Final Project Report

Liam Lowndes

101041818

COMP 2406

Carleton University

Important information

Public IP : 130.117.130.19

Instance name: liamlowndes

Username: liamlowndes

Password: comp2406

Instructions on how to run the system

Navigate to the terminal on the Open Stack console. Move to the directory Home then to Student then Documents then to COURSE PROJECT – MAIN. In this file you can run the command “node server.js” to start the server.

Then navigate to your own command prompt and enter:

- ssh -L 9999:localhost:3000 student@134.117.130.19

Then enter the password:

- comp2406

You can then access my system by entering http://localhost:9999 into your won computers browser. This address also works if you want to access API functionality with postman.

Functionality of the website

There are a lot of features in this website. The first is sessions. You can not access any information on my website until you are logged in. To log into my website, you can use the auto filled log in form that signs you into Eddy’s account. You can also look in my people.json file and use their username/password to sign into any account you would like. When logged in the home pages refreshes and shows you your recommended movies. These movies are chose based of the reviews you have given and the people you follow. Movies are chose randomly and then checked against two criteria: does the movie have a genre that the user has interacted with (by commenting on a movie) and does the movie have any actors that the user follows. An actor is worth two points and a genre is worth one, if the movie scores more than three points it gets put on your home page. By choosing from random videos this makes sure that the list is different, but still relevant, every time.

I implemented search pages so that you could look through all the users/people/movies and filter it based on certain queries. Another thing I added was the header that implemented and authorization feature. If the user was a contributing user, they have the movie submit button and if they are not, they can not access that button or that page. They are not even able to access that page using the API. This also leads into another feature which is that you can submit movies to the database. Once submitted a movie with the same title cannot be submitted again. On top of this when people who are not in the database star in the new movie those people get added to the database. It also updates every actors’ frequent collaborators section of their profile.

I also added the feature to follow and unfollow people or users. This is indicated on what button you see beside their name. You can comment on movies and see your comment underneath them once you refresh the page. This also updates the movies average rating. When it comes to your personal user page you are able to toggle whether you are a contributing user or not as well as view all the users and people that you follow. This data about who you follow is public and can be viewed by any users.

Finally, when it comes to extensions, I am proud with the way my project looks. Both aesthetically and when it comes to mobile use. I kept user experience in mind and tried to make it as easy and clear as possible to perform any step that you would want to take. Although it does not look perfect on mobile devices, the use of bootstrap really makes the webpage a lot more versatile than any vanilla HTML would looks.

Overall design of the system:

This system was built with three core design principles in mind: Model View Control (or MVC), Client-Server Architecture and modularity. This is the idea of splitting up your code so that it only has to handle one aspect of your system. The way that I implemented this was by having client-side JavaScript that handled events once the page was loaded and server-side JavaScript that handled the backend logic that the client does not need to worry about. Each page was made with the templating engine Pug and had its own JavaScript file. These client-side JavaScript files main purpose is to handle button clicks. One example of this is the edit / save changes button that can be found on the client's own user page as well as on the movie pages if the client is a contributing user.

On the client side the edit buttons transform the pages so that you can edit the information on the page. The button then says, "Save Changes" and once clicked it sends that information to the server side. Once received on the server side, this new information updates the database and then alerts the client. The client-side JavaScript then decides how (if at all) it should display this response. This kind of interaction can also be seen with the "follow / unfollow" and "become a contributing user / remove contributing user privileges" buttons. These kinds of interactions are a good example of how I used MVC and client-server architecture to design my system in an organized manner.

There were many benefits of designing my system in a modular way; one of which being that it made it easier to debug. When there was one part of my system that was not working, I could look at that specific spot because it was made with a modular design. If the code were not broken apart like this it would be difficult to tell where the bugs were. Another benefit of this design is that it is extensible. If better server technology were to be invented some time in the future, I could change what the server side looked like and not have to worry about the client side. It also means that if I want to build on top of it in the future it will now be much easier to either remove outdated parts or add new sections of code without effecting the rest of the system.

