**Frontend Development with React.js**

**Project Documentation for Cook Book ( SB Recipes)**

1. **Introduction**
   * **Project Title**: [Cook Book website (Sb recipes)]
   * **Team Members**: A.Mohammmed Afsal(Full stack developer), Abdul Rahman.M(Data base and front-end developer), Santhosh Kumar (Front-end developer), Mohammad Shafee.S(Design architecture for website), D.Surya(Support role)
2. **Project Overview**
   * **Purpose**: To create a modern, responsive, and dynamic web application for food lovers.

To offer an organized way to store and access recipes easily.

To enable searching, filtering, and categorization of recipes for a seamless experience.

Goals:

User-friendly UI: Ensure an intuitive and visually appealing design.

Recipe Management: Allow users to add, edit, delete, and view recipes.

Search & Filter: Implement search functionality for easy recipe discovery.

Favorites & Collections: Enable users to save and categorize favorite recipes.

Responsive Design: Ensure the app works smoothly on all devices.

* + **Features**: 1. User Authentication (Optional)

Sign up, log in, and log out using email/password or social login.

Profile management for personalized recipe collections.

* + 2. Recipe Management

Add, Edit, Delete Recipes – Users can create and modify their own recipes.

Recipe Categories – Organize recipes into categories like Breakfast, Lunch, Dinner, Desserts, etc.

Ingredients & Instructions – Step-by-step guidance for preparing dishes.

* + 3. Search & Filter

Keyword Search – Find recipes by name or ingredients.

Filters – Sort recipes by category, cooking time, dietary preferences (vegan, gluten-free, etc.).

* + 4. Favorites & Collections

Save favorite recipes to personal collections.

Option to create custom folders (e.g., "My Favorite Desserts").

1. **Architecture**
   * **Component Structure**: 1. UI Components (Reusable)

These components can be used across multiple pages.

1.1 Navbar.js (Navigation Bar)

Displays links to home, recipes, favorites, etc.

import { Link } from"react-router-dom";

const Navbar = () => {

return (

<nav>

<h1>Cookbook</h1>

<ul>

<li><Link to="/">Home</Link></li>

<li><Link to="/recipes">Recipes</Link></li>

<li><Link to="/favorites">Favorites</Link></li>

</ul>

</nav>

);

};

export default Navbar;

---

M1.2 RecipeCard.js (Recipe Preview)

Displays recipe thumbnail, title, and a short description.

import { Link } from "react-router-dom";

const RecipeCard = ({ recipe }) => {

return (

<div className="recipe-card">

<img src={recipe.image} alt={recipe.title} />

<h3>{recipe.title}</h3>

<p>{recipe.description}</p>

<Link to={`/recipes/${recipe.id}`}>View Recipe</Link>

</div>

);

};

export default RecipeCard;

1.3 Button.js (Reusable Button)

A simple, customizable button component.

const Button = ({ text, onClick, type = "button" }) => {

return (

<button type={type} onClick={onClick} className="btn">

{text}

</button>

);

};

export default Button;

* + **State Management**: Use Context API or Redux Toolkit for managing global state:

recipeSlice.js (if using Redux)

RecipeProvider.js (if using Context API)

State can hold:

List of recipes

User authentication (if needed)

Filters (category, ingredients, favorites)

* + **Routing**: Use React Router (react-router-dom) for navigation:

import { BrowserRouter as Router, Route, Routes } from "react-router-dom";

import Home from "./pages/Home";

import RecipeList from "./pages/RecipeList";

import RecipeDetail from "./pages/RecipeDetail";

function App() {

return (

<Router>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/recipes" element={<RecipeList />} />

<Route path="/recipes/:id" element={<RecipeDetail />} />

</Routes>

</Router>

);

}

export default App;

1. **Setup Instructions**
   * **Prerequisites**: Setting up a Cookbook Website requires several prerequisites, depending on whether you’re using a CMS (like WordPress) or coding it from scratch. Here’s a breakdown:

1. Define Your Website Type

Static Site: Uses HTML, CSS,and JavaScript.

