CookBook: Your Virtual Kitchen Assistant

1. Introduction

1.1 Project Title:

CookBook: Your Virtual Kitchen Assistant

1.2 Team Details:

Team ID: SWTID1741530027146261

Team Leader	**PAVITHRA R**	**pavthrav2020@gmail.com**

Team Member 1	**PAVITHRA R**	**pavthrav2020@gmail.com**
Team Member 2	**RENISREE R**	**renisree5@gmail.com**
Team Member 3	**DIVYA S**	**divyasanthosh558@gmail.com**
Team Member 4	**JEEVA S**	**jeevajeeva472004@gmail.com**
Team Member 5	**BHAVATHARANI DURGA A R**	**bhavatharanidurgaar@gmail.com**

2. Project Overview

Purpose:

Provide a concise yet comprehensive description of the project, explaining the problem it aims to solve and the intended audience. Mention any specific use cases or practical applications.

Features:

List and briefly explain the core features of the frontend application, such as:

- **User Authentication:** Login and signup functionality with JWT authentication.
- **Responsive Design:** Mobile-first approach with adaptive UI elements.
- **Dynamic UI Elements:** Interactive components like modals, forms, and real-time data updates.

- **API Integration:** Fetch and display data from backend services.
- **Dark Mode Support:** Theming options for better user experience.
- **Accessibility Features:** ARIA attributes, keyboard navigation, and screen reader support.

3. Architecture

Component Structure:

Provide an overview of how the React components are structured. Example:

- **App.js:** Main application wrapper.
- **Components Folder:** Reusable UI components such as buttons, modals, and forms.
- **Pages Folder:** Contains different views such as Home, Dashboard, Profile.
- **Hooks Folder:** Custom hooks to manage state and side effects.

State Management:

Explain how the application manages state across components. If using **Redux**, **Context API**, or third-party libraries like **Zustand**, describe how the state flows.

Routing:

Mention the routing library used (**React Router** or others) and how navigation is structured:

- Define public and private routes.
- Use nested routes for efficient navigation.

4. Setup Instructions

Prerequisites:

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.

- **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 the steps below:

- **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:**

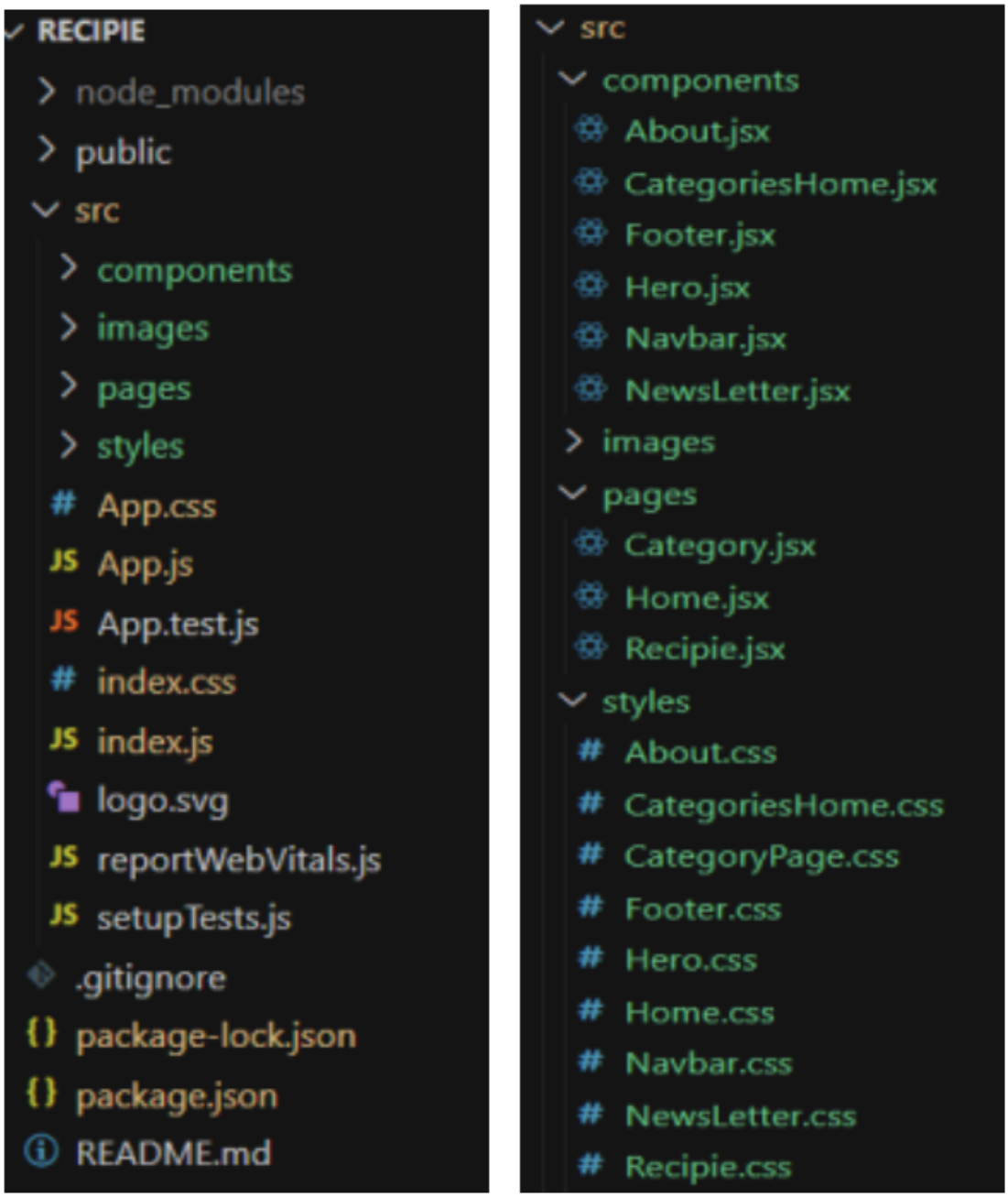
Navigate into the cloned repository directory and install libraries:
cd recipe-app-react

