FIREBASE WEB APP
with ESP32 and ESP8266

Build a Firebase Web App to Control Outputs and Monitor Sensors from Anywhere (Includes Login/Logout Authentication)

Rui Santos and Sara Santos

Firebase Web App with ESP32/ESP8266

Security Notice

Sorry for writing this notice, but the evidence is clear: piracy for digital products is over the entire internet.

For that reason, we've taken certain steps to protect our intellectual property contained in this eBook.

This eBook contains hidden random strings of text that only apply to your specific eBook version that is unique to your email address. You won't see anything different since those strings are hidden in this PDF. We apologize for having to do that. It means if someone were to share this eBook, we know who did it, and we can take further legal consequences.

You cannot redistribute this eBook. This eBook is for personal use and is only available for purchase at:

- https://randomnerdtutorials.com/courses
- https://rntlab.com/shop

Please send an email to the author (Rui Santos - hello@ruisantos.me) if you find this eBook anywhere else.

We want to thank you for purchasing this eBook, and we hope you learn a lot and have fun with it!

Disclaimer

This eBook was written for information purposes only. Every effort was made to make this eBook as complete and accurate as possible. The purpose of this eBook is to educate. The authors (Rui Santos and Sara Santos) do not warrant that the information contained in this eBook is fully complete and shall not be responsible for any errors or omissions.

The authors (Rui Santos and Sara Santos) shall have neither liability nor responsibility to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by this eBook.

Throughout this eBook, you will find some links, and some of them are affiliate links. This means the authors (Rui Santos and Sara Santos) earn a small commission from each purchase with that link. Please understand that the authors have experience with all those products and recommend them because they are useful, not because of the small commissions. Please do not spend any money on products unless you feel you need them.

Other Helpful Links:

- Ask questions in our Forum
- Join Private Facebook Group
- Terms and Conditions

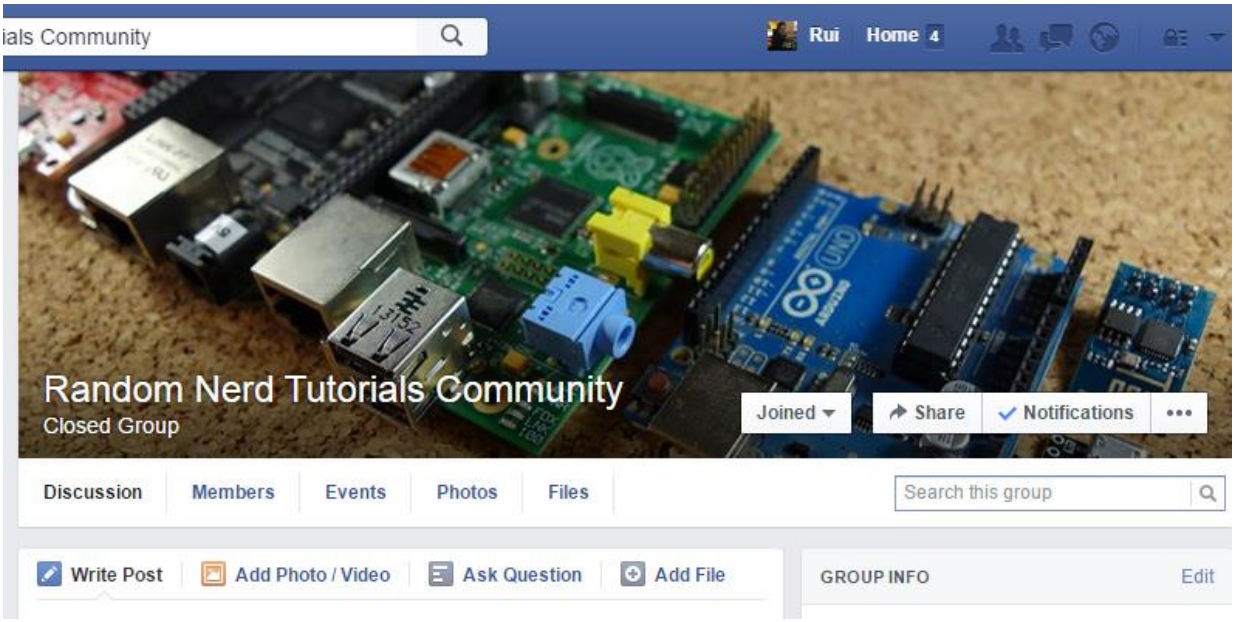
Join the Private Facebook Group

This eBook comes with the opportunity to join a private community of like-minded people. If you purchased this eBook, you can join our private Facebook Group today!

Inside the group, you can ask questions and create discussions about everything related to ESP32, ESP8266, Arduino, Raspberry Pi, etc.

See it for yourself!

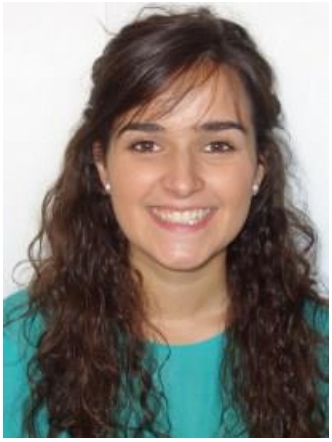
- Step #1: Go to -> http://randomnerdtutorials.com/fb
- Step #2: Click "Join Group" button
- Step #3: We'll approve your request within less than 24 hours.



About the Authors

This eBook was developed and written by Rui Santos and Sara Santos. We both live in Porto, Portugal, and we have known each other since 2009. If you want to learn more about us, feel free to read our about page.

Hi! I'm Rui Santos, the founder of the Random Nerd Tutorials blog. I have a master's degree in Electrical and Computer Engineering from FEUP. I'm the author of the books: "BeagleBone For Dummies "and "20 Easy Raspberry Pi Projects: Toys, Tools, Gadgets, and More!". I was the Technical reviewer of the "Raspberry Pi For Kids For Dummies" book. I have worked at the RNT since 2013.



Hi! I'm Sara Santos, and I work with Rui at Random Nerd Tutorials. I have a master's degree in Bioengineering from FEUP, and I'm the co-author of the "20 Easy Raspberry Pi Projects" book. I create, write and edit the tutorials and articles for the RNT and Maker Advisor blogs and help you with your concerns wherever I can. I love books, writing, cats, and a hot cup of tea.

Table of Contents

PART 0

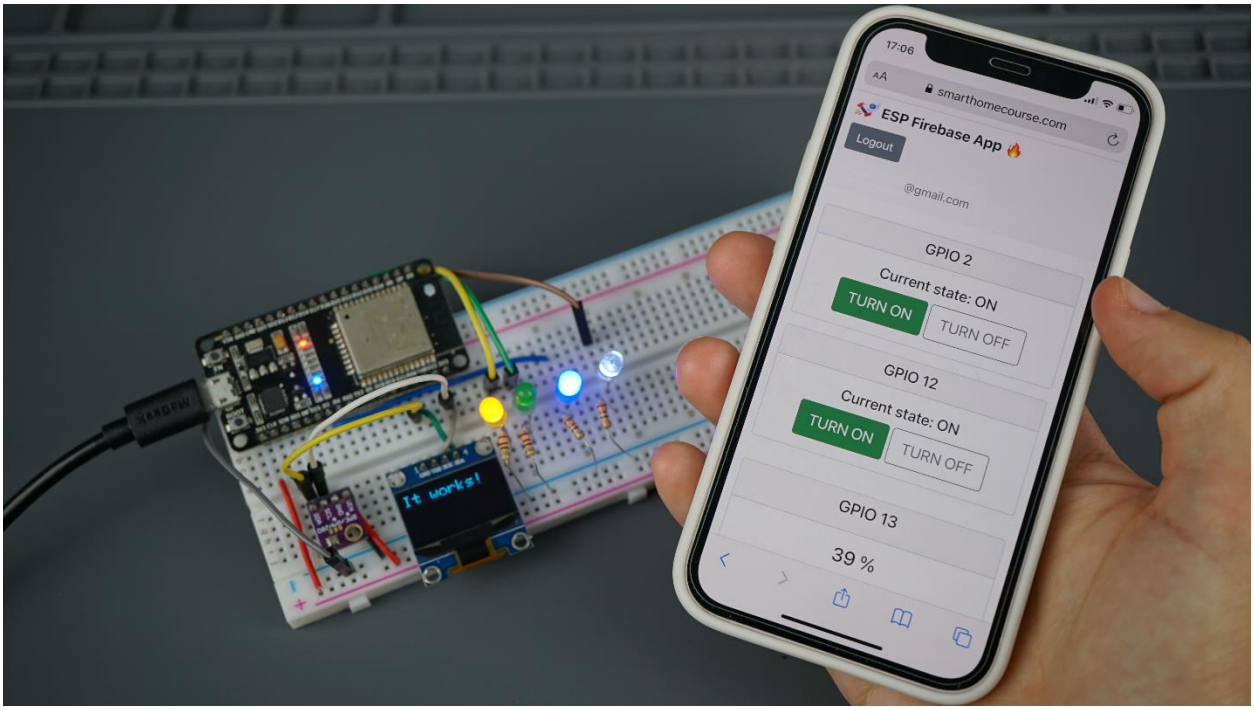
Getting Started

Part 0 introduces the project. It covers what you'll learn, how to follow this project and some recommended prerequisites.

We'll also look at the project's main features and give you a quick introduction to Firebase.

Welcome to "Firebase Web App with the ESP32/ESP8266"

Welcome to the "**Firebase Web App with the ESP32 and ESP8266**" eBook.



Throughout this eBook, you'll build a Firebase web application that can monitor and control your ESP32 and ESP8266 boards from anywhere in the world. The application allows you to control the ESP GPIOs using buttons, sliders and send input data. It also displays sensor readings sent by the ESP32 or ESP8266 boards. All the data is saved on the Firebase Realtime Database. The web application is protected with login using email and password, and your database is protected using database rules.

Is Firebase Free?

Yes, for our application, we'll use the free Spark Pricing Plan. The Firebase services we'll use to build and host the web app are free.

How to Follow this eBook?

The instructions to build the web application are step-by-step style. So, you must follow this eBook linearly. This means you need to follow all the mentioned steps, and you shouldn't skip sections unless otherwise noted.

Download Source Code and Resources

Each section contains the code and resources you need to complete each step and proceed to the following sections. You can download all the resources for a specific example on the corresponding Unit.

Alternatively, you can download the project repository and instantly download all the resources.

- **Download All eBook Resources »**

  (*https://github.com/RuiSantosdotme/Firebase-ESP32-ESP8266-eBook/archive/refs/heads/main.zip*)

eBook Overview

This eBook is divided into five parts:

- **Part 1**: **Creating a Firebase Project**—you'll learn how to create a Firebase project and set up all the necessary services (Authentication and Firebase Realtime Database).
- **Part 2: Organizing your Database and Database Rules**—we'll show how to organize your database and set up database rules to protect your data.
- **Part 3: ESP32/ESP8266: Interacting with the Realtime Database**—interact with the Firebase Realtime Database using your ESP32 and ESP8266 boards: learn how to read and send data to the database.

- **Part 4**: **Creating the Firebase Web App**—set up HTML and JavaScript files to create the web app to control and monitor your boards.
- **Part 5**: **Hosting your Web App (Custom Domain Name)**—host your web app and set up a custom domain name to access your web app anywhere.

What You'll Learn

By following this project, you'll learn about the following subjects:

- **Firebase Project**:
    - Create a Firebase project;
    - Add authentication to your Firebase project (email and password);
    - Add a Realtime Database (RTDB) to your project to save data in JSON format;
    - Organize your RTDB;
    - Protect the RTDB using database rules;
    - Add a web app to your Firebase project to control and monitor your ESP32 and ESP8266 boards;
    - Host your web app in Firebase servers;
    - Add a custom domain to your web app— this requires that you buy a domain name (this step is optional).
- **ESP32/ESP8266**:
    - Authenticate the ESP32 or ESP8266 board as an authorized user with email and password;
    - Write data to the Realtime Database;
    - Stream database—detect any database changes;
    - Run tasks depending on the detected changes.
- **Firebase Web App**:
    - Create a web app and connect it to your Firebase project;

- Build the web page for your app using basic HTML and bootstrap (CSS framework): login/logout modals, buttons, sliders, table for sensor readings, and input fields;
- Listen for database changes to update the web app data using JavaScript;
- Write to the database to control the ESP32 or ESP8266 outputs using JavaScript.

Prerequisites (recommended but not mandatory)

To get a better experience following along with this project, we recommend having some previous knowledge about the following subjects:

Basic programming of the ESP32 and/or ESP8266 with the Arduino Core, like controlling outputs and reading sensors. To get familiar with these boards, you can follow our free tutorials or our premium courses:

- Free ESP32 Tutorials
- Free ESP8266 Tutorials
- Learn ESP32 with Arduino IDE (eBook + Video course)
- Home Automation using ESP8266 (eBook + videos)

Being familiar with VS Code editor and VS Code with the PlatformIO extension. You can follow the next tutorials to get started with this IDE quickly:

- Getting Started with VS Code and PlatformIO IDE for ESP32 and ESP8266
- VS Code Workspaces with ESP32 and ESP8266 Projects

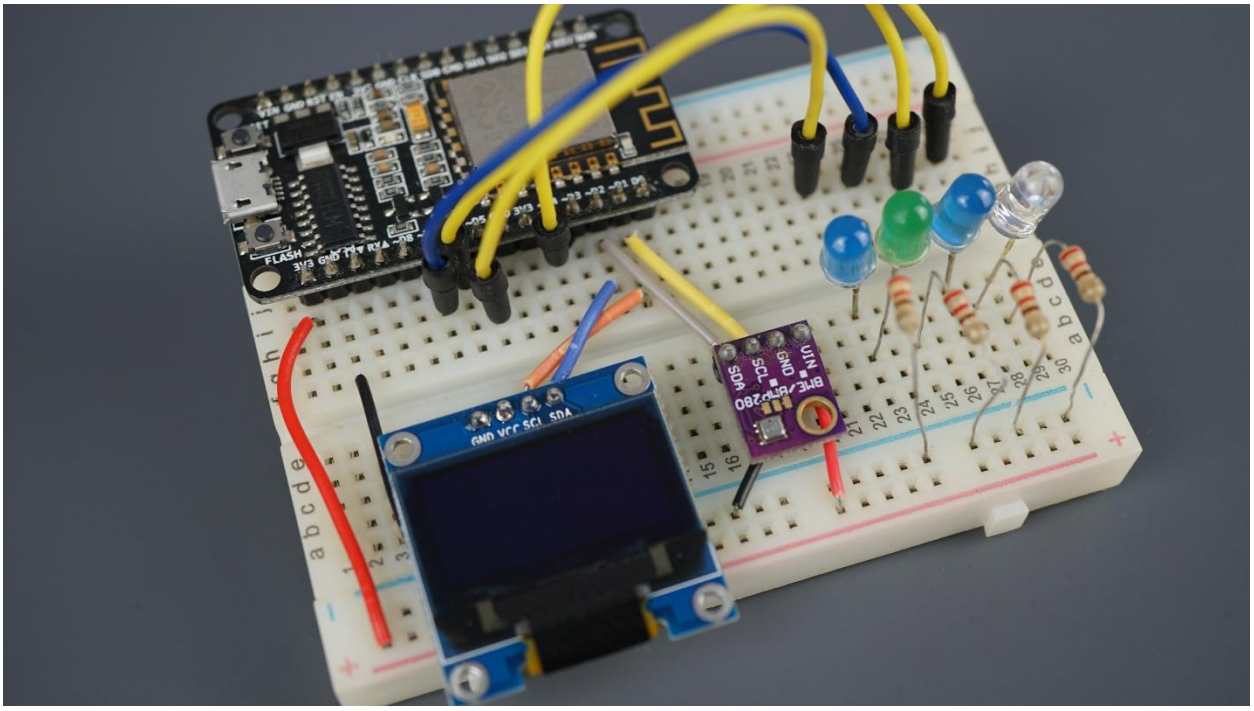
Basic knowledge of HTML, CSS, and JavaScript. We assume that you have basic knowledge about HTML (basic HTML tags), CSS, and JavaScript (variables, functions, and events). A good place to get started quickly with the basics is the W3schools

website. We also cover these topics in our Build Web Servers with the ESP32 and ESP8266 eBook.

If you don't have previous knowledge about these topics, you can still follow along as we provide step-by-step instructions. However, it might be more challenging to understand some steps and modify the project for your specific needs. Nonetheless, we're sure you'll be able to get the app working.

Parts Required

To follow this project, you only need a few electronics components. Here's the list:

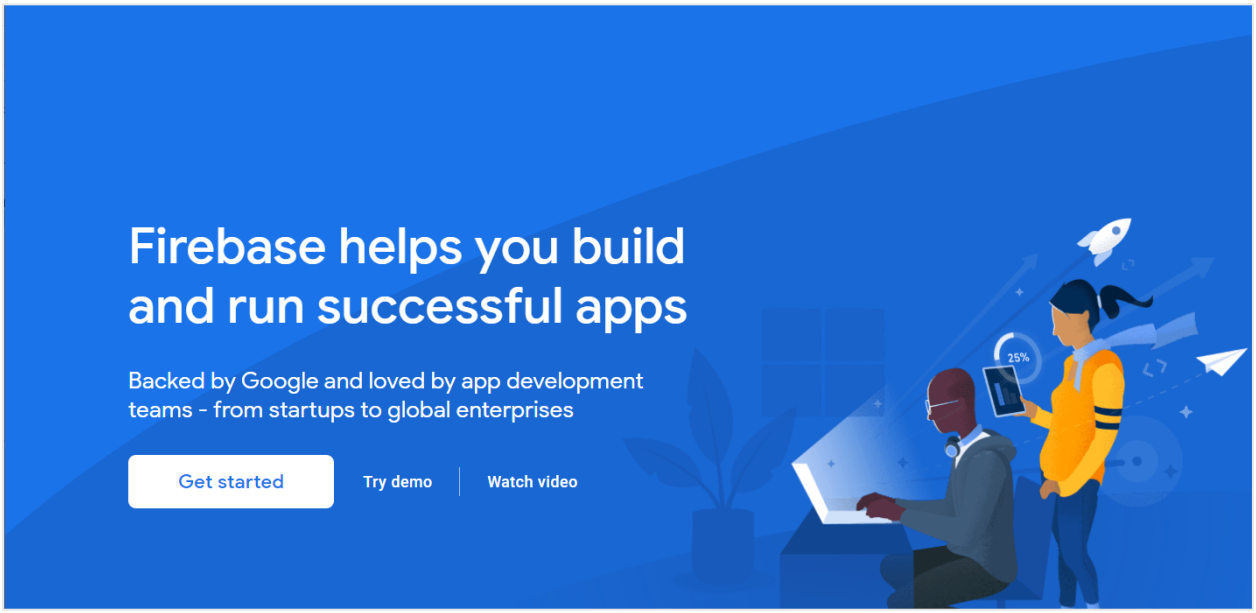


- ESP32 or ESP8266 board;
- BME280 or any other sensor you're familiar with;
- 4x LEDs;
- 4x 220 Ohm resistors;
- SSD1306 0.96inch OLED display;
- Breadboard;
- Jumper wires.

If you click the previous links, they will redirect you to the Maker Advisor website (it's also our website). If you buy your parts through Maker Advisor links, we'll earn a small affiliate commission (you won't pay more for it). By getting your parts through our affiliate links, you are supporting our work. If there's a component or tool you're looking for, we advise you to look at our favorite tools and parts here.

Introduction to Firebase

Firebase is Google's mobile application development platform that helps you build, improve, and grow your app. It has many services used to manage data from any android, IOS, or web application.



The following paragraph clearly explains the advantages of using Firebase:

> *"Firebase is a toolset to "build, improve, and grow your app", and the tools it gives you cover a large portion of the services that developers would normally have to build themselves but don't really want to build because they'd rather be focusing on the app experience itself. This includes things like analytics, authentication, databases, configuration, file storage, push messaging, and the list goes on. The services are hosted in the cloud and scale with little to no effort on the part of the developer."* [1]

[1] Source: https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0

Firebase Products to Build your App

Firebase is a set of tools to *build*, *improve* and *grow* your app. We'll just focus on the building process because we'll create a web app for personal use. If you want to create a web app for commercial use, Firebase offers tools like analytics, predictions, A/B testing, performance monitoring, etc., to improve and grow your app.

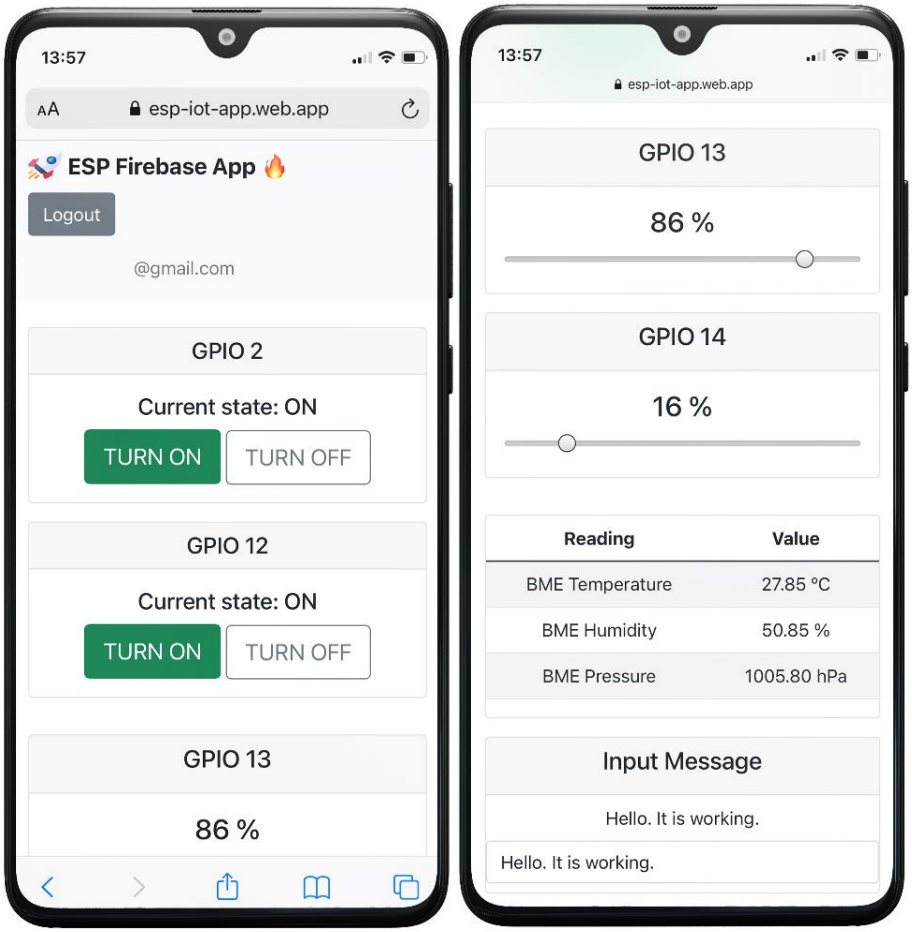
So, let's focus on the building process. Firebase provides the following products to build your app:

- **Authentication**: login/logout and user identity;
- **Realtime database**: realtime, cloud-hosted, NoSQL database; data is stored in a JSON structure;
- **Cloud Firestore**: realtime, cloud-hosted, NoSQL database; data is stored in "documents";
- **Cloud storage**: scalable file storage to upload and download files;
- **Cloud functions**: serverless, event-driven backend;
- **Firebase hosting**: global web hosting with or without custom domain;
- **ML kit**: SDK for common ML (machine learning) tasks.

Throughout our project, we'll use the following products: authentication, realtime database, cloud functions, and Firebase hosting.

Firebase Project Overview

Throughout this eBook, you'll build a web app with Firebase to interact with your ESP32 and/or ESP8266 boards—control outputs and display sensor readings from anywhere. Firebase makes it easy to create a web app to interact with your ESP32 and ESP8266 boards from anywhere in the world. It allows you to store all data in one place and synchronize it throughout all your devices.



Let's go through a quick project overview to understand all the project's features:

- The ESP32/ESP8266 can be controlled and monitored using the Firebase web app we'll build.

- The Firebase web app is protected with login using email and password. Only an authorized user can access the data. Additionally, your database is protected by database rules.
- The web app displays two buttons to control two GPIOs ON and OFF. The GPIO states are saved and updated on the Realtime Database. The ESP32 or ESP8266 detects the database changes and updates the outputs accordingly.
- Two sliders control two outputs with PWM (LEDs, motors, or other peripherals that require PWM). The slider values are saved and updated on the Realtime Database. The ESP32 or ESP8266 detects the database changes and updates the outputs PWM duty cycle accordingly.
- A table displays sensor readings. The web app gets the new readings from the database nodes that the ESP32 is publishing in. We'll save sensor readings from a BME280 sensor.
- Finally, there's also an input field to insert a message to display on an OLED. The message is also saved on the Realtime Database.
- You can access your firebase web app from anywhere in the world to interact with your ESP32/ESP8266 using a custom domain or the default domain name provided by Firebase.
- The web app is mobile responsive.

When it comes to the ESP32/ESP8266, here's the project's features:

- The ESP32/ESP8266 authenticates with username and password. That username and password must belong to an authorized user—otherwise, it won't be able to interact with the Realtime Database.
- The board publishes BME280 sensor readings—temperature, humidity, and pressure—periodically to the Realtime Database.

- The ESP32/ESP8266 listens to changes on the GPIOs states, slider values, and message values.
- Whenever there's a change, it updates the GPIO states, the PWM values, or the message displayed on the OLED.
- When the ESP reboots or starts, it automatically gathers all the information from the database and updates the GPIOs and OLED display accordingly.

Now that you know what this project will be all about, proceed to the following units to get started.

Recommended Reading

- Firebase terms: https://firebase.google.com/terms/
- Firebase billing: https://firebase.google.com/docs/projects/billing/firebase-pricing-plans

For our application, we'll use the **free Spark Pricing Plan**.

"When you're in the initial stages of developing your app, start out with the Spark pricing plan. You **don't need to provide any payment information to get started** using most Firebase features right away! [2]

[2] https://firebase.google.com/docs/projects/billing/firebase-pricing-plans#spark-pricing-plan

PART 1

Creating a Firebase Project

In this part, you'll learn how to create a Firebase project, set up the authentication methods, and create the Realtime Database.

1.1 - Creating a Firebase Project

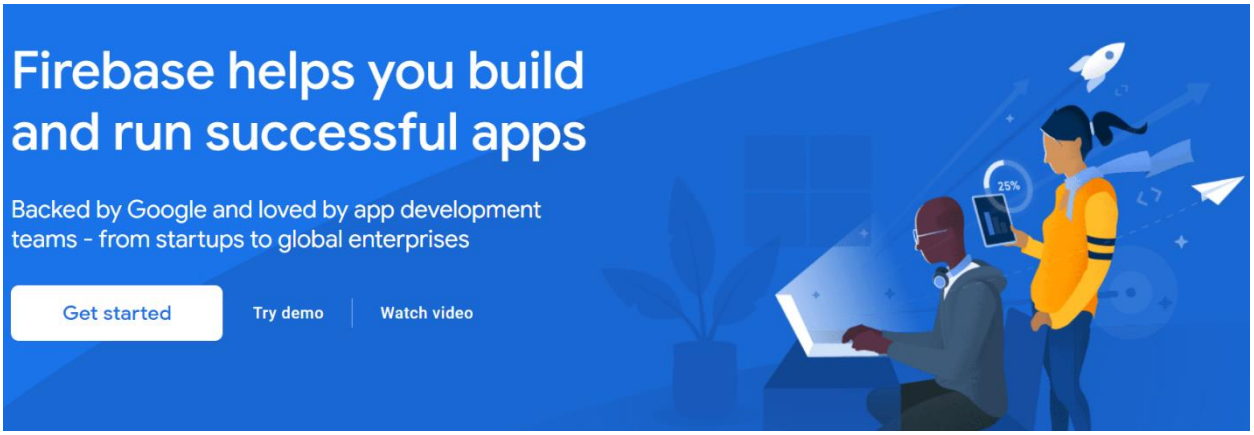
In this Unit, you'll learn how to create a Firebase project and enable the necessary products: authentication, realtime database, and hosting.

Make sure you follow all the described steps and don't miss any instructions.

1) Create a New Project

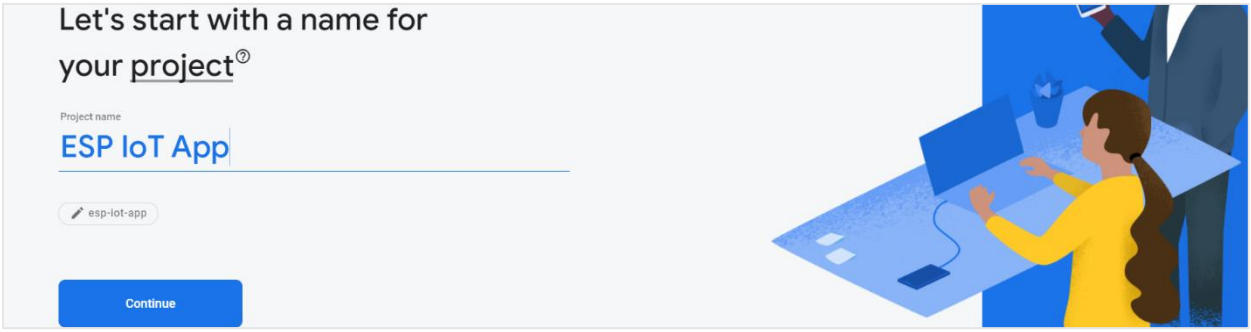
Let's start by creating a new Firebase project.

1) Go to Firebase: firebase.google.com, sign in using a Google Account and click **Get Started**.

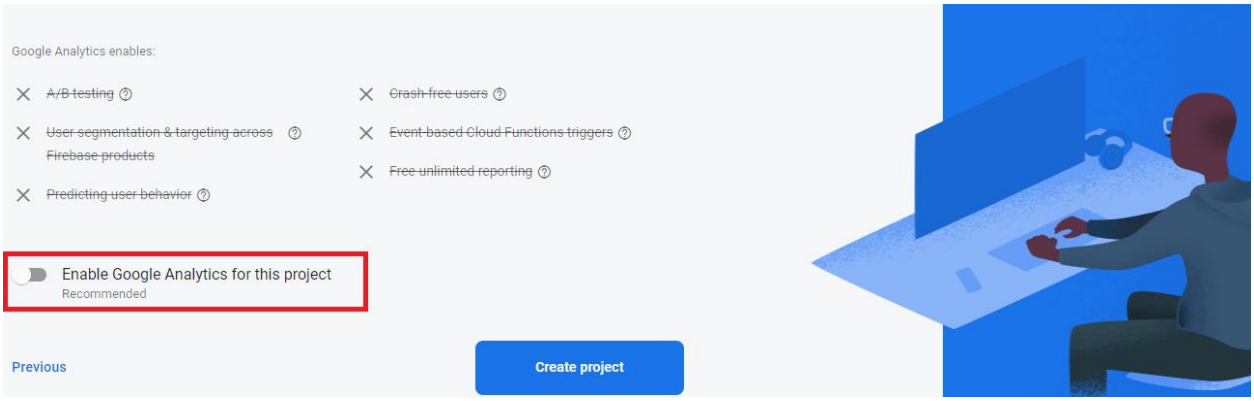


2) Click **Add project** to create a new project.

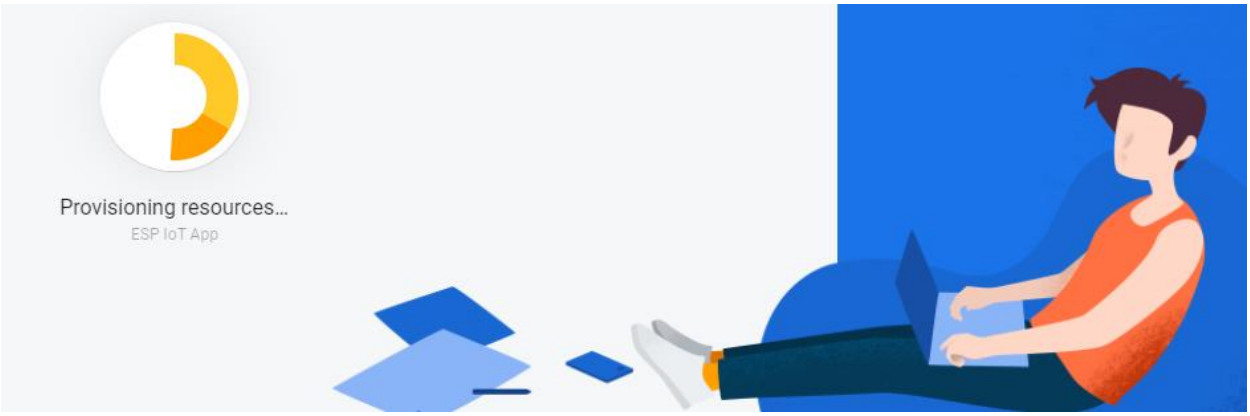
3) Give a name to your project, for example *ESP IoT App,* and press **Continue**.



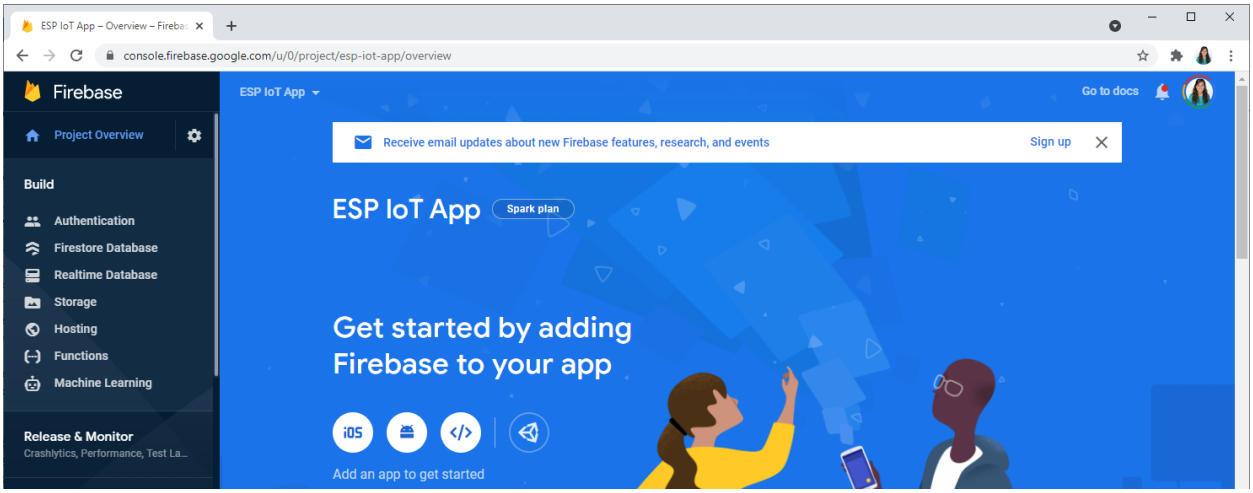
4) Disable the option *Enable Google Analytics for this project* as it is not needed and click **Create project**.



5) It will take a few seconds to set up your project. Click **Continue** when it's ready.



6) You'll be redirected to your Project console page.



Your project is now created! Let's enable and configure the necessary services.

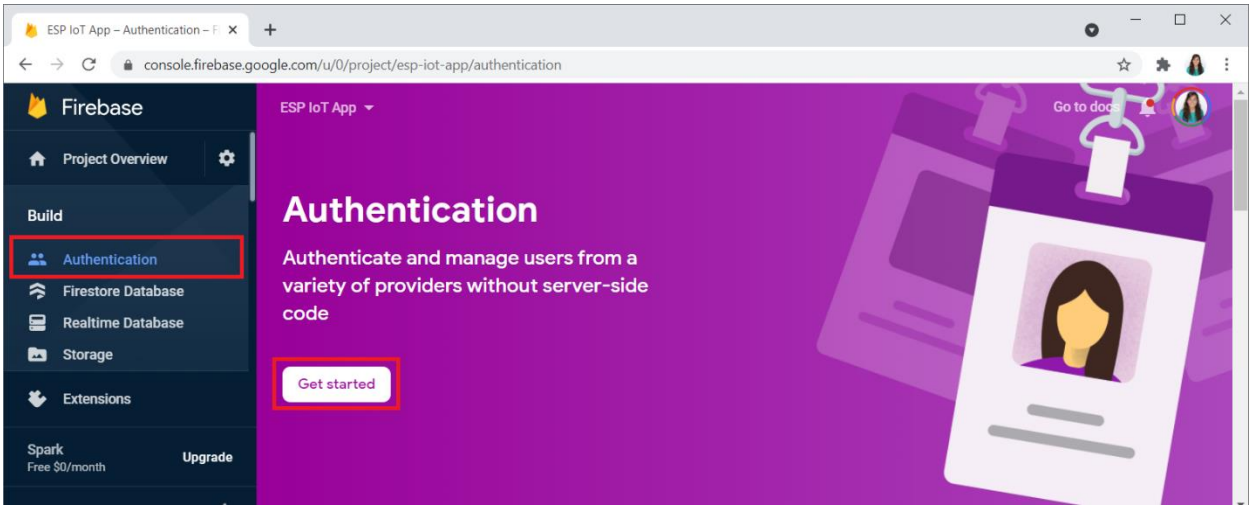
2) Set Authentication Methods

You need to set an authentication method for your app.

Most apps need to know the identity of a user. Knowing a user's identity allows an app to securely save user data in the cloud and provide the same personalized experience across all users' devices.

In other words, it takes care of logging in and identifying the users—like the ESP32/ESP8266 to access the database, or yourself as a user of the web app. It also allows us to set up database rules based on the logged-in user.

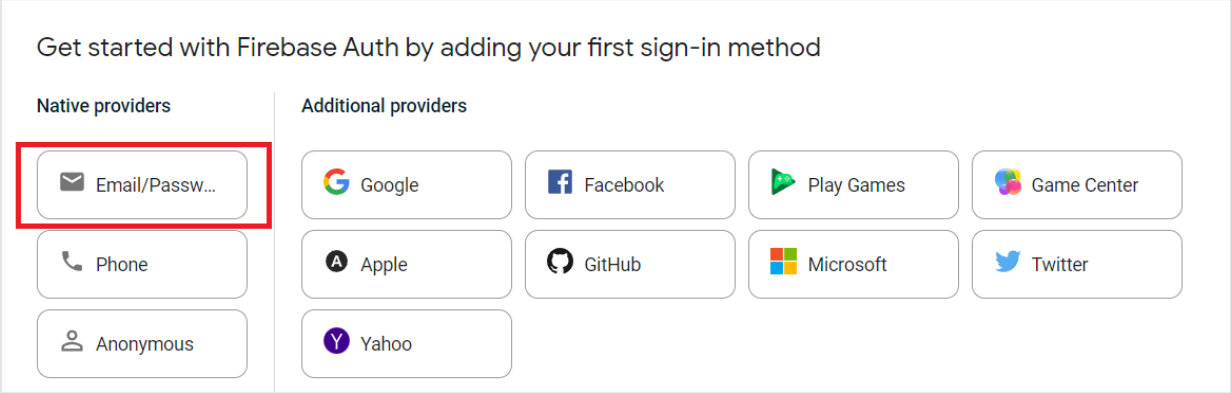
1) On the left sidebar, click on **Authentication** and then on **Get started**.



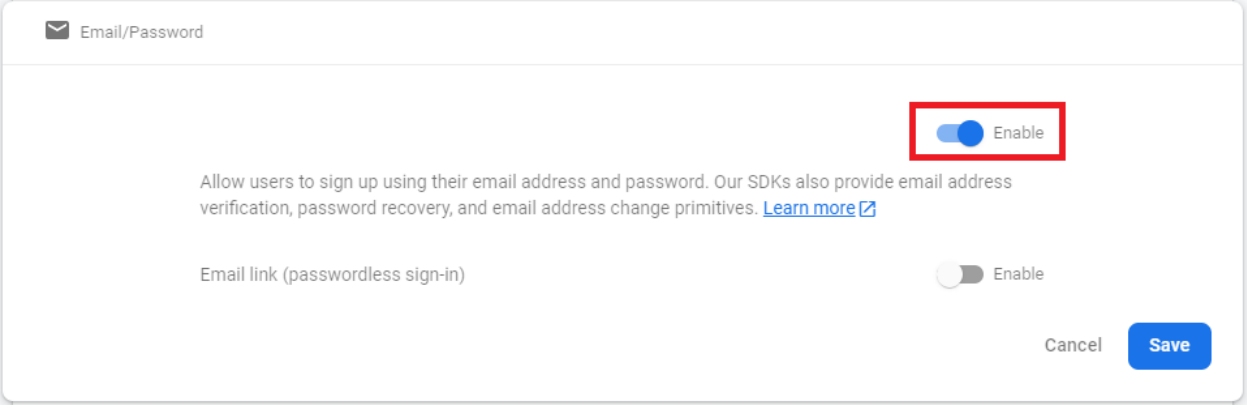
Firebase supports many different authentication methods like email and password, Google Account, Facebook account, anonymous user, etc.

In our project, we'll use email and password to authenticate. Then, only one authorized user that you define can access the app via a login form that we'll add to the app. That's also that user that we'll "emulate" on the ESP boards so that it can access the database.

2) Select the **Email/Password** option.

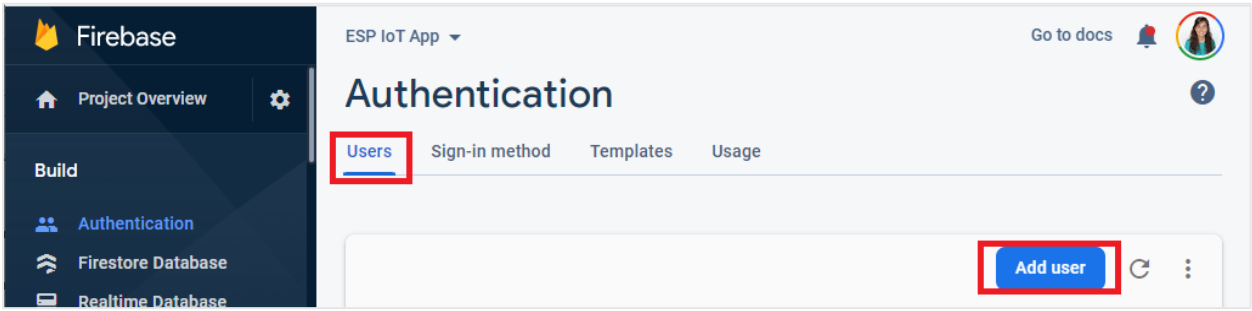


3) Enable the option to allow users to sign up using their email address and password. Then, click **Save**.

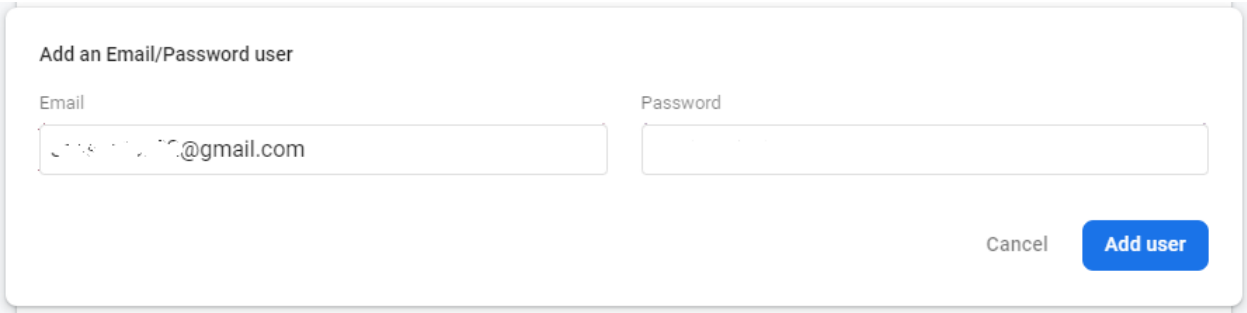


Now, you need to add an authorized user. Our app will only have one authorized user.

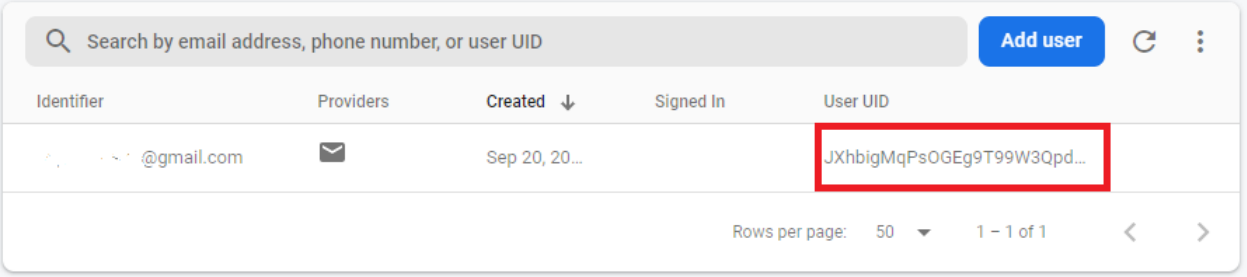
4) Still on the **Authentication** tab, click on the **Users** tab at the top. Then, click **Add user**.



5) Add an email address for the authorized user. It can be your google account email or any other email. You can also create an email for this specific project. Add a password that will allow you to sign in to your app and access the database. Don't forget to save the password in a safe place because you'll need it later. When you're done, click **Add user**.



6) A new user was successfully created and added to the **Users** table.



Notice that Firebase creates a unique UID for each registered user. The user UID allows us to identify the user and keep track of the user to provide or deny access to the database.

3) Create a Realtime Database

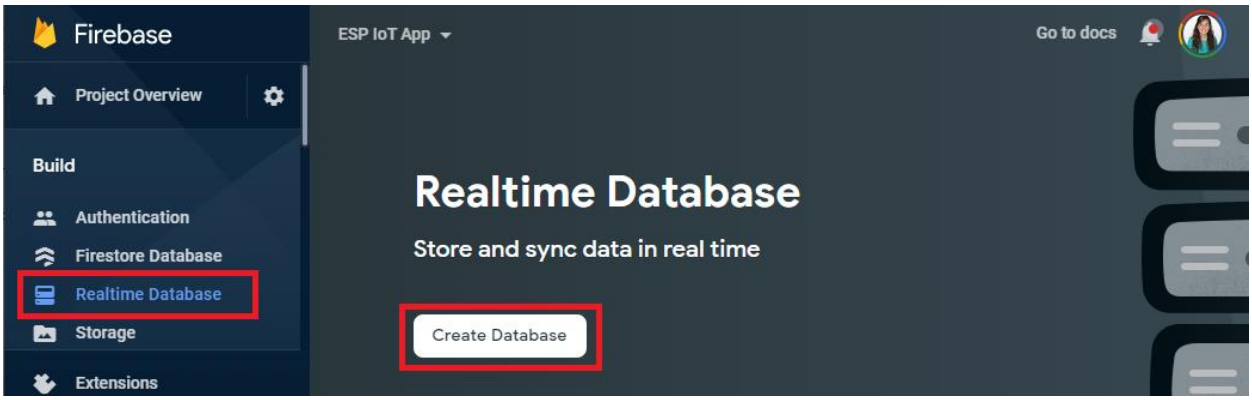
Firebase provides two different options of NoSQL databases: Realtime Database and Firestore Database.

The Realtime Database stores data in JSON format, while the Firestore Database stores data in documents organized into collections. To learn more about the

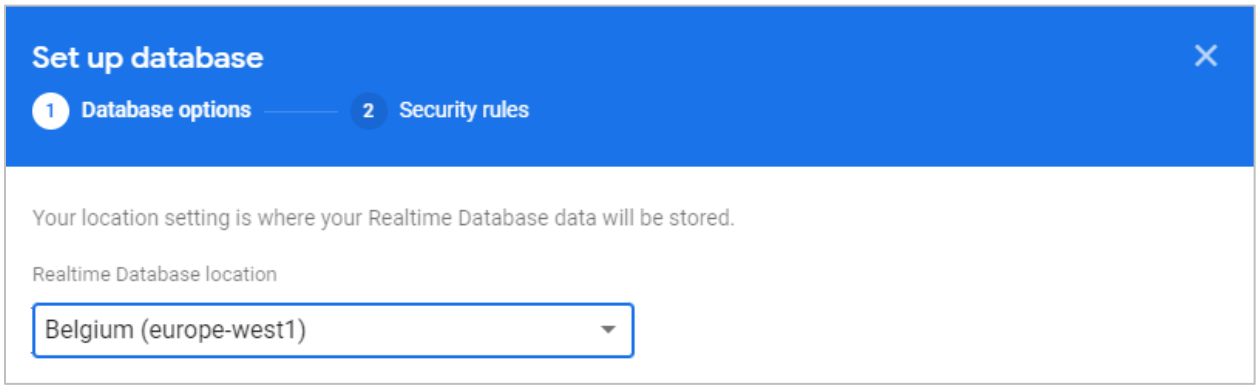
differences between the Realtime Database and the Firestore Database, we recommend reading the Documentation.

In our project, we'll use the Realtime Database (RTDB). Follow the next instructions to add a Realtime Database to your project.

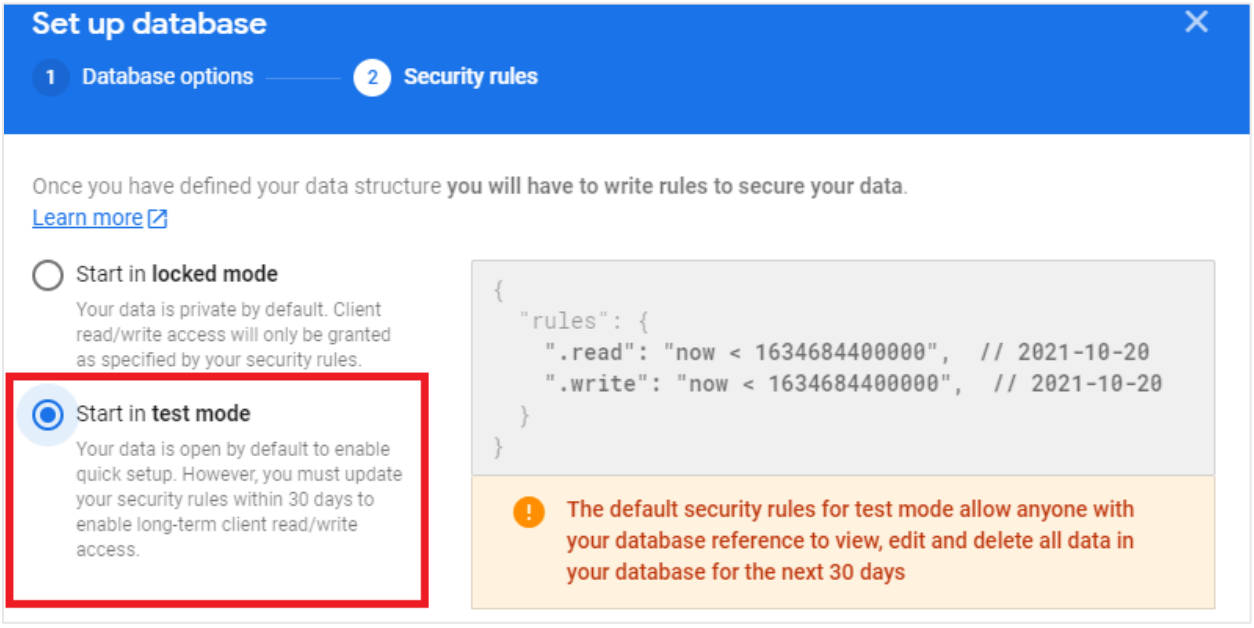
1) On the left sidebar, click on **Realtime Database** and then on **Create Database**.



