Politecnico di Milano

A.A. 2015-2016

Software Engineering 2 Project

myTaxiService

Design Document

Ver. 1

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.

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

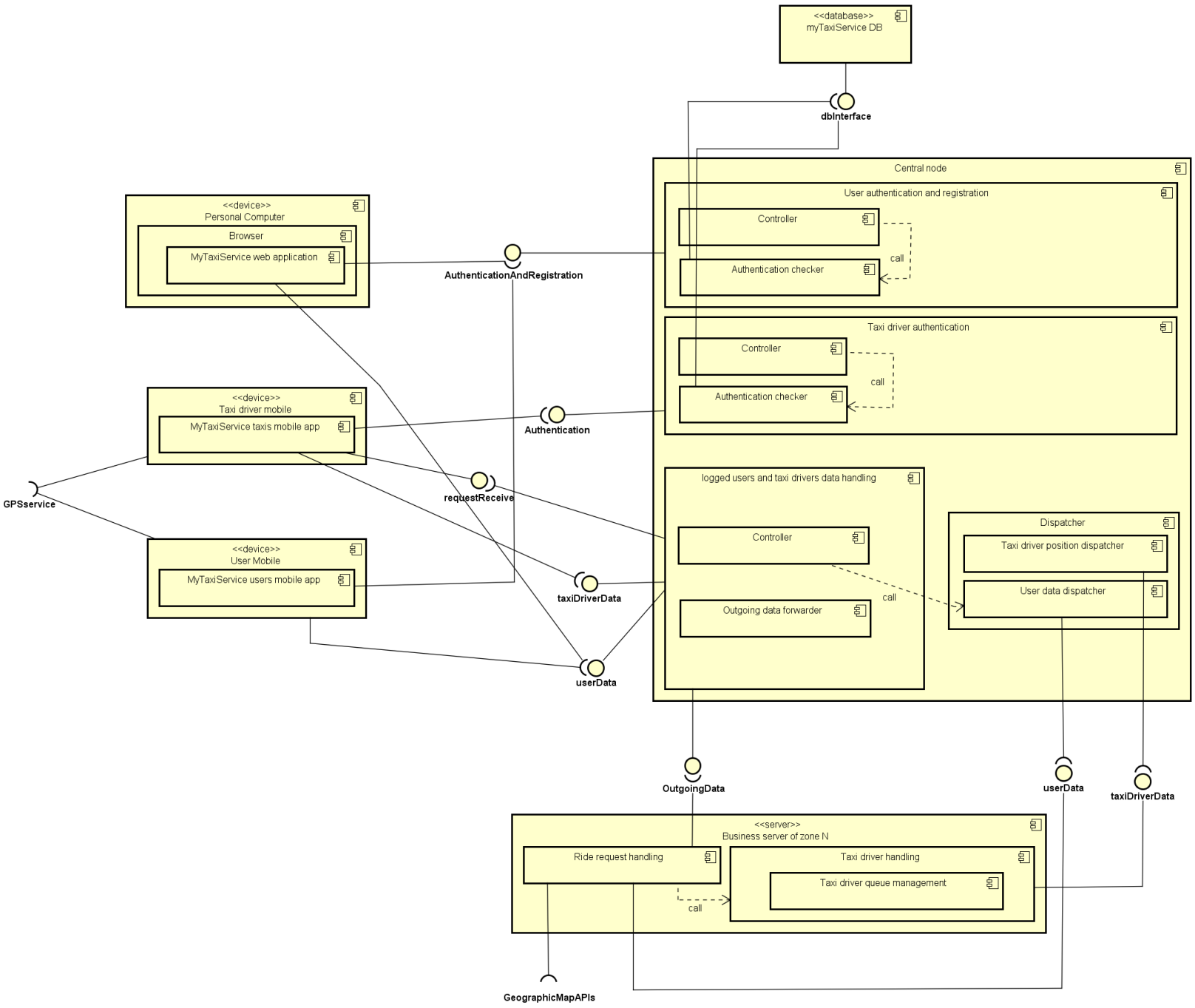
In this section high level components and their interactions will be analyzed using a table which associates a component and overview of its function. Components interacts among them using interfaces which are listed in section 2.6.

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 to 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, which finally computes the request and forwards data with the central node in order to communicate with user and taxi driver involved in the request.

