Name: Awwab Mahdi
Student Number: 101225637

COMP 2406 Final Report

YouTube Video Link: https://youtu.be/WQFf0qMQNog

Start Server

Assuming that the zip file that contained this submission was downloaded and extracted in an appropriate location on the drive. Assuming also that Node.js and mongoDB are installed.

In order to start the server you will need to first ensure that the mongoDB server is running. This can be determined by using Task Manager on Windows(it won't be in the first couple tasks). Once you've ensured the mongo server is running, you can open a terminal window at the location of this report.pdf file. You can then type in the command "**npm install**" and press enter, which will install all modules and dependencies. Next, you need to type in "**node database-initializer.js**" and press enter. This will initialize the mongoDB database and populate the collections. Finally, type in "**node server.js**" and press enter. You should see a message in the terminal ending in "Server listening at localhost:3000". You can then open a new tab in your browser and type in "**localhost:3000**" in your search bar. Pressing enter will take you to the application webpage.

Overall Design and Implementation

2. Discussion and critique of your overall design. See the 'Overall Design and Implementation Quality' section of the marking scheme for ideas on what to include in your analysis. You should also consider some of the key concerns of web application development that we have discussed in class, such as scalability, latency, etc.
In this project I attempted to follow some of the RESTful design principles. One principle I followed was the separation of client and server. The client is largely HTML files and client JS files which send requests and make small local changes to the webpage. Statelessness was a RESTful design principle I was not able to maintain, as the desire to have user sessions required session data to be stored on a server, which creates a dependency on the current configuration. Additionally, caching was not really implemented in the application either. Caching would have resulted in fewer needless changes to the website pages, and would have likely allowed for fewer requests overall. Specific content that would have been useful to be cached would be the logged in user's object and client JS files. In terms of implementing a uniform interface, I believe that I have established the manipulation of resources through representations, self-descriptive messaging, and HATEOAS(hyperlink usage). However, some

resources are not correctly labelled, like profile, which should ideally be “user/userID”, perhaps with some notifier that it’s the current user. For that reason I would say that I only mostly followed the identification of resources sub principle. My system is layered as there is a division between the client, the server, and the database, which each interact through their own APIs.

I used HTTP status codes as necessary to inform the client of changes made to the database and other general changes. I believe that I handled most applicable errors. Most of my relevant functions were also asynchronous. As previously stated, caching and not repeating requests would have minimized data transfer, and I believe some requests could have been avoided as a rule by storing frequently accessed information on the client.

Design Decisions

My design was very rudimentary and simple, essentially completing the parameters and leaving the rest. There were no significant design decisions apart due to the inability to make any.