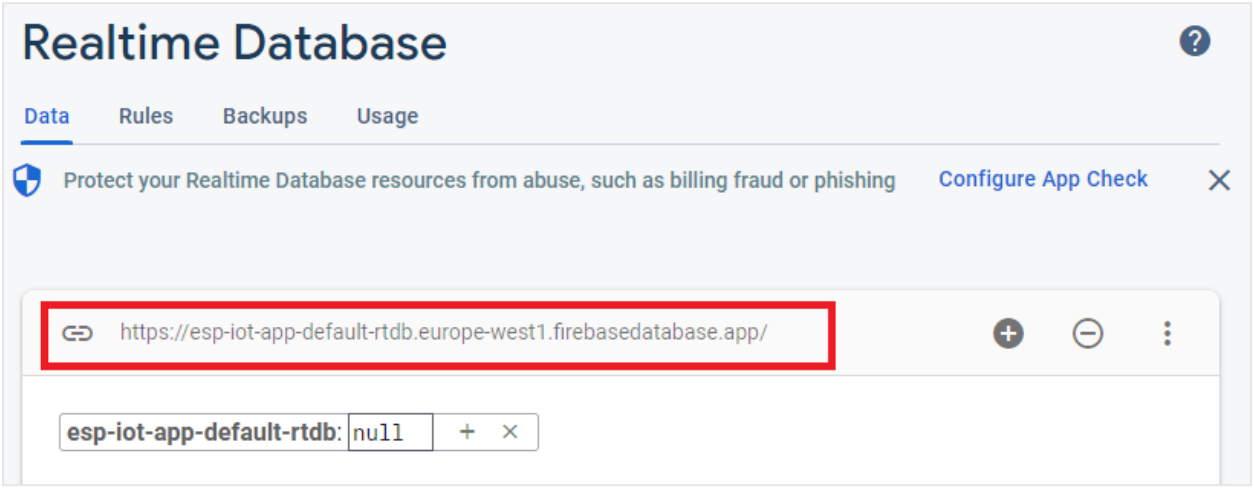
2) Select your database location. It should be the closest to your location (we are from Porto, Portugal). Then, click **Next**.



3) You'll be asked to set up security rules for your database. Select **Start in test mode**. With these database rules, anyone with your database reference can view, edit and delete all data for the next 30 days. But don't worry. In the next section, we'll set up database rules to protect your data. Click **Enable** to finish setting up the Realtime Database.



4) Your database is now created. It is given a unique URL, as highlighted in the following picture. You'll need your database URL later so that the ESP32 or ESP8266 can interact with the database.

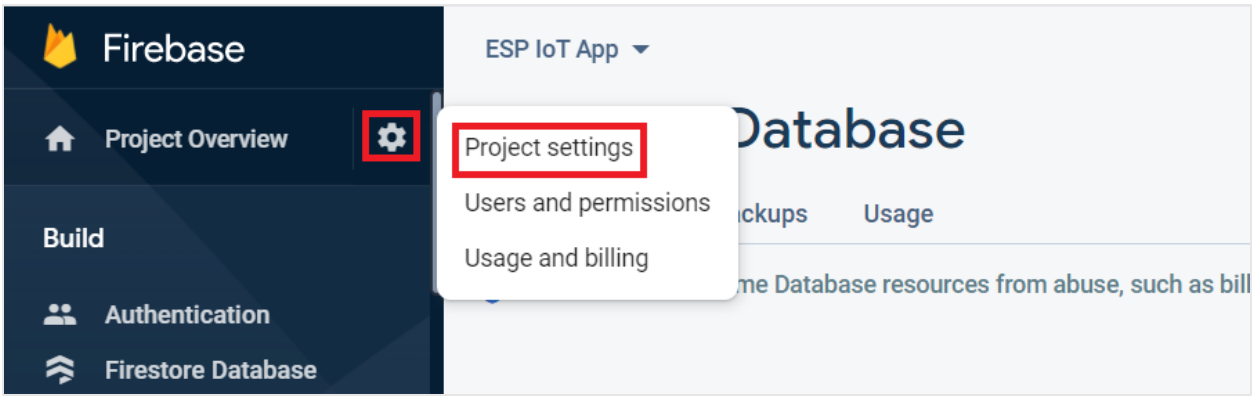


You can create a *.txt* file on your computer to save all the settings like the user UID, database URL, and project API key. Alternatively, you can always come back to the Firebase console to get them.

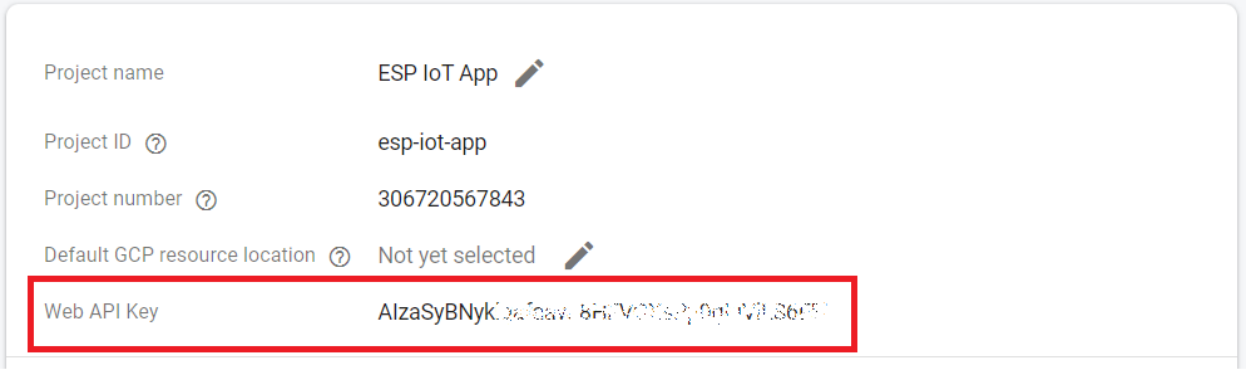
4) Get Project API Key

To interface with your Firebase project using the ESP32 or ESP8266 boards, you need to get your project API key. Follow the next steps to get your project API key.

1) On the left sidebar, click on **Project Settings**.



2) Copy the Web API Key to a safe place because you'll need it later.



Congratulations! Your Firebase project is now all set.

In the next part, we'll take a look at how to organize the database and how to protect your data using database rules.

PART 2

Organizing Your Database and Database Rules

Learn how to organize the data in the Realtime Database to make it easier to set up database rules. Create database nodes to save the data and apply database rules to restrict access. We'll also share helpful database rules that you may want to use in future projects.

2.1 - Organizing Your Database

To better understand what the Firebase Realtime Database is, here's a quote from the Firebase docs:

> *"The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in realtime to every connected client. When you build cross-platform apps with our iOS, Android, and JavaScript SDKs, all of your clients share one Realtime Database instance and automatically receive updates with the newest data"* [3].

All the data stored in the Firebase Realtime Database is stored as JSON objects. So, you can think of the database as a cloud-based JSON tree. When you add data to the JSON tree, it becomes a node with an associated key in the existing JSON structure.

Not familiar with JSON? Read this quick guide.

The best way to organize your data will depend on your project features and how users access the data.

Let's take a look at our project to understand how we should manage the data:

- There's only one user that can access the data—the one you created in the previous section: Set Authentication Methods;
- We need to save the following data in the database:
    - Sensor readings: temperature, humidity, and pressure;
    - Two GPIOs and their corresponding states: 0 (off) or 1 (on)—we'll use GPIO 2 and GPIO 12;

[3] https://firebase.google.com/docs/database

- Two GPIOs and their corresponding PWM duty cycle values: a value between 0 and 100—we'll use GPIOs 13 and 14;
- A message that will be displayed on the OLED display.

Taking into account the data we want to save and that we want to restrict access to one authorized user, here's how we'll structure the database (the key's values are just an example):

- **UsersData**:
  - **<user_uid>***:
    - **outputs**:
      - **digital**:
        - **2**: 0
        - **12**: 1
      - **pwm**:
        - **13**: 20
        - **14**: 80
      - **message**: "I love this app"
    - **sensor**:
      - **temperature**: 25.00
      - **humidity**: 52.50
      - **pressure**: 1006.13

* this is the unique User UID of the logged-in authorized user.

In JSON format, here's how it would look like:

```
{
  "UsersData" : {
    "JXhbigMqPsOGEg9T99WXXXXXXXX"* : {
      "outputs" : {
        "digital" : {
          "2" : 0,
          "12" : 1
        },
        "message" : "I love this app",
        "pwm" : {
          "13" : 20,
          "14" : 80
        }
      },
      "sensor" : {
        "humidity" : 52.50,
        "pressure" : 1006.13,
        "temperature" : 25.00
      }
    }
  }
}
```

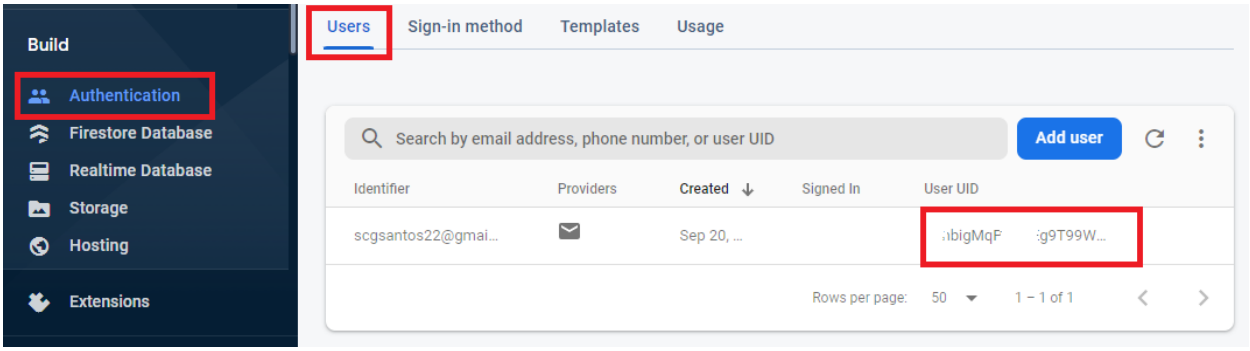
* this is the unique User UID of the logged-in authorized user.

Creating Database Nodes

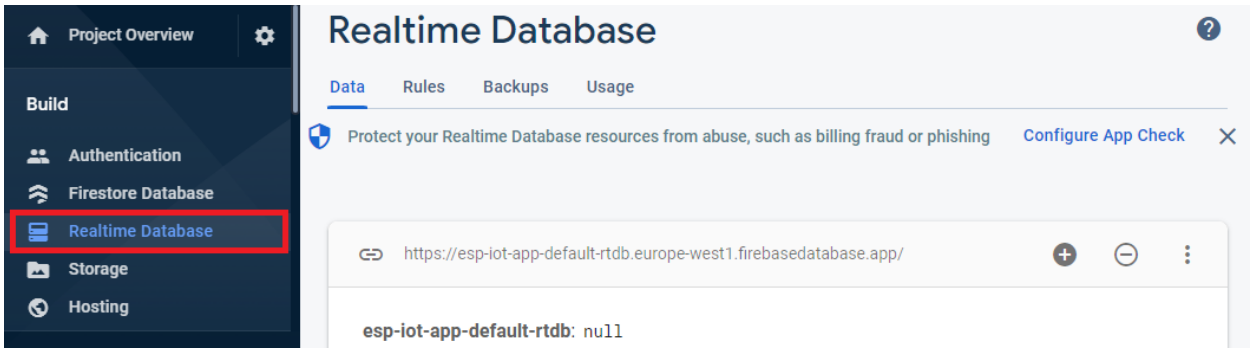
Now let's create the database nodes in our database. You can create the nodes manually by writing the nodes on the Firebase console, on the web app, or via the ESP32/ESP8266. We'll create them manually, so it is easier to follow.

We'll save all the user data inside a node with the authorized user UID. So, get your user UID—follow the next steps.

1) In your Firebase console, select your Project, go to the **Authentication** tab, select **Users**, and copy the user UID from the table.



2) Click on the **Realtime Database** so that we start creating the nodes.



3) You can create the database nodes manually by using the (**+**) icons on the database. However, to prevent typos, we provide a JSON file that you can upload to create the same nodes as ours. Click the link below to download the JSON file.

- Click here to download the JSON file

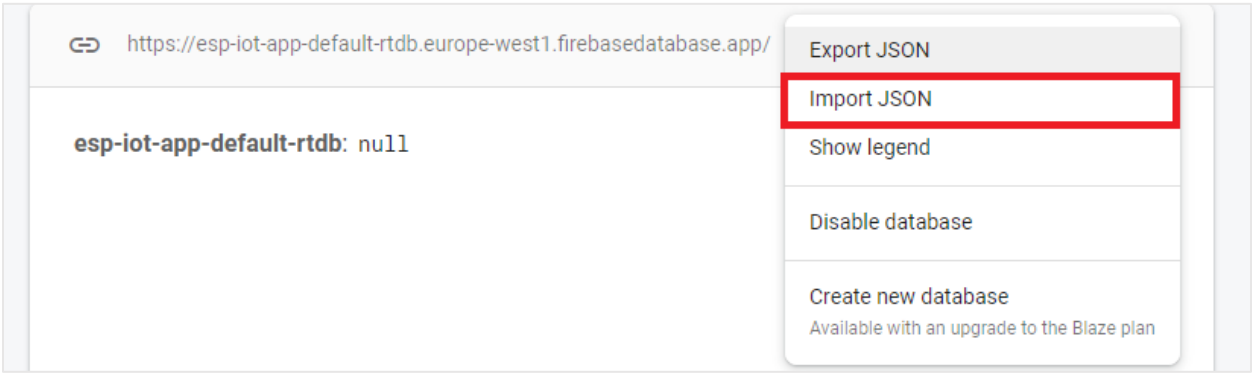
4) Unzip the downloaded folder and open the JSON file. You can use any text editor like Notepad or VS Code, for example. Replace the `REPLACE_WITH_YOUR_USER_UID` text with the user UID you've gotten from step number 1. Then, save your JSON file.

```
{
  "UsersData" : {
    "REPLACE_WITH_YOUR_USER_UID" : {
      "outputs" : {
        "digital" : {
          "2" : 0,
          "12" : 1
```

Your JSON file will look something as follows but with your authorized user UID.

```
{
  "UsersData" : {
    "JXhbigMqPsOGEg9T99W          " : {
      "outputs" : {
        "digital" : {
          "2" : 0,
          "12" : 1
        },
        "message" : "I love this app",
        "pwm" : {
          "13" : 20,
          "14" : 80
        }
      },
      "sensor" : {
        "humidity" : 52.50,
        "pressure" : 1006.13,
        "temperature" : 25.00
      }
    }
  }
}
```

5) Now, go back to your database on the Firebase console. Click on the three-dot icon and select **Import JSON**.



6) Select the JSON file that you've just edited.

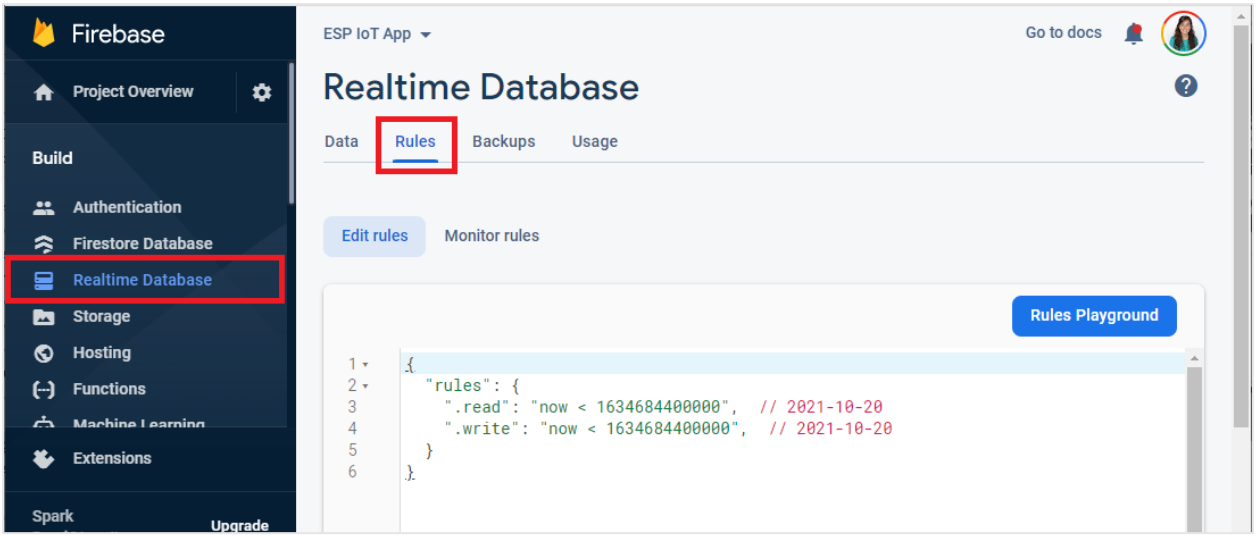
7) Your database should look as shown below but with your unique UID. You can expand the nodes to see all the data.

All the database nodes for this project are created. The values for each node are just arbitrary values so that we can test the database. Later you'll insert actual data into the nodes.

2.2 - Setting Up Database Rules

At this moment, anyone with a reference to your database can read, write and delete data. You don't want this to happen. You only want the authorized user to have access to the data.

1) In your Firebase console, go to the **Realtime Database** and select the **Rules** tab.

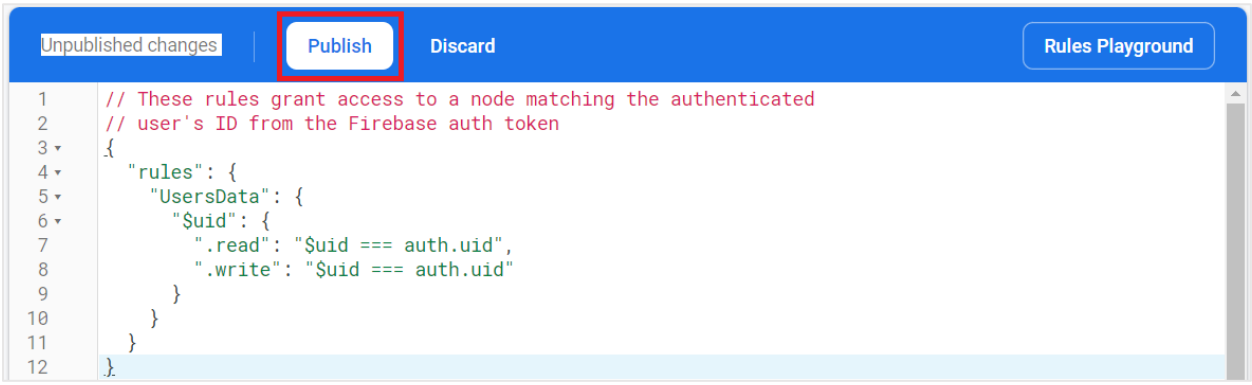


At the moment, your database rules should be similar to mine but with a different due date. You want to protect the data so that only the authorized user can read and write to the database under the node with its UID.

2) To do that, copy the following rules to your database rules.

```
// These rules grant access to a node matching the authenticated
// user's ID from the Firebase auth token
{
  "rules": {
    "UsersData": {
      "$uid": {
        ".read": "$uid === auth.uid",
        ".write": "$uid === auth.uid"
      }
    }
  }
}
```

3) Then, click **Publish**.



And that's it! Your database and database rules are all set! Proceed to the next section to start interacting with the database using the ESP32 or ESP8266 boards.

Other Useful Database Rules

You can change the database rules any time later on. Here are some database rules that might be useful for future projects or if you want to change this project.

No Security

Anyone can read or write to your database. This is useful for testing purposes.

```
{
 "rules": {
    ".read": true,
    ".write": true
  }
}
```

Full Security

No one can access the database. You can only read/write using the Firebase console.

```
{
  "rules": {
    ".read": false,
    ".write": false
  }
 }
```

Only Authenticated Users Can Access the Data

It doesn't matter who the user is. As long as it is authenticated, it can access all database nodes.

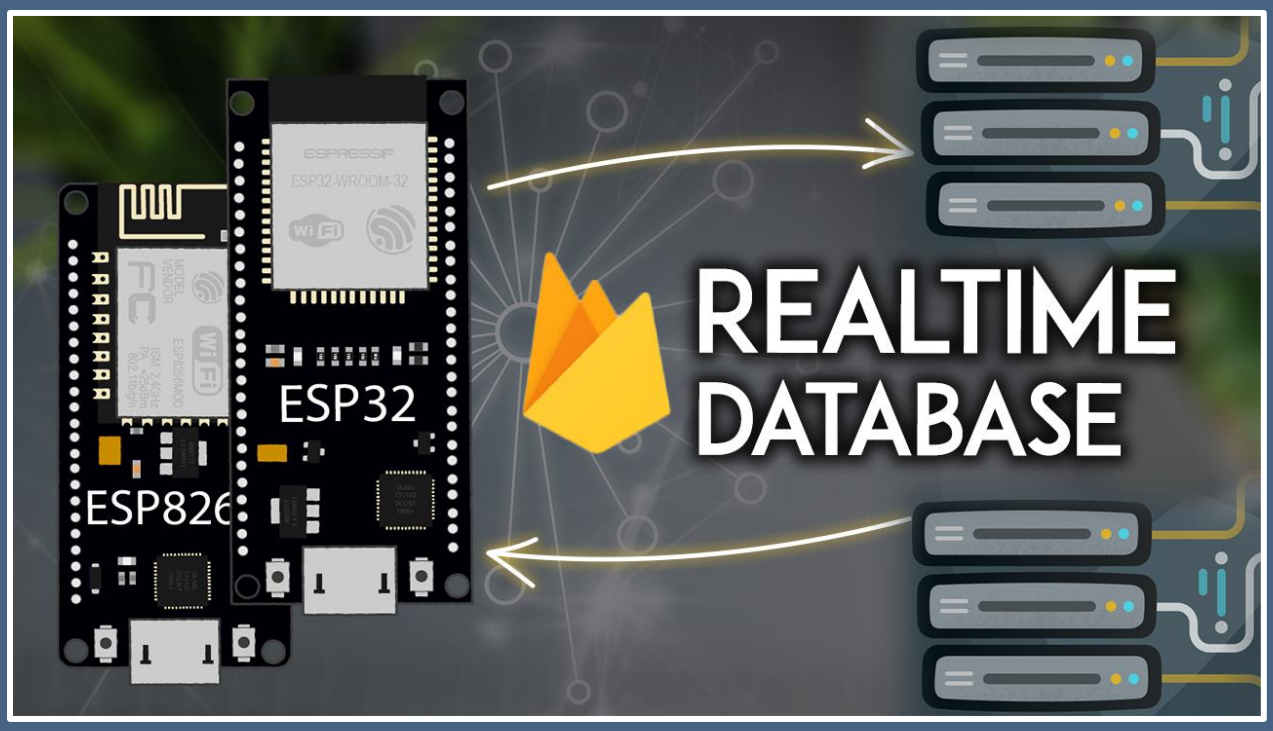
```
{
  "rules": {
    ".read": "auth != null",
    ".write": "auth != null"
  }
}
```

Further Reading

To learn more about database rules, check the Firebase documentation.

PART 3

ESP32/ESP8266: Interacting with the Realtime Database

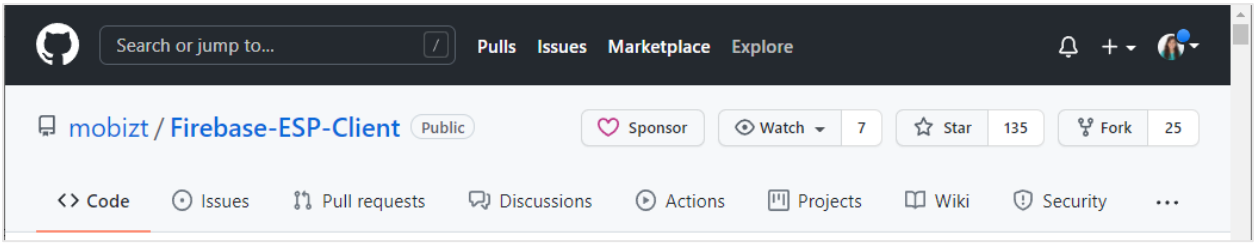


This section will teach you how to interact with the Realtime Database using the ESP32 or ESP8266 boards. You'll learn to authenticate with the ESP board, read and write to the database and automatically detect any database changes.

3.1 - Installing Firebase Library

Now that the Realtime Database is created, you'll learn how to interface the ESP32 and ESP8266 boards with the database.

We'll use the Firebase-ESP-Client Library by Mobizt to interact between the ESP32/ESP8266 and the Firebase project.



Prerequisites

Before proceeding, you must know how to program the ESP32 and ESP8266 boards. You can use Arduino IDE, VS Code with the PlatformIO extension, or any other software. Make sure you have the ESP32 or the ESP8266 add-ons installed. You can follow one of the next tutorials depending on the board and software you're using:

- Installing the ESP32 Board in Arduino IDE
- Installing ESP8266 Board in Arduino IDE
- Getting Started with VS Code and PlatformIO IDE for ESP32 and ESP8266

Firebase-ESP-Client Library

The Firebase-ESP-Client library provides many examples to use Firebase with the ESP32 and ESP8266 boards. You can check all the examples here. It also has detailed documentation explaining how to use the library.

Installation – VS Code + PlatformIO

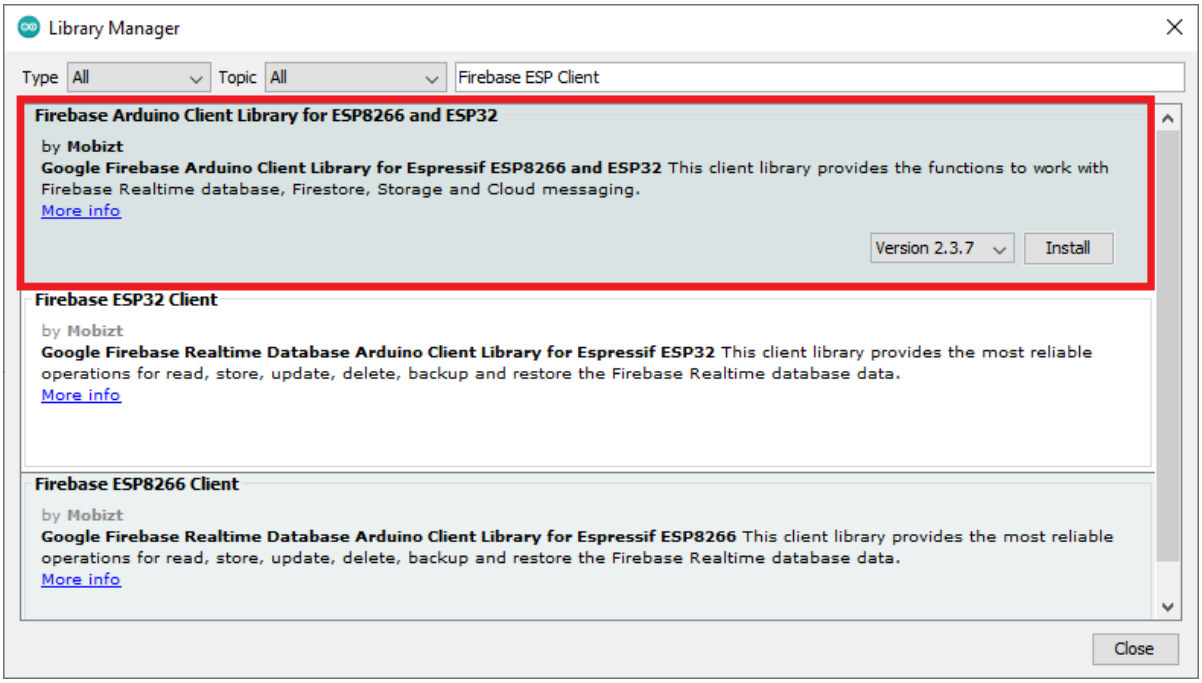
You'll install the Firebase ESP Client library later when you create the project folder.

Installation – Arduino IDE

If you're using Arduino IDE, you can install the library now.

1) Go to **Sketch** > **Include Library** > **Manage Libraries**.

2) Search for *Firebase ESP Client* and install the **Firebase Arduino Client Library for ESP8266 and ESP32** by Mobitz.



Now, you're all set to start programming the ESP32 and ESP8266 boards.

3.2 - Authentication with Email and Password

In this section, you'll learn how to authenticate the ESP32/ESP8266 as a user with an email and password so that it can interact with the Firebase project.

Download the resources for this project:

⇒ Download Sketch folder (**Arduino IDE**)

⇒ Download ESP32 project folder (**VS Code + PlatformIO**)

⇒ Download ESP8266 project folder (**VS Code + PlatformIO**)

Creating a New Project (VS Code)

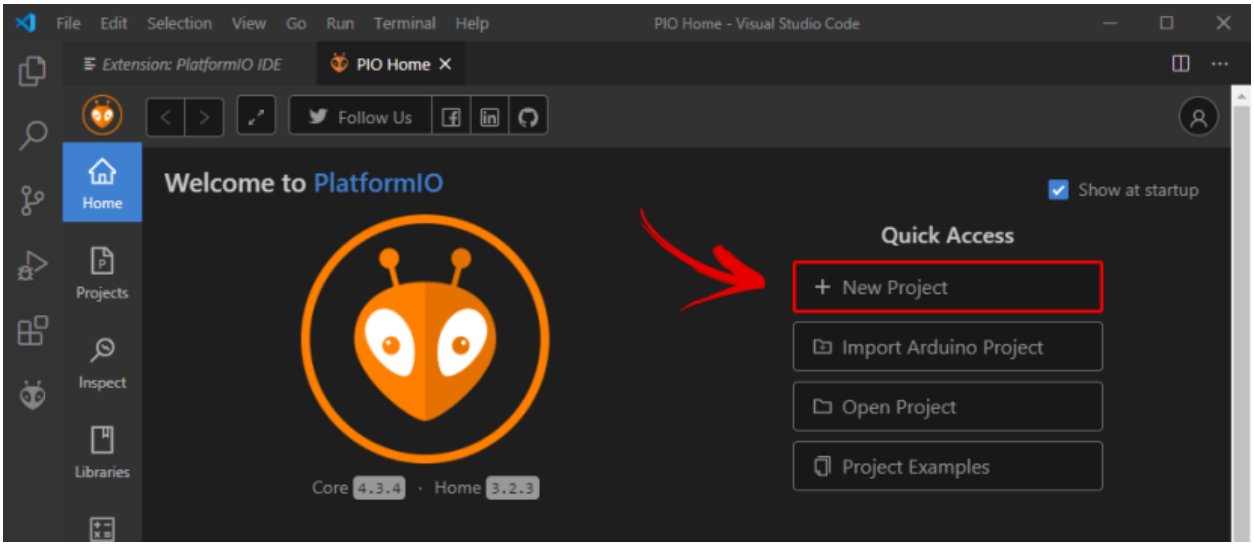
If you're new to VS Code, take a quick look at this section to learn how to create a new project to program your ESP32 and/or ESP8266 boards. You can still use Arduino IDE to program your boards if you prefer.

First, you need to install PlatformIO in VS Code. Follow the next tutorial to learn how to do that:

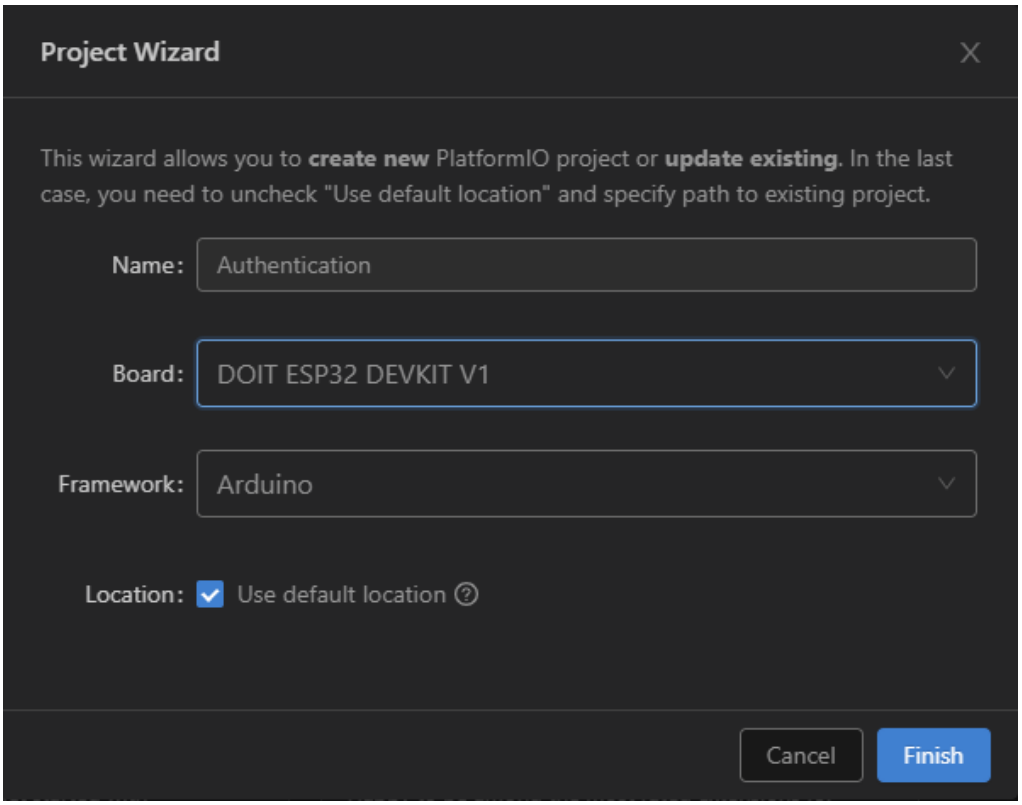
- Getting Started with VS Code and PlatformIO IDE for ESP32 and ESP8266

After installing PlatformIO and making sure everything is working as shown in the previous tutorial, you can create a new project in VS Code to program your boards. Follow the next instructions.

1) On VS Code, click on the Home icon at the blue bottom bar. Then, click on **+New Project** to start a new project. See the screenshot below.



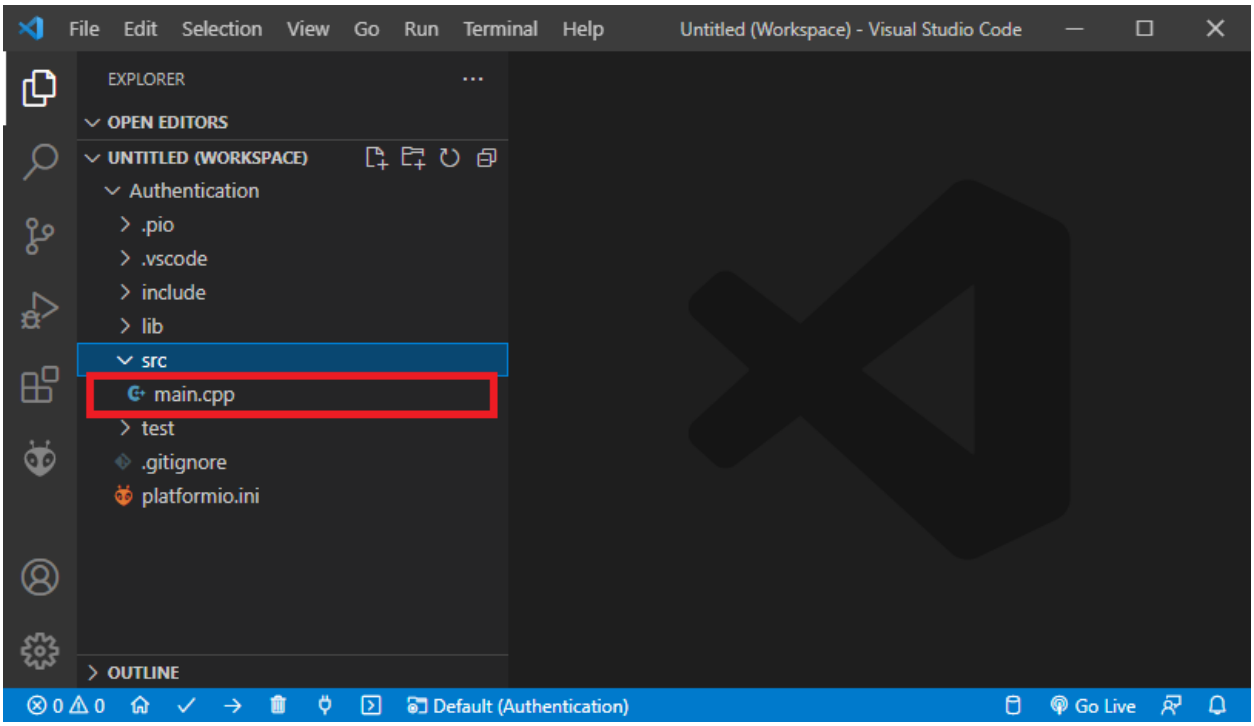
2) Give your project a name (for example, *Authentication*) and select the board you're using—for example, the [DOIT ESP32 DEVKIT V1]. The Framework should be "**Arduino**" to use the Arduino core. You can choose the default location to save your project or a custom location. The default location is in this path **Documents** > **PlatformIO**> **Projects**. After selecting a folder to save your project, click **Finish**.



After creating the project, PlatformIO creates several folders and files for the project. You just need to worry about the *main.cpp* file and the *platformio.ini* file.

The *main.cpp* file is where you write the code you want to upload to the board.

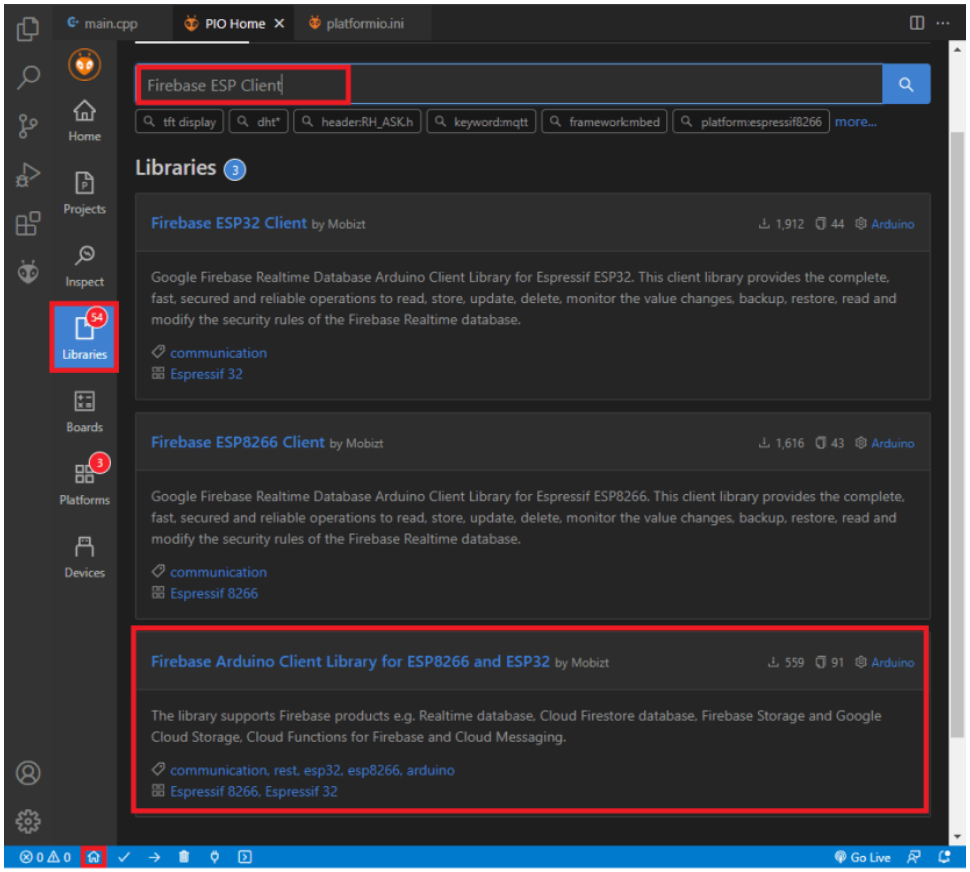
The *platformio.ini* file is the PlatformIO Configuration File for your project. It shows the platform, board, and framework for your project. You can also add other configurations like libraries, upload options, and change the Serial Monitor baud rate, etc.



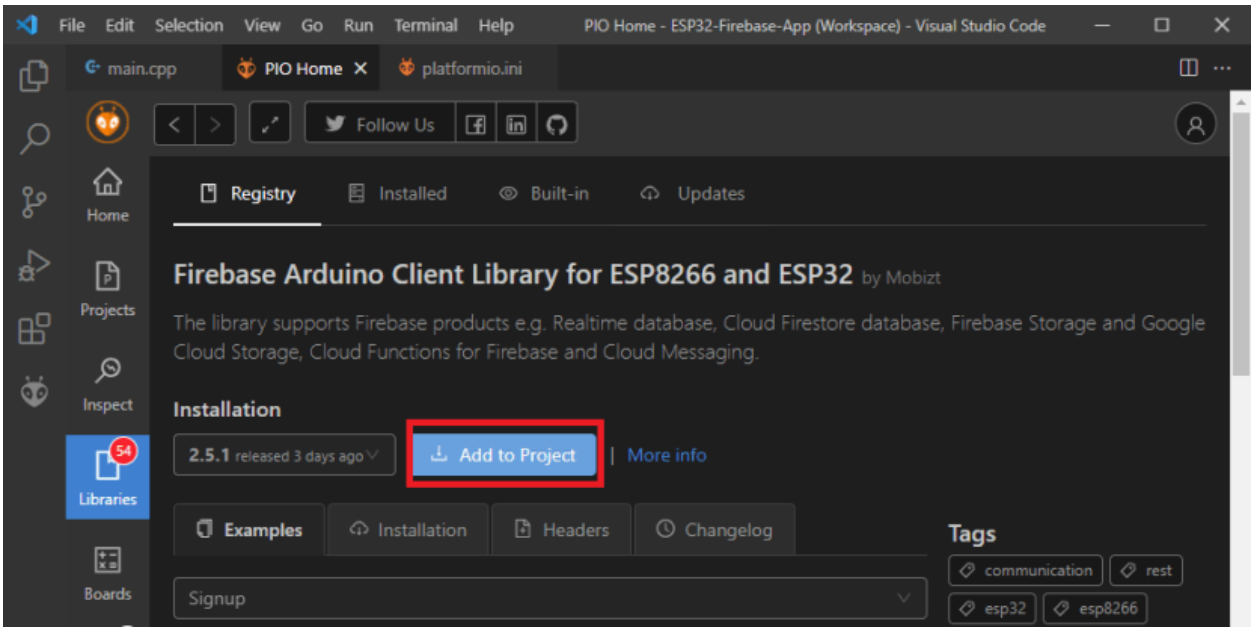
Installing the ESP Firebase Client Library (VS Code)

If you're using VS Code + PlatformIO, follow the next instructions to install the ESP Firebase Client library.

1) Click on the **PIO Home** icon and then select the **Libraries** tab. Search for "*Firebase ESP Client*". Select the **Firebase Arduino Client Library for ESP8266 and ESP32**.



2) Then, click **Add to Project** and select the project you're working on.

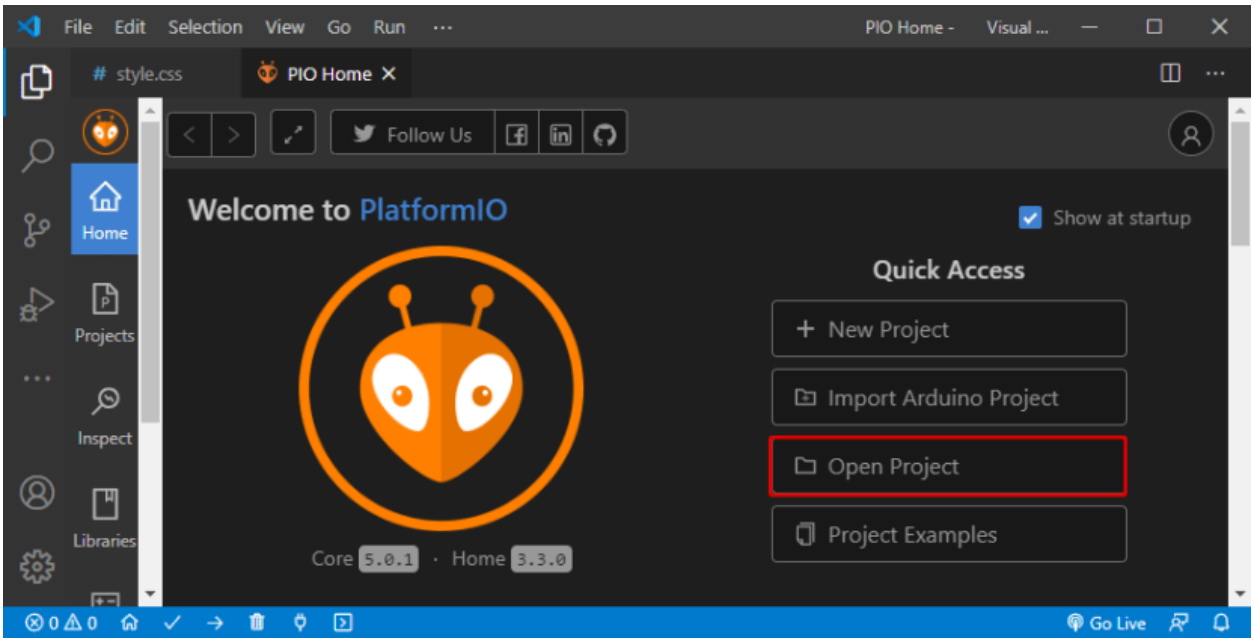


Open an Existing Project Folder (VS Code)

If you prefer to work directly on the project folders provided, you just need to download the project files by clicking on the links provided at the beginning of the Unit. The link will download a .ZIP file with the project folder.

1) First, unzip the folder.

2) To open an existing project folder on PlatformIO, open VS Code, go to PlatformIO Home and click on **Open Project**. Navigate through the files and select your project folder.



PlatformIO will open all the files within the project folder.

Code - Authentication

Create a new project for your ESP32 or ESP8266 board using Arduino IDE or VS Code (alternatively, you can download the project files provided previously).

If you're using VS Code, don't forget to add the following line to your *platformio.ini* file to change the Serial Monitor baud rate to 115200.

```
monitor_speed = 115200
```

Copy the code below to Arduino IDE or to the *main.cpp* file if you're using VS Code.

The code is compatible with both the ESP32 and ESP8266 boards.

```
#include <Arduino.h>
#if defined(ESP32)
  #include <WiFi.h>
#elif defined(ESP8266)
  #include <ESP8266WiFi.h>
#endif
#include <Firebase_ESP_Client.h>

// Provide the token generation process info.
#include "addons/TokenHelper.h"
// Provide the RTDB payload printing info and other helper functions.
#include "addons/RTDBHelper.h"

// Insert your network credentials
#define WIFI_SSID "REPLACE_WITH_YOUR_SSID"
#define WIFI_PASSWORD "REPLACE_WITH_YOUR_PASSWORD"

// Insert Firebase project API Key
#define API_KEY "REPLACE_WITH_YOUR_PROJECT_API_KEY"

// Insert Authorized Email and Corresponding Password
#define USER_EMAIL "REPLACE_WITH_THE_USER_EMAIL"
#define USER_PASSWORD "REPLACE_WITH_THE_USER_PASSWORD"

// Define Firebase objects
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;

// Variable to save USER UID
String uid;

// 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();
}

void setup(){
  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;

  // 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);
  }
  // Print user UID
  uid = auth.token.uid.c_str();
  Serial.print("User UID: ");
  Serial.print(uid);
}

void loop(){

}
```

You need to insert your network credentials (SSID and password), Firebase project API key, and the user email and password—the one you added to the Authentication Users.

How the Code Works

Continue reading to learn how the code works, or skip to the Demonstration section.

Include Libraries

First, include the required libraries. The `WiFi.h` library to connect the ESP32 to the internet (or the `ESP8266WiFi.h` library in case of the ESP8266 board) and the `Firebase_ESP_Client.h` library to interface the boards with Firebase.

```
#include <Arduino.h>
#if defined(ESP32)
  #include <WiFi.h>
#elif defined(ESP8266)
  #include <ESP8266WiFi.h>
#endif
#include <Firebase_ESP_Client.h>
```

You also need to include the following for the Firebase library to work.

```
// Provide the token generation process info.
#include "addons/TokenHelper.h"
// Provide the RTDB payload printing info and other helper functions.
#include "addons/RTDBHelper.h"
```

Network Credentials

Include your network credentials in the following lines so that your boards can connect to the internet using your local network.

```
// Insert your network credentials
#define WIFI_SSID "REPLACE_WITH_YOUR_SSID"
#define WIFI_PASSWORD "REPLACE_WITH_YOU_PASSWORD"
```

Firebase Project API Key and Firebase User

Insert your Firebase project API key—the one you've gotten in this section.

```
#define API_KEY "REPLACE_WITH_YOUR_PROJECT_API_KEY"
```

Insert the authorized user email and the corresponding password—these are the details of the user you've added in this section.

```
// Insert Authorized Email and Corresponding Password
#define USER_EMAIL "REPLACE_WITH_THE_USER_EMAIL"
#define USER_PASSWORD "REPLACE_WITH_THE_USER_PASSWORD"
```

Firebase Objects and Other Variables

The following line defines a `FirebaseData` object.

```
FirebaseData fbdo;
```

The next line defines a `FirebaseAuth` object needed for authentication.

```
FirebaseAuth auth;
```

The following line defines `FirebaseConfig` object needed for configuration data.

```
FirebaseConfig config;
```

The `uid` variable will be used to save the user's UID. We can get the user's UID after the authentication.

```
String uid;
```

initWiFi()

The `initWiFi()` function connects your ESP to the internet using the network credentials provided. You must call this function later in the `setup()` to initialize WiFi.

```
// 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()

In the `setup()`, initialize the Serial Monitor for debugging purposes at a baud rate of 115200.

```
Serial.begin(115200);
```

Call the `initWiFi()` function to initialize WiFi.

```
initWiFi();
```

Assign the API key to the Firebase configuration.

```
config.api_key = API_KEY;
```

The following lines assign the email and password to the Firebase authentication object.

```
auth.user.email = USER_EMAIL;
auth.user.password = USER_PASSWORD;
```

Add the following to the configuration object.

