

Project Documentation

Introduction

Project Title: CookBook

Team Leader

- **Mohanapriya G - 22g307mohana@gmail.com**

Team Member

- **Nikesh K - vinayagarnikesh@gmail.com**
- **Gajalakshmi G - gaji9904@gmail.com**
- **Swathi A - 22g310swathi@gmail.com**

Project Overview

Purpose

CookBook is a revolutionary web application designed to change the way you discover, organize, and create recipes. It caters to both novice and professional chefs, offering a user-friendly interface, robust features, and a vast collection of inspiring recipes.

Features

- Recipes from the MealsDB API
- Visual recipe browsing
- Intuitive and user-friendly design
- Search feature

Architecture

Component Structure

CookBook_NaanMudhalvan/

CookBook-App/

— node_modules/	# Contains all your project dependencies
— public/	# Public assets
— index.html	# Main HTML file

- | | └─ favicon.ico # Favicon for the app
- | | └─ assets/ # Other assets like images, logos, etc.
- | └─ src/ # Source code for the React app
- | | └─ assets/ # Static assets like images, icons, etc.
- | | └─ components/ # Reusable UI components
- | | | └─ Header.js # Header Component
- | | | └─ RecipeCard.js # Recipe Card Component
- | | | └─ SearchBar.js # Search Bar Component
- | | | └─ RecipeList.js # Recipe List Component
- | | | └─ ... # Other reusable components
- | | └─ context/ # React Context API files for state management
- | | | └─ RecipeContext.js # For managing recipe data
- | | | └─ UserContext.js # For managing user-related data (login, preferences)
- | | └─ pages/ # Pages in the app
- | | | └─ HomePage.js # Home page (e.g., display recipes, search)
- | | | └─ RecipeDetailPage.js # Recipe detail page (e.g., view single recipe)
- | | | └─ LoginPage.js # Login page
- | | | └─ DashboardPage.js # User's recipe dashboard
- | | | └─ ... # Other pages
- | | └─ services/ # Files to interact with APIs or backend services
- | | | └─ recipeService.js # Handle API calls for recipe data
- | | | └─ userService.js # Handle user-related API calls (login, profile, etc.)
- | | └─ utils/ # Utility functions
- | | | └─ format.js # Helper functions (e.g., for date, format, etc.)
- | | | └─ validation.js # Form validation helpers

— App.js	# Main app component that combines all the pages
— index.js	# Entry point for React, renders the App component
— styles/	# Global styles (CSS or SCSS)
— index.css	# Main CSS file
— theme.css	# Theme-related CSS
— .gitignore	# Git ignore file
— package.json	# Project configuration and dependencies
— README.md	# Project README file
— yarn.lock / package-lock.json	# Lock file for dependencies

State Management

In the **CookBook** application, the `useState` hook is utilized to manage the application's state. The main state variable here is `categories`, which will hold the list of meal categories fetched from the API. The initial value of `categories` is set to an empty array (`[]`), which will later be populated by the fetched data.

State management can be enhanced with **React Context API** or **Redux** if needed for more complex state handling, such as saving recipes, categories, or user preferences across different components.

Routing

React Router is used in this application to handle navigation between different pages. With the help of `react-router-dom`, we can define routes for different pages such as:

- **Home Page** (Recipe List)
- **Recipe Detail Page**
- **Category Page**

React Router ensures a smooth, single-page app experience where users can navigate between different sections of the app without a full page reload.

Setup Instructions

Prerequisites

- Node.js (version 14.x or higher)
- npm package manager

Installation

1. Clone the repository:
2. `git clone https://github.com/YourUsername/CookBook-Your-Virtual-Kitchen-Assistant.git`
3. Navigate to the project directory:
4. `cd CookBook-Your-Virtual-Kitchen-Assistant`
5. Install dependencies:
6. `npm install`

Folder Structure

src/

```
|-- components/           # Reusable components for the UI
|  |-- RecipeList.js      # Component for displaying all recipes
|  |-- RecipeCard.js      # Component for displaying individual recipe cards
|  |-- IngredientList.js  # Component for displaying the list of ingredients
|  |-- RecipeDetail.js    # Component for displaying recipe details
|-- context/              # Context API for global state management
|  |-- CookbookContext.js # Context for managing recipes and categories
|-- pages/                # Page-level components corresponding to different
routes
|  |-- HomePage.js        # Home page displaying recipe categories and list
|  |-- RecipeDetailPage.js # Page displaying detailed information of a recipe
|-- App.js                # Main component that contains routing logic
|-- index.js              # Entry point for React application
```

Client

- The client-side architecture includes the following:
 - **src/**: Contains the core React application code.
 - **components/**: Reusable UI components (e.g., `RecipeCard`, `RecipeList`, etc.)
 - **context/**: Handles global state management using **React Context API**.
 - **pages/**: Page components corresponding to different routes (e.g., `Home`, `Recipe Details`).

