Design Document

December 11, 2016

Giacomo Bossi

Marco Nanni

Content

1. INTRODUCTION

1.1 Purpose

This software design document describes the architecture and design of PowerEnJoy System, which is a digital management system for a car-sharing service that employs exclusively electric cars.

The document contains a functional description of the system and useful views which show different aspects of PowerEnJoy System.

The document explains the decisions and trade-offs chosen in the design process for the mobile application architecture. It also shows more technical details such as a high level architecture, patterns used, and several different views. Views give to the developers a better explanation of what our system should do, regardless its code implementation.

1.2 Scope

The project PowerEnJoy consists of a digital management system for a car-sharing service based on a mobile application that users can access by providing their credentials. Clients need to be registered since only adult people with a valid driving license can use PowerEnJoy electric cars. Once logged in, users can enjoy all the functionalities provided by the system such as search for an available car (by providing their GPS positions or by specifying an address), reserve a car, drive the car and end the ride.

Contrary to the other car-sharing services available in the city of Milan, PowerEnJoy exclusively employs electric cars. In this way, it also allows users to recharge their rented cars' batteries in specific areas equipped with power grid stations.

Our project also consists of a system installed on PowerEnJoy cars, which permits to get more information about car's employment.

1.3 Definitions, acronyms and abbreviations

- RASD: Requirements Analysis and Specification Document
- DD: Design Document
- SSN: Social Security Number
- DB: Database
- DBMS: Database Management System
- GPS: Global Positioning System
- SoB: System on board
- JEE: Java Enterprise Edition
- SD: Sequence Diagram
- UX: User Experience
- BCE: Boundary Control Entity
- MVC: Model View Controller
- Area = Zone

1.4 Reference documents

- PowerEnJoy RASD
- Assignments 1 and 2 of the specification document
- Mobile Application Architecture Guide, J.D. Meier Alex Homer David Hill Jason Taylor Prashant, 2009 Microsoft Corp.

1.5 Document structure

Our software design document is structured in seven sections, which are:

- **Introduction**

  This section describes the purpose and scope of this Design Document and its differences from the previous one, the Requirements Analysis and Specification Document.

- **Architectural design**

  This section is divided in seven subsections:

  - Overview: it gives an overview of our system architecture and the interaction between components
  - Component view: it shows the system components
  - Deployment view: it shows the components that must be deployed in order to run the application and interact with the system
  - Runtime view: in this part, several sequence diagrams show different possible runtime situations during the employment of the system
  - Component interfaces: here there are interfaces between components
  - Selected architectural styles and patterns: it shows our choices for the development of the application in terms of styles and patterns
  - Other design decisions

- **Algorithm design**

  This part shows the most important algorithm of our application. Pseudo-code is used, so that the algorithm implementation is not taken into account.

- **User interface design**

  This section shows the user experience enjoying our PowerEnJoy application by UX and BCE diagrams.

- **Requirements traceability**

  This part points out how the goals listed in the RASD are linked to the design elements of this document.

- **Effort spent**

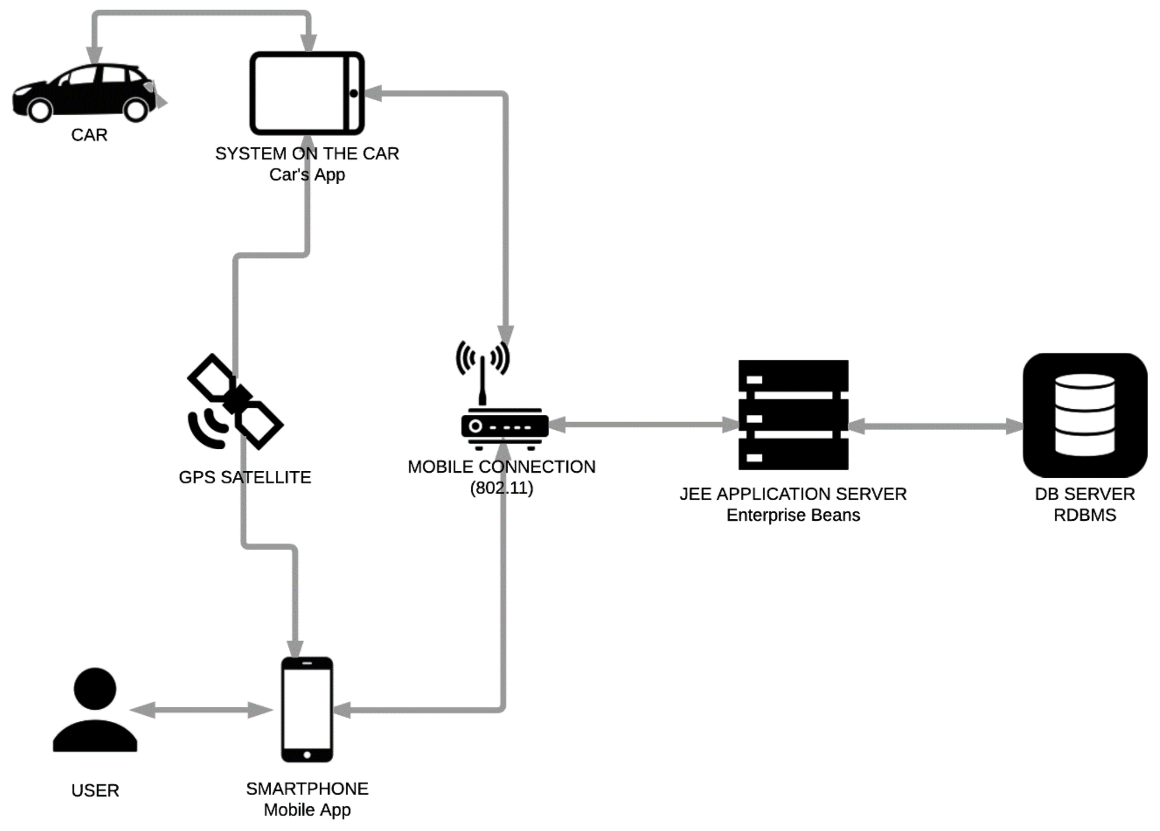
  Members' hours of work.

- **References**

  References used to draw up this document.

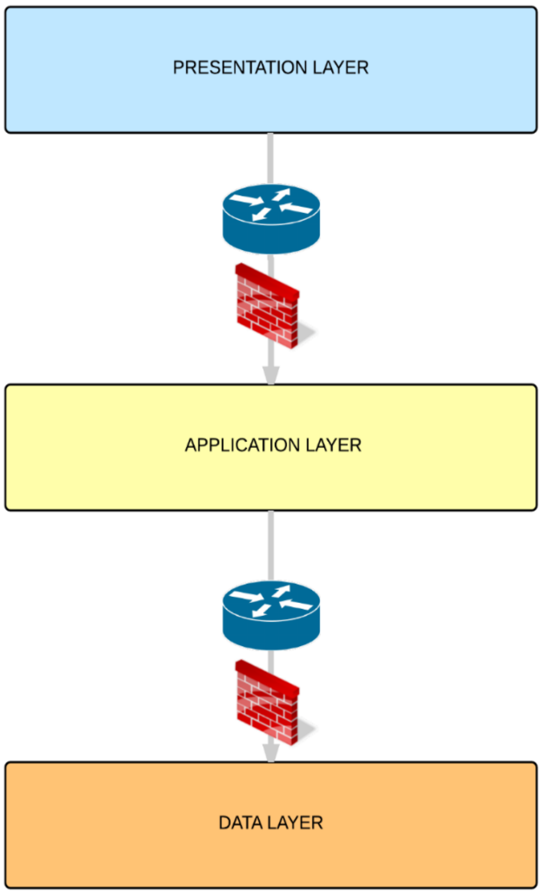
2. ARCHITECTURAL DESIGN

2.1 Overview



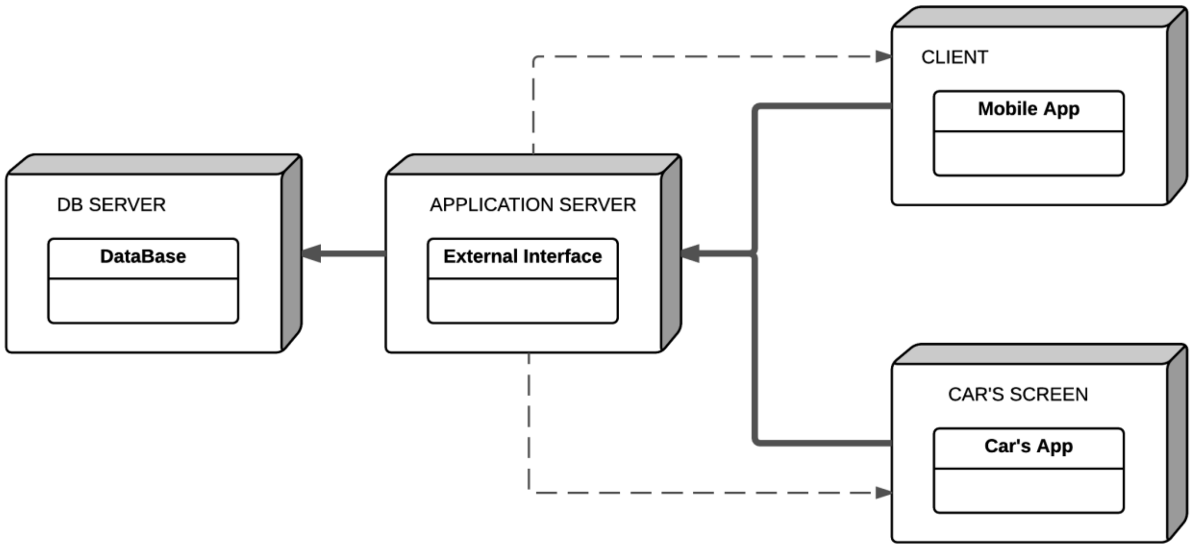
This diagram gives an overview of our system architecture and it can be seen that:

- users can interact with PowerEnJoy system through a mobile App installed on their smartphones
- PowerEnJoy electric cars are equipped with a system on board that sends pieces of information to the Application Server
- both the system on the car and the user's smartphone must be connected to a GPS satellite to get their position and send it to the system by a mobile connection
- JEE Application Server communicates with a Database Server in order to store all users' and cars' pieces of information regarding the system's employment



This figure stresses the level of security that our PowerEnJoy system is going to have. It has to ensure that some crucial operations - such as transactions and identity thefts - will not compromise the user's experience and safety. In this way, it can be seen that one firewall is placed between presentation and application layers and another one between application and data layers.