Assumptions that were made

There were some assumptions made in the making of this project that I will lay out here. The first assumption is that the client will not want to access the website while not logged in. When not logged in every page redirects you back to the homepage until you sign in. I made this assumption because there are a lot of user specific operations that the website can offer, and the website would look bare bones if the user was not logged in and couldn't have many meaningful interactions with it. In the real world this would only be done if the website really wanted to have people commit to making accounts. This could also be a design decision used if you had paid content you did not want non-members to see.

The second assumption I made was that when searching for movies/people/users the client would only ever want to see 100 items at a time. I made it so that the query limit could only by set to 100 and if it were any higher it would default back down to 100. This is because I did not think anybody would reasonably want more than 100 items at a time on one page. I applied this same limit applied for the API.

The final assumption I made was that the user of this website would have no trouble using a website in a typical way. It is important to remember that some people have difficulty interacting with websites due to physical obstacles such as blindness or chronic pain. In this website I did not implement any features that would accommodate for these types of things, but I would try to if I were releasing it to the general public. Since we were not asked to do so I could safely assume that this would not be a problem for the project but as a common design principal it is good to keep accessibility in mind.

Discuss technologies used within the application

There were 7 main technologies that I used within this application and they were Node.js, NPM, Express, Cookies (with express sessions), Bootstrap, Open Stack and Pug.

Node.js

The first of the technologies that I utilized for my application is Node.js. This is what allowed me to run a server in the first place. It is the initial starting place to break my code up between client side and server side which gives the application the Client-Server Architecture. The server-side code runs on JavaScript which is quite convenient when you are coding on both sides of the application. You do not need to context switch in your brain as often which leads to smoother development.

NPM

NPM or Node Package Manager is an addition to Node.js that lets you easily use code that other developers and developer communities have wrote. You can use it to download code files which essentially work like a group of functions that you can require and use without having to know exactly how they work. This creates major opportunities to create applications a lot quicker as well as writing code that is a lot more robust than it would be otherwise. You are not the only one that is working on your code base now, there are other developers who are now working just to make sure that one part of your system is working to the best of its ability. This project extensively uses the NPM module express which is helpful when it comes to routing traffic on your website.

Express

Express is a powerful package from NPM that allows for managing a lot of different cases with not a lot of code. Before using express each individual URL route had to be accounted for in an if statement. This starts to get messy fast when you start adding multiple pages that can also accept query parameters. This is where express comes in. You can set up specific routers for the was some URL's start and then let express take care of what the URL's say all together.

In my system there is a router for the routes: / (or the index), people, users, movies and following. Each of these can take on lots of different forms. For people, users, and movies you can attach search parameters on to the end of them such as:

"http://localhost:3000/movies?year=2000&limit=10&genre=comedy"

This would give you the first page of ten results of movies from the year 2000 in the comedy genre. With express this becomes far easier than it would be with just vanilla Node.js. Of course, it is possible to write your own functions but there is no need in this case when express is very well suited to do the job.

Cookies/sessions express-sessions

Another important technology I used was cookies/sessions, specifically with the NPM module express sessions. Cookies are information stored on your browser and sessions are information that is stored on the browser and server side. Using express session to implement sessions I was able to make a login/logout feature on my website. This meant that a user could log into there account and see things specifically tailored towards them.

Bootstrap

Bootstrap is an open-source CSS framework that I used to make my website look nicer. Bootstrap makes it so that organizing data on your webpage to look good becomes a lot easier.

Open Stack

Open stack is a cloud computing platform that I used to test my code on a real server. Open Stack let us you use an actual server to serve your content. In the real world a website would be stored on a physical server that is running 100% of the time (ideally) and is constantly ready to take requests and serve your website at all hours of the day. By moving my project to open stack my computer no longer must be on for my system to be running which is obviously a big advantage.

Pug

Pug is a template engine that makes it so that you do not need to write an HTML page for every webpage on your site. It accepts variables so that you can send it data and it can load the page according to what you give it. On my website I used it for every webpage because each page had a new way to be loaded depending on the data that the client wanted. For example, the search page could be loaded in three different variations, you could search for: people, users, or movies.

