CookBook: Your Virtual Kitchen Assistant

(React Application)

TEAM ID : SWTID1741242031155530
TEAM LEADER: 1. Kinsu Sharma – kinucs8@gmail.com
TEAM MEMBERS:
2. Aysha Rukshana A – aisha.rukshana@gmail.com
3. Manju Sri S – manjumanjusri.16@gmail.com
4. Kaveya K – k.kaveya10@gmail.com
5. Deepika – dm6997584@gmail.com

Abstract:

For the modern man living life at breakneck speed, virtual assistants have become indispensable for improving the user experience in many areas, especially cooking. This project titled Cook book: Your Virtual Assistant Using React Application will try to develop an interactive intelligent recipe manager system using React.js. The application shall give the users an AI-powered virtual assistant that gives recipe suggestions, shopping lists, and guides them step-by-step through cooking instructions.

The system will use Natural Language Processing (NLP) to allow the virtual assistant to comprehend what the user likes, its restrictions on what food it csan consume, and what food items are readily available. This will ensure that every recipe the user gets is personal. There will be an inclusion of voice commands for its hands-free searching. This will expedite cooking. Also, React.js will give a better dynamic user interface while being extremely responsive, which means Windows, Apple, and Android users can access it smoothly.

Introduction:

With the rapid pace of life in modern times, technology has transformed our approach to mundane activities, and cooking is no exception. Given the growing trend of using virtual assistants, applying smart technologies in recipe organization and meal planning will significantly improve efficiency and convenience. “Cook book: Your Virtual Assistant Using React Application” is an online tool that aims to help users discover, plan, and prepare meals with ease.

Developed with React.js, this application harnesses the strength of new web development to present a smooth, interactive, and intuitive experience. React’s component-based structure provides efficient state management, quicker updates, and dynamic rendering, making it a perfect fit for developing a responsive and interactive application. With the integration of Artificial Intelligence (AI) and Natural Language Processing (NLP), the virtual assistant is able to comprehend user tastes, dietary needs, and ingredients on hand to offer tailored recipe suggestions.

One of the most prominent characteristics of this software is that it can handle voice commands, with users being able to use the assistant without needing to use their hands while

preparing meals. Moreover, the framework includes APIs for ingredient identification, nutritional breakdowns, and step-by-step recipe instructions, allowing for easier preparation of meals. Users can create shopping lists, activate cooking timers, and be provided with real-time support to make cooking simple and convenient.

By integrating contemporary web technologies with AI-powered assistance, this project seeks to make meal planning easy, minimize food loss, and elevate the overall cooking experience of users with varying levels of expertise.

Project Overview:

CookBook's primary goal is to offer an easy-to-use platform that allows users to explore new recipes, save their favorites, and share culinary creations. The application provides a rich user interface where users can browse through recipe categories, access trending dishes, and view recipe details, including ingredients, instructions, and videos.

CookBook integrates with various external APIs via RapidAPI to bring an enriched recipe database and deliver dynamic search capabilities to help users find exactly what they are looking for. The platform's main features include:

- Recipe search by categories, ingredients, and cuisines.
- Ability to save favorite recipes and manage personal collections.
- A visually appealing interface with easy navigation.
- Integration of external recipe APIs to offer a vast recipe library.

The web application is built using **React.js** to ensure a dynamic and responsive user interface, with state management powered by React's hooks and Axios for making HTTP requests to external APIs.

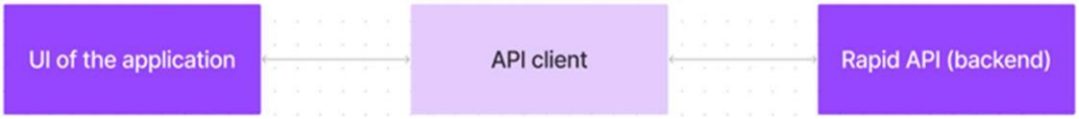
Module Description:

Core Modules of CookBook:

1. **Recipe Search Module:**
   - Enables users to search for recipes based on ingredients, categories, or cuisines.
   - Integrates with the MealsDB API to fetch recipes based on user input.
2. **Recipe Details Module:**
   - Displays detailed information about a recipe, including ingredients, cooking instructions, and tutorial videos.
   - Allows users to view recipes, bookmark them, and share with friends.
3. **User Account Management Module:**
   - Allows users to create accounts, log in, and manage their favorite recipes.
   - Implements authentication and personalized recipe management.
4. **Trending Dishes Module:**
   - Displays popular and trending dishes on the home page, helping users discover what's currently popular.
5. **Category Browsing Module:**

- Allows users to browse recipes by category (e.g., vegetarian, desserts, etc.), making it easy to explore different types of dishes.

Technical 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.

System Analysis:

Feasibility Study

The feasibility study of the CookBook web application covers three main aspects:

1. **Technical Feasibility:**
   - React.js is a well-established framework for building single-page applications (SPA), making it suitable for creating dynamic, real-time user experiences.
   - Integration with external APIs (e.g., MealsDB) through RapidAPI ensures access to a diverse range of recipes without the need to build a backend infrastructure from scratch.
2. **Operational Feasibility:**
   - The platform is designed to be user-friendly, with a focus on intuitive navigation and easy access to recipes, making it accessible to a wide range of users from beginners to professional chefs.
3. **Economic Feasibility:**
   - By utilizing free external APIs (e.g., MealsDB) and open-source libraries (e.g., React.js, Axios), the initial development cost is minimized.
   - The system does not require expensive hardware and software, which ensures that it is cost-effective in terms of development and maintenance.

Existing System:

Currently, there are multiple recipe platforms and applications available, such as AllRecipes, Yummly, and Epicurious. These platforms allow users to search and save recipes, but they often lack comprehensive personalization, community-driven content, or dynamic features. CookBook aims to overcome these shortcomings by offering a more personalized and interactive experience, enabling users to not only discover recipes but also share them and explore them based on categories, trends, and ingredients.

Support System:

CookBook will be supported by:

- **Regular updates and bug fixes**: The system will receive periodic updates to improve functionality, address bugs, and introduce new features.
- **Customer support and feedback**: A support channel will be made available for users to report issues or suggest improvements, ensuring continuous development

System Testing:

System testing involves several testing phases to ensure that the application performs as expected. The following are the key types of testing carried out during the development of CookBook:

1. Unit Testing

- Unit testing focuses on testing individual components or modules of the application, ensuring that each part functions as expected. For example, testing the Recipe Search module to ensure it correctly filters recipes based on the user's query.

2. Integration Testing

- Integration testing ensures that different modules of the application work together as intended. For example, testing the interaction between the search functionality and the API call to fetch recipes.

3. Functional Testing

- Functional testing verifies that the features of the application work as described in the requirements. For instance, testing whether the save recipe functionality correctly adds a recipe to the user's favorites list.

4. Whitebox Testing

- Whitebox testing is carried out to test the internal workings of the application. It ensures that the logic behind fetching and processing recipe data is correct.

5. Blackbox Testing

- Blackbox testing is done from the user's perspective, testing the application's functionality without knowing its internal workings. It focuses on user interface and ensuring that the app behaves as expected from the user's point of view.

Features of CookBooks:

✓ 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.

