Firebase Cloud Messaging

Using Firebase Cloud Messaging you can notify an app client that a new email or other data is available to be synced. You can send notification messages to drive user re-engagement and retention. For use cases such as instant messaging, a message can transfer a payload of up to 4000 bytes to an app client.

Check the official page for more information.

Setup

Before starting to use any Firebase extensions, you are required to follow some initial configuration steps. 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 (iOS requires some additional steps)

Functions

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

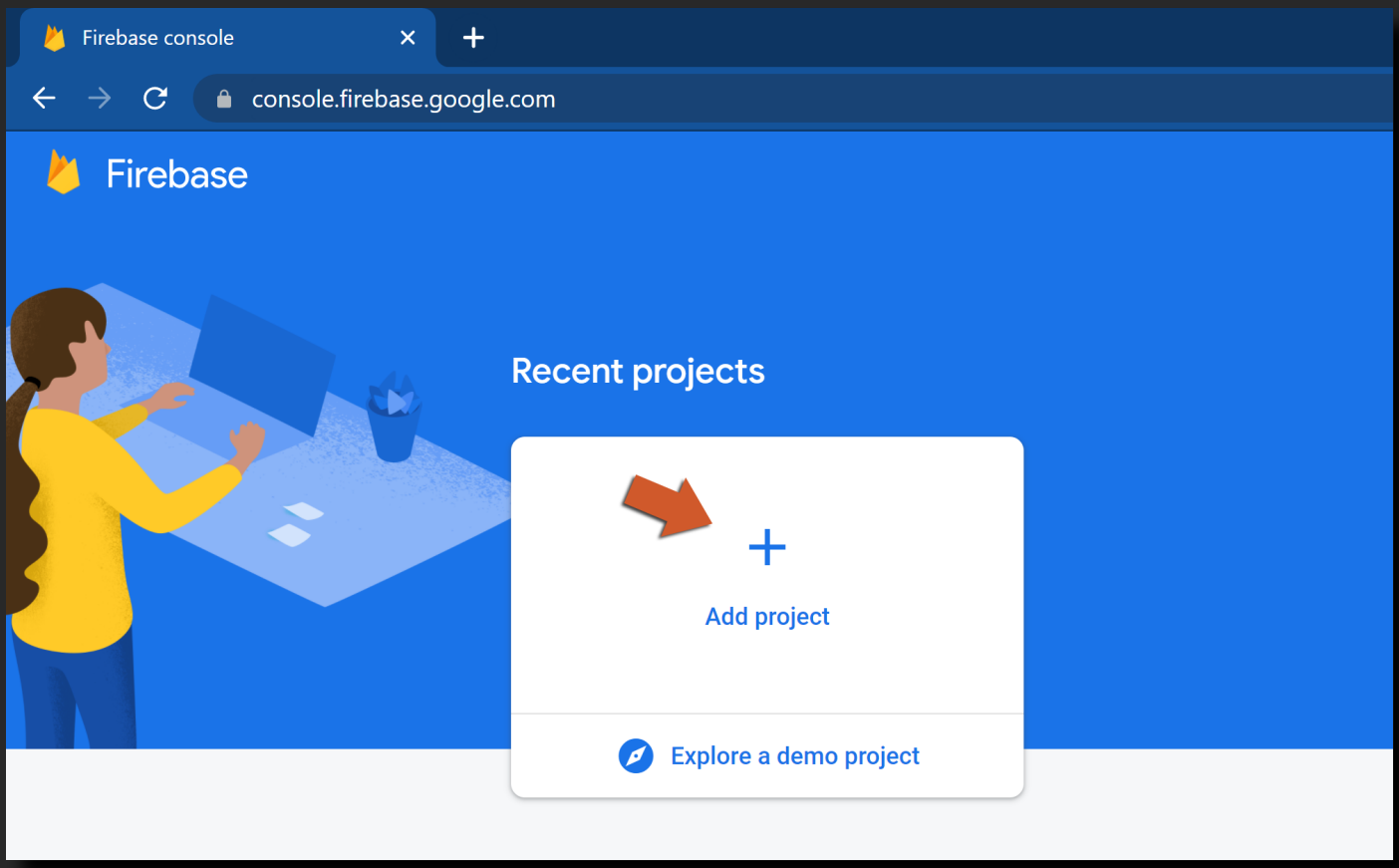
- FirebaseCloudMessaging_DeleteToken
- FirebaseCloudMessaging_GetToken
- FirebaseCloudMessaging_IsAutoInitEnabled
- FirebaseCloudMessaging_SetAutoInitEnabled
- FirebaseCloudMessaging_SubscribeToTopic

- FirebaseCloudMessaging_UnsubscribeFromTopic

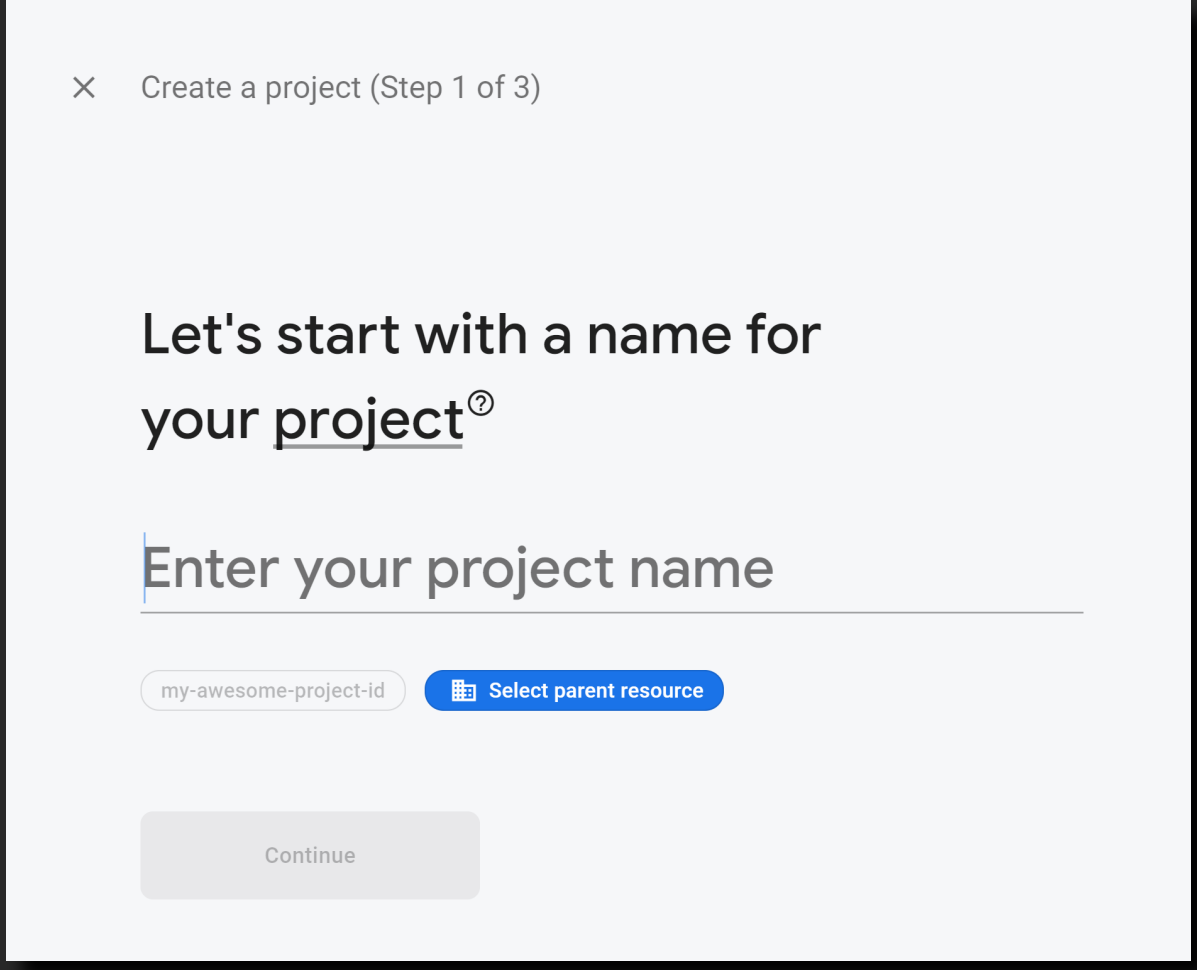
Create Project

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

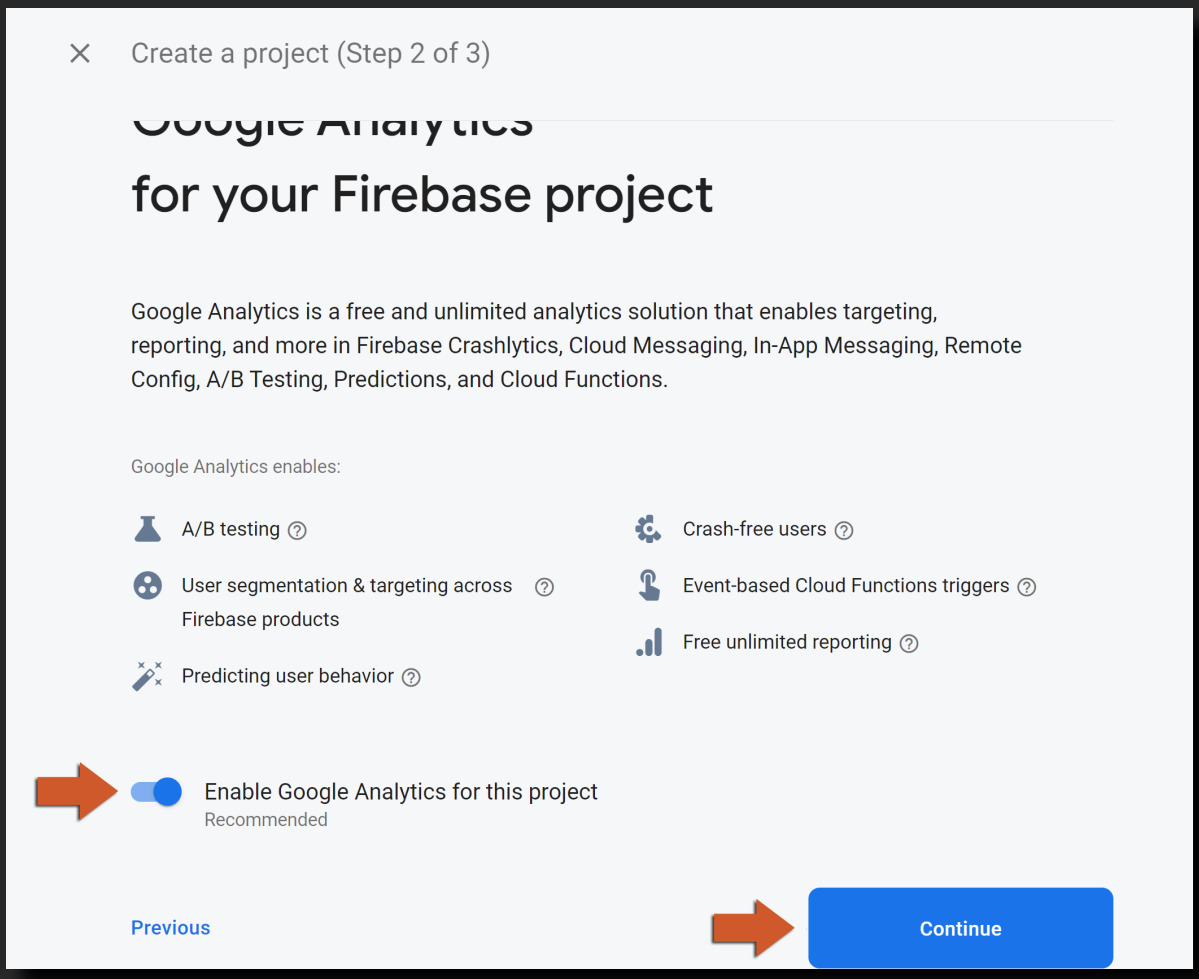
1. Go to the **Firebase Console** web site.
2. Click on **Add Project** to create your new project.



