REVERSE ENGINEERING

Tutorial for Node JS application using Sequelize and Passport,

Walkthrough to understand the codebase by Yvette Waller

Table of Contents

OVERVIEW & PURPOSE

This tutorial serves as a ‘walk-through’ for developers to familiarise themselves with a new codebase. This codebase can then be used to start a new project.

Intention of this Project (What does this codebase do)

This codebase is for password authentication. It allows a user to create an account, log into an account and sign back out securely - all on website files. All user data is stored in a MySQL Database.

SAMPLE USER STORY

As someone who wants to safely log in to "X" site, I want to know my personal details are safely stored in a database so that I don't have to worry about the security of using "X" site.

FILE DIRECTORY MAP

Below is a directory of the file structure with a

- *(+) for new files/folders added in the initialisation phases.*
- *Highlighted files require editing in order to make the code work.*

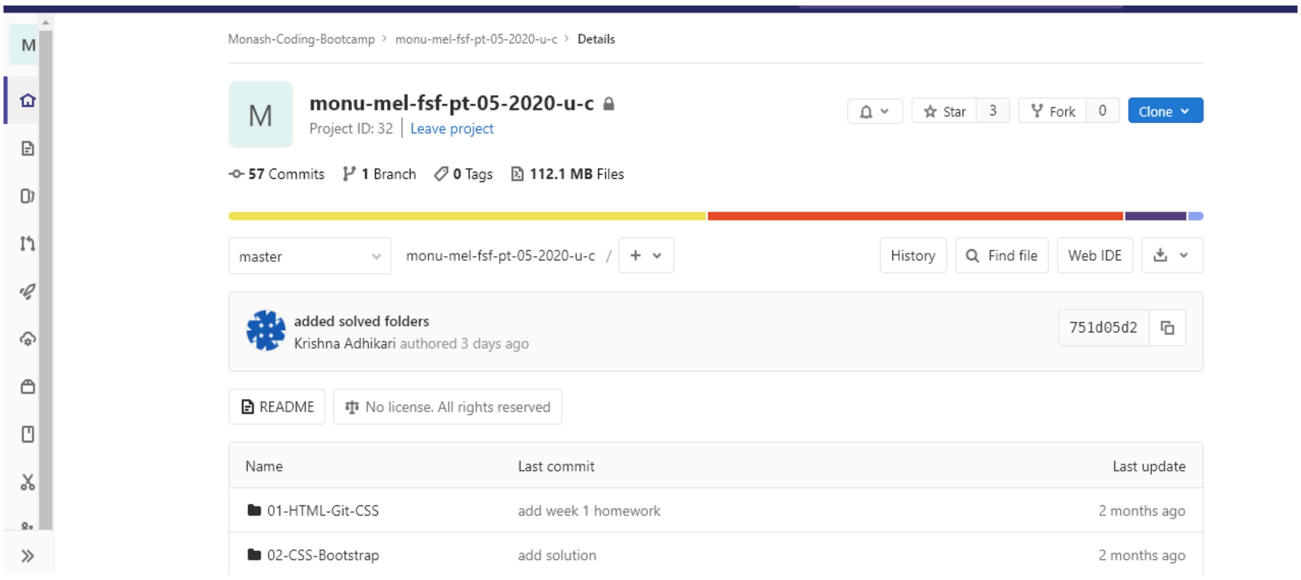
```
🗀Develop
├ 🗁config
|  ├ 🗁middleware
|  |  └ 🖹isAuthenticated.js
|  ├ 🖹config.json
|  └ 🖹passport.js
├ 🗁models
|  ├ 🖹index.js
|  └ 🖹user.js
├ 🗁node_modules (+) (note: sub folders are not  added in
entirety in this tree as it would be too large to display. See the
installed package.json file for required modules on
initialization which will then appear as subfolders in
node_modules folder once installed.
|  ├ 🗁.bin
├ 🗁public
|  ├ 🗁js
|  |  ├ 🖹login.js
|  |  ├ 🖹members.js
|  |  └ 🖹signup.js
|  ├ 🗁stylesheets
|  |  └ 🖹style.css
|  ├ 🖹login.html
|  ├ 🖹members.html
|  └ 🖹signup.html
├ 🗁routes
|  ├ 🖹api-routes.js
|  └ 🖹html-routes.js
├ 🖹database_production.sql  (+)
├ 🖹database_test.sql  (+)
├ 🖹package-lock.json
├ 🖹package.json
├ 🖹passport_demo.sql (+)
├ 🖹server.js
```

WALKTHROUGH OF CODE

Quick Instructions

1. Begin using this codebase by first cloning the repository into your local storage.

Example below, Gitlab repo used:



Monash-Coding-Bootcamp > monu-mel-fsf-pt-05-2020-u-c > Details

monu-mel-fsf-pt-05-2020-u-c
Project ID: 32 | Leave project

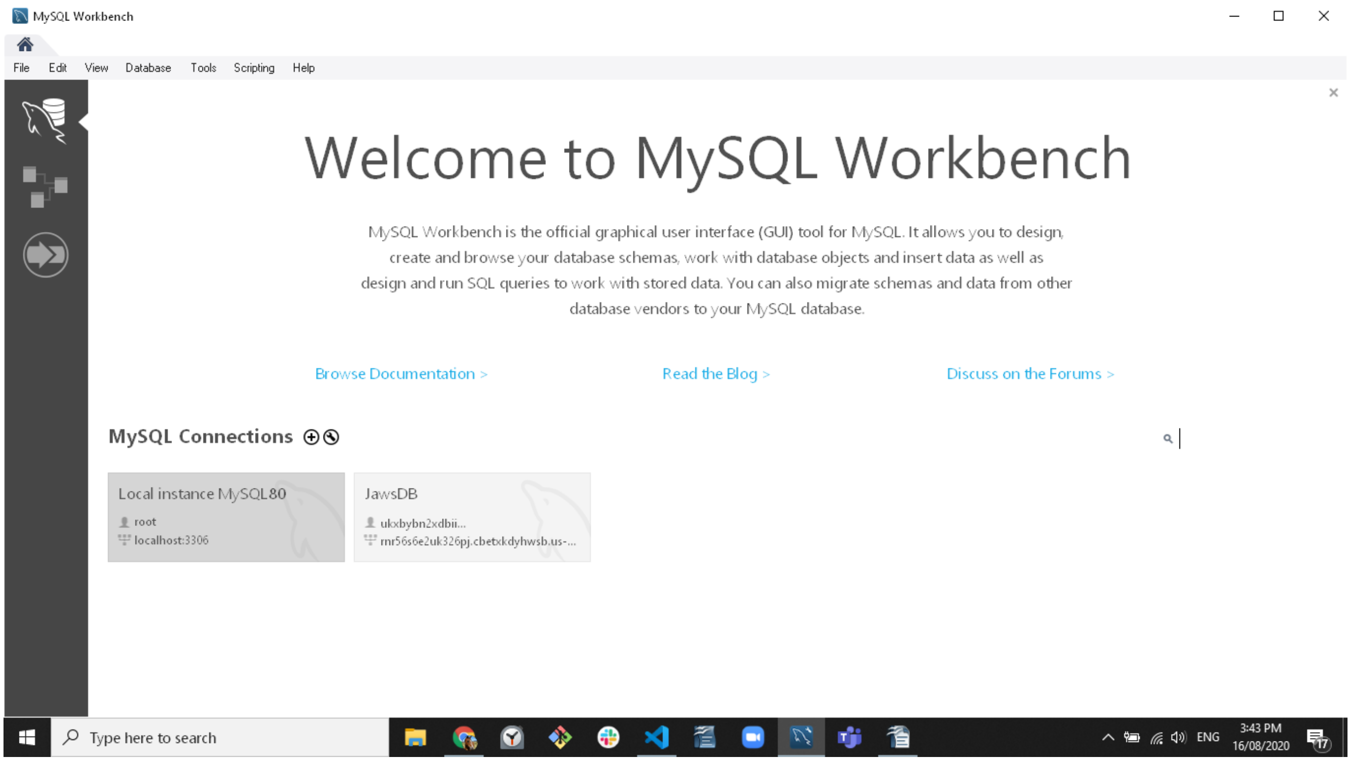
57 Commits · 1 Branch · 0 Tags · 112.1 MB Files

added solved folders
Krishna Adhikari authored 3 days ago · 751d05d2

README · No license. All rights reserved

Name	Last commit	Last update
01-HTML-Git-CSS	add week 1 homework	2 months ago
02-CSS-Bootstrap	add solution	2 months ago

2. Once complete - change all const/lets in code to var to make the code run
3. Create SQL Scripts for 3 x databases as named in the config.json file
4. Open and run scripts in MySQL to create the 3 databases



Welcome to MySQL Workbench

MySQL Workbench is the official graphical user interface (GUI) tool for MySQL. It allows you to design, create and browse your database schemas, work with database objects and insert data as well as design and run SQL queries to work with stored data. You can also migrate schemas and data from other database vendors to your MySQL database.

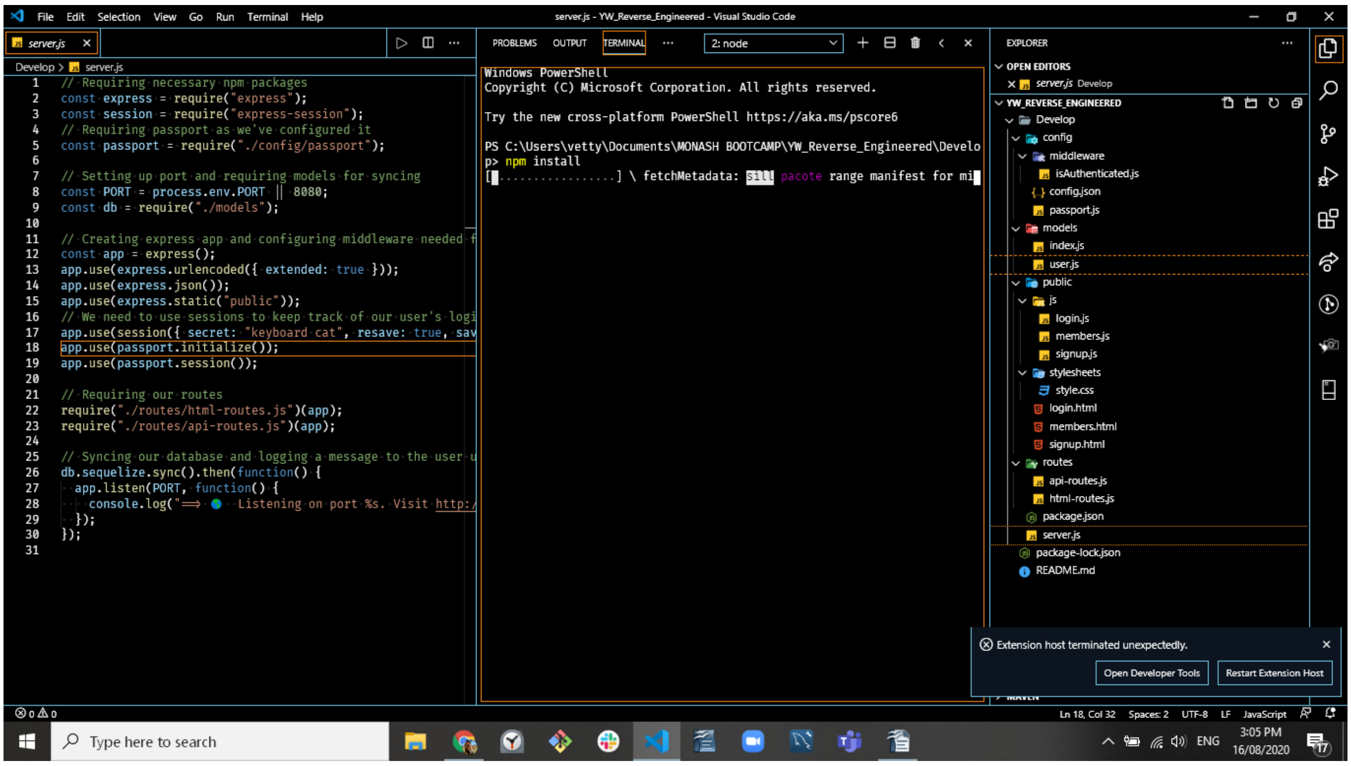
Browse Documentation > Read the Blog > Discuss on the Forums >

MySQL Connections

Local instance MySQL80
root
localhost:3306

JawsDB
ukxbybn2xdbii...
mr56s6e2uk326pj.cbetxkdyhwsb.us-...

