Cloud Storage Overview

Cloud Storage for Firebase is a powerful, simple, and cost-effective object storage service. The Firebase SDKs for Cloud Storage add Google security to file uploads and downloads for your Firebase apps, regardless of network quality.

Check the official page for more information.

Setup

Before starting to use any Firebase extensions, you are required to follow some initial configuration steps.

- Create Project
- Firebase Console
- Platform Setup

Functions

The following functions are provided for working with the Firebase Cloud Storage extension:

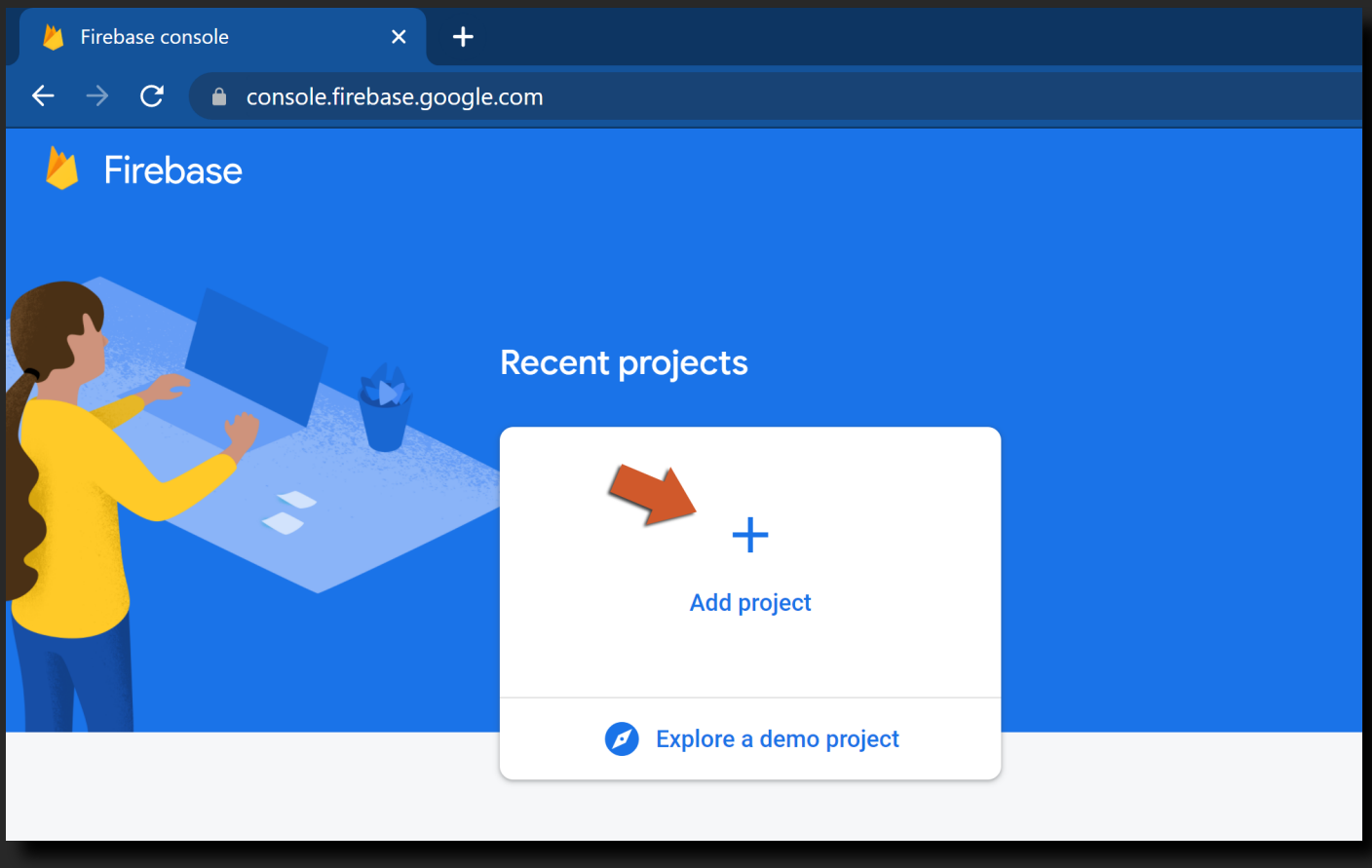
- FirebaseStorage_Cancel
- FirebaseStorage_Delete
- FirebaseStorage_Download

- FirebaseStorage_GetURL
- FirebaseStorage_List
- FirebaseStorage_ListAll
- FirebaseStorage_Upload

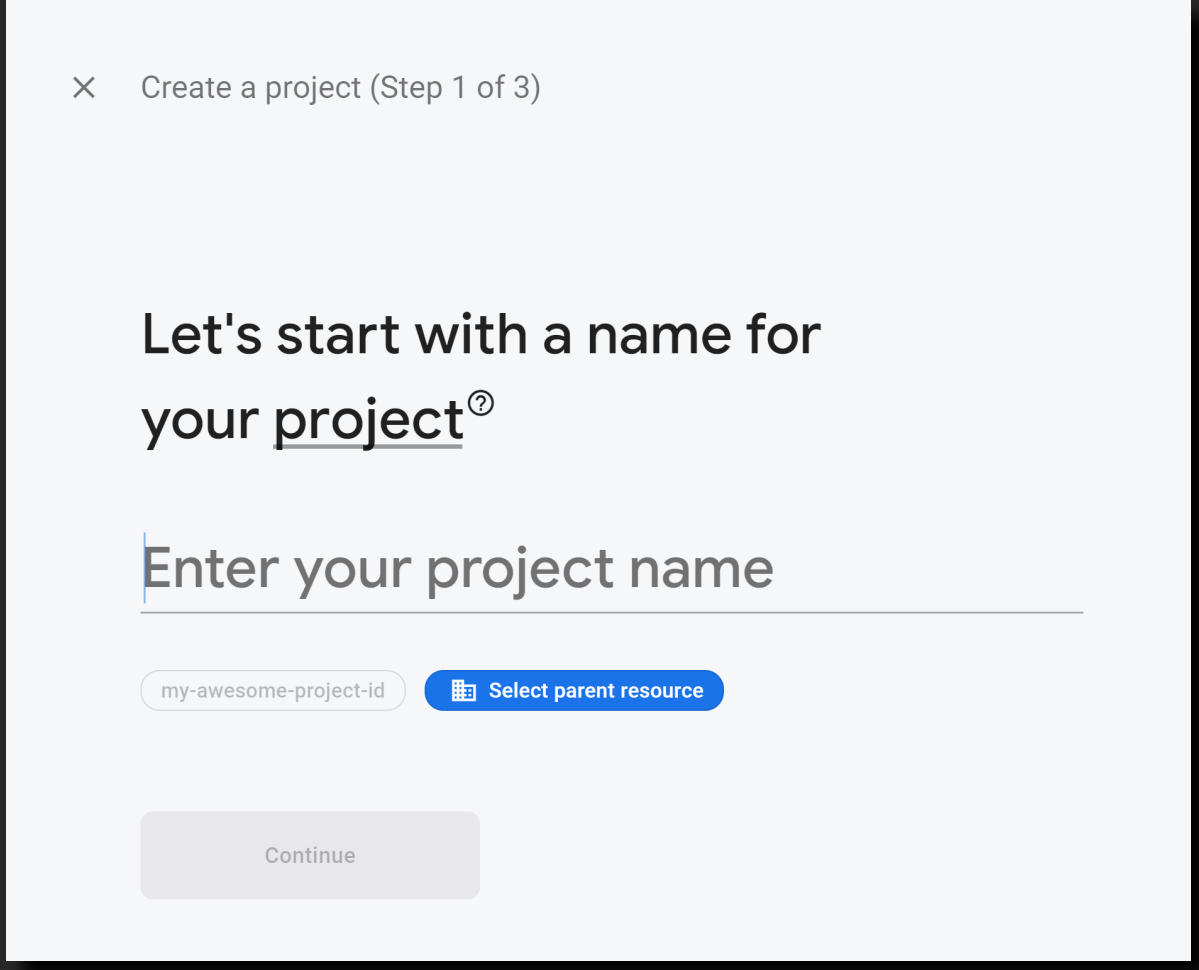
Create Project

Before working with any Firebase functions, you must set up your Firebase project:

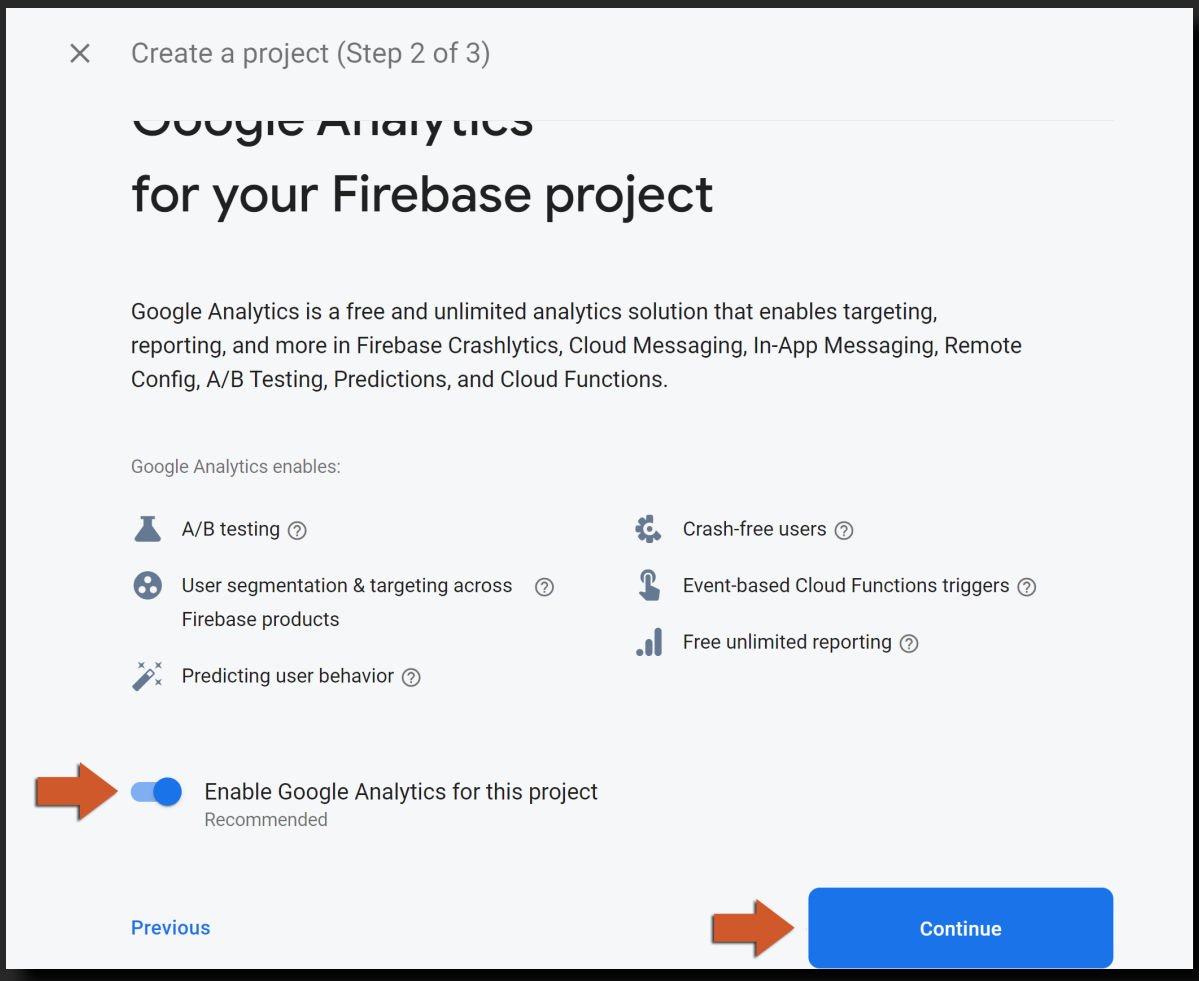
1. Go to the **Firebase Console** web site.
2. Click on **Add Project** to create your new project.



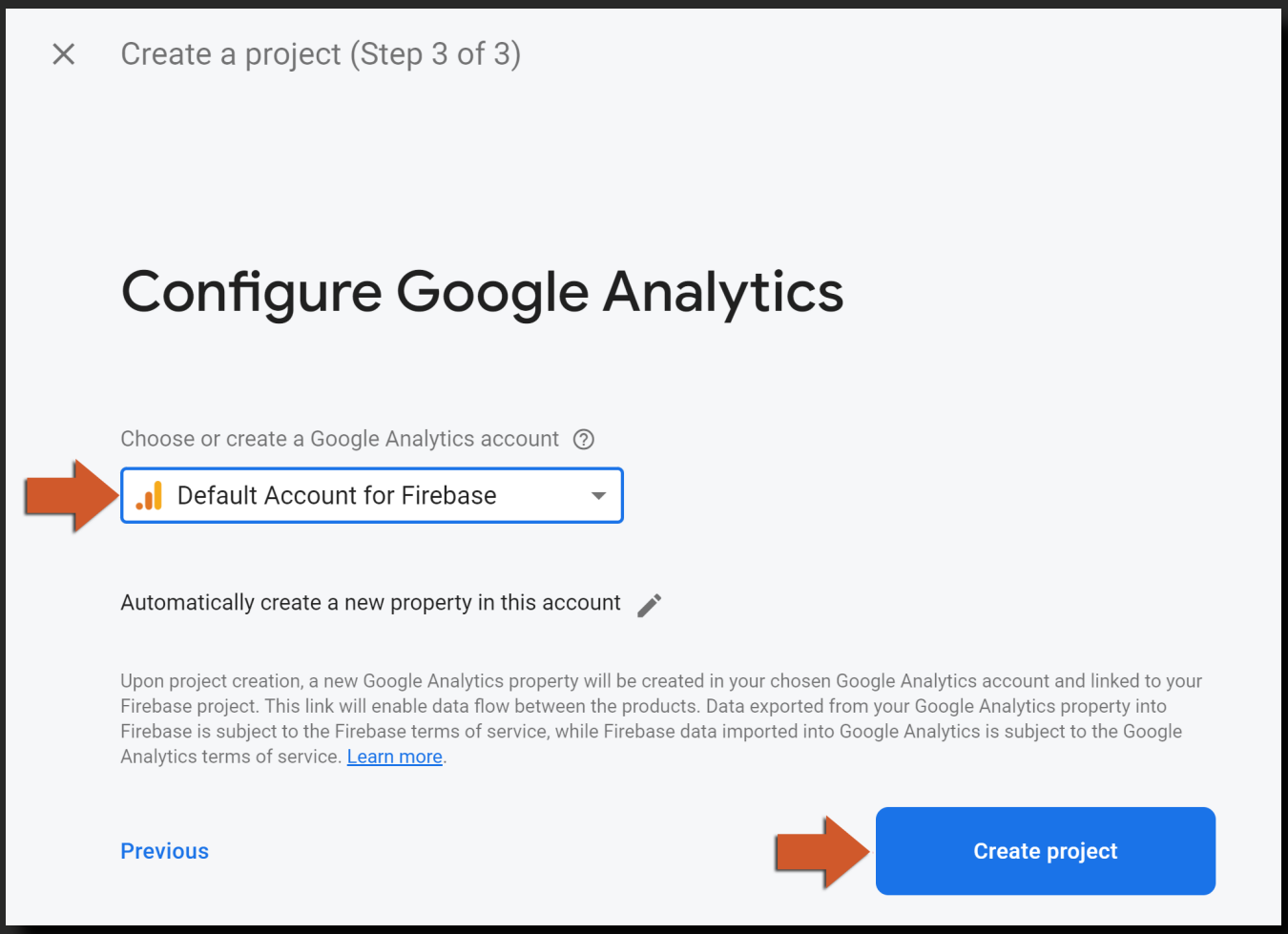
3. Enter a name for your project and click on the **Continue** button.



