Software engineering 2 project
PowerEnJoy

Design document
(DD)

Authors:
Madaffari Federico (matriculation number: 792873) (Computer Science)
Mandrioli Claudio (matriculation number: 849973) (Automation and Control)

Sommario

Introduction

Revision history

Version 2.0

As for the update made for version 3.0 of the RASD in the version 2.0 of this document changes are applied to manage the drop of the domain assumption [D5] (see RASD). According to this the meaning of the unlock action in the sequence diagram related to the start a ride process, as pointed out in the footnote. Also other minor changes are made.

Purpose

The purpose of this document is to provide a high level and functional description of how the software management system of the PowerEnJoy service will be implemented.
The architecture of the software will be defined starting for the definition of the macro components through the low-level ones.
This document is addressed to those who are going to actually implement the system.

Scope

We recall that the service is a car sharing of electric vehicles in a city-perspective (Milan) and that this software product has both the objective to provide the service to the clients and to manage the vehicles involved in the service for what regards they're state of charge and location.
More specific details on the intended service can be found on the RASD document.

Definitions

In this section, we provide some fundamental definitions for the readability of this document:

- API: application programming interface, it is a standard way to interact with another system.
- BEAN: is a server-side component that encapsulates a business logic of an application.
- JEE: Java Enterprise Edition, the computing platform for enterprise software used in this project.
- APPLICATION SERVER: software framework that provides facilities and environment to run mobile and web applications.
- MOBILE APPLICATION: the application supposed to run of devices like smartphones.
- WEB APPLICATION: the application supposed to run on computer devices.
- RASD: Requirements analysis and specification document, when named in this document we refer to the previously produced document for the PowerEnJoy service.
- DD: design document, it is this document.

Moreover, we recall the definitions given for the RASD document that also hold in this context:

- SYSTEM: when this word is used in the document it refers to the digital management system to project
- USER: is the costumer of the PowerEnJoy service, not necessarily already registered to the system
- REGISTERED USER: is the user that registered to the system providing his personal info, therefore in the system there must be stored those data
- EMPLOYEES: are those people that work for the PowerEnJoy company that will take care of cars out of charge
- CAR: in this document my car are intended only those electric cars related to the PowerEnJoy service (model of car: BMW i3)
- START THE ENGINE: when the driver activates the electric motor of a car
- RIDE: is a time span in which a certain user is using a certain car. This time span starts when the user starts the engine (ride start) and ends when all doors of the car are closed and no passengers are on the car (ride end)
- PICK UP THE CAR: this expression is used as a synonym of 'start the ride'
- LOCK THE CAR: when the system (as only the system can perform this action) locks a car at the end of a ride
- UNLOCK THECAR: when the system (as only the system can perform this action) unlocks the car after the user's request
- RESERVE: when a user tells the system that he is going to use a car and therefore car must be marked as unavailable
- RESERVED CAR: a car marked unavailable because some user reserved it
- CHARGE OF THE PAYMENT: is the total amount to be paid by the user after a ride
- PAYMENT SYSTEM: is the external service that provides the payment functionality to the PowerEnJoy service
- APPLY A DISCOUNT: when at the end of a ride the charge of a payment is reduced according to discount policy of the service provider
- CHARGE --% EXTRA: when at the end of a ride the charge of a payment is increased of some percentage according to fine policy of the service provider
- POSITION: coordinates that localize something (a car, a user, some area...)
- USER EXITS THE CAR: is used as a synonym of end of 'end of the ride'
- END OF THE RIDE: when during a ride all the doors are closed and no more passengers are in the car
- AVAILABLE CAR: a car is said to be available when it is not unavailable (see definition of 'unavailable') and moreover it is possible for the users to reserve it
- UNAVAILABLE CAR: a car is said to be unavailable when it is reserved, being used by some user or with the battery level under 20%
- SAFE AREA: predefined set of places where it is possible for the users to park the cars

- CHARGING AREAS: are the parking provided with plugs to recharge the cars of the PowerEnJoy service (these are a subset of safe area)
- POWER GRID STATION: is a synonym of charge area
- STATE OFCHARGE: the state of charge of a car is the level in percentage of the energy stored in the battery
- PRE-DEFINED: when some information is supposed to have been previously provided to the database of the system and therefore it is always available from the activation of the service
- WELL-DEFINED: when some information if defined precisely and without ambiguity

Reference documents

For the writing of this document the following where consulted:

- Assignments AA 2016-2017
- RASD document previously produced

Document structure

- Introduction: provides an overall description of the aim and the contents of this document.
- Architecture design: this section is the core of the document. It describes the sequence of architectural choices done for the project of this software system both from the static perspective and the dynamic one.
- Algorithm design: provides a glance on the relevant and critical parts of the software product.
- User interface design: provides a guideline for how the system is supposed to appear to the users and to interact with them.
- Requirements traceability: provides a set of relations among the requirements listed in the RASD document and how and where they are supposed to be implemented in the software.

Architectural Design

This section provides a general description of the software system including its functionality and concerns related to the overall system and its design.

Overview

System technologies

The PowerEnJoy system will be designed considering the client-server 3-tier architectural style. Each tier requires specific technologies as depicted below:

Web tier

- Dynamic web pages containing XHTML, which are generated by web components.
- Web components developed with Java Server Faces technology, which is a user interface component framework for web applications.

Business Logic tier

- Java Enterprise Edition 7(JEE7) platform supports applications that provide enterprise services in the Java language.
- Enterprise Java Beans (EJB) 3.2, business components that capture the logic that solves or meets the needs of a particular business domain.
- GlassFish AS 4.1, an open-source application server that provides services such as security, data services, transaction support, load balancing, management of distributed applications and supports the JEE7 platform.

Persistence tier

- MySQL Server 5.7, a RDBMS.

Design approach

The design approach is based on a client-server 3-tier distributed system, where each tier is described as follows:

1. **Client tier:** The client tier consists of application clients that access a Java EE server and that are usually located on a different machine from the server. The clients make requests to the server. The server processes the requests and returns a response back to the client. This tier is responsible of translating user actions and displays the output of logic operations into something the user can understand.

2. **Business Logic tier:** The business tier consists of components that provide the business logic for an application. Business logic is code that provides functionality to a particular business domain, like the financial industry, or

an e-commerce site. In a properly designed enterprise application, the core functionality exists in the business tier components. This tier coordinates the application, processes, operation, logical decisions and it performs calculations. It processes data between the client and persistence tiers. The following Java EE technologies are used in the business tier in Java EE applications: Enterprise JavaBeans components, JAX-RS RESTful web services, JAX-WS web service endpoints, Java Persistence API entities.

3. **Persistence tier:** This tier holds the information of the system and it is in charge of storing and retrieving information from a database.

The design process followed a top-down approach, so first the team will identify the tiers and then they split them in components to analyze the inner functionalities. Hence, each component is responsible for certain functionalities and interacts with others.

Overall design

This subsection presents the design model of PowerEnJoy system, specifying the basic relations between packages, use cases and users.

General Package Design

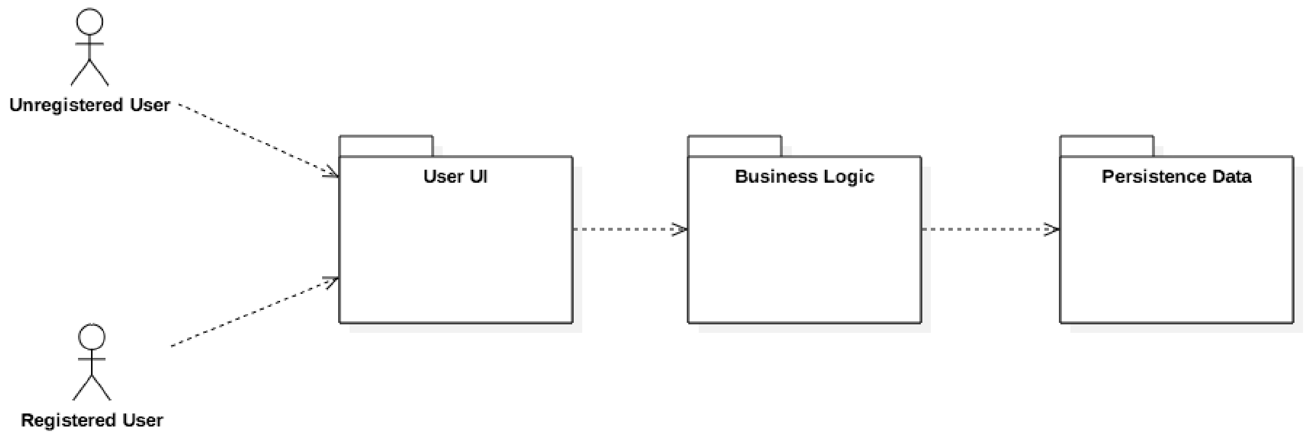
Each tier is split into components and each component must guarantee certain functionalities, which fulfil the requirements. There is a correlation between use cases and package design. In the diagram, we can identify three packages:

- **User UI Package**
  This package contains the user interfaces. It is responsible for the interaction with the user. For instance, it manages the UI request and it refers them to the Business Logic package and retrieving the data back for displaying.

- **Business Logic Package**
  This package contains the business logic components. This package is responsible for handling the User UI package request, processing the information and accessing to Persistence Package to get the data.

- **Persistence data Package**
  This package is responsible for managing the data requests from the Business Logic Package.

Registered and unregistered users can access directly to the User UI package and submit request to accomplish their tasks.



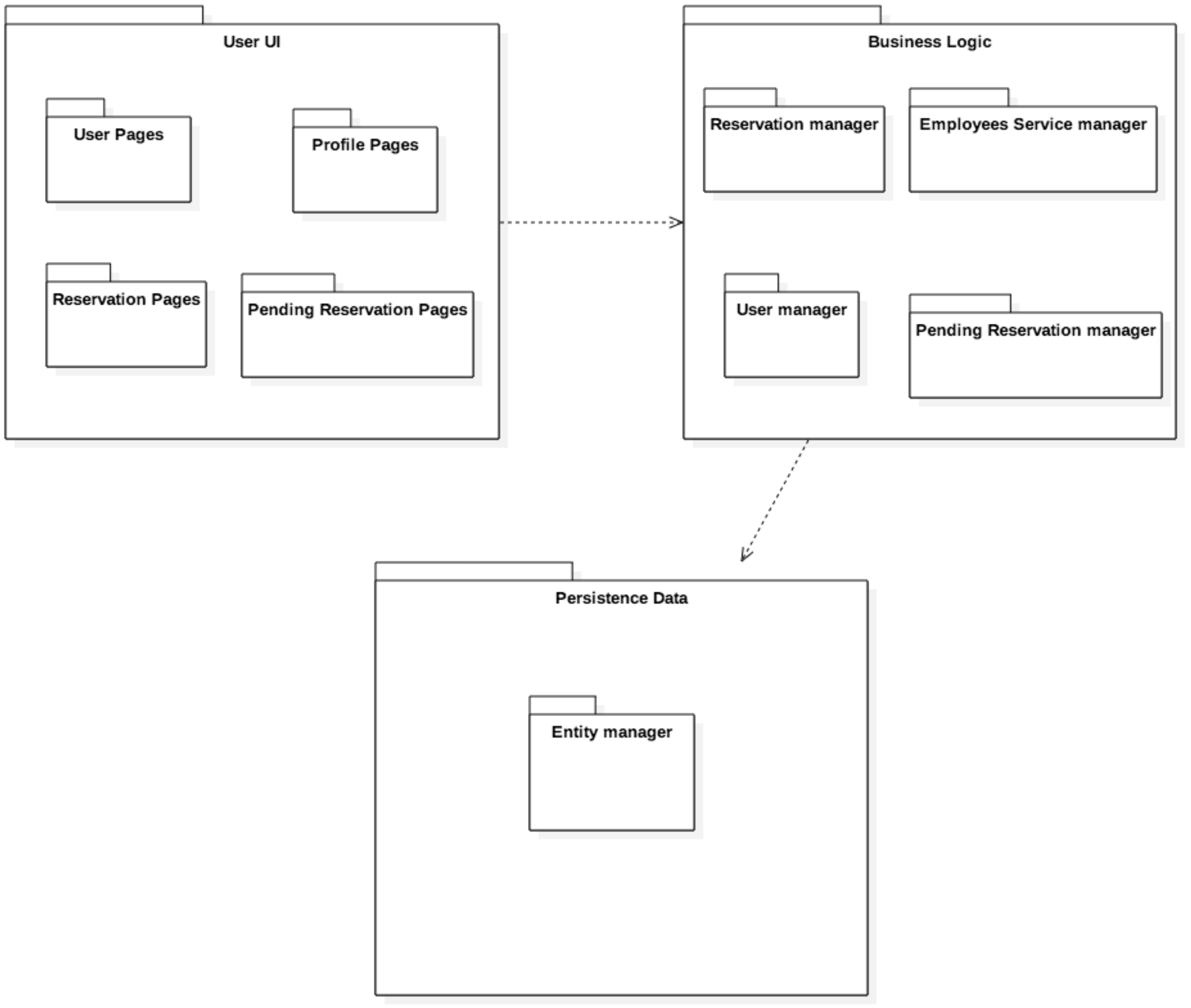
Detailed Package Design

This section will point out the package design in a more precise and detailed way:

- **User UI package**
  - User pages. This package contains the general interfaces for user pages.
  - Profile Pages. This package contains the user interfaces for personal profile.
  - Pending Reservation Pages. This package contains the user interfaces for the pending reservation section.
  - Reservation Pages. This package contains the user interfaces for the reservation section.

