

Authentication Using Passport

Introduction

We have already done manual authentication in the app, now it is time to make the code or application more efficient and secure using a library called **Passport**. We will be setting up a passport for the first time in the application.

Note :

We have to checkout to the master branch back because we have created the manual local authentication branch at a point where the common code is in the master branch. Till signup, the code remains the same when we put in the authentication that is where we differentiate the code.

Glancing through and Installing Passport.js

- Passport is an authenticated middleware for Node.js
- Install the passport.js library using the command { **npm install passport** } in the terminal.
- Passport can be used just for local authentication or we can use different other ways to sign in. We will be learning two ways which are:
 - **passport-local** authentication, and
 - **passport-google-OAuth**.
- Install the passport-local using the command { **npm install passport-local** } in the terminal.

EXTRA: You can look at these requests from the link below -

<http://www.passportjs.org/docs/>

<http://www.passportjs.org/packages/passport-local/>

Setting up Passport.js

- We need to find out whether the user with a particular username and password exists or not.
- If the user having a particular username and password exists, then we are required to set that user in the cookie.
- Passport.js uses a session cookie { session cookie stores all the session information plus it is encrypted }.
- In the config folder create a new file { **passport-local-strategy** }.
- Require passport in the { **passport-local-strategy** } file.
- Require passport-local-strategy inside the same file.
- We need to tell the app to use the passport session.
- Passport also helps in maintaining the session.
- Inside the file { **users_controller.js** } we need to redirect the page after the session is created successfully in the passport.

{ **users_controller.js** }

```
// sign in and create a session for the user
module.exports.createSession = function(req, res){
  return res.redirect('/');
}
```

- Inside the routes folder in { **user.js** } file we have to { import passport } .
- When we need to create a session, we are also required to create a route for that session.

{ user.js }

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

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

router.get('/profile', usersController.profile);

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);

module.exports = router;
```

{ passport-local-strategy }

```
const passport = require('passport');
const LocalStrategy = require('passport-local').Strategy;
const User = require('../models/user');
// authentication using passport
passport.use(new LocalStrategy({
  usernameField: 'email'
},
function(email, password, done){
  // find a user and establish the identity
  User.findOne({email: email}, function(err, user) {
    if (err){
      console.log('Error in finding user --> Passport');
      return done(err);
    }
    if (!user || user.password !== password){
      console.log('Invalid Username/Password');
      return done(null, false);
    }
    return done(null, user);
  });
}));
// serializing the user to decide which key is to be kept in the cookies
passport.serializeUser(function(user, done){
  done(null, user.id);
});
// deserializing the user from the key in the cookies
passport.deserializeUser(function(id, done){
  User.findById(id, function(err, user){
    if(err){
      console.log('Error in finding user --> Passport');
      return done(err);
    }
    return done(null, user);
  });
});
module.exports = passport;
```

EXTRA: You can look at these requests from the link below -

<https://www.npmjs.com/package/express-session>

Express sessions and using passport for authentication

- Session Encrypted cookies - Automatically the user's id will be encrypted and stored into session cookies, which can be done using a library { **express-session** }.
- Install the library using the command { **npm install express-session** } in the terminal.
- We have to require this library below mongoose inside the { **index.js** } file.
- We need to require passport and passport - local - strategy libraries inside the { **index.js** } file.
- We need to add middleware that takes the session cookies and encrypts them, below the place where we had set up the view.

```
// used for session cookie
const session = require('express-session');
const passport = require('passport');
const passportLocal = require('./config/passport-local-strategy');
app.use(express.urlencoded());
app.use(cookieParser());
app.use(express.static('./assets'));
app.use(expressLayouts);
// extract style and scripts from sub pages into the layout
app.set('layout extractStyles', true);
app.set('layout extractScripts', true);
// set up the view engine
app.set('view engine', 'ejs');
app.set('views', './views');
app.use(session({
  name: 'codeial',
  // TODO change the secret before deployment in production mode
  secret: 'blahsomething',
  saveUninitialized: false,
  resave: false,
  cookie: {
    maxAge: (1000 * 60 * 100)
  }
}));

