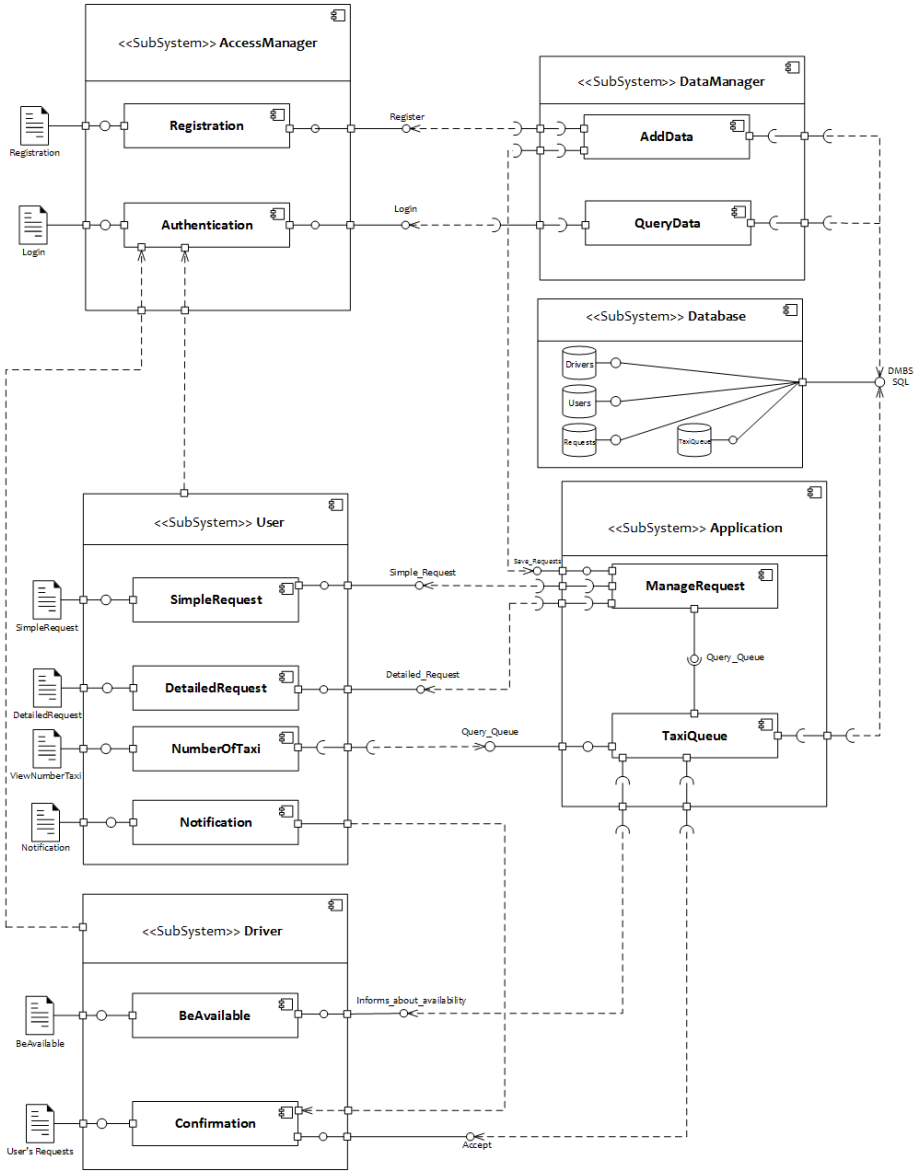
- AccessManager: This sub-system consists of all the components responsible of registration and login process. It authenticates users and drivers, allowing them to use their relative functionalities. It's also responsible for guest's registration.
  - Registration
  - Authentication
- User: This sub-system consists of all the components that implement the functions related to the user's operations such as requesting a taxi for a simple ride, making a reservation for a taxi and viewing the number of available taxis in his zone.
  - SimpleRequest
  - DetailedRequest
  - NumberOfTaxi
  - Notification
- Driver: This sub-system consists of all the components that implement the functions related to the driver's operations such as informing about their own availability and accepting an user's request.
  - BeAvailable
  - Confirmation
- DataManager: This is the major sub-system that is responsible for the security of sensible data. It communicates directly with the database providing several operations on it such as adding new users, adding new drivers, checking saved data with data given by the user during the login process.
  - AddData
  - QueryData
- Database: Essentially, it is formed by the databases and give all the functionalities through which operate directly on the database.

  - Drivers
  - Users
  - Requests
  - TaxiQueue

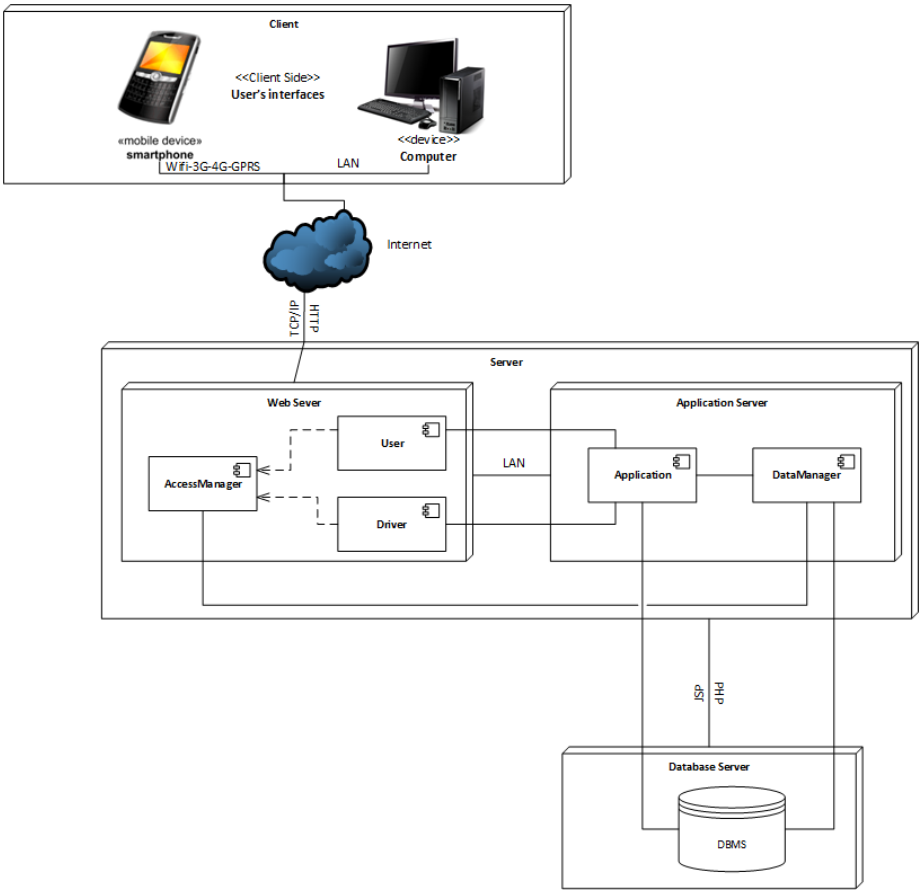
- Application: This is the major sub-system that is responsible of the logic of myTaxi app. In particular it manages all the requests done by the users forwarding them to the drivers according to the policy of forwarding.