- **Business Logic package**
  - Pending Reservation manager. This package contains the logic to manage the pending reservations.
  - Reservation manager. This package contains the logic to manage car reservations.
  - User manager. This package contains the logic to manage system's users.
  - Employee Service manager. This package contains the logic to manage all the employees' tasks.

- **Persistence data package**
  - Entity Manager. This package is responsible for managing data requests.

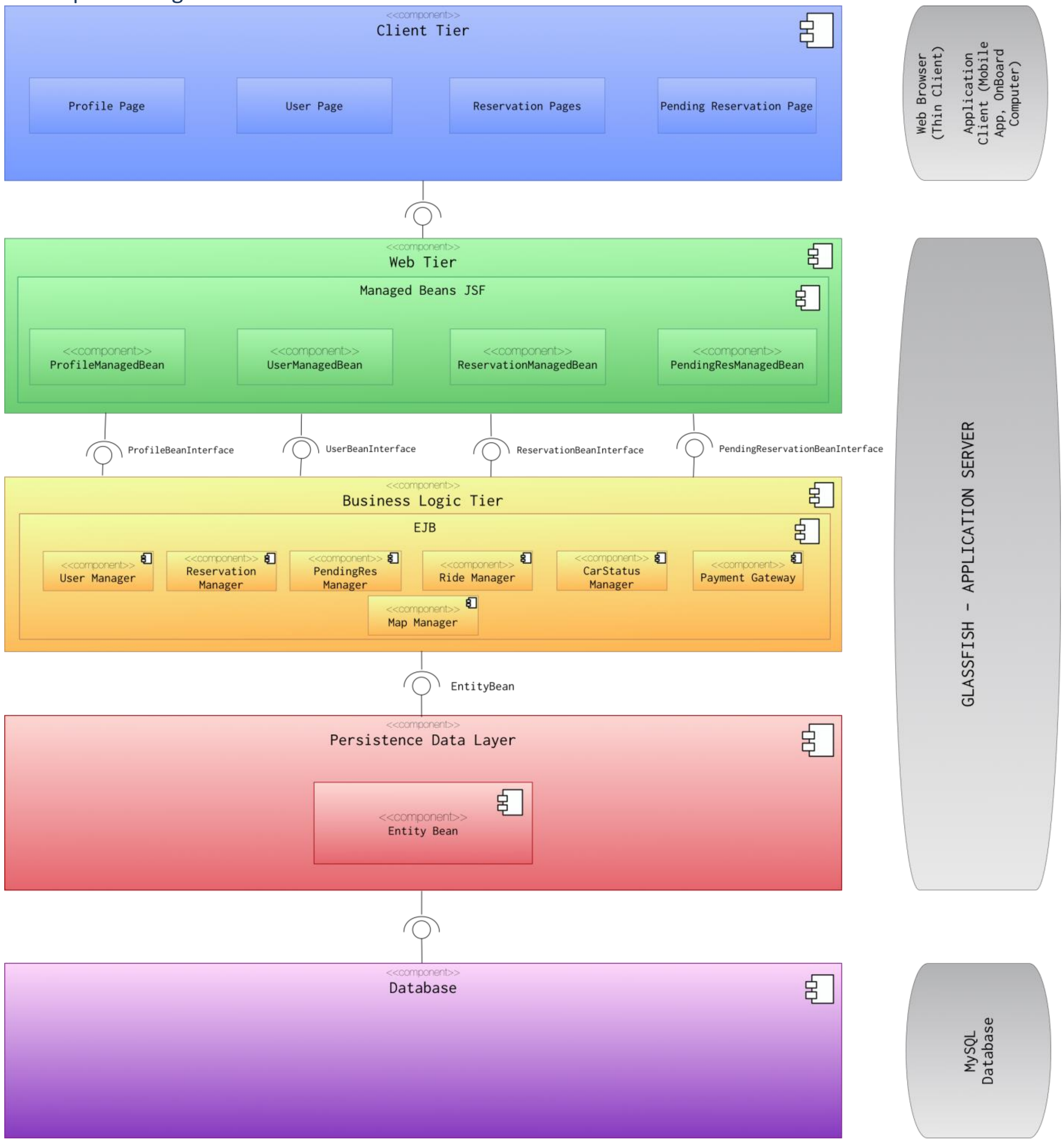

User UI
User Pages
Profile Pages
Reservation Pages
Pending Reservation Pages
Business Logic
Reservation manager
Employees Service manager
User manager
Pending Reservation manager
Persistence Data
Entity manager

High level components and their interaction

PowerEnJoy shall be developed by using a general 3-tier physical architecture, as mentioned in section above. Different components and subcomponents have been identified and classified, in order to have a clearer vision of what these components are going to do. The document will also describe the architecture as it be composed by 5 logical tiers. The approach used is a typical TOP-DOWN analysis. This section gives a general description of the architecture we propose for our system and furthermore it provides a small description of each components. Detailed description is in section Detailed software design.
The next paragraph will study in deep these following points:

- Client Tier and Web Tier represent the Web component.
- Business Logic Tier represents the Business Logic component.
- Persistence Tier represents Persistence Component.
- Database represents the Data Model.

Conceptual design



<<component>>
Client Tier
Profile Page
User Page
Reservation Pages
Pending Reservation Page
Web Browser (Thin Client)
Application Client (Mobile App, OnBoard Computer)
<<component>>
Web Tier
Managed Beans JSF
<<component>> ProfileManagedBean
<<component>> UserManagedBean
<<component>> ReservationManagedBean
<<component>> PendingResManagedBean
ProfileBeanInterface
UserBeanInterface
ReservationBeanInterface
PendingReservationBeanInterface
<<component>>
Business Logic Tier
EJB
<<component>> User Manager
<<component>> Reservation Manager
<<component>> PendingRes Manager
<<component>> Ride Manager
<<component>> CarStatus Manager
<<component>> Payment Gateway
<<component>> Map Manager
EntityBean
<<component>>
Persistence Data Layer
<<component>> Entity Bean
GLASSFISH - APPLICATION SERVER
<<component>>
Database
MySQL Database

The above diagram represents the conceptual design of PowerEnJoy. This diagram is meant to clarify the logical separation between the tiers without showing all the components of the designed software. The final architecture is based on an n-Tier model. It is clear that the user will interact with the XHTML pages and the mobile application. Web Server implements JSF beans to manage user interaction. This web interaction is then supported by the Business tier, which holds on the information provided by the Persistence layer. The Persistence layer will handle the connection to the database and it will manage all the queries needed from the above layers.

Client Tier

The Client Tier is composed by Web Browser (Thin Client) and XHTML pages, but also by Application client (Mobile application). This kind of clients usually interact directly with the business tier, but can also open an HTTP communication channel to interact with the web tier.

Web Tier

The Web Tier is meant to create dynamic pages using Servlet through the JavaServer Faces (JSF/Servlet). It will display data according to the user requests, retrieving the information to be displayed interacting with the Business Tier. It is composed of the web beans, which are part of JavaServer Faces MVC pattern.

Business Logic Tier

The Business Logic Tier is where all the application logic is stored. It is responsible of all the communications between Web Tier and Persistence Tier. Its components are the Enterprise Java Beans (EJB) of GlassFish, named as Managers, just to differentiate from the web beans.

Persistence Data Tier

The persistence tier is composed of the entity beans, which represent the entities described in the RASD document. These entities are fundamental as they represent the connection to our database. They are meant to satisfy certain principles such as information hiding, therefore they should represent a high-level object view of the database application. They will connect this layer to the Database Tier in order to perform insert, update, delete and select operations.

Database

The database is composed of the tables generated using assumptions and needs of the project.

System Specification

The following table displays the technologies will be used during the implementation. All of the technologies are open source technologies.

Component Name	**Technology**
Client Tier	Web Browser and XHTML, Application Client (different program languages but JEE7 compatible)
Web Tier	JavaServer Faces/Servlet
Business Tier	JEE7 with AS GLASSFISH 4.1
Persistence Tier	MySQL 5.7
Database Tier	MySQL 5.7

Component view

In this section, we provide detailed insight into PowerEnJoy structure and implementation. In the following sections, we will provide more details about the general subcomponents, which are object of change in the further phase.

Three main components, which are going to be implemented in the future phases, will compose the PowerEnJoy project.

- Web component
- Business Logic component
- Persistence component

Web Component

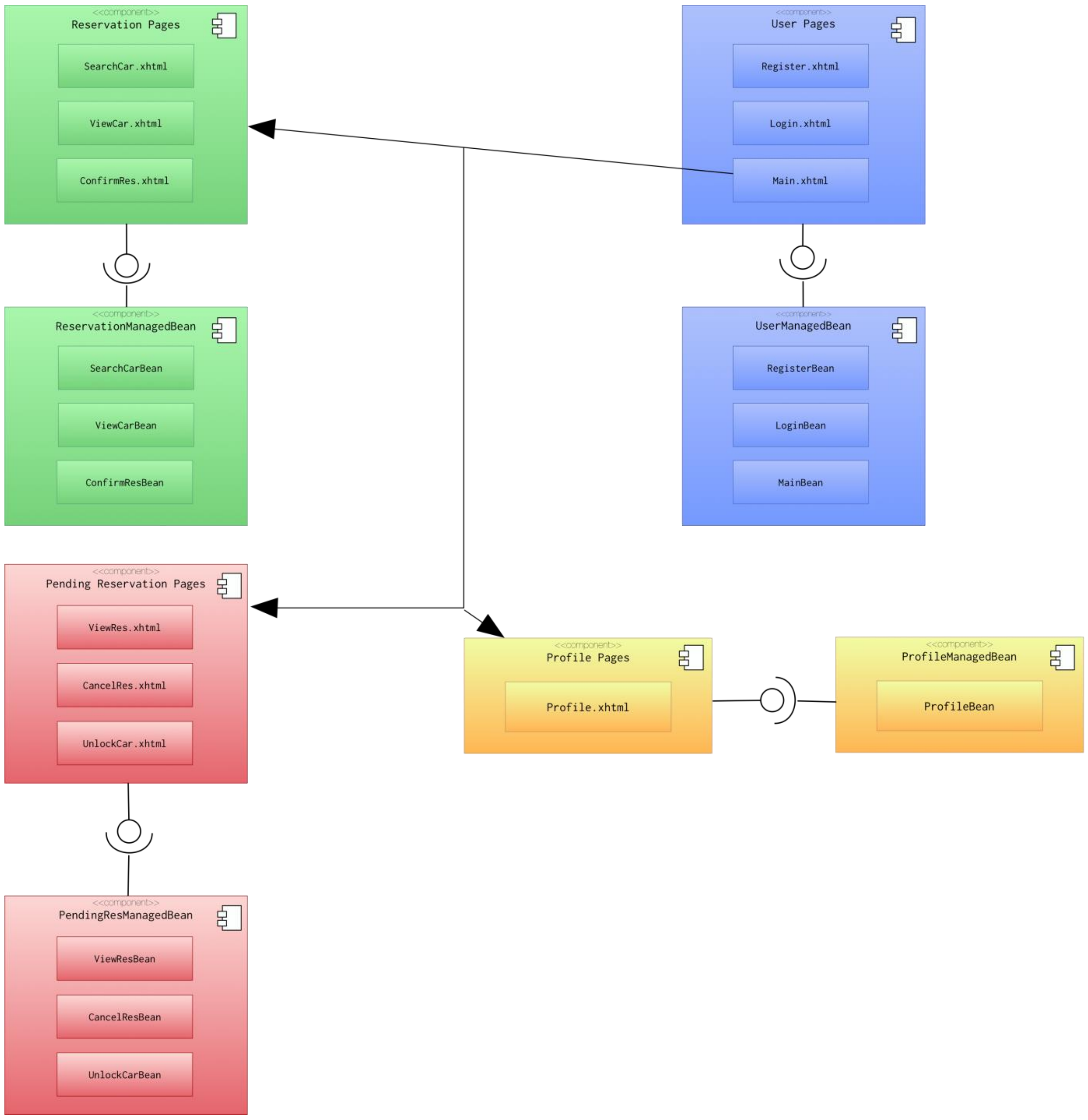
In this section, we provide information about web component implementation. The following diagram shows the subcomponents and their communication.
We have identified four main subcomponents and their related Managed Beans.

