Frontend Development with React.js

Project Documentation

1. Introduction:

CookBook: Your Virtual Kitchen Assistant

Submitted By

S. Srivari(222212212),
S. Sudhakar(222212213),
T. Suriya(222212214),
R. Vasanthan(222212215),
R.Vinoth Kumar(222212216).

2. Project Overview:

Purpose:

The purpose of the Cookbook project is to create a digital platform where users can easily search, browse, and save recipes. The platform aims to provide users with a simple and interactive way to discover new dishes, customize recipes to their liking, and build a personalized recipe collection. The goal is to make cooking accessible, enjoyable, and efficient by offering a variety of recipes that cater to different dietary preferences and skill levels.

Features:

1. **Recipe Search & Filters:**
    - Users can search for recipes by ingredients, cuisine type, dietary restrictions (e.g., vegan, gluten-free), meal type (e.g., breakfast, lunch, dinner), or preparation time.
2. **Recipe Detail Pages:**
    - Each recipe will have a detailed page with ingredients, step-by-step instructions, nutritional information, and photos.
3. **Shopping List Generation:**
    - Users can automatically generate a shopping list based on the ingredients of the selected recipe.
4. **Meal Planning:**
    - Users can plan meals by adding recipes to a weekly meal planner, which helps them organize meals and create shopping lists.
5. **Interactive Cooking Timer:**

- A built-in timer helps users follow cooking steps and ensures they cook each part of the recipe at the right time.

6. **Responsive Design:**
    - The frontend will be responsive and mobile-friendly, allowing users to easily access and navigate the site on any device.

7. **Social Sharing:**
   Users can share recipes on social media or with friends, helping to create a community around cooking.

3. Architecture:

☐ **State Management**

For managing the application state, React Context API will be used, with possible use of useReducer or useState for more granular control within components. Here's how it will work:

➢ Global State via Context API:
- ❖ A UserContext will store user data such as authentication status, saved recipes, and profile settings.
- ❖ A RecipeContext will manage the list of recipes fetched from the API and any filters applied to it.
- ❖ A MealPlannerContext will manage the meal planning and shopping list states.

Each context will wrap the components that need access to specific state, and the context providers will be placed higher up in the component tree (in `App.js`).

➢ useReducer for Complex State:

- ❖ For complex actions like handling recipe searches, saving recipes, or managing meal planner updates, useReducer can be used to manage state transitions in a more predictable manner.
- ❖ Reducers will update the state based on actions (e.g., add/remove recipe, set filter criteria, etc.) and will allow for better debuggability and control.

☐ **Basic Routing Setup**

- The `App.js` component will be the entry point for routing, using the `BrowserRouter` from React Router to wrap the entire application.
- A common structure will use `<Route>` components to map URLs to specific components (e.g., RecipeDetail, RecipeList).

☐ **Dynamic Routing**:

- Routes such as `/recipes/:id` use dynamic parameters to fetch recipe details based on the recipe ID.

4. Setup Instructions

Here are the setup 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

5.Folder Structure

```
cookbook
├── /public
│   ├── index.html
│   └── /assets          # Static files like images, fonts, etc.
├── /src
│   ├── /components      # Reusable UI components
│   │   ├── Header.js
│   │   ├── RecipeCard.js
│   │   ├── RecipeList.js
│   │   ├── SearchFilter.js
│   │   ├── LoginSignup.js
│   │   ├── RecipeDetail.js
│   │   ├── MealPlanner.js
│   │   ├── ShoppingList.js
│   │   └── UserProfile.js
│   ├── /contexts        # Contexts for state management
│   │   ├── UserContext.js
│   │   ├── RecipeContext.js
│   │   └── MealPlannerContext.js
│   ├── /hooks           # Custom hooks (optional)
│   │   ├── useAuth.js
│   │   ├── useRecipes.js
│   │   └── useMealPlanner.js
│   ├── /pages           # Pages representing specific routes
│   │   ├── Home.js
│   │   ├── RecipeSearch.js
│   │   ├── RecipeDetailPage.js
│   │   ├── MealPlannerPage.js
│   │   ├── UserProfilePage.js
│   │   └── LoginSignupPage.js
│   ├── /services        # API calls or data-fetching utilities
│   │   ├── api.js
│   │   └── recipeService.js
│   ├── /styles          # Global styles (CSS, SCSS, or styled-components)
│   │   ├── global.css
│   │   └── theme.css
│   ├── App.js           # Main app component
│   ├── index.js         # Entry point for React app
│   ├── App.css          # Styles specific to App component
│   └── /utils           # Utility functions (e.g., helper functions)
│       └── formatter.js
├── .gitignore
├── package.json
├── README.md
└── /node_modules
```

