**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.

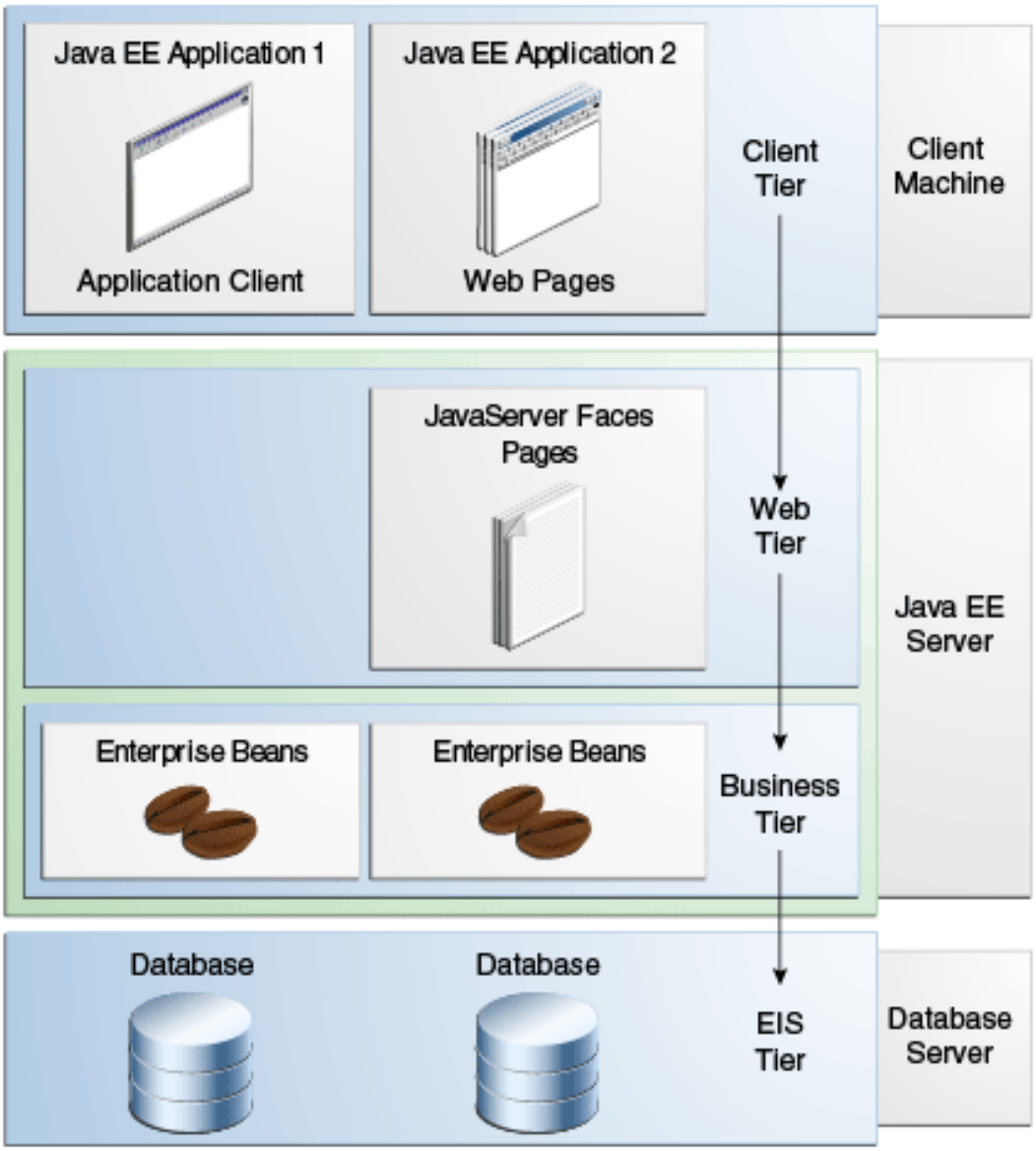


Figure 2.1: JEE 4-tier architecture

## 2.2 High Level Components and their Interaction

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

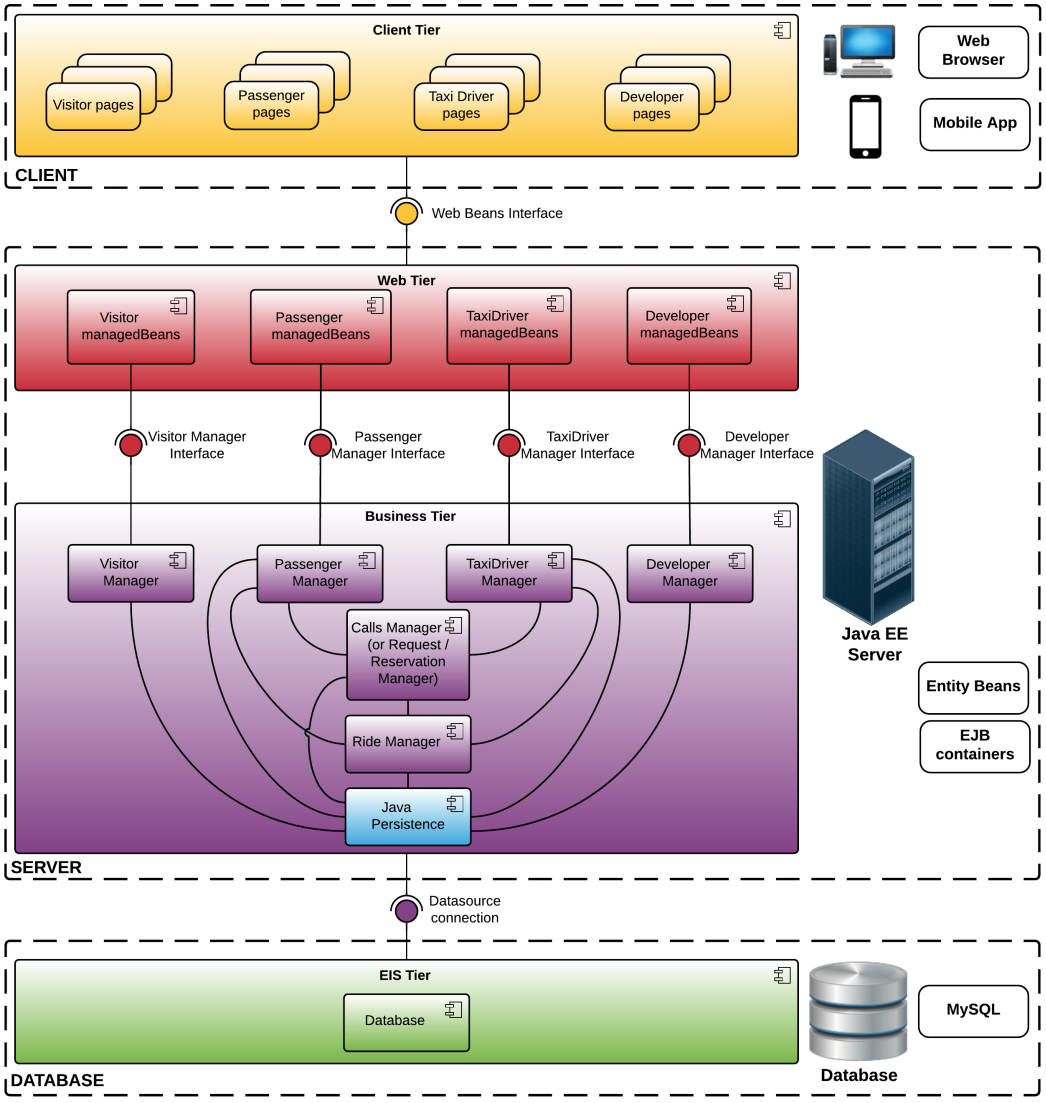


Figure 2.2: High Level Components view and their interaction

## 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.

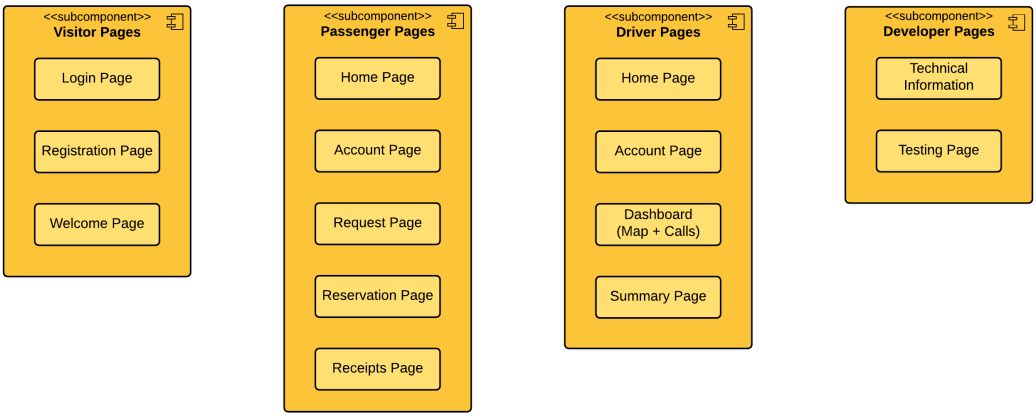


Figure 2.3: Client subcomponents

### 2.3.2 Web Component

The Web component generates dynamic web pages containing XHTML. Web components implements Java Server Faces technology, which is a common user interface component framework for web applications. In this way every user input in all the client pages is managed by these beans, one per each group of pages.

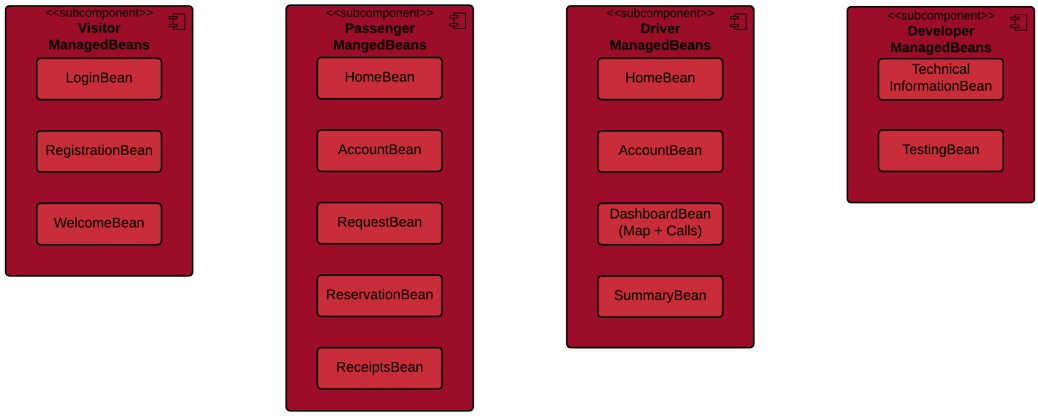


Figure 2.4: Web subcomponents

### 2.3.3 Business Logic Component

The Business Logic component 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 Entity, which holds the information of the system data model, and is in charge of storing and retrieving