**Reverse Engineering Code**

**Tutorial explanation for files and its purpose**

**Table of Contents**

1. Overview
2. Files Review
3. Instructions for Project Improvements

**Overview**

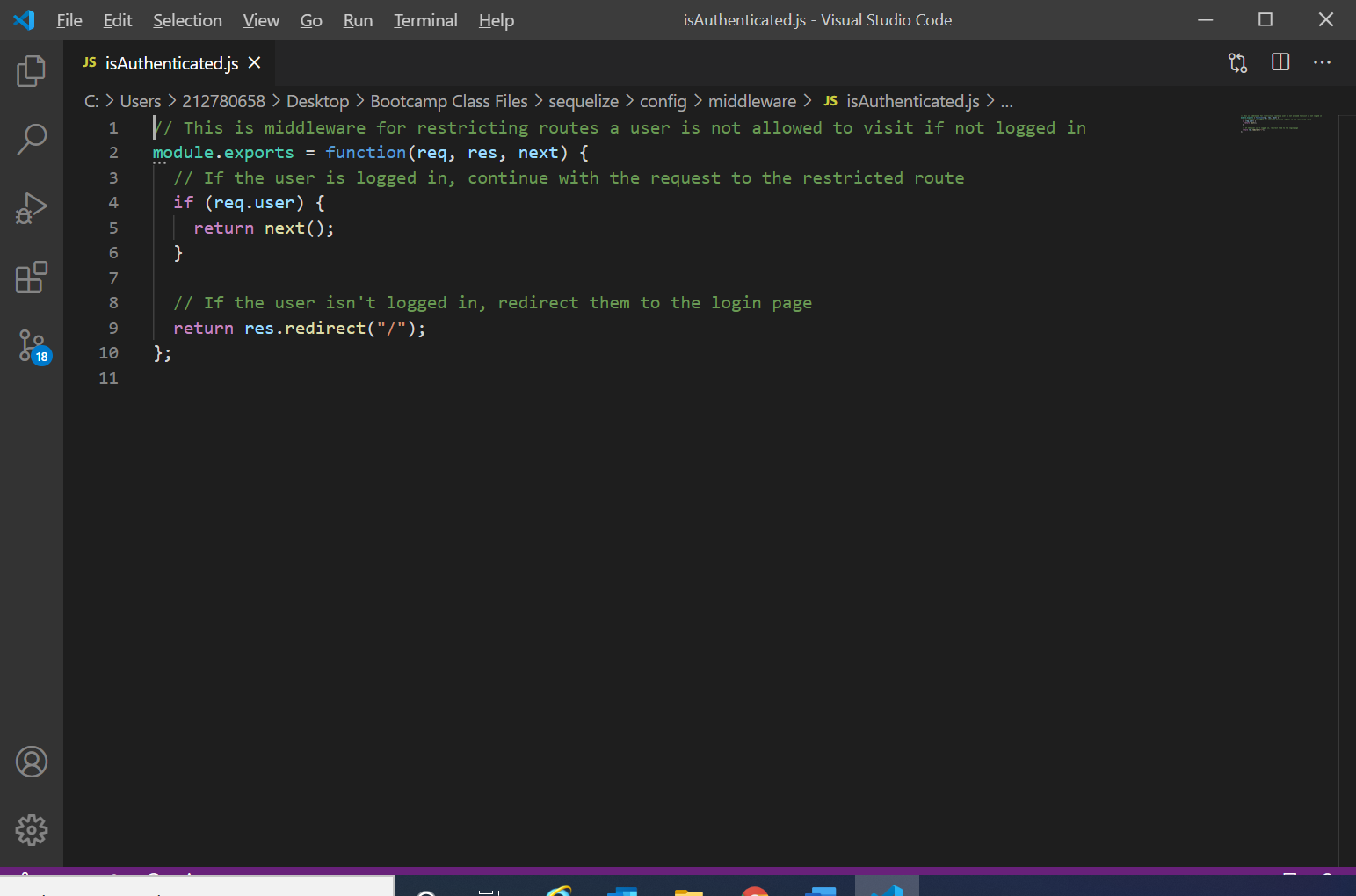
The purpose of this document is to provide a code review of the following files

1. config
   1. middleware
   2. config.json
   3. passport.js
2. models
   1. index.js
   2. user.js
3. public
   1. js
   2. stylesheets
   3. login.html
   4. members.html
   5. signup.html
4. routes
   1. api-routes.js
   2. html-routes.js
5. package.json
6. server.js

