Cook Book

introduction

Project Title : cook book

Team id :NM2025TMID38352

Team Leader :KARTHIKEYAN .S

Team members :

Karthikeyan.S

Kubenthiran.k

Hareesh.V

Madhavan.S

Project Overview

CookBook is designed to revolutionize the way people discover, organize, and create recipes in a digital era. It serves as a comprehensive platform for culinary exploration, aiming to bridge the gap between traditional recipe books and modern web technology.The primary purpose of CookBook is to offer a robust, user-friendly application where both amateur cooks and professional chefs can easily find, organize, customize, and share recipes. By providing intuitive search capabilities, personalized features, and community sharing tools, CookBook empowers users to enhance their cooking experience, improve kitchen efficiency, and spark culinary creativity.

Purpose:

- Simplify recipe discovery with intelligent search functionality.
- Organize user-favorite recipes into categories for easy retrieval.
- Foster a community of culinary enthusiasts to share and collaborate.
- Enable users to contribute their own recipes and experiences.
- Support customization of the cooking experience based on individual preferences.

Features

1. Dynamic Recipe Search
2. User-Friendly Interface
3. Recipe Organization and Management
4. Community Sharing
5. Responsive Design
6. Customization and User Profiles
7. Customization and User Profiles

Architecture

1. Frontend

The frontend is the user-facing part of the CookBook application, designed using **React.js**, which is a component-based JavaScript library optimized for building dynamic and interactive user interfaces.

Responsibilities:

- Render recipe lists, search interface, user profiles, and community sharing pages.
- Handle user interactions: searching recipes, adding or editing recipes, commenting, rating, etc.
- Manage client-side routing (using React Router).
- Communicate with the backend via API calls (REST or GraphQL).
- Provide a responsive design for seamless use across devices (desktop, tablet, mobile).
- State Management: Uses React Hooks (`useState`, `useEffect`, etc.) for managing local and global application state.

Key Components:

- **RecipeList** – Displays a list of recipes with filters and search functionality.
- **RecipeDetail** – Shows detailed recipe information.
- **UserProfile** – Displays and allows editing of user-specific settings.
- **RecipeForm** – Allows users to create or edit recipes.
- **NavigationBar** – Provides navigation controls.
- **CommentsSection** – Enables community interaction via comments.

2. Backend

Although the initial version focuses on the frontend (React.js), a full implementation should include a backend API server to handle persistent data and business logic.

Suggested Stack:

- **Node.js + Express.js** (for API server)
- Handles incoming HTTP requests from the frontend.
- Provides endpoints for CRUD operations:
    - GET /recipes
    - POST /recipes
    - PUT /recipes/:id
    - DELETE /recipes/:id
    - POST /comments
    - GET /users/:id/profile

Responsibilities:

- Data validation and business logic (e.g., rating rules, user authentication).
- Communicate with the database to persist and retrieve data.
- Handle user authentication and session management (e.g., via JWT).
- Provide security layers (rate limiting, CORS, etc.)

3. Database

For efficient storage and retrieval of recipe-related data, a NoSQL database like **MongoDB** is a suitable choice because of its flexibility in handling document-based data such as recipes with varying fields.

Example Data Models:

Recipe Document:

```
{
    "_id": "unique_recipe_id",
    "title": "Classic Lasagna",
    "ingredients": ["Pasta sheets", "Tomato sauce", "Beef", "Cheese"],
    "steps": ["Boil pasta", "Prepare sauce", "Layer and bake"],
    "category": "Main Course",
    "difficulty": "Medium",
    "preparationTime": 60,
    "authorId": "user_id",
    "comments": [
        { "userId": "commenter_id", "text": "Great recipe!", "date": "2025-
09-16T10:00:00Z" }
    ],
    "rating": 4.5
}
```

User Document:

```
{
    "_id": "user_id",
```

```
    "name": "John Doe",
    "email": "john@example.com",
    "passwordHash": "encrypted_password",
    "preferences": { "diet": "Vegetarian", "units": "Metric" },
    "savedRecipes": ["recipe_id1", "recipe_id2"]
}
```