- **services/**: API service functions for fetching data (e.g., fetching recipes from an API).
- **app/store.js**: Global state setup (if Redux is used).

Running the Application

To start the application locally:

```
npm start
```

Demo code: <https://github.com/4700SM143/Cookbook.git>

This will launch the application at `http://localhost:3000`.

Component Documentation

Key Components

- **RecipeList:**
 - Displays a list of recipes fetched from an API or stored in local state.
 - Can be filtered by categories.
- **RecipeCard:**
 - Represents an individual recipe card displayed in the list.
 - Shows basic information about the recipe (e.g., name, short description).
 - Links to the recipe details page.
- **RecipeDetail:**
 - Shows detailed information about a specific recipe, including ingredients and cooking instructions.
 - Uses route parameters to fetch a specific recipe's data.
- **Navbar:**
 - A navigation bar to link between different pages of the app (e.g., home, recipe details, etc.).
- **CategoryList:**
 - Displays available categories of recipes, allowing users to filter recipes based on the category.

Reusable Components

- **RecipeCard**: Displays brief information about a recipe and is used across various parts of the app (in the list view or search results).
- **SearchBar**: Enables users to search for recipes by name, ingredient, or category.

State Management

Global State

CookbookContext: This context manages global states, including:

- **categories:** List of categories to categorize recipes.
- **recipes:** List of recipes fetched from an external API or hardcoded.
- **selectedRecipe:** The currently selected recipe (used in the recipe detail page).

Local State

Used for UI-specific states such as form inputs

User Interface

Screenshots or GIFs showcasing different UI features, such as pages, forms, or interactions.

Styling

CSS Frameworks/Libraries

- The application uses **Ant Design** for consistent and responsive UI components.

Theming

- Custom theming is applied using Ant Design's theming capabilities to align with the application's branding.

Testing

Testing Strategy

- The project uses **Jest** and **React Testing Library** for unit and integration testing of components and logic (like API calls, state management, etc.).

Code Coverage

- **Jest** provides built-in code coverage tools, ensuring that tests cover various parts of the app, such as components (e.g., `RecipeCard`, `RecipeDetail`) and utility functions (e.g., fetching recipes).

Screenshots or Demo

Live Demo

https://drive.google.com/file/d/1t5xweTQCYMqPUJNotchqrrmxzOQk_6Ee/view?usp=drivesdk

Screenshots

https://drive.google.com/drive/folders/1whO5f5gNlc9N7Gtk_CVLXpC9OmlSG5T9?usp=sharing

Known Issues

- **Known Issues for CookBook: Your Virtual Kitchen Assistant**

1. **Recipe and Category Fetching Issues:**

- **Symptoms:** The categories or recipes may not load correctly when the page is first visited or when switching between categories.
 - **Possible Cause:** API rate limits or connectivity issues might cause the data fetching to fail.
 - **Solution:** Ensure that the app handles retries or fallback mechanisms, and notify the user when the data is unavailable. Implement caching if possible to reduce load times.
-

2. **API Rate Limits:**

- **Symptoms:** Data fetching (for recipes, categories, etc.) may occasionally fail or experience delays due to API rate limits imposed by third-party services.
 - **Possible Cause:** APIs like Spoonacular or Edamam may impose limits on the number of requests in a given time frame.
 - **Solution:** Implement proper error handling (e.g., retry mechanisms, fallbacks) and possibly cache responses to minimize API requests. You can also notify users if the app has hit the rate limit and suggest waiting before trying again.
-

3. **Incomplete Recipe Data:**

- **Symptoms:** Certain recipes might not display complete details, such as missing ingredients, cooking instructions, or nutritional information.
 - **Possible Cause:** The data from the API might be incomplete or formatted differently across various recipes.
 - **Solution:** Implement data validation and fallback messages to inform the user when specific data is missing. Additionally, ensure that the app provides a default template for missing information to improve the user experience.
-

4. **Pagination Bug:**

- **Symptoms:** When filtering recipes (by categories, ingredients, etc.), pagination may not update properly, showing the wrong set of recipes or repeating data.
- **Possible Cause:** State management may not be handling the updates to the recipe list and pagination correctly, especially when switching between filters.

