Design Document

November 25, 2016

Document Version 1.1

Matteo Frosi (mat. 875393)
Luca Costa (mat. 808109)

Contents

1 Introduction

1.1 Purpose

This document, addressed mainly to developers, aims to go into further details regarding the specification of the project, described in the RASD. Here more technical features will be described, including:

- The high level architecture
- Some possible design patterns
- The description of the main components and their interaction (Runtime View)
- A brief description of the algorithms on which the software relies
- A more detailed overview of the user interfaces

Although these are the main topics of the document, other minor details will be touched and discussed, such as the architectural style and a brief comparison with other styles and the mapping between the requirements defined in the RASD and the designed elements described in this document.

1.2 Scope

PowerEnJoy is an electrical car sharing service, based on a mobile application. The targets of the service, intended as users, are people that needs to move from a place to another within a city and requires a conveyance to move (because they don't have their own or simply can't use it).
A user can make a reservation for a car, using the mobile app and his/her account, and check for the availability and status of all the cars within his/her position, identified using GPS localization, or a specific one, inserted manually by the user. As stated before, to access the service, the user must possess a private account, so a registration is needed.
The system provides the users a safe way (identification code) to access the cars, and the riding service and keeps trace of the status of all the cars.
Moreover, the system prizes or punishes a respectively good or bad behavior from the users, applying a discount or an overcharge on the cost of a ride. As example, if the user leaves the car without much battery, he/she will have to pay more than the standard cost of the ride, because the car will need to be charged and this operation has a cost. On the other hand, if a user plugs the car before ending the service, it receives a discount.
The system includes other functionality, such as GPS based maps available in every car, an emergency procedure in case an accident occur during a ride and the notification of a car status if the user requested it.

1.3 Definitions, acronyms, abbreviations

- RASD: document about the requirements analysis of the project.
- DD: document about the design choices and the components description of the project.
- GPS: global navigation satellite system that provides location and time information in all weather conditions, anywhere on or near the Earth where there is an unobstructed line of sight to four or more GPS satellites.

- SMS: short message service; it is a notification sent to a mobile phone, we need a GSM gateway to use it.
- GMS gateway: device that allows SMS text messages to be sent and/or received by email, from Web pages or from other software applications by acquiring a unique identifier from the mobile phone's Subscriber Identity Module, or SIM card.
- Push notification/ push message: it is a notification sent to a smartphone using the mobile application.
- API: application programming interface; it is a common way to communicate with another system.
- REST: representational state transfer, it's one way of providing interoperability between computer systems on the Internet.
- RESTFul: REST with no session.
- UX: user experience design
- URL: uniform resource locator.
- MVC: model view controller, it's a design pattern.
- SOA: service oriented architecture, it's a style of software design where services are provided to the other components by application components, through a communication protocol over a network.
- Layer: way of organizing code in sections sharing a common goal. The highest partition includes Presentation Layer, Business or Logic Layer and Data Layer.
- Tier: physical deployment of layers.
- Safe area: it is a specific area where the electric cars of PowerEnJoy service can park. The set of safe areas is pre-defined and owned by the company/society that requested the management system for the service.
- Special safe area: it is a safe area where power grid stations are installed.
- Power grid station: it is an installation that allows the recharge of an electric car.
- Communication primitives: set of instructions and procedure that allow a communication between machines and devices over a network.
- Sensor data retrieval: procedure that consists in getting all the information collected by the sensors of the car.

1.4 Document structure

- Introduction: this section introduces the design document. It gives an overview on what topics will be covered and what aspects described in the RASD will be improved here.
- Architecture Design: this section is divided into different parts, each describing an aspect of the software design.
  - Overview: brief description of the division in tiers and layers of the application.

  - High level components and interaction: high level view of the components of the application and the way they communicate.

- Algorithm Design: this section describes some of the algorithms that the application will rely on. To focus on the algorithm idea and not the fine grained implementation, pseudo-code will be used.
- User Interface Design: this section presents mockups and user experience explained via UX diagrams.
- Requirements Traceability: this section aims to explain how the decisions taken in the RASD are linked to design elements.

2 Architecture design

2.1 Overview

PowerEnJoy relies on a three tier architecture. Referring to the proposed system architecture presented in section 2.3 of the RASD, the following figure represents the tier division of the system.

