Cook Book: Your Virtual Kitchen Assistant

(React Application)

Introduction:

Cook Book is an innovative web platform that transforms how you find, manage, and create recipes. It is designed for both beginner and expert cooks, providing an easy-to-use interface, powerful tools, and a wide range of recipes to inspire culinary creations.

Description:

Welcome to the forefront of culinary exploration with Cook Book!

Our advanced web application is carefully designed to redefine the culinary experience, catering to both passionate home cooks and experienced professional chefs. With an emphasis on a user-friendly interface and a comprehensive set of features, Cook Book is set to revolutionize the way recipes are discovered, organized, and created.

Crafted with a focus on simplicity and aesthetic appeal, Cook Book provides an immersive culinary experience. Users can easily explore a wide range of recipe inspiration with tools like dynamic search, making navigation effortless.

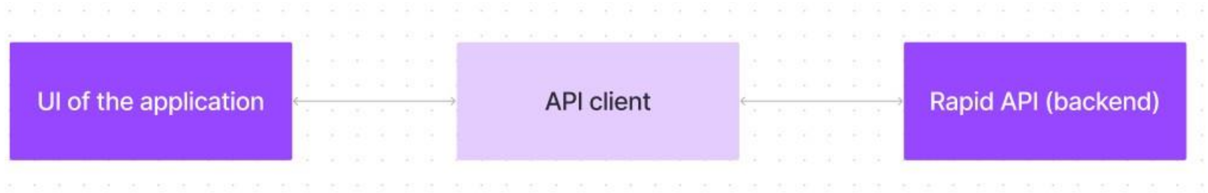
Cook Book is inclusive, welcoming everyone from beginners to expert chefs, and fostering a vibrant community of individuals who share a love for cooking. Our goal is to transform the way users engage with recipes, offering a platform that not only inspires but also encourages collaboration and sharing within the culinary world.

Join us on this exciting journey, where innovation meets tradition, and every interaction with Cook Book brings you closer to endless culinary possibilities. With each feature thoughtfully designed to enhance your cooking adventure, Cook Book is the future of recipe management. Elevate your cooking experience with us, where every recipe is a new exploration waiting to be discovered and enjoyed.

Scenario based introduction:

Sarah rummaged through the fridge, the fluorescent light casting an unappetizing glow on the wilting lettuce and forgotten container of yogurt. Dinnertime with her teenage son, Ethan, was fast approaching, and her usual creative spark was missing. "What are we even going to eat?" Ethan groaned from the door way his phone glued to his ear. Suddenly, a memory surfaced. Her friend, Maya, had been raving about a new recipe platform called Cook Book. Intrigued by the promise of "elevating culinary endeavours" and "a realm of delicious possibilities," Sarah grabbed her laptop. "Hold that thought, Ethan," she declared, a flicker of hope igniting in her eyes. "We might just be about to embark on a delicious adventure."

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.

Project Goals and Objectives:

The primary goal of Cook Book 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 of Cook Books:

✓ **Recipes from the Meals DB 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.

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

Install Dependencies:

- Navigate into the cloned repository directory and install libraries:

```
cd recipe-app-react
npm install
```

✓ **Start the Development Server**:

- To start the development server, execute the following command:

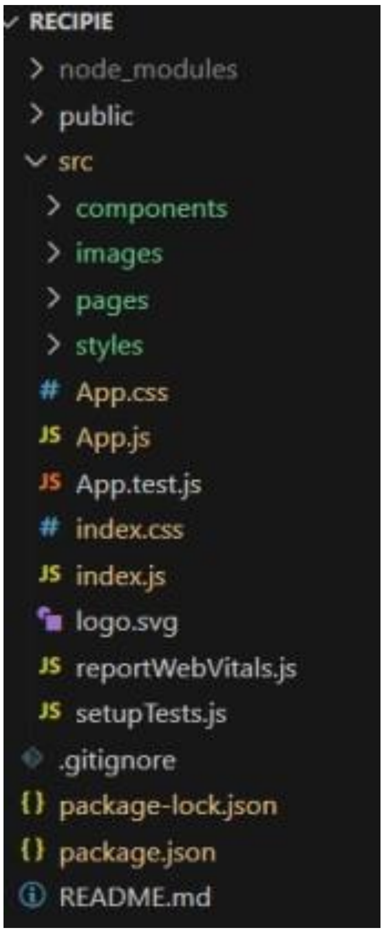
```
npm start
```

Access the App:

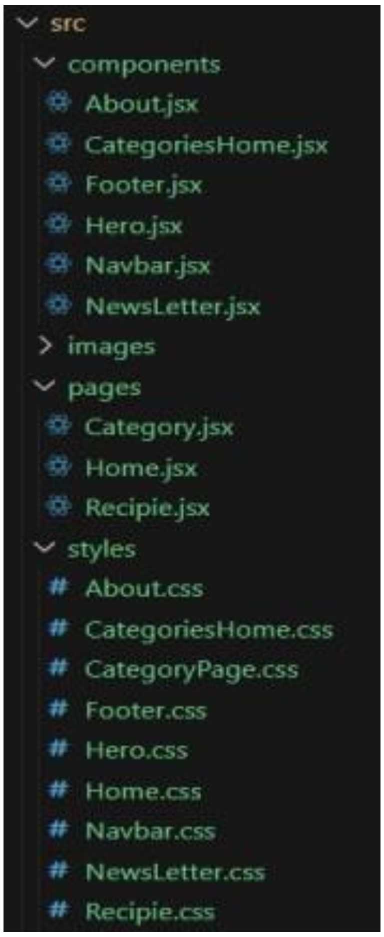
- Open your web browser and navigate to http://localhost:3000.
- You should see the recipe app's homepage, indicating that the installation and setup were successful.

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.

Project structure:



```
RECIPIE
  > node_modules
  > public
  v src
    > components
    > images
    > pages
    > styles
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
    reportWebVitals.js
    setupTests.js
  .gitignore
  package-lock.json
  package.json
  README.md
```



```
v src
  v components
    About.jsx
    CategoriesHome.jsx
    Footer.jsx
    Hero.jsx
    Navbar.jsx
    NewsLetter.jsx
  > images
  v pages
    Category.jsx
    Home.jsx
    Recipie.jsx
  v styles
    About.css
    CategoriesHome.css
    CategoryPage.css
    Footer.css
    Hero.css
    Home.css
    Navbar.css
    NewsLetter.css
    Recipie.css
```

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.

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/view?usp=sharing

Use the code in:

https://drive.google.com/drive/folders/1u8PnV_mE0mwKkH_CvuNpliZtRLJZMqrO?usp=sharing **Milestone 1:**

Project setup and configuration.

- **Installation of required tools**:

  To build Cook Book, 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.

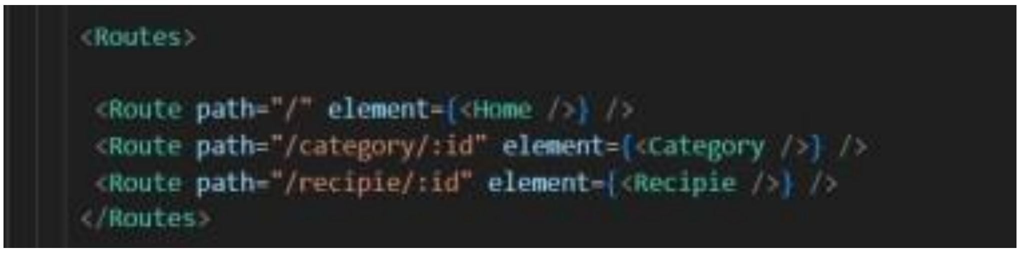
  Open the project folder to install necessary tools, In this project, we use:
    - React Js
    - React Router Dom
    - React Icons
    - Bootstrap/tailwind css
    - Axios

- 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

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 ***the meals db*** Api.