  - ManageRequest
  - TaxiQueue

2.3 Component view

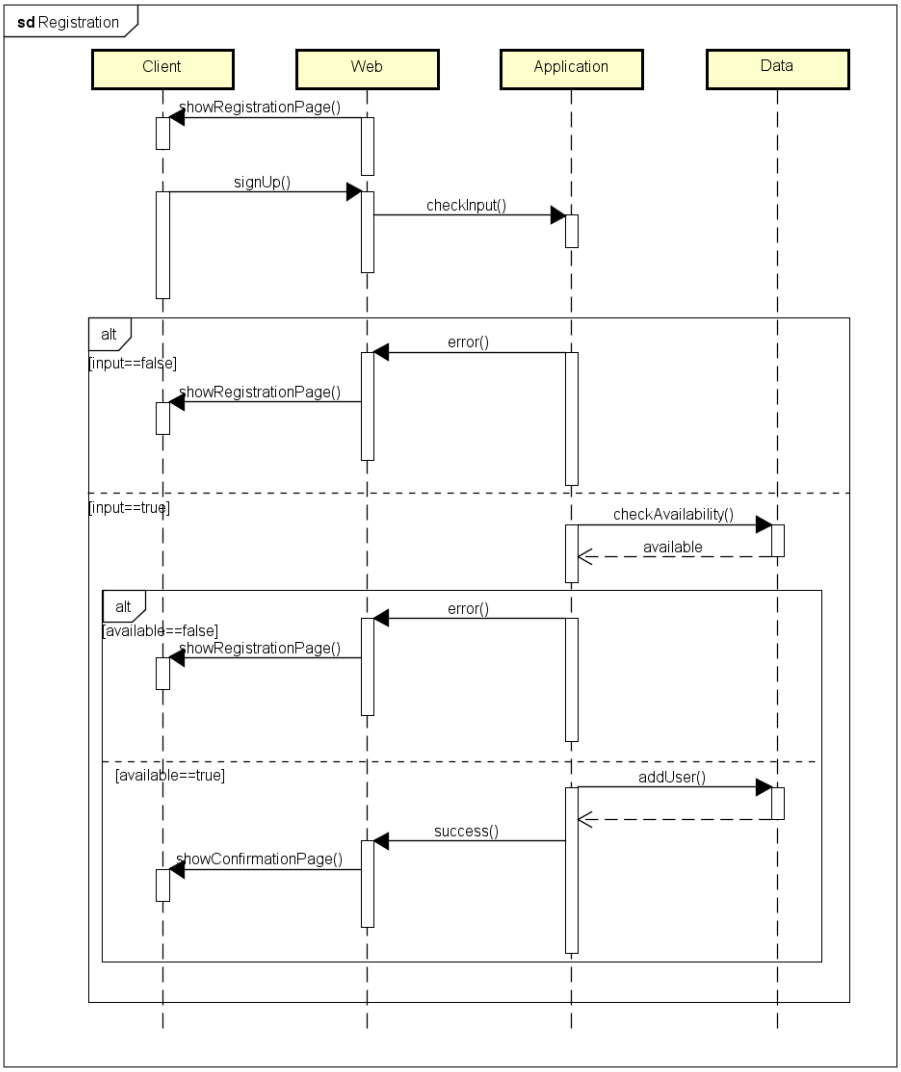


2.4 Deployment view

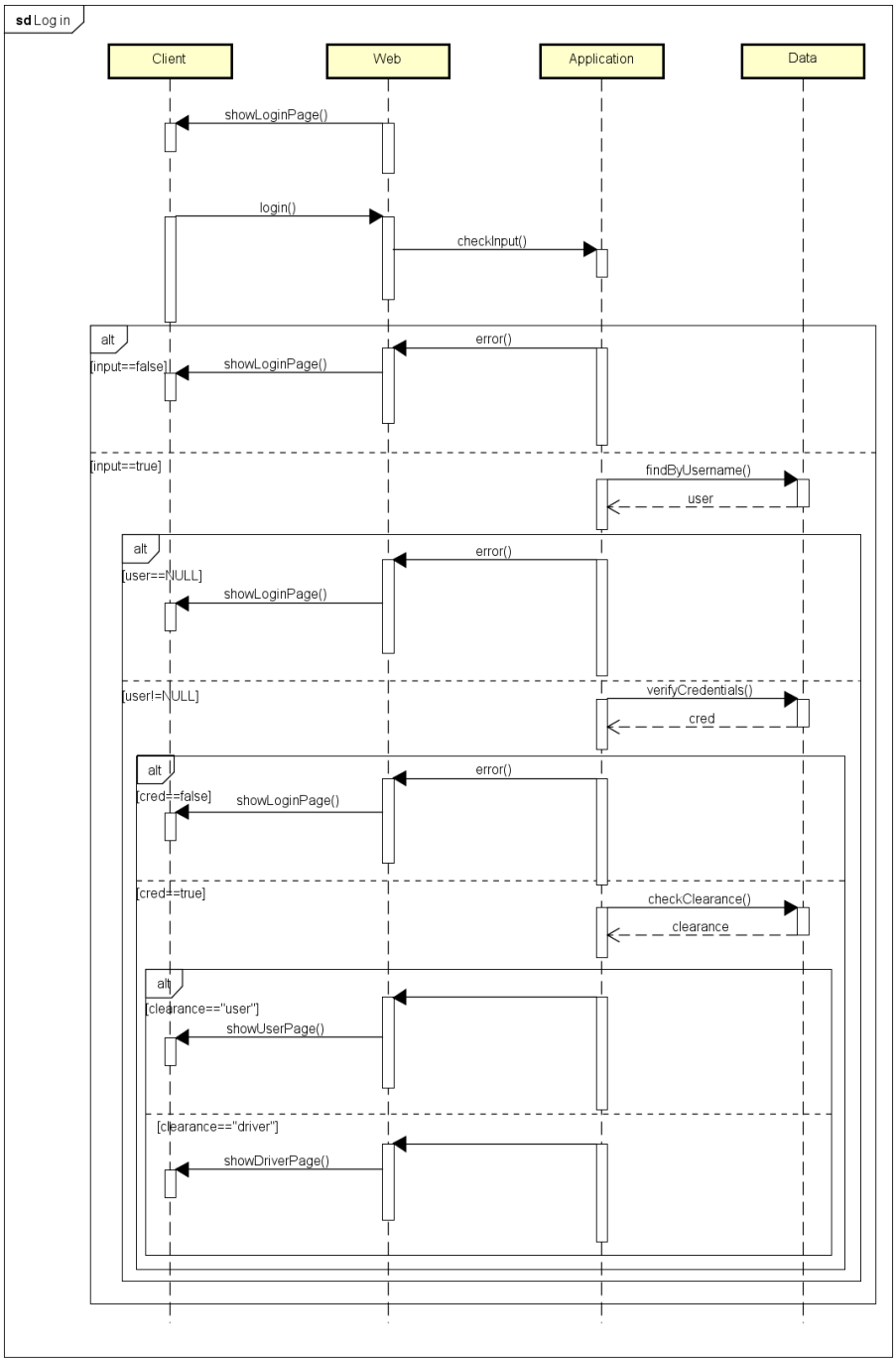


2.5 Runtime view

2.5.1 Registration

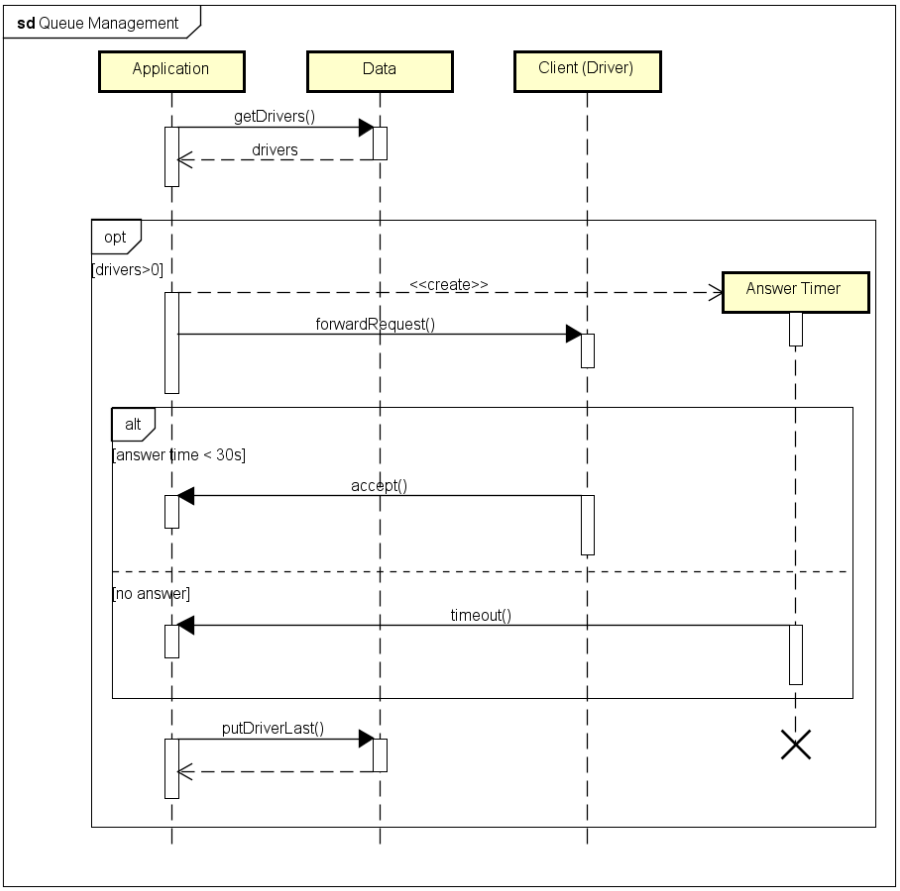


2.5.2 Log in

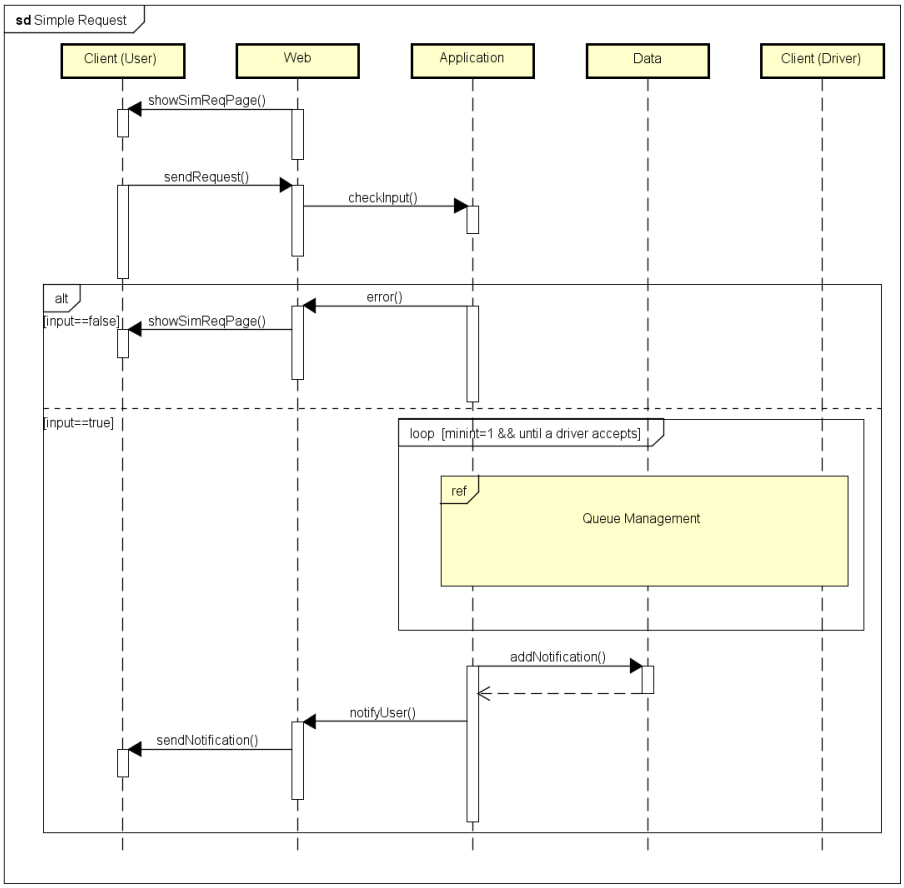

sd Log in
Client
Web
Application
Data
showLoginPage()
login()
checkInput()
alt
[input==false]
showLoginPage()
error()
[input==true]
findByUsername()
user
alt
[user==NULL]
showLoginPage()
error()
[user!=NULL]
verifyCredentials()
cred
alt
[cred==false]
showLoginPage()
error()
[cred==true]
checkClearance()
clearance
