**Implementing Login with Node.js**

**Bradley Dinger & Benjamin Rittenhouse**

Code: <https://github.com/Braddinger13/CIS4282_LoginTutorial.git>

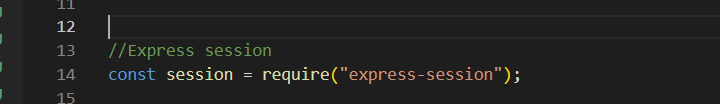
We will now be building login functionality on top of our full CRUD page. This will be a pretty simple implementation as it will have login/logout pages, as well as a view profile page.

1. **Install backend dependency**

In order for us to create a login page that stays persistent throughout our app, we must install **express-session.** This dependency will allow us to persist the user in our express app utilizing the session object.

cd into your server folder and enter “**npm i express-session**”. You will see it appear in your package.json file.

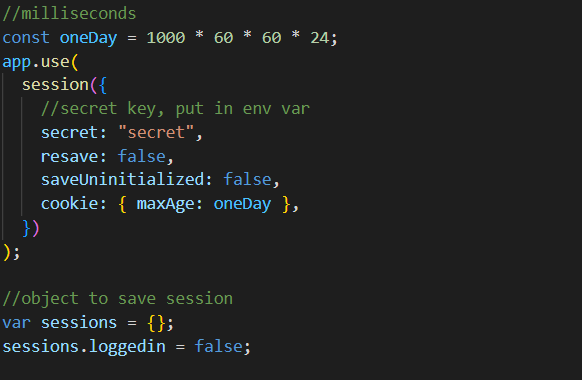
Now, add this import in your **server.js** file like so,



1. **Use session in server.js**

In order for us to access this session, we need to tell the app.

Below is how I added this session and told the app to use it.



This session will persist 1 day if it is not terminated, even if you close the page and reopen it.

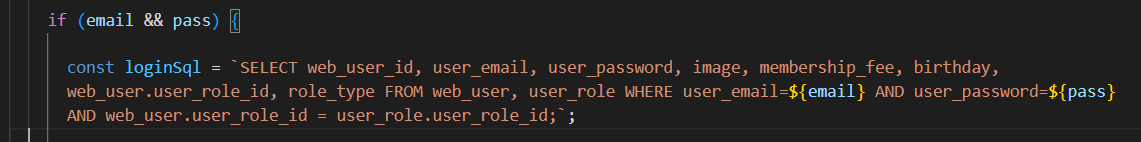
* Entering the secret key into an environment variable would be a good idea.
* The resave parameter, if set to true, will save the session even if it is not modified, you can set this to false.
* The saveUninitialized parameter, if set to true, will save a newly created session even if you never used/changed it. This can also be set to false.
* The cookie parameter can take other variables in that object form, but for now just setting the expiration value, or **maxAge**, is fine.

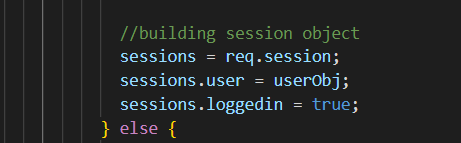
I then create an empty object called **sessions**, and notice the spelling difference from **session,** and initialized it’s loggedin value to false. We will use this object in our api calls.

1. **Login API**

The login api call is long, so I’m going to refer you to the code. Here are the important things to note

* You should add these API calls to the top-level **server.js** file for persistence.
* Also notice that this is a **post** call, and not a **get** call. A post call does not use the URL to pass parameters, but carries it in a request body. You can see email and password being accessed through req.body
* The Login api call checks if the user exists in the database by using a select statement that searches for a user with the entered email and password.



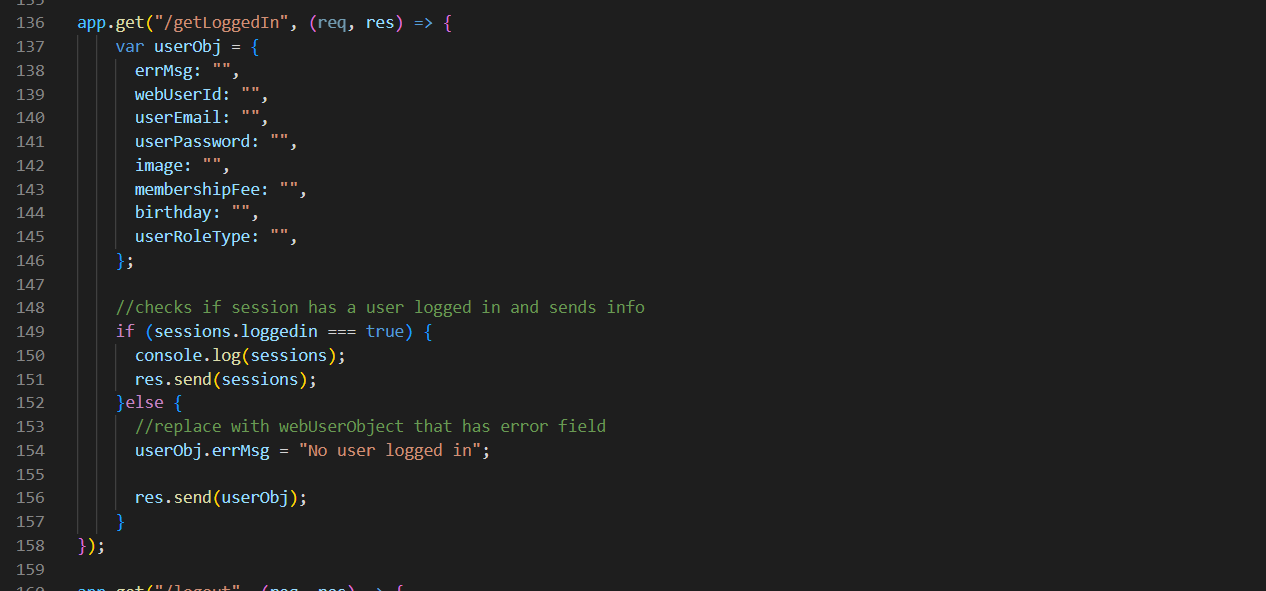
* If the user exists, it will build up a user object given the result. Then we will build the “sessions” object with the session object, user, and setting loggedin to true. 
* If the user does not exist, the user object will just have an error message.
* At the end of the call, the “sessions” object will be passed to the frontend where we will use the user object.

1. **Logout API**

The Logout API is very simple. You simply call destroy() on the session, empty the “sessions” object, and set the loggedin variable to false. A message is sent to the frontend telling it that the session was invalidated.

****

1. **ViewProfile API**

The ViewProfile API is how we can test that the user is persistent. This API checks if the loggedin is true, if so it will return the “sessions” object that holds the user object from the login call.

1. **Tying to frontend**

On the front-end I made 3 simple pages: Login, Logout, and ViewProfile

* Login has input fields for email and password, as well as a button that calls the api.
* Logout simply calls the logout api when the page loads.
* ViewProfile calls the getLoggedIn API and displays the user on screen.

One difference I would like to point out in this code is the use of **axios.** Axios is a dependency that helps with calling API’s. I used this for the post call since it simplifies it. In your client folder, install axios with “**npm i axios**”. Here is how I am calling Login on the frontend:

This gets called when you click the submit button.

As for the other two pages (logout and view), they simply call their respective API calls upon loading the page using useEffect.

Finally, upon logging in with the correct information of a user, you should be able to go to your ViewProfile link and that users info will appear. This information should persist even if you switch routes or exit the page. The only way to change this user is to logout (terminate the session) or login with a different user, which will overwrite this user's data. Here is a basic implementation for reference:

