myTaxiService
Design Document

Jacopo Strada Luca Riva

December 4, 2015

Version 1.0

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

This system design document describes the main design concerns and will have an important role in the development and in the future maintenance of the software itself. The document is addressed to the city government and in particular to its IT department, since many diagrams, architectures and patterns references will be found in the next sections.

1.1 Purpose

The purpose of this document is to explain as clearly as possible every factor that may have created some perplexity in the client.

1.2 Scope

In the document it is discussed the interaction between the system and the actors, at the same time it is possible to find a detailed description of the communication among various components of the system executing certain operations.

1.3 Glossary

1.3.1 Definitions

Client / Passenger / User : Is a person who signed up for this service and their interest is to call a taxi or reserve a ride.

Taxi Driver : Is a person who drives a taxi and would like to be called or reserved for a ride through this service.

1.3.2 Acronyms

API: Application Programming Interface

GPS: Global Positioning System

GUI: Graphic User Interface

HTTPS: Hyper Text Transfer Protocol over Secure Socket Layer

IEEE: Institute of Electrical and Electronics Engineers

IT: Information Technology

RASD: Requirements and Specification Document

UML: Unified Modeling Language

1.3.3 Abbreviations

G*n*: Goal number n

R*n*: Requirement number n

1.4 Reference Documents

- *myTaxiDriver* Specification Document
- *myTaxiDriver* Requirements and Specification Document

1.5 Document Structure

In the second chapter of the document the architectural design of the system is explained by using UML diagrams and a graphic representation of the paradigms employed.

In the third paragraph it is possible to find the pseudo-code of the queue management.

In the fifth paragraph is demonstrated how all the requirements presented in the RASD are effectively satisfied by the system, showing which component is responsible for every specific task.

2 Architectural Design

2.1 Overview

myTaxiService system is, by its nature, a distributed application. In this section we're going to present the hardware and the software components which compose our system and how they interact between themselves.

2.2 High level components and their interaction

Looking at the system in a high-level view, it can be divided in a client side and in a server side.

The **client side** is composed by a web browser or by a mobile application and represent the interface used by users to interact with the system. These clients are all *thin*.

The **server side** is made by a lot of components which can be divided between application logic and data storing. The application logic exposes an API for each component in order to allow the communication with our applications (or applications developed by others) throw the network.

In the following sections we present all these components with the help of UML.

2.3 Component view

Here we're going to present all the components of *myTaxiService* system and their interfaces:

Data Base: This component is used to store all the information about users, zones, queues and drivers positions. An interface for each type of information stored is offered by this component.

Account Manager: All the users are managed by this component. It is responsible for logins and registrations. It has to check all the constraint (user age, driver license, ...) when a user would like to sign up. It is used also for changing taxi drivers state.

Call Manager: Every taxi call is sent to this component. It has to handle the whole process that will bring to the assignment of a call to a specific driver. In order to do this it has to interact with a lot of others components: first of all using the Position Utility the system has to get the correct zone from the client position, then the Queue Manager has to return an available driver in the current zone and finally with the Notification Manager a notification is sent to the client in order to confirm the request.

Queue Manager: This component is in charge of managing the driver's queues, which are stored in the system's database. Its job is to select the right queue for a given zone and return the first driver in the queue, in case of rejection by the first driver it moves them to the last position in the queue: see Algorithm 1 for further details.

Reservation Manager: The reservation manager receives requests for a reservation and checks if it is valid (time and route using the Position Utilities). If a reservation is valid and a taxi driver accepts it, this component will store the reservation in the database.

Notification Manager: The notification manager is employed in order to communicate with drivers and clients, for example when the system receives a call requests and it needs to forward it to an available driver or when a reservation is accepted and the system has to notify the client.

Position Utilities: The Position Utilities is used in order to retrieve a zone from ad address, calculate an extimated time for a call or validate a path for a reservation.

The following diagram represent all the components of our system and how they are connected together.

The two components *myTaxiService MobileApp* and *myTaxiService WebSite* are part of the client side, all the others belog to the server side.

All the available interfaces are also represented.

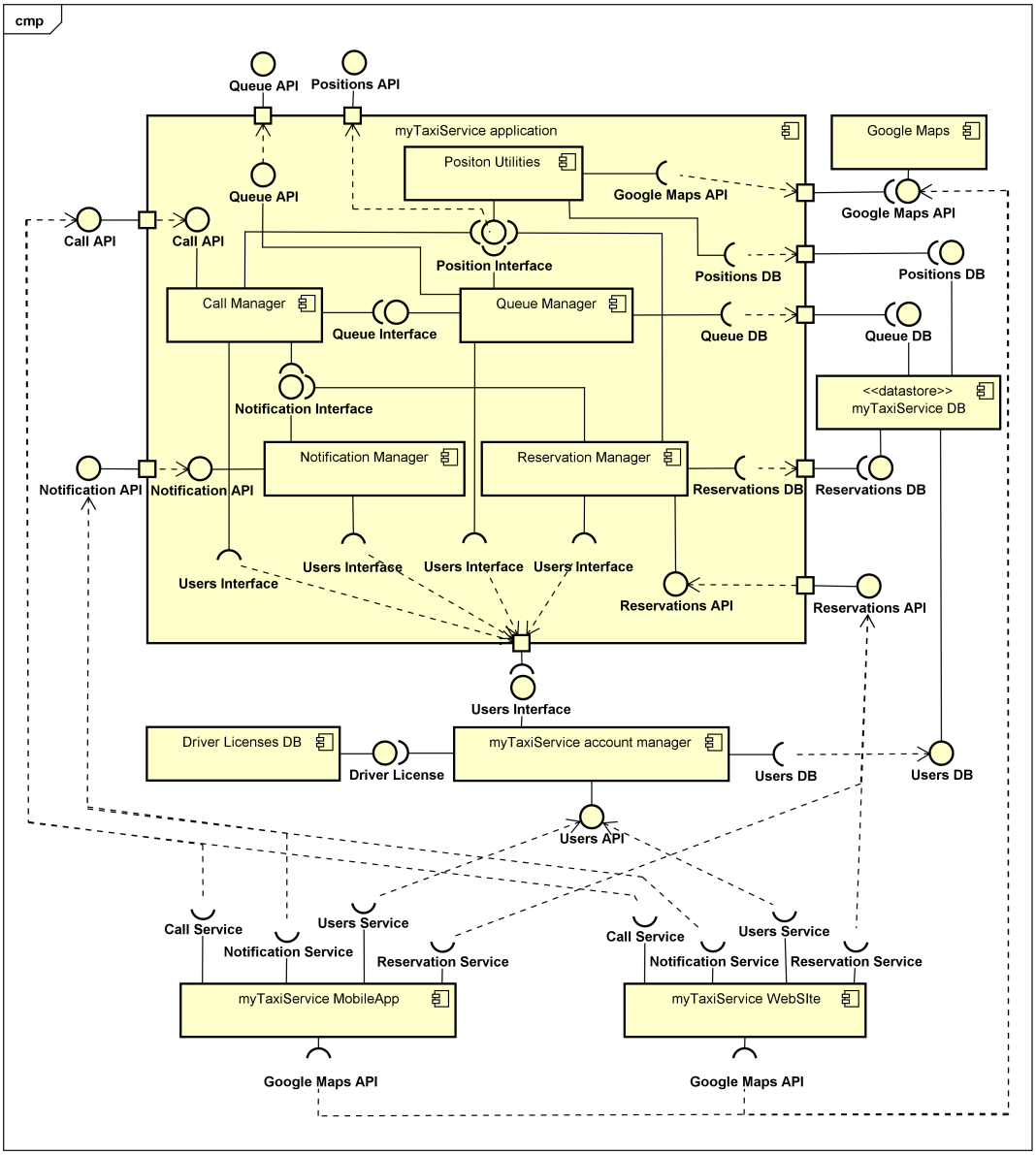

Figure 1: UML Component Diagram

2.4 Deployment view

The following diagram represent the hardware components of the system. For each piece of hardware is also shown what part of software runs on it referring to the components presented in the previous section.

