POLYTECHNIC UNIVERSITY OF MILAN

School of Industrial and Information Engineering

Computer Science and Engineering

Project of Software Engineering 2: MyTaxi Service Design Document

Course Professor: Prof. Elisabetta DI NITTO

Authors:

Mattia CRIPPA 854126

Francesca GALLUZZI 788328

Marco LATTARULO 841399

Academic Year 2015–2016

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

The *Design Document* is a document meant to provide documentation which will be used to help developers in implementing the entire system by providing a general description of the architecture and the design of the system to be built. Within the Design Document are narrative and graphical documentation of the software design for the project including user experience diagrams, sequence diagrams, entity-relation diagrams, component diagrams, and other supporting requirement information.

1.1 Purpose

The purpose of the Design Document is to provide a description of the system specified in the *RASD* complete and detailed enough to allow the proceeding of the software development with a good understanding of which are the components of the system, how they interact, which is their architecture and how they will be deployed.

1.2 Scope

This document refers to the developing of an application called *MyTaxiService*, which is aimed to improve the quality and the efficiency of the taxi service of a large city by using localization, smartphones and IT technologies.

1.3 Definitions, Acronyms, Abbreviations

Definitions

- Session Bean: is a component of the application logic used to model business functions.
- Stateless Session Bean: no state is maintained with the client.
- Stateful Session Bean: the state of an object consists in the values of its instance variables. They represent the state of a unique client/bean session. When the client terminates, the bean is no longer associated with the client.
- Singleton Session Bean: is instantiated once per application and exists for the whole application lifecycle. A single bean instance is shared across and concurrently accessed by clients.
- Java Server Faces: a component-based MVC framework built on top of the Servlet API.

Acronyms and Abbreviations

- RASD: Requirements Analysis and Specification Document
- Java EE: Java Enterprise Edition.
- JSF: Java Server Faces.
- REST: Representational State Transfer.
- XHTML: Extensible HyperText Markup Language.
- EJB: Enterprise Java Beans.
- UX Diagram: User Experience Diagram.

1.4 Reference Documents

- Specification Document: MyTaxiService Project AA 2015-2016.pdf.

- IEEE Std 1016tm-2009 Standard for Information Technology - System Design - Software Design Descriptions.

- RASD v2.0 - CrippaGalluzziLattarulo.pdf

1.5 Document Structure

While the RASD is written for a more general audience, this document is intended for individuals directly involved in the development of MyTaxiService application. This includes software developers, project consultants, and team managers. This document is not meant to be read sequentially; users are encouraged to jump to any section they find relevant. Below is a brief overview of each part of the document.

- **Section 1** → Introduction: This section gives general information about the Design Document of the MyTaxiService project.

- **Section 2** → Architectural Design: This section contains an overall view of the system, describing from different points of view all the components that are part of the system and their interaction. This Section also contains a short explanation about the selected architectural system and the pattern that have been chosen.

- **Section 3** → Algorithm Design: This section contains the definition of any algorithm that is important to describe the system.

- **Section 4** → User Interface Design: This section covers all of the details related to the structure of the graphical user interface (GUI). Readers can view this section for a tentative glimpse of what the final product will look like.

- **Section 5** → Requirements Traceability: This section explain how the requirements defined in the RASD map into the design elements that have defined in this document.

- **Section 6** → References: This section includes any additional information which may be helpful to readers.

Chapter 2

Architectural Design

The System Architecture is a way to give the overall view of a system and to put it in relation to external systems. This allows the reader to have a more complete and general idea of the entire system and at the same time to have a deeper view of the principal components of the system itself.

2.1 Overview

This section provides a general description of the architecture of our system. The system has a 4-tier architecture, following the common Java EE architecture, in which the presentation relies upon the client machines, the server machine takes care of the business logic and the web tier and on a third dedicated machine resides the database. In this document the web application and the Android mobile application are treated as one entity, so all the communication between client and server will pass through the Web Tier. JSF technology will be used for dynamic web pages and an implementation of the REST paradigm will be assumed for communicating with the Android app.

More in details JEE has a four tiered architecture divided as:

- **Client Tier**: This tier contains Application Clients and Web Browsers, and it is the layer that interacts directly with the actors. All the presentation is inside this tier.
- **Web Tier**: This tier manages all the requests that are sent by the client tier, and forwards this requests to the business tier. Symmetrically, it

elaborates all the contents generated by the business tier and sends these contents to the client tier in a proper way (so that the Web browser or the Application client can render all the information).

- **Business Tier**: This tier is responsible for all the elaboration of information and represents the core controller of the entire system. All the application logic resides here under the form of Enterprise Java Beans and Java Entities. This tier is connected to the Database through a Java Persistence API.

- **Data Tier**: Is the main storage for the entire system and usually consists at least of a Database in which all the persistent information needed by the system are stored.

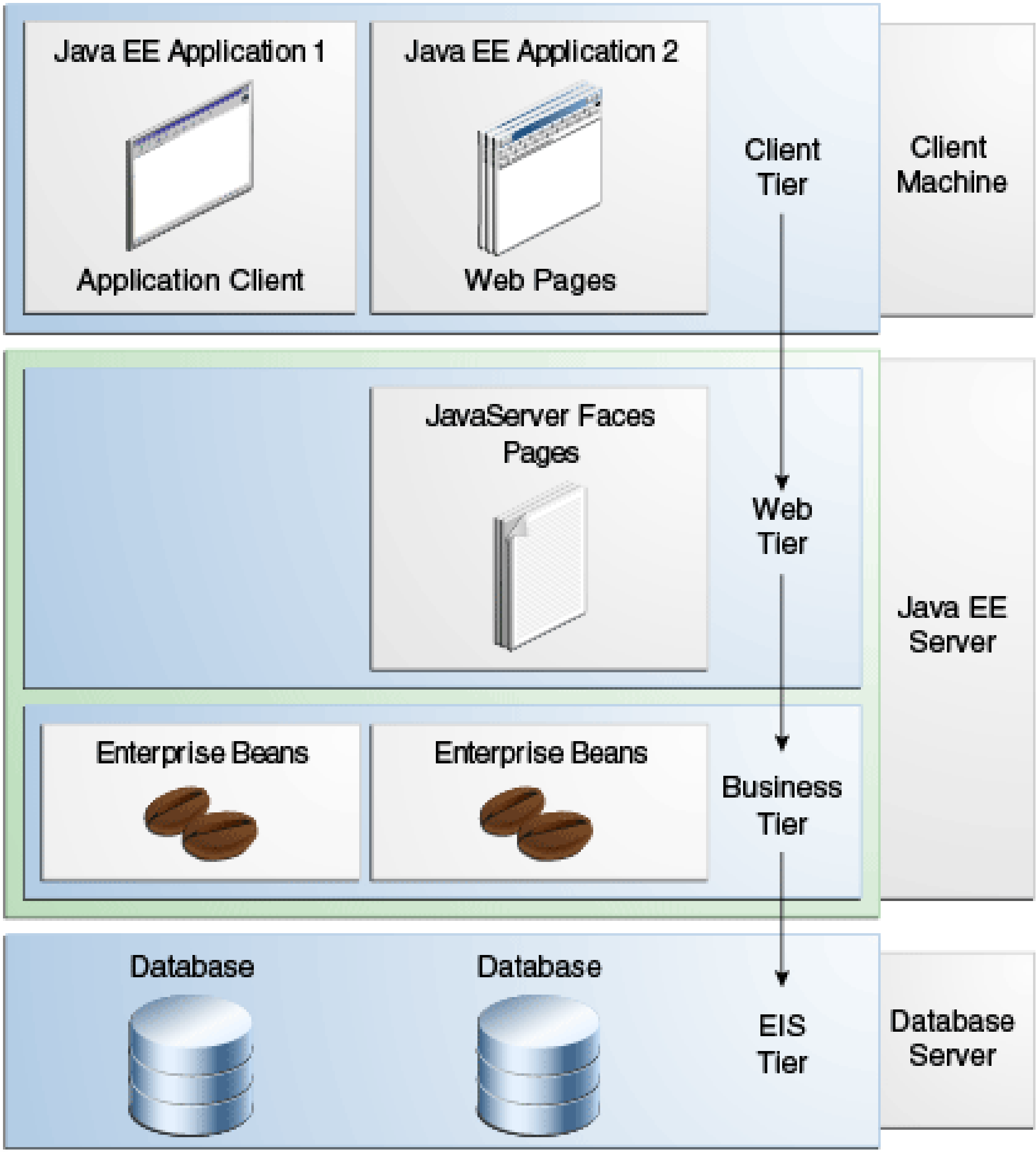

Figure 2.1: JEE 4-tier architecture

2.2 High Level Components and their Interaction

The diagram in Figure 2.2 represents our conceptual high level architecture of the MyTaxiService system.

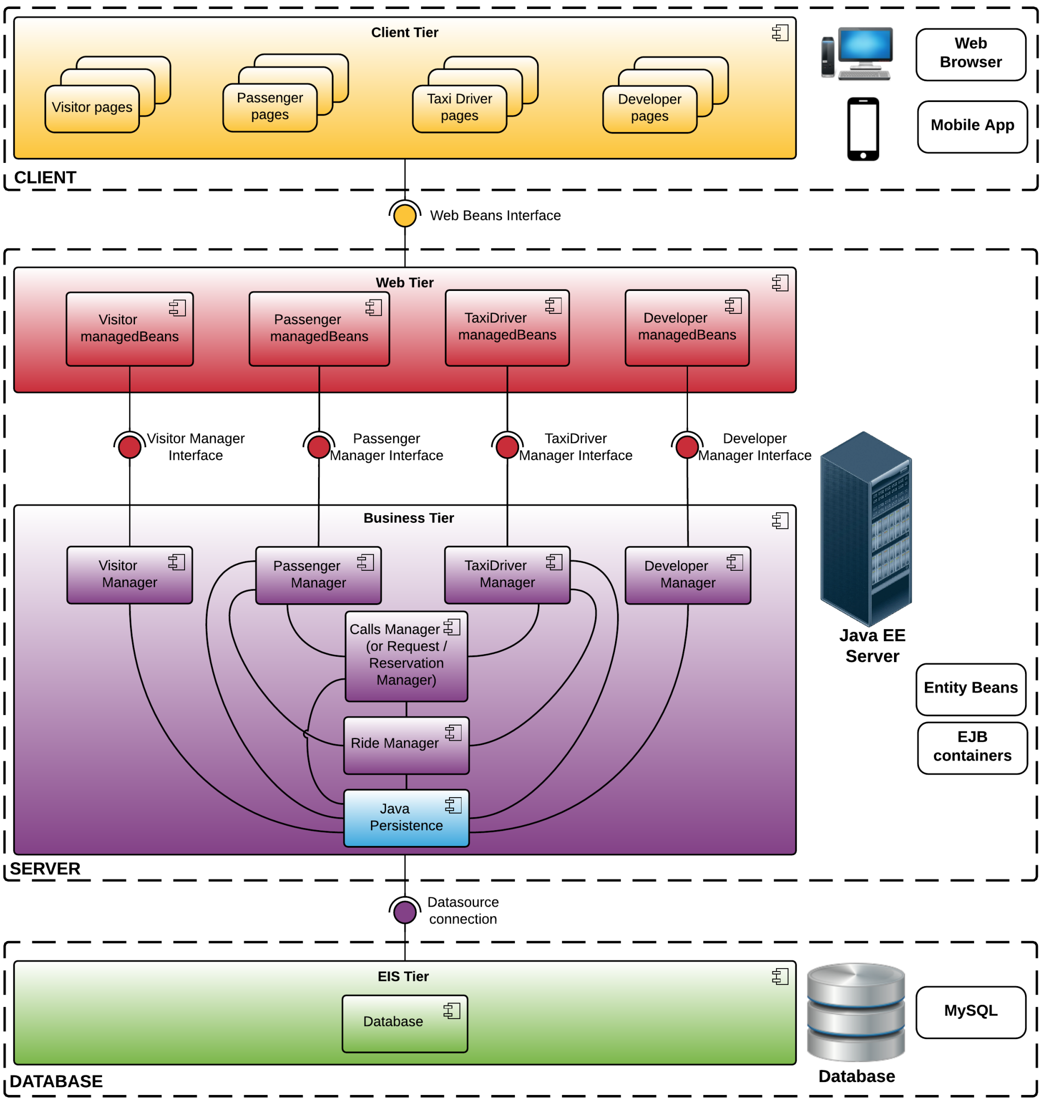

Figure 2.2: High Level Components view and their interaction

2.3 Component View

2.3.1 Client Component

The first component inside the system is the Client component which is responsible of translating user actions and presenting the output of tasks and results into something the user can understand. This component present different interfaces that allows each user to visualize the right pages. Each interface is a subcomponent of the Client components and contains different pages, so different users can visualize different contents with respect to their type.

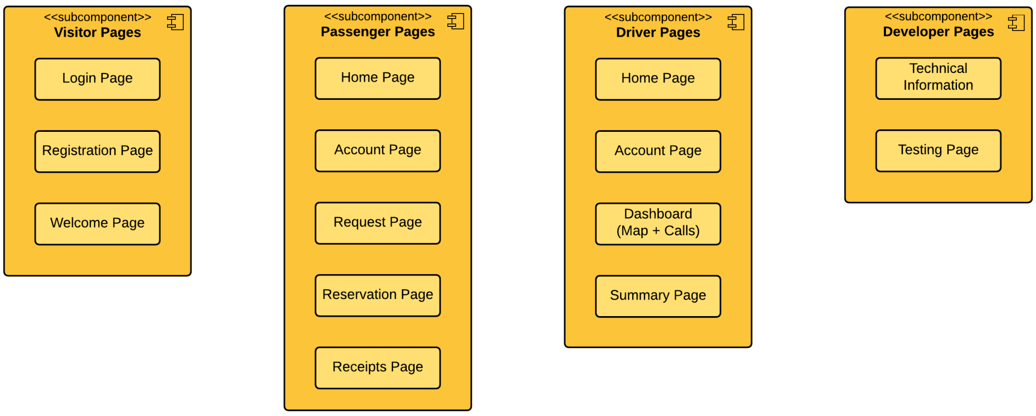

Figure 2.3: Client subcomponents

2.3.2 Web Component

The Web component generates dynamic web pages containing XHTML. Web components are developed with Java Server Faces technology, which is a common user interface component framework for web applications.