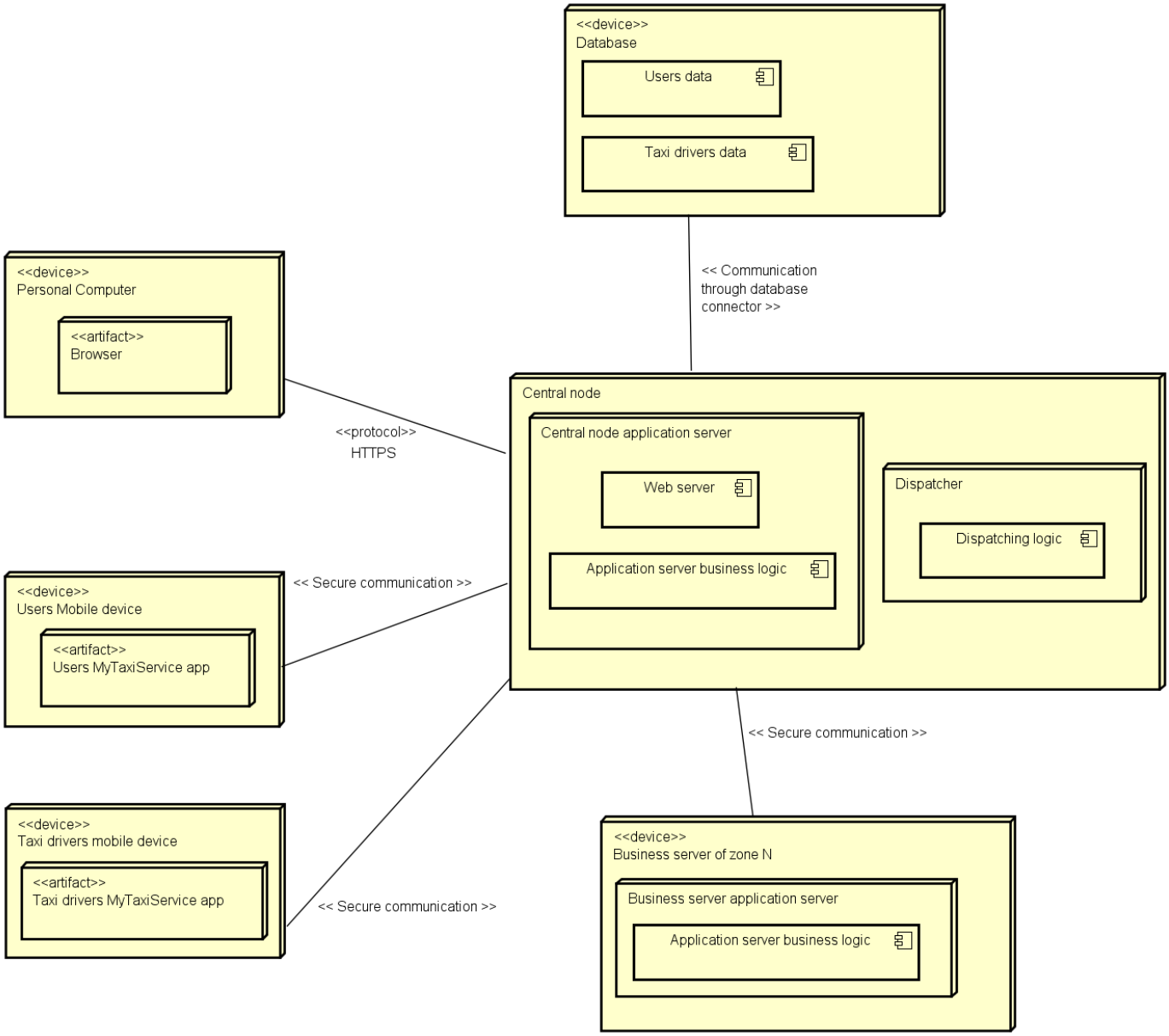

<<database>>
myTaxiService DB
dbInterface
Central node
User authentication and registration
Controller
call
Authentication checker
Taxi driver authentication
Controller
call
Authentication checker
logged users and taxi drivers data handling
Controller
call
Outgoing data forwarder
Dispatcher
Taxi driver position dispatcher
User data dispatcher
<<device>>
Personal Computer
Browser
MyTaxiService web application
AuthenticationAndRegistration
<<device>>
Taxi driver mobile
MyTaxiService taxis mobile app
Authentication
GPSservice
requestReceive
<<device>>
User Mobile
MyTaxiService users mobile app
taxiDriverData
userData
OutgoingData
userData
taxiDriverData
<<server>>
Business server of zone N
Ride request handling
call
Taxi driver handling
Taxi driver queue management
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, an application server which handle registration/authentication requests and ride requests making use of the information taken from the database and from the dispatcher, 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; lastly 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.