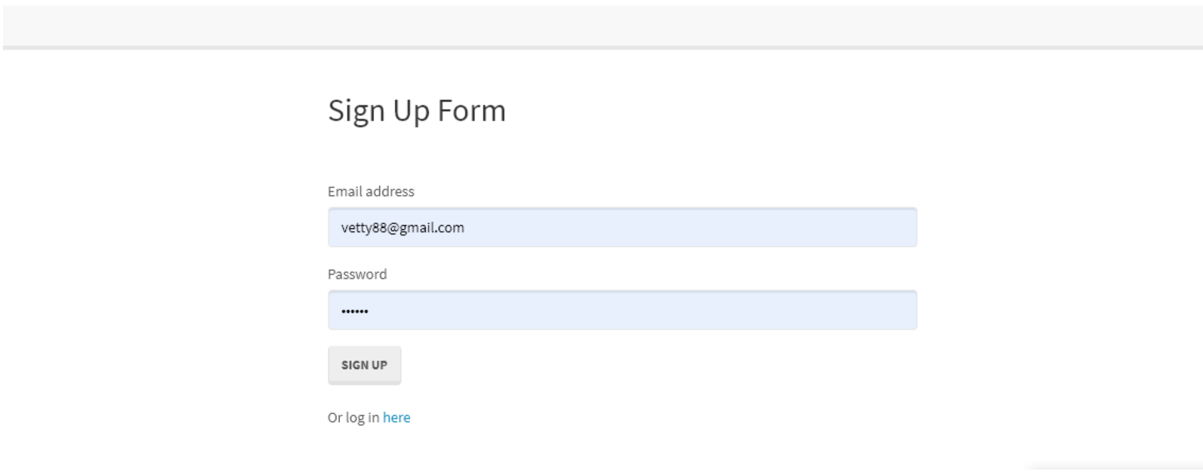
5. Edit the config.js file and include your own personal data (i.e: password for mySQL).
6. Open integrated terminal and run "NPM install" to install the required node modules for this project.



```
// Requiring necessary npm packages
const express = require("express");
const session = require("express-session");
// Requiring passport as we've configured it
const passport = require("./config/passport");

// Setting up port and requiring models for syncing
const PORT = process.env.PORT || 8080;
const db = require("./models");

// Creating express app and configuring middleware needed f
const app = express();
app.use(express.urlencoded({ extended: true }));
app.use(express.json());
app.use(express.static("public"));
// We need to use sessions to keep track of our user's logi
app.use(session({ secret: "keyboard cat", resave: true, sav
app.use(passport.initialize());
app.use(passport.session());

// Requiring our routes
require("./routes/html-routes.js")(app);
require("./routes/api-routes.js")(app);

// Syncing our database and logging a message to the user u
db.sequelize.sync().then(function() {
  app.listen(PORT, function() {
    console.log("==> 🌎  Listening on port %s. Visit http:/
  });
});
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\vetty\Documents\MONASH BOOTCAMP\YW_Reverse_Engineered\Develo
p> npm install
[█.................] \ fetchMetadata: sill pacote range manifest for mi
```

Extension host terminated unexpectedly.

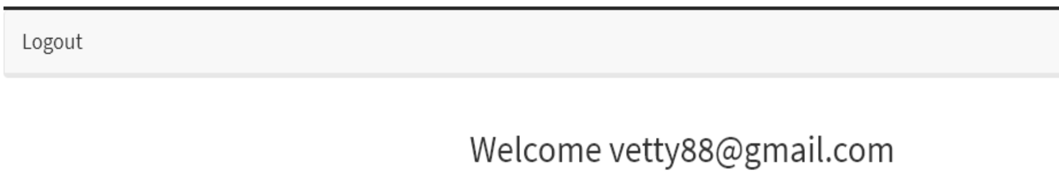
7. Run NPM audit fix if any issues
8. Run Node Server.JS to check server is working
9. Test html features (3 sites) by opening the local server link in your web browser.
    a. 
    Sign Up Form

    Email address

    vetty88@gmail.com

    Password

    ••••••

    b. 

    Welcome vetty88@gmail.com
10. Check the mySQL database to ensure login data is saved.
11. Celebrate!

File Walk Through with Steps

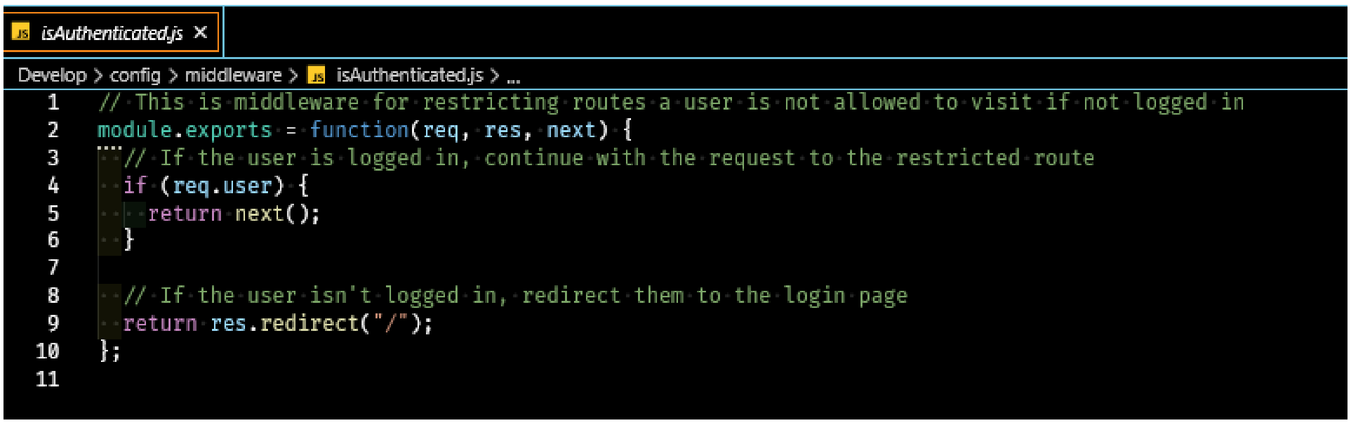
Develop

├ config

│ ├ middleware

│ │ └ isAuthenticated.js

This middleware file restricts routes that the user is not allowed to visit if they are not logged in. For example: the user cannot reach the members page if not logged in. If the user is logged in, it continues with request. This is the security feature of this passport codebase.

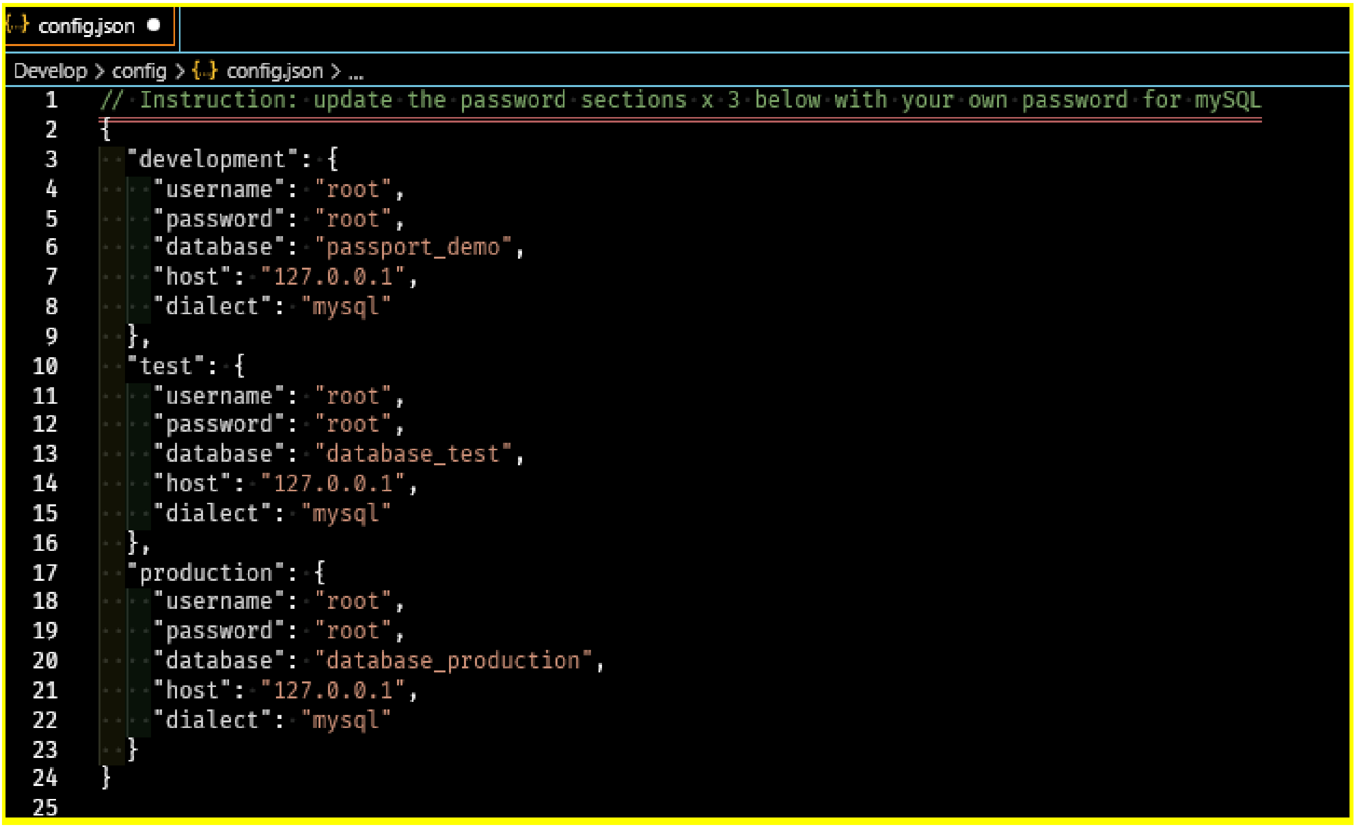

isAuthenticated.js ×

Develop > config > middleware > isAuthenticated.js > ...

```javascript
// This is middleware for restricting routes a user is not allowed to visit if not logged in
module.exports = function(req, res, next) {
  // If the user is logged in, continue with the request to the restricted route
  if (req.user) {
    return next();
  }

  // If the user isn't logged in, redirect them to the login page
  return res.redirect("/");
};

```

│ ├ config.json

middleware connection configuration to connect to server.

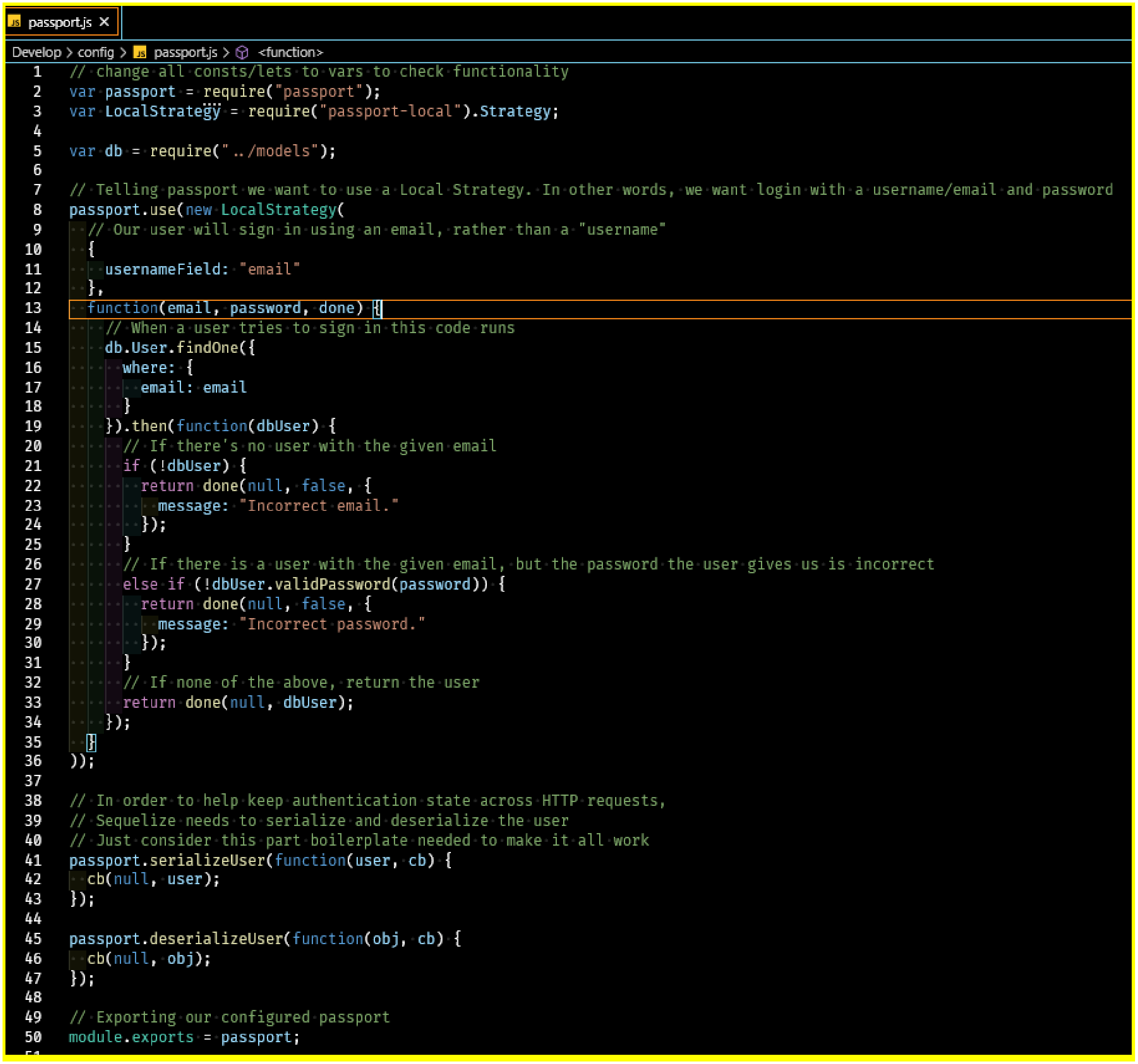

config.json ●

Develop > config > config.json > ...

```json
// Instruction: update the password sections x 3 below with your own password for mySQL
{
  "development": {
    "username": "root",
    "password": "root",
    "database": "passport_demo",
    "host": "127.0.0.1",
    "dialect": "mysql"
  },
  "test": {
    "username": "root",
    "password": "root",
    "database": "database_test",
    "host": "127.0.0.1",
    "dialect": "mysql"
  },
  "production": {
    "username": "root",
    "password": "root",
    "database": "database_production",
    "host": "127.0.0.1",
    "dialect": "mysql"
  }
}

```

│ └ 🗏*passport.js*

Middleware contains javascript logic that tells passport we want to log in with an email address and password.