2.1.1 High level components and their interaction



The high-level component structure consists of four elements:

- Client
- Car Screen
- Application Server
- DB Server

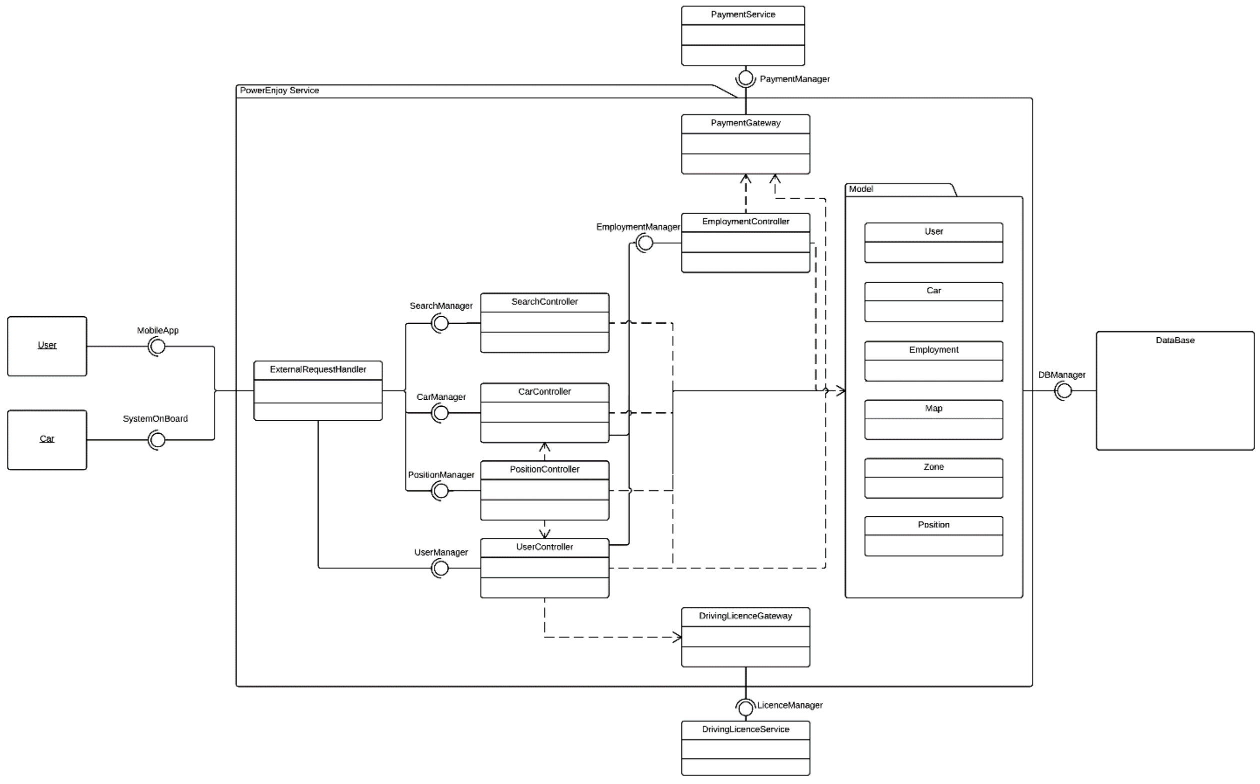
The most important and frequent operation the user can do while enjoying our application is to search for an available car. Once the user has done a search request, the application server shows the user a map with all the available cars located near the specified address or position. Given that the system does not use backtracking, every time a search request is done, the application server transfers the request to the DB server which queries the database in order to get the map with the cars' positions.

From their smartphones, users can also reserve a car. Once the user has reached the car, the system detects if the user's distance from the car is less than a proximity value. If this is true, the user is allowed to open the car.

When a user decides to open the reserved car, the application changes the car's status, sets the begin time of the employment and unlocks the car. After that, the user gets into the car and interacts with the SoB of the car. At this point, on the car's screen, there's another security check; the user has to enter his/her personal pin code, the one provided during the registration phase.

During the ride, the user can see some information on the car's screen such as his/her current charge, the car's battery level, the distance from the nearest charging point and the number of passengers detected by the car's sensor. At the end of the ride the user can plug the car into a power grid and recharge it, before the payment. In this case, the car's application will send its new status to the main system. Once the operation has finished, the system notifies the user his/her total cost for the ride and sends to an external financial service the payment's request.

2.2 Component view



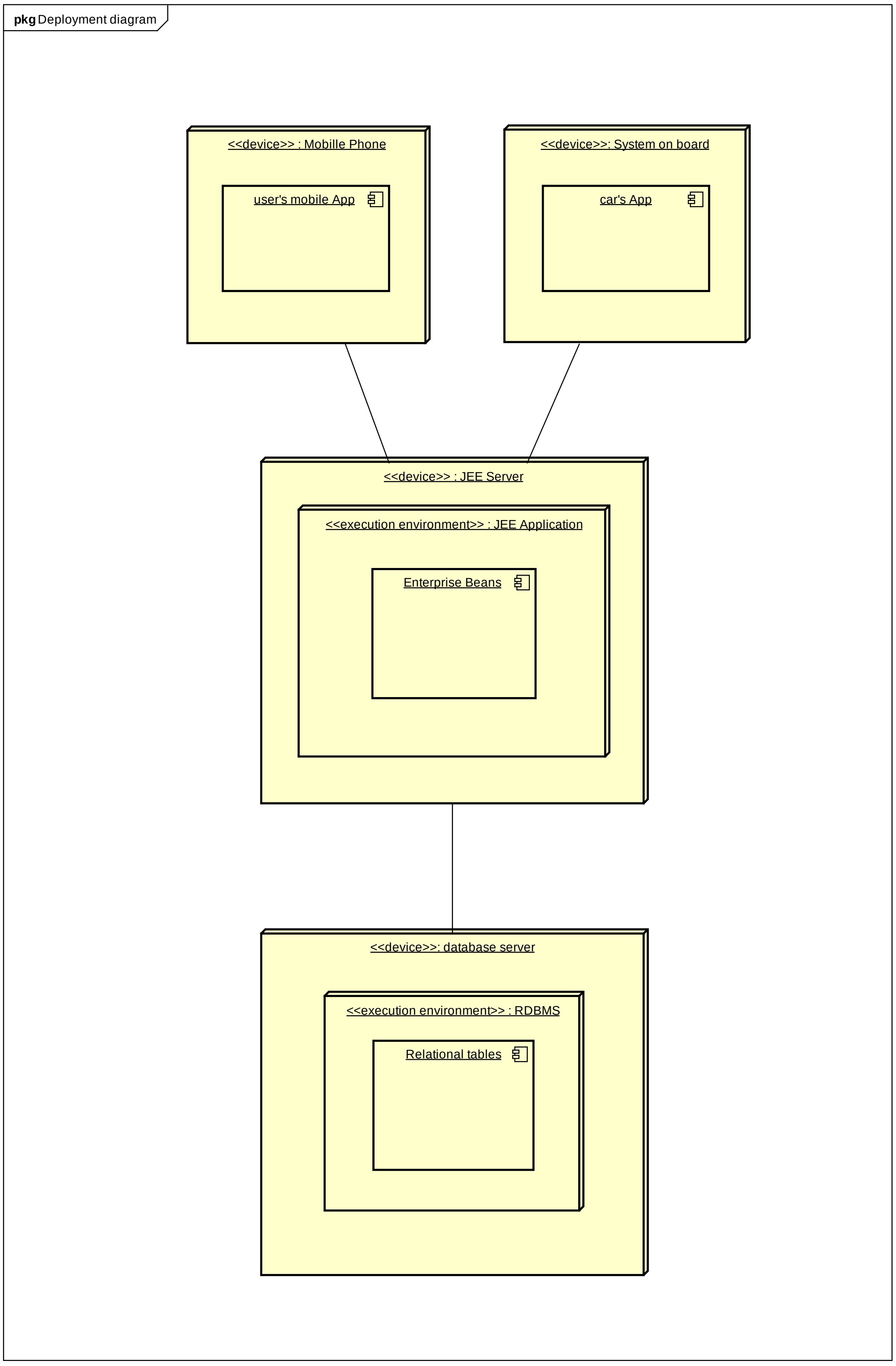
INTERNAL COMPONENT

ExternalRequestHandler	Manages the external request from the Mobile App and SoB and links them to the requested interfaces
CarController	Manages car request and data provided by sensors installed on it
UserController	Manages user request and data
EmploymentController	Manages the creation, the evolution and the end of each employment. It also manages the user's and the car's status
SearchController	Creates a new search over the internal map model
PositionController	Updates the users' and cars' positions
Model	Our simplified representation of the world, the data we interact with
DriverLicenceGateway	Manages the correctness of user data over a Ministerial Service
PaymentGateway	Manages employments' payments and the correctness of user payment data