6.Running the Application

1. Install Dependencies:

First, make sure you've installed all the necessary dependencies for the project. Open a terminal and navigate to the root directory of your project (where `package.json` is located), then run the following command:

```
npm install
```

This will install all the required packages listed in the `package.json` file.

2. Start the Frontend Server:

Once the dependencies are installed, you can start the development server with the following command:

```
npm start
```

This command will:

- Launch the React development server locally.
- Open the app in your default browser (usually at `http://localhost:3000`).

The development server watches for file changes and automatically reloads the app when changes are made, making it easier to work on your project.

3. Accessing the App:

After running `npm start`, you can access the application by navigating to:

```
http://localhost:3000
```

4. Stopping the Server:

To stop the server, press `Ctrl + C` in the terminal where the server is running.

7. Component Documentation

1. **App.js**
   Purpose:
   - The root component that wraps the entire application, initializing routing and global state management via context providers.
2. **Props:**
   - No direct props are passed to `App.js`, but it houses the global providers (like `UserContext` and `RecipeContext`) for managing state across the app.
3. **Header.js**
   Purpose:

- The navigation bar that allows users to access different sections of the app, such as the home page, recipe search, meal planner, and login/logout functionality.

4. **Props:**
    - `user`: Current user data (from context) to display personalized content, such as the user's name and login/logout button.
5. **RecipeCard.js**
   Purpose:
    - A card component that displays a preview of a recipe, including an image, name, and brief description.
6. **Props:**
    - `recipe`: Object containing recipe data like `id`, `name`, `image`, `description`, etc.
    - `onClick`: A function that handles click events to navigate to the recipe details page.
7. **RecipeList.js**
   Purpose:
    - Displays a list of recipes based on user search or category selection.
8. **Props:**
    - `recipes`: An array of recipe objects that will be mapped to `RecipeCard` components.
    - `filters`: Any filters or search terms applied to the recipe list.
9. **RecipeDetail.js**
   Purpose:
    - Displays the detailed information of a single recipe including ingredients, preparation steps, and cooking tips.
10. **Props:**
    - `recipe`: Object containing detailed recipe data like `name`, `ingredients`, `instructions`, etc.
    - `onSaveRecipe`: Function for saving the recipe to the user's profile.
11. **SearchFilter.js**
   Purpose:
    - A set of input fields and dropdowns that allow users to filter recipes by various criteria like ingredients, meal type, dietary restrictions, etc.
12. **Props:**
    - `onFilterChange`: A function that is triggered when a user changes any filter criteria.
    - `filters`: Object that contains the current filter values.

13. **LoginSignup.js**
    Purpose:
    - Form component for logging in or signing up a user.
14. **Props:**
    - `onLogin`: Function to handle user login.
    - `onSignup`: Function to handle user registration.
15. **MealPlanner.js**
    Purpose:
    - Allows users to plan their meals for the week by adding recipes to a calendar.
16. **Props:**
    - `plannedMeals`: List of recipes selected for the weekly meal plan.
    - `onMealPlanChange`: Function to add/remove recipes from the meal plan.
17. **UserProfile.js**
    Purpose:
    - Displays the logged-in user's saved recipes, meal plan, and account settings.
18. **Props:**
    - `userData`: Contains the logged-in user's profile information and saved recipes.

19. **ShoppingList.js**
    Purpose:
    - Displays a list of ingredients needed for recipes in the user's meal plan or saved recipes.
20. **Props:**
    - `ingredients`: List of ingredients required for the selected recipes.
    - `onRemoveIngredient`: Function to remove ingredients from the shopping list.

