(https://andrejgajdos.com/authenticating-users-in-single-page-applications-using-node-passport-react-and-redux/)

Authenticating Users in SPA using Node, Passport, React and Redux (https://andrejgajdos.com/authenticating-users-in-single-page-applications-using-node-passport-react-and-redux/)

Last updated June 23, 2020 by Andrej Gajdos (https://andrejgajdos.com/author/and82/)

Implementing user authentication can be a difficult task, because we can use various libraries and **authentication strategies (https://github.com/jaredhanson/passport/wiki/Strategies)**. There are a lot of tutorials on this topic, but often times they miss fundamental information or don't reflect set up of our project. In this blog post, I don't attempt to write an universal tutorial for user authentication. However, I will point out some parts in authentication flow I didn't find in other tutorials.

We will be building a user authentication in a single page application with Node, React, Redux and Koa combined with Passport. I think this is a standard set up for Node.js projects, but what we will build is principally applicable to any SPA with unidirectional data flow. We will implement local authentication, where users can log in using an email and passport. We will also add authentication with Facebook, which can be used with other social networks and OAuth providers (https://en.wikipedia.org/wiki/List_of_OAuth_providers).

You can find a lot of good tutorials, which will help you implement user authentication in Node.js projects, but all these projects are multiple page applications:

- User Authentication with Passport and Koa (http://mherman.org/blog/2018/01/02/user-authentication-with-passport-and-koa/#.Ww_hwFPRAWp)
- Local Authentication Using Passport in Node.js (https://www.sitepoint.com/local-authentication-using-passport-node-js/)
- Easy Node Authentication: Setup and Local (https://scotch.io/tutorials/easy-node-authentication-setup-and-local)
- Build User Authentication with Node.js, Express, Passport, and MongoDB (https://www.ctl.io/developers/blog/post/build-user-authentication-with-node-js-express-passport-and-mongodb)
- Authenticating Users in a Node App using Express & Passport (Part One) (https://www.danielgynn.com/build-an-authentication-app-using-express-node-passport/)
- Authenticating Node.js Applications With Passport (https://code.tutsplus.com/tutorials/authenticating-nodejs-applications-with-passport--cms-21619)

If you are looking for how to implement user authentication in React.js or Redux, you will probably come across these tutorial, which are very helpful:

- Secure Your React and Redux App with JWT Authentication (https://auth0.com/blog/secure-your-react-and-redux-app-with-jwt-authentication/)
- Tips to handle Authentication in Redux #2 introducing redux-saga (https://medium.com/@MattiaManzati/tips-to-handle-authentication-in-redux-2-introducing-redux-saga-130d6872fbe7)
- ~~Protected routes and Authentication with React and Node.js (https://blog.strapi.io/protected-routes-and-authentication-with-react-and-node-js/)~~

In preview below, you can see the result of this tutorial. You can check the source code on Github (https://github.com/AndrejGajdos/auth-flow-spa-node-react) and deployed demo on ~~Heroku (https://auth-flow-in-spa.herokuapp.com/)~~.

I will be using Koa framework, which is very similar to Express (https://github.com/koajs/koa/blob/master/docs/koa-vs-express.md) and popular authentication middleware library Passport (http://www.passportjs.org/). In this project, I use Redis for:

- storing user's session information
- mocking a database with users

If you are looking for how to use PostgreSQL or MongoDB in Node.js, just check tutorials in the beginning of this article. For developing single page application, we will use React, React Router, Redux, Redux-Saga, Webpack and Twitter Bootstrap.

Getting Started

We will need to have Node (https://nodejs.org/en/), with npm (https://www.npmjs.com/), installed on our machine. We will also need to install Redis (https://redis.io/topics/quickstart). Once all of the prerequisite software is set up, we can create the node application with the following command:

```
$ npm init
```

You'll be prompted to put in basic information for Node project. We will need following dependencies.

```
{
"dependencies": {
"@koa/cors": "^2.2.1",
"axios": "^0.18.0",
"bcrypt": "^2.0.0",
"bootstrap": "^4.1.0",
"classnames": "^2.2.5",
```

```
"dotenv": "^6.0.0",
"find-config": "^1.0.0",
"jquery": "^3.3.1",
"js-cookie": "^2.2.0",
"koa": "^2.5.0",
"koa-body": "^2.5.0",
"koa-bodyparser": "^4.2.0",
"koa-logger": "^3.2.0",
"koa-passport": "^4.0.1",
"koa-redis": "^3.1.2",
"koa-router": "^7.4.0",
"koa-session": "^5.8.1",
"koa-static": "^4.0.2",
"lodash": "^4.17.10",
"passport": "^0.4.0",
"passport-facebook": "^2.1.1",
"passport-local": "^1.0.0",
"popper.js": "^1.14.3",
"react": "^16.3.2",
"react-dom": "^16.3.2",
"react-loadable": "^5.3.1",
"react-modal": "^3.4.4",
"react-redux": "^5.0.7",
"react-router": "^4.2.0",
"react-router-dom": "^4.2.2",
"react-router-redux": "^5.0.0-alpha.9",
"react-tippy": "^1.2.2",
"redis": "^2.8.0",
"redis-commands": "^1.3.5",
"redux": "^4.0.0",
"redux-thunk": "^2.2.0",
}
}
```

3c108fe56e00/raw/7c7a70554fae4c3990774e72a6f4812d8045fcee/package.json)

package.json (https://gist.github.com/AndrejGajdos/5a24ec1a0dc358a4b2748c108fe56e00#file-package-json) hosted with ♥ by GitHub (https://github.com)

For complete list, including development dependencies, check this (https://github.com/AndrejGajdos/auth-flow-spa-node-react/blob/master/package.json) file.

Now, install all the dependencies with command:

```
$ npm i
```

We will also need other configuration files for React project such as **webpack.config.js (https://github.com/AndrejGajdos/auth-flow-spa-node-react/blob/master/webpack.config.js)**, **.babelrc (https://github.com/AndrejGajdos/auth-flow-spa-node-react/blob/master/.babelrc)** and others, which you can see in **root folder (https://github.com/AndrejGajdos/auth-flow-spa-node-react)** of the project.