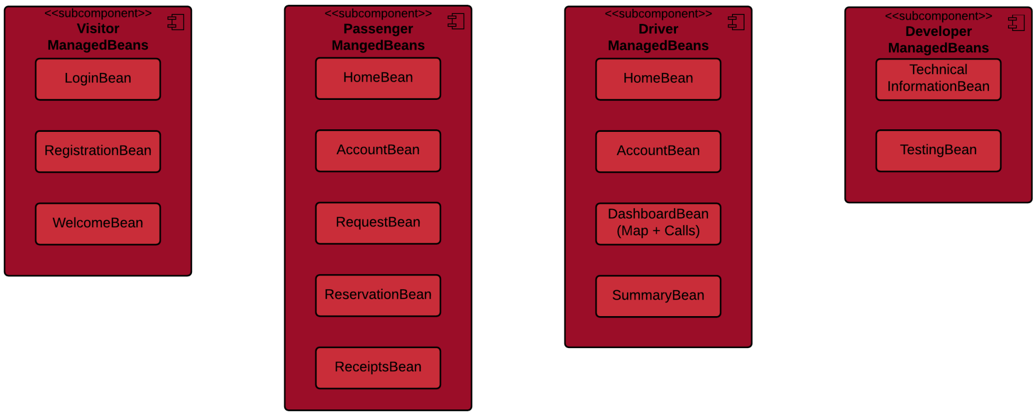

Figure 2.4: Web subcomponents

2.3.3 Business Logic Component

Another component of the system is the Business Logic component, which coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the Client and the Java Persistence, which holds the information of the system data model, and is in charge of storing and retrieving information from a database. More in detail:

- Visitor Manager → Offers functionalities to:
  - Check the validity and correctness of the information provided by the user
  - Create new users and save them into the system;
  - Check if the Login is valid and authenticate users;
  - Trigger the right user manager depending on the type of user that has logged in.
- Passenger Manager → Manages the passenger requests (taxi requests, reservations), the passenger profile and his status.
- Taxi Driver Manager → Manages all the operations made by taxi drivers, like accepting or rejecting incoming calls or ending rides
- Developer Manager → Manage all the operations made by developers (add new features, update the system code and architecture)
- Ride Manager → Offers functionalities to:
  - Create and manage the route for the ride;
  - Keep track of the passengers and the driver involved in the ride, and all the information: duration, distance, fee, route for the entire ride and for each passenger.
- Call Manager → Manages all the passenger's requests/reservations, the taxi queue for every area and the matching for show rides.

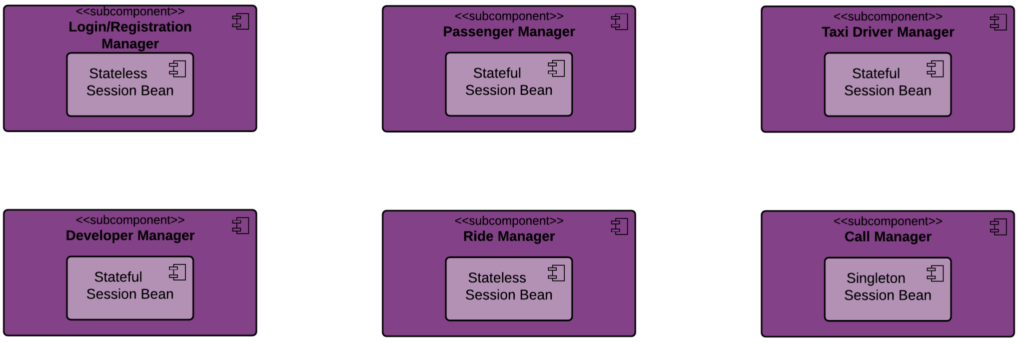

Figure 2.5: Business Logic subcomponents

2.3.4 Database Component

The last component of the system is represented by the Database that contains all the data that are interesting for this application.

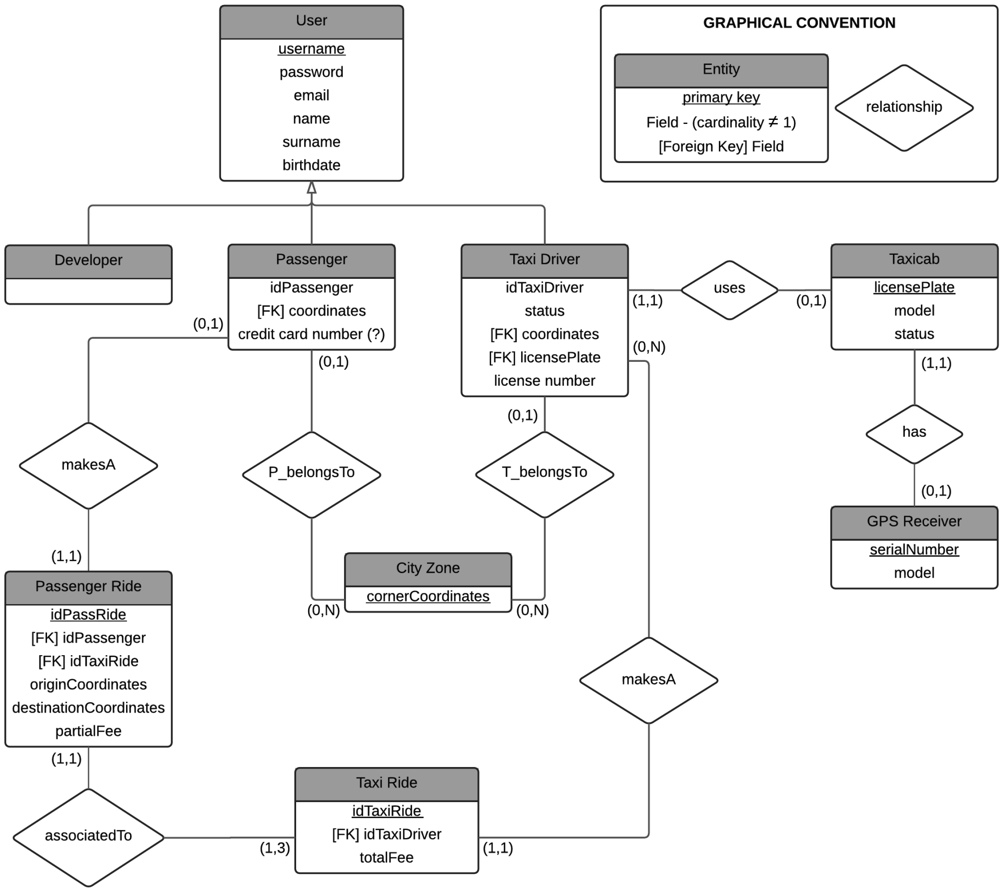

Figure 2.6: Database ER Diagram

2.4 Deployment View

The diagram in Figure 2.7 shows the deployment view of the software product.

