Smart Portable Charger

Claudia Ortiz

Summary

In today's technological world, portable batteries have become indispensable accessories for powering our devices on the go. However, these gadgets have seen minimal innovation beyond increasing capacity over time. Recognizing this, we started to explore the potential of creating a smart portable battery or charging station that offers advanced control features for multiple devices simultaneously. Our research led us to believe that something like this could significantly enhance user experience and address the evolving needs of consumers in the digital age.

The motivation behind our investigation stemmed from the desire to introduce innovation and convenience into the realm of portable power solutions. We were driven by the opportunity to create a product that not only meets the basic need for power but also adds value through intelligent features and enhanced user control. Our exploration was fuelled by the belief that this project could make a positive impact on the lives of our consumers by offering them greater flexibility and control over their device charging experience.

Our journey began with extensive research into existing portable battery solutions and emerging trends in the consumer electronics market. We delved into literature reviews, market analyses, and consumer feedback to gain insights into the current landscape and identify areas for improvement. Additionally, we consulted with experts in the field and conducted surveys to gather firsthand perspectives on user preferences and pain points.

In conclusion, our investigation has highlighted the potential of developing a smart portable battery / charging station. By addressing the limitations of existing solutions and introducing intelligent features, we aimed to redefine the portable power experience for consumers. Through this project, we hope to not only transform our ideas into products but also gain valuable insights and skills that will be invaluable in our future endeavours.

Contents

List of Tables

List of Figures

Introduction

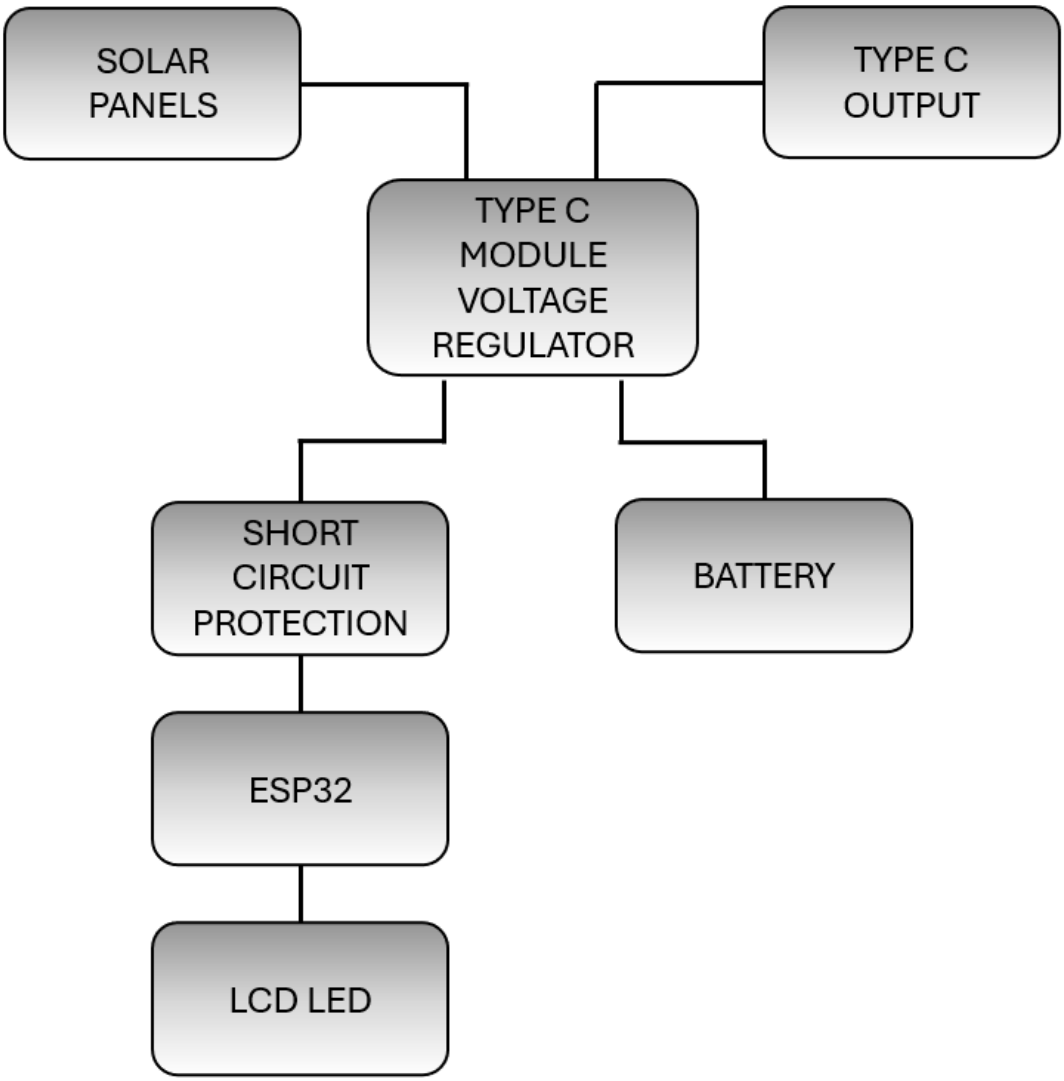
In the process of selecting a project for our team, we carefully considered various ideas, ultimately narrowing down our choices to two finalists: a mini arcade machine and a smart portable charger. The decision-making process involved evaluating the potential interest and novelty of each idea, especially considering our team's lack of prior experience with hardware projects. After presenting the finalists to my teammates, we engaged in discussions and deliberations. Ultimately, we opted for the smart portable charger because we believed it could offer practical utility in our daily lives. This choice aligns with the relevance of the topic – a portable charger addresses a common need for individuals who rely on electronic devices on a daily basis.

The purpose of our project is to create a smart portable charger that not only charges devices efficiently but also incorporates features that enhance user experience. This investigation is valuable as it allows us to explore innovative ways to integrate technology into a commonly used device, making it more versatile and user-friendly. The achievement of this goal would result in a product that not only meets the basic need for device charging but also provides additional functionalities for user convenience.

To gather information for our project, I initiated a thorough research process. This involved delving into topics such as connectivity options, solar charging technology, battery types, types of controllers, and display mechanisms for conveying information. We consulted a variety of sources, including online articles and technical documentation. Additionally, we utilized AI to gather more information about certain topics. Throughout the evolution of the project, I continuously researched information about various components such as plaques, connections, different batteries, OTA connection methods, and how to retrieve data from a device connected to our portable charger. At every phase of our project, I diligently conducted research to stay well-informed and make informed decisions about what was best for our project. This comprehensive approach to gathering information ensured that we had a well-rounded understanding of the technical aspects and possibilities related to our smart portable charger project.

Briefly stated, the selection of the smart portable charger as our project stemmed from careful consideration of its relevance and potential impact on users' lives. The purpose of our investigation is to create a technologically advanced and user-friendly device, addressing not only the basic need for charging but also providing additional features for enhanced user experience. The information gathering process involved thorough research from diverse sources, ensuring a robust foundation for the development of our innovative smart portable charger.

Hardware Design and Components



1 - Block Diagram of the circuit

Controller

For the controller we have selected the **ESP32** because it has an integrated WiFi connection which would make it easier for us to send information to the website. In addition, this controller consumes very little power, so there would be more battery left for charging devices.

Battery

For the battery we thought of using a Lithium Ion 18650 Battery as it is compact, and provides a good capacity as well as having a low price, which for an initial prototype is perfect. Also, this type of battery is rechargeable, which is needed for when it charges itself thanks to the solar panels and the manually charging through a cable.

When selecting the battery, we encountered a problem: the battery provides an electrical voltage of 3.7V while the ESP32 works with 3.3V, to solve this, we thought of using a voltage regulator such as the MCP1700-3302E. But, with the connections we have, we won't need to use that, because the Voltage Regulator Type C Module we have already regulates the voltage.

Button

We will have one **button** to show on the screen, the battery that the power bank has. We have programmed it so it's only showing the information every 5 seconds, so it just shows the amount of battery left when the user needs it.

Type C Module Voltage Regulator

This module will regulate the voltage received from the solar panels to the battery (and vice versa) to the ESP32. By having this, there's no need for us to use the microchip voltage regulator MCP1700-3302E, which I have researched before knowing that our module would do the same job.

Pins

We will have a **USB C pin** for charging the power bank in case the solar panel is not receiving sunlight; and a **Type-A plug pin**, so we can charge by cable the devices.

Screen

We will use a screen that will display the battery that's left if the battery if getting charged through the solar panel or not and the wi-fi connexion.

To display this data, we looked for a 16X1 LCD screen as it is sufficient to display all the data.

When selecting this component, we have found another drawback: it uses a voltage of 5V and the ESP32 uses 3.3V, to solve this we have thought of using a boost type voltage regulator such as the RP402N501E-TR-FE, but, as we can't get one, we used Ohm resistors (with a total resistance value of 127k) to connect both components.

Solar Pannel

The ESP32 works at a voltage range of 2.2 to 3.6V so we will need a solar panel that works at a maximum of 3.3V, because the I/O pins are not 5V-tolerant.

PRICE (each)	WEB LINK OF THE SOLAR PANEL
0.83 €	Mini Solar Panel Polycrystalline Silicon 1W 3V 100MA Outdoor
1.71 €	3V 120mA Mini Solar Panel Polycrystalline Silicon DIY
0.74 €	3V 0.24W 67.5x34.5mm Polycrystalline Silicon Portable Solar Cell Panel Mini Solar Cell Plate with Cable

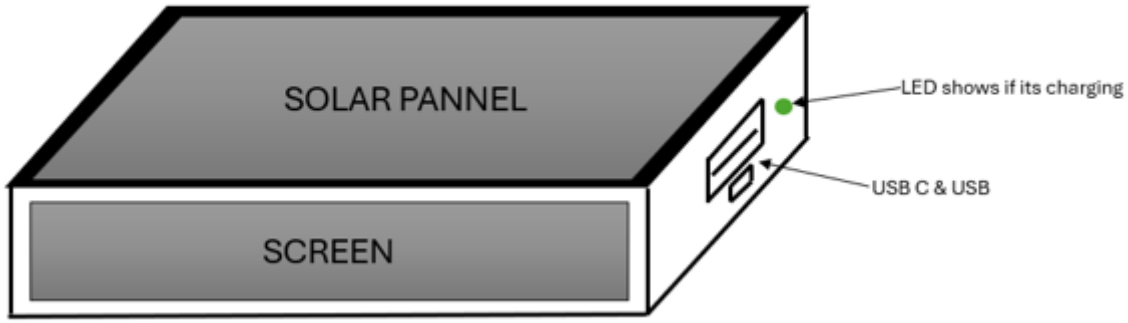
Table 1 – Types of Solar Panels

After looking at these diverse possibilities, the best option is the first option, because it's an outdoor solar panel, the material is polycrystalline silicon, it functions at a power of 3V, and its current is 100MA. When we ordered this component, another extra solar panel came with it, and we used both to have a greater input to the voltage regulator module, so it can later charge the battery.

Case

I designed the case for a 3D printed model, using Fusion360 and we will use PLA filaments for the 3D print. I designed it in a way that can have the space for the 2 output pins (Type C and Type B), the LCD – LED screen, the solar panels; and in the inside, save all of the others components such as the ESP32, the battery, the cable connections, etc.

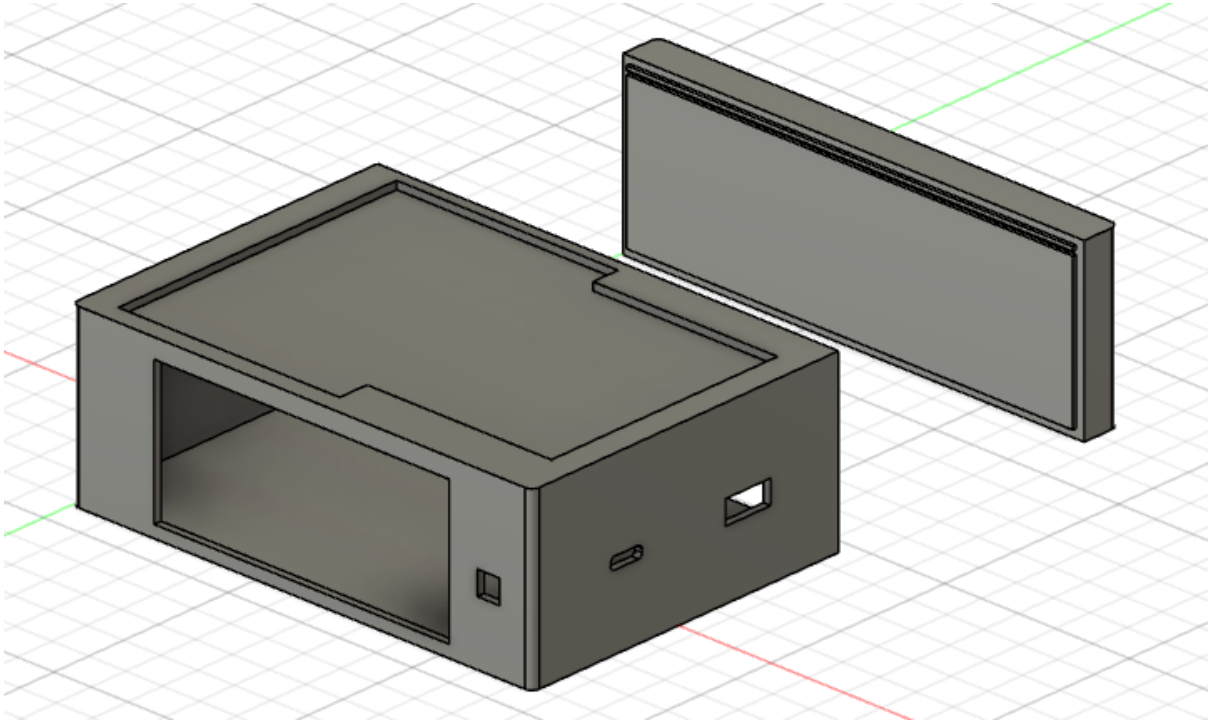
During the first stages of the project, I draw a visual sketch of how could it be designed:



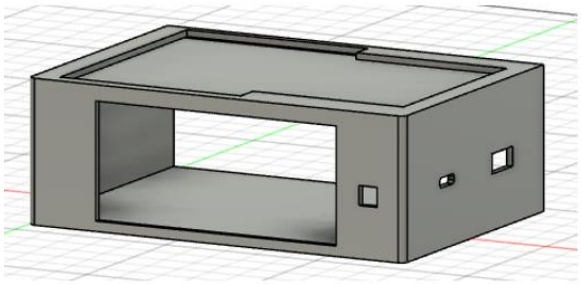
2 – Initial Sketch of the Case

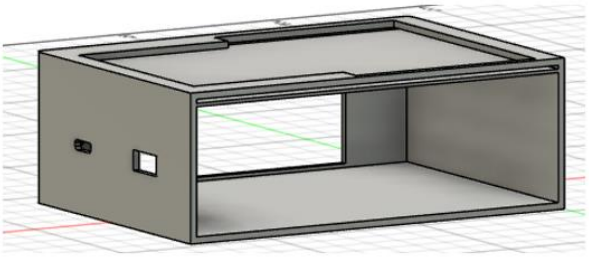
From this sketch, we won't have the light, because the battery will charge only from the solar panels when the battery is running out, so we thought it was no longer necessary. What we added was a button, that when you click on it, it shows, for 5 seconds the battery that the Portable Charger has.