information from a database. In this first release of the Design Document we have focused only on a small number of fundamental elements (Java Beans) necessary to manage the basic functionalities offered to the users by the system. Further additions will be necessary during the development. 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 conceptual architecture of the database is depicted in this diagram using the notation of Entity - Relation Diagram which is useful to individuate all the entities of the system and their mutual relationship.

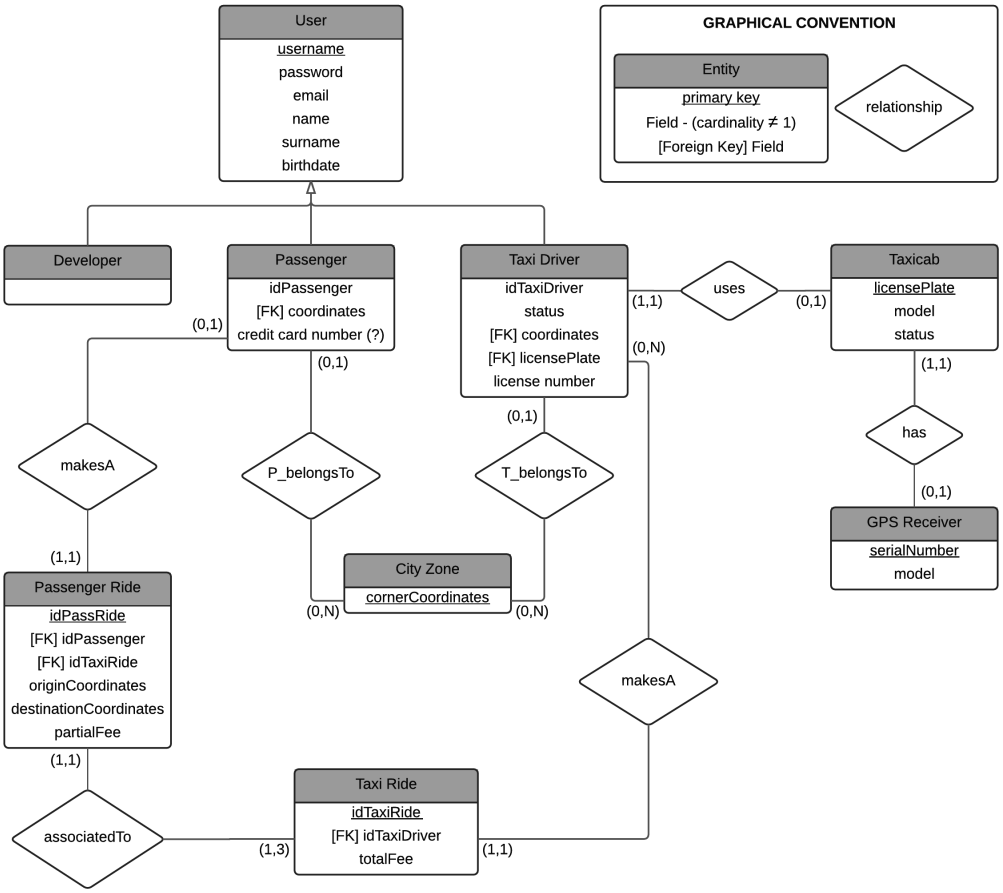


Figure 2.6: Database ER Diagram

## 2.4 Deployment View

The diagram in Figure 2.7 shows the deployment view of the software product. Because of the early stage of the developing of the system, this diagram is deliberately simple and only depicts the distinction between client machines, server machines and database machines at large. Further revisions will go deeper into the hardware architecture of the system and will identify more specific hardware components in which the software will be deployed.

