



Politecnico di Milano  
A.A. 2015-2016  
Software Engineering 2: “myTaxiService”  
DesignDocument

Luiza Bentivoglio, Michele Cantarutti  
4 December 2015

# Summary

<b>1. INTRODUCTION .....</b>	<b>4</b>
1.1 PURPOSE.....	4
1.2 SCOPE.....	4
1.3 DEFINITIONS, ACRONYMS, ABBREVIATIONS.....	4
1.4 REFERENCE DOCUMENTS.....	5
1.5 DOCUMENT STRUCTURE.....	5
<b>2. ARCHITECTURAL DESIGN .....</b>	<b>6</b>
2.1 OVERVIEW.....	6
2.2 HIGHLEVEL COMPONENTS AND THEIR INTERACTION .....	6
2.3 COMPONENTVIEW.....	8
2.4 DEPLOYMENTVIEW.....	9
2.5 RUNTIMEVIEW.....	10
2.5.1 <i>Runtime units</i> .....	10
2.5.2 <i>Request a taxi</i> .....	11
2.5.3 <i>Reserve a taxi</i> .....	12
2.5.4 <i>Cancel a reservation</i> .....	13
2.5.5 <i>Modify a reservation</i> .....	14
2.5.6 <i>Inform about the availability</i> .....	15
2.6COMPONENTINTERFACES.....	16
2.7 SELECTEDARCHITECTURAL STYLES AND PATTERNS.....	17
<b>3. ALGORITHM DESIGN.....</b>	<b>18</b>
<b>4. USER INTERFACE DESIGN.....</b>	<b>20</b>
<b>5. REQUIREMENTS TRACEABILITY.....</b>	<b>31</b>



# 1. Introduction

## 1.1 Purpose

The purpose of this document is to give a functional description of the system. In order to do so, we'll analyze the architectural design, by showing which components will make up our system and how they will interact with one another. We will also show the algorithm design, focusing on the most relevant algorithmic part of the project, which is the way the system manages the requests and how it assigns them to taxi drivers. Lastly, this document will provide an overview on how the user interfaces will look like.

## 1.2 Scope

The scope of this document is to show our choices with regard to the architecture of our system. We'll show our choices about which components to use and how to make them interact through chapter two, whereas the choices in style and design will be discussed at the end of chapter two.

## 1.3 Definitions

CLIENT/USER: these two terms will often be used interchangeably and they're used to describe whoever uses the web application or the mobile application from the client's side, thus passengers and taxi drivers.

PASSENGER: a registered user who is a customer of our service and therefore has an account and can make generic requests to request a taxi.

GENERIC REQUEST: this term is used to identify requests made by passengers to be picked up by a taxi. Requests and reservations belong to this category.

REQUEST: a call made by a passenger to be picked up by a taxi driver within a short time at a specified place.

RESERVATION: a call made by a passenger to be picked up by a taxi driver at a specified time and place. The reservation must occur at least two hours before the ride.

## **1.4 Reference documents**

We used some documents as starting points for our design. Some of these documents are documents taken from the teacher's website, mostly: "7 Design" and "8 Architectural Styles".

Moreover, we used Wikipedia as a starting point to lay out some of the diagrams, mostly for the Component view and the Deployment view diagrams.

## **1.5 Document structure**

Our Design Document is structured like this: the first part will cover architectural design, which is divided into more sections to show how we decided to design our system and which components make it up, how they interact and some possible scenarios represented through sequence diagrams. Lastly, we will show the architectural styles and patterns we decided to use.

## **2. Architectural design**

### **2.1 Overview**

In the next sections we will show the architectural design of our project. At first we will only introduce the high level components of our architecture and describe the main interaction between them, without getting into details. After this introduction, we will refine what we've show and identify sub-components that make up our components. Along with this, we will show what kind of interaction there is between components, by showing who offers a certain service and who uses it. We will also show what executable will be running and on which device they'll be running. Lastly, we'll talk about the architectural styles and patterns we decided to use in our project.

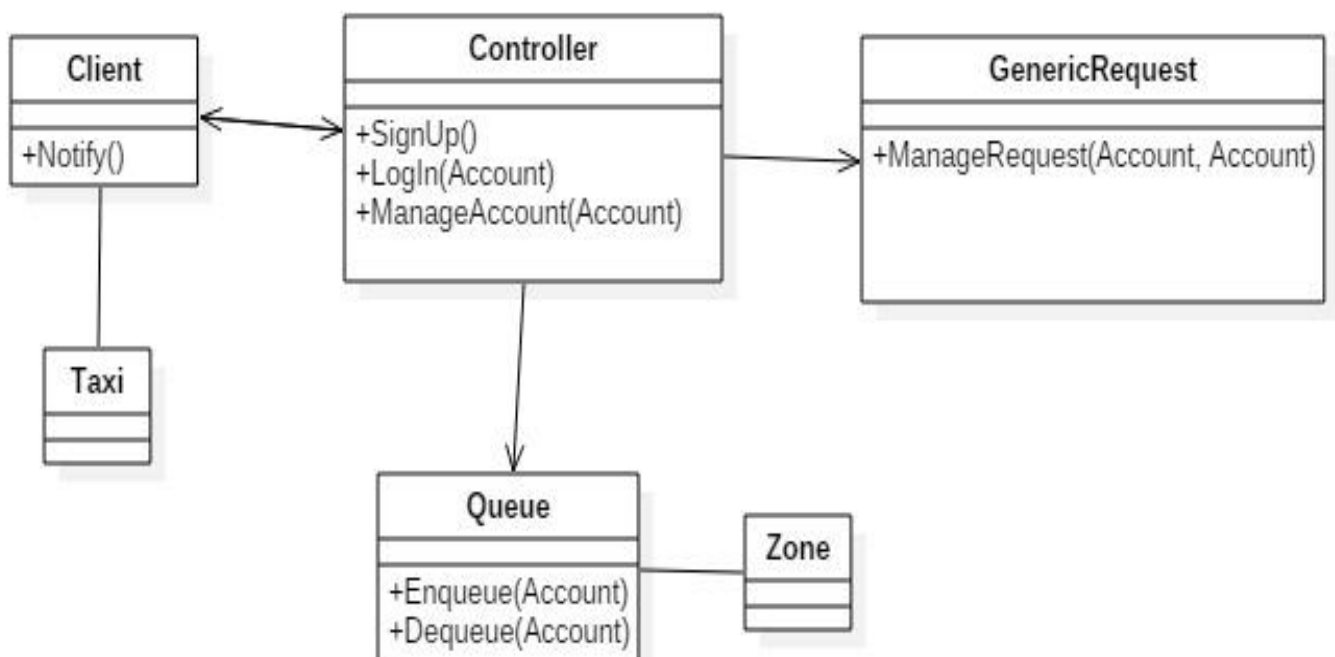
### **2.2 High level components and their interaction**

Here we will introduce the high level components of the architecture of our project. This diagram will provide an overview of the entire system, identifying the main components that will be developed for the product. We tried to use possibly nontechnical terms so that it could be understandable to the administrators of the system, too.

At the core of our system we decided to use a component called "Controller" which has the most important role and offers the greatest number of features. The Controller is a component which accesses data all over the system, for example to gather information about currently available taxis in the queue of a certain zone, or simply to view the list of reservations associated with a certain Passenger. The Controller also offers services, such as the management of the clients' accounts, which includes signing up, signing in, changing their availability status (if they're drivers), making requests (if they're passengers) and it also notifies taxi drivers about cancelled reservations and passengers if there are no available taxis or there is a delay.

