Universitat Politècnica de Catalunya

Information Technology Project

Project Report

Talos

Authors
Llonch Majó Joan
Valverde Zavaleta Luís Jesús
Vidal Sulé Guillermo
Vilella Jam Oriol

June 2, 2024

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Contents

List of Figures

1 Introduction

Safety and security are foundational concepts of any civilization. Ferocious beasts, meteorological cataclysms or simply other men; society cannot blossom without the certainty that, whatever you build today, will not be shattered tomorrow. Therefore, in parallel with many other inventions, the human mind began analyzing potential ways of reducing risks and their impact. Talos is a project based on that idea: ensuring the safety of what you must protect.

The course had multiple constrains regarding the project, primarily focused around the interconnection of multiple and diverse technologies. Consequently, it was envisioned that Talos would encompass components from the highest levels of abstraction (software) to the lowest (electronics). Hence, we decided upon developing a complex robot from scratch, whilst also providing mobile and website interfaces so as to manage it. In other words, this project is a security system, in its entirety.

Although we did not manage to finish most of our objectives, we worked for countless hours while learning many technologies. The software components, which required a lot of work, were almost finished, but the hardware is far from being completed. Regardless, we believe the final delivery was a huge success, as we manage to engineer part of a very complex and intricate system, whilst also providing high-quality documentation.

Since this project has too many concepts and relevant information to expound upon, the following report has been simplified to an overview of its major parts. Each part details their main components, technologies, results and their potential improvements.

2 Important note

Since this document is limited in size, their will be a lot of information which will be omitted. Furthermore, since we already compiled some documentation and tutorials, they will be provided in a github [1] with all the code, but please note that this will take somewhat longer (hopefully we will make it public soon). Hence, this report will only contain the most relevant information. Moreover, certain topics such as the building steps of the robot, which are far too complex and detailed, will be skipped. This is a warning that, regardless of the contents of this writing, it is far from explaining all

the details of Talos. Please take this into consideration.

3 Architecture

To have an overview of all of the project's components and how they interact with each other, figure 1 is provided.

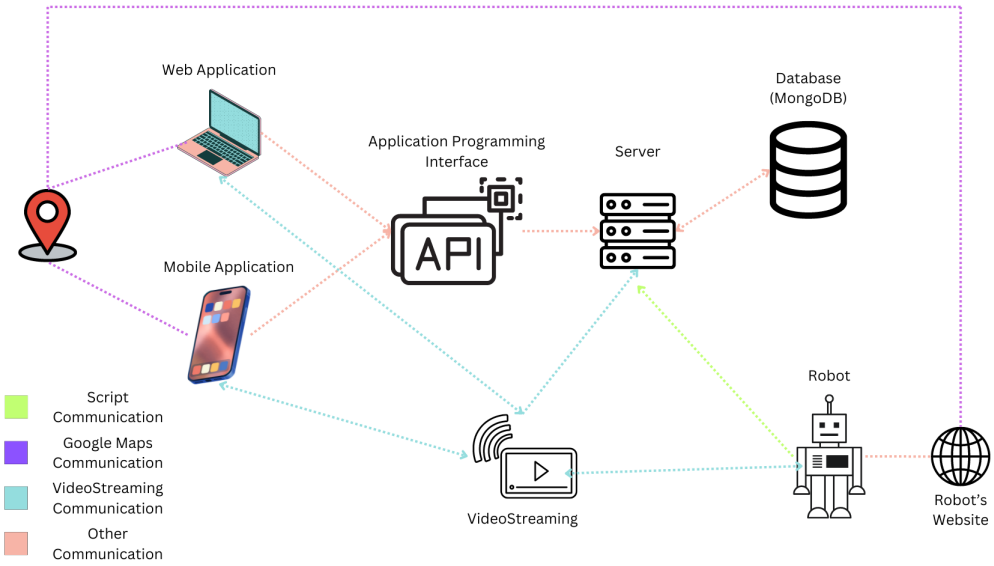

Figure 1: Architecture of Talos. [own compilation]

4 The robot

This section is aimed to describe the main functionalities of the robot and core attributes.

4.1 Initial idea

The robot is the accumulation of different components which should should complement each other. It required the engineering of all of its aspects, as

it should also be compatible with the website and mobile application, whilst maintaining a certain degree of security.

In its inception, the robot was expected to be able to move according to a set of coordinates programmed from a user interface. Furthermore, it should be able to stream the surveillance in real time, whilst also providing some type of AI to detect possible intruders. It would also include sensors responsible for object detection to adjust the route when necessary, as obstacles have to be dodged effectively.

However, the project was far from completed, as it is very complex and time consuming, so only some basic functionalities were implemented.

4.2 How does it work?

This subsection is dedicated to explaining the basics of how the entire system of the robot is expected to work. From programming the route to adapting its movement. Notably, we will already consider that the robot is linked to a user and connected to the internet.

1. Talos is stationary, linked to a user and ready for use. The switch should be turned on and a 9V power supply should be connected to the Arduino.

2. The user reprograms the surveillance route in any interface (website or application). The interface forwards this information to the Lichee Pi 4A through the Internet.

3. The Lichee Pi 4A receives the necessary information, which is sent to the Arduino via UART, a communication protocol.

4. The Arduino UNO, which is constantly polling for information on the Rx pins, receives the message and proccesses it. This is further described in the Doxygen documentation. In short, certain bits are dedicated to codify the type of operation to be performed and others are for the information (think of it like an @IP package, part of it is used to describe how it should be treated and the rest is dedicated to the payload).

5. The route is stored in a vector and the following movement is calculated. The GPS is used to determine the robot's current position and the

navigation sensor is needed to know where it is pointing at. In order to calculate the angle the robot should rotate we need: the unitary vector describing where the robot is pointing at[1] and the unitary vector that represents the distance from its current location to the route's next checkpoint[2]. Finally the following basic formula is used to calculate the rotation angle:

$$cos(\alpha) = \frac{a \cdot b}{||a|| \cdot ||b||}$$

6. Talos activates its wheels to rotate. Having reached the desired angle, the forward movement is started until reached the destination (`currentLocation = checkpointLocation`).

7. The process of calculating the angle and moving is then indefinitely repeated, as their is no battery control, so it would move until it ran out of power.

Some other functionalities such as reading information from the sensors and sending it back to the database has a similar process. A message is sent through the UART interface from the Lichee to the UNO, with the corresponding bits set so that the UNO understands the Lichee is requesting information. Next, the corresponding sensors are read and the information is sent back to the Lichee Pi 4A.

Please check the Github [1] for more information, there are many interesting aspects that are not described here, such as the processing of NMEA messages (what you read when fetching information from the GPS sensors).

4.3 Design

Designing the robot was a complex task, as it required planning an efficient robot which could move and was of a reasonable size. It also required to decide on where and how to place the multiple sensors and components. Sadly, there is no general schematic of the robot, as we lacked the time and skills, so only the 3D printed parts are available.

[1]The navigation sensor is used to determine the robot's angle compared to magnetic North.

[2]It can be calculated by subtracting one point from another and then using basic mathematics to find the angle.

The final design required of us to try multiple implementations as we kept running into problems. For example, the first holders of the DC Motors held them too high, so their weight ended up bending the robot's chassis. Moreover, in multiple occasions the screw wholes had erroneous sizes or where not in the correct place.

Moving on to the movement, we decided on the omnidirection vehicle, which uses **mecannum** wheels. These wheels would allow the robot to move in different directions without having to change their angle, like the cars do. It was designed this way to avoid designing the mechanics of normal turning wheels, they are quite complex and hard to implement. In other words, using mecannum wheels reduced the engineering load off the project.

