Chat Application with Video, Image Upload, and MongoDB Support

3813ICT Software Frameworks
s5270448 Kei Giliam

This is a real-time chat application developed as part of the 3813ICT Assignment Phase 2. It includes MongoDB integration, video chat with PeerJS, image uploads, and socket-based real-time messaging.

Introduction

This project demonstrates a chat system with advanced features such as real-time messaging using **WebSockets**, **video chat** using **PeerJS**, **image uploads**, and **MongoDB** integration for persistent data storage.

This chat application allows users to:

- Register, log in, and update profile pictures.
- Join chat channels and send text or image messages.
- Initiate video calls with online users.

Git Usage & Structure

- **Repository Structure**:
  - The repository is divided into client-side (Angular) and server-side (Node.js) codebases.
  - Frequent commits reflect new features and bug fixes, tracked via branches (e.g., `feature/chat`, `feature/video-call`).
  - Pull requests and merging strategies are used for integrating features.
  - A `README.md` provides project installation instructions.
  - GitHub is used to collaborate and keep version control organized.

Data Structures

1. Users Collection:

```
{
  "_id": "abc123",
  "username": "User1",
  "password": "hashed_password",
  "groups": ["general", "random"],
  "profileImage": "/uploads/profile1.jpg"
}
```

2. Groups Collection:

```
{
  "_id": "group1",
  "name": "General",
  "channels": ["general-chat", "random-chat"]
}
```

3. Channels Collection:

```
{
  "_id": "general-chat",
  "name": "General Chat",
  "group": "group1",
  "messages": []
}
```

4. Messages Collection:

```
{
  "_id": "msg123",
  "username": "User1",
  "message": "Hello, world!",
  "timestamp": "2024-10-17T00:00:00Z",
  "isImage": false
}
```

Client-Server Responsibility Breakdown

Client (Angular):

- Components handle the user interface and interactions.
- Services interact with the backend API and handle real-time socket connections.
- Image uploads and chat messages are handled through HTTP requests to the server.
- PeerJS is used on the client-side for initiating and receiving video calls.

Server (Node.js with MongoDB):

- The backend stores user, group, channel, and message data in MongoDB collections.
- Routes are exposed as REST APIs to manage users, channels, and groups.
- Sockets are used to support real-time chat and notifications.
- PeerJS server manages video call signaling.

Routes & APIs

Route	Method	Parameters	Description
/register	POST	{ username, password }	Register a new user.
/login	POST	{ username, password }	Login and generate a token.
/channels	GET	None	Retrieve all channels.
/channels/:id/messages	GET	{ id }	Get all messages from a channel.
/upload-profile	POST	FormData with profile image	Upload a user's profile image.
/upload-chat-image	POST	FormData with chat image	Upload a chat image as a message.

Angular Architecture

1. **Components**:
   - `LoginComponent`: Handles user login.
   - `RegisterComponent`: Handles new user registration.
   - `ChatComponent`: Manages chat, video, and image interactions.
   - `ChannelComponent`: Displays the available channels.
   - `ProfileComponent`: Allows users to update their profile.
2. **Services**:
   - `AuthService`: Manages authentication and token storage.
   - `ChatService`: Handles socket communication and API interactions.

Real-time Interaction with Sockets

- **Socket.IO** is used to manage real-time chat.
- When a user joins a channel, a notification is sent to other users in the channel.
- As new messages arrive, the chat history is instantly updated via sockets.
- When a user leaves, their status is updated in real-time.

PeerJS Video Chat Integration

1. Client-side setup:

- PeerJS is initialized in the `ChatComponent`.
- Each user gets a unique peer ID upon connection, shared with others through the server.
- Users can initiate a call by clicking on another user's video call button.

2. Server-side PeerJS Setup:

```
npx peerjs --port 9000
```

This server handles the signaling for Peer-to-Peer (P2P) connections.

3. Functionality:

- Once connected, both users can exchange video streams.
- Video streams are displayed in the `caller-video` element in the chat interface.

Image Uploads in Chat

1. How it works:

- Users can upload images as part of their chat messages.
- The uploaded image files are stored on the server under the `/uploads` directory.

2. Image Storage & Retrieval:

- File paths (e.g., `/uploads/chat-image.jpg`) are stored in MongoDB.
- When a user sends an image message, the image URL is sent to all users through the socket.

3. Image Upload Example (ChatComponent):

```
uploadImage(event: any) {
  const file = event.target.files[0];
  const formData = new FormData();
  formData.append('chatImage', file);

  this.chatService.uploadChatImage(formData).subscribe(
    () => console.log('Image uploaded successfully'),
    (error) => console.error('Failed to upload image:', error)
  );
}
```

Testing

1. Backend Tests (Jest):

- Unit tests are implemented to verify API routes and data storage.

Example:

```
test('POST /register creates a new user', async () => {
  const res = await request(app).post('/register').send({ username: 'user',
password: 'pass' });
  expect(res.statusCode).toBe(200);
});
```

2. Frontend Unit Tests (Karma):

- Components are tested to ensure form validation and button states.

Example:

```
it('should disable the register button if form is invalid', () => {
  component.user.username = '';
  component.user.password = '';
  fixture.detectChanges();
  expect(fixture.nativeElement.querySelector('button').disabled).toBeTruthy();
});
```

3. End-to-End Tests (Cypress):

- Tests to verify user flows, such as login, registration, and chat.

Example:

```
describe('Login Page', () => {
  it('should log the user in and redirect to profile', () => {
    cy.visit('/login');
    cy.get('input[name="username"]').type('testuser');F
    cy.get('input[name="password"]').type('password');
    cy.get('button').click();
    cy.url().should('include', '/profile');
  });
});
```

Conclusion

This chat application integrates:

- **MongoDB** for persistent data storage (users, channels, and messages).
- **Socket.IO** for real-time chat functionality.
- **PeerJS** for video chat between users.
- **Image uploads** for both profile pictures and chat messages.
- **Comprehensive testing** through unit, backend, and E2E tests.
- **Proper version control** using Git to track development progress.

This project demonstrates the combination of real-time communication, multimedia support, and a scalable backend.