Below is an overview of project structure. Back-end is located in `server.js`, `serverConfig.js` and `auth.js` files.

```
├── script
│   ├── controllers
│   │   └── auth.js
│   ├── views
│   │   ├── about
│   │   │   └── AboutView.js
│   │   ├── actions
│   │   │   ├── access.actions.js
│   │   │   └── modals.actions.js
│   │   ├── components
│   │   │   ├── App
│   │   │   ├── Header
│   │   │   └── LoginForm
│   │   ├── home
│   │   │   └── HomeView.js
│   │   ├── sagas
│   │   │   ├── access.sagas.js
│   │   │   ├── index.js
│   │   │   └── modals.sagas.js
│   │   ├── state
│   │   │   ├── access.reducers.js
│   │   │   ├── index.js
│   │   │   └── modals.reducers.js
│   │   └── routes.jsx
│   └── server.js
│   └── serverConfig.js
├── index.html
├── index.jsx
├── package.json
└── webpack.config.js
```

Building and Setting up the Server

Now, when we install all npm packages, we can start to implement the server for our Node.js project. We create `server.js` file in `script` folder.

Application Setup server.js

In the beginning of `server.js` file, we just add required modules and create koa application.

```
require('dotenv').config({ path: require('find-config')('.env') });
const Koa = require('koa');
```

```
const Router = require('koa-router');
const koaLogger = require('koa-logger');
const cors = require('@koa/cors');
const bodyParser = require('koa-bodyparser');
const serve = require('koa-static');
const send = require('koa-send');
const path = require('path');
const session = require('koa-session');
const redisStore = require('koa-redis');
const ratelimit = require('koa-simple-ratelimit');
const redis = require('redis');
const config = require('./serverConfig');
const includes = require('lodash/includes');
const app = new Koa();
// this last koa middleware catches any request that isn't handled by
// koa-static or koa-router, ie your index.html in your example
app.use(function* index() {
yield send(this, '/dist/index.html');
});
// don't listen to this port if the app is required from a test script
if (!module.parent) {
app.listen(process.env.PORT || 1337);
console.log('app listen on port: 1337');
}
```

5653f80504278c8/raw/9b084855c0a899854200a08494050fc1a9c2fc9f/server.js)

server.js (https://gist.github.com/AndrejGajdos/203e96fc6c6f7cf185653f80504278c8#file-server-js) hosted with ♥ by GitHub (https://github.com)

If we open the terminal, we can run the application by command:

```
$ node ./script/server.js
```

If we want to refresh our server every time we change files, we need to use nodemon (https://nodemon.io/). Just install with: `npm install -g nodemon` and add new command (https://github.com/AndrejGajdos/auth-flow-spa-node-react/blob/master/package.json#L11) into `package.js` file:

```
"debug": "./node_modules/nodemon/bin/nodemon.js --inspect ./script/serve
```

Now we can initialize Redis and create a mock database with a user.

```
// create mock database with one user
const db = redis.createClient();
db.on('error', (err) => {
console.log(`Redis Error ${err}`);
});
db.set('usersMockDatabase', JSON.stringify([
{
id: 1,
email: 'chouomam@chouman.com',
// "test" -- generated by bcrypt calculator
password:
'$2a$04$4yQfCo8kMpH24T2iQkw9p.hPjcz10m.FcWmgkOhkXNPSpbwHZ877S',
userName: 'Chouomam',
},
]), redis.print);
module.exports = {
db,
};
```

3ed579512455f74/raw/35776cae89661f02f02a38ac3ce2da27953ec787/server.js)

server.js (https://gist.github.com/AndrejGajdos/938237a6adfff19a18ed579512455f74#file-server-js) hosted with ♥ by GitHub (https://github.com)

In the first line, we initialized Redis, then we created a new key `usersMockDatabase` with a value – user object. We should always encrypt passwords before saving them to the database. In the code snippet above, I just pasted encrypted `test` string using online ~~bcrypt-calculator (https://www.dailycred.com/article/bcrypt-calculator)~~.

Next, we move session data out of memory into an external session store (https://github.com/koajs/session#external-session-stores) Redis.

```
app.use(
```

```
session(
{
store: redisStore(),
},
app,
),
);
```

!cae6a273f0a64a4a/raw/fdf135de0991b6573b2fb1f5223ca37aed3f38ef/server.js)

server.js (https://gist.github.com/AndrejGajdos/221f9207b91ae9b4cae6a273f0a64a4a#file-server-js) hosted with ♥ by GitHub (https://github.com)

Routes app/routes.js

We are setting up the router to specify how an application responds to a client requests to a particular endpoint. We will have the following routes.

```
const auth = require('./controllers/auth');
const router = new Router();
router
/* Handle Login POST */
.post('/login', ctx => passport.authenticate('local', (err, user) => {
if (!user) {
ctx.throw(401, err);
} else {
ctx.body = user;
return ctx.login(user);
}
})(ctx))
/* GET User Profile */
.get('/users/profile', auth.getLoggedUser)
/* Handle Logout POST */
.get('/logout', (ctx) => {
ctx.logout();
ctx.body = {};
})
/* Handle Login Facebook */
.get('/auth/facebook', passport.authenticate('facebook'))
.get(
'/auth/facebook/callback',
```

```
passport.authenticate('facebook', {
successRedirect: '/facebook/success/',
failureRedirect: '/',
}),
);
app.use(router.routes()).use(router.allowedMethods());
```

3beb1a19c62ad8/raw/5186dd19143ef11122327c75eb5f0b5a1824538b/server.js)

server.js (https://gist.github.com/AndrejGajdos/74cd18b6533963c5e83beb1a19c62ad8#file-server-js) hosted with ♥ by GitHub (https://github.com)

Passport Setup

Now we initialize passport along with its session authentication middleware. I highly recommend to check this article, which explains **passport authentication flow (http://toon.io/understanding-passportjs-authentication-flow/)**.

```
const passport = require('koa-passport');
app.use(passport.initialize());
app.use(passport.session());
```

54775b18b77beb/raw/9b150395ed099ea4375931fc22ef5cf8bb499570/server.js)

server.js (https://gist.github.com/AndrejGajdos/4fadedb8ac1f504c6a54775b18b77beb#file-server-js) hosted with ♥ by GitHub (https://github.com)

In the first line, you can see, we required `./controllers/auth.js` file, where is handled all the passport implementation. This is where we configure our authentication strategy for local and facebook. We will add required libraries, modules and implement serializing and de-serializing the user information to the session (https://github.com/jaredhanson/passport#sessions).

```
const bcrypt = require('bcrypt');
const passport = require('koa-passport');
const FacebookStrategy = require('passport-facebook').Strategy;
const LocalStrategy = require('passport-local').Strategy;
```

```
const config = require('../serverConfig');
const { db } = require('../server');
const { promisify } = require('util');
const getAsync = promisify(db.get).bind(db);
passport.serializeUser((user, done) => {
done(null, user.id);
});
passport.deserializeUser(async (id, done) => {
try {
let user = null;
await getAsync('usersMockDatabase').then((users) => {
user = JSON.parse(users).find(currUser => currUser.id === id);
});
if (user) {
done(null, user);
} else {
done(null, false);
}
} catch (err) {
done(err);
}
});
```


