3813ICT Assignment Phase 1

- Link to my github: https://github.com/vydang123/3813ICT-Assignment-Phase-1
- clone my repository: https://github.com/vydang123/3813ICT-Assignment-Phase-1.git

GIT

Layout of a Typical Git Repository:

- .git/: This is the directory where Git stores the metadata and object database for your project. This is the most important part of a Git repository and copying this folder elsewhere will clone the repository with its full history.
- Working Directory: The primary area where I'll be making changes to your code. This is the "current snapshot" of my project.
- Index: Also known as the staging area, this is an intermediate area where changes are collected before being permanently stored in the repository history.
- HEAD: This points to the latest commit in the branch that I've checked out.

Approach for Version Control Using Git:

- Initialization: Using git clone to copy an existing repository.
- Add Changes: Make changes in my working directory.
- Stage Changes: Use git add . to stage all the changes in the current directory and its subdirectories for the next commit.
- Commit Changes: Use git commit -m "Description of changes" to permanently store staged changes in the repository history.
- Push/Pull: Use git push to upload local repository content to a remote repository and git pull to download content from a remote repository and integrate it into the local one.

Data Structure

users.json and group-channel.json are imported to mongodb under database 'assignment' as collections 'users' and 'group-channel' respectively.

users collection

Main Data Structures:

- Array of User Objects: Each object in this array represents a user in the system. Attributes of a User Object:
- username: A string representing the username of the user.
- userid: An integer representing the unique ID of the user.
- role: A string representing the role of the user. Examples include groupadmin, superadmin, and user.
- groupids: An array of integers representing the IDs of the groups the user is associated with. In some user objects, there seems to be a slight inconsistency where the key is named groups instead of groupids (and the value is an integer instead of an array). This might be an error in the dataset.
- email: A string representing the user's email address.
- password: A string representing the user's password (usually, passwords shouldn't be stored in plain text for security reasons).
- valid: A boolean indicating whether the user's account is valid or not.

group-channel collection

Main Data Structures:

- Array of Group-Channel Objects: Each object in this array represents a group with its associated channels. Attributes of a Group-Channel Object:
- groupid: An integer representing the unique ID of the group.
- groupname: A string representing the name of the group.
- channels: An array of strings representing the names of channels associated with the group. There seems to be a slight inconsistency where, for some groups, the channels key has an integer value of 0 instead of an array. This might indicate that the group has no channels, but to maintain consistency, it would be better to use an empty array.
- members: An array of integers representing the user IDs of members associated with the group.

REST API

/addChannel

- Route: POST /addChannel
- Parameters: Expects a JSON object in the request body with groupName and groupId properties.
- Return Values: JSON response with success and message properties.
    - If successful, it returns { success: true, message: 'Channel added to group.' }.
    - If there's an error, it returns { success: false, message: 'Error message' }.
- Functionality: This route adds a channel to a specified group. It handles the following:
    - Validates and extracts groupName and groupId.
    - Searches for the group in the database.
    - Adds groupName to the group's channels.
    - Updates the group in the database.
    - Provides appropriate response messages for success and error cases.

/addGroup

- Route: POST /addGroup
- Parameters: Expects a JSON object in the request body containing group information.
- Return Values: JSON response with success and message properties.
    - If successful, it returns { success: true, message: 'Group added successfully.' }.
    - If there's an error, it returns { success: false, message: 'Error message' }.
- Functionality: This route is responsible for creating and adding a new group to the database collection. It handles the following:
    - Connects to the MongoDB database.
    - Validates the request body; if empty, it returns a 400 Bad Request response.
    - Inserts the provided group data into the specified collection.
    - Checks for the success of the insert operation. If successful, it returns a success response.
    - If the insert operation fails, it returns an error response.
    - Ensures the MongoDB client connection is closed regardless of success or failure.