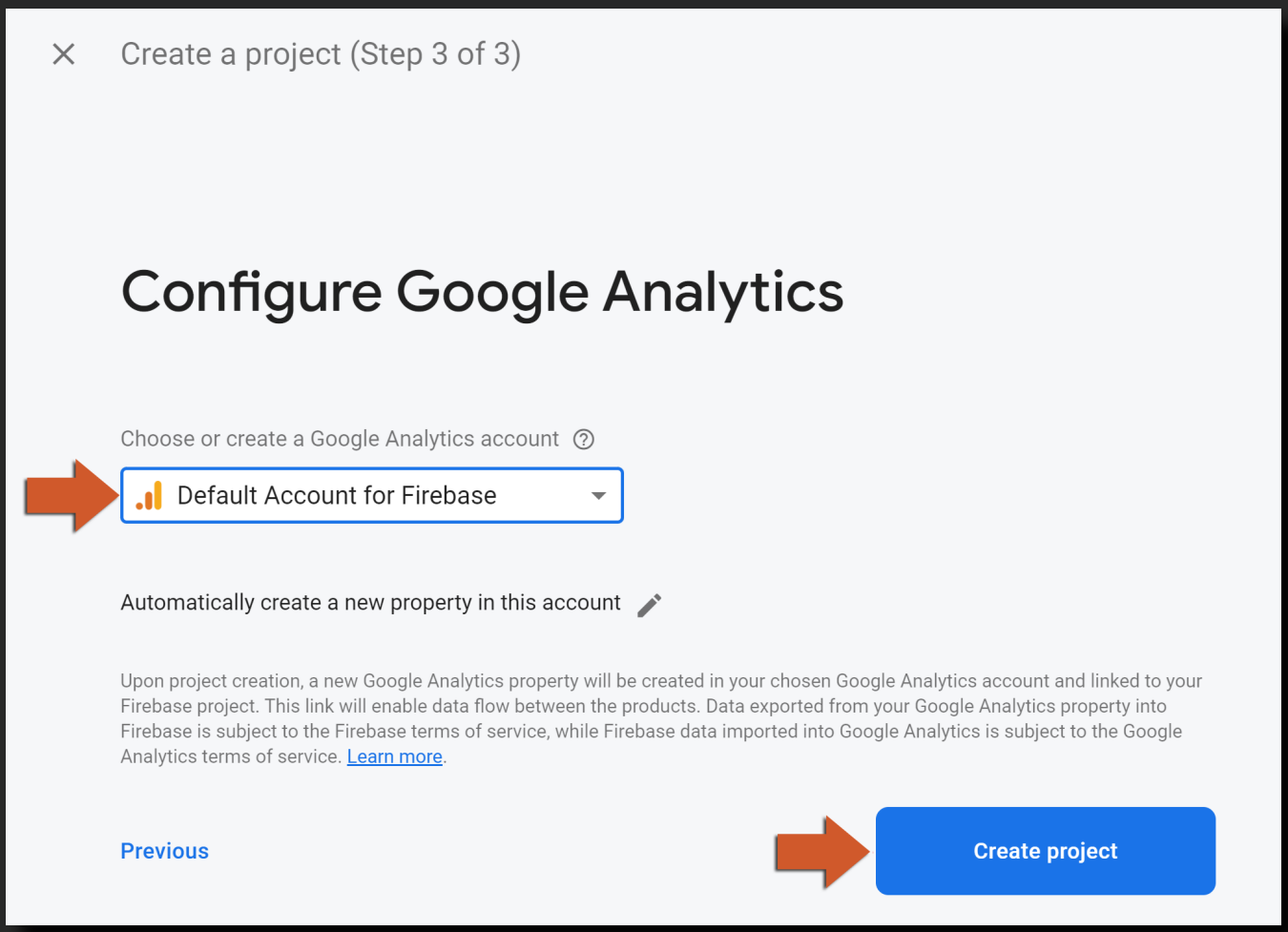
3. Enter a name for your project and click on the **Continue** button.



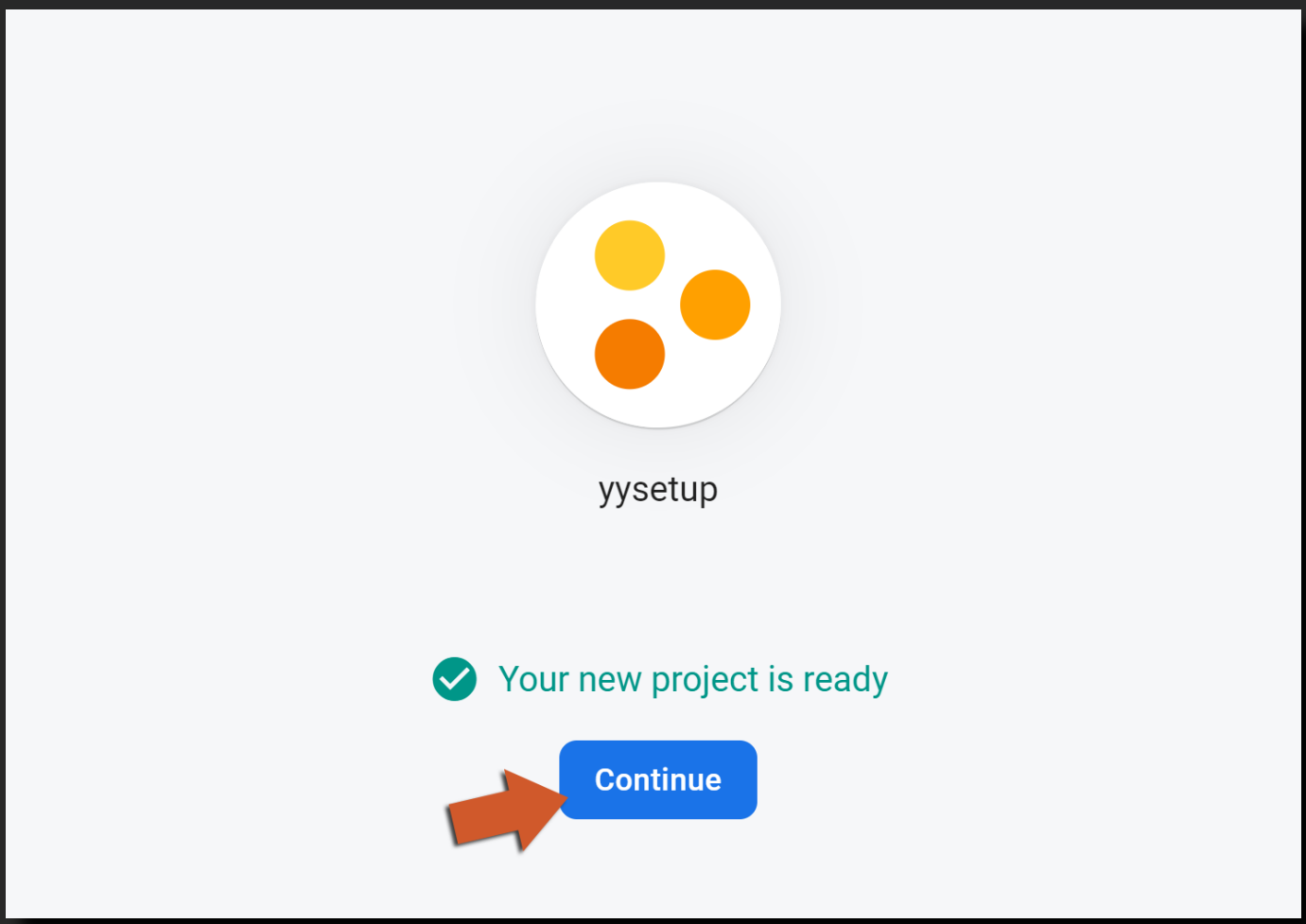
4. On the next page, make sure that **Enable Google Analytics for this project** is enabled and then click the **Continue** button:



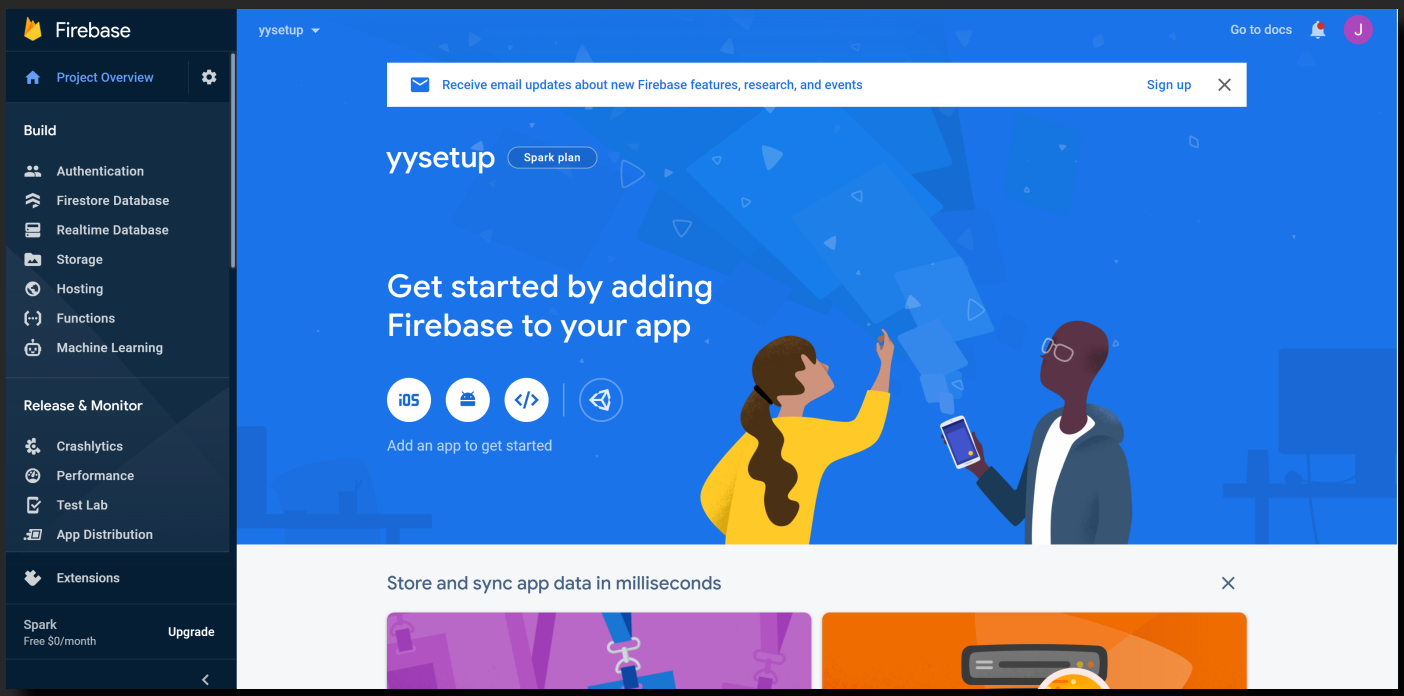
5. Select your account and click the **Create project** button:



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

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):