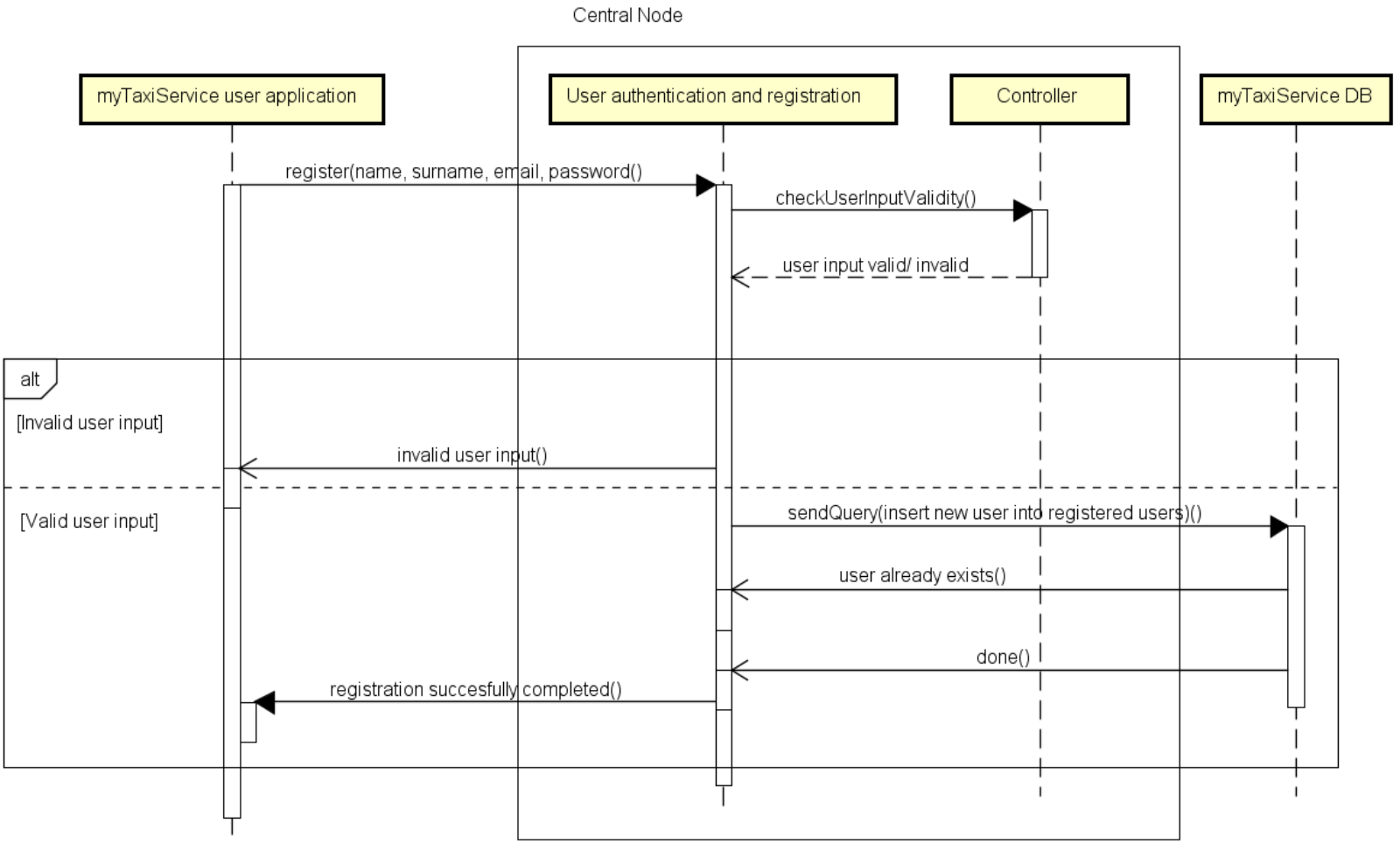

<<device>>
Database
Users data
Taxi drivers data
<< Communication through database connector >>
<<device>>
Personal Computer
<<artifact>>
Browser
<<protocol>>
HTTPS
Central node
Central node application server
Web server
Application server business logic
Dispatcher
Dispatching logic
<<device>>
Users Mobile device
<<artifact>>
Users MyTaxiService app
<< Secure communication >>
<< Secure communication >>
<<device>>
Taxi drivers mobile device
<<artifact>>
Taxi drivers MyTaxiService app
<< Secure communication >>
<<device>>
Business 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.