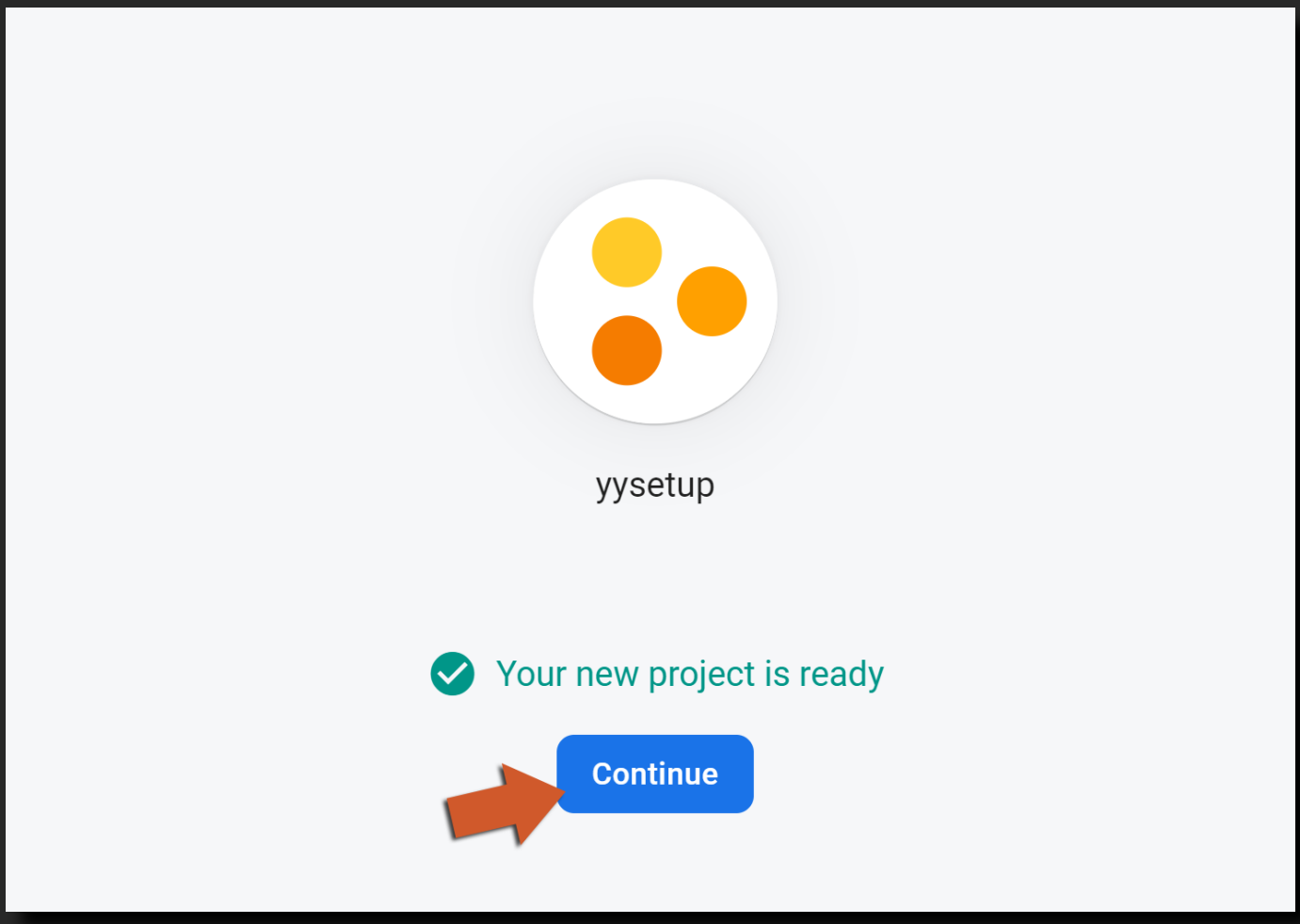
4. On the next page, make sure that **Enable Google Analytics for this project** is enabled and then click the **Continue** button:



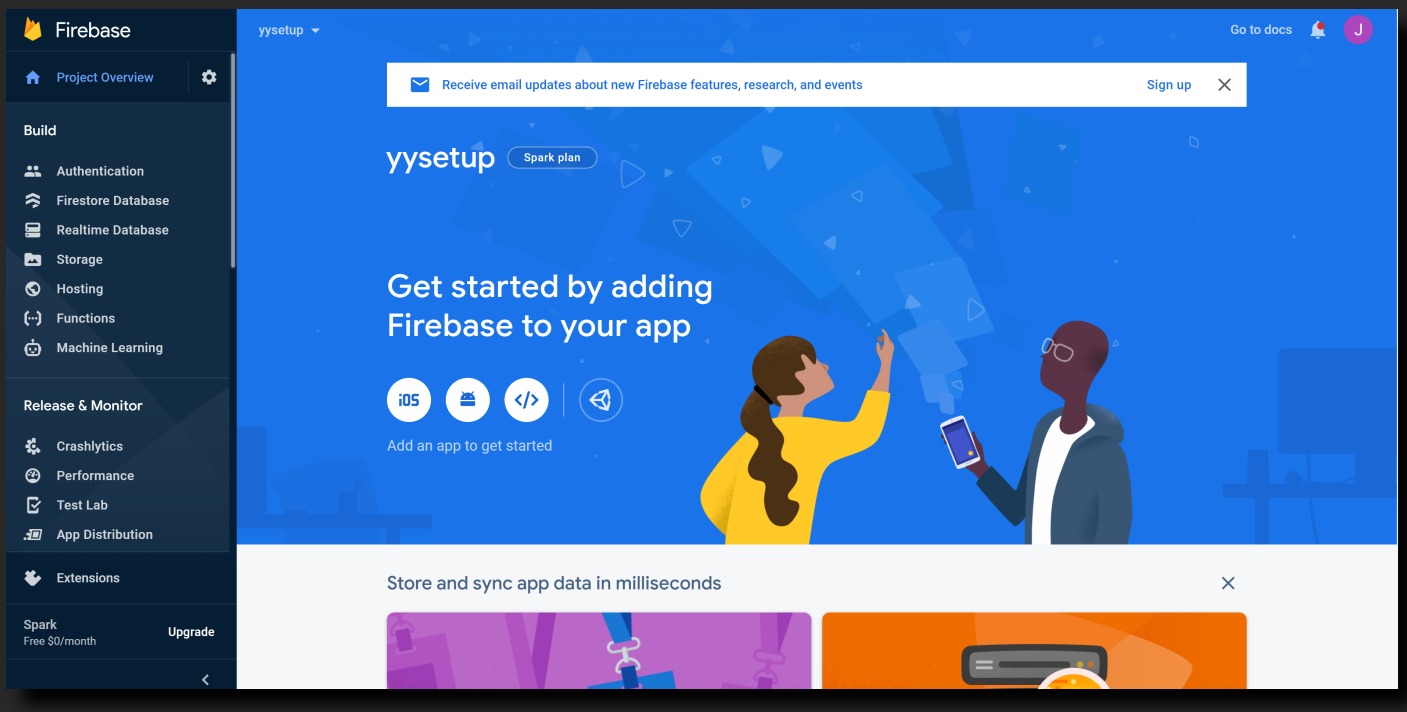
5. Select your account and click the **Create project** button:



6. Wait a moment until you project is created; after a few moments you should see the page shown below:



7. You will now be taken to your new project's home page:

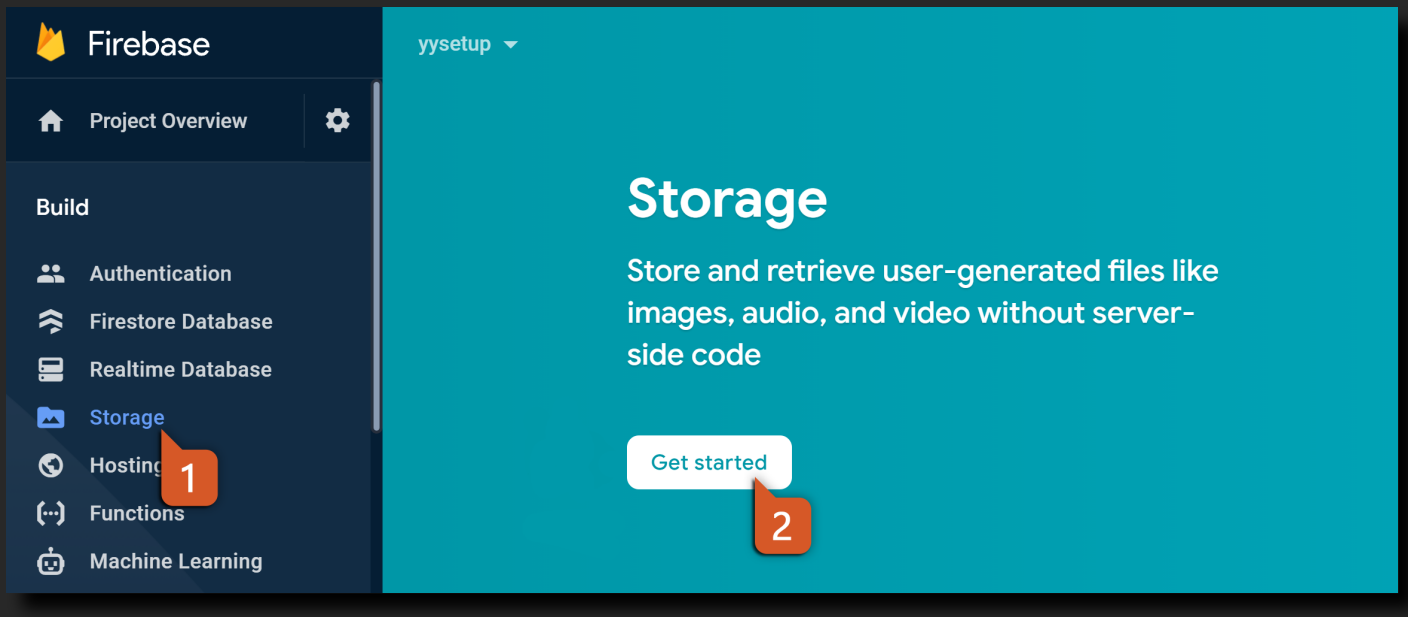


8. Continue your adventure with the Firebase extensions provided for GameMaker!

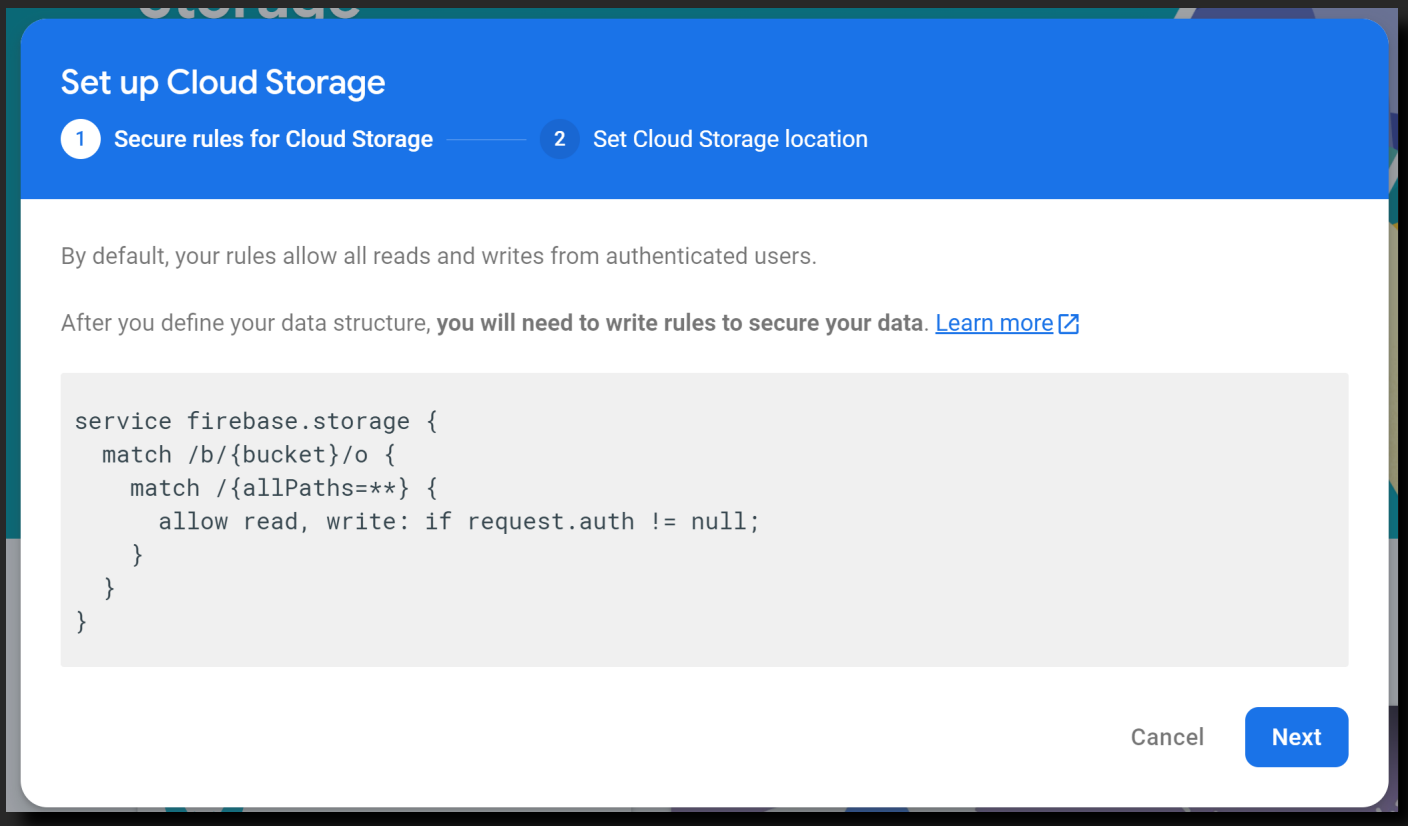
Firebase Console

Before using the Firebase Could Storage extension, you need to configure and enable your storage services in the **Firebase Console**. Follow the steps below to get started:

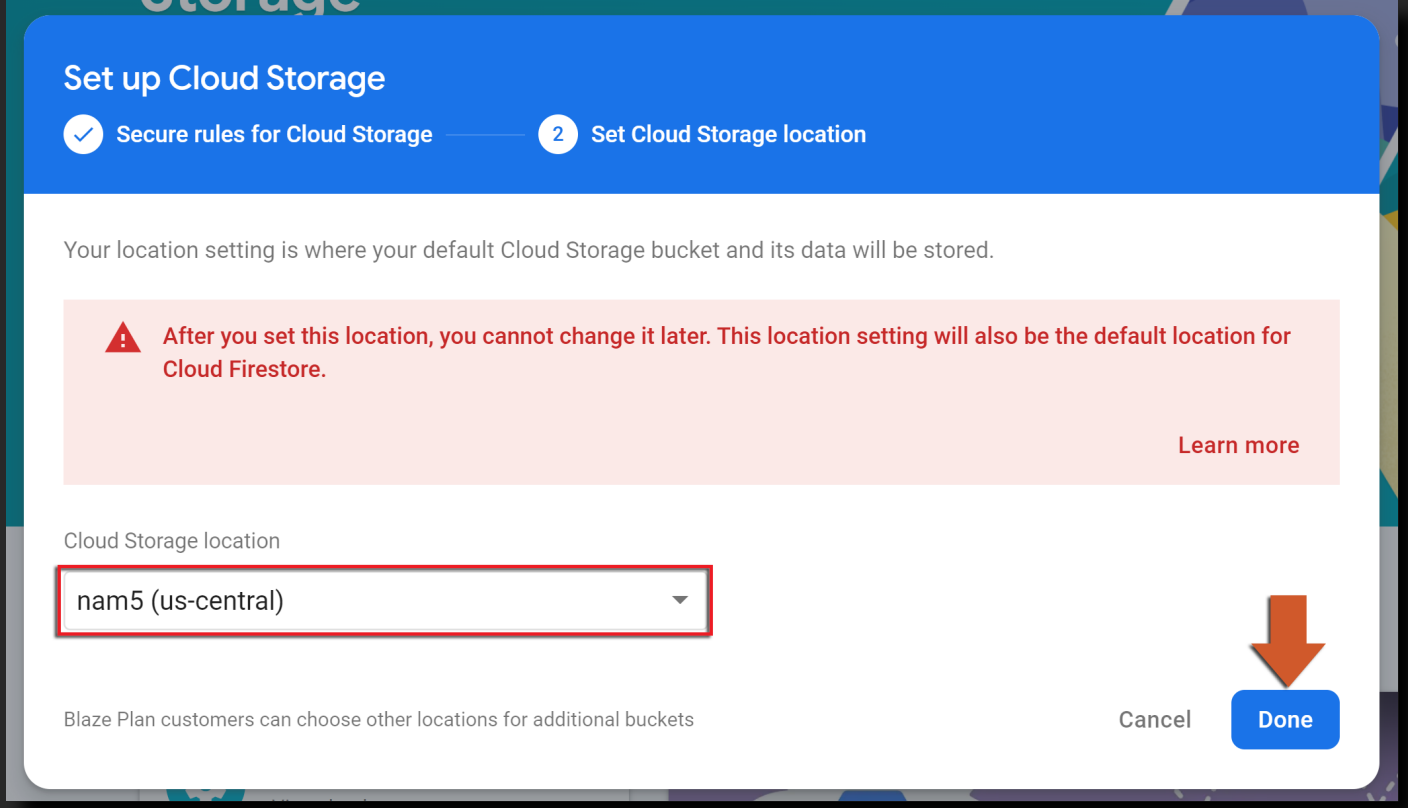
1. Go to **Storage** section and click on the **Get Started** button:



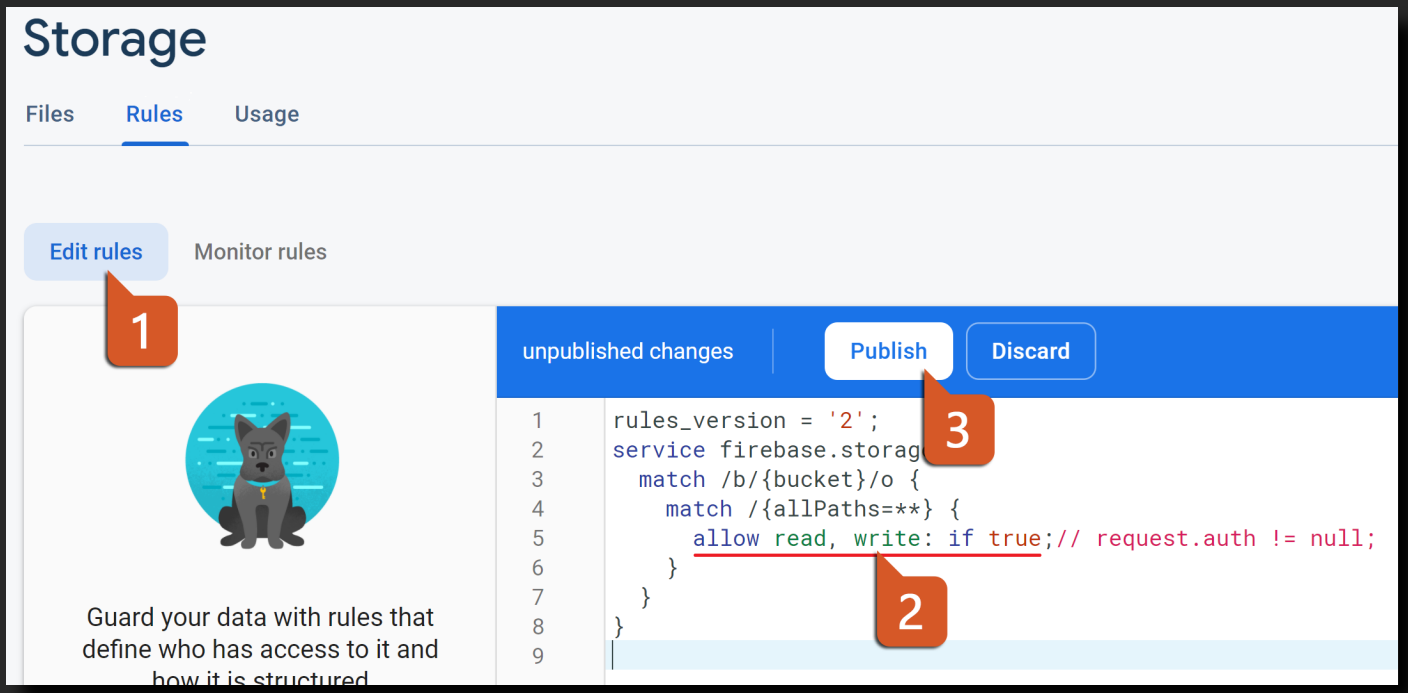
2. You will get a pop-up (we will change the rules later), so click on the **Next** button:



3. Select your **Cloud Storage location** and click on **Done**:



4. Go to **Rules** section and change the rules condition to `true` (this just for testing purposes, and you will need to create your own rules before sending the app to production). Then click on **Publish**.



5. You can now use Firebase Cloud Storage on iOS & Android, and some features on Web.

Platform Setup

Firebase Cloud Storage implementation uses SDK dependencies and therefore is only available on **Android**, **iOS** targets and **Web** with limited functionality. In this section we will cover the required setup necessary to start using the Firebase Cloud Storage extension on your game.

Select your target platform below and follow the simple steps to get your project up and running:

IMPORTANT Web target exports will have limited functionality.

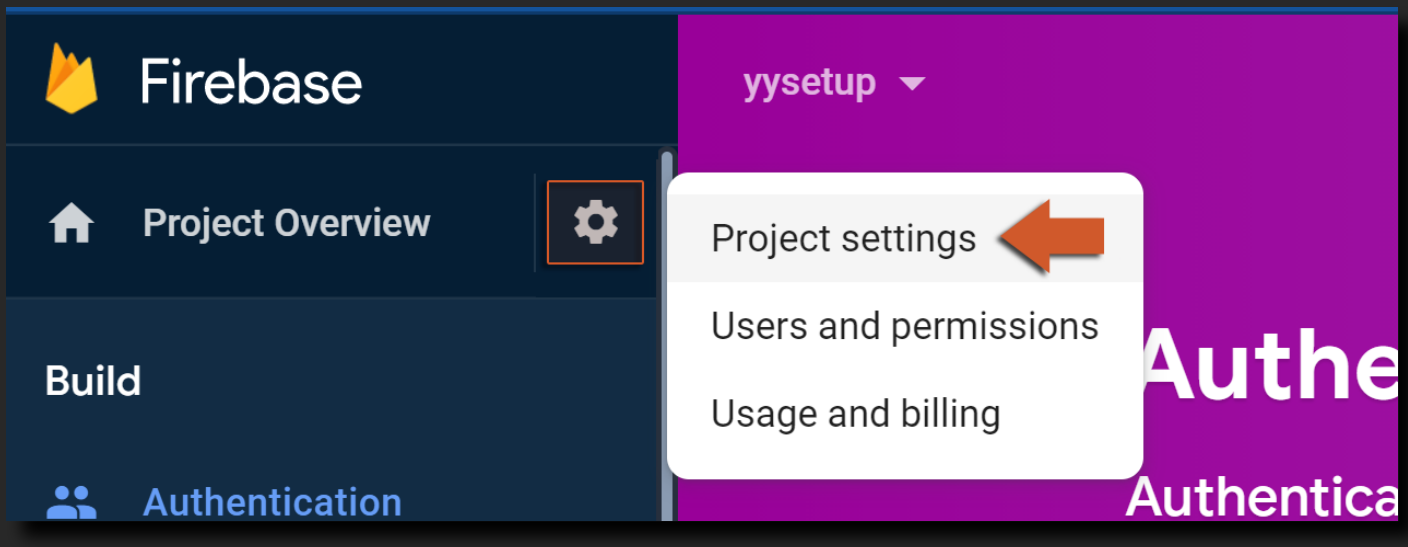
- **Android Setup** (once per project)
- **iOS Setup** (once per project)
- **Web Setup** (once per project)

Android Setup

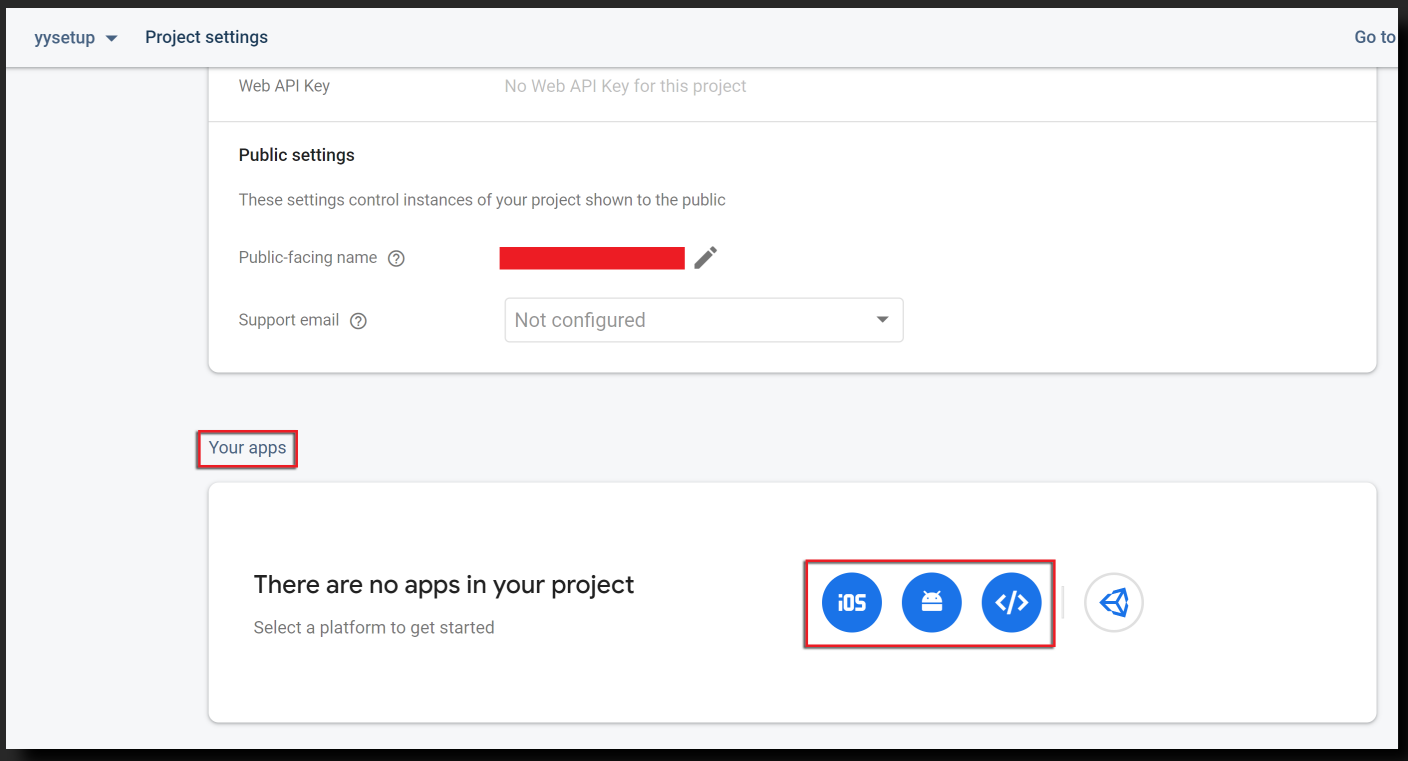
This setup is necessary for all the Firebase modules using Android and needs to be done once per project, and basically involves importing the **google-services.json** file into your project.

IMPORTANT Please refer to **this Helpdesk article** for instructions on setting up an Android project.

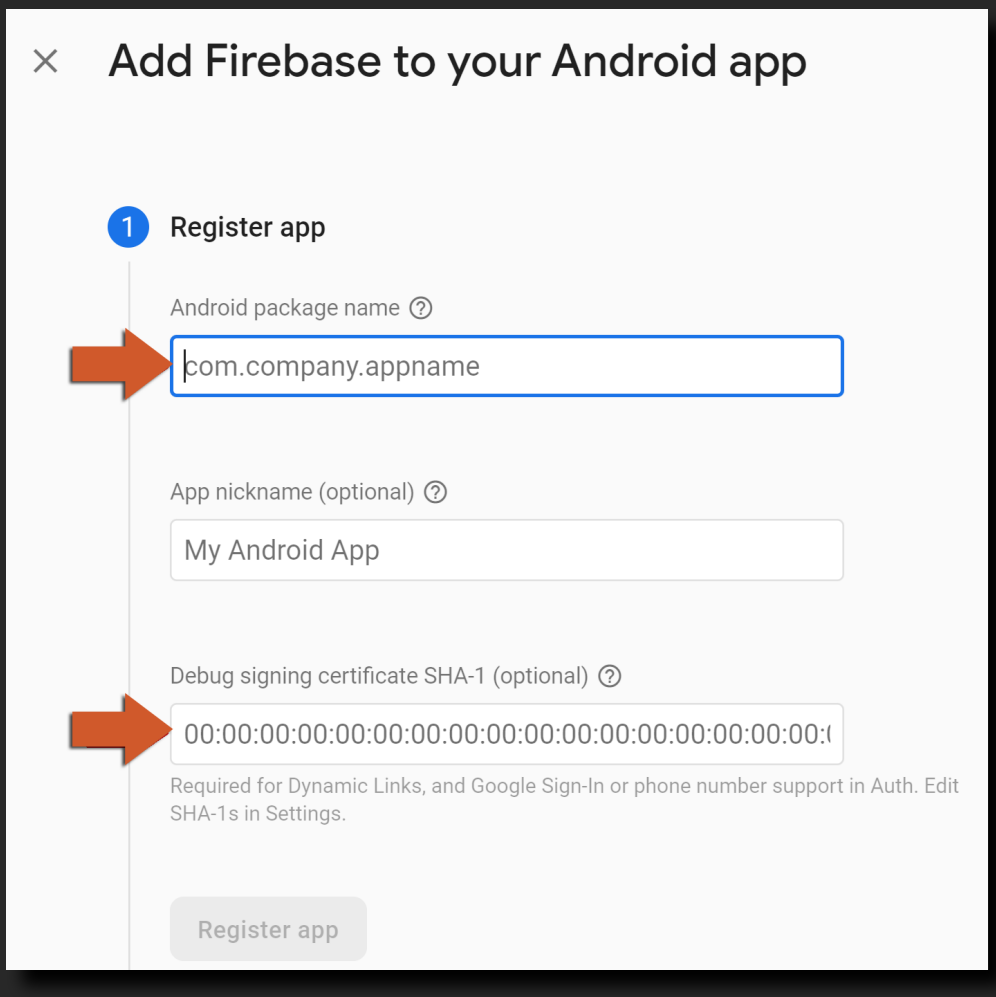
1. Click the **Settings** icon (next to Project Overview) and select **Project settings:**



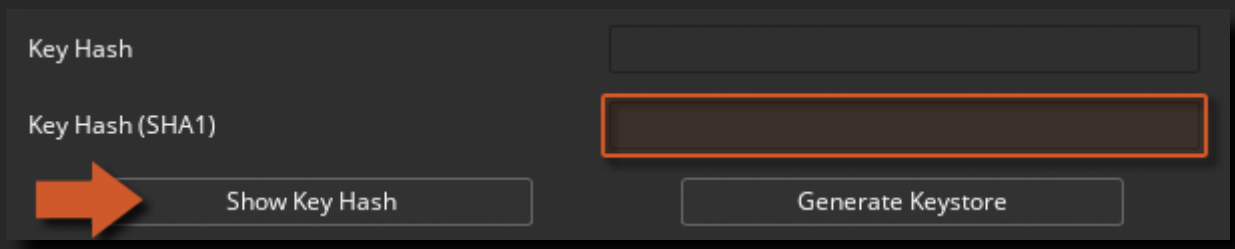
2. Now go to the **Your apps** section and click on the **Android** button:



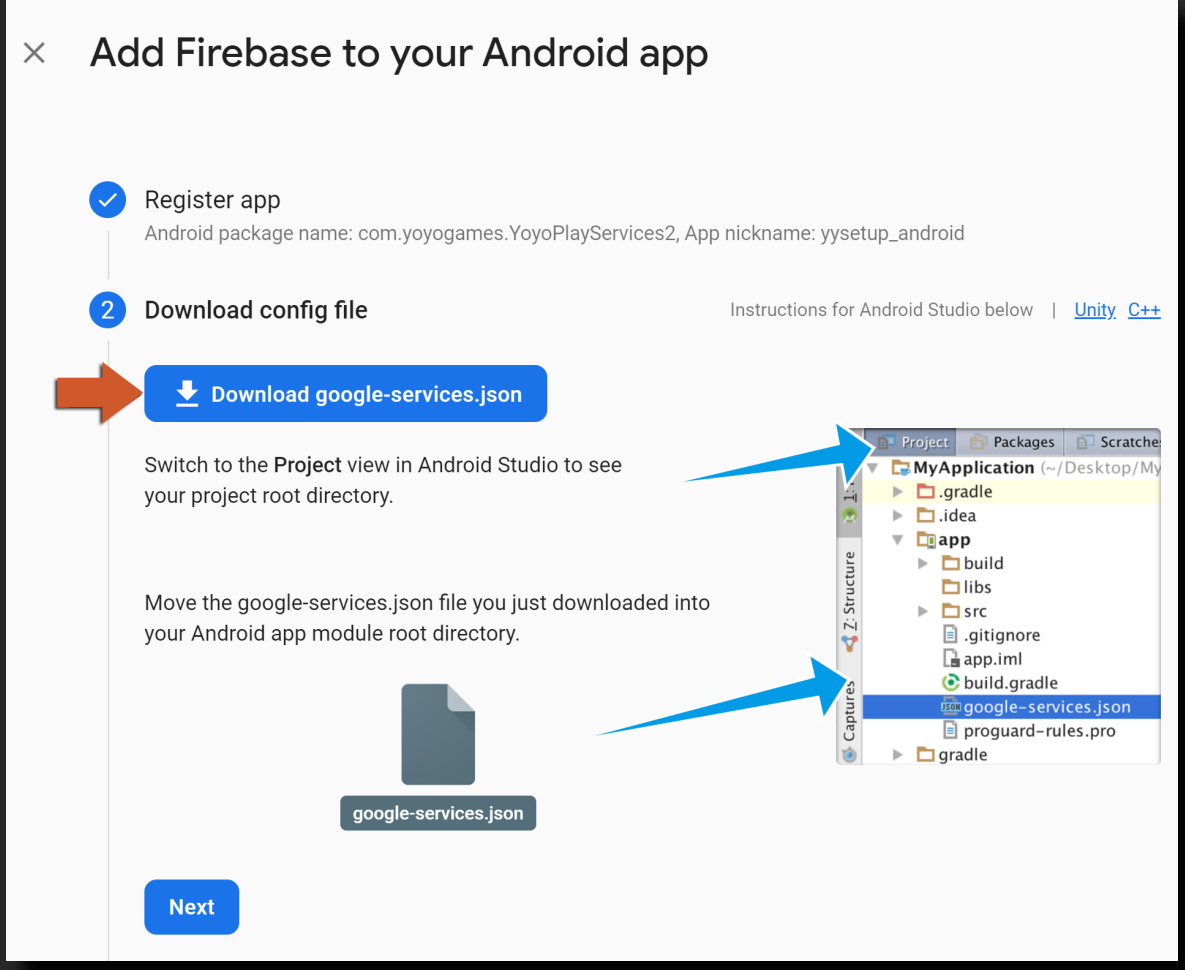
3. On this screen you need enter your **Package name** (required), **App nickname** (optional) and **Debug signing certificate SHA-1** (required if you are using Firebase Authentication).