Passport Local Strategy

Next, we define passport's strategy for handling login using a username and password. At first, we check the database for a user matching the given email. If a user with given email is found, the retrieved user's password is compared to the one provided.

```
passport.use(
new LocalStrategy(
{
usernameField: 'email',
passwordField: 'password',
```

```
},
async (email, password, done) => {
let user = null;
await getAsync('usersMockDatabase').then((users) => {
const currUsers = JSON.parse(users);
user = currUsers.find(currUser => currUser.email === email);
});
if (!user) {
done({ type: 'email', message: 'No such user found' }, false);
return;
}
if (bcrypt.compareSync(password, user.password)) {
done(null, { id: user.id, email: user.email, userName: user.userName });
} else {
done({ type: 'password', message: 'Passwords did not match' }, false);
}
},
),
);
```

3ead0a3ea8e3a5a29/raw/a42fde910a24f8f0f86214c7db4282d1b53b97ad/auth.js)

auth.js (https://gist.github.com/AndrejGajdos/e3e57a16ed18e3f3ead0a3ea8e3a5a29#file-auth-js) hosted with ♥ by GitHub (https://github.com)

Passport Facebook Strategy

Before we implement facebook strategy, we need to create a new facebook application (https://developers.facebook.com/apps/), set it up to enable OAuth and add redirect URIs. You can see more info ~~here (https://medium.com/@tkssharma/authentication-using-passport-js-social-auth-with-node-js-1e1ec7086ded)~~ and here (http://mherman.org/blog/2013/11/10/social-authentication-with-passport-dot-js/). Then we need to copy `clientID`, `clientSecret` and `callbackURL` into `.env` and `serverConfig.js` configuration files.

```
passport.use(
new FacebookStrategy(
```

```
{
clientID: config.facebookAuth.clientID,
clientSecret: config.facebookAuth.clientSecret,
callbackURL: config.facebookAuth.callbackURL,
profileFields: [
'id',
'displayName',
'picture.width(200).height(200)',
'first_name',
'middle_name',
'last_name',
'email',
],
},
(accessToken, refreshToken, profile, done) => {
process.nextTick(async () => {
const facebookUser = {
id: Math.random(),
userName: profile.displayName,
email: profile.emails[0].value,
imgUrl: profile.photos[0].value,
imgHeight: 200,
imgWidth: 200,
userProfileId: profile.id,
};
await getAsync('usersMockDatabase').then((users) => {
// save new user into database
const currUsers = JSON.parse(users);
currUsers.push(facebookUser);
db.set('usersMockDatabase', JSON.stringify(currUsers));
});
return done(null, facebookUser);
});
},
),
);
```

77b4d51fd51b433/raw/f75157b39c299671e3410b10ee51c665eb8860b2/auth.js)

auth.js

(https://gist.github.com/AndrejGajdos/1599721deaa4ec4d777b4d51fd51b433#file

auth-js) hosted with ♥ by GitHub (https://github.com)

Adding Protected Endpoints

Passport also gives the ability to protect access to a specific route. It means that if user tries to access `http://localhost:1337/users/profile` without authenticating, he will be redirected to home page by doing:

```
exports.getLoggedUser = async (ctx) => {
if (ctx.isAuthenticated()) {
const reqUserId = ctx.req.user.id;
let user = null;
await getAsync('usersMockDatabase').then((users) => {
user = JSON.parse(users).find(currUser => currUser.id === reqUserId);
});
if (user) {
delete user.password;
ctx.response.body = user;
} else {
const statusCode = 500;
ctx.throw(statusCode, "User doesn't exist");
}
} else {
ctx.redirect('/');
}
};
```

auth.js

628a850dc5b7135/raw/4ec93d54289ba4054142c0b601fee2bd36919ca2/auth.js)
(https://gist.github.com/AndrejGajdos/a7cd27b4b2997564c628a850dc5b7135#file
auth-js) hosted with ♥ by GitHub (https://github.com)

Rate Limiting

We should also implement Rate limiting (https://en.wikipedia.org/wiki/Rate_limiting) to control how many requests a given consumer can send to the API. All successful API requests include the following three headers with details about the current rate limit status:

- `X-Rate-Limit-Limit` – the number of requests allowed in a given time interval
- `X-Rate-Limit-Remaining` – how many calls user has remaining in the same interval
- `X-Rate-Limit-Reset` – the time when the rate limit will be reset.

Most HTTP frameworks support it out of the box. For example, if you are using Koa and Redis, there is the koa-simple-ratelimit (https://github.com/scttcper/koa-simple-ratelimit) package.

```
app.use(ratelimit({
db,
duration: 60000,
max: 100,
}));
```

1a9ec32ec1597e/raw/4066709ef2e0a4dd3c534e74b6183f917518ec68/server.js)

server.js (https://gist.github.com/AndrejGajdos/9eb7b3cb7d8db60d441a9ec32ec1597e#file-server-js) hosted with ♥ by GitHub (https://github.com)

CORS

When we develop single page applications, we usually deploy front-end code on different server than API. In this case, we need to allow CORS (https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS) and enable Access-Control-Allow-Credentials (https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Credentials).

```
const corsOptions = {
credentials: true,
};
app.use(cors(corsOptions));
```

server.js

:a0863d76595b6/raw/17b3a0a9238f8b6379e612236152063044155f14596d6/server.js)
(https://gist.github.com/AndrejGajdos/0ee2432d1583463431551a0863d76595b6#file-

server-js) hosted with ❤ by GitHub (https://github.com)

Building and Setting up the SPA in React and Redux

Back-end part in Node.js is ready and now we can implement SPA in React and Redux. We will focus on unidirectional data flow in single page applications. The same principles can be applied to other frameworks or libraries for implementing unidirectional data flow such as Flux or MobX. In this project I decided to use popular libraries React, React-Router, Redux and Redux-Saga. Redux is a state container for JavaScript applicatins that describes the state of the application as a single object and Redux-Saga is used to make handling side effects in Redux nice and simple.

Let’s start with implementing React entry point with views and other components.