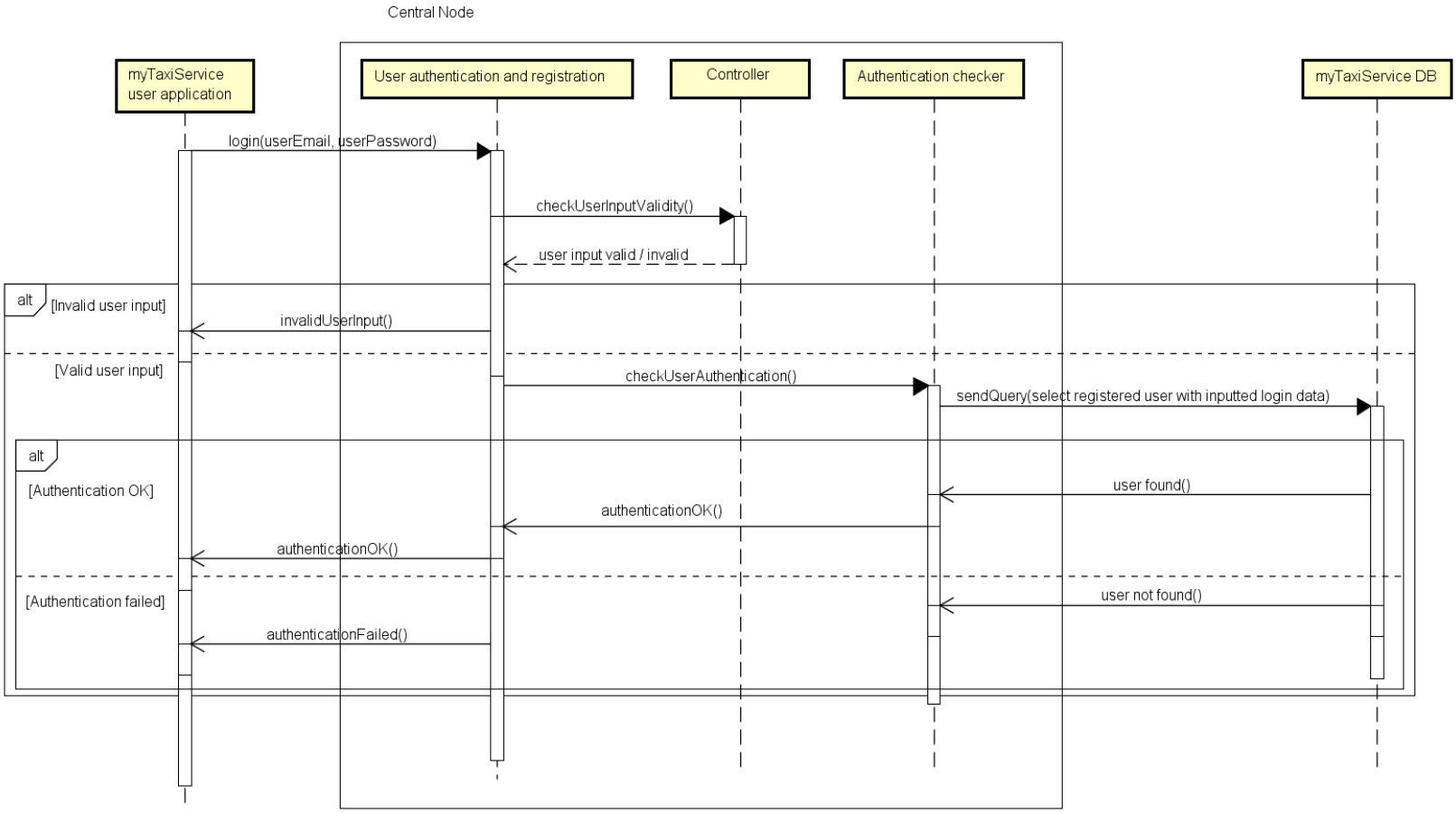
2.5.1 User Registration

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 User authentication and registration component using the Controller and then user data – if the user doesn't already exist – is added to the database.



