Firebase Authentication

Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to your app. It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook and Twitter, and more.

Check the **official page** for more information.

Setup

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

> NOTE The **SDK** version is only available for Android, iOS and Web targets; if you're targeting other devices please follow the steps for **REST API**

- **Create Project** (skip this if you already have a project)
- **Firebase Console** (enabling Firebase Authentication)
- **Platform Setup** (configure SDKs or REST API)

Compatibility

Below is the GameMaker Studio compatibility table for each provider depending on the target platform (keep in mind that the **REST API** can be used on every platform).

Provider	Android	iOS	Web	REST API
Custom	yes	yes	yes	yes
Anonymous	yes	yes	yes	yes
Email/Password	yes	yes	yes	yes
Phone	yes	yes	yes	yes
Google	yes	yes	yes	yes
Apple	yes	yes	yes	yes
Facebook	yes	yes	yes	yes
Google Play	yes	no	no	yes
GameCenter	no	yes	no	no

Auth Configuration

These are the available authentication setup guides that you can follow to implement the desired authentication method in your project.

- Apple
- Custom
- Email
- Facebook
- GameCenter
- Google
- GooglePlay
- Phone

Functions

The following functions are given for working with Firebase Authentication extension:

- FirebaseAuthentication_ChangeDisplayName
- FirebaseAuthentication_ChangeEmail

- FirebaseAuthentication_ChangePassword
- FirebaseAuthentication_ChangePhotoURL
- FirebaseAuthentication_DeleteAccount
- FirebaseAuthentication_GetDisplayName
- FirebaseAuthentication_GetEmail
- FirebaseAuthentication_GetEmailVerified
- FirebaseAuthentication_GetIdToken
- FirebaseAuthentication_GetLocalId
- FirebaseAuthentication_GetPhotoUrl
- FirebaseAuthentication_GetProviderUserInfo
- FirebaseAuthentication_GetUID
- FirebaseAuthentication_GetUserData_raw
- FirebaseAuthentication_LinkWithEmailPassword
- FirebaseAuthentication_LinkWithOAuthCredential
- FirebaseAuthentication_LinkWithPhoneNumber
- FirebaseAuthentication_ReauthenticateWithEmail
- FirebaseAuthentication_ReauthenticateWithOAuth
- FirebaseAuthentication_ReauthenticateWithPhoneNumber
- FirebaseAuthentication_RecaptchaParams
- FirebaseAuthentication_RefreshUserData
- FirebaseAuthentication_SendEmailVerification
- FirebaseAuthentication_SendPasswordResetEmail
- FirebaseAuthentication_SendVerificationCode
- FirebaseAuthentication_SignIn_Anonymously
- FirebaseAuthentication_SignIn_Email

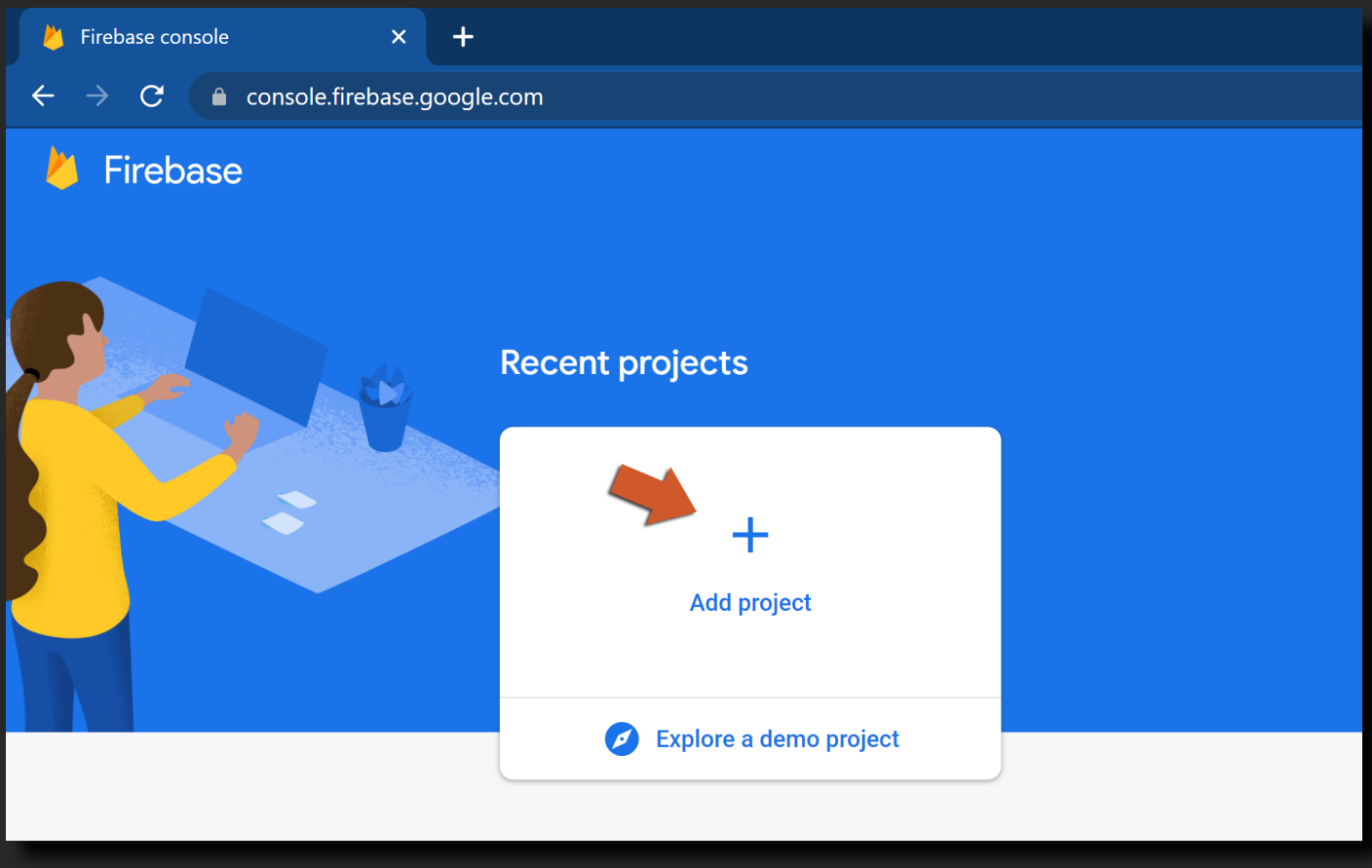
- FirebaseAuthentication_SignIn_OAuth
- FirebaseAuthentication_SignInWithCustomToken
- FirebaseAuthentication_SignInWithPhoneNumber
- FirebaseAuthentication_SignOut
- FirebaseAuthentication_SignUp_Email
- FirebaseAuthentication_UnlinkProvider
- SDKFirebaseAuthentication_LinkWithGameCenter *
- SDKFirebaseAuthentication_LinkWithProvider *
- SDKFirebaseAuthentication_ReauthenticateWithGameCenter *
- SDKFirebaseAuthentication_ReauthenticateWithProvider *
- SDKFirebaseAuthentication_SignIn_GameCenter *
- SDKFirebaseAuthentication_SignInWithProvider *

* these functions are only available on SDK targets (Android, iOS, Web). Check the **compatibility** table above.

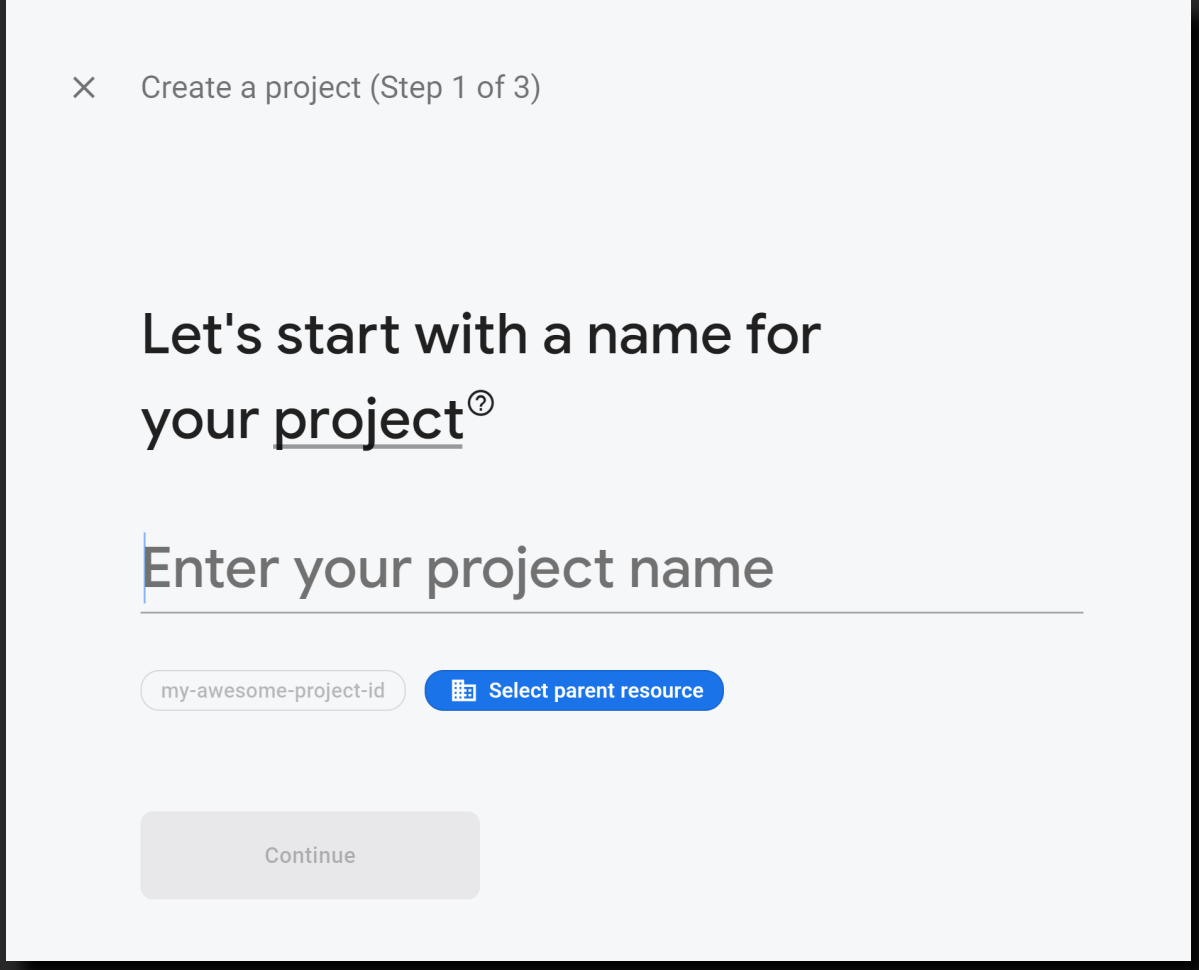
Create Project

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

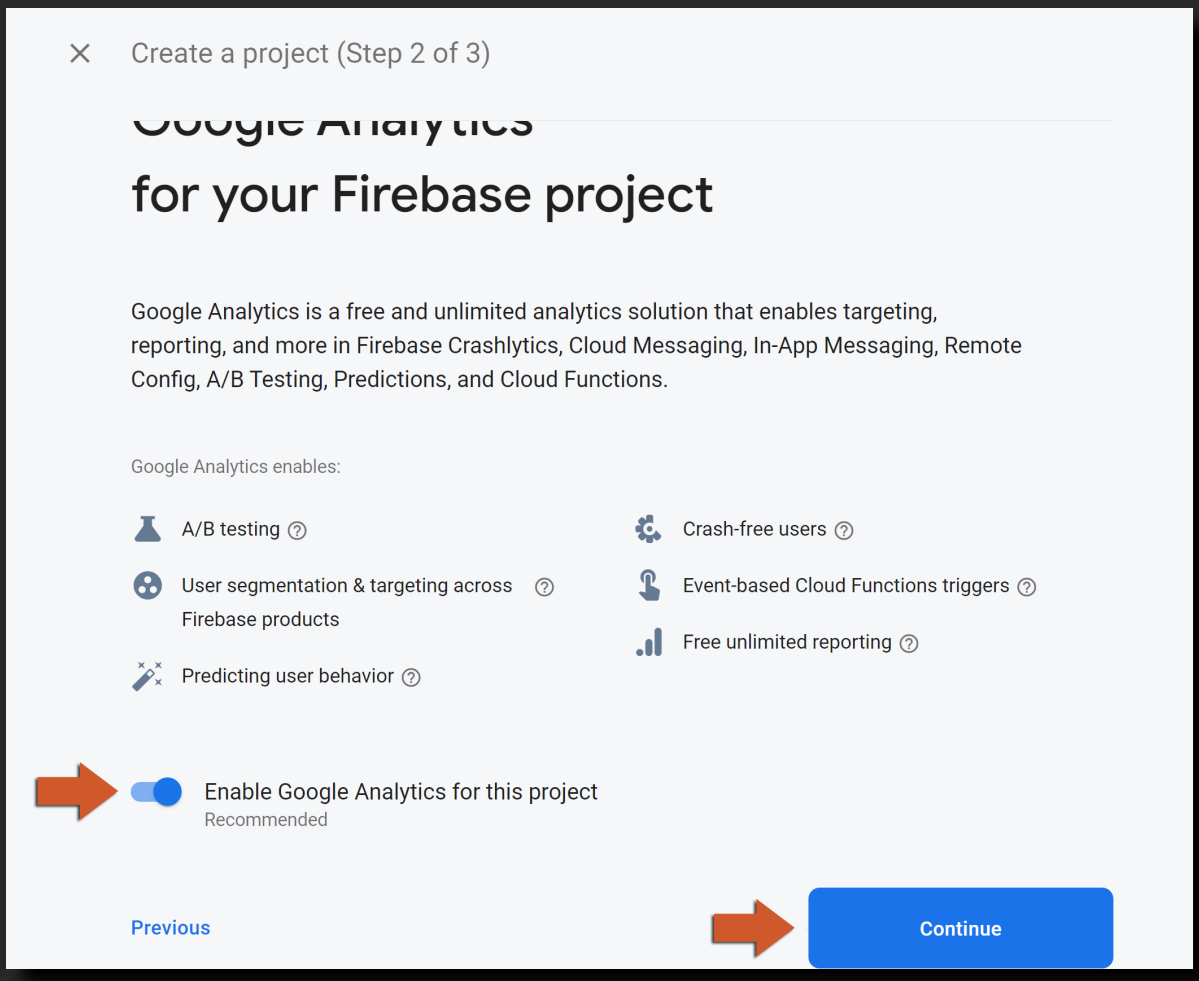
1. Go to the **Firebase Console** web site.
2. Click on **Add Project** to create your new project.



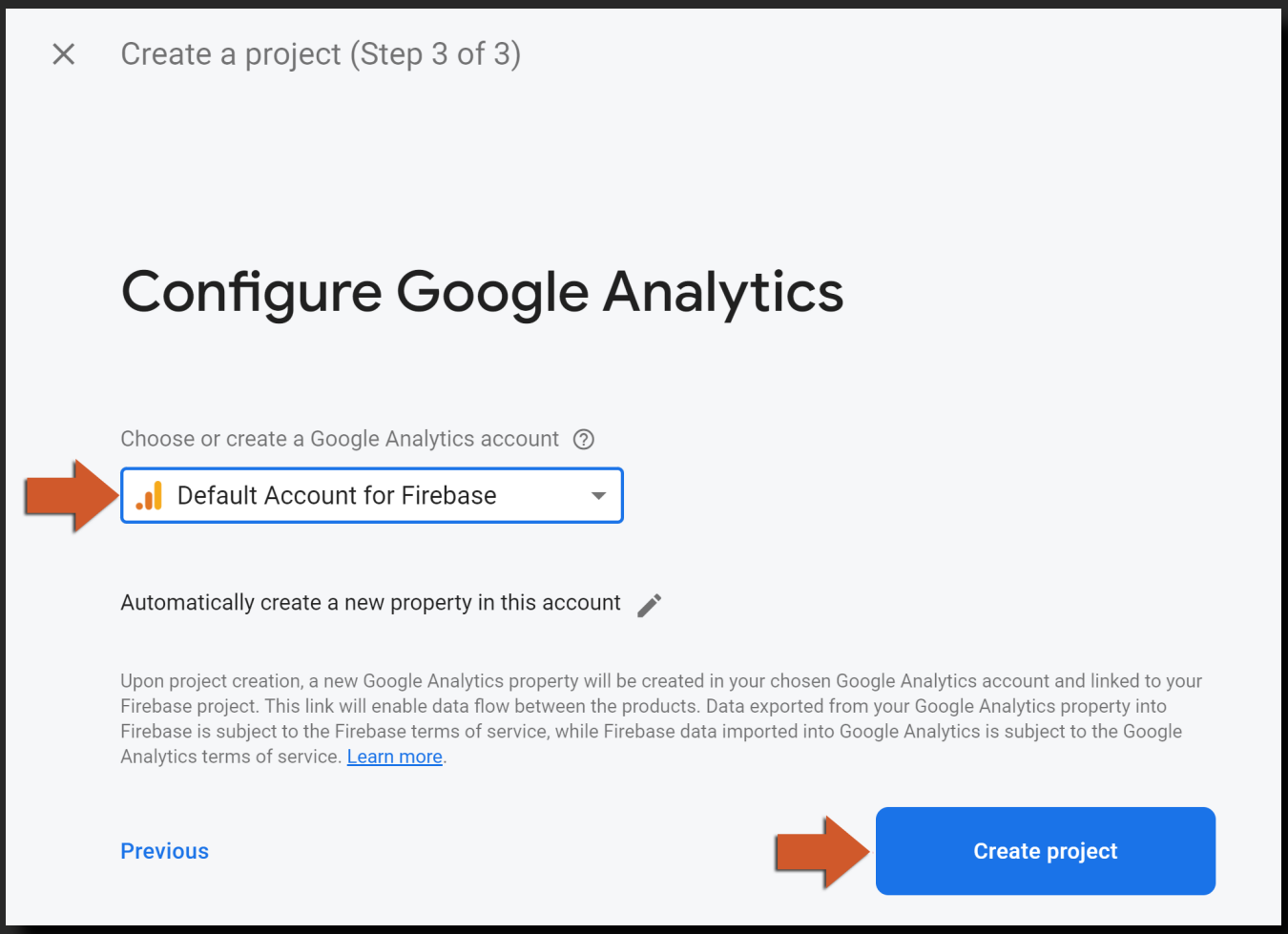
3. Enter a name for your project and click on the **Continue** button.



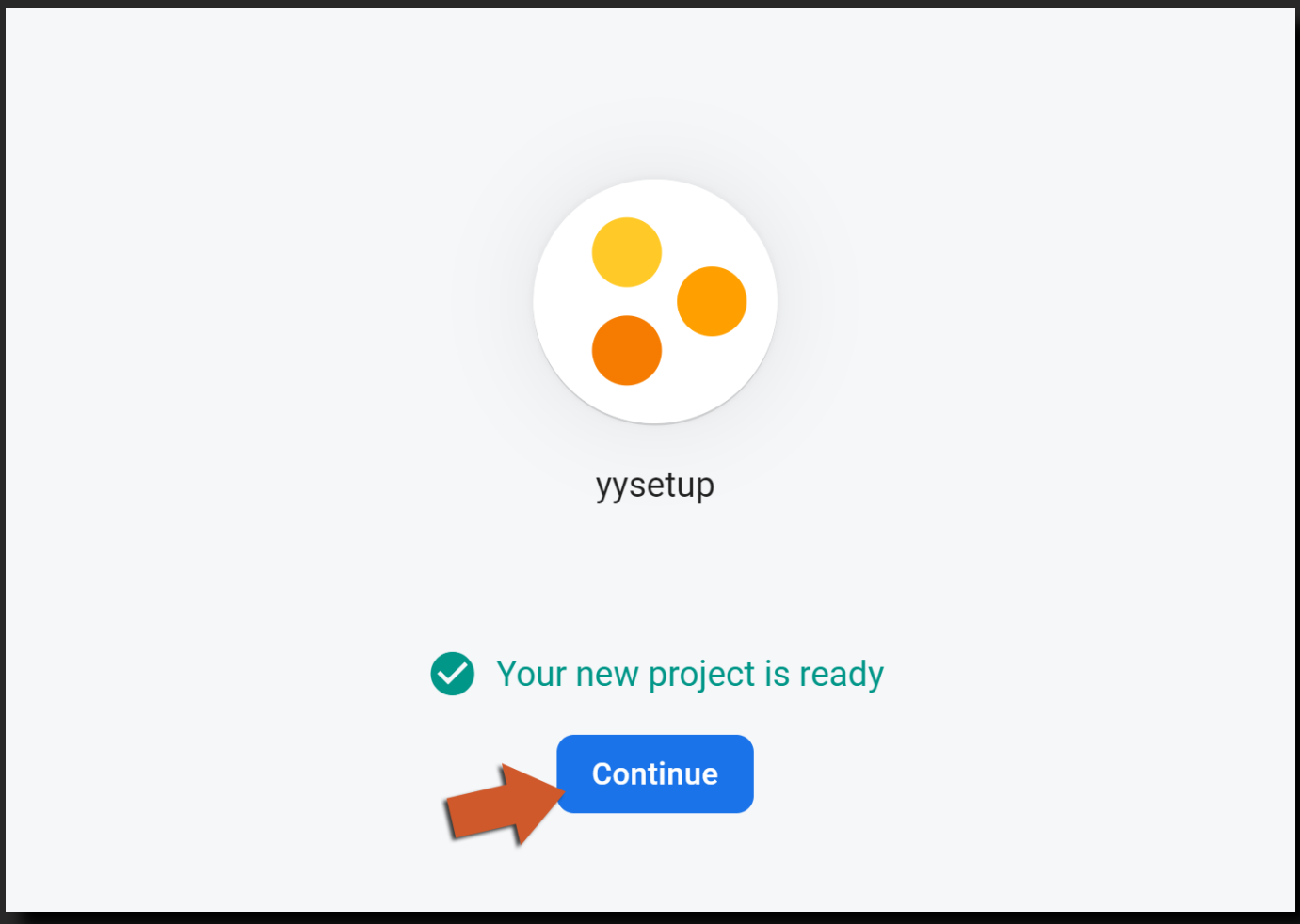
4. On the next page, make sure that **Enable Google Analytics for this project** is enabled and then click the **Continue** button:



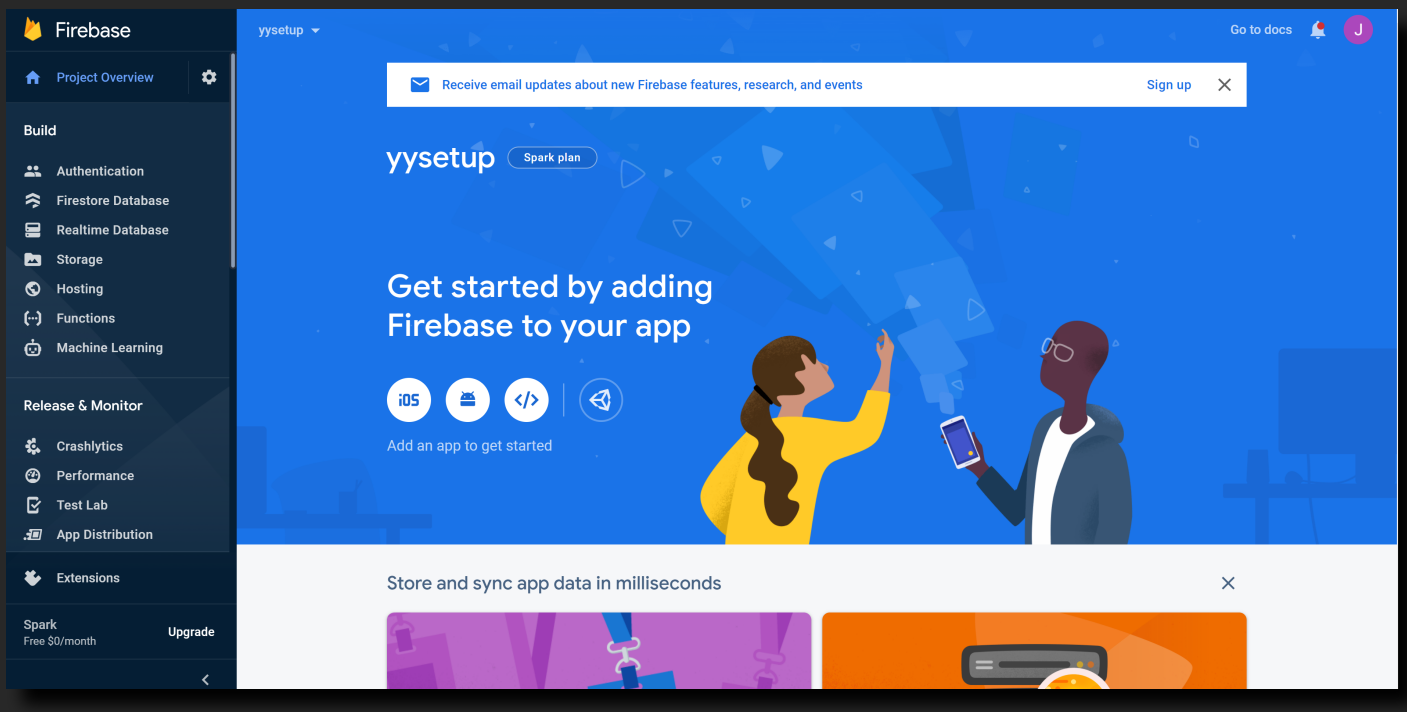
5. Select your account and click the **Create project** button:



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:

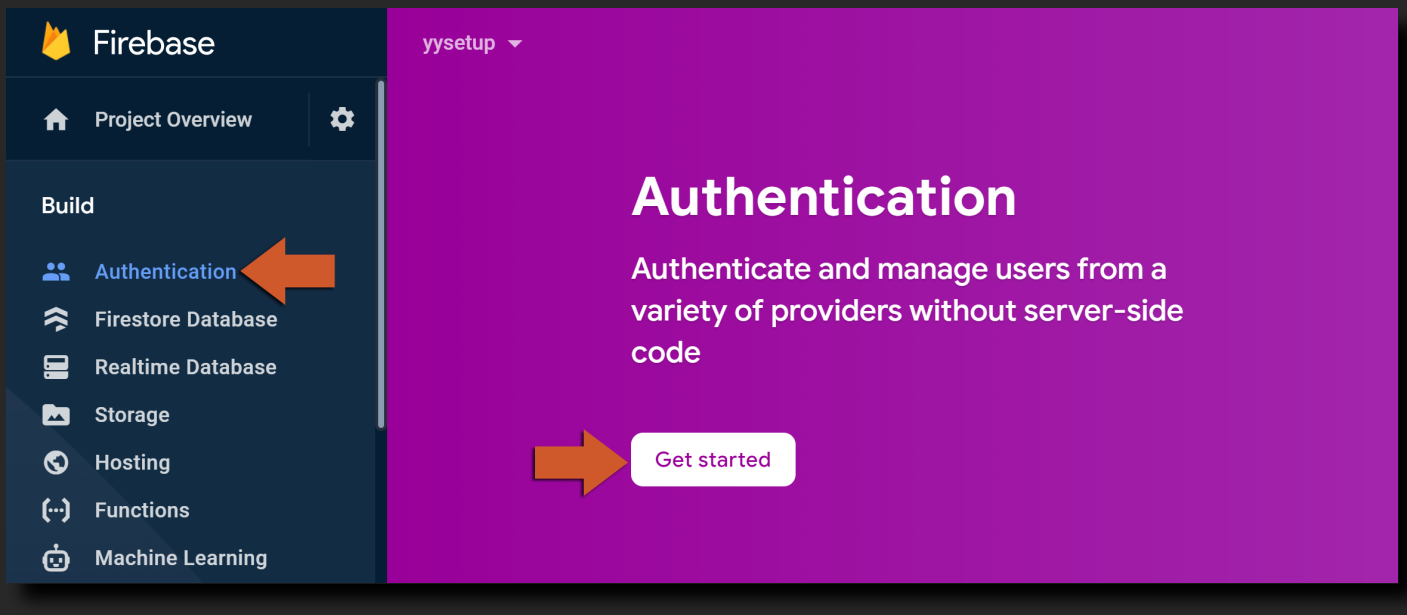


8. Continue your adventure with the Firebase extensions provided for GameMaker Studio 2!

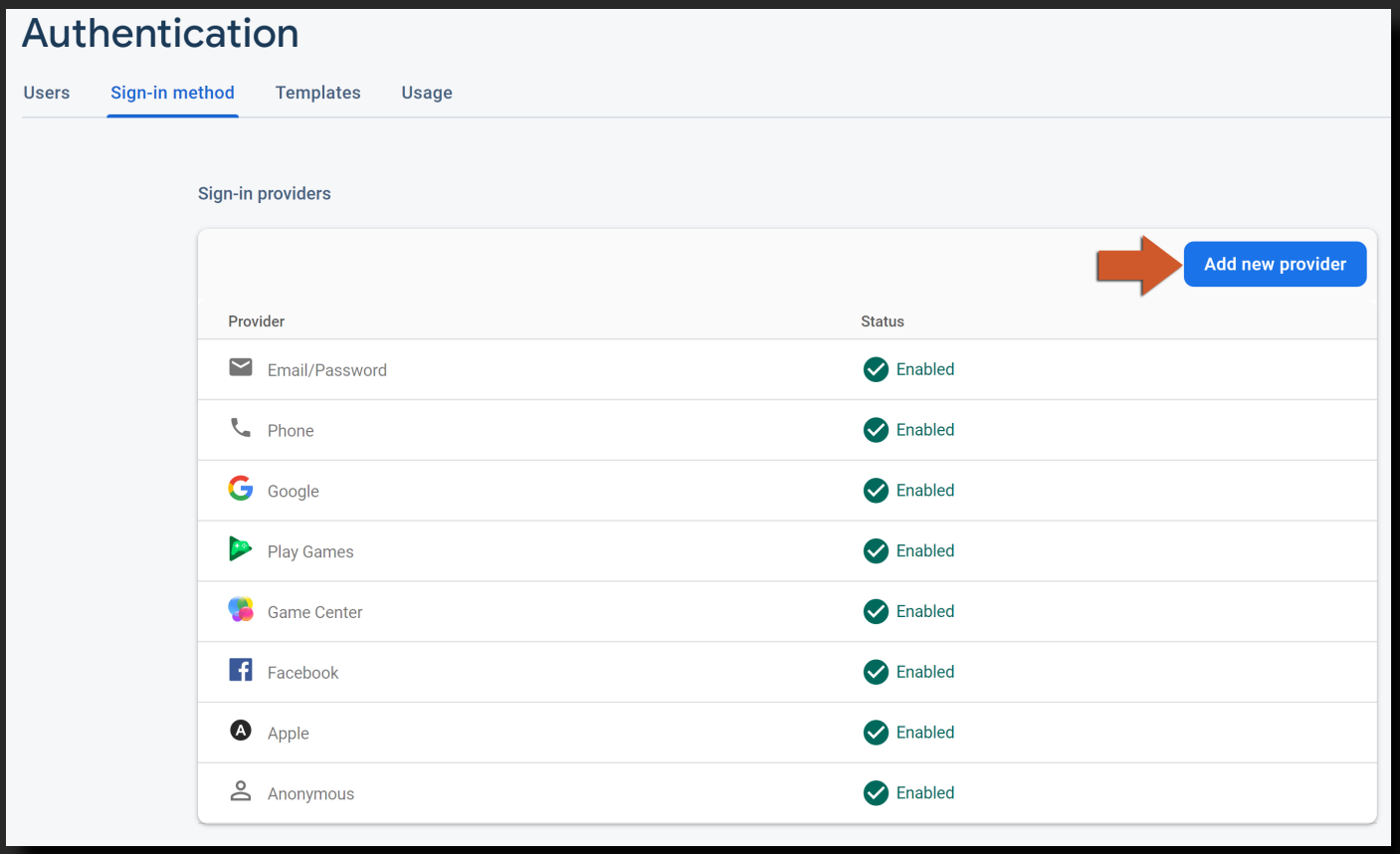
Firebase Console

Before we are able to use the Firebase Authentication extension, we need to enable it within the **Firebase Console**.

1. On your Firebase project, click on **Authentication** and then the **Get Started** button.



2. Go to the **Sign-in method** tab and choose the providers that you want to use (in the image below you can see the providers compatible with GameMaker at the moment)



3. You've now finished configuring the Firebase Console for using the Firebase Authentication extension.

Platform Setup

Firebase Authentication implementation uses both SDK (working on **Android**, **iOS** and **Web**) and REST API that allows it to work on other platforms. In this section we will cover the required setup necessary to start using the Firebase Authentication extension on your game.

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

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

Advanced Configuration

Firebase Authentication by default uses SDKs on **Android**, **iOS** and **Web** targets and uses REST API for all other exports but you can change this behavior (ie.: forcing REST API to be used even on SDK versioned platforms) by changing the macro `FirebaseAuthentication_Library` using this macros:

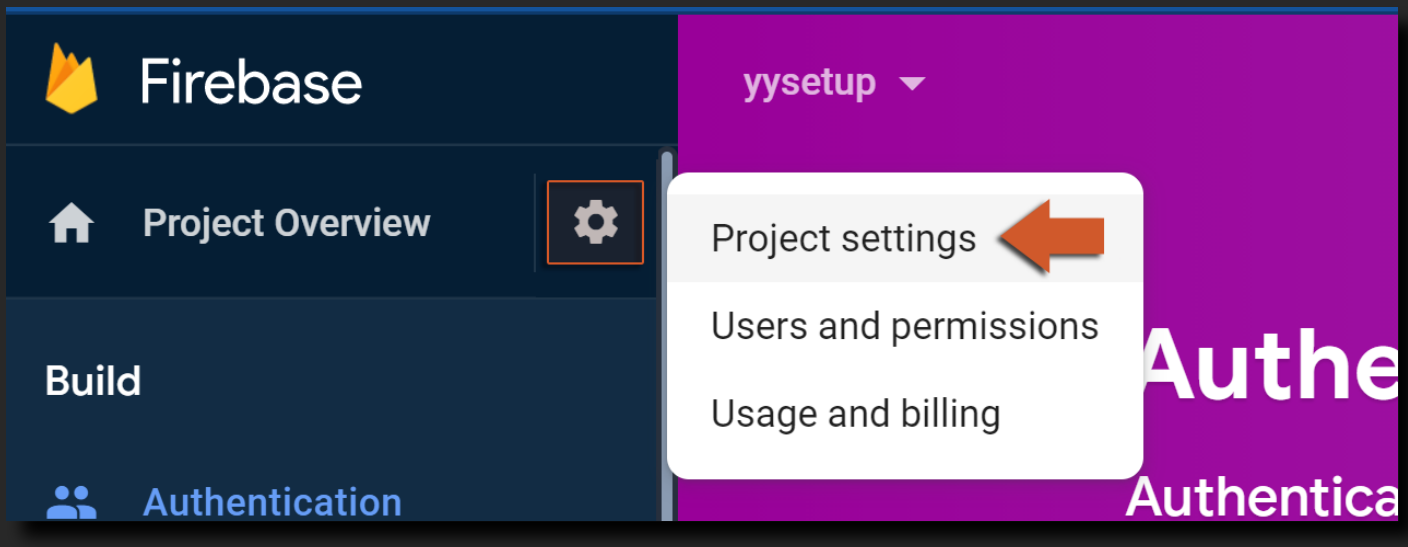
- `FirebaseAuthentication_LibraryOptions_SDKOnly`
- `FirebaseAuthentication_LibraryOptions_SDKWhenAvailable`
- `FirebaseAuthentication_LibraryOptions_RESTOnly`

Android Setup

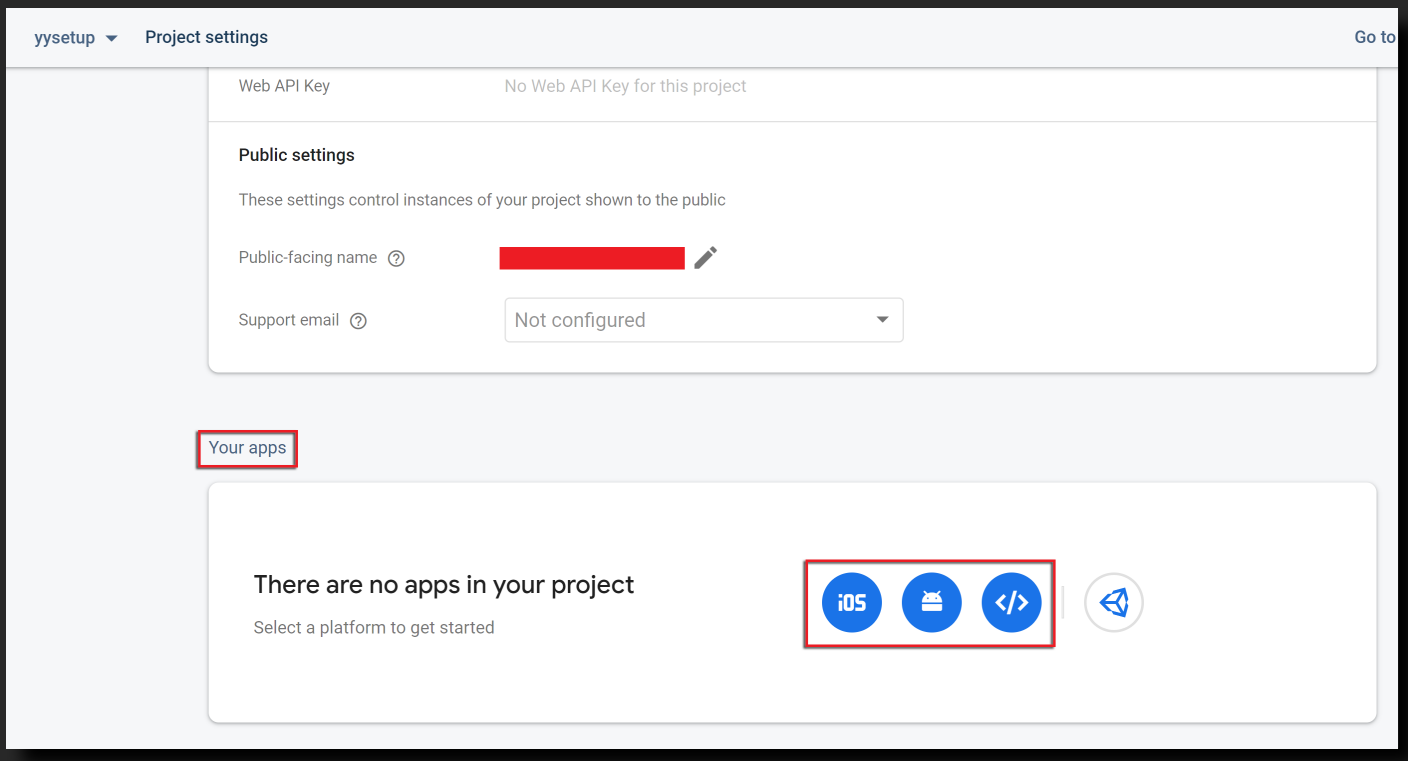
This setup is necessary for all the Firebase modules using Android and needs to be done once per project, and basically involves importing the **google-services.json** file into your project.

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

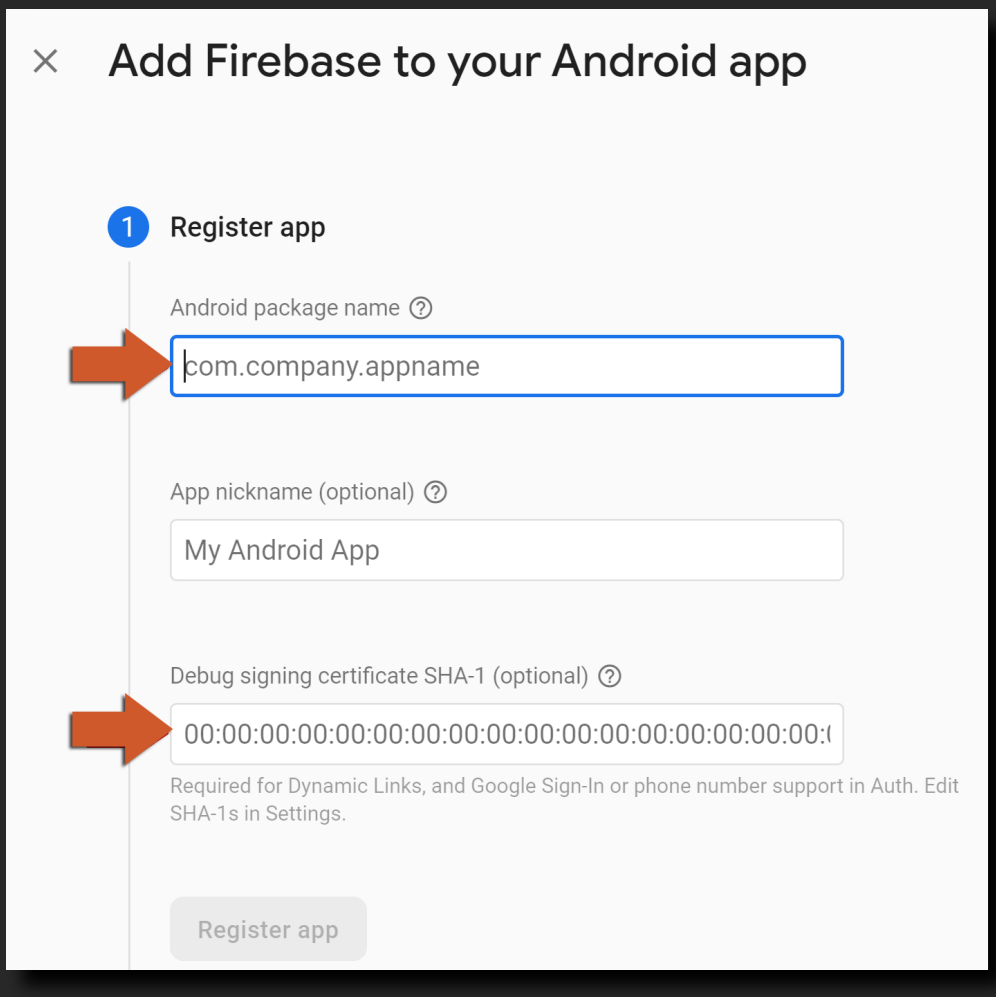
1. Click the **Settings** icon (next to Project Overview) and select **Project settings:**



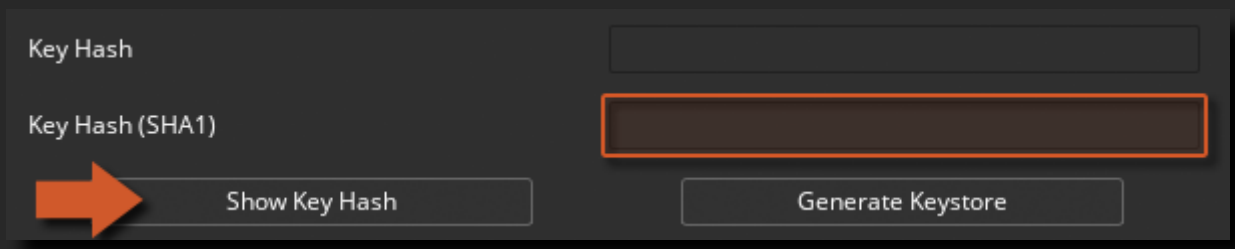
2. Now go to the **Your apps** section and click on the **Android** button:



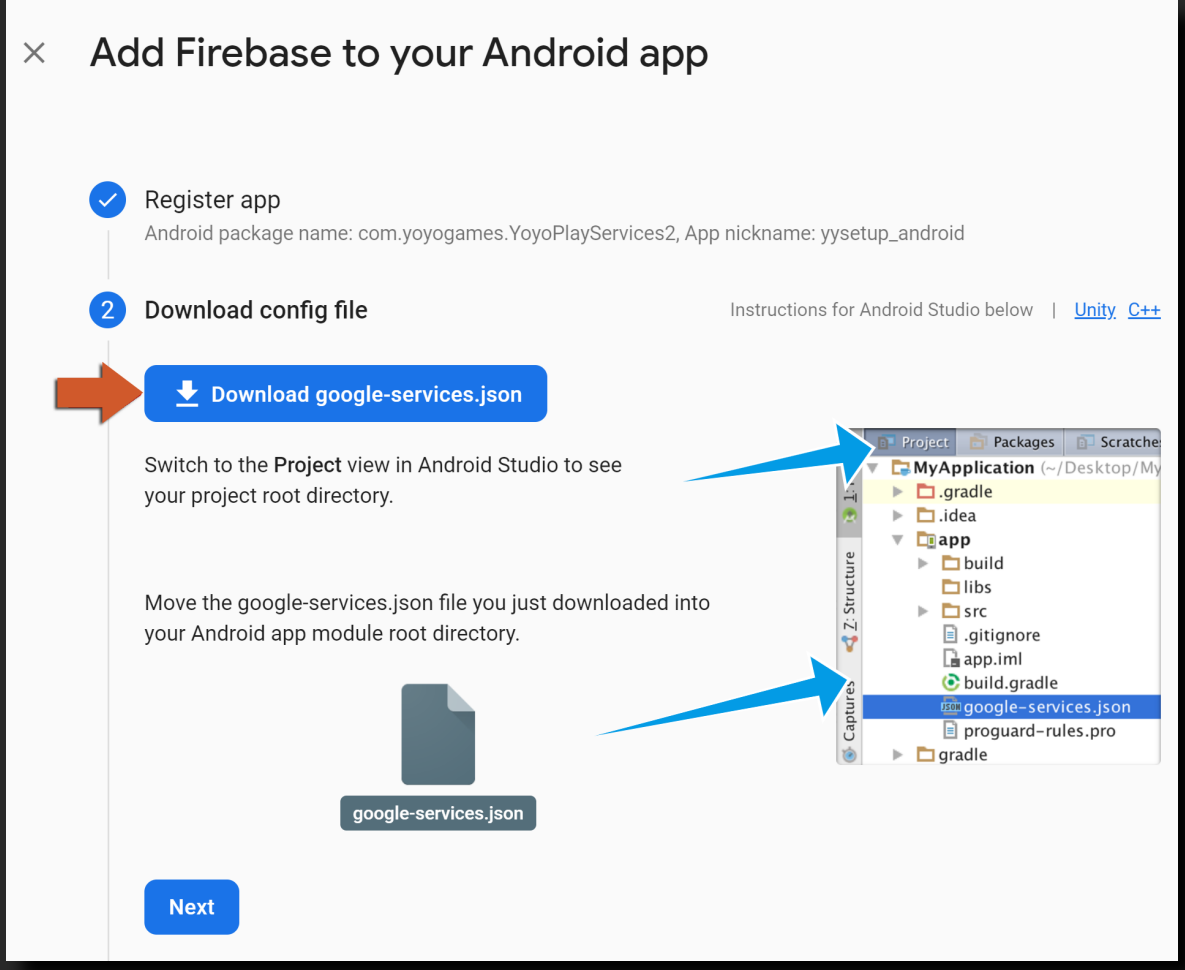
3. On this screen you need enter your **Package name** (required), **App nickname** (optional) and **Debug signing certificate SHA-1** (required if you are using Firebase Authentication).



