

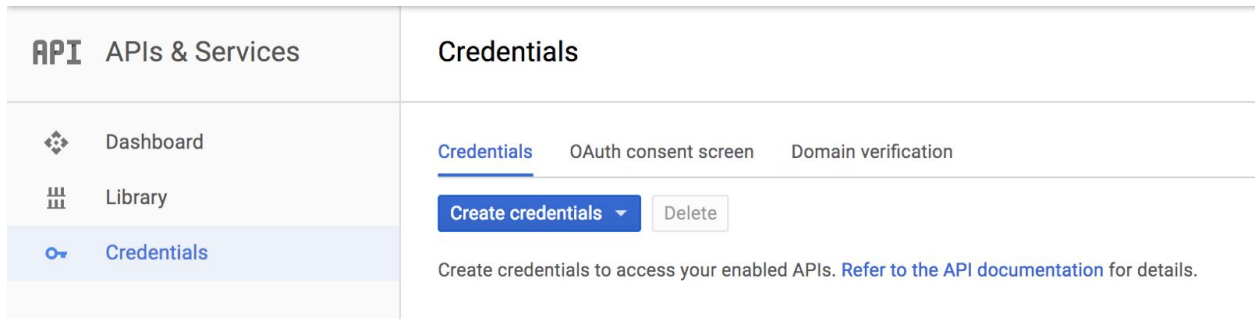
# User Authentication using Google Sign In

## By Rohit Dwivedula

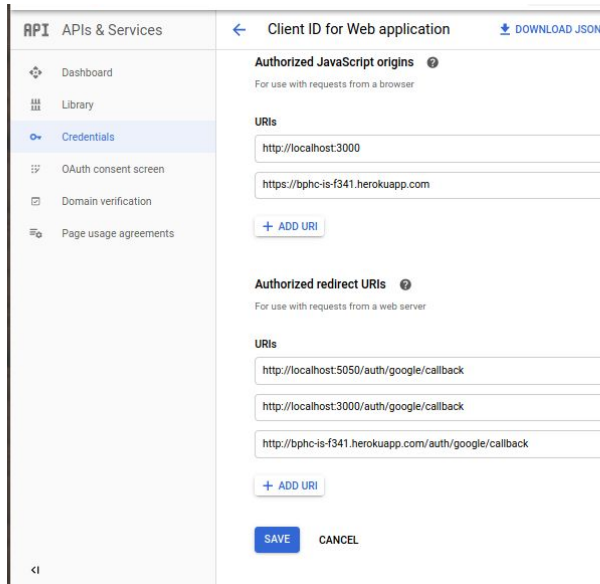
In this tutorial, you will be using PassportJS to setup Google & Facebook sign-in on the web app. This guide will continue building upon the existing CRUD app that was used in the last demo. The source code of the app [can be found here](#). [This link](#) is a deployed version of the same website. You do not necessarily have to use this sign-in mechanism for your app, but it's recommended you try to integrate at least one form of social login (Google Facebook, Github, etc).

### Step 1: Creating Google Client ID and Secret Key

- Go to [Google Developer](#) console and create your application.
- Click on the create credentials and select OAuth credentials. Fill out the details required.



- Select the type of application as web app. Then, fill out the fields as follows and click save.<sup>1</sup> Make sure to replace “bphc-is-f341.herokuapp.com” with the URL you used to deploy Heroku:



<sup>1</sup> This is a security feature. Only sign-in requests from websites listed in “Authorized Javascript Origins” are allowed and once the sign in is done, Google sends the user sign-in information to a link that we specify. That link must be specified in “authorized redirect URLs”.

- d. You'll get your API key and client secret. Copy and save them somewhere.