8. Reusable Components

1. **Configuration/Customization:**
    - This component is highly reusable throughout the app. It can be used in `RecipeList.js`, `MealPlanner.js`, `UserProfile.js`, etc. It adapts based on the `recipe` prop passed to it, ensuring consistency and reducing the need for redundant components.
2. **SearchFilter.js**
    - **Props:**
        - `onFilterChange`: Function to update the filter state when the user modifies a filter.

        - `filters`: Object containing current filter values for things like ingredients, cuisine, and dietary restrictions.
3. **Configuration/Customization:**
    - Can be reused in various parts of the app that require filtering, such as in `RecipeList.js` for searching recipes or in `MealPlanner.js` for meal selection.

9. State Management

Global State

Global state management is handled using **React Context API** to share data across various components.

1. **UserContext**
    - Manages user authentication, profile data, and saved recipes.
    - Provides global access to `user`, `isAuthenticated`, `savedRecipes`, etc., across the app.
2. **Flow:**
    - Once the user logs in, `UserContext` stores their data, allowing components like `Header.js`, `UserProfile.js`, and `RecipeDetail.js` to access the user's information (e.g., saved recipes).
3. **RecipeContext**
    - Handles the list of recipes, including search results and applied filters.
    - Provides global access to the `recipes` and `filters` data.
4. **Flow:**
    - Components like `RecipeList.js` and `SearchFilter.js` interact with `RecipeContext` to display the filtered list of recipes and handle dynamic search.
5. **MealPlannerContext**
    - Manages the meal plan for the user and the shopping list.
    - Provides global access to the user's planned meals and the shopping list.
6. **Flow:**
    - Components like `MealPlanner.js` and `ShoppingList.js` interact with this context to plan meals and generate the corresponding shopping list.

Local State

Local state is handled within individual components using **React's useState** or **useReducer** (for more complex state).

1. **useState**
    - Used in components like `SearchFilter.js` to handle the state of individual filters (e.g., ingredient search, dietary restrictions) before updating the global context.
2. **useReducer**
    - Used in `RecipeContext` for more complex state management, such as handling recipe list filtering or pagination.

Example:

javascript

```
const recipeReducer = (state, action) => {
  switch (action.type) {
    case 'SET_FILTERS':
      return { ...state, filters: action.payload };
    case 'SET_RECIPES':
      return { ...state, recipes: action.payload };
    default:
      return state;
  }
};
```

10. User Interface

For the user interface, you can showcase several UI elements, like:

- **Home Page** with a grid of recipe cards.
- **Recipe Search Page** with a search bar and filter options.
- **Recipe Detail Page** showing detailed information about a single recipe.
- **Meal Planner Page** with an interactive calendar or list for meal planning.
- **Login/Signup Forms** with authentication functionality.

As for **screenshots or GIFs**, I can't generate them directly here, but you can easily capture screenshots or record GIFs using tools like **Loom**, **Snipping Tool**, or **Giphy Capture**.

Styling

CSS Frameworks/Libraries

1. **CSS Pre-Processor (Sass)**
    - If you're using Sass for styling, you can create `.scss` files for modular styles.

Example structure:
scss

```
// _variables.scss

$primary-color: #3498db;

// _mixins.scss

@mixin flex-center {

  display: flex;

  justify-content: center;

  align-items: center;}
```

2.Styled-Components

You may also use **styled-components** for component-level styling. This allows for styling components directly within JavaScript, making the styles co-located with the components.

Example:
jsx

```
import styled from 'styled-components';

const Button = styled.button`

  background-color: ${props => props.primary ? '#3498db' :
'#2ecc71'};

  color: white;

  padding: 10px 20px;

  border: none;

  border-radius: 5px; cursor: pointer;`;
```

3.**Theming**

- The app may include a **light/dark theme** or custom design systems. If you're using **styled-components**, you can implement a theme provider for easy management of colors and global styles.

Example of a theme with `styled-components`:

javascript

```
const theme = {light: {

    background: '#fff',color: '#333', }, dark:

{background: '#333',

    color: '#fff', },};

export default theme
```

11.Testing

Testing is crucial for ensuring the reliability, maintainability, and correctness of your application. In the Cookbook project, the testing strategy includes unit testing, integration testing, and end-to-end testing to verify individual components, interactions, and overall system behavior.

