Firebase Remote Config

Firebase Remote Config is a cloud service that lets you change the behavior and appearance of your app without requiring users to download an app update. When using Remote Config, you create in-app default values that control the behavior and appearance of your app. Then, you can later use the Firebase console or the Remote Config backend APIs to override in-app default values for all app users or for segments of your user base. Your app controls when updates are applied, and it can frequently check for updates and apply them with a negligible impact on performance.

Check the official page for more information.

Setup

Before you can start using any Firebase module extensions you are required to follow some initial configuration. However if you've already done these for any of the other modules you can skip this configuration section and go straight to using the API functions.

- Create Project
- Platform Setup (**Remote Config** enabling steps are required)

Functions

The following functions are given for working with Firebase Remote Config extension:

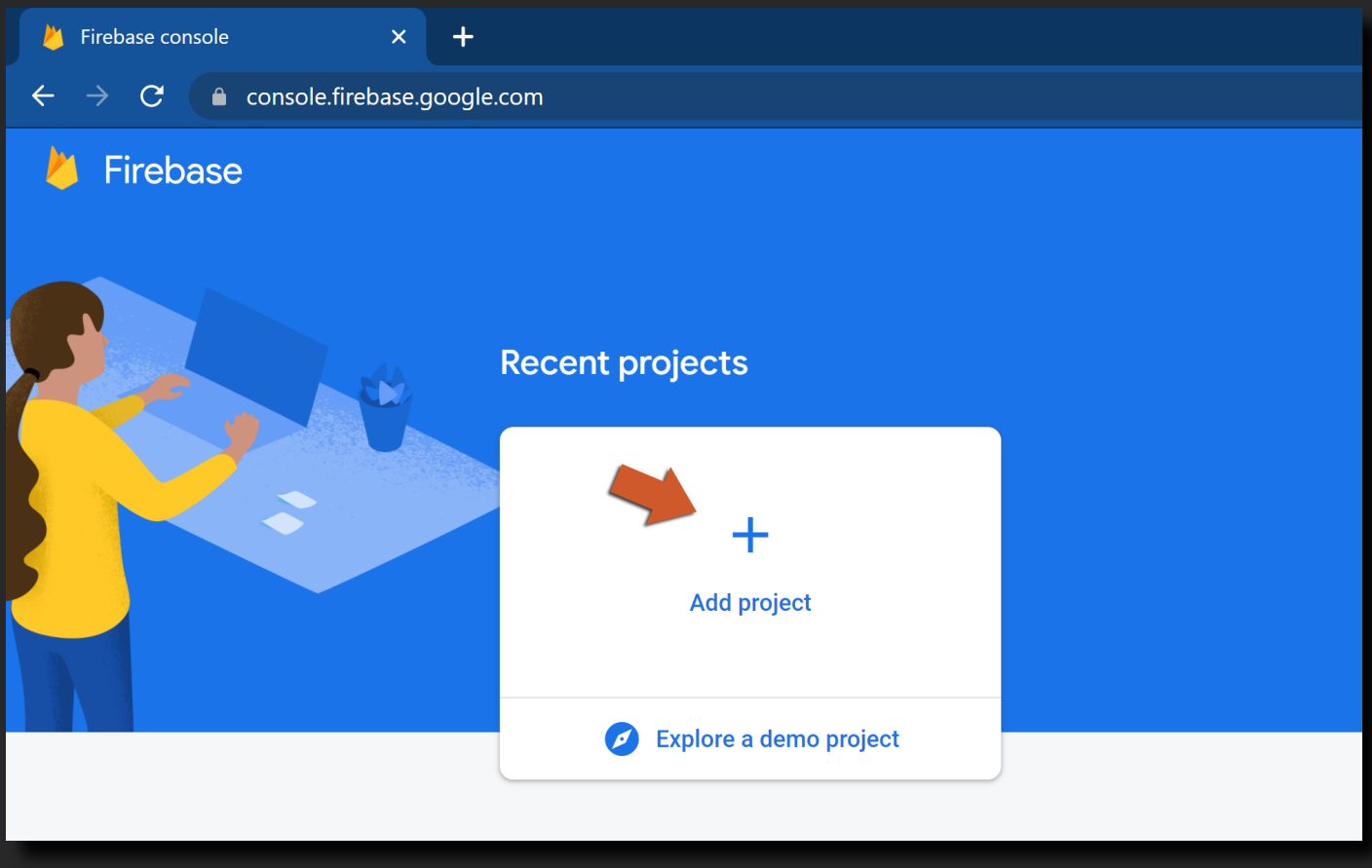
- FirebaseRemoteConfig_FetchAndActivate
- FirebaseRemoteConfig_GetDouble
- FirebaseRemoteConfig_GetKeys
- FirebaseRemoteConfig_GetString

- FirebaseRemoteConfig_Initialize
- FirebaseRemoteConfig_Reset
- FirebaseRemoteConfig_SetDefaultsAsync

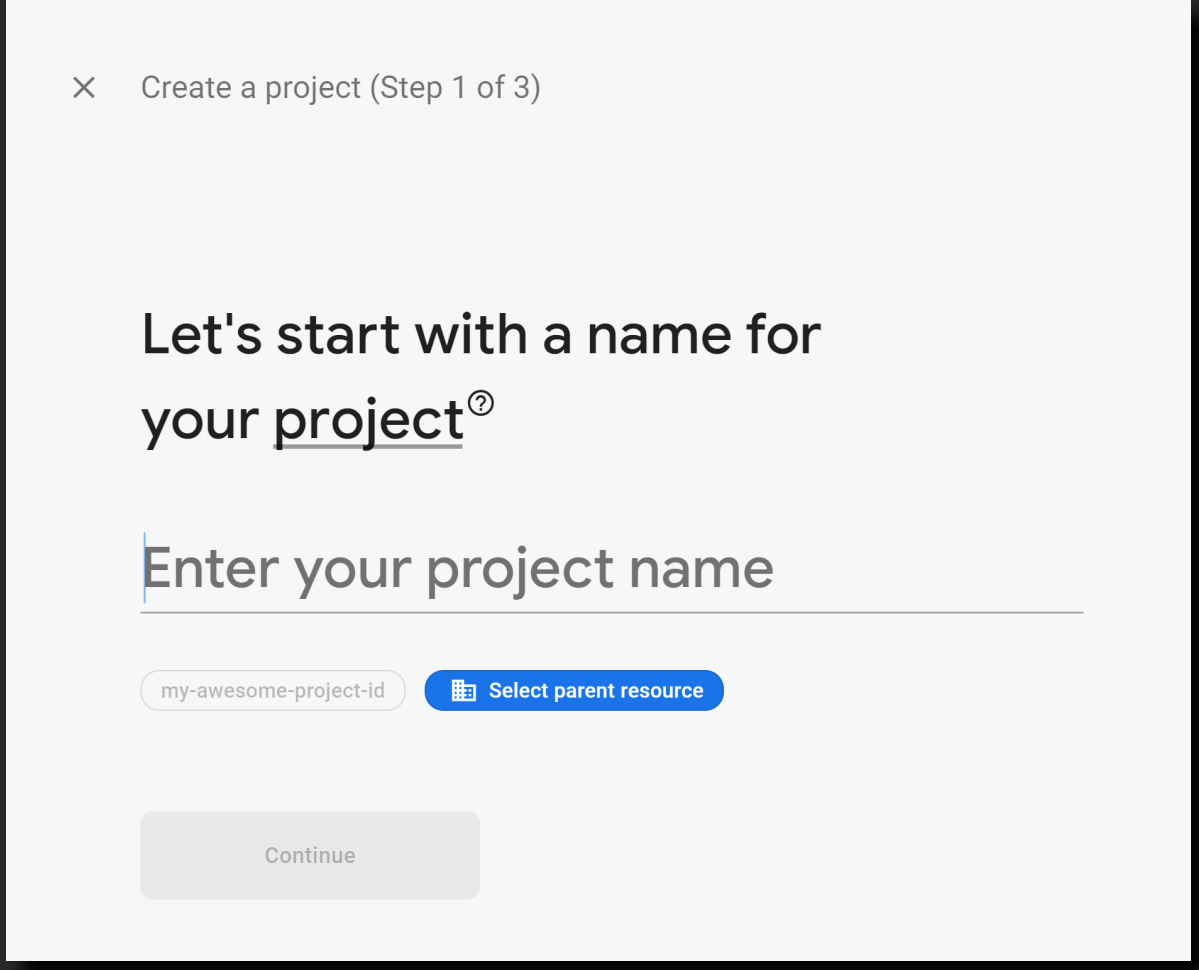
Create Project

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

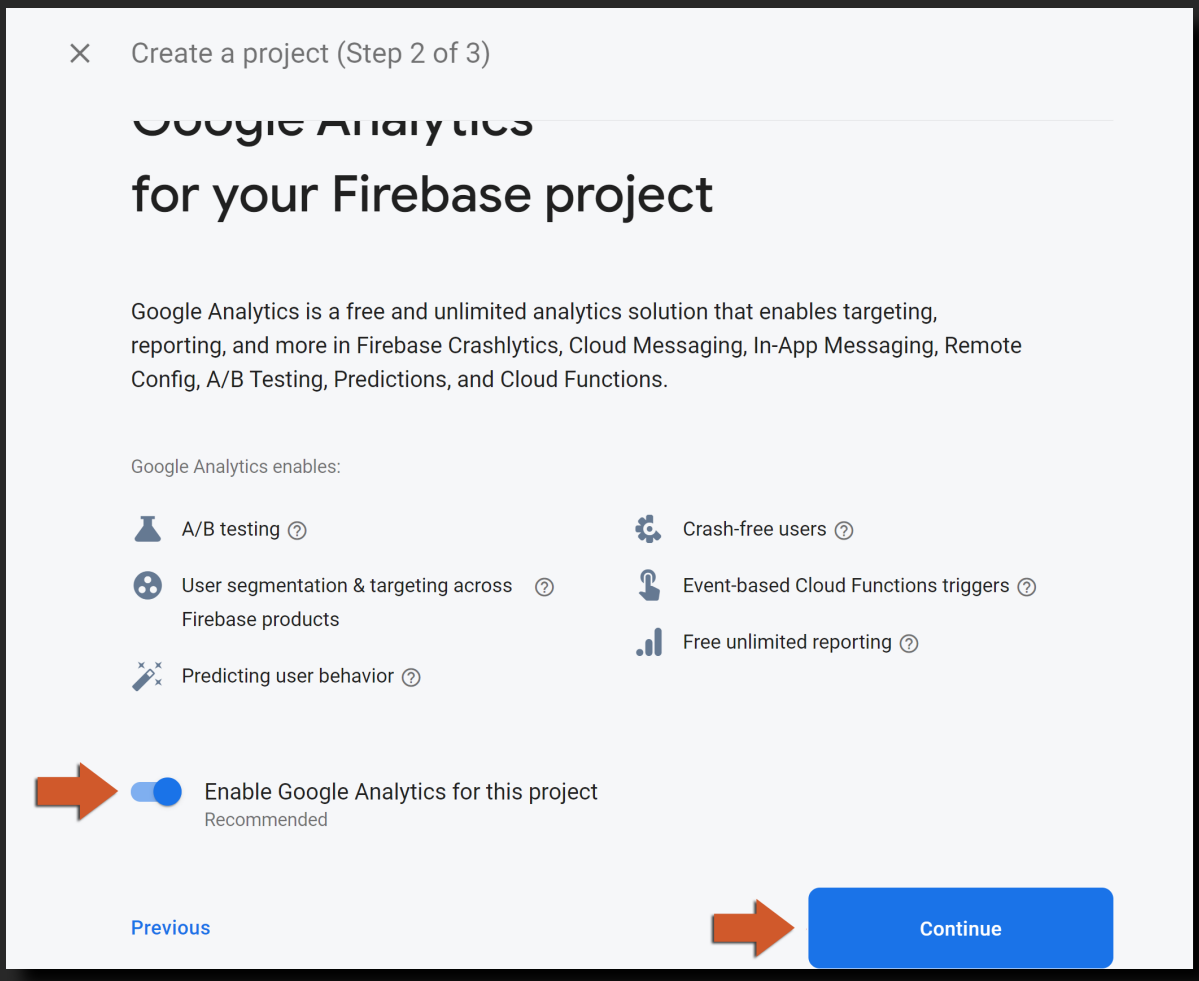
1. Go to the **Firebase Console** web site.
2. Click on **Add Project** to create your new project.



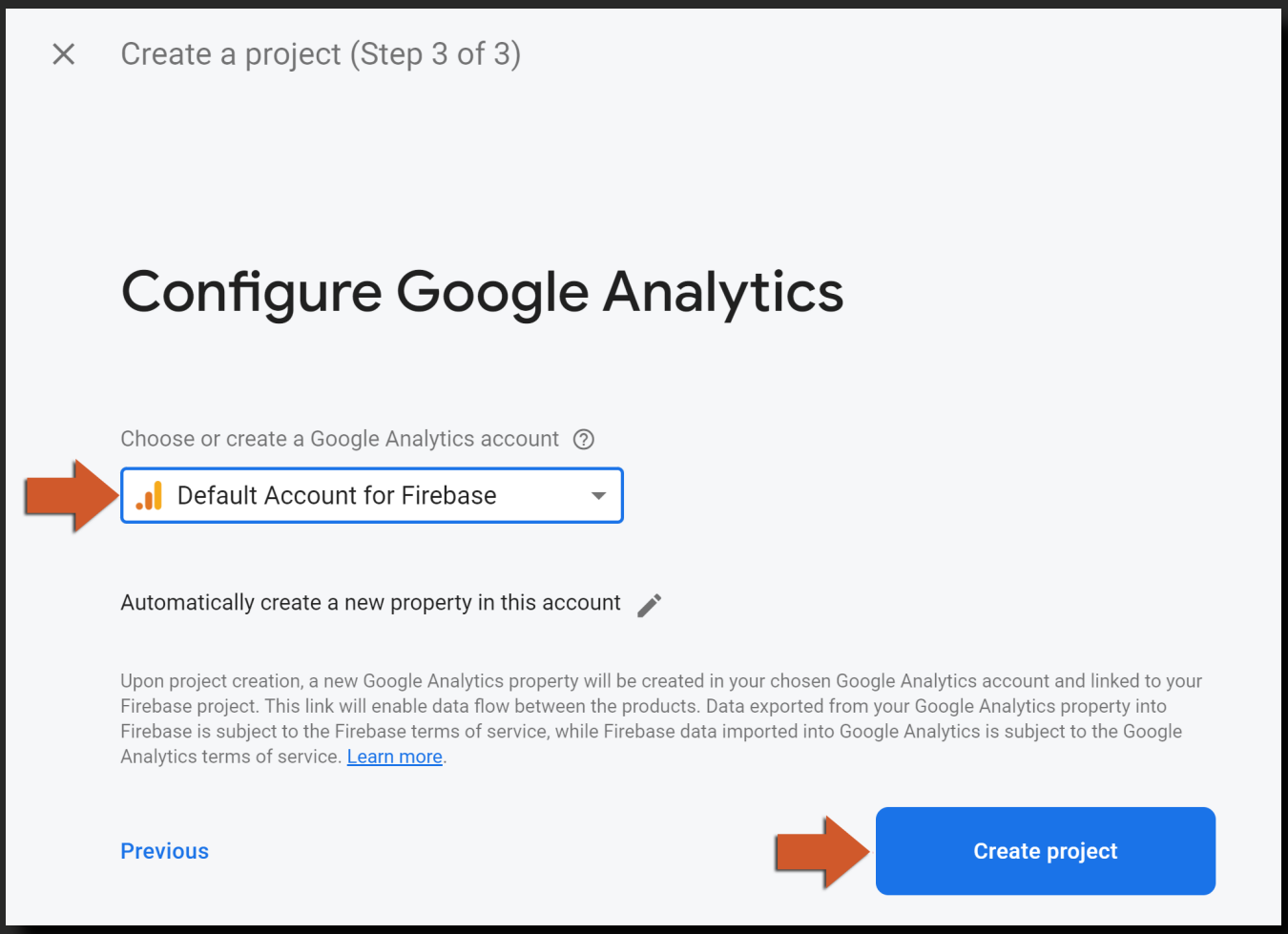
3. Enter a name for your project and click on the **Continue** button.