❖ Also, add the trending dishes in the home page.

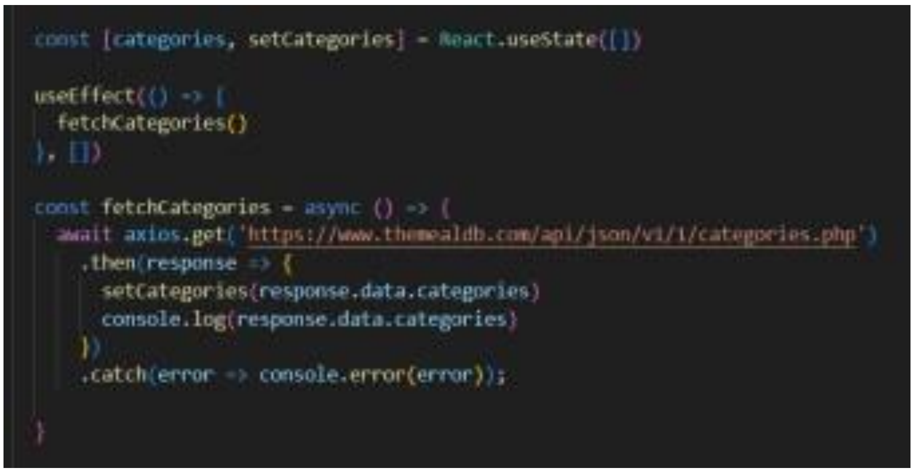
❖ Now, develop the category page to display various dishes under the category.

