

# **Frontend Project**

## **Project title: Cook Book: Your Virtual Kitchen Assistant**

### **Team member**

1. Team Leader -M . Logeswaran

Email – [logeshm@gmail.com](mailto:logeshm@gmail.com)

2. Speaker About the Project – A.Abdul Rahim

Email- arahim96142@gmail.com

3. Document Creator – Anandha Kumar.M

Email – [rossyanand46@gmail.com](mailto:rossyanand46@gmail.com)

## Introduction:

Cook Book 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.

## Project overview:

- **Purpose:**

Cook Book 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.

- **Features of Cook Books:**

1. Recipes from the Meals DB API: Access a vast library of international recipes spanning diverse cuisines and dietary needs.
2. ✓ Visual recipe browsing: Explore recipe categories and discover new dishes through curated image galleries.
3. ✓ Intuitive and user-friendly design: Navigate the app effortlessly with a clean, modern interface and clear navigation.
4. ✓ Search feature: various dishes can be accessed easily through the search feature.

## Technical Architecture:

The user experience starts with the Cook Books 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 Cook Books. 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 Cook Books might integrate external data feeds or functionalities through Rapid API, enriching the user experience without building everything from scratch.

## Setup Instructions

**PRE-REQUISITES:** 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` 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.

**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> To clone and run the Application project from Google drive: Follow below steps:

#### **Get the code:**

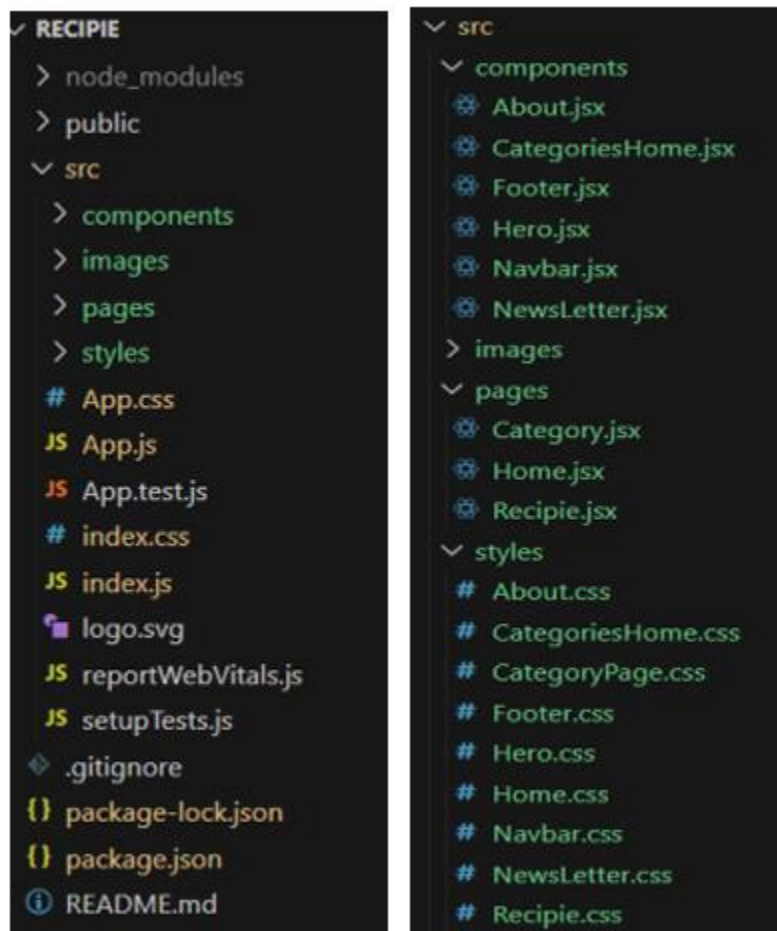
- Download the code from the drive link given below:  
[https://drive.google.com/drive/folders/1u8PnV\\_mE0mwKkH\\_CvuNpliZtRLJZMqrO?usp=sharing](https://drive.google.com/drive/folders/1u8PnV_mE0mwKkH_CvuNpliZtRLJZMqrO?usp=sharing)

- Navigate into the cloned repository directory and install libraries: `cd recipe-app-rea`  
`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. 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.

## 5. 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 act as pages at different url's in the application. The components folder stores all the files, that return the small components in the application. All the styling css files will be stored in the styles folder.

Project Flow: Project demo: Before starting to work on this project, let's see the demo. Demo link:

<https://drive.google.com/file/d/1khMJkccySgKyqRaEZgCpgDACHi572Llj/viewusp=sharing>

**Use the code in:**

<https://github.com/2226054I/CookBook-SWTID1741258526155047>

**Milestone 1:** 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

<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>

**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 themealdb 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 the available categories.

```
const [categories, setCategories] = React.useState([])

useEffect(() => {
  fetchCategories()
}, [])

const fetchCategories = async () => {
  await axios.get('https://www.themealdb.com/api/json/v1/1/categories.php')
    .then(response => {
      setCategories(response.data.categories)
      console.log(response.data.categories)
    })
    .catch(error => console.error(error));
}
```

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/v1/1/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 `[]`.



```

const {id} = useParams();

const [items, setItems] = React.useState([])

useEffect(() => {
  fetchItems(id)
}, [window.location.href])

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)
      console.log(response.data.meals)
    })
    .catch(error => console.error(error));
}
)

```

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/v1/1/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.

```
const {id} = useParams();

const [recipe, setRecipe] = React.useState()

useEffect(() => {
  fetchRecipe()
}, [])

const fetchRecipe = async () => {
  await axios.get(`https://www.themealdb.com/api/json/v1/1/lookup.php?i=${id}`)
    .then(response => {
      setRecipe(response.data.meals[0])
      console.log(response.data.meals[0])
    })
    .catch(error => console.error(error));
}
```

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 (`fetchRecipe` 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 `fetchRecipe` 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 `setRecipe` function. This function, associated with the `useState` hook, allows for modification of the `recipe` state variable. By calling `setRecipe(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..

