Master’s Degree in Software Engineering
Dissertation

OpenCar
The App Platform for Connected Cars

Eduardo Filipe Fernandes da Silva
effsilva@student.dei.uc.pt

Advisor:
Prof. Dr. Filipe Araújo

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Abstract

The *Connected Car* market has been around for a while now, growing from small applications like answering the phone without taking the hands off the wheel to larger ones like getting directions to the last restaurant the driver has been to.

However, as cars improve, the drivers not so much. There is so much to do inside and outside the car that the users become a lot less concerned about the car itself. We believe the functionality of nowadays vehicles infotainment systems can still be extended greatly, and go further than help the driver answer the phone without hands.

The problem could be overcome by keeping track of certain driving behaviours over a given amount of time. Afterwards, the information could be presented in a way that is understandable for the ordinary end-user. Such approach presents many challenges, namely when choosing which events should be considered relevant enough to track, analyse and take conclusions from.

An example of such events can be how many times the wheels locked up under speed. This kind of event not only will cause premature wear on the tyre but will also cause traction loss due to the flat sections created. After happening a few times it may be a good idea to take a look at the tyres to see if they need changing before something bad happens.

This thesis proposes a centre console application capable of identifying and keeping track of this type of event in real-time, coupled to a web application in which the user can consult both raw data and calculated statistics about his vehicles.

This will be achieved using technology from a company that has been for quite some time around this market: INRIX's OpenCar. This technology gives an idea of what this sector of the industry could mean to both software engineers and car manufacturers.

Keywords: Connected Cars, Software Engineering, OpenCar, Business Intelligence, Data Mining, Data Warehousing

Acknowledgements

I would like to thank my thesis advisor Prof. Dr. Filipe Araújo at University of Coimbra not only for his support and counselling during the development of this project but also for believing in the project and always wanting to take it a step further. When I first came to talk to him about this theme so ”out of the box” and with so little time to submit a decent proposal, he backed me up, and I am most grateful for that. Prof. Bruno Cabral at University of Coimbra also had a huge role steering me in the right direction even before I had an idea of what I wanted to do. This thesis could not have existed if it was not for him, therefore I leave my thanks.

I also want to thank all the people who saw and tried out my project working in its final tests and to the *The Driving Club - Coimbra* for giving me lots of feedback and a testing ground that was most useful in the early stages of the development.

Lastly, but not less important, I would like to leave a word of appreciation and gratitude for my family for always supporting me in all my decisions and for the encouraging words through the years that led to this very moment. Also, a thanks to all my friends at the university, both for the support given until the last moment and for these crazy, crazy years. It definitely would not have been the same without any of it. Thank you all.

Contents

List of Figures

List of Tables

Acronyms

ABS Anti-lock Braking System. 42, 55, 60, 65, 73

AGL Automotive Grade Linux. iii, 6–9, 13, 15, 22, 23, 41, 52

API Application Programming Interface. iii, iv, 2, 9, 10, 13, 18, 21, 22, 28, 37, 40, 45, 53, 57–61, 64, 68–71, 76, 77, 81, 82, 89–96

CAGR Compound Annual Growth Rate. 6

CAN Control Area Network. 14, 18, 20, 37

CSS Cascading Style Sheet. 18, 23, 25, 54, 68, 69

ECU Electronic Control Unit. 1, 5, 13, 14

ER Entity-Relationship. iv, 74

FOSS Free and Open Source Software. 23

GPS Global Positioning System. 6, 18

GUI Graphical User Interface. 3, 32, 48

HMI Human Machine Interface. iii, 7–9, 21, 23, 24, 29

HTML HyperText Markup Language. 18–20, 53, 54, 67–69

HTML5 HyperText Markup Language revision 5. 17, 18, 23–25

HTTP HyperText Transfer Protocol. 18, 20, 21, 38

IPC Inter-Process Communication. 9

IS International System. 60

IVI In-Vehicle Infotainment. 3, 6, 7, 9–12, 23, 24

LIN Local Interconnect Network. 14, 18, 20, 37

LTSI Long Term Support Initiative. 9

MOST Media Oriented Systems Transport. 14, 18, 20, 37

MVC Model-View-Controller. 18, 38, 51

MVP Minimum Viable Product. 76

OEM Original Equipment Manufacturer. 7, 12, 14

OS Operating System. 6, 12, 13, 15, 23, 24, 32

RPM Revolutions Per Minute. 2, 55, 56, 58, 59, 65, 73

RTE Run-Time Environment. 13, 14

SDK Software Development Kit. 3, 21–24

SVG Scalable Vector Graphics. 68

TCP/IP Transmission Control Protocol / Internet Protocol. 5

TCS Traction Control System. 55, 58–61, 65, 73

UDP User Datagram Protocol. 37, 38, 47–50, 52, 53

UI User Interface. 44, 76, 78

USB Universal Serial Bus. 26, 29

W3C World Wide Web Consortium. 16

Chapter 1

Introduction

A few decades ago, cars were completely mechanical, rather simple, and over time started acquiring bits and pieces of new and innovative technology such as Electronic Control Unit (ECU) controlled fuel injection or cooling systems. These units were indeed a great leap in making them more efficient, yet they were not that present in them as only a handful of these units were installed in the first times. Today, there are hundreds of these units installed in one single vehicle, opening more and more possibilities for technology to embrace them, being software solutions one of the main targets.

With this market growing at a good rate, many companies have already made available plenty of solutions and applications. Among these solutions, there have been companies focused on producing community-based environments and platforms for developers to try and find more and better solutions, together. An example of such platforms is the very subject of this thesis: the OpenCar Platform.

These solutions have drivers themselves as end-users. The usual driver wants to still be connected to the world when driving the car, which means, he wants to be able to execute tasks he would normally do on his daily life when inside the car without endangering his life by, for example, picking up the phone. However, the *usual driver* is but a portion of the whole public in this market.

1.1 Scope & Motivation

So far, solutions that allow the user to use the phone, both for messaging and calling, have been the most wanted features by users. There are also solutions for music on demand, and even social networking when driving a car. These solutions may cover the needs of the usual driver, but there is more to it than the *usual driver*.

A good way to tackle this rising market would be to aim at a different public. In a society more and more focused on efficiency, cars are slowly becoming autonomous. This might be good news for the *usual driver* but, on the other hand, the drivers that still enjoy a good driving experience on their own are running out of options when buying a new car, often going for older ones.

This specific kind of driver has different goals when driving. They often wonder if they are pushing Revolutions Per Minute (RPM) too high, if they need to have their tyres checked or if they are over-using the gearbox. Just some of the questions that pop in the mind of a regular *gearhead.* This project's main goal is to try and answer these same questions and more.

These motives were the main drive for developing this project, to give this portion of the market's public an option to stay up to date on car technology and still be able to cherish driving, while taking advantage of this small niche's needs without forgetting the rest of the market.

1.2 Methods

As the title of this thesis suggests, the OpenCar Framework will be used. Other technologies were taken into account, however, unlike other platforms OpenCar offers a solid simulator, well-documented API's, code samples and most important of all: easy access to all the car's sensors.

The main idea is to gather information from the various sensors made available by OpenCar's API's and calculate results to report back to the user. These results would consist either in statistic data about the possible condition of the car or the driver's performance in a given time period. This will be achieved by developing an API that will serve the OpenCar application and embed an web application. The OpenCar application will be responsible for detecting a given set of events and collecting the data, sending it to the API where it will be treated and stored. The web application will access that same API to retrieve both raw data and calculated statistics that were previouly saved and processed.

Since OpenCar is rather new, the possibility of testing the software in a real car is pretty much impossible. Given this unfortunate situation, testing the reliability and usability of the software to be developed will be done through a driving simulator with accurate physics, delivering information in a similar format to OpenCar's.

1.3 Objectives

This project will consist mainly in two distinct objectives:

- **Successfully implement and test a demo to prove the potential of the system**

Make a small app that will use some features of the framework and connect it to the rest of the system. This app will simple collect data from a small set of sensors from the car, measuring times from 0-100 km/h accelerations and reporting calculated results about the drivers performance while monitoring a selected set of sensors for related information.

- **Improve the demo to a more professional and full product**

  Once the concept was proven successful, there is a green light to re-use the code and start developing the application that this project will consist, with a larger set of sensors and a much bigger set of results calculated. This will allow the user to receive feedback on his driving, or even to try and trace the cause of a possible accident.

1.4 Document Structure

Apart from this same introductory chapter, this document is separated in multiple chapters.

The *State of the Art* chapter will give a brief introduction to the concept of "*Connected Car*" and describe the major entities currently active on this particular market. The first section is dedicated to organisations that may or may not include various partnerships and are currently providing In-Vehicle Infotainment (IVI) systems from the Graphical User Interface (GUI) to the hardware inside the car. The second section is dedicated to development ecosystems in the form of platforms in which a developer can sign up and use the provided Software Development Kit (SDK)'s to develop an idea that may or may not pass the quality assurance of the company and be later deployed in a real-world car.

The *Methodology* chapter includes a detailed explanation of the process of development that was followed to meet the expectations created. In a first section are explained the the requirements that the project will have to meet once it is finished, followed by its use-cases and respective descriptions. Is also explained how the final product was put to the test, what architecture was initially defined and what technologies were used and why.

The Project chapter will explain thoroughly the development of the various pieces of software that this project contains, containing details from the implementation, diagrams and some views of the final product.

The *Work Plan* chapter will give an overview on the work intended to be done on the next semester, consisting on a list of milestones, a Gantt diagram distributing these same milestones over the semester in the form of monthly tasks. Some implications to the project's development are also described in this section.

The *Conclusions* chapter contains some reflections and opinions about both major and minor situations encountered during the development of this projects, as well as some visions we have about the future of this project.

The *Bibliography* contains references to all the documents and websites visited during the development of this document.

Lastly, the *Annexes* chapter contains documents and information that might be interesting to consult along the reading of this document but are not relevant enough, for example, the list of all *GENIVI* members.

Chapter 2

State of the Art

While concept of *Connected Cars* may be only surfacing lately, it has been around since almost two decades ago. Back in 1998 researchers of *Daimler-Benz Research and Technology* [1] already had the vision of what they called the "*Internet Car*":

"An Internet Car is one which is like any other node on the Internet. Although it is highly mobile, it uses Transmission Control Protocol / Internet Protocol (TCP/IP) to communicate with the other nodes on the Internet. An Internet car can be an Internet client as well as an Internet server. The car in essence becomes an open platform for services to be delivered over the Internet."

Little did they know about how right they were going to be, in a time where wireless bandwidth reached values of 19 kbit/s. As expected, technology evolved and concepts like this started to gain form. Not many years after, it was possible to have remote vehicle diagnostics, a feature that not only saves companies lots of additional costs in recalls and repairs since the companies were able to tell if the car had any problem, remotely. This was certainly a huge breakthrough. In fact that is so big that, if we look at the number of ECU's installed on a vehicle a few years ago, they were meant for crucial tasks such as fuel injection or cut the power to the engine in case of accident. Nowadays, the number of ECU's installed rocketed, with some vehicles having well over 100 ECU's and rising. However, the tendency will be to reduce this number, since the greater the number the greater the risk of failure, as well as making maintainability a real challenge.

Today, we live in a somewhat *app-driven* society where there is an app for pretty much everything. This made automakers seriously re-think their strategy on how to strike the market, which led to small, but useful, features like answering a phone call without taking the hands off the wheel. Now, smartphones are here to stay and many users seek a similar experience while driving since the need to stay connected digitally is getting bigger and bigger. This led to a significant increase in the demand for smartphone-like features on vehicles. So far, simple features like sending voice or text messages, music on demand, or even more complex ones like Global Positioning System (GPS) real-time positioning and path calculation have become somewhat of a standard. But it does not end here. Many companies focus is on enhancing the driving experience using only the vehicle. This means cars will have their own Operating System (OS) running, and consequently their own set of applications as well as the tools to allow the development of more.

According to a press release in 2014 [21], major economies like *eCall* in Europe, *GLONASS* in Russia and *Stolen Vehicle Tracking* in Brazil have already introduced telematics mandates. These mandates imply that information from the vehicle is used, for example, to track a driver's behaviour before an accident, or even call an emergency number automatically when an accident is detected. As the future generations will rely more and more in cloud-based back-end systems, this market research report analysing the supply chain of the of connected cars gives pretty clear expectations to this market in the near future, giving special attention to the continuous increase in value that is expected to reach more than $46 Billion by 2020, at a Compound Annual Growth Rate (CAGR) of 10.82%.

In this chapter we will give an overview of the some of the existing alliances and a few of the most recognised middleware frameworks, such as OpenCar and an explanation about how their services work and how they stand when compared to OpenCar.

2.1 Organisations

Since cars started using so much software, they became software engineering problems as well. Like any other software project, making partnerships is often a rewarding choice.

In this section we will give an overview about the existing organisations as well as their goals, partners and work-flows.

2.1.1 AGL - Automotive Grade Linux

In this section of the document will be given an introduction of what AGL [20] consists and what it represents for the connected car market as an open-source community.

Overview

Automotive Grade Linux is a Linux Foundation workgroup currently working on open-source software solutions for vehicles, more specifically, IVI systems. Although this was their primary objective, AGL will also support other applications like instrument clusters or telematics systems. Apart from active participants from automotive industries, AGL also welcomes any independent developer to their project.

By using Linux Kernel and lots of other open-source projects, AGL is capable to keep up with the demand of better IVI systems for the automotive suppliers, a problem that has been around for too long, and since it is running Linux, there are many open-source software developers willing to maintain it, as well as to improve it.

The main objectives in mind for the creation of the AGL Workgroup were mainly to create an healthy environment for either developers or Original Equipment Manufacturer (OEM)'s to develop new ideas for IVI systems, as well as share their own in order to improve each other, as an open-source community. The choice of using Linux means that there is already many pieces of software that can be used as well as there are many developers already acquainted with it.

Similar to other organisations, AGL Workgroup offers an open, collaborative environment that allows a more direct contact with entities closer to the industry like Automotive OEM's and Tier One suppliers, as well as their semiconductor and software providers. AGL also made available an embedded Linux distribution to hasten the prototyping process, allowing developers to quickly test and have a minimum of quality assured before presenting the product.

Architecture

As can be seen next, AGL's architecture [20] consists mainly in five layers: App/HMI, Application Framework, Services and Operating System, each of which will be described in the next sections.

App/HMI Layer
This is the most high-level layer in the system, being the one that will interact directly with the driver. This layer is responsible for handling all the user's requests, being them either through voice commands or pressing buttons, and will enable functions like phone calls or more complex ones like navigation. It will also be responsible for holding all the respective business logic for the application in question.

AGL gives the liberty of having applications that use a web based framework or a native framework, which does not imply that the developer has to choose one: more than one framework can be included in a system. The coordination of these applications among frameworks will be performed by the Application Framework, which is described in the next section.

Application Framework Layer
Like it was described in the previous section, this layer is mainly responsible for coordinating the applications in the upper layer, providing access to any service needed for the applications and their respective interfaces to work.

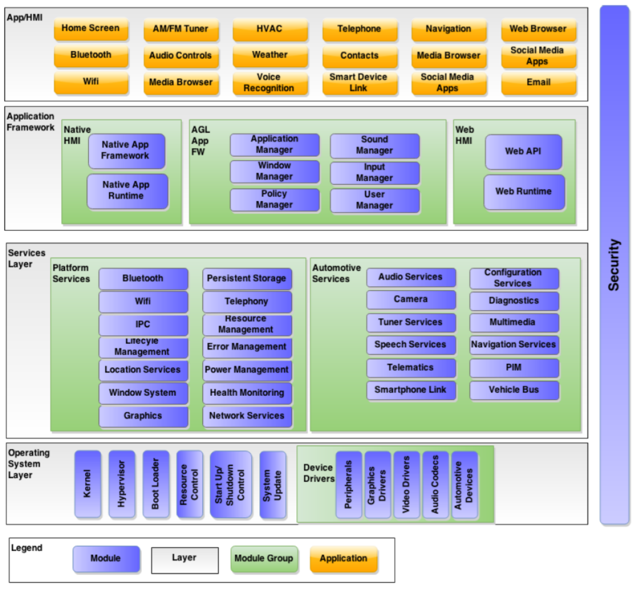


Figure 2.1: AGL Architecture Diagram

As mentioned before, developers are not confined to using a single framework, it is possible to use more than one. The application layer will contain all the code specifically written for that framework. Besides this code, it will also contain all the necessary components to access the lower-level layers in order to serve the application's needs, regardless the framework used.

Services Layer

Following the diagram, it is safe to say that there are clearly two kinds of services in AGL: platform services and automotive services. In a general way, this layer will serve the upper layers via Inter-Process Communication (IPC) type interfaces or subroutines from a given API. These interfaces remain unaltered in a given implementation and it is the Application Framework Layer's job to provide access to these same interfaces for the applications running on the top layer, the App/HMI layer.

Platform services will serve the applications needs in terms of software resources, such as Bluetooth or Wifi connection, location services or even persistent storage needed for an application to run correctly.

Automotive services are more related to what it is going on in the car. These services provide a vastly great variety of information, being it derived from diagnostics or even directly from the car's buses, serving the application with crucial information about, for example, telematics.

Operating System Layer

As mentioned, AGL makes fully use of Linux Kernel. As any other operating system, this means that everything going on the IVI system of the vehicle is generated by this layer. Everything from graphics to processing information coming from the car's buses.

AGL also grants a new release of the same kernel every sixty days, which means that the system is constantly evolving. However, this does not imply good news only. Since cars usually have a design cycle of 4-6 years for IVI systems, it is wise to to be careful when updating the system. There are features and products designed by the open-source community that need to be supported, and there is also the need to update the operating system. There must be a balance.

Like any other project from Linux Foundation, this one has a Linux distribution of its own: Long Term Support Initiative (LTSI) kernel [27]. Currently, this distribution is the only open-source kernel that gets really close to the automotive industry's needs. It already includes multiple automotive -driven components, and since it is fully aligned with Linux LTS it takes advantage of its features and security level.

It is also worth mentioning that LTSI allows for a much more greater set of components and bug-fixes than any other Linux distribution, and it is carefully validated manually with the help of tools to try and ease the process, assuring a greater level of reliability.

2.1.2 The GENIVI Alliance

Before starting with OpenCar, it is important to know what motivated OpenCar to be what it is today. This alliance brought together a great number of companies with the purpose of providing an open environment to enrich the connected car market and already made joint projects with other major stakeholders like AGL and AUTOSAR. This section will give a brief introduction to this alliance.

Overview

The GENIVI [3] Consortium was established in 2009 and is a non-profit organisation aiming to introduce and maintain an open, Linux based, infotainment and connectivity platform for the general transportation industry. It successfully introduced the concept of open-source development to the connected car market, which allowed for IVI projects to be more flexible among vehicles while satisfying the costumers' requests. This greatly reduces the challenges automakers face when delivering the latest functionalities, keeping their customers happy. Besides this flexibility, it also took upon itself all the advantages that come with the open-source concept, such as reusable components and the redeployment of already developed solutions. GENIVI also provides many standards and open-source references.

Besides encouraging the joint development of new solutions, GENIVI also has technical deliverables made available. These deliverables vary from individual software to standard API interfaces or even IVI architectures, all of them benefiting from open-source software.

So far over 140 companies have joined, mainly companies based in Europe. However, not all of them are exactly automakers. GENIVI's members consist mainly in automakers but companies from sectors like electronics or communications that showed interest in the prevail of the IVI systems have also joined.

Compliance

Currently, the GENIVI Alliance has a program that allows its members to participate in its compliance activities. To do so, companies submit their own IVI platform that will then endure a full review process in order to assure if it is really ready to be considered a GENIVI compliant product.

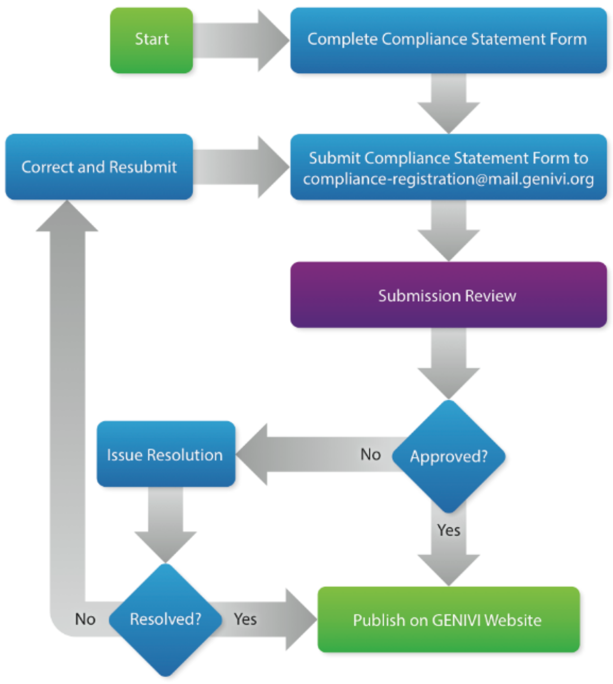

Start
Complete Compliance Statement Form
Correct and Resubmit
Submit Compliance Statement Form to compliance-registration@mail.genivi.org
Submission Review
Issue Resolution
No
Approved?
Yes
No
Resolved?
Yes
Publish on GENIVI Website

Figure 2.2: GENIVI compliance approval process [18]

As straight-forward as the image above may seem, this process is more complicated than that. Once submitted, the GENIVI's assigned reviewers have to clarify every single question they may have with the applicant, and if found any they must correct it (hence the step for "Issue resolution"). Once the reviewers conclude the applicant's platform meets all the requirements, it receives a green light to be published on GENIVI's website and to benefit from all the perks of being a a GENIVI compliant platform.

Architecture Overview

Even not offering a complete IVI solution and a given design, GENIVI does deliver a set of tools that lets current IVI solutions like Tizen and OpenCar make their choices in what components to use.

In the next Fig. [2] below is possible to see which components GENIVI offers as middleware (yellow and purple sections) to any GENIVI compliant IVI solution.

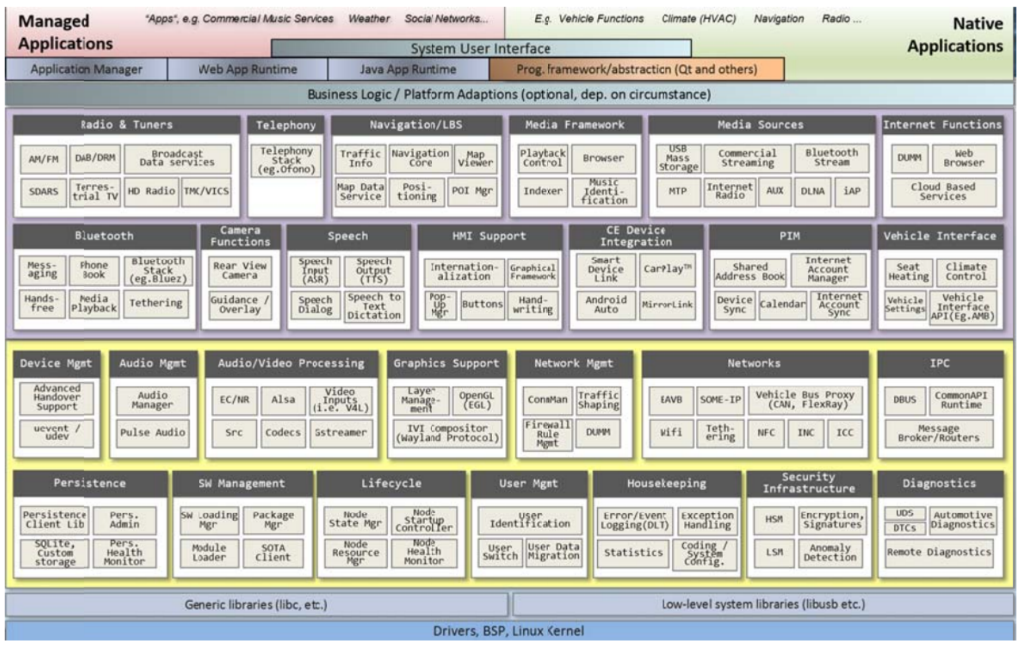


Figure 2.3: GENIVI Architecture

Both yellow and purple sections are considered middleware, GENIVI's main focus, but there is one slight and meaningful difference. While components represented on the yellow section are available in the Open-Source Community (some of them are still in being developed in GENIVI's Open Source Projects section), the components represented in the purple section may need commercial software components in order to be used.

