Politecnico di Milano

A.A. 2015-2016

Software Engineering 2 Project

myTaxiService

Design Document

Ver. 2

December 4, 2015

Alessio Martorana - 860584

1 Introduction

1.1 Purpose

This document represent the Design Document (DD), it regards Data design and on how the interaction between our system and the users should be.

In this document the architecture and design of myTaxiService application are described.

The document explains user interactions, system responses and behavior with the database modeling and structure.

In this document the way in which the use cases and requirements listed in RASD will be implemented.

The document is expected to be ridden mainly by system developers, but also general users of the system could be interested in reading it.

1.2 Scope

System main goal is to supply to users of a big city a simple access to taxi service and to ensure a fair taxi queue management, in order to optimize user request assignment.

The system basic architecture will be a three-tier client/server architecture, composed of: client, server database and server application; also an event architecture is used together with the client/server one in order to set up the *Model View Controller* pattern.

The entire system is divided into smaller components in order to divide responsibilities and to create the best possible design.

1.3 Definitions, acronyms, and abbreviations

- **Guest:** A person who hasn't either signed up or logged in to the application yet and, therefore, could only register to the application or log-in to the application;
- **User:** A person who has successfully logged in to the application and, hence, can use all the features of the application offered to users;
- **Taxi driver:** Driver of one of the taxis handled by the system;
- **Taxi request:** Indicates a taxi request performed by the user through the application;
- **Status :** Indicates if the taxi driver is "Available" or "Unavailable", as specified in the next point;
- **Availability:** Indicates if a taxi driver is "at work" and so could be considered a valid taxi to which the system can forward a user request.

- **RASD:** Requiremente Analysis and Specification Document

1.4 Reference Documents

- Project material: Assignments 1 and 2.pdf
- Project material: IEEE Standard Systems and software engineering - Architecture description
- Project material: IEEE Standard for Information Technology—Systems Design— Software Design Descriptions.pdf

1.5 Document Structure

The document is structured in the following parts:

- *Introduction:* Is where the document is described and architecture and design of the system are introduced.
- *Architectural Design:* In this central part of the document, the particular style of the architecture and design of the system is examined in details; in particular is specified if the system (as often happens) will be divided in smaller components, which role and task every component has and how the components will communicate each other.
- *Algorithm Design:* Here the main algorithms which are used in the system are discussed and analyzed in order to show their role in the system logic.
- *User Interface Design:* Here all the user interfaces of the system are described, also the detailed flow among interfaces is showed.
- *Requirements Traceability:* In this section requirements found in the RASD are associated and made traceable.
- *References:* In this chapter the guidelines examined to create this document are listed.
- *Appendix:* Contains all the information that are not strictly related to the document, but is somehow linked to it like, for example: team working hours.

2 Architectural Design

2.1 Overview

In this section the architecture of the system will be described, the architecture and design of the system will be presented at the highest possible level of detail. The choice of keeping the architecture level high is deliberately taken in order to let the developers free to implement the system in the way which they think it's the best.
As introduced in the scope paragraph of first chapter, system's architecture is a three-tier client/server architecture with an asynchronous paradigm of communication.

2.1.1 Brief architecture description

The architecture is based on "thin" clients interaction with three main entities: central node, dispatcher and business servers. Each server is associated to one city zone and handles the business logic associated to user request serving and taxis management.
Central node component manages user registration and authentication functions together with forwarding of user requests to the dispatcher and forwarding requests results coming to business servers to users.
The dispatcher component deals with forwarding of user request and taxi driver information to the business server of the right zone; lastly, business servers handle user requests – by consulting taxi driver queues - and forward user requests to taxi drivers and responses to users.

2.2 High level components and their interaction

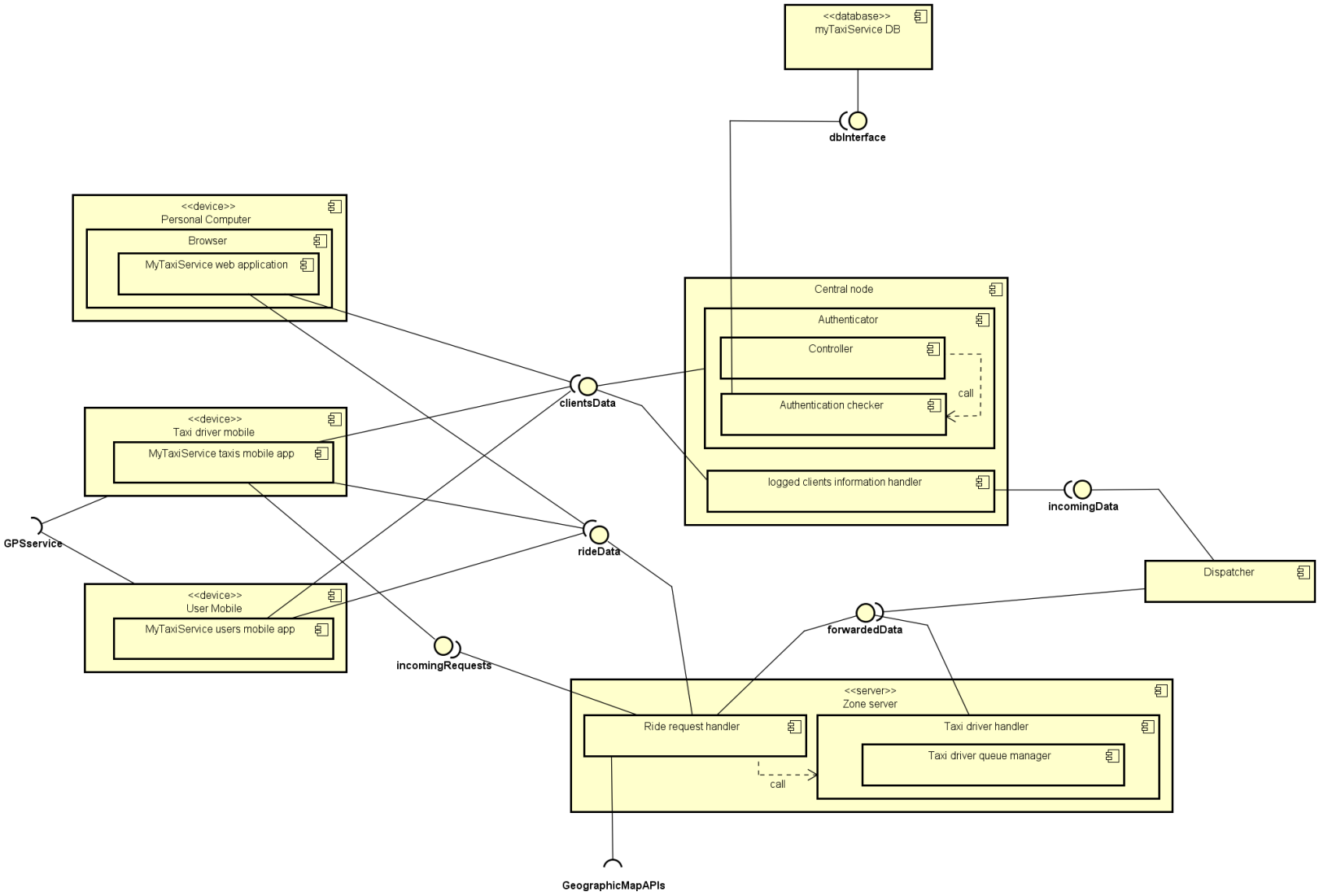
In this section high level components and their interactions will be analyzed using a table which associates a component and overview of its function. Components interacts among them using interfaces which are listed in section 2.6.

COMPONENT	FUNCTION
myTaxiService web application	Permits the user to interacts with the system from his Personal Computer, it directly interacts with the Central Node component in order to communicate users ride information.
myTaxiService users mobile app	Is the component through which users can interact with the system from their mobile device, it directly interacts with the Central Node component in order to communicate users ride information.
myTaxiService taxis mobile app	Is the component used by taxi drivers to accept or decline requests, it also communicates the taxi driver position to Central Node every two minutes.
Central Node	It's the "core" of the system, performs the authentication function and handles communication with authenticated clients, forwarding incoming data to the dispatcher
Dispatcher	It forwards the incoming data received from the Central Node to the Business Server of the right zone.
Business Server of zone N	Is the server which serves the Nth city zone. It receives clients information from the dispatcher, computes and send ride communication data

2.3 Component view

Here the component view is shown. The following diagram shows how the system components have interact and divide basic system functions. The model shows how system's architecture strongly divides responsibilities among components.

More specifically from the diagram can be seen how the central node interacts with the database to register/authenticate users and taxi drivers, it also uses his dispatcher in order to forward users requests and taxi drivers information; the dispatcher takes these information from central node and forwards it to the business server of the right zone.

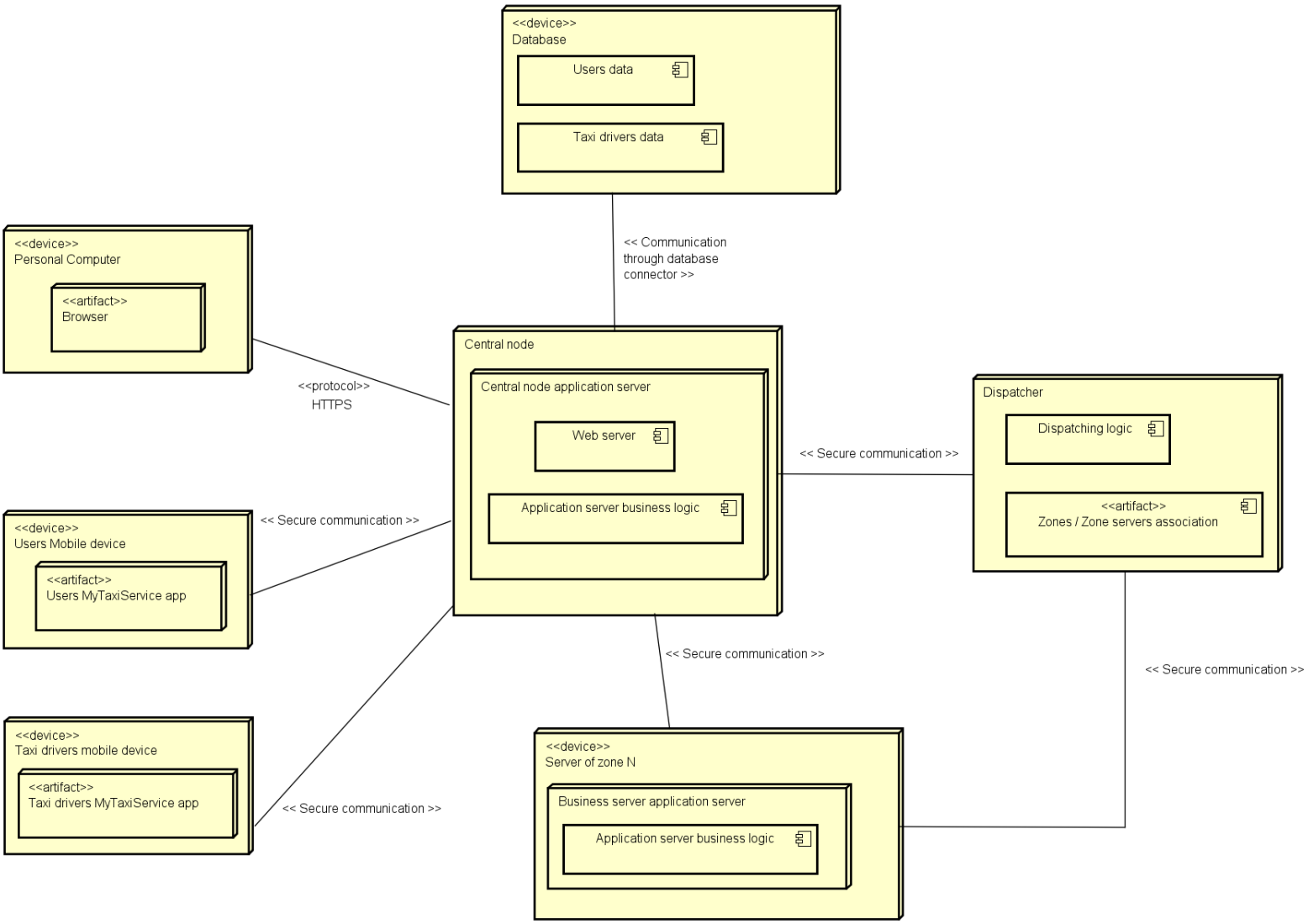

<<database>>
myTaxiService DB
dbInterface
<<device>>
Personal Computer
Browser
MyTaxiService web application
Central node
Authenticator
Controller
Authentication checker
call
clientsData
<<device>>
Taxi driver mobile
MyTaxiService taxis mobile app
logged clients information handler
incomingData
GPSservice
rideData
Dispatcher
<<device>>
User Mobile
MyTaxiService users mobile app
forwardedData
incomingRequests
<<server>>
Zone server
Ride request handler
Taxi driver handler
Taxi driver queue manager
call
GeographicMapAPIs

2.4 Deployment view

The following diagram shows the deployment view, depicting a static view of the nodes. As can be seen from the diagram, the central node is composed by a web server, a logic which handle registration/authentication requests and ride requests making use of the information taken from the database, which simply – as seen in the component view- has a dispatching logic which permits it to forward user requests and taxi driver information to the business server of the correct zone; the business server is composed by an application server which proceeds to the computation of the requests through its application logic. As was said in the *Overview* paragraph of this chapter, system's architecture is described at a very high level, in order to let the higher possible freedom to the developers, so communication among nodes is very highly described too, stating that all the communications channels have to be secure and with only one restriction on the protocol used for communication between the browser and the web server, which has to be *HTTP over Secure Socket Layer (HTTPS)*.