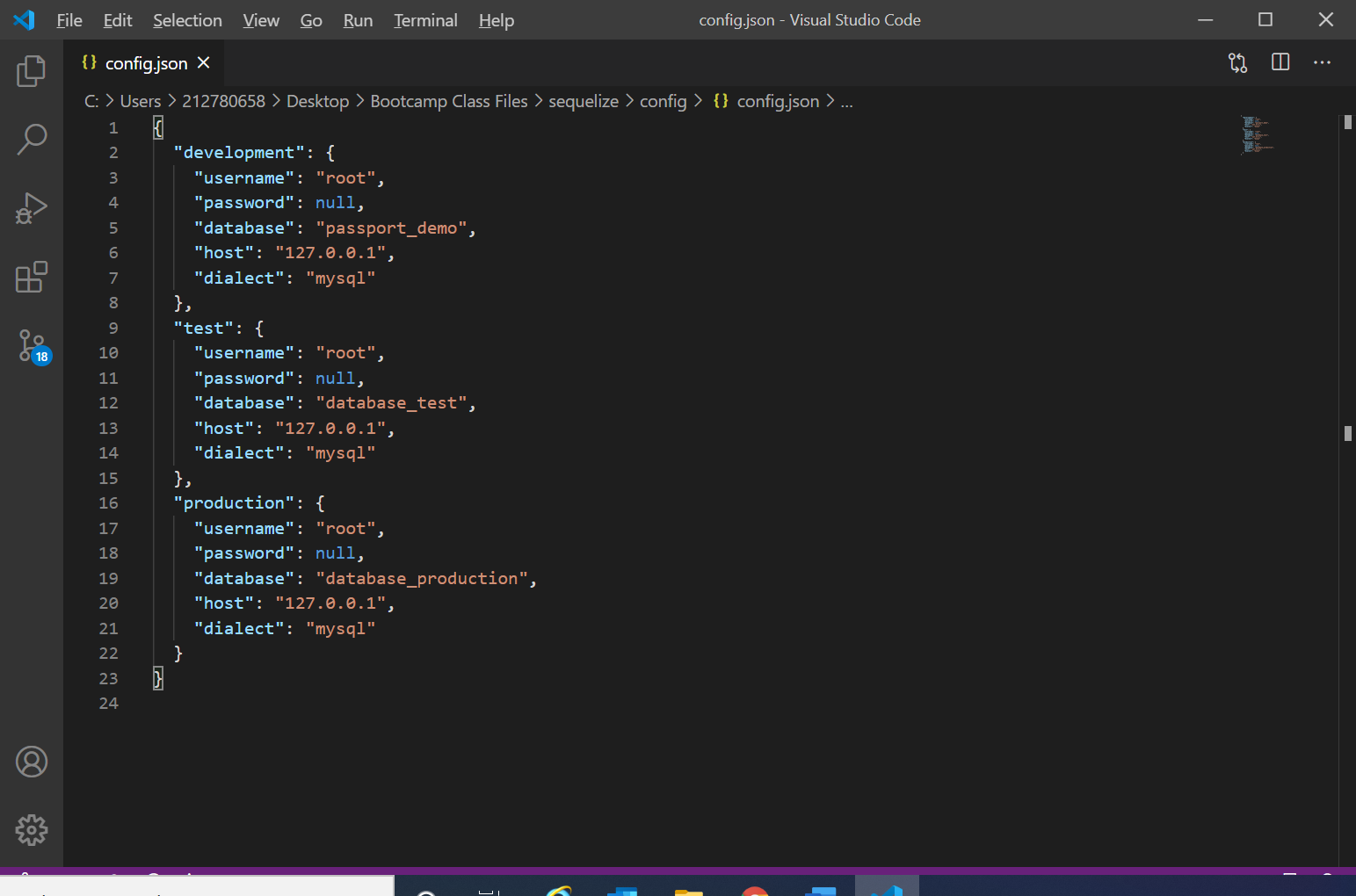
config

* middleware
  + isAuthenticated.js
* config.json
* passport.js

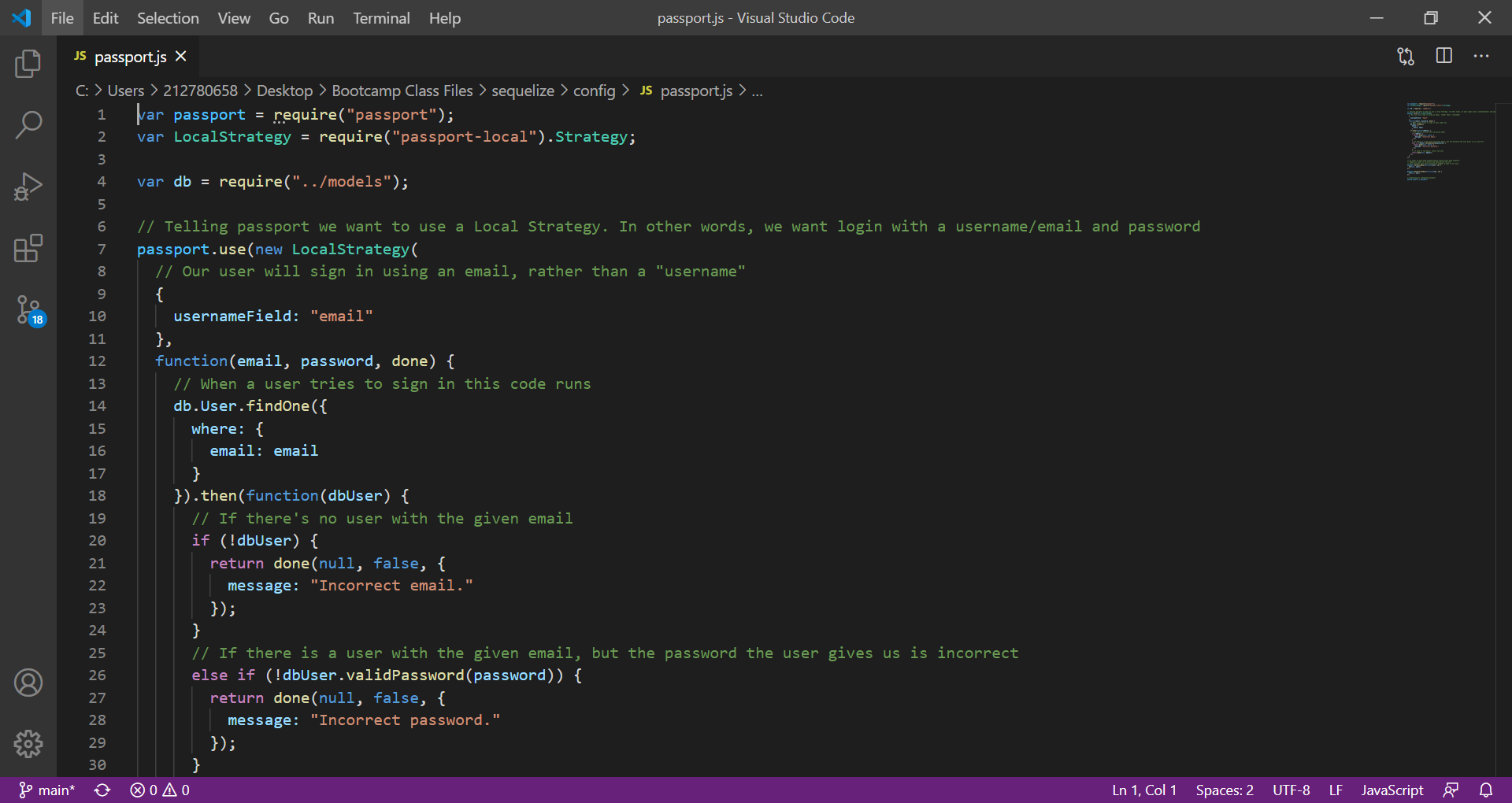
The isAuthenticated.js middleware file restricts the routes the user can visit if they are not logged in. The user will be redirected to the login page if they are not logged in. If the user is logged in, they will be directed to the restricted route.



The config.json file is the middleware connection configuration to the server.

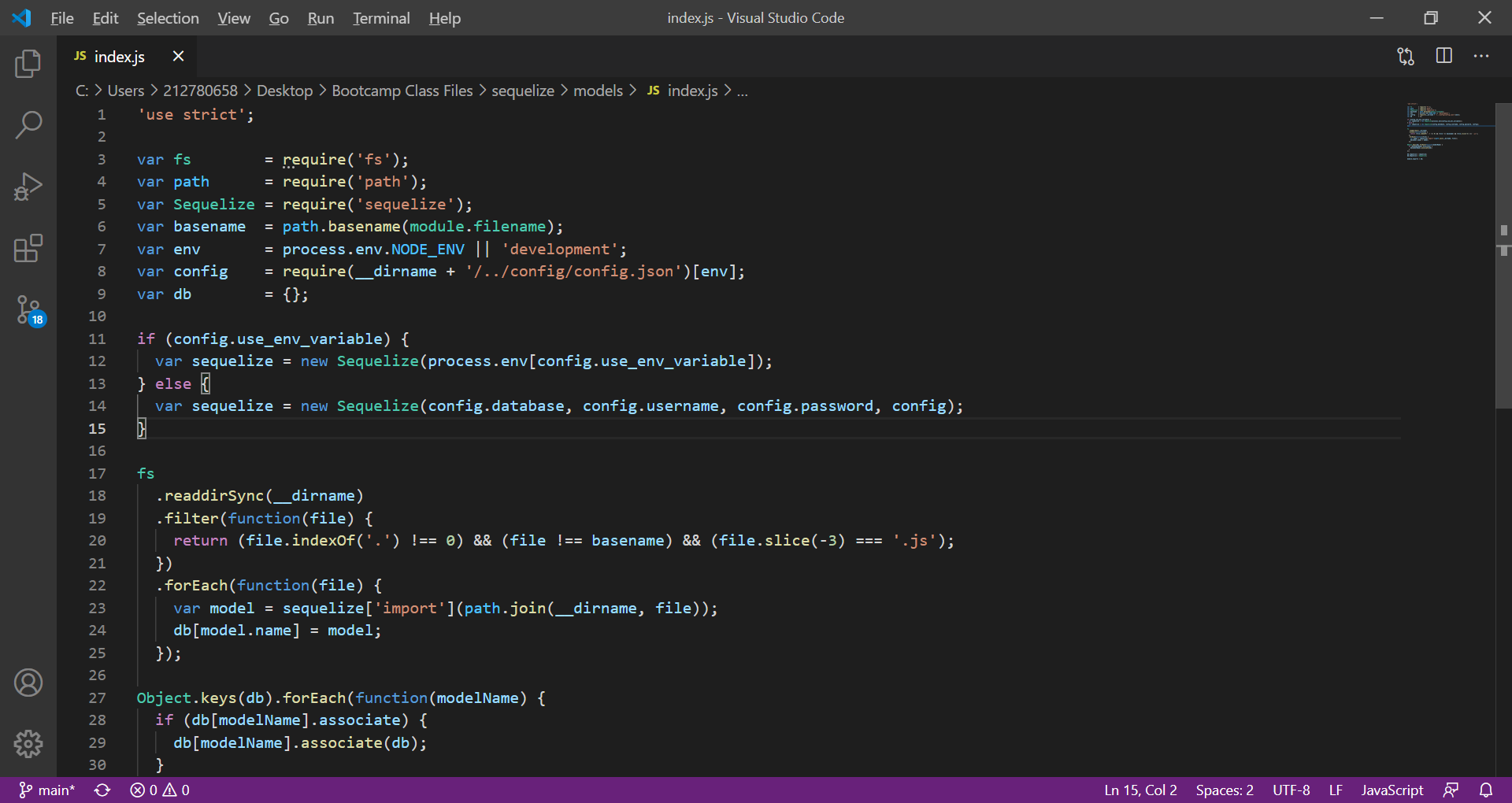


The passport.js file contains javascript that lets passport know we want to login with a username/email and password. This file is dependent on the models folder.

