**INTRODUCTION**

**Project Title: Cookbook**

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.

**Team Member:**

1.Yuvashree.U - **TEAM LEADER**(Implementation) (u.yuvashree1824@gmail.com)

2.Supriya.A – **TEAM MEMBER**(Implementation)( asupriya2002154@gmail.com)

3.Yugabharathi.P – **TEAM MEMBER(**Documentation)(yugabharathibharathi409@gmail.com)

4.Amrutha.S – **TEAM MEMBER**(Documentation) )( amruthakottaiah@gmail.com)

5.Bhavana.L – **TEAM MEMBER**(Documentation)( bhavanapatel527@gmail.com)

**Project overview**

**Purpose:**

CookBook is a cutting-edge web application designed to revolutionize the recipe discovery, organization, and creation process. Our purpose is to provide a user-friendly platform that inspires culinary exploration, fosters collaboration, and simplifies recipe management for cooking enthusiasts and professional chefs alike.

**Features:**

**✓ Recipes from the MealsDB API:** Access a vast library of international recipes spanning diverse cuisines and dietary needs.

**✓ Visual recipe browsing:** Explore recipe categories and discover new dishes through curated image galleries.

**✓ Intuitive and user-friendly design:** Navigate the app effortlessly with a clean, modern interface and clear navigation.

**✓Search feature:** various dishes can be accessed easily through the search feature. **Architecture**

The user experience starts with the CookBooks web application's UI, likely built with a framework like React or Vue.js for a smooth, single-page experience. This UI interacts with an API client specifically designed for CookBooks. This client handles communication with the backend, but with a twist: it leverages Rapid API, a platform providing access to various external APIs. This suggests CookBooks might integrate external data feeds or functionalities through Rapid API, enriching the user experience without building everything from scratch.

**Component Structure:**

1. App: The top-level component that wraps the entire application.

2. Header: The header component that contains the navigation menu.

3. Footer: The footer component that contains copyright information and social media links.

4. Main: The main component that contains the routing structure.

5. RecipeList: A component that displays a list of recipes.

6. RecipeDetail: A component that displays detailed information about a single recipe.

7. SearchBar: A component that allows users to search for recipes.

8. UserProfile: A component that displays user profile information

**State Management:**

CookBook will use the Context API for state management. This will allow us to share global state between components without passing props down manually.

We will create a Context component that wraps the entire application and provides the global state. Individual components can then subscribe to the context and access the global state.

**Routing:**

The **routing structure** of the CookBook recipe application is based on **React Router**. Below is an explanation of how it is structured:

**1. Installing React Router**

To use React Router in your project, install it using:

bash

CopyEdit

npm install react-router-dom

**2. Folder Structure**

The project organizes files into:

* **Pages Folder** → Contains different pages accessed via unique URLs.
* **Components Folder** → Contains reusable components.
* **Styles Folder** → Contains CSS files for styling.

**3. Routing Configuration**

The App.js (or App.tsx for TypeScript) file sets up routing using react-router-dom:

jsx

CopyEdit

import { BrowserRouter as Router, Routes, Route } from "react-router-dom";

import Home from "./pages/Home";

import Category from "./pages/Category";

import Recipe from "./pages/Recipe";

import Navbar from "./components/Navbar";

function App() {

return (

<Router>

<Navbar />

<Routes>

<Route path="/" element={<Home />} />

<Route path="/category/:categoryName" element={<Category />} />

<Route path="/recipe/:id" element={<Recipe />} />

</Routes>

</Router>

);

}

export default App;

**4. Explanation of Routes**

* / → Loads the Home page.
* /category/:categoryName → Loads recipes under a specific category.
* /recipe/:id → Displays a specific recipe’s details.

**5. Navigating with Link**

To navigate between pages without reloading:

jsx

CopyEdit

import { Link } from "react-router-dom";

<Link to={`/category/${categoryName}`}>View Category</Link>

<Link to={`/recipe/${recipeId}`}>View Recipe</Link>

**Setup Instructions**

**Prerequisites:**

1. Node.js: The latest version of Node.js (LTS or higher) installed on your system.

2. npm: The latest version of npm (the package manager for Node.js) installed on your system.

