**COOKBOOK: MY VIRTUAL ASSISTANT**

**INTRODUCTION**:

**Project Title:**

**COOKBOOK: MY VIRTUAL ASSISTANT**

Welcome to **Cookbook**, it is your personal virtual assistant designed to make cooking easier, smarter, and more enjoyable. Whether you’re a beginner learning the basics or a seasoned chef experimenting with new cuisines, Cookbook is here to guide you every step of the way.

**Team Leader : G. Shreyan**

**Team Members:**

* **V. Punithraj**
* **P. Praveenkumar**
* **J. Sivaganesh**
* **R.Vinothkumar**

**Project Overview:**

**Objective:**

The project aims to design and develop \*Cookbook\*, an intelligent virtual assistant that simplifies the cooking experience by providing personalized recipes, step-by-step cooking guidance, meal planning, and grocery management. The assistant will combine user preferences, dietary needs, and available ingredients to deliver tailored culinary support.

**Key Features:**

Personalized Recipe Suggestions – Curates meals based on taste, dietary restrictions, and available ingredients.

Step-by-Step Cooking Guidance – Provides clear instructions via voice or text for hands-free cooking.

Meal Planning & Scheduling – Helps users organize weekly meals for better time and health management.

Smart Grocery Lists– Automatically generates shopping lists from chosen recipes.

Cooking Tips & Substitutions– Offers alternatives for missing ingredients and shares kitchen hacks.

Interactive & User-Friendly Interface – Ensures an engaging, seamless experience across devices.

**ARCHITECTURE:**

The architecture of the **Cookbook** application is meticulously designed to enhance both functionality and maintainability. The core components—primarily found in App.js and RecipeList.js—serve distinct purposes within the application.

### Component Structure

* **App.js**: This is the main component that initializes the application. It is responsible for setting up the overall layout and routing of the application. This file includes the routing logic using react-router-dom, facilitating seamless navigation between various pages such as the home page, recipe details, and user profiles.
* **RecipeList.js**: This component acts as a container for displaying a list of recipes. It retrieves data from state management using the Context API, allowing for an efficient and reactive user interface that dynamically updates as users interact with the application.

### State Management

The Cookbook employs the **Context API** for state management, providing a global state that can be accessed across various components without prop drilling. This approach allows for efficient sharing of recipe data and user preferences, ensuring that all parts of the application are synchronized and up-to-date.

### Routing Navigation

With the use of react-router-dom, the application supports client-side routing, which enables users to navigate between different views without reloading the browser. Such routing enhances user experience by providing instant feedback and smooth transitions, crucial for maintaining user engagement in recipe exploration.

This architecture not only ensures a clean and organized structure but also lays the groundwork for future scalability.

### PRE-REUISIQTES

Here are the key prerequisites for developing a frontend application using React.js:

* **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/>
* Installation instructions: <https://nodejs.org/en/download/package-manager/>

* **React.js**:

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

* 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](about:blank) in your web browser.

* **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.

* **Version Control**: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

 • Git: Download and installation instructions can be found at: <https://git-scm.com/downloads>

* **Development Environment**: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

• Visual Studio Code: Download from <https://code.visualstudio.com/download>

• Sublime Text: Download from <https://www.sublimetext.com/download>

• WebStorm: Download from [https://www.jetbrains.com/webstorm/download](https://www.jetbrains.com/webstorm/download%20)

To get the Application project from drive:

Follow below steps:

**Install Dependencies:**

• Navigate into the cloned repository directory and install libraries:

cd fitness-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](http://localhost:3000/).

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

 You have successfully installed and set up the application on your local machine. You can now proceed with further customization, development, and testing as needed.

### . Project structure

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 are 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.

### Project setup and configuration

* **Project setup and configuration.**

• **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:

o React Js

o React Router Dom

o React Icons

o Bootstrap/tailwind css

o Axios

• For further reference, use the following resources

o https://react.dev/learn/installation

o https://react-bootstrap-v4.netlify.app/getting-started/introduction/ o https://axios-http.com/docs/intro

o https://reactrouter.com/en/main/start/tutorial

### Project Development

### Setup the Routing paths

Setup the clear routing paths to access various files in the application. Develop the Navbar and Hero components Code the popular categories components and fetch the categories from ***themealsdb*** Api.Also, add the trending dishes in the home page. Now, develop the category page to display various dishes under the category. Finally, code the recipe page, where the ingredients, instructions and a demo video will be integrated to make cooking much easier.

**Important Code snips:**

**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 useState HooK**

The code utilizes the useState hook to create 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 (fetch Categories 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 fetch Categories 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. BycallingsetCategories(response.data.categories), the component's state is updated with the fetched list of meal categories.

