Software Engineering 2: PowerEnjoy Design Document

Andrea Pace, Lorenzo Petrangeli, Tommaso Paulon

February 1, 2017

Version 1.0

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to show:

- the high level architecture
- the main design patterns
- the main components
- their interfaces and the way they are connected

1.2 Scope

This project has two targets of people: the user and the operator
The user will be able to sign up and make reservations, while the system will check if the data provided by the users are correct and will compute the cost of the ride.
The operator will receive notifications about problems with cars

1.3 Definitions, acronyms, abbreviations

1.4 Reference documents

- Rasd
- Assignments AA 2016-2017

2 Architectural design

2.1 Overview

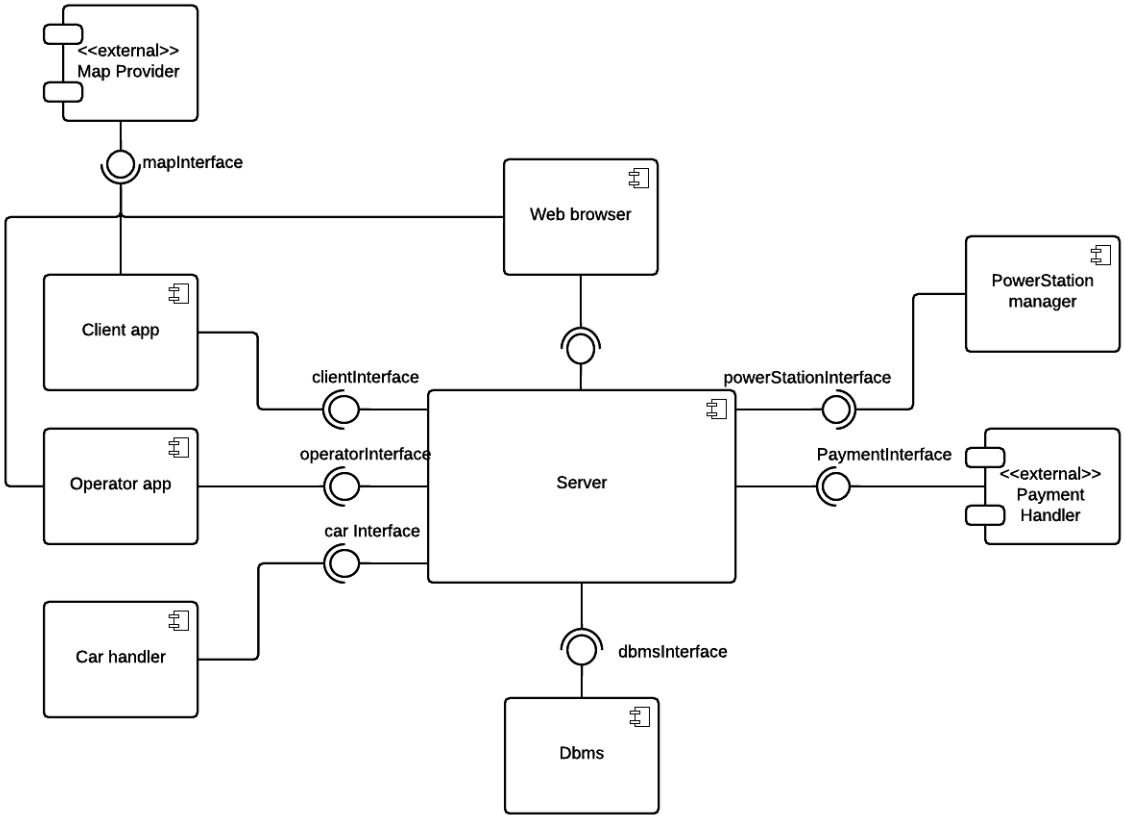
The system has a three tier architecture, we have:

- The client and the operator devices, plus the web server
- an application server which handles the business logic
- a database server

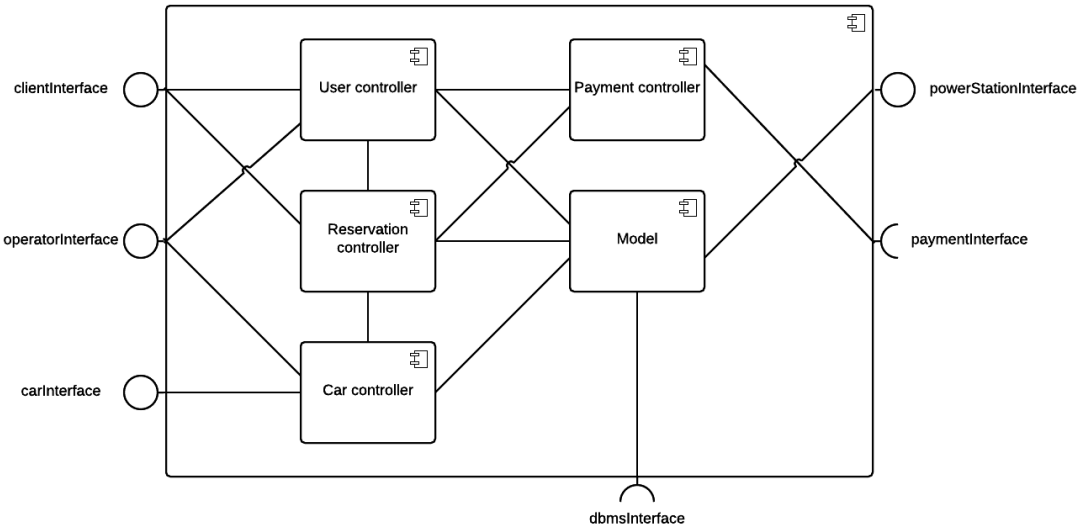
The

2.2 High level components and their interaction

The central server handles the communication between the different components.
The client can send requests from the app or from the web server and receives notifications. The requests that can be sent from the web server are a subset of the requests available via app.
The car handler is the software installed onto each car: it sends notifications about its status(battery level, current charge, position) and receives request for internal state change from the server (reservation and unlocking).
The power station manager is installed in each power station: it only send informations about the number of free plugs.
The payment handler receives payment data and returns a positive answer if the payment is accepted through an external interface.
The map provider simply receives positions and returns a map through an external interface to the client apps and to the browser


<<external>>
Map Provider
mapInterface
Web browser
Client app
clientInterface
PowerStation
manager
powerStationInterface
Operator app
operatorInterface
Server
PaymentInterface
<<external>>
Payment
Handler
car Interface
Car handler
dbmsInterface
Dbms

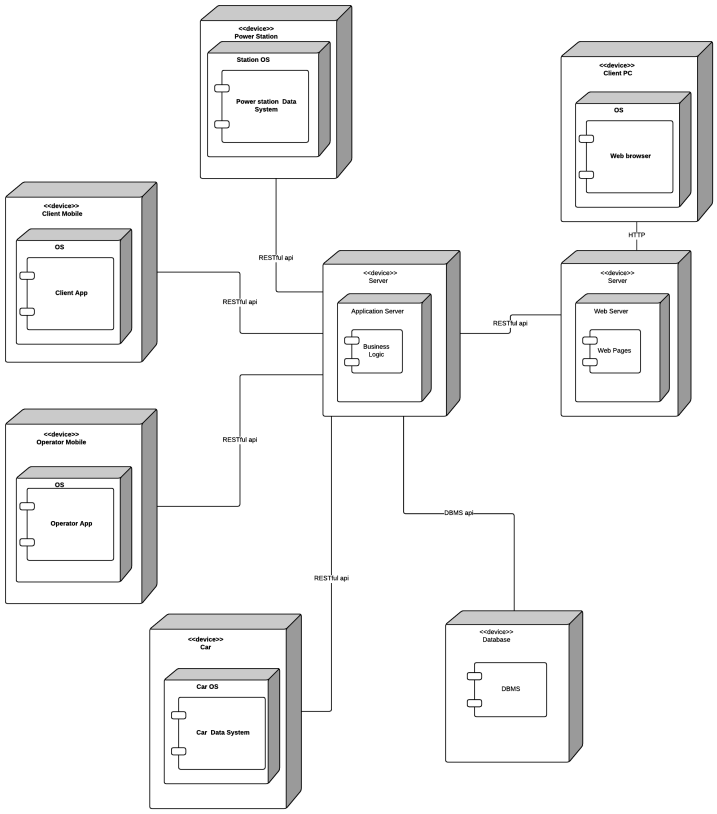
2.3 Component view



- User controller: handles login, registration and in general all the client and the operator data
- Reservation controller: handles the reservation, the unlock requests and submits the final charge to the payment controller. It interacts with the car controller in order to receive data concerning position, number of passengers and every information needed to compute the final charge
- Car controller: manages all the data from the car sensors. It is also in charge of notifying the operator for all the problems stated in the RASD
- Payment controller: manages the real communication with the payment handler. It also interacts with the User controller when the client decides to pay off his debts

The power stations communicate directly with the model, since the only information they share is the number of free plugs.
Every component interacts with the model, which is the only link to the DBMS.

