ESP32-CAM Save Picture in Firebase Storage

In this guide, you'll learn how to take and upload a picture to Firebase Storage using the ESP32-CAM. You'll create a Firebase project with Storage that allows you to store your files. Then, you can access your Firebase console to visualize the pictures or create a web app to display them (we'll do this in a future tutorial). The ESP32-CAM will be programmed using Arduino IDE.

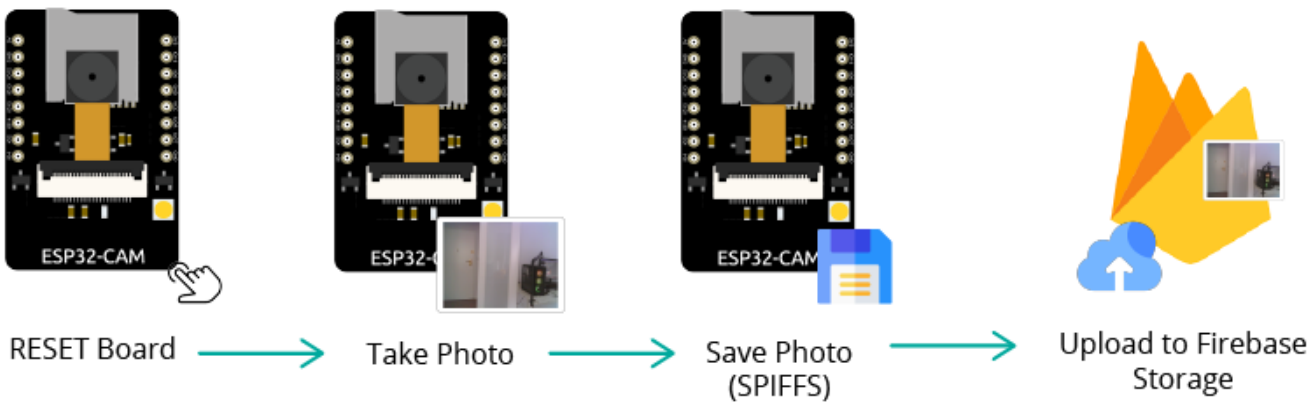
Note: this project is compatible with any ESP32 Camera Board with the OV2640 camera. You just need to make sure you use the right pinout for the board you're using.

What is 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 like authentication, realtime database, hosting, storage, etc.

Project Overview

This simple tutorial exemplifies how to take and send photos taken with the ESP32-CAM to Firebase Storage. The ESP32-CAM takes a picture and sends it to Firebase every time it resets (press the RST button). The idea is that you add some sort of trigger that might be useful for your projects, like a PIR motion sensor or a pushbutton, for example.



- When the ESP32 first runs, it takes a new picture and saves it in the filesystem (SPIFFS);
- The ESP32-CAM connects to Firebase as a user with email and password;
- The ESP32-CAM sends the picture to Firebase Storage;
- After that, you can go to your Firebase console to view the pictures;
- Later, you can build a web app that you can access from anywhere to display the ESP32-CAM pictures (we'll create this in a future tutorial).

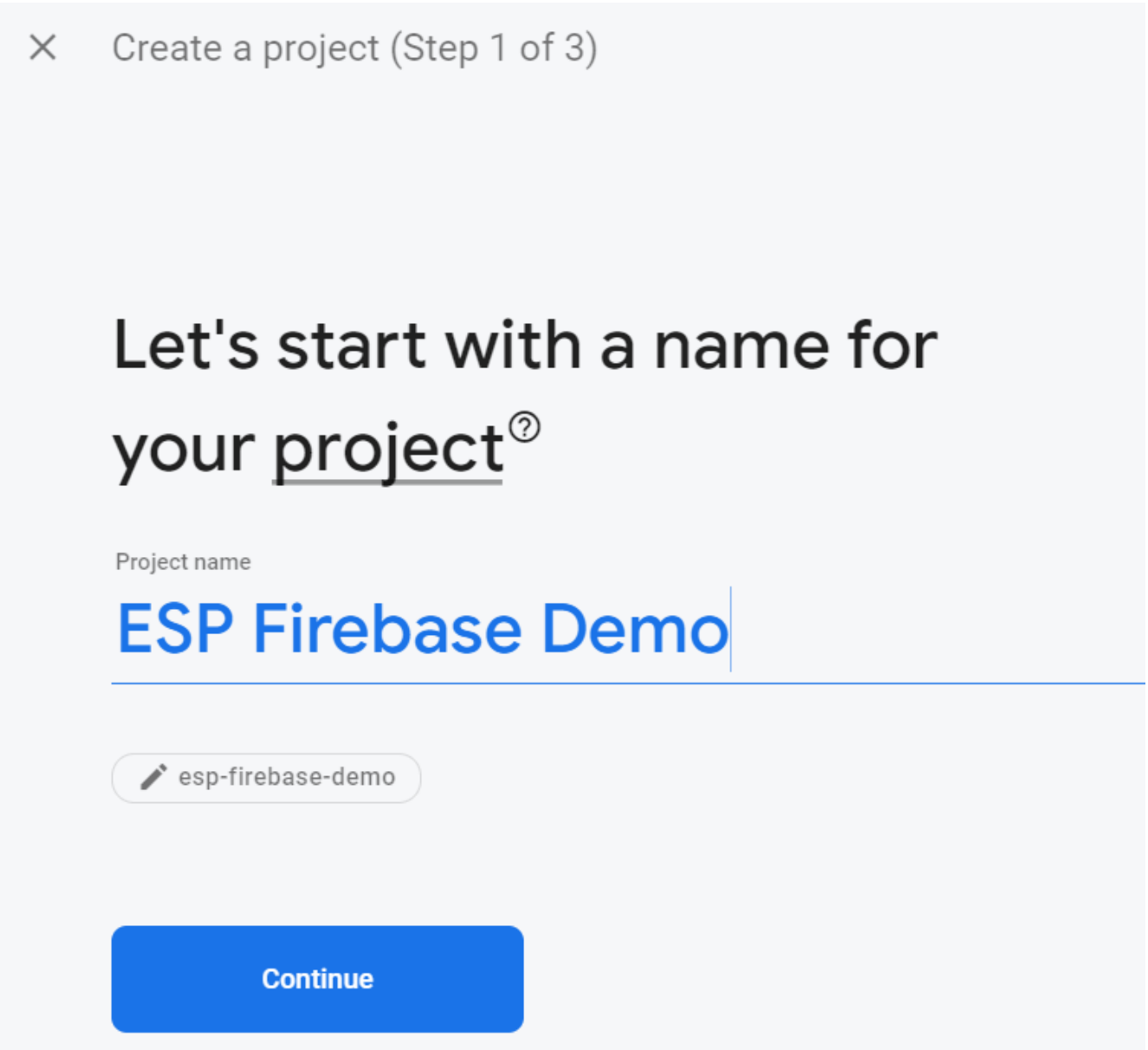
Contents

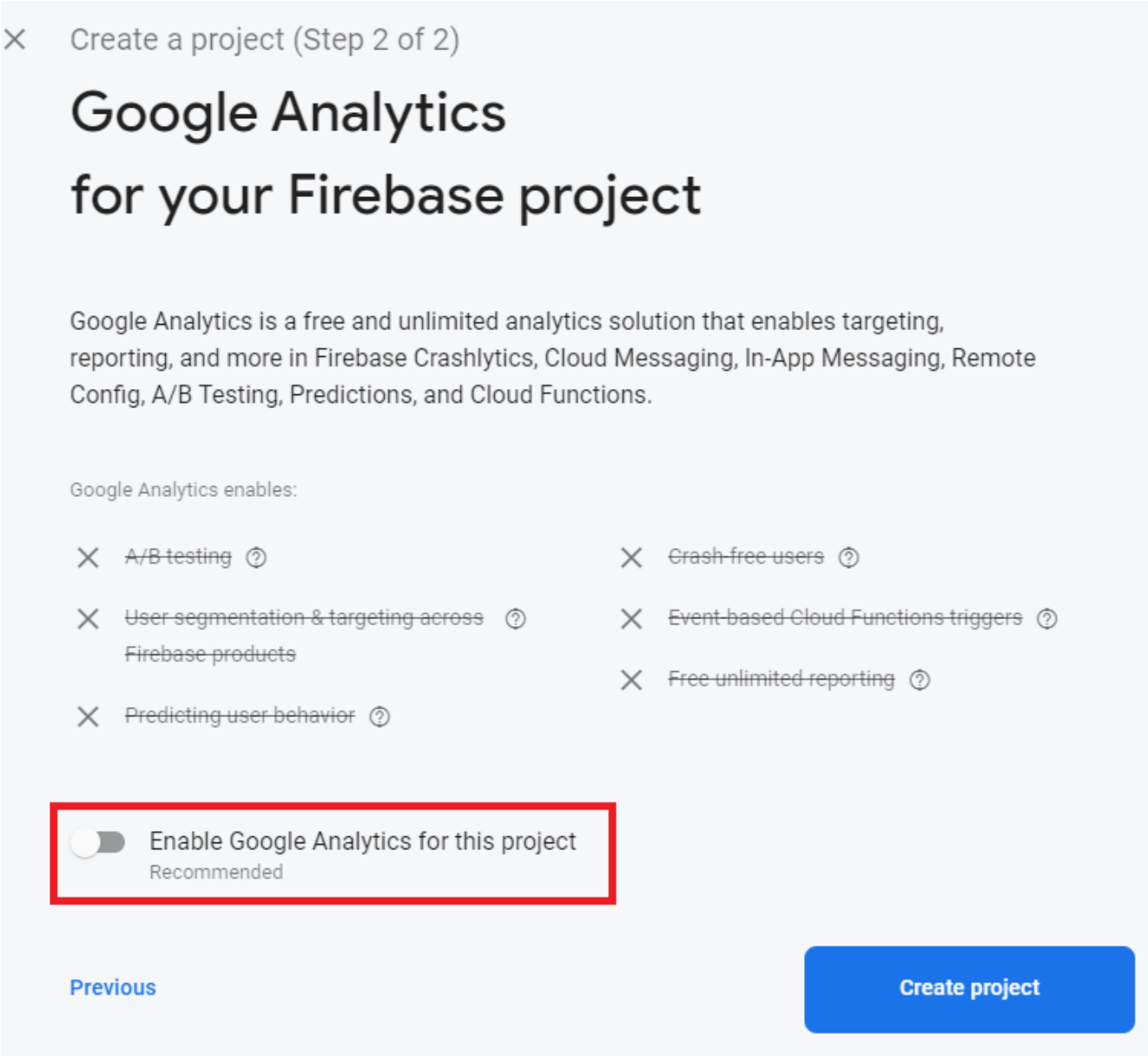
1) Create a Firebase Project

1) Go to Firebase and sign in using a Google Account.