□ Overall Architecture Diagram (Logical Flow)

```
[ User (Browser) ]
       ↓
[ React.js Frontend ]
       ↓ (HTTP API Requests)
[ Backend API (Node.js + Express.js) ]
       ↓ (Database Queries)
[ MongoDB Database ]
```

✅Summary of Architecture Benefits

- **Scalability:** Decoupled frontend and backend allow independent scaling.
- **Responsiveness:** React ensures a fast and smooth user experience.
- **Maintainability:** Component-based frontend enables easy updates and enhancements.
- **Security:** Backend can handle authentication, input validation, and data integrity.
- **Flexibility:** MongoDB stores flexible recipe documents supporting evolving schema.

Setup Instructions and Prerequisites

⚡ 1. Prerequisites

Before setting up the CookBook application, ensure the following tools and dependencies are installed on your development machine:

□ a. Node.js and npm

- **Node.js** is a JavaScript runtime that enables you to run JavaScript code outside of the browser.
- **npm (Node Package Manager)** helps manage packages and dependencies.

□ Download Node.js (includes npm):
https://nodejs.org/en/download/

□ Installation instructions:
https://nodejs.org/en/download/package-manager/

To verify installation, run in terminal:

```
node -v
npm -v
```

✅ b. Git (Version Control)

Git enables version control, collaboration, and project management.

🔗 Download and installation:
https://git-scm.com/downloads

Verify installation:

```
git --version
```

✅ c. Code Editor (IDE)

Use a modern IDE for development. Recommended choices:

- **Visual Studio Code**: https://code.visualstudio.com/download
- **Sublime Text**: https://www.sublimetext.com/download
- **WebStorm**: https://www.jetbrains.com/webstorm/download

✅ d. MongoDB (Optional for Backend)

If implementing a backend, use MongoDB for database storage.

🔗 MongoDB Community Server:
https://www.mongodb.com/try/download/community

🔧 2. Setup Process

✅ a. Clone the Repository

Clone the project repository from your version control system (e.g., GitHub).

```
git clone https://github.com/your-repo/cookbook.git
```

Navigate into the project folder:

```
cd cookbook
```

---

b. Install Dependencies

Install all required Node.js dependencies.

```
npm install
```

This will read `package.json` and install necessary libraries such as React, Axios, etc.

---

c. Start the Development Server

To launch the CookBook app locally:

```
npm start
```

This will:

- Launch the development server.
- Open the app at:
  `http://localhost:3000`

You can now interact with CookBook in your browser.

---

d. (Optional) Backend Server Setup

If a backend API server is provided:

1. Navigate to backend directory:
2. `cd backend`
3. Install dependencies:
4. `npm install`
5. Start the backend server:
6. `npm start`
7. Ensure MongoDB is running:
8. `mongod`

---

e. Access the Application

- Open your browser and navigate to:
  `http://localhost:3000`

You should see the CookBook homepage.

✅3. Post Setup

- Start developing by modifying components in `src/` folder.
- Use Git to track your changes:
- `git add .`
- `git commit -m "Initial setup"`
  `git push origin main`

  installation steps

⚡ Step 1 – Install Prerequisites

Ensure the following are installed:

- **Node.js & npm**: https://nodejs.org/en/download/
- **Git**: https://git-scm.com/downloads
- **Code Editor** (e.g., Visual Studio Code): https://code.visualstudio.com/download
- (Optional) **MongoDB** if using a backend API server: https://www.mongodb.com/try/download/community

---

⚡ Step 2 – Clone the Repository

Open your terminal (Command Prompt, PowerShell, or Bash) and run:

```
git clone https://github.com/your-repo/cookbook.git
```

Replace `https://github.com/your-repo/cookbook.git` with your actual repository URL.

