Politecnico di Milano
School of Industrial and Information Engineering
Master Course in Computer Science and Engineering
DEIB Department

Design **D**ocument (DD)

4th December 2015

Moreno SARDELLA - 859239

Academic Year 2015–2016

Contents

List of Tables

List of Figures

Introduction

0.1 Revision History

Table 1: Detailed history of the document revisions

Name	Date	Note
DD-version4	28/02/2015	- *Request a taxi* sequence diagram fixed - *Start a ride* sequence diagram fixed - Interfaces fixed
DD-version3	31/12/2015	Document completed: - Algorithm Design added - Requirements Traceability added
DD-version2	09/12/2015	Architectural Design added
DD	04/12/2015	Document creation

0.2 Purpose

This document represents the Design Document (DD), describing the high level aspects of the MyTaxiService project design (according to the RASD document), in order to guide the development phase.

0.3 Scope

The Design Document shows the main aspects of the MyTaxiService project, in terms of components and sub-components. They are described both from the software and hardware points of view. It is also shown how they interact each other.

0.4 Definitions, acronyms and abbreviations

0.4.1 Definitions

- **Login**: the procedure through which a guest, entering his/her credentials, authenticates as a user.

- **Logout**: the procedure though which a user disconnects himself/herself from the system.
- **System:** the whole MyTaxiService service.

0.4.2 Acronyms

- **DD:** Design Document
- **ETA**: Estimated Time of Arrival
- **GUI**: Graphic User Interface
- **MVC**: Model View Controller
- **OS**: Operating System
- **RASD**: Requirements And Specications Document
- **UML**: Unied Modeling Language
- **WRT**: With Respect To

0.4.3 Abbreviations

- **ID**: Identier
- **UX**: User Experience

0.5 Reference documents

- [1] Assignments 1 and 2
- [2] Software Architectures and Styles
- [3] Design Patterns - Elements of Reusable Object-Oriented Software - GoF
- [4] Map Representation
- [5] UML - Enrico Amedeo - APOGEO

0.6 Overview

This document is composed by four part:

1. Architectural Design: high level informations about the product with more focus about the logical functions of components, how they are mapped into physical parts and how their interfaces are made. The use cases defined in RASD are rediscribed in a more low level through sequence diagrams.

2. Algorithm Design: definition of the most relevant algorithmic part of the MyTaxiService project

3. User Interface Design: app and web GUIs and releated user experience.

4. Requirements Traceability: mapping of the requirements defined in RASD into previous designed components.

Chapter 1

Architectural Design

1.1 Overview

This chapter describes the system, from both physical and logical point of view. First of all, a high level view shows its components, which are described in a more deeper level in section 1.3. Then, there is a deployment diagram of the physical tiers (section 1.4), followed by the runtime system behaviour (section 1.5). The section 1.6 aims to make clear how the the components interact each other, dening their interfaces. Finally, section 1.7 summarizes the implemented architectural patterns and styles.

1.2 High level components and their interaction

There are 3 main tiers that represent a high level overview of the myTaxiService components.

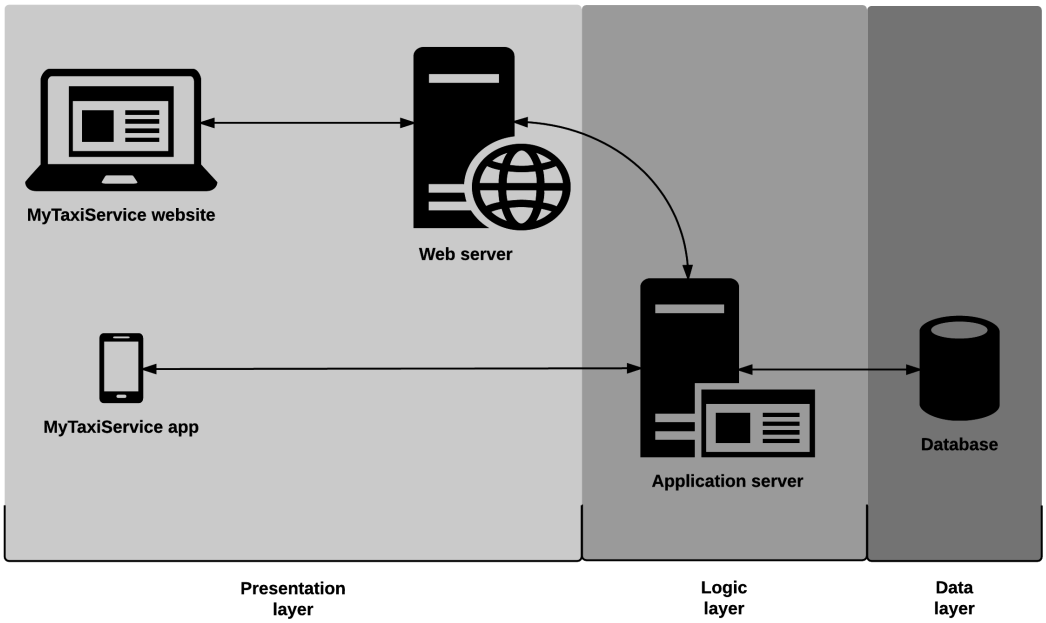


Figure 1.1: High level layers of MyTaxiService

Presentation layer: it is the front-end side of the system, dealing with the direct interaction with the user, who can use the service in two ways:

- via app, which communicates directly with the application server
- via web browser, which communicates with the web server, which provides the dynamic web pages querying the application server.

Logic layer: it is composed by the application server which cares about the whole business logic of MyTaxiService, handling all the operations between users and database.
Data layer: it is the database of MyTaxiService, w.r.t. the class diagram reported in RASD.

1.3 Component view

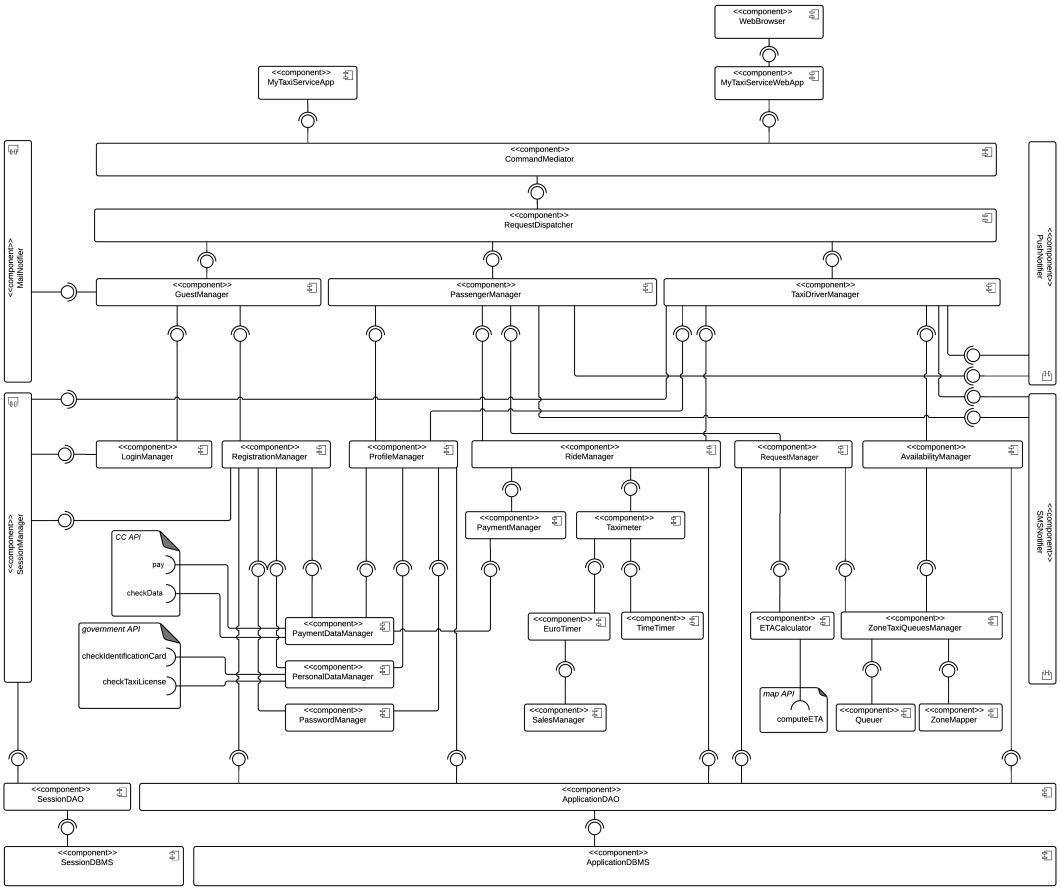


Figure 1.2: Components of MyTaxiService from a software point of view

1.4 Deployment view

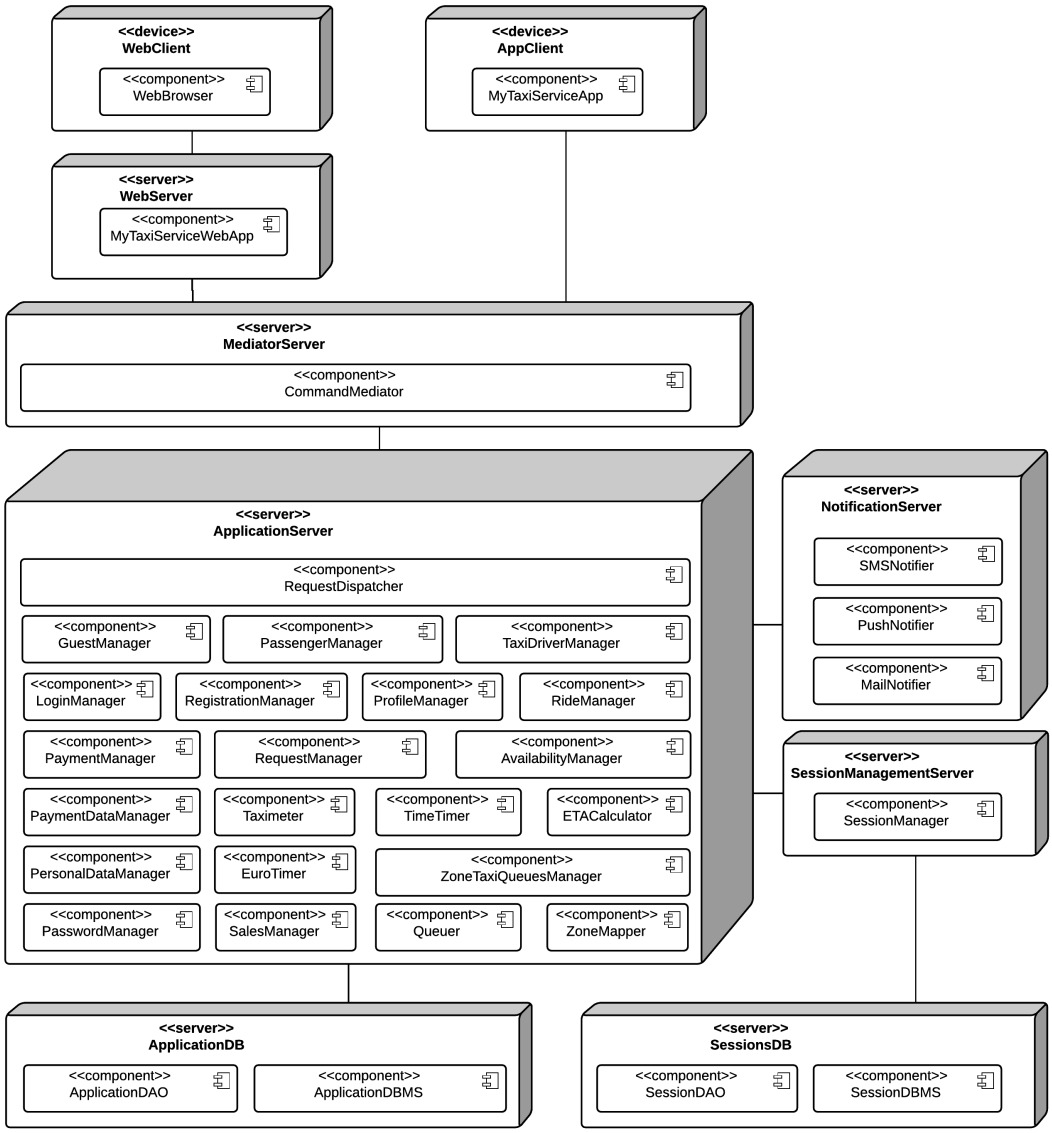


Figure 1.3: Mapping between software and hardware components of MyTaxiService

1.5 Runtime view

Starting from the use case diagram already shown in the RASD document, here are the main use cases explained in terms of detailed interactions between the components described below:

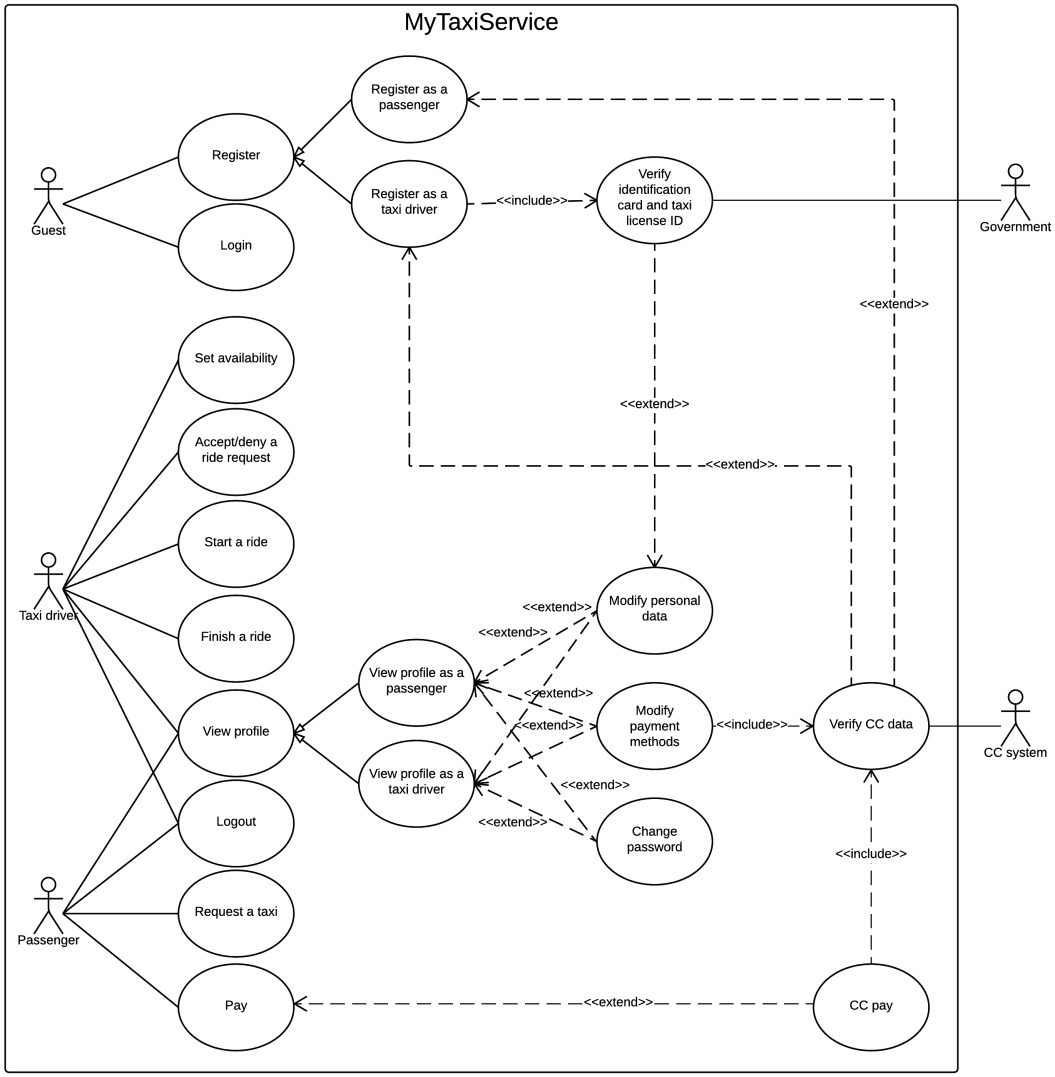


Figure 1.4: Use cases in MyTaxiService

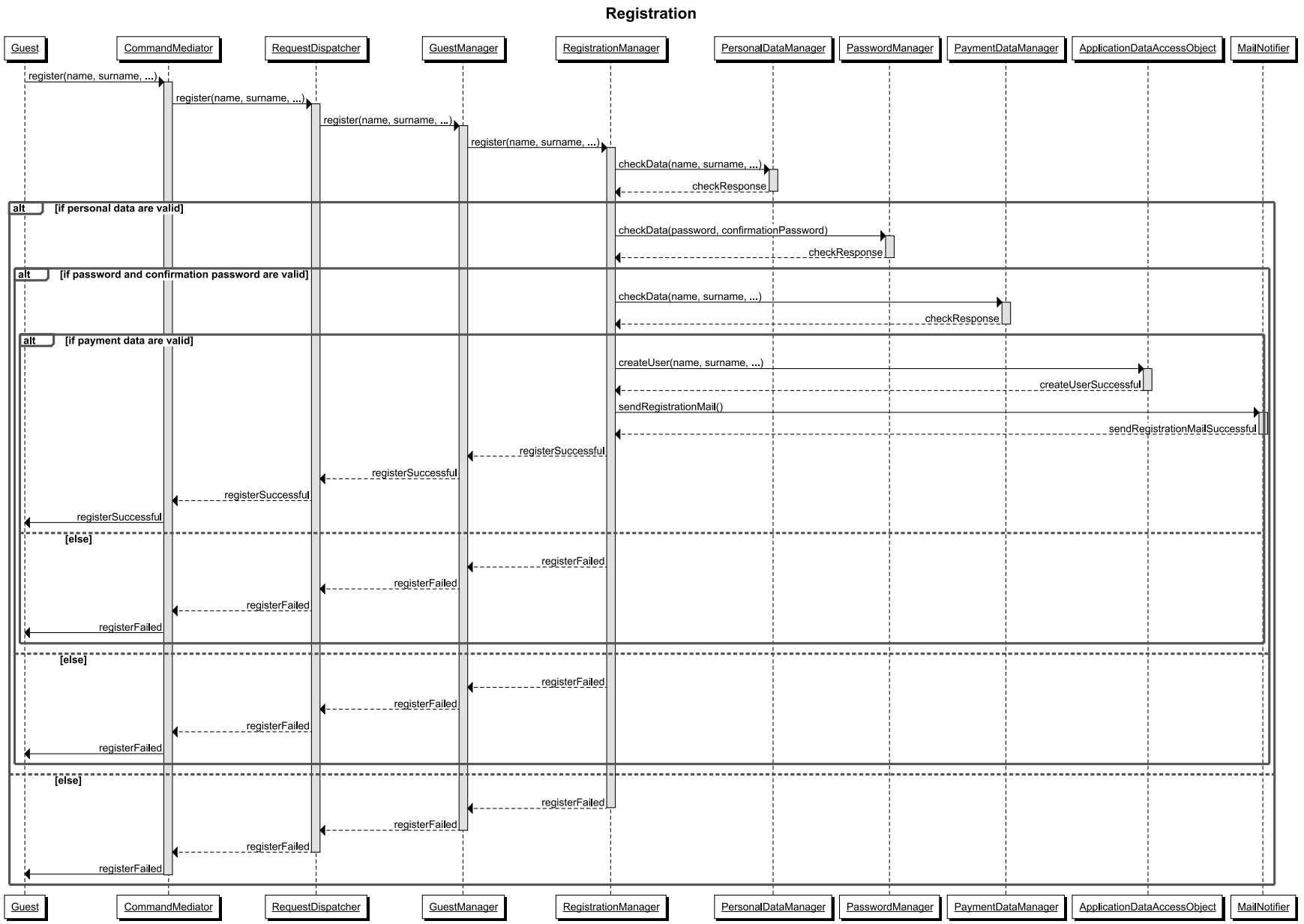


Figure 1.5: Components interactions during registration in MyTaxiService

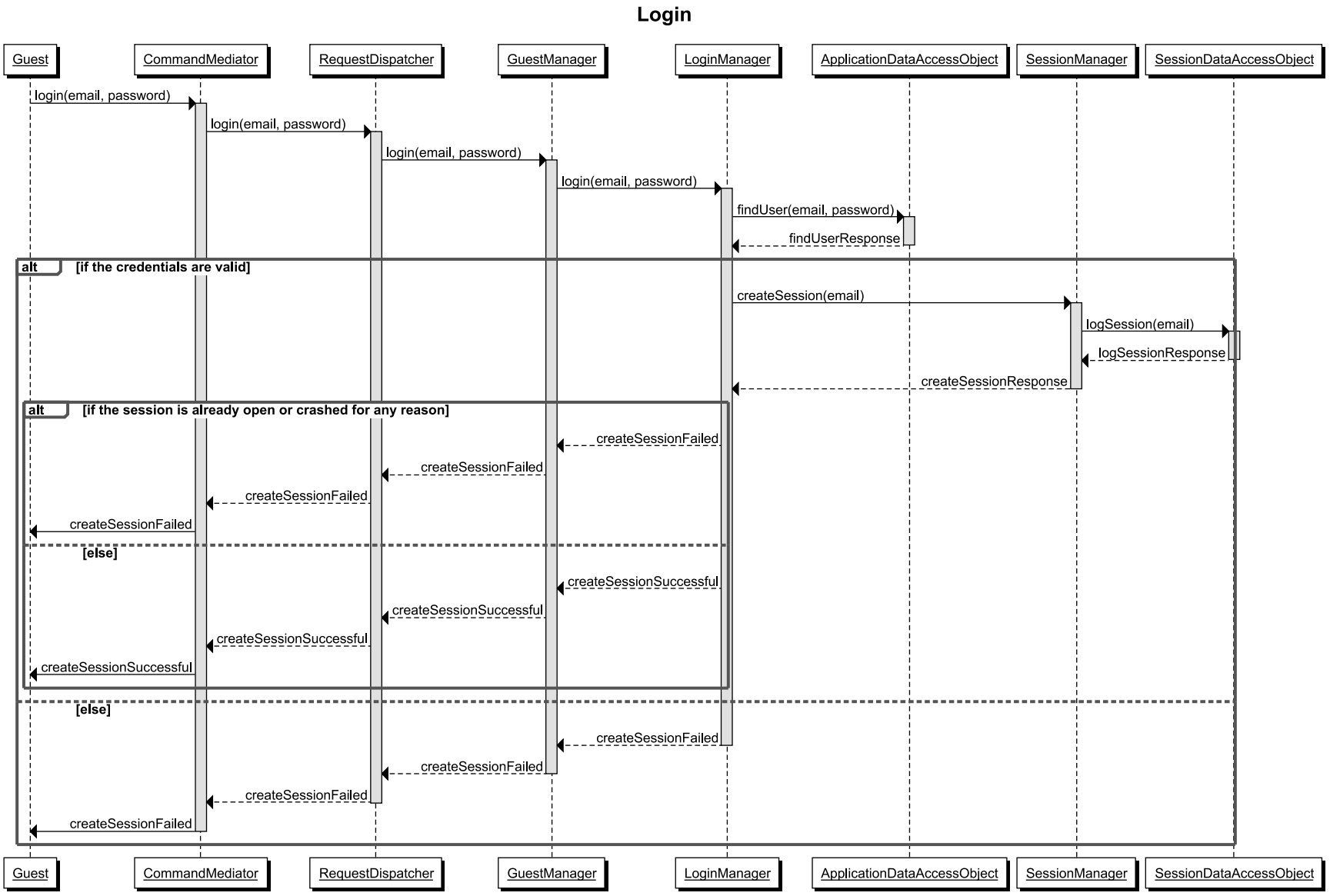


Figure 1.6: Components interactions during login in MyTaxiService

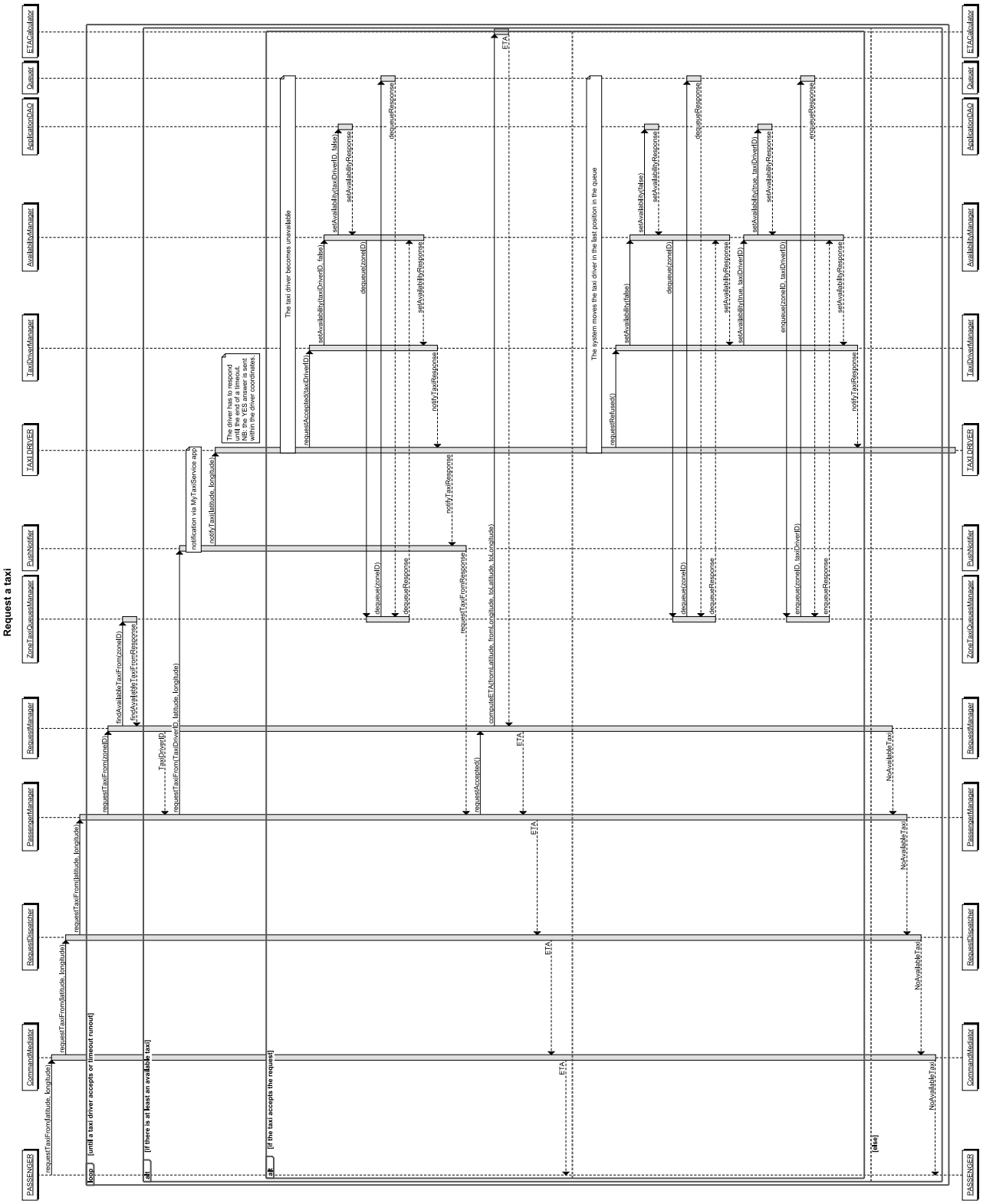


Figure 1.7: Sequence diagram for the request a taxi use case

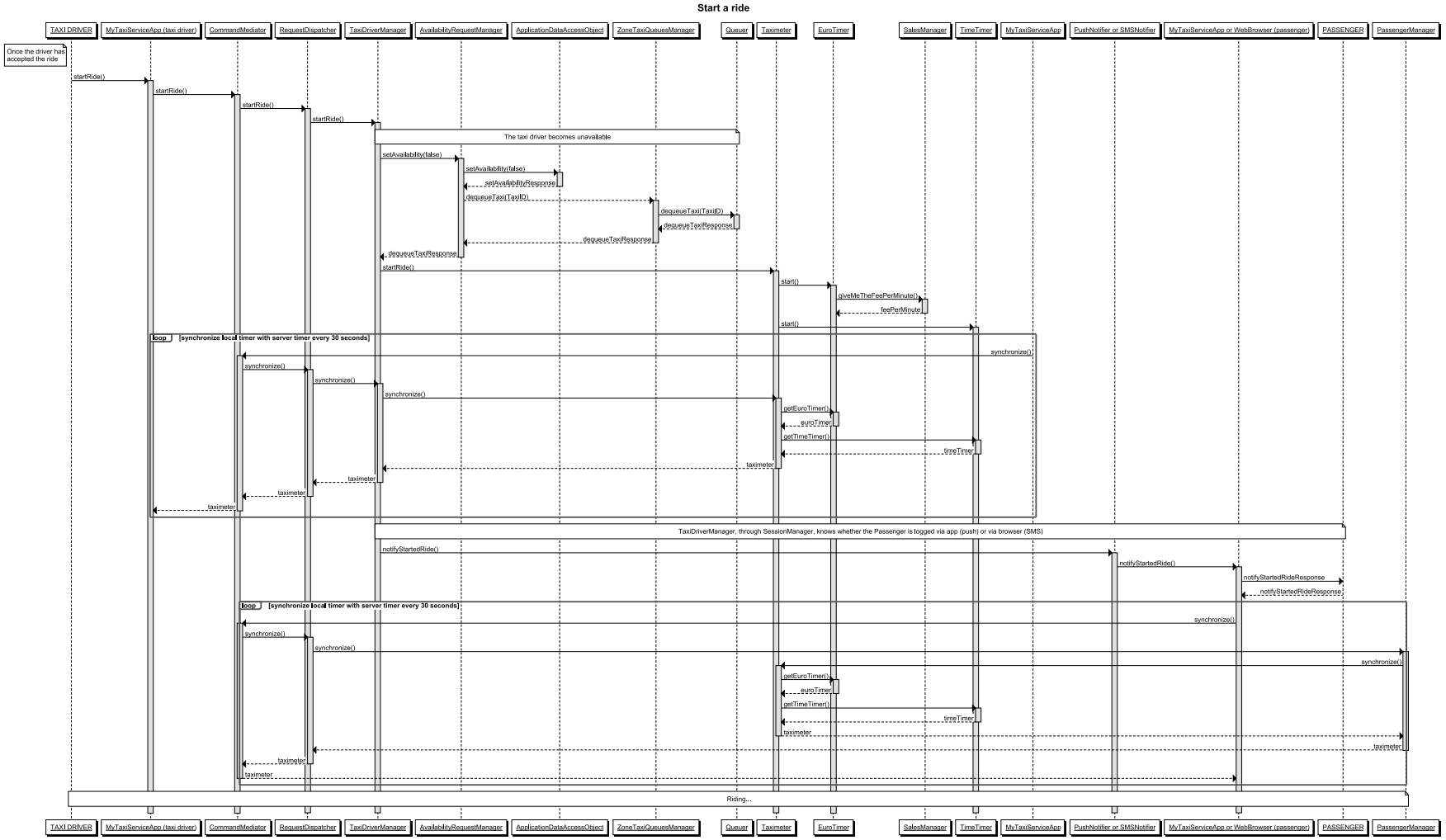


Figure 1.8: Components interactions at the start of a ride in MyTaxiService

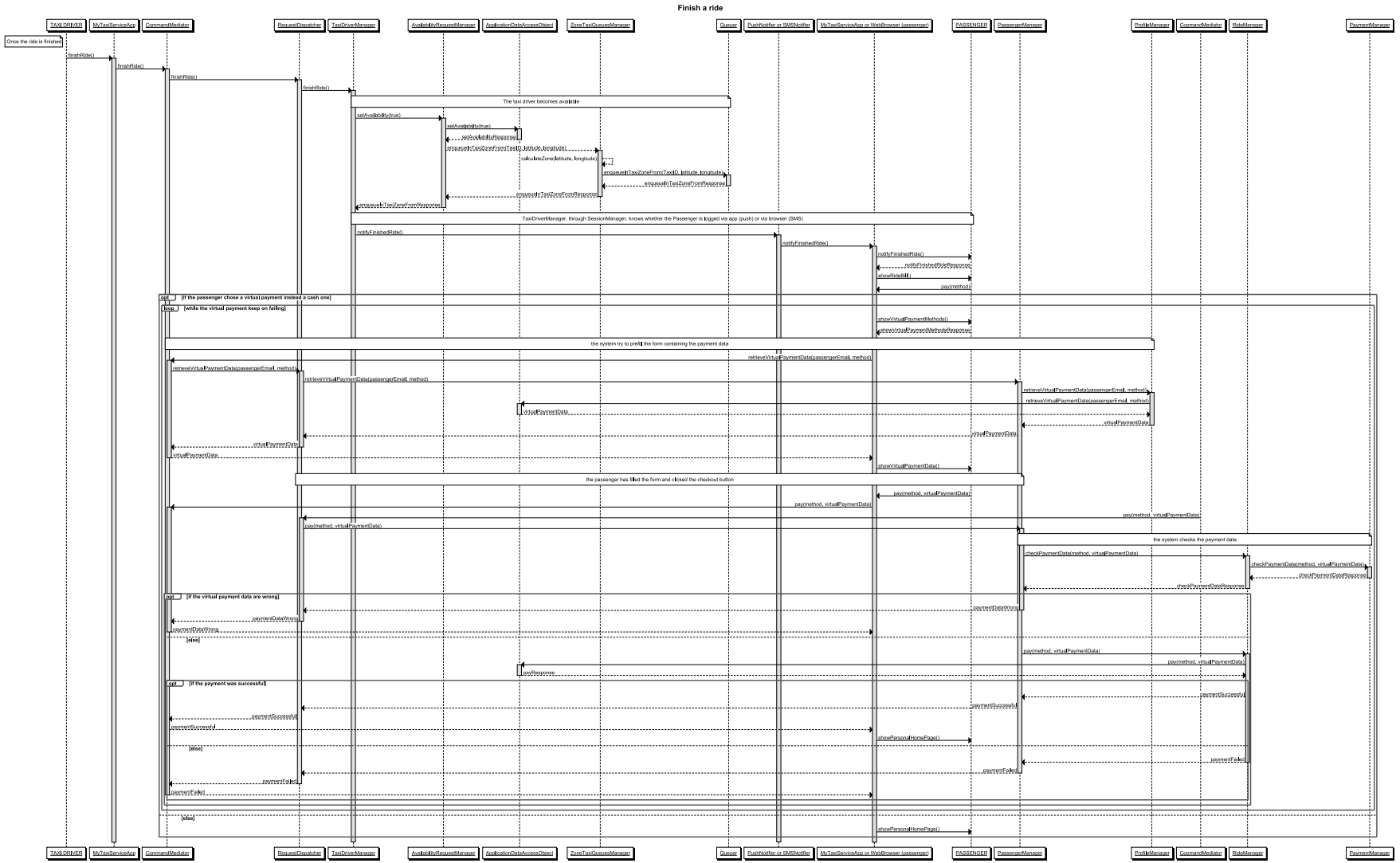


Figure 1.9: Components interactions at the end of a ride in MyTaxiService

1.6 Component interfaces

CommandMediator

- boolean *doCommand*(String command)

Observation: this method translates an external command in an internal request.

RequestDispatcher

- boolean *dispatchRequest*(String request)

Observation: this method calls the appropriate one (of GuestManager, PassengerManager or TaxiDriverManager) based on the content of the parameter *request*

GuestManager

- boolean *register*(String name, String surname, String emailAddress, String password, String cellphoneNumber)
- boolean *register*(String name, String surname, String emailAddress, String password, String cellphoneNumber, PaymentData paymentData)
- boolean *register*(String name, String surname, String emailAddress, String password, String identificationCardID, String taxiLicenseID)
- boolean *register*(String name, String surname, String emailAddress, String password, String identificationCardID, String taxiLicenseID, PaymentData paymentData)
- boolean *login*(String emailAddress, String password)

PassengerManager

- boolean *viewProfile*(String emailAddress)
- boolean *requestTaxi*(float latitude, float longitude)
- boolean *pay*(PaymentData paymentData)
- boolean *logout*(String emailAddress)

TaxiDriverManager

- boolean *viewProfile*(String emailAddress)
- boolean *modifyPersonalData*(String emailAddress, String name, String surname, String emailAddress, String cellphoneNumber)

- boolean *modifyPersonalData*(String emailAddress, String name, String surname, String emailAddress, String identificationCardID, String taxiLicenseID)
- boolean *modifyPaymentsMethods*(String emailAddress, PaymentData paymentData)
- boolean *changePassword*(String emailAddress, String oldPassword, String newPassword)
- boolean *setAvailability*(String taxiDriverID, boolean availability)
- boolean *requestAccepted*(String taxiDriverID)
- boolean *requestRefused*(String taxiDriverID)
- boolean *startRide*(String requestID)
- boolean *finishRide*(String rideID)
- boolean *logout*(String emailAddress)

RegistrationManager

