Design Document

POLITECNICO
MILANO 1863

Software Engineering 2
PowerEnJoy

Authors:

Bianchi Alessandro (Mat. 875035)
Canciani Francesco (Mat. 807056)
Cannone Lorenzo (Mat. 875802)

Document version: 1.1

Academic year:
2016-2017
Release date:
11-12-2016

Contents

1.Introduction

1.1 Purpose

This design document describes the architecture and system design of PowerEnJoy, a digital management system for a car-sharing service that exclusively employs electric cars, which is the result of a project included in the Software Engineering 2 course at "Politecnico di Milano".

The document aims at going into further details with everything that relates to architectural decisions and tradeoffs, made during the design process, by providing several examples and models to justify all our choices to the key design stakeholders, which are the intended audience of this document.

1.2 Scope

The system, named PowerEnJoy, is a software that will run both as a mobile and a web application, that mainly targets Milan's citizens, which are the intended audience of our software-to-be.

The system will provide the functionality normally provided by car-sharing services, that include:

- Allow users to find the locations of available cars from a given address or their current location
- Allow users to reserve the car that they will eventually end up driving
- Allow users to open the car that they reserved through our application, without the need of an external card or additional key
- Allow users to get discounts on their rides if they demonstrate virtuous behaviors with respect to the environment

This new system will be developed to be more efficient and reliable than the existing one, by providing an easier usability experience to the clients and at the same time guaranteeing to the stakeholders a profitable use of the company's resources.

1.3 Definitions, acronyms, abbreviations

- RASD: Requirements Analysis and Specification Document
- DD: Design Document
- SMS: Short Message Service. It is a short text notification sent to a mobile phone. An SMS Gateway is needed to benefit of this service.
- SMS Gateway: External service that PowerEnJoy's system uses to send or receive SMS.
- API: Application Programming Interface. It is a set of subroutine definitions, protocols, and tools used to interface with another system.
- MVC: Model View Controller. It is a software design pattern for implementing user interfaces.
- UX: User Experience
- BCE: Boundary-Control-Entity
- PGS: Power Grid Station
- MSO: Money Saving Option
- SQL: Structured Query Language
- Thin client: lightweight computer that is purpose-built for remote access to a server. It depends heavily on another computer (its server) to fulfill its computational roles.
- RDBMS: Relational Database Management System
- Ionic: a complete open-source SDK for hybrid mobile app development.
- SDK: Software Development Kit
- WildFly: is an application server written in Java. It implements the Java Platform, Enterprise Edition (Java EE) specification. It runs on multiple platforms.

1.4 Reference documents

- Requirements Analysis and Specification Document, previously released
- Specification Document: Assignments AA 2016-2017.pdf
- Example document:
  - Sample Design Deliverable Discussed on Nov. 2.pdf

1.5 Document structure

The document is organized as follows:

1. **Introduction**: provides a synopsis of the document. It identifies its purpose by giving a brief overview of the document and its organization, also by listing the definition of all terms, acronyms and abbreviations that might exist, to properly interpret every aspect of the document.

2. **Architectural Design**: specifies the general architecture of the system. It is divided into several sub-sections:

   2.1. **Overview**: presents the multi-tier architecture of our system;

   2.2. **High level components and their interaction**: provides a global perspective of the system's components and their interaction within each other;

   2.3. **Component view**: displays a precise view of the system's components;

   2.4. **Deploying view**: shows the physical deployment of the system's components in order to allow the application to perform properly;

   2.5. **Runtime view**: presents various UML sequence diagrams with the purpose of describing the way components interact to accomplish specific tasks within our system;

   2.6. **Component interfaces**: provides a diagram of the system's interfaces and the connections between each other;

   2.7. **Selected architectural styles and patterns**: offers an explanation to the architectural decisions that were made along the system's modeling process;

   2.8. **Other design decisions**: adds further information about the choices that were made in the definition of the system's design.

3. **Algorithm Design**: focuses on the definition of the most relevant algorithmic part. Pseudocode is used in order to hide unnecessary implementation details.

4. **User Interface Design**: provides an overview of how the system's user interfaces are going to look like. For this presentation, mockups, UX and BCE diagrams are used.

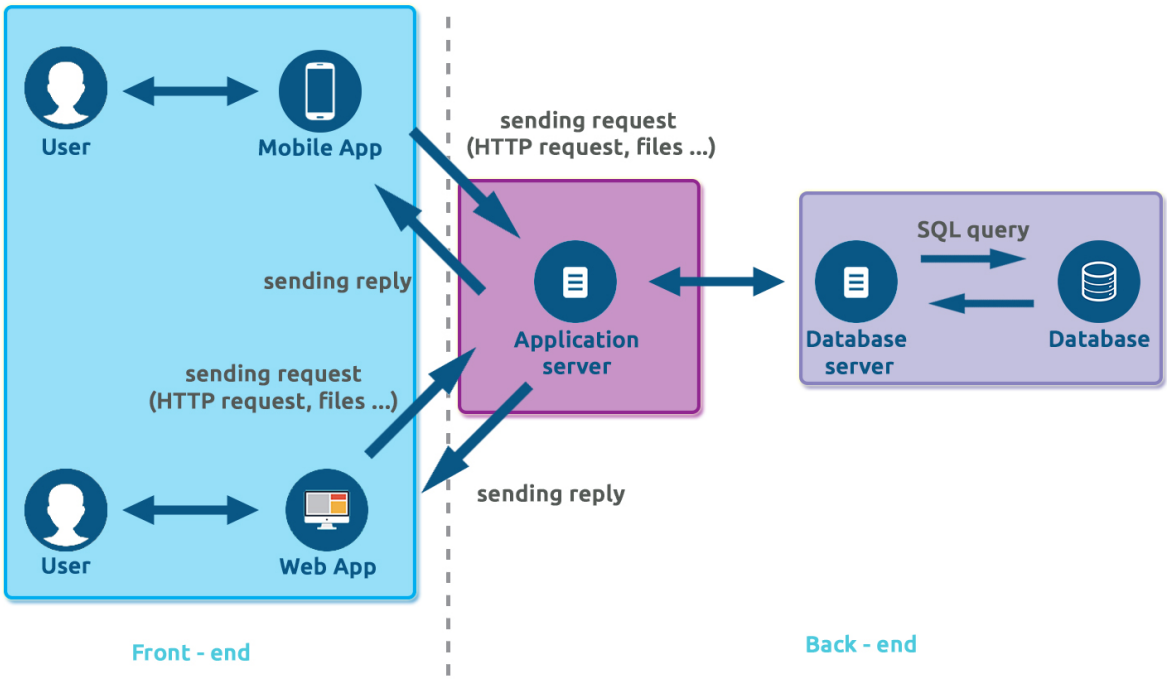
5. **Requirements Traceability**: illustrates how the requirements defined in the 'Requirements Analysis and Specification Document' map to the design elements that are delineated inside this design document.

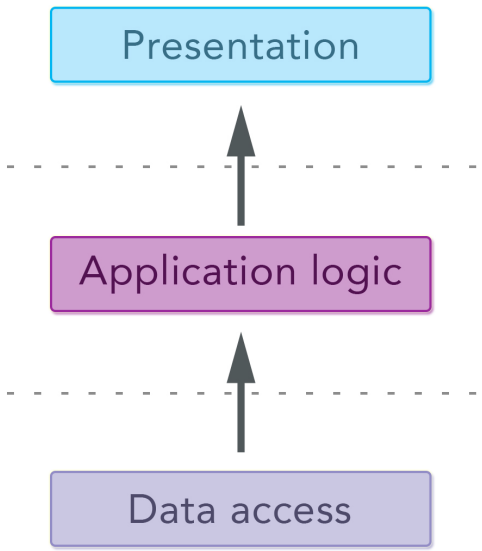
6. **Effort spent**: includes a list of the number of hours each group member has worked towards the fulfillment of the specified assignment.

7. **References**: lists a series of tools that were used to produce this document.

2.Architectural Design

2.1 Overview





PowerEnJoy has a three-tier architecture.