alt
[clearance=="user"]
showUserPage()
[clearance=="driver"]
showDriverPage()

2.5.3 Queue management

This sequence diagram represents how the application manage to forward a request to a driver.

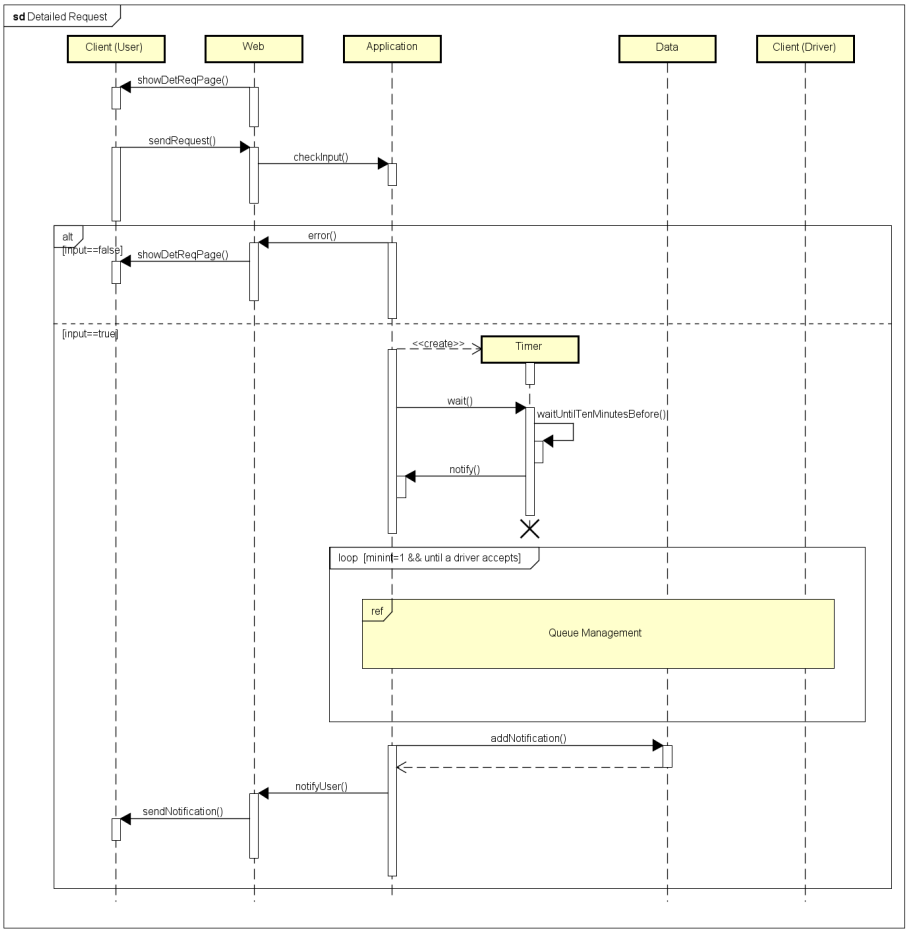


2.5.4 Simple request

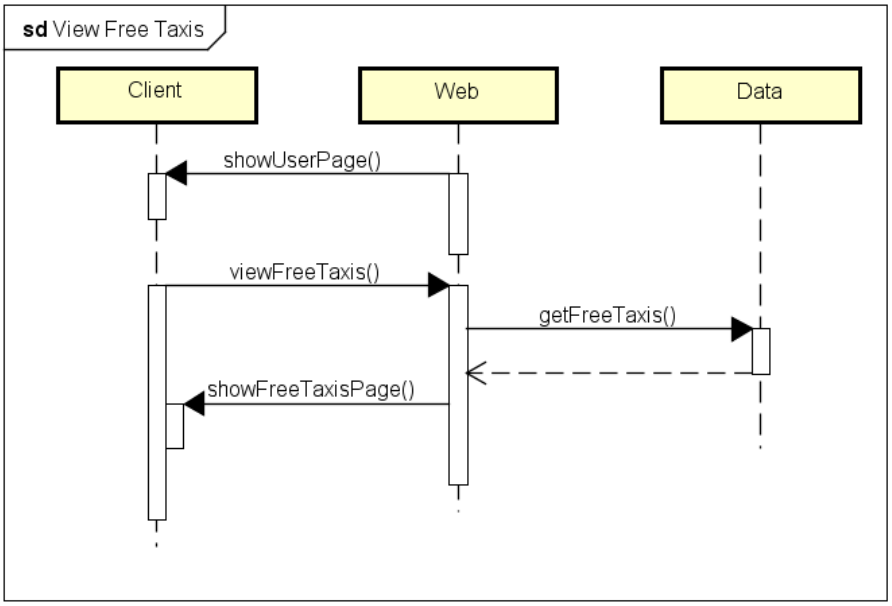


2.5.5 Detailed request

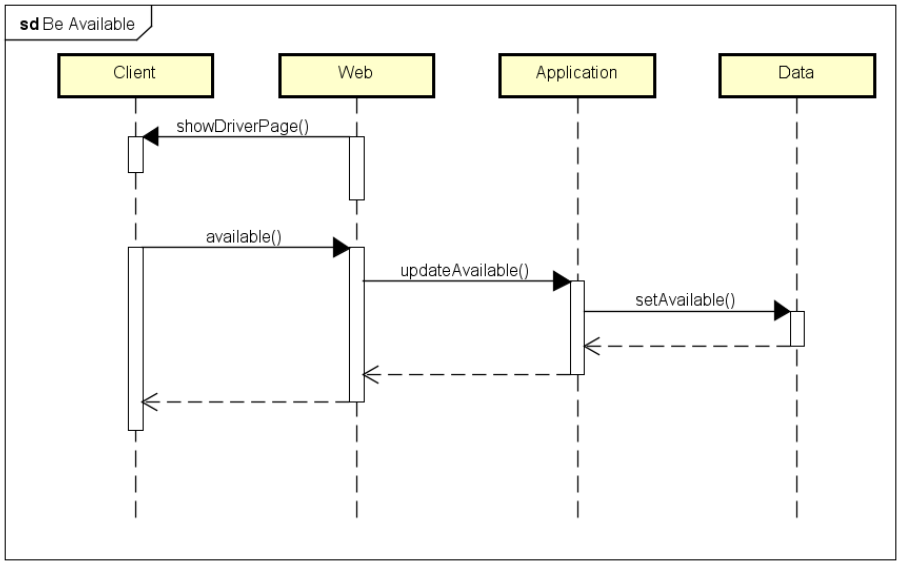
The wait and notify methods belong to the thread class, meaning that when the wait() method is called the thread goes to sleep until the timer wakes it up with the notify() method 10 minutes before the time of the reserved ride.