3. Git: The latest version of Git installed on your system.

**Installation:**

1. Clone the repository: Run the following command in your terminal to clone the CookBook repository:

git clone https://github.com/[username]/cookbook.git

2. Navigate to the project directory: Run the following command to navigate to the project directory:

cd cookbook

3. Install dependencies: Run the following command to install the project dependencies:

npm install

4. Configure environment variables: Create a new file named .env in the project root directory and add the following environment variables:

REACT\_APP\_API\_URL=https://api.cookbook.com

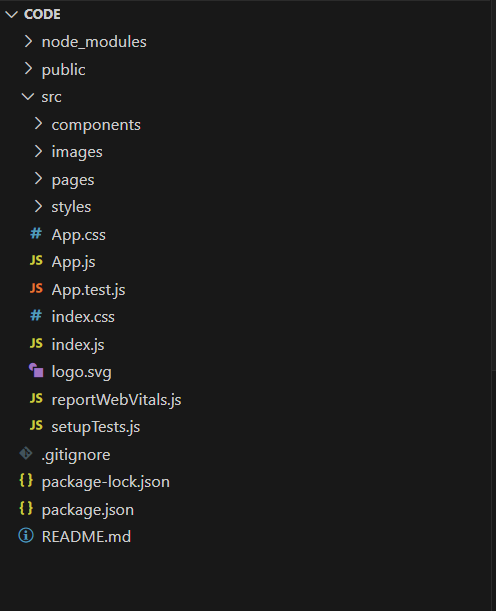
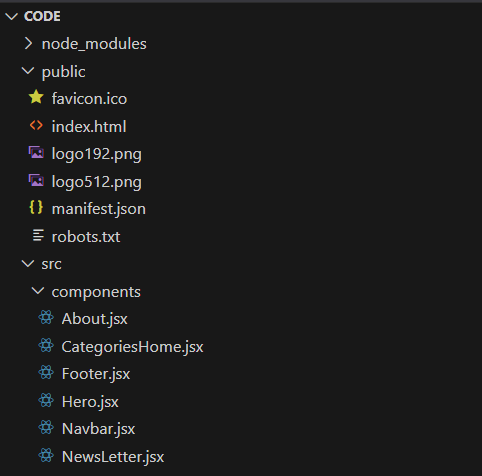
REACT\_APP\_API\_KEY=YOUR\_API\_KEY

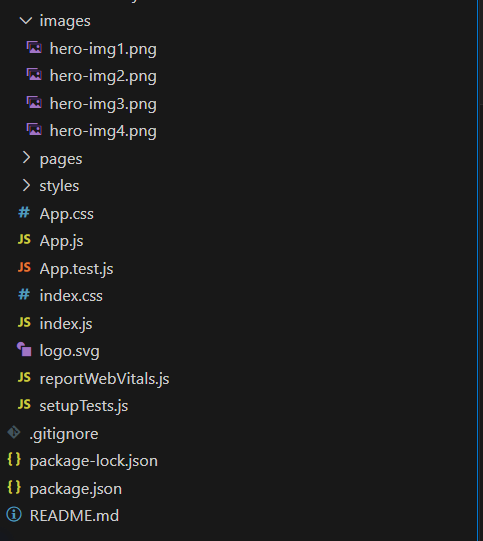
5. Start the application: Run the following command to start the application:

npm start

**Folder Structure**

**Clients:**

** **

****

In this project, we’ve split the files into 3 major folders, Components, Pages and Styles. In the pages folder, we store the files that acts as pages at different url’s in the application. The components folder stores all the files, that returns the small components in the application. All the styling css files will be stored in the styles folder.

**Utilities**

Explanation of Category.js Component

This React component is responsible for displaying a list of meals from a selected category. It fetches data from TheMealDB API and allows users to navigate between different categories.

**1️.Importing Dependencies**

**javascript**

import React, { useEffect } from 'react'

import '../styles/CategoryPage.css'

import { useNavigate, useParams } from 'react-router-dom';

import axios from 'axios';

* React and useEffect → Required for React component and handling side effects (API call).
* useNavigate → Used to navigate programmatically between routes.
* useParams → Retrieves the category ID from the URL.
* axios → Handles API requests.
* CSS (CategoryPage.css) → Styles the page.

**2️.Defining the Component**

**javascript**

const Category = () => {

const navigate = useNavigate(); // Hook for navigation

const { id } = useParams(); // Extract category ID from URL

const [items, setItems] = React.useState([]); // State to store fetched meals

* useNavigate() → Used to programmatically change routes.
* useParams() → Retrieves the category name (id) from the URL.
* useState([]) → Initializes state for storing fetched meals.