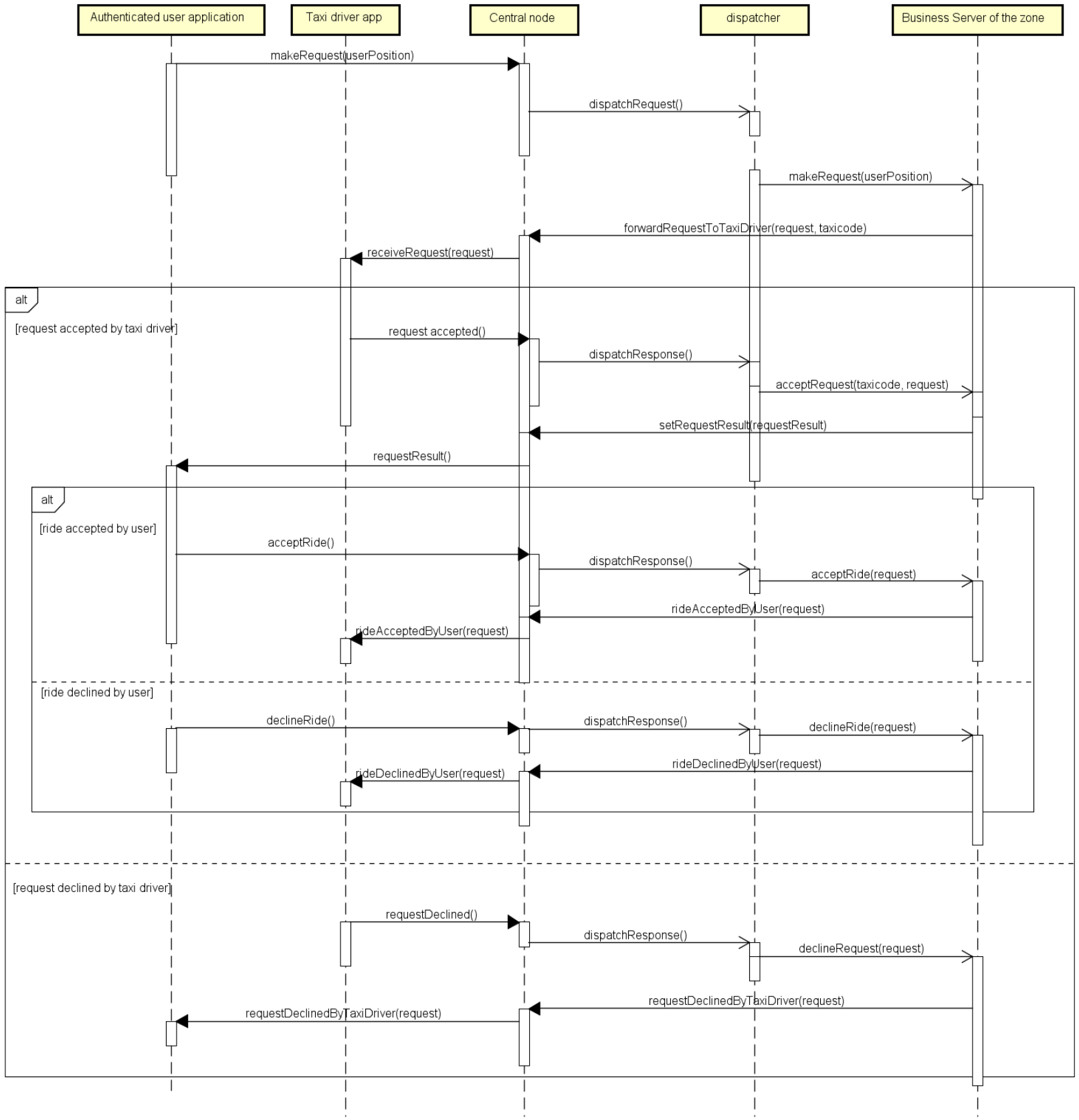
2.5.2 User Login

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 Authentication checker component, which interfaces with the database to check if the user wrote the right credentials.



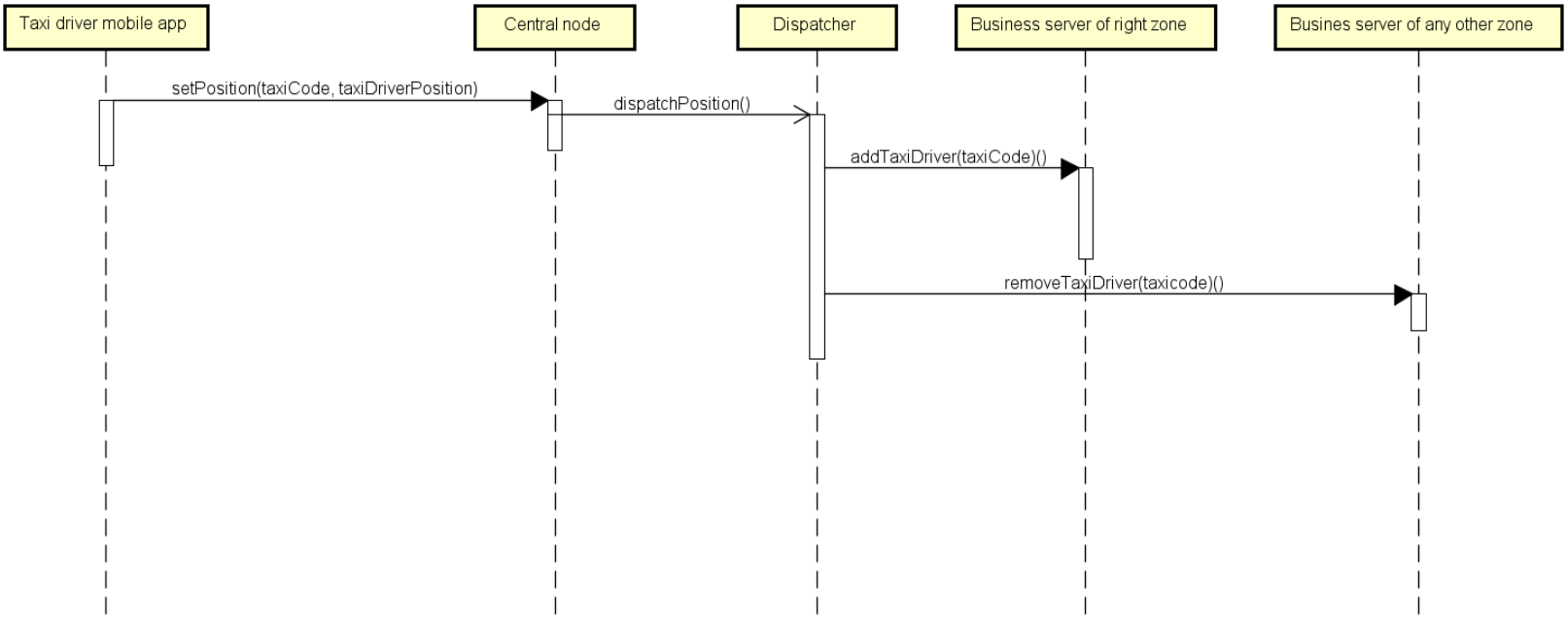
2.5.3 Ride request handling

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 handles the sessions with authenticated users and taxi drivers. Ingoing data, in fact, is forwarded to the business servers of the various zones by the dispatcher; after computation, business servers responses are then sent again to the central node, which can then answer to authenticated clients (users or taxi drivers).


Authenticated user application
Taxi driver app
Central node
dispatcher
Business Server of the zone
makeRequest(userPosition)
dispatchRequest()
makeRequest(userPosition)
forwardRequestToTaxiDriver(request, taxicode)
receiveRequest(request)
alt
[request accepted by taxi driver]
request accepted()
dispatchResponse()
acceptRequest(taxicode, request)
setRequestResult(requestResult)
requestResult()
alt
[ride accepted by user]
acceptRide()
dispatchResponse()
acceptRide(request)
rideAcceptedByUser(request)
rideAcceptedByUser(request)
[ride declined by user]
declineRide()
dispatchResponse()
declineRide(request)
rideDeclinedByUser(request)
rideDeclinedByUser(request)
[request declined by taxi driver]
requestDeclined()
dispatchResponse()
declineRequest(request)
requestDeclinedByTaxiDriver(request)
requestDeclinedByTaxiDriver(request)