```
// 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;
```

Finally, initialize the Firebase library (authenticate) with the configuration and authentication settings we defined earlier.

```
// Initialize the library with the Firebase authen and config
Firebase.begin(&config, &auth);
```

After initializing the library, we can get the user UID by calling `auth.token.uid`. Getting the user's UID might take some time, so we add a while loop that waits until we get it.

```
while ((auth.token.uid) == "") {
  Serial.print('.');
  delay(1000);
}
```

Finally, we save the user's UID in the `uid` variable and print it in the Serial Monitor.

```
uid = auth.token.uid.c_str();
Serial.print("User UID: ");
Serial.print(uid);
```

Demonstration

After uploading the code, open the Serial Monitor at a baud rate of 115200.

Note: if you're using VS Code, you need to add the following line to your `platformio.ini` file to change the baud rate. Then, save your file.

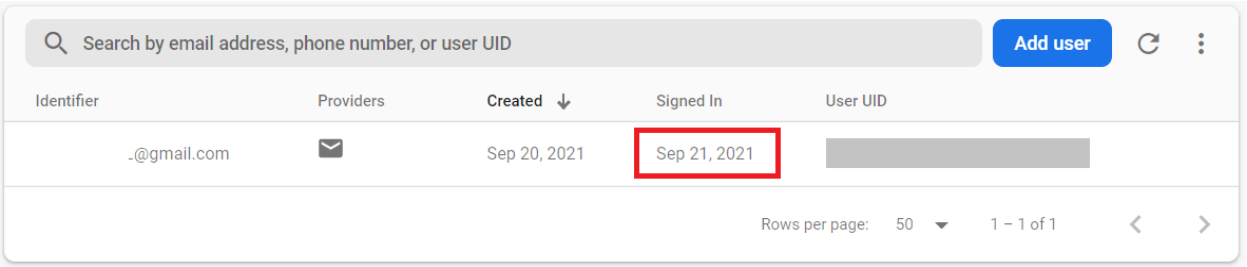
```
monitor_speed = 115200
```

Reset your board by pressing the on-board EN/RST button.

The ESP32/ESP8266 authenticates the user successfully and gets its UID.

```
entry 0x400806a8
Connecting to WiFi .....192.168.1.85

Getting User UID
.Token info: type = id token, status = on request
.Token info: type = id token, status = ready
User UID: JXhbigMqPsOGEg9T99
```

Then, go to your Firebase project console to check if it actually signed in as a user. Go to **Authentication** > **Users**. Now you should have the current date in the **Signed In** field. Additionally, you'll see that the User UID matches the UID printed by your ESP board in the Serial Monitor.



Congratulations! You successfully signed in your ESP32/ESP8266 board as a user. In the next section, you'll learn how to write data to the Firebase Realtime Database.

3.3 - Send Data to the Database: Sensor Readings

In this section, you'll learn how to send data to the Firebase Realtime Database. We'll send temperature, humidity, and pressure from a BME280 sensor. You can use any other sensor you're familiar with as long as you add the required code snippets to get data from that sensor.

Download the resources for this project:

⇒ Download Sketch folder (**Arduino IDE**)

⇒ Download Project folder ESP32 (**VS Code + PlatformIO**)

⇒ Download Project folder ESP8266 (**VS Code + PlatformIO**)

Project Overview

In this section, you'll send BME280 sensor readings—temperature, humidity and pressure—to the Firebase Realtime Database to the following nodes:

- temperature: `UsersData/<user_uid>*/sensor/temperature`
- humidity: `UsersData/<user_uid>*/sensor/humidity`
- pressure: `UsersData/<user_uid>*/sensor/pressure`

* this is the unique User UID of the logged-in authorized user.

Schematic Diagram

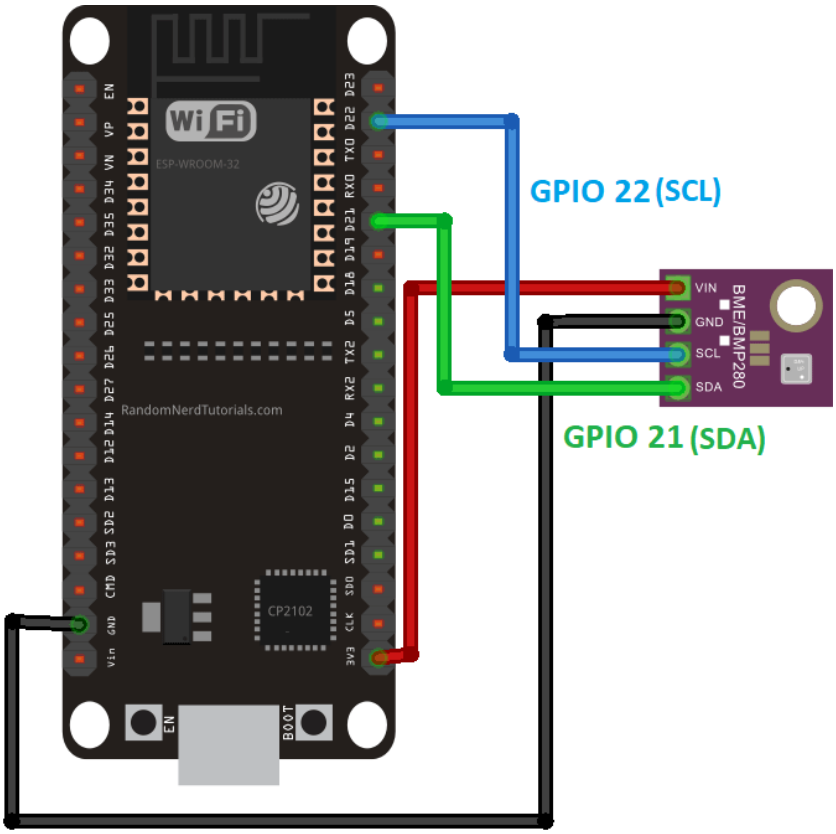
In our project, we'll send BME280 readings to the Firebase Realtime Database. Wire the BME280 sensor to your ESP32 or ESP8266 on the I2C pins:

ESP32	SDA: **GPIO 21**	SCL: **GPIO22**
ESP8266	SDA: **GPIO 4**	SCL: **GPIO 5**

You can also use one of the following diagrams as a reference.

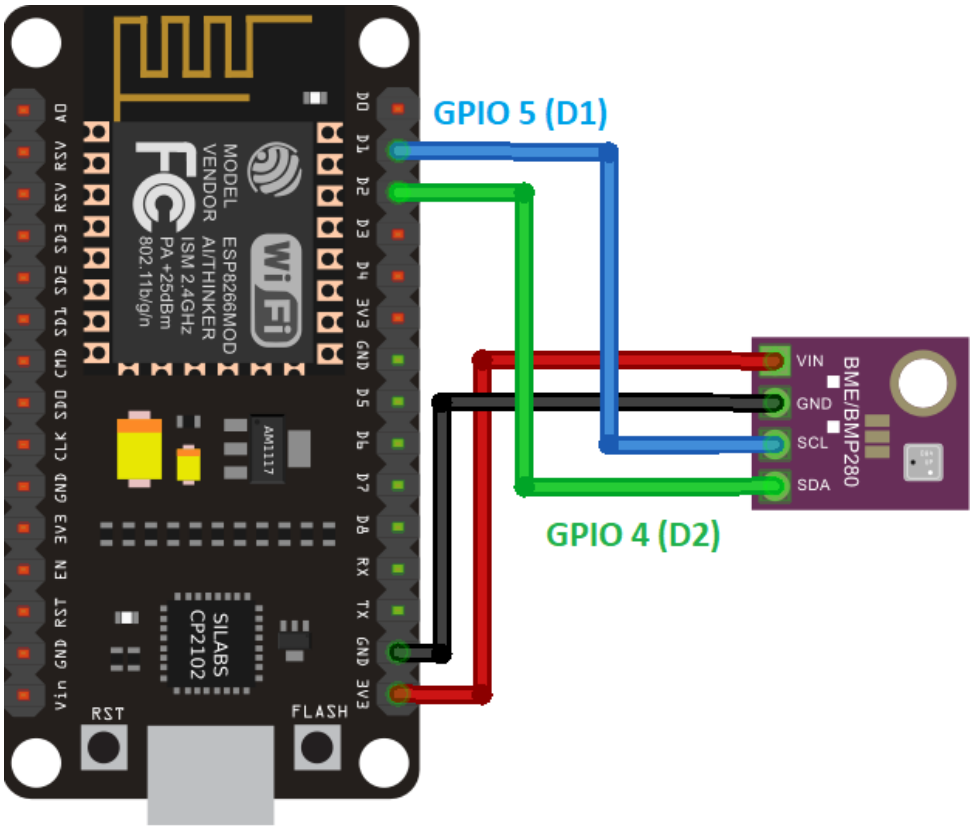
ESP32

Wire the BME280 sensor to your ESP32 board as shown in the following schematic diagram.



ESP8266

If you're using an ESP8266, follow the next diagram instead.



Installing Required Libraries

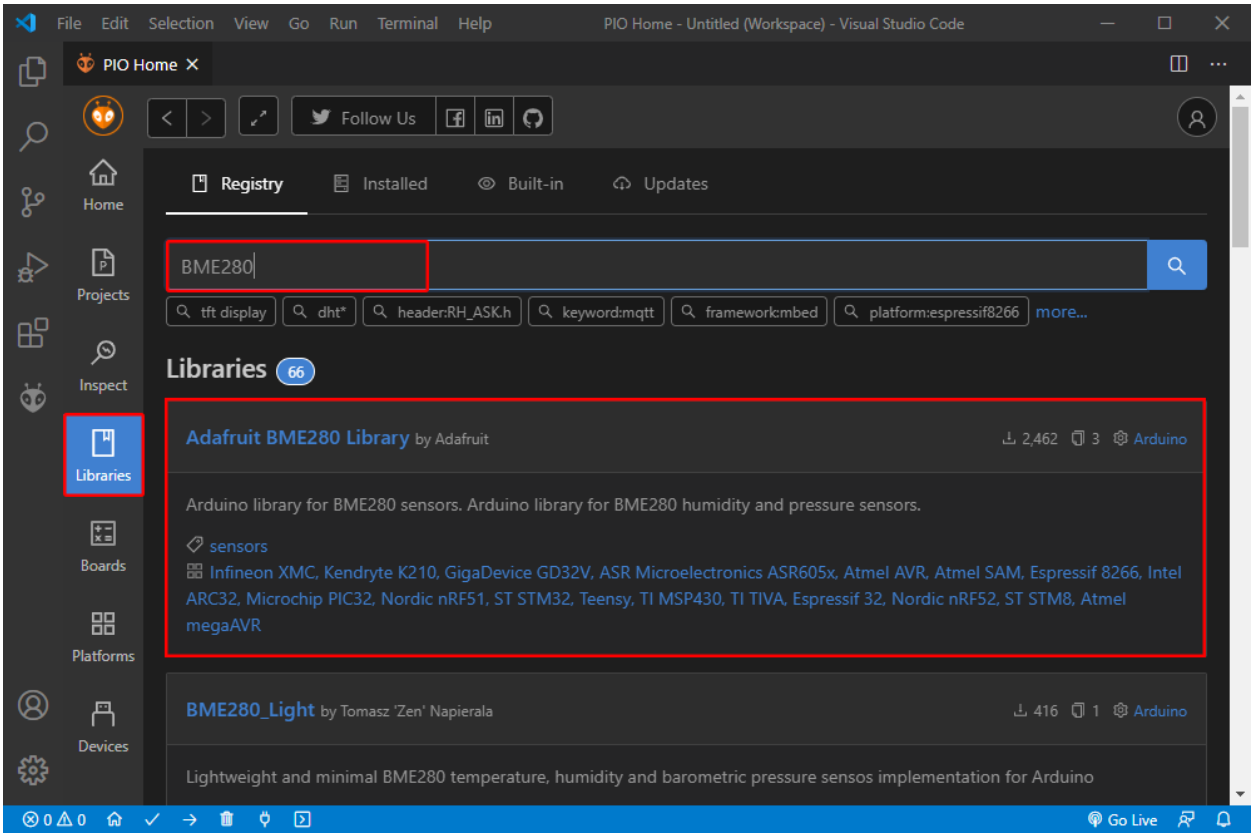
There are different ways to get data from the BME280 sensor. We'll use the Adafruit BME280 library. Follow the next steps to install the library.

VS Code + PlatformIO

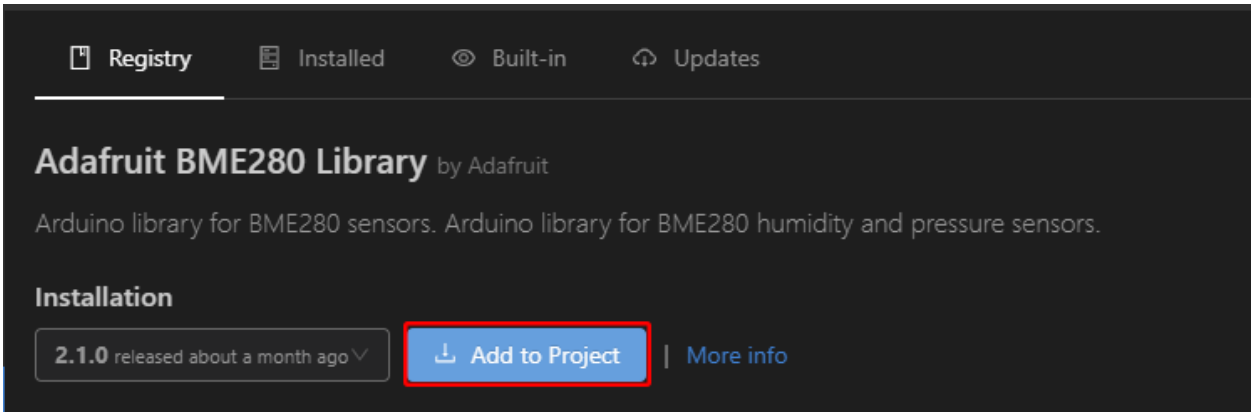
Follow the subsequent instructions if you're using VS Code to install the BME280 library.

1) With your project folder opened, click the **Home** icon to go to PlatformIO **Home**. Select the **Libraries** icon on the left menu.

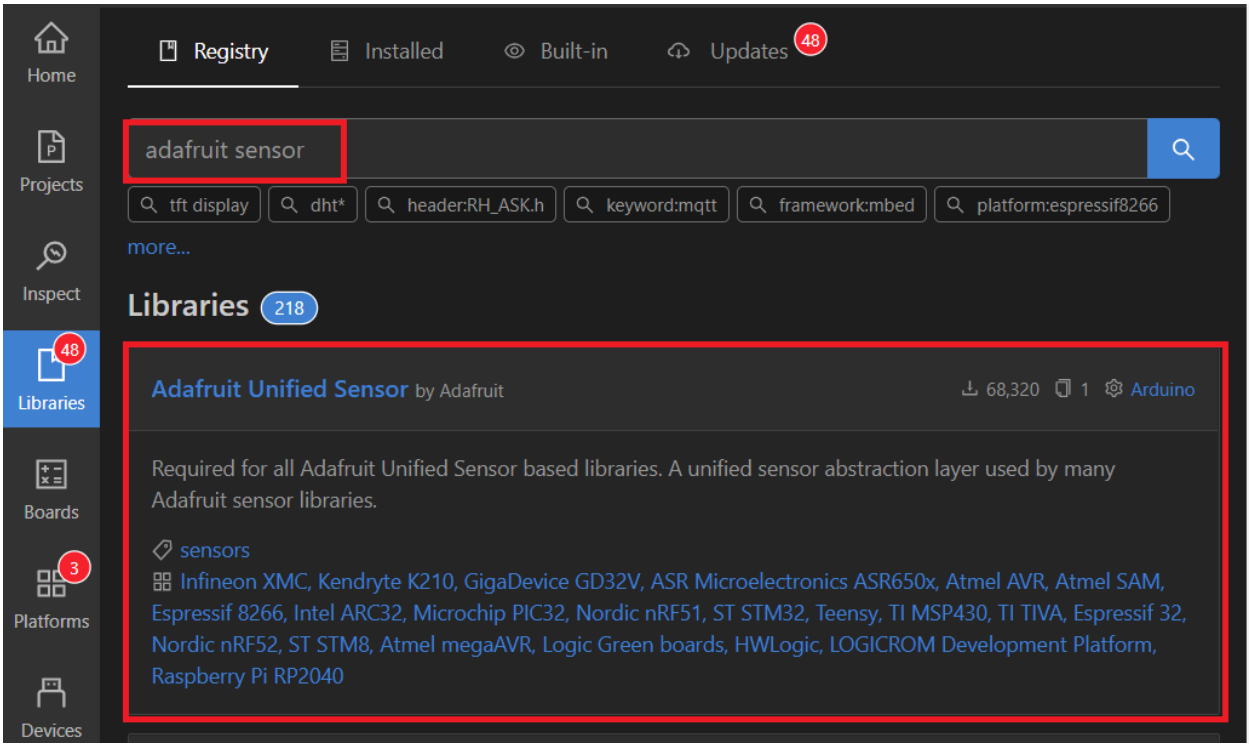
2) Search for the library you want to install—**BME280**.



3) Select the **Adafruit BME280 Library**. Then, click **Add to Project** and select the project where you want to use the library.



4) The Adafruit BME280 library also needs the Adafruit Unified sensor library to work. Repeat steps **2)** and **3)** but search for "*adafruit sensor*" instead and install the **Adafruit Unified Sensor** library.

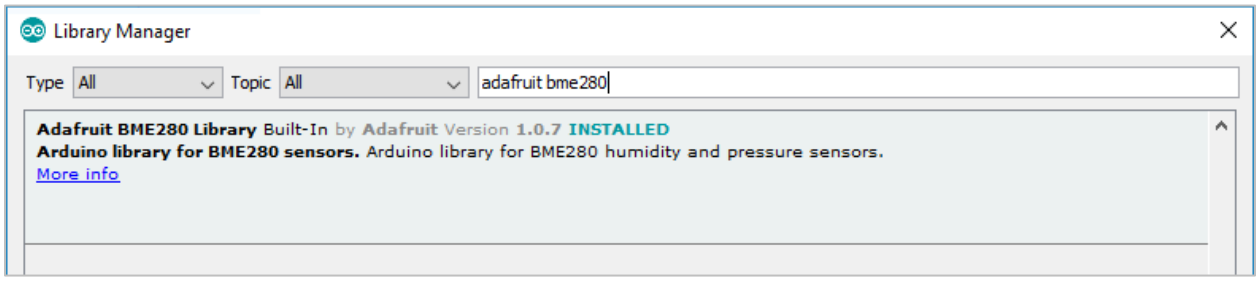


Arduino IDE

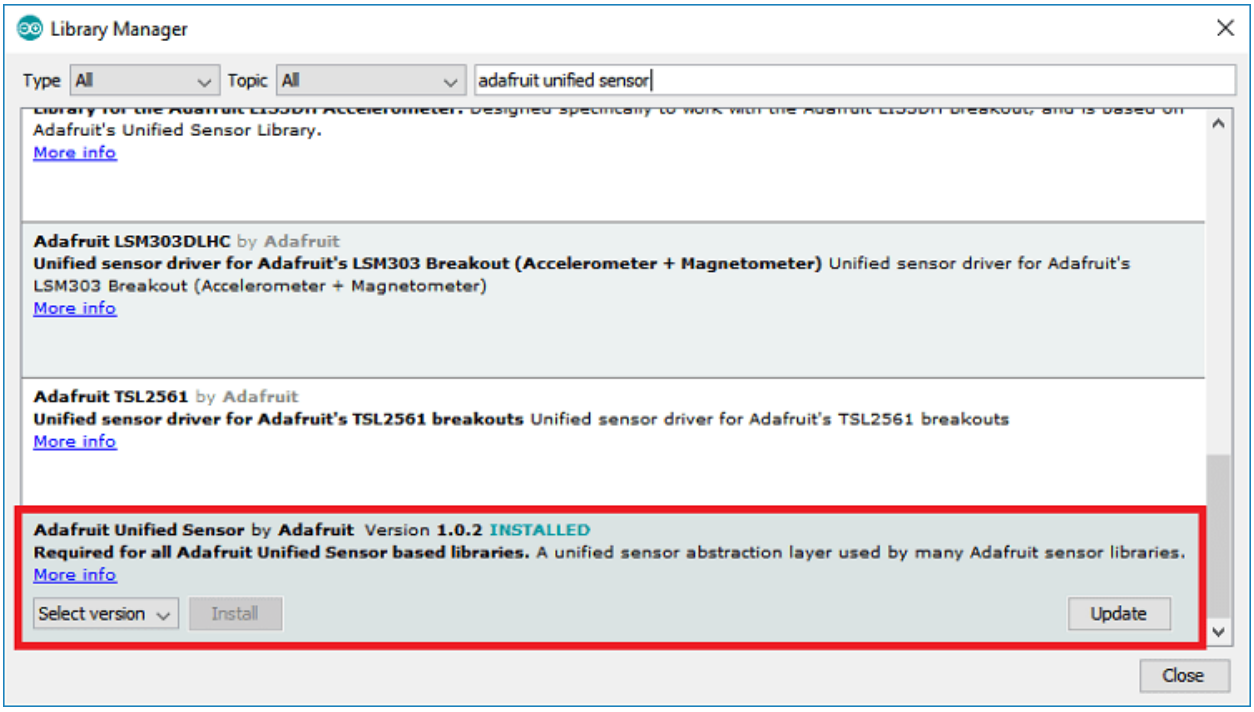
Follow the subsequent instructions if you're using Arduino IDE to install the BME280 library.

1) In the Arduino IDE, go to **Sketch** > **Include Library** > **Manage Libraries**. The Library Manager should open.

2) Search for "*adafruit bme280* " on the Search box and install the library.



3) Then, search for "*Adafruit Unified Sensor*" in the search box. Scroll down to find the library and install it.



4) After installing the libraries, restart your Arduino IDE.

Code - Send Sensor Readings to the Database

Below you can find the code that sends BME280 sensor readings to the database every three minutes. After testing the code and checking that everything is working as expected, we recommend increasing the delay time.

You need to insert your network credentials (SSID and password), Firebase project API key, database URL, and the authorized user email and password.

The new sections of code are highlighted in a light orange color.

```
#include <Arduino.h>
#if defined(ESP32)
  #include <WiFi.h>
#elif defined(ESP8266)
  #include <ESP8266WiFi.h>
#endif
#include <Firebase_ESP_Client.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
// Provide the token generation process info.
```

```
#include "addons/TokenHelper.h"
// Provide the RTDB payload printing info and other helper functions.
#include "addons/RTDBHelper.h"

// Insert your network credentials
#define WIFI_SSID "REPLACE_WITH_YOUR_SSID"
#define WIFI_PASSWORD "REPLACE_WITH_YOUR_PASSWORD"
// Insert Firebase project API Key
#define API_KEY "REPLACE_WITH_YOUR_PROJECT_API_KEY"
// Insert Authorized Email and Corresponding Password
#define USER_EMAIL "REPLACE_WITH_THE_USER_EMAIL"
#define USER_PASSWORD "REPLACE_WITH_THE_USER_PASSWORD"
// Insert RTDB URLefine the RTDB URL
#define DATABASE_URL "REPLACE_WITH_YOUR_DATABASE_URL"

// Define Firebase objects
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;
// Variable to save USER UID
String uid;
// Variables to save database paths
String databasePath;
String tempPath;
String humPath;
String presPath;
// BME280 sensor
Adafruit_BME280 bme; // I2C
float temperature;
float humidity;
float pressure;
// Timer variables (send new readings every three minutes)
unsigned long sendDataPrevMillis = 0;
unsigned long timerDelay = 180000;
// Initialize BME280
void initBME(){
  if (!bme.begin(0x76)) {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
    while (1);
  }
}
// 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();
}
// Write float values to the database
void sendFloat(String path, float value){
```

```
   if (Firebase.RTDB.setFloat(&fbdo, path.c_str(), value)){
      Serial.print("Writing value: ");
      Serial.print (value);
      Serial.print(" on the following path: ");
      Serial.println(path);
      Serial.println("PASSED");
      Serial.println("PATH: " + fbdo.dataPath());
      Serial.println("TYPE: " + fbdo.dataType());
    }
    else {
      Serial.println("FAILED");
      Serial.println("REASON: " + fbdo.errorReason());
    }
}
void setup(){
  Serial.begin(115200);
  // Initialize BME280 sensor
  initBME();
  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;

  // Assign the RTDB URL (required)
  config.database_url = DATABASE_URL;

  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;

  // 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);
  }
  // Print user UID
  uid = auth.token.uid.c_str();
  Serial.print("User UID: ");
  Serial.println(uid);
  // Update database path
  databasePath = "/UsersData/" + uid;
```

```
  // Define database path for sensor readings
  // --> UsersData/<user_uid>/sensor/temperature
  tempPath = databasePath + "/sensor/temperature";
  // --> UsersData/<user_uid>/sensor/humidity
  humPath = databasePath + "/sensor/humidity";
  // --> UsersData/<user_uid>/sensor/pressure
  presPath = databasePath + "/sensor/pressure";

}
void loop(){
  // Send new readings to database
  if (Firebase.ready() && (millis() - sendDataPrevMillis > timerDelay || sendData
PrevMillis == 0)){
    sendDataPrevMillis = millis();

    // Get latest sensor readings
    temperature = bme.readTemperature();
    humidity = bme.readHumidity();
    pressure = bme.readPressure()/100.0F;

    // Send readings to database:
    sendFloat(tempPath, temperature);
    sendFloat(humPath, humidity);
    sendFloat(presPath, pressure);

  }
}
```

How the Code Works

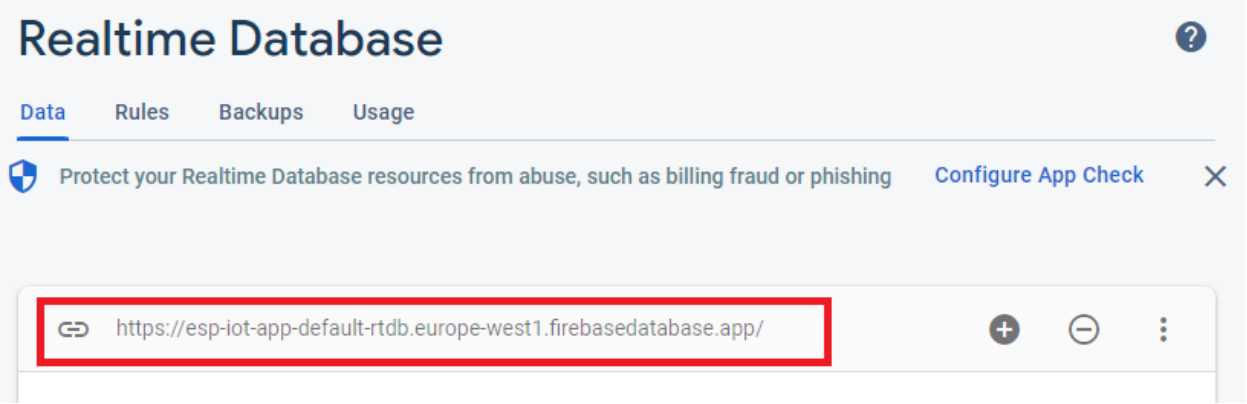
Continue reading to learn how the code works or skip to the Demonstration section.

Database URL

To interact with the database, you need to insert your database URL.

```
// Insert RTDB URLefine the RTDB URL
#define DATABASE_URL "REPLACE_WITH_YOUR_DATABASE_URL"
```

You can get your database URL by going to the Firebase console, selecting the Realtime Database tab, and copying the URL as highlighted below.



Variables' Declaration

The following variables will be used to save the nodes that we'll send the sensor readings. We'll update these variables later in the code when we get the user UID.

```
// Variables to save database paths
String databasePath;
String tempPath;
String humPath;
String presPath;
```

Then, create an `Adafruit_BME280` object called `bme`. This automatically creates a sensor object on the ESP32 or ESP8266 default I2C pins.

```
// BME280 sensor
Adafruit_BME280 bme; // I2C
```

The next variables will hold the temperature, humidity, and pressure readings from the sensor.

```
float temperature;
float humidity;
float pressure;
```

Delay Time

The `sendDataPrevMillis` and `timerDelay` variables are used to check the delay time between each send. In this example, we're setting the delay time to 3 minutes (18000

milliseconds). Once you test this project and check that everything is working as expected, we recommend increasing the delay time.

```
// Timer variables (send new readings every three minutes)
unsigned long sendDataPrevMillis = 0;
unsigned long timerDelay = 180000;
```

initBME()

The `initBME()` function initializes the BME280 library using the `bme` object created previously. Then, you should call this library in the `setup()`.

```
void initBME(){
  if (!bme.begin(0x76)) {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
    while (1);
  }
}
```

Not familiar with the BME280 sensor? Follow this tutorial.

Send Data to the Database

To store data at a specific node in the Firebase Realtime Database, you can use the following functions: `set`, `setInt`, `setFloat`, `setDouble`, `setString`, `setJSON`, `setArray`, `setBlob`, and `setFile`.

These functions return a boolean value indicating the success of the operation, which will be `true` if all of the following conditions are met: 1) server returns HTTP status code 200 and 2) the data types matched between request and response. Only `setBlob` and `setFile` functions that make a silent request to Firebase server, thus no payload response returned.

In our example, we'll send float variables, so we need to use the `setFloat()` function as follows.

```
Firebase.RTDB.setFloat(&fbdo, "DATABASE_NODE", VALUE)
```

The second argument refers to the database node (char variable) that we want to send the data. The third argument is the data we want to send (float variable).

As we'll need to send three float values, we created a function to do that—the `sendFloat()` function that accepts as argument the node path and the value.

```
void sendFloat(String path, float value){
   if (Firebase.RTDB.setFloat(&fbdo, path.c_str(), value)){
      Serial.print("Writing value: ");
      Serial.print (value);
      Serial.print(" on the following path: ");
      Serial.println(path);
      Serial.println("PASSED");
      Serial.println("PATH: " + fbdo.dataPath());
      Serial.println("TYPE: " + fbdo.dataType());
    }
    else {
      Serial.println("FAILED");
      Serial.println("REASON: " + fbdo.errorReason());
    }
}
```

Later, we'll call that function to send sensor readings.

setup()

In the `setup()`, call the `initBME()` function to initialize the BME280 sensor.

```
initBME();
```

Assign the database URL to the Firebase configuration object.

```
config.database_url = DATABASE_URL;
```

After getting the user UID, we can update the database node paths to include the user UID. That's what we do in the following lines.

```
databasePath = "/UsersData/" + uid;
// Define database path for sensor readings
// --> UsersData/<user_uid>/sensor/temperature
tempPath = databasePath + "/sensor/temperature";
// --> UsersData/<user_uid>/sensor/humidity
humPath = databasePath + "/sensor/humidity";
// --> UsersData/<user_uid>/sensor/pressure
presPath = databasePath + "/sensor/pressure";
```

loop()

In the `loop()`, check if it is time to send new readings:

```
if (Firebase.ready() && (millis() - sendDataPrevMillis > timerDelay || sendDataPr
evMillis == 0)){
  sendDataPrevMillis = millis();
```

If it is, get the latest sensor readings from the BME280 sensor and save them on the `temperature`, `humidity`, and `pressure` variables.

```
// Get latest sensor readings
temperature = bme.readTemperature();
humidity = bme.readHumidity();
pressure = bme.readPressure()/100.0F;
```

Finally, call the `sendFloat()` function to send the new readings to the database. Pass as arguments the node path and the value.

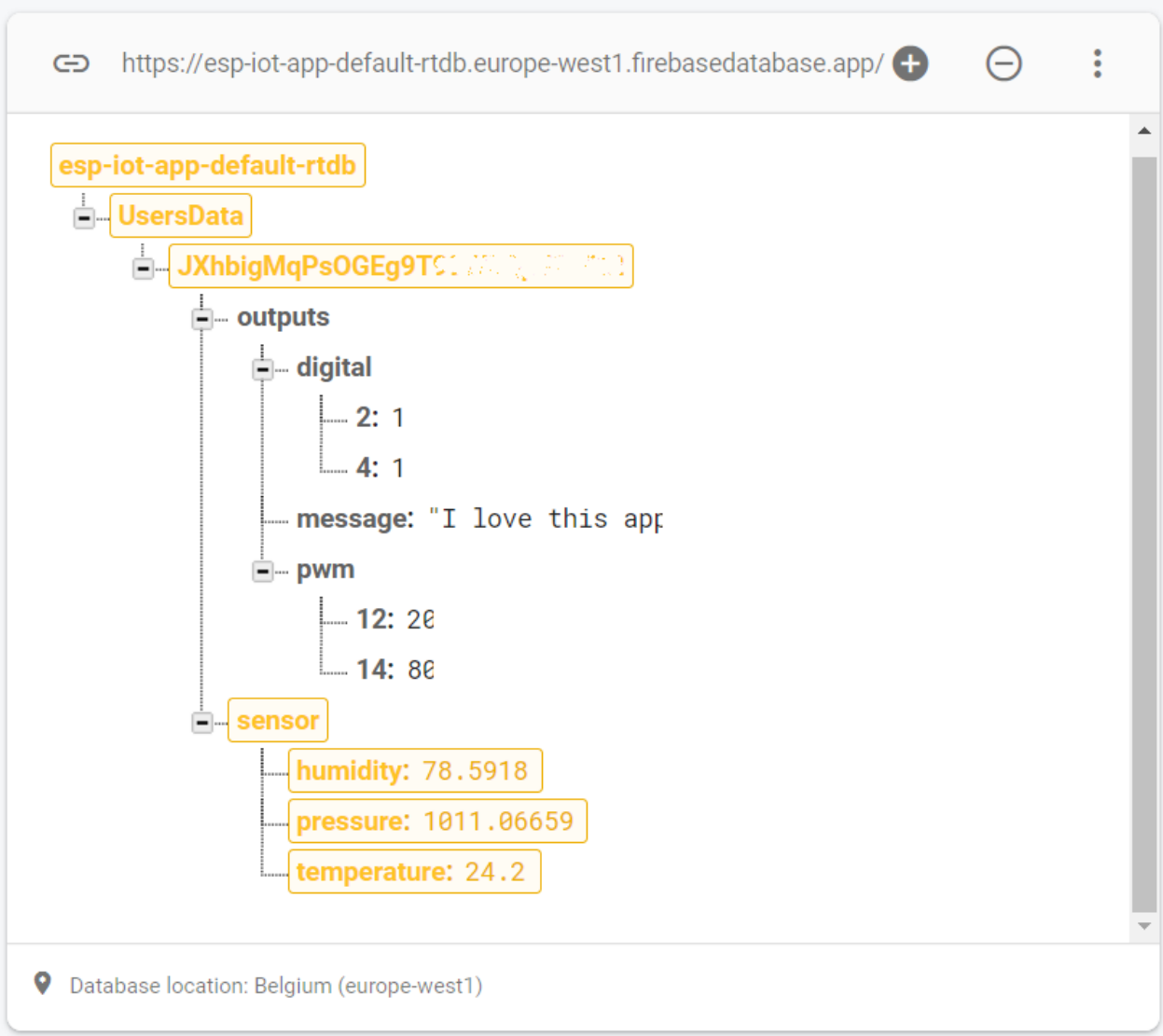
```
// Send readings to database:
sendFloat(tempPath, temperature);
sendFloat(humPath, humidity);
sendFloat(presPath, pressure);
```

Demonstration

After inserting your database URL on the sketch, upload it to your board. Then, open the Serial Monitor at a baud rate of 115200 and reset the board. You should get a success message, and the sensor readings will be published to the corresponding nodes.

```
.Token info: type = id token, status = ready
User UID: JXhbigMqPsOGEg
Writing value: 23.88 on the following path: /UsersData/JXhbigMqPsOGEg9T99W
PASSED
PATH: /UsersData/JXhbigMqPsOGEg9T
TYPE: float
Writing value: 79.67 on the following path: /UsersData/JXhbigMqPsOGEg9T
PASSED
PATH: /UsersData/JXhbigMqPsOGEg9T
TYPE: double
Writing value: 1011.08 on the following path: /UsersData/JXhbigMqPsOGE
PASSED
PATH: /UsersData/JXhbigMqPsOGEg9T99W
TYPE: double
```

On the Firebase console, go to your Realtime Database. It should insert new sensor readings every three minutes. Anytime a change occurs, it blinks in orange color.



Congratulations! You've successfully sent data to the Realtime Database.

In the next Unit, you'll learn how to set up listeners on specific paths so that you know whenever there are database changes on the GPIO states or on the message node (on the `UsersData/<user_uid>/outputs` path).

3.4 - Streaming Database: Listening for Changes

In this Unit, you'll learn how to listen for changes on specific database paths, which allows the ESP32/ESP8266 to know whenever there's a change in the database. This is useful for updating the GPIO states, adjusting the PWM duty cycle, or updating the OLED display message. So, we'll need to set up the following:

- Listening for changes in the GPIO states → change GPIO states accordingly;
- Listening for changes in the PWM values → adjust LEDs brightness;
- Listening for changes in the message value → update message on OLED display.

Streaming Database

In this section, you'll add the lines of code to listen for changes on the `UsersData/<user_uid>/outputs` path to detect whenever there's a change on the GPIO states, PWM values, or on the message.

Download the resources for this project:

⇒ Download Sketch folder (**Arduino IDE**)

⇒ Download Project folder ESP32 (**VS Code + PlatformIO**)

⇒ Download Project folder ESP8266 (**VS Code + PlatformIO**)

The new lines of code for streaming database are highlighted in light orange.

```
#include <Arduino.h>
#if defined(ESP32)
  #include <WiFi.h>
#elif defined(ESP8266)
  #include <ESP8266WiFi.h>
#endif
#include <Firebase_ESP_Client.h>
```

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

// Provide the token generation process info.
#include "addons/TokenHelper.h"
// Provide the RTDB payload printing info and other helper functions.
#include "addons/RTDBHelper.h"

// Insert your network credentials
#define WIFI_SSID "REPLACE_WITH_YOUR_SSID"
#define WIFI_PASSWORD " REPLACE_WITH_YOUR_PASSWORD"

// Insert Firebase project API Key
#define API_KEY "REPLACE_WITH_YOUR_API_KEY"

// Insert Firebase project API Key
#define API_KEY "REPLACE_WITH_YOUR_PROJECT_API_KEY"

// Insert Authorized Username and Corresponding Password
#define USER_EMAIL "REPLACE_WITH_THE_USER_EMAIL"
#define USER_PASSWORD "REPLACE_WITH_THE_USER_PASSWORD"
// Insert RTDB URLefine the RTDB URL
#define DATABASE_URL "REPLACE_WITH_YOUR_DATABASE_URL"

// Define Firebase objects
FirebaseData stream;
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;

// Variable to save USER UID
String uid;

// Variables to save database paths
String databasePath;
String tempPath;
String humPath;
String presPath;
String listenerPath;

// BME280 sensor
Adafruit_BME280 bme; // I2C
float temperature;
float humidity;
float pressure;

// Timer variables (send new readings every other minute)
unsigned long sendDataPrevMillis = 0;
unsigned long timerDelay = 120000;

// Initialize BME280
void initBME(){
  if (!bme.begin(0x76)) {
```

```
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
    while (1);
  }
}

// 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();
}

// Write float values to the database
void sendFloat(String path, float value){
  if (Firebase.RTDB.setFloat(&fbdo, path.c_str(), value)){
    Serial.print("Writing value: ");
    Serial.print (value);
    Serial.print(" on the following path: ");
    Serial.println(path);
    Serial.println("PASSED");
    Serial.println("PATH: " + fbdo.dataPath());
    Serial.println("TYPE: " + fbdo.dataType());
  }
  else {
    Serial.println("FAILED");
    Serial.println("REASON: " + fbdo.errorReason());
  }
}
```

```
// Callback function that runs on database changes
void streamCallback(FirebaseStream data){
   Serial.printf("stream path, %s\nevent path, %s\ndata type, %s\nevent type,
%s\n\n",
                 data.streamPath().c_str(),
                 data.dataPath().c_str(),
                 data.dataType().c_str(),
                 data.eventType().c_str());
  printResult(data); //see addons/RTDBHelper.h
  Serial.println();

  //This is the size of stream payload received (current and max value)
  //Max payload size is the payload size under the stream path since the stream
connected
  //and read once and will not update until stream reconnection takes place.
  //This max value will be zero as no payload received in case of ESP8266 which
  //BearSSL reserved Rx buffer size is less than the actual stream payload.
  Serial.printf("Received stream payload size: %d (Max. %d)\n\n",
data.payloadLength(), data.maxPayloadLength());
}
```

```
void streamTimeoutCallback(bool timeout){
  if (timeout)
    Serial.println("stream timeout, resuming...\n");
  if (!stream.httpConnected())
       Serial.printf("error  code:  %d,  reason:  %s\n\n",  stream.httpCode(),
stream.errorReason().c_str());
}

void setup(){
  Serial.begin(115200);

  // Init BME sensor, OLED, and WiFi
  initBME();
  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;

  // Assign the RTDB URL (required)
  config.database_url = DATABASE_URL;

  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;

  // 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);
  }

  // Print user UID
  uid = auth.token.uid.c_str();
  Serial.print("User UID: ");
  Serial.println(uid);

  // Update database path with user UID
  databasePath = "/UsersData/" + uid;

  // Define database path for sensor readings
```

```
  // --> UsersData/<user_uid>/sensor/temperature
  tempPath = databasePath + "/sensor/temperature";
  // --> UsersData/<user_uid>/sensor/humidity
  humPath = databasePath + "/sensor/humidity";
  // --> UsersData/<user_uid>/sensor/pressure
  presPath = databasePath + "/sensor/pressure";

  // Update database path for listening
  listenerPath = databasePath + "/outputs/";

  // Streaming (whenever data changes on a path)
  // Begin stream on a database path --> UsersData/<user_uid>/outputs
  if (!Firebase.RTDB.beginStream(&stream, listenerPath.c_str()))
    Serial.printf("stream begin error, %s\n\n", stream.errorReason().c_str());

  // Assign a calback function to run when it detects changes on the database
 Firebase.RTDB.setStreamCallback(&stream, streamCallback, streamTimeoutCallback);

  delay(2000);
}

void loop(){

  // Send new readings to database
  if  (Firebase.ready()  &&  (millis()  -  sendDataPrevMillis  >  timerDelay  ||
sendDataPrevMillis == 0)){
    sendDataPrevMillis = millis();

    // Get latest sensor readings
    temperature = bme.readTemperature();
    humidity = bme.readHumidity();
    pressure = bme.readPressure()/100.0F;

    // Send readings to database:
    sendFloat(tempPath, temperature);
    sendFloat(humPath, humidity);
    sendFloat(presPath, pressure);

  }
}
```

How the Code Works

First, you need to create a new `FirebaseData` object called `stream` to handle the data when there's a change on a specific database path.

```
FirebaseData stream;
```

Create a new variable for the listener path called `listenerPath`—this is the database path where we'll be listening for changes.

```
String listenerPath;
```

We'll update the database path later in the code once we get the user UID. Just to recap, the database path that we want to listen for changes is:

```
UsersData/<user_uid>/outputs
```

The `streamCallback()` function will be called whenever there's a change on a specific database path. We'll take a closer look at this function later.

```
void streamCallback(FirebaseStream data){
```

setup()

In the `setup()`, update the listener path after getting the UID.

```
listenerPath = databasePath + "/outputs/";
```

So, the listener path is:

```
UsersData/<user_ui>/outputs/
```

Anytime there is a change in that database node or subsequent child nodes, we'll trigger a callback function.

The following line of code starts a stream to listen for changes on the `listenerPath`.

```
if (!Firebase.RTDB.beginStream(&stream, listenerPath.c_str()))
```

Then, set a callback function to be triggered whenever there's a change on the `listenerPath`—the `streamCallback` function.

```
Firebase.RTDB.setStreamCallback(&stream, streamCallback, streamTimeoutCallback);
```

Let's now take a look at the `streamCallback` function.

```
// Callback function that runs on database changes
void streamCallback(FirebaseStream data){
  Serial.printf("stream path, %s\nevent path, %s\ndata type, %s\nevent type, %s\n\n",
                data.streamPath().c_str(),
                data.dataPath().c_str(),
```

```
                data.dataType().c_str(),
                data.eventType().c_str());
  printResult(data); //see addons/RTDBHelper.h
  Serial.println();

  //This is the size of stream payload received (current and max value)
  //Max payload size is the payload size under the stream path since the stream
connected
  //and read once and will not update until stream reconnection takes place.
  //This max value will be zero as no payload received in case of ESP8266 which
  //BearSSL reserved Rx buffer size is less than the actual stream payload.
    Serial.printf("Received   stream   payload   size:   %d   (Max.   %d)\n\n",
data.payloadLength(), data.maxPayloadLength());
}
```

When the `streamCallback()` function is triggered, an object called `data` of type `FirebaseStream` is passed automatically as an argument to that function. From that object, we can get the stream path, the data path (the full database path where the change occurred, including the value of the lowest child), the data type of that value, and the event type that triggered the stream.

This example simply prints the information received on the `data` stream object. In the next example, we'll see how we can get that data, save it into variables and update the corresponding GPIO states.

Demonstration

After inserting all the required credentials, upload the code to your board. After uploading, open the Serial Monitor at a baud rate of 115200 and press the RST button.

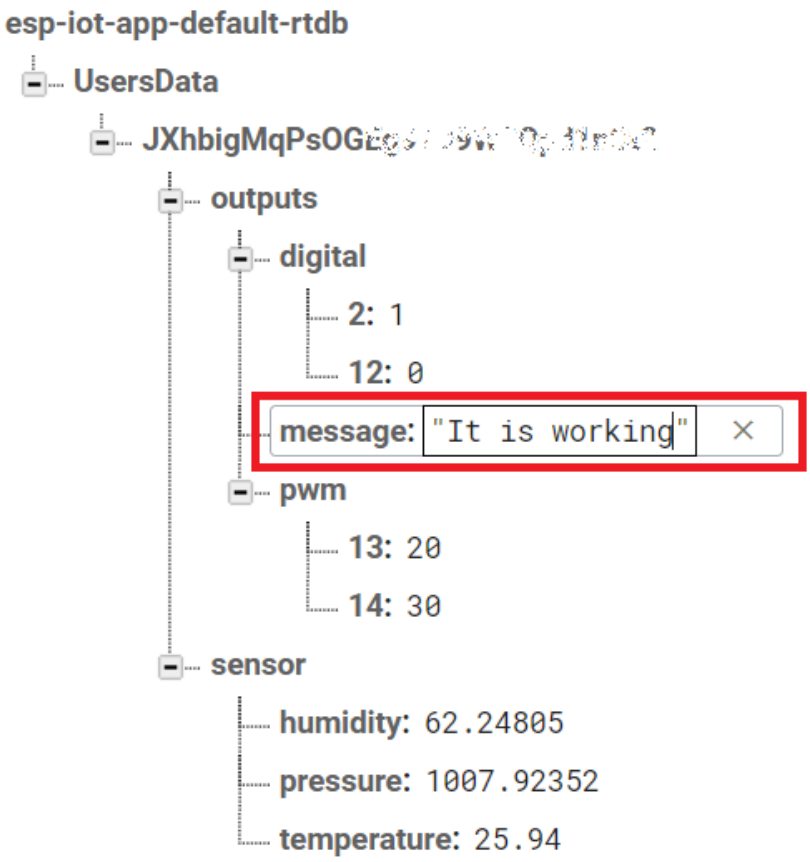
You'll see that when the ESP first runs and connects to the Realtime Database, it is as if it detected a change on the root (`/`) of the listener path. See the highlight in the picture below.

```
Connecting to WiFi .....192.168.1.112

Token info: type = id token, status = on request
Token info: type = id token, status = ready
Getting User UID
User UID: JXhbigMqPsOGEg9T99...
stream path, /UsersData/JXhbigMqPsO...../outputs/
event path, /
data type, json
event type, put

Pretty printed JSON data:
{
        "digital": {
                "2": 1,
                "12": 1
        },
        "message": "I love this app",
        "pwm": {
                "13": 20,
                "14": 80
        }
}
Iterate JSON data:

0, Type: object, Name: digital, Value: {"2":1,"12":1}
1, Type: object, Name: 2, Value: 1
2, Type: object, Name: 12, Value: 1
3, Type: object, Name: message, Value: "I love this app"
4, Type: object, Name: pwm, Value: {"13":20,"14":80}
5, Type: object, Name: 13, Value: 20
6, Type: object, Name: 14, Value: 80

Received stream payload size: 117 (Max. 117)
```

The `data` object contains an object of type `json` (see picture above) that includes all keys and values of the `listenerPath` child nodes. This allows us to know all the values saved on the listener path when the ESP first runs. Later, this is useful to update all the states using those values when the ESP first starts or when it resets.

Now, go to your Firebase console and select your project's Realtime Database. You can manually change any value on the `UsersData/<user_uid>/outputs` path. The ESP will detect that change. To change a value on the database, hover your mouse over the key values, select the field, and enter any other value.

For example, change the text on the message field.



The ESP will detect that change when you submit the new value, as shown in the Serial Monitor.

```
stream path, /UsersData/JXhbigMqP_ /outputs/
event path, /message
data type, string
event type, put

It is working

Received stream payload size: 61 (Max. 117)
```

It prints the event path, the data type (String), the event type, and the new submitted value.

Try changing the other fields and check that the ESP32 or ESP8266 can detect the changes.

```
stream path, /UsersData/JXhbigMqPsOC[illegible]/outputs/
event path, /digital/12
data type, int
event type, put

0

Received stream payload size: 50 (Max. 117)

stream path, /UsersData/JXhbigMqP[illegible]/outputs/
event path, /message
data type, string
event type, put

It is working

Received stream payload size: 61 (Max. 117)

stream path, /UsersData/JXhbigMqP[illegible]/outputs/
event path, /pwm/14
data type, int
event type, put

30

Received stream payload size: 47 (Max. 117)
```

Now that you know how to detect changes on the Realtime Database, you can change the GPIO states, PWM values, and message on the OLED display accordingly. That's what we'll do in the next section.

Streaming Database and Updating States

In this section, the ESP will update the GPIO states, PWM values, and the message on the OLED display accordingly to the data saved on the database. So, we'll need to set up the following:

- Listening for changes in the GPIO states → change GPIO states accordingly;
- Listening for changes in the PWM values → adjust LEDs brightness;
- Listening for changes in the message value → update message on OLED display.