- boolean *register*(String name, String surname, String emailAddress, String password, String cellphoneNumber)
- boolean *register*(String name, String surname, String emailAddress, String password, String cellphoneNumber, PaymentData paymentData)
- boolean *register*(String name, String surname, String emailAddress, String password, String identificationCardID, String taxiLicenseID)
- boolean *register*(String name, String surname, String emailAddress, String password, String identificationCardID, String taxiLicenseID, PaymentData paymentData)

PaymentDataManager

- boolean *checkData*(PaymentData paymentData)
- boolean *pay*(PaymentData paymentData)

PersonalDataManager

- boolean *checkData*(String name, String surname, String cellphoneNumber)

Observation: this method uses the external ones *checkIdentificationCard* and *checkTaxiLicense*

PasswordManager

- boolean *checkData*(String password)

Observation: this method makes password parsing.

MailNotifier

- boolean *send*(String message)

SessionManager

- boolean *createSession*(String emailAddress)

LoginManager

- boolean *login*(String email, String password)

ProfileManager

- boolean *getProfileData*(String emailAddress)
- boolean *setProfileData*(String name, String surname, String emailAddress, String password, String cellphoneNumber)
- boolean *setProfileData*(String name, String surname, String emailAddress, String password, String cellphoneNumber, PaymentData paymentData)
- boolean *setProfileData*(String name, String surname, String emailAddress, String password, String identificationCardID, String taxiLicenseID)
- boolean *setProfileData*(String name, String surname, String emailAddress, String password, String identificationCardID, String taxiLicenseID, PaymentData paymentData)

RideFactory

- boolean *requestTaxiFrom*(int zoneID)
- boolean *requestAccepted*()
- boolean *requestDenied*()

Observation: this methods is called after a taxi request acceptance.

ETACalculator

- boolean *computeETA*(float fromLatitude, float fromLongitude, float toLatitude, float toLongitude)

AvailabilityManager

- boolean *setAvailability*(String taxiDriverID, boolean availability)

ZoneTaxiQueuesManager

- boolean *findAvailableTaxiFrom*(int zoneID)

Observation: this method search, if any, an available taxi driver and returns its user ID (see *Algorithm* section).

Queuer

- boolean *enqueueTaxi*(String taxiDriverID, String zoneID)
- boolean *dequeueTaxi*(String zoneID)

ZoneMapper

- boolean *createZoneQueuesFromMap*(File file)

Observation: this method creates the queues depending on an external file which contains th zone IDs and the needed coordinates of every zone in the city map.