From the diagram it can also be seen that the database has to store users data and taxi drivers data, such as usernames emails and passwords and taxi drivers taxi codes and passwords.

It's important to highlight how nodes without the "<<device>>" stereotype hasn't necessarily to be single devices, in fact "central node" node for example could be implemented with several machines to better balance clients connection load.

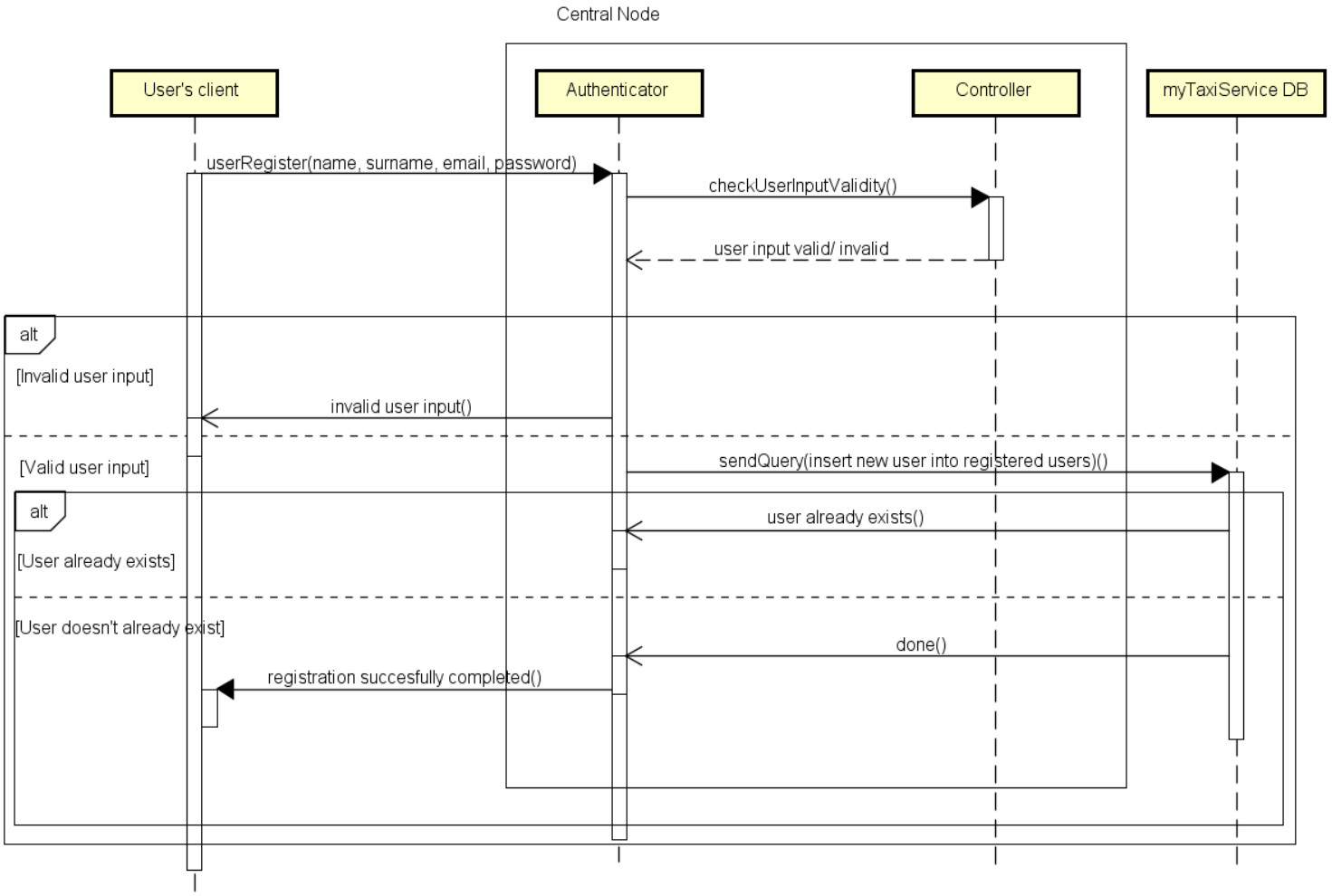

<<device>>
Database
Users data
Taxi drivers data
<< Communication through database connector >>
<<device>>
Personal Computer
<<artifact>>
Browser
<<protocol>>
HTTPS
Central node
Central node application server
Web server
Application server business logic
Dispatcher
Dispatching logic
<<artifact>>
Zones / Zone servers association
<< Secure communication >>
<<device>>
Users Mobile device
<<artifact>>
Users MyTaxiService app
<< Secure communication >>
<< Secure communication >>
<< Secure communication >>
<<device>>
Taxi drivers mobile device
<<artifact>>
Taxi drivers MyTaxiService app
<< Secure communication >>
<<device>>
Server of zone N
Business server application server
Application server business logic

2.5 Runtime view

Here runtime views of some basic functions of the system are shown.

