Frontend Development with React.js

Project Documentation

INTRODUCTION

Project Title:

CookBook - Your virtual kitchen Assistant

(React Application)

Team Members*:*

Rubini.V – **UI/UX Designer & Documentation**

(cs2201111058042@lngovernmentcollege.com)

Gopika . M – **Frontend Developer (React/Framework)**

(cs2201111058015@lngovernmentcollege.com)

Balaji . R – **Frontend Developer (Styling & Theming)**

(cs2201111058007@lngovernmentcollege.com)

Ajith Kumar . A – **Quality Assurance (QA) Tester**

(cs2201111058002@lngovernmentcollege.com)

Arjun. D – ***content Manager and Supporter***

(cs2201111058003@lngovernmentcollege.com)

PROJECT OVERVIEW

Purpose:

The purpose of the " CookBook - Virtual Kitchen Assistant (React application) " project is to create a digital platform that assists users with meal planning, cooking, and recipe management. The app acts as a virtual kitchen assistant by providing features like recipe suggestions, ingredient management, and step-by-step cooking instructions.

Goals of the Project:

1. **Helping users plan meals** based on their dietary preferences and available ingredients.
2. **Offering personalized recipe recommendations** based on user input.
3. **Managing ingredients** and tracking what is available in the kitchen.
4. **Providing detailed cooking instructions** to guide users through recipes.
5. **Enhancing user experience** with an intuitive and interactive interface built in React.

Features:

Here's a streamlined list of key **frontend features** and **functionalities** for my **Cookbook Project**:

✓ **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.

ARCHITECTURE

Component Structure:

1. App Component

- **Role**: The root component that holds everything together.

- **Responsibility**: Manages routing, global state (e.g., user login), and overall layout.

2. Navbar Component

- **Role**: Displays navigation links (Home, Recipes, Search).
- **Responsibility**: Allows users to navigate the app and search for recipes.

3. RecipeList Component

- **Role**: Displays a list of recipes.
- **Responsibility**: Fetches and shows a list of recipes.

4. RecipeCard Component

- **Role**: Shows a preview of a recipe (e.g., title, image).
- **Responsibility**: Renders recipe details in a compact form.

5. RecipeDetail Component

- **Role**: Displays detailed information for a specific recipe.
- **Responsibility**: Shows ingredients, instructions, and other details.

6. SearchResults Component

- **Role**: Displays recipes based on the user’s search query.
- **Responsibility**: Filters and shows recipes based on search.

7. Favorites Component

- **Role**: Shows a list of the user’s favorite recipes.
- **Responsibility**: Allows users to view and manage their favorites.

8. UserAuth Component

- **Role**: Manages user login and sign-up.
- **Responsibility**: Handles user authentication and session.

9. Footer Component

- **Role**: Displays footer content (e.g., copyright, links).
- **Responsibility**: Provides additional info at the bottom of the page.

The component structure focuses on how the individual component is structured with its logic (`Navbar.js`), styling (`Footer.css`), and test file (`App.test.js`).

State Management

For a cookbook project, state management is crucial for handling various components of the app, such as managing the list of recipes, ingredients, user preferences, and possibly even the authentication state. Two popular state management approaches are the **Context API** and **Redux**.

1. Context API:

The **Context API** is a built-in feature in React, and it's great for smaller applications or when you don't need to handle complex state management. It allows you to pass data through the component tree without manually passing props down at every level.

2. Redux:

Redux is a more advanced state management tool and is usually employed for large-scale applications where managing state becomes more complex. It works with actions, reducers, and stores to manage state globally across the application.

Routing:

For a **cookbook project**, routing plays an essential role in navigating between different pages, such as the home page, recipe details, user profile, or search results. If using a routing library like **React Router**, you can structure the application's routing in a way that allows for smooth transitions and a user-friendly experience.

Routing with React Router:

React Router is a popular routing library in React that enables you to handle navigation in your application.

SETUP INSTRUCTION

Pre-Requisites:

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

1. Node.js & npm:

- **Node.js** is the JavaScript runtime required to run React and many other development tools.

  Node.js (v14.x or higher recommended)

- **npm** (Node Package Manager) comes with Node.js and is used to install and manage the dependencies of your project.

  npm (comes with Node.js)

You can **download** and install Node.js from:

https://nodejs.org/

To check the installed version, run:

```
node -v
npm -v
```

2.React Router and React DOM:

If you're using **React Router** for navigation, install it as a dependency.

```
npm install react-router-dom
```

3. Development Tools (Optional):

For development and building the project, you might need these tools:

- **React Scripts**: If you're using Create React App, this package includes scripts for running and building the project.

```
npx create-react-app cookbook-app
```

- **ESLint**: For linting and ensuring code quality.

```
npm install eslint --save-dev
```

