REVERSE ENGINEERING

Tutorial for Node JS application using Sequelize and Passport,

Walkthrough to understand the codebase by Niyati Patel

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

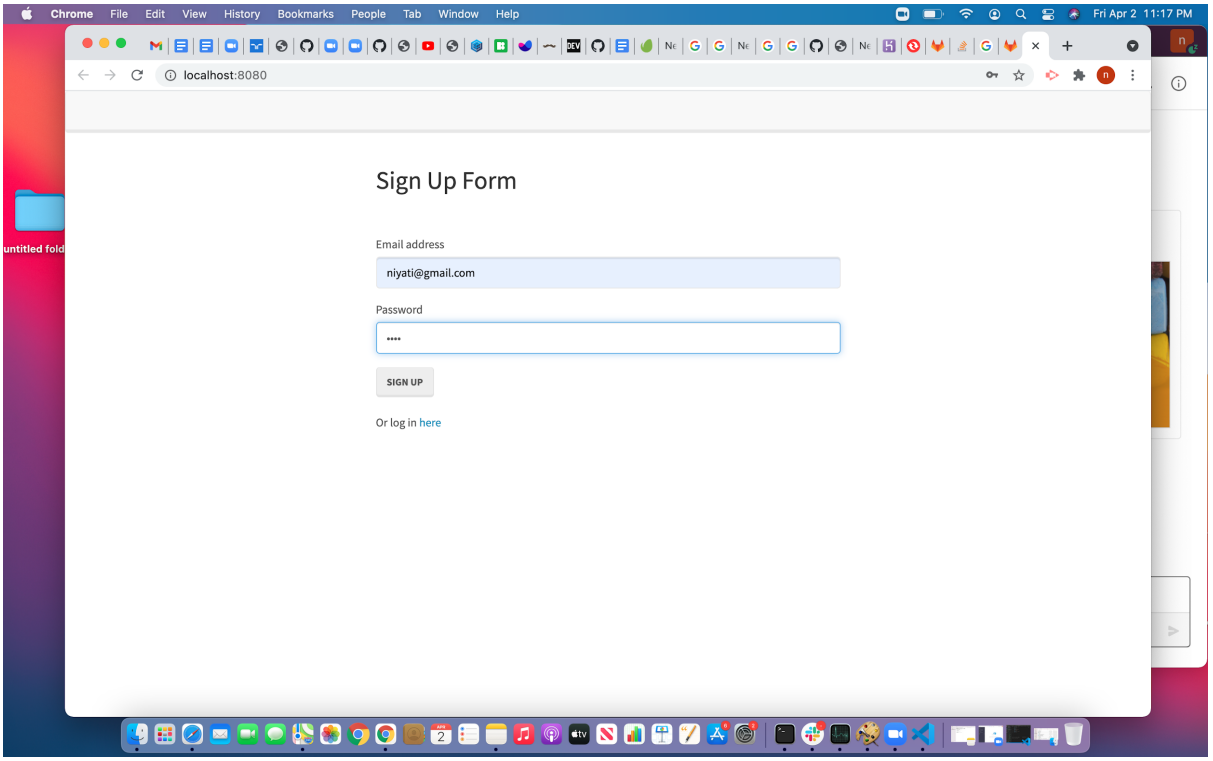
├ passport_demo.sql (+)

├ server.js

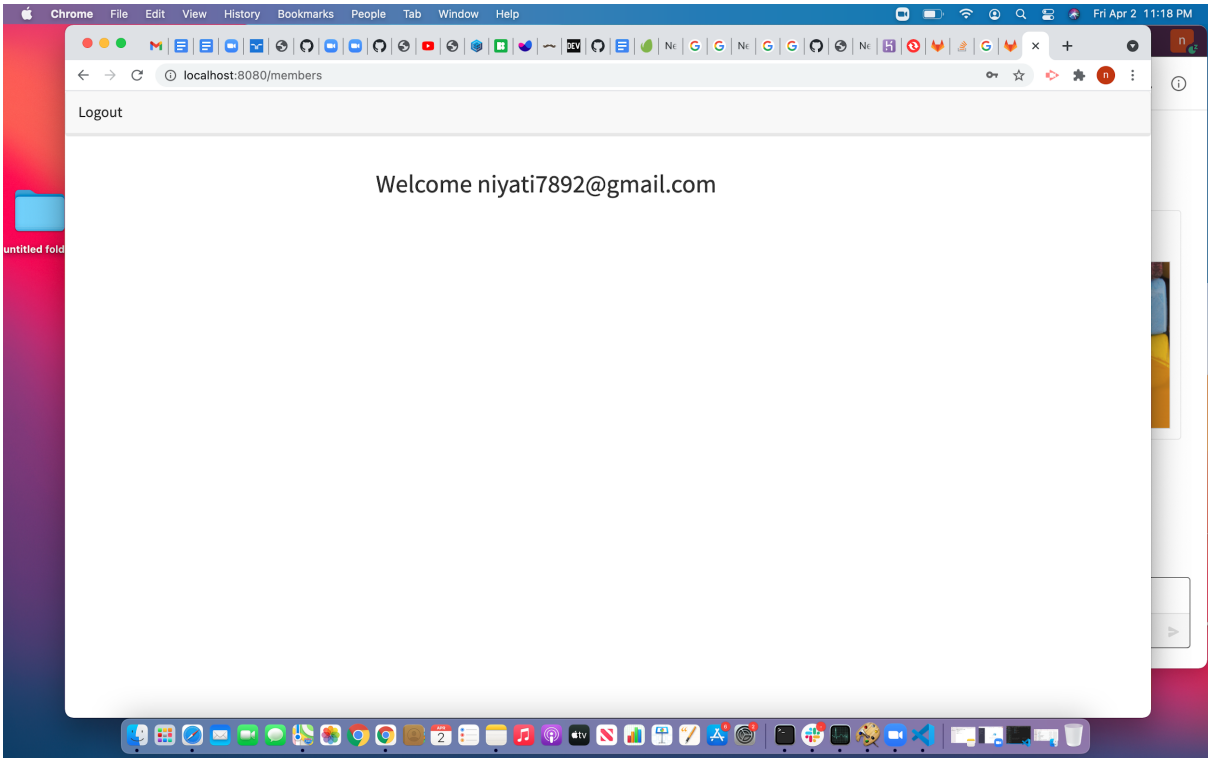
WALKTHROUGH OF CODE

Quick Instructions

1. *Begin using this codebase by first cloning the repository into your local storage.*
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*
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.*
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.*
10. *Check the mySQL database to ensure login data is saved.*
11.
    a.