Navigate into the project directory:

```
cd cookbook
```

⚡ Step 3 – Install Frontend Dependencies

Inside the project directory, run:

```
npm install
```

This will install all required dependencies specified in `package.json`, such as:

- React
- Axios
- React Router
- Other libraries

⚡ Step 4 – Start the Frontend Development Server

Once dependencies are installed, run:

```
npm start
```

This command will:

- Build the project.
- Start the development server.
- Open the app in your browser at:
  `http://localhost:3000`

⚡ Step 5 – (Optional) Backend Setup

If a backend API is part of the project:

1. Navigate to the backend directory (if separate):
2. `cd backend`
3. Install backend dependencies:
4. `npm install`
5. Ensure MongoDB is running:
6. `mongod`
7. Start the backend server:
8. `npm start`

The backend should be available at something like:
`http://localhost:5000/api`

Ensure frontend connects to the correct backend API URL.

⚡ Step 6 – Verify Installation

- Open your browser and visit:
  `http://localhost:3000`
- You should see the CookBook homepage with a functional interface.

- Test adding a recipe, performing a search, or viewing recipe details to verify full functionality.

✅Post Installation Tips

- Use `npm run build` to create a production-ready build.
- Use Git for version control:
- `git add .`
- `git commit -m "Setup project"`
  `git push origin main`

📁 CookBook Folder Structure

```
cookbook/
│
├── public/
│   ├── index.html              # Main HTML template
│   ├── favicon.ico             # App icon
│   └── manifest.json           # Web app manifest
│
├── src/
│   ├── assets/                 # Images, icons, fonts, and other static
assets
│   │   └── logo.png
│   │
│   ├── components/             # Reusable React components
│   │   ├── RecipeList.js
│   │   ├── RecipeDetail.js
│   │   ├── RecipeForm.js
│   │   ├── NavigationBar.js
│   │   └── CommentsSection.js
│   │
│   ├── pages/                  # Page-level components (views)
│   │   ├── HomePage.js
│   │   ├── SearchPage.js
│   │   ├── UserProfilePage.js
│   │   └── NotFoundPage.js
│   │
│   ├── services/               # API services (e.g., Axios setup)
│   │   └── api.js
│   │
│   ├── context/                # React Context API for state management
│   │   └── RecipeContext.js
│   │
│   ├── App.js                  # Main App component
│   ├── index.js                # Entry point
│   └── styles/                 # Global CSS or SCSS files
│       └── app.css
│
├── .gitignore                  # Specifies files/folders to ignore by Git
├── package.json                # Project dependencies and scripts
```

```
├── package-lock.json             # Auto-generated lock file
├── README.md                     # Project description
└── yarn.lock or npm-shrinkwrap.json # (Optional) for dependency version lock
```

✅Key Folder Purpose

Folder/File	Purpose
`public/`	Holds static files served by the app (e.g., index.html, icons).
`src/components/`	Reusable UI components (small parts of the app).
`src/pages/`	Components that represent individual pages (views).
`src/services/`	API calls and services for backend interaction.
`src/context/`	Global state management using React Context.
`src/styles/`	Global or shared CSS stylesheets.
`App.js`	Main app structure and routing setup.
`index.js`	React app entry point.
`.gitignore`	Files to exclude from version control (e.g., node_modules).
`package.json`	Lists dependencies and scripts for npm.

🚀 Running the CookBook Application

✅Step 1: Start the Frontend Development Server

1. Open your terminal or command prompt.
2. Navigate to the project directory:
3. `cd cookbook`
4. Start the React development server:
5. `npm start`

- This will compile the application and launch a local development server.
- By default, the app will be accessible at:
  🌐 http://localhost:3000

✅Step 2: (Optional) Start Backend Server

If your project includes a backend API server:

1. Open a separate terminal window.
2. Navigate to the backend folder:
3. `cd backend`
4. Install backend dependencies (if not done already):
5. `npm install`
6. Start the backend server:
7. `npm start`

- Make sure your MongoDB service is running:
- `mongod`
- By default, the backend API will be accessible at:
  - `http://localhost:5000` (or a different port, depending on your configuration).