Documentation for API

HTTP method: GET

Request header (content type): application/json

URL: http://localhost:3000/movies?params

Result: This will return an array of movie objects that can be queried with the following parameters. Queries start with a question mark after movies and are separated by an ampersand.

- title=string – this will return any movie that contains this string in its title
- genre=string - this will return any movie in that genre
- year=number - this will return any movie in that year
- minrating=number - this will return any movie with this rating or higher
- page=number – gets the next/previous batch of movies

- limit=number – the number of movies that come in each batch (default and max is 100)

HTTP method: GET

Request header (content type): application/json

URL: http://localhost:3000/movies/:movie

Result: Returns a movie object if that movie is in the database. The variable movie should be replaced with a movie title that is case insensitive.

HTTP method: POST

Request header (content type): application/json

URL: http://localhost:3000/movies/

Body: Include in the body one object that has the following key-value pairs. All the key values are capitalized unless otherwise stated

- Title: anyTitle – The title of the movie.
- Year: anyYear – The year the movie was released
- Genre: anyGenre – A list of genres that are separated by a comma then a space
- imdbRating: theRating – The rating it has on imdb. Just the one number, do not include /10 (no capitalized first letter)
- Plot: aPlot – the plot of the movie
- Poster: posterLink – A link to a picture online that can be loaded into an img src attribute
- Actors: anActor – list of actors separated by a comma and a space. (can be single)
- Writer: aWriter - list of writers separated by a comma and a space. (can be single)
- Director: aDirector - list of actors separated by a comma and a space. (can be single)
- Awards: awardList – list of awards won by the movie separated by a comma and a space. (can be single)

Result: Posts a new movie into the database and adds any new people to the database that were not already in it. Also updates frequent collaborators list for everyone involved in the new movie.

HTTP method: GET

Request header (content type): application/json

URL: http://localhost:3000/people

Result: This will return an array of movie objects that can be queried with the following parameters. Queries start with a question mark after movies and are separated by an ampersand.

- name=string – this will return any person that contains this string in their name
- Page=number – gets the next/previous batch of people
- Limit=number – the amount of people that come in each batch (default and max is 100)

HTTP method: GET

Request header (content type): application/json

URL: http://localhost:3000/people/:person

Result: Returns a single object of the person requested. This person is of the format:

- id – unique identity number
- name – name of the person
- movie – array of movie ID's
- jobs – array of jobs that person has had (either Writer, Actor or Director)
- picture – link to an online img resource of the actor
- collab – array of the top collaborators of this person (maximum of 5)

HTTP method: GET

Request header (content type): application/json

URL: http://localhost:3000/users

Result: This will return an array of user objects that can be queried with the following parameters. Queries start with a question mark after movies and are separated by an ampersand.

- name=string – this will return any user that contains this string in their username
- Page=number – gets the next/previous batch of users
- Limit=number – the number of users that come in each batch (default and max is 100)

HTTP method: GET

Request header (content type): application/json

URL: http://localhost:3000/users/:user

Result: Returns a single object of the user requested. This user is of the format:

- id – unique identity number
- username – username of the user
- password – password of the user
- picture – link to an online img resource that the user can pick
- following – array of user ID's which represent the users that the user follows
- followers – array of user ID's which represent the users that the user is following
- reviews – array of review ID's of the reviews this user has written
- bio – the bio that the user can edit
- contributor – Boolean value that is true if they are a contributing user and false if they are not

Recommendation algorithms

There were two recommendation algorithms that I implemented into my website. The first of which is the algorithm that recommends "movies like" a particular movie. You can see this at work on the /movies/:movie page. Each movie has four recommended movies that are displayed underneath the reviews. These movies are picked by moving through the database and finding movies that have at least one genre and one person that matches. So, all the recommended movies are of a similar genre with a similar actor writer and/or director.