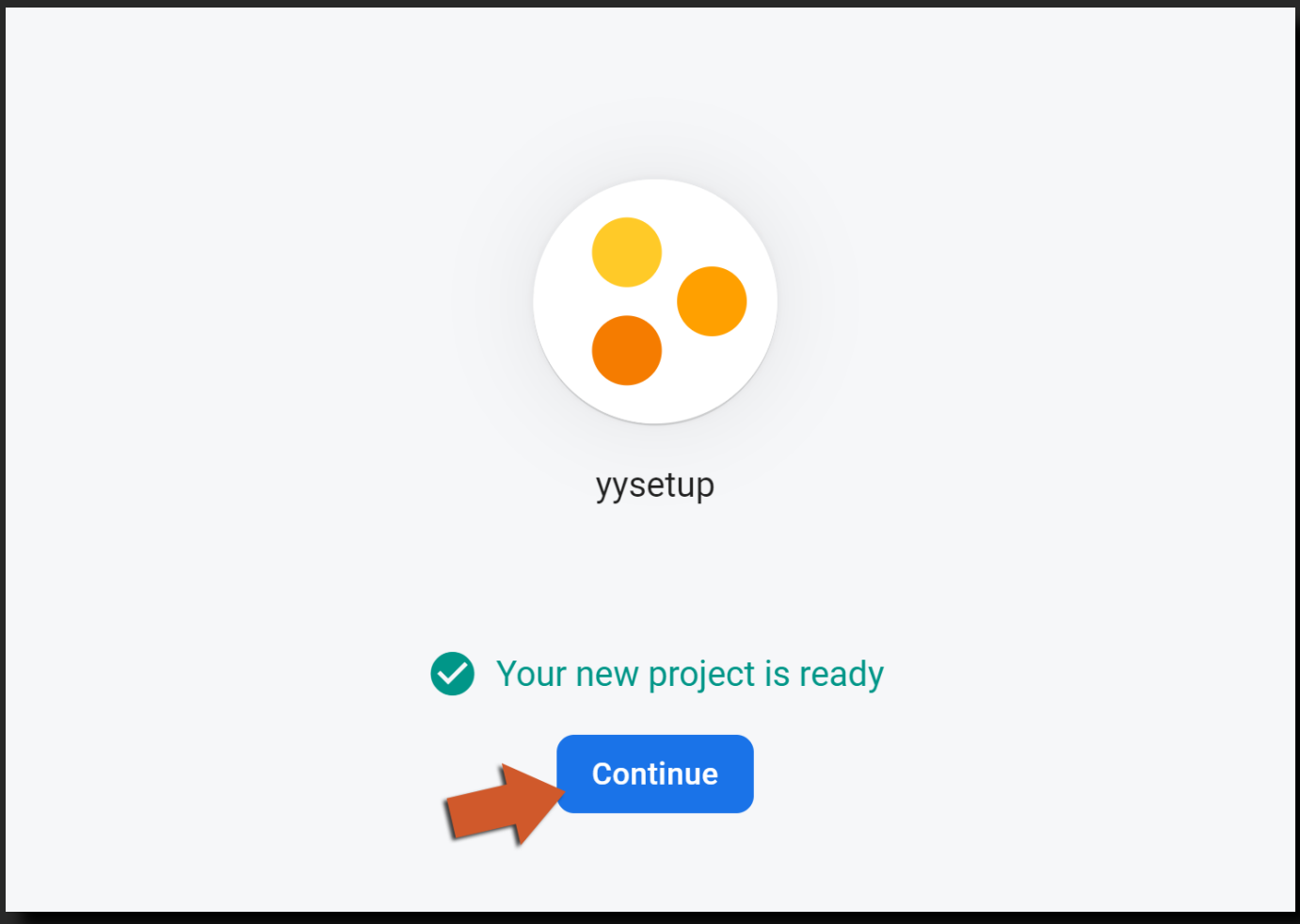
4. On the next page, make sure that **Enable Google Analytics for this project** is enabled and then click the **Continue** button:



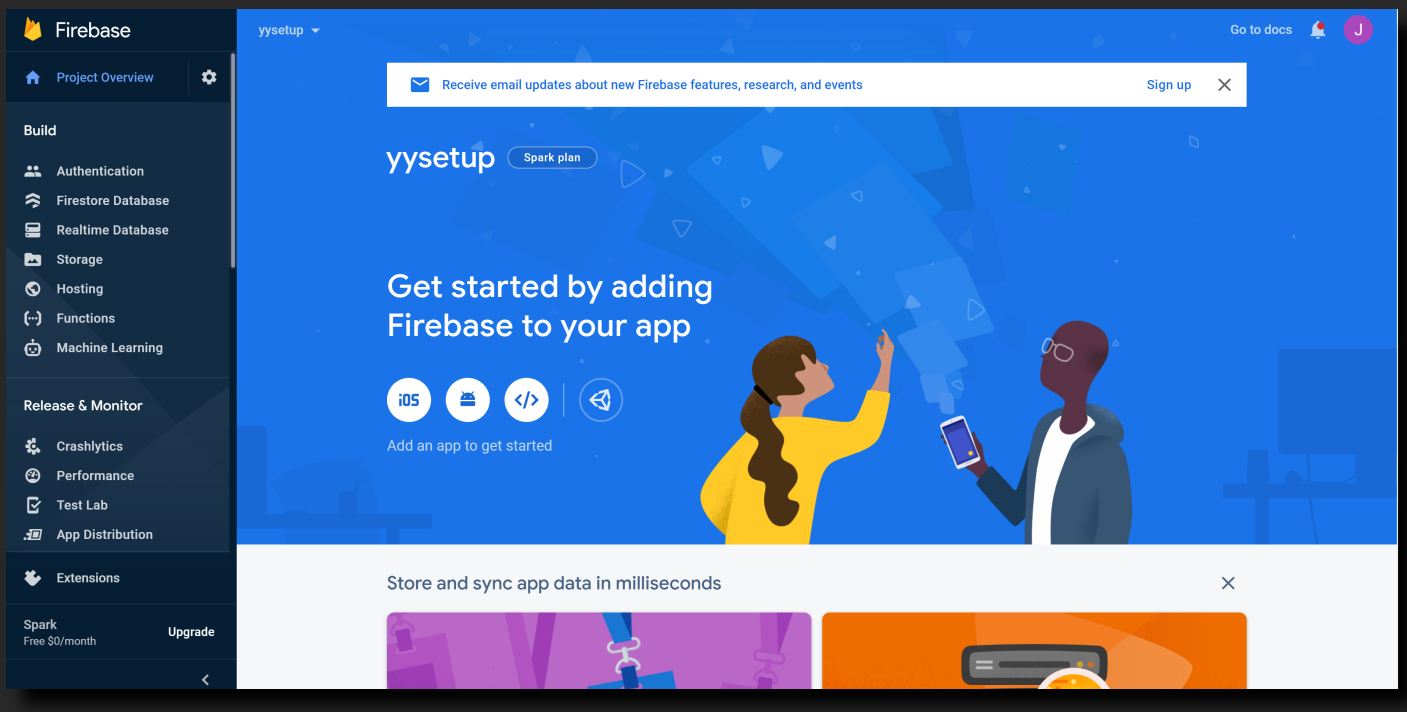
5. Select your account and click the **Create project** button:



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!

Platform Setup

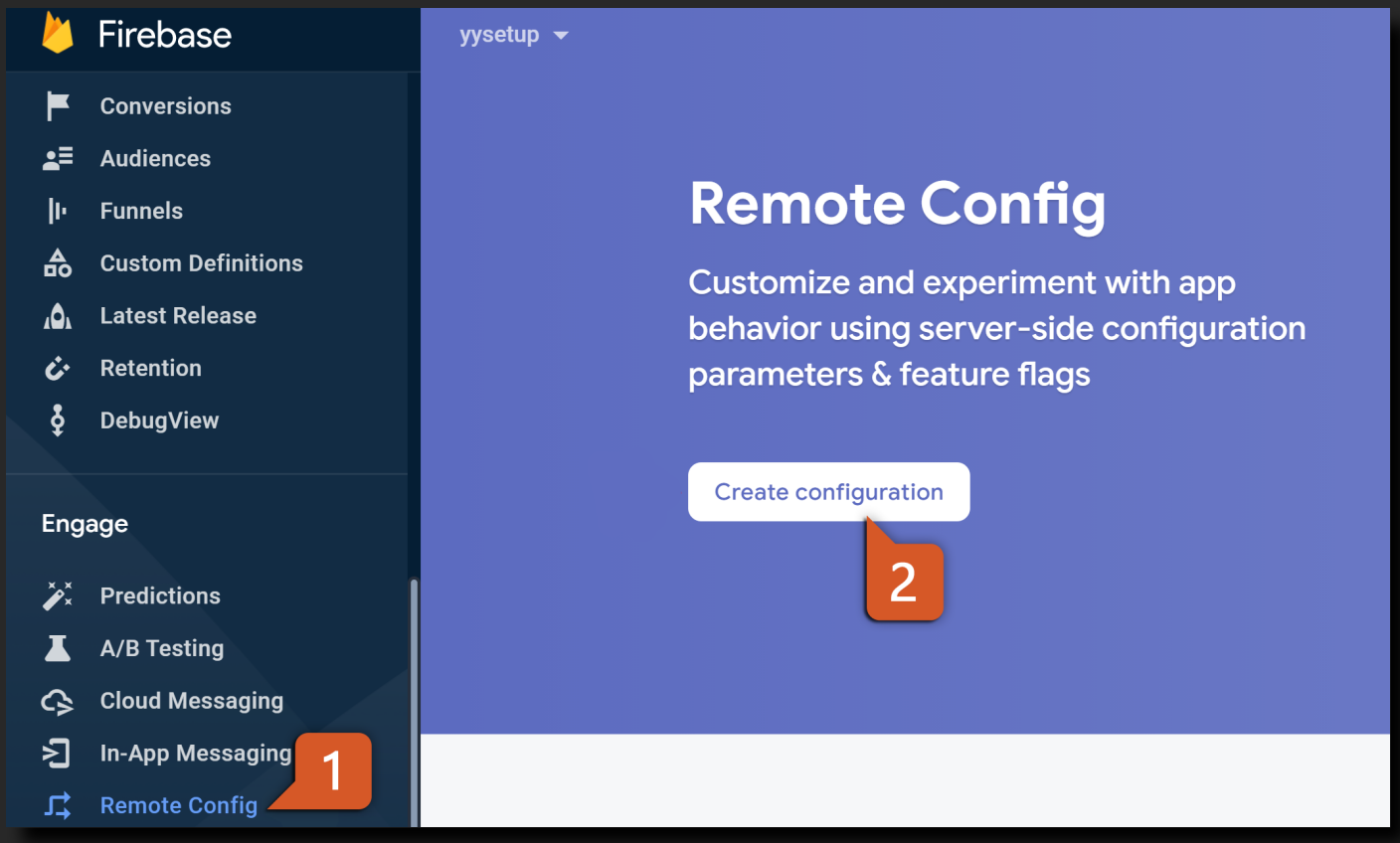
The Firebase Remote Config implementation uses SDK dependencies and therefore is only available on **Android**, **iOS** and **Web** targets. In this section we will cover the required setup necessary to start using the Firebase Remote Config extension in your game.

Select your target platform below and follow the simple steps to get your project up and running (you only need follow these topics once per project):

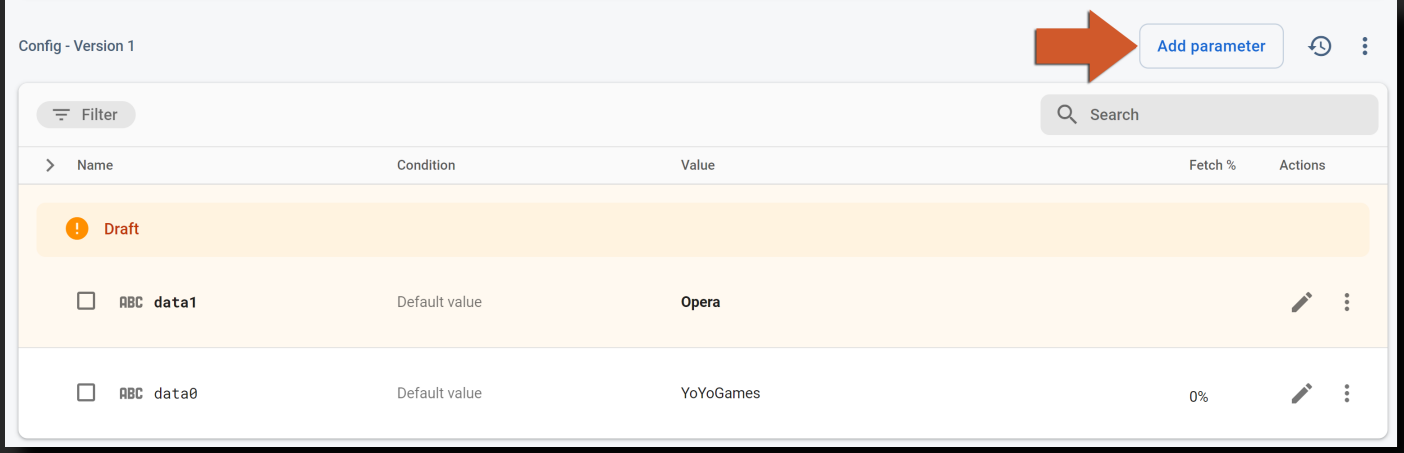
- Android Setup
- iOS Setup
- Web Setup

Enabling Remote Config

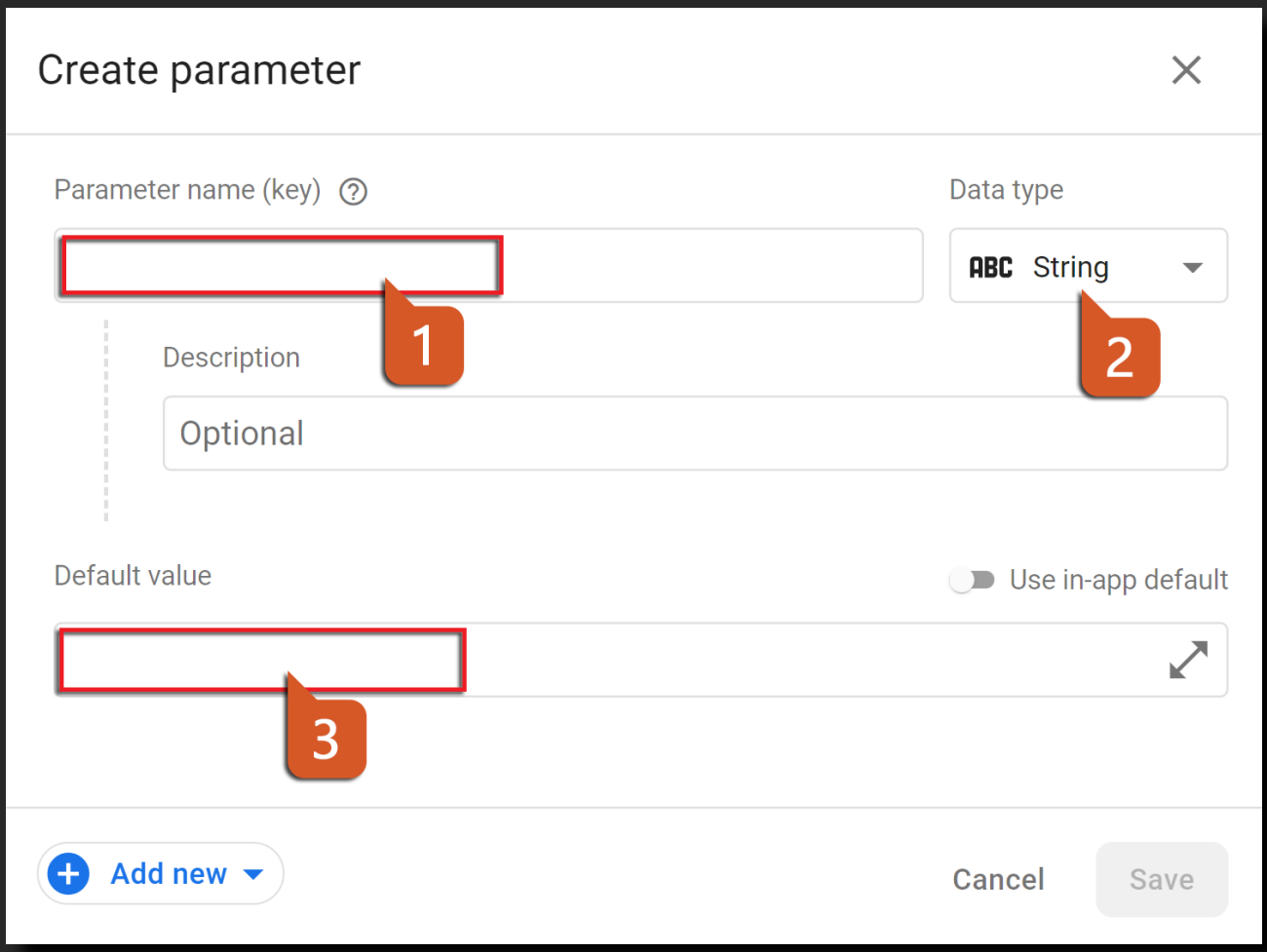
1. Go to **Remote Config** and click on the **Create configuration** button:



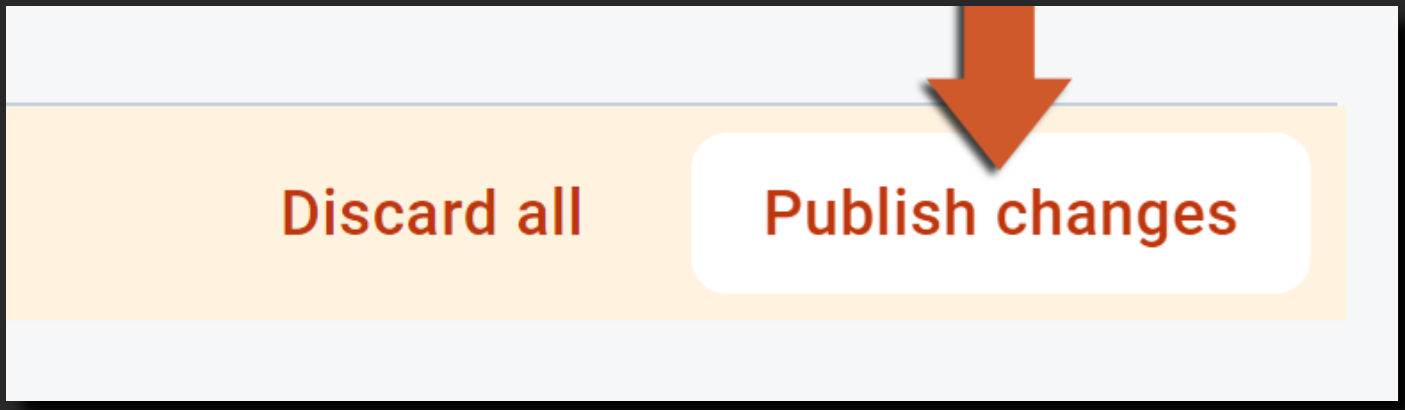
2. Click on the **Add parameter** button:



3. Add a **Parameter name (key)** and a **Default value (string/real),** then click **Save:**



4. After adding all your parameters, don't forget to click **Publish changes:**



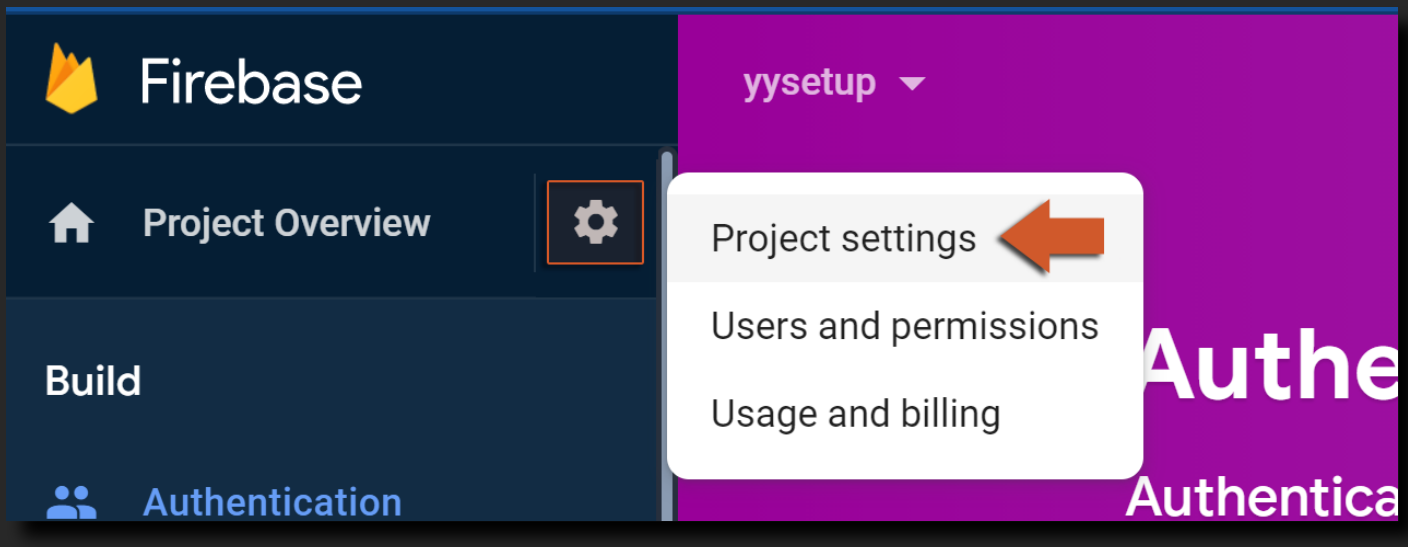
5. You are now all set to use the Remote Config extension!