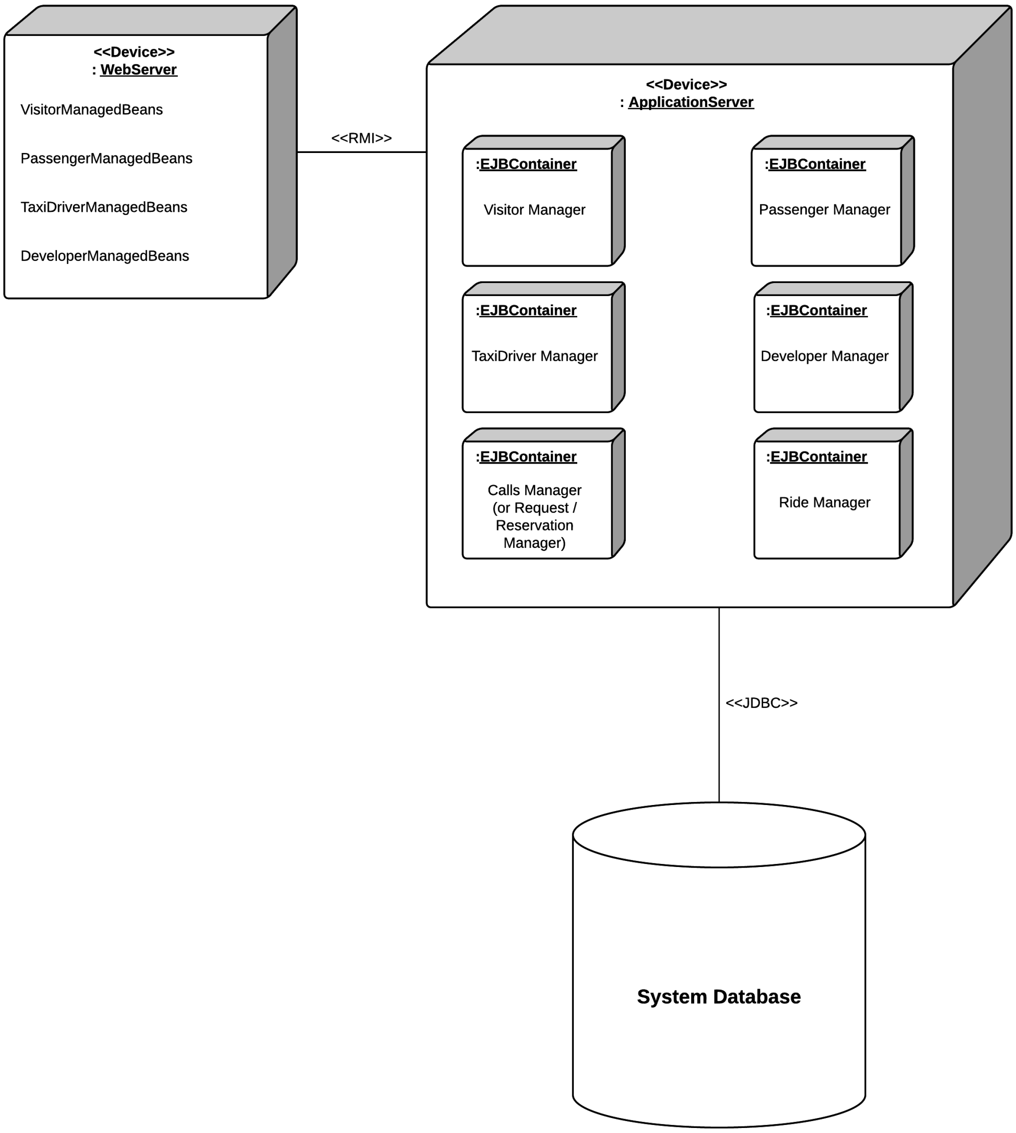

Figure 2.7: Deployment view

2.5 Runtime View

The following diagrams describes the runtime view of MyTaxiService project describing the way components behave in order to accomplish the most important activities of the system. The software product that will be released can be deployed in any JEE Application Server:

- This diagram represents the components that are interested in the taxi request and reservation activities, and their interaction.

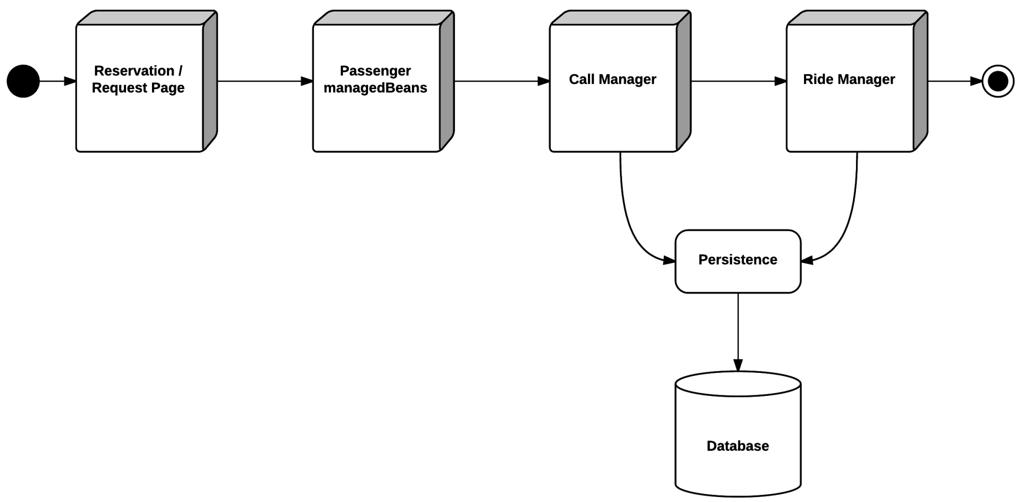

Figure 2.8: Runtime Taxi Request and Reservation

- This diagram represents the components that are interested in the get receipts activity, and their interaction.

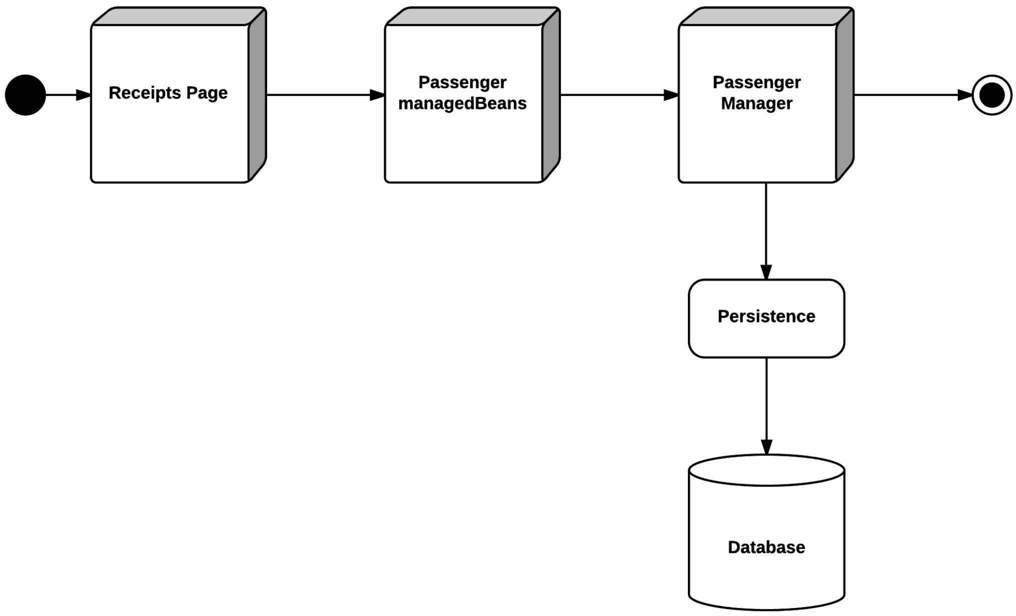

Figure 2.9: Runtime get receipts

- This diagram represents the components that are interested in the modify passenger's account activity, and their interaction.

