Politecnico di Milano

Industrial and Information Engineering
Computer Science and Engineering

Software Engineering II Assignment
PowerEnJoy - car sharing

Academic Year: 2016/2017

Prof. Elisabetta Di Nitto

Alice Segato matr. 875045
Mark Edward Ferrer matr. 876650
Davide Bonacina matr. 876199

PowerEnJoy

Car Sharing App

TABLE OF CONTENT

LIST OF FIGURES

1 INTRODUCTION

1.1 Purpose

This document has the purpose to give more information about the PowerEnJoy System than the Requirement Analysis and Specification Document (RASD).

This document addresses developers and has the objective to identify:

- The architectural design;
- The design choices that we have made;
- How the system components interface with each other;
- The behavior of the system at runtime.

1.2 Scope

The system has the purpose of allowing users and more in general citizens to rent cars easily via mobile app in order to increase people's mobility and decrease city pollution.

To use the application and the service that comes with it, people has to register and join a community of car sharers; after registration, they can rent cars and drive wherever they want to go with the condition to bring back the car in a safe area.

Users can choose a car from the nearest parking suggested by the app based on their GPS position or on the given address, and they can pick it up with a limited time.

Anytime during the driving, users can set the car in pit stop mode to park the car in a sort of "reserved" state where the car is still linked to the user but it is stopped and parked outside of the safe areas.

At the end of the ride, users must return the car in a safe area and the system calculates the final amount with respect to certain situations that give user discounts or surcharges: to help users with low budget, there's also a money saving option that calculates the nearest special parking area to the final destination of the user to get the maximum amount possible of discounts.

Therefore, the final objective of the system is to allow the company to manage the car requests faster and automatically to substitute the previous system, described in section 1.2 of the RASD.

1.3 Definitions, Acronyms, Abbreviations

All the words defined in the RASD at section 1.5 are still valid and they will appear in this document too.

- DD: Design Document.
- RASD: Requirements analysis and Specification Document.
- DB: the database layer, handled by a DBMS.
- UI: User Interface.
- GUI: graphical user interface is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators;

- Application server: the layer, which provides the application logic and interacts with the DB and with the front-ends.
- Back-end: term used to identify the Application server.
- Front-end: the components, which use the application server services, namely the web front-end and the mobile applications.
- Web server: the component that implements the web-based front-end. It interacts with the application server and with the users' browsers.
- MVC: Model-View-Controller.
- JDBC: Java Database Connectivity.
- Java Server Faces (JSF): is a Java specification for building component-based user interfaces for web applications
- XML: Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.
- JAX-RS: Java API for RESTful Web Services (JAX-RS) is a Java programming language API spec that provides support in creating web services according to the Representational State Transfer (REST) architectural pattern.
- JSON: JavaScript Object Notation is an open-standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs.
- HTTP: The Hypertext Transfer Protocol is an application protocol for distributed, collaborative, hypermedia information systems
- EJB: Enterprise JavaBean.
- ACID: Atomicity, Consistency, Integrity and Durability.
- PHP: PHP is a server-side scripting language designed primarily for web development but also used as a general-purpose programming language.
- ODBC drivers: Open Database Connectivity (ODBC) is a standard application programming interface (API) for accessing database management systems (DBMS).
- Acknowledge: is a signal passed between communicating processes or computers to signify acknowledgement, or receipt of response, as part of a communications protocols.
- Swift: Swift is a general-purpose, multi-paradigm, compiled programming language developed by Apple Inc. for iOS
- UIKit framework: UIKit is a high-level Objective-C framework that manages the graphical front-end of the iPhone OS
- Android.java package: Android application package (APK) is the package file format used by the Android operating system for distribution and installation of mobile apps and middleware.

1.4 Reference Documents

- RASD;
- Assignment AA 2016-2017.pdf;
- Sample Design Deliverable on Nov 2.pdf.

1.5 Document Structure

- **Introduction:** this section opens the document and shows the main purpose of the system-to-develop, the structure of the entire document and deepens some aspects introduced in the RASD.

- **Architecture Design:**
    - Overview: this section illustrates the physical deployment of our system;
    - High-level components and their interactions: shows how the different components of the system interface with each other and with the third-party system;
    - Component view: deepens the view of the components and gives more details;
    - Runtime view: this section explains with sequence diagrams, how the system should work during different tasks;
    - Component Interfaces: presents the communication of the different components;
    - Selected architectural designs and patterns: this section describes all the design patterns that we used to model the system and how they work.
    - Other design decisions: the title is self-explaining.
- **Algorithms Design:** this section includes some algorithms that manages particular tasks of the system. The algorithms are written in pseudo-code in order to clarify the behind the scenes of system with the maximum readability.
- **User Interface Design:** it should include design mockups to describe the possible result of the mobile application.
- **Requirements Traceability:** shows where the goals defined previously in the RASD take shape in design elements.

2 ARCHITECTURAL DESIGN

2.1 Overview

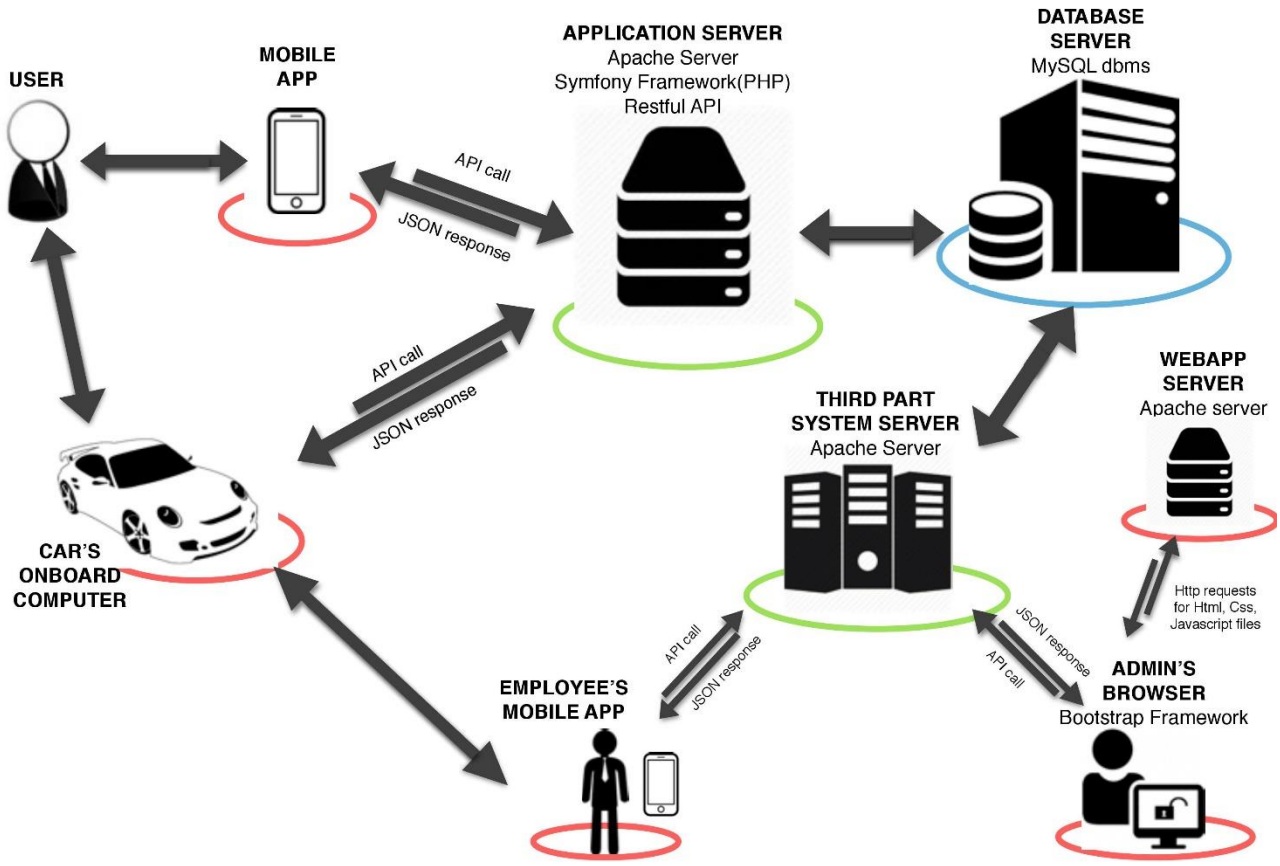

Figure 1

Caption:

Red: Tier 1 – Presentation

Green: Tier 2 – Business logic

Blue: Tier 3 – Data Manipulation

As we can see our PowerEnJoy system has a 3-tier architecture:

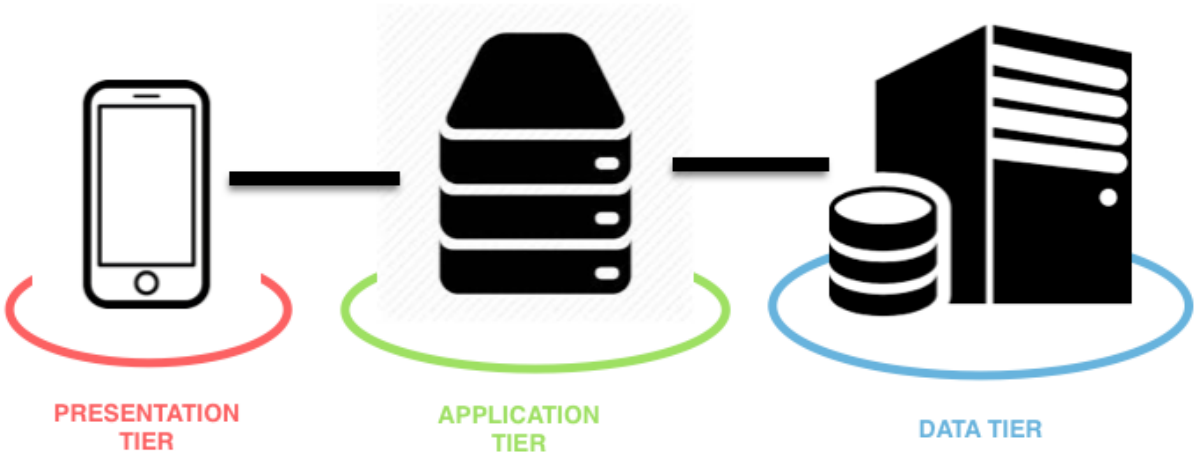

Figure 2

THE PRESENTATION TIER: gives a user access to the application. This layer shows data to the user and optionally permits data input. The two main types of user interface for this layer are the traditional application and the Web-based application (used by the admin of third party system).

