

# Social Authentication

---

## Introduction:: Social Authentication:: Why?

In this lecture, we will be studying third-party or social authentication. We all have seen the buttons like Sign-in using Facebook, log-in using Google or Github, on the various social platforms. This is known as Social Authentication. Whenever we click on that button it does two things - It saves a lot of time and user effort.

- These days since people already have accounts on places like Gmail, Facebook, Github, etc, it is possible to use an authentication service provided by them called **OAuth** { Which means their authentication can be used on my platform as well }.
- We will be using Google login in this module but soon you will gain the necessary understanding to look into other authentication modules.
- We will implement **OAuth** by using cookies.

---

## Creating Credentials on Google

- To be able to use **OAuth - 2** { Google services }, we need to tell **Google** what application is going to use it.
- We need to establish the identity and tell **Google** for security purposes that we are taking out the information of that user. The user also gives out permission.

Visit the link present below-

<http://console.developer.google.com/>

- Go to { **My Projects** } where you have your list of projects and click on a {**new project** }
- Give the name of your project ( codeial -sample ) and click on {**create it**}.
- Go on to the title project and click on create credentials.
- Credentials are for **OAuth -2**.

In **OAuth -1**, we generate the access token. Once we got it, we kept on using it until and unless that token expired.

In **OAuth - 2**, we get only the desired information, your token has a limited set of permissions and your token can anytime be revoked by Google but it is much more secure as compared to **OAuth - 1**.

- Click on the option to create credentials, there we need to click on **OAuth client ID**.

**Warning** - To create an OAuth client ID you must first set up the product name on the consent screen.

- Click on the **configure consent screen**.
- Give your application name there and leave the rest of the fields empty for now and save it.
- The application type is a web application, the name will be the name of your project.
- In **Authorized Javascripts origin** fill { <http://localhost:8000> }.
- In Authorized redirect URLs fill { <http://localhost:8000/users/auth/google/callback> } [ For call back URL we need to mention it here and in code also ].
- Click on the create button to create credentials.
- You will be getting your **client ID** and **client secret**.

---

## Social Authentication:: How does it work

- This involves the third-party authentication provider which establishes the identity of the user.
- There will be a user, third-party authentication provider, and server with the database.
- The user comes and clicks on the sign-in or sign up with google. A pop-up opens in another window that asks us to sign in with google if we are not signed in or we have signed using multiple accounts. For the latter case, we choose from one of those accounts.
- Google then asks for permission from the user, that '**project\_name** is asking for your permission to access your account'. This happens only if the user is accessing the website for the first time.
- When the pop-up opens and we submit some credentials or select an account, the request is redirected to the servers of **Google**. Google checks its database whether the user exists or not. If the user exists, Google checks if and the credentials are correct or not.
- 
- If the authentication is unsuccessful, you will be redirected back to the login page with an error.
- Once the credentials are matched successfully, the user will be redirected to another URL on the same browser for the server which is called using the callback URL with some data of the signed-in user. This is created by the developer of the website.
- 
- The server checks with the database whether the Email ID that has been provided by **Google** is there in the database or not.
- If it is there in the database, the user signs in successfully.

- If, however, if the Email ID is not in the database, then OAuth will create the user credentials in the database, post which the user will be signed in to the website.

---

## Setting up Passport - Google - OAuth

We will be using a library called crypto to generate random, unique passwords.

- Create a file inside config { **passport-google -oauth-2-strategy.js** }.
- We need to install some library for that { npm install passport-google-oauth }.
- Require passport library in { **passport-google -oauth-2-strategy.js** } file.
- Require passport-google-oauth in { **passport-google -oauth-2-strategy.js** } file.
- We need to install the crypto library { npm install crypto } and require it inside { **passport-google -oauth-2-strategy.js** } file.
- Require **users** file from the **models** folder.
- We need to tell passport to use google strategy using passport.use in which we pass in the options - { client ID, client secret, callback URL }.
- We need to put in the callback function which will be taking { access token, refresh token, profile information, and the final function which will be taking callback from this function }.
- If there is some error then console that error and return.
- If we find the user then we will call done without an error and pass the user to it { if found, set this user as req. user } if the user is not found then create the user { if not found, create the user and set it as req. user }.
- While creating the user uses a profile, the profile has different fields { display name, email, password using crypto }.