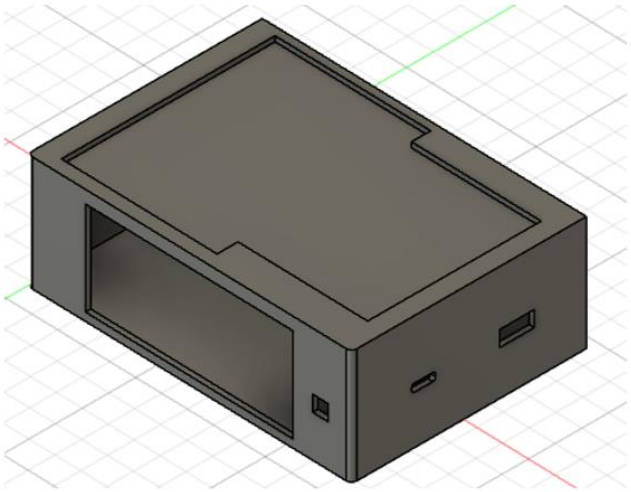
After soldering the components, I measured the spaces we need for the case, and started designing it. Here I show the current design:

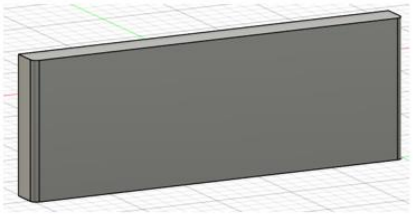


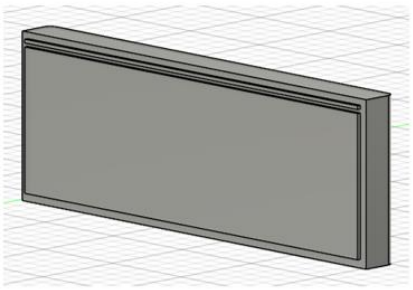
3 – Case Design using Fusion360

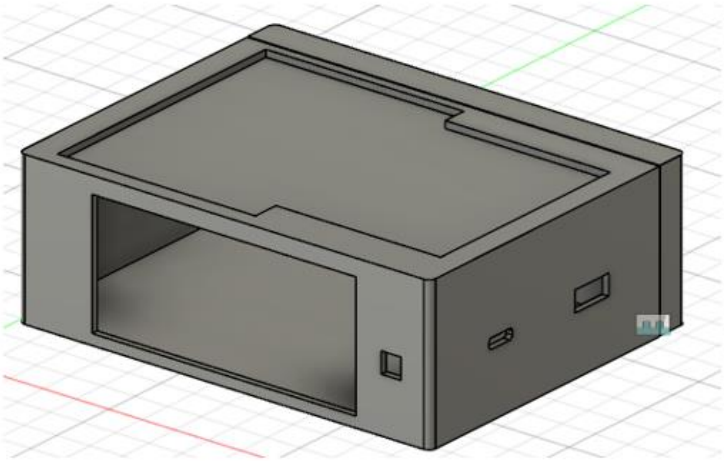


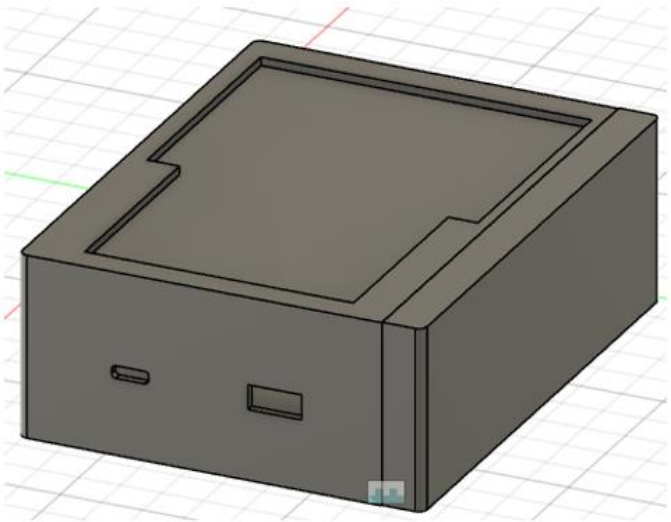












Software Design

Connectivity

For the connectivity I researched for the advantages and disadvantages of the different types of connectivity's we could use in this project.

TYPE	ADVANTAGES	DISADVANTAGES
Ethernet	- Stable and reliable connection. - High data transfer rates.	- Wired connection, which may limit portability. - Requires physical connection, making it less convenient for mobile devices
WiFi	- High data transfer rates, suitable for transmitting real-time information. - Widespread availability and compatibility.	- Higher power consumption compared to some other wireless technologies. - Limited range and potential for interference in crowded environments.
Bluetooth	- Low power consumption in Bluetooth Low Energy (BLE) versions. - Commonly available in smartphones and other devices.	- Limited range compared to other wireless technologies. - Interference from other Bluetooth devices in crowded areas.
ZigBee	- Low power consumption and suitable for low-data-rate applications. - Mesh networking capabilities, enabling extended coverage.	- Limited data transfer rates compared to WiFi. - Not as widely adopted as some other wireless technologies.

LoRa	- Long-range communication capabilities, making it suitable for remote monitoring. - Low power consumption for battery-operated devices.	- Lower data transfer rates compared to WiFi. - Limited support for real-time applications due to latency.

Table 2 – Connectivity

After my thorough research, which was summarised in the table above, we decided that integrating WiFi connectivity in our project was essential due to the necessity of a wireless connectivity, the need for a high-spread technology and the requirement of connecting to both web and app platforms.

After deciding that we were going to use the WiFi connectivity, we then proceeded to decide how were we going to exactly connect the ESP32 data to the Web / App. While considering various options such as Node-Red and Firebase, we ultimately opted for Firebase. Its Realtime Database, compatibility to connect with Android Studio for an app development, and to link a web page we created, aligned perfectly with our project requirements. Further on, I will explain exactly how I created the project in Firebase, the Realtime Database, and how we connected the web and mobile app we created.

Dashboard

Web Development

Through the website the user will be able to monitor the battery level. There are many technologies we could have used such as WordPress or Wix, which would allow us to create an attractive and eye-catching site, however due to the lack of options for the implementation of the custom functions our team wants to add to the project, we decided to create both the website from scratch. For this purpose, we had two alternatives, using PHP or HTML and JavaScript. We opted for the second option due to our previous knowledge and how easy is to learn new features in both html and JavaScript. Using HTML, JavaScript and overall CSS

allows us to design the web with the approach of mobile first, which means designing the site with the user experience from mobile in mind and making it accessible from smartphone devices.

After choosing a technology to work with, for the web design we studied different colour pallets and made the decision of using blue as our main colour for the background. After that, we chose two pallet colours, one for the dark themes, and the other for the clear themes. For both colour schemes we decided to use high contrast colours for people with sight problems, especially for the dark mode which will have the combination of the dark blue (close to black), and yellow. This combination of colours is the one that generates more contrast to the human brain. For the clear theme, on the other hand, we choose a scheme based on the primary colours in the subtractive model (YMCK) that is to say, yellow, magenta, cyan and ‘Key’, which is black.

Also, for the icons we thought the best decision would be to have rounded forms instead of sharps ones or pointy lines, which could be perceived as more aggressive for the common user, rather than the rounded shapes, which create a more inviting look and feel. We saved the sharper forms for elements that need to grab the user's attention.

Lastly, for the web, hosting is a critical aspect of website functionality. Thanks to the ESP32, we are able to integrate the server into the device itself, eliminating the need for external hosting services, at least in the initial prototype stage.

4 - Color Palette

Mobile App

For the mobile app, at first we decided to use the same design philosophy, using the same colour scheme and icons to have a visual coherence with the website and don't duplicate work, but due to lack of communication, they ended up being different.

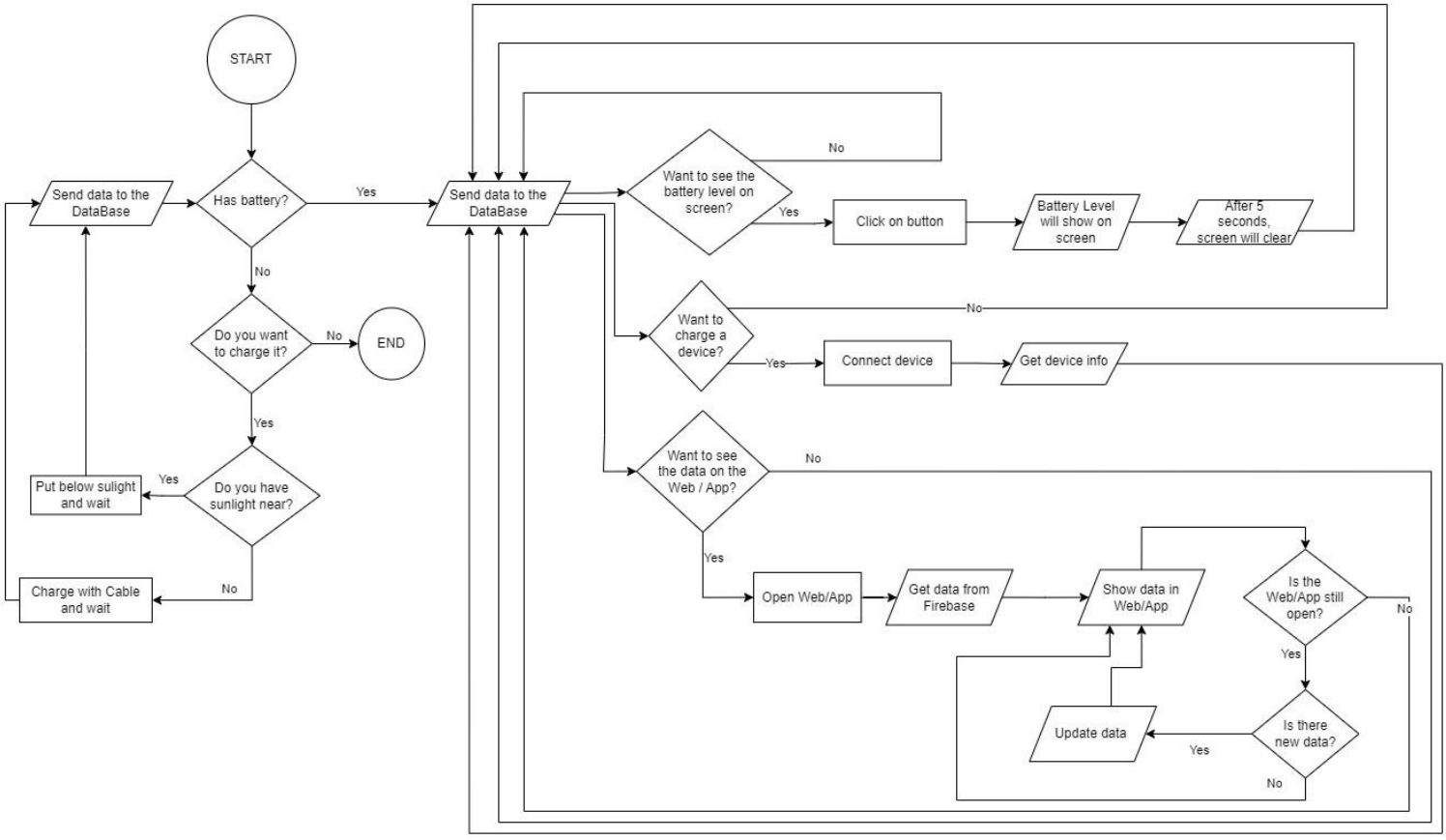
As for the implementation of the app we thought using Android Studio as is a program we have used and will be using during this year and is useful for creating mobile apps easily including the custom functionalities we want to add to this part of the project in a language we are all familiar with as is Java.

At first, we created a first page that would scan the specific QR from the portable charger, so it can access to the data from that specific prototype, and not all of the ones created (to have all portable charger data in the same database, but enabling access each one separately through that QR scanning). As we finally don't have yet a QR due to the fact that we don't have a domain, we programmed that screen, but skipped directly to the main one.

The main one displays information retrieved from a Firebase Realtime Database. It listens for changes in the database and updates the UI accordingly. The *charge()* method updates the UI elements based on the battery percentage values retrieved from the database. It dynamically changes the images and text to represent the charging status of the portable charger and connected devices.

Additionally, it includes a switch to toggle between dark and light modes for the UI.

Flowchart



5 – Flowchart

I've created this flowchart to offer a clear understanding of how our project operates. As represented above, the portable charger continuously transmits information to the real-time database in Firebase, ensuring that we have the most accurate data available. This process

only stops when the portable charger exhausts its battery, and the user opts not to recharge it. It's unusual for the portable charger to run out of battery, as it automatically reloads itself using sunlight received by the solar panels when the remaining charge in our 18650 rechargeable lithium battery is low.

In this flowchart there are 3 main functions:

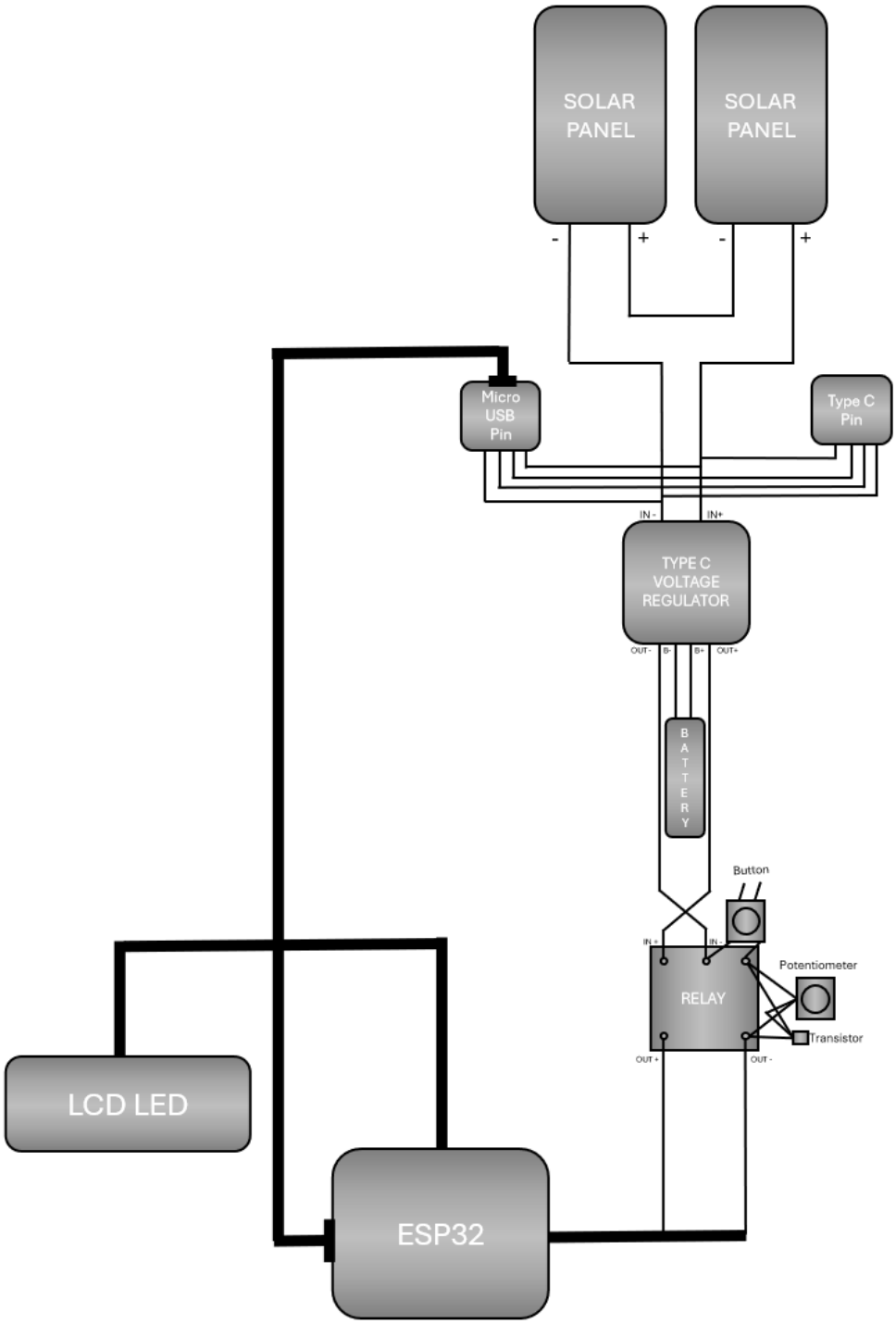
1. To view the battery level on screen: For this to happen, the user will need to click the button, and they will see the battery level displayed for 5 seconds. This feature was implemented for convenience and to avoid constant stimuli for individuals with ADHD from our portable charger.
2. To see the data on the Web or on the App: I have combined the function of viewing the data on both the web and app into a single function because, despite being two different platforms, the operation remains the same. First, the user will need to open the Web/App, which will fetch the data from the database and update the interface with the latest data. If new data is received while the web/app is open, it will be automatically updated. This continuous updating of data on both the web and app will persist until the Web/App is closed.
3. To charge a device: The only thing that the user will need to do if they want to charge a device, will be to simply connect the device to the portable charger. The database will create a new device object and will store new information about this device.