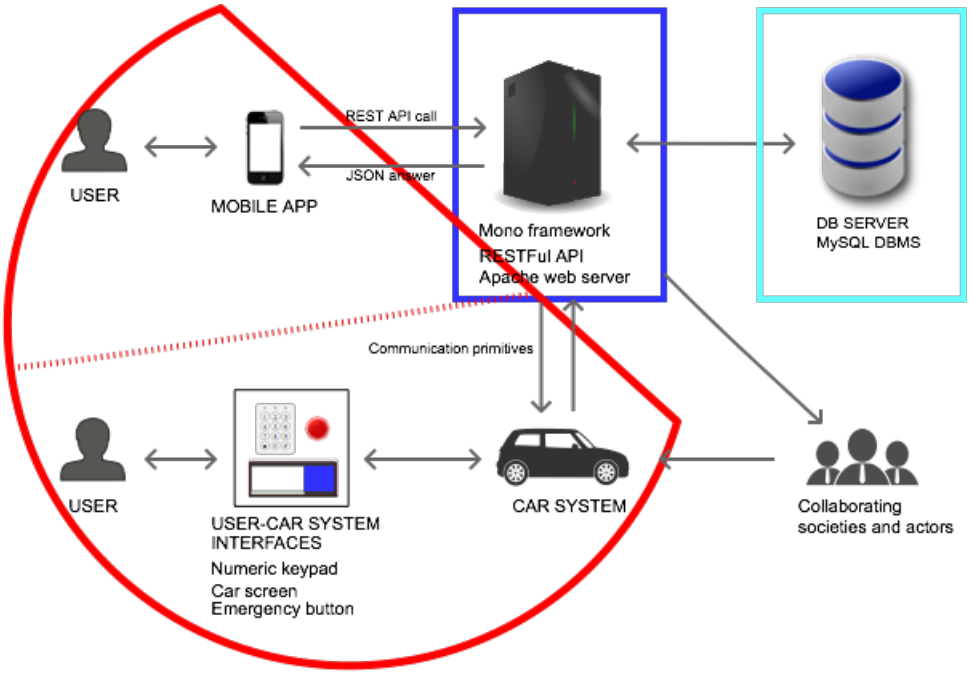

Figure 1: Proposed architecture, tier division - red: client, blue: server, azure: database

The client side includes all the possible ways by which a user can interact with the server (and the service), that are the mobile application and the car itself. The first represents the classical idea of client tier, including a GUI and a minimal amount of processing power that allows the user to have a connection with the rest of the tier, shown in a graphical way, that is more user friendly. The latter is, instead, a particular client tier that acts as a mini server, because it contains more logic than the mobile application. Here there is not a real graphic interface but multiple mechanical interfaces (MUI) can be detected, such as the numeric keypad that make the user able to unlock the car or end the service, the emergency button and the car screen, that is thought more towards an informative interface than an interactive interface. To better understand this concept, follow the same figure, this time divided in high level layers.