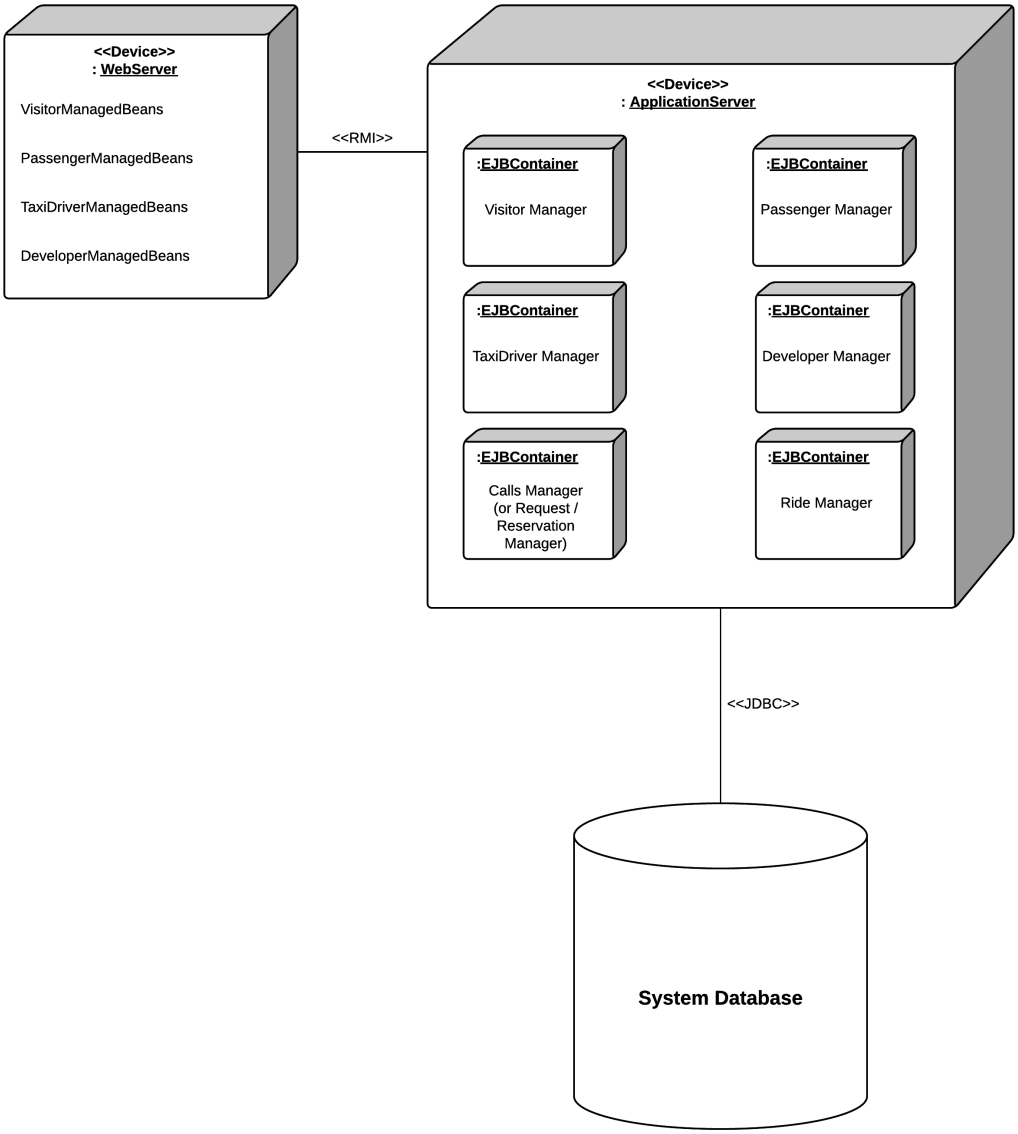


Figure 2.7: Deployment view

## 2.5 Runtime View

The following diagrams depicts the runtime view of MyTaxiService project describing in a simple way how the various components defined until this point behave in order to accomplish some of the most important activities of the system.

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

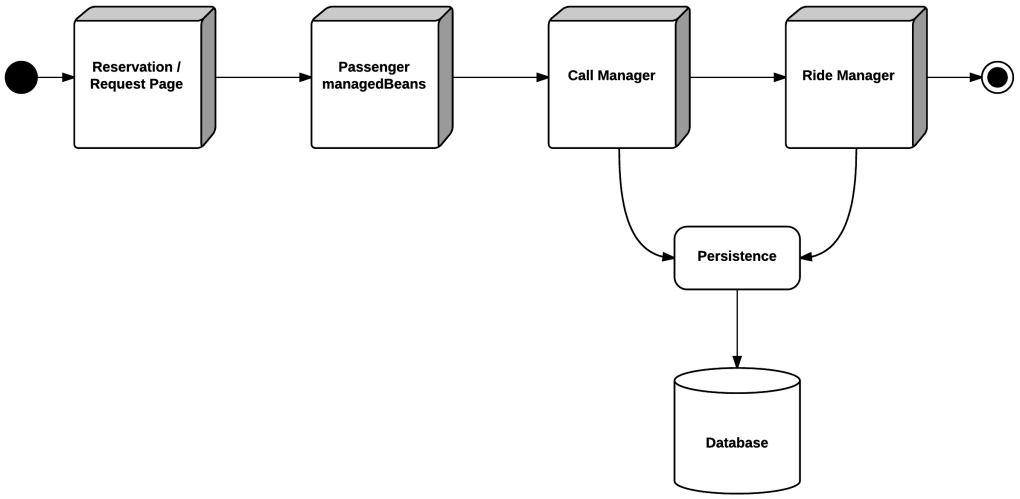


Figure 2.8: Runtime Taxi Request and Reservation

- This diagram represents the activity of showing the receipts to the user

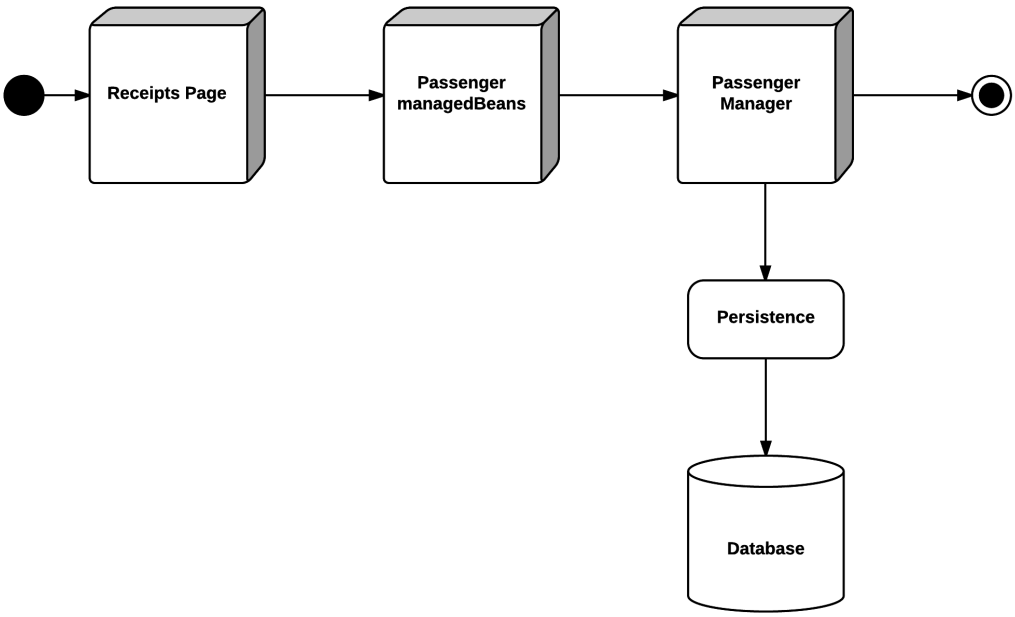


Figure 2.9: Runtime get receipts

- This diagram represents the components that are involved in the modification of the passenger's account, and their interaction

