

# Passport | Strategies and Social Login

SELF GUIDED

## Learning Goals

*After this lesson, you will be able to:*

- Understand what OAuth is and how it works
- Understand why social login has increased its importance nowadays
- Authorize the users in our application through Slack login
- Authorize the users in our application through Google Account login

## Introduction

In the previous lesson, we have seen that **Passport** is a flexible and modular authentication middleware for Node.js, and we have seen how to configure a project with basic authorization.

Now we are going to focus on the modularity and the different strategies we can follow to allow users access to our site. We are going to see what OAuth is and how to configure Passport to work with it. Finally, we are going to describe social login strategies and we configure one of the different options we have.

We will be working on the same project we used in the previous lesson. So open it in your code editor, and let's start working!

## OAuth

Before we start working, it's important to understand what OAuth is. In

modern web applications, authentication can take a variety of forms. We have seen the most traditional way to log in, by providing username and password.

With the appearance of social networking, single sign-on using an OAuth provider (such as Facebook, Slack, or Twitter) has become a popular authentication method.

**OAuth** is an open standard for authorization commonly used as a way for social network users to authorize external websites to access their information, without giving them the passwords.

OAuth provides the users a secure way to access external websites without creating a user account on it, so they can log into our apps with their social network accounts.

Passport offers 300+ authentication mechanisms, known as strategies, that allow integrating our web application with different external services such as Facebook, Twitter, GitHub, Gmail, Instagram, etc.

Social login consists of using your social network profile to authenticate yourself in external websites.

## Social Login

We will work on two different social login configurations: Slack and Google accounts. The first is the one of the most popular chat apps in the world, while the second is one of the most used accounts for email.

## User model

We are going to need two new fields in our `User` model for users who log in with social networks.

```
1 // models/user.js
2
3 const mongoose = require("mongoose");
4 const Schema = mongoose.Schema;
5
6 const userSchema = new Schema(
7   {
8     username: String,
9     password: String,
10    slackID: String,
11    googleID: String
12  },
13  {
14    timestamps: { createdAt: "created_at", updatedAt: "updated_at" }
15  }
16 );
17
18 const User = mongoose.model("User", userSchema);
19
20 module.exports = User;
```

Copy

## Slack

We are going to create the authentication with [Slack](#) as a provider. Support for Slack is implemented by [passport-slack](#) module.

### Slack API

The first thing we have to do is to create an app at [Slack Developers](#). Once we create this account, Slack will provide us the keys, that we will configure in our middleware.

To create a new Slack app, we have to visit the [Slack API](#) page and click on the [Create an app](#) button.

**Start here**

Building Slack apps  
Recent updates  
Best practices  
App blueprints

**App features**

Internal integrations  
Incoming webhooks  
Slash commands  
Bot users  
Dialogs  
Shared Channels  
Enterprise Grid  
Legacy custom integrations  
Developer preview

**Messages**

Introduction

## Your Apps

### Build something amazing.

Use our APIs to build an app that makes people's working lives better. You can create an app that's just for your workspace or create a public Slack App to list in the App Directory, where anyone on Slack can discover it.


[Create an App](#)

Don't see an app you're looking for? [Sign in to another workspace.](#)

We have to indicate the Display Name (the name of our app), and the workspace, in our case we can choose the Ironhack's one.

## Create a Slack App





**Interested in the next generation of apps?**  
We're improving app development and distribution. Join the API Preview period for workspace tokens and the Permissions API.

### App Name

Ironhack Login

Don't worry; you'll be able to change this later.

### Development Slack Workspace



Ironhack



Your app belongs to this workspace—leaving this workspace will remove your ability to manage this app. Unfortunately, this can't be changed later.

By creating a Web API Application, you agree to the [Slack API Terms of Service](#).

Cancel

Create App

Once we have created the app, we will find the keys we need in the “App Credentials” section. In this section, you can see all the information you need to configure the middleware.

## App Credentials

These credentials allow your app to access the Slack API. They are secret. Please don't share your app credentials with anyone, include them in public code repositories, or store them in insecure ways.