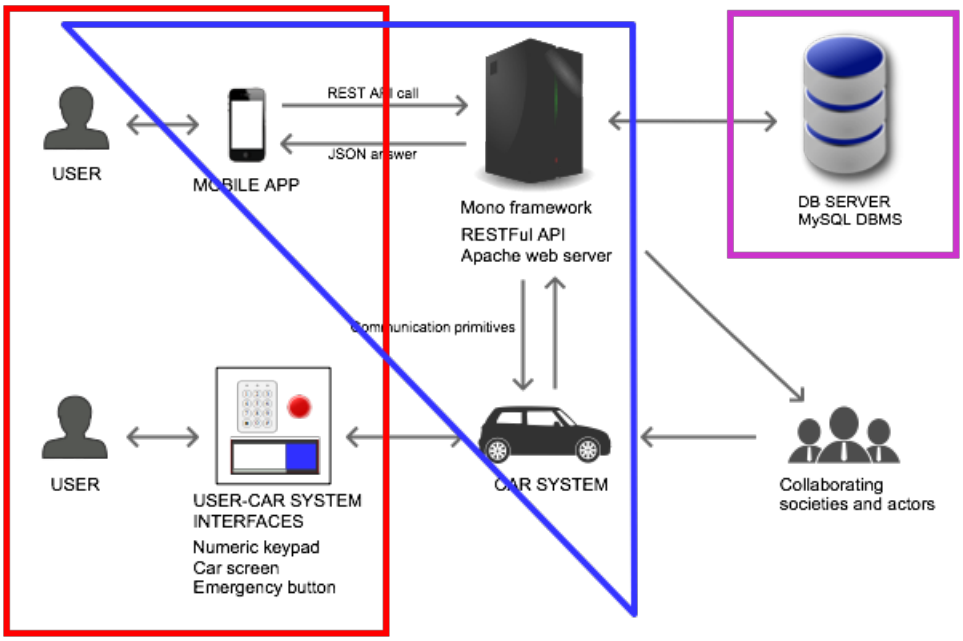

Figure 2: Proposed architecture, layer division - red: presentation, blue: logic/business, purple: data

The mobile application is also part of the logic layer because it can hold some immediate processes, instead of delegating them to the main server. An immediate example could be the elaboration of the GPS based map during the research of an available car.

2.2 Overview of the car system

It should be useful to spend some words to discuss more in detail the structure of the car and the way the user can interact with it, and consequentially with the server. Using multiple sensors, information about the car status can be retrieved, such as battery charge, number of passengers, degradation level of the car components, or even the localization data, obtained with GPS. Such data pool is interfaced with the car system, that resembles the shape of an application. The car app can be considered as a cluster of procedures and interfaces that allows user, car hardware and system to communicate and "know" about each other. Speaking in pattern terms, the car app emulates the controller entity in the MVC pattern, even if only partially. As said in the previous subsection, while the emergency button and the numeric keypad make the user interact with the car in an active way, the car screen acts as a passive element and is simply needed to inform the user of the car status and the service info. We can consider such feature as a way to diminish the interaction user-system to a very strict level, without changing the user perspective of the service. Such reduction lightens the logic needed in the car, making the project more achievable from a budget and complexity point of view.

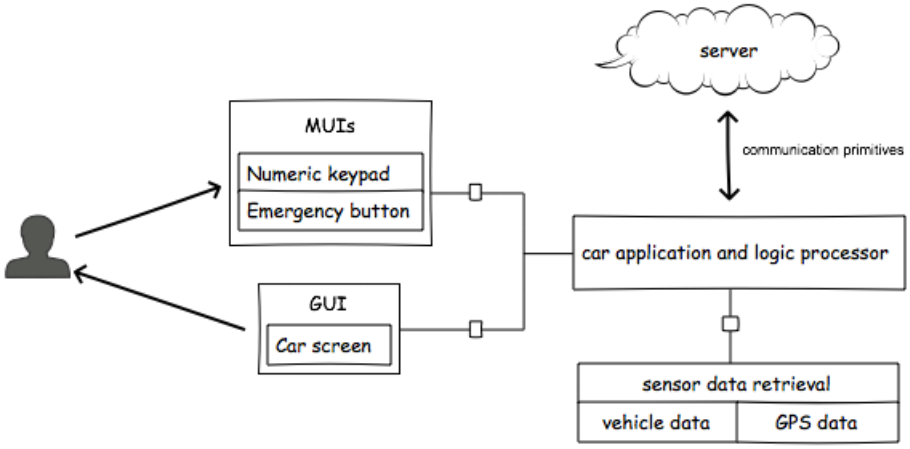

Figure 3: High level car system

2.3 Components of the system

2.3.1 High level view

From an high point of view, the system can be divided into four elements. The users can interact with the server, that is the main central where all the logic is concentrated, directly or with the support of the car. A user can initiate the interaction with the server using his/her mobile application. To proceed in using the service, the user must first be logged into the system and for every action he/she does (monitoring request, reservation request, take back of a reservation, code request and so on), he/she has to wait for the server response, informing the user that the request has been successful or not. Because of this sequentiality of actions, the communication directed from user to server has to follow synchronous rules when proceeding in a certain sequence and leave all the other non-interactive options as to be implemented in an asynchronous way. As example the user shouldn't be prohibited to search for an available car even if he/she already made a reservation, but he/she cannot made a code request if there is no reservation made. A clever way to implement the user-server relationship is to made all the communications asynchronous but thanks to particular patterns, MVC among them, denying the usage of particular services if certain conditions are not met.
The user can also interact indirectly with the server using the car - or it can be said that the car interacts with the server -. Some physical interfaces, such as the emergency button or the numeric keypad, allow the user to modify the condition of a car and, consequentially, of his/her service.
The server/central acts mainly as an elaborator, or referring to the chosen architecture, as a monolithic logic processor. Its job is to coordinate the requests from the users and the change of status of all the cars. As said before, the interaction between user and server can be described with a simple MVC pattern. From a very high view, the user makes an action, communicates it to the server, the server elaborates the request and confirms/rejects the will of the user. The server-car-user relationship is a bit more complex. Firstly, there is a continuous interaction between every car and the server, that follows this sequence:

1. Data retrieval
2. Car status update (car to server)
3. Database update (server to database)

4. [Optional] Car update (server to car), as example the lock following the end of the service

Moreover, to this sequence must be added the interaction with the user, that is both passive and active as said in the previous section.
The server communicates with all the other components using asynchronous messages, because it does not depend on their status but acts as the independent core of the software. Just referring to the user behavior, it would be unthinkable to have the server wait for a user decision, stated that the thinktime can be to long, and each communication would become a bottleneck for the whole system.

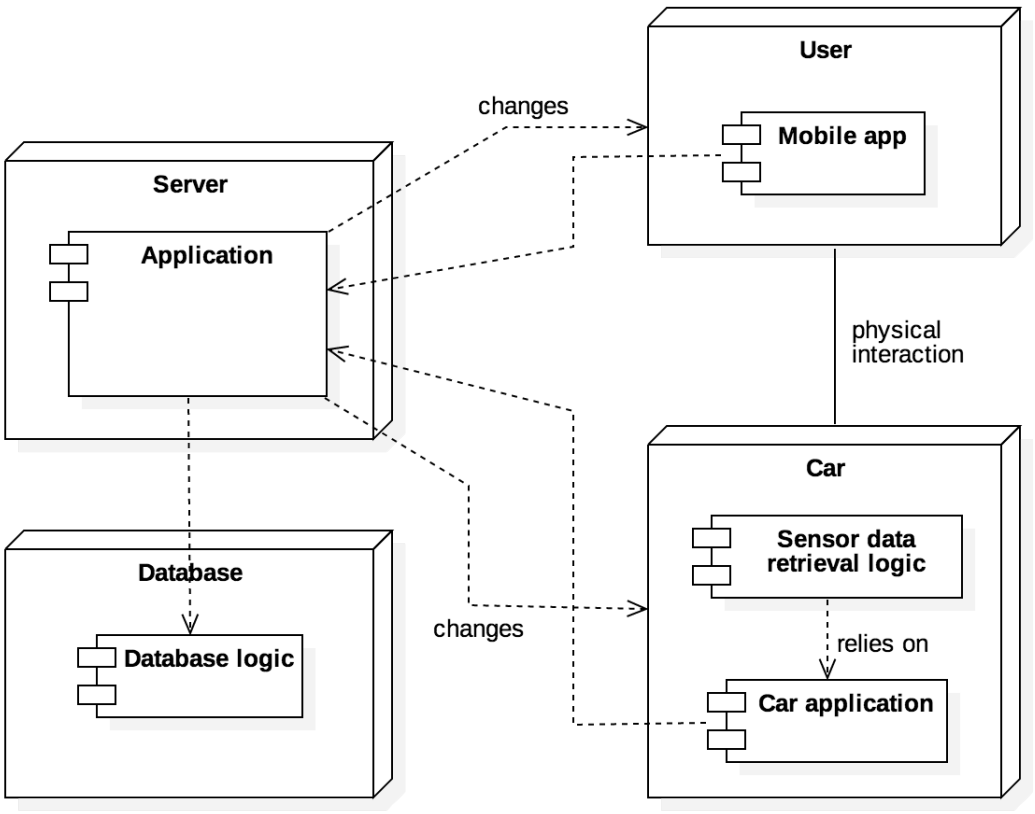

Figure 4: High level view of the components of the system

2.3.2 Detailed views

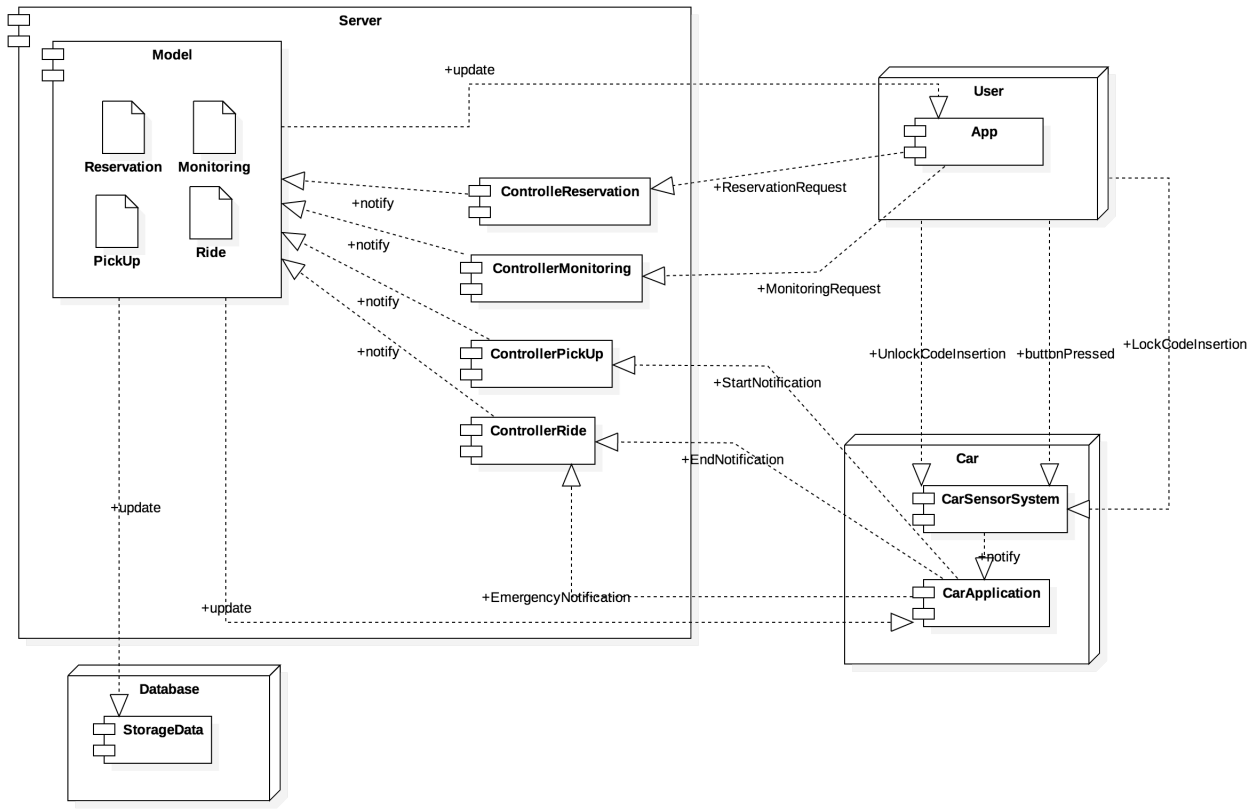

Figure 5: Deeper high level component diagram

Deepening the high level component diagram, each big node can be fragmented into smaller parts, each characterized by its functionality. As previously said, The user can interact directly with the main server/central using his/her mobile application, or, indirectly, using some of the car physical devices (the button and the numeric keypad). Focusing more on the server composition, multiple components to handle and manage every part of the service are needed. Purely from a topological point of view, the server consists of two big areas. One refers to all the structures and components that handles and addresses incoming data from and to the external world, while the other refers to all the functionality and procedures that allow the elaboration of an event and the consequent updates on the server itself, but also on the external world (database, clients or cars).