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 User authentication and registration

interface authenticationAndRegistration {

register (Name name, Surname surname, Email email, Password password);

login (Email userEmail, Password userPassword);

}

2.6.1.2 Taxi driver authentication

interface authentication {

login (TaxiCode taxicode, Password password);

}

2.6.1.3 Logged users and taxi drivers data handling

interface taxiDriverData {

setPosition (TaxiCode taxicode, Position taxiDriverPosition);

setAvailability (TaxiCode taxicode, Boolean taxiDriverAvailability);

acceptRequest (TaxiCode taxicode, Request request);

declineRequest(TaxiCode taxicode, Request request);

}

interface userData {

makeRequest (Position userPosition);

acceptRide();

declineRide();

}

interface outgoingData {

setRequestResult (requestResult requestResult, Request request);

forwardRequestToTaxiDriver(Request request, TaxiCode taxicode);

rideAcceptedByUser(Request request);

rideDeclinedByUser(Request request);

requestDeclineByTaxiDriver(Request request);

avaibilityCorrectlySetted();

}

2.6.2 Business server

interface taxiDriverData {

addTaxiDriver (TaxiCode taxicode);

removeTaxiDriver (TaxiCode taxicode);

setAvailability (TaxiCode taxicode, Boolean taxiDriverAvailability);

acceptRequest (TaxiCode taxicode, Request request);

declineRequest(TaxiCode taxicode, Request request);

}

interface userData {

makeRequest (Position userPosition);

acceptRide (Request request);

declineRide(Request request);

}

2.6.3 myTaxiService taxis mobile app

interface requestReceive {

receiveRequest(Request request);

}

2.6.4 myTaxiService DB

interface dbInterface {

sendQuery(Query query);

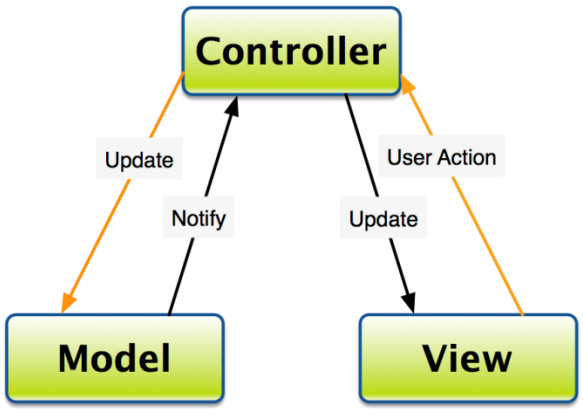
}

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.