Implementation

Parts Required

- ESP32
- 2x Mini Solar Panels
- 18650 Lithium Li-Ion Battery
- Battery Holder (optional, not used in example)
- 18650 Type-C Voltage Regulator (MCP1700-3302E can be used instead)
- 16x02 LCD Display Screen
- B103 Potentiometer
- 127k Ohm Resistors
- USB C to Micro USB 4-Pin Port
- Male Micro USB 4-Pin Port
- USB Female Type A 4-Pin Port
- 2 Buttons
- 12V Relay
- 100K Potentiometer
- 2N2222A Transistor (you can also use the BC547 Transistor)
- Female to Female Jumper Wires
- Male to Female Jumper Wires

Circuit Overview



6 - Circuit Overview

This is a circuit overview, now I will explain how to connect each separate part, but the final circuit should look like the one above.

Short Circuit Protection

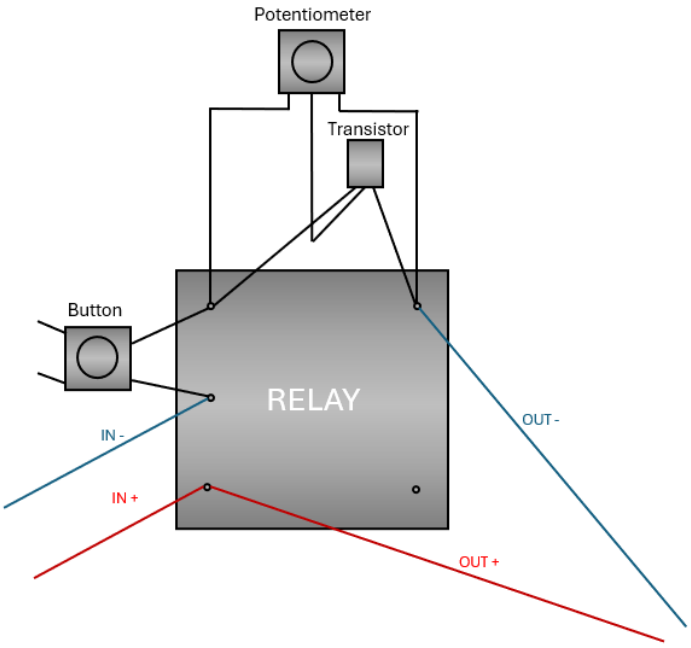
For this part, you will need:

- 4 Jumper Wires
- 12V Relay

- 100K Potentiometer

- 1 Button

And you will need to solder them as it's shown on the following diagram:



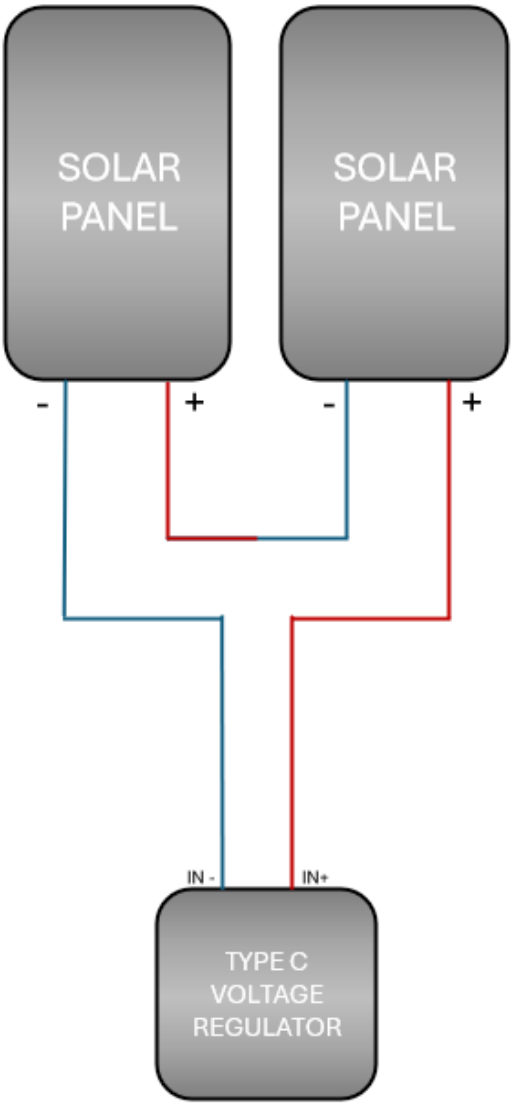
7 - Short-Circuit Protection Diagram

Solar Panels

Now, we will need to connect the Solar Panels with the voltage regulator module.

Each solar panel possesses a negative and a positive cable. We'll take the negative (blue) cable from one solar panel and solder it to the IN- terminal of the voltage regulator module. Similarly, from the other solar panel, we'll do the same with the positive (red) cable but with the IN+ terminal of the voltage regulator module. As for the remaining two cables, we'll solder them together to connect them in series. This approach is chosen because series connection elevates the overall system voltage while maintaining a constant current, whereas parallel connection escalates the total system current while keeping the voltage constant.

We can visualize this explanation with the diagram below.



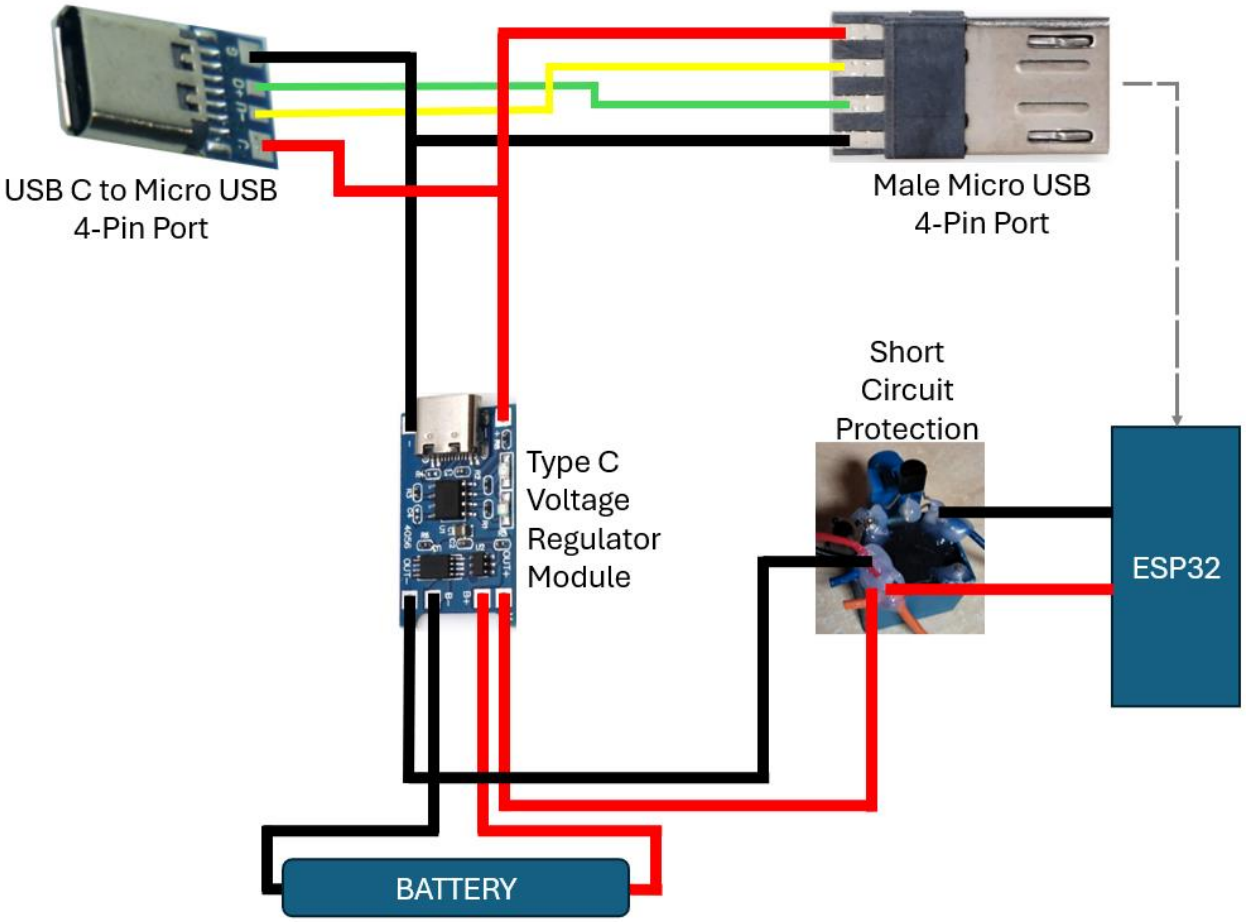
Voltage Regulator, USB Ports, and Battery

The USB C to Micro USB 4-Pin Port will be used to charge the Battery in case the solar panels are not enough. The Male Micro USB 4-Pin Port will be later connected to the ESP32.

The red cable shows the voltage, the black cable shows the ground, and the green and yellow cables are the data (D+ and D-).

As we can see in the diagram below, these ports are connected to the input of the voltage regulator module, so they can charge the battery if needed.

The battery is connected in the output of the voltage regulator module, in the 2 middle spaces, that are specifically dedicated to the battery (B- and B+). The two outer spaces are the ones that go to the Short Circuit Protection, that later will go to the ESP32.



16x2 LDC Display Screen

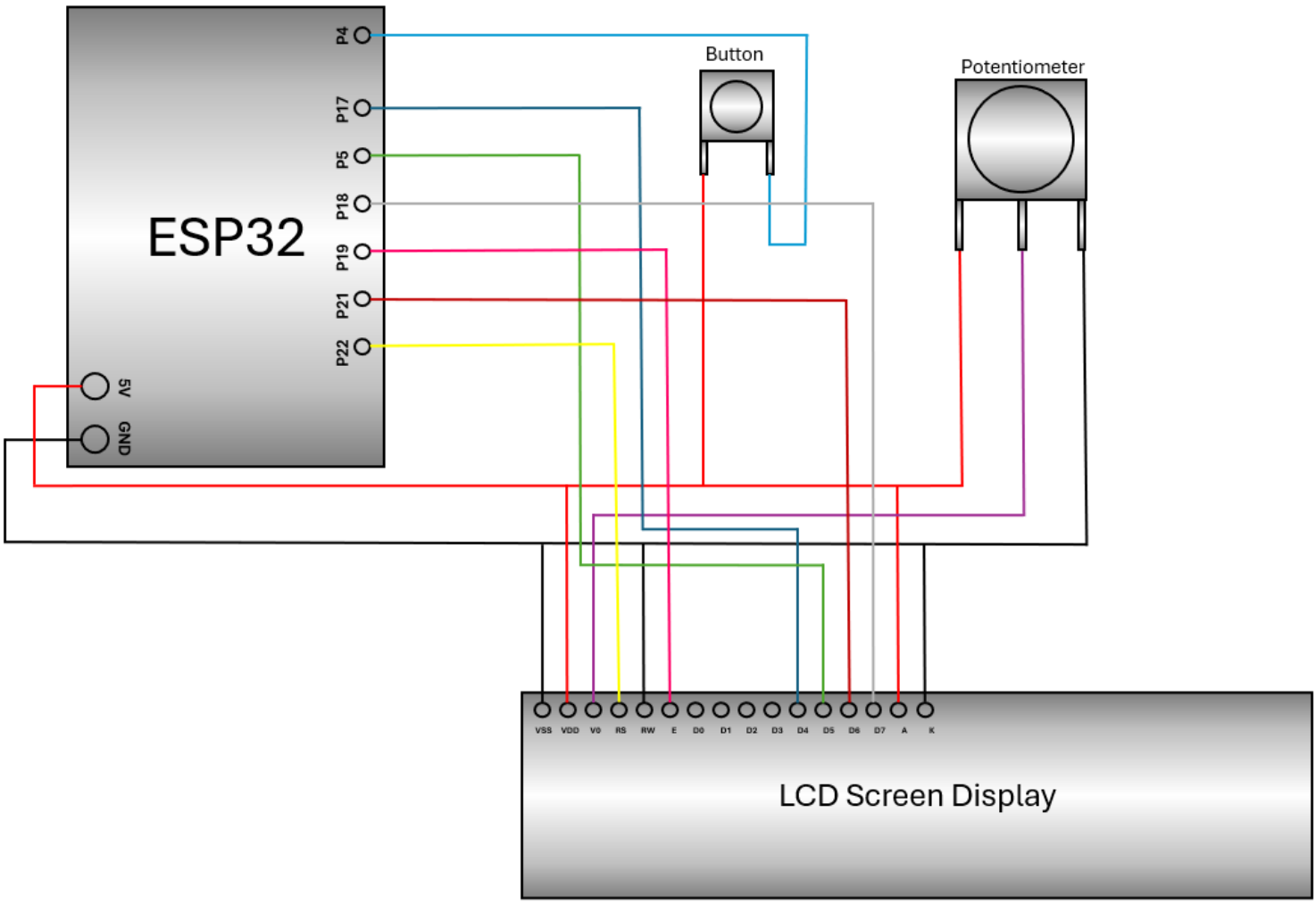
This connection is a bit more complex than the previous ones, as it involves more cables, it requires extra care.

For this part, we will use a button, a B103 Potentiometer, the ESP32, the LCD Display Screen, and many jumper wires.

Personally, I'm not too concerned about the colours, except for the red ones and the black ones, as they represent the 5V and Ground, respectively, and should be easily distinguishable.

From the ESP32 we will use the following pins: the 5V, the Ground, P4, P17, P5, P18, P19, P21, and P22.

We will connect the wires as shown below. As I mentioned before, the choice of wire is not critical, except for the red and black ones. It's essential to pay close attention to the pins being used (the numbers and to where they are connected in the LCD Display Screen), as they'll need to be accurately noted in the code for proper functionality.

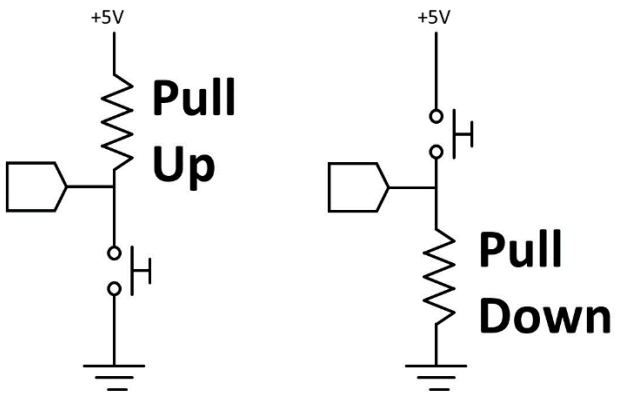


Once the wires are connected, we will need to program the ESP32 to show the information you want when you click the button. I've programmed the ESP32 to show the battery percentage for 5 seconds once the button is clicked.