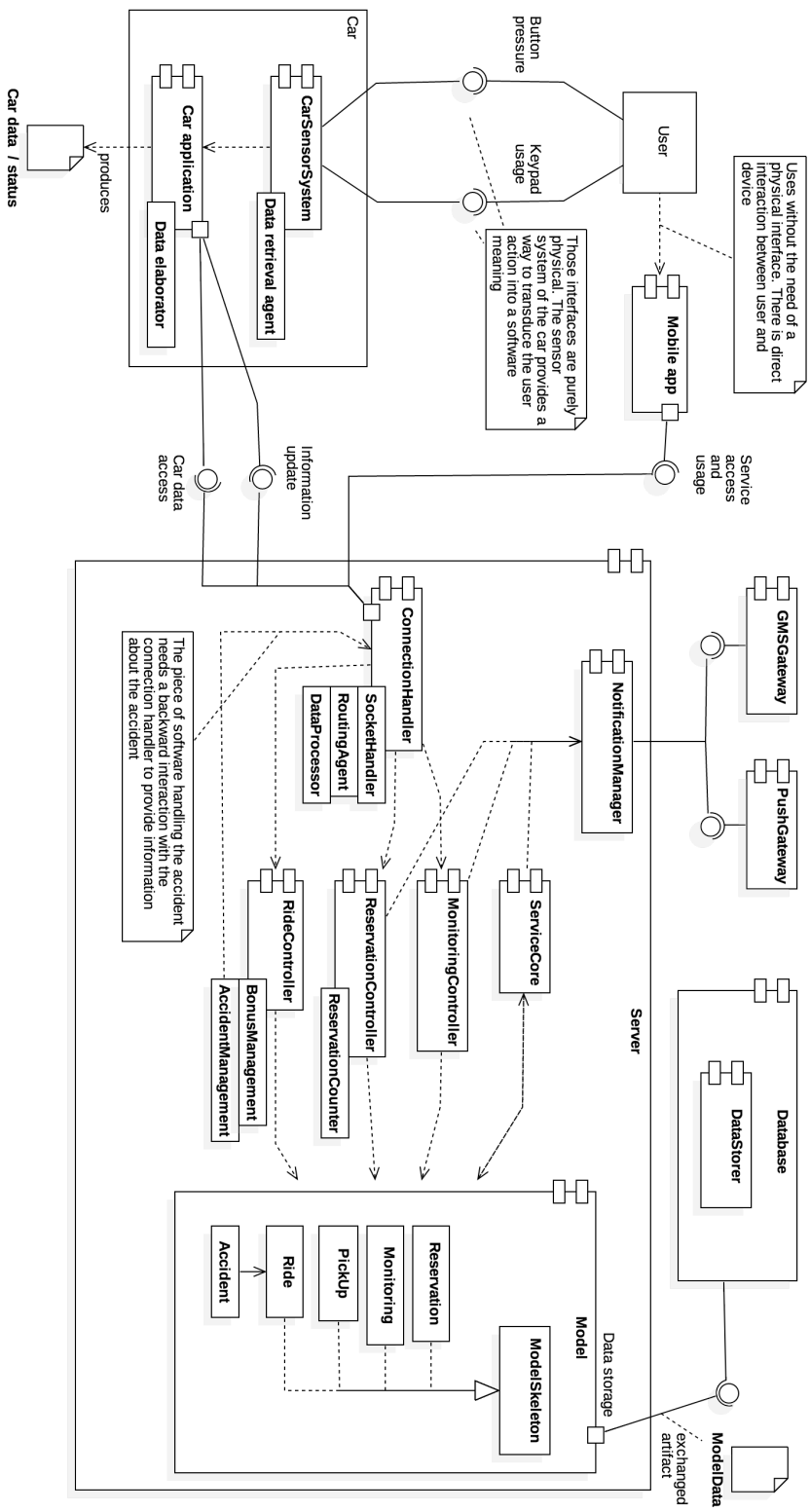

Figure 6: Detailed component diagram

Figure 6 represents an even more detailed component diagram, including sub components, interfaces, products (artifacts) and ports. One of the two area previously described can be seen as a mail office. The component ConnectionHandler, as the name suggests, handles the incoming and outgoing data, sorting it to the specific controller. The other area, composed by the Model and the ServiceCore interacts with the Database, providing it the information about the Model itself. The NotificationManager component takes care of the communication and interaction needed with the user's device. In detail:

- GMSGateway: device that allows SMS text messages to be sent and/or received by email, from software applications.
- PushGateway: device(s) that manages push notifications.
- NotificationManager: component that takes care of the interaction with the user's device.
- ConnectionHandler: component that sorts incoming and outgoing messages (data transfer).
- ServiceCore: contains all the procedures supporting the software (algorithms, functions and so on).
- ModelSkeleton: representation of the analyzed world
- CarApplication: contains all the procedures that elaborates the data retrieved by the sensor system of the car and interacts with the server.
- Database: device used to store the model data

2.4 Architectural styles

3-Tier Architecture : Basically a 3-Tier Architecture is a more complex 1-2 tier architecture.

