Interest Buddy

Problem Statement:

Develop a basic social networking platform that includes the following features:

- Allow users to join and leave the network.
- Enable users to create a simple profile with details such as name, contact information, interests, and hobbies.
- Facilitate messaging between users.
- Help users find others with similar interests.

Project Overview: The project, InterestBuddy is a simple social networking application designed to facilitate connections among individuals based on shared interests and proximity. Built using the MERN stack (MongoDB, Express.js, React, and Node.js) along with additional technologies like Socket.io, TailwindCSS, Daisy UI, and Zustand, InterestBuddy offers a seamless and interactive platform for users to join, create profiles, communicate, and find people with similar interests.

Purpose: The primary purpose of InterestBuddy is to provide users with a platform where they can connect with others who share similar interests. The application aims to foster community building and enhance social interactions by leveraging common hobbies and interests.

Main Features:

User Registration and Authentication:

Users can join the network by creating an account and logging in securely.
Users can also leave the network by deactivating their account.
Authentication and authorization are managed using JWT.

Profile Management:

Users can create their profiles with personal information such as name, contact details, interests, and hobbies.

Messaging:

Users can send and receive messages, enabling direct communication within the network.
Real-time chatting using Socket.io for instant messaging.
Sound notifications for incoming messages to alert users.

Interest Matching:

The application provides functionalities to find people based on shared interests.

Online Status:

The application shows whether a user is online or offline, providing real-time updates on user status.

Global State Management:

Zustand is used for managing global state across the application.

Error Handling:

Robust error handling is implemented on both the server and the client.

Key Outcomes:

- **Enhanced Connectivity:** InterestBuddy successfully creates an environment where users can find and interact with like-minded individuals, enhancing social connections and community building.
- **User-Friendly Interface:** The intuitive design and user-friendly interface of InterestBuddy ensure a smooth user experience, encouraging active participation and engagement.
- **Scalable Architecture:** The use of the MERN stack along with additional technologies provides a robust and scalable architecture, ensuring that the application can handle increasing numbers of users and data efficiently.
- **Secure Communication:** Implementing secure authentication and data handling practices ensures that user information and communications remain private and protected.

Background

With the rapid advancement of technology and the internet, social networking has become a crucial aspect of modern life. People seek platforms where they can connect with others who share similar interests and hobbies. This project, InterestBuddy, aims to fulfill this need by providing a simple and effective social networking application.

Objectives

- To create a user-friendly social network platform.
- To enable users to join and leave the network easily.
- To allow users to create their profiles.
- To provide messaging capabilities for user communication.
- To facilitate finding people with similar interests.
- To display user's online status and enable real-time chatting.
- To ensure secure authentication and authorization.
- To handle errors effectively on both the client and server sides.

Scope

This project focuses on developing a web-based social networking application using the MERN stack with additional technologies such as Socket.io for real-time communication, TailwindCSS and Daisy UI for styling, and Zustand for state management. The application will include core features like user registration, messaging, interest-based user searching, and real-time chat functionality.

Technology Stack

MERN Stack Overview

- **MongoDB:** A NoSQL database for storing user profiles, messages, and other data.
- **Express.js:** A web application framework for building the backend API.
- **React:** A JavaScript library for building the user interface.
- **Node.js:** A JavaScript runtime for executing server-side code.

Additional Technologies

- **Socket.io:** For real-time, bidirectional communication between clients and server.
- **JWT (JSON Web Tokens):** For secure authentication and authorization.
- **Axios:** For making HTTP requests from the frontend to the backend.
- **TailwindCSS:** For utility-first CSS framework to build custom designs.
- **Daisy UI:** For a set of accessible and customizable components.
- **Zustand:** For global state management in React applications.

System Architecture

High-Level Architecture

The application follows a client-server architecture:

- **Frontend (Client):** Built with React, handles user interactions and displays data.
- **Backend (Server):** Built with Node.js and Express, handles business logic and database interactions.
- **Database:** MongoDB stores user profiles, messages, and other persistent data.

Database Schema

User Schema

The **User** schema represents a user in the social network application. Each user has personal information and preferences stored in this schema.

Fields:

- **fullName:** The full name of the user (String, required).
- **username:** A unique username for the user (String, required, unique).
- **password:** The user's password (String, required, minimum length of 6).
- **interest1:** The user's primary interest (String, required).
- **interest2:** The user's secondary interest (String, required).
- **gender:** The user's gender (String, required, must be either "male" or "female").

- **profilePic:** URL to the user's profile picture (String, optional, defaults to an empty string).
- **createdAt:** The date and time when the user was created (automatically managed by Mongoose).
- **updatedAt:** The date and time when the user was last updated (automatically managed by Mongoose).

Message Schema

The **Message** schema represents a message sent between users.

Fields:

- **senderId:** The ID of the user who sent the message (ObjectId, references the **User** schema, required).
- **receiverId:** The ID of the user who receives the message (ObjectId, references the **User** schema, required).
- **message:** The text content of the message (String, required).
- **createdAt:** The date and time when the message was created (automatically managed by Mongoose).
- **updatedAt:** The date and time when the message was last updated (automatically managed by Mongoose).

Conversation Schema

The **Conversation** schema represents a conversation between users, which can include multiple messages.

Fields:

- **participants:** An array of user IDs participating in the conversation (Array of ObjectIds, references the **User** schema).
- **messages:** An array of message IDs that are part of the conversation (Array of ObjectIds, references the **Message** schema, defaults to an empty array).
- **createdAt:** The date and time when the conversation was created (automatically managed by Mongoose).
- **updatedAt:** The date and time when the conversation was last updated (automatically managed by Mongoose).