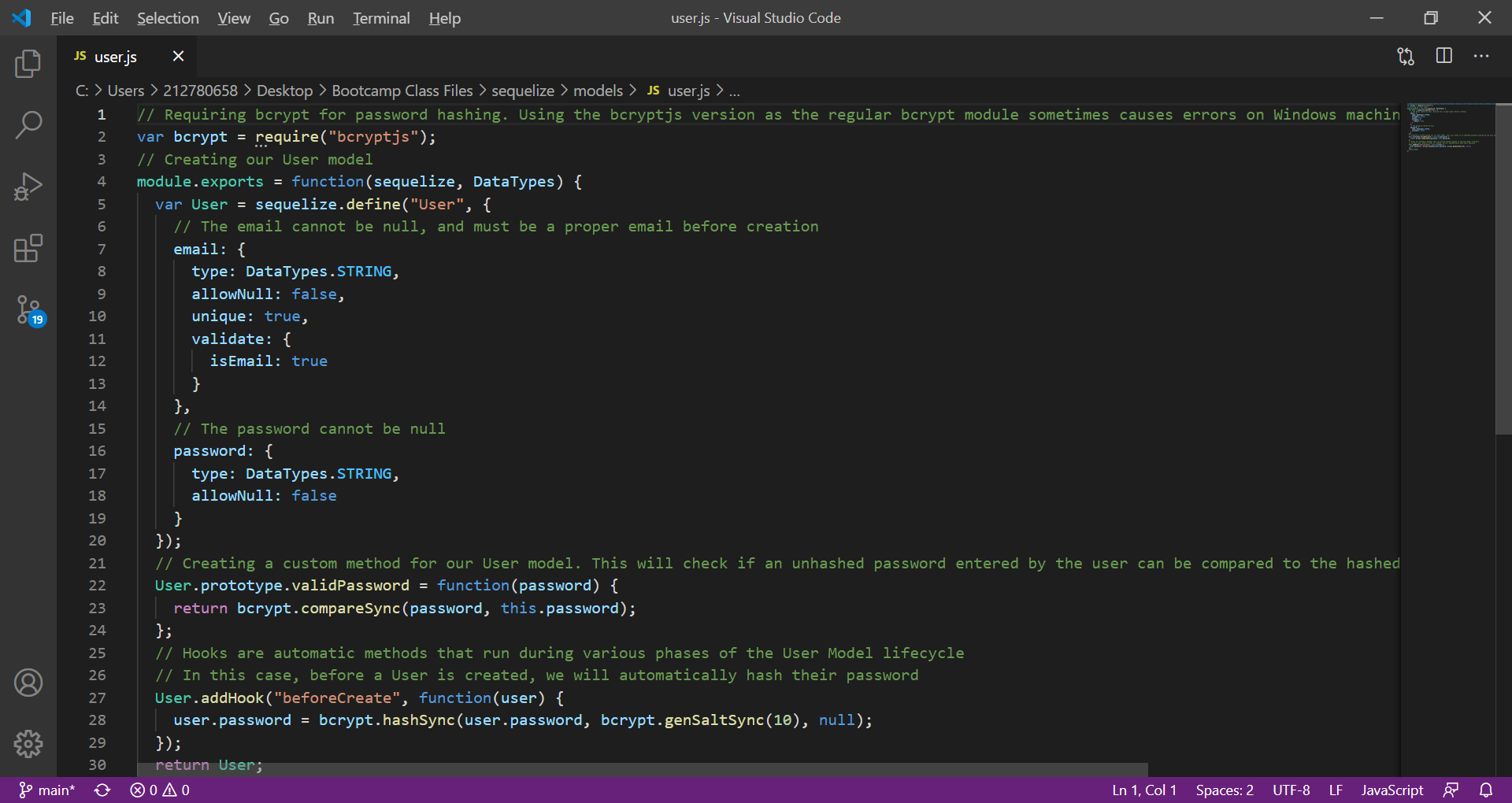


models

The index.js provides the database connection and user login information is imported. This file is dependent on the config.json file.

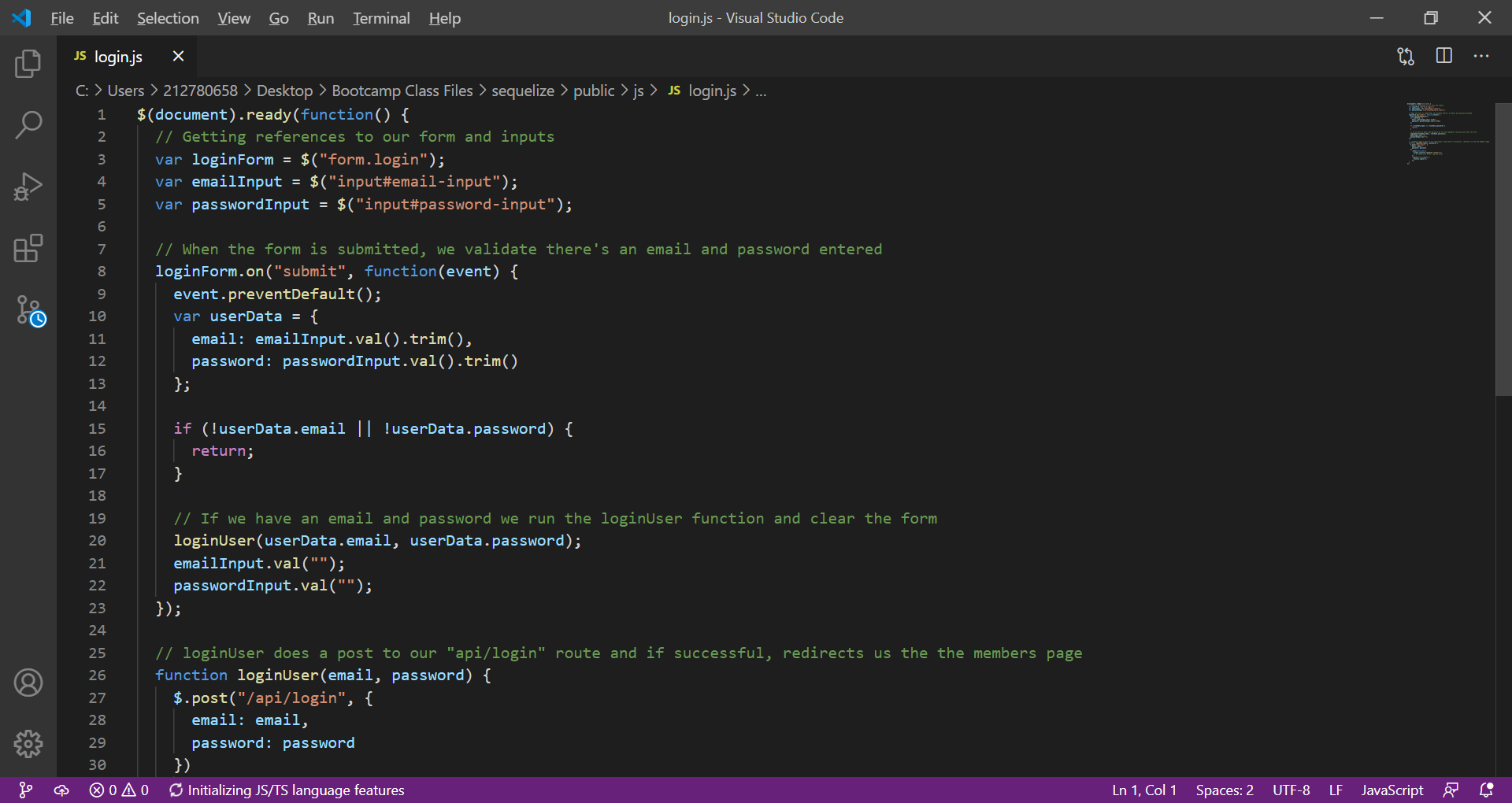


The user.js requires the bcrypt module to perform password hashing. This helps to make the database secure. The javascript defines the database. The user model is created, which consists of a valid email address and a password that cannot be null. A custom method is created to check to see if an un-hashed password was entered by the user. Before the user is created, their password will be automatically hashed.