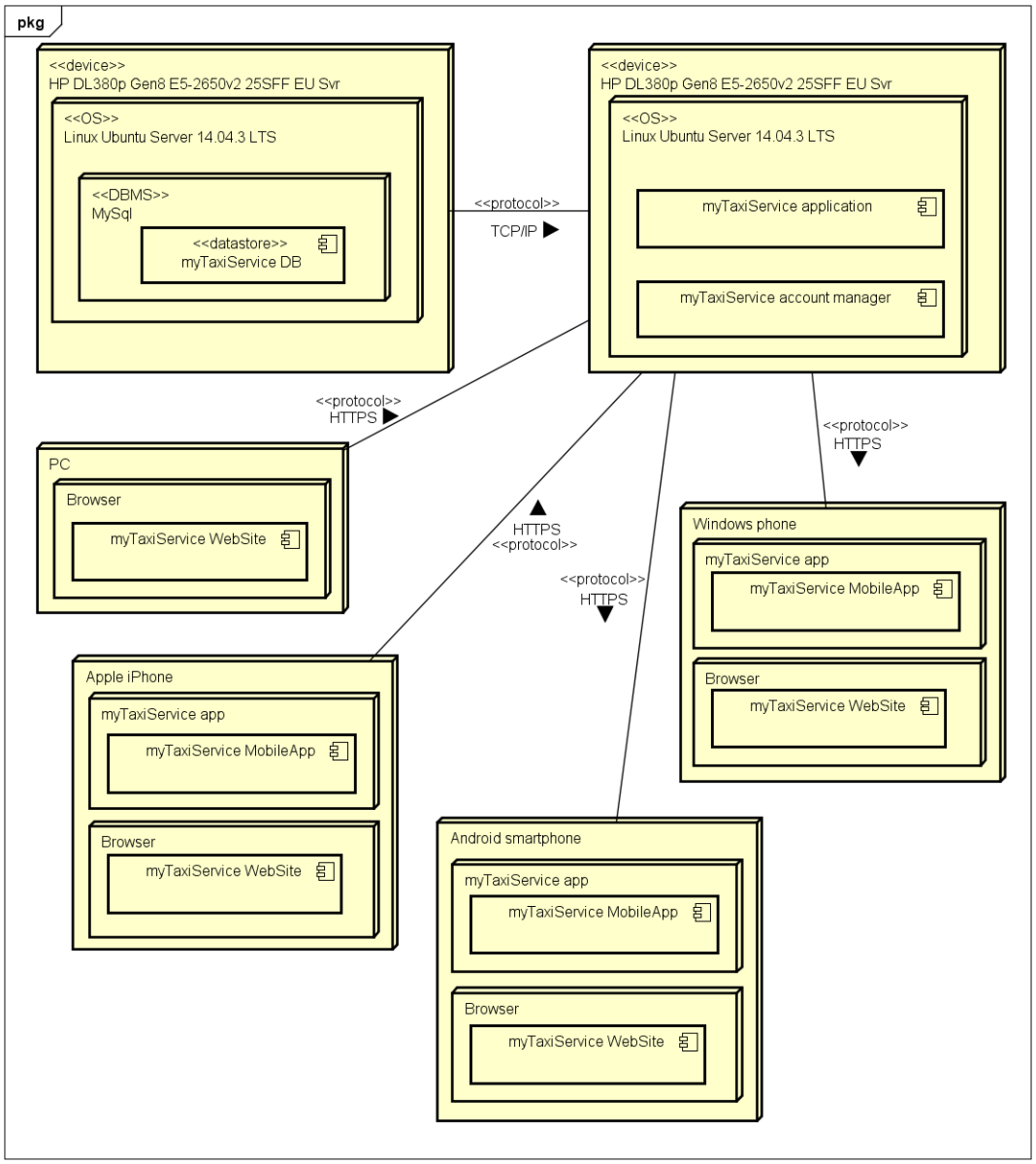

Figure 2: UML Deployment Diagram

2.5 Runtime view

In this section, with some sequence diagrams, are shown the most important interactions between components.