It's important to point out that it might seem like there's a duplication, in fact, the Controller offers a service called ManageAccount to Clients which includes the possibility to make a generic request, and the GenericRequest component offers a service called ManageRequest to the Controller; therefore it might seem like there are two functionalities to create a request. Though, this is not the case, because the ManageAccount service only allows Clients to make a request, but it

won't actually create one, because the Controller first has to check if there any available taxis who can accept the call. The request will be created in the system only after a taxi driver has been assigned to the request, otherwise the request will be dismissed at once, and it wouldn't make sense to save it on the database. The way components are connected to one another is not always the same. For example, the arrow that connects the Controller to the Queue has one direction because only the Controller needs to access the queue (to know which taxis are available), whereas the link connecting the Controller and the Client is bidirectional, because the two components need to talk to each other (because they offer services to each other).

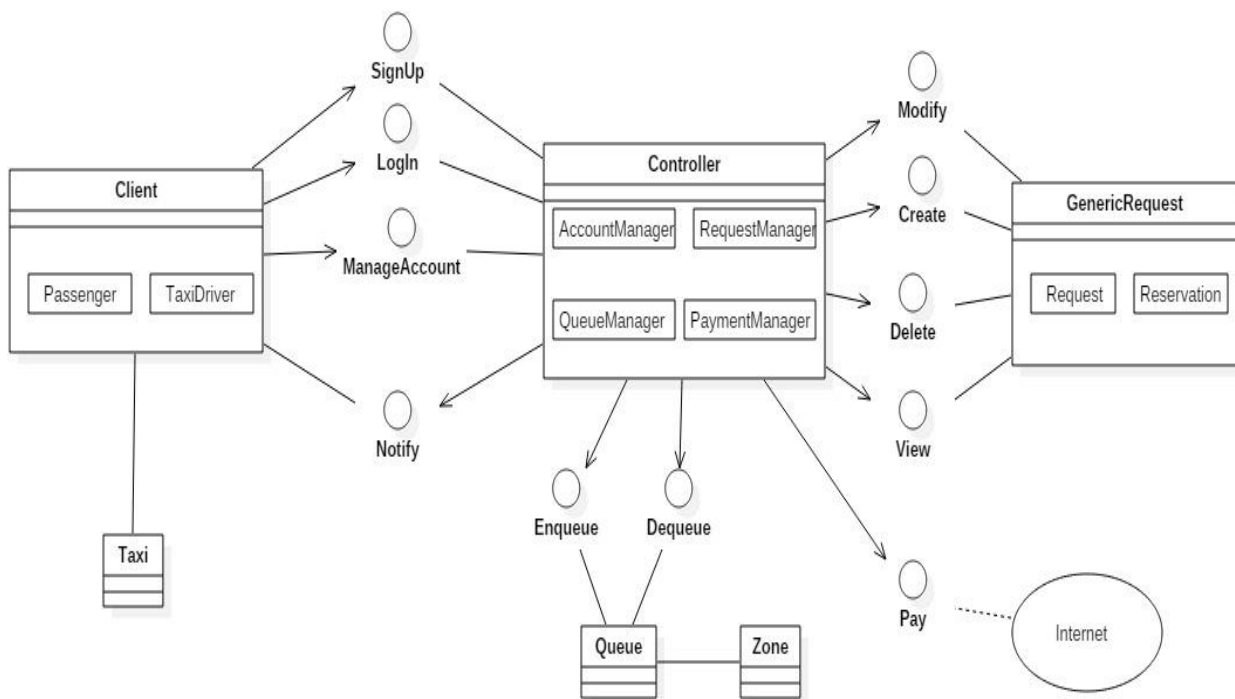


## 2.3 Component view

As we can see in the diagram below, we have specified which sub-components make up the components. In the case of the Client, we decided to use two sub-components called Passenger and TaxiDriver. They can both use the services offered by the Controller, which are the SignUp, SignIn and ManageAccount interfaces, and they both offer the possibility of being notified by the Controller through the Notify interface. The ManageAccount interface allows the Clients to carry out several operations, like viewing past reservations, changing their status, deleting and modifying reservations, and other operations related to their account, which will be further shown below in the UI pictures.

As for the Controller, there are four sub-components, called AccountManager, RequestManager, QueueManager and PaymentManager. Thanks to the Create, Delete, View interfaces offered by the GenericRequest interface, the Controller can manipulate generic requests in the system, so as to create, delete and view them according to what Clients ask it to do. Lastly, the Controller is able to Enqueue and Dequeue taxi drivers according to the changes of their availability statuses, which it receives thanks to the ManageAccount interface.

As we can see in the bottom right corner, there's an object called "Internet" which of course is not strictly a part of our system, but it is still used by the Controller as a means of communication for it to exchange information with whatever banking system is being used, may it be PayPal or something else.

