ChatterBox System Design Document

Introduction

Purpose

ChatterBox is a messaging service prototype designed to facilitate effective communication through text messages, group chats, and AI-powered chatbots. The system aims to provide a reliable and user-friendly platform for individual users to communicate.

Scope

The primary focus is on core messaging functionalities, including user registration, authentication, message management, and group chats. Optional features like video/audio calls and AI chatbots may be added depending on resources and priorities.

Target Audience

Individuals seeking a reliable and intuitive messaging application.

Components

User Interface

- **Technology**: Next.js
- **Description**: A web application with a clean and intuitive interface for users to interact with the messaging service.
- **Libraries**: Bootstrap, Chakra UI, Socket.IO-client

Backend API

- **Technology**: Node.js and Express
- **Description**: RESTful APIs handling user authentication, message management, group chat functionality, and database interactions. Real-time communication is managed using Socket.IO.
- **Libraries**: bcrypt, Cloudinary, JWT, Socket.IO

Database

- **Technology**: MongoDB

- **Description**: Used to store user data, messages, and other relevant information.

Additional Libraries

- **Cloudinary**: For image and video storage.
- **JWT**: For authentication.
- **bcrypt**: For password hashing.
- **Socket.IO**: For real-time messaging.

Data Flow

1. **User Registration**
    - Users create accounts by providing their information, which is stored in the MongoDB database.
2. **Authentication**
    - Users log in with their credentials, receiving JWT tokens for authentication.
3. **Message Sending**
    - Users send messages to other users or groups. Messages are stored in the database and broadcast in real-time to relevant recipients.
4. **Message Receiving**
    - Users receive messages in real-time, and the UI is updated accordingly.
5. **Group Chats**
    - Users can create or join groups. Messages within groups are broadcast to all group members.
6. **Real-Time Updates**
    - Messages are delivered to users in real-time, ensuring a seamless communication experience.

Non-Functional Requirements

1. **Performance**
    - The system should provide a fast and responsive user experience, even under heavy loads.
2. **Security**
    - Sensitive user data, such as passwords, should be encrypted and protected against unauthorized access.
3. **Reliability**
    - The system should be highly available and resilient to failures.
4. **Maintainability**
    - The codebase should be well-structured and easy to maintain and update.

Additional Features

1. **AI Chatbot**
    - An AI-powered chatbot can be integrated to provide automated responses to user queries.

Prototype Setup

Cloning the Repository

1. **Access the Repository**
    - Obtain the URL of the project repository from your version control system (e.g., GitHub).
2. **Clone the Repository**
    - Use the terminal or command prompt to clone the repository to your local machine:
      bash
      Copy code
      `git clone https://github.com/nay-hey/ChatterBox.git`

Setting Up the Environment

1. **Frontend Environment Configuration**
    - Create a `.env.local` file in the root directory of your frontend project:
    - `PORT=3000`
    - `DB_URI= #db url`
    - `JWT_SECRET= #secret key`
    - `JWT_EXPIRE=5d`
    - `NEXT_PUBLIC_GEMINI_API_URL=https://localhost:3000/api/gemini`
    - `NEXT_PUBLIC_GEMINI_API_KEY= #enter key`
2. **Backend Environment Configuration**
    - Create a `.env` file in the root directory of your backend project:
    - `PORT=5000`
    - `DB_URI= #db url`
    - `JWT_SECRET= #secret key`
    - `JWT_EXPIRE=5d`
    - `COOKIE_EXPIRE=5`
    - `NODE_ENV=development`
    - `CLOUDINARY_CLOUD_NAME= #name`
    - `CLOUDINARY_API_KEY= #api key`

- `CLOUDINARY_API_SECRET= #secret key`
- `FRONTEND_URL=http://localhost:3000`

Testing

Frontend Testing

1. **Start the Development Server**
   - Ensure the frontend server is running on port 3000 by executing:
   - `cd frontend`
   - `npm install –force`
   - `npm run dev`
   - Open a web browser and navigate to `http://localhost:3000` to test the frontend features. The frontend should interact with the backend APIs as expected.