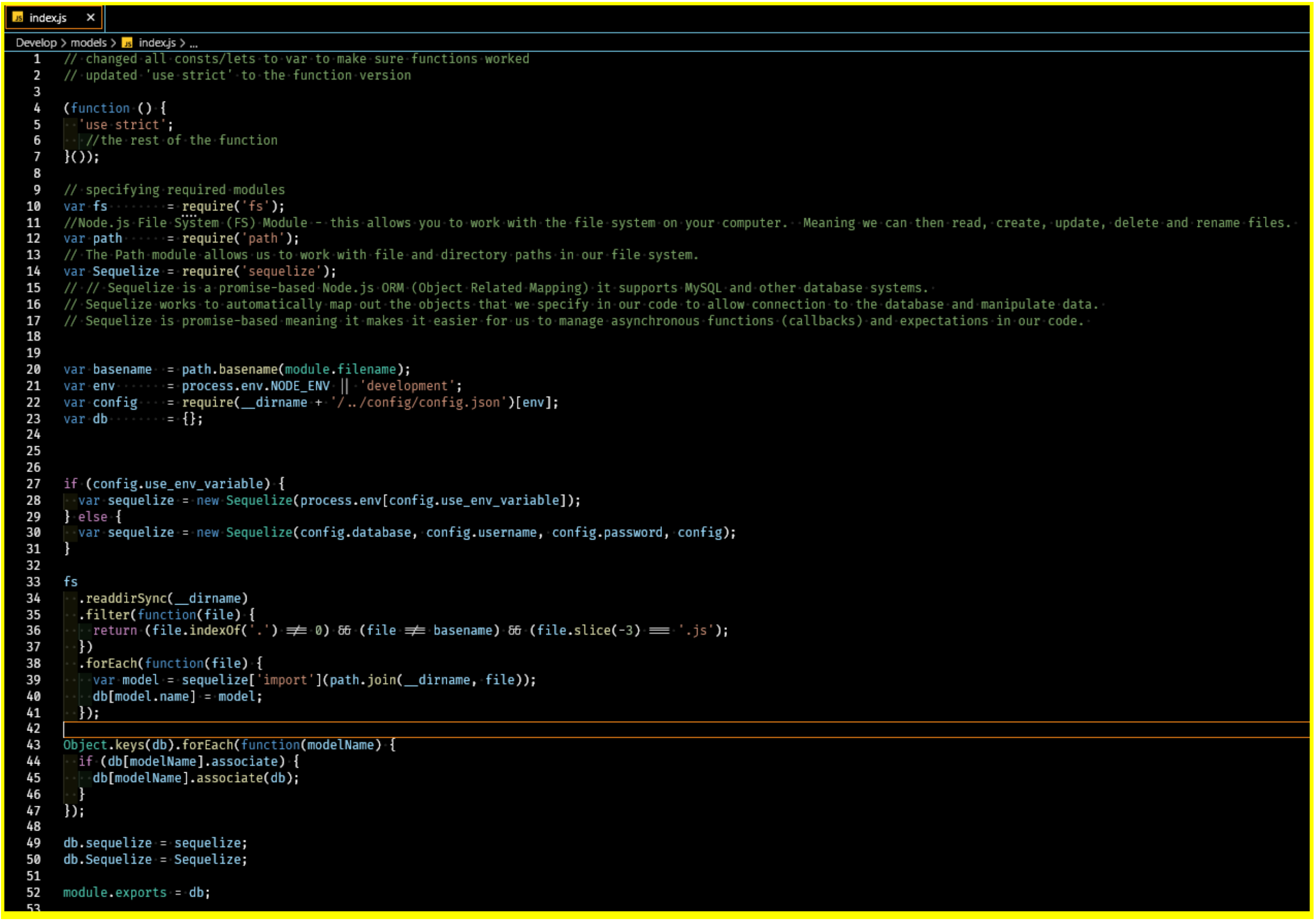


```javascript
// change all consts/lets to vars to check functionality
var passport = require("passport");
var LocalStrategy = require("passport-local").Strategy;

var db = require("../models");

// Telling passport we want to use a Local Strategy. In other words, we want login with a username/email and password
passport.use(new LocalStrategy(
  // Our user will sign in using an email, rather than a "username"
  {
    usernameField: "email"
  },
  function(email, password, done) {
    // When a user tries to sign in this code runs
    db.User.findOne({
      where: {
        email: email
      }
    }).then(function(dbUser) {
      // If there's no user with the given email
      if (!dbUser) {
        return done(null, false, {
          message: "Incorrect email."
        });
      }
      // If there is a user with the given email, but the password the user gives us is incorrect
      else if (!dbUser.validPassword(password)) {
        return done(null, false, {
          message: "Incorrect password."
        });
      }
      // If none of the above, return the user
      return done(null, dbUser);
    });
  }
));

// In order to help keep authentication state across HTTP requests,
// Sequelize needs to serialize and deserialize the user
// Just consider this part boilerplate needed to make it all work
passport.serializeUser(function(user, cb) {
  cb(null, user);
});

passport.deserializeUser(function(obj, cb) {
  cb(null, obj);
});

// Exporting our configured passport
module.exports = passport;
```

├ 🗁*models*

│ ├ 🗏*index.js*

This model connects to the database and imports each users log-in data.



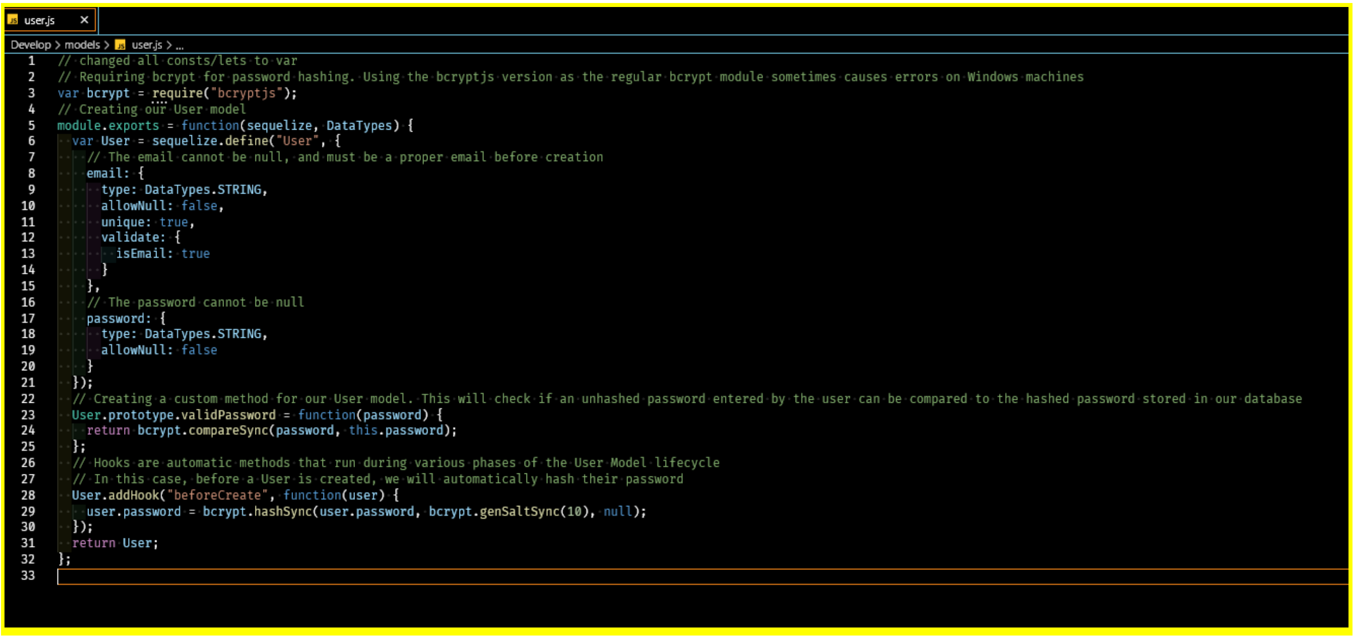
index.js

Develop > models > index.js > ...

```
// changed all consts/lets to var to make sure functions worked
// updated 'use strict' to the function version

(function () {
  'use strict';
  //the rest of the function
}());

// specifying required modules
var fs        = require('fs');
//Node.js File System (FS) Module - this allows you to work with the file system on your computer.  Meaning we can then read, create, update, delete and rename files.
var path      = require('path');
// The Path module allows us to work with file and directory paths in our file system.
var Sequelize = require('sequelize');
// // Sequelize is a promise-based Node.js ORM (Object Related Mapping) it supports MySQL and other database systems.
// Sequelize works to automatically map out the objects that we specify in our code to allow connection to the database and manipulate data.
// Sequelize is promise-based meaning it makes it easier for us to manage asynchronous functions (callbacks) and expectations in our code.


var basename  = path.basename(module.filename);
var env       = process.env.NODE_ENV || 'development';
var config    = require(__dirname + '/../config/config.json')[env];
var db        = {};



if (config.use_env_variable) {
  var sequelize = new Sequelize(process.env[config.use_env_variable]);
} else {
  var sequelize = new Sequelize(config.database, config.username, config.password, config);
}

fs
  .readdirSync(__dirname)
  .filter(function(file) {
    return (file.indexOf('.') !== 0) && (file !== basename) && (file.slice(-3) === '.js');
  })
  .forEach(function(file) {
    var model = sequelize['import'](path.join(__dirname, file));
    db[model.name] = model;
  });

Object.keys(db).forEach(function(modelName) {
  if (db[modelName].associate) {
    db[modelName].associate(db);
  }
});

db.sequelize = sequelize;
db.Sequelize = Sequelize;

module.exports = db;
```

/ └ 🗏user.js

Requires "bcrypt" module for password hashing. This feature makes our database secure even if compromised. This javascript defines what is in our database.



user.js

Develop > models > user.js > ...

```
// changed all consts/lets to var
// Requiring bcrypt for password hashing. Using the bcryptjs version as the regular bcrypt module sometimes causes errors on Windows machines
var bcrypt = require("bcryptjs");
// Creating our User model
module.exports = function(sequelize, DataTypes) {
  var User = sequelize.define("User", {
    // The email cannot be null, and must be a proper email before creation
    email: {
      type: DataTypes.STRING,
      allowNull: false,
      unique: true,
      validate: {
        isEmail: true
      }
    },
    // The password cannot be null
    password: {
      type: DataTypes.STRING,
      allowNull: false
    }
  });
  // Creating a custom method for our User model. This will check if an unhashed password entered by the user can be compared to the hashed password stored in our database
  User.prototype.validPassword = function(password) {
    return bcrypt.compareSync(password, this.password);
  };
  // Hooks are automatic methods that run during various phases of the User Model lifecycle
  // In this case, before a User is created, we will automatically hash their password
  User.addHook("beforeCreate", function(user) {
    user.password = bcrypt.hashSync(user.password, bcrypt.genSaltSync(10), null);
  });
  return User;
};
```

├ 🗀node_modules (+) (note: sub folders are not added in entirety in this tree as it would be too

large to display. See the installed package.json file for required modules on initialization which will then appear as subfolders in node_modules folder once installed. Run NPM install in the console to add the node modules required, any issues - refer to the error code.