## Step 2: Deploy to Heroku

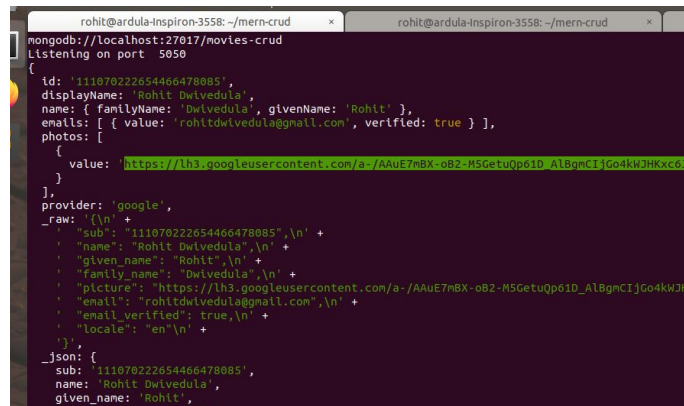
- a. Refer to the [previous guide](#) for step-by-step instructions on deploying to Heroku.
- b. We need to add two more environment variables - to store the GOOGLE\_CLIENT\_ID and the GOOGLE\_CLIENT\_SECRET - set these to the values you got from the previous step.

## How It Works

The codebase of the app has been reorganized a bit from last time - now all the frontend components of the app are located in the client folder and the backend endpoints are defined in. Firstly, we created a schema to store user info - name, email and photo. The photo field is a link to an image of that user.

```
File: models/User.js
var UserSchema = new Schema({
  name: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true,
    unique: true
  },
  photo: {
    type: String
  }
});
```

When a user signs in with Google, our website receives a bunch of information about the user, which will look something like this. The highlighted bit is the URL of the image:



```
rohit@ardula-inspiron-3558: ~/mern-crud
mongodb://localhost:27017/movies-crud
Listening on port 5050
{
  id: '111070222654466478085',
  displayName: 'Rohit Dwivedula',
  name: { familyName: 'Dwivedula', givenName: 'Rohit' },
  emails: [ { value: 'rohitdwivedula@gmail.com', verified: true } ],
  photos: [
    {
      value: 'https://lh3.googleusercontent.com/a-/AAuE7nBX-ob2-M5GetuQp61D_AlBgmCIjGo4kHJHxc630'
    }
  ],
  provider: 'google',
  _raw: '{\n' +
    '  "sub": "111070222654466478085",\n' +
    '  "name": "Rohit Dwivedula",\n' +
    '  "given_name": "Rohit",\n' +
    '  "family_name": "Dwivedula",\n' +
    '  "picture": "https://lh3.googleusercontent.com/a-/AAuE7nBX-ob2-M5GetuQp61D_AlBgmCIjGo4kHJHxc630",\n' +
    '  "email": "rohitdwivedula@gmail.com",\n' +
    '  "email_verified": true,\n' +
    '  "locale": "en"\n' +
    '}',
  _json: {
    sub: '111070222654466478085',
    name: 'Rohit Dwivedula',
    given_name: 'Rohit',
    family_name: 'Dwivedula',
    picture: 'https://lh3.googleusercontent.com/a-/AAuE7nBX-ob2-M5GetuQp61D_AlBgmCIjGo4kHJHxc630',
    email: 'rohitdwivedula@gmail.com',
    email_verified: true,
    locale: 'en'
  }
}
```

We use this data that we receive to create the users (file: passport.js)

```

User.js      x      passport.js      x
clientID: process.env.GOOGLE_CLIENT_ID,
clientSecret: process.env.GOOGLE_CLIENT_SECRET,
callbackURL: "/auth/google/callback"
},
    (accessToken, refreshToken, profile, done) => {
      console.log(profile);
      User.findOne({ email: profile.emails[0].value }).then(existingUser => {
        if (existingUser) {
          done(null, existingUser);
        } else {
          new User({
            googleId: profile.id,
            name: profile.displayName,
            email: profile.emails[0].value,
            photo: profile.photos[0].value.split("?")[0]
          })
            .save()
            .then(user => done(null, user));
        }
      });
    }
  )
);

```

Compare the format of the data we received to what we are saving. **profile** is the entire value that is received by our website (a sample value of profile is in previous screenshot). For example, `profile.emails[0]` selects the primary email of the user.

Once the website is deployed try going to `/movies` directly, without logging in - you'll find yourself being redirected to the main page. Try querying `/api/movies` - you'll see a message saying "Unauthorized".