2.4 Deploying view



2.5 Database structure

2.6 Runtime view

The following sequence diagrams show the interactions between components from the reservation of a car to the end of the ride. The interaction with the external payment handler is shown in a separate diagram since it's always the same process. Every interation with the database is actually handled by the Model component, but we always skip this component in the diagrams.

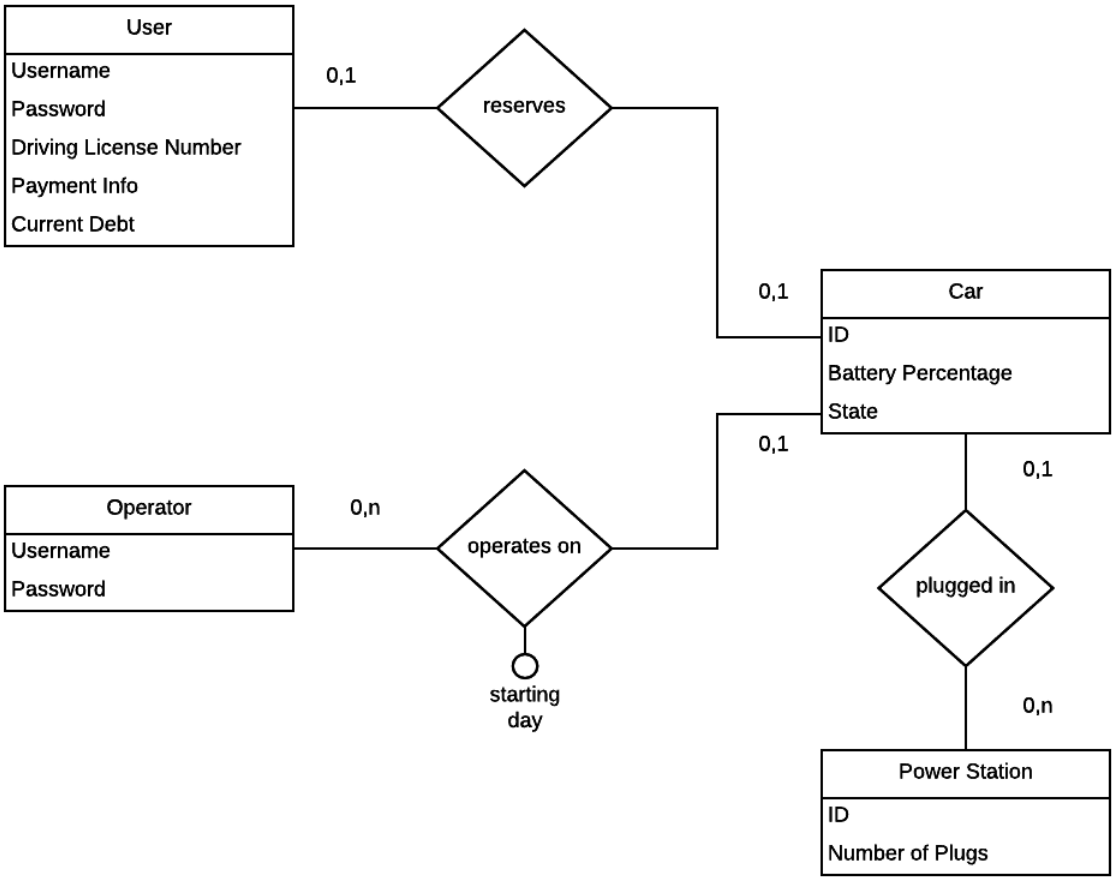
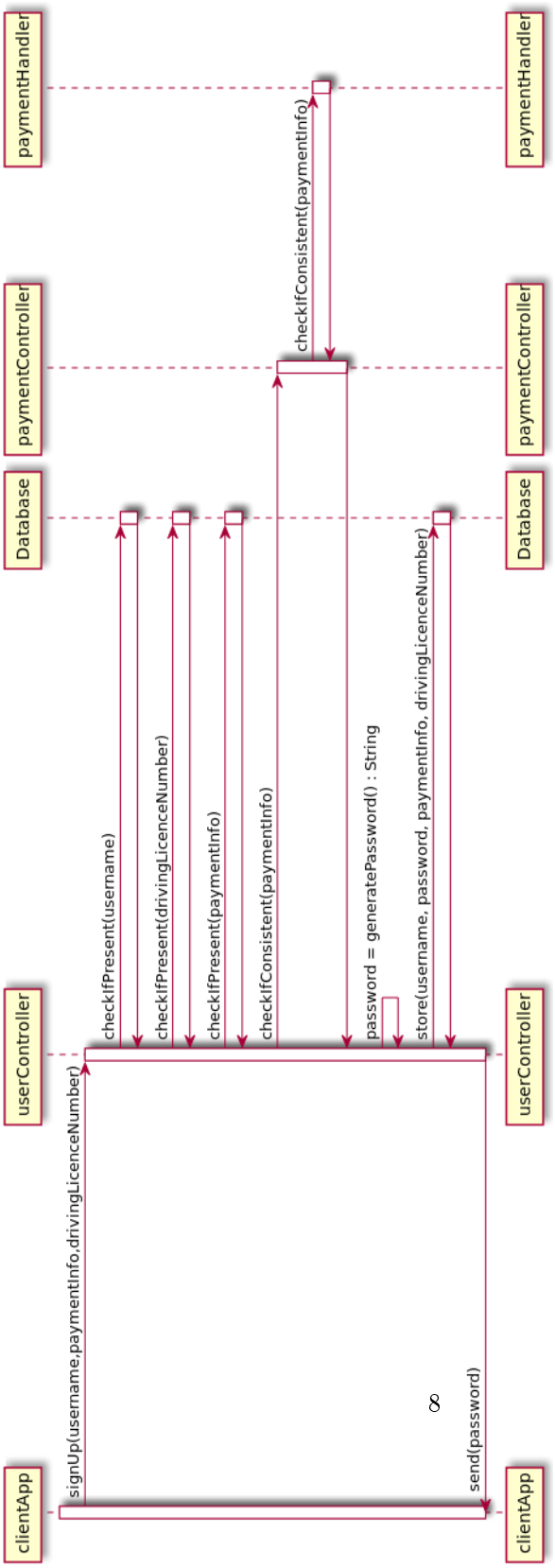