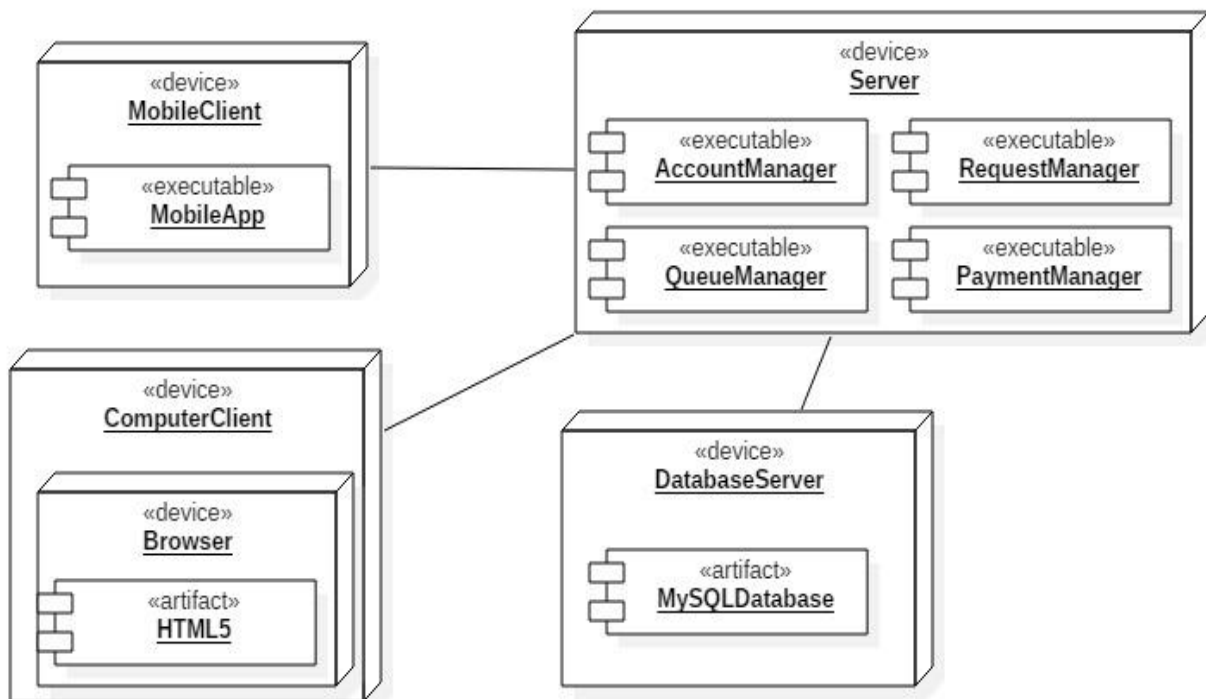




## 2.4 Deployment view

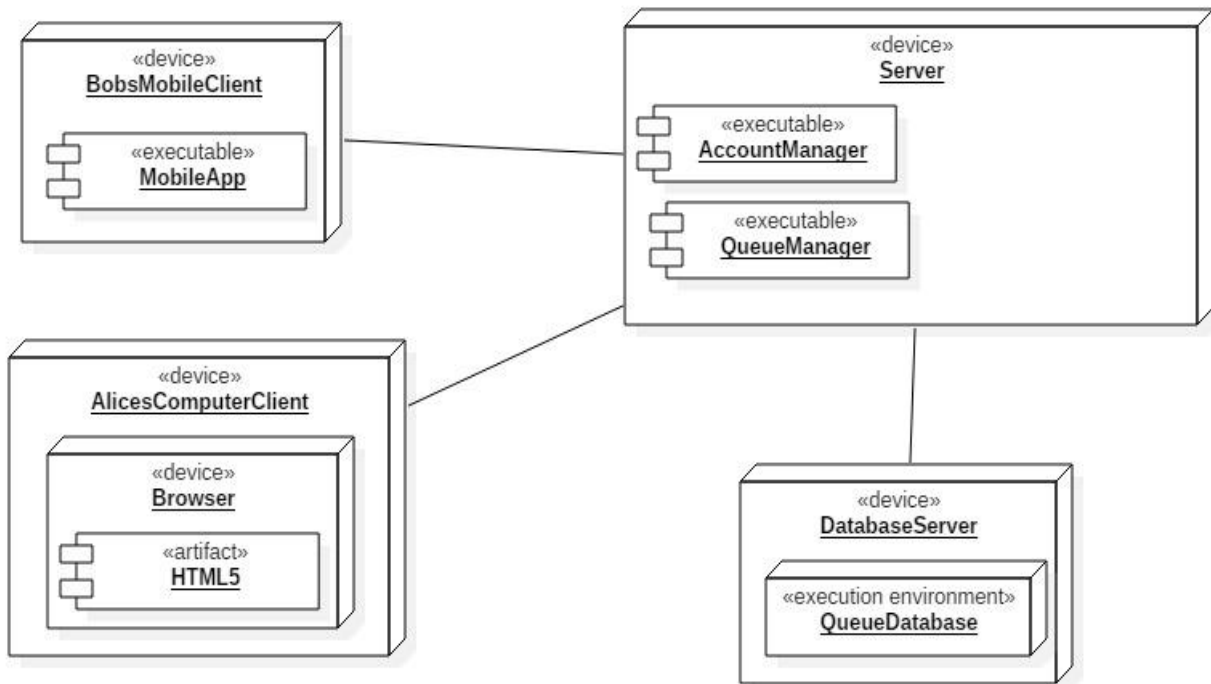
The following diagram is the deployment view, that is the identification of the artifact that needs to be deployed to have the system working. We think there are three main parts. The first part, which is the core of our system, is the server and it has several executables. Each one of them is used for a specific function and the server often needs to get data from a database, therefore the database is without doubt another device that is needed in our system.

The last device in the diagram is the one needed for the client. Because technically the interaction of the mobile client and the computer client differ quite a bit, we felt like it's necessary to point out this difference by actually using two different devices.