EXTERNAL COMPONENT

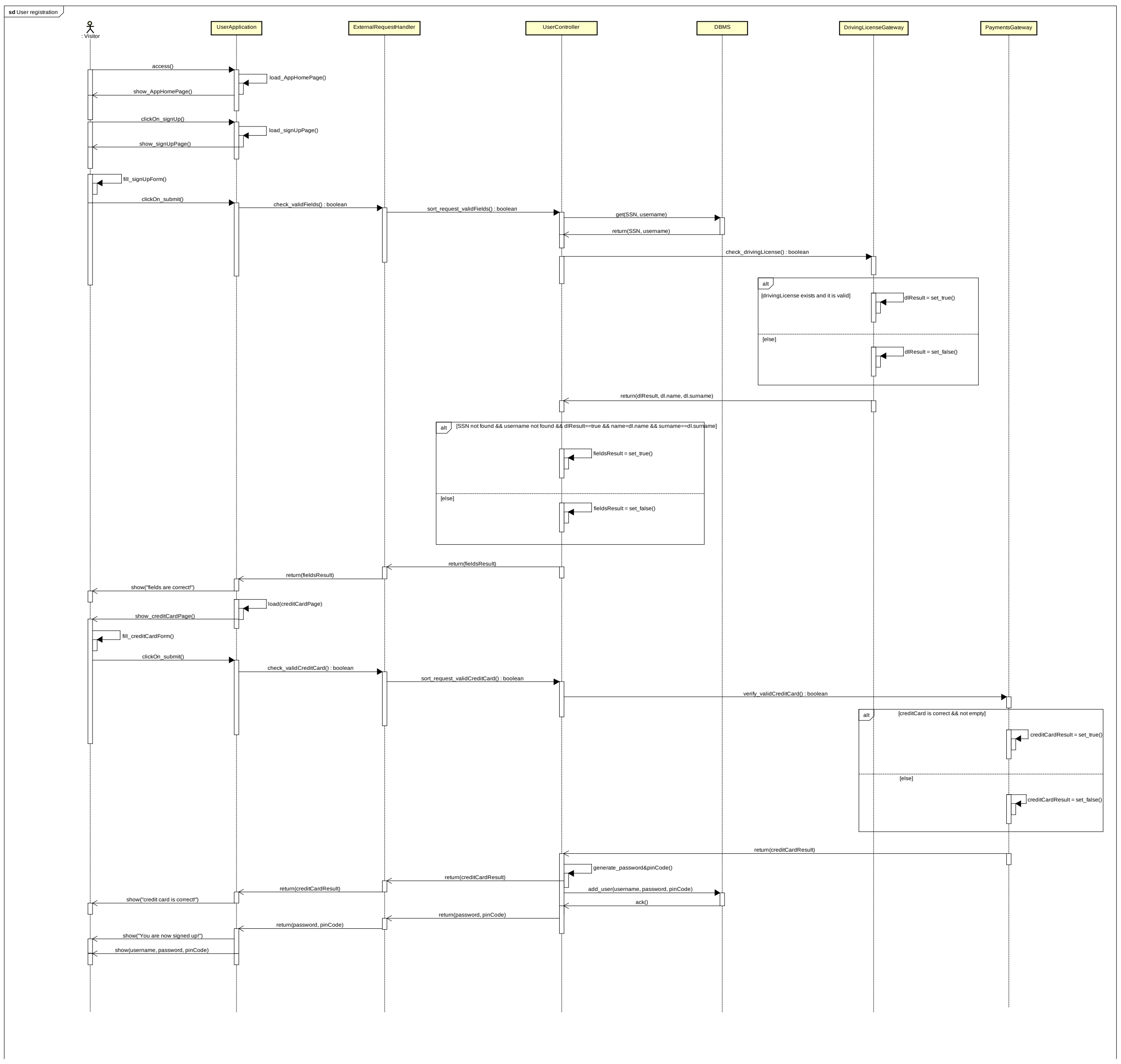
User	The user's app on the smartphone
Car	The car's application on board
DriverLicenceService	An external service that provides a database where the company could verify the correctness of the user information
PaymentService	An external service that provides a payment service
Database	The database used to store persistent data

2.3 Deployment view


pkg Deployment diagram
<<device>> : Mobille Phone
user's mobile App
<<device>>: System on board
car's App
<<device>> : JEE Server
<<execution environment>> : JEE Application
Enterprise Beans
<<device>>: database server
<<execution environment>> : RDBMS
Relational tables

2.4 Runtime view

2.4.1 User registration sequence diagram


sd User registration
: Visitor
UserApplication
ExternalRequestHandler
UserController
DBMS
DrivingLicenseGateway
PaymentsGateway
access()
load_AppHomePage()
show_AppHomePage()
clickOn_signUp()
load_signUpPage()
show_signUpPage()
fill_signUpForm()
clickOn_submit()
check_validFields() : boolean
sort_request_validFields() : boolean
get(SSN, username)
return(SSN, username)
check_drivingLicense() : boolean
alt
[drivingLicense exists and it is valid]
dlResult = set_true()
[else]
dlResult = set_false()
return(dlResult, dl.name, dl.surname)
alt
[SSN not found && username not found && dlResult==true && name=dl.name && surname==dl.surname]
fieldsResult = set_true()
[else]
fieldsResult = set_false()
return(fieldsResult)
return(fieldsResult)
show("fields are correct!")
load(creditCardPage)
show_creditCardPage()
fill_creditCardForm()
clickOn_submit()
check_validCreditCard() : boolean
sort_request_validCreditCard() : boolean
verify_validCreditCard() : boolean
alt
[creditCard is correct && not empty]
creditCardResult = set_true()
[else]
creditCardResult = set_false()
return(creditCardResult)
generate_password&pinCode()
return(creditCardResult)
return(creditCardResult)
add_user(username, password, pinCode)
show("credit card is correct!")
ack()
return(password, pinCode)
return(password, pinCode)
show("You are now signed up!")
show(username, password, pinCode)

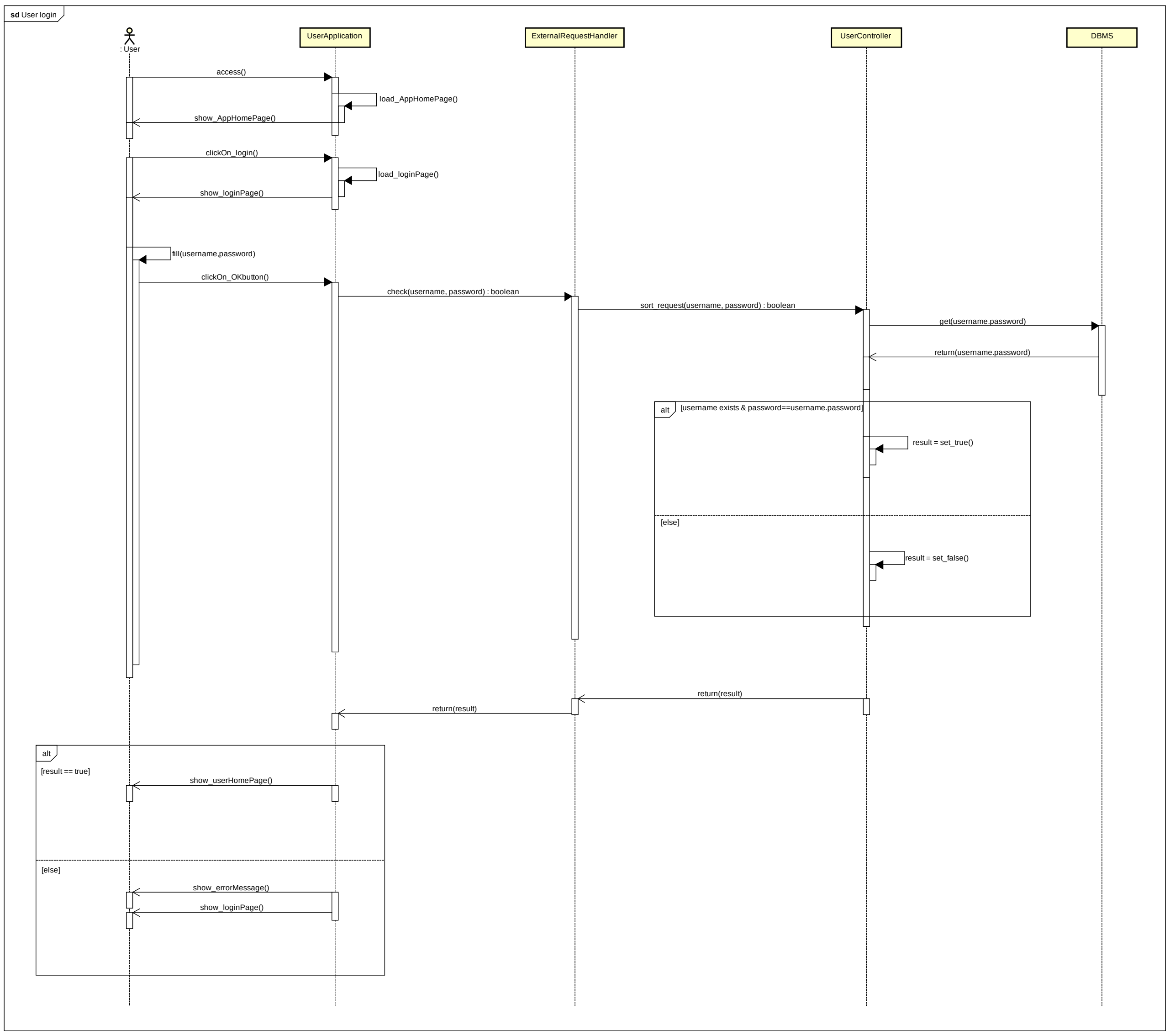
2.4.1 User registration description

In this sequence diagram, it can be seen what a visitor should do in order to become a user of PowerEnJoy application. Once the visitor has accessed the app, he/she has to fill the sign up form by entering his/her personal information and then click on “submit”. Then the user application transfers the request to the External Request Handler, which sorts it again to the User Controller. The User Controller asks the DBMS if the entered SSN already exists in the database (and it means that the user is already logged into the PowerEnJoy App) and if the username already exists (in this case, the user has to enter a different username).

The User Controller also asks the Driving License Gateway if there exists a driving license with the number entered by the user and if it is valid. Then the User Controller checks if the driving license’s owner corresponds to the user.

If it is all correct, the user can proceed with his/her registration by entering credit card’s details. The Payments Gateway has to check the validity of the credit card, and, if it is correct, the User Controller generates password and pin code that user should use in order to enter the application and the car, respectively. The DBMS is designated to add the user’s information (comprising username, password and pin code) to the DB.