Take into consideration that, for this to happen, you should have connected everything as its shown on the Circuit Overview Diagram showed on the early stages of the Implementation section.

To code the functionality of the LCD Display Screen, you should initialise the variables for the button, the pin sensor for the light and initialise the LCD object. For the setup, I initialised the lcd lines (rows and columns), the series baud and that the pin from the button will be an input. Then, I started with a loop designed to continuously monitor the Battery Level and transmit it to the Database (which I'll elaborate in the following pages). Nested within this loop, I created a function that will read the battery level and save it into an integer variable. Additionally, I formulated an if statement conditioned to activate when the button is clicked, triggering a separate function tasked with displaying the Battery Level on the screen.

An interesting element to point out is that, when configuring the pin using the pinMode() function as either an input or output, we opted for INPUT_PULLDOWN rather than INPUT or INPUT_PULLUP. By selecting the pull-down option, we're setting the pin as an input with an internally enabled pull-down resistor. So, when the pin is unconnected, it's pulled down to ground. In contrast, selecting the pull-up option configures the pin as an input with an internally enabled pull-up resistor. Therefore, when the pin is unconnected, it's effectively pulled up to the voltage level of the microcontroller's supply voltage. We have not used the INPUT option, because doing so would indicate that the pin is an input without any internal pull-up or pull-down resistor enabled.



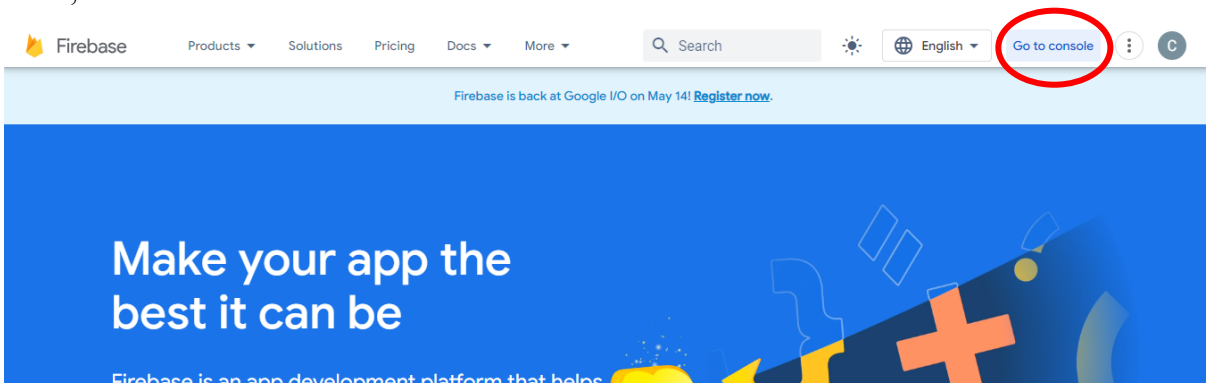
Please refer to Appendix 1 for the code corresponding to the explanation provided above.

Creation of the Firebase Project

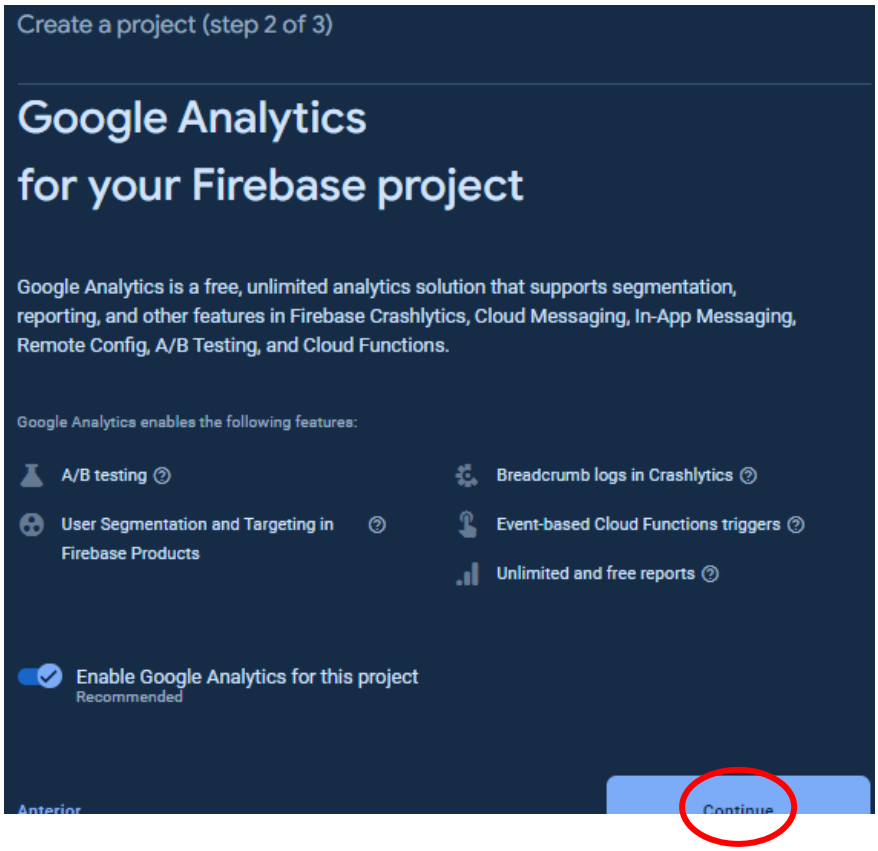
To create a Firebase Project, you should first go to the Firebase web:

https://firebase.google.com/?hl=en-419

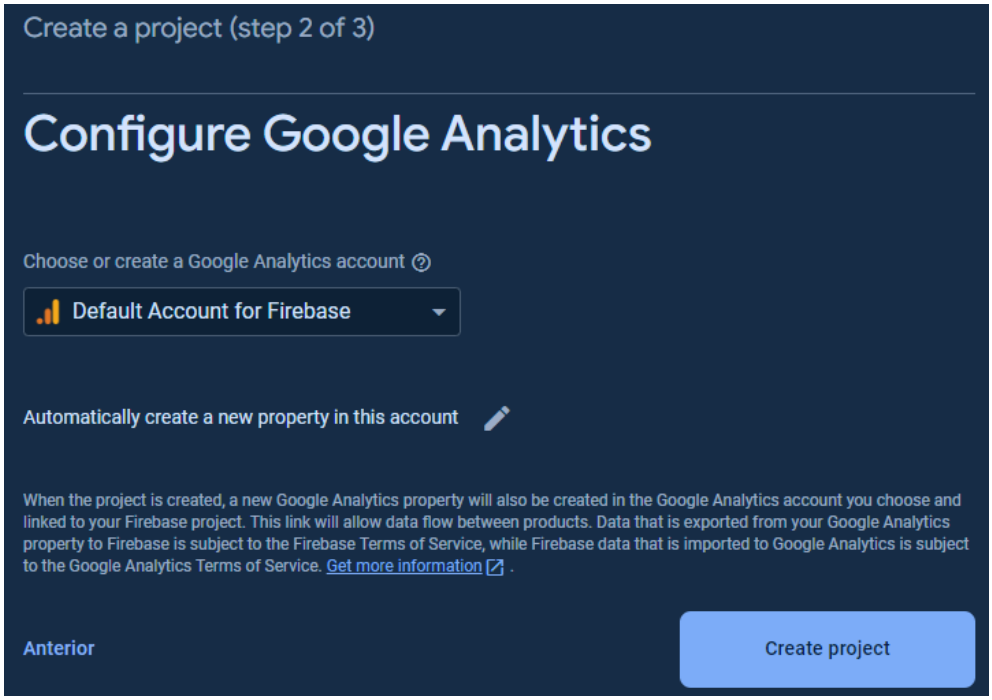
Then, click to "Go To Console"



Click to "Add Project" and it will start by asking you to insert a name for the Project. When you insert it, click "Continue" and, in the next page, click "Continue" again as its shown in the image below.



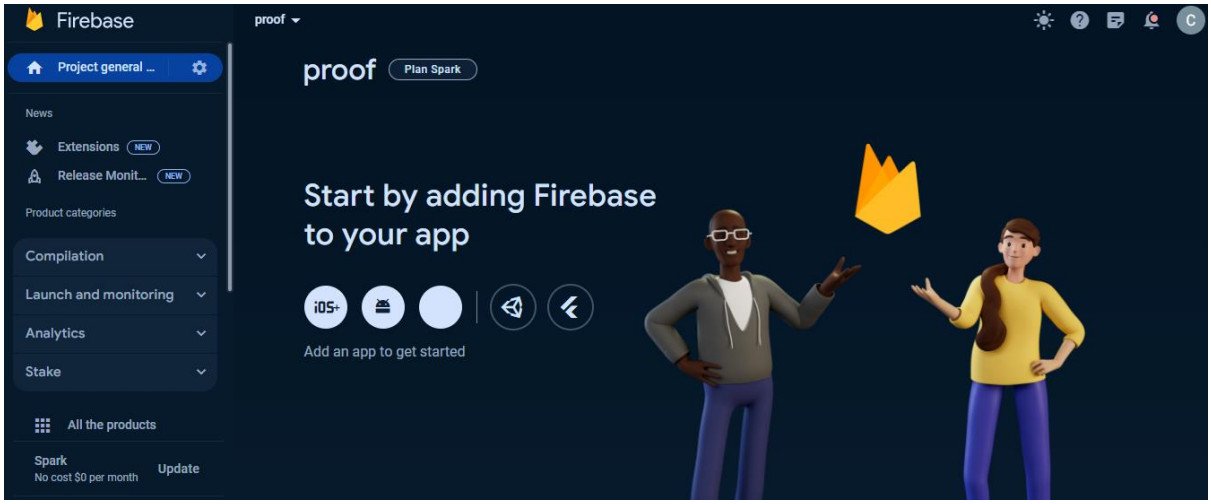
Then, the following page appeared, and we chose to use the Default Account for Firebase, but you can create an account if you prefer.



Finally click on “Create Project”. Sometimes it takes a couple of minutes to create, do not worry.

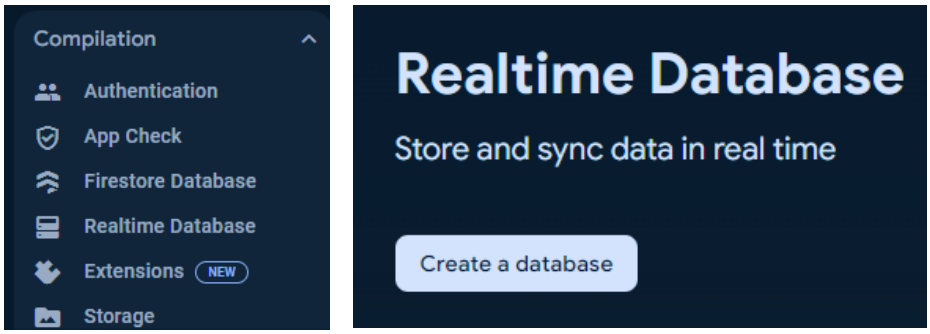
Creation of the Realtime Database

When the project is created, a page like this should appear:



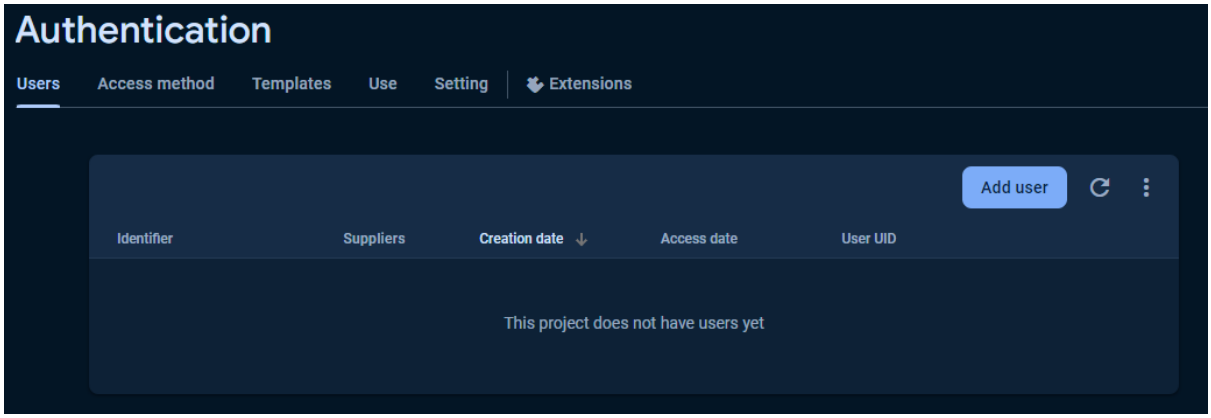
To create the Realtime Database, you should click on “Compilation”, and there, the fourth option is “Realtime Database”. Then, click on “Create a database”, and add a location and

security rules. For our project, I selected the default location, the United States (us-central1), and for the security rules, the “Start in trial mode”.

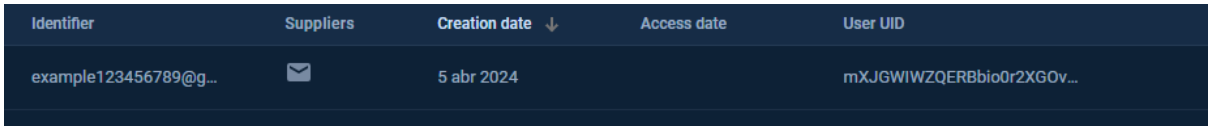


Then, click “Enable”, and the database will be created. Now you can see that the Realtime Database is creates, and you have there the address of the database, and in the header, you can take a look (and edit) at the Rules, Backups, and data about the Use of the database.

For our Project, we also created an Authentication method with an invented email and password, by clicking in Compilation → Authentication → Begin → E-mail: Password (but you can choose any authentication method you like).