❖ 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.



```
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 use State 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 use Effect Hook:

The use Effect hook in React is used to handle side effects, like fetching data from an API. It accepts a callback function, such as fetch Categories, which is executed after the component renders. Additionally, it takes an optional dependency array that determines when the effect should re-run. If the array is empty ([]), it means the effect runs only once, immediately after the component mounts. This setup is useful for one-time tasks like initial data fetching. Without dependencies, the effect doesn't run again unless the component is re-mounted, ensuring efficient performance.

Fetching Data with fetch Categories Function:

An asynchronous function named fetch Categories 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.

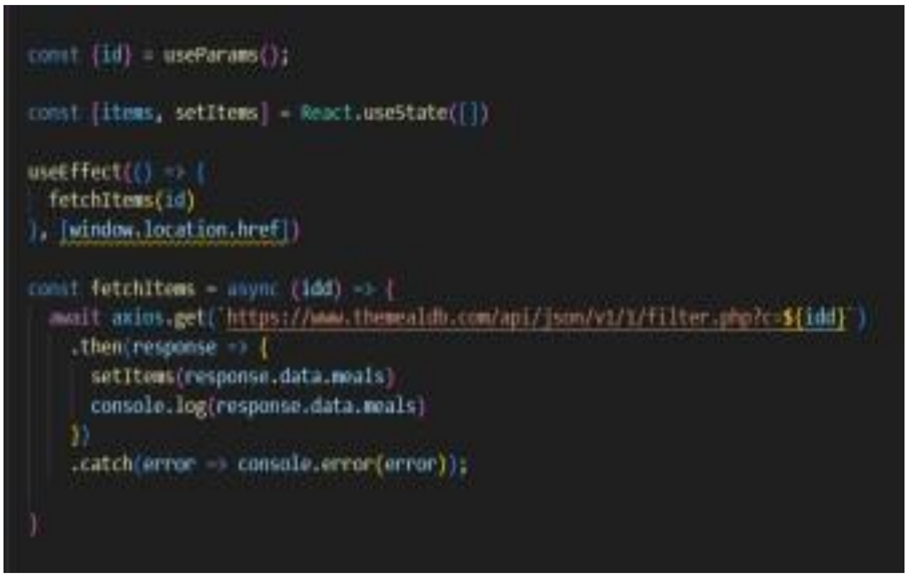
Processing API Response:

The then method is used after the axios. get call to manage a successful API response. Inside the then block, the categories data is extracted from the response and the component's state is updated using the set Categories function. This function, which is tied to the use State hook, allows the categories state variable to be changed. By calling set Categories (response data categories), the component's state is updated with the list of meal categories fetched from the API. This ensures the component reflects the newly retrieved data.

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

To fetch food items under a specific category, you can make an API request using a method like axios get, passing the category ID or name as a parameter in the URL. After receiving the response, you can use the then method to handle the successful data retrieval. Inside the then block, extract the list of food items from the response and update the component's state using a function like set Food Items. This updates the UI to display the food items specific to the selected category. This process ensures that the relevant items are displayed for the chosen category.

Now, with the API request, we fetch all the available food items under the certain category.



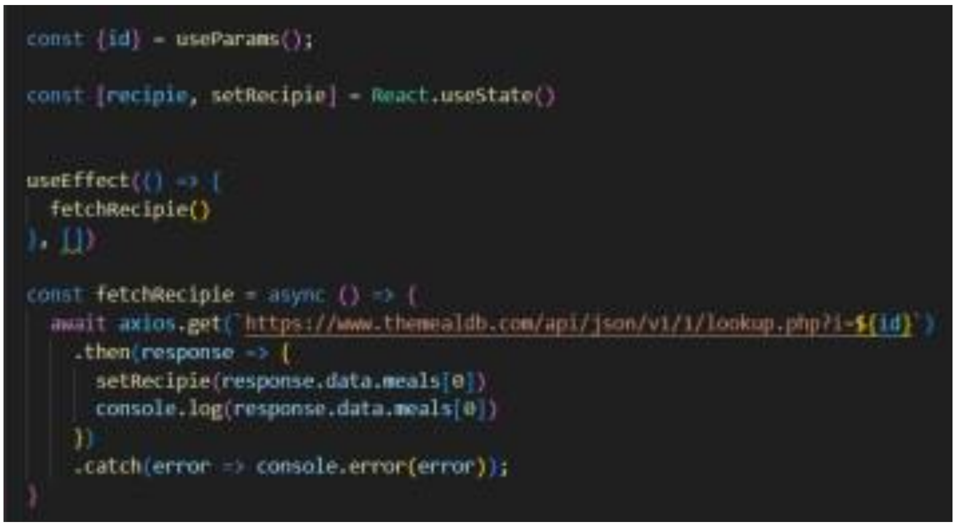
```
const {id} = useParams();

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.

- The use State hook is used to create a state variable called categories, which serves as a storage for the fetched data. Initially, it is set to an empty array [].
- The use Effect hook is used to perform the side effect of fetching data from an API. It takes a callback function (like fetch Categories) and an optional dependency array. The callback is triggered after the component renders and whenever dependencies change. With an empty dependency array ([]), the fetching occurs only once, right after the component mounts.
- The fetch Categories function is asynchronous and handles the API call. It uses axios.get to send a GET request to an API endpoint (e.g., https://www.themealdb.com/api/json/vi/1/categories.php), which returns a list of meal categories in JSON format.
  - • The `then` method is chained to the `axios. get` call to handle a successful response. Inside the `then` block, the categories data is extracted and the component's state is updated using the `set Categories` function. This updates the `categories` state with the fetched data.
  - • An optional error handling mechanism is implemented with a `catch` block. This block catches any errors during the API request and logs the error details to the console using `console error`. It helps identify and troubleshoot issues in the data fetching process.
- 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 [recipie, setRecipie] = React.useState()