2.4.2 User login sequence diagram

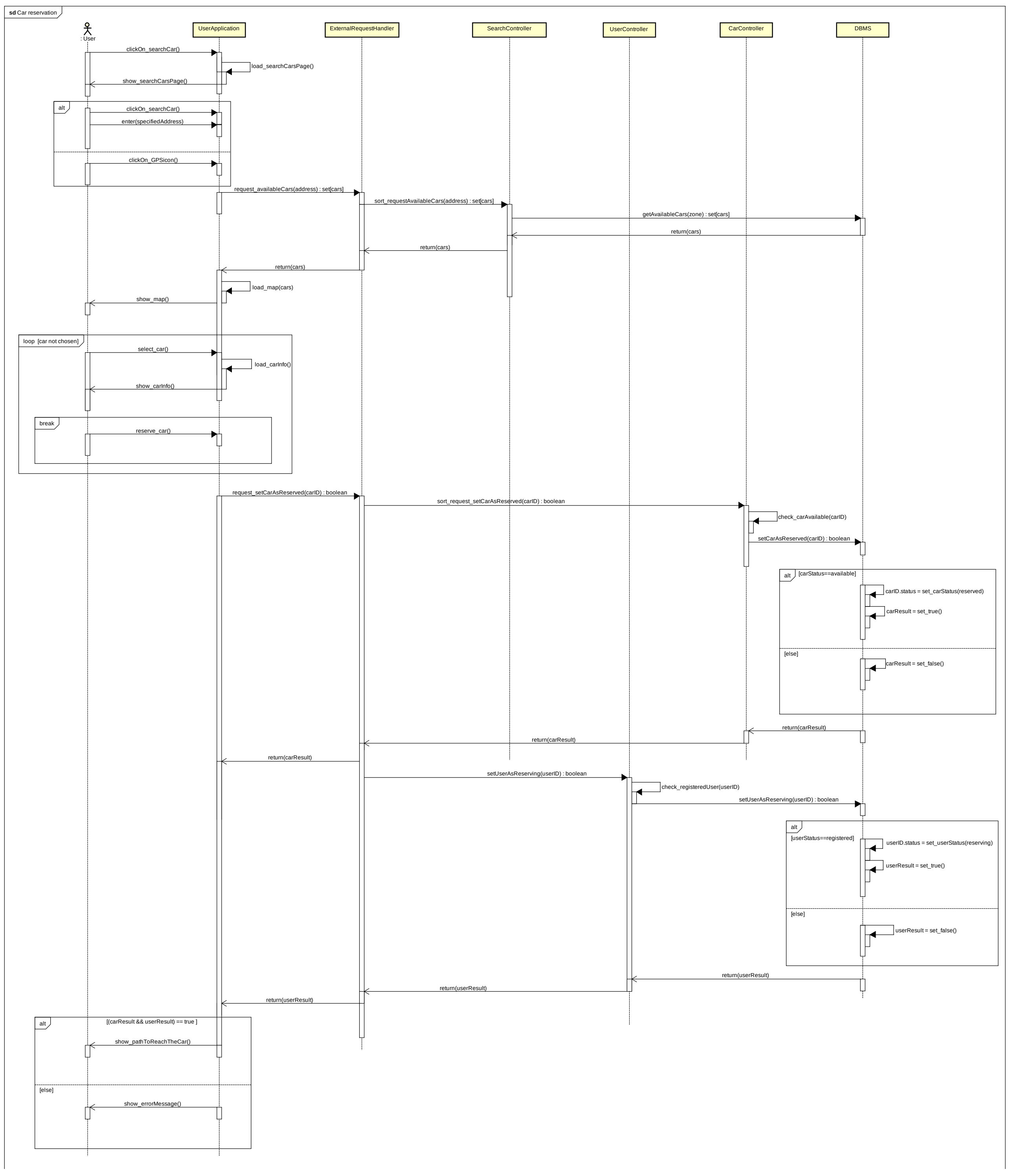


2.4.2 User login description

Once the user has accessed PowerEnJoy App, he/she can log in by entering his/her username and password. The External Request Handler receives the request from the user application and forwards it to the User Controller. This one asks the DBMS to provide the password of the entered username. Then the User Controller checks if the entered password is equal to the password stored in the database.

If username and password are correct, the user will see the user's home page; otherwise he/she will see an error message and the login page again, in order to re-enter correctly username and password.

2.4.3 Car reservation sequence diagram



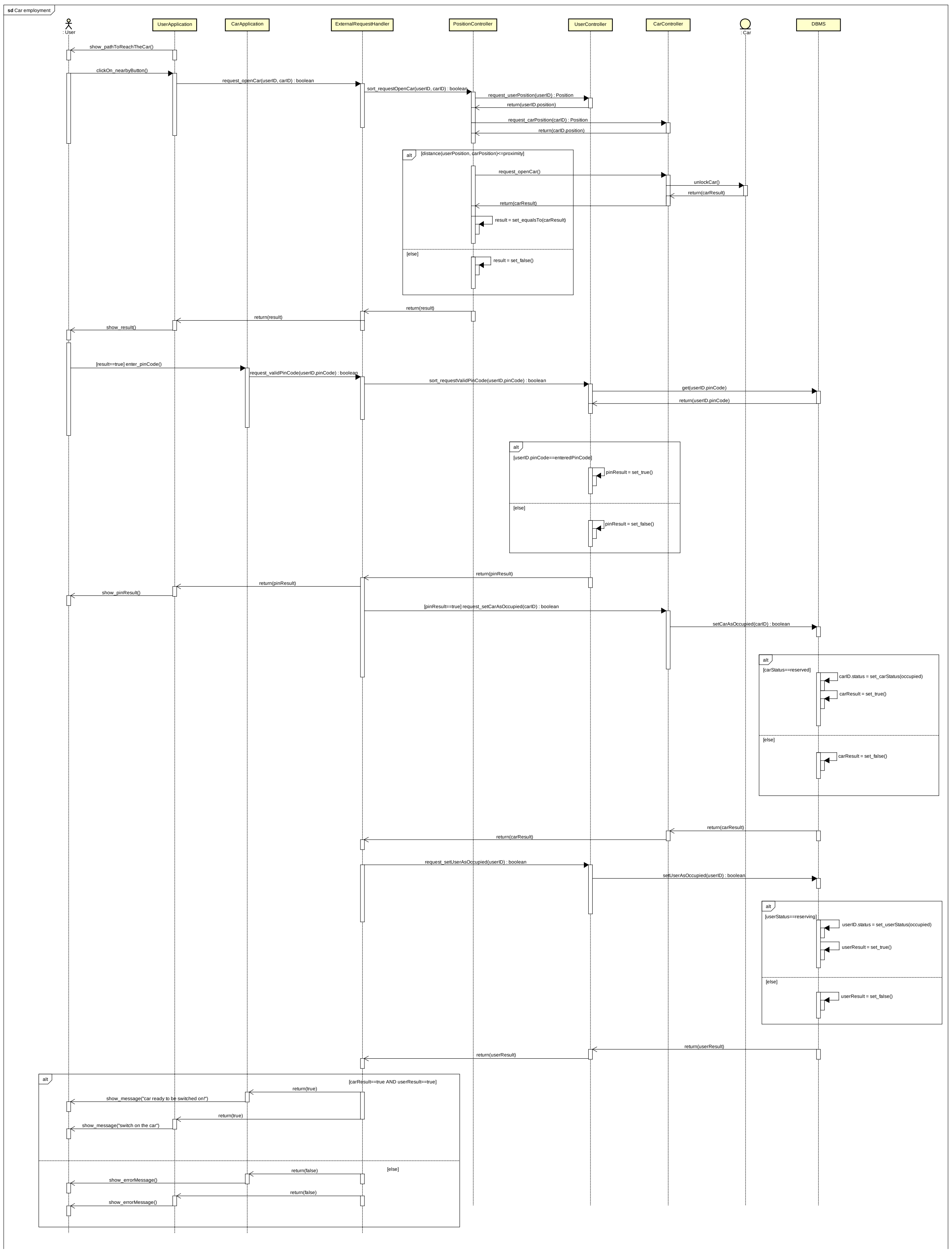
2.4.3 Car reservation description

Once the user has logged in the application, he/she can search for an available car. The user can enter a specified address or click on the GPS icon (in this way, the user's smartphone will localize the user and it will send the user's position to the system). The User Application requests a set of cars parked in that specific area; the request is forwarded to the Search Controller and then to the DBMS, which queries the DB in order to get all the cars parked in that zone. At this point the User Application receives the cars' positions, loads the map and shows it to the user.

Now the user can select a car's icon and the application will show him/her the car's information (such as car's ID, position, battery level and distance). As long as the user doesn't reserve a car, he/she can select all the cars displayed on the map. When the user reserves a car, the DBMS has to verify that the car's status is "available" and set it as "reserved". The DBMS also has to check if the user's status is "registered" and set it as "reserving".

If everything has been successful, the user will see the path to reach the car; otherwise he/she will see an error message.

