NAAN MUDHALVAN PROJECT

UNIVERSITY OF MADRAS

COLLEGE NAME: AGURCHAND MANMULL JAIN COLLEGE

COLLEGE CODE : 1301

SUBJECT: FRONTEND DEVELOPMENT WITH REACT.JS

TOPIC: COOK BOOK

TEAM	ROLE	UNM ID	EMAIL ID
ASHWIN KUMAR K(222205183)	*Team Leader*	*unm130122G144*	*ashwinkumark59@gmail.com*
JEEVA V (222205229)	*Team Member*	*unm130122G147*	*jv5170557@gmail.com*
ASATH KAMEEL S (222205181)	*Team Member*	*unm130122G145*	*kameelasath@gmail.com*
SHAMVELL S(222205295)	*Team Member*	*unm130122G146*	*shamvell390@gmail.com*

Project Documentation

Introduction

- **Project Title: CookBook**
- **Team Members:**

- K ASHWIN KUMAR (TEAM LEAD)[EMAIL: ashwinkumark59@gmail.com]
- V JEEVA [EMAIL: jv5170557@gmail.com]
- S SHAMVELL [EMAIL: vellsham10@gmail.com]
- S ASATHKAMEEL [EMAIL: kameelasath@gmail.com]

Project Overview

- **Purpose**

CookBook 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**
- Recipes from the MealsDB API
- Visual recipe browsing
- Intui ve and user-friendly design
- Search feature

Architecture

- **Component Structure**

CookBook_NaanMudhalvan/

CookBook-App/

```
├── node_modules/              # Contains all your project dependencies
├── public/                    # Public assets
│   ├── index.html # Main HTML file
│   ├── favicon.ico            # Favicon for the app
│   └── assets/                # Other assets like images, logos, etc.
├── src/                       # Source code for the React app
│   ├── assets/                # Static assets like images, icons, etc.
│   ├── components/            # Reusable UI components
│   │   ├── Header.js          # Header Component
│   │   ├── RecipeCard.js      # Recipe Card Component
│   │   ├── SearchBar.js       # Search Bar Component
│   │   ├── RecipeList.js      # Recipe List Component
│   │   └── ...                # Other reusable components
│   ├── context/               # React Context API files for state management
│   │   ├── RecipeContext.js # For managing recipe data
│   │   └── UserContext.js     # For managing user-related data (login, preferences)
│   ├── pages/                 # Pages in the app
│   │   ├
```

```
│   │   ├── HomePage.js # Home page (e.g., display recipes, search)
│   │   ├── RecipeDetailPage.js # Recipe detail page (e.g., view single recipe)
│   │   ├── LoginPage.js # Login page
│   │   ├── DashboardPage.js # User's recipe dashboard
│   │   └── ... # Other pages
│   ├── services/ # Files to interact with APIs or backend services
│   │   ├── recipeService.js # Handle API calls for recipe data
│   │   └── userService.js # Handle user-related API calls (login, profile, etc.)
│   ├── utils/ # Utility functions
│   │   ├── format.js # Helper functions (e.g., for date, format, etc.)
│   │   └── validation.js # Form validation helpers
│   ├── App.js # Main app component that combines all the pages
│   ├── index.js # Entry point for React, renders the App component
│   └── styles/ # Global styles (CSS or SCSS)
│       ├── index.css # Main CSS file
│       └── theme.css # Theme-related CSS
├ .gitignore                    # Git ignore file
├──  package.json # Project configuration and dependencies
├──  README.md # Project README file
└── yarn.lock / package-lock.json # Lock file for dependencies
```

- **State Management**

In the **CookBook** application, the `useState` hook is utilized to manage the application's state. The main state variable here is `categories`, which will hold the list of meal categories fetched from the API. The initial value of `categories` is set to an empty array (`[]`), which will later be populated by the fetched data.

State management can be enhanced with **React Context API** or **Redux** if needed for more complex state handling, such as saving recipes, categories, or user preferences across different components.

- **Routing**

React Router is used in this application to handle navigation between different pages. With the help of `react-router-dom`, we can define routes for different pages such as:

- **Home Page** (Recipe List)
- **Recipe Detail Page**
- **Category Page**

React Router ensures a smooth, single-page app experience where users can navigate between different sec ons of the app without a full page reload.