Another point to be noted in the diagram is the bottom layer. GENIVI standardisation of software relies greatly on Linux Kernel. This said, any IVI solution that is GENIVI compliant will be, ultimately, running Linux in the background.

Members

As was mentioned before, there are over 140 companies already in this alliance. It is possible to find a list of these companies in the Annexes of this document, listed in an industrial view as Original Equipment Manufacturers (Table A.1), First Tiers (Table A.2), OSV, Middleware, Hardware, and Service Suppliers (Table A.3, OpenCar is listed in this sector), Silicon (Table A.4) and others (Table A.4) that do not categorise in any of the above but are still partners.

2.1.3 AUTOSAR

In comparison to GENIVI there are a few similarities worth mentioning about AUTOSAR, being them the running OS and a few aspects in the architecture.

Overview

Much like GENIVI, AUTOSAR [5] was also born from a partnership between multiple companies that took the challenge to establish multiple standards for the automotive industry, being the first ones BMW, Bosch, Continental, Daimler, Chrysler and Volkswagen in 2002.

This alliance is mostly formed by OEM manufacturers and Tier 1 automotive suppliers, all motivated in standardise IVI systems in a way that they will be easily integrated, modified and updated, making the challenges faced by automotive companies when providing good IVI systems a whole lot easier.

So far, AUTOSAR has been successful to define a basic software architecture, structured in layers, which allows the encapsulation of hardware dependencies. This also made possible the integration of new software modules as well as their functional reuse, which is an added value to the partnership increasing the agility of the process of development. Another great feature of AUTOSAR are the standardised interfaces. By using API's to separate AUTOSAR's software layers one can encapsulate functional software components more easily. Just like the interfaces, all the rest both the software and the Run-Time Environment (RTE) implementation specification were also standardised in order to assure bus compatibility and the application itself.

Last but not least, there is the RTE (Run-Time Environment). Unlike GENIVI or AGL, AUTOSAR provides inter- and intra-ECU communication across the whole vehicle network, not necessarily running Linux, that sits right in the middle of the software components and the software modules. However, this implies that all entities connected to this environment respect AUTOSAR's specifications, or else nothing will work properly.

Architecture and Components

As we can see in Fig. 2.4 below, the differences from other platforms of the likes of GENIVI or AGL are clear. The first to be noticed, and probably the biggest, is the AUTOSAR Runtime Environment connecting the software to the hardware instead of a Linux Kernel as OS of choice.

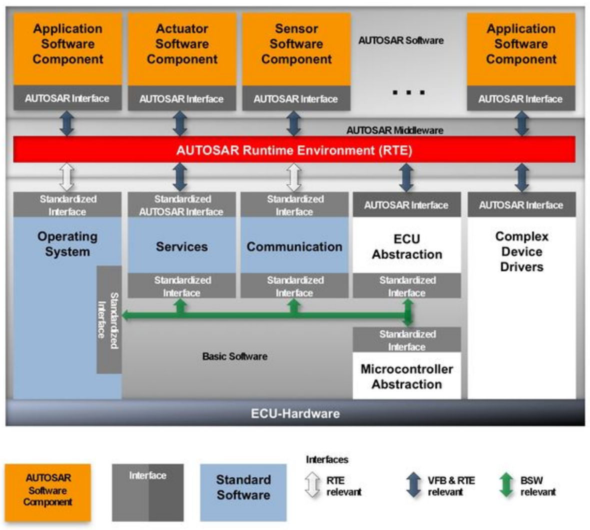


Figure 2.4: AUTOSAR architecture overview [6]

Starting from the top, there is the Software layer. This layer holds all of the software components currently mapped on the ECU. All the interaction of these modules to the hardware interfaces is done solely through the RTE, via an AUTOSAR Interface designed for this purpose that is embedded in the software component.

Connecting the Software layer closer to the hardware there is the RTE Layer. This layer will act mostly as a communication centre for inter and intra-ECU information exchange. This creates an useful abstraction to all AUTOSAR's software components in the sense that all the traffic going on Control Area Network (CAN), Media Oriented Systems Transport (MOST), Local Interconnect Network (LIN) buses of the car become available through a simple interface standardised according AUTOSAR's requirements. However, the applications in the software layer have their own communication requirements, which may cause the RTE layer to be adjusted for certain applications to work. This implies that RTE's may differ greatly from ECU to ECU.

Lower in the architecture it is possible to see the Basic Software Layer. This layer is the one that is closest to the hardware, and the one to provide the RTE all the information the software applications above it requested.

In this layer there is all the standardised software that are specific to the ECU where it is installed as it only makes certain services available such as *Communication*, *Operating System* and *Microcontroller Abstraction*.

Finally, the interfaces. As it can be seen in the diagram above, it is possible to count three different types of interfaces: *AUTOSAR Interface*, *Standardised Interface* and *Standardised AUTOSAR Interface*. The names may be similar but their functions differ.

Standardised Interface
This interface is needed only to serve other standardised components, which is why it appears mostly in the bottom layer, connecting all the Basic Software components.

AUTOSAR Interface
This interface is responsible to describe all the data and services required by an upper component and is implemented according to the AUTOSAR Interface Definition Language. This kind of interfaces are partly standardised, which means that they may or may not include OEM specific requirements. Another advantage of these interfaces is the fact that they allow multiple software components to be distributed among different ECU's, which is a pretty transparent process since the RTE will take care of it.

Standardised AUTOSAR Interface
The difference to the previous mentioned *AUTOSAR Interface* is the fact that these are also standardised in AUTOSAR. These kind of interfaces split into two different categories: the ones used to define AUTOSAR services, which are also standardised (Basic Software Layer) and the ones derived from AUTOSAR Application Interfaces.

2.1.4 Summary

Among these three distinct organisations, the spotlight goes obviously for GENIVI and AUTOSAR. These organisations managed to bring together a lot of companies to work with one single focus: to standardise software for the automotive industry. Even from a market point of view, these companies are a safe bet since the support is enormous and the community is constantly growing.

As for AGL, it seems to have appeared for the single reason that Linux Foundation saw the automotive industry leaning for Linux as OS of choice and decided to have one of their own and jump to the front line by force. However, the support is mostly with GENIVI, an organisation that covers pretty much the same aspects as AGL.

In essence, GENIVI and AUTOSAR are the way to go in automotive software. They are made of fruitful partnerships working together to improve each other, which is the logical thing to do.

2.2 Middleware Platforms

Platforms for developers to come together and develop new solutions have been slowly appearing. Like OpenCar, the main objective is primarily to create a good environment for developers to make their ideas a reality, in a community-based ecosystem.

This section will give a detailed overview of OpenCar and a briefer of its main competitors.

2.2.1 The OpenCar Platform

This platform's main purpose is to not only standardise the access to vehicle sensor data, making it available and easy to use for developers to produce usable and rich applications for that same centre console, but also provide an useful and interactive environment for programmers to independently produce software and have it certified by OpenCar itself. The following sections will explain its main objectives and how they are achieved.

Overview

Firstly created by Jeff Payne back in 2011 and currently an active member of the GENIVI Alliance and the World Wide Web Consortium (W3C), OpenCar has come to build the industry's first updateable application platform and developer ecosystem, to serve the automaker's need for highly integrated, car-centric applications. Currently, OpenCar is partnering with various automakers, suppliers and developers in order to define new standards and tools, transforming both automotive App development and driver experience.

Since the OpenCar platform is rather new, we will start by shortly introducing this technology and associated entities.

INRIX

This company has been in the vanguard of the connected cars market for nearly a decade now. Its main focus and reason to exist is very simple, yet very logical. In a time where drivers and departments of traffic were struggling to acquire traffic information about its condition in real-time by installing expensive sensors in a few roads, INRIX took a different approach.

Instead of using expensive sensors that were both expensive and hard to maintain and install, INRIX suggested that this data was extracted from the vehicles themselves. This way it was possible to have easy access to traffic condition data in real-time, and pretty much everywhere since there was no need to install sensors on the roads, extracting data directly from the vehicles would cover a lot more ground in a more effective way.

With the recent acquirement of OpenCar, INRIX hopes to extend its *Autotelligent* platform to provide an open solution for manufacturers to an even wider portfolio of content, without the need for a smartphone or any other external hardware.

Autotelligent
This INRIX product consists in a cloud-based machine learning platform that works in the background and monitors road conditions before the user starts driving to determine the optimal time to leave and which route would be better considering traffic condition, which is calculated in real-time, therefore adjusting the best route to reach its destination.

It will also give special preference to routes already travelled by the user and "learn" the ones that the user uses more often when calculating routes.

OpenCar Connect

OpenCar Connect was created with the purpose of being the next-generation platform to build, integrate and host automotive-grade applications in a connected car.

OpenCar features an open software development environment backed by HyperText Markup Language revision 5 (HTML5) standards. This provides the advantage of benefiting from the power and diversity of the desktop and mobile app development ecosystems, bringing them to the automaker segment. Besides this, OpenCar also gives developers access to automaker-approved vehicle telematic data, something that has been out of reach so far, allowing them to build rich telematic-driven apps, enhancing even more the driving experience of the users.

How does it work?
As was mentioned before, the framework works through an amount of layers, which can vary depending on where the software is running, either a real vehicle or a simulator. These layers interact with each other in order to connect all the available information in the system.

As we can see in Fig. 2.5, once the the user makes an input, with it being buttons, touch screen or even voice control, this input goes through the corresponding service. This service will be running on the HTML5 Runtime, which is responsible to pass it on to the integration layer, which is in direct contact with the vehicle hardware.

This hardware mainly consists in buses that deliver information about what is going on the vehicle, being the most important the **CAN bus**, a standard, message-based protocol for vehicles, designed to allow micro controllers and other devices to communicate with each other, removing the need to have a host computer, the **MOST bus** a high-speed multimedia

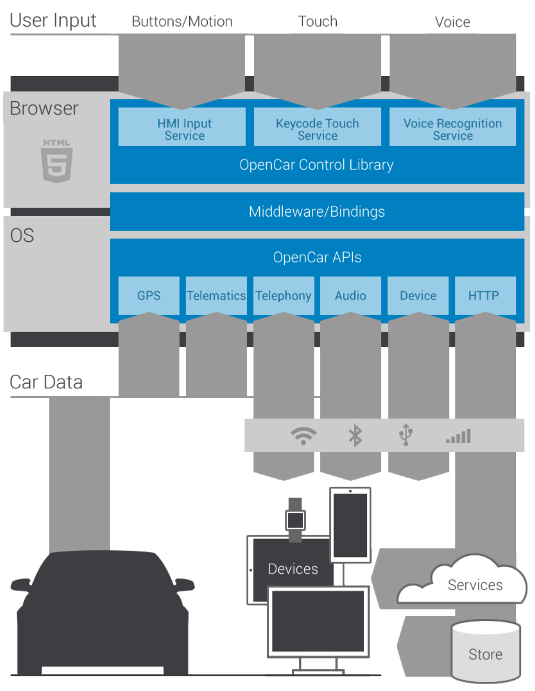


Figure 2.5: OpenCar Connect Overview

network technology responsible for applications running inside or outside the vehicle. It does not differ much from the CAN bus, although MOST is much, much faster, what makes CAN unable to compete, therefore these two being used in in distinct purposes. Finally there is the **LIN bus** that consists in a serial network protocol used for communication between components inside a vehicle. Once again, not much different from CAN in terms of functionality. However, the need for a cheaper option than CAN was growing and some manufacturers started using different to connect the components.

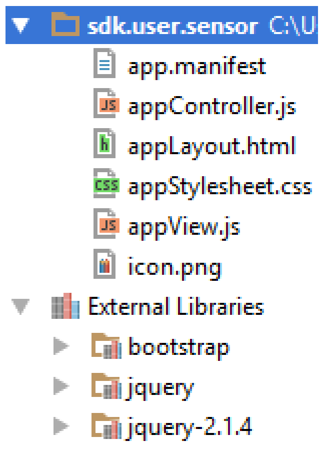
Apart from these vehicle-specific components, it is also responsible for communicating with more generic components, such as GPS, HyperText Transfer Protocol (HTTP) connectivity devices, bluetooth and so on. The integration layer communicates with this hardware through the multiple API's OpenCar provides, making the application able to access telematic information, data from external servers, connect with a nearby device or even store data.

Framework Design Pattern

OpenCar Connect app framework offers an easy and safe way to develop apps for connected cars. In this section will be given a brief overview of its design pattern and architecture.

This framework runs in an HTML5 run-time, being its simulator simply an instance of Google Chrome, allowing for easy debugging and logging. Each OpenCar application runs in two widely separated portions: **Business logic** contained in a *controller* and **Presentation logic** contained in a *view*

As you may have noticed already, the platform follows, in a way, the Model-View-Controller (MVC) pattern.These containers are common Javascript files within a *manifest* file for each application. Besides Javascript, there is also additional HyperText Markup Language (HTML) and Cascading Style Sheet (CSS) files, as well as support for other resources (external modules, libraries).



```
▼ sdk.user.sensor C:\U
    app.manifest
    appController.js
    appLayout.html
    appStylesheet.css
    appView.js
    icon.png
▼ External Libraries
  ▶ bootstrap
  ▶ jquery
  ▶ jquery-2.1.4
```

Figure 2.6: Files included in an OpenCar app

As you can see in Fig. 2.6, the framework allows an easy and effective way to structure the code. Going file to file, we have:

- *app.manifest*

  This file is responsible for storing the app's information, such as id, which file is the controller, view or template, as well as managing its permissions. For example, in order to gain access to the vehicle's Telematics a permission needs to be added in the app's manifest, or else it will not work at all.

- *appController.js*

  Following the framework's architecture, this Javascript file is responsible for all the business logic of the app, calculating and retrieving data to the view, for example.

- *appLayout.html*

  This HTML file is responsible for the apps presentation, not much different from one of a website. It supports templating, as well as external libraries such as Bootstrap.

- *appStyleSheet.css*

  This file works together with the HTML file described previously and is responsible for enhancing the visual look and feel of the application, giving a lot more flexibility when designing apps when compared to simple HTML.

- *appView.js*

  As the name suggests, this Javascript file is responsible for all the view logic of the app. It is from this file that the HTML will constantly receive updates with new information.

- *icon.png*

  This little image file is nothing more than a logo to be shown in the car's console when browsing apps. The framework gives the liberty of having our own logo and showing it in the apps menu. When creating an app we are given a default logo, although it can be easily replaced.

- *External Libraries*

  As was mentioned before, OpenCar allows external modules and libraries. As shown in Fig. 2.6, this particular app is taking advantage of *Bootstrap* and *jQuery* libraries.

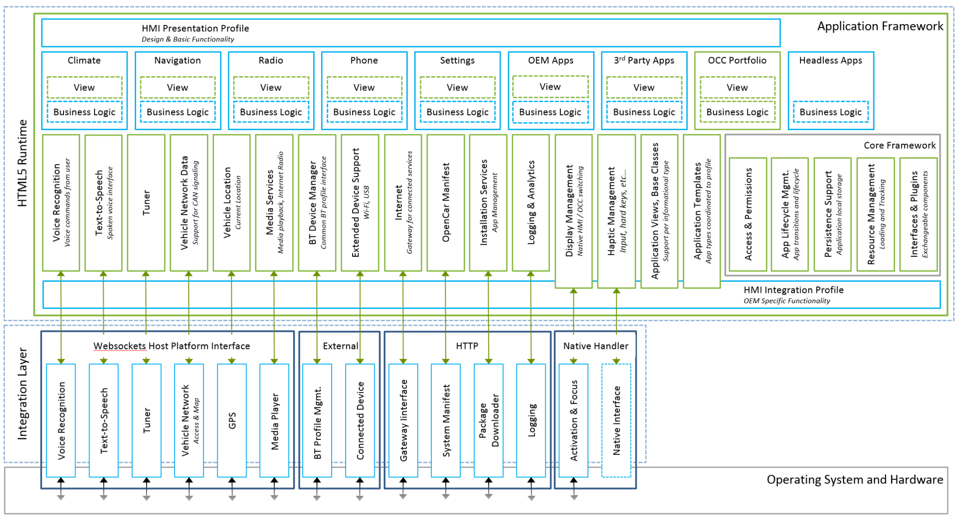


Figure 2.7: Framework Architecture

Integration Layer
As we can see in Fig. 2.7, this layer is responsible for making available services from the native host environment. In an actual vehicle, these services would include access to CAN, MOST, LIN buses and more. Besides these services, there are some that cannot be served through the Javascript layer, these ones will be provided directly as Native Handler, WebSocket Host Platform interfaces, HTTP for gateway service to the Internet and so on.

LocalHost HTTP Layer
In order to provide a more interactive development environment, OpenCar SDK offers a a LocalHost HTTP service that provides a host server to the browser so it can successfully deliver the framework's contents and applications (not present in Fig. 2.7).

Since most developers will not have access to a vehicle running OpenCar, there is not really a "Native Host Layer" to test on, as there would be in an automotive integration on an actual vehicle. It also does not take advantage of the supplied host services such as speech, audio or persistence. Instead, it uses open-source components like *SQLite* or *GStreamer* that would later be implemented in a vehicle, being in many cases applied such solutions in actual vehicles.

This abstraction allows developers to finish an OpenCar app using only the simulation and testing through the current workstation and make it ready to easily being integrated into any automaker platform without major changes to the code.

HMI profiles
An important thing to understand about this framework are definitely the HMI profiles. At this point, the OpenCar Simulator offers two distinct profiles to use and switch between, *Oxygen* and *Hydrogen*. However, it is not limited to just a couple of profiles. More profiles are also made available through *OpenCar InsideTrack* mentioned above. These profiles can be specific to automakers or vehicles allowing a specific look-and-feel for each one, as well vehicle data and native system integration.

The profiles shown in Fig. 2.10 will dictate the functionality of an app according to each one's implementation. Although every profile will support all OpenCar API's, the behaviour of a given API may differ greatly depending on the profile selected. An easy example would be the styling and positioning of buttons. This implies that before developing an OpenCar app there is a tough choice to be made: *will the app be* ***profile-dependent*** *or* ***profile-independent****?* Making the app *profile-independent* will imply special care when coding the app, since it will have to be very well structured to maximise the use of OpenCar "Chambers" (in more practical terms, OpenCar's visual elements).

OpenCar InsideTrack
InsideTrack was created with the purpose of providing automakers and content providers a platform in which they can build apps for new cars and

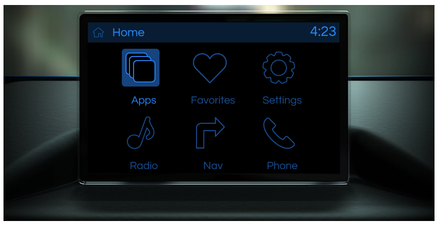


Figure 2.8: Oxygen

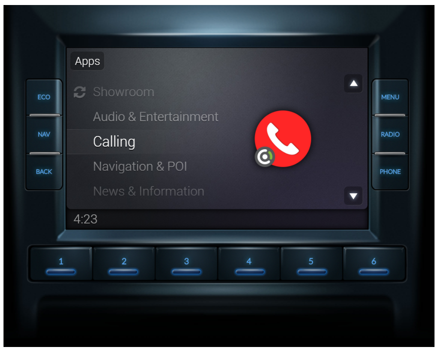


Figure 2.9: Hydrogen

Figure 2.10: Default HMI profiles in OpenCar Simulator

maintain them once they are deployed, which can be done remotely over the entire vehicle life cycle.

It takes the form of a usable online Web platform that gives the option of creating an OpenCar project and submit it for evaluation through the SDK (which is also available for download in the platform), or joining an existing project to work on.

Aside from projects, it is also possible to join a Program. Programs are meant to bring automakers and system integrators closer in order to put together and manage sets of applications for the global market. It allows direct contact with developers, as well as track their status ans securely share files and simulator plugins.

Also available in the InsideTrack platform is detailed information about the usage of the framework and how it connects with the vehicle, including diagrams and brief explanations. When it comes to support the developers, the platform provides documentation needed for understanding each API contained in the OpenCar framework as well as examples of how to use them, separated by SDK versions.

The platform was fully used during the making of this dissertation, being its final result intended to have a functional project submitted.

2.2.2 Tizen

Since Tizen has been integrated into AGL, we believe it is worth mentioning this entity as well as what it represents. According to Jaguar/Land Rover's System Architect **Rudi Streif**, the integration of Tizen into the AGL project was a good starting point [11], leaping AGL to about 50% [10] of where it needs to be.

Overview

Tizen is essentially an open-source, standards-based software platform that allows developers to reach multiple ranges of devices, which can vary from smartphones to smart TV's, being these types of devices classified in different profiles.

Tizen claims to be the "OS of everything", bringing a new user experience across different devices. This allows the user to be connected when outside the car, inside the car, even when watching TV. Once its development is also open-source driven the support is global, which not only complies perfectly with AGL but also allows for a more swift improvement over time.

Tizen IVI

Tizen IVI [26] is one of the many "profiles" mentioned before. As the name suggests, it is directed to the connected car market, offering a platform identical to many others like OpenCar with a Free and Open Source Software (FOSS) OS. This not only allows for applications to be ran into vehicles but also extending it to other "profiles" like smartphones.

In terms of coding, it does not differ much from OpenCar itself by also taking full advantage of the HTML5+CSS+Javascript trio.

2.2.3 QNX Car Platform

QNX [22] is a real-time Unix operating system that has been around for quite some time now, mainly powering critical systems like air traffic control, road signs, medical devices and even nuclear power plants over the past thirty four years. Recently emerged *QNX Car*, an attempt to bring this very system to our everyday vehicle. It is also worth mentioning that QNX is also a *Blackeberry* subsidiary since 2010 [12].

Overview

This platform [23] offers a range of already integrated and optimised technologies from QNX Software Systems and many more partners from their own ecosystem. Similarly to the already mentioned Tizen platform, this one was also designed with maximum flexibility in mind providing developers with lots of options for building IVI systems that keep being updated regularly and reach more mobile devices.

Similarly to other platforms, QNX also features many SDK's (being one of them dedicated to IVI systems). This allows fast-development and reduced testing once the developers are able to test them before deploying the software on an actual car, which also results on reduced costs and risks. Besides this, also has support for Wifi/smartphone connectivity as well as technologies that make possible the development of rich and user-friendly HMI's.

Technology

As expected, QNX made available quite a few technologies that were already developed, and "simply" added a few quality requirements like crash resiliency, embedded optimisation and, of course, fast boot.

Among these technologies the ones proven to be pretty useful in IVI systems are the QNX Neutrino real-time OS, a top-class acoustic echo cancellation and noise reduction, a multimedia an application framework SDK for a range of environments, as well as an HMI framework supporting Qt, HTML5 and others, not much different from other names like Tizen or OpenCar itself. It is also worth mentioning that QNX currently supports smartphone-based solutions like CarPlay and Android Auto, which will both be explained more thoroughly in the sections to come.

In Fig. 2.11 we can see how these technologies connect each other in a given IVI system.

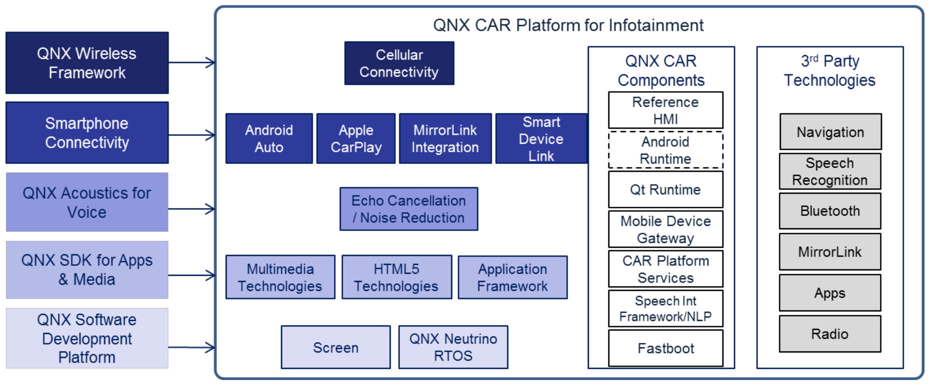


Figure 2.11: QNX technologies [23]

Ecosystem

Similarly to partnerships like GENIVI or AUTOSAR, QNX also has some partners of its own. The platform makes available some implementations from the multiple partners themselves, allowing them to speed up the development and integration processes by using them.

