

# How to integrate SUSI.AI with Google Sign In API



Google Sign-In manages the OAuth 2.0 flow and token lifecycle, simplifying our integration with Google APIs. A user always has the option to [revoke access](#) to an application at any time. Users can see a list of all the apps which they have given permission to access their account details and are using the login API. In this blog post I will discuss how to integrate this API with susi server API to enhance our AAA system.

## Introduction

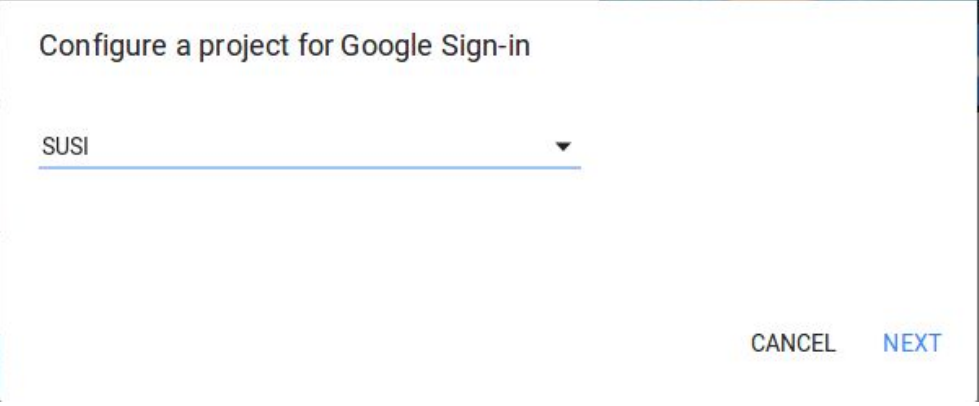
Google Sign-In uses the Open-Auth token system which acts as a security checker which reduces the work of the developers and users. For the users, they don't need separate IDs and password for different web applications as remembering a lot of different passwords can be a hectic job for the users. Also for the developers, it reduces the work on the developers side to worry less about the authentication and security and it also reduces the work of our servers.

In simple way we can say that when a user Sign up using the Google API, a specific token is created for a specific user which can then be stored in our database in json and other formats. Every time a user tries to log In, Google verifies its credentials using its own technology and after verifying it sends the token corresponding to the user and then we can verify that token with our database and then give the server response accordingly. This way it reduces server load on our part and let google worry about the authentication.

# How to set up Google Sign-In API for SUSI

## Step 1: Configuration

Go to Google developers console and select the project where you deployed your server on google cloud and select SUSI.

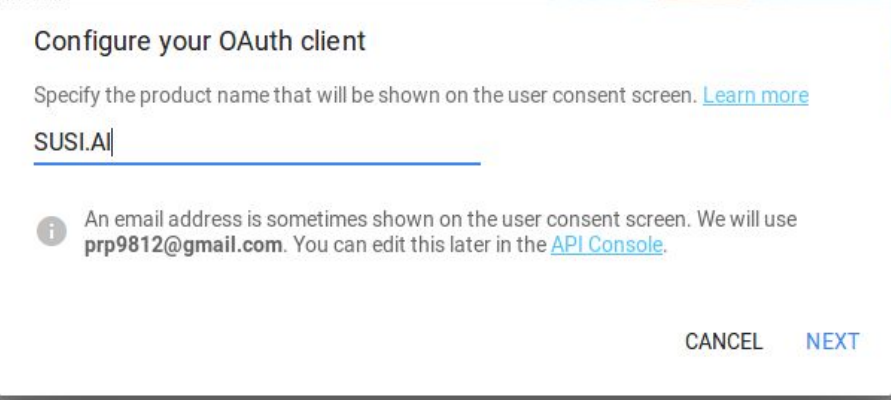


Configure a project for Google Sign-in

SUSI

CANCEL NEXT


After that we need to configure our client name which will be shown on the user consent screen.