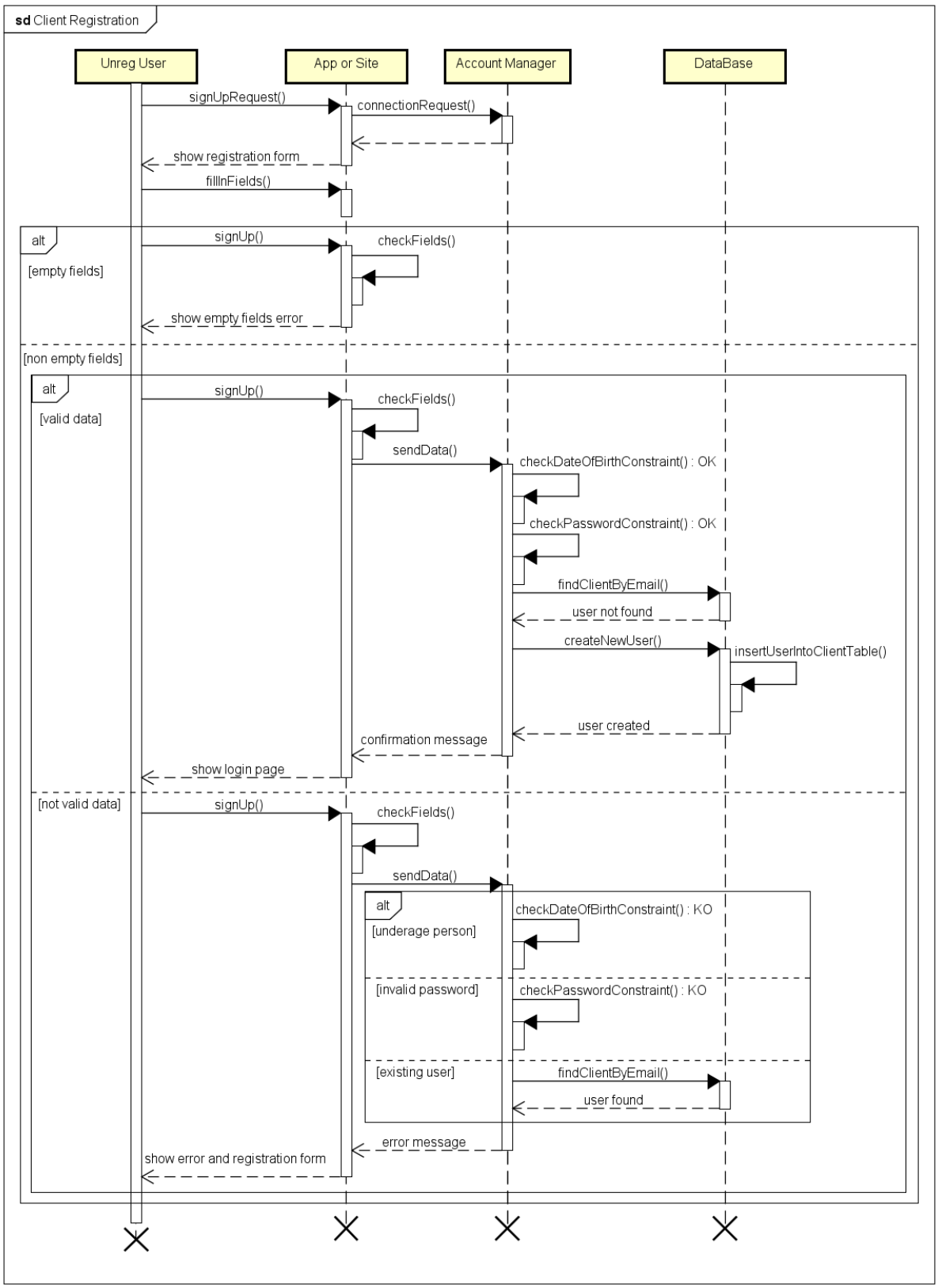

Figure 3: Client Registration UML Sequence Diagram

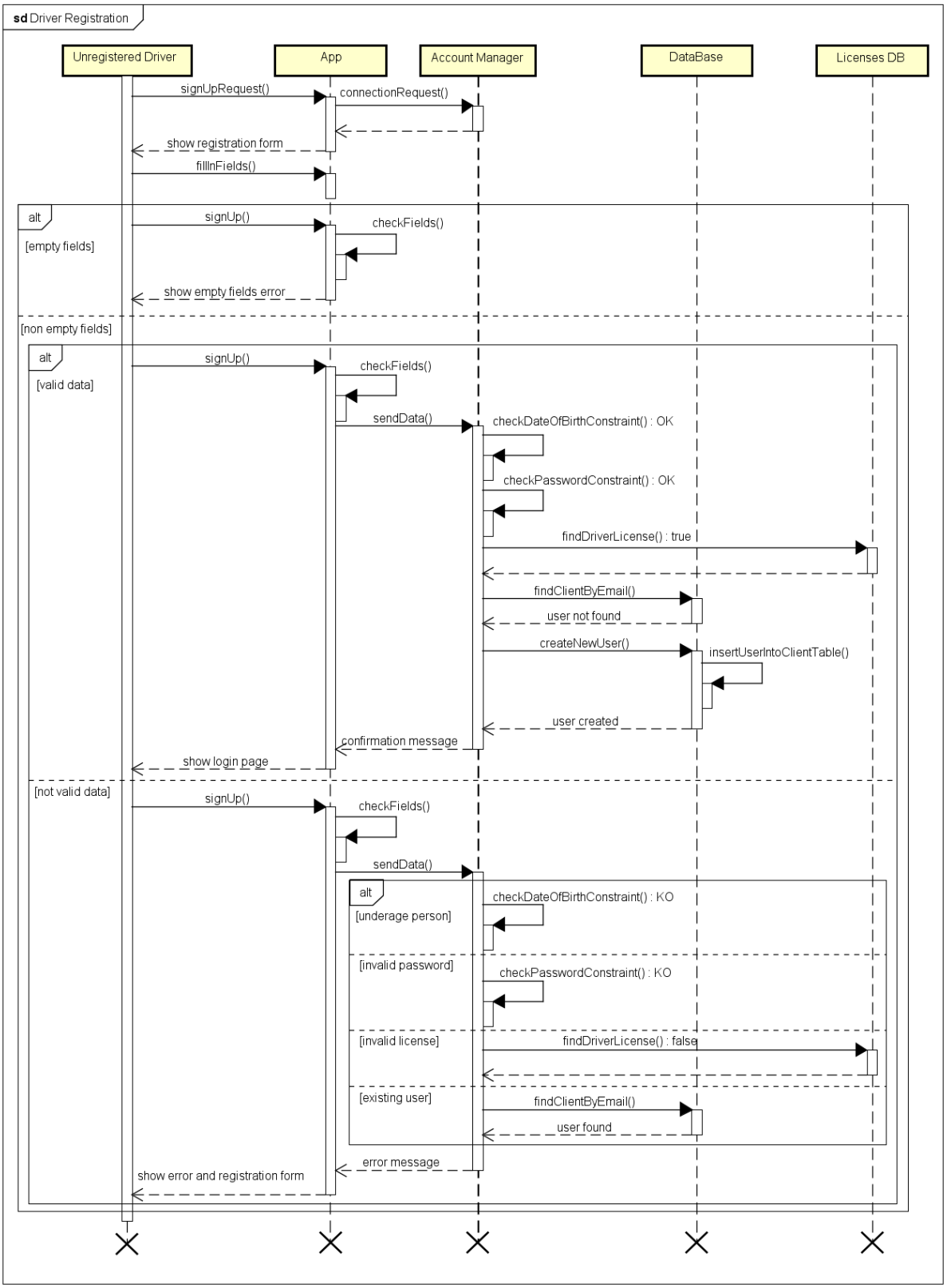

Figure 4: Taxi Driver Registration UML Sequence Diagram

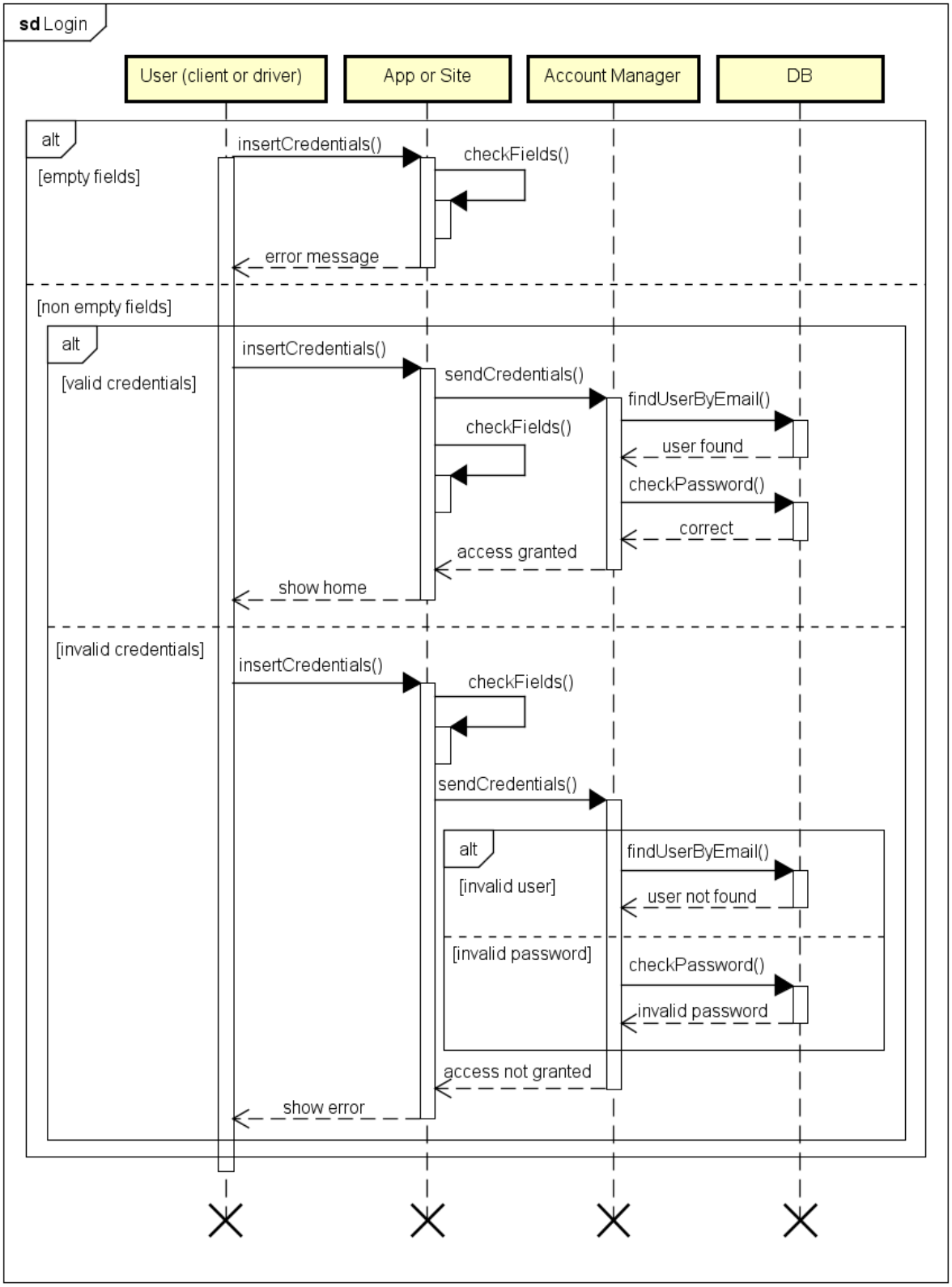

Figure 5: Login UML Sequence Diagram