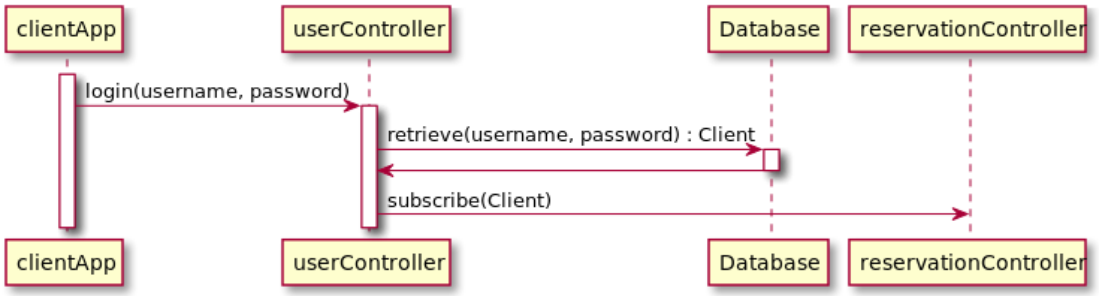
Figure 1: ER diagram

2.6.1 Registration



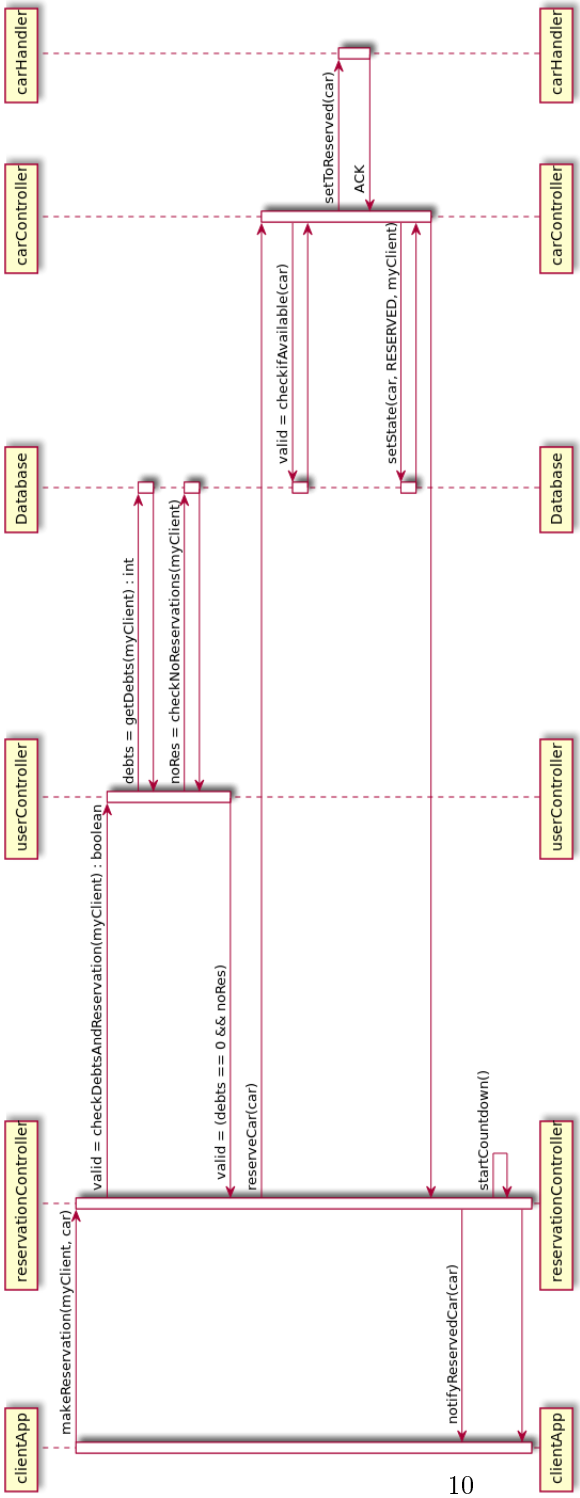
This diagram shows the operations performed when a user signs up. The system checks that every information provided is consistent and not already present in the database. If every information is correct, the system generates a password which is sent to the user and stores the data.

2.6.2 Login



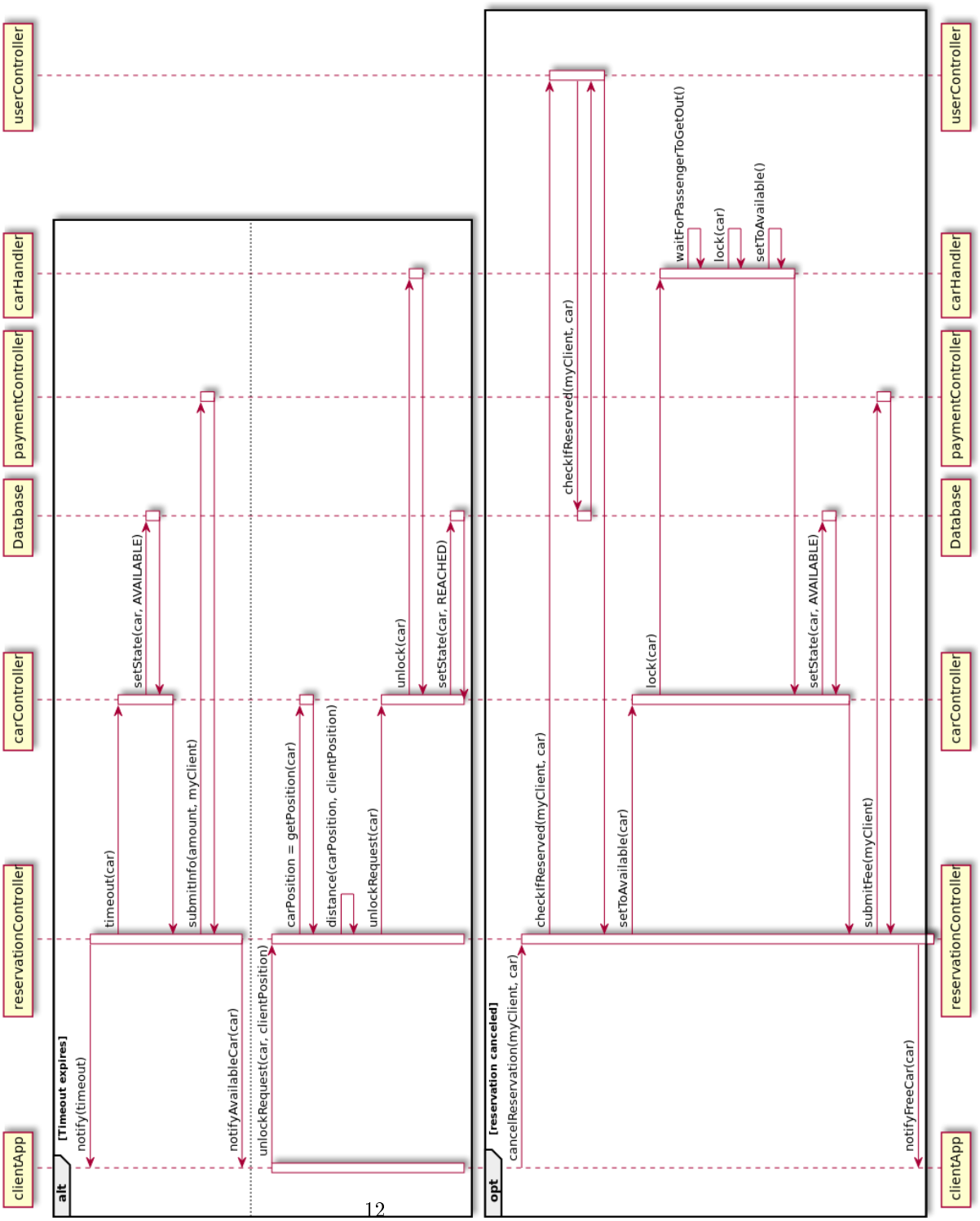
This diagram shows the login process. When a client enters the correct username and password, he subscribes to a topic related to available cars. In this way every time a car is reserved or released the reservation controller sends the current set of available cars to every client. This process is shown in the next diagrams.

