

PowerEnJoy

Design Document

A.Y 2016/2017

Francesco Tinarelli(matr:806146)

Marco Wenzel(matr:878021)

Versione1.1

**Index**

1)Introduction------------------------------------------------------------------3

1.1) Purpose--------------------------------------------------------------3

1.2) Scope-----------------------------------------------------------------3

1.3) Glossary-------------------------------------------------------------4

1.4) Reference Documents--------------------------------------------4

1.5) Document Structure----------------------------------------------4

2)Architectural Design-------------------------------------------------------5

2.1) General Overview-------------------------------------------------5

2.2) Component View--------------------------------------------------6

2.3) Runtime View-----------------------------------------------------13

2.4) Component Interfaces-------------------------------------------17

2.5) Deployment View------------------------------------------------19

2.6) Architectural style and pattern-------------------------------20

3)Code Design------------------------------------------------------------------21

4) User interface Design-----------------------------------------------------24

4.1) Ux diagram--------------------------------------------------------24

4)Requirements Traceability-----------------------------------------------26

5)Used tools--------------------------------------------------------------------29

6) Hours of work--------------------------------------------------------------29

1. Introduction

1.1 Purpose

The Design Document’s purpose is to provide a fully specification of the architecture used for requirements implementation.

The architecture of a software system defines that system in terms of computational components and the interactions among those components.

This document permits to identify:

* High-Level architecture components and their interaction
* The runtime behaviour
* Architectural styles and patterns decision

**1.2 Scope**

We will project and implement PowerEnJoy, it’s a digital management system for a car-sharing service.

This system is based on mobile application and web application, there are two kind of different people that can use the system: Unregistered user or Registered user.

Unregistered user can only visit the web application and register himself on the system.

The system allows registered user to see the available cars in a specific zone or current position area using GPS and through this service is possible to make a reservation for up to 1 hour.

The user can unlock the reserved car logging on the web application and using “unlock car” button, the system provides this operation if the user is nearby to the reserved car.

The system registered some special safe area called: power grid area which is possible plug the car.

The system during the ride show the current charges money on the car screen.

More over the screen show power grid stations and permits to choose the path for the nearest grid station or ones specified by the user.

In addition to the functionality above, the system should incentivize the virtuous behaviours of the users, in fact, if the user provides to plug the car or in generally take care of the car, the system rewards him with discount.

There is no system before this one.

1.3 Glossary

**External Interface:** System interface that permits interaction among other systems.

**API:** Application Programming interface; it’s a common way to communicate with other system, (application running on external interface).

**Component**: this terminology is used in component diagram to specify a piece of the system software that aims to provide a specific service.

**MVC**: model view controller.

**Tier**: it’s a specific software level, it’s a parameter that permits to identify architectural style.

**Client-Server**: is a multi-tier architecture.

**Rasd**: Requirements analysis and specification document.

**Central System:** the business tier part, aims to manage the data for providing services.

**Front-end applications**: terms to indicate all devices belongs to Client tier (Mobile application, web application, screen).

**Service manager**: terms to indicate all component (in component view) aims to provide a service and checks the requirements for this one.

1.4 Reference Documents

For creating this DD document, we used:

* Design document notions: Design part I.pdf, Design part II.pdf
* Structure of Document: Assignments2016/2017.pdf
* JEE implementations: JavaEEOverview.pdf
* Example: Sample design deliverable.pdf

**1.5 Document Structure**

The document is composed by several parts:

1. **Introduction:** brief introduction for describe the document’s scope and purpose.

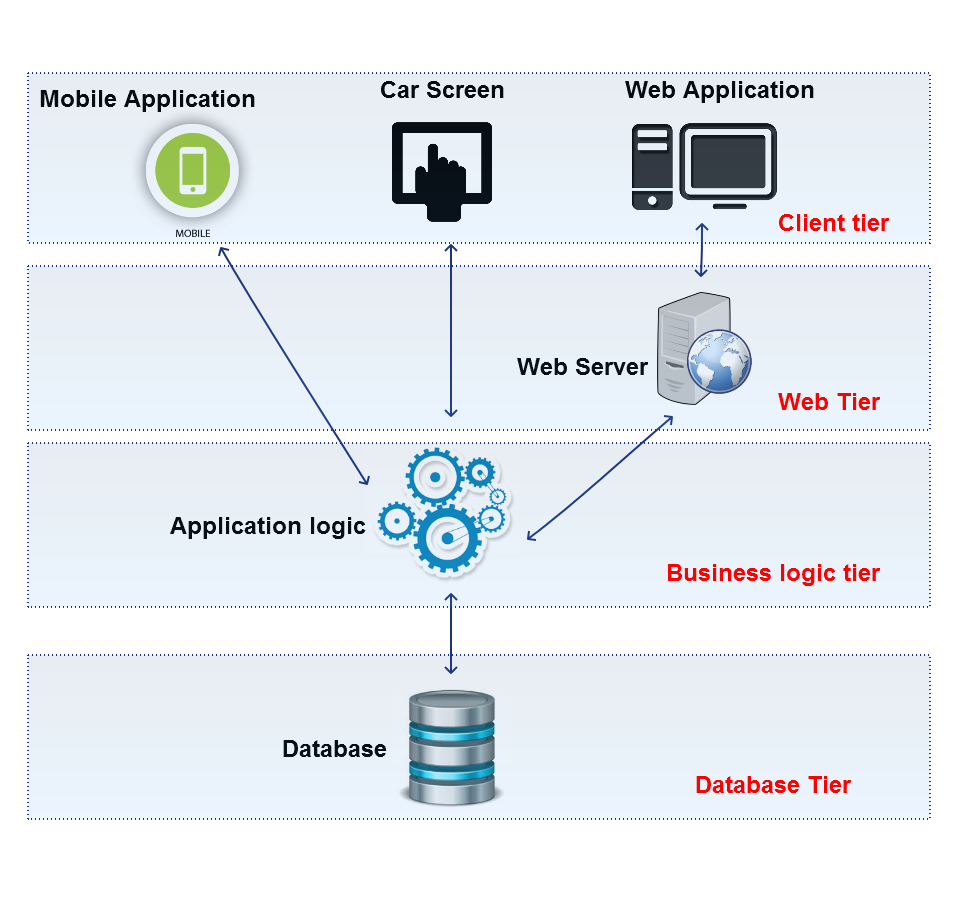
A glossary for explain most common terms used into this document.

1. **Architectural design:** top-down description, for explain the architectural choice and the interaction among components.

Moreover, a dynamic view architecture.

1. **Algorithm Design:** description using pseudocode the most relevant code part.
2. **Requirements traceability:** Explain how the requirements defined in the RASD are provided through architectural components.
3. **Effort Spent:** how much time we spend for redact this document.
4. Architectural Design

2.1 Overview



We add a tier for (and only) web server because the web site must be dynamic, the services must be dynamically provided from the system, the web pages will be creating in web server.

This architecture represents the devices that need to provide services but among tiers there will be protection component like firewall.

We choose to use Client-Server Architecture, the tiers belong to Server-side are: database, business logic and web server.

This architecture doesn’t want represent a physical division but a conceptual division based on software system.

E.g.: there will be a part of system code nested into every car (and provide several service), but this not represent Client tier because it doesn’t have a directly interaction with user. External Interface can be considered part of Client tier.

