Implementation Road map

1. Implement Landing Page

- **Frontend Implementation**:
    - **Step 1**: Create the basic layout for the landing page using **React**, **Material-UI**, and **Tailwind CSS** for styling.
    - **Step 2**: Design the page layout with sections like the header, hero section, feature highlights, and footer.
    - **Step 3**: Make the landing page fully responsive (using Tailwind CSS for responsive classes).
    - **Step 4**: Optionally, add any interactive elements or animations for the landing page (e.g., buttons, scroll animations).
- **Backend Implementation**:
    - **No API needed**: The landing page does not require any API interactions at this point.

---

2. Implement Login Page

- **Frontend Implementation**:
    - **Step 1**: Create the login form using **React Hook Form** for form handling and **Material-UI** for UI components (input fields, buttons).
    - **Step 2**: Add **Yup validation** for validating user input (e.g., email and password).
    - **Step 3**: Style the form using **Tailwind CSS** for a responsive layout.
    - **Step 4**: Implement form submission logic using **Axios** to send a request to the backend API.
    - **Step 5**: Display error messages for incorrect login credentials (based on API response).
- **Backend Implementation**:
    - **Step 1**: Implement the login API route using **NestJS**.
    - **Step 2**: Set up **JWT authentication** to handle login requests. Upon successful login, return a JWT token to the frontend.
    - **Step 3**: Validate login credentials against the **PostgreSQL** database (ensure the user exists, password matches).
    - **Step 4**: Return appropriate error responses for invalid credentials (e.g., 401 Unauthorized).

3. Implement User Registration Page

- **Frontend Implementation**:
    - **Step 1**: Create the registration form using **React Hook Form** for form handling and **Material-UI** components.
    - **Step 2**: Add **Yup validation** for validating user input (e.g., email, password, confirm password).
    - **Step 3**: Style the registration form using **Tailwind CSS** for responsiveness.
    - **Step 4**: Implement form submission logic using **Axios** to send the registration data to the backend API.
    - **Step 5**: Display success messages upon successful registration or error messages based on API responses.
- **Backend Implementation**:
    - **Step 1**: Implement the user registration API using **NestJS**.
    - **Step 2**: Add logic for storing new user data in **PostgreSQL** using **TypeORM**.
    - **Step 3**: Hash the password using **bcrypt** before storing it in the database.
    - **Step 4**: Ensure unique email validation and return appropriate error responses for duplicate entries.
    - **Step 5**: Return a JWT token upon successful registration, allowing the user to be logged in automatically.

---

4. Implement Dashboard (After Login)

- **Frontend Implementation**:
    - **Step 1**: Create the dashboard layout using **Material-UI** components for the UI and **Tailwind CSS** for styling.
    - **Step 2**: Use **React Router** to handle navigation within the dashboard (e.g., to subpages like profile, books, etc.).
    - **Step 3**: Display user-specific information (e.g., books, highlights) by making an API call using **Axios**.
    - **Step 4**: Ensure the user is authenticated by checking the presence of a JWT token in localStorage or a similar mechanism.
- **Backend Implementation**:
    - **Step 1**: Implement a JWT-protected API route that returns user-specific data (e.g., list of books, highlights).
    - **Step 2**: Use **NestJS guards** to protect the dashboard routes, ensuring only authenticated users can access the data.
    - **Step 3**: Query the **PostgreSQL** database for the user's information using **TypeORM** and return it to the frontend.

5. Implement Book and Highlight Management (CRUD)

Book Management:

- **Frontend Implementation**:
    - **Step 1**: Create the interface for adding, editing, and deleting books using **React Hook Form** for handling forms.
    - **Step 2**: Implement API requests using **Axios** to interact with the backend (e.g., create a new book, edit book details, delete a book).
    - **Step 3**: Style the book management interface using **Material-UI** and **Tailwind CSS**.
    - **Step 4**: Display a list of user's books on the dashboard, fetched via API.
- **Backend Implementation**:
    - **Step 1**: Implement CRUD API routes for book management (create, read, update, delete) using **NestJS**.
    - **Step 2**: Store book data in the **PostgreSQL** database, ensuring proper relationships between users and books.
    - **Step 3**: Add necessary validation (e.g., check if the user is authorized to edit/delete a specific book).
    - **Step 4**: Return success/error responses based on API actions (e.g., book successfully created or deleted).

Highlight Management:

- **Frontend Implementation**:
    - **Step 1**: Create the interface for adding, editing, and deleting highlights for a book using **React Hook Form**.
    - **Step 2**: Implement API requests using **Axios** to create, update, or delete highlights.
    - **Step 3**: Display the highlights related to a specific book, fetched from the backend.
- **Backend Implementation**:
    - **Step 1**: Implement API routes for managing highlights (create, update, delete) using **NestJS**.
    - **Step 2**: Store highlight data in the **PostgreSQL** database and relate them to books.
    - **Step 3**: Ensure that only authenticated users can manage their own highlights.

6. Implement Profile Page

- **Frontend Implementation**:
    - **Step 1**: Create the profile page layout using **Material-UI** and **Tailwind CSS**.
    - **Step 2**: Allow the user to update their profile information (e.g., name, email, password) using **React Hook Form**.
    - **Step 3**: Send form data to the backend API using **Axios** for profile updates.
- **Backend Implementation**:
    - **Step 1**: Implement a protected API route for updating user profile information.
    - **Step 2**: Ensure user data is updated securely in **PostgreSQL**.
    - **Step 3**: Return appropriate responses (e.g., profile updated successfully, invalid password).

---

7. Implement Logout Functionality

- **Frontend Implementation**:
    - **Step 1**: Add a logout button to the dashboard or profile page.
    - **Step 2**: On logout, clear the JWT token from localStorage (or session) and redirect the user to the login page.
    - **Step 3**: Invalidate user session data on the frontend.
- **Backend Implementation**:
    - **No need for an API**: Since JWT is stateless, there's no need for backend logic to handle logout.