

**HabitHive Project Documentation**

**Project Overview**

**HabitHive** is a habit tracking web application that helps users set, track, and maintain daily habits. The application features:

* User authentication (login, registration)
* Habit creation, toggling, and tracking
* Habit streak tracking
* Notifications and reminders for habits
* Frontend built with React (TypeScript, Redux)
* Backend built with Node.js and Express
* MongoDB for data storage
* Email reminders and notifications

**Tech Stack**

**Frontend**

* **React**: A JavaScript library for building user interfaces, particularly Single Page Applications (SPA).
* **TypeScript**: A superset of JavaScript that allows static typing to improve code quality and prevent bugs.
* **Redux**: A predictable state container for JavaScript apps, used to manage the state of the application across multiple components.
* **React Router**: A routing library for React that enables navigation between pages without reloading the page.
* **Axios**: A promise-based HTTP client for the browser and Node.js, used to make API requests.
* **React Toastify**: A library for showing toast notifications in React applications.
* **Material-UI**: A popular React component library for building modern and responsive UI designs.

**Backend**

* **Node.js**: A JavaScript runtime built on Chrome's V8 engine, used for building the backend API.
* **Express.js**: A minimal and flexible Node.js web application framework used to handle HTTP requests and define API routes.
* **MongoDB**: A NoSQL database used for storing user data, habits, and their associated histories.
* **Mongoose**: An ODM (Object Data Modeling) library for MongoDB and Node.js, used for schema definition and interaction with the database.
* **Nodemailer**: A module for Node.js that allows sending emails via SMTP.
* **Moment.js**: A JavaScript library used for parsing, validating, manipulating, and displaying dates and times.

**Deployment**

* **Docker**: A platform for developing, shipping, and running applications in containers. Used to containerize both the frontend and backend applications for easy deployment and scalability.

**Features Implemented**

**1. User Authentication**

* **Login & Registration**: Users can register with their email and password, and log in to access their personalized dashboard.
  + **Frontend**: React forms with controlled components, form validation, and state management using Redux.
  + **Backend**: Express routes for handling registration (POST /api/register) and login (POST /api/login), using JWT (JSON Web Tokens) for user authentication.

**2. Habit Management**

* **Habit Creation**: Users can create new habits with details such as title, description, frequency, and reminder time.
  + **Frontend**: Form handling for habit creation.
  + **Backend**: Express API (POST /api/habits) to create a habit and store it in the MongoDB database.
* **Habit Toggle**: Users can mark habits as complete or incomplete. Streaks are tracked based on consecutive days of completion.
  + **Backend**: The toggleHabit controller handles marking habits as completed, updating the habit streak, and storing completion history.
  + **Frontend**: A toggle button for each habit that updates its state and calls the toggleHabit API endpoint.

**3. Habit Streak Tracking**

* **Streaks**: Users are given a streak count that increases when they mark the habit as complete for consecutive days.
  + **Backend**: The streak is calculated based on the lastCompletedDate and compared with today's date. If the habit was completed on consecutive days, the streak increases.
  + **Frontend**: The streak count is displayed on the dashboard for each habit.

**4. Habit Reminders and Notifications**

* **Reminder System**: A fixed reminder system was implemented to notify users about pending habits based on a specific schedule.
  + **Frontend**: A ReminderNotifier component that checks every minute whether a habit's reminder time matches the current time and shows a toast notification if it does.
  + **Backend**: (Not implemented as per your requirement). The frontend handles the reminder notifications, and there is no need for a backend cron job for reminders.

**Note**: The toast notifications are shown every minute (as per the set interval in the frontend code), and reminders pop up for the habits that match the set reminder time.