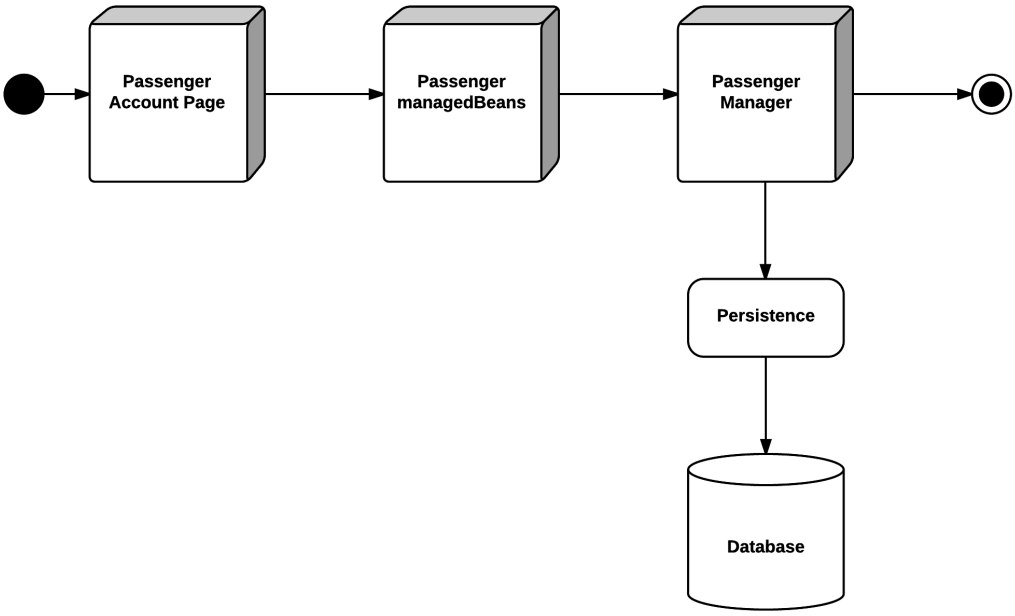


Figure 2.10: Runtime Modify Passenger's Account

- This diagram represents the activity done by the system to let a taxi driver modify his account

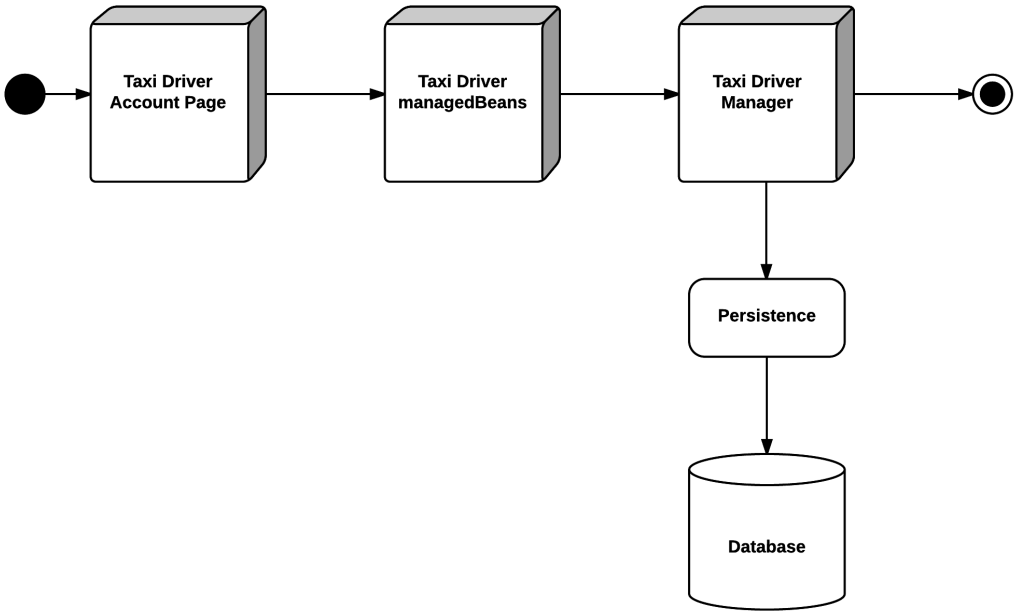


Figure 2.11: Runtime Modify Taxi Driver's Account

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

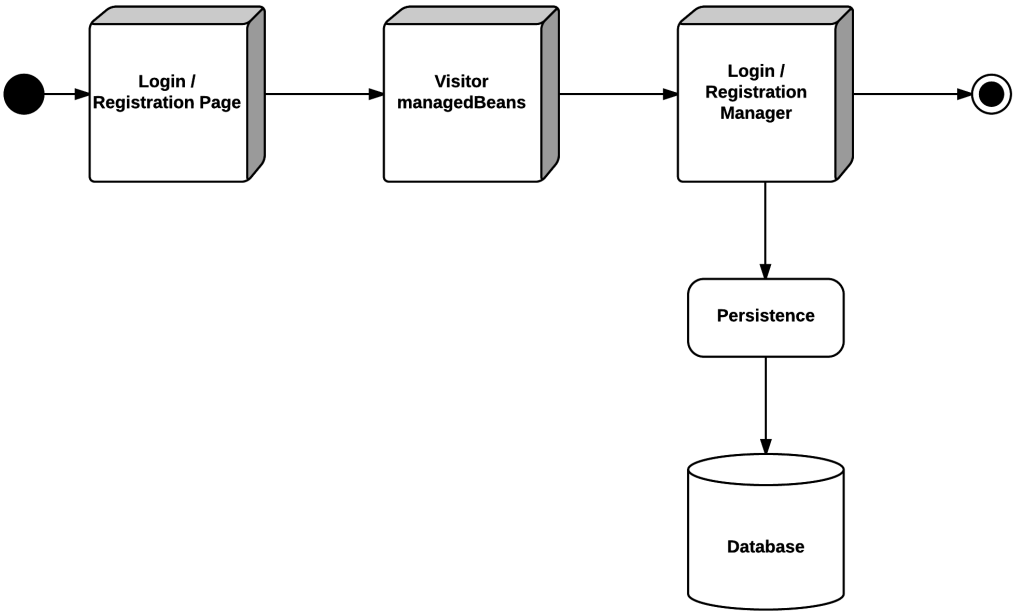


Figure 2.12: Runtime Login and Registration

- This diagram represents the components that are needed during the process of starting a taxi ride, and their interaction

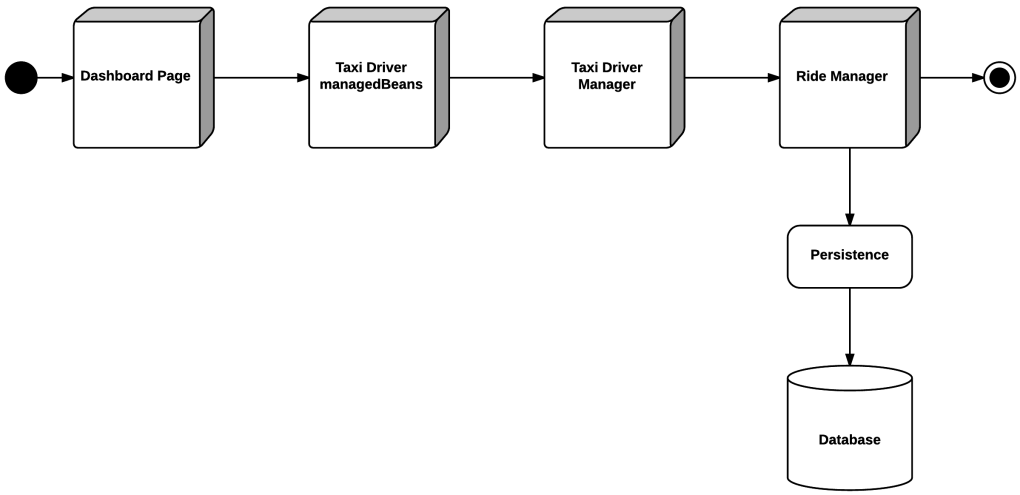


Figure 2.13: Runtime start Taxi Ride

- This diagram represents the components that are involved in the process of showing the summary of the ride to the taxi driver and their interaction