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

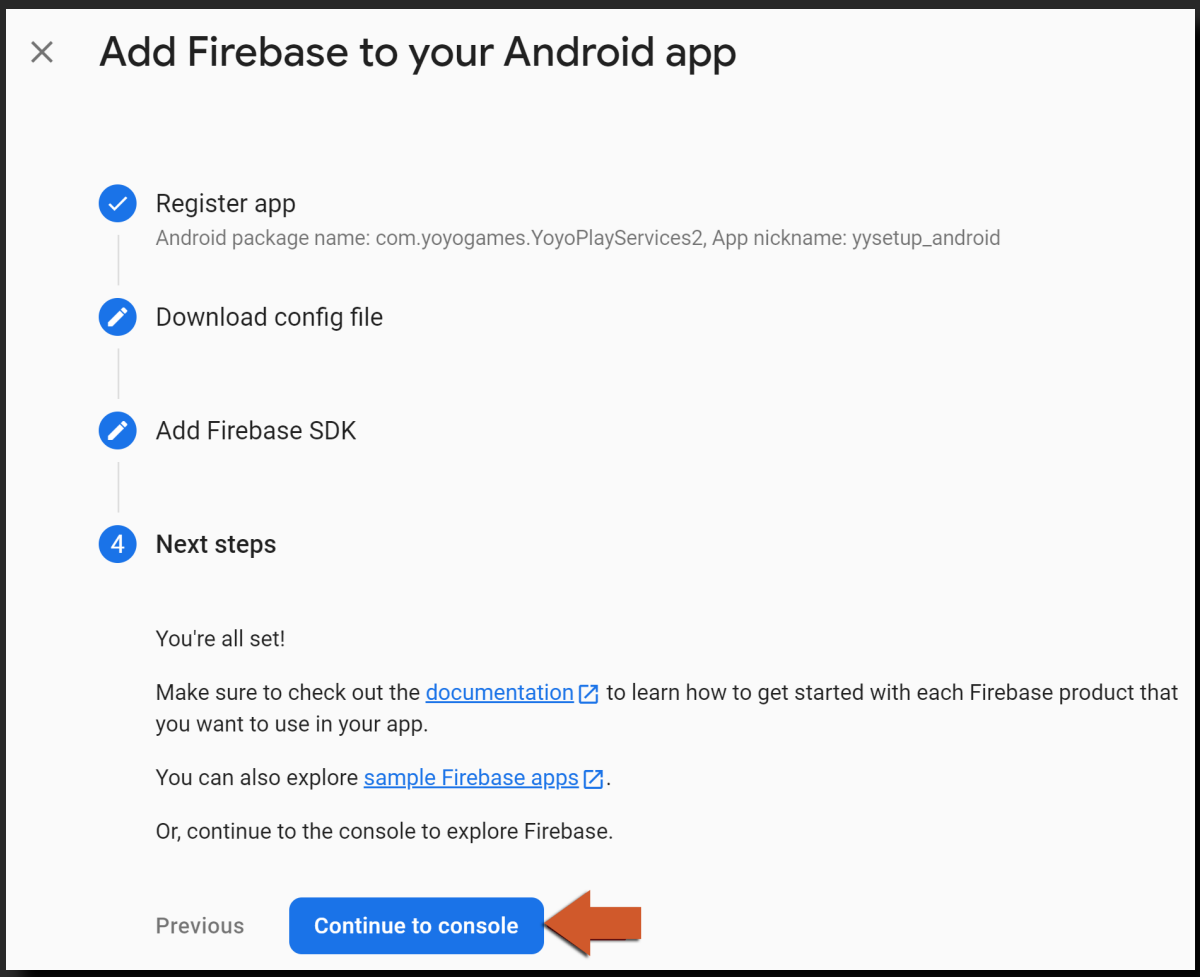


4. Click on **Download google-services.json** (make sure to save this file as we will need it in subsequent steps).

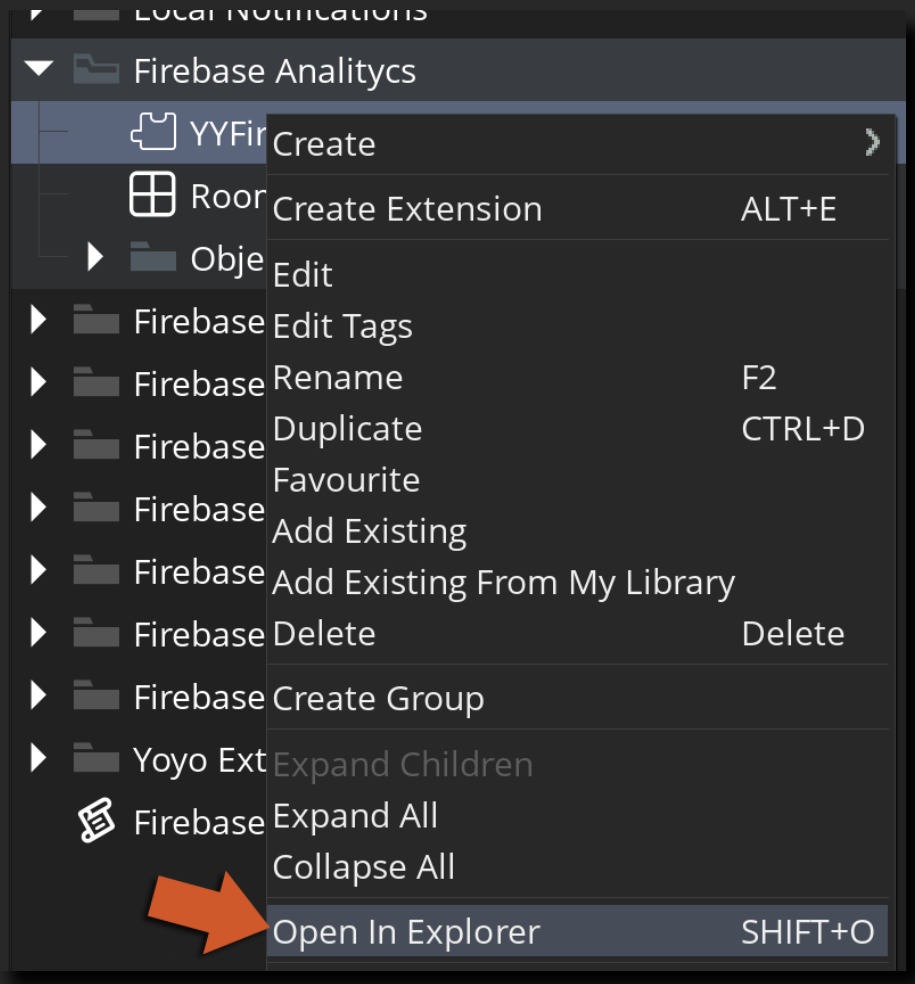


5. Ignore this screen, as this is already done in the extension.

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

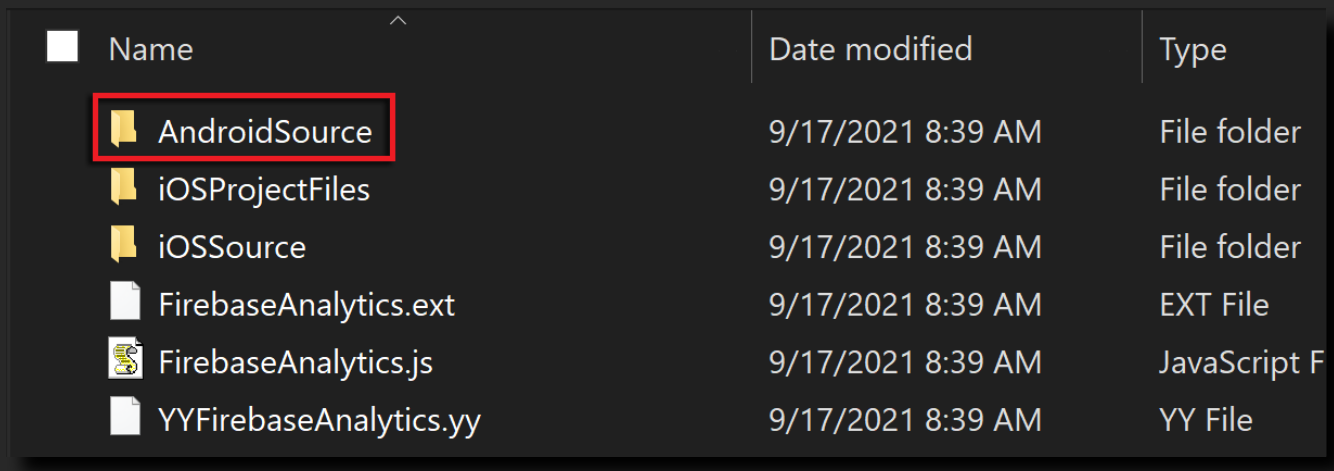


7. Now go into GameMaker Studio 2, right click on your Firebase extension asset and click on **Open In Explorer / Show In Finder.**

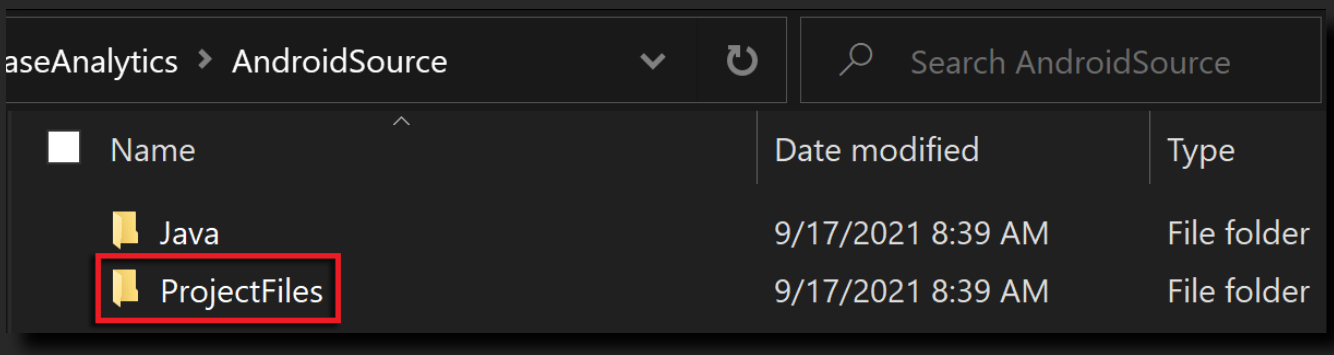


NOTE You only need to perform this set-up using **one** of your Firebase extensions (per project); for this example we'll be using the Firebase Analytics extension.

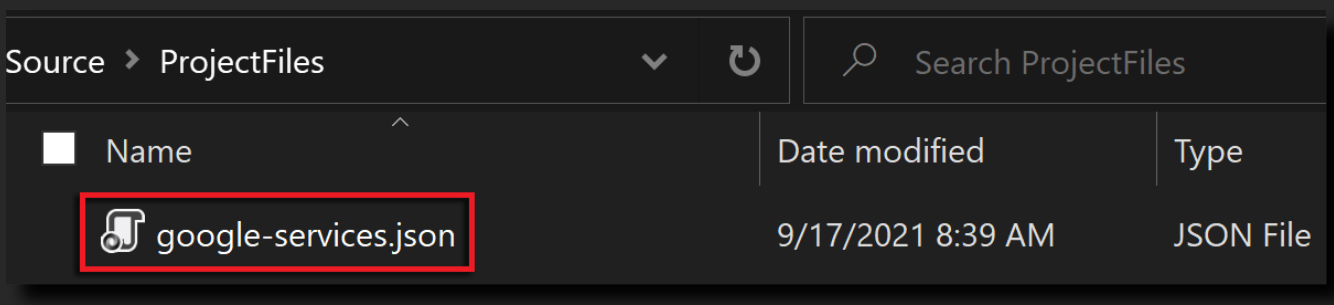
8. Open the **AndroidSource** folder.



9. If the **ProjectFiles** folder does not exist, please create it.



10. Open **ProjectFiles** folder and place your downloaded **google-services.json** file inside it.



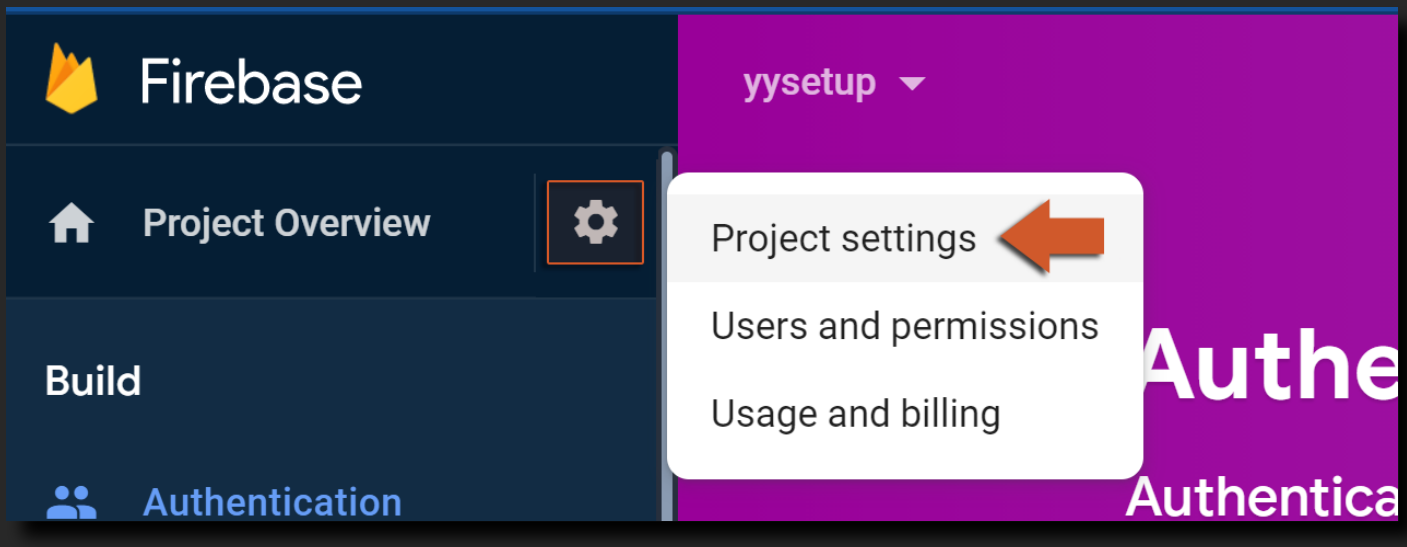
11. You have now finished the main setup for all Firebase Android modules!

iOS Setup

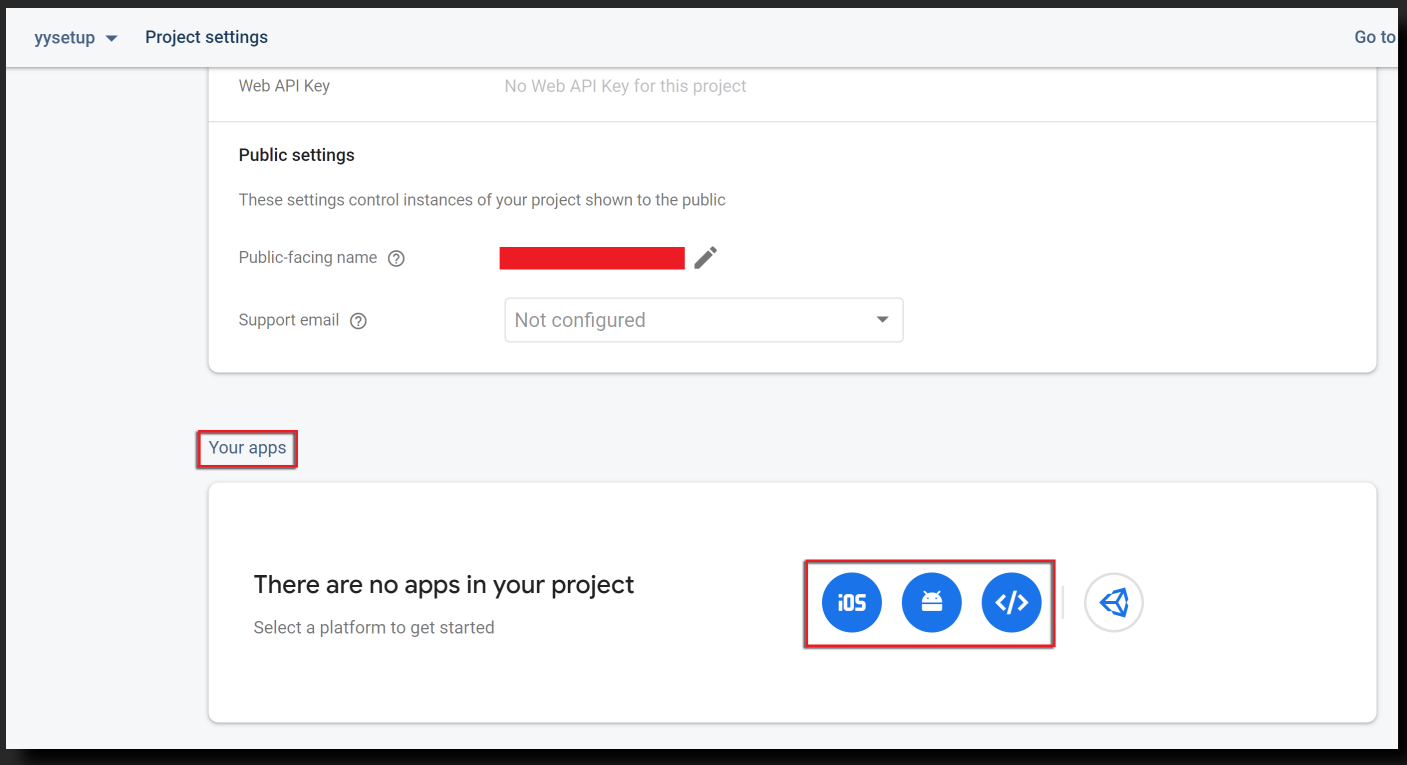
This setup is necessary for all the Firebase modules using iOS and needs to be done once per project, and basically involves importing the **GoogleServices-Info.plist** file into your project.

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

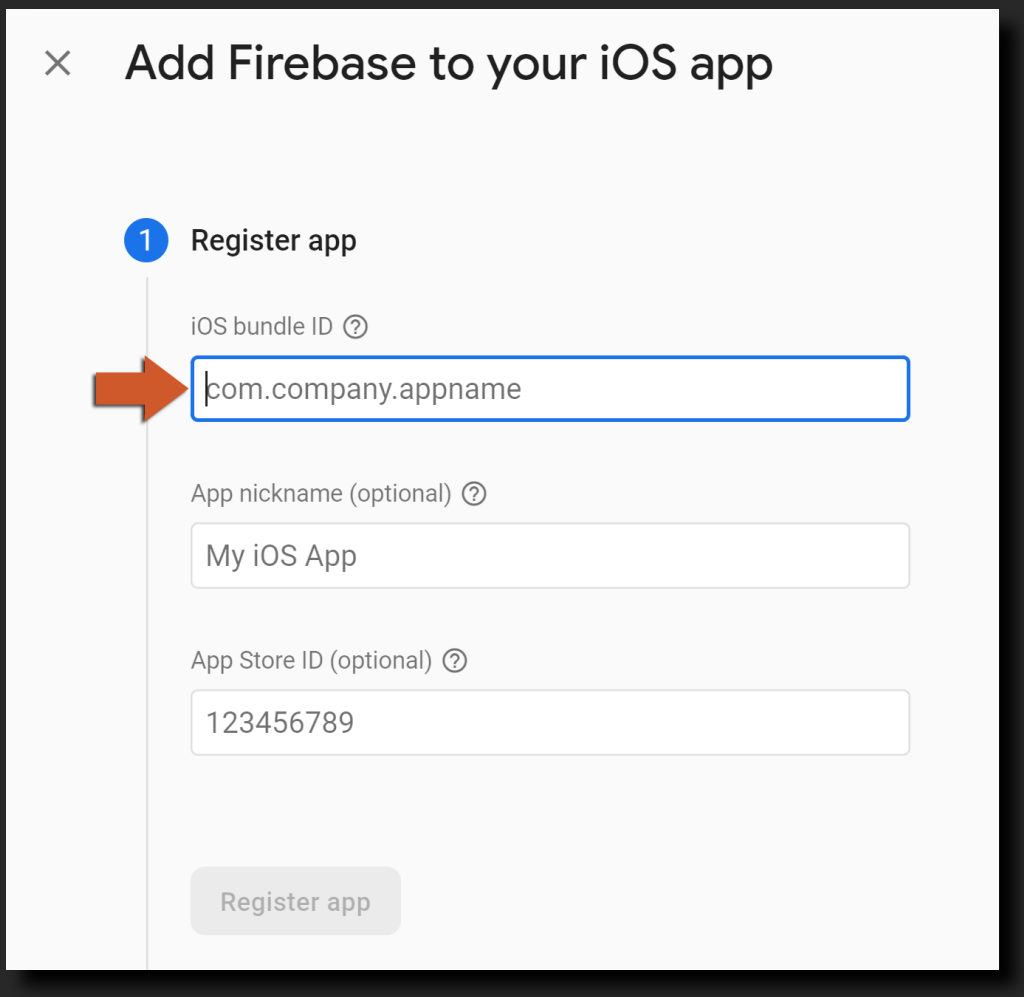
1. Click the **Settings** icon (next to Project Overview) and select **Project settings:**



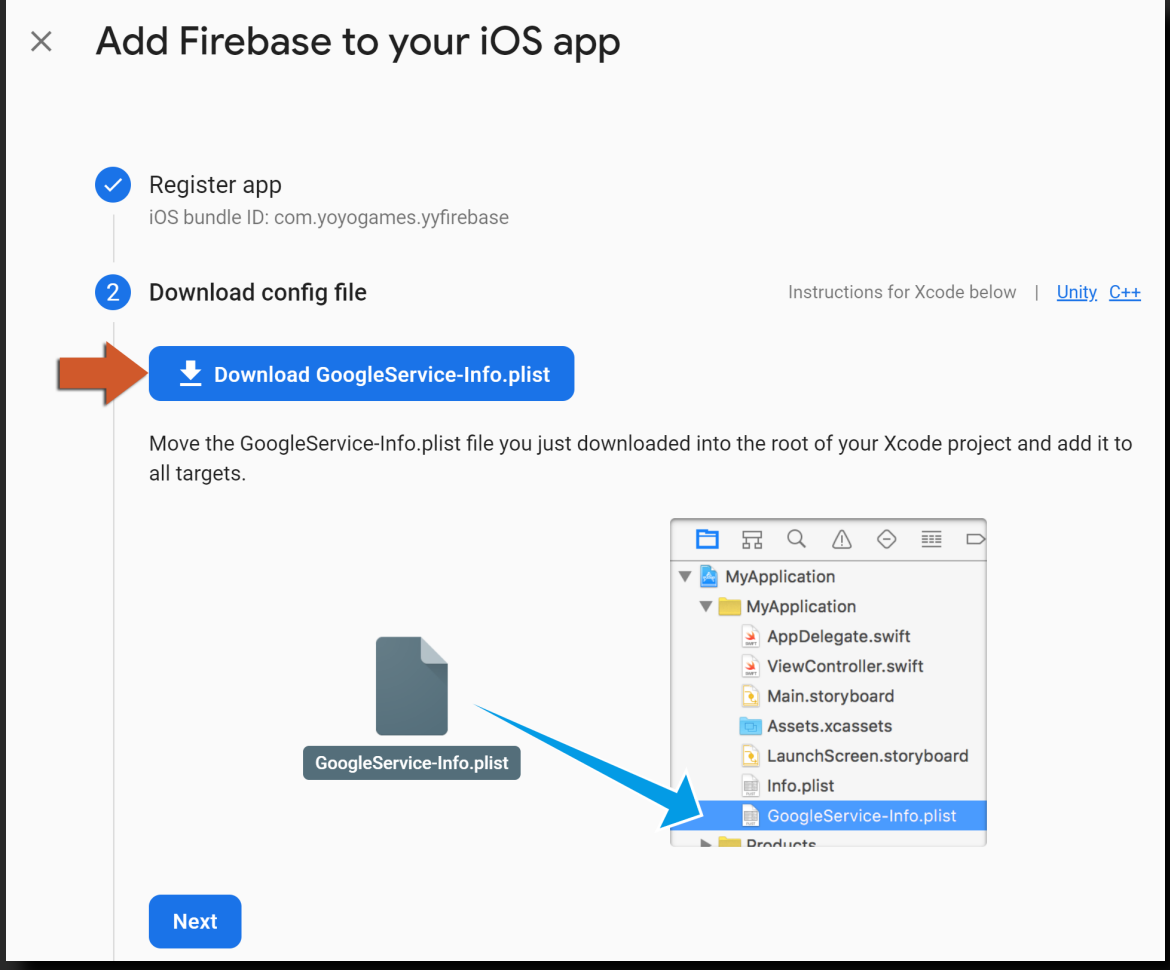
2. Now go to the **Your apps** section and click on the **iOS** button:



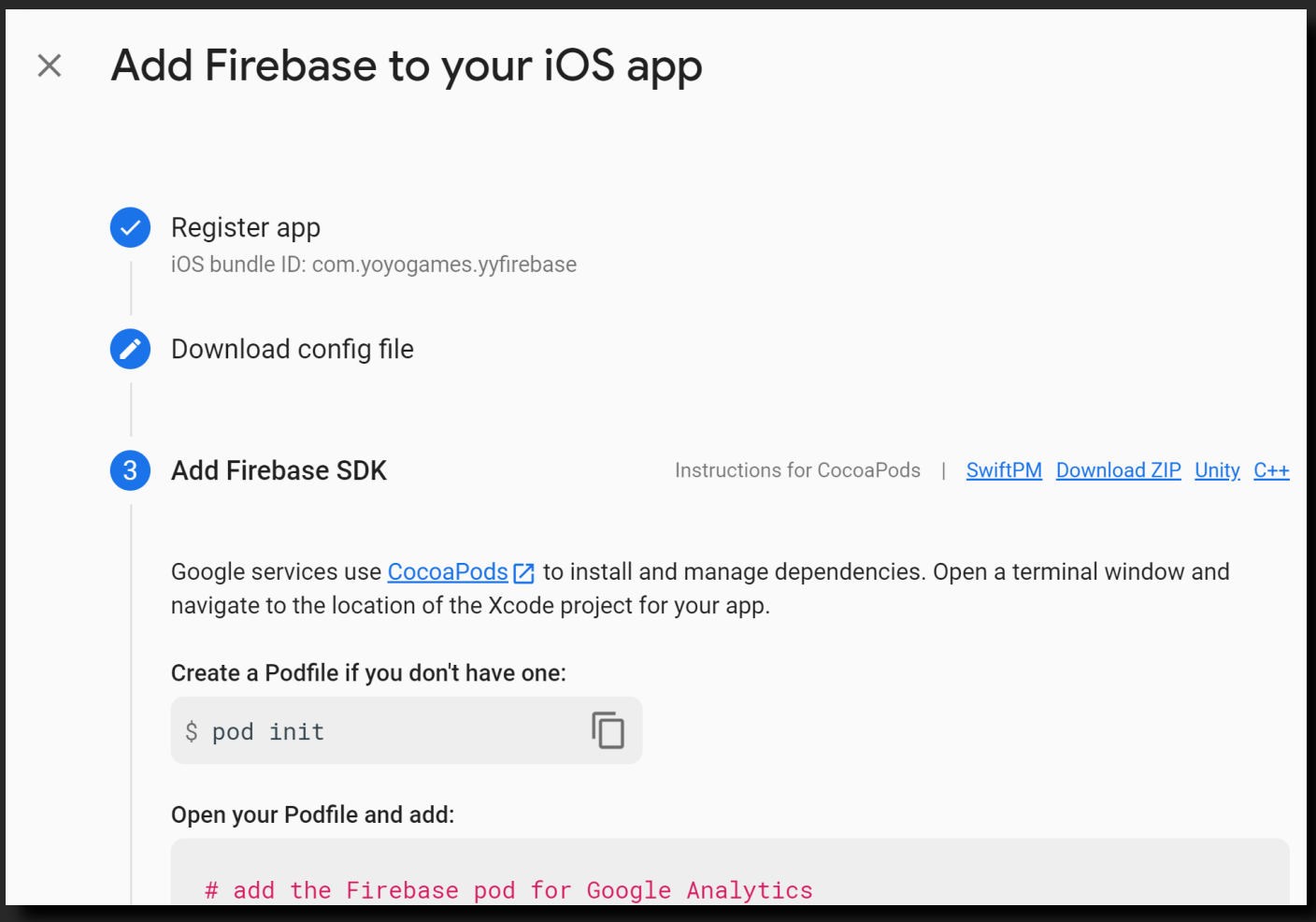
3. Fill the form with your **iOS Bundle ID**, **App nickname** and **AppStore ID** (last two are optional).



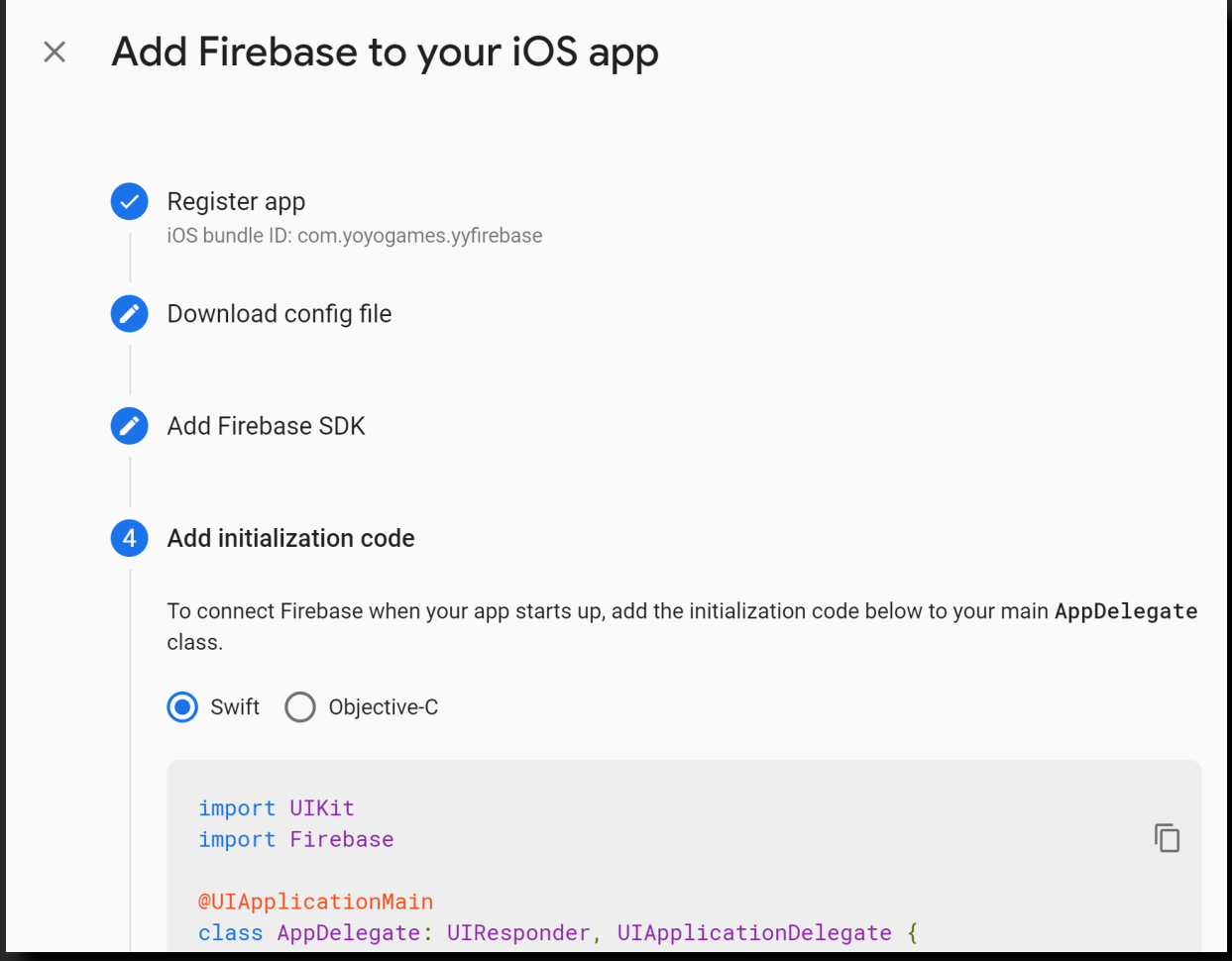
4. Click on **Download GoogleService-info.plist** (make sure to save this file as we will need it in subsequent steps).



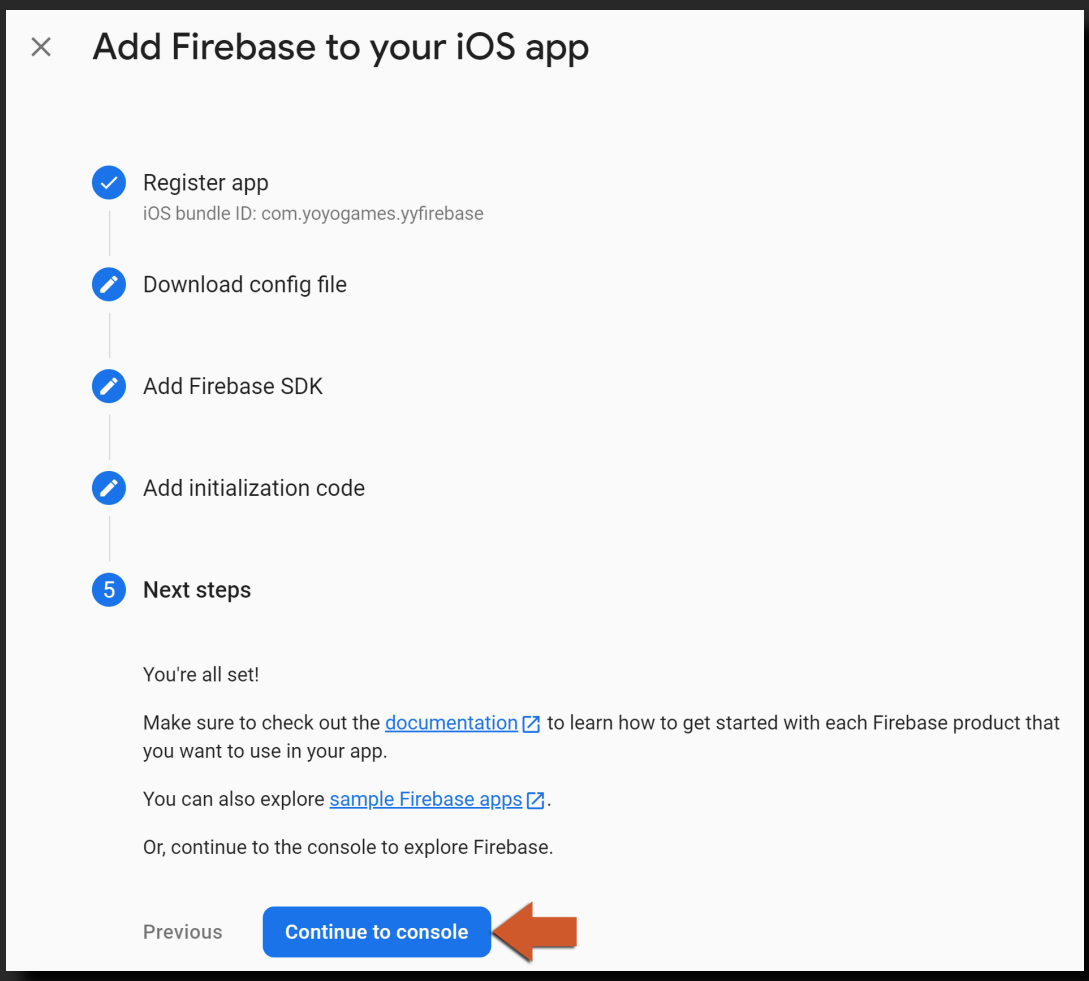
5. Ignore this screen, as this is already done in the extension.



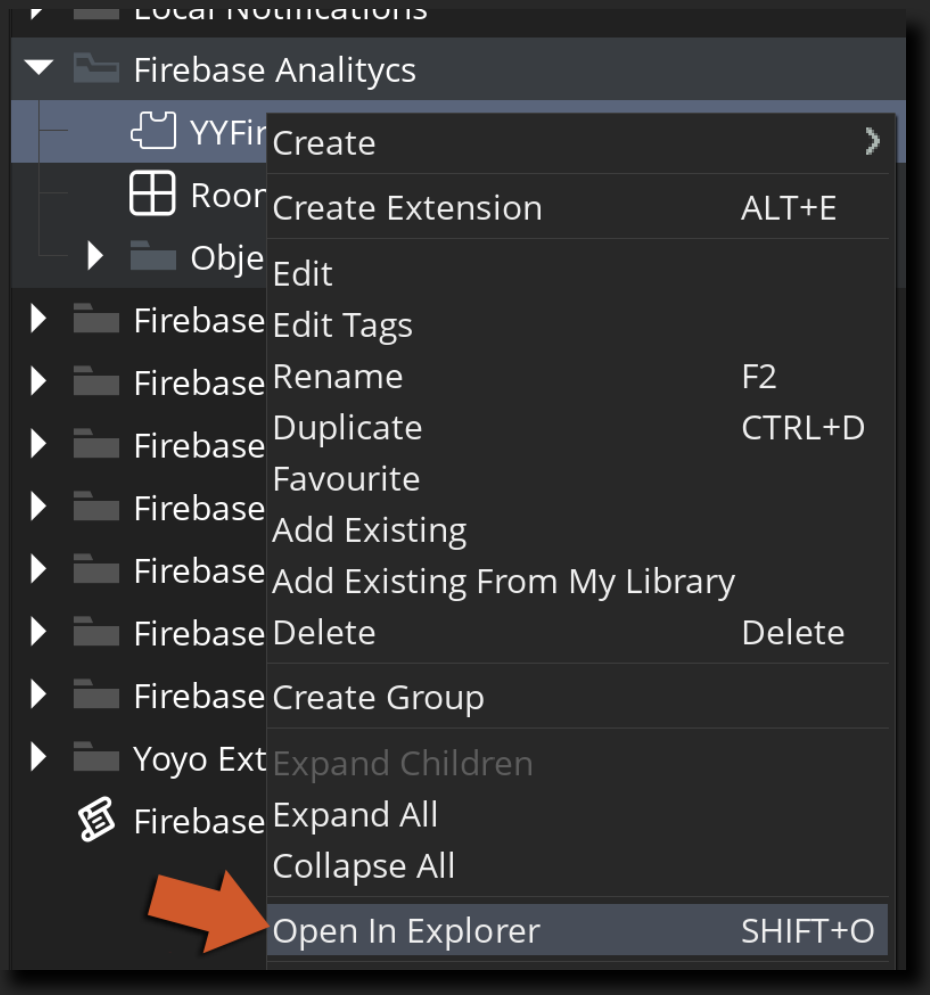
6. Ignore this screen as well, as this is also done in the extension.



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

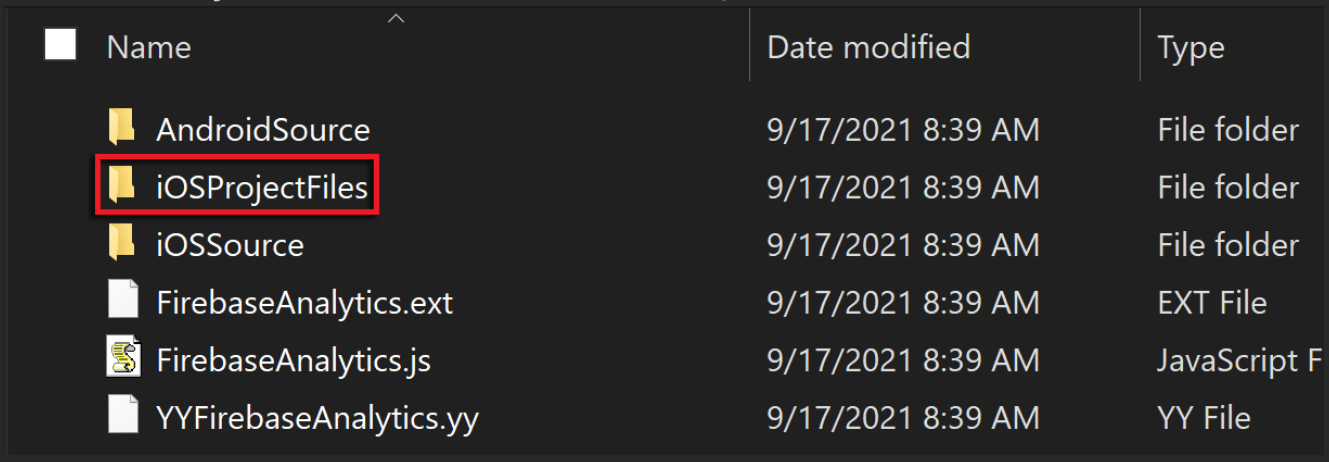


8. Now go into GameMaker Studio 2, right click on your Firebase extension asset and click on **Open In Explorer / Show In Finder.**

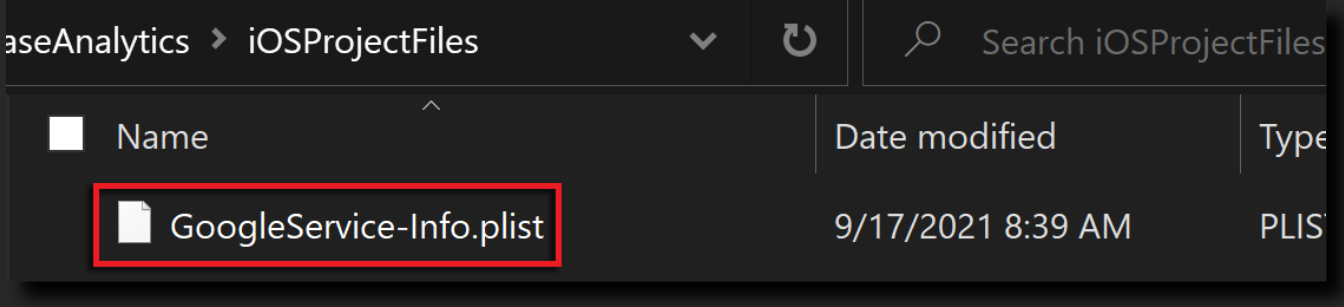


NOTE You only need to perform this set-up using **one** of your Firebase extensions (per project); for this example we'll be using the Firebase Analytics extension.

9. If the **iOSProjectFiles** folder does not exist, please create it.



10. Open the **iOSProjectFiles** folder and place your **GoogleService-Info.plist** file inside it.



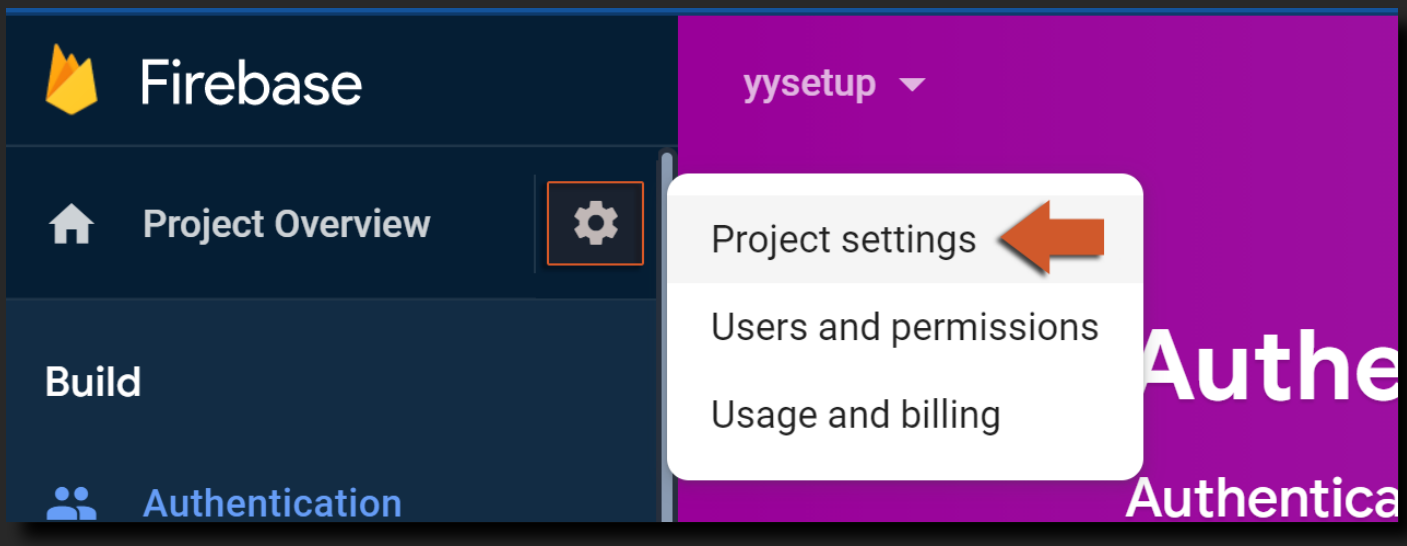
11. Make sure to set up **CocoaPods** for your project *unless* you are only using the REST API in an extension (if one is provided -- not all extensions provide a REST API) or the Firebase Cloud Functions extension (which only uses a REST API).

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

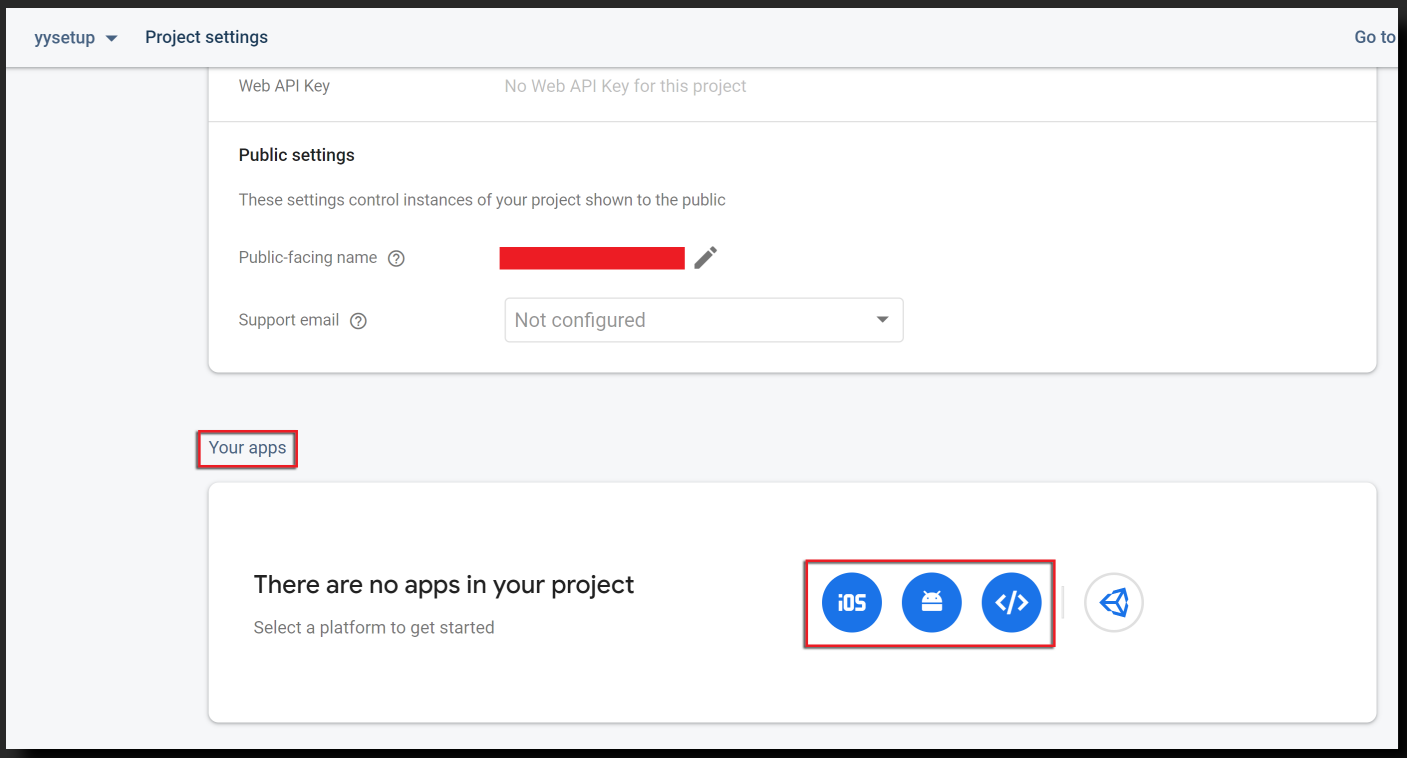
Web Setup

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.