✅Step 3: Open the Application in Browser

- Open your web browser.
- Navigate to:
  - http://localhost:3000
- You should see the **CookBook Home Page**.

✅Step 4: Development Workflow

While developing:

- Changes to the code (in `src/`) will automatically reflect in the browser via hot-reloading.
- You can edit components, pages, and services directly.
- Use your terminal to monitor logs and debug any issues.

✅Step 5: Building for Production

When you're ready to deploy the app:

```
npm run build
```

This will:

- Create an optimized production build in the `build/` folder.
- Serve static files for deployment to hosting services (e.g., Netlify, Vercel).

🍲 CookBook API Documentation

✅Base URL

`http://localhost:5000/api`

🔐 1. Authentication Endpoints

➤Register User

- **URL:** `/auth/register`
- **Method:** `POST`
- **Request Body:**
- ```
  {
    "name": "John Doe",
    "email": "john@example.com",
    "password": "password123"
  }
  ```
- **Response:**
- ```
  {
    "userId": "abc123",
    "token": "jwt_token_here"
  }
  ```

➤Login User

- **URL:** `/auth/login`
- **Method:** `POST`
- **Request Body:**
- ```
  {
    "email": "john@example.com",
    "password": "password123"
  }
  ```
- **Response:**
- ```
  {
    "userId": "abc123",
    "token": "jwt_token_here"
  }
  ```

🍽 2. Recipe Endpoints

➤Get All Recipes

- **URL:** `/recipes`
- **Method:** `GET`
- **Response:**

```
[
  {
    "id": "recipe1",
    "title": "Spaghetti Carbonara",
    "ingredients": ["Spaghetti", "Eggs", "Parmesan", "Bacon"],
    "instructions": "Cook pasta, mix with eggs and cheese, fry bacon."
  },
  {
    "id": "recipe2",
    "title": "Chicken Curry",
    "ingredients": ["Chicken", "Onion", "Curry Powder"],
    "instructions": "Cook chicken, add spices and onions."
  }
]
```

➢Get Recipe by ID

- **URL:** `/recipes/:id`
- **Method:** `GET`
- **Response:**

```
{
  "id": "recipe1",
  "title": "Spaghetti Carbonara",
  "ingredients": ["Spaghetti", "Eggs", "Parmesan", "Bacon"],
  "instructions": "Cook pasta, mix with eggs and cheese, fry bacon."
}
```

➢Create a Recipe

- **URL:** `/recipes`
- **Method:** `POST`
- **Headers:**
  `Authorization: Bearer <token>`
- **Request Body:**

```
{
  "title": "Pancakes",
  "ingredients": ["Flour", "Eggs", "Milk", "Sugar"],
  "instructions": "Mix ingredients, fry on pan."
}
```

- **Response:**

```
{
```

- "message": "Recipe created successfully",
- "recipeId": "new_recipe_id"
- }

➢Update Recipe

- **URL:** `/recipes/:id`
- **Method:** `PUT`
- **Headers:**
  `Authorization: Bearer <token>`
- **Request Body:**
- {
- "title": "Updated Pancakes",
- "ingredients": ["Flour", "Eggs", "Milk", "Honey"],
- "instructions": "Mix ingredients, fry on pan."
- }
- **Response:**
- {
- "message": "Recipe updated successfully"
- }

➢Delete Recipe

- **URL:** `/recipes/:id`
- **Method:** `DELETE`
- **Headers:**
  `Authorization: Bearer <token>`
- **Response:**
- {
- "message": "Recipe deleted successfully"
- }

---

3. Comments Endpoints

➢Add Comment to a Recipe

- **URL:** `/recipes/:id/comments`
- **Method:** `POST`
- **Headers:**
  `Authorization: Bearer <token>`
- **Request Body:**
- {

- `  "comment": "This recipe was amazing!"`
- `}`
- **Response:**
- `{`
- `  "message": "Comment added successfully"`
- `}`