- User Pages
- Profile Pages
- Request Pages
- Reservation Pages

The Web Tier uses JSF technologies and consequently Managed Beans manage user events in the pages. The related beans Managed Beans are:

- UserManagedBeans
- ProfileManagedBean
- RequestManagedBean
- ReservationManagedBean

The following component diagram shows in a detailed view the Web Tier subcomponents.


<<component>>
Reservation Pages
SearchCar.xhtml
ViewCar.xhtml
ConfirmRes.xhtml
<<component>>
User Pages
Register.xhtml
Login.xhtml
Main.xhtml
<<component>>
ReservationManagedBean
SearchCarBean
ViewCarBean
ConfirmResBean
<<component>>
UserManagedBean
RegisterBean
LoginBean
MainBean
<<component>>
Pending Reservation Pages
ViewRes.xhtml
CancelRes.xhtml
UnlockCar.xhtml
<<component>>
Profile Pages
Profile.xhtml
<<component>>
ProfileManagedBean
ProfileBean
<<component>>
PendingResManagedBean
ViewResBean
CancelResBean
UnlockCarBean

User Pages and managed beans

Pages and beans in this section are responsible for everything related to users.

Component Name	
Name	Login.xhtml
Description	UI for user login
Responsibility	1. Display login form

Component Name	
Name	Register.xhtml
Description	UI for user registration
Responsibility	1. Load registration form

Component Name	
Name	Main.xhtml
Description	UI for displaying main menu for a registered user
Responsibility	1. Display user main homepage 2. Display connection to Profiles, Reservation and PendingReservations.
Note	Main.xhtml will detect which is the user role (user, employee) and it will display the appropriate homepage.

Component Name	
Name	LoginBean
Description	Managed Bean for user login
Responsibility	1. Load login form 2. Check login parameters 3. Redirect to main.xhtml

Component Name	
Name	RegisterBean
Description	Managed Bean for user registration
Responsibility	1. Load registration form 2. Check registration parameters 3. Redirect to main.xhtml

Component Name	
Name	MainBean
Description	Managed Bean for displaying main menu for a registered user.
Responsibility	1. Load user main homepage 2. Load connections to Profile, Requests, and Reservations pages. 3. Load custom homepage depending

	on user role.

Profile pages and managed beans

Pages and beans in this section are responsible for everything related to profile.

Component Name	
Name	Profile.xhtml
Description	UI interface for displaying use profile information.
Responsibility	1. Display profile information.

Component Name	
Name	ProfileBean
Description	Managed Bean for displaying user profile information.
Responsibility	1. Load profile information based on user role.

Reservation pages and managed beans

Pages and bean in this section are responsible for everything related to reservation.

Component Name	
Name	SearchCar.xhtml
Description	UI interface for searching cars in selected area.
Responsibility	1. Let user search for cars in selected area.

Component Name	
Name	ViewCar.xhtml
Description	UI interface for displaying cars in selected area
Responsibility	1. Display all cars in selected area.

Component Name	
Name	ConfirmRes.xhtml
Description	UI interface for confirming reservation of certain car.
Responsibility	1. Display information to confirm a reservation.

Component Name	
Name	SearchCarBean
Description	Managed Bean for searching of a car.
Responsibility	1. Load search form. 2. Load functionalities for the searching in selected area.

Component Name	
Name	ViewCarBean
Description	Manages Bean for displaying cars in selected area.
Responsibility	1. Check searching parameters 2. Load search request.

Component Name	
Name	ConfirmResBean
Description	Manage Bean for confirming the reservation of a car
Responsibility	1. Load reservation page 2. Display selected car info 3. Check for confirmation request 4. Redirect to main.xhtml

Pending Reservation pages and managed beans

Pages and beans in this section are responsible for everything related to reservations.

Component Name	
Name	ViewRes.xhtml
Description	UI interface for displaying user reservations.
Responsibility	1. Display Reservations information.

Component Name	
Name	CancelRes.xhtml
Description	UI interface for cancelling a car reservation
Responsibility	1. Display information to cancel a reservation.

Component Name	
Name	UnlockCar.xhtml
Description	UI interface for unlocking a car.
Responsibility	1. Display information to unlock a car.

Component Name	
Name	ViewResBean
Description	Managed Bean for list of reservations.
Responsibility	1. Load reservations. 2. Load functionalities for the reservations.

Component Name	
Name	CancelResBean
Description	Manages Bean for cancel a reservation.
Responsibility	1. Load reservation form 2. Check reservation parameters 3. Redirect to main.xhtml

Component Name	
Name	UnlockCarBean
Description	Manage Bean for unlocking a car
Responsibility	1. Load reservation form 2. Check unlocking requirements 3. Redirect to main.xhtml

Business Logic component

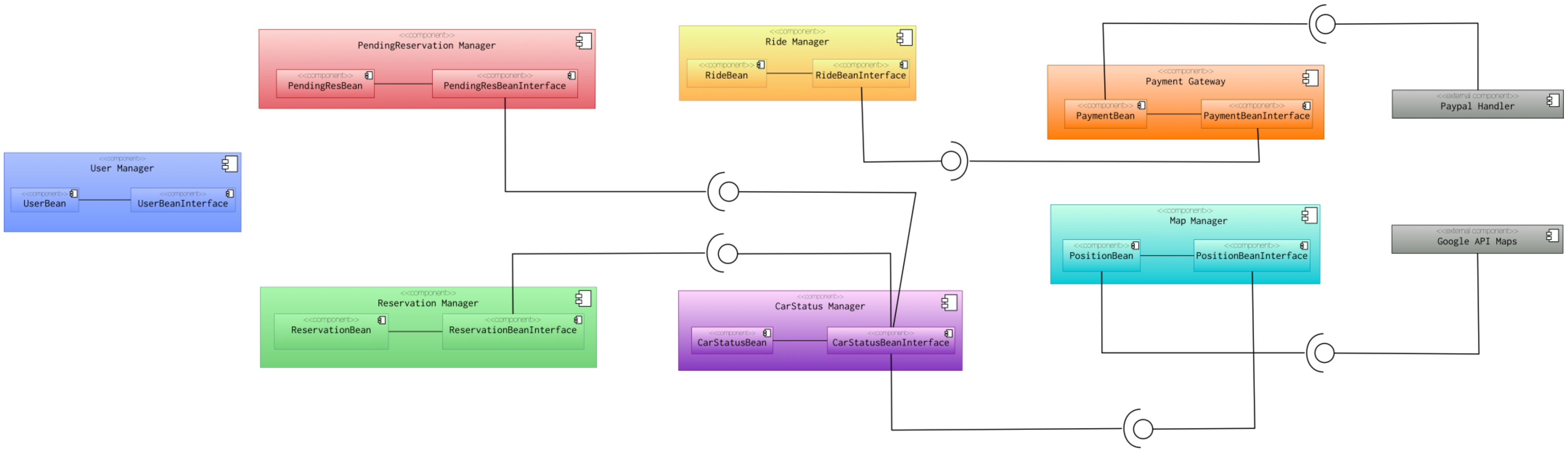
In this section, we provide information for what needs to implement Business Logic components. The following general diagram shows the subcomponents and their communication. There are four main subcomponents.

- User Manager
- Ride Manager
- Reservation Manager
- PendingReservation Manager
- CarStatus Manager
- Payment Gateway
- Map Manager

External systems:

- Paypal Handler
- Google API Maps

Each of these beans will be defined as an EJB Bean.



<<component>>
PendingReservation Manager
<<component>>
PendingResBean
<<component>>
PendingResBeanInterface
<<component>>
Ride Manager
<<component>>
RideBean
<<component>>
RideBeanInterface
<<component>>
Payment Gateway
<<component>>
PaymentBean
<<component>>
PaymentBeanInterface
<<external component>>
Paypal Handler
<<component>>
User Manager
<<component>>
UserBean
<<component>>
UserBeanInterface
<<component>>
Map Manager
<<component>>
PositionBean
<<component>>
PositionBeanInterface
<<external component>>
Google API Maps
<<component>>
Reservation Manager
<<component>>
ReservationBean
<<component>>
ReservationBeanInterface
<<component>>
CarStatus Manager
<<component>>
CarStatusBean
<<component>>
CarStatusBeanInterface

Component Name	
Name	UserManagerBean
Description	Manages all the functionalities related to the administration of the users.
Responsibility	1. User’s registration 2. User’s login 3. User’s logout 4. Account cancellation 5. Info update

Component Name	
Name	UserManagerBeanInterface
Description	Interface is intantiated every time communication between beans is needed.
Responsibility	

Component Name	
Name	ReservationManagerBean
Description	Manages the functionalities related to the search of a car and the car’s reservation. Moreover manages the eventual cancellation of the reservation.
Responsibility	1. Search car in selected position 2. Search car using GPS 3. Reserve a car

Component Name	
Name	ReservationManagerBeanInterface
Description	Interface is instantiated every time communication between beans is needed.
Responsibility	1. Communicate with CarStatusBeanInterface

Component Name	
Name	PendingResManagerBean
Description	Manages all the functionalities provided to the user from the moment of the moment of the reservation to the actual starting of the ride.
Responsibility	1. Cancel a reservation 2. Provide remaining reservation time 3. Unlock the car 4. Provide car position 5. Apply reservation expiration fees

Component Name

Name	PendingResBeanInterface
Description	Interface is intantiated every time communication between beans is needed
Responsibility	1. Communicate with CarStatusBeanInterface

Component Name	
Name	RideManagerBean
Description	Manages all functionalities provided during the ride.
Responsibility	1. Start the engine 2. Display current payment charge 3. Display current battery state of charge 4. Compute final payment charge applying possible fees or discounts

Component Name	
Name	RideManagerBeanInterface
Description	Interface is intantiated every time communication between beans is needed
Responsibility	1. Communicate with PaymentGatewayBeanInterface

Component Name	
Name	CarStatusManagerBean
Description	Manages the status of cars in terms of availability, state of charge and position.
Responsibility	1. Verify low battery state of charge of cars 2. Verify in city boundaries position of cars 3. If needed communicate to employees to move a certain car 4. Verify availability status of a car

Component Name	
Name	CarStatusManagerBeanInterface
Description	Interface is intantiated every time communication between beans is needed
Responsibility	1. Communicate with PositionBeanInterface

Component Name	
Name	PaymentGatewayBean

Description	Whenever the system has to charge some payment to a user this bean provides this functionality
Responsibility	1. Charge ride payment 2. Charge reservation expiration fee

Component Name	
Name	PaymentGatewayBeanInterface
Description	Interface is intantiated every time communication between beans is needed
Responsibility	