The list of partners can be seen in the Annexes section of this document, on Table B.1.

2.2.4 Summary

Since the technologies described previously are but a portion of all the existing ones, it is safe to assume that there are quite a few options in middleware. All of them are very alike in the most important aspects, what makes the choice for one a difficult task.

Ultimately the choice for using OpenCar resided on the simplicity of the framework in terms of code, the quality of the simulator and an aspect easy to forget: the documentation. Besides being a rather new technology, the documentation and examples are well written and cover the most important aspects, leaving to the programmer the sole tasks of knowing JavaScript+CSS+HTML5 and coming up with a great idea.

The fact that OpenCar is also an active member of the GENIVI was also a great reason to keep the technology, since it will be receiving support, now and in the future.

2.3 Smartphone-Based solutions

There is a strong reason as to why this kind of solutions have been gaining terrain over more in-car solutions like the ones already discussed. According to TechRadar [25], cars are not as easy to develop as personal tech devices. By the time a car is finished, the infotainment system installed in the car probably is already severely outdated.

Putting this in more practical terms, an average car gets a system refresh every 2-3 years, and probably a completely new model in 4-6 years. Not to mention luxury cars, this kind of cars can last 10 years without a new system.

However, this does not happen with smartphones. The time between new models is around a year, 2 at most, and the updates are much easier. With this in mind, many solutions regarding the potential of smartphones inside the car started climbing through the market and are now available in multiple cars. This section will give a short introduction to some of the major players in this kind of approach.

2.3.1 Android Auto

Android Auto is, as the name suggests, Google's approach on the connected cars market. As will be explained in this section, it takes full advantage of their already dominant operating system Android.

Overview

In an attempt to make use of the centre console of the car, Google took advantage of its already developed operating system, Android, applying it in a numerous variety of cars in the form of Android Auto.

Similarly to Android, Android Auto benefits from the same support as the original operating system in terms of platform and documentation.

There are already multiple apps originally designed for Android that were adapted to Android Auto, as well as plenty of online documentation of how to give Auto features to a given application.

There is, however, a huge difference from the technologies already discussed in the way the car interacts with the driver.

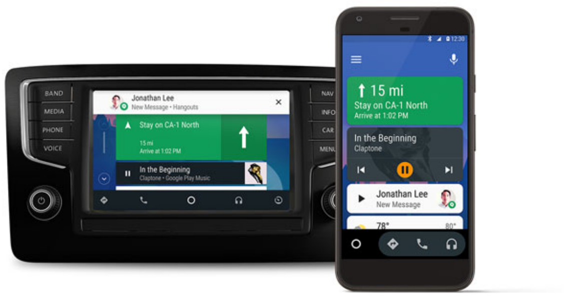


Figure 2.12: Android projecting information in Android Auto [19]

As Fig. 2.12 suggests, Android Auto is not actually a independent runtime like *OpenCar*, instead, the apps are mere projections of the apps on the user's Android smartphone.

How it works

Firstly, the driver must have the *Android Auto* app installed on the smartphone. The app is available through the Google Play Store just like any other Android application.

When inside the car, the driver is able to connect his smartphone to the car, either via Universal Serial Bus (USB) cable, Bluetooth or WiFi, although the latter is still pretty nonexistent. Once that is done, the central console of the car will show a "smartphone-like" interface that allows the driver to use the smartphone through voice commands or buttons on the steering wheel, allowing the driver to stay connected much more safely.

Answering a phone call while driving becomes as easy as saying so inside the car. So far, the main features of Android Auto also include *Google Maps*, *Google Play Music* and *"OK, Google"* [19].

Android Auto enables the user benefit from these Google's services while inside the car, being able to use them in the centre console of the car. However, there is also the possibility for third-party applications to pipe their information through Android Auto to the car, although, they will be more of a read-only application. A practical example of this are messaging apps like WhatsApp or Skype: the user will not be able to interact directly with it through the car's commands, but will be able to receive the message.

In short, the all the car does is solely show information originated by the smartphone, being it the one to do all the heavy lifting [8]. There is currently no way to benefit from Android Auto without an Android smartphone that does not have the Android Auto application installed.

This not only constrains the user to own a certain kind of mobile phone, but also to a very specific operating system since the smartphone HAS to run Android and be, at least, in version 5.0 [19].

2.3.2 Apple CarPlay

Much like the previously discussed Android Auto, Apple CarPlay is Apple's approach on the connected car market, taking fully advantages of Apple's services, as will be discussed in this section.

Overview
Much like Android Auto, Apple CarPlay takes on the task of enhancing the driver's experience through its already developed operating system, iOS. It is safe to say that Apple CarPlay and Android Auto are as much opponents in the connected car market as they are in the smartphones market, making them direct competitors to each other.

As expected, CarPlay also benefits from a developer program, in which is possible to make apps work along with CarPlay inside the car. However, the major difference from Android Auto is the same as the smartphone's: just like the original operating system, iOS, one will need an **MFi** license in order to develop and publish something, instead of Android that allows a much more free community to exist.

Figure 2.13: Usages of CarPlay [4]

As we can see in Fig. 2.13 taken from the Apple's own website, the similarities to Android Auto are obvious, being the main difference the operating system. CarPlay enables the user to use the smartphone through a projection of it on the central console.

How it works
Unlike Android Auto, Carplay does not require the installation of a given application from the Apple Store. Instead, the user simply connects the the iPhone to the car via a Lightning cable (another major difference when compared to Android Auto), and the central console will instantly show an interface similar to the smartphone's, while keeping the main functionalities

of it. The applications will then become available for use through either voice commands or buttons installed in the car.

So far, CarPlay offers an "auto version" of its mains services as well being them *Siri*, *Apple Maps*, *Phone Calls* and *Messaging* app. Although it does not end here. Apple has paved the way for more apps, being them from the Apple Store or manufactured by the automakers themselves, allowing a wider variety of applications to choose from.

However, as was mentioned before, CarPlay consists in nothing more than information coming and going from the smartphone, it is not an in-car system running any kind of iOS. Once again, the smartphone does all the heavy lifting, just like Android Auto. This means that, again, one will need a specific mobile phone running a specific version of a specific operating system, this time being an iPhone 5 or higher.

2.3.3 AppLink - Ford Developer Program

On the matter of smartphone-like solutions, Ford took a very different approach. Instead of limiting the driver to using a single kind of smartphone, it allows the use of both systems. However, there are a few perks worth mentioning about it.

Overview

The feature the separates Ford's method from Android Auto and Apple CarPlay the most is definitely the existence of an embedded system running in the car. This system allows for both Android and iOS to connect with it and enable in-car commands, but in a different way than already discussed.

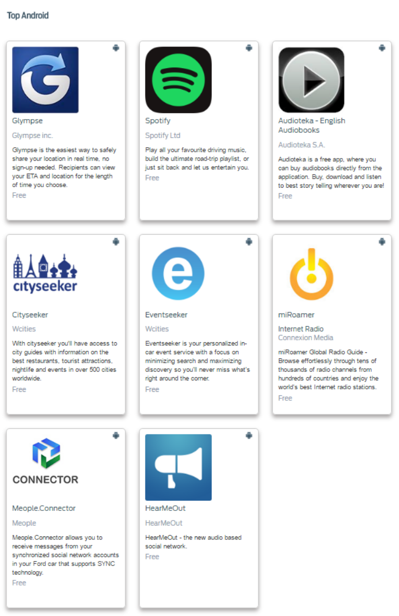
An obvious downside of enabling support for multiple operating systems is the limiting number of apps to use inside the car. Currently, Ford has a catalogue available listing the applications that can be used together with the smartphone, along with every voice and button commands available for the given application.

How it works

Besides the differences pointed out before to other smartphone-like solutions, there is another that is most relevant as well: Ford's infotainment system, SYNC [16], has functionality on its own.

To extend the functionality and the need of the drivers to stay connected, AppLink was introduced. AppLink consists in a set of API's that allow mobile developers to extend a mobile application to the vehicle's HMI. Once connected, the phone will feed information to the centre console, enabling vehicle controls like voice commands, steering wheel buttons and even the touch screen of the console to control a given app.

Although AppLink [14] supports both systems, there are some slight differences. The first one appears when connecting the phone: as suggested by Ford, iPhones must be connected via an USB cable while Android phones simply have to connect via Bluetooth. Once this is done, the user is able to use some of the smartphone's applications through the vehicle's system.



Top Android

Glympse
Glympse inc.
Glympse is the easiest way to safely share your location in real time, no sign-up needed. Recipients can view your ETA and location for the length of time you choose.
Free

Spotify
Spotify Ltd
Play all your favourite driving music, build the ultimate road-trip playlist, or just sit back and let us entertain you.
Free

Audioteka - English Audiobooks
Audioteka S.A.
Audioteka is a free app, where you can buy audiobooks directly from the application. Buy, download and listen to best story telling wherever you are!
Free

Cityseeker
Wcities
With cityseeker you'll have access to city guides with information on the best restaurants, tourist attractions, nightlife and events in over 500 cities worldwide.
Free

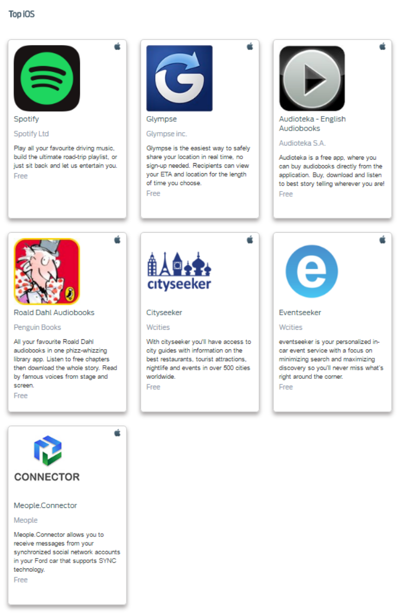
Eventseeker
Wcities
Eventseeker is your personalized in-car event service with a focus on minimizing search and maximizing discovery so you'll never miss what's right around the corner.
Free

miRoamer
Internet Radio
Connexion Media
miRoamer Global Radio Guide - Browse effortlessly through tens of thousands of radio channels from hundreds of countries and enjoy the world's best Internet radio stations.
Free

Meople.Connector
Meople
Meople.Connector allows you to receive messages from your synchronized social network accounts in your Ford car that supports SYNC technology.
Free

HearMeOut
HearMeOut
HearMeOut - the new audio based social network.
Free

Figure 2.14: Android supported applications



Top iOS

Spotify
Spotify Ltd
Play all your favourite driving music, build the ultimate road-trip playlist, or just sit back and let us entertain you.
Free

Glympse
Glympse inc.
Glympse is the easiest way to safely share your location in real time, no sign-up needed. Recipients can view your ETA and location for the length of time you choose.
Free

Audioteka - English Audiobooks
Audioteka S.A.
Audioteka is a free app, where you can buy audiobooks directly from the application. Buy, download and listen to best story telling wherever you are!
Free

Roald Dahl Audiobooks
Penguin Books
All your favourite Roald Dahl audiobooks in one phizz-whizzing library app. Listen to free chapters then download the whole story. Read by famous voices from stage and screen.
Free

Cityseeker
Wcities
With cityseeker you'll have access to city guides with information on the best restaurants, tourist attractions, nightlife and events in over 500 cities worldwide.
Free

Eventseeker
Wcities
eventseeker is your personalized in-car event service with a focus on minimizing search and maximizing discovery so you'll never miss what's right around the corner.
Free

Meople.Connector
Meople
Meople.Connector allows you to receive messages from your synchronized social network accounts in your Ford car that supports SYNC technology.
Free

Figure 2.15: iOS supported applications

Figure 2.16: Ford SYNC catalogue [15]

Fig. 2.16 above shows the entire English version of Ford's App Catalogue for both Android and iOS. As we can see, it is still pretty limited, and even more limited when choosing other languages.

Another downside of AppLink is most definitely it is availability through the different SYNC versions currently being available on Ford vehicles. So far, there are three distinct versions of the software.

SYNC with MyFord (Fig. 2.17) is the most basic one of the three. Even so, it already allows users to complete simple tasks like answering a phone call or listening to music through voice commands. Although it supports AppLink, it does not have support for neither Android Auto or Apple CarPlay. Also, there is no support for an WiFi update to the software.

An upgraded version of the one just described before is **SYNC with MyFord Touch** (Fig. 2.18). Curiously, on this version there is no support at all for AppLink, although there are more features that enhance the driver's experience. It is now possible to adjust climate inside the car, use Ford's own intelligent navigation systems through voice commands. Support for Android Auto and Apple CarPlay is still not supported on this version, neither is WiFi updates capability.

Lastly, there is the **SYNC 3** (Fig. 2.19), the latest and most complete of them all. SYNC 3 brings all of the funcionality of the ones above, while adding more interesting features. AppLink is now fully supported, as well as both Android Auto and Apple CarPlay along with all the features they provide. SYNC 3 also brings to the table the opportunity of connecting remotely to the car allowing the driver to track its location, as well as track its status which include lock/unlock the car, start the engine, check fuel level and so on. To do this however, the driver must have FordPass [17] app installed on the smartphone. Lastly, SYNC 3 also supports automatic updates over WiFi.

Figure 2.17: SYNC with MyFord

Figure 2.18: SYNC with MyFord Touch

Figure 2.19: SYNC 3

Figure 2.20: Ford SYNC's different version's displays [16]

Due to this variety of versions, compatibility is an obvious downside. Since some cars are only available with certain versions of the software (there are also models that are made available with all the versions), this will create a clear constraint on the driver's smartphone choice since there are compatibility issues.

A very practical example: let's imagine the user currently has a Sony Xperia E5 as his smartphone of choice, and just aquired a 2017 Ford Mustang equipped with SYNC 3. Currently, there is no compatibility for his phone with such car, leaving him/her with no choice but to either get a new phone or use the car without the benefits of SYNC 3.

2.3.4 Summary

In a society depending more and more of smartphone's capabilities, there is no denying that using their potential to enhance the driver's experience and keep them connected while driving is a very well played move, as its possible to see from multiple automakers opting for this kind of solution.

In my opinion however, it seems rather limiting since we are only connected to the outside world. Looking at these solutions from a higher point of view, all we are able to do is interact with the smartphone through the car. We do not completely disagree, although we think it would be much more interesting if the driver actually interacted more with the car.

Today's technology allows for technical information to travel through the car's sensors, from a single tire speed to the gear that is engaged in the gearbox. This kind of information is not only interesting to the driver, but also valuable to the right entities. The lack of usage of telematic data in this kind of systems was one of the main factors when choosing OpenCar for this project, which is very complete in that aspect.

2.4 Conclusion

Even if somewhat unknown, the connected car concept has been gaining terrain over the years, providing an ever-growing market that might soon reach a noticeable position among the others. In order to write this chapter we took upon ourselves the task of understanding a minimum of this market and we are glad we did. The amount of diversity in technologies and frameworks is overwhelming, and as if this was not enough the concept itself was already defined a long time ago, suffering architectural modifications over the years.

However, it is also really gratifying to see that many companies have come together to standardise and share knowledge, improving each other's products. This not only adds value to the market, but also gives software engineers and programmers like myself a chance to prove ourselves and, maybe, give something worthy of being successfully integrated, a chance to make cars interesting again.

The choice of making a project using a middleware platform instead of a smartphone-based one was made essentially out of the fact that the smartphone can be quite limited when it comes to connecting it to a car, not to mention that without the smartphone the applications will not have any use.

By using OpenCar, a middleware, OS-agnostic framework, the smartphone becomes a *plus* instead of a *must* which, in my opinion, enhances greatly the driving experience of the user. The application will have an unique GUI, completely independent from smartphone standards and will function on its own, leaving the smartphone as a complementary element to the system.

Chapter 3

Methodology

In this section we will explain more thoroughly what the main objectives of this project are and how they will be achieved, as well as all the requirements and use cases for this software project.

3.1 Requirements

As any other software project, there has to be some requirements to be met by the end of the development process. This section is dedicated to give an high-level perspective of what these requirements will consist.

It is worth mentioning that this system will be a prototype. As so, it might not fully meet a given requirement, specially non-functional.

3.1.1 Non-Functional

Since both the platform and the console application will handle sensible data, some requirements must be met. Once successfully operating, the whole system should meet the following requirements:

- **Security** All the info the user agreed to collect once he used the functionality is considered private. Therefore, a minimum of data security must be assured. In the future, security may not only be digital as it may be subject to some laws since we are dealing with personal, delicate data.
- **Maintainability** With a future deployment in mind, the code should be well organised and separated by modules in order to make upgrading and bugfixing easy tasks. Comments on the code are a must for future reference.
- **Availability** Since the user will want to check information about his driving at any time, the platform should be available at all times. An web-server is advisable.

- **Documentation** Being the technology new, good documentation is required if the project is meant to live on, so new programmers can pick up the work if needed.
- **Fault Tolerance** In order to support he Availability requirement, the platform should be ready to autonomously try to recover from minor errors and keep the connection up.

3.1.2 Functional

In this section are documented the functional requirements of both the console application and the online platform.

Console Application

On Table C.1 we can see all the car's centre console application functional requirements, as well as their respective priorities.

Online Platform

On Table C.2 we can see all the platform's functional requirements, as well as their respective priorities.

3.2 Use-cases

In this section are mentioned all the use cases regarding both the console application and the online platform. Each of them is described concerning all their functionality, being them either user or system related, in the *Annexes* chapter of this document.

3.2.1 Console Application

On the centre console, the user is able to:

- Check information concerning the ongoing trip
- Reset information on Fuel dashboard
- Save information on Fuel dashboard to the online platform
- Get directions to nearest petrol station
- Save 0-100km/h acceleration data
- Estimate 0-100km/h acceleration time
- Change measurement units of console application
- Turn on/off data collection

Also, the application by itself is able to:

- Verify if car is registered on the platform
- Show Telematic Information in real-time
- Register driving events
- Measure 0-100km/h acceleration times
- Register max G-Force during acceleration try-outs
- Suggest best RPM range to change gears
- Show online data

All of these use-cases are carefully detailed in the Appendices of this document, *Appendix F*.

3.2.2 Online Platform

On the online platform, the user is able to:

- Register on the platform
- Login on the platform
- Add car to the platform
- View latest events/warnings
- Analyse Acceleration data for saved measurements
- View full extent of the logs concerning the selected car
- Delete all the logs of a certain event concerning the selected car
- Edit personal information
- View calculated statistics
- View car list
- Switch car
- Delete Car

The platform by itself is also able to:

- Serve the API for the OpenCar application

All of these use-cases are carefully detailed in the Appendices of this document, *Appendix G*.

3.3 Architecture Overview

In this section well be explained every aspect of the architecture of the whole system following Simon Brown's *C4 Model* [7]. In this particular project, there are two possible different architectures to account for, however only one will be followed for the reasons that were previously explained.

3.3.1 High Level perspective

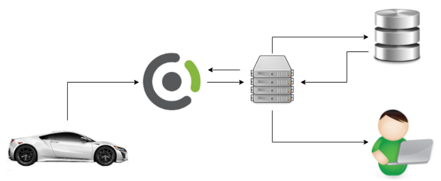


Figure 3.1: Real-world scenario

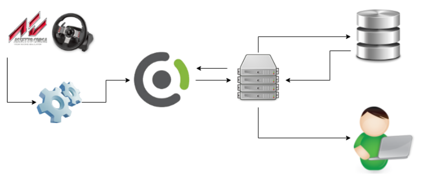


Figure 3.2: Simulated scenario

In Fig. 3.1 and 3.2 is possible to see both architectures from a high-level point of view. On the left we have the real-world scenario, in which OpenCar would receive data from the LIN, CAN and MOST buses of a real road-going vehicle. For the reasons stated previously on this document, such thing will not be possible just yet.

On the right, there is a similar architecture, but now the data comes from a different source. After reading documentation about *Assetto Corsa*, we found out that it was possible to extract a lot of information about the on-going simulation, in real-time. We took some investigating, but in the end we were able to extract that same information via an User Datagram Protocol (UDP) socket.

In order to do so, we had to implement a piece of software that *Assetto* could read and execute while in-game. With that piece of software up and running, all that was needed was a UDP reader to be implemented responsible to listen to that same socket to get the data and feed it to OpenCar's *contoller.js*, taking advantage of its HTTP Layer by using websockets to minimise any probable delay. The code developed in earlier testing proved very useful to accomplish this feat.

However, in earlier tests the MVC design pattern had to be somewhat "broken" in order to achieve the results showed. The logic had to be moved to the *view.js* and leave the *controller.js* untouched, which was not optimal. In this final version, we were able to separate most of the functionality from the the *view.js* and relocate it on the *controller.js*, as it should be. This way, we were able to "trick" the OpenCar simulator and inject data in real-time from a source other than its original websockets.

After the data successfully reaches OpenCar, both architectures work the same. There is a service running on a server that will receive data from the framework, treat it and then store it in a relational database. This same service will also serve the online platform made for the drivers to consult the information about their driving.

The architecture represented in Fig. 3.2 b) will be fully detailed further in this section, since it is the one that will be fully tested and completed during development.

3.3.2 System Context Diagram

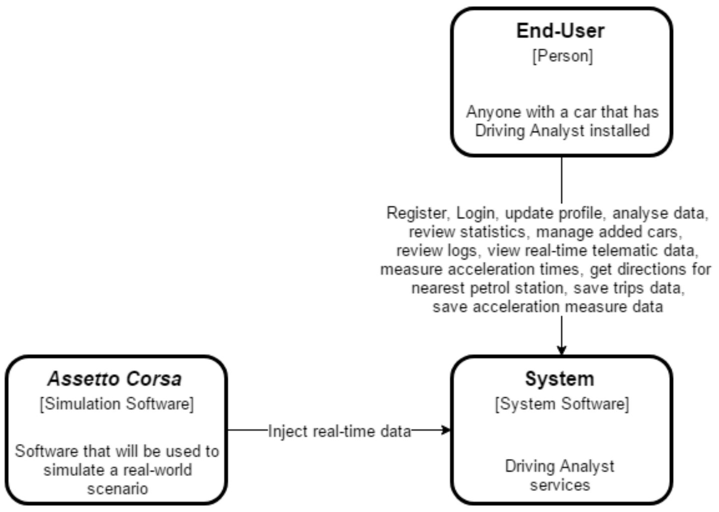

End-User
[Person]
Anyone with a car that has Driving Analyst installed
Register, Login, update profile, analyse data, review statistics, manage added cars, review logs, view real-time telematic data, measure acceleration times, get directions for nearest petrol station, save trips data, save acceleration measure data
Assetto Corsa
[Simulation Software]
Software that will be used to simulate a real-world scenario
Inject real-time data
System
[System Software]
Driving Analyst services

Figure 3.3: System Context diagram

Aside from the diagrams on Fig. 3.2 displayed earlier, this diagram is the top view level. The user has interaction, having the liberty to do a number of operations on a given *System* which receives data in real-time from a simulation software, inducing the user into a real-world scenario.

3.3.3 Container Diagram

The Container Diagram in Fig. 3.4 takes us a step deeper into the architecture. Starting from the top, we still have the same user and simulation software interacting with the system, however, the system is now divided in containers, each showing which technologies are being used to implement their functionality, with a brief description attached. Also shown in this diagram is the way each component communicates with each other.