2.2 Component View

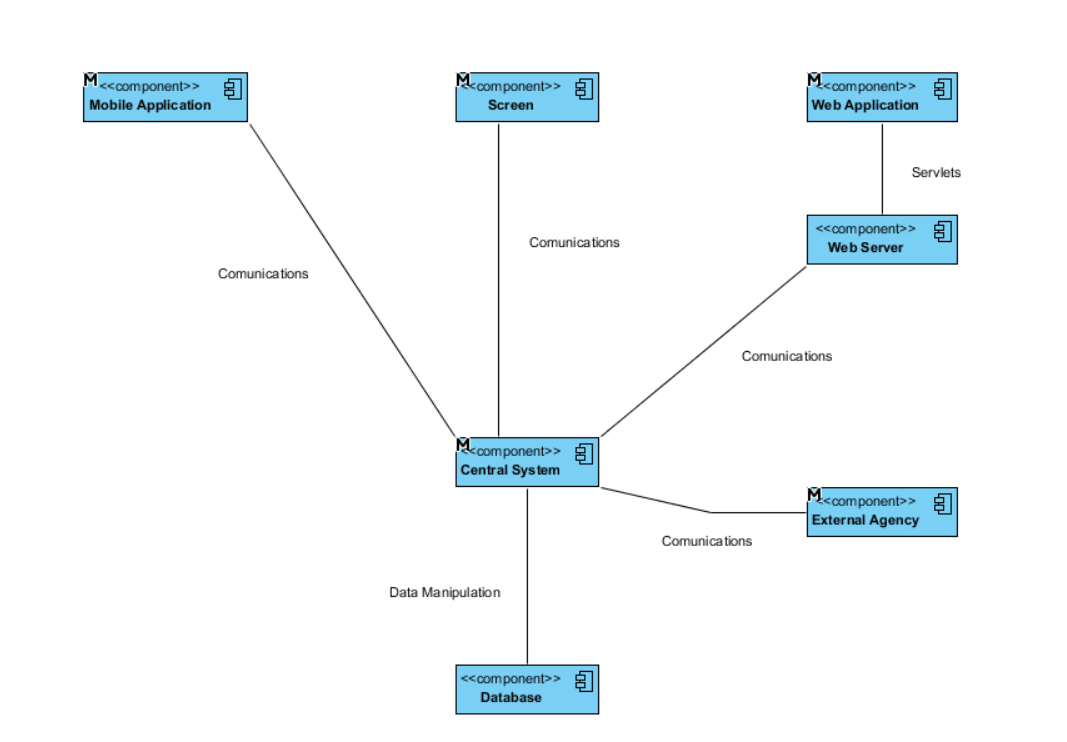
High Level Architecture and their interaction

The view show below is composed by:

* Database: an entity that represents the place where data is stored.
* Business Logic: or also called Central System, is an entity that represents the part of system where data is managed, moreover in this part, system receives/sends data from front-end application.

The interaction between Central System and other front-end application is made in a synchronous way for providing service about the ride.

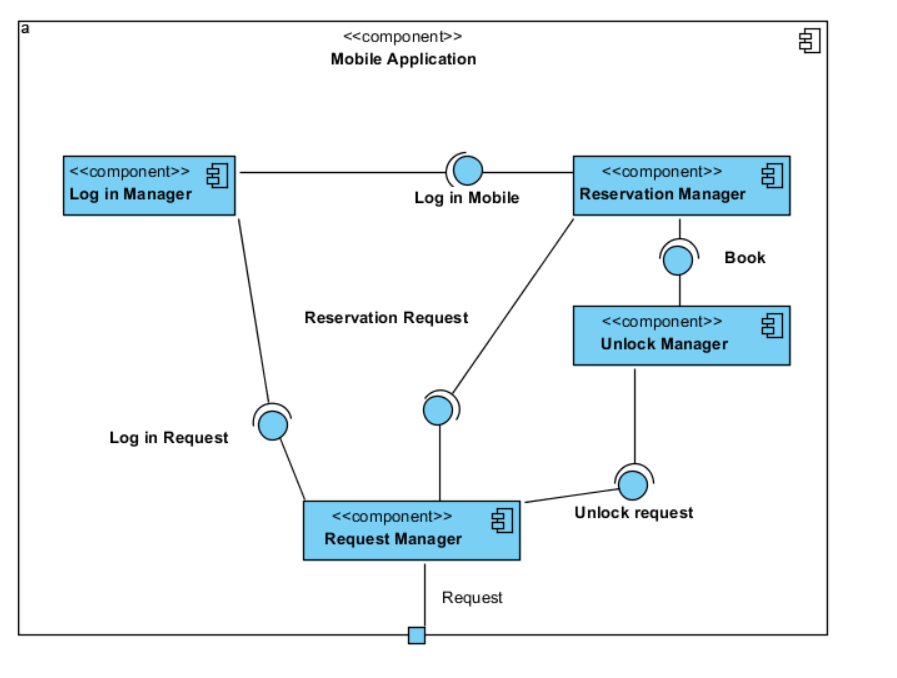
* Web Server: an entity that aims to create web pages dynamically.
* Web Application: front end application that provide the reservation, payment history, registration and log-in service.
* Mobile Application front end application that provide the reservation, unlocking and log-in car service.
* Car screen: front end application that permits to see the grid station positions and calculates the path from the nearest one.



The “M” indicate that components will be expanded.

Component Level Architecture and their interaction

Mobile Application



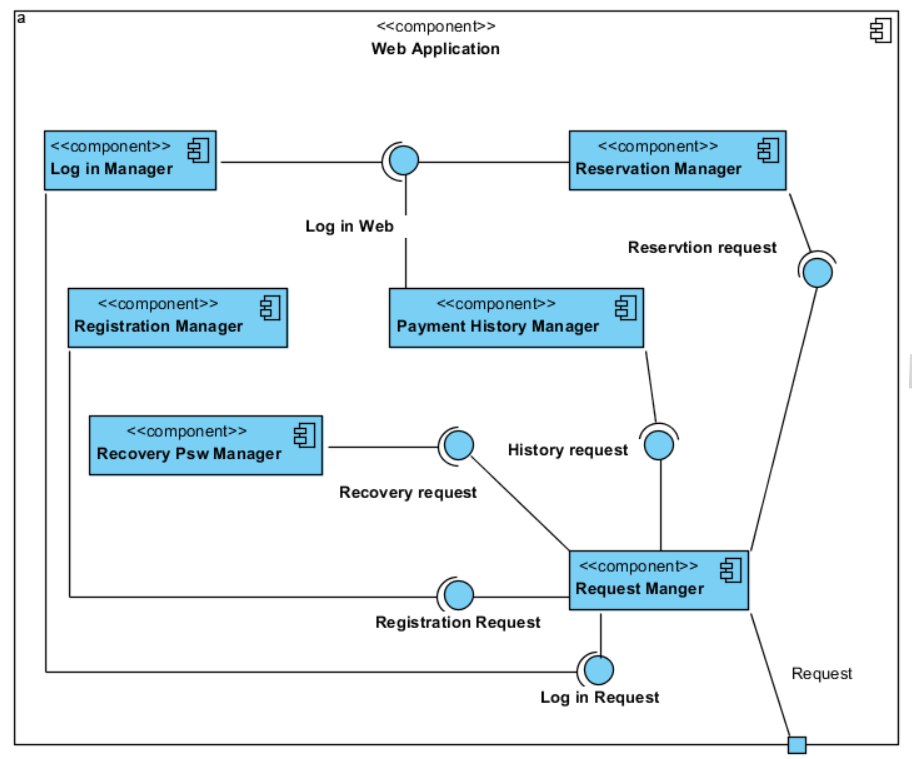
**Log in manager:** manage the log-in service, control syntactical requirements.

**Reservation Manager:** manage the reservation service, provides the map with the available car.

**Unlock Manager:** manage the unlock service, controls if exist a valid reservation.

**Request Manager:** manage the incoming request of service from the service manager and sends the request to the Central System.