2) Click **Get Started** and then **Add project** to create a new project.

3) Give a name to your project, for example: *ESP Firebase Demo*.





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

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

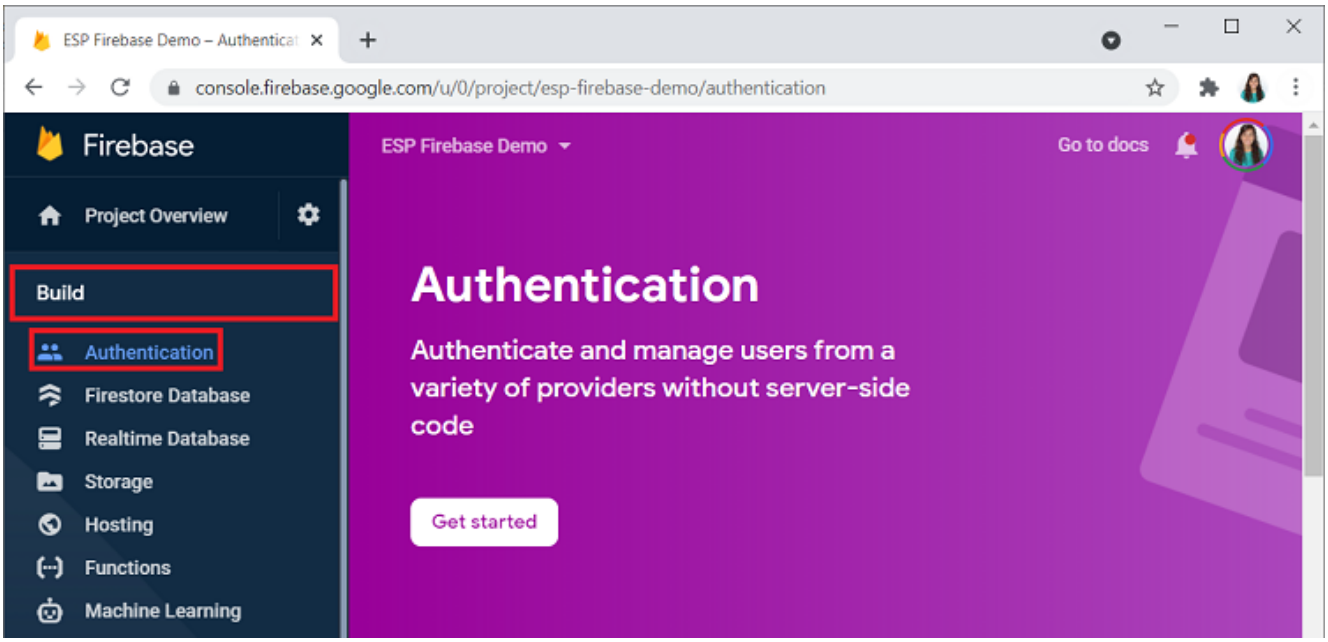
2) Set Authentication Methods

To allow authentication with email and password, first, you need to set authentication methods for your app.

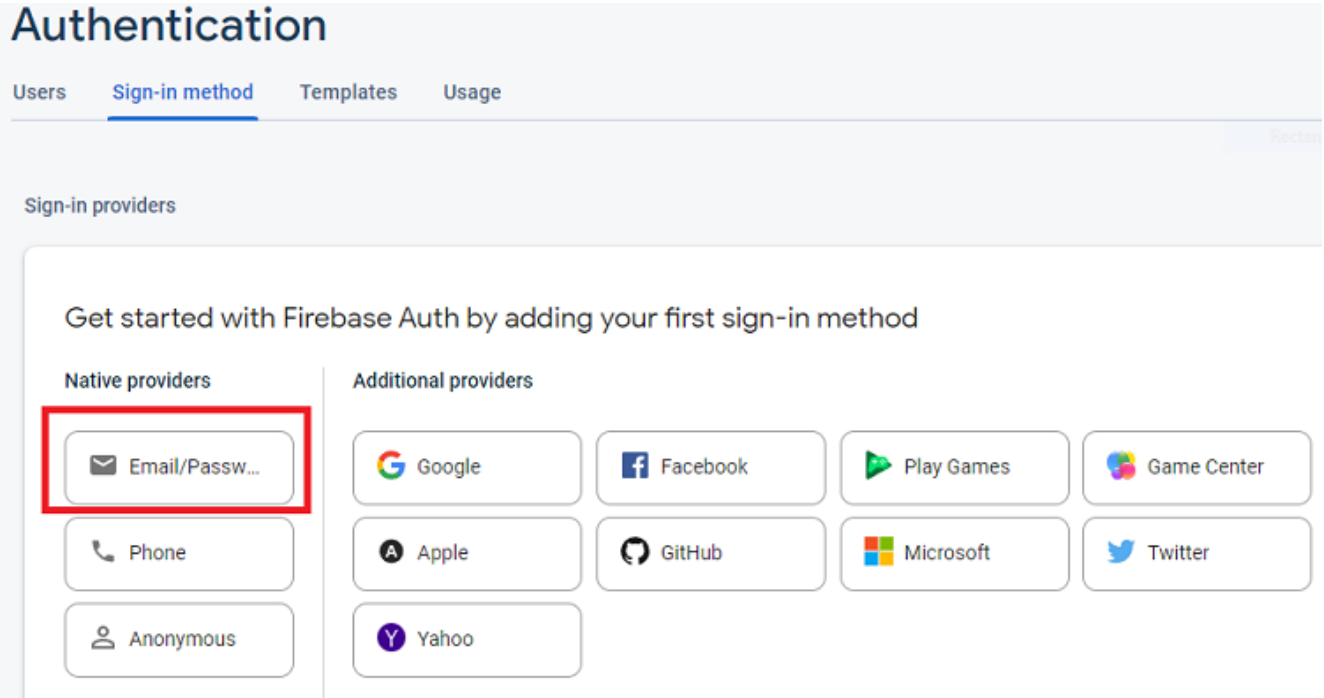
"Most apps need to know the identity of a user. In other words, it takes care of logging in and identifying the users (in this case, the ESP32-CAM). Knowing a

about the authentication methods, you can read the documentation.

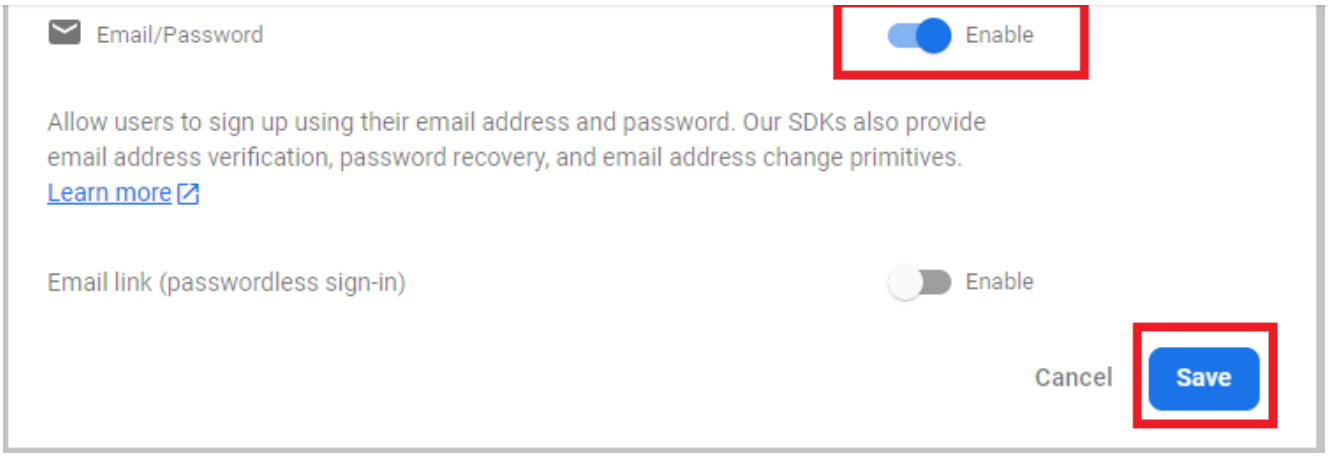
1) On the left sidebar, click on **Authentication** and then on **Get started**.



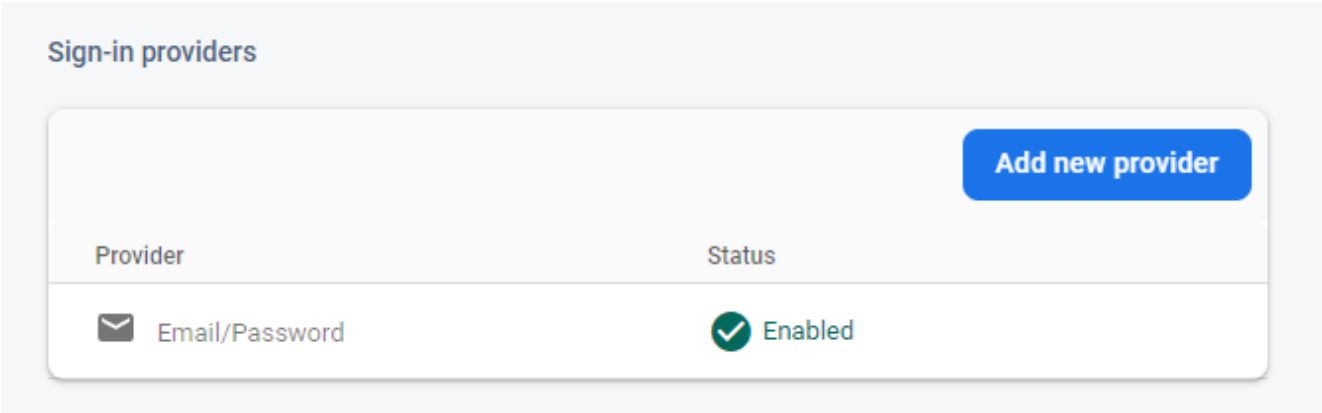
2) Select the Option **Email/Password**.