  - 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.

5. 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

Start the frontend server locally

npm start # Starts the development server

The application will be available at **http://localhost:3000**.

7. Component Documentation

Key Components:

- **Navbar.js:** Handles site navigation.
- **Footer.js:** Displays the footer.
- **Dashboard.js:** Main user dashboard with real-time data.
- **Profile.js:** User profile management.

Reusable Components:

Explain commonly used UI elements:

- **Button.js:** Customizable button component.
- **Modal.js:** Generic modal component for pop-ups.
- **Loader.js:** Displays loading animations.

8. State Management

Global State:

- If using **Redux**, explain actions, reducers, and store setup.
- If using **Context API**, detail context providers and consumers.

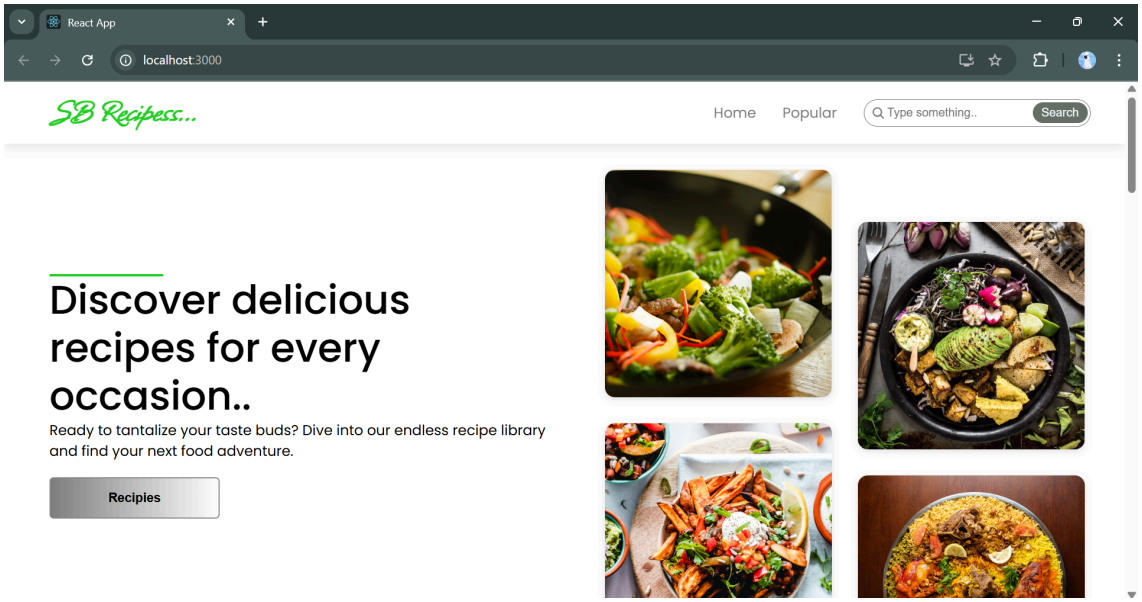
Local State:

- Explain state within individual components using useState and useEffect.

9. User Interface

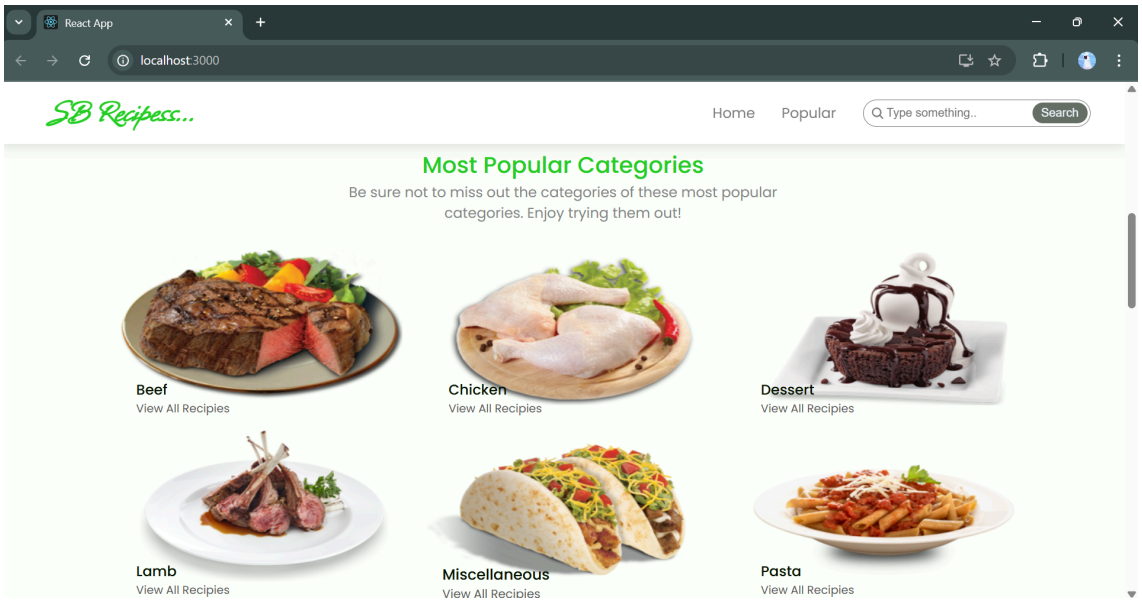
➢ Hero components :