Main React Entry File

At first we define `index.jsx` , which bootstraps the react + redux application by rendering the `App` component (wrapped in a redux ~~Provider (https://github.com/reduxjs/react-redux/blob/master/docs/api.md#provider-store)~~) into the `div` element defined in the base `index.html` .

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>AUTH FLOW</title>
```

```
<meta name="description" content="">
<!-- Mobile-friendly viewport -->
<meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
<div id="app">
</div>
<script type="text/javascript" src="/bundle.js"></script>
</body>
</html>
```

38f6864313efb/raw/e930834f941464eef39334786ecde6bc32f49da2/index.html)

index.html

(https://gist.github.com/AndrejGajdos/2f795a4325afb9defb838f6864313efb#file-index-html) hosted with ♥ by GitHub (https://github.com)

```
import React from 'react';
import ReactDOM from 'react-dom';
import createHistory from 'history/createBrowserHistory';
import { createStore, combineReducers, applyMiddleware } from 'redux';
import { ConnectedRouter, routerReducer, routerMiddleware } from 'react-
router-redux';
import { Provider } from 'react-redux';
import createSagaMiddleware from 'redux-saga';
import { composeWithDevTools } from 'redux-devtools-
extension/developmentOnly';
import 'bootstrap';
import 'bootstrap/dist/css/bootstrap.css';
import 'react-tippy/dist/tippy.css';
import App from 'components/App';
import reducers from 'state/index';
import sagas from 'sagas/index';
const sagaMiddleware = createSagaMiddleware();
const history = createHistory();
const store = createStore(
combineReducers({
...reducers,
router: routerReducer,
}),
composeWithDevTools(applyMiddleware(routerMiddleware(history),
sagaMiddleware)),
);
sagaMiddleware.run(sagas);
ReactDOM.render(
```

```
<Provider store={store}>
<ConnectedRouter history={history}>
<div>
<App />
</div>
</ConnectedRouter>
</Provider>,
document.getElementById('app'),
);
```

7402c6127b8a69/raw/f35e3513dfc466b94f0324998d6bac4406da7c42/index.jsx)

index.jsx (https://gist.github.com/AndrejGajdos/c7c37f74f99e0120817402c6127b8a69#file-index-jsx) hosted with ❤ by GitHub (https://github.com)

React + Redux App Component

The `App` component is the root component for the React application, it contains routes, global alert notification and modal window for logging.

```
import React, { Component } from 'react';
import Header from 'components/Header';
import Routes from 'routes';
import LoginFormModal from 'components/LoginForm';
import './App.scss';
export class App extends Component {
state = {
error: null,
errorInfo: null,
};
componentDidCatch(error, errorInfo) {
this.setState({ error, errorInfo });
}
render() {
const { error, errorInfo } = this.state;
return (
<div>
<Header />
<main>
{!error && <Routes />}
{error && (
```

```
<div className="App__error container mt-3">
<div role="alert" className="alert alert-danger">
<h4>An error occurred. Please reload the page and try again.</h4>
<p className="App__stacktrace">
{process.env.NODE_ENV === 'development' && errorInfo.componentStack}
</p>
</div>
</div>
)}
</main>
<LoginFormModal />
</div>
);
}
}
export default App;
```

13d93ff9c4568fc9/raw/59fddcbed7790d161a87a2f12e6b723234792b8d/App.jsx)

App.jsx (https://gist.github.com/AndrejGajdos/48322de578fdcc6a13d93ff9c4568fc9#file-app-jsx) hosted with ❤ by GitHub (https://github.com)

React + Redux Home Page and About Page

Now we will add home and about views. In home view we added `getProfile` action, because we want see user profile data on this page.

```
import React from 'react';
export default function AboutView() {
return (
<div className="about-view m-3">
<h1>ABOUT PAGE</h1>
</div>
);
}
```

75dc05238/raw/1574e7576f071c87855d693930c879d11d3685bf/AboutView.jsx)

AboutView.jsx (https://gist.github.com/AndrejGajdos/351856b17e24e9ce933896c75dc05238#file

aboutview-jsx) hosted with ❤ by GitHub (https://github.com)

```
import React, { Component } from 'react';
import PropTypes from 'prop-types';
import { connect } from 'react-redux';
import { AsyncAboutView } from 'asyncViews';
import { getProfile } from 'actions/access.actions';
import get from 'lodash/get';
import ProfilePlaceholder from 'assets/img/profile_placeholder.png';
import { isValidURL } from 'utils/utils';
import './homeView.scss';
class HomeView extends Component {
static propTypes = {
userObj: PropTypes.object,
getProfile: PropTypes.func,
};
static defaultProps = {
userObj: {},
getProfile: () => null,
};
state = {
showProfile: false,
};
componentDidMount() {
AsyncAboutView.preload();
}
getProfile = () => {
this.setState({ showProfile: true }, () => {
this.props.getProfile();
});
};
render() {
const { userObj } = this.props;
const { showProfile } = this.state;
let profileImg = null;
let profileWidth = 0;
let profileHeight = 0;
let backgroundSize = null;
if (get(userObj, 'isAuthenticated')) {
if (get(userObj, 'loggedUserObj.imgUrl') &&
isValidURL(userObj.loggedUserObj.imgUrl)) {
profileImg = userObj.loggedUserObj.imgUrl;
profileWidth = userObj.loggedUserObj.imgWidth;
profileHeight = userObj.loggedUserObj.imgHeight;
} else {
```

```
profileImg = ProfilePlaceholder;
}
backgroundSize = 'auto 60px';
if (profileWidth < profileHeight) {
backgroundSize = '60px auto';
}
}
return (
<div className="HomeView m-3">
<h1>HOME PAGE</h1>
{get(userObj, 'isAuthenticated') && (
<button type="button" className="btn btn-outline-dark mt-3" onClick=
{this.getProfile}>
Get Profile
</button>
)}
{get(userObj, 'isAuthenticated') &&
showProfile && (
<div className="d-flex mt-3">
<style
dangerouslySetInnerHTML={{
__html: [
'.ProfileImg {',
` background-image: url(${profileImg});`,
` background-size: ${backgroundSize};`,
'}',
].join('\n'),
}}
/>
<div className="ProfileImg mr-2 rounded-circle" />
<div className="UserInfo d-flex flex-column justify-content-center ml-2">
{get(userObj, 'loggedUserObj.userName') && (
<div className="UserName">
<span>UserName: </span>
<strong>{userObj.loggedUserObj.userName}</strong>
</div>
)}
{get(userObj, 'loggedUserObj.email') && (
<div className="Email">
<span>Email: </span>
<strong>{userObj.loggedUserObj.email}</strong>
</div>
)}
</div>
</div>
```

```
)}
</div>
);
}
}
const mapStateToProps = state => ({
userObj: state.access.user,
});
const mapDispatchToProps = dispatch => ({
getProfile: () => dispatch(getProfile()),
});
export default connect(
mapStateToProps,
mapDispatchToProps,
)(HomeView);
```

2702a0e32/raw/15b99bb0d9289e48944d8e09f5cd736aa6759e01/HomeView.jsx)

HomeView.jsx (https://gist.github.com/AndrejGajdos/3760f04a1b21cc6bfe9107c2702a0e32#file-homeview-jsx) hosted with ♥ by GitHub (https://github.com)

React + Redux LoginForm Component

Next we can implement modal window, where users can log in.