Lastly, the wheels and camera holder were printed using PLA, the most common 3D printing filament. Since it is widely available and is known for its reliability it was decided as the perfect choice. However, for the chassis and wheel holders we used PETG, as it is far more resistant, but it takes longer to print. The chassis needs to be robust and consumes the most filament, so it is important to make it as strong as possible, as it is also responsible of protecting the electronic components.

4.4 Materials

The following list includes the most important materials that are needed to build the robot. It does not include any tools such as screwdrivers and the 3D printer.

- **Lichee Pi 4A** `(1)` [2] - Main SBC responsible of the core functionalities. Used to communicate with the user interfaces and with the Arduino UNO to fetch data from the sensors. Moreover, it also managed the USB camera and hosted the local robot configuration website. It was chosen as it is RISC-V and has a dedicated NPU, which could have been used to run the AI code. Using an SBC intended to reduce the workload of the UNO and simplify the programming of some tasks, as the SBC runs an operating system. The OS is Sipeed's own implementation of Debian, which I modified and recompiled so as to enable the other UART interfaces[3].

[3] I burnt UART1, which is needed to communicate with the UNO, so I had to enable UART2 and UART3. It was fixed by modifying the device tree and recompiling the kernel.

- **Arduino UNO** `(1)`- Development board used to manage most of the sensors. Although we preferred some other more interesting alternatives, Arduino IDE offers a lot of support and libraries, which simplified the programming and configuration.

- **L298N Motor controller** `(2)` [3] - This was needed to control the wheels, primarily because the voltage needed would be too high for the Arduino to supply on its own. Please note that the voltage supplied by the batteries is $> 12V$, so the jumper controlling its internal regulator must be removed and an external 5V supply added (by the Arduino, for example).

- **755 DC Motor** `(4)` [4] - Since it is a mecannum car it needs one motor per wheel. Moreover, since the robot is big and heavy, a powerful motor with high torque was chosen, the 755. Additionally, we decided on a DC motor as it is the simplest one and was a perfect fit for the needs of our project[4].

- **GPS module** `(1)` [5] - Responsible of knowing the robot's current location. It is needed to make Talos an autonomous vehicle and make it follow the route traced by the user.

- **Navigation module** `(1)` - Understanding where the robot is pointing at is paramount before adjusting the movement. It is used to complement the GPS. They are both needed to make the robot autonomous.

- **PLA / PETG printing filament** - Needed to print all the robot parts.

- **M2, M3, M4 and M8 bolts, nuts and washers (x)** - Parts required to assemble the multiple pieces. Bolts were needed in different lengths as well, but in this report the list will not be exhaustive.

- **15.2V batteries** `(2)` [6] - Each motor controller can supply two wheels, so 2 batteries were needed. It can obviously be simplified to have one common power supply (and also put the batteries in series and achieve 30.4V), but our knowledge is limited and wanted to maintain the design simple.

[4]Servo and stepper motors can control the position of the wheels, but it is pointless in this case.

- **XT90 to cable adapter** `(2)` [7] - The battery connector (XT90 female) needed to be adapted to something that we could easily connect to the motor controller or switch.
- **AWG cable** `(1)` [8] - This cable, which supports high current (remember we are using high voltage batteries with 6500mAh), is used to connect the DC motors and the motor controller[5].
- **USB camera** `(1)` [9] - Needed to for the video streaming.
- **Others** - PCB, regulator, switch and many other components, but since they are not that relevant and they would occupy too much space they are exempted from this list.

5 Development tools

Many 3rd party tools were used to design the robot, this is the list:

- **Fusion360** - To design and edit 3D printed parts. It also enabled us to make the prototype of the PCB, but we were unable to print it in time.
- **TinkerCAD** - Useful to quickly modify certain 3D models such as the DC motor holder.
- **Bambu Labs Studio** - Used to prepare the 3D models for the 3D printer (Bambu Labs P1S).
- **Arduino IDE** - IDE in which the Arduino was programmed and flashed.

5.1 Electronics

The electronics of the project encompassed the development of a circuit that integrated all sensors and electrical components. To connect the Arduino UNO and Lichee Pi 4A, a specialized circuit was required to bridge their UART voltage levels of 5V and 1.8V, respectively. However, due to time constraints, this specific circuit could not be developed.

[5]It would be cut with scissors, then we would remove with wire strippers the protector from both ends and finally solder all the filaments together.

5.2 Programming

The code was done in the C++ adaptation made by Arduino, which makes it quite simple. Everything was documented using Doxygen, but this is a topic for another section.

In first place, the programming was based on the paradigm that different modules should be reusable and that all different classes should provide interfaces at different levels. In other words, there would be low-level functions which could be used for precise control over the root and higher level ones, which would perform a certain set of tasks, so that it fit the needs of more and less experienced developers[6]

5.3 Results

The results were somewhat underwhelming, as we were unable to make the robot move. Nonetheless, we were able to control the wheels movement using the UNO through the motor controller. Moreover, it was also possible to read the information from navigation, GPS and speed sensors. However, the communication between the UNO and Lichee Pi 4A was not finished in time, which made it impossible for the user to visualize most of the information.

6 Website

6.1 Objectives

The aim of this website is to provide the user with an intuitive and easily-accessible platform through which it can manage and visualize the robots an their corresponding information. Furthermore, it is also meant to offer specific functionalities, such as being able to customize the surveillance route each robot will trace. Additionally, since this project was undertaken from the perspective of a corporation, it is paramount to ensure the interface is appealing to potential customers. Hence, its main objectives include intuitiveness, attractiveness and powerful display.

[6]For example, there is a function `start()` which initializes the whole robot and sets it in motion, which makes it easy to start programming it. Nonetheless, there are other methods which can perform very specific operations such as set the direction of one particular wheel.

The following aspects and technical details where taken into consideration when designing the website[7]:

- **Real-time visualization** - The user must be able to watch certain parameters of the robot such as CPU Frequency, CPU temperature, speed and camera. In addition, the website need to be able to display the map of where the robot is going and moreover to show the graphs of the variables.
- **Popularize the product** - The aim of the website was also to create a well-structured project that resembles the reality of an actual project as closely as possible and that's why having a website was essential for both understanding and marketing the product.

6.2 Website design

The web design is tailored to meet specific functional and not functional requirements, ensuring a seamless user experience. These requirements encompass usability, excellent user experience, real-time metrics, appealing design, navigational ease, quality content, effective website operation, and user satisfaction. Each aspect is meticulously integrated into the design framework to achieve optimal performance and user engagement.

- **Usability:** The design prioritizes usability by implementing intuitive navigation, clear layout structures, and user-friendly interactions, enabling users to effortlessly engage with the platform.
- **Exceptional User Experience:** Focused attention is placed on crafting an exceptional user experience through thoughtful design elements, responsive interfaces, and seamless interactions, fostering a positive and memorable user journey.
- **Real-time visualization** [8]
- **Aesthetic Appeal:** Aesthetic appeal is a cornerstone of the design philosophy, with careful consideration given to visual elements, typography, color schemes, and imagery to create an engaging and visually captivating interface.

[7]Notably, video streaming is expounded upon in detail in section **??**, as its importance requires an explanation of its own.

[8]Notably,Real-time visualization is expounded upon in detail in section 5.1

- **Navigational Ease:** Efforts are made to ensure navigational ease by implementing logical navigation paths, clear signposting, and intuitive menu structures, allowing users to effortlessly explore and navigate the website.
- **User Satisfaction:** Ultimately, the design is crafted with a singular focus on user satisfaction, aiming to exceed expectations, fulfill user needs, and leave a lasting positive impression, thereby cultivating loyalty and trust among users.

6.2.1 Utilized technologies

The selection of technologies for the web component was driven by a combination of personal interest and practical considerations.JavaScript and React were chosen as the foundation due to a genuine interest in mastering web development and leveraging modern frameworks for building dynamic user interfaces.The wealth of documentation, tutorials, and active developer communities associated with these technologies provided invaluable resources for accelerating development, troubleshooting issues, and staying abreast of best practices in web development.