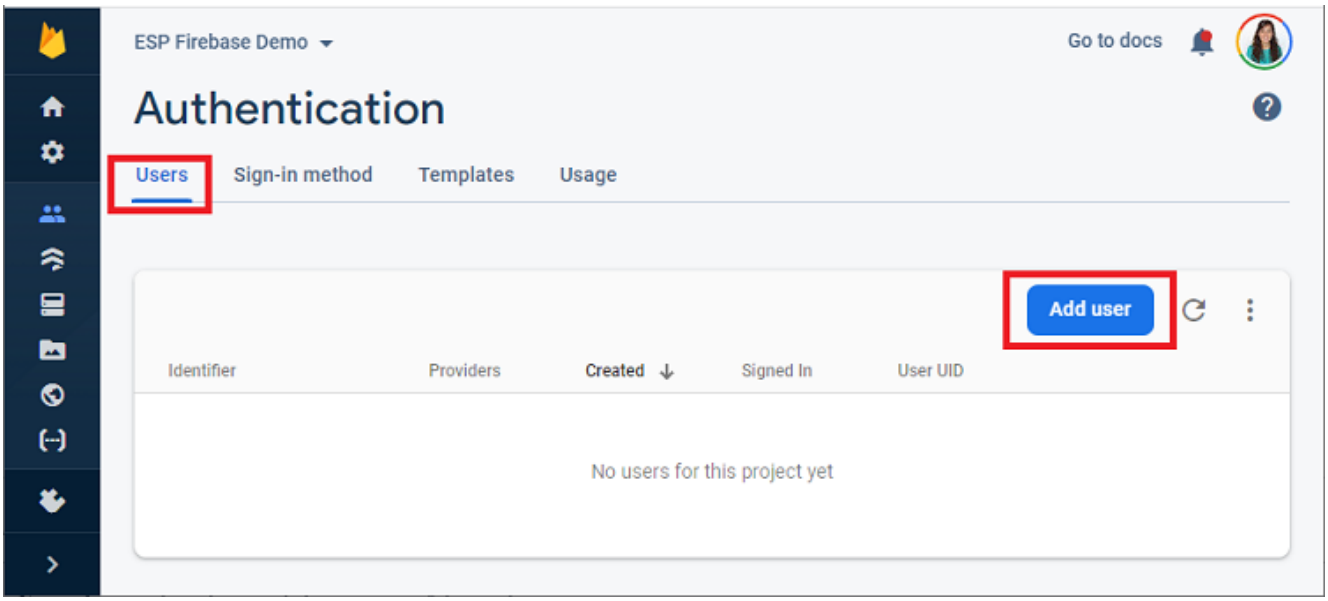
3) Enable that authentication method and click **Save**.



4) The authentication with email and password should now be enabled.

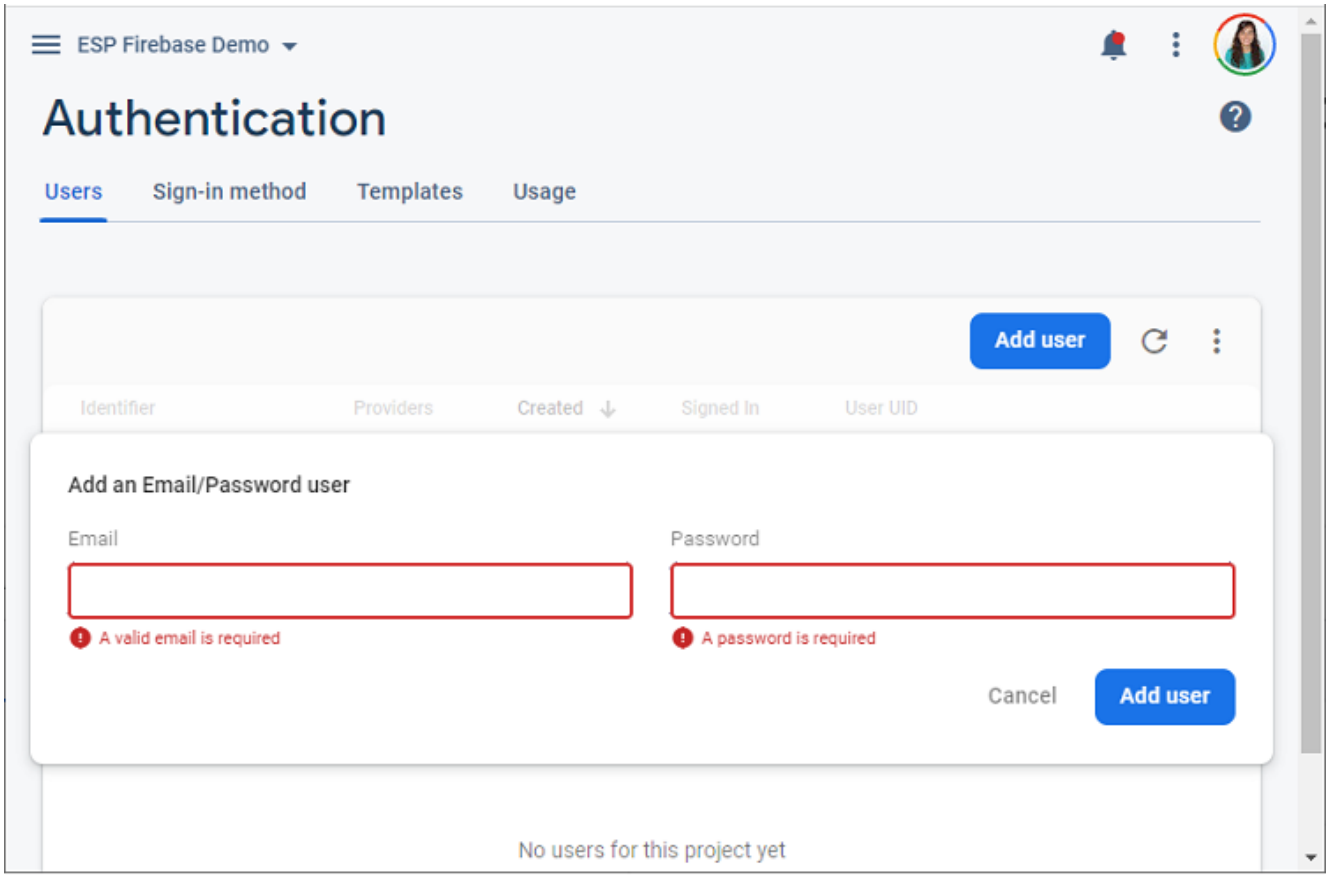


5) Now, you need to add a user. Still on the **Authentication** tab, select the **Users** tab at the top. Then, click on **Add User**.

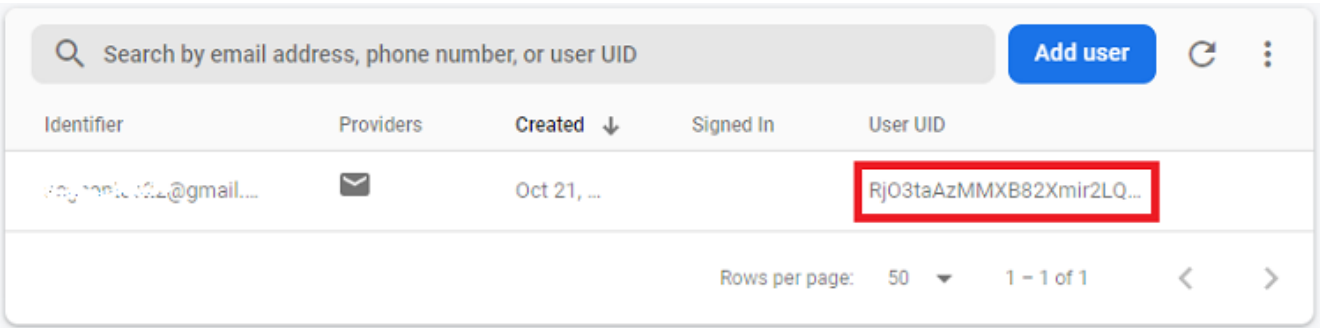


6) Add an email address for the authorized user. It can be your google account email or any other email. You can also create an email for this specific project. Add

When you're done, click **Add user**.