Then, you go to Users → Add user. You add an email and a password, and add the user. Once the user is added, it will appear something similar to this:

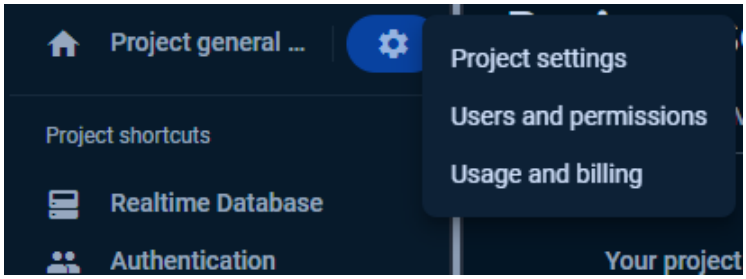


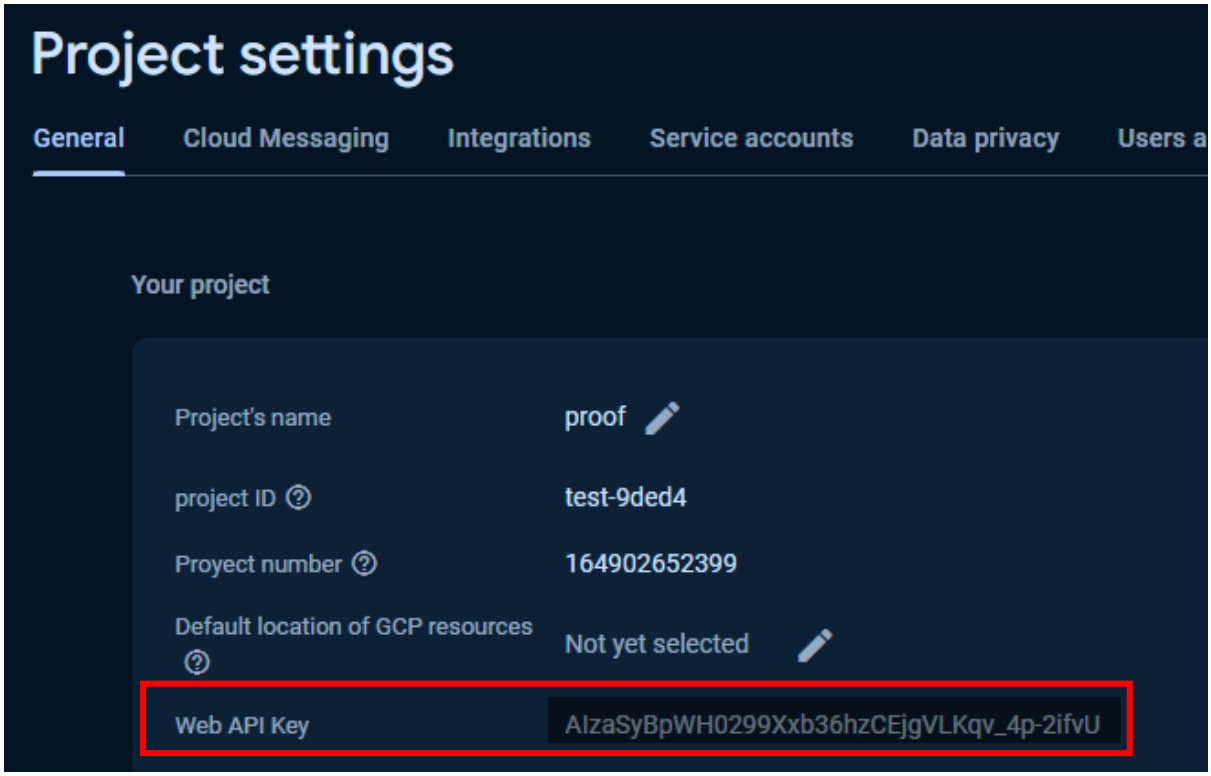
ESP32 Connection to WiFi and Firebase

In the Arduino IDE code, you should first insert your network credentials. The ESP32 will only be able to connect to that network until changed in the code. Then, Insert the Firebase API Key, the Email and Password you created during the Authorisation part of the creation of the Firebase Project.

To know where you can find the API Key of your Firebase project, you can do the following:

Go to Project settings → General → Web API Key:





You can use the structure below to assign the data into variables, for their further use.

```
// Insert your network credentials
#define WIFI_SSID "WiFi-name"
#define WIFI_PASSWORD "wifiPassword"
// Insert Firebase project API Key
#define API_KEY "change_for_API_of_your_project"
// Insert Authorized Email and Password – the example email you created
#define USER_EMAIL "example123456789@gmail.com"
#define USER_PASSWORD "1234567890"
// Insert RTDB URL – to later connect to the Firebase Database
#define DATABASE_URL "https://portable-charger-cb4b5-default-rtdb.firebaseio.com/"
```

Now, you should define Firebase objects to integrate the Arduino Project with the Firebase Realtime Database, Authorisation and Storage; such as the *FirebaseData*, *FirebaseAuth* and *FirebaseConfig*.

It will be useful to create other 2 variables:

- A sendDataPrevMillis variable which will control the frequency of data sending operations to prevent flooding the database with frequent updates. We can control this in the code when writing in the database the percentage of the battery level of the portable charger.
- A signupOK variable which indicates whether the user sign-in process was successful or not. This way, only people authorised to enter data to the database will be able to do it.

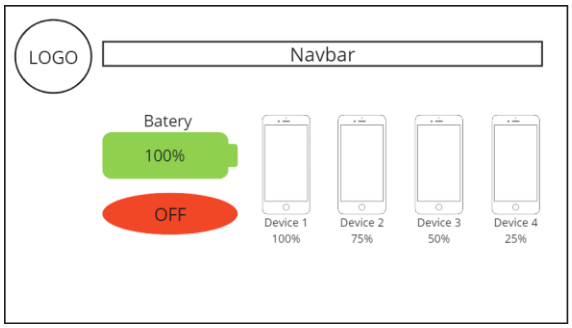
For the WiFi initialization:

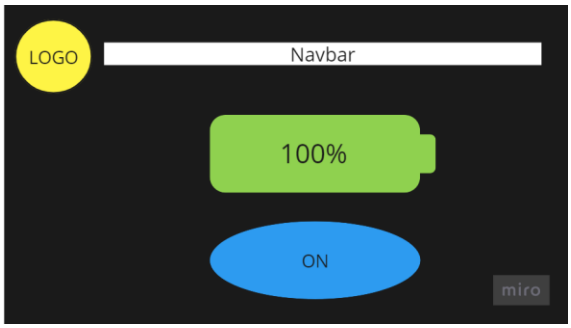
I created a function to initialize the WiFi connection for the ESP32. This function connects the ESP32 to the specified WiFi network using the SSID and password defined earlier. I programmed the code for when it´s connecting it should write in serial '.', and when it found the IP, write the IP to show that it's connecting to the correct one.

Creation of the Web

For the Web, we created a very basic and simple HTML website, and we added CSS for the design and style editing, and we used JavaScript for the interactive elements.

We used *Miro* to create our initial design for the webpage, and it was the following:





The design initially comprised a straightforward ON and OFF display setup. Initially, the plan was for the portable charger to exhibit the names and battery level percentages of both the battery and connected devices when it was ON. On the contrary, when it was OFF or had exhausted its battery, it would switch to a different screen (the one on the right). However, later on, we realized that we hadn't yet figured out to program how to physically turn the portable charger on and off. As a result, we opted to maintain the display configurations for both the ON and battery-depleted states.

The HTML code is straightforward, consisting of a header section and a element designated for displaying battery information. Additionally, two more elements are dynamically created upon retrieving data from the database. Each corresponds to a connected device and displays its name and battery level. While our current setup only supports connecting one device at a time, we've programmed it to showcase how the interface would appear with up to two connected devices.

Feel free to customize the CSS according to your preferences. However, I've included our CSS code in the appendix for reference, despite its simplicity.

Please refer to Appendix 2 for the code corresponding to the Web Page.

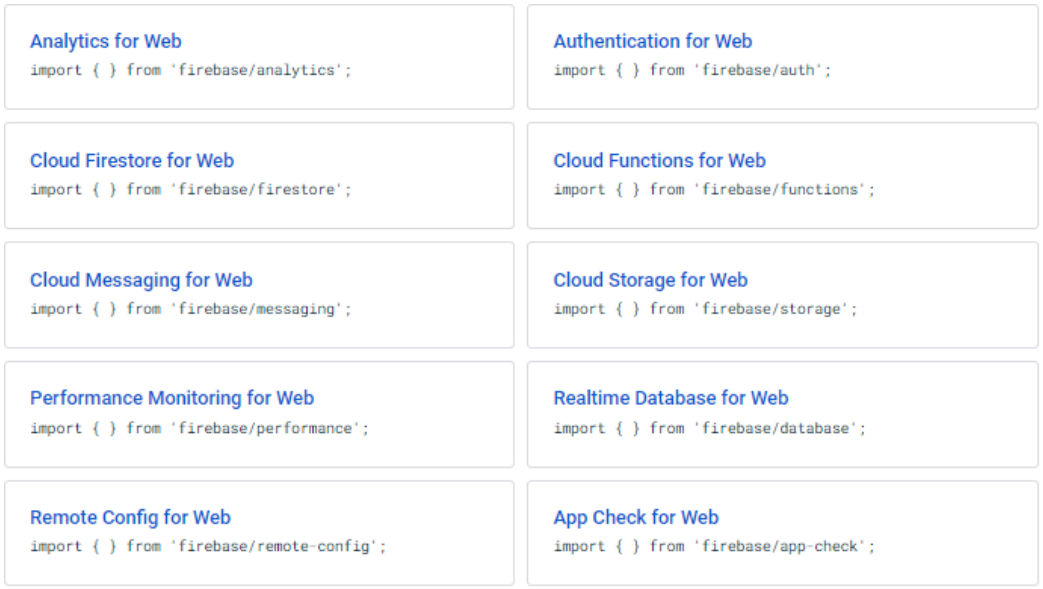
Connection of the Web to Firebase

Initialize Firebase in your app and create a Firebase App object. You should import the project and the necessary services. Later, replace the defined constant as *firebaseConfig* with

your app's Firebase project configuration. Then, you can store what you imported in different variables so you can use them later in the code easier.

For general information (not specific of this project) visit the official Firebase page which will explain in detail how to connect the Firebase project with your JavaScript Web: https://firebase.google.com/docs/web/setup?hl=en

There are general Firebase services available for web:



Web: JavaScript Functions

First, I will provide an explanation of the code functionality, followed by a version of the code with comments for clarity and understanding.

The function *getPeriodic()* is designed to periodically recover data from the Firebase Database. It compares this data with a reference snapshot, which represents the state of the data at the specified location (dataRef) when the function is initially executed as if it was a photo. If there are any changes in the Realtime Database, indicating new data; the callback function (snapshot) => {…} will be activated. This callback function then calls two other functions we've created. First, it clears the existing section content to prepare to upload the updated data. Then, it adds a for each device present in the Realtime Database.

The code will log the device name and battery level percentage in the Console. This allows for when anyone inspects the webpage using browser inspection tools, they can see an easy and clear structure and formatting.

Moreover, the functions *createDeviceDiv()*, *destroySectionContent()*, and *destroyNonBatteryDivs()* will be explained below.

The function *addDevice()* calls the function *createDeviceDiv()* to simply create a div for each device that is in the Realtime Database. As I mentioned before, when there's a change and the callback function (snapshot) => {…} is triggered, it will also call *destroySectionContent()*, that will first check if the section exists, and if it does, it will remove all child elements of that section.

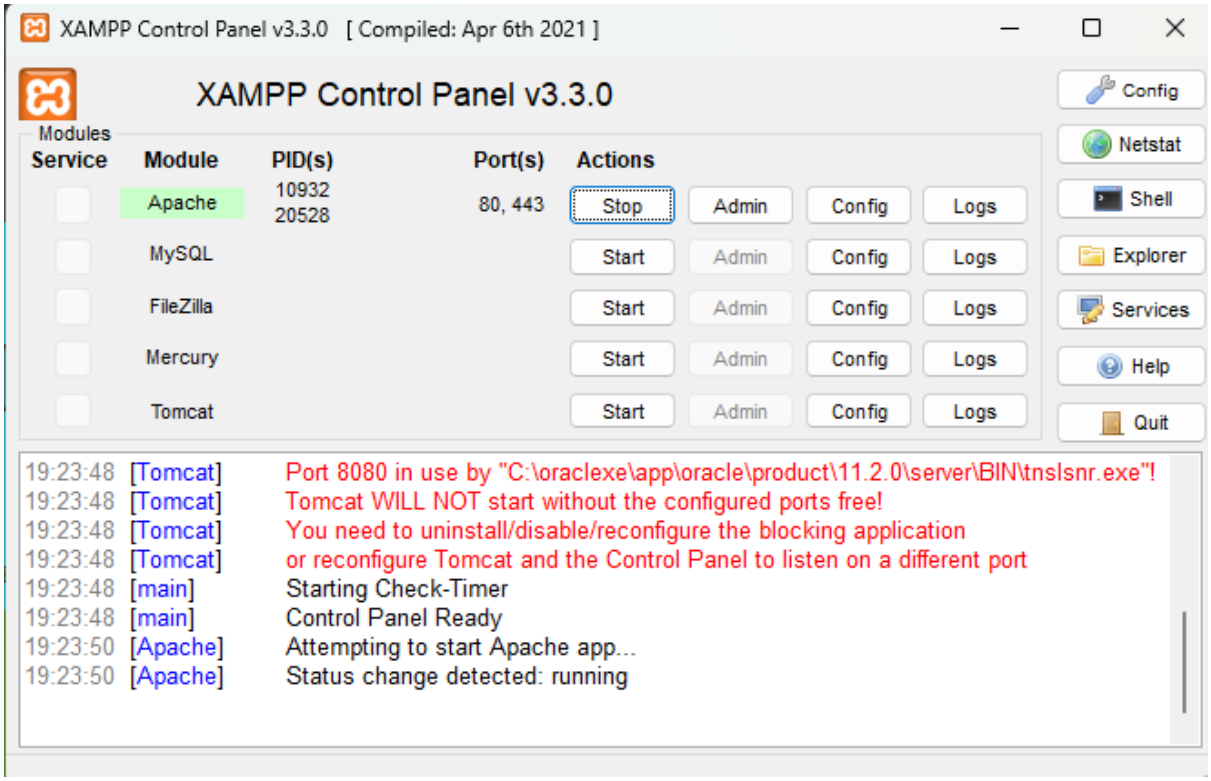
We've also programmed to be able to turn the device On and Off through the web page, as an example of its functionality. However, it's not yet connected/linked to the prototype because we lack the necessary knowledge (this is explained in the "Future Developments" section). Despite not being utilized at the moment, we have programmed it so, when we can link it, we already have the code functionality; so, I'll explain how the code works. The function *destroyNonBatteryDivs()* is triggered when the user clicks the button to turn off the device. When the user clicks the On/Off button, it calls the methods *document.getElementById("off-btu").onclick()* and *document.getElementById("on-btu").onclick()*. When the button is clicked On, it will show all of the information about the battery and devices connected. On the contrary, when the Off button is clicked, it will destroy both device elements, but not the Battery one.

Please refer to Appendix 1 for the corresponding code.

Open the Web ensuring its tied to the Firebase

As we don't have a domain for our web page, in order to open the html while connecting it to Firebase, you should do the following.

1. Install (if you don't already have it) and open, the XAMPP Control Panel.
2. Save the Web Project folder into: C: → xampp → htdocs
3. Start the Apache module.
4. Go to the web browser and write: *http://localhost/webProjectFinal/web-projecto.html*



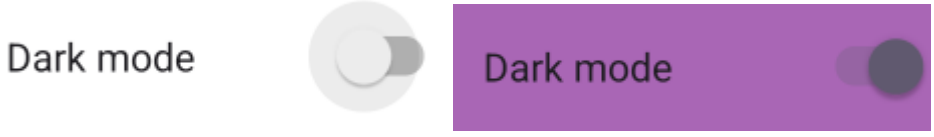
There you will have the html opened properly in order to have the web connected to Firebase. If you *Stop* the Apache module in the XAMPP Control Panel, you risk for the web not to update its information or to work correctly.