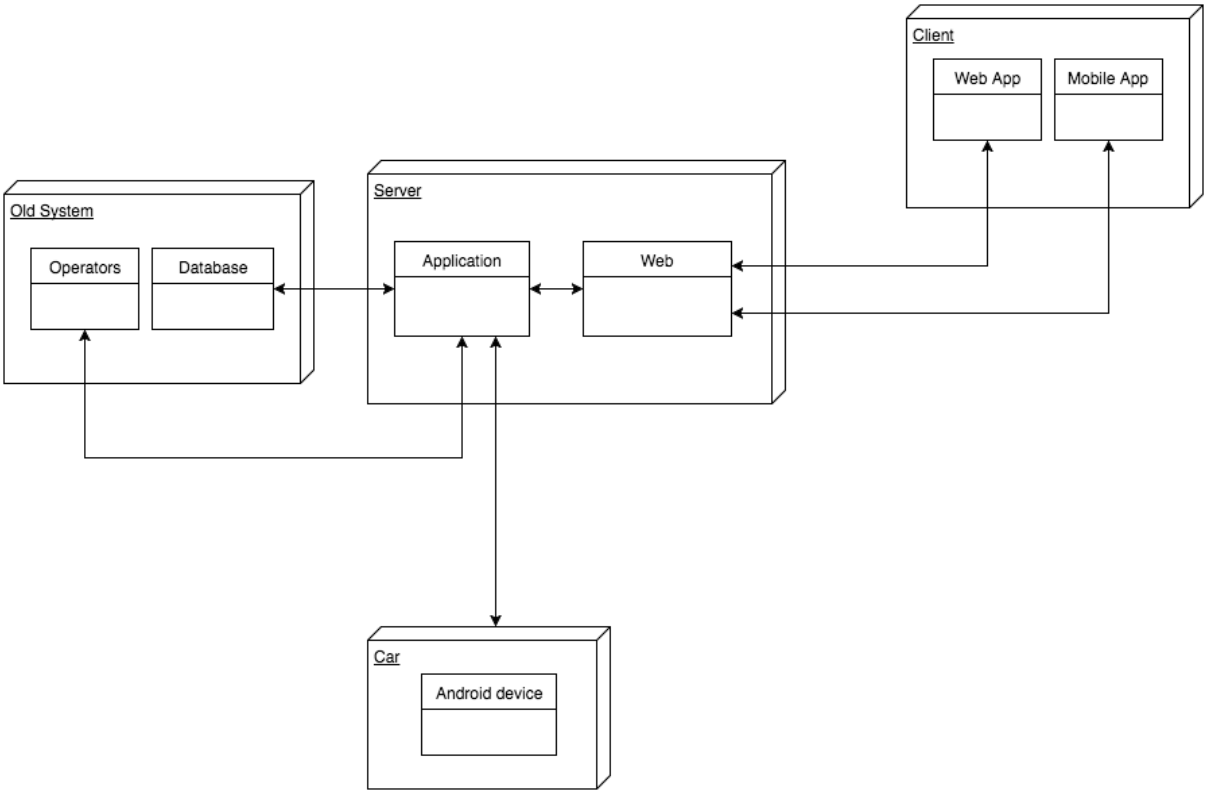
[Front-end level] The client could choose to use the mobile or desktop interface with a dynamic GUI.

[Back-end level] The application server answers to the client's requests and interacts with the database server, which sends SQL queries to the database.

By providing this configuration, we refer to the client as being a thin client, because the servers take care of all the major computational aspects. This configuration, also allows the user to access more securely to the server since all the clients only communicate with the application server and not with any other client using PowerEnJoy's application.

Due to its design, especially for the presence of a client-server architecture based on thin clients that makes requests to a server, this application could someday be deployed to one of the major cloud-computing services available on the market, ensuring an incredible reliability and efficiency in data management and recovery.

2.2 High level components and they interactions



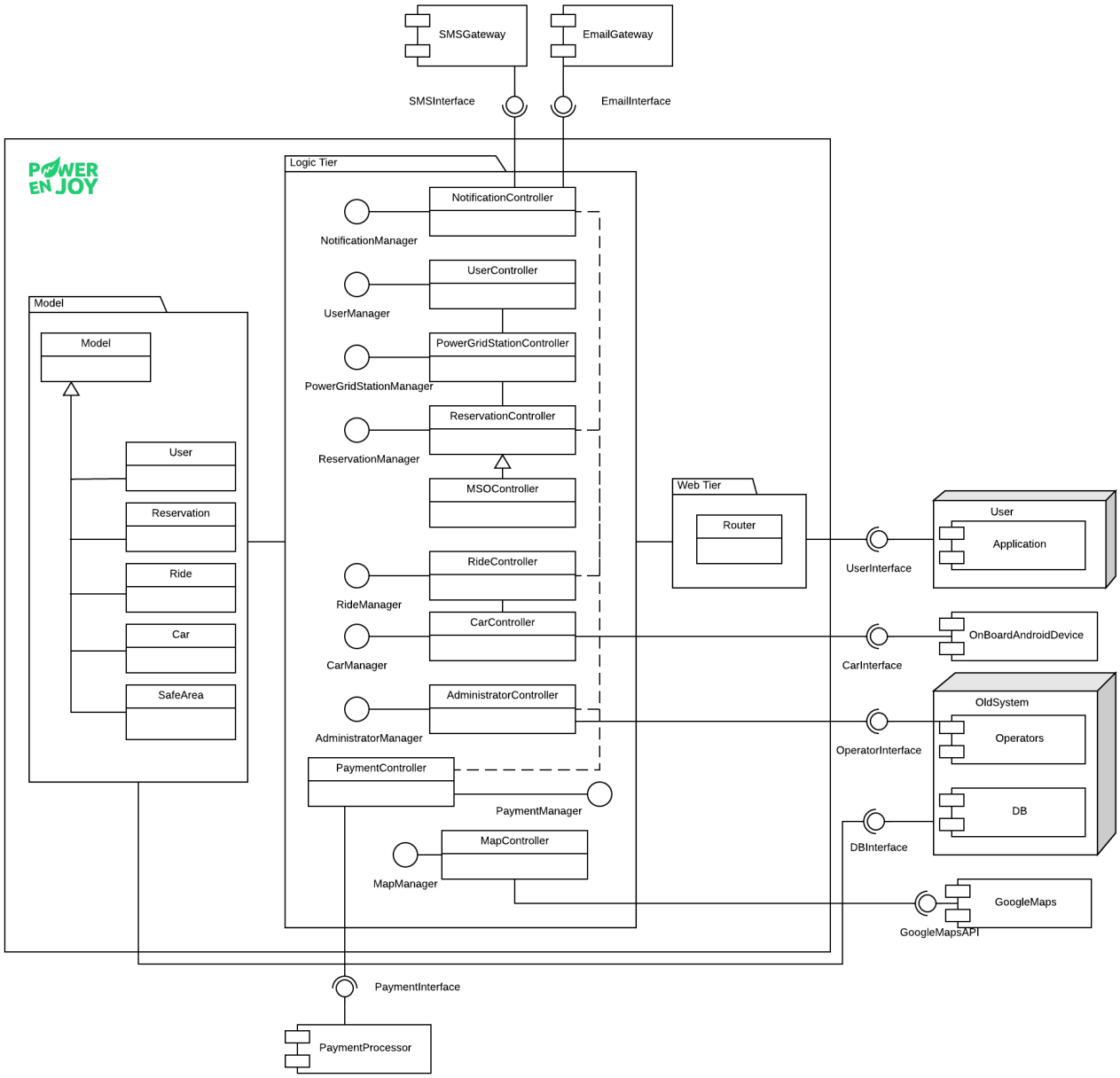
The high level components architecture presents four different major components:

- Server: is the main element of the architecture and only one instance of it exists. It receives requests from several clients and can handle them all at the same time. It is divided into two sub-components:
  - Web: allows communication between server and clients, also taking care of providing a dynamic interface to the clients
  - Application: runs the majority of the computation and elaboration of data required by the system. It interacts with the database and the operators section in the old system, but also with the android device placed inside the vehicles, in order to provide commands to the cars when needed.

(The web server was not included in the previous view because the application server could include a module that acts as a web server but we decided to explicitly state it here in order to have a more clear understanding of the interactions between the server and the clients.)

- Client: can communicate with the server from the mobile application or the web app. It has also the role of initiating the conversation with the server.
- Old System: it is included in the architecture because of its two sub-components:
  - Database: stores persistent data that are needed to run the system efficiently
  - Operators: has the purpose of interacting with the application server (precisely the administration section) in order to manage the assistance of PowerEnJoy's vehicles.
- Car: an Android device is placed inside each vehicle. Its purpose is to control the actuators of the car and acquire information from the sensors, in order to perform all the operations that the application server manages remotely.