Dynamic Site: Uses CMS (WordPress, Joomla) or a web framework (Django, Flask, Node.js).

2. Choose a Domain & Hosting

Domain Name: Purchase from registrars like Namecheap, GoDaddy, or Google Domains.

Web Hosting: Options include:

Shared Hosting (e.g., Bluehost, SiteGround) – For beginners.

Cloud Hosting (e.g., AWS, DigitalOcean) – For scalability.

Managed WordPress Hosting (e.g., WP Engine) – If using WordPress.List software dependencies (e.g., Node.js).

* + **Installation**: Provide a step-by-step guide to clone the repository, install dependencies, and configure environment variables.

1. **Folder Structure**
   * **Client**: public/: Contains static assets accessible in the root URL (e.g., images, favicon, and index.html).

src/: Main source code directory.

assets/: Stores images, styles, and fonts used across multiple components.

components/: Reusable UIcomponents such as buttons, cards, headers, and footers.

pages/: Full-page components that correspond to different routes (e.g., Home, Recipes, About).

routes/: Manages application routing with React Router.

App.js: The main component where the app structure is defined.

index.js: Entry point for rendering the React app.

This structure keeps the project modular, scalable, and maintainable.

* + **Utilities**: hooks/: Custom React hooks for handling logic like fetching data or form management.

context/: Context API for global state management (e.g., recipe data).

services/: Handles API requests and data fetching.

utils/: Helper functions that can be used throughout the app.

Explain any helper functions, utility classes, or custom hooks used in the project.

1. **Running the Application**
   * Provide commands to start the frontend server locally.
     + **Frontend**: npm start in the client directory.
2. **Component Documentation**
   * **Key Components**: 1. Key Components

These components form the core structure of the website.

1.1 Navbar

Purpose:

Displays the website’s navigation menu.

Props:

logo (string): URL for the website logo.

menuItems (array): List of menu links.

Example Usage:

<Navbar logo="/logo.png" menuItems={['Home', 'Recipes', 'About', 'Contact']} />

1.2 RecipeList

Purpose:

Displays a list of recipes, fetched from an API or database.

Props:

recipes (array): List of recipe objects containing { id, title, image, category }.

Example Usage:

<RecipeList recipes={recipeData} />

1.3 RecipeCard

Purpose:

Displays a single recipe preview.

Props:

id (string): Recipe ID.

title (string): Recipe title.

image (string): Recipe image URL.

category (string): Recipe category.

Example Usage:

<RecipeCard id="1" title="Spaghetti Bolognese" image="/spaghetti.jpg" category="Italian" />

1.4 RecipeDetail

Purpose:

Displays detailed information about a recipe.

Props:

id (string): Recipe ID to fetch details.

Example Usage:

<RecipeDetail id="1" />

1.5 CategoryFilter

Purpose:

Filters recipes by category.

Props:

categories (array): List of available categories.

onSelectCategory (function): Callback for selecting a category.

Example Usage:

<CategoryFilter categories={['Italian', 'Mexican', 'Indian']} onSelectCategory={handleCategorySelect} />

---

1.6 SearchBar

Purpose:

Allows users to search for recipes.

Props:

onSearch (function): Callback for search input change.

Example Usage:

<SearchBar onSearch={handleSearch} />

---

1.7 Favorites

Purpose:

Displays the user's saved favorite recipes.

Props:

favoriteRecipes (array): List of favorite recipes.

Example Usage:

<Favorites favoriteRecipes={savedRecipes} /

Document major components, their purpose, and any props they receive.

* + **Reusable Components**: These are commonly used across multiple pages.

2.1 Button

Purpose:

A customizable button used throughout the site.

Props:

text (string): Button label.

onClick (function): Click event handler.

variant (string): Button style (primary, secondary).

Example Usage:

<Button text="View Recipe" onClick={handleClick} variant="primary" />

2.2 Modal

Purpose:

A pop-up modal for displaying additional information.