To set up your own private (login-required) API endpoint follow a syntax similar to what is shown below. The function `isAuthenticated()` checks if the user has logged in. If that check fails, the API returns a status of 401 - Unauthorized.<sup>2</sup> (see below for image)

---

<sup>2</sup> It is customary to use status codes to summarise the status of any HTTP request. For example, status 200 is usually used to indicate that the query/request was a SUCCESS just like how 404 is used for Page Not Found. [List of all HTTP Status Codes](#).

```

1 var {Movie} = require("../models/Movie.js");
2 const passport = require("passport");
3
4 module.exports = app => {
5   app.get("/api/movies", (req,res)=>{
6     // get a list of all movies in the database
7     var movie object = {title: req.body.title, desc: req.body.desc, year: req.body.year, cast: req.body.cast
8     if(req.isAuthenticated()){
9       Movie.find({},(err,obj)) => {
10         if (err) {
11           res.send({status:false, error:err})
12         }
13         else {
14           res.send(obj);
15         }
16       }
17     });
18   }
19   else{
20     res.status(401).send("Unauthorized")
21   }
22 });
23

```

File:

/routes/api/movie.js

In the front end we use these status codes to check if a user has logged in and if he hasn't, we redirect them to the main page.

File: ~/mern-crud/client/src/components/tiles-view.js line #12 to 22.

```

componentDidMount(){
  this.setState(this.props.location.state)
  API.get('/api/movies').then((response) => {
    this.setState({error:false, movies:response.data, loadStatus:true});
    console.log(response.data);
  }).catch(function (error) {
    console.log("ERROR LOADING DATA");
    console.log(error);
    window.location = '/' // redirect to main page
  });
}

```

This code snippet is trying to get the list of movies from the API - if the fetch fails, the page redirects to the home page. [window.location]

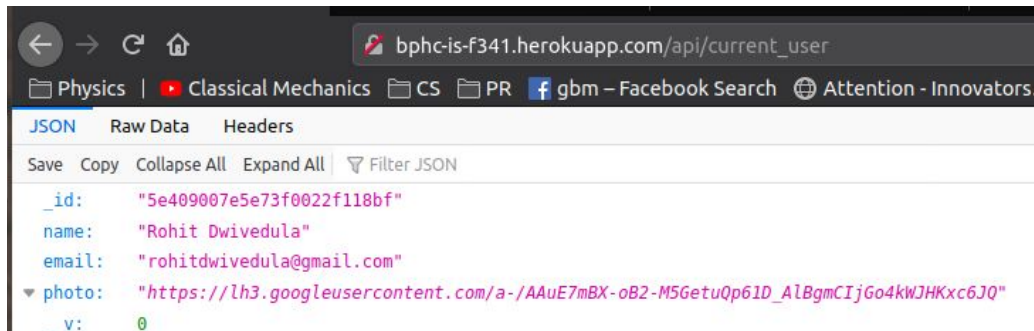
## Resources

1. [A blog app](#) built using MERN and integrated with Google OAuth2.0. If you want to test this out make sure to replace `"passport-google-oauth20": "^1.0.0"` in the package.json with `"passport-google-oauth20": "^2.0.0"`. (the Google sign-in won't work with the out-of-date library). This app is mildly buggy though - adding a blog post doesn't exactly work.
2. There are many ways authentication can be implemented - sessions and cookies are two predominant ways they're implemented. This [article](#) does a decent job of explaining

the basics. This [article](#) discusses some of the tradeoffs involved in choosing what method to use. The app in this document used a token-based authentication system.

## What Next?

1. Try integrating support for a local signup method as well, i.e. signing up with username and password.
2. The code base has an endpoint `api/current_user` - try sending a request to it from the frontend (Axios) and use the returned values to show the user's pictures and name on the component. The data returned by this endpoint looks something like this when a user is signed in:



```
{
  "_id": "5e409007e5e73f0022f118bf",
  "name": "Rohit Dwivedula",
  "email": "rohitdwivedula@gmail.com",
  "photo": "https://lh3.googleusercontent.com/a-/AAuE7mBX-oB2-M5GetuQp61D_AlBgmCIjGo4kWJHKxc6JQ",
  "v": 0
}
```

3. [Use Redux](#) in your application to manage state.