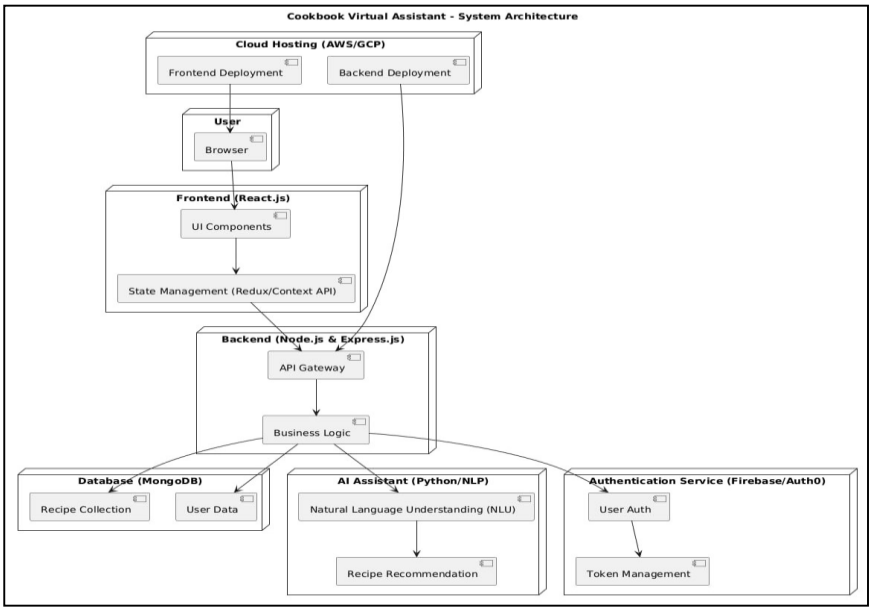
✓ Search feature: various dishes can be accessed easily through the search feature

System Architecture Diagram

The system architecture diagram illustrates the high-level structure of the CookBook application. Below is a conceptual overview of how the system components interact:

```
pgsql                                                                    Copy

+-----------------------------------+
|          User Interface           |
| (React.js Components and Pages)   |
+-----------------------------------+
              |
              v
+-----------------------------------+      +-------------------+
|     Axios HTTP Requests           |<---->|   MealsDB API     |
|   (Fetch Data from RapidAPI)      |      | (Recipe Database) |
+-----------------------------------+      +-------------------+
              |
              v
+-----------------------------------+
|   User Authentication & Storage   |
|  (Local Storage/Browser Storage)  |
+-----------------------------------+
```



System Configuration

Hardware Requirements:

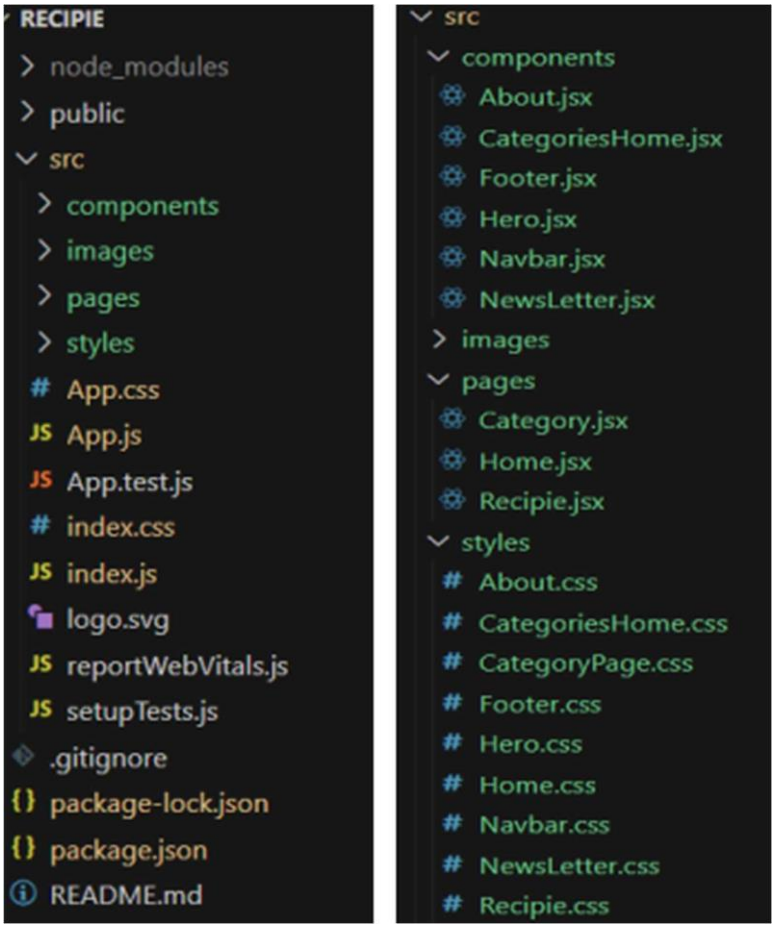
1. **Minimum Requirements:**
   - Processor: 1.8 GHz or higher.
   - RAM: 4 GB or more.
   - Storage: 1 GB of available space for installation and resources.
   - Internet Connection: Required for API requests and updates.
2. **Recommended Requirements:**
   - Processor: 2.4 GHz or higher (Intel i5 or equivalent).
   - RAM: 8 GB or more.
   - Storage: 5 GB of available space for the application and resources.
   - Internet Connection: High-speed connection for optimal experience.