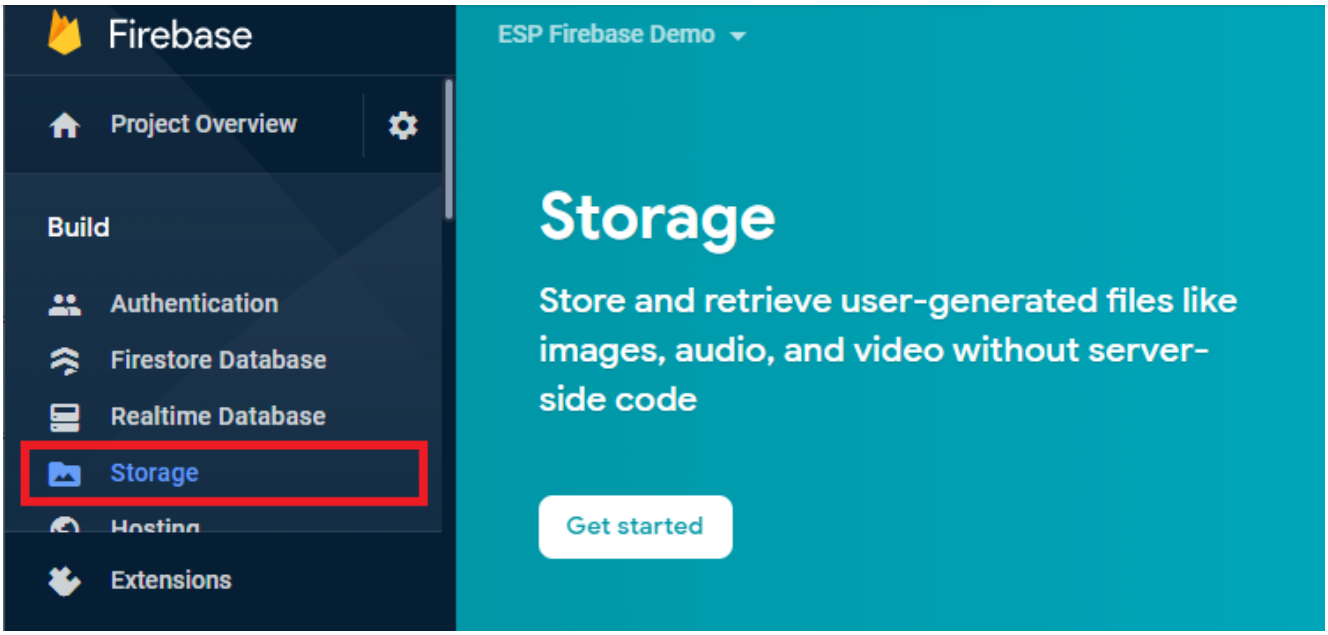


7) A new user was successfully created and added to the **Users** table.

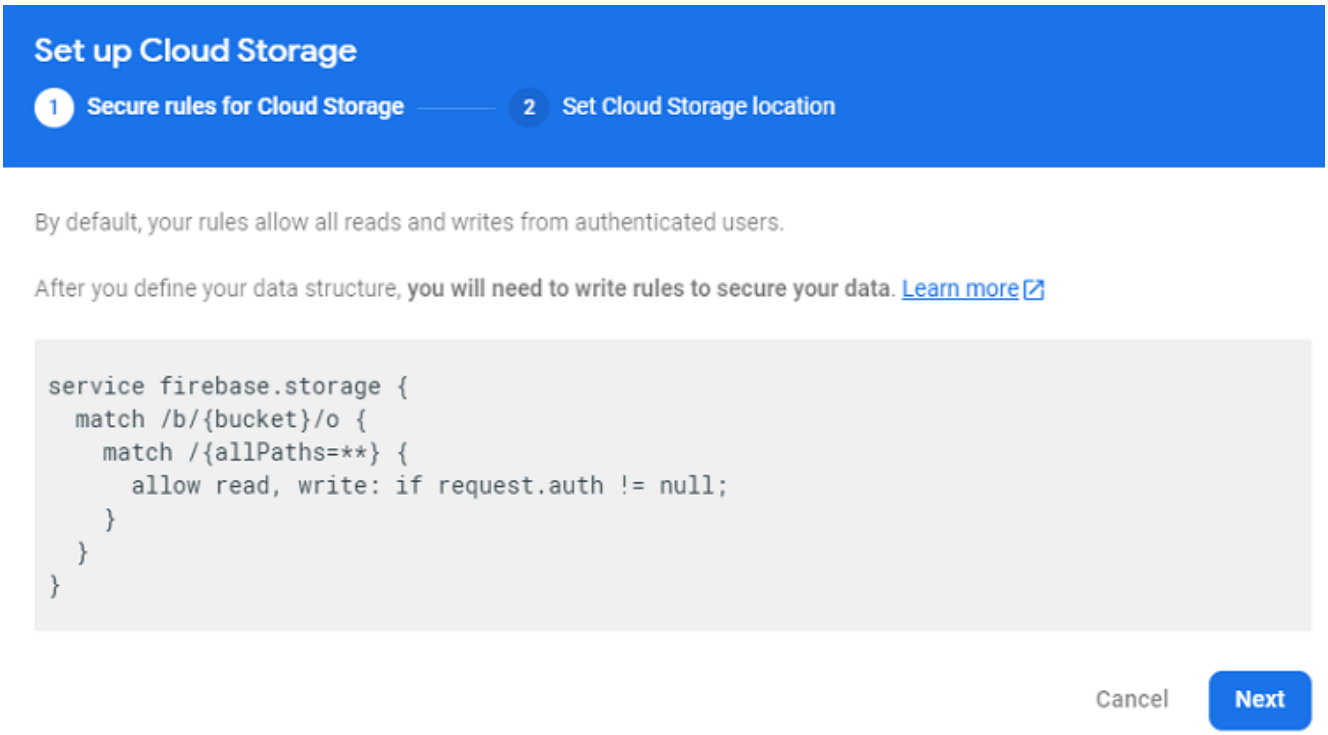


Notice that Firebase creates a unique UID for each registered user. The user UID allows us to identify the user and keep track of the user to provide or deny access to the project or the database. There's also a column that registers the date of the last sign-in. At the moment, it is empty because we haven't signed in with that user yet.

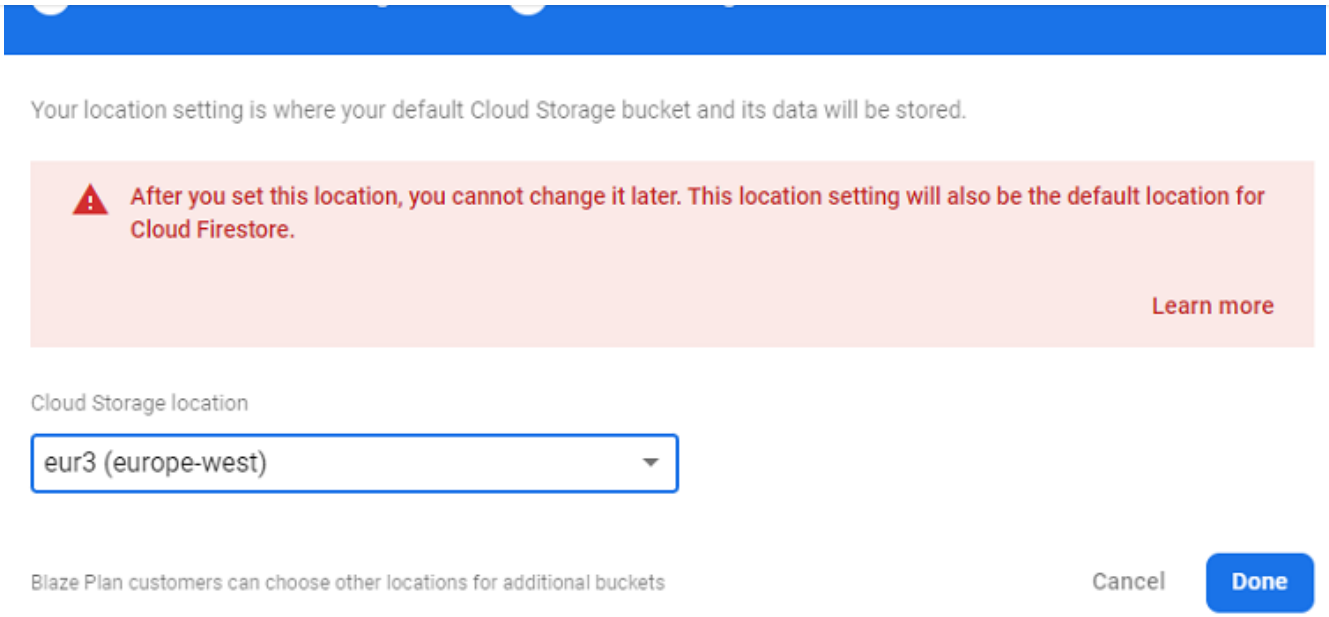
3) Create Storage Bucket



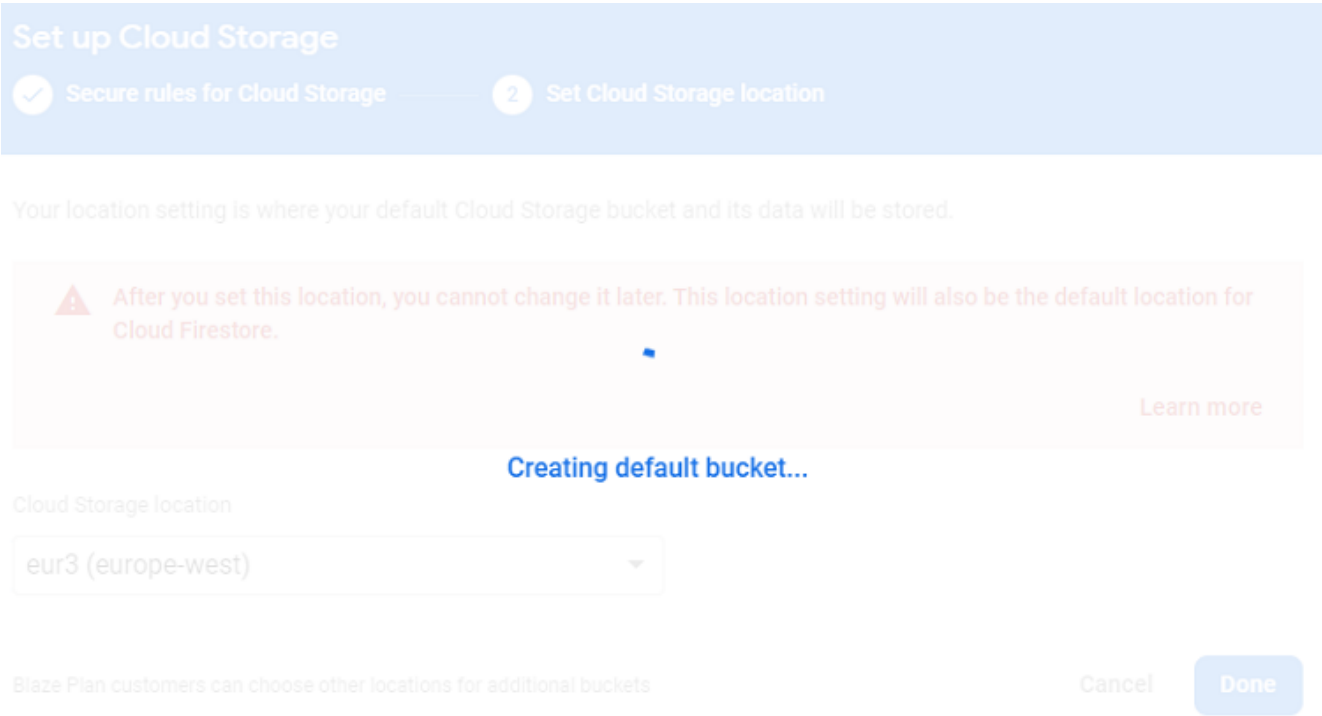
2) Use the default security rules—click **Next**.