Key Node Modules used are bcryptjs, express, express-session, mysql2, passport, passport-local and sequelize.

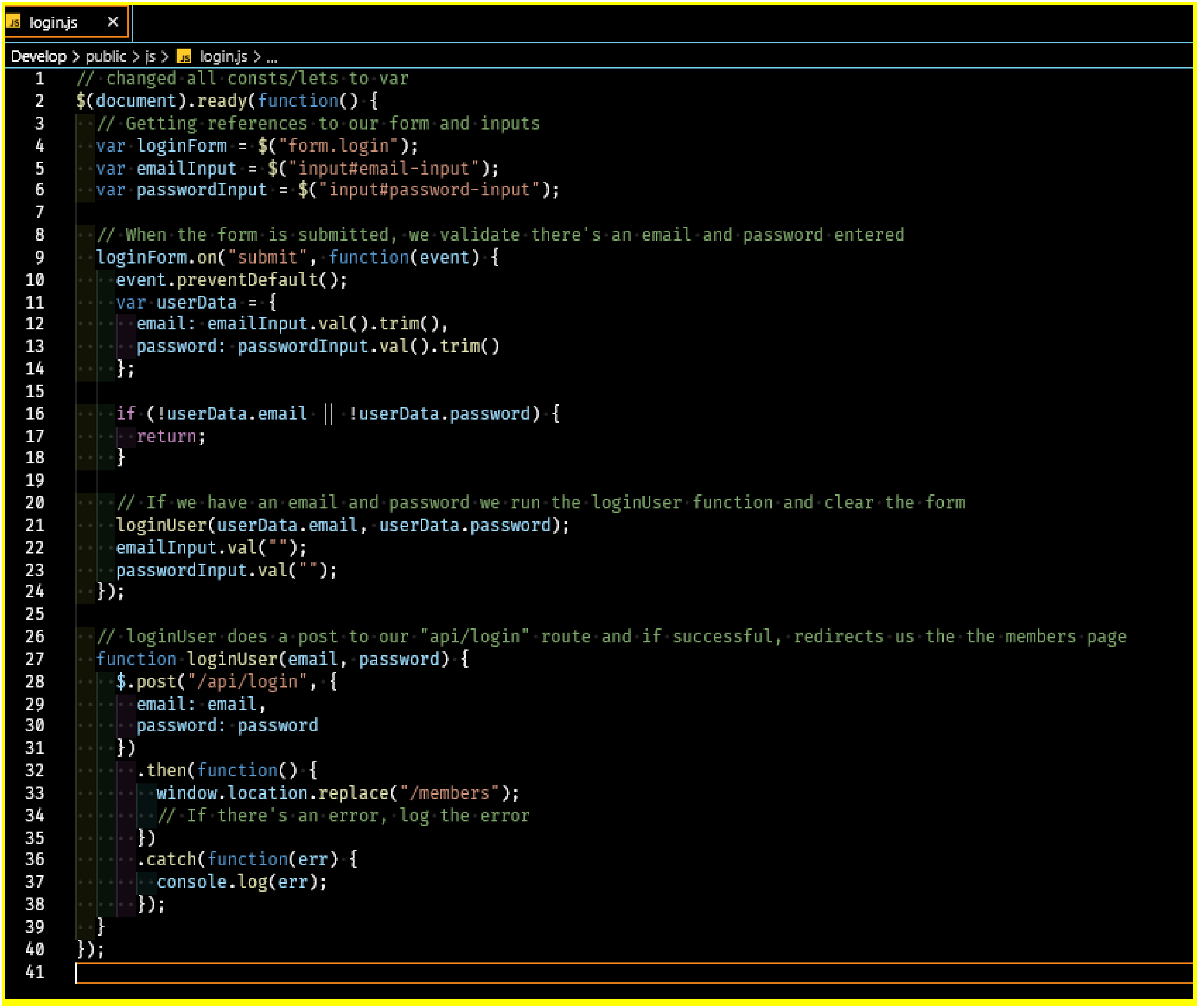
/ ├ 🗁.bin

This folder stores the executable files so that the node module functions work when on the local server.

├ 🗁public

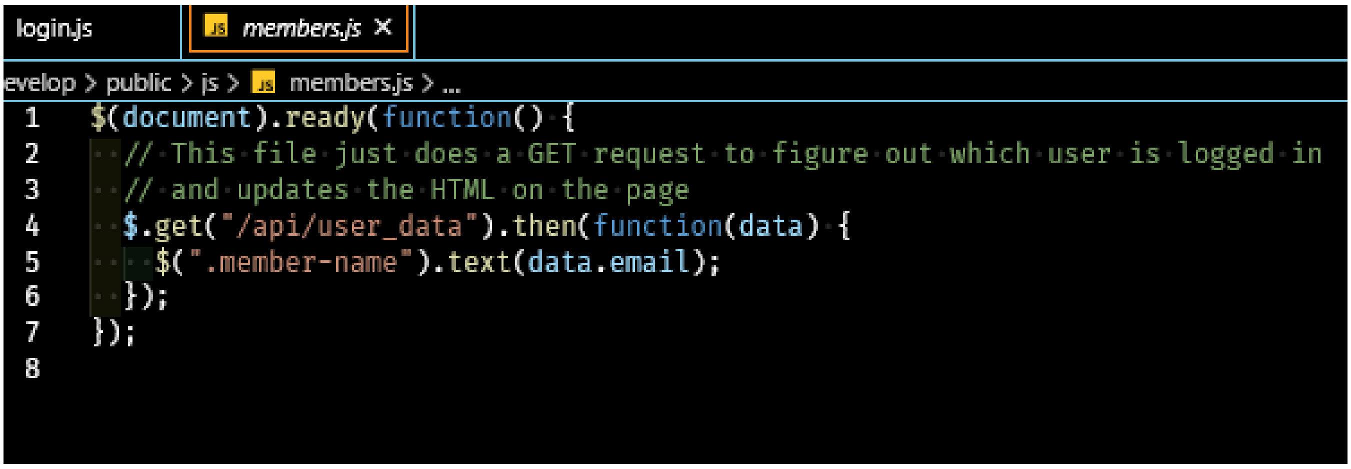
/ ├ 🗁js

/ / ├ 🗏login.js



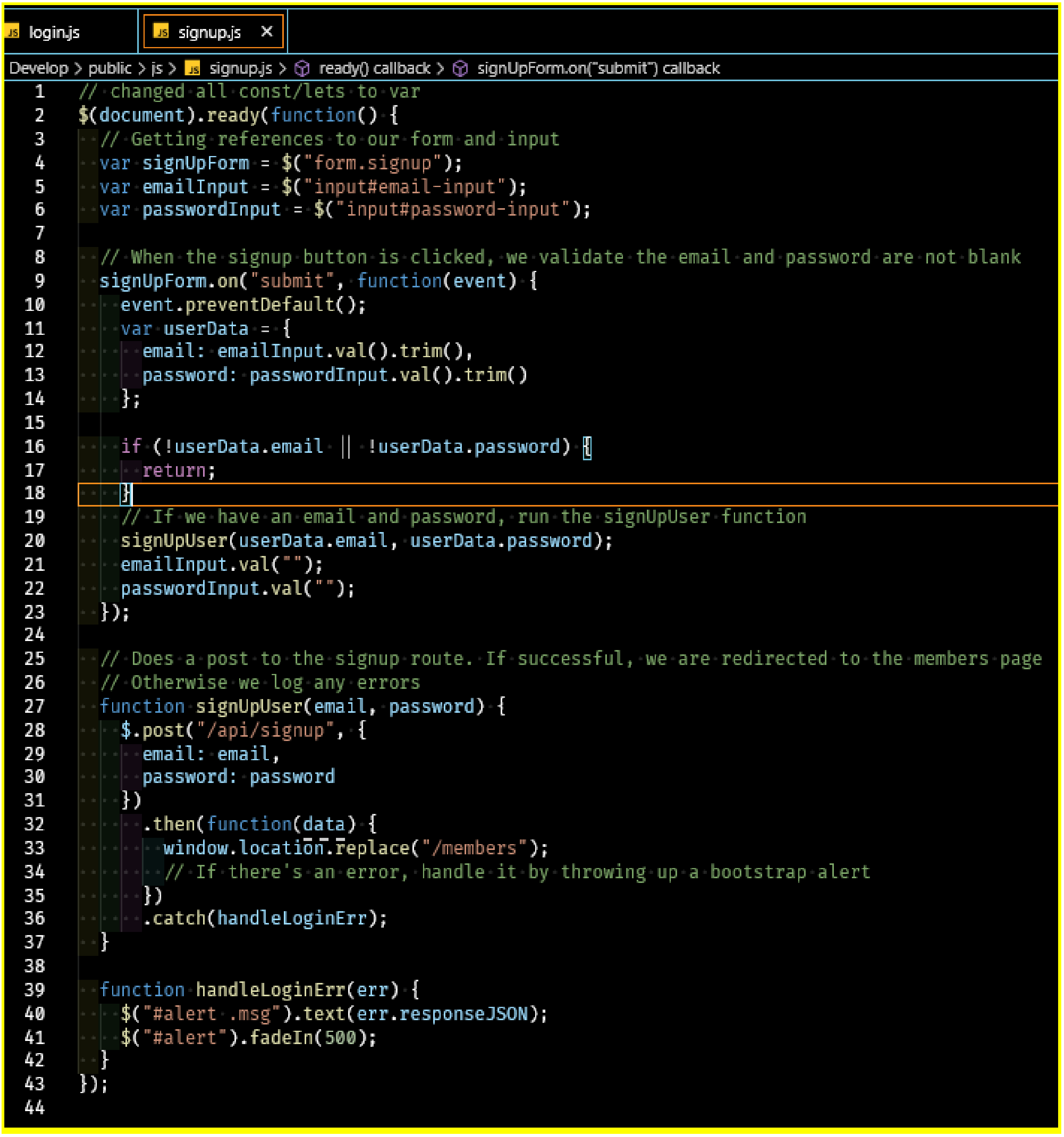
```
// changed all consts/lets to var
$(document).ready(function() {
  // Getting references to our form and inputs
  var loginForm = $("form.login");
  var emailInput = $("input#email-input");
  var passwordInput = $("input#password-input");

  // When the form is submitted, we validate there's an email and password entered
  loginForm.on("submit", function(event) {
    event.preventDefault();
    var userData = {
      email: emailInput.val().trim(),
      password: passwordInput.val().trim()
    };

    if (!userData.email || !userData.password) {
      return;
    }

    // If we have an email and password we run the loginUser function and clear the form
    loginUser(userData.email, userData.password);
    emailInput.val("");
    passwordInput.val("");
  });

  // loginUser does a post to our "api/login" route and if successful, redirects us the the members page
  function loginUser(email, password) {
    $.post("/api/login", {
      email: email,
      password: password
    })
      .then(function() {
        window.location.replace("/members");
        // If there's an error, log the error
      })
      .catch(function(err) {
        console.log(err);
      });
  }
});
```

