Raaddit: Report

Programming Project: Team 5: Report

Ahmed, Matthew, John, Jeremie

12.04.2019

INTRODUCTION

We were tasked with handling of a large data set of a json object and told to make something of it. Given a strict schedule, we decided with a website approach. To start we came up with a few ideas for our website such as the website name, colour scheme and a basic layout of the website. After this we split the workload into 2 categories: Back-End and Front-End. From here we designated roles and began working on the project.

Throughout the process each of us would propose new features and we would discuss the feasibility of each. Our final working program contained the following features:

- Search Bar
- Audio Recognition
- Graphs
- Login and Register System
- Sort By Function
- Creating Posts
- Dynamic Database accessing
- Pie Charts
- Multiple pages
- Collapsible comments

Each of these features proved difficult to implement however our processes are outlined in the following headings: Back-end and Front-end

BACK-END

Processing and sorting the data became our top priority. First we converted the original JSON file into JSON objects and displayed them. However we found this approach to be too slow, even with the small dataset. So we knew we had to try something else.

The solution came after a few days of pondering, where we decided on using an sql database instead. For the demonstration, a MySQL database was used. This database contained the 14 million entries of the large dataset. We wrote a Java program to insert the JSON data into the database. Each JSON entry was parsed using regular expressions to get the data from each key. This data was then inserted into a prepared statement, which was then executed. For the largest data set, this program took about 2 hours to run. The MySQL database was replaced with a smaller H2 database for the purpose of testing.

Now that we had the data we needed a way of sorting it. Luckily sorting in sql is extremely quick. The query class took care of accessing the database. Queries were sent using the processQuery, which would return the relevant ArrayList of posts. The createQuery function was used to turn what the user searched for into a valid SQL query. Comments were retrieved by searching for comments which contained the correct “parent” value.

Login, Register and Creating Posts

A new database was also needed to process the user feature. By creating a new table and combining our widgets with SQL queries we implemented a login and/or register screen and send the username and password as plaintext to the database. This allows a user to register during one session and login with those details in the next. The widgets also inform the user whether a username already exists or their password is wrong.

Being logged in allowed the user to access more features. When logged in, one could upvote posts, which sends a new query to the database and increments the score counter there. As well as this the user could create their own posts. This worked similarly to the login functions. The user is prompted to input the URL and title of their post. A button was added to allow the user to paste the URL from the clipboard. Then this data, along with the current system time and the user’s name was sent to the SQL database and a new post was created. This could then

be seen by the user when returning to the homepage.

FRONT-END

Widgets

To speed up the process, we split up most of our smaller features as widgets. This varied greatly in both looks and functionality. This allowed us to use processing made widgets instead of imported libraries which added uniqueness to our code.

Auto Fill

This feature allowed the user to quickly choose an option to search for instead of putting in the full search word. This worked in a way to allow the most used words to appear in the search bar instead of random words.

Graphs

We made a graph of the most recent users who had posted to the website. In the graph we added the current user along with the total number of posts that user had and total number of comments. When that user was clicked on it would bring up a pie chart with the ratio of posts to comments that the user has made. We also created another pie chart which shows the top 10 most frequent website within 1000 entries of the current query.

Opening

The opening was a very important part of the website as the user would be first welcomed to the website through the opening screen. For this feature we created an animation of the text “Raaddit” being uncovered on screen and when it had completely appeared 5 bars would move up the screen which revealed the website screen at a smooth rate.

Search Bar

For searching up users and posts we created a search bar at the top of the header. This

was completely created from scratch and worked in a way that it would search very fast through the 14 million data set. If a search had multiple pages worth of posts, it would allow you to switch pages.

Voice Recognition

One of the key features of our website was vocal recognition. We knew that most people were going to follow the same approach as what we did so we wrote a javascript code that would convert voice recognition from google search to processing in a form of a string which allowed the query to be searched when activated.

Collapsible Comments

The collapsible comments feature added a whole different dynamic from what others were doing to sort the comments. This featured allowed the user to collapse the comments so it doesn't clutter up the website and tarnish the minimal design that was pursued for the project.

Scroll Bar

The scroll bar allowed for easy navigation looking while looking at the posts or the comments as the user is able to view them quickly and navigate from start to end by dragging the bar up or down. As well as giving the user an indication of how far down the page they are .

CONCLUSION

In conclusion we would deem our project a success after demonstrating it as the judges were very impressed by our colour coordinated outfits, structured layout and professional approach