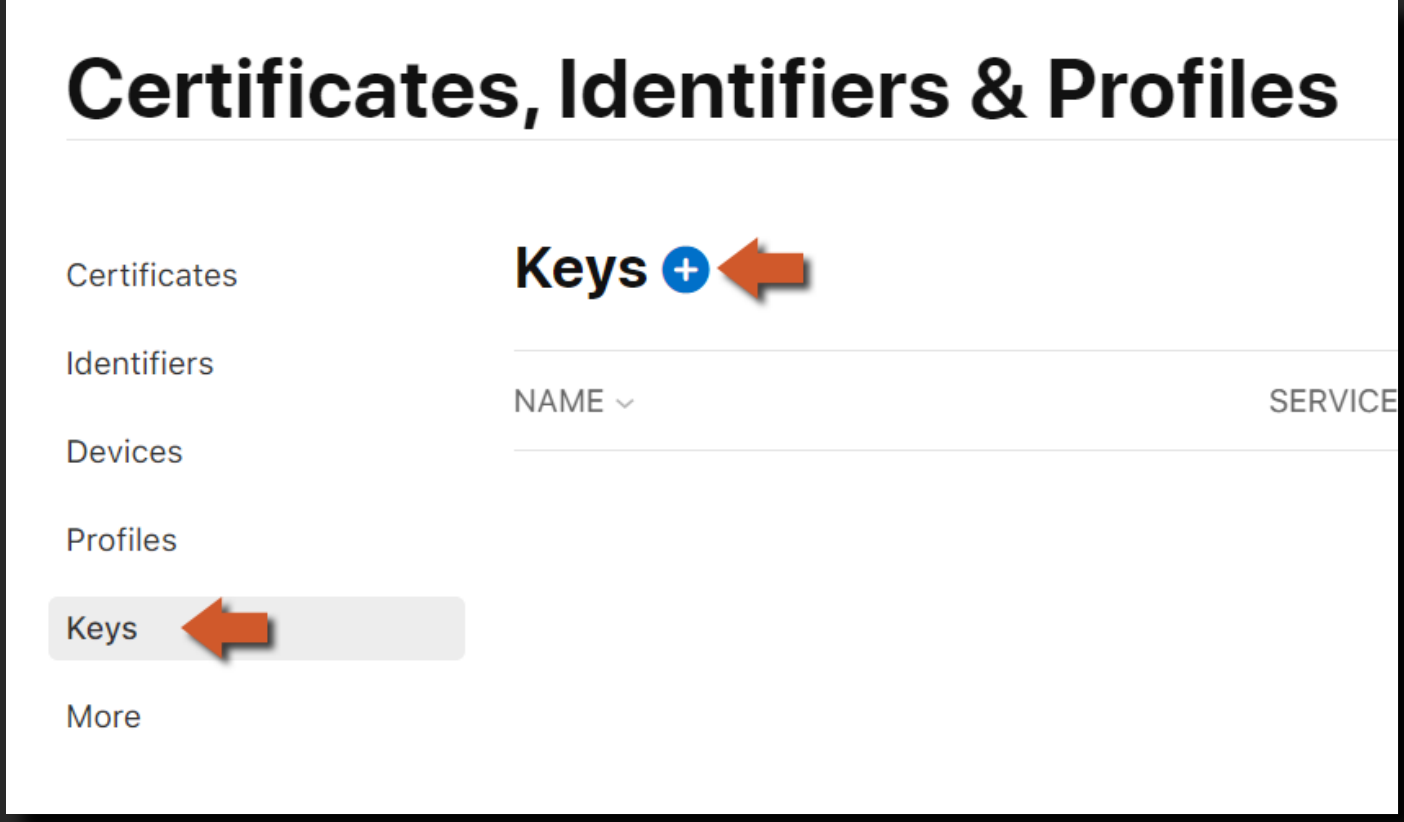
- Android Setup
- iOS Setup

Additional steps for iOS

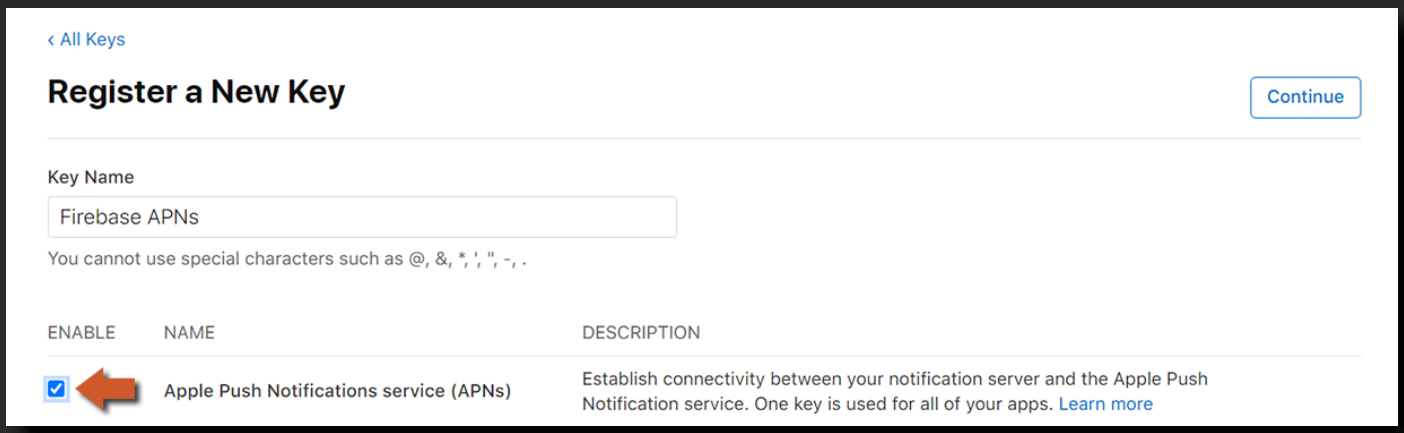
On iOS you will need to retrieve a P8 certificate and upload it to your Firebase project to enable sending push notifications through APNs:

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

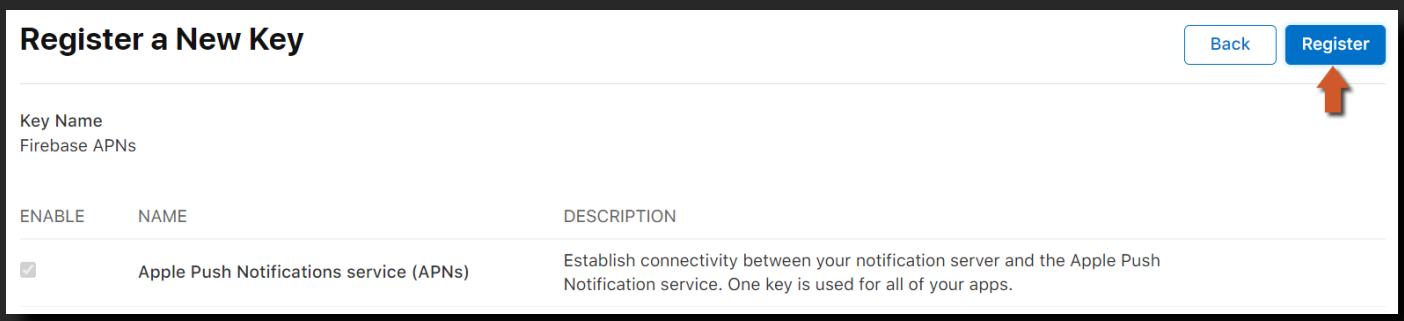
- Select "**Keys**" from the menu on the left, and create a new key by clicking on the plus sign.



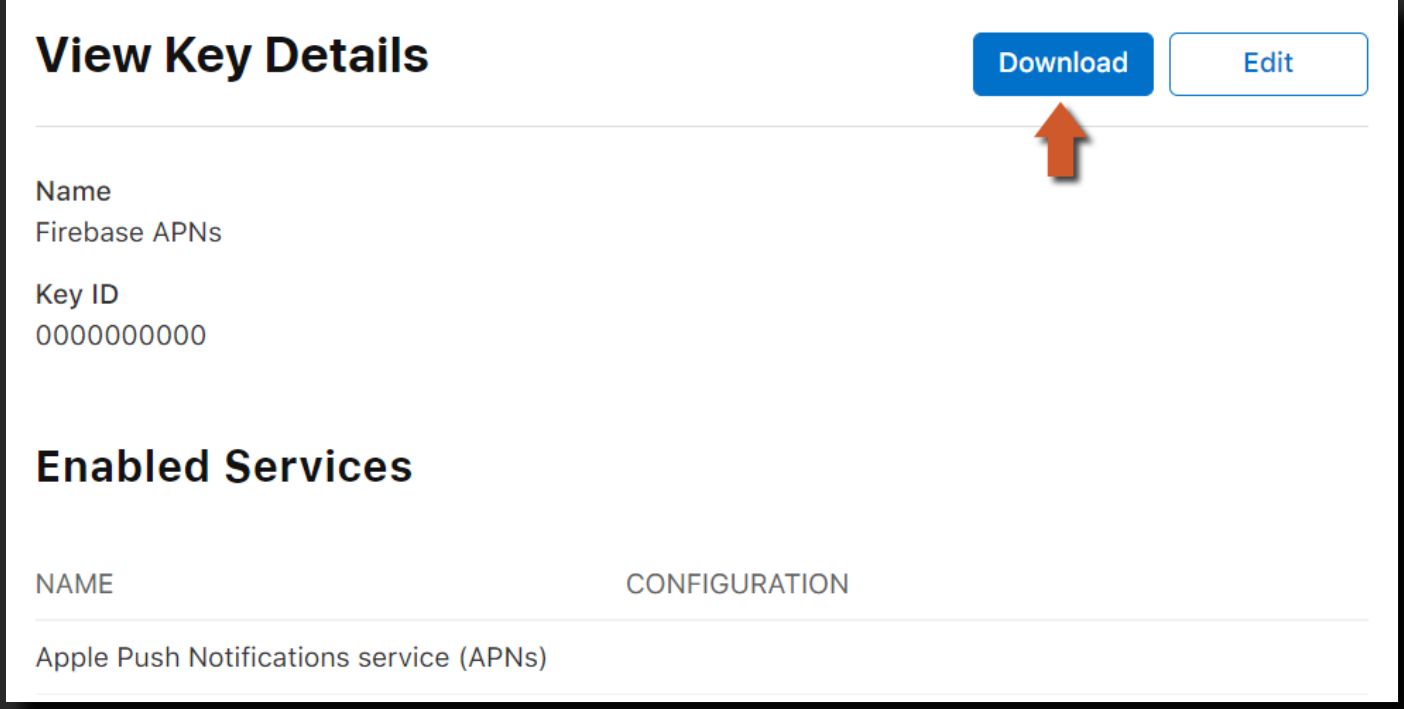
- Enter a **name** for the key, enable **Apple Push Notifications service** (APNs) and click on **Continue**.



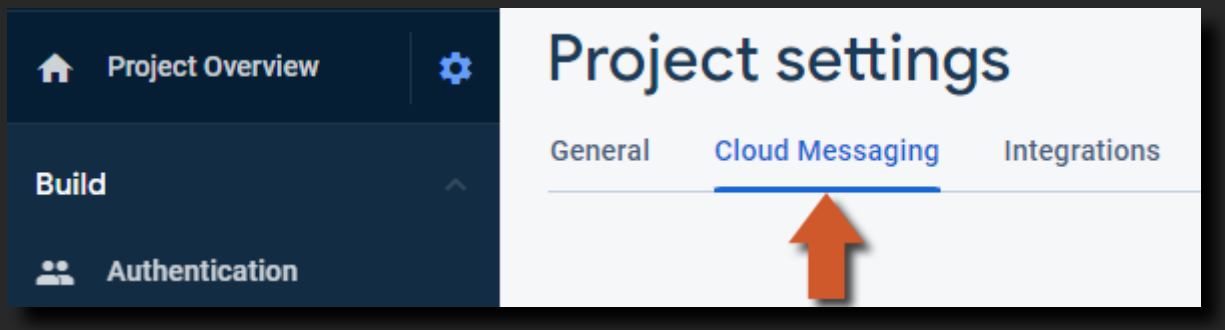
- On the next page, confirm the key details and click on **Register**.



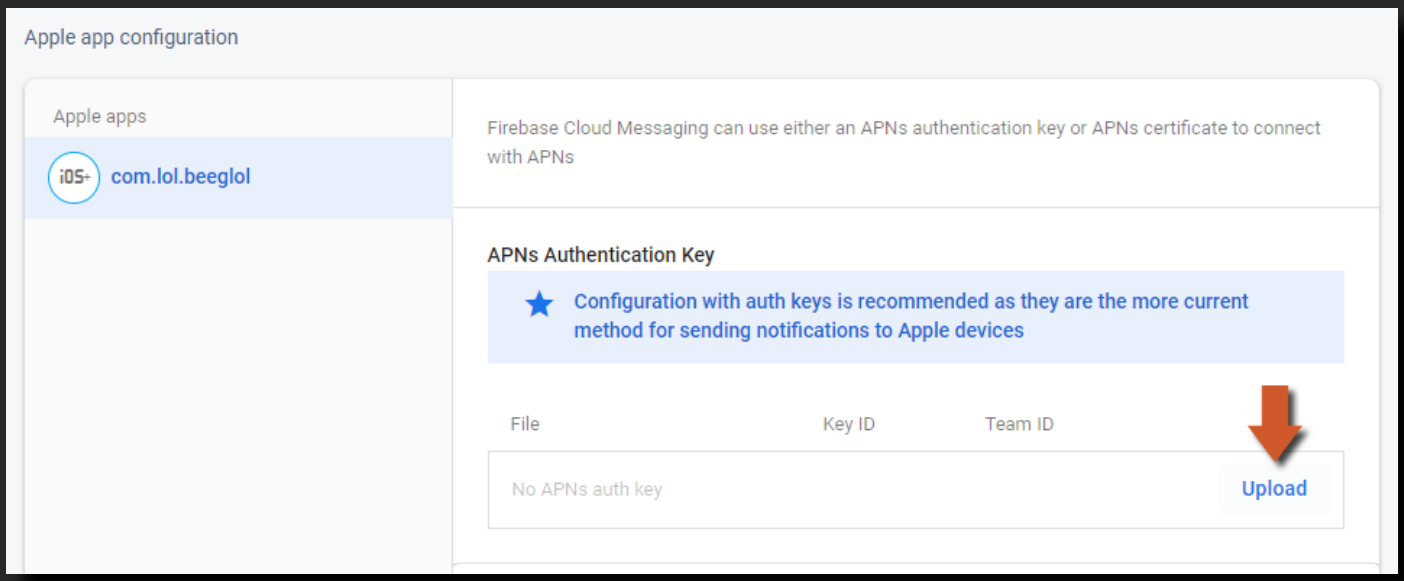
- **Note** the information given here (key ID) and **download** the key as you will not be able to see this screen again.