**3️.Fetching Category Data from API**

**useEffect Hook**

**javascript**

useEffect(() => {

fetchItems(id)

}, [window.location.href]) // Runs when the URL changes

* Runs fetchItems(id) whenever the page URL changes (window.location.href).
* Ensures the component re-fetches data when navigating between categories.

**API Call Function**

**javascript**

const fetchItems = async (idd) => {

await axios.get(`https://www.themealdb.com/api/json/v1/1/filter.php?c=${idd}`)

.then(response => {

setItems(response.data.meals) // Store meals in state

console.log(response.data.meals) // Debugging: logs API response

})

.catch(error => console.error(error)); // Handles errors

}

* Makes an API request to fetch meals based on category (idd).
* Stores the response (response.data.meals) in state (setItems).
* Handles errors with .catch(error => console.error(error)).

**4️.UI Layout**

**Header Section**

**javascript**

<div className="categorypage-head">

<h2>Category: <i>{id}</i></h2>

<div className='categorypage-head-options'>

<p>Other popular categories:</p>

<span>

<button onClick={()=>{ navigate(`/category/Chicken`); fetchItems('Chicken')} }>Chicken</button>

<button onClick={()=> {navigate(`/category/Vegetarian`); fetchItems('Vegetarian')}}>Vegetarian</button>

<button onClick={()=> {navigate(`/category/Starter`); fetchItems('Starter')}}>Starter</button>

<button onClick={()=> {navigate(`/category/Seafood`); fetchItems('Seafood')}}>Seafood</button>

<button onClick={()=> {navigate(`/category/Dessert`); fetchItems('Dessert')}}>Dessert</button>

</span>

</div>

</div>

* Displays category name dynamically ({id}).
* Shows popular category buttons, allowing navigation to different categories.
* Clicking a button:
  + Changes the route (navigate('/category/Chicken')).
  + Calls fetchItems('Chicken') to update displayed meals.

**Displaying Fetched Meals**

**javascript**

<div className="categorypage-body">

{items ?

<div className="food-items">

{items.map((item) => (

<div className="food-item" onClick={()=> navigate(`/recipie/${item.idMeal}`)}>

<img src={item.strMealThumb} alt="food-item" />

<h4>{item.strMeal}</h4>

</div>

))}

</div>

: "Loading..."}

</div>

* Checks if items exist before rendering.
* Maps over items array, rendering each meal as:
  + An image (item.strMealThumb).
  + A name (item.strMeal).
  + On Click: Navigates to the recipe details page (/recipie/${item.idMeal}).

**5️ . Exporting the Component**

**javascript**

export default Category;

* Allows this component to be imported elsewhere in the project.

**Running the Application**

**✓ Get the code:**

• Download the code from the drive link given below: <https://drive.google.com/drive/folders/1u8PnV_mE0mwKkH_CvuNpliZtRLJZMqrO?usp=sharing>

**Install Dependencies:**

• Navigate into the cloned repository directory and install libraries: cd recipe-app-react

npm install

**✓ Start the Development Server:**

• To start the development server, execute the following command:

npm start

**Access the App:**

• Open your web browser and navigate to <http://localhost:3000>.

• You should see the recipe app's homepage, indicating that the installation and setup were successful.

**Component Documentation**

**Key Components**

**1. Navbar Component**

* Purpose: Provides navigation throughout the application.
* Props:
  + links (array): List of navigation links.
  + logo (string): URL for the logo image**.**

**2. Hero Component**

* Purpose: Serves as the main banner, introducing the application with a call-to-action button.
* Props:
  + title (string): Main heading text.
  + description (string): Short description of the app.
  + ctaText (string): Call-to-action button text.

**3. Popular Categories Component**

* Purpose: Displays a list of popular recipe categories.
* Props:
  + categories (array): List of category objects containing id, name, and image.

**4. Trending Dishes Component**

* Purpose: Highlights trending dishes with images.
* Props:
  + dishes (array): List of dish objects containing id, name, image, and rating.

**5. Category Page Component**

* Purpose: Displays a list of dishes based on the selected category.
* Props:
  + category (string): Selected category name.
  + dishes (array): List of dishes in the category.

**6. Recipe Page Component**

* Purpose: Displays detailed information about a selected recipe.
* Props:
  + recipe (object): Contains recipe details such as name, ingredients, instructions, image, and videoUrl.

**7. Newsletter Subscription Component**

* Purpose: Allows users to subscribe to a newsletter for recipe updates.
* Props:
  + onSubscribe (function): Function to handle user subscription.