### Client ID

2432150752.359151972724

### Client Secret

••••••••••

Show

Regenerate

You'll need to send this secret along with your client ID when making your [oauth.access](#) request.

### Verification Token

zULw75gt9SKJqpadRkmv9pdl

Regenerate

For interactive messages and events, use this token to verify that requests are actually coming from Slack. Slash commands and interactive messages will both use this verification token.

Note how you have to press the “Show” button on the App Secret to reveal the secret key in the panel.

Then you have to go in the “**OAuth & Permissions**” tab to add the following Redirect URL: `http://localhost:3000/auth/slack/callback`

Lesson Passport Sl...

Settings

Basic Information

Collaborators

Install App

Manage Distribution

Features

Incoming Webhooks

Interactive Components

Slash Commands

**OAuth & Permissions**

Event Subscriptions

Bot Users

User ID Translation

Slack ♥

Help

Contact

Policies

Our Blog

## OAuth & Permissions

### OAuth Tokens & Redirect URLs

#### Tokens for Your Workspace

These tokens were automatically generated when you installed the app to your team. You can use these to authenticate your app. [Learn more.](#)

**OAuth Access Token**

xoxp-2432150752-203263376467-424007622753-bb718801fb7ea699a6a791b3772

Copy

Reinstall App

#### Redirect URLs

You will need to configure redirect URLs in order to automatically generate the Add to Slack button or to distribute your app. If you pass a URL in an OAuth request, it must (partially) match one of the URLs you enter here. [Learn more.](#)

Redirect URLs

http://localhost:3000/auth/slack/callback

✎ 🗑

Add New Redirect URL

Save URLs

## Configuration

First, we have to install the `passport-slack` package to be able to use it in the app:

```
1 $ npm install passport-slack
```

Copy

In our `app.js`, let's add the following code:

Copy

```
1 // app.js
2
3 const passport = require("passport");
4 const User = require("../models/user");
5
6 const SlackStrategy = require("passport-slack").Strategy;
7
8 passport.use(
9   new SlackStrategy(
10     {
11       clientID: "your Slack client id here",
12       clientSecret: "your Slack client secret here",
13       callbackURL: "/auth/slack/callback"
14     },
15     (accessToken, refreshToken, profile, done) => {
16       // to see the structure of the data in received response:
17       console.log("Slack account details:", profile);
18
19       User.findOne({ slackID: profile.id })
20         .then(user => {
21           if (user) {
22             done(null, user);
23             return;
24           }
25
26           User.create({ slackID: profile.id })
27             .then(newUser => {
28               done(null, newUser);
29             })
30             .catch(err => done(err)); // closes User.create()
31         })
32         .catch(err => done(err)); // closes User.findOne()
33     }
34   )
35 );
```

Remember, we have to insert our Slack app credentials: the client ID and the client secret.



This is all the configuration we have to change to be able to login with Slack accounts!

## Routes File

Now, we have to change our routes to use the Slack strategy to log in our application. To do that, we will create two different routes:

```
1 // routes/auth-routes.js
2
3 const passport = require("passport");
4
5 router.get("/auth/slack", passport.authenticate("slack"));
6 router.get(
7   "/auth/slack/callback",
8   passport.authenticate("slack", {
9     successRedirect: "/private-page",
10    failureRedirect: "/" // here you would navigate to the class.
11  })
12 );
```

Copy

## View

In the last part of the configuration, we have to create the login route. We don't need a form to log in the application. Instead, we will redirect the user to Slack, and they will have to give us permissions to access their data.

We will add the link in the home page, `views/index.hbs` as it follows:

```
1 <a href="/auth/slack">Login With Slack</a>
```

Copy

This is the route we have defined in the routes, that will call the `passport.authenticate("slack")` method to log the user in.

These are the different pieces of Slack and Local Strategies. The logout, protected routes, and views are the same in both. The main difference is the user information we are provided with.

## Google Account

Next we are going to allow users to authenticate themselves through their Google Account. You may notice how similar this process is to the Slack login.