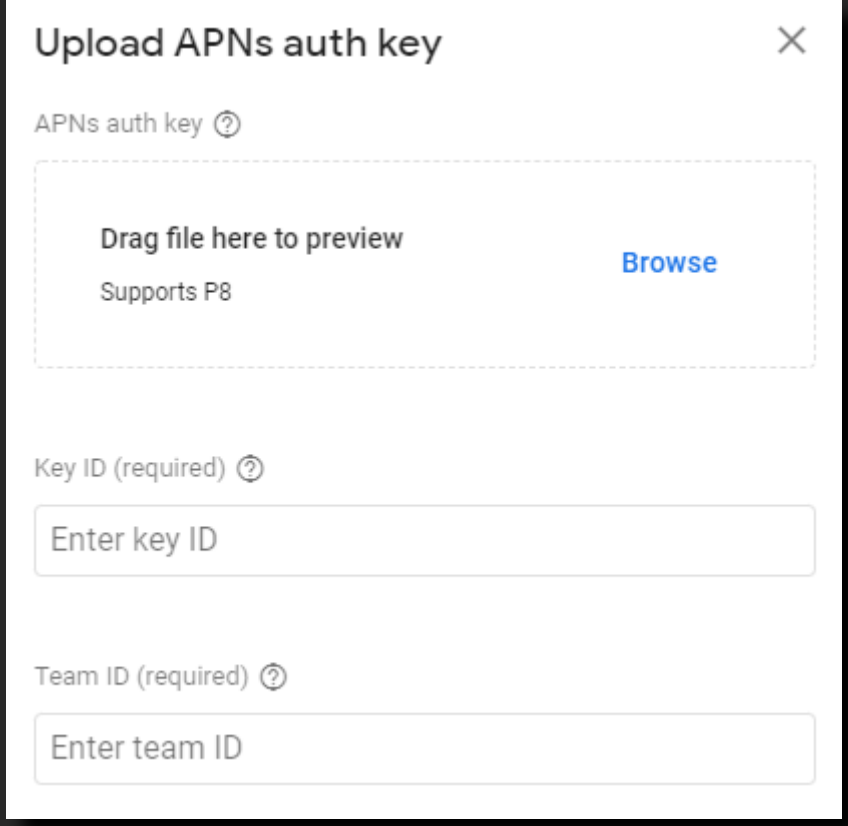
- Go to the dashboard for your Firebase project and open the **Project Settings**. Here, open the **Cloud Messaging** tab.



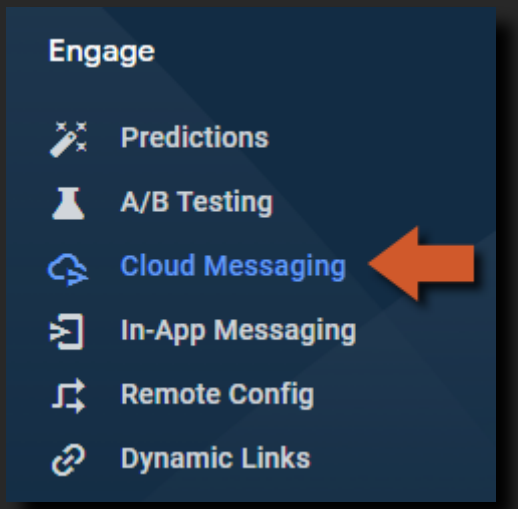
- Select your iOS application, and under "**APNs Authentication Key**", press **Upload** to upload your key.



- 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.

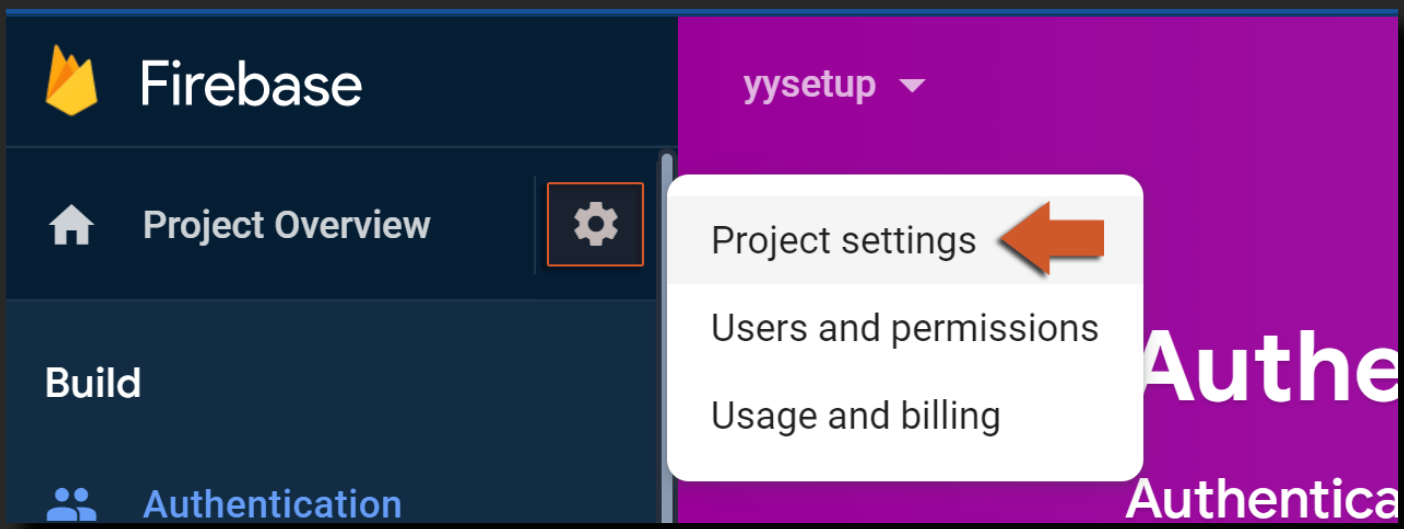


Android Setup

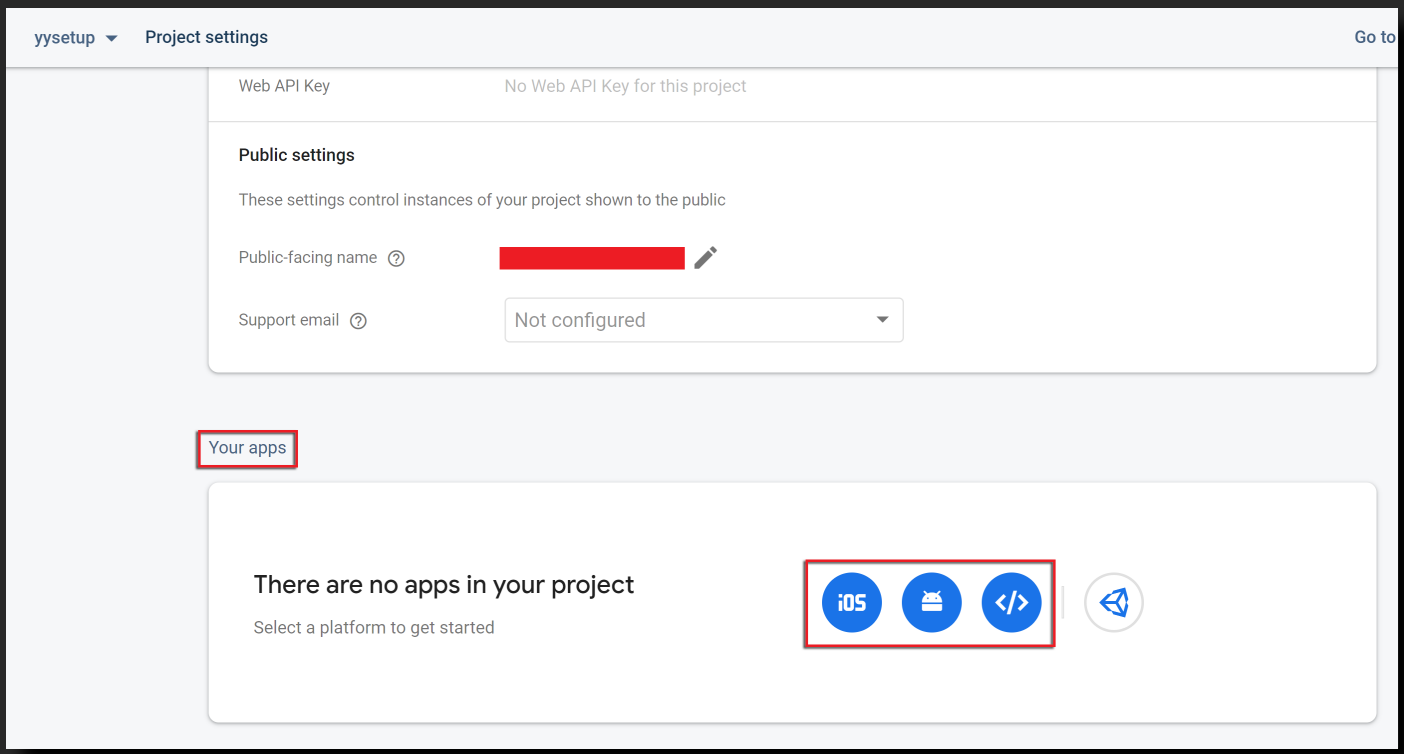
This setup is necessary for all the Firebase modules using Android and needs to be done once per project, and basically involves importing the **google-services.json** file into your project.

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

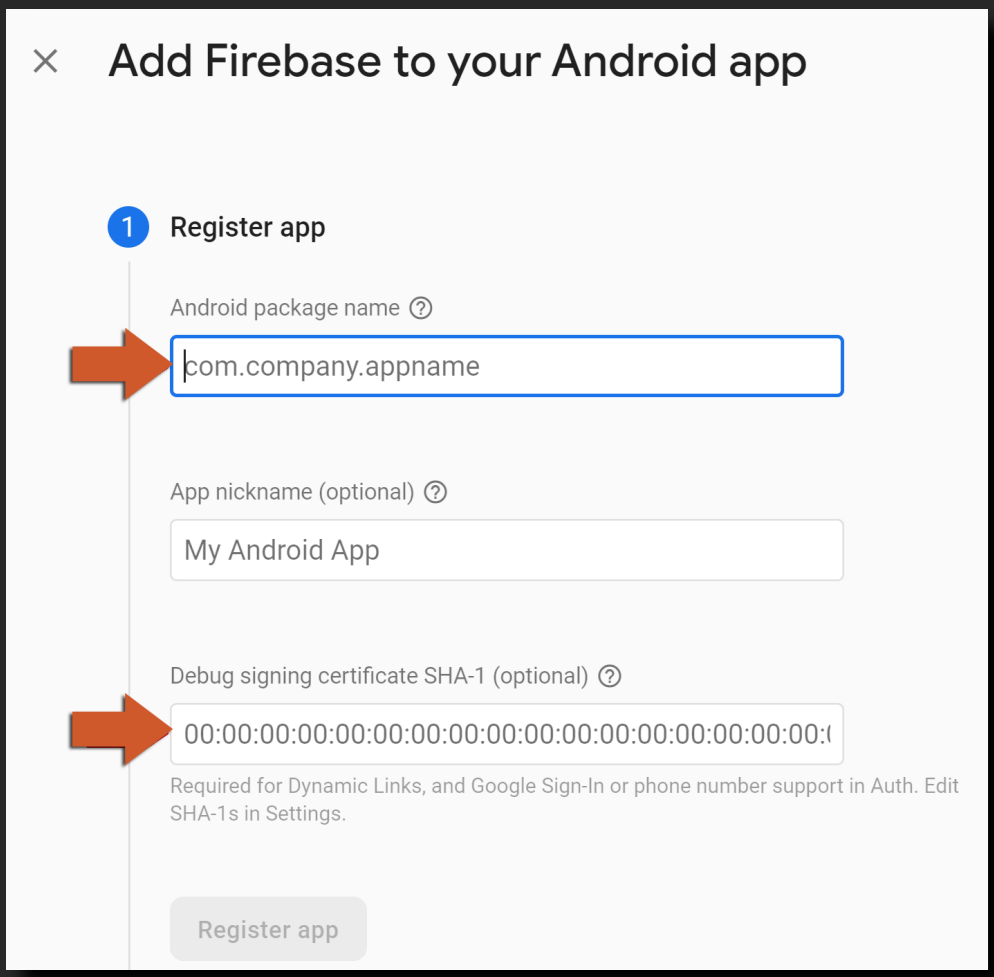
1. Click the **Settings** icon (next to Project Overview) and select **Project settings:**



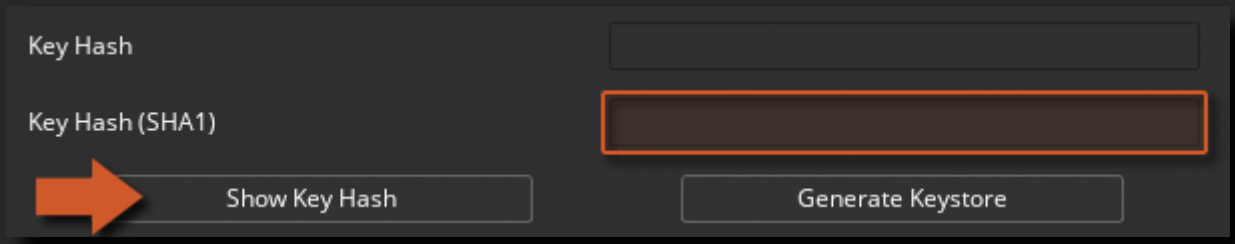
2. Now go to the **Your apps** section and click on the **Android** button:



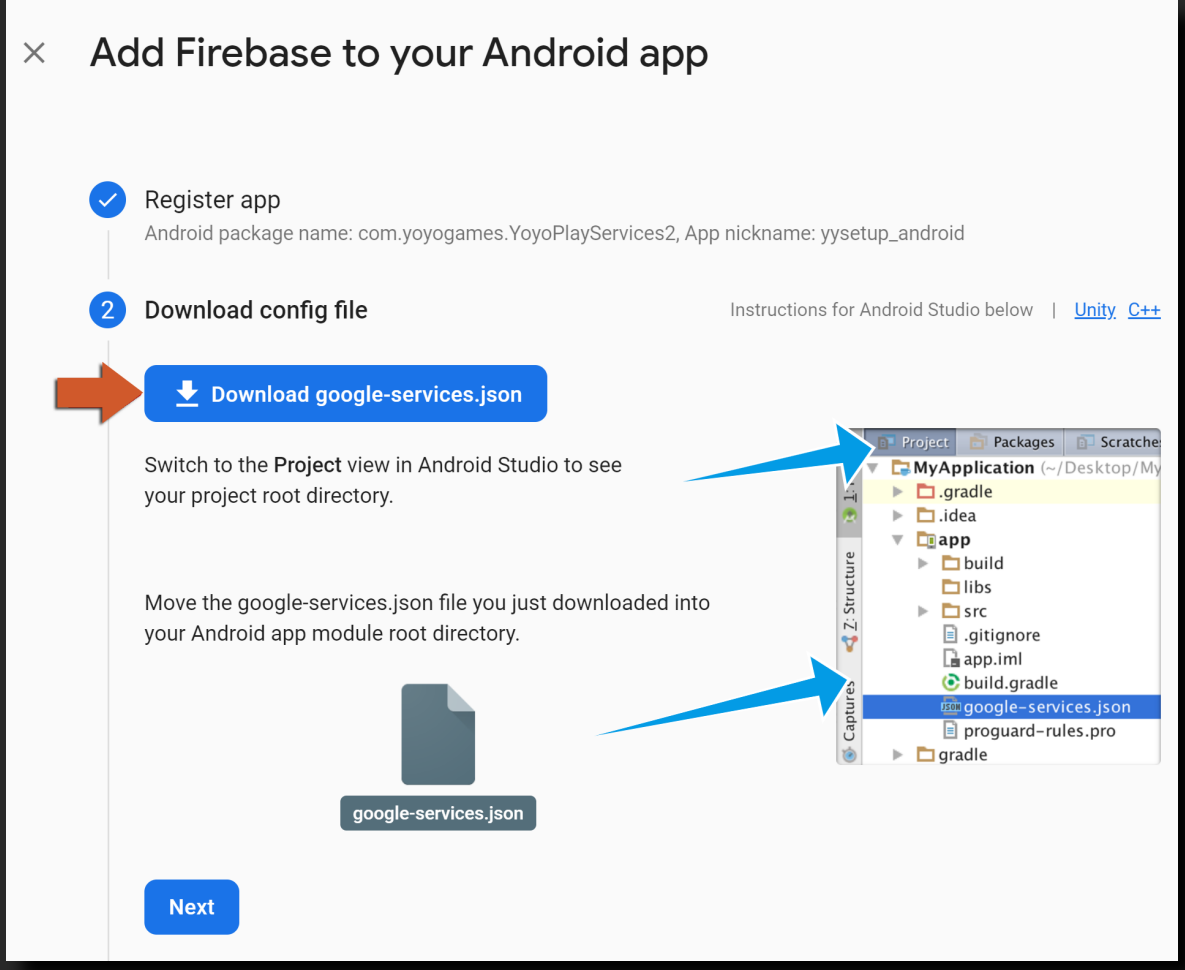
3. On this screen you need enter your **Package name** (required), **App nickname** (optional) and **Debug signing certificate SHA-1** (required if you are using Firebase Authentication).



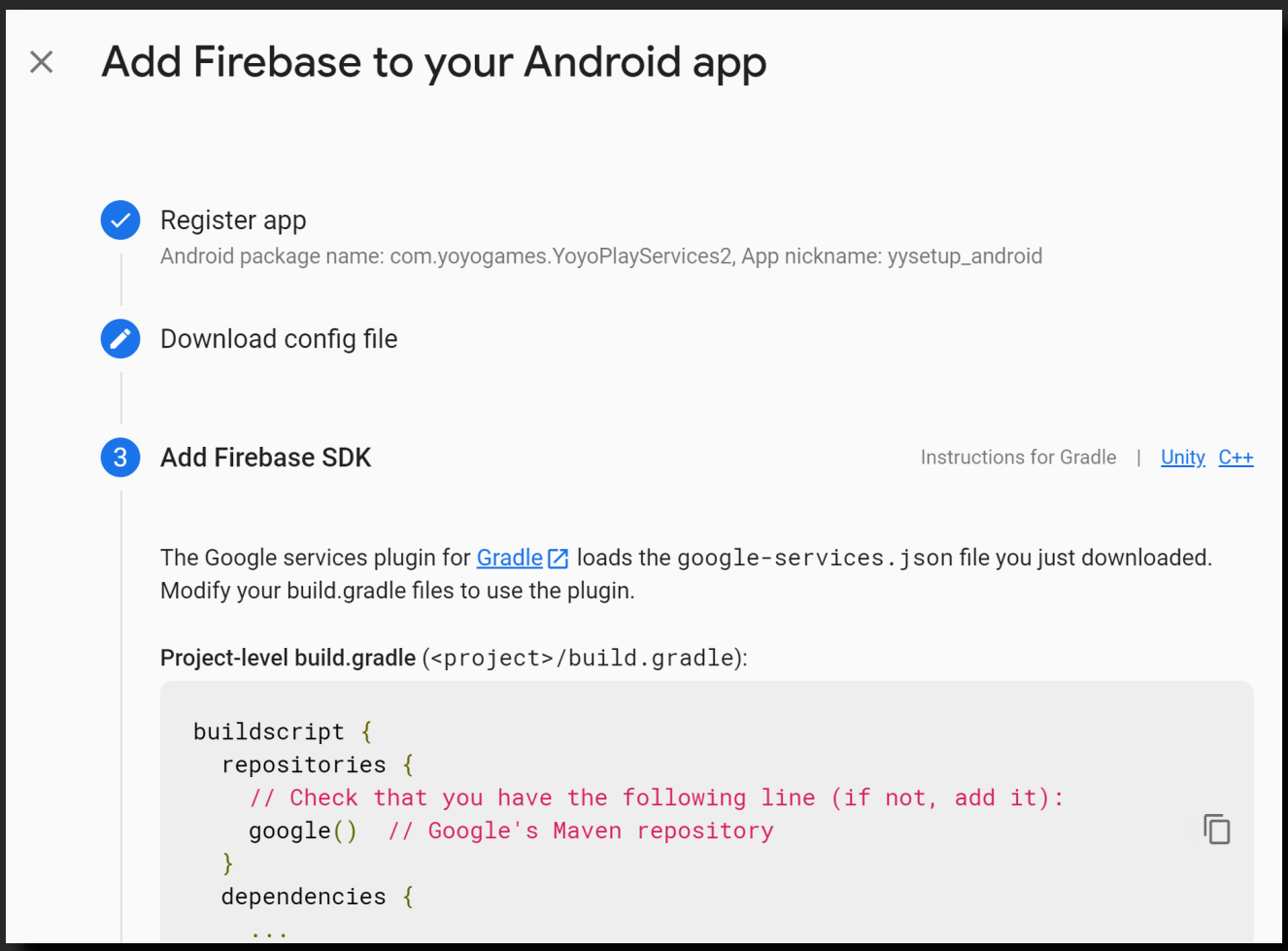
You can get your package name from the **Android Game Options**, and your **Debug signing certificate SHA-1** from the **Android Preferences** (under Keystore):



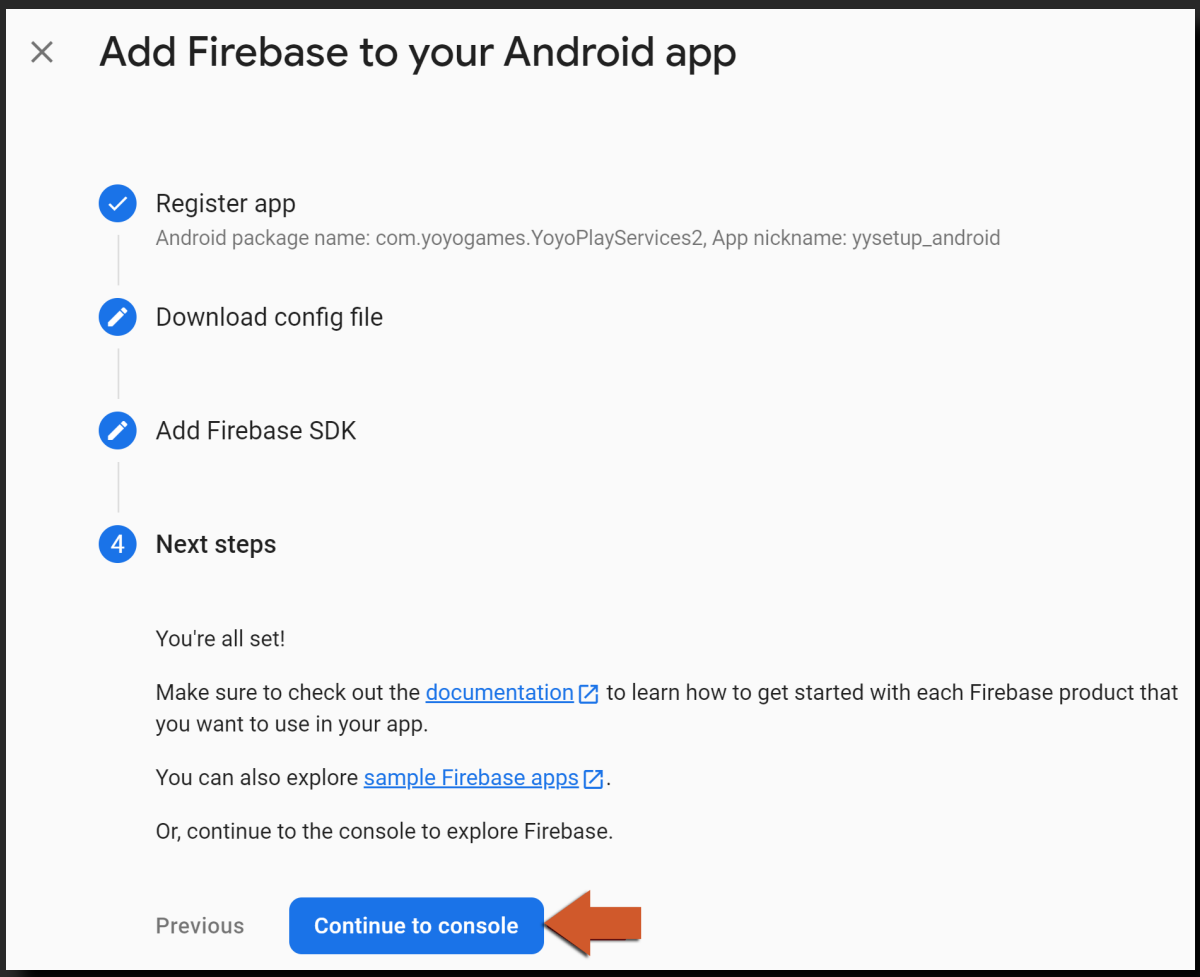
4. Click on **Download google-services.json** (make sure to save this file as we will need it in subsequent steps).