4. Version Control (Optional but Recommended):

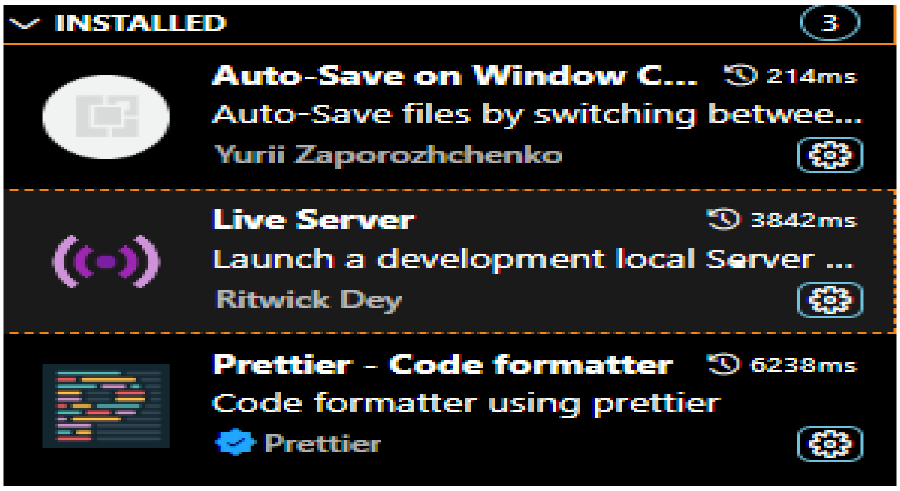
- **Git**: Version control is essential for managing code changes and collaboration.

  You can install **Git** from:

https://git-scm.com/

5. Other Tools Installed:

- **Prettier** (for code formatting)
- **Auto-save**
- **Live Server**



Auto-Save on Window C... 214ms
Auto-Save files by switching betwee...
Yurii Zaporozhchenko

Live Server 3842ms
Launch a development local Server ...
Ritwick Dey

Prettier - Code formatter 6238ms
Code formatter using prettier
Prettier

Summary of Key Dependencies:

1. **Node.js & npm** – For running the project.
2. **React & React DOM** – For building and routing the user interface.
3. **Redux** (if using) – For state management.
4. **Development Tools** – For building and running the app.
5. **Testing Libraries** – For testing.

Installation:

Here's a step-by-step guide to cloning a repository, installing dependencies, and configuring environment variables for documentation:

Step 1: Clone the Repository

1. Open your terminal or command prompt.
2. Navigate to the directory where you want to store the project.
3. Clone the repository by running the following command:

```
git clone <repository_url>
```

Replace `<repository_url>` with the actual URL of the repository (e.g., `https://github.com/username/repository.git`).

4. Change into the project directory:

```
cd <repository_name>
```

Step 2: Install Dependencies

Using npm(for Node.js)

1. If the project uses `npm`, ensure you have Node.js installed. You can check if Node.js is installed with:

```
node -v
```

2. Install the project dependencies:

```
npm install
```

Step 4: Run the Project

1. For Node.js projects, you can run the project using:

```
npm start
```

2. Check the documentation or README for specific instructions on how to run or build the project.

Step 5: Verify Installation

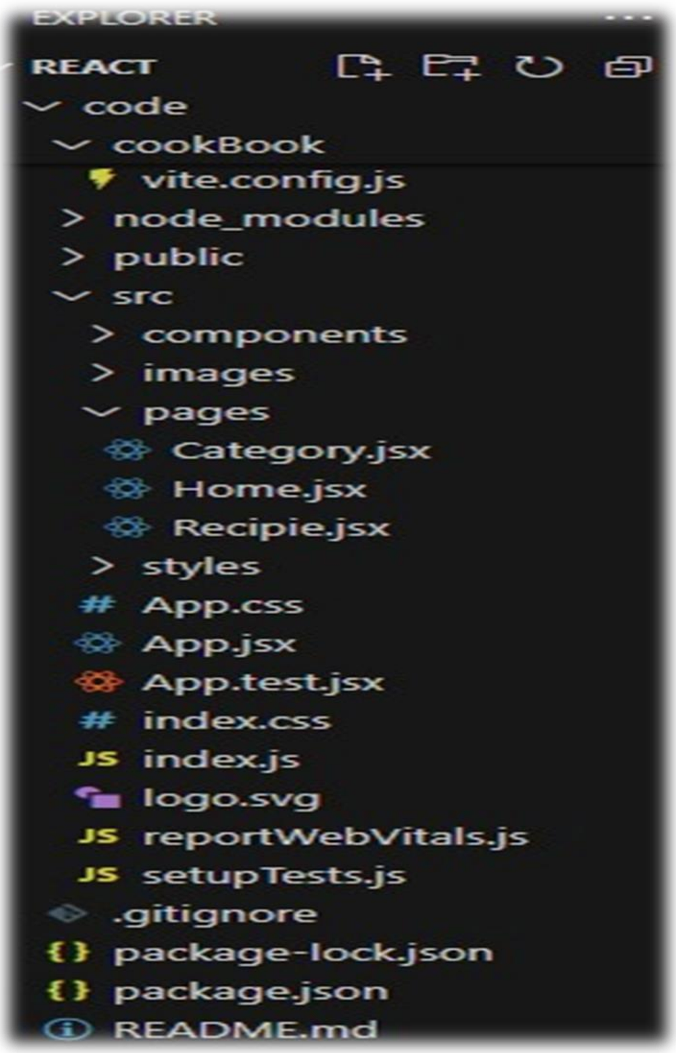
1. Open a browser or use a tool like `curl` to visit the local server

   (**e.g. ,** ***http://localhost:3000 or http://127.0.0.1:5000***) to verify that the project is running successfully.