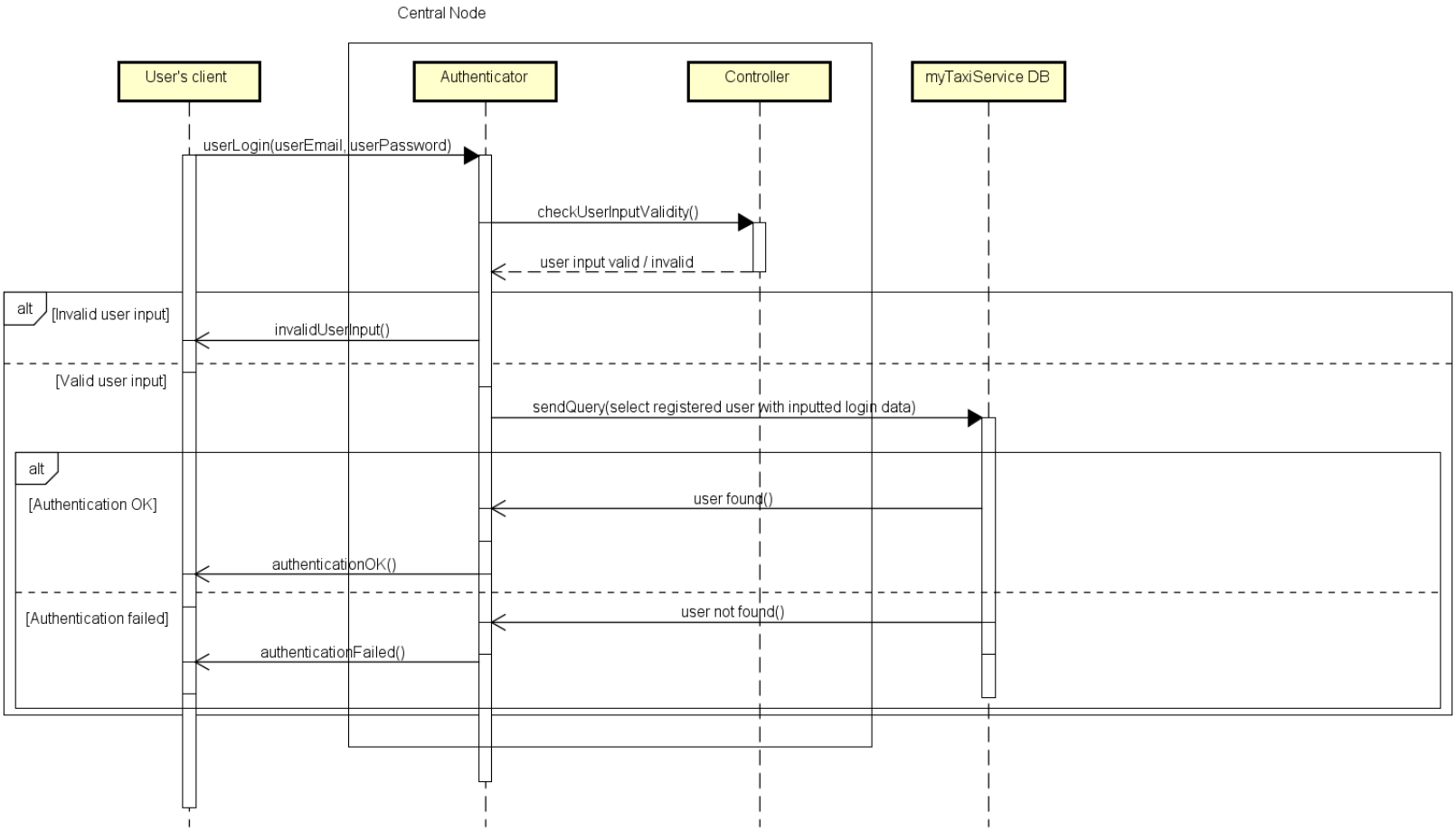
2.5.1 User Registration

As could be seen in the component view, user registration function is accomplished by the central node, so a deeper view of the actions can be seen in this sequence diagram, where the user input is validated by the Authenticator component using his Controller and then user data – if the user doesn't already exist – is added to the database.



2.5.2 User Login

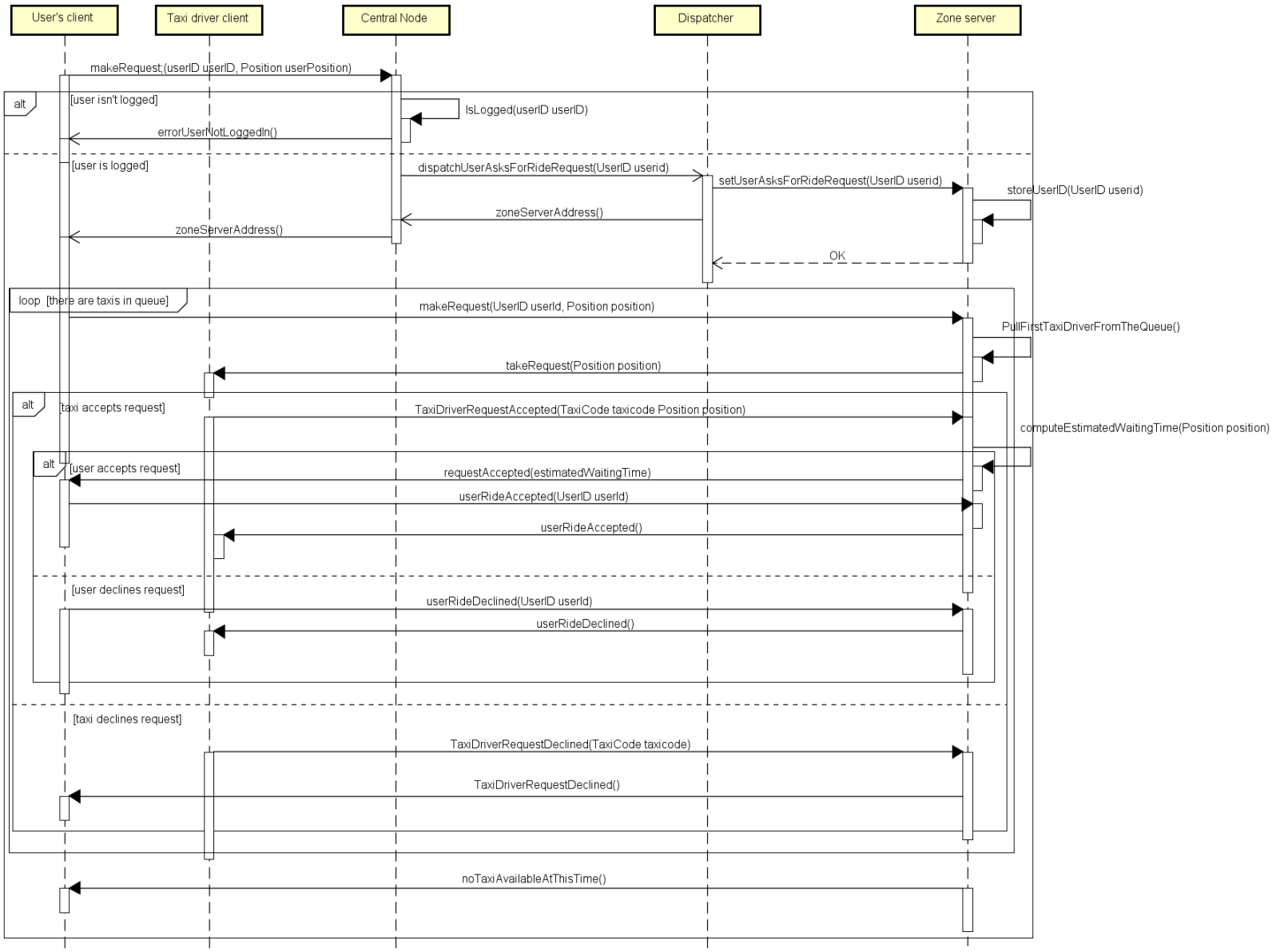
Like the registration function, login function is accomplished by the central node too. Once again user input is validated by the Controller and the authentication is checked by the Authenticator component, which interfaces with the database to check if the user wrote the right credentials (user's credential are in database).