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



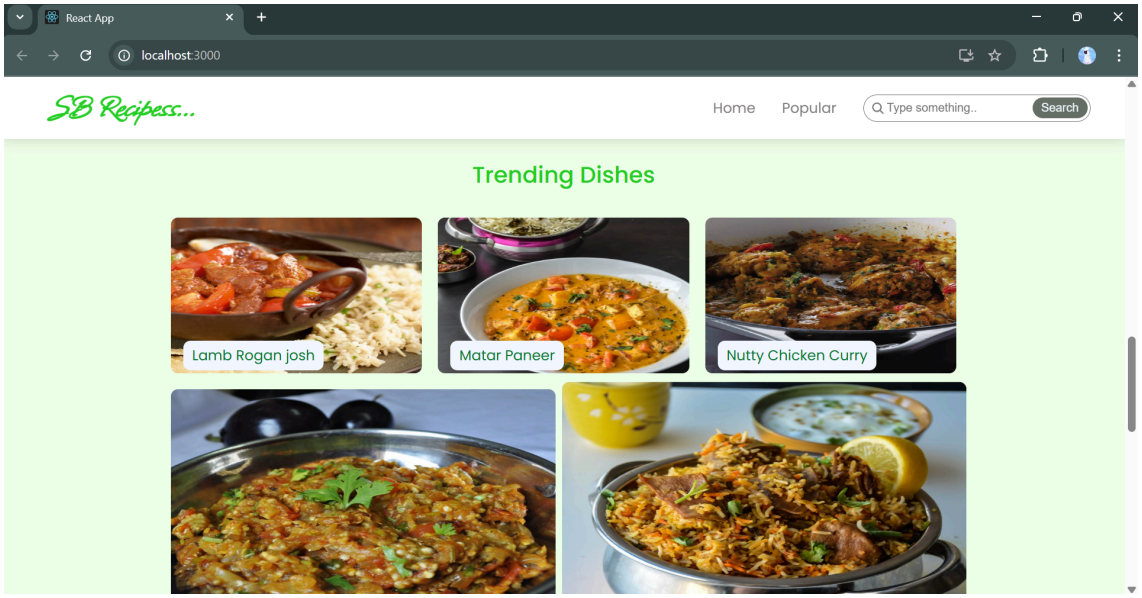
➢ Popular categories:

This component contains all the popular categories of recipes.



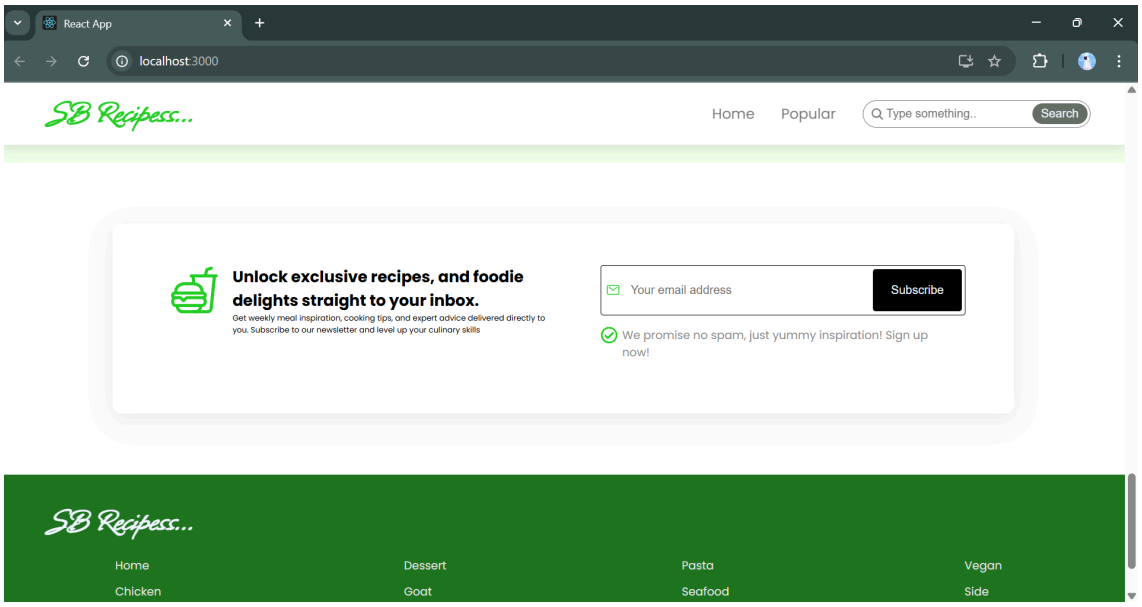
➢ Trending Dishes:

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



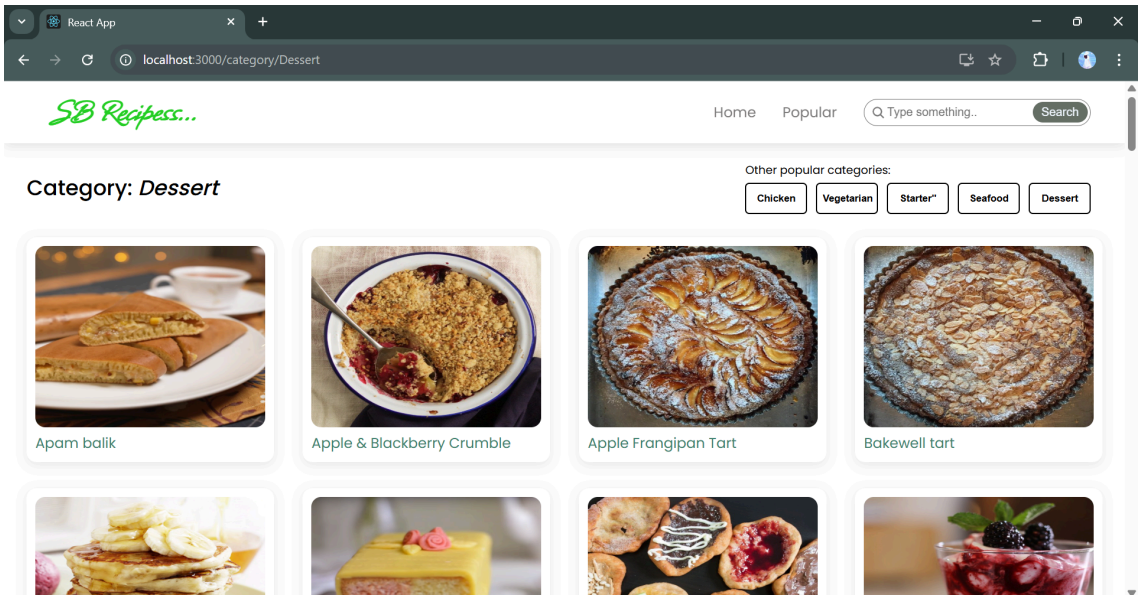
➢ News Letter :

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



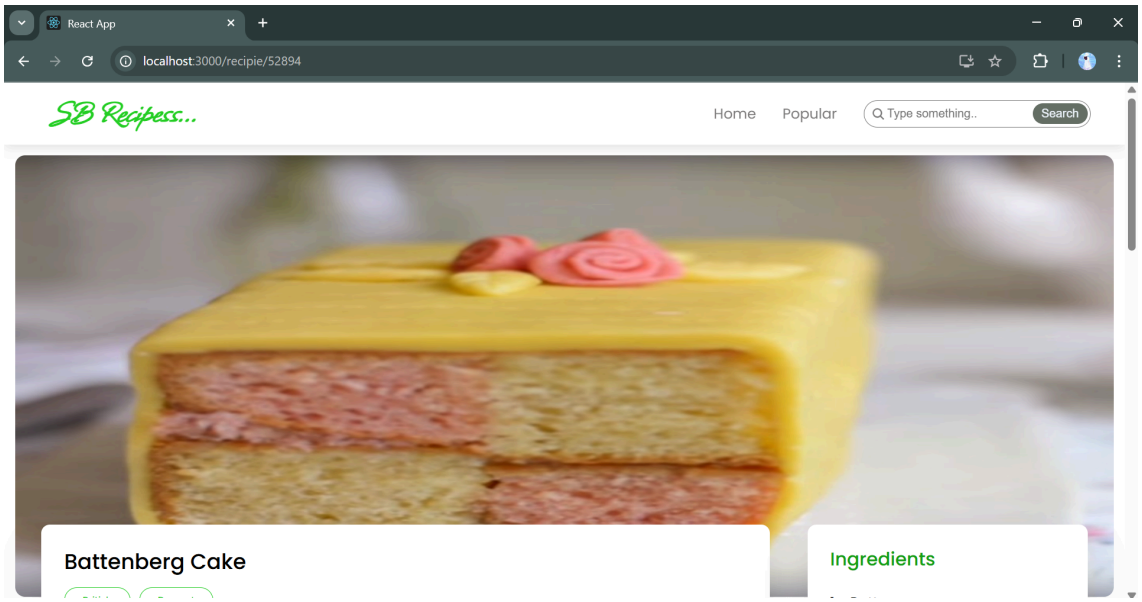
➢ Category dishes page :