FOLDER STRUCTURE

Client:

- **`src/`**: Contains the main application code.
    - **`components/`**: Reusable UI elements (e.g., `Button`, `Navbar`).
    - **`pages/`**: Page components for different routes (e.g., `HomePage`, `AboutPage`).
    - **`assets/`**: Static files (e.g., images, icons).
    - **`styles/`**: Global CSS or styled-components.
    - **`hooks/`**: Custom React hooks for reusable logic (e.g., `useAuth`, `useFetch`).
    - **`contexts/`**: React contexts for global state management.
    - **`services/`**: API service files (e.g., `api.js`).



```
EXPLORER                ...
REACT
∨ code
  ∨ cookBook
     vite.config.js
   > node_modules
   > public
   ∨ src
     > components
     > images
     ∨ pages
        Category.jsx
        Home.jsx
        Recipie.jsx
     > styles
     App.css
     App.jsx
     App.test.jsx
     index.css
     index.js
     logo.svg
     reportWebVitals.js
     setupTests.js
   .gitignore
   package-lock.json
   package.json
   README.md
```

Utlities:

- **Helper Functions**: Standalone functions for specific tasks (e.g., `formatDate`, `validateEmail`).
- **Utility Classes**: Classes for managing tasks (e.g., `ApiService`, `LocalStorage`).
- **Custom Hooks**: Reusable hooks for managing state or side effects (e.g., `useLocalStorage`, `useFetch`).

This structure ensures the application is organized, modular, and maintainable.

Running The Application

To start the frontend server locally, follow these steps:

1. **Navigate to the Client Directory**: Open your terminal and change to the `client` directory (or the directory where your React project is located):

```
cd client
```

2. **Install Dependencies** (if you haven't already): Before running the application, make sure all dependencies are installed:

```
npm install
```

3. **Start the Frontend Server**: To start the React development server locally, run the following command:

```
npm start
```

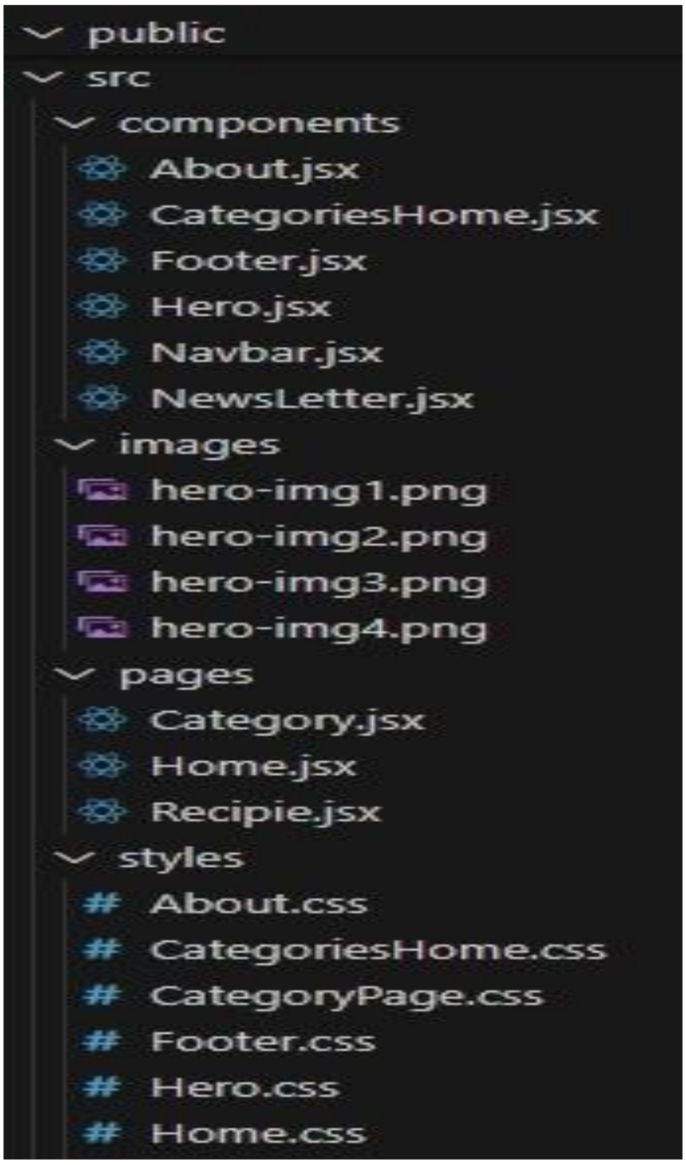
4. **Access the Application**: After running the command, the application should be available at `http://localhost:3000` in your web browser.