2.6.3 Reservation



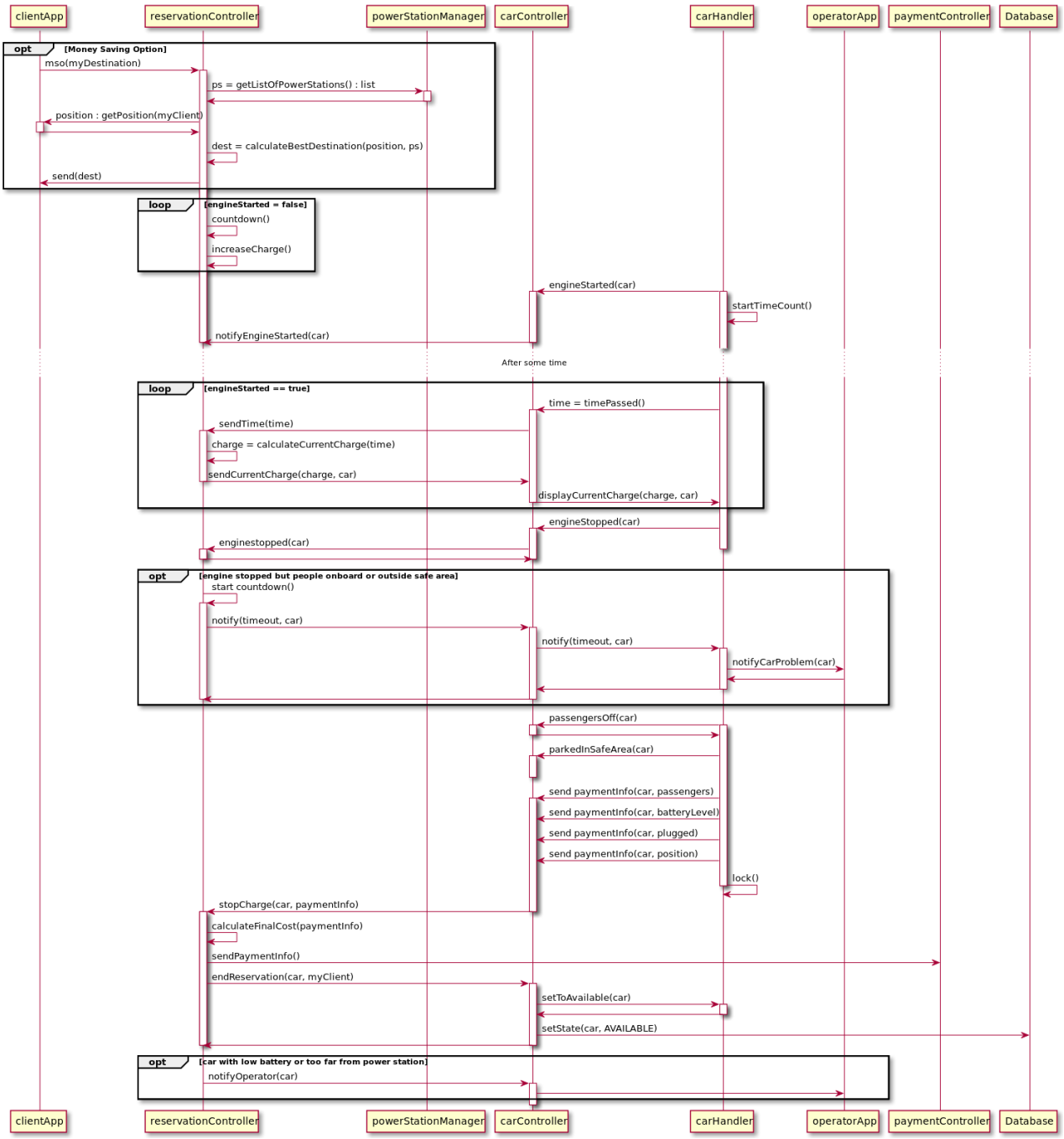
This diagram represents the reservation process. Every call is synchronous since no process can proceed without the previous operations.
This particolar view shows a situation where all the conditions to make a reservation are met. If any control fails the process simply stops. If the reservation is confirmed, the reservation controller broadcasts this information, so the user app is always up to date

2.6.4 Car unlocking



This diagram shows several situations. If the timeout expires before the unlock request the car becomes available again, and the client pays a fee and the availability of the car is broadcasted to all the clients
If the unlock request arrives before the countdown expires, the system checks if the client's position is close enough. In this view we show the system behaviour when this condition is met. A reservation cancel can occur in this part of the process. In this case the system checks if there is a real reservation and if this occurs the reservation is canceled. According to the requirements we should check if the engine is already ignited, but we simply don't accept a reservation cancel after that moment. As usual the availability of the car is broadcasted.

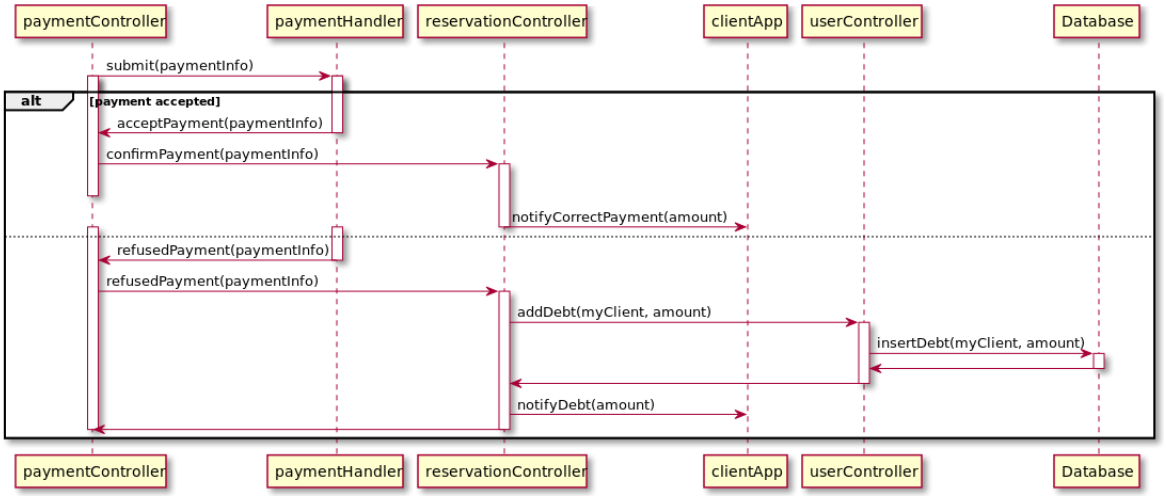
2.6.5 End of the Ride



The first optional part concerns the money saving option. The list returned

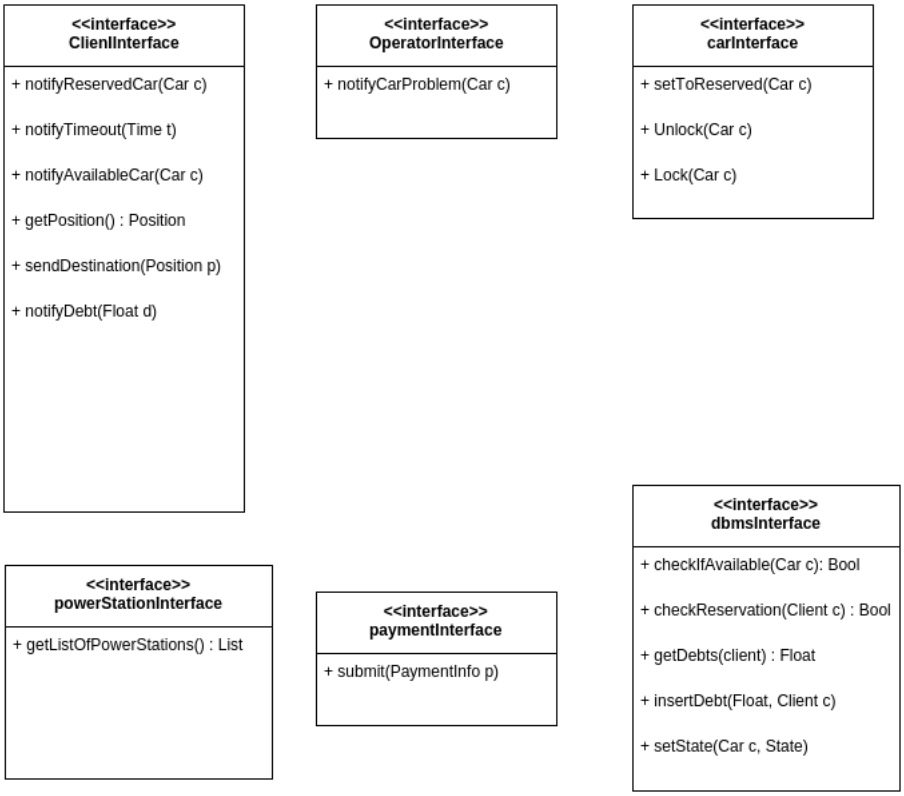
by the powerStationManager contains the number of free plugs of each power stations.
The system is notified of this option after the unlocking of the car.
Then the system periodically increases the client charge if he/she doesn't ignite the engine.
During the ride the system periodically sends the current charge to the car, in order to display the amount to the user.
Then we have different behaviours if the client parks the car in a safe area or not. If the client leaves the car outside the safe area or doesn't leave the car the current charge keeps increasing. In those cases an operator is notified.
In the final part an operator is notified of a car left too far from the nearest power station or left with low battery

2.6.6 Payment process



This diagram shows the two situation when a payment is submitted to the payment handler.

2.7 Component Interfaces



These interfaces provide access to the client apps and to external components.

2.8 Other design decisions

For what concerns the safe zones we decided to use an XML file with all the information required to identify such areas. The set of zones is predefined by our customer, but with this choice they can be updated quite easily.