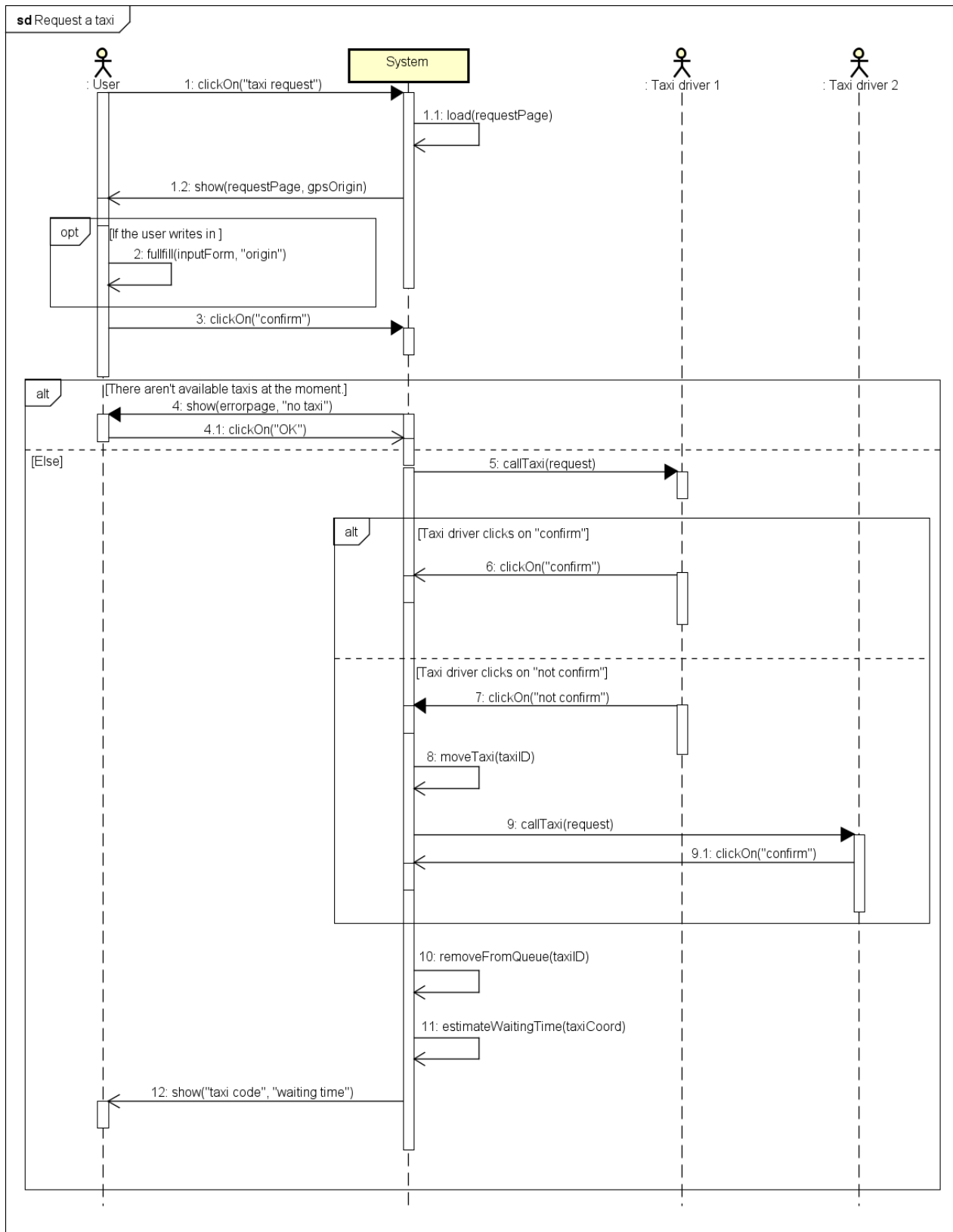


## 2.5 Runtime view

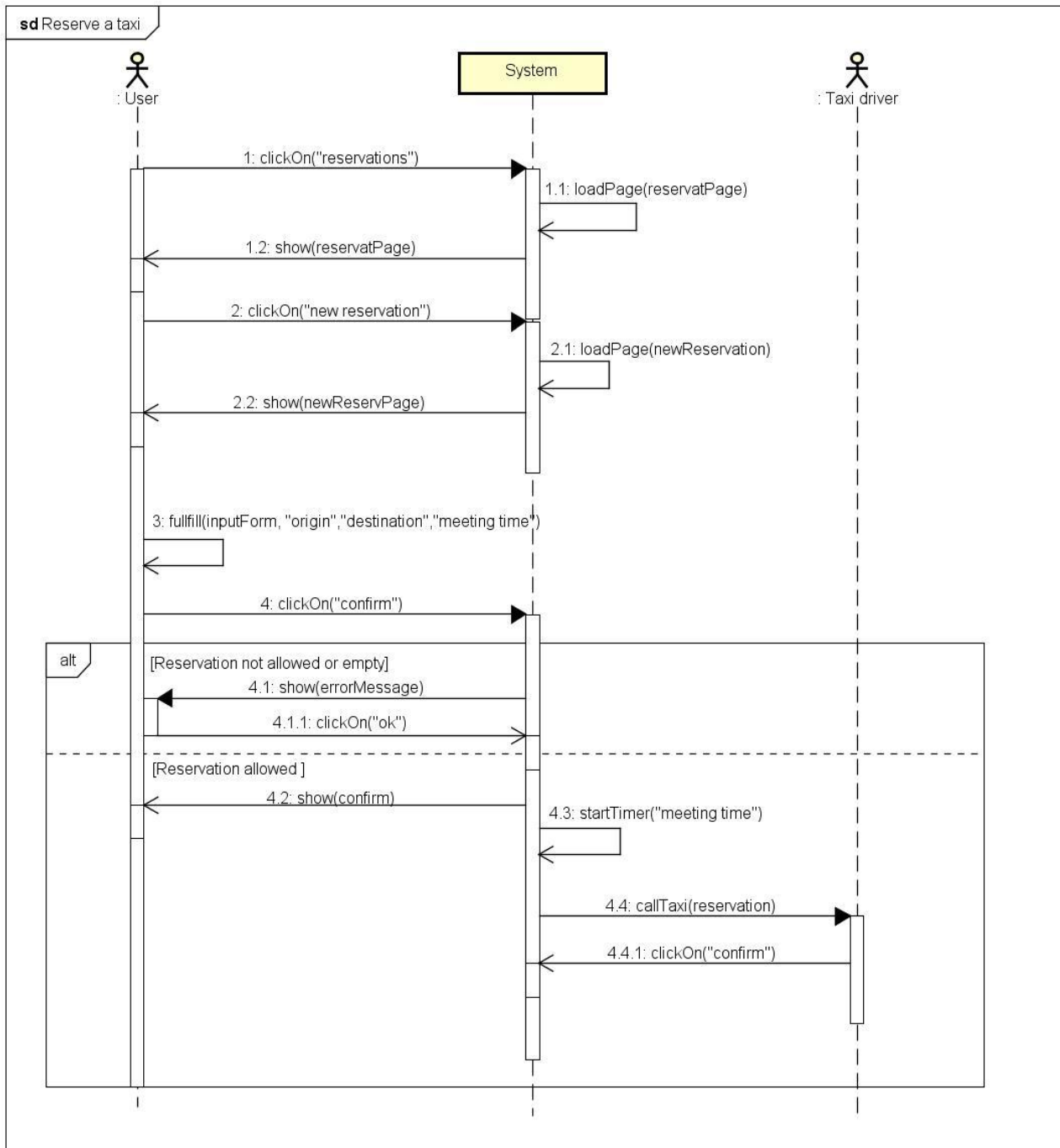
### 2.5.1 Runtime units



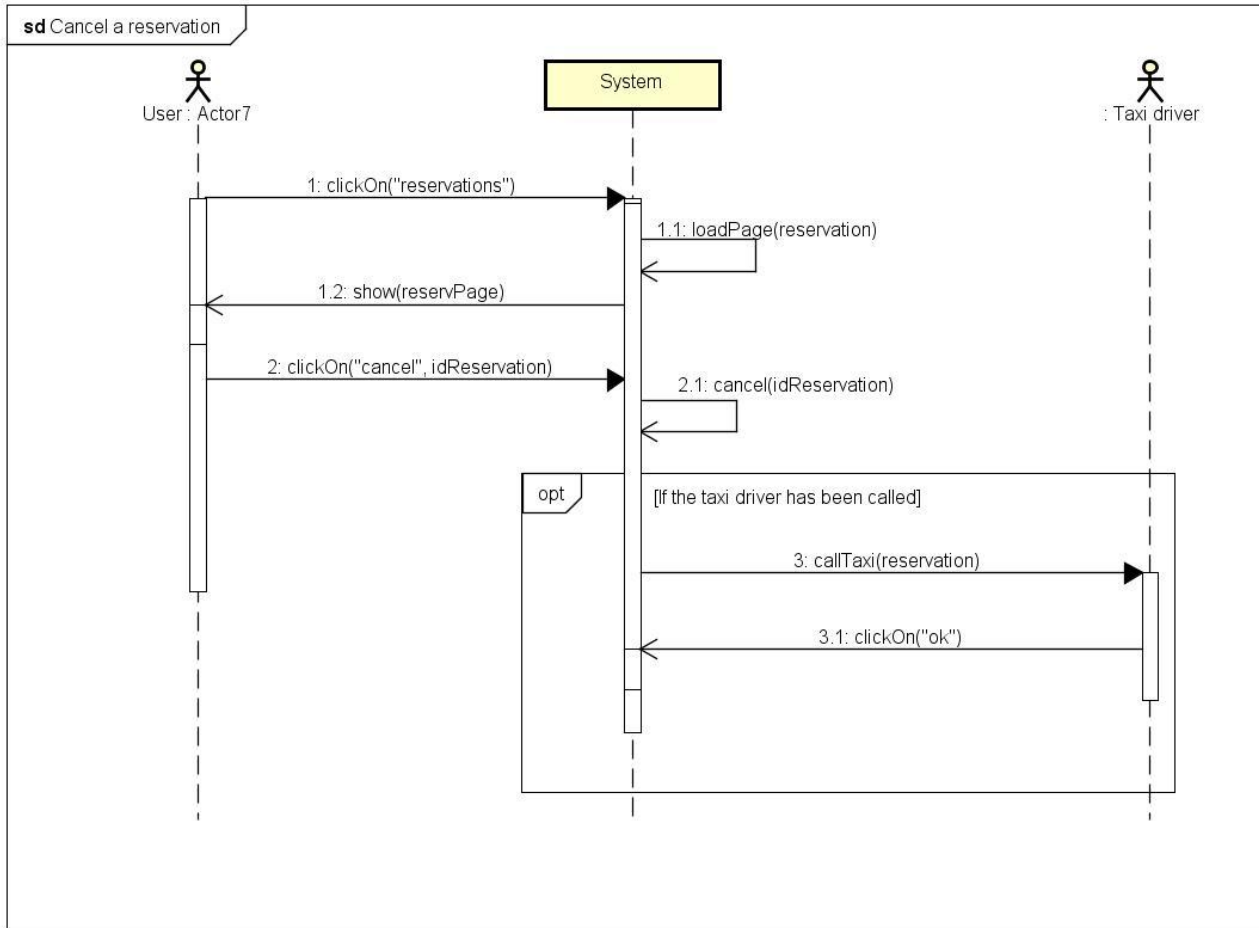
## 2.5.2 Request a taxi



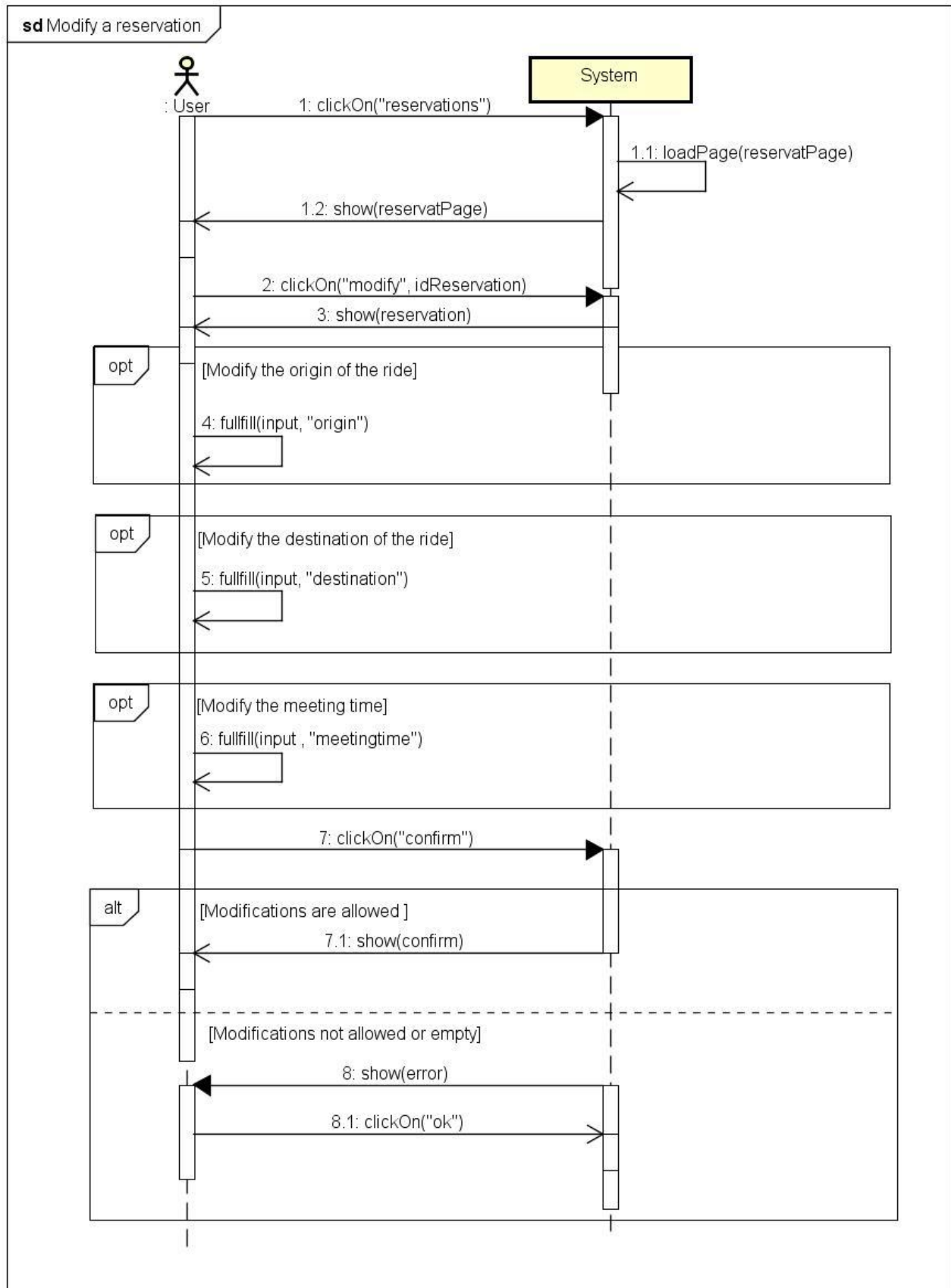
## 2.5.3 Reserve a taxi



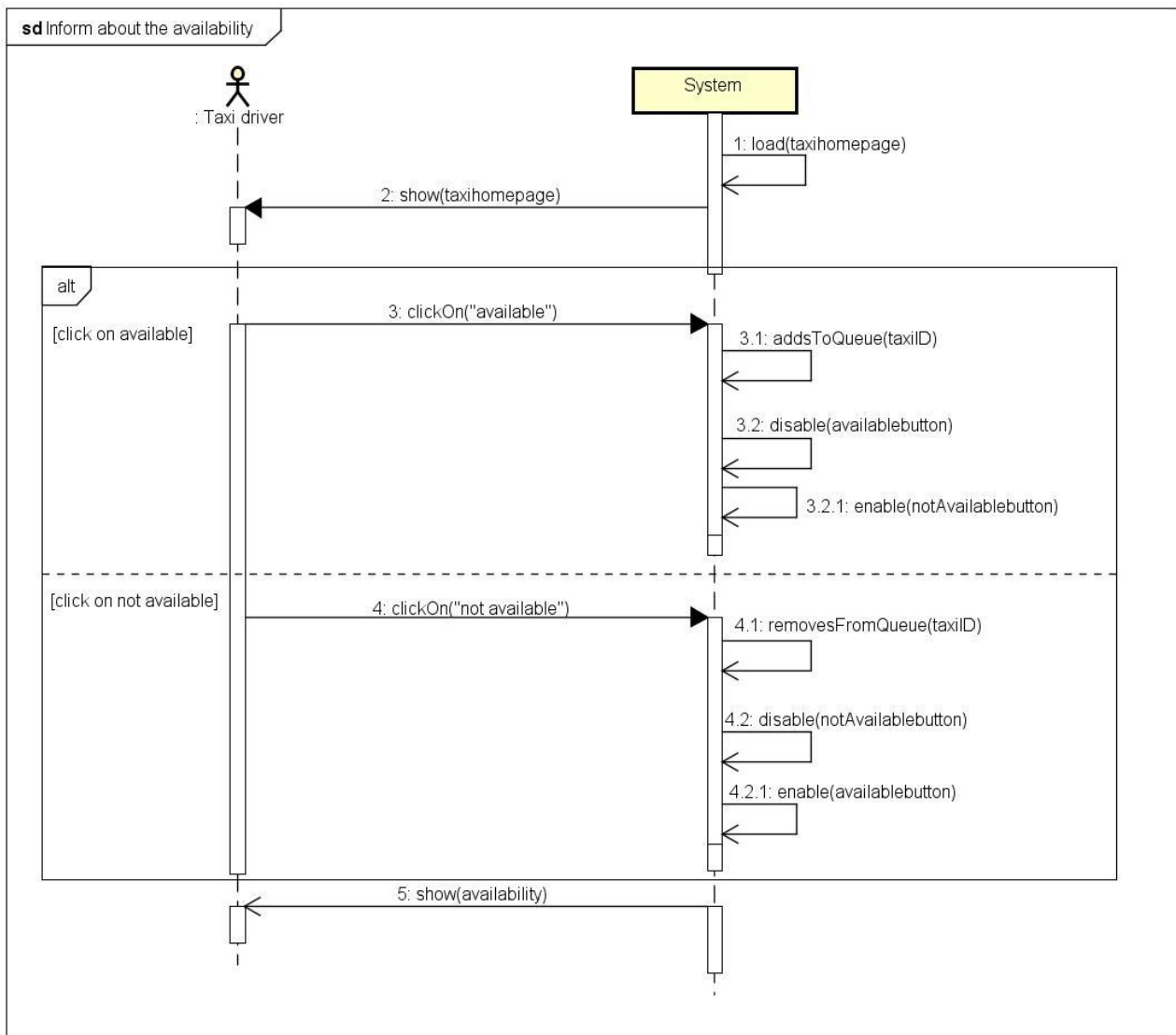
## 2.5.4 Cancel a reservation



## 2.5.5 Modify a reservation



## 2.5.6 Inform about the availability



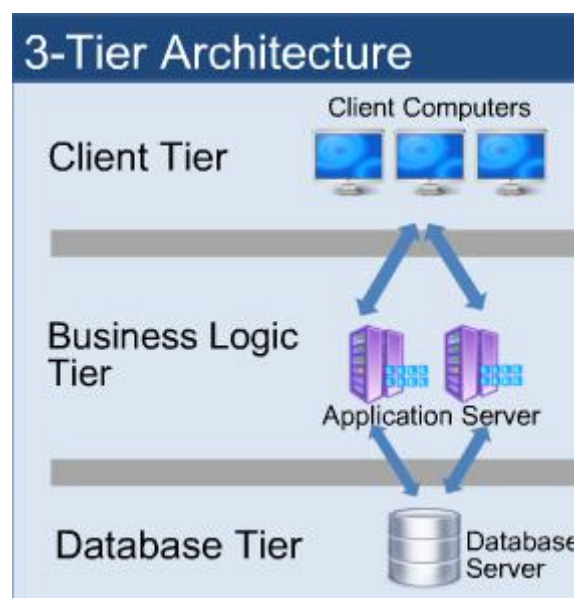
## 2.6 Component interfaces

INTERFACE	INPUT	OUTPUT
SignUp	User data	Personal page
LogIn	User data	Personal page
ManageAccount	The option that the user has clicked or touched	A page is shown according to the clicked option
Notify	Username	None
Enqueue	The taxi driver who's changed his/her availability status to AVAILABLE	Success or failure
Dequeue	The taxi driver who's changed his/her availability status to UNAVAILABLE	Success or failure
Modify	The genericrequest received from ManageAccount, which is one the passenger wants to modify	Success or failure
Create	The generic request received from ManageAccount, which is one the passenger wants to create	Success or failure
Delete	The generic request received from ManageAccount, which is one the passenger wants to delete	Success or failure