2.5.6 View free taxis



2.5.7 Be available



2.6 Component interfaces

- AccessManager
  - Registration:Register: This interface allows a guest to register himself into the system by giving the mandatory information.
  - Authentication:Login: This is the interface that allows the users/drivers to log into the system. This will guide the user/driver to his relative home page.
- DataManager
  - AddData: This component uses interfaces provided by other components in order to grant a major grade of security to the data tier interposing between the presentation tier and the data one.
  - QueryData: This component, like AddData, communicates with the data tier querying it in order to check user's information given by the user interface.
- Database
  - The relevant interfaces given by this tier are the most common known methods through which it's possible to operate, manipulate and querying a database.
- User
  - SimpleRequest:Simple_Request: This is the interface that allows the users to ask for a taxi for a simple ride.
  - DetailedRequest:Detailed_Request: This is the interface that allows the users to reserve a taxi.
  - NumberOfTaxi: This component uses a required interface provided by the TaxiQueue in order to query it and know the number of available taxi in relation to the current position given by the GPS.
  - Notification: This is a component that only has a user's interface that depends on the confirmation of the user's own request.
- Driver
  - BeAvailable:Informs_About_Availability: This is the interface that allows a driver to inform the system about his current availability. It can be changed by both system and driver according to what is written in the RASD.
  - Confirmation:Accept: This is the interface through which a driver can accept an incoming user's request forwarded by the system.
- Application

- ManageRequest:Save_Request: This interface is given to the manager data component in order to retrieve information about requests and save them in the database.
- ManageRequest:Taxi_Queue: This interface is the major central point of the application because it communicates with the other components whose tasks are to handle requests incoming from the users and forward them correctly to the drivers.
- TaxiQueue:Query_Queue: This interface only allows to query the number of available taxis.
- TaxiQueue: This component, with the required interfaces it needs, is the central point of our application because, as written before, it is responsible of the correct management of requests, users and drivers on whose the service provided by our application is based.

2.7 Selected architectural styles and patterns

myTaxi System will be developed under two main architectural styles/patterns. Development of the project will be done in MVC architectural style and also 3 tier Client/Server Architecture. Client can browse the internet and access myTaxi application.

- MVC Architecture Style (Model – View – Controller)

  MVC separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other.

  1. The Model component that manages the system data and associated operations on that data.
  2. The View component that defines and manages how the data is presented to the user.
  3. The Controller component that manages user interaction and passes these interactions to the View and the Model.

  We will use this MVC Style for myTaxi system because there are multiple ways to view and interact with the data. We have decided to use MVC architectural style to separate the application logic with the interface. The main advantage of this style is that it allows the data to change independently of its representation and vice versa.

- Three-Tier Client Server Architecture

  In a client-server architecture, the functionality of the system is organized into services, with each service delivered from a separate server. Clients are users of these services and access servers to make use of them. We will use this 3-Tier Client Server Architecture, because data in a shared database has to be accessed from a range of locations, in this way servers can be replicated. In particular, there are three tiers:

1. Data Tier

   The data tier maintains the applications data such as Users' data, Drivers' data, Request data, TaxiQueue data and the SQL queries. It stores these data in a relational database management system (RDBMS). All the connections with the RDBMS are managed in this tier.

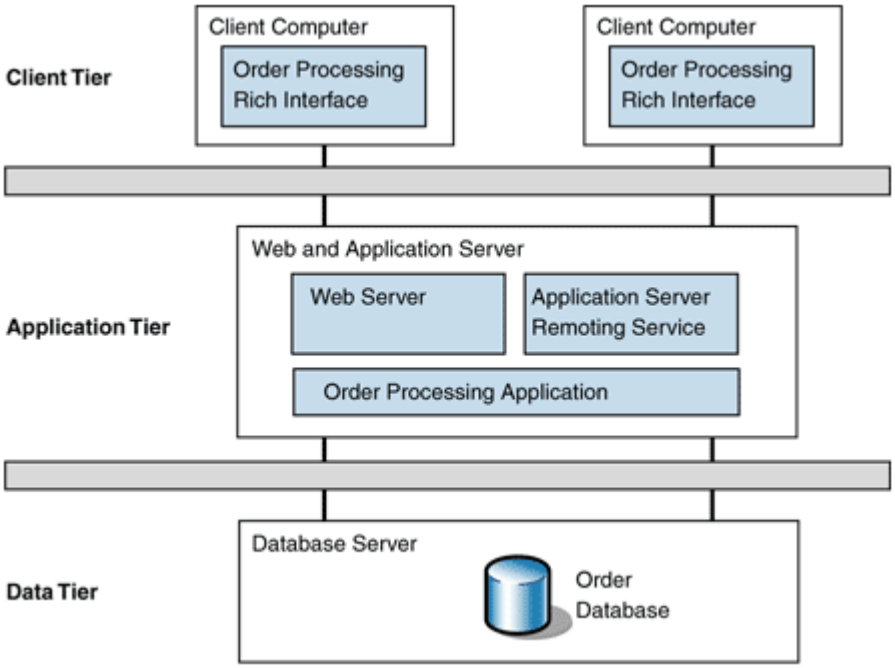
2. Application Tier

   The application tier (web/application server) implements the business logic, controller logic and presentation logic to control the interaction between the application's clients and data. Business rules enforced by the business logic dictate how clients can't access application data directly and how applications process data.

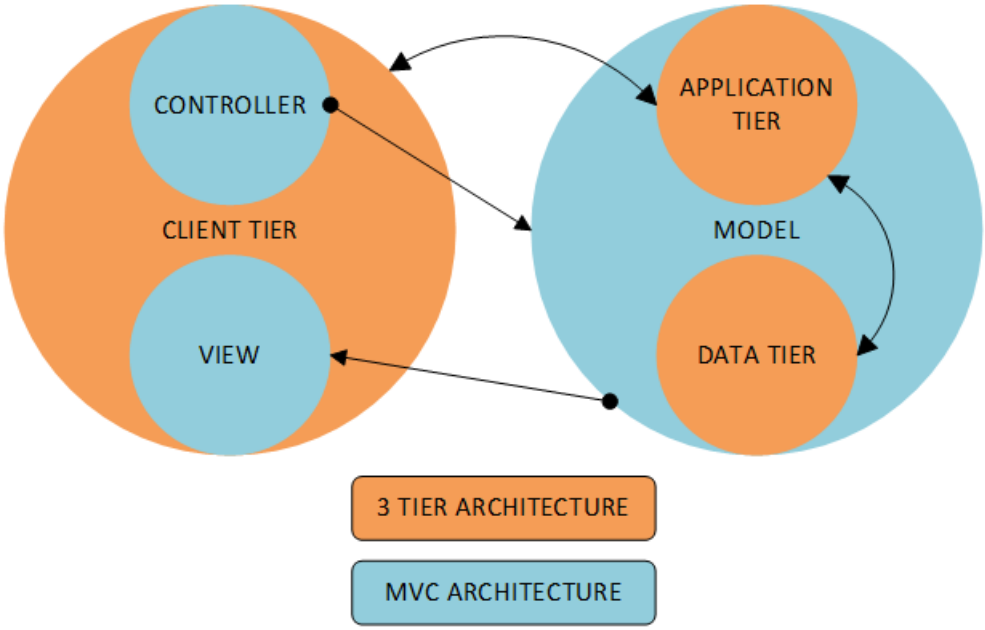
3. Client Tier

   The client tier is the application's user interface connecting data entry forms and client side applications. It displays data to the user. User interact directly with the application through user interface. The client tier interacts with the web/application server to make requests and to retrieve data from the database. It then displays to the user the data retrieved from the server.

In the picture below it's shown how the 3-tier architecture style works:



In the picture below it's shown how we have combined the two style/pattern. On the presentation tier we have the view and the controller that invokes methods on the model in which there are the two other tiers. In particular, a change in the model come from the controller to the application tier which then forwards it to the data tier.



2.8 Other design decisions

- Object oriented software development methods
  - Improved software maintainability
  - Faster development
  - Lower cost development
  - Improved software development productivity
  - Higher quality software
- Three-tier client server architecture
  - As more users access the system a three-tier solution is more scalable than the other solutions because you can add as many middle tiers as needed to ensure good performance.
  - Security is also the best in the three-tier architecture because the middle layer protects the database tier.
- MVC Architectural Pattern

- It should interact with other machines or users effectively.
- For more efficient interaction, the system should have flexible interfaces.
- MVC can be taken as for a popular and easy to handle web application development style that has the feature of separating the presentation and intermediate logics.
- It's easy to code and provides well defined interfaces within each logic.

3 Algorithm design

3.1 Registration

The algorithm takes in account the completeness, the correctness and the availability of the information inserted.

After the guest submits the information the application checks if all the mandatory fields are filled and if the "confirmation" fields, such as password and email, correspond to the respective previous ones.

If one of the above doesn't apply, the application shows again the registration page to the guest. If they do, then it looks up into the database to verify that the username and email inserted are not already in use.

If they don't, then the user is added to the database and is shown a confirmation page, inviting him to log in., otherwise the registration page is shown again.