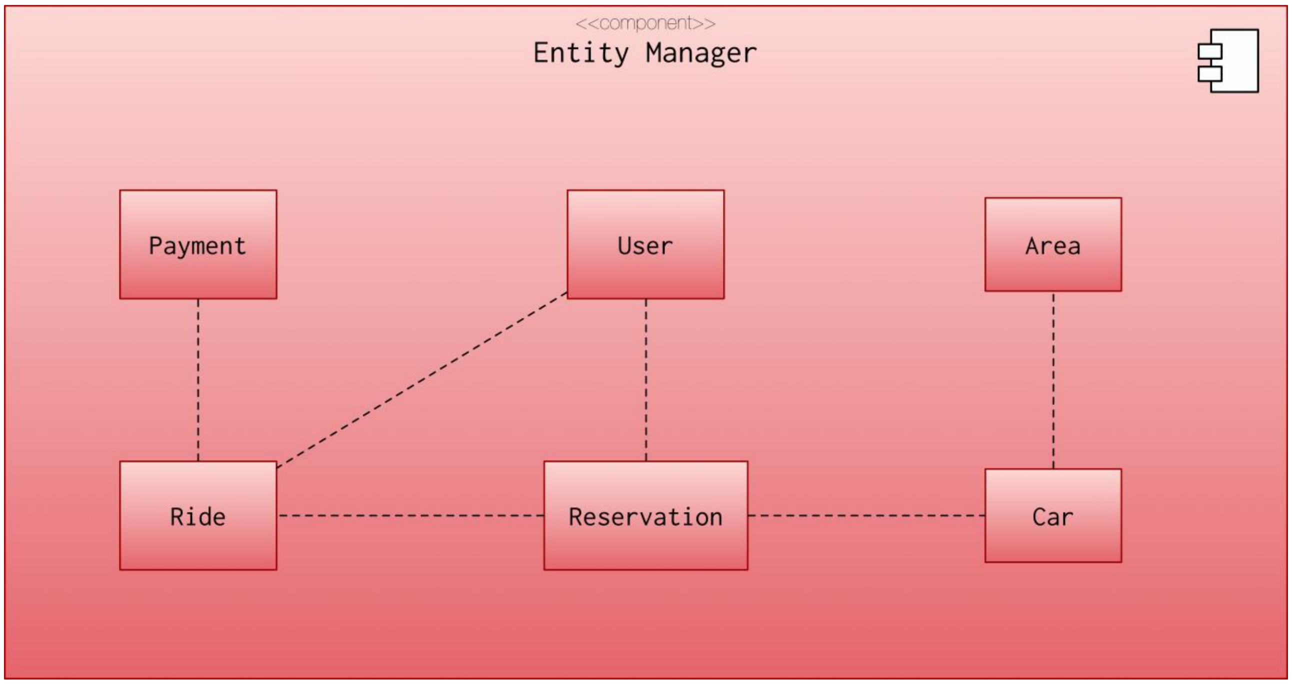
Component Name	
Name	MapManagerBean
Description	Manages all functionalities related to the positioning of cars
Responsibility	1. Update car position 2. Find position of a selected car 3. Receive car position auto update

Component Name	
Name	MapManagerBeanInterface
Description	Interface is intantiated every time communication between beans is needed
Responsibility	

Component Name	
Name	PaypalHandler
Description	External component that manges the final part of payment
Responsibility	1. Finalize the payment

Component Name	
Name	GoogleApiHandler
Description	External component that provides maps to the system
Responsibility	1. Allows to locate given positions in the maps

Persistence component



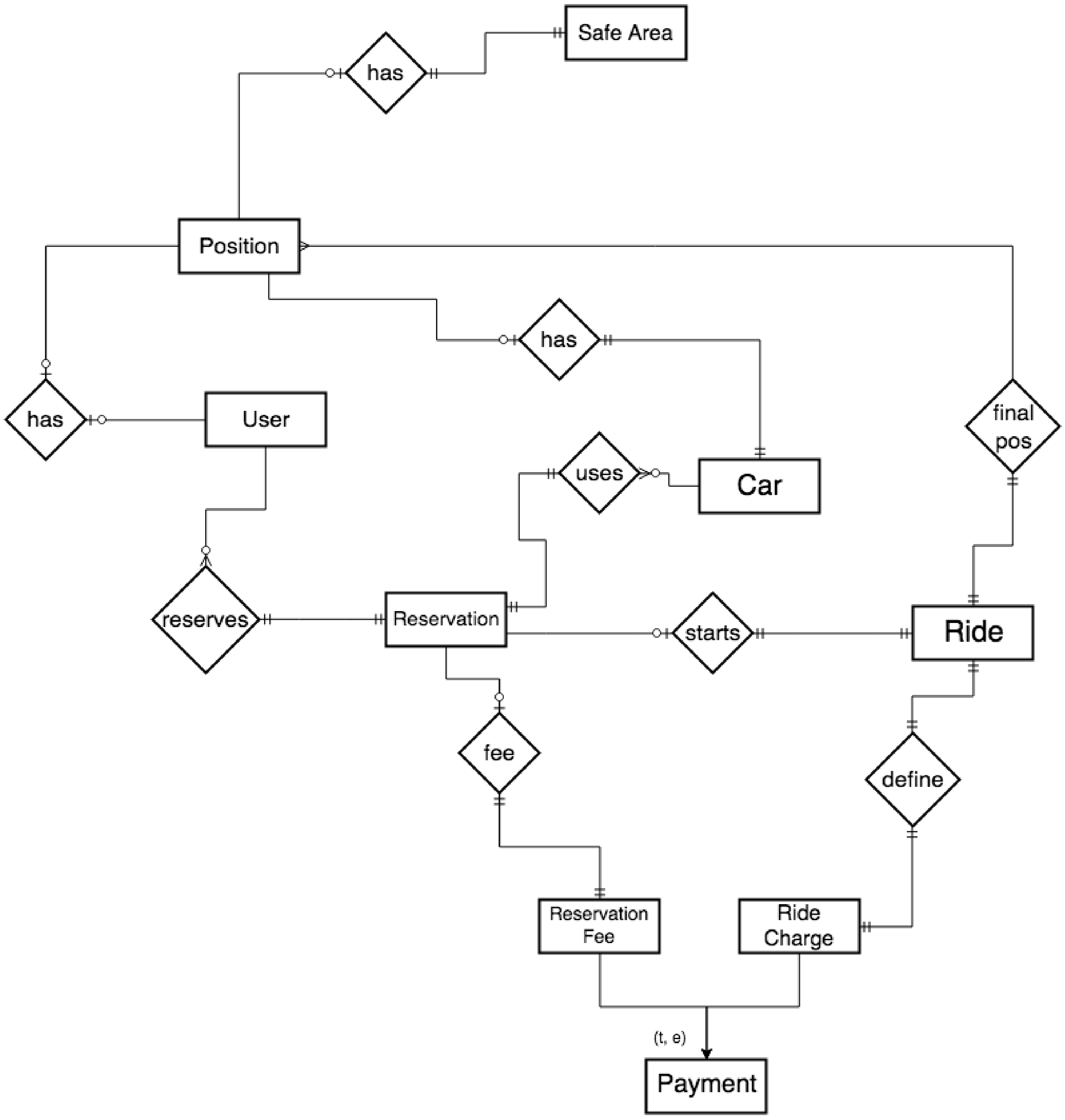
The Persistence component diagram above shows all the entities in the application. All of these entities represent a high-level object of the tables in the database, which will be used in our application. Therefore, all of the entities have inside all the attributes, which are in the associated table in the database.
In the following table, there is a short description of the entities:

Entity	Description
User	Represent the users in the system
Car	Represent the cars in the system
Reservation	Represent the reservations in the system
Ride	Represent the rides in the system
Area	Represent the safe and charging area in the system
Payment	Represent the payments in the system

Database Model Structure

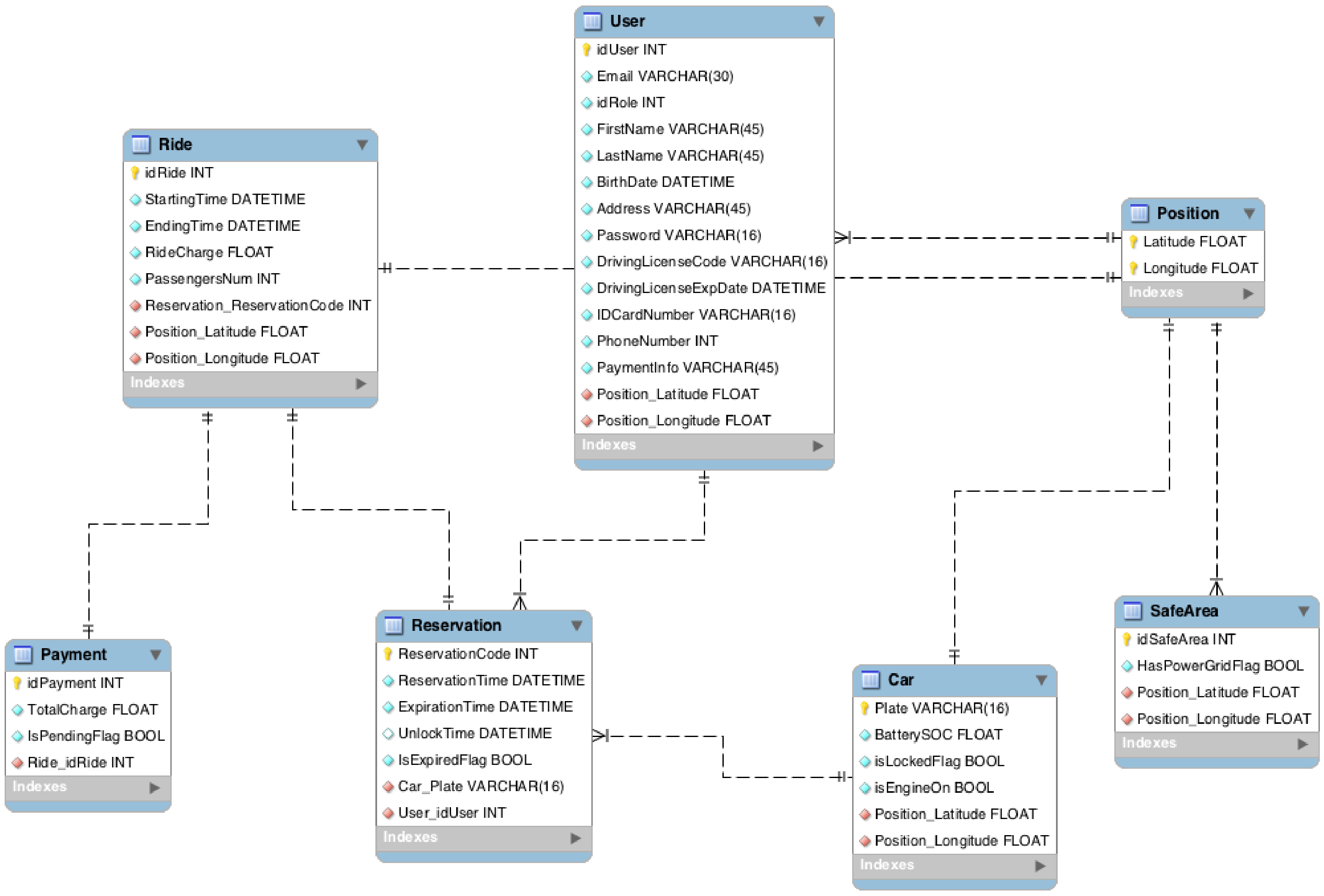
In the following section, we will provide a detail description of the database by providing its both views: the conceptual and logical design diagrams. The Entity Relationship Diagram (ER) will represent the conceptual design of the database.

Entity Relationship Diagram



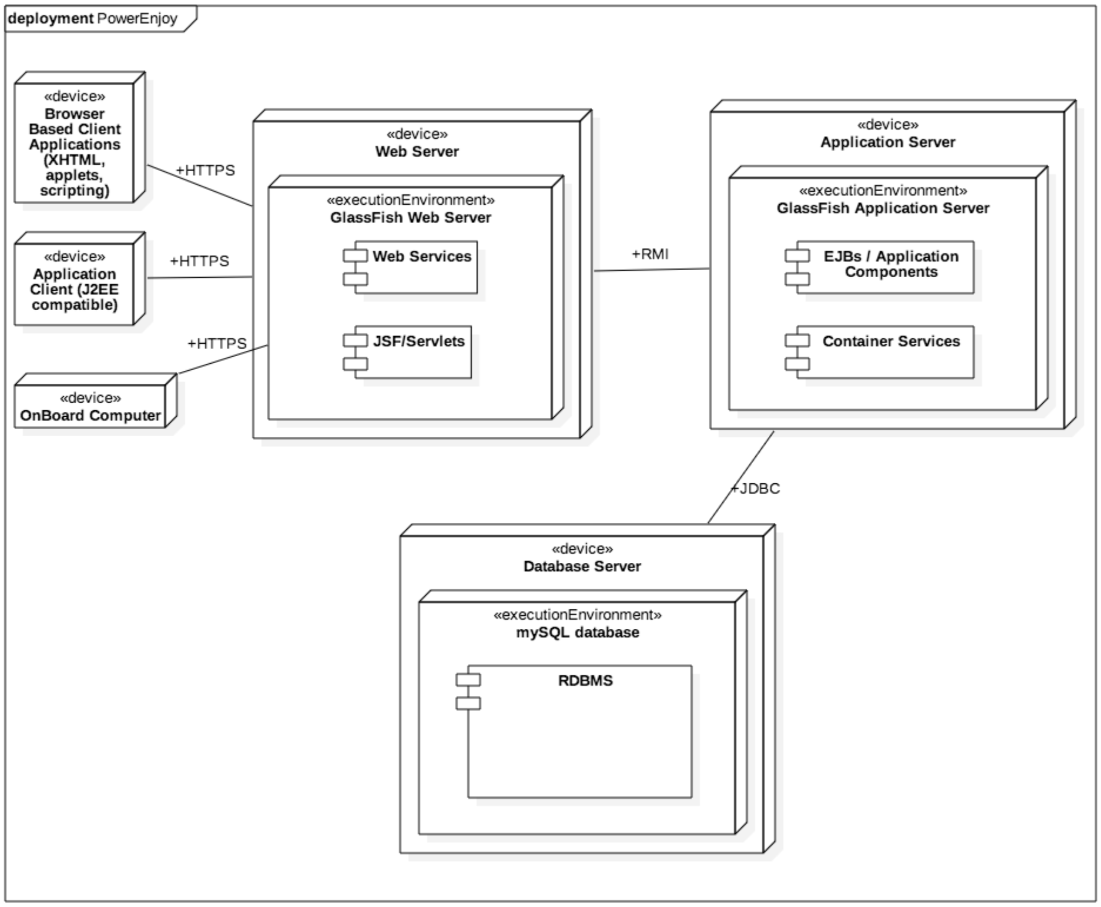
Logic Design

The logical design (LD) diagram represents the final implementation of the database. It is based on the previous ER diagram.