2.3 Component view



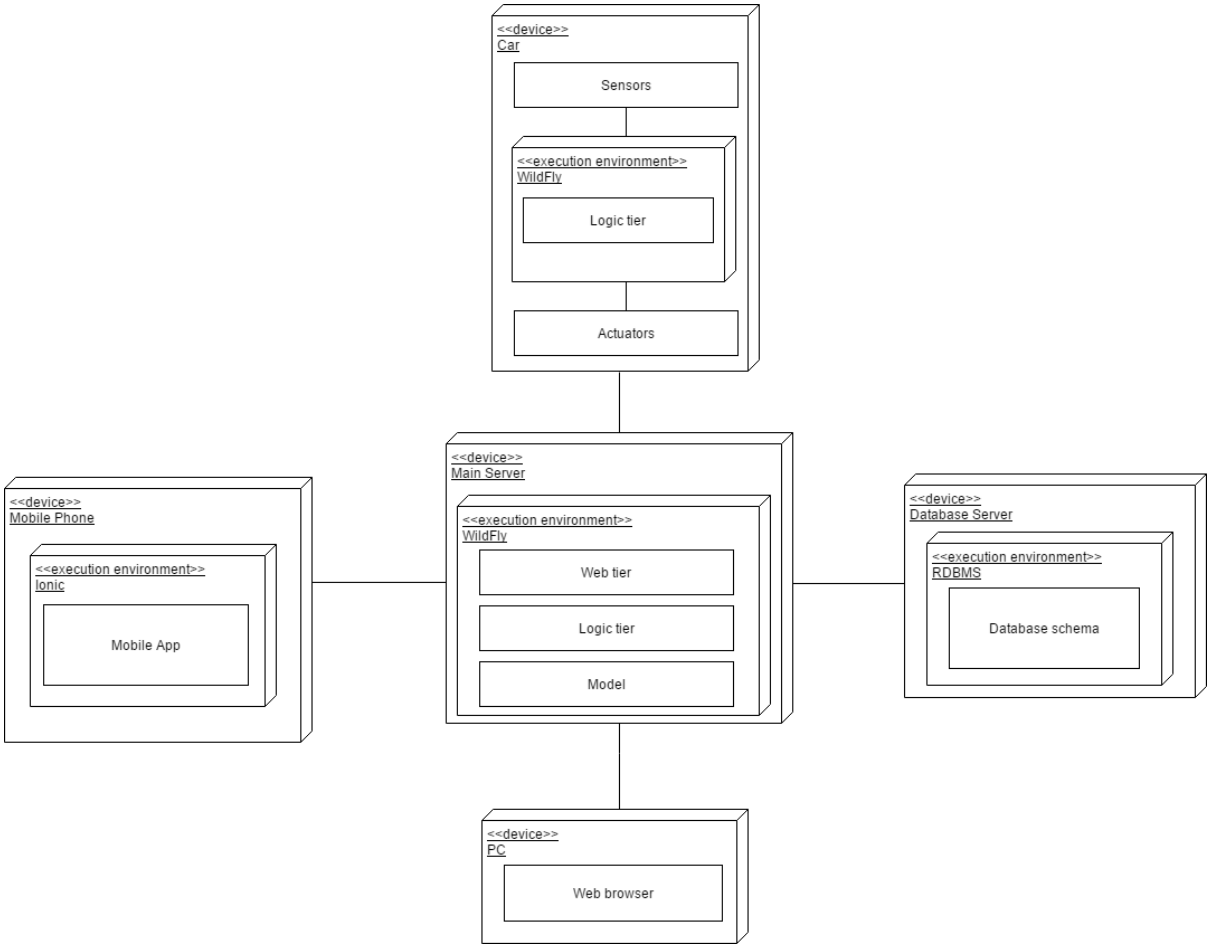
This diagram provides a general idea of all the components that are inside PowerEnJoy's system and their interaction with all the third-party components that live outside PowerEnJoy's system but that are needed in order to run the application correctly. The system interacts with them through specific interfaces that are shown in the picture.

Following, there is a list of all the components presented in the diagram, with a very brief definition of them:

- SMSGateway: Third-party gateway to allow the sending and the receiving of SMS
- EmailGateway: Third-party gateway to allow the sending and the receiving of emails
- NotificationController: Controller to manage notifications to the user
- User-->Application: PowerEnjoy's application in the user's device (mobile/web app)
- Router: Routes requests from the user's application to the related controller
- UserController: Controller to manage user related activities (e.g. login, signup)
- RideController: Controller to manage ride related activities (e.g. duration, timeout)
- PowerGridStationController: Controller to manage PGS related activities (e.g. available parking spots, PGS locations)
- ReservationController: Controller to manage reservation related activities (e.g. reserving a car, money saving option)
- MSOController: Controller to manage money saving option related activities (e.g. computing the best power grid station for the money saving reservation)
- CarController: Controller to manage car related activities (e.g. commands to send to the android device placed inside each car)
- AdministratorController: Controller to manage admin related activities (e.g. block/unblock users, communication with old system's operators section)
- PaymentController: Controller to manage payment related activities (e.g. ride discounts, commands to the third-party payment processor)
- MapController: Controller to manage map related activities (e.g. interfacing with Google Maps)
- OnBoardAndroidDevice: Android device, placed inside each vehicle, that controls the actuators of the car and acquires information from the sensors to control it remotely.

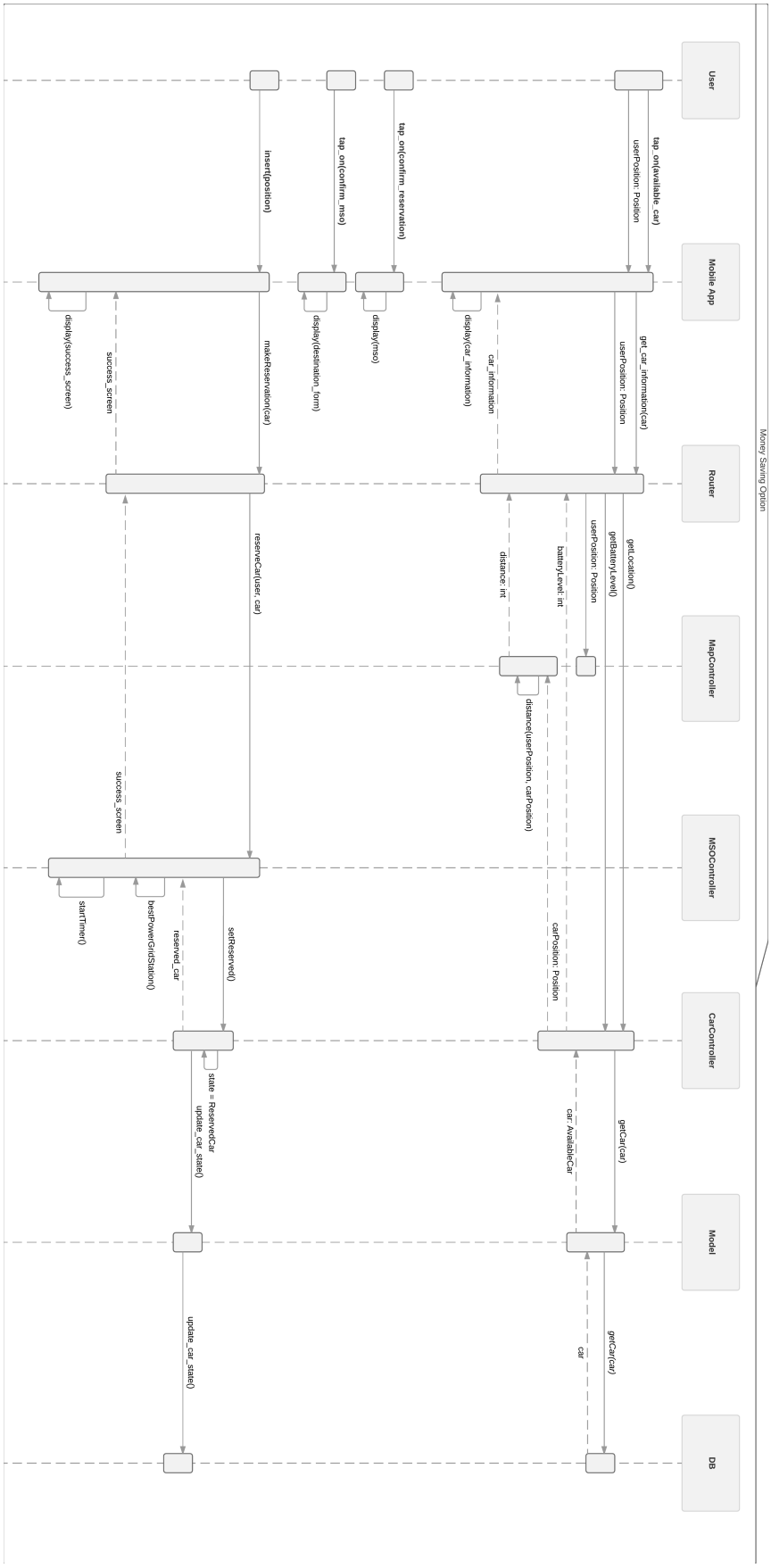
- OldSystem-->Operators: Old system's functionality that manages operators for the assistance on cars that need it. It interfaces with PowerEnJoy's administration.
- PaymentProcessor: Third-party payment processor that handles all the transactions from the users' payment channels.
- GoogleMaps: Third-party map service used to perform map related activities (e.g. computing distances, location objects on a map)
- DB: Database used to store persistent data.
- Model: Representation of the modeled world (i.e. previously defined class diagram)