Chrome File Edit View History Bookmarks People Tab Window Help
Fri Apr 2 11:17 PM
localhost:8080
Sign Up Form
Email address
niyati@gmail.com
Password
SIGN UP
Or log in here

b.


Chrome File Edit View History Bookmarks People Tab Window Help
Fri Apr 2 11:18 PM
localhost:8080/members
Logout
Welcome niyati7892@gmail.com

File Walk Through with Steps

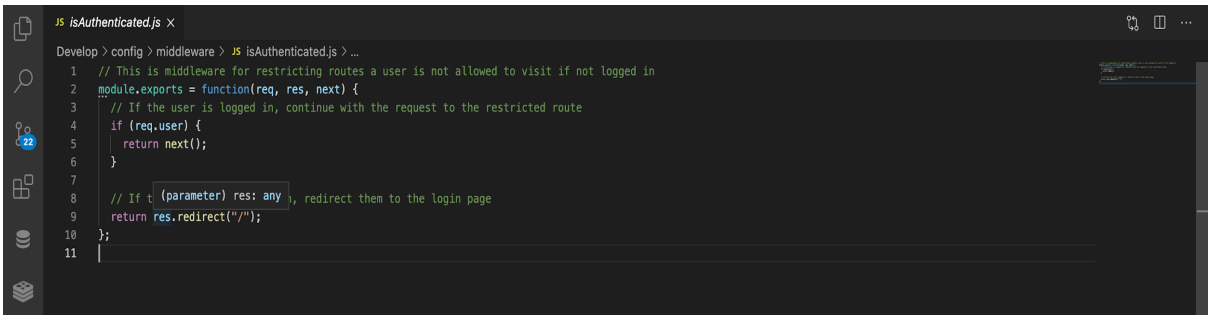
📦Develop

├ 📂config

| ├ 📂middleware

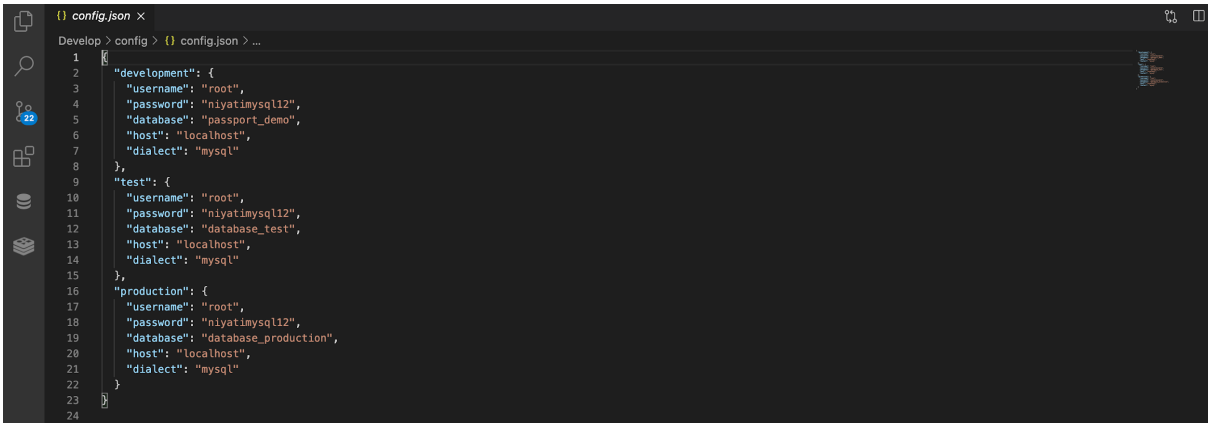
| | └ 📜isAuthenticated.js

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.



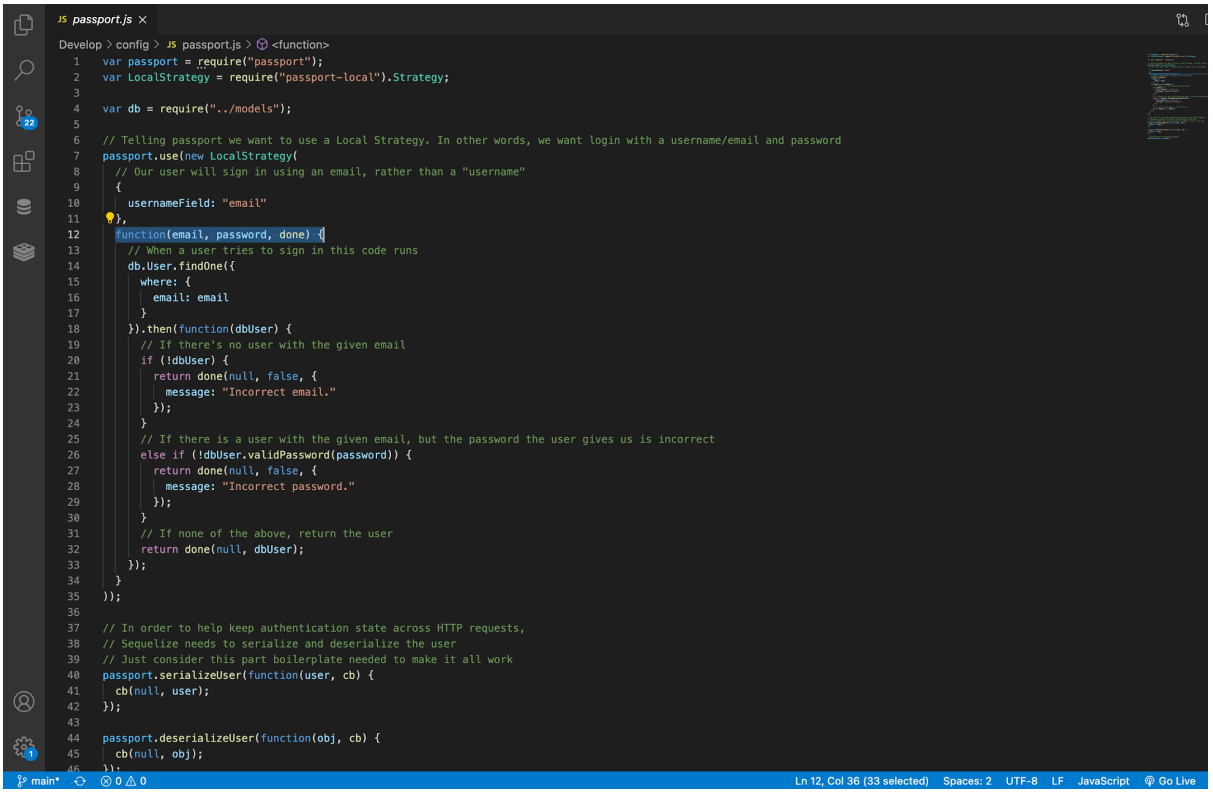
| ├ 📜config.json

middleware connection configuration to connect to server.



| └ 📜passport.js

Middleware contains javascript logic that tells the passport we want to log in with an email address and password. Its show in line 13



├ 📂*models*

| ├ 📜*index.js*

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

1.This file requires the following npm packages: fs, path, sequelize, path, and dotenv(.env). It also requires the config.json file from the config folder.