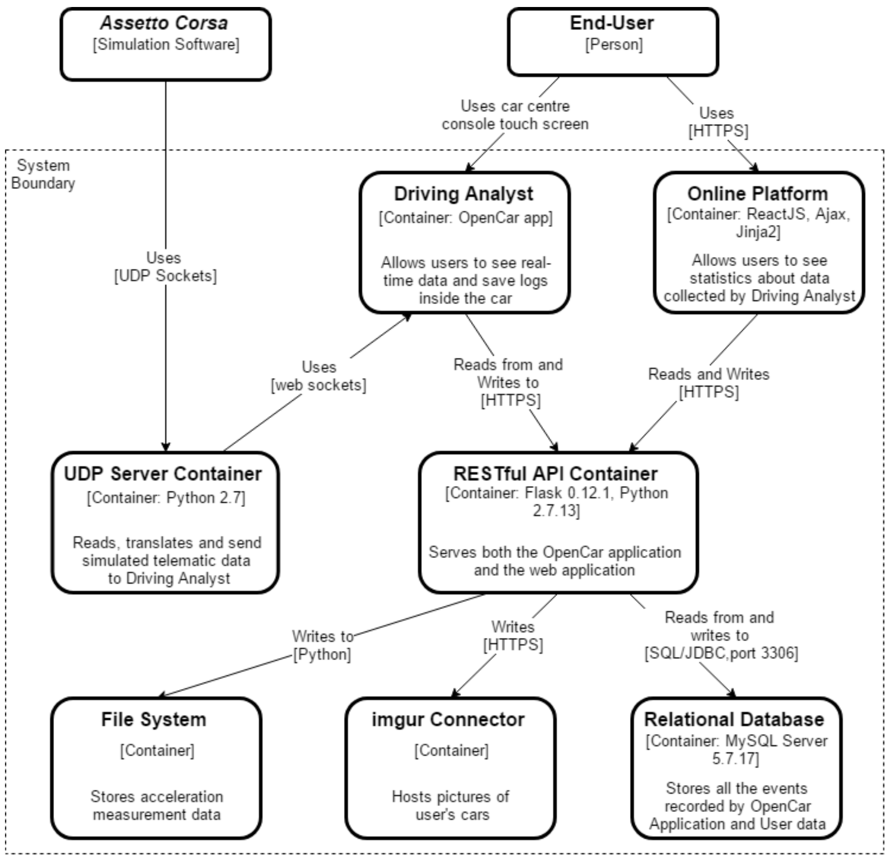

Assetto Corsa
[Simulation Software]
End-User
[Person]
Uses car centre console touch screen
Uses
[HTTPS]
System Boundary
Uses
[UDP Sockets]
Driving Analyst
[Container: OpenCar app]
Allows users to see real-time data and save logs inside the car
Online Platform
[Container: ReactJS, Ajax, Jinja2]
Allows users to see statistics about data collected by Driving Analyst
Uses
[web sockets]
Reads from and Writes to
[HTTPS]
Reads and Writes
[HTTPS]
UDP Server Container
[Container: Python 2.7]
Reads, translates and send simulated telematic data to Driving Analyst
RESTful API Container
[Container: Flask 0.12.1, Python 2.7.13]
Serves both the OpenCar application and the web application
Writes to
[Python]
Writes
[HTTPS]
Reads from and writes to
[SQL/JDBC,port 3306]
File System
[Container]
Stores acceleration measurement data
imgur Connector
[Container]
Hosts pictures of user's cars
Relational Database
[Container: MySQL Server 5.7.17]
Stores all the events recorded by OpenCar Application and User data

Figure 3.4: Container diagram

3.3.4 Components Diagrams

The diagrams in this section are related to the one on the previous section, simplifying some objects and detailing others.

A Components diagram takes us even deeper into the architecture, now showcasing the specific components of each part of the software. The one in Fig. 3.5 takes the diagram shown before in Fig. 3.4 and breaks down what was once before *UDP Server Container* and the *Driving Analyst* into smaller components components, so it is possible to see the information flow inside the containers.

Similarly to Fig. 3.5, Fig. 3.6 breaks down the *RESTful API Container* shown in Fig. 3.4 into several components, and as we can see, the system currently uses many components to achieve its goals. The fact that they are so divided was a decision made to make the code easier to read, debug and maintain.

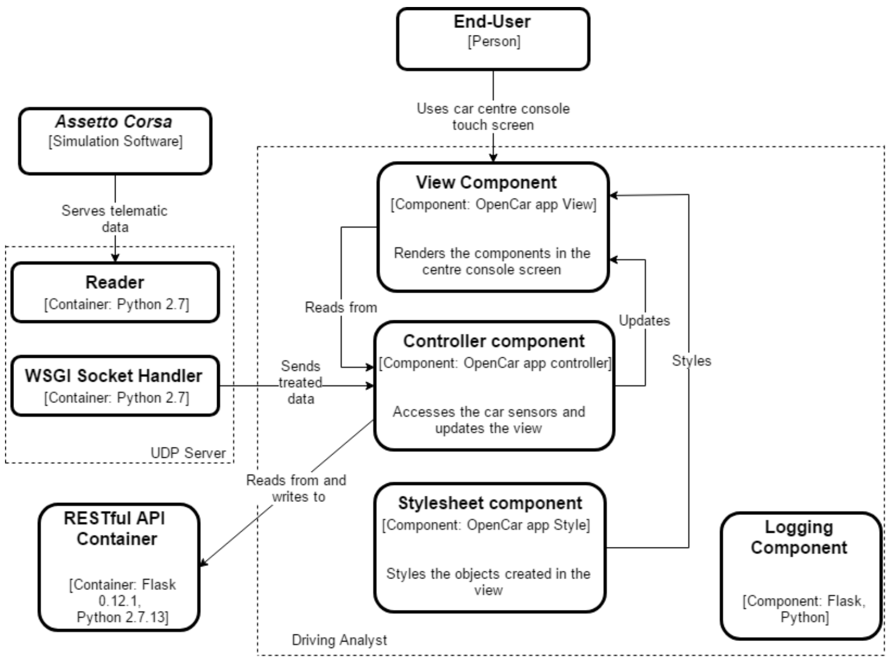

End-User
[Person]
Uses car centre console touch screen
Assetto Corsa
[Simulation Software]
Serves telematic data
Reader
[Container: Python 2.7]
WSGI Socket Handler
[Container: Python 2.7]
UDP Server
Sends treated data
View Component
[Component: OpenCar app View]
Renders the components in the centre console screen
Reads from
Updates
Styles
Controller component
[Component: OpenCar app controller]
Accesses the car sensors and updates the view
Reads from and writes to
RESTful API Container
[Container: Flask 0.12.1, Python 2.7.13]
Stylesheet component
[Component: OpenCar app Style]
Styles the objects created in the view
Logging Component
[Component: Flask, Python]
Driving Analyst

Figure 3.5: Component diagram of Opencar Application

3.4 Testing/Validation

As was mentioned several times in this document before, the technology to be used in this project is rather new. All the investigation done for this project was made purely out of documentation and information available in the respective company's websites.

This said, there was currently no way to test the final product in a real-world scenario, in a real car. An exchange of emails with INRIX's developer support assured that it would not be possible. With this in mind, the idea of using driving simulators came through, something that was already being done by AGL itself which meant that the idea was validated.

3.4.1 The Simulated Environment

The choice of using *Assetto Corsa* by *Kunos Simulazioni* [24] was made not only considering how accurate and precise the physics are when compared to other similar software but also because it allows the extraction of data from the vehicle that is being simulated, in real-time. There is documentation about how to extract such information and, on the opposite of OpenCar, an already well-established community.

As we can see in Fig. 3.7, an early demo developed with the sole purpose of proof of concept, we were now able to inject telematic data coming from Assetto Corsa into OpenCar simulator, in real-time. This will give a much

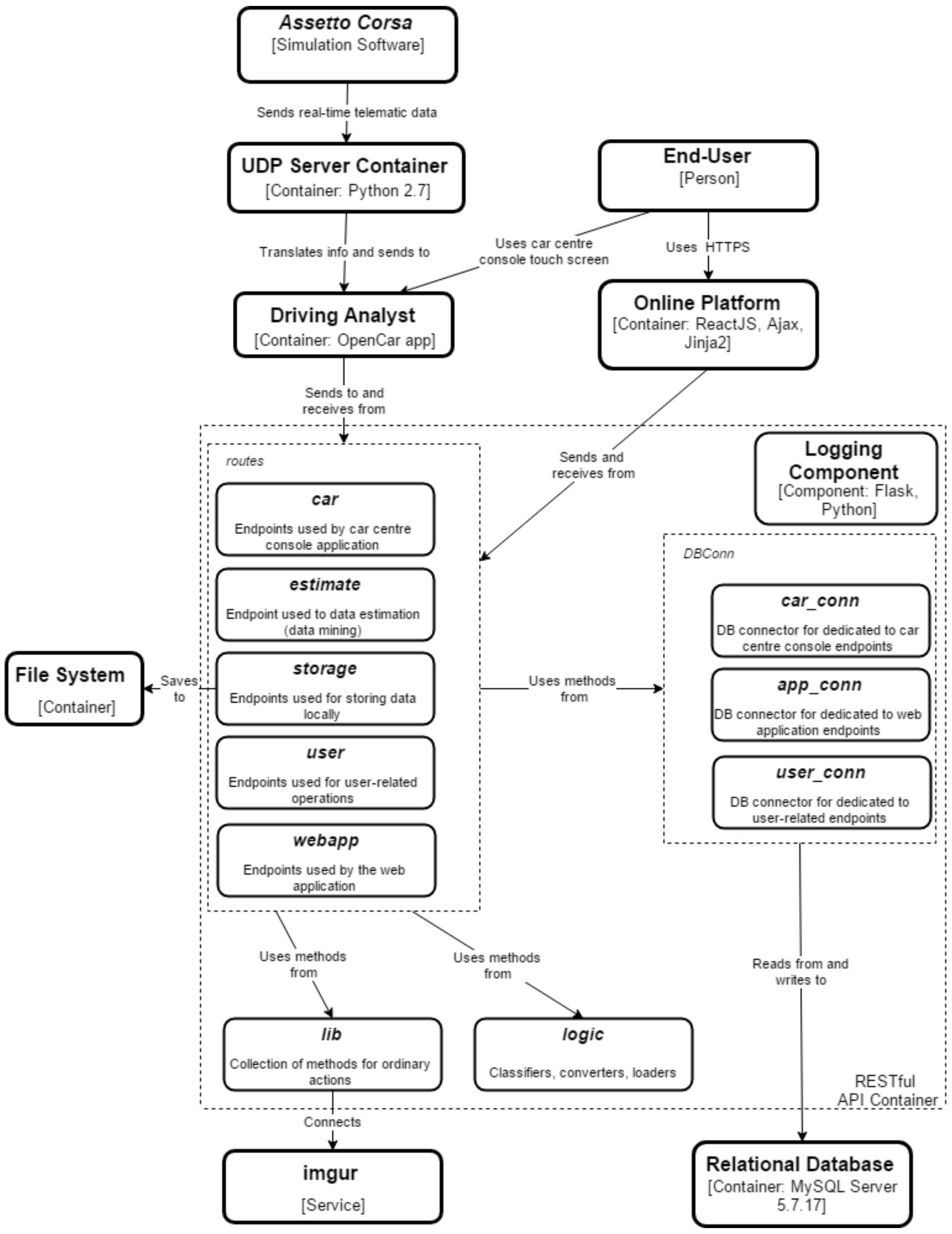
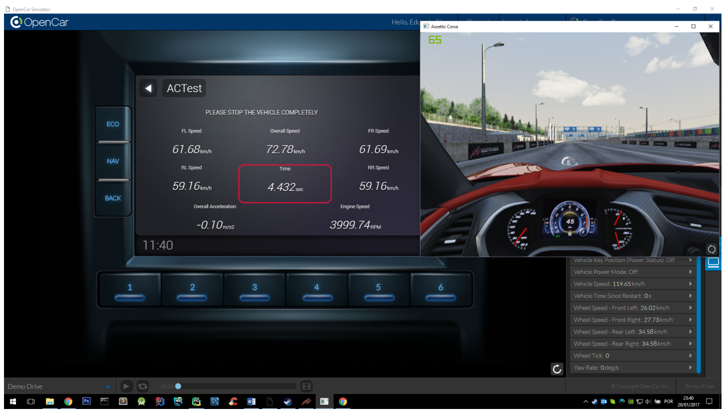

Assetto Corsa
[Simulation Software]
Sends real-time telematic data
UDP Server Container
[Container: Python 2.7]
End-User
[Person]
Translates info and sends to
Uses car centre console touch screen
Uses HTTPS
Driving Analyst
[Container: OpenCar app]
Online Platform
[Container: ReactJS, Ajax, Jinja2]
Sends to and receives from
Sends and receives from
routes
car
Endpoints used by car centre console application
estimate
Endpoint used to data estimation (data mining)
storage
Endpoints used for storing data locally
user
Endpoints used for user-related operations
webapp
Endpoints used by the web application
Logging Component
[Component: Flask, Python]
DBConn
car_conn
DB connector for dedicated to car centre console endpoints
app_conn
DB connector for dedicated to web application endpoints
user_conn
DB connector for dedicated to user-related endpoints
File System
[Container]
Saves to
Uses methods from
Uses methods from
Uses methods from
Reads from and writes to
lib
Collection of methods for ordinary actions
logic
Classifiers, converters, loaders
RESTful API Container
Connects
imgur
[Service]
Relational Database
[Container: MySQL Server 5.7.17]

Figure 3.6: Component diagram of the system API



PLEASE STOP THE VEHICLE COMPLETELY

FL Speed 61.68km/h
Overall Speed 72.78km/h
FR Speed 61.69km/h
RL Speed 59.16km/h
Time 4.432sec
RR Speed 59.16km/h
Overall Acceleration -0.10m/s2
Engine Speed 3999.74RPM

Vehicle Key Position (Power Status): Off
Vehicle Power Mode: Off
Vehicle Speed: 119.65km/h
Vehicle Time Since Restart: 0s
Wheel Speed - Front Left: 26.02km/h
Wheel Speed - Front Right: 27.73km/h
Wheel Speed - Rear Left: 34.58km/h
Wheel Speed - Rear Right: 34.58km/h
Wheel Tick: 0
Yaw Rate: 0deg/s

Figure 3.7: Assetto Corsa injecting data into OpenCar simulator

more realistic environment for the final product to be validated in a more realistic way than the originally thought of with OpenCar sliders.

After crossing the telematic data provided by *Assetto Corsa* with all the data provided by OpenCar, the result was Table D.1.

3.4.2 Tests conducted

The simulation

To test this architecture we needed both the functional architecture, even if partially, and test drivers. That was provided by *The Driving Club - Coimbra* [9], a simulation centre close to the university. This centre currently offers a simulation experience using the very same simulation software that was intended for this project, what created a perfect opportunity to test part of the architecture in terms of reliability.

The tests consisted simply in installing a sample of the software developed to be used on this project on their simulators and let the drivers do the rest, unaware of the software. For hours straight the software was able to retrieve all the information that was theoretically possible to extract, at a surprisingly fast pace, with no crashes. At this point, the extraction of telematic data in real-time and the injection in the OpenCar simulator was tested and approved.

The interfaces

Apart from these tests, usability tests were also conducted on the online platform and the OpenCar application.

As was expected, the OpenCar application left the testers divided. They were able to understand most of what was being shown except for more specific matters like G-force measurement or the *Acceleration Measurement* screen. Of course, some enthusiasts recognised them right away with no difficulties, finding them pretty interesting.

As for the online platform the results could be better at first. Common data like fuel consumption and distance travelled were understood very quickly, however terms like *wheel-spin* and *wheel-lock* were somewhat unknown. To minimise this issue, several descriptions were added to the interface, as well some interpretations and what to do with those same values. However, the *Acceleration* tab was once again overlooked by the *common driver* and much more appreciated by the the enthusiast crowd.

Other tests

Aside from the reliability and usability tests described above, there were conducted some more tests on what kind of values the simulation could return in certain situations, which proved to be very stable. However, since this project will not be reaching the market anytime soon due to questions of, for example data ownership, and since it is not considered a *critical system*, unity testing and and white-box testing were not performed.

However, the usability tests using the complete system were long and exhaustive with little to no errors or crashes, which is a very good sign.

Concluding, since one of the main goals of this project was to give something useful for an everyday life while taking attention to the enthusiast share of the market, the results were overall satisfactory.

3.5 Technologies

In addition to OpenCar, this project will require a few more tools to achieve its goals. This section will explain the technologies being used and why.

The choice for using Python as the main programming language, more specifically version 2.7.12, was made mostly based on the simplicity of the language and to speed up the development process, given how simple it is and the previous experience we have programming in Python. In spite of this Python has already been used in early tests to extract data from *Assetto Corsa* and the driving simulator itself also gives the chance to program new features in Python if desired, so it is only logical to keep the language through the whole system.

For developing the API, Flask seemed to be the way to go. Flask is a micro-framework for Python based on Werkzeug and Jinja2 [13]. Since the choice for using Python was already made, the choice for the framework to use to develop the service that will connect the whole system had to be designed for Python. Flask allows for really fast prototyping, being a "just-enough" for the purposes intended here. With the mindset of boosting the development this micro-framework was chosen.

In terms of storage, there were at least three options at on the table: SQLite, MySQL and PostgreSQL. SQLite was out of question since it does not handle well multi-user applications. Between the popular MySQL and the open-source PostgreSQL, the decision was ultimately to use MySQL since it is more appropriated for web-sites and web-applications, which is the final goal of this project, not to mention the low level of difficulty to work with.

Finally, in order to give the online platform a fresh and fast look, the ReactJS was chosen as the framework to implement the User Interface (UI) of the web application. This choice was made purely out of the limited experience we have in designing UI's for the web, therefore we will be using one that we can do something minimally attractive.

Chapter 4

The Project

This section will describe in detail all the software that was developed during this project, as well as show its final aspect. It will cover many aspects from the design to the file structure.

4.1 Assetto Corsa developed add-on

As was said before, the simulation software we are using is capable of reading multiple parameters regarding the on-going simulation, in real-time. With this in mind, all that was needed was a way to make this data available outside the simulation.

After some investigation about the inner functioning of the game, we found out that it has a UDP socket that already puts out a nice set of parameters. However, there is more information to extract by other means, and that is the reason for the existence of this so called *add-on*.

Assetto Corsa allows developers to make Python apps that can run inside the game since it has a version of a Python interpreter embedded. However, it is not as simple as writing any Python script. It had specific modules to load, as well as predefined methods, each with their own function.

```python
import sys, ac, acsys, time, os, sys, platform, json

if platform.architecture()[0] == "64bit":
    sysdir='apps/python/SharedMemory/DLLs/stdlib64'
else:
    sysdir='apps/python/SharedMemory/DLLs/stdlib'
sys.path.insert(0, sysdir)
os.environ['PATH'] = os.environ['PATH'] + ";."
try:
    import socket
except ImportError as e:
        import traceback
        ac.console(str(e))
    (...)
    from sim_info import info
    (...)
    def acMain(ac_version):
        global l_fl, l_fr, l_rl, l_rr, fl, fr, rl, rr
        appWindow = ac.newApp("SharedMemory")
        (...)
        return "SharedMemory"

    def acUpdate(deltaT):
        global l_fl, l_fr, l_rl, l_rr, fl, fr, rl, rr
        (...)
```

Above is a portion of the add-on running inside the game, with only the important parts visible. As we can see, some libraries are well-known to the common Python programmer. However, libraries of the likes of *ac* and *acsys* are not since they are specific to the game. These are the libraries that allow the reading of information in real-time.

When defining an app for this particular simulation software, there are two methods that must be always implemented, *acMain(ac_version)* and *acUpdate(deltaT)*. The first one is responsible for initialising all the necessary variables and any GUI that it might contain and return the application name, so it appears in-game. The second one is executed *deltaT* times per second, so we want to read our information there so we can get it in real-time.

Apart from *game-specific* methods, the rest of the code is pretty straightforward. On the top we must tell the app if we are running the game on a 32-bit or 64-bit machine so we can import the right libraries. After, try to import the *socket* module so we can open the previously mentioned UDP socket. Finally, on the end of the *acUpdate* method we simply use the *json* module to wrap all the info collected, open the socket, send the package and close the socket. This process happens as many times as the *acUpdate()* method executes, which means the socket dictates how fast our simulation will be in terms of real-time data feeding.

Above on the left we can see the file structure of the software in question. As you may be wondering, those really are *.pyd* Python files. The *_ctypes.pyd* is used to read some of the classes that contain information, the *_socket.pyd* one to open the socket and the *unicodedata.pyd* to convert the whole package into a datagram so it can be sent via UDP . These files are needed since the game has a very specific Python interpreter and there is not much liberty when importing modules. As for the rest, *SharedMemory.py* contains the

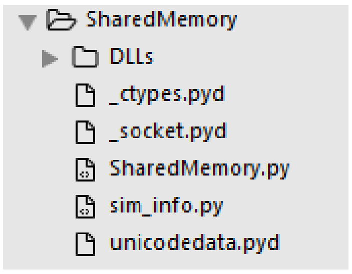


Figure 4.1: Assetto Corsa Add-On file structure

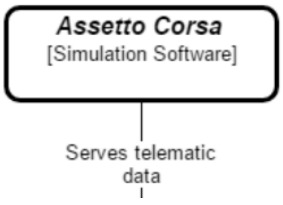


Figure 4.2: Component representation

code that was just described in this section and the *sim_info.py* contains auxiliary Python classes, used to read the data more easily. This file is made according to the documentation made available by *Kunos Simulazioni*. On the right we can see where it fits in figures 3.5 and 3.4.

4.2 UDP Reader

Now that we have the information being read and sent, we need to "catch" it. To do so, the UDP Reader was created. This piece of software will simply listen to both the socket of the game and the socket created in the add-on that is described in the previous section.

This software is made of two fairly simple Python scripts, a *reader.py* that keeps the server up and running with a websocket and the *core.py* where the information is received and translated to be sent.

```
@sockets.route('/get')
def echo_socket(ws):
    while not ws.closed:
        message = ws.receive()
        ws.send(dumps(sendToOpenCar()))

if __name__ == "__main__":
    from gevent import pywsgi
    from geventwebsocket.handler import WebSocketHandler
    server = pywsgi.WSGIServer(('', 5000),
                               app,
                               handler_class=WebSocketHandler)
    server.serve_forever()
```

Apart from the *imports*, this is all the code required to keep the websocket server running. Whenever the application running in OpenCar sends a message, this server will reply with the contents read in the *sendToOpenCar()* method. This operation may run forever until the connection is broken (either this websocket server shuts down or the OpenCar simulator shuts down) and happens pretty much in real-time, with almost no delay from the game.

```
# send Handshake
sock.sendto(Handshaker(1, 1, HANDSHAKE), (host, port))
data, addr = sock.recvfrom(ctypes.sizeof(HandshakerResponse))
resp = HandshakerResponse()
ctypes.memmove(ctypes.addressof(resp), data,
               ctypes.sizeof(HandshakerResponse))
# Get info from car
sock.sendto(Handshaker(1, 1, SUBSCRIBE_UPDATE), (host, port))
data, addr = sock.recvfrom(ctypes.sizeof(RTCarInfo))
update = RTCarInfo()
ctypes.memmove(ctypes.addressof(update), data,
               ctypes.sizeof(RTCarInfo))
# Collect SharedMemory data
sm = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sm.bind(('127.0.0.1', 5001))
data = sm.recv(1024)
sm_data = json.loads(data.decode('utf-8'))
# SEND MESSAGE TO SOCKET
payload = {(...)}
return payload
```

This code sample is taken from the previously mentioned *sendToOpenCar()* method. As was mentioned before, thanks to the add-on developed we now have two UDP sources. However, the one embedded in the game took a little more work since we had to follow a very strict methodology defined by the game developers. It is noticeable mainly by the need to perform a handshake first, and handle memory as if we were programming in *C* language.

This method reads from both sockets, and returns a single dictionary that is then sent to OpenCar.

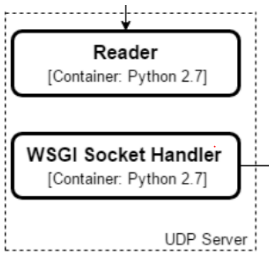


Figure 4.3: Component representation

On the architecture, this software is represented as *UDP Server Container* on Fig. 3.5, where the method explained above fits in the *Reader* container.

4.3 Console Application

Figure 4.4: Driving Analyst logo

In this section will be explained in detail the various features and decisions made in the development process of the OpenCar application, baptised as ”Driving Analyst”.

4.3.1 Controller structure

As was previously explained on chapter 2, the OpenCar framework advises the use of a MVC design pattern. This subsection will explain how the **C**ontroller was made.

As advised, in the controller should be all the business logic needed for the application, and that includes subscribing sensors across the car.

```
  const telematics = new TelematicsAPI();

// Engine Speed
telematics.subscribe(TelematicsAPI.Event.ENGINE_SPEED, rpm => {
    this.getView().updateEngineSpeed(rpm);
});
```

This little code sample here would allow us to subscribe the event of the car’s engine speed changing. Every time it changed it would then call the *view* and execute one of its functions, that would most likely change something on the interface. However, in the meantime there was no way we could test this on a real car and as such, we would be limited to the sliders on the OpenCar simulator (Fig. 4.5).