View	The genericrequest received from ManageAccount, which is one the passenger wants to view	Success or failure



## 2.7 Selected architectural styles and patterns

We decided to use a client-server architecture for our system, a three-tier architecture to be precise, which is made up of the client tier, the business logic tier and the database tier. We believe the client should be as “thin” as possible because many users will be using the mobile application and therefore they will be on a device that does not have a great computational capacity. Moreover we don’t want to store too much data on mobiles, because this might annoy users, given the limited storage capacity of mobiles. This distribution will allow us to distribute different tasks to different parts of the system: the presentation task will be given to the client tier, the business logic will be given to the server and the data management will be given to the database. Lastly, during the implementation of the coding part of the project, this architecture will get along well with the MVC pattern (model, view, controller).



As for patterns, the most important pattern we used for our project is the decorative pattern. This can be seen in the SignUp process for example. In fact, instead of creating two completely different processes, we decided to use the same one, which is enriched (thus “decorated”) with a different feature according to the user who will use it. In the case of the taxi driver, the SignUp process requires the user to type in a registration code, whereas this doesn’t happen for Passengers, who are asked if they want to add an additional payment method instead. Though, the rest of the SignUp process is the same for both kinds of users.

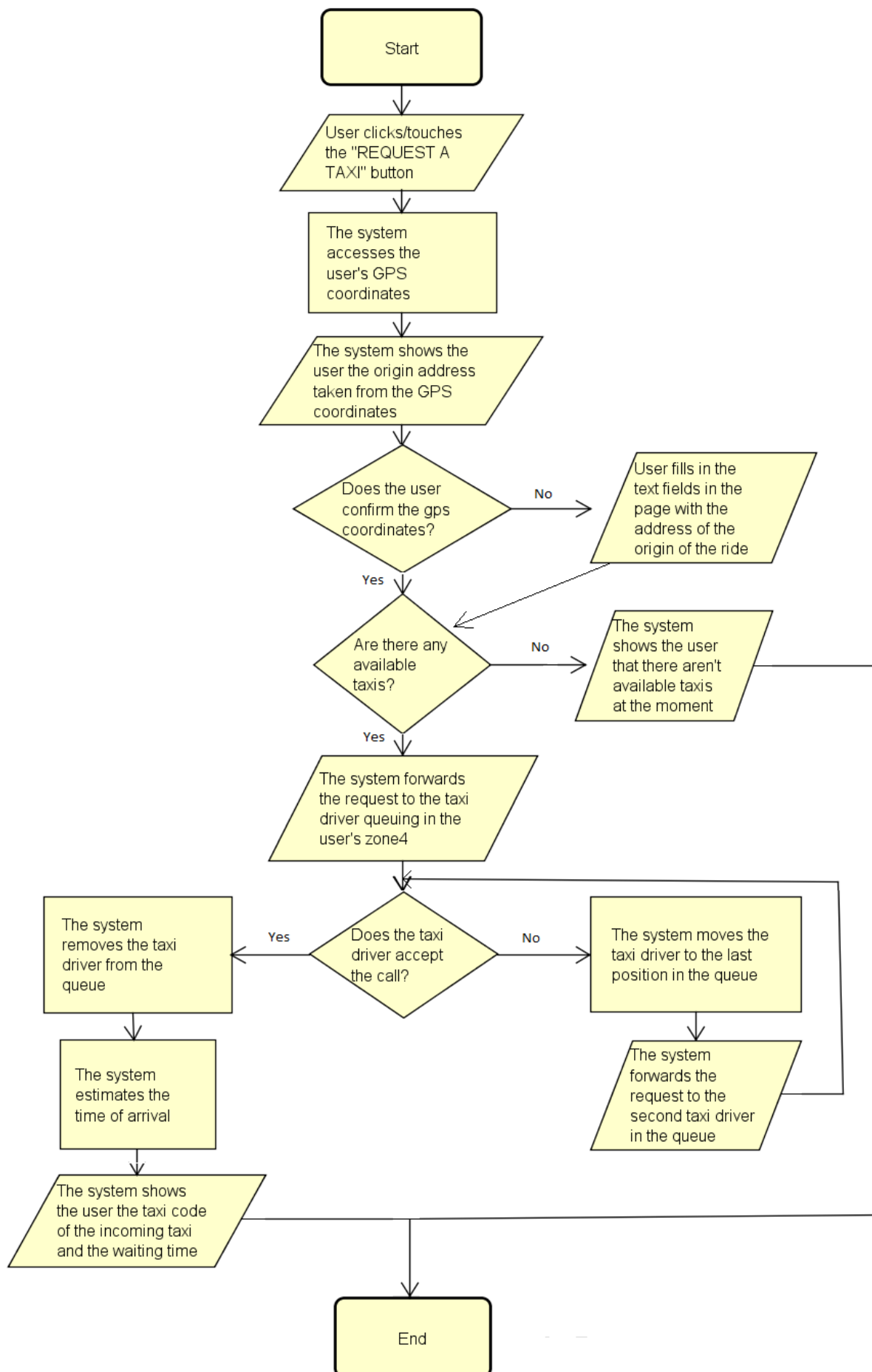
### 3. Algorithm design

Having to decide which algorithmic part of our project to describe, we decided to focus on the part that manages requests, because we feel like it's the very core of our project.

As it's shown in the flow chart, the whole process begins when a passenger requests a taxi by either clicking on the "request a taxi" button, or touching it if he/she is using the mobile version of the application. In order to make things faster for the passenger, the system will automatically gather information about the current location of the passenger through the GPS, so that he/she won't have to type it down, unless the displayed address is wrong, in which case the passenger will have to write it by himself/herself.