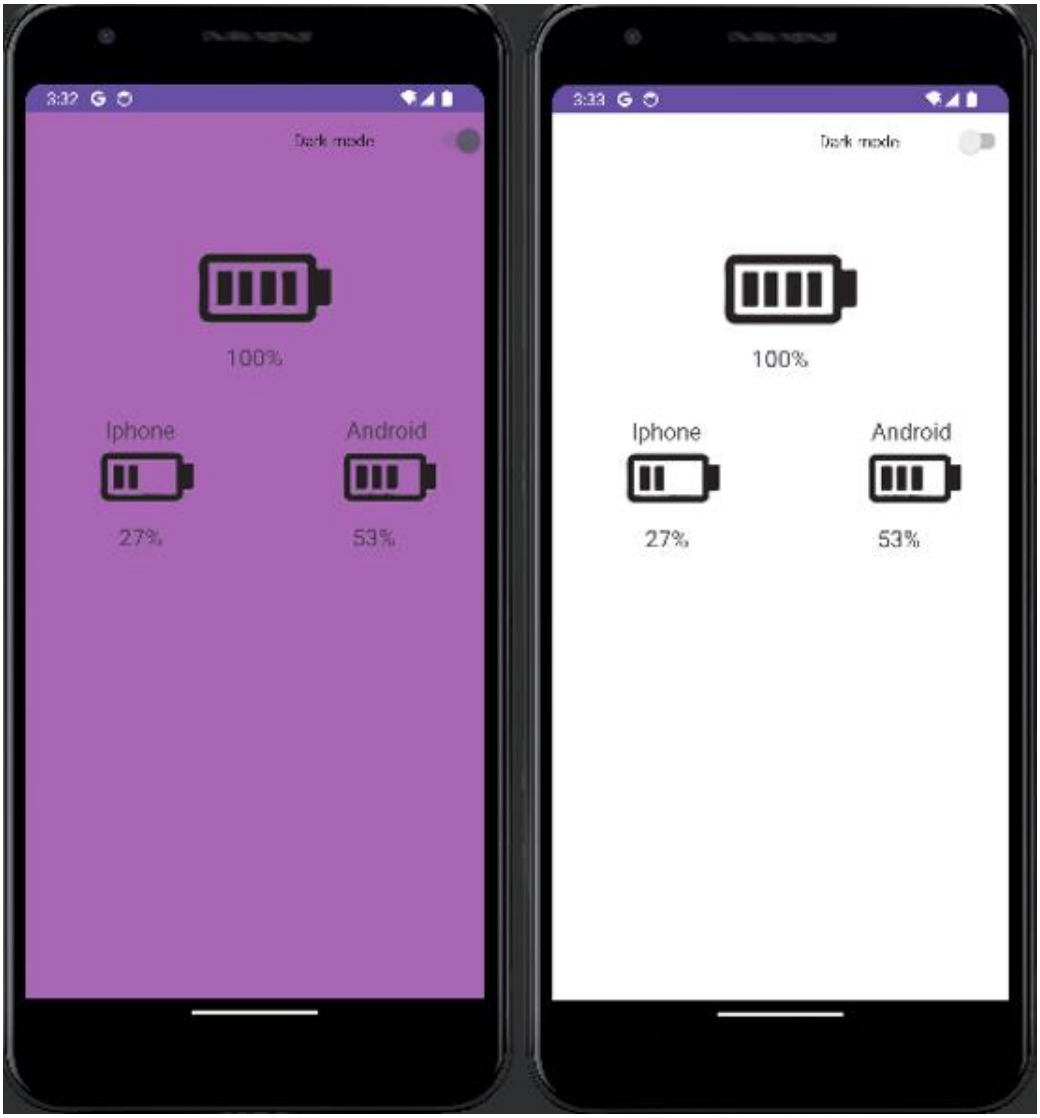
Creation of the Mobile App

For the mobile app, we wanted to provide a battery percentage display for the propotype, along with the names and battery percentage of the devices, which are in the Firebase Realtime database.

An interesting feature we implemented is that depending on the battery level, the battery icon changes depending on its level. one line between 0-25%, two between 26-50%, three between 51-75%, and the full four lines between 76-100%.

Users could personalize the app's look through a light and dark mode toggle, allowing customization of appearance by preference. Varied sentence structures and lengths were employed to boost both perplexity and burstiness for a more human voice.





8 - Mobile App

Mobile App: Realtime Database and Firebase integration

In this section, I'll explain some key features of the Mobile App's coding that enhance its functionality and user experience. From real-time data sync with Firebase to dynamic battery level display and dark mode implementation.

When the app starts, various views are set up to represent the user interface elements. This includes TextViews for battery percentages and device names, ImageViews for battery icons, and a Switch to toggle dark and light modes.

The app also connects to the Firebase Realtime Database using a ValueEventListener on a DatabaseReference. This listener triggers the *onDataChange()* method whenever the database data is updated. This ensures the app always has the latest and most accurate information.

Inside the *onDataChange()* method, the battery information is retrieved from the database. This includes the charge level of the charger and two devices, along with their names. Then, the retrieved data is passed to the *charge()* method to update the user interface.

The *charge()* method is responsible for updating the user interface based on the battery information obtained from the database. It sets the visibility and images of the ImageView widgets based on the battery levels, allowing users to easily visualize the charge status of each device.

The *onClick()* method attached to the Switch enables users to toggle between dark and light modes. When the Switch is checked (indicating dark mode), it changes the background color of the activity to a darker shade, providing users with a customizable visual experience.

All in all, by integrating Firebase Realtime Database functionality with Android UI components and offering a dark mode option, this code enhances the user experience by providing real-time battery information and customization features.

Testing & Evaluation

During the initial weeks of the project, as we were still in the research phase and had limited experience with connecting components and using the ESP32 — it being my first time using it — we began attempting to connect the battery to the ESP32.

When I researched which battery to use, I thought of the 9V **battery**, because I already had one we could try and use, and it had a high voltage to work with. However, I soon realized that the battery I chose would not be suitable for our prototype. This was due to the 9V battery's inability to be recharged, which posed a risk of damaging the ESP32 if it received a voltage of 9V, potentially rendering it unusable.

While searching for the **18650 rechargeable battery** we encountered the problem that any shop we went in Athlone didn't have this kind of battery. Therefore, we needed to buy this type of battery online, and suffer the time delivery to try and test the prototype.

While we were waiting for some of the components to arrive, I was exploring potential solutions for managing **overcharging** issues with the device; and during my research, I came across the concept of **short-circuit protection**. As we started building this component, we had to make sure the correct soldering of the positive and negative wires. This was essential to ensure that the current flowed properly and did not pass the positive terminal to the ground or vice versa.

Despite the short-circuit offering several benefits, there is one potential problem that could arise with its use. When creating this component, we need to take care of the potentiometer and its positioning. Depending on where it points, it may function correctly or not because its orientation determines the resistance value it presents to the circuit. If the potentiometer is not properly aligned, it could introduce inconsistencies in the circuit, and affect the overall performance and accuracy of the device.

So, the short-circuit protection, when set up correctly, protects the ESP32 and the battery of overheating and overcharging. So, following the diagram showed in a previous section, you will be able to build up in the correct way this short-circuit protection component.

We had an idea to create a specific **QR** for each prototype, so anyone could access the web and app through that QR, with the specific information gathered from that prototype. While researching how to create a QR to place in the prototype, we found the problem that we could only create a temporary QR without paying. But, as we don't have yet a domain, we can't create the QR.

I bought a Type-C to Type-C cable to test whether we could **charge a device** through the voltage regulator module's input port. However, during testing, we encountered the issue that the setup didn't work as expected. After some investigation, a couple of days later, we identified the problem: the cable was connected to the module's input port, while the battery was connected to the output port. Therefore, when the cable was connected to the module, the connected device attempted to charge the battery instead.

To **connecting** the Arduino project to the web and app, I researched on how to use Node-Red to connect it all. But everything I researched was too difficult to do, or it was contradicted by the next thing I read. So then, I researched about using **Firebase**. I researched on how to create a Firebase project, and then add a Web (HTML) project, and a Mobile App project. Everything I researched looked easier and I started connecting the web and mobile app project. Everything seemed correctly while connecting Firebase with both projects, by following the Firebase guidelines.

While programming the **LCD display**, I encountered some errors. Initially, when connecting the wires between the LCD and the ESP32, I needed to ensure everything was correctly connected. Unfortunately, I mistakenly swapped two wires: the 5V and ground wires were connected to the K pin and the A pin, respectively. As a result, the display didn't function properly. Upon rechecking the wire connections, I discovered the mix-up and corrected it by connecting the 5V wire to the A pin and the ground wire to the K pin. Then, the LCD display worked perfectly.

When soldering the **solar panels** to the voltage regulator module, we tested what voltage both the input and output pins from the module gave. We realised that soldering them **in series** was better than soldering them in parallel, because it gave a higher voltage output.

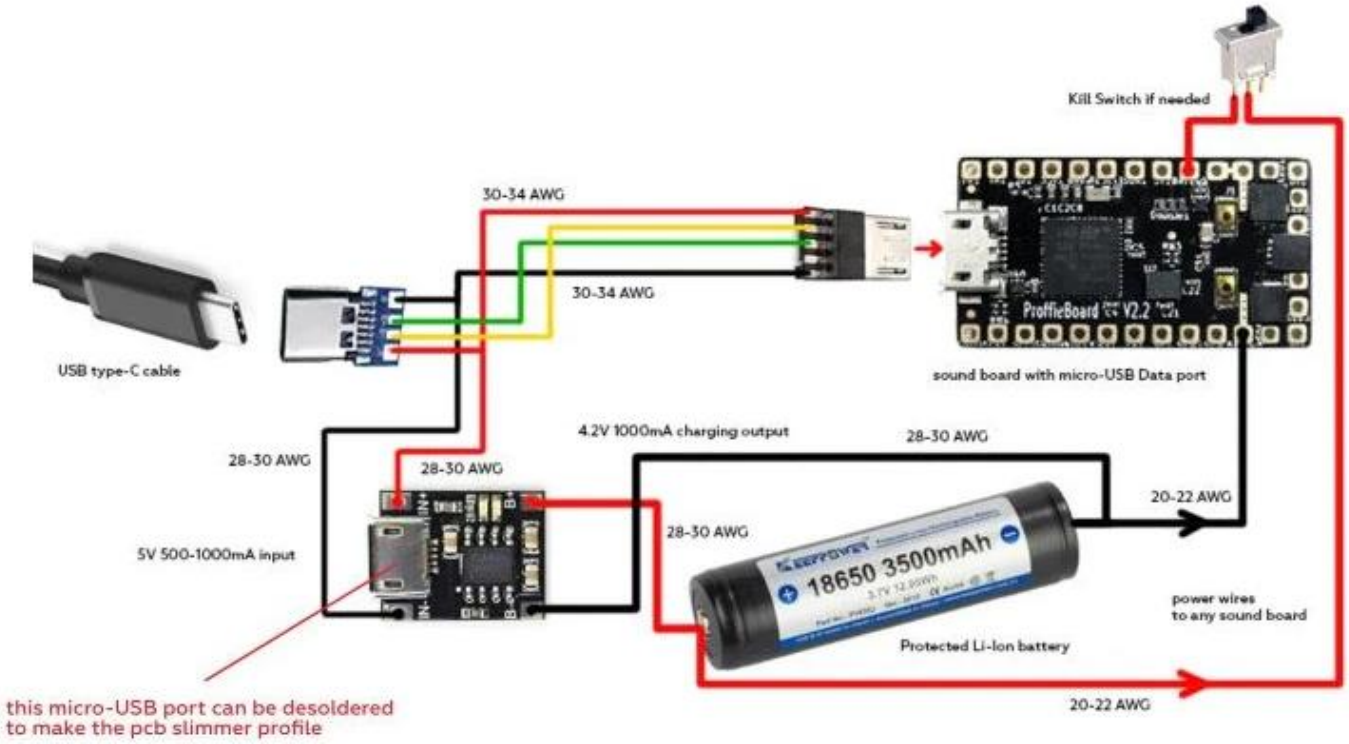
When looking how much voltage gave out the module, we realised that it gave very little voltage just with the energy received by the solar panels. To give a higher voltage, we should get more panels or just one panel, but it being very powerful, such as 6V or more.

While coding the ESP32 to connect to **WiFi**, we encountered some challenges regarding which **libraries** to install and how to establish the WiFi connection. However, even after successfully connecting the ESP32 to the WiFi network, we faced issues with analog pin readings. Some pins that worked fine in code without WiFi functionality failed to produce the expected results when integrated with WiFi-related code.

When trying to read the battery level using **analog pins** while the ESP32 was connected to WiFi, I initially used pin 2 and later pin 4 for convenience. However, despite these attempts, the readings were inaccurate, prompting further investigation into the underlying issue. I then discovered that certain pins on the ESP32 did not function properly with analog pins while the device was connected to WiFi.

After some research, I found out that the ESP32 features two ADC (Analog-to-Digital Converter) peripherals: ADC1 and ADC2. The ADC1 peripheral is fine to use at the same time as WiFi (pins 32 to 39). But the ADC2 needs to be handled carefully as it is also used by the WiFi peripheral (pins 0, 2, 4, 12 to 15 and 25 to 27).

After having all the necessary components connected, we purchased additional ports from The Saber Armory to expand the functionality of our portable charger. Following the wiring diagram provided on the website (shown below), we expected to **charge a device** through the portable charger. However, upon testing, we discovered that the connected components were actually charging the portable charger instead of the intended device.



9 - The Saber Armory connection diagram

Another challenge appeared when attempting to operate the ESP32 without the need for it to be connected to the computer via a cable. Research led me to discover that an Over-the-Air (OTA) connection was required for this purpose. However, we have yet to resolve this issue. Additionally, during our investigation, we found that providing the ESP32 with only 3.3V was insufficient to power it and ensure proper functionality of the components. We determined that a minimum of 5V was necessary for optimal performance.

Towards the end of the project, we encountered another challenge while attempting to retrieve data from the connected device, specifically the device name and its battery percentage level. Unfortunately, we have not yet resolved this issue. As a workaround, we simulated the scenario of having two devices connected to the portable charger by manually creating these two devices in the database. This allowed us to demonstrate how the data would be displayed on the web and app interfaces if there were 2 devices connected.

During the development of the web and app interfaces, we encountered difficulties in ensuring dynamic data updates without the need for manual refreshing. To address this challenge, we implemented a solution where the web/app continuously monitors the real-time database to verify if the displayed information matches the data stored. If any inconsistencies are detected, the system automatically clears the current display and updates it with the most recent information from the database.

We finally attempted to charge a device (a mobile phone) by connecting the USB female Type A 4-Pins port, specifically by linking its two lateral pins to the ground pin and the 5V pin of the ESP32. While we successfully enabled the phone to charge, it remains uncertain whether the charging occurred through the portable charger or as a result of the connection to the computer via cable.

Future Development

As we finish the project, there are numerous areas where I wish I would've had time and knowledge to enhance and expand upon, to elevate the project to the next level. Here are some of the potential future developments for the project

Enhanced Web Interface: One of the next steps in the development of our project involves refining the appearance and functionality of the web interface. This includes customizing the CSS to create a more polished and professional look. By updating the web page's styling, we aim to improve user experience and make the interface more visually appealing.

Custom 3D Case Design: As part of our project's evolution, we plan to 3D print the case I designed for the portable charger. These custom cases will not only provide aesthetic enhancements but also ensure optimal protection and organization of the components within the charging station.

Integration of NFC Charging: In the future, it would be a nice step to integrate NFC (Near Field Communication) technology into our portable charger system. This will enable users to charge their devices wirelessly by simply placing them near the NFC-enabled charging station, in case users don't have a cable in hand. Implementing NFC charging functionality will require thorough research and development to ensure seamless integration and compatibility with various devices.

Automatic Stop Charging Feature: To optimize battery health and prevent overcharging, we plan to implement an automatic stop charging feature. This feature will be configurable through the app, allowing users to set a specific battery percentage threshold at which charging will automatically stop. Once the connected device's battery reaches the designated percentage, charging will cease, preventing unnecessary power consumption and prolonging battery life.

Support for Type-B USB Charging: Another planned enhancement is to enable charging of devices through a Type-B USB port, commonly known as the rectangular USB port. This

functionality will expand the versatility of our portable charger system, allowing users to charge a wider range of devices using different types of USB connections. Additionally, the system will be designed to retrieve and display the name and battery percentage level of devices connected through the Type-B USB port, providing users with comprehensive charging information.

Enhanced Battery Monitoring and Management: Implementing advanced battery monitoring and management features can further improve the efficiency and longevity of the portable charger system. This includes integrating real-time battery status monitoring capabilities, such as temperature monitoring, charge/discharge cycle tracking, and battery health diagnostics. Incorporating intelligent battery management algorithms can optimize charging protocols based on battery chemistry and condition, ensuring safe and optimal charging performance. By enhancing battery monitoring and management capabilities, users can gain greater insight into their device's battery health and maximize the lifespan of their batteries.