**Reusable Components**

**1. Button Component**

* Purpose: A customizable button used throughout the app.
* Props:
  + text (string): Button label.
  + onClick (function): Click handler.
  + variant (string): Style variant (primary, secondary, etc.).
  + disabled (boolean): Whether the button is disabled.

**2. Card Component**

* Purpose: Displays information in a card format.
* Props:
  + title (string): Card title.
  + image (string): URL for card image.
  + description (string): Short text description.
  + onClick (function): Click handler.

**3. Loader Component**

* Purpose: Displays a loading animation while fetching data.
* Props:
  + size (string): Size of the loader (small, medium, large).
  + color (string): Color of the loader.

**4. Search Bar Component**

* Purpose: Allows users to search for recipes.
* Props:
  + onSearch (function): Function to handle search input.
  + placeholder (string): Placeholder text for the input field.

**5. Modal Component**

* Purpose: Displays pop-up dialogs (e.g., detailed recipe instructions).
* Props:
  + isOpen (boolean): Controls visibility.
  + onClose (function): Function to close the modal.
  + content (JSX): The content to be displayed inside the modal.

**State Management**

State management is the practice of handling application state, ensuring data consistency and responsiveness.

1. **Global State**

Global state refers to data shared across multiple components. It is usually managed using state management libraries like Redux, Context API, or Vuex.

**What to Document?**

State Variables: The global states and their purposes.

State Flow: How state updates propagate in the application.

State Management Tools: Frameworks or libraries used (e.g., Redux, Context API).

**Example (React Context API)**

Import React, { createContext, useState, useContext } from ‘react’;

Const ThemeContext = createContext();

Export function ThemeProvider({ children }) {

Const [theme, setTheme] = useState(“light”);

Return (

<ThemeContext.Provider value={{ theme, setTheme }}>

{children}

</ThemeContext.Provider>

);

}

// Usage

Function App() {

Const { theme, setTheme } = useContext(ThemeContext);

Return (

<div className={theme}>

<button onClick={() => setTheme(theme === “light” ? “dark” : “light”)}>

Toggle Theme

</button>

</div>

);

}

1. **Local State**

Local state is confined to a single component and is used to manage UI behavior, form inputs, or other component-specific data.

**What to Document?**

State Variables: Local states used within the component.

State Management Methods: Functions used to update the local state.

Scope: Where and how the local state is used.

**Example (React useState Hook)**

Import { useState } from ‘react’;

Function Counter() {

Const [count, setCount] = useState(0);

Return (

<div>

<p>Count: {count}</p>

<button onClick={() => setCount(count + 1)}>Increment</button>

</div>

);

}

**USER INTERFACE**

The document contains details about the CookBook application, including its features, project setup, and API integration. However, for screenshots or GIFs showcasing different UI features, you might need to extract images from the document or from the demo link:

**🔗 Project Demo Link:**[**Demo Link**](file:///C:\Users\uyuva\AppData\Local\Microsoft\Windows\INetCache\IE\Videos\Screen%20Recordings\Screen%20Recording%202025-03-06%20190647.mp4)

**🔗 Code Repository & Assets:**[**Project Link**](file:///C:\Users\uyuva\AppData\Local\Microsoft\Windows\INetCache\IE\0QLNWHX1\code-20250305T051621Z-001\code)

**Styling**

**CSS Frameworks/Libraries**

@import url('https://fonts.googleapis.com/css2?family=Poppins:wght@200;300;400;500;600;700;800;900&family=Teko:wght@400;600;700&family=Whisper&display=swap');

body {

  margin: 0;

  font-family: 'Poppins', sans-serif;

  -webkit-font-smoothing: antialiased;

  -moz-osx-font-smoothing: grayscale;

}

code {

  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',

    monospace;

}

/\*

font-family: 'Poppins', sans-serif;

font-family: 'Teko', sans-serif;

font-family: 'Whisper', cursive; \*/

**State Management**

In a React application, state management plays a crucial role in maintaining and updating data across various components. It can be categorized into **Global State** and **Local State**.

**Global State Management**

Global state refers to data that needs to be accessed and updated across multiple components. Managing this efficiently ensures consistency and enhances the user experience.

**How Global State Flows Across the Application**

1. **Centralized Storage** – Global state is usually stored in a central location such as **React Context API, Redux, Recoil, or Zustand**.
2. **State Updates** – Components update the global state using **actions (Redux)** or **setters (Recoil/Zustand)**, ensuring consistency.
3. **State Consumption** – Any component needing access to global data can subscribe to the state and receive updates.