/addUserToChannel

- Route: POST /addUserToChannel

- Parameters: Expects a JSON object in the request body with username, groupname, and channelname properties.
- Return Values: JSON response with success and message properties.
    - If successful, it returns { success: true, message: 'User added to the channel within the group.' }.
    - If there's an error, it returns { success: false, message: 'Error message' }.
- Functionality: This route handles the following actions:
    - Validates and extracts username, groupname, and channelname from the request body.
    - Connects to the MongoDB database.
    - Searches for the user and group in their respective collections.
    - Checks if the user, group, and channel exist; if not, it returns relevant error responses.
    - Adds the user to the list of group members for the channel.
    - Updates the group information in the database.
    - Adds the channel name to the user's list of channel names.
    - Updates the user's information in the database.
    - Responds with a success message after successful user addition.
    - In case of errors, it provides a server error response.
    - Ensures the MongoDB client connection is closed in the finally block.

/addUserToGroup

- Route: POST /addUserToGroup
- Parameters: Expects a JSON object in the request body with username and groupname properties.
- Return Values: JSON response with success and message properties.
    - If successful, it returns { success: true, message: 'User added to group.' }.
    - If there's an error, it returns { success: false, message: 'Error message' }.
- Functionality: This route handles the following actions:
    - Validates and extracts username and groupname from the request body.
    - Connects to the MongoDB database.
    - Searches for the user and group in their respective collections.
    - Checks if the user and group exist; if not, it returns relevant error responses.
    - Adds the user to the list of group members.
    - Updates the group information in the database.

    - Adds the group name to the user's list of group names.
    - Updates the user's information in the database.
    - Responds with a success message after successful user addition.
    - In case of errors, it provides a server error response.
    - Ensures the MongoDB client connection is closed in the finally block.

/createGroup

- Route: POST /createGroup
- Parameters: Expects a JSON object in the request body containing group information.
- Return Values: JSON response with success and message properties.
    - If successful, it returns { success: true, message: 'Group created successfully.' }.
    - If there's an error, it returns { success: false, message: 'Error message' }.
- Functionality: This route handles the following actions:
    - Connects to the MongoDB database.
    - Retrieves the new group data from the request body.
    - Retrieves the collection of groups from the database.
    - Determines the new group's ID by counting the existing documents and incrementing the count.
    - Inserts the new group into the collection.
    - Checks for the success of the insert operation. If successful, it returns a success response.
    - If the insert operation fails, it returns an error response.
    - Ensures the MongoDB client connection is closed in the finally block.

/deleteChannelFromGroup/:channelId/:groupId

- Route: DELETE /deleteChannelFromGroup/:channelId/:groupId
- Parameters:
    - channelId: Extracted from the URL parameters.
    - groupId: Extracted from the URL parameters.
- Return Values: JSON response with success and message properties.
    - If successful, it returns { success: true, message: 'Channel removed from group.' }.
    - If there's an error, it returns { success: false, message: 'Error message' }.
- Functionality: This route handles the following actions:

    - Connects to the MongoDB database.
    - Retrieves channelId and groupId from the URL parameters.
    - Retrieves the collection of groups from the database.
    - Searches for the group with the specified groupId.
    - If the group is not found, it returns a 400 Bad Request response.
    - Filters the channels in the group to remove the specified channelId.
    - Updates the group in the database to reflect the removal of the channel.
    - Responds with a success message after the channel removal.
    - In case of errors, it provides a server error response.
    - Ensures the MongoDB client connection is closed in the finally block.

/api/deleteGroup/:groupName

- Route: DELETE /api/deleteGroup/:groupName
- Parameters:
    - groupName: Extracted from the URL parameters.
- Return Values: JSON response with message property.
    - If the group is successfully deleted, it returns a 200 OK response with a message like: { message: 'Group deleted successfully.' }.
    - If the group is not found, it returns a 404 Not Found response with a message like: { message: 'Group not found.' }.
    - If there's a server error, it returns a 500 Internal Server Error response with a message like: { message: 'An error occurred while deleting the group.' }.
- Functionality: This route handles the following actions:
    - Connects to the MongoDB database.
    - Retrieves the groupName from the URL parameters.
    - Attempts to find and delete a group with the specified groupName in the 'group-channel' collection.
    - Checks the deletedCount property of the result to determine if the group was deleted successfully.
    - Returns a relevant response based on the success or failure of the deletion.
    - In case of errors, it provides a server error response.
    - Ensures the MongoDB client connection is closed in the finally block.