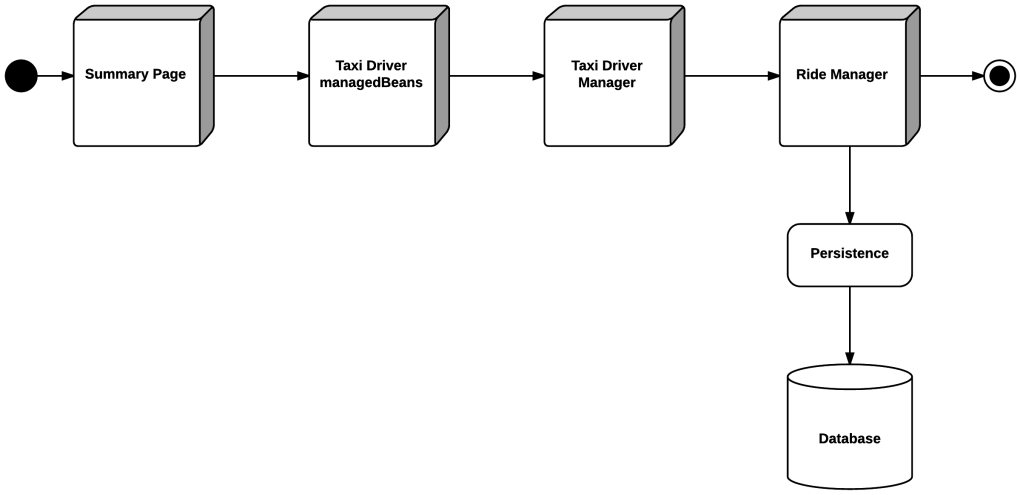


Figure 2.14: Runtime Summary

## 2.6 Component Interfaces

Here are identified some functions offered by the Beans of the Business Tier:

### Visitor Manager

The Visitor Manager should expose some methods like:

- *createNewUser* to add a new user to the database
- *verifyLogin* and *verifyRegistration* to check if the information provided by the user in the Login or Registration Pages are correct

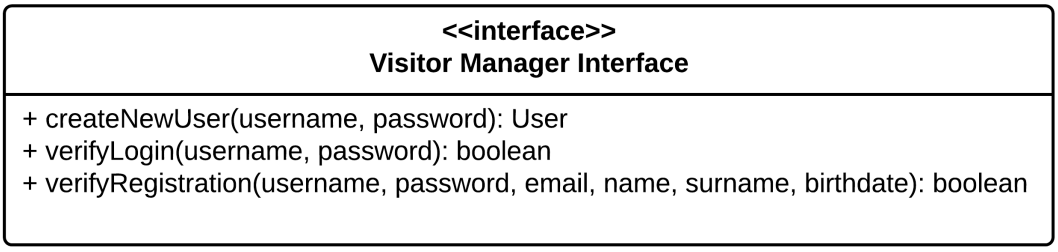


Figure 2.15: Visitor Manager Interface

**Passenger Manager**

The Passenger Manager should expose some methods like:

- *getPassengerInformation* to retrieve all the information about a certain passenger stored in the database
- *getPassengerLocation* to retrieve the passenger position from it's smartphone GPS
- *getPassengerState* to obtain the current state of the passenger
- *setPassengerState* to modify the current state of the passenger
- *updatePassengerInformation* to update the information about a passenger stored into the database

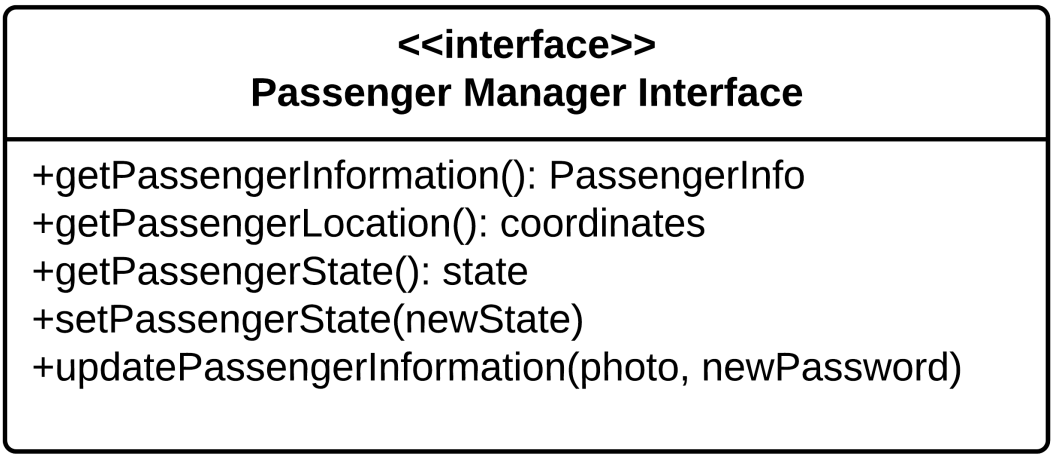


Figure 2.16: Passenger Manager Interface

### Taxi Driver Manager

The Taxi Driver Manager should expose some methods like:

- *getTaxiDriverInformation* to retrieve all the needed information about a taxi driver stored into the database
- *getTaxiDriverLocation* to retrieve the taxi driver position from it's GPS device
- *updateTaxiDriverInformation* to update the information about a taxi driver stored into the database
- *checkTaxiDriverAvailability* to check the current status of the taxi driver

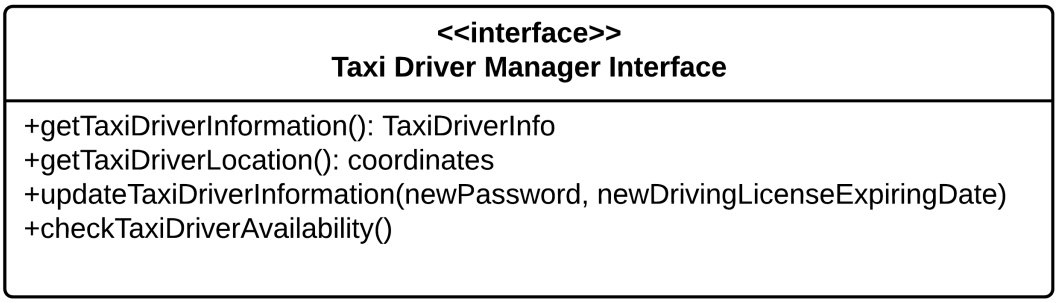


Figure 2.17: Taxi Driver Manager Interface

**Developer Manager**

The Developer Manager should expose some methods like:

- *updateDeveloperInformation* to update the information about a developer stored into the database
- *codeInspector* to see the whole system's code
- *addFeature* to write code
- *updateCode* to update the whole system's code