```
import React, { Component } from 'react';
import PropTypes from 'prop-types';
import { withRouter } from 'react-router';
import { connect } from 'react-redux';
import Cookies from 'js-cookie';
import get from 'lodash/get';
import includes from 'lodash/includes';
import { login } from 'actions/access.actions';
import toggleLogin from 'actions/modals.actions';
import cn from 'classnames';
import ReactModal from 'react-modal';
import { Tooltip } from 'react-tippy';
import './loginForm.scss';
class LoginForm extends Component {
static propTypes = {
loginUser: PropTypes.func,
```

```
toggleLogin: PropTypes.func,
openLogin: PropTypes.bool.isRequired,
userObj: PropTypes.object,
error: PropTypes.any,
};
static defaultProps = {
error: null,
userObj: {},
loginUser: () => null,
toggleLogin: () => null,
};
state = {
error: null,
};
componentWillReceiveProps(nextProps) {
if (nextProps.userObj && nextProps.userObj.isAuthenticated) {
this.closeModal();
}
if (nextProps.error) {
this.setState({ error: nextProps.error });
}
}
handleInputChange = (e) => {
const nextState = {};
nextState[e.target.name] = e.target.value;
this.setState(nextState);
};
closeModal = () => {
this.props.toggleLogin(false);
};
handleSubmitForm = (e) => {
e.preventDefault();
this.props.loginUser(this.email.value, this.password.value);
};
handleFocusInput = (e) => {
const label = document.querySelector(`[for=${e.target.id}]`);
if (!label.classList.contains('active')) {
label.classList.add('active');
}
this.resetError();
};
handleBlurInput = (e) => {
if (!e.target.value) {
document.querySelector(`[for=${e.target.id}]`).classList.remove('active');
}
```

```
};
onClickModalWindow = () => {
this.resetError();
};
resetError = () => {
if (this.errorElement && this.errorElement.length > 0) {
this.setState({ error: null });
}
};
onFacebookLogin = () => {
const inOneHour = new Date(new Date().getTime() + 60 * 60 * 1000);
Cookies.set('lastLocation_before_logging', this.props.location.pathname, {
expires: inOneHour });
window.location.href = `${window.location.origin}/auth/facebook`;
};
render() {
const { openLogin } = this.props;
const { error } = this.state;
const customModalStyle = {
overlay: {
backgroundColor: 'rgba(0, 0, 0, 0.75)',
},
};
const errorMessage = error ? error.message : '';
return (
<ReactModal
isOpen={openLogin}
contentLabel="Modal"
ariaHideApp={false}
closeTimeoutMS={500}
onRequestClose={this.closeModal}
style={customModalStyle}
className={{
base: 'loginForm',
afterOpen: 'loginForm_after-open',
beforeClose: 'loginForm_before-close',
}}
>
<div onClick={this.onClickModalWindow}>
<h5 className="loginForm__heading mx-auto mb-2 text-center">Sign
in</h5>
<div className="loginForm__formContainer d-flex flex-column px-3 py-4">
<button
type="button"
className="btn facebook-login-container mx-auto mt-2 rounded"
```

```
onClick={this.onFacebookLogin}
>
Continue with Facebook
</button>
<div className="form-divider mt-3">
<span className="d-flex flex-row">
<strong className="loginForm__dividerLabel">or</strong>
</span>
</div>
<form className="loginForm__form d-flex flex-column mx-auto mb-2"
onSubmit={this.handleSubmitForm}>
<span className="loginForm__formHeader my-3 text-center">Sign in with
your email</span>
<div className="form-group">
<div
className={cn('form-group', 'mb-4', {
'has-error': includes(get(error, 'type'), 'email'),
})}
>
<Tooltip
html={<span>{errorMessage}</span>}
open={includes(get(error, 'type'), 'email')}
onRequestClose={() => this.setState({ error: null })}
>
<input
type="email"
className="form-control floatLabel"
id="registerInputEmail"
required
onChange={this.handleInputChange}
onFocus={this.handleFocusInput}
onBlur={this.handleBlurInput}
autoComplete="email"
ref={el => (this.email = el)}
/>
<label htmlFor="registerInputEmail">Email</label>
</Tooltip>
</div>
<div className={cn('form-group', { 'has-error': includes(get(error, 'type'),
'password') })}>
<Tooltip
html={<span>{errorMessage}</span>}
open={includes(get(error, 'type'), 'password')}
onRequestClose={() => this.setState({ error: null })}
>
```

```
<input
type="password"
className="form-control floatLabel mt-2"
id="registerInputPassword"
required
onChange={this.handleInputChange}
onFocus={this.handleFocusInput}
onBlur={this.handleBlurInput}
autoComplete="current-password"
ref={el => (this.password = el)}
/>
<label htmlFor="registerInputPassword">Password</label>
</Tooltip>
</div>
</div>
<button type="submit" className="btn loginForm__signIn">
Sign in
</button>
</form>
</div>
</div>
</ReactModal>
);
}
}
const mapStateToProps = state => ({
userObj: state.access.user,
error: state.access.error,
openLogin: state.toggleModal.login,
});
const mapDispatchToProps = dispatch => ({
loginUser: (email, password) => dispatch(login(email, password)),
toggleLogin: newState => dispatch(toggleLogin(newState)),
});
export default withRouter(
connect(
mapStateToProps,
mapDispatchToProps,
)(LoginForm),
);
```

0390bd98c/raw/124560c82fee42d8aa9583edb5d756c939430047/LoginForm.jsx)

LoginForm.jsx (https://gist.github.com/AndrejGajdos/2dbd34af3cb99f959120ce50390bd98c#file-loginform-jsx) hosted with ♥ by GitHub (https://github.com)

In `LoginForm` we implemented actions `loginUser` used for logging user with email and password. Another action `toggleLogin` is used for changing modal open state. Let's say we want to display modal window for logging on other pages and in this case it is better to keep modal open state in a reducer instead of component's state.

We don't use an action for logging users with Facebook, you can see the explanation below. We need to save our current URL location in cookies, which we will need after Facebook successful login redirect.

