Hangout - Technical Design Document

Table of Contents

1. Introduction and System Overview

Hangout is a real-time chat application developed using the MERN stack. The application enables users to register, log in, select an avatar, and communicate with other registered users in real time. It offers a seamless chatting experience with support for emojis.

Key Features:

- User registration and login with password hashing.
- Avatar selection using the Multiavatar API.
- Real-time messaging using Socket.IO.
- Deployment on Render (backend), AWS Amplify (frontend) and MongoDB Atlas(Database).

2. Architecture

Hangout follows a three-tier architecture comprising:
- **Frontend**: Built with React.js, responsible for user interaction and interface.
- **Backend API**: Developed using Node.js and Express.js, handling server-side logic and database operations.
- **Database**: MongoDB for storing user and message data.

Component Interactions:

- The frontend communicates with the backend API for user authentication, avatar selection, and messaging operations.
- The backend uses MongoDB for persistent storage of user and message data.
- Real-time communication is facilitated by Socket.IO between the frontend and backend.

3. Backend Design

Technologies Used: Node.js, Express.js, MongoDB, Mongoose, bcrypt, dotenv.

Project Structure:-

```
server/
├── controllers/
│   ├── messageController.js
│   └── userController.js
├── models/
│   ├── messageModel.js
│   └── userModel.js
├── routes/
│   ├── auth.js
│   └── messages.js
├── .env
└── index.js
```

API Design:

- **POST /api/auth/register**: Registers a new user.
- **POST /api/auth/login**: Logs in an existing user.
- **POST /api/auth/logout**: Logs out a user.
- **GET /api/auth/allUsers**: Fetches all registered users.
- **POST /api/messages/addMsg**: Sends a message.
- **GET /api/messages/getMsg**: Retrieves chat history between users.
- **POST /api/auth/setAvatar**: Sets a user's avatar.

Database Schema:

- User Model:
  username: String (required, unique, min: 3, max: 20),
  email: String (required, unique, max: 50),
  password: String (required, min: 8),
  isAvatarImageSet: Boolean (default: false),
  avatarImage: String (default: "").

- Message Model:
  message.text: String (required),
  users: Array,
  sender: ObjectId (ref: User, required).

4. Frontend Design

Technologies Used: React.js, Styled Components, React Router DOM, Socket.IO client.

Project Structure:-

```
public/
├ ── src/
│   ├ ── assets/
│   ├ ── components/
│   ├ ── pages/
│   ├ ── utils/
│   ├ ── App.js
│   ├ ── index.css
│   └ ── index.js
├ ── .env
```

Routing:

- /register: User registration page.
- /login: User login page.
- /setAvatar: Avatar selection page.
- /: Chat interface page.

5. Real-Time Chat Implementation

Real-time communication in Hangout is powered by Socket.IO. This enables instant message delivery between users.
The backend serves as the WebSocket server, while the frontend uses the Socket.IO client library.

6. Security Considerations

- Passwords are hashed using bcrypt before storage in the database.
- Plain text passwords are never logged or stored.

7. Deployment Plan

- **Backend Deployment**: Hosted on Render.
- **Frontend Deployment**: Hosted on AWS (Amplify).
- **Database Hosting**: MongoDB Atlas

8. Future Enhancements

- Adding Google Login for seamless authentication.
- Group chats to enable multiple users to communicate simultaneously.
- File sharing capabilities within chat.

9. Conclusion

Hangout combines real-time communication and modern web technologies to create a user-friendly chat application. Its scalable architecture and deployment on AWS ensure reliability, while future enhancements aim to improve user experience further.