/ / ├ 🗏members.js



```
$(document).ready(function() {
  // This file just does a GET request to figure out which user is logged in
  // and updates the HTML on the page
  $.get("/api/user_data").then(function(data) {
    $(".member-name").text(data.email);
  });
});
```

| | └ 📃signup.js

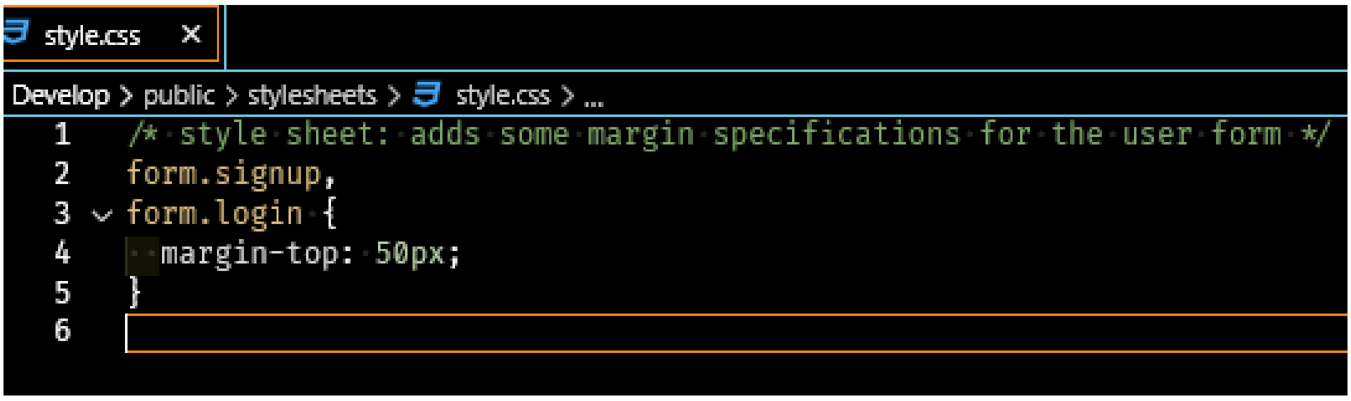

login.js   signup.js ×

Develop > public > js > signup.js > ready() callback > signUpForm.on("submit") callback

```
// changed all const/lets to var
$(document).ready(function() {
  // Getting references to our form and input
  var signUpForm = $("form.signup");
  var emailInput = $("input#email-input");
  var passwordInput = $("input#password-input");

  // When the signup button is clicked, we validate the email and password are not blank
  signUpForm.on("submit", function(event) {
    event.preventDefault();
    var userData = {
      email: emailInput.val().trim(),
      password: passwordInput.val().trim()
    };

    if (!userData.email || !userData.password) {
      return;
    }
    // If we have an email and password, run the signUpUser function
    signUpUser(userData.email, userData.password);
    emailInput.val("");
    passwordInput.val("");
  });

  // Does a post to the signup route. If successful, we are redirected to the members page
  // Otherwise we log any errors
  function signUpUser(email, password) {
    $.post("/api/signup", {
      email: email,
      password: password
    })
      .then(function(data) {
        window.location.replace("/members");
        // If there's an error, handle it by throwing up a bootstrap alert
      })
      .catch(handleLoginErr);
  }

  function handleLoginErr(err) {
    $("#alert .msg").text(err.responseJSON);
    $("#alert").fadeIn(500);
  }
});

```

/   ├ 🗁*stylesheets*

/   /   └ 🖹*style.css*

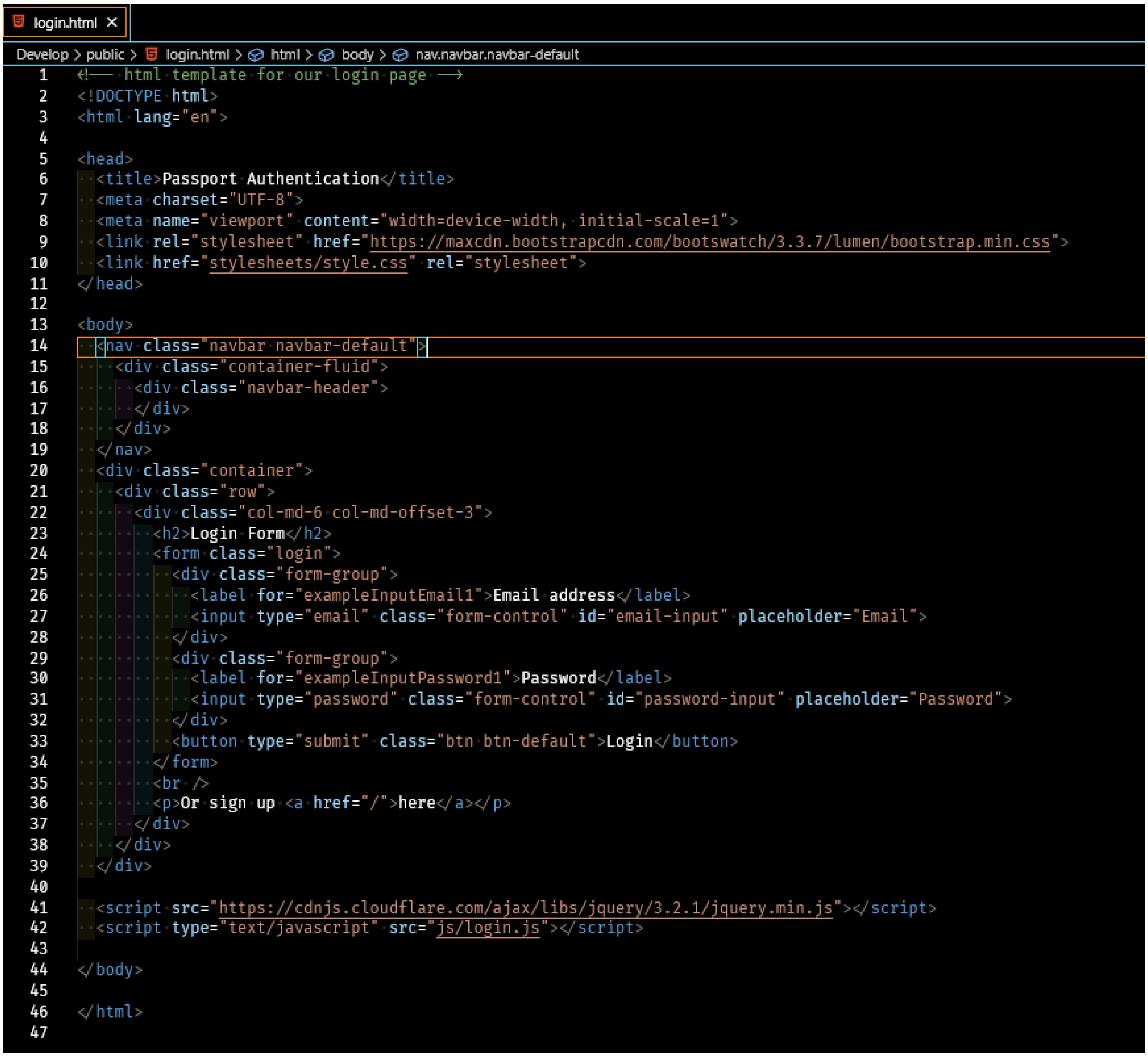


style.css ✕

Develop > public > stylesheets > style.css > ...

```css
/* style sheet: adds some margin specifications for the user form */
form.signup,
form.login {
  margin-top: 50px;
}

```

/ ├ 📃login.html

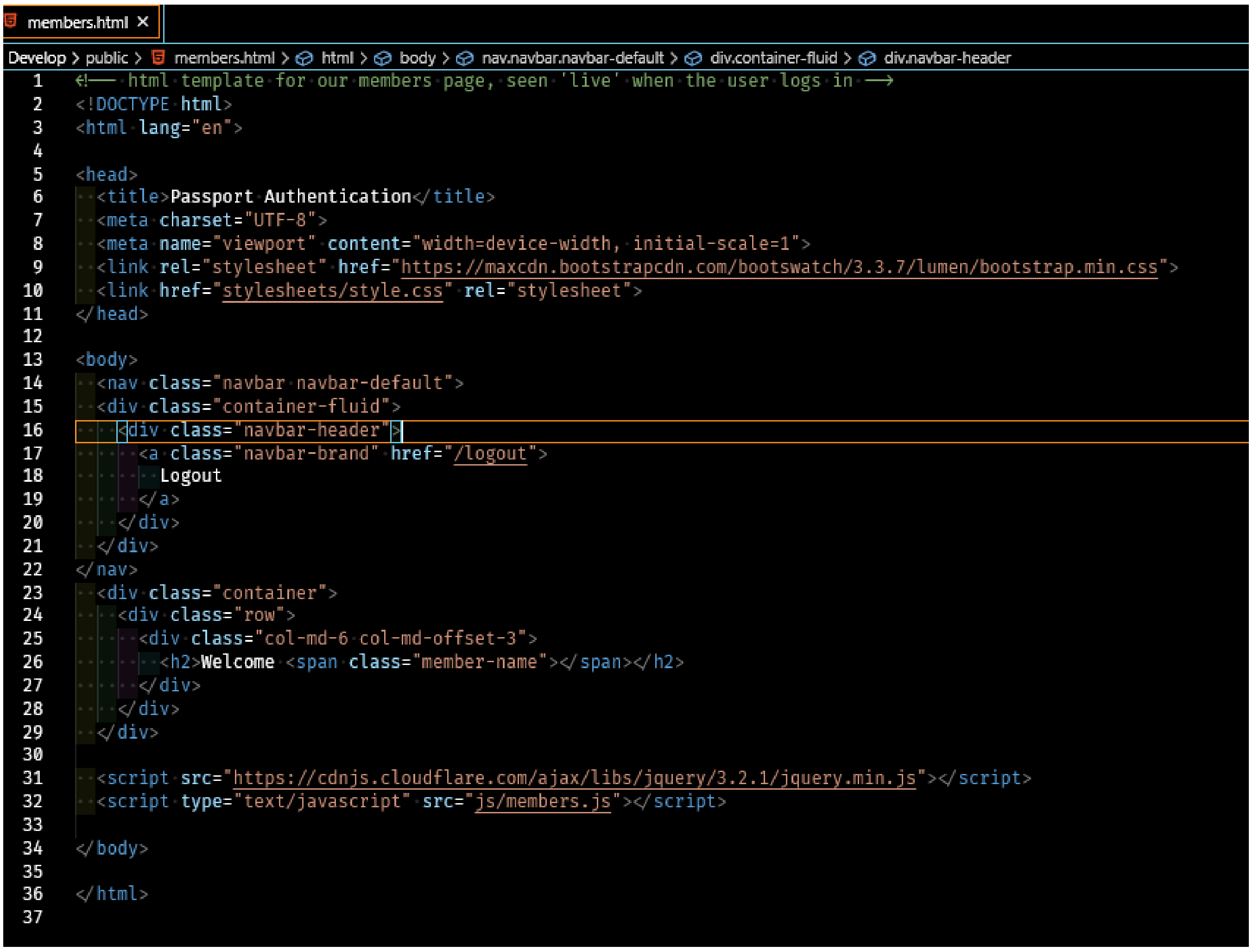


login.html ✕

Develop > public > login.html > html > body > nav.navbar.navbar-default

```html
<!-- html template for our login page -->
<!DOCTYPE html>
<html lang="en">

<head>
  <title>Passport Authentication</title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootswatch/3.3.7/lumen/bootstrap.min.css">
  <link href="stylesheets/style.css" rel="stylesheet">
</head>

<body>
  <nav class="navbar navbar-default">
    <div class="container-fluid">
      <div class="navbar-header">
      </div>
    </div>
  </nav>
  <div class="container">
    <div class="row">
      <div class="col-md-6 col-md-offset-3">
        <h2>Login Form</h2>
        <form class="login">
          <div class="form-group">
            <label for="exampleInputEmail1">Email address</label>
            <input type="email" class="form-control" id="email-input" placeholder="Email">
          </div>
          <div class="form-group">
            <label for="exampleInputPassword1">Password</label>
            <input type="password" class="form-control" id="password-input" placeholder="Password">
          </div>
          <button type="submit" class="btn btn-default">Login</button>
        </form>
        <br />
        <p>Or sign up <a href="/">here</a></p>
      </div>
    </div>
  </div>

  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
  <script type="text/javascript" src="js/login.js"></script>

</body>

</html>