Web Application



**Log-in manager:** manage the log-in service, control syntactical requirements.

**Reservation Manager:** manage the reservation service, provides the map with the available car.

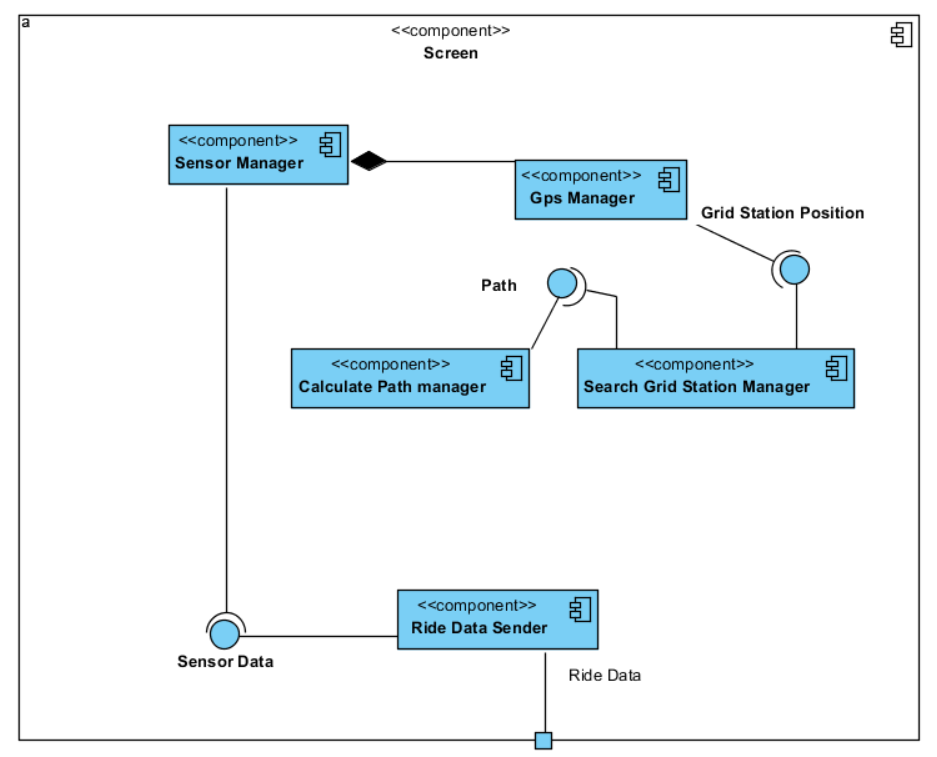
**Registration Manager:** manage the registration service, control syntactical requirements.

**Recovery Psw Manager:** manage the password recovery service, control syntactical requirements.

**Payment History Manager:** manage the payment history service, permits to show the reservation timer (and the unresolved payment).???

**Request Manager:** manage the incoming request of service from the service manager and sends the request to the web server.

Car Screen Application



**Sensor Manager:** manage all components inside the car, permits to capture passenger number, battery level and car position.

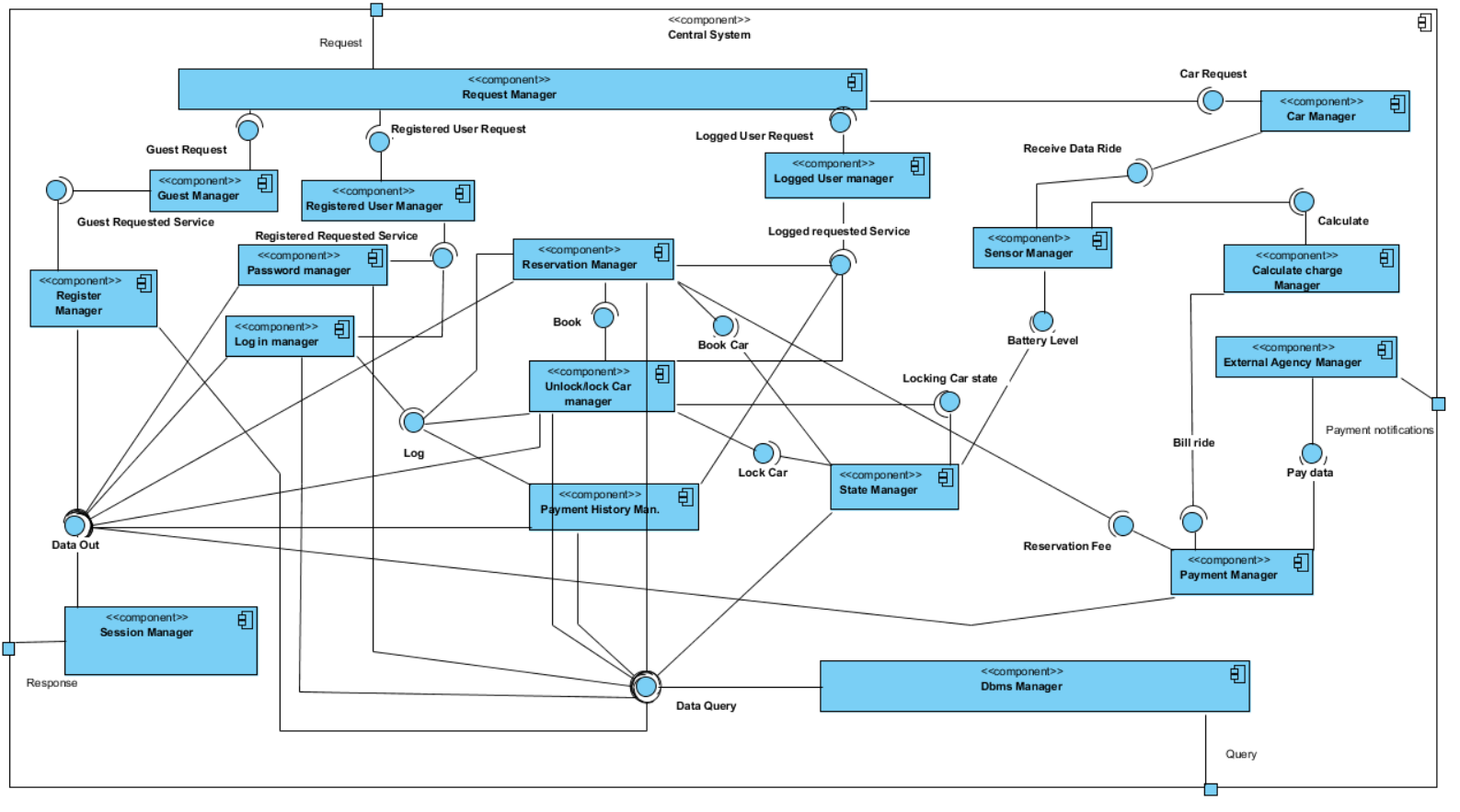
**Gps Manager:** manage all incoming data from gps and permits to show on the car screen.

**Search Grid Station manager:** manage the car gps data and permits to search the nearest grid station or another one specified by the user.

**Calculate Path manager:** permits to calculate and update on screen the distance from the car to the specific station.

**Ride Data Sender:** send all data ride to the Central System for calculating the charge ride and creates the bill ride.

Business Logic Application

****

**Request Manager:** manage the incoming request from the front-end applications, manages a buffer for these requests and recognizes the type of the user request, sends this request to specific managers.

**Guest Manager/Registered User Manager/Logged User Manager/Car Manager:** manage requests from the same type of user, manages a buffer for these requests and send these request to the specific manager that provides the requested service.

**Register Manager:** Provide registration service and check the requirements validity, this component manages the single user request (create a query) and sending to dbms manager for executing this one.

Then dbms response, the manager provides requirements control.

**Password Manager:** Provide password recovery service and check the requirements validity, this component manages the single user request (create a query) and sending to dbms manager for executing this one.

Then dbms response, the manager provides requirements control.

**Log in Manager:** Provide log in service and check the requirements validity, this component manages the single user request (create a query) and sending to dbms manager for executing this one.

**Reservation Manager:** Provide reservation service, calculate the reservation time and check the requirements validity, this component manages the single user request (create a query) and sending to dbms manager for executing this one.

Then dbms response, the manager provides requirements control.

**Unlock Car Manager:** Provide unlocking service and check the requirements validity, this component manages the single user request (create a query) and sending to dbms manager for executing this one.

Then dbms response, the manager provides requirements control.

**Payment History Manager:** Provide payment history service and check the requirements validity, this component manages the single user request (create a query) and sending to dbms manager for executing this one. (???incosistency)

Then dbms response, the manager provides requirements control.

**Sensor Manager:** Receives and manages all sensor data for permits other manager to use them.

**Calculate Charge Manager:** Manage all ride data provided from Sensor manager and creates the bill ride keep in mind the possible discounts or fee.

**Payment Manager:** Manage all bill ride incoming reservation manager and calculate charge manager, prepares the bill notification to send to external agency.