Backend Testing

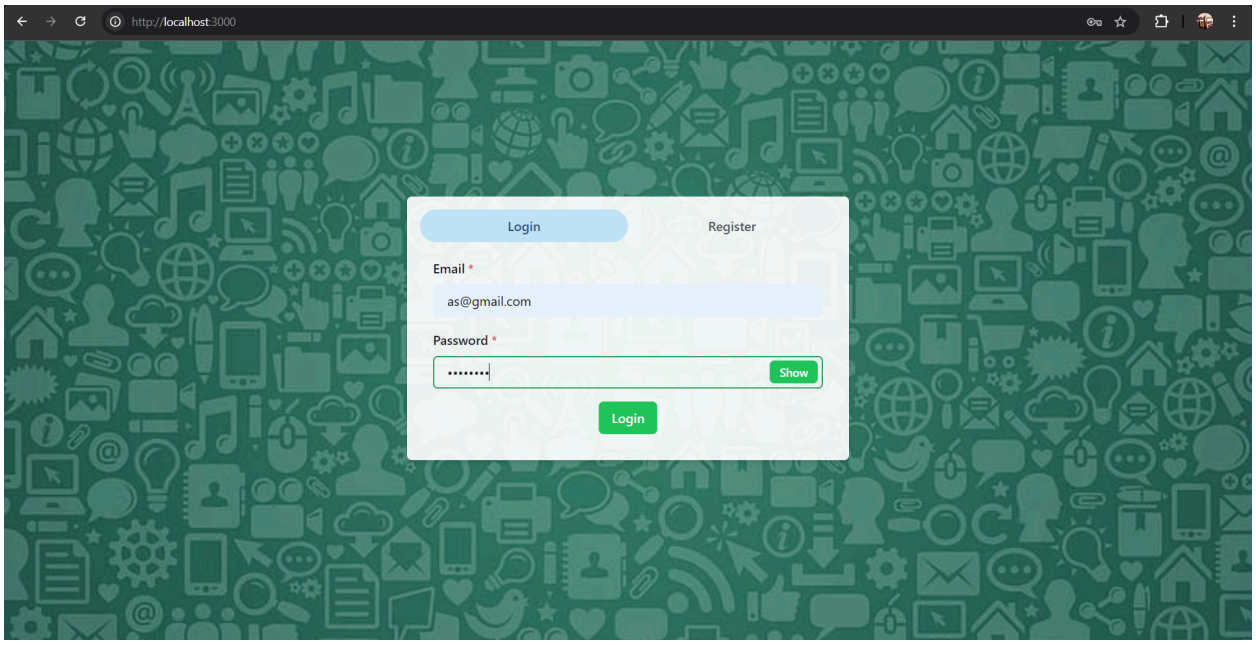
1. **Start the Backend Server**
   - Ensure the backend server is running on port 5000 by executing:
   - `cd backend`
   - `npm install`
   - `node index.js`
2. **Integrated Testing**
   - With both servers running, interact with the application through the frontend interface. This approach allows you to test the complete end-to-end functionality, including user registration, authentication, message sending/receiving, and group chat features.
   - Monitor the browser console and network requests to check for any issues or errors.

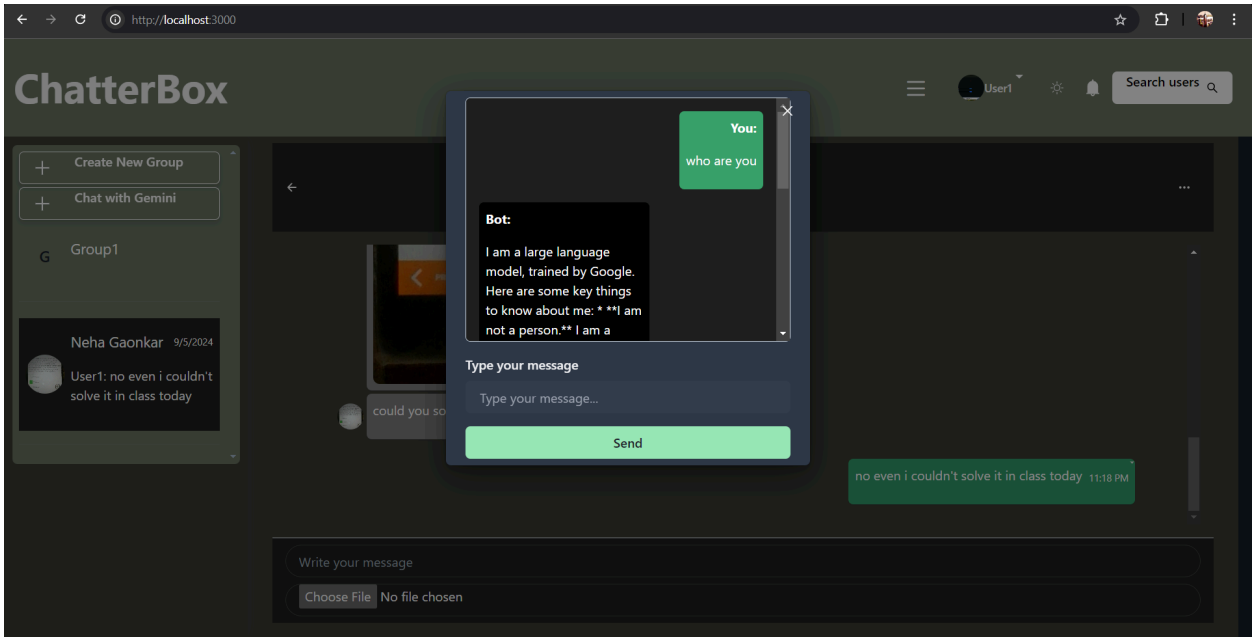
Alternative Testing Methods

1. **API Endpoint Testing**
   - While integrated testing is the primary method, you can also use tools like Postman or Curl for testing individual API endpoints if needed. This can be useful for debugging or testing specific backend functionalities in isolation.

Conclusion

This document provides a comprehensive overview of the ChatterBox prototype, including its system design, setup instructions, and the libraries used. The backend server runs on port 5000, and the frontend server runs on port 3000. Ensure both servers are running simultaneously to test the full functionality of the application.





Interactive Demo of ChatterBox