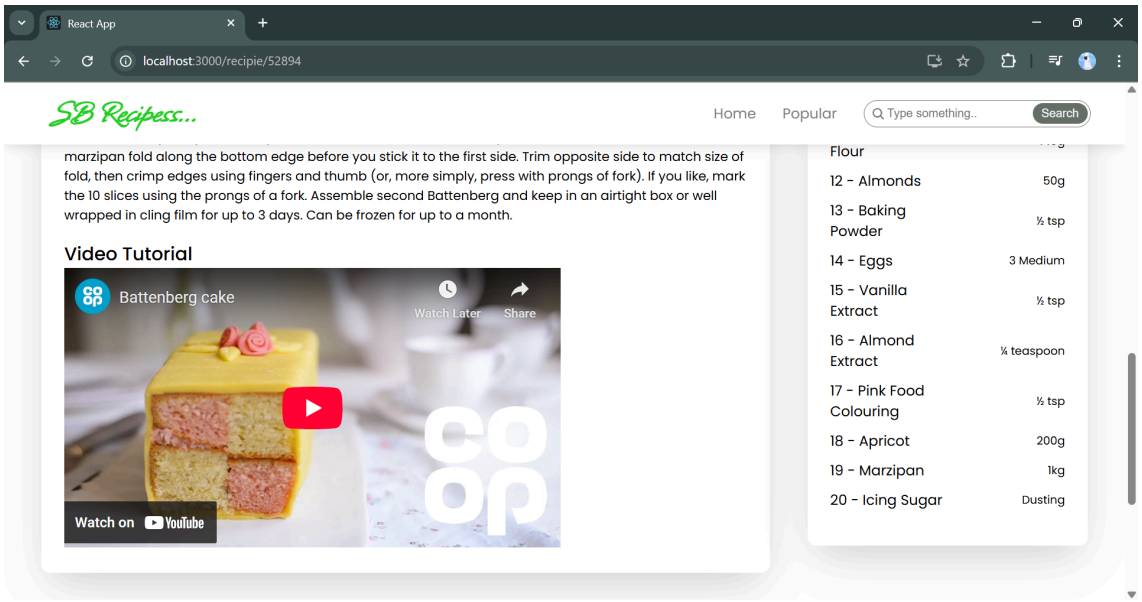
The category page contains the list of dishes under a certain category.



➢ Recipe page :

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





10. Styling

CSS Frameworks/Libraries:

Mention styling choices such as:

- **Tailwind CSS** for utility-based styling.
- **Material-UI** for prebuilt UI components.

- **Styled-Components** for CSS-in-JS approach.

Theming:

Describe if dark mode, custom themes, or design tokens are implemented.

11. Testing

Testing ensures application reliability and smooth feature integration. It involves three main levels:

1. Unit Testing

Tests individual components or functions in isolation to verify correctness.

- **Focus:** Small, isolated units like functions, React components, or hooks.
- **Tools:** Jest (default React testing framework), React Testing Library.

2. Integration Testing

Ensures different components or services interact correctly.

- **Focus:** API calls, state updates, component communication.
- **Tools:** React Testing Library, Mock Service Worker (MSW).

3. End-to-End (E2E) Testing

Simulates real user interactions and tests the full application flow.

- **Focus:** UI, APIs, and database interactions.
- **Tools:** Cypress, Playwright.

12. Demo Video

- A link to a **live demo** :- https://drive.google.com/file/d/1Fu2hVL_bMWt_Hd6uw3tH-3Krlt8NLyjY/view?usp=sharing

13. Known Issues

Document any **current bugs or limitations**, such as:

- UI responsiveness issues.
- API endpoint failures.
- Browser compatibility concerns.
- Youtube video being in private

14. Future Enhancements

Outline **possible improvements**, like:

- Implementing real-time updates using WebSockets.
- Adding internationalization support.
- Improving performance through lazy loading and code splitting.
- **Optimizing Accessibility:** Enhancing support for screen readers and keyboard navigation.
- **Performance Improvements:** Utilizing memoization, lazy loading, and efficient rendering practices.