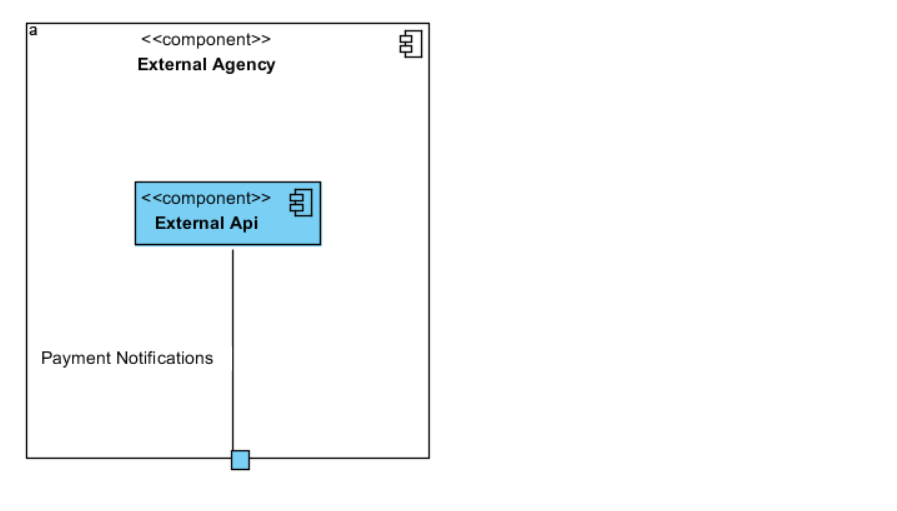
**External Agency Manager**: Manages the connection with External Agency Interface System and notified it with bill notifications.

**State Manager:** Manage the state of the car whenever occur.

**Dbms Manager:** Receives query from service manager and interrogate the database, sends the database response to the correspond service manager.

**Session Manager:** Manages all outgoing response request through received from the service managers an output buffer and send this one to the respective front-end applications.

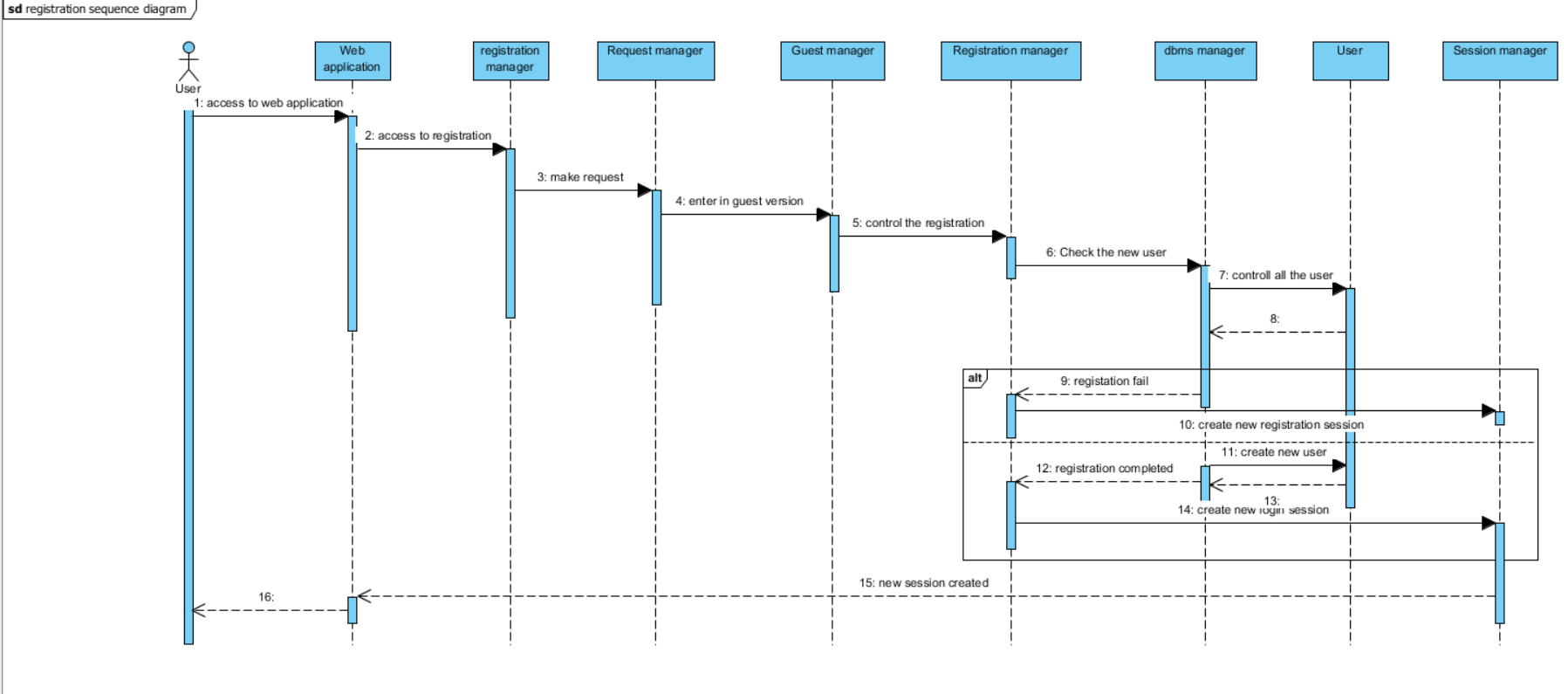
**External Agency**

****

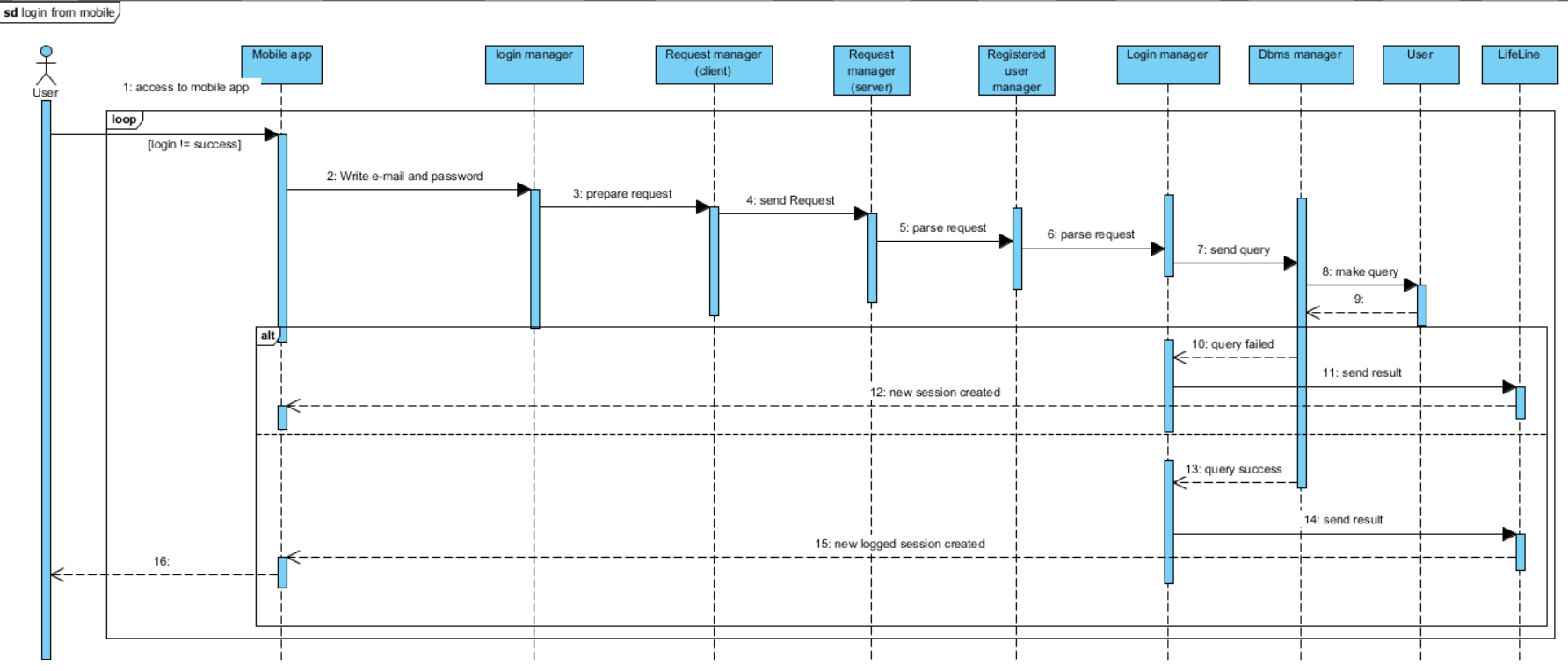
**External Api:** Manages the incoming notifications from Central System.