2.9 Selected architectural styles and patterns

2.9.1 Protocols and architecture

The application is divided into three tiers: Database, application logic and thin client: this choice seems appropriate with the nature of our application and thin client has been chosen in order for our app to be used on devices with poor/average computational resources.
The design followed mainly a top-down approach since we start from a high-level perspective, even if some bottom-level components were already available, like the external components.

We have a relational database with ACID properties who communicates with the application logic tier through the dbms APIs.
The application logic server interacts with the web server and the clients with RESTful api. This approach has several advantages: being a stateless protocol means that no session of the client is saved on the server, each request is independent from previous requests and contains all the information to be interpreted correctly.
The application server is made of stateless components: this ensures that multiple instances of the components could be created and they could serve any request in the same way in order to guarantee scalability.
The thin client layer handles the GUI, receives notifications and can send requests.
We also use HTTP between the web server and the clients.
The client app comunicates with the map provider through its API.

Then we chose to design the application logic server with an approach driven by the requirements, as shown in the Requirements Traceability section, which shows how the requirements are fullfilled by the system components

2.9.2 Design Patterns

The mvc pattern concerns the whole system: the model contains the internal representation of the world, the controllers are the handlers of the business logic and the views are represented by the clients applications. In this way any of the three conponents can be changed regardless of the others.
Then also a pub/sub pattern is used: the reservation controller notifies all the currently logged users about reserved and released cars. The user subscribes to this topic when he logs in.

3 Algorithm design

3.1 Calculate cost:

the algorithm will start considerating the amount of money calculated by the carHandler, which depends only on the duration of the ride. Then the algorithm will apply penalties, if any :

- if a car is left at more than 3 KM from the nearest power station or with more than 80% of the battery empty,the algorithm charges 30% more

And then benefits will be considered:

- if the car sensor detected more than two passenger during the travel the algorithm applies a 10% discount
- if the car is left with more than 50% of battery, the algorithm applies a 20% discount

- if the car is plugged into a power grid and it's left in a safe area, the algorithm applies a 30% discount

where a x% discount(charge) on a value y means: y = y - y*x/100 (y = y + y*x/100)

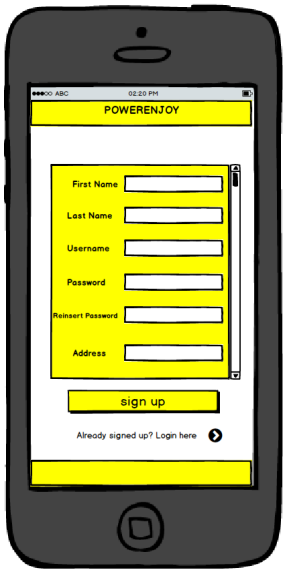
3.2 Power station for money saving option:

The chosen power station will be the one closest to the requested destination with a percentage of free plugs greater or equal to the average of the percentage of the free plugs of all the power stations.

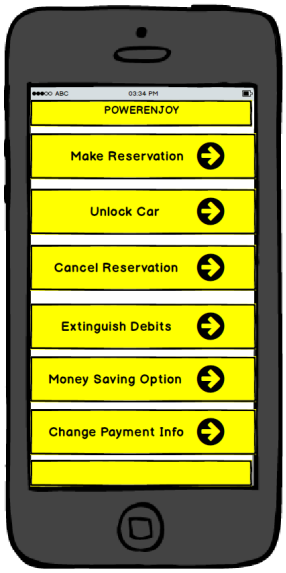
4 User Interface design

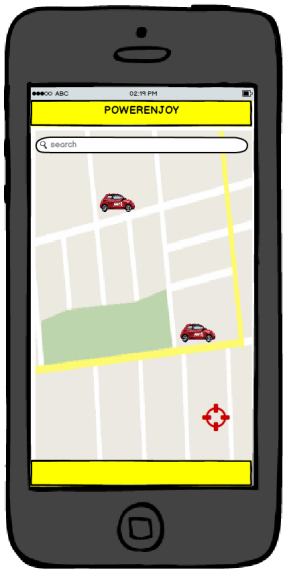
4.1 Mockups

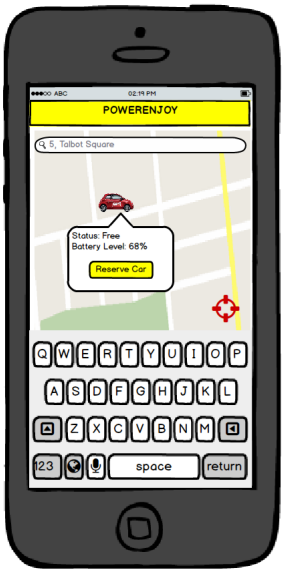
4.1.1 Client App

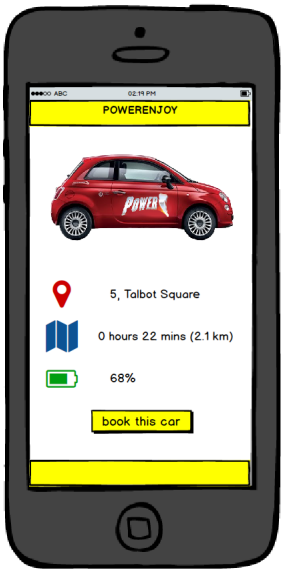



ABC
02:05 PM
POWERENJOY
Username
Password
login
Not yet registered? Register now!

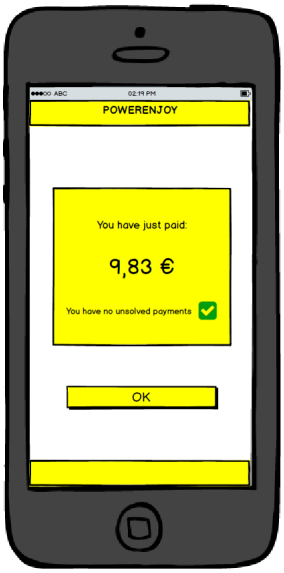

ABC
03:34 PM
POWERENJOY
Make Reservation
Unlock Car
Cancel Reservation
Extinguish Debits
Money Saving Option
Change Payment Info

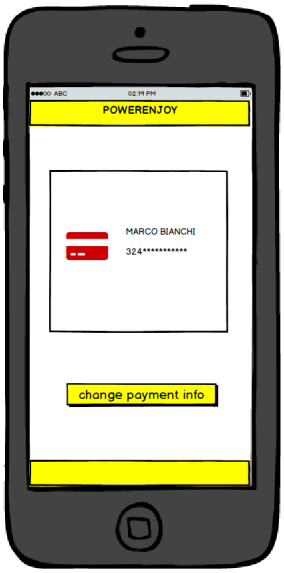

POWERENJOY
search


POWERENJOY
5, Talbot Square
Status: Free
Battery Level: 68%
Reserve Car
space
return