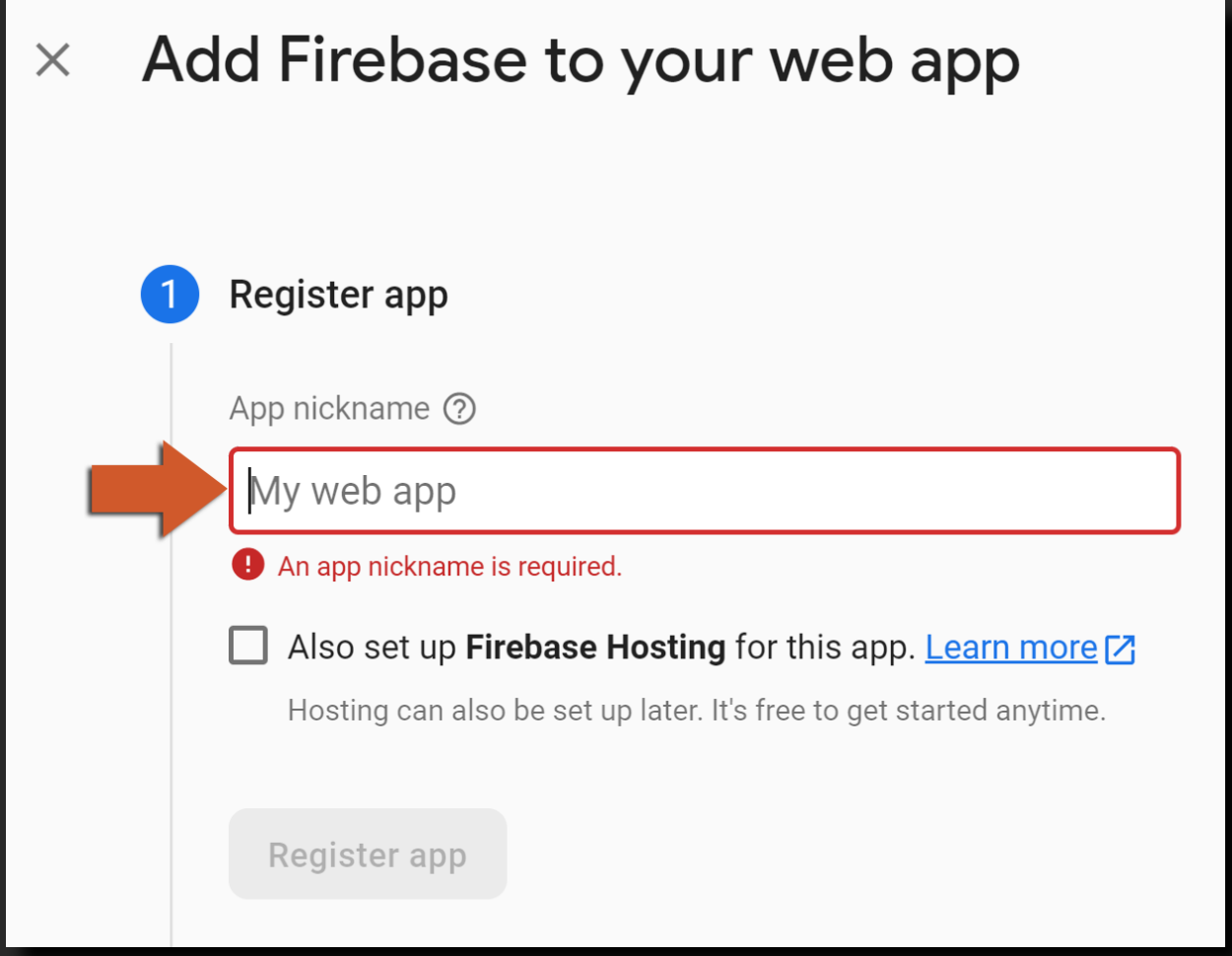
1. Click the **Settings** icon (next to Project Overview) and then **Project settings:**



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



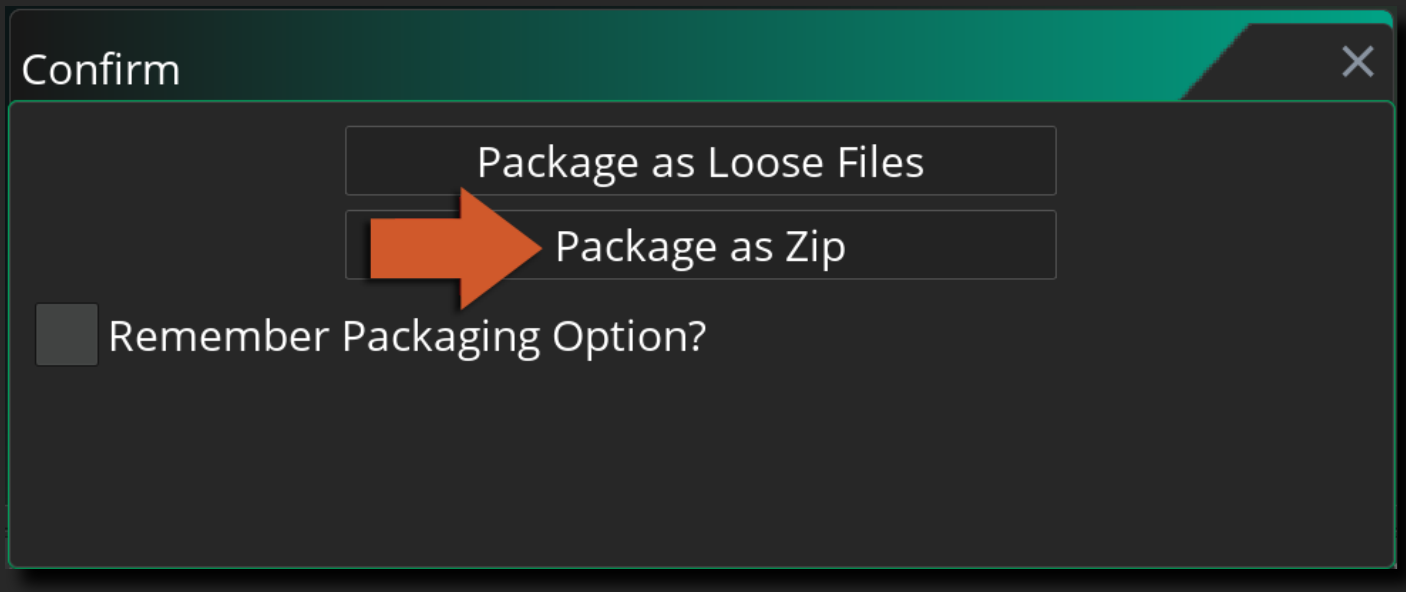
3. Enter your App nickname (required):



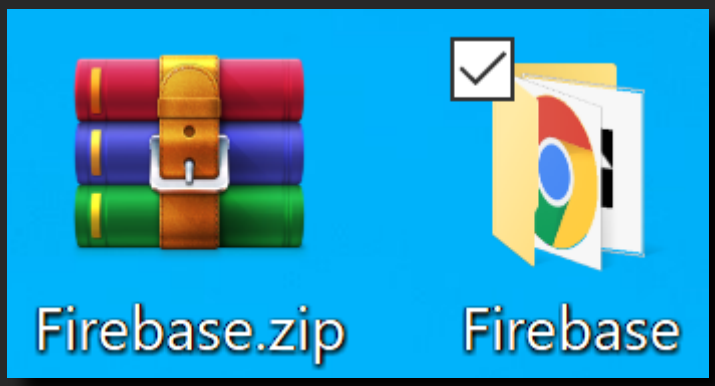
4. On this screen, just copy the **firebaseConfig** struct:

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

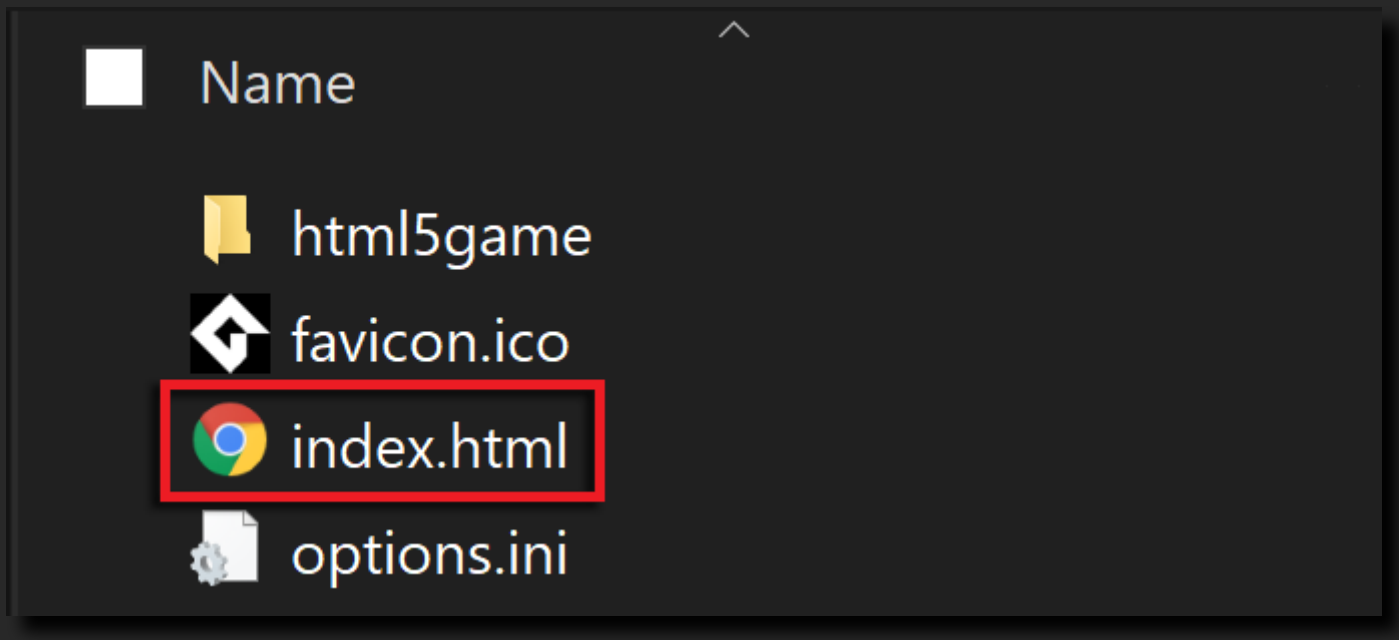
6. Choose the **Package As Zip** option:



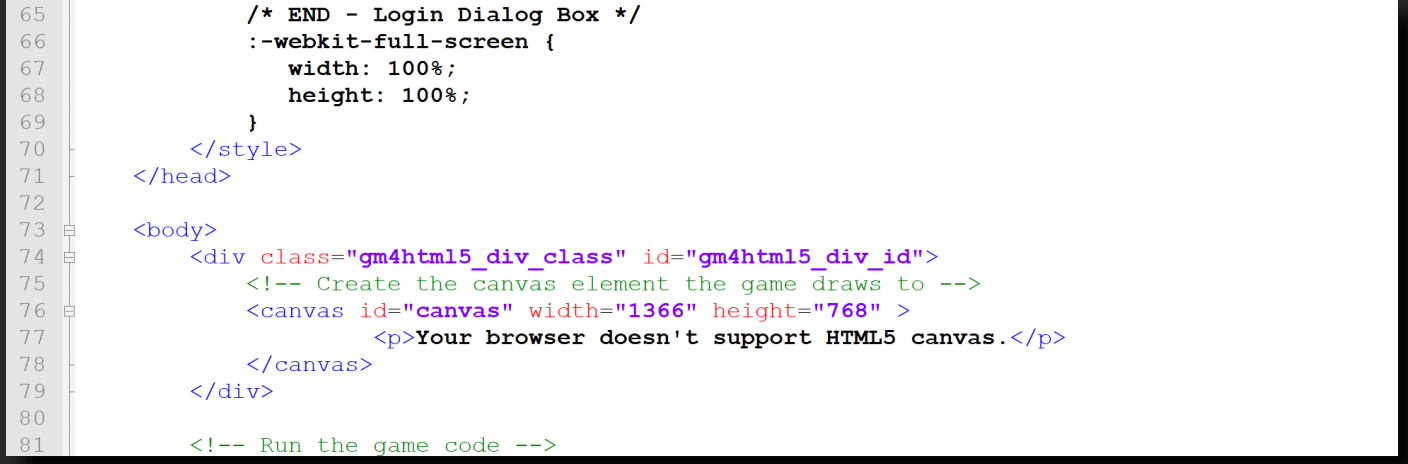
7. Locate the created package and **extract it** into a folder.



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).



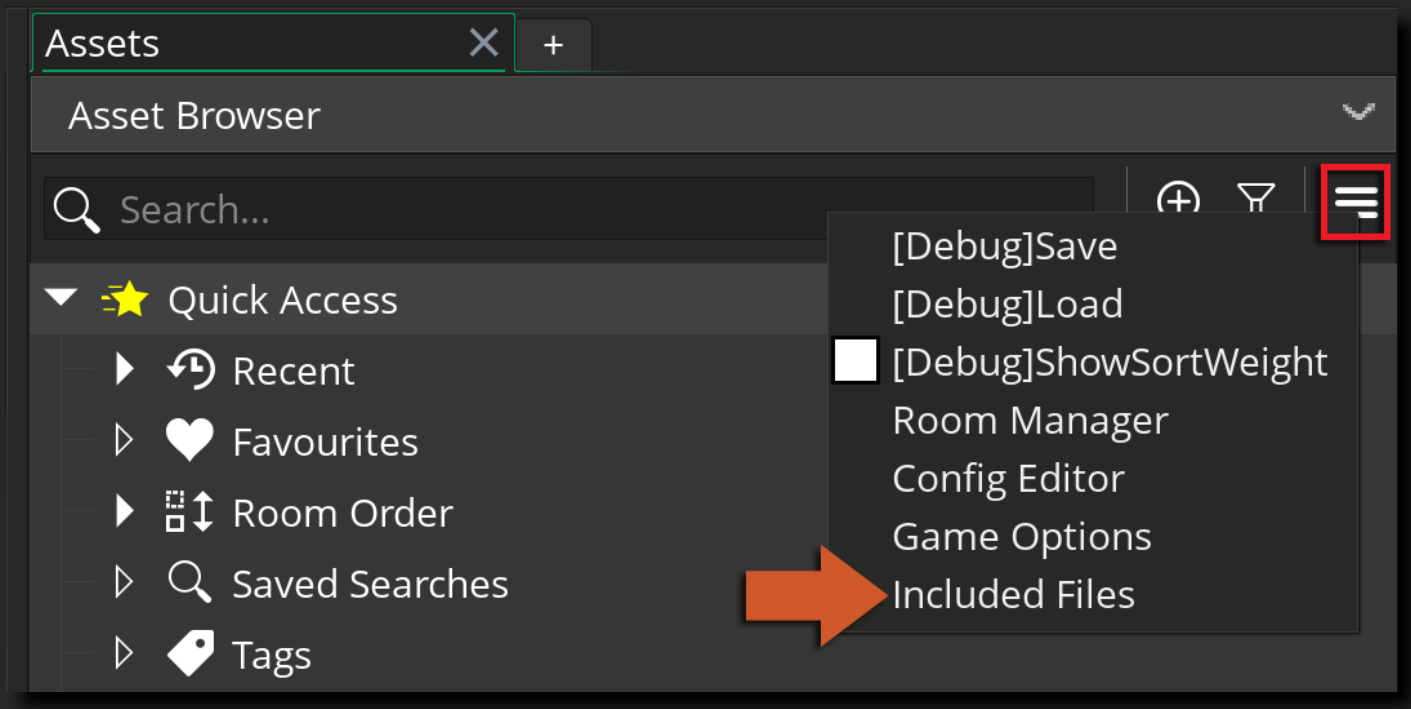
10. Now we need to add the following code between the `</head>` and `<body>` tags (line 72 in the **html.index** image above):

```
<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>
```

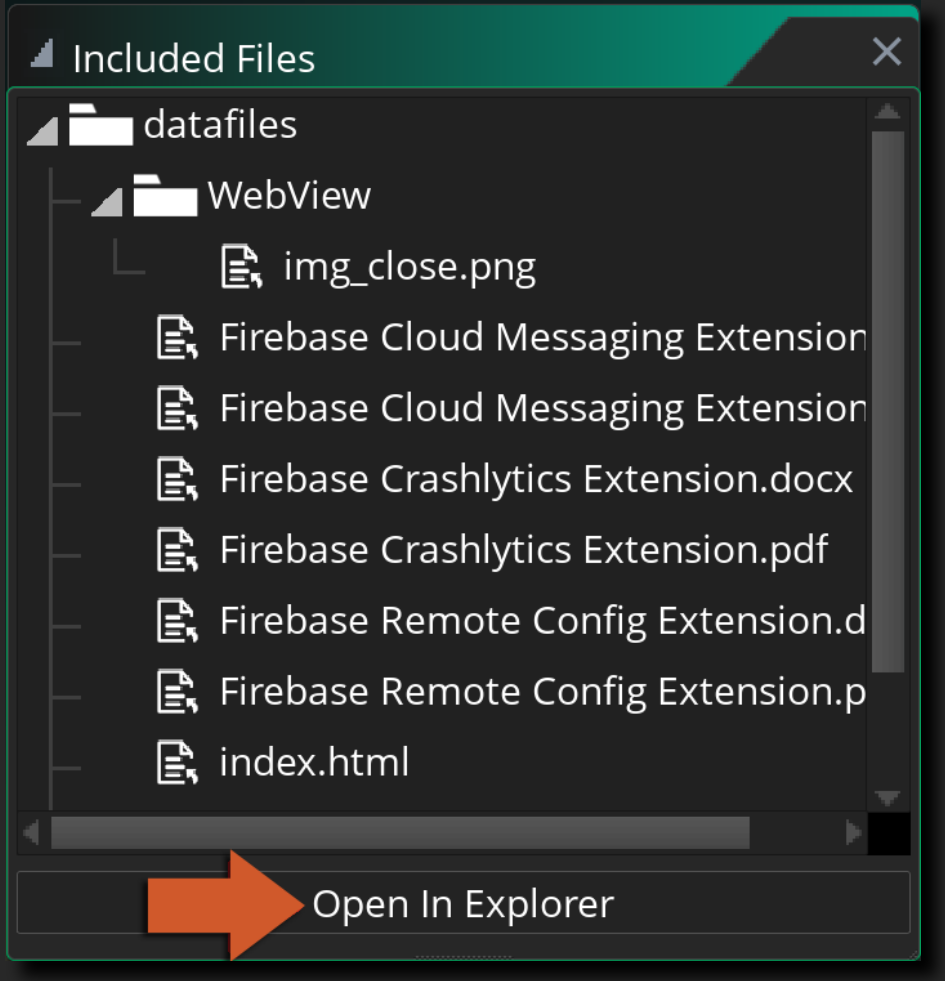
11. Replace the **const firebaseConfig** part with the one copied in step 4:

```
                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">
```

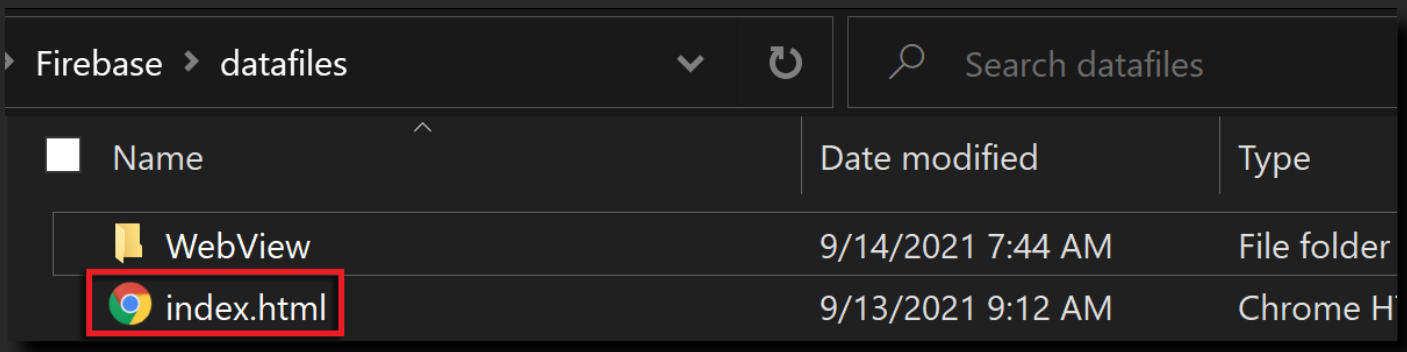
12. Go back into GameMaker and open your **Included Files** folder.



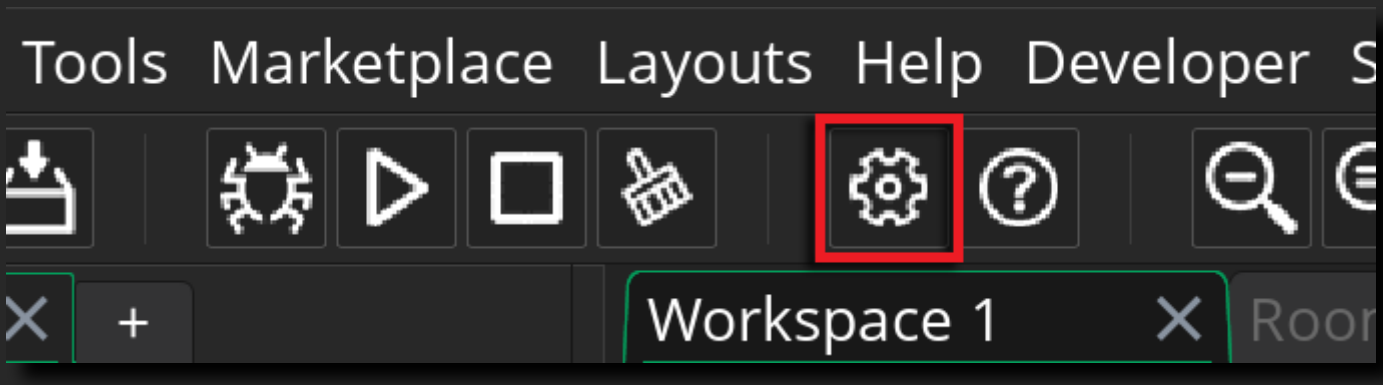
13. Press the **Open in Explorer** button:



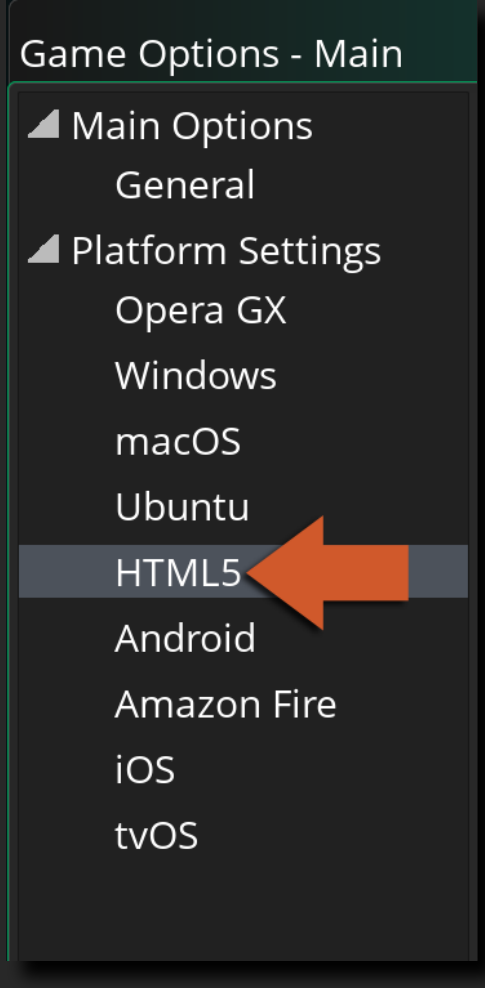
14. Place your **index.html** file inside the folder that opens (/datafiles).



