UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS ESPECÍFICAS DE TELECOMUNICACIÓN

Front-end implementation for an automatized car parking

Autor:

D. Adrián Mazaira Hernández

Tutor:

Dr. D. Juan Carlos Aguado Manzano

D. Daniel Castaño Navarro

VALLADOLID, JULIO 2020

TRABAJO FIN DE GRADO

TÍTULO:	**Front-end implementation for an automatized car parking**
AUTOR:	**D. Adrián Mazaira Hernández**
TUTOR:	**Dr. D. Juan Carlos Aguado Manzano**
	D. Daniel Castaño Navarro
DEPARTAMENTO:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

PRESIDENTE:	**Dr. D. Ignacio de Miguel Jiménez**
VOCAL:	**Dr. D. Ramón Durán Barroso**
SECRETARIO:	**Dr. D. Juan Carlos Aguado Manzano**
SUPLENTE:	**Dr. Dª Noemí Merayo Álvarez**
SUPLENTE:	**Dr. D. Juan Blas Prieto**

FECHA:
CALIFICACIÓN:

ABSTRACT

Cities are undergoing a change in mobility due to factors such as climate change, high urban air pollution, the economic crisis, high population density in urban centres, or the COVID-19 crisis. Some of the proposals that seek to provide a solution to this new mobility paradigm are car sharing platforms or electric vehicles. This Final Degree Project combines these two concepts together with the autonomous vehicle, presenting an app for an electric vehicle carsharing service based on autonomous parking. This application has been developed in Flutter, Google's framework for creating multi-platform applications, so that the app can be released in the two most important mobile operating systems of the moment, Android, and iOS. In addition, the style guides of Material design have been followed as well as the design trends of other similar car sharing services apps. The app allows users to access the car sharing service, allowing users to find the nearest parking lot and obtain a vehicle from it, and also serves as a tool for service administrators, so that it allows them to maintain, develop and improve the system more easily.

Keywords:

carsharing, app, Flutter, mobility, autonomous car, connected car.

RESUMEN

Las ciudades están sufriendo un cambio en la movilidad debido a factores como el cambio climático, la elevada contaminación del aire de las ciudades, la crisis económica, la alta densidad de población en núcleos urbanos, o la crisis del COVID-19. Algunas de las propuestas que buscan aportar una solución a este nuevo paradigma de movilidad son las plataformas de carsharing o los vehículos eléctricos. Este Trabajo de Fin de Grado aúna estos dos conceptos junto con el de vehículo autónomo, presentando una app para un servicio de carsharing de vehículos eléctricos basado en aparcamiento autónomo. Esta aplicación ha sido desarrollada en Flutter, el framework de Google para crear aplicaciones multiplataforma, de forma que la aplicación pueda ser lanzada para los dos sistemas operativos móviles más importantes del momento, Android e iOS. Además, se han seguido las guías de estilo de Material Design asi como las tendencias de otras aplicaciones similares de servicios de carsharing. La app permite, por una parte acceder a los usuarios al servicio de carsharing, permitiendo a los usuarios encontrar el parking más cercano y obtener un vehículo de este, y por otra parte, sirve como herramienta para los administradores del servicio, de forma que permite mantener, desarrollar y mejorar el sistema de manera más fácil.

Palabras clave:

carsharing, app, Flutter, movilidad, vehículo autónomo, vehículo conectado.

ACKNOWLEDGMENT

I would like to thank all the TwizyLine team for all the effort and passion they all have put into this project. From our tutor Juan Carlos who has always encouraged and helped us to develop the project and has taught us even from the Word comments, to our coaches from Renault, Daniel and Jairo, for the support and advice, and to my partners Ignacio, Mario and Samuel for the countless hours we have spent developing something that goes further than just a Final Degree Project. Thanks also to my life partner Lucía for always pushing me forward, and thanks to my family for all the support and education they have provided me.

Table of contents

Table of figures

1. INTRODUCTION

1.1 Motivation

The world is facing a period of far-reaching changes and tough challenges. The number of current issues that must be solved is growing, starting with the climate change, going through high population urban problems and ending with the economic crisis. Urban mobility is central to these problems, accounting for example for 40% of all CO2 emissions of road transport and up to 70% of other pollutants from transport [1]. In this context, Renault and Segula have launched the Twizy Contest initiative, a competition addressed to students around the world. In words of S. Schmidtlin and V. Fournier: “The mobility of tomorrow is undergoing fundamental change. People, goods, urban services, and mobility objects will be transformed! To better anticipate these evolutions, Renault is taking the initiative to create an experimental challenge in creativity: Twizy Contest. (…) We hope that this initiative will see emerge promising ideas (…)” [2].

This project arises as a solution for the challenge proposed by the Twizy Contest 2020, a sustainable mobility solution on the Paris 2024 Olympic Games. The proposed solution should be innovative, frugal and durable, preferably using Twizy as part of it. The organization only gives these few indications to develop the project, so more context should be given to understand the real scope of the project.

As stated above, this challenge comes from the fact that from the past few years, a change in mobility is happening. There are some reasons that explain the change trigger, which should be considered to develop the project. In the following lines there is a detailed description for some of these changes:

- **Climate change consciousness.** People is getting more environmental awareness. In 2018 the Pew Research Center’s Spring 2018 Global Attitudes Survey published that 67% of people interviewed in 26 different countries considered global warming the most serious problem the planet was facing. This consciousness has suffered a significative increase since 2013 [3]. Movements like Greta Thunberg’s “Fridays for Future” put again the environment debate on the focus, after the continuous failures of Kyoto protocol [4], Paris Agreements [5] or Climate meets [6].
- **Urban pollution.** Cities’ air quality has become a big trouble for the biggest Europe cities like Madrid or Paris, but it is also becoming a more common problem for smaller cities like Valladolid. According to the World Health Organization (WHO) there are 4.2 million deaths every year as a result of exposure to ambient (outdoor) air pollution, and 91% of the world’s population lives in places where air quality exceeds WHO guideline limits [7]. Cities often suffer from “high pollution episodes”. When this happens, traffic has to be hardly restricted. In addition, Europe fines those cities that exceed the European Union air quality limits.
- **Cities traffic restrictions.** This pollution problems, in addition to the traffic jams and parking troubles derived from the massive personal vehicle use, led the city halls to implement traffic restrictions. Those restrictions force citizens to either buy less pollutant vehicles (not affordable for everyone) or either adopt more ecologically responsible mobility models. Those can be public transport, carsharing platforms, bikes, or scooters.

- **The Electric Vehicle Rises.** These changes have driven to the growth of the electric vehicle. This kind of vehicle has some advantages. They are quiet, nicer to drive (the par response is cleaner than in combustion cars), and their mechanics are simpler. But above all, they do not emit polluting gases. This has led city halls to potentiate those vehicles. City halls have undertaken some incentives such as free parking in pay parking zones, or access to areas restricted to polluting vehicles.
- **Economic crisis.** The global crisis started in 2008, harming family's economy. This crisis brought out the so-called collaborative economy. This new economy encouraged the access to shared goods and services without the traditional need of owning them. In automotive, this led to a change in the minds of drivers, who now consider the costs of renting their vehicles instead of owning them. The crisis was also a breeding ground for carpooling platforms like *BlaBlaCar* or *Amovens* [8]. The global rising on fuel prices also encouraged people to find more economic means of transport.
- **COVID-19**, which has strongly hit Europe and the entire world. This pandemic will for sure change mobility and sharing platforms. The public transport, and overall the metro has been one of the most outbreak points. This could lead some public transport users move to carsharing services, either for fear or public transport capacity restrictions. Sharing companies will also have to change, to avoid future contagions, shared vehicles will need to implement disinfection measures.

Paris, the scenario where the challenge of the contest unfolds, also faces these challenges, and the city hall has invested a lot in reducing traffic and pollution. In 2015 Paris topped the world lists for air pollution [9], after which several measures have been taken.

To solve this problem, first of all, the city hall increased the net of bike paths [10], which is a reinforcement of the Paris bike sharing program, *Velib*, operational since 2007 [11]. No doubt, this has contributed to designating Paris as the 8th most bike-friendly city in the world in 2019, according to Copenhagenize Index [12], clearly improving its previous place, the 17th by 2015.

Secondly, Paris has increased its green zones and pedestrian sidewalks. Some parts of the city center have been pedestrianized. One of the most notorious projects was pedestrianizing the lower quays of the river Seine [13].

Finally, Anne Hidalgo (current mayor) has declared an "anti-cars war". From 2016 most contaminant vehicles cannot drive through Paris center. To identify these vehicles, which includes diesel vehicles before 2006 and gasoline vehicles before 1997, a classification system called Crit'Air has been implemented [14]. Drivers must put on their cars a different sticker depending on how much their car pollutes. The major wants to ban diesel vehicles in Paris by 2020 [15], whereas France goal is to ban the sales of petrol and diesel cars by 2040 [16]. These bans on pollutant vehicles are joined by non-pollutant vehicles bonifications, like free parking, access to the Low Emission Zone [17] or more than 800 public charging stations according to chargemap.com [18].

Also, in 2016 a car free policy called "Paris respire" started to work. Paris respire frees from cars some parts of the city at certain times of the week. For instance, the Champs Elysées are car free on the first Sunday of every month [19].

Paris is an evidence that the mobility model is changing. By 2016 Paris homes owning one or more cars were about 35.2% [20]. Harris interactive survey on European cities on August 2019 reveals us some keys. About 54% of Parisians admit they use their car less than once a week.

Half of car owners said to be ready to get rid of their cars. Public transport is highly valued by Parisians, and they also value taxis, bikes, or scooters to get rid of their cars. 79% of Parisians have a positive opinion of car sharing, 74% think that car sharing could help improve travel in their city and 78% would like to have more car sharing services in their own neighborhood [21].

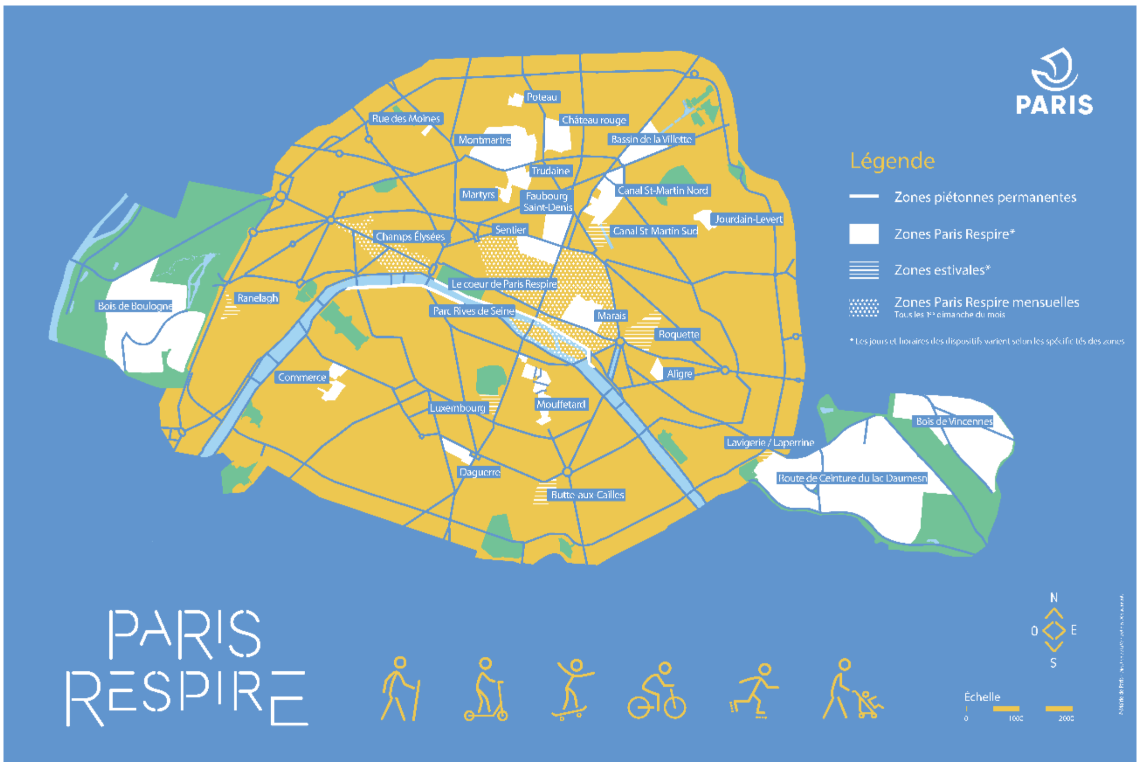


Fig. 1 Paris Respire map [19]

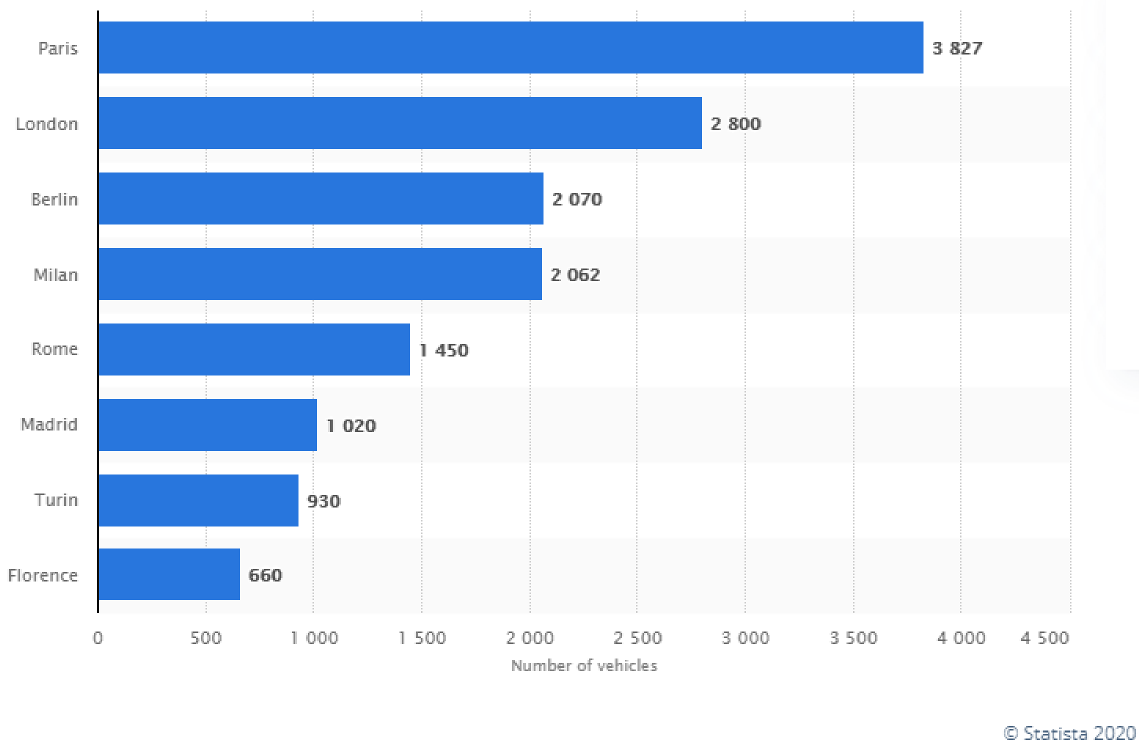


Fig. 2 Number of carsharing vehicles in selected European cities in 2016 [22]

All these features that characterized Paris as a city where the mobility change is happening are part of the solution that we propose for the Twizy Contest. As mention above, the contest asks for a mobility solution for Olympic Games in Paris, and our proposal is the TwizyLine project. TwizyLine arises from three main ideas: Autonomous driving, electric cars, and carsharing platforms, which are central to what Paris currently represent. TwizyLine works as a carsharing platform based on autonomous parking for electric cars. Giving and efficient and technology leading solution to mobility in cities.

TwizyLine key is its autonomous parking shown in Fig. 4. Users leave the vehicle in the leaving zone. Then they get out the vehicle and abandon the infrastructure trough turnstiles. Once the leaving zone is free of people, a server assigns the vehicle a spot regarding its battery level. Once the vehicle knows its destiny it will use the parking items to guide itself. Those items are magnetic bands for line tracking guidance, and RFID tags for decision making at the bifurcations of the line. The vehicle implements all the necessary modifications to drive itself, follow the line, read the RFID tags plus proximity sensor to avoid collisions. The vehicle informs the server of every step it takes inside the parking, but when it is outside, it is still connected to it and periodically sends its location, battery level, or error log. For users to get access to the carsharing service there is an app where they can find the closest TwizyLine parking and request a vehicle.

Fig. 3 Logotype and symbol of TwizyLine

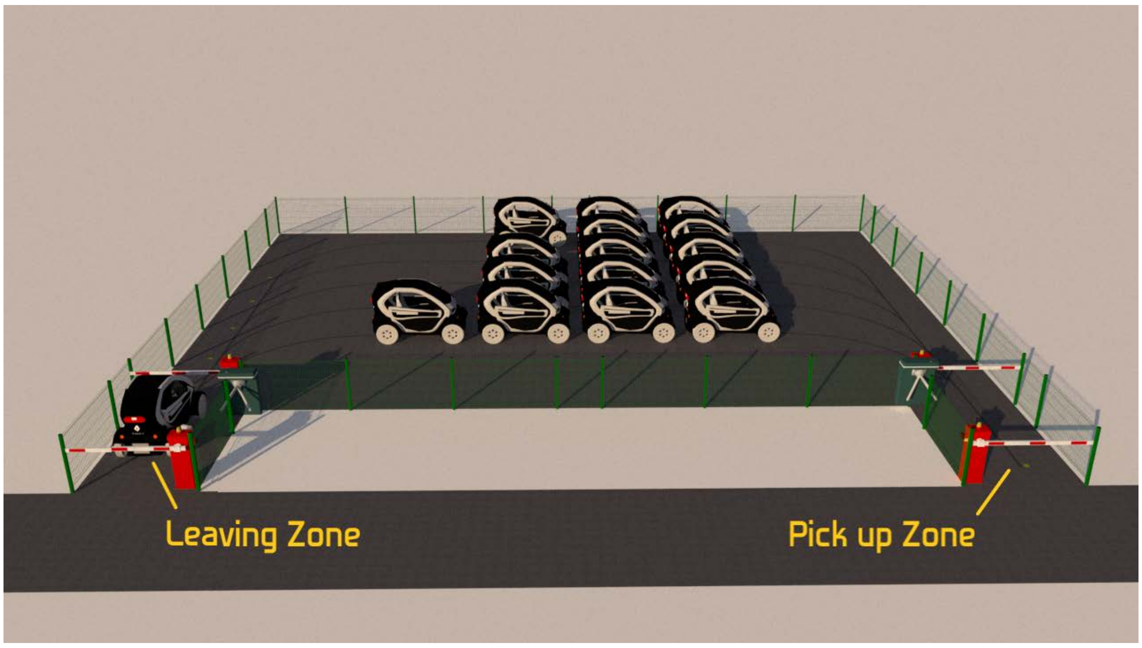


Fig. 4 TwizyLine parking 3D representation

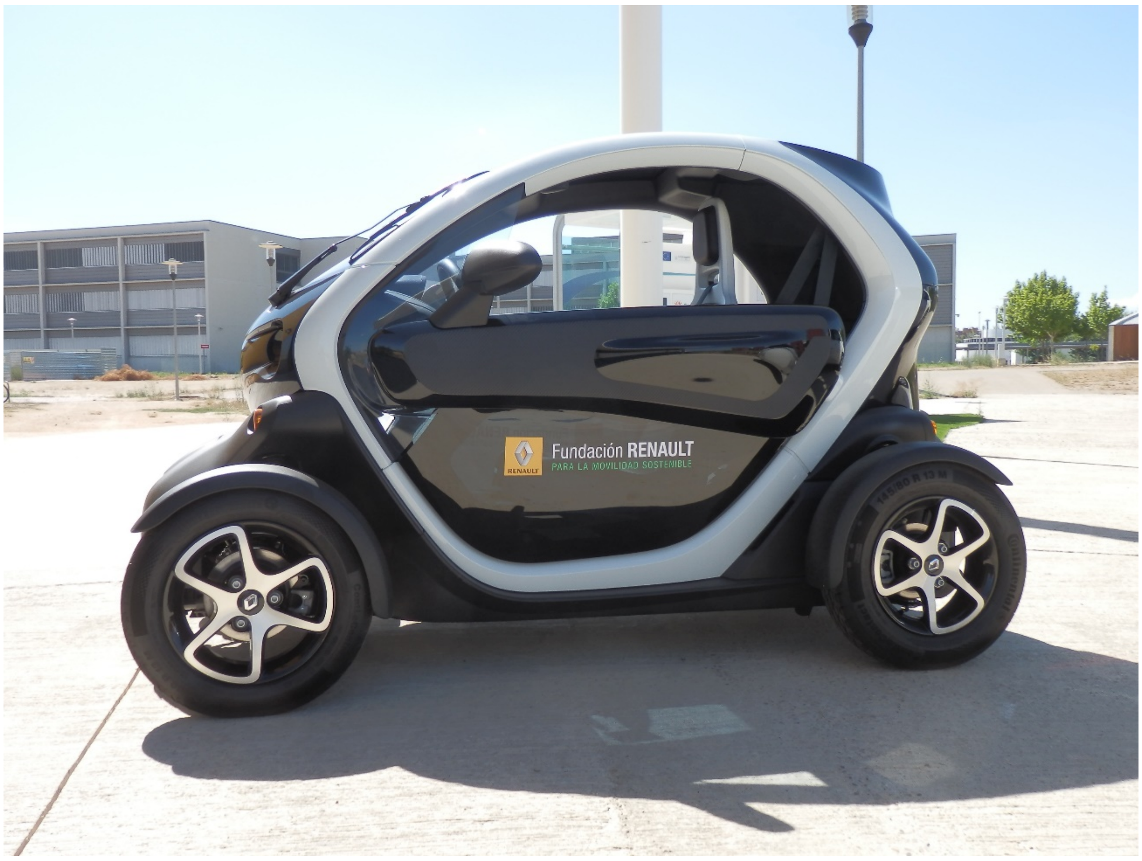


Fig. 5 Renault Twizy

TwizyLine fits in with the recent approach of Paris authorities to the posed challenges, besides, looking at our project as a carsharing service, it has some advantages against other equivalent services:

- Regarding climate change, urban pollution, and cities traffic restrictions TwizyLine takes advantage of the Renault Twizy (Fig. 5). Renault Twizy is a full electric vehicle. This means it does not emit pollutant emissions and it is not affected by traffic restrictions to pollutant vehicles. It is a small vehicle, perfect for urban displacements, and this features also makes TwizyLine parking lots able to host more vehicles in less space. Additionally, users do not access the parking, so the spots have no space to let people get in or out the Twizy. And there is no need for bigger spots for disabled people either as they would get in or out the car at the leaving zone or pick up zone respectively as any other user.
- Regarding the context, Paris and its Olympic Games, as Fig. 2 shows, by 2016 Paris was the European city with the higher number of carsharing vehicles. Carsharing future in Paris looks promising. About 20% of Parisians already use a carsharing service [23]. And TwizyLine could be a perfect solution for big events. Moreover, if we talk about Olympic Games, an event that brought 11551 athletes and 1.17 million tourists to Rio during their games in 2016 [24, 25]
- Regarding economic crisis and COVID-19. TwizyLine is based on parking zones, enclosed places where only the Twizys can enter (this is also a plus for legal restrictions to autonomous vehicles). It would not be too hard to install nebulizers at the parking entrance. So once the users are out of the vehicle and it has entered the parking it gets disinfected. Once the vehicle gets inside the parking pedestrians cannot access to them,

so Twizys would not be contaminated. Also, the Renault Twizy because of its reduced size and its absence of windows makes it easier to be disinfected. The 2008 crisis brought out the collaborative economy, and the after-COVID19 crisis is expected to be even worse. So, these sharing platforms would likely increase.

TwizyLine team is formed by Ignacio Royuela, Samuel Pilar, and myself, the three of us are engineering students and Mario Martin, journalism student. It is supervised by Juan Carlos Aguado, teacher at University of Valladolid, Daniel Castaño, and Jairo Gurdiel, coaches from Renault.

To achieve the objectives of such a complex project, TwizyLine has been divided into five different parts, four of which have been developed in the final degree project reports of the members of the team. These five parts are:

- Front-end implementation for TwizyLine service: It is described in this report and it consists of a hybrid smartphone app. This application allows users to access the service. At the same time, it provides the service administrators with a powerful tool to manage and keep control of the system.
- The control and communications modules installed on a Renault Twizy. Which consists of two Humming Boards configured to control the Twizy and communicate with the server described in [26] These modules connections include GPS, GPRS, CAN, RFID, Magnetic and Proximity sensors. This information can be found at the reference [27].
- The back-end implementation, which consists on a server developed using python and MySQL, implemented on a Raspberry Pi 3. This server hosts the MQTT broker and manages the communication with the app and the vehicles and between them. It is also in charge of guiding the vehicles through the parking and manage their battery status or faults. This information can be found at reference [26]
- The market research and communication strategy, as well as some law analysis and brand image can be found at reference [28].
- Vehicle modifications and systems integration. Once every previous part is done, we will have to physically adapt the vehicle, installing the necessary elements and putting all together to work. Plus, all the testing and validation. This will be reflected in reference [29].

As seen in this division, TwizyLine project covers a wide range of innovative technologies. From autonomous vehicle to connected vehicle, Vehicle To Infrastructure (V2I) or hybrid smartphone apps.

Every member of the team works to win Twizy Contest 2020 in the short term. But TwizyLine project aims much further. TwizyLine aims to be a revolution in carsharing and help to solve the problems of the XXI century cities. We have defined some objectives to get to this goal. In the next section the specific objectives for this Final Degree Project are detailed.

1.2 Objectives

The main objective of TwizyLine is to provide a clean and effective mean of transport for the cities. This will be accomplished with our autonomous parking based carsharing system. As specified in section 1.1, this final degree project focuses on developing a Front-End for the system. As a result, the main objective of this work is to provide users access to the service, allowing them to locate the nearest TwizyLine stations and get a vehicle from them.

Other secondary objectives are:

- To provide the service operators a valuable tool to manage the system.
- To provide a hybrid application, available for the two main operative systems, Android and IOs.
- To create a simple and easy to use application for both, users and administrators.
- To create a communication via between service providers and users. On the one hand via the app stores reviews, and on the other hand via the website accessible through the app.

1.3 Steps and methods

The steps followed to carry out this project were mainly the next ones:

Analysis and study stage: Market research to know the main carsharing services apps and their characteristics. The different options to build the app were also studied.

Defining the project structure stage: After the study stage, the project was defined considering the information collected earlier. This stage included:

- Establishing the app functionalities and dividing the app into different screens for each functionality.
- Establishing the separation between users and admins.
- Designing the app structure and every screen layout.
- Deciding the language to work with, using different criteria like: budget, expected performance, kind of app, layout complexity, time or language experience.
- Defining the protocols and message format to communicate with server.
- Establishing the difference between appearance code and functionality code and creating the project files.

Developing process stage: This process was made ensuring best functionality, appealing design, and the best user experience. This stage included:

- Connecting the app to the server and creating the functions to exchange the information.
- Building every screen of the app following the design of the previous stage.

Validating app stage: the app functionalities were validated, implementing error handling and aesthetic modifications to fill the deficiencies found in validation.

Integration and testing stage: As part of a bigger project, this application had to be integrated with the rest of it. The integration and the testing of the whole project included:

- Group integration to ensure perfect understanding between app, server, and vehicle.
- Tests on both physical and virtual devices.
- Releasing to beta tester group for external testing.
- Fixing errors and any founding deficiencies.

1.4 Available resources

The code was developed using both IDEs Android Studio and Visual Studio Code. All the progress was tracked using GitHub repository. Also, a Raspberry Pi 3 model B+ has been used as a server.

The testing was carried out on different devices shown in Table 1

Google Pixel 2 (Android Virtual Device)	OS*: Android 8.0 Screen size: 5” Aspect ratio: 16:9 Resolution: 1080x1920: 420dpi**
BQ Aquaris A4.5 (physical device)	OS: Android 7.0 Screen size: 4.5” Aspect ratio: 16:9 Resolution: 540x960: 240dpi
BQ Aquaris E5 4G (physical device)	OS: Android 6.0.1 Screen size: 5” Aspect ratio: 16:9 Resolution: 1280x720: 440dpi
Huawei P8 Lite 2017 (physical device)	OS: Android 8.0 Screen size: 5.2 “ Aspect ratio: 16:9 Resolution: 1920x1080: 424dpi
Xiaomi Mi A3 (physical device)	OS: Android 10.0 – Android One Screen size: 6.088” with waterdrop notch Aspect ratio: 19.5:5 Resolution: 1560 x 720: 282dpi
Huawei P smart 2018 (physical device)	OS: Android 9.0 Screen size: 5.7” Aspect ratio: 18:9 Resolution: 2160x1080: 427 dpi

Table 1 Devices used during the testing process

*OS: Operative System

**dpi: Density on Pixels per Inch

1.5 Structure of the memory

This document is a technical report about the development of an app for the TwizyLine carsharing service. The document is divided in five chapters: Introduction, State of the Art, Implementation, Use cases and Conclusions.

The document starts with the introduction chapter which aims to give the reader the context of the project and the motivations that drove the members of the team to create a carsharing platform based on an autonomous parking.

The document continues with the state of the art whose objective is to provide an overview about the current available technologies to build mobile applications. The advantages and disadvantages of each technology together with a market research that shows the trends in mobility apps, explain the reasons to choose Flutter framework and the design decisions taken.

The third chapter presents a description of the app designed layouts and the implemented functionalities. It is also given a detailed explanation of the operations of the app.

In the fourth chapter, it is addressed the test and integration process. Several use cases are defined and tested together with the rest of parts developed in the whole TwizyLine project to check that everything is working as intended.

Finally, some conclusions and future lines are presented.

2. STATE OF THE ART

2.1 Introduction

In chapter 1 it is introduced the change in mobility that is happening. This change includes carpooling, carsharing, bike sharing, motorbike sharing, or even electric scooter sharing services. But this change cannot be understood without the epicenter of our current lives: The smartphone offers us the possibility of accessing those services at any time anywhere. Most of the tasks that used to be done using a computer are now accessible through our smartphones. Fig. 6 shows this trend, from 2016 there are more mobile users that desktop users across the world. In Spain, for instance, since 2006 there are more phone lines than habitants [30]. Nowadays smartphones have become an essential device, but the smartphone expansion cannot be understood without the expansion of the apps.

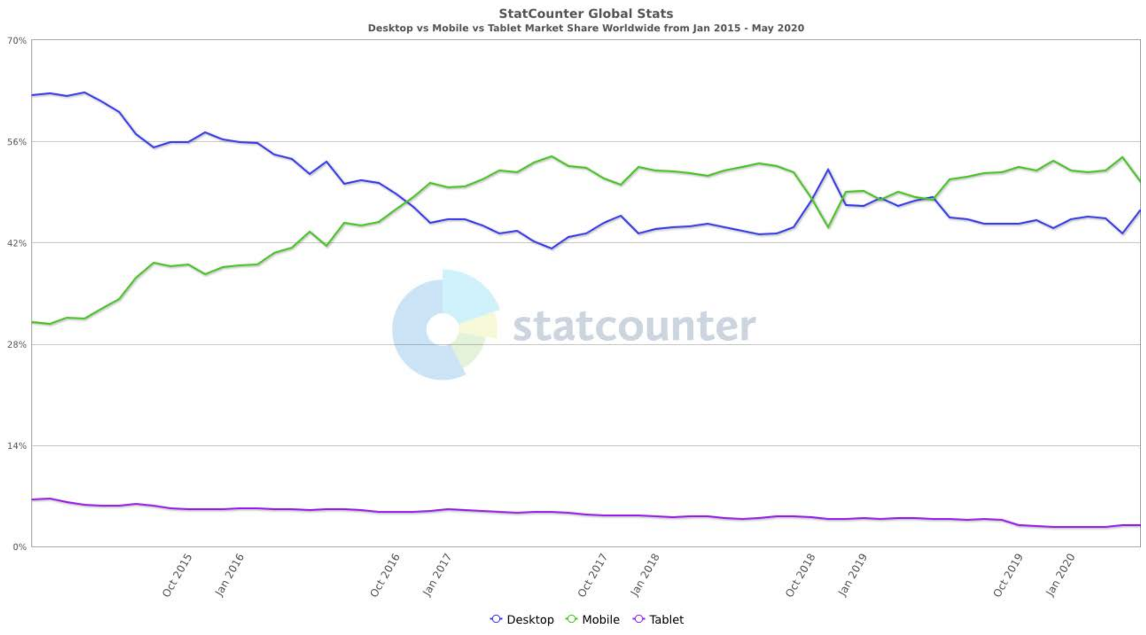


Fig. 6 Desktop vs Mobile vs Tablet Share Worldwide Jan 2015 – May 2020 [31]

Carsharing services need from these apps, their users must be able to find their vehicles or parking lots, access the vehicles, book, or lock cars, and let the service know when they are done with the vehicle. The smartphone is the tool that offers them all these functions in one single device. ZITY [32], Ubeeqo (formerly called Bluemove in Spain) [33], SHARENOW (formerly called Car2Go) [34], Free2Move [35], or getaround (formerly called drivy) [36] are examples of carsharing services in Paris with their own app. Some of these apps are analyzed in section 2.7.

This chapter presents the importance of apps and the development of those. First, it is explained the expansion of apps and how the market has changed. The market is also analyzed to discover the current trends that drove to take the decision of how to build the app described in this work. It is made a classification and comparison of different kind of apps and different ways to build them. After, it is detailed some aspects of Flutter, the chosen framework to build the app, and it is explained the reasons of this decision. And finally, it is presented a research about similar apps and how those apps inspired this work.

2.2 The expansion of apps

The exponential growth of smartphones and wide band connections went along with the expansion of apps. There are more than 3.9 million apps published in the main markets as shown in Fig. 7. Moreover, the use of apps keeps increasing, as it can deduced form Fig. 8, which shows the app downloads from 2016 to 2020. These apps have a huge number of different uses, as it is shows Fig. 9, which speaks of the many different uses we give to our smartphones. Indeed, there are apps for a wide variety of applications, from games to work.

Even further, mobile apps have replaced websites. According to Flurry 2014 report [37], mobile users spent 86% of the time on apps, and 14% on web browses. There are some keys that explain this apps zenith [38] [39] [40] [41]:

- **Speed:** Mobile apps are usually faster than websites. One of the reasons is that websites need to fetch the data from the server, but apps store their data locally. Another reason is the different performance speed between JavaScript (websites) and app frameworks.
- **Personalization:** Apps can let users define their preferences. And they also offer content based on their users' interests, location, or usage behaviour.
- **Online and offline:** The apps ability to work offline is the biggest different between apps and websites.
- **Notifications:** There are two types of notifications: In-app notifications; which are displayed only while the user is using the app, and Push notifications; these notifications can be shown at any time regardless of what the user is doing on its device. Push notifications are a powerful tool that has to be used carefully to avoid deteriorating the user experience.
- **Device features:** Mobile apps can easier access to device features like GPS, camera, NFC, etc. Even browsers can also access some of these features, apps can access to a wider range, and websites are constrained by the browser features.
- **Design and user experience:** While websites rely on the browser features apps are not constrained by browser elements like the address bar. Apps can also implement more advanced gestures like tap, swipe, drag, hold… Improving the user experience.
- **Brand Presence:** Apps increase users brand loyalty and help to influence users' perception just by being present in the user's device. The app icon acts as mini advertisement for the brand, even when the user is not actively using the app.

Further in this document it is detailed the current differences between websites and apps as well as the different kinds of apps. It is also compared the advantages and disadvantages of each one and how they influenced on the decision of how to build the app. Even though our app needs internet connection to work, considering the features here explained and others that are explained further it was decided to build an app. This decision would allow to take advance of all the apps features, and a website would be developed too. The app, which needs to be installed would be necessary to get access to the service, and the website would be used to share information about the service available for anyone. Information like how does the system work or how to start using it. The website would mainly be used to get new user to the service. The app would use the website too, launching the website to give the users this kind of information inside the app if necessary. This way it would not be necessary to duplicate the information in both the app and the website.