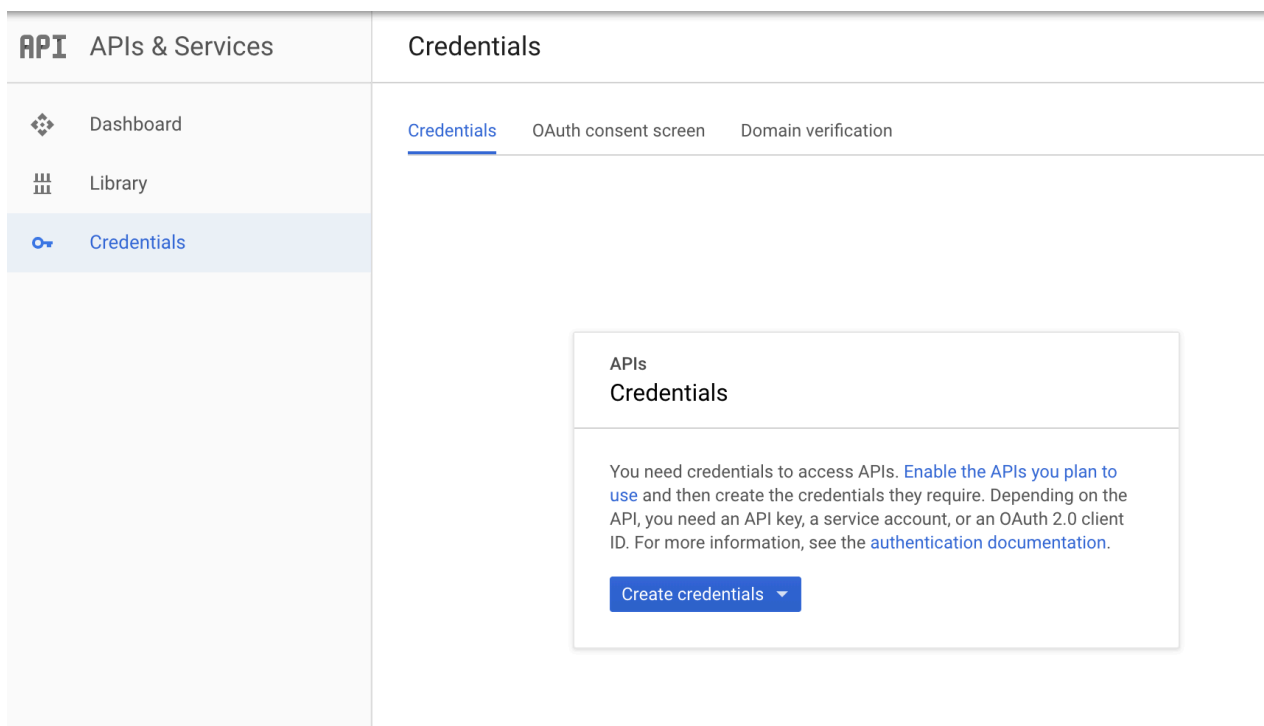
Google accounts can be configured by using different OAuth strategies. We are going to use [passport-google-oauth20 npm package](#). This npm package is based on the Passport strategy for authenticating with Google using the OAuth 2.0 API.

## New App in the Google Developers Console

Before we move forward into using previously mentioned npm package, we have to register a new application in the [Google Developers Console](#).


Steps to follow:

1. Once you go to the Google Developers Console website, click on the **Credentials** tab in the left vertical navigation bar:




2. As shown in the image above, a pop up will be shown. Click on the **Create credentials** button and pick the **OAuth client ID** option.

3. You will be redirected to the *Create OAuth client ID* page and given to choose what kind of application you are building. Pick **Web application**. An additional form will open. Follow the given instructions in the image below:

 Create OAuth client ID

For applications that use the OAuth 2.0 protocol to call Google APIs, you can use an OAuth 2.0 client ID to generate an access token. The token contains a unique identifier. See [Setting up OAuth 2.0](#) for more information.

**Application type**


☒ Web application  choose Web app


☐ Android [Learn more](#)

☐ Chrome App [Learn more](#)

☐ iOS [Learn more](#)

☐ Other

**Name** 

Passport-Google-Auth  give the app a meaningful name



**Restrictions**

Enter JavaScript origins, redirect URIs, or both [Learn More](#)

Origins and redirect domains must be added to the list of Authorized Domains in the [OAuth consent settings](#).