Software Requirements:

1. **Operating System:**
   - Windows 10 or higher, macOS, or Linux.
2. **Software:**
   - **Node.js** and **npm** (for running JavaScript and managing dependencies).
   - **React.js** (for building the user interface).
   - **Axios** (for API requests).
   - **Bootstrap** or **Tailwind CSS** (for styling the application).
   - **Visual Studio Code** or any IDE for development.
3. **Browsers:**
   - Google Chrome, Firefox, or Microsoft Edge for best compatibility.

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'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.

Milestone 2: Project Development

- ❖ Setup the Routing paths

Setup the clear routing paths to access various files in the application.

```
<Routes>

 <Route path="/" element={<Home />} />
 <Route path="/category/:id" element={<Category />} />
 <Route path="/recipie/:id" element={<Recipie />} />
</Routes>
```

❖ 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 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.

➢ **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.

- It leverages the useState hook to establish a state variable named categories. This variable acts as a container to hold the fetched data, which is initially set to an empty array [].
- 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 (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.
- The fetchCategories function is an asynchronous function responsible for handling the API interaction. This function utilizes the axios.get method to make a GET request to a predetermined 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.

- 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 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.
- 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.

➢ **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 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 state using the setRecipie function. This function, associated with the useState hook, allows for 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.

User Interface snips:

➢ **Hero components**

The hero component of the application provides a brief description about our application and a button to view more recipes.

➢ **Popular categories**

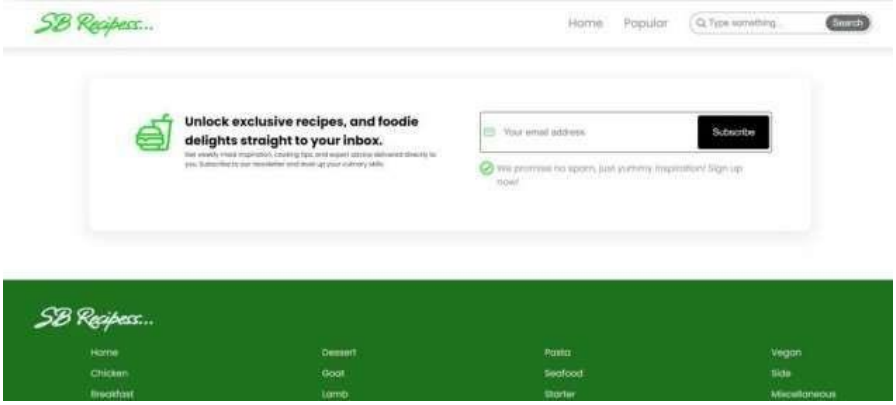
This component contains all the popular categories of recipes..

➢ **Trending Dishes**

This component contains some of the trending dishes in this application.