2.4.4 Car employment sequence diagram



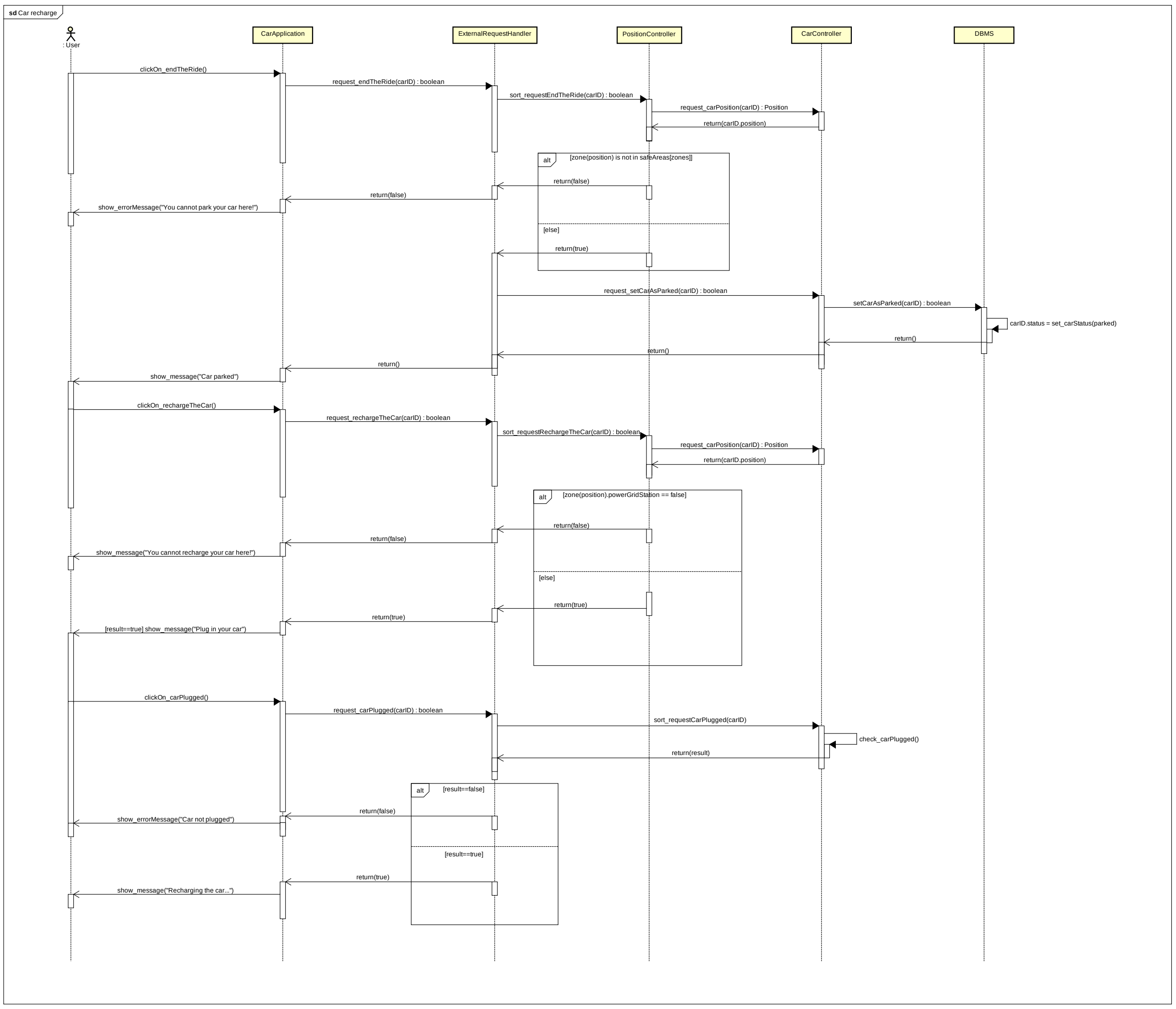
2.4.4 Car employment description

Once the user has reserved and reached the car, he/she should click on “I’m near the car” button. In this way, the User Application sends a request to the External Request Handler, in order to get the car unlocked. The request is sorted to the Position Controller, which gets the user’s position from the User Controller and the car’s position from the Car Controller. If the distance from the user to the car is less than the proximity value, the Position Controller asks the Car Controller to unlock the car.

Then, the user can enter the car and enter his/her pin code (provided by the system during the registration phase). The User Controller checks if the entered pin code is correct, and, if the user’s status is “reserving”, sets the status as “occupied”. In the same way, the Car Controller verifies that the car’s status is “reserved” and sets it as “occupied”.

If everything has been successful, the user can switch on the car; otherwise he/she will see an error message on the car’s screen.

2.4.5 Car recharge sequence diagram

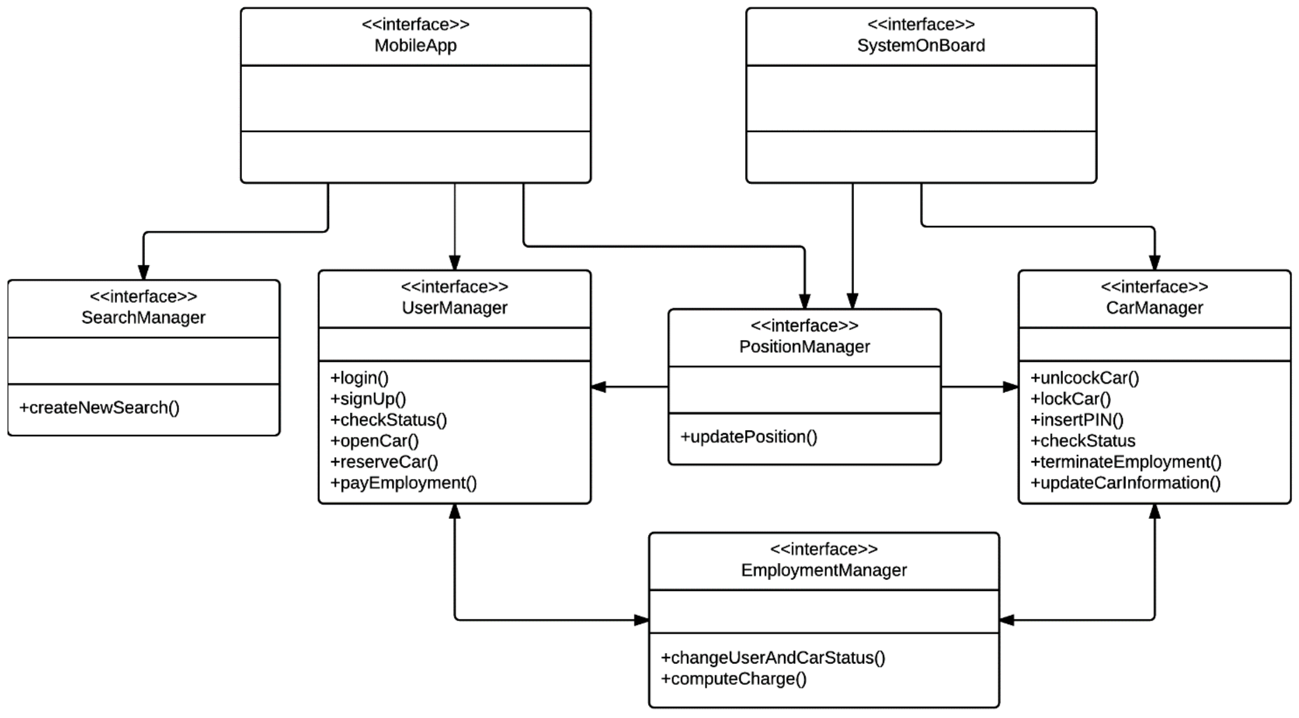

sd Car recharge
: User
CarApplication
ExternalRequestHandler
PositionController
CarController
DBMS
clickOn_endTheRide()
request_endTheRide(carID) : boolean
sort_requestEndTheRide(carID) : boolean
request_carPosition(carID) : Position
return(carID.position)
alt
[zone(position) is not in safeAreas[zones]]
return(false)
return(false)
show_errorMessage("You cannot park your car here!")
[else]
return(true)
request_setCarAsParked(carID) : boolean
setCarAsParked(carID) : boolean
carID.status = set_carStatus(parked)
return()
return()
return()
show_message("Car parked")
clickOn_rechargeTheCar()
request_rechargeTheCar(carID) : boolean
sort_requestRechargeTheCar(carID) : boolean
request_carPosition(carID) : Position
return(carID.position)
alt
[zone(position).powerGridStation == false]
return(false)
return(false)
show_message("You cannot recharge your car here!")
[else]
return(true)
return(true)
[result==true] show_message("Plug in your car")
clickOn_carPlugged()
request_carPlugged(carID) : boolean
sort_requestCarPlugged(carID)
check_carPlugged()
return(result)
alt
[result==false]
return(false)
show_errorMessage("Car not plugged")
[result==true]
return(true)
show_message("Recharging the car...")

2.4.5 Car recharge description

Once the user has ended the ride, he/she clicks the “end the ride” button on the car App. The Position Controller verifies that the car is parked in a safe area and the Car Controller sets the car’s status as “parked”.

Now the user can decide to recharge his/her car: in this way the Position Controller checks if the car is parked in an area equipped with charging points. Then the user can plug the car into a power grid and start recharging the car.

2.5 Component interfaces



2.6 Selected architectural styles and patterns

Car's App:

Aggregator
In order to create a screen view that displays different information from different sources, we have selected an Aggregator pattern. This pattern allows us to collect and store individual related messages, combine these messages, and publish a single aggregated message to the output channel. Thanks to this choice, the application on the car will be able to display car's information provided by the On-board computer, and online information provided by the central system.

Mobile App:

Request Response
Users can interact with PowerEnJoy system by accessing the PowerEnJoy mobile application. In order to be more responsive to each user's request, we have decided to implement a Request-Response Communication Pattern.
This pattern allows us to have a two-way message communication mechanism that permits clients to receive a response for every message they have sent.

Common Patterns:

Point-to-point Channel
To ensure that a message is sent to only one device (car's system or user's smartphone), we have chosen a communication based on a Point-to-Point Channel.

MVC
The Model View Controller pattern has been used in our project design in some parts; applications interact with view components and the internet communication will provide local updates to the system.

2.7 Other design decisions

We have decided to split the map in several zones, in order to simplify the search and the management of the distribution in the city. Each zone has a central power grid station easy to reach from each user, where there are some charging points in which the user can plug the car and recharge it. This decision could interfere with the request of the application of the 3km range discount, but we have thought that the distance is a simpler vision of the physiognomy of a real city, so three kilometers range should be applied to a single zone of the city.

3. ALGORITHM DESIGN

In this part we will explore the most relevant algorithm of the project.