app.use(passport.initialize());
app.use(passport.session());
```

Setting Current Authenticated User

Let us see the parameters that are important

- We have set up the passport authentication { The user is now getting an identity established on the server and that identity is saved in a session cookie using an express session that is then communicating from the browser to the server }.
- saveUninitialized: whenever there is a request that is not initialized { when the user has not logged in }, we don't need to store extra data in the session cookies.
- Resave: When the identity is established we don't have to rewrite or save the data if it is not changed.
- We will be sending the data from the server about the user to the ejs files.

- We have to first check if the user is authenticated or not using the `passport.checkAuthenticated` function that we will create using three parameters `req`, `res`, and `next`. This function will internally use the `isAuthenticated` function that is present in `passport.js`.
- If the user is signed in, pass on the request to the next function { controller's action }. If the user is not signed in, then redirect the user to the sign-in page
- Once the user is signed in, set the users for the views using the `setAuthenticatedUser` function that will again take three parameters `req`, `res`, and `next`. This function will internally use the `isAuthenticated` function that is present in `passport.js` in which the `req. user` contains the currently signed-in user from the session cookie.
- Inside the routes folder in the { **user.js** }, we have to use all the functionalities that we have implemented.

Passing User data to views and restricting page access

- We need to restrict the accessibility of sign-in and sign-up pages only when the user is signed out.

```
{ users_controllers.js }
```

```
// render the sign up page
module.exports.signUp = function(req, res){
  return res.render('user_sign_up', {
    title: "Codeial | Sign Up"
  })
}
```

```
// render the sign in page
module.exports.signIn = function(req, res){
  return res.render('user_sign_in', {
    title: "Codeial | Sign In"
  })
}
```

- We have to access the data of the user on the profile page first that is inside the views folder in the { **user_profile** }.

```
{ user_profile }
```

```
<link rel="stylesheet" href="/css/user_profile.css">
<h1>
  Codeial / Profile Page
</h1>

<p> <%= user.name %> </p>
<p> <%= user.email %> </p>
```

Setting up Mongo store for session cookies

- The session cookies get reset every time the server restarts.
- We need some persistent storage to keep the cookies on the server. Hence if we store it inside MongoDB, we shall retain the data.

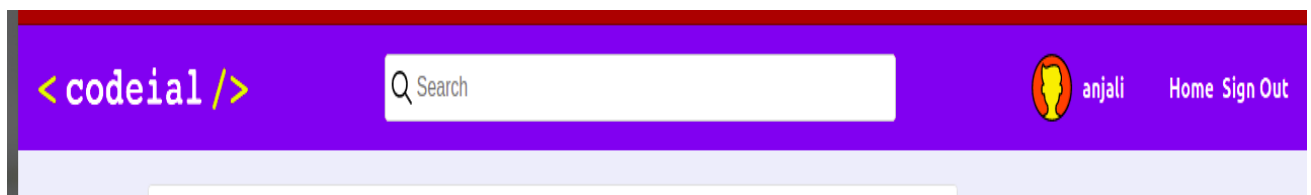
- We will be using Mongo Store for persistent storage and a library called connect-mongo.
- Install the library using the command { npm install connect-mongo } in the terminal.
- We have to require the mongo store library inside the { index.js } file the important parameters at the important parameters require one argument that is session.
- We need to define another key in the app. use a method that is a store.

Creating Sign Out

We will create an action for signing out and see how signing out works with Passport.js

- In the views folder inside the { _header. ejs } create a list of authentication items.
- If the user is signed in, then we need to show the user's name and a link to sign out.
- If the user is not signed in, then we need to show the user the sign-in link as well as the sign-up link.
- **Pages of different links are shown below**

{ Sign-out }



{Sign-in}

`<codeial />`


Q Search

Home Signup Login

Log In

[Forgot Password ?](#)

Log In

 Sign in with google

{ Sign-up }


`<codeial />`

Q Search

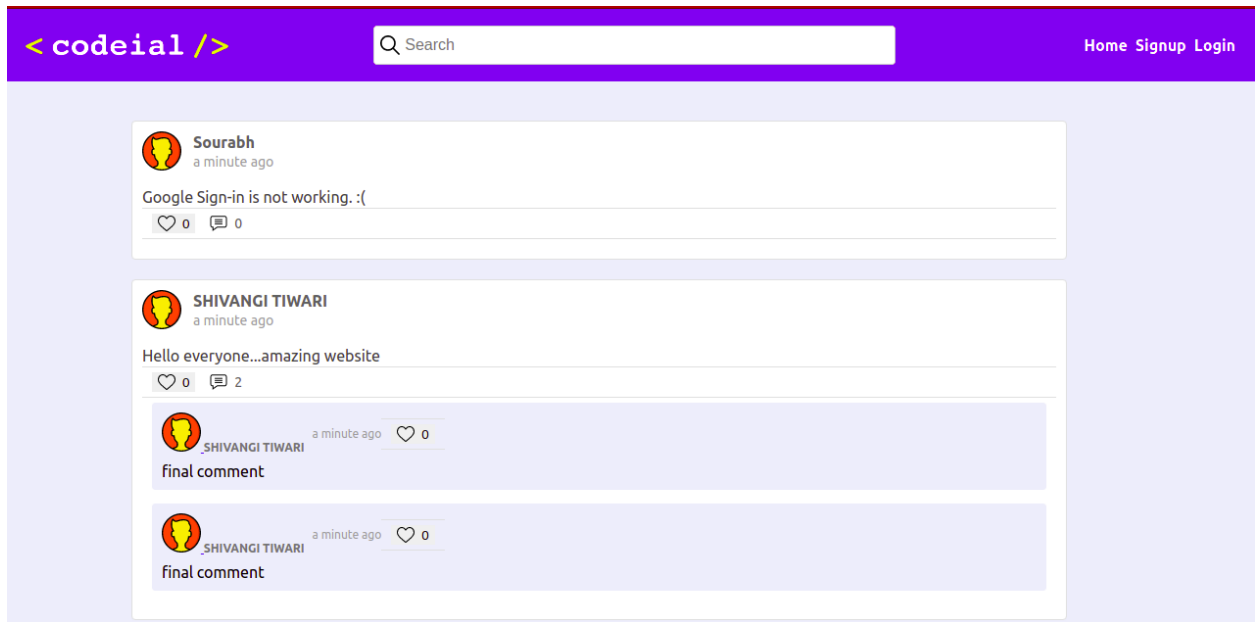
Home Signup Login

Sign Up

Sign Up

 Sign in with google

{ home page }



{ _header.ejs. }