```
onFacebookLogin = () => {
const inOneHour = new Date(new Date().getTime() + 60 * 60 * 1000);
Cookies.set('lastLocation_before_logging', this.props.location.pathname, {
expires: inOneHour });
window.location.href = `${window.location.origin}/auth/facebook`;
};
```

)f8a3c80dd/raw/b8243b75dbec82d6639148edbc1a73df94959d84/LoginForm.jsx)

LoginForm.jsx (https://gist.github.com/AndrejGajdos/529c3d4d9e6cd161b7ccc59f8a3c80dd#file-loginform-jsx) hosted with ♥ by GitHub (https://github.com)

> *" Why is xhr getting blocked on the same url a browser can access?*
>
> *Because it is a cross-domain request, and as such the remote party would have to allow that request first, which is what is referred to as CORS.*
>
> *Facebook does not allow its login dialog to be loaded via script from different domains – for the obvious reason that users need to be able to be verify which site they are sending their login credentials to via the browser address bar, to avoid phishing.*
>
> *You can not load the FB login dialog via XHR/AJAX in the background; you need to call/redirect to it in the top window instance. "*

stackoverflow.com (https://stackoverflow.com/a/39221789/968379)

Redux Actions Folder

Now we can implement actions.

```
import * as ActionTypes from 'constants/actionTypes';
export const login = (email, password) => ({
type: ActionTypes.LOGIN_REQUESTED,
email,
password,
});
export const logout = () => ({
type: ActionTypes.LOGOUT_REQUESTED,
});
export const getProfile = () => ({
type: ActionTypes.PROFILE_REQUESTED,
});
```

1933ad9f/raw/70ac5598f40ab09e1841e93c10eec8b4c72ed62a/access.actions.js)

access.actions.js

(https://gist.github.com/AndrejGajdos/e73d81235b8421cedab6dee11933ad9f#file-

access-actions-js) hosted with ♥ by GitHub (https://github.com)

```
import * as ActionTypes from 'constants/actionTypes';
const toggleLogin = newState => ({
type: ActionTypes.TOGGLE_MODAL_REQUESTED,
newState,
});
export default toggleLogin;
```

57ad0be2/raw/970f31a881e66e562142c16c537a191ffa11462d/modals.actions.js)

modals.actions.js

(https://gist.github.com/AndrejGajdos/953f84298dd231f323f2ae8757ad0be2#file-modals-actions-js) hosted with ♥ by GitHub (https://github.com)

Redux Sagas Folder

Next we will implement sagas.

```
import { all, fork } from 'redux-saga/effects';
import { watchGetProfile, watchLogout, watchLogin } from './access.sagas';
import watchToggleModal from './modals.sagas';
export default function* rootSaga() {
yield all([
fork(watchGetProfile),
fork(watchLogin),
fork(watchLogout),
fork(watchToggleModal),
]);
}
```

4ed0bccdc9b7512/raw/b624653ab05cac91de5a5b1828aee0210e3465ac/index.js)

index.js

(https://gist.github.com/AndrejGajdos/9314338e6d1f0956e4ed0bccdc9b7512#file-index-js) hosted with ♥ by GitHub (https://github.com)

```
import { put, takeLatest } from 'redux-saga/effects';
import * as ActionTypes from 'constants/actionTypes';
function* toggleModal(action) {
const { newState } = action;
if (!newState) {
// if login modal is closed, reset error
```

```
yield put({ type: ActionTypes.LOGIN_FAILED, error: null });
}
yield put({ type: ActionTypes.TOGGLE_MODAL_SUCCEEDED, newState });
}
export default function* watchToggleModal() {
yield takeLatest(ActionTypes.TOGGLE_MODAL_REQUESTED, toggleModal);
}
```

:3787caf04/raw/ea5e4cca445e9c9fe4911c9414a18c9169c23929/modals.sagas.js)

modals.sagas.js

(https://gist.github.com/AndrejGajdos/665860d737649b9513365ac3787caf04#file

modals-sagas-js) hosted with ♥ by GitHub (https://github.com)

```
import { put, call, takeLatest } from 'redux-saga/effects';
import Cookies from 'js-cookie';
import * as ActionTypes from 'constants/actionTypes';
import { get, post } from 'utils/api';
import lGet from 'lodash/get';
function* login(action) {
const { email, password } = action;
try {
const response = yield call(
post,
'/login',
{ email, password },
);
const inOneWeek = new Date(new Date().getTime() + (1000 * 60 * 60 * 24 *
7));
Cookies.set('auth__flow__spa__loggedUserObj', response.data, { expires:
inOneWeek });
yield put({ type: ActionTypes.LOGIN_SUCCEEDED, user: response.data });
} catch (error) {
if (lGet(error.response, 'data')) {
yield put({ type: ActionTypes.LOGIN_FAILED, error: error.response.data });
} else {
yield put({ type: ActionTypes.LOGIN_FAILED, error });
}
}
}
export function* watchLogin() {
yield takeLatest(ActionTypes.LOGIN_REQUESTED, login);
}
function* logout() {
try {
```

```
yield call(
get,
'/logout',
);
Cookies.remove('auth__flow__spa__loggedUserObj');
yield put({ type: ActionTypes.LOGOUT_SUCCEEDED });
} catch (error) {
if (lGet(error.response, 'data')) {
yield put({ type: ActionTypes.LOGOUT_FAILED, error: error.response.data });
} else {
yield put({ type: ActionTypes.LOGOUT_FAILED, error: error.response });
}
}
}
export function* watchLogout() {
yield takeLatest(ActionTypes.LOGOUT_REQUESTED, logout);
}
function* getProfile() {
try {
const response = yield call(
get,
'/users/profile',
);
const inOneWeek = new Date(new Date().getTime() + 1000 * 60 * 60 * 24 *
7);
Cookies.set('auth__flow__spa__loggedUserObj', response, { expires:
inOneWeek });
yield put({ type: ActionTypes.PROFILE_SUCCEEDED, user: response });
} catch (error) {
yield put({ type: ActionTypes.PROFILE_FAILED, error: error.response });
}
}
export function* watchGetProfile() {
yield takeLatest(ActionTypes.PROFILE_REQUESTED, getProfile);
}
```

0d31a7f0e/raw/8e0681dd21c1514e51e785004c020aa2a28d1fed/access.sagas.js)

access.sagas.js (https://gist.github.com/AndrejGajdos/e5ee07616d34e01b3f257c50d31a7f0e#file-access-sagas-js) hosted with ♥ by GitHub (https://github.com)

We don't want to lose user data, when we refresh a page after login. If we take a look at login (https://github.com/AndrejGajdos/auth-flow-spa-node-react/blob/master/script/views/sagas/access.sagas.js#L7) saga worker, we can see that response data, which contains user object is stored in cookies. We can store user data somewhere else, but we shouldn't do it in the reducer, because reducers should have no side effects. We need to get user data in a reducer, when we initialize state (https://redux.js.org/recipes/structuring-reducers/initializing-state).

Ajax API Calls

Now we can implement API calls.