3.2 Log in

After the guest inputs his credentials, the application first checks if both fields are filled, then looks for the username in the database to see if it exists. If it does, it checks if the password matches too.

If all of these steps are verified, the guest is logged in and is shown his personal page, otherwise the application shows the login page again.

After the password verification, the application will check the privileges of the account that is about to log in and will consequently show the respective page, be it user's or driver's.

3.3 Simple request

After the user clicks on the "Simple request" button, he is shown the simple request page.

After he inserts the origin of his ride and press the "Request" button, the application will then begin to allocate the first available taxi, otherwise if there's none it will just wait until one becomes available.

After forwarding the request to the taxi, the system will wait 30 seconds, if the driver answers the call he will be in charge for that request, or else the system forwards the request to the next available taxi in queue. In both cases the driver will be put at the end of the queue.

Once the taxi is allocated, the user is informed with a notification containing the code of the taxi and the waiting time.

3.4 Detailed request

After the user click on the "Detailed request" button, he is shown the detailed request page.

The user must fill the form and click on the "Request" button. The application will then check if all the fields are filled and if time and date are formally

correct. If that's so, the system will wait until 10 minutes before the ride and then begin to allocate a taxi for the request.

The following taxi allocation, queue management and notification works as the one in the simple request.

3.5 View free taxis

The users can look at the number of available taxis in their zone. By clicking on the "View available taxis" button, the application will query the database about it, and then return the result to a new page.

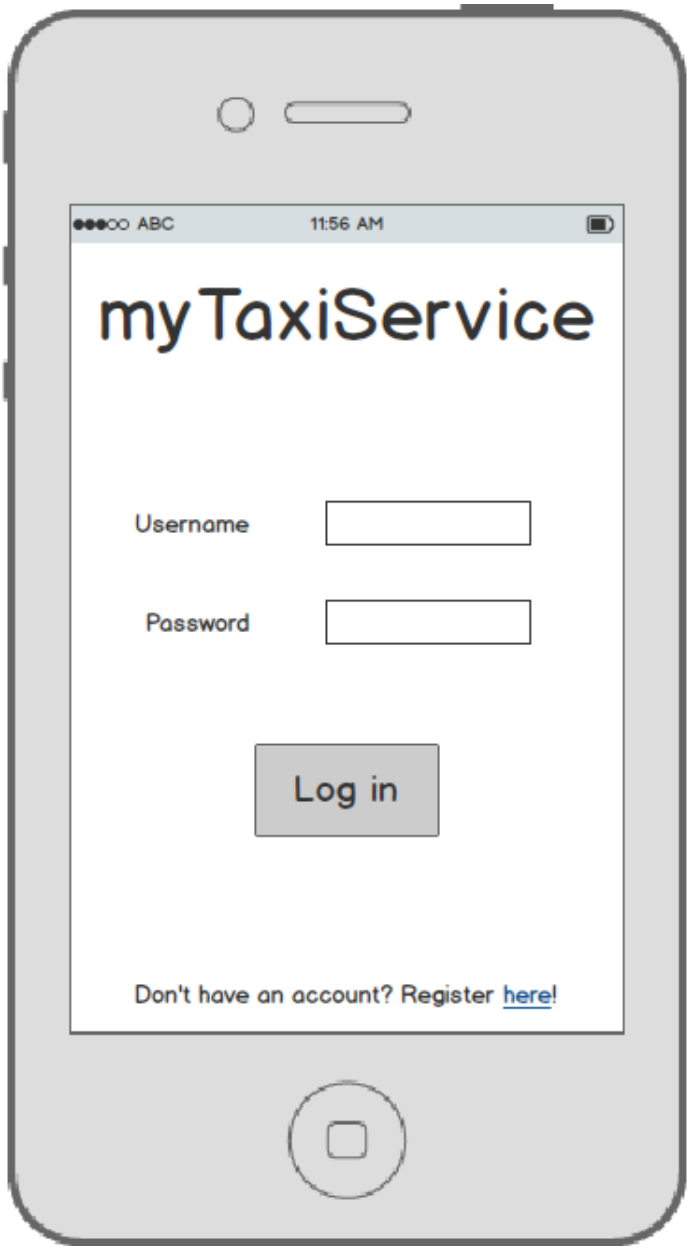
3.6 Be available

The drivers can use the switch on their personal page to toggle their own availability. After the button is switched, the application changes in the database the "available" field of the driver from "false" to "true" or vice versa.

4 User interface design

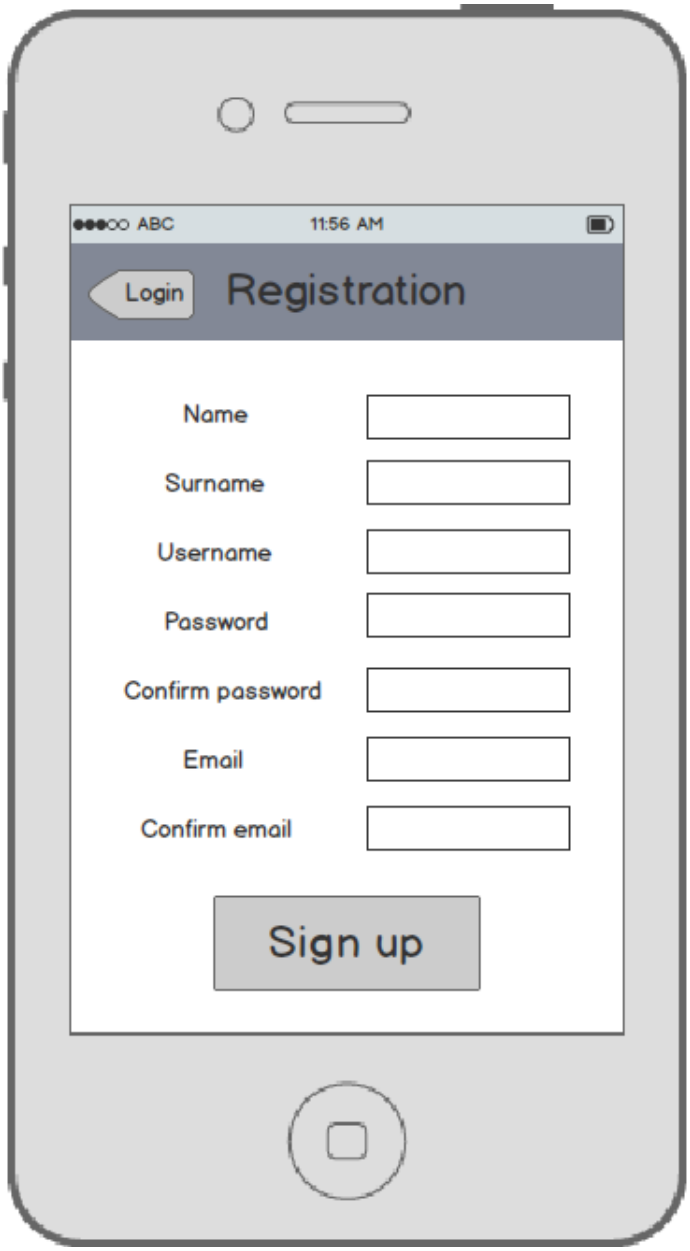
4.1 Log in page

This is the page that shows up when the app is opened. From here guests can log in or register themselves if they don't have an account yet.



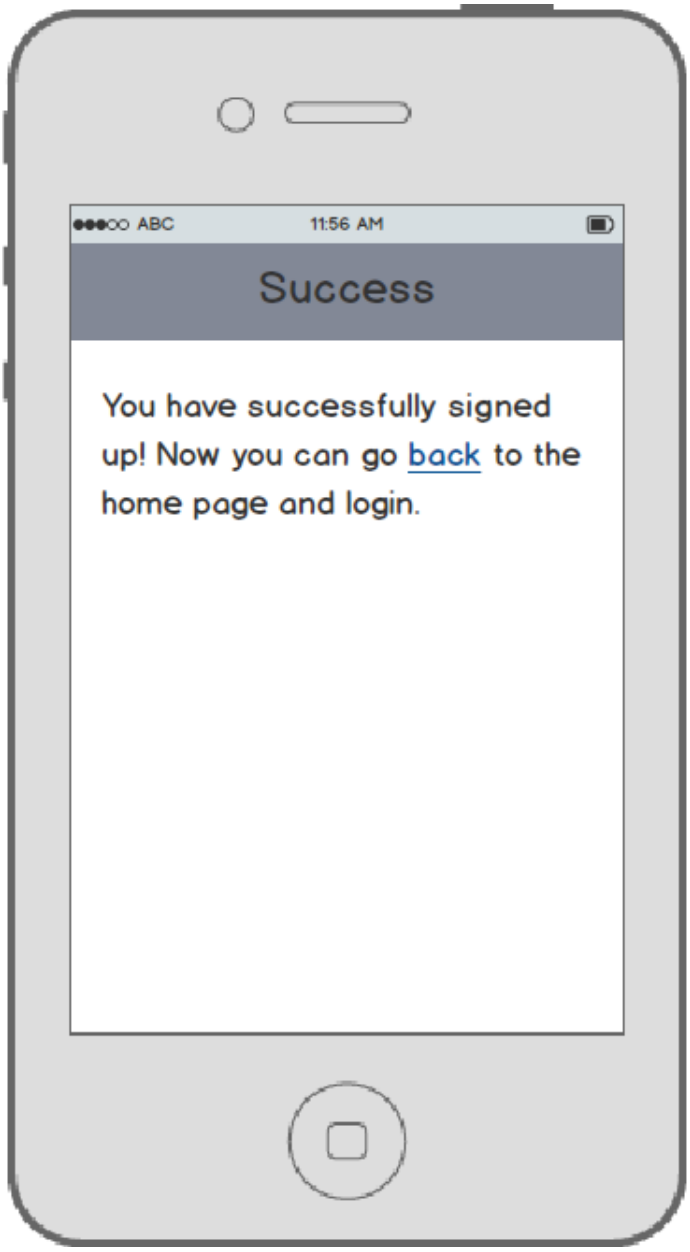
4.2 Registration page

Here guests can register themselves by filling all the mandatory fields (the ones shown).