6.3 Web development

To comprehend the construction of the entire website application, it is imperative to begin by examining the content of our App.js file. This file serves as the principal component of the application, primarily because it is where all other components of the user interface are rendered. Furthermore, within this file, we establish the routing and navigation for all views or pages of the web application, thereby dictating the flow of user interaction.

```jsx
import React from 'react';
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import SignIn from './pages/Signin/SignIn';
import Home from './pages/Home/Home';
import SignUp from './pages/Signup/SignUp';
//import Dashboard from './scenes/Dasboard2'; // Importa el componente del dashboard
import './app.css';
import Dashboard2 from './scenes/Dashboard2';
import RobotStatistics  from './scenes/RobotStatistics';
import { APIProvider } from '@vis.gl/react-google-maps';

const App = () => {
  return (
    <div>
      <BrowserRouter>
        <Routes>
          <Route index element={<Home />} />
          <Route path="/home" element={<Home />} />
          <Route path="/signin" element={<SignIn />} />
          <Route path="/signup" element={<SignUp />} />
          <Route path="/dash" element={<Dashboard2 />} />
          <Route path="/robots/:robotname" element={<RobotStatistics />} />
        </Routes>
      </BrowserRouter>

    </div>
  );
}

export default App;
```

As we can see in the picture we have six different routes, Home which is the root page, which serves as the initial page displayed in the browser, from which users can navigate to other pages.Afterward, we have the sign-in and sign-up forms, which serve as the login and registration forms, respectively. In Dashboard2, we present the user dashboard, and in Robot Statistics, we display the statistics for each individual robot associated with that user. We will address these components in detail in this sequence, as it reflects the order a typical user might follow while navigating through our website.

6.3.1 Home

The Home component is designed to create all the front-end elements that will be available on the Internet, accessible to anyone with an Internet connection. Consequently, we decided to make it a responsive and visually appealing website to encourage user engagement and drive registrations and purchases.

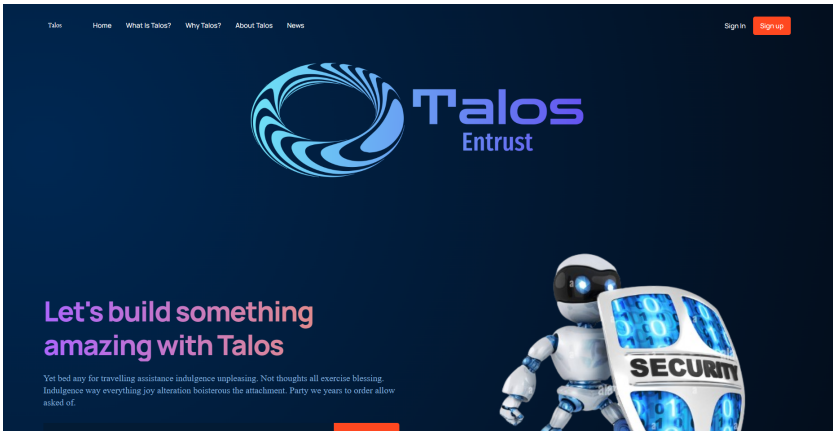

Figure 2: Web header

At the top of the page, as shown in the image, there is a navigation bar that redirects users to the corresponding sections of the page for each name. Additionally, there are two buttons in the upper right corner where users can register or log in.

In the center, we have the project logo, as it is important to introduce our project. The current image is a placeholder; we initially intended to use a photograph of our robot. However, due to construction delays, we selected this picture to provide users with an idea of our project's essence.

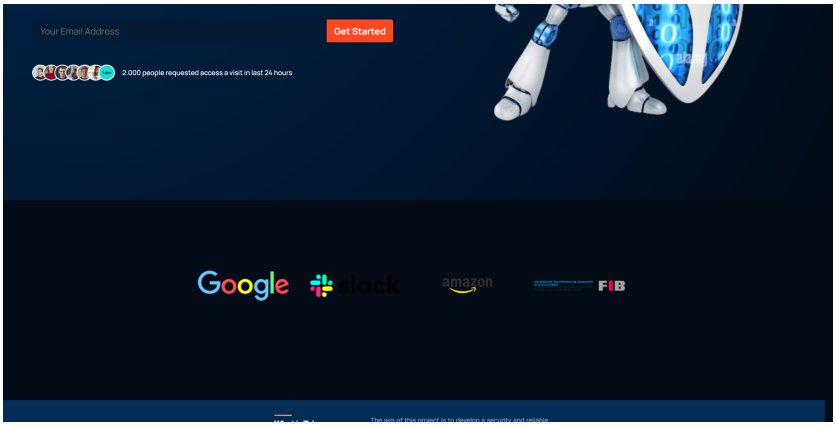

Figure 3: Web Brand

In the web's brand section, we selected some of the more popular brands to give our site a sense of authenticity and credibility. During the plan-

ning phase, we researched several well-known websites to gather ideas and inspiration.

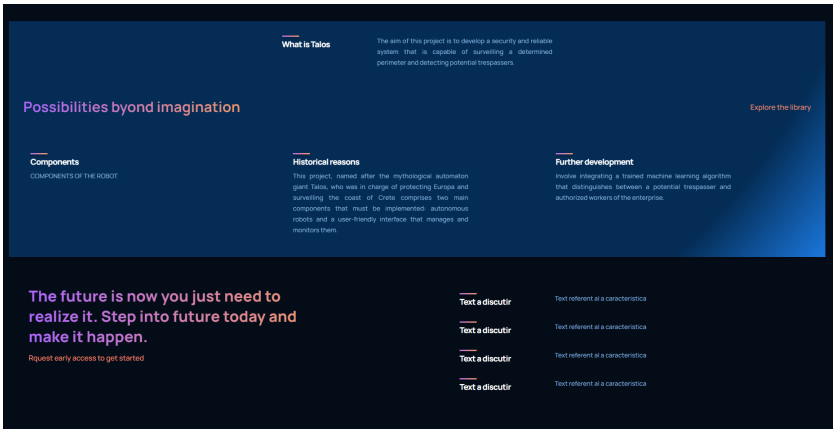

Figure 4: web what Talos and Features

The purpose of the "What is Talos" section was to introduce our project, explain our motivations for undertaking it, and provide the historical context behind the name "Talos." We aimed to give readers a comprehensive understanding of our vision and the significance of the project's name choice.

Additionally, we intended to create a subsection detailing each hardware component of the robot, complete with photos of the robot's internal structure. This subsection would have described each part, its function, and the rationale for its inclusion in our design. However, due to the complexity involved in assembling the robot, we were unable to complete the construction on time, and thus, this detailed breakdown was never realized.

In the "Features" section, our goal was to market the robot and make it appealing to users, effectively selling the product by highlighting its most attractive attributes.

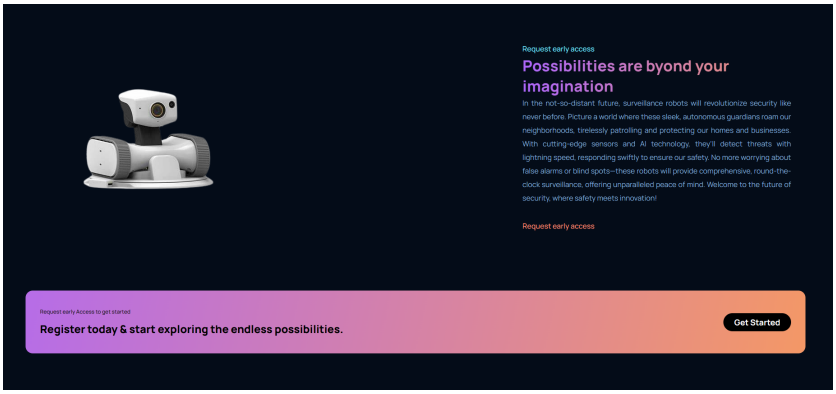

Figure 5: web Possibilities

In the "Possibilities" section, we aimed to discuss the potential of our project, outlining its medium to long-term prospects. We intended to highlight the future improvements and enhancements that could be incorporated to make the project even better.