PushNotifier

- boolean *send*(String message)

SMSNotifier

- boolean *send*(String cellphoneNumber, String message)

RideManager

- boolean *startRide*(String requestID)
- boolean *finishRide*(String requestID)
- boolean *checkPaymentData*(PaymentData paymentData)
- boolean *pay*(PaymentData paymentData)

PaymentManager

- boolean *checkPaymentData*(PaymentData paymentData)

Taximeter

- boolean *startRide*()

Observation: this method starts the euro timer and the time timer.

- boolean *synchronize*()

Observation: this method is called in order to synchronize the local timer with the server one.

Eurotimer

- boolean *start*()

Observation: this method starts the euro timer.

- boolean *getEuroTimer*()

SalesManager

- boolean *getFeePerMinute*()

TimeTimer

- boolean *start*()

Observation: this method starts the time timer.

- boolean *getTimeTimer*()

1.7 Selected architectural styles and patterns

The whole system is based on a ***Client-Server approach***, in which thin clients (WebApp and MobileApp) communicate with the fat servers.
The second design decision is to adopt a ***3-layer approach***: Presentation, Logic and Data.
As can be seen from the component diagram, every component below *GuestManager*, *PassengerManager* and *TaxiDriverManager* are specific plugins, each of which has a related role in the system (***plug-in architecture***).
Further decisions consist in the following patterns:

- ***Mediator*** in *CommandMediator*
- ***Adapter*** in *RequestDispatcher*
- ***MVC*** in *MyTaxiServiceApp* and *MyTaxiServiceWebApp* (AngularJS could be used)

1.8 Other design decisions

Other design decision must occur in order to satisfy the non functional requirements listed in the RASD document:

- **Portability**: the code language of the mobile app must be portable (an example could be java+swift), of the other part of the system could be anything
- **Usability**: the GUI design is carried out through the most minimal approach, using 5 main colours (black, white, yellow, green and red), and through a wizard approach
- **Reusability**: the timeouts and the maps are recorded in external files (XML could be used)
- **Maintainability**: every software component has its own role in the system, decoupled from the others
- **Performant**: the whole system can adapt its throughput wrt the demand using the scalability, offered by a cloud-based approach
- **Security**: the cloud provider has to ensure a proper protection of personal data and information

Chapter 2

Algorithm Design

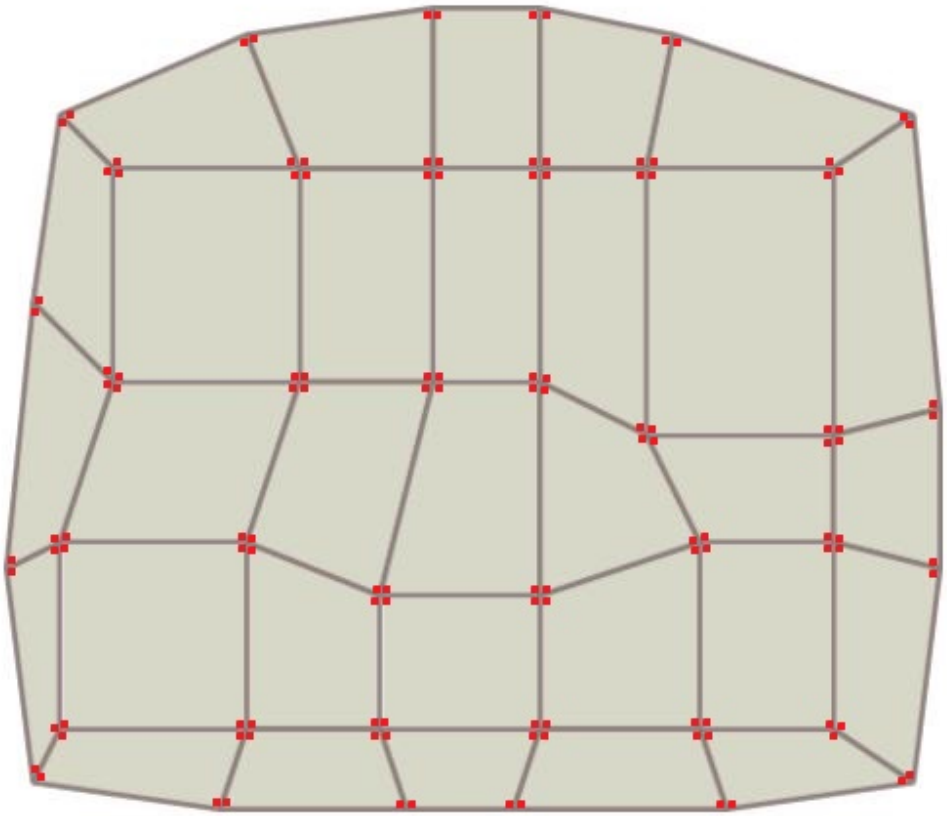
2.1 PassengerManager.requestTaxiFrom(latitude, longitude)

The following pseudocode represents how the system manages a request.
First of all, it tries to detect an available taxi driver in the zone in which the passenger is located (this search has to be done in n minutes).
Then, if the previous search fails, the system try to redo the same search in the adjacent zones parsed in this order: north, east, south, west (also this search has to be done in n minutes).
If all of these searches do not find an available taxi which accepts the request, the method returns an empty string, managed by the frontend through a user notification.
Observation: n has to be defined in an external file.

Algorithm 2.1 PassengerManager.requestTaxiFrom(latitude, longitude)

```
1:  function requestTaxiFrom(latitude, longitude)
2:      //try to find an available taxi from the passenger zone
3:      zoneID ← getZoneIDFrom(latitude, longitude) //see figure below
4:      start(timeout)
5:      repeat
6:          taxiDriverID ← ZoneTaxiQueuesManager.findAvailableTaxiFrom(zoneID)
7:      until timeout = 0 || taxiDriverID != ""
8:      /*if there are no available taxi in the passenger zone,
9:      try to find it from the adjacents ones*/
10:     if taxiDriverID == "" then
11:         adjacentsZoneID ← getAdjacentsZoneID(zoneID)
12:         start(timeout)
13:         repeat
14:             zoneID ← adjacentsZoneID.next()
15:             taxiDriverID ← ZoneTaxiQueuesManager.findAvailableTaxiFrom(zoneID)
16:         until timeout = 0 || taxiDriverID != ""
17:     end if
18:     return taxiDriverID
19: end function
```

```
1: function ZoneTaxiQueuesManager.findAvailableTaxiFrom(zoneID)
2:     taxiDriverID ← ""
3:     queue ← getQueue(zoneID)        ▷ if there are no available taxi drivers, the
   queue is empty
4:     if queue != empty then
5:         taxiDriverID ← queuer.dequeue(queue)
6:     end if
7:     return taxiDriverID
8: end function
```



Every red point (GPS coordinates) represents a vertex of the belonging zone. In order to detect the user zone, it is necessary to find the closest vertex to the user GPS coordinates.
Observation: the coordinates have to be collected in an external file, loaded in the system boot phase.

Chapter 3

User Interface Design

This section provides an enough detailed version of the user experience, showing the GUIs of the app during the main use cases of MyTaxiService.[1]

[1] The user experience through the website is equal to the app one, so it is not necessary to show the website GUIs.