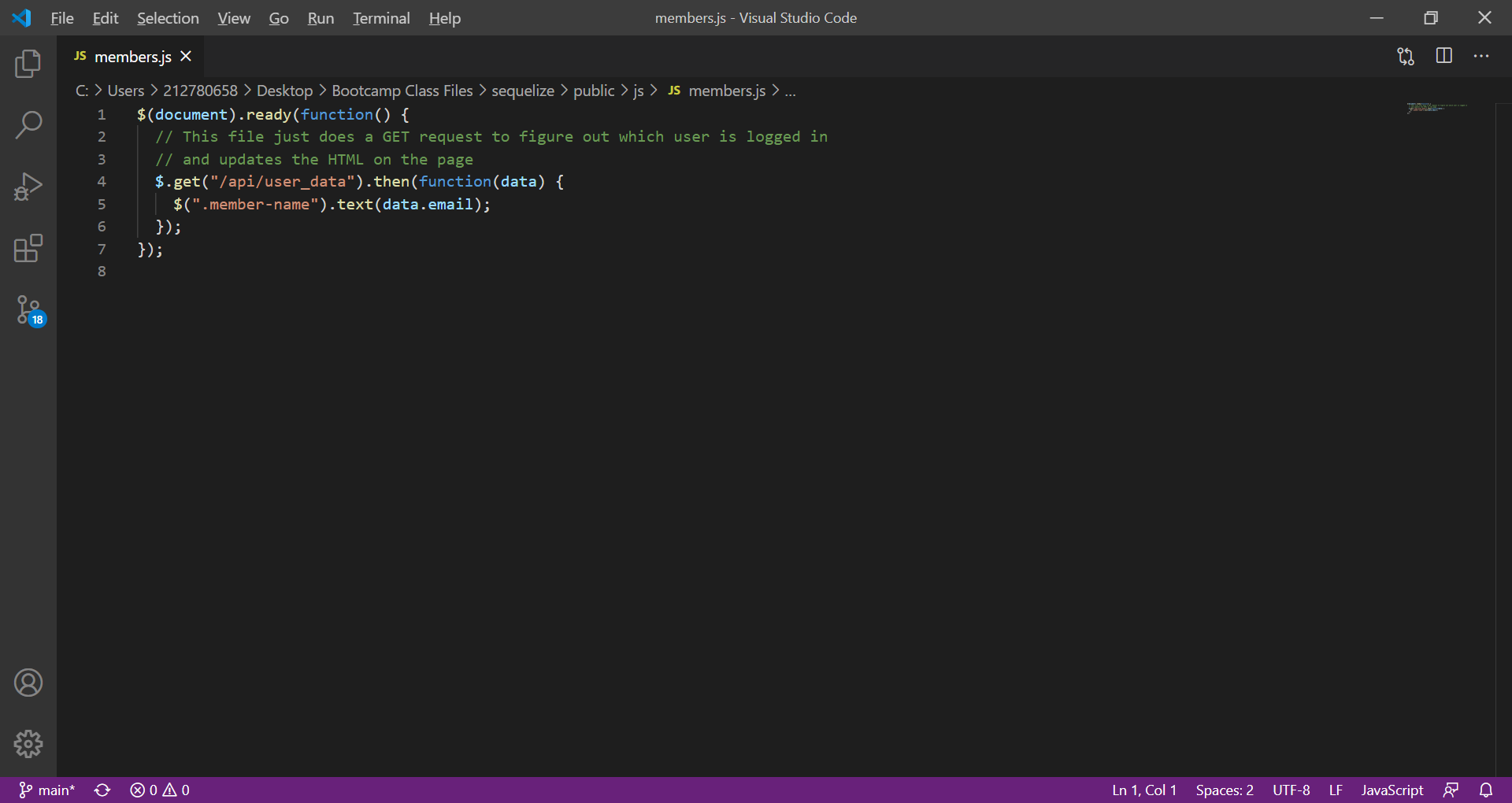


public

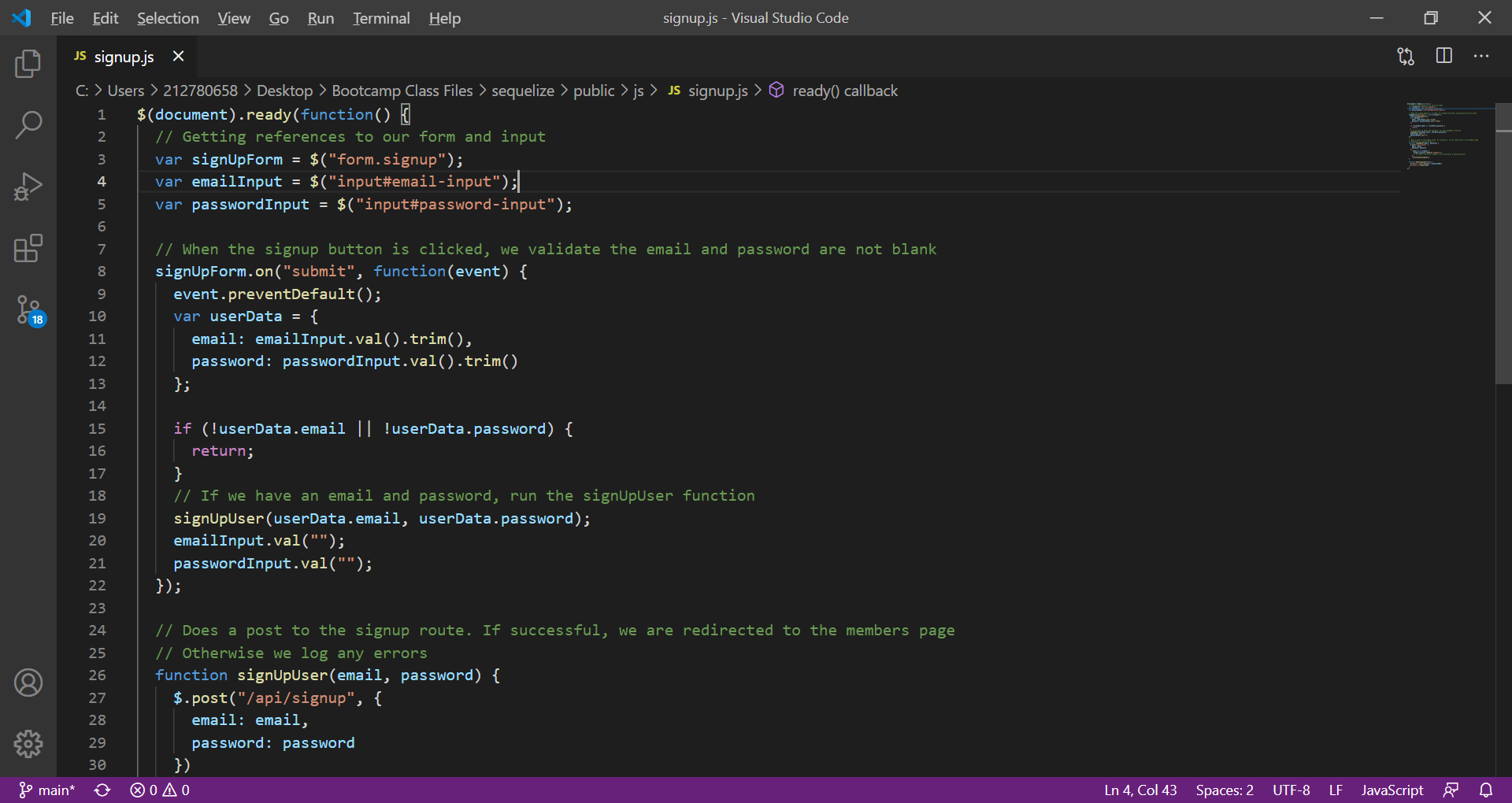
login.js provides the login form which requires the user’s email address and password to be provided. The sign up link is also provided.



