**FRONT DEVELOPMENTWITH REACT.js**

**PROJECT DOCUMENTATION**

1. **INTRODUCTION:**

Project Title: CookBook[Your Virtual Kitchen Assistant]

Team Members: Jayashree.B, Preethi.V, Priya.S, Anitha.M.

1. **PROJECT OVERVIEW:**

Purpose: The primary goal of CookBook is to provide a user-friendly platform that caters to individuals passionate about cooking, baking, and exploring new culinary horizons. Our objectives include:

* User-Friendly Experience: Create an interface that is easy to navigate, ensuring users can effortlessly discover, save, and share their favourite recipes.
* Comprehensive Recipe Management: Offer robust features for organizing and managing recipes, including advanced search options.
* Technology Stack: Leverage modern web development technologies, including React.js, to ensure an efficient, and enjoyable user experience.

**Features:**

* Recipes from the MealsDB API: Access a vast library of international recipe spanning diverse cuisines and dietary needs.
* Visual recipe browsing: Explore recipe categories and discover new dishes through acurated 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

1. **ARCHITECTURE:**

* Component Structure: HTML, CSS, and JavaScript; Basic knowledge of HTMLfor creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

State Management: State Management with useState Hook;

* The code utilizes the useState hook to create a state variable named categories. This variable acts as a container to hold the fetched data, which in this case is a list of meal categories. Initially, the categories state variable is set to an empty array [].

Routing: Explain the routing structure if using react-router or another routing library.

1. **SETUP INSTRUCTION:**

* Prerequisites: Node.js and npm; Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the local environment.
* It provides a scalable and efficient platform for building network applications.
* Install Node.js and npm on your development machine, as they are required to run JavaScript on the server-side.

**Download:** <https://nodejs.org/en/download/>

* Installationinstructions: https://nodejs.org/en/download/packagemanager/Installation:
* Installation of required tools; To build CookBook, we'll need a developer's toolkit. We'll use React.js for the interactive interface, React Router Dom for seamless navigation, and Axios to fetch news data. For visual design, we'll choose either Bootstrap or Tailwind CSS for pre-built styles and icons. Open the project folder to install necessary tools, In this project, we use:
  + React Js
  + React Router Dom
  + React Icons
  + Bootstrap/tailwind css
  + Axios

For further reference, use the following resources

<https://react.dev/learn/installation>

https://react-bootstrap-v4.netlify.app/getting-started/introduction/

https://axios http.com/docs/intro

<https://reactrouter.com/en/main/start/tutorial>

5. **FOLDER STRUCTURE:**

**Client:**

****

* The components folder stores all the files, that returns the small components in the application.
  + In the pages folder, we store the files that acts as pages at different url’s in the application. All the styling css files will be stored in the styles folder.

Utilities: HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of your app , CSS for styling, and JavaScript for client-side interactivity is essential.

1. **RUNNING THE APPLICATION:**

Running the React App: With the React app created, you can now start the development server and see your React application in action.

Start the development server: npm start This command launches the development server, and you can access your React app at http://localhost:3000 in your web browser.

1. **COMPONENT DOCUMENTATION:**

Key Components: Each Component Usually specific to your application or design system, key components are the fundamental building pieces of your system. Although they are essential to the system's fundamental identity or functionality, they are not intended to be reused in other situations. Reusable Components:

Reusable Parts Reusable parts are made to be utilized in a variety of settings. Because of their greater flexibility and abstract character, they can be used for a variety of functions throughout the system.Because of their greater flexibility and abstract character, they can be used for a variety of functions throughout the system.

8. **STATE MANAGEMENT:**

**Global State:** State Management with these State Hook;

The code utilizes the useState hook to create a state variable named categories. This variable acts as a container to hold the fetched data, which in this case is a list of meal categories. Initially, the categories state variable is set to an empty array [].