You can get your package name from the **Android Game Options**, and your **Debug signing certificate SHA-1** from the **Android Preferences** (under Keystore):

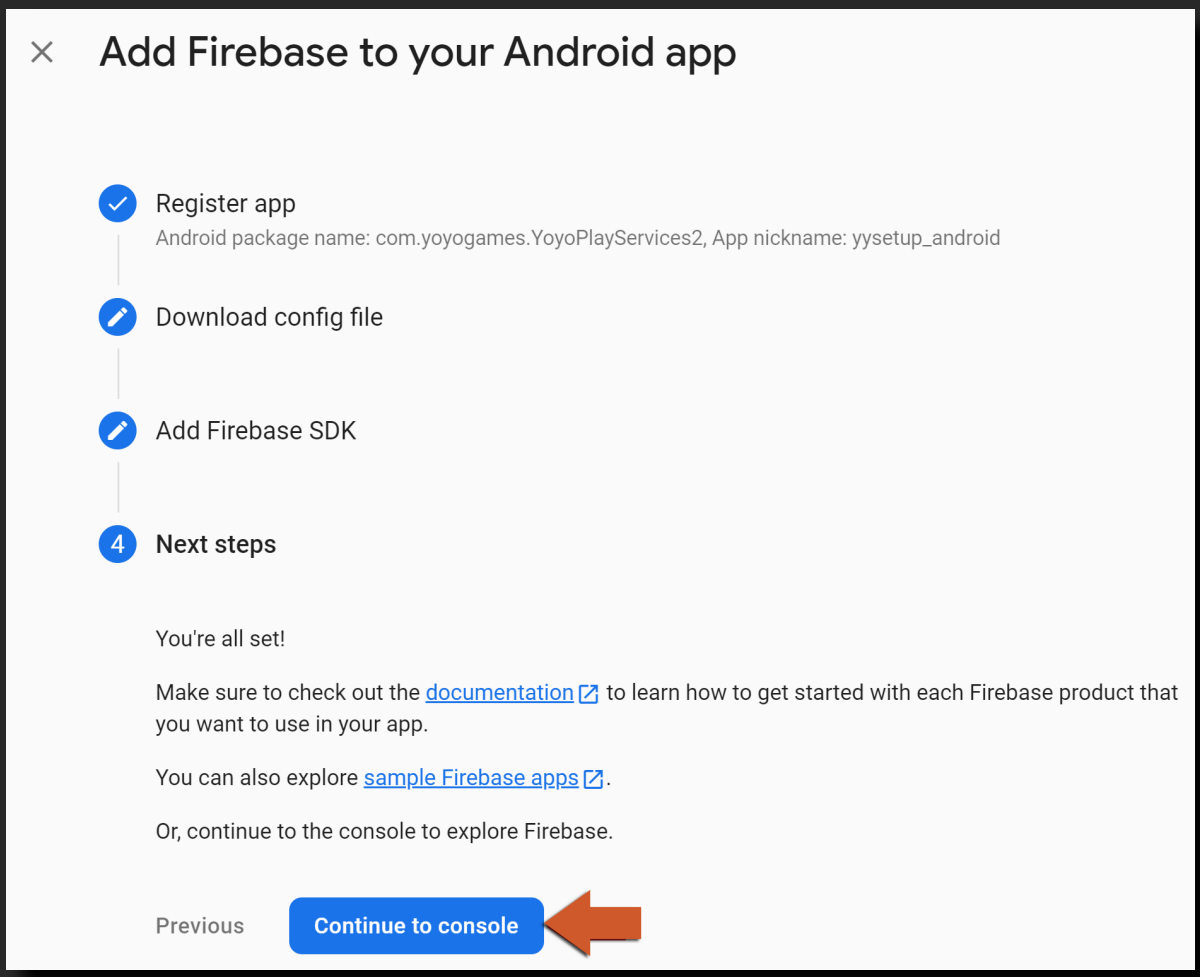


4. Click on **Download google-services.json** (make sure to save this file as we will need it in subsequent steps).

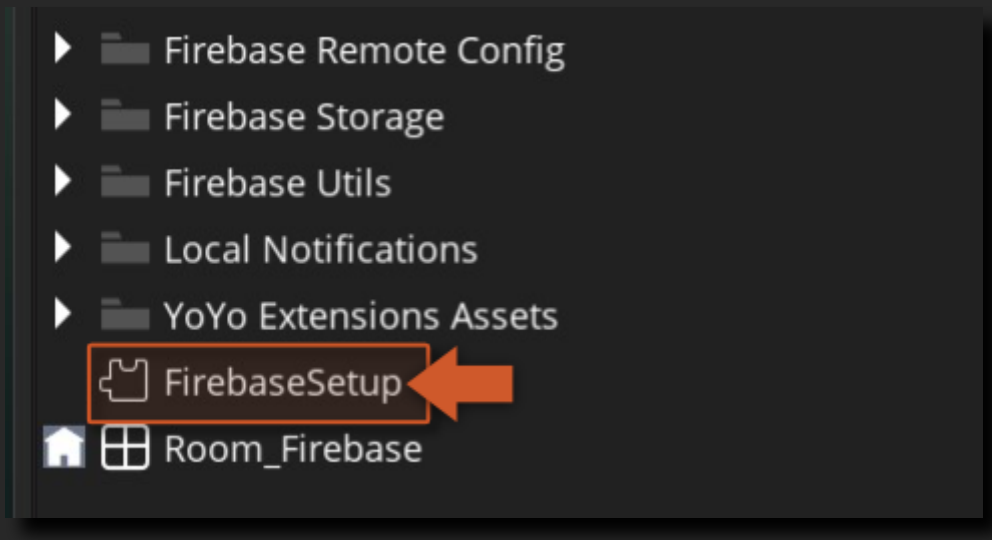


5. Ignore this screen, as this is already done in the extension.

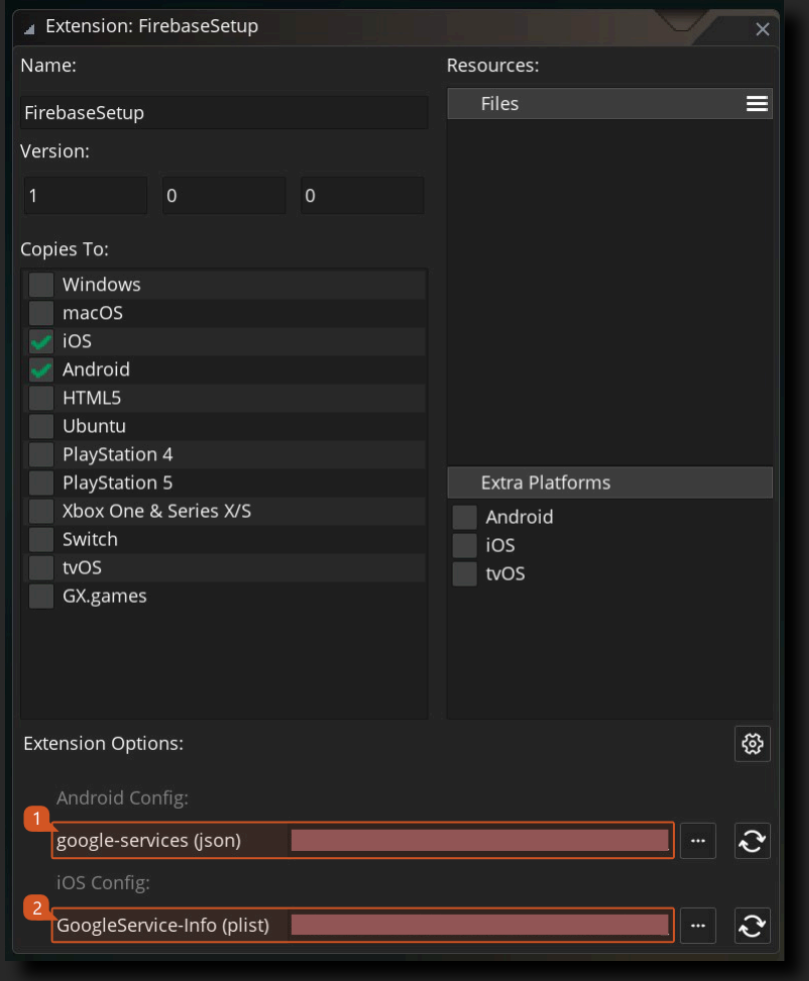
6. Click on the **Continue to console** button.



7. Now go into GameMaker, double click the extension **FirebaseSetup** asset.



8. In the extension panel just fill in the paths for the correct files (Android and/or iOS).



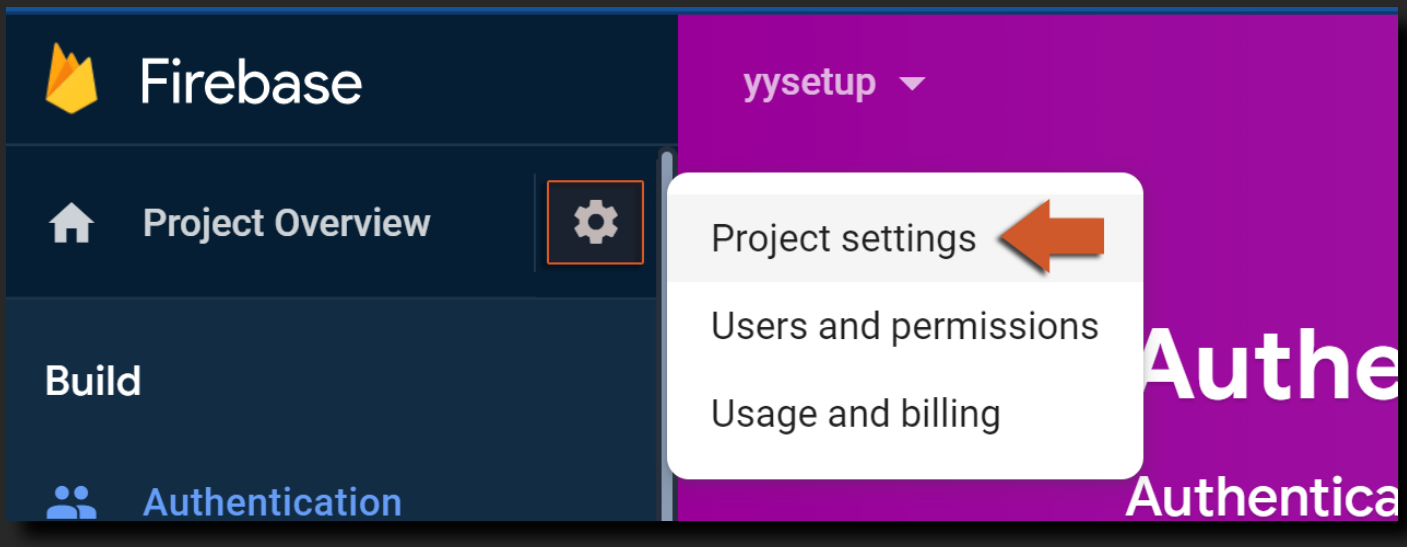
9. You have now finished the main setup for all Firebase Android modules!

iOS Setup

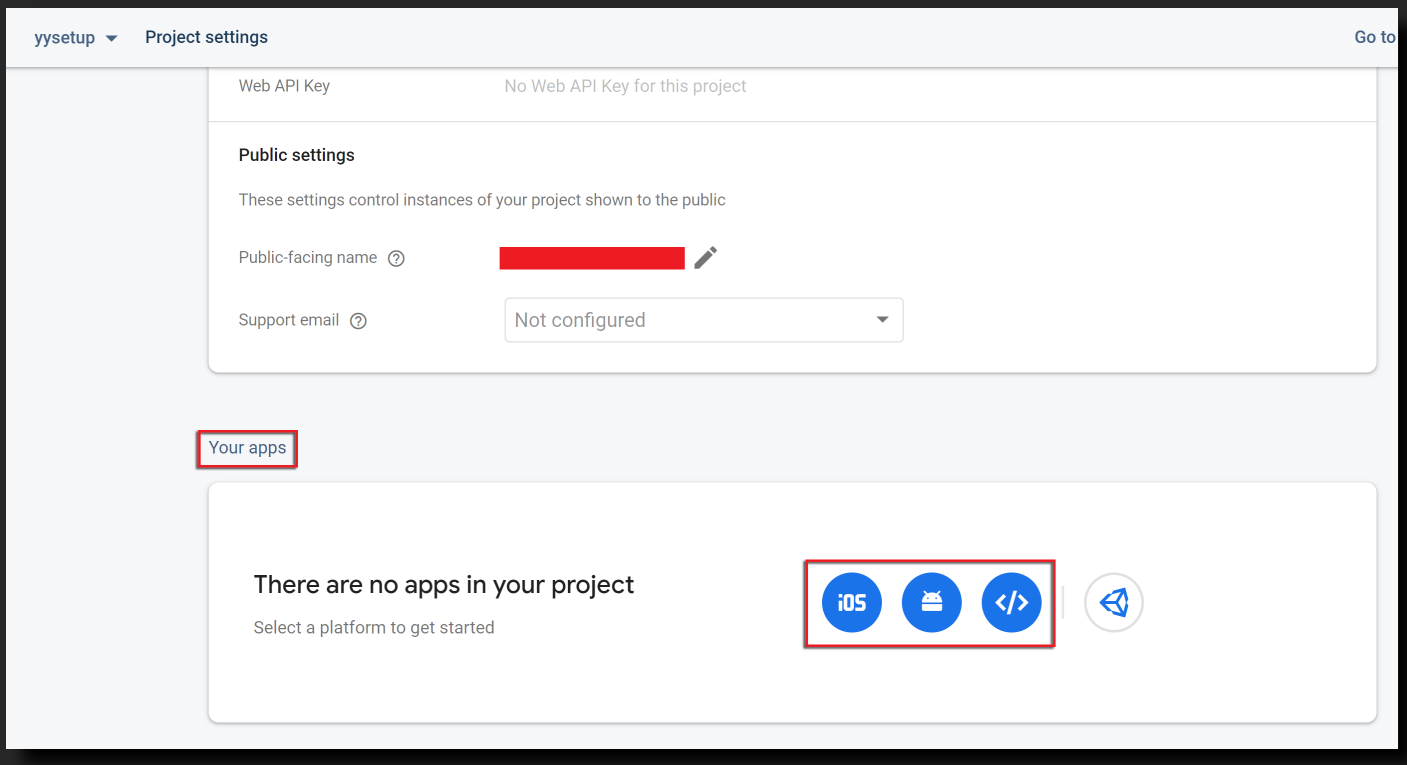
This setup is necessary for all the Firebase modules using iOS and needs to be done once per project, and basically involves importing the **GoogleServices-Info.plist** file into your project.