3.1 Payment computation

```
class Employment(){
...
Car carEm;
Time StartUseTime, endUseTime;
float charge = 0;
float discount= 0;
...
```

```
/* the result of this function is displayed on the Car’s App during the ride, it computes the charge
without any discount. (millisecond sleep time) */
```

```
function runTimeCharge(){
    if(startUseTime){
        while !(endUseTime){
            begin;
            sleep(60000);
            charge = charge + * Constaints.MINUTEFEE;
            end;
        }
    }
}
```

```
/* the result of this function is displayed on the Car’s App at the end of the ride so it computes the
overall charge with any possible applicable discount. Hence this is the real charge that has to
payed from the customer */
```

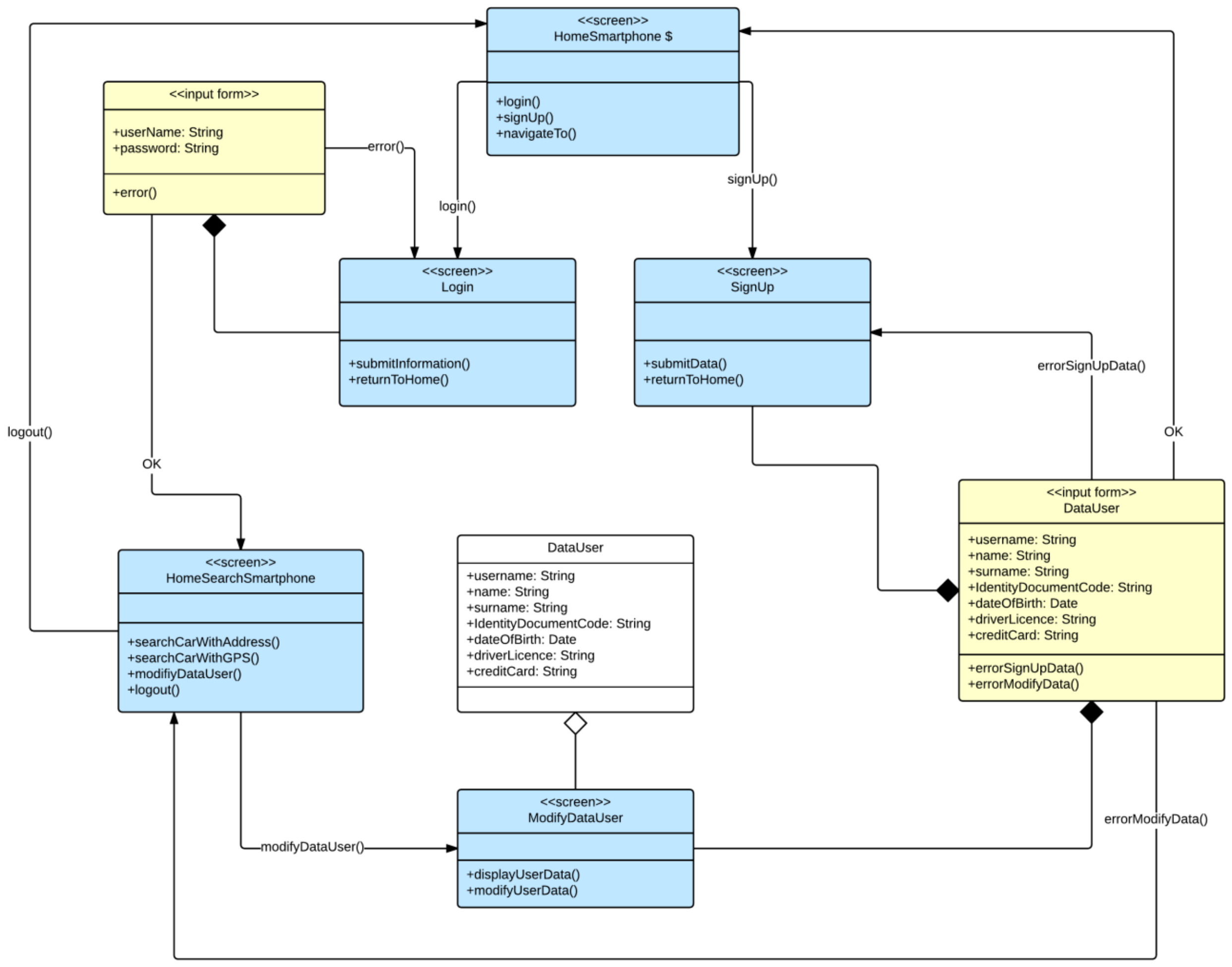
```
function calculateCharge(Map map){
    Zone currentZone = map.searchZoneByPosition(carEm.position)
    if(carEm.passenger>=2)
        discount= discount + 0,1;
    if(carEm.batteryLevel> 50)
        discount= discount + 0,2;
    if(carEm.charging= true)
        discount = discount + 0,3;
    if((distance(carEm.position, currentZone.powerGridStation) > 3000) ||
    (carEm.batteryLevel<20))
        discount = discount - 0,3;
    charge = charge - charge * discount;
}
```

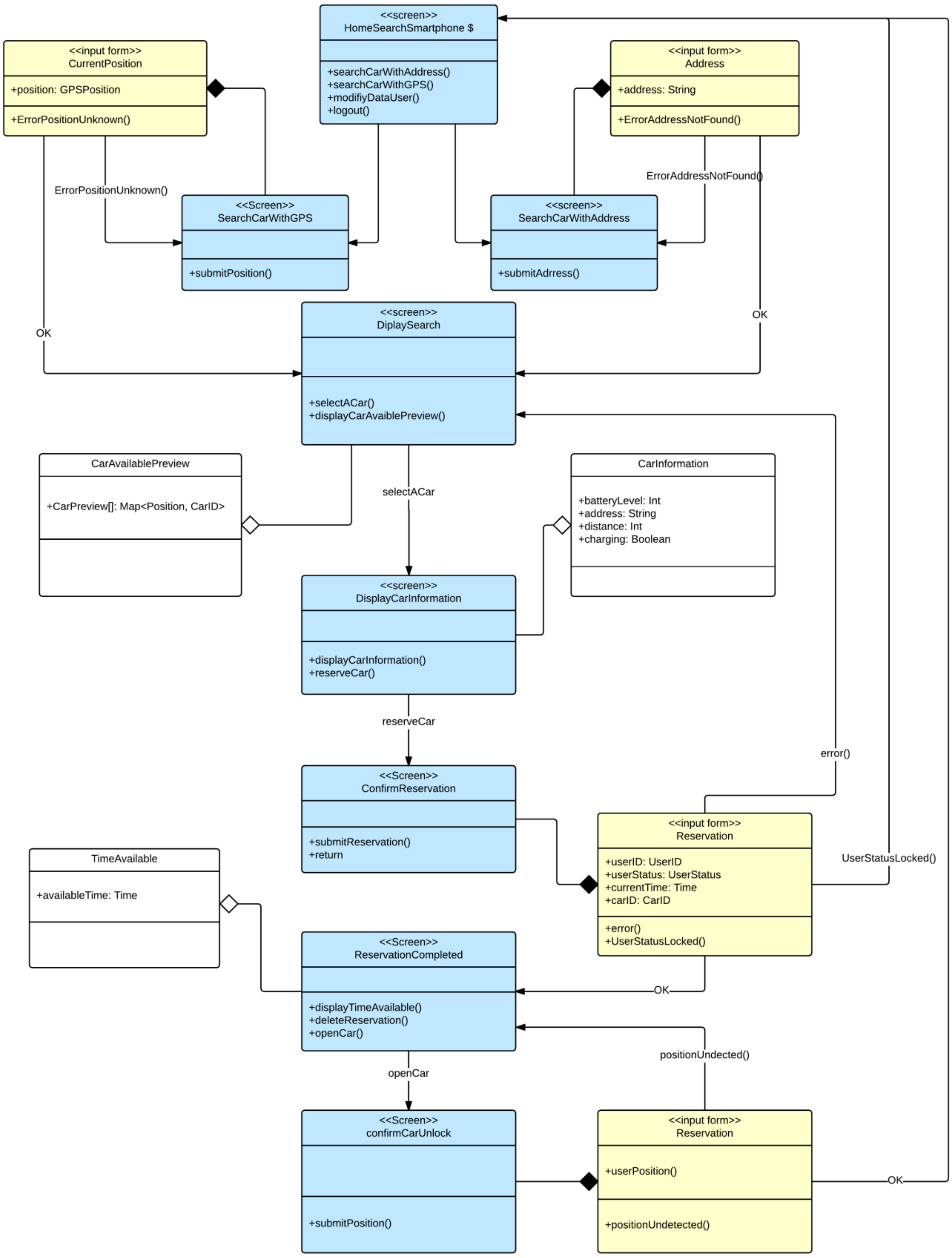
4. USER INTERFACE DESIGN

4.1 Mock-ups

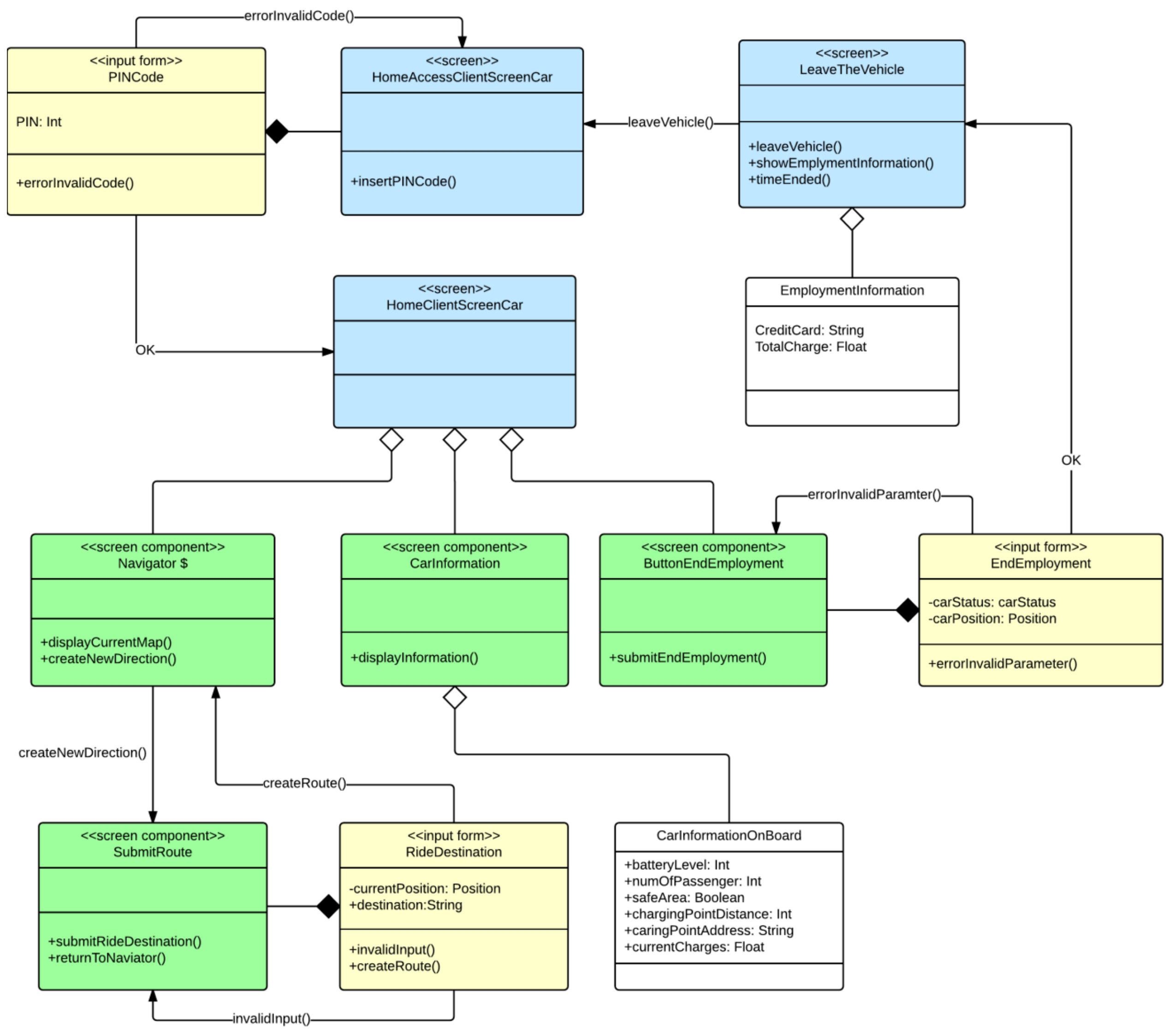
Mock-ups have already been showed in RASD section 5.