```
import axios from 'axios';
import { API_ADDRESS } from 'config';
export function get(path, params) {
const url = `${API_ADDRESS}${path}`;
return axios({
method: 'get',
url,
params,
withCredentials: true,
}).then(resp => resp.data);
}
export function post(path, data, params) {
const url = `${API_ADDRESS}${path}`;
return axios({
method: 'post',
url,
data,
params,
withCredentials: true,
});
}
```

69221feea36d8fe4f9/raw/4091a8720f1a9f1b450924101573fbaf4902707d/api.js)

api.js

(https://gist.github.com/AndrejGajdos/a74c4af8e7a7f769221feea36d8fe4f9#file-

api-js) hosted with ♥ by GitHub (https://github.com)

As you can see, we are using withCredentials (https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/withCredentials) property. If we use CORS and don't set `withCredentials` to `true`, isAuthenticated returns false and this passport's method doesn't work.

Redux Reducers Folder

Next, we can implement reducers.

```
import access from 'state/access.reducers';
import toggleModal from 'state/modals.reducers';
export default {
access,
toggleModal,
};
```

1590ead36cc1b99/raw/3a84597f90320d28165170a6b3fd0e3505ff196b/index.js)

index.js

(https://gist.github.com/AndrejGajdos/bdab2dce6cf11df9b1590ead36cc1b99#file-index-js) hosted with ♥ by GitHub (https://github.com)

```
import * as ActionTypes from '../constants/actionTypes';
export default function toggleModal(state = { login: false }, action) {
switch (action.type) {
case ActionTypes.TOGGLE_MODAL_SUCCEEDED:
return { login: action.newState };
default:
return state;
}
}
```

103ce/raw/92182d83c02d50883564328cbba131a198635131/modals.reducers.js)

modals.reducers.js

(https://gist.github.com/AndrejGajdos/ee010509a997c9cda2924f5eb95103ce#file-modals-reducers-js) hosted with ♥ by GitHub (https://github.com)

```
import Cookies from 'js-cookie';
import * as ActionTypes from '../constants/actionTypes';
```

```
const initialState = {
user: {
isAuthenticated: typeof Cookies.get('auth__flow__spa__loggedUserObj') !==
'undefined',
loggedUserObj: Cookies.getJSON('auth__flow__spa__loggedUserObj'),
},
error: null,
};
export default function access(state = initialState, action) {
switch (action.type) {
case ActionTypes.LOGIN_SUCCEEDED:
case ActionTypes.PROFILE_SUCCEEDED: {
return {
...state,
user: {
...state.user,
isAuthenticated: true,
loggedUserObj: action.user,
},
error: null,
};
}
case ActionTypes.LOGIN_FAILED: {
return {
...state,
error: action.error,
};
}
case ActionTypes.PROFILE_FAILED:
case ActionTypes.LOGOUT_SUCCEEDED: {
return {
...state,
user: {
isAuthenticated: false,
},
error: null,
};
}
default:
return state;
}
}
```

)78d93c/raw/97719b3594d6a1e697f5cb7d970bb77c705aa30d/access.reducers.js)

access.reducers.js

(https://gist.github.com/AndrejGajdos/dd3d83296cc87109b433b9a08b78d93c#file

access-reducers-js) hosted with ♥ by GitHub (https://github.com)

React Router Routes

Now we can define routes.

```
import React from 'react';
import Loadable from 'react-loadable';
const Loading = () => (
<div />
);
export const AsyncAboutView = Loadable({
loader: () => import('about/AboutView'),
loading: () => <Loading />,
});
export const AsyncHomeView = Loadable({
loader: () => import('home/HomeView'),
loading: () => <Loading />,
});
```

ea4e909828/raw/fd5cb65125fce7b41dff236c116562dd049f9118/asyncViews.jsx)

asyncViews.jsx

(https://gist.github.com/AndrejGajdos/67386e94f76c53f6377714ea4e909828#file-

asyncviews-jsx) hosted with ♥ by GitHub (https://github.com)

```
import React from 'react';
import { Switch, Route } from 'react-router-dom';
import Cookies from 'js-cookie';
import { Redirect } from 'react-router';
import { AsyncHomeView, AsyncAboutView } from 'asyncViews';
export default function Routes() {
return (
<Switch>
<Route path="/" exact component={AsyncHomeView} />
<Route path="/about" component={AsyncAboutView} />
<Route
path="/facebook/success/"
render={() => (
<Redirect
```

```
to={{
pathname: Cookies.get('lastLocation_before_logging'),
state: { loadUser: true },
}}
/>
)}
/>
</Switch>
);
}
```

5ad0074a3f74f112/raw/aca939cc71f8716c2259c0566f55cfda06cf402e/routes.jsx)

routes.jsx (https://gist.github.com/AndrejGajdos/9ca6bfc98bf62abb5ad0074a3f74f112#file-routes-jsx) hosted with ♥ by GitHub (https://github.com)

After facebook successful redirect, we want to enter the page where we logged in (https://gist.github.com/AndrejGajdos/529c3d4d9e6cd161b7ccc59f8a3c80dd#file-loginform-jsx-L3). If we want to recognize in a React component, that we were redirected, we can use location state (https://github.com/ReactTraining/react-router/blob/master/packages/react-router/docs/api/location.md).

After successful redirecting, we set a state `loadUser`, which will be used in `componentWillReceiveProps` method in `Header` component to get user profile data.

If we want to make routes accessible only to authenticated users, we need to check documentation or tutorials on protected (authentication) routes (https://reacttraining.com/react-router/web/example/auth-workflow) for our routing library.

React + Redux Header Component