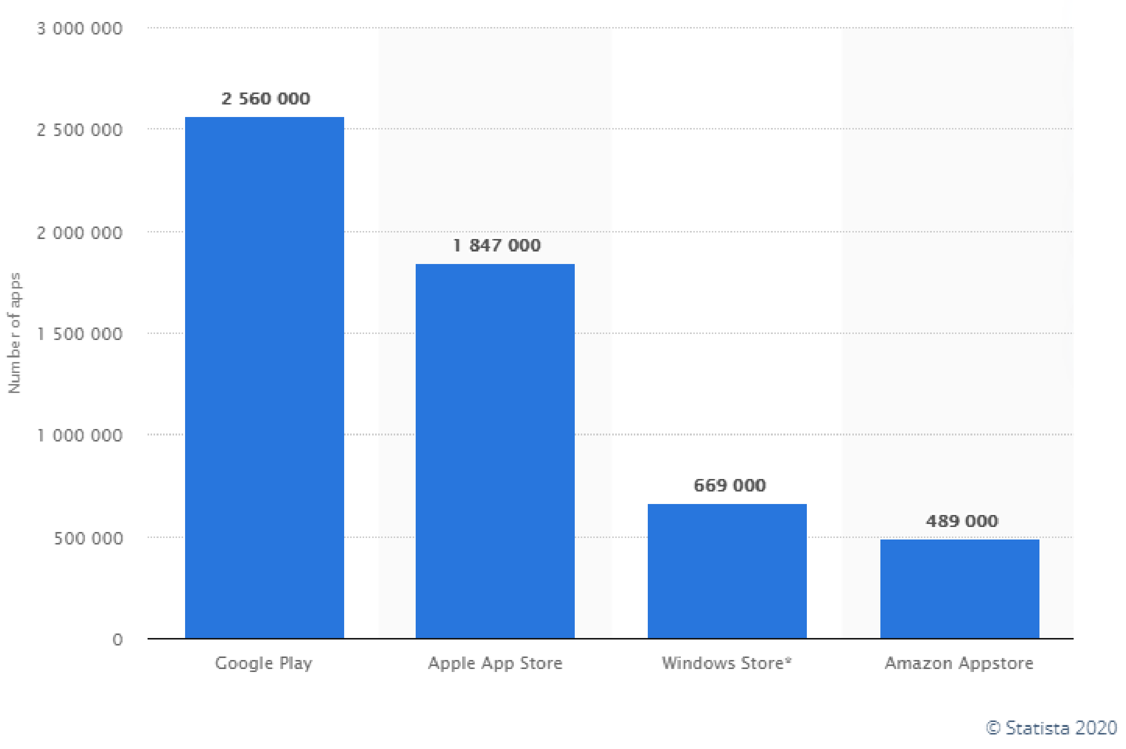


Fig. 7 Number of apps available in leading app stores as of 1st quarter 2020 [42]

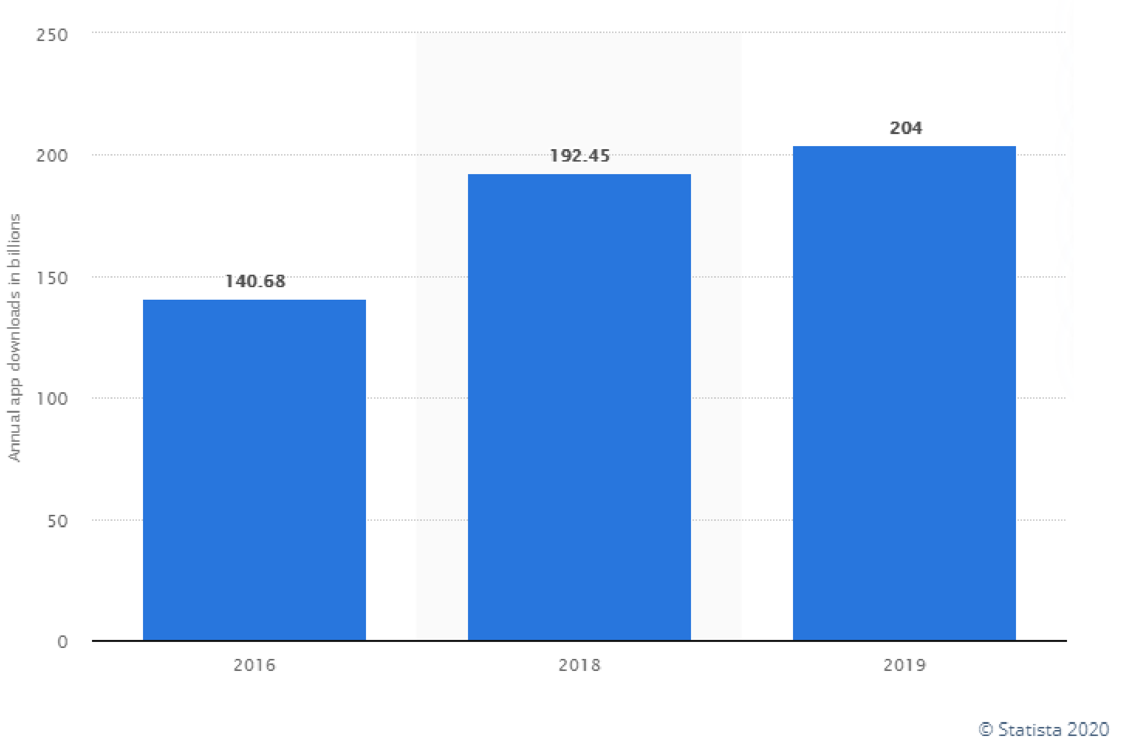


Fig. 8 Number of mobile app downloads worldwide from 2016 to 2019(in billions) [43]

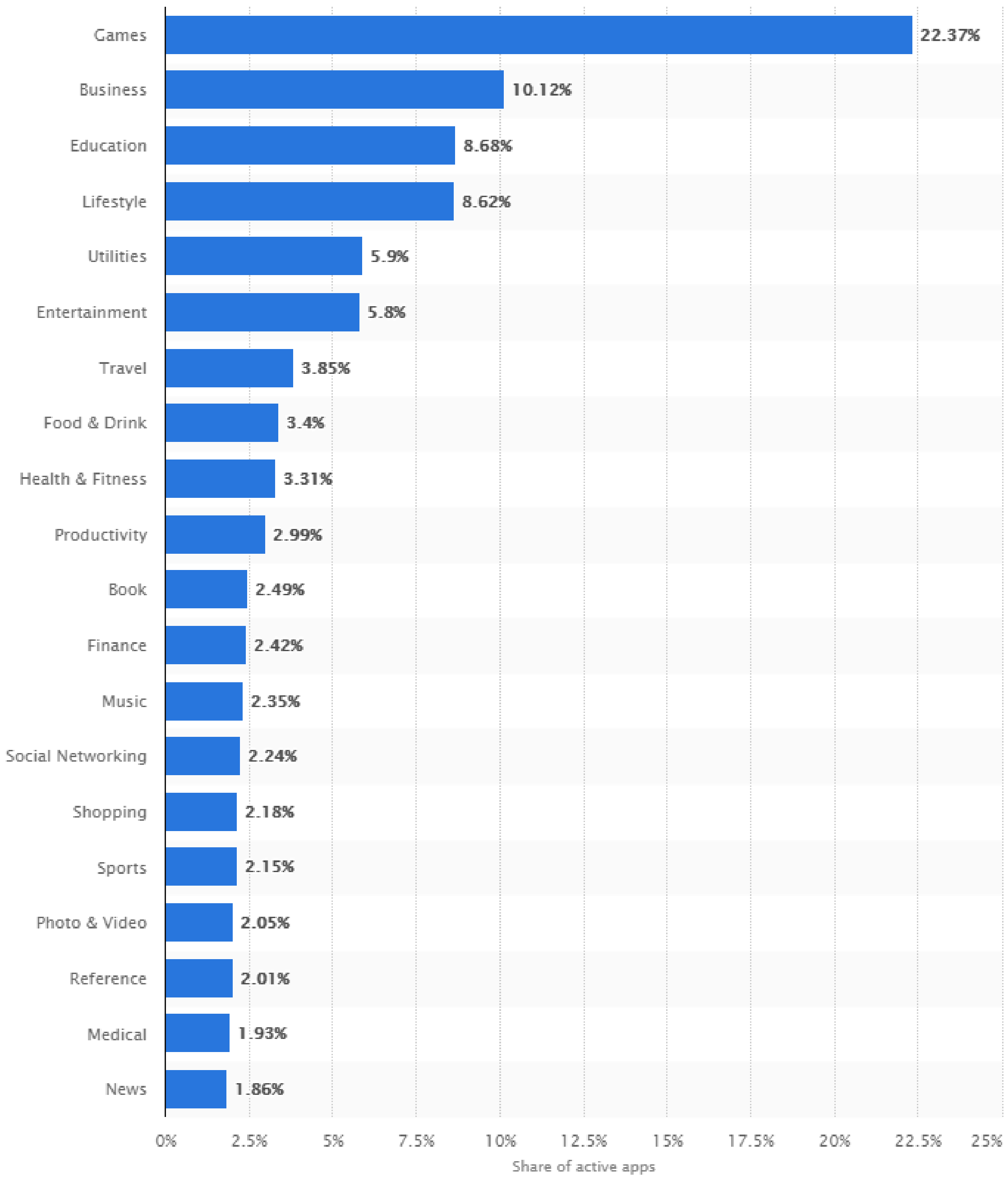


Fig. 9 Most popular Apple App Store categories in November 2019, by share of available apps @Statista [44]

2.3 Android vs iOS

When programming a new app in a smartphone, there are certain aspects of the target smartphone that must be considered. These aspects include size of the screen, resolution, connection capabilities [45], but perhaps one of the most important is the operative system of the mobile [46]. However, the market has changed significantly regarding phones operative systems (OS). Symbian OS was the first modern smartphone operative system. In 2009, Symbian had half of the market share, and nowadays we could consider it to be disappeared, as it is shown in Fig. 10.

In Fig. 10, we can observe how Android is now the main worldwide smartphone OS, while iOS is almost its only competitor. Although this rivalry is not symmetric in the entire world. As we can see in Fig. 11 and Fig. 12, while Android has around 70% of the market share, in North America the rivalry is closer, about 50% each (In North America Apple sells more than 50% of market smartphones and tablets) [47].

This data shows that Android and iOS are the two main OS, and how their rivalry differs depending on location. This information plus the Twizy Contest context are crucial to decide which kind of app to build. Twizy contest put up the challenge in the Olympic Games of Paris 2024, an international event with visitors from all over the globe. The next point will define the different kinds of apps an its features. The context given in this point will help to understand the decision taken.

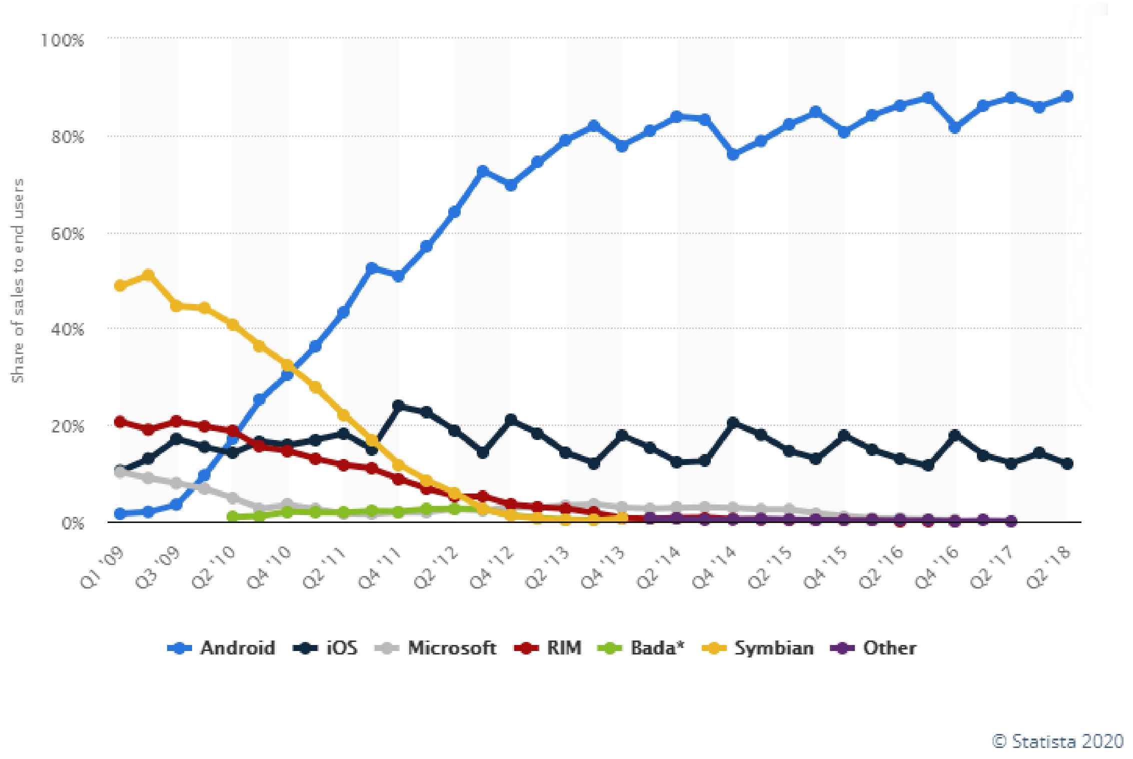


Fig. 10 Global mobile OS market share in sales to end users from 1st quarter 2009 to 2nd quarter 2018 [48]

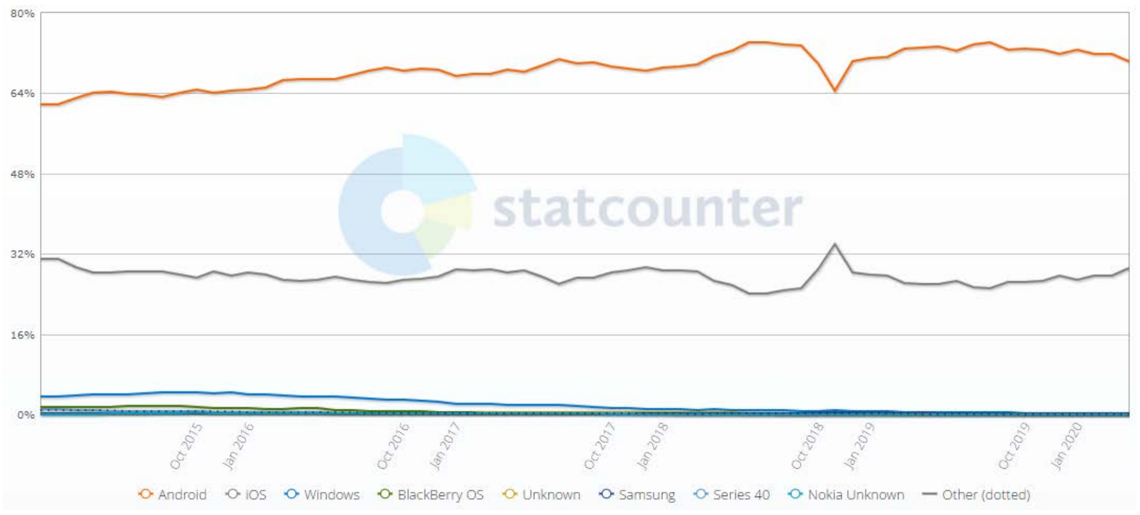


Fig. 11 Mobile Operating System Market Share Europe Jan 2015 - Apr 2020 [49]

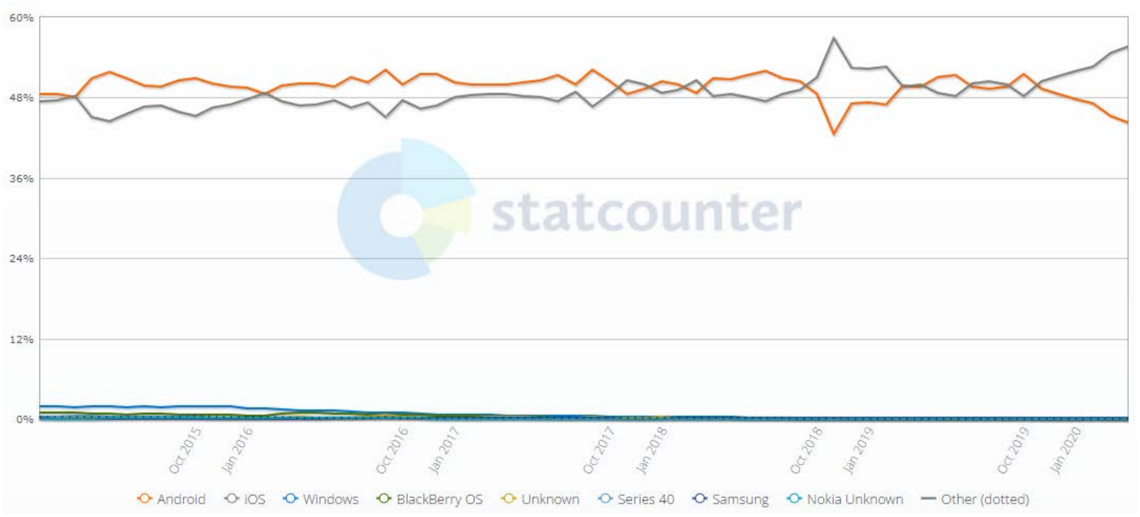


Fig. 12 Mobile & Tablet Vendor Market Share North America Jan 2015 - Apr 2020 [50]

2.4 App Development

Regarding app development, there are some key questions to be answered: The target OS (solved in the previous section), the available budget, the expected performance, or the kind of application. This section will address those questions and how the answers drove to the decision of how to build the app.

It can be found four different kind of apps: Native Apps, Web Apps, Generated Apps and Hybrid Apps [51], [52], [53]. They are all explained below.

2.4.1 Native Apps

Native Apps are developed in a classic programming language, as for example Java/kotlin for Android and Objetive-C/Swift for iOS. Native apps offer the best performance, appearance, and safety [51]. Therefore, they are highly recommended for games and financial apps. Nevertheless, they are also the most expensive option as the code is only useful for the platform for which is developed. This means to make a native app for Android and iOS it is necessary to make two different apps [54].

2.4.2 Web Apps

Web Apps are a trend from the past few years [55]. They can be divided at the same time into two different options, Web Apps and Responsive design webs [56].

Web Apps are run inside a web browser tab. They do not require any installation but need internet connection. The app is allocated in the cloud and users access to it. This has the advantage that any update in the software will arrive to all users at the same time. There is no need to download a new version of the app like it happens with Native apps. This can be useful when finding security bugs, for example [56]. They are written in web languages (HTML, CSS, JavaScript) which are wide known. It makes these apps easier and faster to develop. The disadvantages are that the performance is lower than the native apps, and they cannot access to all the device functionalities or local storage [57].

Progressive web apps (PWA) are a sub kind of web apps [58]. There is not a clear definition of what a PWA is. But there are 3 features any PWA must have [59]:

- Secure contexts (HTTPS)
- One or more Service Workers
- A manifest file

PWA are designed to be installable, capable and reliable. They try to offer the best of web apps and native apps (see Fig. 13). And they are responsive to fit any mobile device, tablet, or desktop. This apps can be accessed through the web browser, but they can be installed too. This way the app will be treated as a native app by the OS. It can have a launcher icon on the desktop (not just a shortcut) an even can appear in the applications menu list. This also lets the app run in its own window instead of a browser tab. There is no need of URL bar or browser interface. This changes the user perception of the app as it looks the same way a native app looks like. It enables more customization, like loading page or the desktop icon. Having a launcher icon on the desktop also increases the user's loyalty to the site. In addition, they can be installed from the browser without the need of an app store. [60] [61]

PWA apps are faster than common web apps, their performance is closer to native apps. Additionally, they are connectivity independent. This is because the service worker manages asset catching and can serve a cached version when the connection is bad or non-existent. And this can also increase the performance working offline even when the connection is good. [62]

They are also re-engageable**.** Accessing the re-engagement user interface (UI) of the OS like Push Notifications. This apps can access more hardware features of the device [63].

They should also be discoverable for search engines, safe and linkable. And like common web apps they have the advantage of being always up to date as the updates take place in the server instead on every user app version. [64]

These apps have been there for a while, but they were way slower than native apps [63]. This is changing with the new browser technologies. The current major browsers are now compatible with this technology [64], [65].

PWA are an option for developers to be considered in the immediate future as they are getting more and more popular [66]. Twitter is a good example of a PWA [67]

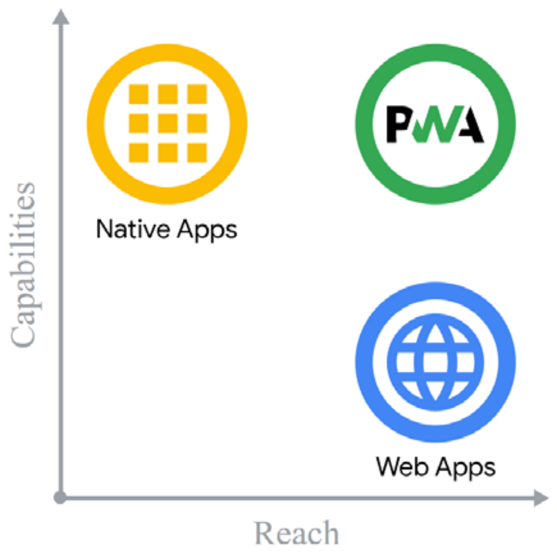


Fig. 13 Capabilities vs. reach of native apps, web app, and progressive web apps. [58]

Responsive webs are common webs configured and designed to fit different screen sizes and resolutions. The content comprises, expands, or reorganizes itself to be visible on any device and improve the user experience [68]. These are simple because they are just a website and can be viewed on any device with a web browser and internet connection. Fig. 14 shows a clarifying example of a responsive web design [69].

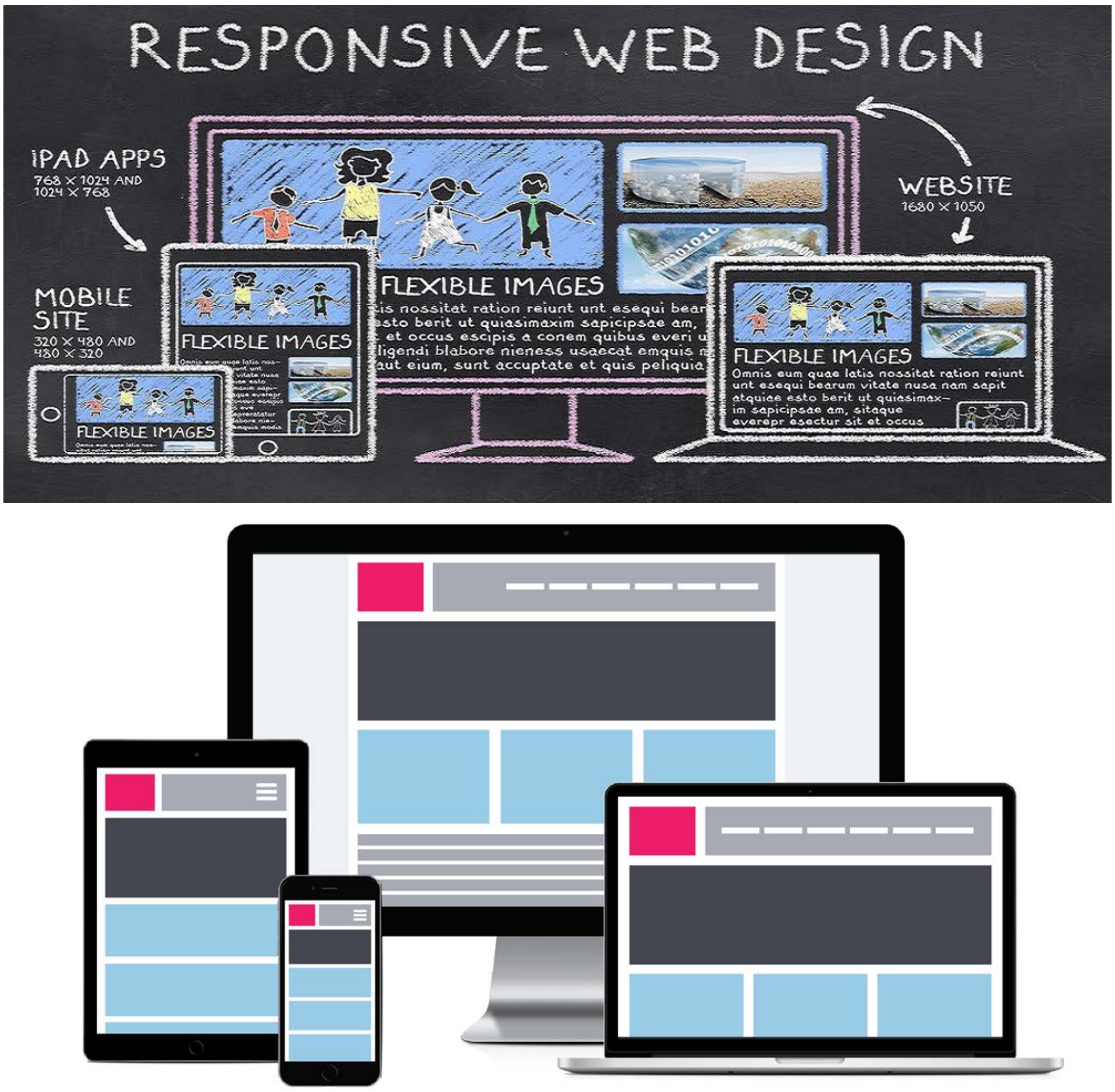


Fig. 14 Responsive web design example

2.4.3 Generated Apps

Generated Apps are developed using specific tools. Each of these tools provides its own techniques, languages, and libraries. This means it is necessary to learn the tool language and get used to the IDE to develop this kind of apps. These tools allow to generate the developed code in different platforms. They claim that the generated apps are native. But this kind of apps are called fake native apps because they do not satisfy all the native apps features, like the best performance or appearance. Either they cannot access all the device hardware capacities [51]. Some of these tools are Xamarin, GeneXus, or PencilCase.

2.4.4 Hybrid Apps

Hybrid Apps try to put together the best from native and web apps [70]. As shown in Fig. 15 Hybrid apps have the access to device capabilities advantages of native apps, and the platform affinity advantages of web apps. Hybrid apps can access all the device capacities, some of them natively and some of them using plugins [71]. This kind of Apps is cross platform. This means that with a single code it is possible to release the application on different platforms [72].

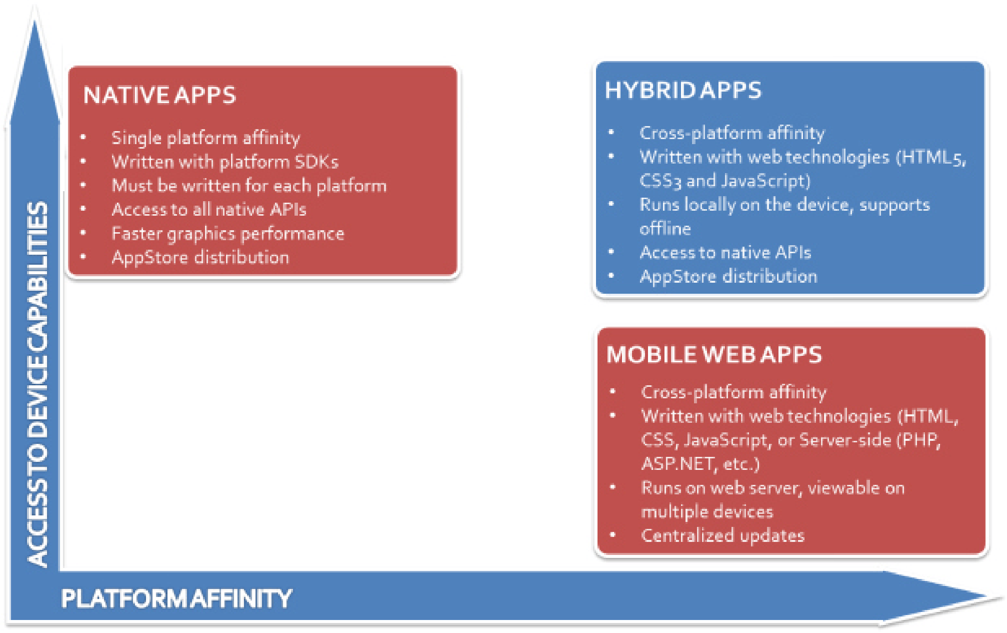

NATIVE APPS
Single platform affinity
Written with platform SDKs
Must be written for each platform
Access to all native APIs
Faster graphics performance
AppStore distribution
HYBRID APPS
Cross-platform affinity
Written with web technologies (HTML5, CSS3 and JavaScript)
Runs locally on the device, supports offline
Access to native APIs
AppStore distribution
MOBILE WEB APPS
Cross-platform affinity
Written with web technologies (HTML, CSS, JavaScript, or Server-side (PHP, ASP.NET, etc.)
Runs on web server, viewable on multiple devices
Centralized updates
ACCESS TO DEVICE CAPABILITIES
PLATFORM AFFINITY

Fig. 15 Acces to device capabilites vs Platform affinity in Native, Hybrid and Mobile Web apps [73]

Programming hybrid apps needs some specific frameworks. Nowadays, there are many in the market, as it can be seen in Fig. 16, but PhoneGap, Ionic, React Native and Flutter are four of the most popular frameworks for programming hybrid apps development as shown in Fig. 16 (Unity is focused on game development, even though it can be used to build non-games applications too [74] and Adobe PhoneGap is an open-source distribution of Apache Cordoba, which is also open source, plus additional features and tools [75]). React Native is the most popular framework as we can see in Fig. 16. Although Flutter is gaining popularity as we can check in Fig. 17 and Fig. 18. Even though all of them build hybrid apps there are big differences between them. The programing language they use or the way they render the layouts will define their apps performance and appearance [76].

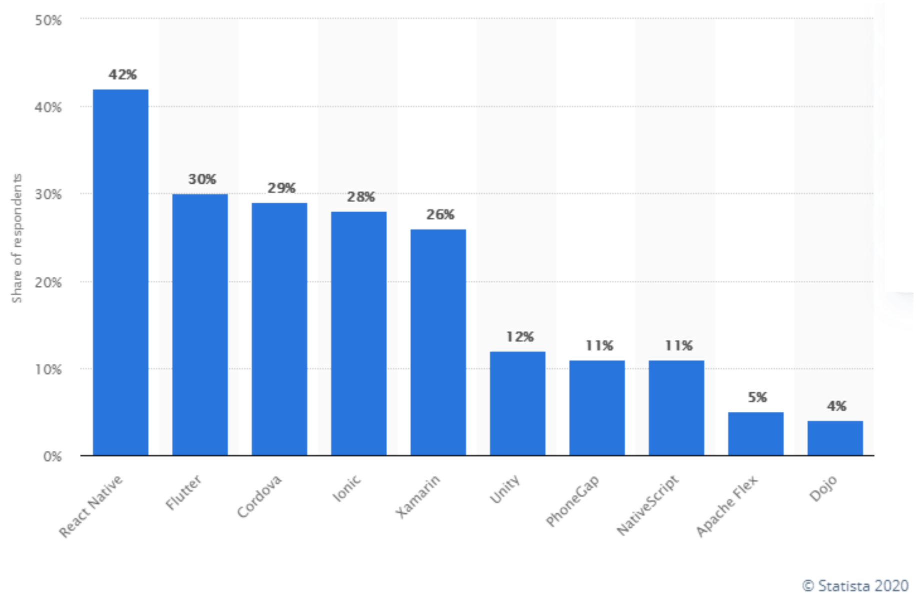


Fig. 16 Cross-platform mobile frameworks used by software developers worldwide as of 2019 [77]

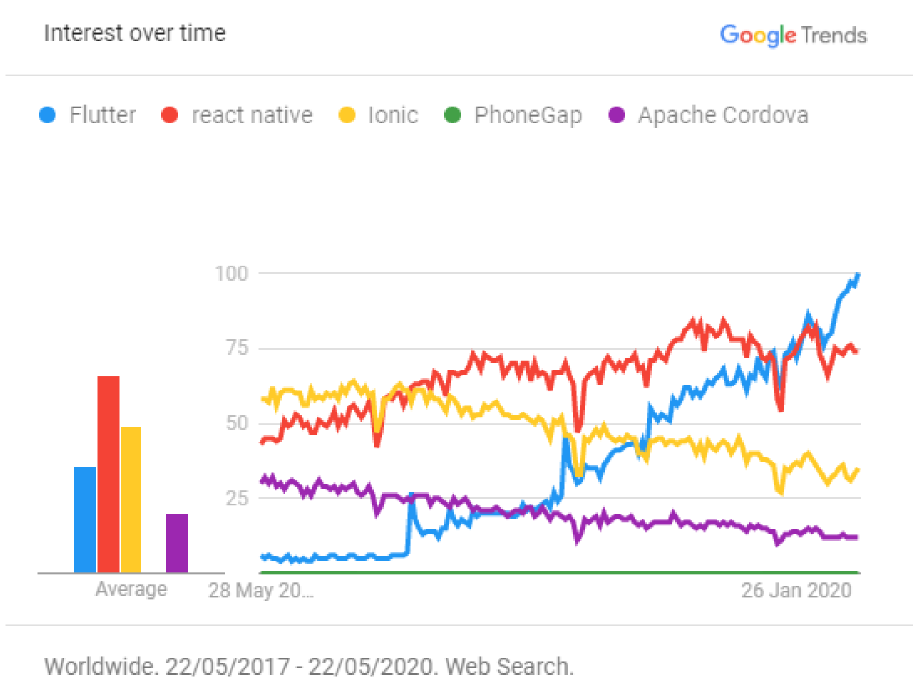


Fig. 17 Google Trends Cross Platform frameworks comparison last 3 years [78]

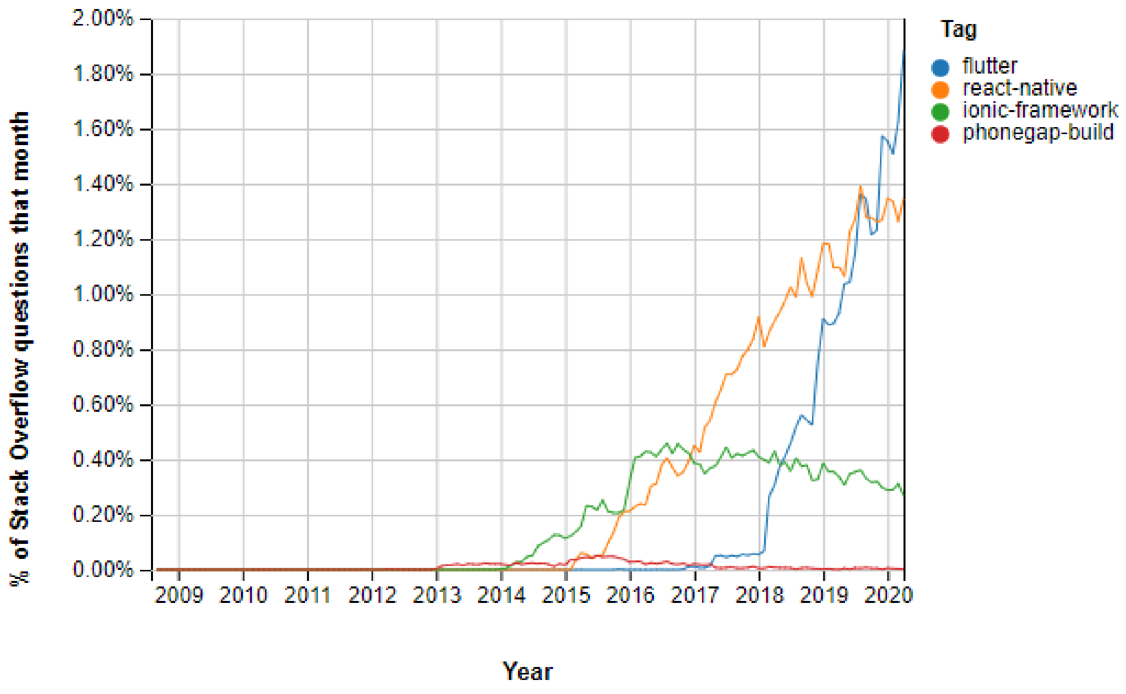
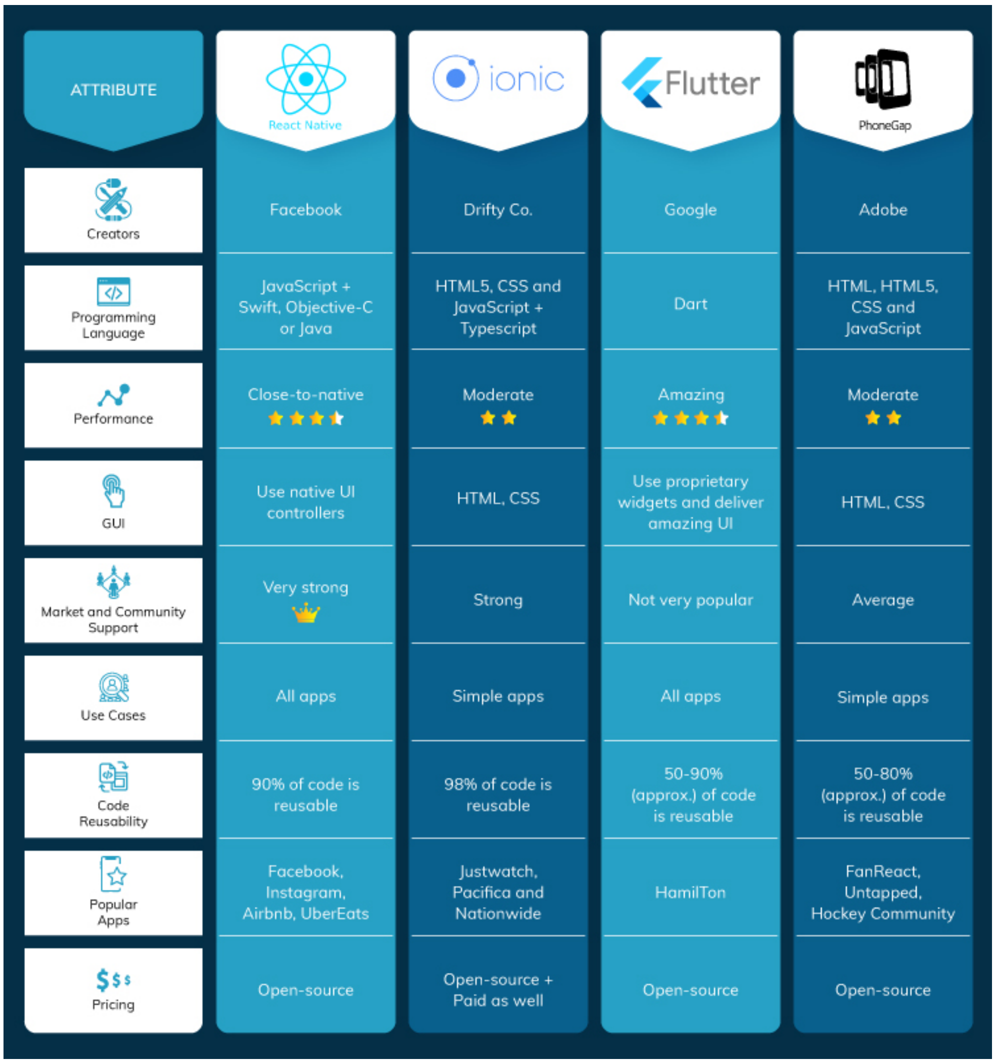


Fig. 18 Stack Overflow Trends Cross Platform frameworks comparison since 2008 [79]

As we can see in Fig. 19, Ionic and PhoneGap (both based on Apache Cordova [80]) apps are written using common web languages, therefore their layouts are more similar to websites [81], [82]. Thus, these apps are executed inside a WebView, as shown in Fig. 22. A WebView is a browser bundled inside the framework (native or hybrid), used to display web content inside an app [83]. On the other hand, Flutter and React Native apps are written using JavaScript and Dart respectively. The biggest difference between these two frameworks is the way they render layouts, which can be seen in Fig. 20 and Fig. 21. While React Native renders native components, linking their components to native components, Flutter renders its own components called widgets, without needing to create bridges to link those widgets to native code [84]. In conclusion, Flutter and React Native offer a better and closer to native performance and look and feel, than Ionic and PhoneGap. The use of web technologies in Ionic and PhoneGap decreases the performance against Flutter or React Native Apps as those use their own render engines [85], and makes the User Interface (UI) closer to Web Apps than to native apps. Additionally, React Native and Flutter can be used into existing native apps [86], [87]. Flutter has a bit of advantage against React Native regarding the performance, as Flutter does not need to link components to native code [84].