Deployment view

This section will provide a possible solution of deployment architecture.


deployment PowerEnjoy
«device» Browser Based Client Applications (XHTML, applets, scripting)
«device» Application Client (J2EE compatible)
«device» OnBoard Computer
+HTTPS
+HTTPS
+HTTPS
«device» Web Server
«executionEnvironment» GlassFish Web Server
Web Services
JSF/Servlets
+RMI
«device» Application Server
«executionEnvironment» GlassFish Application Server
EJBs / Application Components
Container Services
+JDBC
«device» Database Server
«executionEnvironment» mySQL database
RDBMS

The diagram above summarize devices and connection between the architecture components. The major features of the platform are:

- A multi-tiered application model
- A server-side component model

The client tier comprises Internet browsers, which submits HTTP request and downloads HTML pages from a Web Browser, application clients and the onboard computer. These last are developed with different technologies and they send HTTP request to the Web Server, but they must be compatible with the J2EE structure.

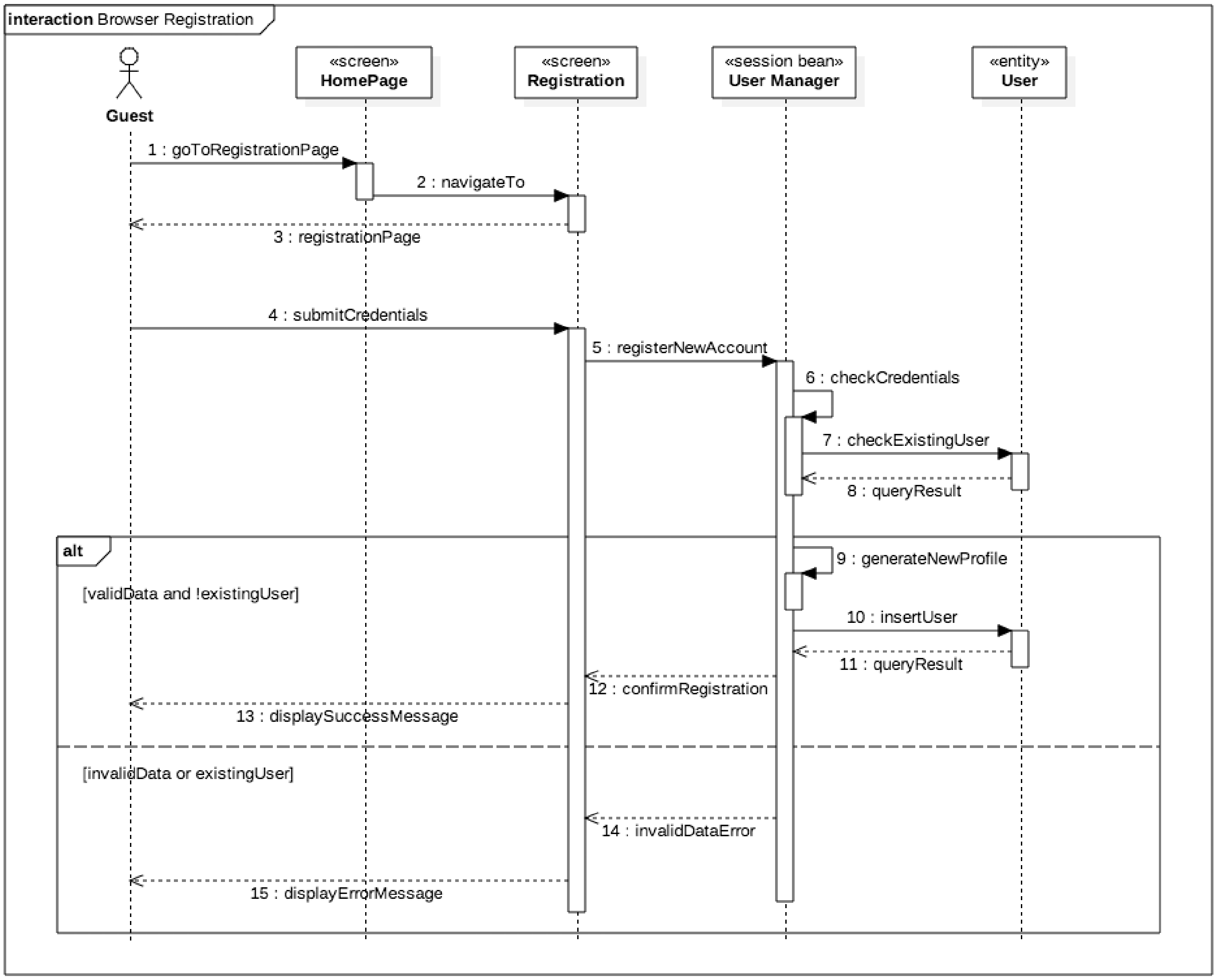
The Web tier runs a Web server to handle client requests and responds to these requests by invoking J2EE servlets or JavaServer Faces (JFPs). The server depending on the type of user request invokes Servlets. They query the business logic tier for the required information to satisfy the request and then format the information for return to the user via the server.

The business components of the Application Server comprise the core business logic for the application. They are realized by EJBs. EJBs receive requests from servlets in the Web tier, satisfy them usually by accessing some data sources, and return the results to the servlet. EJB components are hosted by a J2EE environment known as the EJB container.
Two different and separated machine host respectively the web server and the application server. They communicate with an implementation of Java RMI API. Developers can write remote interfaces between clients and servers and implement them using Java technology and the Java RMI APIs.

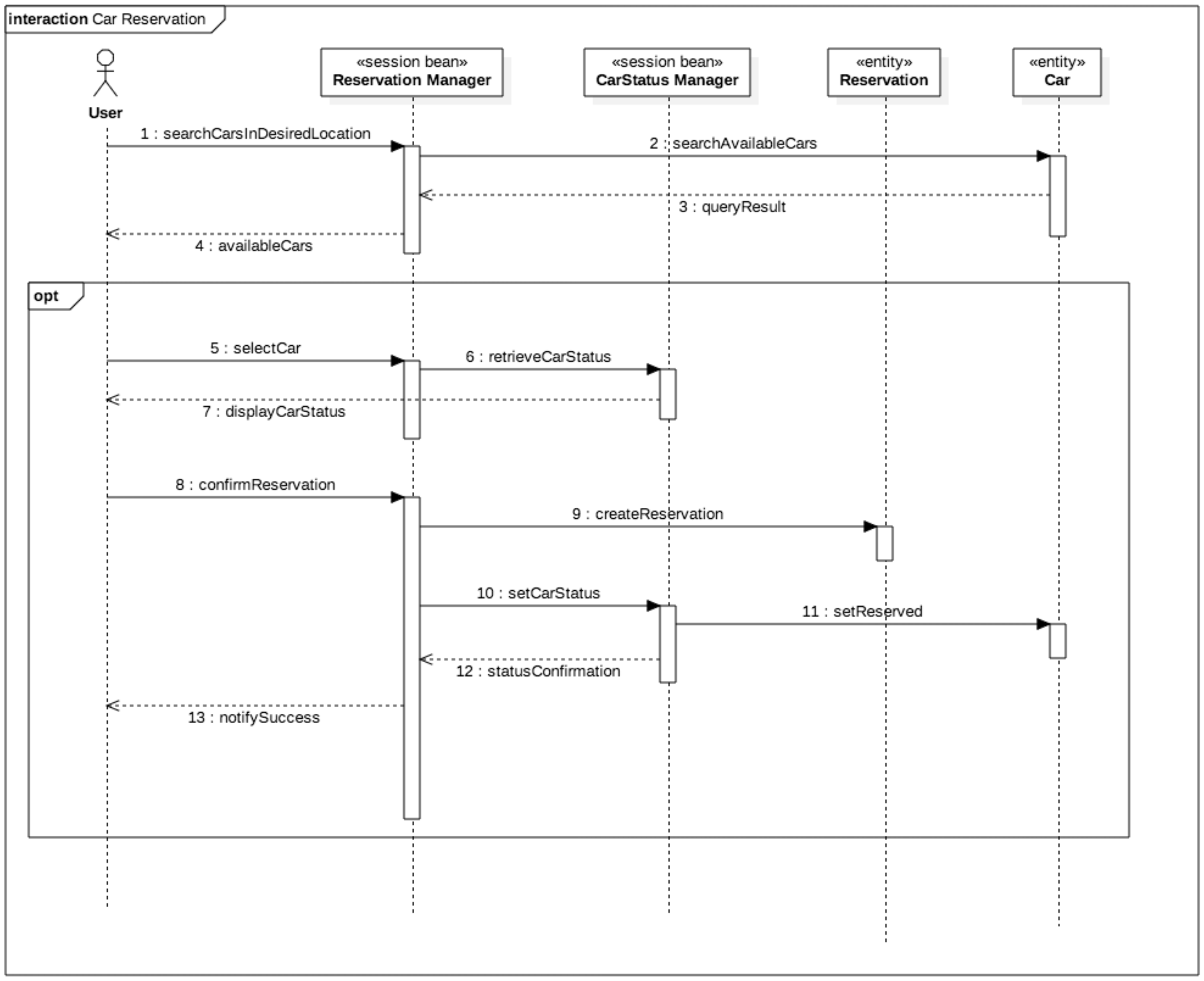
Another machine host the database tier, which consists of one or more relational databases. The application servers queries the RDBMS (Relational Database Management System) using Java Database Connectivity (JDBC).