The system will check whether there are any available taxis in the zone where the passenger currently is and if it can't find at least one, it will notify the passenger that no one will be able to come pick him/her up. Otherwise, the system will forward the request to the first available taxi driver. If the taxi driver doesn't accept the call, he/she will be moved to the last position in the queue and the request is forwarded to the second taxi driver.

When a taxi driver accepts the call, he/she is removed from the queue. Then, the time of arrival is estimated and shown to the passenger, along with the code of the incoming taxi.



## 4. User interface design

This is the page the user sees when he/she goes on the myTaxiService page to sign up for the service. This page is shown on the web application, which means the user is accessing it through a computer and therefore we can assume the user is a passenger, because taxi drivers can only access it through a mobile. Because of this, the user is only asked for basic data like name, surname, password, email and telephone number but a registration code is not required, unlike for taxi drivers.



The screenshot shows a web browser window with the address bar displaying <http://www.mytaxiservice/signup>. The page features a navigation menu with links: [Home](#), [Request a taxi](#), [Reservations](#), [Payment](#), and [Log out](#). Below the navigation menu, there are five input fields for user registration: Name, Surname, Password, Email, and Telephone. Each field is preceded by its respective label. Below the Telephone field, there is a link: [Add a payment method \(optionally\) ▼](#). At the bottom right of the form, there is a button labeled CONFIRM.

<a href="#">Home</a>	<a href="#">Request a taxi</a>	<a href="#">Reservations</a>	<a href="#">Payment</a>	<a href="#">Log out</a>
Name:	<input type="text"/>	Surname:	<input type="text"/>	
Password:	<input type="text"/>	Email:	<input type="text"/>	
Telephone:	<input type="text"/>			
<a href="#">Add a payment method (optionally) ▼</a>				
<div>CONFIRM</div>				

As we can see in the pictures, the mobile version shows a couple of minor differences. First of all, at the very beginning, the user is asked whether he/she is a passenger or a taxi driver. Depending on what they have selected, the following page will be slightly different. If the user selects “Passenger” then the following page will ask for basic data, plus, optionally, what additional payment method they would like to use. Otherwise, if the user selects Taxi Driver, they will be asked for basic data along with the registration code, which has been given to the taxi driver by the administrator of the system (which is the government of the city in the case of this project).



myTaxiService

Name

Surname

Telephone

Email

[Add a payment method \(opt\)](#)

CONFIRM

This is a wireframe of a mobile application registration screen. It features a header with a hamburger menu icon and the text 'myTaxiService'. Below the header, there are four text input fields labeled 'Name', 'Surname', 'Telephone', and 'Email'. A blue hyperlink 'Add a payment method (opt)' is positioned below the email field. At the bottom of the form area is a grey rectangular button labeled 'CONFIRM'. The entire screen is enclosed in a rounded rectangle with a thin black border, and a circular home button is visible at the very bottom.

myTaxiService

Name

Surname

Telephone

Email

Code

CONFIRM

This is a wireframe of a mobile application registration screen, similar to the one on the left but with a different set of fields. It features a header with a hamburger menu icon and the text 'myTaxiService'. Below the header, there are five text input fields labeled 'Name', 'Surname', 'Telephone', 'Email', and 'Code'. A grey rectangular button labeled 'CONFIRM' is positioned below the 'Code' field. The entire screen is enclosed in a rounded rectangle with a thin black border, and a circular home button is visible at the very bottom.

This is the page that is shown to the user when he/she is making a request. To make things faster and easier for the passenger, the system will get the current location of the passenger through geolocation, but he/she is able to modify if he/she wants to. By clicking “confirm origin” the passenger will send the request and then it will be managed by the system.

The screenshot shows a web browser window with the address bar displaying `http://mytaxiservice.com/requestataxi`. The page has a navigation menu with links: [Home](#), [Request a taxi](#), [Reservations](#), [Payment](#), and [Log out](#). On the left, a user profile for 'Mickey Mouse' is shown with icons for a person, email, phone, flag, and credit card. The profile details are: Name: Mickey Mouse (with an edit icon), Email: michey.mouse@live.it, Phone: 3358671922, Language: Italiano (in a dropdown menu), and Payment: Paypall. On the right, there is a map showing a location pin. Below the map, the 'Origin' is set to 'via Voto 30' in a text box. At the bottom right, there is a button labeled 'CONFIRM ORIGIN'.

In this case, the pictures represent the same situation as before, but they're on the mobile application instead. The picture on the left shows the basic menu that the passengers can see from their mobile application, which allows them to make requests, reservations, manage payments or log out.

As show in the picture on the right, just like before, instead of asking the origin of the ride, the system will automatically suggest the current location of the passenger as the origin of the ride (the location is known to the system thanks to the GPS). Either way, if the address obtained through the GPS is wrong or if the passenger would like to be picked up somewhere else, he/she can simply type down an address by himself/herself.

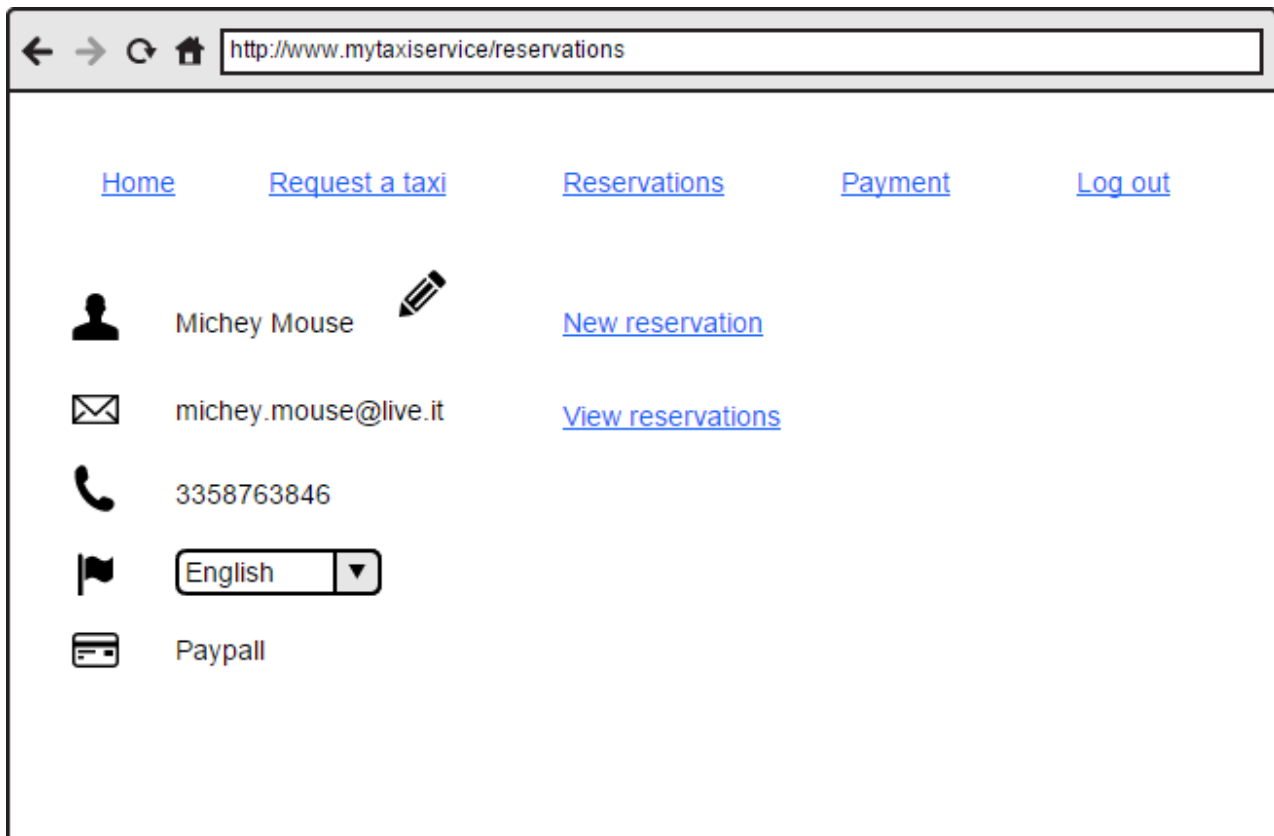





The following pictures show what happens when passengers decide to manage their reservations when they are on their computer. First of all, they can either choose if they want to make a new reservation or choose if they want to view their past and future reservations.