1. Unit Testing

Unit testing focuses on testing individual components in isolation to ensure that each function, hook, or component behaves as expected. We will use Jest and React Testing Library for unit testing.

Tools:

- Jest: The testing framework that provides utilities to run tests, assertions, and mock functions.
- React Testing Library: A testing utility for React that encourages testing components in a way that reflects how users interact with them (i.e., focusing on DOM elements rather than component internals).

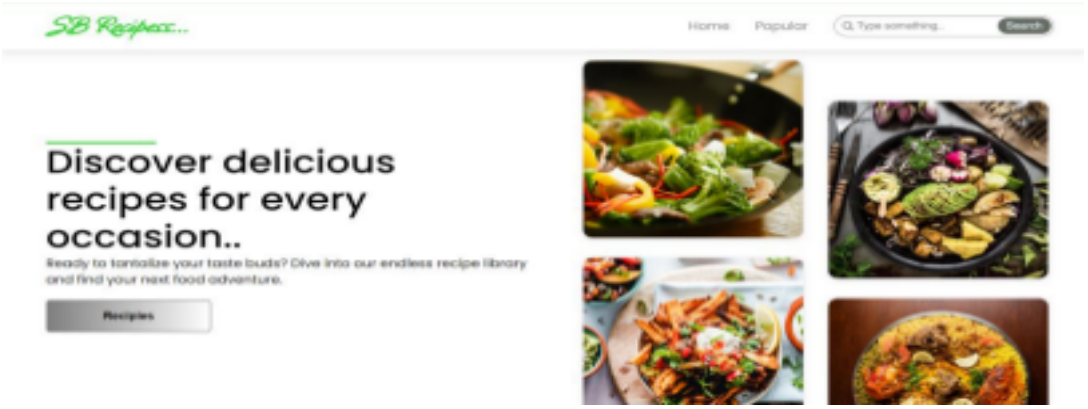
What to Test:

- Functional Components: Test that components render correctly based on given props and state.
- Event Handlers: Test user interactions like button clicks, form submissions, etc., to ensure they trigger the expected functions and state changes.
- Helper Functions: Test utility functions (e.g., `formatDate`, `calculateNutritionalInfo`) in isolation to ensure they return the expected values.

12.Screenshots or Demo:

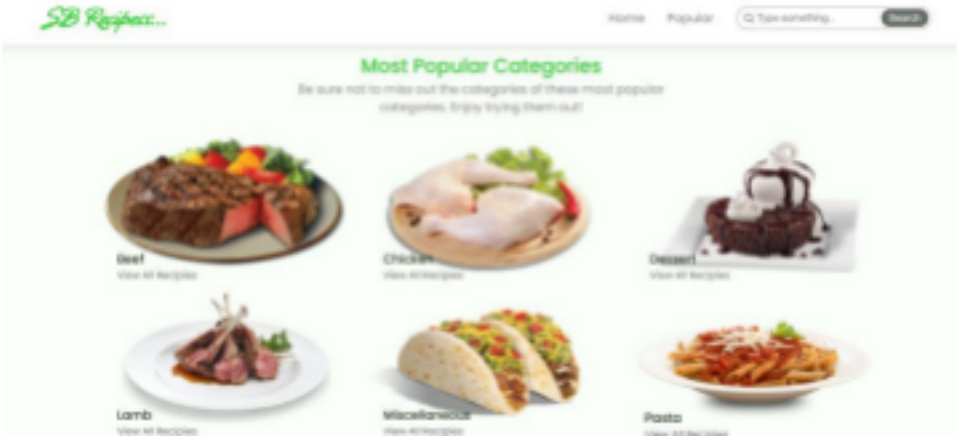
➢ **Hero components**

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



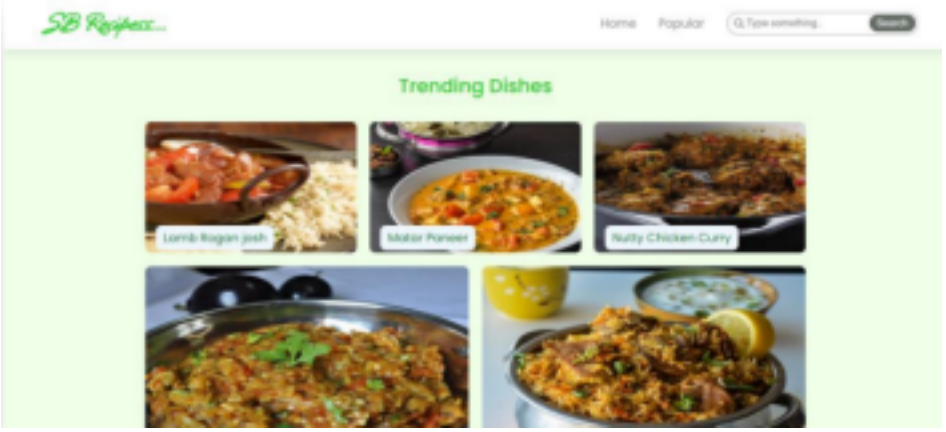
➢ **Popular categories**

This component contains all the popular categories of recipes..