That's it! Your frontend server should now be running locally.

COMPONENT DOCUMENTATION

Key Components:

1. `Header`
   - **Purpose**: Displays the top navigation bar of the application.
2. **`Footer`**
   - **Purpose**: Displays the footer content at the bottom of the application.
3. **`HomePage`**
   - **Purpose**: The main landing page that displays featured content and navigation links.



```
public
src
  components
    About.jsx
    CategoriesHome.jsx
    Footer.jsx
    Hero.jsx
    Navbar.jsx
    NewsLetter.jsx
  images
    hero-img1.png
    hero-img2.png
    hero-img3.png
    hero-img4.png
  pages
    Category.jsx
    Home.jsx
    Recipie.jsx
  styles
    About.css
    CategoriesHome.css
    CategoryPage.css
    Footer.css
    Hero.css
    Home.css
```

Reusable Components

1. `Button`
    - **Purpose**: A reusable button component used across various pages and components.
2. `InputField`
    - **Purpose**: A reusable input field for forms and user inputs.
3. **Card**
    - **Purpose**: Displays content in a card layout, used for showcasing items such as products or articles.

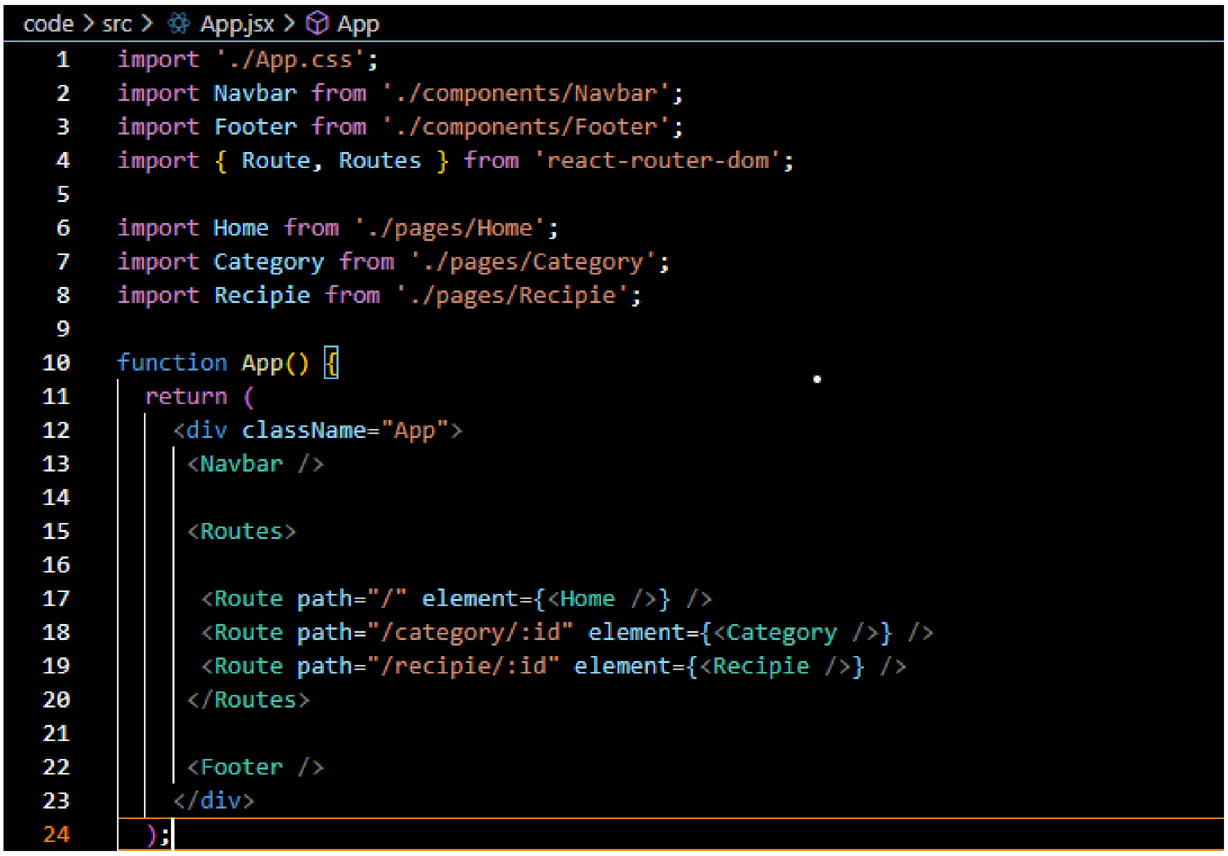
This section provides a simple overview of the major and reusable components, their purposes, and the props they receive for flexibility and customization.