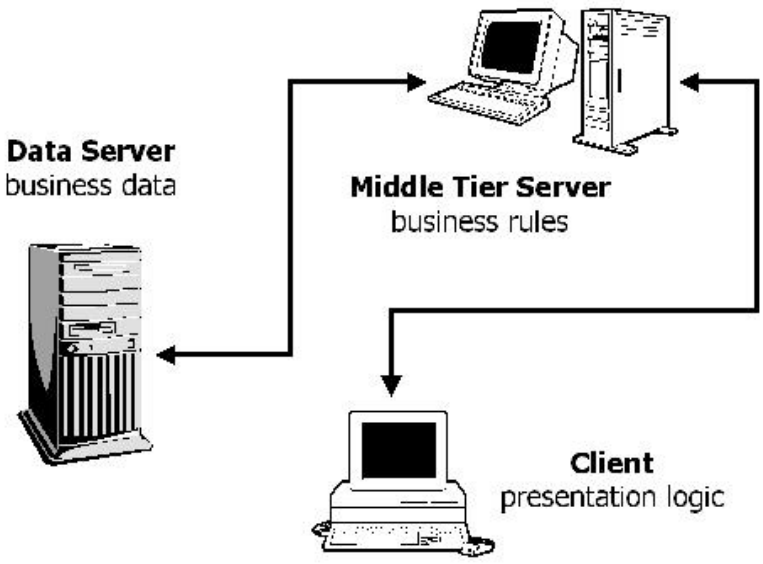


2.7.2 Three-tier Client/Server architecture

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

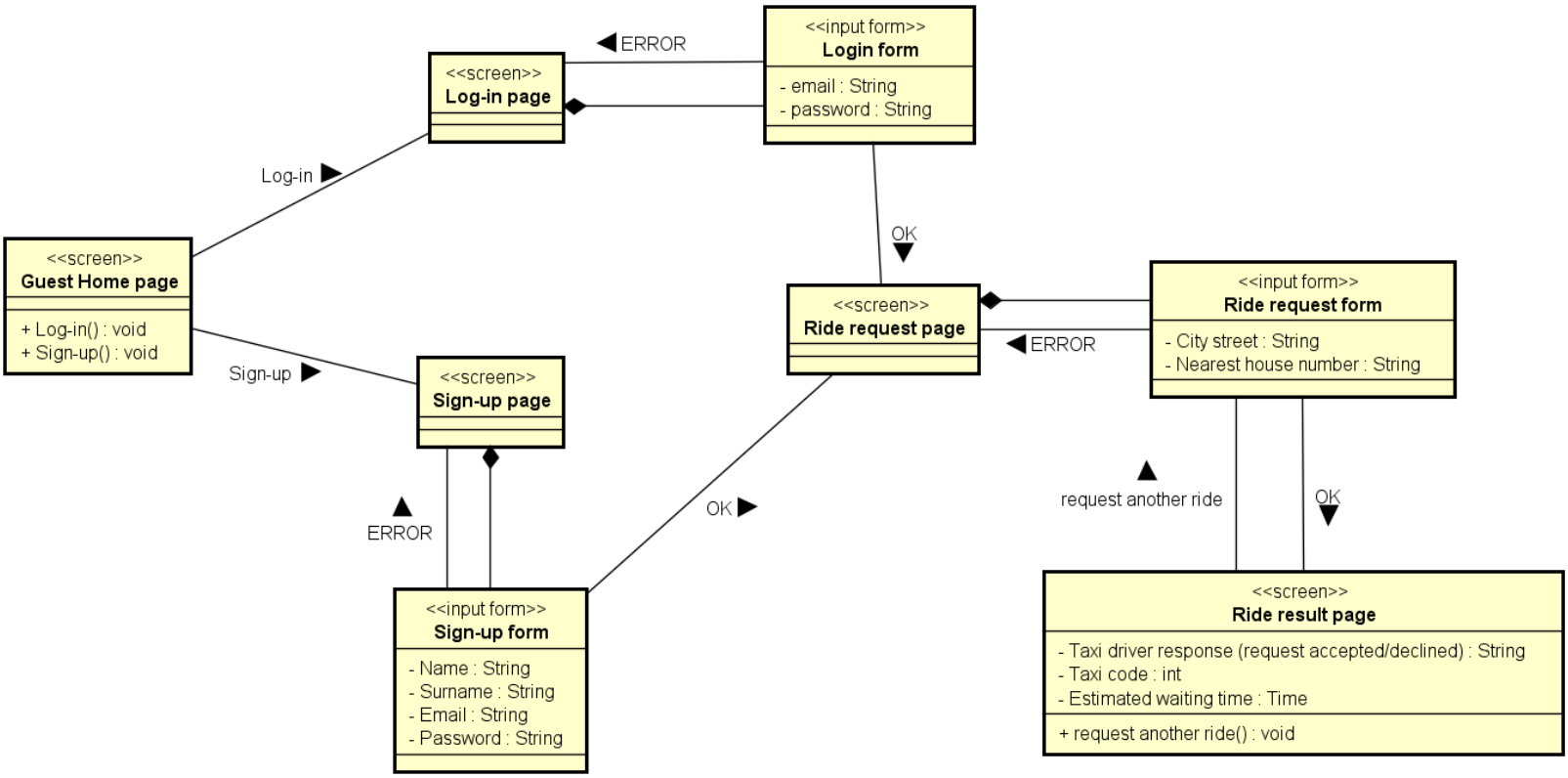
In this paragraph the user experience given by the system is described through a User Experience diagram.
The diagram is represented by extending the Class Diagram notation by with some stereotypes:

- <<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.

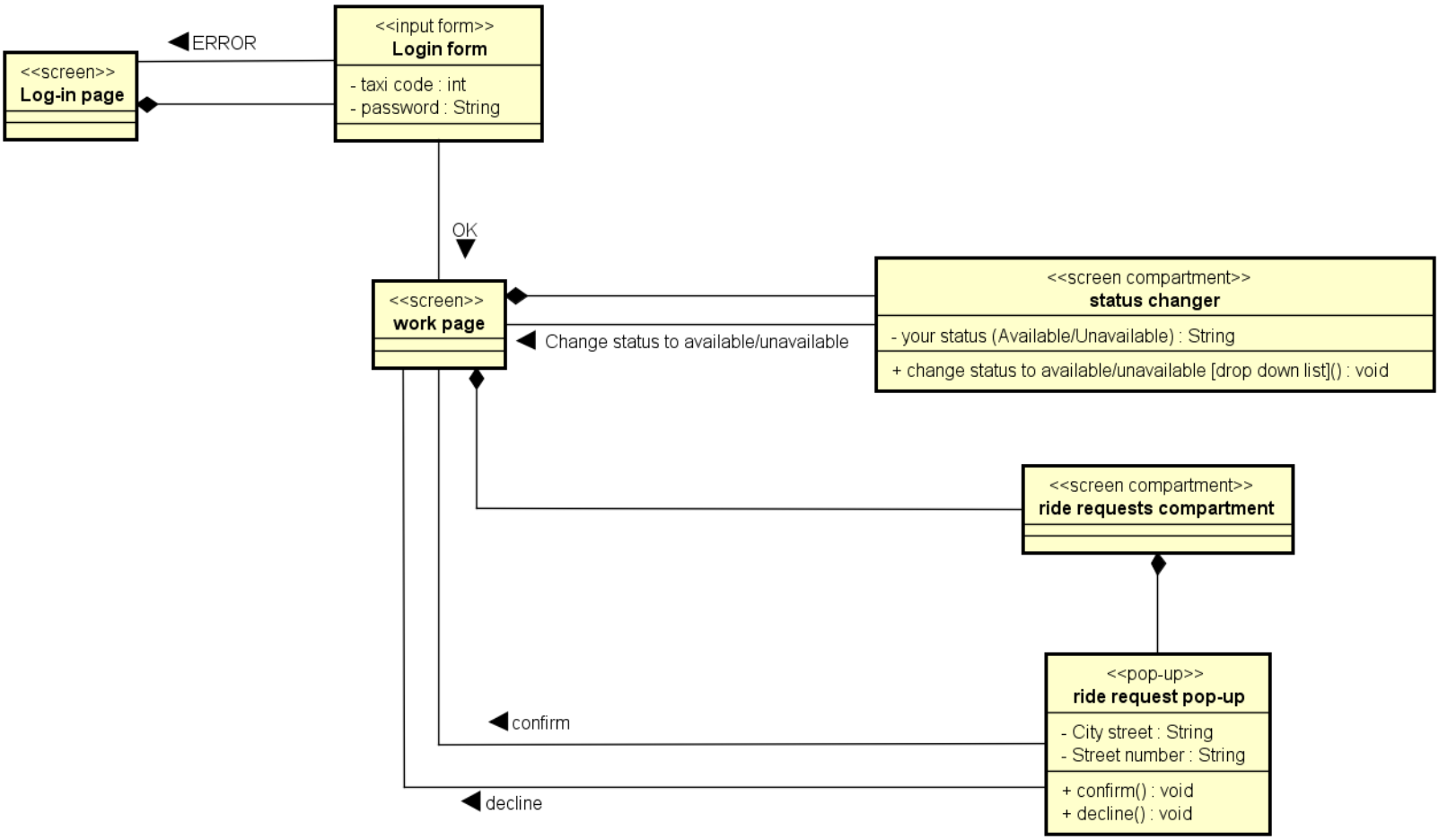
The arrows represent the flow followed by the user in the application by performing some operation.
Composition and aggregation relations are also used with their usual meaning, in order to represent the relationship among components of a page.