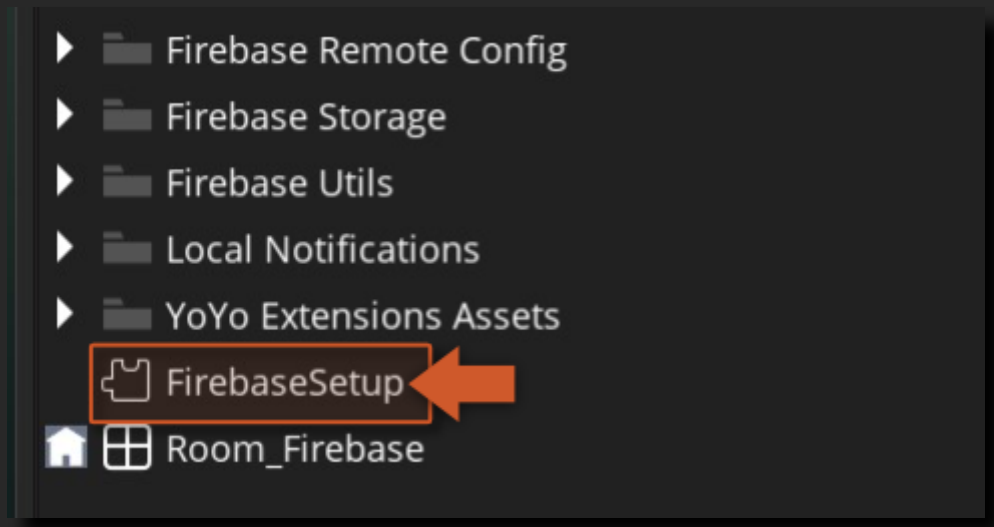
5. Ignore this screen, as this is already done in the extension.



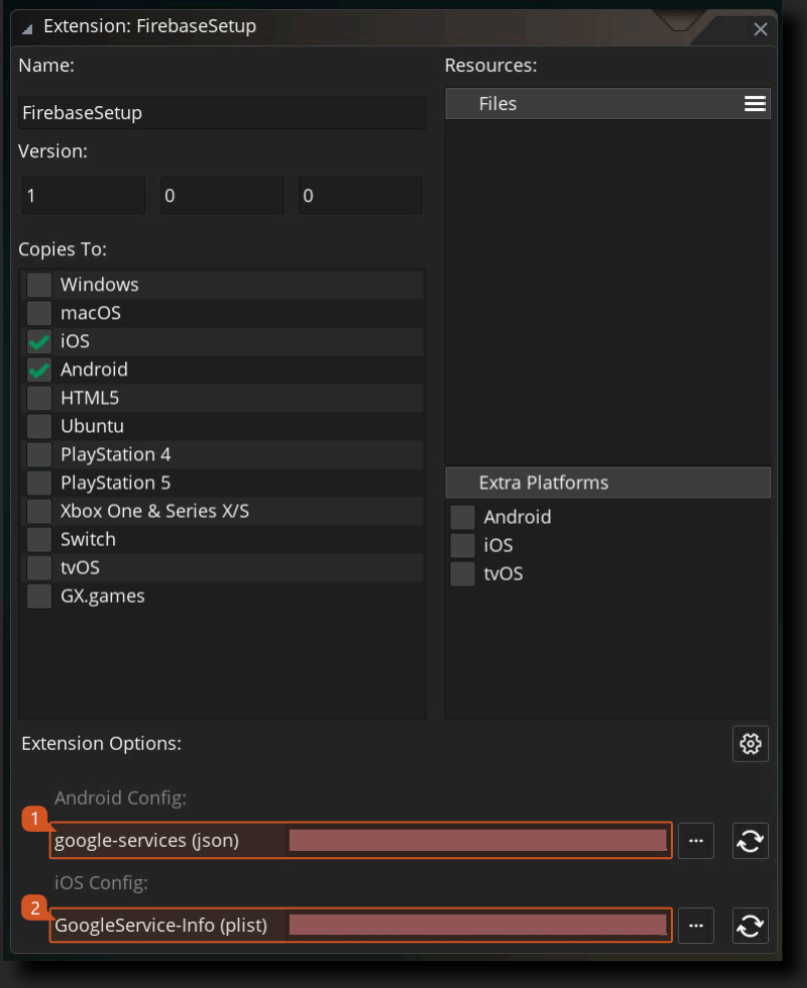
6. Click on the **Continue to console** button.



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).



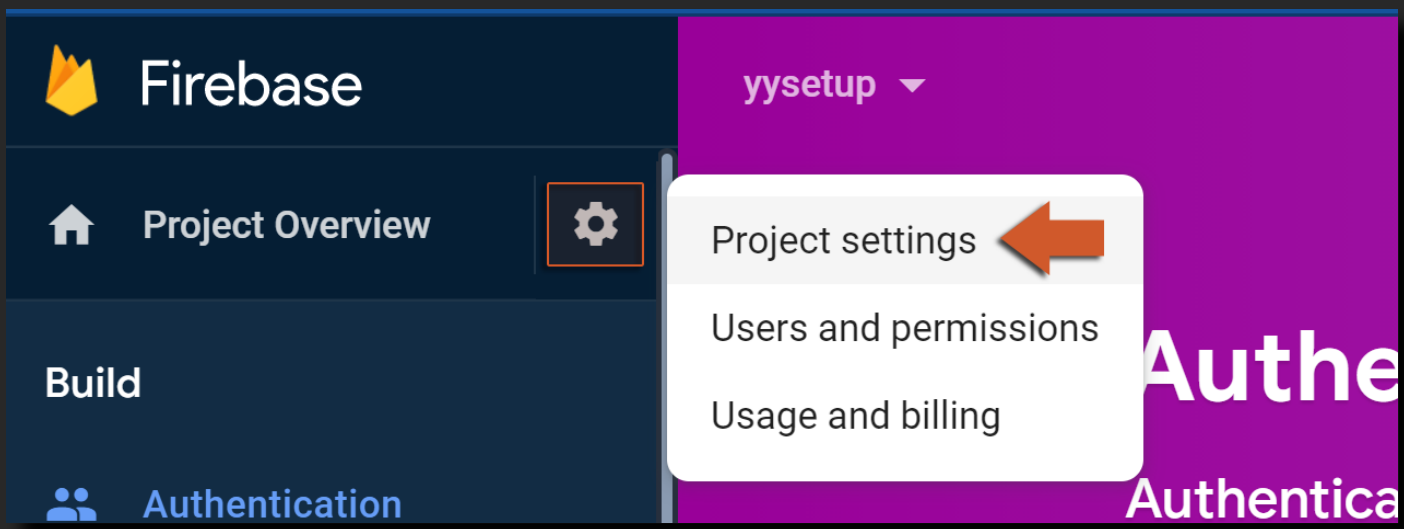
9. You have now finished the main setup for all Firebase Android modules!

iOS Setup

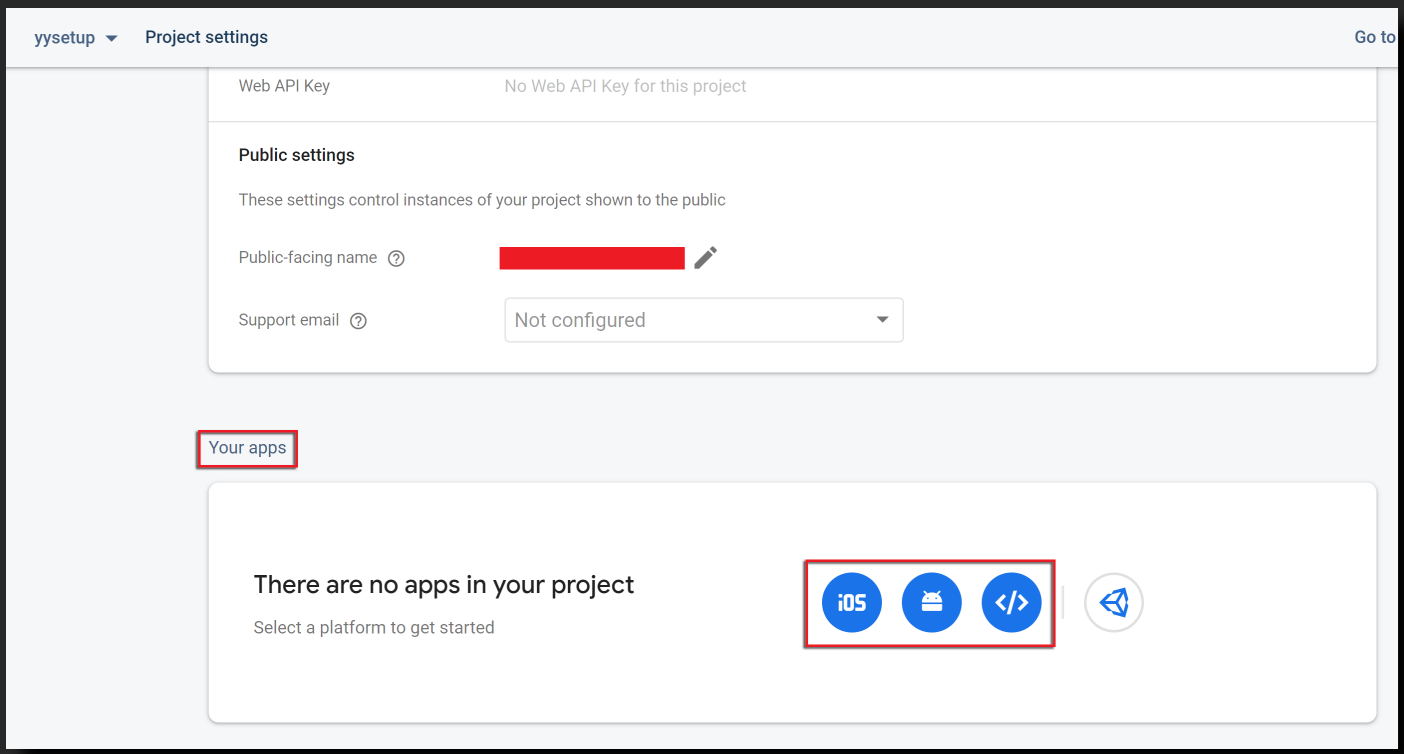
This setup is necessary for all the Firebase modules using iOS and needs to be done once per project, and basically involves importing the **GoogleServices-Info.plist** file into your project.

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

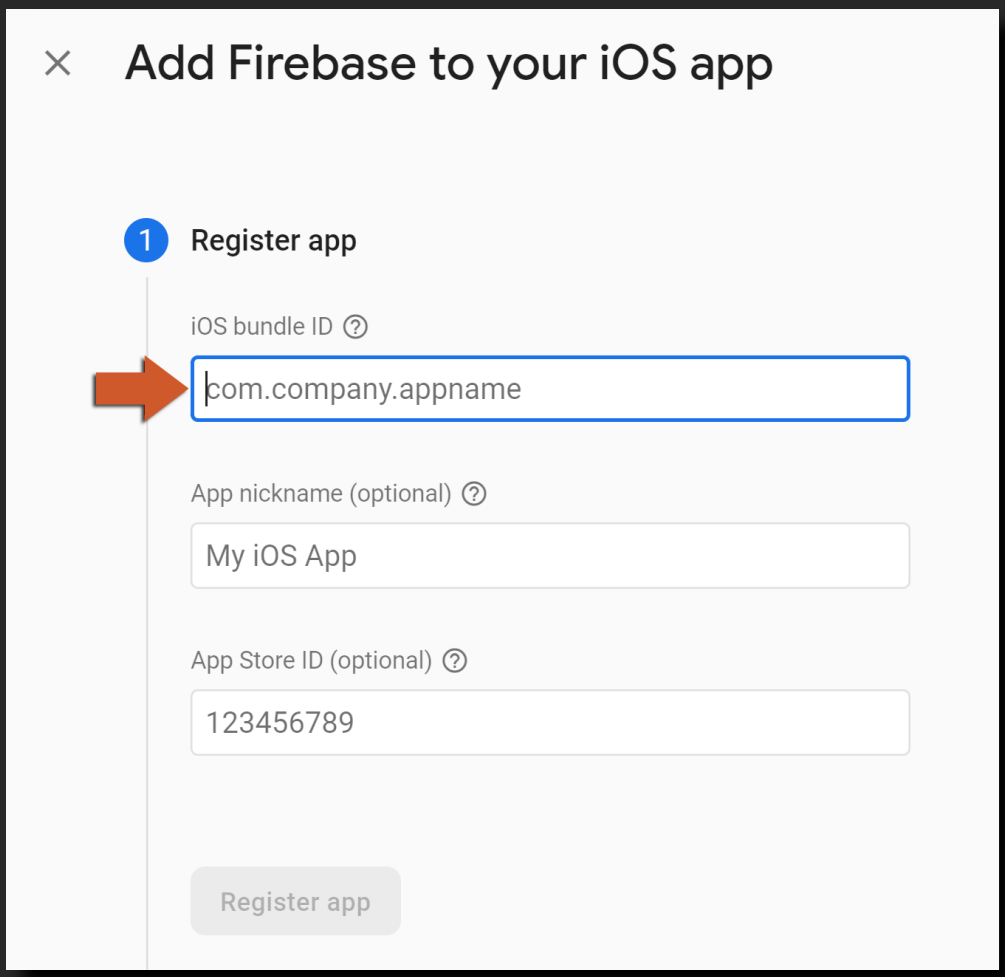
1. Click the **Settings** icon (next to Project Overview) and select **Project settings:**