2.4 Deployment view



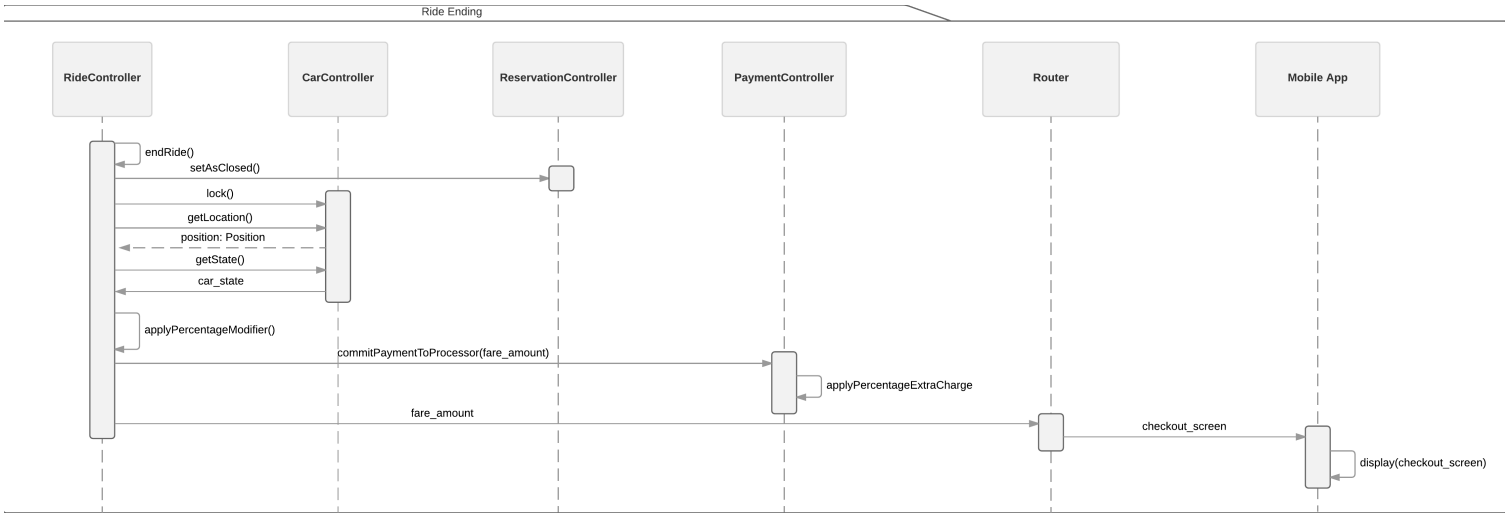
2.5 Runtime view

2.5.1 Money Saving Option



This sequence diagram represents the reservation activity. Furthermore, it is focused on those reservations whose Money Saving Option is enabled. Once the user has selected and reserved the car among the available ones, they are asked to provide the system their ride destination in order to be notified about the best Power Grid Station in which to leave the car parked. Each message forwarded to the DB represents a SQL query. Each of them, is denoted by italicized text.

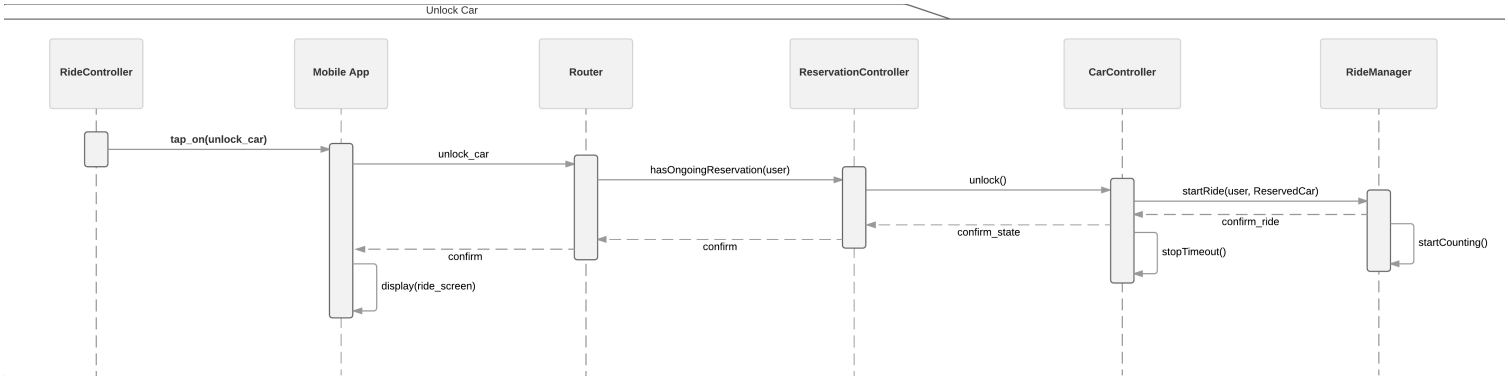
2.5.2 Ride ending



This diagram shows the management of an ended ride.

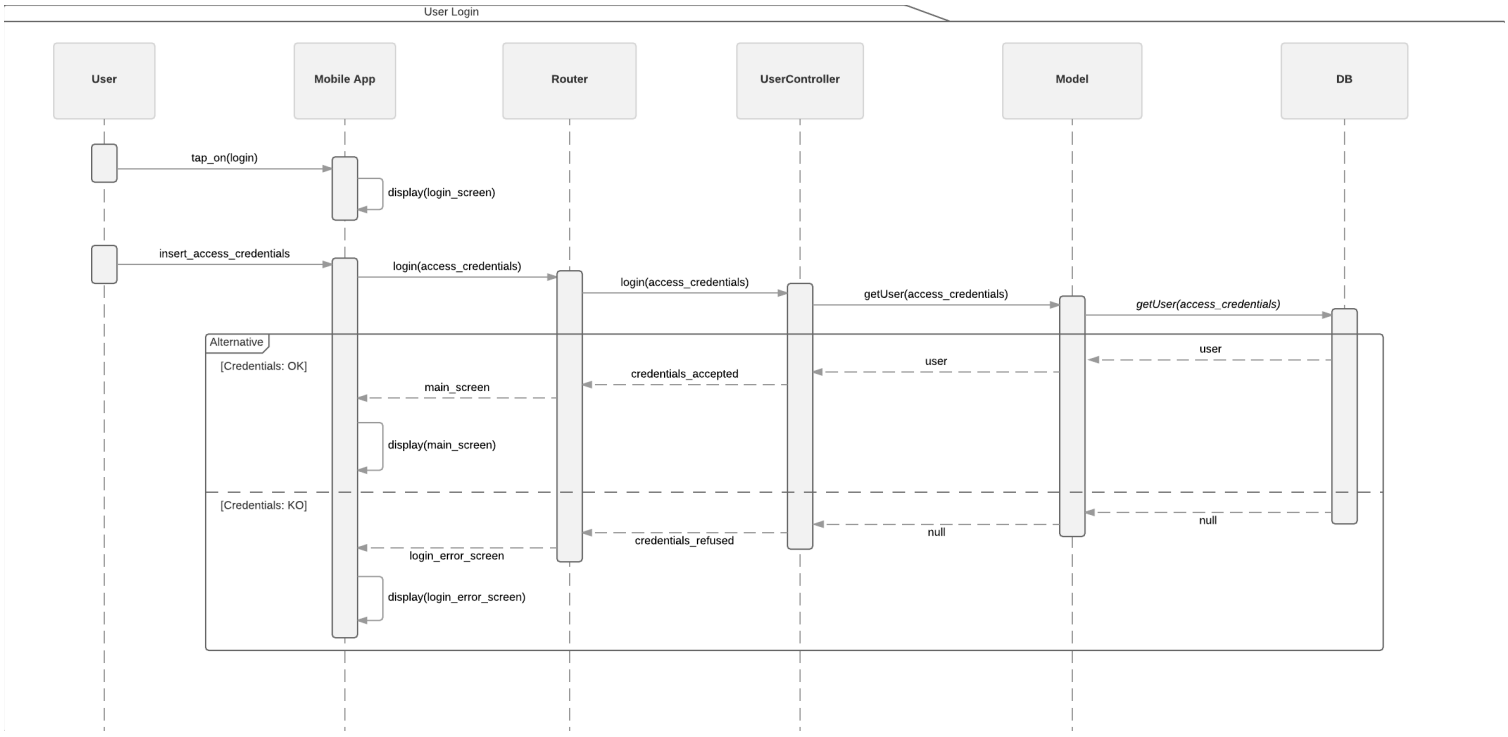
The cost modifiers are calculated and forwarded to the payment manager which ensures that every fee will be applied. The corresponding reservation for the ride gets closed and the user is notified about their checkout.