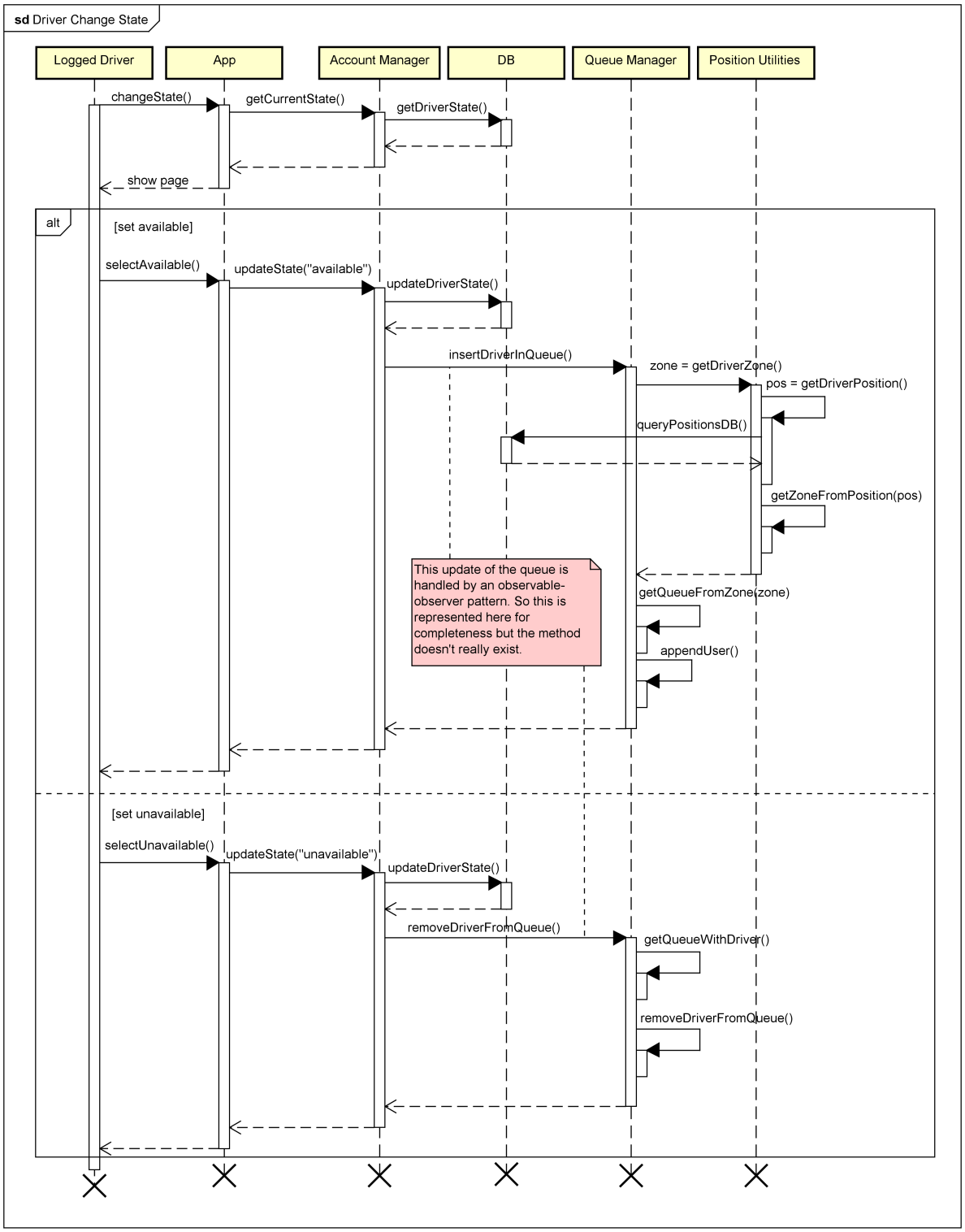

Figure 6: Taxi Driver Changes State UML Sequence Diagram

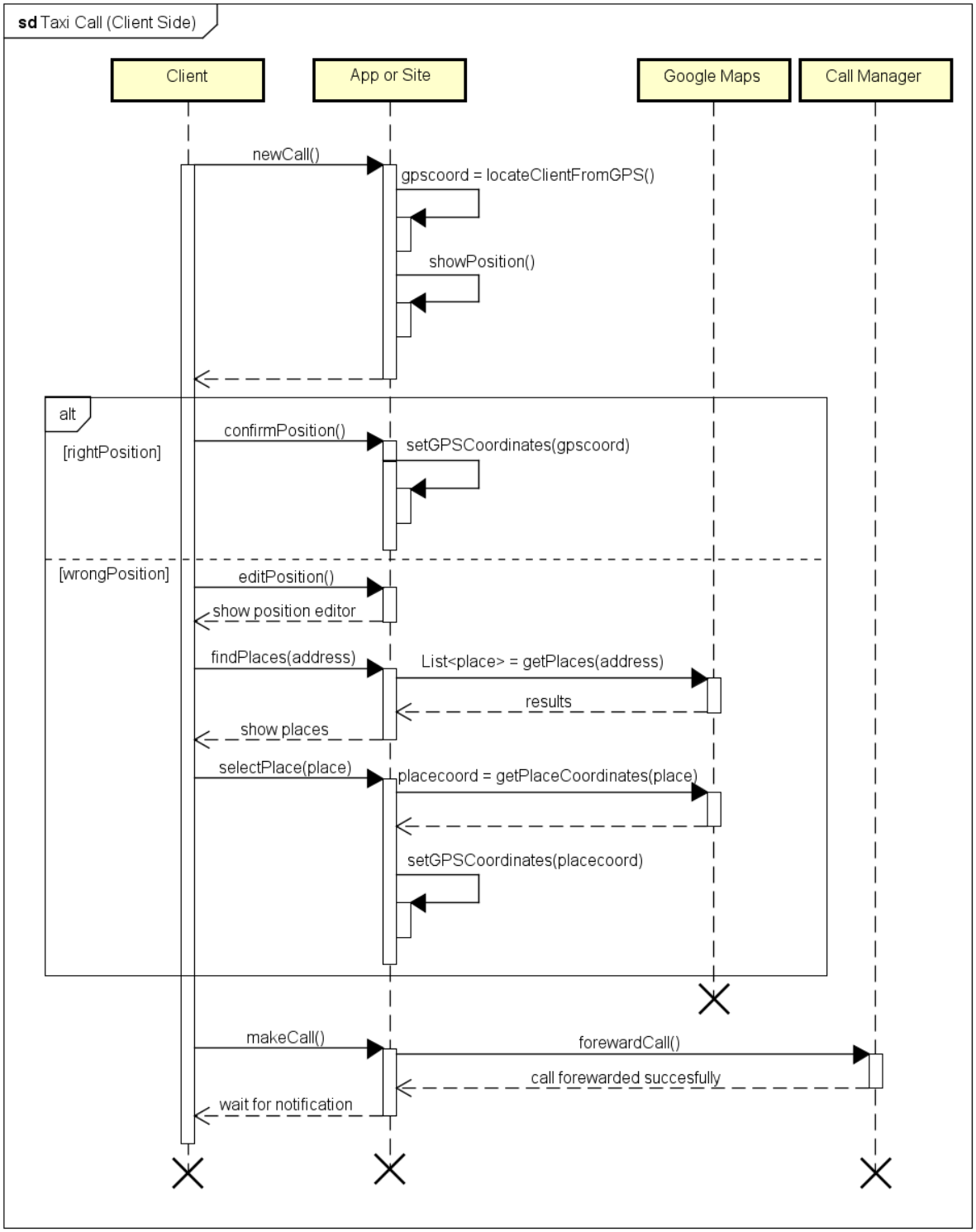

Figure 7: Taxi Call Client Side UML Sequence Diagram

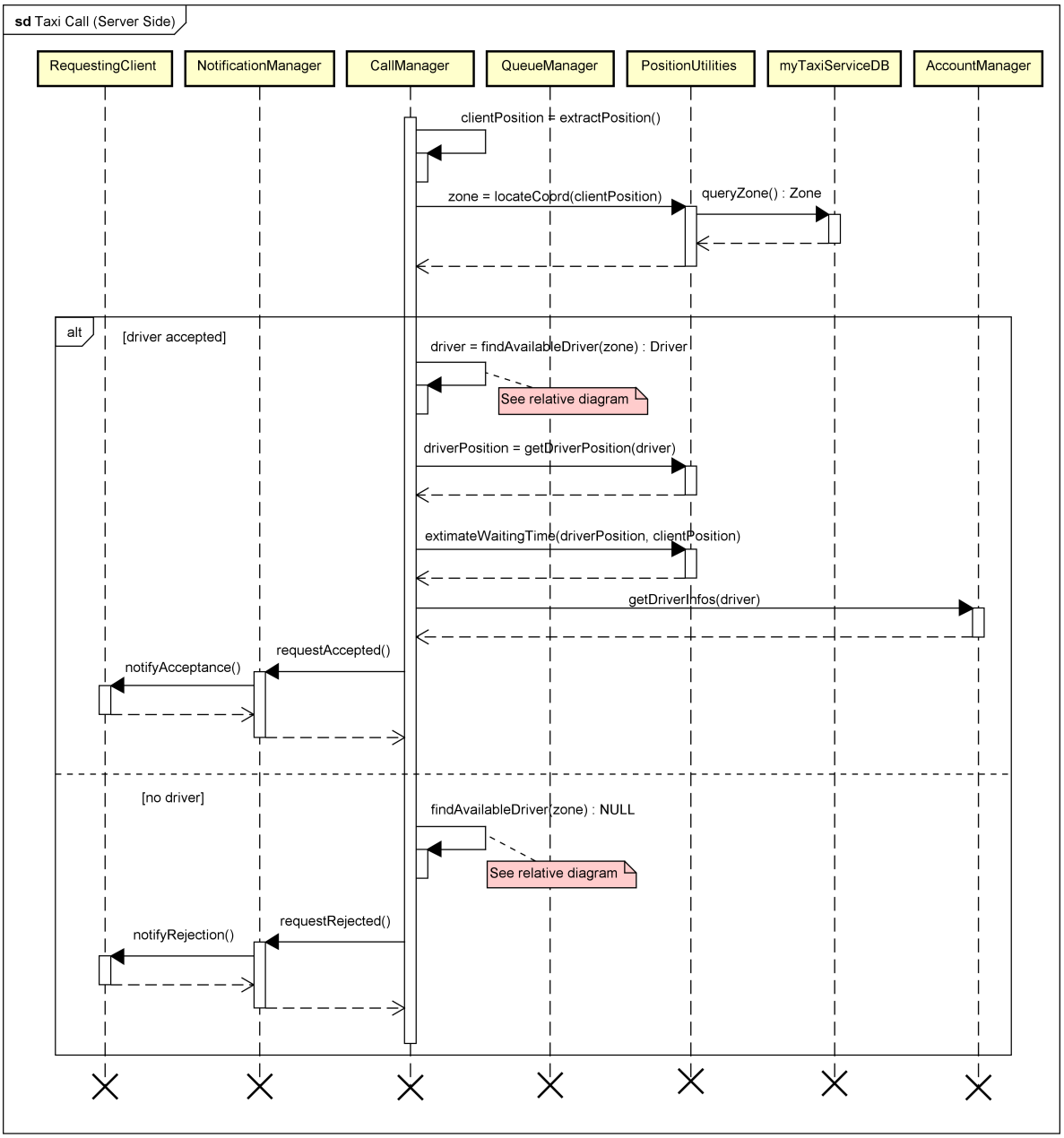

Figure 8: Taxi Call Server Side UML Sequence Diagram:
See Figure 9 in order to understand what happens in the *findAvailableDriver()* method.

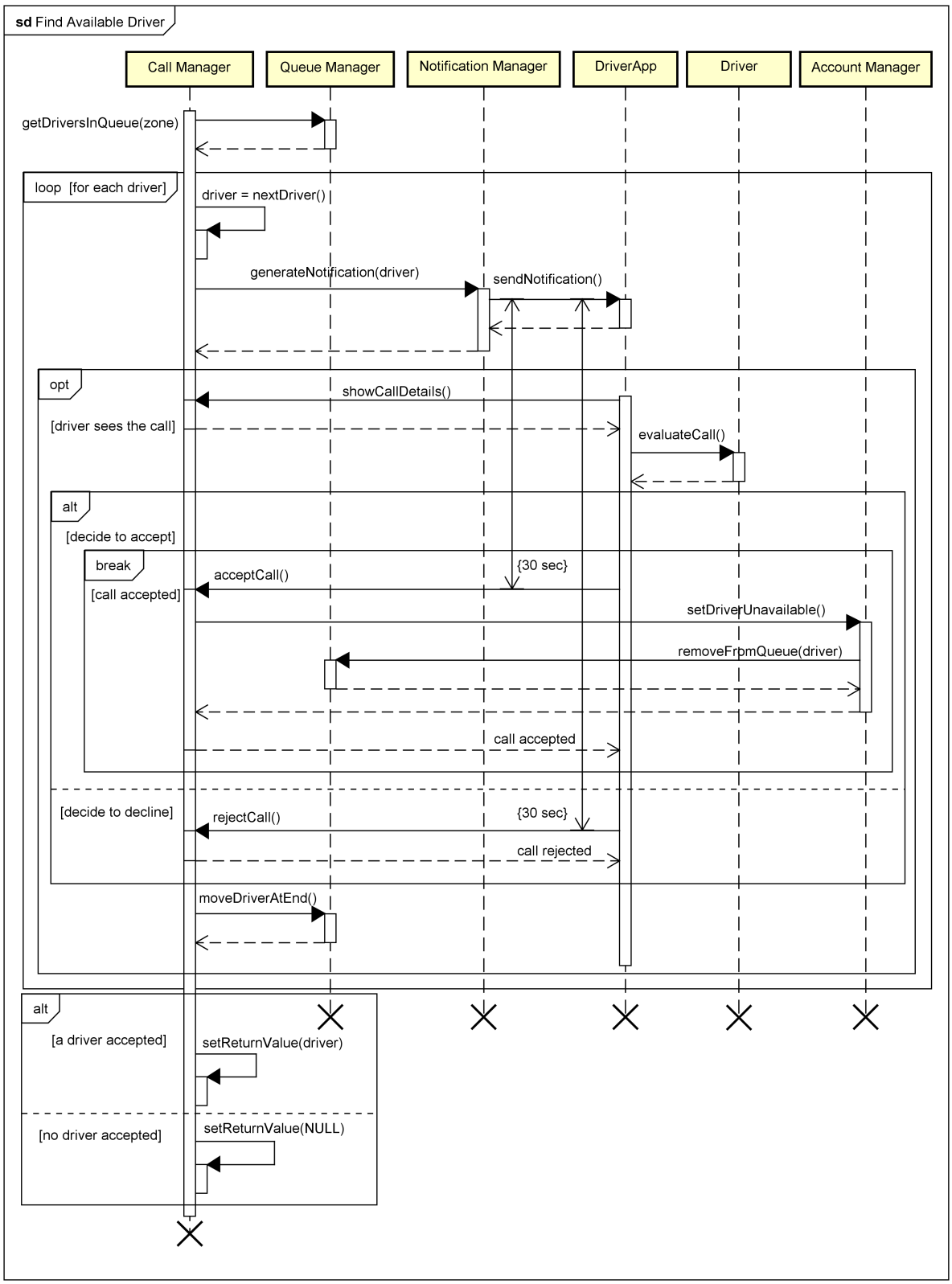

Figure 9: Find Available Driver UML Sequence Diagram:
For other details relative to this diagram you can see also Algorithm 1