/deleteUser/:email

- Route: DELETE /deleteUser/:email
- Parameters:
    - email: Extracted from the URL parameters.
- Return Values: JSON response with a message property.
    - If the user is successfully deleted, it returns a 200 OK response with a message like: { message: 'User deleted successfully.' }.
    - If the user is not found, it returns a 404 Not Found response with a message like: { message: 'User not found.' }.
    - If there's a server error, it returns a 500 Internal Server Error response with a message like: { message: 'Error deleting user.' }.
- Functionality: This route handles the following actions:
    - Validates the presence of the email in the URL parameters. If it's missing, it returns a 400 Bad Request response.
    - Connects to the MongoDB database.
    - Attempts to find and delete a user with the specified email in the 'users' collection.
    - Checks the deletedCount property of the result to determine if the user was deleted successfully.
    - Returns a relevant response based on the success or failure of the deletion.
    - In case of errors, it provides a server error response.
    - Ensures the MongoDB client connection is closed in the finally block.

/getGroups

- Route: GET /getGroups
- Parameters: No request body or URL parameters.
- Return Values: JSON response containing an array of group data or an error message.
    - If successful, it returns an array of group data in a JSON response.
    - If there's a server error, it returns a 500 Internal Server Error response with a message like: { message: 'Server error.' }.
- Functionality: This route handles the following actions:
    - Connects to the MongoDB database.
    - Retrieves the 'group-channel' collection from the database.
    - Retrieves all groups from the 'group-channel' collection.
    - Converts the groups to an array.
    - Sends the array of group data as a JSON response.
    - In case of errors, it provides a server error response.

    - Ensures the MongoDB client connection is closed in the finally block.

/getUsers

- Route: GET /getUsers
- Parameters: No request body or URL parameters.
- Return Values: JSON response containing an array of user data or an error message.
    - If successful, it returns an array of user data in a JSON response.
    - If there's a server error, it returns a 500 Internal Server Error response with a message like: { message: 'Error retrieving users.' }.
- Functionality: This route handles the following actions:
    - Connects to the MongoDB database.
    - Retrieves the 'users' collection from the database.
    - Retrieves all users from the 'users' collection.
    - Converts the users to an array.
    - Sends the array of user data as a JSON response.
    - In case of errors, it provides a server error response.
    - Ensures the MongoDB client connection is closed in the finally block.

/postLogin

- Route: POST /login
- Parameters: Expects a JSON object in the request body with email and pwd properties.
- Return Values: JSON response with valid property and user information, or a response indicating login failure.
    - If login is successful, it returns a JSON response like:

      {

      "valid": true,

      "user": {

      "userid": "user.userid",

      "username": "user.username",

      "role": "user.role",

"groupnames": ["group1", "group2"],

"email": "user.email",

"channelnames": ["channel1", "channel2"]

}

}

- If login fails, it returns { valid: false }.

- Functionality: This route handles the following actions:
  - Connects to the MongoDB database.
  - Checks if the request body is empty. If it is, it returns a 400 Bad Request response.
  - Retrieves the email and pwd from the request body.
  - Searches for a user in the 'users' collection with the provided email and pwd.
  - If a user is found, it sends a JSON response with the user's information and valid: true.
  - If no user is found, it sends a response with valid: false.
  - Closes the MongoDB client connection.

/postRegister

- Route: POST /register
- Parameters: Expects a JSON object in the request body with user information, including email and username. Optionally, it can include an action to list users.
- Return Values: JSON responses with valid property and, in some cases, additional information.
  - If successful registration, it returns { valid: true }.
  - If the email is already in use, it returns { valid: false, message: 'Email already in use.' }.
  - If the username already exists, it returns { valid: false, message: 'Username already exists. Choose another.' }.
  - If the action is 'listUser' (super-admin page), it returns a list of users as { users: [user1, user2, ...] }.
  - If there's a server error, it returns { valid: false, message: 'Error in registration.' }.
- Functionality: This route handles the following actions:

    - Connects to the MongoDB database.
    - Checks if the request body is empty. If it is, it returns a 400 Bad Request response.
    - Retrieves the new user data from the request body, including the action.
    - If the action is 'listUser', it retrieves and returns a list of all users in the 'users' collection.
    - Checks if the email already exists in the 'users' collection; if it does, it returns a response indicating that the email is in use.
    - Checks if the username already exists in the 'users' collection; if it does, it returns a response indicating that the username already exists.
    - If the registration is successful, it assigns a new userid and inserts the new user into the 'users' collection.
    - Provides appropriate success or error responses.
    - Closes the MongoDB client connection.