Android Setup

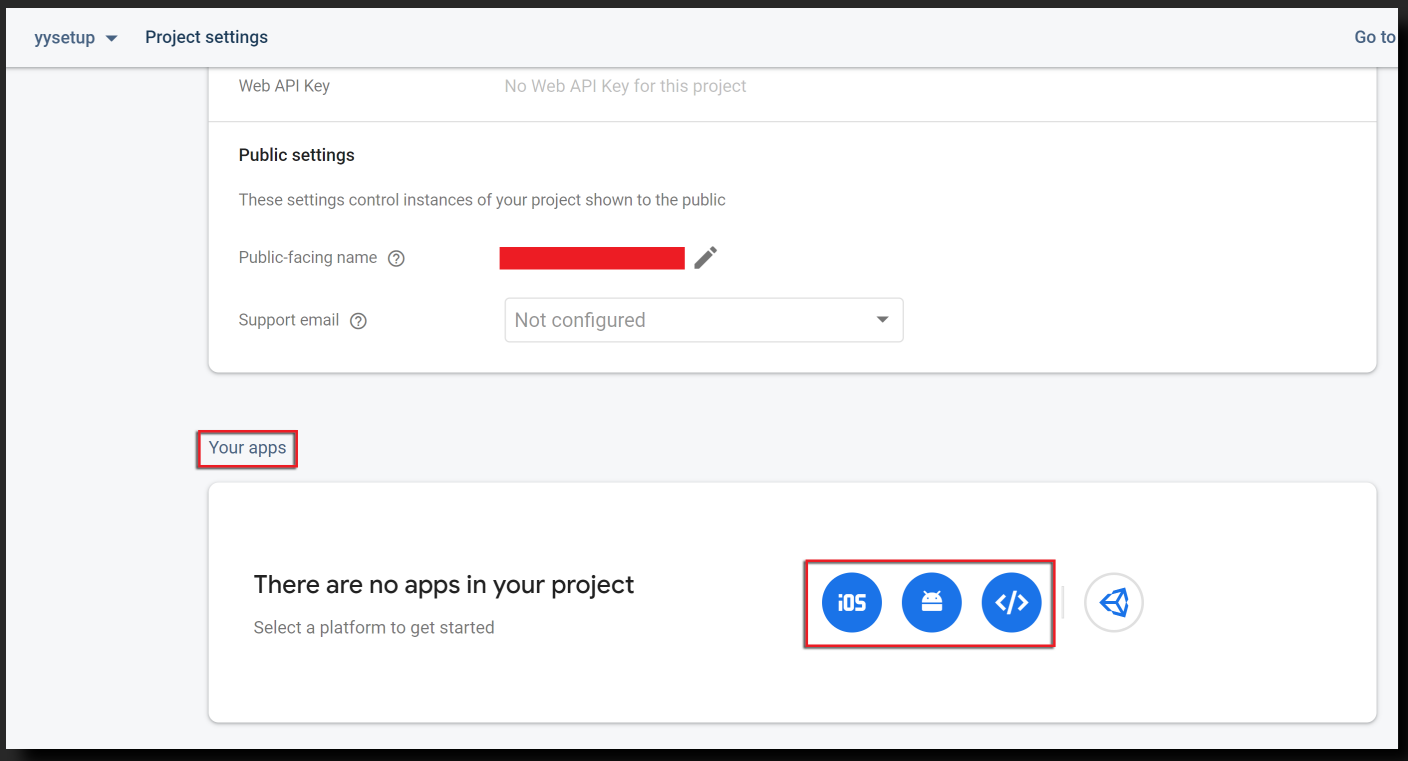
This setup is necessary for all the Firebase modules using Android and needs to be done once per project, and basically involves importing the **google-services.json** file into your project.

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

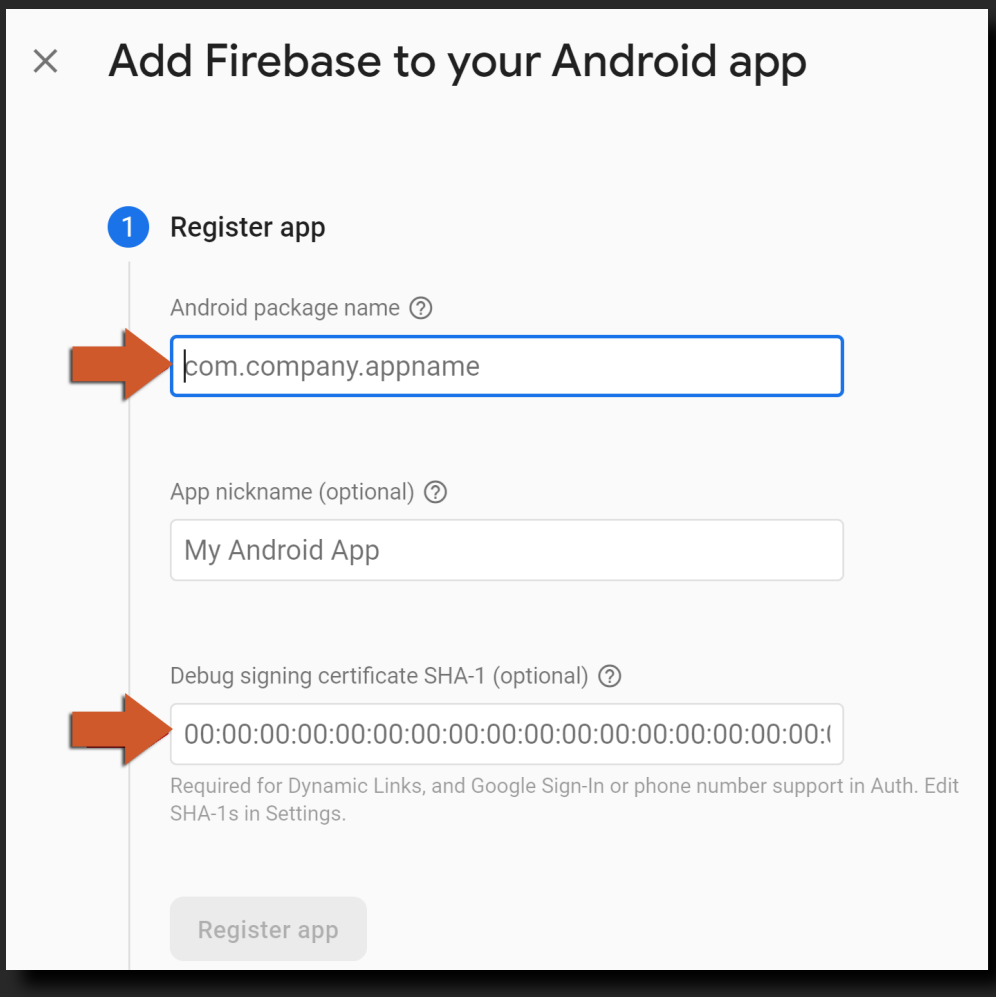
1. Click the **Settings** icon (next to Project Overview) and select **Project settings:**



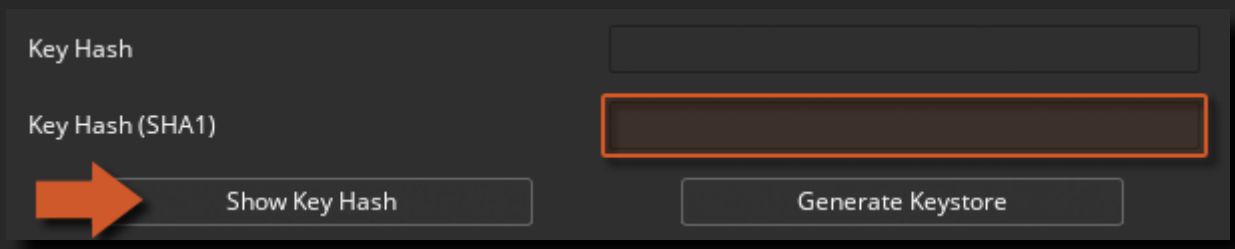
2. Now go to the **Your apps** section and click on the **Android** button:



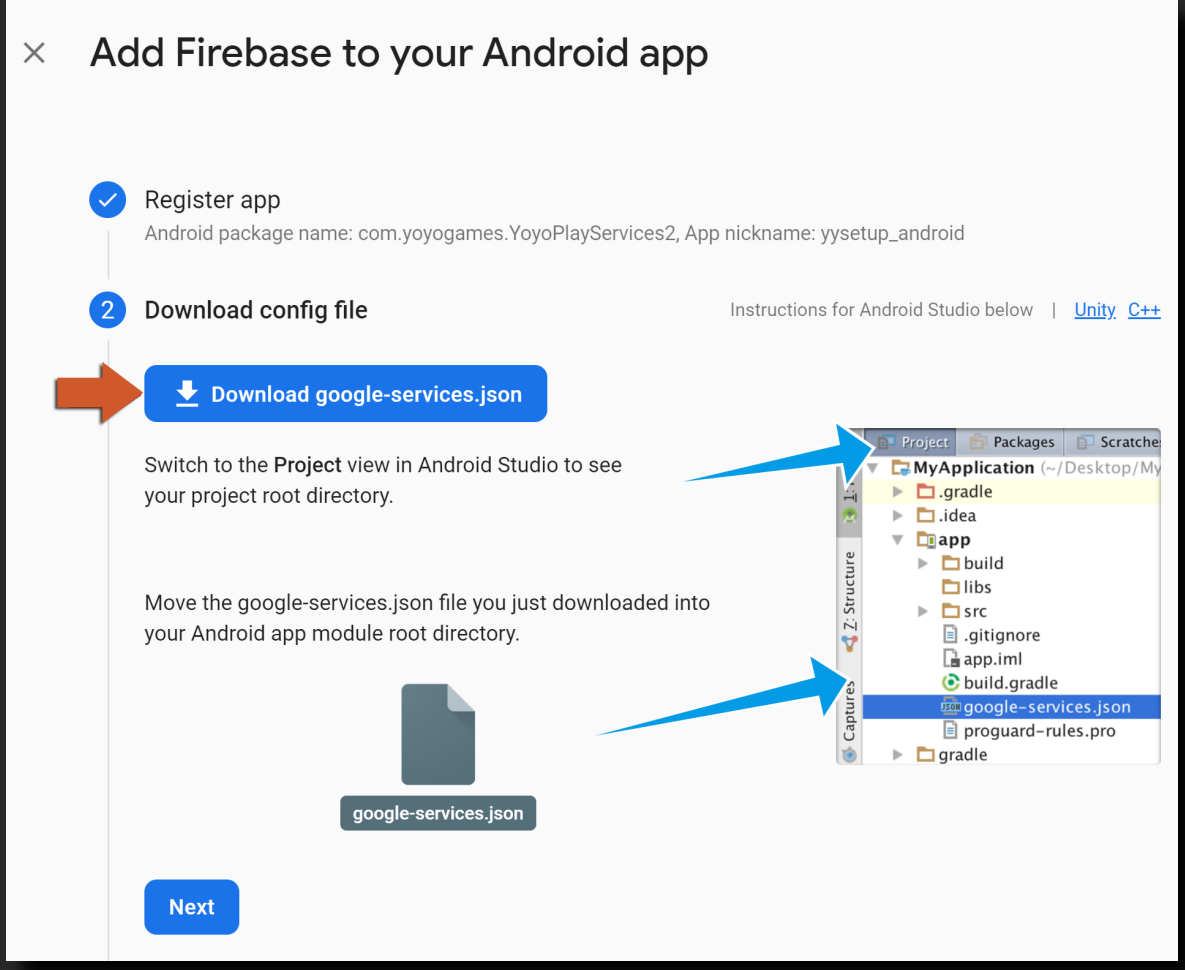
3. On this screen you need enter your **Package name** (required), **App nickname** (optional) and **Debug signing certificate SHA-1** (required if you are using Firebase Authentication).