State Management

Global State:

Purpose: Global state management is used to store and manage data that needs to be accessed or modified by multiple components across the application.

Example: App.jsx


code > src > App.jsx > App

```jsx
import './App.css';
import Navbar from './components/Navbar';
import Footer from './components/Footer';
import { Route, Routes } from 'react-router-dom';

import Home from './pages/Home';
import Category from './pages/Category';
import Recipie from './pages/Recipie';

function App() {
  return (
    <div className="App">
     <Navbar />

     <Routes>

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

     <Footer />
    </div>
  );
```

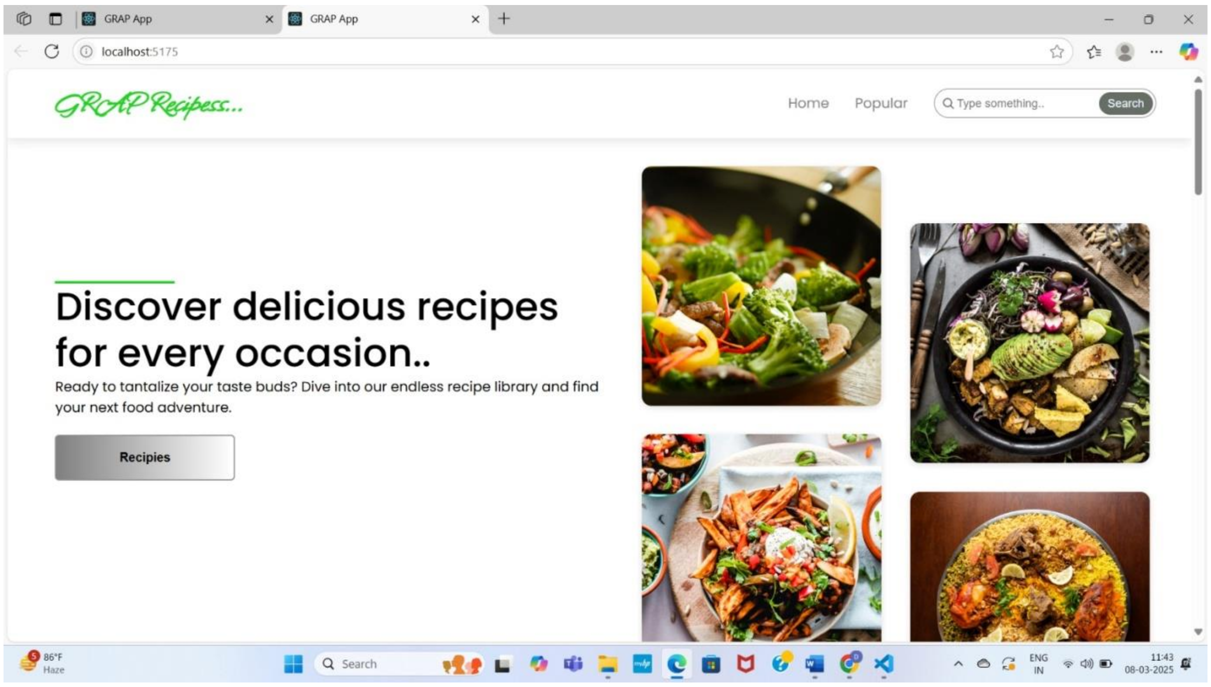
Local State:

Purpose: Local state is used to store data or control behavior within a specific component (i.e., data that does not need to be shared across the entire application).