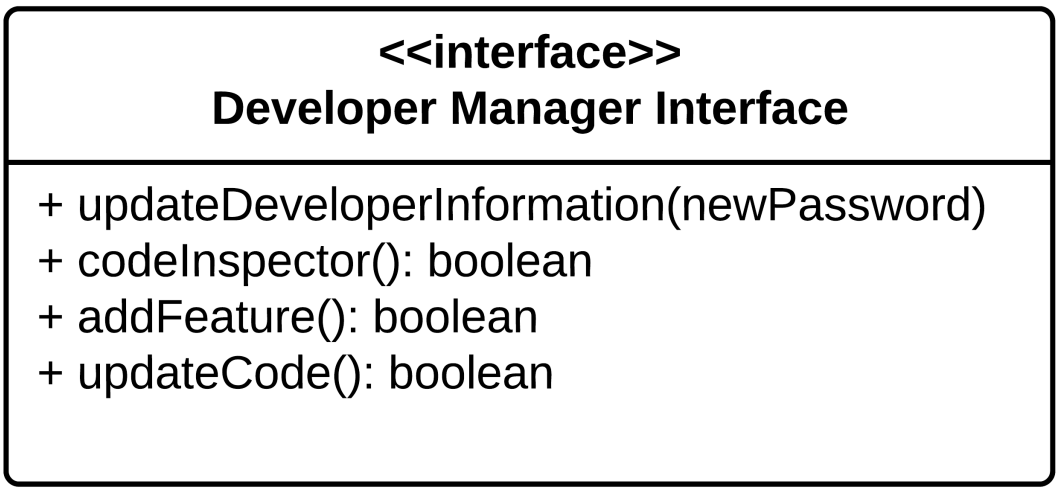


Figure 2.18: Developer Manager Interface

**Calls Manager**

The Calls Manager should expose some methods like:

- *createNewCall* to elaborate an incoming request or reservation from the ManagedBeans of the web tier
- *updateCall* to modify the status of a pending call
- *searchForRide* to start the research of a feasible ride that fulfills the client's request / reservation (origin, destination, sharing) and thus the research of an available taxi driver

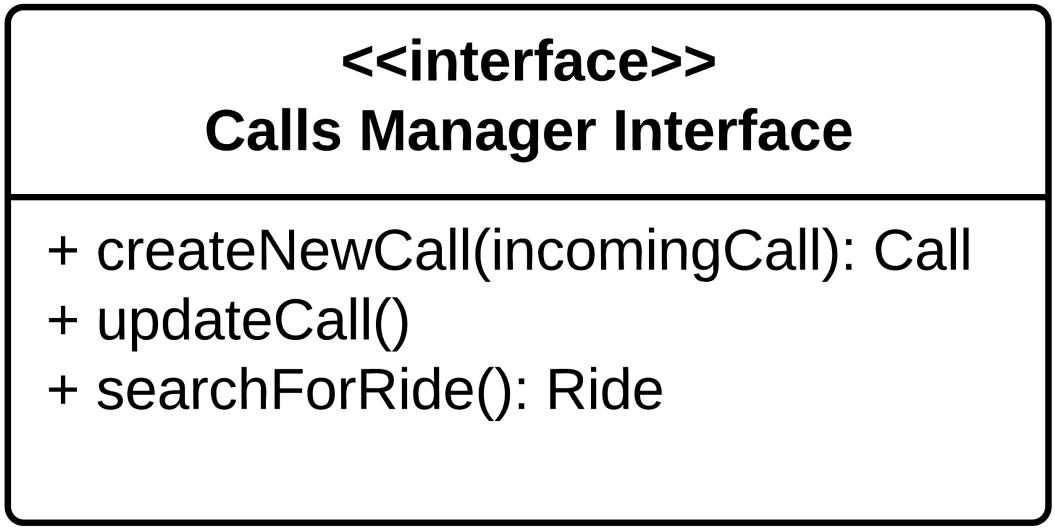


Figure 2.19: Calls Manager Interface

**Ride Manager**

The Ride Manager should expose some methods like:

- *createNewRoute* to receive the information about the selected ride and elaborate the optimal route for the taxi driver
- *collectRideData* to collect and store all the useful data about the ride (i.e. durations, number of passengers, total fee, fee per passenger, taxi driver, length)
- *exportRideData* to share collected data with other component or systems

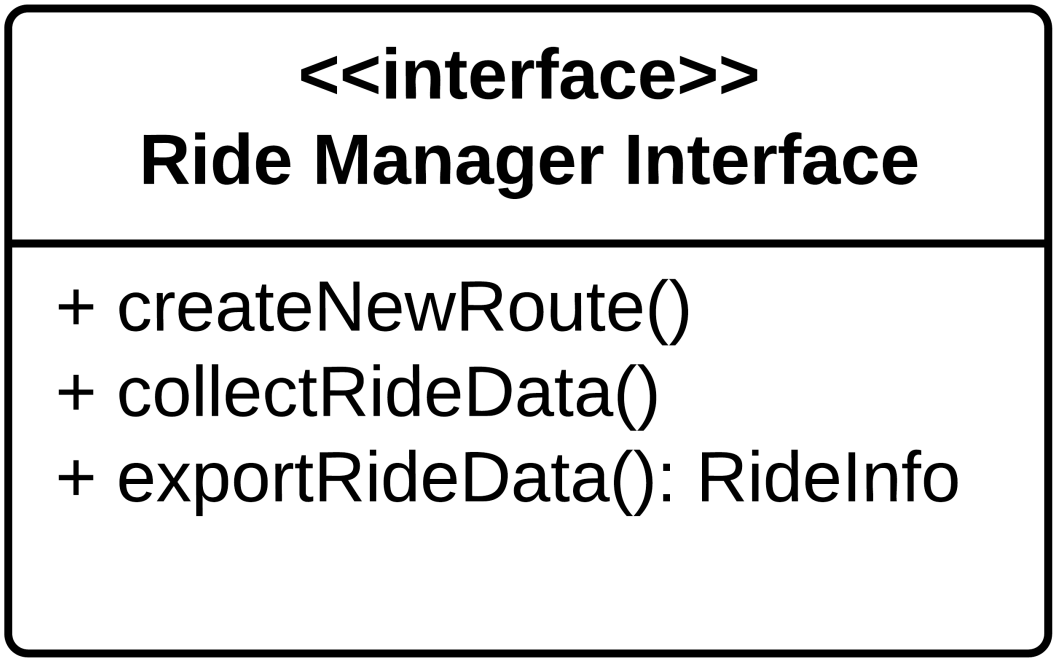


Figure 2.20: Ride Manager Interface

## 2.7 Selected Architectural Styles and Patterns

During the developing of this document and the construction of the system we have focused on keeping the architecture and the general behavior of the system as simple as possible, trying to adapt everything to the architecture of Java EE which is a very good starting point for a system like ours. In our mind this initial simplicity and modularity leaves the right amount of space for further improvements and strengthening of the system. Moreover, given the nature of the system (a taxi sharing application) we think that this 4-tier architecture that relies upon the Model - View - Controller pattern is the one that most fits the requirements and grants optimal performances in terms of reliability, availability and performances of the system.