**Fetching the food items under a particular category**

Now, with the API request, we fetch all the available food items under the certain category. This React code snippet manages data fetching from an API.

* + 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.

**Fetching Recipe details**

With the recipe id, we fetch the details of a certain recipe. This React code manages fetching recipe data from an API and storing it within a state variable.

* + It leverages the useState hook to establish a state variable named recipe (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 state using the setRecipie function. This function, associated with the useState hook, allows for modification of the recipe 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.

### Installation Steps

1. **Clone the Repository**by opening the terminal or command prompt and run the following command:

* git clone https://github.com/<your-username>/cookbook.git
* Replace <your-username> with your GitHub username.

1. **Navigate to the Project Folder** Change into the project directory by executing:

* cdreact-demo1

1. **Install Dependencies** Install the necessary packages by running:

* npm install

1. **Start the Development Server** Launch the application with the following command:

* npm start
* This should open your default web browser at http://localhost:3000, where you can see the **Cookbook** application in action.

### Project Folder Structure

The Cookbook project follows a structured folder layout to facilitate easy navigation and understanding.

* **/src**: Contains the core application code.
  + **/components**: Holds reusable UI components.
  + **/data**: Includes Context API setup for state management.
  + **/pages**: Contains different views or pages of the app.

This structure aids both new developers and project maintainers in locating relevant files promptly.

## Running the Application and Component Documentation

To launch the **Cookbook** application, follow these straightforward steps:

1. **Start the Development Server**: After completing the setup instructions, execute the following command in your terminal:

* npm start
* The application will be accessible at http://localhost:3000.

### Key Components

#### RecipeCard.js

The RecipeCard component is crucial for displaying individual recipes in a visually appealing format. It includes:

* **Props**: Receives details like title, image, and summary.
* **Functionality**: Allows users to view recipe details and navigate to the corresponding page when clicked.

#### RecipeDetail.js

The RecipeDetail component provides an in-depth view of a selected recipe.

* **Props**: Accepts recipe id to fetch relevant data.
* **Features**: Displays ingredients, instructions, and user reviews, ensuring users have all the information they need at their fingertips.

These components form the backbone of user interaction in the Cookbook application, enhancing the overall user experience.

## User Interface and Styling

The **Cookbook** application boasts an intuitive user interface that prioritizes ease of use and aesthetics.

### Layout and Responsive Design

The layout is designed with flexibility in mind, utilizing a **responsive design** approach. This ensures that users can enjoy a seamless experience across various devices, from desktops to tablets and smartphones. Key features include:

* **Grid-based Structure**: Recipes are arranged in an easily navigable grid format.
* **Mobile Optimization**: Touch-friendly elements enhance usability on mobile devices.

### Styling Approach

The application employs robust CSS frameworks, including **Styled-components** and **Bootstrap**, to create a visually appealing UI.

* **Styled-components**: Enable scoped styling for components, facilitating maintainable and dynamic designs.
* **Bootstrap**: Provides pre-defined styles and responsive grid systems, accelerating development time while ensuring consistency.

Together, these tools contribute to a polished and engaging user experience within the Cookbook application.

## Testing and Future Enhancements

### Testing Strategy

To ensure the reliability and maintainability of the **Cookbook** application, a testing strategy focusing on **unit** and **integration testing** has been implemented, utilizing **Jest** and **React Testing Library**.

* **Unit Testing**: This involves testing individual components in isolation to ensure that each function behaves as expected. Key unit tests include:
  + Verifying the rendering of each component (e.g., RecipeCard, RecipeDetail).
  + Testing utility functions that handle recipe data manipulation.
* **Integration Testing**: This approach tests how components work together within the application. It covers scenarios such as:
  + User interactions, like adding or editing recipes.
  + Ensuring the Context API correctly updates and reflects states across different components.

**Screenshots or Demo**

* Link to a demo showcasing the application’s features and design :

<http://sensational-narwhal-15726b.netlify.app>

**KNOWN ISSUES:**

* It may have some performance inexperience on giving the exact search result due to the inrefinement.
* We can’t save our progress in the application.

**FUTURE ENCHANCEMENTS:**

* 1. Voice Integration with Smart Devices

Seamless integration with Alexa, Google Assistant, kitchen smart appliances for hands-free operation.

* 2. Augmented Reality (AR) Cooking Mode
* Step-by-step visual guidance projected onto real ingredients or utensils through AR glasses or a Smartphone camera.
* 3. AI-Powered Nutrition Advisor

Advanced health tracking with calorie counting, nutrient breakdown, and diet plan suggestions based on fitness goals.