➤Get Comments for a Recipe

- **URL:** `/recipes/:id/comments`
- **Method:** `GET`
- **Response:**
- `[`
- `  {`
- `    "commentId": "cmt1",`
- `    "user": "Jane Doe",`
- `    "comment": "Loved it!",`
- `    "timestamp": "2025-09-16T10:20:30Z"`
- `  },`
- `  {`
- `    "commentId": "cmt2",`
- `    "user": "John Smith",`
- `    "comment": "Easy and delicious recipe.",`
- `    "timestamp": "2025-09-16T11:45:00Z"`
- `  }`
- `]`

□ Authentication

□ 1. User Registration

- **Endpoint:**
  `POST /api/auth/register`
- **Request Body Example:**
- `{`
- `  "name": "John Doe",`
- `  "email": "john@example.com",`
- `  "password": "password123"`
- `}`
- **Response Example:**
- `{`
- `  "userId": "abc123",`
- `  "token": "jwt_token_here"`
- `}`

- **Function:**
  Registers a new user and returns a JSON Web Token (JWT) for further authentication.

□ 2. User Login

- **Endpoint:**
  `POST /api/auth/login`
- **Request Body Example:**
- ```
  {
    "email": "john@example.com",
    "password": "password123"
  }
  ```
- **Response Example:**
- ```
  {
    "userId": "abc123",
    "token": "jwt_token_here"
  }
  ```
- **Function:**
  Authenticates a user and returns a JWT to be used in future authenticated requests.

□ 3. Protected Endpoints

For endpoints like creating, updating, or deleting recipes, users must include the **Authorization header** in their HTTP request.

- **Example Header:**
- `Authorization: Bearer <your-jwt-token>`
- If the token is valid, the user is allowed to perform the action.
- Otherwise, the server responds with:
- ```
  {
    "error": "Unauthorized"
  }
  ```

□ 4. Token Expiration and Security

- The JWT token has an expiration time (e.g., 24 hours).
- Once expired, the user must log in again to get a new token.
- This ensures that user sessions are secure.