2.5.3 Unlock car



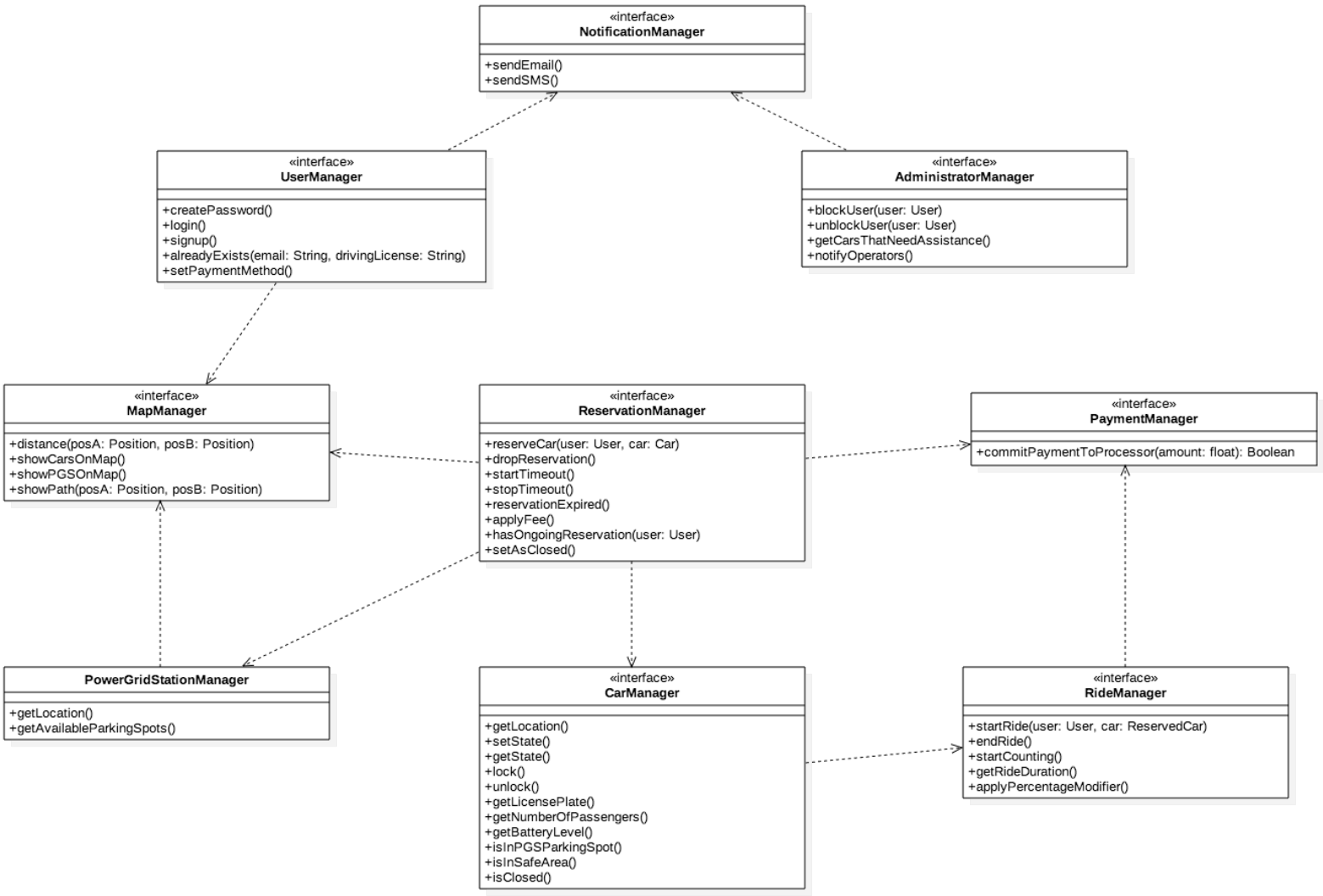
The car unlocking sequence diagram is self explanatory. The user which is close to the car can unlock it by selecting the option from their smartphone.

2.5.4 User Login



This is the login sequence diagram. Once the user provides their credentials to the system, a query to the database is made to look for the corresponding record. The database returns the user object to the User manager.

2.6 Component interfaces



The interfaces presented here are the ones that were presented in the component view chapter. They list some of the methods that need to be implemented in the respective controller.

2.7 Selected architectural styles and patterns

2.7.1 Overall architecture

Our architecture is divided into three tiers:

1. Thin Client
2. Application logic
3. Data access

A three-tier architecture is a client-server architecture in which each tier is developed and maintained as an independent module on a separated platform.

2.7.2 Design patterns

MVC Model-View-Controller has been inserted into our system in order to successfully relate the user interface to underlying data model. It is a widely used pattern in software development that allows flexibility inside our system, by separating concerns into three divided components. Being able to isolate these three components makes the code easier to reuse and independently test, which makes it a good reason for us to put it in the design of PowerEnJoy's system.

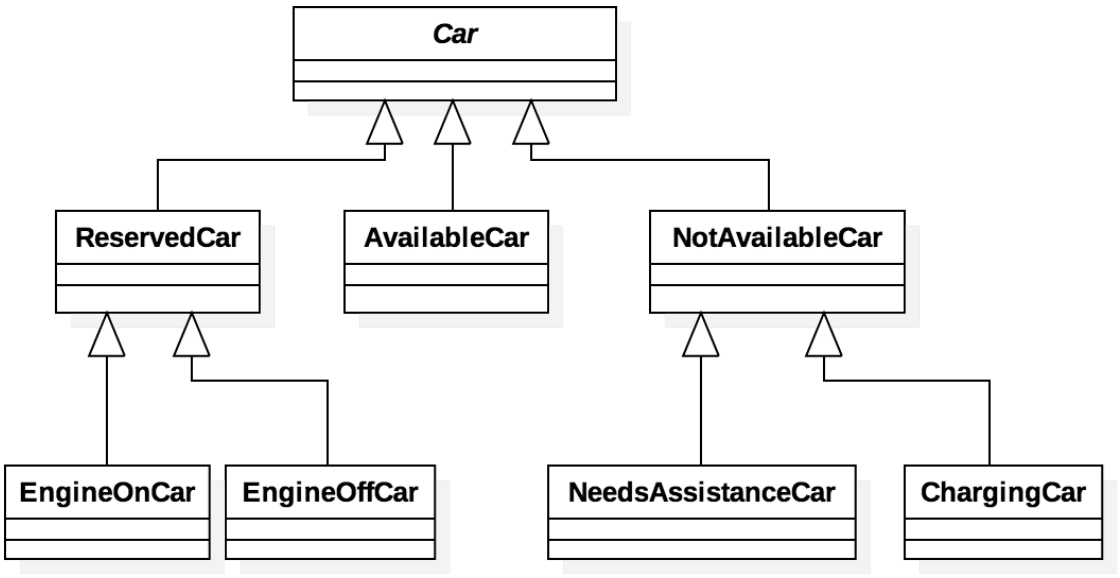
Client-Server Our system strongly relies on a Client-Server model. The clients are PowerEnJoy customers' devices, which can be the user's' mobile application or the user's web browser. The clients are thin clients, because they depend heavily on PowerEnJoy's server to fulfill their computational roles. Due to this fact, the application can also run on low-resources devices.

This modeling approach has been chosen for several reasons:

- Centralization: there is a centralized control, which makes a lot easier managing data, since all the files are stored in the same place.
- Scalability: any element, can be upgraded when needed. The server can also handle a growth in the number of clients that connect to it.
- Accessibility: the server can be accessed from various platforms.
- Security: clients only have access to the server and not to other clients, which enhances security between them.

State Due to a good extent of states in which the car could be, we decided to implement a state pattern. This pattern was used to encapsulate varying behavior for the same object (Car) based on its internal state. The car object can change its behavior at runtime without resorting to large monolithic conditional statements and thus improving maintainability.