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

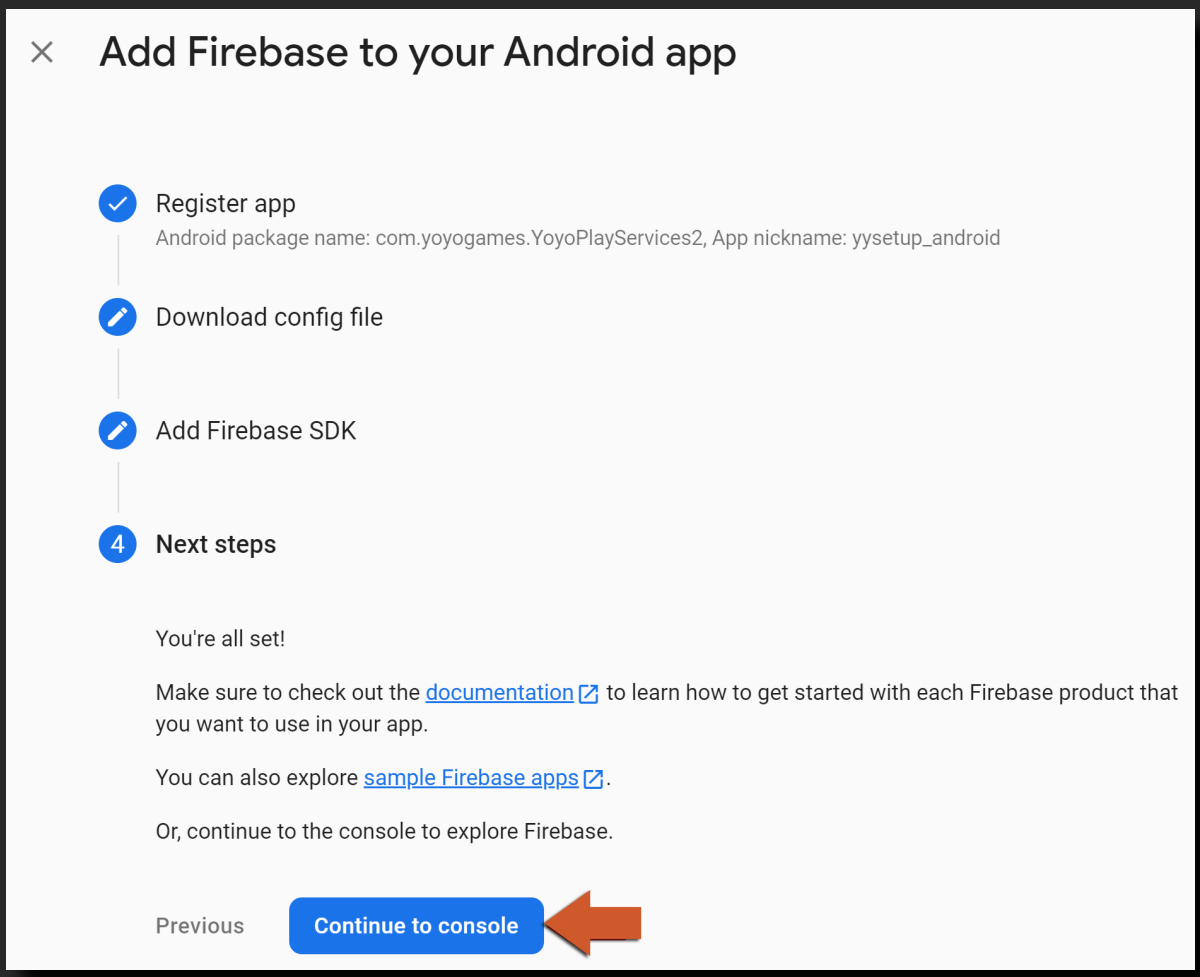


4. Click on **Download google-services.json** (make sure to save this file as we will need it in subsequent steps).

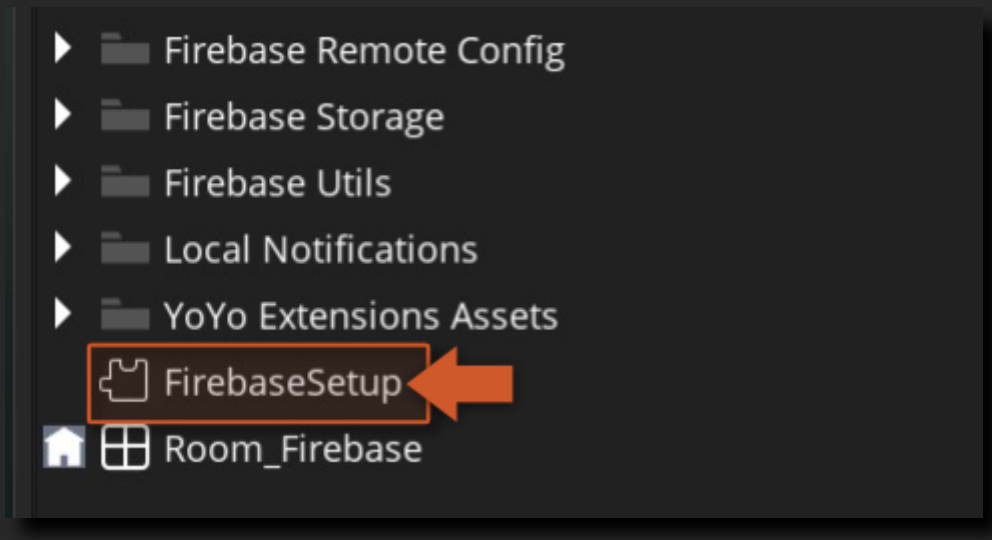


5. Ignore this screen, as this is already done in the extension.

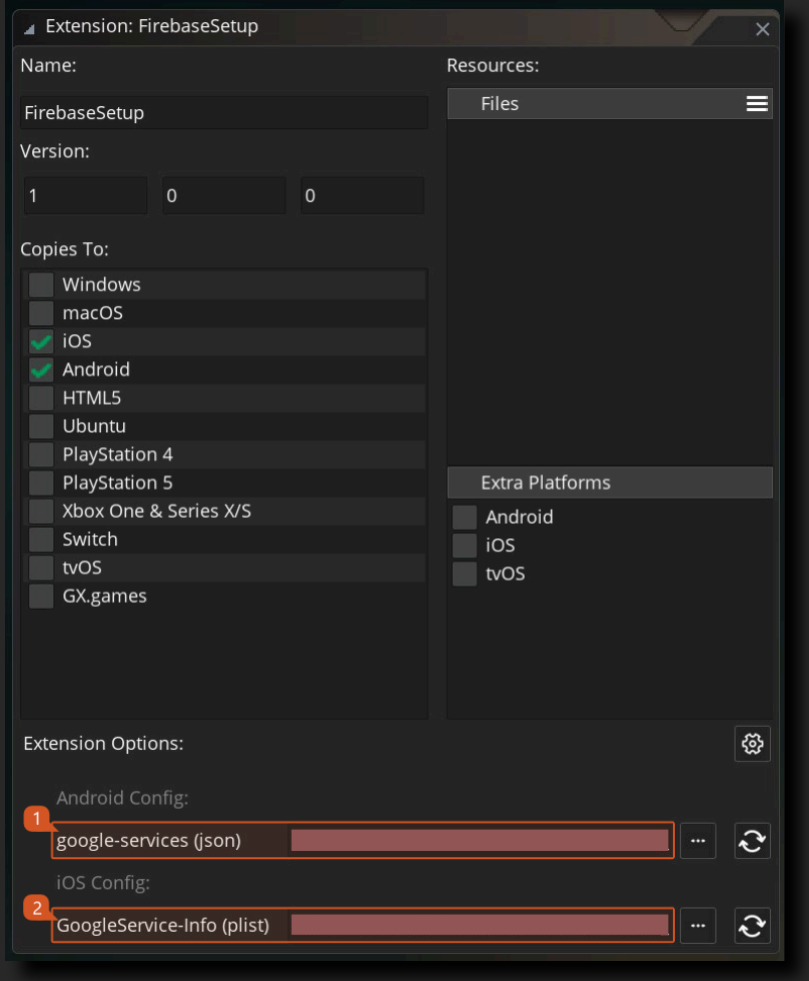
6. Click on the **Continue to console** button.



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).



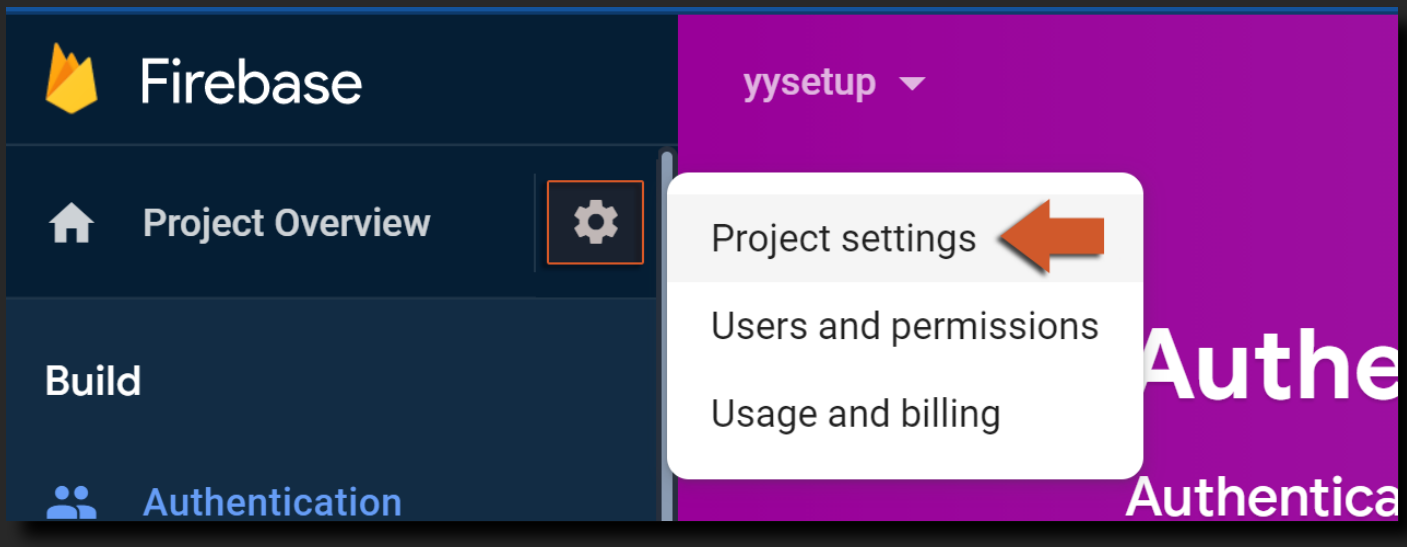
9. You have now finished the main setup for all Firebase Android modules!

iOS Setup

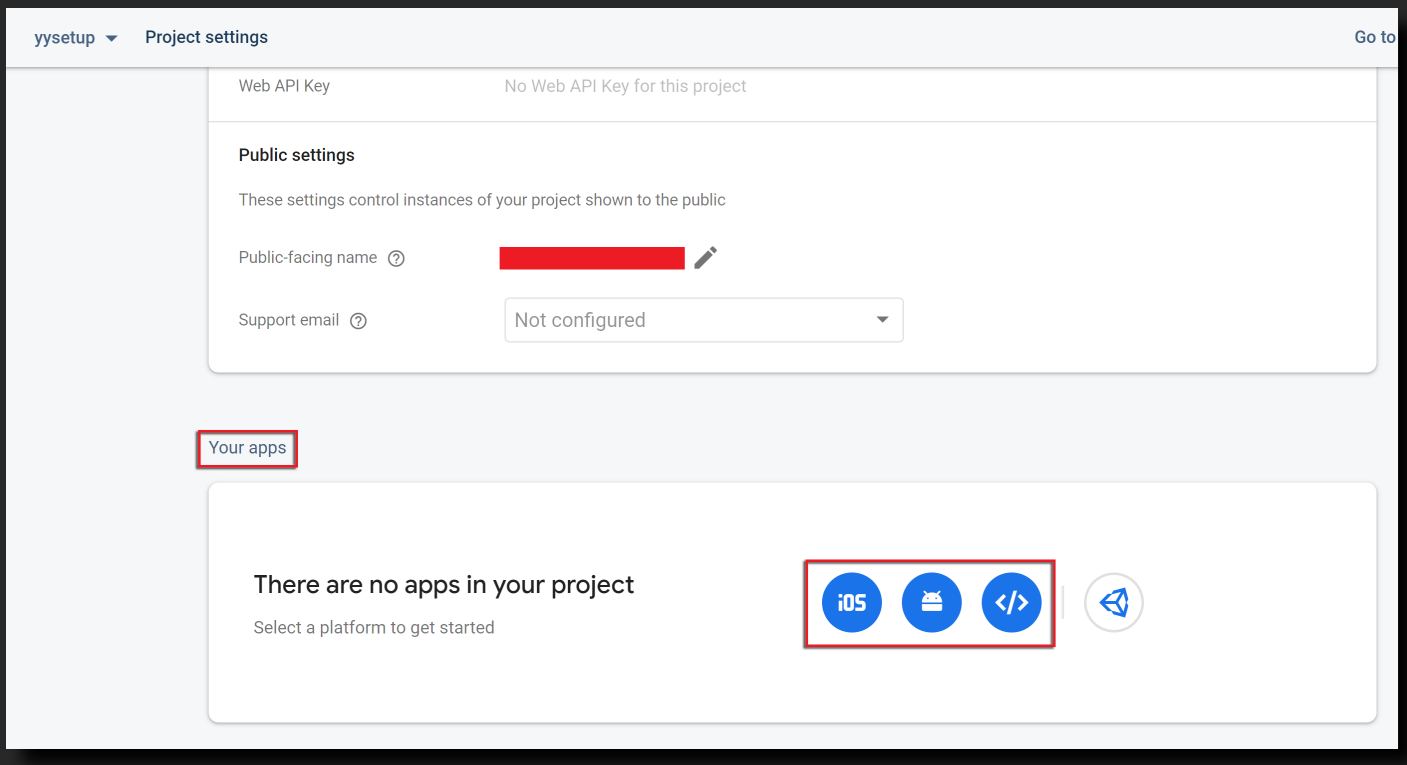
This setup is necessary for all the Firebase modules using iOS and needs to be done once per project, and basically involves importing the **GoogleServices-Info.plist** file into your project.

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

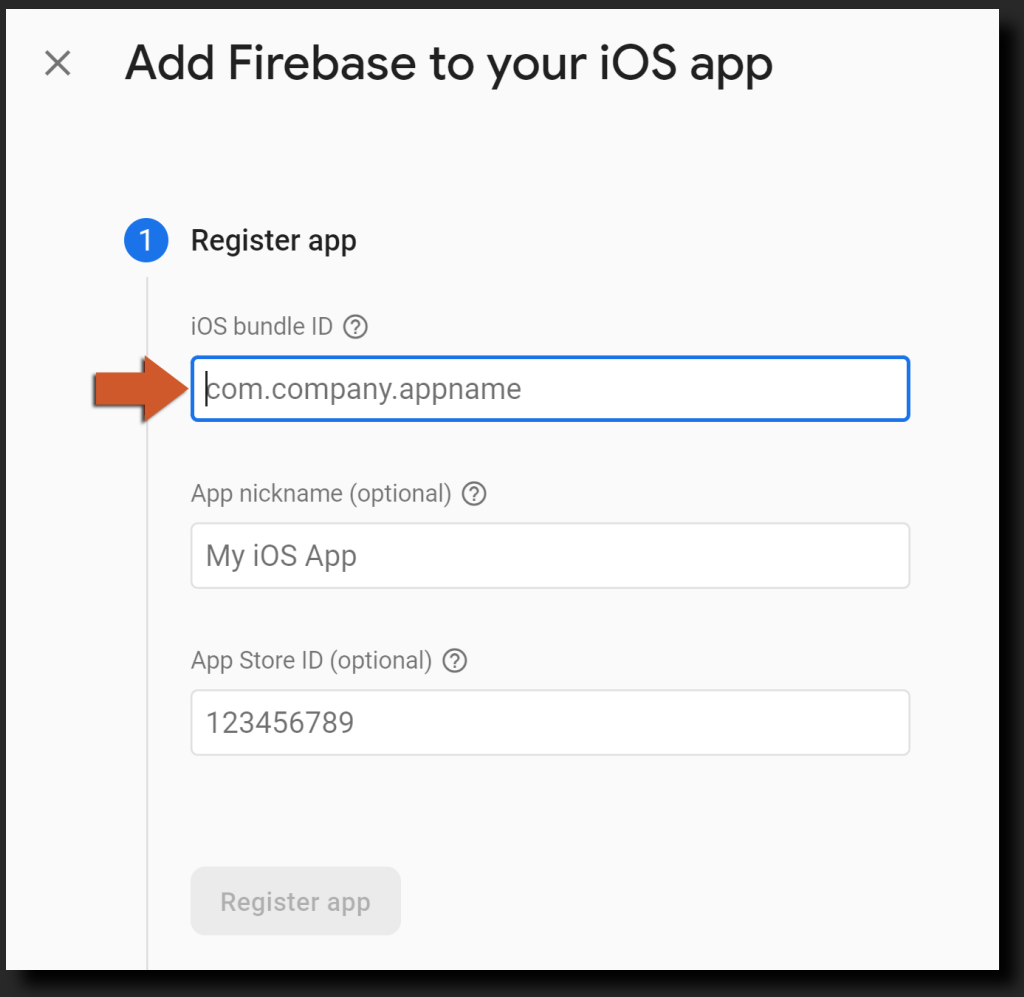
1. Click the **Settings** icon (next to Project Overview) and select **Project settings:**



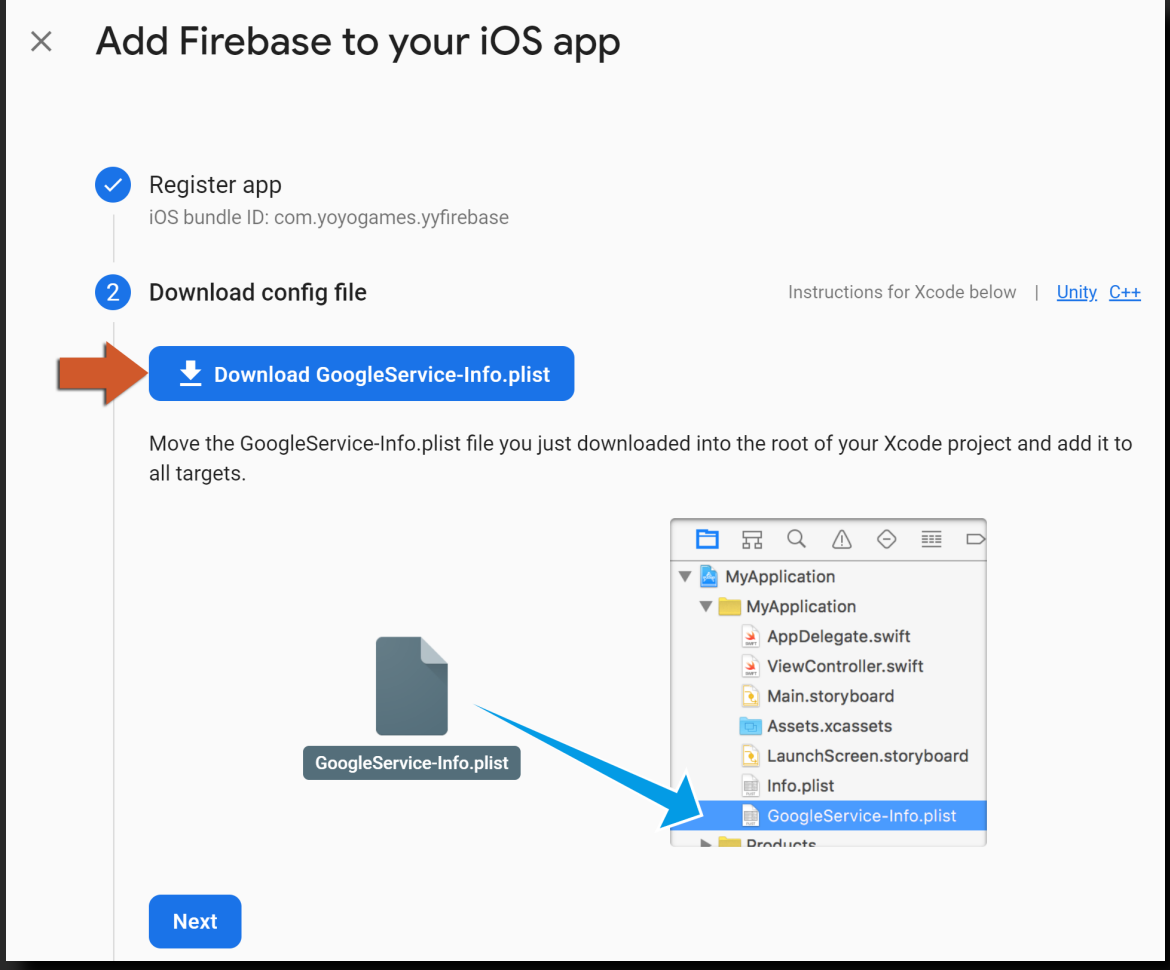
2. Now go to the **Your apps** section and click on the **iOS** button:



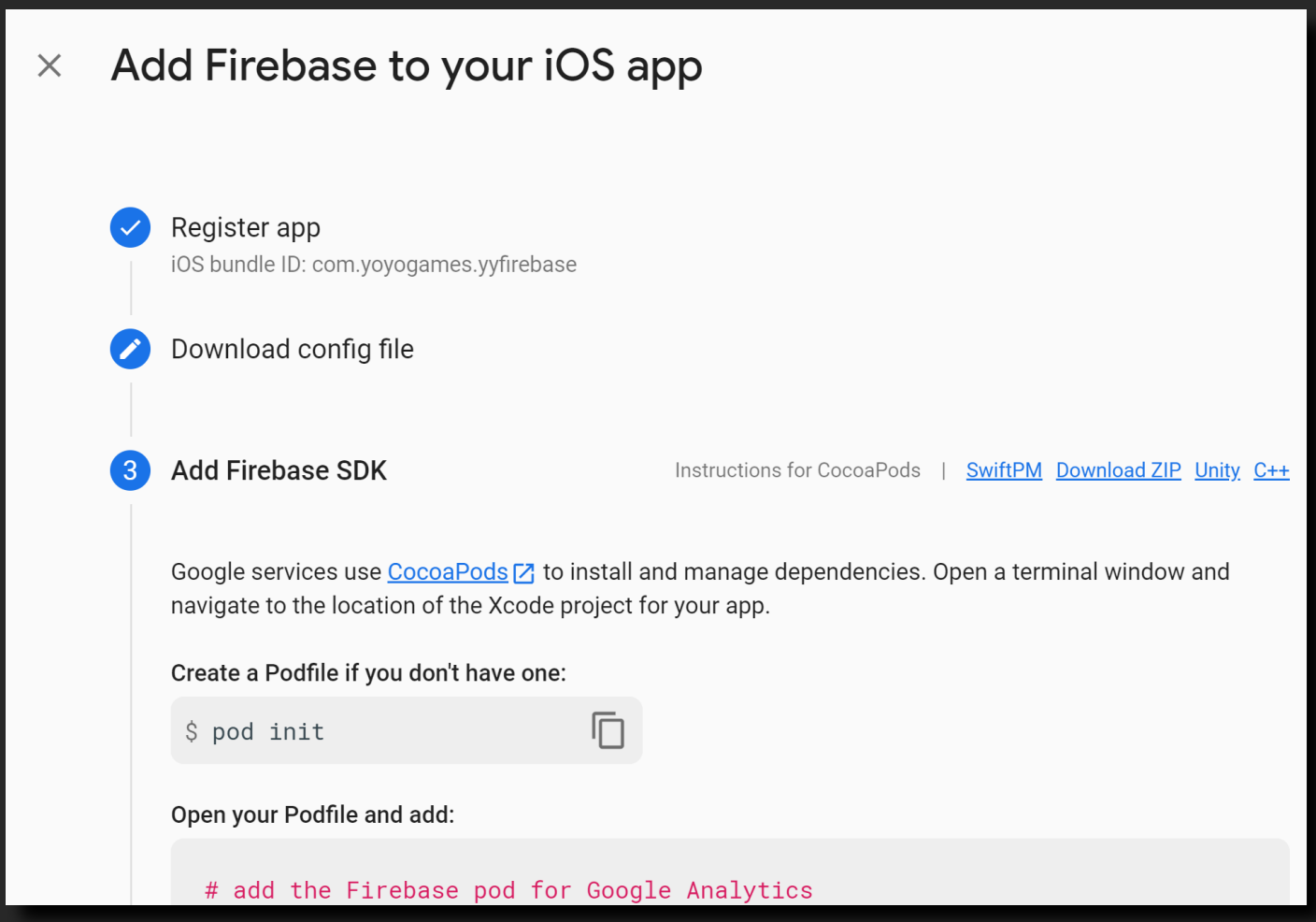
3. Fill the form with your **iOS Bundle ID**, **App nickname** and **AppStore ID** (last two are optional).