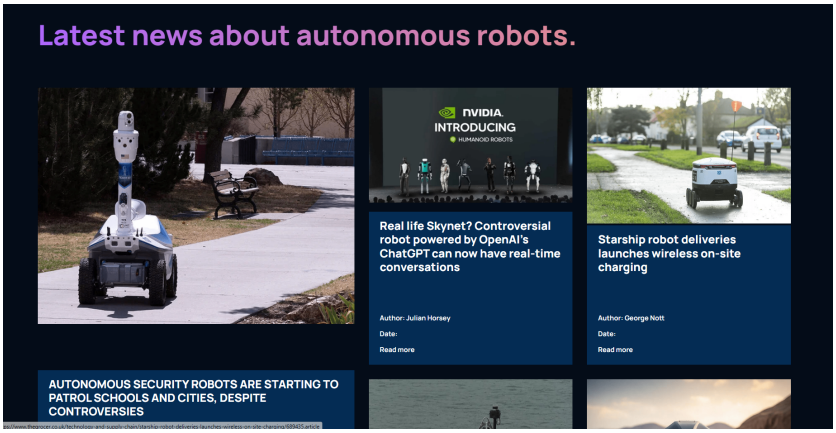

Figure 6: web News

The aim of incorporating current news about autonomous robots on our website is twofold. Firstly, it serves to keep the user informed about new discoveries, advancements, and research in the field, catering to those interested in technology and innovation. Additionally, it also serves to generate interest among users and present the robot in a more appealing manner.

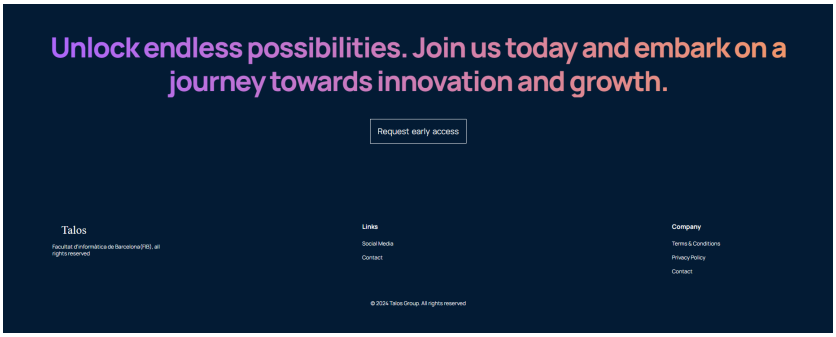

Figure 7: web Footer

We have finally incorporated a Footer on the website, with the intention to later expand it to include topics such as data privacy policy, contact information including email or phone number, location details, and social media links. Additionally, we plan to include attributions and copyright information. All of this was to be expanded upon if time allowed within the project, with links to privacy policies, copyright information, and contact details.[9]

6.3.2 Sign in/ Sign up

The objective behind creating the sign-in and sign-up forms was to have robust and secure forms. To achieve this, I used the background image of the logo, as aesthetically it was a pleasant image without being distracting, with soft colors. Additionally, making the form translucent ensured that it did not disrupt the image and provided a good user experience.

In terms of security, we implemented several measures. Firstly, passwords must be a minimum of eight characters, and users are required to confirm their password to minimize input errors. We also check if the user already exists; if so, the user is notified. Once registered, the server returns various user fields, but we only store the token, as it allows us to perform all subsequent requests to the endpoint and redirect the user to the dashboard.

However, we encountered several issues with this component, particularly regarding making requests to the endpoint and redirecting to the dashboard, which we will address later.

[9]All the problems and challenges of the creation of the web will be explained in the section, as its importance requires an explanation of its own.

Figure 8: Web Sign Up

6.3.3 Dashboard

The dashboard component may appear simpler than it actually is. Firstly, it's worth highlighting that this component is designed to enable toggling between light and dark modes using the lightbulb icon positioned at the top right corner of the screen. Implementing this functionality was one of the most challenging aspects of the website, as it required creating a component called theme.js where different colors and variables were defined for switching between light/dark mode. This component is part of the topbar of the dashboard, which not only includes this functionality but also features a settings and notifications button. The intention was to extend it to enable receiving real-time notifications and provide some user profile configuration options. Additionally, there's a dropdown menu in the user button that allows logging out the user, which essentially clears the user's token and returns them to the sign-in page. There were also profile buttons for changing the profile picture, although this additional functionality has not been implemented.

Regarding the sidebar, we've essentially created a section where users can upload their profile picture and display their username for personalization. In the sidebar, there's a "robot" section that redirects users to the page where robots are displayed. Other functionalities could not be implemented due to time constraints, but there is potential for future expansion to add more features and interaction to the website. Additionally, this menu expands or collapses based on the button at the top left corner.

Regarding the section for listing the user's robots, this was one of the most challenging components to implement due to its complexity. It's designed to display the robots in a table format, showcasing various robot components or fields that may be of interest to the user. These include the robot's name, creation and modification dates, as well as the robot ID and user ID. Additionally, we've added a button that appears grey if the robot is inactive and green if the robot is active, providing information to the user. Clicking on the robot's name redirects the user to the page displaying statistics for that specific robot, which we'll also discuss further later on. The intention behind this design is to allow users to view all robots upon entry, providing them with an overview, and allowing them to explore individual robots in detail if desired.

Lastly, it's worth mentioning that in the dashboard, we've implemented a layer of security called "withAuth" that wraps around the component. This layer ensures that access to the dashboard is only granted if the user has a valid token. Otherwise, access will be denied, and the user will be redirected to the sign-in page to log in. This measure prevents unauthorized access via URL manipulation in the browser.

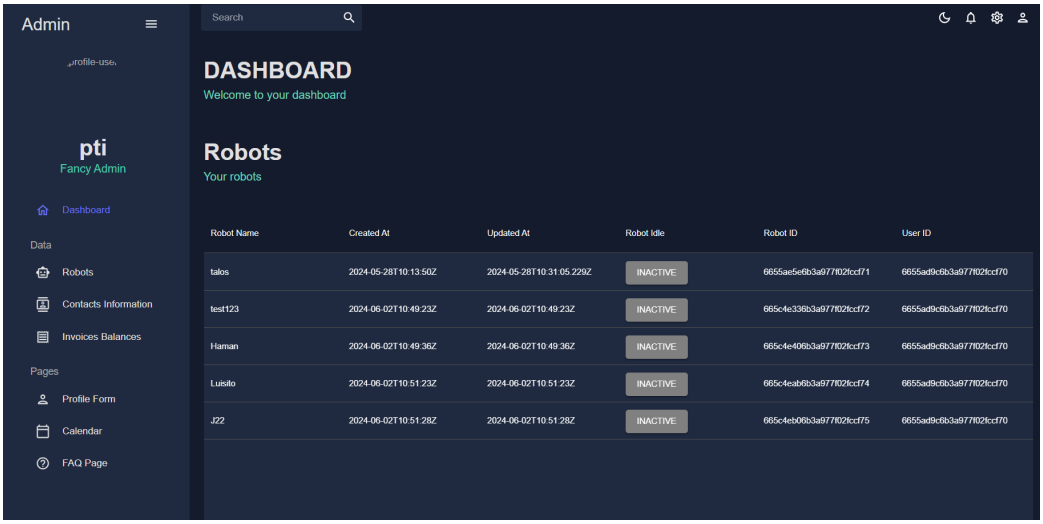

Figure 9: User Dashboard

6.3.4 Robot Statistics

In the statistics section of the robot, we see that it shares the same header as the dashboard. Additionally, the component is designed to support light/dark mode and includes a menu.