3.1 Registration

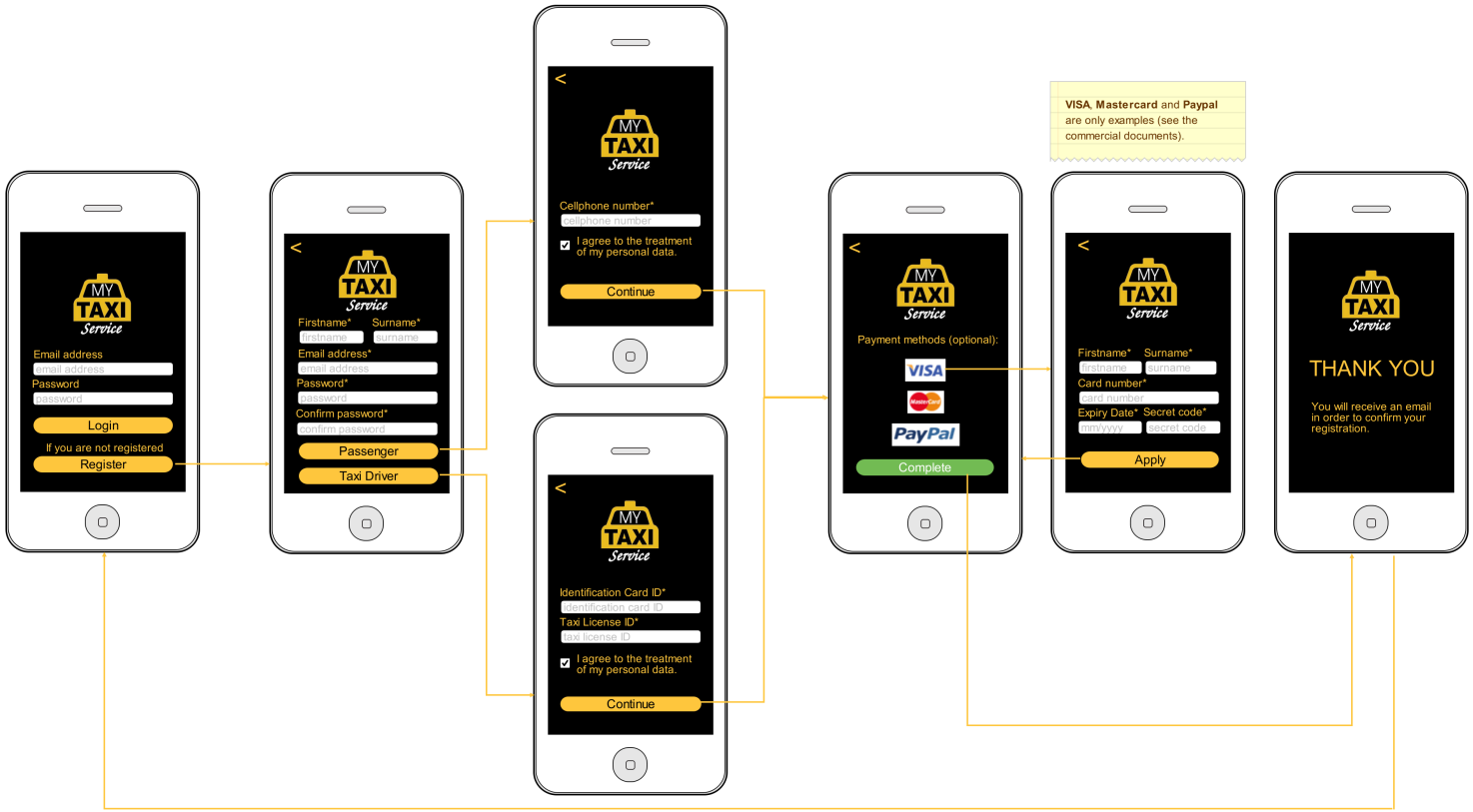


Figure 3.1: User experience in the *registration* phase

3.2 Login

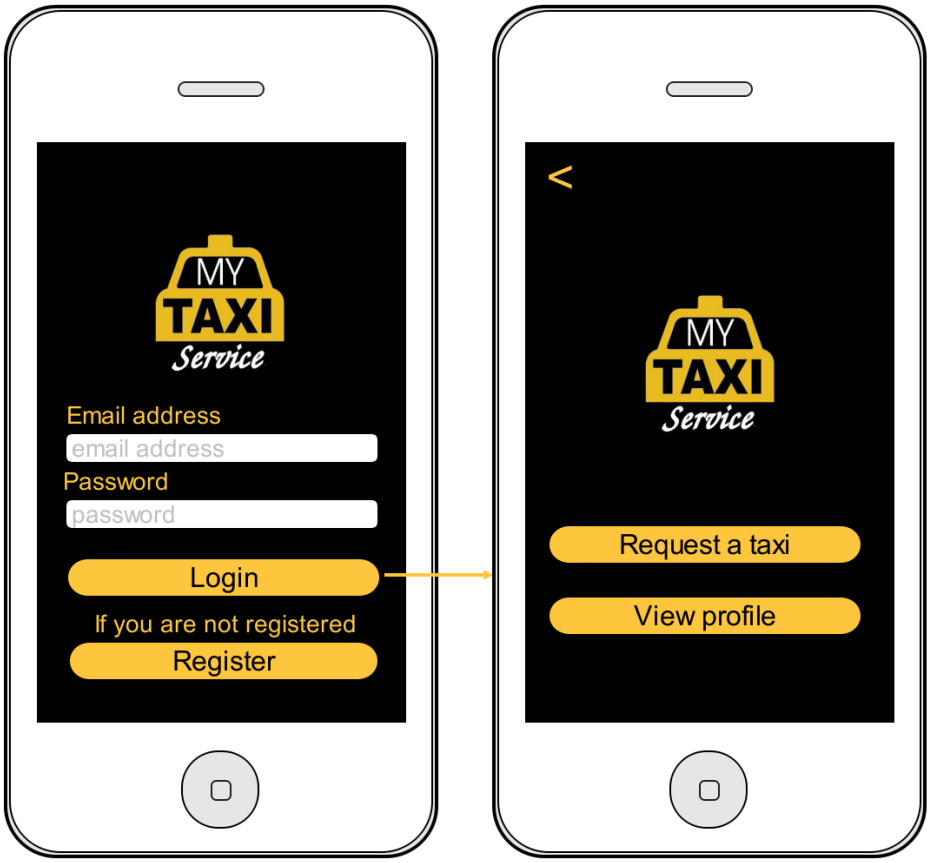


Figure 3.2: User experience in the *login* phase - PASSENGER

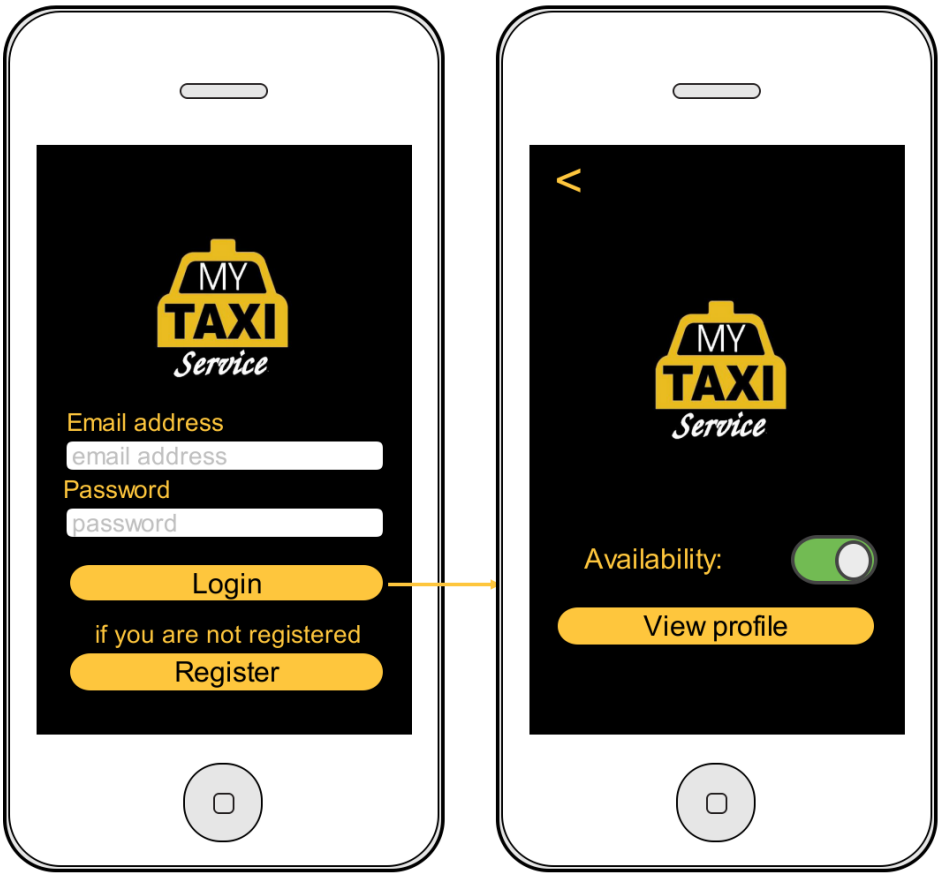


Figure 3.3: User experience in the *login* phase - TAXI DRIVER

3.3 View Profile

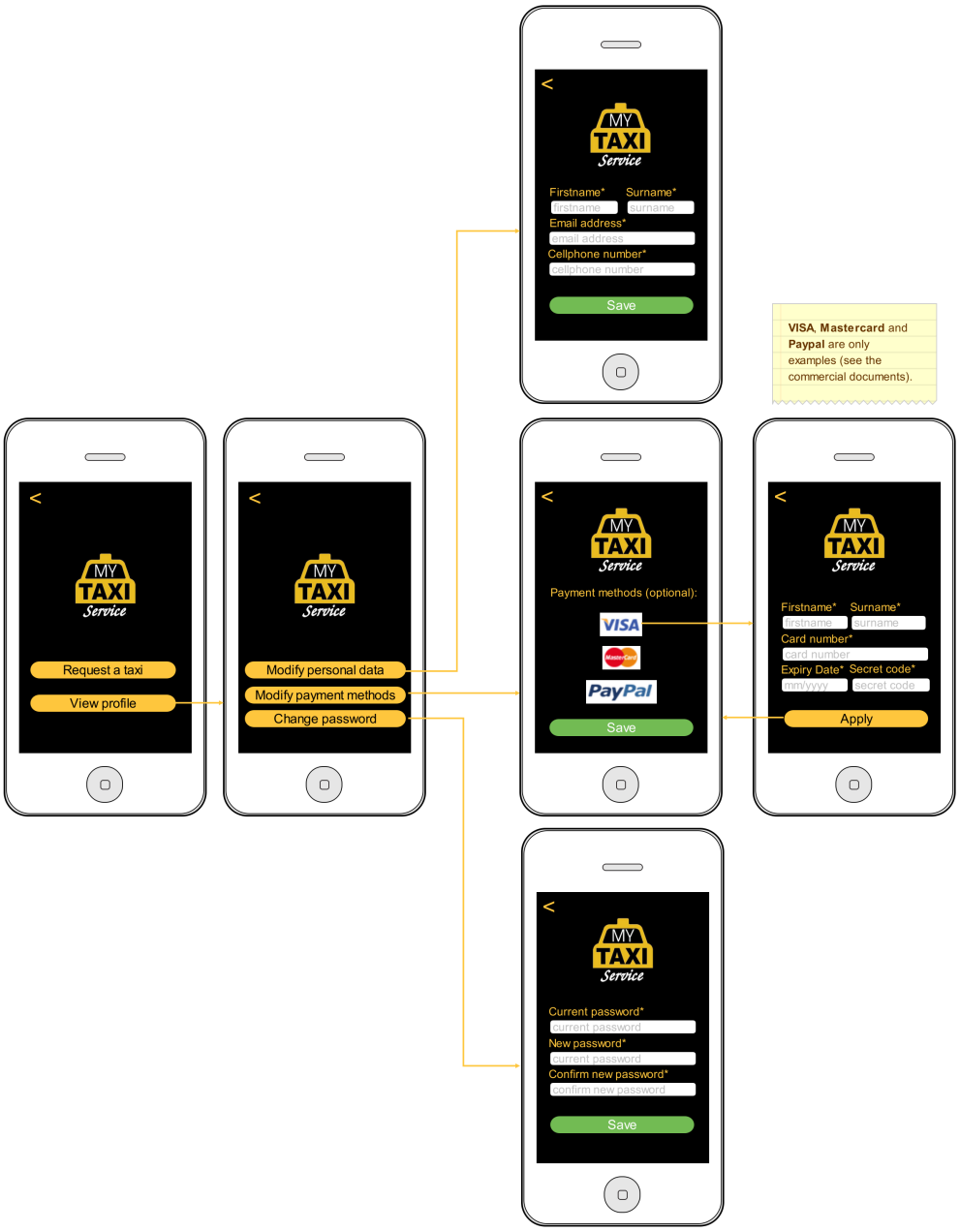


Figure 3.4: User experience in the *view profile* phase - PASSENGER

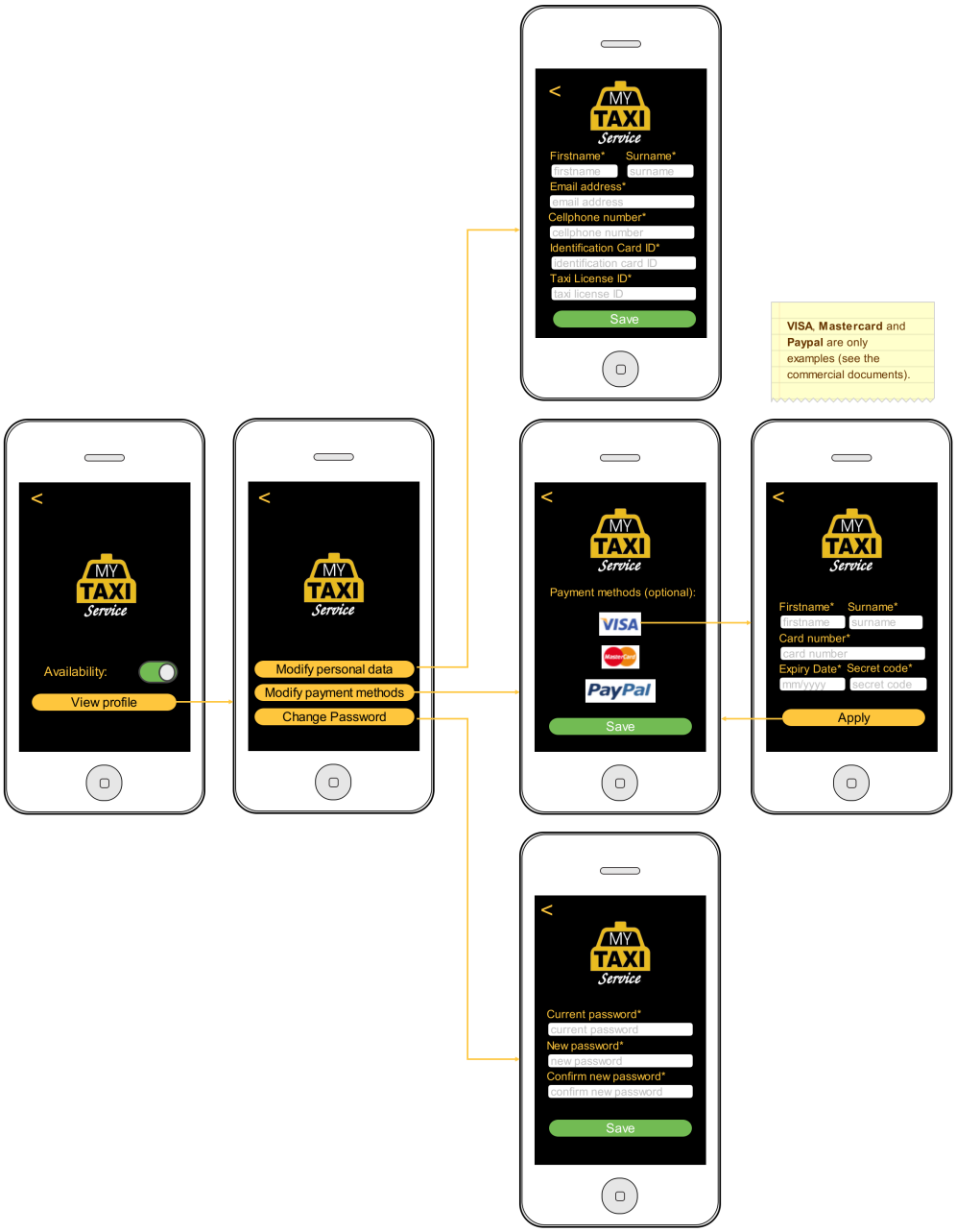


Figure 3.5: User experience in the *view profile* phase - TAXI DRIVER

3.4 Set Availability

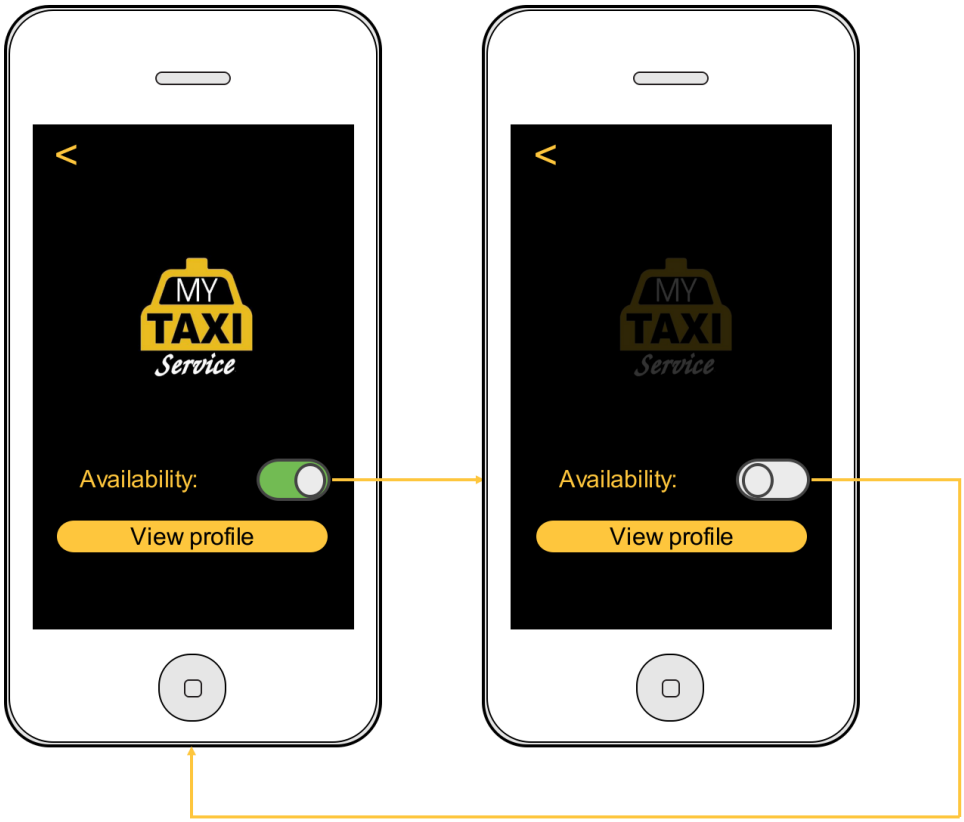


Figure 3.6: User experience in the *availability setting* phase

3.5 Manage a Ride

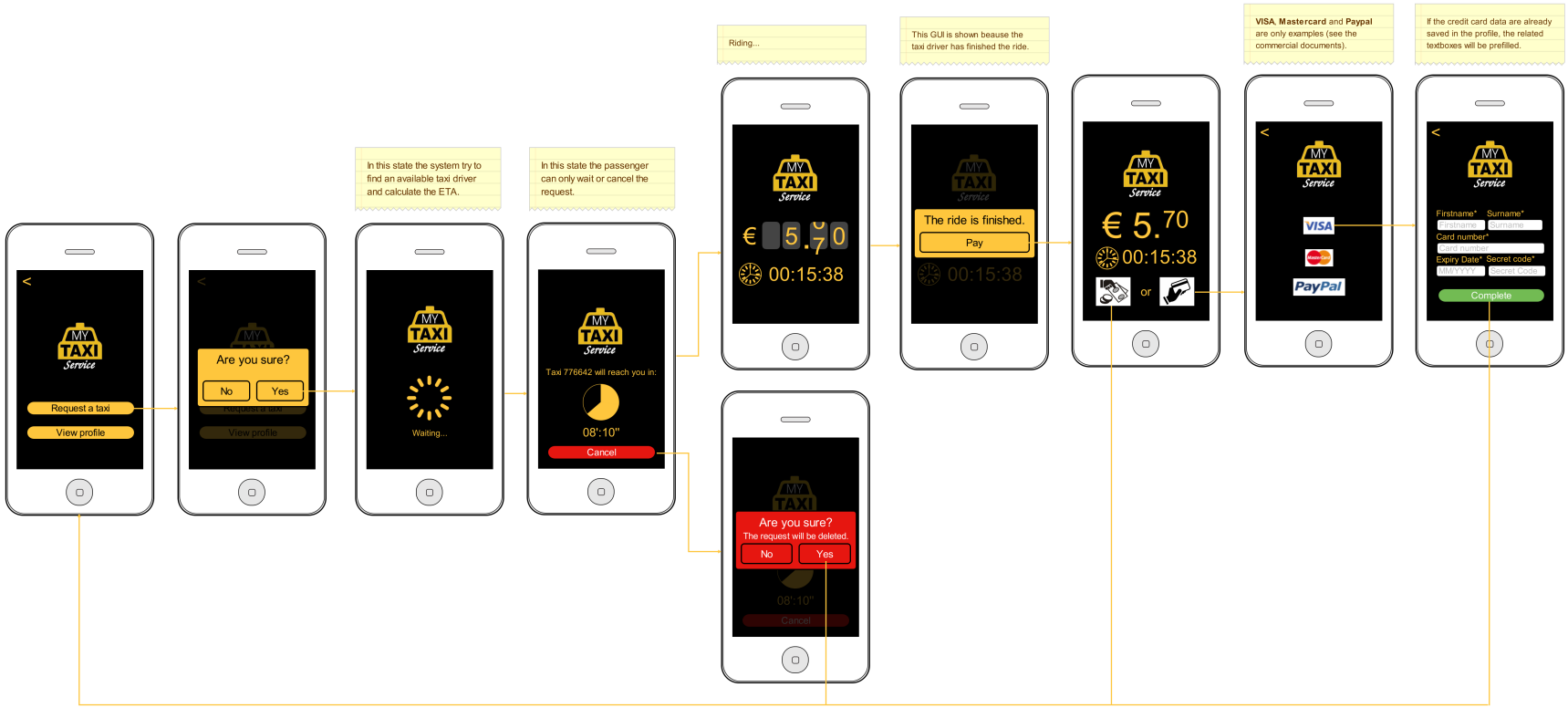


Figure 3.7: User experience in the *ride* phase - PASSENGER

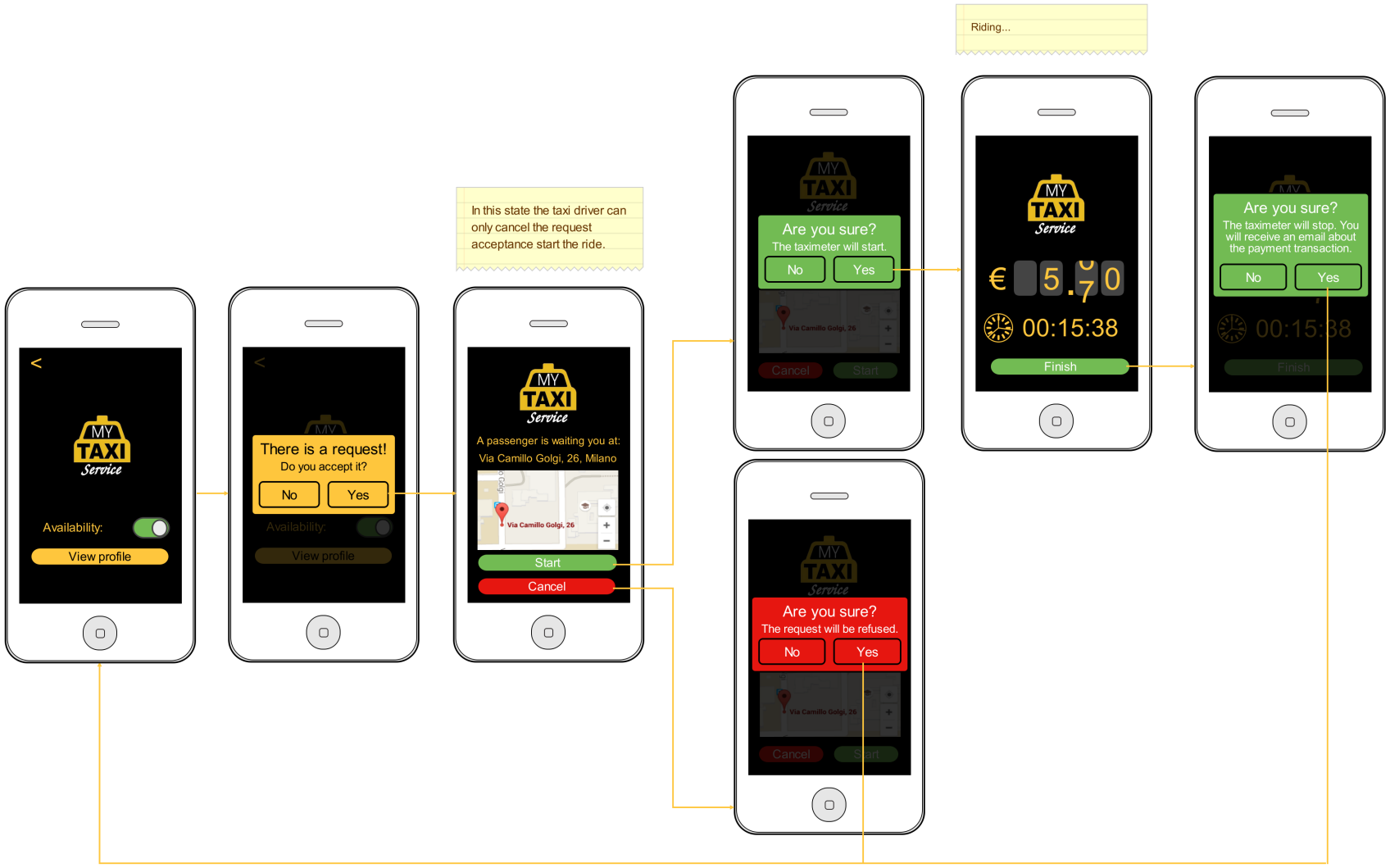


Figure 3.8: User experience in the *ride* phase - TAXI DRIVER

Chapter 4

Requirement Traceability

In order to take care about all the functional requirements, it is required to map them to the software components which satisfy them.

Allow a guest to register

MyTaxiServiceApp/MyTaxiServiceWebApp, CommandMediator, RequestDispatcher, GuestManager, RegistrationManager, PersonalDataManager, PaymentDataManager, PasswordManager, ApplicationDAO, ApplicationDMBS, MailNotifier

Allow a guest to login

MyTaxiServiceApp/MyTaxiServiceWebApp,CommandMediator, RequestDispatcher, GuestManager, LoginManager, SessionManager, SessionDAO, SessionDBMS

Allow a user to modify his/her data

MyTaxiServiceApp/MyTaxiServiceWebApp,CommandMediator, RequestDispatcher, PassengerManager, TaxiDriverManager, ProfileManager, PersonalDataManager, PaymentDataManager, PasswordManager, ApplicationDAO, ApplicationDBMS

Allow a user to logout