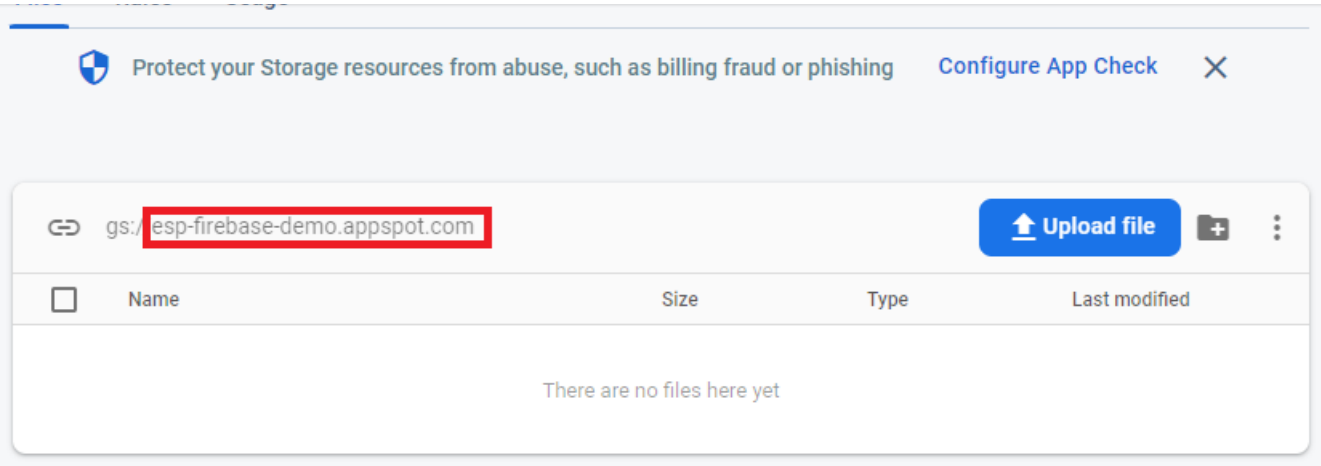
3) Select your storage location—it should be the closest to your country.



4) Wait a few seconds while it creates the storage bucket.



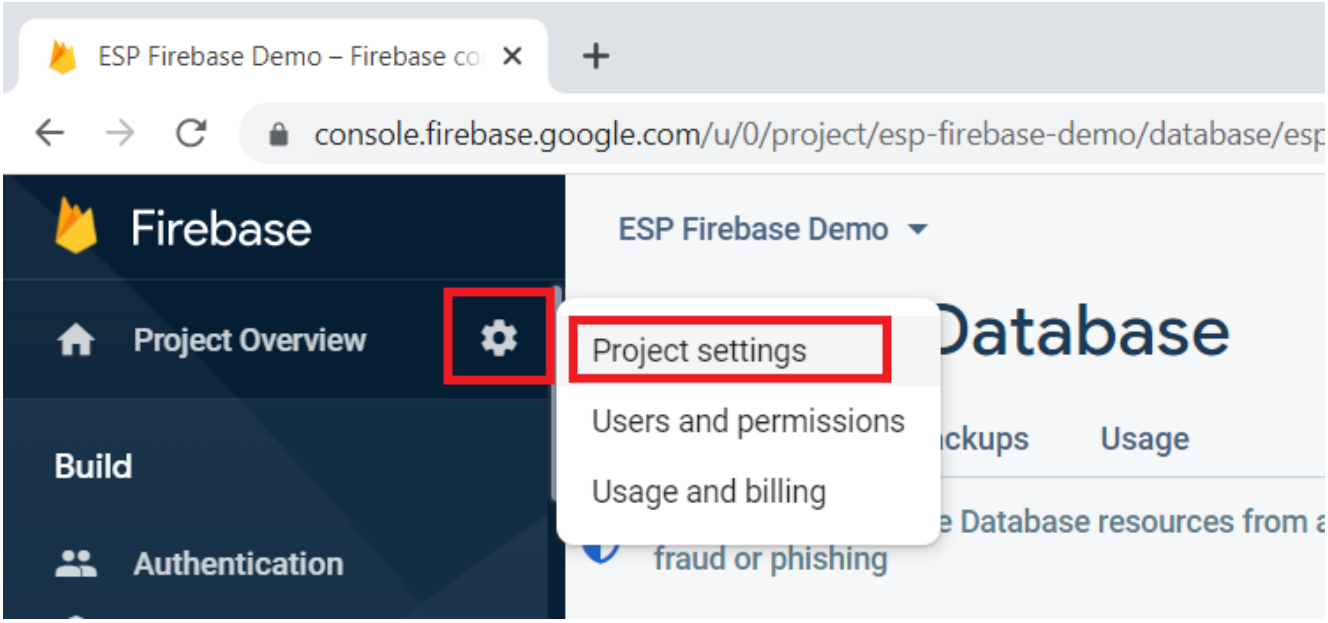
5) The storage bucket is now set up. Copy the storage bucket ID because you'll need it later (copy only the section highlighted with a red rectangle as shown below).



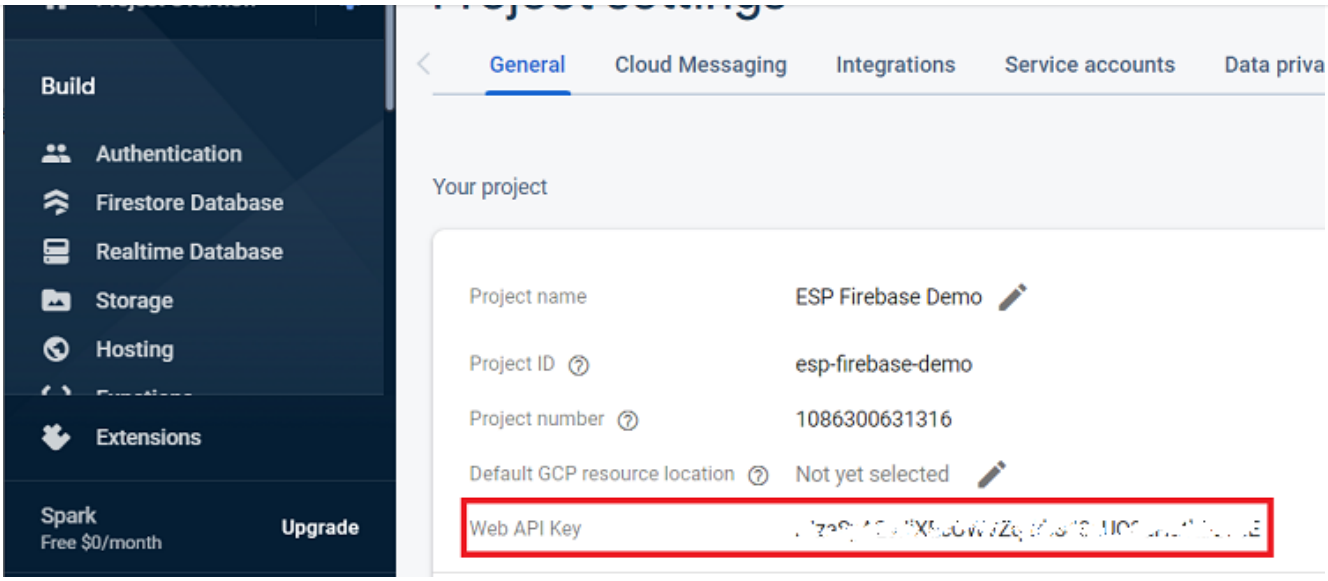
4) Get Project API Key

To interface with your Firebase project using the ESP32-CAM, 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.



---

5) ESP32-CAM – Send Pictures to Firebase Storage

Before proceeding with the tutorial, make sure you check the following prerequisites.

Installing the ESP32 add-on

We'll program the ESP32-CAM board using Arduino IDE. So you need the Arduino IDE installed as well as the ESP32 add-on. Follow the next tutorial to install it, if you haven't already.