Among the new features, we first have the robot's name, which updates based on the specific robot. The URL also updates to provide a unique URL for each robot, allowing for a dedicated page for each robot's statistics. This routing is managed in App.js, where the robot's name is used to ensure that the URL is unique and varies according to each robot.

Regarding the user interface, it includes a video stream button that redirects to the live stream if the robot is active. It also displays graphs for CPU usage, temperature, and the robot's speed.

Lastly, we have integrated the Google Maps API, which was a challenging task. We encountered issues determining whether the robot had a route. If no route was found, the map centers on a default location (Barcelona); if a route is present, the map centers on the first point of that route.

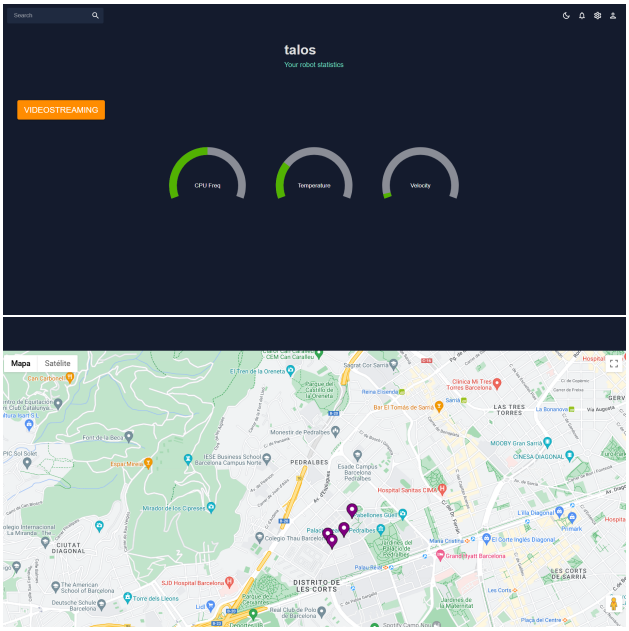

Figure 10: Robot Statistics

6.4 Challenges

6.4.1 Home

When discussing the challenges encountered in developing the Home component, the primary difficulty revolved around acquiring a comprehensive understanding of JavaScript, React, and CSS. This entailed learning how these technologies function, particularly within the React framework, and discerning the nuances involved in structuring the web application coherently to align with the project's objectives. Regarding CSS, the challenge primarily lay in comprehending the function of each attribute and crafting a web design that harmonized colors, spacing, and image scaling to ensure adaptability across various screen sizes and maintain user-friendliness.

Once these foundational aspects were mastered, we proceeded with the web development as planned, up until the implementation of news features. Our initial concept involved incorporating rotating news items to enhance interactivity and engagement for curious users. However, upon investigating the feasibility of this idea, we encountered significant obstacles, particularly the scarcity of readily available APIs specifically tailored to robot-related news, most of which were proprietary and incurred costs. Consequently, we opted for a static approach, curating compelling news stories sourced from reputable sources on the internet.

Lastly, it's worth highlighting the functionality associated with the sign-in and sign-up buttons, which presented novel challenges in terms of web routing implementation, as we had not previously tackled this aspect, leading to routing issues within the web application.

6.4.2 Sign in / Sign Up

In the sign-in and sign-up processes, we encountered few issues apart from the challenge mentioned before of integrating web routing comprehensively and making requests to the database. Specifically, when attempting to make requests, we encountered CORS policy errors. To address this, we opted to use XML requests instead of Axios or Fetch, as XML requests allowed us to log in and register successfully without modifying anything on the endpoint. Additionally, since it was our first time making requests to the database, we had to research how to do so correctly and securely.

6.4.3 Dashboard

In the dashboard, we encountered several challenges. The first one was setting up the routing correctly so that a personalized dashboard would appear for each user, and it would dynamically adjust based on the user's profile, preventing unauthorized access through URL modifications. However, the primary issue arose when listing the robots, as we utilized a component from the MUI Materials library (DataGrid). This component takes an input and displays it in a table format with predefined fields, listing the different robots associated with a user.

To achieve this, we initially made requests to the endpoint to retrieve all robots belonging to a specific user and their corresponding fields. Subsequently, we formatted the data to ensure compatibility with the component. Directly using the JSON returned by the component proved ineffective, as it was unable to recognize or display the fields properly. Consequently, we structured a vector to represent each robot, containing its various fields. We then selected the most pertinent fields to display to the user in the table and implemented button interactions to reflect the status of each robot.

It is noteworthy that we initially simulated robot data in a file to assess its behavior and determine which fields were essential for testing purposes. Once this preliminary step was completed, we proceeded with database queries to obtain the requisite information.

Finally, the most challenging issue we encountered was how to display each robot individually from the table showcasing all robots. Initially, we considered integrating this functionality directly into the dashboard, but found it unsatisfactory. Therefore, we opted to assign each robot its own URL and page to present the data in a user-friendly manner. To achieve this, we needed to include a link in each robot's name that would redirect users to a specific URL dedicated to that robot.

To address this challenge, we extensively researched web routing and how to create a dedicated page for each object. Eventually, we devised a viable solution whereby clicking on the robot's name would redirect users to its unique URL, triggering the loading of the RobotStatistics component. This component is responsible for making a request for the specific robot's data and rendering it accordingly.

6.4.4 Robot Statistics

Regarding robot statistics, we didn't encounter many issues. The simplest one was deciding on the type of chart we wanted to use to display the robot data. We also utilized a component from the MUI Materials library, which offered great versatility for presenting the data, and there was documentation available for customizing the component on the library's website.

However, the primary challenge we faced in this regard was integrating Google Maps into the website. None of us in the group were familiar with this process, so we had to extensively research and identify the library that offered the most customization options.

7 Mobile Application

7.1 Objectives

As we stated in the Proposal, the Mobile Phone Application shares identical characteristics with the Web Application. It offers capabilities such as live video streaming from the robot's camera and access to its sensors data including temperature and frequency readings. Moreover, there is one component that wasn't mentioned before. This enhancement involves the incorporation of a functionality allowing users to visualize the trajectory of the robot by integrating with the Google Maps API.

The main goal of the Mobile Application is to be able to visualize the robot's data and take advantage of the fact that we always carry our mobile phones with us, thus allowing us to access the robot's data from anywhere. While the website is primarily used to promote our product, the mobile application is intended for controlling the robot and instantly knowing if anything has happened to it.

Initially, our goal was to design an aesthetically pleasing and responsive interface for the robot. Considering these criteria, we immediately opted for Flutter due to its extensive libraries and documentation, which greatly aided our mobile development process. Additionally, the availability of pre-built visually appealing components eased our workload and aligned well with the objectives and the mobile application.

In summary, our choice of Flutter for mobile app development has not only allowed us to achieve a visually appealing and responsive interface but also effectively fulfilled the objectives of our mobile application.

7.2 Mobile Application design

The mobile application design is made to fulfill both functional and non-functional requirements, aiming to deliver a seamless and intuitive user experience. Key considerations include usability, real-time data visualization, aesthetic appeal, navigational simplicity, effective functionality, and user satisfaction. Each aspect is meticulously integrated into the design framework to optimize performance and enhance user engagement.