⚡ 5. Example Authentication Workflow
  ```

1. **Register User**
    - Send POST request to `/auth/register`.
    - Receive a `token`.
2. **Login User**
    - Send POST request to `/auth/login`.
    - Receive a `token`.
3. **Access Protected Endpoints**
    - Add header:
    - `Authorization: Bearer <token>`
    - Example:
    - `curl -X POST http://localhost:5000/api/recipes \`
    - `-H "Authorization: Bearer <token>" \`
    - `-H "Content-Type: application/json" \`
      `-d`
      `'{"title":"Pancakes","ingredients":["Flour","Milk","Eggs"],"instr`
      `uctions":"Mix and cool`

User interface

1. Landing Page

✅Purpose

The landing page is the first impression of CookBook. It provides an overview of the application's value, attracting users to explore recipes and features.

✅Key Sections

- **Header/Navbar:**
    - Logo
    - Links: Home, About, Features, Login / Register
- **Hero Section:**
    - Large, engaging headline (e.g., "Discover Your Next Favorite Recipe")
    - Call-to-action button (e.g., "Explore Recipes")
- **Features Overview:**
    - Icons + short descriptions of key features:
        - Search Recipes
        - Add Your Own Recipes
        - Personalized Recommendations
- **Testimonials Section:**
    - User quotes praising the app experience.
- **Footer:**
    - Contact info, social media links, about link, privacy policy.

---

2. User Dashboard (Freelancer / Home Chef)

✅Purpose

A personalized space for users to manage their recipes, view favorites, and update profile settings.

✅Key Components

- **Sidebar Menu:**
  - Dashboard Overview
  - My Recipes
  - Favorites
  - Add Recipe
  - Profile Settings
  - Logout
- **Dashboard Overview:**
  - Summary cards:
    - Number of recipes added
    - Favorite recipes count
    - Recent comments
- **Recipe List:**
  - Displays user's own recipes in card/grid format.
  - Edit / Delete button for each recipe.
- **Quick Action Buttons:**
  - Add New Recipe
  - Edit Profile

---

3. Admin Panel

✅Purpose

Admin panel allows administrators to manage users, recipes, and monitor the system.

✅Key Components

- **Sidebar Menu:**
  - Dashboard Home
  - Manage Users
  - Manage Recipes
  - Reports / Logs
  - System Settings
- **Dashboard Home:**

    - Analytics charts (e.g., total users, active recipes, system usage)
- **Manage Users Table:**
  | User ID | Name | Email | Role | Status | Actions (Edit / Delete) |
- **Manage Recipes Table:**
  | Recipe ID | Title | Creator | Status | Actions (Approve / Delete) |
- **Reports Section:**
    - Logs of errors, user reports, or flagged recipes.

---

4. Project Details Page (Recipe Details Page)

✅Purpose

Displays detailed information about a specific recipe.

✅Key Sections

- **Recipe Header:**
    - Recipe title
    - Recipe image
    - Creator name + avatar
- **Ingredients List:**
    - Clear bulleted list of ingredients.
- **Instructions Section:**
    - Step-by-step numbered instructions for preparing the recipe.
- **Comments Section:**
    - List of user comments.
    - Input field to add a new comment.
- **Actions:**
    - Edit / Delete (for recipe owner)
    - Rate the recipe (optional)
    - Favorite button

✅ Testing

Milestone 1 – Setup and Basic Functionality

- ✅Verified project setup and successful installation.
- ✅Tested initial page load (Landing Page, Home Page).
- ✅Checked basic navigation (Navbar links, routing).

Milestone 2 – User Authentication

- ✅Registered new users with valid/invalid data.

- ✅Tested login with correct and incorrect credentials.
- ✅Verified JWT token generation and expiration.
- ✅Checked that protected endpoints return “Unauthorized” when no token is present.

 Milestone 3 – Recipe Management

- ✅Added new recipes using the Add Recipe form.
- ✅Verified form validation (empty fields, invalid inputs).
- ✅Edited and deleted recipes; ensured correct behavior.
- ✅Confirmed only recipe owners can edit/delete their own recipes.

 Milestone 4 – Admin Panel

- ✅Admin could view the user and recipe management tables.
- ✅Validated actions: Approve / Delete user accounts and recipes.
- ✅Checked system analytics display correctly.

 Milestone 5 – User Interface Testing

- ✅Ensured responsive design on multiple devices (desktop, tablet, mobile).
- ✅Tested that interactive elements (buttons, forms) work as expected.
- ✅Verified correct display of comments and ratings.

⚡ **2. Tools Used for Testing**

 Postman

- Used to test backend APIs (Authentication, Recipe CRUD).
- Examples:
    - **Test Register API:**
        - Send POST request to `/api/auth/register`.
        - Confirm response returns user ID and JWT token.
    - **Test Create Recipe API:**
        - POST request to `/api/recipes`.
        - Verified Authorization header is required.
        - Checked response for successful recipe creation.

 Chrome Developer Tools

- Used to debug front-end behavior and inspect network requests.
    - **Elements Tab:**
        - Inspect DOM structure, verify that components render correctly.
    - **Console Tab:**

        - Checked for JavaScript errors or warnings during user interaction.
    - **Network Tab:**
        - Monitored API requests/responses.
        - Verified request headers (especially Authorization).
    - **Application Tab:**
        - Inspected localStorage to confirm JWT token is stored correctly.

✓ **Example Manual Test Cases:**

Test Case	Description	Expected Result	Actual Result
User Registration	Register user with valid input	201 Created with token	Passed
Login with wrong Password	Login with incorrect password	401 Unauthorized	Passed
Add Recipe	Add a recipe with valid data	201 Created	Passed
Access Protected Endpoint	Access recipes without token	401 Unauthorized	Passed
Responsive Design	Test UI on mobile	Layout adjusts correctly	Passed