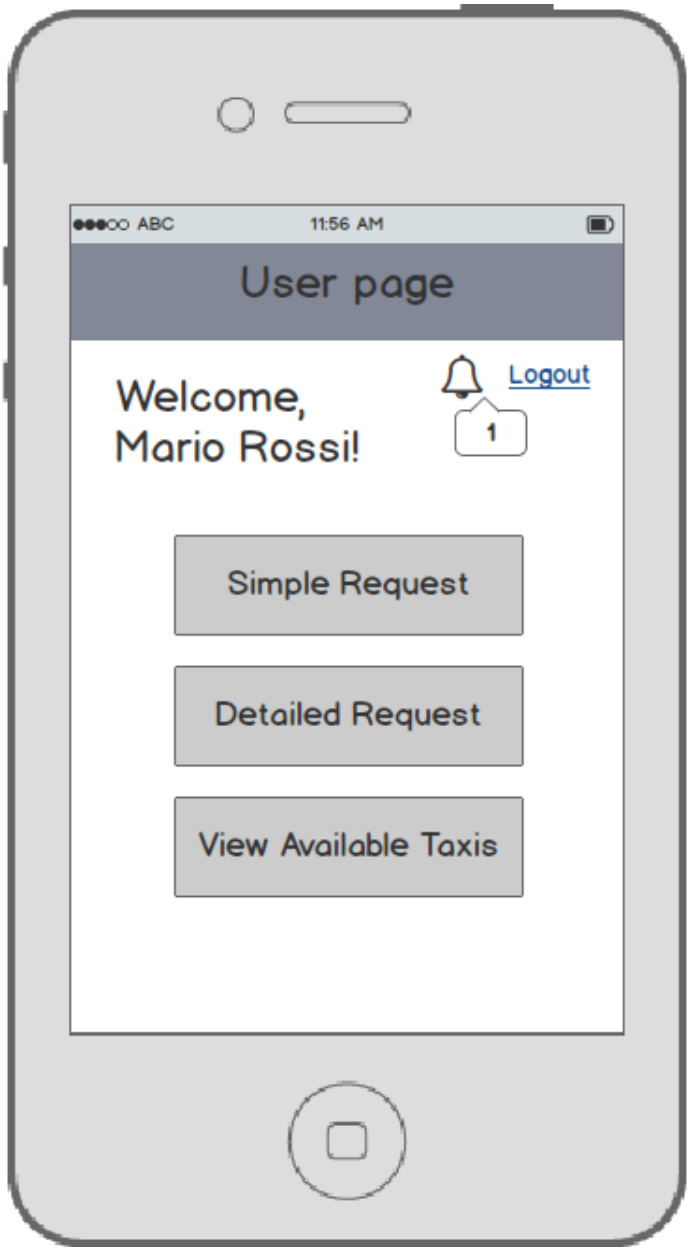
4.3 Confirmation page

If the registration was successful this page shows up.



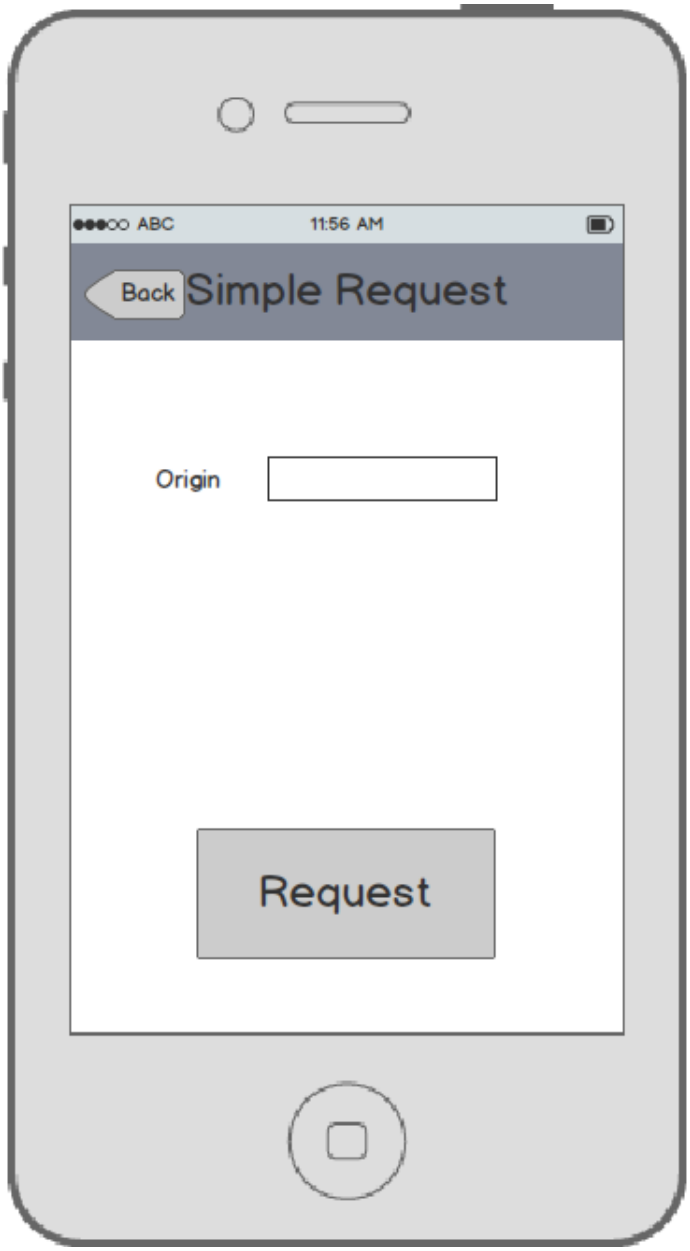
4.4 User page

After logging in, users see this page from which they can use all of their functionalities: request a ride, see available taxis in their area and read notifications.



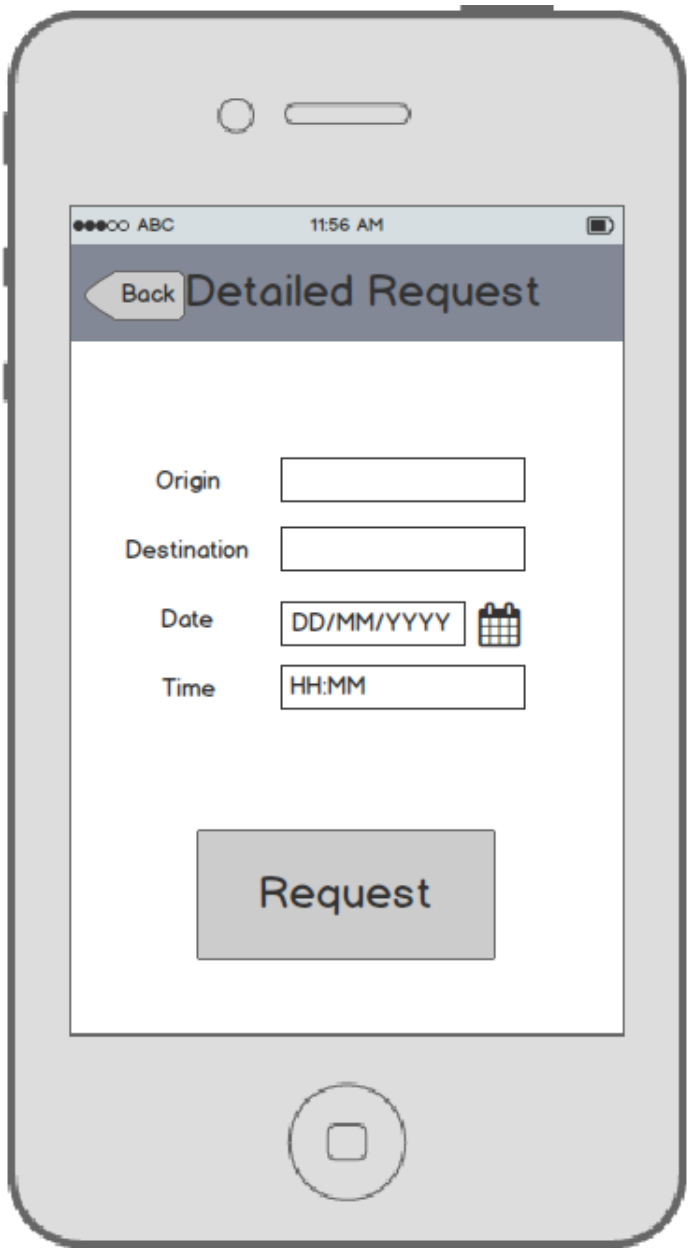
4.5 Simple Request

Users can request a ride by inserting its origin, that is where the driver should pick them up.