2.5.3 Ride request handling

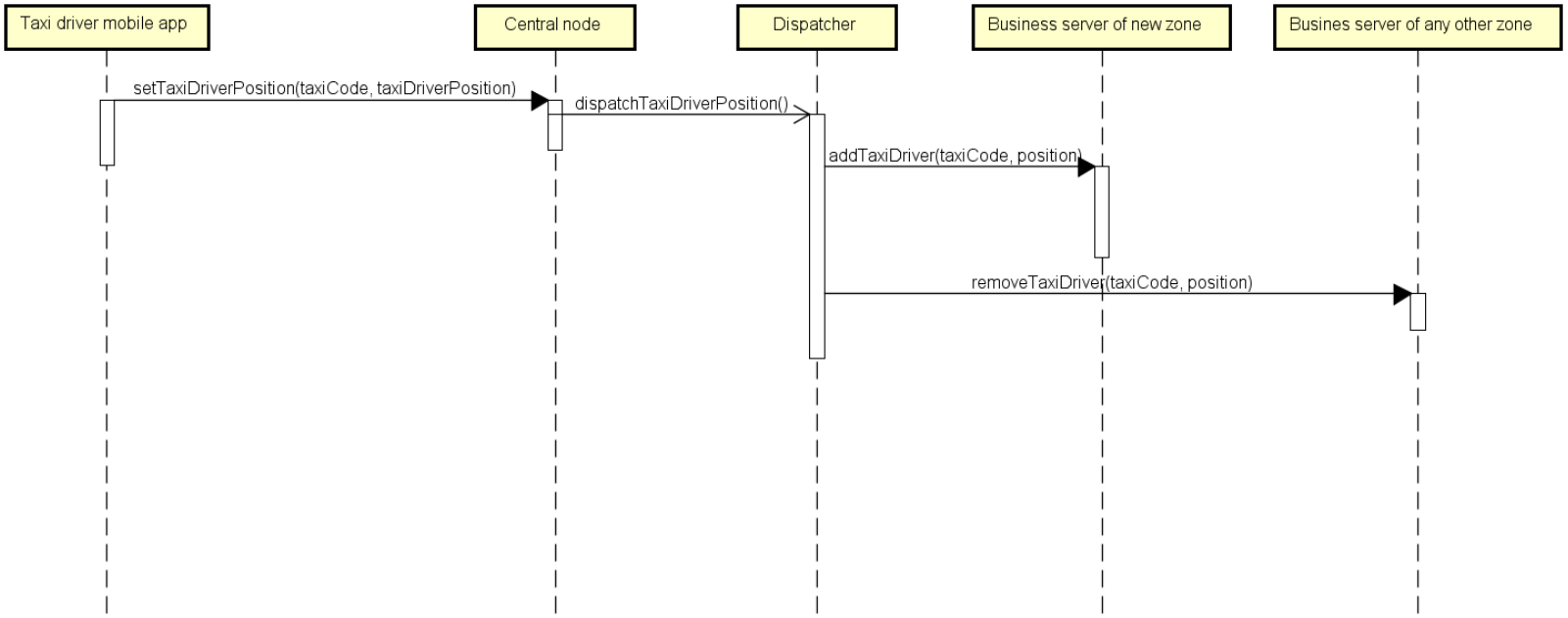
In this sequence diagram the handling of a ride request (in all cases) could be seen. The main thing to notice is that the central node only checks whether the user is logged or not and, in the case he is, forwards data to the dispatcher and – later – answers the user's client with the zone server address. Ingoing data, in fact, is forwarded to the business server of the city zone by the dispatcher. After enabling of the user, the business server's address is sent from the dispatcher to the central node and then again to the client; from that moment on, communication only happens between user's client and the zone server.

It's important to notice the "on demand" (asynchronous) type of service offered by this architecture and the level of security guaranteed by the "enabling" of the user to communicate with the zone server.


User's client
Taxi driver client
Central Node
Dispatcher
Zone server
makeRequest;(userID userID, Position userPosition)
alt
[user isn't logged]
IsLogged(userID userID)
errorUserNotLoggedIn()
[user is logged]
dispatchUserAsksForRideRequest(UserID userid)
setUserAsksForRideRequest(UserID userid)
storeUserID(UserID userid)
zoneServerAddress()
zoneServerAddress()
OK
loop [there are taxis in queue]
makeRequest(UserID userId, Position position)
PullFirstTaxiDriverFromTheQueue()
takeRequest(Position position)
alt
[taxi accepts request]
TaxiDriverRequestAccepted(TaxiCode taxicode Position position)
computeEstimatedWaitingTime(Position position)
alt
[user accepts request]
requestAccepted(estimatedWaitingTime)
userRideAccepted(UserID userId)
userRideAccepted()
[user declines request]
userRideDeclined(UserID userId)
userRideDeclined()
[taxi declines request]
TaxiDriverRequestDeclined(TaxiCode taxicode)
TaxiDriverRequestDeclined()
noTaxiAvailableAtThisTime()

2.5.4 Taxi Driver position communication

In the following sequence diagram is showed how taxi driver position is dynamically handled by dispatching the “addTaxiDriver” message to the business server of the zone in which the taxi driver is at that moment and by dispatching the “removeTaxiDriver” message to all other business servers associated to a different zone. Of course the implementation is supposed to ignore “addTaxiDriver” messages if the taxi driver is still registered into the business server of the zone (taxi driver hasn’t changed zone of the city) and ignore “removeTaxiDriver” messages in case the taxi driver isn’t already registered into the business server.