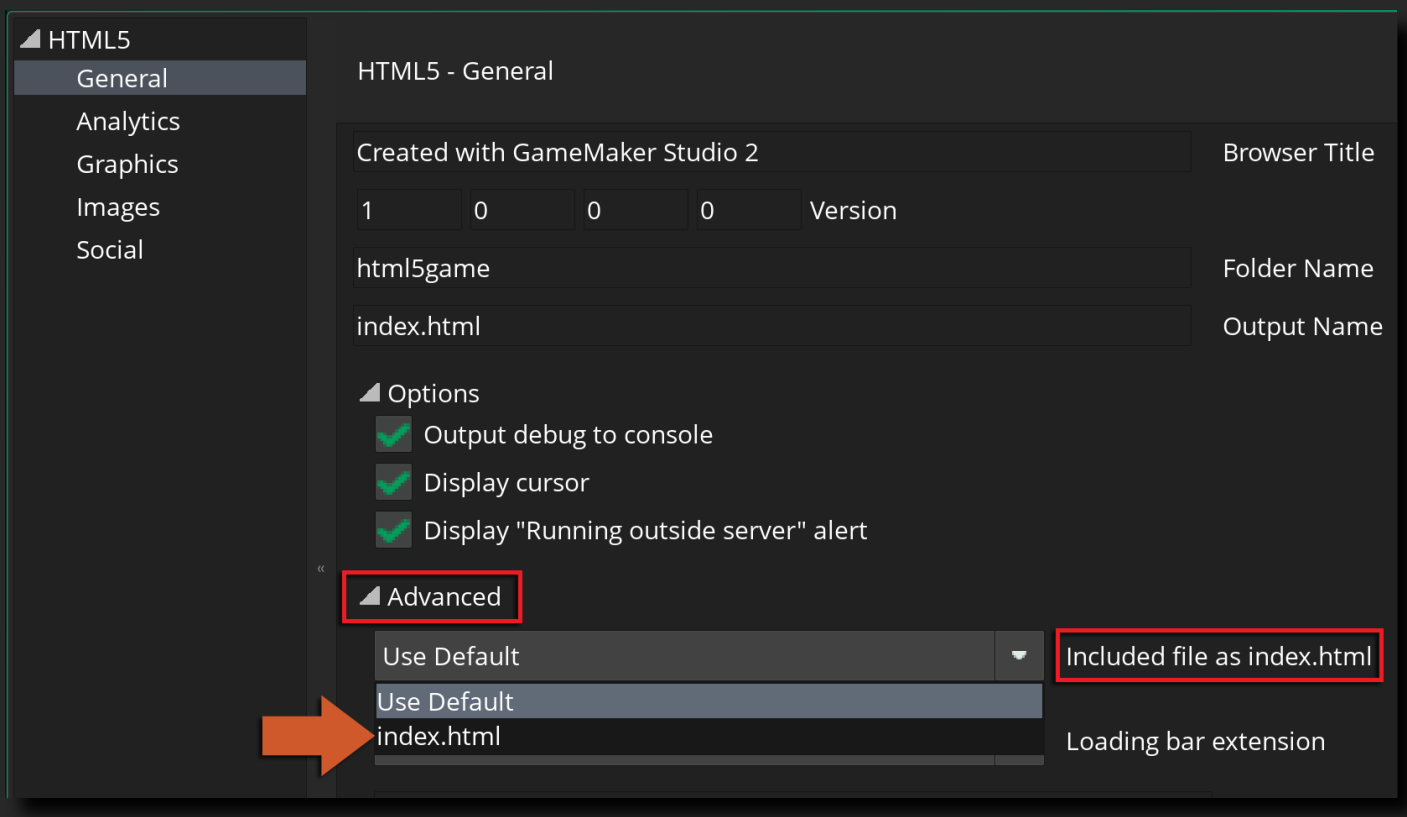
15. Back in GameMaker, click on the **Game Options button.**



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).

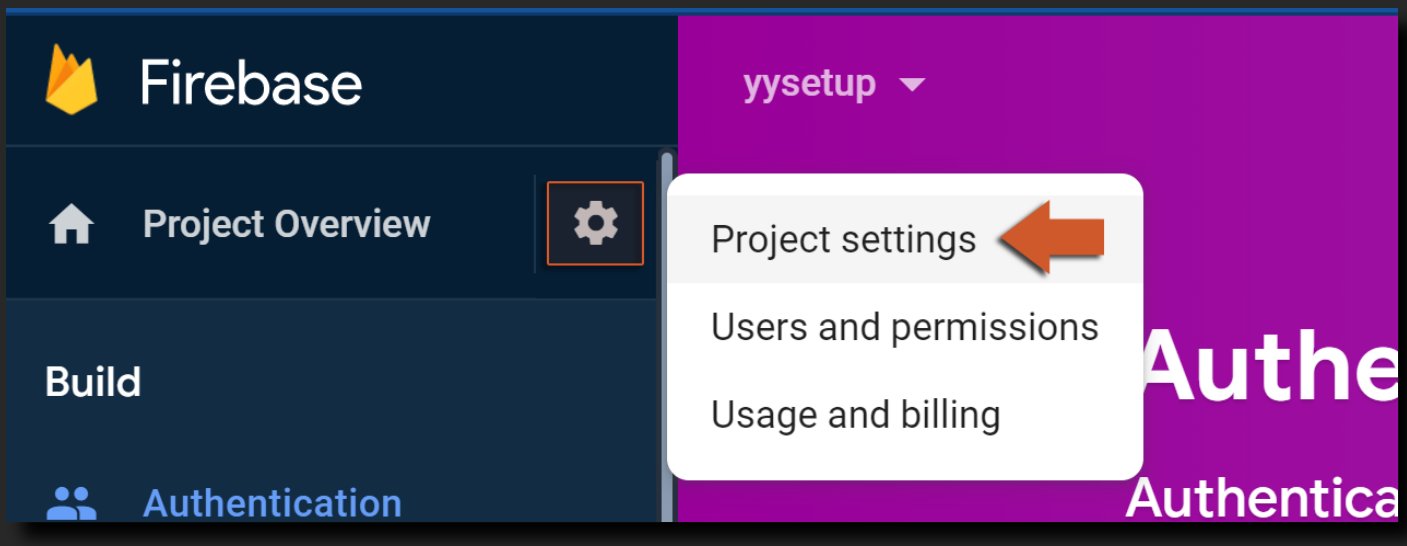


18. Press **Apply** and the main setup for all Firebase Web modules is finished!

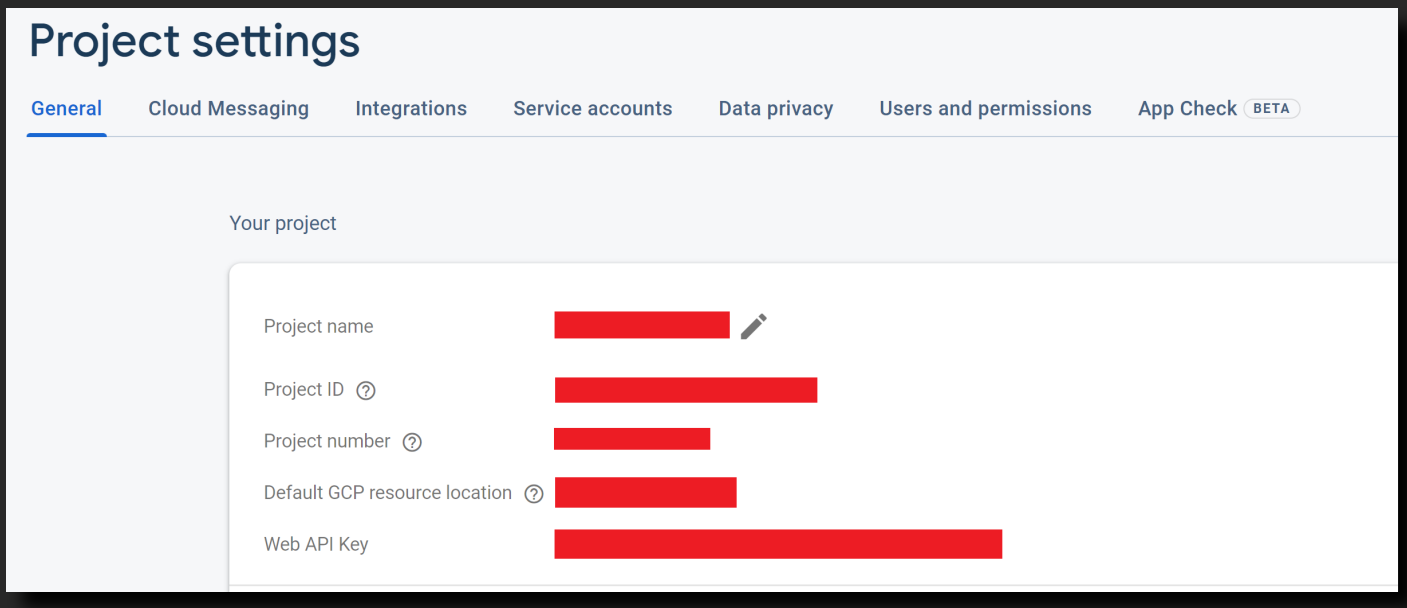
REST API Setup

This setup is necessary for syncing the Firebase Authentication console with the REST API implementation.

1. On your Firebase console, click on the **Settings** icon (next to **Project Overview**) and then on **Project settings:**



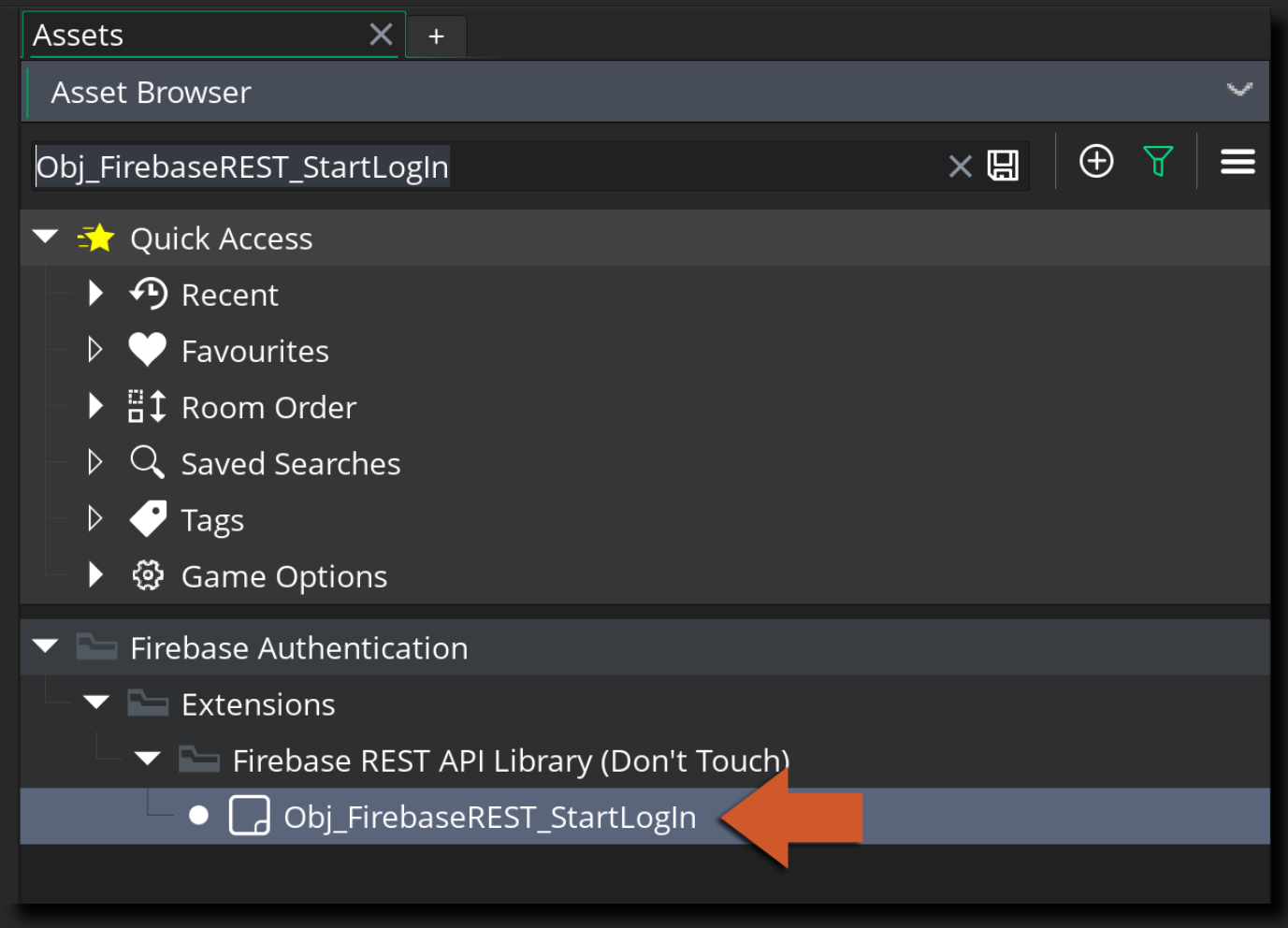
2. Now copy your **Web API Key** (shown at the bottom):



3. In your GameMaker Studio project, open the script **FirebaseAuthentication_Init** and paste your **Web API Key** in the macro **Firebase_WebAPIKey** (keep it inside the quotes):

```
#macro Firebase_WebAPIKey ""

#macro FirebaseAuthentication_LibraryOptions_SDKOnly "SDKs_Only"
#macro FirebaseAuthentication_LibraryOptions_SDKWhenAvaiable "SDKs_When_Available"
#macro FirebaseAuthentication_LibraryOptions_RESTOnly "REST_API_Only"

#macro FirebaseAuthentication_Library FirebaseAuthentication_LibraryOptions_SDKWhenAvaiable
```

4. Add an instance of **Obj_FirebaseREST_StartLogIn** in your very first room.



5. Now you have Firebase Authentication on all your REST API exports!

Apple Authentication

This guide will get you up and running using Apple authentication.

Prerequisites

These are the requirements for this setup:

- **Apple SignIn** extension (download from the **marketplace**)

Parameters

You will need to obtain these parameters:

- **token:** Obtained from the callback of `AppleSignIn_CrossPlatform_AuthoriseUser` function (included in the Apple SignIn extension)
- **provider:** `"apple.com"`
- **token_kind:** `"id_token"`
- **redirect_uri:** `"https://***.firebaseapp.com/__/auth/handler"`, this value can be obtained from the **Firebase Console** → **Authentication** → **Sign-In Method** → **Sign-In Providers** → **Apple**

Functions

The following functions are provided for signing-in / linking / re-authenticating the user:

- **FirebaseAuthentication_SignIn_OAuth** (if the user is logged out)
- **FirebaseAuthentication_LinkWithOAuthCredential** (if the user is logged in but has no provider)
- **FirebaseAuthentication_ReauthenticateWithOAuth** (if the user is logged in and already linked to the Apple provider)

Custom Authentication

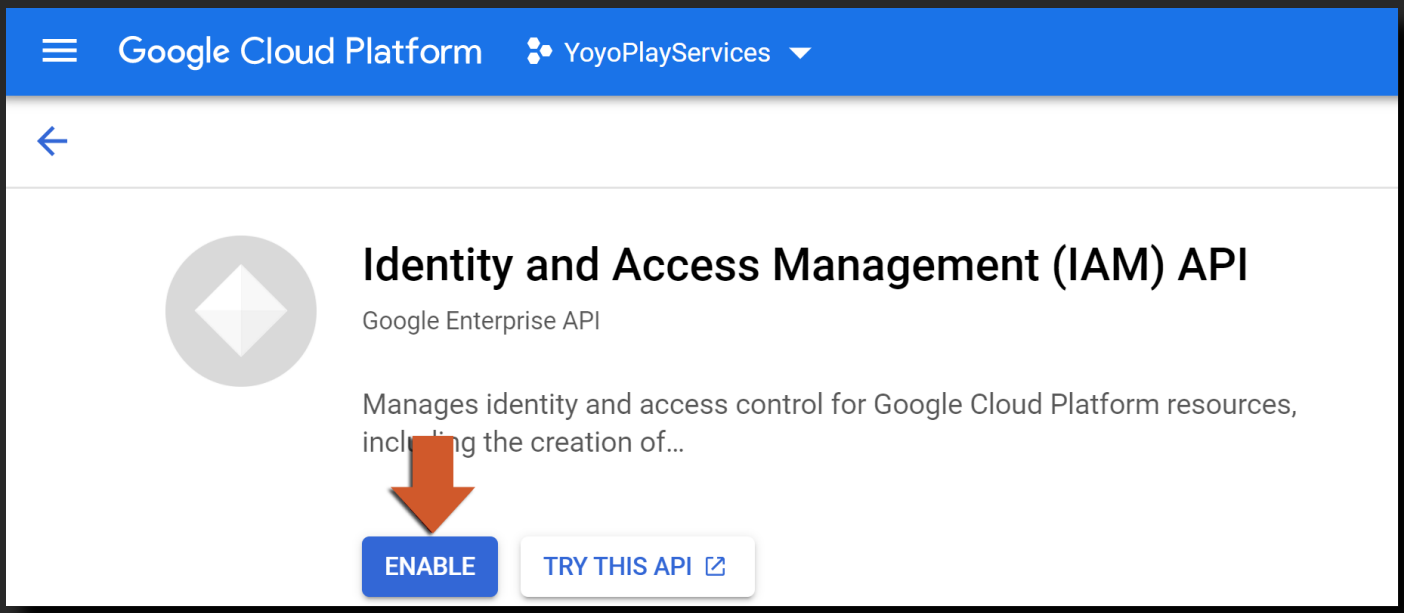
Custom Authentication gives to you the ability to sign in using any string as uid and is a really powerful feature, however it does mean that you will be responsible for creating your own unique ids (or using uids from other providers) and encoding/decoding your ids.

> **IMPORTANT** You are required to upgrade to the Blaze plan on Firebase to be able to use this feature.

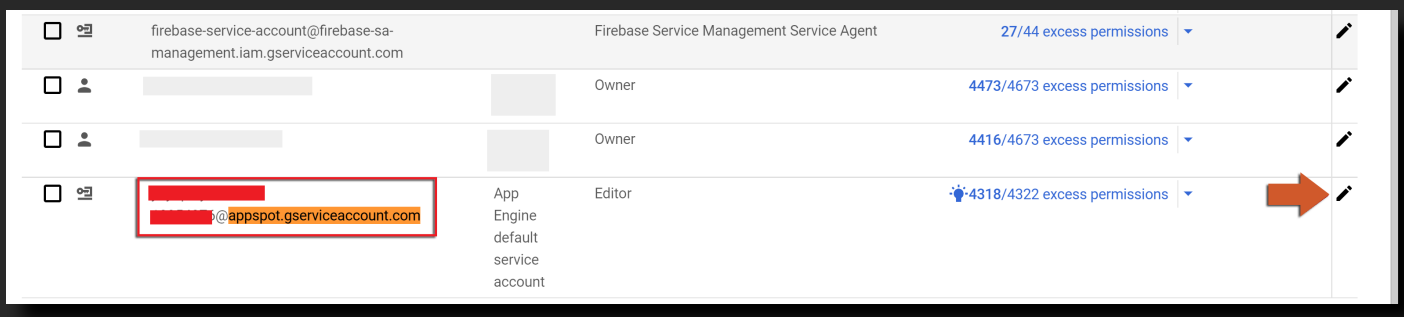
Prerequisites

The following are the requirements for this setup:

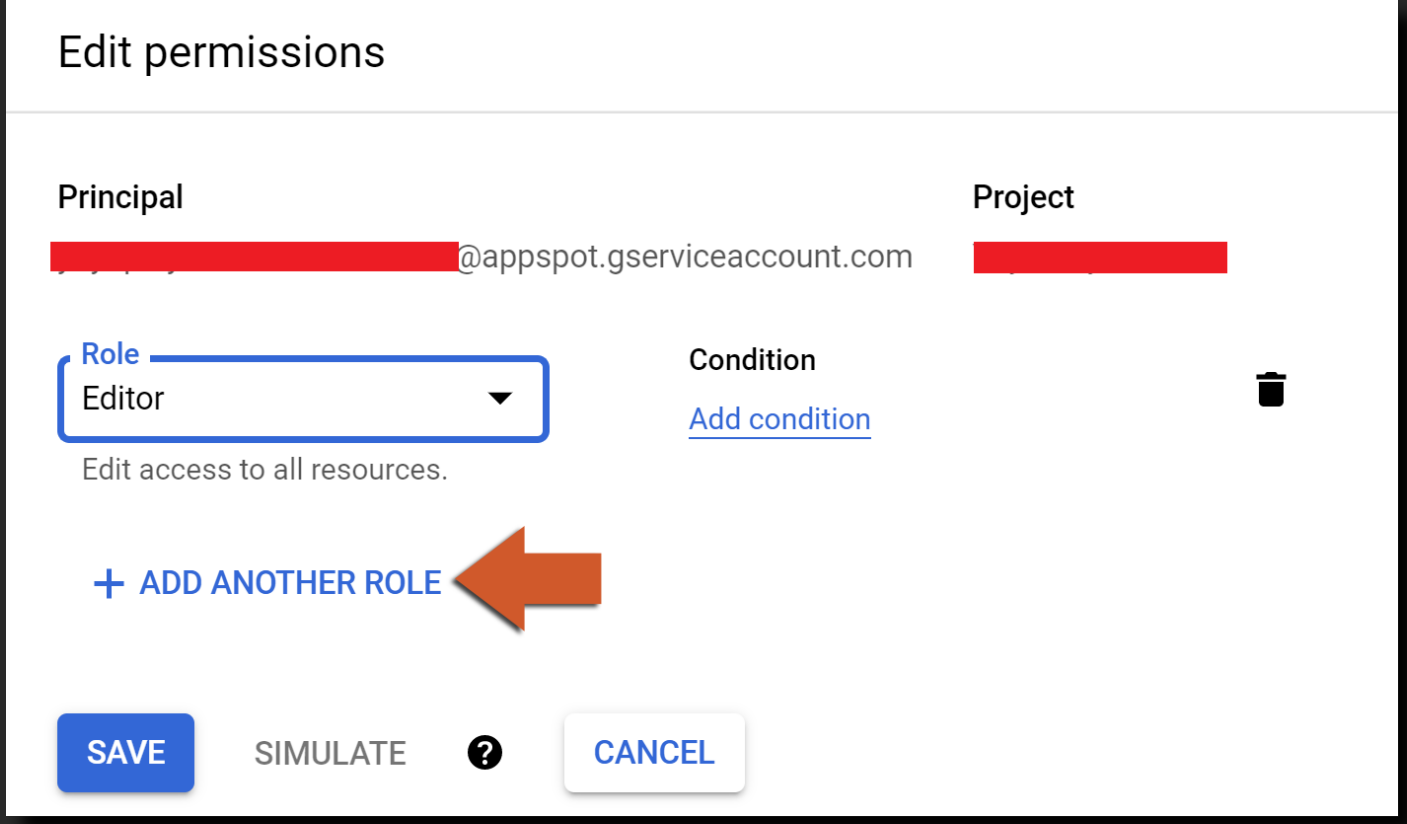
1. Enable **Identity and Access Management (IAM) API** on the **Google Cloud** console.