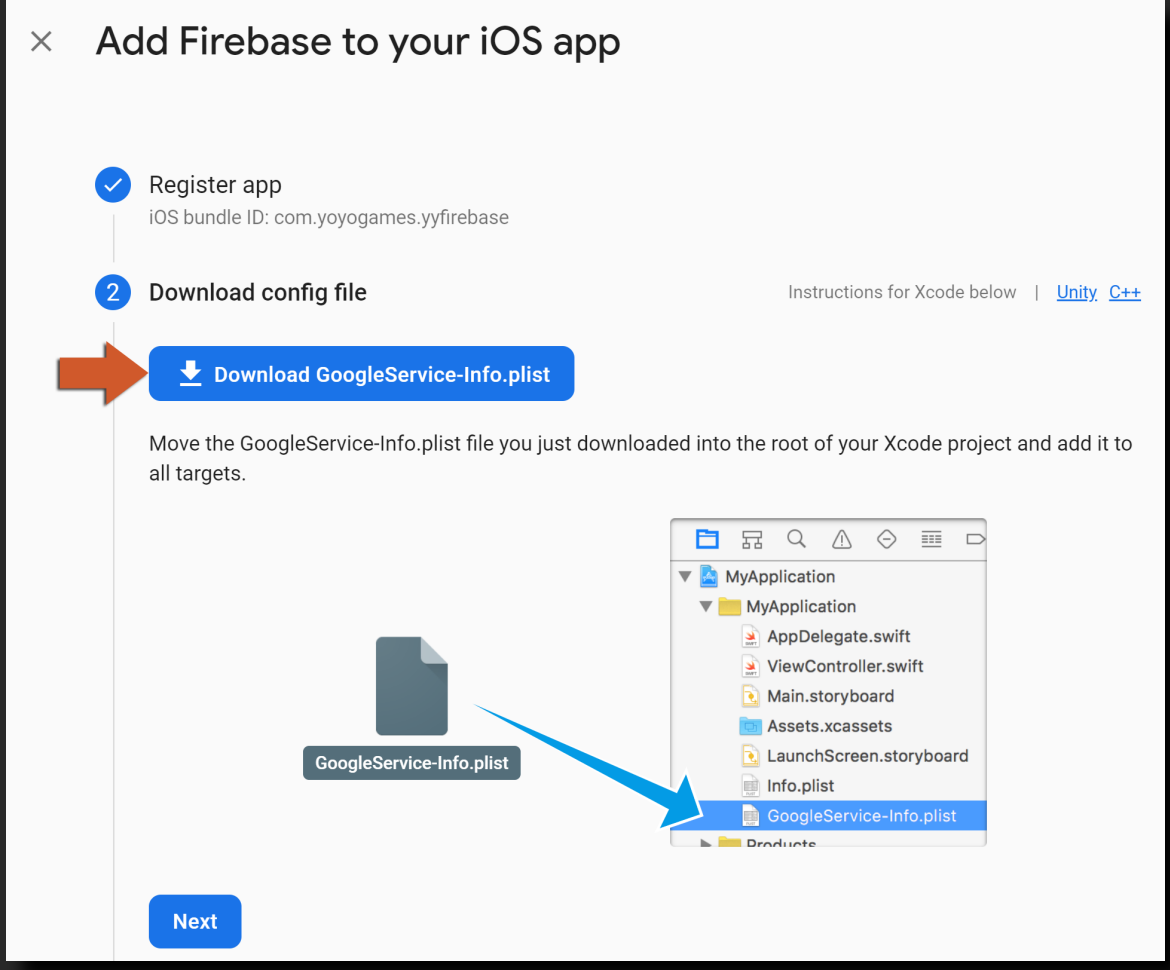
2. Now go to the **Your apps** section and click on the **iOS** button:



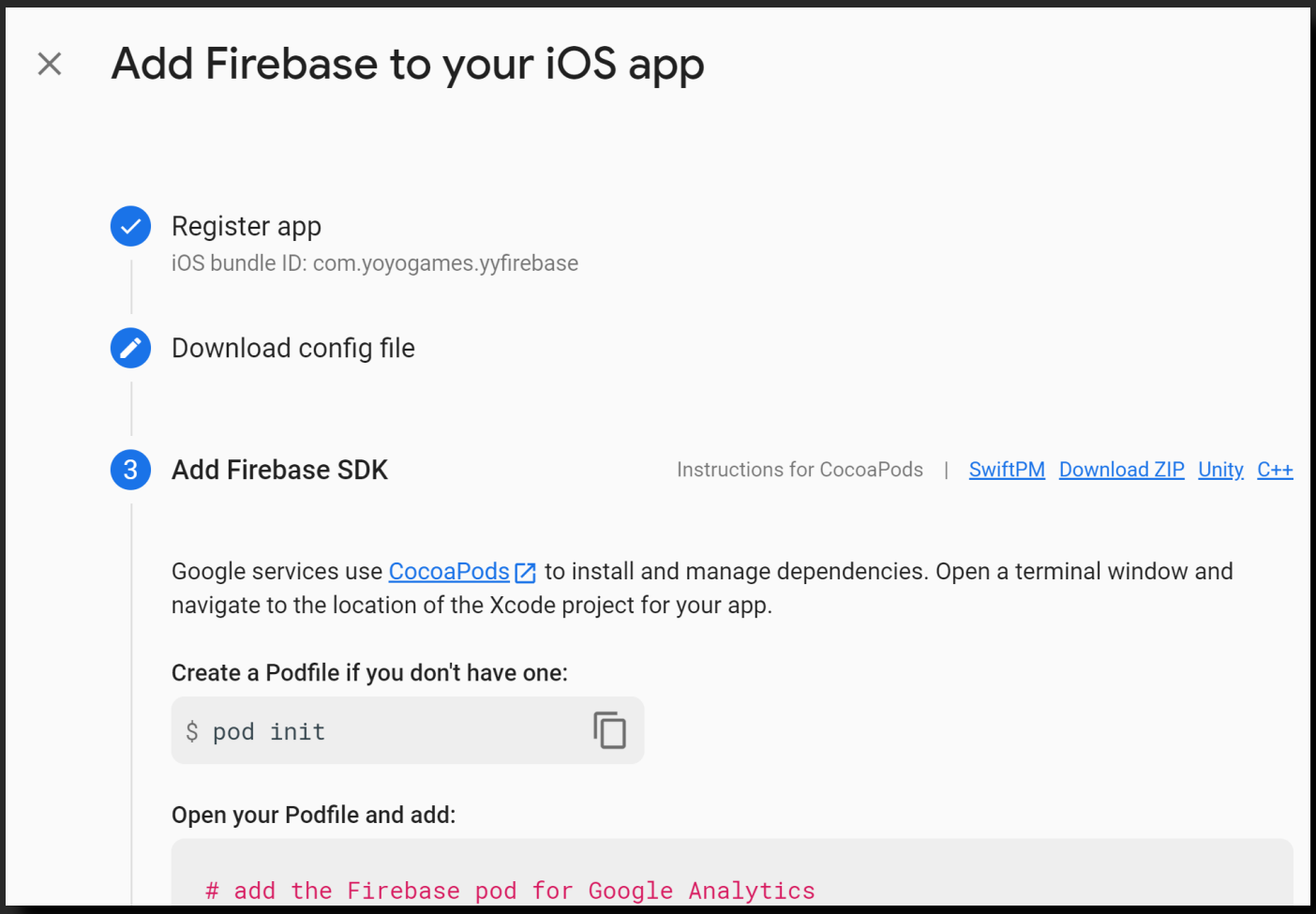
3. Fill the form with your **iOS Bundle ID**, **App nickname** and **AppStore ID** (last two are optional).



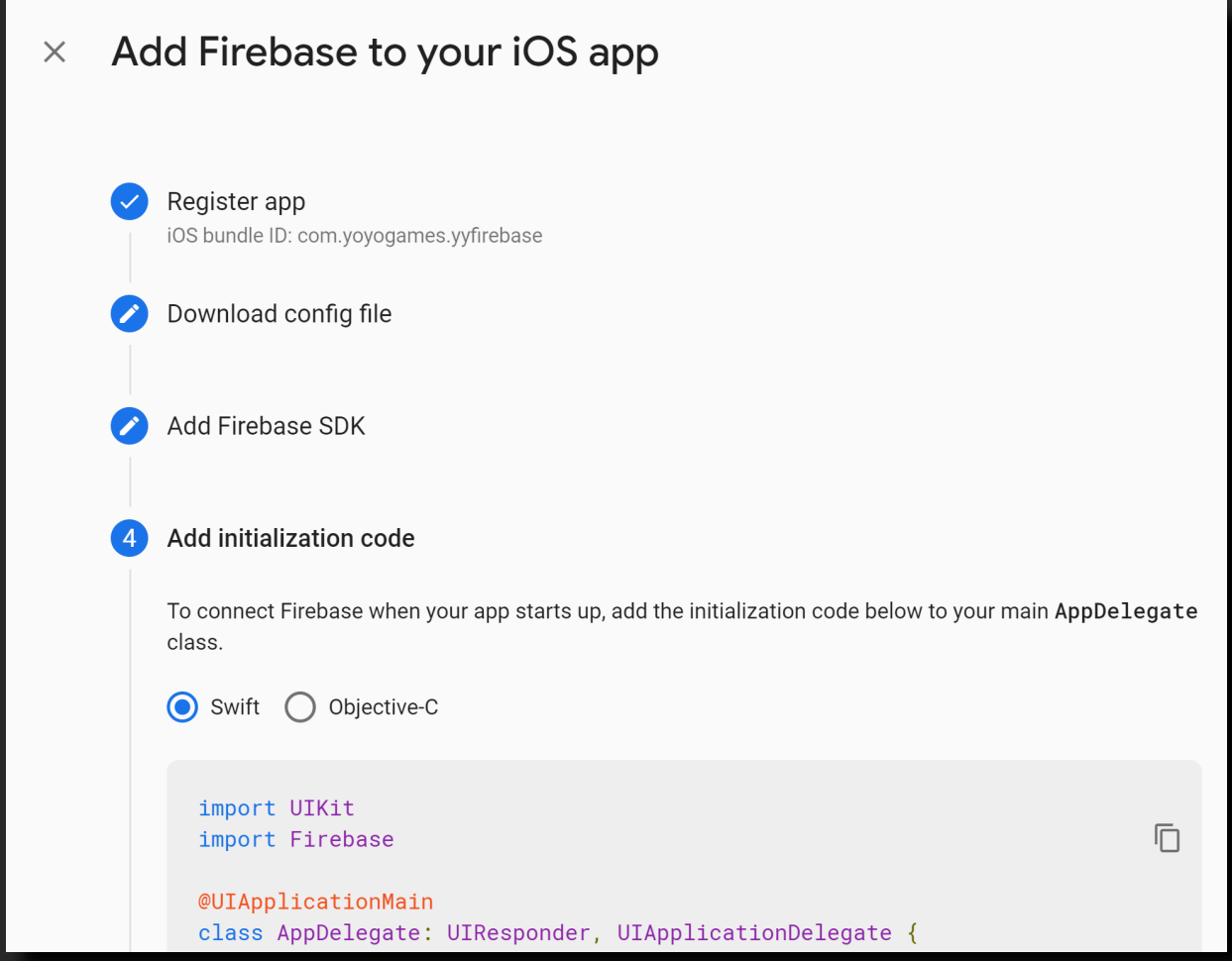
4. Click on **Download GoogleService-info.plist** (make sure to save this file as we will need it in subsequent steps).