2.6 Component interfaces

Here component interfaces and relative methods are listed according to the component which implements them.

2.6.1 Central node

2.6.1.1 Authenticator – Logged clients information handler

interface clientsData {

userRegister (Name name, Surname surname, Email email, Password password);
userLogin (Email userEmail, Password userPassword);
makeRequest (userID userID, Position userPosition);
taxiDriverLogin (TaxiCode taxicode, Password password);
setTaxiDriverPosition (TaxiCode taxicode, Position taxiDriverPosition);
setTaxiDriverAvailability (TaxiCode taxicode, Boolean taxiDriverAvailability);

}

2.6.2 Dispatcher

interface incomingData {

dispatchTaxiDriverPosition (TaxiCode taxicode, Position position);

dispatchTaxiDriverAvailability (TaxiCode taxicode, Boolean taxiDriverAvailability);

dispatchUserAsksForRideRequest (UserID userid);

}

2.6.3 Zone server

interface forwardedData {

addTaxiDriver (TaxiCode taxicode, Position position);

removeTaxiDriver (TaxiCode taxicode, Position position);

setTaxiDriverAvailability (TaxiCode taxicode, Boolean taxiDriverAvailability);

setUserAsksForRideRequest (UserID userId);

}

interface rideData {

makeRequest (UserID userId, Position position);

userRideAccepted(UserID userId);

userRideDeclined(UserID userId);

TaxiDriverRequestAccepted (TaxiCode taxicode);

TaxiDriverRequestDeclined (TaxiCode taxicode);

}

2.6.4 MyTaxiDriver taxis mobile app

interface incomingRequests {

takeRequest(Position position);

}

2.6.5 myTaxiService DB

interface dbInterface {

sendQuery (Query query);

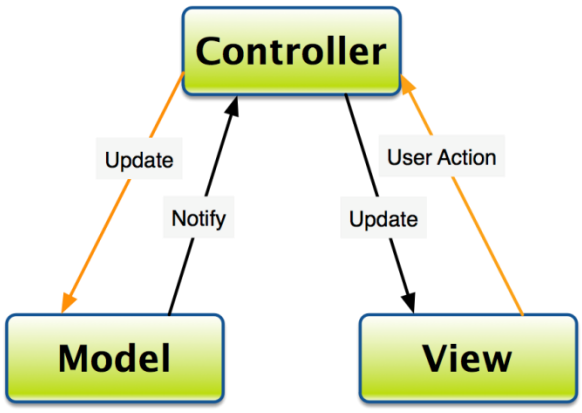
}

2.7 Selected architectural styles and patterns

2.7.1 Model View Controller

The Model–view–controller (MVC) design pattern is an architectural pattern which aims to clearly divide application's user interface, data model and control logic in order to allow modification on the application components with a minimal impact in others. In the MVC design pattern so, the application is divided in *Model*, *view* and *controller* components, where: the model component embodies data representation and business logic of the application, the controller component embodies the control logic and interacts with the model and the view component embodies the application interfacing logic, typically supplying an user interface element.

In myTaxiService it has been applied in the logic of the center node component to better divide responsibilities and roles among subcomponents.

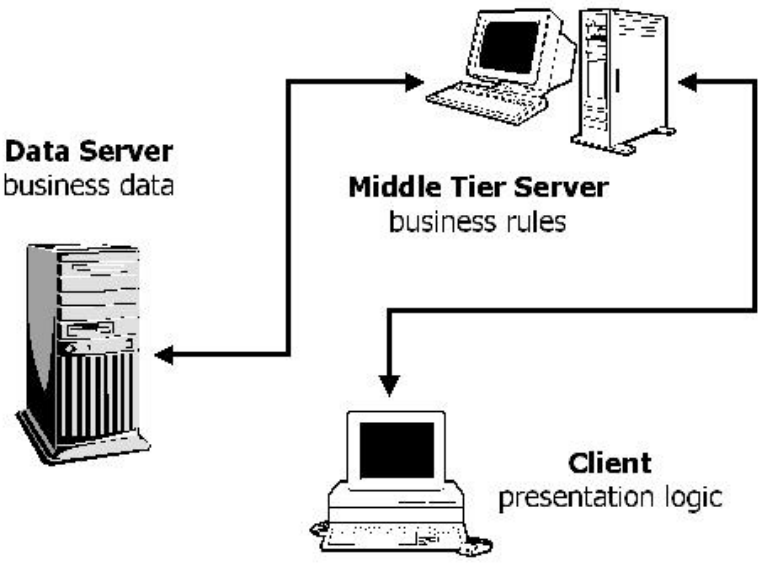


2.7.2 Three-tier Client/Server architecture

The client–server architecture is a distributed application structure that divides a system workflow between providers servers which provide some service and clients, who request the service.

In the “three-tier” architectural scheme, the architecture is divided in: interface level, application logic level and data level. This partition has the advantage to keeping responsibilities clearly divided decoupling interface, logic and data.

In myTaxiService it has been applied basically all over the system, in fact the system is based on the communication of users and taxi drivers clients with several city zone servers.



3 User interface design

The user interface has been extensively described in RASD document, here the user experience will be analyzed.