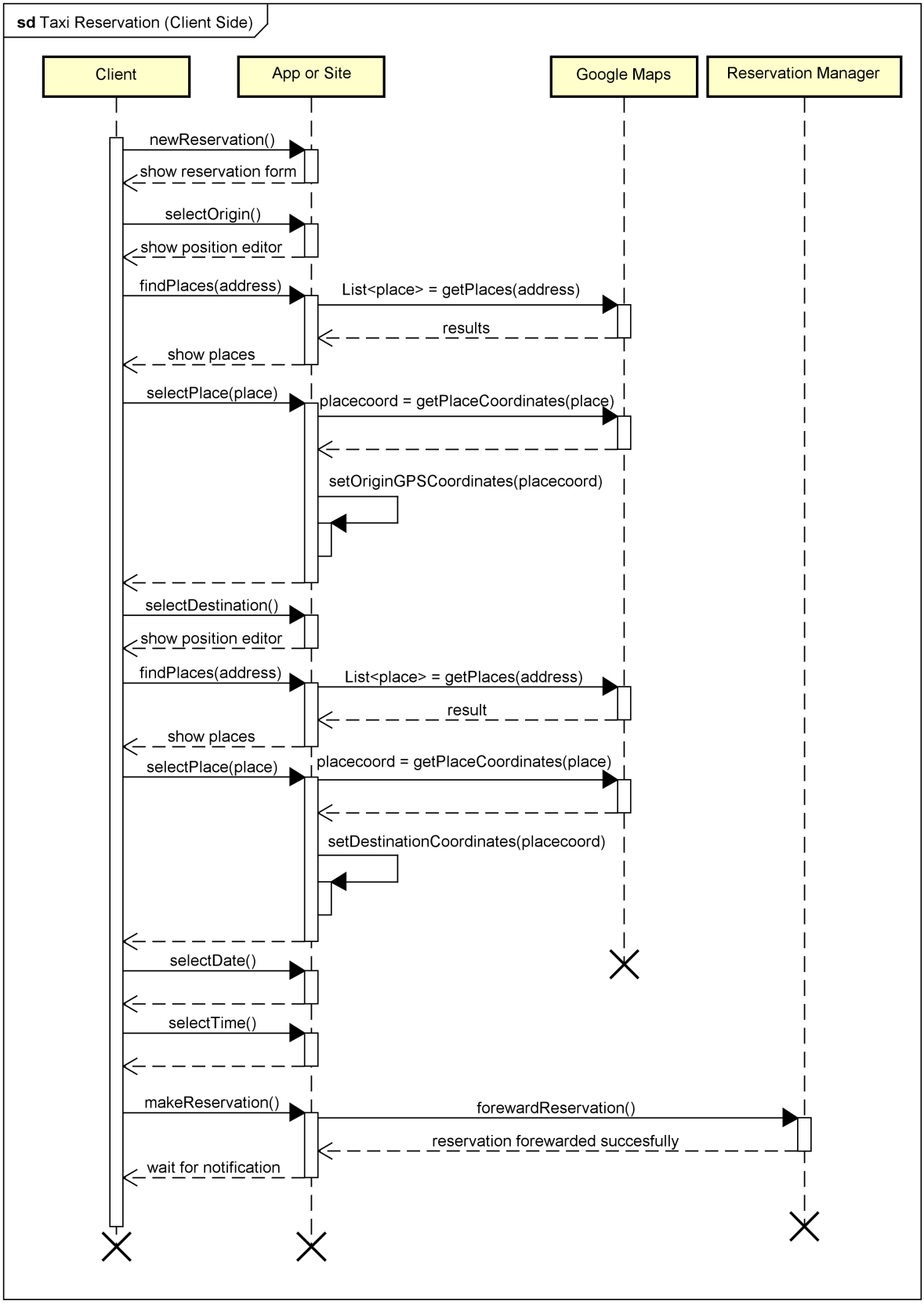

Figure 10: Taxi Reservation Client Side UML Sequence Diagram

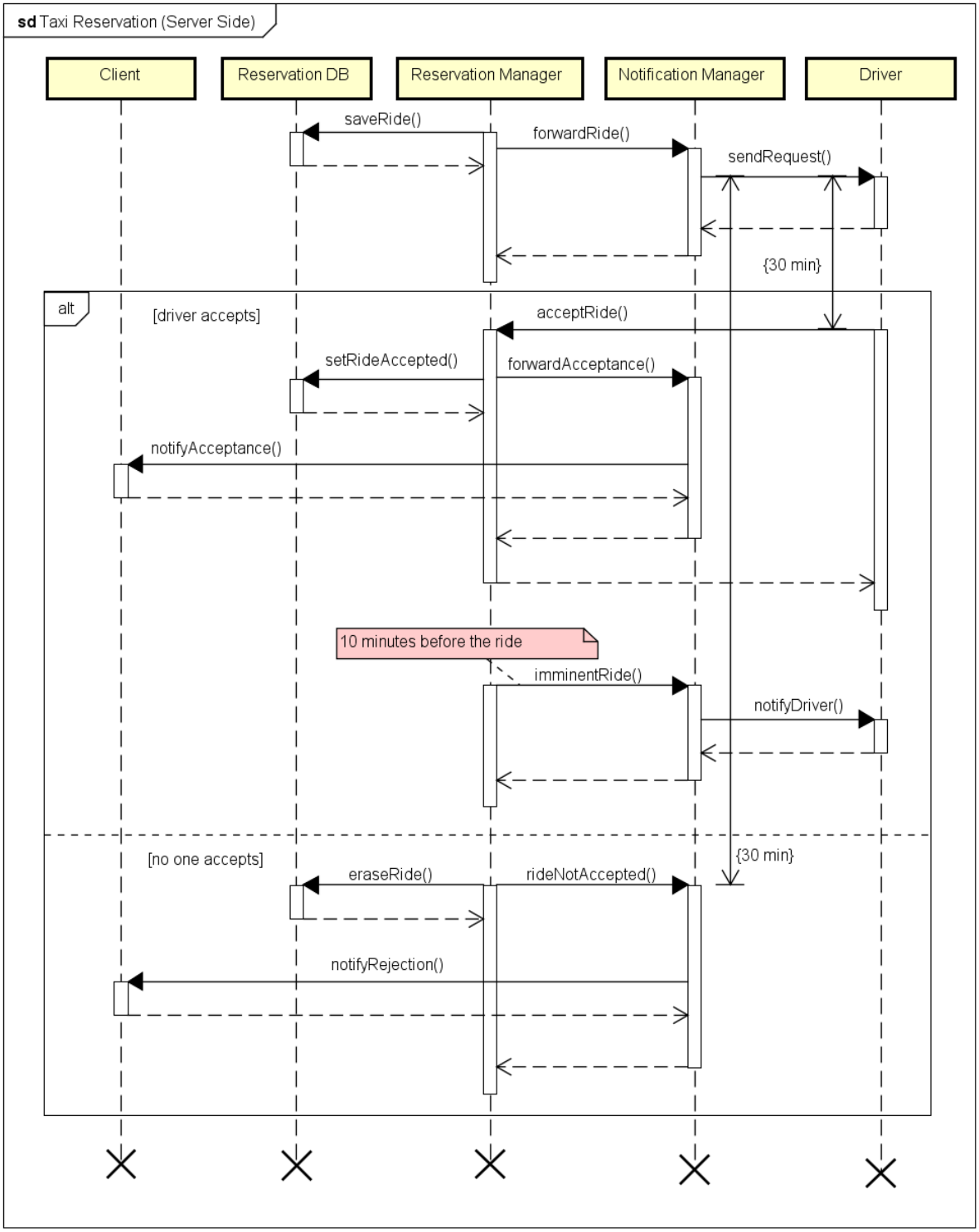

Figure 11: Taxi Reservation Server Side UML Sequence Diagram

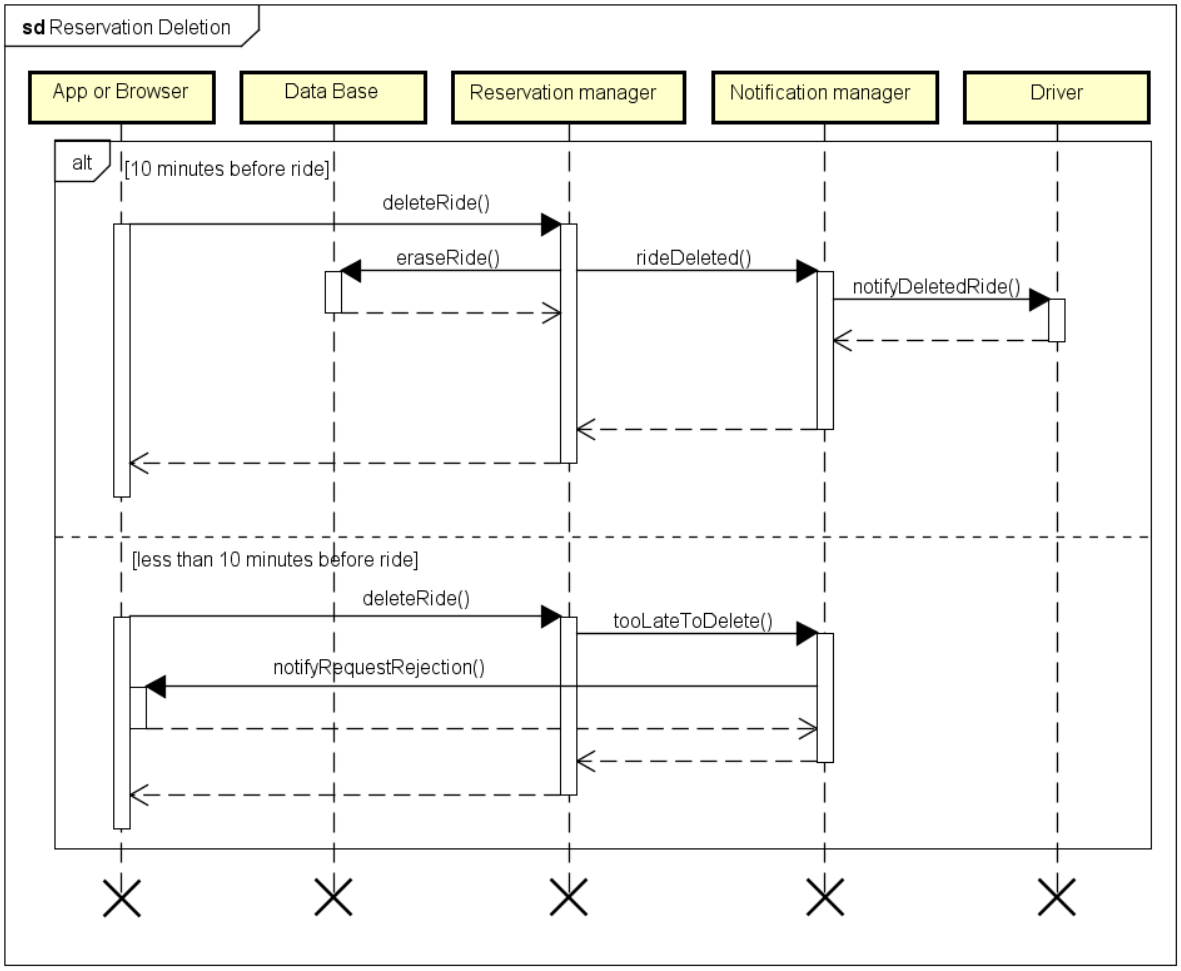

Figure 12: Taxi Reservation Deletion UML Sequence Diagram

2.6 Component Interfaces

Data Base: (The following methods are not available as APIs but are only private)

- *createNewClient(email, name, surname, dateOfBirth, phone, password)*
- *createNewDriver(email, name, surname, license, phone, password)*
- *checkPassword(userId, password)*
- *findUserByEMail(email)*
- *getDriverState(email)*
- *updateDriverState(email, state)*
- *getZones()*
- *saveRide(origin, destination, date, time)*
- *setRideAccepted(rideId, driver)*
- *eraseRide(rideId)*

Account Manager:

- *login(mail, password)*
- *registerClient(email, firstName, lastName, telephoneNumber, password)*
- *registerDriver(email, firstName, lastName, telephoneNumber, license, password)*
- To receive information about an user: *clientInfo(email)* or *drievrInfo(email)*
- To change a driver state when they are taking a call: *changeDriverState(email, state)*

Call Manager:

- To call a taxi: *forwardCall(email, GPSPosition)*
- To show the details of a call (should be used by a driver): *showCallDetails(call)*

Queue Manager:

- To retrieve the drivers' queue of a certain zone: *driversQueue(zone)*

Reservation Manager:

- *newReservation(email, origin, destination, time)*
- to show the details of a reservation: *showReservationDetails(reservation)*

Notification Manager:

- *sendNotification(list<email >, message)*

Position Utilities:

- to find the zone associated to a given coordinate: *findZone(GPSCoordinate)*

2.7 Selected Architectural Styles and Patterns

For this service was selected a *three-tier* architecture: Data, Application Logic and GUI are separated and there are two levels of firewalls in order to keep a high level of security.

In Figure 13 you can see a graphical representation of this architecture.

