

**3813ICT**

Phase 2 Documentation

Lora Varagic

S5134143

**GitHub Structure**

**Branches**

The repository features one branch named, main. It has been my main development branch during the development of the assignment.

**Folders/files**

cypress: Includes files related to end-to-end testing (e2e)

data/db: This folder is dedicated to database-related files, which help backend data handling.

src: This folder contains the main source code of the application, with components and services related to Angular development. This is where most of the front-end logic is located.

cert.pem & key.pem: SSL certificate and private key files for securing HTTPS connections.

server.js, server.ts: These files handle the server logic and Socket.IO integration

Commits and troubleshooting

Throughout the course of development there have been 59 commits made to the repository. Frequent commits helped me troubleshoot and revert changes when code breaking bugs arose. Had I not had this version control in place I wouldn’t have been able to successful complete the assignment. Commits were not made as frequently as I would have liked, however I often got caught up coding in long sessions that I had forgotten to update the GitHub.

**Data Structures Used**  
On the client-side, users, groups, and messages are represented as objects. Users are represented by an “id”, “email”, “password” and “role”, which are stored in both local storage during sessions and in MongoDB on the server.

Groups represent different chat channels, with each group containing an `id` and a `name`. Messages include a “groupId”, “sender”, “messageContent”,” timestamp”, and an “imageUrl”. These messages are transmitted via Socket.IO in real-time and are then stored using MongoDB. On the server-side, MongoDB is used for storage, storing users, groups, and messages.

**Client and Server Responsibilities**  
The client-side (Angular) is responsible for handling user interaction, sending and receiving chat messages, managing video calls, and updating the UI in real-time. It uses services to communicate with the back-end server through both HTTP requests and WebSockets. The server-side (Node.js and MongoDB) handles API routes, real-time communication, and persistent data storage. It processes incoming HTTP requests for fetching and deleting messages and manages real-time message broadcasting through Socket.IO. MongoDB is used to store messages, groups, and users. The server also ensures that any changes in data, such as message deletion or updates, are synchronized across all clients.

**Routes and API**  
The server uses API routes to allow communication between the client and the database. The “/messages/:groupId” route (GET) fetches all messages for a specific group, returning them as a JSON array. The “/messages/:messageId” route (DELETE) is used to delete a specific message, by its unique message ID. WebSocket-based events handle real-time communication. The “message” event handles the sending and receiving of messages in real-time, while “userJoined” and “userLeft” events broadcast changes in user presence within a group. These routes ensure that the client can interact with the database effectively and keep the chat application synchronized in real-time.

**Angular Architecture**  
The Angular application is structured into components and services. The main chat interface is managed by the “Group2PageComponent”, which is responsible for rendering messages, sending chat data, and managing video calls. This component interacts with the “SocketService” to communicate with the server for real-time message updates. The “PeerService” manages video calls, and the “ImgUploadService” is responsible for handling image uploads. The application's routes ensure proper navigation between different group chat pages and facilitate role-based access for users.

**Client-Server Interaction**  
The interaction between the client and server occurs through both WebSocket communication and REST API calls. When a message is sent, the “SocketService” sends out the message to the server using Socket.IO. The server listens for the “message” event, stores the message in MongoDB, and broadcasts it back to all connected users. This makes sure that messages are displayed in real-time. Also, when a message is deleted, a HTTP DELETE request is sent to the server, which removes the message from MongoDB, which then updates the client-side UI by removing the message from chat.

**Testing**

Testing can be carried out by running npx cypress run in the root of the app, e.g chat-system