**Libraries and Packages Used**

**Frontend**

1. **React** (react, react-dom) – For building the user interface and rendering the app.
2. **Redux** (@reduxjs/toolkit, react-redux) – For managing the global state of the app.
3. **React Router** (react-router-dom) – For routing between pages.
4. **Axios** (axios) – For making HTTP requests to the backend API.
5. **React Toastify** (react-toastify) – For showing toast notifications to the user.
6. **Material UI** (@mui/material) – For pre-built components to create a consistent and responsive UI.
7. **Moment.js** (moment) – For handling date manipulation and comparison in habit tracking.
8. **TypeScript** (typescript, @types/react, @types/react-dom, etc.) – For static typing and type safety in the project.

**Backend**

1. **Node.js** – The JavaScript runtime used for the backend.
2. **Express.js** – Web framework for building the backend API.
3. **Mongoose** – ODM library for MongoDB, used to define schemas and interact with the database.
4. **Bcrypt.js** – For hashing user passwords before storing them in the database.
5. **JSON Web Token (JWT)** (jsonwebtoken) – For user authentication and generating secure tokens for logged-in users.
6. **Nodemailer** – For sending email reminders to users.
7. **Moment.js** – For handling date manipulations on the backend.
8. **Dotenv** – For loading environment variables securely from a .env file.

**Implementation Details**

**1. Setting Up the Backend**

1. Install dependencies:

bash

Copy code

npm install express mongoose bcryptjs jsonwebtoken nodemailer moment dotenv

1. **Create the User Model**:
   * The User schema stores user information such as email, password, and JWT for authentication.
2. **Create the Habit Model**:
   * The Habit schema includes fields such as title, description, streak, completed, lastCompletedDate, and history.
3. **Set Up API Routes**:
   * **POST /api/register**: Handles user registration.
   * **POST /api/login**: Handles user login and generates JWT.
   * **POST /api/habits**: Handles creating habits.
   * **PATCH /api/habits/:id/toggle**: Handles toggling habit completion and streaks.
4. **Send Email Reminders (Optional)**:
   * If you choose to implement email notifications, set up Nodemailer as shown earlier and schedule email reminders (if desired).

**2. Setting Up the Frontend**

1. Install dependencies:

bash

Copy code

npm install react-redux @reduxjs/toolkit axios react-router-dom react-toastify material-ui moment

1. **Create Redux Slices**:
   * **UserSlice**: Handles the user state (logged-in status).
   * **HabitSlice**: Handles the list of habits, habit completion, and streak data.
2. **Create Pages and Components**:
   * **Login & Register Pages**: Form components for logging in and registering users.
   * **Dashboard**: Displays habits, their streaks, and the toggle button to mark them as complete.
   * **ReminderNotifier**: Component that checks habit reminder times and displays toast notifications.

**Project Directory Structure**

pgsql

Copy code

HabitHive/

├── backend/

│ ├── controllers/

│ │ ├── habitController.ts

│ │ └── userController.ts

│ ├── models/

│ │ ├── habit.ts

│ │ └── user.ts

│ ├── routes/

│ │ ├── habitRoutes.ts

│ │ └── userRoutes.ts

│ ├── config/

│ │ └── nodemailer.ts

│ ├── server.ts

│ └── .env

├── frontend/

│ ├── src/

│ │ ├── components/

│ │ │ ├── HabitCard.tsx

│ │ │ ├── LoginForm.tsx

│ │ │ └── ReminderNotifier.tsx

│ │ ├── pages/

│ │ │ ├── Dashboard.tsx

│ │ │ ├── Login.tsx

│ │ │ └── Register.tsx

│ │ ├── redux/

│ │ │ ├── habitSlice.ts

│ │ │ ├── userSlice.ts

│ │ │ └── store.ts

│ │ ├── utils/

│ │ └── App.tsx

└── docker-compose.yml

**Conclusion**

This is a full-featured habit tracker application with secure user authentication, habit creation and tracking, and reminder notifications. We've used modern tools like React, Redux, and Node.js, with a focus on security, scalability, and usability.

The next steps would involve refining the reminder system, adding more user-friendly features like habit analytics, and further improving the app's overall UI/UX.