Runtime view

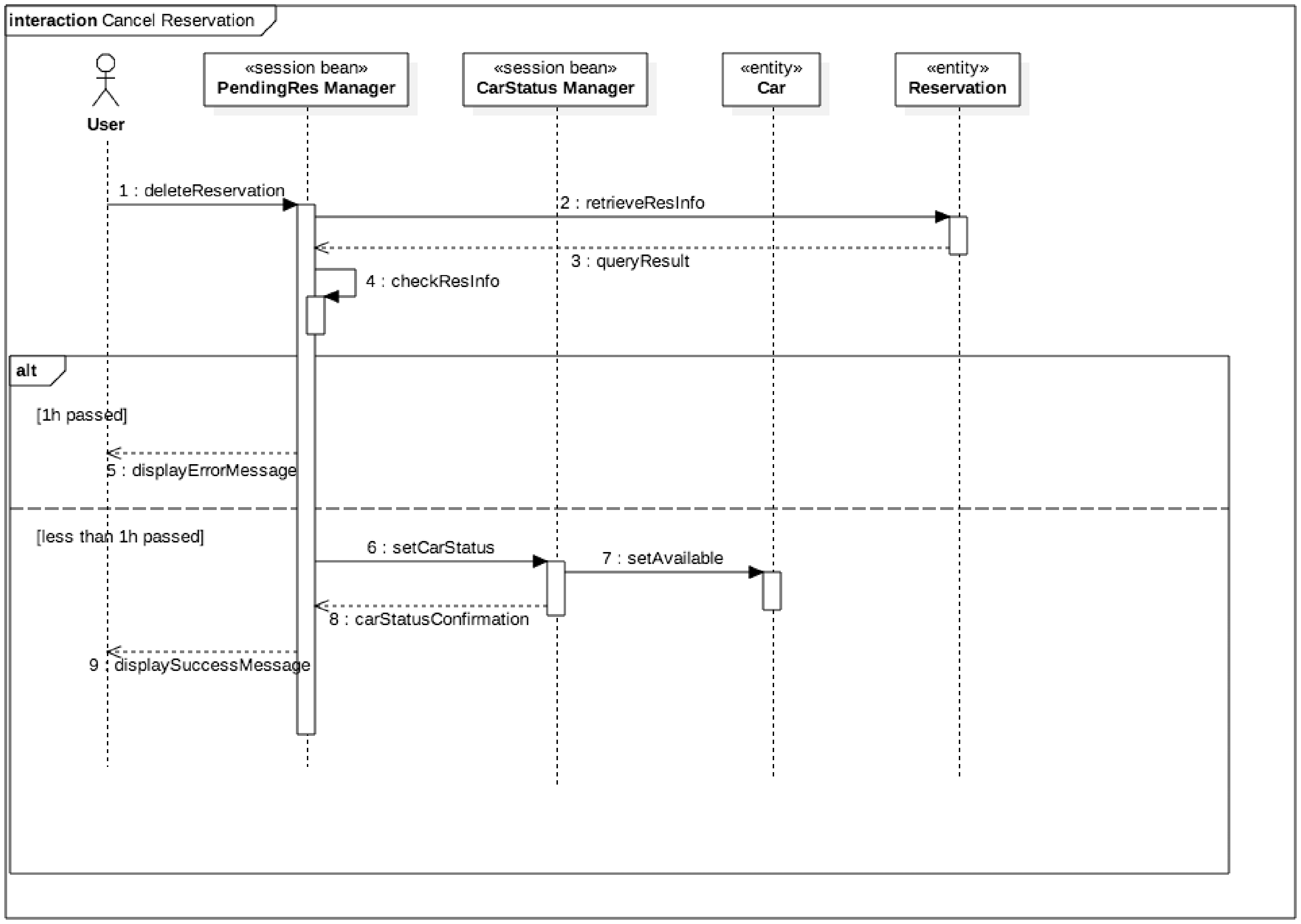
This section describes the dynamic behaviour of the system in the most relevant cases. The following sequence diagrams highlight the runtime interactions between clients, servers and the database.



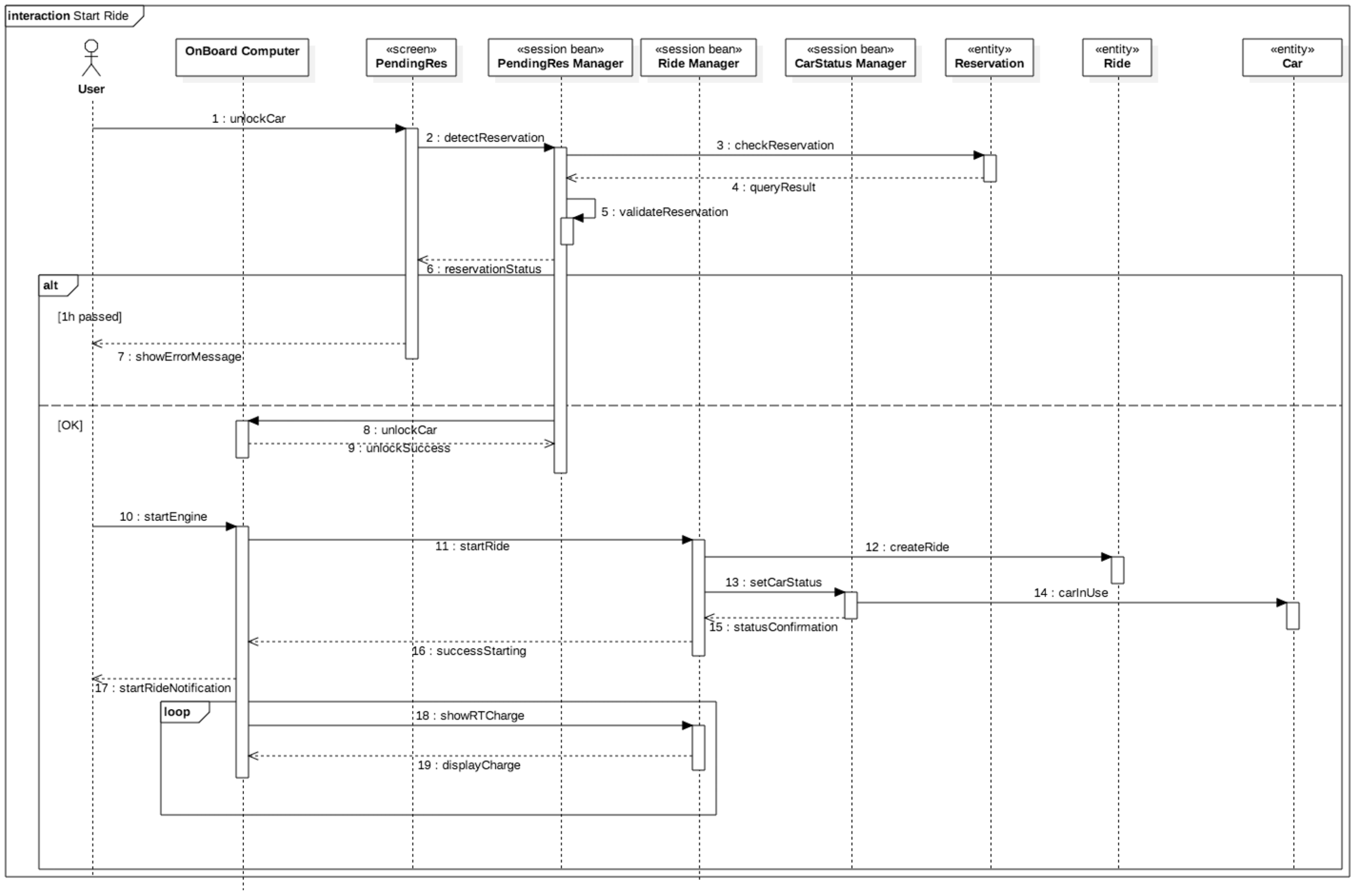
Sequence diagram of the registration process via the web browser



Sequence Diagram of the car search and reservation process

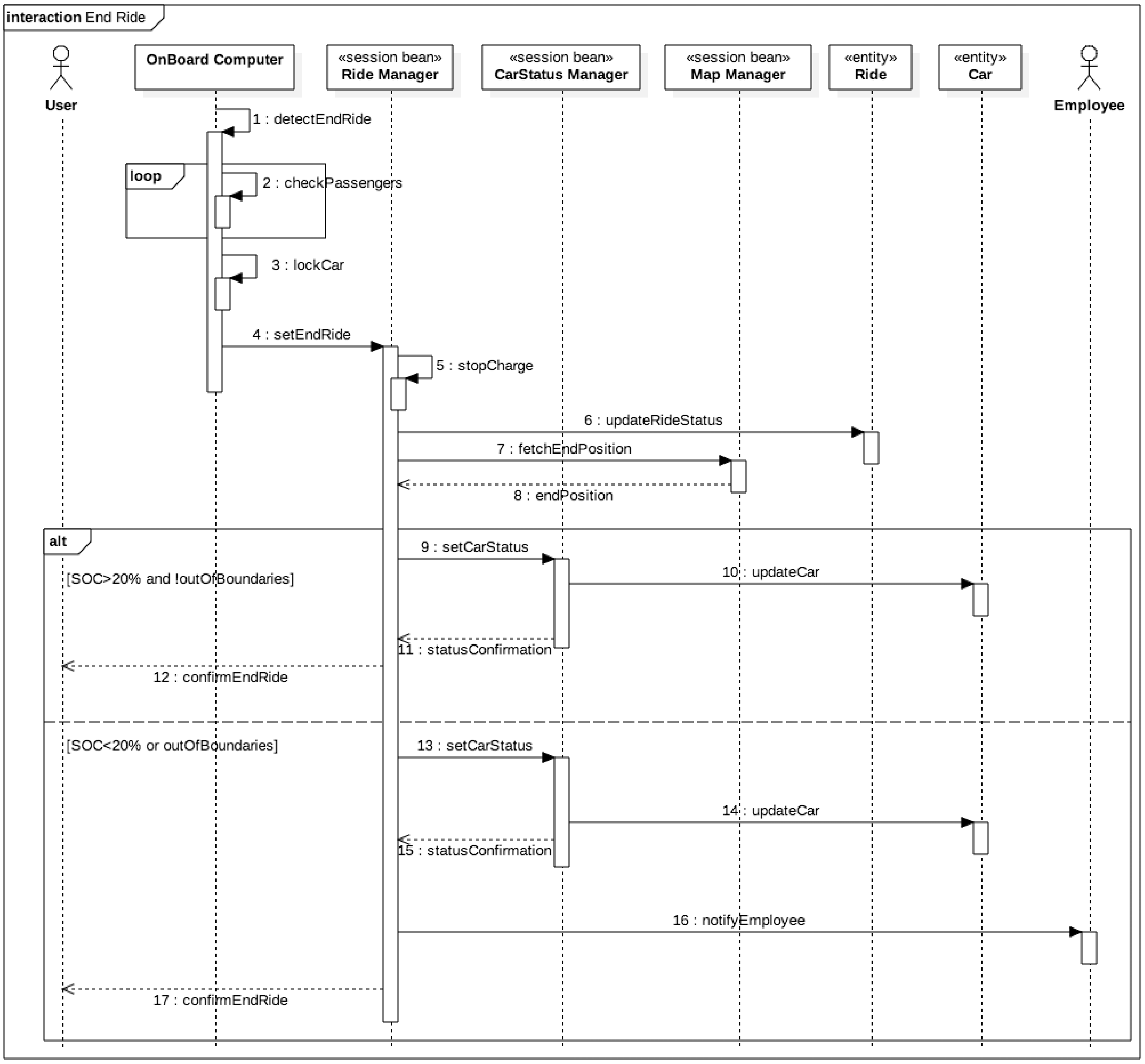


Sequence Diagram of the cancel reservation process

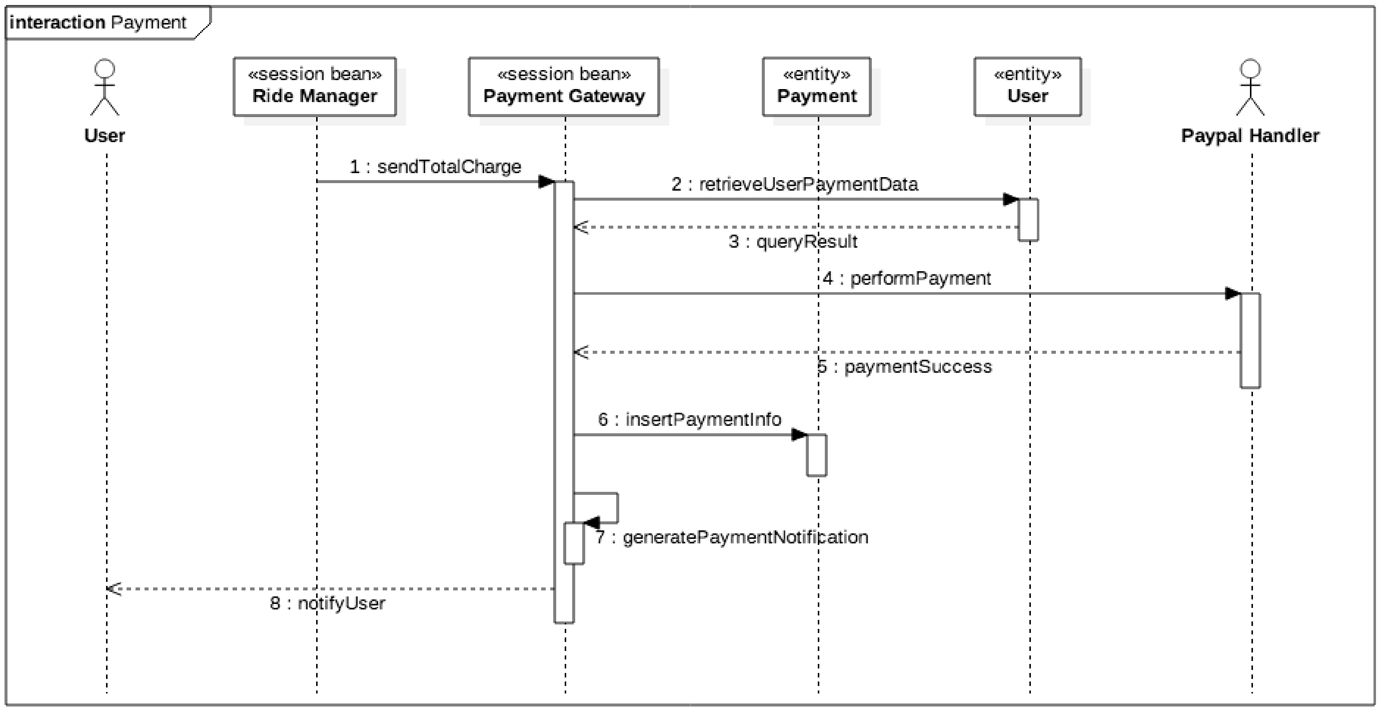


Sequence Diagram of the start ride process[1]

[1] Even though it is not modeled in this sequence diagram, when actually implementing the software must be present a check of the code shown by the car and inserted in the app to verify that the user is actually nearby.