API Design

The API for the Interest Buddy application is designed to handle user authentication, profile management, messaging, and interest-based user matching. The API endpoints are structured to be RESTful, ensuring clear and predictable interactions between the client and server.

User Authentication and Management Endpoints

The authentication and user management endpoints handle user registration, login, and logout operations. These endpoints are managed by the **auth.controller.js** file.

Endpoints:

- **POST /api/auth/signup**

  Description: Registers a new user.

  Response:

  - **201 Created:** User successfully registered.
  - **400 Bad Request:** Validation error or missing required fields.

- **POST /api/auth/login**

  Description: Authenticates a user and returns a JWT token.

  Response:

  - **200 OK:** Returns JWT token.
  - **401 Unauthorized:** Invalid credentials.

- **POST /api/auth/logout**

  Description: Logs out a user.

  Response:

  - **200 OK:** User successfully logged out.

Messaging Endpoints

The messaging endpoints handle the retrieval and sending of messages between users. These endpoints are managed by the **message.controller.js** file and protected by the **protectRoute** middleware to ensure only authenticated users can access them.

Endpoints:

- **GET /api/messages/:id**
    - **Description:** Retrieves messages for a conversation with a specific user.
    - **Parameters:**
        - **id:** The ID of the user with whom the conversation is held.
    - **Response:**
        - **200 OK:** Returns an array of messages.
        - **401 Unauthorized:** User is not authenticated.
        - **404 Not Found:** Conversation not found.
- **POST /api/messages/send/:id**
    - **Description:** Sends a message to a specific user.
    - **Parameters:**
        - **id:** The ID of the user to whom the message is being sent.
    - **Response:**
        - **201 Created:** Message successfully sent.
        - **401 Unauthorized:** User is not authenticated.
        - **400 Bad Request:** Validation error or missing required fields.

Middleware: protectRoute

The **protectRoute** middleware ensures that only authenticated users can access certain endpoints. It verifies the presence and validity of a JWT token in the request headers.

Middleware Implementation: The middleware checks for the JWT token in the request headers. If the token is valid, it decodes the token to get the user's ID and fetches the user data (excluding the password) to attach to the request object. If the token is missing or invalid, it responds with a 401 Unauthorized status.

Summary of API Endpoints

- **User Authentication and Management:**
  - **POST /api/auth/signup** - Registers a new user.
  - **POST /api/auth/login** - Authenticates a user and returns a JWT token.
  - **POST /api/auth/logout** - Logs out a user.
- **Messaging:**
  - **GET /api/messages/:id** - Retrieves messages for a conversation.
  - **POST /api/messages/send/:id** - Sends a message to a user.

These endpoints ensure secure and efficient handling of user authentication and messaging within the Interest Buddy application.

Frontend Structure

- **Components:**
  - **App**: Main application component.
  - **Register**: User registration form.
  - **Login**: User login form.
  - **MessageList**: Display messages.
  - **SendMessage**: Form to send messages.

UserList: List of users with similar interests.

Chat: Real-time chat interface.

- **State Management:**

  Using Zustand for global state management.
- **Styling:**

  TailwindCSS and Daisy UI for styling and UI components.

Implementation

Backend Development

- **Setup and Configuration:**

  Initialize Node.js project with Express.
  Connect to MongoDB using Mongoose.
  Set up Socket.io for real-time communication.
- **API Implementation:**

  Define routes and controllers for user authentication, profile management, messaging, interest matching, and online status.
  Implement JWT for secure authentication and authorization.
- **Data Validation and Error Handling:**

  Use middleware for input validation and error handling on the server side.

Frontend Development

- **Setup and Configuration:**

  Initialize React project.
  Configure React Router for navigation.
- **Component Development:**

  Develop and style components for registration, login, profile management, messaging, user listing, and chat.
- **State Management:**

  Implement global state management using Zustand.

- **API Integration:**

  Use Axios to connect frontend components with backend APIs.
  Integrate Socket.io for real-time chat functionality.
- **Styling:**

  Utilize TailwindCSS and Daisy UI for responsive and accessible design.

Middleware and Other Tools

- **Middleware:**

  JWT middleware for protecting routes.
  Error handling middleware.
- **Additional Tools:**

  Axios for HTTP requests.
  TailwindCSS and Daisy UI for responsive design.
  Socket.io for real-time communication.
  Zustand for state management.

Features and Functionality

User Registration and Authentication

- Users can create an account with a unique username and password.
- Users can log in to the network securely.
- Secure authentication and authorization using JWT.

Profile Management

- Users can create a profile with their name, username, interests, and hobbies.
- Users can update their profile information at any time.

Messaging

- Users can send and receive messages.
- Real-time chatting using Socket.io for instant messaging.
- Sound notifications for incoming messages to alert users.

Interest Matching

- Users can find others with similar interests.
- The application has a search feature to search for people based on shared interests.

Online Status

- The application shows whether a user is online or offline, providing real-time updates on user status.

Global State Management

- Zustand is used for managing global state across the application.

Error Handling

- Robust error handling is implemented on both the server and the client.

Deployment

The deployment strategy for the Interest Buddy application ensures that both the frontend and backend are accessible and performant for users.

Deployment Platform

For deploying Interest Buddy, we used ***Render***, a modern cloud platform that simplifies the deployment process for web applications and offers a seamless experience for both the frontend and backend services.

Hosting Services

- **Backend Hosting:** The backend services, including the API and database connections, are hosted on Render. This setup ensures that the server-side logic is efficiently managed and scalable.
- **Frontend Hosting:** The frontend of the application, built with React, is also hosted on Render, allowing for streamlined management and deployment of static assets.

Future Enhancements

- Implement features like group chats and media sharing.
- Implement Update Profile feature