**Project Documentation**

CookBook: Your Virtual Kitchen Assistant.

**1. Introduction**

### • Project Title: CookBook: Your Virtual Kitchen Assistant.

### • Team ID: NM2025TMID33776

### • Team Leader: BAJERIA KANI A

### • Team Members: REEM FATHIMA H FAZILA FATHIMA A SASMILA RASHIN S

**2. Project Overview**

• **Purpose:** A cookbook project can serve various purposes such as documenting family recipes, creating a tool for learning and skill development in cooking, sharing a specific cuisine,promoting food literacy and community empowerment or providing an accessible resource for health-conscious eaters.

• **Features:**

Welcome to the forefront of culinary exploration with CookBook!

Our cutting-edge web application is meticulously crafted to transcend the boundaries of culinary experiences, catering to the tastes of both passionate cooking enthusiasts, and seasoned professional chefs. With an emphasis on an intuitive user interface and a robust feature set, CookBook is poised to revolutionize the entire recipe discovery, organization, and creation process.

Designed with a commitment to user-friendly aesthetics, CookBook immerses users in an unparalleled culinary adventure. Navigate seamlessly through a vast expanse of culinary inspiration with features such as dynamic search effortlessly.

From those taking their first steps in the kitchen to seasoned professionals, CookBook embraces a diverse audience, nurturing a dynamic community united by a shared passion for the art of cooking. Our vision is to reshape how users interact with recipes, presenting a platform that not only sparks inspiration but also fosters collaboration and sharing within the vibrant culinary community.

Embark on this gastronomic journey with us, where innovation seamlessly intertwines with tradition. Every click within CookBook propels you closer to a realm of delicious possibilities. Join us and experience the evolution of recipe management, where each feature is meticulously crafted to offer a glimpse into the future of culinary exploration. Elevate your culinary endeavours with CookBook, where every recipe becomes an adventure waiting to be discovered and savoured.

**3. Architecture**

• **Frontend:** React.js with Bootstrap and Material UI

• **Backend:** Node.js and Express.js managing server logic and API endpoints

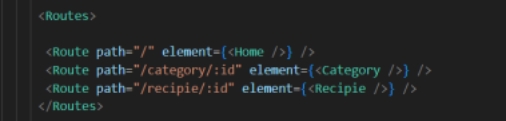
• **Database:** MongoDB stores user data, project information, applications, and chat

Messages



? 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.

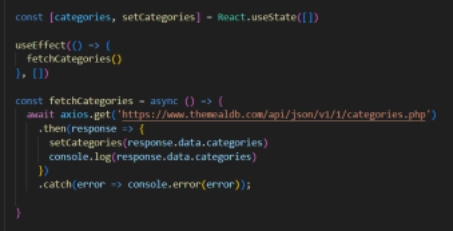
? Now, develop the category page to display various dishes under the category.

? 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 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:

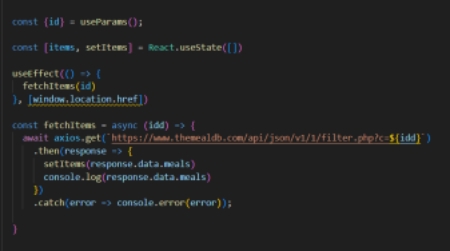
An asynchronous function named fetchCategories is defined to handle the API interaction. This function utilizes the axios.get method to make a GET request to a specified API endpoint (https://www.themealdb.com/api/json/vi/1/categories.php in this example). This particular endpoint presumably returns 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.

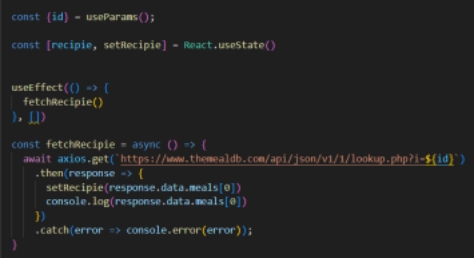
The fetchCategories function is an asynchronous function responsible for handling the API interaction. This function utilizes the axios.get method to make a GET request to a predetermined API endpoint (https://www.themealdb.com/api/json/vi/1/categories.php in this example). This particular endpoint presumably 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.