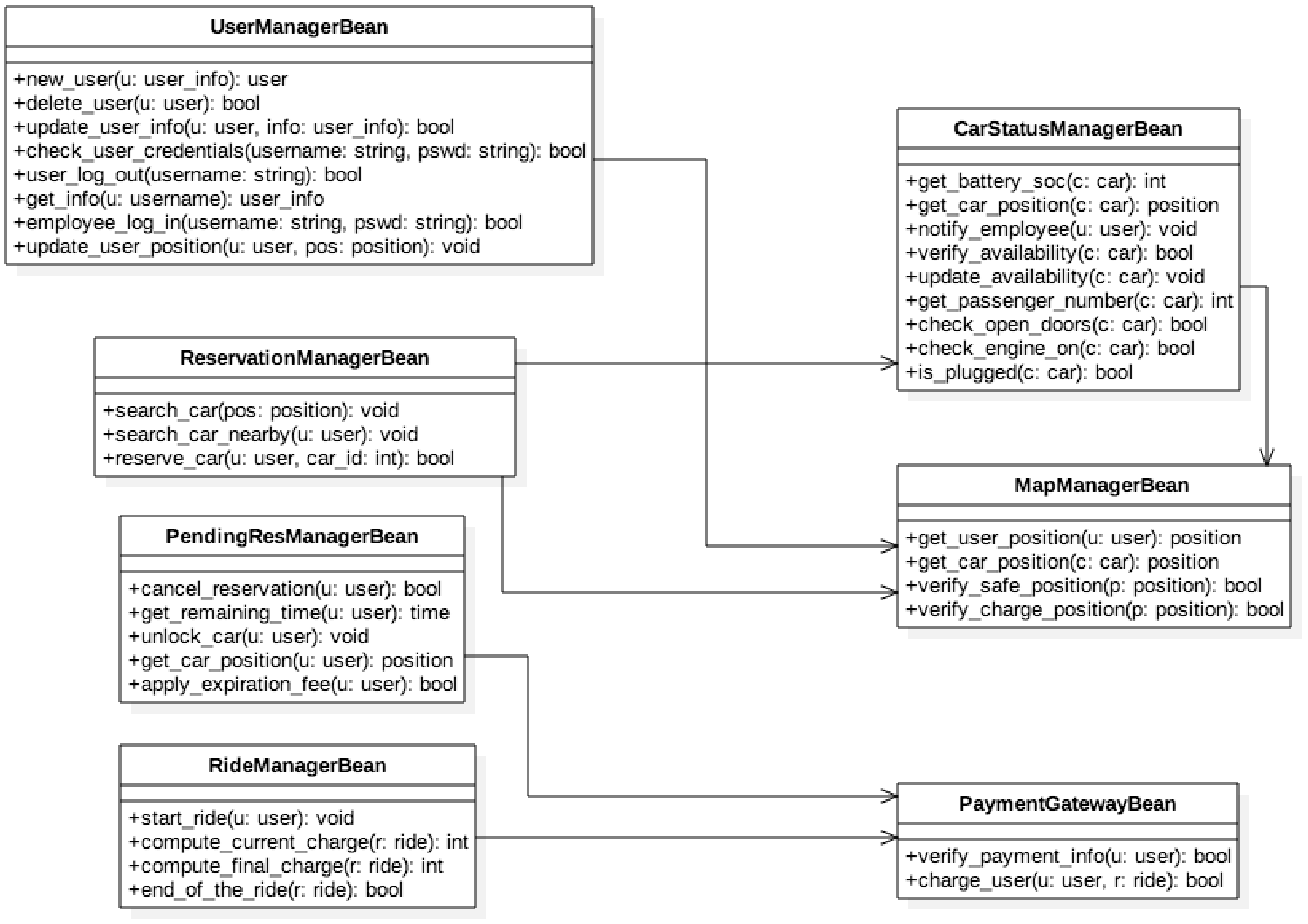
Sequence Diagram of the end ride process [See the definitions paragraph to understand when a ride ends]



Sequence Diagram of the payment process

Component interfaces

This section describes the interfaces of the project components defining the main methods that each EJB will implement and also describing they're main interaction. When actually implementing the system further methods and interfaces can be defined.


UserManagerBean
+new_user(u: user_info): user
+delete_user(u: user): bool
+update_user_info(u: user, info: user_info): bool
+check_user_credentials(username: string, pswd: string): bool
+user_log_out(username: string): bool
+get_info(u: username): user_info
+employee_log_in(username: string, pswd: string): bool
+update_user_position(u: user, pos: position): void
CarStatusManagerBean
+get_battery_soc(c: car): int
+get_car_position(c: car): position
+notify_employee(u: user): void
+verify_availability(c: car): bool
+update_availability(c: car): void
+get_passenger_number(c: car): int
+check_open_doors(c: car): bool
+check_engine_on(c: car): bool
+is_plugged(c: car): bool
ReservationManagerBean
+search_car(pos: position): void
+search_car_nearby(u: user): void
+reserve_car(u: user, car_id: int): bool
MapManagerBean
+get_user_position(u: user): position
+get_car_position(c: car): position
+verify_safe_position(p: position): bool
+verify_charge_position(p: position): bool
PendingResManagerBean
+cancel_reservation(u: user): bool
+get_remaining_time(u: user): time
+unlock_car(u: user): void
+get_car_position(u: user): position
+apply_expiration_fee(u: user): bool
RideManagerBean
+start_ride(u: user): void
+compute_current_charge(r: ride): int
+compute_final_charge(r: ride): int
+end_of_the_ride(r: ride): bool
PaymentGatewayBean
+verify_payment_info(u: user): bool
+charge_user(u: user, r: ride): bool

Selected architectural styles and patterns

In this section, we shortly described the patterns used during the design of the system.

The main used patterns are:

- Top down
- Multi-tier
- Client server
- MVC

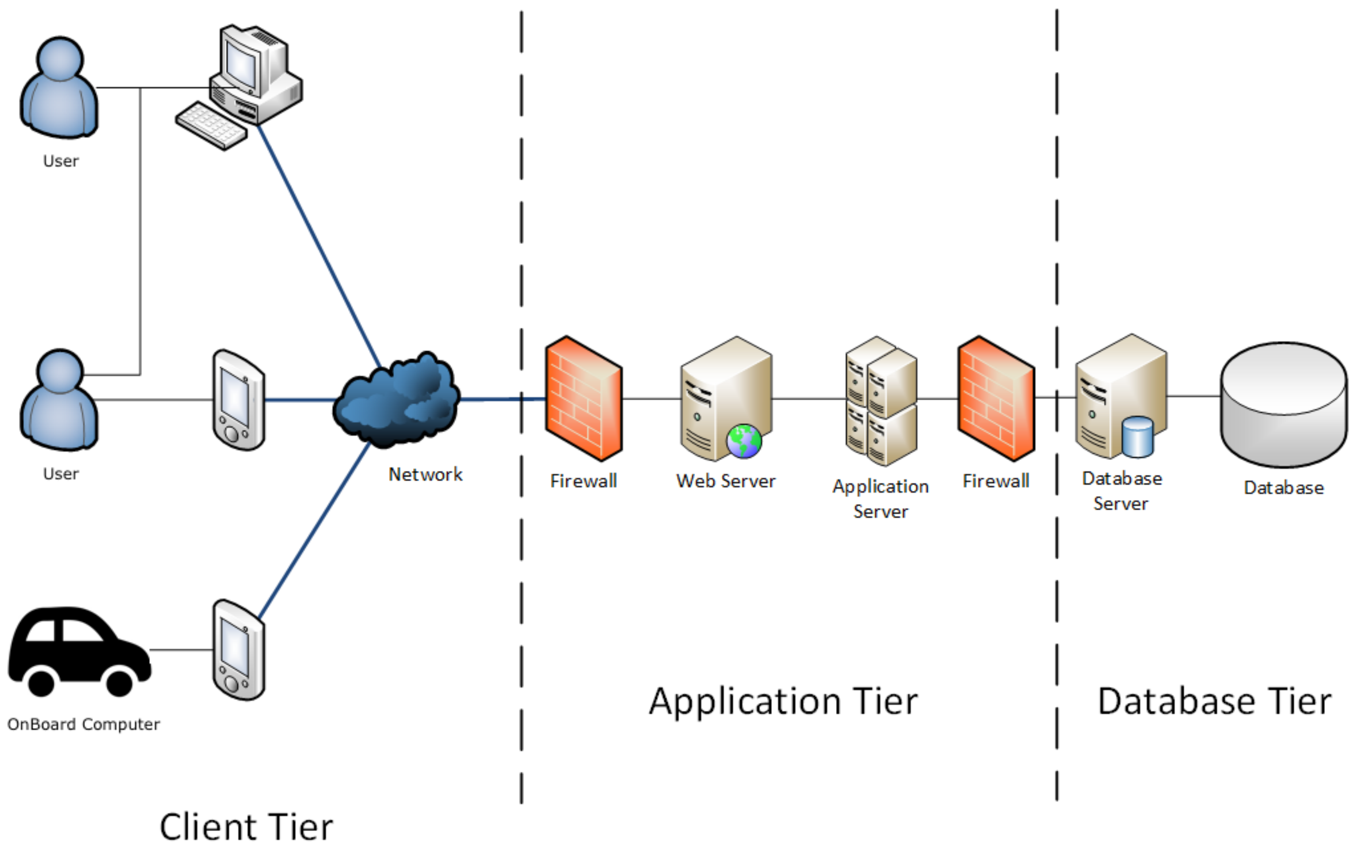
Top down

The used approach to the architecture definition is Top-Down that means that we started from defining the main macro-elements (mainly from the multi-tier point of view) toward the lower-level ones.
The main exception to this way of reasoning is about the management of payments and of maps: as those functionalities are assumed to be provided by external components and therefore they are present from the starting of the design phase.

Multi-tiered architecture

Our system, as shown in the picture below, is organized in five different logical tiers:

- Client tier
- Web tier
- Business logic tier
- Database



This choice is made in order to provide hiding information and simplify the task of each of them up to the user.
The role of each of them is here specified:

Client tier

The client tier comprises an Internet browser that submits HTTP requests and downloads HTML pages from a Web server. In an application not deployed using a browser, standalone Java clients or applets can be used; these communicate directly with the business component tier.

Web tier

The Web tier runs a Web server to handle client requests and responds to these requests by invoking J2EE servlets or JavaServer Faces (JSFs). The server, depending on the type of user request, invokes Servlets. They query the business logic tier for the required information to satisfy the request and then 30 format the information for return to the user via the server. The code is invoked by the JSF mechanism and takes responsibility for formatting the dynamic portion of the page.

Business logic tier

The business components comprise the core business logic for the application. They are realized by EJBs. EJBs receive requests from servlets in the Web tier, satisfy them by accessing some data sources, and return the results to the servlet. EJB components are hosted by a J2EE environment known as the EJB container, which supplies a number of services to the EJBs it hosts including transaction and life-cycle management, state management, security, multi-threading, and resource pooling. The Business Application servers uses EJB and session bean in order to fulfill its tasks.

There are different kinds of EJB that provide different functionalities; here they are listed:

Session beans contain business logic and provide services for clients. The two types of session bean are stateless and stateful. The application uses both the two types of session bean.
- A stateless session bean does not keep any state information of any client. A client will get a reference to a state less session bean in a container and can use it to make many calls on an instance of the bean. However, between each successive service invocation, a client is not guaranteed to bind to any particular stateless session bean instance. The EJB container is delegated to call to stateless session beans as needed, so the client can never be certain which bean instance it will actually talk to.
In this project for instance the user manager beans will possibly be stateless as they are not supposed to permanently store themselves user's info but just to write them in the database.