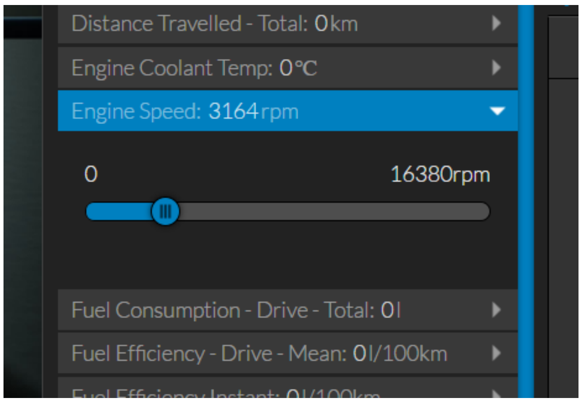

Distance Travelled - Total: 0km
Engine Coolant Temp: 0°C
Engine Speed: 3164rpm
0
16380rpm
Fuel Consumption - Drive - Total: 0l
Fuel Efficiency - Drive - Mean: 0l/100km

Figure 4.5: OpenCar Simulator slider

This would be enough if we were monitoring only one sensor, but it is not the case since we are monitoring a large number of sensors and values, therefore the need to introduce a driving simulator on the project, a method that has already been used by AGL themselves.

On the subsection before we covered how the information was being release via the UDP Reader, now we only needed to connect it to the OpenCar simulator. In order to do that all the code regarding the *TelematicsAPI* has been deleted and this has taken its place:

```
var view = this.getView();
// Open a web socket
var ws = new WebSocket("ws://localhost:5000/get");
ws.onopen = function () {
    // Web Socket is connected, send data using send()
    ws.send("Message to send");
};
ws.onclose = function () {
    // websocket is closed.
    Log.log("Connection is closed...");
};
ws.onmessage = function (evt) {
    /* Update values */
    var info = JSON.parse(evt.data);
    var engineRPM = parseFloat(info.engineRPM).toFixed(0);
    view.updateEngineSpeed(engineRPM);
    (...)
}
```

This sample of code does exactly the same as the one before, but now ignores the OpenCar's slider and, assuming the UDP Reader and the Add-On are running, will read data from the simulation software instead, in real-time.

The main differences are, of course, the use of a websocket instead of the original OpenCar's *TelematicsAPI* to update the values and the need to store the *getView()* value outside the *onMessage* method due to being in different scopes. In order to calculate and detect events in real-time, all the logic had to be moved inside the *onMessage()* method as well. Curiously enough, this is not a complete "cheating on the system" act since the OpenCar's simulator uses websockets all the time internally.

Aside from the websocket implementation, there are also several methods declared that are responsible for communicating with the API using OpenCar's *InternetDataAccessAPI*. These can be called either from the controller or the view.

4.3.2 View structure

To make the development more organised and readable, the methodology adopted was to treat every single component on the screen as an object. By doing this, every dashboard is, in a low level view, a container.

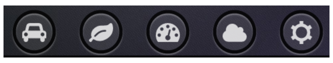


Figure 4.6: Action bar in Driving Analyst app

As we can see in figure 4.6, there are a total of five different options. This means that there are five containers of objects. The OpenCar framework allows the creation of objects directly using HTML. Although this seems temptingly easier, the container allows us to simply show or hide a whole container, instead of hiding or showing one by one.

```
const rpm = new TextView({
    id: 'rpm_style',
    text: '0000 RPM'
});
```

Here we can see how the object is declared. This particular one shows a text box with the engine's current speed.

```
#rpm_style {
    position: absolute;
    left: 65px;
    font-size: 26px;
}
```

When declaring objects it is possible to give them an ID that can be used by the CSS *stylesheet*, just like an HTML tag.

The *text* attribute is given simply to initialise the text box, so it does not start empty. As was said in the previous subsection the controller will call for functions declared on the view, giving them new parameters. Following the very same example as before, here is an implementation of the *updateEngineSpeed()* function:

```
updateEngineSpeed(value) {
    rpm.setText(value + " RPM");
}
```

This way, every time the controller receives a new value, will call this method and the changes will be visible by the user on his centre console.

Finally, in order to add this text box to a container, called *Pane* in the framework, is as simple as this:

```
/* DRIVE INFO VIEW Panel construction */
const drivePane = new Pane();
(...)
drivePane.addChild(rpm);
```

Now the *drivePane* contains the object, and can be hidden from the console hiding all its contents. This method is then repeated to all the components that are visible in the centre console, throughout the five different *Panes*.

By doing this, every time the user chooses a different option on the action bar, all the software needs to do is to show the one selected, and hide the other four. This made the code a lot more organised, and easy enough to find and fix possible problems or even add more and more objects to the application, which is one of its non-functional requirements.

4.3.3 Final Aspect & Functionality

In this subsection will be displayed and explained the full extent of the centre console's application functionality and design.

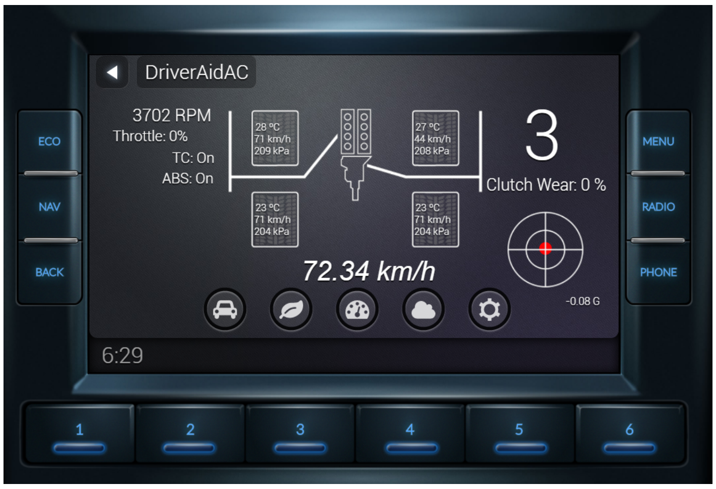


Figure 4.7: Centre's console application main dashboard

When opening the application the first screen the user will see is the one pictured in Fig. 4.7. On the left we can see some general information regarding the engine, such as its speed in RPM and the throttle. Is also shown the current status of the Anti-lock Braking System (ABS) and Traction Control System (TCS). Whenever one these technologies is activated and engages a little icon will blink as well.

On the right is displayed the current gear the car has engaged and the current condition of the clutch. Right under this text there is what we can call a "G-meter". That red circle will bounce upwards or sideways according to the car's generated g-forces, either from turning or accelerating/braking, giving the driver an idea of how much force his car is generating.

Right in the middle of the screen we can see the four tyres, all of them with temperature, pressure and linear speed being monitored in real-time. Whenever the car experiences wheel-spin, uneven pressure values on a given axle or the tyres too hot, the graph will display a warning.

Lastly, we have the current car's speed being shown just below the graphic.

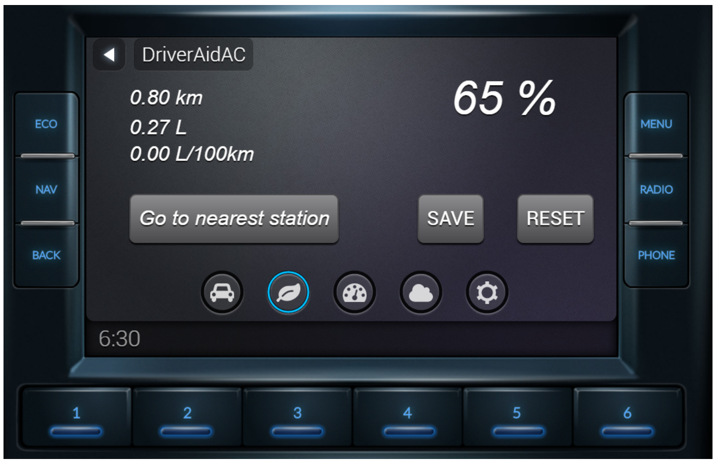


Figure 4.8: Centre's console application fuel dashboard

By pressing the leaf logo, the user is shown the dashboard pictured in Fig. 4.8. On the right is possible to see how much fuel he currently has in his car in percentage, while on the left we can see how the current distance travelled in kilometres, how much fuel has been spent during that distance in litres and how much the car is spending in a L per 100km relation.

The instant L/100km calculation might not be as accurate as a real car since the simulation software does not give any information regarding specific parameters such as engine air-flow, injector's size, among other mechanical specific measures. Instead, is calculated using RPM, throttle input, and the car's speed.

First, we must calculate how many litres per second the car is burning:

$$L/s = \frac{Throttle * RPM * 0.0030}{1000}$$

Now, using the L/s (Liters per second) value

$$L/100km = \frac{360000 * L/s}{speed}$$

It is not ideal, but gives a close approximation and since this is only for simulation purposes is a pretty good enough method.

Aside from text, we can see three buttons. Starting from the right, the button *RESET* resets the values that are being shown on the upper left corner of the dashboard to zero. This is particularly useful considering that the *SAVE* button will save on the database the values that are currently being shown.

Lastly the button *Go to nearest station* is pretty self-explanatory. Once pressed it will show the user what we can see in figure 4.9.

In here the application takes full advantage of the OpenCar's *MapDialogConfig* API to show the driver directions to the nearest fuel station in a

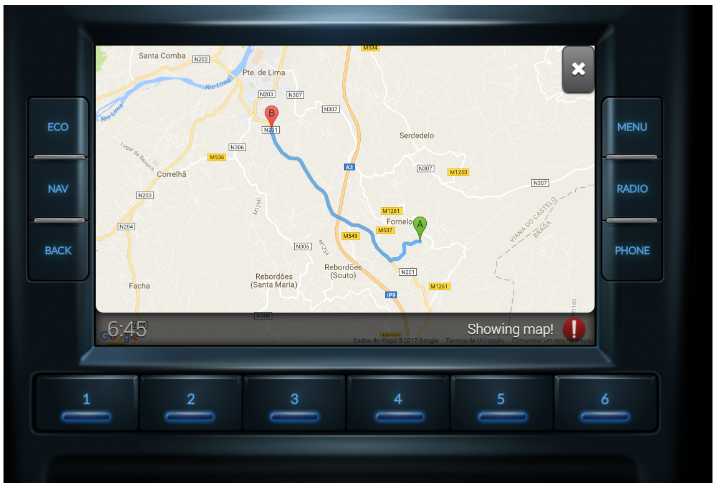


Figure 4.9: Fuel dashboard - Nearest Station

radius of 20km thanks to Google's *PlacesAPI*. However, since the simulation software uses fictional places, this feature cannot be fully tested.

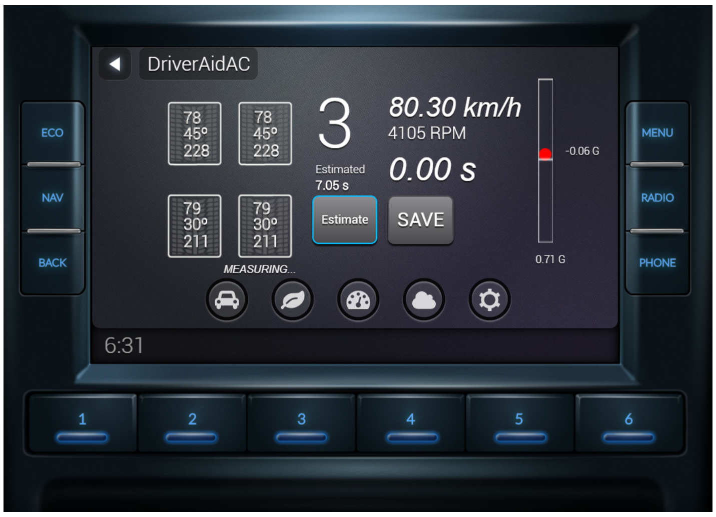


Figure 4.10: Centre's console application acceleration dashboard

In Fig. 4.10 is pictured undoubtedly the most complex and interesting feature of the whole application, and would surely be adopted by many, many driving enthusiasts: measuring and comparing 0-100km/h acceleration times.

Similarly to the main dashboard mentioned earlier, we can see the current conditions of the tyres (top-down: tire linear speed, temperature, pressure), now in a bigger size for easier inspection.

Taking inspiration on some high-performance vehicles the gear is now shown in the middle of the screen, with the speed and RPM right next to it giving a more sporty feel and just below the RPM value is displayed the last successful 0-100 acceleration. In addition to that, the G-meter is still there, but now it only measures G forces on the x-axis (accelerating/braking) and keeps record of the maximum force generated. Lastly, right above the action bar we can see instructions given by the application for the user to know exactly how he should proceed to successfully record his times.

The buttons *SAVE* and *Estimate* play a huge role in this application. Every time the user registers a valid acceleration time, he can press the button *SAVE* and it will send all the data recorded during the acceleration directly to the API, where it is stored for later study.

During an acceleration measurement, the application is capable of recording the following information:

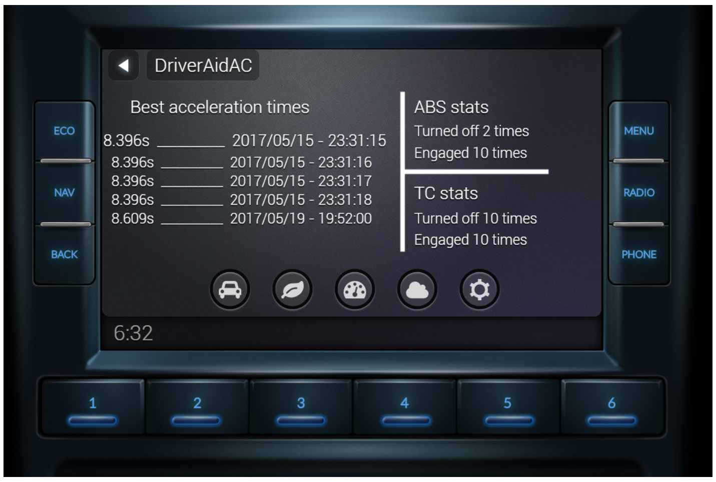
- The car's linear speed
- Each wheel speed during the experiment
- Each wheel initial temperature
- Each wheel initial pressure
- Throttle pedal position during the experiment
- Engine speed during the experiment
- G-Force values generated during the experiment
- RPM values in which gears were engaged during the experiment
- Instants in which gears were engaged during the experiment
- Overall time of the experiment
- TCS status

All this information is then put together and sent to the API.

Last but not least, there is the *Estimate* button. As the name suggests, it can estimate how much time the driver will take to reach 100km/h from standing still based on the current conditions of the car and the driver's previous performances.

Once the driver presses the button, the application register the current tyre's complete status, RPM range, TCS status and throttle position and send it to a specific route on the API. It will then take all the recorded acceleration times and feed them to a multiple linear regression algorithm that will then compare what it "learned" with the current conditions that were just sent, and return an estimation of what the driver could perform on his next trial. The only drawback of this feature is that it relies on the driver to record as many acceleration times as possible in order to improve the algorithm's accuracy.

Not only that, but by estimating the possible outcome of the acceleration test, the application will also calculate the probably best RPM range the gears should be engaged based on his best performances and assist the driver by switching colours on the gear display. The colours go from red, yellow and green (worst to best).



Best acceleration times

8.396s ________ 2017/05/15 - 23:31:15
8.396s ________ 2017/05/15 - 23:31:16
8.396s ________ 2017/05/15 - 23:31:17
8.396s ________ 2017/05/15 - 23:31:18
8.609s ________ 2017/05/19 - 19:52:00

ABS stats
Turned off 2 times
Engaged 10 times

TC stats
Turned off 10 times
Engaged 10 times

Figure 4.11: Centre's console application online stats

Like the main screen that was firstly described, the screen shown in Fig. 4.11 screen is purely informative and does not require any input from the user or other features.

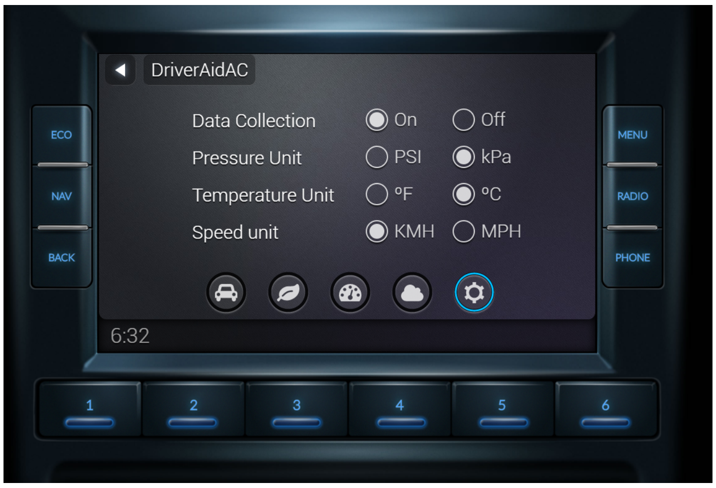
Here the driver can check his best five acceleration times ever recorded on the API as well as some minor statistics concerning both ABS and TCS technologies, showing the total amount of times that the car recorded any action on behalf of each one.

This information is available in a lot more detail on the online platform which will be explained later on this document.

Considering that people from different nations have different tastes in unit measurement, the screen shown in Fig. 4.12 is definitely a must. The user is able to change the unit of temperatures, pressures and speed.

However, this is merely visual. The user might be seeing the values he chose to but the calculations and all the logic being applied to the application is made using International System (IS) units and any value is also stored using these same units. This way, there are no discrepancies on the calculations and the user gets to keep the units he likes the most to see on his centre console.

Another important aspect is the *Data Collection* status. Currently the application is able to record a series of events experienced by the car that



Data Collection [✓ On] Off
Pressure Unit PSI [✓ kPa]
Temperature Unit ºF [✓ ºC]
Speed unit [✓ KMH] MPH

Figure 4.12: Centre's console application settings

the user might or might not notice on its own. The events being recorded are:

- ABS turned On/Off
- ABS engaged
- TCS turned On/Off
- TCS engaged
- Wheel-spin
- Wheel-lock
- Major G-forces on the suspension
- Uneven tyre pressure on a given axle of the car
- Data Collection tuned On/Off

These events are recorded alongside the time-stamp of when it happened, and are silently sent to the API for further study on the online platform.

However, if the data collection is turned off the application simply records when it was turned off and will not record anything until it is back on again. This is made to give the users a choice to preserve their privacy at the wheel.

4.4 Platform

In this section will be explained both the structure and functionality of the online platform included in this project.

4.4.1 Final Aspect & Functionality

This section will show and explain the web application's final aspect and functionality of every screen available.

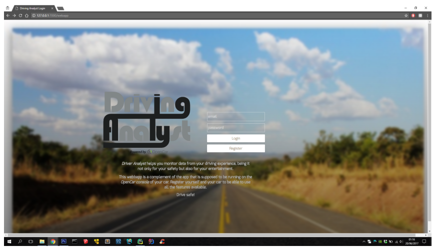


Figure 4.13: Login page

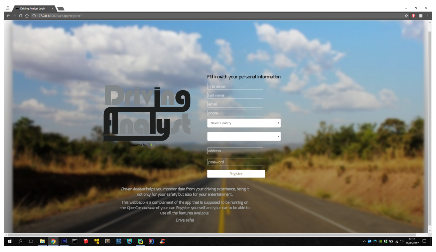
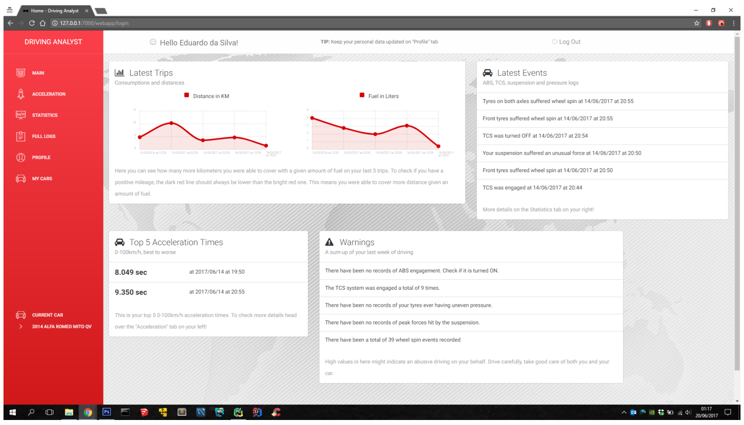


Figure 4.14: Register page

Figure 4.15: Login & Register pages

In figure 4.15 are shown the first pages the end-user will ever see when starting to use *Driving Analyst*. The login page is pretty straight-forward: the user simply inserts his credentials and clicks "Login" to enter the application. If something goes wrong while login a message will appear on top of the fields stating a possible reason for it to go wrong.

The register page, obviously, has a few more fields. The user will have to fill them in with his personal information and click "Register", and then wait for an email on his inbox containing an authentication link. Only when navigating to that link the account will be activated.



Hello Eduardo da Silva!

TIP: Keep your personal data updated on "Profile" tab

Latest Trips
Consumptions and distances

Here you can see how many more kilometers you were able to cover with a given amount of fuel on your last 5 trips. To check if you have a positive mileage, the dark red line should always be lower than the bright red one. This means you were able to cover more distance given an amount of fuel.

Latest Events
ABS, TCS, suspension and pressure logs

Tyres on both axles suffered wheel spin at 14/06/2017 at 20:55
Front tyres suffered wheel spin at 14/06/2017 at 20:55
TCS was turned OFF at 14/06/2017 at 20:54
Your suspension suffered an unusual force at 14/06/2017 at 20:50
Front tyres suffered wheel spin at 14/06/2017 at 20:50
TCS was engaged at 14/06/2017 at 20:44

More details on the Statistics tab on your right!

Top 5 Acceleration Times
0-100km/h, best to worse

8.049 sec	at 2017/06/14 at 19:50
9.350 sec	at 2017/06/14 at 20:55

This is your top 5 0-100km/h acceleration times. To check more details head over the "Acceleration" tab on your left!

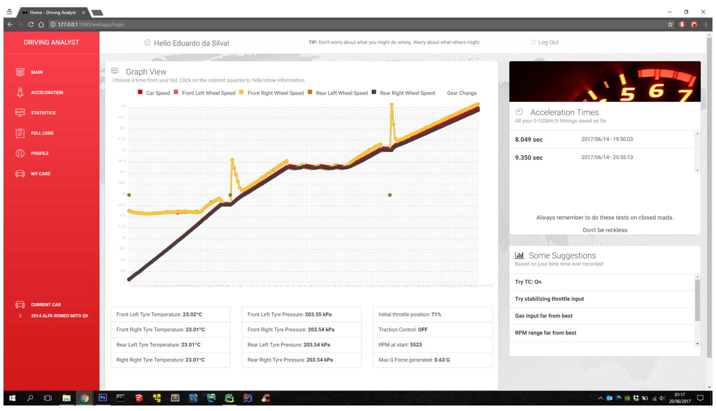
Warnings
A sum-up of your last week of driving

There have been no records of ABS engagement. Check if it is turned ON.
The TCS system was engaged a total of 9 times.
There have been no records of your tyres ever having uneven pressure.
There have been no records of peak forces hit by the suspension.
There have been a total of 39 wheel spin events recorded

High values in here might indicate an abusive driving on your behalf. Drive carefully, take good care of both you and your car.

Figure 4.16: Web application "Main" dashboard

In figure 4.16 is shown the first tab available on the sidebar. This dashboard shows information regarding the latest events recorded by the current car (indicated on the bottom of the sidebar), being these events automated (tyre monitoring, etc) or saved by the user (trips, acceleration times).



Hello Eduardo da Silva!

TIP: Don't worry about what you might do wrong. Worry about what others might.

Choose a time from your list. Click on the colored squares to hide/show information.

Front Left Tyre Temperature: **23.02°C**
Front Right Tyre Temperature: **23.01°C**
Rear Left Tyre Temperature: **23.01°C**
Right Right Tyre Temperature: **23.01°C**

Front Left Tyre Pressure: **203.55 kPa**
Front Right Tyre Pressure: **203.54 kPa**
Rear Left Tyre Pressure: **203.54 kPa**
Rear Right Tyre Pressure: **203.54 kPa**

Initial throttle position: **71%**
Traction Control: **OFF**
RPM at start: **5523**
Max G Force generated: **0.63 G**

All your 0-100km/h timings saved so far

8.049 sec	2017/06/14 - 19:50:03
9.350 sec	2017/06/14 - 20:55:13

Always remember to do these tests on closed roads.