- **Usability:** The mobile application prioritized usability by incorporating intuitive touch-based navigation, clear interface structures, and user-friendly interactions, facilitating effortless engagement with the app.
- **Real-time Data Visualization:** Users should have the capability to monitor specific parameters of the robot, including CPU frequency, CPU temperature, speed, and camera feed. Additionally, the mobile application should be equipped to showcase the robot's trajectory on a map and display graphs to visualize the variables.
- **Aesthetic Appeal:** Ensuring visual appeal is a fundamental aspect of the mobile app design, with attention paid to visual elements, typography, color schemes, and imagery to craft a aesthetically pleasing interface made specifically for viewing on a mobile device.
- **Navigational Simplicity:** Efforts are made to ensure navigational simplicity by implementing intuitive gestures, clear menu structures, and logical flow, allowing users to effortlessly navigate the app and access its features.
- **Responsiveness:** The mobile app design prioritizes responsiveness, ensuring that the interface adapts to different screen sized and orientations. This approach guarantees an optimal user experience across various mobile devices, enhancing accessibility and usability.

7.2.1 Utilized technologies

As mentioned earlier, the technologies utilized include Flutter for application development, utilizing the Dart programming language. Additionally, we employed the Google Maps API for visualizing the robot's path. Lastly,

our own API was utilized for user management, robot management, video streaming, and retrieving robot information.

7.3 Mobile Development

Our mobile application's functionality is straightforward: it utilizes a stacked page approach, where each page overlaps the previous one. This allows us to control which page is displayed to the user at any given time. Additionally, when the back button is pressed while on the login page, the app will close. This design choice ensures the app remains responsive, as failing to exit when the back button is clicked would result in a poor user experience.

7.3.1 Login and Registration

We begin with the login and registration page, designed with simplicity and aesthetics in mind, serving as a mean to authenticate users and grant access to their individual accounts. To begin with, we will talk about how we manage the users sessions and how can we know a user is logged in or logged out.

To determine whether a user is logged in, we use a polling method that continuously checks if the user has a stored token. If the token is present, we display the dashboard page; if not, we show the login/register page. This is implemented using **StreamBuilder**, a useful class in Flutter designed to respond to events in real-time.

Figure 11: Dashboard or Login/Register Code

As shown in the picture above, we have a **StreamBuilder** that continuously checks for the token saved in the *prefs* variable. If the token is found, it redirects you to the HomePage, which is the dashboard. If the token is not found, it redirects you to the LogInOrRegisterPage, which handles both the login and registration processes.

To clarify, the LogInOrRegisterPage checks if the button has been pressed and changes a boolean variable accordingly. Depending on the state of this variable, it returns a different page. This behavior is illustrated in the picture below.

```dart
class LoginOrRegisterPage extends StatefulWidget {
  const LoginOrRegisterPage({super.key});


  @override
  State<LoginOrRegisterPage> createState() => _LoginOrRegisterPageState();
}

class _LoginOrRegisterPageState extends State<LoginOrRegisterPage> {
  bool showLoginPage = true;

  void togglePages(){
    setState((){
      showLoginPage = !showLoginPage;
    });
  }

  @override
  Widget build(BuildContext context) {
    if(showLoginPage){
      return LoginPage(
        onTap: togglePages,
      );
    } else {
      return RegisterPage(
        onTap: togglePages,
      );
    }
  }

}
```

Figure 12: Login or Register Code

In addition, our application includes a dashboard where most of the functionality resides. On one hand, robots are displayed in a grid pattern that adjusts according to the number of robots in your profile. You can scroll to view robots that are positioned lower in the grid. On the other hand, we have a more complex requirement: parsing information received from the API and transforming it into models and objects. Essentially, we created our own package to convert JSON strings into objects. You can find all the

models under the **packages** section in the GitHub project. In addition to parsing JSON into objects, we also need to create a *BlocBuilder* object that dynamically builds the robots when you refresh the page. It's crucial to handle cases where information about a robot fails to load, as displaying null values in the UI is not acceptable. Thus, we ensure that only complete and valid robot information is displayed on the page. This two aspects I have just mentioned, where the hardest by far of all the mobile application.

```
class ApiRobotsRepo implements RobotsRepo {


  Future<List<RobotEntity>> getUserRobots() async {
    final prefs = await SharedPreferences.getInstance();
    List<RobotEntity> robots = List.empty();
    var headers  = {'Content-Type': 'application/json', 'token': prefs.get('token').toString()};
    try {
      var url = Uri.parse("http://nattech.fib.upc.edu:40342/robot/listar");
      http.Response response =
        await http.get(url, headers: headers);

      if(response.contentLength == 0 || response.body == "null"){
        return List.empty();
      } else {
        final List parsedList = jsonDecode(response.body);
        log(parsedList.toString());
        List<RobotEntity> map = (parsedList).map((i) => RobotEntity.fromJson(i)).toList();
        return map;
      }
    } catch (e){
      log(e.toString());
      rethrow;
    }
  }

  @override
  Future<List<RobotEntity>> getRobots() async {
    try{
      return getUserRobots();
    } catch (e) {
      log(e.toString());
      rethrow;
    }
  }
}
```

Figure 13: Transformation from JSON to *RobotEntity*

Finally, we reach the last components of the application: the HomeDetails page, the HomeStats page, the MapPage, and the VideoStreaming page. Let's delve into each of them. Honestly, once you have the robot data, your

main focus is on presenting the information in an appealing manner. The difficult part is already behind us.

7.3.2 Home Details page

This page is responsible for displaying the robot name, the time when the robot is created and the last time the robot has been updated.

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
      backgroundColor: Colors.lightBlue[50],
      body: Padding(
        padding: const EdgeInsets.all(30),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            const Text(
              "Statistics",
              style: TextStyle(
                fontSize: 36,
                fontWeight: FontWeight.bold,
              ), // TextStyle
            ), // Text
            const SizedBox(height: 40),
            EditItem(
              title: "Name",
              widget: TextField(readOnly: true, controller: _controllerName),
            ), // EditItem
            const SizedBox(height: 40),
            EditItem(
              title: "CreatedAt",
              widget: TextField(readOnly: true, controller: _controllerCreated),
            ), // EditItem
            const SizedBox(height: 40),
            EditItem(
              title: "UpdatedAt",
              widget: TextField(readOnly: true, controller: _controllerUpdated),
            ), // EditItem
          ],
        ), // Column
      )); // Padding // Scaffold
}
}
```

Figure 14: Code from Home Details Page

7.3.3 Home Statistics Page

This page is tasked with presenting the statistics of a robot in an aesthetically pleasing manner. Primarily, it showcases the robot's CPU temperature and CPU frequency. Utilizing a component, it animates these values and visually represents the information in an engaging manner.

```
@override
Widget build(BuildContext context){
  return Scaffold(
    body: Center(
      child: isLoading ? Column(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: <Widget>[
          CustomPaint(
          foregroundPainter: CircleProgress(temperatureAnimation.value, true),
          child: SizedBox(
            width: 200,
            height: 200,
            child: Center(
              child: Column(
                mainAxisAlignment: MainAxisAlignment.center,
                children: <Widget>[
                  const Text('Temperature'),
                  Text(
                    '${temperatureAnimation.value.toInt()}',
                    style: const TextStyle(fontSize: 50, fontWeight: FontWeight.bold),
                  ), // Text
                  const Text(
                    '°C',
                    style: TextStyle(
                      fontSize: 20,
                      fontWeight: FontWeight.bold
                    ) // TextStyle
                  ) // Text
                ] // <Widget>[]
              ) // Column
            ) // Center
          ), // SizedBox
        ), // CustomPaint
```

Figure 15: Code that represents the temperature of the robot's CPU

```
class _MapPageState extends State<MapPage> {

  final Set<Marker> markers = new Set();

  static const LatLng position = const LatLng(41.389486, 2.113405);
  bool isNull = false;