ATTRIBUTE	React Native	ionic	Flutter	PhoneGap
Creators	Facebook	Drifty Co.	Google	Adobe
Programming Language	JavaScript + Swift, Objective-C or Java	HTML5, CSS and JavaScript + Typescript	Dart	HTML, HTML5, CSS and JavaScript
Performance	Close-to-native ★★★½	Moderate ★★	Amazing ★★★½	Moderate ★★
GUI	Use native UI controllers	HTML, CSS	Use proprietary widgets and deliver amazing UI	HTML, CSS
Market and Community Support	Very strong 👑	Strong	Not very popular	Average
Use Cases	All apps	Simple apps	All apps	Simple apps
Code Reusability	90% of code is reusable	98% of code is reusable	50-90% (approx.) of code is reusable	50-80% (approx.) of code is reusable
Popular Apps	Facebook, Instagram, Airbnb, UberEats	Justwatch, Pacifica and Nationwide	HamilTon	FanReact, Untapped, Hockey Community
Pricing	Open-source	Open-source + Paid as well	Open-source	Open-source

Fig. 19 Comparison – React Native vs. Ionic vs. Flutter vs. PhoneGap [85]

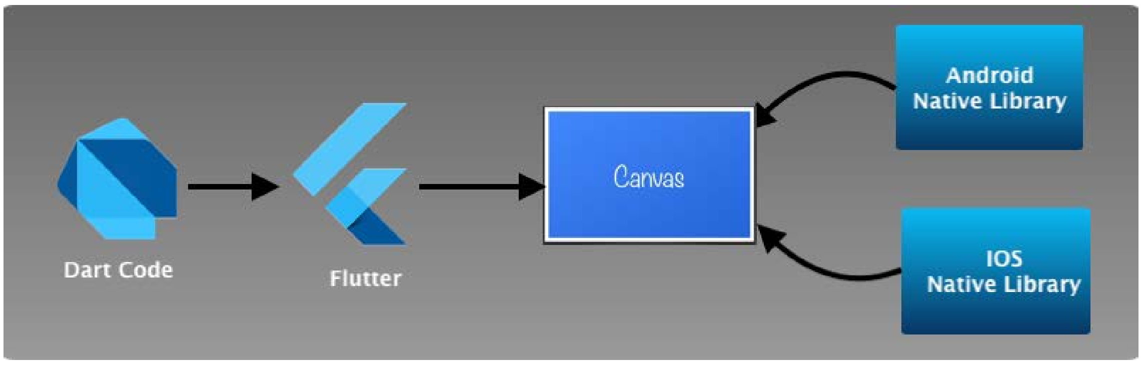


Fig. 20 Flutter rendering schema [88]

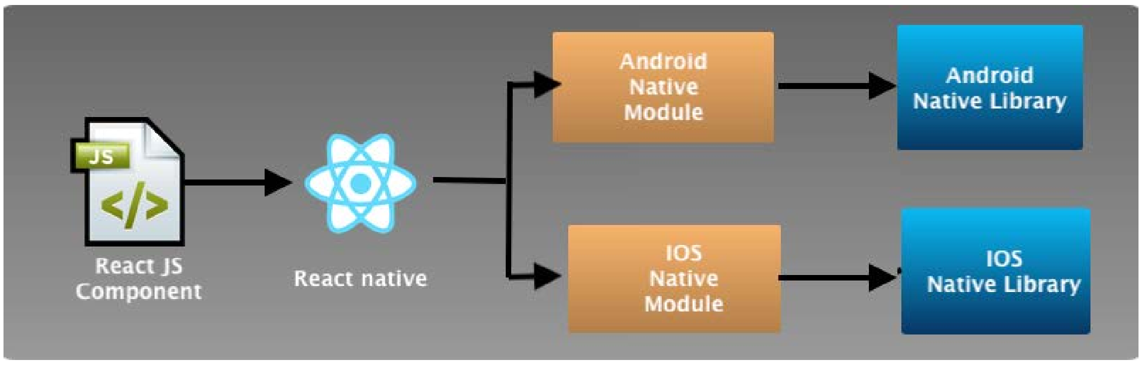


Fig. 21 React Native rendering schema [88]

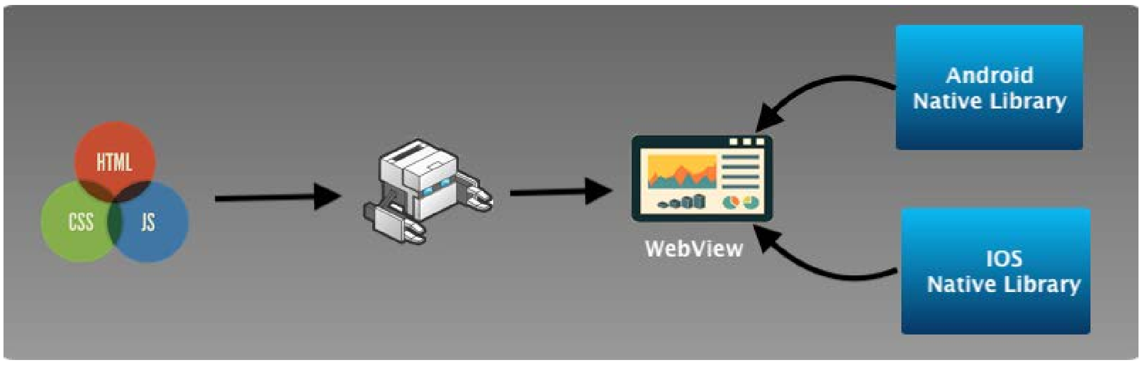


Fig. 22 Phone Gap rendering schema [88]

The app described in this project has been developed using Flutter. This means it is a hybrid app. The reasons for this decision are explained in 2.6.5. Once it is decided what kind of app to develop and the framework to do so, one of the facts left to specify is the design. Next section will explain the style guidelines followed.

2.5 Style guidelines

Regarding app development, is a good idea to follow some guidelines to ensure a cohesive product design and user experience. It is worth it to note that the development team members can change over the time as well as trends or features. Style guidelines are the tools that guarantee that the user interface design will remain coherent and user experience will not degrade over the time [89].

The frameworks that use web programing languages can use web design guidelines like W3C standards [90]. Apple has its own human interface guidelines, described in [91]. And Google has Material Design, their style guidelines for Android [92] and Flutter. Material comes bundled with Flutter [93]. There is a Material library that implements widgets that follow Material design principles. To use those widgets the app has to be “a Material App”. This means the root widget is MaterialApp [94]. The other bundled design with Flutter is Cupertino [95]. Cupertino widgets are iOS style. Developers can choose between those native designs, or exclusively use the standard widgets library, or mix widgets from both libraries [96], or customize existing widgets or build their own set of customs widgets, or even make platform aware widgets [97]. The app described in this document follows material guidelines, but also uses a lot of customized widgets. Some of the style decisions, like the theme color or some brand logos disposition in certain layouts are explained in the Final Degree Project by Mario Martín [28]. Below in this section there are explained some Material Design keys.

In the own words of Google: “Material is an adaptable system of guidelines, components, and tools that supports the best practices of user interface design. Backed by open-source code, Material streamlines collaboration between designers and developers, and helps teams quickly build beautiful products” [98]. So, it is a group of guidelines defined by Google in 2014 for Android design, but they also apply to the web and other platforms. It permits a high customization, balancing form and function. It is adaptable, not only to mobile devices but multiplatform too. For instance, one of the Material recommendations is the use of rounded corners for rectangle shapes [99].

In this design colors, shades, surfaces, edges, and depth take the main role. It focuses not only on where to place those objects but when (movement). The movement is a hallmark of material. Animations gain importance, which consider speed and direction. It tries to give a picture of order and hierarchy. And it is inspired by the physical world, trying to follow its laws of light, movement, and space [100], [101].

2.6 Flutter

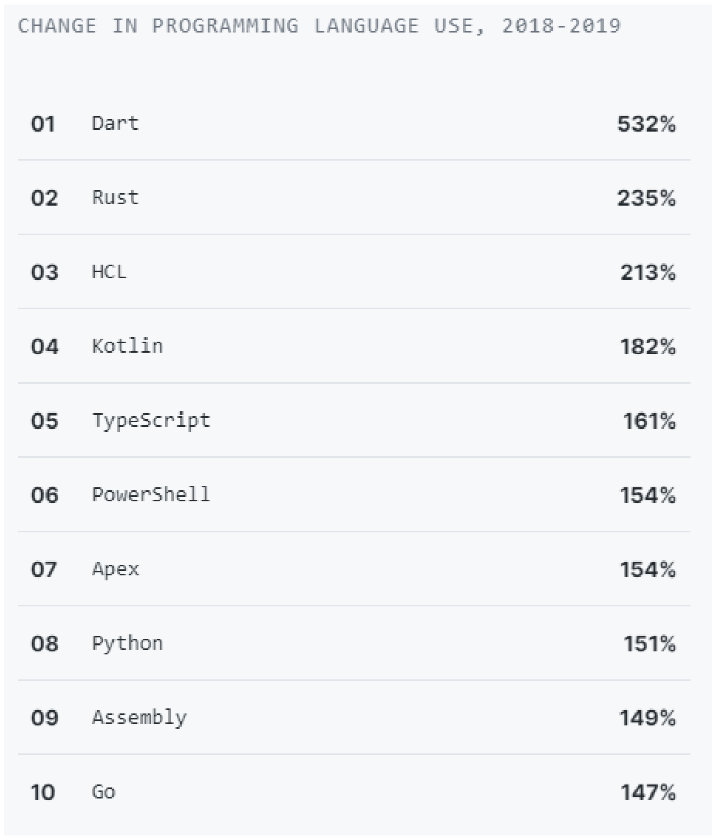
Flutter has been the chosen framework to develop the app. This section will detail some of the flutter aspects and the reasons of this decision.

2.6.1 What is Flutter?

Google defines Flutter in this way: “Flutter is Google’s UI toolkit for building beautiful, natively compiled applications for mobile, web, and desktop from a single codebase” [102].

Flutter was presented by Google in 2017. Its first stable version was Flutter 1.0, released on December the 4th of 2018 [103]. And its use has increased since then, as we saw in Fig. 17 and Fig. 18. Moreover, in 2019 Dart, which is the flutter’s programming language, was the language that increased the most, as it is shown in the Fig. 23. In the own words of GitHub: “With Flutter in our trending repositories, it’s not surprising that Dart gained contributors this year.” [104].

Flutter is free and open source [105]. This makes flutter bigger, creating a big community and making easier to find documentation. An important part of flutter is plugins, most of them are created by the community of developers [106]. As we have seen in previous sections, Flutter SDK is a hybrid app development tool. Flutter apps offer a good performance and native look and feel.



CHANGE IN PROGRAMMING LANGUAGE USE, 2018-2019

01	Dart	532%
02	Rust	235%
03	HCL	213%
04	Kotlin	182%
05	TypeScript	161%
06	PowerShell	154%
07	Apex	154%
08	Python	151%
09	Assembly	149%
10	Go	147%

Fig. 23 GitHub stats: Change in programming language use, 2018-2019

2.6.2 Dart

Flutter’s programming language is Dart. Dart is an object-oriented language created by Google. The Dart project was founded by Lars Bak and Kasper Lund and unveiled in 2011. Its first stable version, Dart 1.0 was released in 2013. Since then, Dart 2.0, released in August 2018, introduced important changes in the language. At the beginning the project was oriented to web apps. Google even pretended to add a Dart VM to Google Chrome web browser, although this idea was dropped in 2015 [107].

Nowadays the only browser implementing a Dart VM is Dartium, a special build of Chromium [108]. But dart applications can be run on any modern browser as it can be compiled to JavaScript [109].

Flutter choses Dart to help developers to quickly build quality apps for multiple platforms, as it is stated in its website: “Dart is a client-optimized language for fast apps on any platform” [110]. Dart code can be compiled to ARM and x64 for mobile, desktop and server apps, and optimized JavaScript for the web [111].

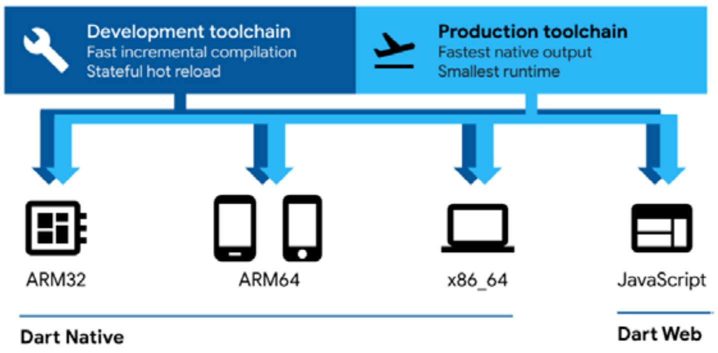


Fig. 24 Dart flexible compiler technology [111]

Dart enables one of Flutter's most interesting features, Hot Reload, which allows to inject updated source code into the executing Dart Virtual Machine. This can rebuild the UI in less than a second, reducing the developing time and improving the developer experience [112].

Dart is easy to learn as it is similar to languages like Java, Swift or JavaScript. It has a C-style syntax. Dart is optimized for building user interfaces, and it is an object-oriented, class-based, garbage collected (automatic memory management) and single threaded language [112].

Despite of being a single thread language, Dart supports futures, streams, background work, …, and anything one could need to build a modern, asynchronous (and reactive in the case of Flutter) app. All the dart code runs in an Isolate. An Isolate is a little space in the machine with its own private part of memory running a single thread running an event loop. Any flutter app runs at least the main Isolate. This can be enough for most simple apps. But when processing becomes more demanding, there is the possibility of creating a new isolate to run this function while leaving the main thread free to render the widget tree. These two Isolates are isolated, even though one of them is the parent of the other. They do not share memory and they can only work together passing messages to each other. This might seem more difficult than programming languages like C++ or Java where there is memory sharing, but this memory isolation has some advantages. For example, there is no need to block the memory as any thread can only access its own memory [113].

The event loop is what makes dart asynchronous code possible. Every time an event occurs, like a screen tap, or I/O event from the disk, it is added to the events queue. The event loop just processes one by one at every completed loop. Flutter in focus video about the Event Loop explains: "All of the high-level APIs and language features that Dart has for asynchronous programming — futures, streams, async and await — they're all built on and around this simple loop." [114]

In the app version described in this work all the code runs in the main Isolate.

2.6.3 Flutter architecture and features

Flutter architecture is layered as we can see in Fig. 25. Each layer is built upon its previous layer. The lowest layer is the embedder. It takes care of platform-specific tasks like surface settings, thread settings or plugins. This layer is minimal, so the platform simply provides a canvas and the rest of the rendering-related logic occurs inside Flutter, this makes a good cross-platform consistency [115].

Regarding Flutter Engine, Flutter GitHub repository describes it as: "The Flutter Engine is a portable runtime for hosting Flutter applications. It implements Flutter's core libraries, including animation and graphics, file and network I/O, accessibility support, plugin architecture, and a Dart runtime and compile toolchain" [116]. The Engine is implemented in C/C++. It uses Skia Graphics Library, a 2D graphics library already used before in Chrome, Android, or Mozilla Firefox. The use of its own rendering engine gives a higher performance in rendering and animations. Flutter aims to provide from 60 frames per second (fps) performance to 120 fps on devices capable of it. The Engine also includes Text and Dart, which encompasses Dart Runtime and Garbage Collection as well as Just in Time (JIT) support for debug mode, and native "arm" compiling by Ahead of Time (AOT) support for release and profile modes [115], [117].

The framework is the top layer and thus the most used. It comprises the widgets, animation, or gestures. We can see how the framework is written in Dart. At the top is where is placed the code written by the developer. The developer will not be limited by (Original Equipment

Manufacturer) OEM widget sets. Flutter widgets come from Material (see Material Design section) and Cupertino (iOS look) libraries, although developers can build their own widgets composing others [118].

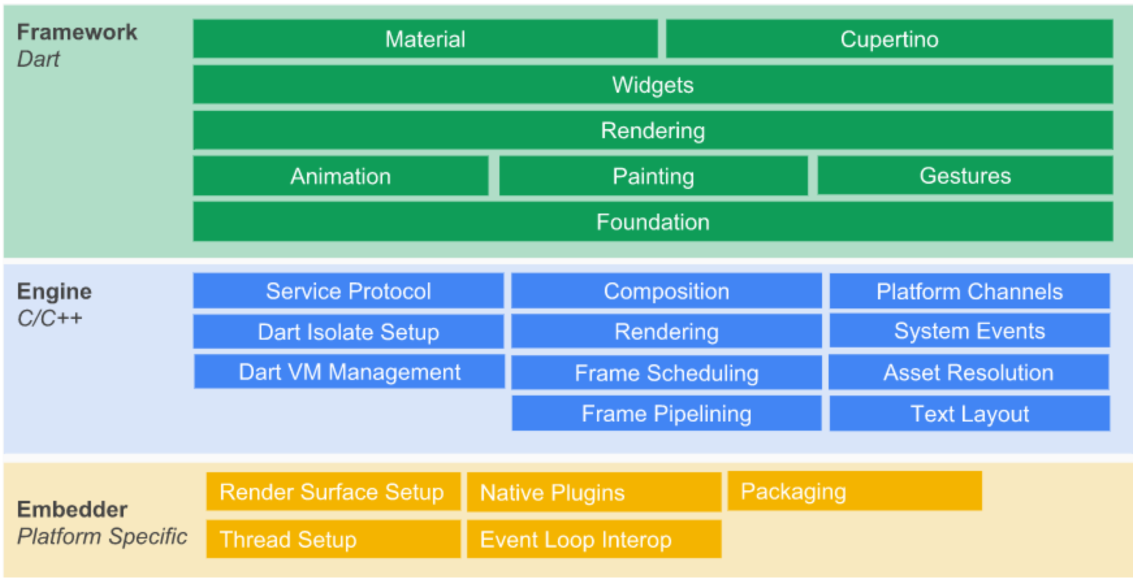


Fig. 25 Flutter system overview [119]

One of the main advantages of Flutter is its architecture and definition allow a higher productivity of the developers. Flutter claims to increase productivity due to [117]:

- Only one codebase is enough to develop to both Android and iOS
- Hot Reload and test units accelerate the developing
- Do more with less code. Flutter takes a declarative approach. This means that flutter builds its user interface to reflect the state of the app. Fig. 26.
- Also flutter architecture is designed to get the most with the least code. The framework's upper layers are used more often than the lowers.

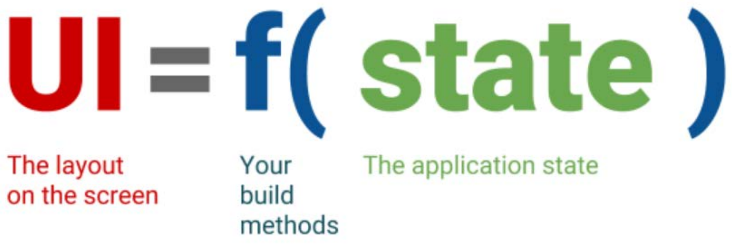


Fig. 26 Flutter is declarative [120]

2.6.4 Widgets: Key concept in Flutter

Widgets are the key concept of Flutter, which is in charge of defining most of the program structure. Widgets are Dart classes that expand a Flutter Class [121] and they are flutter basic user interface elements [94]. Although widgets are the elements that build UI, not all widgets are visible. Widgets can define from structural elements (like a button or a menu) to Style elements (like a font color) to Design aspects (like padding). Even layout models are widgets in Flutter [119].

Flutter UIs are built composing multiple widgets. This is an innovative way to create layouts and possibilities are unlimited [122]. Each widget is wrapped inside a parent widget and inherits some of its properties [123].

When the application starts, the main function calls the runApp() method, which takes the first widget and makes it the root of the widget tree. The widget tree is formed by all the widgets that form the app. A second level is the Element tree, which is formed by a representation of the widgets displayed at the moment. There is a third lower level tree. The RenderObjects tree. When Flutter builds the UI it looks at the RenderObject tree. Widgets configures, Element manages and RenderObjects paints [124].

These separation in three levels allows flutter to optimize the performance reusing the parts of the tree that does not change instead of rebuilding the entire tree every time something changes [125].

Widgets are immutable. But UIs are never immutable. They are constantly changing. As said before, Flutter is declarative; only one code path defines what the UI should look like for any given state. This is because Flutter separates between its widgets and their state. Widgets describe the configuration for an element, and elements are mutable representations of a Widget in a particular location in the tree.

In Flutter any widget is either a subclass of Stateful Widget or Stateless Widget. Any widget has a build method and stores some properties. And any widget is immutable. But we will see now how stateless and stateful widgets differently manage this [126].

Stateless widgets store their properties on final variables [94]. They are only built once.

Stateful widgets create State objects that store their state. While Widget objects are temporary, State objects are persistent between calls to build() method. The values that contain the state can change. When this happens the state object calls setState() method, and the framework redraws the widget [127].

This separation between the appearance and the state allows other widgets to treat stateless or stateful widgets the same way. A parent widget does not have to worry about losing its child widget state, the framework does the hard job of finding and reusing existing state objects when appropriate. Flutter lets the developers describe the current UI state and leaves the state transitioning to the framework [119].

2.6.5 Why Flutter?

Twizy Contest 2020 asked for a mobility solution for the Paris Olympic Games 2024. Olympic Games are an international event, where people from all over the world would go to Paris. This means multiple devices, brands, and OS. Moreover, our budget and time for our project was limited. That is why it was decided to develop a multiplatform app. So, only one code would have to be developed for both platforms, reducing time and cost while not sacrificing performance or user experience. Flutter is backed up by Google, this brings the necessary confidence in knowing that it will not be abandoned and it will likely increase.

Now that the available technologies have been introduced and the taken decision has been explained it will be done a research and analysis of mobility apps to decide the design of our own.

2.7 Current mobility apps comparison

Before starting coding, it was carried out in-depth market research to analyze the current trends in mobility apps. This analysis is important when it comes to designing some of the app layouts and deciding the app structure. This section will analyze some carsharing and public transport apps that operate in Paris. After this analysis conclusions will be extracted and it will be explained how these conclusions inspired our app design.

2.7.1 Carsharing Paris apps

ZITY

ZITY is a carsharing company owned by Ferrovial (Spanish) and Renault (French). They work in Madrid since December 2017 with 725 cars, and they have recently started working in Paris (March 2020) with around 500 vehicles. ZITY fleet is formed only by Renault ZOE 100% electric vehicles [23, 32, 128].

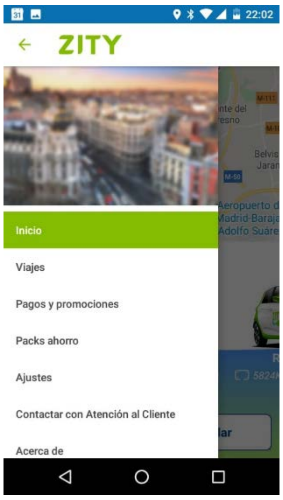


Fig. 27 ZITY splash screen *Fig. 28 ZITY loading map* *Fig. 29 ZITY marker display* *Fig. 30 ZITY menu*

In Fig. 27 we can see how ZITY bets for a custom brand splash screen (The splash screen is the static screen or animation that is displayed to the user while the app loads its internal elements. This screen usually displays the brand logo). The loading screen (Fig. 28) shows a loading bar and transparency that lets see a map. The main screen is a map that shows markers wherever there is a vehicle of the fleet. A nice feature of this app is that displays a green zone where the carsharing service operates. When a marker is tapped it displays a widget that shows information about the car battery, range, and license plate (Fig. 29). And it gives the option to book the car or rent it. There is also a details button that provides more information about the car and the service fare. The tapped marker also shows its location and can launch google map to get directions to it. The top bar contains an icon button that can display the menu where we can navigate to some other views, like settings or our trips log (Fig. 30). The colors of this app are mainly green and blue.

To use ZITY it is necessary to download the app (available on Android and iOS) and register. In order to register it is needed a valid email and phone number as well as a selfie and scanning the user's driving license. Registration is free. The app requires location permissions and it supports multiple languages (English, Spanish and French checked). The app has more than 100.000 downloads in the Play Store, a rate of four stars over five and more than 5.000 reviews.

To get inside the vehicle the users use the app. The app will then inform the user about the current damages that the car has and will request to inform if any other damage not reported is perceived. The vehicle inside has instructions and phone charge wires. Users can drive around with the vehicle and even park it somewhere and leave it locked for a short period paying a different rate. Once the user is done with the vehicle, they can park it anywhere and release the car with the app, then the cost accumulated will be displayed.

These vehicles are electric but do not include chargers. A company team finds low battery vehicles and charges them.

Ubeeqo

Ubeeqo is another carsharing platform operating in Paris. But they also operate in London, Hamburg, Milan, Madrid, Barcelona, Copenhagen, and Switzerland. They have a wide variety of vehicles. This service works a bit different than other carsharing platforms. Each vehicle has a designed parking spot, and users have to park it there again once they are done with it [129].

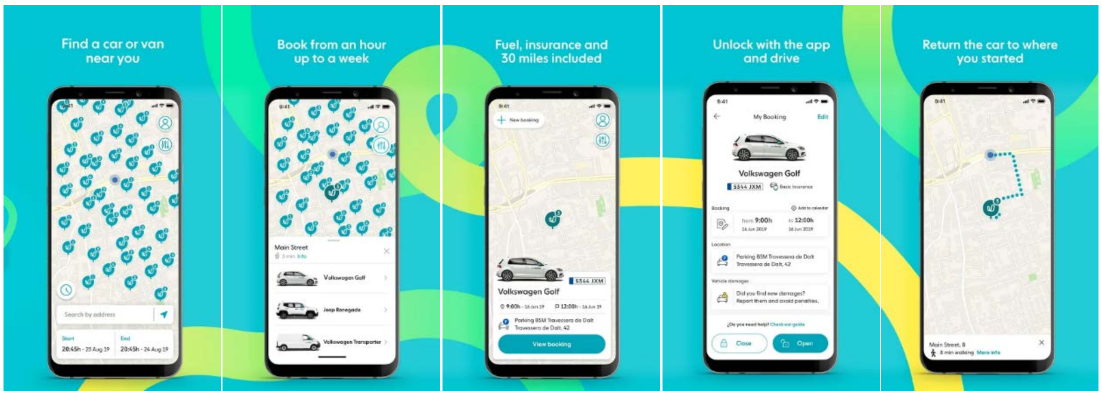

Find a car or van near you
Book from an hour up to a week
Fuel, insurance and 30 miles included
Unlock with the app and drive
Return the car to where you started

Fig. 31 Ubeeqo screenshots [129]

Ubeeqo main page (Fig. 31) is also a map. Markers pop up wherever there is a car of their fleet. Once one of those markers are tapped, we can see the vehicle info and the service fee. There is also a searching bar and a button to display the menu.

Ubeqoo app (available on Android and iOS) main color is blue. To use Ubeeqo it is necessary to register, but it is no needed to download the app. The service can be used through the website too. In order to register it is needed a valid email and phone number as well scanning the users driving license and upload a selfie holding the driving license. Registration is free. The app requires location permissions and it supports multiple languages (English and Spanish checked). The app has more than 100.000 downloads in the Play Store, a rate of four stars over five and more than 1.500 reviews

To get inside the vehicle users can use the app or a contactless card. Ubeeqo recommends having a contactless card as some vehicles can be parked in underground parking slots where there can be signal problems with the phone. This contactless card can be Ubeeqo own contactless card or credit contactless card or even some public transport contactless card. Once inside the vehicle the user will also have to inform if any non-reported damage is perceived. To see the reported damages there are some paper documentation inside the vehicle glove box. Here it can also be found a terminal that the user will use to inform of the damages, get instructions and get the car keys. There is also a fuel card as fuel is included in the service rate.

Once the user is done with the vehicle, they must return it in the same parking where it was picked up. If electric, the vehicle has to be plugged in when returned. The vehicle release process is done by means of the glove box terminal not the app.

Ubeeqo offers different rates and options for short distance and long distance carsharing as well as the option to book a vehicle.

Free2Move

Free2Move is the PSA Group carsharing platform. Free2Move shows vehicles from different carsharing services, or even some bike or scooter sharing services. They operate in more than 33 cities of Europe and USA, including Madrid, Lisbon, Paris, Berlin, or London. In Paris, this service has some advantages, like free parking on paid parking areas [130].

Fig. 32 F2M Splash screen

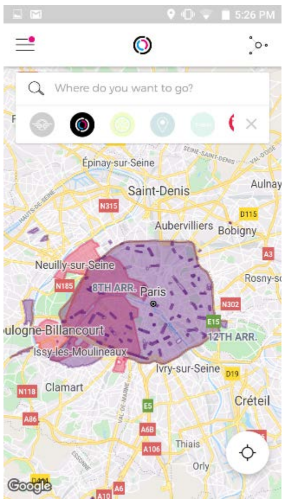


Fig. 33 F2M Map page

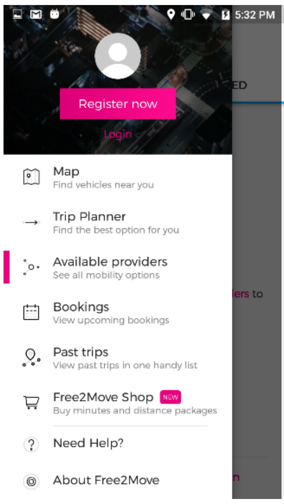


Fig. 34 F2M Menu display

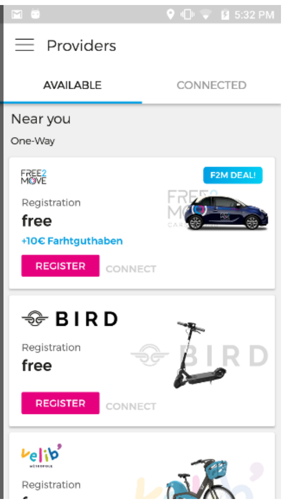


Fig. 35 F2M Providers list

*F2M: Free 2" Move

The app screenshots show a very simple splash screen, as it shows only their brand logo (Fig. 32). The main view is also a map (Fig. 33). From it we can see the operating zone of the platform, display the menu, configure filters or alerts, or check the available providers list. There is also a searching bar.

The bar displays different markers for each service. When the user taps any of these markers a widget shows up displaying the vehicle name, image, range, location, and license plate. And the service name and fees. And it gives the option to reserve it.

This app (available on Android and iOS) uses colors black, blue, and pink, like their brand logo. It requires location permissions and it supports multiple languages (English and Spanish checked). The app has more than 500.000 downloads in the Play Store, a rate of three stars over five and more than 3.000 reviews

As this platform offers a wide variety of services, the app has multiple options. Some of the French providers are (Fig. 35) Car2Go, moov'in Paris, travelcar, ubeeqo, Peugeot rent, citrone rent&smile, ds rent (carsharing) coup (motor scooters), velib, donkey (bikes) VOI, or tier (electric scooters)

2.7.2 Public transport Paris apps

There are several apps about the public transport in Paris. I have analyzed the official apps from the RATP operator: Next Stop Paris and RATP apps (both available on Android and iOS).

RATP

RATP app provides combined information about all the public transport in Paris. From the Metro to the Bus or even some private companies like scooters renting.

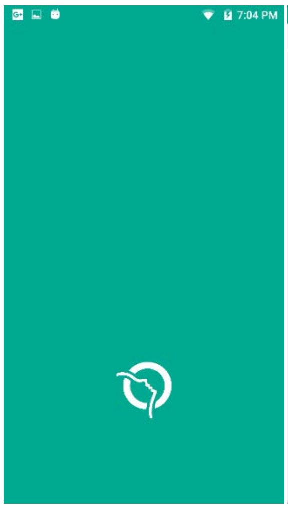
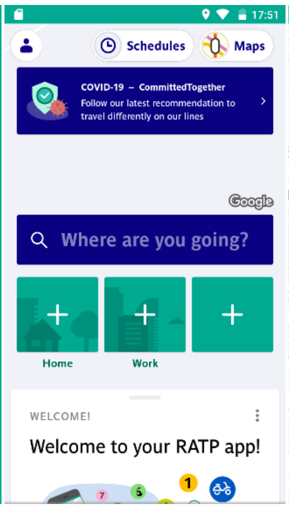


Fig. 38 RATP splash screen


17:51
Schedules
Maps
COVID-19 – CommittedTogether
Follow our latest recommendation to travel differently on our lines
Where are you going?
Home
Work
WELCOME!
Welcome to your RATP app!
Fig. 37 RATP main page

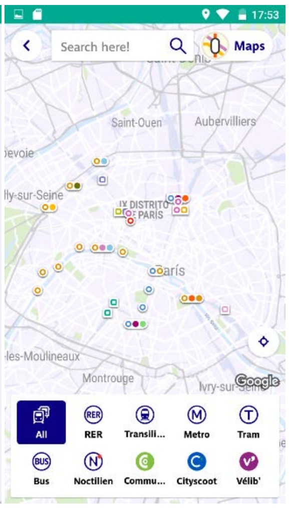
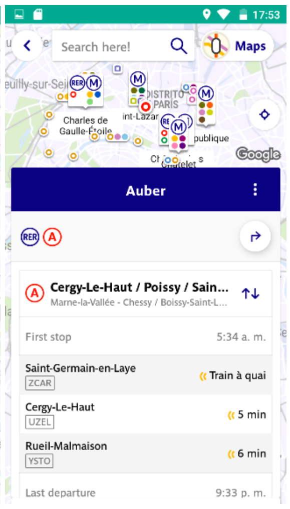


Fig. 36 RATP map page



Cergy-Le-Haut / Poissy / Sain...
Marne-la-Vallée - Chessy / Boissy-Saint-L...

First stop	5:34 a. m.
Saint-Germain-en-Laye ZCAR	Train à quai
Cergy-Le-Haut UZEL	5 min
Rueil-Malmaison YSTO	6 min
Last departure	9:33 p. m.

Fig. 39 RATP marker display

In these pictures we can see RATP splash screen is quite simple, as it shows only their brand logo (Fig. 38). RATP bets for a busier main page (Fig. 37) rather than just the map view. This main page displays some information about COVID-19 (in the screenshots from the Play Store this space seems empty, letting see the map more clearly) as well as some icon shortcuts to save home or work location. The map is still the background of this main page (Fig. 36). The main page can drive you to either the map or either a slide from bottom widget with some advices or other information. We could also display the menu, check schedules or the transport map for each mean of transport. Once in the map, there is a bottom bar that let us choose what kind of markers we want to be displayed. Once we tap on any of those markers, we can see the schedules of that station or get directions to it (Fig. 39). There is also a search bar to find stations.

RATP main colors are green and blue. This app requires location permissions too and it supports multiple languages (English and French checked). The app has more than 5.000.000 downloads in the Play Store, a rate of two stars over five and more than 68.000 reviews.