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

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

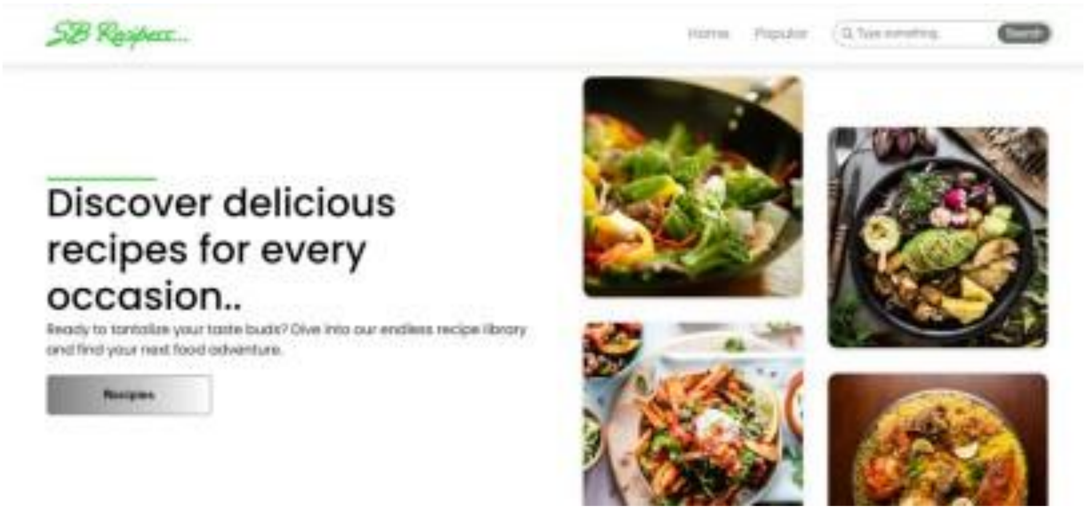
This React code manages fetching recipe data from an API and storing it within a state variable.

- It leverages the use State hook to establish a state variable named rec (which is initially empty). This variable acts as a container to hold the fetched recipe data.
- The use Effect hook comes into play to execute a side effect, in this instance, fetching data from an API endpoint. The hook takes a callback function (fetch Recipe 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 fetch Recipe 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 recipe 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 set Recipe function. This function, associated with the use State hook, allows for modification of the recipe state variable. By calling set Recipe (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.

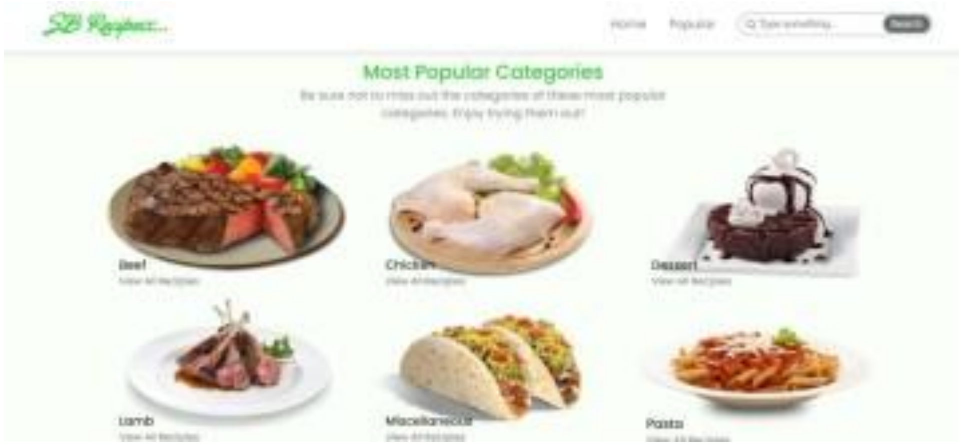


Discover delicious recipes for every occasion..

Ready to tantalize your taste buds? Dive into our endless recipe library and find your next food adventure.

➢ **Popular categories**

This component contains all the popular categories of recipes..