IMPORTANT Please refer to **this Helpdesk article** for instructions on setting up an iOS project.

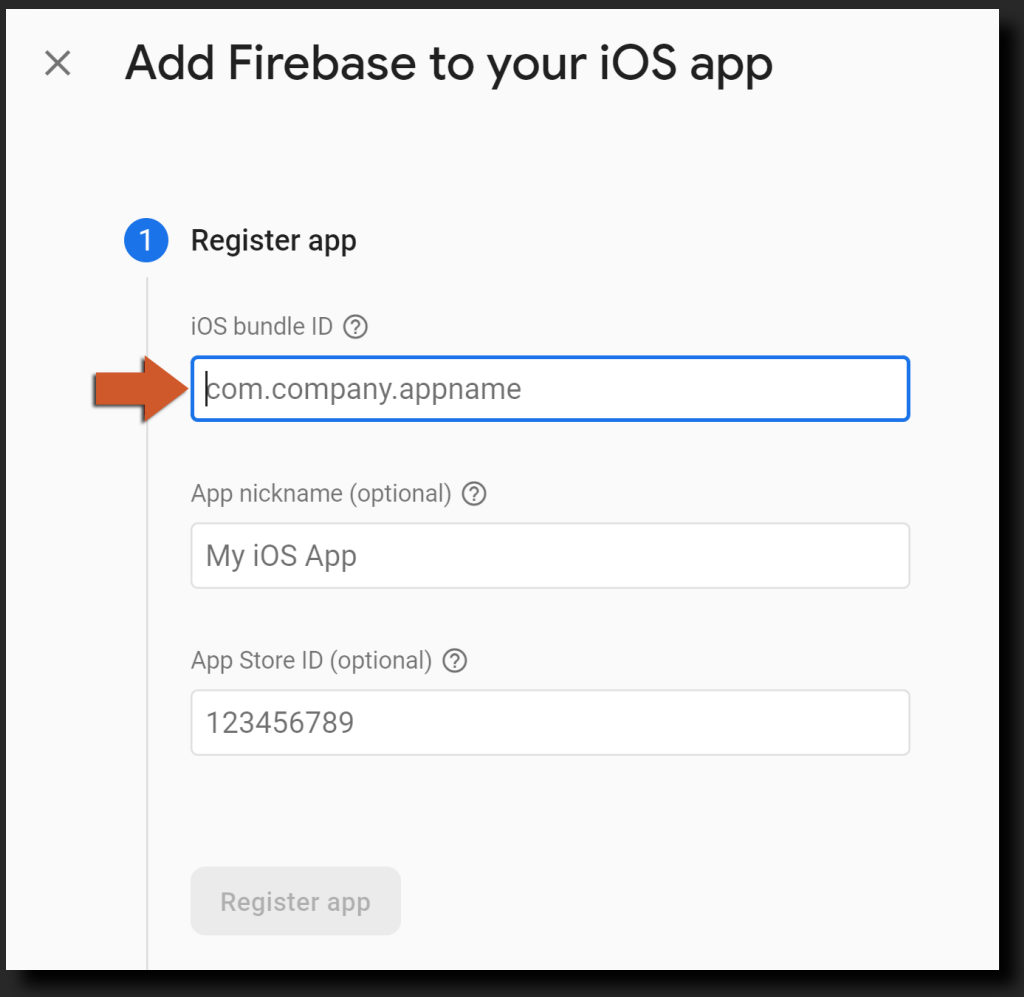
1. Click the **Settings** icon (next to Project Overview) and select **Project settings:**



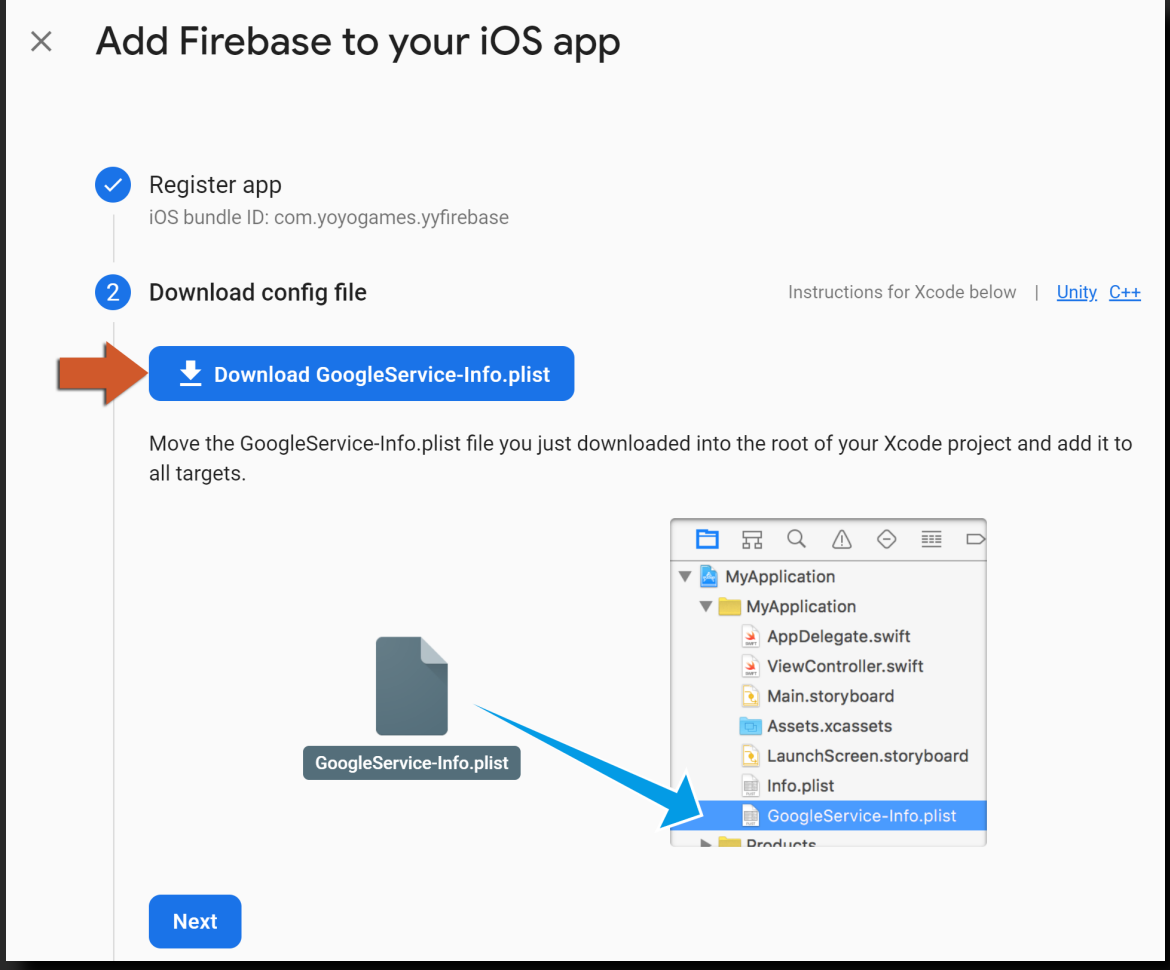
2. Now go to the **Your apps** section and click on the **iOS** button:



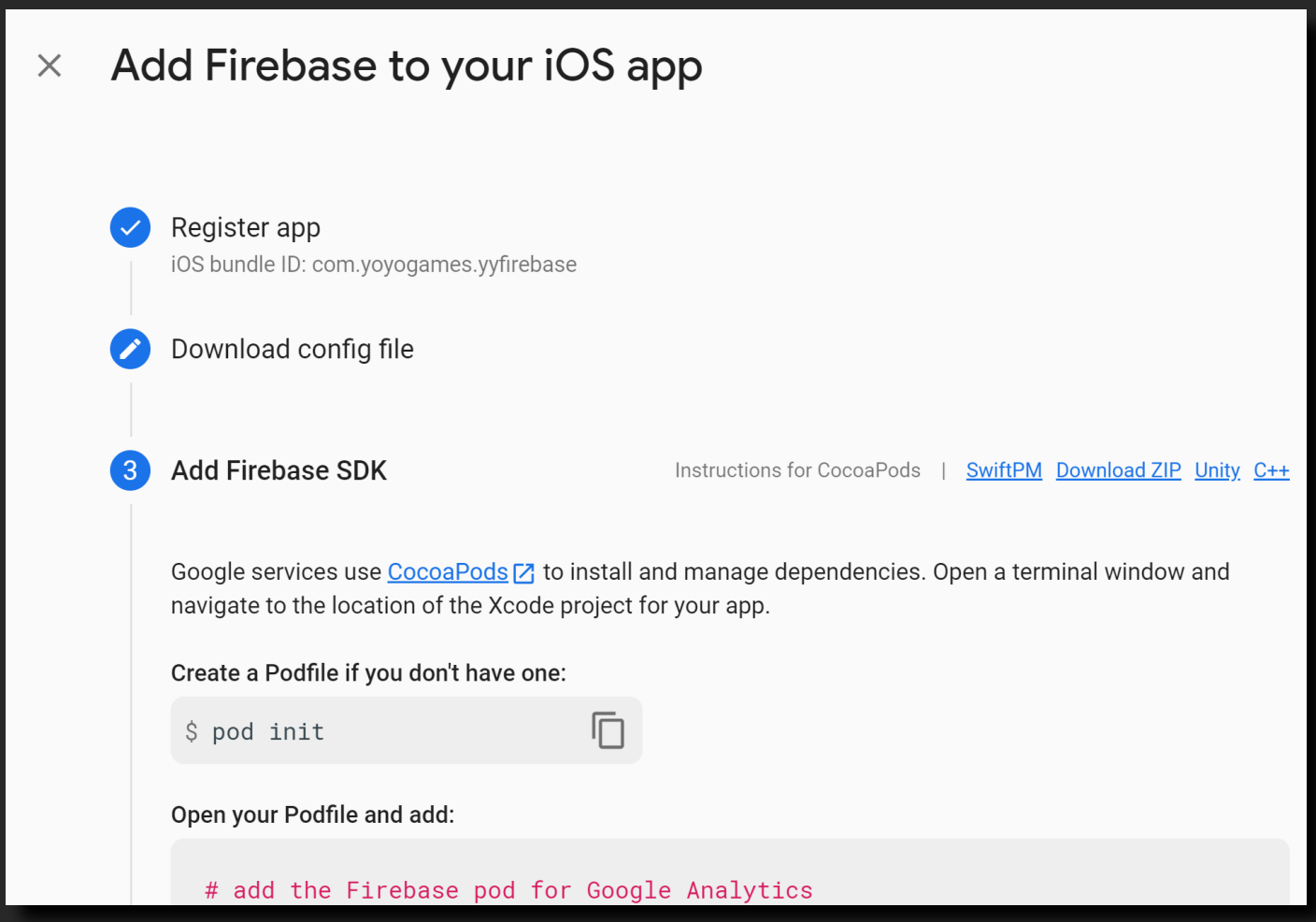
3. Fill the form with your **iOS Bundle ID**, **App nickname** and **AppStore ID** (last two are optional).



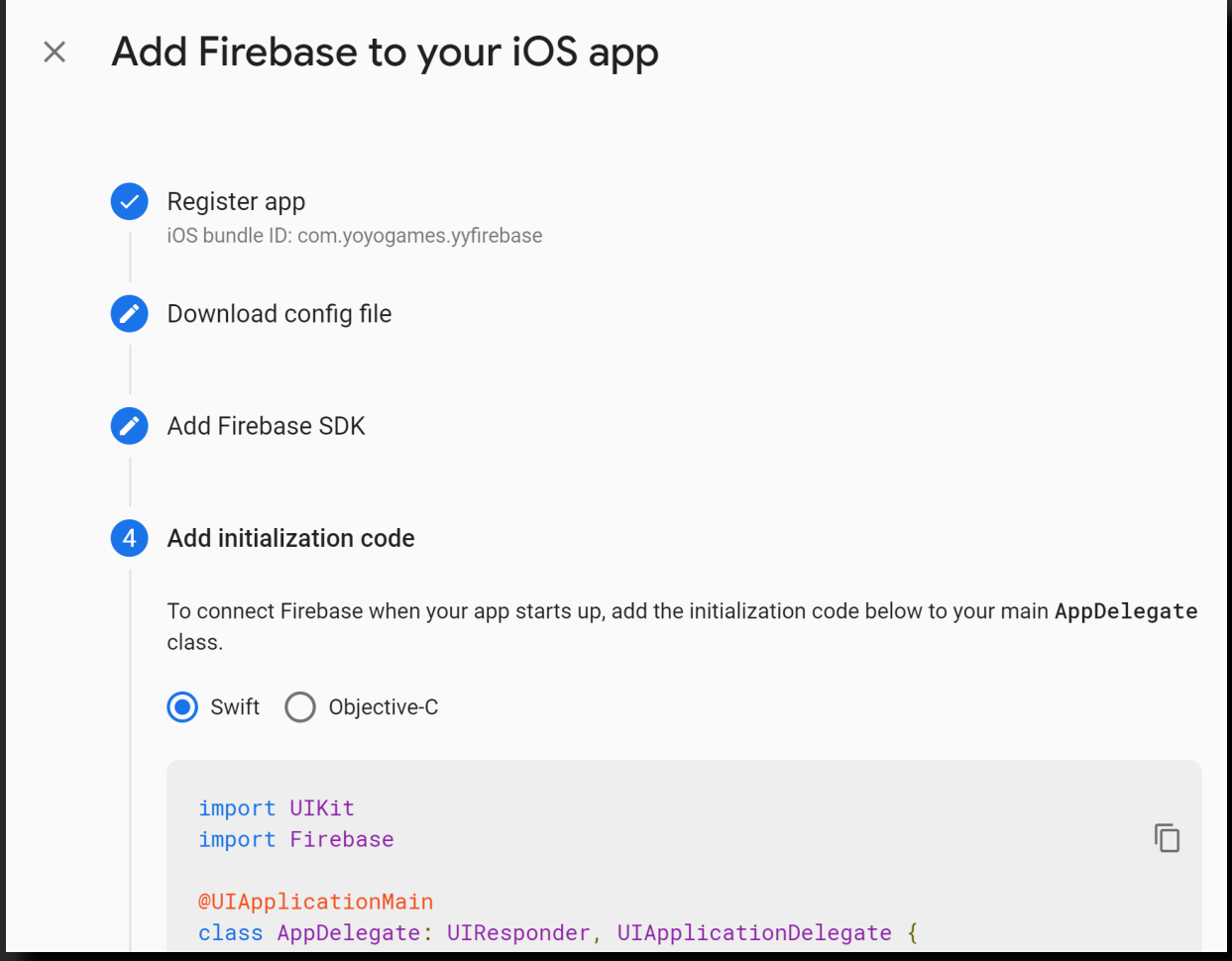
4. Click on **Download GoogleService-info.plist** (make sure to save this file as we will need it in subsequent steps).



