Software Frameworks (3813ICT)

Assignment Report – Milestone 1

Matthew Prendergast (s5283740)

CONTENTS

1.0 INTRODUCTION

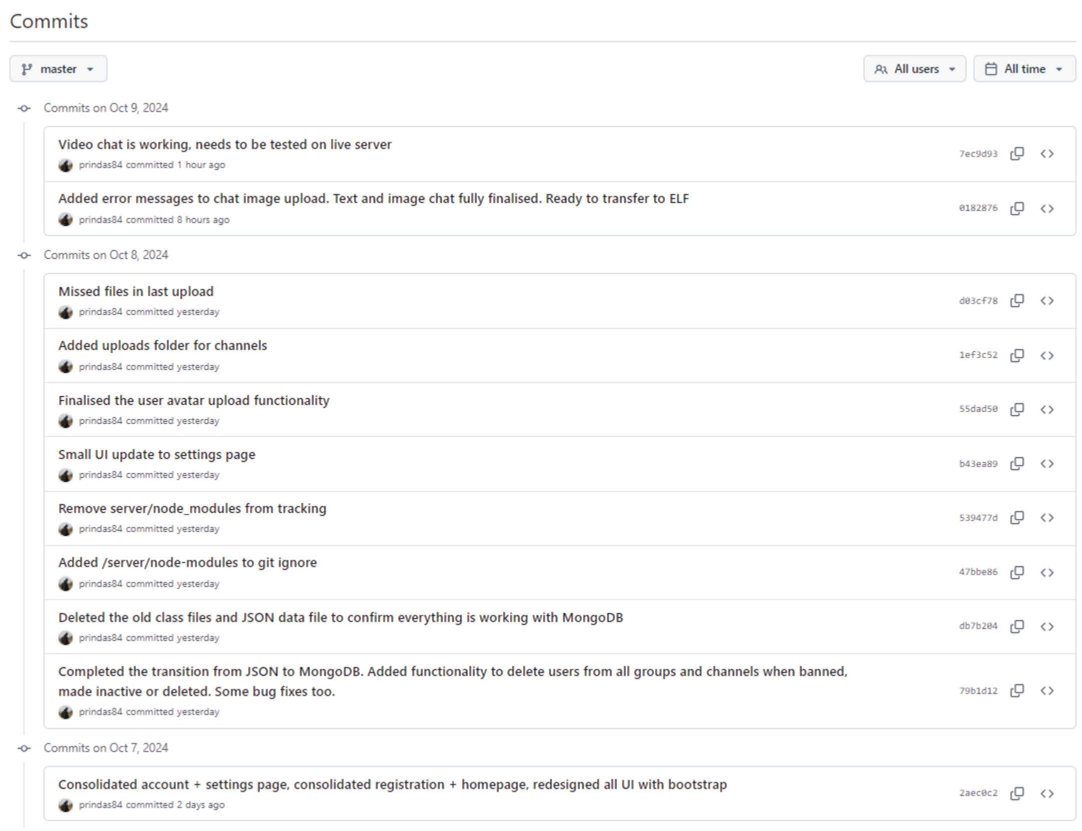
This project involves the development of a real-time text and video chat system using the MEAN stack, comprising MongoDB, Express, Angular, and Node.js, along with Socket.io for real-time communication and WebRTC for video chat functionality. The system enables users to interact within various groups and channels, with a structured permission hierarchy that includes three roles: Super Admin, Group Admin, and User.

The Super Admin has the highest level of authority, including the power to promote users to admin roles, manage all groups, and remove users. Group Admins can create and manage their own groups and channels, as well as manage users within those groups. Regular users can join group conversations and request group membership.

The chat application is developed with a modular approach, maintaining distinct server-side and front-end architectures to support efficient real-time communication and scalability. This document outlines the system's architecture, data structures, and the interactions between client and server.