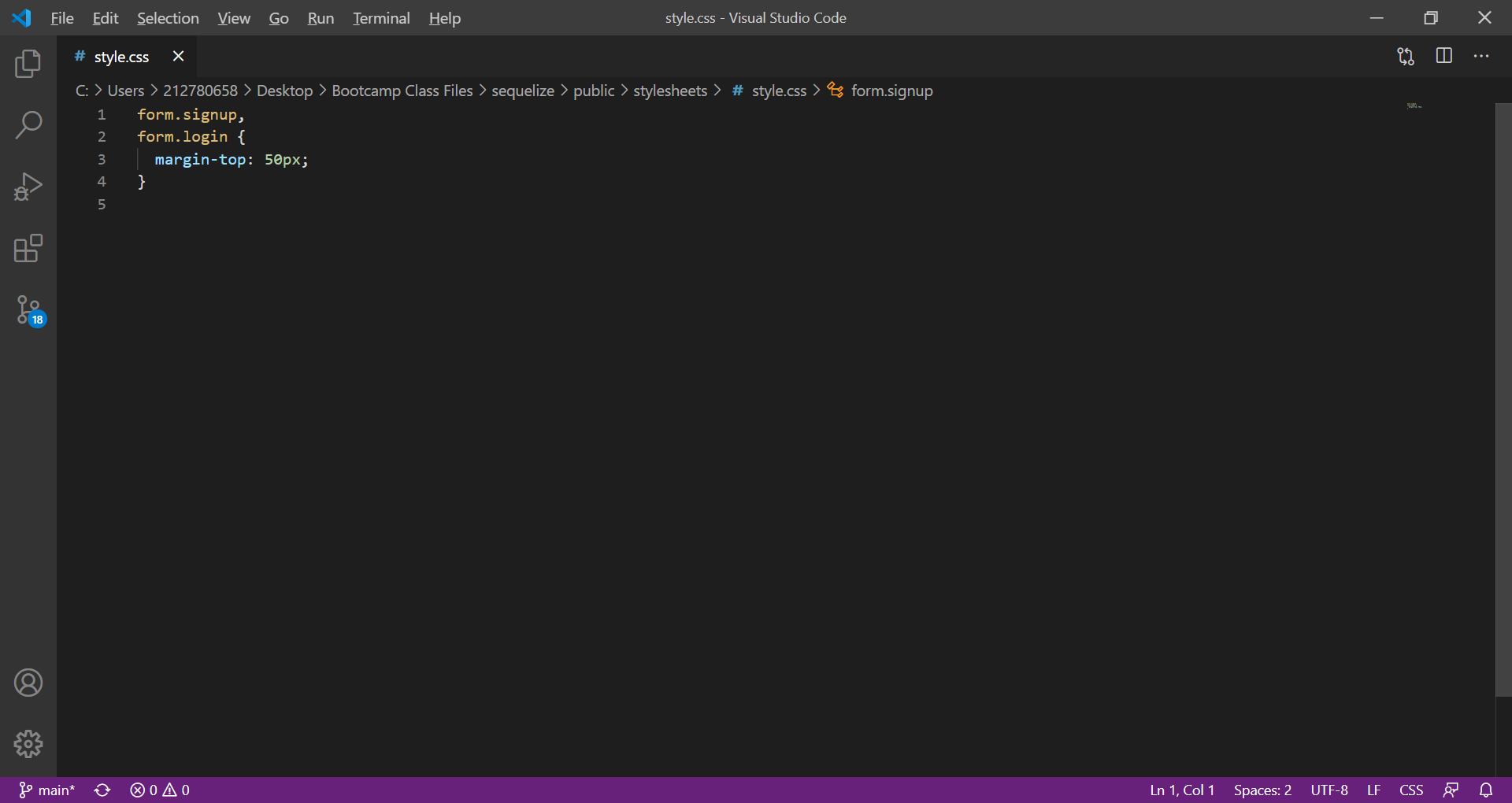
members.js file performs the GET request to determine the user that is logged in and updates the HTML displayed on the page.



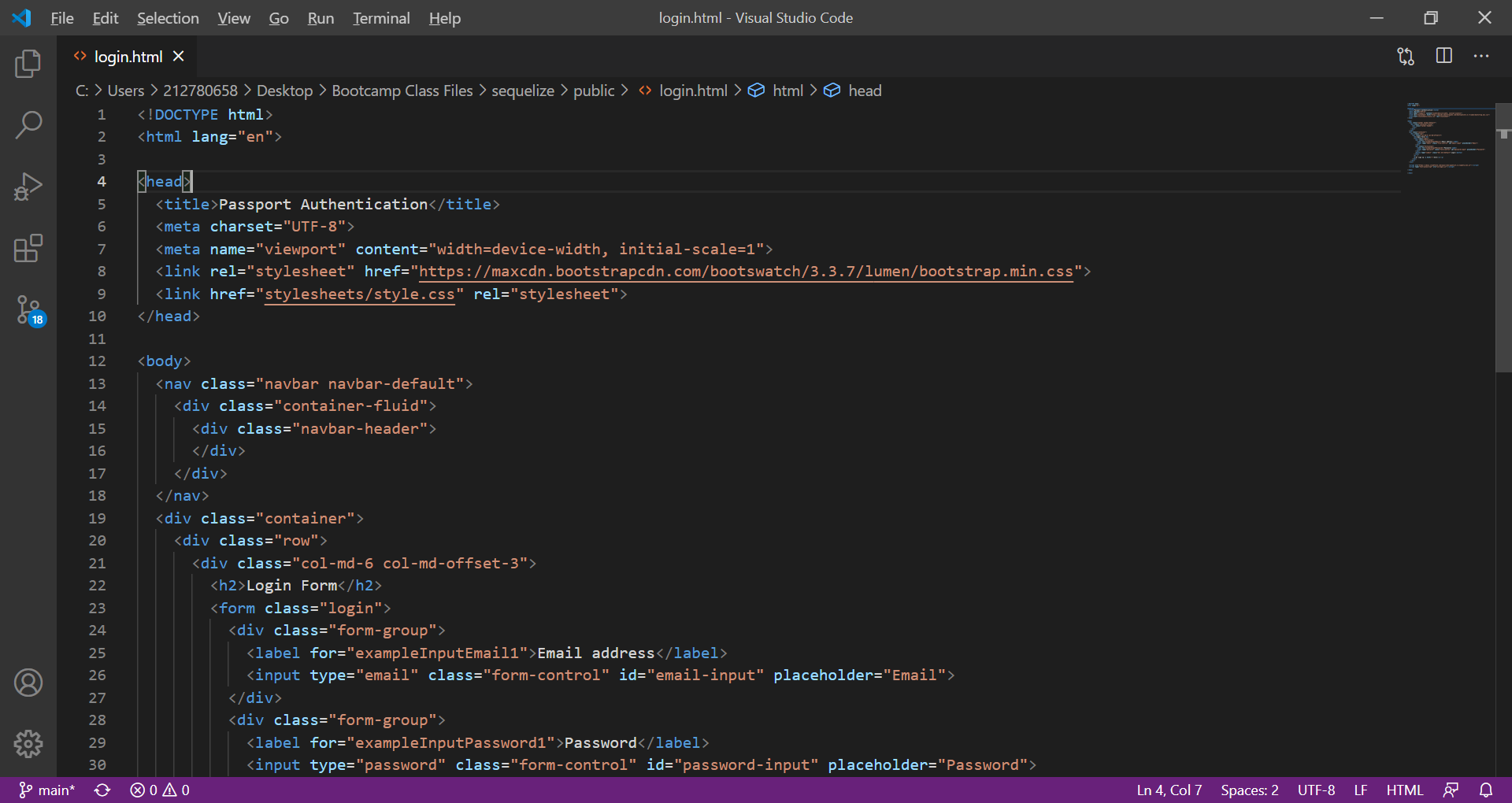
signup.js file gets the references to the form and input. The email address and password will be validated to make sure they are not blank when the sign up button is clicked. If the email and password are present, the signUpUser function will run. It then will do a post to the signup route. If it is successful, the user will be redirected to the members page.