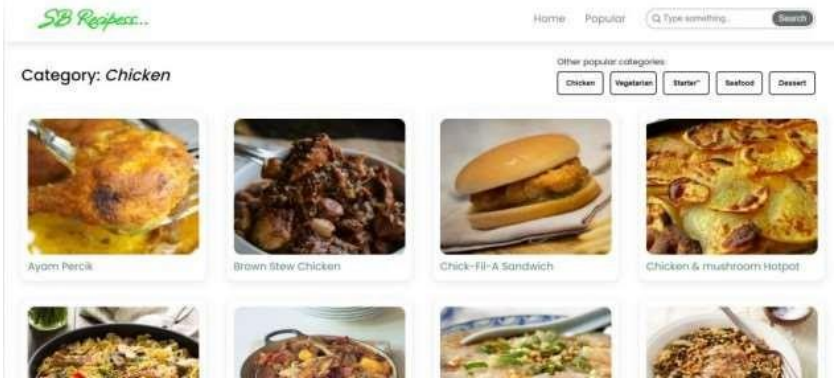
➢ **News Letter**

The news letter component provides an email input to subscribe for the recipe newsletters.



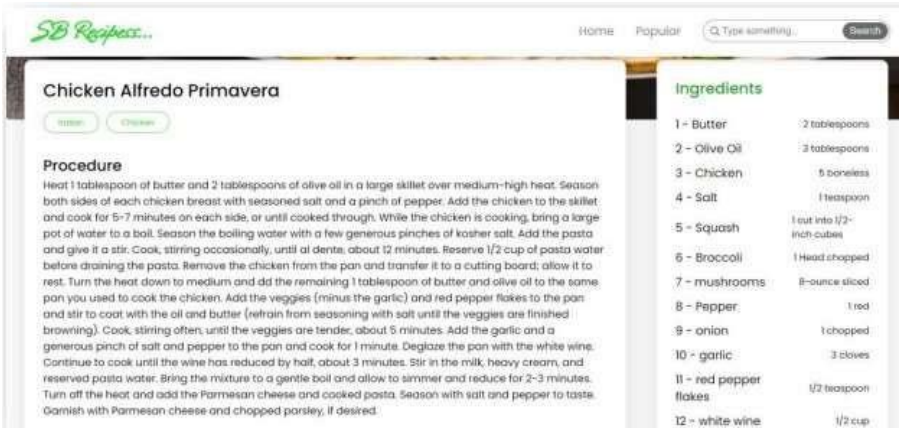
➢ **Category dishes page**

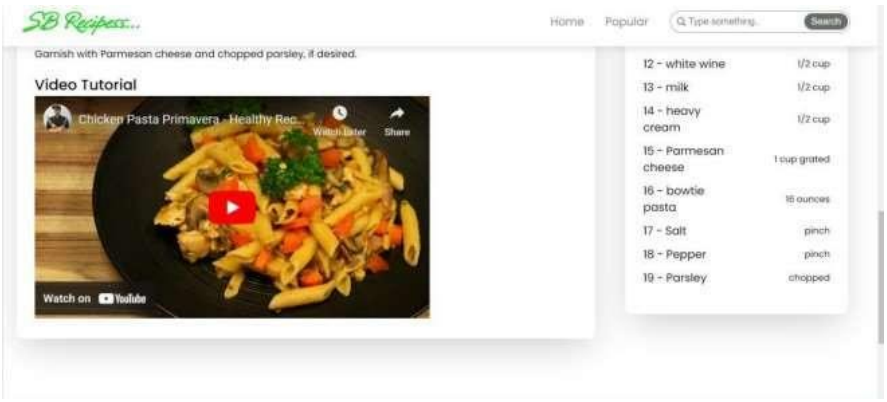
The category page contains the list of dishes under a certain category.



➢ **Recipe page**

The images provided below shows the recipe page, that includes images, recipe instructions, ingredients and even a tutorial video.





Conclusion:

CookBook is an innovative web application that simplifies the culinary journey by offering easy access to diverse recipes, enabling users to explore new cooking techniques and share their creations with a community of fellow cooking enthusiasts. The platform leverages modern web technologies, integrates with external APIs, and provides a robust system architecture to ensure a seamless and engaging user experience. Through thorough testing and careful design, CookBook ensures that users can navigate the app with ease, discover new dishes, and enhance their cooking skills.

PROJECT DEMO LINK :

https://drive.google.com/file/d/1OejPt7VgbIkS8blSWa29JHbAzrVz6NLW/view?usp=drives dk