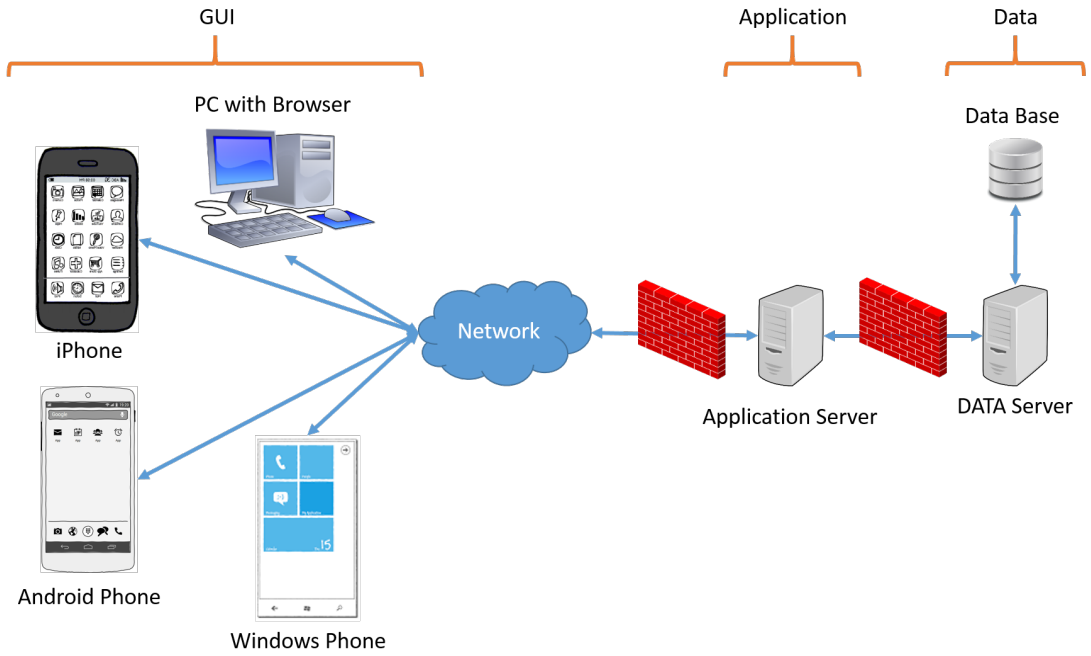

Figure 13: Architecture Representation

About the paradigm we decided to combine two important patterns:

- the **client-server**
- the **publisher-subscriber**

The client server is used for all the communications which are composed by a request, made by the client, and a response, given by the server.

The publisher-subscriber is needed for the notification service.

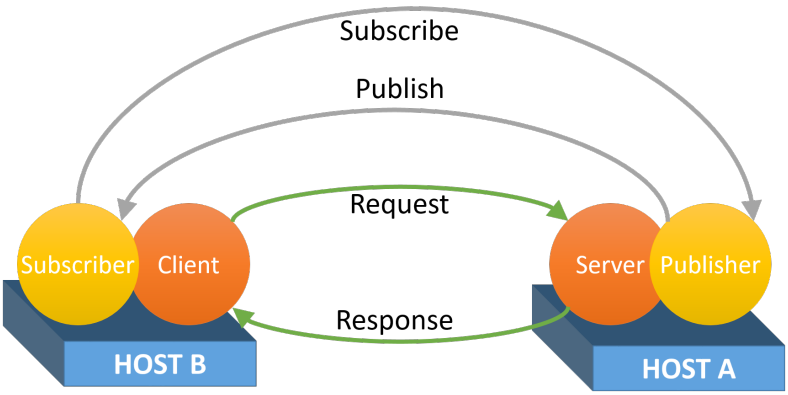

Figure 14: Paradigm Representation

3 Algorithm Design

myTaxiService is not a very algorithmic system but there is a little part that is worth describing. This part is the managing of the queue when a call arrives and is presented in the following algorithm.

Algorithm 1: Find Available Driver

Require: the queue of a zone
Ensure: the return value is the driver who accepted the ride or NULL if nobody accepted
  for all drivers in queue (only once) **do**
    send a notification to the driver
    wait for 30 seconds (or less if a response is received)
    if the driver has accepted **then**
      remove driver from the queue
      set driver state as *unavailable*
      send a notification to the client
      return current driver
    else if the driver has rejected **or** the driver hasn't responded **then**
      move the driver from the begin to the end of the queue
    end if
  end for
  return NULL

4 User Interface Design

A complete description of how the user interfaces of our system will look like was already included in our *Requirements and Specification Document* document. You can find it in Section 3.3 *External Interfaces* of the specified document.

5 Requirements traceability

Requirement		***Designated system element***
G1	**R1**	Validation made by the *account manager*
	R2	
	R3	
	R4	The *account manager* checks if the e-mail address is already inside the *Users DB*
	R5	The *account manager* checks if the e-mail address is already inside the *Users DB* and the app or browser shows only the login page and the registration form
G2	**R1**	Validation made by the *account manager*
	R2	
	R3	
	R4	The *account manager* checks if the driver license is inside the *taxi drivers DB*
	R5	The *account manager* checks if the e-mail address is already inside the *Users DB*
	R6	The *account manager* checks if the e-mail address is already inside the *Users DB* and the app or browser shows only the login page and the registration form
G3	**R1**	Validation made by the *account manager*
	R2	
	R3	
	R4	
G4	**R1**	Validation made by the *account manager*
	R2	
	R3	
G5	**R1**	Validation made by the *account manager*
	R2	*App* or *browser* running on a device with enabled GPS and connected to the internet
	R3	
	R4	
G6	**R1**	Validation made by the *account manager*
	R2	The *Queue manager* receives a positive answer from one of the taxi
G7	**R1**	Validation made by the *account manager*
	R2	The *Queue manager* receives a positive answer from one of the taxi
G8	**R1**	Validation made by the *account manager*
	R2	Time out for the *Queue manager* which was waiting for a positive answer
G9	**R1**	Validation made by the *account manager*
	R2	*Driver app* running on their smartphone
	R3	*Position utilities* stored driver position in the *positions' DB*
G10	**R1**	Validation made by the *account manager*

Table 1: Requirements traceability part 1

Requirement		***Designated system element***
G11	**R1**	Validation made by the *account manager*
	R2	*Account manager* checks the *users DB*
	R3	*Queue manager* works with the *position utilities* to select the right queue in the *queue DB*
	R4	*Notification manager* sends a notification to the driver app
G12	**R1**	Validation made by the *account manager*
	R2	*Account manager* checks the *users DB*
	R3	*Queue manager* updates driver's state working with the *account manager*
	R3	*Queue manager* works with the *position utilities* to select the right queue in the *queue DB*
G13	**R1**	Zones are saved in the *myTaxiServiceDB*
	R2	*Position utilities* locate drivers in a zone, *queue manager* updates the zone's queue, the *account manager* checks driver's state in the *users DB*
	R3	
	R4	
G14	**R1**	The *call manager receives the taxi call*
	R2	*Queue manager* works with the *position utilities* to locate drivers in the zone, then the *notification manager* sends the request to the first driver in the queue
	R3	The *queue manager* receives a positive answer, then the *notification manager* sends a notification to the client
	R4	The *queue manager* wait for 30 seconds an answer from the first client then selects the second driver and asks to the *notification manager* to send a message to them
	R5	The *queue manager* updates the *queue DB* positioning the driver at the bottom of their zone's queue
G15	**R1**	Validation made by the *account manager*
	R2	Validation made by the *reservation manager*
	R3	
G16	**R1**	The *reservation manager* receives the client's reservation
	R2	The *reservation manager* retrieves the driver's accounts from the *account manager*, then asks the *notification manager* to send the reservation details
	R3	The *reservation manager* waits an answer for 30 minutes, then it interrupts the reservation process
	R4	
	R5	If the *reservation manager* receives a confirmation it asks the *notification manager* to notify the client
	R6	The *reservation manager* asks the *notification manager* to notify the client of the rejection of the reservation if 30 minutes have passed and no driver answered positively
	R7	The *reservation manager* asks the *notification manager* to notify the drivers 10 minutes before the time of the reservation they accepted
G17	**R1**	The *account manager* updates the position and the state of the driver's in the *users DB*

Table 2: Requirements traceability part 2

6 Appendix

6.1 Software and Tools used

ShareLatex: This web application was used to redact this document in a collaborative way. (`https://it.sharelatex.com/`)

Astah Professional: This desktop application was used to create all the others UML Diagrams. (`http://astah.net/`)

6.2 Hours of Work

We spent approximately the following amount of hours to redact this document:

Riva Luca: 29

Strada Jacopo: 29