```

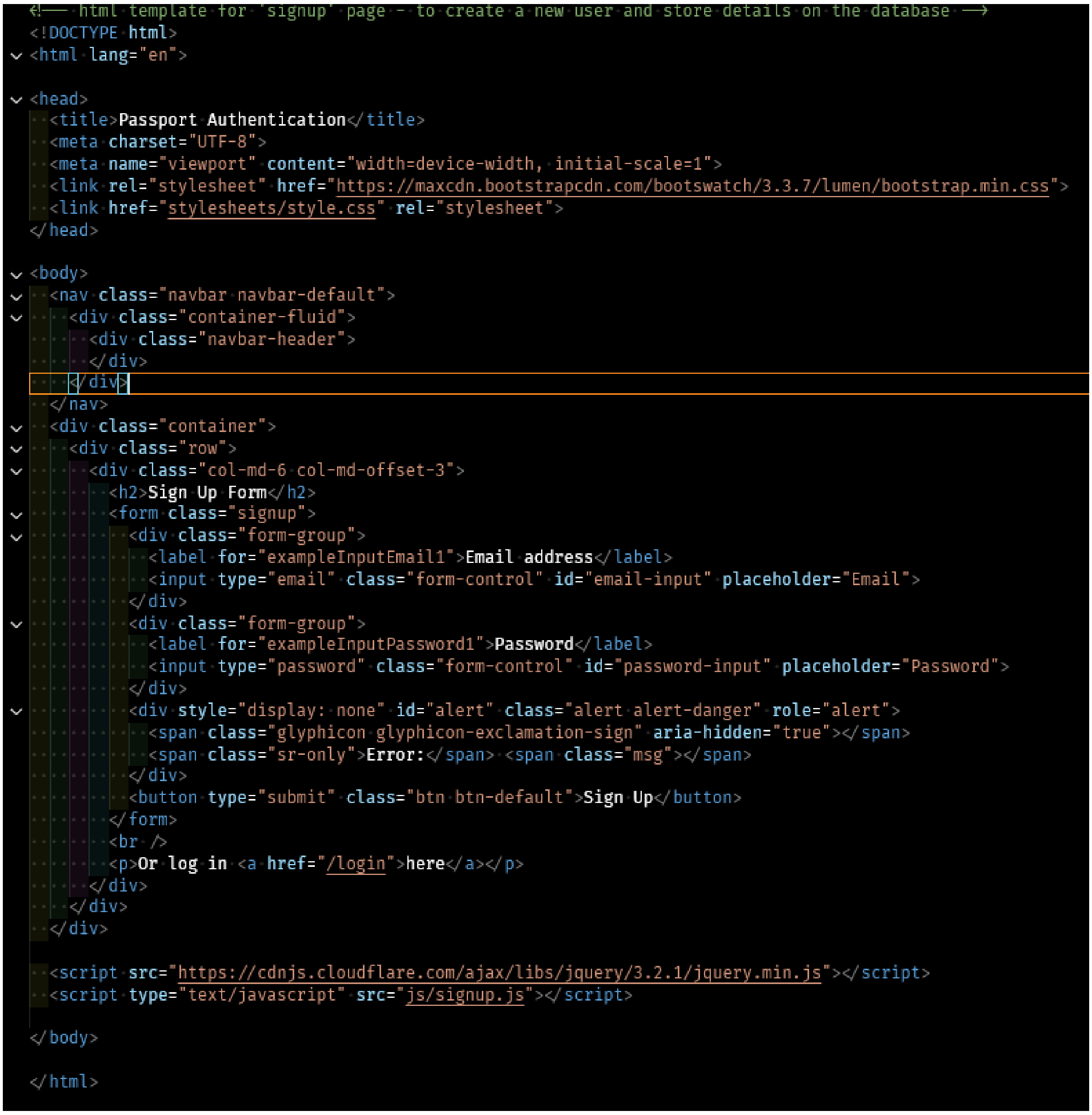
/ ├ 📃members.html



members.html ✕
Develop > public > members.html > html > body > nav.navbar.navbar-default > div.container-fluid > div.navbar-header

```html
<!-- html template for our members page, seen 'live' when the user logs in -->
<!DOCTYPE html>
<html lang="en">

<head>
  <title>Passport Authentication</title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootswatch/3.3.7/lumen/bootstrap.min.css">
  <link href="stylesheets/style.css" rel="stylesheet">
</head>

<body>
  <nav class="navbar navbar-default">
  <div class="container-fluid">
    <div class="navbar-header">
      <a class="navbar-brand" href="/logout">
        Logout
      </a>
    </div>
  </div>
</nav>
  <div class="container">
    <div class="row">
      <div class="col-md-6 col-md-offset-3">
        <h2>Welcome <span class="member-name"></span></h2>
      </div>
    </div>
  </div>

  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
  <script type="text/javascript" src="js/members.js"></script>

</body>

</html>

```

/ └ 📃*signup.html*

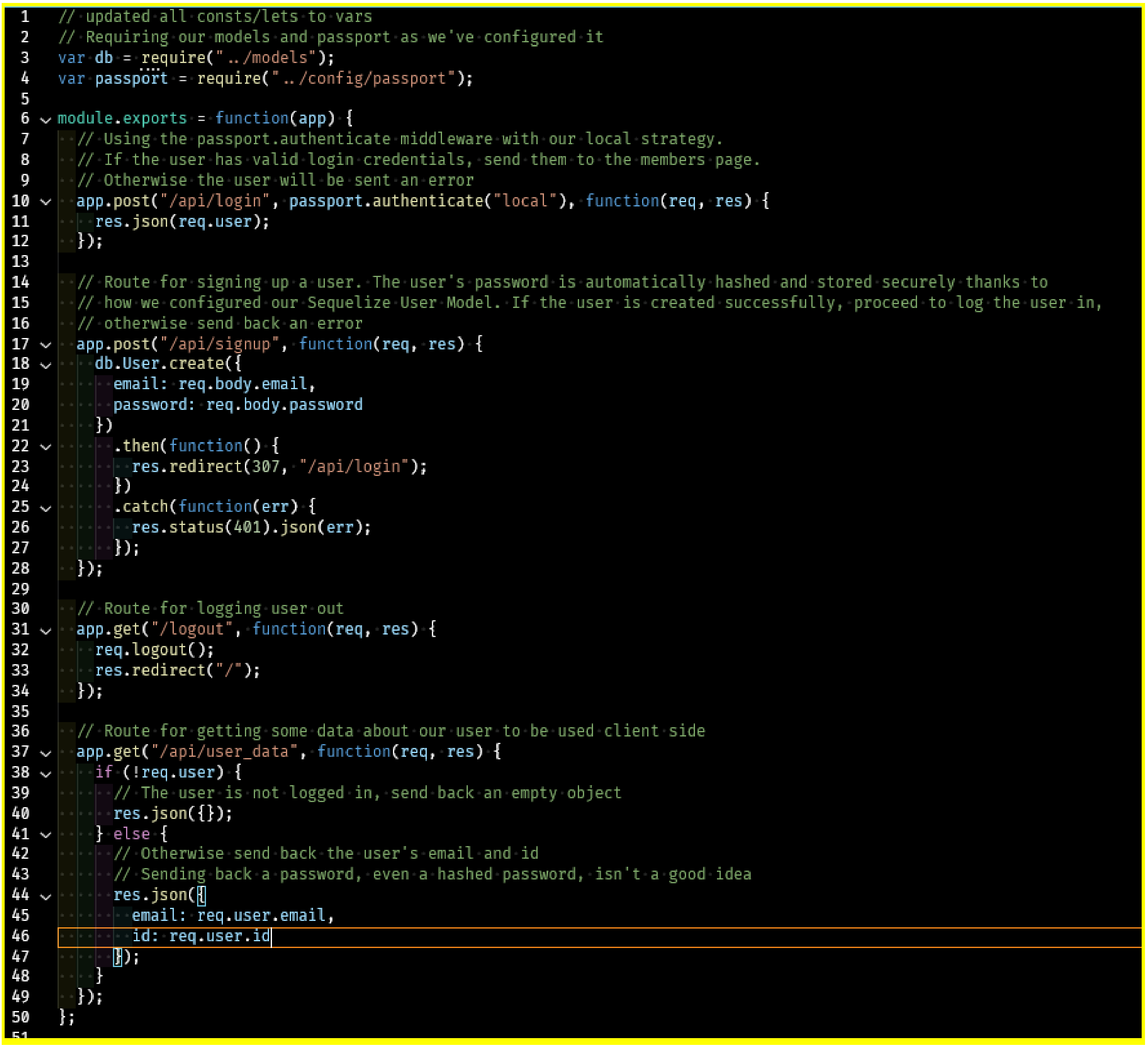


```html
<!-- html template for 'signup' page - to create a new user and store details on the database -->
<!DOCTYPE html>
<html lang="en">

<head>
  <title>Passport Authentication</title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootswatch/3.3.7/lumen/bootstrap.min.css">
  <link href="stylesheets/style.css" rel="stylesheet">
</head>

<body>
  <nav class="navbar navbar-default">
    <div class="container-fluid">
      <div class="navbar-header">
      </div>
    </div>
  </nav>
  <div class="container">
    <div class="row">
      <div class="col-md-6 col-md-offset-3">
        <h2>Sign Up Form</h2>
        <form class="signup">
          <div class="form-group">
            <label for="exampleInputEmail1">Email address</label>
            <input type="email" class="form-control" id="email-input" placeholder="Email">
          </div>
          <div class="form-group">
            <label for="exampleInputPassword1">Password</label>
            <input type="password" class="form-control" id="password-input" placeholder="Password">
          </div>
          <div style="display: none" id="alert" class="alert alert-danger" role="alert">
            <span class="glyphicon glyphicon-exclamation-sign" aria-hidden="true"></span>
            <span class="sr-only">Error:</span> <span class="msg"></span>
          </div>
          <button type="submit" class="btn btn-default">Sign Up</button>
        </form>
        <br />
        <p>Or log in <a href="/login">here</a></p>
      </div>
    </div>
  </div>

  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
  <script type="text/javascript" src="js/signup.js"></script>

</body>

</html>
```

├ 🗁routes

| ├ 🗏api-routes.js

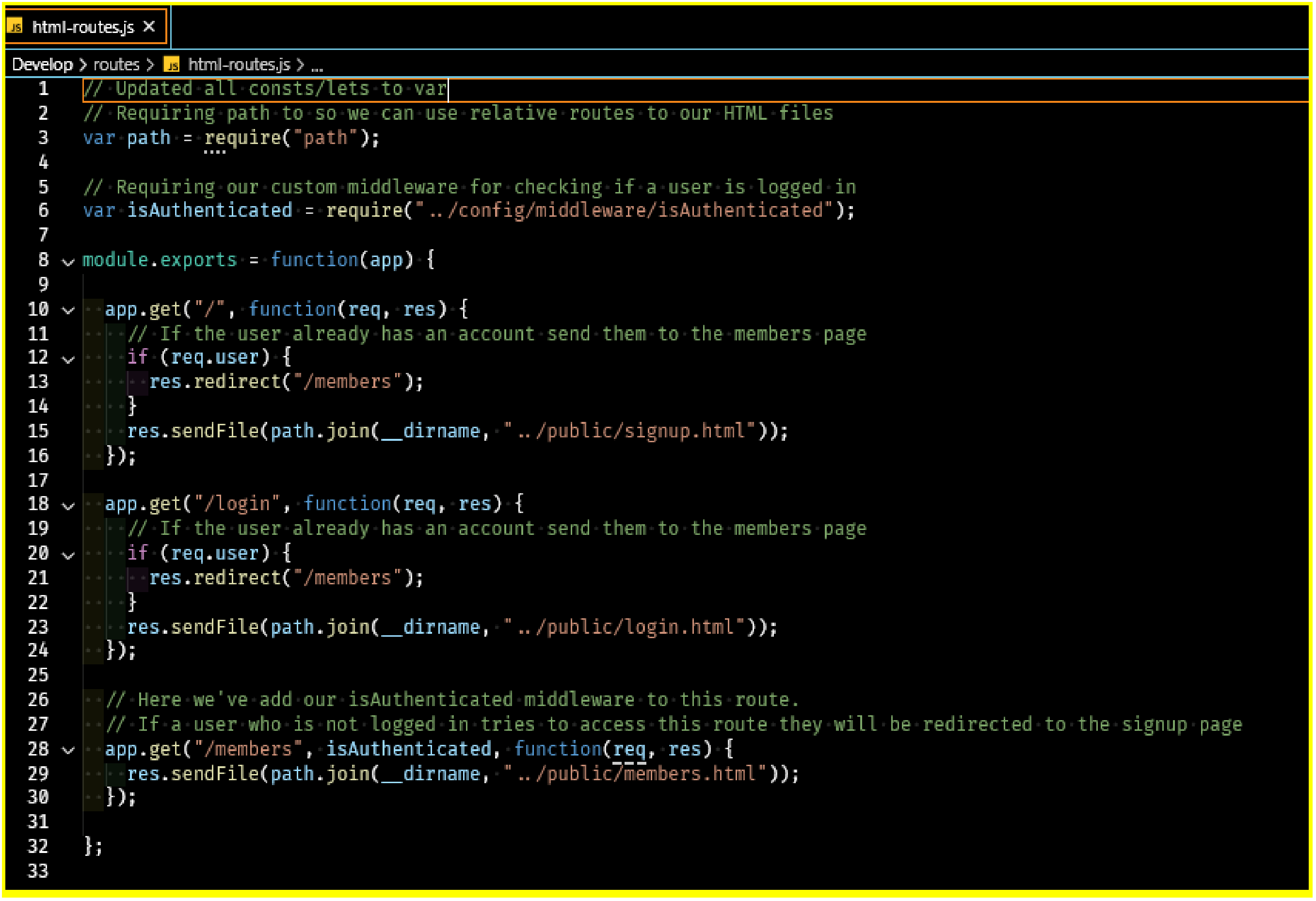
Contains routes for signing in, logging out and getting users specific data to be displayed client on the client (browser) side.



```
// updated all consts/lets to vars
// Requiring our models and passport as we've configured it
var db = require("../models");
var passport = require("../config/passport");

module.exports = function(app) {
  // Using the passport.authenticate middleware with our local strategy.
  // If the user has valid login credentials, send them to the members page.
  // Otherwise the user will be sent an error
  app.post("/api/login", passport.authenticate("local"), function(req, res) {
    res.json(req.user);
  });

  // Route for signing up a user. The user's password is automatically hashed and stored securely thanks to
  // how we configured our Sequelize User Model. If the user is created successfully, proceed to log the user in,
  // otherwise send back an error
  app.post("/api/signup", function(req, res) {
    db.User.create({
      email: req.body.email,
      password: req.body.password
    })
      .then(function() {
        res.redirect(307, "/api/login");
      })
      .catch(function(err) {
        res.status(401).json(err);
      });
  });

  // Route for logging user out
  app.get("/logout", function(req, res) {
    req.logout();
    res.redirect("/");
  });

  // Route for getting some data about our user to be used client side
  app.get("/api/user_data", function(req, res) {
    if (!req.user) {
      // The user is not logged in, send back an empty object
      res.json({});
    } else {
      // Otherwise send back the user's email and id
      // Sending back a password, even a hashed password, isn't a good idea
      res.json({
        email: req.user.email,
        id: req.user.id
      });
    }
  });
};
```

/ ┗ 📃html-routes.js

Routes that check whether the user is signed in, whether user already has account etc and sends them to the correct html page.