Underneath, a state tree with an hierarchical subdivision is presented:



3.Algorithm Design

Below, pseudocode of the most critical algorithmic part of the system is presented.

Finding the best power grid station for MSO

```
function bestPowerGridStation (Position destination) {

     PowerGridStation bestPowerGrid = new PowerGridStation()
     // MAX_FLOAT is the highest possible positive float value
     float minPenalty = MAX_FLOAT



     //Loop for every PowerGridStation p that is within 2000m
from the destination and has at least one parking spot
available
     for each PowerGridStation p such that
(p.position.distanceFrom(destination) <= 2000 && !
p.availableParkingSpots.isEmpty())
          float penalty = penalty(p, destination)
          if (penalty < minPenalty)
               bestPowerGrid = p
               minPenalty = penalty
     // If two powerGridStations have the same penalty value,
the closest to the destination is selected
     else if (penalty = minPenalty &&
p.position.distanceFrom(destination) <
bestPowerGrid.position.distanceFrom(destination))
               bestPowerGrid = p
               minPenalty = penalty
     return bestPowerGrid
}
```

```
function penalty (PowerGridStation p, Position destination) {
     // Define weights values
     float distanceWeight = 0.7
     float availabilityWeight = 0.3



     // Distance is in meters
     // Distance is multiplied by 0.01 to have a comparable
order of magnitude with the number of available parking spots
     float distance = p.position.distanceFrom(destination) *
0.01
     int busyParkingSpots = p.parkingSpots -
p.availableParkingSpots.size()


     return distance * distanceWeight + busyPlugs *
availabilityWeight
}
```

4.User interface Design

4.1 Mockups

Desktop mockup:

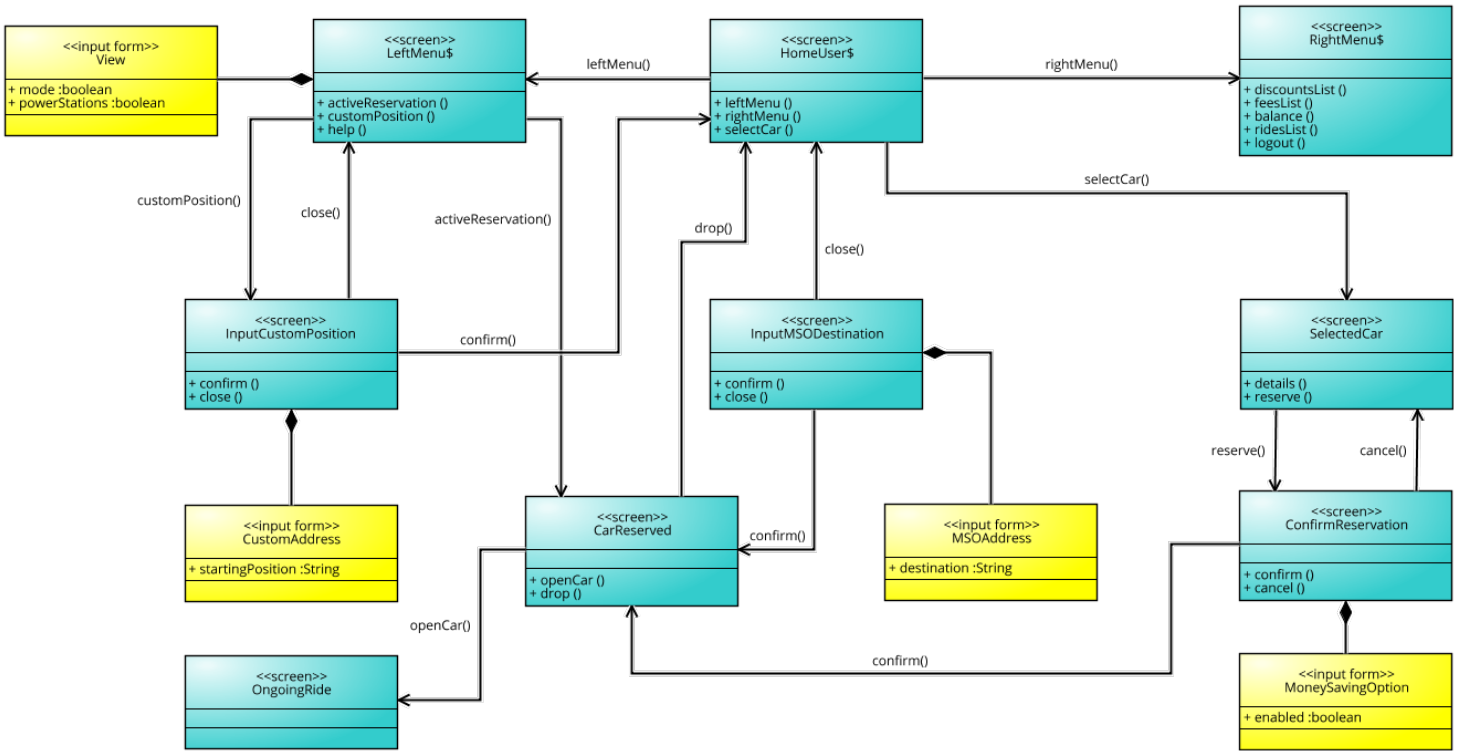
https://projects.invisionapp.com/share/WD9JAT2BN

Mobile mockup:

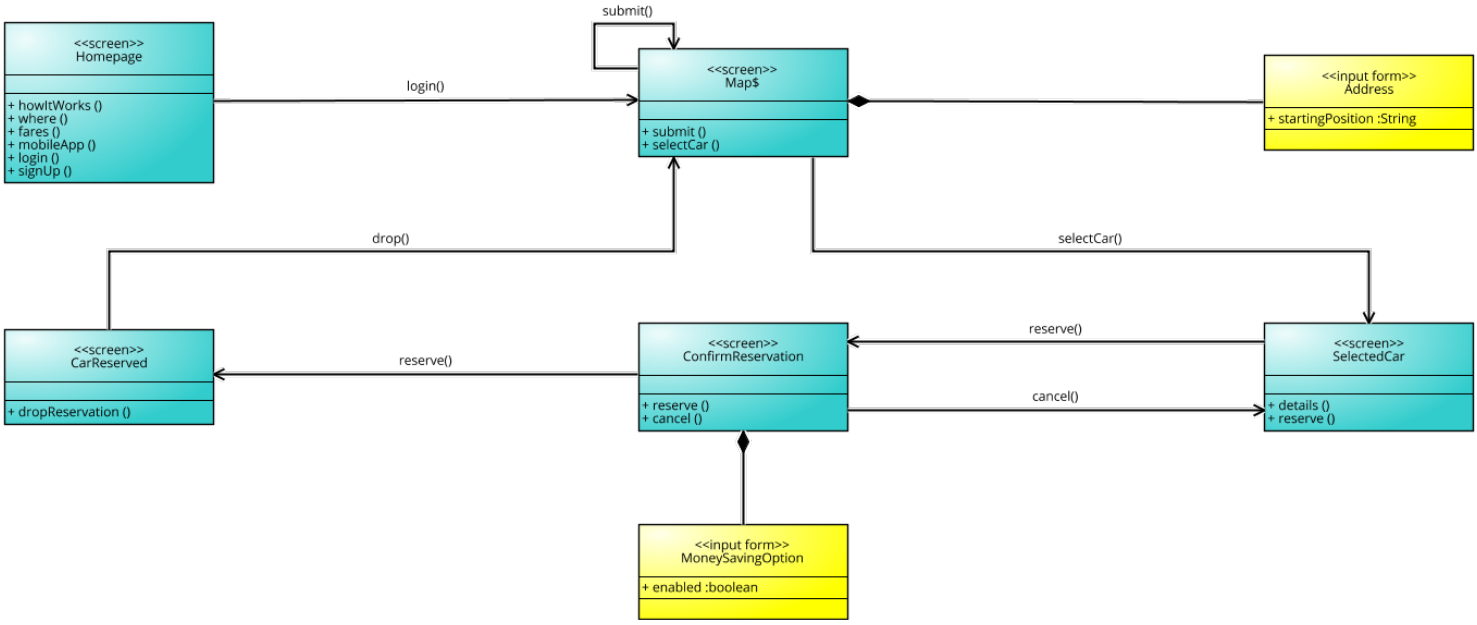
https://projects.invisionapp.com/share/BN9555RP7#/screens/200915521_Splash

4.2 UX diagrams

UX (user experience) diagrams are presented, in order to provide a clear explanation of the possible actions that a user can perform inside PowerEnJoy's applications.



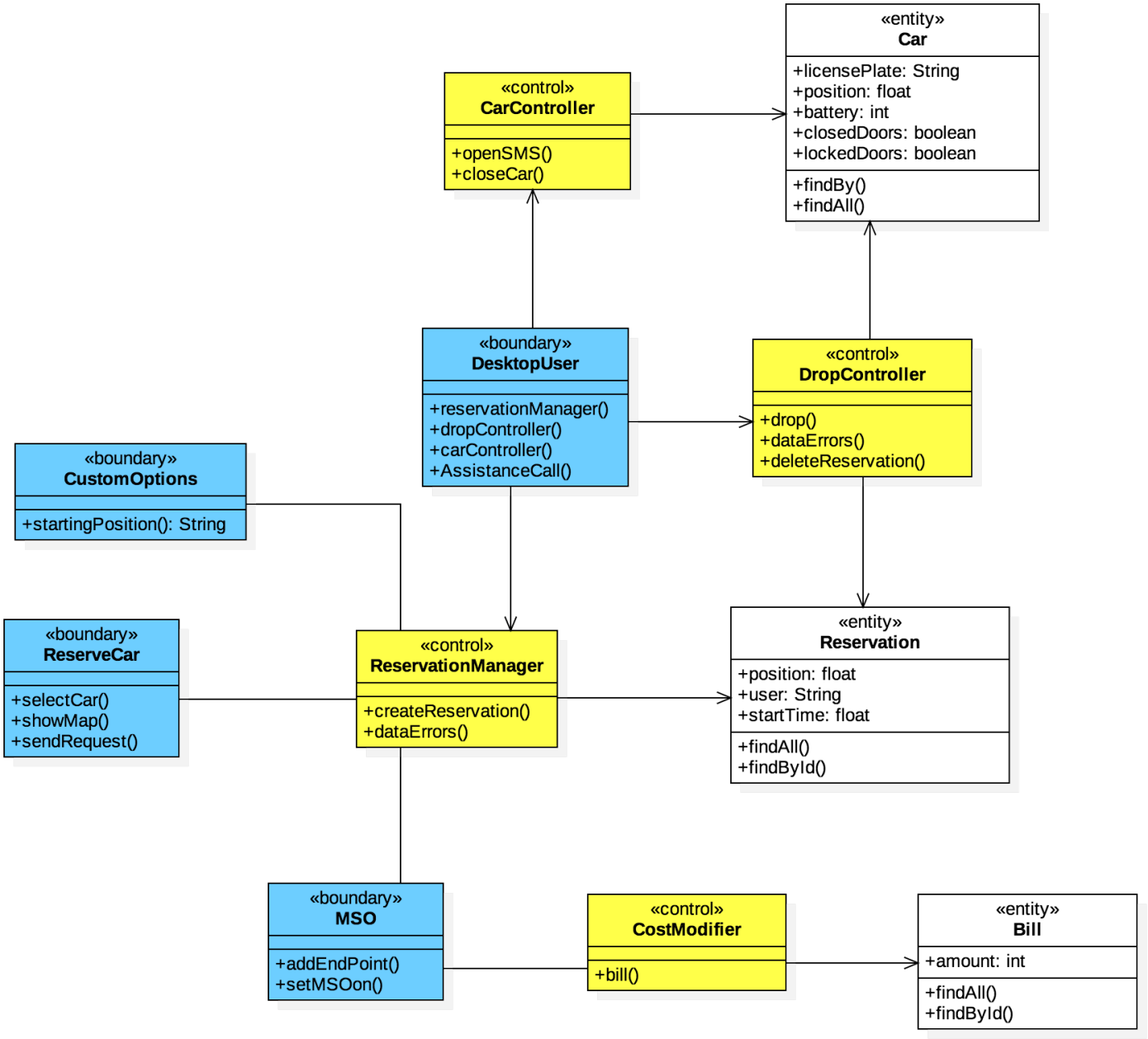
UX user mobile



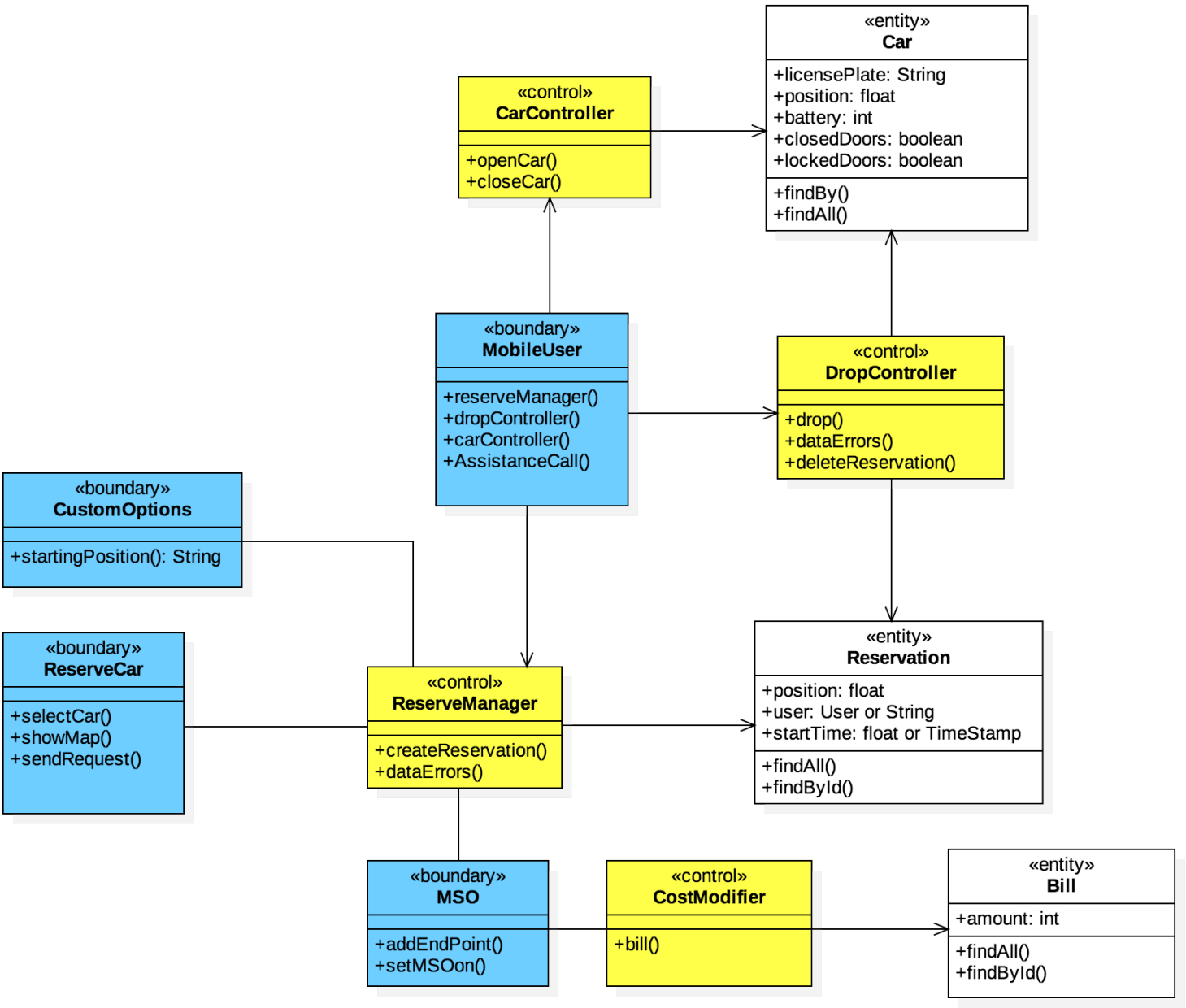
UX user desktop

4.3 BCE diagrams

BCE (Boundary Control Entity) diagrams are also introduced in the document, to show how each action, performed by the user, is managed inside PowerEnJoy's system and how these actions are linked within our model.



Desktop User


«entity»
Car
+licensePlate: String
+position: float
+battery: int
+closedDoors: boolean
+lockedDoors: boolean
+findBy()
+findAll()
«control»
CarController
+openCar()
+closeCar()
«boundary»
MobileUser
+reserveManager()
+dropController()
+carController()
+AssistanceCall()
«control»
DropController
+drop()
+dataErrors()
+deleteReservation()
«boundary»
CustomOptions
+startingPosition(): String
«boundary»
ReserveCar
+selectCar()
+showMap()
+sendRequest()
«control»
ReserveManager
+createReservation()
+dataErrors()
«entity»
Reservation
+position: float
+user: User or String
+startTime: float or TimeStamp
+findAll()
+findById()
«boundary»
MSO
+addEndPoint()
+setMSOon()
«control»
CostModifier
+bill()
«entity»
Bill
+amount: int
+findAll()
+findById()

Mobile User

5.Requirements Traceability

The system's design was outlined aiming to optimally fulfill the goals and the requirements that were specified in the RASD.