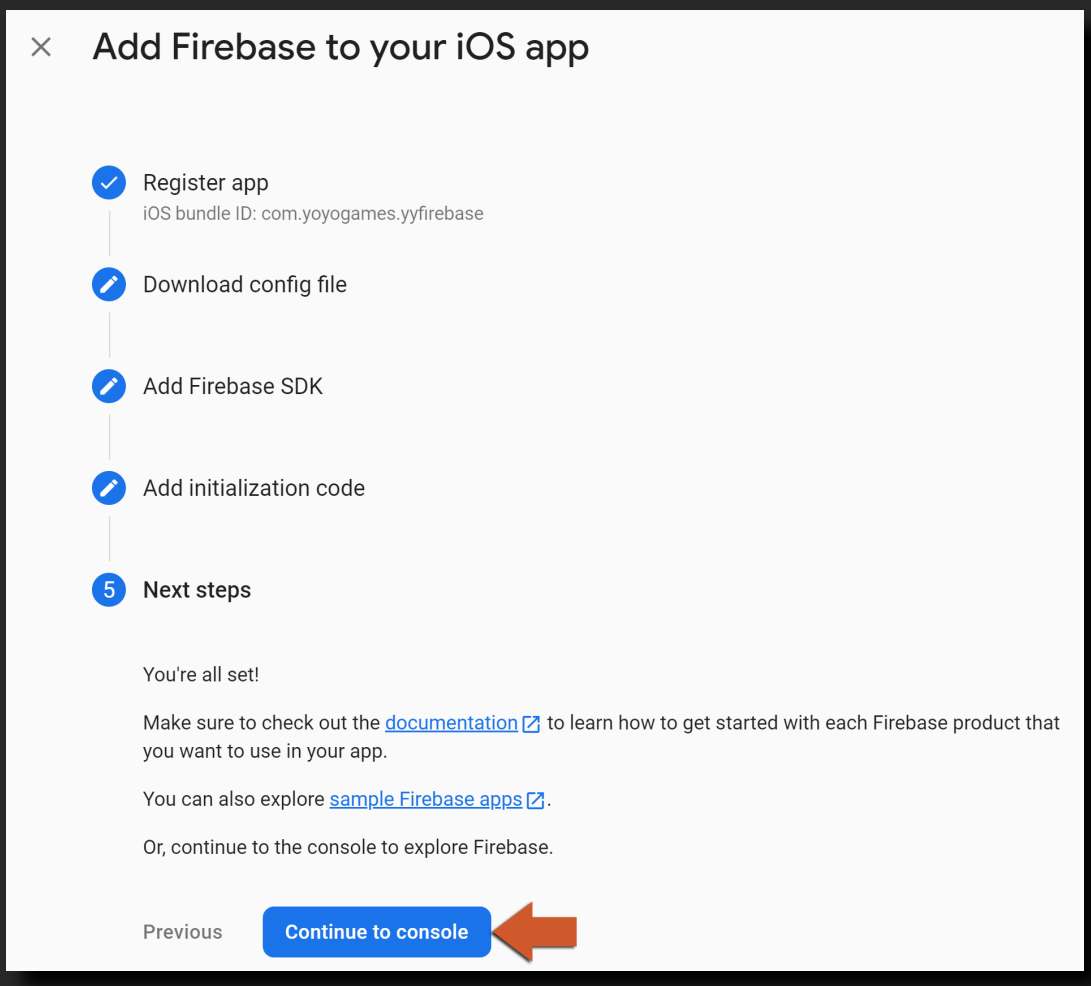
5. Ignore this screen, as this is already done in the extension.



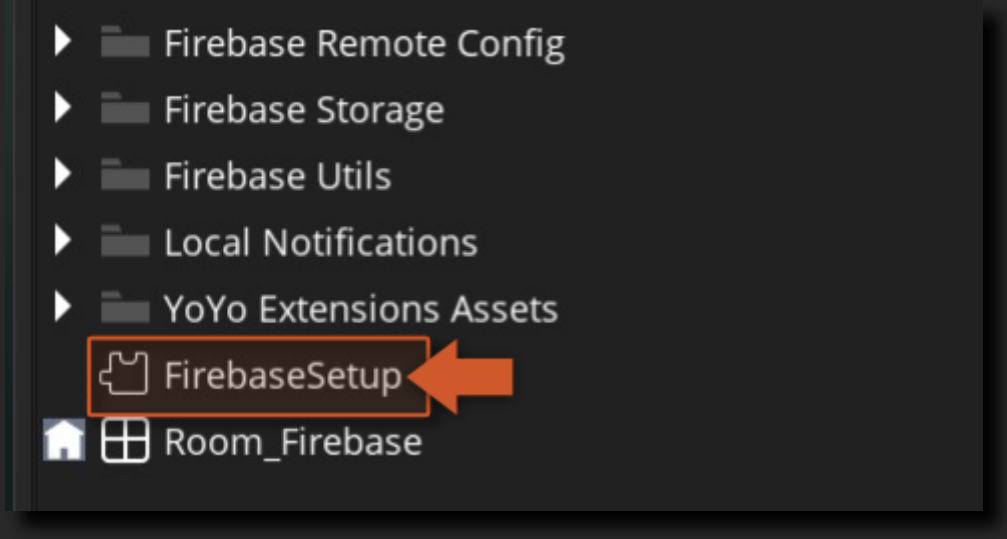
6. Ignore this screen as well, as this is also done in the extension.



7. Click on the **Continue to console** button:



8. Now go into GameMaker, double click the extension **FirebaseSetup** asset.



9. In the extension panel just fill in the paths for the correct files (Android and/or iOS).

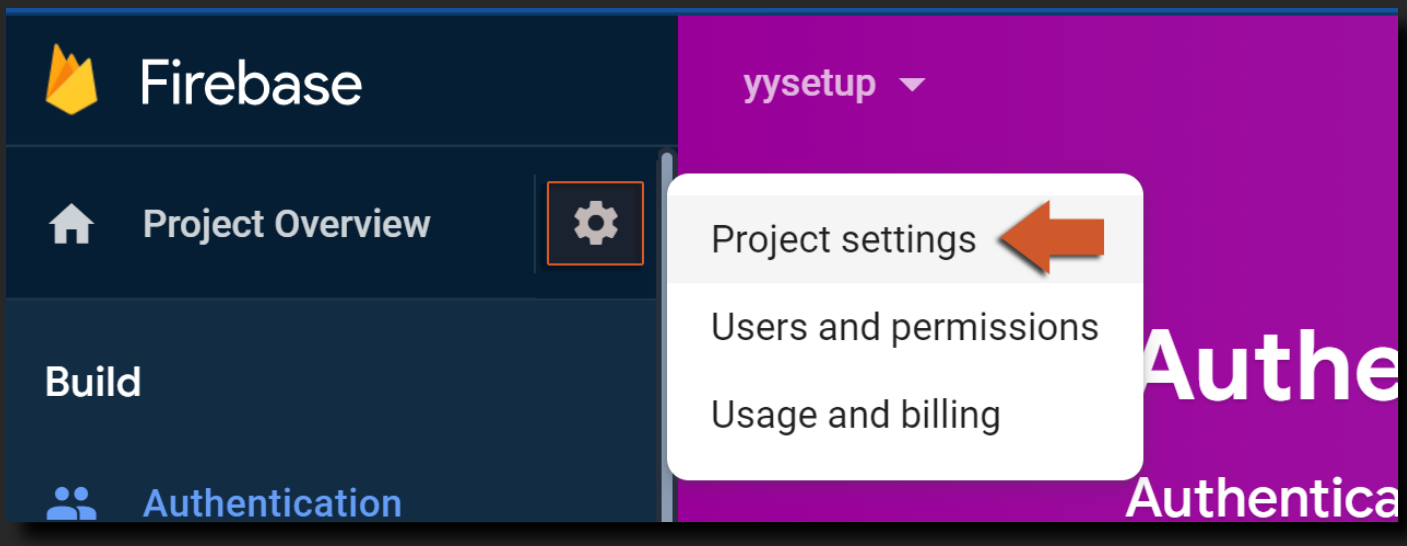
10. Make sure to set up **CocoaPods** for your project *unless* you are only using the REST API in an extension (if one is provided -- not all extensions provide a REST API) or the Firebase Cloud Functions extension (which only uses a REST API).

11. You have now finished the main setup for all Firebase iOS modules!

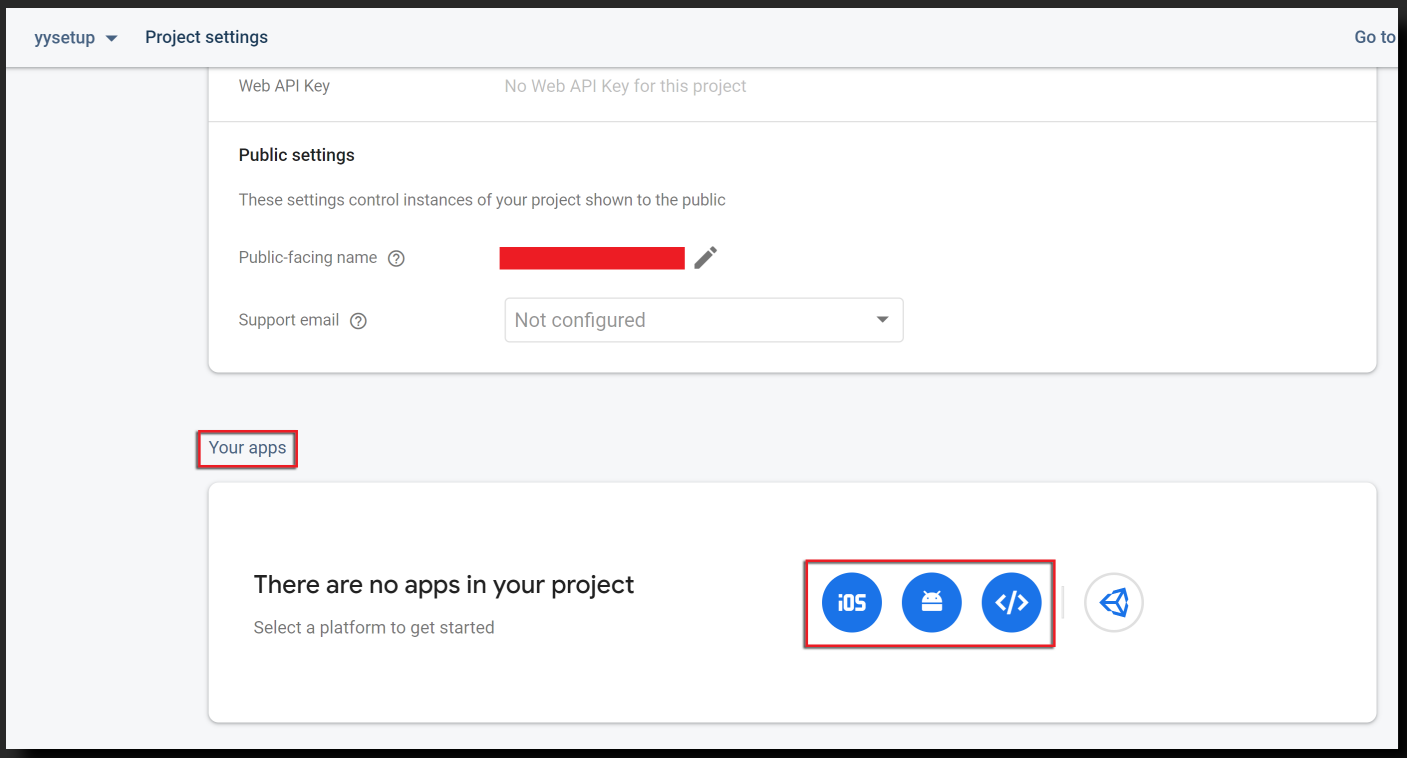
Web Setup

This setup is necessary for all the Firebase modules using Web export and needs to be done once per project, and basically involves adding Firebase libraries and your Firebase values to the **index.html** file in your project.

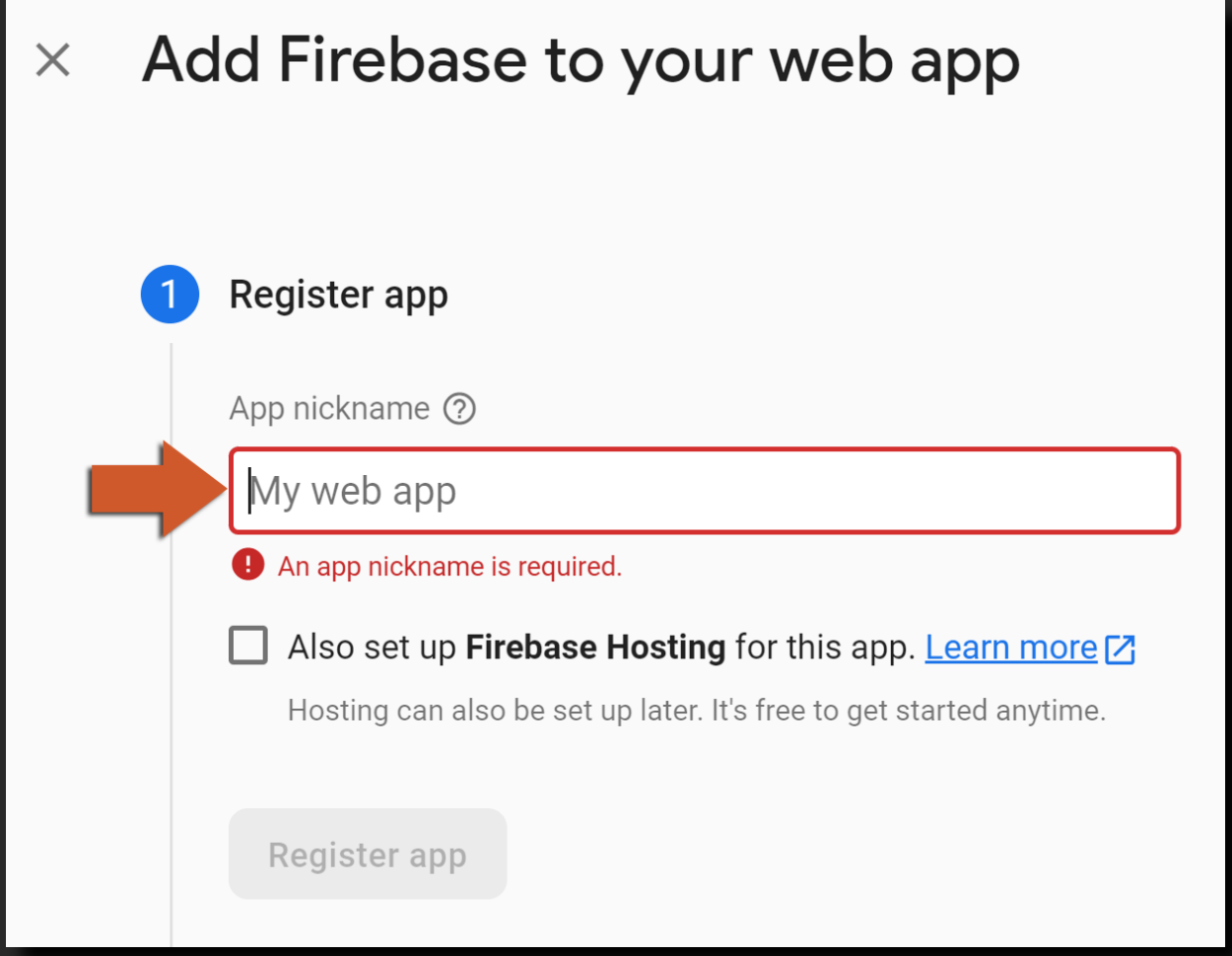
1. Click the **Settings** icon (next to Project Overview) and then **Project settings:**



2. Now go to the **Your apps** section and click on the **Web** (</>) button:



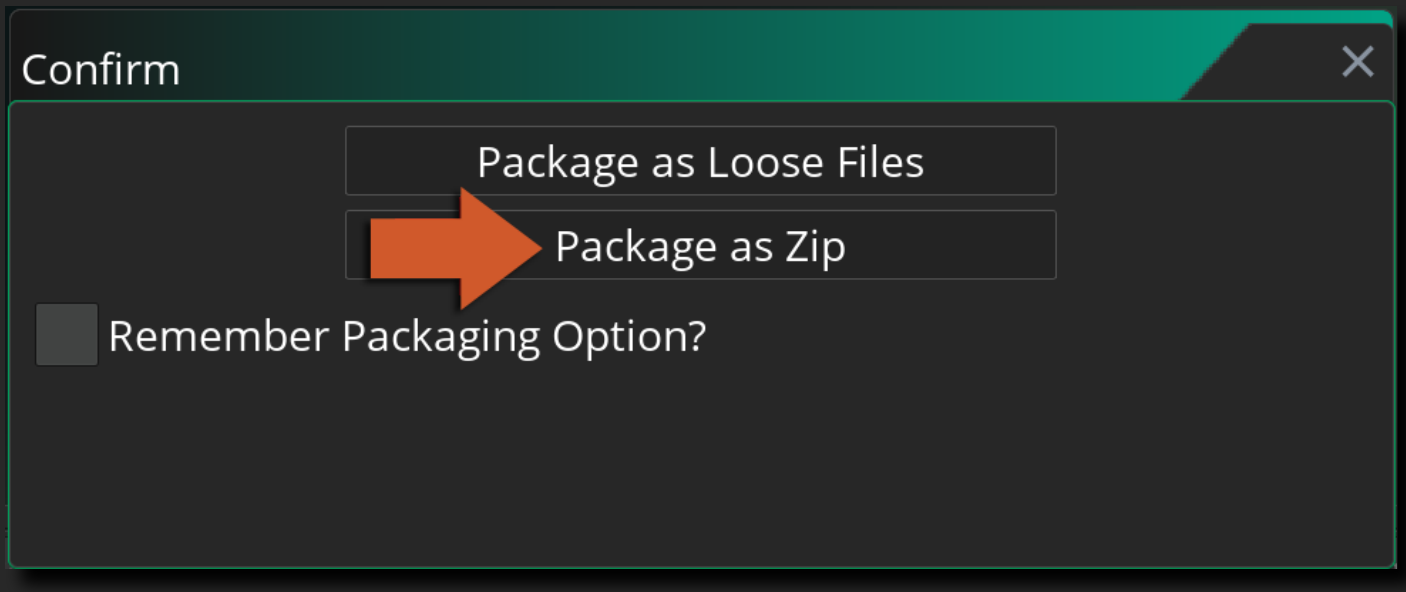
3. Enter your App nickname (required):