POWERENJOY
5, Talbot Square
0 hours 22 mins (2.1 km)
68%
book this car

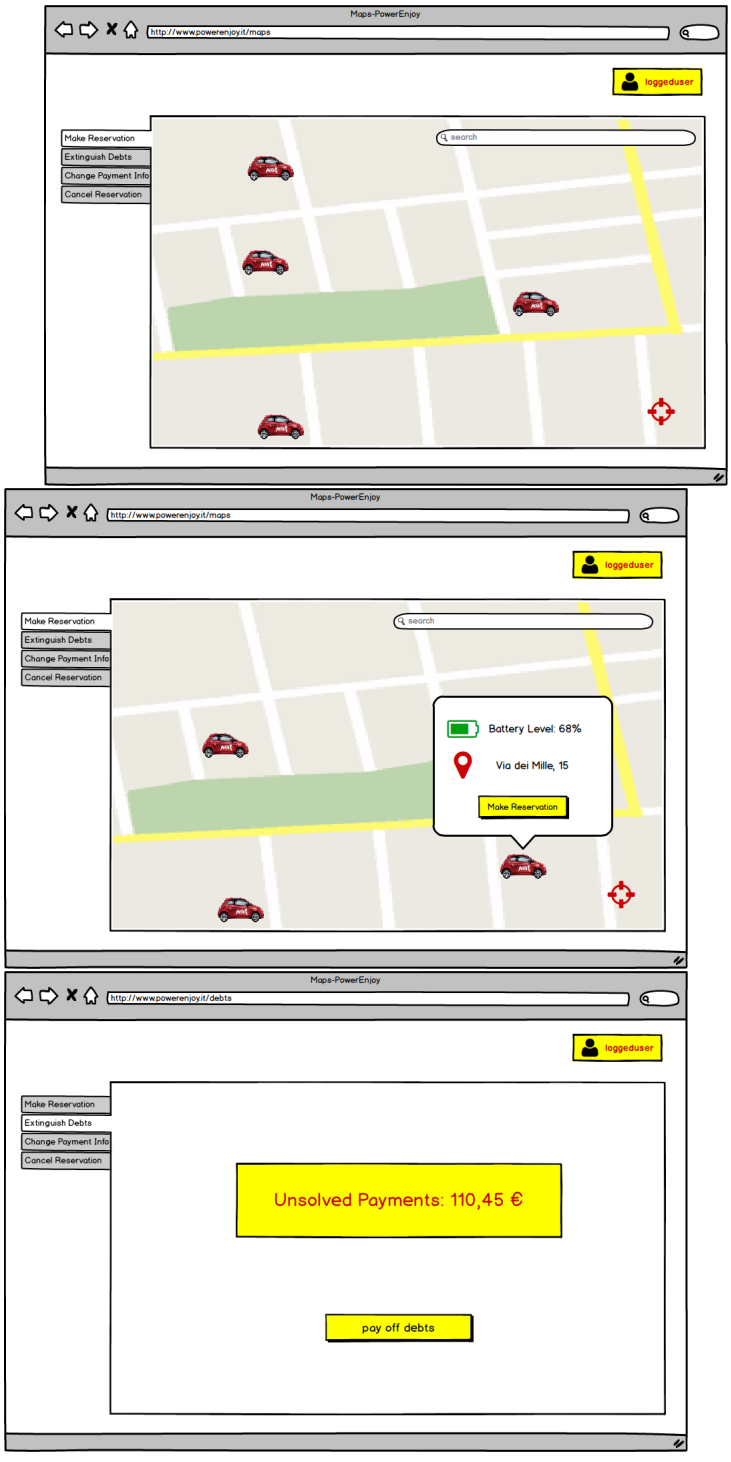

POWERENJOY
5, Talbot Square
43:33
unlock car



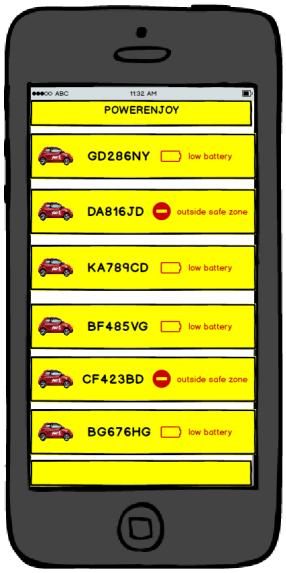


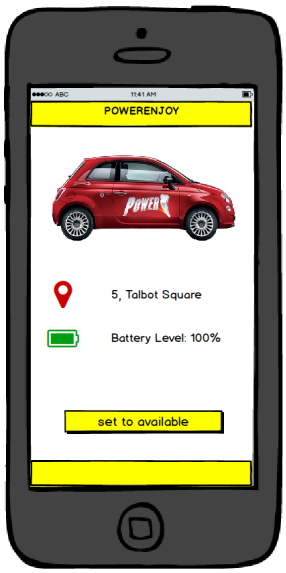
4.1.2 Client Web

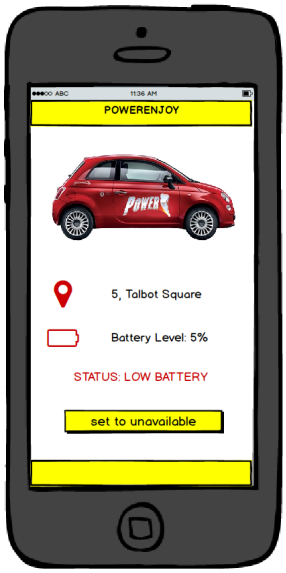
The web interface offers a subset of the functionality available with the mobile app: money saving option and car unlocking are not supported via web app.


Maps-PowerEnjoy
http://www.powerenjoy.it/maps
loggeduser
Make Reservation
Extinguish Debts
Change Payment Info
Cancel Reservation
search
Maps-PowerEnjoy
http://www.powerenjoy.it/maps
loggeduser
Make Reservation
Extinguish Debts
Change Payment Info
Cancel Reservation
search
Battery Level: 68%
Via dei Mille, 15
Make Reservation
Maps-PowerEnjoy
http://www.powerenjoy.it/debts
loggeduser
Make Reservation
Extinguish Debts
Change Payment Info
Cancel Reservation
Unsolved Payments: 110,45 €
pay off debts

