Abhash Sharma
CS290 Spring 2020
May 17, 2020

Activity 6: Practice

To practice all the topics from this week and last week, I tried to write and run the "toDo" list and the "httpRequest" examples from the lectures. My goal was to purposefully break this or that portion of the code so I could understand the effect those specific lines of code have on the overall application.

The first error I ran into was "EADDRINUSE". This meant that the port I was using was already in use. So, I went in and changed the port. Then, last week when learning Node.js, I did not have to type in my username and password for github.com. This week, I had to do it everytime I made a git pull request from Putty. This was a nuisance which I was able to solve by changing the repository from private to public.

Next to practice session, I ran the "inSessions.js" page with tweaks. First, I commented out "app.use(session())". This broke the entire intSessions.js page. This means "app.use(sessions())" is vital to using sessions in the application. Not only that a "secret", as in "app.use(sessions({secret: "some secret"});", is required to run sessions. Furthermore, "secret" needs to have some value. Specifically, it's value needs to be a string, a number will not work.

I ran into most trouble when destroying sessions. I could not figure out where to place the destroy session code without destroying the functionality of the entire web application. It is not logical to add "req.session.destroy()" within the route, because when a user is adding their name and then adding items to the to do list, their session has to stay intact. In this scenario, the only place where a session can end is if the user deliberates wants the session to end. Say the user is walking away from their computer and they do not want others to access their to do list on the same browser. So, I added an "End Session" button and in it I added the functionality to destroy a session. This worked. When a user clicks on the "End Sessions" button, the current session is destroyed. But, it is not perfect. The user has to manually refresh the page to wipe all the text from their session.

From sessions I moved onto testing HTTP requests, POST data, and nested asynchronous calls. Following along the code for "helloHttp.js" and "helloOrganization.js" was intuitive. We have already completed assignments for each of these concepts on their own. We have made asynchronous get and post calls. The only new item this week is that now we are combining both techniques into one route. That is, we are sending GET request, POST request, and only after we have received response from our request, we will render the page, which will contain data received from our request. With a fair bit of back and forth I was able to write programs that were similar to "helloHttp.js" and "helloOrganization.js". I found that separating each callback into its own function alleviated a lot of confusion that comes with nesting multiple function into one.

In this section I ran into problems with "credentials.js" file. Everytime I started node for "helloHttp.js", an error would occur that said "credentials.js module not found". I went through the lecture video to understand my error, but I could not find any. Then, instead of ignoring the "credentials.js" file from git, I decided to add it and see if that would fix the problem.

While uploading "credentials.js" did solve this problem, doing so did not sit well with me. The credentials file has my API key and to keep it secure from prying eyes, I must be able to hide it. So, I tried something else. I changed "var credentials = require('./credentials.js');" to var credentials = require('./credentials.js.template');". That worked. To verify if that was the issue, while running “helloOrganization.js” I kept “./credentials.js”. Node threw a “Cannot find module” error. Then, adding “.template” fixed the issue.

In the end I have only scratched the surface of all the way in which these codes can break, and all the ways they can help us create better websites. I have enjoyed fiddling with the code for this activity and will continue to experiment with it.