*2.*A variable is declared as DB and is an empty object. It then uses a conditional statement and uses .env function. The .env functionality is an 'environmental variable' created outside the program (typically through the operating system) and put inside a new variable if it is not an environmental variable then it will configure the variable, pass it through the database - username and password tables.

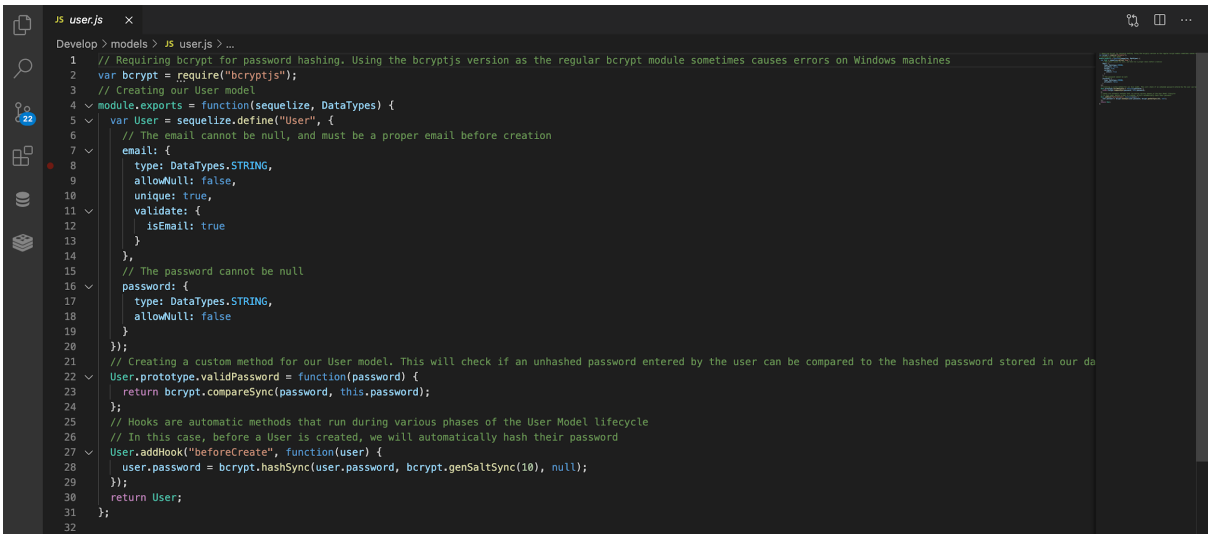
*3.*Lines 17-25 use the file system or 'fs' npm module. The FS functions enable a synchronous connection to the DB directory to the index.js file. The Filter function checks that in that connection call there is something to read, cannot be this file and cannot end in .js. Each model file is then put n an object to export.

4. Lines 27-31 with the model files retrieved it will use the associate function to the model if it matches.

*5.*The last lines export the sequelize library.

┃ ┗ 📜*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.



┣ 📂*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.