Next Stop Paris

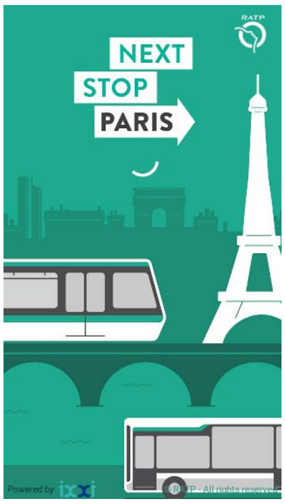


Fig. 43 NSP splash screen

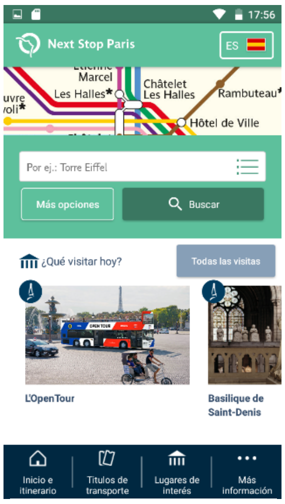


Fig. 42 NSP main page

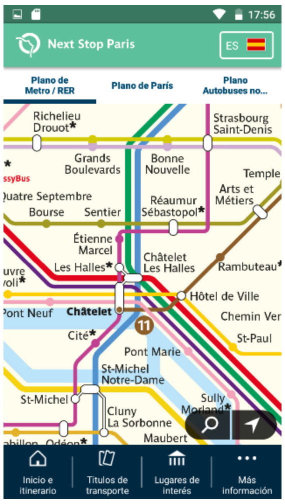


Fig. 41 NSP map page

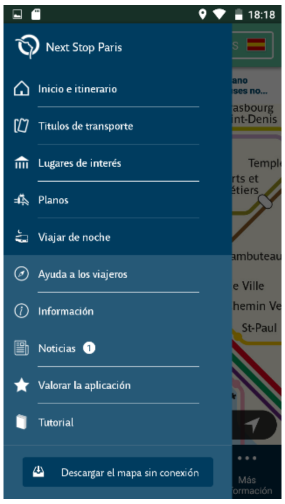


Fig. 40 NSP menu display

*NSP: Next Stop Paris

This app is more focused on tourists. It gives information about places of interest, metro, and bus. It is supported by RAPT, the same as the RAPT app, so the app is quite similar to the previous one.

In the screenshots we can see how its splash screen is more sophisticated (Fig. 43). The main page (Fig. 42) is very similar, with the map (Fig. 41), a search bar, and some tourist info. There is a common bottom bar across the views that let us navigate through the three pages: the main view, the get tickets view and the menu display (Fig. 40).

Next Stop Paris main colors are green and blue too. It requires location permissions and it supports multiple languages (English and Spanish checked). The app has more than 1.000.000 downloads in the Play Store, a rate of more than four stars over five and more than 14.000 reviews.

2.7.3 Conclusion

The carried-out market research shows some common characteristics in the design of carsharing apps that must be considered for our own:

- Simple splash screen displaying the brand logo.
- Company colors are used as the theme colors for the app, remarking the brand image.
- Simple screen layouts, leaning towards images and icons over text. The main screen is always a map, whose markers display the information about the vehicle. Actions must request the minimum possible touches, avoiding unnecessary deployable submenus.
- Other common features are: vehicle reservation and identification of vehicle damages before use.
- Easy and quick start. Download, register and start enjoying. Registering process usually requires personal identification data as well as uploading the driving license.
- Multiple language support.
- Available on both Android and iOS.

The reader might have noted that these apps have a large number of downloads and reviews in the market store, this provides a valuable feedback channel for the service providers, not only about the app but about the service itself.

Our app has similarities with this apps, it supports multiple languages, require location permissions and uses the map view as the main page. The market research showed the mobility apps trends to follow and the app has been built accordingly. Even though there is still a long developing process to be done to improve the app and add functionalities. The next chapter details how the app has been built, its design and its functionalities.

3. IMPLEMENTATION

In the previous chapter, the main current technologies to program apps, as well as the main characteristics of the current carsharing apps were studied. In this chapter all this information is used together to design and app for our service. Thus, this chapter describes the app development procedure. It explains from the background functions that connect the app to the server, to every screen layout. The app structure is detailed as well as how the flutter widget tree works. First the user side is introduced and later the admin side functions are described. It is also explained cross functionalities like internationalization, the app bar dropdown menu or the permissions request.

3.1 Criteria for the App design

As it was showed in 2.5 this app follows some style guidelines, and as explained in 2.7 there are some trends in the mobility apps that should also be followed. Therefore, TwizyLine App should offer users an intuitive interface to register, log in, and access to the service. It will be written in Flutter, which allows to release the app in the two current main operative systems, Android and iOS.

The user experience should be clear and easy. Therefore, the app should show a login page the first time it is installed on the user’s device, giving users two options: Log in with an existent user or register in the platform. Once the user would have completed one of those steps, the app should request the user for location permissions. And only once those permissions are given (otherwise the app should not work) it would get the user current position.

There is needed a main screen to give users an easy way to find the closest parking and request a vehicle. This screen should be a map which would display markers wherever there is a TwizyLine station. When the user taps on any of those markers, a pill would show up, providing the user information about the parking name, the available cars in it, and the free spots. From this pill the user could request a vehicle just tapping on the pill. The app should then inform the user about its assigned Twizy and provide them with a method to access the pick-up zone, for example a QR code. A nice functionality to add would be a way to get directions to the desired parking, as most users would come from other cities or countries and will likely need to be oriented.

About the admin side, it is important to provide them with useful functionalities and an easy user experience in order to improve the failure response and reduce the time that it takes admins to find and solve any trouble with the system. Thus, the appearance here is not so important. The admin functionalities include: Error log page, where the administrator could see any warning or error emitted by any car and solve it or discard it. Commands page, where the administrator could send to a vehicle specific commands, or receive responses from it, without passing through the server. Management page, where the administrator could check any parking situation or any vehicle log. And Map, similar to the common user page, but with the added functionality of showing the fleet vehicles position.

3.2 First steps. Connecting to the MQTT broker.

All the information between the back-end and the front-end (our app) of TwizyLine project is transmitted by using MQTT protocol. This is a machine-to-machine and IoT protocol based on a publisher/subscriber paradigm. The decision to use MQTT as the technology to connect the front-end and back-end is explained and can be fully read in the Final Degree Thesis report of the TwizyLine member Samuel Pilar [26]. The messages that the app exchanges with the server and the vehicles can be found in ANNEX 1. In ANNEX 2 it can be found the errors that can be handled through MQTT messages.

To connect the app and the server, it was necessary to install a plugin, define the topics and messages to be used, code the necessary functions to handle the communication and set several streams to pass the data through the app. All of this work is done in the mqtt.dart file.

When started working on this app the first challenge was connecting the phone with the server using MQTT. In order to get this job done the best option is to use an appropriate plug-in for flutter. Flutter plugins can be found at [106], in which can be found the information about the plugin, the changelog, use examples, installing instructions, version log and scoring. There is a huge number of plugins for diverse utilities. Some of them are uploaded by the flutter team, but most of them are uploaded by the community of developers that joins Flutter. Some of those developers are verified publishers. My app only uses verified publishers' plugins. Thus, it was decided to use mqtt_client plugin for flutter [131]. This plugin was, at the time when the develop of the app started, the only certified plugin that offered an MQTT client. It also offers a nice example on its documentation that was used as a base for the function that connects the app with the MQTT broker [132]. Another option, released on June 16th of 2020, is universal_mqtt_client plugin [133].

To install a plugin in flutter you just need to add some lines to the pubspec.yaml file and execute the command Pub get. These plugins can be updated, or you could check if they are outdated using some commands.

After that, the mqtt.dart file must be configured. This file does not contain any widget or class, as it does not create anything "displayable". The file contains instances of the MQTT plugin classes, which need the broker IP direction, to leave the client identifier on blank and to use the default port (1883).

MQTT is an IoT protocol that works with topics. The broker publishes topics and users subscribe to them, getting the data shared on those topics and being able to send their own data through them. My partner Samuel Pilar gives a more detailed explanation to MQTT in [26]. The app currently subscribes to 9 different topics divided into common user topics and admin user topics. Starting with the common user topics:

- "Log": This topic is only used during the login or logon process. Once the user has logged into the system the app unsubscribes from this topic. All users subscribe to this topic.
- "App": This topic keeps subscribed during the entire lifecycle of the app. The client subscribes to it once he/she is logged. It is used to get the periodic information the server sends, updating the parking state; free spots and available Twizys. All users subscribe to this topic. Users do not send anything through this topic.
- "username/app": This is a custom topic for every user. It is their personal channel to communicate with the server. In this topic the server puts the information and the client reads it. It is used to check if the user has admin permissions or not, to get the parking areas location at the start, and to get the QR code for the turnstile and the information about the assigned Twizy when requested.
- "username/server": This topic is the same as the "username/app". But the user writes data on it and the server reads from it.

The topics that are exclusively for user admins are the next ones:

- "admin": This topic would be the equivalent to the "app" topic but for admins. All admins subscribe to it and get the periodic error report from it. Admins do not send anything through this topic.
- "username/admin": This is a custom topic for every admin. In this topic the server puts the information and the client reads it. It is used to get the error report from the server on the start, to get the parking state to represent it, or the car log to draw its table.
- "username/admin_server": This topic is the same as the "username/admin" but here the client writes, and the server reads. It is used to request the necessary information to represent a parking, to request a car log, or the error report from a car. And also, to inform the server that an error has been fixed.
- "+/location": MQTT allows to create sub topics inside topics, this way you can subscribe to the topic vehicle1 that will presumably transport information about the entire vehicle, or you can subscribe to the topic vehicle1/tires, that will presumably transport information just about the tires state. The + means all topics, so with +/location the client subscribes to any topic that contains a subtopic called location. The app needs in this case the topic vehicleId, which is unique for every Twizy of the fleet. Each of those have a subtopic called location that periodically sends the position of the Twizy. This topic is subscribed when the admin accesses their map page, so they can see all the Twizy's real time location.
- "+/info": Similar to the previous topic. This topic contains the responses Twizy sends when it receives a command. This topic is subscribed when the admin accesses the commands page.

Dart allows to use streams to handle communications. Dart streams are a sequence of asynchronous events. As mention in the previous chapter during the analysis of Dart language, Dart is a single-threaded language, so streams work like if they were running on the background but they are actually running on the same thread. This means that it is not a good idea to make an event iterable the next way:

```
while(1) {
  if(somethingHappens){
    return 0;
  }
}
```

as the thread will just keep forever in the while(1) loop, as it will not let other functions work because somethingHappens will never happen.

The right way to handle the streams is to use stream controllers, which were defined in mqtt.dart file the next way:

```
StreamController<ReceivedMessages> controller =
    StreamController<ReceivedMessages>.broadcast();
Stream stream = controller.stream.asBroadcastStream();
```

There are two kinds of streams in dart: single subscription or broadcast streams. Single subscription streams can only be listened by one function. These streams can be problematic if the widget containing the function rebuilds or if they are not properly closed before listening to them again. So just for simplicity all the streams are broadcast in our app.

Once the necessary elements to handle the connection have been introduced it will be explained the functions that connect with the server and use those elements. To establish the connection with the broker it has been coded the mqttStart function. This function has a return type Future<int>. Dart futures represent a computation that does not complete immediately. Normal functions return a value when complete, but future functions return futures, these objects will contain at some point the result value, but, while not, the thread can continue working (or not if we use the await call) and the future will tell when it is ready.

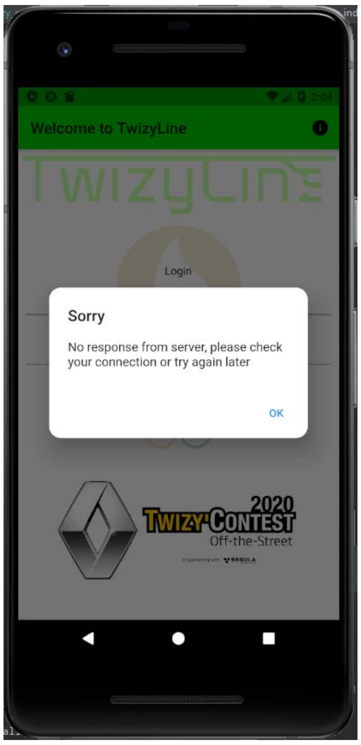
The mqttStart function is called from the login page (this is the first page displayed to the users when they open the app). This function starts the communication with the MQTT broker and manages the authentication process as it sends the broker the user credentials and returns a different value depending on whether the broker accepts the user credentials or not. The mqttStart function gets the user and password that the user enters in the login page as parameters. The first thing this function does is defining the connect message with parameters like the keep alive period, the client identifier or if authentication is necessary or not.

The username is the only client identifier, no other parameter is used. This username obviously must be unique. This is checked when the user registers on the platform, if the new user tries to register with a username that already exists, the app will request them to use a different one. The authentication is activated on the broker, and users must be registered in a certain way. There is a file the server checks where all the users username and password are saved. This file encrypts the passwords to ensure security. The server adds the user credentials to this file when they register.

Once the connect message is defined, the mqttStart function tries to connect with the MQTT broker. If there is no internet connection or the MQTT broker is down, the user gets an alert message explaining the failure.

Once the connection is established the app can receive or publish messages. The MQTT client has a change notifier object which is listened to get notifications of published updates to each subscribed topic. Every time a message is received on any of the subscribed topics, the function decides where each message goes. Depending on the topic in which the message was received each message is placed into its corresponding stream. This change notifier keeps listening the whole time.

During the use of the app, there are some requests to the server. Every request has associated a timer, so if there is no response from the server a message is displayed to the user. This message is shown as an alert dialog like the one shown in Fig. 44. This alert dialog has rounded corners, as Material design guidelines recommend (see section 2.5).



Sorry

No response from server, please check your connection or try again later

OK

Fig. 44 No response error alert

The main reasons that can cause a lack of response are: A problem with the user's internet connection or the server being down for some reason.

3.3 main()

main () function is the app base and it starts the app. This function is the first executed and it manages the checkout of existing user credentials, the device orientation settings, the internationalization settings, and sets the root widget to MaterialApp and also sets the first screen to be displayed to the user depending on whether the Smartphone has the credentials saved from previous sessions or not.The main() function first call is to:

```
WidgetsFlutterBinding.ensureInitialized();
```

This ensures the service binding is working before it is accessed. This is necessary because the first thing the app does is checking it has stored the user credentials. If so, those credentials are used to automatically log in and the map page is loaded, but if not, the login page is displayed.

To store and load the user credentials it is used the plugin shared_preferences developed by flutter.dev. This plugin provides a persistent store for simple data. It was chosen because it is a certified plugin and moreover it is developed by the flutter team. It is also recommended in the flutter documentation found in [134]. However, this package stores the data in plain text, which is not recommended for password storing. Thus, obviously a future line of the project is to modify this and safely store the user data using encryption. It has already been studied the way to do this. The flutter_keychain package uses KeyChain (iOS) and KeyStore (Android) for secure storing. Package documentation can be found at [135]. This package should be soon implemented in the app.

As mentioned before if there are no stored user credentials the login page is displayed, but if there are stored credentials a procedure is carried out before displaying the map page. The function in charge of checking and getting the credentials is loadUserCredentials(). This function is defined in the auxiliar.dart file. Once the credentials are gotten it is checked if the user has admin permissions or not. After that, all the necessary topics are subscribed, which, as mentioned in the previous section, are not the same for common users and admins. Personal topics like username/app are defined at this point, as these topics contain the username on its name. Finally, this function informs the server that a new user is going to be logged, sending the user credentials. This has to be done because the server and the broker are not the same. The broker is aware of the new user logged as it is necessary to authenticate into the MQTT broker to be able to send or receive messages, but the server is not aware because the server and the MQTT broker are different programs. The server uses the MQTT broker the same way the app does. The only difference between the app and the server is that, at the moment, the server and the broker run in the same machine, but this could change.

The main function also sets the orientation:

```
await SystemChrome.setPreferredOrientations([
  DeviceOrientation.portraitUp,
]);
```

By default, the app changes the layout as the device orientation changes. Setting the orientation to “portrait Up” prevents the app from turning the layout, thus it will always display the app with the vertical layout regardless of the device orientation.

Finally, it starts the app by calling:

```
runApp(MyApp());
```

The MyApp class builds the root widget of the app. This widget returns a MaterialApp widget. This widget is necessary to use some Material widgets. Then this widget defines the localization delegates. Those are necessary to internationalize the app. Our app currently supports English and Spanish for the user interaction (admin pages are only available in English). Language is automatically switched depending on the phone’s settings language. This is possible because of flutter_localizations package. This package and how to internationalize an app are explained in the flutter docs: [136].

The MaterialApp widget contains the next lines where it defines the localization delegate and the supported languages as explained before.

```
localizationsDelegates: [
  const TwizyLocalizationsDelegate(),
  GlobalMaterialLocalizations.delegate,
  GlobalWidgetsLocalizations.delegate,
],
supportedLocales: [
  const Locale('en'), // English
  const Locale('es'), // Spanish
],
```

And there is another dart file localizations.dart that contains the class TwizyLocalizations. This class defines the strings translations and the getters to use them.

```
class TwizyLocalizations {
  TwizyLocalizations(this.locale);

  final Locale locale;

  static TwizyLocalizations of(BuildContext context) {
    return Localizations.of<TwizyLocalizations>(context, TwizyLocalizations);
  }

  static Map<String, Map<String, String>> _localizedValues = {
    'en': {
      'title': 'Hello World',
    },
    'es': {
      'title': 'Hola Mundo',
    },
  };

  String get title {
    return _localizedValues[locale.languageCode]['title'];
  }
```

Whenever it is desired to use an internationalized string, it has to be done the way:

```
TwizyLocalizations.of(context).title
```

This string will automatically show “Hello world” or “Hola mundo” depending on the phone set language.

The root widget defines the app title, the theme, and the home screen. In our case the default home will vary depending on whether there are found user credentials or not, or if the user has admin permissions or not.

The theme primary color chosen has been a light green. The green color is commonly associated with ecology. More aesthetical details can be found in the Final Degree Thesis report of the TwizyLine member Mario Martin [28].

In Fig. 45 it can be seen a simplification of the app widget tree. It can be observed that MaterialApp is the root widget of the app. The top widgets of every screen are painted red. AdminIndexedStack widget can show four different screens, each screen top widget is painted purple. The ErrorLog screen is the first displayed when an admin logs in.

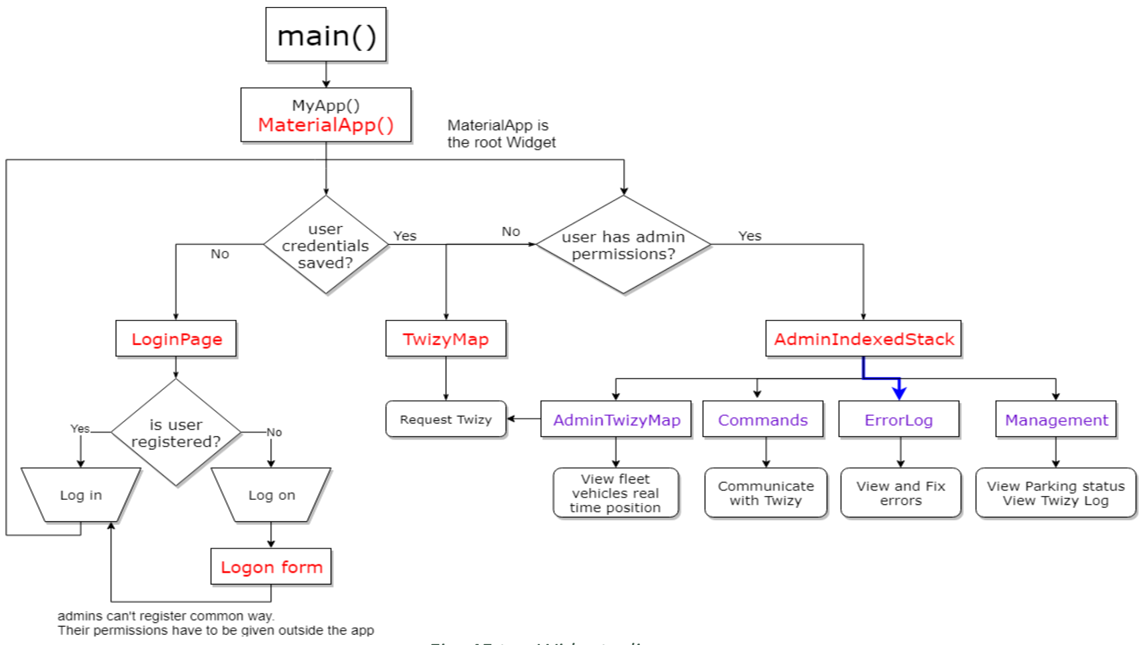


Fig. 45 top Widgets diagram

At this point the nonvisual files of the app have been explained. Now the app screens will be described as well as the functions they have. The screens can be divided into three segments: Common screens, user screen and admin screens. They will be explained in this order.

3.4 Login page

The login page is the first screen displayed to the user when they open the app. It manages the login process and can redirect the user to the logon form too. This screen also implements form validation for the username and password fields.

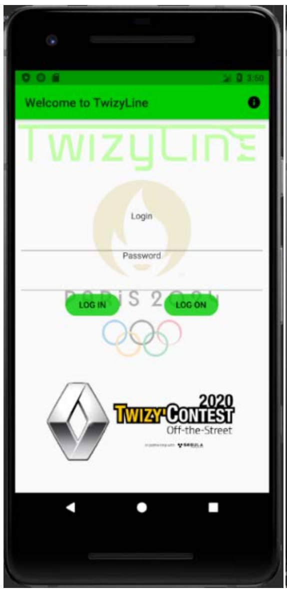


Fig. 46 Login screen view

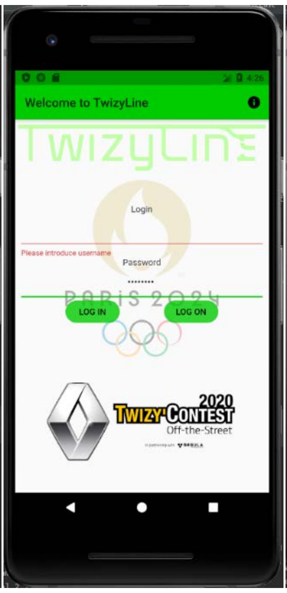


Fig. 47 Login screen view 2

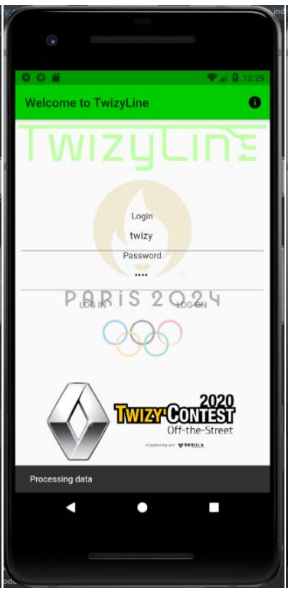


Fig. 48 Login screen view 3

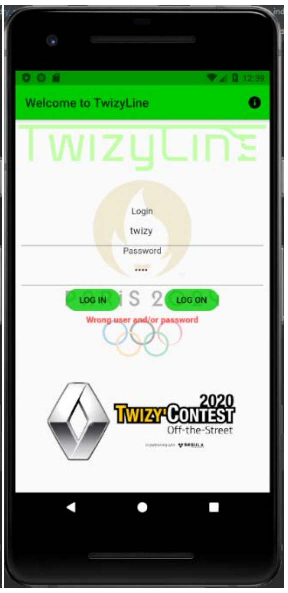


Fig. 49 Login screen view 4

The login screen, showed in Fig. 46 is displayed to the user the first time the app is opened. This screen displays TwizyLine logo, Paris 2024 logo, Renault logo, and Twizy Contest Logo. The buttons are rounded, following material guidelines, to give a better appearance. Aesthetic is important here, as this is the first screen users would see. That is why the brands logos are placed here too. This screen is not scrollable, but it is designed to fit any device.

There are two form fields, the username and the password field, and two buttons, Log in and Log on. Log in button sends the server the request to log in. The app first checks if the form fields are filled or not. Also, the fields length is limited.

If the users try to submit and they have not filled the fields an error is shown. This can be seen in Fig. 47. Also, in this picture, it can be seen that the password input is obscured. When the login button is pressed a *SnackBar (*a *SnackBar* is a message displayed from the bottom of the screen inside a colored container. They are used to give the user some short information) shows up saying "Processing data" and the buttons are disabled so they cannot be pressed. To show that those buttons are disabled the color changes to transparent. This can be seen in Fig. 48. If username and/or password are wrong a message is displayed. This can be seen in Fig. 49.

After the user presses the log in button the app sends the username and the password through MQTT to the server. This is a double step authentication, as the first step is authenticating to the MQTT broker, if this authentication did not work the app would not even be able to send those credentials to the server.

Once the credentials are sent, the server checks if the user has admin permissions or not. Then, the app needs to wait for the server response, but as mentioned before, Dart is single threaded. There is no clear way to wait for a response. Futures allow to wait for a function to be complete without freezing the main thread. Although when listening to a stream inside a function, once the stream receives a message it is considered listened, and the function completes. It was necessary to keep that function running while waiting for the expected message. To do this, it was decided to use a while loop, but it was necessary to keep the main thread running, otherwise the message would never be received. And only once the desired message is received, the function can return a value. It was also added a timer so if no message is received the function returns a value anyways. Here is the code:

```dart
int ifAdmin = await checkIfAdmin();
Future<int> checkIfAdmin() async {
  bool timeout = false;
  int admin = -1;
  Timer timer = Timer(Duration(seconds: 10), () {
    timeout = true;
    // here I can't use return
  });

  StreamSubscription<ReceivedMessages> streamLoginSubscription =
  streamUser.listen((value) {
    List<String> message = value.payload.split('/');
    if (message[0] == 'ADMIN') {
      timer.cancel();
      admin = int.parse(message[1]);
      timeout = true;
      // here I can't use return
    }
  });

  while (!timeout) {
    await asyncSleep(100);
  }
  await streamLoginSubscription.cancel();
  return admin;
}
```

The desired message is either ADMIN/0 or ADMIN/1. This function will return 1 or 0 and then the app will process this value and decide what to do. If the timer expires then it returns -1. And the app will process the error.

The asyncSleep function stops the current function but does not block the thread. This is thanks to futures. This method allows to keep listening the stream and return when the desired message is received. Here is its definition:

```
Future<void> asyncSleep(int milliseconds) =>
    Future<void>.delayed(Duration(
      milliseconds: milliseconds,
    ));
```

When the user successfully logs in, the user credentials are saved. Once these credentials are saved the login page is not displayed anymore unless the user logs out. The credentials load is explained in main(). The function in charge of saving the credentials is saveUserCredentials(). This function uses the shared_preferences package like loadUserCredentials. It also unsubscribes the app from the "log" topic and defines the personal topic names.

At the top right it can be seen an info button. This button should redirect to our website where information about TwizyLine service could be found. This is still a future line though. The other button is the log on button. This button redirects to the log on form.

3.5 Logon page

The logon screen, shown in Fig. 50, displays a form so the user can register into the service. It asks the user for its name and surname, employment, username, email, and password. It can be used to register a new user to the service. The fields implement form validation and length limitation.

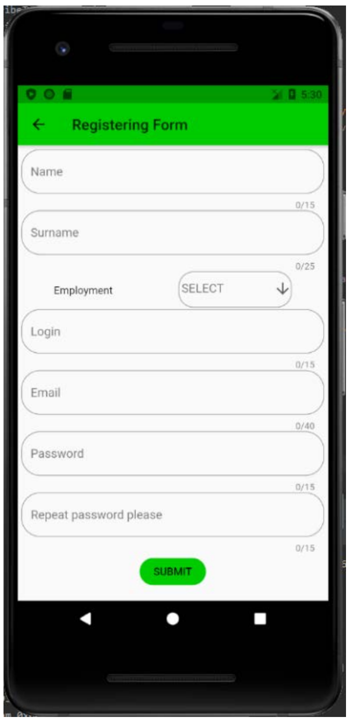


Fig. 50 Registering Form

This is a scrollable screen to make the user able to fill all the fields. The fields borders are rounded as well as the button.

The employment is asked through a dropdown list. The options are athlete, journalist, staff, or visitor. This information could be used to give different priorities to users when requesting a vehicle for example.

When the submit button is pressed, the app checks that all fields are filled. The username must be unique, and the email field checks if it contains an '@' character. The password is required to be introduced twice and the app makes sure the two fields have the same information. Also, all the fields have length limitation.

When users fill one of the fields, the keyboard "OK" button sends the user to the next field. This provides a more fluent experience.

To check that the username is available the app connects to the server using a generic user (obviously this user does not have admin permissions for security reasons), then it sends the user information. If this information is valid, the app gets a confirm message from the server. It logs out from the generic account and automatically logs in as the new user. The server side of this procedure is explained in the Final Degree Thesis report of the TwizyLine member Samuel Pilar [26].

When the app sends the log on parameters (name surname, employment…), it has to wait for the server response. This scenario is very close to the one explained in the previous section, so this function is very similar to the checkIfAdmin() function. The received message will be either "Confirm" or "Deny". If the message is "Deny" it will be followed by an error code that will help to explain the user why those values are not valid to register. For instance, if the username is already in use.

Admin users cannot register though the app. This special kind of users have to be given permissions directly on the server.

One of the future lines of the project is to request users to scan their driving license to ensure users can drive the vehicles of the fleet.

Login and logon pages are the two screens common for both users and admins. Even though admins cannot register through the logon page they can access to it. The map page is the only page available for users (In addition to the login and logon pages) , and it is explained in the next section.

3.6 Twizy Map page

This is the main screen of the app on the user side. This screen is displayed to the user once they are logged. Moreover, when the user has already logged in at least once, this screen is the first displayed when the app starts.

The first time the user opens the app, it requests location permissions, as it is shown in Fig. 51 While the map is loaded, it is displayed a gif "drawing" TwizyLine's slogan as shown in Fig. 52. Once the map is built, the app sends the server a request to get the parking positions so it can place their markers. While the app gets the response, it is displayed a loading icon, as it is shown in Fig. 53.

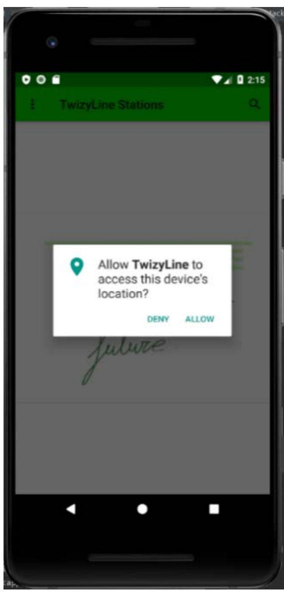


Fig. 51 Asking for permissions

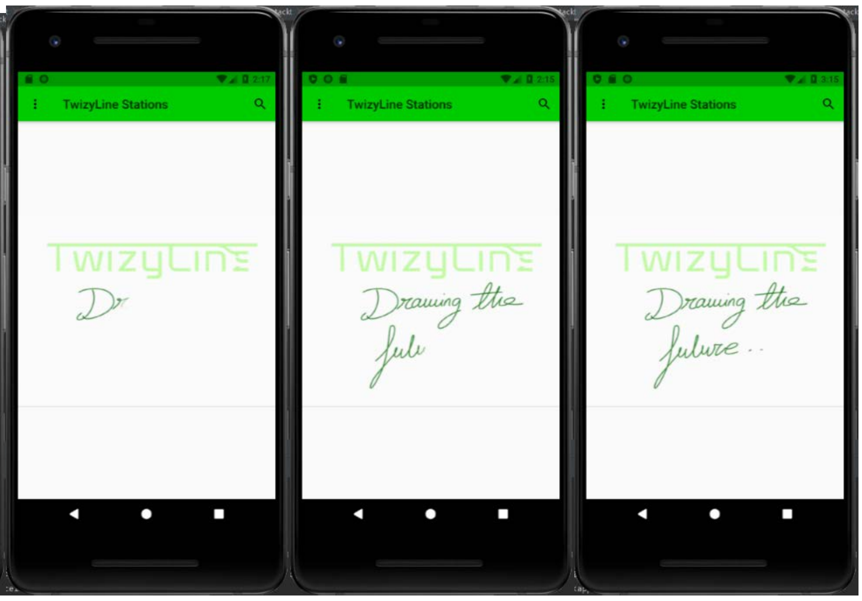


Fig. 52 Loading Map

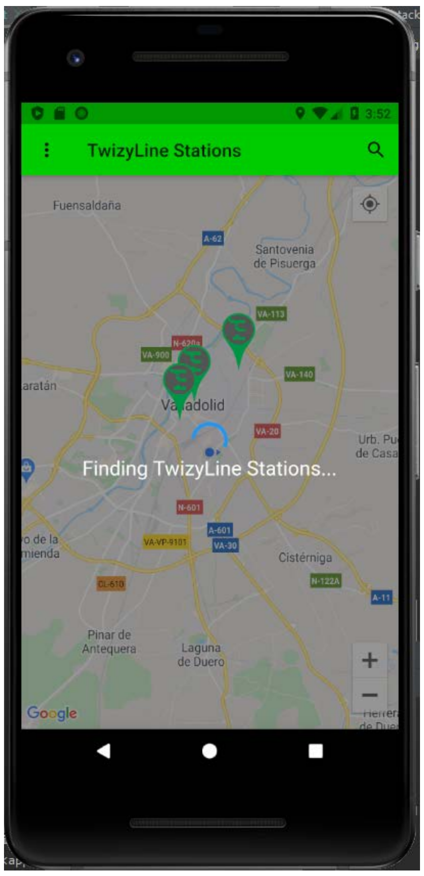


Fig. 53 Finding TwizyLine Stations view

Once the server sends the first parking information, the user can see and interact with the map. This map widget shown in Fig. 54 is part of the google_maps_flutter plugin package. It is a fully functional map view, similar to the Google Maps App but with some restrictions. For instance, this map cannot display routes or show commerce information touching over them. But it is possible to navigate through it scrolling with the fingers, zoom in or out either with gestures or with the buttons located at the right bottom of the screen, and it centers the map view on users location when the right top button is pressed. The map firstly loads centered on the user location as well. The markers are customized, displaying TwizyLine's logo.

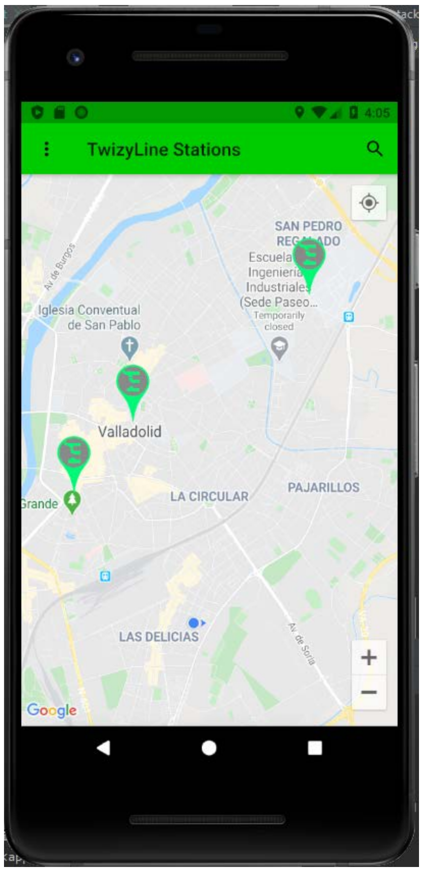


Fig. 54 Map and Markers view

The app takes the right marker size depending on the device resolution. There are different resolutions for the same image, each of them saved in a different folder: assets/1.0x , assets2.0x, assets3.0x, etc. The app gets the device Pixel Ratio, rounds it, and uses the closest image. This allows to see the markers almost with the same size on different resolution screens. This is explained in [137], and the code that implements this selection is shown below:

```
    //First get the device pixel ratio
    double dpi = MediaQuery.of(context).devicePixelRatio;
    // With this ratio the marker image is properly scaled
    twizilineIcon = await BitmapDescriptor.fromAssetImage(
        ImageConfiguration(devicePixelRatio: dpi),
        'assets/twizyline_marker.png');
```

When the users tap on any of the markers, the screen centers on that marker, and it displays a pill with the parking name, the free spots and the available Twizys in it (see Fig. 55).

The pill also displays the Renault icon and a Twizy icon. To request a Twizy from a selected parking area, users must tap the pill. The app will then display a loading icon while the petition is processed (see Fig. 56).