USER INTERFACE

➢ Hero components

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

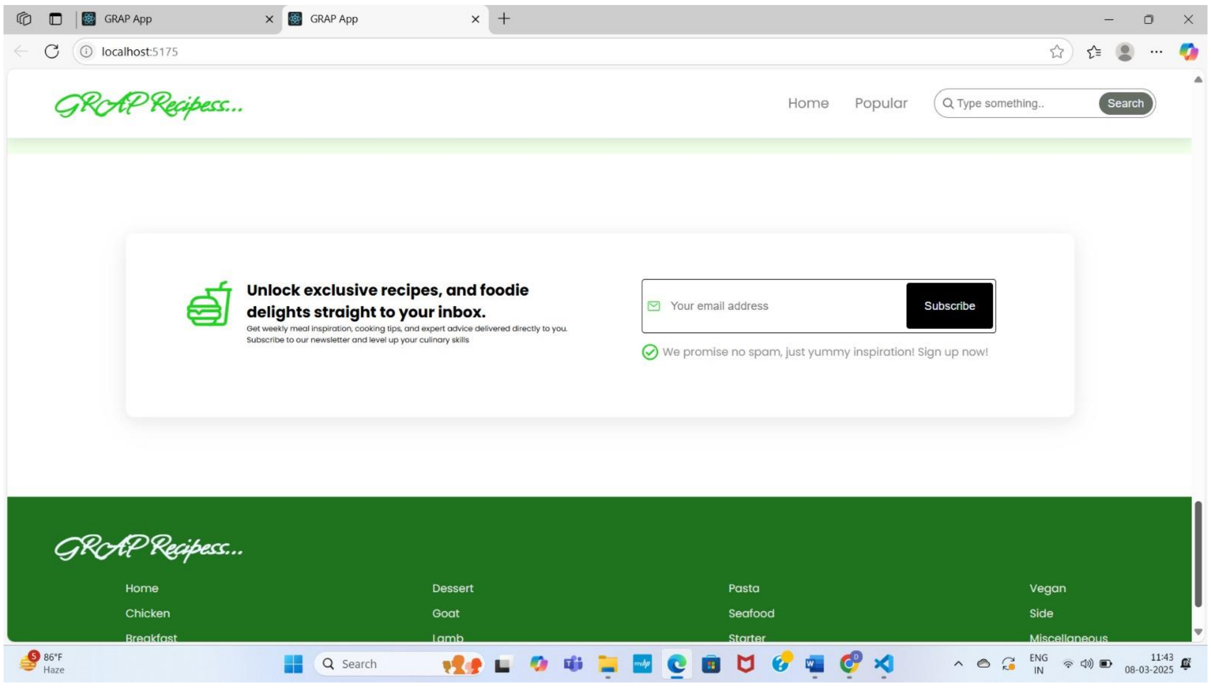


Discover delicious recipes for every occasion..

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

- ## News Letter

The news letter componentprovides an email input to subscribefor the recipe newsletter.



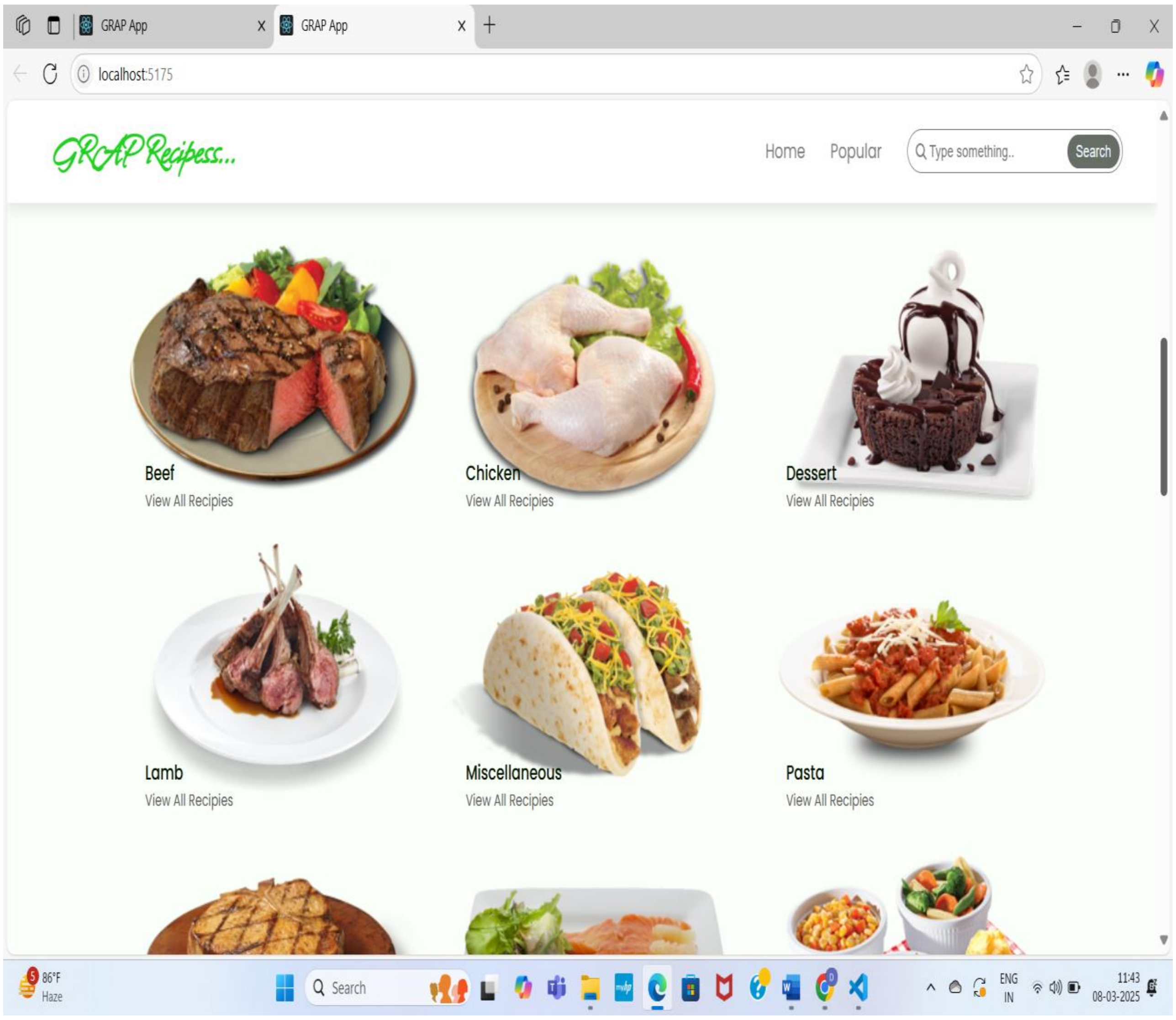
Unlock exclusive recipes, and foodie delights straight to your inbox.