Installing Required Libraries

Besides the libraries installed in the previous sections, you'll also need to install the following libraries to interface with the OLED display:

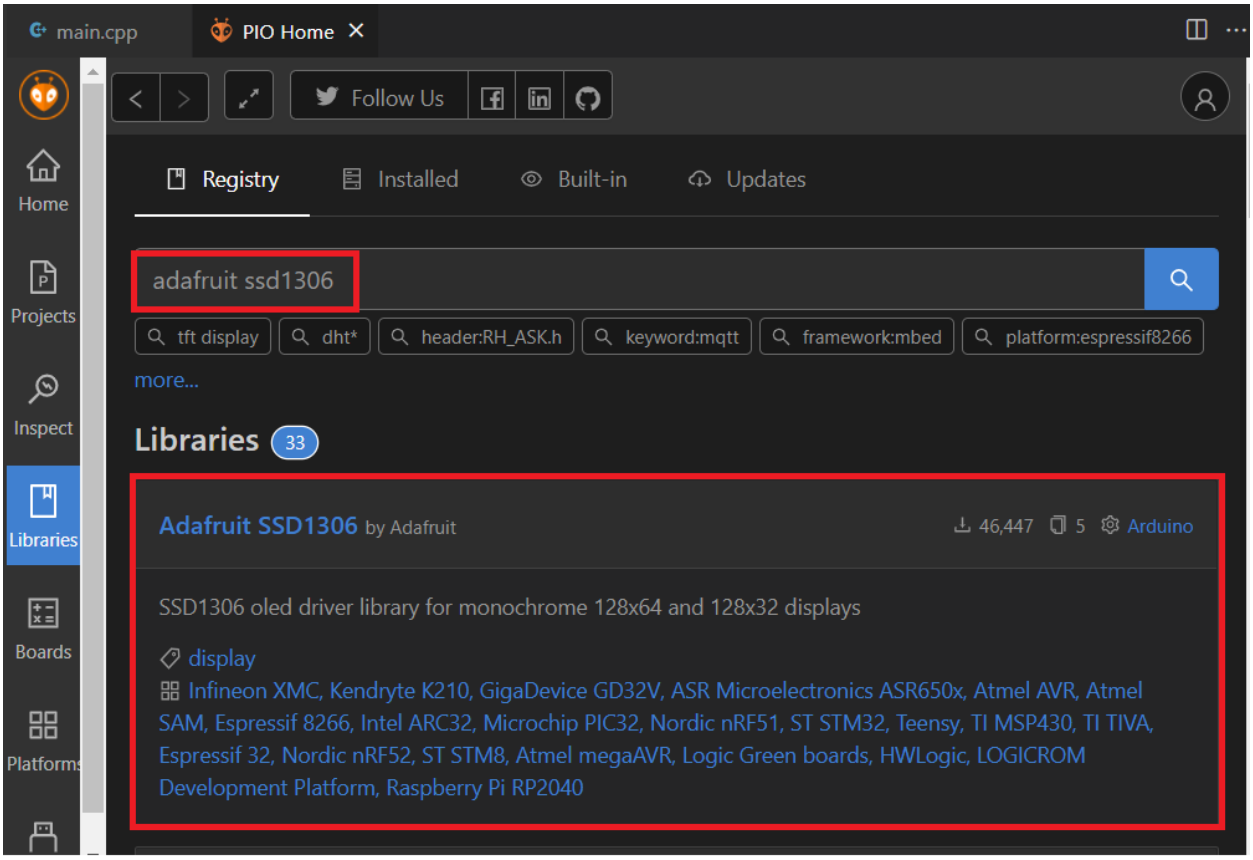
- Adafruit SSD1306
- Adafruit GFX

VS Code + PlatformIO

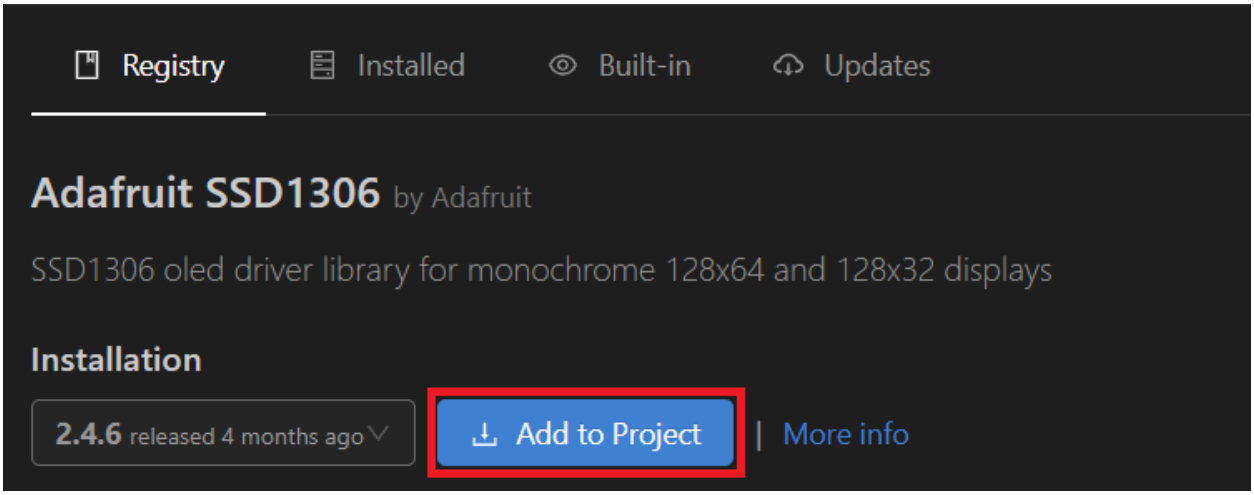
Follow the subsequent instructions if you're using VS Code.

1) With your project folder opened, click the **Home** icon to go to **PlatformIO Home**. Select the **Libraries** icon on the left sidebar.

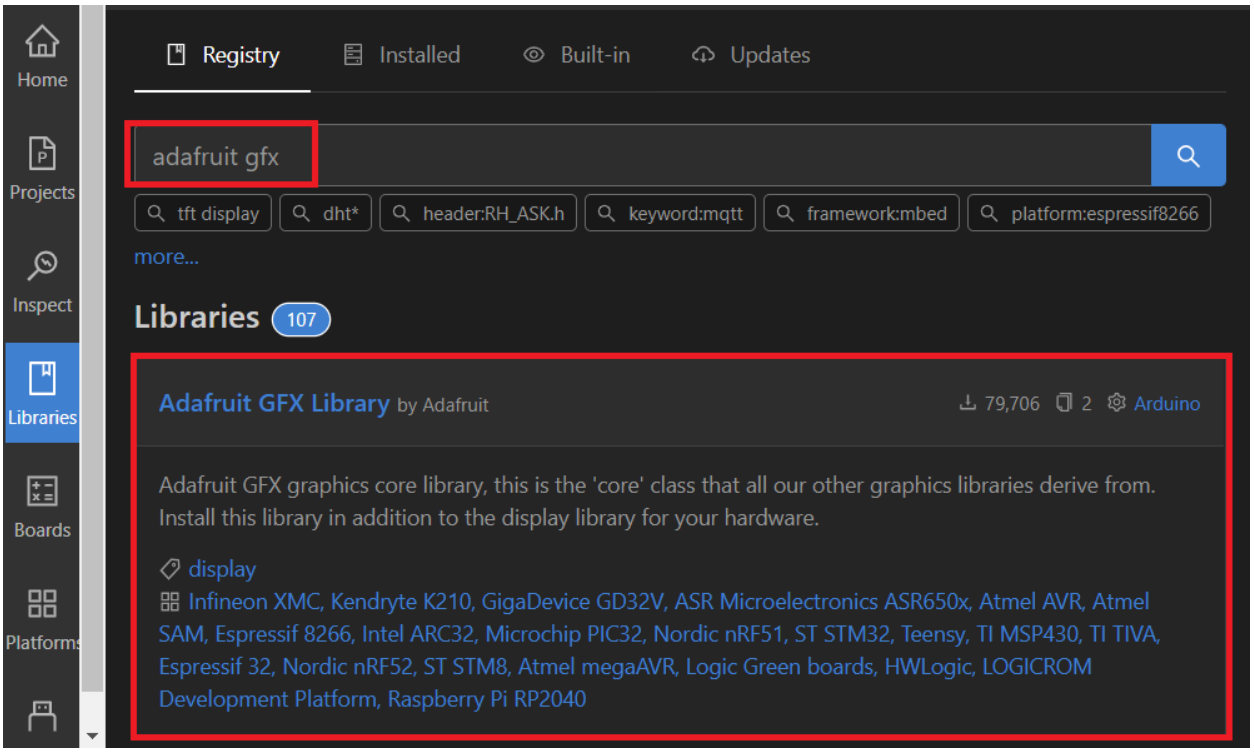
2) Search for the library you want to install—**Adafruit SSD1306**.



3) Select the library you want to include in your project. Then, click **Add to Project** and select the project where you want to use the library.



4) The Adafruit SSD1306 library also needs the Adafruit GFX library to work. Repeat steps **2)** and **3)** but search for "**adafruit GFX**" and install the Adafruit GFX library.

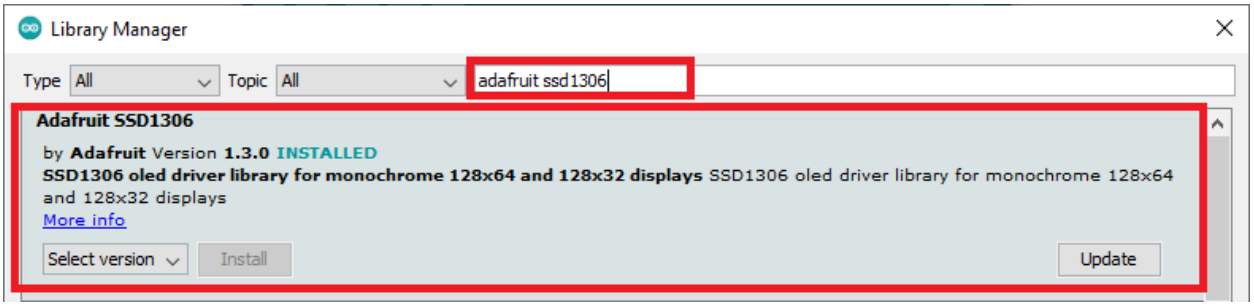


Arduino IDE

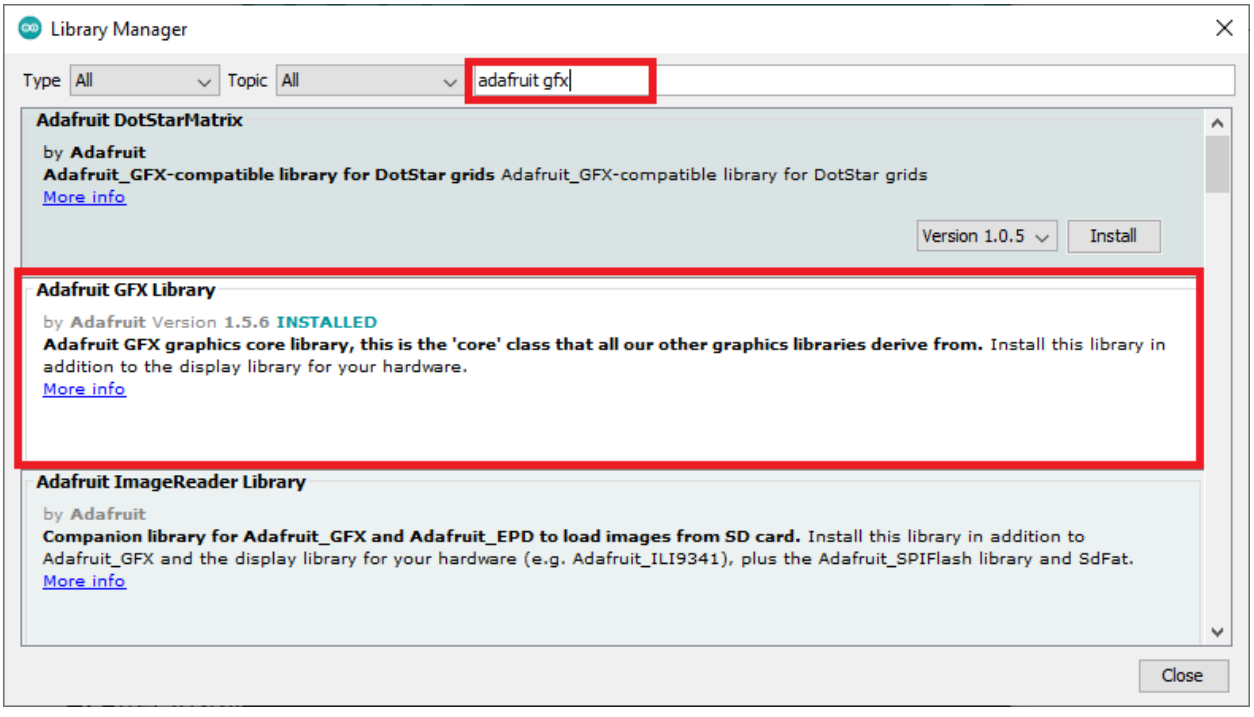
Follow the subsequent instructions if you're using Arduino IDE to install the SSD1306 library.

1) In the Arduino IDE, go to **Sketch** > **Include Library** > **Manage Libraries**. The Library Manager should open.

2) Search for "**adafruit ssd1306** " on the Search box and install the library.



3) Then, search for "**adafruit GFX**" in the search box. Scroll down to find the library and install it.



4) After installing the libraries, restart your Arduino IDE.

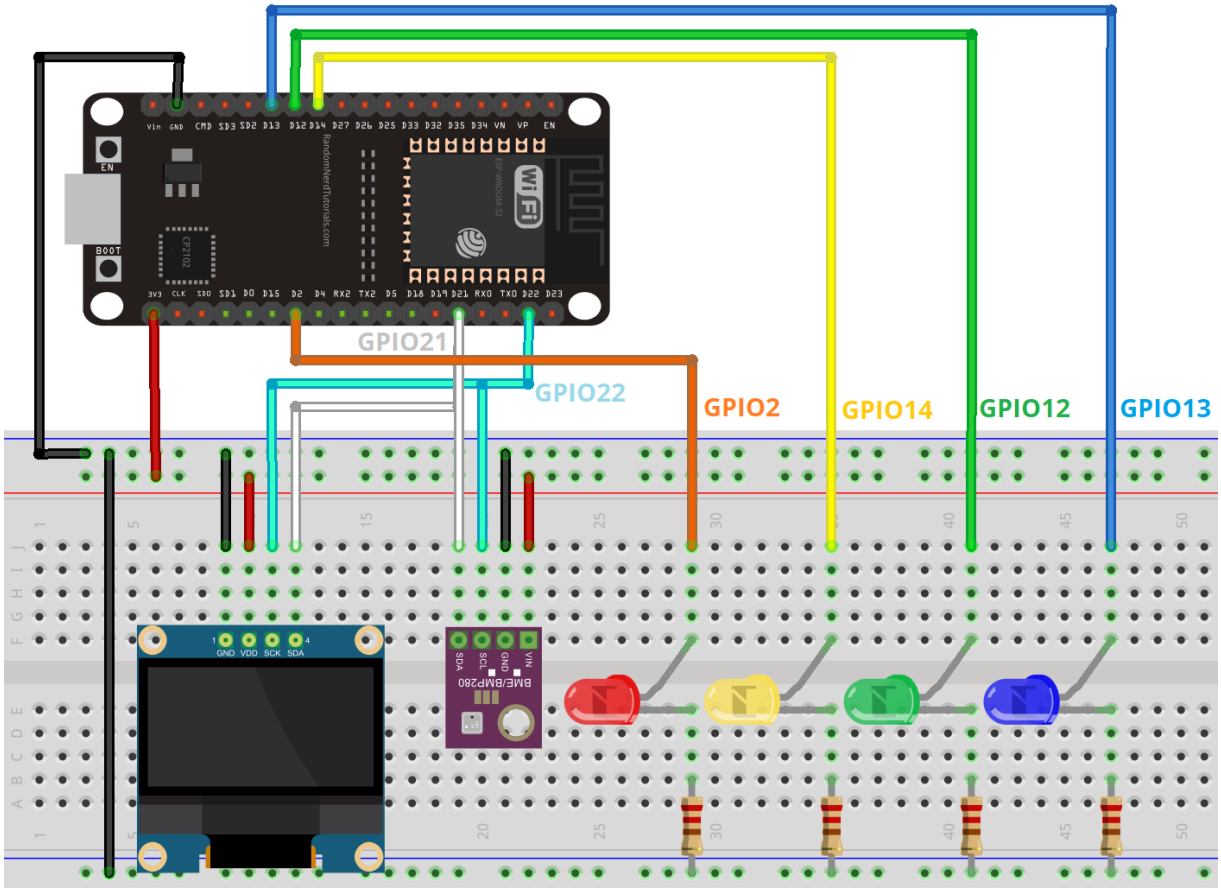
Schematic Diagram

Connect four LEDs and an OLED display to the circuit with the BME280 that you already have from previous units. The following table shows all the connections.

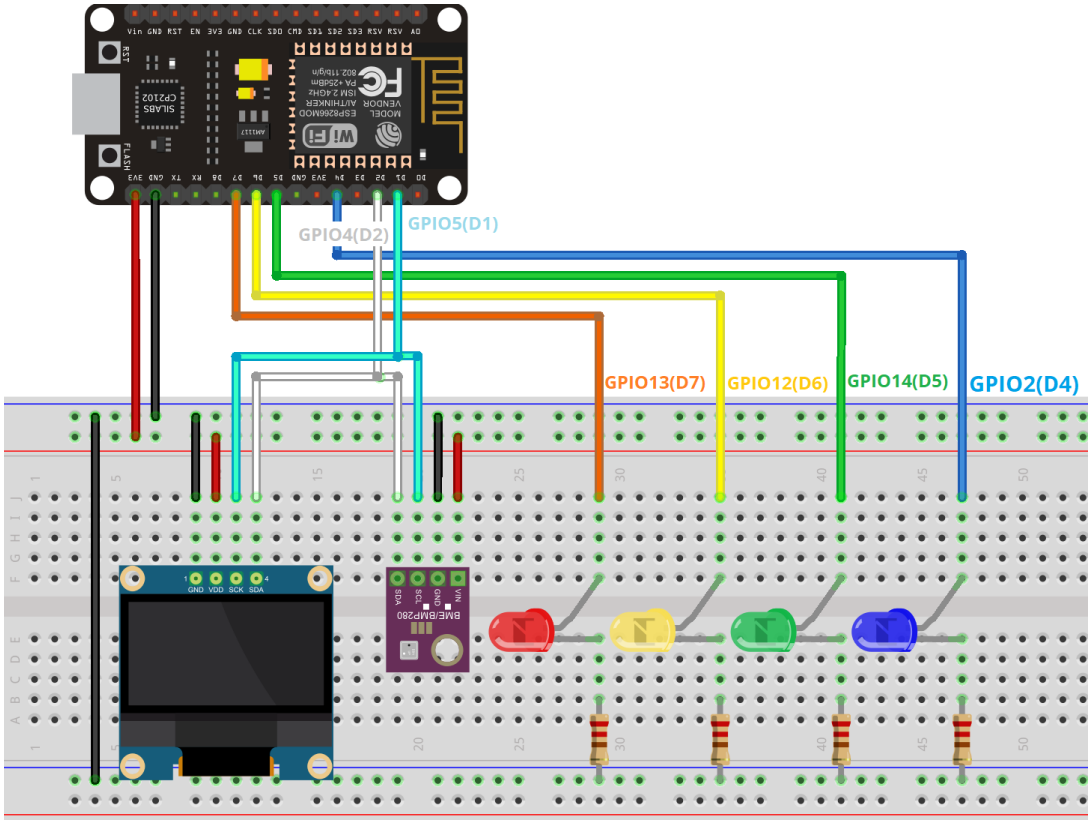
Component	ESP32	ESP8266
BME280	SDA (GPIO21), SCL (GPIO22)	SDA (GPIO4), SCL (GPIO5)
OLED Display	SDA (GPIO21), SCL (GPIO22)	SDA (GPIO4), SCL (GPIO5)
LED 1	GPIO2	GPIO2 (D4)
LED 2	GPIO12	GPIO12 (D6)
LED 3	GPIO13	GPIO13 (D7)
LED 4	GPIO14	GPIO14 (D5)

You can also follow the schematic diagrams below.

ESP32



ESP8266



Code - Streaming Database

We've added the needed lines to save the database's values and control the outputs and OLED accordingly. The new lines are highlighted in a light orange color.

The code for the ESP32 and ESP8266 boards are slightly different when it comes to output PWM signals. Apart from those changes, the sketches are identical.

⇒ Download Sketch folder ESP32 (**Arduino IDE**) *

⇒ Download Sketch folder ESP8266 (**Arduino IDE**) *

⇒ Download Project folder ESP32 (**VS Code + PlatformIO**)

⇒ Download Project folder ESP8266 (**VS Code + PlatformIO**)

* Use ESP Firebase Client Library above version 2.5.4

ESP32

Here's the code for the ESP32. You need to insert your network credentials (SSID and password), Firebase project API key, database URL, and the authorized user email and password to make it work.

```
#include <Arduino.h>
#include <WiFi.h>
#include <Firebase_ESP_Client.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

// Provide the token generation process info.
#include "addons/TokenHelper.h"
// Provide the RTDB payload printing info and other helper functions.
#include "addons/RTDBHelper.h"

// Insert your network credentials
#define WIFI_SSID "REPLACE_WITH_YOUR_SSID"
#define WIFI_PASSWORD "REPLACE_WITH_YOUR_PASSWORD"

// Insert Firebase project API Key
#define API_KEY "REPLACE_WITH_YOUR_PROJECT_API_KEY"

// Insert Authorized Username and Corresponding Password
#define USER_EMAIL "REPLACE_WITH_THE_USER_EMAIL"
#define USER_PASSWORD "REPLACE_WITH_THE_USER_PASSWORD"
// Insert RTDB URLefine the RTDB URL
#define DATABASE_URL "REPLACE_WITH_YOUR_DATABASE_URL"

// Define Firebase objects
FirebaseData stream;
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;

// Variable to save USER UID
String uid;

// Variables to save database paths
String databasePath;
String tempPath;
String humPath;
String presPath;
String listenerPath;

// BME280 sensor
```

```
Adafruit_BME280 bme; // I2C
float temperature;
float humidity;
float pressure;

//OLED Display
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

// Timer variables (send new readings every other minute)
unsigned long sendDataPrevMillis = 0;
unsigned long timerDelay = 120000;

// Declare outputs
const int output1 = 2;
const int output2 = 12;
const int slider1 = 13;
const int slider2 = 14;

// Variable to save input message
String message;

// Setting PWM properties
const int freq = 5000;
const int slider1Channel = 0;
const int slider2Channel = 1;
const int resolution = 8;

// Initialize BME280
void initBME(){
  if (!bme.begin(0x76)) {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
    while (1);
  }
}

// Initialize OLED
void initOLED(){
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;);
  }
  display.clearDisplay();
}

// Display messag on OLED display
void displayMessage(String message){
  display.clearDisplay();
  display.setTextSize(2);
  display.setCursor(0,0);
  display.setTextColor(WHITE);
  display.print(message);
  display.display();
```

```
}

// 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();
}

// Write float values to the database
void sendFloat(String path, float value){
  if (Firebase.RTDB.setFloat(&fbdo, path.c_str(), value)){
    Serial.print("Writing value: ");
    Serial.print (value);
    Serial.print(" on the following path: ");
    Serial.println(path);
    Serial.println("PASSED");
    Serial.println("PATH: " + fbdo.dataPath());
    Serial.println("TYPE: " + fbdo.dataType());
  }
  else {
    Serial.println("FAILED");
    Serial.println("REASON: " + fbdo.errorReason());
  }
}

// Callback function that runs on database changes
void streamCallback(FirebaseStream data){
  Serial.printf("stream path, %s\nevent path, %s\ndata type, %s\nevent type, %s\n
\n",
                data.streamPath().c_str(),
                data.dataPath().c_str(),
                data.dataType().c_str(),
                data.eventType().c_str());
  printResult(data); //see addons/RTDBHelper.h
  Serial.println();

  // Get the path that triggered the function
  String streamPath = String(data.dataPath());

  /* When it first runs, it is triggered on the root (/) path and returns a JSON
with all key
  and values of that path.So, we can get all values from the database and updated
 the GPIO
  states, PWM, and message on OLED*/
  if (data.dataTypeEnum() == fb_esp_rtdb_data_type_json){
    FirebaseJson *json = data.to<FirebaseJson *>();
    FirebaseJsonData result;
    if (json->get(result, "/digital/" + String(output1), false)){
```

```
      bool state = result.to<bool>();
      digitalWrite(output1, state);
    }
    if (json->get(result, "/digital/" + String(output2), false)){
      bool state = result.to<bool>();
      digitalWrite(output2, state);
    }
    if (json->get(result, "/message", false)){
      String message = result.to<String>();
      displayMessage(message);
    }
    if (json->get(result, "/pwm/" + String(slider1), false)){
      int pwmValue = result.to<int>();
      ledcWrite(slider1Channel, map(pwmValue, 0, 100, 0, 255));
    }
    if (json->get(result, "/pwm/" + String(slider2), false)){
      int pwmValue = result.to<int>();
      ledcWrite(slider2Channel, map(pwmValue, 0, 100, 0, 255));
    }
  }

  // Check for changes in the digital output values
  if(streamPath.indexOf("/digital/") >= 0){
    // Get string path lengh
    int stringLen = streamPath.length();
    // Get the index of the last slash
    int lastSlash = streamPath.lastIndexOf("/");
    // Get the GPIO number (it's after the last slash "/")
    // UsersData/<uid>/outputs/digital/<gpio_number>
    String gpio = streamPath.substring(lastSlash+1, stringLen);
    Serial.print("DIGITAL GPIO: ");
    Serial.println(gpio);
    // Get the data published on the stream path (it's the GPIO state)
    if(data.dataType() == "int") {
      int gpioState = data.intData();
      Serial.print("VALUE: ");
      Serial.println(gpioState);
      //Update GPIO state
      digitalWrite(gpio.toInt(), gpioState);
    }
    Serial.println();
  }

  // Check for changes in the slider values
  else if(streamPath.indexOf("/pwm/") >= 0){
    // Get string path lengh
    int stringLen = streamPath.length();
    // Get the index of the last slash
    int lastSlash = streamPath.lastIndexOf("/");
    // Get the GPIO number (it's after the last slash "/")
    // UsersData/<uid>/outputs/PWM/<gpio_number>
    int gpio = (streamPath.substring(lastSlash+1, stringLen)).toInt();
    Serial.print("PWM GPIO: ");
    Serial.println(gpio);
```

```
    // Get the PWM Value
    if(data.dataType() == "int"){
      int PWMValue = data.intData();
      Serial.print("VALUE: ");
      Serial.println(PWMValue);
      // Set the duty cycle (PWM) on the corresponding channel
      if (gpio == slider1){
        ledcWrite(slider1Channel, map(PWMValue, 0, 100, 0, 255));
      }
      if (gpio == slider2){
        ledcWrite(slider2Channel, map(PWMValue, 0, 100, 0, 255));
      }
    }
  }

  // Check for changes in the message
  else if (streamPath.indexOf("/message") >= 0){
    if (data.dataType() == "string") {
      message = data.stringData();
      Serial.print("MESSAGE: ");
      Serial.println(message);
      // Print on OLED
      displayMessage(message);
    }
  }
  //This is the size of stream payload received (current and max value)
  //Max payload size is the payload size under the stream path since the stream c
onnected
  //and read once and will not update until stream reconnection takes place.
  //This max value will be zero as no payload received in case of ESP8266 which
  //BearSSL reserved Rx buffer size is less than the actual stream payload.
  Serial.printf("Received stream payload size: %d (Max. %d)\n\n", data.payloadLen
gth(), data.maxPayloadLength());
}

void streamTimeoutCallback(bool timeout){
  if (timeout)
    Serial.println("stream timeout, resuming...\n");
  if (!stream.httpConnected())
    Serial.printf("error code: %d, reason: %s\n\n", stream.httpCode(), stream.err
orReason().c_str());
}

void setup(){
  Serial.begin(115200);

  // Init BME sensor, OLED, and WiFi
  initBME();
  initOLED();
  initWiFi();

  // Initialize Outputs
  pinMode(output1, OUTPUT);
  pinMode(output2, OUTPUT);
```

```
// configure PWM functionalitites
ledcSetup(slider1Channel, freq, resolution);
ledcSetup(slider2Channel, freq, resolution);
// attach the channels to the GPIOs to be controlled
ledcAttachPin(slider1, slider1Channel);
ledcAttachPin(slider2, slider2Channel);

// 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;

// Assign the RTDB URL (required)
config.database_url = DATABASE_URL;

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;

// 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);
}

// Print user UID
uid = auth.token.uid.c_str();
Serial.print("User UID: ");
Serial.println(uid);

// Update database path with user UID
databasePath = "/UsersData/" + uid;

// Define database path for sensor readings
// --> UsersData/<user_uid>/sensor/temperature
tempPath = databasePath + "/sensor/temperature";
// --> UsersData/<user_uid>/sensor/humidity
humPath = databasePath + "/sensor/humidity";
// --> UsersData/<user_uid>/sensor/pressure
presPath = databasePath + "/sensor/pressure";

// Update database path for listening
```

```
  listenerPath = databasePath + "/outputs/";

  // Streaming (whenever data changes on a path)
  // Begin stream on a database path --> UsersData/<user_uid>/outputs
  if (!Firebase.RTDB.beginStream(&stream, listenerPath.c_str()))
    Serial.printf("stream begin error, %s\n\n", stream.errorReason().c_str());

  // Assign a calback function to run when it detects changes on the database
  Firebase.RTDB.setStreamCallback(&stream, streamCallback, streamTimeoutCallback);

  delay(2000);
}

void loop(){
  // Send new readings to database
  if (Firebase.ready() && (millis() - sendDataPrevMillis > timerDelay || sendData
PrevMillis == 0)){
    sendDataPrevMillis = millis();

    // Get latest sensor readings
    temperature = bme.readTemperature();
    humidity = bme.readHumidity();
    pressure = bme.readPressure()/100.0F;

    // Send readings to database:
    sendFloat(tempPath, temperature);
    sendFloat(humPath, humidity);
    sendFloat(presPath, pressure);

  }
}
```

ESP8266

Use the following code if you're using an ESP8266 board. You need to insert your network credentials (SSID and password), Firebase project API key, database URL, and the authorized user email and password to make it work.

```
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <Firebase_ESP_Client.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

// Provide the token generation process info.
#include "addons/TokenHelper.h"
// Provide the RTDB payload printing info and other helper functions.
```

```
#include "addons/RTDBHelper.h"

// Insert your network credentials
#define WIFI_SSID "REPLACE_WITH_YOUR_SSID"
#define WIFI_PASSWORD " REPLACE_WITH_YOUR_PASSWORD"

// Insert Firebase project API Key
#define API_KEY "REPLACE_WITH_YOUR_PROJECT_API_KEY"

// Insert Authorized Username and Corresponding Password
#define USER_EMAIL "REPLACE_WITH_THE_USER_EMAIL"
#define USER_PASSWORD "REPLACE_WITH_THE_USER_PASSWORD"
// Insert RTDB URLefine the RTDB URL
#define DATABASE_URL "REPLACE_WITH_YOUR_DATABASE_URL"

// Define Firebase objects
FirebaseData stream;
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;

// Variable to save USER UID
String uid;

// Variables to save database paths
String databasePath;
String tempPath;
String humPath;
String presPath;
String listenerPath;

// BME280 sensor
Adafruit_BME280 bme; // I2C
float temperature;
float humidity;
float pressure;

//OLED Display
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

// Timer variables (send new readings every other minute)
unsigned long sendDataPrevMillis = 0;
unsigned long timerDelay = 120000;

// Declare outputs
const int output1 = 2;
const int output2 = 12;
const int slider1 = 13;
const int slider2 = 14;

// Variable to save input message
String message;
```

```
// Initialize BME280
void initBME(){
  if (!bme.begin(0x76)) {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
    while (1);
  }
}

// Initialize OLED
void initOLED(){
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;);
  }
  display.clearDisplay();
}

// Display messag on OLED display
void displayMessage(String message){
  display.clearDisplay();
  display.setTextSize(2);
  display.setCursor(0,0);
  display.setTextColor(WHITE);
  display.print(message);
  display.display();
}

// 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();
}

// Write float values to the database
void sendFloat(String path, float value){
  if (Firebase.RTDB.setFloat(&fbdo, path.c_str(), value)){
    Serial.print("Writing value: ");
    Serial.print (value);
    Serial.print(" on the following path: ");
    Serial.println(path);
    Serial.println("PASSED");
    Serial.println("PATH: " + fbdo.dataPath());
    Serial.println("TYPE: " + fbdo.dataType());
  }
  else {
    Serial.println("FAILED");
    Serial.println("REASON: " + fbdo.errorReason());
```

```
  }
}

// Callback function that runs on database changes
void streamCallback(FirebaseStream data){
  Serial.printf("stream path, %s\nevent path, %s\ndata type, %s\nevent type, %s\n
\n",
                data.streamPath().c_str(),
                data.dataPath().c_str(),
                data.dataType().c_str(),
                data.eventType().c_str());
  printResult(data); //see addons/RTDBHelper.h
  Serial.println();

  // Get the path that triggered the function
  String streamPath = String(data.dataPath());

  /* When it first runs, it is triggered on the root (/) path and returns a JSON
with all key
  and values of that path.So, we can get all values from the database and updated
 the GPIO
  states, PWM, and message on OLED*/
  if (data.dataTypeEnum() == fb_esp_rtdb_data_type_json){
    FirebaseJson *json = data.to<FirebaseJson *>();
    FirebaseJsonData result;
    if (json->get(result, "/digital/" + String(output1), false)){
      bool state = result.to<bool>();
      digitalWrite(output1, state);
    }
    if (json->get(result, "/digital/" + String(output2), false)){
      bool state = result.to<bool>();
      digitalWrite(output2, state);
    }
    if (json->get(result, "/message", false)){
      String message = result.to<String>();
      displayMessage(message);
    }
    if (json->get(result, "/pwm/" + String(slider1), false)){
      int pwmValue = result.to<int>();
      analogWrite(slider1, map(pwmValue, 0, 100, 0, 255));
    }
    if (json->get(result, "/pwm/" + String(slider2), false)){
      int pwmValue = result.to<int>();
      analogWrite(slider2, map(pwmValue, 0, 100, 0, 255));
    }
  }

  // Check for changes in the digital output values
  if(streamPath.indexOf("/digital/") >= 0){
    // Get string path lengh
    int stringLen = streamPath.length();
    // Get the index of the last slash
    int lastSlash = streamPath.lastIndexOf("/");
    // Get the GPIO number (it's after the last slash "/")
```

```
    // UsersData/<uid>/outputs/digital/<gpio_number>
    String gpio = streamPath.substring(lastSlash+1, stringLen);
    Serial.print("DIGITAL GPIO: ");
    Serial.println(gpio);
    // Get the data published on the stream path (it's the GPIO state)
    if(data.dataType() == "int") {
      int gpioState = data.intData();
      Serial.print("VALUE: ");
      Serial.println(gpioState);
      //Update GPIO state
      digitalWrite(gpio.toInt(), gpioState);
    }
    Serial.println();
  }

  // Check for changes in the slider values
  else if(streamPath.indexOf("/pwm/") >= 0){
    // Get string path lengh
    int stringLen = streamPath.length();
    // Get the index of the last slash
    int lastSlash = streamPath.lastIndexOf("/");
    // Get the GPIO number (it's after the last slash "/")
    // UsersData/<uid>/outputs/PWM/<gpio_number>
    int gpio = (streamPath.substring(lastSlash+1, stringLen)).toInt();
    Serial.print("PWM GPIO: ");
    Serial.println(gpio);
    // Get the PWM Value
    if(data.dataType() == "int"){
      int PWMValue = data.intData();
      Serial.print("VALUE: ");
      Serial.println(PWMValue);
      analogWrite(gpio, map(PWMValue, 0, 100, 0, 255));
    }
  }

  // Check for changes in the message
  else if (streamPath.indexOf("/message") >= 0){
    if (data.dataType() == "string") {
      message = data.stringData();
      Serial.print("MESSAGE: ");
      Serial.println(message);
      // Print on OLED
      displayMessage(message);
    }
  }
  //This is the size of stream payload received (current and max value)
  //Max payload size is the payload size under the stream path since the stream c
onnected
  //and read once and will not update until stream reconnection takes place.
  //This max value will be zero as no payload received in case of ESP8266 which
  //BearSSL reserved Rx buffer size is less than the actual stream payload.
  Serial.printf("Received stream payload size: %d (Max. %d)\n\n", data.payloadLen
gth(), data.maxPayloadLength());
}
```

```
void streamTimeoutCallback(bool timeout){
  if (timeout)
    Serial.println("stream timeout, resuming...\n");
  if (!stream.httpConnected())
    Serial.printf("error code: %d, reason: %s\n\n", stream.httpCode(), stream.err
orReason().c_str());
}

void setup(){
  Serial.begin(115200);

  // Init BME sensor, OLED, and WiFi
  initBME();
  initOLED();
  initWiFi();

  // Initialize Outputs
  pinMode(output1, OUTPUT);
  pinMode(output2, OUTPUT);
  pinMode(slider1, OUTPUT);
  pinMode(slider2, OUTPUT);

  // 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;

  // Assign the RTDB URL (required)
  config.database_url = DATABASE_URL;

  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;

  // 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);
  }

  // Print user UID
  uid = auth.token.uid.c_str();
```

```
  Serial.print("User UID: ");
  Serial.println(uid);

  // Update database path with user UID
  databasePath = "/UsersData/" + uid;

  // Define database path for sensor readings
  // --> UsersData/<user_uid>/sensor/temperature
  tempPath = databasePath + "/sensor/temperature";
  // --> UsersData/<user_uid>/sensor/humidity
  humPath = databasePath + "/sensor/humidity";
  // --> UsersData/<user_uid>/sensor/pressure
  presPath = databasePath + "/sensor/pressure";

  // Update database path for listening
  listenerPath = databasePath + "/outputs/";

  //Recommend for ESP8266 stream, adjust the buffer size to match your stream dat
a size
  stream.setBSSLBufferSize(2048 /* Rx in bytes, 512 - 16384 */, 512 /* Tx in byte
s, 512 - 16384 */);

  // Streaming (whenever data changes on a path)
  // Begin stream on a database path --> UsersData/<user_uid>/outputs
  if (!Firebase.RTDB.beginStream(&stream, listenerPath.c_str()))
    Serial.printf("stream begin error, %s\n\n", stream.errorReason().c_str());

  // Assign a calback function to run when it detects changes on the database
  Firebase.RTDB.setStreamCallback(&stream, streamCallback, streamTimeoutCallback);

  delay(2000);
}

void loop(){
  // Send new readings to database
  if (Firebase.ready() && (millis() - sendDataPrevMillis > timerDelay || sendData
PrevMillis == 0)){
    sendDataPrevMillis = millis();

    // Get latest sensor readings
    temperature = bme.readTemperature();
    humidity = bme.readHumidity();
    pressure = bme.readPressure()/100.0F;

    // Send readings to database:
    sendFloat(tempPath, temperature);
    sendFloat(humPath, humidity);
    sendFloat(presPath, pressure);
  }
}
```

How the Code Works

Continue reading to learn how the new lines of code work or skip to the Demonstration section.

Include Libraries

First, you need to include the `Adafruit_GFX` and `Adafruit_SSD1306` libraries to be able to interface with the OLED display.

```
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
```

OLED Display Declaration

Define the OLED display width and height. We're using a 0.96 inch OLED display (128x64 pixels).

```
//OLED Display
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
```

Then, create an `Adafruit_SSD1306` object called `display` with the defined width and height and on the default ESP I2C pins (`&Wire`) to interact with the OLED display.

```
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
```

Other Variables

Declare the output GPIOs. The `outpu1` and `output2` are for the digital GPIOs (ON/OFF).

```
const int output1 = 2;
const int output2 = 12;
```

The `slider1` and `slider2` variables are for the PWM GPIOs (LED brightness).

```
const int slider1 = 13;
const int slider2 = 14;
```

Create a variable called `message` that will hold the message we want to display on the OLED.

```
String message;
```

Set the PWM properties for the PWM pins—you don't need this if you're using an ESP8266.

```
// Setting PWM properties
const int freq = 5000;
const int slider1Channel = 0;
const int slider2Channel = 1;
const int resolution = 8;
```

Not familiar with PWM on the ESP32 or ESP8266? Read this ESP32 tutorial or this ESP8266 tutorial.

OLED Functions

The `initOLED()` function initializes the OLED display. We'll call this function in the `setup()` section to initialize the OLED.

```
// Initialize OLED
void initOLED(){
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;);
  }
  display.clearDisplay();
}
```

The `displayMessage()` function prints a message on the OLED display. It accepts as argument a String variable with the message you want to display.

```
void displayMessage(String message){
  display.clearDisplay();
  display.setTextSize(2);
  display.setCursor(0,0);
  display.setTextColor(WHITE);
  display.print(message);
  display.display();
}
```

setup()

In the `setup()`, call the `initOLED()` function to initialize the OLED display.

```
initOLED();
```

Define the digital output pins `output1` and `output2` as `OUTPUT`s using the `pinMode()` function.

```
pinMode(output1, OUTPUT);
pinMode(output2, OUTPUT);
```

Configure the PWM properties for the GPIOs—each GPIO is assigned a different PWM channel. Then, to control the signal on a specific GPIO, you'll refer to its channel. You don't need this if you're using an ESP8266 board.

```
// configure PWM functionalitites
ledcSetup(slider1Channel, freq, resolution);
ledcSetup(slider2Channel, freq, resolution);
// attach the channels to the GPIOs to be controlled
ledcAttachPin(slider1, slider1Channel);
ledcAttachPin(slider2, slider2Channel);
```

streamCallback() Function

Let's now take a look at the `streamCallback` function. We want to update the GPIO states, PWM values, or the OLED display message whenever there's a change in their corresponding nodes.

Let's take a look at the first lines of code of the `streamCallback` function. The following lines print some information about the stream path, listener path, data type of the new value, and the event type.

```
void streamCallback(FirebaseStream data){
  Serial.printf("stream path, %s\nevent path, %s\ndata type, %s\nevent type, %s\n\n",
                data.streamPath().c_str(),
                data.dataPath().c_str(),
                data.dataType().c_str(),
                data.eventType().c_str());
```

Previously, we've seen that when the ESP first runs, this function is triggered, and the data object contains a JSON object with all listener path keys and corresponding values. We want to get those values and update our peripherals accordingly.

First, we check if the stream is of type JSON.

```
if (data.dataTypeEnum() == fb_esp_rtdb_data_type_json){
```

If it is, we want to get the keys' values. The ESP Firebase Client library has its own integrated JSON library. The first thing we need to do is to convert the data stream to a `FirebaseJson` object as follows:

```
FirebaseJson *json = data.to<FirebaseJson *>();
```

To get the data saved on the keys, we need to create a `FirebaseJsonData` object (the `result` variable).

```
FirebaseJsonData result;
```

Then, we can use the `get()` method on the `FirebaseJson` object to get the data we want. That method accepts as arguments a variable of type `FirebaseJsonData` (`result`) where it will save the data, and the database path. The last argument is of type bool (`true`=prettify, `fase`=don't prettify). It returns a boolean value indicating the success of the operation.

In the following lines, we get the value saved on the `/digital/2` path. The value returned can be either 0 or 1, so need to convert it to a boolean value and save it in a boolean variable (`state`).

```
if (json->get(result, "/digital/" + String(output1), false)){
  bool state = result.to<bool>();
```

Then, we update the GPIO 2 (`output1` variable) state using the `digitalWrite()` function.

```
digitalWrite(output1, state);
```

We follow a similar procedure for the other output (`output2` variable).

```
if (json->get(result, "/digital/" + String(output2), false)){
  bool state = result.to<bool>();
  digitalWrite(output2, state);
}
```

In the case of the `message` variable, we need to convert the data to a String variable. Then, we call the `displayMessage()` function we created previously to display the message on the OLED display.

```
if (json->get(result, "/message", false)){
  String message = result.to<String>();
  displayMessage(message);
}
```

Now, we need to get the values for the LEDs that are controlled using PWM (`slider1` and `slider2`).

The following lines get the pwm value for `slider1` and save it in the `pwmValue` variable. Then, it calls the `ledcWrite()` function to adjust the PWM channel dutty cycle. The `ledcWrite()` function accepts as arguments the PWM channel and the duty cycle. The duty cycle is a value between 0 and 255. That's why we use the `map()` function to convert from the 0-100 to the 0-255 range.

```
if (json->get(result, "/pwm/" + String(slider1), false)){
  int pwmValue = result.to<int>();
  ledcWrite(slider1Channel, map(pwmValue, 0, 100, 0, 255));
}
```

If you're using and ESP8266 board, you can use the `analogWrite()` function instead. It accepts as arguments the GPIO you want to control and the PWM duty cycle.

```
if (json->get(result, "/pwm/" + String(slider1), false)){
  int pwmValue = result.to<int>();
  analogWrite(slider1, map(pwmValue, 0, 100, 0, 255));
}
```

We follow a similar procedure for the other LED we want to control with PWM (`slider2`).

```
if (json->get(result, "/pwm/" + String(slider2), false)){
  int pwmValue = result.to<int>();
  ledcWrite(slider2Channel, map(pwmValue, 0, 100, 0, 255));
}
```

These previous lines of code update the peripherals state when the ESP first runs or when it resets.

Now, we need to add the lines of code to get a value on a specific node whenever there's a change. When you insert a new value on the database, we've seen that it returns something as follows.

```
stream path, /UsersData/JXhbigMqP[illegible]/outputs/
event path, /message
data type, string
event type, put

It is working

Received stream payload size: 61 (Max. 117)

stream path, /UsersData/JXhbigMqP[illegible]/outputs/
event path, /pwm/14
data type, int
event type, put

30

Received stream payload size: 47 (Max. 117)
```

So, by getting the event path, we know where the change occurred and what we need to update.

We save the event path on the `streamPath` variable.

```
String streamPath = String(data.dataPath());
```

For example, let's see how to detect changes on the `outputs/digital/2` path to detect changes on the GPIO 2 state.

First, we check if the `streamPath` contains "`/digital/`" by using the `indexOf()` function. The `indexOf()` function locates a character or String within another String.

It returns the start index of the substring if found or -1 if not found. So if it is 0 or a positive number, we know the event was triggered on the `/digital` child node.

Then, we know that the GPIO number is the number after the last slash (`/`). So, to get the GPIO number, we simply cut the `streamPath` string to get that value. We save the GPIO number on the `gpio` variable.

```
// Get string path lengh
int stringLen = streamPath.length();
// Get the index of the last slash
int lastSlash = streamPath.lastIndexOf("/");
// Get the GPIO number (it's after the last slash "/")
// UsersData/<uid>/outputs/digital/<gpio_number>
String gpio = streamPath.substring(lastSlash+1, stringLen);
```

Then, we get the data published on the stream path and save it in the `gpioState` variable as follows:

```
// Get the data published on the stream path (it's the GPIO state)
if(data.dataType() == "int") {
  int gpioState = data.intData();
  Serial.print("VALUE: ");
  Serial.println(gpioState);
```

Finally, we update the GPIO state:

```
digitalWrite(gpio.toInt(), gpioState);
```

We follow a similar procedure for the PWM values, but we adjust the LEDs' duty cycle accordingly.

```
// Check for changes in the slider values
else if(streamPath.indexOf("/pwm/") >= 0){
  // Get string path lengh
  int stringLen = streamPath.length();
  // Get the index of the last slash
  int lastSlash = streamPath.lastIndexOf("/");
  // Get the GPIO number (it's after the last slash "/")
  // UsersData/<uid>/outputs/PWM/<gpio_number>
  int gpio = (streamPath.substring(lastSlash+1, stringLen)).toInt();
  Serial.print("PWM GPIO: ");
  Serial.println(gpio);
  // Get the PWM Value
  if(data.dataType() == "int"){
    int PWMValue = data.intData();
```

```
    Serial.print("VALUE: ");
    Serial.println(PWMValue);
    analogWrite(gpio, map(PWMValue, 0, 100, 0, 255));
  }
}
```

Finally, the following lines detect whenever there's a change on the `/message` path and update the OLED display with the new message.

```
else if (streamPath.indexOf("/message") >= 0){
  if (data.dataType() == "string") {
    message = data.stringData();
    Serial.print("MESSAGE: ");
    Serial.println(message);
    // Print on OLED
    displayMessage(message);
  }
}
```

Demonstration

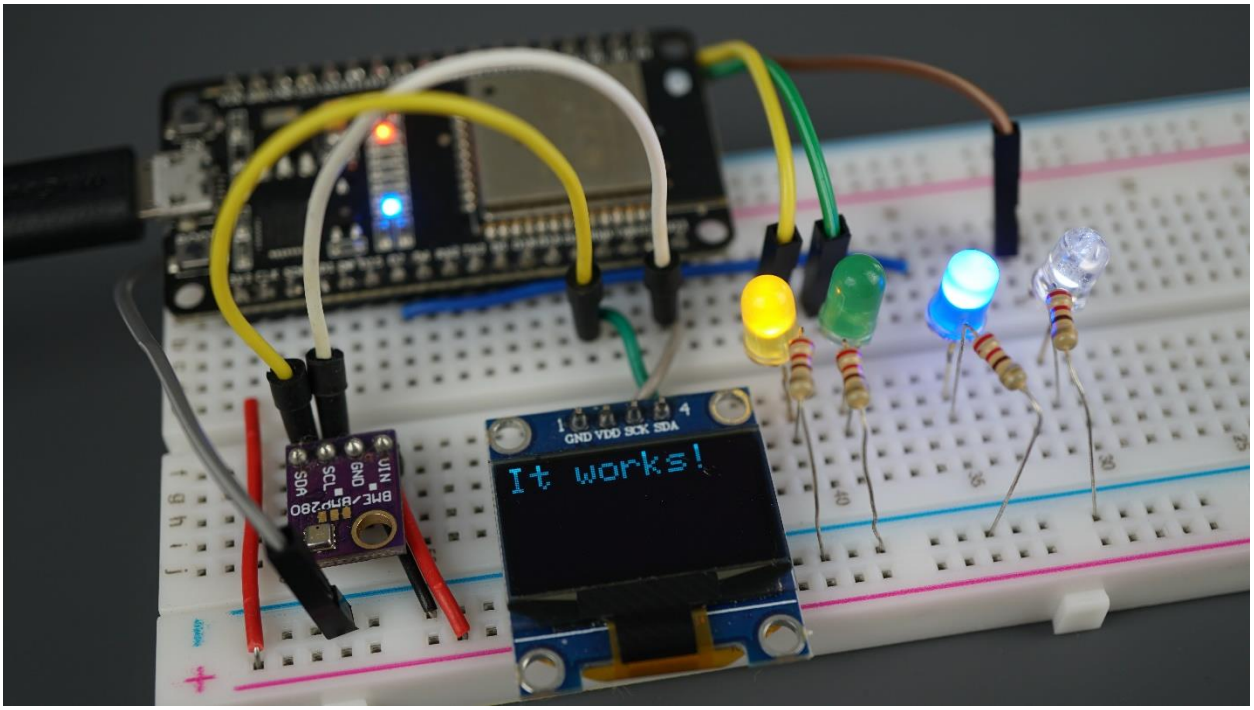
The code is now complete. Let's test if everything is working as expected.

After inserting your credentials, upload the code to your board. Then, open the Serial Monitor at a baud rate of 115200 and reset the board.