Don't be reckless

Based on your best time ever recorded

Try TC: On
Try stabilizing throttle input
Gas input far from best
RPM range far from best

Figure 4.17: Web application "Acceleration" dashboard

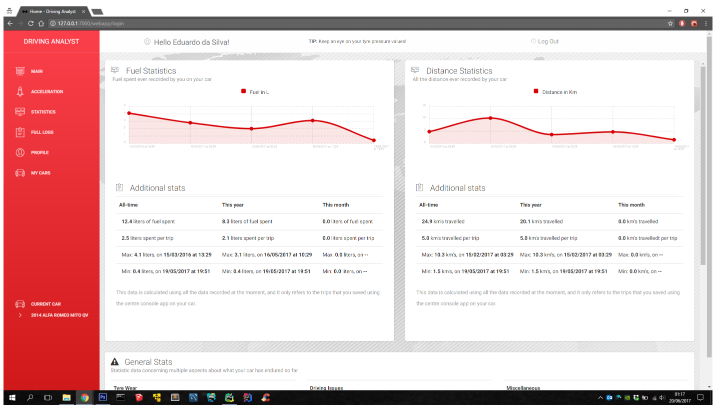
Next to the *Main dashboard* is the *Acceleration dashboard* shown in Fig. 4.17. This dashboard allows the users to fully analyse their saved acceleration times. On the right side of the screen the user will find a list of all the saved acceleration times he ever recorded with the current car. Simply by clicking one of them, the graph will re-draw itself, as well as the values below it.

The graph is represents six aspects of the being monitored during the time it took (X-Axis) to reach 100km/h (Y-Axis). These aspects are:

- **Car's speed** The vehicle's linear speed, as a whole (the same that would appear on the speedometer).
- **Rotating speed of each wheel** Since wheels speed unit is radians/sec, it cannot be directly compared to the the car's linear speed. In order to do that the angular speed was multiplied by the *wheel's radius * 3.6*. The result is an approximate speed value that the car should be going with this much angular speed, which provides an easy way to spot (and therefore detect) wheel spin events. For example, in figure 4.17 while the car's speed is at zero, the front wheels are already spinning at an angular speed of approximately 40km/h.
- **Gear Changes** The little green dots on the graph are representing the instant in which a new gear was engaged on the gearbox.

On top of the graphic, there is its legend stating which colours are representing what, and even allows for the user to simply uncheck a certain colour which will cause the graphic to re-draw again, now without showing the data-set that was just unchecked.

Lastly, just below the acceleration times list the user has a small list of suggestions. These suggestions are made by comparing his best acceleration measure of all time with the one in question and tell what went different in pretty much any aspect that is seen on the graph and the table below, trying to state why a measure was worse than is best.



Hello Eduardo da Silva!

TIP: Keep an eye on your tyre pressure values!

Fuel Statistics

Fuel spent ever recorded by you on your car

Additional stats

All-time	This year	This month
12.4 liters of fuel spent	**8.3** liters of fuel spent	**0.0** liters of fuel spent
2.5 liters spent per trip	**2.1** liters spent per trip	**0.0** liters spent per trip
Max: **4.1** liters, on **15/03/2016 at 13:29**	Max: **3.1** liters, on **16/05/2017 at 10:29**	Max: **0.0** liters, on **--**
Min: **0.4** liters, on **19/05/2017 at 19:51**	Min: **0.4** liters, on **19/05/2017 at 19:51**	Min: **0.0** liters, on **--**

This data is calculated using all the data recorded at the moment, and it only refers to the trips that you saved using the centre console app on your car.

Distance Statistics

All the distance ever recorded by your car

Additional stats

All-time	This year	This month
24.9 km's travelled	**20.1** km's travelled	**0.0** km's travelled
5.0 km's travelled per trip	**5.0** km's travelled per trip	**0.0** km's travelledt per trip
Max: **10.3** km's, on **15/02/2017 at 03:29**	Max: **10.3** km's, on **15/02/2017 at 03:29**	Max: **0.0** km's, on **--**
Min: **1.5** km's, on **19/05/2017 at 19:51**	Min: **1.5** km's, on **19/05/2017 at 19:51**	Min: **0.0** km's, on **--**

This data is calculated using all the data recorded at the moment, and it only refers to the trips that you saved using the centre console app on your car.

General Stats

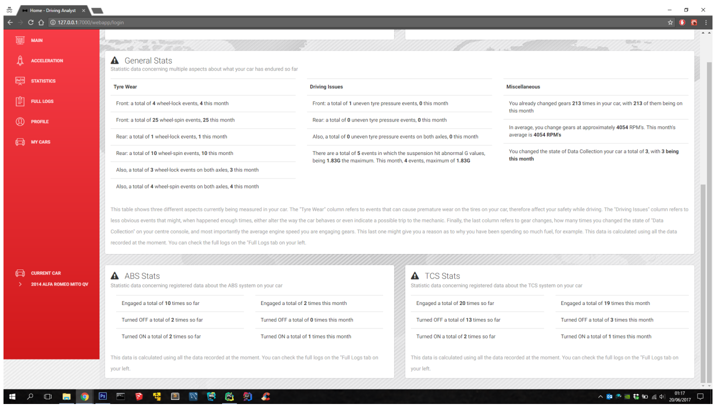
Statistic data concerning multiple aspects about what your car has endured so far

Tyre Wear | Driving Issues | Miscellaneous

Figure 4.18: Web application "Statistics" upper dashboard

The tab pictured in Fig. 4.18 does not have as much functionality as the last one, however, has a lot more information. By using all the data ever recorded by the car, the API calculates a set of interesting facts and statistics about the driving experience so far.

On the upper part of the dashboard the user is able to check statistics about the trips he chose to save on the centre console application. For both fuel spent and distance travelled, is possible to check all-time, yearly and monthly totals, as well as his largest and smallest values in fuel and distance.



General Stats

Statistic data concerning multiple aspects about what your car has endured so far

Tyre Wear	Driving Issues	Miscellaneous
Front: a total of **4** wheel-lock events, **4** this month	Front: a total of **1** uneven tyre pressure events, **0** this month	You already changed gears **213** times in your car, with **213** of them being on this month
Front: a total of **25** wheel-spin events, **25** this month	Rear: a total of **0** uneven tyre pressure events, **0** this month	In average, you change gears at approximately **4054** RPM's. This month's average is **4054 RPM's**
Rear: a total of **1** wheel-lock events, **1** this month	Also, a total of **0** uneven tyre pressure events on both axles, **0** this month	You changed the state of Data Collection your car a total of **3**, with **3 being this month**
Rear: a total of **10** wheel-spin events, **10** this month	There are a total of **5** events in which the suspension hit abnormal G values, being **1.83G** the maximum. This month, **4** events, maximum of **1.83G**	
Also, a total of **3** wheel-lock events on both axles, **3** this month		
Also, a total of **4** wheel-spin events on both axles, **4** this month		

This table shows three different aspects currently being measured in your car. The "Tyre Wear" column refers to events that can cause premature wear on the tires on your car, therefore affect your safety while driving. The "Driving Issues" column refers to less obvious events that might, when happened enough times, either alter the way the car behaves or even indicate a possible trip to the mechanic. Finally, the last column refers to gear changes, how many times you changed the state of "Data Collection" on your centre console, and most importantly the average engine speed you are engaging gears. This last one might give you a reason as to why you have been spending so much fuel, for example. This data is calculated using all the data recorded at the moment. You can check the full logs on the "Full Logs tab on your left.

ABS Stats

Statistic data concerning registered data about the ABS system on your car

Engaged a total of **10** times so far	Engaged a total of **2** times this month
Turned OFF a total of **2** times so far	Turned OFF a total of **0** times this month
Turned ON a total of **2** times so far	Turned ON a total of **1** times this month

This data is calculated using all the data recorded at the moment. You can check the full logs on the "Full Logs tab on your left.

TCS Stats

Statistic data concerning registered data about the TCS system on your car

Engaged a total of **20** times so far	Engaged a total of **19** times this month
Turned OFF a total of **13** times so far	Turned OFF a total of **3** times this month
Turned ON a total of **2** times so far	Turned ON a total of **1** times this month

This data is calculated using all the data recorded at the moment. You can check the full logs on the "Full Logs tab on your left.

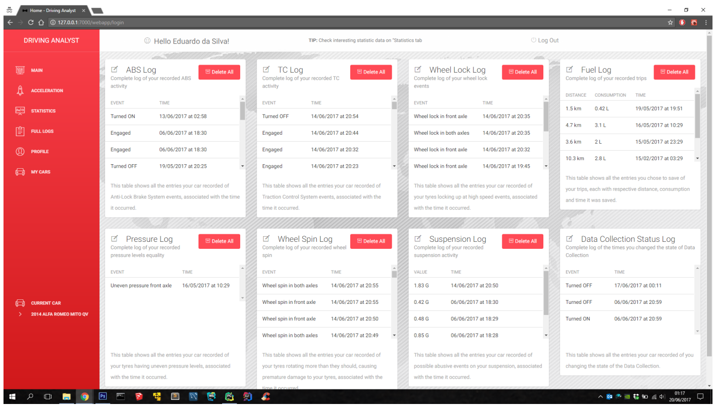
Figure 4.19: Web application "Statistics" lower dashboard

On the lower part shown in Fig. 4.19 it gets more interesting. The first table divides the information in three sections:

- **Tyre wear** This section shows statistics about events recorded that are related to the tyres on the current car. All the values shown in here are to be taken into account since once they get high enough it means tyre abuse, which means tyres getting too worn out and likely to cause an accident in the future.

- **Driving Issues** This sections shows statistics about specific aspects on the car that might jeopardise the driving experience without major visible signs, being them tyre pressures on a given axle or suspension being abused too much.
- **Miscellaneous** This section is not really related to the driving itself, however, it might be giving useful information to the user, for example, the average RPM he changes gear can be related to the fuel consumption he is experiencing.

Lastly, the bottom two tables show statistics referring to the ABS and TCS systems, giving both all-time and monthly totals regarding *ON*, *OFF* and *ACTION* events



Hello Eduardo da Silva!

TIP: Check interesting statistic data on "Statistics tab

ABS Log
Complete log of your recorded ABS activity

EVENT	TIME
Turned ON	13/06/2017 at 02:58
Engaged	06/06/2017 at 18:30
Engaged	06/06/2017 at 18:30
Turned OFF	19/05/2017 at 20:25

This table shows all the entries your car recorded of Anti-Lock Brake System events, associated with the time it occurred.

TC Log
Complete log of your recorded TC activity

EVENT	TIME
Turned OFF	14/06/2017 at 20:54
Engaged	14/06/2017 at 20:44
Engaged	14/06/2017 at 20:32
Engaged	14/06/2017 at 20:23

This table shows all the entries your car recorded of Traction Control System events, associated with the time it occurred.

Wheel Lock Log
Complete log of your wheel lock events

EVENT	TIME
Wheel lock in front axle	14/06/2017 at 20:35
Wheel lock in both axles	14/06/2017 at 20:35
Wheel lock in front axle	14/06/2017 at 20:32
Wheel lock in front axle	14/06/2017 at 19:45

This table shows all the entries your car recorded of your tyres locking up at high speed events, associated with the time it occurred.

Fuel Log
Complete log of your recorded trips

DISTANCE	CONSUMPTION	TIME
1.5 km	0.42 L	19/05/2017 at 19:51
4.7 km	3.1 L	16/05/2017 at 10:29
3.6 km	2 L	15/05/2017 at 23:29
10.3 km	2.8 L	15/02/2017 at 03:29

This table shows all the entries you chose to save of your trips, each with respective distance, consumption and time it was saved.

Pressure Log
Complete log of your recorded pressure levels equality

EVENT	TIME
Uneven pressure front axle	16/05/2017 at 10:29

This table shows all the entries your car recorded of your tyres having uneven pressure levels, associated with the time it occurred.

Wheel Spin Log
Complete log of your recorded wheel spin

EVENT	TIME
Wheel spin in both axles	14/06/2017 at 20:55
Wheel spin in front axle	14/06/2017 at 20:55
Wheel spin in front axle	14/06/2017 at 20:50
Wheel spin in both axles	14/06/2017 at 20:49

This table shows all the entries your car recorded of your tyres rotating more than they should, causing premature damage to your tyres, associated with the time it occurred.

Suspension Log
Complete log of your recorded suspension activity

VALUE	TIME
1.83 G	14/06/2017 at 20:50
0.42 G	06/06/2017 at 18:30
0.48 G	06/06/2017 at 18:29
0.85 G	06/06/2017 at 18:28

This table shows all the entries your car recorded of possible abusive events on your suspension, associated with the time it occurred.

Data Collection Status Log
Complete log of the times you changed the state of Data Collection

EVENT	TIME
Turned OFF	17/06/2017 at 00:11
Turned OFF	06/06/2017 at 20:59
Turned ON	06/06/2017 at 20:59

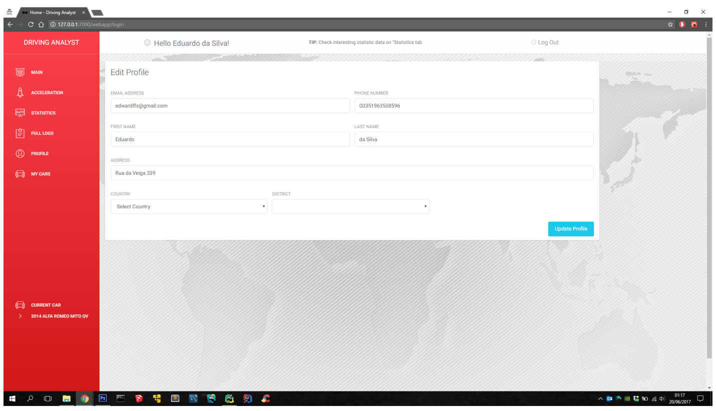
This table shows all the entries your car recorded of you changing the state of the Data Collection.

Figure 4.20: Web application "Full Logs" dashboard

In Fig. 4.20 is shown the *Full Logs* tab. As the name says, the user is given the liberty to scour all the logs ever recorded by the current selected car in its most raw state, ordered by the timestamp in descending order.

As you might have noticed, the user can also delete a log, however, he can never delete the logs of when he changed the state of the *Data Collection* log. This decision was made considering the potential this application might have to, for example, insurance companies.

This data, in this level of detail, can indicate a possible reason for an accident and so by giving the user the ability to delete that same reason will make the application useless. In order to serve the interest of entities like insurance companies and still respect the privacy of the users, a user can delete all the logs he wants except the *Data Collection status* log. This means insurance companies will know whenever the users changed the setting, if they ever changed it at all, since it comes activated by default at start-up.



Hello Eduardo da Silva!

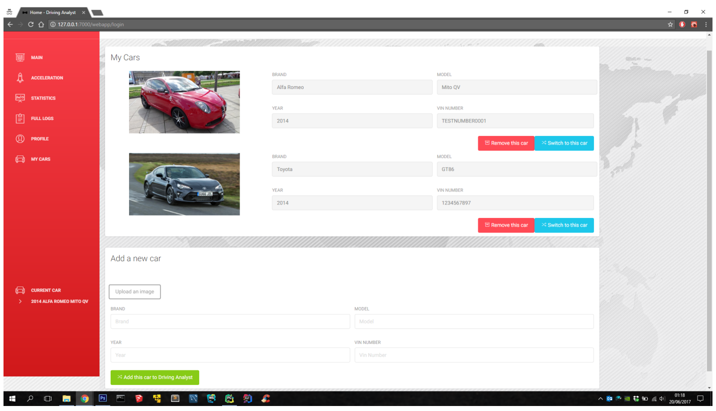
TIP: Check interesting statistic data on "Statistics tab

Edit Profile

EMAIL ADDRESS	PHONE NUMBER
edwardffs@gmail.com	00351963508596
FIRST NAME	LAST NAME
Eduardo	da Silva
ADDRESS	
Rua da Veiga 339	
COUNTRY	DISTRICT
Select Country	

Figure 4.21: Web application "Profile" dashboard

On the tab shown in Fig. 4.21 the user can edit any field of the information he entered upon registration and save it. While editing the platform will also show warnings about the fields being edited.



My Cars

BRAND	MODEL
Alfa Romeo	Mito QV
YEAR	VIN NUMBER
2014	TESTNUMBER0001
BRAND	MODEL
Toyota	GT86
YEAR	VIN NUMBER
2014	1234567897

Add a new car

BRAND	MODEL
Brand	Model
YEAR	VIN NUMBER
Year	Vin Number

Figure 4.22: Web application "My Cars" dashboard

Lastly, On figure 4.22 is shown the basis of all the information previously showcased in this section. In here the user is able to switch between any of its cars, which will cause for the information on all the other tabs (except the "Profile" one) to change contents, showcasing only the data regarding the chosen car.

The user is also able to delete any of his cars, that will cause all of its logs to be deleted as well from his account.

Lastly, the user can add new cars to his "garage" on the platform. To do so, he simply needs to fill some information regarding the new car and upload a picture of it for future identification.

Once the car is added, the user can only consult it, in any instance can he ever change any of the fields.

4.4.2 Structure

As was explained before, the online platform is built using ReactJS. That is true, however, there are other technologies at play here. This section will detail those same technologies and explain how the ReactJS code was organised providing maximum maintainability and modularity.

Since the login and register operations are operations that, besides not requiring a lightning fast response time, would have a completely different design than the one chosen for the web application. With this in mind, both login and register pages are rendered using HTML and *Jinja2*. To put it simple, *Jinja2* is a templating engine that comes with Flask by default and allows for dynamic creation of HTML code and has a really close relationship to Flask. This tool seemed a "good-enough" tool for these two operations.

As for the others, they are built using pure ReactJS code, carefully organised by small components that are rendered only when needed.

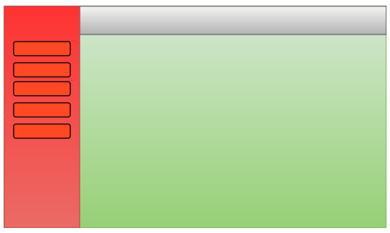


Figure 4.23: Web Application Structure

On Fig. 4.23 is represented the main structure of the web application. As we can see there are three distinct components: the navigation bar represented in grey, the sidebar represented in red and the content in green.

On the navigation bar the user can see his current name, log out the platform and read some driving recommendations and tips about the platform that change randomly.

The side bar, however, is a lot more functional than that. Every time a user clicks one of the buttons, the content of the page (green) field is filled with the respective components. This means, the web page is actually re-rendered instead of reloaded, thanks to ReactJS.

To make the design of the website *Bootstrap* was used whenever possible. Besides being the most popular HTML, CSS, and JavaScript framework for developing responsive "mobile-first" web sites, it is completely free to download and use.

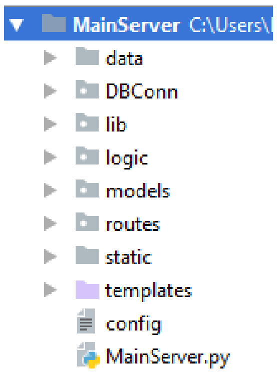
Finally, to aid in showing graphic information on the platform was used *Chartist*, a simple responsive charting library built with Scalable Vector Graphics (SVG). This allowed to easily build graphics dynamically with information provident from the API in its pretty much raw state.

4.5 API / Server

This section will explain in detail all there is to know about the server that holds both the API and the online platform.

4.5.1 File Structure

As was mentioned before on this document, this piece of software is written in Python, using a micro-framework called Flask. This framework does not have any restriction as to how the files should be organised so far. However, that is no excuse to tidy up the code.



```
MainServer C:\Users\
    data
    DBConn
    lib
    logic
    models
    routes
    static
    templates
    config
    MainServer.py
```

Figure 4.24: API File Structure

On the left we can see the file structure of the API. Each of those folders represents a different kind of Python scripts, with different purposes, with the exception of the first folder whose purpose is solely to save data from mainly acceleration times. **DBConn** contains what could be called *beans* in other frameworks. Each script contains a class with similar attributes to the model in question (for example, to handle operations with a *User* we would use a *user_conn* object) that is responsible to exclusively communicate with the database. This way the only scripts that are in contact with the database are located in one place instead of being scattered all over the code.

The **lib** and **logic** contain utilitarian methods that are used over and over by the API. These contain mainly methods for conversions, security, content validation, email sending, treat data, among others.

The **models** folder contains all the classes required by *SQLAlchemy* to successfully create and manage the database. There is a script for each class, each with a set of attributes and respective keys like any other database. At the server start-up these are loaded and either create or modify the database according to the changes made.

In the **routes** folder are the scripts that contain each and every route the API currently has available. They are separated in scripts, differentiated by whoever accesses them (a car, a user or the platform).

The **static** and **templates** folders contain all the code required for the front-end to work properly, being all the HTML files in the *templates* folder (as required by the Flask framework) and all the others (Javascript and CSS) are located inside the *static* folder.

Finally, there are two files separated from all the others. The **config** one holds vital information that should not be scattered all over the code, like email credentials. The **MainServer** script is responsible for initialising all the scripts that were described before and leave the server up and running.

4.5.2 Structure Diagram

In this section will be presented some diagrams representing the multiple views of the project, accompanied by a brief explanation.

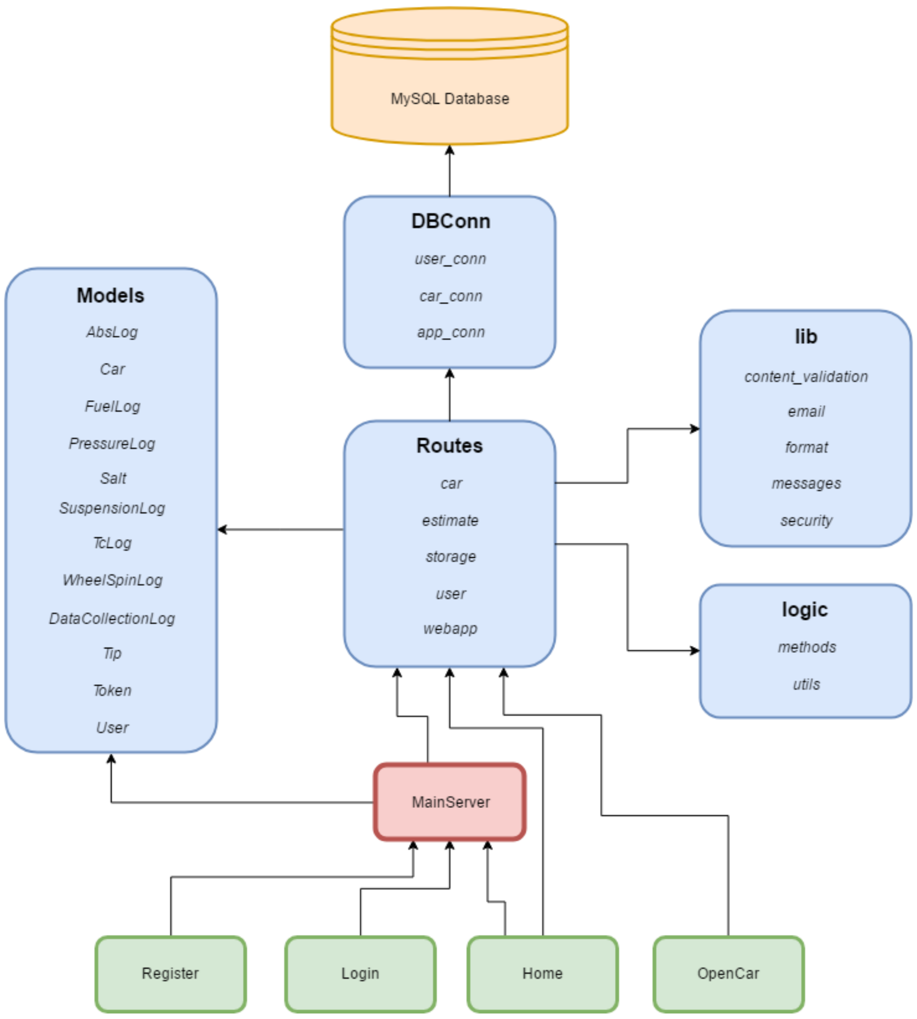


Figure 4.25: API file diagram

In this diagram we can visualise all that was explained in the previous subsection. Starting from the top, there is the database that only interacts with the scripts in the **DBConn** folder.

