UofT Exchange

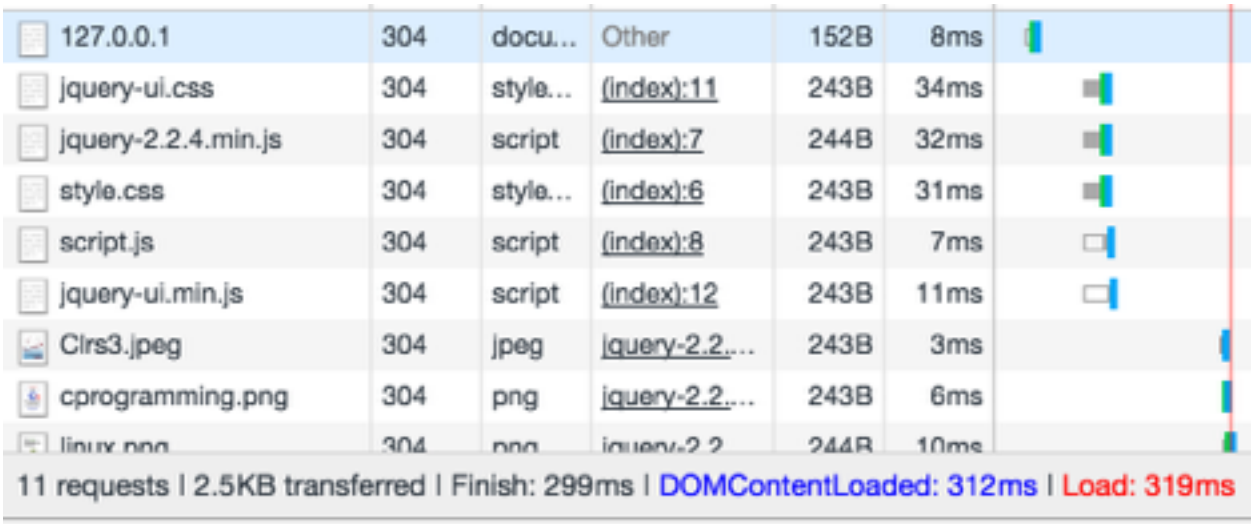
Design of the Application
Our design architecture follows that of the MVC, the user makes requests by clicking on buttons and filling in forms in the front end and the backend processes data and sends appropriate responses for each requests and the front end updates accordingly.

To start, the user fills in the form to sign up on the website. Then data that the user fills in is then sent to the database for storage and when the user tries to log in with that same account, they we use the login info they've provided to to verify it on the server that sends back a response. Depending on the response we change the view of page, i.e. if they've successfully logged in, we show the user menu on left, else we display an error message to the user. Similarly, when a user wants to view they profile by clicking the profile button, we get that response on the server and send back the user information request. Then the user could choose to update the info which is another request which then we update the information in the database then display the new information to the user. Most of the actions that the user takes involves a series of processes. We send the corresponding data for each response to the server then we process the data and send an response to the front end. The front end then gets the response that changes the view (DOM) of the site according to the website. The main role of the server is to process delicate data, the role of the front end is the receive user input and the role of the front end script is to process the response from the server to generate an corresponding view for the user.

Testing
We used mocha, expect.js, request and chai for our testing. Some general test cases we have is determining if an user is authorized to access a specific part of the app. For example, when an user is not logged in, going to the index page returns an ok status code, and we check if that status code matched out expectations. Also, if a regular user tries to access the admin page, we would expect a forbidden status code and we test that. Some other test cases we have is on authentication, if the authentication process fails, it will send a 400 status code and the user should be logged on, else the user should be directed to the user menu page with status code 302. We had some test cases on on that as well.

Security
In the early phases of you app development process, we did not address any security vulnerabilities. Since the back end of our app did not use place holders for queries, I was able to run ' UNION SELECT email, password FROM users WHERE email LIKE ' in the input field for search query and get all of usernames and passwords that are stored in the database. Similarly, when updating user information, I could remove that input constraints by changing the DOM locally then change the email to the profile information of the person that I want to affect, submitting the information changes the information on the other user. So the frameworks we used to prevent these are express-validator and expess-session. To prevent cases like the first one from happening, we escaped all user input before processing it and validated that inputs are in the correct format in the backend using express-validator and used placeholders for SQL statements. To prevent things in the second case from happening, we use express-session to store relevant information that we need to validate an user. By doing this, we are able to prevent SQL injects and cross-site scripting.

Performance
In the front end, on our live host we minified our html, css and javascripts as well we used gzip to compress our responses. By using these performance enhancements the files we request from the server are a lot less in size now, previous the images files are at least 100kb after compression the request of the index pages is a lot less.



In our backend, we initially decided to use a NoSQL database system. But in the early phases of our implementation, we were told that even though that NoSQL is very flexible in terms of storing information, it is not as efficient as a relational DBMS. So, later we optimized our queries to be as efficient as our knowledge of database takes us. However, we couldn't see much of a performance difference between our data set is quite small, but we believe that if we had a bigger data set, we might see an performance improvement.

Live app:
https://evening-sands-50730.herokuapp.com/

Github:
https://github.com/yanrs17/CSC309A4-Large-App

Enhancements
Some enhancement we made was that we created a responsive design for the website, applicable for devices from 320, 480, 768, 1024 and 1280 and above. We believe this enhancement no longer restricts our app to desktop users and opens it up to the mobile platform.

We also implemented an sticky navigation bar for devices with large screen so it provides a lot accessibility to the other features of our app.

We have implemented a view so that given any where in the site the user is in, the menu for all features of the app is visible and accessible by the user, we believe that this improve is really crucial to our app, because I for one do not like scrolling around looking for menu options at the

top of the screen. I would rather prefer navigate to another section of the app, once I am finished with this one.

We included an image slider to the index page of our app to show case some of the contents we offer at this site. This gives the user a general sense of what we offer at the site, without exploring too much around the site and later find out what the site does.

Also, we implemented a set of basic features that are accessible to all users of the site without logging it. So if an user use want to look up the contents offered at our site, he/she can just do it with out signing up and signing in. But by logging into the side with an account, the user is presented with some other features such as messaging, view other user's profile, liking and commenting on offerings that enhances user experience.