First, it will load the data saved on the database, and it should update the LEDs states and the message on the OLED display.

Then, go to your Firebase Realtime Database and manually change the digital values for GPIO 2 and 12, the message, and the PWM values for GPIOs 13 and 14.

The ESP should detect the changes almost instantly. It displays all the information related to the event on the Serial Monitor, and it updates the states of all your peripherals.



Congratulations! The ESP32/ESP8266 code is complete. Your ESP boards are communicating with the Firebase Realtime Database: listening for changes and sending sensor readings.

Finally, instead of changing those values manually on the database to control the ESP GPIOs, we'll build a web app with a nice interface with buttons and sliders to control the ESP via a web page that you can access from anywhere.

Let the ESP run this code and proceed to the next section to create your Firebase Web App.

PART 4

Creating a Firebase Web App

In this Part, you'll create a Firebase web app to control and monitor your ESP32/ESP8266 board from anywhere. The web app contains buttons and sliders to control the GPIOs, an input field to write a text message to the OLED, and a table to display sensor readings. This web app makes the bridge between the Realtime Database and the ESP32 and ESP8266 boards.

4.1 - Installing Required Software

In this section, you'll install the required software to start building your Firebase web app that you can access from anywhere to monitor and control your ESP32 or ESP8266 boards.

To create your Firebase web app, you need to install the following software if you haven't already:

- Visual Studio Code
- Node.JS LTS version
- Node.js Extension Pack (VS Code)
- Firebase Tools

Continue reading to learn how to install all the necessary tools.

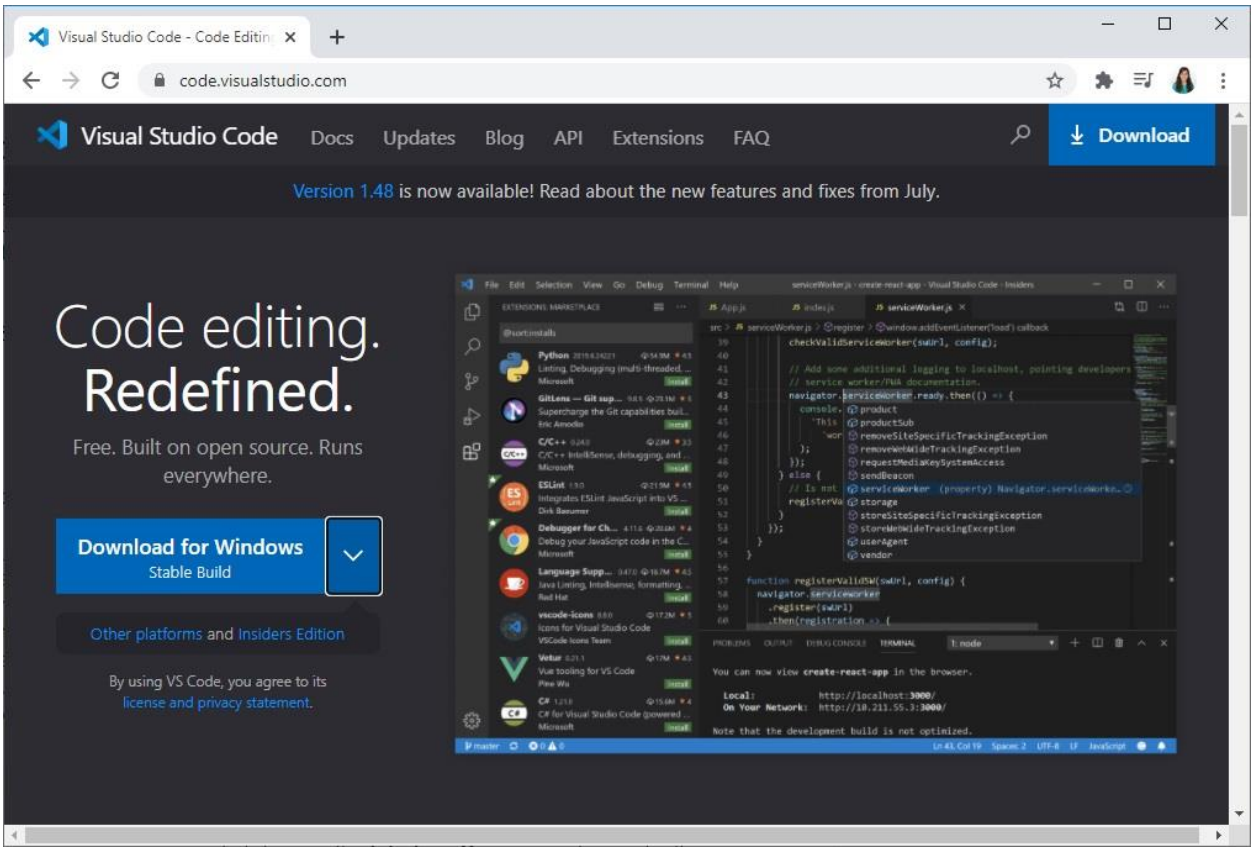
Installing Visual Studio Code

To create your Firebase app, you need to install Visual Studio Code. If you didn't install VS Code previously, now that's the time to do it. You can still program your ESP32 and ESP8266 boards using Arduino IDE if you prefer, but you need VS Code for building the Firebase web app and deploying it.

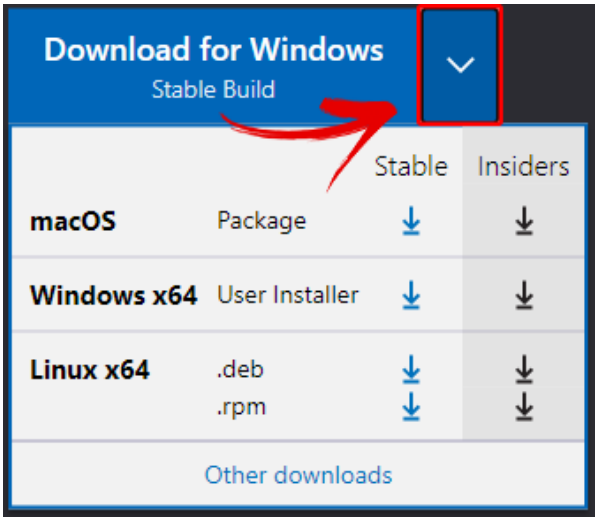
Follow the proper instructions depending on your operating system.

Windows

1) Go to https://code.visualstudio.com/ and download the stable build for your operating system.

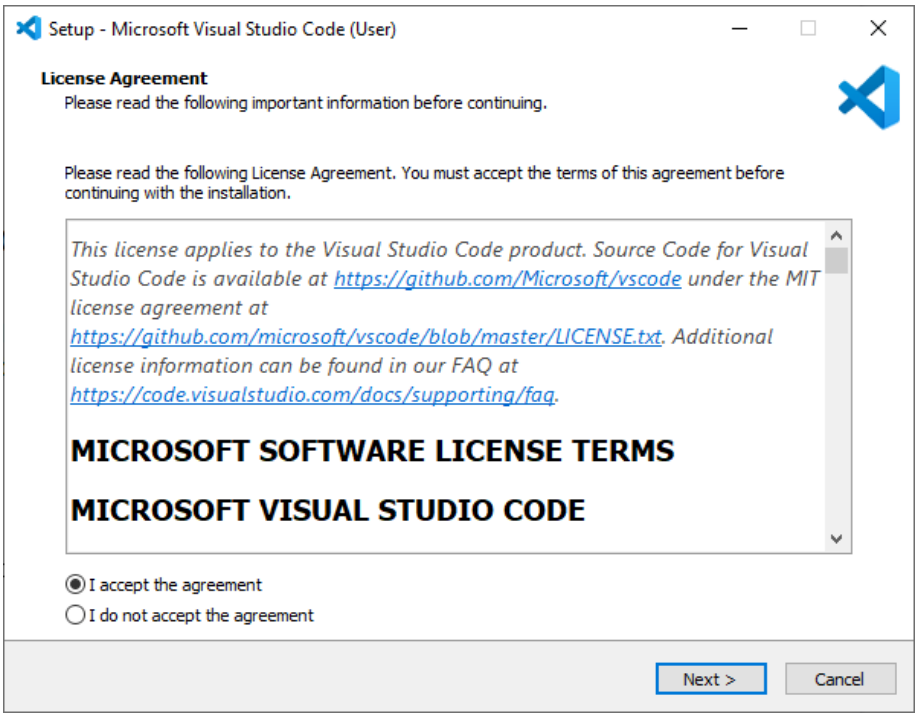


You can click on the arrow to select the correct build for your OS.

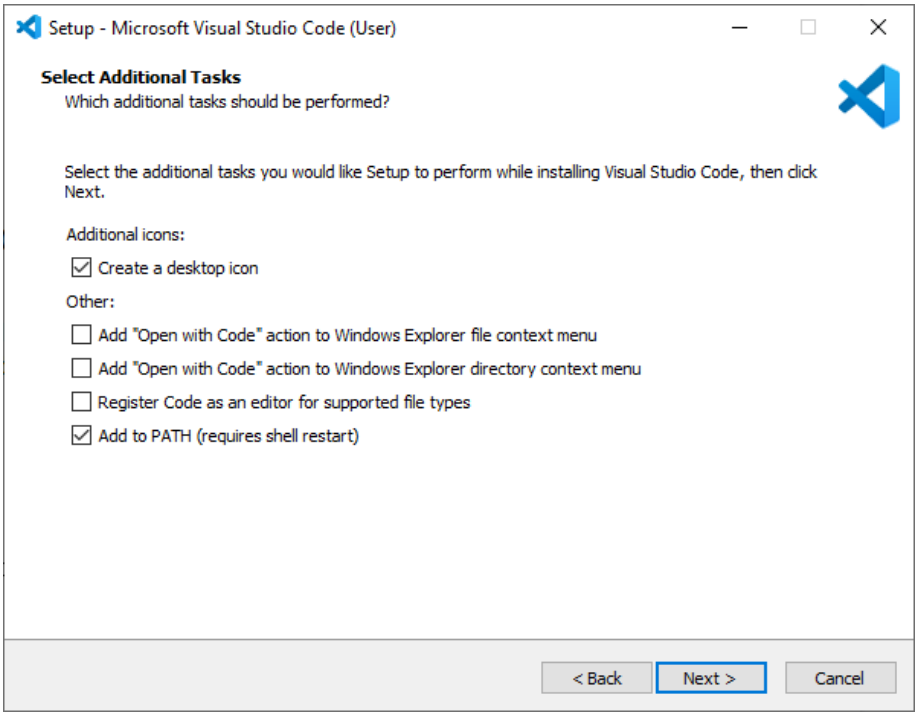


2) Click on the installation wizard to start and follow all the steps to complete the installation.

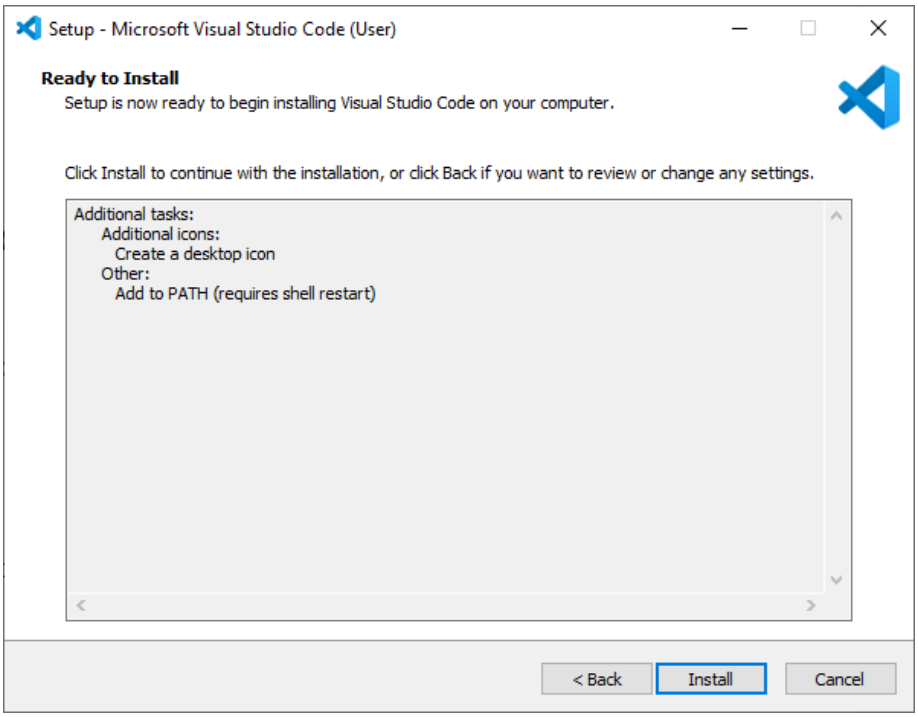
3) Accept the agreement and click the **Next** button.



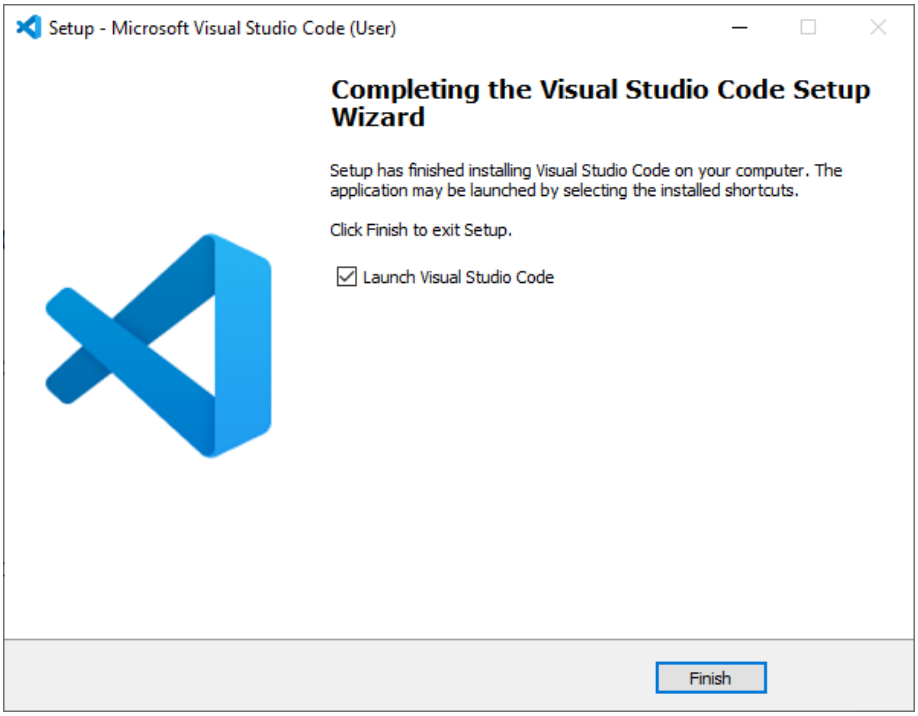
4) Select the following options and click **Next**.



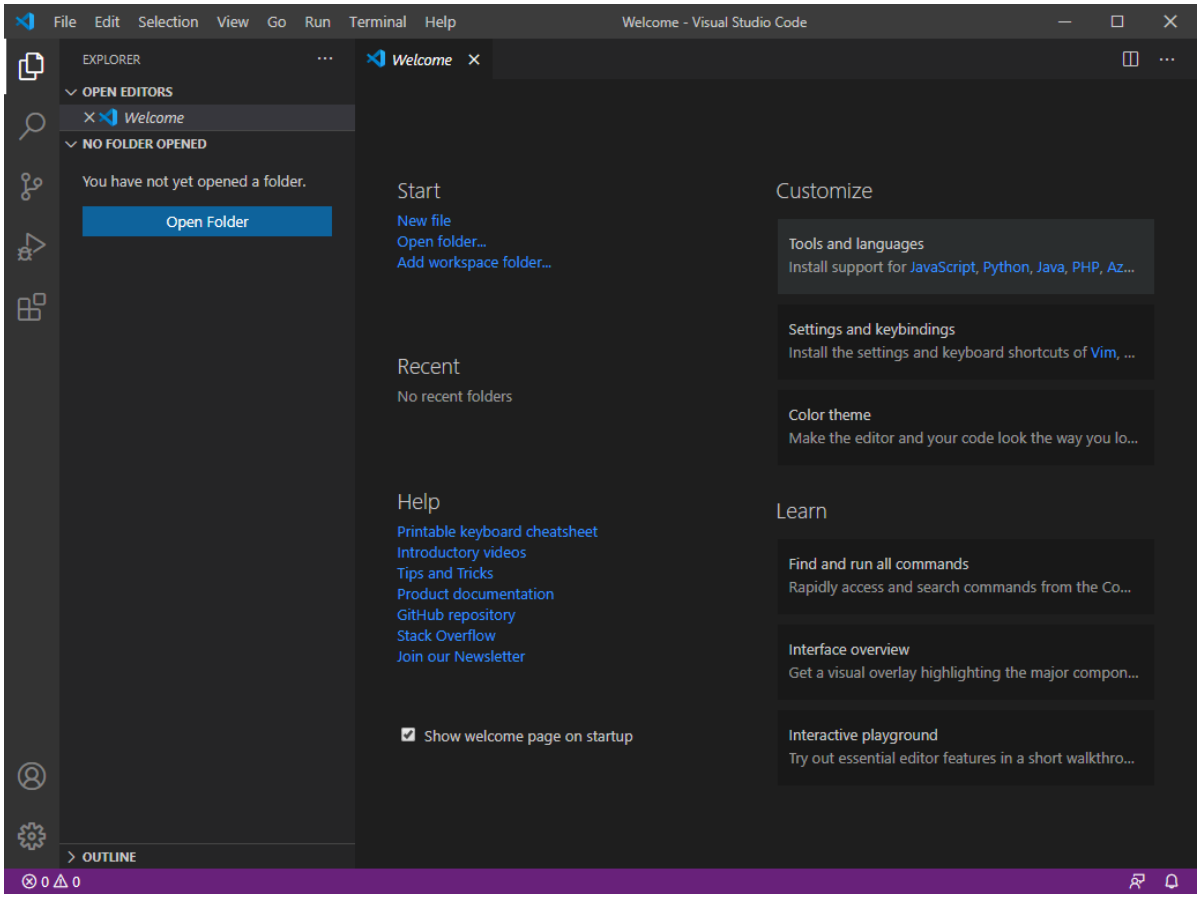
5) Click the **Install** button.



6) Finally, click **Finish** to complete the installation.



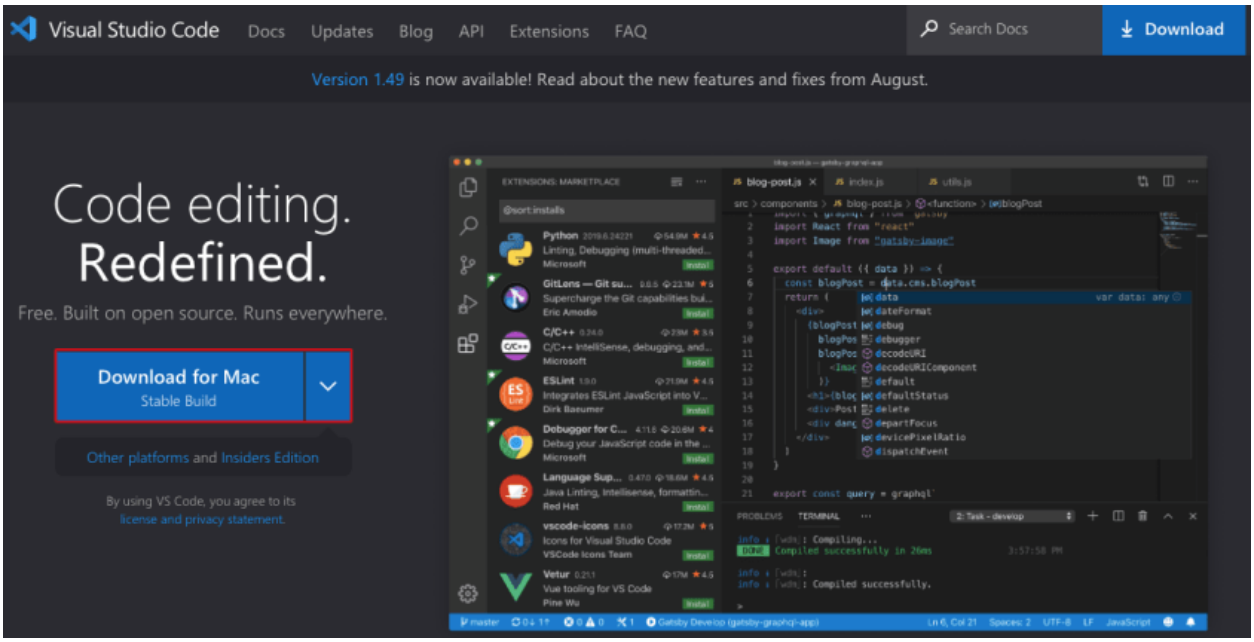
7) Open VS Code, and you'll be greeted by a Welcome tab with the newest version's released notes.



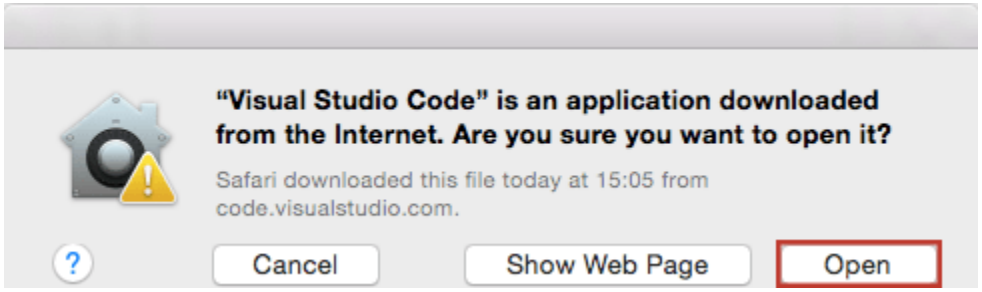
That's it. You installed Visual Studio Code successfully on your Windows computer.

Mac OS X

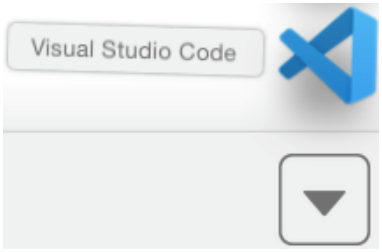
1) Go to https://code.visualstudio.com/ and download the stable build for your operating system (Mac OS X).



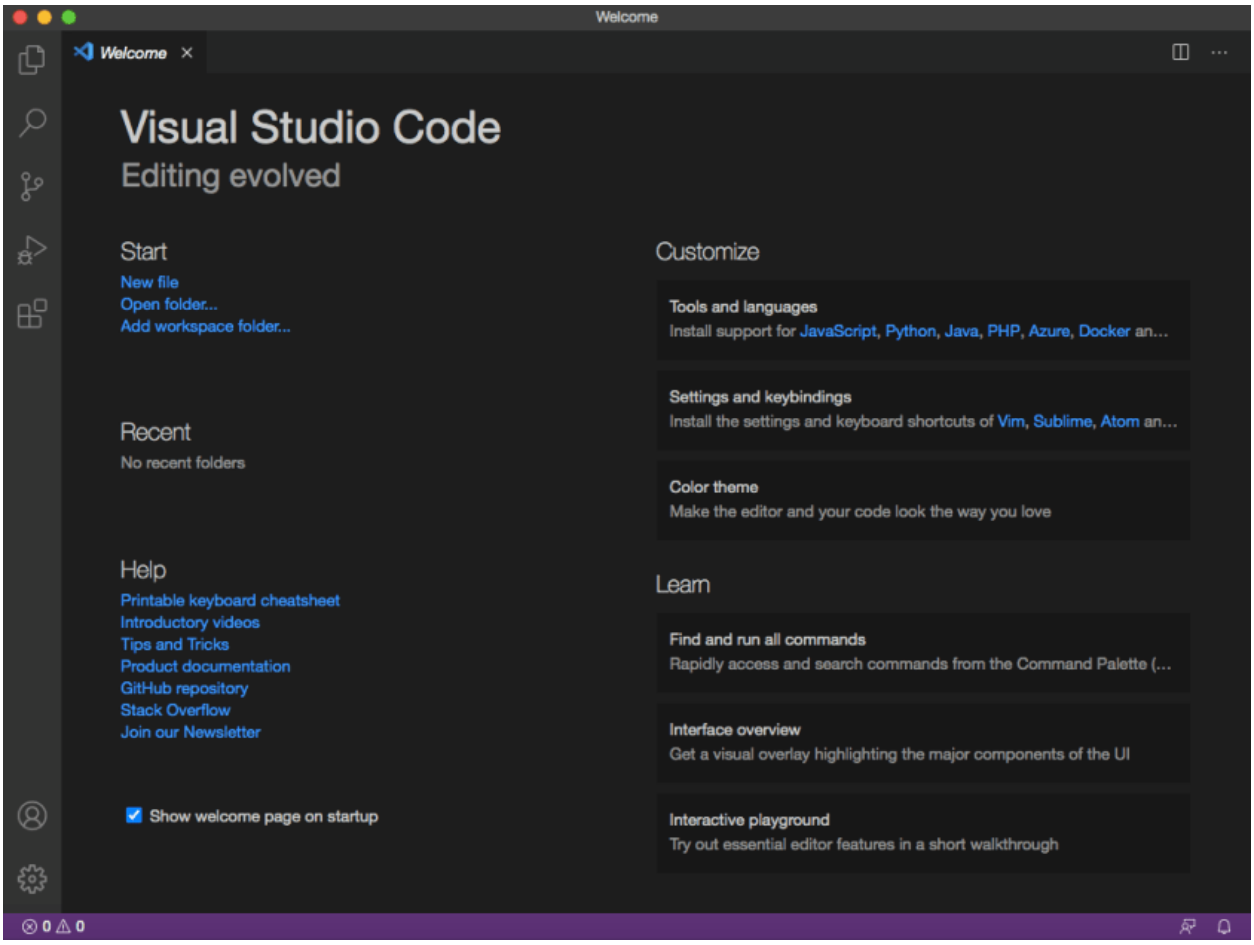
2) After downloading the Visual Studio Code application file, you'll be prompted with the following message. Press the **Open** button.



Or open your *Downloads* folder and open Visual Studio Code.



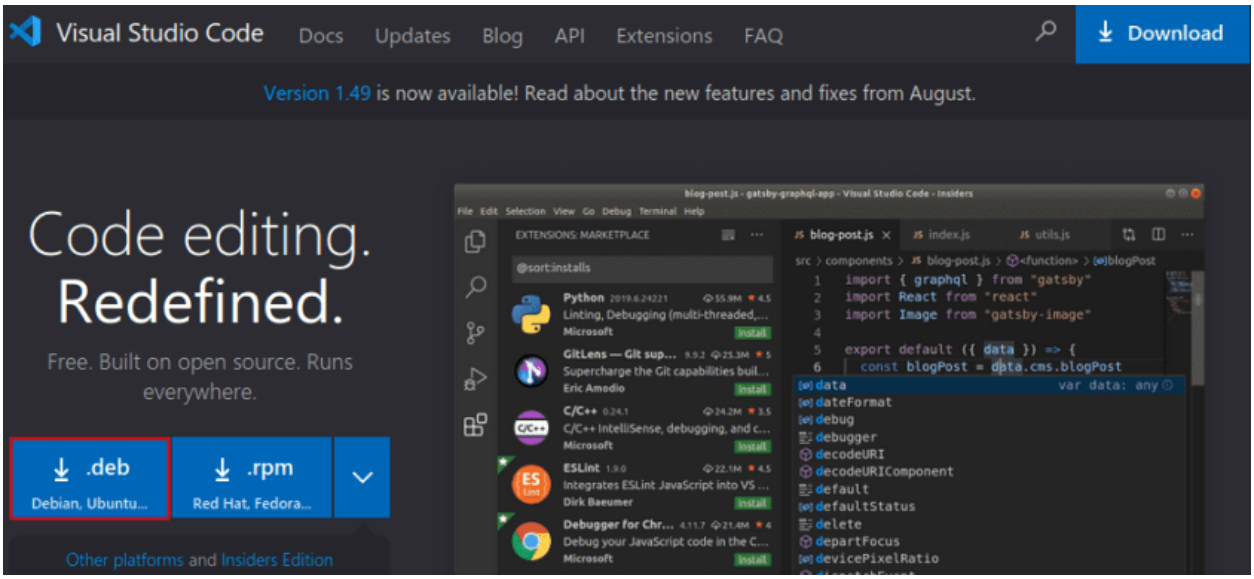
3) You'll be greeted by a Welcome tab with the latest version's released notes.



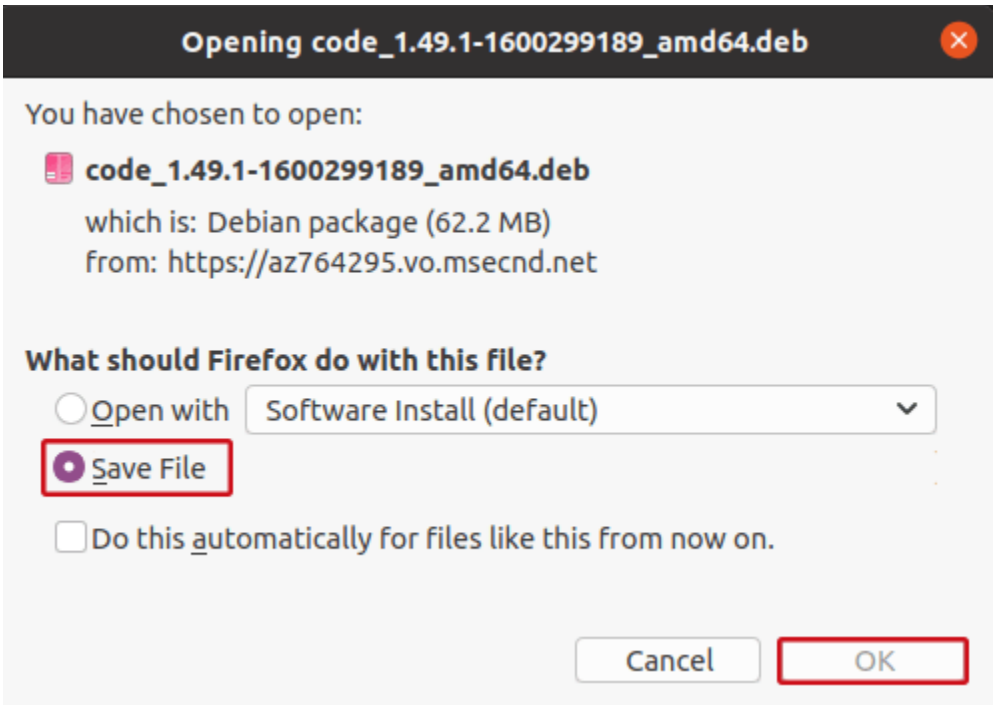
That's it. You successfully installed Visual Studio Code.

Linux Ubuntu

1) Go to https://code.visualstudio.com/ and download the stable build for your operating system (Linux Ubuntu).



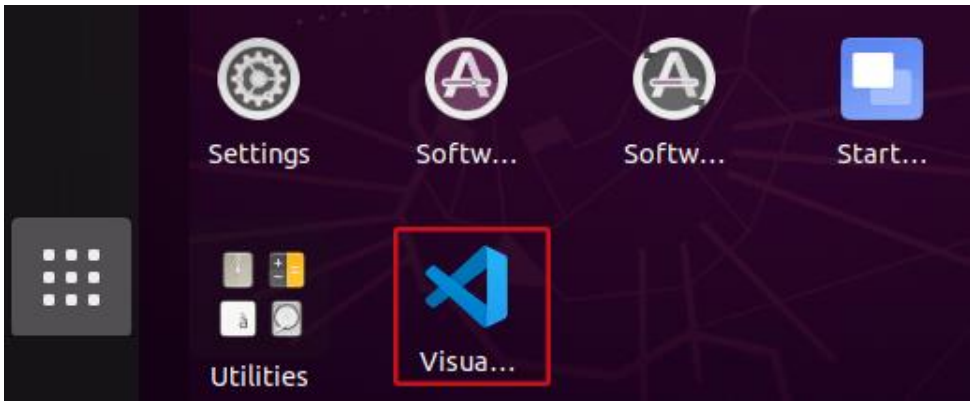
2) Save the installation file:



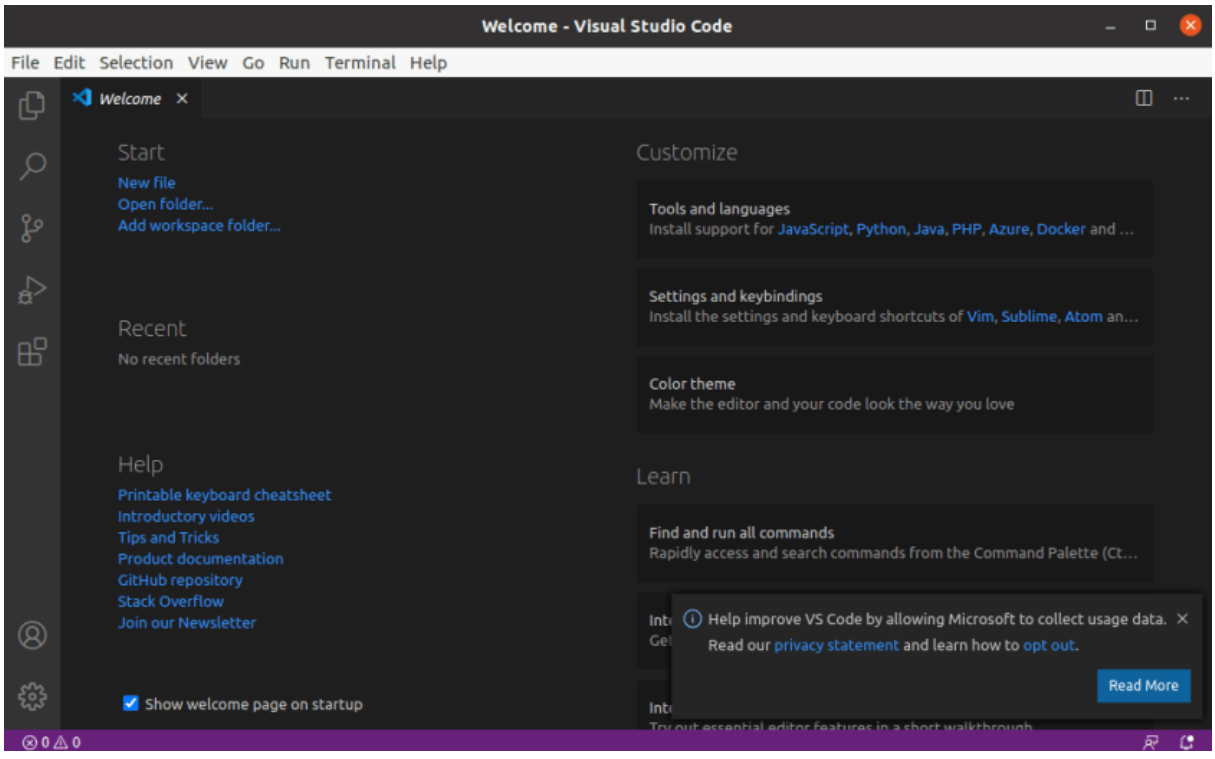
3) To install it, open a Terminal window, navigate to your *Downloads* folder and run the following command to install VS Code.

```
$ cd Downloads
~/Downloads $ sudo apt install ./code_1.49.1-1600299189_amd64.deb
```

When the installation is finished, VS Code should be available in your applications menu.



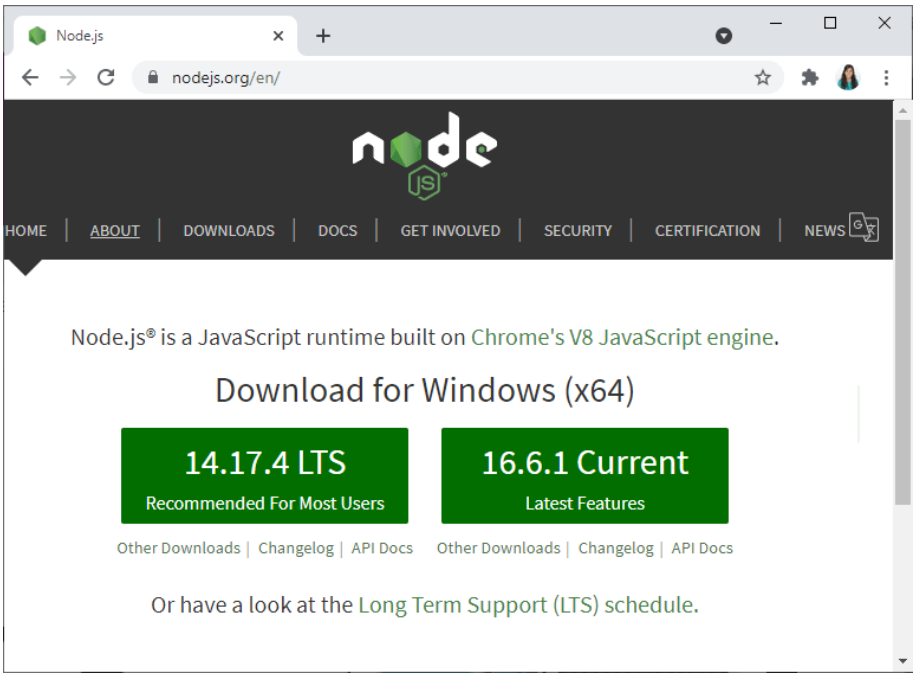
4) You'll be greeted by a Welcome tab with the newest version's released notes.



That's it. You successfully installed Visual Studio Code.

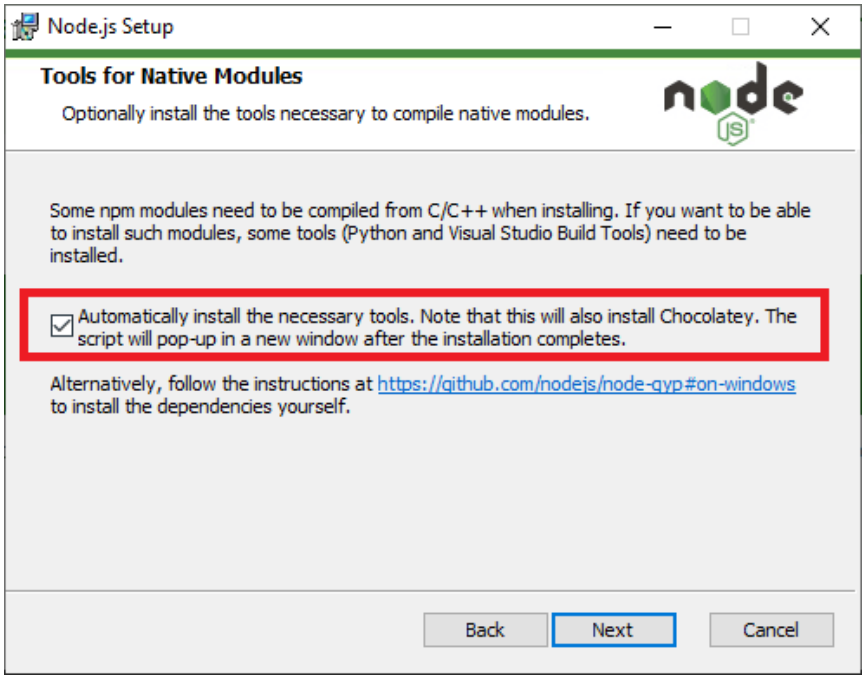
Installing Node.js

1) Go to nodejs.org and download the LTS version.

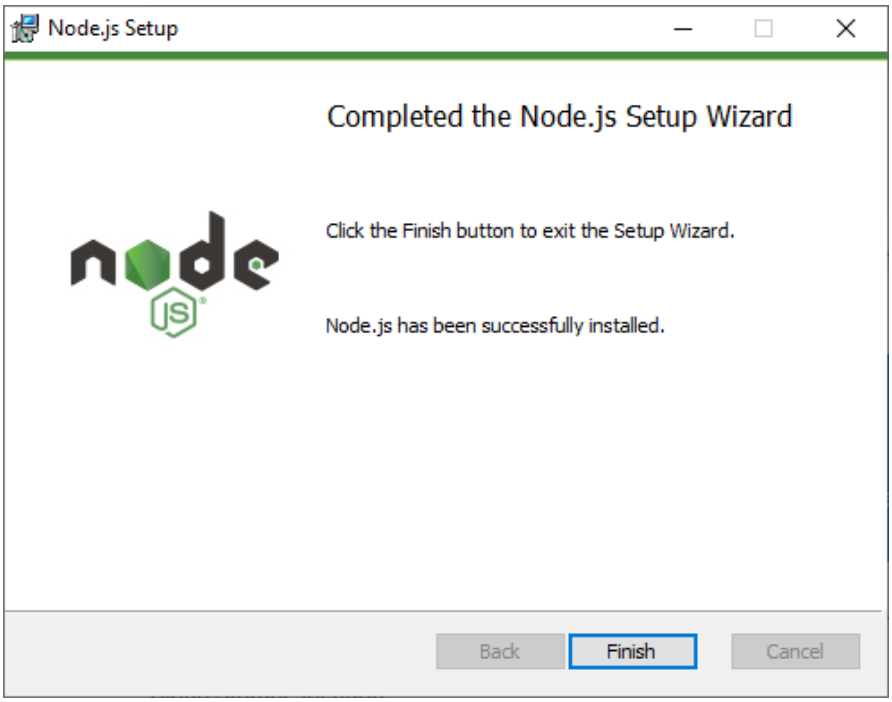


2) Run the executable file and follow the installation process.

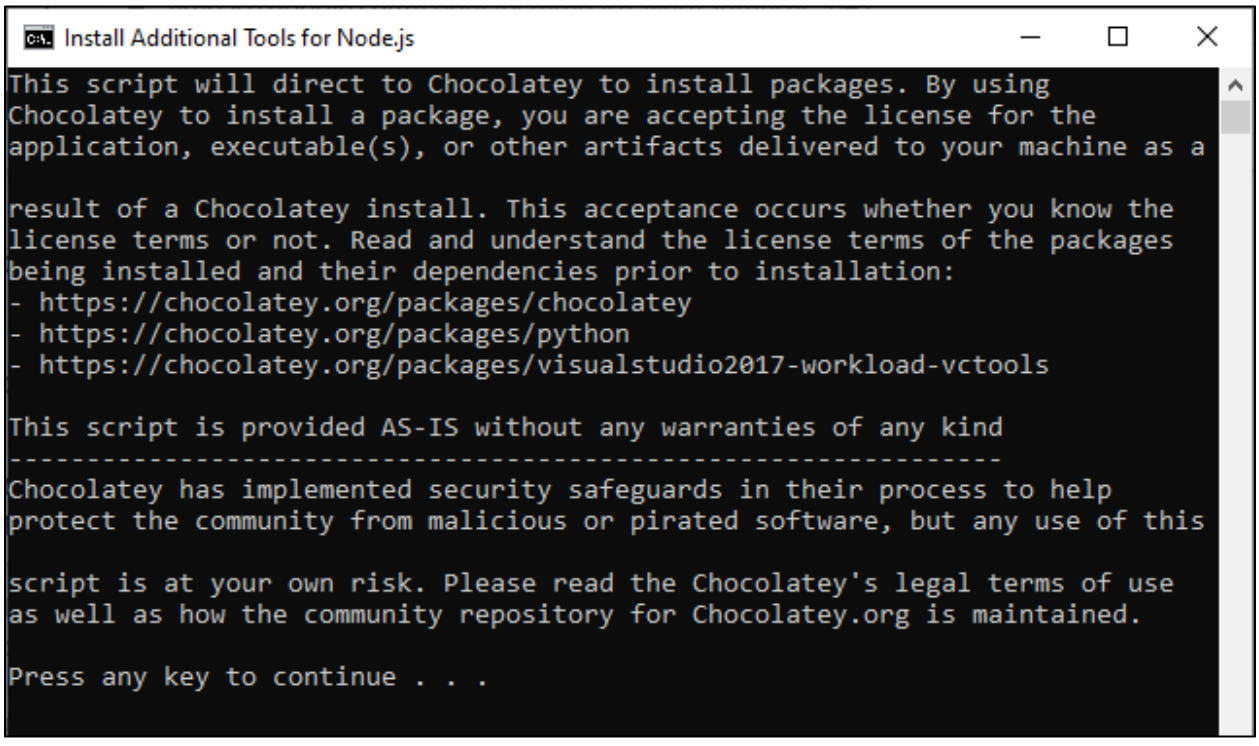
3) Enable to install all the necessary tools automatically.



4) When it's done, click **Finish**.



5) A Terminal window will open to install the Additional Tools for Node.js. When it's done, click any key to continue. When it's finished, you can close the Terminal Window.

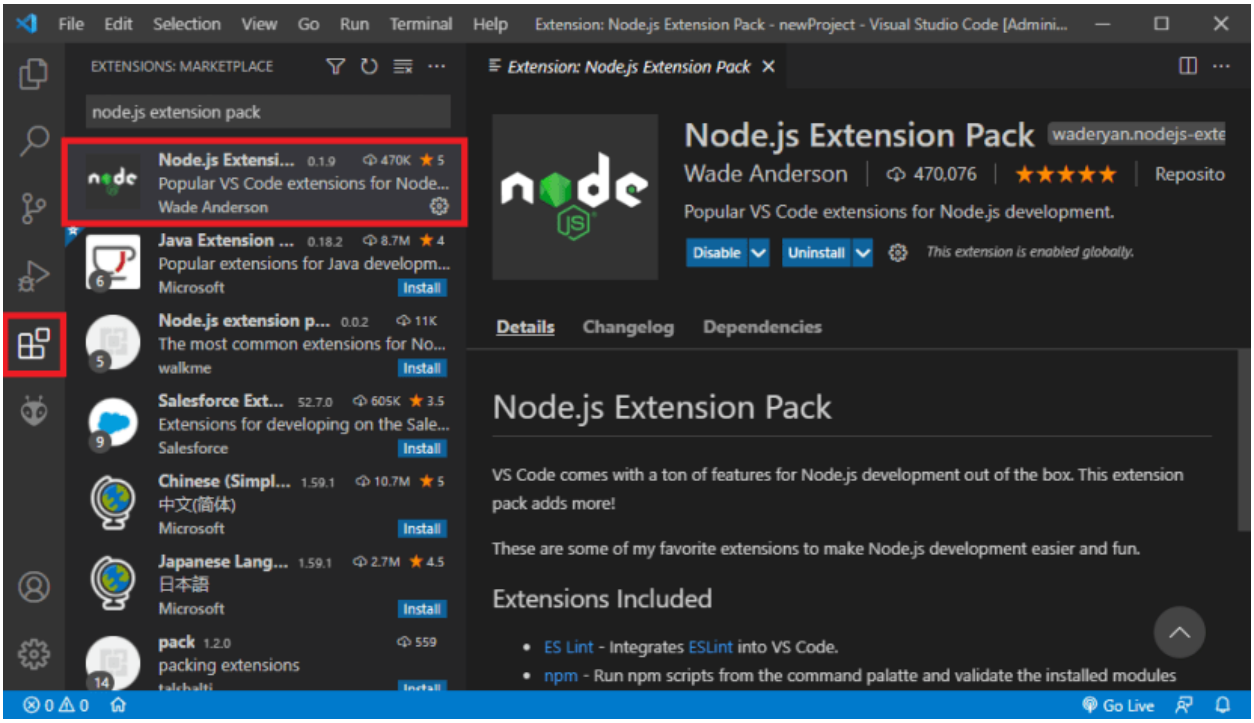


Installing Node.js Extension Pack (VS Code)

1) Open VS Code.

2) Click on the extensions tab and search for "**node.js extension pack**".

3) Click on the **Install** button at the window at the right.



4) Restart VS Code.

Installing Firebase Tools (VS Code)

1) Open VS Code. Close all opened projects, if any.

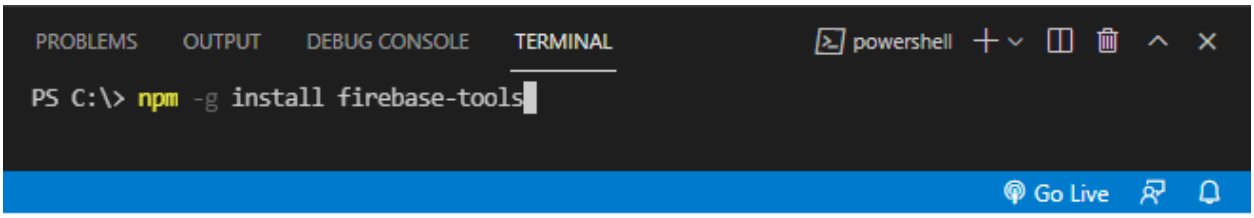
2) Open a new Terminal window. Go to **Terminal** > **New Terminal**.

3) Run the following command to change to the `C:\` path (you can install it in any other path):

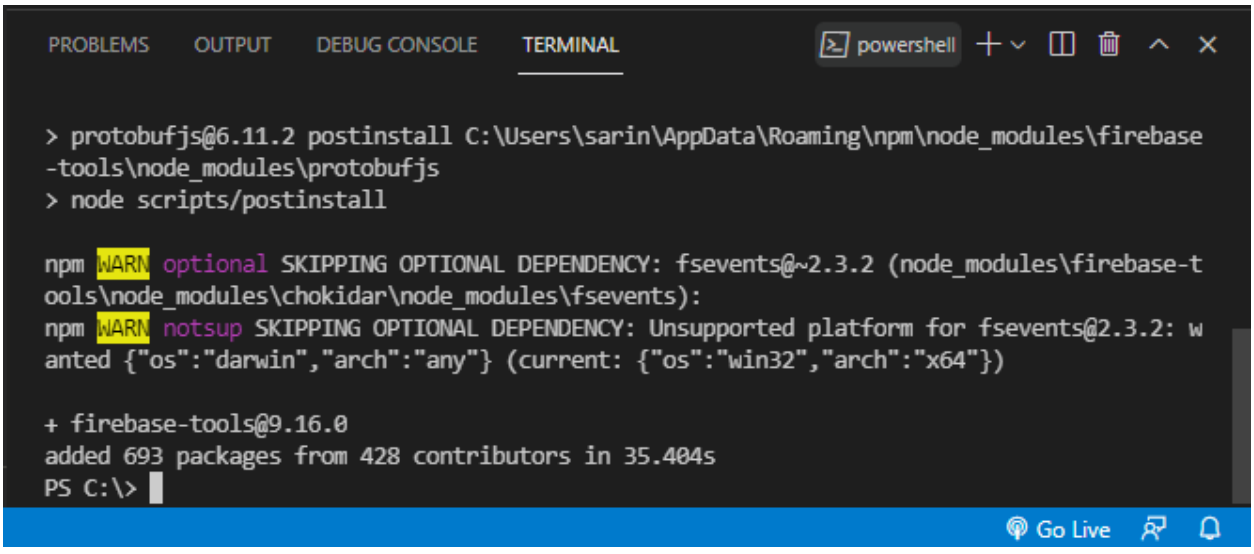
```
cd \
```

4) Run the following command to install firebase tools globally:

```
npm -g install firebase-tools
```



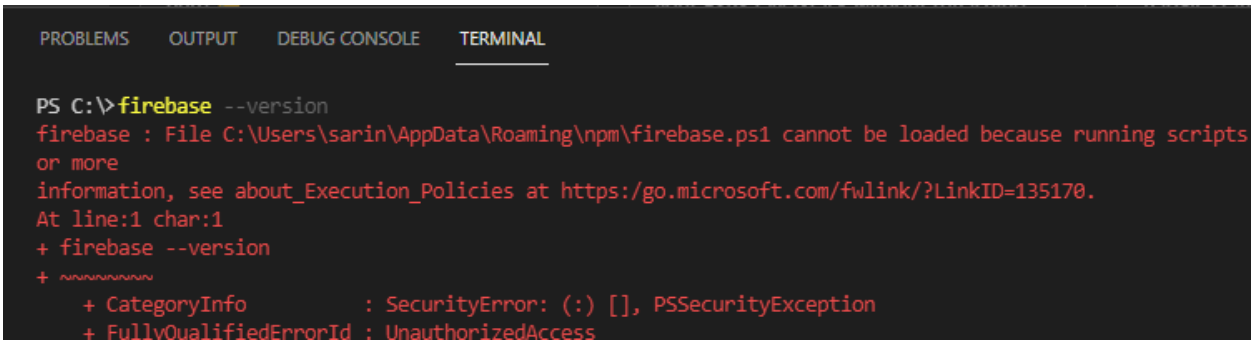
5) Firebase tools will be installed, and you'll get a similar message on the Terminal window.