4.2 UX diagrams

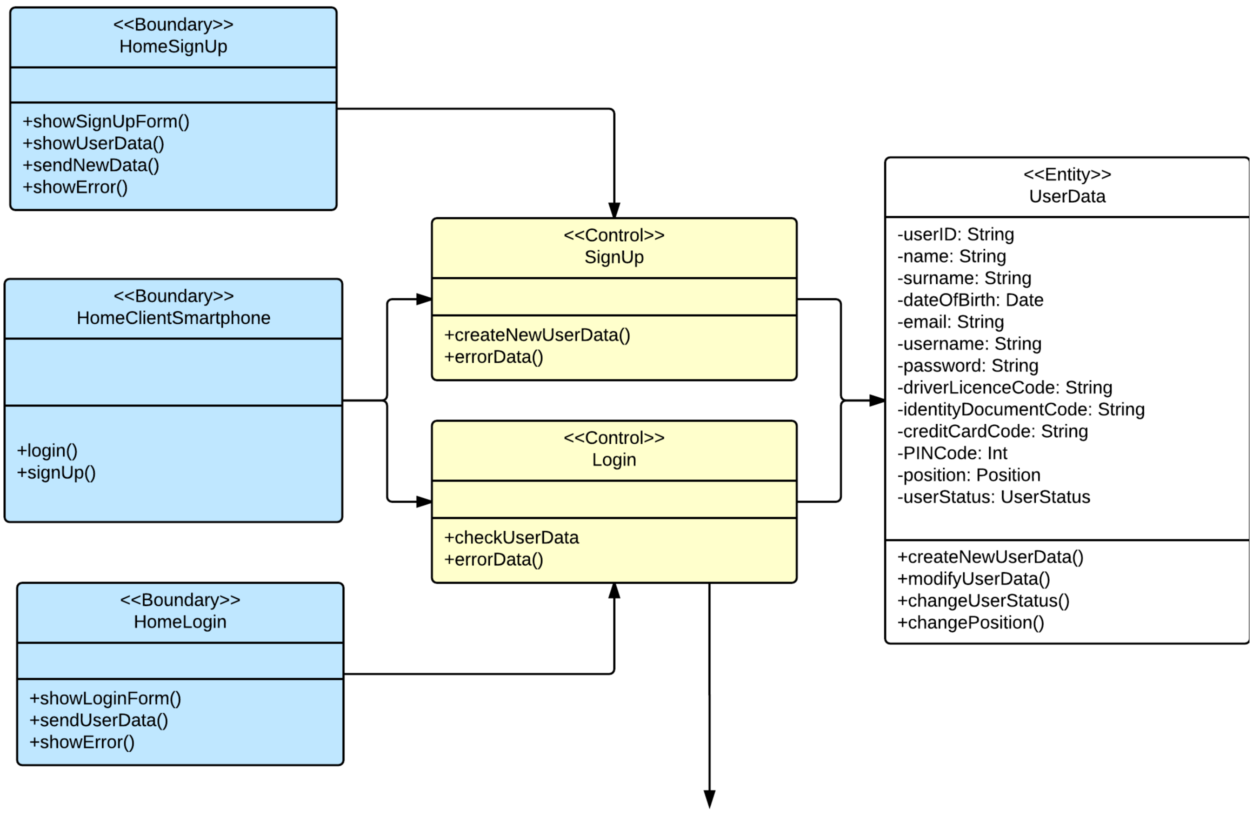


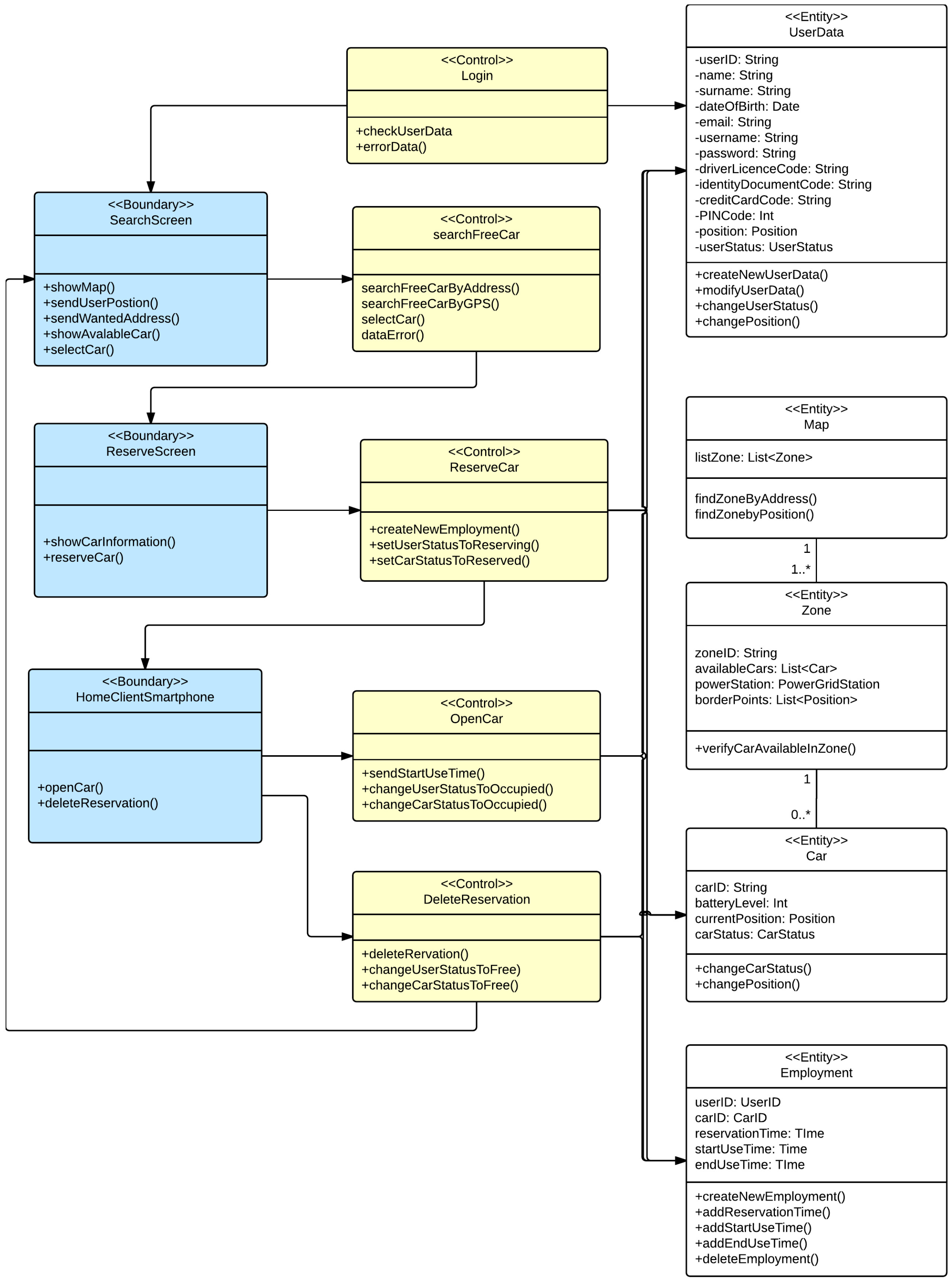


<<screen>> HomeSearchSmartphone $
+searchCarWithAddress()
+searchCarWithGPS()
+modifiyDataUser()
+logout()
<<input form>> CurrentPosition
+position: GPSPosition
+ErrorPositionUnknown()
<<input form>> Address
+address: String
+ErrorAddressNotFound()
ErrorPositionUnknown()
ErrorAddressNotFound()
<<Screen>> SearchCarWithGPS
+submitPosition()
<<screen>> SearchCarWithAddress
+submitAdrress()
OK
OK
<<screen>> DiplaySearch
+selectACar()
+displayCarAvaiblePreview()
CarAvailablePreview
+CarPreview[]: Map<Position, CarID>
selectACar
CarInformation
+batteryLevel: Int
+address: String
+distance: Int
+charging: Boolean
<<screen>> DisplayCarInformation
+displayCarInformation()
+reserveCar()
reserveCar
error()
<<Screen>> ConfirmReservation
+submitReservation()
+return
<<input form>> Reservation
+userID: UserID
+userStatus: UserStatus
+currentTime: Time
+carID: CarID
+error()
+UserStatusLocked()
UserStatusLocked()
TimeAvailable
+availableTime: Time
<<Screen>> ReservationCompleted
+displayTimeAvailable()
+deleteReservation()
+openCar()
OK
positionUndected()
openCar
<<Screen>> confirmCarUnlock
+submitPosition()
<<input form>> Reservation
+userPosition()
+positionUndetected()
OK