```
<header>
  Page Header
  <ul>
    <% if (locals.user){ %>
      <li>
        <a href="/users/profile">
          <%= user.name %>
        </a>
      </li>
      <li>
        <a href="/users/sign-out">Sign Out</a>
      </li>
    <% }else{ %>
      <li>
        <a href="/users/sign-in">Sign In</a>
      </li>
      <li>
        <a href="/users/sign-up">Sign Up</a>
      </li>
    <% } %>
  </ul>
</header>
```

- We have to create an action inside the controllers folder in the { **users_controller.js** } file.

```
{ users_controller.js }
```

```
// sign in and create a session for the user
module.exports.createSession = function(req, res){
  return res.redirect('/');
}

module.exports.destroySession = function(req, res){
  req.logout();

  return res.redirect('/');
}
```

- We have to create a route inside the route folder in the { **users.js** } file.

```
{ users.js }
```

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

module.exports = router;
```

Summarizing

- We installed the passport library and local strategy package and required them.
- We created a new strategy allocated to a middleware wherein we declared the username field to be an email.
- If the user is found we returned using the call back done with error null and the user.

- If the user is not found or the password doesn't match we return.
- If there is an error we return the error.
- The done function is the callback function.
- Passport. serialize and passport. deserialize is used to set id as a cookie in the user's browser and to get the id from the cookie when it is then used to get user info in a callback.
- We have serialized the user that picks out the information from the user which needs to be fetched in the session cookie.
- We have deserialized the user that picks the id from the session cookie and converting it into a user by finding it in the database, for that we have imported the user from the models.
- We have a **check authentication** method that checks whether the user is authenticated or not.
 - If the user is authenticated it will return to the next function that is to be called.
 - If the user is not authenticated then take the user back to the sign-in page.
- We need to access the authenticated users in the views. For that, we use the **Set Authenticated** user method.
- Whenever a passport is initialized or a passport session is being used, an authenticated user is also being set.
- The session is maintained by the **express-session** library
- We just use the session, keep the name of the session, use the secret key to encrypt the data that is present.
- Mongo Store contains the express session that is used to store the session information even when the server restarts it remains in the database so that the signed-in users don't get reset in case the server restarts.
- In the routes folder inside the { **users.js** } file, we have created a session using the passport. authenticate wherein the local authentication is used.

- In case of failure, it is redirected back to the sign-in page.
- In case of success **usersController.createSession** action is being called that is, in turn, taking the user to the home page.
- We wanted the profile page to be accessible only when the user is signed in so we put a check of **passport.checkAuthentication** { It checks whether the user is authenticated and then passes the control to the next function. If the user is not authenticated then it sends to the sign-in page.
- We have created a header so that the website is usable whenever the user is logged in or logged out a different set of things are displayed on the top corner.
- If locals. user is there then user name and link to user's profile along with sign-out button is displayed.
- In case the user is not signed in then sign in and the sign-up link is present.