2.3 Runtime View

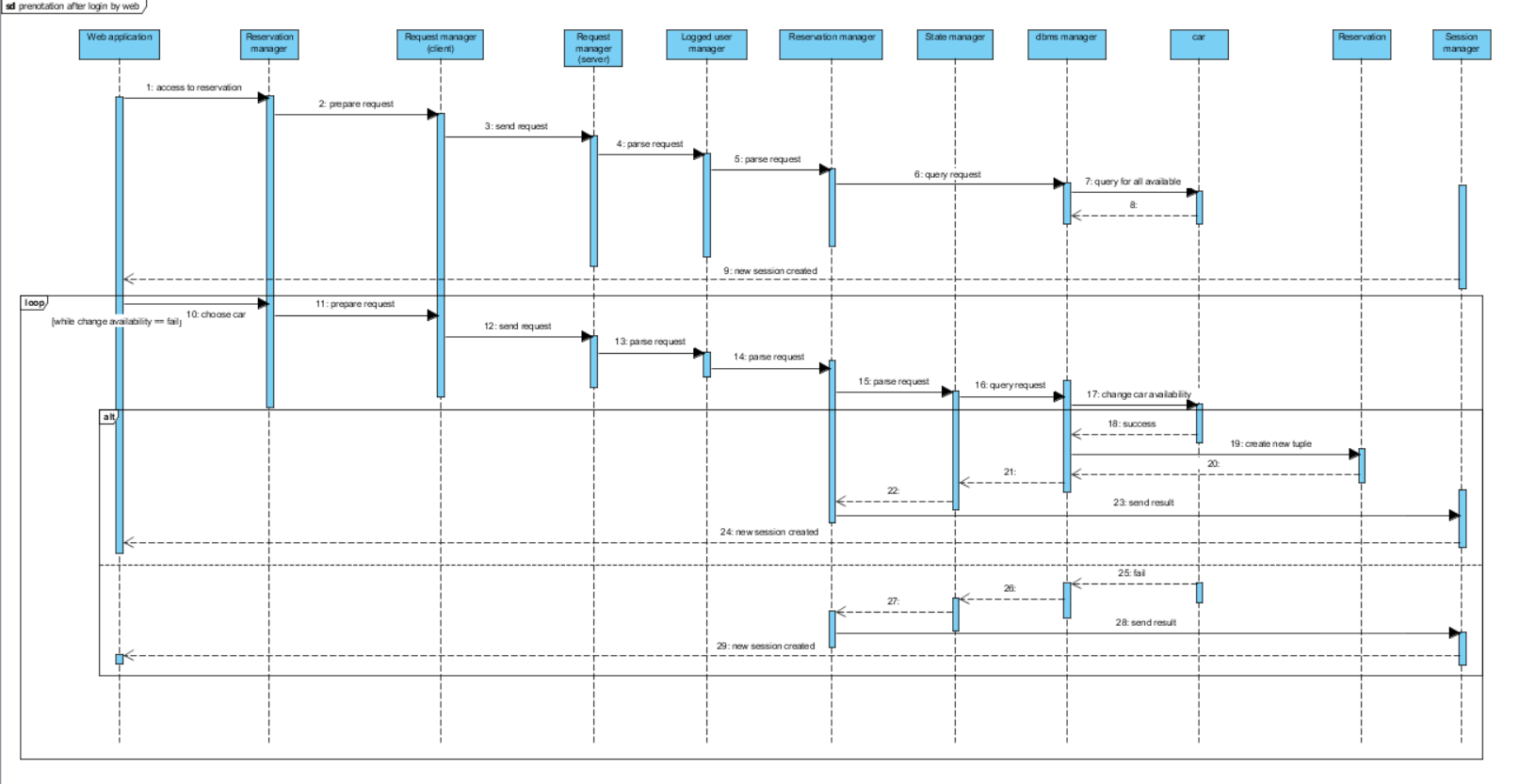
**Registration**

****

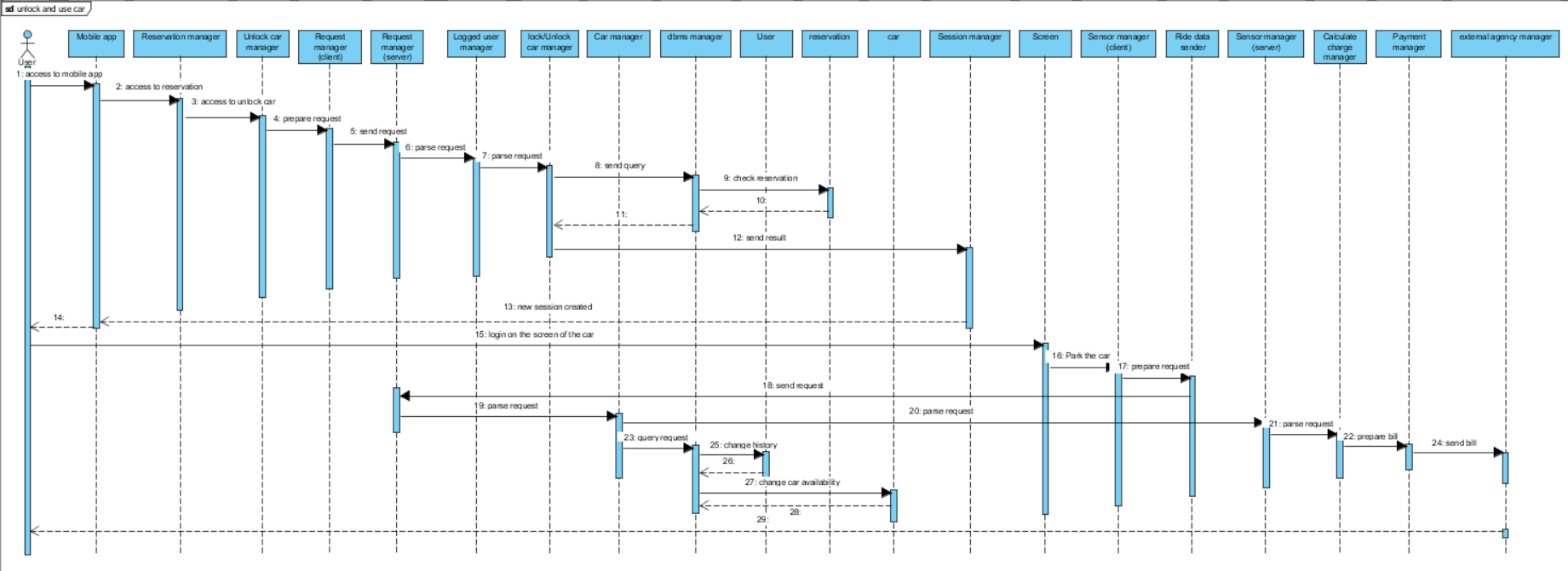
**Log in**

****

**Reservation**



**Unlock car**

****

2.4 Component Interfaces

In this part of the documents there are the descriptions of the main methods for each component described in the component view, for each part of the system code there are getter and setter method, but we choose to not show here.