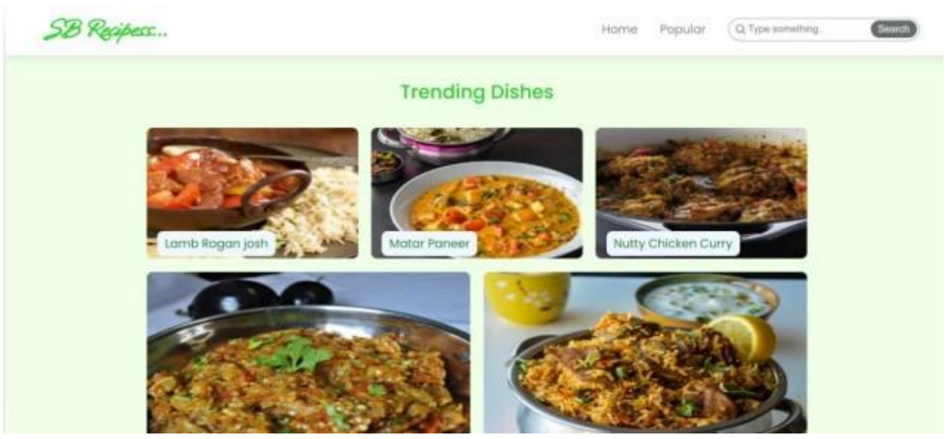


Most Popular Categories

Be sure not to miss out the categories of these most popular categories. Enjoy trying them out!

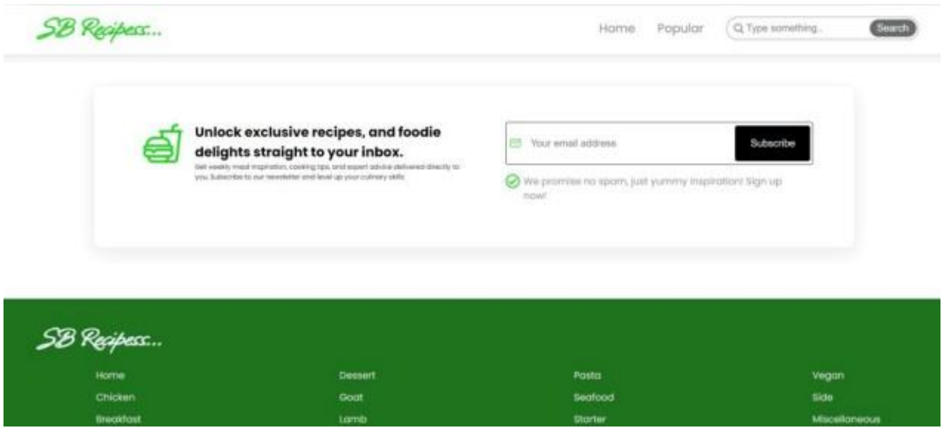
➢ **Trending Dishes**

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.



Unlock exclusive recipes, and foodie delights straight to your inbox.

Get weekly meal inspiration, cooking tips, and expert advice delivered directly to you. Subscribe to our newsletter and level up your culinary skills