- A stateful session bean can maintain state information about the conversation. Once a client gets a reference to a stateful session bean, subsequent calls to the bean using this reference are guaranteed to go to the same bean instance. The container creates a new, dedicated stateful session bean for each client that creates a bean instance. Therefore, clients can store any state information they wish in the bean and can be assured that it will still be there the next time they access that bean. EJB containers assume responsibility for managing the life cycle of stateful session beans. In the project, this kind of beans will be possibly used for managing rides as there is the necessity of keeping track on the real-time evolution of some information.

Entity beans are used for representing business data objects. In an entity bean, the data members map directly to some data items stored in the associated database. Entity beans are accessed by a session bean that provides business-level client services. The data, which the bean represents, are mapped automatically to the associated persistent data store (e.g. the database) by the container. The container is responsible for loading the data to the bean instance and writing changes back to the persistent data storage at appropriate times, such as the start and end of a transaction.

Database tier

This consists of one or more databases, which EJBs must query to process requests. JDBC drivers are used for databases, which are Relational Database Management Systems (RDBMS).

In the below table, we resume some pros and cons of the proposed structure:

	ADVANTAGES	**DISASVANTAGES**
Development point of view	• Complex application rules easy to implement in application server. • Business logic disjoint from database server and client, which improves performance. • Changes to business logic automatically enforced by server – changes require only new application server software to be installed. • Application server logic is portable to other database server platforms. • Greater degree of flexibility • Security can be defined for each service, and at every level	• A complex structure to develop, harder to be built. • It is more difficult to set up and maintain.
Performance point of view	• Superior performance for medium to high volume environments. • Increased performance, as tasks are shared between servers	• The physical separation of application servers, containing business logic functions, and database servers, containing databases, may moderately affect performance

Client server

The client server pattern is a natural choice in this application. The idea is that many different users can communicate through they're devices with the centralized system which allows them to rent the cars. Moreover, the system coordinates in a centralized way the employees to make them manage low-state-of-charge or out-of-boundaries cars.

In the below table, we resume some pros and cons of the proposed structure:

ADVANTAGES	**DISADVANTAGES**
• Centralization: a centralized server is able to properly	• Network load: too many clients requests can overload the

coordinate car reservation requests • All permanent data are stored in a unique place that allows easier management and search • Concentration of data also allows for easier back-up • Scalability and upgrade becomes trivial: adding users does not imply changing the architecture and upgrades can be easily made on the server • Accessibility: users can access the service from different platforms	network and decrease system performances • Server is critical: this architecture is exposed to server fails that can make the whole service stop

MVC pattern

The proposed architecture implements the Model-View-Control pattern in which software elements are divided in order to perform 3 different tasks.

- **Model:** represents the business data that govern access to and modification of the data. The model notifies views when it changes and lets the view query the model about its state. It also lets the controller access application functionality encapsulated by the model.
  In the presented architecture, the 'model' task is performed by the persistence data layer and the databases.
- **View**: renders the contents of a model. It gets data from the model and specifies how that data should be presented. It updates data presentation when the model changes. A view also forwards user input to a controller.
  In the presented architecture, the 'view' task is performed by the client tier.
- **Controller**: The controller defines application behaviour. It dispatches user requests and selects views for presentation. It interprets user inputs and maps them into actions to be performed by the model. A controller selects the next view to display based on the user interactions and the outcome of the model operations.
  In the presented architecture, the 'controller task' is performed by the web tier and the business logic tier

The pros of the MVC pattern are:

- Separation of design concerns: this separation allows flexibility, modification on only one of the components will minimally affect others.
- Maintainability: components are associated to precise tasks making easier to decrease complexity.
- Promotes division of labours: developers with different skill sets are able to focus on their core skills and collaborate through clearly defined interface.

Algorithm design

In this section, we will list the features that are critical from the point of view of the functionalities provided by the system. Moreover, it will be defined why they are critical and which properties they're implementation has to satisfy.

We focused on the sequent features:

- Availability of a car
- Computation of the charge amount of a ride

Availability of a car

This feature is critical, as the system must strictly avoid that different users reserve the same car (this would make the service lose many clients).
A car is said to be not-available if it is either reserved, being used in a ride, in a low state of charge or out of city boundaries.
The main criticality emerges from the distribution of the system: different users may ask for a reservation of the same car at the same time as in that moment the car is actually available. The system must detect this situation and assign the car to only one of them.

Computation of charge amount of a ride

This feature is critical, as applying the wrong charge would strongly damage the quality of the PowerEnJoy or significantly decrease the income of the service provider.
The charging amount related to a ride is defined as follows:

- Firstly, it is computed as proportional to the time duration of the ride (and this is the charge amount to be displayed on real time on the car)
- At the end on the ride eventual discount or fee situations must be detected
- Among those only the biggest discount is applied
- Differently all fees are going to be applied
- As they are expressed in percentage fees and discounts must all be applied with respect the amount computed proportionally to the time

Here below are listed the discount and fee situations with the associated price variation.

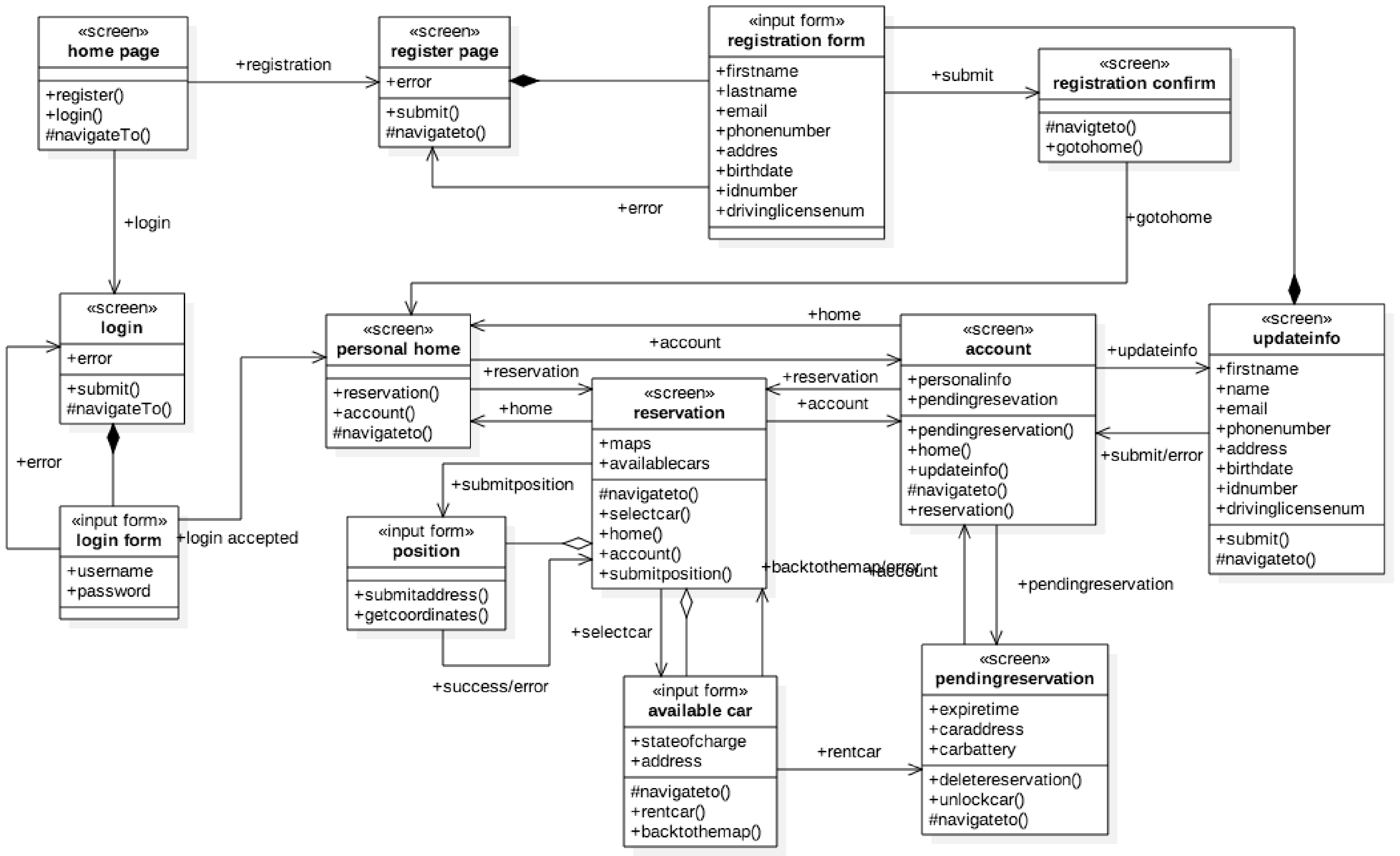
DISCOUNTS	
Passengers during main part of the ride were strictly greater than 2 (driver included)	-10%
At the end of the ride the battery state of charge is strictly higher than 50%	-20%
At the end of the ride the car is left in a charging area and is plugged in	-30%

FEES	
At the end of the ride the car is left more than 3km far from the nearest charging station	+30%
At the end of the ride the car is left with a battery state of charge strictly lower than 20%	+30%
At the end of the ride the car is left out of city boundaries or in a parking which doesn't belong to the safe areas	+20%

Note that if a car is left plugged in a charging station but with low battery the discount and the fee will compensate (which seems to be reasonable).

User interface design

This section presents the user interfaces of the web application that will be implemented on the client side. The formalism used to describe the UX is the class diagram and it describes how the web application is supposed to appear to the user.



Requirements traceability

This section recalls all the functional requirements that were pointed out in the RASD document and maps them to the project packages.

Product functions

Product functionalities can be divided in 3 sets:

- Managing users
- Managing cars
- Managing rides (car choice, reservation, ride info, payment)

Managing users

[FR1]: register through both mobile app and web browser
 [FR1.1]: define and manage a password to access the service
[FR2]: update personal info
[FR3]: update payment info
[FR4]: delete their profile
[FR5]: manage at the same time users with multiple roles (the active user and the employee)
[FR6]: user can cancel his reservation within 1h

Managing the cars

[FR7]: know on real time of each car's state of charge
 [FR7.1]: the system must be able to communicate via web with the car
[FR8]: know on real time each car position
 [FR8.1]: cars must be provided with a geo-localization system sufficiently accurate
[FR9]: communicate to the employees to go and move an out of charge car to a power grid station
 [FR9.1]: the system must be able to communicate with the employees
[FR10]: the system must tell employees to bring back a car that has been left outside the city boundaries

Managing rides

[FR11]: allow users to find available cars using current location
 [FR11.2]: determine whether a car is available or not
[FR12]: allow users to find available cars around a manually selected location
[FR13]: allow users to reserve a single car per time
 [FR13.1]: a reserved car must be marked as un-available
 [FR13.2]: allow user to cancel a reservation
[FR14]: allow users to unlock cars
[FR15]: make users know on real time the battery state of charge
[FR16]: make users know on real time the payment amount
 [FR16.1]: the system must compute on real time the cost of the ride
 [FR16.2]: the system has to automatically determine discounts and fines
  [FR16.2.1]: if multiple discounts are available, only one of them is going to be applied

FUNCTIONAL REQUIREMENTS	IMPLEMENTED FUNCTIONALITIES						
	USER MANAGER BEAN	RESERVATION MANAGER BEAN	PENDING RESERVATION BEAN	RIDE MANAGER BEAN	CARSTAUS MANAGER BEAN	PAYMENT GATEWAY BEAN	MAP MANAGER BEAN
FR 1	X					X	
FR 1.1	X						
FR 2	X						
FR 3	X					X	
FR 4	X						
FR 5	X						
FR 6			X				
FR 7					X		
FR 7.1					X		
FR 8					X		X
FR 8.1					X		X
FR 9					X		X
FR 9.1					X		
FR 10					X		
FR 11		X					X
FR 11.1		X					
FR 12		X					X
FR 13		X					
FR 13.1		X			X		
FR 13.2			X				
FR 14			X		X		
FR 15				X	X		
FR 16				X			
FR 16.1				X		X	
FR 16.2				X			
FR 16.2.1				X			

Effort spent

Madaffari Federico

- 26/11/16 (3h) group work
- 28/11/16 (1.5h)
- 1/12/16 (2h)
- 3/12/16 (4h) group work
- 4/12/16 (3h) group work
- 5/12/16 (3h) group work
- 7/12/16 (5h) group work
- 11/12/16 (1.5h)

Mandrioli Claudio

- 25/11/16 (1h)
- 26/11/16 (3h) group work
- 1/12/16 (1.5h)
- 3/12/16 (4h) group work
- 4/12/16 (3h) group work
- 5/12/16 (3h) group work
- 7/12/16 (5h) group work
- 11/12/16 (1.5h)