Represented in blue are all the different folders containing their respective scripts. The **Routes** is not located in the middle by chance. It contains the core scripts of this software, hence the need to communicate with pretty much all the other folders.

In red is the main script that initialises everything. It must therefore communicate with the **Routes** folder in order to put up all the routes needed and with the **Models** folder so it can setup the Database correctly.

Lastly, there are there is the user-level, being the **Register**, *Login* and *Home* pages from the online platform and the *OpenCar* all the routes the console application uses either to store or retrieve data.

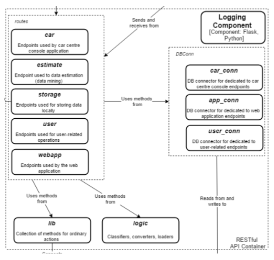

routes
car
Endpoints used by car centre console application
estimate
Endpoint used to data estimation (data mining)
storage
Endpoints used for storing data locally
user
Endpoints used for user-related operations
webapp
Endpoints used by the web application
Sends and receives from
Logging Component
[Component: Flask, Python]
DBConn
car_conn
DB connector for dedicated to car centre console endpoints
app_conn
DB connector for dedicated to web application endpoints
user_conn
DB connector for dedicated to user-related endpoints
Uses methods from
Uses methods from
Uses methods from
Reads from and writes to
lib
Collection of methods for ordinary actions
logic
Classifiers, converters, loaders
RESTful API Container

Figure 4.26: Component representation

On the left we can see the API representation in Fig. 3.6 in previous sections. As we notice, it does not differ much from the diagram in Fig. 4.25.

4.5.3 Routes Implemented

In this section are listed and explained all the routes that compose the API present on this project on Tables E.1 and E.2, separated by the ones being used by the centre console's application and the ones being used by the online platform application.

4.6 Database

In this section will be explained in detail the whole structure of the database present on this project and all of its contents.

4.6.1 Tables

The database working on this project has a total of 14 tables, being them:

- **user** Table containing all the information regarding the users of the platform
- **token** Table containing the login tokens of each user on the platform. This token is updated on every login

- **salt** Table containing *"salt"* strings. These strings are generated on the *register* procedure of a new user and is used to concatenate to the user password hash, providing an extra layer of security
- **car** Table containing all the information regarding the cars of every user
- **tip** Table containing a list of sentences that keep changing on the online platform's navigation bar. A slight "easter-egg" for its users.
- **abslog** Table containing every registered activity of the ABS system in all the cars currently on the platform, logging ON, OFF and ACTION events with a timestamp associated.
- **tclog** Table containing every registered activity of the TCS system in all the cars currently on the platform, logging ON, OFF and ACTION events with a timestamp associated.
- **fuellog** Table containing every registered activity of the trips the users ever chose to save record, in all the cars currently on the platform
- **gearchangerpmlog** Table containing every gear engaged event by the users while driving, logging the RPM value in which a gear was engaged.
- **pressurelog** Table containing every event recorded of cars having uneven pressure on their tyres, logging which axle has the uneven values and a timestamp associated.
- **suspensionlog** Table containing every event recorded of the cars' suspension hitting unusual G-force values, therefore damaging the suspension, logging the value registered with a timestamp associated
- **wheellocklog** Table containing every event of the wheels locking up after hard-braking on the user's cars, logging which axle experienced this event and a timestamp associated.
- **wheelspinlog** Table containing every event of the wheels spinning faster or slower than the car itself, causing the car to become unstable, logging which axle experienced this event and a timestamp associated.
- **datacollectionlog** Table containing all the times the user has changed the setting *Data Collection* on the centre console's application of his car, logging OFF events with a timestamp associated

4.6.2 ER Diagram

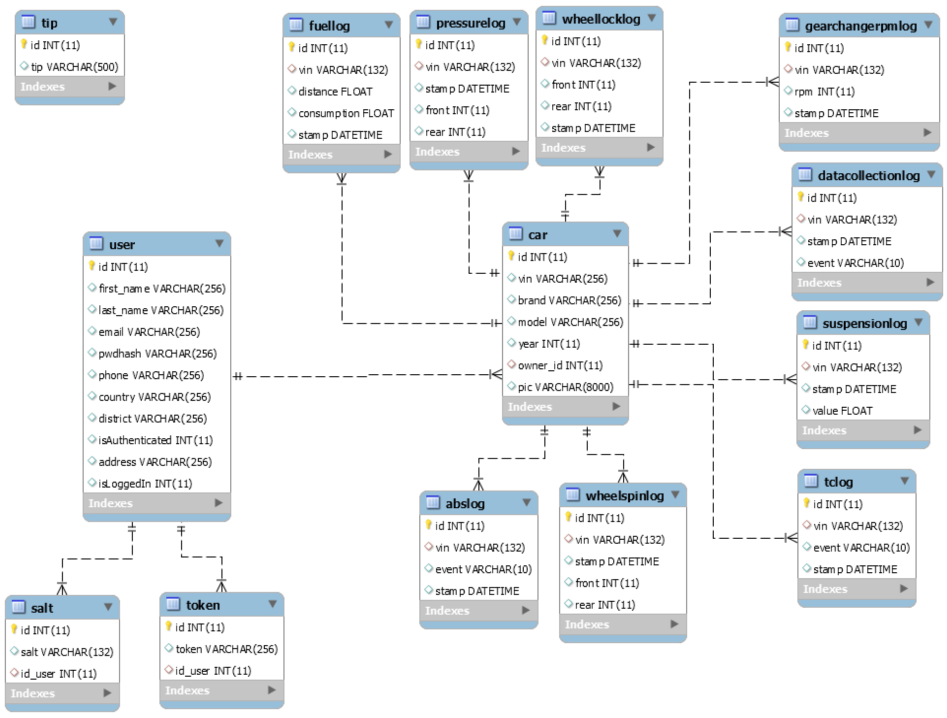


Figure 4.27: Database ER diagram

Chapter 5

Work Plan

In this section we will detail the main flags to hoist in order to successfully reach the end of development and have a final product worth presenting. An overview of the possible implications will also be given, since these same implications can compromise the healthy development of the project

5.1 Milestones

In order to successfully complete the project at hand, the progress will be made through achieving a series of milestones. These milestones consist in developing certain components of the system while trying to connect them in the process. Ideally, the development should be gradually documented as well.

The first milestones will consist more in an investigative procedure, being them:

- **Identify relevant events to register** It comes without question that OpenCar provides lots of information. One must not get lost in such a variety of information, therefore, and given the data provided by the driving simulator, a clever selection of the most crucial events to monitor and related sensors must be made.
- **Determine how the events will be classified** Having the events selected, it is time to classify them. This classification should consists in distinguishing them, for example, into dangerous actions that could have been the cause of an accident, bad behaviour or simply statistic data.
- **Determine how these events will be re-created through the simulator** Once determined all the events and respective classifications, some scenarios must be defined in order to re-create situations

in which the events previously mentioned would be triggered in a real-world context.

- **Study the best data format and database schema** Lastly, the format the data to be travelling across the system and how this data will be saved in the database must be well defined in order to avoid any refactor in the future.

The next phase is to start studying how the application will look and function, being the main milestones to achieve:

- **Study the look and feel of the in-vehicle application** Although OpenCar's main purpose in this project is to collect data, it does not mean that it should not benefit from a usable and pretty enough display. The in-vehicle application should do what is expected and still be easy to use.
- **Develop the in-vehicle application** This milestone will most definitely one of the longest and hardest to achieve. This includes both developing the look studied before and all the logic needed to extract data, as well as defining how frequently the sensors are checked, how the data is temporarily saved, among other minor details.
- **Improve the connector developed in the early stages of this project** Since the tests made previously were a success the most part of the software can be re-used, more specifically the server that is connecting the driving simulator to the OpenCar simulator. However, it will have to suffer improvements in order to satisfy the needs of this project as well as further testing, this time more enduring.
- **Develop the API** Once the in-vehicle application and the connector reach an Minimum Viable Product (MVP) state or higher, it is time for start developing the API and its routes. This milestone is due to take a long time developing and testing since it will connect the whole system together.
- **Study the look and feel of the online web-application** Much like the in-vehicle application, the online application should also benefit from a visibly comfortable and usable UI.
- **Develop the web application** On a final phase of the development the platform would be implemented, making full use of the routes available in the API.
- **Document the software** Last but not least, the software should be well documented in order to promote a possible continuous integration.

5.2 Setbacks

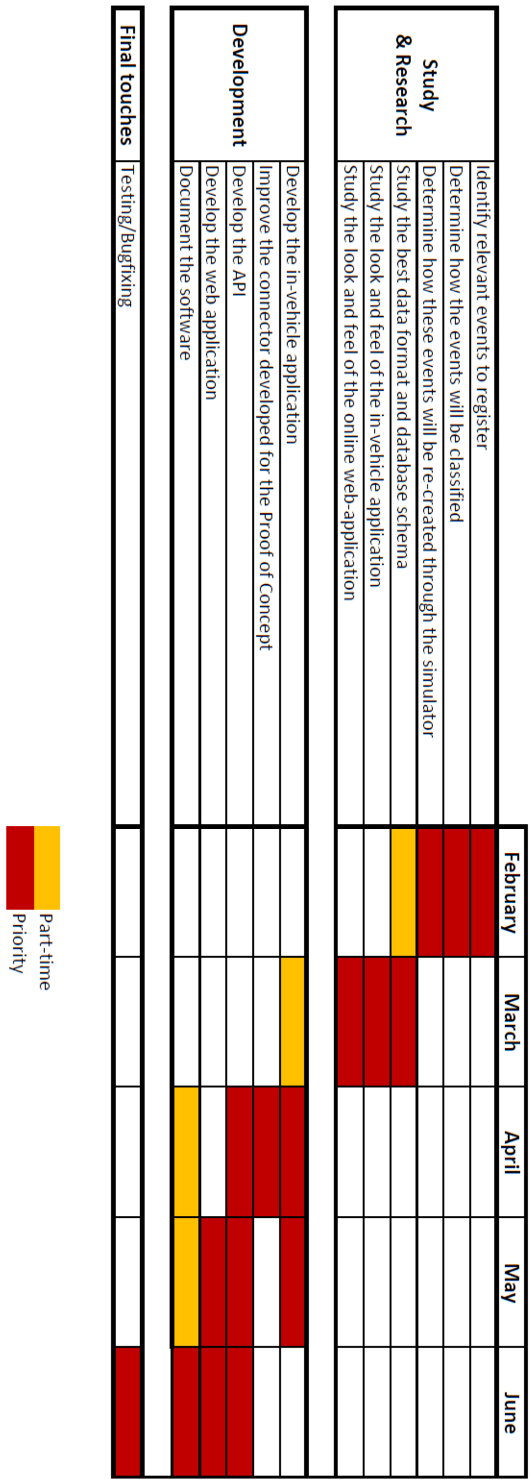
Every software project has its implications and risks, and this project is no different. In this section we will list and describe some situations that could possibly provide setbacks to this project.

- **Developer Support** As was mentioned before, we will be limited to documentation and code samples while developing with OpenCar's framework. So far developer support has shown commitment and interest in helping, however, it may take too long to answer or not answer at all. This can harm greatly the development process of this project.
- **Bugfixing & Research** As any software project, bugfixing will be a long and enduring phase of the project. It must not take too long stealing time to other important matters or worse, stall the development. The same goes to researching for a method to solve a given problem or even fix an error, this should not take too long as well.

5.3 Gantt diagram

In this section is presented the *Gantt* diagram displaying the planning done to the semester in which this project is developed, considering the descriptions given in the previous sections.

As we can see in Fig. 5.1, the first month is dedicated to identifying events that might be of interest and how the data collected will be treated. The following month is dedicated to studying the UI's of both the in-vehicle application and the web-application in order to properly start the development in the next months, knowing exactly what the applications will require/produce. When developing is supposed to gradually document the software, even though it is not a priority task.



		February	March	April	May	June
Study & Research	Identify relevant events to register	🟥				
	Determine how the events will be classified	🟥				
	Determine how these events will be re-created through the simulator	🟥				
	Study the best data format and database schema	🟨	🟥			
	Study the look and feel of the in-vehicle application		🟥			
	Study the look and feel of the online web-application		🟥			
Development	Develop the in-vehicle application		🟨	🟥	🟥	
	Improve the connector developed for the Proof of Concept			🟥		
	Develop the API			🟥	🟥	🟥
	Develop the web application				🟥	🟥
	Document the software			🟨	🟨	🟥
Final touches	Testing/Bugfixing					🟥

🟨	Part-time
🟥	Priority

Figure 5.1: Gantt diagram

Chapter 6

Conclusion

Upon the start of this project, the concept of *Connected Cars* was somewhat unknown, as we barely understood what it actually meant. The development of this project made us look into it not only as a rising market, but also as an opportunity to apply our expertise in software engineering in a sector that has been denied to programmers for as long as a few years ago.

This project in particular was challenging. Besides the technology being rather new, the search for a way to validate and test the final product once it was finished was short but a difficult one. This meant hours spent on documentation and emails exchanged with developer support. Even after that was done, there was a need to understand this market, namely what was already available and what can still be offered, which also meant a research both on major entities on this market and already successful projects.

However, the ultimate challenge was to understand a few physics concepts that would make our idea work and actually provide some usefulness to its possible users. Understanding the behaviour of a car to the minimum detail was the first step. Concepts like heat, pressure, friction, variations of speed both on the car as a whole and independently, and even the basics of an engine functioning had to be taken into account to provide the product being presented in this document and the best results possible. Luckily, the simulation software accurate physics engine helped by a great margin, since we were able to test multiple scenarios without the cost of damaging any real car while monitoring multiple variables about the vehicle behaviour.

The final result of this project is a functional centre console application using OpenCar framework, an API that provides both real-time telematic information and statistics based on that same information for the user to see. All of this is happening in a controlled simulated environment that we were able to connect to the OpenCar simulator in order to get the most realistic possible results given the tools at our disposal. This telematic data is then used to calculate statistics to be shown in a light, user-friendly web application along with graphical information and even the logs in a more raw state for the most meticulous users.

In the tests conducted, the user was able to see what the car had been enduring for the past times and be compelled to take additional care of his car in a real-world scenario and, consequently, of himself while driving. Not only that, the centre console application can also monitor and warn him in real-time of these same events that were being recorded on the the database. As for the non-automated functionality of the application, the user was also able to save information and analyse it in detail on the online platform as was expected. All of this on a stable, simulated environment. It is safe to say that the tests looked promising and show that this software has potential both among driving enthusiasts and ordinary drivers.

However, it could prove useful to many other situations. One that is probably the best bet if this product ever reaches the market are the insurance companies. Since this software is capable of telling when the wheels of a car lost traction or if the traction control is turned on or off is something that could mean these companies paying or not for an accident, since turning off this feature is only recommended for very skilled drivers, however still dangerous, and losing traction could happen easily on a snowy road.

In the future, we plan to completely rework the *Controller* component of the OpenCar application to remove all the code that was reading from the simulation software and revert back to its original architecture in which the application reads directly from what could someday be a real car. This way the application will be ready to be submitted on the INRIX platform for further evaluation, and consequently get some feedback on how to improve it. This evaluation follows a certain set of parameters to guarantee the quality of the product, of course, but once it has a green light there is hope in one day seeing *Driving Analyst* on the road, on a real car this time, helping drivers day in day out.

Depending on its success at both the eyes of OpenCar's professionals and the users, a rework of the web application and improvements on the API are also something to look at, as well as new ways to improve the statistics. This will mean more people working on the project from different areas although with the same mindset: bring the driving experience closer to the user, safely.

Bibliography

[1] Matthias Stuempfle Akhtar Jameel, Axel Fuchs. Internet multimedia on wheels: Connecting cars to cyberspace, 1998.

[2] GENIVI Alliance. Reference architecture. https://www.genivi.org/sites/default/files/resource_documents/GENIVI_Reference_Architecture_29Oct2015.pdf, 2015.

[3] GENIVI Alliance. Faq. https://www.genivi.org/about-genivi, 2016.

[4] Apple. Apple carplay - the ultimate copilot. http://www.apple.com/ios/carplay/, 2016.

[5] AUTOSAR. Autosar - about. https://www.autosar.org/about/basics/, 2016.

[6] AUTOSAR. Autosar - technical overview. https://www.autosar.org/about/technical-overview/ecu-software-architecture/, 2016.

[7] Simon Brown. C4 model, *Software Architecture for Developers*, 2012.

[8] Android Central. Android auto. http://www.androidcentral.com/android-auto, 2016.

[9] The Driving Club Coimbra. https://www.facebook.com/thedrivingclub/, 2016.

[10] Gadget Daily. Automotive grade linux. https://www.gadgetdaily.xyz/wwdc-tickets-sell-out-in-just-under-two-minutes/, 2012.

[11] Embedded Computing Design. Integrating linux into automotive systems for the long haul. http://embedded-computing.com/articles/integrating-systems-the-long-haul/, 2013.

[12] Wikipedia EN. Qnx. https://en.wikipedia.org/wiki/QNX, 2016.

[13] Flask. Flask. http://flask.pocoo.org/, 2017.

[14] Ford. Applink. https://developer.ford.com/pages/applink/, 2016.

[15] Ford. Ford applink catalogue. http://www.ford.co.uk/OwnerServices/SYNC-and-Bluetooth-Support/SYNC-App-Link-Catalogue#/, 2016.

[16] Ford. Ford sync. http://www.ford.com/technology/sync/, 2016.

[17] Ford. Welcome to fordpass. https://www.fordpass.com/, 2016.

[18] GENIVI. Genivi compliant - get started. https://www.genivi.org/genivi-compliant-get-started, 2016.

[19] Google. Android auto. https://www.android.com/auto/, 2016.

[20] Automotive Grade Linux. Automotive grade linux requirements specification. www.automotivelinux.org, 2015.

[21] MarketsAndMarkets. Connected car market worth $46.69 billion by 2020, 2013.

[22] Wikipedia PT. Qnx. https://pt.wikipedia.org/wiki/QNX, 2016.

[23] QNX. Qnx car - about. http://www.qnx.com/content/qnx/en/products/qnxcar/index.html, 2016.

[24] Kunos Simulazioni. Assetto corsa. http://www.assettocorsa.net/en/, 2017.

[25] TechRadar. Apple carplay: everything you need to know about ios in the car. http://www.techradar.com/news/car-tech/apple-carplay-everything-you-need-to-know-about-ios-in-the-car-1230381, 2016.

[26] Tizen. About. https://developer.tizen.org/tizen/about, 2016.

[27] LTSI Workgroup. What is ltsi? http://ltsi.linuxfoundation.org/what-is-ltsi, 2012.

Appendices

Appendix A

GENIVI Member List

Table A.1: GENIVIS's Original Equipment Manufactures

Original Equipment Manufacturers	BMW Group
	Great Wall Motors
	Honda
	Hyundai Motors Group
	Jaguar / Land Rover
	Mercedes-Benz R&D
	Nissan Motor Co. Ltd.
	PSA Groupe
	Renault SAS
	SAIC Motor Passenger Vehicle
	Volvo Car Corporation

Table A.2: GENIVI's First Tier Members

First Tiers	AISIN AW	Hyundai Mobies Co.
	Alpine Electronics R&D	LG Electronics
	ALPS Electric Europe	Magneti Marelli
	AppDirect	Mitsubishi Electric Corporation
	Clarion Co.	Peiker Acustic
	Continental Automotive	Pioneer Corporation
	Delphi	Robert Bosh Car Multimedia
	Denso Corporation	Trend Micro
	Harman International Industries	Visteon Corporation
	Huizhou Desay SV Automotive	

Table A.3: GENIVI's OSV, Middleware, Hardware, and Ser-vice Suppliers

Abalta Technologies	Ericsson AB	Neusoft Technology Solutions
Accenture	Excelfore Corporation	NNG
Access Europe	FPT Software Hanoi Company Limited	NTT DATA MSE Corporation
Actia Nordic	Fujitsu Semiconductor Europe	OBIGO
Airbiquity	Garmin Switzerland	**OpenCar**
Allgo Embedded Systems	GlobalLogic	OpenMobile World Wide
Altera Corp.	Green Hills Software	OpenSynergy
Argus Cyber-Security	HCL Technologies Limited	Palamida
Aricent Group	Hortonworks Inc.	PathPartner Technology
Arkamys	Huizhou Foryou General Electronics	Pelagicore
ATS Advanced Telematic Systems	IAV	PolySync
Audiokinetic	Igalia	QuEST Global Engineering Services
AutoNavi Software	Integrated Computer Solutions	Rogue Wave Software
AVE AutoMedia	Intive	Sasken Communication Technologies
BearingPoint	Irdeto	Shenyang MXNavi
Black Duck Software	itemis AG	Smartcar
Capgemini	IVIS Co.	Suntec Software (Shanghai)
CARFIT	Karamba Security	Tata Consultancy Services
Cinemo	Konsulko Group	TATA ELXSI
Codethink	KPIT Technologies	Telemotive AG
Cogent Embedded	Link Motion	The Qt Company
Collabora Limited	Luxosoft	t1nnos
CTAG	Mapbox	Tom Tom International
Cybercom	MediaTek	Tuxera
Drive Time Metrics	Mentor Graphics Corporation	UIEvolution
Elektrobit Automotive	Mobica Limited	Wind River
EnGIS Technologies	Murata Manufacturing	Workfrom
EPAM Systems	Myine Electronics	
Ericpol	Navis Automotive Systems	

Table A.4: GENIVI's Silicon and Other Members

Silicon	Analog Devices
	ARM
	CSR Technology
	Intel
	ISSI
	NVIDIA
	NXP Semiconductors Netherlands B.V.
	Renesas Electronics
	ROHM
	Telechips
	Texas Instruments Incorporated
Other	W3C

Appendix B

QNX Partner List

Table B.1: QNX Partners

Company	Integration
Apple	Mobile device connectivity
Best Parking	Location-based parking search
Digia	Qt commercialization
Elektrobit	Embedded navigation
Freescale	Silicon vendor
Google	Mobile device connectivity
Hear Planet	Internet audio streaming service
HERE	Embedded navigation
Intel	Silicon vendor
JQuery	JavaScript framework
Livio (Ford Subsidiary)	Mobile device connectivity
Nuance	Speech recognition
NVIDIA	Silicon vendor
OpenSynergy and Cybercom	Bluetooth
Pandora	Streaming internet radio
Parkopedia	Location-based parking search
Qualcomm	Silicon vendor
RealVNC	MirrorLink connectivity
RedBend	FOTA software updates
Renesas	Silicon vendor
Sencha	JavaScript framework
Slacker	Streaming internet radio
Soundtracker	Internet music streaming service
Texas Instruments	Silicon vendor
Wcities Eventseekr	Location-based event service

Appendix C

Functional Requirements

Table C.1: Centre console's application functional requirements

ID	**Description**	**Priority**
01	Allow the User to check if his car is already registered in a given account	MUST
02	Allow the User to visualise a dashboard with real-time telematic information about the car's current status	MUST
03	Allow the User to visualise real-time information about the trip he is currently making	MUST
04	Allow the User to save the current information about the trip he is making on the database	MUST
05	Allow the User to reset the current information about the trip he is making	MUST
06	Allow the User to receive directions to the nearest petrol station	SHOULD
06	Allow the User to measure 0-100km/h acceleration times	MUST
07	Allow the User to estimate 0-100km/h acceleration times given the current conditions and previous performances	MUST
08	Allow the User to save the last 0-100km/h acceleration time in the database	MUST
09	Allow the User to visualise small statistics about his car's ABS activity	SHOULD
10	Allow the User to visualise small statistics about his car's TCS activity	SHOULD
11	Allow the User to visualise his top 5 best acceleration times	SHOULD
12	Allow the User to switch off the Data Collection	MUST
13	Allow the User to change the temperature unit	SHOULD
14	Allow the User to change the pressure unit	SHOULD
15	Allow the User to change the speed unit	SHOULD
16	Allow the User to navigate between screens through an action bar	MUST