Mobile application

**Login manager**: login (String email, String password)

**Reservation manager:** reserve (Car carToReserve, User userWhoReserve)

**Unlock manager:** unlockCar (Car carToUnlock, User userWhoWantToUnlock)

**Request manager:** sendRequest (Request request)

Web application

**Login manager:** login (String email, String password)

**Reservation manager:** reserve (Car carToReserve, User userWhoReserve)

**Registration manager:** register (String email, String name, String surname, String cityOfBirth, Date dateOfBirth, String address, Int phoneNumber, String personCode, Int bankCode)

**Payment history manager:** getPayment (), getReservation ()

**Recovery password manager:** recover (String personCode)

**Request manager:** sendRequest (Request request)

Screen

**Sensor manager:** getPowerGridSensor (), getPassengerSensor (), getBatterySensor (), getGPS (), getDistanceToNearestPowerGrid (), getCurrentCharge ()

**Gps manager:** getDistance ()

**Search power grid station manager:** getPowerGridStation ()

**Calculate path Manager:** getDistance (String addressStart, String addressArrive)

**Ride data sender:** sendData ()

Business logic application

**Request manager:** parseRequest (Request request)

**Guest manager:** parseRequest (Request request)

**Logger user manager:** parseRequest (Request request)

**Registration manager:** sendQuery (String email, String personCode)

sendPassword (String email, String newPassword)

**Session manager:** sendData (Data data, User user)

sendEmail (String email, String datas)

**Registered user manager:** parseRequest (Request request)

**Password manager:** sendQuery (String personCode)

sendPassword (String email, String password)

**Payment History manager:** sendQuery (User user)

**Login manager:** sendQuery (String email, String password)

enterMainMenu (User user)

**Reservation manager:** parseRequest (), timeFinish (),

makeReservation (User user, Car car)

**State manager:** makeCarAvailable (Car car), makeCarUnavailable (Car car)

controlCarState (Car car)

**Unlock/lock car manager:** lockCar (Car car, User user), unlockCar (Car car, User user)

**Car manager:** parseRequest (Request request)

**Sensor manager:** getPowerGridSensorData (), getPassengerSensorData (), getBatterySensorData (), getGpsData ()

**Calculate charge manager:** checkPossiblediscountorFee (SensorData discountType, SensorData gpsPosition)

calculateCharge (Int timeOfDrive, Bool powerGridSensor,

Bool passengerSensor, Bool batterySensor,

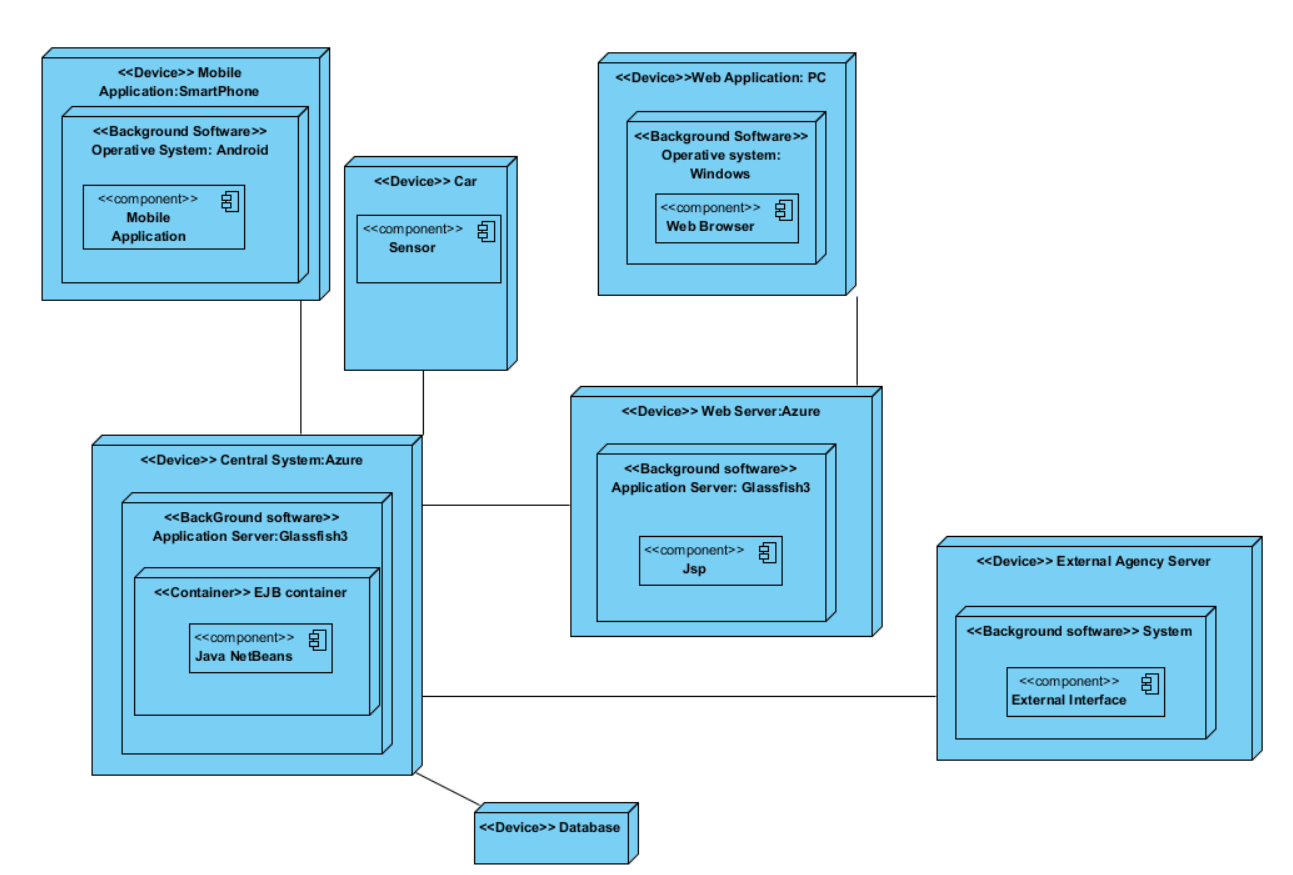
Bool tooMuchDistanceFromPowerGrid)

**Payment manager:** sendBill (Bill bill)

**External agency manager:** sendBillToExternalAgency (Bill bill)

**Dbms manager:** makeQuery (Query query), addTuple (), removeTuple (), changeTuple ()

2.5 Deployment Diagram



This deployment diagram shows the conceptual software architecture level; the first level is marked with <<device>> for indicate that this level is a hardware level.

In the car node we don’t insert any software level because we want underline the facts that the car system is the same of the central system.

The interactions among the nodes don’t keep count about the architecture level.

The specifications of the architectural and design choice will be explaining in the next chapters.

**2.6 Architectural styles and pattern**

All system is based on client server approach with thin clients (only front-end application) and fat server.

We choose this approach because this is the better one for providing different service at the same time.

Our architecture is a 4-tier architecture, from the classic 3-tier we adding a fourth level between logic and presentation: web tier.

This tier permits to create dynamic web pages, we have thought that is absolutely useful for the service provided via web application, e.g.: Available car map.

The whole system is built on an application server, Glassfish 3.

We choose Glassfish 3, instead Tomcat (it’s only a web container), because permits to reuse the code, provides communication controls and isolates the services, moreover, provides several non-functional requirements.