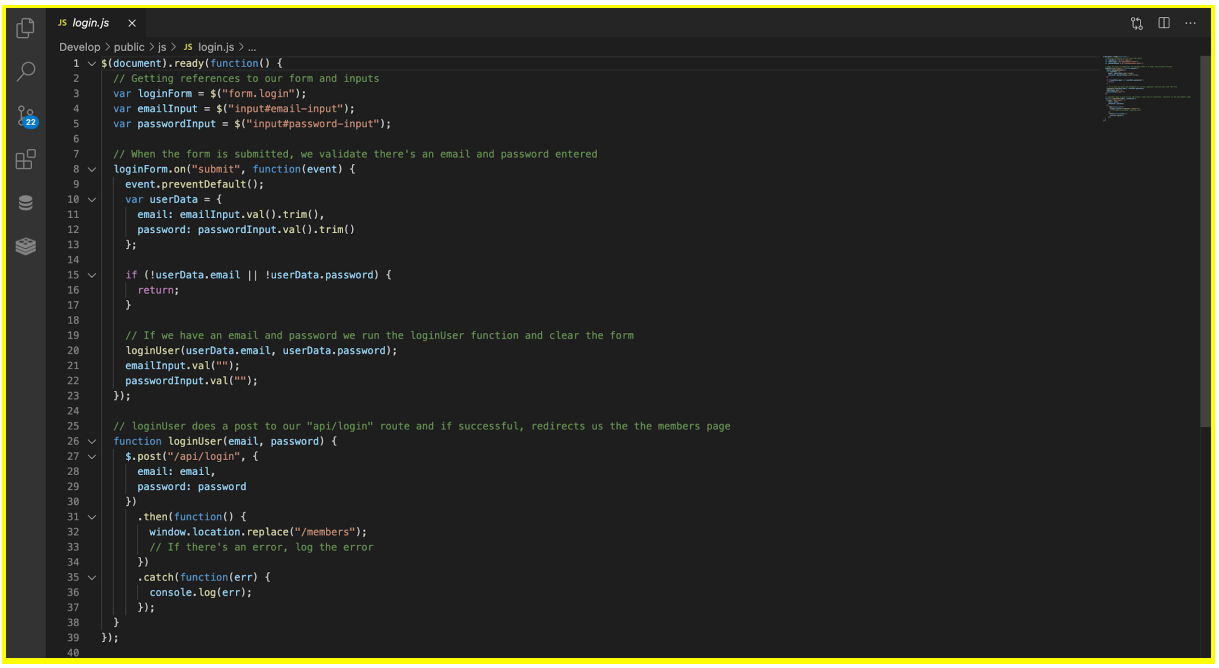
┃ ┣ 📂*.bin*

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

├ 📂*public*

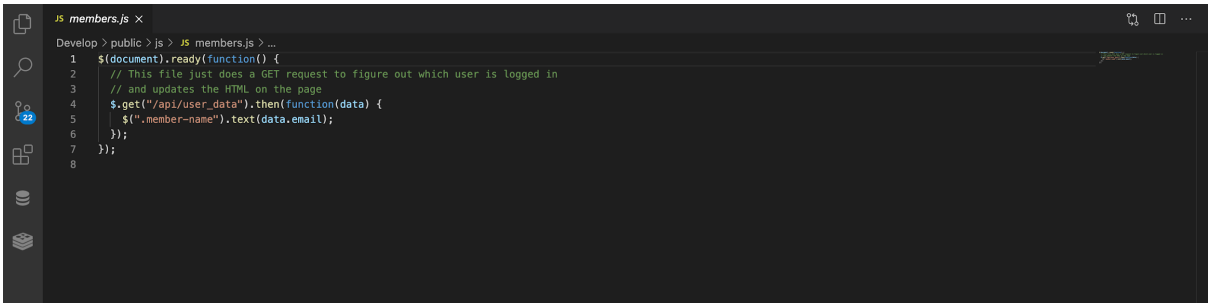
│ ├ 📂*js*

│ │ ├ 📜*login.js*



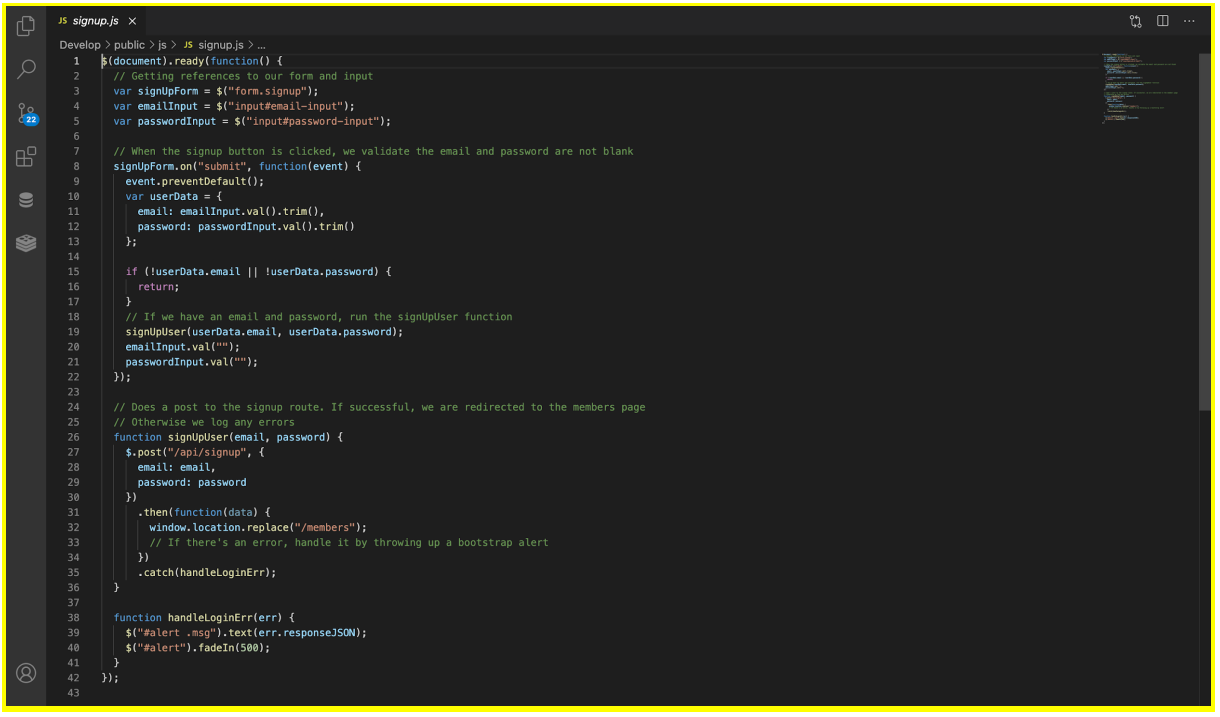
```
$(document).ready(function() {
  // Getting references to our form and inputs
  var loginForm = $("form.login");
  var emailInput = $("input#email-input");
  var passwordInput = $("input#password-input");

  // 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);
  });
});
```

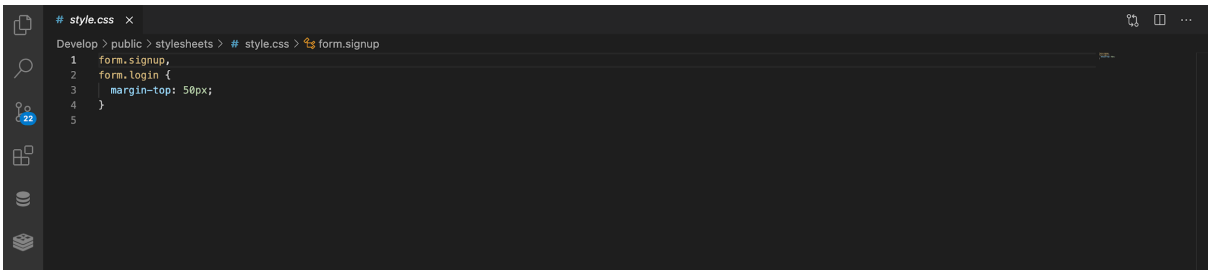
```
|  |  └ 📜signup.js
```



```
signup.js
Develop > public > js > signup.js > ...
```

```
$(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);
  }
});
```

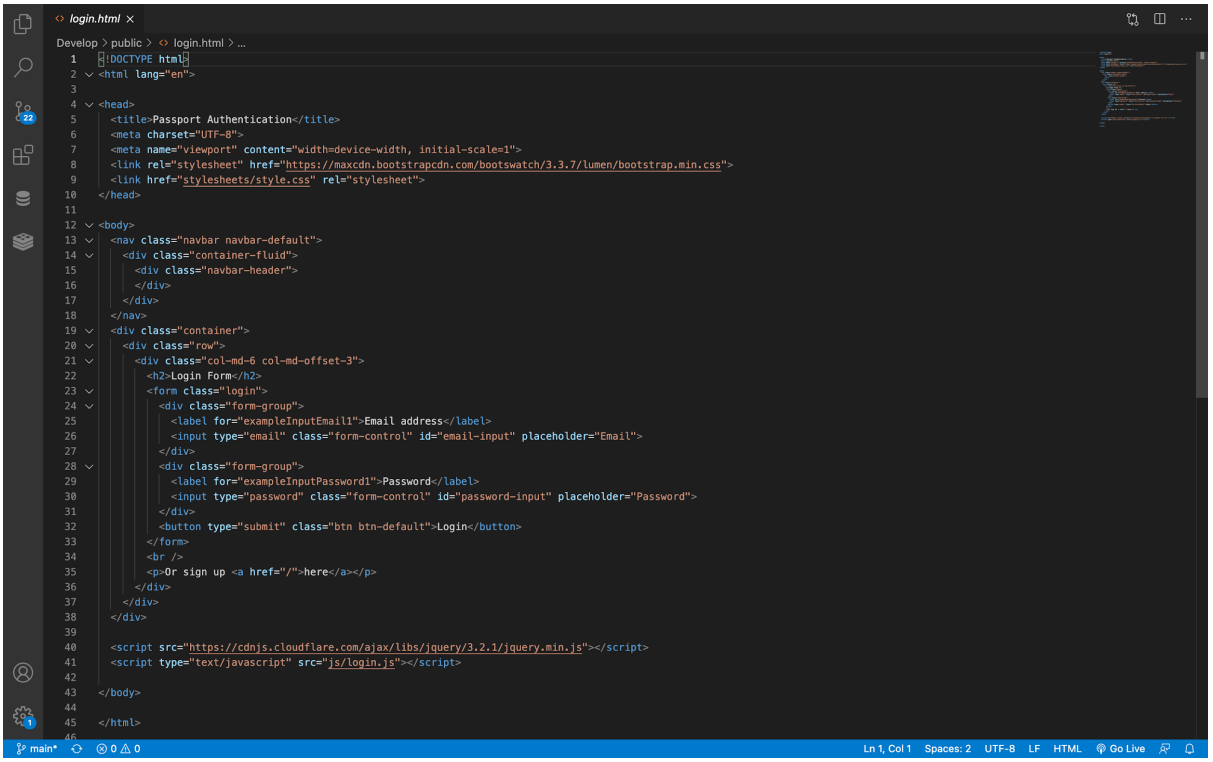
```
|  ├ 📂stylesheets
|  |  └ 📜style.css
```



```
style.css
Develop > public > stylesheets > style.css > form.signup
```

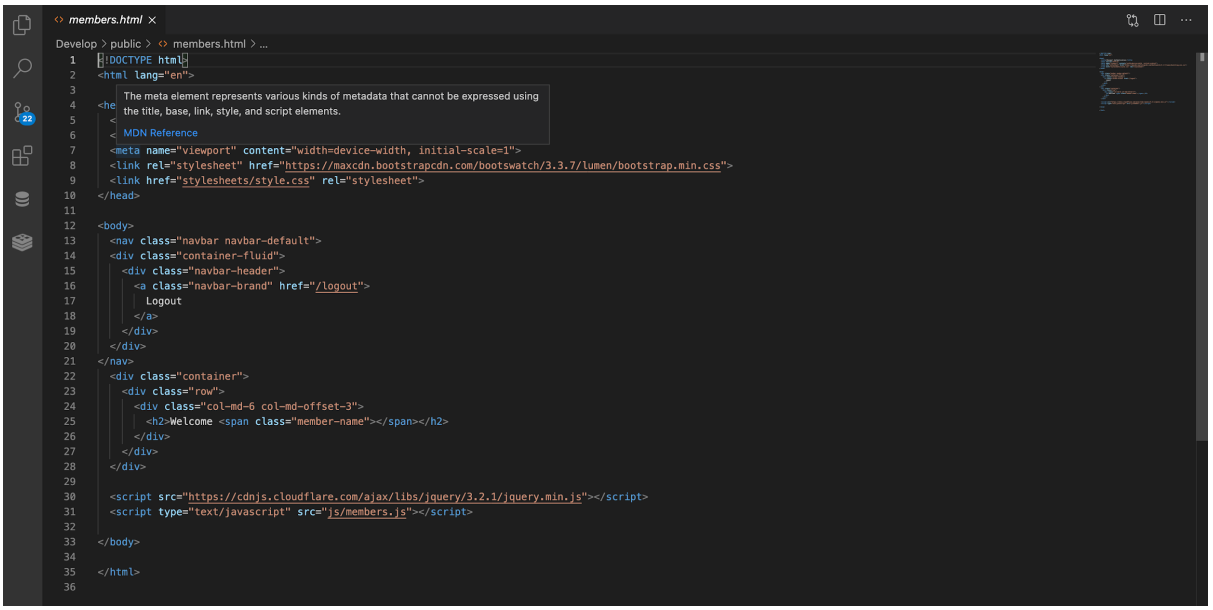
```
form.signup,
form.login {
  margin-top: 50px;
}
```

| ├ 📜*login.html*



```
<!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*

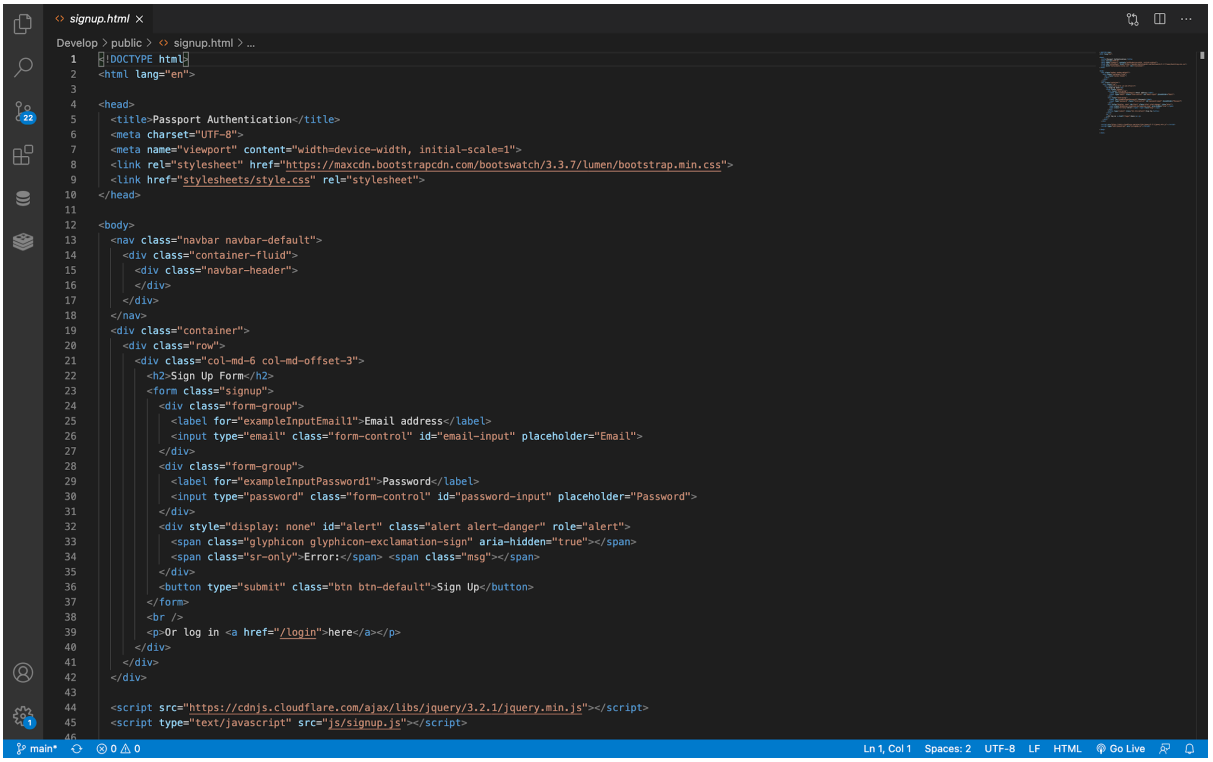