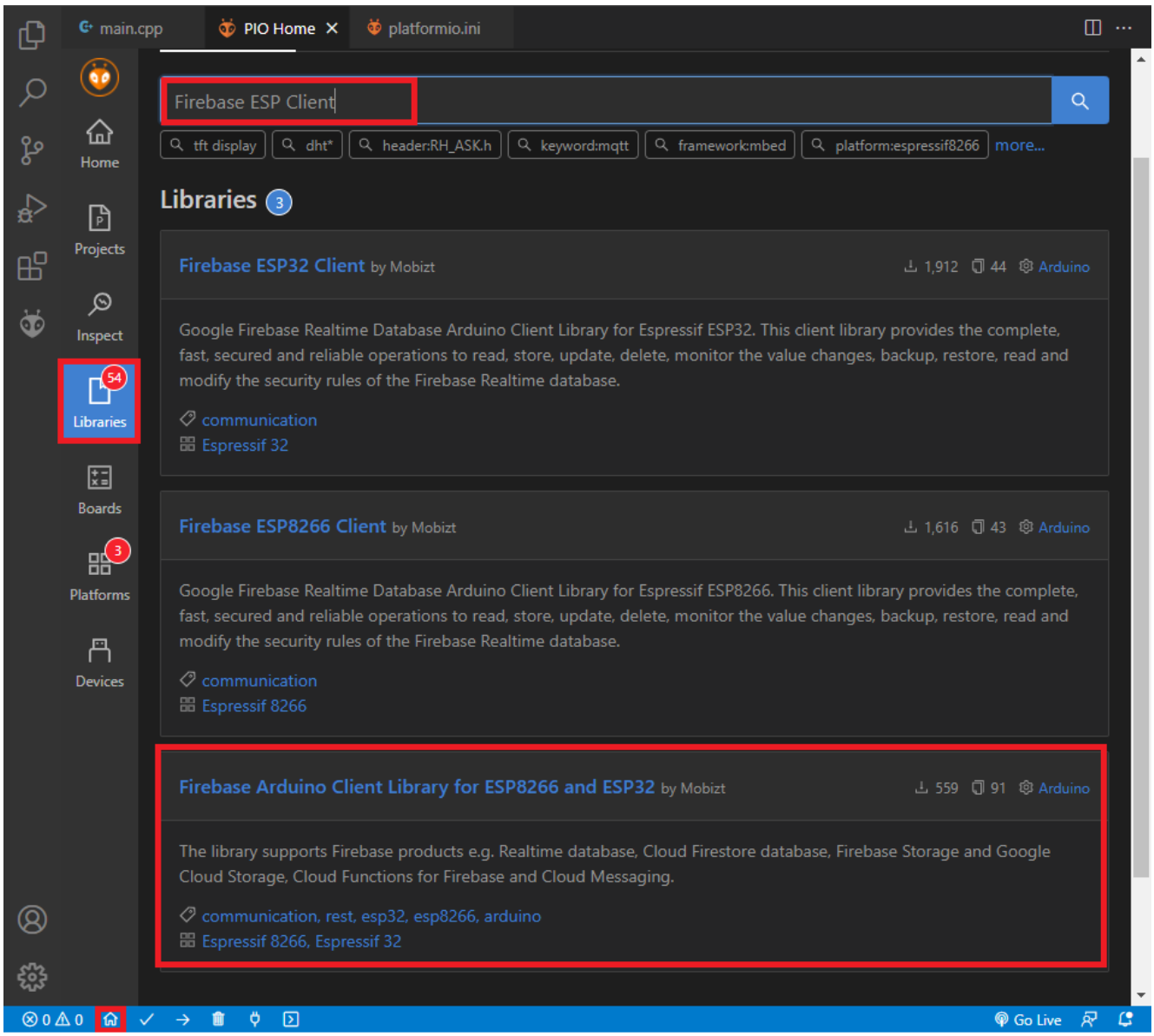
- Installing the ESP32 Board in Arduino IDE (Mac OS X and Linux instructions)

Installing ESP Firebase Client Library

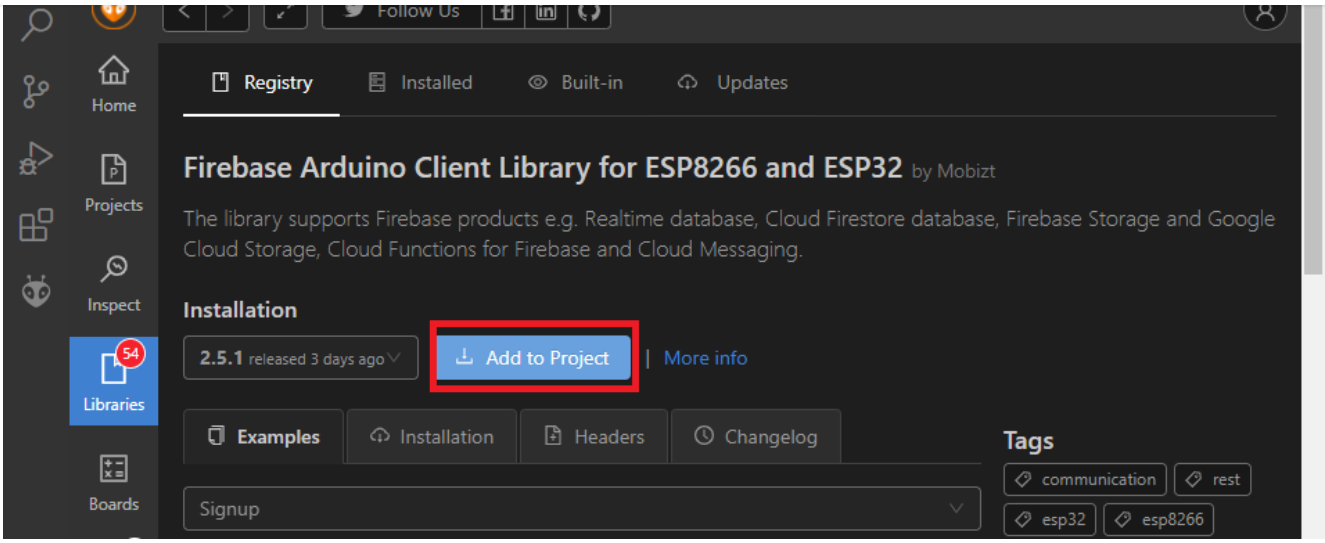
The Firebase-ESP-Client library provides several examples to interface with Firebase services. It provides an example that shows how to send files to Firebase Storage. Our code we'll be based on that example. So, you need to make sure you have that library installed.

Installation – VS Code + PlatformIO

the **Firebase Arduino Client Library for ESP8266 and ESP32**.



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



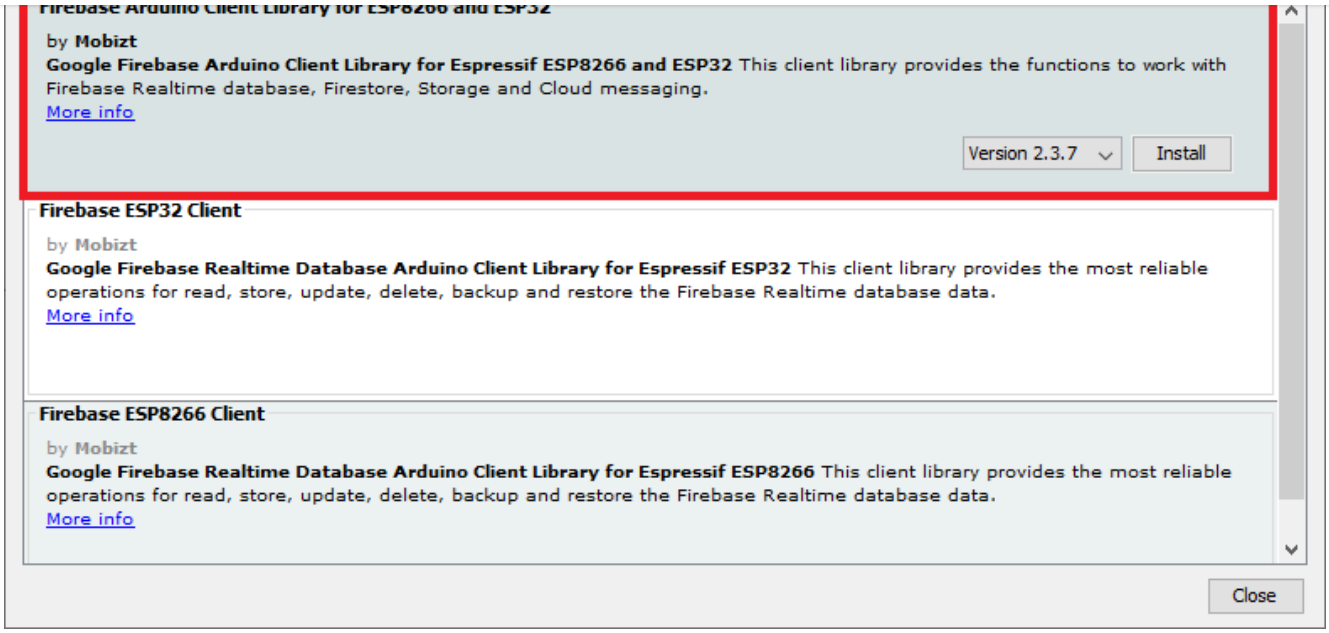
Also, change the monitor speed to 115200 by adding the following line to the `platformio.ini` file of your project:

```
monitor_speed = 115200
```

Installation – Arduino IDE

If you're using Arduino IDE, follow the next steps to install the library.

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-CAM board to send pictures to Firebase Storage.

ESP32-CAM Send Pictures to Firebase – Code

Copy the following code to your Arduino IDE, or to the `main.cpp` file if you're using VS Code. It takes a picture and sends it to Firebase when it first boots.

```
/*********
  Rui Santos
  Complete instructions at: https://RandomNerdTutorials.c

  Permission is hereby granted, free of charge, to any pe
  The above copyright notice and this permission notice s

  Based on the example provided by the ESP Firebase Clier
*********/

#include "WiFi.h"
#include "esp_camera.h"
#include "Arduino.h"
#include "soc/soc.h"           // Disable brownout proble
#include "soc/rtc_cntl_reg.h"  // Disable brownout proble
```

```
#include <FS.h>
#include <Firebase_ESP_Client.h>
//Provide the token generation process info.
#include <addons/TokenHelper.h>

//Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

View raw code

You need to insert your network credentials, storage bucket ID, and project API key for the project to work.

This sketch was based on a basic example provided by the library. You can find more examples here.

How the Code Works

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

Libraries

First, include the required libraries.

```
#include "WiFi.h"
#include "esp_camera.h"
#include "Arduino.h"
#include "soc/soc.h"           // Disable brownout problems
#include "soc/rtc_cntl_reg.h"  // Disable brownout problems
#include "driver/rtc_io.h"
#include <SPIFFS.h>
#include <FS.h>
#include <Firebase_ESP_Client.h>
```

Network Credentials

Insert your network credentials in the following variables so that the ESP can connect to the internet and communicate with Firebase.