6) Test if Firebase was successfully installed with the following command:

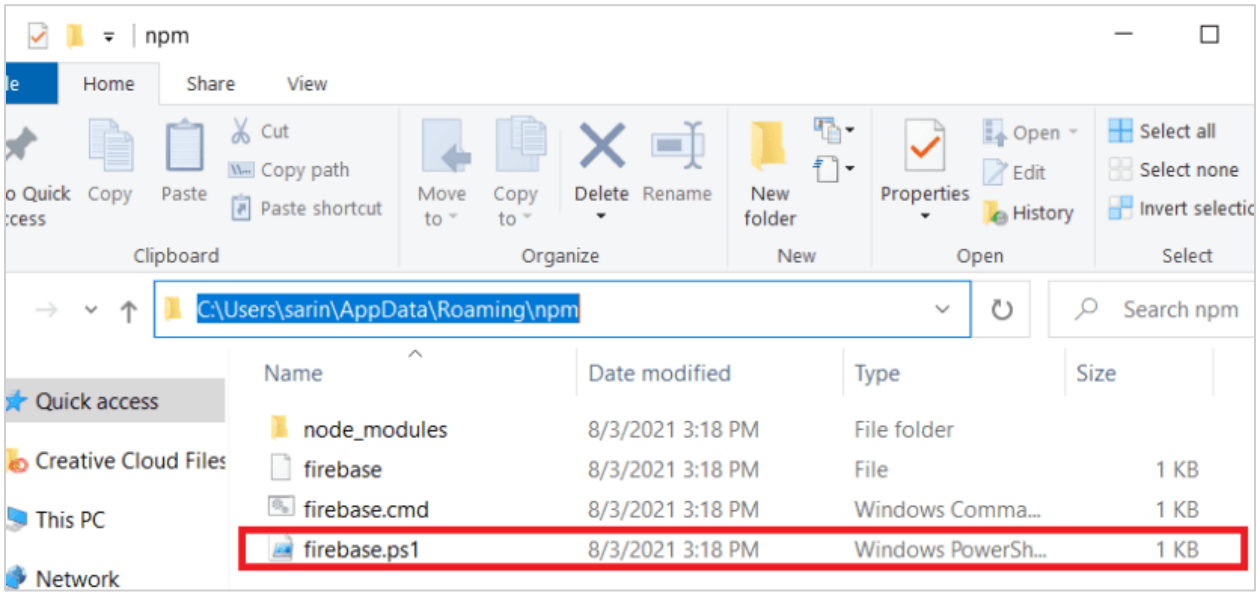
```
firebase --version
```

In my case, using a Windows PC, I get the following error.



As you can see in the error message, there's an error message related to the *firebase.ps1* file on the `C: \Users\username\AppData\Roaming\npm` path.

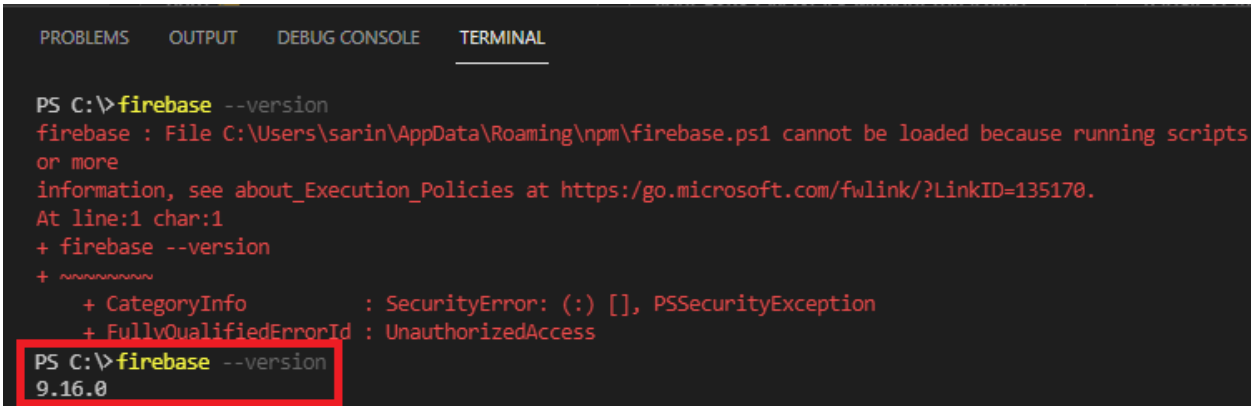
Go to that path and delete the *firebase.ps1* file.



Go back to VS Code, and rerun the following command.

```
firebase --version
```

This time, it should return the Firebase Tools version without any error.



All software needed to create your Firebase web app is now installed. You can proceed to the next section.

4.2 - Setting Up a Firebase Web App Project

Before creating the Firebase web app, you need to set up a Firebase project on VS Code. These are the steps that you'll follow in this section:

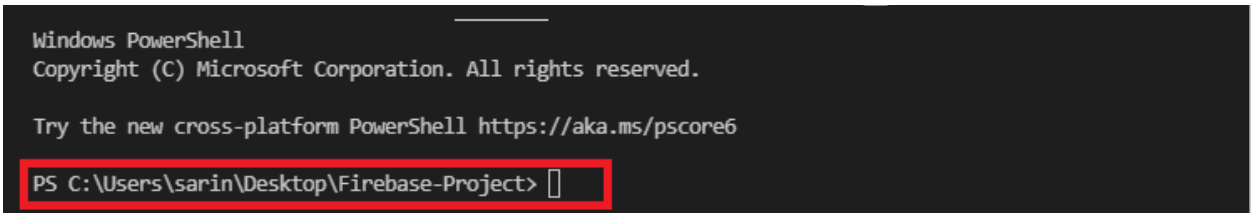
1. Creating a Project Folder
2. Firebase Login
3. Initializing Web App Firebase Project
4. Adding Firebase To Your App

1) Creating a Project Folder

1) Create a folder on your computer to save your Firebase project—for example, *Firebase-Project*.

2) Open VS Code. Go to **File** > **Open Folder...** and select the folder you've just created.

3) Go to **Terminal** > **New Terminal**. A new Terminal window should open on your project path.

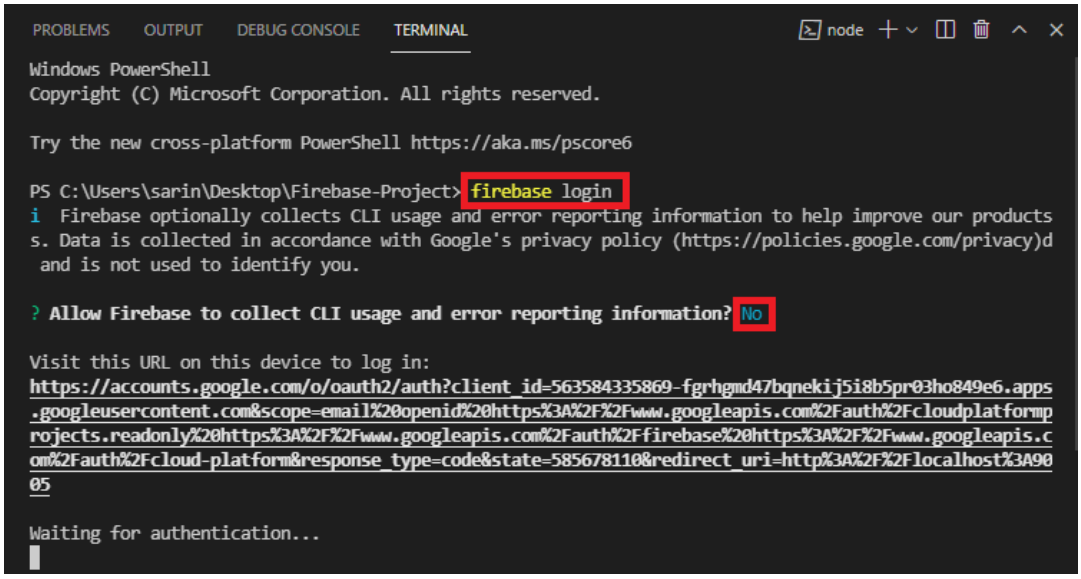


2) Firebase Login

1) On the previous Terminal window, type the following and then press Enter:

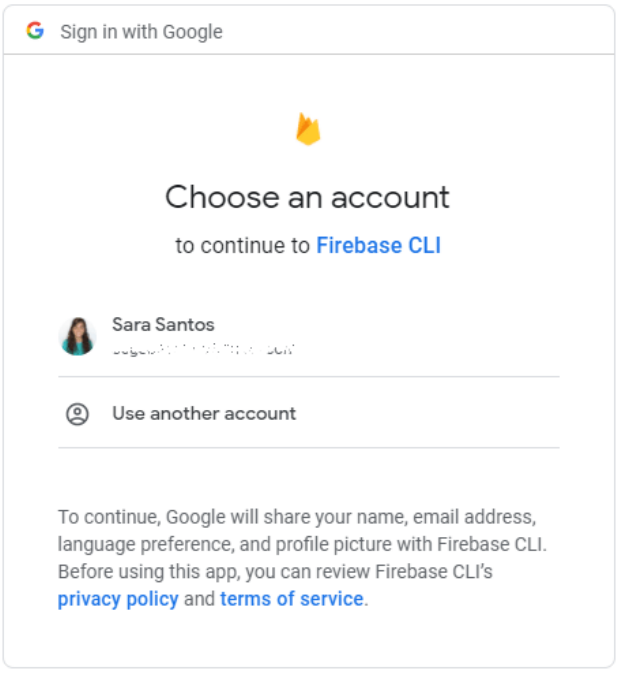
```
firebase login
```

2) You'll be asked to collect CLI usage and error reporting information. Enter "n" and press Enter to deny.

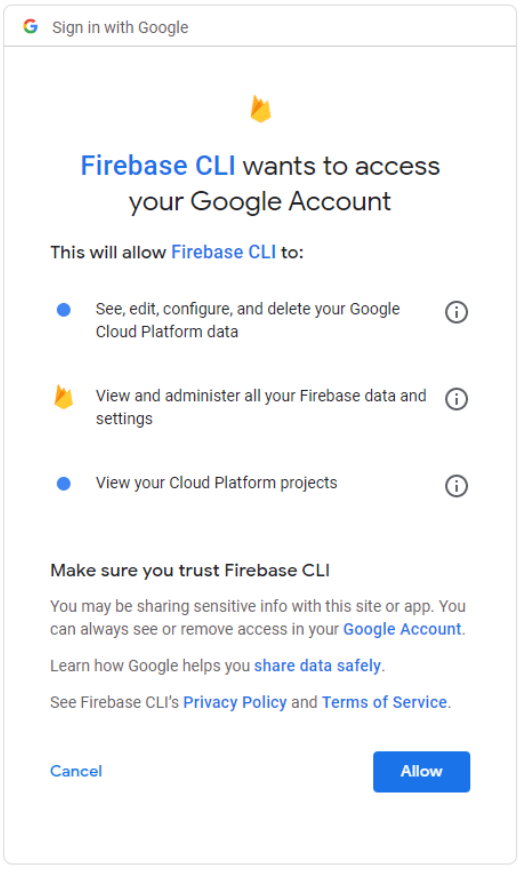


Note: If you are already logged in, it will show a message saying: "Already logged in as user@gmail.com".

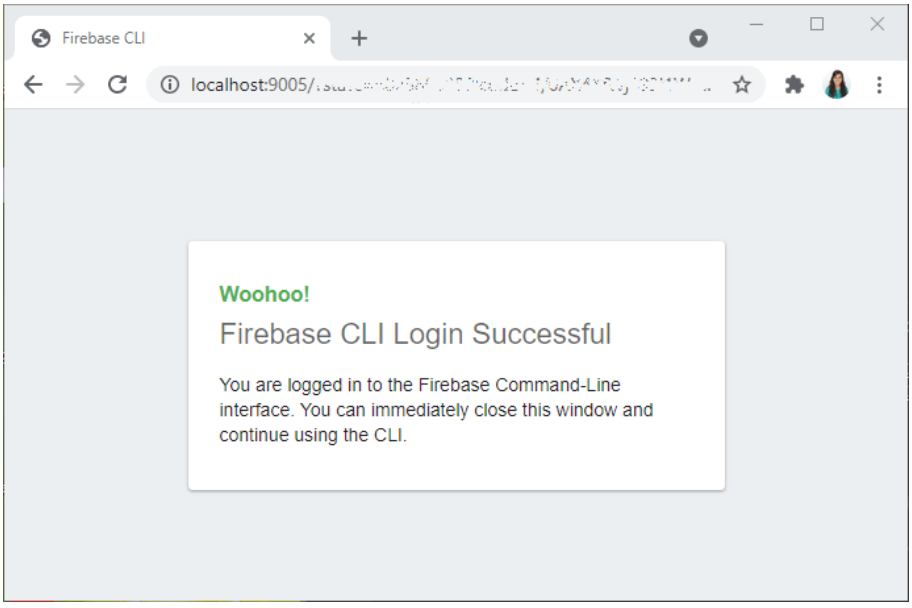
3) After this, it will pop up a new window on your browser to login into your firebase account.



4) Allow Firebase CLI to access your google account.



5) After this, Firebase CLI login should be successful. You can close the browser window.

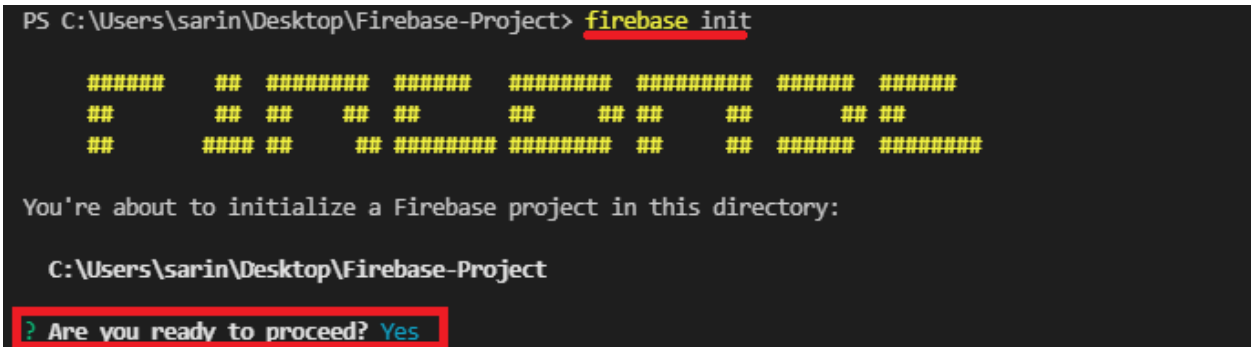


3) Initializing Web App Firebase Project

1) After successfully login in, run the following command to start a Firebase project directory in the current folder.

```
firebase init
```

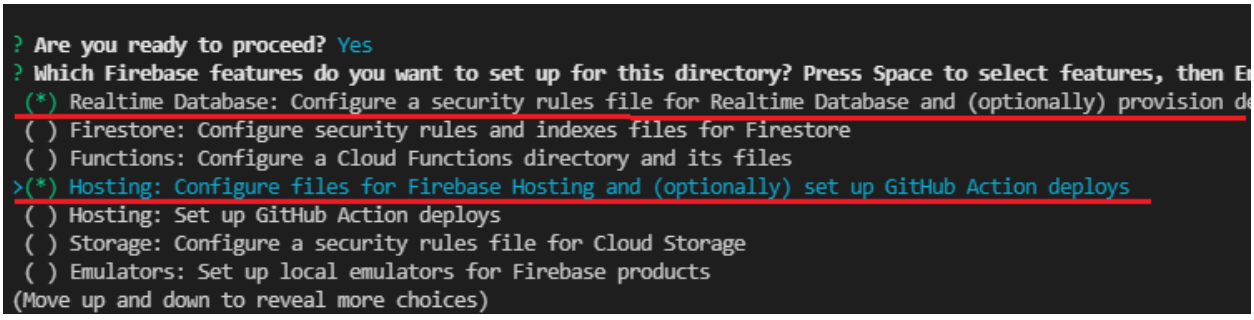
2) You'll be asked if you want to initialize a Firebase project in the current directory. Type **Y** and hit Enter.



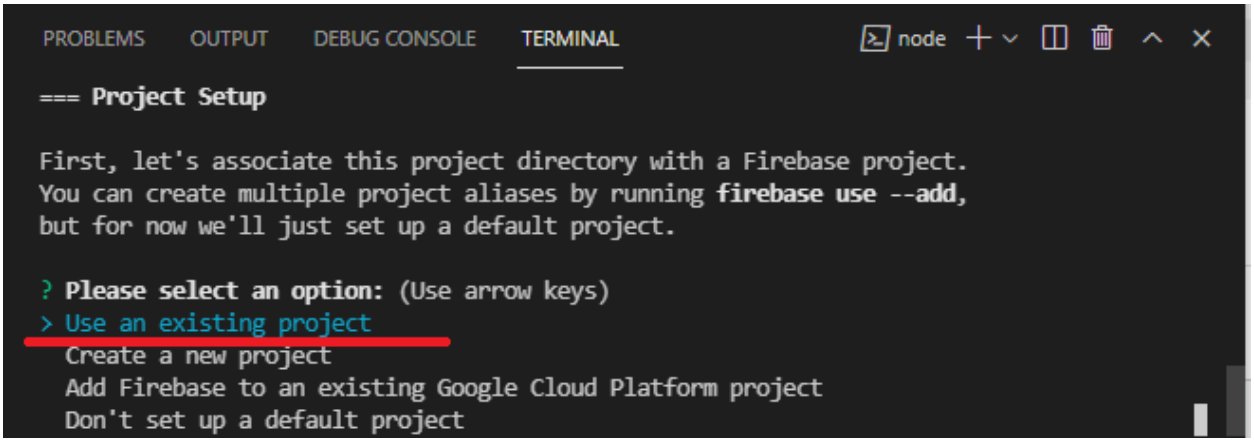
3) Then, use the up and down arrow keys and the Space key to select the options. Select the following options:

- **√ Realtime Database**: Configure security rules file for Realtime Database and (optionally) provision default instance.
- **√ Hosting**: Configure files for Firebase Hosting and (optionally) set up GitHub Action deploys

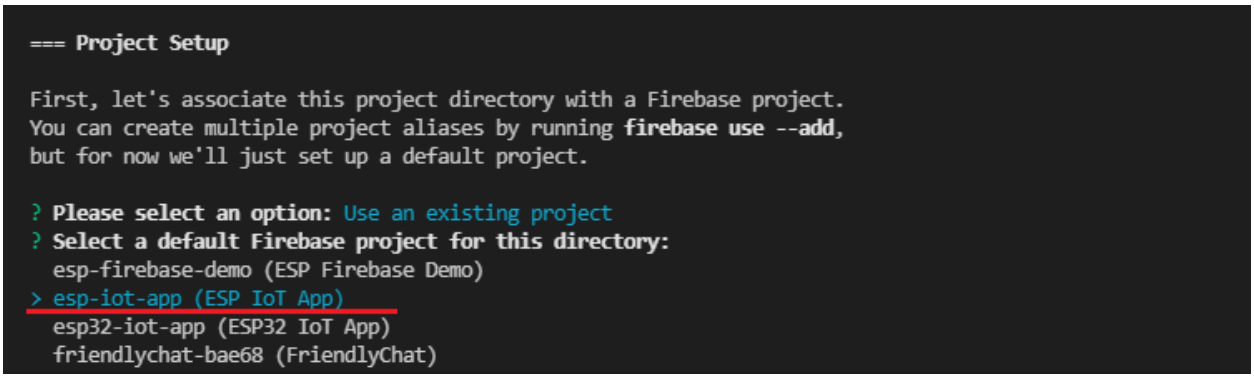
The selected options will show up with a green asterisk. Then, hit Enter.



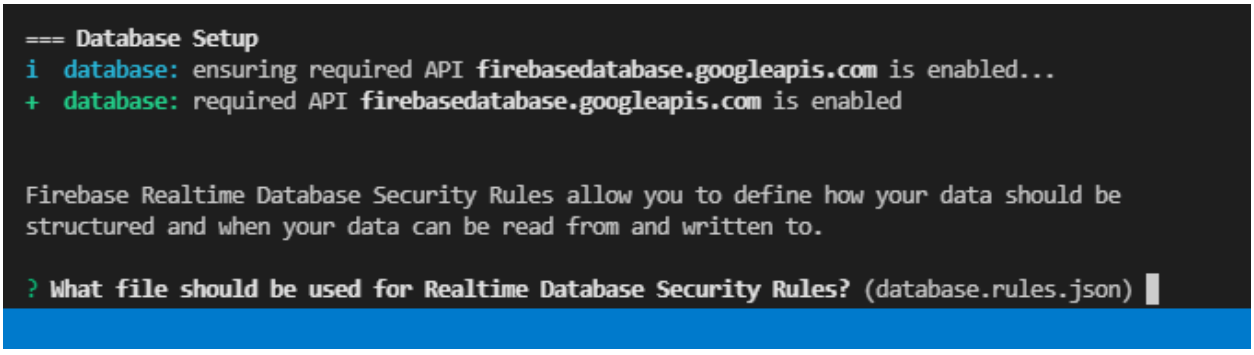
4) Select the option "Use an existing project"—it should be highlighted in blue—then, hit Enter.



5) After that, select the Firebase project for this directory—it should be the project created previously. In my case, it is called **esp-iot-app**. Then, press the Enter key.



6) Press **Enter** on the following question to select the default database security rules file: "What file should be used for Realtime Database Security Rules? "

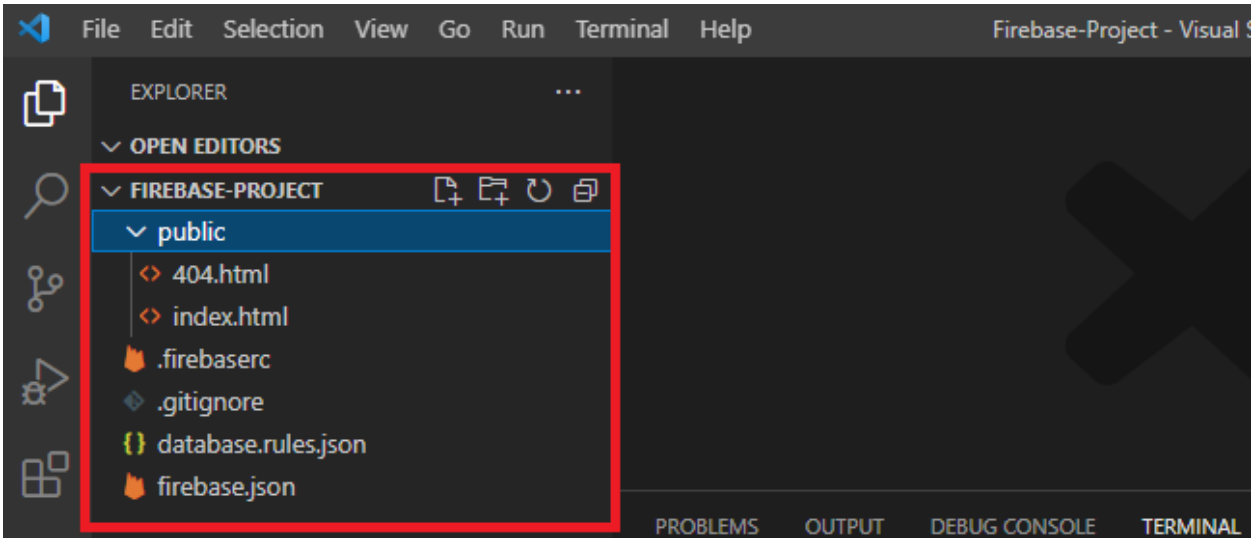


7) Then, select the hosting options as shown below:

- What do you want to use as your public directory? Hit **Enter** to select *public*.
- Configure as a single-page app (rewrite urls to /index.html)? **No**
- Set up automatic builds and deploys with GitHub? **No**

```
=== Hosting Setup

Your public directory is the folder (relative to your project directory) that
will contain Hosting assets to be uploaded with firebase deploy. If you
have a build process for your assets, use your build's output directory.

? What do you want to use as your public directory? public
? Configure as a single-page app (rewrite all urls to /index.html)? No
? Set up automatic builds and deploys with GitHub? No
+  Wrote public/404.html
+  Wrote public/index.html

i  Writing configuration info to firebase.json...
i  Writing project information to .firebaserc...

+  Firebase initialization complete!
```

8) The Firebase project should now be initialized successfully. Notice that VS code created some files under your project folder.



The *index.html* file contains some HTML text to build a web page. For now, leave the default HTML text. The idea is to replace that with your own HTML text to build a custom web page for your needs. We'll do that in one of the following sections.

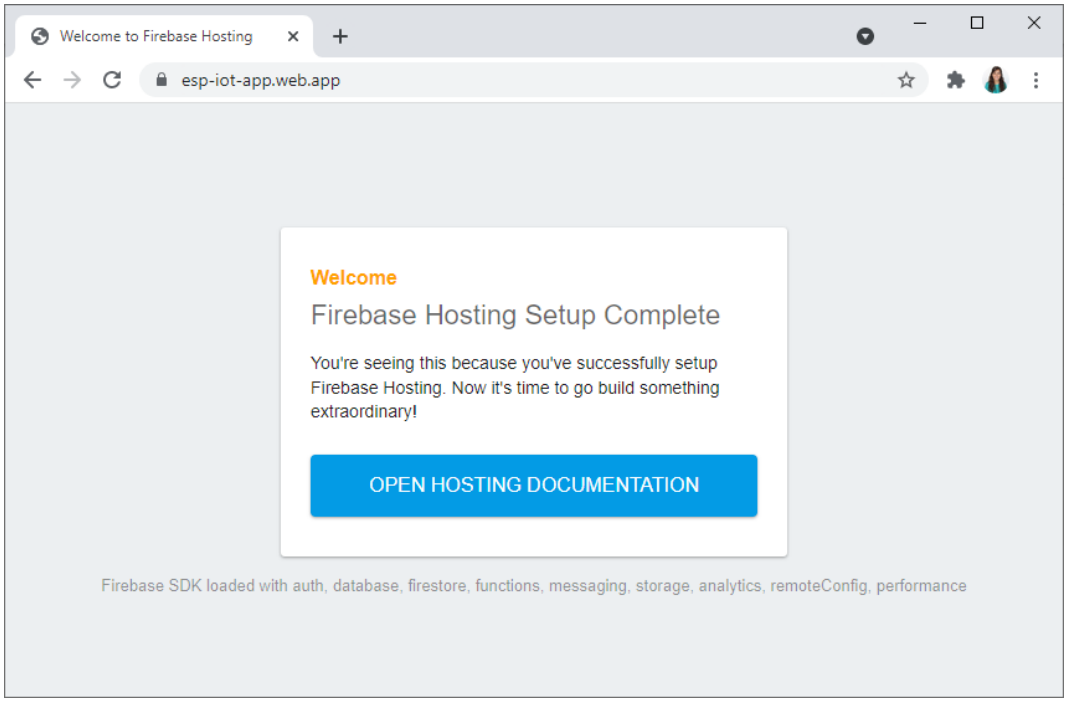
9) To check if everything went as expected, run the following command on the VS Code Terminal window.

```
firebase deploy
```

```
i  hosting[esp-iot-app]: beginning deploy...
i  hosting[esp-iot-app]: found 2 files in public
+  hosting[esp-iot-app]: file upload complete
i  database: releasing rules...
+  database: rules for database esp-iot-app-default-rtdb released successfully
i  hosting[esp-iot-app]: finalizing version...
+  hosting[esp-iot-app]: version finalized
i  hosting[esp-iot-app]: releasing new version...
+  hosting[esp-iot-app]: release complete

+  Deploy complete!

Project Console: https://console.firebase.google.com/project/esp-iot-app/overview
Hosting URL: https://esp-iot-app.web.app
PS C:\Users\sarin\Desktop\Firebase-Project>
```

You should get a **Deploy complete!** message and an URL to the Project Console and the Hosting URL.

10) Copy the hosting URL and paste it into a web browser window. You should see the following web page. You can access that web page from anywhere in the world.



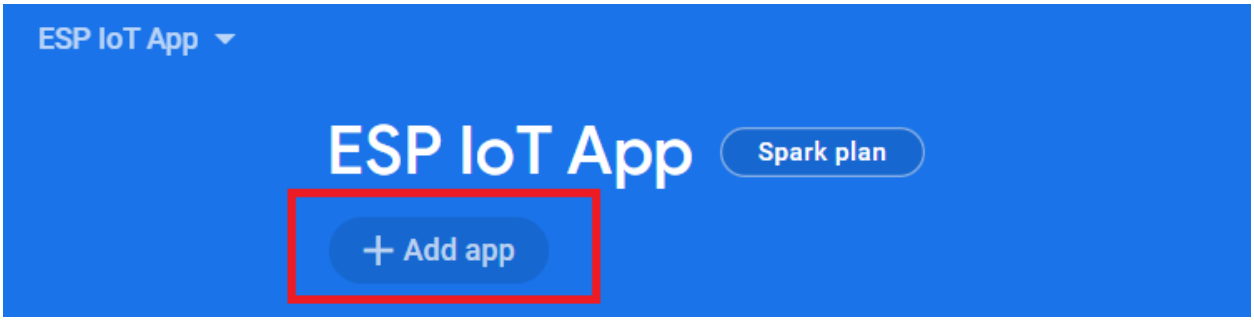
Congratulations, you've set up your Firebase app project correctly.

The web page you've seen previously is built with the HTML file placed in the *public* folder of your Firebase project. By changing the content of that file, you can create your own web app. That's what we're going to do in the next Unit.

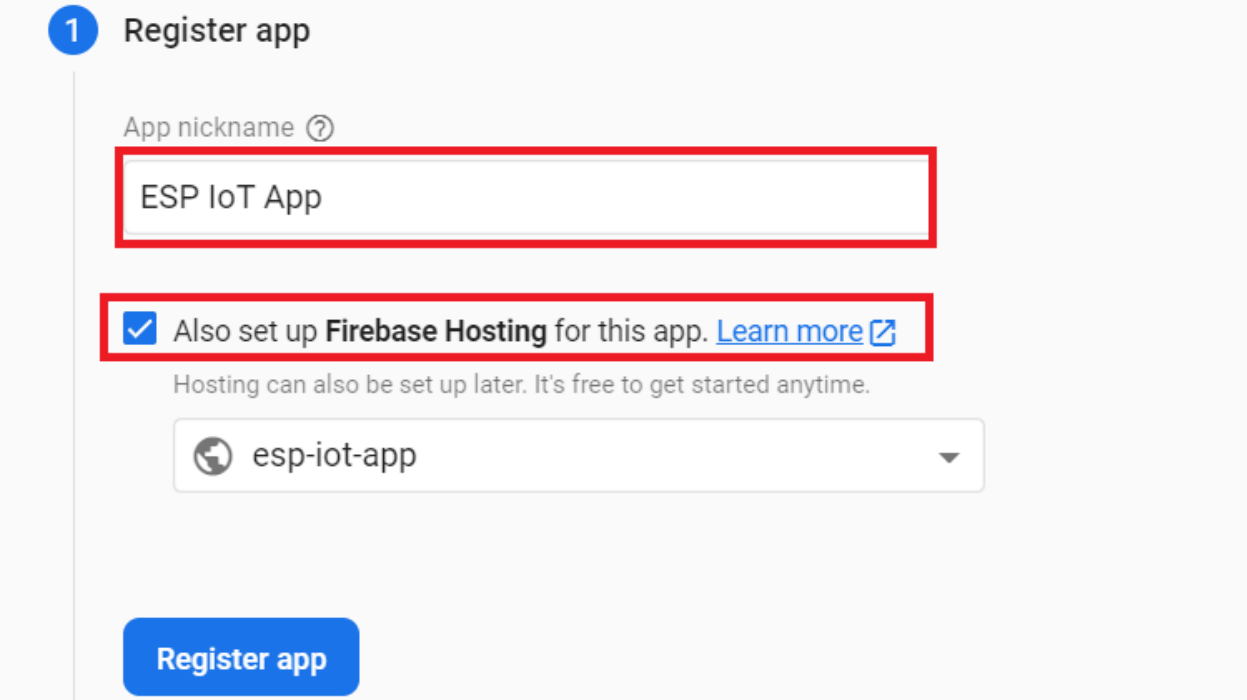
4) Adding Firebase To Your App

Before proceeding to the next Unit, you need to add Firebase to your app.

1) Go to your Firebase console and select your project. Then, click on **+Add** app button and then on the web icon **</>** to add a web app to the Firebase project.



2) Give your app a name. Then, check the box next to **√ Also set up Firebase Hosting for this App**. Click **Register app**.



3) Then, copy the `firebaseConfig` object and save it because you'll need it later.

```
// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: "AIzaSyBNyk                                ",
  authDomain: "esp-iot-app.firebaseapp.com",
  databaseURL: "https://esp-iot-app-default-rtdb.europe-west1.firebase
  projectId: "esp-iot-app",
  storageBucket: "esp-iot-app.appspot.com",
  messagingSenderId: "            ",
  appId: "1:30                                        "
};
```

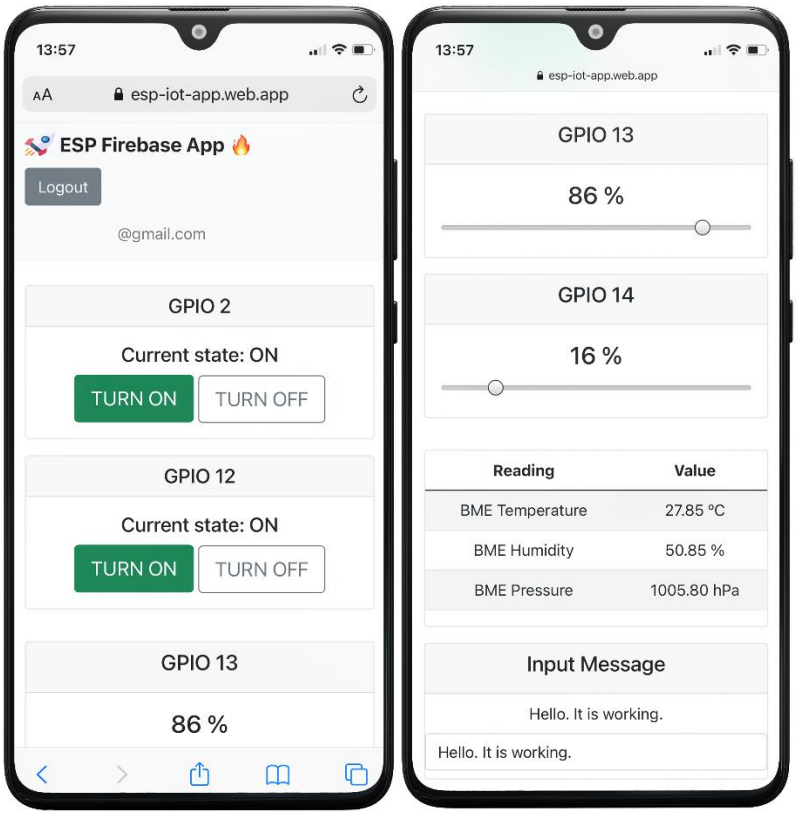
After this, you can also access the `firebaseConfig` object if you go to your Project settings in your Firebase console.

4) Click **Next** on the proceeding steps, and finally on **Continue to console**.

In the next section, we'll start creating the web app to control and monitor your ESP32 or ESP8266 boards from anywhere.

4.3 - Creating the Firebase Web App

In this section, you'll create the Firebase web app to interact with the ESP32 and ESP8266 boards using HTML and JavaScript.



Project Overview

Just to recap, here's a list of the web app features:

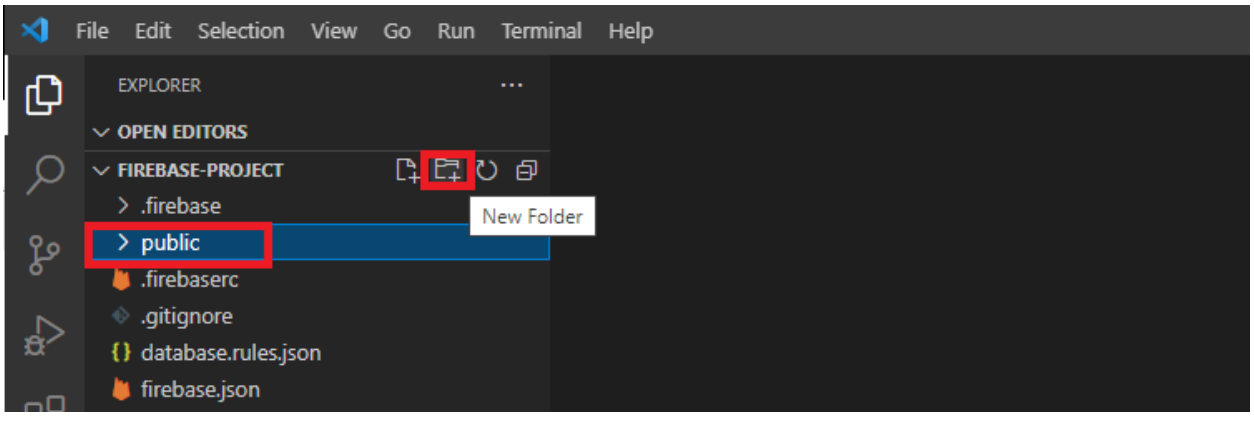
- The Firebase web app is protected with login using email and password. This means that only an authorized user can access the data. Additionally, your database is protected by database rules;
- The web app displays two buttons to control two GPIOs ON and OFF (GPIO 2 and GPIO 12). The GPIO states are saved on the database. The ESP detects the database changes and updates the outputs accordingly;

- Two sliders control outputs with PWM (LEDs, motors, or other peripherals that require PWM). The slider values are saved and updated on the database. The ESP detects the database changes and updates the outputs accordingly;
- A table displays sensor readings. The web app gets the new readings from the database nodes that the ESP is publishing in;
- Finally, there's also an input field to insert a message to display on an OLED. The message is saved on the database;
- You can access your firebase web app from anywhere in the world to interact with your ESP32 and ESP8266 boards.

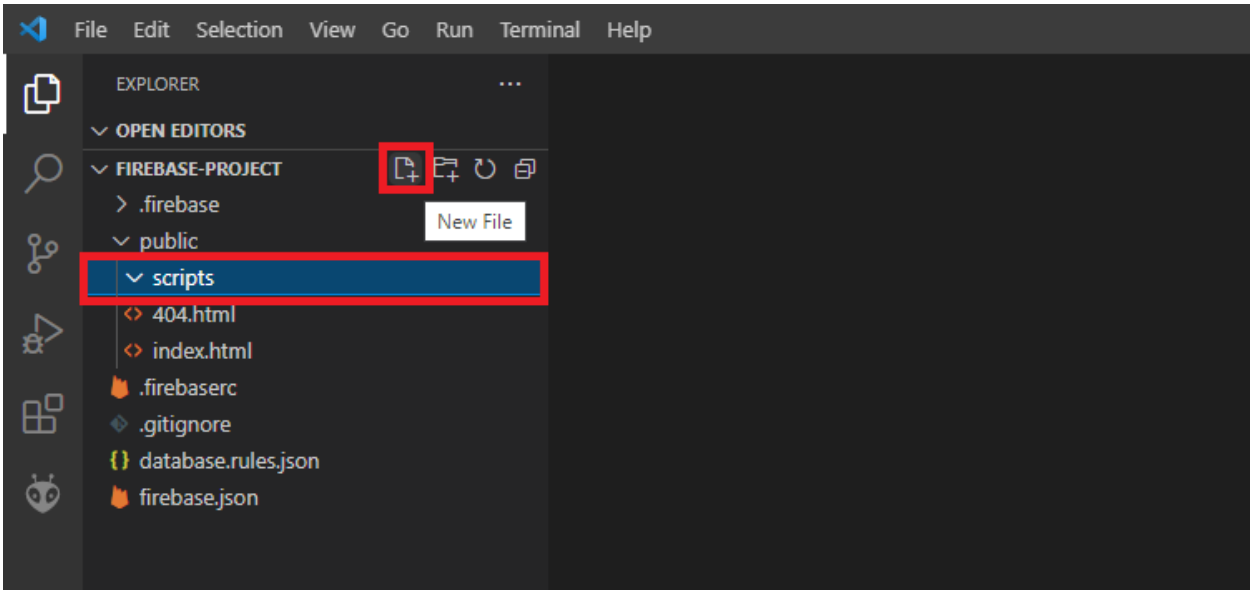
Creating the JavaScript Files

We'll add two JavaScript files to our project. One called *auth.js* to handle everything related to the authentication, and another called *index.js* to control the user interface.

1) With your project opened from the previous Unit, create a new folder called *scripts* inside the *public* folder. You can do that in VS Code. Select the *public* folder and click on the **+folder** icon.

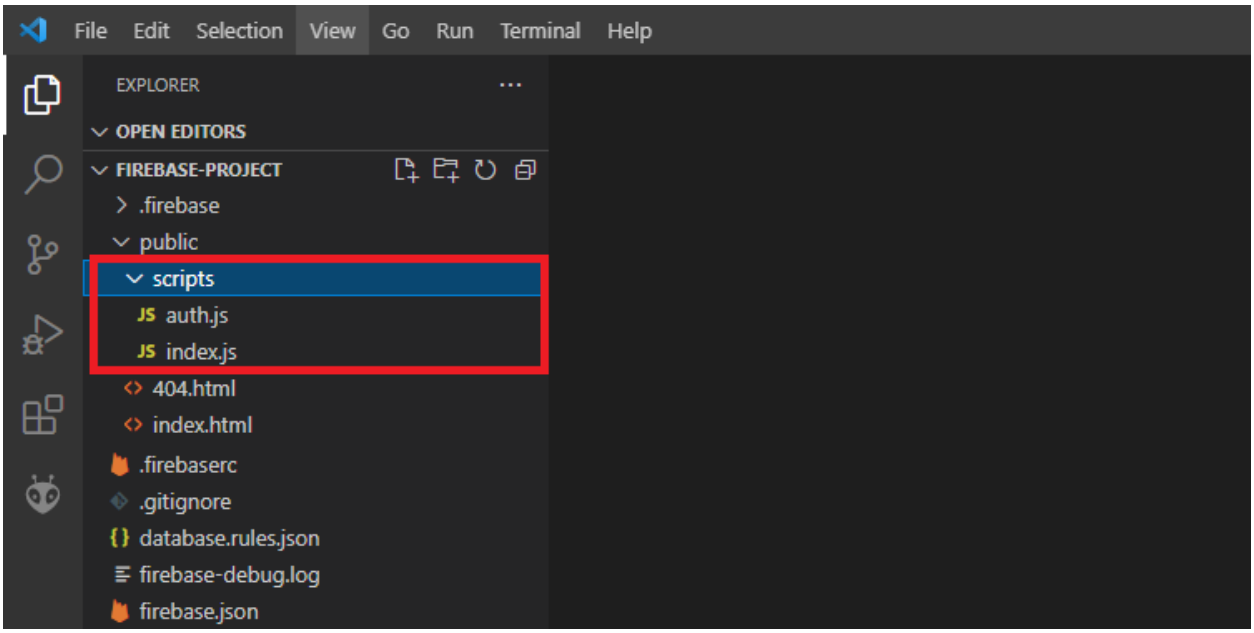


2) Then, create two files inside the newly created *scripts* folder. To do that, select the *scripts* folder and then click on the **+file** icon.



3) Create one file called *index.js*.

4) Then, create another file called *auth.js*. You should have two JavaScript *.js* files inside the *scripts* folder, as shown in the following picture.



Add a Logo

The web page will display a little logo on the left side of the navigation bar. If you want to add a logo, you should move an image file called *logo.png* to the *public* folder. You can use our image or any other image you like. It should be square.

⇒ Download logo.png file

To move an image to the *public* folder, you can drag the image file from your computer to the *public* folder on VS Code.

Building the Web Page (HTML File)

Let's start by creating the layout of the web app. We made the layout using HTML and Bootstrap 5. Bootstrap is a CSS framework with CSS-based design templates to build a responsive web page. Explaining exactly how Bootstrap framework works and all its features goes beyond the scope of this eBook. So, if this is your first time using Bootstrap, we recommend watching or reading a quick tutorial to get you familiar with how it works. Here are some resources to help you get started:

- Bootstrap 5 Documentation
- W3Schools Bootstrap 5

Copy the following to your *index.html* file.

⇒ Download index.html file

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
```

```
    <title>ESP IoT Firebase App</title>

    <!-- update the version number as needed -->
    <script src="https://www.gstatic.com/firebasejs/8.8.1/firebase-app.js"></script>

    <!-- include only the Firebase features as you need -->
    <script src="https://www.gstatic.com/firebasejs/8.8.1/firebase-auth.js"></script>
   <script src="https://www.gstatic.com/firebasejs/8.8.1/firebase-database.js"></script>

    <script>
      // REPLACE WITH YOUR APP CONFIG OBJECT
      var firebaseConfig = {
        apiKey: "AIzaSyBNyklqafaavq8HFVXXXXXXXXXXXXX",
        authDomain: "esp-iot-app.firebaseapp.com",
       databaseURL: "https://esp-iot-app-default-rtdb.europe-west1.firebasedatabase.app",
        projectId: "esp-iot-app",
        storageBucket: "esp-iot-app.appspot.com",
        messagingSenderId: "306720XXXXXXX",
        appId: "1:30672XXX7843:web:f5d899575XXXXXXXXX3796"
      };

      // Initialize firebase
      firebase.initializeApp(config);
      // Make auth and database references
      const auth = firebase.auth();
      const db = firebase.database();

    </script>

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/css/bootstrap.min.css" rel="
stylesheet" integrity="sha384-KyZXEAg3QhqLMpG8+8fhAXLRk2vvoC2f3B09zVXn8CA5QIVfZOJ3BCsw2P0 p/We"
crossorigin="anonymous">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

  </head>
  <body  class="min-vh-100 d-flex flex-column justify-content-between">
    <!--NAVBAR-->
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
      <div class="container d-flex justify-content-between">
        <div class="navbar-header">
          <a class="navbar-brand" href="#">
            <img src="logo.png" alt="" width="30" height="24" class="d-inline-block align-text-top">
            <strong>ESP Firebase App 🔥</strong>
          </a>
        </div>
        <div class="navbar-collapse justify-content-end" id="navbarNav">
          <ul class="navbar-nav">
            <li class="nav-item">
              <a class="nav-link" href="#" id="login-link">
                    <button  id="loginBtn"  class="btn  btn-secondary"  data-bs-
toggle="modal" data-bs-target="#login-modal">Login</button>
              </a>
            </li>
            <li class="nav-item">
             <a class="nav-link" href="#" id="logout-link" style="display: none;">
```

```
                <button id="logoutBtn" class="btn btn-secondary navbar-btn" data-
bs-toggle="modal" data-bs-target="#logout-modal">Logout</button>
              </a>
            </li>
          </ul>
          <span class="navbar-text" id="user-details" style="display: none;">
            User details
          </span>
        </div>
      </div>
    </nav>
    <!-- LOGIN MODAL -->
    <div class="modal fade" id="login-modal" tabindex="-1" aria-labelledby="exampleModalLabel"
aria-hidden="true">
      <div class="modal-dialog">
        <div class="modal-content">
          <div class="modal-header">
            <h5 class="modal-title">Login</h5>
            <button type="button" class="btn-close" data-bs-dismiss="modal" aria-
label="Close"></button>
          </div>
          <div class="modal-body">
            <p>Insert your email and password to login.</p>
            <form id="login-form">
              <div class="mb-3">
                <label for="inputEmail" class="form-label">Email address</label>
                <input type="email" class="form-control" id="input-email">
              </div>
              <div class="mb-3">
                <label for="inputPassword" class="form-label">Password</label>
                <input type="password" class="form-control" id="input-password">
              </div>
              <button type="submit" class="btn btn-primary">Login</button>
            </form>
          </div>
        </div>
      </div>
    </div>
    <!-- LOGOUT MODAL -->
    <div class="modal fade" id="logout-modal" tabindex="-1">
      <div class="modal-dialog">
        <div class="modal-content">
          <div class="modal-header">
            <h5 class="modal-title">Logout</h5>
            <button type="button" class="btn-close" data-bs-dismiss="modal" aria-
label="Close"></button>
          </div>
          <div class="modal-body">
            <p>Are you sure you want to logout?</p>
          </div>
          <div class="modal-footer">
              <button id="logout" type="button" class="btn btn-primary" data-bs-
dismiss="modal" aria-label="Close">Logout</button>
```

```
          <button type="button" class="btn btn-secondary" data-bs-dismiss="modal"
aria-label="Close">Cancel</button>
          </div>
        </div>
      </div>
    </div>
    <!-- PAGE CONTENT -->
    <!-- Content if user is logged out -->
    <div id="signedOutMessage" class="text-center" style="display: none;">You are
logged out or user doesn't have permissions.</div>
    <!-- Content if user is logged in -->
   <div class="container mt-4 mb-4" id="dashboardSignedIn" style="display: none;">
     <!--Container for buttons-->
     <div class="row mb-3">
       <!-- button 1-->
       <div class="col-sm mb-3 text-center">
         <div class="card">
           <h5 class="card-header">GPIO 2</h5>
           <div class="card-body">
             <h5 class="card-title">Current state: <span id="btn1State"></span></h5>
              <button id="btn1On" class="btn btn-lg btn-block btn-success">TURN ON</button>
              <button id="btn1Off" class="btn btn-lg btn-block btn-outline-secondary">TURN OFF</button>
           </div>
         </div>
       </div>
       <!-- button 2-->
       <div class="col-sm mb-3 text-center">
         <div class="card">
           <h5 class="card-header">GPIO 12</h5>
           <div class="card-body">
             <h5 class="card-title">Current state: <span id="btn2State"></span></h5>
             <button id="btn2On" class="btn btn-lg btn-block btn-success">TURN ON</button>
             <button id="btn2Off" class="btn btn-lg btn-block btn-outline-secondary">TURN OFF</button>
           </div>
         </div>
       </div>
     </div>
     <!--Container for sliders-->
     <div class="row mb-3">
       <!-- Slider 1 -->
       <div class="col-sm mb-3 text-center">
         <div class="card">
           <div class="card-header">
             <h4>GPIO 13</h4>
           </div>
           <div class="card-body">
             <h2 class="card-title" id="sld1Value"></h2>
            <input id="sld1" type="range" min="0" max="100" style="width:100%;">
           </div>
         </div>
       </div>
        <!-- Slider 2 -->
        <div class="col-sm mb-3 text-center">
         <div class="card">
```

```
          <div class="card-header">
            <h4>GPIO 14</h4>
          </div>
          <div class="card-body">
            <h2 class="card-title" id="sld2Value"></h2>
            <input id="sld2" type="range" min="0" max="100" style="width:100%;">
          </div>
        </div>
      </div>
    </div>
    <!--Container for sensor readings-->
    <div class="row mb-3">
      <div class="col-mb-3 text-center">
        <!-- Table with Sensor Readings -->
        <div class="card">
          <table class="table table-striped">
            <thead>
              <tr>
                <th scope="col">Reading</th>
                <th scope="col">Value</th>
              </tr>
            </thead>
            <tbody>
              <tr>
                <td>BME Temperature</td>
                <td id="bmeTemp"></td>
              </tr>
              <tr>
                <td>BME Humidity</td>
                <td id="bmeHumi"></td>
              </tr>
              <tr>
                <td>BME Pressure</td>
                <td id="bmePres"></td>
              </tr>
            </tbody>
          </table>
        </div>
      </div>
    </div>
    <!-- Container for input message -->
    <div class="row">
      <div class="col-mb-2 text-center">
        <!-- Input Message 1 -->
        <div class="card">
          <div class="card-header">
            <h4>Input Message</h4>
          </div>
          <p id="input1Text" class="m-2 jus"></p>
          <form>
            <div class="form-group mb-2">
              <input type="text" class="form-control" style="width: 100%;" id="input1">
            </div>
          </form>
```

```
          </div>
      </div>
    </div>
  </div>
  <!--Container for footer-->
  <div class="container-fluid footer navbar-fixed-bottom bg-light">
    <footer class=" text-center text-lg-start">
      <div class="text-center p-3">
        Powered by:
       <a class="text-dark" href="https://randomnerdtutorials.com/">Random Nerd Tutorials</a>
      </div>
    </footer>
  </div>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/js/bootstrap.bundle.min.js" integrity="sha384-U1DAWAznBHeqEIlVSCgzq+c9gqGAJn5c/t99JyeKa9xxaYpSvHU5awsuZVVFIhvj" crossorigin="anonymous"></script>
    <script src="scripts/auth.js"></script>
    <script src="scripts/index.js"></script>
  </body>
</html>
```

Adding Your App Config Object

Now, you need to edit the *index.html* file with your own Firebase configuration object.

The one you've retrieved earlier in this section: Adding Firebase To Your App.

Add your configuration object in the following lines. It is highlighted in the HTML text in a different color.