MyTaxiServiceApp/MyTaxiServiceWebApp,CommandMediator, RequestDispatcher, PassengerManager, TaxiDriverManager, LoginManager, SessionManager, SessionDAO, SessionDBMS

Allow a passenger to request a taxi

MyTaxiServiceApp/MyTaxiServiceWebApp,CommandMediator, RequestDispatcher, PassengerManager, RideFactory, ETACalculator, ZoneTaxiQueuesManager, Queuer, ZoneMapper

Allow a passenger to choose the payment method for each ride

MyTaxiServiceApp/MyTaxiServiceWebApp,CommandMediator, RequestDispatcher, PassengerManager, RideManager, PaymentManager, PaymentDataManager, ApplicationDAO, ApplicationDBMS

Allow a taxi driver to inform the system about his/her availability

MyTaxiServiceApp/MyTaxiServiceWebApp,CommandMediator, RequestDispatcher, TaxiDriverManager, AvailabilityManager, ZoneTaxiQueuesManager, Queuer, ZoneMapper, ApplicationDAO, ApplicationDBMS

Allow a taxi driver to confirm/deny an incoming request

MyTaxiServiceApp/MyTaxiServiceWebApp,CommandMediator, RequestDispatcher, TaxiDriverManager, RideFactory, ETACalculator, ZoneTaxiQueuesManager, Queuer, ZoneMapper, ApplicationDAO, ApplicationDBMS, PushNotifier/SMSNotifier

Allow a taxi driver to confirm/deny the start of a ride

MyTaxiServiceApp/MyTaxiServiceWebApp,CommandMediator, RequestDispatcher, TaxiDriverManager, RideManager, Taximeter, EuroTimer, SalesManager, TimeTimer, ApplicationDAO, ApplicationDBMS

Allow a taxi driver to confirm the end of a ride

MyTaxiServiceApp/MyTaxiServiceWebApp,CommandMediator, RequestDispatcher, TaxiDriverManager, AvailabilityManager, ZoneTaxiQueuesManager, Queuer, ApplicationDAO, ApplicationDBMS, PushNotifier/SMSNotifier

The system has to assign an ID to every taxi driver

MyTaxiServiceApp/MyTaxiServiceWebApp, CommandMediator, RequestDispatcher, GuestManager, RegistrationManager, PersonalDataManager, PaymentDataManager, PasswordManager, ApplicationDAO, ApplicationDMBS

The system has to assign dinamically every taxi driver to a zone queue