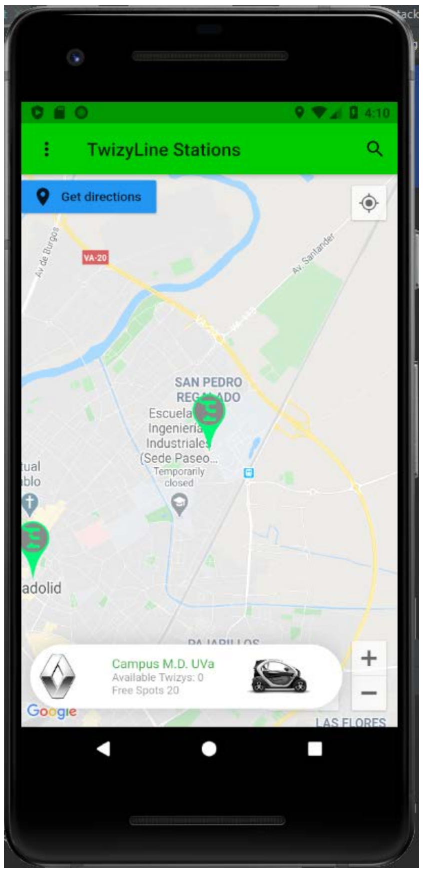


Fig. 55 Marker tapped view

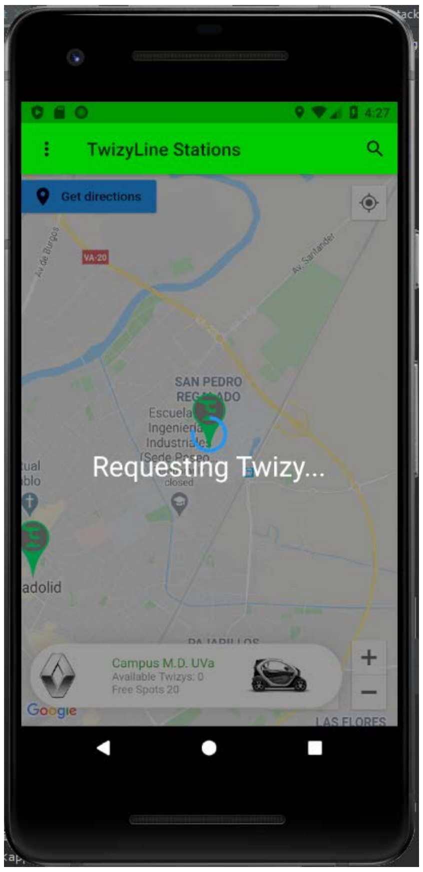
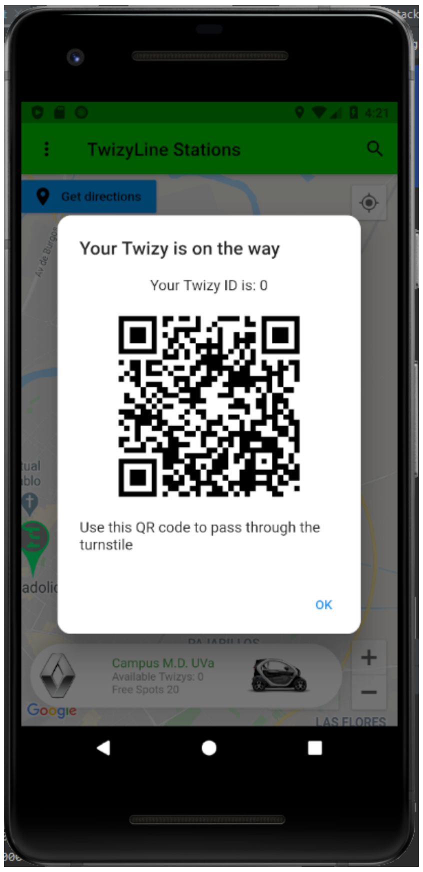


Fig. 56 Requesting Twizy

If the request is successfully processed, then the server will assign a Twizy to the user and will generate a QR code to pass through the turnstile. The app will show the user its assigned Twizy ID (this number will be painted on the car) and the QR code (see Fig. 57). You might have noted that the "TWIZY OUT" message found in ANNEX 1 does not contain information about the QR code. This functionality is still being built, so in the future the server will generate the unique QR code for each Twizy assignment and send it to the user. As for now the app always displays a certain QR code to show how it would look like.



Your Twizy is on the way

Your Twizy ID is: 0

Use this QR code to pass through the turnstile

Fig. 57 Twizy assigned

If for some reason the server cannot give the user a Twizy or there is any trouble with the communication, the app will inform the user (see Fig. 58).

Tapping on a marker, also produces the "Get directions" button to show up. If tapped, this button opens Google Maps on Android or Maps on iOS and gives the user directions to get to the selected parking area (see Fig. 59).

These are the actions that a normal user can perform in the map screen. These actions come with a set of actions from the application itself, that are going to be describe in following lines. Starting from the beginning, when the map is built, the app sends a message to the server requesting the list of parking sites, their status and location. There is a timer to wait for this response. Once the timer expires, the app unsubscribes from the topic where this list is sent. If no parking information has been received the app shows an alert error (see Fig. 60).

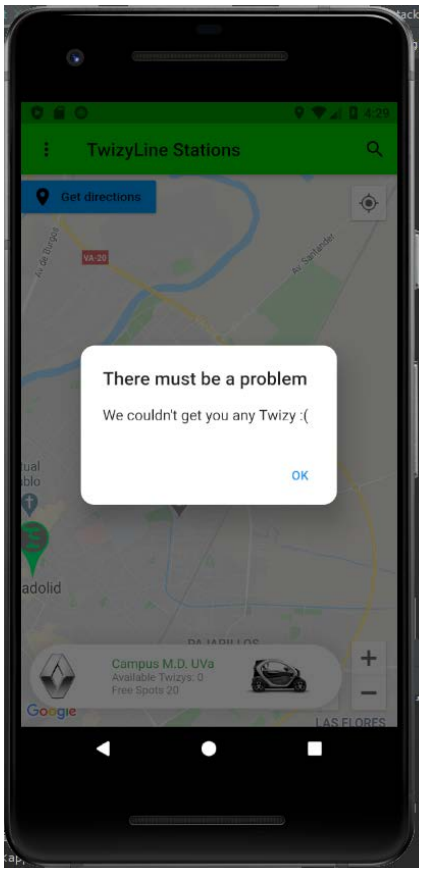


Fig. 58 Couldn't get a Twizy

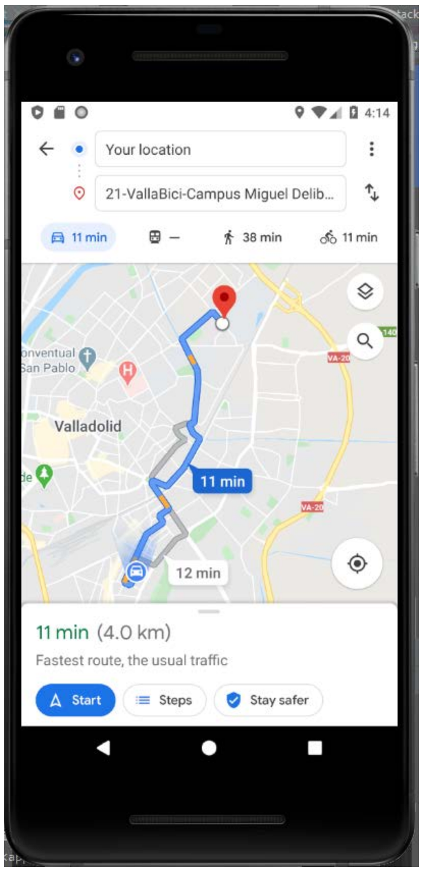
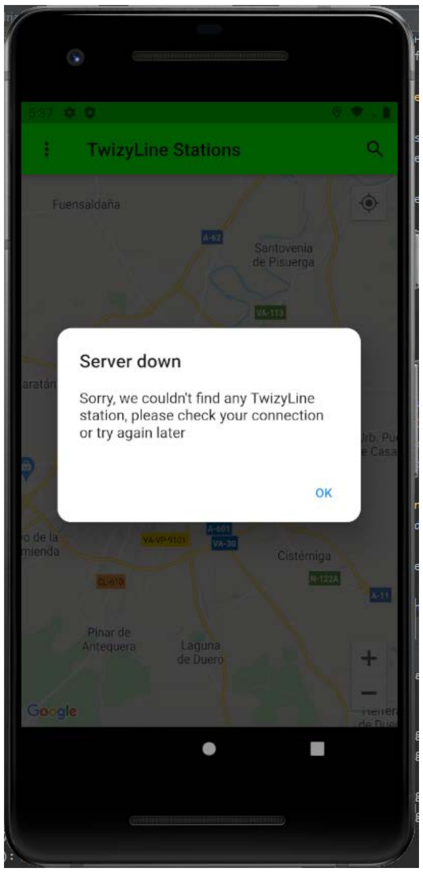


Fig. 59 Google Maps directions



Server down

Sorry, we couldn't find any TwizyLine station, please check your connection or try again later

OK

Fig. 60 No parking sites found error

But if at least one parking information is received, it will be displayed on the map. When the timer expires, the app will unsubscribe from the topic anyways, but no error will be displayed.

The map widget is part of the google_maps_flutter package. The map has a list of markers that can be displayed. A marker is an item displayed on the map, which nowadays is sort of standardized for the most of the app maps as it is shown in Fig. 61. However, in our app, a custom marker has been designed as it is shown in Fig. 62. This custom marker reinforces the corporative image.

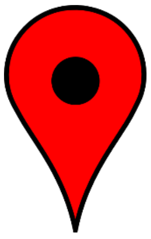


Fig. 61 Typical Marker in maps apps

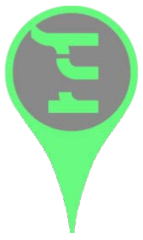


Fig. 62 TwizyLine custom Marker

Marker class keeps the marker Id, position, and the behaviour on tap action. There is a function listening to the stream where this information is sent. When this information arrives, the parking id and location are added to a list of markers, the _markers list. Other information received, like the available Twizys or free spots, is stored in a list of PinInformation objects, the TwizyLineStations List. PinInformation is a class created for the app to hold the parking information. Each Marker object is linked to its PinInformation object.

```
streamUserSubscription = streamUser.listen((value) {
  List<String> message;
  message = value.payload.split("/");
  if (message[0] == 'NEW-PARK') {
    final index = twizylineStations.length;
    setState(() {
      twizylineStations.add(
        PinInformation(
          parkingID: message[1],
          locationName: message[2],
          location:
              LatLng(double.parse(message[3]), double.parse(message[4])),
          availableTwizy: int.parse(message[5]),
          freeSpots: int.parse(message[6]),
          pinPath: 'assets/twizy_icon.png',
          avatarPath: 'assets/renault_icon.png',
          labelColor: Colors.green,
          index: twizylineStations.length,
        ),
      );
      _markers.add(
        Marker(
            markerId: MarkerId(message[1]),
            position:
                LatLng(double.parse(message[3]), double.parse(message[4])),
            // If set to true it doesn't show the two buttons at the bottom
            // right of the screen that redirect to Google Maps App, but it
            // wouldn't center the map on the marker on tap either. The
            // routing button has been reimplemented as it is overlapped by
            // infoPill but the center map function hasn't been reimplemented
            // so will keep it false for now.
            consumeTapEvents: false,
            onTap: () {
              setState(() {
                currentlySelectedPin = twizylineStations[index];
                pinPillPosition = 0;
                _routeButtonVisible = true;
              });
            },
            icon: twizilineIcon),
      );
      _noMarkersYet = false;
    });
  }
});
```

The PinInformation object has an index parameter. This parameter is used to find the object in the TwizyLineStations list. As the index parameter is final (every variable assigned inside a stream.listen is final. Its value is stored and not its memory direction), every Marker object stores a different index value on its onTap method. This is the way markers and their info pill are linked.

The same function also listens to another stream where the parking sites information is refreshed. The server sends a periodic message with each parking id, available Twizys and free spots. The app then checks its list of PinInformation objects and for every parking id updates its status. The code that performs this refresh action is shown below:

```
    streamAppSubscription = streamApp.listen((value) {
      List<String> message = value.payload.split('/');
      if (message[0] == 'INFO-PARK') {
        setState(() {
          for (PinInformation pin in twizylineStations) {
            if (message[1] == pin.parkingID) {
              pin.availableTwizy = int.parse(message[2]);
              pin.freeSpots = int.parse(message[3]);
            }
          }
        });
//        print('INFO PARKING ACTUALIZADA');
      }
    });
```

This makes the information displayed on the app reliable. As this periodic message is send every few seconds.

When the user requests a Twizy tapping the pill, the app sends a message to the server. This message contains the id from the parking where the Twizy is being requested. The server answers with the Twizy Id and the QR code.

3.7 Internationalization

As I mentioned before, the users interface is internationalized. Login, Logon and Map screens can display all their text in both English and Spanish. As well as alerts and errors. The internationalization is shown in figures from Fig. 63 to Fig. 70.

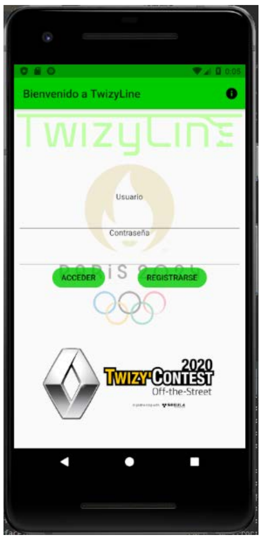


Fig. 63 Login screen in Spanish

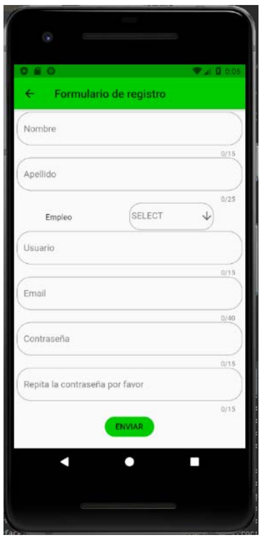


Fig. 64 Logon form in Spanish

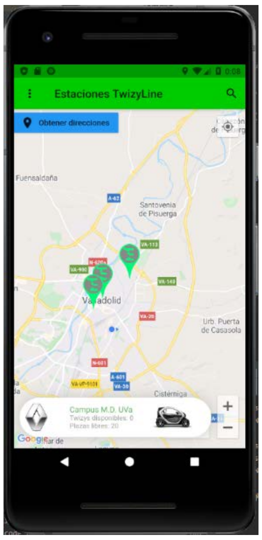
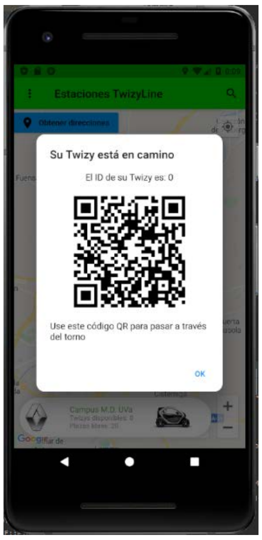


Fig. 65 Map view in Spanish

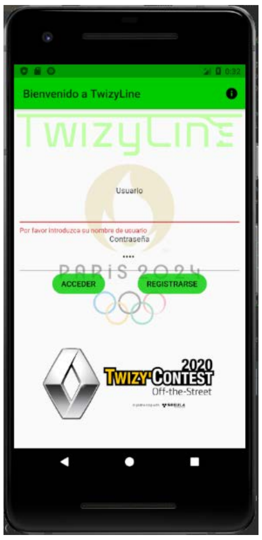


Su Twizy está en camino

El ID de su Twizy es: 0

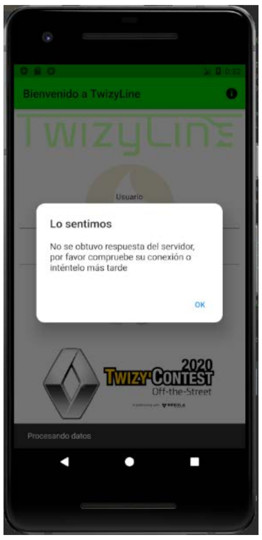
Use este código QR para pasar a través del torno

Fig. 66 Assigned Twizy view in Spanish



Por favor introduzca su nombre de usuario

Fig. 67 Validation error message in Spanish

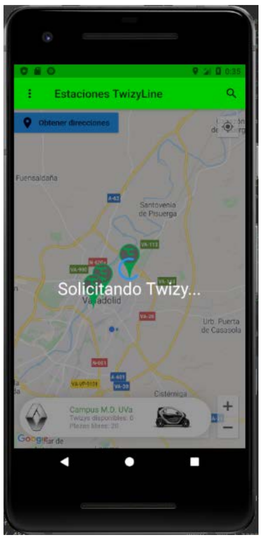


Lo sentimos

No se obtuvo respuesta del servidor, por favor compruebe su conexión o inténtelo más tarde

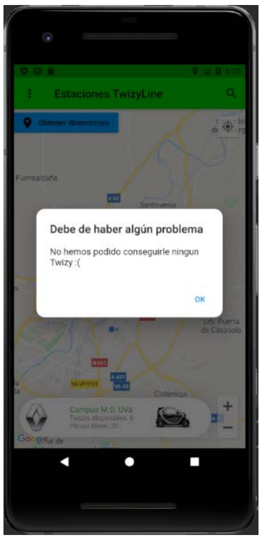
OK

Fig. 68 Server connection error in Spanish



Solicitando Twizy...

Fig. 69 Loading message in Spanish



Debe de haber algún problema

No hemos podido conseguirle ningun Twizy :(

OK

Fig. 70 Twizy assignment error in Spanish

3.8 Admin app resources

At this point we have seen the app as a mechanism to get access to TwizyLine service. But, the app is a valuable tool for admins too. This would improve the service quality, making TwizyLine easier to maintain, manage and improve. There are four different tools for admins:

- The map screen: a common user map screen plus all the fleet vehicles real time position.
- The error log screen: a list of active errors that can be fixed or dismissed
- The manage screen: a table to see the state of a determined parking area or a certain vehicle log.
- The commands screen: an interface to directly send and receive commands from a Twizy.

These tools are also useful for developers to test the system.

The 4 screens are part of an IndexedStack Widget. This widget preserves the state of all its children. It only displays one of its children at a time. It uses an index parameter to choose which child is displayed, as it can be seen in the following lines of code:

```
IndexedStack(
  index: _index,
  children: <Widget>[
    adminTwizyMap(),
    adminErrorLog(),
    adminManagement(),
    adminCommands(),
  ],
),
```

Preserving the state in Flutter is very important and sometimes is not trivial. Most of the app state is managed by the framework, like textures. But other aspects have to be managed by the developer. The state of a stateful widget contains all the values that can change. Those are all the data needed in order to rebuild the UI at any moment. For instance, the error log state contains the information about how many errors there are, and all the info displayed on every card. If this state was not preserved, every time this screen is accessed it would be empty, and we would have to request to the server the list of errors every time. Server requests usually take time, and that worsens the user experience. Preserving the state allows to always have this information available. It also allows, for instance, displaying a new error card when it is broadcasted by the server. Even if it was received while we were on a different screen.

The state managed by the developer can be divided in two conceptual types. Ephemeral state and app state.

The ephemeral state barely needs to be accessed by other parts of the widget tree. It is contained inside a single widget. To manage this state, we will only need a Stateful widget. The state variables can be modified using the setState() method. This method makes the framework schedule a build for this State object. Examples of ephemeral state are the current selected tab in a Bottom navigation bar or the current page in a PageView.

When the state has to be shared across different parts of the app and kept between user sessions then it is app state. There are several ways to manage the state in Flutter, like Redux, BLoC/Rx, Inherited model or Scoped model. A list of state management options can be found in the reference [138]The simplest approach is probably the provider package. This package was at first implemented on this app, but then some troubles managing the context appeared and so it was used directly the IndexedStack widget to manage all the state information, which solved all the problems found. Those problems were related to the fact that with the provider package it becomes easier to lift the state of some objects, but it is not so simple to lift the state of the context itself, so a new context is generated by the framework every time the user goes out the screen and back in, this makes impossible some tasks like displaying *SnackBars*. This is because the *SnackBar* depends on the context currently displayed, but the order to display the *SnackBar* is sent when the user taps a button that is displayed by the error dialog that is displayed when the user taps on one of the error cards. Because the cards are created when the error is generated, but the *SnackBar* has to be displayed when the user taps the button, if the user had gone out and back in the scree between those two events, the context would have changed as it is not lifted by the provider package, therefore the context would not match and the *SnackBar* could not be displayed [139] [140].

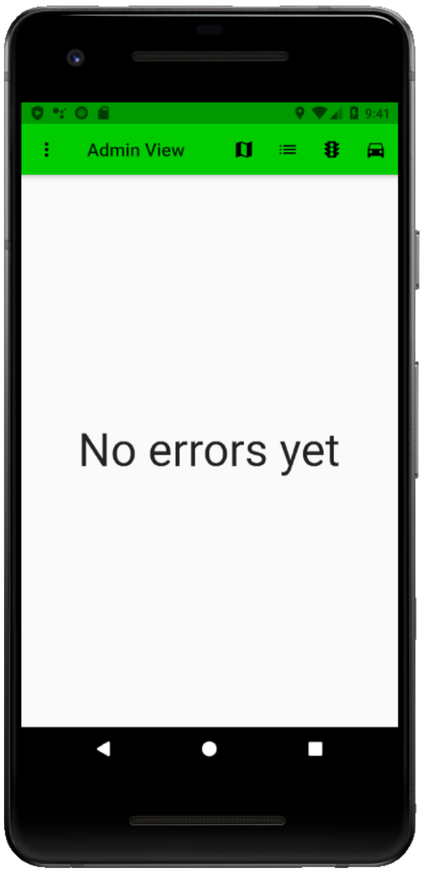


Fig. 71 Error Log view

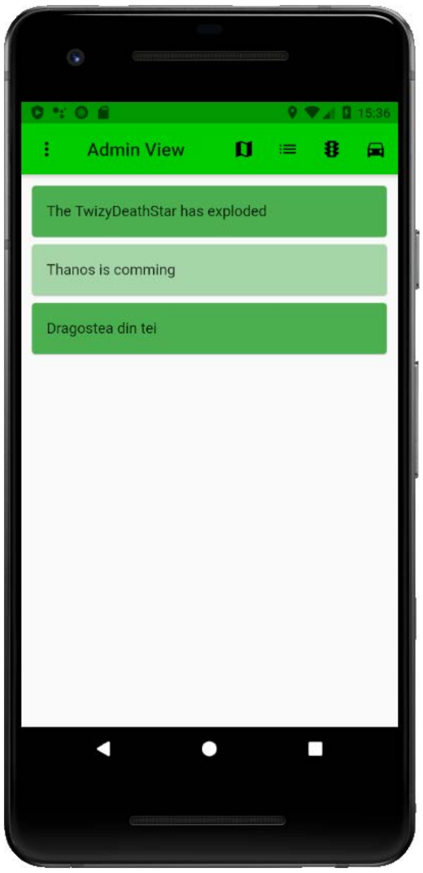
To switch between the four screens, it was decided to use the upper bar. This bar displays four different icons, each one drives to one of the four admin screens. This can be seen in Fig. 71. The map icon drives to the Map page. The list icon drives to the error log page, this page is the first displayed when an admin logs in. The traffic light icon drives to the management page. And the car icon drives to the commands page.

Every time an icon from the bar is pressed, not only does the index parameter change, but it also pauses and resumes the different streams used in the screens. So only the necessary streams are active on each screen. Also, when the widget is disposed (this means that the widget will not be displayed anymore and it is removed from the tree), the app cancels all the streams. This is important in order to lighten up the work the app has to do, but also to reduce the size of the app. If streams are not canceled, the information they keep could remain, making the app take up more space on disk.

The four screens of the admin mode are described in the next sections.

3.8.1 Error log

This page is displayed first once the admin logs in. When the admin logs in the app sends the server a request for the errors list. The server has a mailbox with the current errors, and it sends it to the app as a response to this request. If the server finds any other error, it will broadcast it to all the active admins, and the app will automatically display the new error. Every error is displayed in its own card. The cards alternate colors to improve visibility as shown in Fig. 75. Initially, the card only shows the error title, and once it is pressed, it displays more information. When there are no errors to fix, it shows a message “No errors yet” as shown in Fig. 71. The errors that can be received are shown in ANNEX 2.

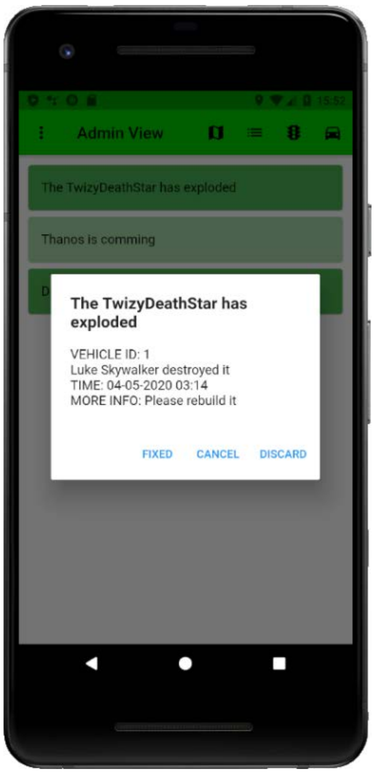


The TwizyDeathStar has exploded

Thanos is comming

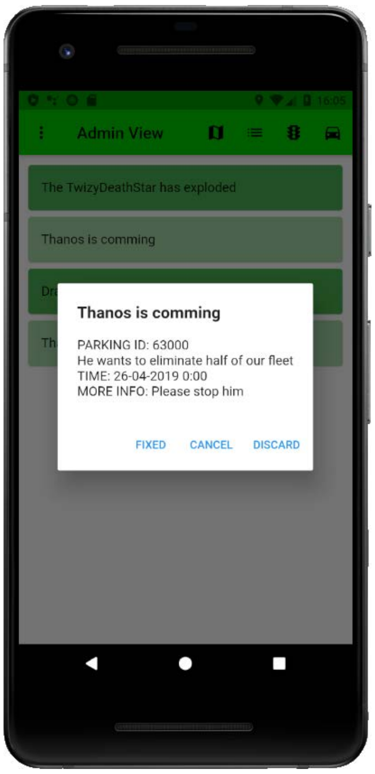
Dragostea din tei

Fig. 75 Error Log screen with errors



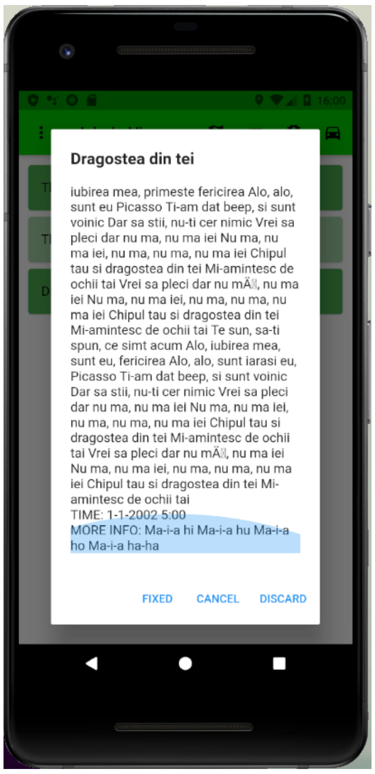
The TwizyDeathStar has exploded

VEHICLE ID: 1
Luke Skywalker destroyed it
TIME: 04-05-2020 03:14
MORE INFO: Please rebuild it



Thanos is comming

PARKING ID: 63000
He wants to eliminate half of our fleet
TIME: 26-04-2019 0:00
MORE INFO: Please stop him



Dragostea din tei

iubirea mea, primeste fericirea Alo, alo, sunt eu Picasso Ti-am dat beep, si sunt voinic Dar sa stii, nu-ti cer nimic Vrei sa pleci dar nu ma, nu ma iei Nu ma, nu ma iei, nu ma, nu ma, nu ma iei Chipul tau si dragostea din tei Mi-amintesc de ochii tai Vrei sa pleci dar nu mĂ, nu ma iei Nu ma, nu ma iei, nu ma, nu ma, nu ma iei Chipul tau si dragostea din tei Mi-amintesc de ochii tai Te sun, sa-ti spun, ce simt acum Alo, iubirea mea, sunt eu, fericirea Alo, alo, sunt iarasi eu, Picasso Ti-am dat beep, si sunt voinic Dar sa stii, nu-ti cer nimic Vrei sa pleci dar nu ma, nu ma iei Nu ma, nu ma iei, nu ma, nu ma, nu ma iei Chipul tau si dragostea din tei Mi-amintesc de ochii tai Vrei sa pleci dar nu mĂ, nu ma iei Nu ma, nu ma iei, nu ma, nu ma, nu ma iei Chipul tau si dragostea din tei Mi-amintesc de ochii tai
TIME: 1-1-2002 5:00
MORE INFO: Ma-i-a hi Ma-i-a hu Ma-i-a ho Ma-i-a ha-ha

Fig. 72 Vehicle error *Fig. 73 Parking error* *Fig. 74 Scrollable error*

Figures Fig. 72, Fig. 73, and Fig. 74 show how the information about errors is displayed. An alert dialog shows up with the error title, the vehicle or parking id, the date when the error happened

and some more useful info for the admin. The app automatically detects if the error is from a vehicle (Fig. 74a) or from a parking (Fig. 74b) depending on the id. The alert dialog is scrollable, so long messages can be seen too, as it is shown in Fig. 74c. The alert dialog has 3 buttons. The cancel button behaves just like tapping outside the alert dialog, it just hides the alert dialog. The discard button deletes the error card from the screen, but the server still keeps the error, so next time the admin opens the app this error will show up on the list again. The fixed button also deletes the error card from the list, but it sends a message to the server too. This message informs the server that the error has been fixed, so it can delete it from the mailbox. Then the server will store in the database the error fixed and the admin who solved it.
As IndexedStack keeps the state, the list will keep its elements while the admin does not close the app.

3.8.2 Map

The admin map is the same as the common user map. The only difference is that admin map displays the fleet vehicles position on real time. The vehicles markers show the vehicle ID when tapped. That kind of markers are shown in Fig. 76.

Fig. 76 Twizy Marker

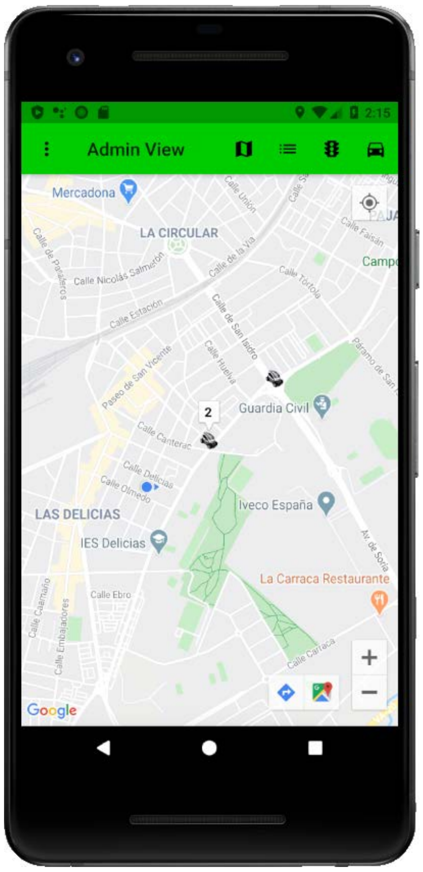
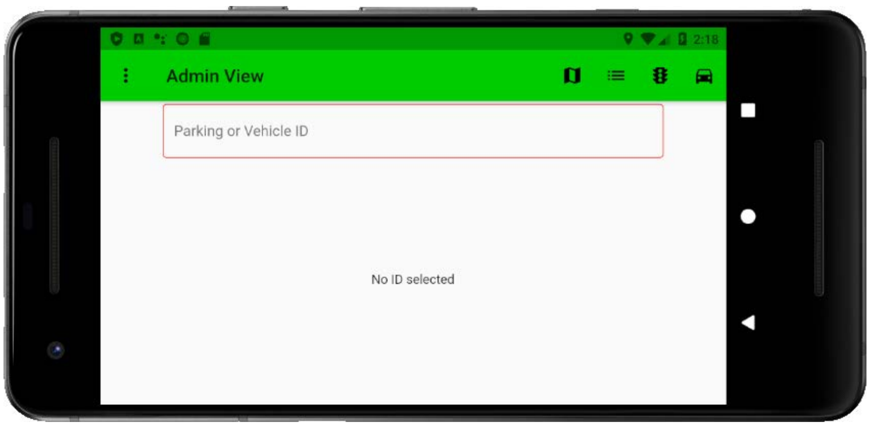


Fig. 77 Admin Map Twizy location

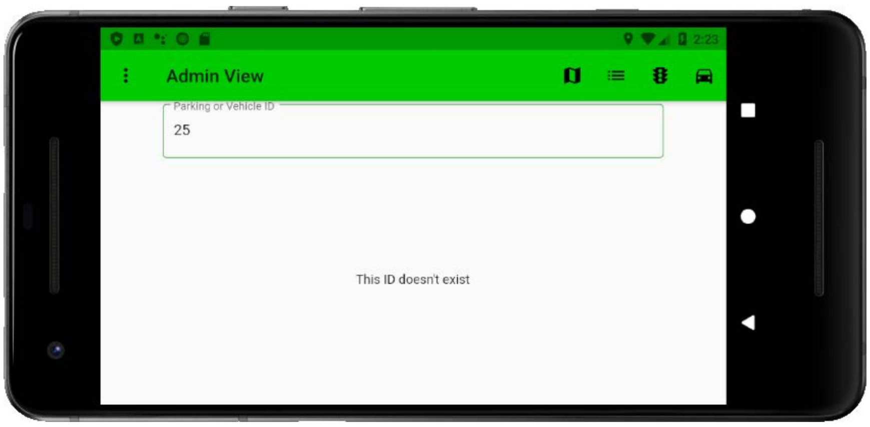
3.8.3 Management

Management screen can display a table to see a parking status or a vehicle log. The Fig. 78 shows a search button where the admin can insert the desired ID. While no ID is introduced the search field frame looks red and the screen says: "No ID selected". If the ID is out of the valid range or it is not assigned to any known vehicle or parking, the app will inform the user as shown in Fig. 79.



No ID selected

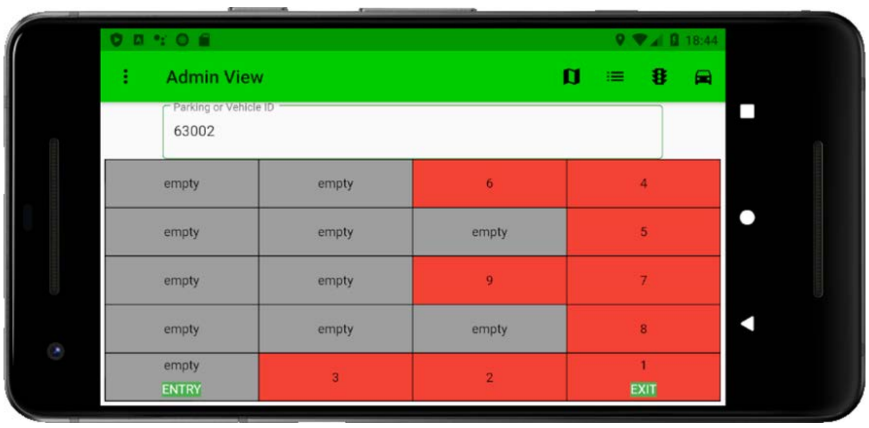
Fig. 78 Management screen first view



This ID doesn't exist

Fig. 79 Management screen non-existent ID

If the ID matches a parking ID, then the app sends a message to the server requesting the info about the desired parking. The app displays the received information through a table, which is organized so that it represents the parking. So, it has as many cells as parking spots. The app displays on red the spots where there is a Twizy, and it shows the ID of the vehicle filling the spot. If the spot is empty the cell appears on grey. The table also shows where the entrance and exit are regarding the grid of the parking slots. This gives the admin a clear image of the parking disposition and status. Fig. 80 shows the screen that the app would display representing the parking with ID 63002, and Fig. 81 shows a simulation of how this parking would look like in the real world, proving that the app actually represents the parking status.