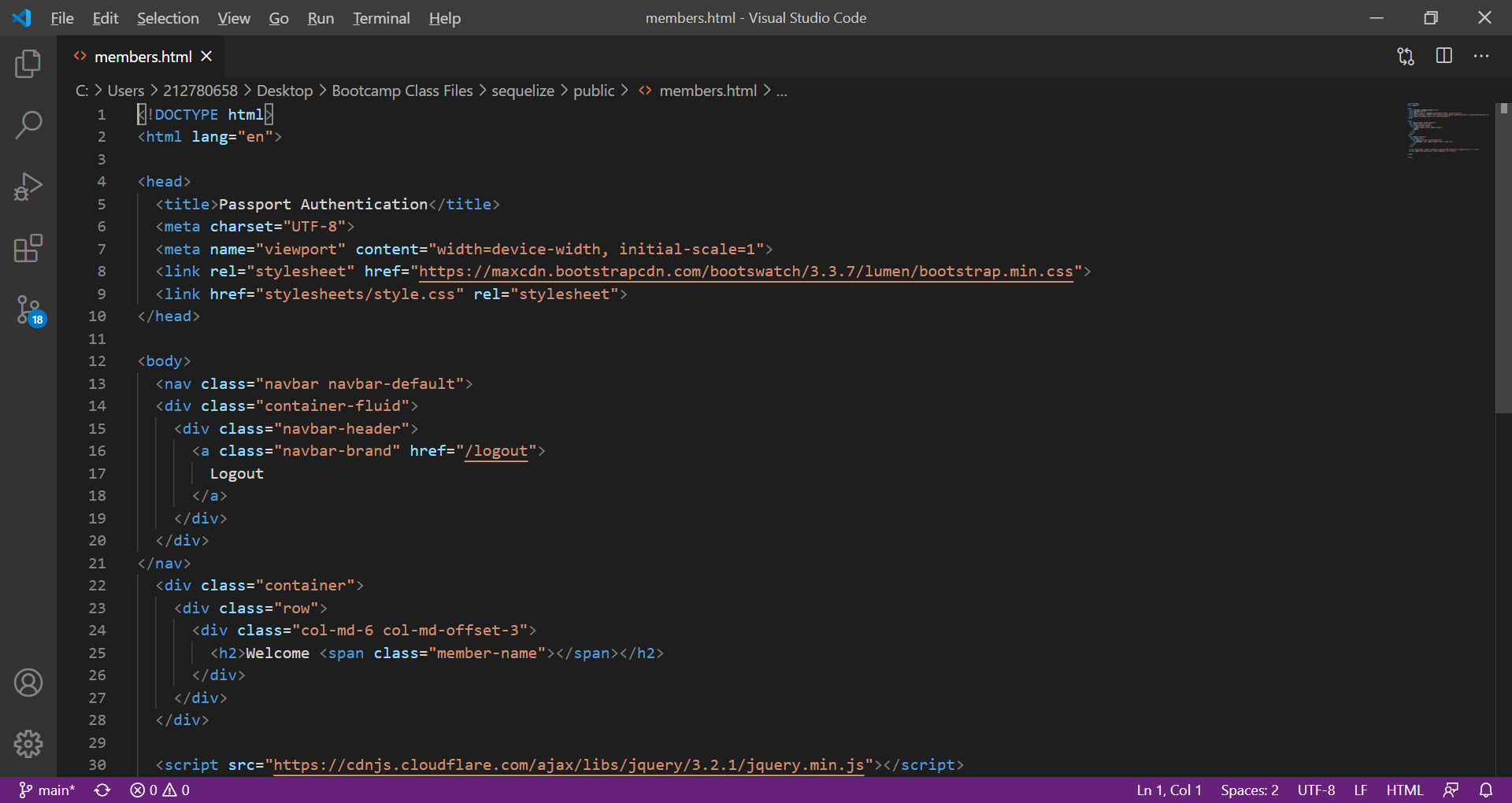
The stylesheet defines the margin top to 50 px for the signup and login forms..



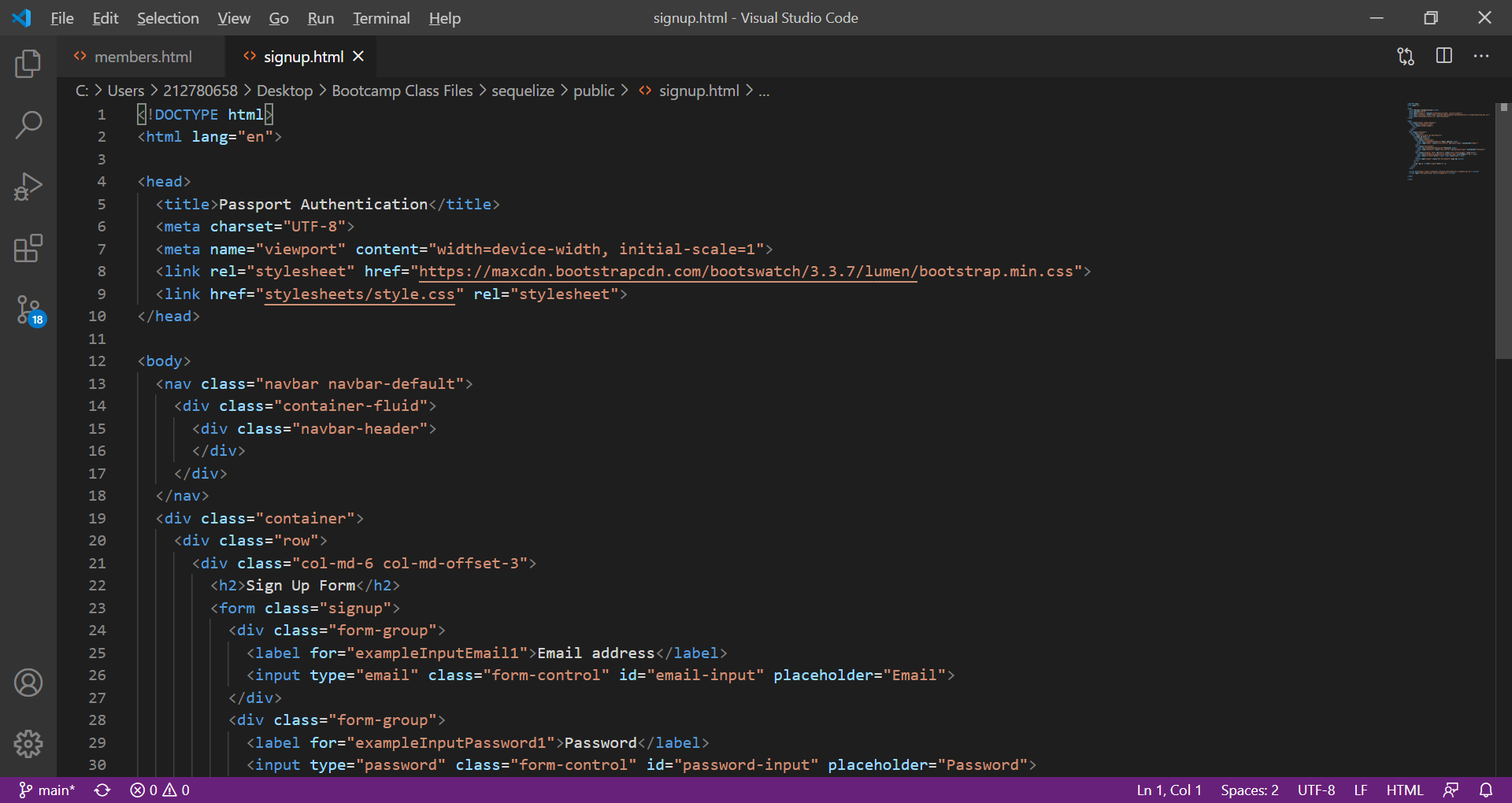
login.html uses bootstrap and the stylesheet. It requests the email address, password to login. If the user doesn’t have this information, they can sign-up for an account.



members.html displays the welcome message and the logout button in the nav bar. This file is dependent on the stylesheet.