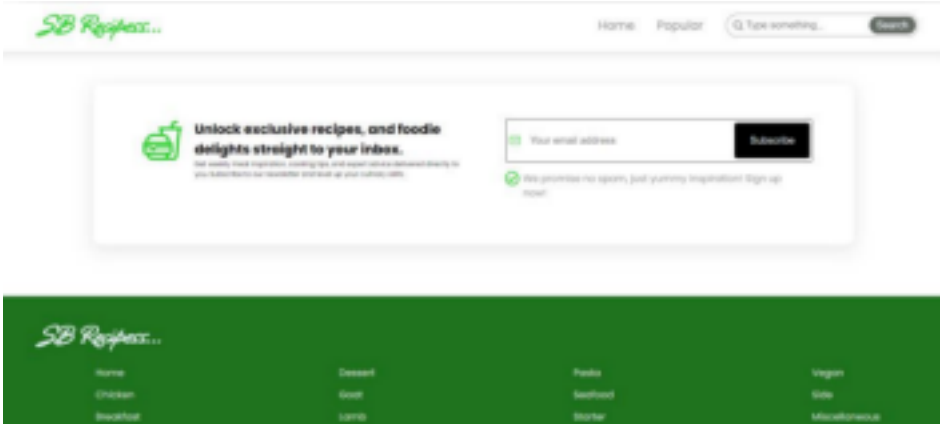
➢ **Trending Dishes**

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



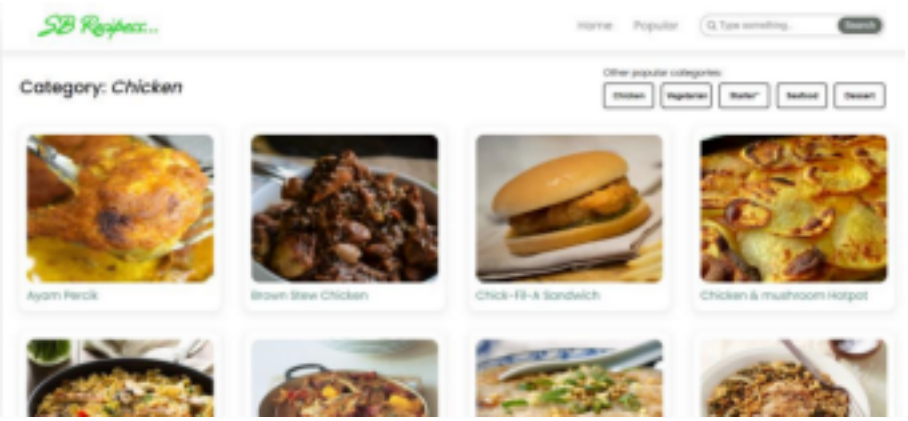
➢ **News Letter**

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



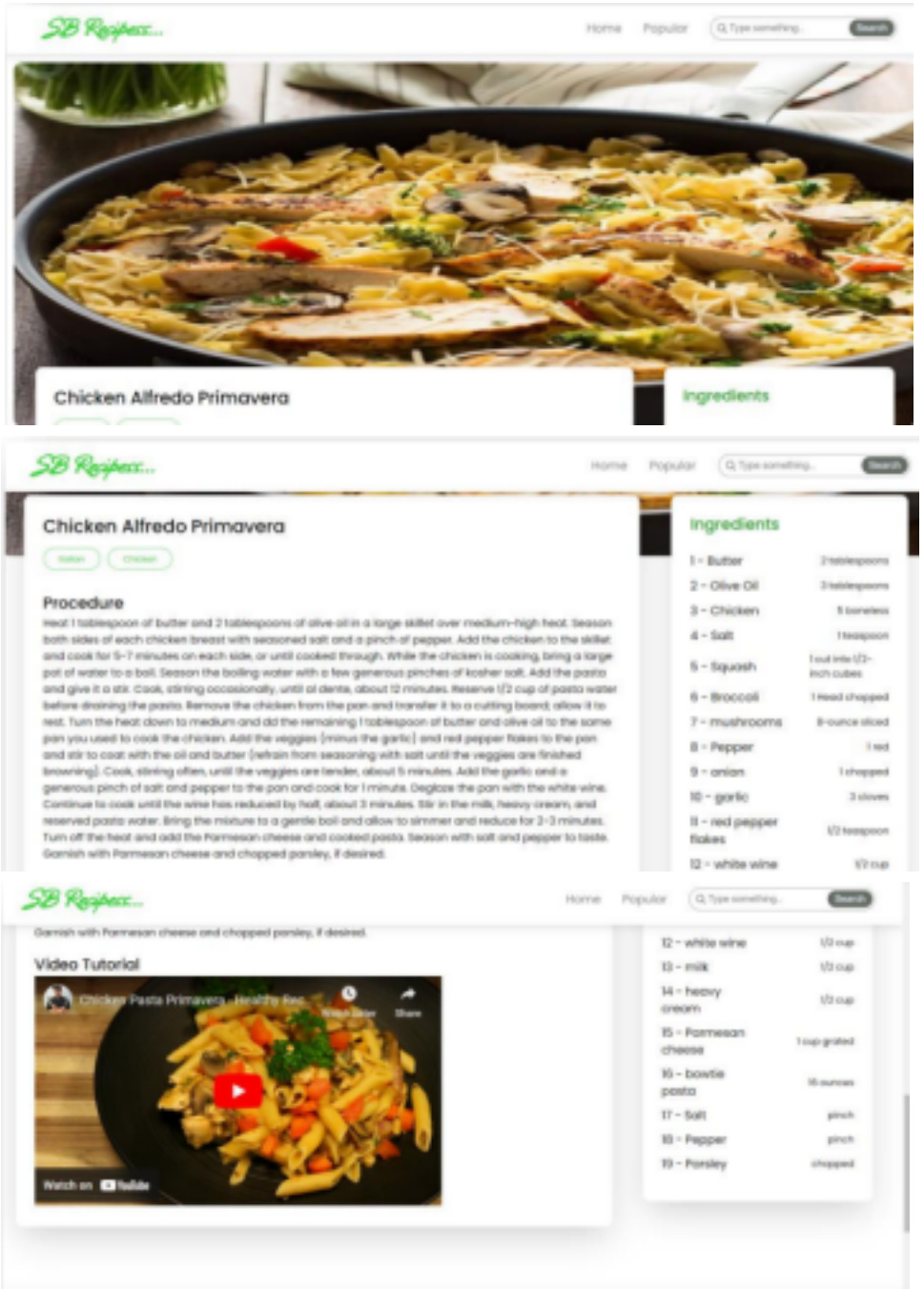
➢ **Category dishes page**

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.



13.Known Issues

Below is a list of known bugs and issues that developers or users might encounter while using or developing the Cookbook application. These issues are currently being addressed and will be fixed in upcoming updates.

User Authentication Flow

- **Issue**: In some cases, users may experience delayed login/logout behavior after authentication (especially if there is a heavy load on the backend API).
- **Impact**: Users may be logged in but the app doesn't immediately reflect the change, or it takes several seconds for the login/logout state to update.