2.0 GIT REPOSITORY ORGANISATION

GitHub URL: https://github.com/prindas84/chat


Commits

Commits on Oct 9, 2024

Video chat is working, needs to be tested on live server
prindas84 committed 1 hour ago
7ec9d93

Added error messages to chat image upload. Text and image chat fully finalised. Ready to transfer to ELF
prindas84 committed 8 hours ago
0182876

Commits on Oct 8, 2024

Missed files in last upload
prindas84 committed yesterday
d03cf78

Added uploads folder for channels
prindas84 committed yesterday
1ef3c52

Finalised the user avatar upload functionality
prindas84 committed yesterday
55dad50

Small UI update to settings page
prindas84 committed yesterday
b43ea89

Remove server/node_modules from tracking
prindas84 committed yesterday
539477d

Added /server/node-modules to git ignore
prindas84 committed yesterday
47bbe86

Deleted the old class files and JSON data file to confirm everything is working with MongoDB
prindas84 committed yesterday
db7b204

Completed the transition from JSON to MongoDB. Added functionality to delete users from all groups and channels when banned, made inactive or deleted. Some bug fixes too.
prindas84 committed yesterday
79b1d12

Commits on Oct 7, 2024

Consolidated account + settings page, consolidated registration + homepage, redesigned all UI with bootstrap
prindas84 committed 2 days ago
2aec0c2

3.0 DATA STRUCTURES

The server-side data structures used in the chat application primarily represent users, groups, and channels, as well as the connections between them. These structures are implemented as JavaScript objects and handled through the Node.js environment using Express and related modules.

3.1 User Data Structure

The user data structure represents a chat user and stores essential information such as username, email, role, and group memberships. The key attributes are:

- **id:** A unique identifier for the user. If an ID is not provided when creating a user instance, it auto-increments using a lastUserId variable to ensure uniqueness.
- **username:** The username of the user, which is unique and used for logging in.
- **password:** The user's password.
- **email:** The unique email address associated with the user.
- **firstName:** The user's first name.
- **surname:** The user's last name.
- **avatar:** The path or URL to the user's avatar image, which may be displayed next to messages in chats.
- **permission:** This field holds the user's role (Super Admin, Group Admin, Chat User).
- **active:** A boolean indicating whether the user is active in the system on banned.

This structure is essential for managing the authentication and permission system across different roles within the chat application.

3.2 Group Data Structure

Groups represent collections of users who can communicate in channels. Each group has its own set of administrators and users. Key attributes include:

- **id:** A unique identifier for the group. If an ID is not provided when creating a group instance, it auto-increments using a lastGroupId variable to ensure uniqueness.
- **groupName:** The name of the group, which is a required field and represents the identifier for the group within the system.
- **creator:** The user who created the group. This is typically the Group Admin, and they have the authority to manage the group's settings and members.
- **admins:** An array containing the list of administrators for the group. Initially, this includes the creator but can be expanded to include other users who have admin permissions in the group. These admins have the ability to manage members, channels, and other group settings.
- **members:** An array of users who have been approved to join the group. These users can participate in the group’s channels once they are added as members.

- **interested:** An array of users who have expressed interest in joining the group but are not yet approved by an admin or the creator. These users await approval before becoming full members.
- **banned:** An array of users who have been banned from the group. These users cannot access the group or its channels, and banning is typically performed by a group admin or creator.
- **channels:** An array of Channel objects belonging to the group. Each group can have multiple channels for discussions, and these channels are managed by the group admins.

3.3 Channel Data Structure

Channels are subdivisions within groups where the actual communication (chatting) occurs. Users within a group can join various channels. The key attributes are:

- **id:** A unique identifier for the channel. If an ID is not provided when creating a channel instance, it auto-increments using a lastChannelId variable to ensure uniqueness.
- **channelName:** The name of the channel, representing the discussion topic or focus within a group. This is a required field and serves as the unique identifier for the channel within the group.
- **creator:** The user who created the channel, typically a Group Admin. The creator has administrative control over the channel, including managing members and setting permissions.
- **admins:** An array containing the list of administrators for the channel. Initially, this includes the creator, but more admins can be added. Admins have the authority to manage channel members and settings.
- **members:** An array of users who are part of the channel. Members can participate in discussions and contribute to the chat within the channel once they are added.
- **Messages:** An array of message objects sent by users of the channel.