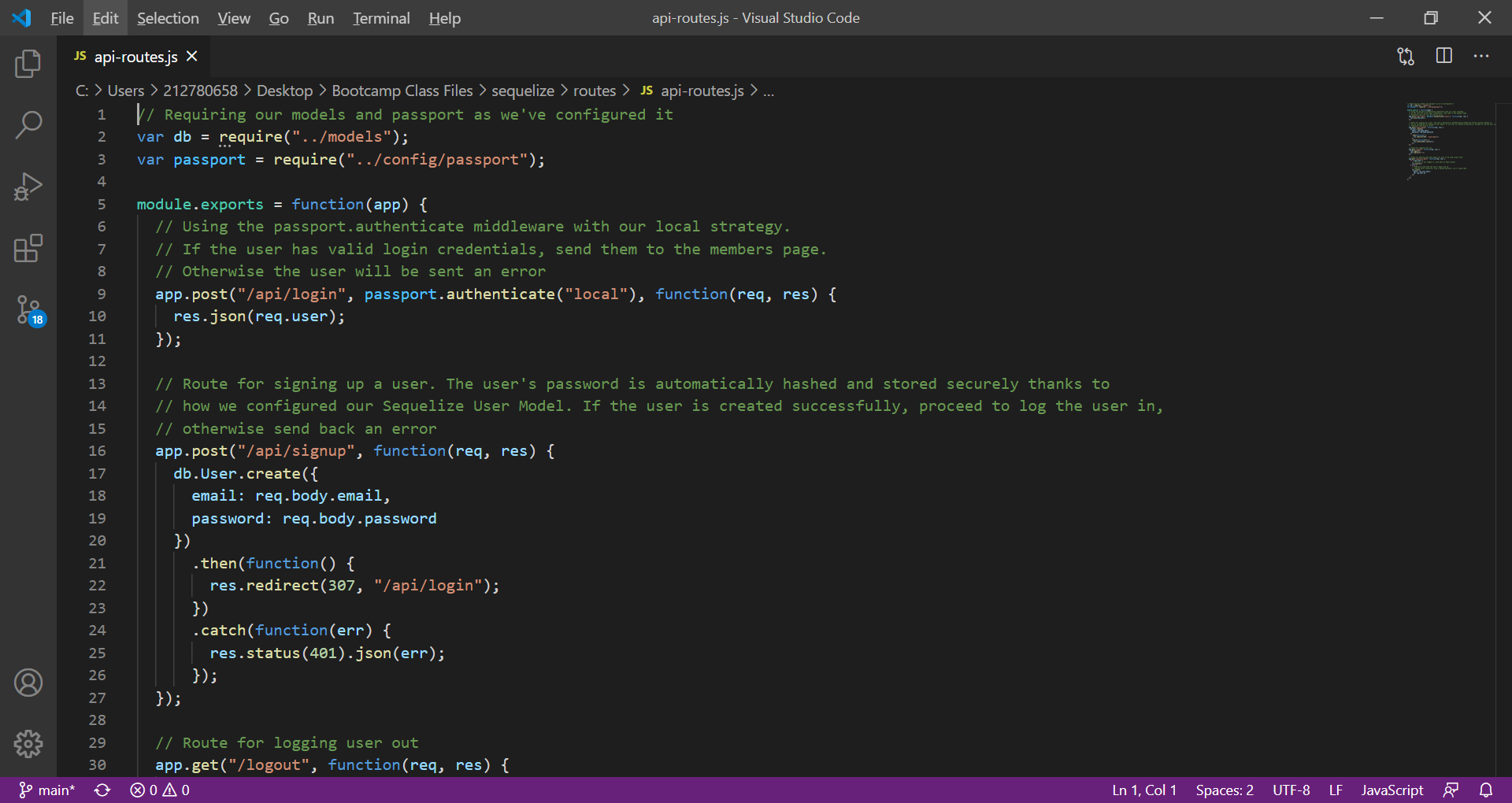


signup.html allows the user to register by providing the email address and password. There is also the option to login if they already have an account. This file is dependent on the stylesheet.