MyTaxiServiceApp/MyTaxiServiceWebApp,CommandMediator, RequestDispatcher, TaxiDriverManager, AvailabilityManager, ZoneTaxiQueuesManager, Queuer, ZoneMapper, ApplicationDAO, ApplicationDBMS

The system has to forward a passenger request to the first available taxi driver

MyTaxiServiceApp/MyTaxiServiceWebApp,CommandMediator, RequestDispatcher, PassengerManager, RideFactory, ETACalculator, ZoneTaxiQueuesManager, Queuer, ZoneMapper

The system has to notify the passenger about the acceptance of his/her taxi request

MyTaxiServiceApp/MyTaxiServiceWebApp,CommandMediator, RequestDispatcher, TaxiDriverManager, RideFactory, ETACalculator, ZoneTaxiQueuesManager, Queuer, ZoneMapper, PushNotifier/SMSNotifier

The system has to move in the last queue position a taxi driver who either denied a request or not answer it in 5 minutes

MyTaxiServiceApp/MyTaxiServiceWebApp,CommandMediator, RequestDispatcher, TaxiDriverManager, RideFactory, ETACalculator, ZoneTaxiQueuesManager, Queuer, ZoneMapper, ApplicationDAO, ApplicationDBMS, PushNotifier/SMSNotifier

The system has to start a taximeter when the ride begin and stop it when it ends

MyTaxiServiceApp/MyTaxiServiceWebApp,CommandMediator, RequestDispatcher, TaxiDriverManager, RideManager, Taximeter, EuroTimer, SalesManager, TimeTimer, ApplicationDAO, ApplicationDBMS

The system has to compute the bill when the ride ends

MyTaxiServiceApp/MyTaxiServiceWebApp,CommandMediator, RequestDispatcher, TaxiDriverManager, RideManager, Taximeter, SalesManager, ApplicationDAO, ApplicationDBMS

Appendix A

Document Information

A.1 Effort

Approximately **100 hours** have been spent making this document.

A.2 Tool Used

- **LyX**: www.lyx.org
- **Moqups**: https://moqups.com/
- **WebSequenceDiagrams**: https://www.websequencediagrams.com/
- **LucidChart**: https://www.lucidchart.com/