3.1 User experience

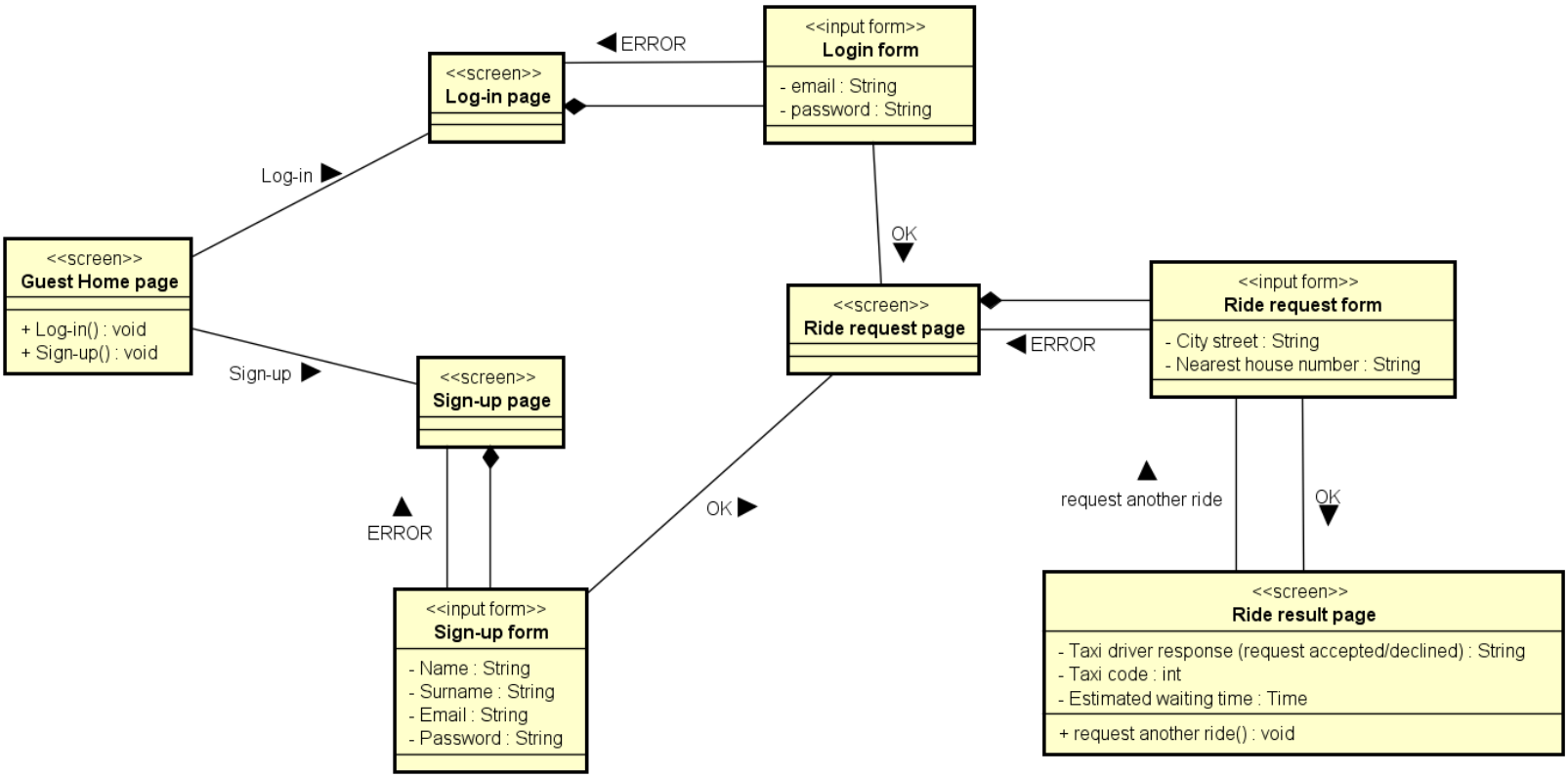
In this paragraph the user experience given by the system is described through a User Experience diagram.
The diagram is represented by extending the Class Diagram notation by with some stereotypes:

- <<screen>>: represents a web or mobile page of the application.
- <<screen compartment>>: represents a graphical component inside a page.
- <<pop-up>>: represents some information that pops out on the screen.
- << input form >>: represents a set of input fields or other input components inside a page.

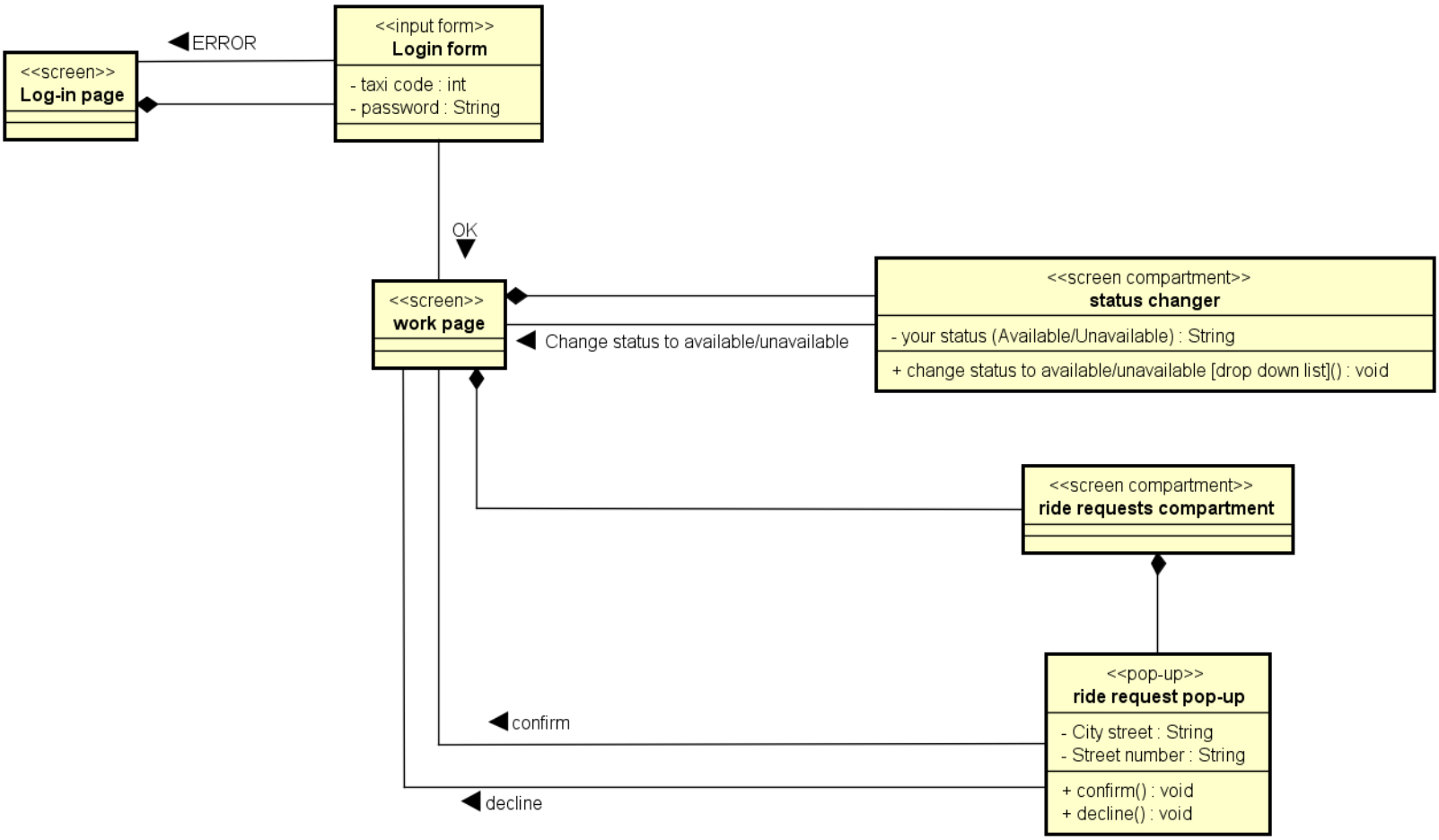
The arrows represent the flow followed by the user in the application by performing some operation.
Composition and aggregation relations are also used with their usual meaning, in order to represent the relationship among components of a page.