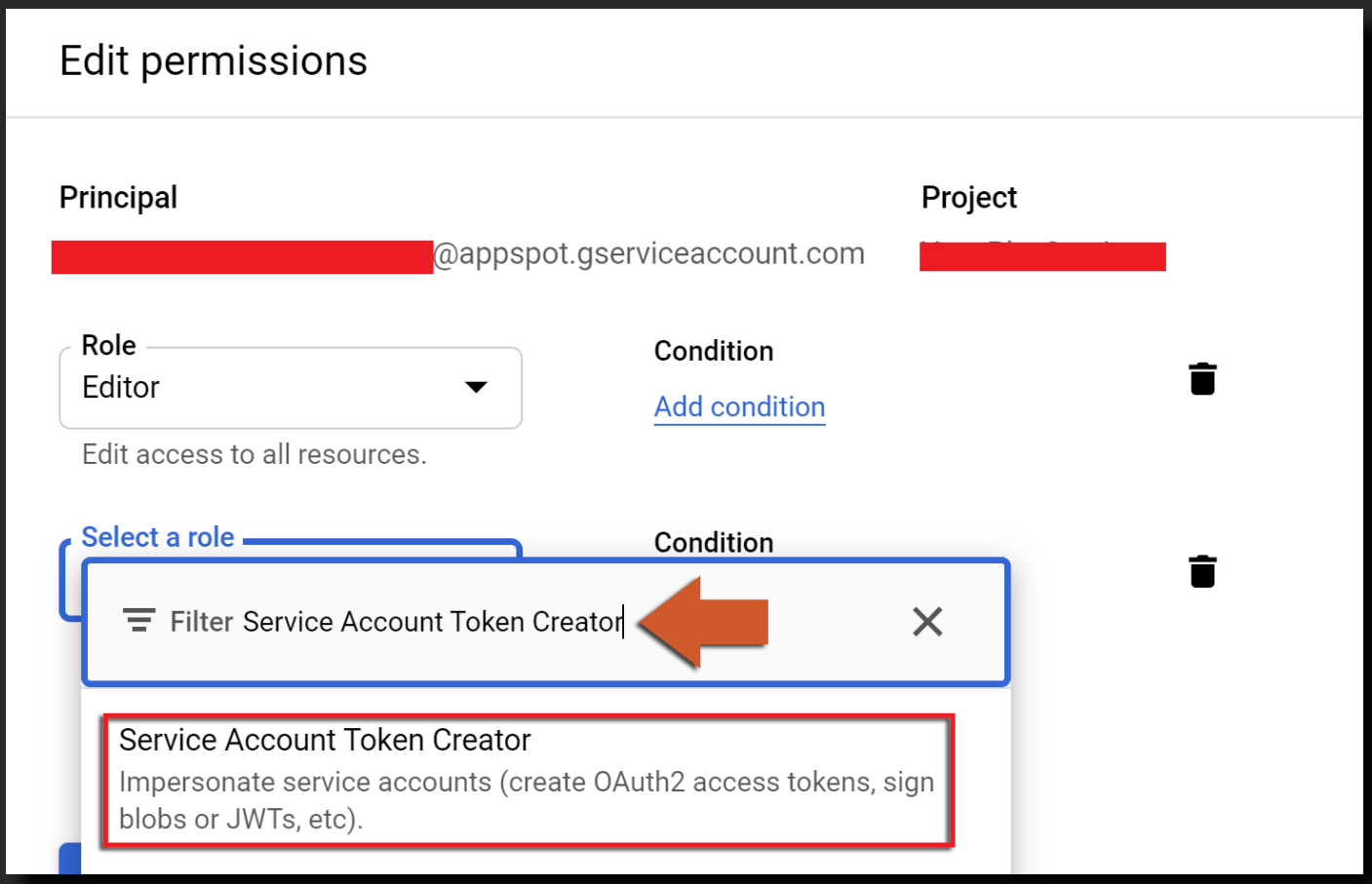
2. Go to **IAM & Admin**, search for a service named `****@appspot.gserviceaccount.com` (where `****` is your Firebase project's **Project ID**) and press the **Edit** icon.



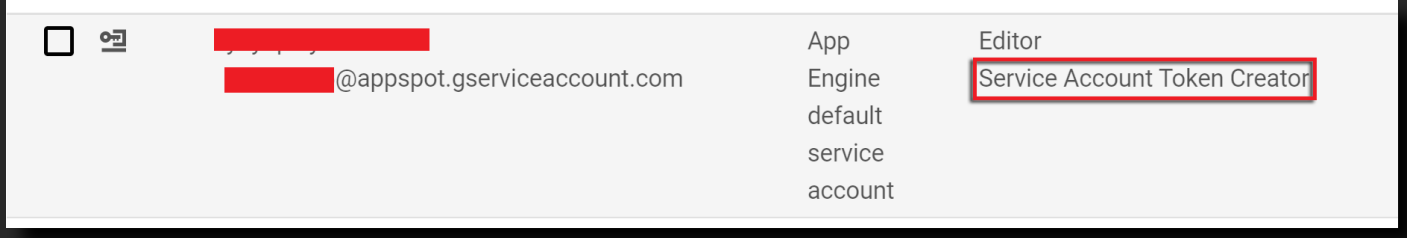
3. On the editor pop-up, click on the "**ADD ANOTHER ROLE**" button:



4. Select the **Service Account Token Creator** role:



5. This is how your services should look like now:



6. Now you are allowed to create new custom accounts in the Firebase Authentication system, however you will need a server running for creating **tokens** (for this step we will use the **Firebase Cloud Functions** extension) with the following code:

```
exports.customSignUp = functions.https.onRequest((req, res) =>
{
    cors(req, res, () =>
    {
        //And decode here (your uid should be encoded on you GMS project and decoded
here)
        let uid = req.body.uid;

        // Here are some official documentation links on how to create custom tokens
        // https://firebase.google.com/docs/auth/admin/create-custom-tokens
        // https://firebase.google.com/docs/reference/admin/node/
admin.auth.Auth-1#createcustomtoken

        admin.auth().createCustomToken(uid).then((customToken) =>
        {
            res.status(200).send({"customToken":customToken});
            return true;
        })
        .catch((error) =>
        {
            res.status(400).send({"message":error});
            return false;
        });
    });
});
```

Read how to create and deploy functions in the **Firebase Cloud Functions** extension documentation.

Parameters

You will need to obtain the following parameters:

- **token:** Obtained from the callback using `http_request` to the `customSignUp` API method created above.

Functions

This is the function provided for signing in the user:

- **FirebaseAuthentication_SignInWithCustomToken** (if the user is logged out)

Email Authentication

This guide will get you up and running using Email authentication.

Functions

The following functions are provided for signing-in / linking / re-authenticating the user:

- **FirebaseAuthentication_SignUp_Email** (if the user is not signed up)
- **FirebaseAuthentication_SendEmailVerification**
- **FirebaseAuthentication_SendPasswordResetEmail**
- **FirebaseAuthentication_SignIn_Email** (if the user is logged out)
- **FirebaseAuthentication_LinkWithEmailPassword** (if the user is logged in but has no provider)
- **FirebaseAuthentication_ReauthenticateWithEmail** (if the user is logged in and already linked to the email provider)

Facebook Authentication

This guide will get you up and running using Facebook authentication.

INFO The authentication works with both SDK and REST API versions.

Prerequisites

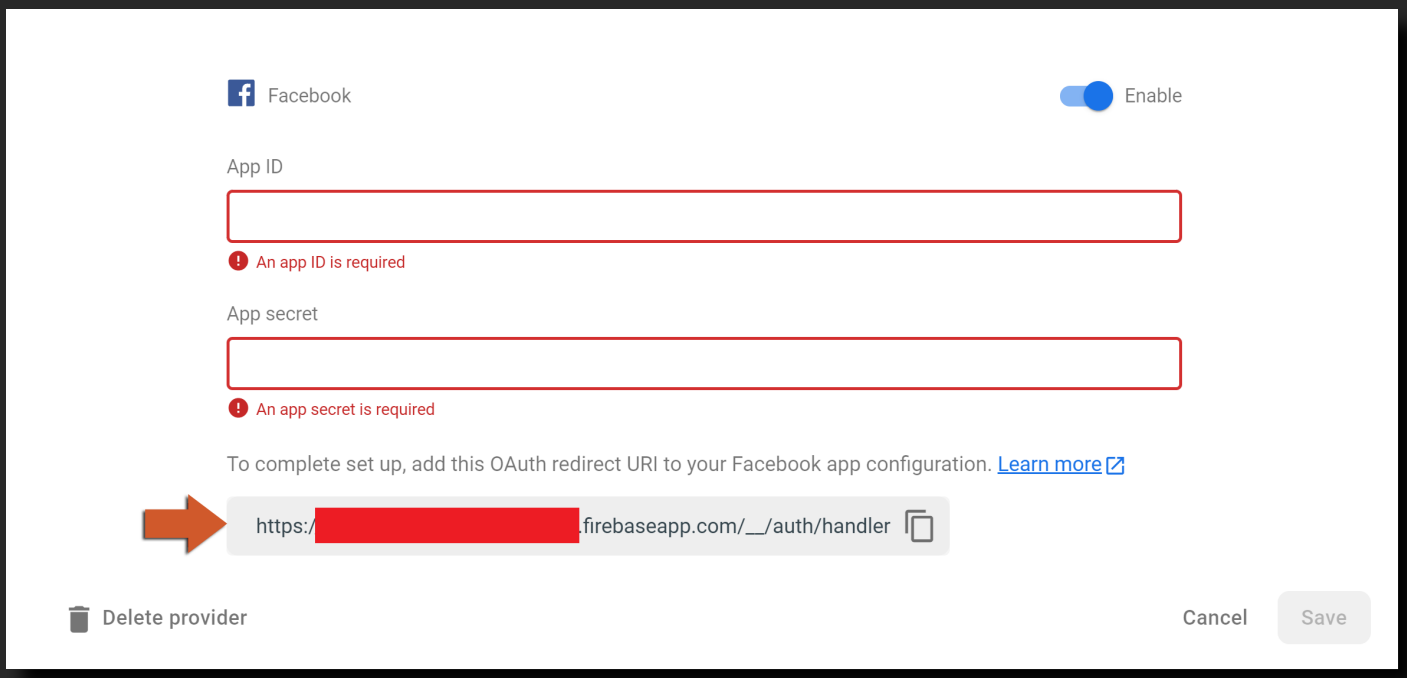
These are the requirements for this setup:

- Facebook Extension (download from the marketplace)

Parameters

You will need to obtain these parameters:

- **token:** Obtained from the call to the `fb_asscesstoken` function (included in the Facebook Extension)
- **provider:** `"facebook.com"`
- **token_kind:** `"access_token"`
- **redirect_uri:** This value can be obtained from **Firebase Console** → **Authentication** → **Sign-In Method** → **Sign-In Providers** → **Facebook** (see the image below)



Functions

The following functions are provided for signing-in / linking / re-authenticating the user:

- **FirebaseAuthentication_SignIn_OAuth** (if the user is logged out)
- **FirebaseAuthentication_LinkWithOAuthCredential** (if the user is logged in but has no provider)
- **FirebaseAuthentication_ReauthenticateWithOAuth** (if the user is logged in and already linked to the phone provider)

GameCenter Authentication

This guide will get you up and running using **GameCenter** authentication.

> **IMPORTANT** The SDK is required for this authentication meaning it will only work for iOS targets.

Prerequisites

These are the requirements for this setup:

- **GameCenter** extension (download from the **marketplace**)
- Log in using the `GameCenter_LocalPlayer_Authenticate` function

Functions

The following functions are provided for signing-in / linking / re-authenticating the user:

- **SDKFirebaseAuthentication_SignIn_GameCenter** (if the user is logged out)
- **SDKFirebaseAuthentication_LinkWithGameCenter** (if the user is logged in but has no provider)
- **SDKFirebaseAuthentication_ReauthenticateWithGameCenter** (if the user is logged in and already linked to the GameCenter provider)

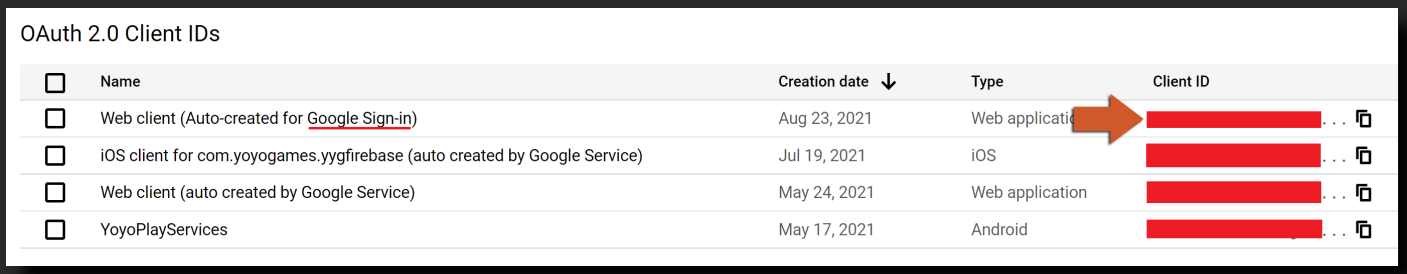
Google Authentication

This guide will get you up and running using Google authentication.

Prerequisites

These are the requirements for this setup:

- The **Google SignIn** extension (download from the **marketplace**).
- Get your **webClientID**, generated by **Google Sign-in** from the **Cloud Console** (as shown on the image below)



- In your project you need to call `GoogleSignIn_Show` with the project's **webClientID**.

Parameters

You will need to obtain these parameters:

- **token:** Obtained from the callback of the `GoogleSignIn_Show` function (included in the **Google SignIn** extension)
- **provider:** `"google.com"`
- **token_kind:** `"id_token"`
- **redirect_uri:** `""`

Functions

The following functions are provided for signing-in / linking / re-authenticating the user:

- **FirebaseAuthentication_SignIn_OAuth** (if the user is logged out)
- **FirebaseAuthentication_LinkWithOAuthCredential** (if the user is logged in but has no provider)
- **FirebaseAuthentication_ReauthenticateWithOAuth** (if the user is logged in and already linked to the phone provider)

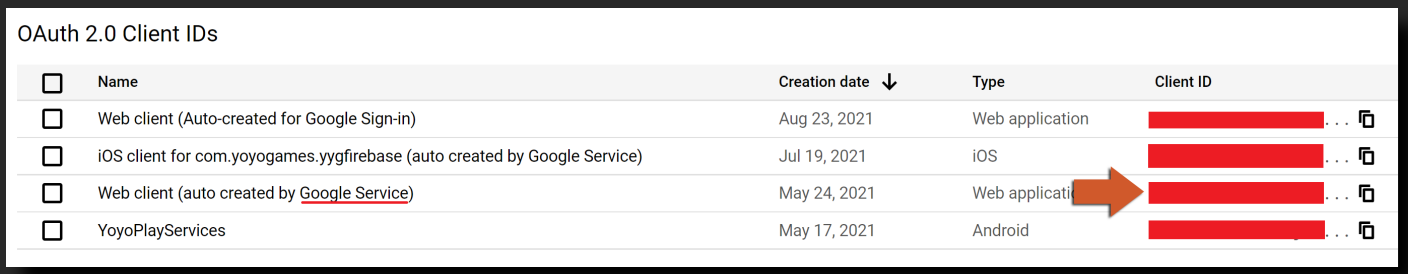
Google Play Services Authentication

This guide will get you up and running using Google Play Services authentication.

Prerequisites

These are the requirements for this setup:

- **Google Play Services** extension (download from the **marketplace**)
- Get your **webClientID**, generated by **Google Services** from the **Cloud Console** (as shown on the image below)



- In your project you need to call `GooglePlayServices_SetClientID` with the project's **webClientID** before calling `GooglePlayServices_StartSignInIntent`.

Parameters

You will need to obtain these parameters:

- **token:** Obtained from the call to the `GooglePlayServices_GetServerAuthCode` function (included in the Google Play Services extension)
- **provider:** `"playgames.google.com"`
- **token_kind:** `"serverAuthCode"`
- **redirect_uri:** `""`

Functions

The following functions are provided for signing-in / linking / re-authenticating the user:

- **FirebaseAuthentication_SignIn_OAuth** (if the user is logged out)
- **FirebaseAuthentication_LinkWithOAuthCredential** (if the user is logged in but has no provider)
- **FirebaseAuthentication_ReauthenticateWithOAuth** (if the user is logged in and already linked to the phone provider)

Phone Authentication

Phone authentication requires you to follow these 5 steps:

1. Get your **recaptchaSiteKey** from the **FirebaseAuthentication_RecaptchaParams** function callback.
2. Solve the reCAPTCHA puzzle, for this you will need to host a website (demonstration included in Included Files).
3. The previous step will return you a **recaptchaToken**, which needs to be passed into your game (in the example we use **Firebase Real Time Database** extension to achieve this).
4. Send the verification code to the user, using the function **FirebaseAuthentication_SendVerificationCode** and get the **sessionInfo** from the callback.
5. You should now have **SMS code** (request the user for it) and the **sessionInfo**.

Functions

The following functions are provided for signing-in / linking / re-authenticating the user:

- **FirebaseAuthentication_SignInWithPhoneNumber** (if the user is logged out)
- **FirebaseAuthentication_LinkWithPhoneNumber** (if the user is logged in but has no provider)
- **FirebaseAuthentication_ReauthenticateWithPhoneNumber** (if the user is logged in and already linked to the phone provider)

FirebaseAuthentication_ChangeDisplayName

This function changes a user's display name 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 call requires re-authentication.

Syntax:

```
FirebaseAuthentication_ChangeDisplayName(name)
```

Argument	Type	Description
name	string	The new display name to be assigned to the current user.

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_ChangeDisplayName"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_ChangeDisplayName("YoYoUser");
```

In the code above we initiate a task to change the display name of the user. The function call will then return a listener ID (`listenerId`) that can be used inside an Async Social event.

```
if (async_load[? "type"] == "FirebaseAuthentication_ChangeDisplayName")
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message("The display name was changed.");
    }
    else
    {
        show_debug_message("There was an error changing the display name.");
    }
}
```

The code above matches the response against the correct event **type**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_ChangeEmail

This function changes a user's email 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 call requires re-authentication.

Syntax:

```
FirebaseAuthentication_ChangeEmail(email)
```

Argument	Type	Description
email	string	The new email address to be assigned to the current user.

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_ChangeEmail"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_ChangeEmail("YoYoUser@yoyo.com");
```

In the code above we initiate a task to change the email of the user. The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "type"] == "FirebaseAuthentication_ChangeEmail")
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message("The email was changed.");
    }
    else
    {
        show_debug_message("There was an error changing the email.");
    }
}
```

The code above matches the response against the correct event **type**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_ChangePassword

This function changes a user's password 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 call requires re-authentication.

Syntax:

```
FirebaseAuthentication_ChangePassword(password)
```

Argument	Type	Description
password	string	The new password for the current user account.

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_ChangePassword"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_ChangePassword("Password123!");
```

In the code above we initiate a task to change the password of the user. The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "type"] == "FirebaseAuthentication_ChangePassword")
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message("The email was changed.");
    }
    else
    {
        show_debug_message("There was an error changing the email.");
    }
}
```

The code above matches the response against the correct event **type**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_ChangePhotoURL

This function changes a user's photo URL 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 call requires re-authentication.

Syntax:

```
FirebaseAuthentication_ChangePhotoURL(url)
```

Argument	Type	Description
url	string	The new photo url to be assign to the current user account.

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_ChangePhotoURL"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_ChangePhotoURL(global.photoURL);
```

In the code above we initiate a task to change the photo URL of the user to a new value store in a global variable (`global.photoURL`). The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "type"] == "FirebaseAuthentication_ChangePhotoURL")
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message("The photo URL was changed.");
    }
    else
    {
        show_debug_message("There was an error changing the photo URL.");
    }
}
```

The code above matches the response against the correct event **type**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_DeleteAccount

This function deletes the current user account 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 call requires re-authentication.

Syntax:

```
FirebaseAuthentication_DeleteAccount()
```

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_DeleteAccount"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_DeleteAccount();
```

In the code above we initiate a task to delete the current user account. The function call will then return a listener ID (`listenerId`) that can be used inside an Async Social event.

```
if (async_load[? "type"] == "FirebaseAuthentication_DeleteAccount")
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message("The account was deleted.");
    }
    else
    {
        show_debug_message("There was an error deleting the account.");
    }
}
```

The code above matches the response against the correct event **type**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_GetDisplayName

This function returns the display name of the current user.

Syntax:

```
FirebaseAuthentication_GetDisplayName()
```

Returns:

```
String
```

Example:

```
var displayName = FirebaseAuthentication_GetDisplayName();
```

The code above will retrieve the current user's display name and store it in a local variable.

FirebaseAuthentication_GetEmail

This function returns the email of the current user.

Syntax:

```
FirebaseAuthentication_GetEmail()
```

Returns:

```
String
```

Example:

```
var email = FirebaseAuthentication_GetEmail();
```

The code above will retrieve the current user's email and store it in a local variable.

FirebaseAuthentication_GetEmailVerified

This function returns whether or not the email of the current user is verified.

Syntax:

```
FirebaseAuthentication_GetEmailVerified()
```

Returns:

```
Boolean
```

Example:

```
var emailVerified = FirebaseAuthentication_GetEmailVerified();

if (!emailVerified)
{
    listenerId = FirebaseAuthentication_SendEmailVerification();
}
```

The code above will check if the current user's email is verified, and if it's not, it'll call FirebaseAuthentication_SendEmailVerification to start the verification process.

FirebaseAuthentication_GetIdToken

This function requests a callback to get the current **idToken** 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:

```
FirebaseAuthentication_GetIdToken()
```

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_GetIdToken"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
value	string	The idToken (if status == 200) OPTIONAL
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_GetIdToken();
```

In the code above we request for the current session's **idToken**. The function call will then return a listener ID (`listenerId`) that can be used inside an Async Social event.

```
if (async_load[? "type"] == "FirebaseAuthentication_ReauthenticateWithEmail")
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message(json_parse(async_load[? "value"]));
    }
    else
    {
        show_debug_message(async_load[? "errorMessage"]);
    }
}
```

The code above matches the response against the correct event **type**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_GetLocalId

This function returns the local identifier (uid) of the current user.

> NOTE This function returns the same value as FirebaseAuthentication_GetUID.

Syntax:

```
FirebaseAuthentication_GetLocalId()
```

Returns:

```
String
```

Example:

```
var uid = FirebaseAuthentication_GetLocalId();
```

The code above will retrieve the local identifier (uid) of the current user and store it in a local variable.

FirebaseAuthentication_GetPhotoUrl

This function returns the URL of the current user's photo.

Syntax:

```
FirebaseAuthentication_GetPhotoUrl()
```

Returns:

```
String
```

Example:

```
var photoUrl = FirebaseAuthentication_GetPhotoUrl();
```

The code above will retrieve current user's photo URL and store it in a local variable.

FirebaseAuthentication_GetProviderUserInfo

This function returns an array of structs with the information of each provider of the current user (or an empty string if unavailable).

Below is a sample of the returned information string parsed as an array (using the json_parse function) for a user authenticated with Apple, Facebook, Google Play Games and Phone.


