COMP 3100 – Web Programming

Project Iteration 3 - Summary

Group 20: -

Mohammed Farhan - 201852902

Hussein Abdelrahman - 201762846

Functionalities-

In Iteration 2 of making our Twitch data visualizing web application, much of the server-side code had been implemented, with only some functionalities missing. With this Iteration, the code gets complete with its client-side component, written with a mix of HTML/CSS/jQuery.

In this version of the application, it is able to render the data on a web browser, presenting it in a clean, efficient manner, and allows the user to perform the documented functionalities on the data.

With the latest submission,

- The application correctly executes and carries out server side operations/code to get connected with MongoDB and deploy the twitch data to a respective collection.
- Adding to the functionalities implemented in Iteration 2, three more operations have been registered, namely,
  - to list and get a static version of all the data, before it can be updated to make certain fields of collection (Streamer) objects adjusted with current day values of the respective streamers
  - a method to display a sample of the data, only including the more relevant streamer objects to be displayed when using the 'List' functionality on the client-side
  - a functionality on the client side to display graphically a summarized 'look' of some relevant statistics in the data
- The second aforementioned functionality had to be implemented because it would be inefficient to display the entire dataset, while one could just find the information of the streamer they wish to lookup via the 'Find' functionality, while also taking into account the large number of Twitch API requests it would take to load all the data within a short period, leading to more error prone data retrieval.
- Another miniature controller(in contrast to the general controller for carrying out most tasks) had to be created for retrieving all of the static, unchanged information from the original dataset, in order to use the non-updatable(as of current) fields in charting the summary for the data. Moreover, these select fields are more significant for plotting graphs with, in contrast to other fields. The daily views(one of the fields that gets updated to actual current values) of streamers could be a potentially good statistic to plot, but many streamer objects will have null values for users who are not live then, while for graphing we'd need an array of non-null values.
- Client side employs extensive use of flexbox in CSS to stylize the HTML components accordingly, and utilizes the Charts.js library for plotting graphs when the 'Summary' function/button is used. In addition, certain styles make use of libraries from Google Fonts.
- All listed streamer data on the frontend can now be updated/refreshed with up-to-date values simply by clicking the 'List' button again.
- The functionality for sorting data on basis of select fields could not be implemented and was dismissed, while the idea of adding a customized search for specific streamer, involving multiple parameter matching has been dismissed as well, since it wouldn't make much sense, as the 'name' field is a good enough primary key to lookup any individual streamer.

- A potential functionality that could be implemented in the future is a customized search for certain fields, listing groups of streamers(if any) with field(s) matching the non-numerical field in question. For example, to list all streamers having Korean as their channel language, or to list streamers playing a specific game from a subset of the collection that is updated with real time values.

Views-

The views section is comprised of the HTML frontend code, the CSS for supplying the styles to corresponding components, and Javascript/jQuery scripts for handling events and managing components on the client side. The scripts include,

- add.js - adds events for when user creates a new streamer entry to be added to the database collection
- find.js - manages events for retrieving specific documents from the collection, and for updating/deleting said document
- list.js - adds events to 'List' button for dynamically generating table listing subset of relevant data from the collection; on clicking 'List' again, all displayed data is refreshed
- graphs.js - manages events for creation and display of bar/pie charts showcasing summarized data with the help of Chart.js
- main.js - handles display logic for each functionality(button) on the main tab, whether to show or hide respective opened tabs

Frontend Components-

The webpage is comprised of a variety of HTML elements, structured and organised in a visually appealing manner, with related components such as the tab buttons, bar charts and pie charts grouped and wrapped in parent 'div' containers, which have been made responsive via flexbox and Charts.js. The tabs for 'Add' and 'Find' buttons display and utilise a 'form', thereby employing minimalistic yet efficient technique for data entry and retrieval. The 'List' button shows a table, made pretty using CSS, listing a bunch of information to the user, while passively keeping the data up to date. Finally, 'Summary' provides the user with charts that are detailed and responsive, yet elegant in presentation, with the help of the Chart,js library.

Animation Effects-

While this version of the app does not employ the use of animation much in the presentation of its client side, it does employ Charts.js, which has been used to create dynamic graphs that get plotted in short, sleek animations, all the while being responsive to the user.

Execution-

Here, it is assumed node.js and MongoDB have been pre-installed.

To run the code, make sure MongoDB is running, by going to its bin directory through the command prompt/terminal and running: mongo

In newer versions this may not work; use this instead: mongod

Usually, the path is, C:\Program Files\MongoDB\Server\5.0\bin

Once mongo is running in the background, open a terminal in the code's root folder/directory, for running the server-side code, and install the dependencies/modules specified in the package.json file on the terminal before running the script :-

Install dependencies - npm install

Run the script via - node app.js

Once the script is running, the application can be viewed via entering in the web browser, localhost:3000