THE APPLICATION LAYER: It contains all the logical part of the application; it can also communicate with the Database for the retrieval or the insertion of Data.

THE DATA LAYER: This is the actual DBMS access layer. It can be accessed through the application layer. This layer consists of data access components to aid in resource sharing and to allow clients to be configured without installing the DBMS libraries and ODBC drivers on each client.

At this point, we would like to point out some important observations:

- our server is going to be used by many different clients that we do not have control over;
- we want also to be able to update the server regularly without needing to update the client software.

So we need to minimize the coupling between client and server components in this application. In order to cover this requirements, we decided to use **RESTful API**. RESTful web services, as the name suggests, are resources on the web that can be used to get specific information. These services basically portray the working of the REST API. The client requests a resource from the server and the server sends back the response.

In this way on the Client there will be not a static GUI but a dynamic one that is generated on client side.

2.2 High level components and their interaction

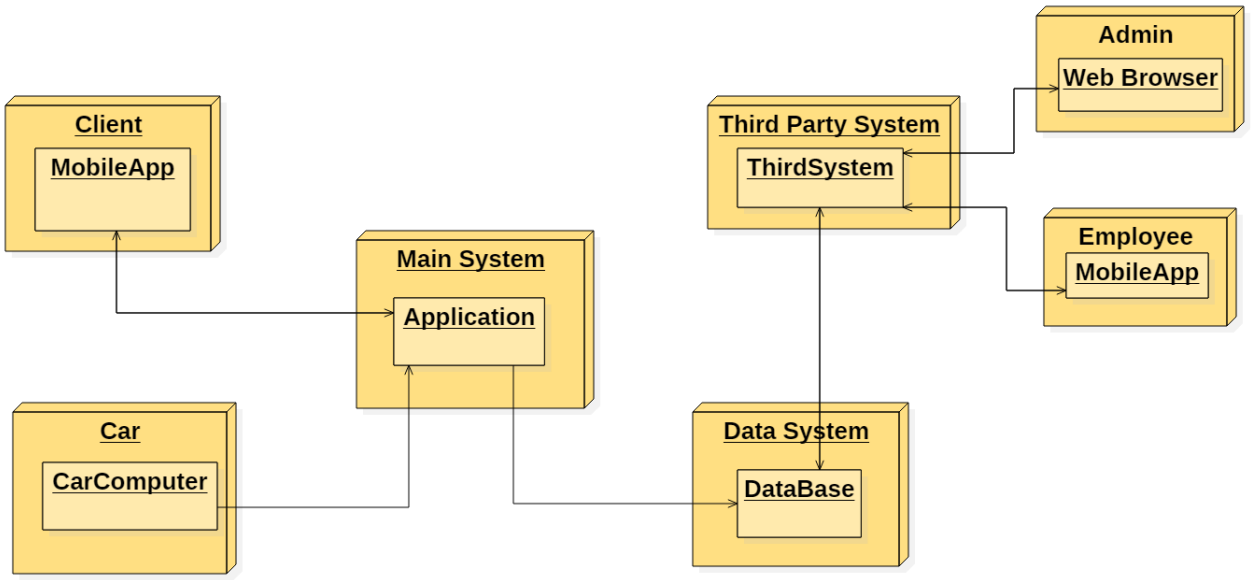

Figure 3

The high-level components architecture is composed of seven different elements. The main element is a singleton, the central called "**Main System**". The Main System receives requests of reservations from another element: the Client. Clients can initiate this communication only from mobile application. This communication is made in asynchronous way since the client, that initiates the communication, has to wait the answer of the main System that acknowledges him that his request has been taken into account. In this way, according to the request of the client, System provides him, for example, after search the availability of the car, all the possible parking areas where to pick the car up.

The main System will also send an asynchronous message to the client in the form of email to inform about, for example, the receipt of the booking (with the place and the number of the car) or the receipt of the bill for the payment.

The main System communicates also with the car: after the user tries to unlock the car, the main system provides user's credentials, in this way the **car computer** will recognize him and unlock the car, starting charging his bill. The communication between the main System and the car Computer is also asynchronous; in this way, the car sends to the main System all the feedbacks about its state and position, so the main system will know all the information about all cars.

The main system communicates also with a third type of component, the Third Part System through the database.

The last type of component is the database. The database still manages registration of new users, car states, quantities of cars and plugs, information about car distribution in the city, information about user's receipt, account information, trips, booking etc. Therefore, the main System and the Third part system communicates synchronously with the old database to extract the right information when needed.

2.3 Component view

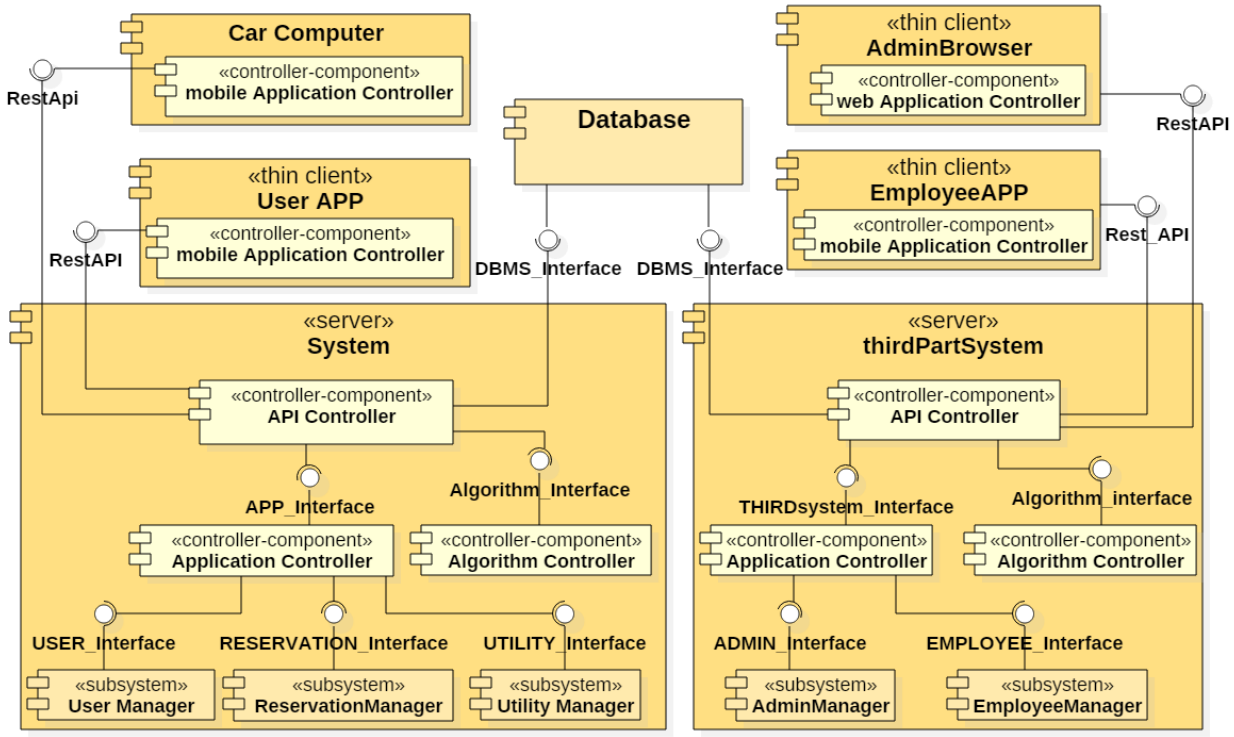

Figure 4

In the following component diagram the architecture of the system has been expanded. In the diagram, each application of the system (System, third part system, user app, employee app, car computer and admin browser) is represented, together with its own components. Moreover, for the sake of clarity, some subsystems have been introduced, to gather logically similar components. Each component may offer or require some interfaces.