3.1.1 User web application and mobile app

User web application and mobile app user experience are represented with a unique diagram since are structured in the same way.



3.1.1 Taxi driver mobile app



4 Requirements traceability

Requirement		Design Solution	
		Components:	**Design solution overview**
R1	System shall store user information: name, surname, e-mail, password	• myTaxiService user web application/myTaxiService user mobile app • Central node: *Authenticator* subcomponent	Central node through its *Authenticator* subcomponent communicates with myTaxiService database
R2	System shall allow the user to sign-up into the system	• myTaxiService user web application/myTaxiService user mobile app • Central node: *Authenticator* subcomponent	Central node through its *Authenticator* subcomponent checks user input and if the user isn't already present in the database, commands user data storing
R3	System shall not allow a user to register himself more than one time with the same username	• myTaxiService user web application/myTaxiService user mobile app • Central node: *Authenticator* subcomponent	Central node through its *Authenticator* subcomponent queries the database and checks if the username already exists
R4	System shall check that username supplied in the registration must not be empty	• myTaxiService user web application/myTaxiService user mobile app	The user interface logic checks if the field is empty
R5	System shall check that password supplied in the registration must not be empty	• myTaxiService user web application/myTaxiService user mobile app	The user interface logic checks if the field is empty
R6	System shall allow the user to log-in into the system	• myTaxiService user web application/myTaxiService user mobile app • Central node: *Authenticator* subcomponent	Central node through its *Authenticator* subcomponent checks user input and checks if the user's data is in the database

R7	System shall allow the user to request a taxi ride	• myTaxiService user web application/myTaxiService user mobile app • Central node: *logged clients information handler* • Dispatcher • Business server of zone N	Central node receives the ride request from the myTaxiSevice user application and forwards it to the dispatcher, which will forward it to the business server of the right zone (see runtime view)
R8	If an user makes a request and a taxi driver accepts the request, the system shall notify the user that his request has been accepted	• myTaxiService user web application/myTaxiService user mobile app • myTaxiService taxis mobile app • Central node: *logged clients information handler* subcomponent • Dispatcher • Server of zone N	Server of zone N receives the confirmation/decline of some user's request from the myTaxiService taxis application and can send it to the myTaxiService user application
R9	In case a user request is accepted, system shall send the user the taxi code of the taxi serving the request and the estimated arriving time	• myTaxiService user web application/myTaxiService user mobile app • myTaxiService taxis mobile app • Central node: *logged clients information handler* subcomponent • Dispatcher • Server of zone N	Server of zone N receives the confirmation of some user's request from a taxi driver, the taxi driver app also sends taxi's driver taxi code, so it can be sent to the client together with the computed estimated waiting time

R10	In case a user request is confirmed, system shall prompt the user if he wants to confirm or decline the taxi ride	• myTaxiService user web application/myTaxiService user mobile app • myTaxiService taxis mobile app • Central node: *logged clients information handler* subcomponent • Dispatcher • Server of zone N	Server of zone N receives ride data from user myTaxiService application, so it can communicate with the user to prompt if he wants to accept/decline an offered ride
R11	The system shall decline a user's request if all the taxi drivers in the taxi drivers city zone queue have declined the request or the city zone queue is empty	• Server of zone N • myTaxiService taxis mobile app	If all the taxi drivers in the zone city queue of the server of zone N have declined the user request or if the city zone queue is empty, the server of zone N communicates the user that his request is declined
R12	System shall allow the taxi driver to log-in into the system	• myTaxiService taxis mobile app • Central node: *Authenticator* subcomponent • Dispatcher • Server of zone N	Central node through its *Authenticator* subcomponent checks user input and checks if the taxi driver's data is in the database
R13	System shall allow the taxi driver to inform the system about his availability at any time	• myTaxiService taxis mobile app • Central node: *logged clients information handler* subcomponent • Dispatcher • Server of zone N	Central node through is *logged clients information handler* subcomponent receives availability status from myTaxiDriver taxi drivers mobile app and then forwards the information to the dispatcher, which will then send it to the server of the right zone

R14	The system shall allow taxi driver to be only in “available” or “unavailable” status at any time	• myTaxiService taxis mobile app	The user interface logic only offers status’ choice between “available” or “unavailable”; moreover when the status is “available” taxi driver can only update his status to “unavailable” and vice versa
R15	If a taxi driver is in the “available” status must be put in the queue of only one city zone	• myTaxiService taxis mobile app • Server of zone N	When the server of zone N receives the taxi driver status and finds that it is put to “available”, inserts the taxi driver back into the queue
R16	If a taxi driver is in the “unavailable” status, he must not be put in any city zone queue	• myTaxiService taxis mobile app • Server of zone N	When the server of zone N receives the taxi driver status and finds that it is put to “unavailable”, removes the taxi driver from the queue
R17	If a taxi driver sets his status to “available” the system shall put him at the bottom of the queue which corresponds with the city zone where he is located	• myTaxiService taxis mobile app • Server of zone N	When the server of zone N receives the taxi driver status and finds that it is put to “available”, inserts the taxi driver back into the queue: at the bottom of it
R18	If a taxi driver is in the “available” status, system shall allow taxi driver to receive user requests and to confirm or decline user’s requests	• myTaxiService taxis mobile app • Central node: *logged clients information handler* subcomponent • Dispatcher • Server of zone N	When a user sends a ride request, server of zone N can retrieve the taxi driver from the queue and communicate with him about user’s request (see runtime view)
R19	System shall offer well documented Application Programming Interfaces in order to allow development of additional services	The whole system	The API are supplied offering a public access to the interfaces described in this document and creating a precise documentation on how to use them

R20	If a taxi driver declines a user's request, system shall put him to the bottom of the queue and shall put the second taxi driver of the queue in the first position	• Server of zone N: *Taxi driver handler* subcomponent	Every city-zone server handles his Taxi Driver queue through the *Taxi driver handler* subcomponent so that if a taxi driver declines a request, it is placed at the last position of the queue and other taxi drivers in the queue go up of a position

5 Appendix

For the creation of this document I spent 50 hours.