```
// REPLACE WITH YOUR APP CONFIG OBJECT
var firebaseConfig = {
  apiKey: "AIzaSyBNyklqafaavq8HFVXXXXXXXXXXXXX",
  authDomain: "esp-iot-app.firebaseapp.com",
  databaseURL: "https://esp-iot-app-default-rtdb.europe-west1.firebasedatabase.app",
  projectId: "esp-iot-app",
  storageBucket: "esp-iot-app.appspot.com",
  messagingSenderId: "306720XXXXXXX",
  appId: "1:30672XXX7843:web:f5d899575XXXXXXXXX3796"
};
```

After adding your `firebaseConfig` object, save the *index.html* file.

Run the App

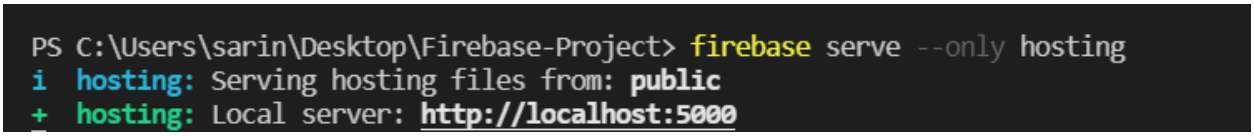
Now that you have configured the project with your app settings, you can run the project for the first time. As the web app is not finished yet, we can run it locally instead of deploying it to Firebase every time we make changes.

With your Firebase web app project opened on VS Code, open a new Terminal window. In VS Code, go to **Terminal** > **New Terminal**.

Then, run the following command.

```
firebase serve --only hosting
```

You should get a similar response:

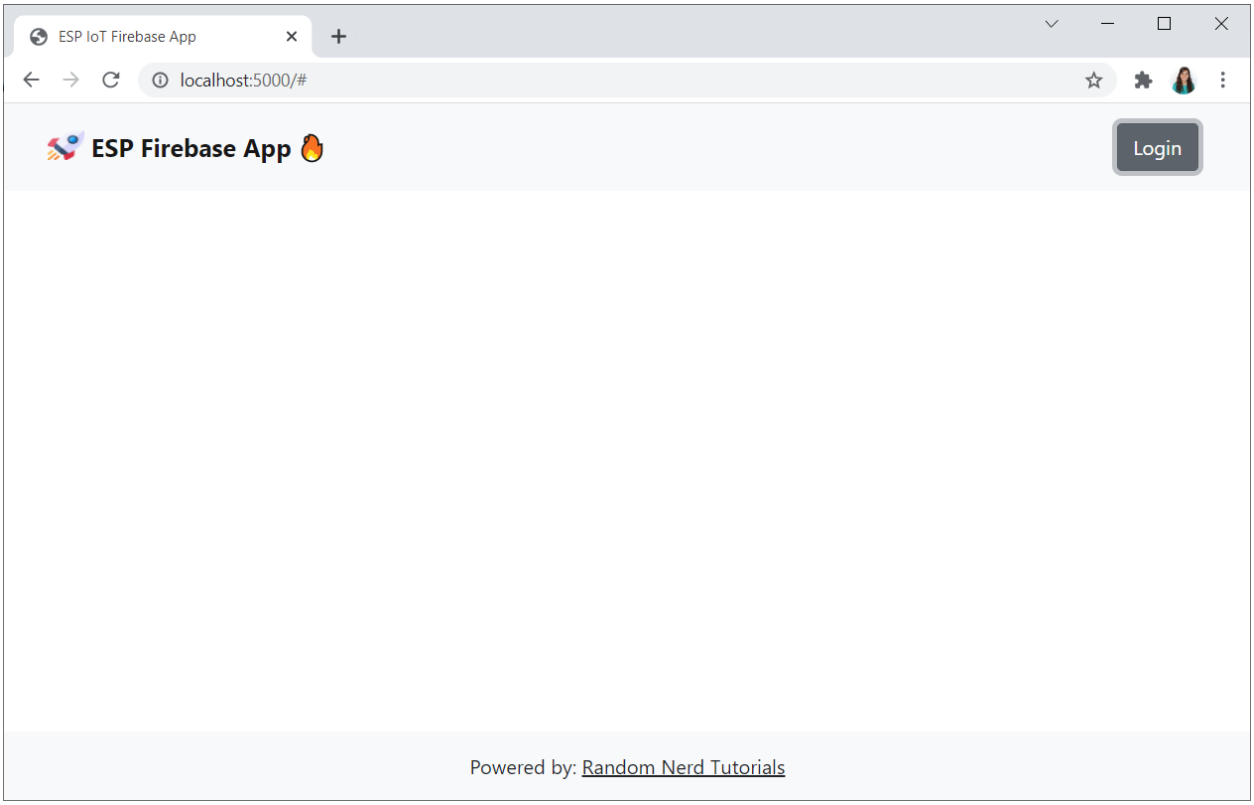


The web app should now be available on your local network at the following URL: `http://localhost:5000`. All the files located under the *public* folder are served.

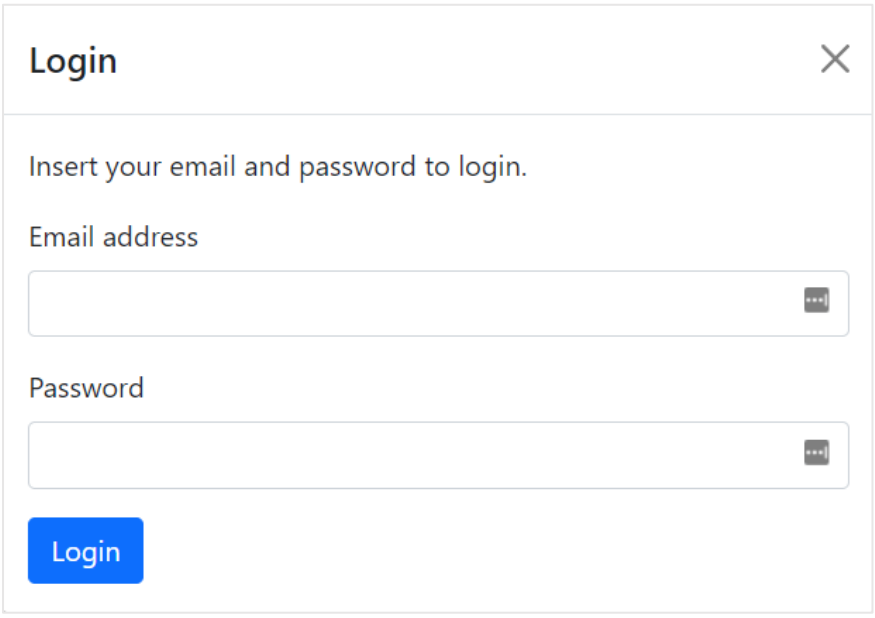
Open your browser on your local network and type

```
http://localhost:5000
```

You should get access to the following web app user interface (UI).



You can click on the **Login** button, and a new window will pop up to enter an email address and a password. At the moment, it isn't working yet.



We'll add some code to our JavaScript files, and everything will start working. We've only prepared the UI so far.

Setting up User Login (JavaScript)

Now let's implement user sign-in using Firebase authentication. We'll implement sign-in using email and password.

Copy the following to the *auth.js* file that you've created previously.

⇒ Download auth.js file

```
// listen for auth status changes
auth.onAuthStateChanged(user => {
  if (user) {
    console.log("user logged in");
    console.log(user);
    setupUI(user);
    var uid = user.uid;
    console.log(uid);
  } else {
    console.log("user logged out");
    setupUI();
  }
});

// login
const loginForm = document.querySelector('#login-form');
loginForm.addEventListener('submit', (e) => {
  e.preventDefault();
  // get user info
  const email = loginForm['input-email'].value;
  const password = loginForm['input-password'].value;
  // log the user in
  auth.signInWithEmailAndPassword(email, password).then((cred) => {
    // close the login modal & reset form
    $('#login-modal').modal("hide");
    loginForm.reset();
  });
});

// logout
const logout = document.querySelector('#logout');
logout.addEventListener('click', (e) => {
  e.preventDefault();
  auth.signOut();
});
```

Then, save the file. This file takes care of everything related to the login and logout of the user.

Login

The following lines are responsible for logging in the user.

```
const loginForm = document.querySelector('#login-form');
loginForm.addEventListener('submit', (e) => {
  e.preventDefault();
  // get user info
  const email = loginForm['input-email'].value;
  const password = loginForm['input-password'].value;
  // log the user in
  auth.signInWithEmailAndPassword(email, password).then((cred) => {
    // close the login modal & reset form
    $('#login-modal').modal("hide");
    loginForm.reset();
   });
});
```

We create a variable that refers to the login form HTML element called `loginForm`.

```
const loginForm = document.querySelector('#login-form');
```

If you go back to the *index.html* file, you can see that the form has the `login-form` id.

We add an event listener of type `submit` to the form. This means that the subsequent instructions will run whenever the form is submitted.

```
loginForm.addEventListener('submit', (e) => {
```

You can get the submitted data as follows.

```
const email = loginForm['input-email'].value;
const password = loginForm['input-password'].value;
```

If you go back to the HTML file, you'll see that the input fields contain the following ids `input-email` and `input-password` for the email and password, respectively.

Now that we have the inserted email and password, we can try to log in to Firebase. To do that, pass the user's email address and password to the following method: `signInWithEmailAndPassword`:

```
auth.signInWithEmailAndPassword(email, password).then((cred) => {
```

After signing in, we close the login window.

```
auth.signInWithEmailAndPassword(email, password).then((cred) => {
  // close the login modal & reset form
  $('#login-modal').modal("hide");
  loginForm.reset();
});
```

Logout

The following snippet is responsible for logging out the user.

```
const logout = document.querySelector('#logout');
logout.addEventListener('click', (e) => {
  e.preventDefault();
  auth.signOut();
});
```

When the user is logged in, a logout button is visible in the upper right corner. That button has the `logout` id (see on the HTML file). So, first, we create a variable called `logout` that refers to the logout button.

```
const logout = document.querySelector('#logout');
```

Then, we add an event listener of type `click`. This means the subsequent instructions will run whenever the button is clicked.

```
logout.addEventListener('click', (e) => {
```

When the button is clicked, we sign out the user with `signOut`.

```
auth.signOut();
```

Auth State Changes

To keep track of the user authentication state—to know if the user is logged in or logged out, there is a method called `onAuthSateChanged` that allows you to receive an event whenever the authentication state changes.

```
// listen for auth status changes
auth.onAuthStateChanged(user => {
  if (user) {
```

```
    console.log("user logged in");
    console.log(user);
    setupUI(user);
    var uid = user.uid;
    console.log(uid);
  } else {
    console.log("user logged out");
    setupUI();
  }
});
```

If the user returned is `null`, the user is currently signed out. Otherwise, it is currently signed in.

In both scenarios, we print the current user state to the console and call the `setupUI()` function. We haven't created that function yet (we'll create it in the next section), but it will be responsible for handling the user interface accordingly to the authentication state.

When the user is logged in, we pass the `user` as an argument to the `setupUI()` function. In this case, we'll display the complete user interface to control and monitor the ESP boards, as you'll see later.

```
console.log("user logged in");
console.log(user);
 setupUI(user);
```

We also get the user UID that we'll need later to insert and read data from the database.

```
var uid = user.uid;
console.log(uid);
```

If the user is logged out, we call the `setupUI()` function without any argument. In that scenario, we'll simply display a message informing that the user is logged out and doesn't have access to the interface (as we'll see later).

```
} else {
  console.log("user logged out");
  setupUI();
}
```

Testing the Login

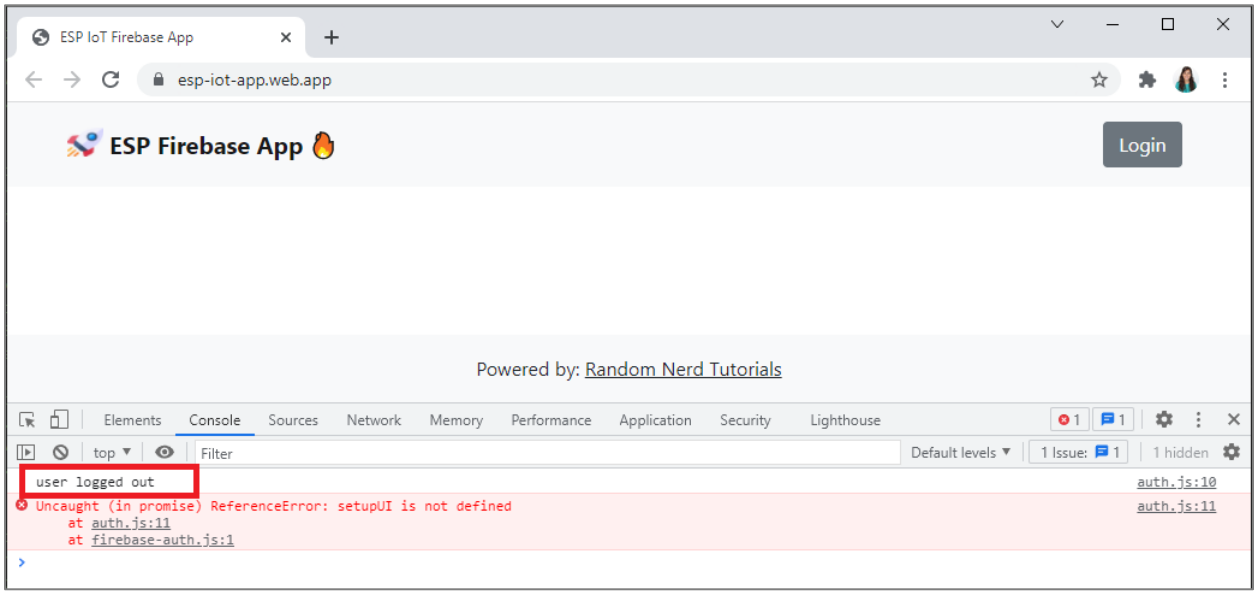
Save the file and go to the following URL to access the web page.

```
http://localhost:5000
```

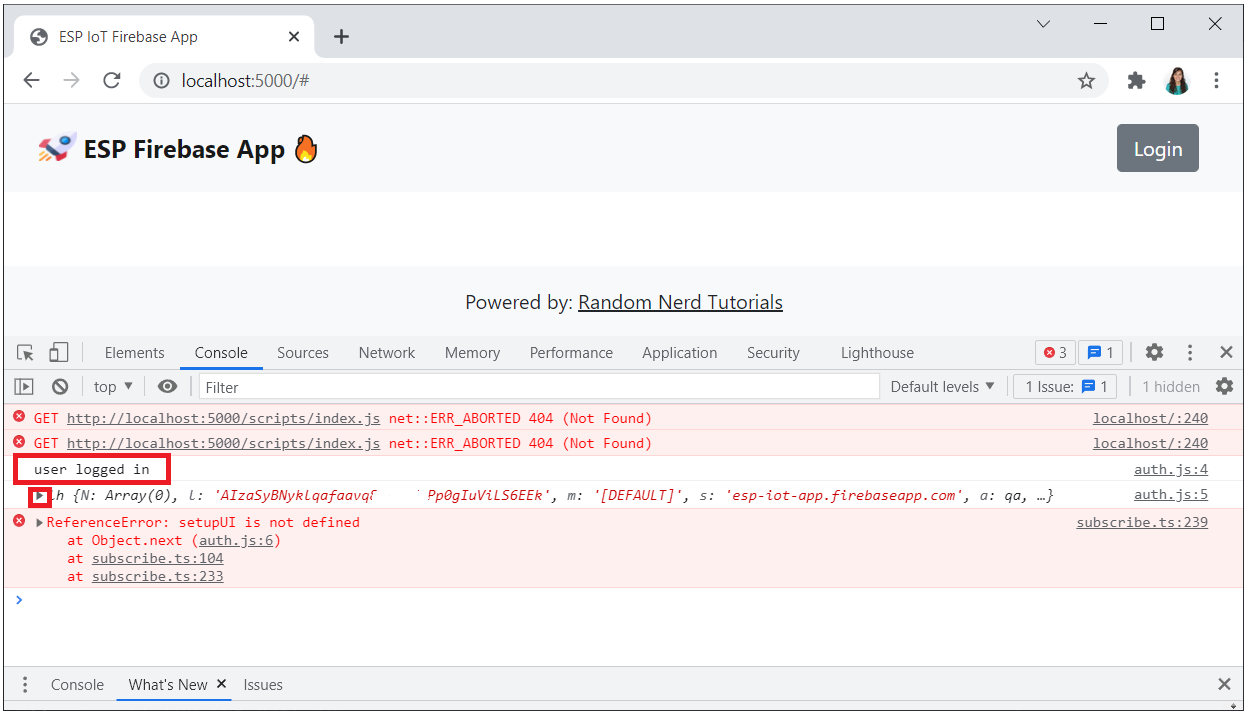
On your browser, hard refresh your web page by holding down **CTRL** and then pressing **F5**. If you're on a Mac, hold down **Cmd** and **Shift** and then press **R**.

After that, press **CTRL**+**SHIFT**+**J** on Windows or **Cmd**+**Option**+**J** on Mac to open the JavaScript console.

At the moment, it should display a "*user logged out*" message (ignore the other error messages). This means that the `onAuthStateChanged` function is working.



Now, click on the **Login** button and enter the email and password of the authorized user—the one you created in this section: Set Authentication Methods. Now, it should say *user logged in* and display several details about the user.



If the web page is not showing, you may have accidentally closed the local web server. So, you'll need to type the following command on the VS Code Terminal again:

```
firebase serve --only hosting
```

The authentication part of the project is done. Now, we need to handle the UI: display the web page content if the user is logged in and allow the user to read and write from the database to show sensor readings and control the ESP32/ESP8266 GPIOs. And in case the user is logged out, hide the user interface and show a message.

Setting up UI (JavaScript)

Copy the following to the *index.js* file that you've created previously.

⇒ Download index.js file

```
// DOM elements
const logoutNavElement = document.querySelector('#logout-link');
const loginNavElement = document.querySelector('#login-link');
const signedOutMessageElement = document.querySelector('#signedOutMessage');
const dashboardElement = document.querySelector("#dashboardSignedIn");
const userDetailsElement = document.querySelector('#user-details');
```

```
// MANAGE LOGIN/LOGOUT UI
const setupUI = (user) => {
  if (user) {
    //toggle UI elements
    logoutNavElement.style.display = 'block';
    loginNavElement.style.display = 'none';
    signedOutMessageElement.style.display ='none';
    dashboardElement.style.display = 'block';
    userDetailsElement.style.display ='block';
    userDetailsElement.innerHTML = user.email;

    // get user UID to get data from database
    var uid = user.uid;
    console.log(uid);

    // Database paths (with user UID)
    var dbPathBtn1 = 'UsersData/' + uid.toString() + '/outputs/digital/2';
    var dbPathBtn2 = 'UsersData/' + uid.toString() + '/outputs/digital/12';
    var dbPathSlider1 = 'UsersData/' + uid.toString() + '/outputs/pwm/13';
    var dbPathSlider2 = 'UsersData/' + uid.toString() + '/outputs/pwm/14';
    var dbPathBmeTemp = 'UsersData/' + uid.toString() + '/sensor/temperature';
    var dbPathBmeHum = 'UsersData/' + uid.toString() + '/sensor/humidity';
    var dbPathBmePres = 'UsersData/' + uid.toString() + '/sensor/pressure';
    var dbPathInput1 = 'UsersData/' + uid.toString() + '/outputs/message';

    //////// Button 1 - GPIO 2 ////////
    var btn1State = document.getElementById('btn1State');
    var dbBtn1 = firebase.database().ref().child(dbPathBtn1);

    // Button 1 - GPIO 2 - Update state message on web page
    dbBtn1.on('value', snap => {
      if(snap.val()==1) {
        btn1State.innerText = "ON";
      }
      else {
        btn1State.innerText = "OFF";
      }
    });

    // Button 1 - GPIO 2 - Update database upon button click
    const btn1On = document.getElementById('btn1On');
    const btn1Off = document.getElementById('btn1Off');

    btn1On.onclick = () => {
      firebase.database().ref().child(dbPathBtn1).set(1);
    }
    btn1Off.onclick = () => {
      firebase.database().ref().child(dbPathBtn1).set(0);
    }

    ////////  Button 2 - GPIO 12 ////////
    var btn2State = document.getElementById('btn2State');
    var dbBtn2 = firebase.database().ref().child(dbPathBtn2);
```

```
    // Button 2 - GPIO 12 - Update state message on web page
    dbBtn2.on('value', snap => {
      if(snap.val()==1) {
          btn2State.innerText = "ON";
      }
      else {
          btn2State.innerText = "OFF";
      }
    });

    // Button 2 - GPIO 12 - Update database upon button click
    const btn2On = document.getElementById('btn2On');
    const btn2Off = document.getElementById('btn2Off');

    btn2On.onclick = () => {
      firebase.database().ref().child(dbPathBtn2).set(1);
    }
    btn2Off.onclick = () => {
      firebase.database().ref().child(dbPathBtn2).set(0);
    }

    //////// Sensor Readings - BME Temperature - Update web page with new values
from database ////////
    var dbBmeTemp = firebase.database().ref().child(dbPathBmeTemp);
    const bmeTemp = document.getElementById('bmeTemp');

    dbBmeTemp.on('value', snap => {
      // Celsius degrees
      bmeTemp.innerText = snap.val().toFixed(2) + " ºC";
      // Fahrenheit degrees
      //bmeTemp.innerText = snap.val().toFixed(2) + " ºF";
    });

    //////// Sensor Readings - BME Humidity - Update web page with new values from
database ////////
    var dbBmeHumi = firebase.database().ref().child(dbPathBmeHum);
    const bmeHumi = document.getElementById('bmeHumi');

    dbBmeHumi.on('value', snap => {
      bmeHumi.innerText = snap.val().toFixed(2) + " %";
    });

    //////// Sensor Readings - BME Pressure - Update web page with new values from
database ////////
    var dbBmePres = firebase.database().ref().child(dbPathBmePres);
    const bmePres = document.getElementById('bmePres');

    dbBmePres.on('value', snap => {
      bmePres.innerText = snap.val().toFixed(2) + " hPa";
    });

    ////////  Slider 1 - GPIO 13  ////////
    var sld1Value = document.getElementById('sld1Value');
    var dbSld1 = firebase.database().ref().child(dbPathSlider1);
```

```
    const sld1 = document.getElementById('sld1');

    // Slider 1 - GPIO 13 - Update slider text value on web page
    dbSld1.on('value', snap => {
      sld1Value.innerText = snap.val() + " %";
      sld1.value = snap.val();
    });

    // Slider 1 - GPIO 13 - Update database slider value
    sld1.onchange = () => {
      firebase.database().ref().child(dbPathSlider1).set(Number(sld1.value));
    }

    //////// Slider 2 - GPIO 14 ////////
    var sld2Value = document.getElementById('sld2Value');
    var dbSld2 = firebase.database().ref().child(dbPathSlider2);
    const sld2 = document.getElementById('sld2');

    // Slider 2 - GPIO 14 - Update slider text value on web page
    dbSld2.on('value', snap => {
      sld2Value.innerText = snap.val() + " %";
      sld2.value = snap.val();
    });

    // Slider 2 - GPIO 14 - Update database slider value
    sld2.onchange = () => {
      firebase.database().ref().child(dbPathSlider2).set(Number(sld2.value));
    }

    //////// Input 1 - Message ////////
    var dbInput1 = firebase.database().ref().child(dbPathInput1);
    const input1 = document.getElementById('input1');
    const input1Text = document.getElementById('input1Text');
    // Input 1 - Update input text on web page
    dbInput1.on('value', snap => {
      input1Text.innerText = snap.val();
    });
    // Input 1 - Update database input 1 value
    input1.onchange = () => {
      firebase.database().ref().child(dbPathInput1).set(input1.value);
    }

  // if user is logged out
  } else{
    // toggle UI elements
    logoutNavElement.style.display = 'none';
    loginNavElement.style.display = 'block';
    signedOutMessageElement.style.display ='block';
    dashboardElement.style.display = 'none';
    userDetailsElement.style.display ='none';
  }
}
```

Let's take a quick look at the JavaScript file.

Getting HTML Elements

First, we create variables to refer to several elements on the UI interface by referring to their ids. To identify these elements, we recommend that you take a look at the HTML file provided and find the elements with the refered ids.

```
const logoutNavElement = document.querySelector('#logout-link');
const loginNavElement = document.querySelector('#login-link');
const signedOutMessageElement = document.querySelector('#signedOutMessage');
const dashboardElement = document.querySelector("#dashboardSignedIn");
const userDetailsElement = document.querySelector('#user-details');
```

The `logoutNavElement` corresponds to the section of the navigation bar that displays the logout button. The `loginNavElement` corresponds to the section of the navigation bar that displays the login button.

The `signedOutMessageElement` corresponds to the section of the web page that displays a logged out message.

The `dashboardElement` contains the user interface with buttons, sliders, table, etc., when the user is logged in.

When the user is logged in, we display its email on the navigation bar. The `userDetailsElement` refers to that section of the navigation bar.

setupUI() Function

Then, we create the `setupUI()` function that will handle the UI accordingly to the state of the user authentication.

In the *auth.js* file, we called the `setupUI()` function with the `user` argument `setupUI(user)` if the user is logged in; or the function without argument `setupUI()` when the user is logged out.

So, let's check what happens when the user is logged in.

```
if (user) {
```

We define which parts of the UI should be visible or invisible. When the user is logged in, we want to show the logout button to log out at any time. To do that, we can set its display style to `block`.

```
logoutNavElement.style.display = 'block';
```

We hide the login button because the user is already logged in. To hide an element, we can set the display style to `none`.

```
loginNavElement.style.display = 'none';
```

We also hide the logout message.

```
signedOutMessageElement.style.display ='none';
```

The dashboard that contains all the controls should be visible as well as the section to show the logged in user details (the email).

```
dashboardElement.style.display = 'block';
userDetailsElement.style.display ='block';
```

Finally, we can get the logged in user email with `user.email` and display it in the `userDetailsElement` section as follows:

```
userDetailsElement.innerHTML = user.email;
```

User UID and Database Paths

After we have a logged-in user, we can get its UID with `user.uid`.

```
// get user UID to get data from database
var uid = user.uid;
console.log(uid);
```

After getting the user UID, we create variables to refer to the database paths where we save the data.

These are the database paths for the digital output pins (GPIOs 2 and 12).

```
var dbPathBtn1 = 'UsersData/' + uid.toString() + '/outputs/digital/2';
var dbPathBtn2 = 'UsersData/' + uid.toString() + '/outputs/digital/12';
```

The following refers to the database paths for the slider values to control GPIOs 13 and 14.

```
var dbPathSlider1 = 'UsersData/' + uid.toString() + '/outputs/pwm/13';
var dbPathSlider2 = 'UsersData/' + uid.toString() + '/outputs/pwm/14';
```

The following database paths contain the sensor readings.

```
var dbPathBmeTemp = 'UsersData/' + uid.toString() + '/sensor/temperature';
var dbPathBmeHum = 'UsersData/' + uid.toString() + '/sensor/humidity';
var dbPathBmePres = 'UsersData/' + uid.toString() + '/sensor/pressure';
```

And finally, the following database path contains the message we want to display on the OLED.

```
var dbPathInput1 = 'UsersData/' + uid.toString() + '/outputs/message';
```

Handle Buttons

The following section handles what happens when you click on the button that controls GPIO 2.

```
//////// Button 1 - GPIO 2 ////////
var btn1State = document.getElementById('btn1State');
var dbBtn1 = firebase.database().ref().child(dbPathBtn1);
// Button 1 - GPIO 2 - Update state message on web page
dbBtn1.on('value', snap => {
  if(snap.val()==1) {
    btn1State.innerText = "ON";
  }
  else {
    btn1State.innerText = "OFF";
  }
});
// Button 1 - GPIO 2 - Update database upon button click
const btn1On = document.getElementById('btn1On');
const btn1Off = document.getElementById('btn1Off');
btn1On.onclick = () => {
  firebase.database().ref().child(dbPathBtn1).set(1);
}
btn1Off.onclick = () => {
  firebase.database().ref().child(dbPathBtn1).set(0);
}
```

First, we create a variable that refers to the section of the HTML page that displays the button state (ON or OFF).

```
var btn1State = document.getElementById('btn1State');
```

Then, you create a reference for the firebase database path that contains the button1 (GPIO 2) state—we called it `dbBtn1`.

```
var dbBtn1 = firebase.database().ref().child(dbPathBtn1);
```

Update the GPIO State

The following line listens for changes on that database path.

```
dbBtn1.on('value', snap => {
```

We can get the value of that path with `snap.val()`. If the value is 1, the GPIO state is ON. So, we change the `btn1State` content to "`ON`".

```
if(snap.val()==1) {
  btn1State.innerText = "ON";
}
```

Otherwise, the GPIO is OFF, so we change the state to "`OFF`".

```
else {
  btn1State.innerText = "OFF";
}
```

Change Database Values

We also need to handle what happens when you click on the ON or OFF buttons. You want to change the value saved on the GPIO database path depending on the button clicked (OFF→0; ON→1).

First, you get a reference to the buttons by referring to their ids.

```
// Button 1 - GPIO 2 - Update database upon button click
const btn1On = document.getElementById('btn1On');
const btn1Off = document.getElementById('btn1Off');
```

Then, we define what happens on we click on the buttons.

Basically, when you click on the ON button, you set a new value on that database path, in this case `1`.

```
btn1On.onclick = () => {
  firebase.database().ref().child(dbPathBtn1).set(1);
}
```

When you click on the OFF button, you set `0` as the new value for that path.

```
btn1Off.onclick = () => {
  firebase.database().ref().child(dbPathBtn1).set(0);
}
```

We follow a similar procedure for GPIO 12 buttons, but we change the ids and database path accordingly.

```
////////  Button 2 - GPIO 12 ////////
var btn2State = document.getElementById('btn2State');
var dbBtn2 = firebase.database().ref().child(dbPathBtn2);

// Button 2 - GPIO 12 - Update state message on web page
dbBtn2.on('value', snap => {
  if(snap.val()==1) {
    btn2State.innerText = "ON";
  }
  else {
    btn2State.innerText = "OFF";
  }
});
// Button 2 - GPIO 12 - Update database upon button click
const btn2On = document.getElementById('btn2On');
const btn2Off = document.getElementById('btn2Off');

btn2On.onclick = () => {
  firebase.database().ref().child(dbPathBtn2).set(1);
}
btn2Off.onclick = () => {
  firebase.database().ref().child(dbPathBtn2).set(0);
}
```

Display Sensor Readings

The following snipet is responsible for updating the table with new sensor readings from the database.

```
//////// Sensor Readings - BME Temperature - Update web page with new values from database ////////
var dbBmeTemp = firebase.database().ref().child(dbPathBmeTemp);
const bmeTemp = document.getElementById('bmeTemp');
dbBmeTemp.on('value', snap => {
  // Celsius degrees
  bmeTemp.innerText = snap.val().toFixed(2) + " ºC";
  // Fahrenheit degrees
  //bmeTemp.innerText = snap.val().toFixed(2) + " ºF";
});

//////// Sensor Readings - BME Humidity - Update web page with new values from database ////////
var dbBmeHumi = firebase.database().ref().child(dbPathBmeHum);
const bmeHumi = document.getElementById('bmeHumi');

dbBmeHumi.on('value', snap => {
  bmeHumi.innerText = snap.val().toFixed(2) + " %";
});

//////// Sensor Readings - BME Pressure - Update web page with new values from database ////////
var dbBmePres = firebase.database().ref().child(dbPathBmePres);
const bmePres = document.getElementById('bmePres');

dbBmePres.on('value', snap => {
  bmePres.innerText = snap.val().toFixed(2) + " hPa";
});
```

Let's take a quick look at how to update the temperature value on the table (it is similar for humidity and pressure).

First, we create a reference to the database path that contains the temperature values.

```
var dbBmeTemp = firebase.database().ref().child(dbPathBmeTemp);
```

Then, create a variable that corresponds to the HTML elements where we'll insert the temperature readings—it has the `bmeTemp` id.

```
const bmeTemp = document.getElementById('bmeTemp');
```

The following line detects whenever there's a new value on the database path:

```
dbBmeTemp.on('value', snap => {
```

When that happens, we get the new value and update the interface.

```
// Celsius degrees
bmeTemp.innerText = snap.val().toFixed(2) + " ºC";
// Fahrenheit degrees
//bmeTemp.innerText = snap.val().toFixed(2) + " ºF";
});
```

Handle Sliders

The following snipet is responsible for handling the sliders: update the slider value on the web page, and write new slider values to the database.

```
////////  Slider 1 - GPIO 13  ////////
var sld1Value = document.getElementById('sld1Value');
var dbSld1 = firebase.database().ref().child(dbPathSlider1);
const sld1 = document.getElementById('sld1');
// Slider 1 - GPIO 13 - Update slider text value on web page
dbSld1.on('value', snap => {
  sld1Value.innerText = snap.val() + " %";
  sld1.value = snap.val();
});

// Slider 1 - GPIO 13 - Update database slider value
sld1.onchange = () => {
  firebase.database().ref().child(dbPathSlider1).set(Number(sld1.value));
}

//////// Slider 2 - GPIO 14 ////////
var sld2Value = document.getElementById('sld2Value');
var dbSld2 = firebase.database().ref().child(dbPathSlider2);
const sld2 = document.getElementById('sld2');

// Slider 2 - GPIO 14 - Update slider text value on web page
dbSld2.on('value', snap => {
  sld2Value.innerText = snap.val() + " %";
  sld2.value = snap.val();
});

// Slider 2 - GPIO 14 - Update database slider value
sld2.onchange = () => {
  firebase.database().ref().child(dbPathSlider2).set(Number(sld2.value));
}
```

Let's just look at slider 1. It works similarly to slider 2.

The `sld1Value` refers to the HTML section that displays the value selected on the slider.

```
var sld1Value = document.getElementById('sld1Value');
```

The `dbSld1` is a reference to the slider 1 database path.

```
var dbSld1 = firebase.database().ref().child(dbPathSlider1);
```

The `sld1` refers to the slider element itself.

```
const sld1 = document.getElementById('sld1');
```

The following line listens for changes on the database slider path.

```
dbSld1.on('value', snap => {
```

Whenever there's a change on the value, we update the value displayed on the web page as well as the slider position.

```
sld1Value.innerText = snap.val() + " %";
sld1.value = snap.val();
```

Whenever you move the slider, it updates the value on the database.

```
// Slider 1 - GPIO 13 - Update database slider value
sld1.onchange = () => {
  firebase.database().ref().child(dbPathSlider1).set(Number(sld1.value));
}
```

Input Message Field

The following snippet handles what happens when you insert a new value on the input field.

```
//////// Input 1 - Message ////////
var dbInput1 = firebase.database().ref().child(dbPathInput1);
const input1 = document.getElementById('input1');
const input1Text = document.getElementById('input1Text');
// Input 1 - Update input text on web page
dbInput1.on('value', snap => {
  input1Text.innerText = snap.val();
});
// Input 1 - Update database input 1 value
input1.onchange = () => {
  firebase.database().ref().child(dbPathInput1).set(input1.value);
}
```

The `dbInput1` variable is a reference to the database.

```
var dbInput1 = firebase.database().ref().child(dbPathInput1);
```

The `input1` variable refers to the input field—the place where you write the message.

```
const input1 = document.getElementById('input1');
```

The `input1Text` refers to the HTML section that displays the current message.

```
const input1Text = document.getElementById('input1Text');
```

Whenever there's a change on the message database path, update the message text with the new value.

```
dbInput1.on('value', snap => {
  input1Text.innerText = snap.val();
});
```

Finally, save the new message to the database when a new message is written on the input field.

```
input1.onchange = () => {
  firebase.database().ref().child(dbPathInput1).set(input1.value);
}
```

Logged Out UI

The following snippet handles the UI when the user logs out.

```
// if user is logged out
} else{
  // toggle UI elements
  logoutNavElement.style.display = 'none';
  loginNavElement.style.display = 'block';
  signedOutMessageElement.style.display ='block';
  dashboardElement.style.display = 'none';
  userDetailsElement.style.display ='none';
}
```

When a user is logged out, hide the logout button and show the login button.

```
logoutNavElement.style.display = 'none';
loginNavElement.style.display = 'block';
```

Display the logged out message and hide all the dashboard elements, including the user details.

```
signedOutMessageElement.style.display ='block';
dashboardElement.style.display = 'none';
userDetailsElement.style.display ='none';
```

Testing the Web App User Interface

Save the *index.js* file.

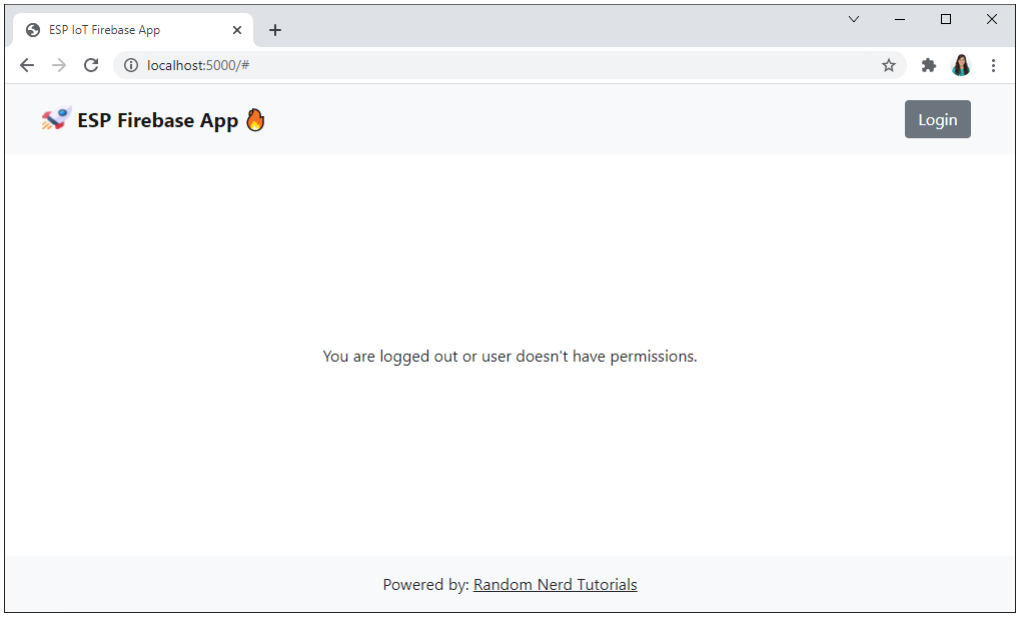
Go to the following URL to access the web page.

```
http://localhost:5000
```

Hard refresh your web page by holding down **CTRL** and then pressing **F5**. If you're on MAC, hold down **Cmd** and **Shift** and then press **R**. After that, press **CTRL**+**SHIFT**+**J** on Windows or **Cmd**+**Option**+**J** on MAC to open the JavaScript console to check if everything is going as expected.

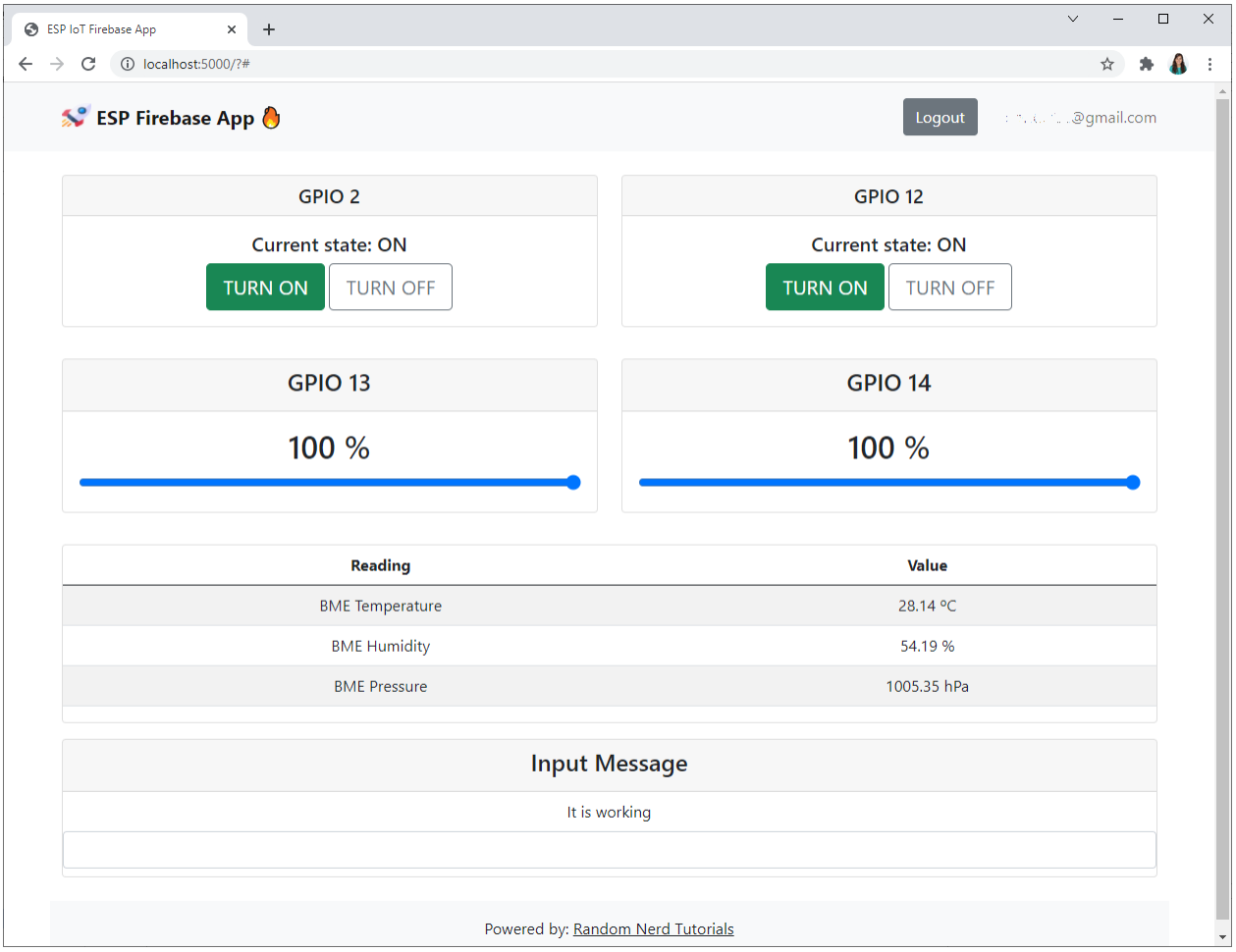
You should get access to the dashboard to control your ESP32 or ESP8266 board.

Test the interface. Test the login and logout buttons. When you are logged out, you can't access the dashboard, and it should display the following message.



```
"You are logged out or user doesn't have permissions."
```

If you login with the authorized user, you should get access to the dashboard.



With the ESP running the code provided in the "Code - Streaming Database" section, you should be able to control it and monitor the sensor readings through the app interface.

Click the ON and OFF buttons to check if the LEDs turn on and off. Drag the sliders to control the brightness of the LEDs. Check that new readings arrive every three minutes. Finally, write a message on the input field to display on the OLED.

At the same time, you can check that the app writes new values to the database.

If you open Arduino IDE, you should see that it detects whenever there's a change and sends new sensor readings.

Deploy Your App

After testing that everything is working as expected, you can deploy your web app to Firebase.

Firebase offers a free hosting service to serve your assets and web apps. Then, you can access your web app from anywhere.

To do that, you need to deploy your app. With the Firebase project opened on VS Code, open a new Terminal window and run the following command.

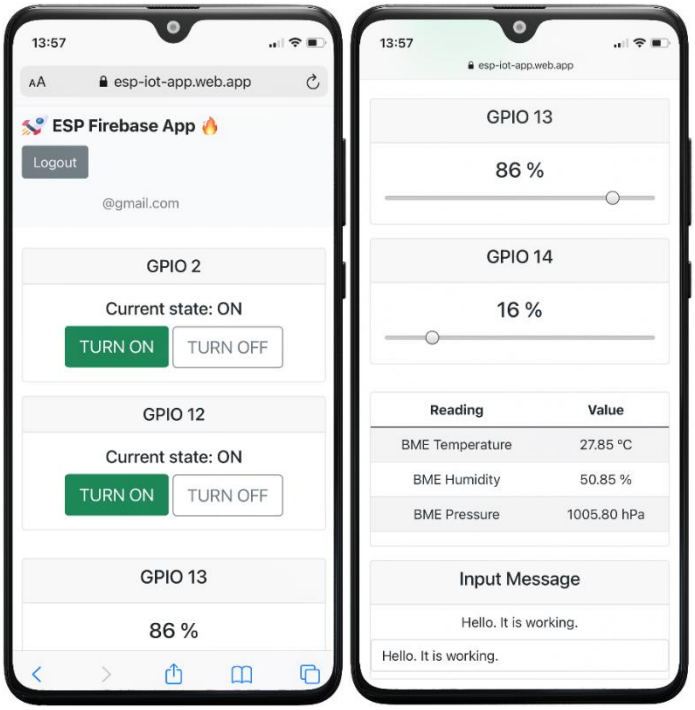
```
firebase deploy
```

The console should display the following:

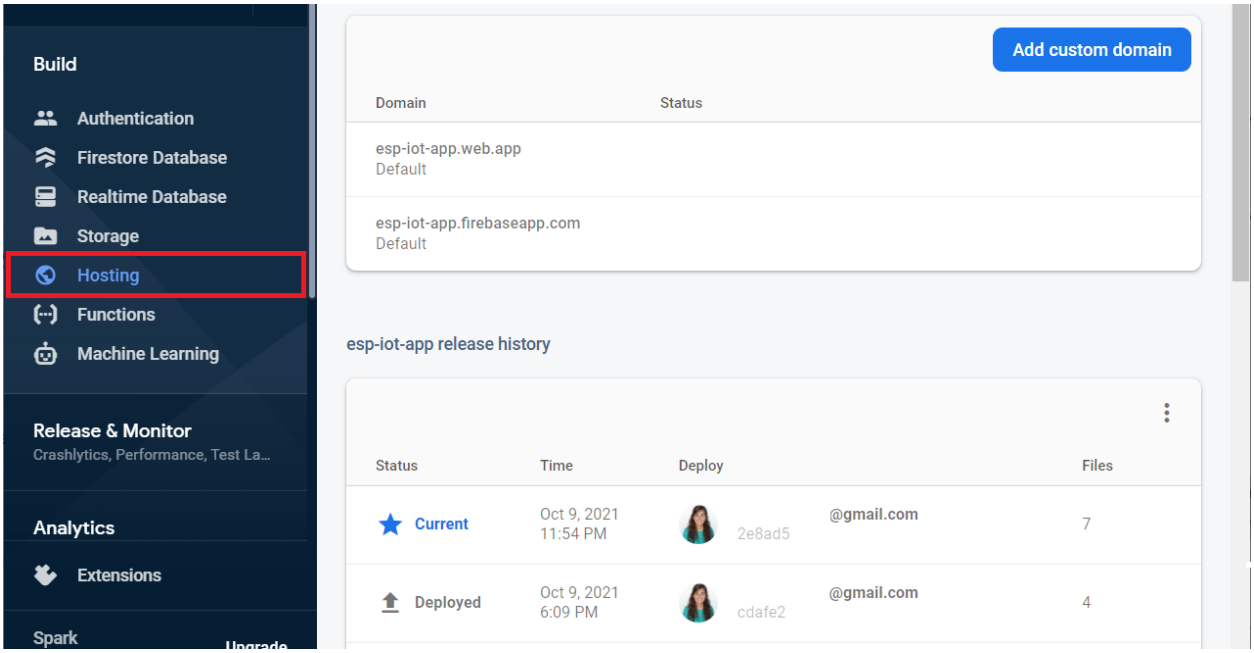
```
PS C:\Users\sarin\Desktop\Firebase-Project> firebase deploy

=== Deploying to 'esp-iot-app'...

i  deploying database, hosting
i  database: checking rules syntax...
+  database: rules syntax for database esp-iot-app-default-rtdb is valid
i  hosting[esp-iot-app]: beginning deploy...
i  hosting[esp-iot-app]: found 5 files in public
+  hosting[esp-iot-app]: file upload complete
i  database: releasing rules...
+  database: rules for database esp-iot-app-default-rtdb released successfully
i  hosting[esp-iot-app]: finalizing version...
+  hosting[esp-iot-app]: version finalized
i  hosting[esp-iot-app]: releasing new version...
+  hosting[esp-iot-app]: release complete

+  Deploy complete!