```
<!DOCTYPE html>
<html lang="en">

<he
  <
  <
  <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>
```

The meta element represents various kinds of metadata that cannot be expressed using the title, base, link, style, and script elements.

MDN Reference

signup.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.

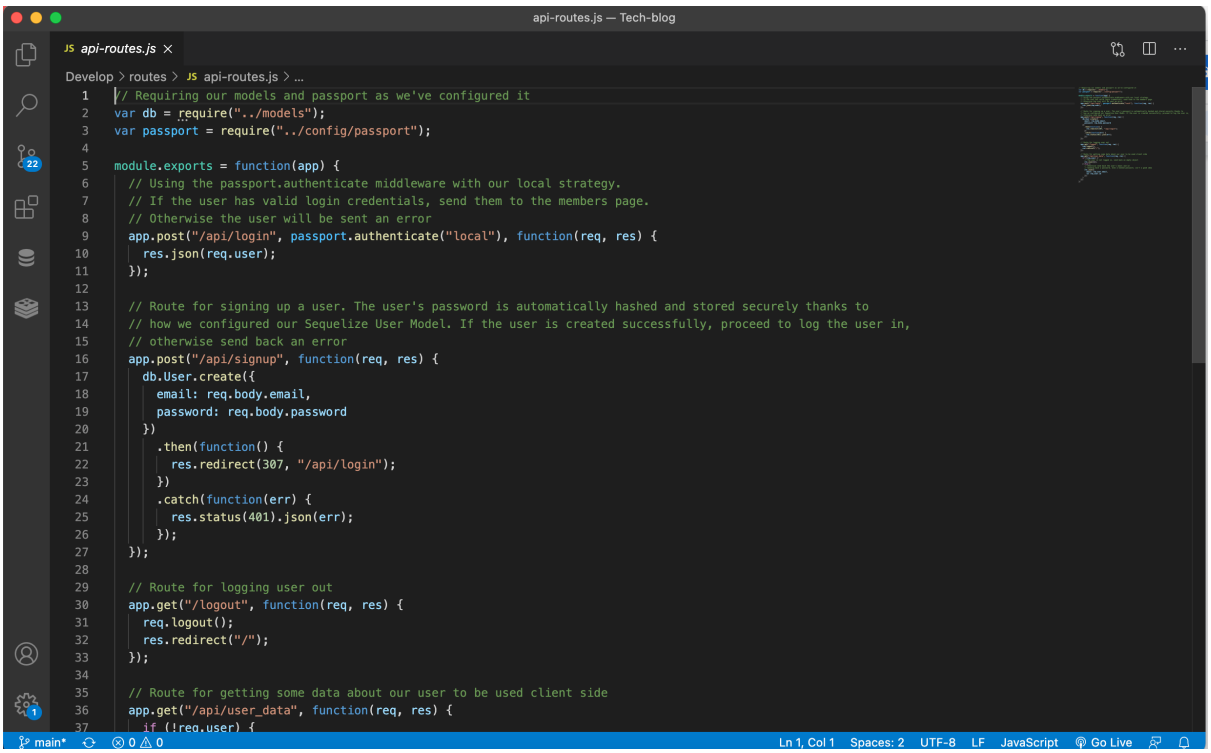
1. This file requires the models folder and buts it inside the db variable. It also requires the passport file and puts it inside the passport variable.

2. Lines 5-11 logging the user out. And it sets up a route for It uses the function passport.authenticate middleware to validate user login credentials, if successful it will send them to the main page, if unsuccessful the user will get an error message.

3. Lines 16-27 creates an API route for new users that sign up. It creates a post into the DB, using the User table, it creates a new user with the email and password attributes. If the new user is created successfully it will log the user in, if unsuccessful it will send back a 401-error message.

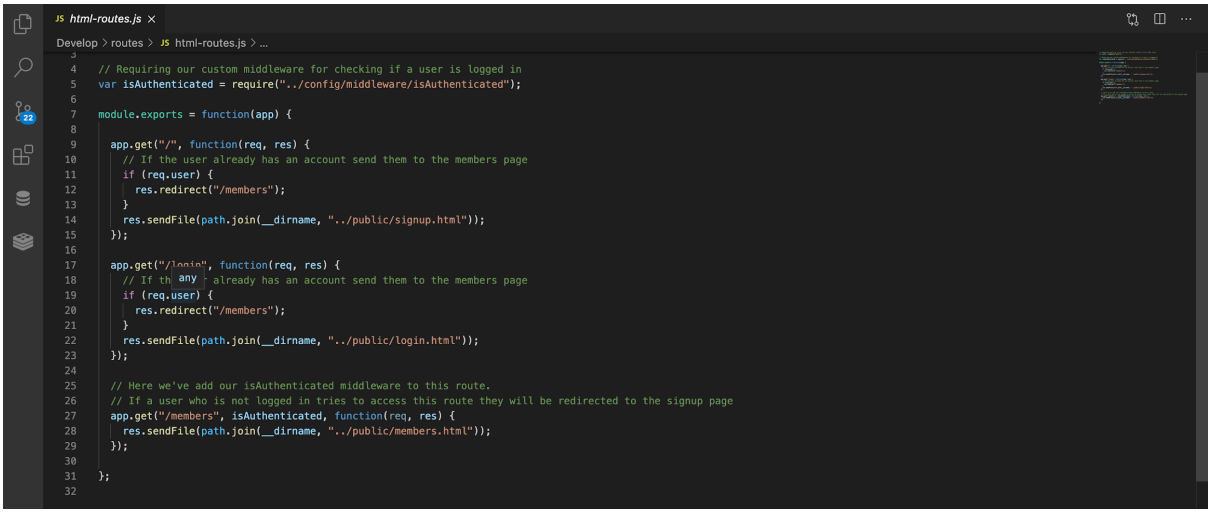
4. Lines 30-33 create a route for users to log out, redirecting them to the home page once logged out.

5. Lines 36-49 create a GET route to send some user data (email and user id) to the end user (browser) if the user is logged in. If the user is not logged in it will send an empty object.



| └ 📜*html-routes.js*

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



⊢ 📜*database_production.sql (+)*

database_production.sql — Tech-blog

database_production.sql ×

database_production.sql

Run SQL