Setup Instructions

Prerequisites

- Node.js (version 14.x or higher)
- npm package manager

Installation

1. Clone the repository:

```
2. git clone ⊕GitHub - Ashwinkumar027/ashwin_27
```

3. Navigate to the project directory:

```
4. cd CookBook-Your-Virtual-Kitchen-Assistant
```

5. Install dependencies:

```
6. npm install
```

Folder Structure

```
src/
|-- components/              # Reusable components for the UI
| |-- RecipeList.js          # Component for displaying all recipes
| |-- RecipeCard.js          # Component for displaying individual recipe cards
| |-- IngredientList.js      # Component for displaying the list of ingredients
| |-- RecipeDetail.js        # Component for displaying recipe details
|-- context/                 # Context API for global state management
| |-- CookbookContext.js     # Context for managing recipes and categories
|-- pages/                   # Page-level components corresponding to different routes
| |-- HomePage.js            # Home page displaying recipe categories and list
| |-- RecipeDetailPage.js    # Page displaying detailed information of a recipe
|-- App.js                   # Main component that contains routing logic
|-- index.js                 # Entry point for React application
```

Client

The client-side architecture includes the following:

- **src/**: Contains the core React applica on code.
  - **components/**: Reusable UI components (e.g., RecipeCard, RecipeList, etc.)
  - **context/**: Handles global state management using **React Context API**.
  - **pages/**: Page components corresponding to different routes (e.g., Home, Recipe Details).
  - **services/**: API service func ons for fetching data (e.g., fetching recipes from an API).
  - **app/store.js**: Global state setup (if Redux is used).

Running the Application

To start the application locally:

```
npm start
```

This will launch the application at `http://localhost:3000`.

Component Documentation

Key Components

RecipeList:

- Displays a list of recipes fetched from an API or stored in local state.
- Can be filtered by categories.

RecipeCard:

- Represents an individual recipe card displayed in the list.
- Shows basic informa on about the recipe (e.g., name, short descrip on).
- Links to the recipe details page.

RecipeDetail:

- Shows detailed informa on about a specific recipe, including ingredients and cooking instruc ons.

- Uses route parameters to fetch a specific recipe's data.
- **Navbar**:
- A naviga on bar to link between different pages of the app (e.g., home, recipe details, etc.).
- **CategoryList**:
- Displays available categories of recipes, allowing users to filter recipes based on the category.
- **Reusable Components**
- **RecipeCard**: Displays brief information about a recipe and is used across various parts of the app (in the list view or search results).
- **SearchBar**: Enables users to search for recipes by name, ingredient, or category.
- **State Management**
- **Global State**

CookbookContext: This context manages global states, including:

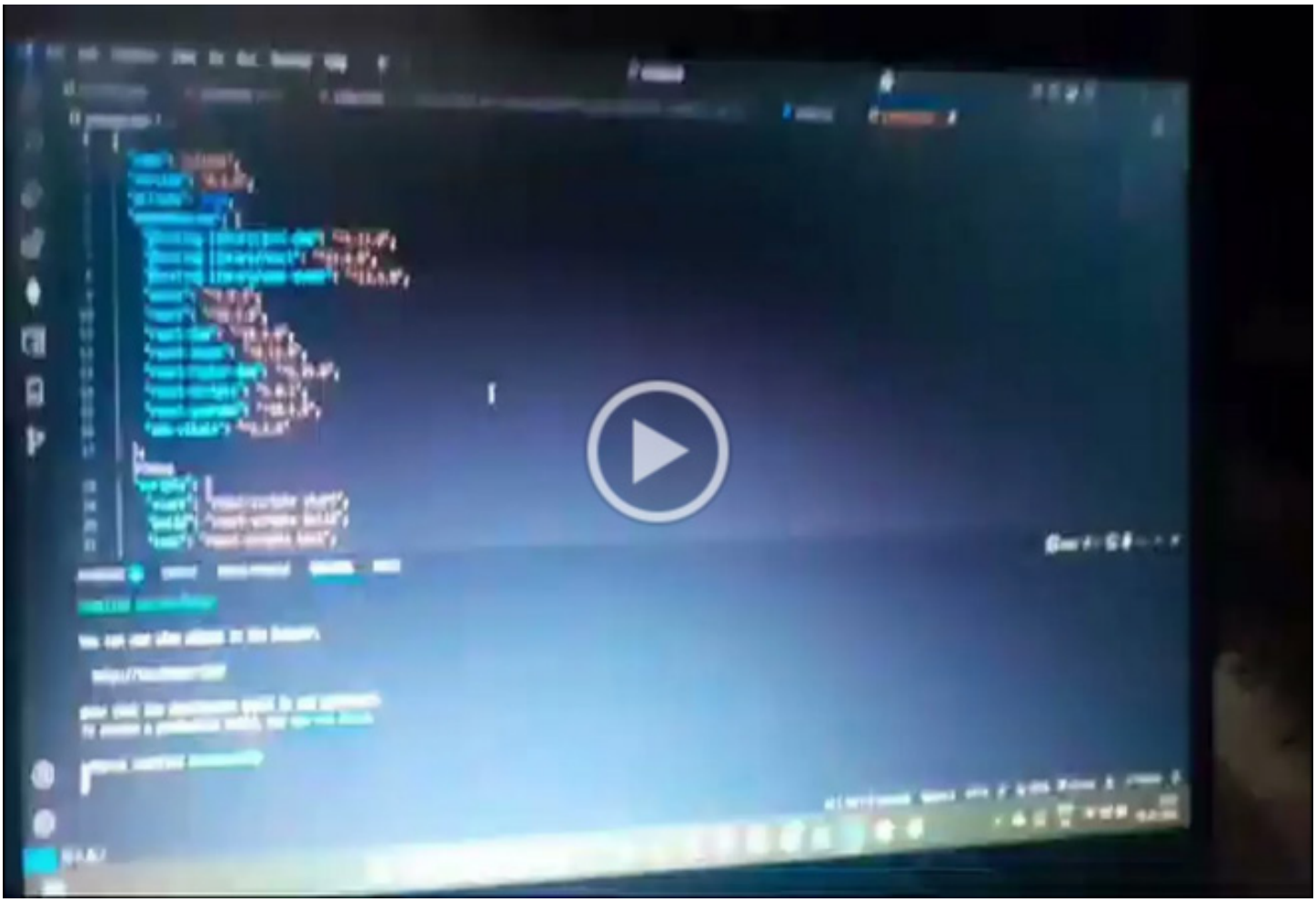
- **categories**: List of categories to categorize recipes.
- **recipes**: List of recipes fetched from an external API or hardcoded.
- **selectedRecipe**: The currently selected recipe (used in the recipe detail page).
- **Local State**

Used for UI-specific states such as form inputs

- **User Interface**

Screenshots or GIFs showcasing different UI features, such as pages, forms, or interactions.

- **Styling**
- **CSS Frameworks/Libraries**
- The application uses **Ant Design** for consistent and responsive UI components.
- **Theming**
- Custom theming is applied using Ant Design's theming capabilities to align with the application's branding.
- **Testing**
- **Testing Strategy**
- The project uses **Jest** and **React Testing Library** for unit and integration testing of components and logic (like API calls, state management, etc.).
- **Code Coverage**
- **Jest** provides built-in code coverage tools, ensuring that tests cover various parts of the app, such as components (e.g., RecipeCard, RecipeDetail) and utility functions (e.g., fetching recipes).
- **Screenshots or Demo**
- **Live Demo**



- https://drive.google.com/file/d/1fabw0uDoCZSLiTExXgAKxRG6dggELMBo/view?usp=drivesdk
-

Known Issues

Known Issues for CookBook: Your Virtual Kitchen Assistant

1. **Recipe and Category Fetching Issues**:
   - **Symptoms**: The categories or recipes may not load correctly when the page is first visited or when switching between categories.
   - **Possible Cause**: API rate limits or connec vity issues might cause the data fetching to fail.
   - **Solu on**: Ensure that the app handles retries or fallback mechanisms, and no fy the user when the data is unavailable. Implement caching if possible to reduce load mes.

1. **API Rate Limits**:
   - **Symptoms**: Data fetching (for recipes, categories, etc.) may occasionally fail or experience delays due to API rate limits imposed by third-party services.
   - **Possible Cause**: APIs like Spoonacular or Edamam may impose limits on the number of requests in a given me frame.
   - **Solu on**: Implement proper error handling (e.g., retry mechanisms, fallbacks) and possibly cache responses to minimize API requests. You can also no fy users if the app has hit the rate limit and suggest wai ng before trying again.

1. **Incomplete Recipe Data**:
   - **Symptoms**: Certain recipes might not display complete details, such as missing ingredients, cooking instruc ons, or nutri onal informa on.