# Chapter 3

# Algorithm Design

**Research of a Taxi Driver in the queue:**

This algorithm describes how the system manages the research of a taxi drivers. It will use the *Dequeue* operation in order to extract the first taxi driver and the *Call* function in order to propose him a ride. If the first taxi driver accepts the request, this will be assigned to him, otherwise he will be enqueued again using the *Enqueue* operation 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

```
1: procedure SearchTaxiDriver(Q)
2:     if (Q.head == Q.tail) then                          ▷ Queue is empty
3:     else
4:         for i := 0 to Q.lenght do    ▷ The system knows the length of the
   Queue, so it makes exactly Q.length controls
5:             x ← Dequeue(Q)          ▷ Classical operation for managing the
   extraction of an element from the Queue
6:             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
7:             if (acceptance == true) then
8:                 return x
9:             else
10:                Enqueue(Q, x)          ▷ Classical operation for managing the
   insertion of an element from the Queue
11:            end if
12:        end for
13:    end if
14: 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".

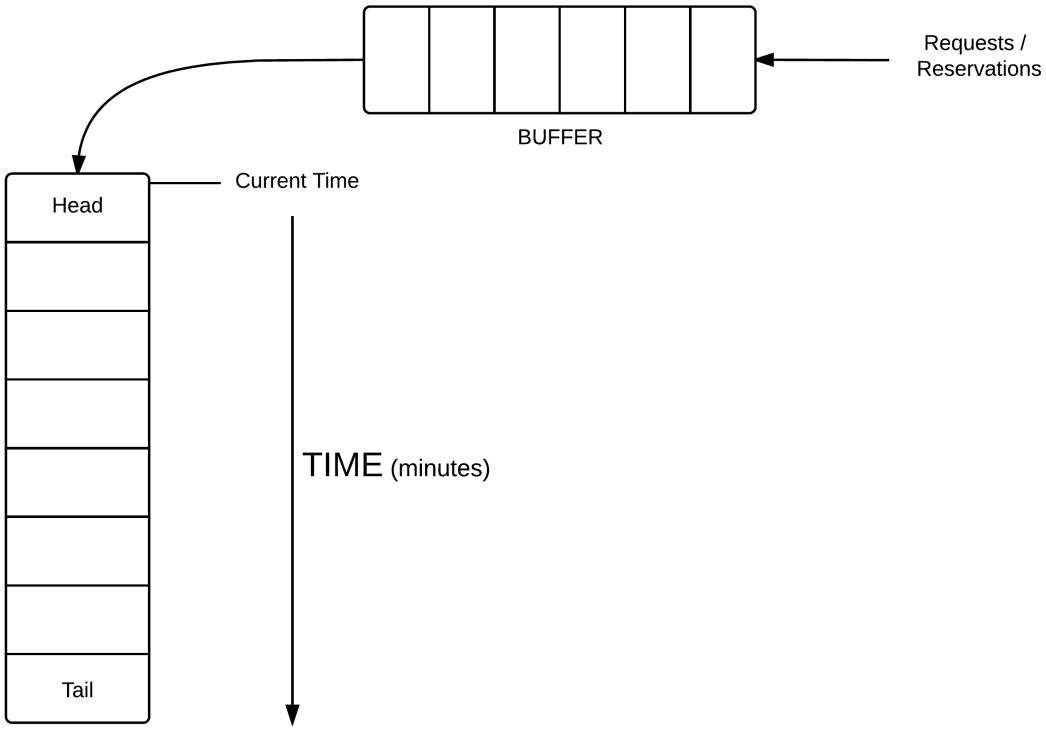


Figure 3.1: Representation of interaction between Queue and Buffer

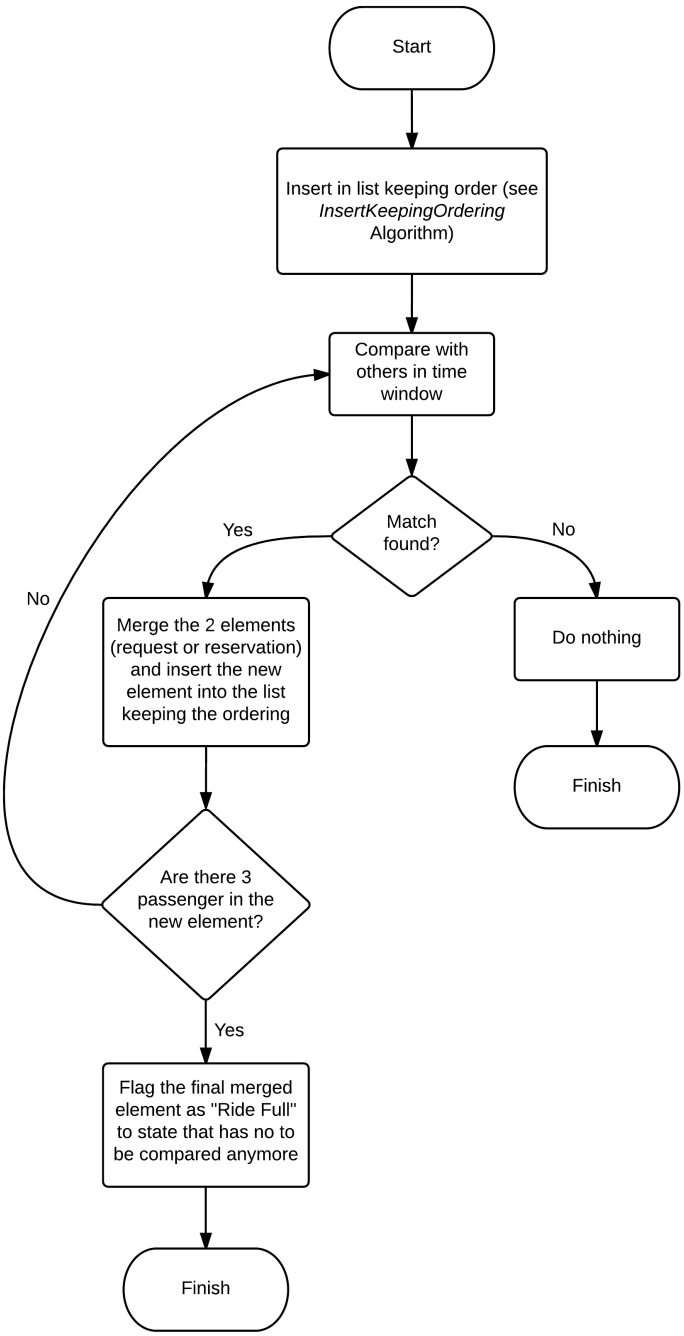


Figure 3.2: Representation of the Shared Ride Management Algorithm

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

**Algorithm 2** Check for Compatibility

```
1: procedure CheckForCompatibility(x, List)
2:     for each element e in Sublist(x) do ▷ Sublist(x) indicates the portion of the list starting from the successor of element x
3:         if ((e.startTime ≤ x.startTime + 10) && (e.destinationArea == x.DestinationArea) && (e.flagAsFullRide == false)) then
4:             MergeRides(x, e)
5:             break()
6:         end if
7:     end for
8: end procedure
```

**Algorithm 3** Merge Rides

```
1: procedure MergeRides(x, y)
2:     Creates a new element z with the same originArea and destinationArea of x and y
3:     if x.startTime < y.startTime then
4:         z.startTime ← x.startTime
5:     else
6:         z.startTime ← y.startTime
7:     end if
8:     All the other information about the request/reservation are copied from x and y into z
9: end procedure
```

**Algorithm 4** Insert an element in List keeping the order

```
1: procedure InsertKeepingOrdering(x, List)
2:     for each element e in List do
3:         if (e.startTime ≥ x.startTime) then
4:             e.prev.next ← x     ▷ e.prev.next indicates the attribute next of the element pointed by e.prev
5:             x.prev. ← e.prev
6:             x.next ← e
7:             e.prev ← x
8:             break()
9:         end if
10:    end for
11: 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.