The reader can find here, under the list of all PowerEnJoy system's goals, how the requirements, previously stated in the RASD, map to the design elements that we have defined in this document:

- [G1] Allow registered users to find the location of available cars within a certain distance from their current location or from a specified address.

Requirements	Design elements
[R1] The system must be able to detect each car's location according to each car's GPS.	CarController MapController
[R2] The system must be able to detect each registered user's location according to the user's device GPS.	UserController MapController
[R3] The system must be able to compute the precise (walking) distance between two different GPS coordinates.	MapController

- [G2] Allow registered users to reserve a single car, among the available ones, for up to one hour before picking it up.

Requirements	Design elements
[R1] The system must not allow non registered user to reserve a car.	ReservationController
[R2] The system must not allow blocked users to reserve a car.	ReservationController
[R3] The system has to tag the car available again if a car is not picked-up within one hour from the reservation.	ReservationController
[R4] The system must allow registered user to drop a reservation for a car without any fee if they do it within one hour from the start of the reservation.	ReservationController
[R5] The system has to tag the car available again if the reservation that refers to it has been dropped by the user.	ReservationController CarController
[R6] The system must not allow a user to have more than one active reservation at the same time.	ReservationController
[R7] The system must not show in the application all the cars that are not available cars.	CarController MapController

Requirements	Design elements
[R8] The system must keep one car not available for reservation from the moment when a user reserves it until the moment when the car locks automatically at the end of the ride.	RideController CarController
[R9] The system must show into the application the license plate of the car that the user reserves.	ReservationController CarController

- [G3] Allow registered users to unlock the car that they previously reserved through the PowerEnJoy application.

Requirements	Design elements
[R1] The system must not let unreserved car to be unlocked.	ReservationController
[R2] The system must not allow any user to unlock a car for which they do not have an active reservation for.	ReservationController CarController
[R3] The system must allow cars unlocking only through the PowerEnJoy application.	CarController

- [G4] Incentivize the virtuous behaviors of the users to do something good for the environment.

Requirements	Design elements
[R1] The system must be able to detect how many passengers the car is carrying during a ride.	CarController RideController
[R2] The system must be able to detect each car's battery level.	CarController
[R3] The system is aware of a previously defined set of coordinates, in which each of the stored coordinates corresponds to a specific power grid station's location.	PowerGridStationController
[R4] The system must be able to determine if a car has been parked inside one of the power grid station's parking spot or not.	PowerGridStationController CarController
[R5] The system must start a 1-minute timeout from when the car locks automatically only if the car has been parked inside one of the power grid station's parking spot.	PowerGridStationController CarController RideController
[R6] The system must be able to tell if a car has been plugged into the power grid or not.	CarController

Requirements	Design elements
[R7] The system must be able to tell if the registered user has enabled the money saving option or not.	ReservationController
[R8] The system must provide information about the station where to leave the car to get a discount for each user that enables the money saving option.	MSOController MapController PowerGridStationController
[R9] The system must be able to detect how many power plugs are available at a certain power grid station.	PowerGridStationController
[R10] The system must be able to compute which are the power grid station that, with the addition of a car in that station's area, would guarantee a uniform distribution of cars in the city.	MapController PowerGridStationController
[R11] The system must apply a discount of 10% on the last ride for each user that took at least two other passengers onto the car.	PaymentController CarController
[R12] The system must apply a discount of 20% on the last ride for each user that left the car with no more than 50% of the battery empty.	PaymentController CarController

Requirements	Design elements
[R13] The system must apply a discount of 30% on the last ride for each user that takes care of plugging the car into the power grid to charge it.	PaymentController CarController
[R14] The system must apply a discount of 35% on the last ride for each user that enables the money saving option and actually plugs the car into the power grid of the station that the app provided them.	PaymentController ReservationController CarController PowerGridStationController
[R15] The system must compute an algebraic sum of the discount and fee percentages if there are more than one for the same ride, which means that discounts and fees percentages are cumulative.	PaymentController
[R16] The system must commit the total cost of the ride to the payment processor only after one minute from the automatic locking of the car.	PaymentController CarController RideController

- [G5] Guarantee the system's administration a proper use of the company's resources to maximize its own profit.

Requirements	Design elements
[R1] The system must be able to determine if a user's payment was successful or not.	CarController RideController
[R2] The system must be able to block users for which there was an unsuccessful payment.	AdministratorController PaymentController
[R3] The system must notify the user whenever their account gets blocked due to an unsuccessful payment.	NotificationController EmailGateway
[R4] The system must charge a user for a 1€ fee if they do not pick up the car within one hour from the reservation.	PaymentController ReservationController
[R5] The system must be able to detect "bad user behaviors" and block each user that is suspected of this type of conduct.	ReservationController AdministratorController
[R6] The system must be able to compute which is the nearest power grid station from a given car.	MapController CarController PowerGridStationController

Requirements	Design elements
[R7] The system must charge a user of 30% more on their last ride if the car is left at more than 3 KM from the nearest power grid station.	MapController CarController PowerGridStationController PaymentController
[R8] The system must charge a user of 30% more on their last ride if the car is left with more than 80% of the battery empty.	CarController PaymentController
[R9] The system must charge the user for a given amount of money per minute as soon as they ignite the car's engine.	RideController CarController PaymentController
[R10] The system must stop charging the user for a ride as soon as the car locks automatically.	RideController CarController
[R11] The system must lock a car automatically as soon as the car is parked in a safe area, the engine has been turned off, there are no passengers inside the car and all the doors are closed.	CarController MapController

Requirements	Design elements
[R12] The system must lock a car automatically only if all the previous conditions are met.	CarController
[R13] The system must notify the system's administration which are the cars that need assistance.	CarController AdministratorController
[R14] The system must be able to charge the users whenever the company receives a fee from Milan's traffic corps regarding the incorrect behavior of the user on the road.	PaymentController
[R15] The system must make a car available again if the car was charging but it eventually got to the full battery level.	CarController
[R16] The system must use the maximum number of passengers that were onto a car from the beginning of a ride to the end of a ride, to apply the passengers discount.	CarController PaymentManager
[R17] The system must not allow the registration of two users with the same email or the same driving license.	UserController

Requirements	Design elements
[R18] The system must provide new users with a password to access the application.	UserController
[R19] The system must accept only credit cards or Paypal as payment channels for each user.	UserController

- [G6] Allow registered users to properly enjoy the benefits of car sharing services, without having to worry about anything else besides driving the car.

Requirements	Design elements
[R1] The system must notify the users of the current charges for their ride through a screen on the car.	RideController

6.Effort spent

6.1 Alessandro Bianchi (25.30h)

- 18/11 3h
- 21/11 1.30h
- 24/11 2h
- 25/11 3h
- 28/11 1h
- 31/11 2.30h
- 1/12 2h
- 2/12 2h
- 4/12 2h
- 6/12 1.30h
- 7/12 1h
- 8/12 2h
- 9/12 2h

6.2 Francesco Canciani (18h)

- 18/11 3h
- 20/11 1h
- 22/11 2h
- 24/11 1h
- 26/11 1h
- 27/11 2h
- 2/12 2h
- 3/12 2h
- 4/12 1h
- 7/12 1h
- 8/12 2h

6.3 Lorenzo Cannone (18h)

- 18/11 3h
- 19/11 1h
- 22/11 1h
- 23/11 1h
- 26/11 2h
- 28/11 2h
- 1/12 2h
- 3/12 2h
- 5/12 1h
- 6/12 2h
- 8/12 1h

7.References

7.1 Used tools

A list of tools that we used to complete this document include:

- Google Documents: chapters write down
  - https://gsuite.google.com/intl/it/products/docs/
- Apple Pages: PDF creation
  - http://www.apple.com/it/pages/
- Lucidchart: sequence diagrams
  - https://www.lucidchart.com/
- Signavio Process Editor: UX Diagrams
  - http://www.signavio.com/
- Google Drive: file sharing and version updating
  - https://www.google.it/intl/it/drive/
- Photoshop CC, InVision: mockup
  - http://www.adobe.com/it/products/photoshop.html
  - https://www.invisionapp.com/
- GitHub: document release
  - https://github.com/
- Drawlo: Deployment view, High level components
  - https://www.draw.io/
- StraUML: component view, component interfaces, BCE diagrams
  - http://staruml.io/

Changelog

- Version 1.1: added sensors and actuators in car component in deployment view.
- Version 1.0: initial release.