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

<table>
<tr><th colspan="2">Requirement</th><th colspan="2">Design Solution</th></tr>
<tr><td rowspan="2">R1</td><td rowspan="2">System shall store user information: name, surname, e-mail, password</td><td>Components:</td><td>Design solution overview</td></tr>
<tr><td>myTaxiService user web application/myTaxiService user mobile app
Central node: user authentication and registration subcomponent</td><td>Central node through its User authentication and registration subcomponent communicates with myTaxiService database</td></tr>
<tr><td>R2</td><td>System shall allow the user to sign-up into the system</td><td>myTaxiService user web application/myTaxiService user mobile app
Central node: user authentication and registration subcomponent</td><td>Central node through its User authentication and registration subcomponent checks user input and if the user isn’t already present in the database, commands user data storing</td></tr>
<tr><td>R3</td><td>System allow the user to log-in into the system</td><td>myTaxiService user web application/myTaxiService user mobile app
Central node: user authentication and registration subcomponent</td><td>Central node through its User authentication and registration subcomponent checks user input and checks if the user’s data is in the database</td></tr>
</table>

R4	System shall allow the user to request a taxi ride	• myTaxiService user web application/myTaxiService user mobile app • Central node: *logged users and taxi drivers data handling* subcomponent • 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
R5	System shall notify the user if his request has been confirmed	• myTaxiService user web application/myTaxiService user mobile app • myTaxiService taxis mobile app • Central node: *logged users and taxi drivers data handling* subcomponent • Dispatcher • Business server of zone N	Business Server receives the confirmation/decline of some user's request from the myTaxiService taxis application and can send it to the myTaxiService user application
R6	In case a user request is confirmed, system shall send to 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 users and taxi drivers data handling* subcomponent • Dispatcher • Business server of zone N	Business Server 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

R7	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 users and taxi drivers data handling* subcomponent • Dispatcher • Business server of zone N	Central node receives ride data from user myTaxiService application, so a user can accept/decline an offered ride
R8	System shall allow the taxi driver to log-in into the system	• myTaxiService taxis mobile app • Central node: *Taxi driver authentication* subcomponent • Dispatcher • Business server of zone N	Central node through its *Taxi driver authentication* subcomponent checks user input and checks if the taxi driver's data is in the database
R9	System shall allow the taxi driver to inform the system about his availability	• myTaxiService taxis mobile app • Central node: *logged users and taxi drivers data handling* subcomponent • Dispatcher • Business server of zone N	Central node through is *logged users and taxi drivers data handling 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 right business server of zone N

R10	System shall allow the taxi driver to receive user requests from the system	• myTaxiService taxis mobile app • Central node: *logged users and taxi drivers data handling* subcomponent • Dispatcher • Business server of zone N	When a user sends a ride request, business server can retrieve the taxi driver from the queue and send the response to his myTaxiDriver taxis mobile app
R11	System shall allow the taxi driver to confirm or decline users requests	• myTaxiService taxis mobile app • Central node: *logged users and taxi drivers data handling* subcomponent • Dispatcher • Business server of zone N	Taxi driver can communicate confirmation or decline of user's request at the Central node, which forwards it to the dispatcher, which send it to the business server
R12	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.
R13	System shall guarantee a fair management of taxi queues	• Business server of zone N: *Taxi driver handling* subcomponent	Every city-zone server handles his Taxi Driver queue through the *Taxi driver handling* 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.