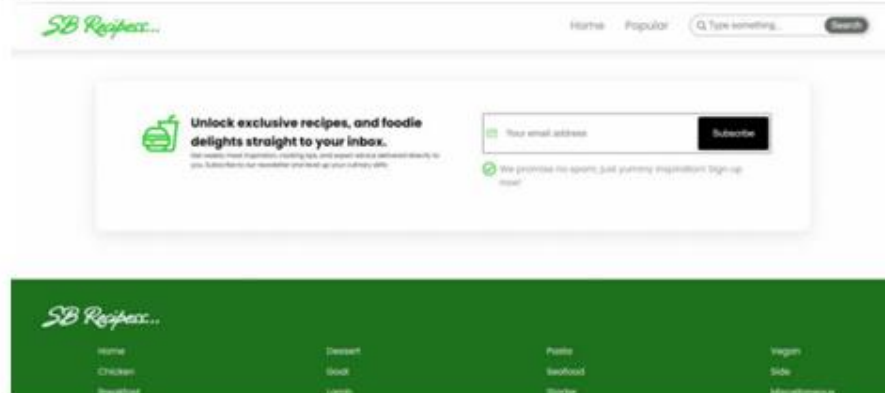


➤ Trending Dishes This component contains some of the trending dishes in this application.



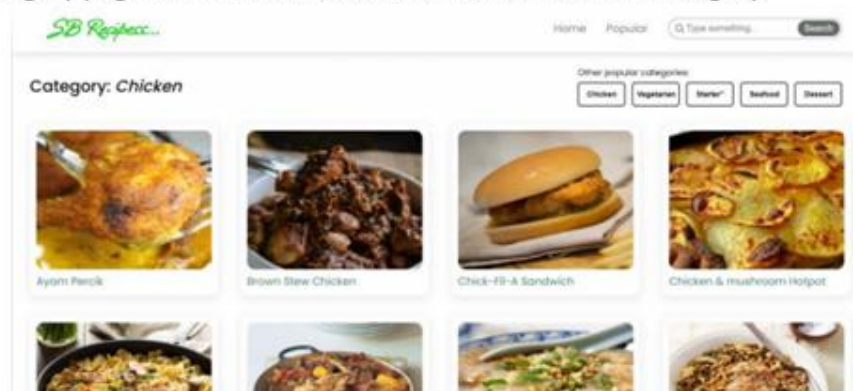
### ➤ News Letter

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



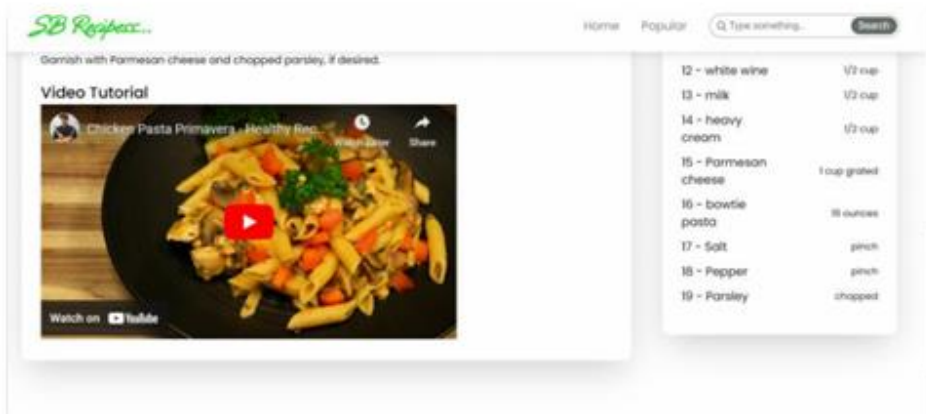
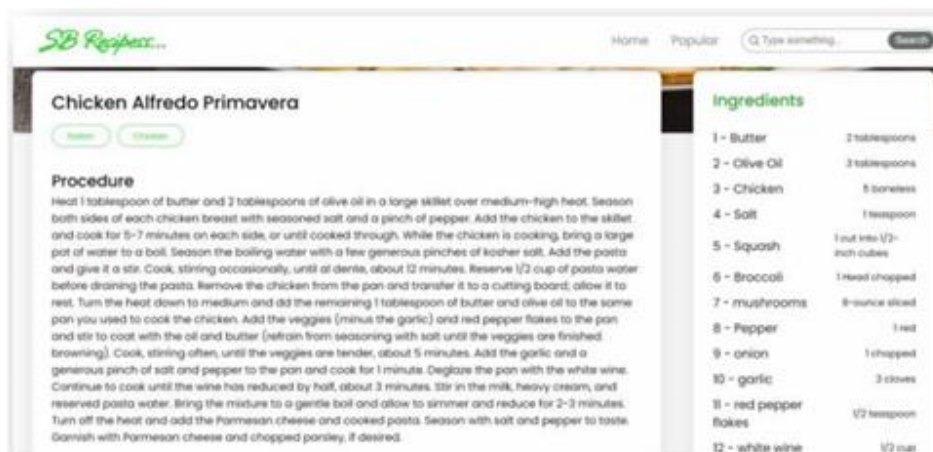
### ➤ Category dishes page

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.



**Demo link:**

<https://drive.google.com/drive/folders/1q9W8Dz7MJQOUexIXs2SiWSBZCkZEXh-N>

THANK YOU