DBMS II
<u>Assignment 2 Report</u>

Short Summary on how the Application is Built:

This is a short description on how we built the web application. Work was divided into two parts, 2 members handling backend and 2 members handling frontend. Finally, in the last few days we tried to link both. Some of the libraries we tried to use from the references were outdated and we tried to search out for modern implementations of them.

In the first two weeks, we searched for the technologies and programming languages that can be used, decided on what all to be used and then learnt those technologies required. By working in pairs, work went in parallel and challenges arose were solved together.

We searched and learnt from different Community Question and Answer websites. From those we got enough clarity on what to be done for our project and how to make it more like a formal CQA website.

The database provided had many null values and it was corrupted. So we decided on making our own schema and populating it with our own data.

System architecture :

Front End:

- Index.js - The first file that is triggered by the script addition in index.html
- App.js - This file is called by the Index.js. It contains all the routes.
- Login.js - This is the home route of the web application. It has a form which takes in Username and Password to verify from the database using ‘axios’ module. If you are a new user, you can use the “Signup?” button.
- Signup.js - The “signup?” button in the Login.js file leads here. It also has a form to accept a new user. It takes in the required information and sends it to the server using axios.post method. If there is no existing user with the same name, you are redirected to the login page, where you can login using your information.
- Home.js - This file loads the home page of the application once the valid information is entered into the form of Login page. We pass the username information from the login page to access required information from the server.

There is a search box and a set of radio buttons, that can be used to filter the posts.

- Searchbyuserid.js - This file is triggered when the radio button of “by user id” is active and some text is in the textbox and the “Go” button is clicked. This renders a new page with all the posts created by the user with the user id from the submitted form in home page.
- Searchbytag.js - Similar to the Searchbyuserid, this opens up a new page with all the corresponding posts when the radio button is switched on for the searchbytag field and the “Go” button is clicked.
- Searchbymultipletags.js - Similar to the searchbyuserid and searchbytag, this also renders a new page with all the posts with the submitted tags as a sub array of their tags. One has to use a comma delimiter in the text box while specifying the tags.
- Myprofile.js - This file is triggered when the profile button is clicked in the home page. This renders the page personal to a particular user. For a logged in user we only give permission to edit a post in this page as this page contains only the posts which are created by the particular user. We give permissions to edit, add and delete a post in this page. We also display personal information like location ans about me in this page.
- Comments.js - This file is rendered when the comments button of a particular post is clicked in any of the pages containing a post. It has all the comments of the corresponding particular post. This does not display upvotes and downvotes of the post but only the content of the post and comments are rendered.
- Addpost.js - This file is rendered when the add post button is clicked in the profile page. It displays a form to take in the required information to create a post. When the form is submitted, the user is taken back to the profile page and can see the new post in the posts list there.
- Editpost.js - This file similar to the add post file, is rendered when the edit button of a particular post is clicked in the profile page. The user is taken to a new page with a form that has the attributes filled corresponding to the post before clicking edit, and the user can make changes and click submit which takes the user back to the profile page where he can reflect the changes made.
- Deletepost.js - This file is is executed once the delete button of a particular page is clicked in the profile page. The post is deleted and the page is reloaded to reflect the changes.
- Upvotes and Downvotes - For any given post upvotes and downvotes are buttons, which are clickable and they change their values when clicked.

Back End:

Index.js :- This has all the logic corresponding to the updates, insertions and deletions received from the client. There are several get and post methods to update, insert, delete, get information from the database. They are as follows.

- get("/users") - this returns information of all the users in the database.
- get("/home") - this returns the information of all the posts in the database to display them in the home page at the client end.
- post("/myposts") - this returns the information of all the posts corresponding to a logged in user whose name is passed in as a request, that is the reason for this being a post method.
- post("/userposts") - this returns the information of all the posts corresponding to a user whose id is passed in as a request, that is the reason for this being a post method.
- post("/tagposts") - this returns the information of all the posts whose tags array contains a particular tag , which is passed in as a request to the method.
- post("/multipletagposts") - similar to the previous method, this returns all the posts whose tags array contains the particular set of tags which are passes in as a request to the method.
- post("/bio") - this returns the personal information that is to be displayed in the profile section of a user at the client end. This method takes in the username as a request.
- get("/votes") - this returns all the information regarding all the votes in the database. This is used in the home page to display upvote count and downvote count of a particular post using another post method.
- get("/tags") - this returns all the tag information present in the database. This is used similar to the previous method to get tags of a particular post in another post method.
- app.post("/comments") - this returns all the comments corresponding to a particular post whose post_id is given as a request to the method.
- app.post("/login") - this is used when a person tries to log in. A javascript object 'validity' is sent back as true if the user exists. Else false is sent to the client end.
- app.post("/signup") - this is used when a new user tries to sign in. It returns true if the user is new and adds the information to the users table. It returns false if the user already exists with the same name.
- app.post("/deletepost") - this method takes in the particular post id and deletes it from the posts relation using a delete clause.
- app.post ("/editpost") - this takes in the changed information from the client and updates the post table to reflect the corresponding changes in the database.
- app.post("/addpost") - this takes in the particular user id who is trying to add a post and the information of the post and adds it using an insert clause.

- app.post(“/addcomment”) - this method takes in the user name and details of the comment and adds the comment to the database using an insert clause. It also updates the posts relation to increase the comments_count of the corresponding post.
- app.post(“upvotehandle”) - this method is invoked when upvote button is clicked on any post that has it. This maintains the required relations between the number of downvotes, number of upvotes and updates or deletes or inserts them accordingly.
- app.post(“downvotehandle”) - this method is invoked when downvore button is clicked on any post that has it. This also maintains the required relations between the number of downvotes, number of upvotes and updates or deletes or inserts them accordingly.
- app.post(“autobyuserid”) - this method is invoked when search field is being typed in. It takes text and searches for usernames having this as a substring in the database and returns them, which are used for autocomplete.

- Cookie is added if a user logs in freshly. If he presses logout, then the cookie is cleared. If cookie already exists, then the user is automatically logged in.
- script.sql - This file contains schema of the database that we designed.

Programming Languages Used :

- Frontend - HTML, CSS, JavaScript, React, Axios
- Backend - Node, Express, Database(Postgresql)

Contribution of Each Group Member:

1. CS21BTECH11009 - Sahishnu - Backend: Node, Express, Database(Postgresql)
2. CS21BTECH11025 - Sriram - Backend: Node, Express, Database(Postgresql)
3. CS21BTECH11045 - Rishi Manoj - Frontend: React, Axios, HTML,CSS, JavaScript
4. CS21BTECH11029 - Vignan - Frontend: React, Axios, HTML, CSS, JavaScript