Configure your OAuth client

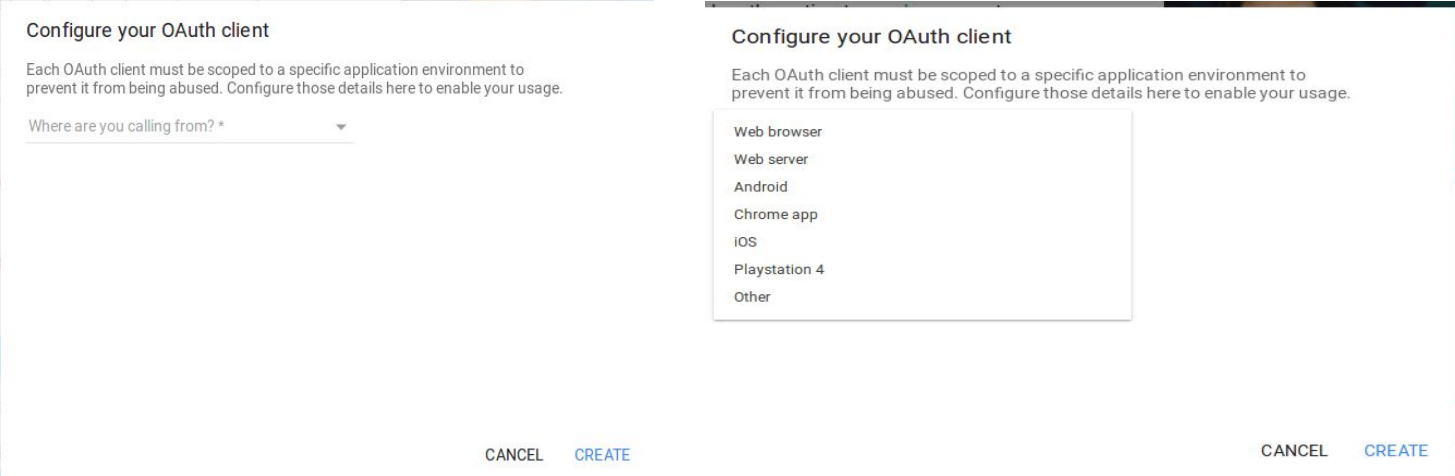
Specify the product name that will be shown on the user consent screen. [Learn more](#)

SUSI.AI

 An email address is sometimes shown on the user consent screen. We will use **prp9812@gmail.com**. You can edit this later in the [API Console](#).

CANCEL NEXT

After that we need to configure What type of environment we are calling oauth from for example a web server, web client, android or ios platform.



Configure your OAuth client

Each OAuth client must be scoped to a specific application environment to prevent it from being abused. Configure those details here to enable your usage.