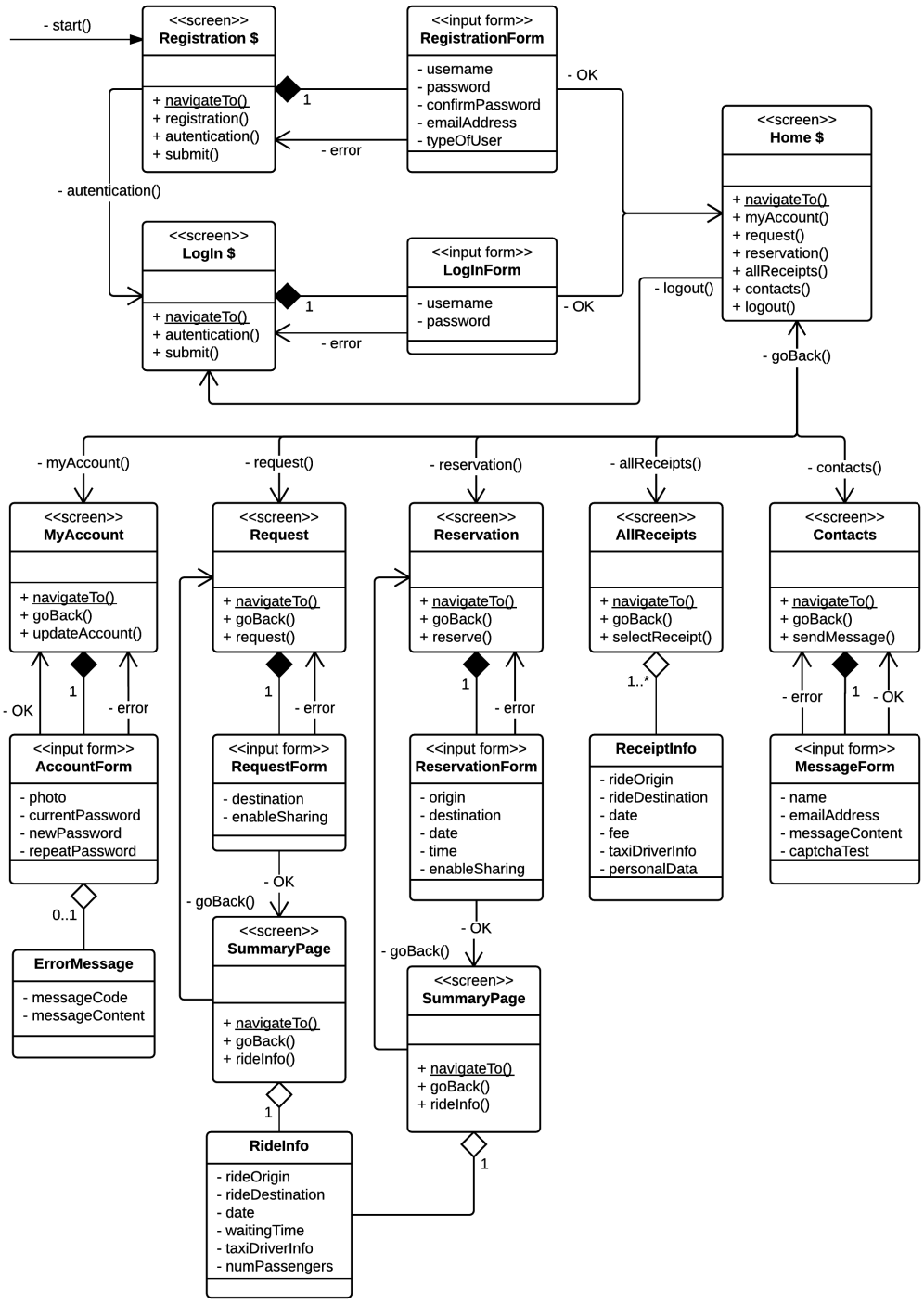


Figure 4.1: UX Diagram - Passenger Application Interface

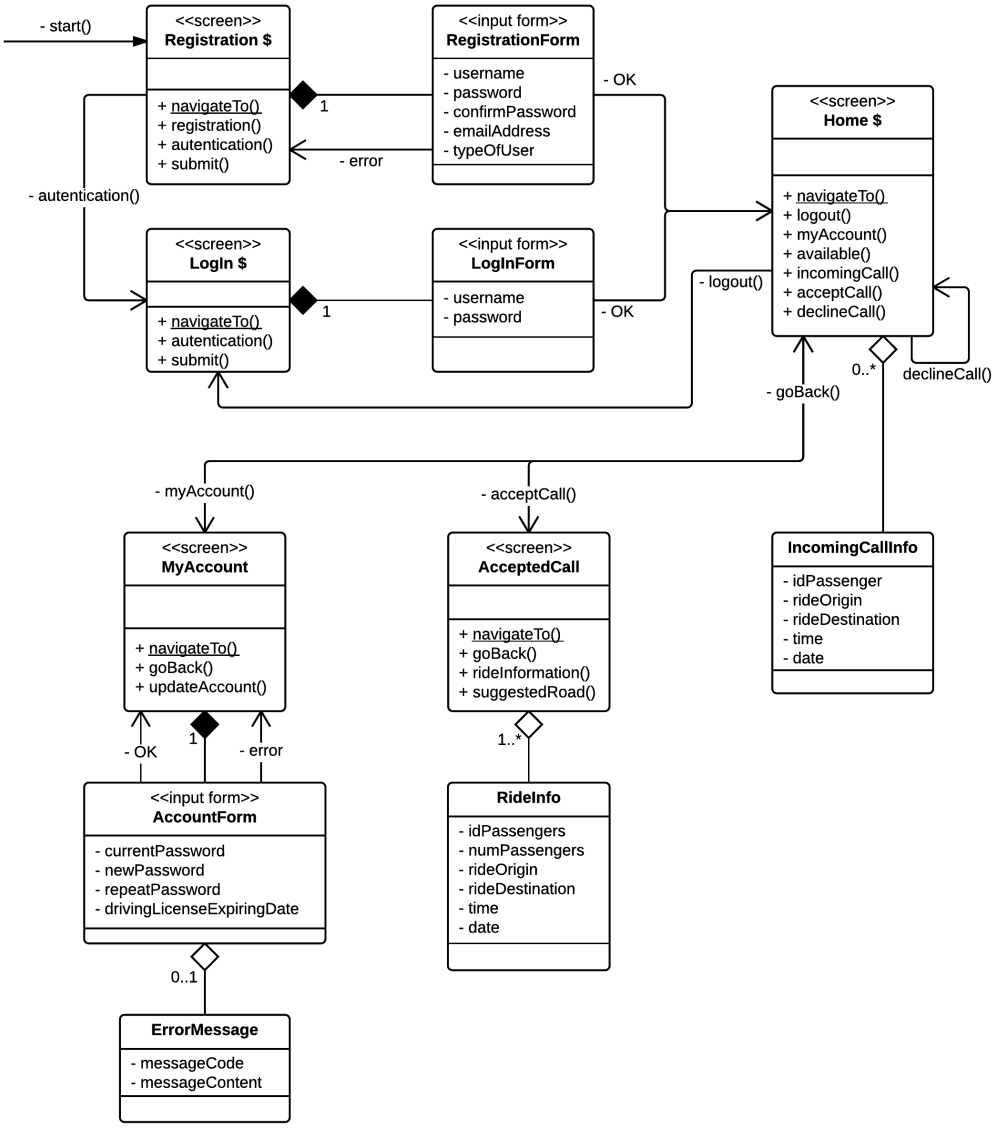


Figure 4.2: UX Diagram - Taxi Driver Application Interface

# 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

Book referenced to documentation about algorithms:
Book "Introductions to algorithms" by Cormen, Leiserson, Rivest, Stein - MIT Press (3rd edition)

## 6.2 DD Modifications

**Business Logic subcomponents:**

- Type error: *LoginRegistration Manager* changed in *Visitor Manager*

## 6.3 Working Hours

| First Name | Last Name | Total Hours |
|---|---|---|
| Mattia | Crippa | 23h |
| Francesca | Galluzzi | 25h |
| Marco | Lattarulo | 26h |