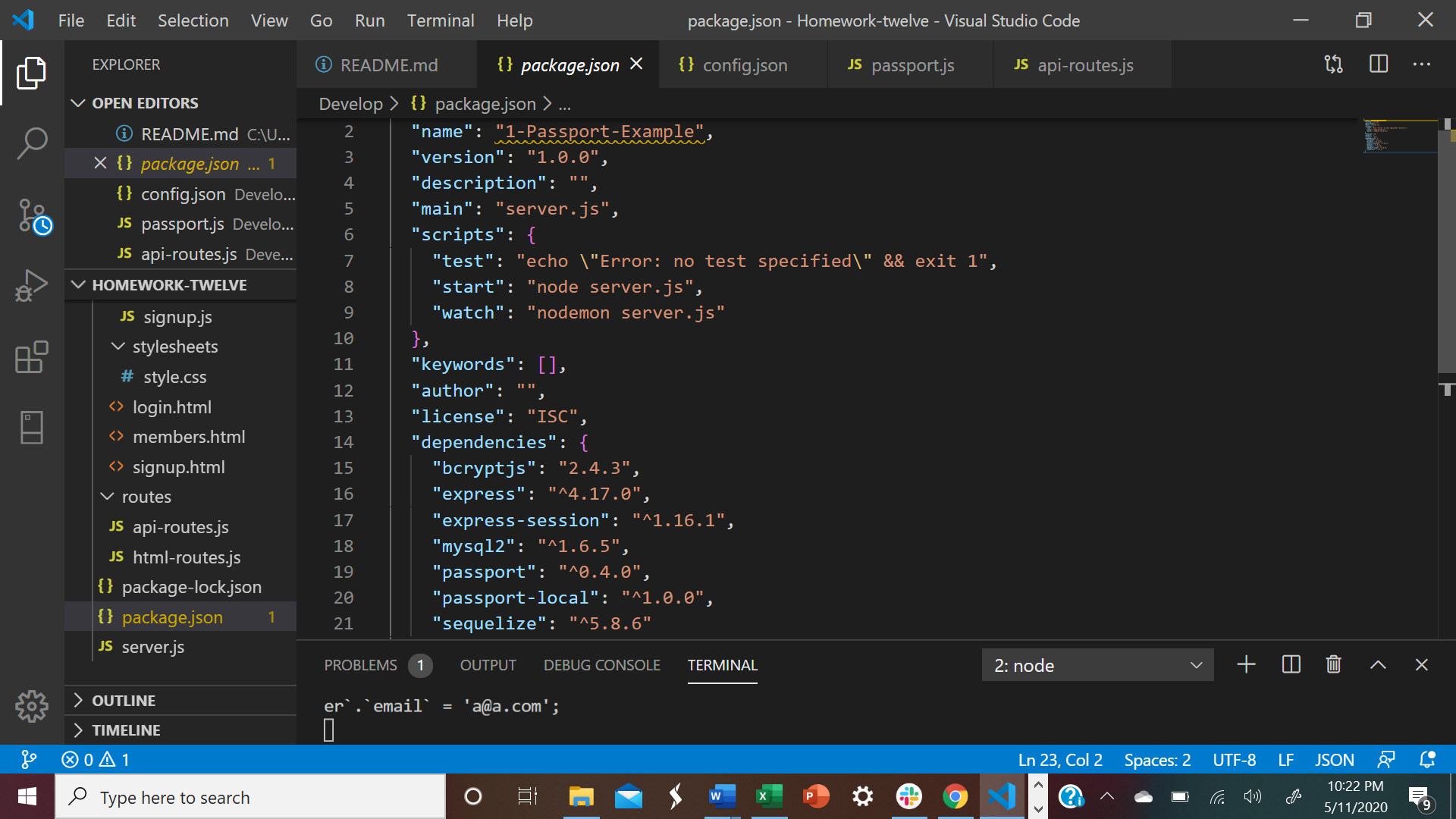
**PART ONE**

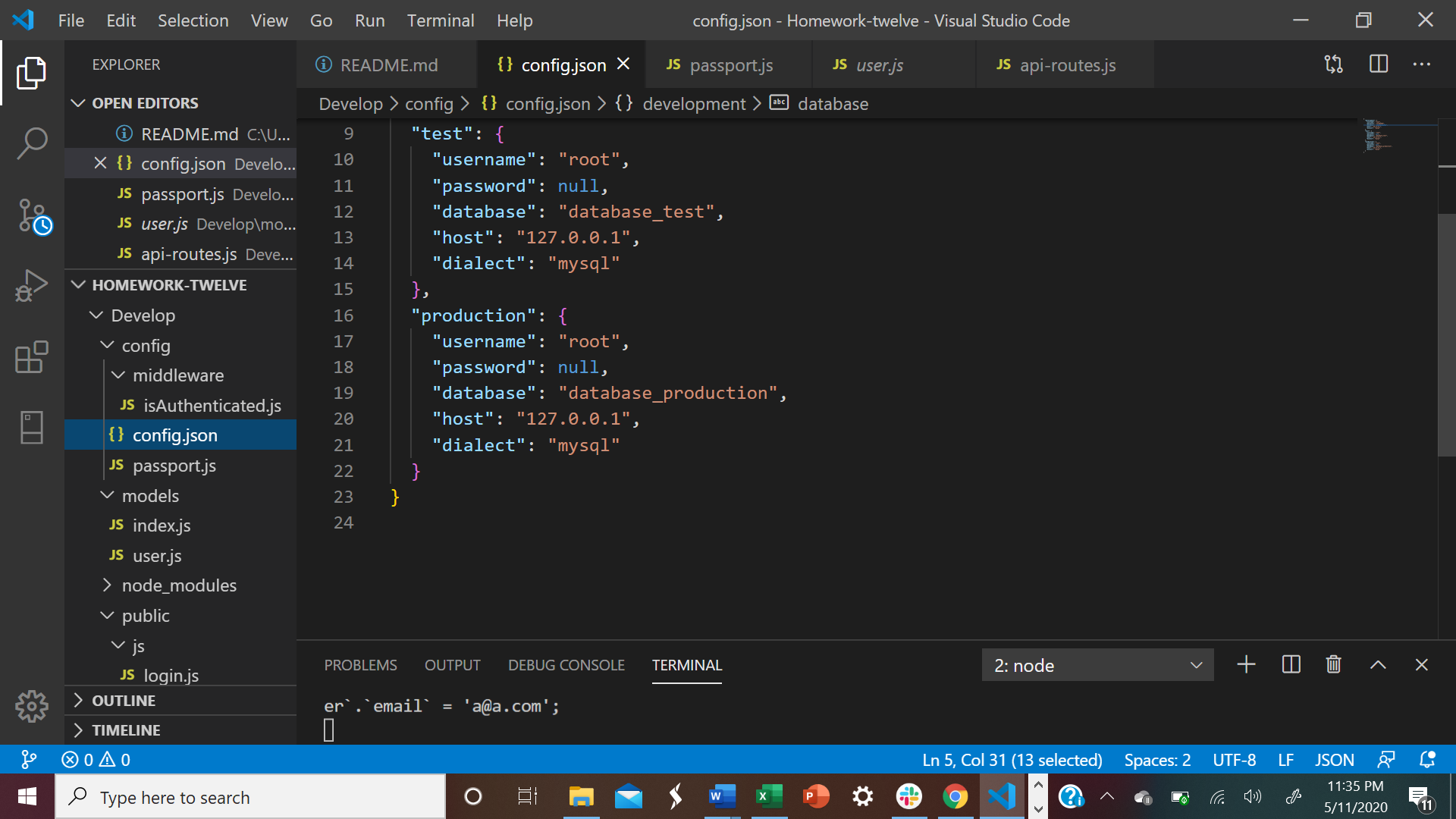
**package.Json-package-Lock.Json- .node\_modules – server.js**

The user can find the **package.json** file in the project root. All npm packages contain this file. This file holds various metadata relevant to the project. This file is used to give information to npm that allows it to identify the project as well as handle the project's dependencies. The name field is the name of the project, in this case “1-Passport-Example. The version field is used by npm to make sure that the right version of the package is being installed. The main field is the main entry point for the library, which in this assignment is the **server.js** file. The dependencies field is the list of all the dependencies of the project that are available on npm-so when the user runs require(<library name>), require resolves this call to require (<package.json:main>). The dependencies field is the list of dependencies in the project that are available on npm. As a user, you first want to install your dependencies in the command line by running “npm install.” All the dependencies will be installed in your program after running this command. By running this command in the root directory of the project, it will install all the dependencies to **./node\_modules** directory which contain all of our build tools. In this case, we have the following dependencies that will need to be installed:

* Bcrypt.js – a password hashing function that allows proper password encryption
* Express - minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.
* Express-session – the express session module stores session ID in a cookie and session data on the server
* Mysql2 – offers better perfromance over node-mysql
* ****Passport – an express-compatible authentication middleware for Node. It is designed to authenticate requests.
* Passport-local - session based authentication is at the root of the passport-local strategy. This method of authentication is “server-side”, which means our Express application and database work together to keep the current authentication status of each user that visits our application.
* Seqeulize – a powerful library in javascript that makes it easy to manage a SQL database

Our **package-lock.json** is automatically generated for any operations where npm modifies either the node\_modules tree or package.json. This file keeps track of the exact version of every package that is installed so that the project is producible in the same way even if packages are updated by their maintainers.

Once the dependencies are installed, we run the ‘sequelize init:config and sequelize init:models’ in the command line while at the root of our project directory. This creates the following new files and folders for us:

* config/config.json
  + the ‘config.json’ stores JSON with details about the environment we’ll be running our database in. The name of our database is passport\_demo. The user must make sure to alter the contents of the “development” object to match their local MYSQL database to ultimately successfully make a connection to our database.
* models/index.js
  + The index.js file is the directory where we will define all of our sequelize models. First it figures out which database it should use based on whether we’re deployed to heroku (“production”) or running locally (“development”) and will use the appropriate configuration inside config.json. We can also optionally specify a database to be used for testing if we want. Then it goes through every other JavaScript file inside our models folder and runs them through Sequelize. It gives our models all of Sequelize’s helper methods and makes sure that all of the associations between models are properly set up. It exports an object we will use to interface with Sequelize in our other files.

**User.js**

In this file, we create a model for a User. It contains the code to create a Sequelize model where we export a function that takes in 2 variables – sequelize and DataTypes- which are provided to us by index.js. “sequelize: is our connection to our database. DataTypes are used to define the type of data each property on our model should be. Inside of our function we run the “sequlize.define” method – passing it two arguments. The name of our model, in this case “User”, is a string and then the object describinh our model’s schema. Each property represents a column in the database – the columns being email and passowrd. It is important to note that sequelize by default will give us an auto-incramenting primary-key id, an updatedAt and a createdAt column.

**Server.js**

This file is the initial starting point for the Node/Express server. Here we require our entire models folder – index.js and user.js. The sequlize property on the db object is our connection to our passport\_demo database. The built in ‘sync’ method in sequlize creates our ‘users’ table using our models. We start our express server after our database use sync’ed.

Let’s take a look at the functionality of our application.

Our default page or root route is the signup.html page as indicated in our html routes. When the user inputs data on the sign-up.html page. The user’s password is then encrypted using passport authentication. The user is then redirected to the members.html route. When the user inputs and submits an email and password on the front-end signup.html page on the client side, we call the signUpUser function in the signup.js script- passing in the email and the password. When the submit button is clicked on the signup.html, we submit a post request and the email and password are validated using the script from passport.js. We make a connection to our model user.js, creating the email and password into our database using the sequelize create function as found in our api-routes, returning the object dbUser, which is the data in our database. Once we create an account, we are directed to our members html route, where our email appears on the front-end.

Once the user account is created the user will be directed to login on the login.html page if they have not logged in. Once logged in, the user will go to members.html.

If the user is not logged in, we are redirected to the login.html page. The user will enter their email and password, as indicated in login.js. The function loginUser will call the post request, passing in user input through passport authentication as indicated in line 9 of our api post route in our api-routes.js. Passport (as used in passport js) takes in the email and password, makes a communication to our database and verifies the user’s email and password.

If the user’s email and password are valid, our user’s data is returned. We return to our api routes where on line 10 we will send back a json of user’s data. We are then directed to login.js, where we are directed to the members.html. Since we are a valid user/isAutheticated we will be redirected to the members.html or redirected to the signup page.

When the user logs outs, they are redirected to the default root which is the signup.html.