```
//Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Firebase Project API Key

Insert your Firebase project API key—see this section: 4) Get Project API Key.

```
// Insert Firebase project API Key
#define API_KEY "REPLACE_WITH_YOUR_FIREBASE_PROJECT_API_KEY.
```

User Email and Password

Insert the authorized email and the corresponding password—see this section: 2) Set Authentication Methods.

```
#define USER_EMAIL "REPLACE_WITH_THE_AUTHORIZED_USER_EMAIL"
#define USER_PASSWORD "REPLACE_WITH_THE_AUTHORIZED_USER_PASS
```

Firebase Storage Bucket ID

Insert the Firebase storage bucket ID, e.g *bucket-name.appspot.com*. In my case, it is `esp-firebase-demo.appspot.com`.

```
define STORAGE_BUCKET_ID "REPLACE_WITH_YOUR_STORAGE_BUCKET_I
```

Picture Path

The `FILE_PHOTO` variable defines the SPIFFS path where the picture will be saved. It will be saved with the name `photo.jpg` under the `data` folder.

```
#define FILE_PHOTO "/data/photo.jpg"
```

ESP32-CAM Pin Definition

The following lines define the ESP32-CAM pins. This is the definition for the ESP32-CAM AI-Thinker module. If you're using another ESP32-CAM module, you need to modify the pin definition—check this tutorial: ESP32-CAM Camera Boards: Pin and GPIOs Assignment Guide.

```
// OV2640 camera module pins (CAMERA_MODEL_AI_THINKER)
#define PWDN_GPIO_NUM     32
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM      0
#define SIOD_GPIO_NUM     26
#define SIOC_GPIO_NUM     27
#define Y9_GPIO_NUM       35
#define Y8_GPIO_NUM       34
#define Y7_GPIO_NUM       39
#define Y6_GPIO_NUM       36
#define Y5_GPIO_NUM       21
#define Y4_GPIO_NUM       19
#define Y3_GPIO_NUM       18
#define Y2_GPIO_NUM        5
#define VSYNC_GPIO_NUM    25
#define HREF_GPIO_NUM     23
#define PCLK_GPIO_NUM     22
```

Other Variables

```
boolean takeNewPhoto = true;
```

Then, we define Firebase configuration data objects.

```
//Define Firebase Data objects
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig configF;
```

The `taskCompleted` is a boolean variable that checks if we successfully connected to Firebase.

```
bool taskCompleted = false;
```

checkPhoto() Function

The `checkPhoto()` function checks if the picture was successfully taken and saved in SPIFFS.

```
// Check if photo capture was successful
bool checkPhoto( fs::FS &fs ) {
  File f_pic = fs.open( FILE_PHOTO );
  unsigned int pic_sz = f_pic.size();
  return ( pic_sz > 100 );
}
```

capturePhotoSaveSpiffs() Function

The `capturePhotoSaveSpiffs()` function takes a photo and saves it in the ESP32 filesystem.

```
  camera_fb_t * fb = NULL; // pointer
  bool ok = 0; // Boolean indicating if the picture has beer
  do {
    // Take a photo with the camera
    Serial.println("Taking a photo...");

    fb = esp_camera_fb_get();
    if (!fb) {
      Serial.println("Camera capture failed");
      return;
    }
    // Photo file name
    Serial.printf("Picture file name: %s\n", FILE_PHOTO);
    File file = SPIFFS.open(FILE_PHOTO, FILE_WRITE);
    // Insert the data in the photo file
    if (!file) {
      Serial.println("Failed to open file in writing mode");
    }
    else {
      file.write(fb->buf, fb->len); // payload (image), payl
      Serial.print("The picture has been saved in ");
      Serial.print(FILE_PHOTO);
      Serial.print(" - Size: ");
      Serial.print(file.size());
      Serial.println(" bytes");
    }
    // Close the file
    file.close();
    esp_camera_fb_return(fb);

    // check if file has been correctly saved in SPIFFS
    ok = checkPhoto(SPIFFS);
  } while ( !ok );
}
```

The `initWiFi()` function initializes Wi-Fi.

```
void initWiFi(){
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
}
```

initSPIFFS() Function

The `initSPIFFS()` function initializes the SPIFFS filesystem.

```
void initSPIFFS(){
  if (!SPIFFS.begin(true)) {
    Serial.println("An Error has occurred while mounting SPI
    ESP.restart();
  }
  else {
    delay(500);
    Serial.println("SPIFFS mounted successfully");
  }
}
```

initCamera() Function

The `initCamera()` function initializes the ESP32-CAM.

```
void initCamera(){
 // OV2640 camera module
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
```

```
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;

  if (psramFound()) {
    config.frame_size = FRAMESIZE_UXGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
  } else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
  }
  // Camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err)
    ESP.restart();
  }
}
```

In the `setup()`, initialize the Serial Monitor, Wi-Fi, SPIFFS, and the camera.

```
// Serial port for debugging purposes
Serial.begin(115200);
initWiFi();
initSPIFFS();
// Turn-off the 'brownout detector'
WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);
initCamera();
```

Then, assign the following settings to the Firebase configuration objects.

```
// Assign the api key
configF.api_key = API_KEY;
//Assign the user sign in credentials
auth.user.email = USER_EMAIL;
auth.user.password = USER_PASSWORD;
//Assign the callback function for the long running token ge
configF.token_status_callback = tokenStatusCallback; //see c
```

Finally, initialize Firebase.

```
Firebase.begin(&configF, &auth);
Firebase.reconnectWiFi(true);
```

loop()

In the `loop()`, take a new picture and save it to the filesystem.

```
if (takeNewPhoto) {
  capturePhotoSaveSpiffs();
```

Finally, send the picture to Firebase.

```
if (Firebase.ready() && !taskCompleted){
  taskCompleted = true;
  Serial.print("Uploading picture... ");

  //MIME type should be valid to avoid the download problem.
  //The file systems for flash and SD/SDMMC can be changed i
   if (Firebase.Storage.upload(&fbdo, STORAGE_BUCKET_ID /* F
    Serial.printf("\nDownload URL: %s\n", fbdo.downloadURL()
  }
  else{
    Serial.println(fbdo.errorReason());
  }
}
```

The command that actually sends the picture is `Firebase.Storage.upload()`:

```
Firebase.Storage.upload(&fbdo, STORAGE_BUCKET_ID, FILE_PHOTO
```

This function returns a boolean variable indicating the success of the operation.

It accepts as the second argument, the storage bucket ID. Then, the path where the file is saved; the storage type (it can be SPIFFS or SD Card*); the path where the file will be saved in the Firebase storage; and finally, the mime type.

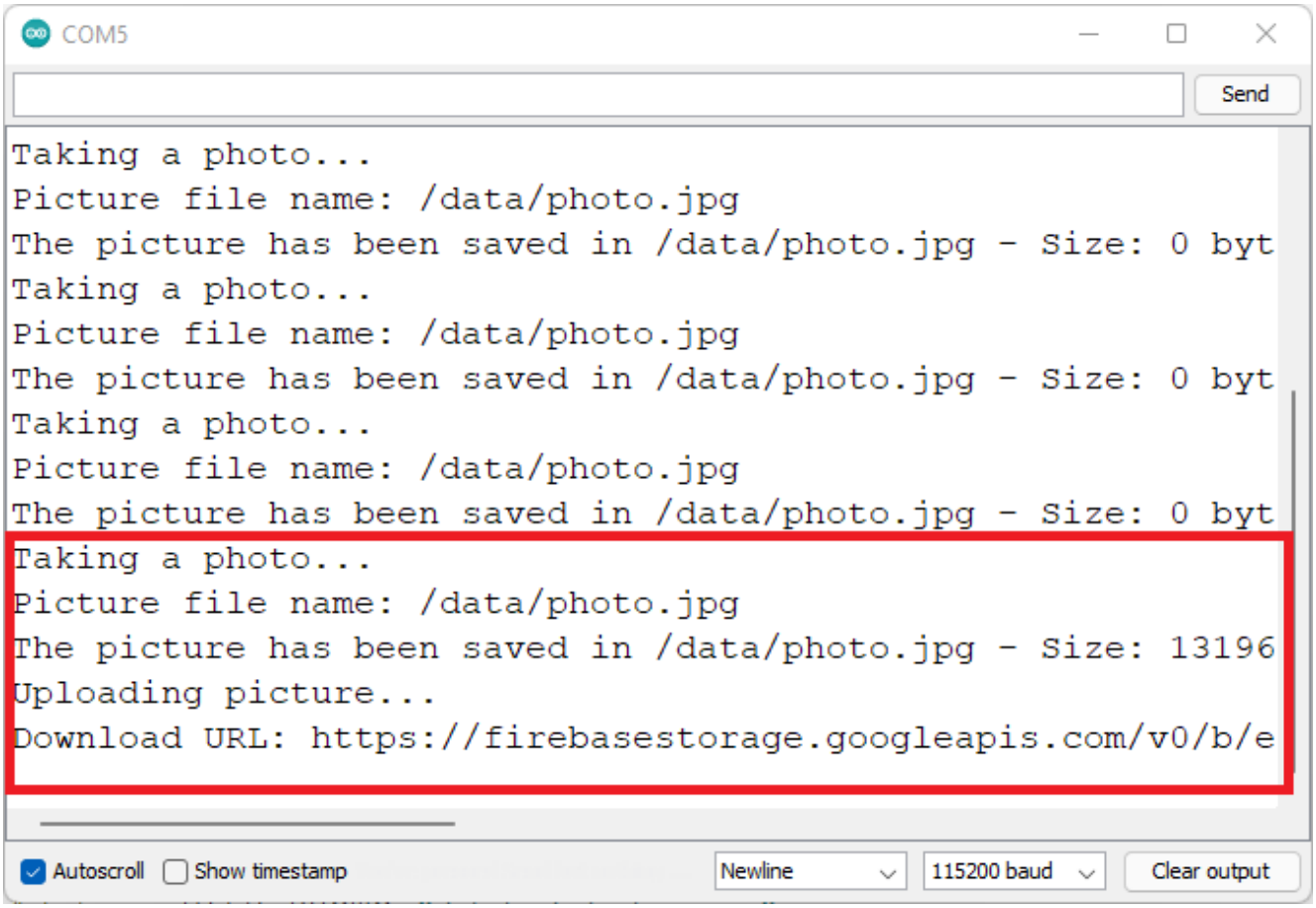
*we were not able to make this example work with the ESP32-CAM + microSD card. If anyone knows a solution, please share.