  @override
  void initState(){
    super.initState();
    int? length = widget.robot.ruta?.length;
    if(widget.robot.ruta == null){
      isNull = true;
    } else {
      for(int i = 0; i < length!; ++i){
        double latitud = widget.robot.ruta?[i].latitud ?? 0.0;
        double longitud = widget.robot.ruta?[i].longitud ?? 0.0;
        log(longitud);
        markers.add(Marker(
          markerId: MarkerId('test${i}'),
          position: LatLng(latitud, longitud),
          icon: BitmapDescriptor.defaultMarker
        )); // Marker
      }
    }
  }
```

Figure 16: Code that visualizes the map and the waypoints of the robot

7.3.4 Map Page

This was the easiest of all the pages as we have the markers inside the robot object. Then, we just need to display that information on the map. This can be seen on the picture below:

7.3.5 Video streaming Page

Video streaming is a crucial feature of our surveillance robot project. It enables real-time video streaming from the robot's cameras. This way we can monitor the robot's surroundings in real-time. To implement video stream-

ing, we used the **opencv** library, which is a popular open-source computer vision library. It provides a wide range of functionalities for image and video processing, including video streaming. For this specific implementation, we used two scripts which will be included in the GitHub project. One script handles the client connection (i.e., the robot), while the other manages the server, which listens for incoming connections.

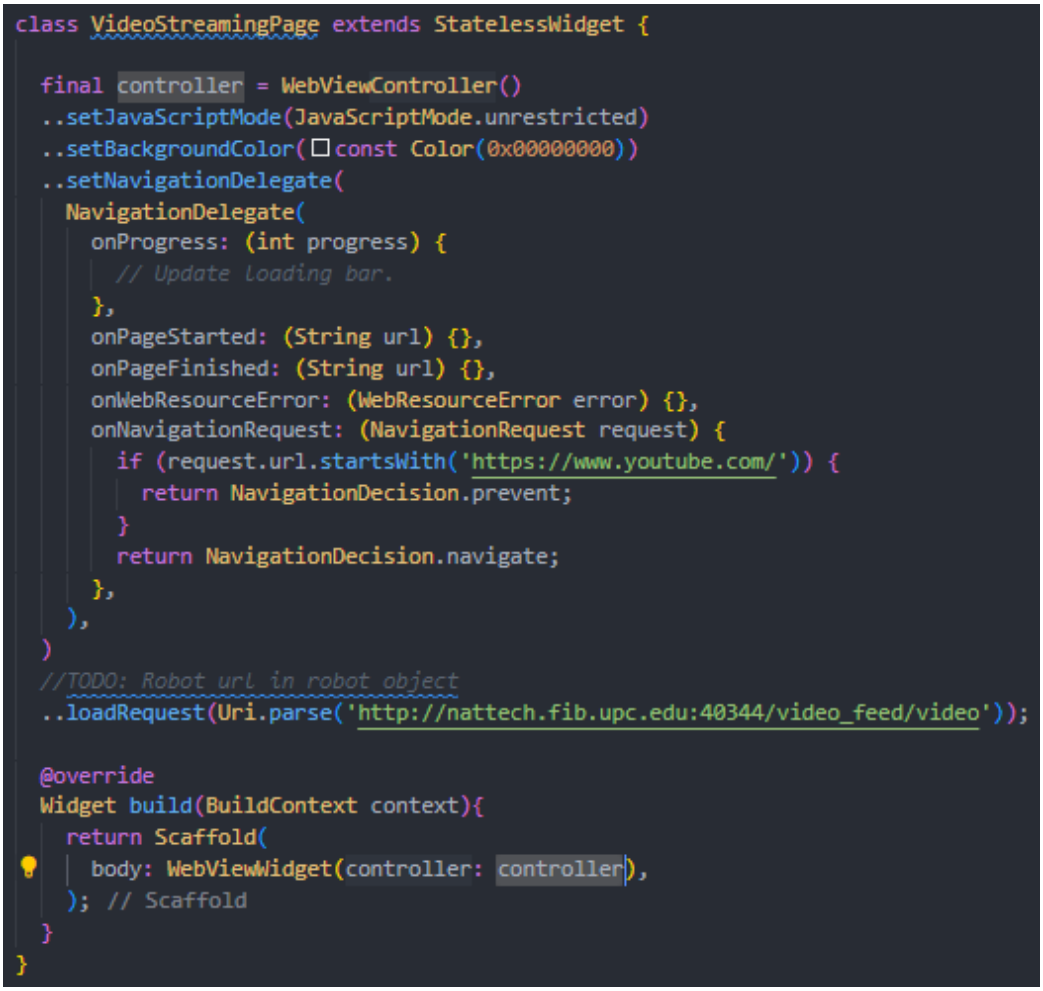

Figure 17: Code used for the Video Streaming Page

8 API

8.1 Objectives

The REST API we have developed is a set of endpoints that allow different components of the system to communicate with each other such as our web/app with our database. These endpoints are used to retrieve and update data in the system. The API is designed to be flexible and can be extended to support new features and functionalities in the future. We have also implemented authentication and authorization mechanisms to ensure that only authorized users can access certain endpoints. We mainly can differentiate between robots and users endpoints. We have used the the programming language Golang and we have connected the API to a MongoDB database.

8.2 API design and development

We mainly can differentiate between robots and users endpoints.

8.2.1 Users

The user model is a collection of fields that represent the user's information. These fields include the user's username, email, password, and ID that is unique to each user. The user model also includes a field for the time it was created and the time it was last updated. The user model also contains a field for a token that is used to authenticate the user. The token is generated when the user is registered and is valid for a certain period of time. The token is updated every time the user logs in.

The following endpoints are used to manage users, allowing them to register, login, and consult their information. The endpoints are divided into two categories: public and private. The public ones are accessible to anyone and the private one is only accesible trhough a token.

- **/users/signup** This endpoint is used to register a new user and is a POST request where you have to provide the user's information such as username, email, and password in a JSON format. The endpoint returns a JSON response with the user's token if the registration is successful. The user is created and their information is stored in the database and can be logged in. The endpoint uses an internal function

called SignUp to handle the registration process and inside the function, it checks if the username and email already exist in the database. If they do, it returns an error message. The function also hashes the password with another internal function called HashPassword and stores it in the database. This endpoint is public because you do not need to proivide a token in the header of the request.

- **/users/login:** This endpoint is used to log in as a user and is a POST request where you have to provide the user's username and password in a JSON format. The endpoint returns a JSON response with all user's information if the login is successful. The endpoint uses an internal function called Login to handle the login process and inside the function, it checks if the username and password match the ones stored in the database through an internal function called VerifyPassword. If they do, it returns the user's information. If the login fails, it returns an error message.This endpoint is public because you do not need to proivide a token in the header of the request.

- **/users/consulta** This endpoint is used to obtain user information and is a GET request. You need to provide the user's token in the header of the request to authenticate the user. The endpoint uses an internal function called Consult to obtain the user's information from the database based on the token provided. The function returns a JSON response with the user's information. This endpoint is private because you need to provide a token in the header of the request to access it.

8.2.2 Robots

The robot model is a collection of fields that represent the robot's information. These fields include the robot's name, id, idle, state and the id of the user owning the robot. The robot model also includes a field for the time it was created and the time it was last updated. The robot model contains a struct called Macros that contains the information of some aspects of the robot such as the frequency of the cpu, the temperature and the velocity of the robot. Finally this model conatains the route that the robot is following and that route is represented by an array of points which point is composed by two values latitude and longitude.

The following endpoints are used to manage robots, allowing the user to create and delete robots among other functions. All the endpoints are private and the system needs to provide the specific token of their owner in the header of the request to access them.

- **/robot/alta** This endpoint is used to create a robot and is a POST request where you have to provide the robot's name in a JSON format. The endpoint returns a JSON response with a message indicating if the creation was successful or not. The endpoint uses an internal function called CreateRobotForUser to handle the creation process and inside the function, it checks if the robotname already exists in the full set of robots of the user in the database. All the fields of the robot are filled with default values.

- **/robot/baja/:robotname** This endpoint is used to delete a robot and is a POST request where you have to specify the name of the robot in the URL. We match the robot with the owner thanks to the token provided in the header of the request. The endpoint returns a JSON response with a message indicating if the deletion was successful or not. The endpoint uses an internal function called DeleteRobotForUser to handle the deletion process and inside the function, it checks if the robotname exists in the full set of robots of the user in the database.

- **/robot/listar** This endpoint is used to obtain the information of all the robots of the user and is a GET request. The endpoint returns a JSON response with an array of robots with their information. The endpoint uses an internal function called ListUserRobots to handle the listing process and inside the function, it checks if the user has any robots in the database. In the case of not having any robots, the endpoint returns an empty array.

- **/robot/consulta/:robotname** This endpoint works as the /robot/listar endpoint but instead of returning the information of all the robots of the user, it returns the information of an specific robot that matches the name provided in the URL. The endpoint uses an internal function called ConsultUserRobot to handle the obtaining process and inside the function, it checks if the robotname exists in the full set of robots of the user in the database. If the robotname does not exist, the endpoint returns an error message.