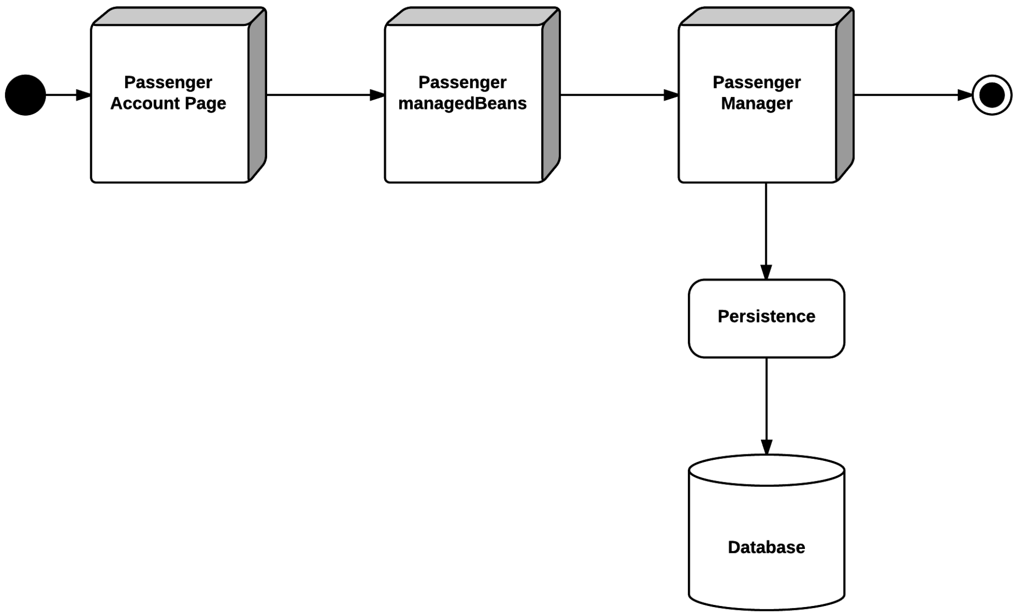

Figure 2.10: Runtime Modify Passenger's Account

- This diagram represents the components that are interested in the modify taxi driver's account activity, and their interaction.

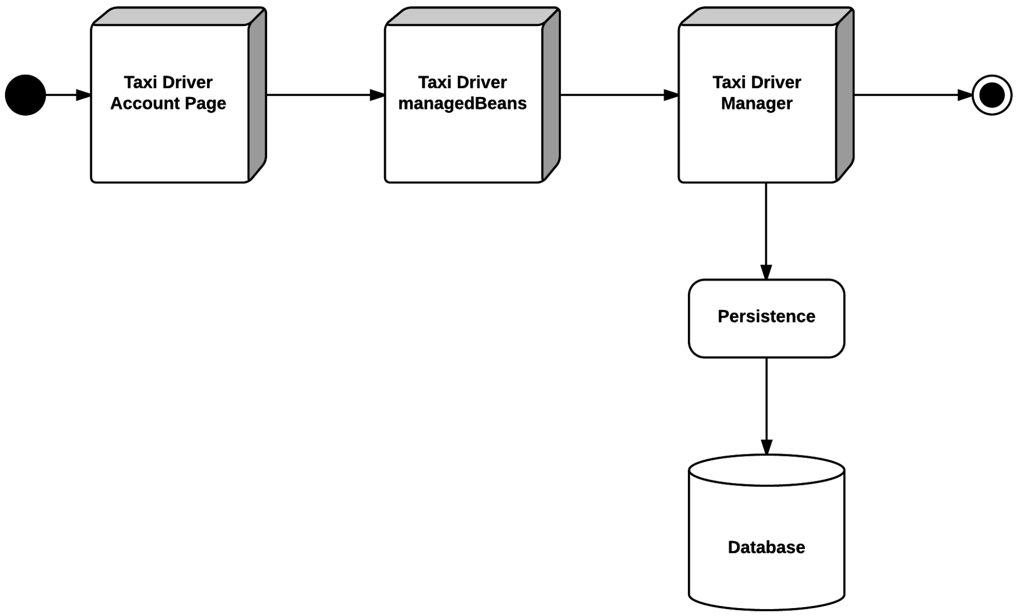

Figure 2.11: Runtime Modify Taxi Driver's Account

- This diagram represents the components that are interested in the login and registration activities, and their interaction.

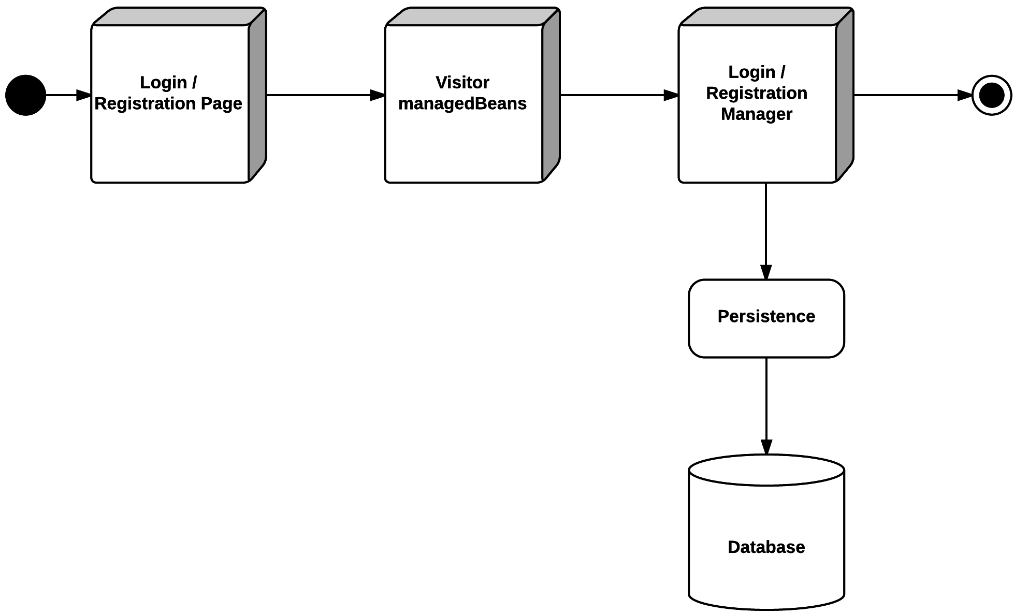

Figure 2.12: Runtime Login and Registration

- This diagram represents the components that are interested in the start taxi ride activity, and their interaction.

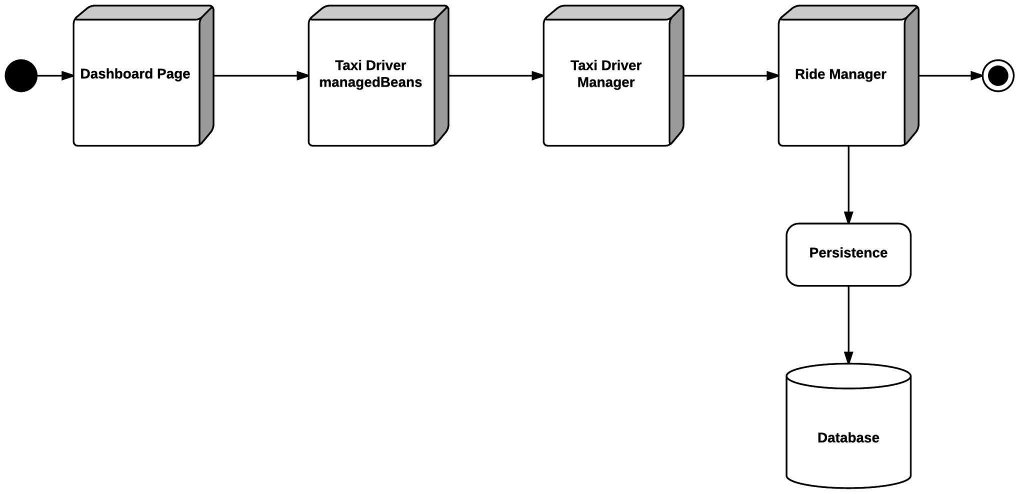

Figure 2.13: Runtime start Taxi Ride

- This diagram represents the components that are interested in the summary activity, and their interaction.