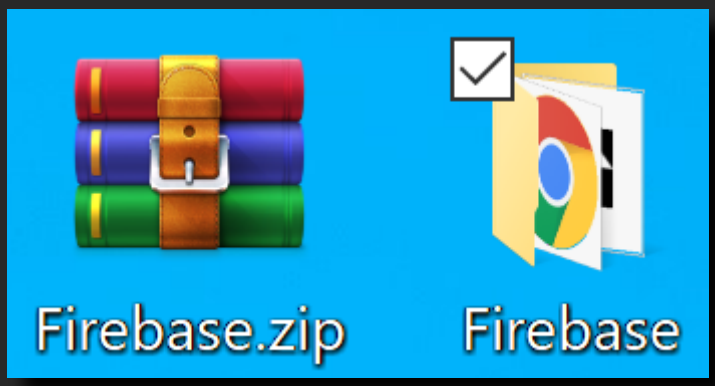
4. On this screen, just copy the **firebaseConfig** struct:

5. Now go back into GameMaker Studio 2 and **build** your project.

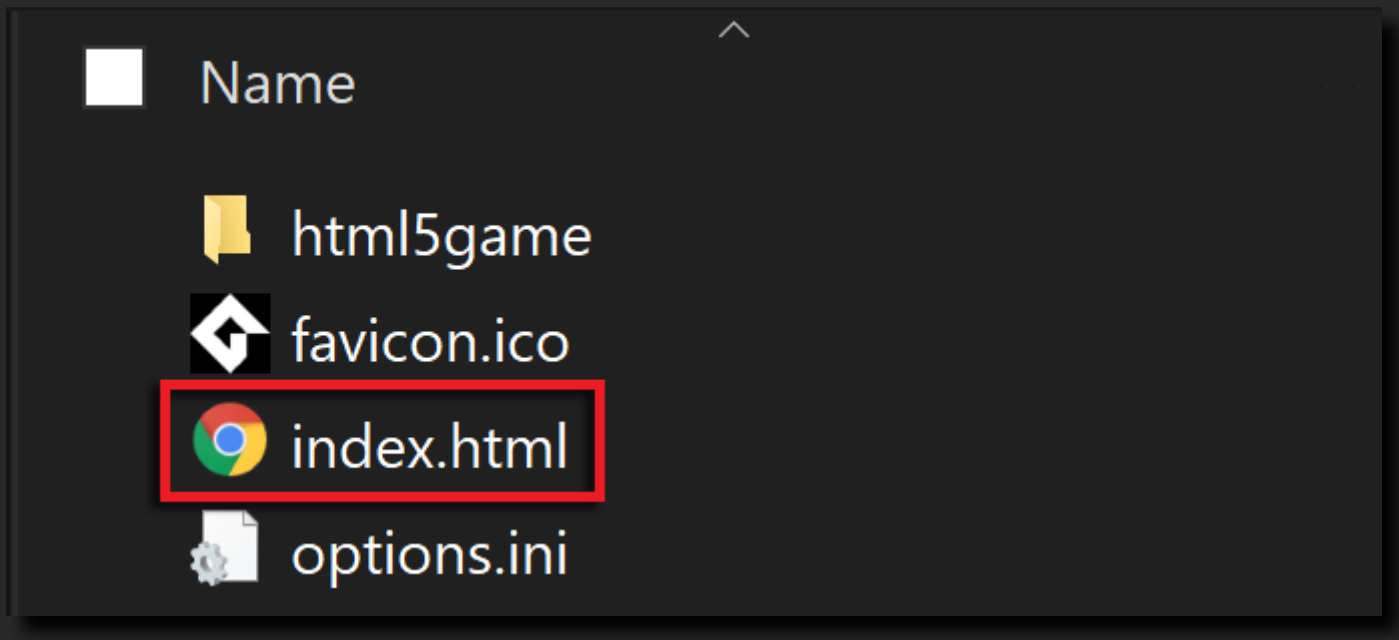
6. Choose the **Package As Zip** option:



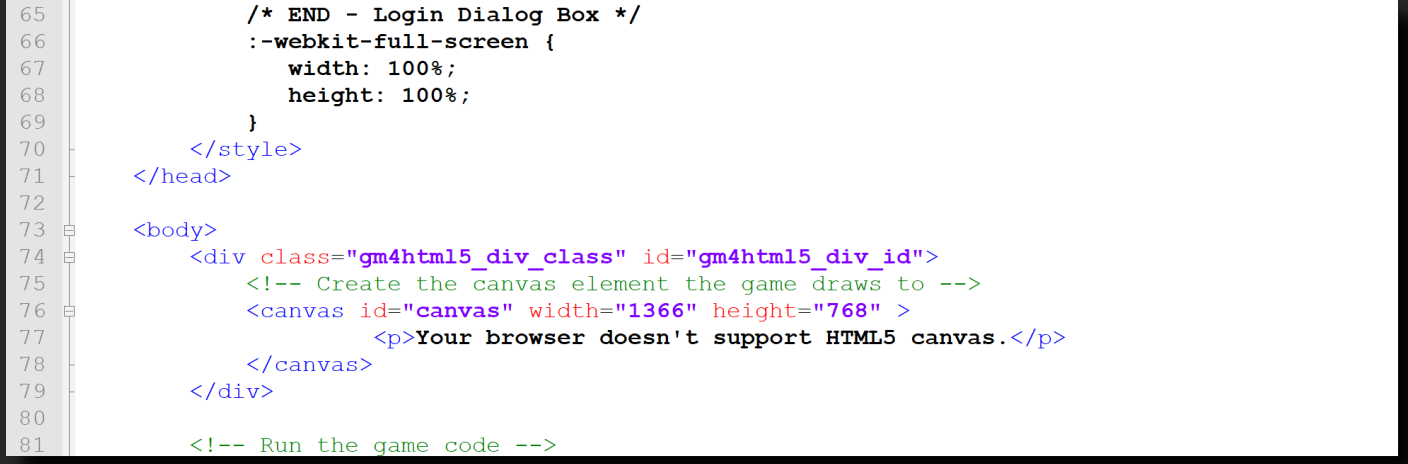
7. Locate the created package and **extract it** into a folder.



8. Open the extracted folder and look for an **index.html** file.



9. Open the **index.html** file in Notepad++ or Visual Studio Code (or any other text editor you prefer).



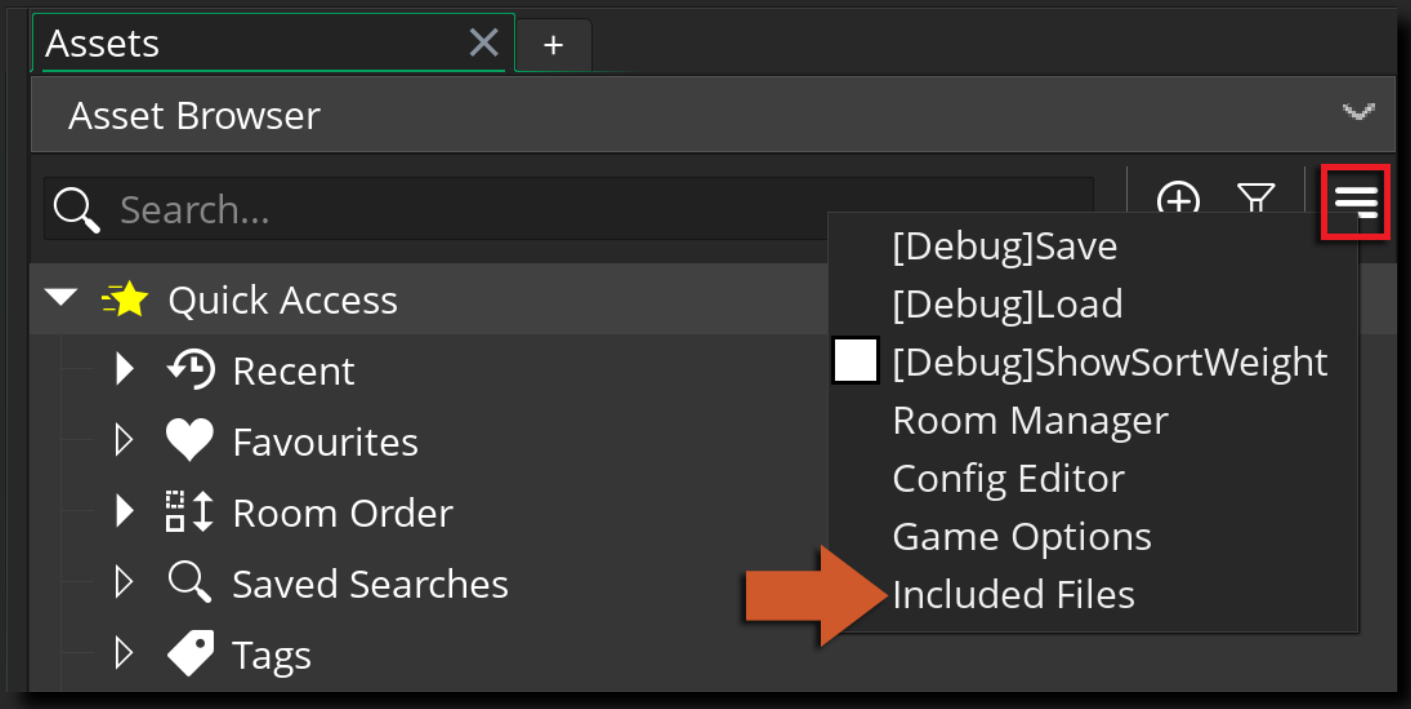
10. Now we need to add the following code between the `</head>` and `<body>` tags (line 72 in the **html.index** image above):

```
<script src="https://www.gstatic.com/firebasejs/8.9.1/firebase-
app.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.9.1/firebase-
analytics.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.9.1/firebase-
auth.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.9.1/firebase-
database.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.9.1/firebase-
firestore.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.9.1/firebase-remote-
config.js"></script>

<script>
  const firebaseConfig = {
    apiKey: "",
    authDomain: "",
    databaseURL: "",
    projectId: "",
    storageBucket: "",
    messagingSenderId: "",
    appId: "",
    measurementId: ""
  };
  firebase.initializeApp(firebaseConfig);

</script>
```

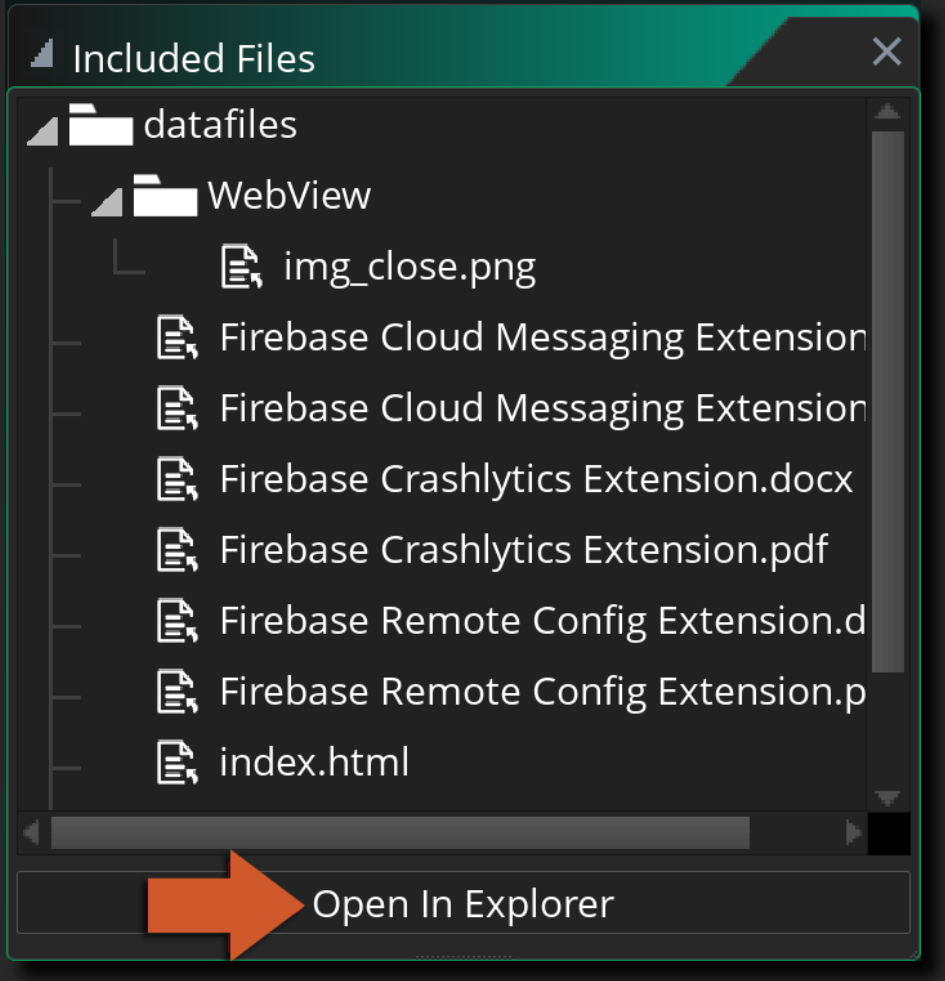
11. Replace the **const firebaseConfig** part with the one copied in step 4:

```
                height: 100%;
            }
        </style>
    </head>

<!-- Firebase -->
<script src="https://www.gstatic.com/firebasejs/8.9.1/firebase-app.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.9.1/firebase-analytics.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.9.1/firebase-auth.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.9.1/firebase-database.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.9.1/firebase-firestore.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.9.1/firebase-remote-config.js"></script>


<script>

  // Your web app's Firebase configuration
  // For Firebase JS SDK v7.20.0 and later, measurementId is optional
  const firebaseConfig = {
    apiKey: "[redacted]",
    authDomain: "[redacted]",
    databaseURL: "[redacted]",
    projectId: "[redacted]",
    storageBucket: "[redacted]",
    messagingSenderId: "[redacted]",
    appId: "[redacted]",
    measurementId: "[redacted]"
  };

  firebase.initializeApp(firebaseConfig);

</script>



    <body>
        <div class="gm4html5_div_class" id="gm4html5_div_id">
```

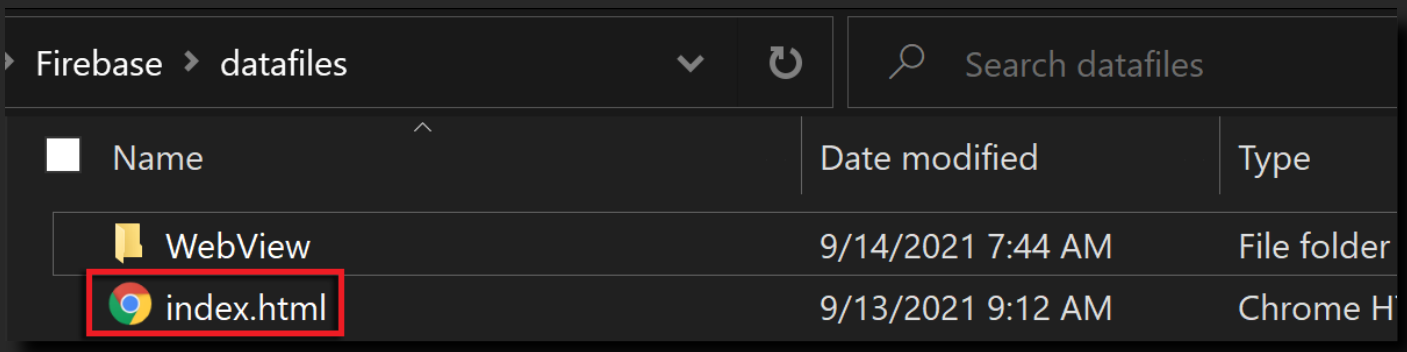
12. Go back into GameMaker and open your **Included Files** folder.