/removeUser

- Route: POST /removeUserFromGroup
- Parameters: Expects a JSON object in the request body with groupname and username properties.
- Return Values: JSON response with success and message properties.
    - If successful, it returns { success: true, message: 'User removed from group.' }.
    - If the groupname or username is missing, it returns a 400 Bad Request response with a message indicating the missing parameters.
    - If the group or user is not found, it returns a 400 Bad Request response with a message indicating that the group or user is not found.
    - If there's a server error, it returns a 500 Internal Server Error response with a message like: { success: false, message: 'Server error.' }.
- Functionality: This route handles the following actions:
    - Validates the presence of groupname and username in the request body. If either is missing, it returns a 400 Bad Request response.
    - Connects to the MongoDB database.
    - Searches for the group with the specified groupname.
    - If the group is not found, it returns a 400 Bad Request response.
    - Searches for the user with the specified username.
    - If the user is not found, it returns a 400 Bad Request response.

    - Removes the groupname from the user's groupnames array (if it exists) and updates the user in the database.
    - Removes the userid from the group's members array (if it exists) and updates the group in the database.
    - Responds with a success message after removing the user from the group.
    - In case of errors, it provides a server error response.
    - Ensures the MongoDB client connection is closed in the finally block.

/updateUserRole

- Route: PUT /updateUserRole
- Parameters: Expects a JSON object in the request body with userId and newRole properties.
- Return Values: JSON response with a message property.
    - If the user's role is successfully updated, it returns a message like: { message: "User role updated successfully." }.
    - If the user is not found, it returns a 404 Not Found response with a message like: { message: "User not found." }.
    - If there's a server error, it returns a 500 Internal Server Error response with a message like: { message: "Error updating user role." }.
- Functionality: This route handles the following actions:
    - Validates the presence of userId and newRole in the request body.
    - Connects to the MongoDB database.
    - Searches for the user with the specified userId.
    - If the user is found, it updates the user's role with the new role.
    - Responds with a success message after updating the user's role.
    - If the user is not found, it returns a 404 response indicating that the user is not found.
    - In case of errors, it provides a server error response.
    - Ensures the MongoDB client connection is closed in the finally block.

Angular Architecture

Services

UserService (user.service.ts):

- Description: A service that contains methods for making HTTP requests related to user functionalities.
- Methods:
    - getUsers(): Fetches all users.
    - registerUser(userData): Registers a user with default properties and the given userData.
    - deleteUser(userId): Deletes a user by the given user ID.
    - updateUserRole(userId, newRole): Updates the role of a user.
    - Error Handling: Contains a private method handleError to handle HTTP errors and logs them.

ChatService (chat.service.ts):

- Description: Service to handle chat functionalities like sending messages and listening for new messages using Socket.io.
- Properties:
    - socket: Object for the Socket.io client-side connection.
- Methods:
    - sendMessage(message: string): Sends a chat message to the server.
    - getMessages(): Returns an Observable that emits incoming messages.

Components

1. AppComponent (app.component.ts & app.component.html):

- Description: Main component that displays the navigation bar and contains the login and logout functionality.
- Properties:
    - title: Application's title.
    - email, password: For handling login input fields.
    - errorMessage: For showing an error message during login.
    - userList: A hardcoded list of users to check during login (this seems like a temporary solution).
- Methods:
    - login(): Handles the login logic.
    - logout(): Clears session storage and redirects to the login page.