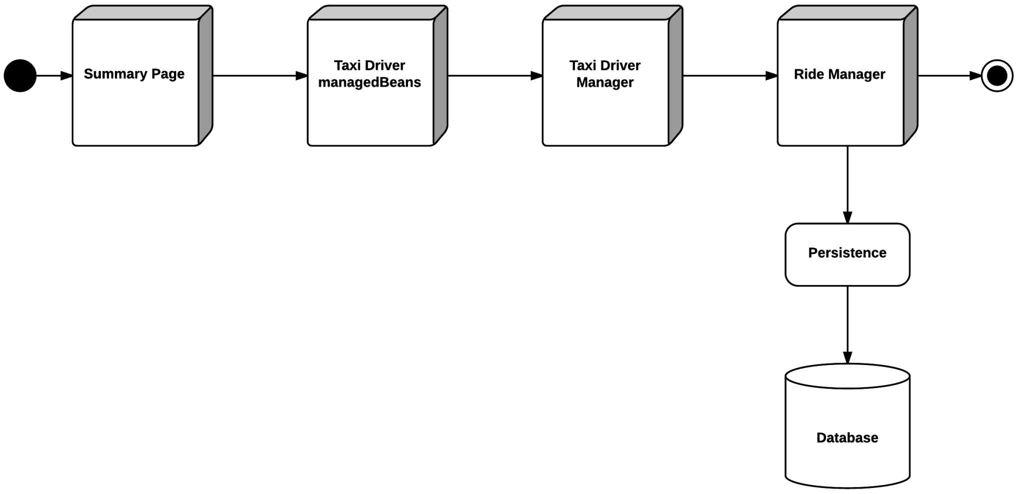

Figure 2.14: Runtime Summary

2.6 Component Interfaces

2.7 Selected Architectural Styles and Patterns

Chapter 3

Algorithm Design

Queue Management:

It's possible to create a queue implementing the Java Interface Queue, which is a collection designed for holdings elements prior to processing. Besides basic Collection operations, queues provide additional insertion, extraction, and inspection operations. Queues typically order elements in a FIFO (first-in-first-out) manner. The *head* of the queue is that element which would be removed by a call to **remove()** or **poll()**. In a FIFO queue, all new elements are inserted at the *tail* of the queue. This Interface provides some methods to manage the queue:

- **add(e)** → Method that allow to insert a new element e in the queue
- **remove()** and **poll()** → Methods that remove and return the head of the queue. Methods differ only in their behavior when the queue is empty: the remove() method throws an exception, while the poll() method returns null.
- **element()** and **peek()** → Methods that return, but do not remove, the head of the queue.

Research of a Taxi Driver in the queue:

The system uses a FIFO queue to manage the requests to the taxi drivers. It will use the method element() in order to extract without removing the first taxi driver of the queue and it will send the request to him. If the first taxi driver accepts the request, this will be assigned to him and he will be dequeued, otherwise he will be dequeued and the system will use the new first element of the queue to detect the new taxi driver who will receive the request, and so on until a taxi driver accepts the request.

Algorithm 1 Research of a Taxi Driver

```
procedure SearchTaxiDriver(Q)
    if (Q.head == Q.tail) then                          ▷ Queue is empty
    else
        for i := 0 to Q.lenght do
            x ← Dequeue(Q)          ▷ Classical operation for managing the
extraction of an element from the Queue
            acceptance ← Call(x)              ▷ Function that represent
the call that the system makes to the driver in order to propose a ride. It
returns true if the taxi driver accepts the call and is willing to make the
ride, otherwise returns false if the taxi driver declines the call
            if (acceptance == true) then
                return x
            else
                Enqueue(Q, x)       ▷ Classical operation for managing the
insertion of an element from the Queue
            end if
        end for
    end if
end procedure
```

Shared Ride Management:

This is the algorithm that is responsible to manage the Shared Rides. Here there are some general information about the algorithm:
There is one sharing list per every taxi area ordered by the starting time of the ride. Match between elements of the list is done under a time window of 10 minutes after the selected starting time (for reservation) and after the moment of the call (for request).When is time to start assigning a driver to the request/reservation and start the ride, the system deletes the corresponding

element from the list (is always the head of the list because of the ordering). There is also a buffer (Figure 3.1) to make sequential the adding of new requests/reservations to the list and avoid conflicts.
The flowchart diagram in Figure 3.2 represent the generic algorithm which describes how the system manage the functionality of "Ride Sharing".