Conclusion

Our project aimed to develop a smart portable charger that not only efficiently charges electronic devices but also enhances the user experience with innovative features. Motivated by its practical utility, especially because each day individuals heavily rely on electronic devices, we chose this project for its potential impact. The project involved both hardware and software components, with a focus on usability, functionality, and reliability.

The hardware design consisted of various components, including the controller (ESP32), battery, button, voltage regulator, USB ports, LCD display screen, and solar panels. Each component played a crucial role in the system's operation, from power management to user interface interaction.

On the software side, the project involved developing a dashboard, web interface, and mobile app for monitoring and controlling the charger system remotely. Connectivity to Firebase Realtime Database facilitated real-time data exchange between the device and the user interface components.

Implementation of the project required careful consideration of parts required, circuit design, and integration of various modules. Testing and evaluation were conducted to ensure the system's functionality and reliability under different scenarios.

Looking ahead, future development opportunities include enhancing the web interface with customized CSS, 3D printing custom cases for the charger system, integrating NFC charging capabilities, implementing automatic stop charging feature, and supporting charging through Type-B USB ports. These enhancements would further improve the system's usability, expand its functionality, and enhance user experience.

In conclusion, this project represents a successful endeavour in creating a versatile and efficient portable battery charger system with advanced monitoring and control features. It demonstrates the integration of hardware and software components to address practical challenges and meet user needs in the realm of portable power solutions.

References

Type	Reference example	In-text citation
Website	Mouser electronics, 11 January 2007	*RP402N501E-TR-FE Nisshinbo	Mouser*(2007) https://www.mouser.es/ProductDetail/Nisshinbo/RP402N501E-TR-FE?qs=sGAEpiMZZMutXGli8Ay4kIusfOp5CmK9HK%252BFUcbv0Uk%3D&as_qdr=y15&_gl=1*1hrvolu*_ga*MjA1Mzk0MjM3NS4xNzAxMDAwOTEw*_ga_15W4STQT4T*MTcwMTE4MTU3Ni40LjEuMTcwMTE4NDU5OS41MC4wLjA. [24 nov 2023]
Website	Mouser electronics, 11 January 2007	*MCP1700-3302E/TO Microchip Technology	Mouser* (2007) https://www.mouser.es/ProductDetail/Microchip-Technology/MCP1700-3302E-TO?qs=h7tZ5KkzNMMPEB66r2rMQw%3D%3D [24 nov 2023]
Website	Farnell Components (Ireland) Ltd	*MC11605A6W-SPR-V2 - Alphanumeric LCD, 16 x 1, 5V, English, Japanese, Reflective* https://ie.farnell.com/midas/mc11605a6w-spr-v2/display-alphanumeric-16x1-nobacklight/dp/2674220?gad_source=1&gclid=CjwKCAiAsIGrBhAAEiwAEzMlC6LjNv1SJpK0gbbI5mNHb0_E2iDY2IDKM9uPX_ks3-QrXAdm-62ETRoCwxoQAvD_BwE&CMP=KNC-GIE-GEN-SHOPPING-PerformanceMax-Med-Margin-Test-1573&gross_price=true [24 nov 2023]
Website	Amazon (15/11/2023)	EEMB(2023) *EEMB Lithium Polymer battery 3.7V 2000mAh 103454 Lipo Rechargeable Battery Pack with wire JST Connector for Bluetooth Speaker and Dashcam-confirm device & connector polarity before purchase* https://www.amazon.co.uk/EEMB-2000mAh-103454-Rechargeable-Connector/dp/B08214DJLJ/ref=asc_df_B08214DJLJ/?tag=go

		ogshopuk-21&linkCode=df0&hvadid=463170821467&hvpos=&hvnetw=g&hvrand=10657995482494258613&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=1007902&hvtargid=pla-952213741191&mcid=b821b9ab22043636b36a4067db3927d8&th=1 [24 nov 2023]
Website (no date)	RadioShuttle	RadioShuttle *ESP32 Alimentado por Batería* https://www.radioshuttle.de/es/medias-es/informaciones-tecnicas/esp32-alimentado-por-bateria/ [23 nov 2023]
Website	Amazon (15/11/2023)	Hailege (2023) Hailege ESP32 ESP-32S ESP-WROOM-32 Development Board 2.4GHz Dual-Mode WiFi + Bluetooth Dual Cores Microcontroller Processor Integrated with Antenna RF AMP Filter AP STA for Arduino IDE https://www.amazon.co.uk/ESP-WROOM-32-Development-Dual-Mode-Microcontroller-Integrated/dp/B07YKBY53C/ref=asc_df_B07YKBY53C/?tag=googshopuk-21&linkCode=df0&hvadid=658814687030&hvpos=&hvnetw=g&hvrand=10707268939199282360&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=1007902&hvtargid=pla-860392095046&psc=1&mcid=8ebdb19acd703d809ea4a87eda3fba3c [23 nov 2023]
Pdf	For research	ESP32_CHIP_Pin_List_EN.numbers-Cover
Website	Random nerd tutorials	Power ESP32/ESP8266 with Solar Panels (includes battery level monitoring) https://randomnerdtutorials.com/power-esp32-esp8266-solar-panels-battery-level-monitoring/ [28 nov 2023]

Website	For setting up the Firebase with the web	https://firebase.google.com/docs/web/setup?hl=en
Website	Queries about foreach in json and JavaScript	https://stackoverflow.com/questions/37900558/javascript-json-array-foreach
Video	Short Circuit Protection	https://www.youtube.com/watch?v=UR-gqoUbLUc
Video	OTA ESP32 programming	https://www.youtube.com/watch?v=NyX7RtKpN8k
Website	AliExpress Solar Panels	https://es.aliexpress.com/item/1005006028436478.html?srcSns=sns_WhatsApp&spreadType=socialShare&bizType=ProductDetail&social_params=60460598968&aff_fcid=147e66eafa5b46748aa280eea405b11e-1708538463054-03825-_EQ12jzb&tt=MG&aff_fsk=_EQ12jzb&aff_platform=default&sk=_EQ12jzb&aff_trace_key=147e66eafa5b46748aa280eea405b11e-1708538463054-03825-_EQ12jzb&shareId=60460598968&businessType=ProductDetail&platform=AE&terminal_id=2ed97600946c4a8580438f4879818a6a&afSmartRedirect=y
Website	AliExpress Voltage Regulator Module	https://es.aliexpress.com/item/1005005455017968.html?srcSns=sns_WhatsApp&spreadType=socialShare&bizType=ProductDetail&social_params=60460600421&aff_fcid=25b34d97718f4f6a9d5a06cecd299414-1708538465541-09026-_EvgKJmV&tt=MG&aff_fsk=_EvgKJmV&aff_platform=default&sk=_EvgKJmV&aff_trace_key=25b34d97718f4f6a9d5a06cecd299414-1708538465541-09026-_EvgKJmV&shareId=60460600421&businessType=ProductDetail&platform=AE&terminal_id=2ed97600946c4a8580438f4879818a6a&afSmartRedirect=y
Website	Sketch initial visual design of web and app	https://miro.com/app/board/uXjVNLONPFo=/
Website	Firebase Authentication	https://randomnerdtutorials.com/esp32-esp8266-firebase-authentication/#firebase-project-api-key

Website	Power ESP32 through solar panels	https://randomnerdtutorials.com/power-esp32-esp8266-solar-panels-battery-level-monitoring/
Website	18650 Battery Shop we bought it from	https://ie.rs-online.com/web/p/speciality-size-rechargeable-batteries/8801558?cm_mmc=IE-PLA-DS3A-_-google-_-CSS_IE_EN_Pmax_Test-_--_-&matchtype=&&s_kwcid=AL!14853!3!!!!x!!&gad_source=1&gclid=CjwKCAiA_tuuBhAUEiwAvxkgThBBuDvEdvNy-y7RyMot_jwaQ2GE7-xluzXVIxmG6MD9eoZw1rGltRoCW-oQAvD_BwE&gclsrc=aw.ds
Website	Button and LCD led display	https://esp32io.com/tutorials/esp32-button-led
Website	Firebase Web Setup	https://firebase.google.com/docs/web/setup?hl=en
Website	Where we bought the Type A Port	https://www.irishelectronics.ie/product/USB-Female-Type-A-Port-4-Pin-DIP-90
Website	The Sabor Armory – components	https://thesaberarmory.com/products/usb-type-c-port-and-lithium-ion-battery-charging-kit?variant=39724539609124
Website	Research – not overcharge battery	https://emariete.com/en/co2-meter-with-battery-well-done/
Website	Research – how to power ESP32 from phone	https://forum.arduino.cc/t/how-to-power-esp32-from-a-mobile-phone/877904/4
Website	Pullup / pulldown resistors in ESP32	https://forum.arduino.cc/t/pull-up-or-pull-down-resistors-on-esp32/921334/3 https://forum.arduino.cc/t/esp32-pins-that-support-pullup/1173356/4
Website	Data logging into Firebase	https://randomnerdtutorials.com/esp32-data-logging-firebase-realtime-

	Realtime Database	database/#:~:text=Go%20to%20Sketch%20%3E%20Include%20Library,to%20interact%20with%20the%20database.
Website	Firebase - Read and write data in web	https://firebase.google.com/docs/database/web/read-and-write?hl=en#web-modular-api_1
Website	LCD display	https://www.makerguides.com/interfacing-esp32-and-16x2-lcd-parallel-data-without-i2c/

Appendix 1: Code – Arduino IDE

```
// ----------------------------------------------------------------------------
// ----------------------------- LIBRARIES ------------------------------------
// ----------------------------------------------------------------------------
#include <WiFi.h> //For the WiFi Connection
#include <Firebase_ESP_Client.h> // For the Firebase Connection
#include "LiquidCrystal.h" // For the LCD Display Screen
#include "addons/TokenHelper.h" // Provide the token generation process info.


// ----------------------------------------------------------------------------
// ---------------------------- DEFINITIONS -----------------------------------
// ----------------------------------------------------------------------------
// Insert your network credentials
#define WIFI_SSID "name_of_the_WiFi"// change this to the WiFi name
#define WIFI_PASSWORD "password123" // change to the WiFi password

// Insert Firebase project API Key (Change to your Project API)
#define API_KEY "AIzaSyC93r2rZB_YIKmp-rZhc5_RtfK8ocolKhQ"

// Insert Authorized Email and Corresponding Password
#define USER_EMAIL "example123456789@gmail.com"
#define USER_PASSWORD "1234567890"

// Insert RTDB URL (Change to your Realtime Database)
#define DATABASE_URL "https://portable-charger-cb4b5-default-
rtdb.firebaseio.com/"

// ----------------------------------------------------------------------------
// ----------------------- OBJECTS AND VARIABLES ------------------------------
// ----------------------------------------------------------------------------
// Define Firebase objects
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;

// Variable to save USER UID
unsigned long sendDataPrevMillis = 0;
bool signupOK = false;

// Analog pin for sensor that will give us the battery level
const int pinSensorBat = 33;

// initialize the library with the numbers of the interface pins
// Create An LCD Object. Signals: [ RS, E, D4, D5, D6, D7 ]
LiquidCrystal lcd(22, 19, 17, 5, 21, 18);

//BUTTON
```

```
int pinPulse = 4;


// ---------------------------------------------------------------------------
// ---------------------------- INITIALIZE WIFI ------------------------------
// ---------------------------------------------------------------------------
void initWiFi() {
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to WiFi ..");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print('.');
    delay(1000);
  }
  Serial.println(WiFi.localIP());
  Serial.println();
}


// ---------------------------------------------------------------------------
// ------------------------------ SETUP --------------------------------------
// ---------------------------------------------------------------------------
void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  lcd.clear();
  Serial.begin(115200);
  // Determine that the pin from the button will be to receive
  pinMode(pinPulse, INPUT_PULLDOWN);

  pinMode(pinSensorBat, INPUT_PULLDOWN);

  Serial.begin(115200);
  // Initialize WiFi
  initWiFi();
  // Assign the api key (required)
  config.api_key = API_KEY;
  // Assign the user sign-in credentials
  auth.user.email = USER_EMAIL;
  auth.user.password = USER_PASSWORD;

  Firebase.reconnectWiFi(true);
  fbdo.setResponseSize(4096);

  // Assign the callback function for the long-running token generation task
  config.token_status_callback = tokenStatusCallback; // see
addons/TokenHelper.h

  // Assign the maximum retry of token generation
  config.max_token_generation_retry = 5;
```

```
  /* Assign the RTDB URL (required) */
  config.database_url = DATABASE_URL;
  // Initialize the library with the Firebase authen and config
  Firebase.begin(&config, &auth);

  // Getting the user UID might take a few seconds
  Serial.println("Getting User UID");
  while ((auth.token.uid) == "") {
    Serial.print('.');
    delay(1000);
  }
  signupOK = true;
}


// ------------------------------------------------------------------------
// ------------------------------- LOOP -----------------------------------
// ------------------------------------------------------------------------
void loop() {
  // Read analog sensor data
  int percentageValue = readBattery();
  // Write in the Database the Battery Level
  writeDB(percentageValue);

  Serial.println(percentageValue);
  delay(1000);

  // If the button is Clicked, show in Screen the Battery Level
  if (digitalRead(pinPulse) == HIGH) {
    writeInScreen(percentageValue);
  }
}


// ------------------------------------------------------------------------
// ------------------------- WRITE IN DATABASE ----------------------------
// ------------------------------------------------------------------------
void writeDB(int percentageValue){
  if (Firebase.ready() && signupOK && (millis() - sendDataPrevMillis > 15000
|| sendDataPrevMillis == 0)) {
    sendDataPrevMillis = millis();

    // Write the percentage value to the database path
    if (Firebase.RTDB.setInt(&fbdo, "Battery/Percentage", percentageValue)) {
      Serial.println("PASSED");
    } else {
      Serial.println("FAILED");
      Serial.println("REASON: " + fbdo.errorReason());
    }
```

```
  }
}

// ------------------------------------------------------------------------
// ----------------------------- READ BATTERY -----------------------------
// ------------------------------------------------------------------------
int readBattery(){
  int sensorValue = analogRead(pinSensorBat);

  // Map the sensor value to a percentage (0 to 100)
  int percentageValue = map(sensorValue, 0, 4095, 0, 100);
  return percentageValue;
}

// ------------------------------------------------------------------------
// ------------------- SHOW INFO IN LCD LED SCREEN  -----------------------
// ------------------------------------------------------------------------
void writeInScreen(int percentageValue) {
  // (note: line 1 is the second row, since counting begins with 0):
  String Value = (String)percentageValue;
  lcd.setCursor(0, 0);
  lcd.print("    Battery:    ");
  lcd.setCursor(0, 1);
  lcd.print("      "+Value+"%");

  Serial.println("Clicked, inside");
  delay(5000); // after 5 secs
  lcd.clear(); // apaga la pantalla simulando caracteres en blanco

  Serial.println("Screen turns off");
}
```

Appendix 2: Code – Web Page

HTML

```
<!DOCTYPE html>
<html>

<head>
    <link rel="stylesheet" href="web-projecto.css">
    <meta charset="UTF-8" />
    <title>Project 3</title>

    <script src="script.js" type="module"></script>
</head>

<body id="body" class="on">
    <nav>
        <header>
            <h1>Project 3</h1>
        </header>
        <ul>
            <li><a href="#">Home</a></li>
            <li><a href="#">Contact</a></li>
            <li><a href="#">About Us</a></li>
        </ul>
    </nav>

    <div>
        <Button id="off-btu">OFF</Button>
        <Button id="on-btu" style="display: none;">ON</Button>
        <Button id="create">create</Button>
        <Button id="get">get</Button>
        <Button id="destroy" onclick="destroyDivsInSection()">destroy</Button>
        <Button id="update">update</Button>
    </div>

    <script src="script.js"></script>
</body>
</html>
```