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

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

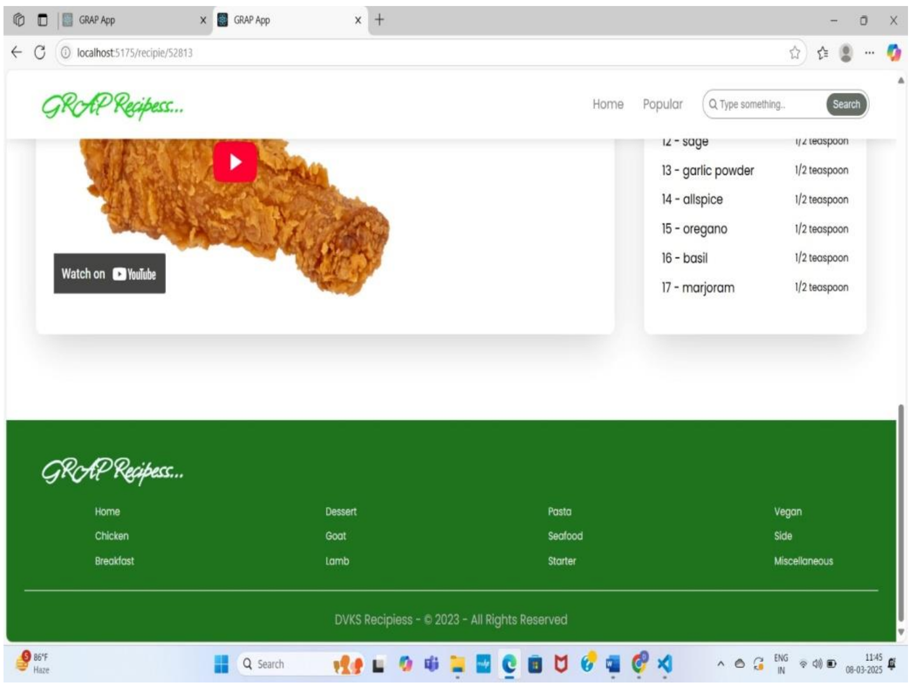
- ## Popular Categories

This component contains all popular caegories of recipes.