63002

empty	empty	6	4
empty	empty	empty	5
empty	empty	9	7
empty	empty	empty	8
empty ENTRY	3	2	1 EXIT

Fig. 80 Management screen parking representation

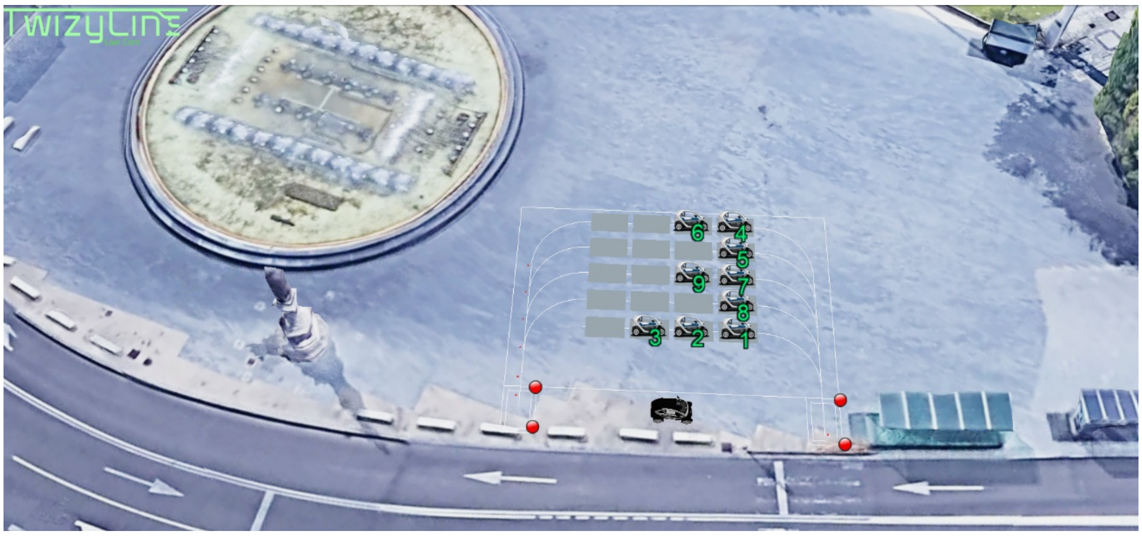
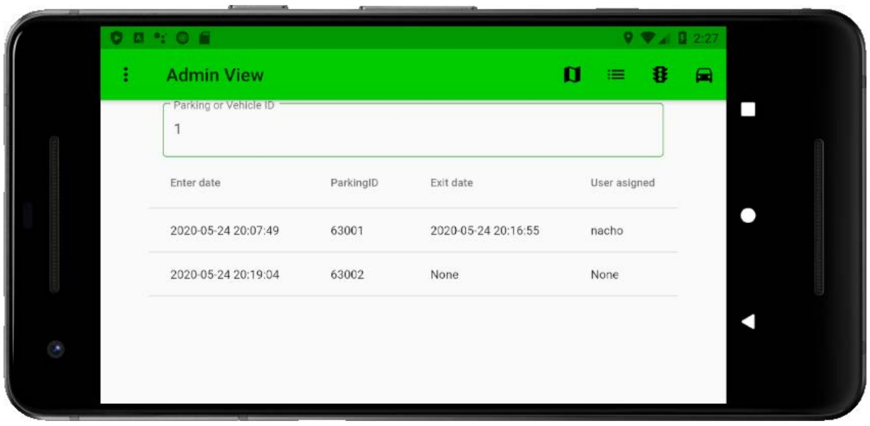


Fig. 81 Parking status

This widget is not scrollable, but it resizes the table to fit the screen (this would change if parkings grow making it impossible to display the whole parking without scrolling). In the following code the app measures the available screen and adapts the table to it. To do so, the app gets from the server the necessary data to represent the parking, and calculates the available space for every cell. This code also checks if a cell is empty or not to paint it on the right color and fill it with the corresponding vehicle ID or the "ENTRY" or "EXIT" message when it has to.

```
void getParkingRepresentation() async {
    int i, j, k;
    String spotStatus;
    MaterialColor color;
    List<TableRow> rows = new List();
    List<DataRow> dataRows = new List();
    final keyContext = stickyKey.currentContext;
    double height = 0;
    // This makes sure the orientation is landscape and the keyContext is not
    // null so the height can be properly measured
    try{
      while (MediaQuery.of(context).orientation != Orientation.landscape ||
          keyContext == null) {
        await asyncSleep(100);
      }
    } catch(e){
      print("getParkingRepresentation wasn't loaded");
    }
    final box = keyContext.findRenderObject() as RenderBox;
    height = box.size.height;
    rows.clear();
    streamParkingManagementSubscription = streamAdmin.listen((value) {
      rows.clear();
      List<String> message = value.payload.split('/');
      //PARKING ID
      if (id != null && int.parse(id) >= 63000) {
        if (message[0] == "PARK") {
          var matrix = new List.generate(int.parse(message[1]),
              (_) => new List<Container>(int.parse(message[2])));
          for (i = 0; i < int.parse(message[1]); i++) {
            for (j = 0; j < int.parse(message[2]); j++) {
              spotStatus = "empty";
              color = Colors.grey;
              for (k = 3; k < message.length; k++) {
                List<String> spotInfo = message[k].split(';');
                if (i == int.parse(spotInfo[0]) - 1 &&
                    j == int.parse(spotInfo[1]) - 1) {
                  spotStatus = spotInfo[2];
                  color = Colors.red;
                }
              }
              // Draws ENTRY on the first cell
              if (i == 0 && j == 0) {
                matrix[i][j] = Container(
                  height: height / double.parse(message[1]) -
                      5 / double.parse(message[1]),
                  color: color,
                  child: Column(
                    mainAxisAlignment: MainAxisAlignment.spaceAround,
                    children: [
                      Text(spotStatus),
                      Text(
                        "ENTRY",
                        style: TextStyle(
                            color: Colors.white, backgroundColor:
Colors.green),
                      ),
                    ],
                  ),
                );
                // Draws EXIT on the first row's last cell
              } else if (i == 0 && j == int.parse(message[2]) - 1) {
                matrix[i][j] = Container(
                  height: height / double.parse(message[1]) -
                      5 / double.parse(message[1]),
                  color: color,
                  child: Column(
                    mainAxisAlignment: MainAxisAlignment.spaceAround,
                    children: [
                      Text(spotStatus),
                      Text(
                        "EXIT",
                        style: TextStyle(
                            color: Colors.white, backgroundColor:
Colors.green),
                      ),
                    ],
                  ),
                );
              } else {
                matrix[i][j] = Container(
                  height: height / double.parse(message[1]) -
                      5 / double.parse(message[1]),
                  color: color,
                  child: Center(
                    child: Text(spotStatus),
                  ),
                );
              }
            }
            rows.insert(0, new TableRow(children: matrix[i]));
          }
          setState(() {
            nonexistentID = false;
            rows = rows;
            table = Table(
              border: TableBorder.all(),
              children: rows.length != 0
                  ? rows
                  : [
                      TableRow(
                        children: [
                          CircularProgressIndicator(),
                          Center(
                            child: Text("Introduce parking or vehicle ID"),
                          )
                        ],
                      )
                    ],
            );
          });
        } else if (message[0] == "PARK-ERR") {
          setState(() {
            nonexistentID = true;
          });
        }
```

On the other hand, if the ID matches a vehicle ID, the app will display a table with the historic log of the vehicle. The table lists the times the vehicle has entered a parking. Every row shows the enter date, the parking ID, the exit date, and the user assigned to the Twizy. If the vehicle is still inside a parking, the exit and user columns will display “None”.



Parking or Vehicle ID: 1

Enter date	ParkingID	Exit date	User asigned
2020-05-24 20:07:49	63001	2020-05-24 20:16:55	nacho
2020-05-24 20:19:04	63002	None	None

Fig. 82 Management screen vehicle log

Once the table is displayed either the parking or vehicle message request is resent periodically. This updates the table making the representation reliable.

3.8.4 Commands

The commands screen is designed to improve the system development and help on diagnosis and repairs. The screen shows a search button where the admin must enter the vehicle ID, 8 buttons that are used to send commands to the vehicle, and a window at the bottom where received messages are displayed. The errors that can be seen in this window can be found in ANNEX 2. While the admin does not enter any ID, the buttons are disabled and the bottom field displays a message “SELECT VEHICLE ID”. Also, the ID field shows the frame on red.

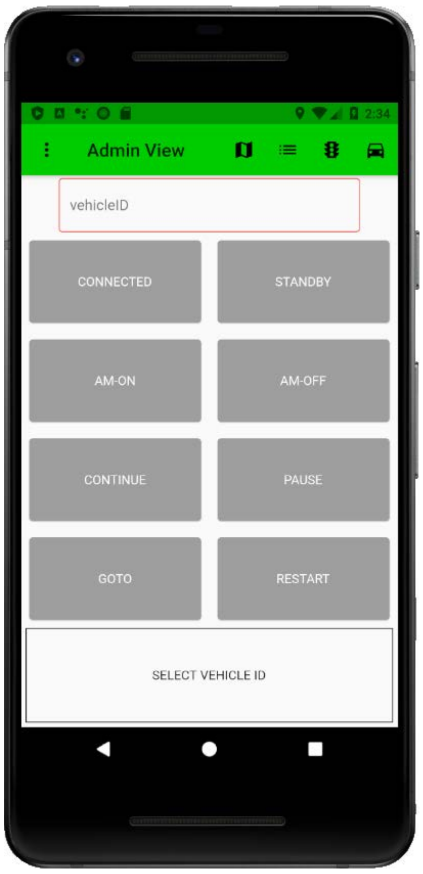


Fig. 83 Commands screen first view

Once the ID Is introduced the search field frame changes to green, the buttons are enabled, and the bottom field looks empty. There is a button for every command the car can receive. The information about the commands and messages that use the Twizy can be found at the Final Degree Thesis report of the TwizyLine member Ignacio Royuela [27]. To give a brief explanation, there are 8 buttons. CONNECTED, STANDBY, AM-ON, AM-OFF, CONTINUE, PAUSE, GOTO y RESTART. The CONNECTED button is used to connect the vehicle to the app if the server is down or the car is being repaired, the STANDBY, AM-ON and AM-OFF buttons are used to choose the vehicle operation mode. If the vehicle is in autonomous mode (AM-ON) some commands can be sent to it. Those commands are given through the buttons CONTINUE (which makes the vehicle start moving), PAUSE (which makes the vehicle stop moving), or GOTO (which can define a route to follow). The RESTART button is used to restart the vehicle and it must be pressed when the vehicle has an error. The messages that the app and the vehicle exchange can be found at ANNEX 1.

All the buttons work the same way. They light in a chosen color when the proper message is received. All the messages received are shown at the bottom of the screen. This reinforces the ability of the admin to know what is happening at every moment, as there is a straight relation between the button that lights and the last message received. So, for example, when an admin presses the PAUSE button, it will not light until the message OK from the car is received.

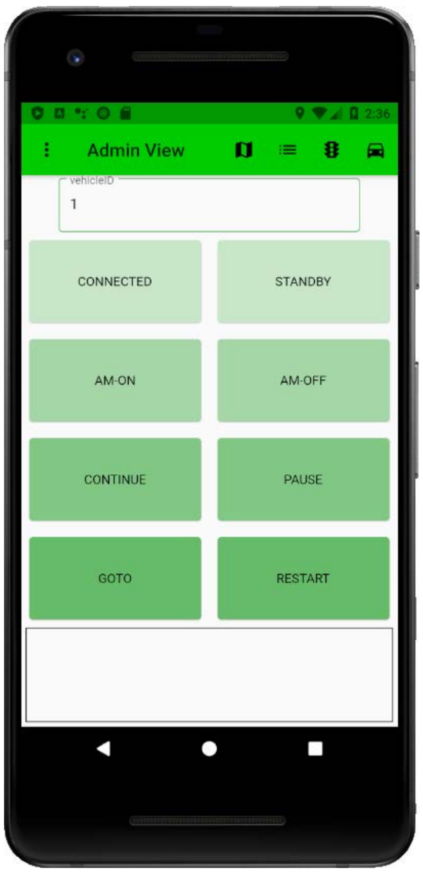


Fig. 84 Commands screen ID introduced

Two situations may arise when operating the vehicles. In the first situation, the car is already turned on and it has logged into the server, at this point the app starts receiving messages from the vehicle and can send any command to the vehicle. In the second situation, the car is off and therefore it is not connected to the server. In this case, when the car is turned on, the admin can handle the connection procedure just in case the server is not available at that time. The procedure is the next one: When the vehicle turns on it starts sending a “connect” message. This message is sent periodically until someone answers it. The app receives the connect message and shows it at the bottom of the screen. At the same time, the button CONNECTED lights in yellow as shown in Fig. 85, and the admin has five seconds to tap on it.

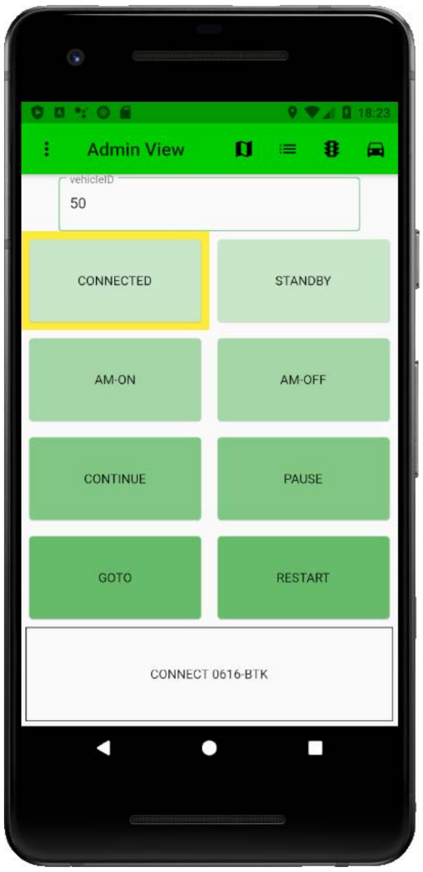
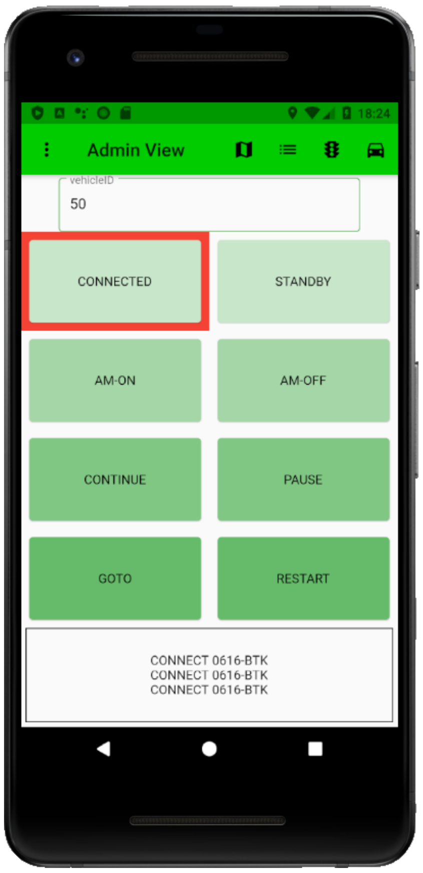


Fig. 85 Commands screen Connect received

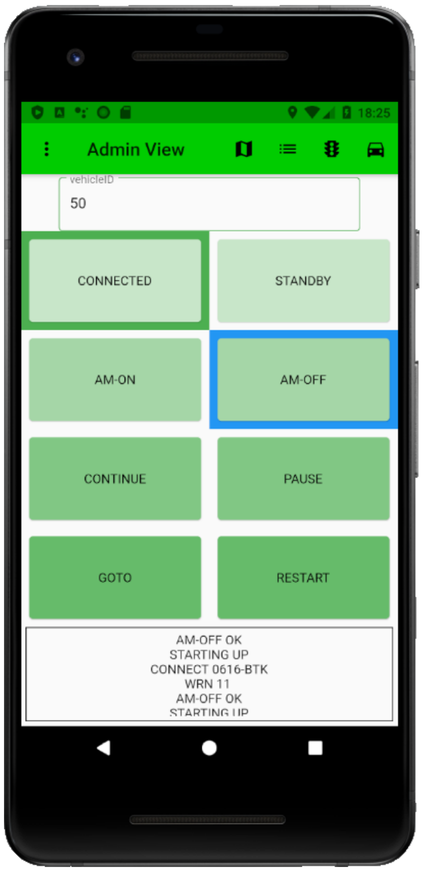
If the admin does not hit CONNECTED on time, the button lights red until the message is received again as shown in Fig. 86. During the time that the button lights red if the admin presses connected it will not work. When a new “connect” message is received the button lights yellow again and the admin has again five seconds to tap on the CONNECTED button.

Once the admin taps on the CONNECTED button, the app sends an answer to the car and it considers to be connected, stopping sending the connect message. Moreover, the button CONNECTED will light in green for 5 seconds. After that, the vehicle starts by default in am-off mode, this is, it is not in autonomous mode.



```
CONNECT 0616-BTK
CONNECT 0616-BTK
CONNECT 0616-BTK
```

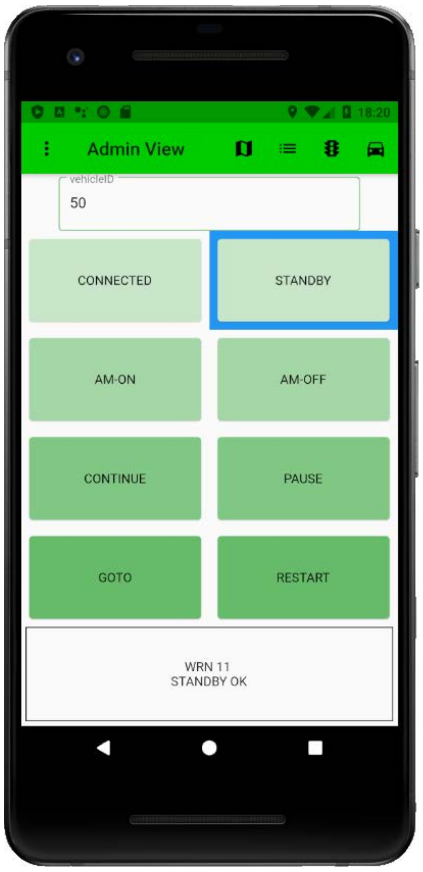
Fig. 86 Commands screen Connect missed



```
AM-OFF OK
STARTING UP
CONNECT 0616-BTK
WRN 11
AM-OFF OK
STARTING UP
```

Fig. 87 Commands screen Vehicle connected

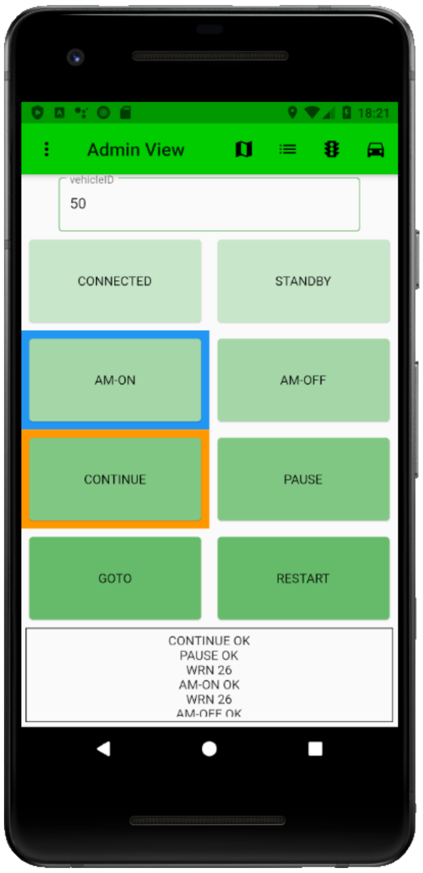
The vehicle can be either in “AM OFF” (manual) or “AM ON” (autonomous) or “STANDBY” mode (This mode is activated when the vehicle stops inside a parking. This could happen because it found an end of row RFID tag or it detected a vehicle in front of it). These modes are self-excluding, and only one can be selected at the same time. When one of them is selected the respective button lights in blue color, and the other 2 do not light.



WRN 11
STANDBY OK

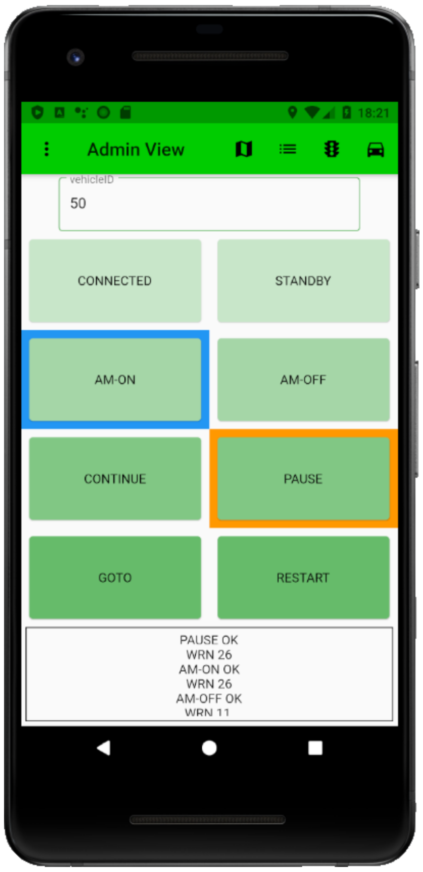
Fig. 88 Commands screen Standby mode

Once the vehicle is on “AM ON” mode it can be either on “CONTINUE” or “PAUSE” mode. The default is “CONTINUE” mode. CONTINUE and PAUSE modes are self excluding too, and their respective buttons light in orange button. When the vehicle is in the CONTINUE mode, it is moving in autonomous mode. If the vehicle is in the PAUSE mode it is in the autonomous mode too, but it is stopped (for instance because it found a vehicle in front or an obstacle). The pause mode is the previous step to the STANDBY mode. When a vehicle enters a row where there are more Twizys, it follows the line until it finds a Twizy, and then it stops and informs the server, after a timeout expires the vehicle switches from AM-ON mode to STANDBY mode.



CONTINUE OK
PAUSE OK
WRN 26
AM-ON OK
WRN 26
AM-OFF OK

Fig. 89 Commands screen Continue mode



PAUSE OK
WRN 26
AM-ON OK
WRN 26
AM-OFF OK
WRN 11

Fig. 90 Commands screen Pause mode

The GOTO command can be used to give the car a route. It provides the admin four text fields (see Fig. 91). This command format is: "GOTO (initialSpeed [km/h]) (defaultFork [L or R]) [T(RFID) [(newSpeed [km/h])]...] [V(RFID) (newSpeed [km/h])...] S(RFID)", as it is explained more detailly in the Final Degree Thesis report of the TwizyLine member Ignacio Royuela in [27]. There are mandatory fields to specify the INITIAL SPEED, the DEFAULT FORK and the End of Row RFID tag identifier. The INITIAL SPEED is the speed the vehicle starts the route with, and if not changed this speed remains constant until the car stops. The DEFAULT FORK indicates if the vehicle must turn left or right by default in bifurcations. The End of Row RFID is used to tell the vehicle when to stop. In this field the admin only needs to enter the RFID tag, and the app will place the 'S' before the RFID tag ID by itself. Once the vehicle reads this RFID tag it will stop. Just with these three options a route can be given to the vehicle.

There is another optional field used to give more complex routes to the vehicle. The OPTIONAL DECISION RFID field has a more complex format. Every order gives instructions to the vehicle to do when finding the specified RFID tag identifier. If the order set the vehicle to turn to the opposite direction to that established in DEFAULT FORK, it is indicated by a 'T' followed by the RFID tag ID after which the turn must be done, with no space between. If the order also changes the speed plus the direction, it is indicated by the speed in Km/h after the RFID tag identifier, separated by one space. If the order just changes the speed of the vehicle after a determined RFID tag identifier is found, it is indicated by a 'V' followed by the RFID tag identifier with no space between, and the speed to take after the tag, specified in Km/h, separated by one space. Different orders can be concatenated using one space as separator between each one.

Every field shows a hint text once tapped, helping to understand how every field should be filled. The fields also have a maximum length (but the decision RFID field) and are validated by the app after the send button is pressed Fig. 94. This validation includes checking the mandatory fields are filled, the defaultFork field is either L or R but no other character, the initial speed is not over 25 km/h, or the end of row RFID field only contains numeric characters. This validation is followed by a proper keyboard display (text keyboard when alphanumeric characters are accepted, as it is shown in Fig. 92, and numeric keyboard when only numbers are accepted, as it is shown in Fig. 93).

The info button next to the send button displays a dialog with instructions for the admin to understand how the GOTO command works (see Fig. 95).

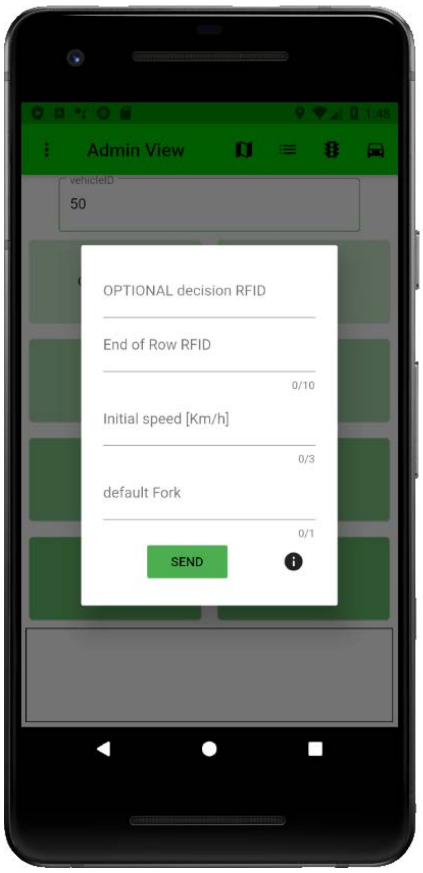


Fig. 91 GOTO command fields

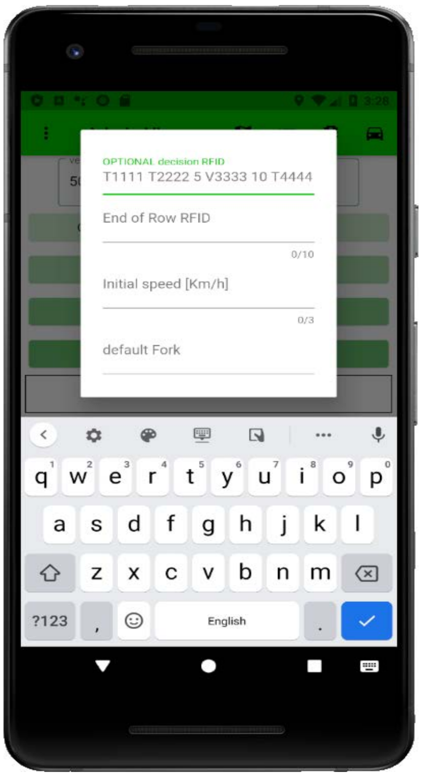


Fig. 92 GOTO command: field hint text keyboard

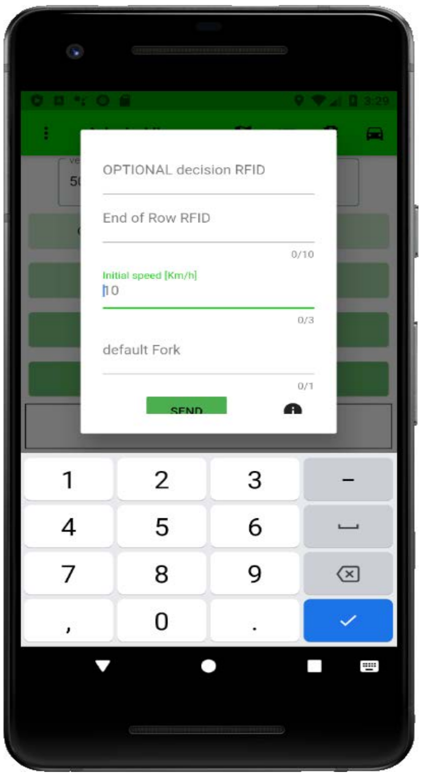


Fig. 93 GOTO command: field hint numeric keyboard

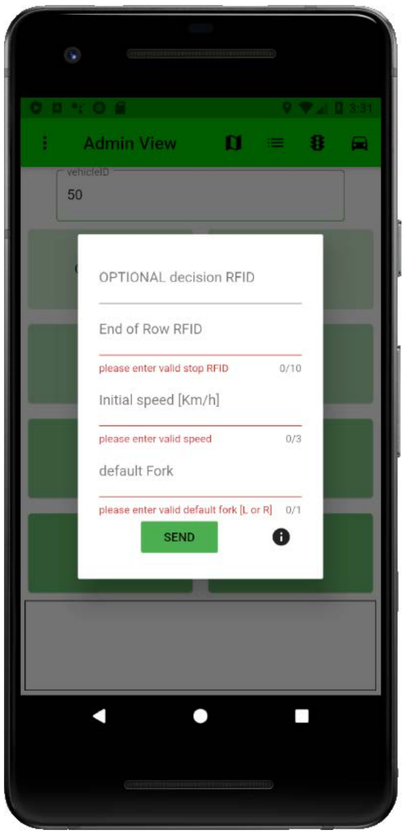
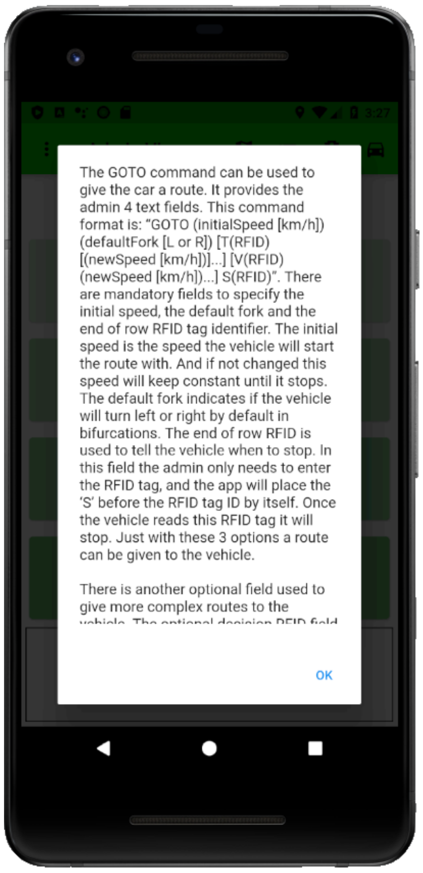

Admin View
vehicleID
50
OPTIONAL decision RFID
End of Row RFID
please enter valid stop RFID 0/10
Initial speed [Km/h]
please enter valid speed 0/3
default Fork
please enter valid default fork [L or R] 0/1
SEND

Fig. 94 GOTO command: fields validation

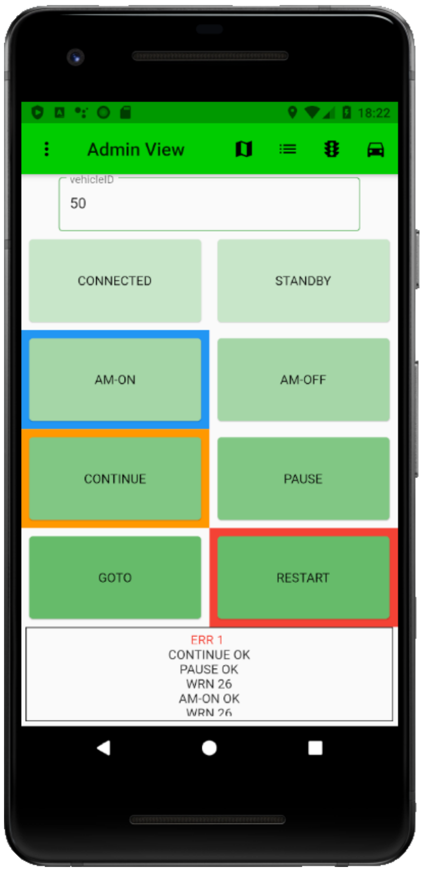


The GOTO command can be used to give the car a route. It provides the admin 4 text fields. This command format is: "GOTO (initialSpeed [km/h]) (defaultFork [L or R]) [T(RFID) [(newSpeed [km/h])]...] [V(RFID) (newSpeed [km/h])...] S(RFID)". There are mandatory fields to specify the initial speed, the default fork and the end of row RFID tag identifier. The initial speed is the speed the vehicle will start the route with. And if not changed this speed will keep constant until it stops. The default fork indicates if the vehicle will turn left or right by default in bifurcations. The end of row RFID is used to tell the vehicle when to stop. In this field the admin only needs to enter the RFID tag, and the app will place the 'S' before the RFID tag ID by itself. Once the vehicle reads this RFID tag it will stop. Just with these 3 options a route can be given to the vehicle.

There is another optional field used to give more complex routes to the

Fig. 95 GOTO command: instructions

During all the operation with the buttons that has been described above, whenever an error is received it is displayed on red at the bottom window. Also, the RESTART button starts flashing red. This is because errors need the car to be restarted to be fixed.



ERR 1
CONTINUE OK
PAUSE OK
WRN 26
AM-ON OK
WRN 26

Fig. 96 Commands screen Error and Restart

The commands functionality is the only communication between the app and the car that does not pass through the server. Like any communication in TwizyLine, it works by using MQTT. So, it necessarily goes through the MQTT broker. But the broker and the server are not the same program, and they could even be working on different machines. The fact that this functionality does not need the server to work means it could be used to move the vehicles inside a parking even if the server is fallen. The app and the vehicle itself implement safety mechanisms to make sure the tool is safe to use.

3.9 Dropdown menu

At the top left of the screen there is a three points icon at the app bar. This icon is visible either on the admin view or the users view. This button deploys a dropdown menu as shown in Fig. 97. This menu displays the option to log out, together with contact and about options.

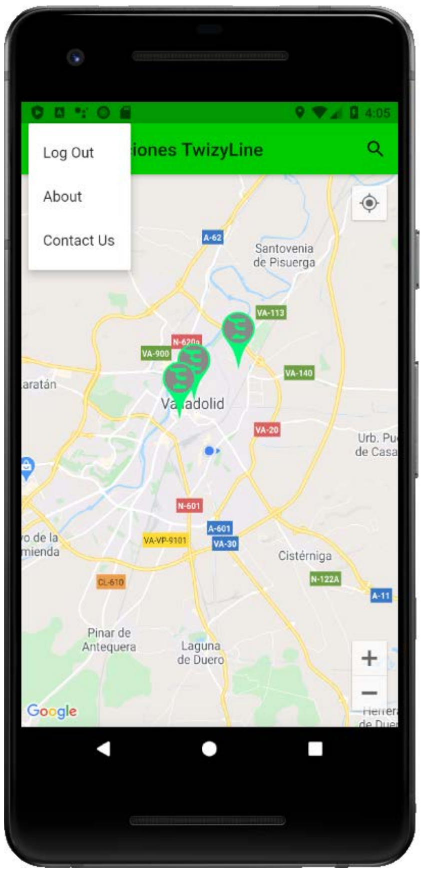


Fig. 97 Dropdown menu

The log out button deletes the saved credentials and sends the user to the login page. The contact and about us buttons should redirect to our website where more info would be given. Our website is still a future line tough. The website can be directly displayed on the app using a WebView. The decision to use a website instead of creating a new page in the app is to reduce costs and time. The website would be available as well for people that do not have the app installed. The information about the service should be available to anyone to attract further users to use the service. So, we could develop a responsive website once, available for everyone. The website would host information relative to how the system works, frequently asked questions, contact information or TwizyLine brand info. Displaying this website in the app is more efficient than creating the content twice and maintaining both.

4. USE CASES. TEST AND INTEGRATION

As it was explained in chapter 1, this Final Degree Project is part of a bigger one, which is mainly described in the Final Degree Projects of Samuel Pilar [26] and Ignacio Royuela [27]. A great part of the three projects were developed during the pandemic COVID-19, which means that they were mainly developed in an isolated way. As a consequence, the test and integration phase, which is crucial in most of the team projects, was essential in this one. This chapter presents the use cases considered to test and integrate the different parts of the project, and in this report, the focus is mainly on the results of the app. The results for the other parts of the project are included in the respective thesis of the other members of the team.

Consequently, this chapter describes the interaction between the user and the app as well as the interaction between the app and the back end. It is analyzed not only the success cases but also the error cases, showing how the app can give a reasonable response to any failure.

In order to run the tests, a data base with several cars, users, admins and the rest of elements necessary to run them was created. This data base includes several parking areas in the city of Valladolid and there were given a specific configuration of cars. The detailed information about the initial conditions of the test can be read in the Final Degree Thesis report of the TwizyLine member Samuel Pilar [26]

The use cases included in this chapter are, in the first case, the procedure by which a user logs into the app, finds the nearest car park and requests a vehicle from it. And secondly the procedure by which an administrator can, in addition to viewing errors recorded in the server database, receive errors in real time and flags them as resolved.

4.1 Use case 1. User gets a car from a parking.

The first use case is the procedure that allows a user to get a vehicle from a parking. The initial conditions of the test are the next ones: the user has already been registered in the system but is still not logged in, and the app does not have its credentials stored. Moreover, the user has its smartphone language set to English, and has no administrator permissions. Regarding the app, the app has not location permission in the smartphone. The test must check that the user can sign in, find the desired parking area and request a vehicle. The message exchange diagram can is shown in Fig. 100.

As mentioned before the user is not logged in. The user is registered in the system, but the app does not have its credentials stored. This could happen if the user just got a new phone, if they just logged out, or if they cleaned the app stored data through the phone settings. Thus, the first screen displayed is the login page. Once in this screen (see **¡Error! No se encuentra el origen de la referencia.**) the user just has to enter its credentials (username and password) and press the login button. When the login button is pressed, a *SnackBar* shows up saying "Processing data" and the buttons turn to transparent background and grey content and they are invalidated, so if the user tries to press them it will not produce any action. The buttons remain invalid until the server answers. The results of all this process are shown in Fig. 98. and Fig. 99.