2.3.1 Main System

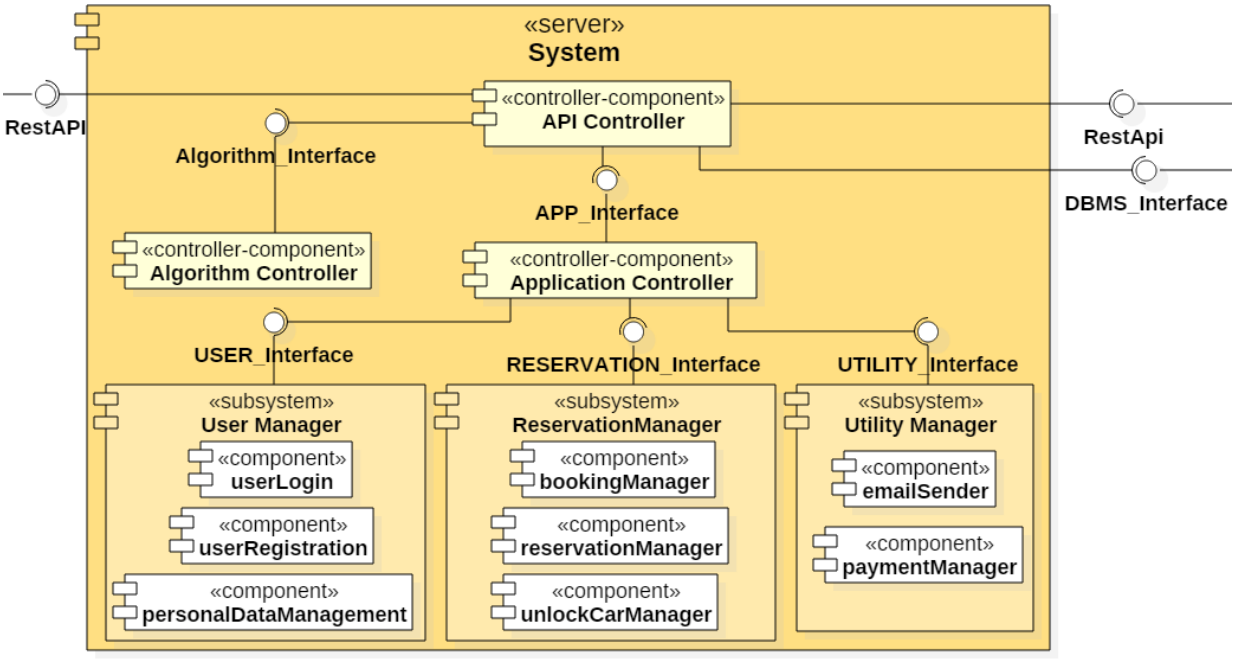

Figure 5

Application Server
The business logic is implemented in the application server tier using Java EE; it runs on Apache Server. The access to the DBMS is not implemented with direct SQL queries: instead, it is completely wrapped by the Java Persistence API. The business logic is implemented by custom-built stateless Enterprise JavaBeans (EJB). The application server implements a RESTful API using JAX-RS to allow the clients (web tier and mobile client) to use the services offered by the EJBs.

API Controller
The API controller replicate many functionalities of the Application Controller, written over. Even there is a duplication of some functionalities, we decided to maintain this controller because we wanted a clear separation between the API and Application. The API controller expose the DBMS and other components of the application to external requests. In case of a car's book requests (for example) it allow the algorithm to communicate with the DBMS. In case of a mobile use, it makes possible the data exchange from DBMS and application controller.

Application Controller:
The application controller, manage the communication between clients and internal components. It provides the ability to exchange data from an internal component to a client (and so from a client to the system). This interface is unique and does not depend by other components that are going to use it. It dispatches the message coming from the client to the right internal component. It manages the ride requests and ride bookings, by call to DBMS and Algorithm.

Algorithm Controller
The main Algorithm controller has to provide a communication between the algorithm pool, and the processes that need one of them. This component contains all the logic for the execution of various tasks that need a complex algorithm, such as the good distribution of the cars with the money saving option.

User Manager
This component manages all the user management features, namely: user login, user registration, user deletion, user profile editing. It also provides a function to confirm the email address provided by the user with the token sent by email, to see and change the user information and the method of payment.

Reservation Manager
This component manages all the requests of booking. It searches if there are available vehicles, it checks, with the help of the Algorithms component, the nearest parking with available cars, it gives the possibility to book a car. It manages also all the information about the reservation of a vehicle for example the state of the car, where it is placed, how much time the user has to reach the car before that the reservation will expire, and finally manages also the unlocking of the car.

Utility Manager
This component manages the sending of some notification throughout the email and the subtraction of money for the trip.

Database

Figure 6

The database tier runs MySQL Community Edition: the DBMS has to support transactions and ensure ACID properties. The DBMS will not be internally designed because it is an external component used as a "black box" offering some services: it only needs to be configured and tuned in the implementation phase. The database can communicate only with the business logic tier using the standard network interface. Security restrictions will be implemented to protect the data from unauthorized access: the database must be physically protected and the communication has to be encrypted. Access to the data must be granted only to authorized users possessing the right credentials. Every software component that needs to access the DBMS must do so with the minimum level of privilege needed to perform the operations. All the persistent application data are stored in the database. The conceptual design of the database is illustrated by this E-R diagram:

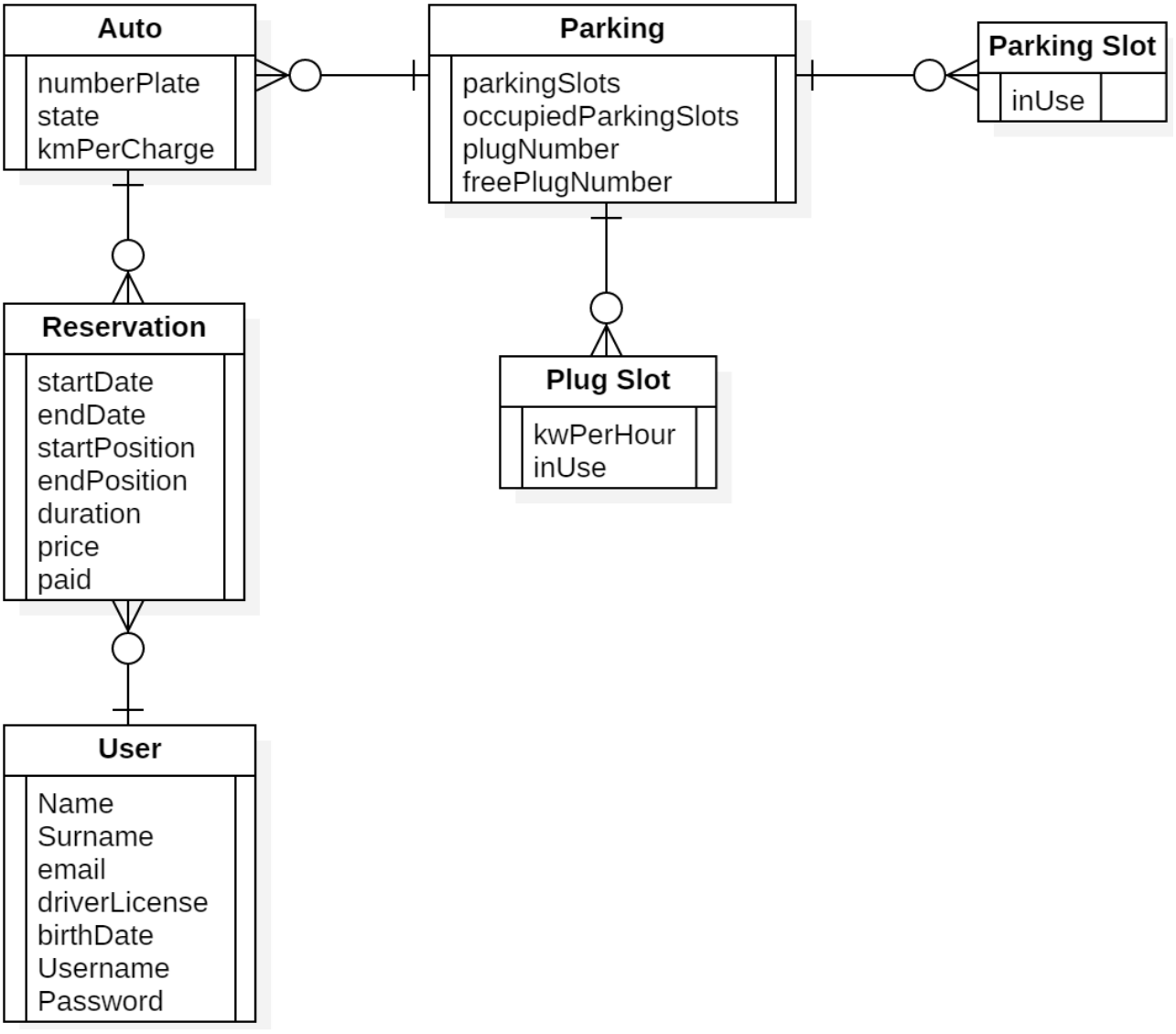

Figure 7

User App

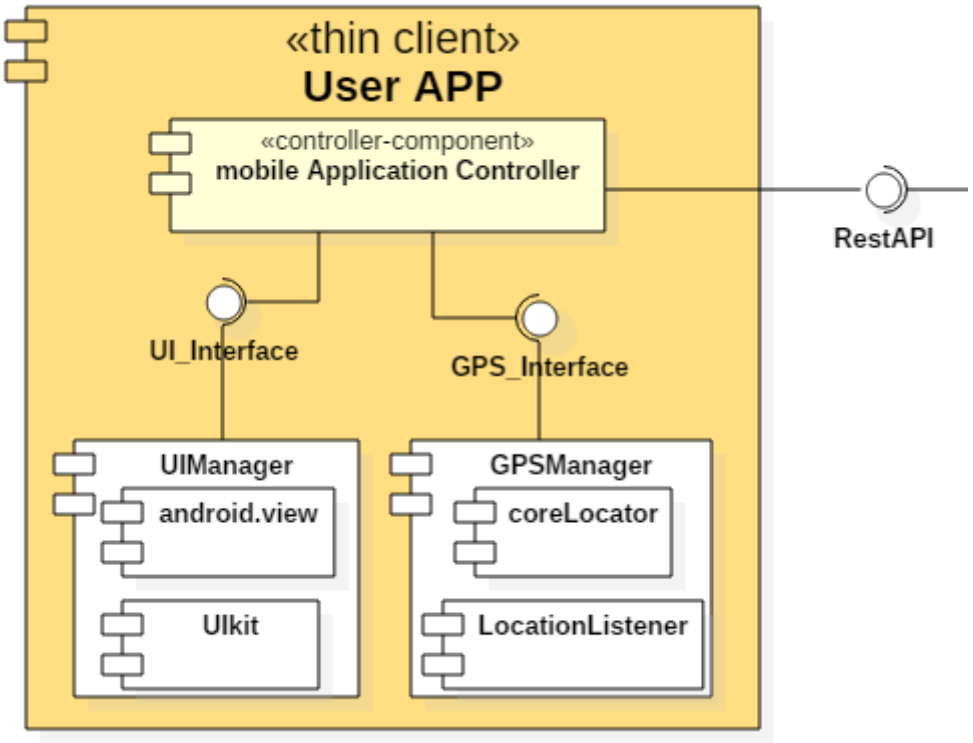

Figure 9

The user APP implementation depends on a specific platform. The iOS application is implemented in Swift and mainly uses UIKit framework to manage the UI interface. Instead, the Android application is implemented in Java and mainly uses android.view package for graphical management. The application core is composed by a **Mobile application controller** which translates the inputs from the UI into remote functions calls via RESTful APIs. The controller also manages the interaction with the GPS component using Core-Location framework in iOS app and LocationListener interface in the Android one.

Mobile Application Controller

This is the main controller for the mobile application. It has to show to user the correct view and correct data, in order to make a best fruition of the service in mobile. Via this controller can be made ride request and all the other ride managements in a mobile environment. It has also the charge of retrieve the GPS coordinates, and send them to server.

Car Computer

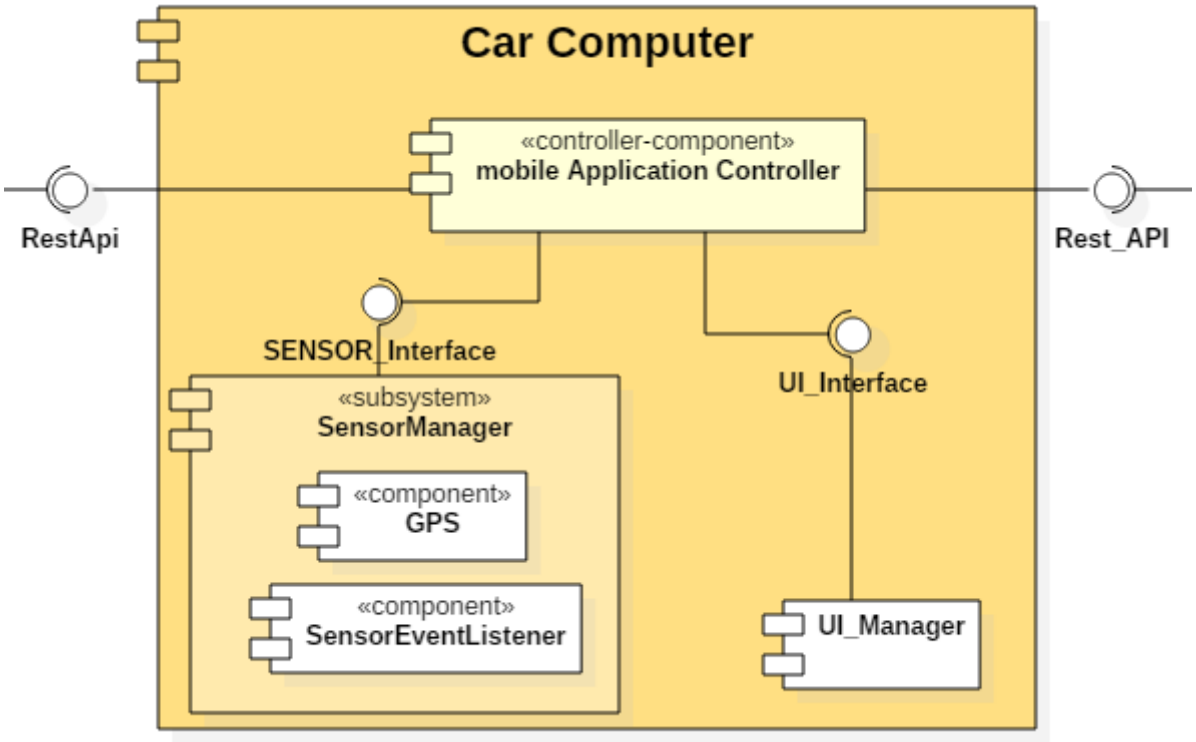

Figure 10

The Car computer is seen as a device comparable to a tablet or a smartphone, so with the same architectural structure. It is saw as a thin Client so on it there is a component that is equal with the Mobile Application Controller that we have just seen in the User APP, in facts it works based on the same principles. We also assumed that this component sends all the information retrieved from car sensors via CAN-bus technology that allows the gathering of all detections in a single bus and then dispatched to the main system.
The application that reads the message sent from the CAN-bus is a third party product and we will not specify how it works in this document, the only important thing is that it will send data to the employee application for cars administration purpose.

2.3.2 Third Part System

As we can see the two system as the same structure, we use also the same controller component for the same task and the same logic. As we can see the two systems communicate only throughout the DBMS. We will focalize only on the different parts:

Application Server

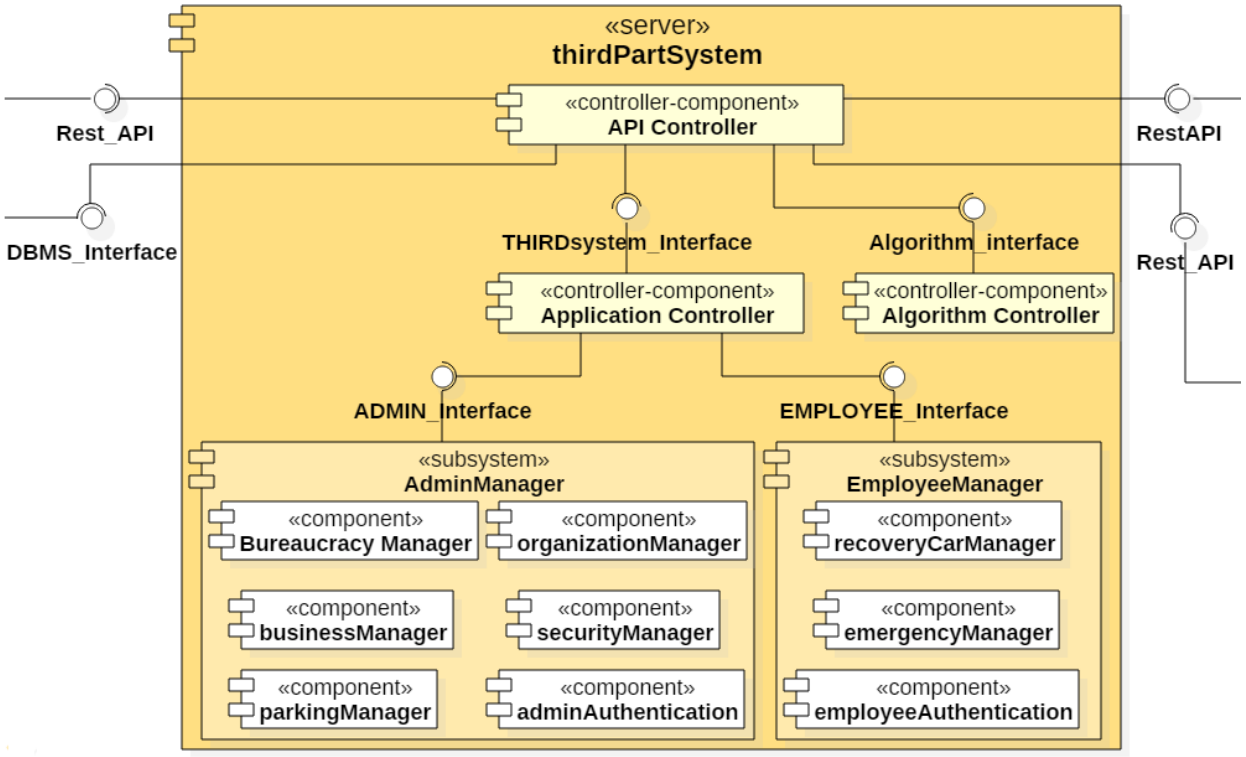

Figure 11

Here we can see the two main subsystem:

AdminManager

This component is used from the Admin that access to the system with a Browser and from there can operate very important organization action. This component manages the action of the admin: as bureaucracy thing (fine and assurance expiry), as maintenance and security of the system.

EmployeeManager

This component manages all the task of the employee as the relocation of the car, the plugging of car and all the tasks that he must do to keep the entire carSharing System working.