3.3 Reported User Data Structure

The ReportedUser class is designed to track users who have been reported by other users for inappropriate behaviour or violations of group or channel rules. These reports are reviewed and handled by the Super Admin, who has the authority to take corrective actions, such as banning the user or removing them from the system. The key attributes are:

- **id:** This is a unique identifier for each reported user. The ID is auto-incremented with each new report to ensure that every report is distinct.
- **user:** This attribute references the user who has been reported. It links to an instance of the User class, providing access to details such as the reported user's username, email, and permission level.

- **reason:** This attribute stores the reason for the report, provided by the user who filed the complaint. It describes the nature of the infraction or violation, which can range from harassment to violating group policies or other misconduct within the system.

3.5 Utility Functions

The utility functions used in the server-side code assist in managing data structures. For example, they may help validate user inputs, handle errors, or ensure data integrity across different operations like creating, updating, or deleting users, groups, or channels.

4.0 ANGULAR ARCHITECTURE

The Angular architecture of this chat application is structured in a modular way, adhering to the principles of component-based development, which enhances maintainability and scalability.

4.1 File Structure

- **app.component:** The main component that bootstraps the application and includes common elements across the application, such as navigation and layout. It contains the core HTML (app.component.html), styles (app.component.css), and logic (app.component.ts).
- **Modules and Components:** The application is divided into specific feature components:
    - **account:** Handles user account-related functionalities, such as displaying and managing account details.
    - **account-menu:** A sub-component that provides a user account menu for navigation.
    - **login:** Manages the user login page and authentication.
    - **register:** Handles the registration of new users.
    - **groups:** Manages group-related functionalities such as displaying available groups.
    - **view-group:** Displays detailed information about a specific group, including channels and members.
    - **settings:** Manages user settings, allowing users to configure various preferences.
    - **users:** Displays and manages the list of users in the system.
    - **video-chat:** Prepares the structure for video chat functionality, although it is not implemented yet.

4.2 Services

- **authorise.service.ts:** Manages user authentication and authorisation throughout the application, ensuring users are correctly authenticated before accessing protected routes.
- **page-meta.service.ts:** Manages metadata and page title settings for better user experience and accessibility.

4.3 Routing

The routing configuration is managed through app.routes.ts, where different paths are defined to link components to specific URLs. This provides seamless navigation between login, registration, groups, settings, and more.

4.4 Environments

The environment configuration (environment.ts and environment.prod.ts) manages the setup for different deployment environments, ensuring the app can adapt based on whether it's in development or production.

This modular and service-oriented architecture enables the Angular application to be easily extended and maintained, with a focus on separating concerns and promoting code reuse across different components.

5.0 NODE.JS SERVER ARCHITECTURE

The Node.js server for this chat application is built using Express.js, a web framework that simplifies routing and HTTP request handling. The server is structured to provide clear separation of concerns, making it easy to maintain and extend.

5.1 File Structure Overview

The server is organised into key folders and files that define its functionality:

routes/: Contains the route definitions for handling user and group-related operations. These routes define the API endpoints that the front-end Angular application will interact with.

- **user-routes.js:** Manages user-related functionalities such as registration, authentication, and user management.
- **group-routes.js:** Handles operations related to group creation, membership management, and channel administration.

utilities/: Contains helper functions used throughout the application to perform common tasks, such as data validation.

- **utilities.js:** Defines general utility functions for simplifying repetitive tasks and ensuring consistency.

node_modules/: This directory contains all the external libraries and dependencies required by the server, which are managed via npm.