**Local State Management**

Local state is used within individual components to manage UI changes and interactions without affecting other parts of the application.

**Handling Local State in Components**

1. **useState Hook** – Used for managing simple component-level states like form inputs, modal visibility, etc.
2. **useReducer Hook** – Helpful for more complex state transitions within a component.

**Testing**

**Testing Strategy for CookBook Application**

Testing is a crucial aspect of ensuring the stability, reliability, and performance of the CookBook application. The testing strategy includes unit testing, integration testing, and end-to-end (E2E) testing using modern testing frameworks.

**1. Unit Testing**

Unit testing is performed on individual components to verify their functionality in isolation.

* Tools Used: Jest, React Testing Library
* Scope:
  + Test individual React components (e.g., Hero, Navbar, PopularCategories).
  + Validate component rendering and UI changes.
  + Check for correct API calls using mock functions (e.g., Axios mocks).
  + Ensure props and state updates work correctly.

**2. Integration Testing**

Integration tests verify that multiple components work together correctly.

* Tools Used: Jest, React Testing Library, Mock Service Worker (MSW) for API mocking.
* Scope:
  + Verify API data is fetched and displayed correctly (e.g., fetching recipe categories).
  + Ensure state management updates correctly when interacting with the UI.
  + Simulate user interactions like button clicks and form submissions.

**3. End-to-End (E2E) Testing**

E2E tests validate the entire application flow, ensuring all components work together as expected.

* Tools Used: Cypress, Playwright
* Scope:
  + Test full user workflows, such as searching for recipes, navigating between pages, and viewing recipe details.
  + Validate UI elements are displayed correctly across different screen sizes.
  + Ensure API calls and responses are handled properly.

**Code Coverage**

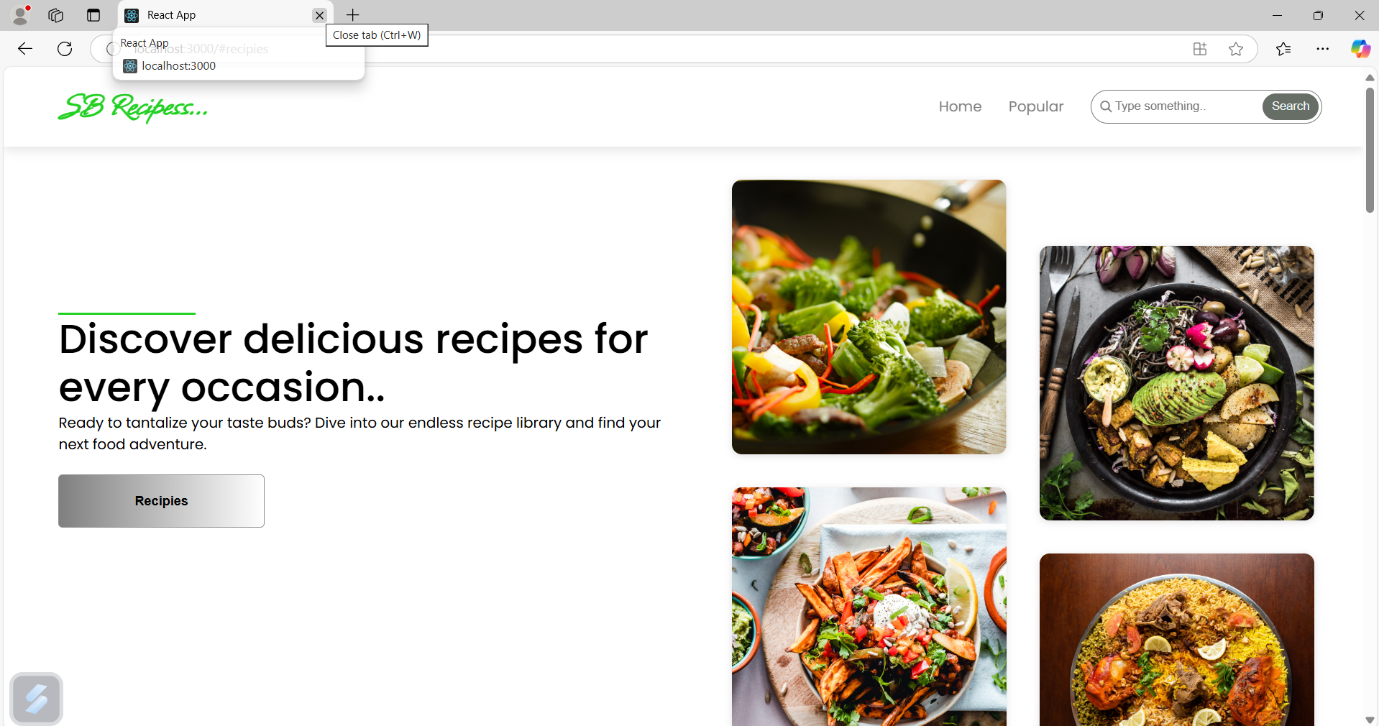
To ensure sufficient test coverage, the following tools and techniques are used:

* Jest + Istanbul (nyc): Measures line, function, and statement coverage.
* React Testing Library: Ensures important UI elements are tested.
* Cypress Dashboard: Tracks E2E test execution and failure rates.

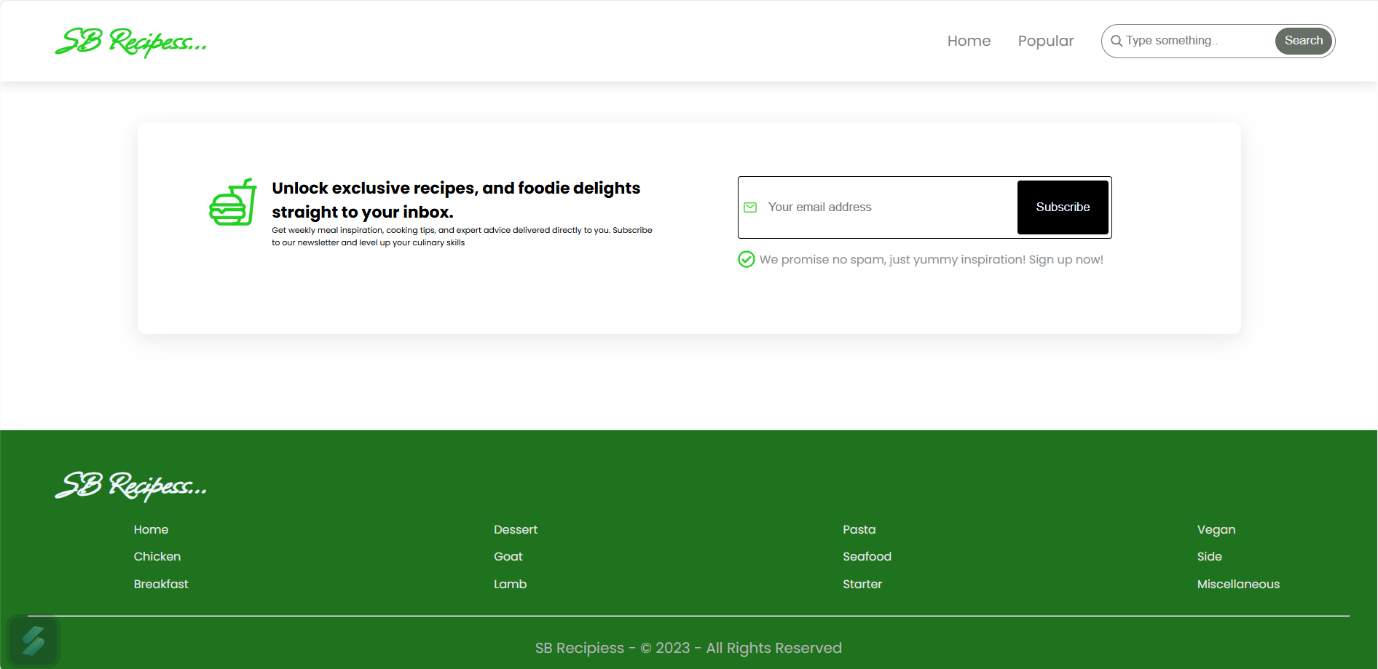
To generate a test coverage report, use:

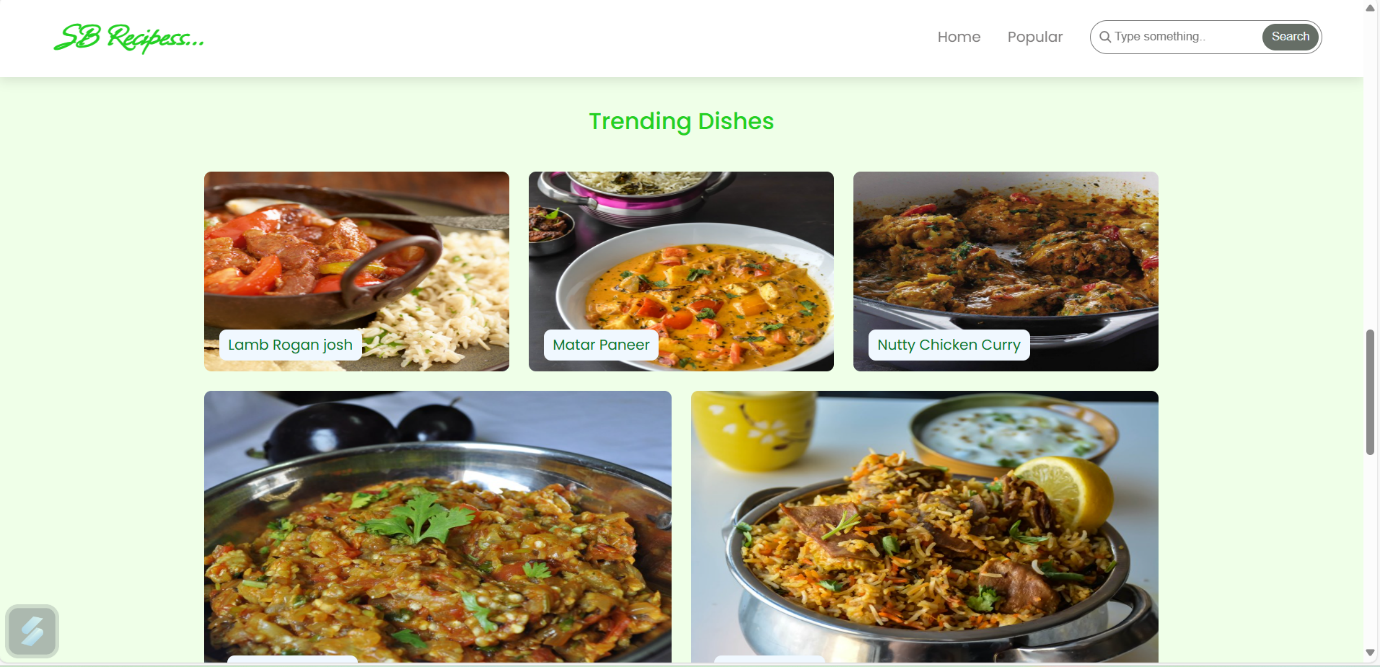
**npm test -- --coverage**

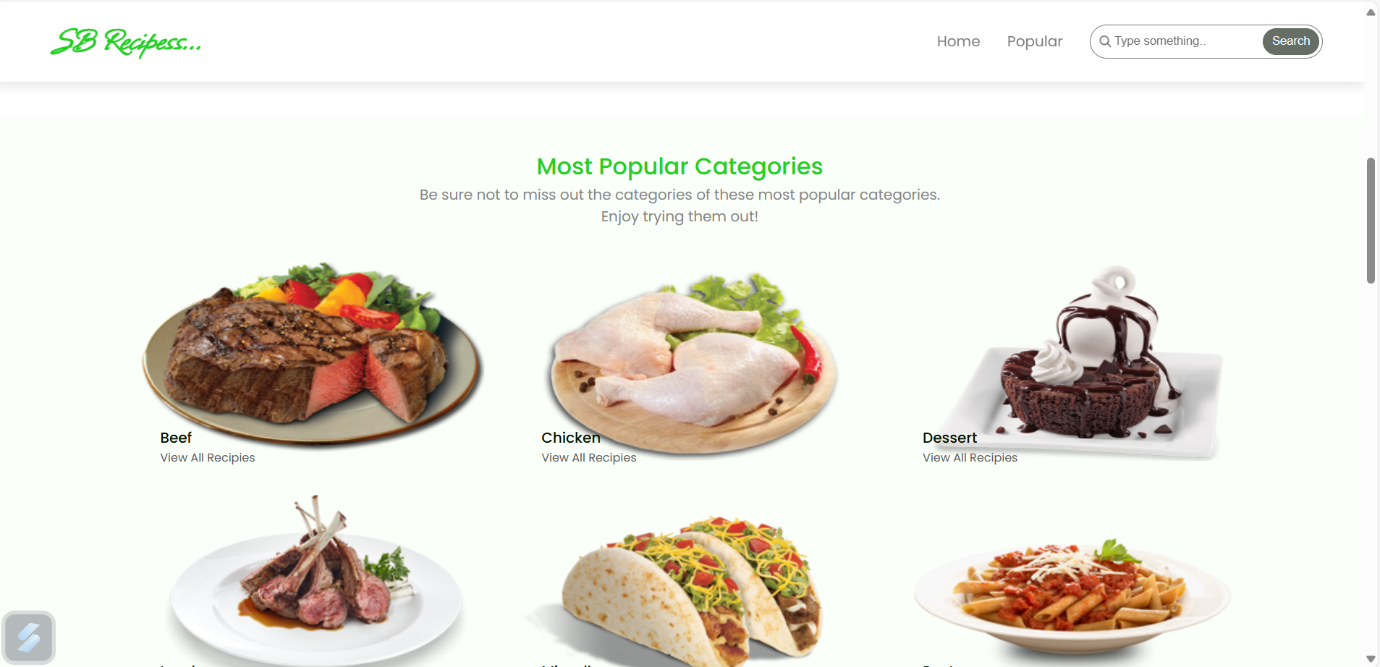
This generates a report detailing which lines of code were tested.