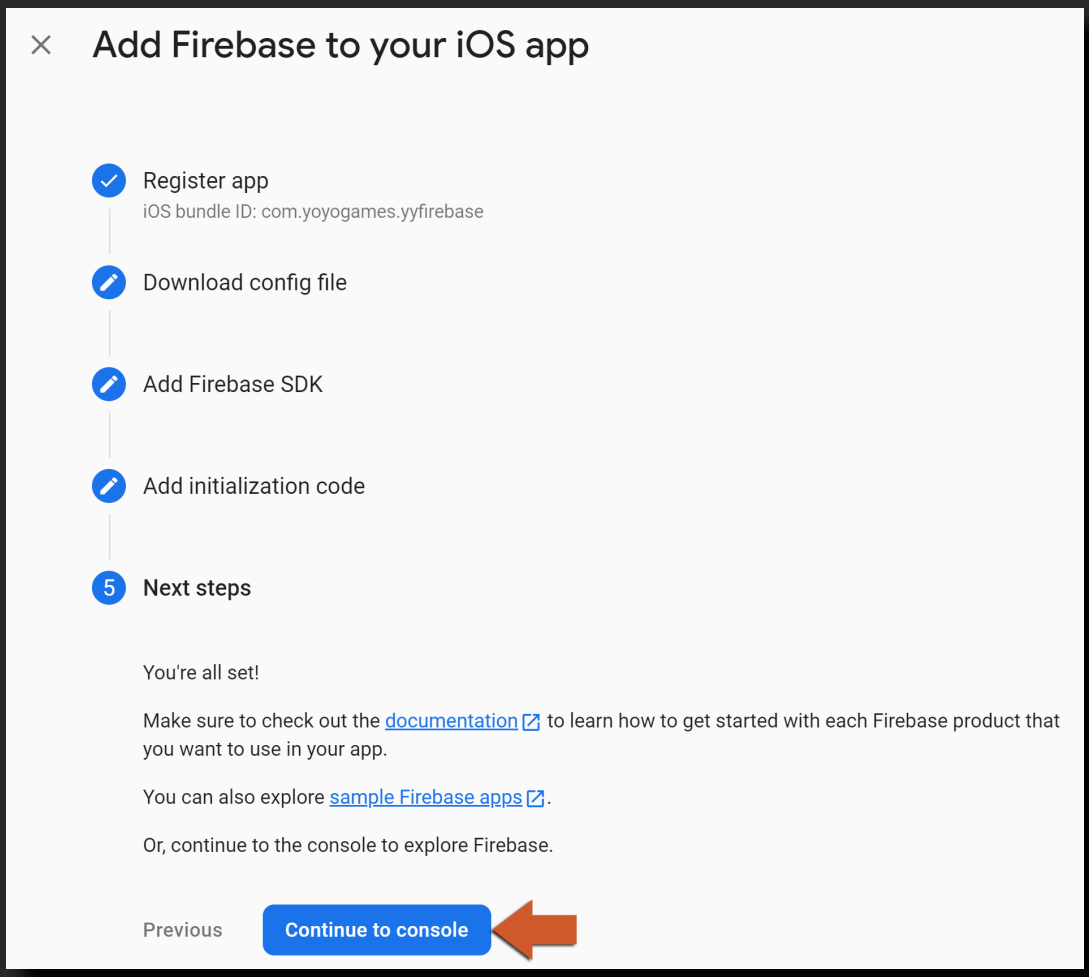
5. Ignore this screen, as this is already done in the extension.



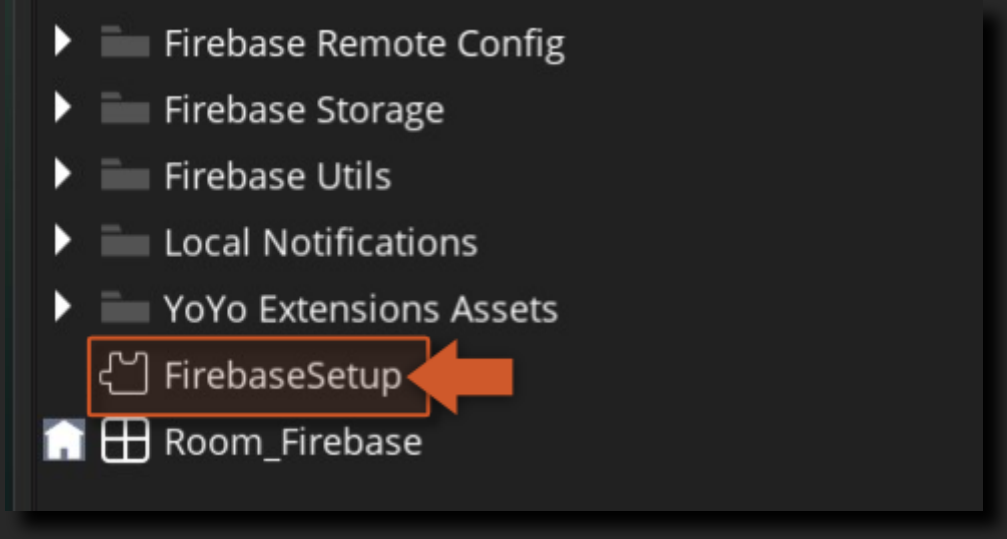
6. Ignore this screen as well, as this is also done in the extension.



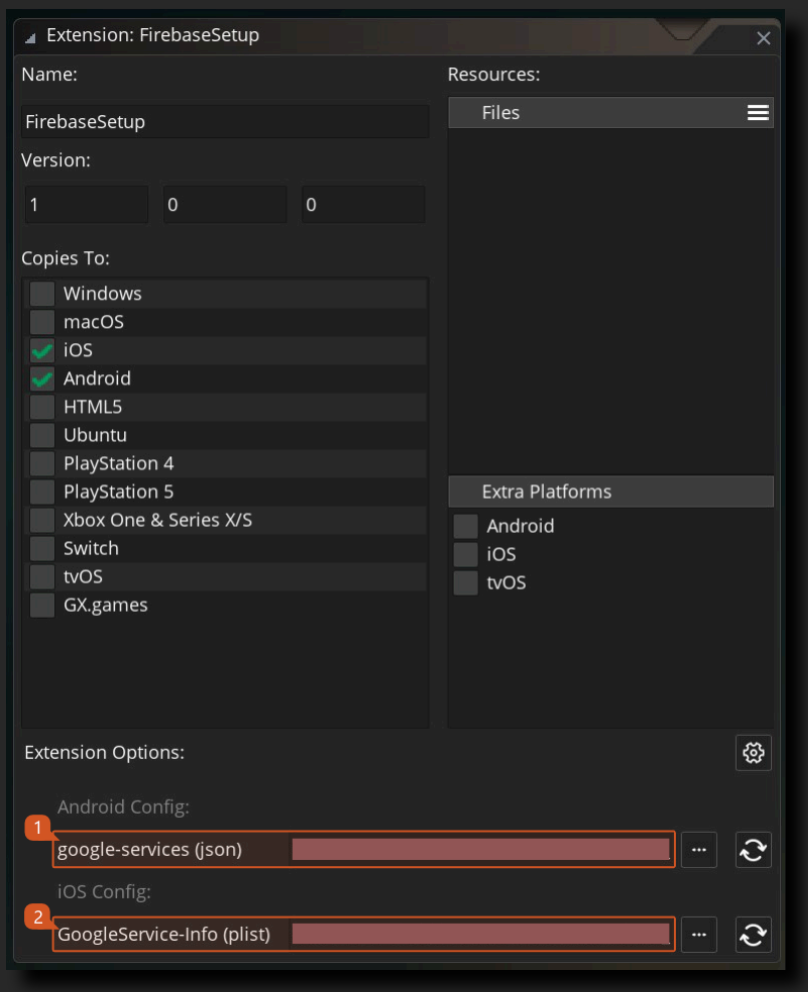
7. Click on the **Continue to console** button:



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!

FirebaseCloudMessaging_DeleteToken

Deletes the FCM registration token for this Firebase project. Note that if auto-init is enabled, a new token will be generated the next time the app is started. Disable auto-init (using the function FirebaseCloudMessaging_SetAutoInitEnabled) to avoid this behavior.

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

Syntax:

```
FirebaseCloudMessaging_DeleteToken();
```

Returns:

```
N/A
```

Triggers:

```
Asynchronous Social Event
```

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

Example:

```
FirebaseCloudMessaging_DeleteToken()
```

In the code above we request for the FCM token to be deleted. The function callback can be caught inside an Async Social event.

```
if(async_load[?"type"] == "FirebaseCloudMessaging_DeleteToken")
{
    if(async_load[?"success"])
    {
```

```
        show_debug_message(FCM token deleted")
    }
}
```

The code above matches the response against the correct event **type**, and provides a success message if **success** is true.

FirebaseCloudMessaging_GetToken

Requests the FCM registration token for this Firebase project. This sends information about the application and the device where it's running to the Firebase backend. See FirebaseCloudMessaging_DeleteToken for information on deleting the token.

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

Syntax:

```
FirebaseCloudMessaging_GetToken();
```

Returns:

```
N/A
```

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The constant `"FirebaseCloudMessaging_GetToken"`
success	boolean	Whether or not the function task succeeded.
value	string	The FCM registration token.

Example:

```
FirebaseCloudMessaging_GetToken()
```

In the code above we request for the current FCM token. The function callback can be caught inside an Async Social event.

```
if(async_load[?"type"] == "FirebaseCloudMessaging_GetToken")
{
    if(async_load[?"success"])
    {
        global.fcmToken = async_load[? "value"];
    }
}
```

The code above matches the response against the correct event **type**, and if the tasks succeeds it stores the token value into a global variable (`global.fcmToken`).

FirebaseCloudMessaging_IsAutoInitEnabled

Returns whether FCM auto-initialization is enabled or disabled.

Syntax:

```
FirebaseCloudMessaging_IsAutoInitEnabled()
```

Returns:

```
Bool
```

Example:

```
if (FirebaseCloudMessaging_IsAutoInitEnabled())
{
    FirebaseCloudMessaging_SetAutoInitEnabled(false)
}
```

The code above checks if auto-initialization is enabled and if it is it disables it (using FirebaseCloudMessaging_SetAutoInitEnabled).

FirebaseCloudMessaging_SetAutoInitEnabled

Enables or disables auto-initialization of Firebase Cloud Messaging.

When enabled, Firebase Cloud Messaging generates a registration token on app startup if there is no valid one (see FirebaseCloudMessaging_GetToken) and periodically sends data to the Firebase backend to validate the token. This setting is persistent across app restarts.

> **NOTE** By default, Firebase Cloud Messaging auto-initialization is enabled.

Syntax:

```
FirebaseCloudMessaging_SetAutoInitEnabled(enabled)
```

Argument	Type	Description
enabled	boolean	Whether auto-initialization should be turned on or off.

Returns:

```
N/A
```

Example:

```
if (FirebaseCloudMessaging_IsAutoInitEnabled())
{
    FirebaseCloudMessaging_SetAutoInitEnabled(false)
}
```

The code above checks if auto-initialization is enabled (using the FirebaseCloudMessaging_IsAutoInitEnabled function) and if it is disables it.

FirebaseCloudMessaging_SubscribeToTopic

Subscribes the user to the given topic in the background. The subscription operation is persistent and will keep retrying until it is successful. This uses the FCM registration token to identify the app instance, generating one if it does not exist (see FirebaseCloudMessaging_GetToken), which periodically sends data to the Firebase backend when auto-init is enabled. To delete the data, delete the token (see FirebaseCloudMessaging_DeleteToken).

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

Syntax:

```
FirebaseCloudMessaging_SubscribeToTopic(topic)
```

Argument	Type	Description
topic	string	The name of the topic to subscribe to.

Returns:

```
N/A
```

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The constant `"FirebaseCloudMessaging_SubscribeToTopic"`
success	boolean	Whether or not the function task succeeded.
value	string	The name of topic subscription requested.

Example:

```
FirebaseCloudMessaging_SubscribeToTopic("my_awesome_topic")
```

In the code above we request a subscription to a topic (`"my_awesome topic"`). The function callback can be caught inside an **Async Social** event.

```
if(async_load[?"type"] == "FirebaseCloudMessaging_SubscribeToTopic")
{
    var _topic = async_load[? "topic"];

    if(async_load[?"success"])
    {
        show_debug_message(Subscription to " + _topic + " SUCCEEDED")
    }
    else
    {
        show_debug_message(Subscription to " + _topic + " FAILED")
    }
}
```

The code above matches the response against the correct event **type**, and if the tasks succeeds it stores the topic value in a local variable (`_topic`) and logs the success of the operation.

FirebaseCloudMessaging_UnsubscribeFromTopic

Unsubscribes from a previously subscribed topic (see FirebaseCloudMessaging_SubscribeToTopic) in the background. The unsubscribe operation is persistent and will keep retrying until it is completed.

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

Syntax:

```
FirebaseCloudMessaging_UnsubscribeFromTopic(topic)
```

Argument	Type	Description
topic	string	The name of the topic to unsubscribe from.

Returns:

```
N/A
```

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The constant `"FirebaseCloudMessaging_UnsubscribeFromTopic"`
success	boolean	Whether or not the function task succeeded.
value	string	The name of topic subscription requested.

Example:

```
FirebaseCloudMessaging_UnsubscribeFromTopic("my_awesome_topic")
```

In the code above we request for the subscription to the topic `"my_awesome topic"` to be canceled. The function callback can be caught inside an Async Social event.

```
if(async_load[?"type"] == "FirebaseCloudMessaging_UnsubscribeFromTopic")
{
    var _topic = async_load[? "topic"];

    if(async_load[?"success"])
    {
        show_debug_message(Subscription removed successfully")
    }
    else
    {
        show_debug_message(Subscription could not be removed")
    }
}
```

The code above matches the response against the correct event **type**, and if the tasks succeeds it stores the topic value in a local variable (`_topic`) and logs the success of the operation.