- **Solution:** Ensure that the pagination logic is correctly tied to the filtered data and that the state is updated properly. Debug and log the pagination state to find the root cause of the problem.
-

5. Mobile Responsiveness Issues:

- **Symptoms:** Some UI elements, such as the recipe cards or navigation bar, may not adjust properly for smaller screens or mobile devices.
 - **Possible Cause:** The layout or CSS might not be optimized for all screen sizes, especially on devices with smaller screens.
 - **Solution:** Ensure that the app uses **responsive design** principles, such as **media queries** and **flexbox** or **CSS Grid**. Test across different devices and screen sizes using browser dev tools or physical devices to catch any layout issues.
-

6. Slow Loading Performance:

- **Symptoms:** The app might take longer to load, particularly when dealing with large recipe datasets or when the user switches between different categories or pages.
 - **Possible Cause:** Large datasets or inefficient data fetching methods can cause the app to take longer to render.
 - **Solution:** Implement pagination or infinite scrolling to limit the amount of data rendered at once. Use **lazy loading** or **code splitting** to optimize initial load times. Consider caching API responses or using **Web Workers** to offload data processing in the background.
-

7. Search Functionality Bug:

- **Symptoms:** The search bar may not return the expected results or may have issues handling multiple queries in quick succession.
 - **Possible Cause:** Search logic might not be properly debounced, causing unnecessary API calls or search results to be delayed.
 - **Solution:** Implement **debouncing** or **throttling** for search inputs to minimize excessive API calls. Ensure that search queries are properly sanitized and handled to avoid issues like missing or partial matches.
-

8. Recipe Detail Page Loading:

- **Symptoms:** The recipe detail page may take longer to load, or users might encounter a loading spinner for an extended period without the page content being displayed.
 - **Possible Cause:** The API request for a specific recipe might take time to resolve, especially with large or complex recipes.
 - **Solution:** Optimize the detail page's loading state by showing a skeleton loader or placeholders while the content loads. Alternatively, use **React Suspense** for lazy loading components or data fetching.
-

9. UI/UX Inconsistencies Between Pages:

- **Symptoms:** There might be inconsistencies in UI elements across different pages, such as font sizes, button styles, or layout alignments.
 - **Possible Cause:** Lack of a consistent global stylesheet or improper use of CSS components across different pages.
 - **Solution:** Standardize your styling using a **CSS framework** (like **Ant Design** or **Bootstrap**) and ensure a consistent theme is applied across the app. If using custom styles, ensure that global styles are well-defined and scoped to avoid overrides.
-

10. User Preferences Not Saved:

- **Symptoms:** Users' saved recipes, preferences, or previously viewed recipes are not retained between sessions or page reloads.
 - **Possible Cause:** No mechanism for storing user preferences (e.g., in **localStorage**, **sessionStorage**, or a backend database).
 - **Solution:** Implement **localStorage** or **sessionStorage** to retain user data between sessions. Alternatively, you could integrate with a backend (e.g., Firebase or a custom API) to persist user data across sessions.
-

11. Ingredient List Display Issues:

- **Symptoms:** The list of ingredients may not display properly (e.g., ingredients appear out of order or some ingredients are missing).
- **Possible Cause:** Data formatting issues or the API response might be inconsistent.
- **Solution:** Ensure the ingredient list is properly parsed and formatted before rendering. Handle missing or malformed ingredient data gracefully by displaying a default message or skipping incomplete entries.

Future Enhancements

- **Persistent Storage:** Store recipes, user preferences, and categories in a backend database (e.g., Firebase, MongoDB, etc.).
- **Search Functionality:** Implement a search bar that allows users to search recipes by name or ingredients.
- **User Authentication:** Implement user login to save favorite recipes and track added recipes