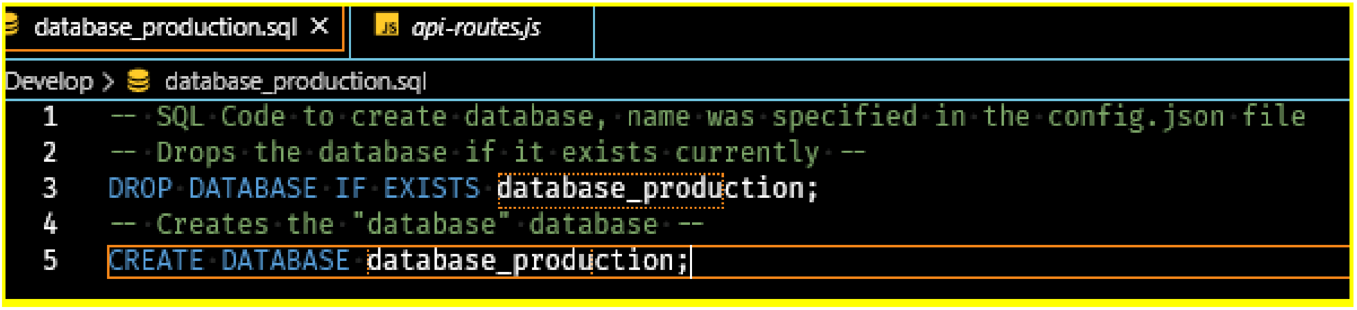


html-routes.js ×

Develop > routes > html-routes.js > ...

```javascript
// Updated all consts/lets to var
// Requiring path to so we can use relative routes to our HTML files
var path = require("path");

// Requiring our custom middleware for checking if a user is logged in
var isAuthenticated = require("../config/middleware/isAuthenticated");

module.exports = function(app) {

  app.get("/", function(req, res) {
    // If the user already has an account send them to the members page
    if (req.user) {
      res.redirect("/members");
    }
    res.sendFile(path.join(__dirname, "../public/signup.html"));
  });

  app.get("/login", function(req, res) {
    // If the user already has an account send them to the members page
    if (req.user) {
      res.redirect("/members");
    }
    res.sendFile(path.join(__dirname, "../public/login.html"));
  });

  // Here we've add our isAuthenticated middleware to this route.
  // If a user who is not logged in tries to access this route they will be redirected to the signup page
  app.get("/members", isAuthenticated, function(req, res) {
    res.sendFile(path.join(__dirname, "../public/members.html"));
  });

};
```

├ 📄database_production.sql (+)

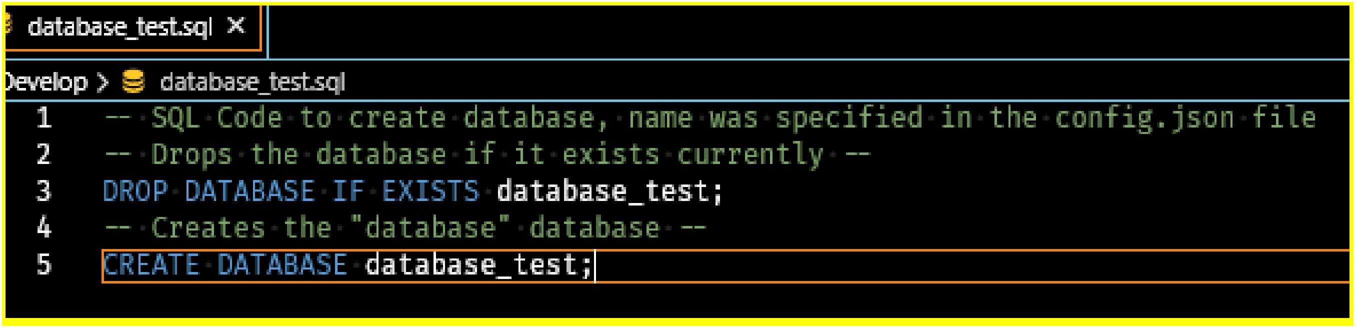


database_production.sql × | api-routes.js

Develop > database_production.sql

```sql
-- SQL Code to create database, name was specified in the config.json file
-- Drops the database if it exists currently --
DROP DATABASE IF EXISTS database_production;
-- Creates the "database" database --
CREATE DATABASE database_production;
```

├ 📄database_test.sql (+)



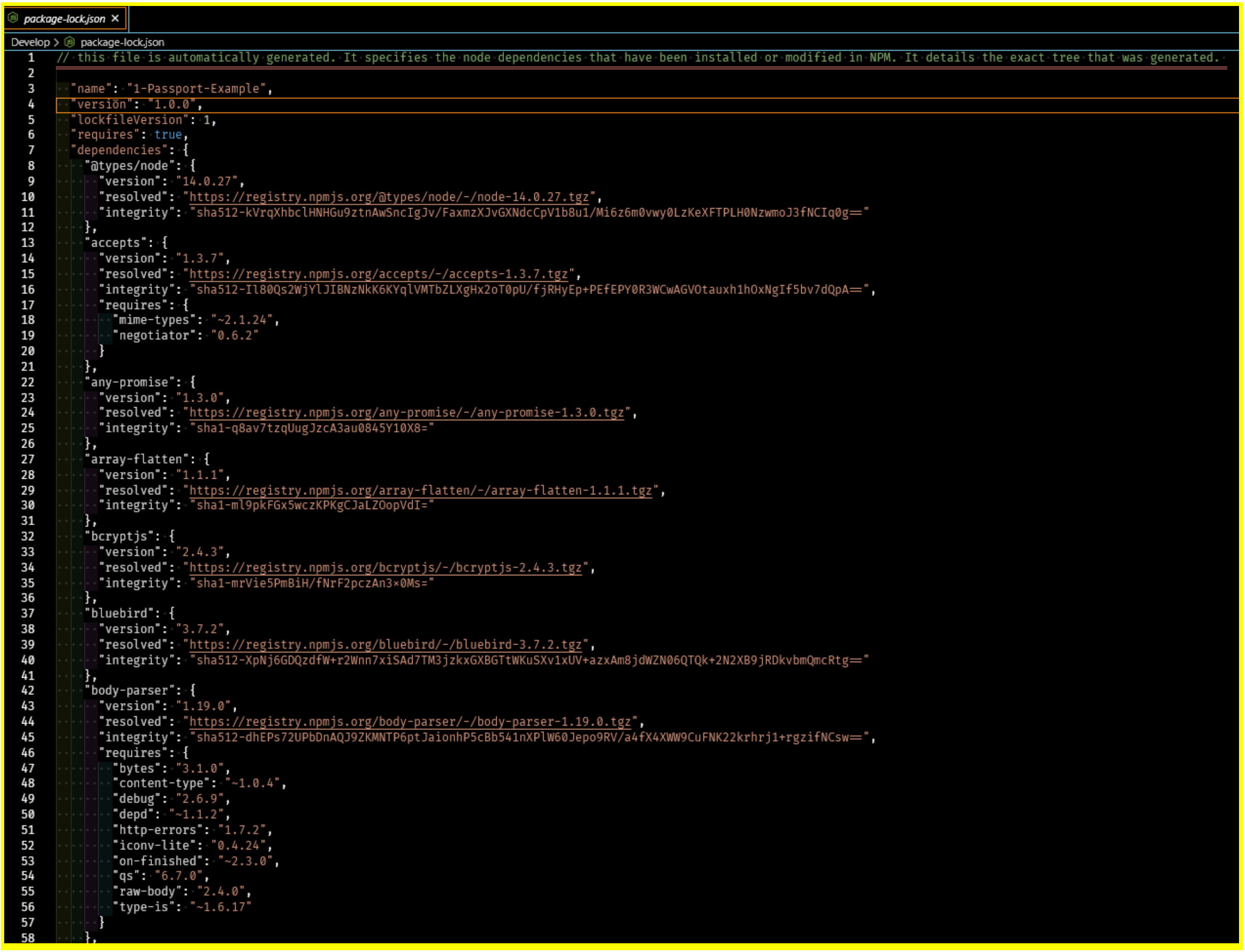
database_test.sql ×

Develop > database_test.sql

```sql
-- SQL Code to create database, name was specified in the config.json file
-- Drops the database if it exists currently --
DROP DATABASE IF EXISTS database_test;
-- Creates the "database" database --
CREATE DATABASE database_test;
```

├ 📄package-lock.json

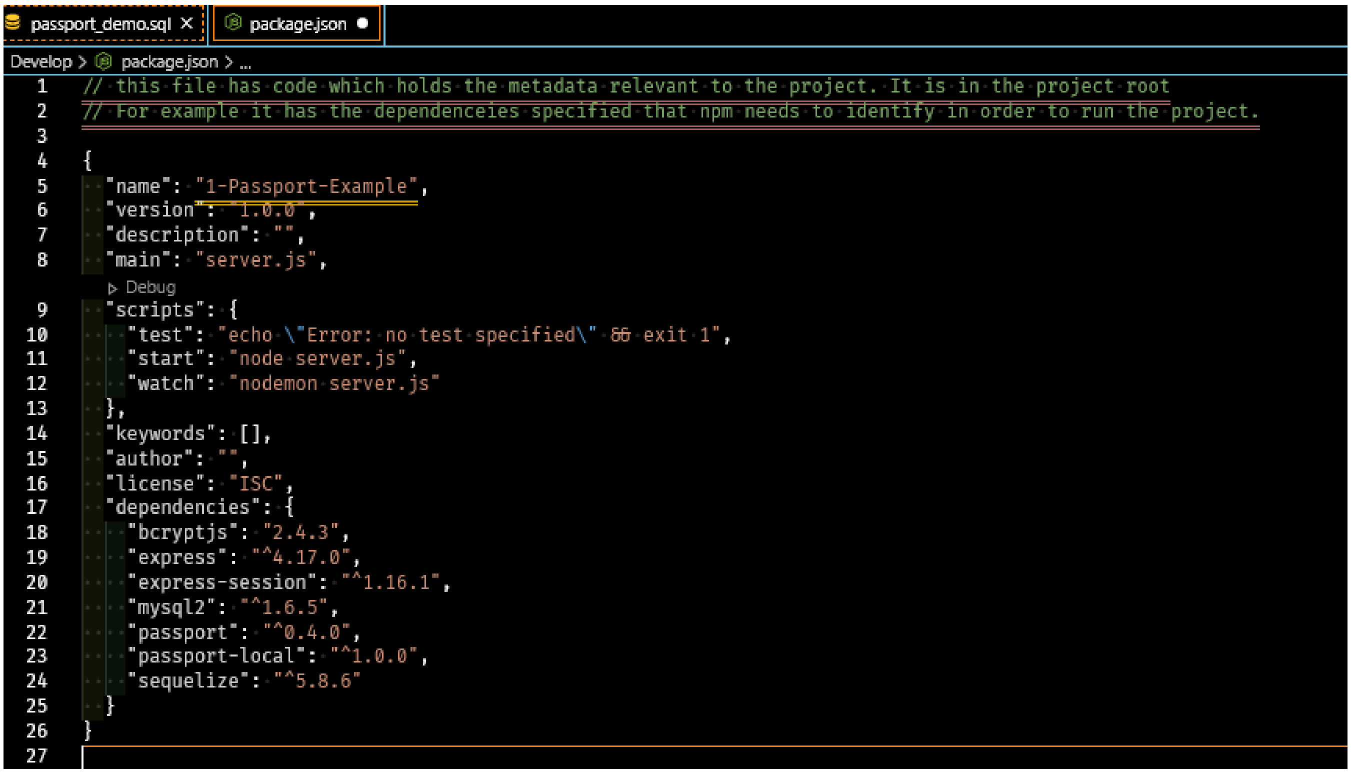
Picture below is one screen view of this file. It continues on to line 693 once all node modules are installed



```
// this file is automatically generated. It specifies the node dependencies that have been installed or modified in NPM. It details the exact tree that was generated.

  "name": "1-Passport-Example",
  "version": "1.0.0",
  "lockfileVersion": 1,
  "requires": true,
  "dependencies": {
    "@types/node": {
      "version": "14.0.27",
      "resolved": "https://registry.npmjs.org/@types/node/-/node-14.0.27.tgz",
      "integrity": "sha512-kVrqXhbclHNHGu9ztnAwSncIgJv/FaxmzXJvGXNdcCpV1b8u1/Mi6z6m0vwy0LzKeXFTPLH0NzwmoJ3fNCIq0g=="
    },
    "accepts": {
      "version": "1.3.7",
      "resolved": "https://registry.npmjs.org/accepts/-/accepts-1.3.7.tgz",
      "integrity": "sha512-Il80Qs2WjYlJIBNzNkK6KYqlVMTbZLXgHx2oT0pU/fjRHyEp+PEfEPY0R3WCwAGVOtauxh1hOxNgIf5bv7dQpA==",
      "requires": {
        "mime-types": "~2.1.24",
        "negotiator": "0.6.2"
      }
    },
    "any-promise": {
      "version": "1.3.0",
      "resolved": "https://registry.npmjs.org/any-promise/-/any-promise-1.3.0.tgz",
      "integrity": "sha1-q8av7tzqUugJzcA3au0845Y10X8="
    },
    "array-flatten": {
      "version": "1.1.1",
      "resolved": "https://registry.npmjs.org/array-flatten/-/array-flatten-1.1.1.tgz",
      "integrity": "sha1-ml9pkFGx5wczKPKgCJaLZOopVdI="
    },
    "bcryptjs": {
      "version": "2.4.3",
      "resolved": "https://registry.npmjs.org/bcryptjs/-/bcryptjs-2.4.3.tgz",
      "integrity": "sha1-mrVie5PmBiH/fNrF2pczAn3x0Ms="
    },
    "bluebird": {
      "version": "3.7.2",
      "resolved": "https://registry.npmjs.org/bluebird/-/bluebird-3.7.2.tgz",
      "integrity": "sha512-XpNj6GDQzdfW+r2Wnn7xiSAd7TM3jzkxGXBGTtWKuSXv1xUV+azxAm8jdWZN06QTQk+2N2XB9jRDkvbmQmcRtg=="
    },
    "body-parser": {
      "version": "1.19.0",
      "resolved": "https://registry.npmjs.org/body-parser/-/body-parser-1.19.0.tgz",
      "integrity": "sha512-dhEPs72UPbDnAQJ9ZKMNTP6ptJaionhP5cBb541nXPlW60Jepo9RV/a4fX4XWW9CuFNK22krhrj1+rgzifNCsw==",
      "requires": {
        "bytes": "3.1.0",
        "content-type": "~1.0.4",
        "debug": "2.6.9",
        "depd": "~1.1.2",
        "http-errors": "1.7.2",
        "iconv-lite": "0.4.24",
        "on-finished": "~2.3.0",
        "qs": "6.7.0",
        "raw-body": "2.4.0",
        "type-is": "~1.6.17"
      }
    },
```