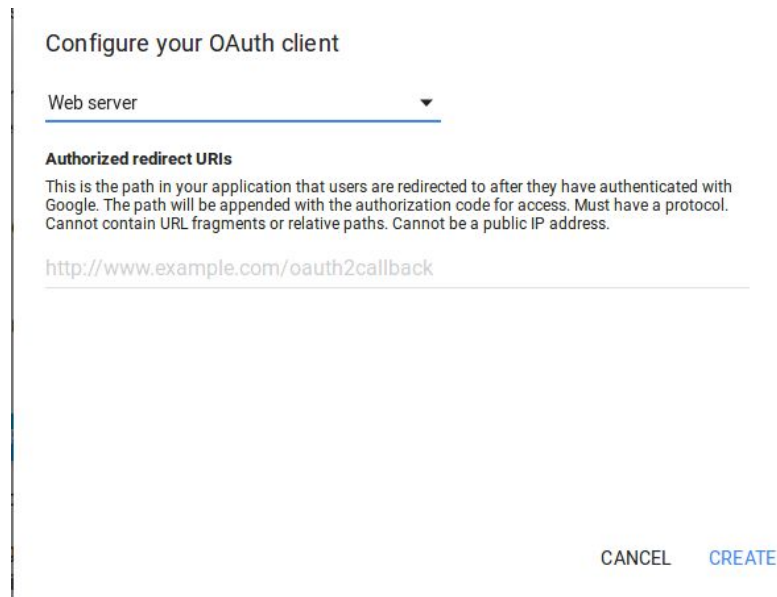
Where are you calling from? \*

Web browser  
Web server  
Android  
Chrome app  
iOS  
Playstation 4  
Other

CANCEL CREATE

CANCEL CREATE

Since we are integrating with the server API we will select web server



Configure your OAuth client

Web server ▼

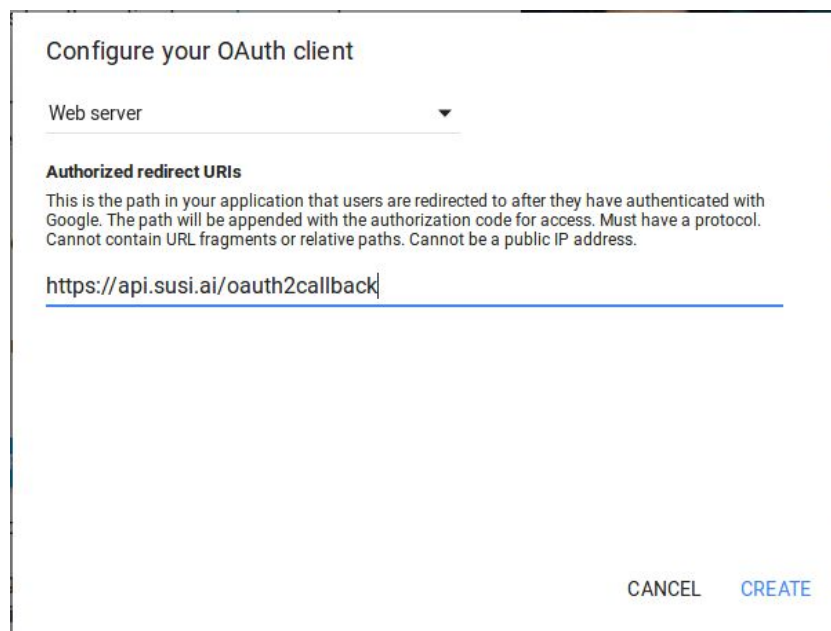
**Authorized redirect URIs**

This is the path in your application that users are redirected to after they have authenticated with Google. The path will be appended with the authorization code for access. Must have a protocol. Cannot contain URL fragments or relative paths. Cannot be a public IP address.

http://www.example.com/oauth2callback

CANCEL CREATE

Then we need to give the path on which the google API will redirect us after successful validation.



Configure your OAuth client

Web server ▼

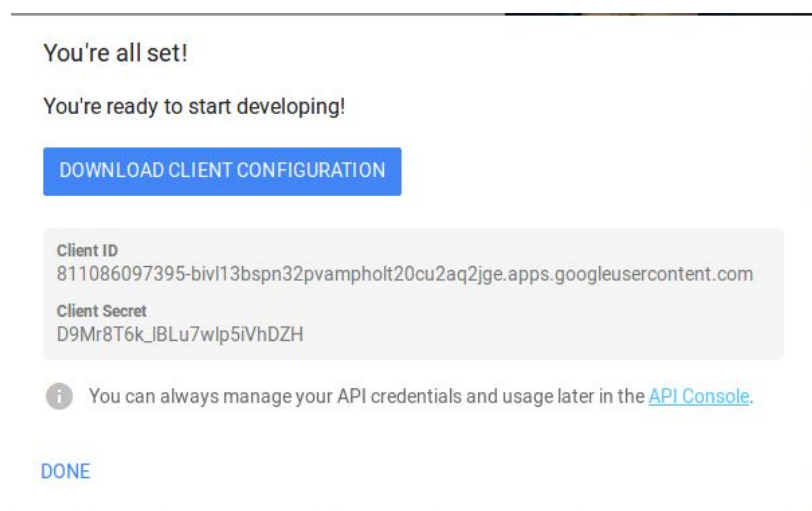
**Authorized redirect URIs**

This is the path in your application that users are redirected to after they have authenticated with Google. The path will be appended with the authorization code for access. Must have a protocol. Cannot contain URL fragments or relative paths. Cannot be a public IP address.

https://api.susi.ai/oauth2callback

CANCEL CREATE

This is the last step, after that we will get our client ID and our server IDs which we will use in the corresponding steps.




You're all set!

You're ready to start developing!