- Specialized (there is 1 more tier dedicated to a certain task).
- Flexible (if something goes wrong with a tier, the other 2 can work as well).
- Secure (the client will not have direct access to the database).
- Performing (tasks are shared between servers).
- Deployed as a Monolith.
- Simple to test and deploy.

Hexagonal : The core is a business logic implemented by modules with attached several services by specific adapters.

- Deployed as a Monolith.
- Simple to test and deploy.

A Hexagonal architecture is more suitable for bigger system than our. Moreover there are more interfaces to be implemented.
For these reasons we choose the 3-Tier Architecture.

Microservices : set of smaller, interconnected services. Each one has its own hexagonal architecture.

- Easy to deploy independently.

- Each service has its own database.

Microservice pattern is more simple and efficient than a monolith. In fact, every microservice is a mini-monolith. The worse part is that different microservices have different databases and sometimes there is incoherence between them.
For these reasons we choose the 3-Tier architecture.

SOA : architecture is split up into various services.

- Scalable.
- Reusable.
- Easy testing and debugging.
- Flexible.

SOA architecture is not suitable for GUI functionality because it requires a heavy data exchange.
For these reason we choose 3-Tier architecture.

3 Document further information

3.1 References

- RASD, before Version 4.5
- Assignments AA 2016-2017.pdf
- `https://msdn.microsoft.com/en-us/library/ee658116.aspx`, containing some information about layers design
- `http://alistair.cockburn.us/Hexagonal+architecture`, contains some information about the Hexagonal architecture
- `https://www.nginx.com/blog/introduction-to-microservices/`, contains some information about the Microservices architecture
- `https://8thlight.com/blog/uncle-bob/2011/09/30/Screaming-Architecture.html`, contains some advices over the architecture choice and implementation
- `http://www.uxbooth.com/articles/8-must-see-ux-diagrams/`, contains some examples of user experience diagrams
- Sample Design Deliverable Discussed on Nov. 2.pdf
- Paper on the green move project.pdf
- Second paper on the green move project.pdf

3.2 ChangeLog

- [21/11/2016] [**Version 0.1**] :: Added first part of the document introduction, including preface, Purpose, Scope and Definitions. ChangeLog and Hours of work sections also introduced.
- [21/11/2016] [**Version 0.2**] :: Added the References and Document structure sections.
- [23/11/2016] [**Version 0.3**] :: Added the overview of the architecture of the project and a brief description of the car system, so subsections 2.1 and 2.2.
- [23/11/2016] [**Version 0.4**] :: Added the description of the different architectural styles that were considered and their brief comparison.
- [24/11/2016] [**Version 1.0**] :: Added high view of the components of the system with a detailed description, subsubsection 2.3.1.
- [24/11/2016] [**Version 1.1**] :: Added two detailed component diagrams, subsubsection 2.3.2

3.3 Hours of work

Matteo Frosi

[Before 21/11/2016]: 3.00 hours (spent in analyzing well known architectures, to study and apply them at our problem).
[21/11/2016]: 2.00 hours (further analysis of an applicable architecture to our problem, alleging to the classic 3 Tier architectures).
[21/11/2016]: 1.00 hours (writing of the first part of the document).
[23/11/2016]: 2.00 hours (writing of the overview section of the document and drawing of architectures images)
[24/11/2016]: 2.00 hours (drawing up of the component view and writing of the component high view section)
[24/11/2016]: 2.30 hours (drawing of the component in a more detailed view)
[25/11/2016]: 1.00 hours (writing of the section about the detailed component diagrams)

Luca Costa

[Before 21/11/2016]: 2.30 hours (spent learning about the various architectures and apply them to our problem)
[22/11/16]: 1.00 hours (spent learning about a possible implementation of the different architecture styles)
[23/11/16]: 3.00 hours (spent writing comparison between architectures and informing about DD document in general)
[24/11/2016]: 2.00 hours (learning about how to draw a component diagram and its effective first drawing)