├ 📃*package.json*

Contains all package info, node modules used, version info etc .

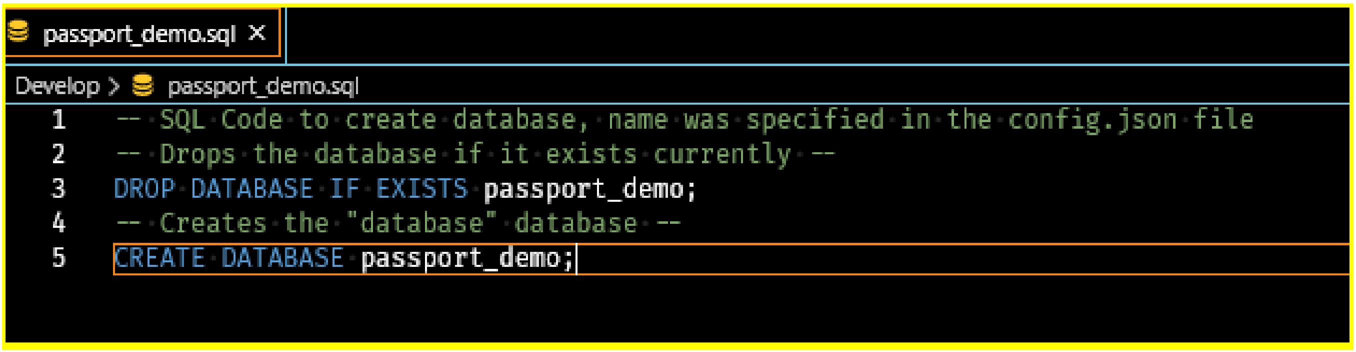


passport_demo.sql × | package.json ●

Develop > package.json > ...

```
// this file has code which holds the metadata relevant to the project. It is in the project root
// For example it has the dependenceies specified that npm needs to identify in order to run the project.

{
  "name": "1-Passport-Example",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js",
    "watch": "nodemon server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "2.4.3",
    "express": "^4.17.0",
    "express-session": "^1.16.1",
    "mysql2": "^1.6.5",
    "passport": "^0.4.0",
    "passport-local": "^1.0.0",
    "sequelize": "^5.8.6"
  }
}
```

├ *passport_demo.sql (+)*



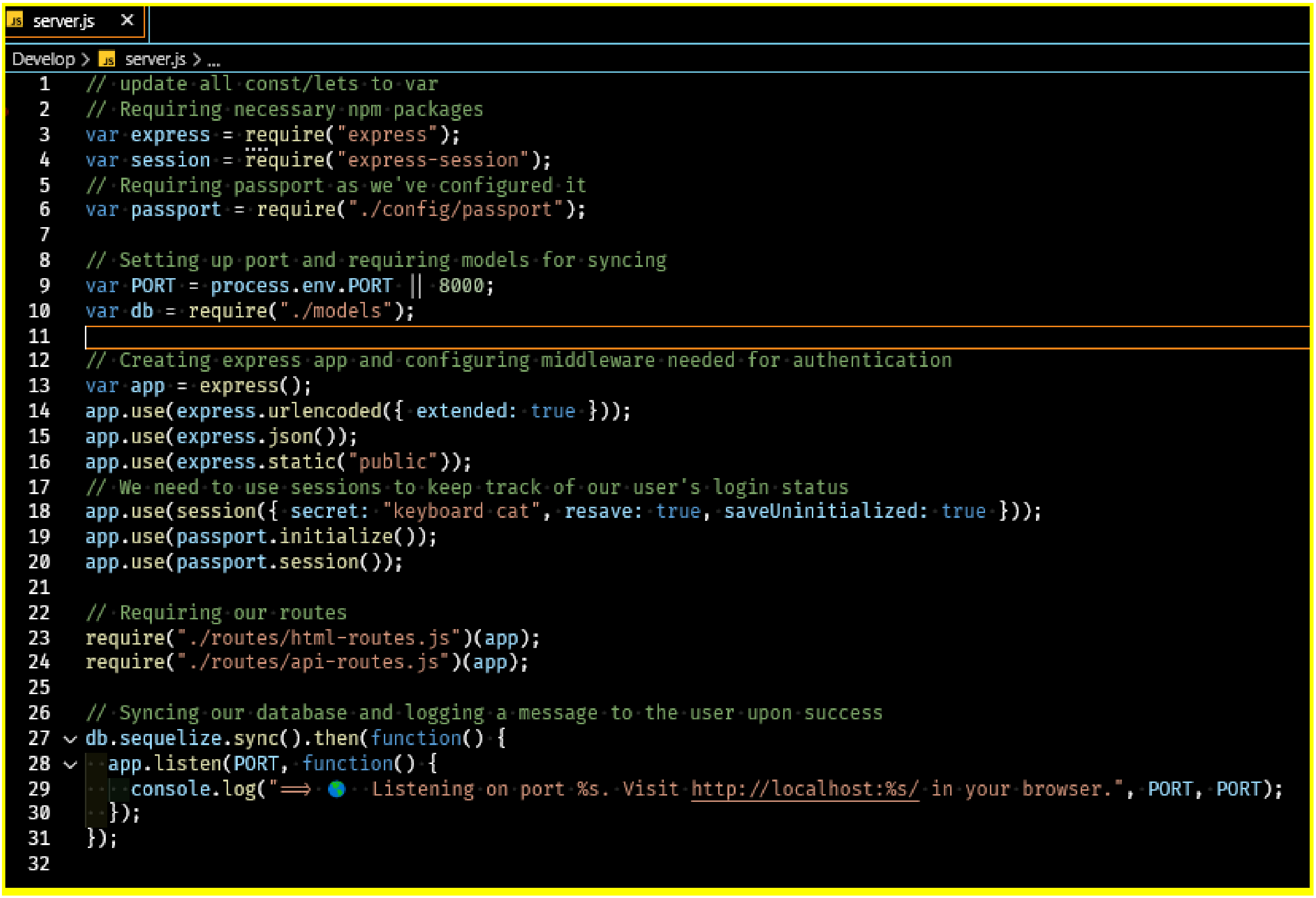
passport_demo.sql ×

Develop > passport_demo.sql

```sql
-- SQL Code to create database, name was specified in the config.json file
-- Drops the database if it exists currently --
DROP DATABASE IF EXISTS passport_demo;
-- Creates the "database" database --
CREATE DATABASE passport_demo;
```

├ *server.js*

Requires packages, sets up PORT, creates express and middleware, creates routes and syncs database / logs message in terminal on successful connection to server.



```javascript
// update all const/lets to var
// Requiring necessary npm packages
var express = require("express");
var session = require("express-session");
// Requiring passport as we've configured it
var passport = require("./config/passport");

// Setting up port and requiring models for syncing
var PORT = process.env.PORT || 8000;
var db = require("./models");

// Creating express app and configuring middleware needed for authentication
var app = express();
app.use(express.urlencoded({ extended: true }));
app.use(express.json());
app.use(express.static("public"));
// We need to use sessions to keep track of our user's login status
app.use(session({ secret: "keyboard cat", resave: true, saveUninitialized: true }));
app.use(passport.initialize());
app.use(passport.session());

// Requiring our routes
require("./routes/html-routes.js")(app);
require("./routes/api-routes.js")(app);

// Syncing our database and logging a message to the user upon success
db.sequelize.sync().then(function() {
  app.listen(PORT, function() {
    console.log("==> 🌎  Listening on port %s. Visit http://localhost:%s/ in your browser.", PORT, PORT);
  });
});
```

TESTING

Where to test

1. Console log (connection)
2. Browser (html, routing)
3. Database (routing).

HOW TO IMPROVE CODEBASE

Describes what can be done next to start your project right.

A few tips on what you can do next to improve this code:

- Change all vars to consts and lets where relevant
- Add warnings when the user tries to sign up with a email that is already in the database
- Add warning when the password does not meet requirements
- Create some additional features in the members page to customise such as 'date joined' and edit password section.
- Potentially use AJAX to streamline API call functions
- Move isAuthenticated to within the html routes file
- Integrate the app into a dummy website to practically demonstrate its features. A relevant example might be a shopping site/ login to access wishlist, shopping cart, checkout etc. Another example might be a forum.
- Create github and heroku repos to host your new project and REMEMBER to create a .gitignore file for your node modules!

RESOURCES

- Haverbeke, M. Asynchronous Programming :: Eloquent JavaScript. (2020). Retrieved 21 August 2020, from https://eloquentjavascript.net/11_async.html
- Kadwill, T. Using Sequelize ORM with Node.js and Express. (2020). Retrieved 21 August 2020, from https://stackabuse.com/using-sequelize-orm-with-nodejs-and-express/
- Monash University/ Trilogy Education. (2020). Retrieved 21 August 2020, from https://monash.bootcampcontent.com/monash-coding-bootcamp/monu-mel-fsf-pt-05-2020-u-c/tree/master/14-Full-Stack
- NPM. npm-package-lock.json | npm Documentation. (2020). Retrieved 21 August 2020, from https://docs.npmjs.com/configuring-npm/package-lock-json.html#:~:text=Description,regardless%20of%20intermediate%20dependency%20updates.
- Open JS Foundation. Path | Node.js v14.8.0 Documentation. (2020). Retrieved 21 August 2020, from https://nodejs.org/api/path.html
- Open JS Foundation. What is the file `package.json`? | Node.js. (2020). Retrieved 21 August 2020, from https://nodejs.org/en/knowledge/getting-started/npm/what-is-the-file-package-json/#:~:text=All%20npm%20packages%20contain%20a,as%20handle%20the%20project's%20dependencies.&text=The%20package.,-json%20file%20is
- W3 Schools. Node.js File System Module. (2020). Retrieved 21 August 2020, from https://www.w3schools.com/nodejs/nodejs_filesystem.asp