**3813ICT Assignment: 1**

Git Link:

https://github.com/GrumpyGoat123/3813ICTassignment.git

**Git**

The layout of the git repository simply contains the project folder and a README file. Default branch is main. During the development process, commits where done frequently usually after every stable milestone was achieved so that the application could run without any errors with every commit. Examples of a ‘stable milestone’ include things like new features to the application, a new completed route, improved design of a section/page and any major bug fixes/improvements that i potentially made previously.

**Data structures**

There are three main data structures which were used in the program. Each structure is stored in its own separate JSON file all stored /data on the server side of the application.

The first structure is called users, which is just an object with a username and their corresponding password to go with it for every user that has an account. This data will be used for login purposes only. This structure is separated from the rest of the user’s information for security reasons. By having a separate file just for the users file, the risk of accidently revealing sensitive information is minimized as this information will remain untouched unless a user is logging in or creating an account.

The second structure is called extendedUsers. This structure is used to store information about each user. This information includes userid, username, useremail, userrole. All this information is used to create a certain profile for each user and is where all the user’s data is stored except for the user’s password.

The third structure is called groups. This structure stores all the information for the groups including the users who are in the groups, the channels/rooms for each group and the users who are in those channels. A group has a name, group users, rooms, room name, and room users.

**REST API**

**Login:** Allows the user to login

**Route:** /login

**Parameters:** username, password

**Return** **Value:** userid, userlogin, username, useremail, userroll

**Description:** This route takes in a username and password when a user tries to login. In then checks if the username and password are correct by comparing the values to ‘users’ data structure. If the values are correct, it will return the users information by grabbing it from the data structure ‘extendedUsers’ as well as a true statement saying the user is logged in. It will then store this information in local storage on the front end. If the information is incorrect it will simply produce an alert and deny the user from progressing.

**Create Group:** Creates a new group and stores it in groups.

**Route:** /crtGrp

**Parameters:** groupname

**Return Value:** groupObject

**Description:** This route takes the group name as a parameter and creates a new object with keys of groupname, users[] and rooms[]. This then stores the new group into the groups JSON file and returns it.

**Delete Group:** Deletes a group from the data.

**Route:** /dltGrp

**Parameters:** groupname

**Return Value:** groupObject

**Description:** This route takes a group name as a parameter and finds it in the groups data structure. Once found, it deletes the it from the file and returns it.

**Create Room:** Creates a room and stores it in groups

**Route:** /crtRoom

**Parameters:** groupname, roomname

**Return value:** room, users

**Description:** This route takes a group name and a room name as a parameter and stores into the groups data structure. It first loops through to find the correct group and then pushes into the rooms key with a new room object (roomName, users[]).

**Delete room:** Deletes room from the data

**Route:** /dltRoom

**Parameters:** groupname, roomname

**Return value:** room, users

**Description:** This route takes a group name and a room name and deletes the room from the data and returns the existing rooms. It first loops through the groups to find the correct group and then loops through the rooms within that group to find the correct room. Once found it deletes it.

**Add user to group:** Adds a user to the group data

**Route:** /addUserGroup

**Parameters:** groupname, username

**Return value:** groupusers

**Description:** This route takes a group name and a username as a parameter and stores it into the groups data structure. It does this by first looping through and finding the correct group then inserting the name into that groups user array.

**Delete user from group:** Deletes a user from the group data

**Route:** /dltUserGroup

**Parameters:** groupname, username

**Return value:** groupusers

**Description:** This route takes a group name and a username as a parameter and find it in the groups data structure then deletes it. It does this by first looping through the group array and finding the correct group. Then it finds the username within that group in the users array and deletes it.

**Add user to room:** Adds a user into the room data inside group data structure

**Route:** /addUserRoom

**Parameters:** groupname, username, roomname

**Return Value:** roomusers

**Description:** This route takes a group name, username and a room name as a parameter and finds it inside the group data structure to add a new user into the room. It does this by first looping through the group’s name to find the correct group, then within it finds the correct room inside the rooms key and then stores the username inside the user’s key within the room. It then returns the room user’s array.

**Delete user from room:** Deletes a user from the room data inside group data structure

**Route:** /dltUserRoom

**Parameters:** groupname, username, roomname

**Return Value:** roomusers

**Description:** This route takes a group name, username and a room name as a parameter and finds it inside the group data structure to delete a user from the room. It does this by first looping through the group’s name to find the correct group, then within it finds the correct room inside the rooms key and then loops through the user’s array to find the correct user. Once found it deletes it and returns the new array.

**Create user:** Creates a new user or updates a user’s existing information

**Route:** /crtUser

**Parameters:** userid, username, useremail, userrole

**Return value:** users

**Description:** This route takes the users information as a parameter and creates a new user or updates information on an existing user. This will depend on if the user who is logged in as the role/access to do this. This route checks if the username exists and if it does not it will store a new user into the user’s data structure with the new information given.

Each route has specific error catching parameters within them that check if names exist and what happens if things go wrong. All return different things and alert statement is called if anything goes wrong during the process of these actions.

**Architecture:**

The front end of the application is split of into four components: Home, login, chat and profile.

**Home:** The home page is just a start of page displaying how to use the app and some background. This is just a simple start of page

**Login:** This page is used for the user to login into the app. It asks for a username and password and has a submit button. Once entered and the credentials are correct it will send the user into the chat page.

**Chat:** This is the main space of the program. This is where the users can chat with each other (implementation in phase 2) and most of the functionality of the app is at. This page will allow the user to create groups and rooms and join groups and rooms.

**Profile:** This is the final page and is used for editing your profile information. This page can be used to change your own profile and/or edit other users profile information and creating new users. This page is mainly used for super admins and group admins. Users can also logout from this page