- **Possible Cause**: The data from the API might be incomplete or forma©ed differently across various recipes.
- **Solu on**: Implement data valida on and fallback messages to inform the user when specific data is missing. Addi onally, ensure that the app provides a default template for missing informa on to improve the user experience.

1. **Pagina on Bug**:
   - **Symptoms**: When filtering recipes (by categories, ingredients, etc.), pagina on may not update properly, showing the wrong set of recipes or repea ng data.
   - **Possible Cause**: State management may not be handling the updates to the recipe list and pagina on correctly, especially when switching between filters.
   - **Solu on**: Ensure that the pagina on logic is correctly ed to the filtered data and that the state is updated properly. Debug and log the pagina on state to find the root cause of the problem.

1. **Mobile Responsiveness Issues**:
   - **Symptoms**: Some UI elements, such as the recipe cards or naviga on bar, may not adjust properly for smaller screens or mobile devices.
   - **Possible Cause**: The layout or CSS might not be op mized for all screen sizes, especially on devices with smaller screens.
   - **Solu on**: Ensure that the app uses **responsive design** principles, such as **media queries** and **flexbox** or **CSS Grid**. Test across different devices and screen sizes using browser dev tools or physical devices to catch any layout issues.

1. **Slow Loading Performance**:
   - **Symptoms**: The app might take longer to load, par cularly when dealing with large recipe datasets or when the user switches between different categories or pages.
   - **Possible Cause**: Large datasets or inefficient data fetching methods can cause the app to take longer to render.
   - **Solu on**: Implement pagina on or infinite scrolling to limit the amount of data rendered at once. Use **lazy loading** or **code spli«ng** to op mize ini al load mes. Consider caching API responses or using **Web Workers** to offload data processing in the background.

1. **Search Func onality Bug**:
   - **Symptoms**: The search bar may not return the expected results or may have issues handling mul ple queries in quick succession.
   - **Possible Cause**: Search logic might not be properly debounced, causing unnecessary API calls or search results to be delayed.
   - **Solu on**: Implement **debouncing** or **thro©ling** for search inputs to minimize excessive API calls. Ensure that search queries are properly sani zed and handled to avoid issues like missing or par al matches.

1. **Recipe Detail Page Loading**:
   - **Symptoms**: The recipe detail page may take longer to load, or users might encounter a loading spinner for an extended period without the page content being displayed.

- **Possible Cause**: The API request for a specific recipe might take me to resolve, especially with large or complex recipes.
- **Solu on**: Op mize the detail page's loading state by showing a skeleton loader or placeholders while the content loads. Alterna vely, use **React Suspense** for lazy loading components or data fetching.

1. **UI/UX Inconsistencies Between Pages**:
   - **Symptoms**: There might be inconsistencies in UI elements across different pages, such as font sizes, bu©on styles, or layout alignments.
   - **Possible Cause**: Lack of a consistent global stylesheet or improper use of CSS components across different pages.
   - **Solu on**: Standardize your styling using a **CSS framework** (like **Ant Design** or **Bootstrap**) and ensure a consistent theme is applied across the app. If using custom styles, ensure that global styles are well-defined and scoped to avoid overrides.

1. **User Preferences Not Saved**:
   - **Symptoms**: Users' saved recipes, preferences, or previously viewed recipes are not retained between sessions or page reloads.
   - **Possible Cause**: No mechanism for storing user preferences (e.g., in **localStorage**, **sessionStorage**, or a backend database).
   - **Solu on**: Implement **localStorage** or **sessionStorage** to retain user data between sessions. Alterna vely, you could integrate with a backend (e.g., Firebase or a custom API) to persist user data across sessions.

1. **Ingredient List Display Issues**:
   - **Symptoms**: The list of ingredients may not display properly (e.g., ingredients appear out of order or some ingredients are missing).
   - **Possible Cause**: Data forma«ng issues or the API response might be inconsistent.
   - **Solu on**: Ensure the ingredient list is properly parsed and forma©ed before rendering. Handle missing or malformed ingredient data gracefully by displaying a default message or skipping incomplete entries.

• Future Enhancements

- **Persistent Storage**: Store recipes, user preferences, and categories in a backend database (e.g., Firebase, MongoDB, etc.).
- **Search Func onality**: Implement a search bar that allows users to search recipes by name or ingredients.
- **User Authen ca on**: Implement user login to save favorite recipes and track added recipes