➢ Recipe page

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

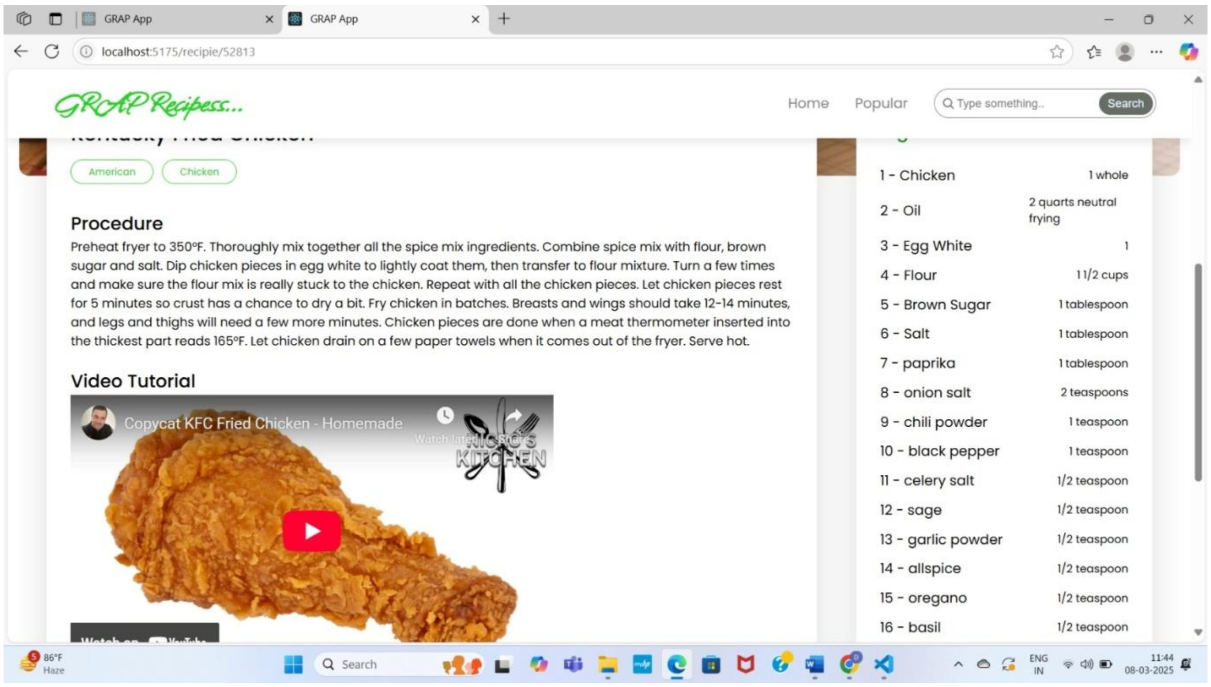


12 - sage	1/2 teaspoon
13 - garlic powder	1/2 teaspoon
14 - allspice	1/2 teaspoon
15 - oregano	1/2 teaspoon
16 - basil	1/2 teaspoon
17 - marjoram	1/2 teaspoon

GRAP Recipess...

Home	Dessert	Pasta	Vegan
Chicken	Goat	Seafood	Side
Breakfast	Lamb	Starter	Miscellaneous

DVKS Recipiess - © 2023 - All Rights Reserved



Procedure

Preheat fryer to 350ºF. Thoroughly mix together all the spice mix ingredients. Combine spice mix with flour, brown sugar and salt. Dip chicken pieces in egg white to lightly coat them, then transfer to flour mixture. Turn a few times and make sure the flour mix is really stuck to the chicken. Repeat with all the chicken pieces. Let chicken pieces rest for 5 minutes so crust has a chance to dry a bit. Fry chicken in batches. Breasts and wings should take 12-14 minutes, and legs and thighs will need a few more minutes. Chicken pieces are done when a meat thermometer inserted into the thickest part reads 165ºF. Let chicken drain on a few paper towels when it comes out of the fryer. Serve hot.

Video Tutorial

1 - Chicken	1 whole
2 - Oil	2 quarts neutral frying
3 - Egg White	1
4 - Flour	1 1/2 cups
5 - Brown Sugar	1 tablespoon
6 - Salt	1 tablespoon
7 - paprika	1 tablespoon
8 - onion salt	2 teaspoons
9 - chili powder	1 teaspoon
10 - black pepper	1 teaspoon
11 - celery salt	1/2 teaspoon
12 - sage	1/2 teaspoon
13 - garlic powder	1/2 teaspoon
14 - allspice	1/2 teaspoon
15 - oregano	1/2 teaspoon
16 - basil	1/2 teaspoon

Here’s how to showcase UI features:

1. **Home Page**
   - **Screenshot**: Display the main layout with featured items.
2. **Login Form**

    - **Screenshot**: Show the form with email, password fields, and the submit button.
3. **Interactive Button**
    - **GIF**: Show the button being clicked to trigger an action (e.g., opening a modal).
4. **Modal Dialog**
    - **GIF**: Show a modal opening and closing when interacted with.
5. **Product Page**
    - **Screenshot**: Display the product details with images, price, and description.
6. **Form Validation**
    - **GIF**: Show how error messages appear on invalid input.

STYLING

- **CSS Frameworks/Libraries**: We use **[CSS Frameworks/Libraries]**

  (e.g., Bootstrap, Tailwind CSS, Styled-Components) for consistent styling and layout, along with **[pre-processors]** like Sass for better maintainability.

- **Theming**: Custom theming is implemented using **[Theme Provider]** or CSS variables to allow dynamic switching of themes.

TESTING

- **Testing Strategy**: We use **Jest** and **React Testing Library** for unit, integration, and end-to-end testing of components to ensure reliability.

- **Code Coverage**: Tools like **Jest's coverage report** are used to ensure sufficient test coverage across the codebase.

FUTURE ENHANCEMENTS

- **New Components**: Add components like **Recipe Search**, **Recipe Filter**, and **Shopping List** for better user interaction.
- **Animations**: Implement smooth **transitions** when switching between recipes or adding to the shopping list.

- **Enhanced Styling**: Improve the design with **responsive layouts** for mobile devices, and add a **dark mode** option for user preference.
- **Recipe Sharing**: Enable **social media sharing** and allow users to **rate and review** recipes.
- **Personalization**: Add a **favorites section** for users to save their favorite recipes.

Known Issues:

1. Too many or too few recipes.
2. Untested recipes.
3. Bad design or layout.
4. Low-quality photos.
5. Hard-to-find ingredients.
6. Not right for the target audience.
7. Cultural mistakes.
8. Technical problems (for digital books).
9. Typos or errors.
10. Copyright issues.
11. Scaling recipes wrong.
12. High costs to produce.

Project Demo Link:

https://drive.google.com/file/d/1ABIRWsMBuGDq8Sd9n-au7x4gjDfWzNdw/view?usp=sharing