- **/robot/refresh-macros/:robotname** This endpoint is used to update the macros of a robot and is a POST request where you have to provide the robot's name in the URL and the new macros in a JSON format. The endpoint returns a JSON response with all the information of the robot and the new macros updated. The endpoint uses an internal function called RefreshMacrosRobot to handle the updating process and inside the function, it checks if the robotname exists in the full set of robots of the user in the database.

- **robot/refresh-route/:robotname** This endpoint is used to update the route of a robot and is a POST request where you have to provide the robot's name in the URL and the new route in a JSON format. The endpoint returns a JSON response with a message indicating if the updating was successful or not. The endpoint uses an internal function called RefreshRoute to handle the updating process and inside the function, it checks if the robotname exists in the full set of robots of the user in the database.

- **robot/desactivar/:robotname** This endpoint is used to desactivate a robot and is a POST request where you have to provide the robot's name in the URL. The endpoint returns a JSON response with a message indicating if the desactivation was successful or not. The endpoint uses an internal function called DesactivateRobot to handle the desactivation process and inside the function, it checks if the robotname exists in the full set of robots of the user in the database.

8.2.3 Authentication and Authorization

The system uses a token-based authentication and authorization mechanism. We use a JWT (JSON Web Token) to generate and verify tokens. A JWT is a JSON-based token that is used to securely transmit information between parties. It is a compact and self-contained format that can be easily transmitted over various communication channels. The JWT is signed with a secret key, which is used to verify its authenticity and integrity. The JWT contains three parts: the header, the payload, and the signature. The header contains information about the token, such as the type of token and the algorithm used to sign it. The payload contains the user's information, in our case the user's ID, username, email and time of expiration. The signature is a hash of the header and payload, which is used to verify the authenticity of the token.

The system uses the JWT library to generate and verify tokens. When a user logs in or register for first time, the system generates a token and stores it in the database. The token is then included in the header of the request when the user makes a request to the system. The system verifies the token thanks to the middleware by checking the signature and the payload. If the token is valid, the system allows the user to access the requested resource. If the token is invalid, the system returns an error message.

8.2.4 Utilized technologies

The technologies used for the API implementation were the Golang programming language for development and connecting to the MongoDB database. Due to complications in installing the latest versions of this database, we had to downgrade to version 4.4.8. Other support tools we used to facilitate testing and work more comfortably included Postman, which helped us test the API requests, and MongoDB Compass, which allowed us to visualize the content (documents) of the database

9 Robot website

9.1 Objectives

We designed a website to configure the robot, essentially creating a link between the user and a robot instance. The website consisted of three fields: one for the username, another for the password, and the last one for the robot's name. Once the three fields were filled out, the user and password were validated through the login. If successful, a token was obtained, and the creation of the robot instance in the database proceeded. After this, a file was created with the robot's configuration, consisting of the owner's name (the username) and the robot's name. Once this was done, the route assignment for the robot was carried out using the Google API, which allowed us to visualize a map and select points, from which we obtained the latitude and longitude of each point. We then made a request to the API to carry out the route assignment.

9.2 Web robot design and development

We can differentiate the robot's website into two parts: the first part is where the robot is linked with the user, and the second part is where we select the route our robot will follow.

9.2.1 Linking robot and user

As I mentioned earlier, this part is based on linking the robot with the user and creating a configuration file for the robot. With the three fields I mentioned: Username, Password, Robotname. These are the fields we will work with because once they are entered, we will use the username and password to log in through a request to our API. If the request is successful, we will get the token to authenticate ourselves in future requests. If the login is not successful, nothing will happen. Once logged in and with the token in our possession, we will proceed to create the robot by making a creation request with the specified token and robotname. If this request is successful, the configuration file with the username, password, and robotname will be created.

9.2.2 Route assignment

Once the configuration file is ready, we can use the Google Maps integration on the webpage to instruct the robot to follow a specific route. Each click on the map places a marker, representing a waypoint for the robot. When you click on the map, the latitude and longitude are captured and stored n a variable. After selecting your desired route, clicking a button ("Store All Markers") will trigger an API call to save the information, ensuring it is correctly displayed on both the webpage and the mobile application.

9.2.3 Utilized technologies

The technologies used for the development of this website, which is integrated into the robot, are Flask, which uses the Python programming language, along with the Google Maps API for route assignment, and our own API for making requests that allow us to query and validate data from our database, as well as its storage, such as creating a new robot instance with its subsequent route assignment.

References

[1] *PTI Github*. URL: https://github.com/guillermovidalsule/PTI.

[2] *Lichee Pi 4A*. URL: https://sipeed.com/licheepi4a.

[3] *Motor controller*. URL: https://www.amazon.es/dp/B07XJSMLGQ?psc=1&ref=ppx_yo2ov_dt_b_product_details.

[4] *755 DC motor*. URL: https://www.amazon.es/dp/B075D85KVV?psc=1&ref=ppx_yo2ov_dt_b_product_details.

[5] *GPS module*. URL: https://www.amazon.es/gp/product/B08CZSL193/ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&psc=1.

[6] *15.2V Battery*. URL: https://www.amazon.es/dp/B09YVHF2PH?psc=1&ref=ppx_yo2ov_dt_b_product_details.

[7] *XT90 to cable adapter*. URL: https://www.amazon.es/dp/B0CDPZ4XML?psc=1&ref=ppx_yo2ov_dt_b_product_details.

[8] *c-AWG-cable*. URL: https://www.amazon.es/dp/B0BCG93BM9?psc=1&ref=ppx_yo2ov_dt_b_product_details.

[9] *USB camera module*. URL: https://www.amazon.es/dp/B07QGZCF8N?psc=1&ref=ppx_yo2ov_dt_b_product_details.

A Talos 3D printed parts

Talos was built using multiple 3D printed parts. Although the models of these parts can be found in the github, we believed convenient to add a brief description of them and their purpose.

1. **Wheel base** `(4)` - Central part of the wheel that holds most of it together.
2. **Left wheel roller holder** `(4)` - Needed in order to hold the rollers in place.
3. **Right wheel roller holder** `(4)` - Idem, but for right-oriented wheels.
4. **Rollers** `(40)` - Cylindrical components that are placed along the longest diameter of the wheel.
5. **DC motor gripper** `(4)` - Part that joins the DC motor and the wheel together.
6. **DC Motor holder** `(4)` - Part that holds the DC motor to the bottom chassis.
7. **USB camera holder** `(1)` - Part that holds the USB camera module to the top chassis.
8. **Bottom chassis** `(1)` - Important part that holds most of the weight of the robot.
9. **Top chassis** `(1)` - Responsible of protecting the electronics of the robot. It is also the part where most of these electronic components are screwed to.