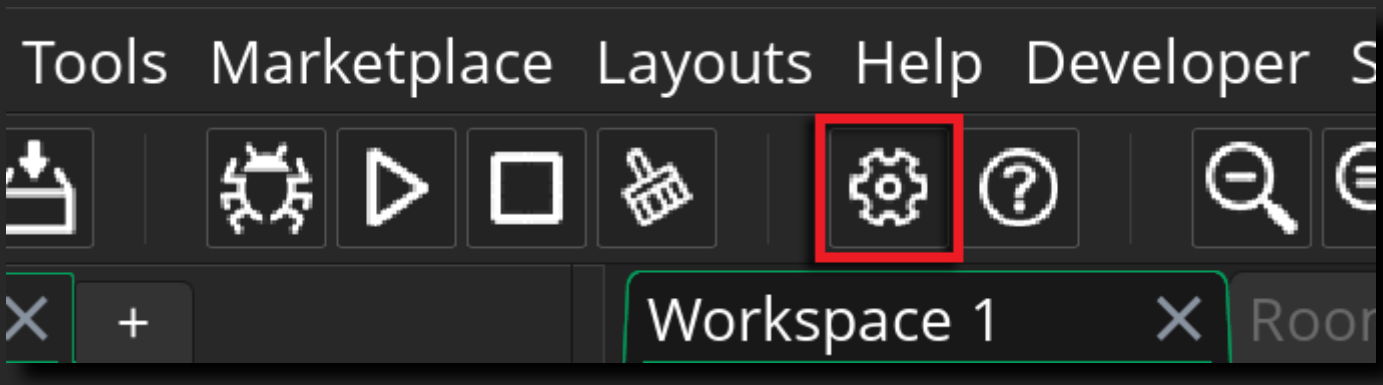
13. Press the **Open in Explorer** button:



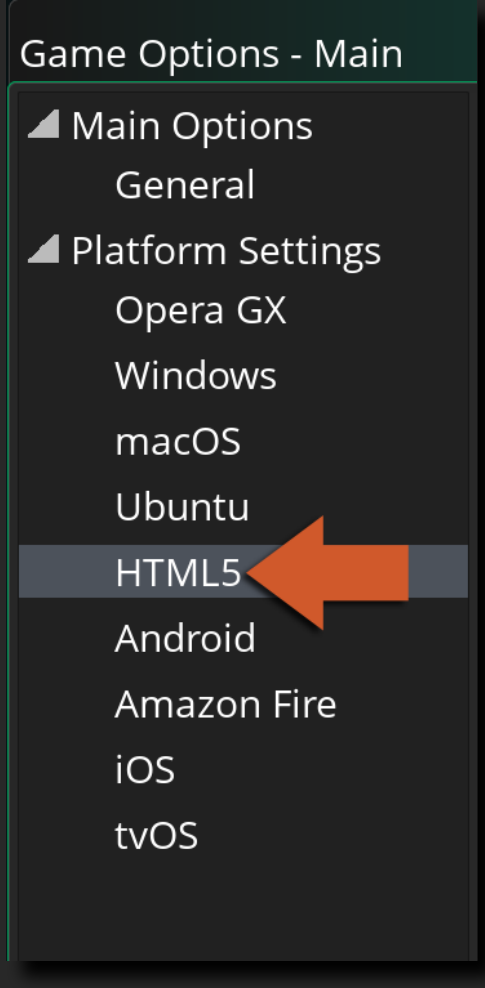
14. Place your **index.html** file inside the folder that opens (/datafiles).



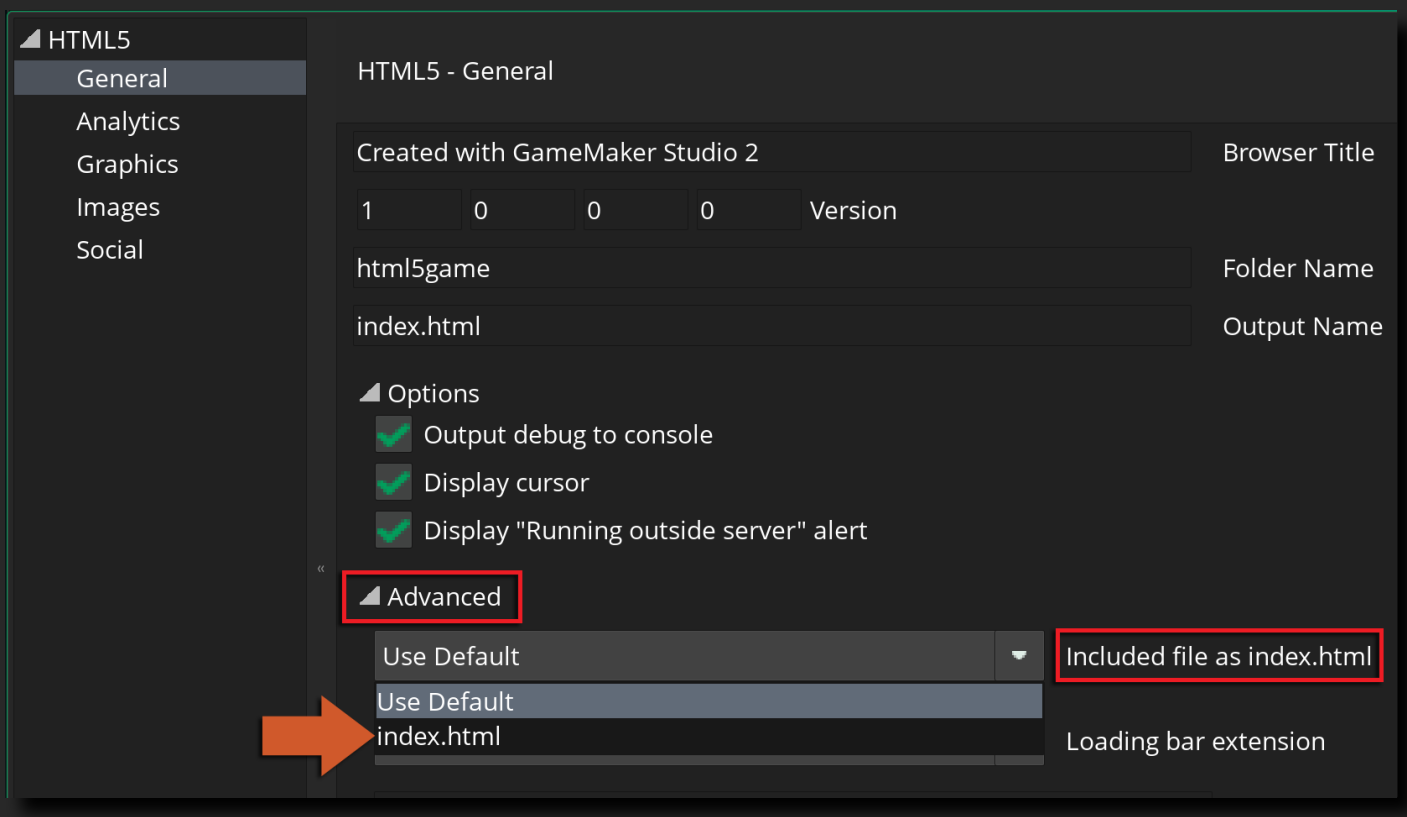
15. Back in GameMaker, click on the **Game Options button.**



16. Go into the **HTML5** platform settings



17. In the **Advanced** section go to the "Include file as index.html" dropdown and select the **index.html** option (this is the file we have just added to the included files).



18. Press **Apply** and the main setup for all Firebase Web modules is finished!

FirebaseStorage_Cancel

This function is used to cancel an upload (see FirebaseStorage_Upload) or download (see FirebaseStorage_Download) process.

Syntax:

```
FirebaseStorage_Cancel(listener)
```

Argument	Description
listener	The upload/download asynchronous listener

Returns:

```
N/A
```

Example:

```
listener = FirebaseStorage_Upload(localfile, path);

// After some time
FirebaseStorage_Cancel(listener);
```

The code above will start a file upload to the given `path` (using the FirebaseStorage_Upload function) and after a while cancel it.

FirebaseStorage_Delete

This function deletes a path in the Firebase Cloud Storage and returns a listener identifier that can be used for tracking progress on the request.

This is an asynchronous function that will trigger the **Async Social** event during progress and when finished.

Syntax:

```
FirebaseStorage_Delete(firebasePath, [bucket])
```

Argument	Type	Description
firebasePath	string	The remote path on the Firebase Cloud Storage server
bucket	string	Other Firebase Storage bucke OPTIONAL

Returns:

```
Real (Asynchronous Listener ID)
```

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseStorage_Delete"`
listener	real	The asynchronous listener ID.
path	string	The remote path.
success	bool	Whether or not the download succeeded (if finished OPTIONAL

Example:

```
var uid = FirebaseAuthentication_GetUID();
listenerId = FirebaseStorage_Delete("UserProfiles/" + uid + "/img.png");
```

The code above uses the user identifier (provided by the function `FirebaseAuthentication_GetUID()` from the Firebase Authentication extension) to create a remote path and then deletes a file in that path. The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "type"] == "FirebaseStorage_Delete")
{
    if (async_load[? "success"])
    {
        show_debug_message("File was deleted");
    }
    else
    {
        show_debug_message("File could not be deleted");
    }
}
```

The code above matches the response against the correct event **type** and logs whether the task was completed successfully or not.

FirebaseStorage_Download

This function is used to download files from a remote path and returns a listener identifier that can be used for tracking progress on the request.

This is an asynchronous function that will trigger the **Async Social** event during progress and when finished.

IMPORTANT This function is not available on the Web target.

Syntax:

```
FirebaseStorage_Download(localPath, firebasePath, [bucket])
```

Argument	Type	Description
localPath	string	The local path where the files will be downloaded
firebasePath	string	The remote path on the Firebase Cloud Storage server
bucket	string	Other Firebase Storage bucke OPTIONAL

Returns:

```
Real (Asynchronous Listener ID)
```

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseStorage_Download"`
listener	real	The asynchronous listener ID.
path	string	The remote file path.
localPath	string	The local file path.

success	bool	Whether or not the download succeeded (if finished OPTIONAL
transferred	real	Number of transferred bytes (if NOT finished OPTIONAL
total	real	Total number of bytes (if NOT finished OPTIONAL

Example:

```
var uid = FirebaseAuthentication_GetUID()
listenerId = FirebaseStorage_Download("profilePic.png", "UserProfiles/" + uid + "/
img.png")
```

The code above uses the user identifier (provided by the function `FirebaseAuthentication_GetUID()` from the Firebase Authentication extension) to create a remote path string, then downloads a file from the server to a local path. The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "type"] == "FirebaseStorage_Download")
{
    if (ds_map_exists(async_load[? "success"]))
    {
        if (async_load[? "success"])
        {
            global.percent = 100;
            img = sprite_add(async_load[?"localPath"], 0, 0, 0, 0, 0);
            show_message_async("Download SUCCESS")
        }
        else
        {
            global.percent = 0;
            show_message_async("Download FAILED")
        }
    }
    else
    {
        global.percent = 100 * async_load[? "transferred"] / async_load[? "total"];
    }
}
```

The code above matches the response against the correct event **type** and if the download has not finished yet (`"success"` key is not present) then calculates the download percentage and stores it inside a global variable (`global.percent`). When the download finishes it checks if it was a successful download and if so it adds the downloaded file as a sprite (using the **sprite_add** function).

NOTE Always remember to free sprites added this way when you no longer need them, otherwise it can lead to memory leaks.

FirebaseStorage_GetURL

This function asynchronously retrieves a long lived download URL with a revocable token. This can be used to share the file with others, but can be revoked by the developer in the Firebase Console if desired.

This is an asynchronous function that will trigger the **Async Social** event when the task is finished.

Syntax:

```
FirebaseStorage_GetURL(firebasePath, [bucket])
```

Argument	Type	Description
firebasePath	string	The remote path on Firebase Cloud Storage server.
bucket	string	Other Firebase Storage bucke OPTIONAL

Returns:

```
Real (Asynchronous Listener ID)
```

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The constant `"FirebaseStorage_GetURL"`
listener	real	The asynchronous listener ID.
path	string	The remote file path.
success	bool	Whether or not the task succeeded
value	string	The URL of the file (if task succeeded OPTIONAL

Example:

```
listenerId = FirebaseStorage_GetURL(path);
```

The code above tries to retrieve an URL from a remote path on the Firebase Cloud Storage server. The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "type"] == "FirebaseStorage_GetURL")
{
    if (async_load[? "success"])
    {
        show_debug_message(async_load[? "path"] + " URL: " + async_load[? "value"]);
    }
    else
    {
        show_debug_message("There was an error retrieving the URL");
    }
}
```

The code above matches the response against the correct event **type** and if the task was successful it logs the retrieved URL for the specific path.

FirebaseStorage_List

This function queries the contents of a Firebase Cloud Storage path, if `pageToken` is provided the query will start after the page number provided. It also returns a listener identifier that can be used for tracking progress on the request.

This is an asynchronous function that will trigger the **Async Social** event when the task is finished.

Syntax:

```
FirebaseStorage_List(path, maxResults, [pageToken, bucket])
```

Argument	Description
path	The remote Firebase path to query
maxResults	The max number of results
pageToken	Token of the last reques OPTIONAL
bucket	Other Firebase Storage bucke OPTIONAL

Returns:

```
Real (Asynchronous Listener ID)
```

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseStorage_List"`
listener	real	The asynchronous listener ID.
path	string	The remote path.
success	bool	Whether or not the task succeeded
files	string	A JSON formatted string of an array of file paths OPTIONAL

folders	string	A JSON formatted string of an array of folder paths OPTIONAL
pageToken	real	The current query page.

Example:

```
listenerId = FirebaseStorage_List("/testing", 5);
```

The code above queries for files and folders inside the path `"/testing"`, returning a maximum of `5` results. The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "type"] == "FirebaseStorage_List")
{
    if (async_load[? "success"])
    {
        global.arrayOfFiles = json_parse(async_load[? "files"]);
        global.arrayOfFolders = json_parse(async_load[? "folders"]);
    }
}
```

The code above matches the response against the correct event **type** and if the task was successful it stores the files and folders paths parsed as arrays (using the **json_parse** function) inside global variables (`global.arrayOfFiles` and `global.arrayOfFolders`).

FirebaseStorage_ListAll

This function queries all the contents of a Firebase Cloud Storage path and returns a listener identifier that can be used for tracking progress on the request.

This is an asynchronous function that will trigger the **Async Social** event when the task is finished.

Syntax:

```
FirebaseStorage_ListAll(path, [bucket])
```

Argument	Description
path	The remote Firebase path to query
bucket	Other Firebase Storage bucke OPTIONAL

Returns:

```
Real (Asynchronous Listener ID)
```

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseStorage_ListAll"`
listener	real	The asynchronous listener ID.
path	string	The remote path.
success	bool	Whether or not the task succeeded
files	string	A JSON formatted string of an array of file paths OPTIONAL
folders	string	A JSON formatted string of an array of folder paths OPTIONAL

Example:

```
listenerId = FirebaseStorage_ListAll("/testing");
```

The code above queries for all files and folders inside the path `"/testing"`. The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "type"] == "FirebaseStorage_ListAll")
{
    if (async_load[? "success"])
    {
        global.arrayOfFiles = json_parse(async_load[? "files"]);
        global.arrayOfFolders = json_parse(async_load[? "folders"]);
    }
}
```

The code above matches the response against the correct event **type** and if the task was successful it stores the files and folders paths parsed as arrays (using the **json_parse** function) inside global variables (`global.arrayOfFiles` and `global.arrayOfFolders`).

FirebaseStorage_Download

This function uploads files to a given path and returns a listener identifier that can be used for tracking progress on the request.

This is an asynchronous function that will trigger the **Async Social** event during progress and when finished.

IMPORTANT This function is not available on the Web target.

Syntax:

```
FirebaseStorage_Upload(localPath, firebasePath, [bucket])
```

Argument	Type	Description
localPath	string	The local path to a file
firebasePath	string	The remote path on the Firebase Cloud Storage server
bucket	string	Other Firebase Storage bucket OPTIONAL

Returns:

```
Real (Asynchronous Listener ID)
```

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseStorage_Upload"`
listener	real	The asynchronous listener ID.
path	string	The remote file path.
localPath	string	The local file path.

success	bool	Whether or not the upload succeeded (if finished OPTIONAL
transferred	real	Number of transferred bytes (if NOT finished OPTIONAL
total	real	Total number of bytes (if NOT finished OPTIONAL

Example:

```
var uid = FirebaseAuthentication_GetUID()
listenerId = FirebaseStorage_Upload("profilePic.png", "UserProfiles/" + uid + "/
img.png")
```

The code above uses the user identifier (provided by the function `FirebaseAuthentication_GetUID()` from the Firebase Authentication extension) to create a remote path string, and then uploads a file to the remote server. The function call will return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "type"] == "FirebaseStorage_Upload")
{
    if (!ds_map_exists(async_load[? "success"]))
    {
        global.percent = 100 * async_load[? "transferred"] / async_load[? "total"];
    }
    else
    {
        show_debug_message("The upload terminated");
    }
}
```

The code above matches the response against the correct event **type** and if the upload has not finished yet (`"success"` key is not present) then calculates the upload percentage and stores it inside a global variable (`global.percent`).