CSS

```
nav {background-color: #00eeff; padding: 10px 0;}
ul {list-style-type: none; margin: 0; padding: 0; text-align: left;}
li {display: inline; margin-right: 20px;}
li:last-child { margin-right: 0; }
```

```
a {color: rgb(0, 0, 0); text-decoration: none; font-size: 18px;}
a:hover {color: yellow;}
.inline {display: inline-block; margin:10px;}
.on {margin: 0; height: 120vh; background: radial-gradient(at center, yellow,
green); font-size: 350%; border-radius: 50%;}
.off{margin: 0; height: 120vh; background: black; font-size: 350%; border-
radius: 50%;}
.batery-off{background-color: aqua;}
.hide {display: none;}

section div{ display: inline; margin-right: 10px;}

#off-btu{background-color: red;}
#on-btu{background-color: blue;}
```

JAVASCRIPT (JS)

```
// Import the functions you need from the SDKs you need
import { initializeApp } from
"https://www.gstatic.com/firebasejs/10.8.1/firebase-app.js";
import { getAnalytics } from
"https://www.gstatic.com/firebasejs/10.8.1/firebase-analytics.js";
import { getDatabase, set, get, update, remove, ref, child, onValue } from
"https://www.gstatic.com/firebasejs/10.8.1/firebase-database.js";

// Your web app's Firebase configuration
const firebaseConfig = {
    apiKey: "AIzaSyAhHTMKTnRq53rNlmOZhwURFtoUtv0U9do",
    authDomain: "portable-charger-cb4b5.firebaseapp.com",
    databaseURL: "https://portable-charger-cb4b5-default-rtdb.firebaseio.com",
    projectId: "portable-charger-cb4b5",
    storageBucket: "portable-charger-cb4b5.appspot.com",
    messagingSenderId: "1000373813112",
    appId: "1:1000373813112:web:3aaa62c4273232dee9928d",
    measurementId: "G-FTF0KZW7QX"
};
var on = true;
// Initialize Firebase
const app = initializeApp(firebaseConfig);
const analytics = getAnalytics(app);
const database = getDatabase();
var data = null;

function getPeriodico() {
    const dataRef = ref(database, '/');

    // Retrieve data once
    get(dataRef).then((snapshot) => {
```

```
        data = snapshot.val();
        //console.log(data);
    }).catch((error) => {
        console.error("Error retrieving data:", error);
    });

    // Listen for new additions
    onValue(dataRef, (snapshot) => {
        //addDevice();
        data = snapshot.val();
        console.log(data.Battery);
        destroySectionContent();
        addDevice();
    });
}
function addDevice() {
    // Create a new div element
    const newDiv = document.createElement('div');
    newDiv.id = 'battery'; // Assign the id "battery" to newDiv


    // Get the section in HTML where you want to insert the div
    const section = document.getElementById('divSection');

    // Create and set attributes for the image element
    const newImage = document.createElement('img');
    newImage.src = 'batería.png'; // Set the image source
    newImage.alt = 'Battery Image'; // Set the alt text for accessibility

    // Set width and height for the image
    newImage.style.width = '150px';
    newImage.style.height = '150px';

    // Create a text paragraph element
    const newText = document.createElement('p');
    newText.textContent = data.Battery.Percentage + '%'; // Set text content

    // Append the image and text elements to the new div
    newDiv.appendChild(newImage);
    newDiv.appendChild(newText);

    // Insert the new div into the section
    section.appendChild(newDiv);

    console.log("Start");
    Object.keys(data.Battery).forEach(key => {
        // Check if the key starts with "Device"
        if (key.startsWith('Device')) {
            const deviceInfo = data.Battery[key];
```

```
            const deviceDiv = createDeviceDiv(deviceInfo);
            section.appendChild(deviceDiv);
            console.log(`Device Name: ${deviceInfo.Name}`);
            console.log(`Percentage: ${deviceInfo.Percentage}`);
            console.log('------------------------');
        }
    });
}


function createDeviceDiv(deviceInfo) {
    const deviceDiv = document.createElement('div');
    deviceDiv.classList.add('device-container');
    const newImage = document.createElement('img');

    newImage.style.width = '150px';
    newImage.style.height = '150px';
    newImage.src = 'movil.png'; // Set the image source
    newImage.alt = 'Mobile Image'; // Set the alt text for accessibility

    deviceDiv.appendChild(newImage);

    const percentagePara = document.createElement('p');
    percentagePara.textContent = `${deviceInfo.Percentage} %`;
    deviceDiv.appendChild(percentagePara);

    return deviceDiv;
}


function destroySectionContent() {
    // Get the section element by its id
    var section = document.getElementById("divSection");
    // Check if the section exists
    if (section) {
      // Remove all child elements of the section
      while (section.firstChild) {
        section.removeChild(section.firstChild);
      }
    } else {
      console.error('Section with id ' + sectionId + ' not found.');
    }
}

function destroyNonBatteryDivs() {
  console.log("Adios");
  // Get the section element by its id
  var section = document.getElementById("divSection");
```

```
  // Check if the section exists
  if (section) {
    // Get all div elements inside the section
    var divs = section.querySelectorAll('div');

    // Loop through each div element
    divs.forEach(function(div) {
      // Check if the div does not have the id "battery"
      if (div.id !== 'battery') {
        // Remove the div element
        div.remove();
      }
    });
  } else {
    console.error('Section with id ' + sectionId + ' not found.');
    }
}

document.getElementById("body").onload = getPeriodico();
document.getElementById("off-btu").onclick = function()
{
  on = false;
  destroyNonBatteryDivs(); // Reemplaza 'mySection' con el ID de tu sección
  document.getElementById("off-btu").style.display = "none";
  document.getElementById("on-btu").style.display = "inline";
};

document.getElementById("on-btu").onclick = function()
{
  on = true;
  getPeriodico(); // Reemplaza 'mySection' con el ID de tu sección
  document.getElementById("on-btu").style.display = "none";
  document.getElementById("off-btu").style.display = "inline";
};
document.getElementById("destroy").onclick = destroySectionContent();
```

Appendix 3: Code – Mobile App

```
package com.example.smartbattery;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.graphics.Color;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.ImageButton;
import android.widget.ImageView;
import android.widget.Switch;
import android.widget.TextView;
import android.widget.Toast;

import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

import java.util.Random;


public class JustOne extends AppCompatActivity {
    private TextView textCharger, textName1, textName2, textBattery1,
textBattery2;
    private ImageView imageCharger, imageBattery1, imageBattery2;
    private Switch dark;

    ///////////DB
    private DatabaseReference mDatabase;
    ///////////DB

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_just_one);

        imageCharger = (ImageView) findViewById(R.id.imageCharger);
        textCharger = (TextView) findViewById(R.id.textCharger);

        imageBattery1 = (ImageView) findViewById(R.id.imageBattery1);
        textName1 = (TextView) findViewById(R.id.textName1);
        textBattery1 = (TextView) findViewById(R.id.textBattery1);

        imageBattery2 = (ImageView) findViewById(R.id.imageBattery2);
        textBattery2 = (TextView) findViewById(R.id.textBattery2);
        textName2 = (TextView) findViewById(R.id.textName2);

        ///////////DB
        mDatabase = FirebaseDatabase.getInstance().getReference();
        mDatabase.child("Battery").addValueEventListener(new
ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                if(snapshot.exists()){
                    String name1="", name2="";
                    int bat1=0, bat2=0;
```

```
                    int charge =
Integer.parseInt(snapshot.child("Percentage").getValue().toString());

if(!snapshot.child("Device1").child("Name").getValue().toString().equals(""
)) {
                        name1 =
snapshot.child("Device1").child("Name").getValue().toString();
                        bat1 =
Integer.parseInt(snapshot.child("Device1").child("Percentage").getValue().t
oString());
                    }

if(!snapshot.child("Device2").child("Name").getValue().toString().equals(""
)) {
                        name2 =
snapshot.child("Device2").child("Name").getValue().toString();
                        bat2 =
Integer.parseInt(snapshot.child("Device2").child("Percentage").getValue().t
oString());
                    }
                    charge(charge, bat1, bat2, name1, name2);
                }
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {
                Toast toast1 =
Toast.makeText(JustOne.this,error.toString(), Toast.LENGTH_SHORT);
                toast1.show();
            }
        });
        dark = (Switch) findViewById(R.id.modo_oscuro);
        dark.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if(dark.isChecked()){

findViewById(R.id.all).setBackgroundColor(Color.parseColor("#A966B5"));
                }
                else{

findViewById(R.id.all).setBackgroundColor(Color.parseColor("#FFFFFF"));
                }
            }
        });
    }

    public void readDb(){

    }

    public void charge(int charger, int battery1, int battery2, String
name1, String name2){
        if(battery1!=-1){
            imageBattery1.setVisibility(View.VISIBLE);
            if(battery1>=0&&battery1<=25)

imageBattery1.setImageResource(R.mipmap.no_charge_foreground);
            else if(battery1>25&&battery1<=50)

imageBattery1.setImageResource(R.mipmap.half_charge_foreground);
```

```
            else if(battery1>50&&battery1<=75)

imageBattery1.setImageResource(R.mipmap.morehalf_charge_foreground);
            else

imageBattery1.setImageResource(R.mipmap.full_charge_foreground);
            textName1.setVisibility(View.VISIBLE);
            textName1.setText(name1);
            textBattery1.setVisibility(View.VISIBLE);
            textBattery1.setText(Integer.toString(battery1)+"%");
        }
        else{
            imageBattery1.setVisibility(View.INVISIBLE);
            textName1.setVisibility(View.INVISIBLE);
            textBattery1.setVisibility(View.INVISIBLE);
        }
        if(battery2!=-1){
            imageBattery2.setVisibility(View.VISIBLE);
            if(battery2>=0&&battery2<=25)

imageBattery2.setImageResource(R.mipmap.no_charge_foreground);
            else if(battery2>25&&battery2<=50)

imageBattery2.setImageResource(R.mipmap.half_charge_foreground);
            else if(battery2>50&&battery2<=75)

imageBattery2.setImageResource(R.mipmap.morehalf_charge_foreground);
            else

imageBattery2.setImageResource(R.mipmap.full_charge_foreground);
            textName2.setVisibility(View.VISIBLE);
            textName2.setText(name2);
            textBattery2.setVisibility(View.VISIBLE);
            textBattery2.setText(Integer.toString(battery2)+"%");
        }
        else{
            imageBattery2.setVisibility(View.INVISIBLE);
            textName2.setVisibility(View.INVISIBLE);
            textBattery2.setVisibility(View.INVISIBLE);
        }

        if(charger>=0&&charger<=25)
            imageCharger.setImageResource(R.mipmap.no_charge_foreground);
        else if(charger>25&&charger<=50)
            imageCharger.setImageResource(R.mipmap.half_charge_foreground);
        else if(charger>50&&charger<=75)

imageCharger.setImageResource(R.mipmap.morehalf_charge_foreground);
        else
            imageCharger.setImageResource(R.mipmap.full_charge_foreground);
        textCharger.setText(Integer.toString(charger)+"%");

    }
}
```

```
package com.example.smartbattery;

import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;

import android.app.Activity;
import android.app.Application;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.net.Uri;
import android.os.Build;
import android.os.Bundle;
import android.util.Log;
import android.util.SparseArray;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.webkit.URLUtil;
import android.widget.Toast;

import com.google.android.gms.vision.CameraSource;
import com.google.android.gms.vision.Detector;
import com.google.android.gms.vision.barcode.Barcode;
import com.google.android.gms.vision.barcode.BarcodeDetector;

import java.io.IOException;

public class MainActivity extends AppCompatActivity {

    private CameraSource cameraSource;
    private SurfaceView cameraView;
    private final int MY_PERMISSIONS_REQUEST_CAMERA = 1;
    private String token = "";
    private String tokenanterior = "";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Intent i = new Intent(MainActivity.this,JustOne.class);
        startActivity(i);

        cameraView = (SurfaceView) findViewById(R.id.camera_view);
        initQR();
    }

    public void initQR() {

        // creo el detector qr
        BarcodeDetector barcodeDetector =
                new BarcodeDetector.Builder(this)
                        .setBarcodeFormats(Barcode.ALL_FORMATS)
                        .build();

        // creo la camara
        cameraSource = new CameraSource
                .Builder(this, barcodeDetector)
                .setRequestedPreviewSize(1600, 1024)
                .setAutoFocusEnabled(true) //you should add this feature
                .build();
```

```
        // listener de ciclo de vida de la camara
        cameraView.getHolder().addCallback(new SurfaceHolder.Callback() {
            @Override
            public void surfaceCreated(SurfaceHolder holder) {

                // verifico si el usuario dio los permisos para la camara
                if (ActivityCompat.checkSelfPermission(MainActivity.this,
android.Manifest.permission.CAMERA)
                        != PackageManager.PERMISSION_GRANTED) {

                    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
                        // verificamos la version de ANdroid que sea al
menos la M para mostrar
                        // el dialog de la solicitud de la camara
                        if (shouldShowRequestPermissionRationale(
                                android.Manifest.permission.CAMERA)) ;
                        requestPermissions(new
String[]{android.Manifest.permission.CAMERA},
                                MY_PERMISSIONS_REQUEST_CAMERA);
                    }
                    return;
                } else {
                    try {
                        cameraSource.start(cameraView.getHolder());
                    } catch (IOException ie) {
                        Log.e("CAMERA SOURCE", ie.getMessage());
                    }
                }
            }

            @Override
            public void surfaceChanged(SurfaceHolder holder, int format,
int width, int height) {
            }

            @Override
            public void surfaceDestroyed(SurfaceHolder holder) {
                cameraSource.stop();
            }
        });

        // preparo el detector de QR
        barcodeDetector.setProcessor(new Detector.Processor<Barcode>() {
            @Override
            public void release() {
            }


            @Override
            public void receiveDetections(Detector.Detections<Barcode>
detections) {
                final SparseArray<Barcode> barcodes =
detections.getDetectedItems();

                if (barcodes.size() > 0) {

                    // obtenemos el token
                    token = barcodes.valueAt(0).displayValue.toString();

                    // verificamos que el token anterior no se igual al
```

```
actual
                    // esto es util para evitar multiples llamadas
empleando el mismo token
                    if (!token.equals(tokenanterior)) {

                        // guardamos el ultimo token proceado
                        tokenanterior = token;
                        Log.i("token", token);

                        Intent i = new
Intent(MainActivity.this,JustOne.class);
                        startActivity(i);

                        /*if (URLUtil.isValidUrl(token)) {
                            // si es una URL valida abre el navegador
                            Intent browserIntent = new
Intent(Intent.ACTION_VIEW, Uri.parse(token));
                            startActivity(browserIntent);
                        } else {
                            // comparte en otras apps
                            Intent shareIntent = new Intent();
                            shareIntent.setAction(Intent.ACTION_SEND);
                            shareIntent.putExtra(Intent.EXTRA_TEXT, token);
                            shareIntent.setType("text/plain");
                            startActivity(shareIntent);
                        }*/

                        new Thread(new Runnable() {
                            public void run() {
                                try {
                                    synchronized (this) {
                                        wait(5000);
                                        // limpiamos el token
                                        tokenanterior = "";
                                    }
                                } catch (InterruptedException e) {
                                    // TODO Auto-generated catch block
                                    Log.e("Error", "Waiting didnt work!!");
                                    e.printStackTrace();
                                }
                            }
                        }).start();

                    }
                }
            }
        });

    }
}
```