```
1  DROP DATABASE IF EXISTS database_production;
```

Run SQL

```
2  CREATE DATABASE database_production;
3
```

⊢ 📜*database_test.sql (+)*

database_test.sql ×

database_test.sql

Run SQL

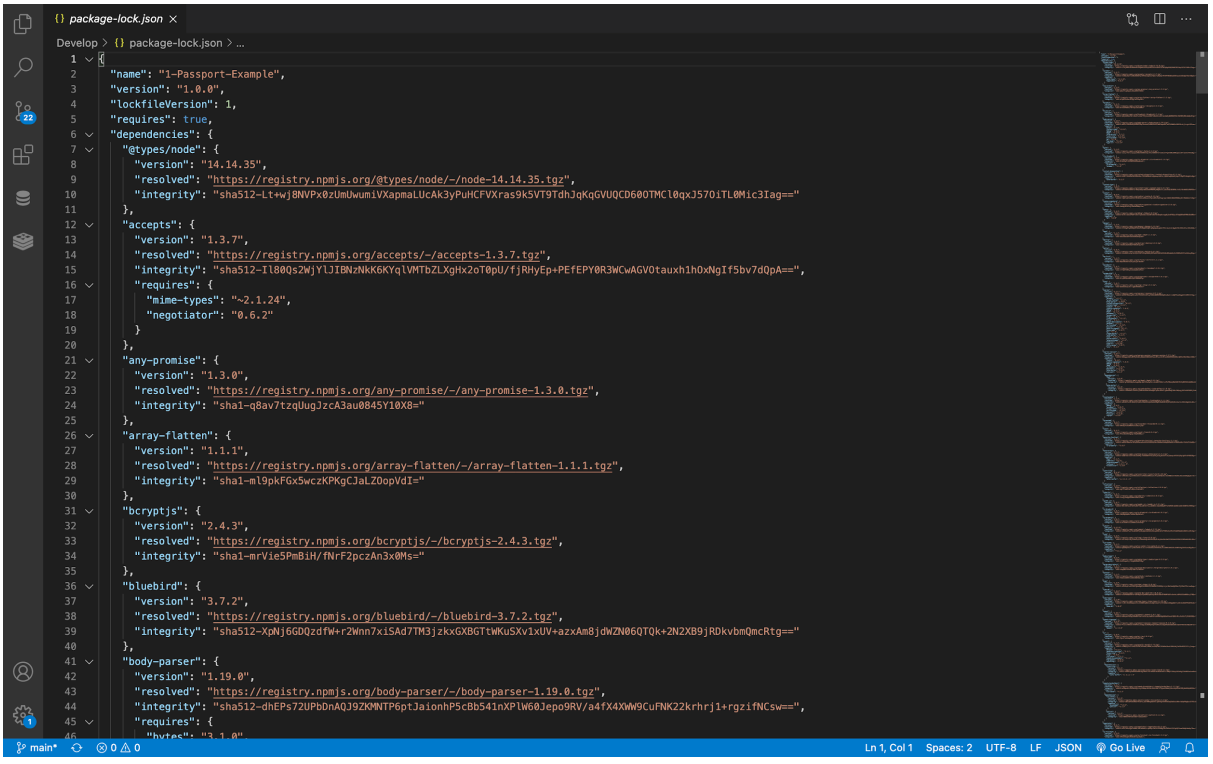
```
1  DROP DATABASE IF EXISTS database_test;
2
```

Run SQL

```
3  CREATE DATABASE database_test;
4
```

⊢ 📜*package-lock.json*

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



package-lock.json ×

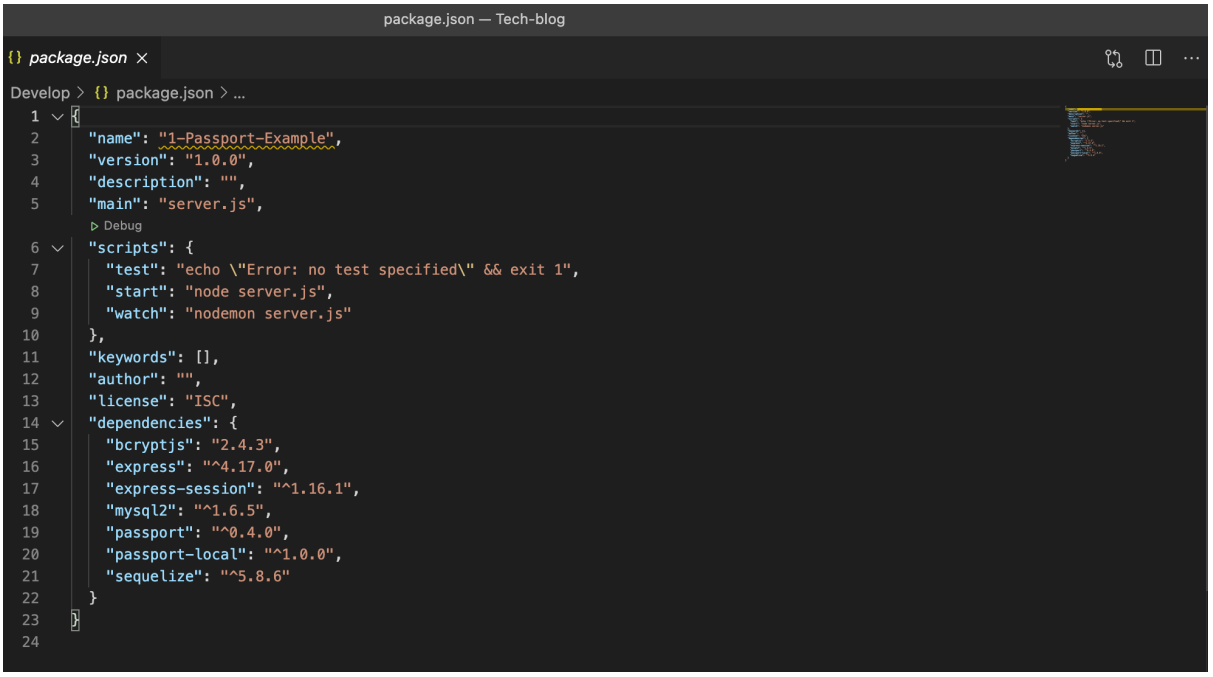
Develop > {} package-lock.json > ...

```
{
  "name": "1-Passport-Example",
  "version": "1.0.0",
  "lockfileVersion": 1,
  "requires": true,
  "dependencies": {
    "@types/node": {
      "version": "14.14.35",
      "resolved": "https://registry.npmjs.org/@types/node/-/node-14.14.35.tgz",
      "integrity": "sha512-Lt+wj8NVPx0zUmUwumiVXapmaLUcAk3yPuHCFVXras9k5VT9TdhJqKqGVUQCD60OTMCl0qxJ57OiTL0Mic3Iag=="
    },
    "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",
```

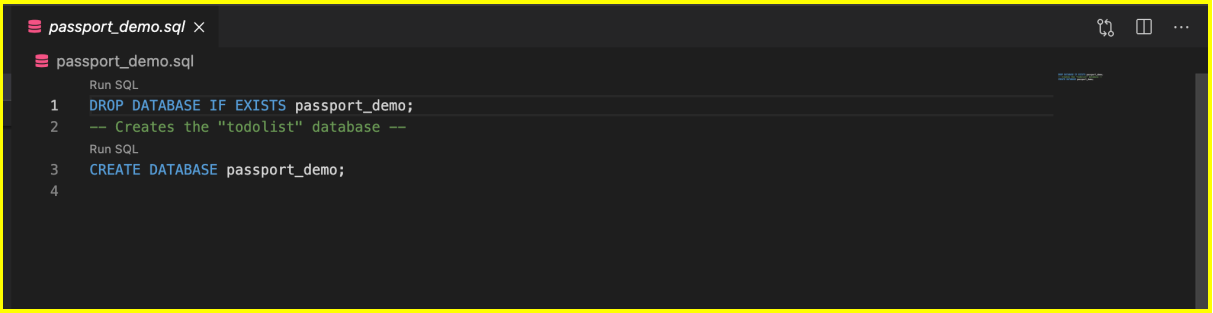
main* ⊗ 0 ⚠ 0 — Ln 1, Col 1 Spaces: 2 UTF-8 LF JSON Go Live

├ 📜*package.json*

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



├ 📜*passport_demo.sql (+)*



├ 📜*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.

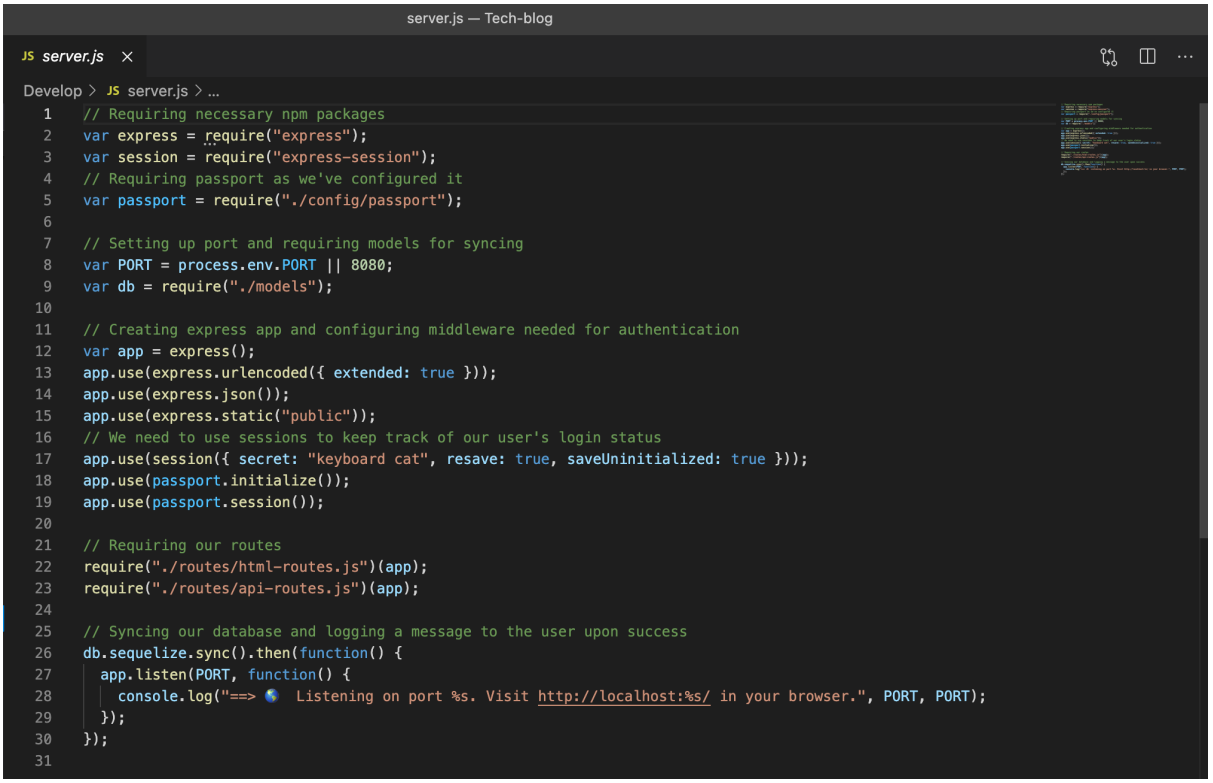
1. This file requires several npm packages – express, express-session and it also requires passport.

2. It first sets a PORT connection to a database, the database can be local or external as indicated on the config.json file on the config folder.

3. Lines 12-19 it creates a variable using the express library app function. Using the app function from the express library it configures the middleware needed for authentication. The last three lines track the user's login status by creating a session from the moment they log in.

4. Lines 22-23 require the API routes files in the directory and use the app function to fun.

5. Lines 26-30 – this function syncs the databases, it then console.log a message when the connection is successful.



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!