Key feature of the system

I think the key features of the system are the search pages. When it comes to a database, I think it is very important to have a system in which you can easily move through that data and find what you are looking for. In this database there are over 9000 movies and over 27,000 people. You can use multiple different search queries to have these resources displayed to you in a user-friendly way. I think that is the power of this website, it is a data visualization tool. I believe that the search functionality is not only useful but also user friendly. You can interact with the data normally by viewing the website in your browser, but you can also access the API. Given that you can access the data in different ways as well as it being organized and easy to search through, I feel like search functionality is the key feature to this system.

How my project incorporated:

Scalability

My project is scalable because it has implemented technologies and techniques that make scaling the size of all aspects of the website easier. The first step towards this was using express, not having to have an if statement branch for every possible URL is a scalable solution. Utilizing the routers in express I can easily handle more webpages and more queries with not a lot of code. This same idea was implemented with pug. By having a template engine that can handle the creating of similar webpages makes it a lot easier to scale the website up and add different pages.

Robustness

One of the main aspects of robustness in this project is its reliance on some NPM modules. I have extensively used express and express sessions, and this is robust because I know that there is a large group of people working on the reliability of these modules. When you include code that has been made and used by many other developers you can be more confident that it has been thoroughly tested and improved. So, by using these NPM modules in my project I know they have the robustness from the testing of many other developers.

Extensibility

My project was built in a modular way which leads to a lot of extensibility. To begin with, my client and server are separated so wither or can be swapped out and you would not have to worry about it breaking anything. By separating the website up like this I can also make sure that I am doing the heavy computations (like database updates/management and running recommendation algorithms) on the server side rather than on the client's computer. This means that the client does not have to worry about running computations on a potentially slow machine and just has to load in the data once processed.

Another extensible aspect of my project is the routers set up with express. Now, anytime that I want to add a new page that will needs a new URL route I can easily set up something with express without much effort. I can then combine this with the power of the pug template and extending the website to offer more functionality has become a lot easier.

Maintainability

You can update the data in the database from the website which makes maintaining the database a lot easier. Rather than having to go into the code and files it is nice to be able to visualize your data on the screen. This makes maintaining the integrity of the data a lot easier than if you only had the code as reference. It is also maintainable because it incorporates the design decisions above so that running into issues becomes a lot less frequent. The key to maintainability is preparation. In making a website that is scalable, robust, user friendly and extensible you make it a lot easier to maintain.

User Experience

This project incorporates user experience in a lot of different ways. They first way is a bright and simple design. It is important to not overwhelm the user with a lot of data and a lot of interaction opportunities on one page. On my website I tried to make each page as easy to understand as possible with clearly defined and labelled buttons and links. I used bootstrap and CSS styling so that the page was nicer to look at. I also implemented a header that scrolled with the screen to make sure that every page should only ever be a few clicks away from any other page. I also offer an API which makes it so that the client can interact with the data on my site in a way that suits them best.

Improvements that could be made

There are few things in this project that I believe could be greatly improved upon. The first of which is being able to create a new user or person. The way that I would have implemented this would have been in a similar way to the movie submit page. If I continued to work on this project those functionalities would be where I started. I also wanted to add two editing features to the website. The first a way for the user to customize their page like being able to update their bio. The second thing I could have added was a way for contributing users to edit existing movies. The way I would have don this would be by packaging the response from the client into one JSON object and parsing it on the server side so that I could update my various database files.

Those were features I did not get to add but I feel like I demonstrated the knowledge in other ways. Like toggling the follow for users and people as well as allowing for reviews to be posted on the movies. By posting doing this I believe I showed competency in taking in user input and updating my system accordingly.

There are also some bugs in my program that I would have liked to fix. One which is the notification system does not seem to work properly. My approach to this however was to treat it how I did the reviews which I was able to implement mostly (I updated the movie data with the review, but I did not update the user). In doing so I would try and update the user data like I did the movie data to hold a notifications array that could be read out once on the notifications page. Although there are some bugs, I would say that I showed adequate competency when it comes to what I was trying to achieve with this project.

Although it is disappointing to not have a perfect error free program I think my use of express, sessions, client-server design as well as many other technologies and design principals made for a good display of my skills.