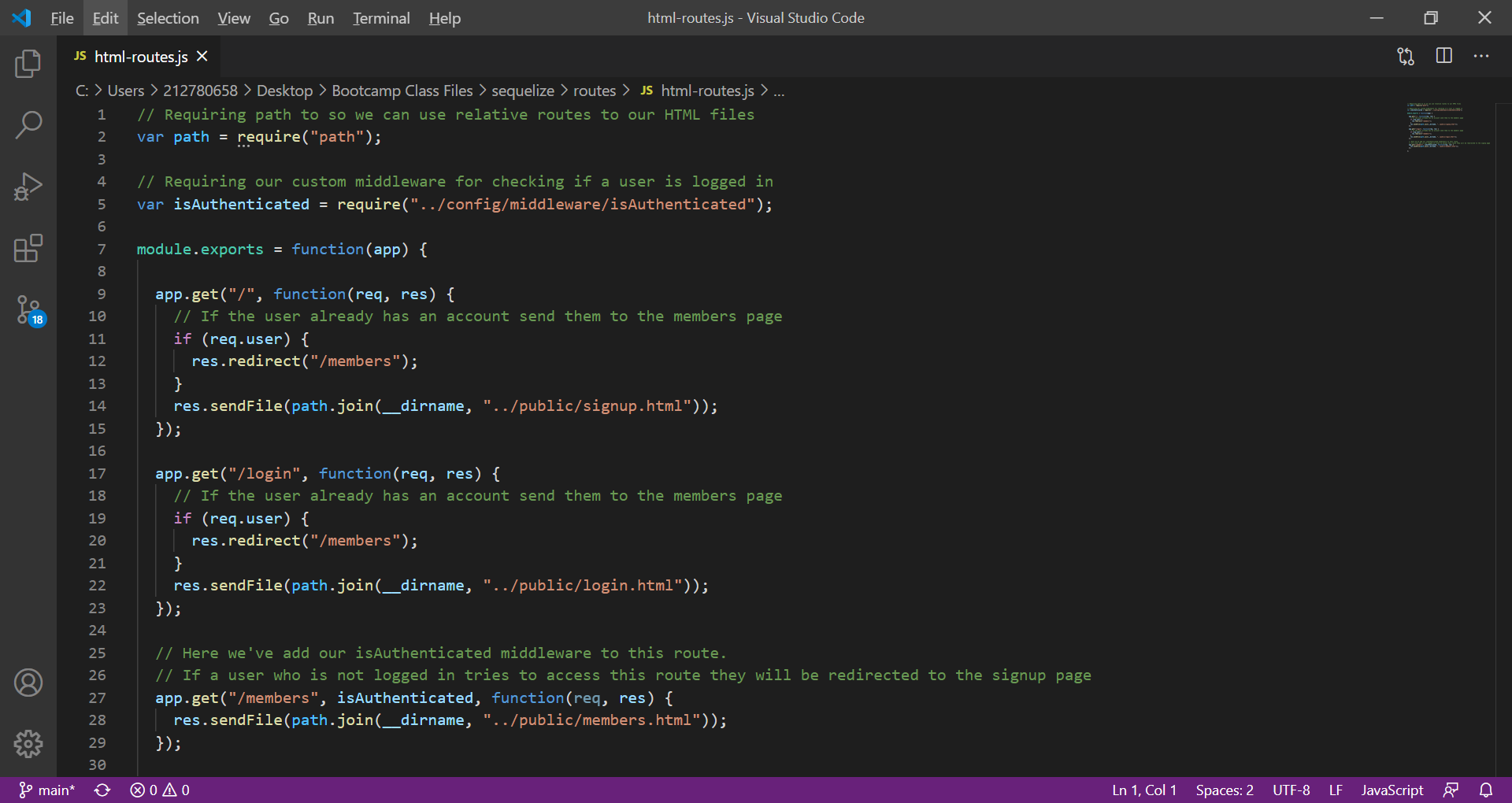


routes

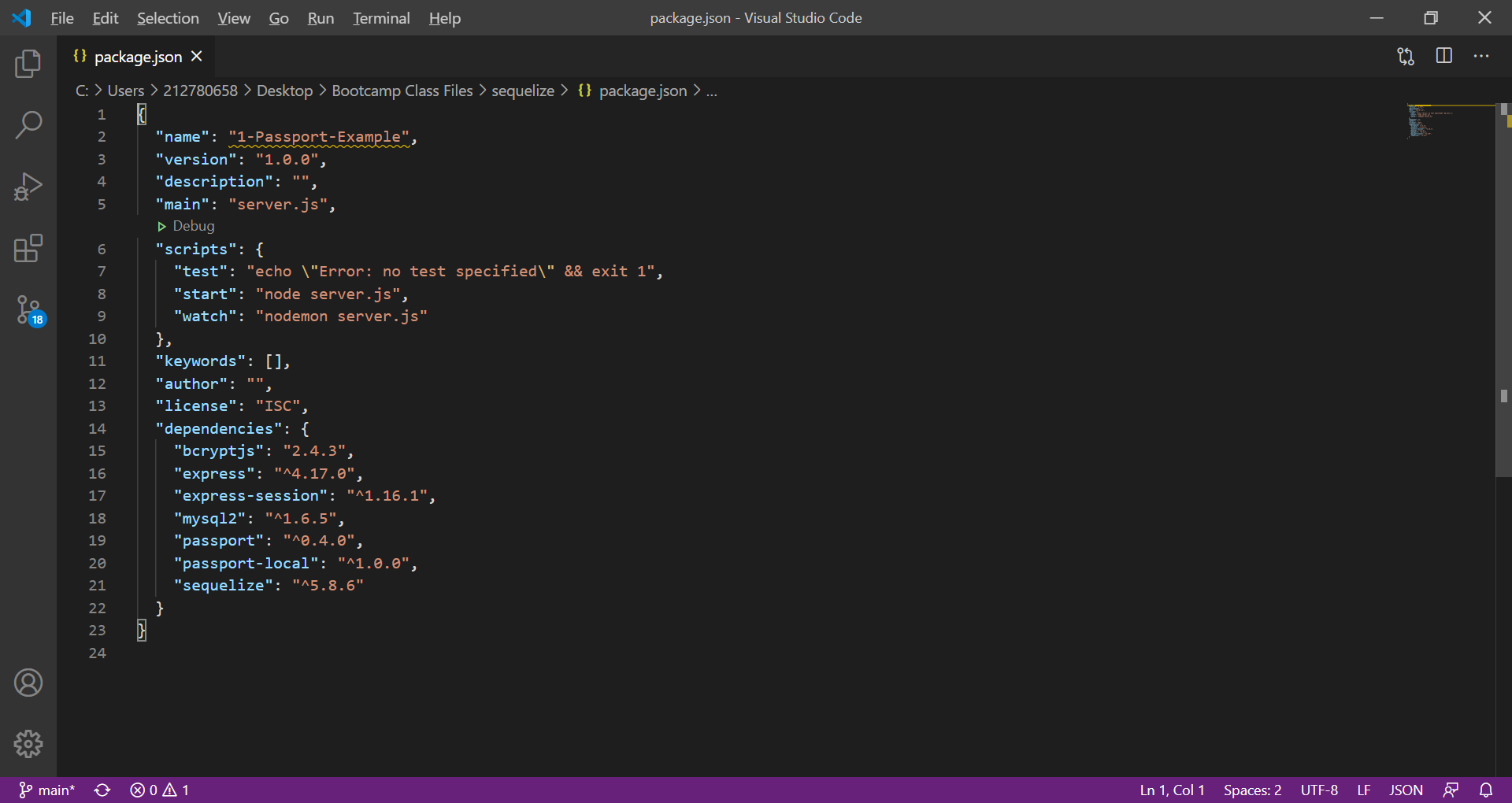
api-routes.js file contains the routes for signing in, logging out, and retrieving user data to be displayed on the browser. This file is dependent on the passport file in the config folder.



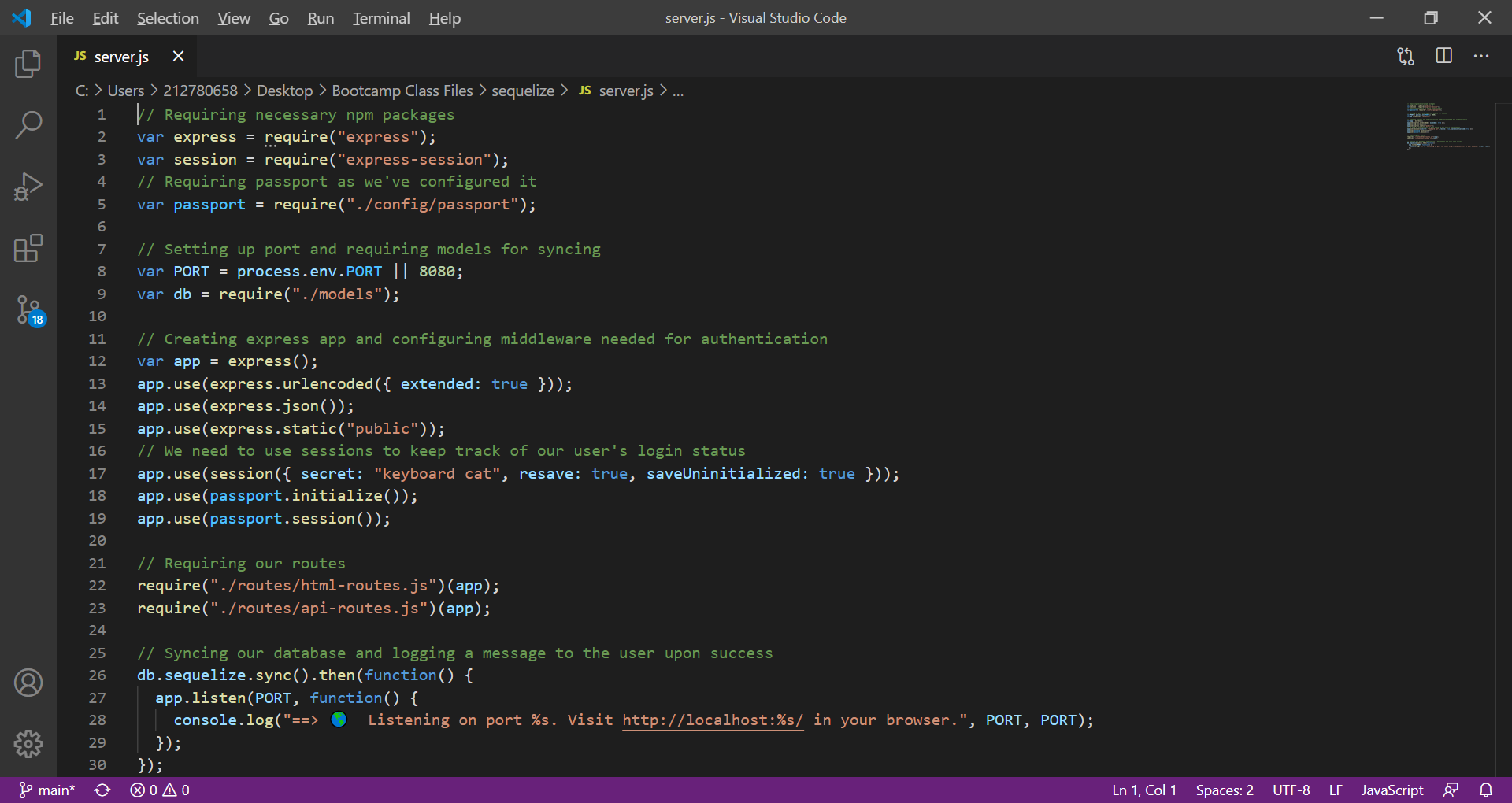
html-routes.js checks to see if the user is signed in. If they are not, it will redirect them to setup an account. This file is dependent on the signup file in the public folder.



package.json contains all package information, the node modules that will be used, version, description and other details about the package.



server.js requires packages, sets up PORT, creates express and middleware This file also creates the routes and syncs the database and logs messages in the terminal if the connection to the server is successful. This file is dependent on the passport file in the config folder. It also depends on the html and api routes in the routes folder.



**Project Improvements**

I have identified the following project improvements that can be implemented to update the code:

* Add alerts when a user’s email address already exists in the database during sign-up.
* Provide the user with password complexity requirements above the password text box.
* Provide the user with a warning if the password requirements are not met.
* Provide a user profile section where the user can update their email address and password once logged in.