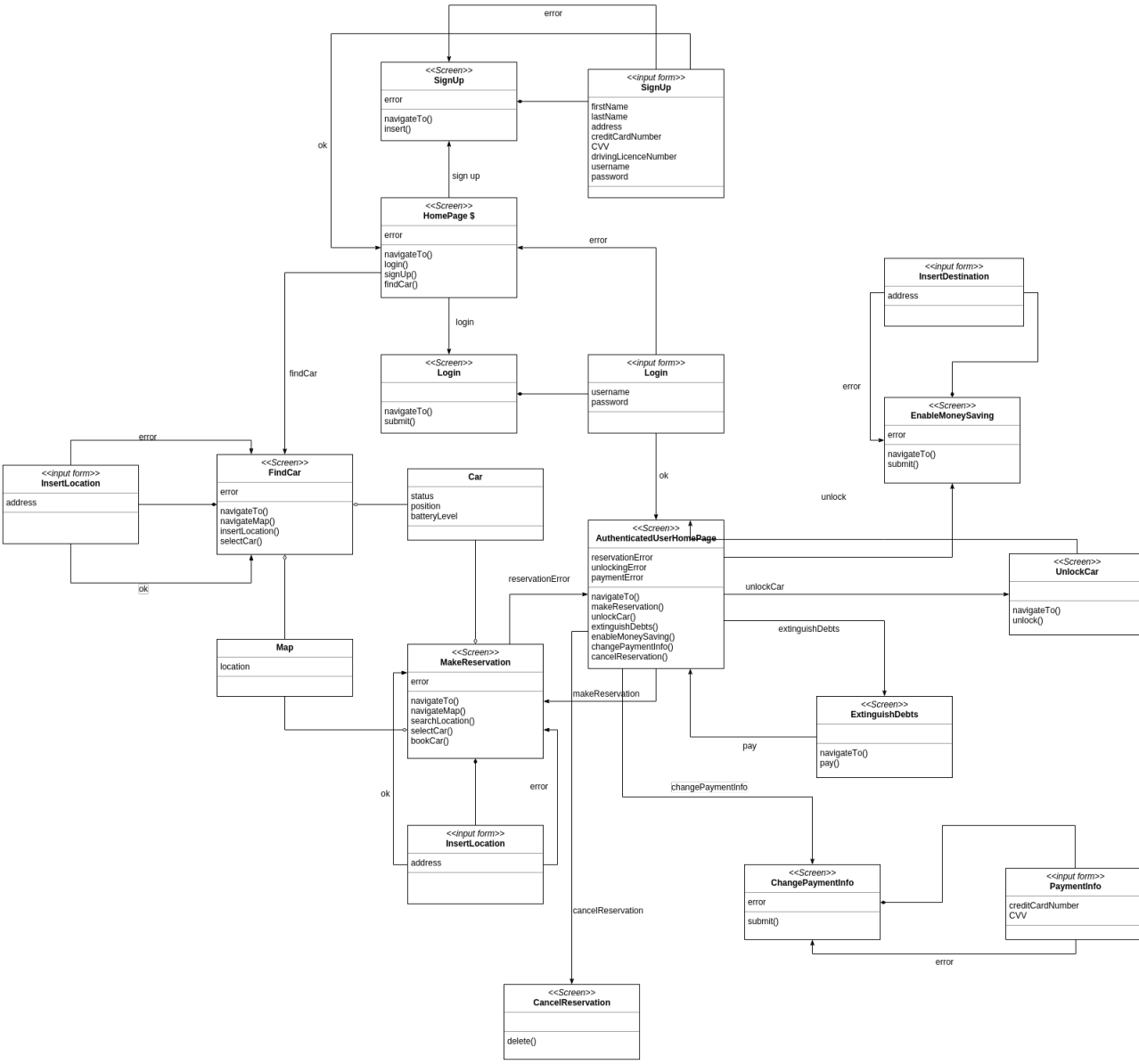
4.1.3 Operator App

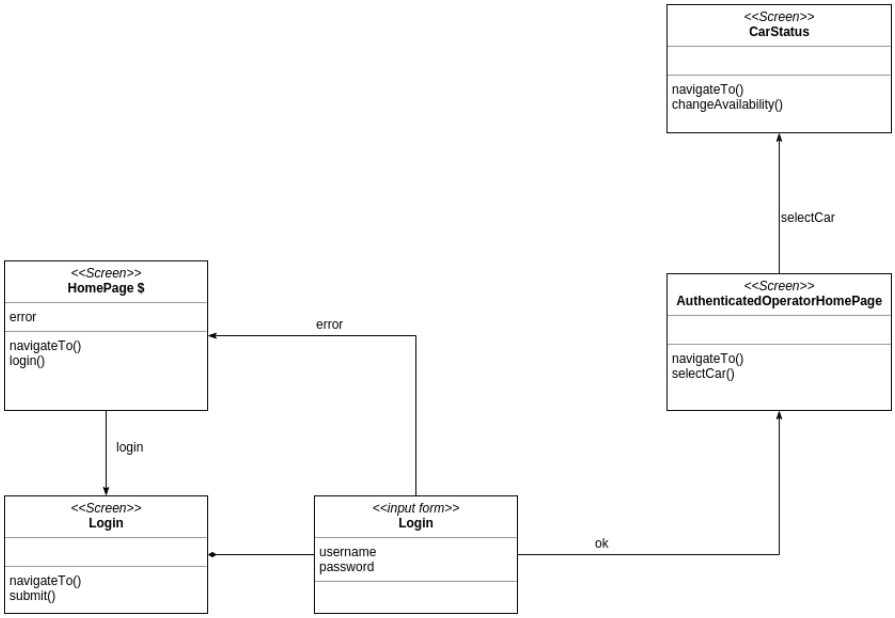




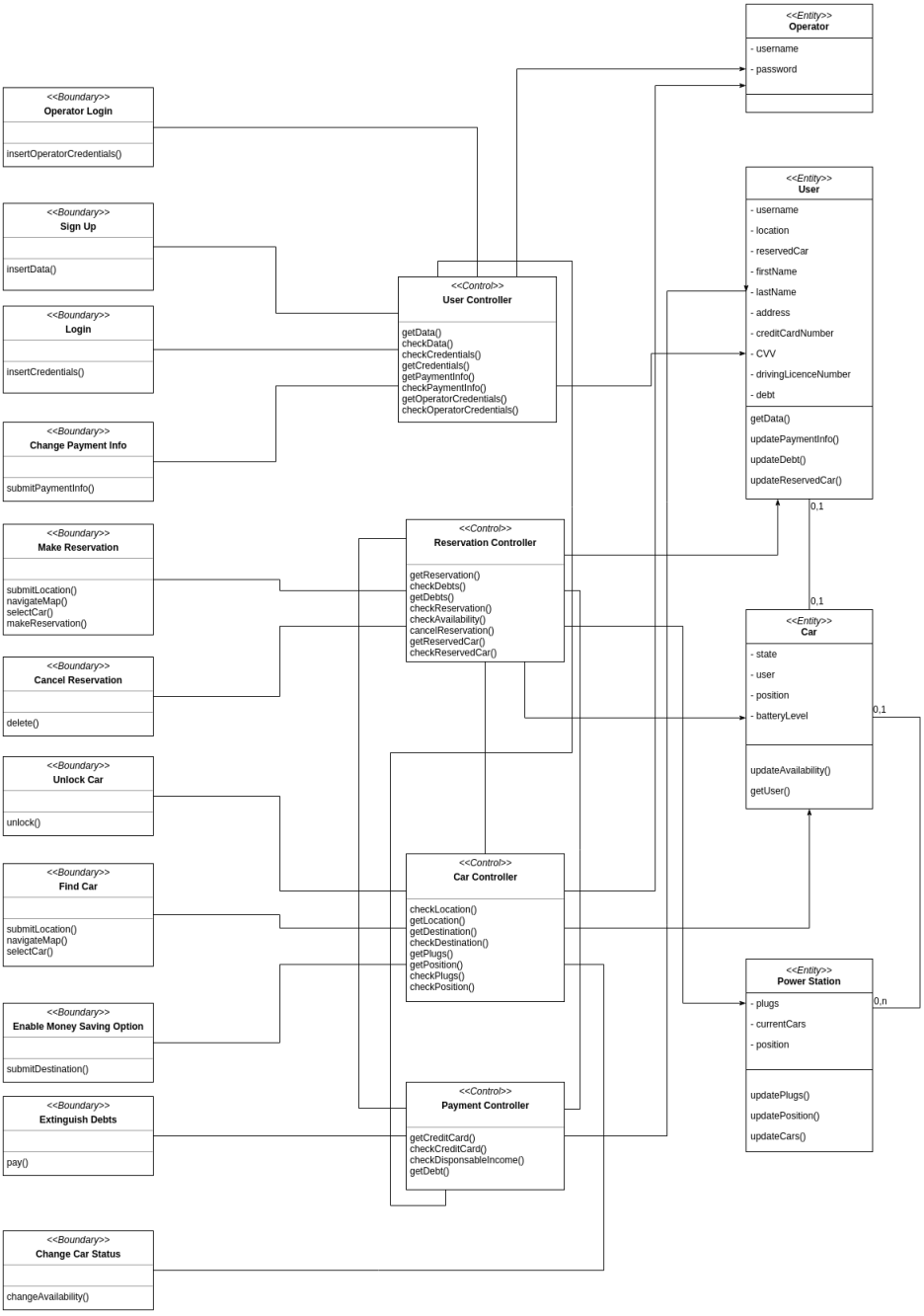

ABC
11:36 AM
POWERENJOY
POWER
5, Talbot Square
Battery Level: 5%
STATUS: LOW BATTERY
set to unavailable

4.2 Ux diagram




<<Screen>>
CarStatus
navigateTo()
changeAvailability()
selectCar
<<Screen>>
AuthenticatedOperatorHomePage
navigateTo()
selectCar()
error
<<Screen>>
HomePage $
error
navigateTo()
login()
login
<<Screen>>
Login
navigateTo()
submit()
<<input form>>
Login
username
password
ok

4.3 Bce diagram



5 Requirement traceability

This is the detailed correspondence between requirements and our system components

[G1] Allow users to sign up, providing their driver license number, payment information and a username

#	**Requirement**	**Mapping on architecture**
1	check whether the username is consistent	User Controller
2	check whether the driving license number is consistent	User Controller
3	check whether the payment information is consistent	Payment Handler, Payment Controller
4	check whether the username, the driving license ad the payment information are not already registered	Model, User Controller
5	send back a (valid) password only if the conditions listed above are true	User Controller, Client App
6	store credentials, payment information, drive license number, username and password	User Controller, Model

[G2] Allow users to sign in

#	**Requirement**	**Mapping on architecture**
1	let the user log in only if the provided username and password are correct	User Controller, Model, Client App

[G3a] Show available cars within a certain distance from current location

#	**Requirement**	**Mapping on architecture**
1	detect the position of every PowerEnjoy car	Car Handler, Car Controller, Model
2	detect the current position of the user	Client App
3	the system should be provided with a map of the area covered by PowerEnjoy	Map provider
4	check whether the specified distance is positive	Client App
5	provide a map with available cars within the specified distance from the user's position	Map provider, Client App, Model
6	Include battery charge among the information about a car	Client App, Model

[G3b] Show available cars within a certain distance from a specified address

#	**Requirement**	**Mapping on architecture**
1	know the position of every PowerEnjoy car	Car Handler, Car Controller, Model
2	check whether the specified distance is positive	Client App
3	the system should be provided with a map of the area covered by PowerEnjoy	Map provider
4	check whether the address is consistent	Client App
5	provide a map with available cars within the specified distance from the specified address	Client App, Map provider, Model
6	include battery charge among the information about a car	Client App, Model