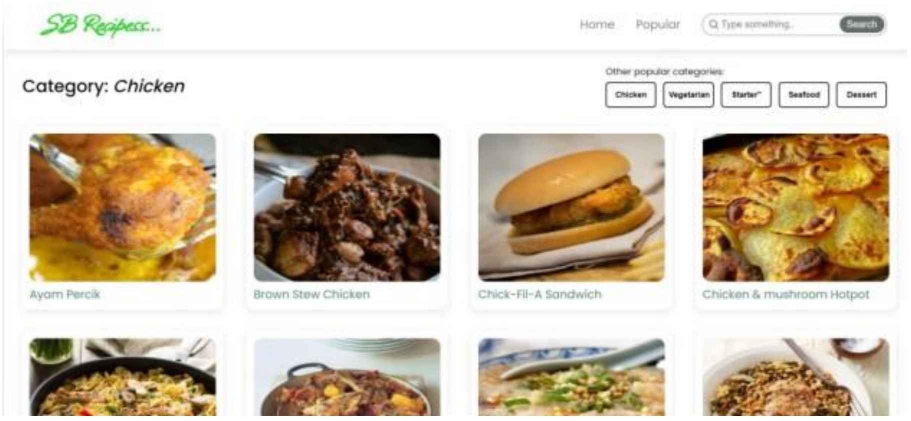
Your email address

Subscribe

We promise no spam, just yummy inspiration! Sign up now!