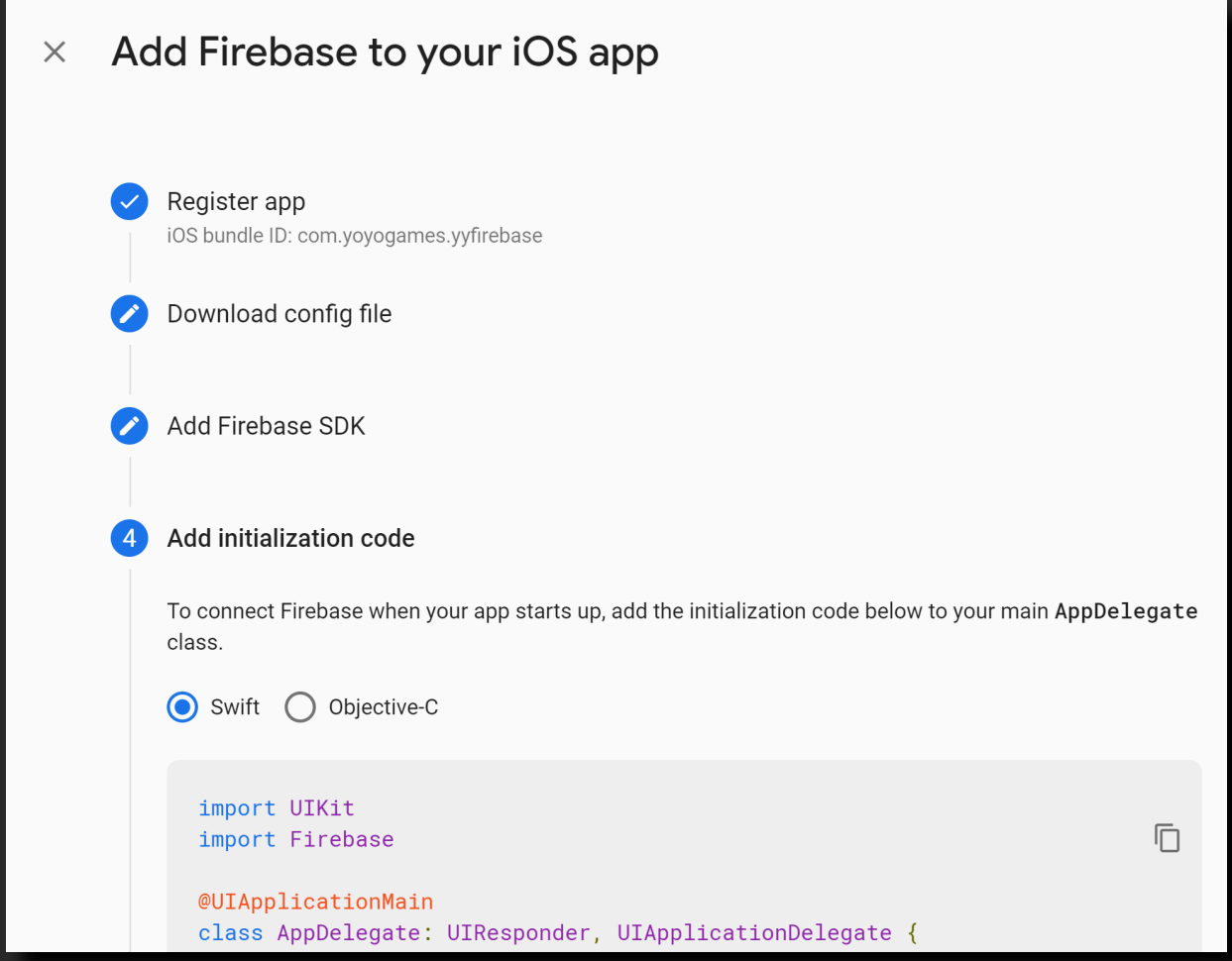
4. Click on **Download GoogleService-info.plist** (make sure to save this file as we will need it in subsequent steps).



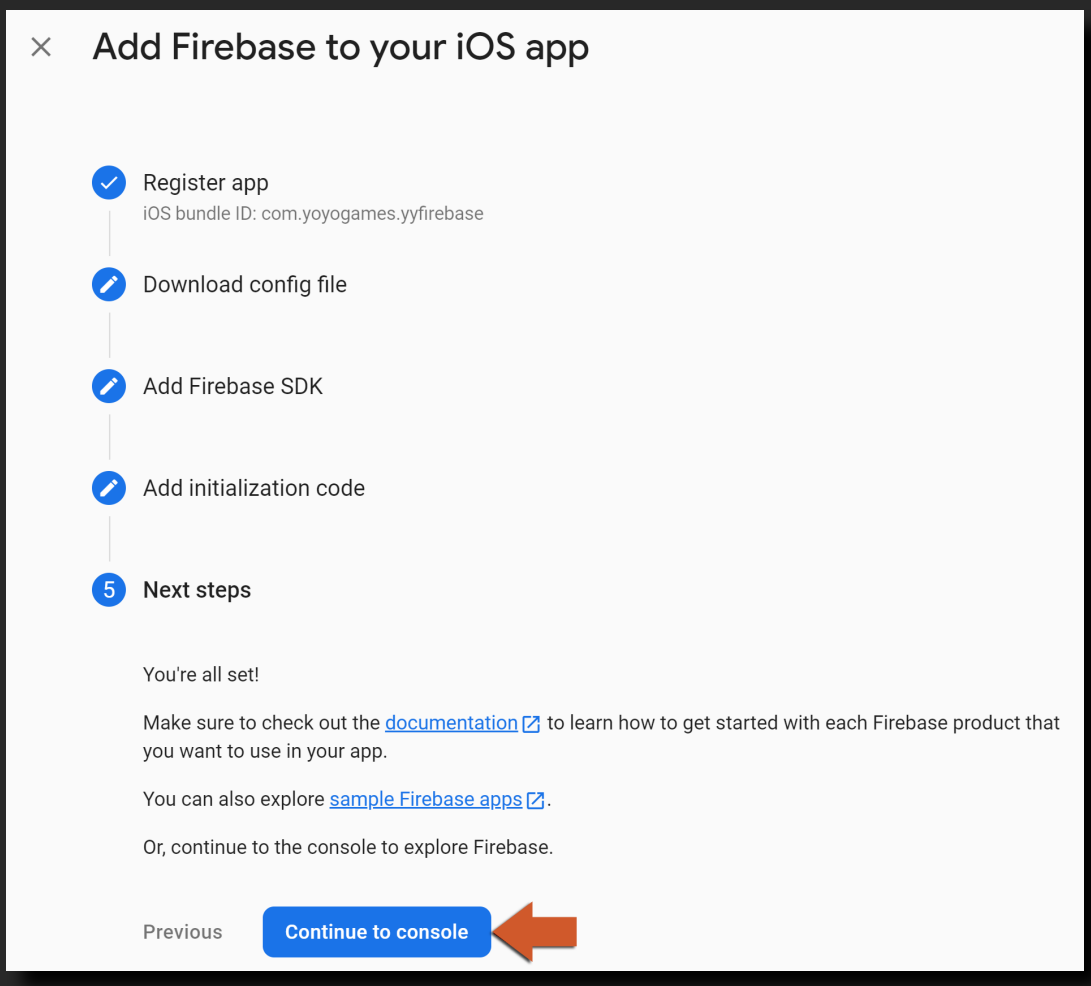
5. Ignore this screen, as this is already done in the extension.



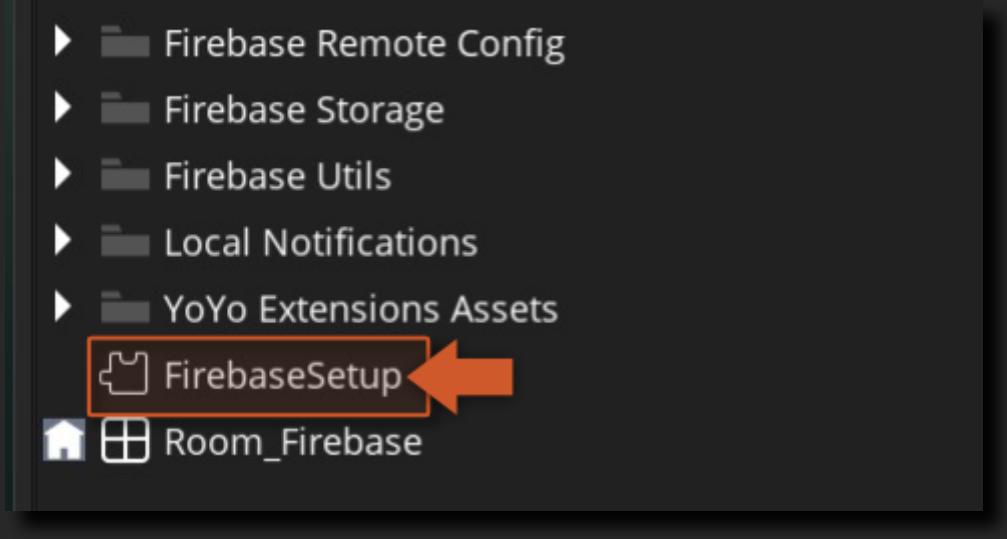
6. Ignore this screen as well, as this is also done in the extension.



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).

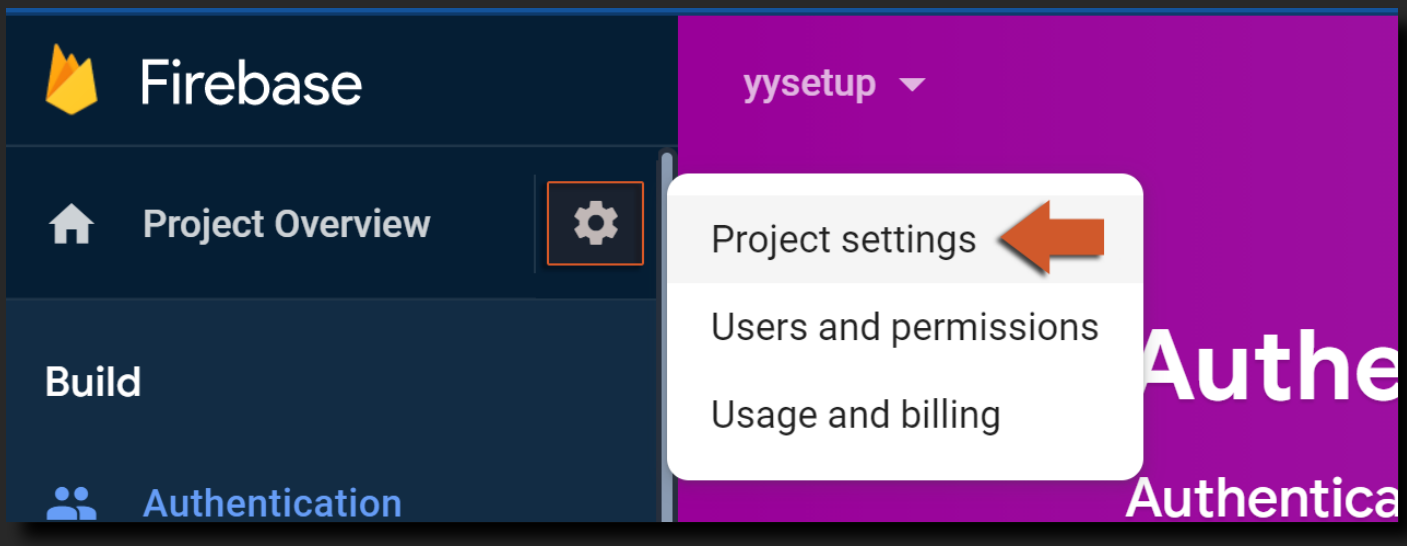
10. Make sure to set up **CocoaPods** for your project *unless* you are only using the REST API in an extension (if one is provided -- not all extensions provide a REST API) or the Firebase Cloud Functions extension (which only uses a REST API).

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

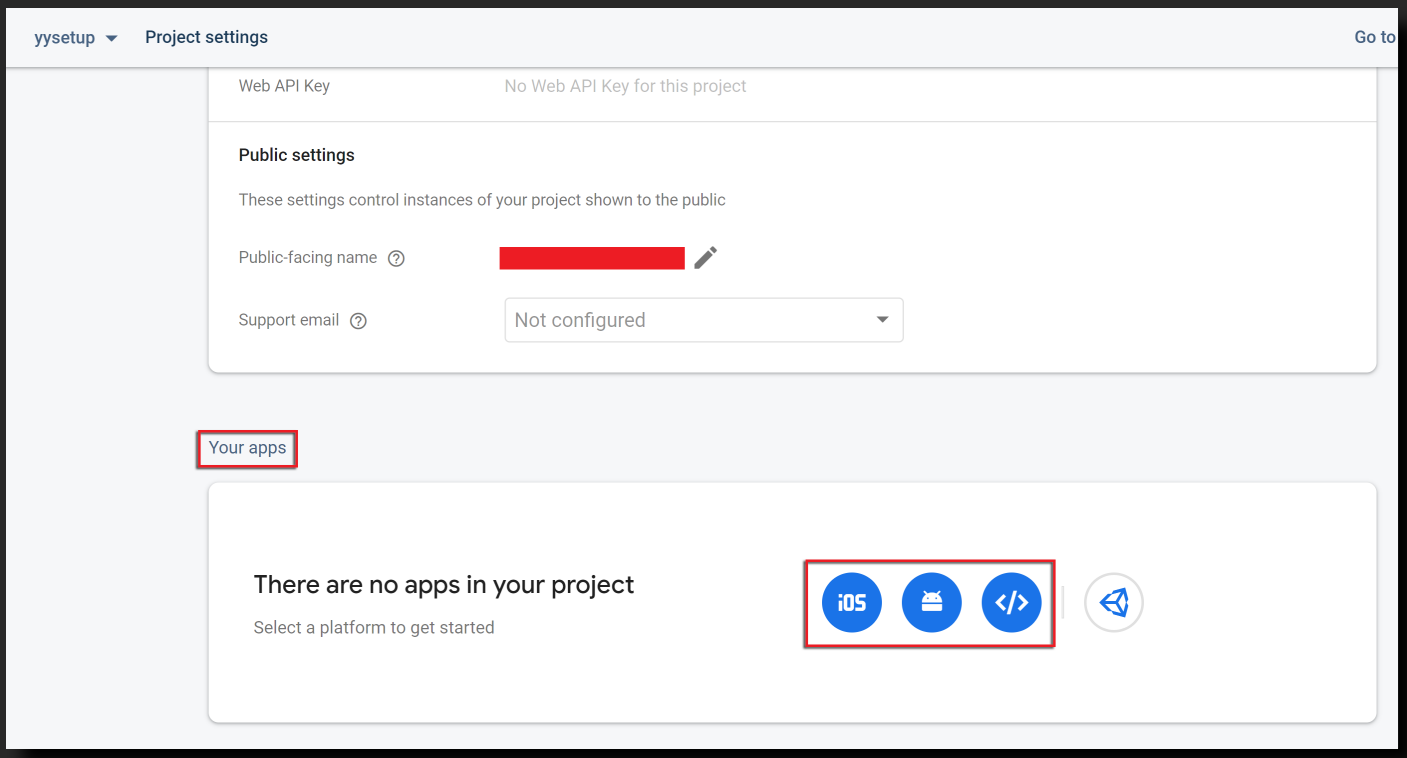
Web Setup

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.