When they click “View reservations” they’re shown all their reservations, both the past and the future ones. Each reservation is shown with its corresponding details, such as the date, time, origine and destination of the ride. Next to each reservation are two buttons which allow passengers to delete or modify a certain reservation.

Lastly, the third picture shows what passengers see when they click “New reservation”. The interface is very easy and it simply asks passengers to fill in the fields with all necessary data, such as the date, meeting time, origin and destination of the ride.



[Home](#)
[Request a taxi](#)
[Reservations](#)
[Payment](#)
[Log out](#)




Mickey Mouse





mickey.mouse@live.it

3358763846


English

Paypal



▼ Origin	▼ Destination	▼ Date	▼ Time	
Via Golgi 1	Via Mazzini 13	02/12/15	9.00	 
Via Bruni 2	Via Mazzini 13	06/12/15	10.20	 

[Home](#)
[Request a taxi](#)
[Reservations](#)
[Payment](#)
[Log out](#)




Mickey Mouse

mickey.mouse@live.it

3358671922

Italiano

Paypal



Origin

Destination

Date

25

▼

OCT

▼

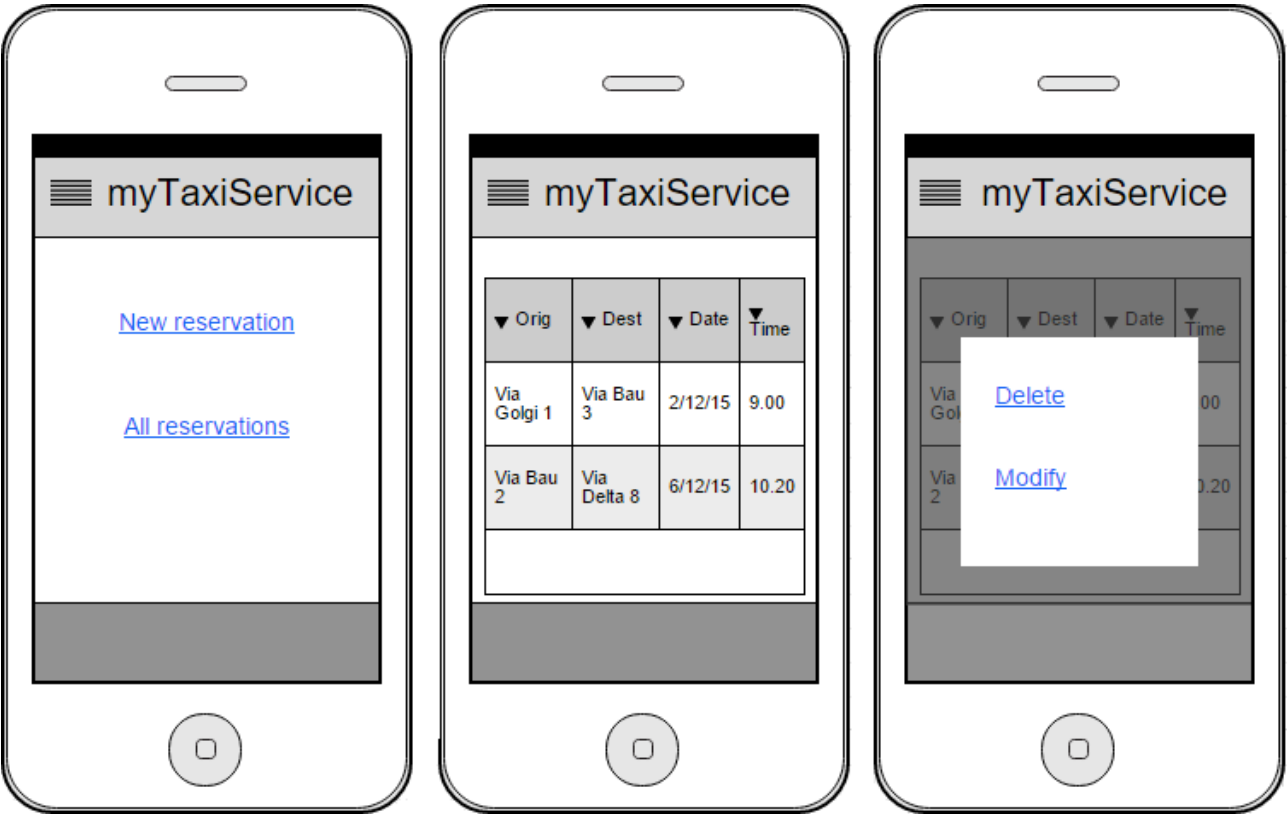
2016

▼

Meeting time

CONFIRM

The next four pictures show the same situations as before, but they're on the mobile application instead. As we can see, there are no major differences and passengers are allowed to choose whether they want to make a new reservation or view the ones they've already made, along with the possibility of deleting and modifying them.







In the next two pictures we can see what passengers can see when they want to use the payment feature. For example, if they've selected PayPal as a payment method, they will be able to select the ride they want to pay for, through their PayPal account they've linked to their myTaxiService account. Once they've selected a ride, a price will show up, and by clicking or touching "Confirm" they'll be able to pay and end the operation. Otherwise, if they want, they can pay using cash instead, without having to use the application.


[←](#) [→](#) [↻](#) [🏠](#)

[Home](#) [Request a taxi](#) [Reservations](#) [Payment](#) [Log out](#)




Mickey Mouse







michey.mouse@live.it



3358671922



▼



Paypall

Select the ride :  ▼

Price :

CONFIRM



In the left picture we can see what the taxi drivers can see when they want to change their status. The interface is very simple: when they're available, their only option is to change their status to unavailable if and when they want to, and when they're unavailable, their only option is to change their status to available.

The picture in the middle shows what the taxi drivers can see when they receive a call. Some details are shown, like the name of the passenger, the phone number and the address so that they know where they have to go to pick up their customer. At the bottom of the page they're shown an option, to decide whether they want to accept the call or not.

The right picture shows what the taxi drivers see when a reservation has been cancelled by a passenger. This is basically a message to notify the taxi driver that he/she doesn't have to take care of the call because the passenger has cancelled it.



## 5. Requirements traceability

The requirements that were defined in the RASD can be found in the design elements that are defined in this document.

For example, as we can see in the component view, the possibility to sign up is given by an interface called `SignUp`, which is a service offered by the `Controller` component. More features regarding the possibility to make, view, delete or modify a request are also shown in the component view and they're offered by the `ManageAccount` interface. This generic interface is divided into more specific interfaces, called `Create`, `Delete`, `View` and `Modify` which are offered by the `GenericRequest` component to the `Controller` component. Therefore, this shows how the requirements are mapped into the design elements and exactly which element offers a certain service.