`crypto.randomBytes(20).toString('hex')`

{ passport-google -oauth-2-strategy.js }

```
const googleStrategy = require('passport-google-oauth').OAuth2Strategy;
const crypto = require('crypto');
const User = require('../models/user');

// tell passport to use a new strategy for google login
passport.use(new googleStrategy({
  clientID: '<YOUR_GOOGLE_CLIENT_ID>', // e.g. asdfghjkkadhajsghjk.apps.googleusercontent.com
  clientSecret: '<YOUR_GOOGLE_CLIENT_SECRET>', // e.g. _ASDFA%KFEJWIASDFASD#FAD-
  callbackURL: "http://localhost:8000/users/auth/google/callback",
},

function(accessToken, refreshToken, profile, done){
  // find a user
  User.findOne({email: profile.emails[0].value}).exec(function(err, user){
    if (err){console.log('error in google strategy-passport', err); return;}
    console.log(accessToken, refreshToken);
    console.log(profile);

    if (user){
      // if found, set this user as req.user
      return done(null, user);
    }else{
      // if not found, create the user and set it as req.user
      User.create({
        name: profile.displayName,
        email: profile.emails[0].value,
        password: crypto.randomBytes(20).toString('hex')
      }, function(err, user){
        if (err){console.log('error in creating user google strategy-passport', err); return;}

        return done(null, user);
      });
    }
  });
});

module.exports = passport;
```

## Using Google Auth:: With a Link

Place the sign-in button, make the routes and get it working.

- Inside the **routes** folder and in **{users.js}** file we will be creating two routes - { when we will click on the button for google sign-in for fetching the data when google

fetches the data from the database and sends it back to the routes which is the callback URL }.

- The scope is the information that we are looking to fetch - { profile, email}.
- Until and unless we ask for a refresh token it is not coming up.

**{ user.js }**

```
const express = require('express');
const router = express.Router();
const passport = require('passport');

const usersController = require('../controllers/users_controller');

router.get('/profile/:id', passport.checkAuthentication, usersController.profile);
router.post('/update/:id', passport.checkAuthentication, usersController.update);

router.get('/sign-up', usersController.signUp);
router.get('/sign-in', usersController.signIn);

router.post('/create', usersController.create);

// use passport as a middleware to authenticate
router.post('/create-session', passport.authenticate(
  'local',
  {failureRedirect: '/users/sign-in'},
), usersController.createSession);

router.get('/sign-out', usersController.destroySession);

router.get('/auth/google', passport.authenticate('google', {scope: ['profile', 'email']}));
router.get('/auth/google/callback', passport.authenticate('google', {failureRedirect: '/users/sign-in'}), usersController.createSession);

module.exports = router;
```

## { Pages for sign-in and sign-up }

<codeial />


Q Search

Home Signup Login

### Log In

[Forgot Password?](#)

Log In

 Sign in with google


<codeial />

Q Search

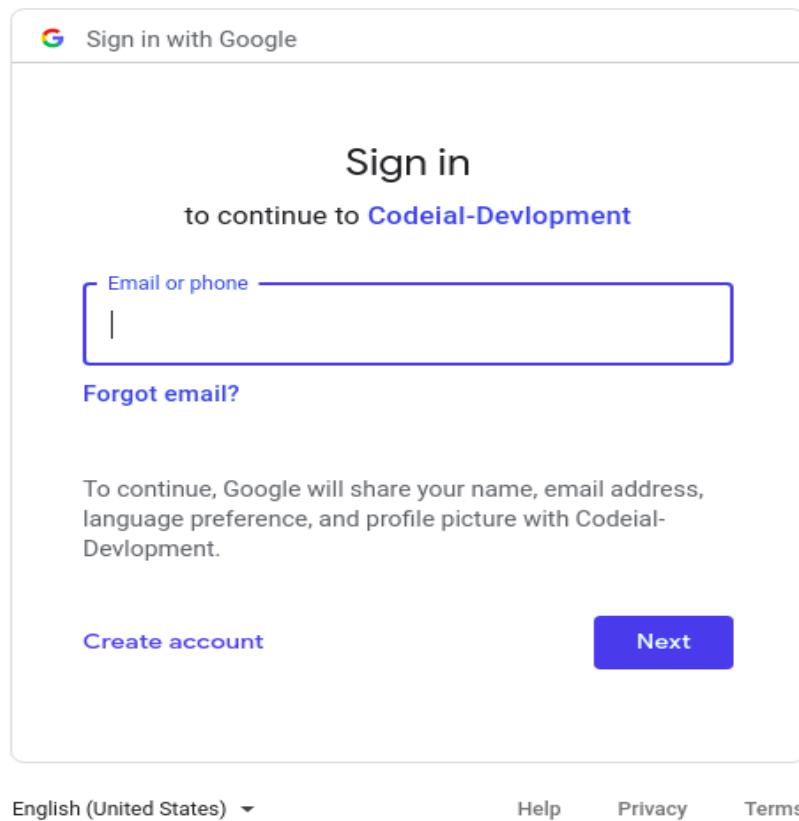
Home Signup Login

### Sign Up

Sign Up

 Sign in with google

## { Google Auth }



Sign in with Google

Sign in  
to continue to **Codeial-Development**

Email or phone

[Forgot email?](#)

To continue, Google will share your name, email address, language preference, and profile picture with Codeial-Development.

[Create account](#) [Next](#)

English (United States) ▼ [Help](#) [Privacy](#) [Terms](#)

---

## Summarizing Google Sign in

- We have used three different strategies for authentication.
- Database, different models, relationships between models.
- Aspects related to MVC [Models, Views, Controllers] architecture.