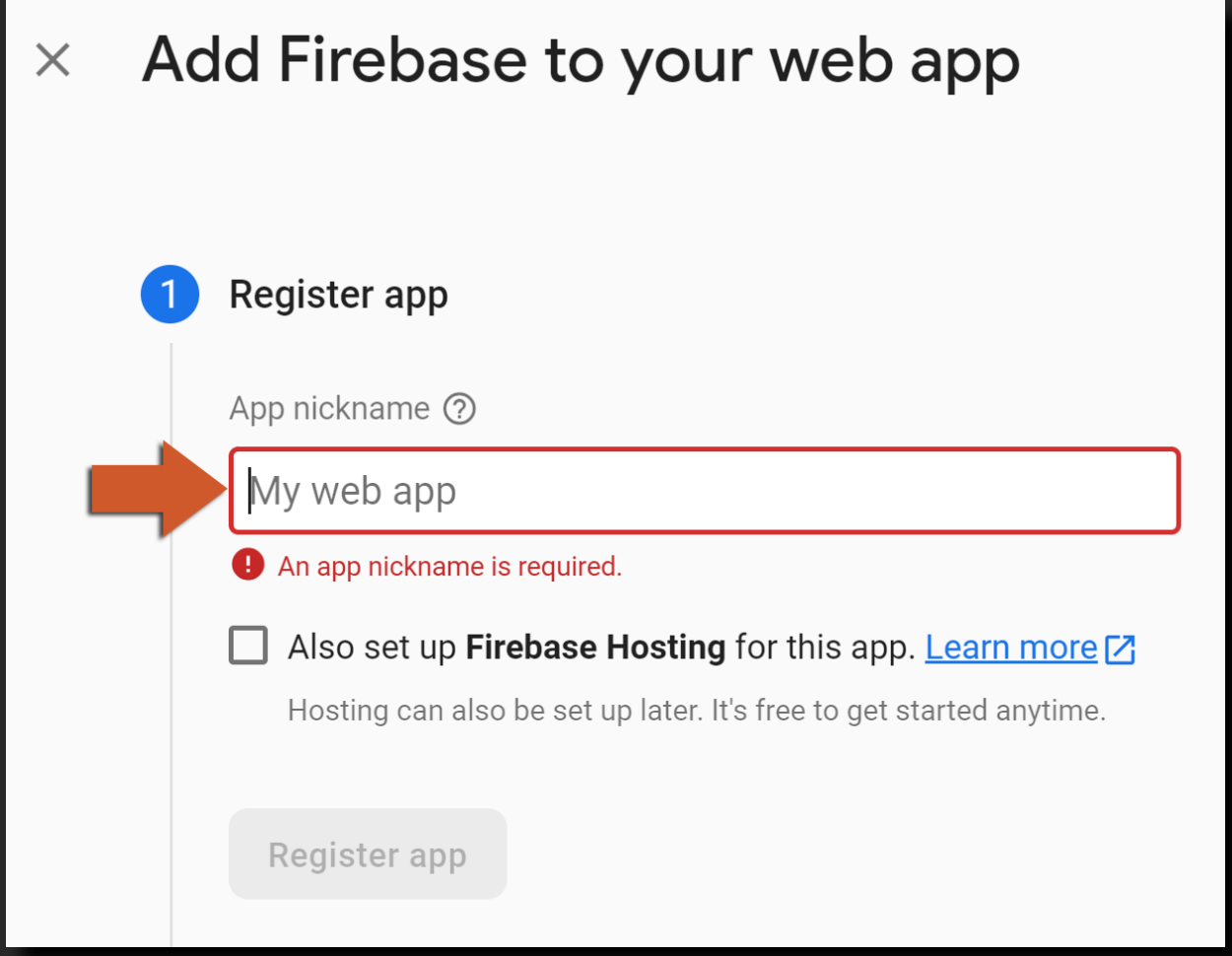
1. Click the **Settings** icon (next to Project Overview) and then **Project settings:**



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



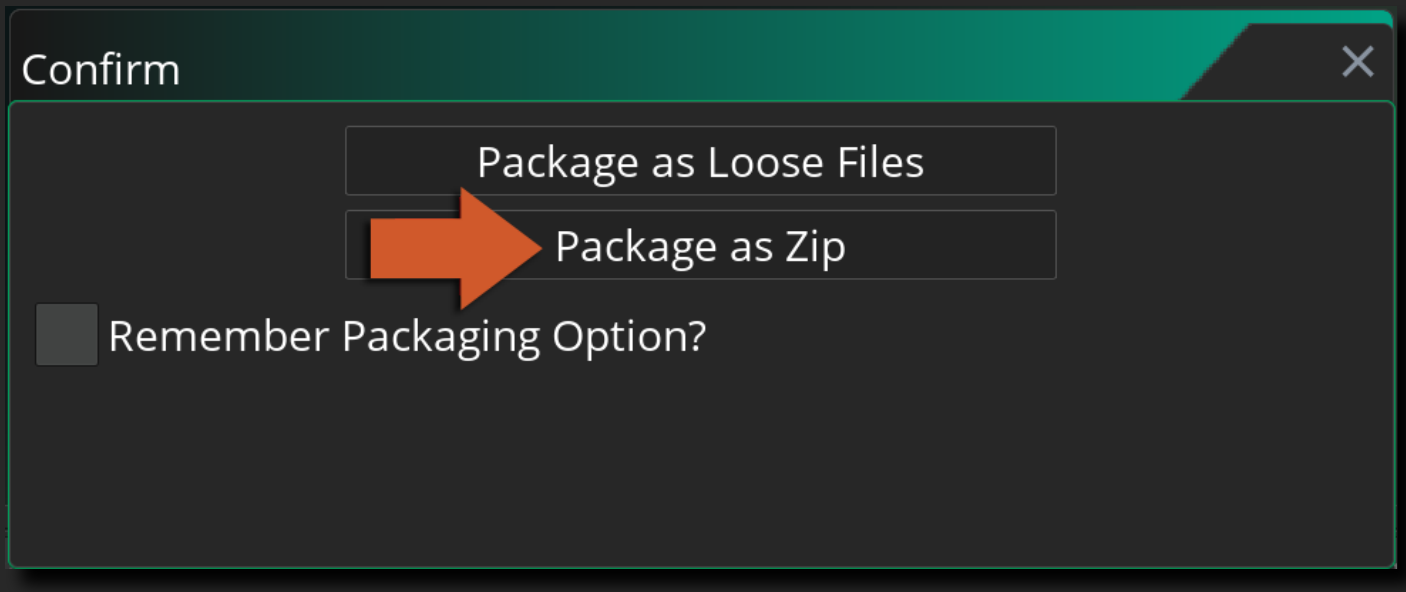
3. Enter your App nickname (required):



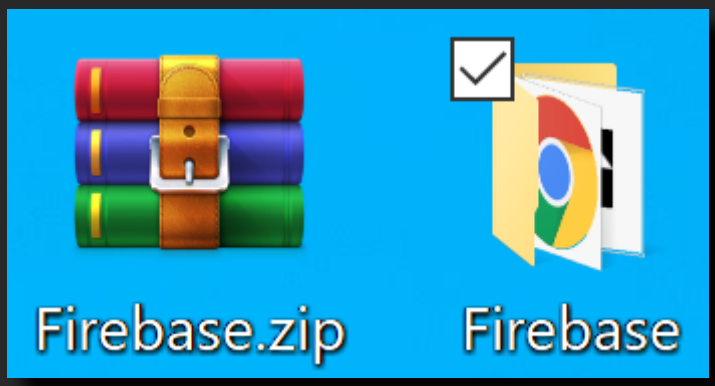
4. On this screen, just copy the **firebaseConfig** struct:

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

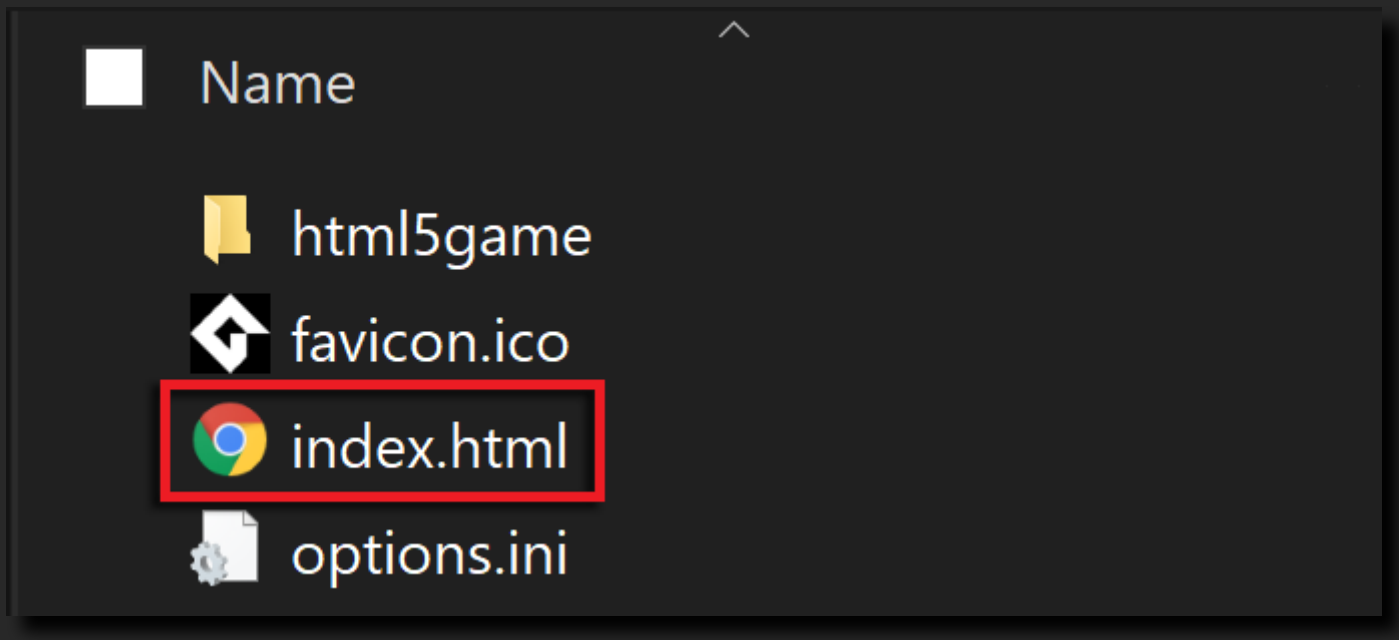
6. Choose the **Package As Zip** option:



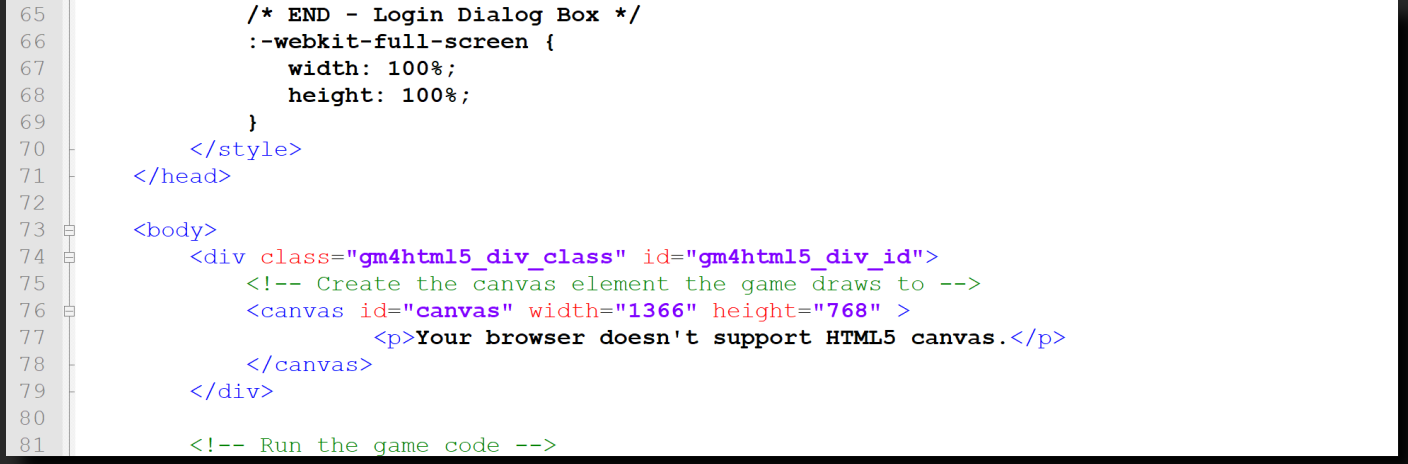
7. Locate the created package and **extract it** into a folder.



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).



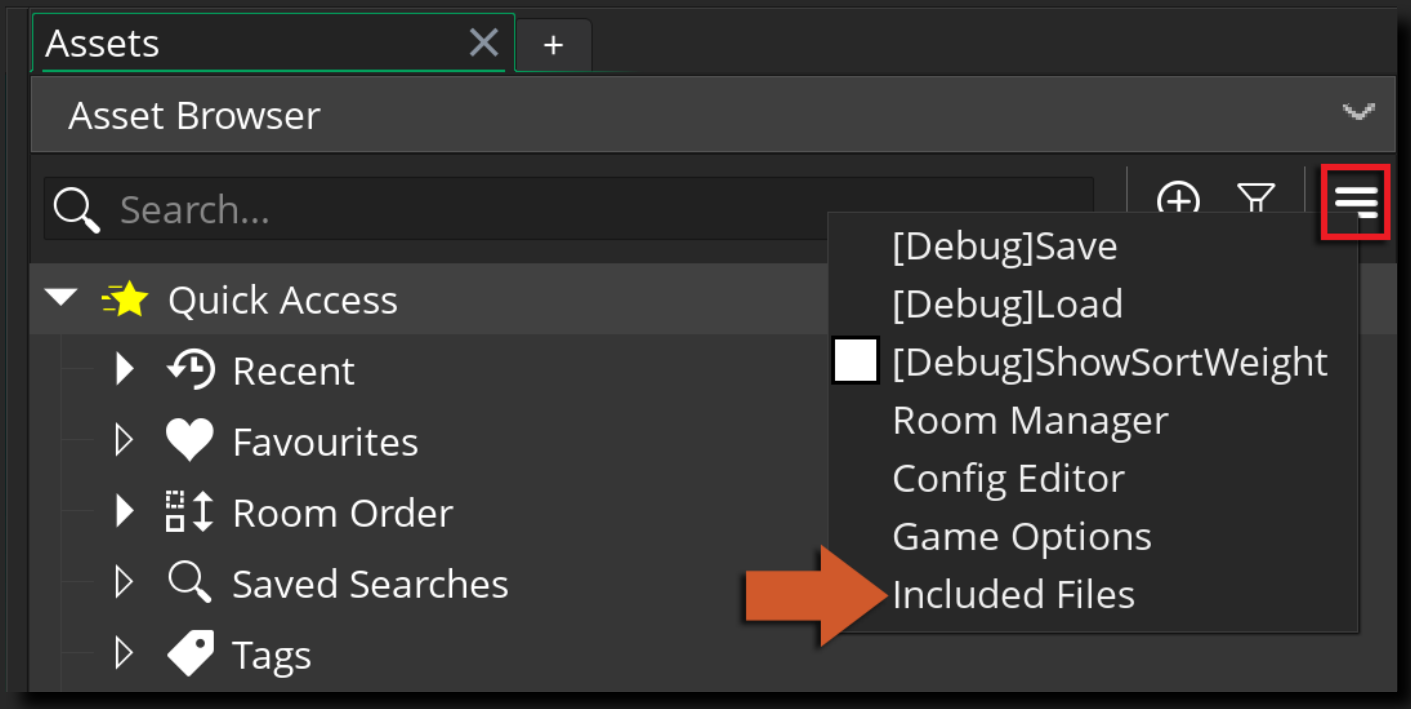
10. Now we need to add the following code between the `</head>` and `<body>` tags (line 72 in the **html.index** image above):

```
<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>
```

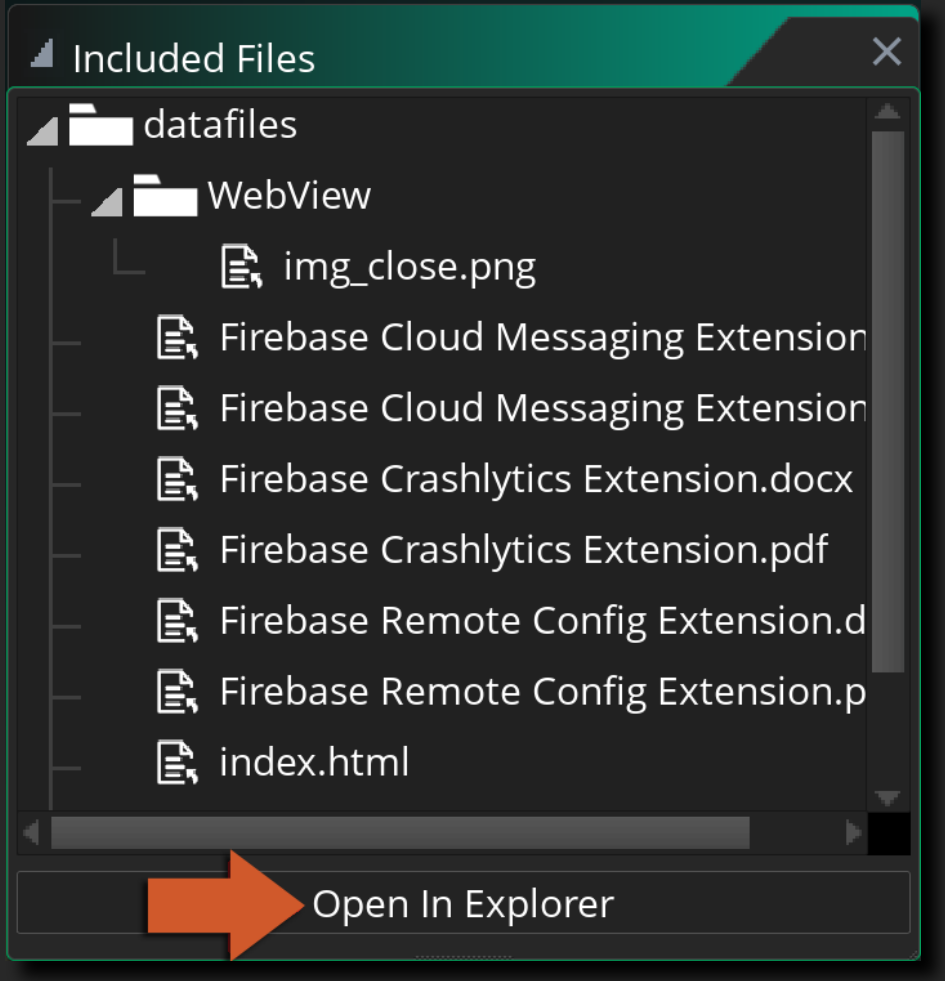
11. Replace the **const firebaseConfig** part with the one copied in step 4:

```
                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">
```

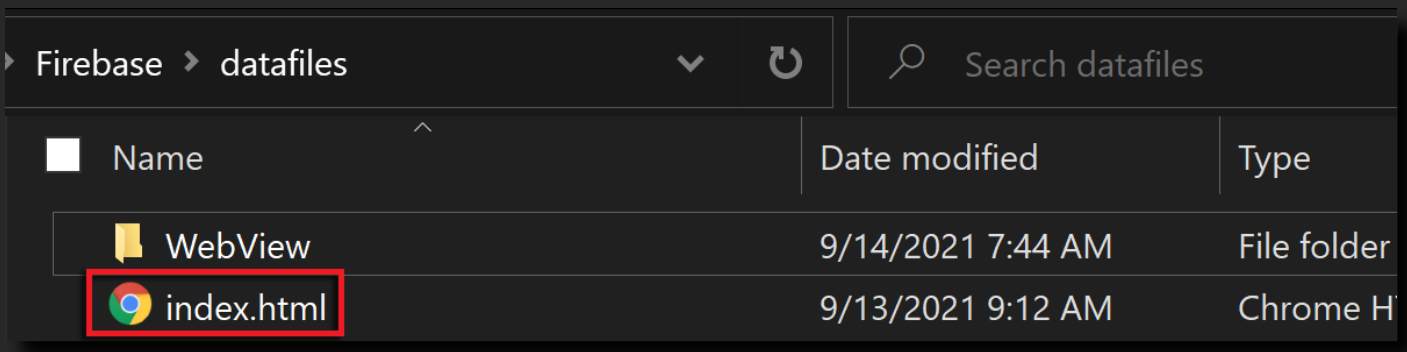
12. Go back into GameMaker and open your **Included Files** folder.