**Authorized JavaScript origins**



For use with requests from a browser. This is the origin URI of the client application. It can't contain a wildcard ([https://\\*.example.com](#)) or a path ([https://example.com/subdir](#)). If you're using a nonstandard port, you must include it in the origin URI.

http://localhost:3000  set default origin 


Type in the domain and press Enter to add it

**Authorized redirect URIs**

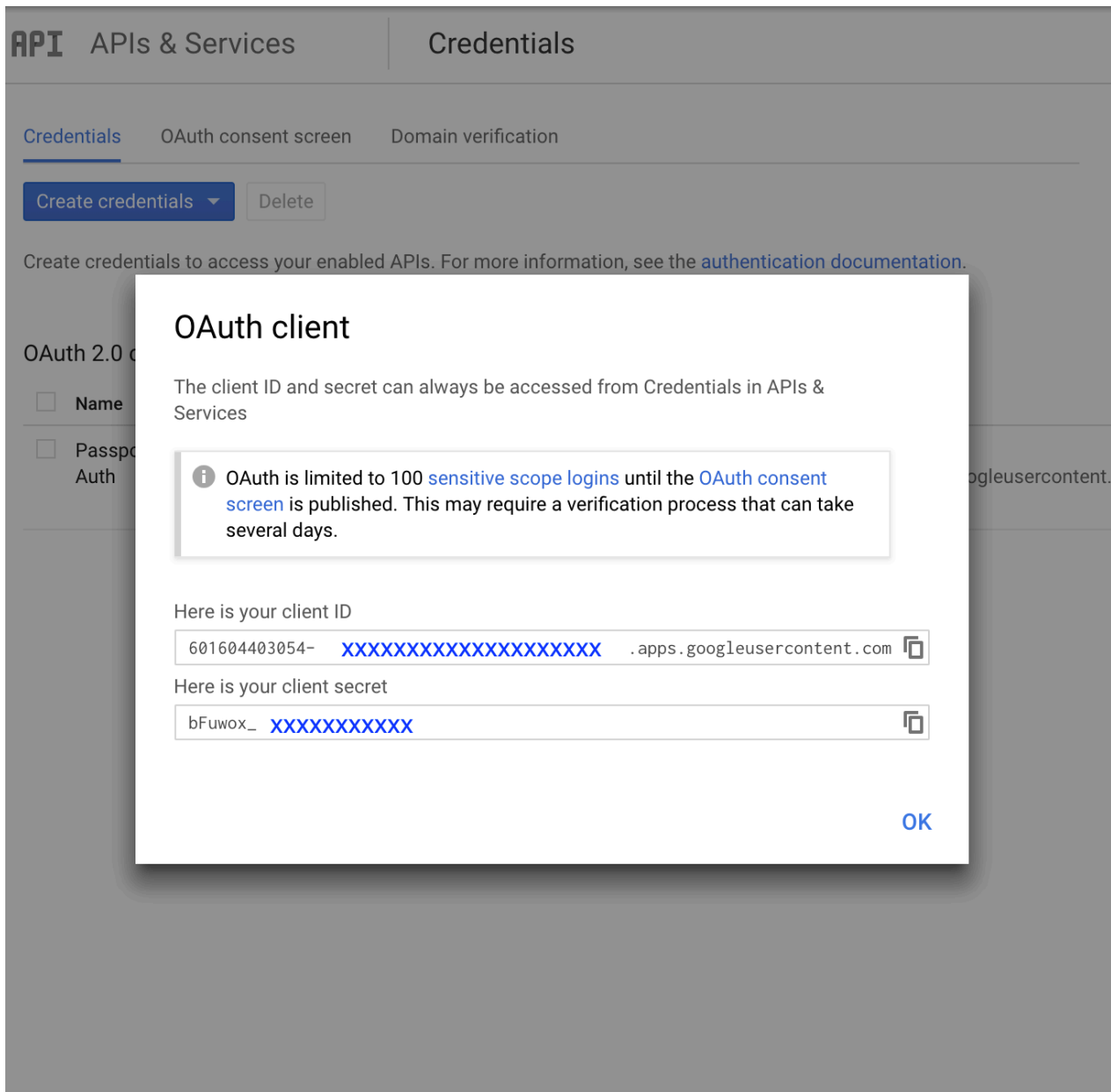
For use with requests from a web server. This is the path in your application that users are redirected to after they have authenticated with Google. The path will be appended with the authorization code for access. Must have a protocol. Cannot contain URL fragments or relative paths. Cannot be a public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