DOWNLOAD CLIENT CONFIGURATION

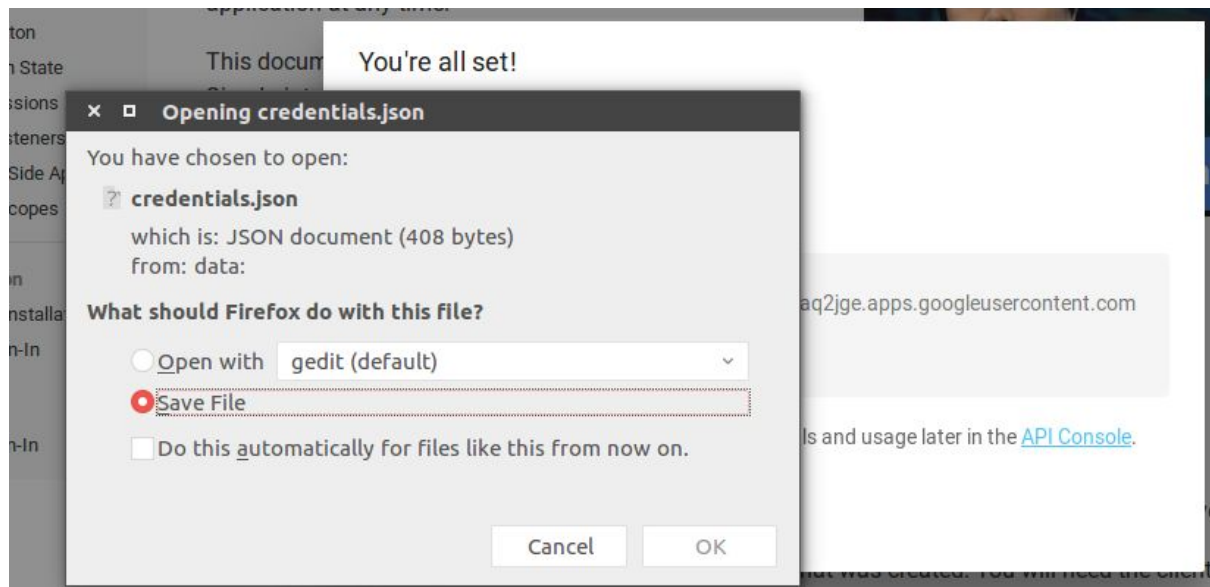
**Client ID**  
811086097395-biv13bspn32pvampholt20cu2aq2jge.apps.googleusercontent.com

**Client Secret**  
D9Mr8T6k\_IBLu7wlp5iVhDZH

 You can always manage your API credentials and usage later in the [API Console](#).

DONE

We can then download our client configuration file which is of json type format of name credentials.json. We need to download and save this file in our server directory



After configuration is complete, take note of the client ID that was created. You will need the client ID to complete the next steps. (A client secret is also created, but you need it only for server-side operations.).

## Step 2: Initialisation

You must include the Google Platform Library on your web pages that integrate Google Sign-In.

```
<script src="https://apis.google.com/js/platform.js" async defer></script>
```

## Step 3: Specify your app's client ID

Specify the client ID you created for your app in the Google Developers Console with the google-signin-client\_id meta element.

replace the <YOUR\_CLIENT\_ID> with the id we saved at the end of the configuration step.

```
<meta name="google-signin-client_id"
content="YOUR_CLIENT_ID.apps.googleusercontent.com">
```

## Step 4: Add a Google Sign-In button

To add the Google Sign-In button to SUSI we will use an automatically rendered sign-in button. Google provide us with a simple and easy way to automatically configures itself to have the appropriate text, logo, and colors for the sign-in state of the user and the scopes you request.

To create a Google Sign-In button that uses the default settings, add a div element with the class g-signin2 to your sign-in page:

```
<div class="g-signin2" data-onsuccess="onSignIn"></div>
```

We also need to Follow Google Branding policies for the sign in button [here](#) if we want to manually configure the styling of our sign In button.

## Step 5: Fetch profile information of users and store them in our database.

After you have signed in a user with Google using the default scopes, you can access the user's Google ID, name, profile URL, and email address.

To retrieve profile information for a user, use the [getBasicProfile\(\)](#) method.

Code block:

```
function onSignIn(googleUser) {  
  var profile = googleUser.getBasicProfile();  
  console.log('ID: ' + profile.getId()); // Do not send to your backend! Use an ID token
```

```
instead.  
console.log('Name: ' + profile.getName());  
console.log('Image URL: ' + profile.getImageUrl());  
console.log('Email: ' + profile.getEmail()); // This is null if the 'email' scope is not present.  
}
```

This will give us the profile information of the user which can be securely stored in our database and use it to enhance user experience without violating their privacy policies.

**Step 6:** To enable users to sign out of SUSI without signing out of Google, we can do it by adding a sign-out button or link to chat.susi.ai. To create a sign-out link, attach a function that calls the [GoogleAuth.signOut\(\)](#) method to the link's onclick event.

```
<a href="#" onclick="signOut();">Sign out</a>  
<script>  
  function signOut() {  
    var auth2 = gapi.auth2.getAuthInstance();  
    auth2.signOut().then(function () {  
      console.log('User signed out.');    });  
  }  
</script>
```

## References:

<https://developers.google.com/identity/sign-in/web/sign-in>

<https://www.youtube.com/watch?v=Oy5F9h5JqEU>

[https://www.youtube.com/watch?v=j\\_31hJtWjlw](https://www.youtube.com/watch?v=j_31hJtWjlw)

<https://www.youtube.com/watch?v=zZt8SFivjps>