Employee App

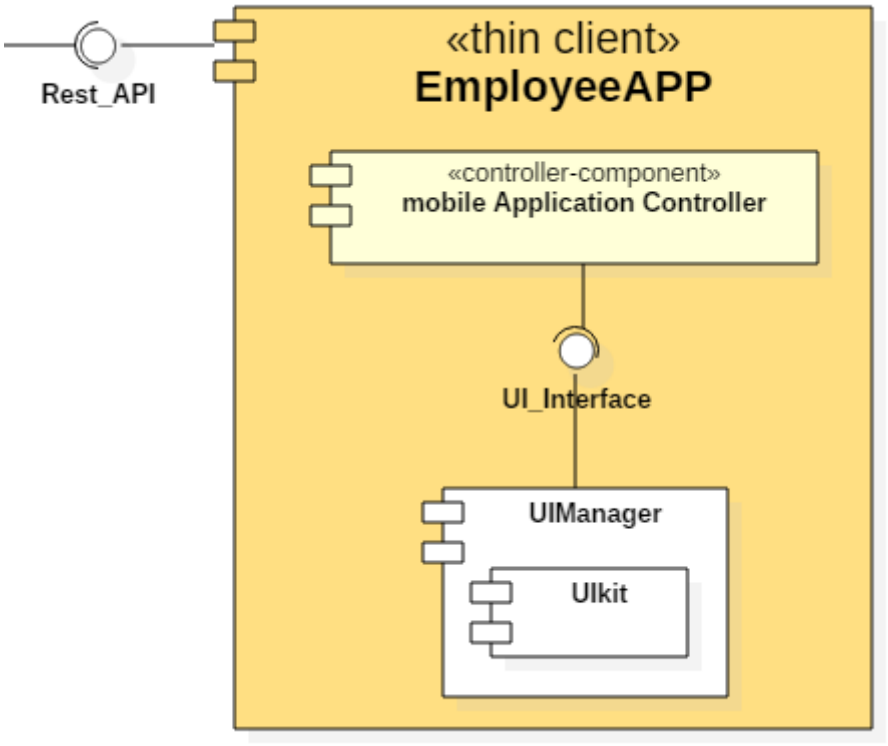

Figure 12

Admin Browser

Figure 13

The web server is implemented using Java EE web components, namely JavaServer Faces (JSF), which is a server-side framework based on MVC. The web server runs on an Apache Server. The web tier only implements the presentation layer: all the business logic is handled by the application server tier. The web tier uses the RESTful interface of the application tier. Using JSF, the view is written as XML files and is completely separated from the logic of the web server. This enables us to write a modular web service. The web server architecture is composed simply by JSF and by a controller class that takes the admin input and translates it in API requests.

2.4 Deploying view

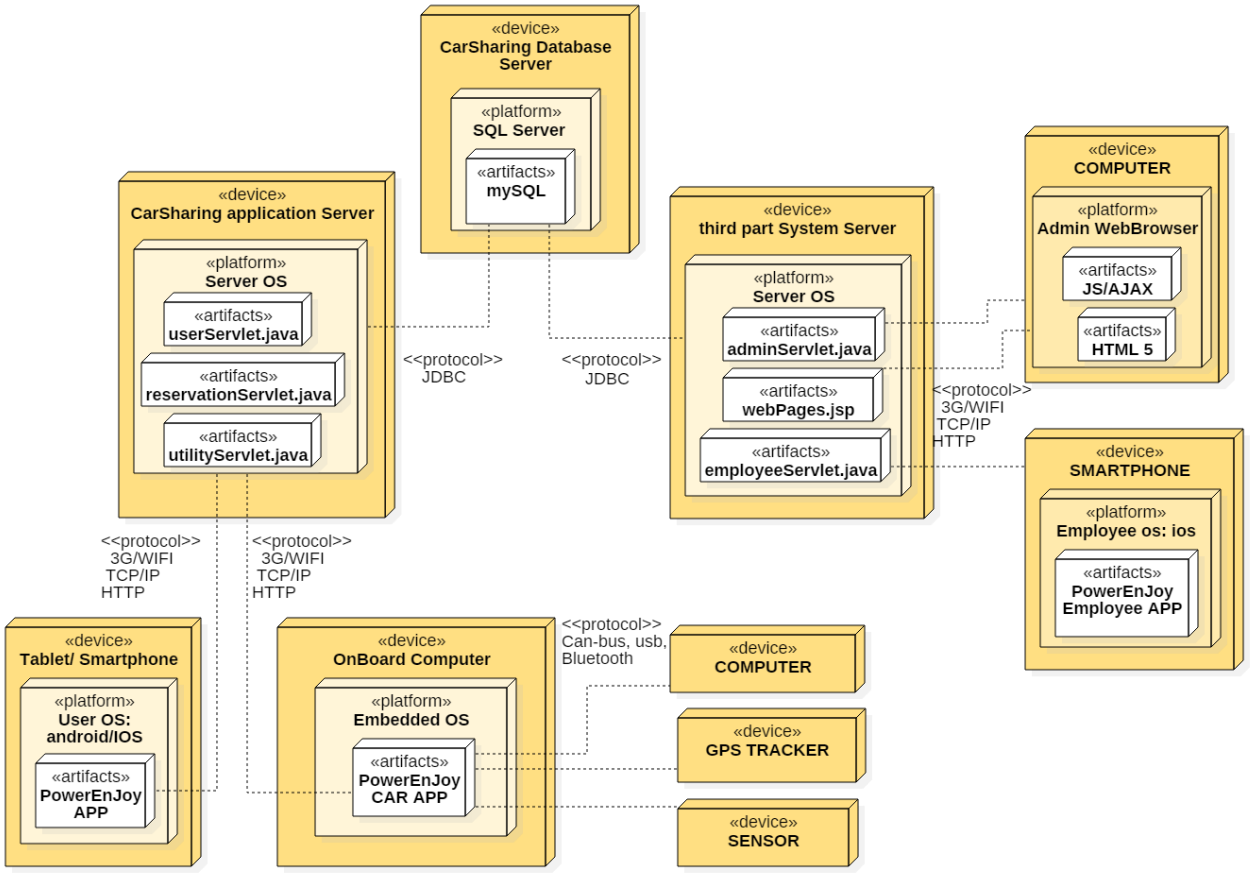

Figure 14

After focusing on the logical structure of the system and on the interactions within, it is appropriate to provide a lower level view on the architecture of the system. In particular, the deployment diagram shows the distribution of the concrete software over the various computational nodes of our system. To reflect the architecture outlined, the diagram develops over three types of hardware systems: **tablet/smartphone**, **computer** and **onBoardComputer** are the three possible client machines, then we have the **carSharing application server** and the **Third System application server**, intended for the business logic, and the **CarSharing Database server**, dedicated to the database. The main interactions between the components have been drawn to give a more complete view on the architecture of the system.

2.5 Runtime view

2.5.1 User Log In

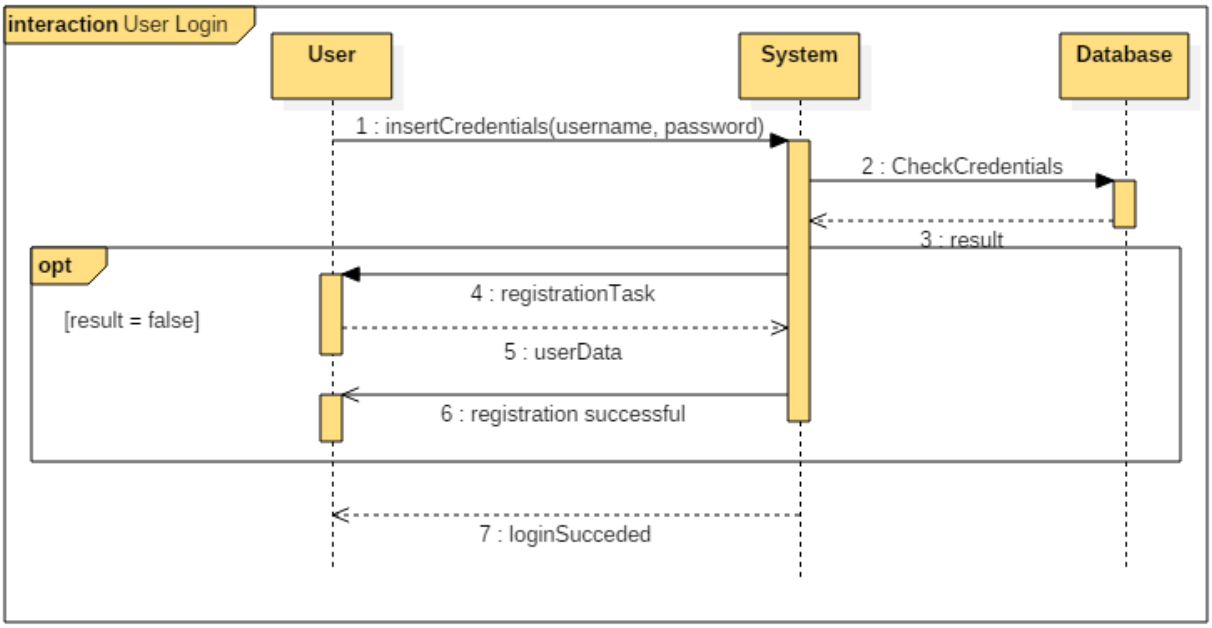

Figure 15

In order to login in the application, the user has to input his credentials, including the password received via email. The client sends all the information that user inserted to the application server that checks the existence of the combination within the database: if the combination exists, then it returns to the client the main screen of the application, from where the user can do different actions. Otherwise, if the system does not match any combination in the database, sends to the client the form for the registration task, specified in the next sequence diagram, and after the user inserts all the requested data in the correct way, system returns a message of successful registration.

