Project Overview

This document outlines the backend, frontend, database design and the deployment process of MERN stack web application that uses Express.js for the backend, with MongoDB as the database and Socket.io for real-time communication.

GitHub Repo: https://github.com/amanrock005/fullStackChatApp

Backend

/backend

```
|--- /controllers      // contains the logic for handling HTTP requests
|--- /lib              // contains reusable utilities (e.g DB connection, cloudinary config)
|---/middlewares       // contains middleware function for authentication and validation
|---/models            // contains API route definitions
|--index.js            // main server entry point
```

index.js

It is the entry point for the backend application, where we initialize the Express server and configure middleware, routes, and socket connection

- Express.js is used to handle HTTP request and route them to the appropriate controllers.
- Dotenv is used to manage environment variables.
- Cookie-parser handles cookie parsing to manage user sessions.
- Cors is configured to allow cross-origin requests between the frontend and backend.
- Socket.io is integrated to allow real-time messaging capabilities.
- Routes are modularized into different files
    - auth.route.js for authentication
    - Message.route for messaging
- A connection to mongodb is established via the connectDB() function from lib/db.js

Middleware (auth.middleware.js)

Middleware is used for authentication and route protection. The middleware ensures that only authenticated users can access protected routes.

Routes (auth.routes.js & message.route.js)

1. auth.route.js: Handles user authentication (signup, login, logout, profile updates).
2. message.route.js: Handles message-related operations(get the list of users, send message to them)

Controllers (auth.controller.js, message.controller.js)
It contains the logic of the HTTP request that is defined in the auth.route.js file. It handles the functionality of how to login a user, logout out, signup etc. Message controller file to send messages between sender and receiver.

Database design

MongoDB is an open-source, NoSQL database designed for scalability and flexibility. It stores data in a JSON-like format called BSON, which allows for dynamic and hierarchical data structure.

Document is the basic unit of data in MongoDB. It is similar to a record in RDBMS. Collection is a group of documents. It is similar to a table in RDBMS.

User model: To store user fullname, email, password and profilepic.
Message model: Stores receiverId, senderId, textmessage, attachment.

Frontend

/frontend

```
|--- /components        // reusable UI components
|--- /pages             // page-level components for routing
|---/lib                // utility function and helper libraries
|---/stores             // state management using Zustand
|--App.jsx              // Root application component
|---main.jsx            // Entry point of the React app
```

Scripts

dev - starts the development server - npm run dev
build - builds the production-ready app - npm run build
lint - runs ESLint to check code quality
preview - starts a local server to preview the production build - npm run preview

Dependencies

1. Core libraries:
    - React: Used for building UI.

- React router dom: for client-side routing.
- Zustand: State management library.

2. UI Enhancements:
    - DaisyUI: Tailwind CSS-based component library for pre-styled elements.
    - TailwindCSS: Utility-first CSS framework.

3. Network
    - Axios: for making API requests.
    - React Hot Toast: for showing toast notification.

4. Real-time communication:
    - Socket.io client: for real-time communication with the backend.

5. Developer tools:
    - ESLint: linting and code quantity.
    - Vite: Development server and build tools.

Hosting

Render is a fully managed cloud platform designed for deploying web applications, API's databases, and static sites with minimal configuration.

Hosting web service on render

1. Create a render account
2. Connect your git repo
3. Create a new service
4. Specify the branch
5. Set the build command
6. Set the start command
7. Add env variables
8. Deploy app
9. Monitor and manage