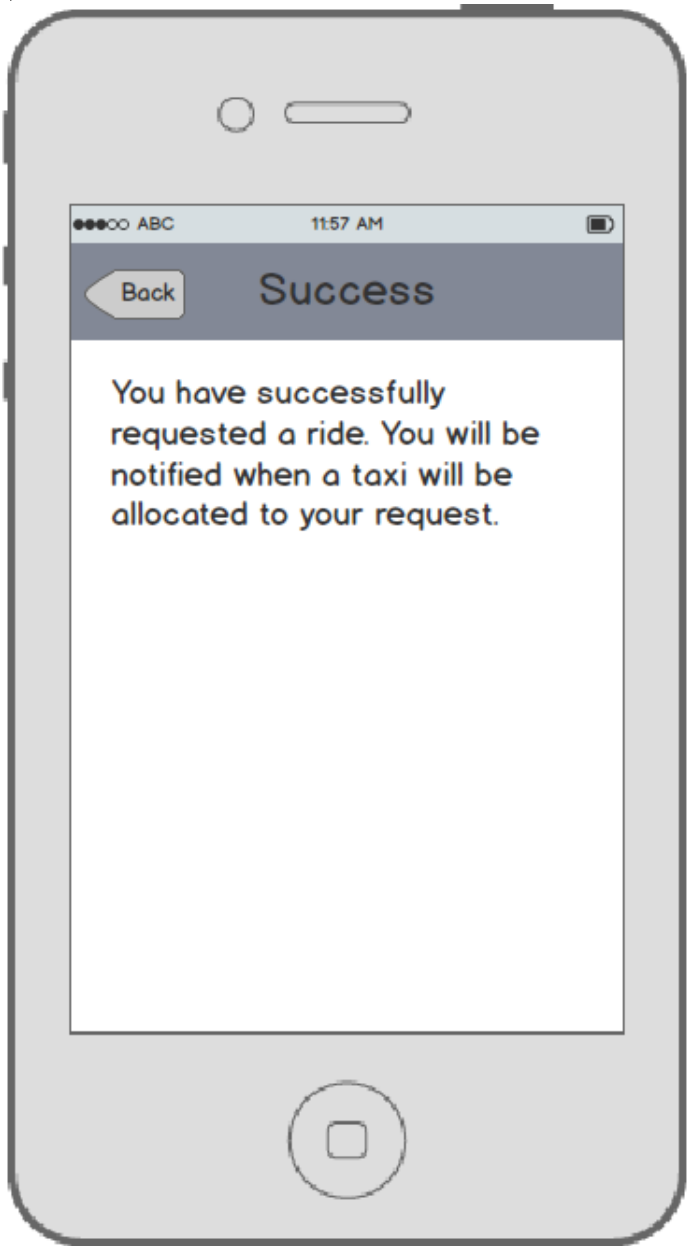
4.6 Detailed Request

Users can make a detailed request by filling this form.



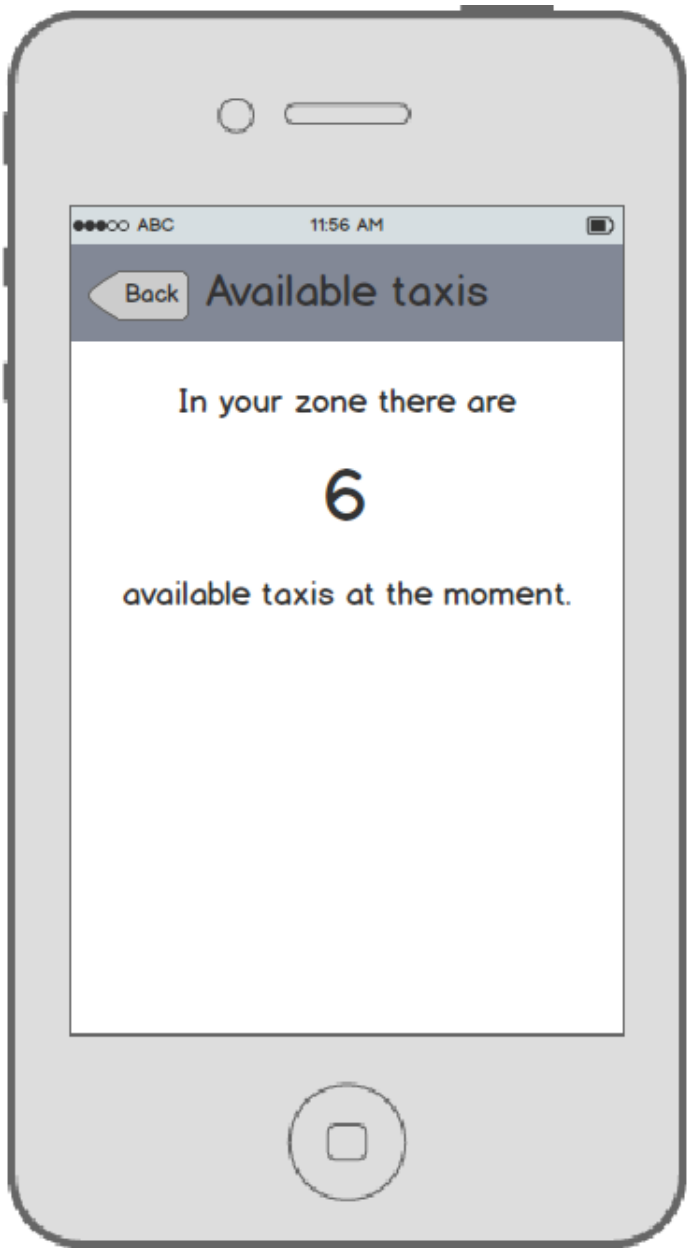
4.7 Request confirmation page

Once a request has been dispatched, this page is shown to the user as confirmation,



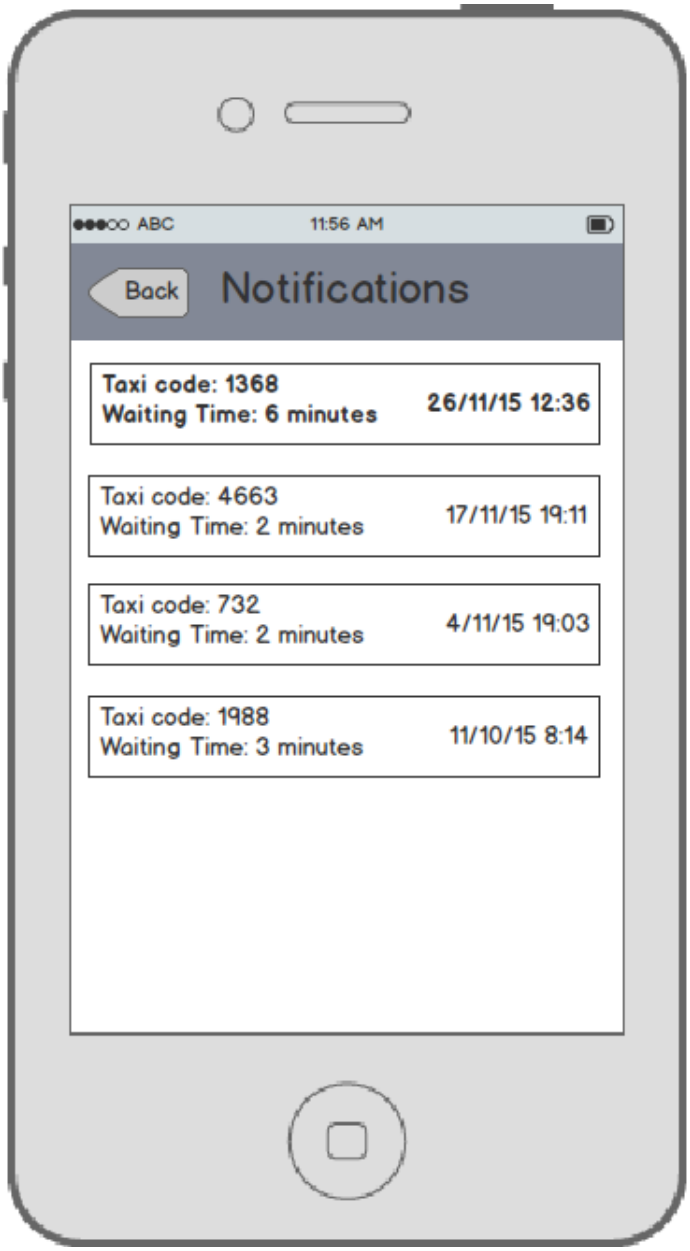
4.8 Available taxis page

This page shows the number of available taxis in the area that the user is in.