2.5.2 User Registration

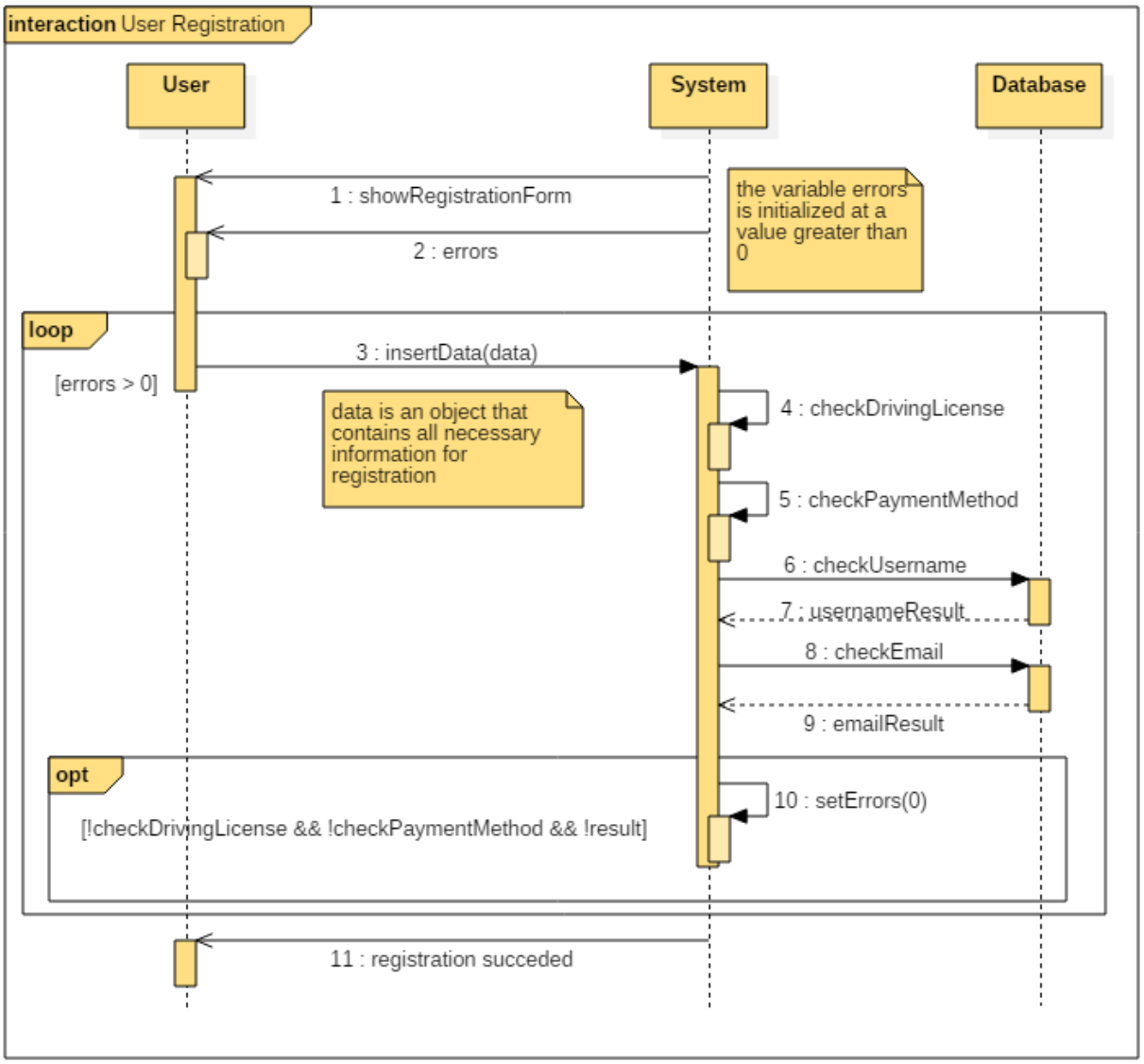

Figure 16

If the user is an unregistered user, he has to click on the “register” button in order to join the community. The system sends to the client a registration form that has to be filled with different data, such as name, surname, username, email address, birth date, driving license number and payment method. The “errors” variable sent within the form is a variable that controls the number of text fields to be reinserted: system increases this variable in case of mismatches during the several checks that it performs in order to verify user’s data and until the variable is not equal to 0, the client remains in the registration form. At the end of the task, if the data are all correct, system sends a message of successful registration.

2.5.3 Car Pick Up

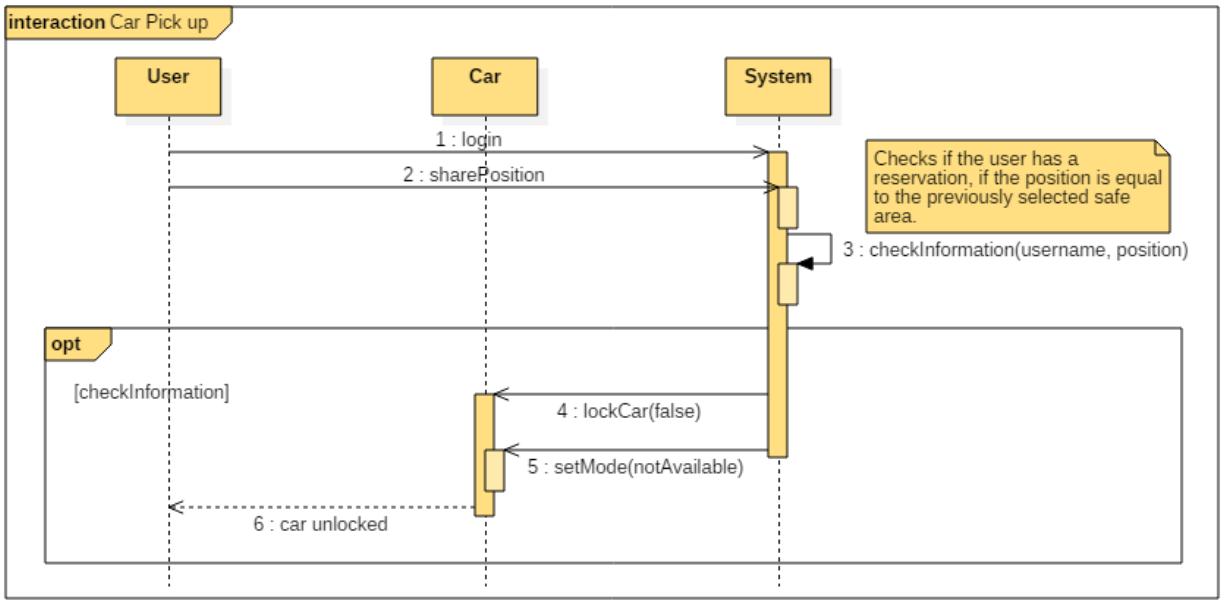

Figure 17

In order to pick up the reserved car, the user has to login in the client app, share his position with the GPS to the system that checks if the user has a reservation and if the position is equal to the position of the safe area selected during the booking. If system matches this information, it unlocks the car, sets the right car state and returns the control to the user that can get the car.

2.6 Component interfaces

GPS Interface
To convert addresses to GPS coordinates we use the google Maps Geocoding API. Geocoding is the process of converting addresses into geographical coordinates, which you can use to place markers on a map, or position the map Also Data is available in JSON and XML format. It will possible to use Google Maps Roads API. The Google Maps Roads API allows you to map GPS coordinates to the geometry of the road, and to determine the speed limit along those road segments. The API is available via a simple HTTPS interface, and exposes two services:

DBMS Interface
The DBMS Interface is thinking as a generic one because it can be of many types (or example is MySQL). The DBMS Controller is not related to the specific DBMS, because it's only an abstraction of this interface and it should work with everyone. This interface is fundamental to do operation on the real database.

REST API Interface (API Controller to front-ends)
The front-ends of the system (the web application and the mobile app) shall communicate with the application server using the back-end programmatic interface implemented as a RESTful interface over the HTTPS protocol. The RESTful interface is implemented in the application server using JAX-RS and uses XML as the data representation language.

Algorithm Interface
This Interface allows the call to the method of some algorithm for example the save money algorithm to manage the discount or the addition charging to the user.

APP Interface
This interface links the component Application Controller and the rest of application: It allows the communication between API controller and the Mobile Application from a side, and the logic of the application on the other side. It also permit the communication between the Algorithm and the DBMS with the logic part of the Application.

UI Interface
This interface is the set of commands or menus through which the user communicates with the program. So it is all the thinks displayed on the app to make the user and the program interacting.

USER interface
Offers the methods useful to log in a User. It interacts with PersonalDataManager component to complete the login and start the user session. Provides the methods for the insertion of a new customer in the database, validating his data. Provides the methods that check the personal credentials in the database. Provides all the methods to validate personal data, for instance the correctness of the name (it cannot contain numbers) or of a birthdate (it shall not be in the future).

UTILITY Interface
Offers the method to detract money for the payment and sent the email notification of the bill or a notification of some event. Provides the methods for the notification of client systems.

RESERVATION Interface
Provides the methods to make a reservation. It interacts with other components for data validation and storing. Provides methods to see car state information and to unlock the car.

SENSOR Interface
Sensor interface connect directly to a sensor element and provide the needed signal conditioning to extract an accurate signal for monitoring and control systems. Fully integrated sensor signal conditioners connect directly to the sensor, perform conditioning, and optionally fulfill data communication functions.

ADMIN Interface
This interface permits to: manage basic software configuration, Create and configure groups, Create and manage databases, Backup and restore content, manage all the maintenance of the system of carSharing, organize the work for the employee, control the bureaucracy part of the system as fine or insurance expiry, control the equal distribution of cars in the city.

EMPLOYEE Interface
This interface offers the methods to organize the work of the employee, like the relocation of cars, the emergency problem etc.