After inserting the required credentials, upload the code to your ESP32-CAM. If you don't know how to upload code to the ESP32-CAM, you can follow the next tutorial(s):

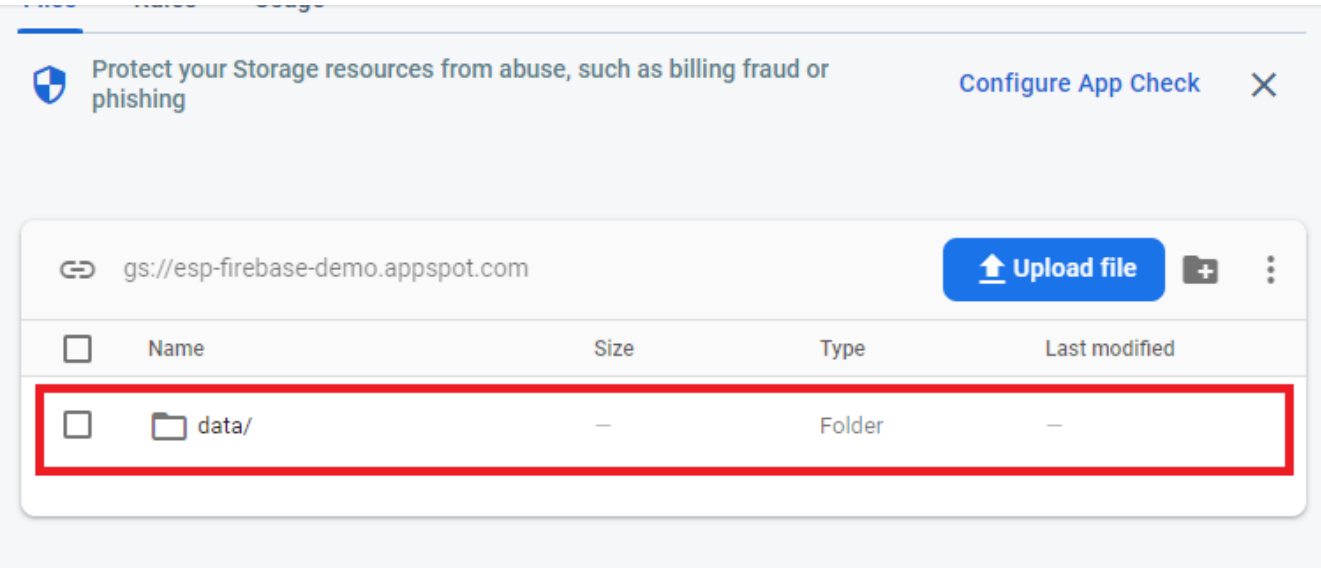
- How to Program / Upload Code to ESP32-CAM AI-Thinker (Arduino IDE)
- Upload Code to ESP32-CAM AI-Thinker using ESP32-CAM-MB USB Programmer

After uploading the code, open the Serial Monitor at a baud rate of 115200. Press the ESP32-CAM on-board RST button.

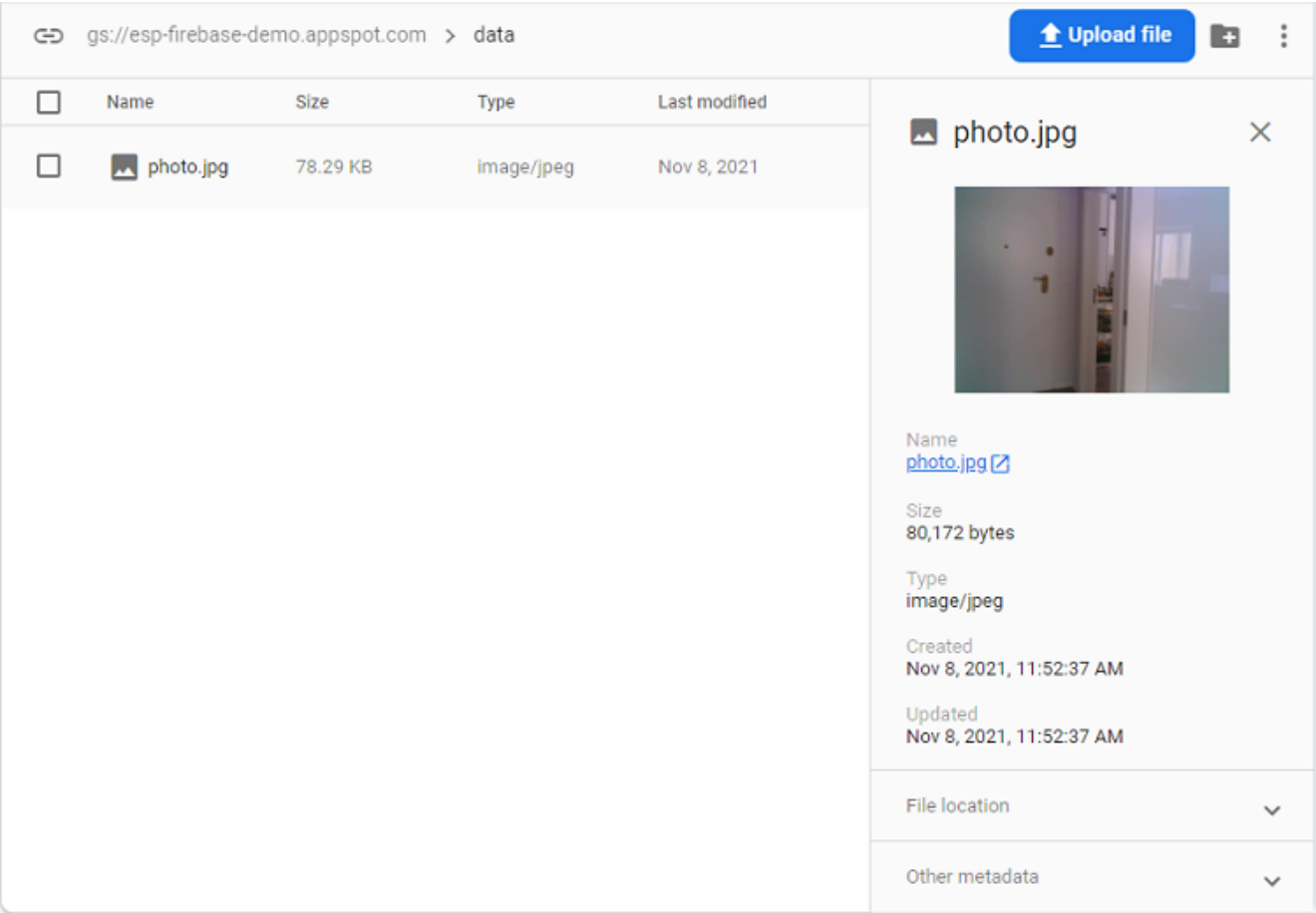
It will attempt to take a picture and will send it to Firebase Storage.



Now, go to your Firebase console, and select the **Storage** tab. There should be a folder called `data` that contains your picture.



You can check some metadata about the picture and view it in full size. You can also access the image by accessing the Download URL printed on the Serial Monitor.



Wrapping Up

going to the Firebase console, or you can build a web app to display those files (we'll do this in a future tutorial).

We've shown a simple example about sending a picture taken with the ESP32-CAM to the Firebase Storage. The example is as simple as possible so that you can understand the basics. The idea is to modify the project to make something useful—like taking a picture and uploading it to Firebase storage when motion is detected, when a door opens or when you press a button.

We have other **Firebase tutorials** that you may like:

- ESP32: Getting Started with Firebase (Realtime Database)
- ESP32 with Firebase – Creating a Web App
- ESP32/ESP8266: Firebase Authentication (Email and Password)
- ESP32/ESP8266 Firebase: Send BME280 Sensor Readings to the Realtime Database

Learn more about the ESP32-CAM with our resources:

- **Build ESP32-CAM Projects (eBook)**
- **Read all our ESP32-CAM Projects, Tutorials and Guides**

Learn how to create a Firebase Web App to control outputs and monitor sensors from anywhere:

- **Firebase Web App with the ESP32 and ESP8266 eBook**

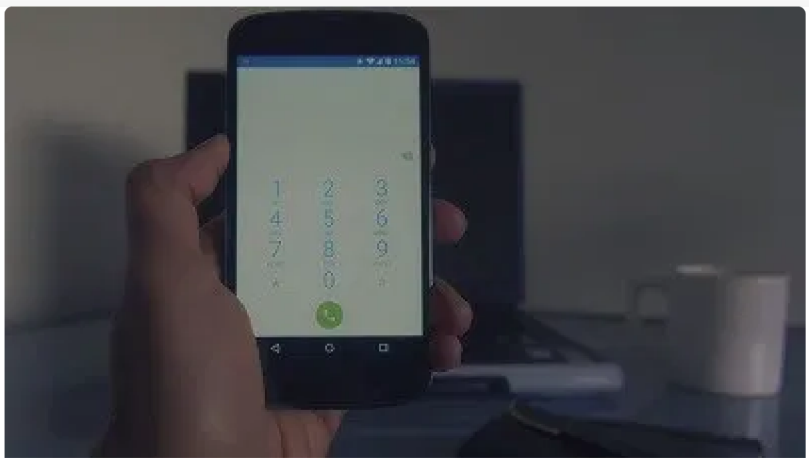
We hope you found this tutorial useful.

Thanks for reading.

[eBook] Build Web Servers with ESP32 and ESP8266 (2nd Edition)

Build Web Server projects with the ESP32 and ESP8266 boards to control outputs and monitor sensors remotely. Learn HTML, CSS, JavaScript and client-server communication protocols **DOWNLOAD »**

Recommended Resources



Build a Home Automation System from Scratch » With Raspberry Pi, ESP8266, Arduino, and Node-RED.