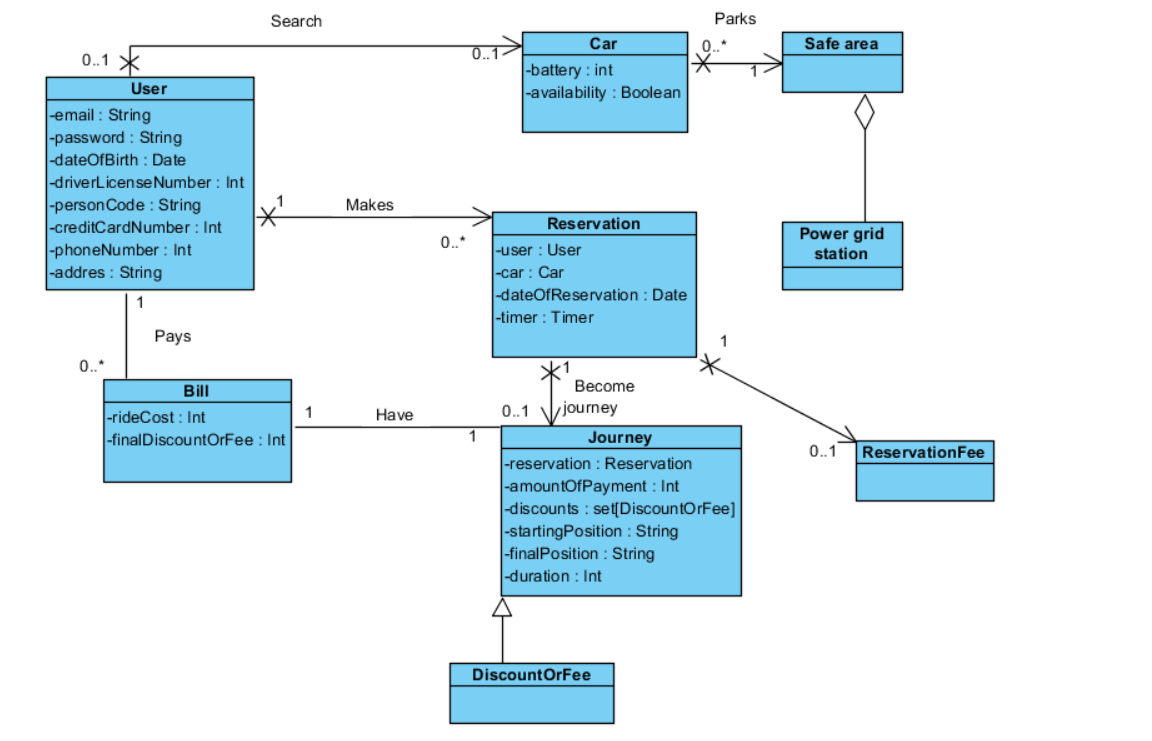
Glassfish permits improve service security and persistence.

**Design Pattern**

* **MVC:** this pattern covers all system architecture, the request is sent from front-end application (client tier), manages into business logic tier, this tier interrogates the database (database tier), and the response will be send to front-end application.
* **Client-Server:** as we already explain all system is based on this pattern, this pattern is formed by request (client) and response (server) in a synchronous way.
* **Adapter:** this pattern will be use into Request manager to receive and manage all request from different type of front-end applications.
* **Observer:** this pattern will be use into State Manager (observer) and Sensor Manager, Reservation Manager (observable); we choose to use this pattern because we have thought there could be possible system implementation in the future and the state car could be change in more way.

1. Algorithm Design

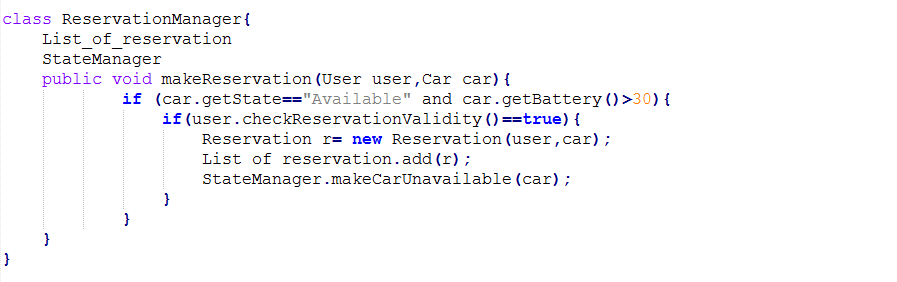
For explaining better the code designs we re-insert the class diagram, for understanding where the explained functions are situated:



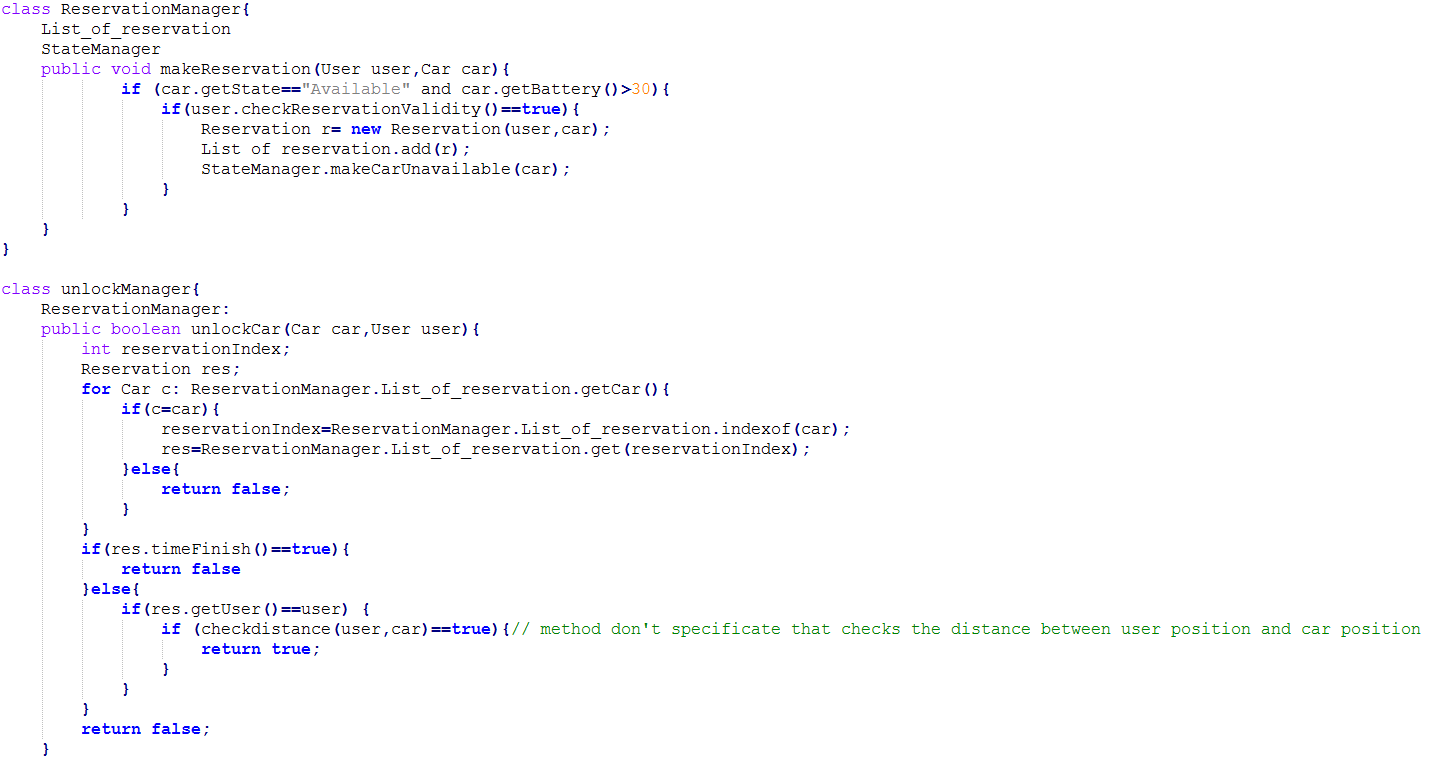
**Create a bill for the ride**



**Create a reservation**

****

**Unlock Car**

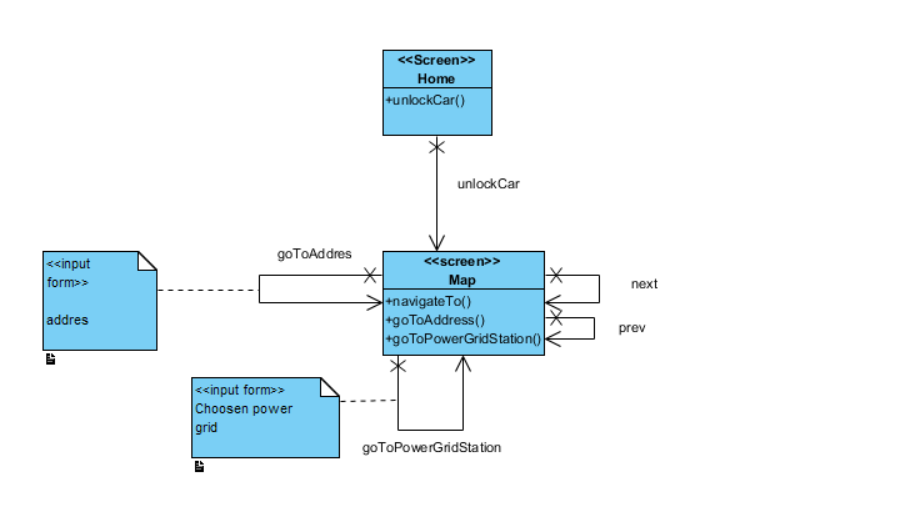
****

1. **User Interface Design**

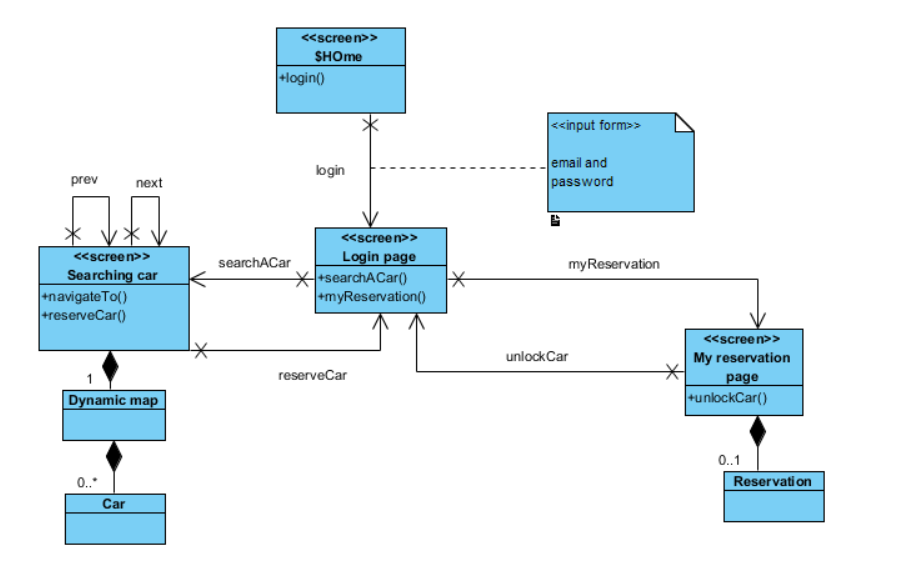
**Ux diagrams**

We use this diagrams to explain how the user performs main actions:

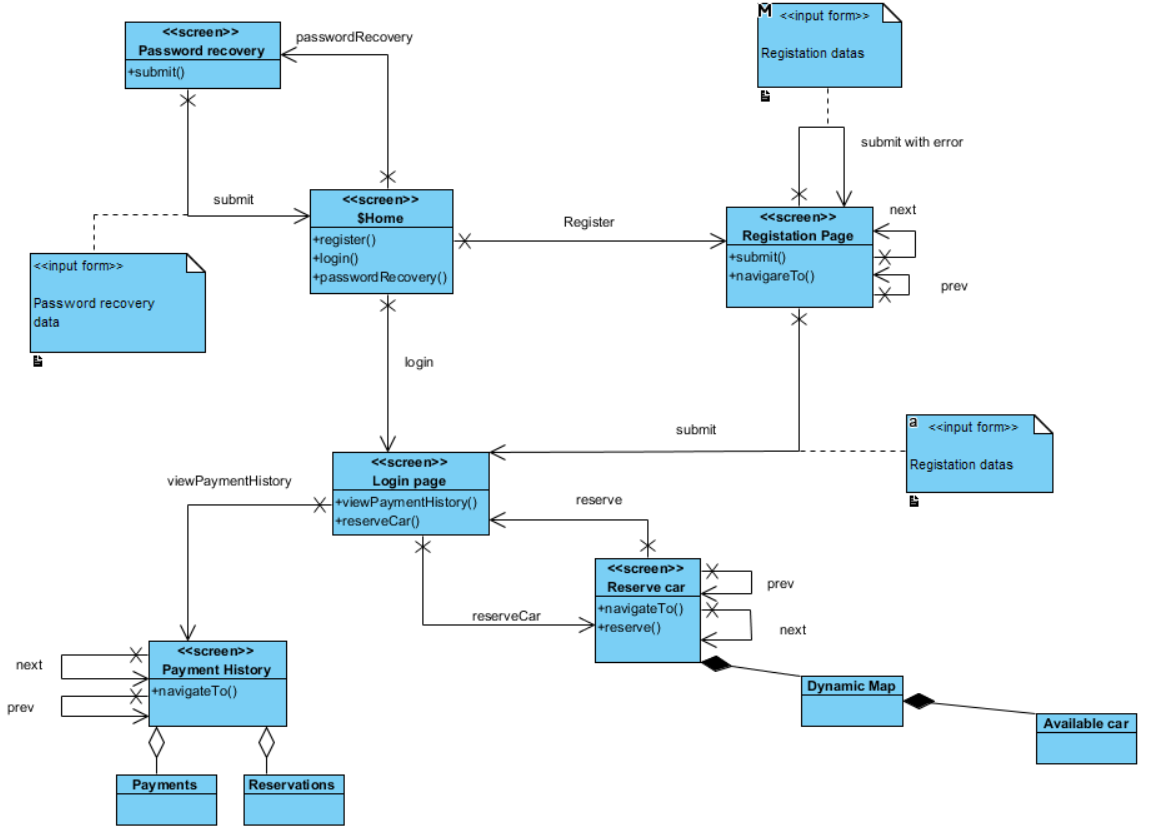
* **Car**



* **Mobile Application**

****

* **Web Application**

****

1. **Requirements Traceability**

**[G1]: Allow to register on the system**

* Registration manager
* Request manager(Client)
* Request manager(Server)
* Guest manager
* Register manager
* Dbms manager
* Session Manager

**[G2]: Allow to log in on the system**

* Login manager (Web or mobile)
* Request manager(Client)
* Request manager(Server)
* Registered user manager
* Login manager
* Dbms manager
* Session Manager

**[G3]: Allow to find the locations of available cars within a certain distance from their current position or from a specified address**

* Reservation manager (Web or mobile)
* Request manager(Client)
* Request manager(Server)
* Logged user manager
* Reservation manager
* State manager
* Dbms manager
* Session manager

**[G4]: Allow to reserve a single car for up to 1 hour**

* Request manager(Server)
* Logged user manager
* Reservation manager
* Dbms manager
* Session Manager

**[G5]: Allow to pay a fee if he misses the reservation**

* Reservation manager
* Payment manager
* External agency manager
* Dbms manager
* Session Manager

**[G6]: Allow to tell the system she’s nearby**

* Reservation manager (only mobile application)
* Unlock car manager

**[G7]: Allow to see the current charges**

* Sensor manager(screen)

**[G8]: Allow to reach grid stations location**

* Sensor manager(screen)
* Search grid station manager
* Calculate path manager

**[G9]: Allow to receive a discount if the user has at least two passengers**

* Sensor manager(screen)
* Ride data sender
* Request manager(server)
* Car manager
* Sensor manager(server)
* Calculate manager
* Payment manager
* Session Manager

**[G10]: Allow to receive a discount if the user left the car with no more than 50% battery empty**

* Sensor manager(screen)
* Ride data sender
* Request manager(server)
* Car manager
* Sensor manager(server)
* Calculate manager
* Payment manager
* Session Manager

**[G11]: Allow to receive a discount if the user left the car in a Power grid station and takes care of plugging the car**

* Sensor manager(screen)
* Ride data sender
* Request manager(server)
* Car manager
* Sensor manager(server)
* Calculate manager
* Payment manager
* Session Manager

**[G12]: Allow to receive a fee if the user left the car more than 3km from the nearest power station or left the car with more than 80% of the battery empty**

* Sensor manager(screen)
* Ride data sender
* Request manager(server)
* Car manager
* Sensor manager(server)
* Calculate manager
* Payment manager
* Session Manager

**[G13]: Allow to see an e-mail sent by the system with the bill ride and payment resume, at the end of this one**

* Payment manager
* Session manager

**[G14]: Allow to user to pay immediately after at the end of the ride**

* Payment manager
* External agency manager

1. **Used Tools**

**Visual Paradigm:** creates diagrams

**Word:** redacts the document

**Paint:** fix diagram images

**Git-Hub:** publication and transmission

1. Hours of work

Marco Wenzel:25h

Francesco Tinarelli:25h