2.7 Selected architectural styles and patterns

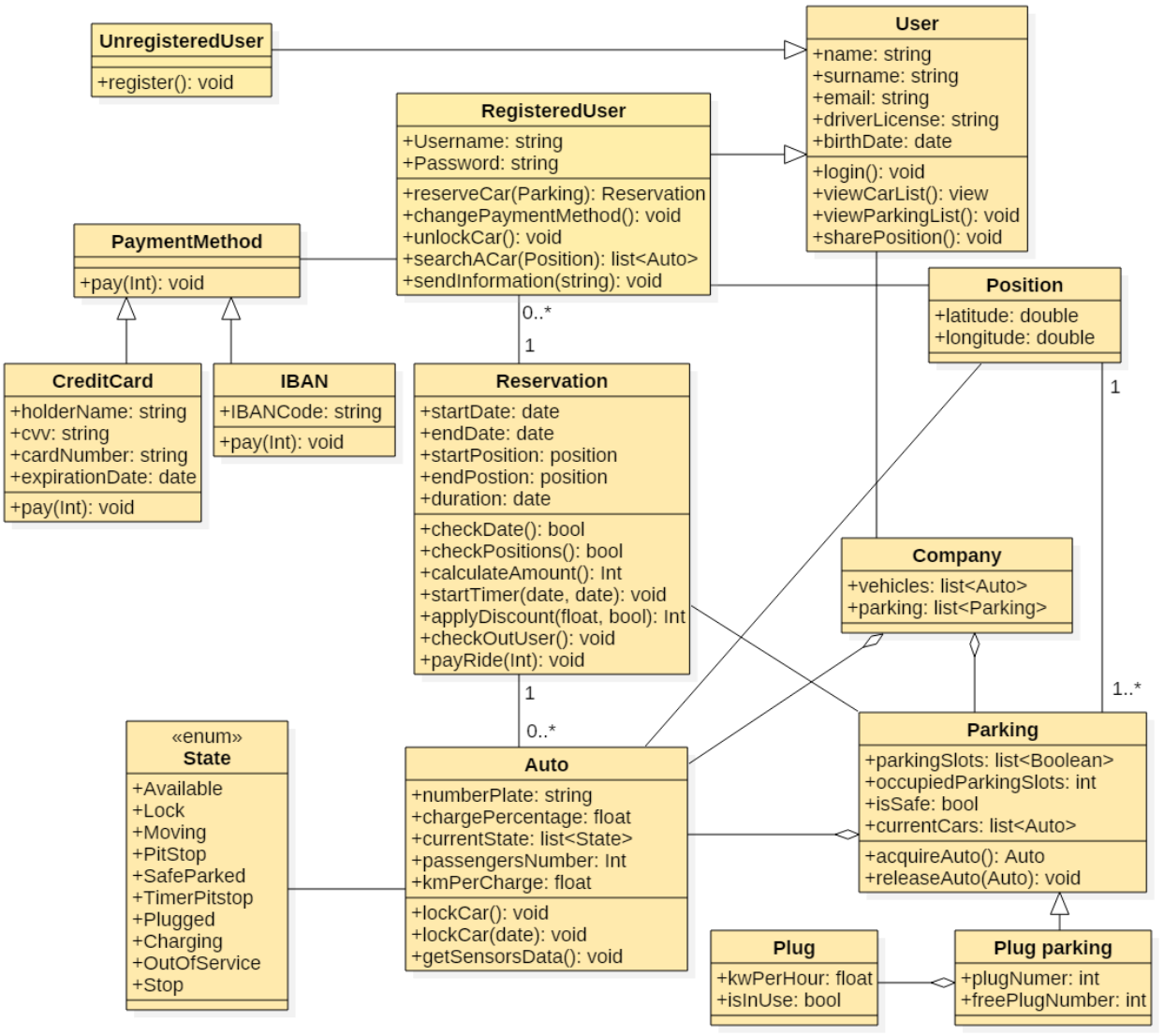

Figure 18

2.7.1 Overall architecture

Our application will be divided into 3 tiers, as illustrated in the section 2.1 of this document:

- Database;
- Business Logic Layer;
- Presentation Layer (thin client).

2.7.2 Protocols

The protocols that we use for our system are:

- RESTful API: we decided to use Representational State Transfer (REST) is a style of architecture based on a set of principles that describe how networked resources are defined and addressed, and paired with JSON, allows us to define also the communication between the application server and the clients. Since our system implements this architecture, it has these properties:
  - State and functionality are divided into distributed resources;
  - Every resource is uniquely addressable using a uniform and minimal set of commands (typically using HTTP commands of GET, POST, PUT, or DELETE over the Internet);
  - The protocol is client/server, stateless, layered, and supports caching.
- MySQL Protocol: This protocol links the application server (that acts as a MySQL client) with the Database, which is a MySQL database. It is implemented by Connectors, a proxy and a communication between master slave replicated servers. It gives a set of ready-made Java functions in order to better manage the database from the application server.

2.7.3 Design patterns

Architectural patterns:

- Client-Server: this pattern best fits the purpose of the system offering a service to users when they need it for these reasons:
  - It can reflect the real use of the system: many users that require a single service through a common method and network;
  - It defines a common set of application operations available to different clients and coordinates the response in each operation;
  - It allows the server to manage distribution and allocation of common finite resources according to all the clients in the best way.
- Model View Controller: this pattern best fits the system shape and the interaction of its components:
  - The model: the resources of the system
  - The controller: the program on the servers that handles resources and their allocation. It implements all the logic of the system, dealing with the users requests and distributing the resources in the best way between the users.
  - The view: the application that manages the user interaction with the system. It provides a GUI granting ease of use, makes structured requests to the server according to the user's will and shows the server answers in a user-friendly way.

Creational patterns:

- Object pool: to implement the "Car" objects, visible in the class diagram, and their use. In this case, the vehicles we deal with are finite and doesn't change during the runtime. For this reason, an object pool helps with the reuse of existing objects: saving resources of the system and the giving a structure to the dynamic of rentable/not rentable cars. The reservation uses a "Car" object and when it finishes to use it, this returns to the parking pool corresponding to its location.

Behavioral patterns:

- Strategy: to implement the payment method. This lets the system enable an algorithm's behavior at runtime. In this case is particularly useful for the ability to manage the payment method chosen by the user;
- Observer: to allow the system to broadcast the state of the cars to the user clients in order to avoid conflicts during a simultaneous reservation procedure on different clients.

2.8 Other design decisions

In order to simplify the implementation of city map, we include in our system the Google API for the managing of route creation and all the tasks linked to this operation.

3 ALGORITHM DESIGN

3.1 Money Saving Option

```
Parking DestinationAnalyzer(double x, double y) {
        double x = parking.get(0).posX, y = parking.get(0).posY;
        list<Parking> availableParking = new ArrayList<Parking>(), slcParking = new ArrayList<Parking>();
        Parking[] availableParking = searchParking(parking), slcParking;
        for(double distance = 1000; slcParking.size() <= 0; distance+=100)
                for (park of parking)
                        if (distanceCalculator(posX, posY, x, y) < distance) {
                                x = park.posX;
                                y = park.posY;
                                slcParking.add(park);
                        }
        Parking finalParking = slcParking.get(0);
        for(park of slcParking)
                if(park.isPlugParking && park.freePlugNumber > 0)
                        if(!finalParking.isPlugParking ||(finalParking.isPlugParking &&
                           distanceCalculator(finalParking.posX, finalParking.posY, x, y) >
                           distanceCalculator(park.posX, park.posY, x, y)))

                                finalParking = park;

                else if((!finalParking.isPlugParking)
                        &&(park.occupiedParkingSlots <= finalParking.occupiedParkingSlots)
                        &&(park.parkingSlots.size() > finalParking.parkingSolts.size()))

                        finalParking = park;
        return finalParking;
}
List<Parking> searchParking(List<Parking> parking){
        List<Parking availableParking = new ArrayList<Parking>();
        for(park of parking)
                if(park.occpiedParkingSlots < park.parkingSlots.size())
                        availableParking.add(park);
        return availableParking;
}
```

This algorithm explains how the money saving option selects the parking where the User should leave the car in order to take a discount.

The algorithm works as follows:

- Checks for parking at a certain distance from the designed point. If no parking is found it increases the distance of search until it can find at least one parking with a free parking slot.
- The system selects the most suitable parking giving priority to plug parking in order to avoid the necessity of moving the car to charge. If no plug parking is found the system checks the number of occupied parking slots in every other parking near the area provided and simply selects the one with the less number of cars and the greater number of parking slots.

This method grants the uniform distribution of cars between the parking because in case of absence of plug parking it distributes the car trying to fill the gap between the occupied parking slots from one parking to another.

3.2 Reservation

```
List<Parking> searchACar(Position pos) {
        double distance = 1000;
        List<Parking> availableParking = new ArrayList<Parking>();
        for(double distance = 1000; available.size() > 0; distance+=100)
                for (park of parking)
                        if (dCalc(park.position.latitude, park.position.longitude, x, y) < distance
                           && availableVehicleCheck(park))

                                available.add(park);
        return availableParking;
}
bool availableVehicleCheck(Parking park){
        for(vehicle of park.auto)
                for(currentState state of vehicle)
                        if(currentState == State.available)
                                return true;
        return false;
}
```

These are some key functions that explain how the searching of available cars is handled in order to give the possibility to the user to make a reservation. These actions are triggered from a request of the user, through the application, to find the available cars near him or near the given position. Reaching the system, this request triggers the serchACar function that checks for all the parking near the position. The selection follows two checks:

- Distance check: the dCalc function works using Google Maps API, sends a request to Google Maps server and receives the distance between two geographical points.
- Car availability check: it checks if the parking detected has at least one available car

The result of serchACar function is sent to the user that can proceed to book a car. The selected parking area is sent back to the system that checks again if there's an available car and then proceed to the change of vehicle state, the reservation object creation and one-hour timer activation.