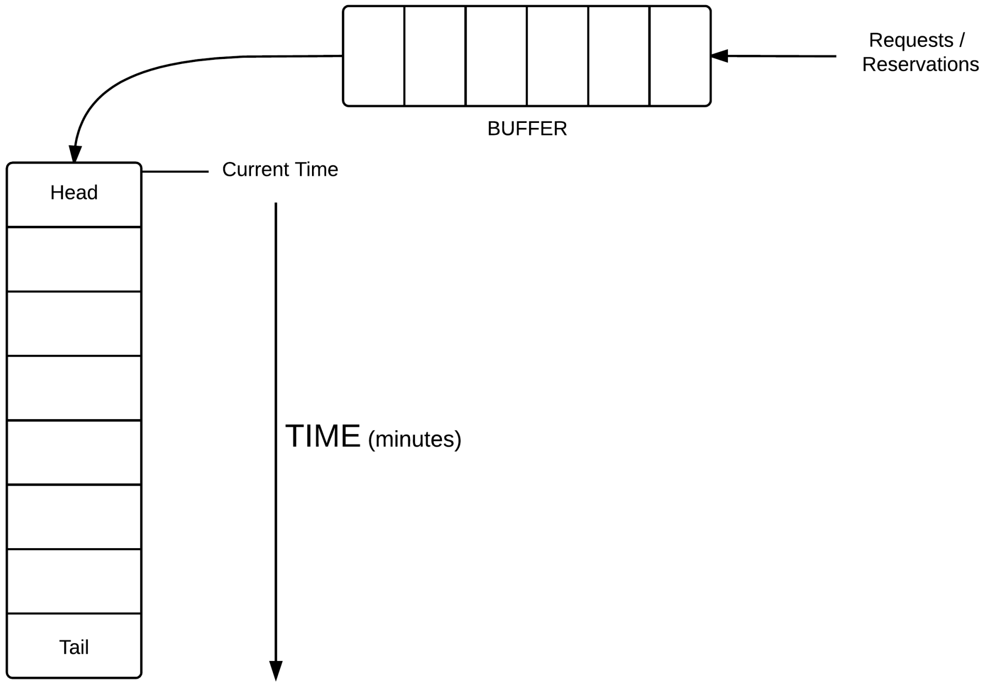

Figure 3.1: Representation of interaction between Queue and Buffer

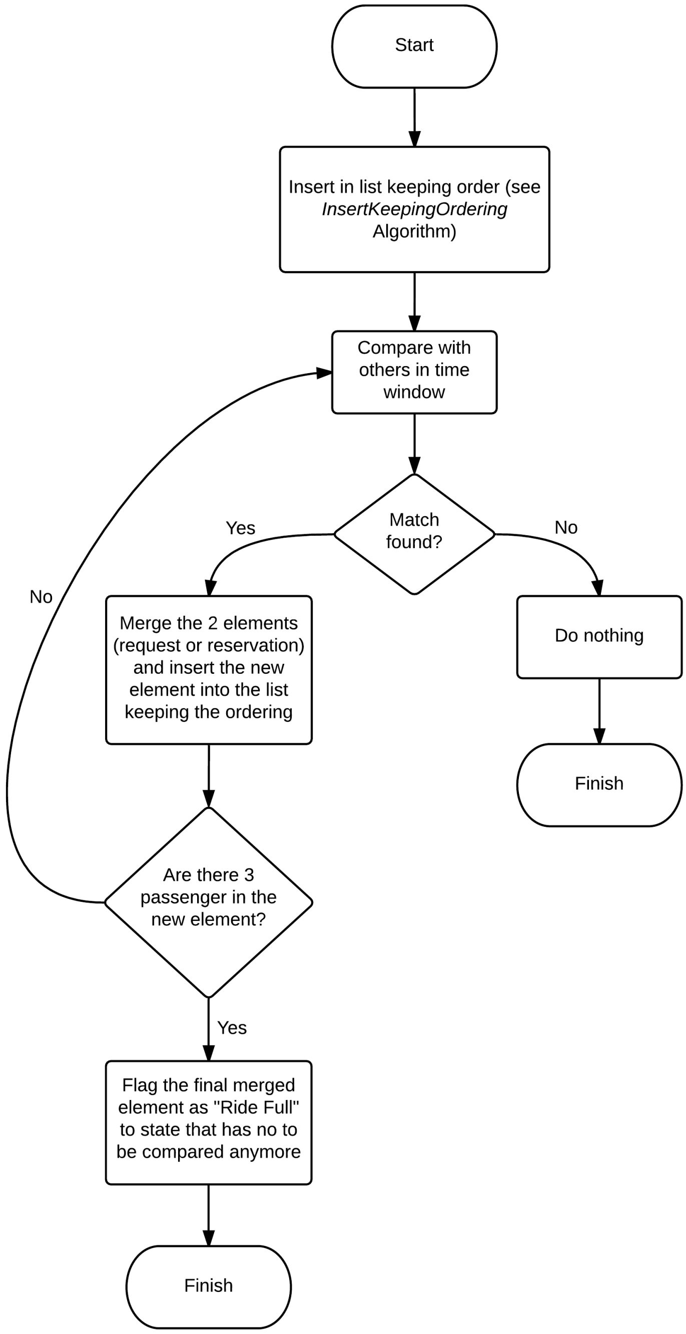

Figure 3.2: Representation of the Shared Ride Management Algorithm

The following algorithms describe in details the behavior of the list and how the Shared Ride Management Algorithm works:

Algorithm 2 Check for Compatibility

procedure CHECKFORCOMPATIBILITY($x, List$)
  for *each element e* **in** $Sublist(x)$ **do** ▷ *Sublist(x)* indicates the portion of the list starting from the successor of element x
    if $((e.startTime \leq x.startTime + 10)$ **&&** $(e.destinationArea == x.DestinationArea)$ **&&** $(e.flagAsFullRide == false))$ **then**
      MergeRides(x, e)
      break()
    end if
  end for
end procedure

Algorithm 3 Merge Rides

procedure MERGERIDES(x, y)
  Creates a new element z with the same originArea and destinationArea of x and y
  if $x.startTime < y.startTime$ **then**
    $z.startTime \leftarrow x.startTime$
  else
    $z.startTime \leftarrow y.startTime$
  end if
  All the other information about the request/reservation are copied from x and y into z
end procedure

Algorithm 4 Insert an element in List keeping the order

```
procedure InsertKeepengOrder(x, List)
    for each element e in List do
        if (e.startTime ≥ x.startTime) then
            e.prev.next ← x      ▷ e.prev.next indicates the attribute next of the element pointed by e.prev
            x.prev. ← e.prev
            x.next ← e
            e.prev ← x
            break()
        end if
    end for
end procedure
```

Chapter 4

User Interface Design

Here are presented the UX Diagrams for the User Interface of both the Passengers and Taxi Drivers applications. Ux Diagrams are meant to show a detailed schema about the web site navigation done by the users of the system. For the complete mockups refer to the RASD document.