**4. Setup Instructions**

• **Prerequisites:**

Here are the key prerequisites for devel+oping 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](https://naanmudhalvan.smartinternz.com/Student/guided_project_info/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

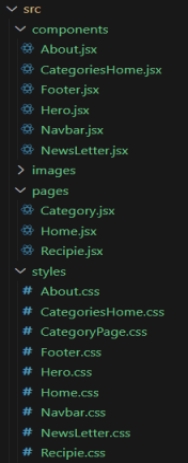
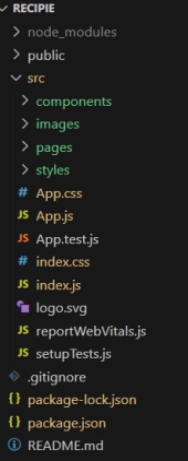
**Access the App:**

• 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.

1. **Folder 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.

**6. Running the Application**

• **Frontend:**

cd client

npm start •

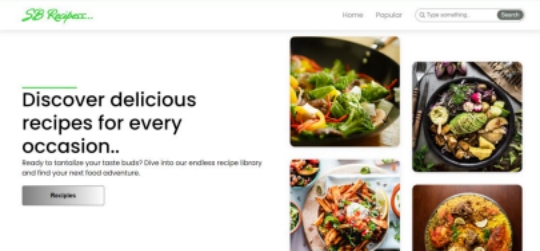
**Backend:**

cd server npm

start

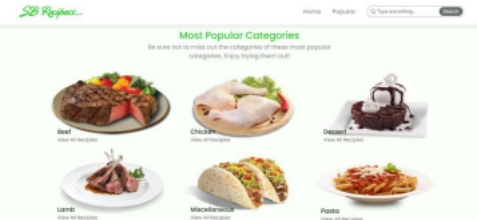
**? Hero components**

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



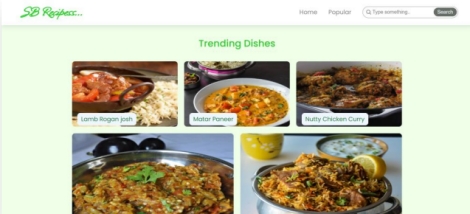
? **Popular categories**

This component contains all the popular categories of recipes..



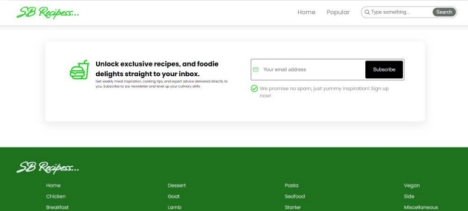
? **Trending Dishes**

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



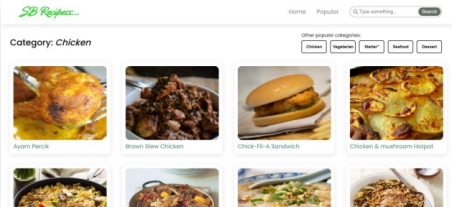
? **News Letter**

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



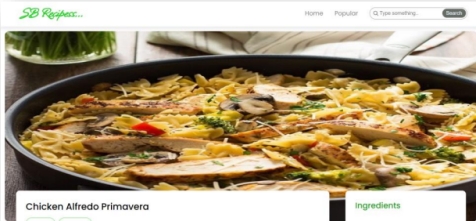
**? Category dishes page**

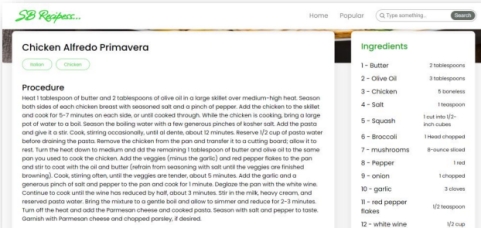
The category page contains the list of dishes under a certain category.

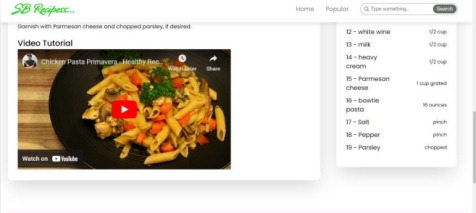


**? Recipe page**

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







**7. API Documentation**

• **User:**

**–** /api/user/register

**–** /api/user/login

• **Projects:–** /api/projects/create

**–** /api/projects/:id • **Applications:** /api/apply

• **Chats:**

**–** /api/chat/send

**–** /api/chat/:userId

**8. Authentication**

• JWT-based authentication for secure login

• Middleware protects private routes

**9. User Interface**

• Landing Page

• Freelancer Dashboard

• Admin Panel

• Project Details Page

**10. Testing**

• Manual testing during milestones

• Tools: Postman, Chrome Dev Tools

**11. Screenshots or Demo**

**12. Known Issues**

**13. Future Enhancements**