- **Workaround**: Refresh the page to see the updated login/logout state.

Recipe Search Filter Not Clearing Properly

- **Issue**: When users apply multiple filters to the recipe search (e.g., cuisine type, ingredients), the filter states sometimes don't reset correctly when the "Clear Filters" button is clicked.
- **Impact**: If the user applies a new filter after clearing the previous one, the previous filter might still be applied, resulting in incorrect search results.
- **Workaround**: Manually adjust filters after clearing, or use the browser's "reload" feature to reset all filters.

Meal Planner Date Selection Bug

- **Issue**: Users cannot always select a date in the future when planning meals. This bug occurs sporadically and is related to how the calendar component interacts with the state.
- **Impact**: Users may not be able to plan meals beyond the current day, limiting the meal planning functionality.
- **Workaround**: Users can manually select a date within the current week, or wait for the fix in the next release.

14. Future Enhancements

The Cookbook app is constantly evolving, and there are several exciting potential features and improvements planned for future releases. These enhancements will aim to improve the user experience, increase functionality, and keep the app visually appealing and modern. Below are some potential future enhancements:

1. Enhanced User Authentication

- **Feature**: **Social Media Login**
  - **Description**: Add support for logging in with social media accounts like Google, Facebook, or Apple. This would streamline the authentication process, making it quicker and more convenient for users.
  - **Benefit**: Users would be able to sign in faster without remembering another set of credentials.
- **Feature**: **Multi-Factor Authentication (MFA)**
  - **Description**: For added security, allow users to enable multi-factor authentication for their accounts, providing an extra layer of protection.
  - **Benefit**: Increased security for user accounts and sensitive data.

2. Recipe Management Features

- **Feature**: **Recipe Ratings & Reviews**

    - **Description**: Allow users to rate recipes and leave reviews. This will help other users decide whether to try a recipe and foster a community around cooking.
    - **Benefit**: Provides social proof and community engagement, making the app more interactive and helpful.
- **Feature**: **Recipe Comments**
    - **Description**: Allow users to leave comments or tips under each recipe (e.g., cooking modifications, substitutions, or experiences).
    - **Benefit**: Creates a more interactive and social environment within the app and encourages users to share personal insights.
- **Feature**: **Shopping List Suggestions**
    - **Description**: Based on the recipes the user selects, the app will suggest ingredients they may need to add to their shopping list.
    - **Benefit**: Streamlines the meal planning and grocery shopping experience for users.

3. Voice Integration & Accessibility

- **Feature**: **Voice-Controlled Cooking Assistant**
    - **Description**: Integrate voice commands to help users navigate the app hands-free while cooking. For example, users could ask, "What's the next step?" or "How much salt do I need?"
    - **Benefit**: Improves accessibility and convenience for users while they are cooking.
- **Feature**: **Text-to-Speech for Recipe Instructions**
    - **Description**: Allow users to have the recipe instructions read aloud to them step-by-step, enhancing the accessibility for visually impaired users.
    - **Benefit**: Makes the app more inclusive, allowing users to follow along with recipes without needing to read the screen.

15. Conclusion:

The Cookbook application has been designed to offer users an intuitive, engaging, and feature-rich platform for discovering recipes, planning meals, and managing their grocery lists. With a focus on usability and accessibility, the app aims to streamline the cooking process for users of all skill levels, helping them make informed choices, save time, and enhance their culinary experiences.

As the application continues to evolve, the proposed future enhancements and improvements, such as social media logins, personalized recipe suggestions, and interactive meal planning features, will significantly enrich the user experience. The addition of community-driven features, responsive design improvements, and greater accessibility will further empower users to connect, explore, and share their culinary journeys.

With ongoing development, continuous user feedback, and a clear roadmap for future enhancements, the Cookbook app is positioned to become a trusted companion in kitchens worldwide, making cooking and meal planning an enjoyable and seamless experience for all.

Demolink:
https://drive.google.com/drive/folders/1mDLpFaDVPxXncC6Drua4beAVnE_IDZRH?usp=sharing