4 USER INTERFACE DESIGN

The User Interface Mockups are already included in the RASD at section 3.2.

In order to clarify how the User Experience should be, we insert in this document a state diagram where every state represents a specific section or view of the mobile application (we suggest to refer to the RASD at the section 3.2 for the binding of the states with the mockups).

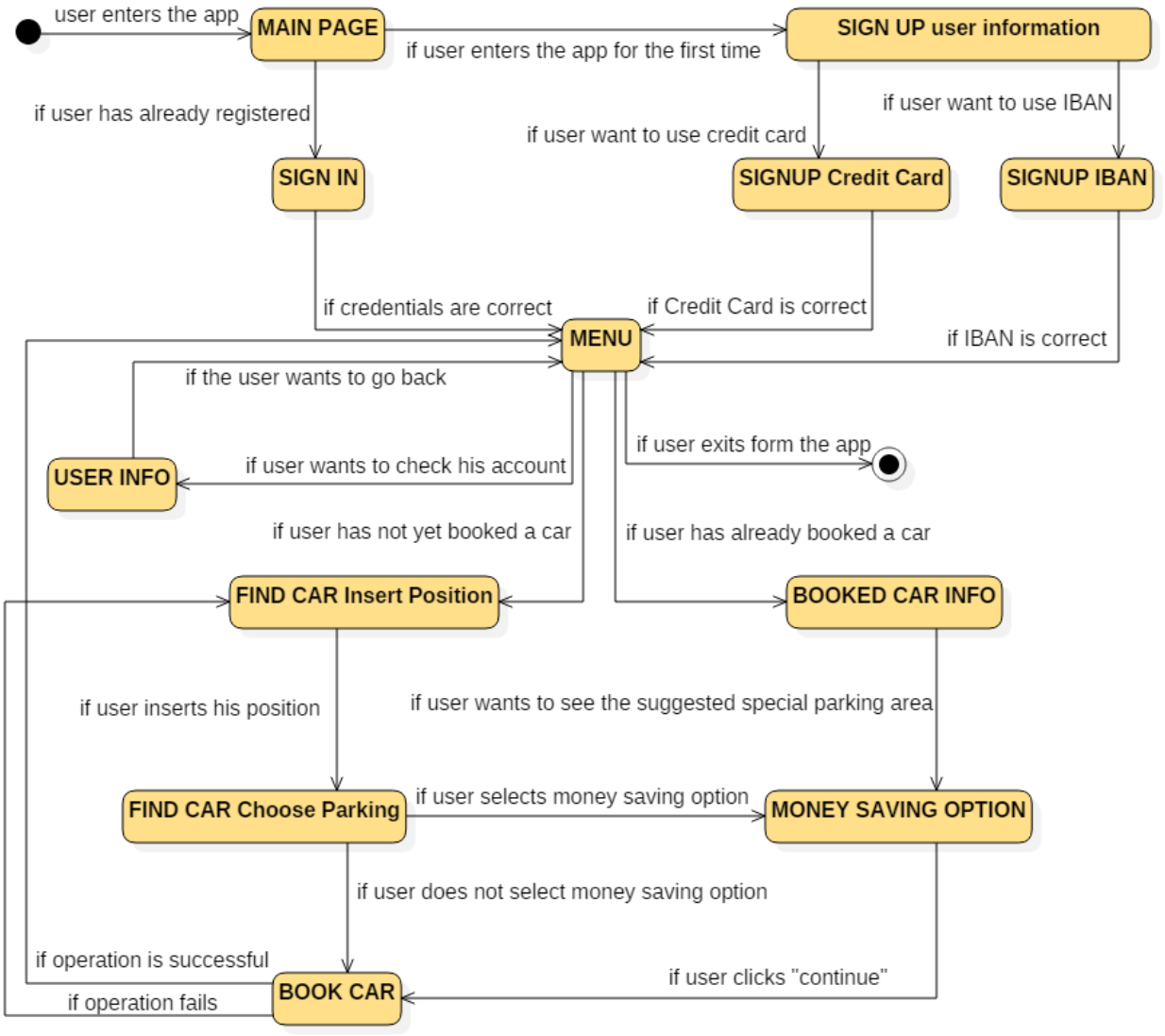

Figure 19

5 REQUIREMENTS TRACEABILITY

The system has to fulfill the goals defined previously in the RASD. We put a list below that contains all the goals defined previously but in a shorter shape with every component that fulfills them one by one.

- [G1] Users must have the possibility to register to the system;
  - UserAPP.AuthenticationManager.Registration;
  - System.UserManager.UserRegistration;
  - System.DataManagement;
  - Database
- [G2] The user can select the parking lot from a subset of them where to pick the car;
  - UserAPP.ReservationManager.ReservationCreation;
  - System.ReservationManager.ReservationManager;
  - System.ReservationManager.BookingManager;
  - Car.GPS;
- [G3] The user will be able to reserve a car for up to one hour from the pickup, from the list of the available ones in the selected parking lot;
  - System.ReservationManager.ReservationManager;
  - System.ReservationManager.BookingManager;
  - Car.GPS;
  - System.DataManagement;
  - Database;
- [G4] The system must control that every reserved car will be picked in the time range;
  - UserAPP.UtilityUserManager.UnlockCarManager;
  - UserAPP.UtilityUserManager.NotificationReader;
  - System.ReservationManager.ReservationManager;
  - System.ReservationManager.BookingManager;
  - System.ReservationManager.UnlockCarManager;
  - Car.SensorManager.UnlockCar;
- [G5] The system must charge the user by a certain amount of Euros per minute;
  - Car.GPS;
  - Car.SensorManager.UnlockCar;
  - System.ReservationManager.UnlockCarManager;
  - System.ReservationManager.PaymentManager;
  - UserAPP.UtilityUserManager.Payment;
- [G6] The system must provide the user the possibility to select the money saving mode;
  - UserAPP.ReservationManager.ReservationCreation;
  - System.ReservationManager.ReservationManager;
  - System.ReservationManager.BookingManager;
  - Car.GPS;
- [G7] To close the bill, the user must park the car in one of the predefined parking areas.
  - System.ReservationManager.ReservationManager;
  - System.ReservationManager.BookingManager;
  - Car.GPS;
  - System.ReservationManager.PaymentManager;
  - UserAPP.UtilityUserManager.Payment;
  - System.DataManagement;
  - Database;

6 REFERENCES

6.1 Used tools

The tools that we used to write this document are:

- WPS Writer and Microsoft Office Word: document text editors;
- StarUML: UML diagram editor, used to design the UML diagrams;
- GitHub: hosting website for software development, used as shared working directory.

7 HOURS OF WORK

ALICE				
GIORNO	ORA INIZIO	ORA FINE	OGGETTO	ORE LAVORO
28/11	16.00	20.30	Overview, high component, component	4.30.00
29/11	15.00.00	18.00.00	deployment and component interfaces	3.00.00
04/12	15.00.00	20.00.00	deployment overview	5.00.00
05/12	15.00	21.00.00	overview, high component description	6.00.00
06/12	16.00	22.00	component view	6.00.00
07/12	14.00.00	22.00.00	component view and description	8.00.00
07/12	23.20	0.00.00	interface view and description	0.40.00
08/12	0.00.00	1.00.00	interface view and description	1.00.00
08/12	17.00.00	20.00.00	deployment and description	3.00.00
09/12	14.00.00	15.30.00	dd	1.30.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
			TOTALE	
				38.40.00

MARK				
GIORNO	ORA INIZIO	ORA FINE	OGGETTO	ORE LAVORO
28/11/2016	17.30	19.00.00	Started to write DD	1.30.00
28/11	20.30.00	21.10.00	Alternative Component Diagram	0.40.00
30/11/16	14.30.00	16.15.00	Continued DD and started Sequence Diagrams	1.45.00
02/11	12.00.00	14.30.00	Sequence diagrams	2.30.00
05/11	17.30.00	21.30.00	Modified DD, corrected Class Diagram, started Algorithms	4.00.00
06/11	18.00.00	21.45.00	Modified DD, started UX, started sequence diagrams descriptions	3.45.00
08/11	12.15	15.30.00	Sequence diagrams descriptions, design state diagram	3.15.00
08/11	18.00.00	22.00.00	Added some sections to DD	4.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				TOTALE
				21.25.00

DAVIDE				
GIORNO	ORA INIZIO	ORA FINE	OGGETTO	ORE LAVORO
28/11	21.00.00	23.00.00	Class diagram	2.00.00
29/11	15.20.00	17.20.00	Class diagram	2.00.00
02/12	15.30.00	17.30.00	Design pattern	2.00.00
03/12	22.00.00	23.20.00	Added functions to component interface	1.20.00
04/12	9.30.00	11.00.00	ER model	1.30.00
06/12	21.00.00	23.00.00	Algorithm part	2.00.00
07/12	15.00.00	16.30.00	Algorithm part	1.30.00
08/12	10.00.00	12.20.00	Design Document	2.20.00
08/12	14.00.00	16.00.00	Design Document	2.00.00
08/12	17.10.00	20.00.00	Design Document	2.50.00
08/12	20.59.00	23.59.00	Design Document	3.00.00
10/12	17.00	17.40	Finished Design Document	0.40.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				TOTALE
				23.10.00

8 CHANGELOG

V 1.1: explicated assumption on car computer and its communication with main server for detected data dispatch.

V 1.2: added new design pattern: observer.

V 1.3: little text fixes.