**Screenshot**

****

****

****

**Known Issues**

**1. API Limitations & Dependencies**

* The application relies on the MealsDB API for recipe data. If the API service is down or reaches its request limit, the app may fail to fetch recipes.
* Some categories or recipes might not always be available if they are removed from the API source.
* Potential latency issues when retrieving data from external sources, especially during peak hours.

**2. UI/UX Inconsistencies**

* Some components (such as category pages and recipe details) may load slower due to large image sizes or API response times.
* The search functionality may not always return relevant results due to limited filtering options.
* The mobile version may have minor responsiveness issues on small-screen devices.

**3. Error Handling & Edge Cases**

* If the API request fails, error messages are not always displayed to the user, leading to an unclear user experience.
* No caching mechanism is implemented, leading to repetitive API calls even when navigating back to previously loaded content.
* If an invalid recipe ID is provided, the app might crash instead of gracefully handling the error.

**4. Authentication & Security Concerns**

* Currently, there is no user authentication implemented, making it impossible for users to save favorite recipes or maintain personalized settings.
* No rate-limiting or request throttling is in place, which could make the application vulnerable to excessive API requests.

**5. Code & Performance Optimization**

* The useEffect() hook calls API requests without a caching layer, leading to unnecessary re-fetching of data.
* Some components may re-render unnecessarily, affecting performance, especially on lower-end devices.
* Large dependencies (such as Bootstrap and Tailwind CSS being used together) may increase bundle size, leading to longer load times

**FUFUTURE ENHANCEMENT**

**FUTURE SCOPE**

The **CookBook** application has significant potential for future enhancements and expansions. Here are some areas of future scope:

**1. AI and Personalization**

* **AI-based Recipe Recommendations:** Implement machine learning to suggest recipes based on user preferences, past searches, and dietary restrictions.
* **Voice Assistance Integration:** Integrate with Google Assistant or Alexa for hands-free cooking guidance.
* **Image Recognition for Ingredients:** Allow users to upload images of ingredients, and the app will suggest recipes.

**2. Community and Social Features**

* **User-Generated Recipes:** Enable users to add and share their own recipes with the community.
* **Social Media Integration:** Allow users to share recipes directly on platforms like Instagram, Facebook, and Pinterest.
* **Live Cooking Sessions:** Host live cooking events or tutorials within the app.

**3. Enhanced Features and Functionality**

* **Meal Planning & Grocery List:** Provide weekly meal planning options and automatically generate grocery lists.
* **Dietary and Allergy Filters:** Allow filtering recipes based on vegan, keto, gluten-free, or allergen-friendly options.
* **Multilingual Support:** Expand the language options to cater to a global audience.

**4. Monetization Strategies**

* **Subscription Model:** Offer premium content such as exclusive recipes, video tutorials, or personalized diet plans.
* **Affiliate Marketing:** Partner with grocery stores for ingredient purchases directly from the app.
* **Sponsored Content:** Collaborate with food brands and professional chefs for sponsored recipes.

**5. Cross-Platform Expansion**

* **Mobile App Development:** Expand the web app to mobile platforms (Android and iOS) for a more seamless experience.