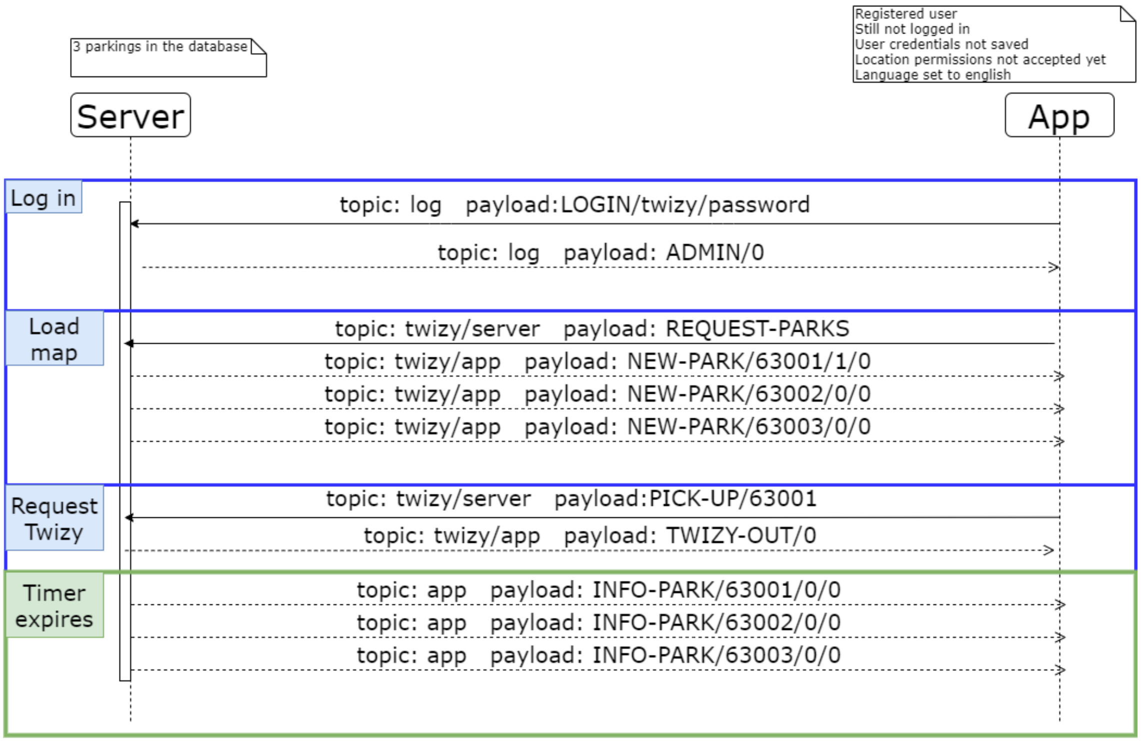

3 parkings in the database
Registered user
Still not logged in
User credentials not saved
Location permissions not accepted yet
Language set to english
Server
App
Log in
topic: log payload:LOGIN/twizy/password
topic: log payload: ADMIN/0
Load map
topic: twizy/server payload: REQUEST-PARKS
topic: twizy/app payload: NEW-PARK/63001/1/0
topic: twizy/app payload: NEW-PARK/63002/0/0
topic: twizy/app payload: NEW-PARK/63003/0/0
Request Twizy
topic: twizy/server payload:PICK-UP/63001
topic: twizy/app payload: TWIZY-OUT/0
Timer expires
topic: app payload: INFO-PARK/63001/0/0
topic: app payload: INFO-PARK/63002/0/0
topic: app payload: INFO-PARK/63003/0/0

Fig. 100 Use case 1 message exchange diagram

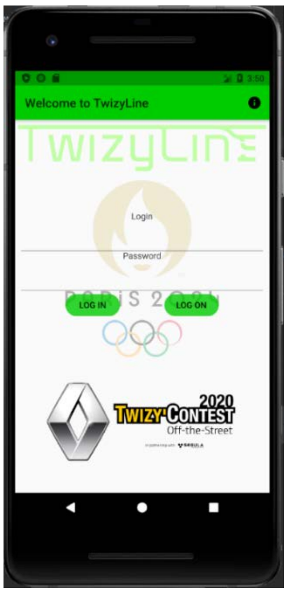


Fig. 99 Login screen view

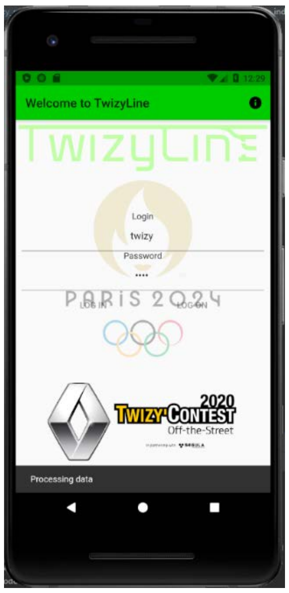
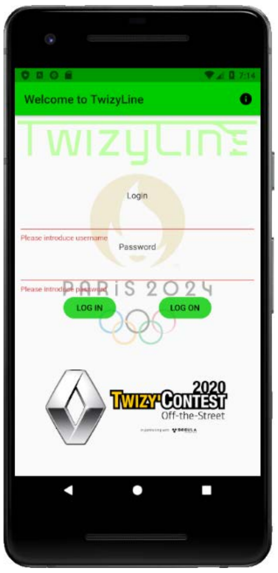


Fig. 98 Login screen processing

If the user presses the button and none of the fields are filled, the app displays a warning, as well as if one of the fields is empty or if the credentials are not valid. All these situations were tested and the results can be seen in Fig. 101, Fig. 102, Fig. 103, and . Fig. 104.

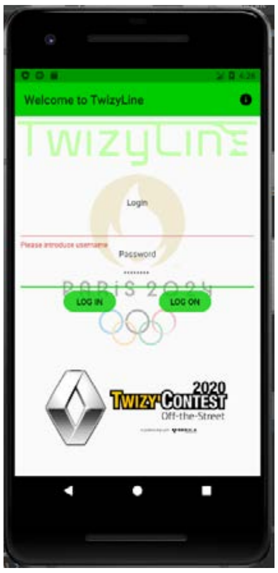
If there is no response from the server, a timer expires and an alert dialog is displayed informing that there was no response and suggesting the user to check their internet connection or try again later. To perform this test, the connection was drop down and the result can be seen in Fig. 105.



Please introduce username

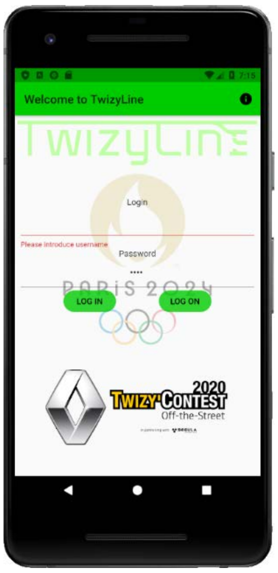
Please introduce password

Fig. 101 LS both fields empty



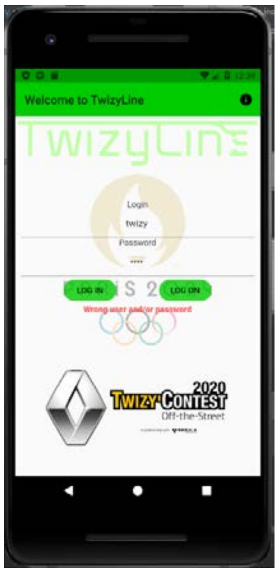
Please introduce username

Fig. 102 LS login field empty



Please introduce username

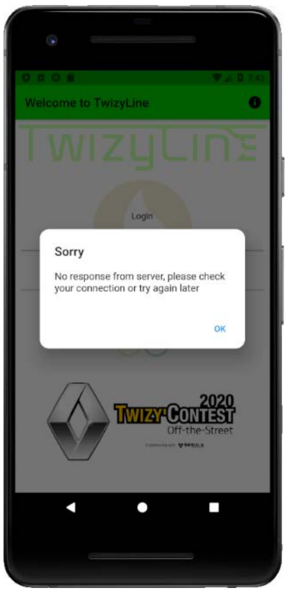
Fig. 103 LS password field empty



Wrong user and/or password

Fig. 104 LS invalid credentials

*LS: Login Screen



Sorry

No response from server, please check your connection or try again later

OK

Fig. 105 alert dialog no response from server

The app connects with the MQTT broker, which has a file with the usernames and passwords stored. These passwords are encrypted. The broker checks this file and returns 0 if valid or 1 if not valid. If the response is 0, the app subscribes to the topic “log” and sends the server the message “LOGIN/user/password” (In our scenario: “LOGIN/twizy/1234” as shown in Fig. 100) through the topic “log”. This topic is only used during the login or logon process, and once the user is logged in, the app unsubscribes from this topic. This redundancy is for a double security, because both the server and the broker check that the user is valid.

Once the user is logged in, the app unsubscribes from the “log” topic and subscribes to the topics “*username/app*” (twizy/app because that is the username entered in **¡Error! No se encuentra el origen de la referencia.***)* and “app”. The “app” topic is a general topic for broadcast messages, and the “Twizy/app” topic is unique for every user (every user has a unique username). Topics where explained in section 3.2, and more information about MQTT in this project can be found in the Final Degree Project by Samuel Pilar [26]). This topic is used by the server to send messages to the app, and the app publishes its messages to the server using the topic "*twizy/server*". The "Twizy/app" topic receives a message from the server with the format "ADMIN/(0 or 1)". This is because when the server receives the LOGIN message it also checks if the user has administrator permissions or not. If the user has administrator permissions, the server answers with "ADMIN/1" and if not, it answers with "ADMIN/0". Because our user does not have admin permissions the received message was ADMIN/0 as shown in *Fig. 100*. The app waits for the server response. Once the response is received, the app stores the user credentials, so the next time the user opens the app there is no need to load the login page and the map view is shown directly. As explained previously, if there is no response from the server, the app shows an alert dialog To perform this test the server was shut down and it was checked that the app displayed the alert dialog shown in Fig. 105.

At this point of our scenario the user has successfully logged in, and the app has subscribed to the "app" and "Twizy/app" topics. The app is about to load the map view, but the app permissions are unknown. Thus, the app asks the user for location permissions, as shown in Fig. 106a. While the permissions are accepted and the map is loaded it was checked the app displays an animation "drawing " TwizyLine slogan: "Drawing the future" as shown in Fig. 106b. As well it was checked that if the user does not accept the location permissions, the alert dialog requesting them is displayed again, as the app cannot work properly without those permissions.

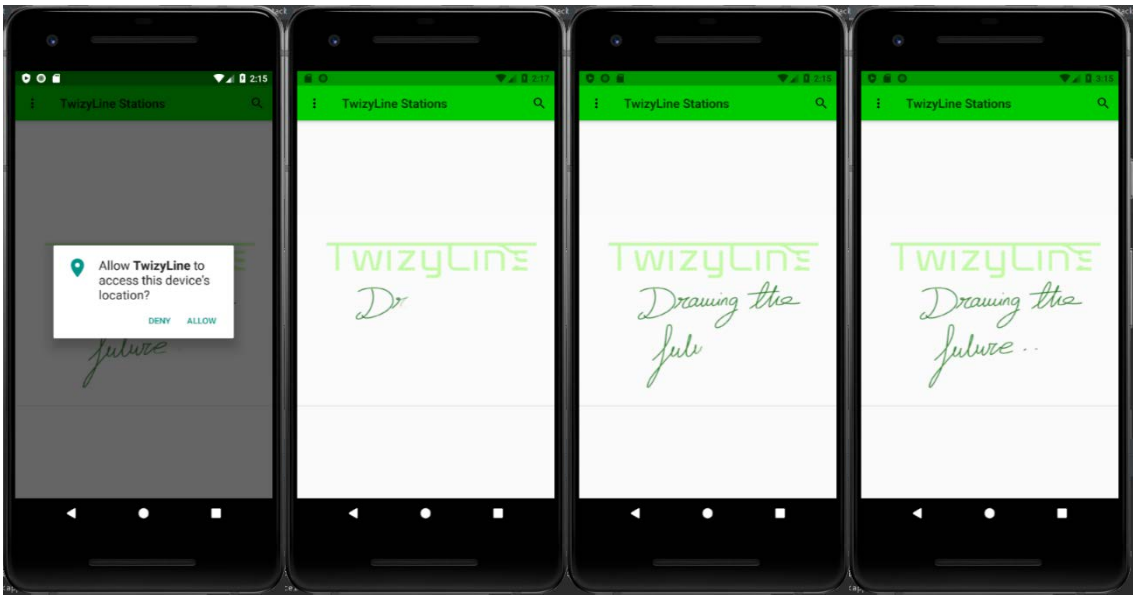


Fig. 106 Asking for permissions

Fig. 107 Loading Map

The next part of the test involves getting the parking areas information from the server. Once the map view is built it is displayed to the user, but there is a translucid grey layer over it that displays the message "Finding TwizyLine stations…" and a circular progress indicator as shown in Fig. 108. This layer is displayed while the app gets the parking locations from the server. When the map page is loaded the app sends to the server a message "REQUEST-PARKS" using the topic "*twizy/server*" as shown in Fig. 100. Once the server receives this message it answers with a message "NEW-PARK/id_parking/name_park/latitude/longitude/available Twizys/free spots" for each parking area that it has stored in the database as shown in Fig. 100. Each of those parking lots is stored in a list, and a marker is displayed on the map. In our case, the data base contained information of three parking areas, as it can be seen in the Final Degree Project by

Samuel Pilar in [26]. If no parking location is received, a timer expires, and the app displays the alert dialog shown in Fig. 109 .

Because the server periodically updates the parking information status, during the test, after getting a vehicle from one of the parking lots, three “INFO-PARK/id_parking/availableYwizys/freeSpots” messages are received (one per each parking stored in the server database) as shown in Fig. 100, and the information pill updates its information.

Once the parking locations are received, the list of parking lots is created, and the markers are displayed on the map. With the test database, the result obtained is shown in Fig. 110.

When the user touches one of the markers from the map the app displays the screen shown in Fig. 111. When the user clicks on the pill, the app sends a “PICK-UP/id_parking” message to the server as shown in Fig. 114. This message contains the parking id from the parking that is represented by the pressed marker. While the app waits for the response from the server to this message, it is displayed the screen shown in Fig. 112. The server then answers with the message “TWIZY-OUT /id_vehicle” as shown in Fig. 100. Then the app displays the screen shown in Fig. 113. This screen displays the ID of the assigned Twizy. If there is no response from the server a timer expires, or if there are not available Twizys in the parking or the server cannot assign a Twizy to the user for some reason, an alert dialog is displayed. All these situations were tested and Fig. 114 was displayed.

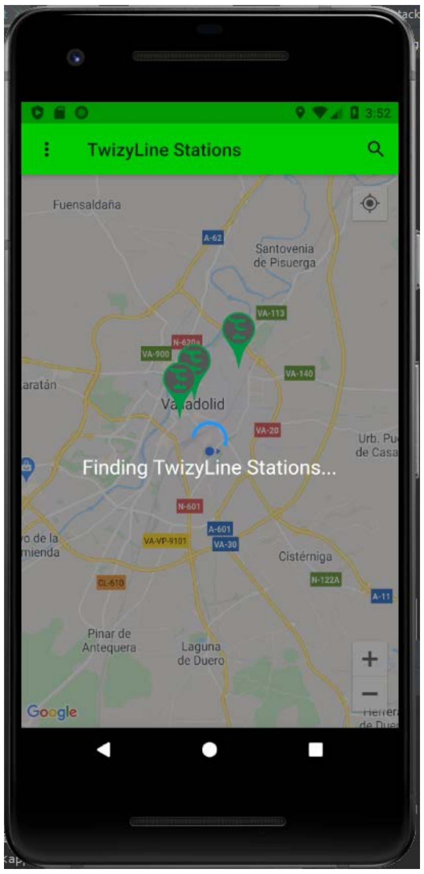
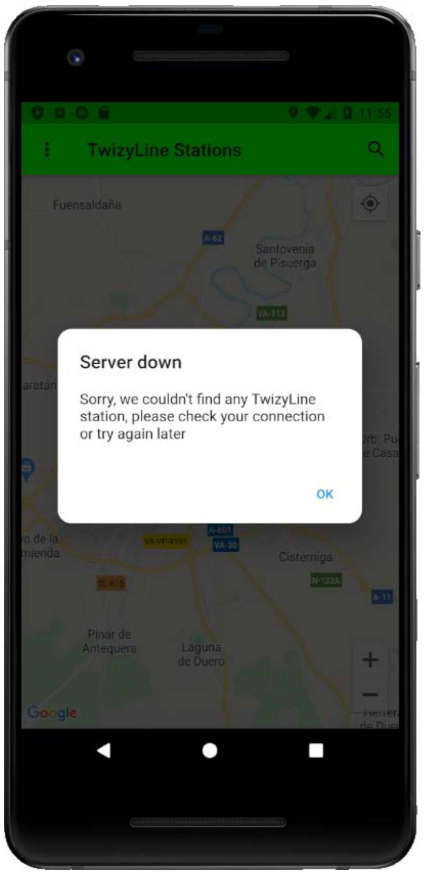


Fig. 108 Finding TwizyLine stations layer



Server down

Sorry, we couldn't find any TwizyLine station, please check your connection or try again later

Fig. 109 No parking location received

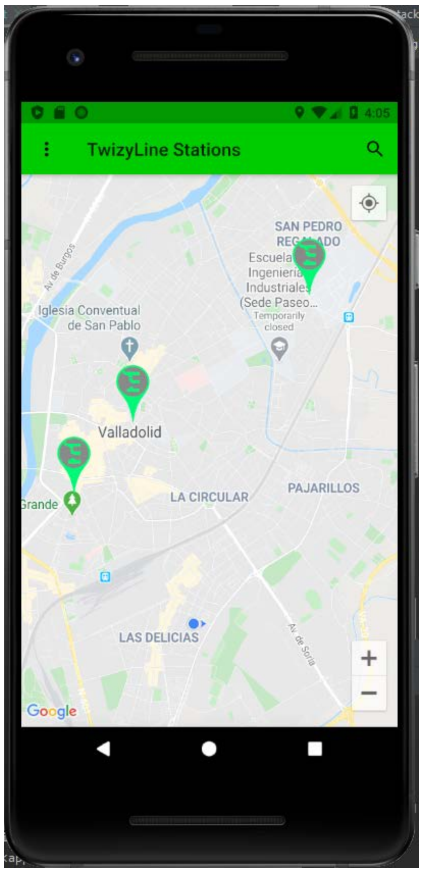


Fig. 110 Map view with markers

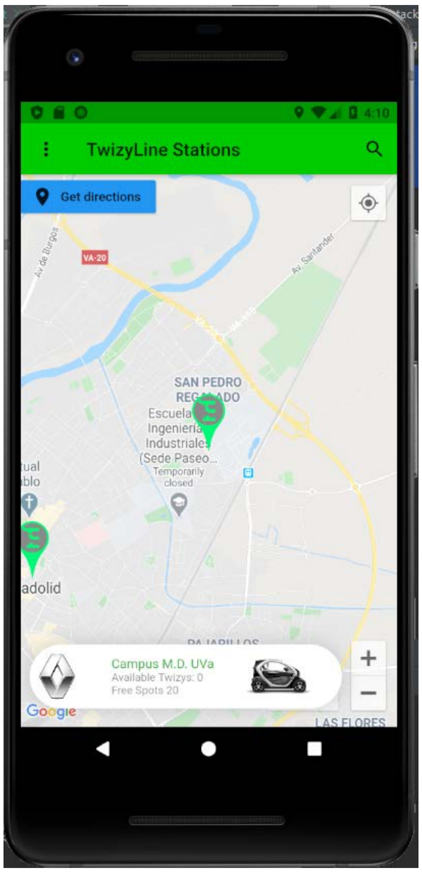


Fig. 111 Marker pressed view

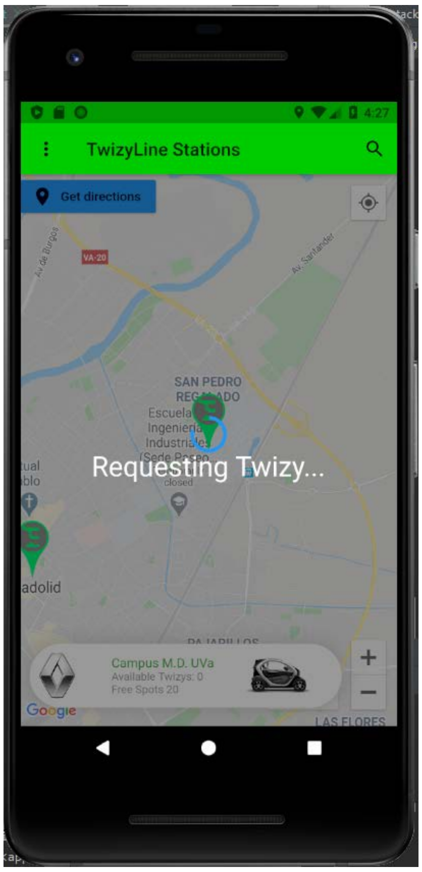
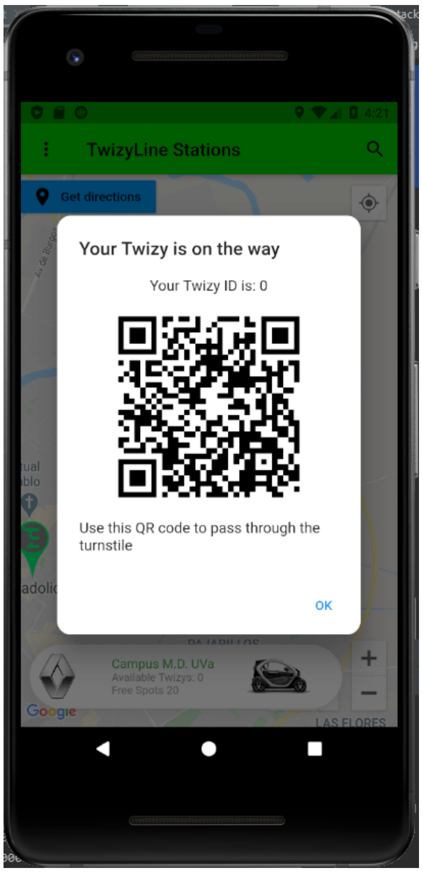


Fig. 112 waiting for server response to Twizy request



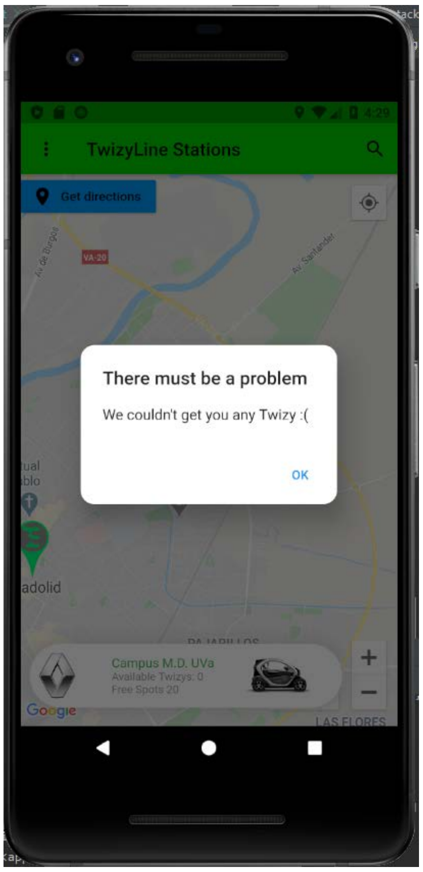
Your Twizy is on the way

Your Twizy ID is: 0

Use this QR code to pass through the turnstile

OK

Fig. 113 Twizy assigned screen



There must be a problem

We couldn't get you any Twizy :(

OK

Fig. 114 Error getting a Twizy

4.2 Use case 2. Admin fixes an error.

The second use case is the procedure that allows an admin to fix an error of a vehicle. The initial conditions of the test are the next ones: the user has already been registered in the system, the user just logged in, and the app has the user credentials stored. Moreover, the user has administrator permissions, thus the user is an admin. The admin username is "lucia". Regarding the app, the app has location permission in the smartphone. The test must check that the admin can see the error information, receive errors in real time and fix them. The message exchange diagram is shown in Fig. 115 Use case 2 message exchange diagram.

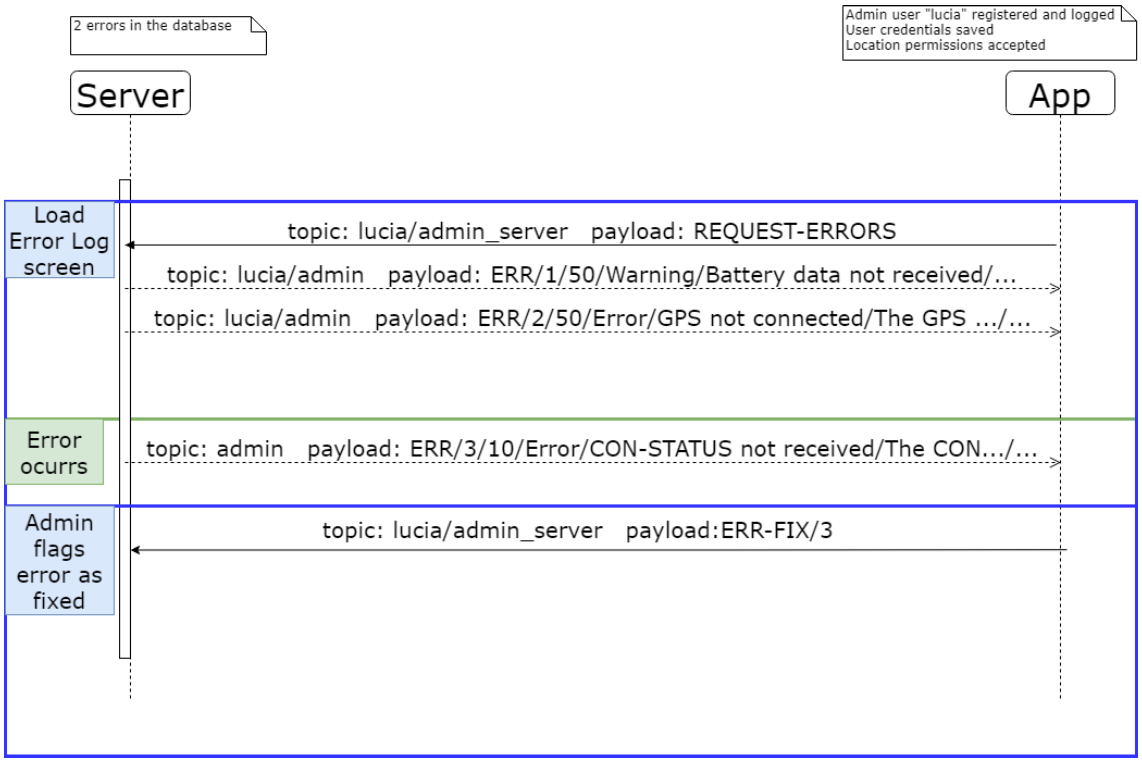
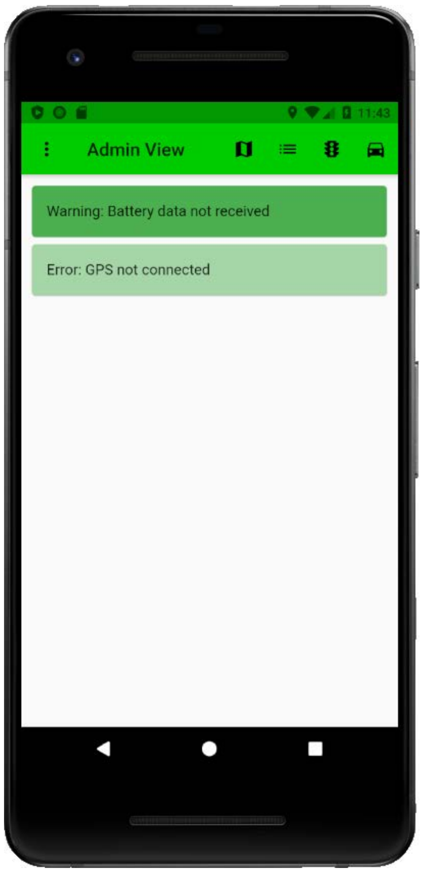

2 errors in the database
Admin user "lucia" registered and logged
User credentials saved
Location permissions accepted
Server
App
Load Error Log screen
topic: lucia/admin_server payload: REQUEST-ERRORS
topic: lucia/admin payload: ERR/1/50/Warning/Battery data not received/...
topic: lucia/admin payload: ERR/2/50/Error/GPS not connected/The GPS .../...
Error ocurrs
topic: admin payload: ERR/3/10/Error/CON-STATUS not received/The CON.../...
Admin flags error as fixed
topic: lucia/admin_server payload:ERR-FIX/3

Fig. 115 Use case 2 message exchange diagram

The first screen displayed to an admin when they log in is the error log screen shown in Fig. 116. While this screen is being built the app sends the server the request to get the errors stored in the database. This request format is "ERR-FIX/id_fix", and this message is sent through the MQTT topic lucia/admin_server as shown in Fig. 115 Use case 2 message exchange diagram. In response to this request the server answers with a message "ERR/id_fix/[id_vehicle OR id_parking]/fail/explanation/error_date/more_info" per each error stored in the database. This error is sent through the lucia/admin topic, which is a personal topic for each user. The id_fix of the error is a unique and auto incremental identifier of the error in the database. This identifier is used to flag the error as fixed. The other fields are used to provide information to the admin. At this point the admin can see two errors in its error log screen (see Fig. 116) and can also see more information about them pressing their card. The information displayed is shown in Fig. 117 . In our test an error in the vehicle with ID 10 was generated, this produced the message"ERR/3/10/Error/ The CON_STATUS frame is not being received. Is the CON-MOD on? Is the CAN-Bus connection of both modules correct?/ 01-07-2020 23:45/None?" to be sent through the topic "admin" as show in Fig. 115. This is a broadcast topic subscribed by all the active app admin users, therefore every admin that has the app opened updates its error log screen with this new error. The server stores the error in the database as well, so those admins that do not have the app active at the moment the error occurs, will receive the message from the database when they open the app again.

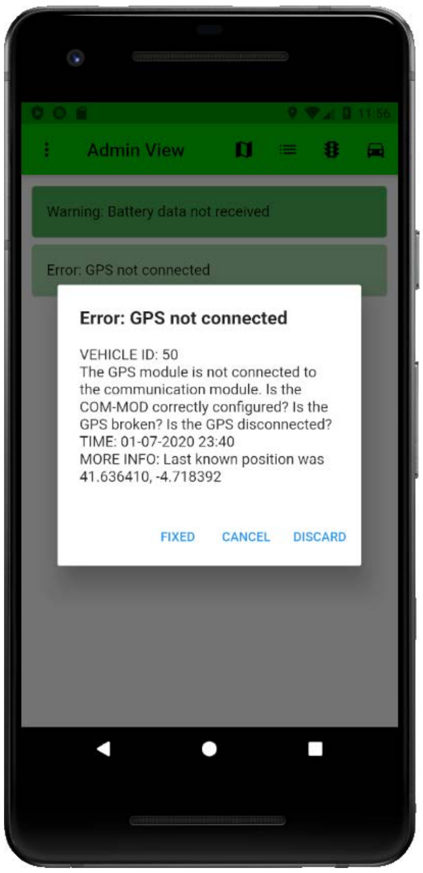
Now the admin can see 3 errors in its error log screen as shown in Fig. 118. Then the admin fixes the error (For instance, reconnecting a wire that had been disconnected) Now that the error is fixed the admin proceeds to flag the error as fixed using the alert dialog in the error log screen as shown in …. When pressing the button, the message “ERR/3” is sent through the “lucia/admin_server” topic, a *SnackBar* is displayed (see Fig. 120), the server deletes this error from its database, and sends the vehicle a RESTART message.



Warning: Battery data not received

Error: GPS not connected

Fig. 116 Errors from the database



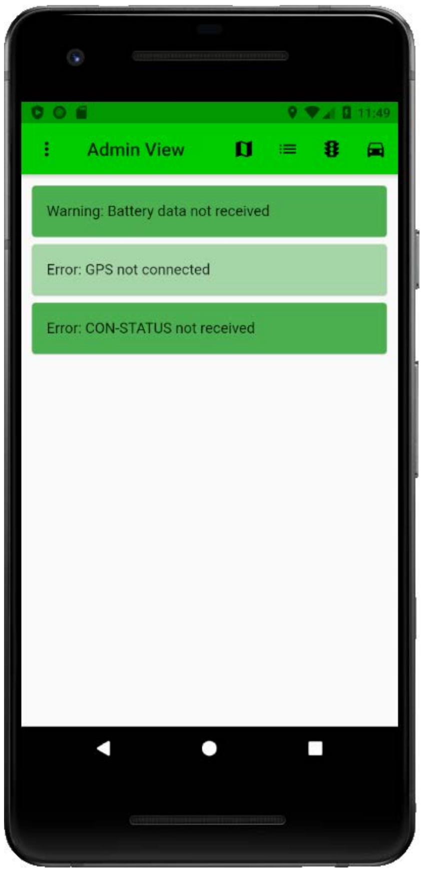
Warning: Battery data not received

Error: GPS not connected

Error: GPS not connected

VEHICLE ID: 50
The GPS module is not connected to the communication module. Is the COM-MOD correctly configured? Is the GPS broken? Is the GPS disconnected?
TIME: 01-07-2020 23:40
MORE INFO: Last known position was 41.636410, -4.718392

Fig. 117 Error information displayed

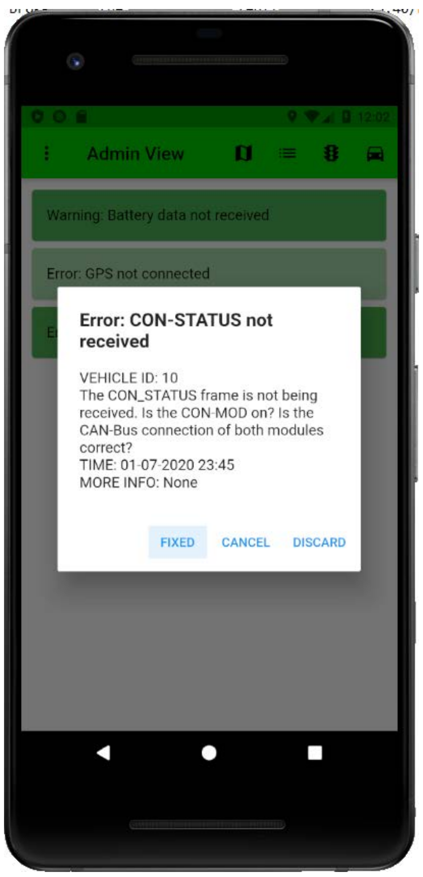


Warning: Battery data not received

Error: GPS not connected

Error: CON-STATUS not received

Fig. 118 Real time error received



Error: CON-STATUS not received

VEHICLE ID: 10
The CON_STATUS frame is not being received. Is the CON-MOD on? Is the CAN-Bus connection of both modules correct?
TIME: 01-07-2020 23:45
MORE INFO: None

Fig. 119 Admin flags the error as fixed

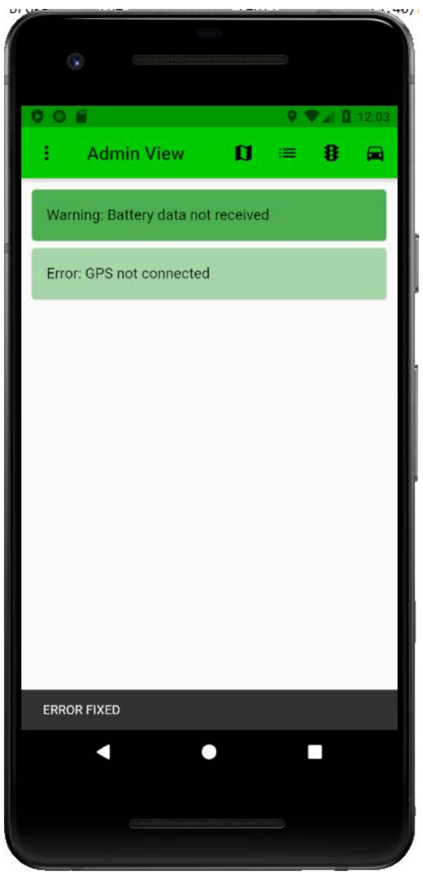


Fig. 120 SnackBar informs the error has been fixed

5. CONCLUSIONS AND FUTURE LINES

This app cannot be understood without the context of TwizyLine project. I hope this app can be a visual support for the project, reflecting the effort and passion that my partners and I are putting in this project. This application opens the door to TwizyLine to be a carsharing service. Developing in a framework like flutter has definitely been a challenge. And using MQTT, a protocol for IOT to connect the app and the server has not been easy either. But I hope those decisions demonstrate to be good. Flutter allows us to develop just one app for the two main smartphone OS worldwide. And MQTT permitted developing just one server for both cars and users.

Summarizing, the specific objectives this project has reached are:

- Developing an app that can be released on the current two main smartphone OS with just one code.
- Designing an easy, beautiful, and clear UI for users, allowing them to access to the TwizyLine carsharing service in multiple languages.
- Providing the service administrators with a useful tool to develop, improve and manage TwizyLine.
- Integrating the app into the rest of TwizyLine systems and testing everything works properly.

There is obviously a lot of work to be done. Some of the future lines of the app are:

- Cyphering MQTT communications through TLS. This would improve the security as the messages would not be sent on plain text.
- Safely storing the credentials that the app saves when login in.
- Implementing a splash screen at the start of the app. This would improve the app appearance and remark the TwizyLine brand. Also improving the loading map animation as for now it is a gift embedded in the screen, but it should be a canvas animation.
- Implementing notifications. Those would be useful for example to inform the user that the Twizy they just returned safely arrived to its parking spot or to inform administrators that a new error has emerged
- Implementing a search bar in the map screen to find parking stations by their name.
- Switching the order the vehicle log table is displayed in the management screen for admins and limiting the logs displayed to 25.
- Adding scroll ability to the parking representation in the management screen for the parking lots that are too big to be represented in the device screen size.
- Requesting users to scan their driving license. The required license could vary depending on the Twizy model (45 Km/h max or 80 Km/h max) and the local regulation.
- Highlighting the closest parking station when the map is built.
- Limiting to users the possibility to request a Twizy only when they are close to a parking. Using geolocation or QR scanning close to the parking.
- Improving the Error log window so an error card disappears if another admin fixes it. (Requires server modifications)
- Implementing Isolates to light the work of the main thread.

- Implementing services, so the app does not lose connection to the server when it is on the background, and implementing reconnection if the connection is lost and properly informing the user if the connection could not be retrieved.
- Improving error handling. For instance, differentiate between server errors and user connection errors, or using the INFO-PARK message when a parking breaks down to inform the user by changing the colour of the parking marker and displaying a warning message when the marker is pressed (requires server modifications).
- Developing TwizyLine website to provide support to this app and help TwizyLine to grow.
- Testing the app on an iOS device.
- Adding other languages translation to make the app appropriate for a multicultural event as Olympic Games are.
- Some other aspects must be discussed on the context of the service more than the system. Like the option to book a Twizy.

An app is always changing and improving. And there is a long way to walk.

6. AWARDS RECEIVED AND MERITS

TwizyLine project is our proposal for Twizy contest 2020, organized by Renault in collaboration with its partner Segula. TwizyLine has been awarded in the national phase and will represent Spain in the international final.

TwizyLine project has as well been awarded as winner of one of the PROMETEO awards. PROMETEO program supports projects from Valladolid University students with the aim to get a market-oriented prototype.

As part of the obligations of PROMETEO awards, TwizyLine project has been presented in a patent application.