2. LoginComponent (login.component.ts & login.component.html):

- Description: Component for handling the user login using a form.
- Properties:
  - userpwd: Object holding the email and password for the user trying to log in.
- Methods:
  - loginfunc(): Logs the user in by making an HTTP request using the httpClient. If the login is successful, it saves user details to sessionStorage and navigates to the dashboard.

3. ProfileComponent (profile.component.ts & profile.component.html):

- Description: Allows users to view and edit their profile details.
- Properties:
  - userProfile: An object that holds the user's profile data, like username and user ID.
- Methods:
  - saveChanges(): Saves changes made by the user to sessionStorage.

4. SuperAdminComponent

- Description: Component for super admins, allowing them to register new users, view a list of users, and update or delete users.
- Properties:
  - newUser: Object holding the details (username, email, password) of a user to be added.
  - users: Array storing the list of all users.
- Methods:
  - onSubmit(): Handles the submission of the registration form. Registers a new user via the UserService, adds the new user to the local users list, and provides feedback on success or error.
  - fetchUserGroups(): Fetches the list of users and updates the users property.
  - upgradeToSuperAdmin(user: any): Upgrades a user to the role of 'superadmin'.

    - upgradeToGroupAdmin(user: any): Upgrades a user to the role of 'groupadmin'.
    - degradeToUser(user: any): Degrades a user to the role of 'user'.
    - updateUserRole(user: any, newRole: string): A private utility function for updating the role of a user, both locally and on the server side.
    - deleteUser(user: any): Deletes a user both locally and on the server side using UserService.

5. GroupAdminComponent:

- Description: Component that provides administrative functionality for managing groups, including CRUD operations on groups, channels, and user-group associations.
- Properties:
    - newGroup: Object representing a new group with properties groupid, groupname, and channels.
    - selectedUser: Holds the selected user.
    - selectedGroup: Holds the selected group.
    - selectedChannel: Holds the name of the selected channel.
    - users: Array of user objects.
    - groups: Array of group objects.
    - newChannelName: String representing the name of a new channel.
    - selectedGroupForChannel: Holds the group where the channel should be added.
- Methods:
    - ngOnInit(): Lifecycle hook that fetches all users and groups and checks user role on initialization.
    - fetchAllUsers(): Fetches all users and sets the users property.
    - fetchAllGroups(): Fetches all groups and sets the groups property.
    - addGroup(): Adds a new group.
    - addUserToGroup(user: any, group: any): Adds a user to a group.
    - addChannelToGroup(): Adds a channel to a specified group.
    - removeUserFromGroup(user: any, group: any): Removes a user from a group.
    - deleteChannelFromGroup(channel: string, group: any): Deletes a channel from a specified group.
    - deleteGroup(group: any): Deletes a specified group.
- Dependencies:

    - UserService: Not directly utilized in the provided code, but it's injected and might be used in a more detailed implementation.
    - HttpClient: Used for making HTTP requests.
    - Router: Used for navigating to other components/views.

6. RegisterComponent:

- Description: Component responsible for user registration, allowing a new user to provide a username, email, and password.
- Properties:
    - newUser: Object that holds the details of the user being registered which includes username, email, and password.
- Methods:
    - onSubmit(): This method gets triggered when the form is submitted. It invokes the registerUser method from the UserService to register a new user. Depending on the response from the server, it alerts the user about the success or failure of the registration process.

7. DashboardComponent:

- Description: A component displaying the user's dashboard. The content (specifically links) displayed is dependent on the user's role (user, groupadmin, superadmin).
- Properties:
    - userRole: A string that captures the role of the logged-in user. This role is retrieved from the session storage and used to conditionally render content.
- Methods:
    - ngOnInit(): An Angular lifecycle hook that gets triggered when the component is initialized. In this method, the user's role is fetched from the session storage and set to the userRole property. If the role is not found, an error is logged to the console.

8. ChatComponent:

- Description: A component that provides chat functionality. Users can send and view messages.
- Properties:
    - message: A string to capture the message typed by the user.
    - messages: An array of strings that keeps a list of messages sent.

- Methods:
    - sendMessage(): Sends the user's message using the ChatService. It also adds the message to the local messages list and resets the input field.