http://localhost:3000/auth/google/callback  set default redirect 

Type in the domain and press Enter to add it

 create the application

4. And voila! A pop up containing your client ID and client secret will be shown. Save them temporary somewhere since very soon you will need to add them in your code.



After clicking **Ok**, you will get redirected to the *Credentials* page where you will see the name of your app when it was created and your Client ID. In case you lose your credentials, you can always come back and recreate them by clicking on *Create credentials* button in the upper left corner of the page.

Now, let's set up our code.

## Configuration

Again, we are going to start over the base project we created, with the basic authorization process. First of all, we have to install the package we are going to use:

```
1 $ npm install passport-google-oauth20
```

Copy

In our `app.js`, let's add the following code:

```
1 // app.js
2
3 const GoogleStrategy = require("passport-google-oauth20").Strategy;
4
5 passport.use(
6   new GoogleStrategy(
7     {
8       clientId: "your Google client id here",
9       clientSecret: "your Google client secret here",
10      callbackURL: "/auth/google/callback"
11    },
12    (accessToken, refreshToken, profile, done) => {
13      // to see the structure of the data in received response:
14      console.log("Google account details:", profile);
15
16      User.findOne({ googleID: profile.id })
17        .then(user => {
18          if (user) {
19            done(null, user);
20            return;
21          }
22
23          User.create({ googleID: profile.id })
24            .then(newUser => {
25              done(null, newUser);
26            })
27            .catch(err => done(err)); // closes User.create()
28        })
29        .catch(err => done(err)); // closes User.findOne()
30    }
31  )
32 );
```

Copy

Don't forget we have to insert the Client ID and the Client Secret. With this, we finished the configuration. The next thing we have to do is to change our routes to use the correct methods during the login process.

## Routes File

In the routes file, we have to add the routes which will handle login using the Google credentials:

```
1  // routes/auth-routes.js
2
3  router.get(
4    "/auth/google",
5    passport.authenticate("google", {
6      scope: [
7        "https://www.googleapis.com/auth/userinfo.profile",
8        "https://www.googleapis.com/auth/userinfo.email"
9      ]
10   })
11 );
12 router.get(
13   "/auth/google/callback",
14   passport.authenticate("google", {
15     successRedirect: "/private-page",
16     failureRedirect: "/" // here you would redirect to the login
17   })
18 );
```

Copy

This will open a new window in the browser where the user will have to authorize the app to get Google Account data. Now we have to configure the view by adding the login link.

## View

We will update the login link in the main page, located in `/views/index.hbs`, just if there is no user logged on the application:

```
1  {{#unless user}}
2    Home page
3    <div>
4      <a href="/auth/google">Login With Google</a>
5    </div>
6  {{/unless}}
7
```

Copy

This will call the URL we just defined in our routes, and it will call Google authorization page to complete the authentication process.

This is all that we need to do to have the social login with Google account in our application. As you may notice, both social logins are very similar when it comes to the process of setting them up in our app. The other provider configurations work the same way, so you should be able to configure whichever provider you need in your app.

## Summary

We have seen that today, with the rise of social networks, OAuth has become very common way to allow social logins in our application. The key benefit is having the account created and later on authenticated without us creating a password.

We have also seen how we can configure two different providers, Slack and Google accounts, to allow our users to use their external accounts to access our application.

## Extra Resources

- [NPM package docs - passport-google-oauth20](#)
- [Slack Account Social Login with Passport](#)
- [Facebook Social Login with Passport](#)
- [Twitter Social Login with Passport](#)

Mark as completed

PREVIOUS

← Passport  
| Sign  
Up,  
Login,  
Logout

NEXT

Authentication →