Project Console: https://console.firebase.google.com/project/esp-iot-app/overview
Hosting URL: https://esp-iot-app.web.app
PS C:\Users\sarin\Desktop\Firebase-Project>
```

Congratulations! You successfully deployed your app. It is now hosted on a global CDN using Firebase hosting. You can access your web app from anywhere on the Hosting URL provided. In my case, it is `https://esp-iot-app-web.app`.

The web app is responsive, and you can access it using your smartphone. Now you have complete control over your ESP32/ESP8266 from anywhere in the world.



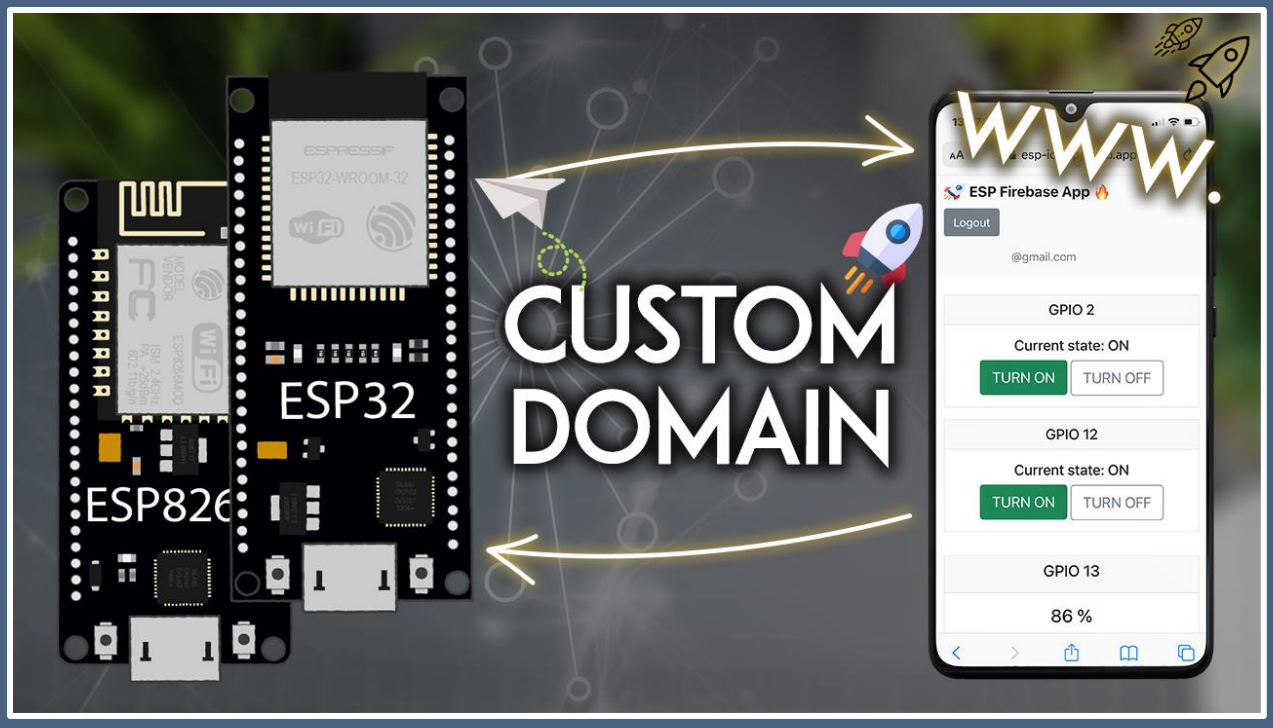
Go to your project's Firebase console **Hosting** tab. You can see your app domains, deploy history, and you can even roll back to previous versions of your app.



Instead of going to the URL provided by Firebase to access your app, you can add a custom domain name. Go to the next part to learn how to add a custom domain name to your web app.

PART 5

Hosting Your Web App (Custom Domain Name)



At this moment, your web app is fully functional, and you can access it from anywhere using the Firebase-generated domains. But if you want to take a step further and access your web app through your custom domain name, follow the steps in this part.

5.1 - Adding a Custom Domain

Follow this section to add a custom domain to your app.

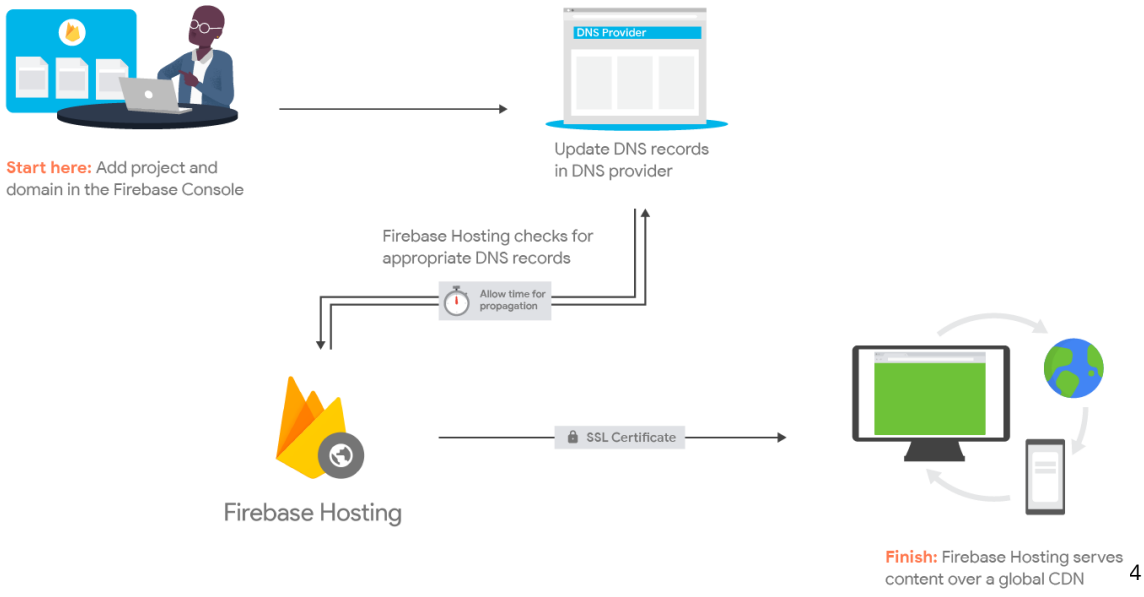
Buy a Domain Name

Before proceeding, you need to register a domain name. There are many places where you can buy a domain name. If you search "buy domain name" on Google, you'll find several options. We suggest godaddy.com, or namecheap.com. But you can register a domain name from any other domain provider.

We'll use a domain name that we bought on godaddy.com, but the instructions are similar for other domain providers.

Connect a Custom Domain

Firebase Hosting provisions an SSL certificate for each of your domains and serves your content over a global CDN.

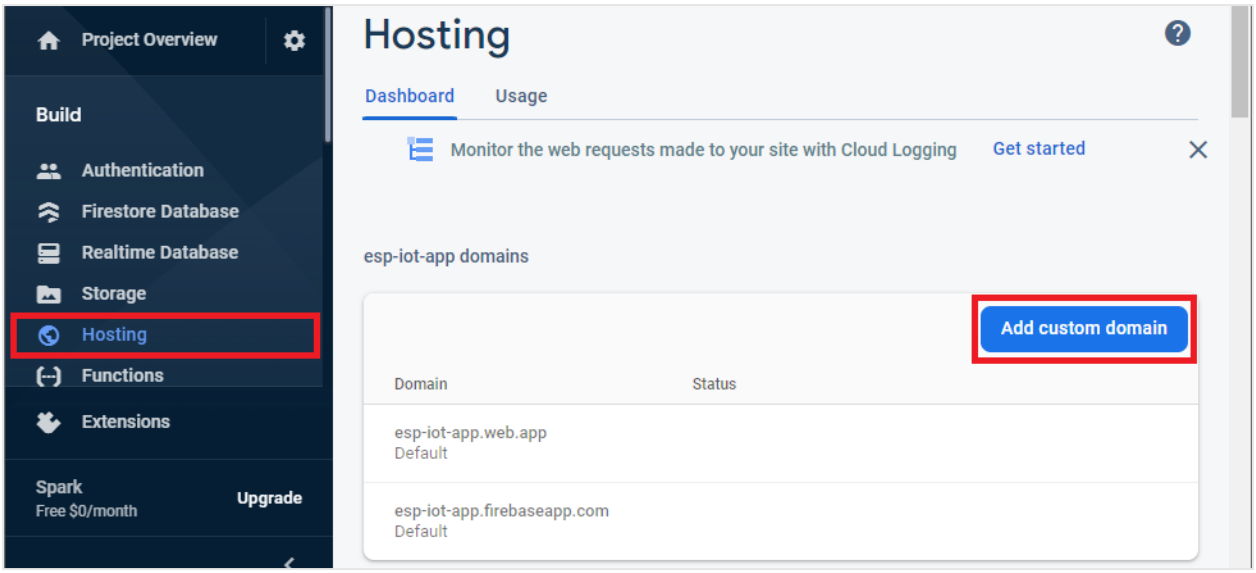


[4] Image source: https://firebase.google.com/docs/hosting/custom-domain

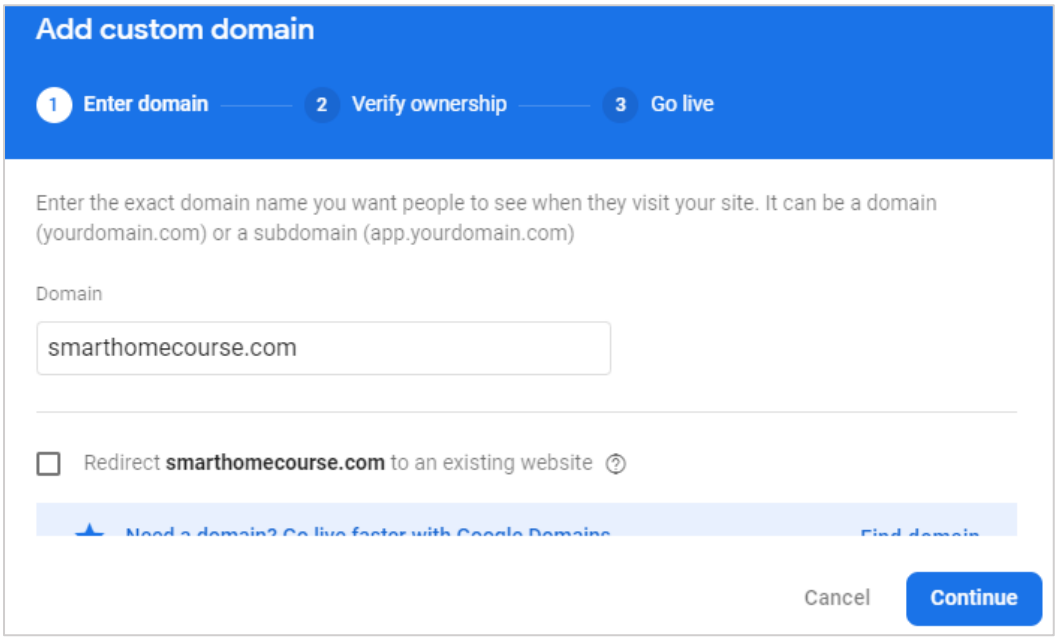
Adding a custom domain name to your app is very simple. You can follow our instructions, or you can follow the official instructions at the following link:

- https://firebase.google.com/docs/hosting/custom-domain

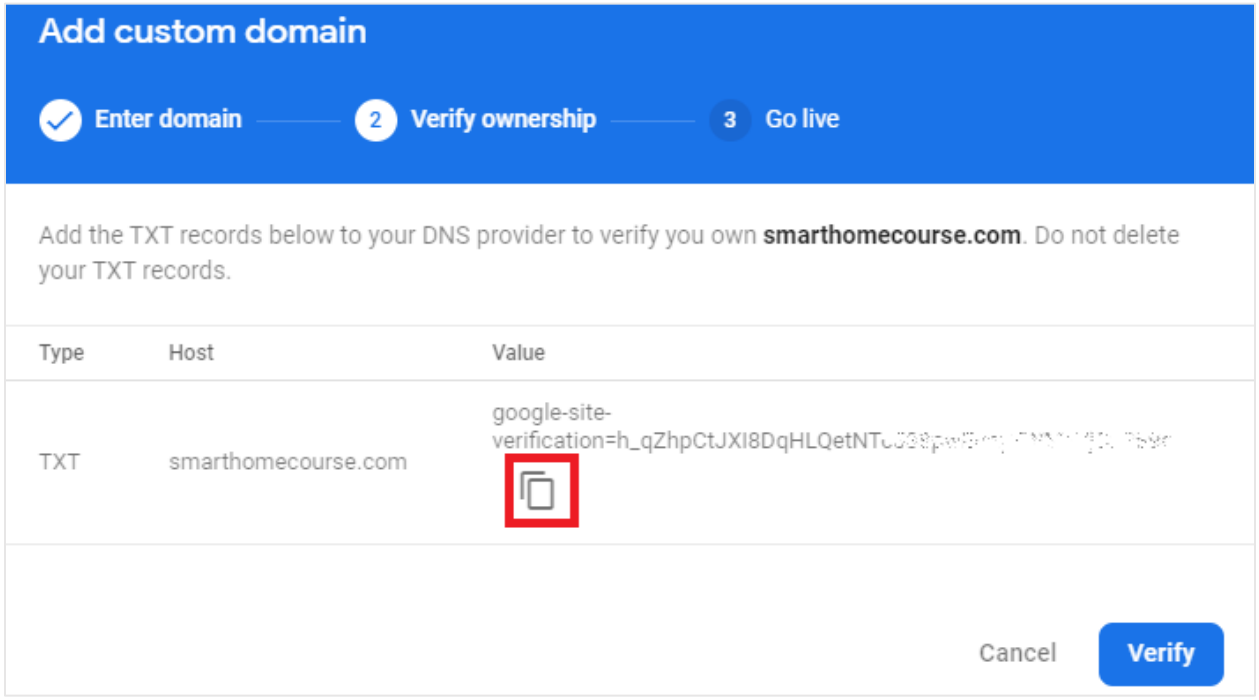
1) Go to your Firebase project console, select the **Hosting** tab and then, click on the **Add custom domain** button.



2) Insert your domain name. In my case, I'll use the domain name `smarthomecourse.com` that I bought on GoDaddy. Then, click on **Continue**.



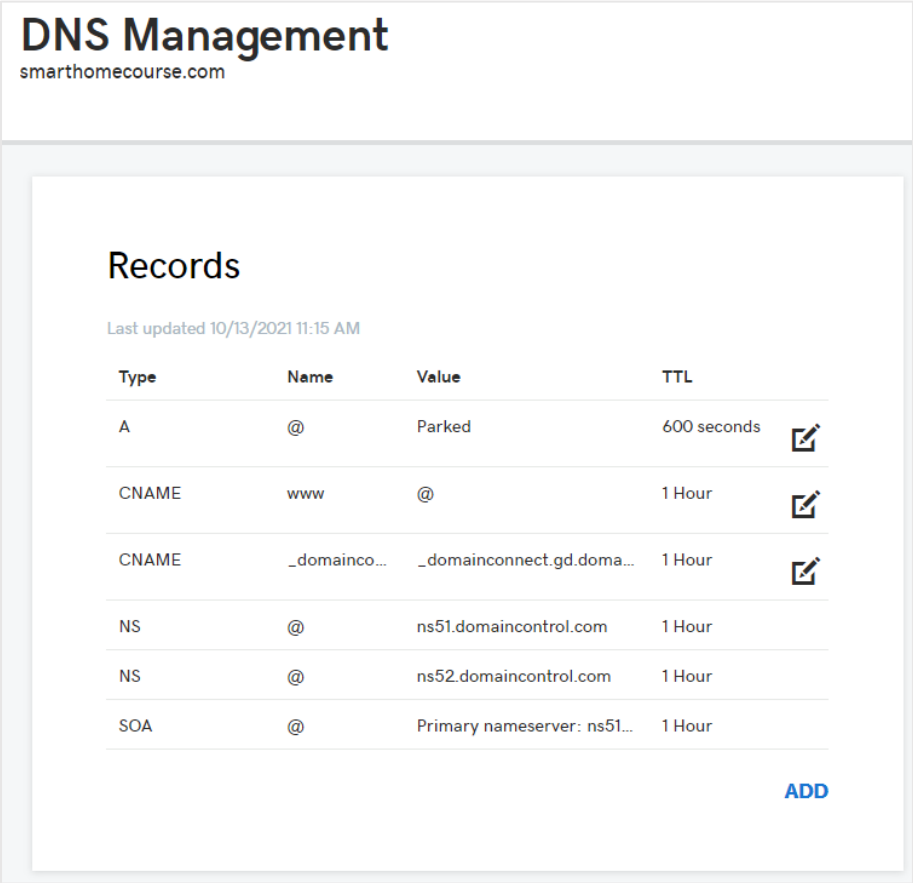
3) The new window shows the TXT records. Click on the copy icon to copy it to the clipboard. Firebase Hosting checks for this value to prove your domain ownership.



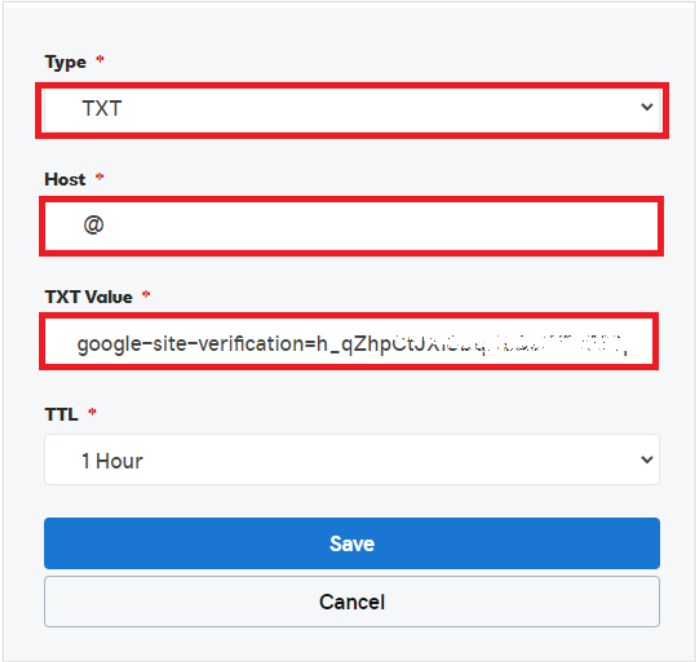
4) Go to your account on your domain provider. In my case, it is GoDaddy.

Click on the **DNS** link to go to the DNS management section and add a new record. The interface might be different depending on your domain provider, but the instructions are the same—you have to look for the DNS management section.

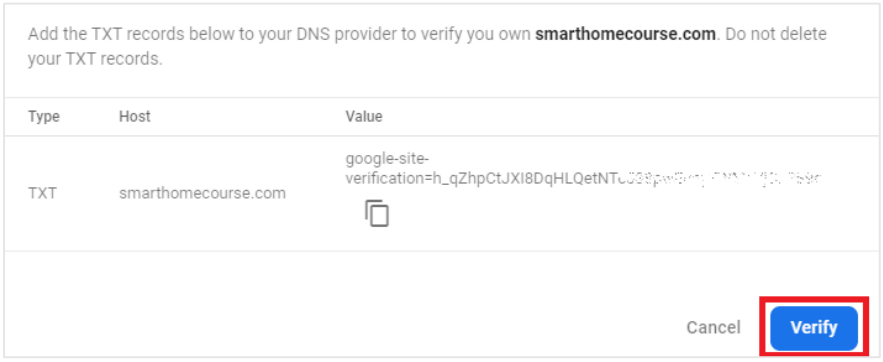
5) Once you're on the DNS Management section, click on **ADD** to add a new record.



6) Then, select the options in the picture below. The TXT Value is the value you copied from the Firebase console in step **3).** Then, click **Save**.

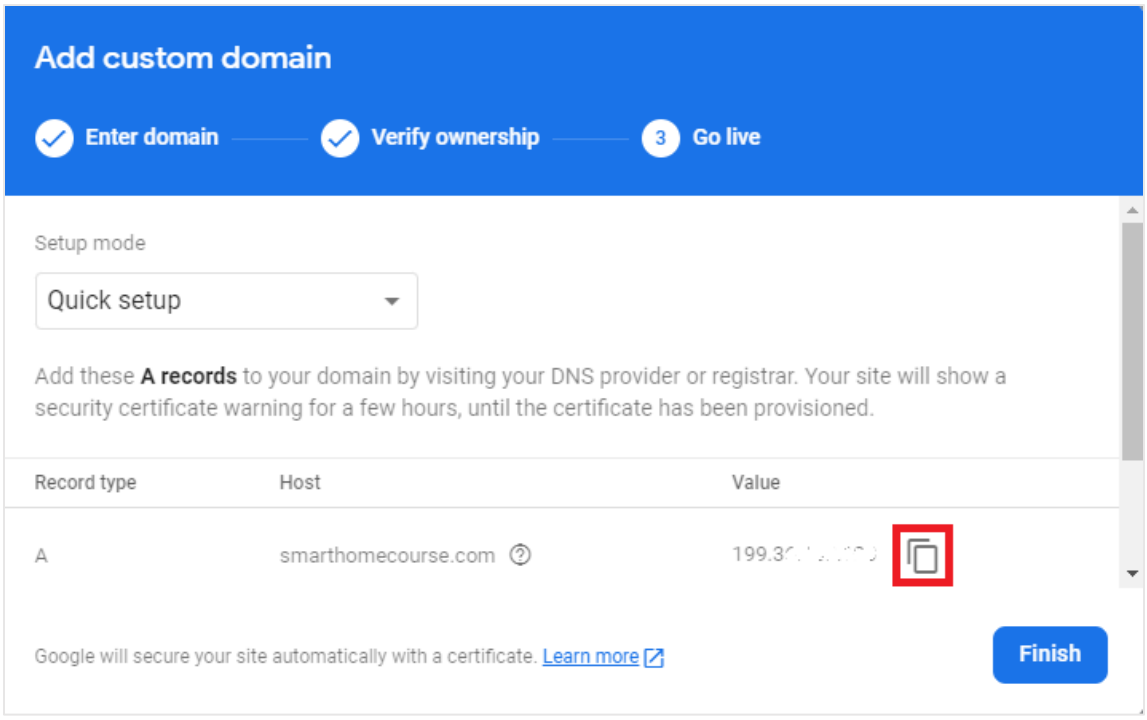


7) Go back to Firebase and click on **Verify**. It might take up to 24 hours for the propagation of your updated TXT records, but it usually only takes a few minutes.

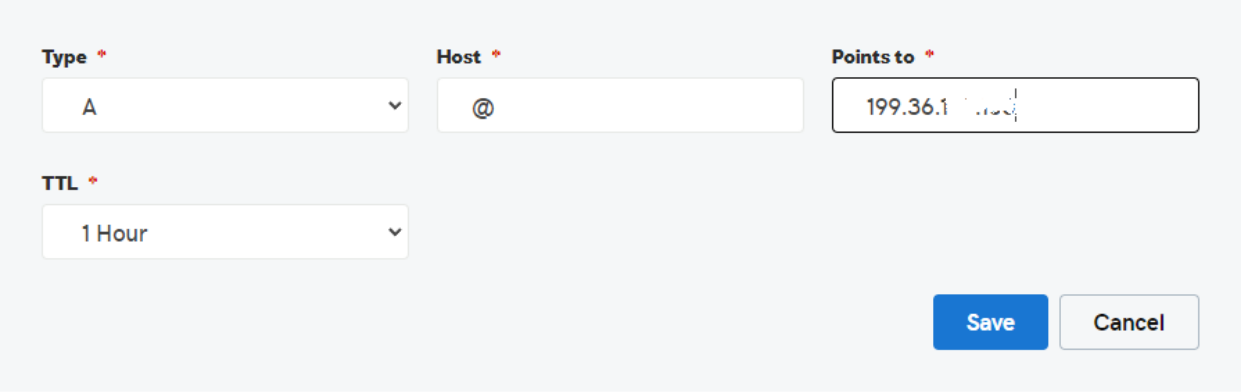


If you get an error message, it means your records have not propagated, or the values may be incorrect. Meanwhile, you can click on **Cancel** to close that window and reopen it later. You'll be prompted to re-enter your domain name when you reopen the window.

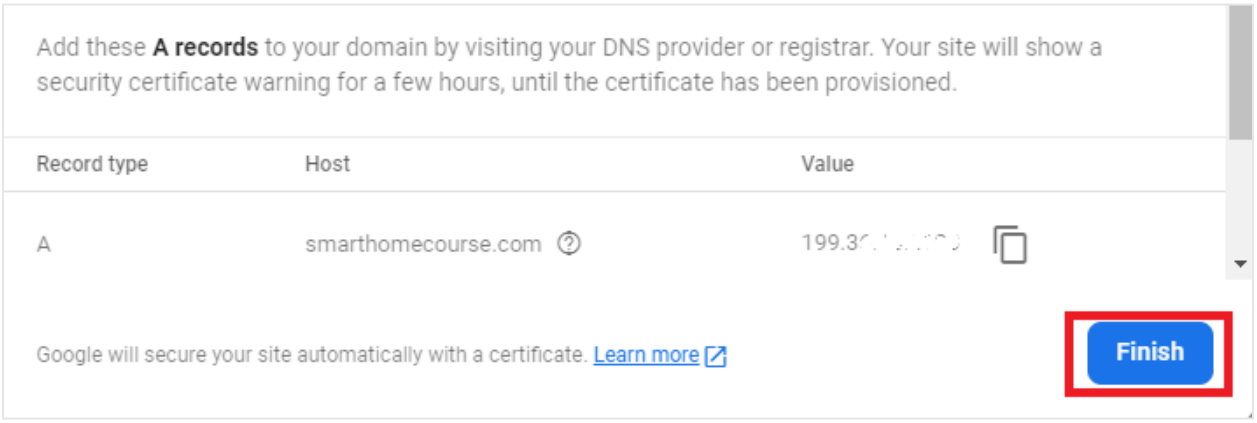
8) When it's ready, the following window will open after clicking the **Verify** button. It shows the **A** records—click on the copy icon to copy it to the clipboard. Google will secure your site automatically with an SSL certificate.



9) Go back to your domain name provider's DNS management section and create a DNS **A** records pointing to your Firebase Hosting. In the "**Points to**" field, insert the A records (IP address) you copied in the previous step. Your domain provider may list this term as "Data", "Points To", "Content", "Address", or "IP Address". Click **Save** to save your DNS A records.

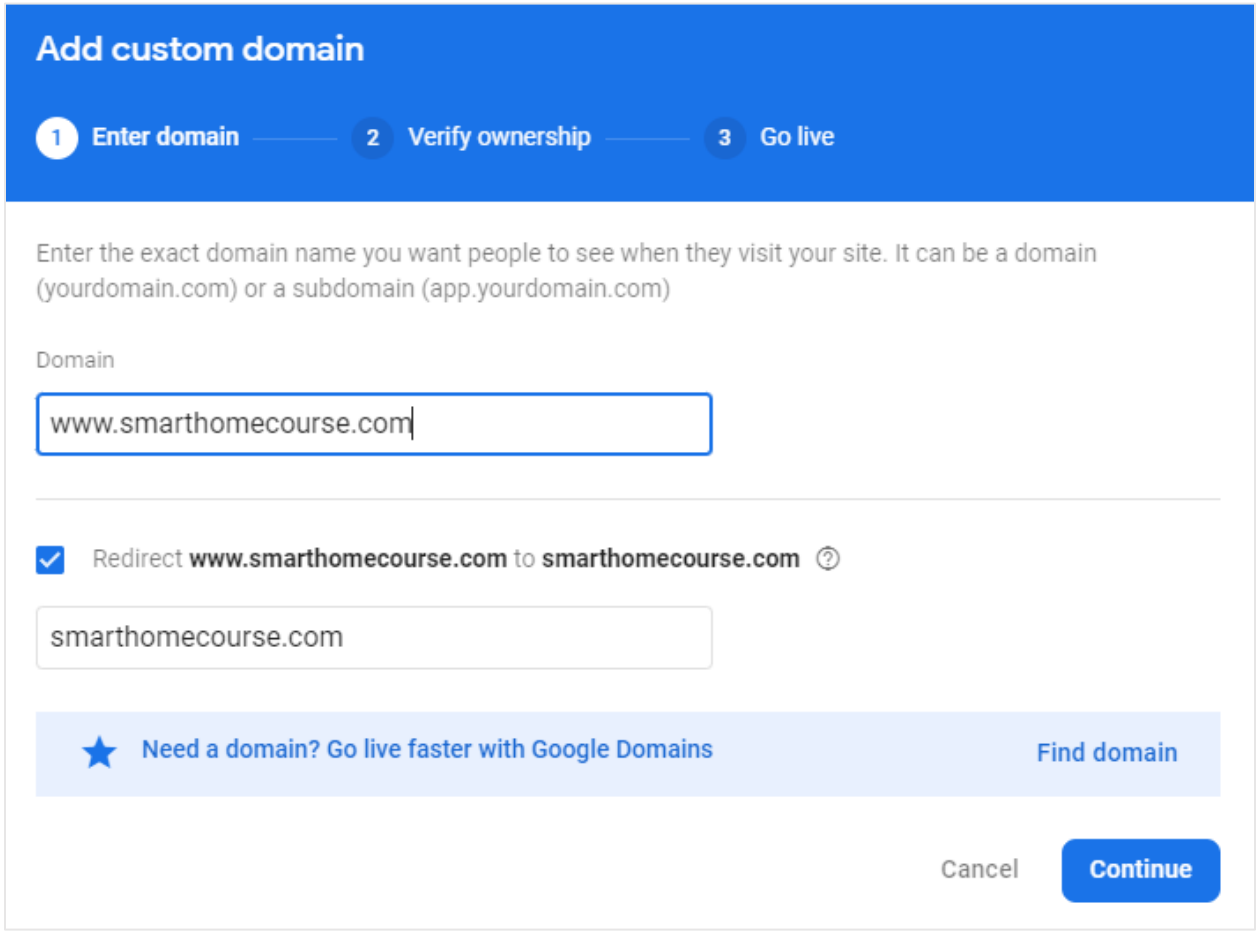


10) Go back to Firebase and click on **Finish**.

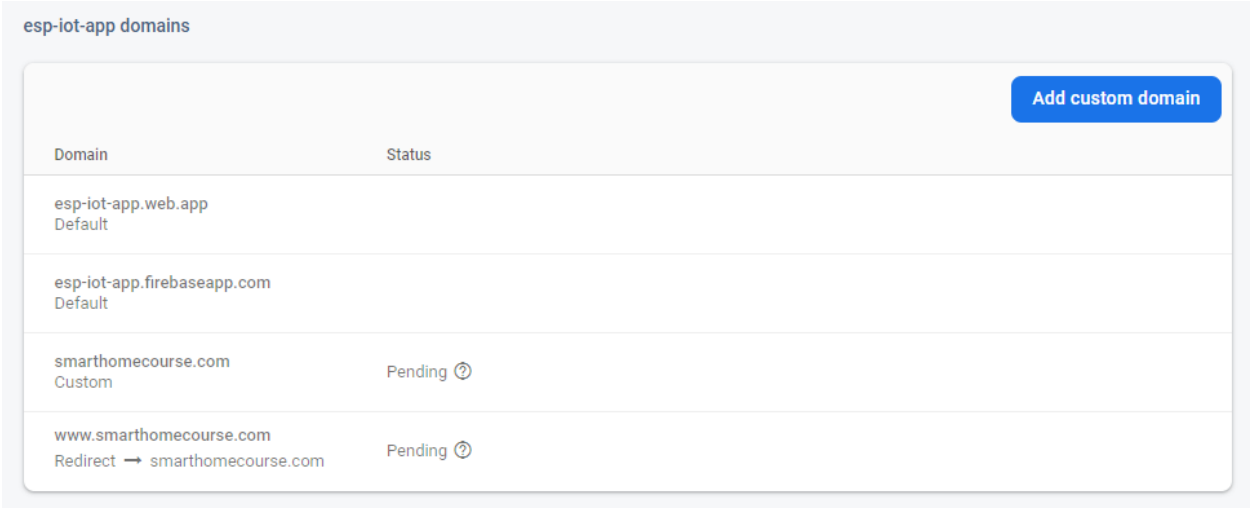


Allow time for your SSL certificate to be provisioned. It may take up to 24 hours. But usually, it just takes a few minutes.

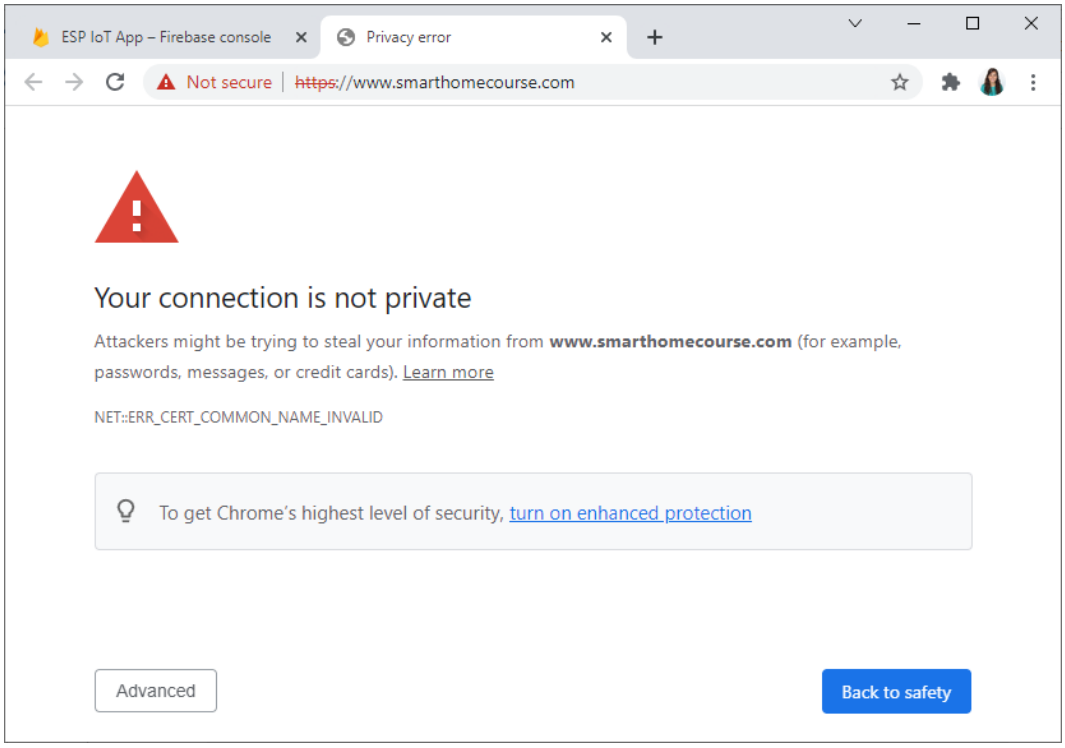
11) Go back to your **Hosting** window on the Firebase console. The new domain should be visible under your app domains. You can add the www redirect if you wish.



12) While you wait for the SSL certificate, you see there's a *pending* status next to your newly added domains.

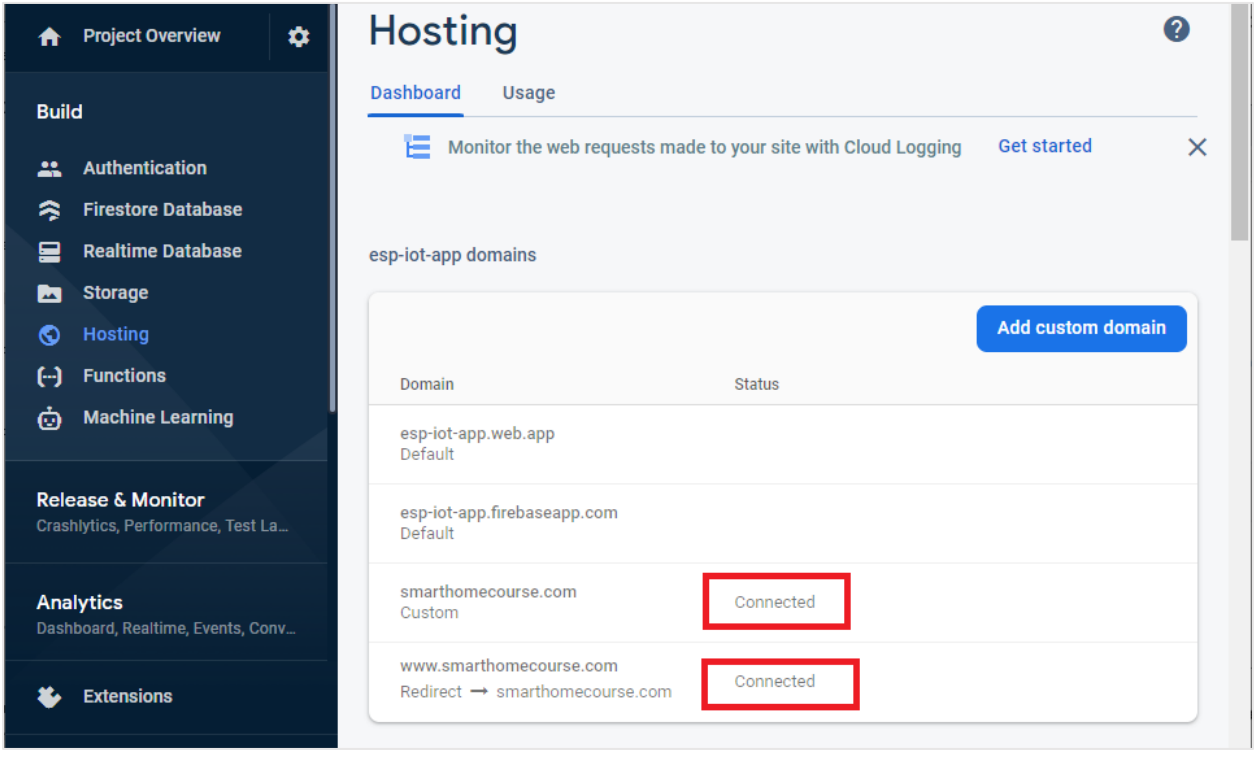


If you try to access your domain at the moment, it will show a privacy error. You can still access your website by clicking on **Advanced** and clicking on the link provided.

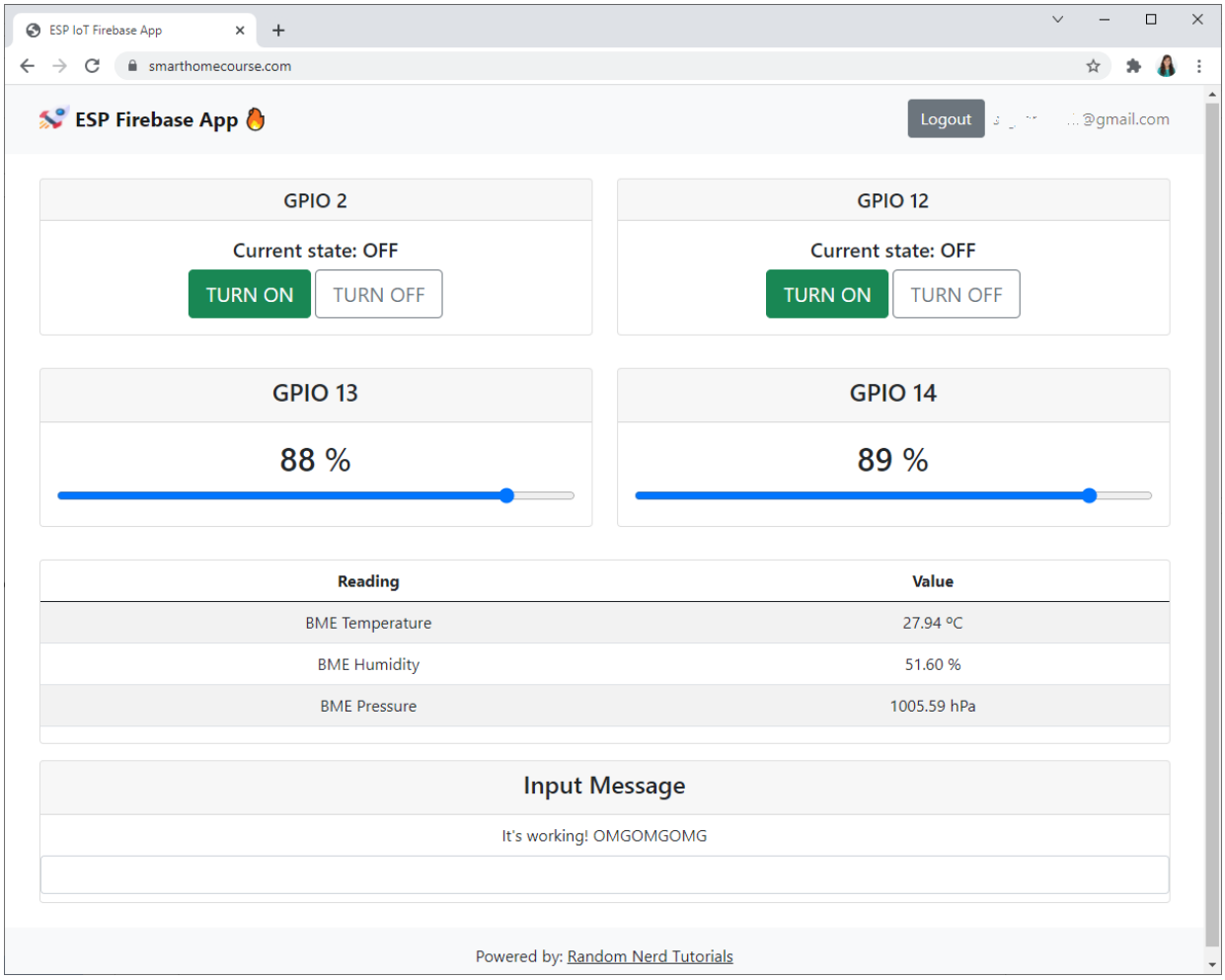


You can also wait a few minutes or hours for the SSL certificate to be provisioned.

Once everything is set up, the status changes to *Connected,* as shown in the following screenshot.



13) Now, you can access the web app using your custom domain name, and your website has an SSL certificate.

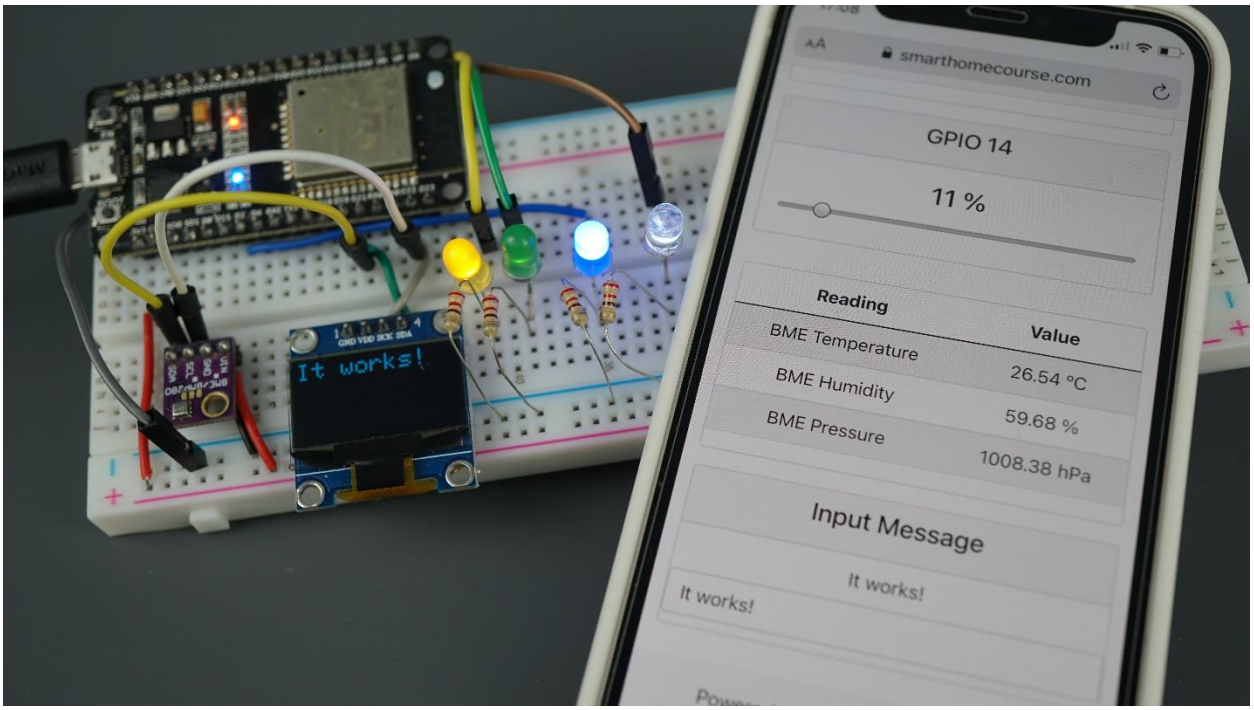


Now, you can access your own web app using your custom domain name to control and monitor your ESP32 and ESP8266 boards from anywhere.

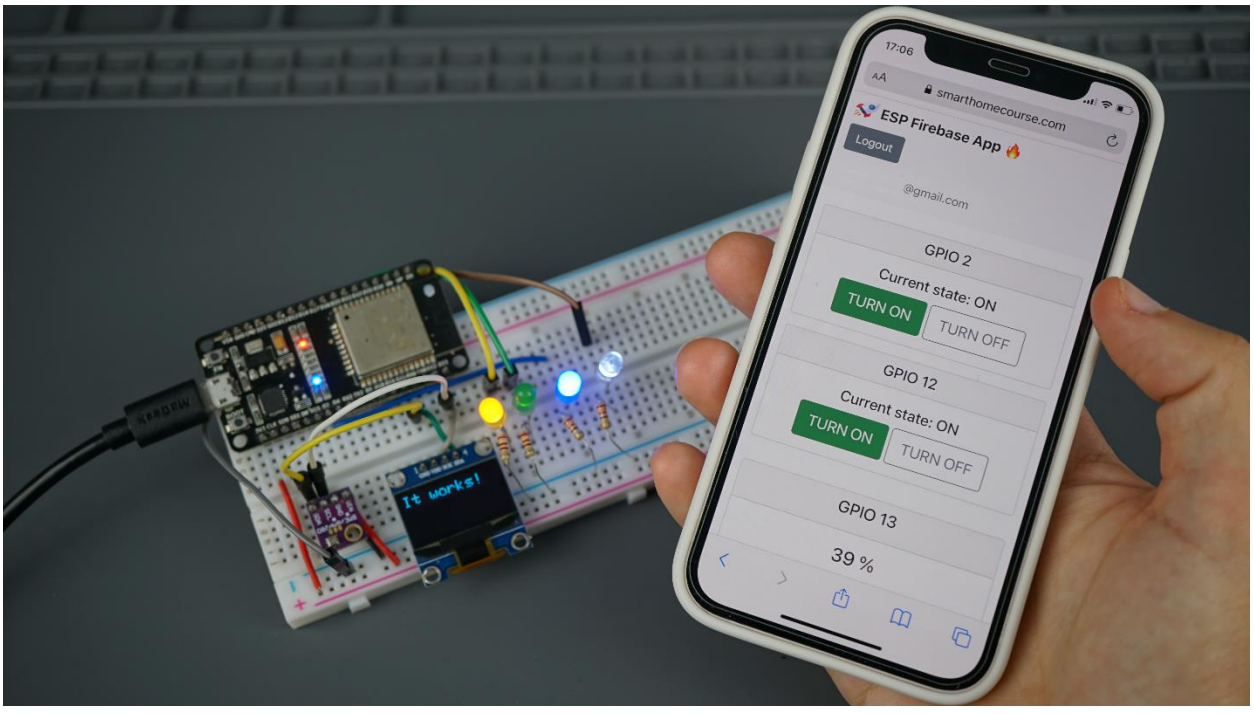
Isn't that exciting?

The following page shows some pictures of the final result.

You can check the current sensor readings on the table.



And you can control your GPIOs using the buttons and sliders, and write a message on the input field to display on the OLED.



Congratulations! You successfully built the Firebase web app project!

CONGRATULATIONS

For Completing This Project

Congratulations for Completing this Project!

If you followed all the steps described in this eBook, you now have a Firebase web application that you can access from anywhere using your own domain name to control and monitor the ESP32 and ESP8266 boards.

Here's a quick summary of what you've learned with this project:

- Set up a Firebase project and add the necessary services;
- Add a Realtime Database to your Firebase project to save data in JSON format;
- Add authentication methods: email and password;
- Interact with the Realtime Database using the ESP32 and ESP8266 boards: writing data to the database and streaming database to detect changes;
- Create a Firebase project app;
- Add login/logout to your app using Firebase SDK for JavaScript;
- Detect database changes (JavaScript);
- Write data to the database (JavaScript);
- Deploy and Host your app;
- Add a custom domain name to your app.
- And much more...

We hope you had fun building this project and that you've learned something new!

If you have something you would like to share (your projects, suggestions, feedback) use the Private Forum, Facebook group, or our Support form.

Good luck with all your projects,

Rui Santos and *Sara Santos*

Other RNT Courses/eBooks

Random Nerd Tutorials is an online resource with electronics projects, tutorials, and reviews. Currently, Random Nerd Tutorials has more than 250 free blog posts with complete tutorials using open-source hardware that anyone can read, remix and apply to their projects.

To keep free tutorials coming, there's also paid content or what we like to call "premium content". To support Random Nerd Tutorials, you can download premium content here. If you enjoyed this eBook, make sure you check all our courses and resources.

Here's a list of all the courses/eBooks available to download at Random Nerd Tutorials:

- Learn ESP32 with Arduino IDE (Bestseller)
- Build Web Servers using ESP32 and ESP8266
- Build ESP32-CAM Projects using Arduino IDE
- Home Automation Using ESP8266
- MicroPython Programming with ESP32/ESP8266
- 20 Easy Raspberry Pi Projects (available on Amazon)
- Build a Home Automation System for $100
- Arduino Step-by-step Projects Course
- Android Apps for Arduino with MIT App Inventor 2
- Electronics For Beginners eBook