```
import React, { Component } from 'react';
import PropTypes from 'prop-types';
import { withRouter } from 'react-router';
import { connect } from 'react-redux';
import { NavLink } from 'react-router-dom';
```

```
import get from 'lodash/get';
import ProfilePlaceholder from 'assets/img/profile_placeholder.png';
import { isValidURL } from 'utils/utils';
import { logout, getProfile } from 'actions/access.actions';
import toggleLogin from 'actions/modals.actions';
import './header.scss';
class Header extends Component {
static propTypes = {
/* Router */
location: PropTypes.any.isRequired,
history: PropTypes.any.isRequired,
/* Redux */
userObj: PropTypes.object,
logoutUser: PropTypes.func,
getProfile: PropTypes.func,
toggleLogin: PropTypes.func,
};
static defaultProps = {
userObj: {},
logoutUser: () => null,
toggleLogin: () => null,
getProfile: () => null,
};
componentWillReceiveProps(nextProps) {
if (get(nextProps.location, 'state.loadUser')) {
this.props.getProfile();
this.props.history.replace({ state: null });
}
}
handleLoginClick = () => {
this.props.toggleLogin(true);
};
onLogoutClick = () => {
this.props.logoutUser();
};
render() {
const { userObj } = this.props;
let userInfoEle = null;
let profileImg = null;
let profileWidth = 0;
let profileHeight = 0;
if (get(userObj, 'isAuthenticated')) {
if (get(userObj, 'loggedUserObj.imgUrl') &&
isValidURL(userObj.loggedUserObj.imgUrl)) {
profileImg = userObj.loggedUserObj.imgUrl;
```

```
profileWidth = userObj.loggedUserObj.imgWidth;
profileHeight = userObj.loggedUserObj.imgHeight;
} else {
profileImg = ProfilePlaceholder;
}
let name = get(userObj, 'loggedUserObj.email');
if (get(userObj, 'loggedUserObj.userName')) {
name = userObj.loggedUserObj.userName;
}
let backgroundSize = 'auto 30px';
if (profileWidth < profileHeight) {
backgroundSize = '30px auto';
}
userInfoEle = (
<div className="navbar-signed d-flex">
<style
dangerouslySetInnerHTML={{
__html: [
'.img-profile {',
` background-image: url(${profileImg});`,
` background-size: ${backgroundSize};`,
'}',
].join('\n'),
}}
/>
<div className="img-profile mr-2 rounded-circle" />
<div className="dropdown profile-label d-flex align-items-center">
<a
href="#"
className="dropdown-toggle Header__dropdown"
data-toggle="dropdown"
role="button"
aria-haspopup="true"
aria-expanded="false"
>
{name}
</a>
<div className="dropdown-menu dropdown-menu-right pull-right">
<li className="dropdown-menu__item text-center">
<button className="btn w-100 rounded-0" onClick={this.onLogoutClick}>
Logout
</button>
</li>
</div>
</div>
```

```
</div>
);
} else {
userInfoEle = (
<ul className="navbar-nav navbar-right">
<li>
<a onClick={this.handleLoginClick}>Sign&amp;nbsp;in</a>
</li>
</ul>
);
}
return (
<div className="Header">
<header role="navigation">
{/*
********** NAVBAR MENU
*/}
<nav className="navbar navbar-light navbar-expand-lg bg-light">
<div className="container-fluid">
<a className="navbar-brand" href="#">
Navbar
</a>
<button
className="navbar-toggler"
type="button"
data-toggle="collapse"
data-target="#bs-example-navbar-collapse-1"
aria-controls="bs-example-navbar-collapse-1"
aria-expanded="false"
aria-label="Toggle navigation"
>
<span className="navbar-toggler-icon" />
</button>
{/* Collect the nav links, forms, and other content for toggling */}
<div className="collapse navbar-collapse" id="bs-example-navbar-collapse-
1">
<ul className="nav navbar-nav navbar-left h-100">
<li>
<NavLink className="Header__navLink pl-2 ml-2" to="/" exact>
Home
</NavLink>
</li>
<li>
<NavLink className="Header__navLink pl-2 ml-2" to="/about">
About
```

```
</NavLink>
</li>
</ul>
</div>
{userInfoEle}
</div>
</nav>
</header>
</div>
);
}
}
const mapStateToProps = state => ({
userObj: state.access.user,
});
const mapDispatchToProps = dispatch => ({
logoutUser: () => dispatch(logout()),
toggleLogin: newState => dispatch(toggleLogin(newState)),
getProfile: () => dispatch(getProfile()),
});
export default withRouter(
connect(
mapStateToProps,
mapDispatchToProps,
)(Header),
);
```

6dcdf858146f/raw/8770e00fc8e33d5b1da393bd511a78309b312bb0/Header.jsx)

Header.jsx (https://gist.github.com/AndrejGajdos/34060f74a454ea6af9b26dcdf858146f#file-header-jsx) hosted with ♥ by GitHub (https://github.com)

Conclusion

In this tutorial we implemented user authentication in a single page application using Node, Passport, React and Redux. The purpose of this article was to provide whole solution for authenticating with an email and password, and authenticating with an OAuth provider. We shouldn't forget to implement rate limiting. If we host back-end and front-end on different servers, we need to use

CORS and enable http headers for cross-site authentication. We can apply this tutorial in any SPA with unidirectional data flow. If you see any ways to improve this article, let me know please.

‹ List of Free Online Tools For Front-end Web Development (updated 2020) (https://andrejgajdos.com/list-of-free-online-tools-for-front-end-web-development/)

The Toolkit of a Freelance Full Stack Web Developer (https://andrejgajdos.com/the-toolkit-of-a-freelance-full-stack-web-developer/) ›

authentication (https://andrejgajdos.com/tag/authentication/) koa (https://andrejgajdos.com/tag/koa/) koa.js (https://andrejgajdos.com/tag/koa-js/) node (https://andrejgajdos.com/tag/node/) node.js (https://andrejgajdos.com/tag/node-js/) oauth (https://andrejgajdos.com/tag/oauth/) passport (https://andrejgajdos.com/tag/passport/) passport-facebook (https://andrejgajdos.com/tag/passport-facebook/) passport-local (https://andrejgajdos.com/tag/passport-local/) passport.js (https://andrejgajdos.com/tag/passport-js/) React (https://andrejgajdos.com/tag/react/) react-router (https://andrejgajdos.com/tag/react-router/) React.js (https://andrejgajdos.com/tag/react-js/) redux (https://andrejgajdos.com/tag/redux/) redux-saga (https://andrejgajdos.com/tag/redux-saga/)