5.2 Modules and Dependencies

The server primarily relies on the following modules:

- **Express.js:** Used for setting up the HTTP server and handling routes. It provides a straightforward structure for defining API endpoints and managing requests and responses.
- **Body-parser:** A middleware that allows the server to parse incoming request bodies in JSON format. This is used for handling form submissions and API requests from the client side.

5.3 Server Functionality

The current server setup handles core features necessary for the functioning of the chat application:

- **User Management:** The server supports user registration and authentication using basic username/password validation.
- **Group and Channel Management:** The server enables creating and managing groups and channels, allowing admins to organise users into different discussion spaces.

Although Socket.io and Peer.js for real-time chat and video functionality are not yet implemented, the foundational architecture of the server is designed to accommodate these features in the future.

6.0 SERVER-SIDE ROUTES

The server-side routes define the RESTful API endpoints that the front-end Angular application interacts with to perform various operations, such as user registration, group creation, and membership management. These routes are structured under the user-routes.js and group-routes.js files, ensuring a modular and organised approach.

6.1 User Routes (Defined in user-routes.js)

The user-routes.js file defines the core API endpoints related to user authentication, management, and reporting. These routes facilitate user interactions with the system, such as logging in, registering new accounts, updating user profiles, and handling reported users. Below is a breakdown of the key routes:

- **POST /users/auth:** Handles user authentication by validating the username and password provided in the request body. If the credentials are correct, it returns a success message along with the user's details (including permission and status); otherwise, an error message is returned.
- **POST /users/register:** Facilitates the registration of new users. It validates the request body to ensure the username and email are not duplicates using the validateRegistration utility. If the registration is successful, the new user is added to the users array, and a success response is returned with the user's details.
- **PUT /users/updateUser:** Allows users to update their profile information. It checks for duplicate usernames or emails before updating the user's data in the users array. If the update is successful, it returns the updated user information; otherwise, an appropriate error message is sent.
- **DELETE /users/deleteUser/:** Deletes a user by their ID. The route finds the user in the users array using the id parameter and removes the user if they exist. A success message is returned if the deletion is successful; otherwise, an error message is sent if the user is not found.
- **POST /users/reportUser:** Handles the reporting of users for inappropriate behaviour. The route expects a user ID and a reason for reporting. It creates a new ReportedUser instance and adds it to the reportedUsers array. A success message is returned if the user is reported successfully.
- **POST /users/removeReportedUser:** This route allows a reported user to be removed from the reportedUsers array, typically after the report has been resolved by the Super Admin. It looks for the reported user by their ID and removes them from the list if found.
- **GET /users/getReportedUsers:** Returns the list of all users who have been reported. This route is useful for Super Admins to review pending reports.
- **GET /users/users:** Returns a complete list of all users currently in the system. This route can be used to display user information for administrative purposes.
- **POST /users/upload-avatar/:id:** Allows a user to upload an avatar image, stores it in the server file system, updates the user's avatar path in the database, and returns the updated user data.
- **GET /users/avatar:** Retrieves a user's avatar image based on the file path provided in the query, validating the file path and returning the avatar file if it exists.

These routes ensure that user management and reporting functionalities are properly handled on the server side, providing the necessary API for front-end operations such as logging in, registering, and moderating user behaviour.

6.2 Group Routes (Defined in group-routes.js)

The group-routes.js file defines the core API endpoints for managing groups, channels, and user membership within those groups. These routes allow users to create groups, manage group admins and members, create channels, and handle user permissions and bans within the chat system. Below is a summary of the key routes:

- **POST /groups/addChannelMembers:** Adds the current user to a specific channel within a group, either as an admin or a member, based on the user's role within the group. If the user is a group creator or admin, they are added as a channel admin; otherwise, they are added as a member if they already belong to the group.
- **POST /groups/addGroupAdmin:** Promotes a user to an admin role in a specific group. This route moves the user from the group's member list to the admin list if they are not already an admin.
- **POST /groups/approveRegistration:** Approves a user's request to join a group. This route checks if the requesting user has the necessary permissions (group creator, super admin, or group admin) and adds the user to the group's members list.
- **POST /groups/banUser:** Bans a user from a group by removing them from the admins, members, and interested lists. The user is then added to the group's banned list to prevent further participation.
- **POST /groups/createChannel:** Creates a new channel within a group. Only group creators or admins are authorised to create channels. The channel is associated with the group and added to its channel list.
- **POST /groups/createGroup:** Allows users to create a new group, with the creator being automatically added as the group admin. This route is essential for establishing new chat groups and organising users into discussion spaces.
- **POST /groups/deleteChannel:** Deletes a specific channel from a group. Only group creators or admins are authorised to delete channels.
- **DELETE /groups/deleteGroup/:** Deletes a group by its ID. This removes the group and all associated channels from the system.
- **POST /groups/deregister:** Deregisters a user's interest in joining a group by removing them from the group's interested list.
- **POST /groups/leaveChannel:** Allows a user to leave a channel within a group. The user is removed from the channel's admins or members list, but channel creators cannot leave the channels they created.
- **POST /groups/leaveGroup:** Enables a user to leave a group. The user is removed from the group's admins or members list, depending on their role in the group.
- **POST /groups/register:** Allows a user to register their interest in joining a group. This route adds the user to the group's interested list for admin review and approval.
- **POST /groups/removeAdmin:** Demotes an admin from a group and adds them to the members list. Only group creators, super admins, or other admins are authorised to perform this action.

- **POST /groups/removeUser:** Removes a user from all group lists (admins, members, interested, and banned). This route is used for completely disassociating a user from the group.
- **GET /groups/groups:** Returns a list of all groups in the system. This route can be used to display available groups to users.
- **GET /groups/view-group/:** Retrieves detailed information about a specific group, including members, channels, and admins. If the user requesting the group is not part of the group, they will not receive the group's details.
- **POST /groups/addMessage:** Adds a new message to a specified channel within a group. Validates group and channel existence before appending the message and returning a success response.
- **GET /groups/getMessages:** Retrieves all messages for a specified channel within a group. If the group or channel is not found, an error message is returned.
- **GET /groups/group-image:** Retrieves an image from a specified group's directory based on the provided file path. If the file is not found, returns a 404 error.
- **POST /groups/uploadImage/:id:** Uploads an image to the specified group's directory. Validates file upload and group existence, then returns the image path upon successful upload.

These routes form the backbone of group and channel management in the chat system, enabling users to organise into groups and participate in channels while allowing admins and creators to manage group membership and permissions effectively.

7.0 CLIENT-SERVER INTERACTION

The interaction between the Angular front-end and the Node.js server is primarily facilitated through RESTful API calls. The client sends HTTP requests to the server to perform operations such as user authentication, group and channel management, and user reporting. The server responds with data that updates the Angular application's state, providing real-time feedback to the user.

7.1 Authentication and User Management

When a user logs in or registers, the Angular application communicates with the server via API routes such as /users/auth and /users/register. These routes validate the user's credentials or create a new user, and the response from the server is used to update the client's state, showing success or failure messages.

7.2 Group and Channel Management

The Angular front-end interacts with the server through routes like /groups, /createGroup, and /createChannel. When a user creates or modifies a group or channel, the client sends a request containing the necessary data (e.g., group name, creator). The server processes the request and sends back updated group or channel data, which is then rendered in the client.

7.3 User Reporting and Admin Actions

Users can report other users through the client interface, sending requests to the server via the /reportUser route. The server adds the reported user to a list of reported users, which is then accessible through the /getReportedUsers endpoint for Super Admins to review. Similarly, admin actions such as banning users or promoting group admins are handled through appropriate server routes, with the front-end sending the necessary data and the server updating the groups or users accordingly.

7.4 Data Synchronisation

For every interaction, whether it's joining a group, creating a channel, or updating user profiles, the server-side routes ensure that the requested changes are synchronised between the client and server. This ensures that the Angular app remains consistent with the latest data from the server, providing users with an up-to-date view of their interactions.