```
[
    {
        "email":"******@privaterelay.appleid.com",
        "providerId":"apple.com",
        "rawId":"**********",
        "federatedId":"**********"
    },
    {
        "displayName":"*********",
        "photoUrl":"https:**********",
        "providerId":"facebook.com",
        "rawId":"*********",
        "federatedId":"********"
    },
    {
        "displayName":"****",
        "photoUrl":"https:**********",
        "providerId":"playgames.google.com",
        "rawId":"**********",
        "federatedId":"**********"
    },
    {
        "phoneNumber":"**********",
        "providerId":"phone"
    }
]
```


Syntax:

```
FirebaseAuthentication_GetProviderUserInfo()
```

Returns:

```
String (JSON formatted string representation of an array)
```

Example:

```
var providerUserInfo = FirebaseAuthentication_GetProviderUserInfo()
```

The code above will return a JSON formatted string of an array containing provider information.

FirebaseAuthentication_GetUID

This function returns the unique identifier (uid) of the current user.

> NOTE This function returns the same value as FirebaseAuthentication_GetLocalId.

Syntax:

```
FirebaseAuthentication_GetUID()
```

Returns:

```
String
```

Example:

```
var uid = FirebaseAuthentication_GetUID();
```

The code above will retrieve the unique identifier (uid) of the current user and stores it in a local variable.

FirebaseAuthentication_GetUserData_raw

This function returns a raw user representation as a JSON formatted string.

Below is a sample of the returned information string parsed as a struct (using the json_parse function) with information for some of the supported providers: Apple, Facebook, Google Play Games and Phone.


```
{
   "kind":"identitytoolkit#GetAccountInfoResponse",
   "users":[
      {
         "email":"**********",
         "localId":"***********8",
         "emailVerified":true,
         "phoneNumber":true,
         "providerUserInfo":[
            {
               "email":"******@privaterelay.appleid.com",
               "providerId":"apple.com",
               "rawId":"**********",
               "federatedId":"***********"
            },
            {
               "displayName":"*********",
               "photoUrl":"https:**********",
               "providerId":"facebook.com",
               "rawId":"*********",
               "federatedId":"********"
            },
            {
               "displayName":"****",
               "photoUrl":"https:***********",
               "providerId":"playgames.google.com",
               "rawId":"**********",
               "federatedId":"***********"
            },
            {
               "phoneNumber":"***********",
               "providerId":"phone"
            }
         ]
      }
   ]
}
```


NOTE It is recommended to use one of the `FirebaseAuthentication_Get*` functions to get the information you need.

Syntax:

```
FirebaseAuthentication_GetUserData_raw();
```

Argument	Description
arg	The argument to be passed in

Returns:

```
String (JSON formatted string representation of a struct)
```

Example:

```
var raw_user_data = FirebaseAuthentication_GetUserData_raw()
```

The code above will return a JSON formatted string of an array with raw user data.

FirebaseAuthentication_LinkWithEmailPassword

This function establishes a link between an email/password pair and the current user (used for Email based authentication). The returned value is 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:

```
FirebaseAuthentication_LinkWithEmailPassword(email, pass)
```

Argument	Type	Description
email	string	The email to link the current user to.
pass	string	The password to link to the current user account.

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_LinkWithEmailPassword"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
value	string	User raw data (if status == 200) OPTIONAL
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_LinkWithEmailPassword("super_user@example.com",
"myPass123!");
```

In the code above we attempt to link the current user with a email/password pair. The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "type"] == "FirebaseAuthentication_LinkWithEmailPassword")
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message(json_parse(async_load[? "value"]));
    }
    else
    {
        show_debug_message("There was an error linking email/pass to current user.");
    }
}
```

The code above matches the response against the correct event **type**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_LinkWithOAuthCredential

This function establishes a link between an OAuth credential and the current user (used for **Apple**, **Facebook**, **Google** and **GooglePlay** based authentications). The returned value is 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:

```
FirebaseAuthentication_LinkWithOAuthCredential(token, token_kind, provider,
[redirect_uri]);
```

Argument	Type	Description
token	string	The provider token.
token_kind	string	One of the provided strings `"id_token"`, `"access_token"` or `"serverAuthCode"`
provider	string	The provider identifier (e.g. `"apple.com"`, `"google.com"`, etc)
redirect_uri	string	OPTIONAL Only for REST API (can be checked on the Firebase Console for your provider)

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_LinkWithOAuthCredential"`
listener	real	The asynchronous listener ID.

status	real	The HTTP status response code (see reference)
value	string	User raw data (if status == 200) OPTIONAL
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_LinkWithOAuthCredential(token, token_kind,
provider, redirect_uri);
```

In the code above we attempt to link the current user with an OAuth credential. The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "type"] == "FirebaseAuthentication_LinkWithEmailPassword")
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message(json_parse(async_load[? "value"]));
    }
    else
    {
        show_debug_message("There was an error linking OAuth credential to current
user.");
    }
}
```

The code above matches the response against the correct event **type**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_LinkWithPhoneNumber

This function establishes a link between a phone number and the current user. The returned value is 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:

```
FirebaseAuthentication_LinkWithPhoneNumber(token, token_kind, provider);
```

Argument	Type	Description
phone	string	The phone number, including country number (ie.: `"+44123456789"`)
code	string	The code from the SMS.
session_info	string	Session info from the re-catcha puzzle.

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_LinkWithPhoneNumber"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
value	string	User raw data (if status == 200) OPTIONAL
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_LinkWithPhoneNumber(phone, code, sessionInfo);
```

In the code above we attempt to link the current user with a phone number. The function call will then return a listener ID (`listenerId`) that can be used inside an Async Social event.

```
if (async_load[? "type"] == "FirebaseAuthentication_LinkWithPhoneNumber")
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message(json_parse(async_load[? "value"]));
    }
    else
    {
        show_debug_message("There was an error linking phone number to current
user.");
    }
}
```

The code above matches the response against the correct event **type**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_ReauthenticateWithEmail

This function re-authenticates the current user using an email/password (used for Email based authentication). The returned value is 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:

```
FirebaseAuthentication_ReauthenticateWithEmail(email, pass)
```

Argument	Type	Description
email	string	The email to link the current user to.
pass	string	The password to link to the current user account.

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_ReauthenticateWithEmail"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
value	string	User raw data (if status == 200) OPTIONAL
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_ReauthenticateWithEmail("super_user@example.com",
"myPass123!");
```

In the code above we attempt to re-authenticate the current user with a email/password pair. The function call will then return a listener ID (`listenerId`) that can be used inside an Async Social event.

```
if (async_load[? "type"] == "FirebaseAuthentication_ReauthenticateWithEmail")
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message(json_parse(async_load[? "value"]));
    }
    else
    {
        show_debug_message(async_load[? "errorMessage"]);
    }
}
```

The code above matches the response against the correct event **type**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_ReauthenticateWithOAuth

This function re-authenticates the current user using an OAuth credential (used for **Apple**, **Facebook**, **Google** and **GooglePlay** based authentications). The returned value is 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:

```
FirebaseAuthentication_ReauthenticateWithOAuth(token, token_kind, provider,
[redirect_uri]);
```

Argument	Type	Description
token	string	The provider token.
token_kind	string	One of the provided strings `"id_token"` or `"access_token"` or `"serverAuthCode"`
provider	string	The provider identifier (ie.: `"apple.com"`, `"google.com"`, etc)
redirect_uri	string	OPTIONAL Only for REST API (can be checked on the Firebase Console for your provider)

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_ReauthenticateWithOAuth"`
listener	real	The asynchronous listener ID.

status	real	The HTTP status response code (see reference)
value	string	User raw data (if status == 200) OPTIONAL
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_ReauthenticateWithOAuth(token, token_kind,
provider, redirect_uri);
```

In the code above we attempt to re-authenticate the current user with an OAuth credential. The function call will then return a listener ID (`listenerId`) that can be used inside an Async Social event.

```
if (async_load[? "type"] == "FirebaseAuthentication_ReauthenticateWithOAuth")
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message(json_parse(async_load[? "value"]));
    }
    else
    {
        show_debug_message(async_load[? "errorMessage"]);
    }
}
```

The code above matches the response against the correct event **type**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_ReauthenticateWithPhoneNumber

This function re-authenticates the current user using a phone number. The returned value is 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:

```
FirebaseAuthentication_ReauthenticateWithPhoneNumber(phone, code, session_info);
```

Argument	Type	Description
phone	string	The phone number, including country number (ie.: `"+44123456789"`)
code	string	The code from the SMS.
session_info	string	Session info from the re-catcha puzzle.

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_ReauthenticateWithPhoneNumber"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
value	string	User raw data (if status == 200) OPTIONAL
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_ReauthenticateWithPhoneNumber(phone, code,
sessionInfo);
```

In the code above we try to re-authenticate the current user with a phone number. The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "type"] == "FirebaseAuthentication_ReauthenticateWithPhoneNumber")
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message(json_parse(async_load[? "value"]));
    }
    else
    {
        show_debug_message(async_load[? "errorMessage"]);
    }
}
```

The code above matches the response against the correct event **type**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_RecaptchaParams

This function requests a **recaptchaSiteKey** 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:

```
FirebaseAuthentication_RecaptchaParams()
```

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_RecaptchaParams"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
value	string	A json formatted string with **recaptchaSiteKey** (if status == 200) OPTIONAL
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_RecaptchaParams();
```

In the code above we attempt to get the recaptcha parameters that can be used for phone number authentication (see **FirebaseAuthentication_LinkWithPhoneNumber**). The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "type"] == "FirebaseAuthentication_RecaptchaParams")
{
    if (async_load[? "status"] == 200)
    {
        var data = json_decode(async_load[? "value"]);
        var recaptchaSiteKey = data.recaptchaSiteKey;
    }
    else
    {
        show_debug_message(async_load[? "errorMessage"]);
    }
}
```

The code above matches the response against the correct event **type**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_RefreshUserData

This function can be used to refresh the user data 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:

```
FirebaseAuthentication_RefreshUserData()
```

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_RefreshUserData"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
value	string	A json formatted string with the **recaptchaSiteKey** value (if status == 200) OPTIONAL
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_RefreshUserData();
```

In the code above we attempt to refresh the current user's data. The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "type"] == "FirebaseAuthentication_RefreshUserData")
{
    if (async_load[? "status"] == 200)
    {
        var data = json_decode(async_load[? "value"]);
        show_debug_message(data);
    }
    else
    {
        show_debug_message(async_load[? "errorMessage"]);
    }
}
```

The code above matches the response against the correct event **type**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_SendEmailVerification

This function sends an email verification to the current user (used for Email based authentication) 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:

```
FirebaseAuthentication_SendEmailVerification(email)
```

Argument	Type	Description
email	string	The email to be verified.

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_SendEmailVerification"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_SendEmailVerification();
```

In the code above we attempt to send an email verification to the current user. The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "type"] == "FirebaseAuthentication_RefreshUserData")
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message("Email verification succeeded");
    }
    else
    {
        show_debug_message(async_load[? "errorMessage"]);
    }
}
```

The code above matches the response against the correct event **type**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_SendPasswordResetEmail

This function sends a password reset email to the current user (used for Email based authentication) 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:

```
FirebaseAuthentication_SendPasswordResetEmail(email)
```

Argument	Type	Description
email	string	The email to send the password reset email to.

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_SendPasswordResetEmail"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
value	string	The return value (if status == 200) OPTIONAL
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_SendPasswordResetEmail("myemail@example.com");
```

In the code above we attempt to send a password reset email to the current user. The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "type"] == "FirebaseAuthentication_SendPasswordResetEmail")
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message("Email verification succeeded");
    }
    else
    {
        show_debug_message(async_load[? "errorMessage"]);
    }
}
```

The code above matches the response against the correct event **type**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_SendVerificationCode

This function sends a verification code and requests for a **sessionInfo** key (used for **Phone** based authentication), it then 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:

```
FirebaseAuthentication_SendVerificationCode(phoneNumber, recaptchaToken)
```

Argument	Description
phoneNumber	The phone number to send the verification code to, including country code (ie.: `"+44123456789"`)
recaptchaToken	The recaptcha token (obtained from solving the reCAPTCHA puzzle).

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_SendVerificationCode"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see **reference**)
value	string	A json formatted string with the **sessionInfo** value (if status == 200) OPTIONAL

errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_SendVerificationCode(phoneNumber, recaptchaToken);
```

In the code above we attempt to send a verification code to the current user. The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "type"] == "FirebaseAuthentication_SendVerificationCode")
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message("Verification code sent!");
    }
    else
    {
        show_debug_message(async_load[? "errorMessage"]);
    }
}
```

The code above matches the response against the correct event **type**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_SignIn_Anonymously

This function signs in a user anonymously 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:

```
FirebaseAuthentication_SignIn_Anonymously()
```

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_SignIn_Anonymously"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
value	string	A json formatted string with the user's raw data (if status == 200) OPTIONAL
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_SignIn_Anonymously();
```

In the code above we attempt to sign in the current user anonymously. The function call will then return a listener ID (`listenerId`) that can be used inside an Async Social event.

```
if (async_load[? "type"] == "FirebaseAuthentication_SignIn_Anonymously")
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message("Signed in successfully!");
    }
    else
    {
        show_debug_message(async_load[? "errorMessage"]);
    }
}
```

The code above matches the response against the correct event **type**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_SignIn_Email

This function signs in a user with an email and password (used for Email based authentication) 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:

```
FirebaseAuthentication_SignIn_Email(email, pass)
```

Argument	Tyoe	Description
email	string	The email to sign in with.
pass	string	The password associated with the email above (not the actual email's password).

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_SignIn_Email"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
value	string	A json formatted string with the user's raw data (if status == 200) OPTIONAL
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_SignIn_Email("myemail@example.com",
"mypass123!");
```

In the code above we attempt to sign in the current user using an email/password. The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "type"] == "FirebaseAuthentication_SignIn_Email")
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message("Signed in successfully!");
    }
    else
    {
        show_debug_message(async_load[? "errorMessage"]);
    }
}
```

The code above matches the response against the correct event **type**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_SignIn_OAuth

This function signs in a user with an OAuth credential (used for Apple, Facebook, Google and GooglePlay based authentications) 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:

```
FirebaseAuthentication_SignIn_OAuth(token, token_kind, provider, [redirect_uri])
```

Argument	Type	Description
token	string	The provider token.
token_kind	string	One of the provided strings `"id_token"`, `"access_token"` or `"serverAuthCode"`
provider	string	The provider identifier (ie.: `"apple.com"`, `"google.com"`, etc)
redirect_uri	string	OPTIONAL The redirect URI if using REST API (can be checked on the Firebase Console for your provider)

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_SignIn_OAuth"`
listener	real	The asynchronous listener ID.

status	real	The HTTP status response code (see reference)
value	string	A json formatted string with the user's raw data (if status == 200) OPTIONAL
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_SignIn_OAuth(token, token_kind, provider,
redirect_uri);
```

In the code above we attempt to sign in the current user using an OAuth credential. The function call will then return a listener ID (`listenerId`) that can be used inside an Async Social event.

```
if (async_load[? "type"] == "FirebaseAuthentication_SignIn_OAuth")
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message("Signed in successfully!");
    }
    else
    {
        show_debug_message(async_load[? "errorMessage"]);
    }
}
```

The code above matches the response against the correct event **type**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_SignInWithCustomToken

This function tries to sign in a user with a given custom token. Use this function after you retrieve a Firebase Auth Custom Token from your server (used with Custom authentication). The function 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:

```
FirebaseAuthentication_SignInWithCustomToken(customToken)
```

Argument	Type	Description
customToken	string	The user's Firebase custom token.

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_SignInWithCustomToken"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
value	string	A json formatted string with the user's raw data (if status == 200) OPTIONAL
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_SignIn_OAuth(token, token_kind, provider,
redirect_uri);
```

In the code above we attempt to sign in the current user using an OAuth credential. The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "id"] == listenerId)
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message("Signed in successfully!");
    }
    else
    {
        show_debug_message(async_load[? "errorMessage"]);
    }
}
```

The code above matches the response against the correct event listener **id**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_SignInWithPhoneNumber

This function signs in a user with a phone number (used for **Phone** based authentication) 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:

```
FirebaseAuthentication_SignInWithPhoneNumber(phone, code, sessionInfo)
```

Argument	Type	Description
phone	string	The phone number, including country code (ie.: `"+44123456789"`)
code	string	The code from the SMS.
session_info	string	Session info from the re-captcha puzzle.

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_SignInWithPhoneNumber"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
value	string	A json formatted string with the user's raw data (if status == 200) OPTIONAL
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_SignInWithPhoneNumber(phone, code, sessionInfo);
```

In the code above we attempt to sign in the current user using a phone number. The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "id"] == listenerId)
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message("Signed in successfully!");
    }
    else
    {
        show_debug_message(async_load[? "errorMessage"]);
    }
}
```

The code above matches the response against the correct event listener **id**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_SignOut

This function signs out the current user.

Syntax:

```
FirebaseAuthentication_SignOut()
```

Returns:

```
N/A
```

Example:

```
FirebaseAuthentication_SignOut();
```

The code above will log the current user out of the Firebase Authentication system.

FirebaseAuthentication_SignUp_Email

This functions signs up a user with an email and password (used for **Email** based authentication) 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:

```
FirebaseAuthentication_SignUp_Email(email, pass)
```

Argument	Type	Description
email	string	The email to to be used during sign-up.
pass	string	The password to be associated with the email provided (not the actual email's password)

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_SignUp_Email"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see **reference**)
value	string	A json formatted string with the user's raw data (if status == 200) OPTIONAL
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_SignUp_Email("myemail@example.com",
"mypass123!");
```

In the code above we attempt to sign up the current user using an email and password. The function call will then return a listener ID (`listenerId`) that can be used inside an Async Social event.

```
if (async_load[? "id"] == listenerId)
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message("Signed up successfully!");
    }
    else
    {
        show_debug_message(async_load[? "errorMessage"]);
    }
}
```

The code above matches the response against the correct event listener **id**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

FirebaseAuthentication_UnlinkProvider

This function unlinks a provider from the current user 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:

```
FirebaseAuthentication_UnlinkProvider(provider_identifer)
```

Argument	Description
provider_identifer	The provider identifier (e.g. `"apple.com"`, `"google.com"`, `"phone"`, etc)

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_UnlinkProvider"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_UnlinkProvider("apple.com");
```

In the code above we attempt to unlink the provider `"apple.com"` from the current user. The function call will then return a listener ID (`listenerId`) that can be used inside an Async Social event.

```
if (async_load[? "id"] == listenerId)
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message("Unliked provider successfully!");
    }
    else
    {
        show_debug_message(async_load[? "errorMessage"]);
    }
}
```

The code above matches the response against the correct event listener **id**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

SDKFirebaseAuthentication_LinkWithGameCenter

This function links a Game Center account to the current user (used for GameCenter based authentication) 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 only available on iOS.

Syntax:

```
SDKFirebaseAuthentication_LinkWithGameCenter()
```

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_LinkWithGameCenter"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
value	string	A json formatted string with the user's raw data (if status == 200) OPTIONAL
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_LinkWithGameCenter();
```

In the code above we attempt to link the current user to a game center account. The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "id"] == listenerId)
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message("Account linked successfully!");
    }
    else
    {
        show_debug_message(async_load[? "errorMessage"]);
    }
}
```

The code above matches the response against the correct event listener **id**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

SDKFirebaseAuthentication_LinkWithProvider

This function links the user account with a specific provider (the user needs to be signed in) 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 only available on Android, iOS and Web.

Syntax:

```
SDKFirebaseAuthentication_LinkWithProvider(provider)
```

Argument	Type	Description
provider	string	The provider identifier (ie.: `"apple.com"`, `"google.com"`, etc)

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"SDKFirebaseAuthentication_LinkWithProvider"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
value	string	A json formatted string with the user's raw data (if status == 200) OPTIONAL

errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = SDKFirebaseAuthentication_LinkWithProvider("apple.com");
```

In the code above we attempt to link the current user to the `"apple.com"` provider (see **Apple** for more information on this authentication method). The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "id"] == listenerId)
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message("Account linked successfully!");
    }
    else
    {
        show_debug_message(async_load[? "errorMessage"]);
    }
}
```

The code above matches the response against the correct event listener **id**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

SDKFirebaseAuthentication_ReauthenticateWithGameCenter

This function re-authenticates the user using a Game Center account (used for GameCenter based authentication) 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 only available on iOS.

Syntax:

```
SDKFirebaseAuthentication_ReauthenticateWithGameCenter()
```

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"FirebaseAuthentication_ReauthenticateWithGameCenter"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
value	string	A json formatted string with the user's raw data (if status == 200) OPTIONAL
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = FirebaseAuthentication_ReauthenticateWithGameCenter();
```

In the code above we attempt to re-authenticate the current user using a game center account. The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "id"] == listenerId)
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message("Account linked successfully!");
    }
    else
    {
        show_debug_message(async_load[? "errorMessage"]);
    }
}
```

The code above matches the response against the correct event listener **id**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

SDKFirebaseAuthentication_ReauthenticateWithProvider

This function re-authenticates the user account with a specific provider (the user needs to be signed in) 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 only available on Android, iOS and Web.

Syntax:

```
SDKFirebaseAuthentication_ReauthenticateWithProvider(provider)
```

Argument	Type	Description
provider	string	The provider identifier (ie.: `"apple.com"`, `"google.com"`, etc)

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"SDKFirebaseAuthentication_ReauthenticateWithProvider"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
value	string	A json formatted string with the user's raw data (if status == 200) OPTIONAL

errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = SDKFirebaseAuthentication_ReauthenticateWithProvider("apple.com");
```

In the code above we attempt to re-authenticate the current user to the `"apple.com"` provider (see **Apple** for more information on this authentication method). The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "id"] == listenerId)
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message("Account re-authenticated successfully!");
    }
    else
    {
        show_debug_message(async_load[? "errorMessage"]);
    }
}
```

The code above matches the response against the correct event listener **id**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

SDKFirebaseAuthentication_SignIn_GameCenter

This function signs in the current user with a Game Center account (used for GameCenter based authentication) 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 only available on iOS.

Syntax:

```
SDKFirebaseAuthentication_SignIn_GameCenter()
```

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"SDKFirebaseAuthentication_SignIn_GameCenter"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
value	string	A json formatted string with the user's raw data (if status == 200) OPTIONAL
errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = SDKFirebaseAuthentication_SignIn_GameCenter();
```

In the code above we attempt to sign in the current using a game center account. The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "id"] == listenerId)
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message("Account linked successfully!");
    }
    else
    {
        show_debug_message(async_load[? "errorMessage"]);
    }
}
```

The code above matches the response against the correct event listener **id**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.

SDKFirebaseAuthentication_SignInWithProvider

This function signs in the user using the browser account native to a specific provider 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 only available on Android, iOS and Web.

Syntax:

```
SDKFirebaseAuthentication_SignInWithProvider(provider)
```

Argument	Type	Description
provider	string	The provider identifier (ie.: `"apple.com"`, `"google.com"`, etc)

Returns:

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

Triggers:

```
Asynchronous Social Event
```

Key	Type	Description
type	string	The string `"SDKFirebaseAuthentication_SignInWithProvider"`
listener	real	The asynchronous listener ID.
status	real	The HTTP status response code (see reference)
value	string	A json formatted string with the user's raw data (if status == 200) OPTIONAL

errorMessage	string	The readable error message OPTIONAL

Example:

```
listenerId = SDKFirebaseAuthentication_SignInWithProvider("apple.com");
```

In the code above we attempt to sign in the current user using the `"apple.com"` provider (see **Apple** for more information on this authentication method). The function call will then return a listener ID (`listenerId`) that can be used inside an **Async Social** event.

```
if (async_load[? "id"] == listenerId)
{
    if (async_load[? "status"] == 200)
    {
        show_debug_message("User signed in successfully!");
    }
    else
    {
        show_debug_message(async_load[? "errorMessage"]);
    }
}
```

The code above matches the response against the correct event listener **id**, checks for a **status** value of `200` meaning that it was successful and logs the result of the task.