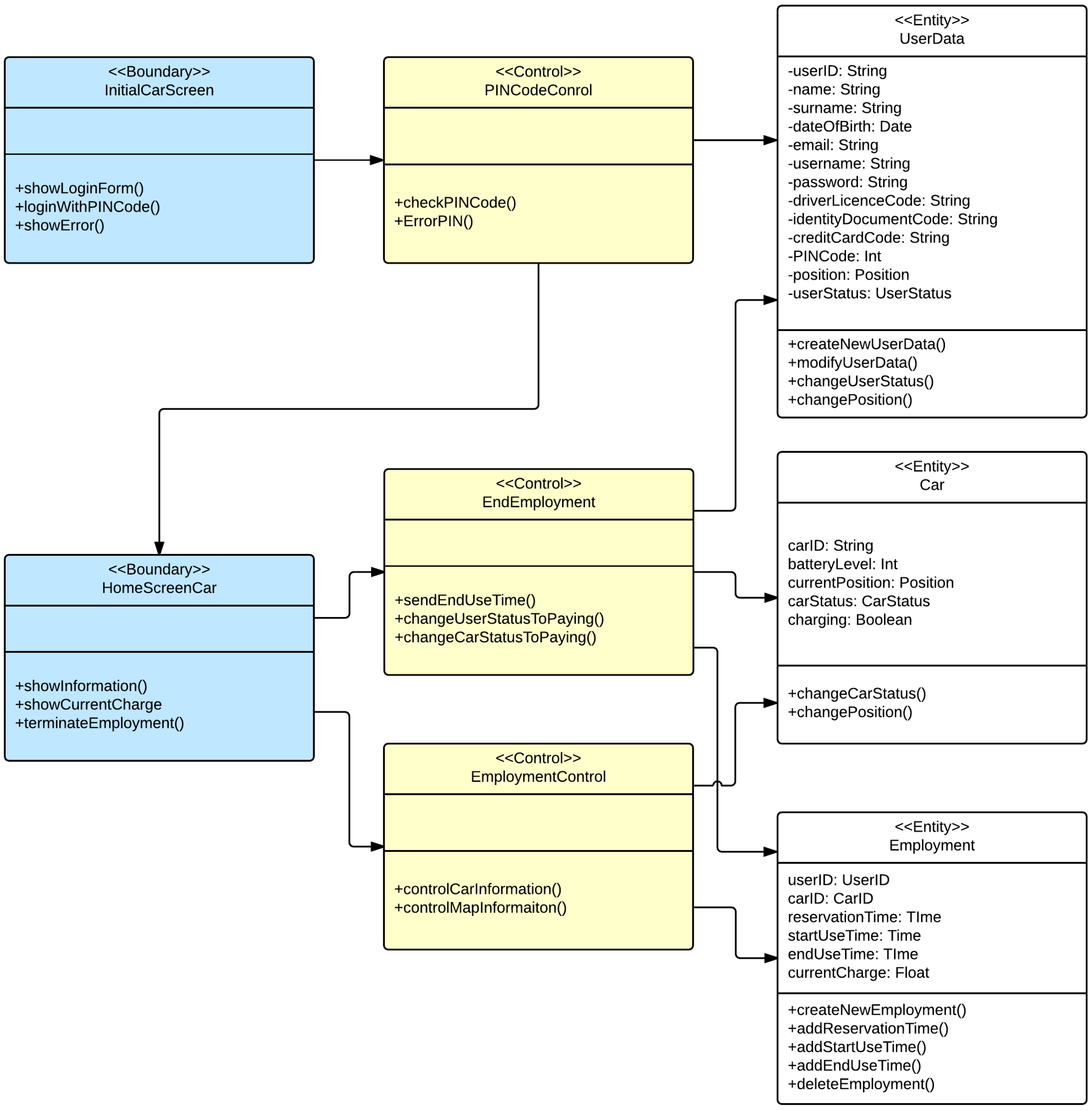

errorInvalidCode()
<<input form>>
PINCode
PIN: Int
+errorInvalidCode()
<<screen>>
HomeAccessClientScreenCar
+insertPINCode()
leaveVehicle()
<<screen>>
LeaveTheVehicle
+leaveVehicle()
+showEmplymentInformation()
+timeEnded()
EmploymentInformation
CreditCard: String
TotalCharge: Float
OK
<<screen>>
HomeClientScreenCar
OK
errorInvalidParamter()
<<screen component>>
Navigator $
+displayCurrentMap()
+createNewDirection()
<<screen component>>
CarInformation
+displayInformation()
<<screen component>>
ButtonEndEmployment
+submitEndEmployment()
<<input form>>
EndEmployment
-carStatus: carStatus
-carPosition: Position
+errorInvalidParameter()
createNewDirection()
createRoute()
<<screen component>>
SubmitRoute
+submitRideDestination()
+returnToNaviator()
<<input form>>
RideDestination
-currentPosition: Position
+destination:String
+invalidInput()
+createRoute()
CarInformationOnBoard
+batteryLevel: Int
+numOfPassenger: Int
+safeArea: Boolean
+chargingPointDistance: Int
+caringPointAddress: String
+currentCharges: Float
invalidInput()

4.3 BCE diagrams





<<Control>>
Login
+checkUserData
+errorData()
<<Entity>>
UserData
-userID: String
-name: String
-surname: String
-dateOfBirth: Date
-email: String
-username: String
-password: String
-driverLicenceCode: String
-identityDocumentCode: String
-creditCardCode: String
-PINCode: Int
-position: Position
-userStatus: UserStatus
+createNewUserData()
+modifyUserData()
+changeUserStatus()
+changePosition()
<<Boundary>>
SearchScreen
+showMap()
+sendUserPostion()
+sendWantedAddress()
+showAvalableCar()
+selectCar()
<<Control>>
searchFreeCar
searchFreeCarByAddress()
searchFreeCarByGPS()
selectCar()
dataError()
<<Entity>>
Map
listZone: List<Zone>
findZoneByAddress()
findZonebyPosition()
1
1..*
<<Boundary>>
ReserveScreen
+showCarInformation()
+reserveCar()
<<Control>>
ReserveCar
+createNewEmployment()
+setUserStatusToReserving()
+setCarStatusToReserved()
<<Entity>>
Zone
zoneID: String
availableCars: List<Car>
powerStation: PowerGridStation
borderPoints: List<Position>
+verifyCarAvailableInZone()
1
0..*
<<Boundary>>
HomeClientSmartphone
+openCar()
+deleteReservation()
<<Control>>
OpenCar
+sendStartUseTime()
+changeUserStatusToOccupied()
+changeCarStatusToOccupied()
<<Entity>>
Car
carID: String
batteryLevel: Int
currentPosition: Position
carStatus: CarStatus
+changeCarStatus()
+changePosition()
<<Control>>
DeleteReservation
+deleteRervation()
+changeUserStatusToFree)
+changeCarStatusToFree()
<<Entity>>
Employment
userID: UserID
carID: CarID
reservationTime: TIme
startUseTime: Time
endUseTime: TIme
+createNewEmployment()
+addReservationTime()
+addStartUseTime()
+addEndUseTime()
+deleteEmployment()



<<Boundary>>
InitialCarScreen
+showLoginForm()
+loginWithPINCode()
+showError()
<<Control>>
PINCodeConrol
+checkPINCode()
+ErrorPIN()
<<Entity>>
UserData
-userID: String
-name: String
-surname: String
-dateOfBirth: Date
-email: String
-username: String
-password: String
-driverLicenceCode: String
-identityDocumentCode: String
-creditCardCode: String
-PINCode: Int
-position: Position
-userStatus: UserStatus
+createNewUserData()
+modifyUserData()
+changeUserStatus()
+changePosition()
<<Boundary>>
HomeScreenCar
+showInformation()
+showCurrentCharge
+terminateEmployment()
<<Control>>
EndEmployment
+sendEndUseTime()
+changeUserStatusToPaying()
+changeCarStatusToPaying()
<<Entity>>
Car
carID: String
batteryLevel: Int
currentPosition: Position
carStatus: CarStatus
charging: Boolean
+changeCarStatus()
+changePosition()
<<Control>>
EmploymentControl
+controlCarInformation()
+controlMapInformaiton()
<<Entity>>
Employment
userID: UserID
carID: CarID
reservationTime: TIme
startUseTime: Time
endUseTime: TIme
currentCharge: Float
+createNewEmployment()
+addReservationTime()
+addStartUseTime()
+addEndUseTime()
+deleteEmployment()

5. REQUIREMENTS TRACEABILITY

Given that the design document has to fulfil the requirements expressed in the RASD, here there is a list of connections between goals and design elements of PowerEnJoy system.

- [G2] People who have downloaded the app must be able to sign up
  - User application
  - External request controller
  - User controller
  - Driving license gateway
  - Payments gateway

- [G3] People who have already signed up must be able to log in
  - User application
  - External request controller
  - User controller

- [G4] Users must be able to modify their personal information
  - User application
  - External request controller
  - User controller

- [G5] Users must be able to search all the available cars within a certain distance from their current location or from a specified address
  - User application
  - External request controller
  - User controller
  - Car controller
  - Position controller
  - Search controller

- [G7] Users must be able to reserve a car
  - User application
  - External request controller
  - User controller
  - Car controller

- [G9] [G11] Users must be able to enter the car they have reserved and drive it
  - User application
  - External request controller
  - User controller
  - Car controller
  - Position controller

- [G12] Users must be notified of their current charges during their rides
  - User application
  - External request controller
  - User controller
  - Employment controller

- [G14] [G15] [G16] Users must be able to park their rented car in a safe area, end their ride and pay
  - User application
  - External request controller
  - User controller
  - Car controller
  - Position controller
  - Employment controller
  - Payments gateway

6. EFFORT SPENT

Giacomo Bossi:

20/11: 2hrs	preparatory meeting
22/11: 2hrs	preparatory meeting
25/11: 3hrs	preparatory meeting
30/11: 3hrs	preparatory meeting
01/12: 2hrs	
02/12: 4hrs	
03/12: 3hrs	
04/12: 2hrs	
05/12: 3hrs	
06/12: 3hrs	
07/12: 1hr	
10/12: 2hrs	

Nanni Marco:

20/11: 2hrs	preparatory meeting
22/11: 2hrs	preparatory meeting
25/11: 3hrs	preparatory meeting
30/11: 3hrs	preparatory meeting
01/12: 3hrs	
02/12: 3hrs	
03/12: 3hrs	
04/12: 2hrs	
05/12: 4hrs	
07/12: 5hrs	
11/12: 1hr	

7. REFERENCES

7.1 Used tools

- LucidChart for component diagrams, UX diagrams, BCE diagrams
- Astah Professional for deployment diagram, sequence diagrams