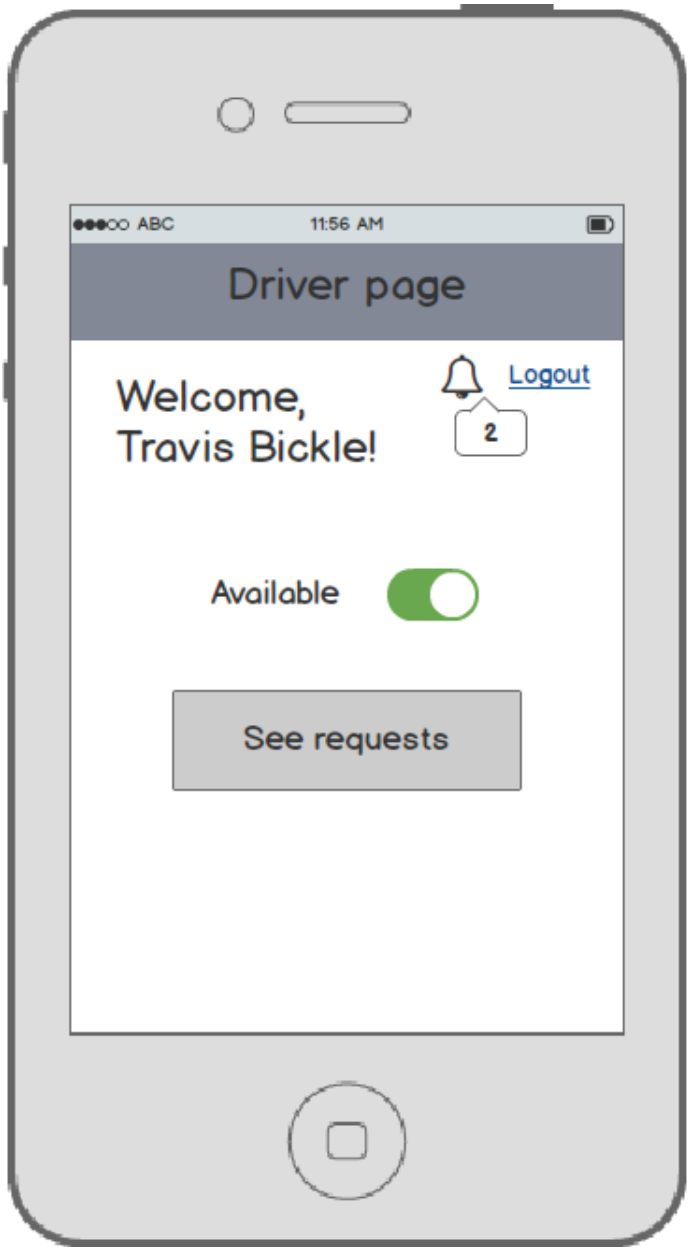
4.9 User notifications page

Here users can see notifications about the requests they made. The new ones that hasn't been read yet are written in bold.



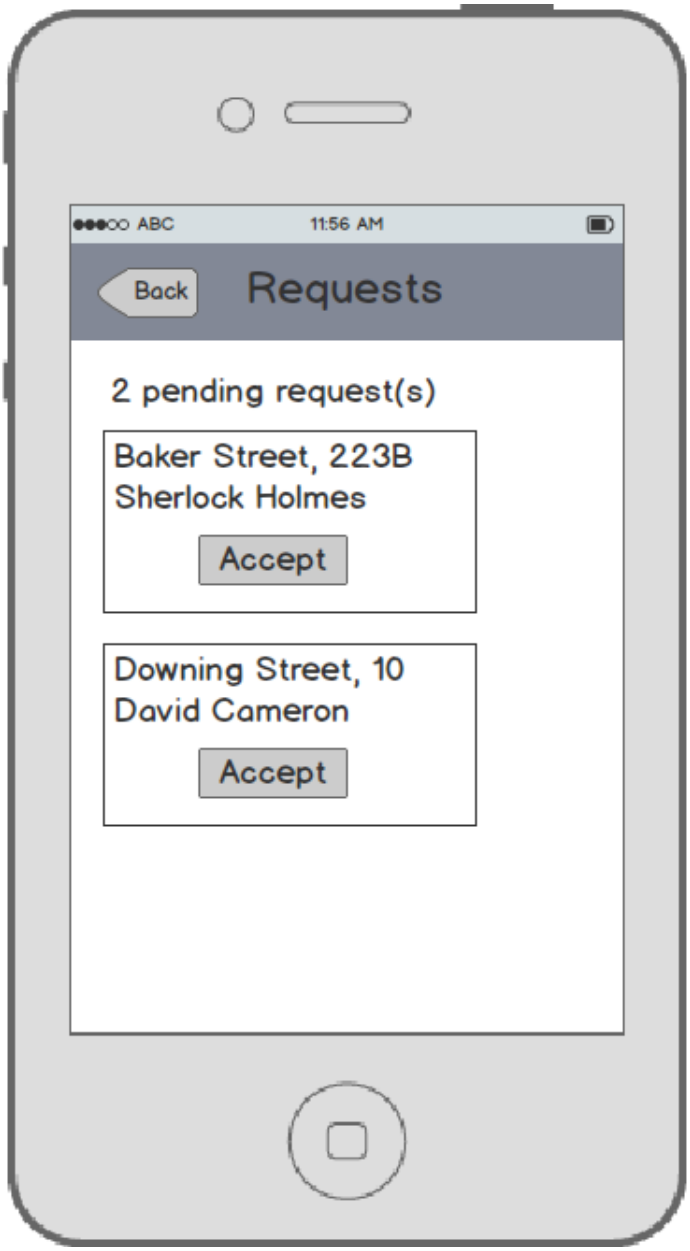
4.10 Driver page

This is the page that driver see after they log in. They can set themselves as available and see incoming requests.



4.11 Requests page

In this page drivers can see pending requests. If a driver is first in line, he can receive multiple requests but obviously can only accept one.



5 Requirements traceability

Here we're going to go through all the goals in the RASD to check if the requirements are satisfied.

5.1 Guests requirements

- [G1] Register themselves into the system

  Guests can only the see login, registration and confirmation pages, because when they login they are automatically "promoted" to users or drivers

  The filling of the mandatory fields of the form is guaranteed by the algorithm, who discards also if they "confirmation" fields don't match or if the username or email are already in use.

5.2 User requirements

- [G2] Log into the system

  This too is guaranteed by the algorithm, which checks if the fields are filled, then if the username exists and in the end checks if the username matches the password.

- [G3] See number of taxis of the zone he's in

  The application is able to guarantee this through the "View free taxis" function, which queries the database for the information.

- [G4] Make request for a simple ride

  The algorithm for this function checks if the "origin" field is empty. If it is, the request can not be fulfilled.

- [G5] Make request for a detailed ride

  As before, the algorithm makes sure that the input is not empty and formally correct.

5.3 Driver requirements

- [G6] Set themselves as available

  The driver has this function in the personal page and so he can toggles his availability.

- [G7] Read and accept requests

  In his personal page the driver can access to the page where he can read the incoming requests and eventually accept them. The application then proceeds to allocate the driver to the ride and set himself as unavailable.

5.4 System requirements

- [G8] Notify passengers after the confirmation of a simple request

  The algorithm of the simple ride request takes care of this by notifying the user as soon as the taxi is allocated.

- [G9] Notify passengers 10 minutes before the ride reserved through a detailed request

  The algorithm waits until 10 minutes before the ride and then begins to look for a taxi to allocate and notifying the user as soon as it's done.

- [G10] Forward requests to the first taxi in queue

  The application, after a ride request, always forwards it to the first taxi in queue of the user's zone. Then, when a driver is allocated, the application puts it in the last position and sets him as unavailable.

- [G11] After 30 seconds, forward the request to the second taxi in queue and put the first at the end

  A timer ensures that the driver answers a request within 30 seconds, otherwise he is set as unavailable and put in the last position. The latter will cause the queue to shift by a position and the request will then be forwarded to the driver who is now in the first one.

6 References

6.1 Software and tools used

- LyX: to redact and format this document.
- Microsoft Visio: to make the Component and Deployment Views.
- Astah Professional: to make Sequence Diagrams for the Runtime View.
- Balsamiq Mockups: to draw the user interfaces of the application.

6.2 Hours of work

This is the time spent to redact this document:

- Massimo Schiavo: 24 hours.
- Marco Edoardo Cittar: 24 hours.