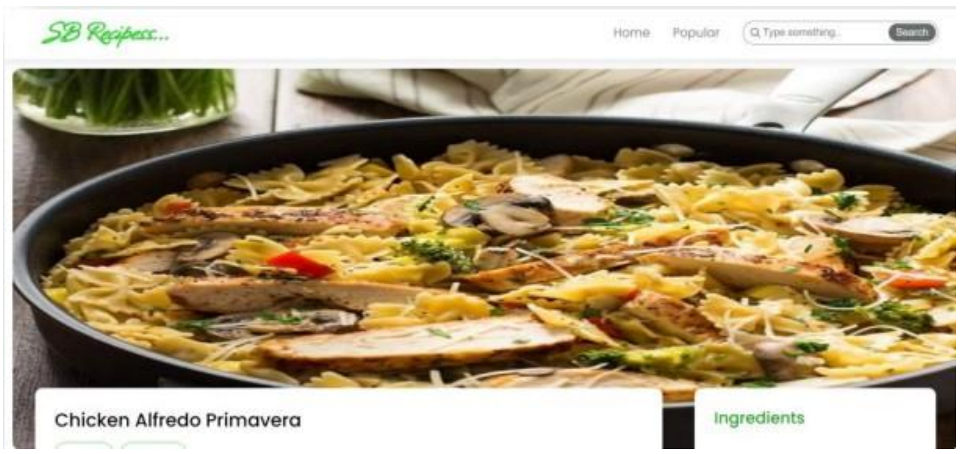
➢ **Category dishes page**

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



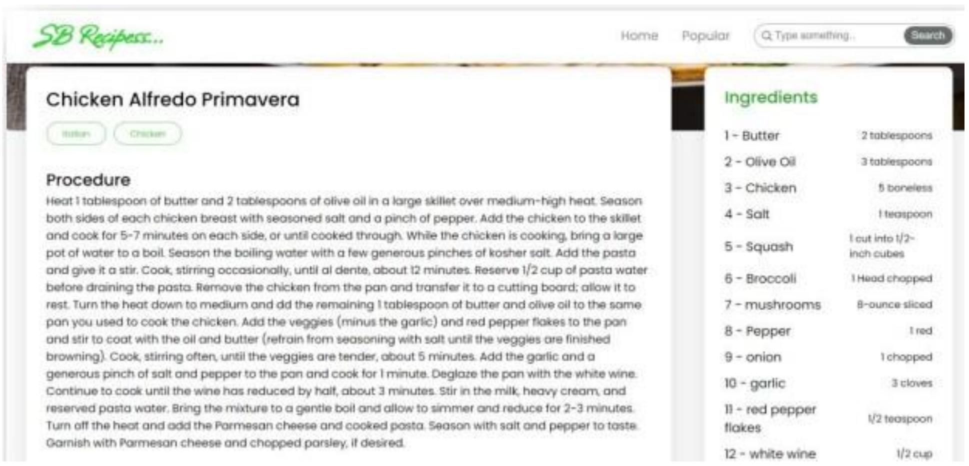
➢ **Recipe page**

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



Chicken Alfredo Primavera

Ingredients



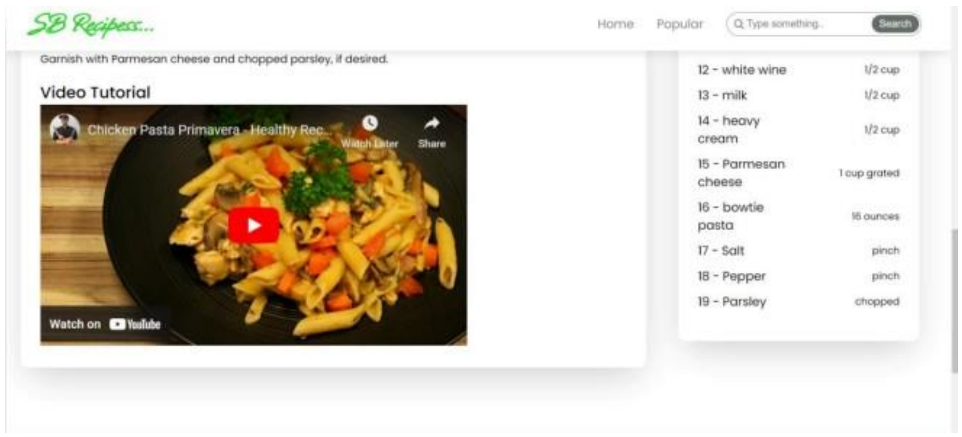
Chicken Alfredo Primavera

Procedure

Heat 1 tablespoon of butter and 2 tablespoons of olive oil in a large skillet over medium-high heat. Season both sides of each chicken breast with seasoned salt and a pinch of pepper. Add the chicken to the skillet and cook for 5-7 minutes on each side, or until cooked through. While the chicken is cooking, bring a large pot of water to a boil. Season the boiling water with a few generous pinches of kosher salt. Add the pasta and give it a stir. Cook, stirring occasionally, until al dente, about 12 minutes. Reserve 1/2 cup of pasta water before draining the pasta. Remove the chicken from the pan and transfer it to a cutting board; allow it to rest. Turn the heat down to medium and dd the remaining 1 tablespoon of butter and olive oil to the same pan you used to cook the chicken. Add the veggies (minus the garlic) and red pepper flakes to the pan and stir to coat with the oil and butter (refrain from seasoning with salt until the veggies are finished browning). Cook, stirring often, until the veggies are tender, about 5 minutes. Add the garlic and a generous pinch of salt and pepper to the pan and cook for 1 minute. Deglaze the pan with the white wine. Continue to cook until the wine has reduced by half, about 3 minutes. Stir in the milk, heavy cream, and reserved pasta water. Bring the mixture to a gentle boil and allow to simmer and reduce for 2-3 minutes. Turn off the heat and add the Parmesan cheese and cooked pasta. Season with salt and pepper to taste. Garnish with Parmesan cheese and chopped parsley, if desired.

Ingredients

1 - Butter	2 tablespoons
2 - Olive Oil	3 tablespoons
3 - Chicken	5 boneless
4 - Salt	1 teaspoon
5 - Squash	1 cut into 1/2-inch cubes
6 - Broccoli	1 Head chopped
7 - mushrooms	8-ounce sliced
8 - Pepper	1 red
9 - onion	1 chopped
10 - garlic	3 cloves
11 - red pepper flakes	1/2 teaspoon
12 - white wine	1/2 cup



Garnish with Parmesan cheese and chopped parsley, if desired.

Video Tutorial

12 - white wine	1/2 cup
13 - milk	1/2 cup
14 - heavy cream	1/2 cup
15 - Parmesan cheese	1 cup grated
16 - bowtie pasta	16 ounces
17 - Salt	pinch
18 - Pepper	pinch
19 - Parsley	chopped

Project demo link:
https://drive.google.com/file/d/1khMJkccySgKyqRaEZgCpgDACHi572Llj/view?usp=sharing

*** Happy coding!! ***