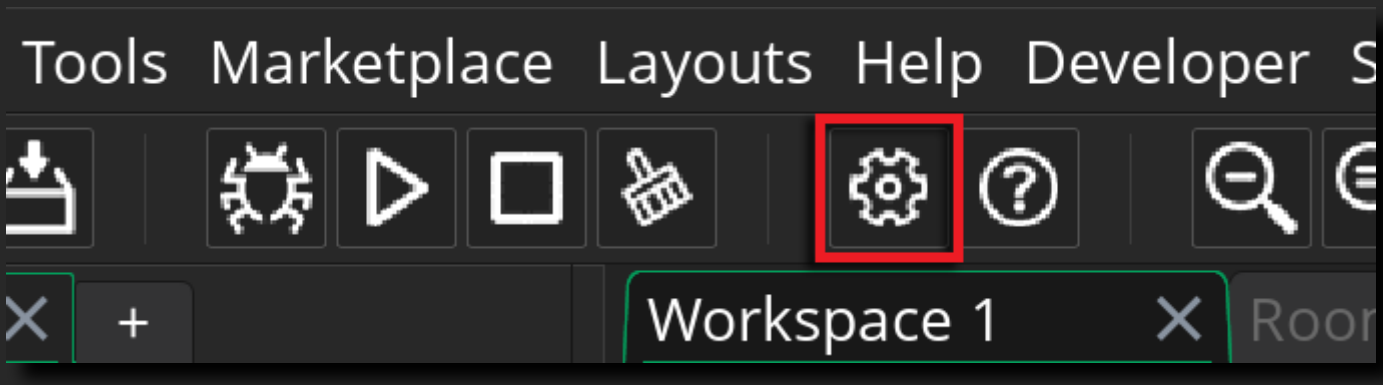
13. Press the **Open in Explorer** button:



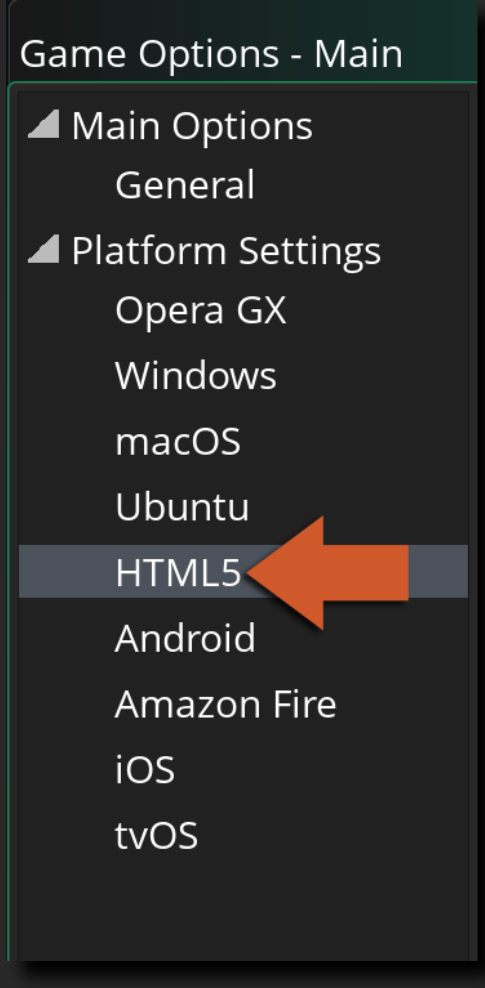
14. Place your **index.html** file inside the folder that opens (/datafiles).



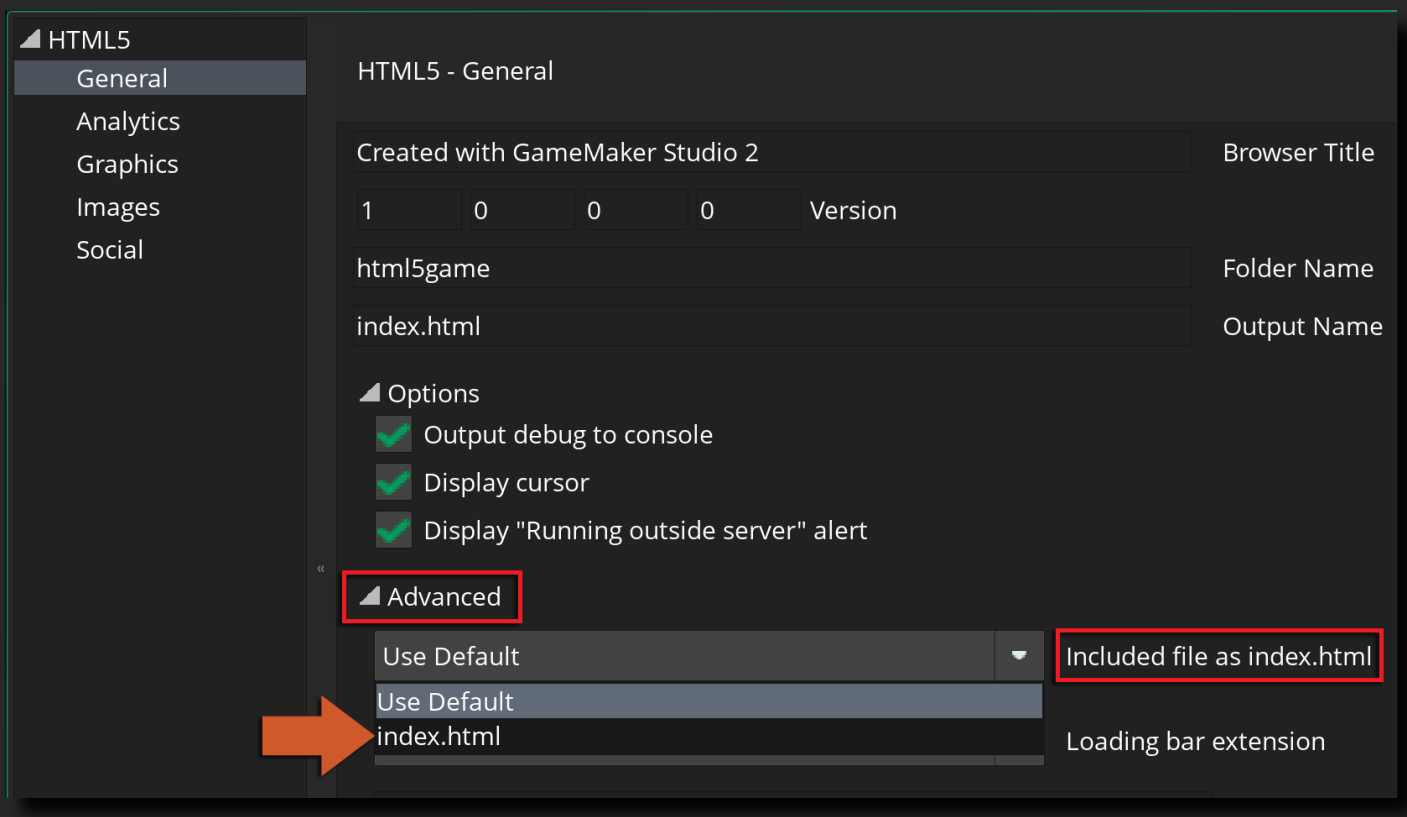
15. Back in GameMaker, click on the **Game Options button.**



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!

FirebaseRemoteConfig_FetchAndActivate

This function asynchronously fetches and then activates the fetched configs. If the time elapsed since the last fetch from the Firebase Remote Config backend is more than the set minimum fetch interval, configs are fetched from the backend. After the fetch is complete, the configs are activated so that the fetched key-value pairs take effect.

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

Syntax:

```
FirebaseRemoteConfig_FetchAndActivate()
```

Returns:

```
N/A
```

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseRemoteConfig_FetchAndActivate"`
success	boolean	Whether or not the function task succeeded.

Example:

```
FirebaseRemoteConfig_FetchAndActivate()
```

In the code above we are fetching and activating the Remote Config data from the Firebase servers. The function callback can be caught inside an Async Social event.

```
if (async_load[? "type"] == "FirebaseRemoteConfig_FetchAndActivate")
{
    if (async_load[? "success"])
    {
        var keysJSON = FirebaseRemoteConfig_GetKeys();
        var keys = json_parse(keysJSON);
        var count = array_length(keys);
        show_debug_message("Remote Config Data:");

        for(var i = 0 ; i < count; i++)
        {
            var key = keys[i];
            show_debug_message("key: " + FirebaseRemoteConfig_GetString(key));
        }
    }
}
```

The code above matches the response against the correct event **type**, and if the function task succeeded it gets all keys (using the function **FirebaseRemoteConfig_GetKeys**) that are afterwards parsed into an array (using the **json_parse** function). Finally it gets all their values as strings (using **FirebaseRemoteConfig_GetString**) and logs them to the console.

FirebaseRemoteConfig_GetDouble

This function returns the parameter value for the given key as a real. It evaluates the value of the parameter in the following order:

1. The activated value, if the last successful FirebaseRemoteConfig_FetchAndActivate contained the key.
2. The default value, if the key was set with FirebaseRemoteConfig_SetDefaultsAsync.
3. The default real value: `0` (zero).

Syntax:

```
FirebaseRemoteConfig_GetDouble(key)
```

Argument	Description
key	The key of the remote config parameter to get the value from.

Returns:

```
N/A
```

Example:

```
var magicNumber = FirebaseRemoteConfig_GetDouble("MagicNumber")
```

The code above reads the **double** value parameter for the key `"MagicNumber"` that is stored in the Remote Config Extension map.

FirebaseRemoteConfig_GetKeys

Returns a JSON formatted string of all the available keys.

Syntax:

```
FirebaseRemoteConfig_GetKeys();
```

Returns:

```
String (json formatted string)
```

Example:

```
var keysJSON = FirebaseRemoteConfig_GetKeys();
var keys = json_parse(keysJSON);
var count = array_length(keys);
show_debug_message("Remote Config Data:");

for(var i = 0 ; i < count; i++)
{
     var key = keys[i];
     show_debug_message("key: " + FirebaseRemoteConfig_GetString(key)));
}
```

The code above gets all keys (using the function **FirebaseRemoteConfig_GetKeys**) that are afterwards parsed into an array (using the **json_parse** function). Finally it gets all their values as strings (using **FirebaseRemoteConfig_GetString**) and logs them to the console.

FirebaseRemoteConfig_GetString

This function returns the parameter value for the given key as a string. It evaluates the value of the parameter in the following order

1. The activated value, if the last successful FirebaseRemoteConfig_FetchAndActivate contained the key.
2. The default value, if the key was set with FirebaseRemoteConfig_SetDefaultsAsync.
3. The default string value: `""` (empty string).

Syntax:

```
FirebaseRemoteConfig_GetDouble(key)
```

Argument	Description
key	The key of the remote config parameter to get the value from.

Returns:

```
String
```

Example:

```
var configuration = FirebaseRemoteConfig_GetString("Configuration")
```

The code above reads the **string** value parameter for the key `"Configuration"` that is stored in the Remote Config Extension map.

FirebaseRemoteConfig_Initialize

This function initializes the Firebase Remote Config extension and sets the minimum interval between successive fetch calls. Any fetches performed within the interval of the last fetch will use values returned during that fetch.

NOTE The default fetch interval is 3600 seconds.

Syntax:

```
FirebaseRemoteConfig_Initialize(fetchInterval)
```

Argument	Type	Description
fetchInterval	real	The minimum fetch Interval duration in seconds.

Returns:

```
N/A
```

Example:

```
FirebaseRemoteConfig_Initialize(1000)
```

The code above initializes the Firebase Remote Config extension and sets the fetch interval to `1000` seconds.

FirebaseRemoteConfig_Reset

This function deletes all activated, fetched and defaults configs and resets all Firebase Remote Config settings.

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

Syntax:

```
FirebaseRemoteConfig_Reset()
```

Returns:

```
N/A
```

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseRemoteConfig_Reset"`
success	boolean	Whether or not the function task succeeded.

Example:

```
FirebaseRemoteConfig_Reset();
```

In the code above we request for the reset of the Remote Config value. The function callback can be caught inside an Async Social event.

```
if (async_load[? "type"] == "FirebaseRemoteConfig_Reset")
{
    if (async_load[? "success"])
    {
        show_debug_message("FirebaseRemoteConfig_Reset() SUCCEEDED")
    }
```

```
    else
    {
        show_debug_message("FirebaseRemoteConfig_Reset() FAILED")
    }
}
```

The code above matches the response against the correct event **type**, and logs the success of the operation.

FirebaseRemoteConfig_SetDefaultsAsync

This function asynchronously sets the default configs using the given JSON formatted string with a set of key-value pairs. Contained values can only be strings or reals.

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

Syntax:

```
FirebaseRemoteConfig_SetDefaultsAsync(jsonString)
```

Argument	Type	Description
jsonString	string	The json formatted representation of a struct with default key/value pairs.

Returns:

```
N/A
```

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseRemoteConfig_SetDefaultsAsync"`
success	boolean	Whether or not the function task succeeded.

Example:

```
var jsonData = { name: "John", age: 10 };

FirebaseRemoteConfig_SetDefaultsAsync(json_stringify(jsonData));
```

In the code above we request to set the default values for the parameters inside the struct (`jsonData`) the struct is then converted to JSON formatted string (using the built in function **json_stringify**). The function callback can be caught inside an **Async Social** event.

```
if (async_load[? "type"] == "FirebaseRemoteConfig_SetDefaultsAsync")
{
    if (async_load[? "success"])
    {
        show_debug_message("Default values were set successfully!");
    }
    else
    {
        show_debug_message("Default values failed to be set!");
    }
}
```

The code above matches the response against the correct event **type**, and logs the success of the operation.