[G4] Let users reserve a single car

#	**Requirement**	**Mapping on architecture**
1	allow the user to choose a car from the ones available	Client App, Reservation Controller
2	prevent the user from choosing a car if he's already reserved one	Model
3	once the car is reserved , it shouldn't be shown in the map between available cars	Reservation Controller, Client App

[G5] If a car is not picked up within an hour from the reservation, the system tags the car as available again, and the reservation expires; the user pays a fee of 1 EUR;

#	**Requirement**	**Mapping on architecture**
1	start a countdown of a predefined time when the user selects a car	Reservation Controller
2	show the user how much time is left	Client App
3	detect if the user reaches the car he reserved	Client App
4	make the car visible in the map to every user if the countdown ends and the user has not notified the system	Reservation Controller, Model, Client App
5	increase the amount of money that the user has to pay if the car si not reached before the countdown ends	User Controller
6	cancel the user's reservation after the countdown ends	Reservation Controller, User Controller, Model

[G6] A user that reaches a reserved car must be able to tell the system he's nearby, so the system unlocks the car and the user may enter. From that moment on, the user periodically pays an additional amount of money after a predefined time if he/she doesn't start the engine

#	**Requirement**	**Mapping on architecture**
1	let the user ask to unlock the car he reserved	Client App
2	unlock a car when asked to and when the user's distance(via GPS) from the car is lesser than a predefined value	Client App, Model
3	charge the user with a predefined amount of money every time a predefined time has passed	User Controller, Reservation Controller

[G7] As soon as the engine ignites, the system starts charging the user for a given amount of money per minute; the user is notified of the current charges through a screen on the car.

#	**Requirement**	**Mapping on architecture**
1	detect when the engine ignites	Car Handler, Car Controller
2	measure how long the user keeps the engine running	Car Handler
3	charge the user for an amount of money directly proportional to how long the engine is running, starting when the engine ignites	Car Handler, Car Controller
4	show the current charges through a screen on the car	Car Handler

[G8] The system stops charging the user as soon as the car is parked in a safe area and the user exits the car; at this point the system locks the car automatically. Then the system submits to the payment handler the total amount of money that the user has to pay.

#	**Requirement**	**Mapping on architecture**
1	detect when the engine turns off	Car Handler, , Car Controller
2	check whether the car is left in a safe area with the engine turned off	Car Handler, Car Controller
3	detect when all the passengers leave the car	Car Handler
4	stop charging the user when the conditions listed above are satisfied	Reservation Controller
5	lock the car	Car Handler
6	interact with the payment handler by submitting the final amount of money and the payment information of the user	Payment Handler, Payment Controller, User Controller

[G9] The final amount of money can be increased or decreased depending on the user behaviour. Parking too far from a power grid station or leaving a car with low battery will result in a penalty. Leaving the car with a certain amount of battery charge, taking other passengers onto the car or plugging the car into the power grid inside a safe zone will result in a benefit. An operator must be notified when a car is left with low battery charge and when a car is left ouside the safe area form more than a predefined time.

#	**Requirement**	**Mapping on architecture**
1	detect the battery charge of the car	Car Handler
2	check whether the car is plugged into the power grid	Car Handler
3	calculate the distance between the car and a power grid station	PowerStation Manager, Car Handler, Car Controller, Reservation Controller
4	detect the number of passengers during the travel	Car Handler
5	notify an operator if a car is left with battery charge below a predefined percentage	Car Handler, Car Controller, Operator App
6	detect when the engine is turned off	Car Handler
7	notify an operator when a car is left with its engine turned off outside the safe area	Car Handler, Car Controller, Model, Operator App

[G10] If the user enables the money saving option, he/she can input his/her final destination and the system provides information about the station where to leave the car to get a discount. This station is determined to ensure a uniform distribution of cars in the city and depends both on the destination of the user and on the availability of power plugs at the selected station

#	Requirement	Mapping on architecture
1	detect when the money saving option is chosen	Client App
2	let the user select a destination	Client App
3	choose the power station closest to the user's destination with a number of free plugs greater or equal to the average number of free plugs among all the power station owned by PowerEnjoy	Reservation Controller, PowerStation Manager

[G11] If the payment handler notifies the system about incorrect payment informations or about an insufficient balance, the user is prevented from reserving other cars. He can change his payment information and communicate to the system that he wants to pay off his debt. If the payment doesn't go wrong, the user can reserve cars again

#	Requirement	Mapping on architecture
1	receive notifications about successful payments, wrong informations or insufficient balance from the payment handler	Payment Controller, Payment Handler,
2	prevent the user from reserving a car if the payment handler sends a notification about wrong informations or insufficient balance	User Controller, Reservation Controller
3	let the user change his payment informations and ask for a new payment	Client App, User Controller
4	let the user reserve cars again if the payment is successful	User Controller

G[12] The user must be able to cancel his reservation until he starts the engine. He still has to pay a fee of 1 EUR

#	Requirement	Mapping on architecture
1	check if the user has really reserved a car	User Controller, Model,
2	check if the engine has been started	Car Handler, Car Controller
3	increase the amount of money that the user has to pay if the reservation is canceled	User Controller
4	lock the car as soon as soon as no one is inside	Car Handler, Car Controller
5	make the car visible to every user again	Reservation Controller, Client App

[G13] Every operator must be able to change any car status to unavailable. Then they must be able to revert the status back to available

#	**Requirement**	**Mapping on architecture**
1	let the operators change the status of the car	Operator App, Car Controller, Car Handler, Model

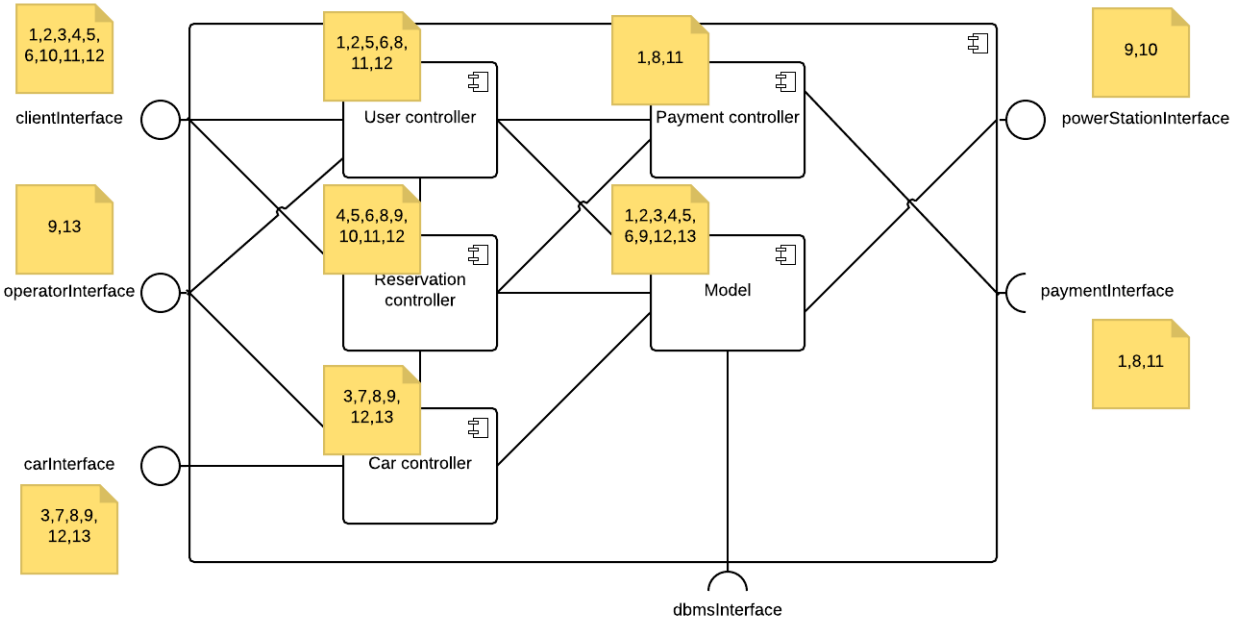


Figure 2: Correspondence between goals and components

6 Hours of work

- Andrea Pace: 30 hours
- Lorenzo Petrangeli: 20 hours
- Tommaso Paulon: 15 hours

7 Changelog

v1.1 added registration and login sequence diagrams
v1.2 minor changes to Unlock sequence diagram and add ER diagram

v1.3 better specification of the architecture and the requirements traceability