Table C.2: Platform's Functional Requirements

ID	**Description**	**Priority**
01	Allow the User to register a new account on the platform	MUST
02	Allow the User to log in the platform	MUST
03	Allow the user to register a new car on his account	MUST
04	Allow the User to visualise the latest events recorded by the currently chosen car	MUST
05	Allow the User to visualise and analyse in detail all of his acceleration times graphically	MUST
06	Allow the User to visualise calculated statistics about aspects related to his driving and a given car	MUST
07	Allow the User to visualise the full extent of the logs recorded by a given car	MUST
08	Allow the user to delete any of his logs of the car that is currently selected	SHOULD
09	Allow the user to upload a picture of his car upon registration	SHOULD
10	Allow the user to edit his personal information	MUST
11	Allow the user to add new cars to his account	SHOULD
12	Allow the user to switch between cars in his account	SHOULD
13	Allow the user to delete a car from his account	SHOULD
14	Allow the user to log out from the platform	MUST

Appendix D

Assetto Corsa/OpenCar: Information cross-check

Table D.1: Telematic Data provided by *Assetto Corsa* coincident with Open-Car's

speed_Kmh	Vehicle's current speed
isAbsEnabled	Check if ABS is turned ON
isAbsInAction	Check if ABS is taking action
isTcEnabled	Check if Traction Control is turned ON
isTcInAction	Check if Traction Control is taking action
accG_vertical	Vertical acceleration of the vehicle
accG_horizontal	Horizontal acceleration of the vehicle
accG_frontal	Frontal acceleration of the vehicle
gas	Gas pedal position
brake	Brake pedal position
clutch	Clutch pedal position
engineRPM	Engine speed
steer	Steering angle
gear	Gear engaged
wheelAngularSpeed[4]	Individual rotating speed of each wheel
wheelPressure[4]	Individual pressure value of each wheel
wheelTemperature[4]	Individual temperature value of each wheel
distanceTravelled	Overall distance travelled since the start of the simulation
tyreRadius[4]	Individual diameter of each wheel
maxFuel	Maximum capacity of the vehicle's gas tank
currentFuel	Current fuel currently on the vehicle's gas tank

Appendix E

Routes Implemented

Table E.1: Routes Implemented for the centre console's application

Route	Method	Headers	Payload	Description
/checkvin	POST	None	VIN Number	Method that checks if the car is registered on the platform
/suspensionHit	POST	None	VIN Number, Timestamp, Value	Method that inserts records of any unusual G-Force experienced by the car's suspension
/unevenPressure	POST	None	VIN Number, Timestamp, Front, Rear	Method that inserts records of the car's axles ever having uneven pressure on the tyres
/wheelLock	POST	None	VIN Number, Timestamp, Front, Rear	Method that inserts records of the car locking the wheels up under hard-braking, in each axle
/wheelSpin	POST	None	VIN Number, Timestamp, Front, Rear	Method that inserts records of the car spinning the wheels, in each axle
/rpmLog	POST	None	VIN Number, RPM, Timestamp	Method that registers every gear change event and register the respective engine speed and timestamp
/tcevent	POST	None	VIN Number, Event, Timestamp	Method that registers ON, OFF and ACTION events happening in the car's ABS system
/absevent	POST	None	VIN Number, Event, Timestamp	Method that registers ON, OFF and ACTION events happening in the car's TCS system
/fuel	POST	None	VIN Number, Consumption Distance, Timestamp	Method that registers data from trips the user chooses to save on the database
/absstats	POST	None	VIN Number	Method that retrieves a small set of information regarding the ABS activity so far, bringing it to the car's centre console
/tcstats	POST	None	VIN Number	Method that retrieves a small set of information regarding the TCS activity so far, bringing it to the car's centre console
/dataCollection	POST	None	VIN Number, Timestamp	Method that registers a timestamp of when the driver turns off the Data Collection setting on the centre console's application
/estimate	POST	None	VIN Number, WheelTemperature[4], WheelPressure[4], RPM, TCS State, Throttle Position	Method that receives the current car's conditions for acceleration measurement and returns an estimate of how much time he could do, given previous attempts
/accdata	POST	None	VIN Number, WheelTemperature[4], WheelPressure[4], Speed[x], FrontLeftWheelSpeed[x], FrontRightWheelSpeed[x], RearLeftWheelSpeed[x], RearRightWheelSpeed[x], RPM[x], gearChangeRPM[x], gearChangeTime[x], Time, TC State, Throttle Position	Method that saves all the data collected in and acceleration measurement to the server.
/besttimes	POST	None	VIN Number	Methods that returns a list of the top five accelerations times from the server to show in the centre console's application

Table E.2: Routes Implemented for the web application

Route	Method	Headers	Payload	Description
/webapp/login	POST	Content-type, Accept	Username, Password	Method that checks the credentials of the user and redirects the user to the web application
/webapp/register/user	POST	Content-type, Accept	First Name, Last Name, Email, Password, Phone, Country, State, Address	Method that registers the user on the platform and sends the authentication email
/webapp/authenticate/	GET	None	None	Method that authenticates a user's account after registration
/logout	GET	Content-type, Accept, Token	VIN Number, Timestamp, Front, Rear	Method that inserts records of the car locking the wheels up under hard-braking, in each axle
/dashboardInfo	GET	Content-type, Accept, Token	None	Method that returns data to the "Main" dashboard
/profileInfo	GET	Content-type, Accept, Email	None	Method that returns the current profile info of the logged user
/getCars	GET	Content-type, Accept, Token	None	Method that returns the list of cars currently registered on the user's account
/tips	GET	None	None	Method that returns a random tip from the database
/updateProfile	POST	Content-type, Accept, Token	First Name, Last Name, Email, Password, Phone, Country, State, Address	Method that updates the profile info with the data inserted on the platform
/getAccelerationList	POST	Content-type, Accept, Token	VIN Number	Method that returns the list of all the acceleration measurements and respective timestamps
/getAccInfo	POST	Content-type, Accept, Token	VIN Number, Filename	Method that retrieves all the information regarding a specific acceleration measure of a given car
/getAbsFullLog	GET	Content-type, Accept, Token, VIN Number	None	Method that returns all the records of ABS activity of a given car
/getTcFullLog	GET	Content-type, Accept, Token, VIN Number	None	Method that returns all the records of TCS activity of a given car
/getSuspensionFullLog	GET	Content-type, Accept, Token, VIN Number	None	Method that returns all the records of unusual suspension activity of a given car
/getPressureFullLog	GET	Content-type, Accept, Token, VIN Number	None	Method that returns all the records of uneven pressure on a given car's tyres
/getFuelFullLog	GET	Content-type, Accept, Token, VIN Number	None	Method that returns all the records of distances and cinsumptions of a given car
/getWheelSpinFullLog	GET	Content-type, Accept, Token, VIN Number	None	Method that returns all the records of wheel-spin activity of a given car
/getWheelLockFullLog	GET	Content-type, Accept, Token, VIN Number	None	Method that returns all the records of wheel lock-up activity of a given car
/getDataCollectionFullLog	GET	Content-type, Accept, Token, VIN Number	None	Method that returns all the logs of the driver ever turning off theData Collection setting on the centre console's application
/getFuelStats	GET	Content-type, Accept, Token, VIN Number	None	Method that returns statistic data regarding fuel consumption of a given car
/getDistanceStats	GET	Content-type, Accept, Token, VIN Number	None	Method that returns statistic data regarding distance travelled of a given car
/getAbsStats	GET	Content-type, Accept, Token, VIN Number	None	Method that returns statistic data regarding ABS activity of a given car
/getTcStats	GET	Content-type, Accept, Token, VIN Number	None	Method that returns statistic data regarding TCS activity of a given car
/getMiscStats	GET	Content-type, Accept, Token, VIN Number	None	Method that returns various statistical information of a given car
/deleteCar	DELETE	Content-type, Accept, Token, VIN Number	None	Method that removes a car and all its logs from the database
/addCar	POST	Content-type, Accept, Token	Brand, Model, Year, VIN Number, Picture	Method that adds a car to a given user's account
/deleteAbsLog	DELETE	Content-type, Accept, Token, VIN Number	None	Method that deletes all the logs regarding ABS activity from a given car
/deleteTcLog	DELETE	Content-type, Accept, Token, VIN Number	None	Method that deletes all the logs regarding TCS activity from a given car
/deleteFuelLog	DELETE	Content-type, Accept, Token, VIN Number	None	Method that deletes all the logs regarding fuel consumption and distance travelled from a given car
/deletePressureLog	DELETE	Content-type, Accept, Token, VIN Number	None	Method that deletes all the logs regarding uneveven pressure events from a given car
/deleteSuspensionLog	DELETE	Content-type, Accept, Token, VIN Number	None	Method that deletes all the logs regarding unusual suspension activity from a given car
/deleteWheelSpinLog	DELETE	Content-type, Accept, Token, VIN Number	None	Method that deletes all the logs regarding wheel-spin activity from a given car
/deleteWheelLockLog	DELETE	Content-type, Accept, Token, VIN Number	None	Method that deletes all the logs regarding wheel lock-up activity from a given car

Appendix F

Console Application Use-Case Tables

Title	**Verify if car is registered on the platform**
ID	UC01
Primary actor	OpenCar application
Description	1. The use case starts as soon as the user launches the application on the OpenCar's centre console. The application itself will request the API for confirmation if the car is already registered to an account on the platform. 2. If it does not, the application will not launch, instead, it will show instructions on how to register the car on the platform.
Assumptions	OpenCar working correctly, API is up and running.
Input	Vehicle's VIN number
Output	Confirmation if the car is indeed registered on the platform
Exceptions	1. OpenCar is malfunctioning 2. API is down
Priority	High

Title	**Show Telematic Information in real-time**
ID	UC02
Primary actor	OpenCar application
Description	1. The use case starts as soon as the user launches the application on the OpenCar's centre console and it successfully confirms that the car is registered. It will then show a dashboard containing various real-time data concerning the car's running status
Assumptions	OpenCar working correctly, API is up and running. Car is registered on the platform.
Input	None
Output	Real-Time telematic data
Exceptions	1. OpenCar is malfunctioning 2. API is down 3. Car is not yet registered in any account on the platform
Priority	High

Title	**Register driving events**
ID	UC03
Primary actor	OpenCar application
Description	1. The use case starts as soon as the user launches the application on the OpenCar's centre console and it successfully confirms that the car is registered.While the user is driving, it will be alert to recognise any events that might cause premature wear on some of the car's parts, as well as jeopardise the user's driving experience.
Assumptions	OpenCar working correctly, API is up and running. Car is registered on the platform.
Input	Telematic data associated with timestamps
Output	None
Exceptions	1. OpenCar is malfunctioning 2. API is down 3. Car is not yet registered in any account on the platform
Priority	High

Title	Show information concerning the ongoing trip
ID	UC04
Primary actor	End-user
Description	1. The use case starts as soon as the user launches the application on the OpenCar's centre console and it successfully confirms that the car is registered. By pressing the "leaf" button on the action bar the user will be shown a screen containing information about the ongoing trip, such as fuel, distance and consumption efficiency.
Assumptions	OpenCar working correctly, API is up and running. Car is registered on the platform.
Input	"Leaf" button pressed on the action bar
Output	Dashboard containing real-time information about the ongoing trip
Exceptions	1. OpenCar is malfunctioning 2. Car is not yet registered in any account on the platform
Priority	High

Title	Reset Information on Fuel dashboard
ID	UC04a
Primary actor	End-user
Description	1. Once pressed the "leaf" button on the action bar the user will be shown a screen containing information about the ongoing trip. The use case starts as soon as the user presses the "Reset" button on the centre's console application screen. It will then zero all the values except the fuel level.
Assumptions	OpenCar working correctly, API is up and running. Car is registered on the platform.
Input	"Reset" button pressed on the screen
Output	Distance, fuel consumption and fuel efficiency values are zeroed
Exceptions	1. OpenCar is malfunctioning 2. Car is not yet registered in any account on the platform 3. User did not press the right button
Priority	High

Title	Save Information on Fuel dashboard to the online platform
ID	UC04b
Primary actor	End-user
Description	1. Once pressed the ”leaf” button on the action bar the user will be shown a screen containing information about the ongoing trip. The use case starts as soon as the user presses the ”Save” button on the centre’s console application screen. It will then send the current fuel consumption and distance, associated with a timestamp, to the online platform.
Assumptions	OpenCar working correctly, API is up and running. Car is registered on the platform.
Input	”Save” button pressed on the screen
Output	Distance, fuel consumption and timestamp are sent to the online platform
Exceptions	1. OpenCar is malfunctioning 2. Car is not yet registered in any account on the platform 3. User did not press the right button
Priority	High

Title	Show directions to nearest petrol station
ID	UC04c
Primary actor	End-user
Description	1. Once pressed the ”leaf” button on the action bar the user will be shown a screen containing information about the ongoing trip. The use case starts as soon as the user presses the ”Go to nearest station” button on the centre’s console application screen. It will then receive the current car’s coordinates and give directions to the nearest petrol station, while showing the corresponding map.
Assumptions	OpenCar working correctly, API is up and running. Car is registered on the platform.
Input	”Go to nearest station” button pressed on the screen
Output	Map is shown on console’s screen, as well as directions to the nearest station
Exceptions	1. OpenCar is malfunctioning 2. Car is not yet registered in any account on the platform 3. User did not press the right button
Priority	High

Title	**Measure 0-100km/h acceleration times**
ID	UC05
Primary actor	OpenCar Application
Description	1. Once pressed the ”tachometer” button on the action bar the user will be shown a screen containing information about current status of the car, similar to the first dashboard. The use case starts as soon as the user follows the application’s instructions just above the action bar. If followed correctly, the application will measure and show the acceleration time in seconds.
Assumptions	OpenCar working correctly, API is up and running. Car is registered on the platform. User follows application’s instructions
Input	”Tachometer” button pressed on the action bar, instructions followed
Output	Acceleration time measurement, shown in seconds.
Exceptions	1. OpenCar is malfunctioning 2. Car is not yet registered in any account on the platform 3. User did not press the right button on the action bar 4. User did not follow instructions correctly
Priority	High

Title	**Save 0-100km/h acceleration data**
ID	UC05a
Primary actor	End-user
Description	1. Once pressed the ”tachometer” button on the action bar the user will be shown a screen containing information about current status of the car, similar to the first dashboard. Once there, the user can follow the application’s instructions and successfully execute a 0-100km/h acceleration. The use case starts as soon as the user executes a successful acceleration measurement and presses the button ”Save”
Assumptions	OpenCar working correctly, API is up and running. Car is registered on the platform. Successful acceleration measurement executed
Input	”Save” button pressed on the console screen
Output	Detailed telematic data sent to the API
Exceptions	1. OpenCar is malfunctioning 2. Car is not yet registered in any account on the platform 3. User did not press the right button on the action bar 4. User did not execute a successful acceleration measurement
Priority	High

Title	**Estimate 0-100km/h acceleration time**
ID	UC05b
Primary actor	End-user
Description	1. Once pressed the ”tachometer” button on the action bar the user will be shown a screen containing information about current status of the car, similar to the first dashboard. Once there, the user can simulate the starting state of an acceleration measurement and press the button ”Estimate”. It will then calculate an estimated time considering all the previous ones and the current car’s conditions.
Assumptions	OpenCar working correctly, API is up and running. Car is registered on the platform. There are already some measurements saved
Input	”Estimate” button pressed on the console screen
Output	Estimated acceleration time
Exceptions	1. OpenCar is malfunctioning 2. Car is not yet registered in any account on the platform 3. User did not press the right button on the action bar 4. User does not have any acceleration measurement saved
Priority	High

Title	**Register max G-Force during acceleration try-outs**
ID	UC05c
Primary actor	OpenCar application
Description	1. Once pressed the ”tachometer” button on the action bar the user will be shown a screen containing information about current status of the car, similar to the first dashboard. Once there, while the user is measuring acceleration times the application will register the strongest G force generated since the application started and show it on screen.
Assumptions	OpenCar working correctly, API is up and running. Car is registered on the platform. User is trying to measure acceleration times
Input	None
Output	Strongest G-Force generated since application start
Exceptions	1. OpenCar is malfunctioning 2. Car is not yet registered in any account on the platform 3. User did not press the right button on the action bar 4. User does not try to measure acceleration times
Priority	High

Title	**Suggest best RPM range to change gears**
ID	UC05c
Primary actor	OpenCar application
Description	1. Once pressed the ”tachometer” button on the action bar the user will be shown a screen containing information about current status of the car, similar to the first dashboard. The use case starts as soon as the user estimates an acceleration value. From that moment on, the gear number will change colour according to the best calculated RPM range to change gear.
Assumptions	OpenCar working correctly, API is up and running. Car is registered on the platform. User estimated the acceleration time at least once
Input	”Estimate” button pressed at least once
Output	Colour change according to best RPM range to change gear
Exceptions	1. OpenCar is malfunctioning 2. Car is not yet registered in any account on the platform 3. User did not press the right button on the action bar 4. User does not try to estimate acceleration times
Priority	High

Title	**Show online data**
ID	UC06
Primary actor	OpenCar application
Description	1. Once pressed the ”cloud” button on the action bar the user will be shown a screen containing information about previous events that the car registered concerning the ABS and TCS systems, as well as a top five best acceleration times ever recorded. Information that is available also on the online platform
Assumptions	OpenCar working correctly, API is up and running. Car is registered on the platform.
Input	None
Output	Online data stored on the database is shown
Exceptions	1. OpenCar is malfunctioning 2. API is down 3. Car is not yet registered in any account on the platform 4. User did not press the right button on the action bar
Priority	High

Title	**Change measurement units of console application**
ID	UC07a
Primary actor	End-user
Description	1. Once pressed the "cog" button on the action bar the user will be shown a screen containing switches to change how the values are shown on the other screens regarding the unit. By switching the sliders the application will convert the values to the selected unit in all other screens.
Assumptions	OpenCar working correctly, API is up and running. Car is registered on the platform.
Input	Sliders change
Output	Convert values to selected unit
Exceptions	1. OpenCar is malfunctioning 2. Car is not yet registered in any account on the platform 3. User did not press the right button on the action bar
Priority	Medium

Title	**Turn on/off data collection**
ID	UC07b
Primary actor	End-user
Description	1. Once pressed the "cog" button on the action bar the user will be shown a screen containing switches to change how the values are shown on the other screens regarding the unit. The first switch turns off the data collection feature described in use-case UC03. This will not only ignore all those events but will also register when the switch was turned either ON or OFF
Assumptions	OpenCar working correctly, API is up and running. Car is registered on the platform.
Input	Data collection status is changed
Output	Data collection event is registered with timestamp associated. Events will be registered or not depending on current status of the switch
Exceptions	1. OpenCar is malfunctioning 2. API is down 3. Car is not yet registered in any account on the platform 4. User did not press the right button on the action bar
Priority	Medium

Appendix G

Online Platform Use-Case Tables

Title	**Serve the API for the OpenCar application**
ID	UC01
Primary actor	System
Description	1. Every time the OpenCar application requires a connection to the API either GET or POST information, the platform should have the routes available for the effect
Assumptions	OpenCar working correctly, API is up and running.
Input	HTTP requests
Output	Data stored/withdrawn
Exceptions	1. OpenCar is malfunctioning 2. API is down
Priority	High

Title	**Register on the platform**
ID	UC02
Primary actor	End-user
Description	1. his use-case starts when the user opens the login page of the application and presses the ”Register” button. The user must then proceed to fill in some personal data about him. Once that is done, the user will receive an email to authenticate his account. Only then is the user able to successfully login on the platform
Assumptions	Server is up and running. Credentials do not exist
Input	Personal data
Output	Account created
Exceptions	1. Server is down
Priority	High

Title	**Login on the platform**
ID	UC03
Primary actor	End-user
Description	1. This use-case starts when the user opens the login page of the application and inserts the credentials set upon the registration on the platform, pressing ”Login” button after. 2. If for some reason the user cannot login, a message will appear with a possible reason why.
Assumptions	Server is up and running. Account created and authenticated
Input	Login Credentials
Output	Access to online platform
Exceptions	1. Server is down 2. Account does not exist 3. Bad credentials 4. Account not authenticated
Priority	High

Title	Add car to the platform
ID	UC04
Primary actor	End-user
Description	1. This use-case starts when the user either logs in for the first time and currently has no cars on the platform or when the user wants to add more by going into ”My Cars” tab. In here the user uploads a picture of the car in question and fills in some basic information to help both the system and the user to identify the car in the future.
Assumptions	Server is up and running. Account created and authenticated. Car not registered yet
Input	Car info
Output	Car registered on the platform
Exceptions	1. Server is down 2. Car already exists in someone else’s account
Priority	High

Title	View latest events/warnings
ID	UC05
Primary actor	End-user
Description	1. This use-case starts when the user logs in the platform and goes into the ”Main” tab on the sidebar. In here he can view details of the latest info that was stored from the current selected car.
Assumptions	Server is up and running. Account created and authenticated.
Input	None
Output	Information about latest events
Exceptions	1. Server is down 2. The car has no events recorded
Priority	Medium

Title	Analyse Acceleration data for saved measurements
ID	UC06
Primary actor	End-user
Description	1. This use-case starts when the user logs in the platform and goes into the ”Acceleration” tab on the sidebar. In here the user can thoroughly analyse details of any stored 0-100km/h acceleration measurement currently stored. The user will also get some suggestions to improve the worst measures based on the best ones ever recorded.
Assumptions	Server is up and running. Account created and authenticated.
Input	Choose measurement
Output	Graphical data, ambient conditions and suggestions
Exceptions	1. Server is down 2. The car has no events recorded
Priority	High

Title	**View full extent of the logs concerning the selected car**
ID	UC07
Primary actor	End-user
Description	1. This use-case starts when the user logs in the platform and goes into the ”Full Logs” tab on the sidebar. In here the user can thoroughly analyse every event ever recorded by the car in question in their ”raw” state.
Assumptions	Server is up and running. Account created and authenticated.
Input	None
Output	All information available about events recorded by the car
Exceptions	1. Server is down 2. The car has no measures stored
Priority	High

Title	**Delete all the logs of a certain event concerning the selected car**
ID	UC07a
Primary actor	End-user
Description	1. This use-case starts when the user logs in the platform and goes into the ”Full Logs” tab on the sidebar. In here the user can also press the ”Delete” button on each box to delete all of its content. The only event that cannot be deleted is the ”Data Collection” log.
Assumptions	Server is up and running. Account created and authenticated.
Input	”Delete” button pressed
Output	All the information about a given event of the selected car is deleted
Exceptions	1. Server is down 2. The car has no data recorded
Priority	Low

Title	**Edit personal information**
ID	UC08
Primary actor	End-user
Description	1. This use-case starts when the user logs in the platform and goes into the ”Profile” tab on the sidebar. In here the user can check the current information on the database concerning him and can also change it if he wants.
Assumptions	Server is up and running. Account created and authenticated.
Input	Data changed, ”Update” button pressed
Output	Personal information successfully edited
Exceptions	1. Server is down
Priority	Low

Title	View calculated statistics
ID	UC09
Primary actor	End-user
Description	1. This use-case starts when the user logs in the platform and goes into the ”Statistics” tab on the sidebar. In here the user can consult useful statistics concerning his driving and the state of the car currently selected. This data is calculated using all the data available from registered events
Assumptions	Server is up and running. Account created and authenticated.
Input	None
Output	Statistical data about the car and the driving experience
Exceptions	1. Server is down 2. The car has no data recorded
Priority	High

Title	View car list
ID	UC10
Primary actor	End-user
Description	1. This use-case starts when the user logs in the platform and goes into the ”My Cars” tab on the sidebar. In here the user can consult all the cars he ever added to the platform
Assumptions	Server is up and running. Account created and authenticated.
Input	None
Output	List of cars
Exceptions	1. Server is down 2. The user has no cars
Priority	High

Title	Switch car
ID	UC11
Primary actor	End-user
Description	1. This use-case starts when the user logs in the platform and goes into the ”My Cars” tab on the sidebar. In here the user can switch between cars to consult data. By changing car, all the data on the other tabs will change to its corresponding car.
Assumptions	Server is up and running. Account created and authenticated.
Input	Car choice
Output	Information re-calculated now regarding the chosen car
Exceptions	1. Server is down 2. The user has only one or no cars
Priority	High

Title	**Delete Car**
ID	UC12
Primary actor	End-user
Description	1. This use-case starts when the user logs in the platform and goes into the ”My Cars” tab on the sidebar. In here the user can delete a given car from his account. This will not only delete the car but every single information about it on the server, mainly logs
Assumptions	Server is up and running. Account created and authenticated.
Input	Car delete button pressed
Output	Car and all its records are deleted from the database
Exceptions	1. Server is down 2. The user has no cars
Priority	High