**Local State:** Recipes, attributes, files, libraries, templates are the local components used

**9. USER INTERFACE:**

React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications. Install React.js, a JavaScript library for building user interfaces. Create a new React app: npx create-react-app my-react-app Replace my-react-app with your preferred project name. Navigate to the project directory: cd my-react-app

**10. STYLING:**

**CSS Frameworks/Libraries**: Basic knowledge of HTMLfor creating the structure of your app,

CSS for styling, and JavaScript for client side interactivity is essential.

**11. TESTING:**



**Testing Strategy:** Fetching all the available categories :Here, with the API request to Rapid API, we fetch all the available categories. This code snippet demonstrates how to fetch data from an API and manage it within a React component. It leverages two key functionalities: state management and side effects. State Management with useStateHook:The code utilizes the useState hook to create a¬ state variable named categories. This variable acts as a container to hold the fetched data, which in this case is a list of meal categories. Initially, the categories state variable is set to an empty array []. Fetching Data with useEffect Hook: The useEffect hook is employed to execute¬a side effect, in this instance, fetching data from an API. The hook takes a callback function (fetchCategories in this case) and an optional dependency array. The callback function is invoked after the component renders and whenever the dependencies in the array change. Here, the dependency array is left empty [], signifying that the data fetching should occur only once after the component mounts. Fetching Data with fetchCategories Function: An asynchronous function named¬ fetchCategories is defined to handle the API interaction. This function utilizes the axios.get method to make a GET request to a specified API endpoint (https://www.themealdb.com/api/json/vi/1/categories.php in this example). This particular endpoint presumably returns a JSON response containing a list of meal categories. Processing API Response: The .then method is chained to the axios.get call to handle a¬ successful response from the API. Inside the .then block, the code retrieves the categories data from the response and updates the React component's state using the setCategories function. This function, associated with the useState hook, allows for modification of the categories state variable. By calling setCategories(response.data.categories), the component's state is updated with the fetched list of meal categories. Code Coverage: It leverages the useState hook to establish a state variable named recipie (which is¬ initially empty). This variable acts as a container to hold the fetched recipe data. The useEffect hook comes into play to execute a side effect, in this instance, fetching¬ data from an API endpoint. The hook takes a callback function (fetchRecipie in this case) and an optional dependency array. The callback function is invoked after the component renders and whenever the dependencies in the array change. Here, the dependency array is left empty [], signifying that the data fetching should occur only once after the component mounts. The fetchRecipie function is an asynchronous function responsible for handling the API¬ interaction. This function likely utilizes the axios.get method to make a GET request to a predetermined API endpoint, the exact URL construction of which depends on a recipeId retrieved from somewhere else in the code (not shown in the snippet). The code snippet employs the .then method, which is chained to the axios.get call, to¬ handle a successful response from the API. Inside the .then block, the code retrieves the first recipe from the data.meals array in the response and updates the React component's for state using the setRecipie function. This function, associated with the useState hook, allows modification of the recipie state variable. By calling setRecipie(response.data.meals[0]), the component's state is updated with the fetched recipe data, effectively making it available for use throughout the component. An optional error handling mechanism is incorporated using the .catch block. This block¬ is designed to manage any errors that might arise during the API request. If an error occurs, the .catch block logs the error details to the console using the console.error method. This rudimentary error handling mechanism provides a way to identify and address potential issues during the data fetching process.

**12. SCREENSHOTS or DEMO: Project demo link:**

<https://drive.google.com/file/d/1khMJkccySgKyqRaEZgCpgDACHi572Llj/view?usp=sharing>

**13. KNOWN ISSUES:**

Known problems with a cookbook could be:

1. Compatibility Problems: Some parts might not function properly in certain browsers or with older software versions.

2. Performance bottlenecks: In large-scale applications, specific components may result in slowdowns.

**14. FUTURE ENHANCEMENT:**

A cookbook's potential future improvements could be:

1. **Better Compatibility:**

More software versions and browsers are supported.

1. **Performance Optimizations:**

Improvements to lower resource consumption and loading times.

1. **Error Handling:**

Stronger management of failure scenarios and edge cases.