Platform Setup

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

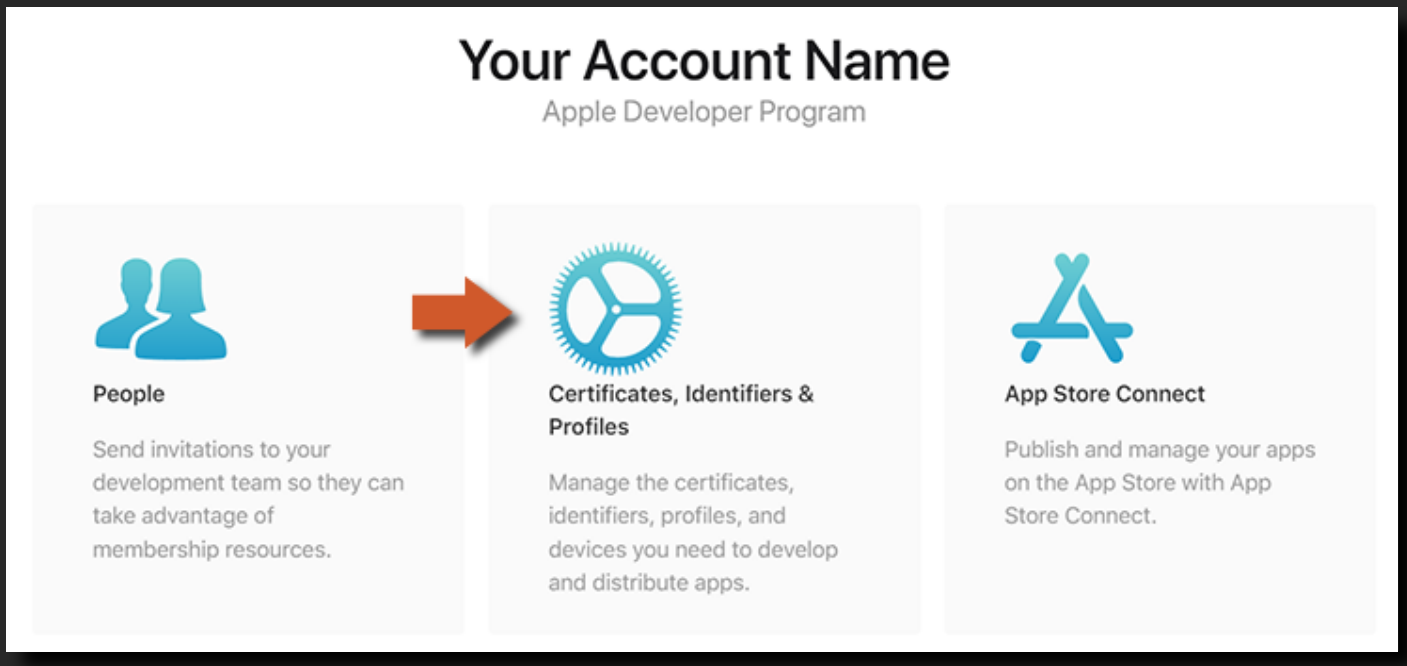
Select your target platform below and follow the simple steps to get your project up and running (you only need follow this setup once per project):

- **Android Setup**
- **iOS Setup**

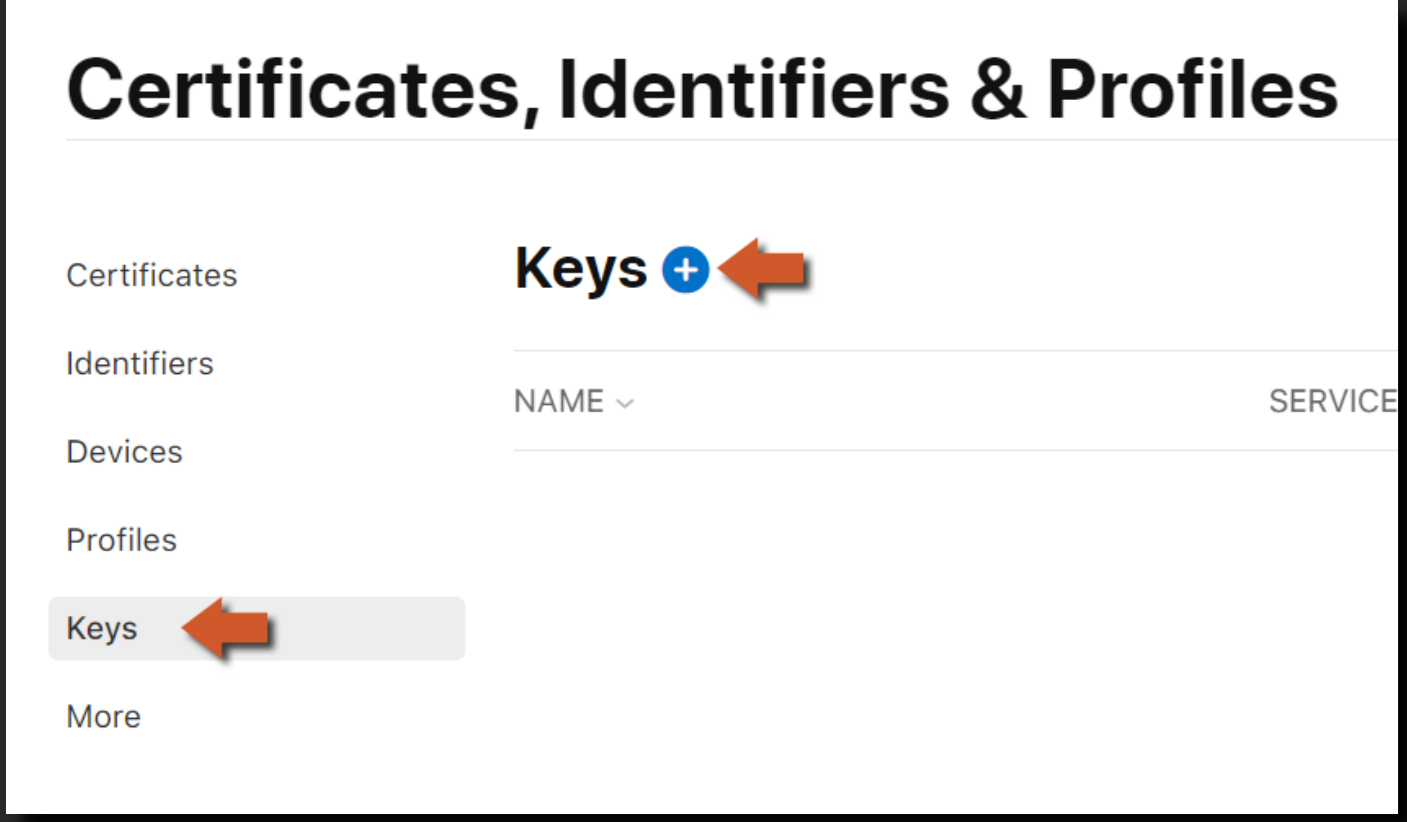
Additional steps for iOS

On iOS you will need to retrieve a P8 certificate and upload it to your Firebase project to enable sending push notifications through APNs:

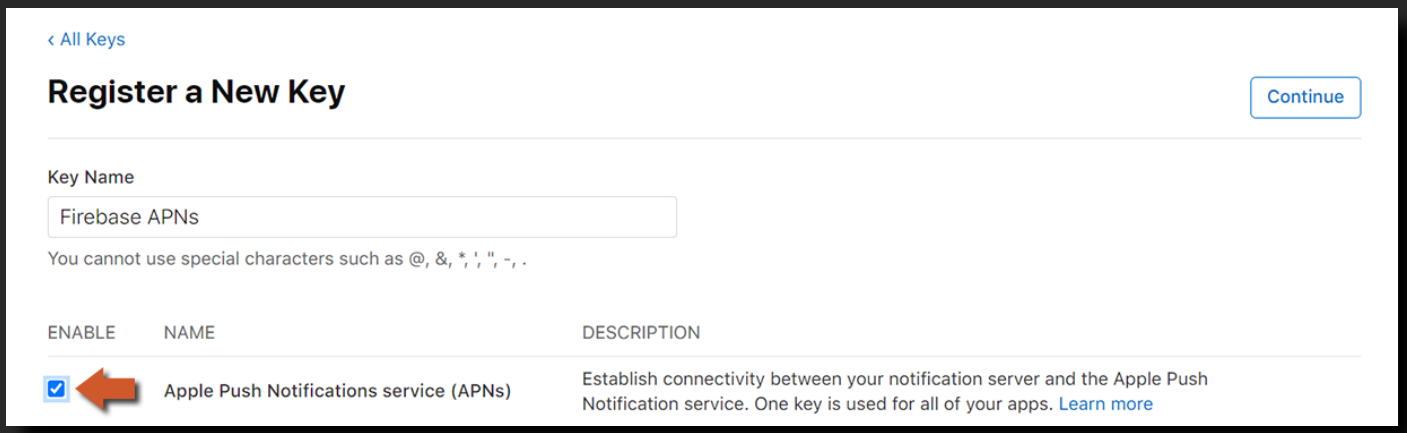
- Head to the **Apple Developer** site and select "**Certificates, Identifiers & Profiles**".



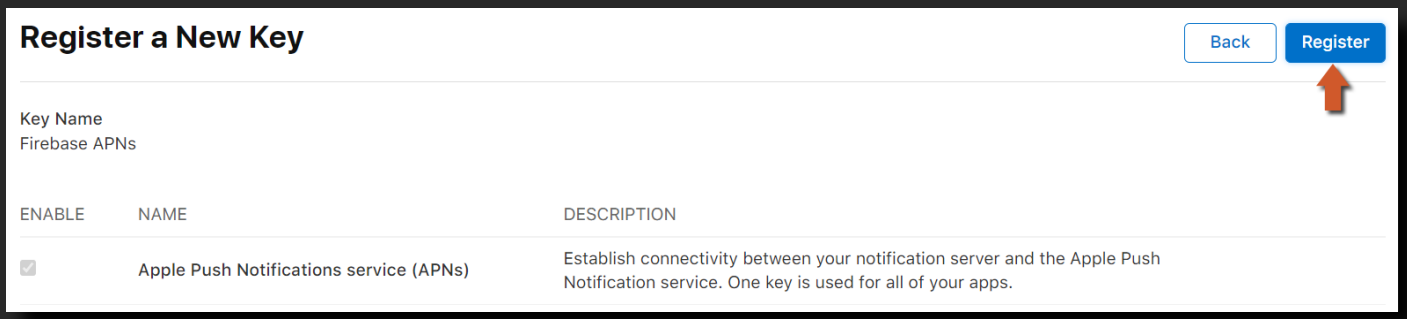
- Select "**Keys**" from the menu on the left, and create a new key by clicking on the plus sign.



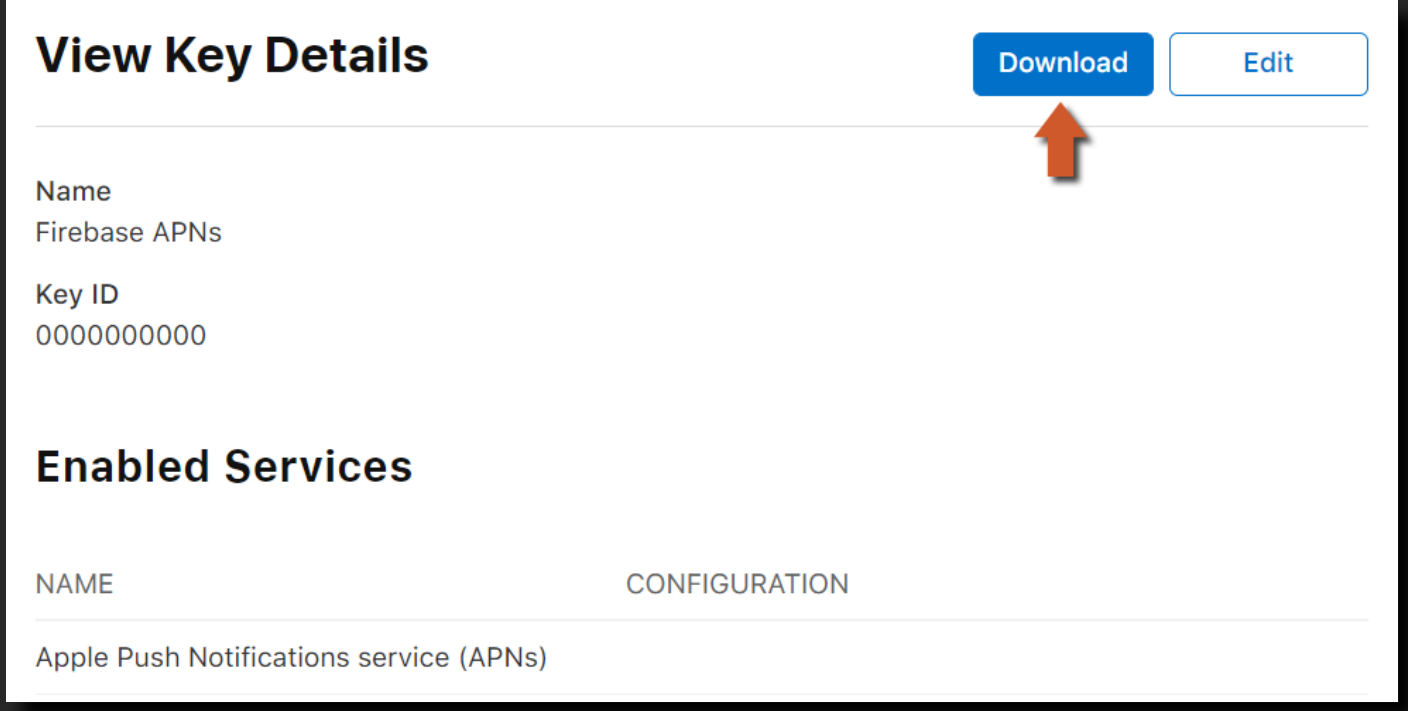
- Enter a **name** for the key, enable **Apple Push Notifications service** (APNs) and click on **Continue**.



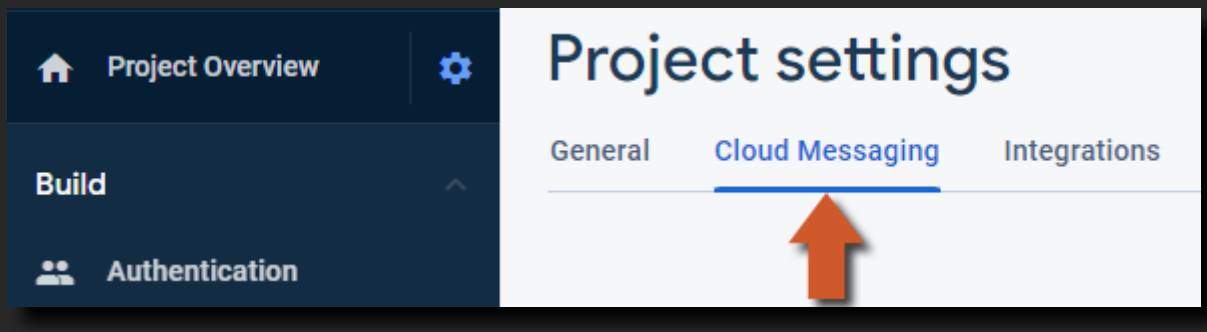
- On the next page, confirm the key details and click on **Register**.



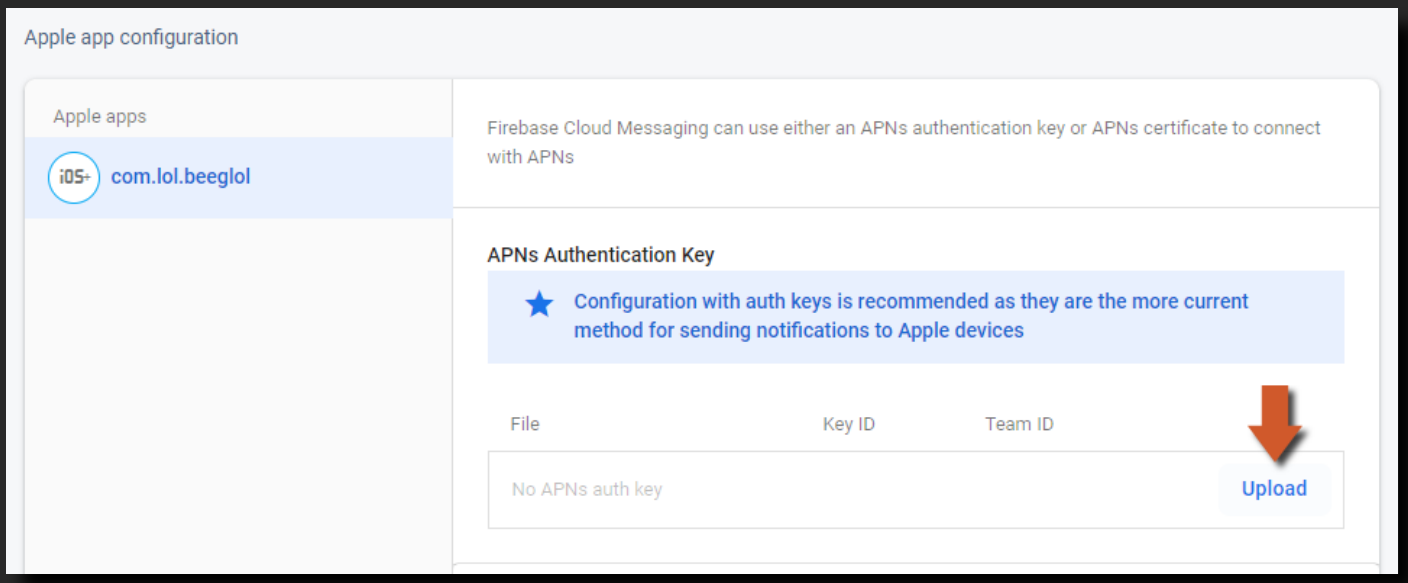
- **Note** the information given here (key ID) and **download** the key as you will not be able to see this screen again.



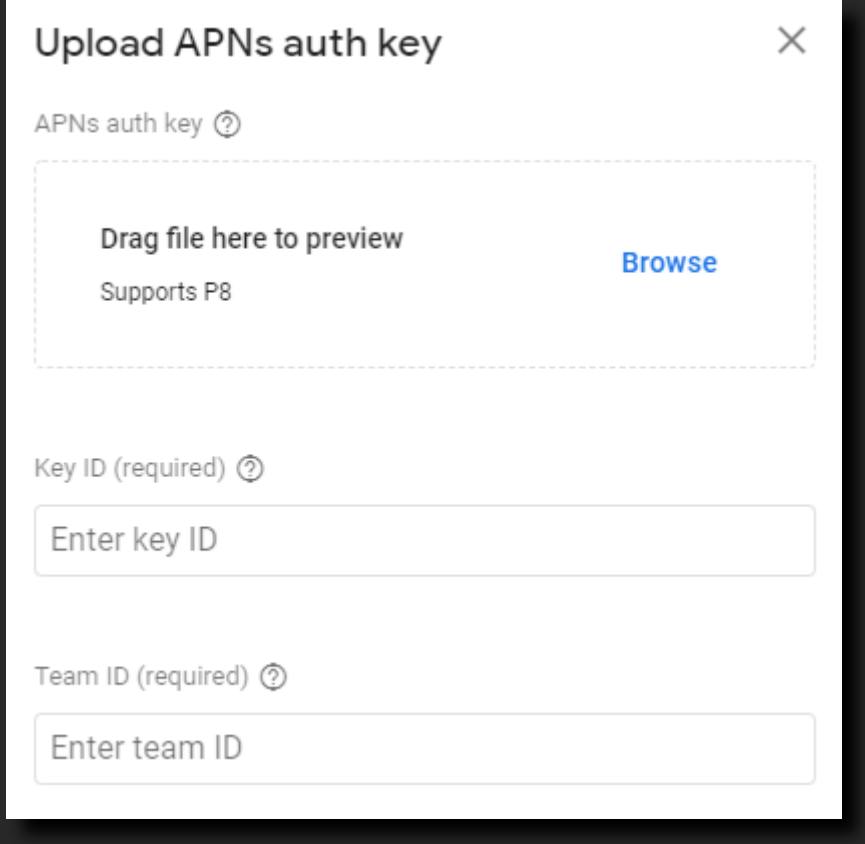
- Go to the dashboard for your Firebase project and open the **Project Settings**. Here, open the **Cloud Messaging** tab.



- Select your iOS application, and under "**APNs Authentication Key**", press **Upload** to upload your key.



- Here, upload your P8 file and enter the other required details that you retrieved from the Apple Developer site.



You can now send notifications to the iOS client game by going under **"Engage"** and selecting **"Cloud Messaging"** on your Firebase dashboard.