References

[1] "ec.europa.eu Urban mobility," 24 June 2020. [Online]. Available: https://ec.europa.eu/transport/themes/urban/urban_mobility_en. [Accessed 24 June 2020].

[2] "twizycontest.com," [Online]. Available: https://www.twizycontest.com/. [Accessed 24 June 2020].

[3] J. P. a. C. Huang, "Climate Change Still Seen as the Top Global Threat, but Cyberattacks a Rising Concern," 2019.

[4] C. Napoli, "Understanding Kyoto's Failure," Jhons Hopkins Univerity Press, 2021.

[5] S. Leahy, "www.nationalgeographic.com, Most countries aren't hitting 2030 climate goals, and everyone will pay the price," November 2019. [Online]. Available: https://www.nationalgeographic.com/science/2019/11/nations-miss-paris-targets-climate-driven-weather-events-cost-billions/. [Accessed 16 June 2020].

[6] A. Morton, "www.theguardian.com UN climate talks: Australia accused of 'cheating' and thwarting global deal," 15 December 2019. [Online]. Available: https://www.theguardian.com/environment/2019/dec/16/un-climate-talks-australia-accused-of-cheating-and-thwarting-global-deal. [Accessed 16 June 2020].

[7] "World Health Organization: Air Pollution," [Online]. Available: https://www.who.int/health-topics/air-pollution#. [Accessed 1 June 2020].

[8] N. M. S. Lozano, "La Economía Colaborativa: Factores desencadenantes y comparación con la economía de mercado," 2016.

[9] L. MALYKHINA, "frace24.com, Paris briefly tops world charts for air pollution," 20 March 2015. [Online]. Available: https://www.france24.com/en/20150320-paris-city-smog-pollution-plume-labs-hidalgo-public-transport-diesel. [Accessed 24 June 2020].

[10] "planvelo.paris," [Online]. Available: https://planvelo.paris/. [Accessed 24 June 2020].

[11] L. Bliss, "bloomberg.com, how paris shifted away from the car," 19 January 2018. [Online]. Available: https://www.bloomberg.com/news/articles/2018-01-19/how-paris-shifted-away-from-the-car. [Accessed 24 June 2020].

[12] "copenhagenixenidex.eu," 2019. [Online]. Available: https://copenhagenizeindex.eu/. [Accessed 1 June 2020].

[13] A. Nossiter, "The New York Times, The Greening of Paris Makes Its Mayor More Than a Few Enemies," 5 October 2019. [Online]. Available: https://www.nytimes.com/2019/10/05/world/europe/paris-anne-hildago-green-city-climate-change.html. [Accessed 24 June 2020].

[14] "ministere de la transition ecologique et solidaire Certificats qualité de l'air : Crit'Air," 15 June 2020. [Online]. Available: https://www.ecologique-solidaire.gouv.fr/certificats-qualite-lair-critair. [Accessed 2024 June 2020].

[15] "bbc.com Paris Mayor Anne Hidalgo calls for ban on diesel cars by 2020," 7 December 2014. [Online]. Available: https://www.bbc.com/news/world-europe-30368504. [Accessed 24 June 2020].

[16] A. C. a. A. Vaughan, "theguardian.com, France to ban sales of petrol and diesel cars by 2040," 6 July 2017. [Online]. Available: https://www.theguardian.com/business/2017/jul/06/france-ban-petrol-diesel-cars-2040-emmanuel-macron-volvo#:~:text=France%20will%20end%20sales%20of,Emmanuel%20Macron's%20government%20has%20announced.. [Accessed 24 June 2020].

[17] A. Dusant, "oecdobserver.com, Paris leads the way in electromobility," November 2015. [Online]. Available: https://oecdobserver.org/news/fullstory.php/aid/5305/Paris_leads_the_way_in_electro-mobility.html. [Accessed 20 June 2020].

[18] "Chargemap.com," [Online]. Available: https://chargemap.com/cities/paris-FR. [Accessed 18 June 2020].

[19] "Paris.fr Paris respire," [Online]. Available: https://www.paris.fr/pages/paris-respire-2122. [Accessed 1 June 2020].

[20] "insee.fr Équipement automobile des ménages en 2017," 25 June 2019. [Online]. Available: https://www.insee.fr/fr/statistiques/2012694#tableau-TCRD_001_tab1_departements. [Accessed 2 June 2020].

[21] "Harris Interactive Mobility in European metropolitan areas," August 2019. [Online]. Available: https://harris-interactive.fr/wp-content/uploads/sites/6/2019/09/Harris-Report-Mobility-in-European-metropolises-Getaround.pdf. [Accessed 02 June 2020].

[22] S. R. Department, "statista.com Car sharing: number of vehicles in selected European cities 2016," 21 September 2016. [Online]. Available: https://www.statista.com/statistics/667728/number-car-sharing-vehicles-selected-european-cities/. [Accessed 02 June 2020].

[23] "Renault y Ferrovial expandirán su servicio de carsharing de Madrid a otras ciudades," *cincodias.elpais.com,* 25 February 2020.

[24] wikipedia, "Juegos Olímpicos de Río de Janeiro 2016," [Online]. Available: https://es.wikipedia.org/wiki/Juegos_Ol%C3%ADmpicos_de_R%C3%ADo_de_Janeiro_2016. [Accessed 17 June 2020].

[25] RTVE.es/EFE, "rtve.es," RTVE, 23 August 2016. [Online]. Available: https://www.rtve.es/noticias/20160823/rio-janeiro-recibio-117-millones-turistas-durante-juegos-olimpicos-410000-eran-extranjeros/1390282.shtml. [Accessed 17 June 2020].

[26] S. P. Arnanz, "Back-end implementation for an automatized car parking," Valladolid, 2020.

[27] I. R. González, "Four level autonomous vehicle for an automatized parking," Valladolid, 2020.

[28] M. M. Fernández, "Plan de Comunicación de TwizyLine: plataforma de carsharing con aparcamiento autónomo"," Valladolid, 2020.

[29] I. R. G. S. P. A. M. M. F. Adrián Mazaira Hernández, "TwizyLine Technologies," Valladolid, 2020.

[30] I. Abril, "cincodieas.elpais.com," 5 June 2006. [Online]. Available: https://cincodias.elpais.com/cincodias/2006/06/05/empresas/1149514783_850215.html. [Accessed 02 June 2020].

[31] "gs.statcounter.com Desktop mobile tablet platform market share worldwide monthly-201501-202005," [Online]. Available: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide/#monthly-201501-202005. [Accessed 19 June 2020].

[32] "zity.eco," [Online]. Available: https://zity.eco/. [Accessed 24 June 2020].

[33] "ubeeqo.com," [Online]. Available: https://www.ubeeqo.com/en/carsharing-paris. [Accessed 25 June 2020].

[34] "share-now.com," [Online]. Available: https://www.share-now.com/fr/en/. [Accessed 25 June 2020].

[35] "free2move.com," [Online]. Available: https://fr.free2move.com/. [Accessed 25 June 2020].

[36] "getaround.com," [Online]. Available: https://fr.getaround.com/. [Accessed 25 June 2020].

[37] "Flurry Analytics: Apps Solidify Leadership Six Years into the Mobile Revolution," 1 April 2014. [Online]. Available: https://www.flurry.com/post/115191864580/apps-solidify-leadership-six-years-into-the-mobile. [Accessed 19 June 2020].

[38] K. Technologies, "Medium, 9 Advantages of Mobile Apps over responsive eCommerce websites," 28 march 2017. [Online]. Available: https://medium.com/@KNOWARTH/9-advantages-of-mobile-apps-over-responsive-ecommerce-websites-6aed1e6db0d8. [Accessed 19 June 2020].

[39] N. Deshdeep, "vwo, 10 reasons mobile apps are better," 4 June 2020. [Online]. Available: https://vwo.com/blog/10-reasons-mobile-apps-are-better. [Accessed 19 June 2020].

[40] J. Badalian, "MobileSmith Health, html5 vs native debate is over," 20 May 2015. [Online]. Available: https://www.mobilesmith.com/html5-vs-native-debate-is-over/. [Accessed 19 June 2020].

[41] E. Spence, "Forbes, The Mobile Browser Is Dead, Long Live The App," 2 April 2014. [Online]. Available: https://www.forbes.com/sites/ewanspence/2014/04/02/the-mobile-browser-is-dead-long-live-the-app/. [Accessed 19 June 2020].

[42] J. Clement, "statista.com Number of apps available in leading app stores as of 1st quarter 2020," 4 May 2020. [Online]. Available: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/. [Accessed 19 May 2020].

[43] J.Clement, "statista.com Annual number of global mobile app downloads 2016-2019," 17 Jan 2020. [Online]. Available: https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/. [Accessed 19 May 2020].

[44] J.Clement, "statista.com Most popular Apple App Store categories 2020," 22 June 2020. [Online]. Available: https://www.statista.com/statistics/270291/popular-categories-in-the-app-store/. [Accessed 19 May 2020].

[45] S. W. P. A. Tim A. Majchrzak, "Comparing the Capabilities of Mobile Platforms for Business App Development," 2015.

[46] Imaginovation, "medium.com 5 Decisions to Make Before Developing a Mobile App," 25 October 2017. [Online]. Available: https://medium.com/@Imaginovation/5-decisions-to-make-before-developing-a-mobile-app-7b832977f484. [Accessed 2 July 2020].

[47] "gs.statcounter.com Mobile-tablet vendor market share in north america monthly-201501-202004," [Online]. Available: https://gs.statcounter.com/vendor-market-share/mobile-tablet/north-america/#monthly-201501-202004]). [Accessed 22 5 2020].

[48] S. O'Dea, "statista.com Global mobile OS market share 2009-2018, by quarter," 27 February 2020. [Online]. Available: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/. [Accessed 20 May 2020].

[49] "gs.statcounter.com mobile OS market share Europe monthly-201501-202004," [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/europe/#monthly-201501-202004. [Accessed 20 May 2020].

[50] "gs.statcounter.com mobile OS market share North America monthly-201501-202004," [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/north-america/#monthly-201501-202004. [Accessed 20 May 2020].

[51] "shandweb.com Apps hibridas vs nativas vs generadas," [Online]. Available: https://shandweb.com.mx/shandweb/article/apps-hibridas-vs-nativas-vs-generadas/49. [Accessed 20 June 2020].

[52] A. A. Mohamed Lachgar, "Decision Framework for Mobile Development Methods," *International Journal of Advanced Computer Science and Applications,* no. 8, 2017.

[53] Imaginovation, "Medium, App Development Decisions: Native, Web or Hybrid?," 24 January 2018. [Online]. Available: https://medium.com/@Imaginovation/app-development-decisions-native-web-or-hybrid-31c103f9b4e1. [Accessed 2 July 2020].

[54] M. Fitzgerald, "Medium, Android vs. iOS App Development: Which Is the Better Choice for Your Business in 2019?," 9 October 2019. [Online]. Available: https://medium.com/better-programming/android-vs-ios-app-development-which-is-the-better-choice-for-your-business-in-2019-933b20e4d9b8. [Accessed 2 July 2020].

[55] V. Collins, "Forbes, The Decline Of The Native App And The Rise Of The Web App," 5 April 2019. [Online]. Available: https://www.forbes.com/sites/victoriacollins/2019/04/05/why-you-dont-need-to-make-an-app-a-guide-for-startups-who-want-to-make-an-app/#1bb9a3a86e63. [Accessed 2 July 2020].

[56] "IONOS, Conceptos básicos: definición de web app y ejemplos," 7 March 2019. [Online]. Available: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-una-web-app-y-que-clases-hay/. [Accessed 2 July 2020].

[57] M. Fitzgerald, "Medium, Native, Hybrid, or Web Apps: What Is the Best Approach for Lasting Success?," 4 October 2019. [Online]. Available: https://medium.com/better-programming/native-hybrid-or-web-apps-what-is-the-best-approach-for-lasting-success-91afbb872d89. [Accessed 2 July 2020].

[58] P. L. Sam Richard, "web.dev, What are Progressive Web Apps?," 6 1 2020. [Online]. Available: https://web.dev/what-are-pwas/. [Accessed 21 5 2020].

[59] A. Russell, "Infrequently Noted, What, Exactly, Makes Something A Progressive Web App?," 12 Spetember 2016. [Online]. Available: https://infrequently.org/2016/09/what-exactly-makes-something-a-progressive-web-app/. [Accessed 2 July 2020].

[60] P. LePage, "Tu primera Progressive Web App," 15 May 2019. [Online]. Available: https://developers.google.com/web/fundamentals/codelabs/your-first-pwapp/. [Accessed 2 July 2020].

[61] "MDN web docs, Progressive web apps (PWAs)," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps. [Accessed 2 july 2020].

[62] "Wikipedia, Progressive Web Application," [Online]. Available: https://en.wikipedia.org/wiki/Progressive_web_application. [Accessed 2 July 2020].

[63] F. B. Alex Rusell, "Infrequently Noted, Progressive Web Apps: Escaping Tabs Without Losing Our Soul," 15 June 2015. [Online]. Available: https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/. [Accessed 2 July 2020].

[64] "MDN, Introduction to progressive web apps," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Introduction#Advantages_of_web_applications. [Accessed 2 July 2020].

[65] M. Firtman, "Medium, Progressive Web Apps in 2020," 2 January 2020. [Online]. Available: https://medium.com/@firt/progressive-web-apps-in-2020-c15018c9931c. [Accessed 2 July 2020].

[66] BoostTypers, "Medium, Why PWA is so popular right now?," 12 June 2019. [Online]. Available: https://medium.com/boosttypers/why-pwa-is-so-popular-right-now-d35d35218d09. [Accessed 2 July 2020].

[67] "developers.google.com, Twitter Lite PWA Significantly Increases Engagement and Reduces Data Usage," 3 November 2019. [Online]. Available: https://developers.google.com/web/showcase/2017/twitter. [Accessed 2 July 2020].

[68] tecnoweb, "Aplicaciones nativas, aplicaciones web y aplicaciones híbridas," September 2013. [Online]. Available: http://tecnologiasdeinternet.com/2013/09/aplicaciones-moviles-iphone-android/. [Accessed 2 July 2020].

[69] W. Jobe, "Native Apps Vs. Mobile Web Apps," *International Journal of Interactive Mobile Technologies (iJIM),* 2013.

[70] "IONOS, Hybrid apps - combining the best of web and native apps," 2 November 2016. [Online]. Available: https://www.ionos.com/digitalguide/websites/web-development/hybrid-apps-combining-the-best-of-web-and-native-apps/. [Accessed 2 July 2020].

[71] C. Griffith, "Ionic, What is Hybrid App Development?," [Online]. Available: https://ionicframework.com/resources/articles/what-is-hybrid-app-development. [Accessed 2 July 2020].

[72] R. Chernysh, "Medium, Hybrid development: Ionic, React Native or Flutter?," 6 January 2020. [Online]. Available: https://medium.com/webspace-team/hybrid-development-ionic-react-native-or-flutter-5aefa3025b6f. [Accessed 2 July 2020].

[73] A. F. Gutierrez, "blogthinkbig.com, Aplicaciones web vs. aplicaciones nativas vs. aplicaciones híbridas," 20 February 2013. [Online]. Available: https://blogthinkbig.com/aplicaciones-web-nativas-hibridas. [Accessed 20 June 2020].

[74] "Android authority: How to create non-games apps in unity," [Online]. Available: https://www.androidauthority.com/make-unity-apps-1073017/. [Accessed 28 June 2020].

[75] J. MacFadyen, "phonegap, Our Continued Commitment," 14 February 2017. [Online]. Available: https://phonegap.com/blog/2017/02/14/continued-commitment/. [Accessed 28 June 2020].

[76] M. Kremer, "ionic, Comparing Cross-Platform Frameworks," [Online]. Available: https://ionicframework.com/resources/articles/ionic-vs-react-native-a-comparison-guide. [Accessed 2 July 2020].

[77] S. Liu, "statista.com, Cross-platform mobile frameworks used by software developers worldwide in 2019 and 2020," 3 12 2019. [Online]. Available:

https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/. [Accessed 22 5 2020].

[78] "Google Trends," [Online]. Available: https://trends.google.com/trends/explore?date=2017-05-22%202020-05-22&q=%2Fg%2F11f03_rzbg,react%20native,%2Fg%2F1q6l_n0n0,%2Fg%2F11bc5kmsh6,%2Fm%2F06znsr5. [Accessed 22 05 2020].

[79] "Stack Overflow Trends," [Online]. Available: https://insights.stackoverflow.com/trends?tags=flutter%2Creact-native%2Cionic-framework%2Cphonegap-build. [Accessed 22 5 2020].

[80] "cordova.apache.org," [Online]. Available: https://cordova.apache.org/. [Accessed 2 July 2020].

[81] "Adobe PhoneGap, Embedding the Webview," [Online]. Available: http://docs.phonegap.com/tutorials/develop/1-embed-webview/ios/. [Accessed 2 July 2020].

[82] "ionic, UI Components," [Online]. Available: https://ionicframework.com/docs/components. [Accessed 2 July 2020].

[83] P. Walsh, "METACERT, What is a WebView?," 22 April 2015. [Online]. Available: https://developer.metacert.com/blog/what-is-a-webview/. [Accessed 2 July 2020].

[84] Codemagic, "Medium, Flutter vs React Native: A Developer's Perspective," 24 January 2019. [Online]. Available: https://medium.com/flawless-app-stories/flutter-vs-react-native-a-developers-perspective-8914ca240a89. [Accessed 2 July 2020].

[85] P. Patel, "FasTrax INFO TECH, React Native vs. Ionic vs. Flutter vs. PhoneGap – What to Choose in 2020?," 21 4 2020. [Online]. Available: https://www.ftxinfotech.com/blog/react-native-vs-ionic-vs-flutter-vs-phonegap/. [Accessed 22 5 2020].

[86] "flutter.dev, Add Flutter to existing app," [Online]. Available: https://flutter.dev/docs/development/add-to-app. [Accessed 2 July 2020].

[87] "reactnative.dev, Integration with Existing Apps," [Online]. Available: https://reactnative.dev/docs/integration-with-existing-apps. [Accessed 2 July 2020].

[88] A. Sharma, "Medium, Flutter v/s React native and Phonegap frameworks," 4 4 2019. [Online]. Available: https://medium.com/@atul.sharma_94062/understanding-difference-between-flutter-react-native-and-phonegap-frameworks-7721229068d5. [Accessed 22 5 2020].

[89] M. Bowers, "toptotal.com, Creating a UI Style Guide for Better UX," [Online]. Available: https://www.toptal.com/designers/ui/ui-styleguide-better-ux. [Accessed 2 July 2020].

[90] "w3c Standards webdesign," [Online]. Available: https://www.w3.org/standards/webdesign/. [Accessed 22 June 2020].

[91] "developer.apple.com human interface guidelines," [Online]. Available: https://developer.apple.com/design/human-interface-guidelines/. [Accessed 22 June 2020].

[92] "developer.android.com Design," 27 December 2019. [Online]. Available: https://developer.android.com/design. [Accessed 23 June 2020].

[93] "flutter.dev, Material Components widgets," [Online]. Available: https://flutter.dev/docs/development/ui/widgets/material. [Accessed 2 July 2020].

[94] "flutter.dev, Introduction to widgets," [Online]. Available: https://flutter.dev/docs/development/ui/widgets-intro. [Accessed 2 July 2020].

[95] "flutter.dev, Cupertino (iOS-style) widgets," [Online]. Available: https://flutter.dev/docs/development/ui/widgets/cupertino. [Accessed 2 July 2020].

[96] M. Saeed, "Medium, Flutter: Material + Cupertino make together ❤️," 11 November 2019. [Online]. Available: https://medium.com/@sendtosaeed2/flutter-material-cupertino-make-together-a3d2d7849548. [Accessed 23 June 2020].

[97] S. Kulinski, "Medium, Do Flutter apps dream of platform aware widgets?," 1 March 2018. [Online]. Available: https://medium.com/flutter/do-flutter-apps-dream-of-platform-aware-widgets-7d7ed7b4624d. [Accessed 23 June 2020].

[98] "material.io," [Online]. Available: https://material.io/. [Accessed 22 5 2020].

[99] "material.io about shape," [Online]. Available: https://material.io/design/shape/about-shape.html. [Accessed 26 June 2020].

[100] "material.io, Introduction," [Online]. Available: https://material.io/design/introduction#principles. [Accessed 2 July 2020].

[101] E. Perez, "El androide libre, ¿Qué es Material Design?," 9 November 2014. [Online]. Available: https://elandroidelibre.elespanol.com/2014/11/que-es-material-design.html. [Accessed 2 July 2020].

[102] "flutter.dev," [Online]. Available: https://flutter.dev/. [Accessed 23 5 2020].

[103] "Wikipedia, Flutter," [Online]. Available: https://en.wikipedia.org/wiki/Flutter_(software). [Accessed 2 July 2020].

[104] "GitHub octoverse," 2019. [Online]. Available: https://octoverse.github.com/#top-languages. [Accessed 22 5 2020].

[105] "Flutter comunity," [Online]. Available: https://flutter.dev/community. [Accessed 2 July 2020].

[106] "pub.dev," [Online]. Available: https://pub.dev/. [Accessed 25 June 2020].

[107] "Wikipedia, Dart (programming language)," [Online]. Available: https://en.wikipedia.org/wiki/Dart_(programming_language). [Accessed 2 July 2020].

[108] "Dartium: Chromium with the Dart VM," [Online]. Available: https://dartdoc.takyam.com/tools/dartium/. [Accessed 2 July 2020].

[109] "dart2js: Dart-to-JavaScript compiler," [Online]. Available: https://dart.dev/tools/dart2js. [Accessed 2 July 2020].

[110] "dart.dev," [Online]. Available: https://dart.dev/. [Accessed 26 May 2020].

[111] "dart.dev, Platforms," [Online]. Available: https://dart.dev/platforms. [Accessed 2 July 2020].

[112] Flutter, "Why Flutter uses Dart," 12 November 2019. [Online]. Available: https://www.youtube.com/watch?v=5F-6n_2XWR8. [Accessed 2 July 2020].

[113] K. Walrath, 25 July 2019. [Online]. Available: https://medium.com/dartlang/dart-asynchronous-programming-isolates-and-event-loops-bffc3e296a6a. [Accessed 2 July 2020].

[114] A. Brogdon, "Youtube Video: Isolates and Event Loop - Flutter in Focus," 14 June 2019. [Online]. Available: https://www.youtube.com/watch?reload=9&v=vl_AaCgudcY. [Accessed 27 May 2020].

[115] Alibaba Tech, "Medium, Making the Most of Flutter: From Basics to Customization," 1 August 2018. [Online]. Available: https://medium.com/hackernoon/making-the-most-of-flutter-from-basics-to-customization-433171581d01. [Accessed 2 July 2020].

[116] "Flutter engine," [Online]. Available: https://chromium.googlesource.com/external/github.com/flutter/engine/. [Accessed 29 June 2020].

[117] K. Tolmachev, "What is Flutter and why it is a game-changer in app development," 3 December 2019. [Online]. Available: https://blog.quickblox.com/what-is-flutter-and-why-it-is-a-game-changer-in-app-development/. [Accessed 2 July 2020].

[118] "medium.com The layer cake," 21 December 2018. [Online]. Available: https://medium.com/flutter-community/the-layer-cake-widgets-elements-renderobjects-7644c3142401. [Accessed 29 June 2020].

[119] "flutter.dev/docs Technical overview," [Online]. Available: https://flutter.dev/docs/resources/technical-overview. [Accessed 23 June 2020].

[120] "flutter.dev, Start thinking declaratively," [Online]. Available: https://flutter.dev/docs/development/data-and-backend/state-mgmt/declarative. [Accessed 2 July 2020].

[121] M. A. Rodríguez, "Desarrollo de Aplicaciones Móviles con Flutter," Apuntes D.A.D.M. E.T.S.I.T. UVA, Valladolid.

[122] Google developers, "Building your first Flutter Widget," 18 May 2018. [Online]. Available: https://www.youtube.com/watch?v=W1pNjxmNHNQ. [Accessed 2 July 2020].

[123] Google Developers, "Inherited Widgets Explained - Flutter Widgets 101 Ep. 3," 19 November 2018. [Online]. Available: https://www.youtube.com/watch?v=Zbm3hjPjQMk&t=118s. [Accessed 2 July 2020].

[124] Flutter, "How Flutter renders Widgets," 15 November 2019. [Online]. Available: https://www.youtube.com/watch?v=996ZgFRENMs&t=323s. [Accessed 2 July 2020].

[125] Google Developers, "When to use Keys - Widget Flutter 101 Ep. 4," 26 November 2018. [Online]. Available: https://www.youtube.com/watch?v=kn0EOS-ZiIc&t=320s. [Accessed 2 July 2020].

[126] "Adding interactivity to your Flutter app," [Online]. Available: https://flutter.dev/docs/development/ui/interactive. [Accessed 2 July 2020].

[127] "flutter.dev, StatefulWidget class," [Online]. Available: https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html. [Accessed 2 July 2020].

[128] F. García, "elmundo.es/motor, El servicio de carsharing Zity inicia operaciones en Madrid a 21 céntimos el minuto," 20 December 2017. [Online]. Available: https://www.elmundo.es/motor/2017/12/18/5a381a13268e3e90268b45f2.html. [Accessed 24 June 2020].

[129] "Ubeeqo app on Play Store," [Online]. Available: https://play.google.com/store/apps/details?id=es.sw.bluemove&hl=en. [Accessed 29 May 2020].

[130] "media.groupe-psa.com Free2Move Paris extends its car sharing service to Issy-les-Moulineaux," 12 March 2019. [Online]. Available: https://media.groupe-psa.com/en/free2move-paris-extends-its-car-sharing-service-issy-les-moulineaux#:~:text=Free2Move%20is%20a%20free%2Dfloating,spaces%20(ex%2D%20Autolib).. [Accessed 24 June 2020].

[131] darticulate.com, "pub.dev mqtt_client," 13 July 2019. [Online]. Available: https://pub.dev/packages/mqtt_client. [Accessed 2 July 2020].

[132] S. Hamblett, "github.com: mqtt_client," [Online]. Available: https://github.com/shamblett/mqtt_client/blob/master/example/mqtt_server_client.dart. [Accessed 26 June 2020].

[133] soft2tec.com, "pub.dev/packages universal_mqtt_client," 16 June 2020. [Online]. Available: https://pub.dev/packages/universal_mqtt_client. [Accessed 2 July 2020].

[134] "flutter.dev/docs shared preferences," [Online]. Available: https://flutter.dev/docs/get-started/flutter-for/android-devs#how-do-i-access-shared-preferences. [Accessed 25 June 2020].

[135] appmire.be, "pub.dev pacakges flutter_keychain," 5 March 2019. [Online]. Available: https://pub.dev/packages/flutter_keychain. [Accessed 27 June 2020].

[136] "flutter.dev/docs Internationalizing Flutter apps," [Online]. Available: https://flutter.dev/docs/development/accessibility-and-localization/internationalization. [Accessed 3 June 2020].

[137] "flutter.dev/docs Adding assets and images," [Online]. Available: https://flutter.dev/docs/development/ui/assets-and-images. [Accessed 27 June 2020].

[138] "flutter.dev/docs state management," [Online]. Available: https://flutter.dev/docs/development/data-and-backend/state-mgmt/options. [Accessed 29 June 2020].

[139] Flutter, "Pragmatic State Management in Flutter (Google I/O'19)," 9 May 2019. [Online]. Available: https://www.youtube.com/watch?v=d_m5csmrf7I&t=3s. [Accessed 2 July 2020].

[140] Flutter, "Pragmatic State Management Using Provider (The Boring Flutter Development Show, Ep. 24)," 3 July 2019. [Online]. Available: https://www.youtube.com/watch?v=HrBiNHEqSYU. [Accessed 3 July 2020].

[141] "sae.org," 11 december 2018. [Online]. Available: https://www.sae.org/news/press-room/2018/12/sae-international-releases-updated-visual-chart-for-its-%E2%80%9Clevels-of-driving-automation%E2%80%9D-standard-for-self-driving-vehicles. [Accessed 18 June 2020].

[142] T. N. a. R. Wu, "linkedin.com, Mobile App Security: Native v.s. Flutter," 22 July 2019. [Online]. Available: https://www.linkedin.com/pulse/mobile-app-security-native-vs-flutter-rick-wu/. [Accessed 25 June 2020].

ANNEX 1

This annex shows the table of messages that the app exchanges with the server and the vehicles. The table columns detail the message publisher, subscriber, topic, payload, and a brief explanation as well. The broker redirects to the app the messages between the Twizy and the server when in the commands screen. The topics are explained in 3.2.

Publisher	Subscriber	Topic	Payload	Explanation
Twizy	Server	vehicleID/battery	[0-100]OR[-1]	-1 when no battery signal from the vehicle is received
Twizy	Server	vehicleID/location	[(latitude),(longitude)]OR[No signal]OR[GPS not connected]	Real time vehicle position
Twizy	Server	vehicleID/info	CONNECT regist_plate	Connection request
Twizy	Server	vehicleID/info	TIMEOUT	Obstacle timer exceeded
Twizy	Server	vehicleID/info	RFID (ID number)	Identification number of the RFID tag detected
Twizy	Server	vehicleID/info	WRN (error code) [(attribute)]	Warning info
Twizy	Server	vehicleID/info	ERR (error code) [(attribute)]	Error info
Twizy	Server	vehicleID/info	STARTING UP	System is starting up
Twizy	Server	vehicleID/info	AM-ON OK	The vehicle has correctly switched to autonomous mode
Twizy	Server	vehicleID/info	AM-OFF OK	The vehicle has switched to normal mode correctly
Twizy	Server	vehicleID/info	CONTINUE OK	The vehicle will continue
Twizy	Server	vehicleID/info	PAUSE OK	The vehicle will stop
Twizy	Server	vehicleID/info	STANDBY OK	The vehicle has correctly switched to standby mode
Twizy	Server	vehicleID/info	GOTO OK	Correct reception of the GOTO message. The vehicle begins to move.
Server	Twizy	vehicleID/order	CONNECTED	Established connection
Server	Twizy	vehicleID/order	STANDBY	Enter power save mode
Server	Twizy	vehicleID/order	AM-ON	Go to autonomous operation
Server	Twizy	vehicleID/order	AM-OFF	Go to normal operation
Server	Twizy	vehicleID/order	CONTINUE	Continue after sending PAUSE message
Server	Twizy	vehicleID/order	PAUSE	Stop the car in autonomous operation

Server	Twizy	vehicleID/order	GOTO (initialSpeed km/h) (defaultFork L o R) [T(RFID) [(newSpeed km/h)]...] [V(RFID) (newSpeed km/h)...] S(RFID)	Route that the vehicle has to follow. T=Turn right until the next rfid S=Stop
Server	Twizy	vehicleID/order	RESTART	Restarts vehicle after ERROR
Application	Server	log	LOGIN/user/password	Information about the client is sent
Application	Server	log	LOGON/user/name/surname/employment/password	Information about new client to log on is sent
Server	Application	log	CONFIRM/username	Verifies and accepts the data writen by the client
Server	Application	log	DENY/username/ErrCode	Verifies and denies the data writen by the client
Server	Application	user/app	ADMIN/0 o 1	In orden to check if the user is an admin
Application	Server	user/server	REQUEST-PARKS	The servers responses with as many NEW-PARK messages as parkings it has in the database
Server	Application	user/app	NEW-PARK/id_parking/name_park/latitud/longitud/availableTwizys/freeSpots	New parking to show in the map
Server	Application	app	INFO-PARK/id_parking/[availableTwizys]/[freeSpots]	Periodical information to refresh the data about the origin and destiny parkings
Application	Server	user/server	PICK-UP/id_parking	The client gets out the Twizy
Server	Application	user/app	TWIZY-OUT /id_vehicle	Confirms that the Twizy is going to the pick-up zone and informs the vehicle license plate
Application	Server	user/admin_server	REQUEST-ERRORS	The server asnwers in topic user/admin with as many ERR messages as unresolved erros it has in its database
Server	Admin	admin or user/admin	ERR/id_fix/[id_vehicle o id_parking]/fail/explanation/error_date/more_info	An error will be sent to the app if something is going wrong
Admin	Server	user/admin_server	ERR-FIX/id_fix	Admin says that one error has been already fixed
Admin	Server	user/admin_server	ASK-PARK/id_parking	Admin asks for the status of a specific parking
Server	Admin	user/admin	PARK/num_row/num_column/[(row_pos;column_pos;id_car)/...]	Control over the parkings for the admin
Server	Admin	user/admin	PARK-ERR/ErrCode	Just in case the admin selects an inexistent parking
Admin	Server	user/admin_server	ASK-CAR/id_car	Admin asks for information about a specific car
Server	Admin	user/admin	CAR/enter_date;id_parking;exit_date;user/.../...	Server provides the information about in which parkings and which users have taken a specific car
Server	Admin	user/admin	CAR-ERR/ErrCode	Just in case the admin selects an inexistent car

ANNEX 2

This annex shows the different errors that vehicles and the server can emit. When a vehicle sends an error, it is processed by the server, who stores it in its database and broadcast it to any connected app user too. Those errors can be seen in the error log screen, where there will be more information about the error provided by the server. The vehicle errors can be seen in the app when in the commands screen too. In this case only the decimal code of the error will be displayed in the window at the bottom of this screen. The server errors are displayed in the app on different situations. For instance, the "username is not unique error" would appear in the logon screen when a new user tries to log on with a username already in use. Those errors are translated to the languages supported by the app.

Source	Dec Code	Hex Code	Type of error	Failure name	Attribute	Explanation
Communication Module	1	0x001	Fault	CON-STATUS not received	-	The CON_STATUS frame is not being received. Is the CON-MOD on? Is the CAN-Bus connection of both modules correct?
Communication Module	2	0x002	Fault	Control Module does not respond to CON_STATUS signal - Cannot change status	-	CON-MOD does not change status when ordered to do so. Is the CON-MOD correctly configured?
Communication Module	3	0x003	Fault	Control Module does not respond to PAUSE signal	-	When sending a PAUSE or CONTINUE command from the COM-MOD, the CON-MOD does not respond correctly as the PAUSE signal does not change. Is the CON-MOD correctly configured?
Communication Module	4	0x004	Fault	GOTO-ACK not received after several attempts	-	CON-MOD does not send an ACK when it has to do so. Is the CON-MOD correctly configured?
Communication Module	5	0x005	Warning	Malformed GOTO frame received from Communication Module	-	
Communication Module	6	0x006	Fault	Unexpected frame from Control Module	ID of frame received	CON-MOD is sending out signals that don't make sense right now. Is CON-MOD well configured?
Communication Module	9	0x009	Fault	GPS not connected	Last coordinates (if any)	The GPS module is not connected to the communication module. Is the COM-MOD correctly configured? Is the GPS broken? Is the GPS disconnected?
Communication Module	10	0x00A	Warning	GPS is not receiving signal for a long time	Last coordinates (if any)	The GPS module is not receiving any signal from the satellites for a long time. Is the GPS broken?

Communication Module	11	0x00B	Warning	Battery data not received	Last battery data received (if any)	The CAN message informing about the battery charge on the vehicle's CAN-bus is not detected. Is there a problem in the vehicle?
Communication Module	17	0x011	Fault	Network is unreachable	-	COM-MOD is not connected to the internet. Is the COM-MOD connected to the Internet?
Communication Module	18	0x012	Fault	Connection refused - Mosquitto service is not running	-	MQTT messages are arriving at a machine where the mosquito service is not running. Is the mosquitto service running on the server?
Communication Module	19	0x013	Fault	Can't connect with server	-	MQTT messages are not arriving at any machine. Is the IP address right?
Communication Module	20	0x014	Fault	Connection refused - Not authorised	-	The MQTT broker is rejecting the connection due to no authentication. Is the client sending the correct authentication data?
Communication Module	21	0x015	Fault	Connection refused - Bad username or password	-	The MQTT broker is rejecting the connection due to a bad authentication. Is the client sending the correct username and password?
Communication Module	22	0x016	Fault	Connection refused - Server unavailable	-	Not tested
Communication Module	23	0x017	Fault	Connection refused - Invalid client identifier	-	Not tested
Communication Module	24	0x018	Fault	Connection refused - Incorrect protocol version	-	Not tested
Communication Module	25	0x019	Fault	Back-end is not responding	-	The server is not responding with the message CONNECTED when called. Is the server running?
Communication Module	26	0x01A	Warning	Unexpected MQTT message	Message received	The server is sending an incorrect message. Is the server well configured?
Communication Module	27	0x01B	Fault	Lost connection with MQTT server	-	Ping responses from the server are not beeing received. Has CON-MOD lost the internet connection? Has the MQTT broker stopped?
Control Module	129	0x081	Fault	COM_STATUS not received	-	The COM_STATUS frame is not being received. Is the COM-MOD on? Is the CAN-Bus connection of both modules correct?
Control Module	130	0x082	Fault	RFID-ACK not received	-	
Control Module	131	0x083	Fault	CON-ERR-ACK not received	-	

Control Module	132	0x084	Warning	Unexpected frame received from Communication Module	ID of frame received	
Control Module	133	0x085	Warning	Malformed GOTO frame received from Communication Module	-	
Server	257	0x101	ERROR	Username is not unique		The username passed on the LOGON is not vaild because it is already in use
Server	258	0x102	ERROR	Parking busy. Try again to get your Twizy		In that moment the origin Park is busy, so the user must try again.
Server	259	0x103	Warning	That parking id does not exit		Administrator is asking for information about a parking which does not exit.
Server	260	0x104	Warning	That car id does not exit		Administrator is asking for information about a Twizy which does not exit.