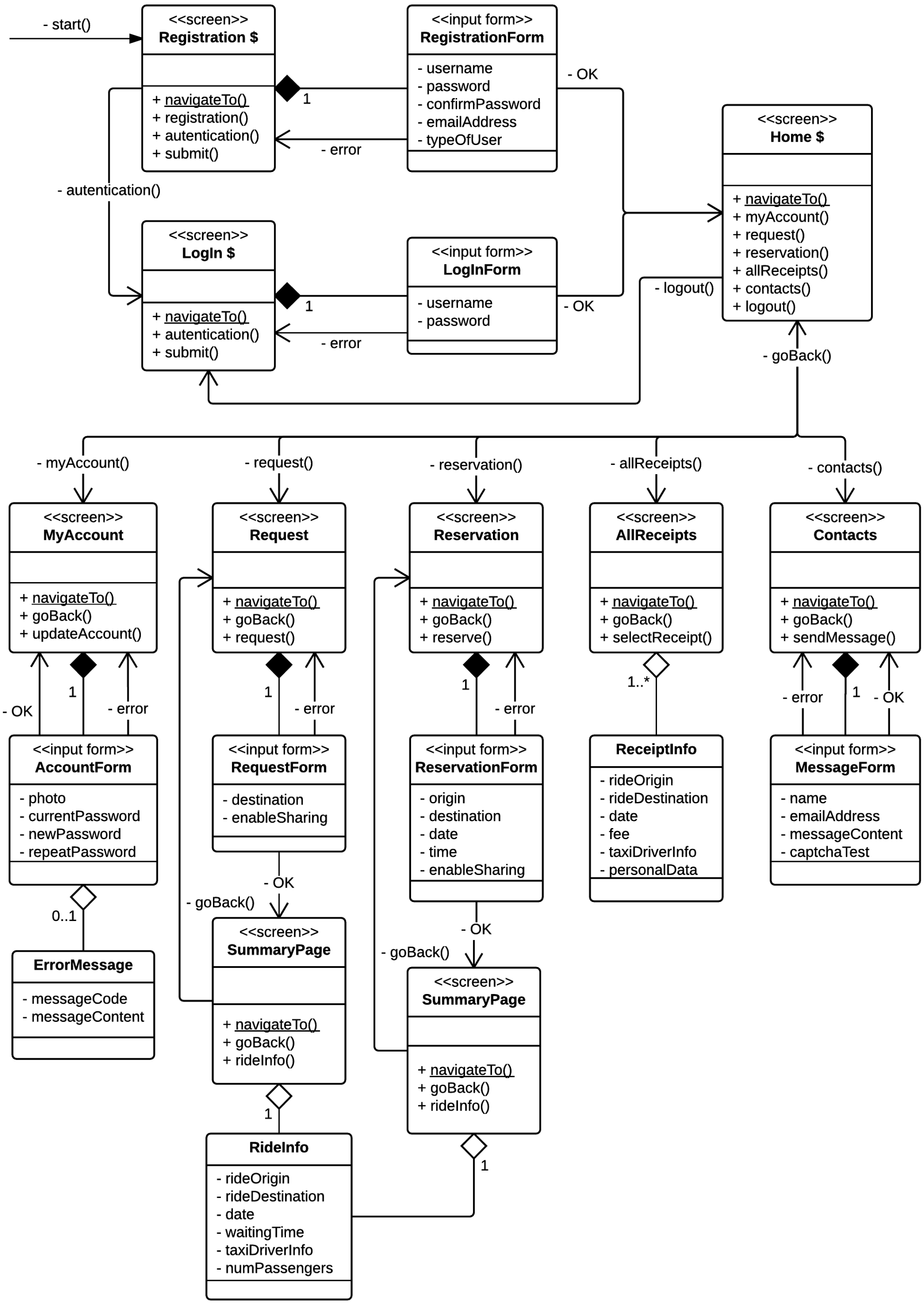

Figure 4.1: UX Diagram - Passenger Application Interface

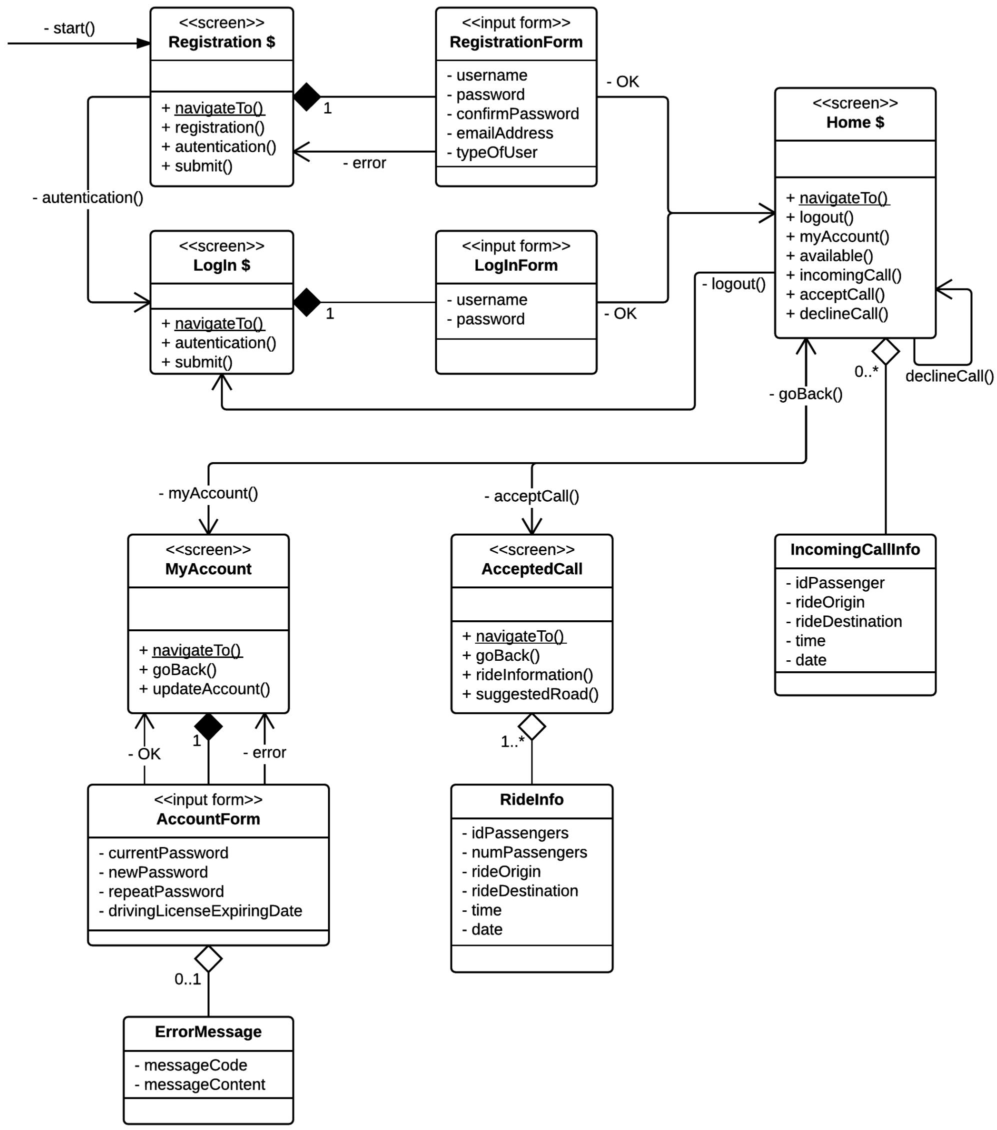

Figure 4.2: UX Diagram - Taxi Driver Application Interface

Chapter 5

Requirements Traceability

R01 check the validity and correctness of the information provided by the visitor (personal information, password)

- This requirement is satisfied by the validity and correctness controls inside the Visitor Manager

R02 check if the user is already registered into the system

R03 check if username and password provided by the visitor correspond to an existing user, authorized to use the system

R04 prevent unauthorized or banned users from accessing the system

- These requirements are satisfied by allowing the Visitor Manager to query the Database

R05 obtain the passenger location

- This requirement is satisfied in a Request by retrieving data from the GPS embedded in the passenger's smartphone and in a Reservation by allowing the Passenger to manually insert through the application interface his/her location (refer to UX Diagram - Passenger Application Interface)

R06 access the queue associated to the right taxi zone

R07 check the availability of the taxi drivers

R08 iteratively contact all the taxi drivers of the queue starting from the first one until one of them accepts the call

R09 iteratively search for an available taxi driver inside adjacent zones in the case that the right zone has an empty queue or all the contacted taxi drivers had declined the request

- These requirements are satisfied within the algorithm "Research of a Taxi Driver in the queue zone or in an adjacent queue zone"

R10 obtain the taxi driver position and estimate the time needed by the taxi driver to reach the passenger

- This requirement is satisfied by retrieving data from the taxicab GPS locator

R11 obtain the taxicab unique identifier from the taxi drivers database

- This requirement is satisfied by allowing the Ride Manager to query the Database

R12 check for each request or reservation if the passenger had selected the sharing function

R13 compare routes that start from the same taxi zone and determine whether or not they can be merged into one, according to specific rules of comparison

R15 elaborate an optimal route for taking every passenger to the right destination and show it to the taxi driver

- These requirements are satisfied within the algorithm "Shared Ride Management"

R14 calculate the correct distribution of the fee according to specific rules based on the percentage of the kilometers shared with others or traveled alone

- This requirement is satisfied within the algorithm "Fee Calculation Function"

R16 keep track of the actual route followed by the taxi driver and keep track of the actual duration of the ride

- This requirement is satisfied by retrieving data from the taxicab GPS locator

R17 update the database information for each user

- This requirement is satisfied by allowing the Passenger Manager and the Taxi Driver Manager to query and update the Database and by allowing the users to insert through the application interface the new data (refer to UX Diagram - Passenger Application Interface and UX Diagram - Taxi Driver Application Interface)

R18 monitor and collect inputs from taxi drivers

- This requirement is satisfied by allowing Taxi Drivers to interact with the system through an appropriate application interface (refer to UX Diagram - Taxi Driver Application Interface)

R19 contact taxi drivers and forward them all the information about the proposed request (position of the passenger, destination of the passenger, sharing option enabled or not)

R20 retrieve the taxi location and the locations of all the passengers of the ride

- This requirement is satisfied within the algorithm "Research of a Taxi Driver in the queue zone or in an adjacent queue zone"

R21 Access and query the map provider service to obtain an updated map with information about traffic, smashes, road construction sites

- This requirement is satisfied by exploiting the Google Maps API

R22 Update the system code and architecture

- This requirement is satisfied by some functionalities inside the Developer Manager

Chapter 6

References

6.1 External References

Link referenced to documentation about JEE architecture:
http://docs.oracle.com/javaee/6/tutorial/doc/bnaay.html

Link referenced to documentation about the Java Interface Queue:
http://docs.oracle.com/javase/7/docs/api/java/util/Queue.html

Link referenced to documentation about modeling:
http://www.agilemodeling.com

6.2 RASD Modifications

New Domain Properties:

- [D01] Taxicabs are all equals. They have a maximum of 3 passenger seats and they are all owned by a specific company.
- [D02] Every taxi driver uses always the same taxicab
- [D09] The total fee of a ride is calculated considering only the total kilometers of the ride, given a specific fee per km

Changes in Functional Requirements:

- [R10] obtain the taxi driver position and estimate the time needed by the taxi driver to reach the passenger

Add new functionality "Update Account":

- add new use case "Update Account" and its description
- add new functional requirement [R17]
- add new goals [G08] and [G13]
- mockup modification for the Passenger Home Page and Taxi Driver Home Page add new states in the State Chart Diagram for Passengers and State Chart Diagram for Taxi Drivers