Props:

isOpen (boolean): Controls modal visibility.

onClose (function): Callback to close modal.

Example Usage:

<Modal isOpen={showModal} onClose={() => setShowModal(false)}>

<p>Recipe Details</p>

</Modal>

2.3 Loader

Purpose:

Displays a loading animation when fetching data.

Example Usage:

{isLoading ? <Loader /> : <RecipeList recipes={recipeData} />}

This documentation provides an overview of the key and reusable components for a React.js

1. **State Management**
   * **Global State**: Global state is data that needs to be accessed and updated across multiple components in the application. In a Cookbook website, examples of global state include:

User Authentication (Logged-in user info)

Favorite Recipes (User’s saved recipes)

Shopping List (A list of ingredients the user wants to buy)

Theme Settings (Dark mode, language preference)

How Global State Flows in the Application

1. Centralized State Store

A global state is stored in a context provider (React Context API) or a state management library like Redux/Zustand/Recoil.

2. State Provider at the Root Level

The state provider wraps the entire application so all components can access it.

3. Components Access the State

Components subscribe to specific pieces of state and update it when necessary.

Example Using React Context API

import React, { createContext, useContext, useState } from 'react';

// Create Global State Context

const RecipeContext = createContext();

// Provider Component

export const RecipeProvider = ({ children }) => {

const [favoriteRecipes, setFavoriteRecipes] = useState([]);

// Function to add a recipe to favorites

const addFavorite = (recipe) => {

setFavoriteRecipes([...favoriteRecipes, recipe]);

};

return (

<RecipeContext.Provider value={{ favoriteRecipes, addFavorite }}>

{children}

</RecipeContext.Provider>

);

};

// Custom Hook to Use the Context

export const useRecipes = () => useContext(RecipeContext);

How to Use Global State in a Component

import React from 'react';

import { useRecipes } from '../context/RecipeContext';

const RecipeCard = ({ recipe }) => {

const { addFavorite } = useRecipes();

return (

<div>

<h3>{recipe.title}</h3>

<button onClick={() => addFavorite(recipe)}>Add to Favorites</button>

</div>

L );

};

export default RecipeCard;

* + **Local State**:Local state is data that is confined within a single component. It is used for UI interactions that don’t need to be shared across components.

In a Cookbook website, local state is used for:

Form Inputs (Adding a new recipe)

Modal Visibility (Showing/hiding recipe details)

Dropdown Selection (Filtering recipes by category)

Handling Local State in Components

React’s useState is commonly used for managing local state within components.

Example: Managing a Recipe Search Input

import React, { useState } from 'react';

const SearchBar = () => {

const [searchTerm, setSearchTerm] = useState('');

return (

<input

type="text"

placeholder="Search recipes..."

value={searchTerm}

onChange={(e) => setSearchTerm(e.target.value)}

/>

);

};

export default SearchBar;

Example: Toggling a Recipe Details Modal

import React, { useState } from 'react';

const RecipeDetails = ({ recipe }) => {

const [isOpen, setIsOpen] = useState(false);

return (

<div>

<button onClick={() => setIsOpen(!isOpen)}>Show Details</button>

{isOpen && <p>{recipe.description}</p>}

</div>

);

};

export default RecipeDetails;

1. **User Interface**
   * screenshots or GIFs showcasing different UI features, such as pages, forms, or interactions, in screenshot pages
2. **Styling**

* **CSS Frameworks/Libraries:**

**To style the Cookbook website effectively, you can use the following:**

* **1. Tailwind CSS – A utility-first framework that speeds up development with pre-defined classes.**

**Example usage:**

**<button className="bg-green-500 text-white py-2 px-4 rounded">Get Recipe</button>**

* **2. Bootstrap – A component-based CSS framework with pre-built styles and responsive grids.**

**Example usage:**

**<button className="btn btn-primary">View Recipe</button>**

* **3. Chakra UI – A React-based component library with built-in theming and accessibility.**

**Example usage:**

**<Button colorScheme="teal">Cook Now</Button>**

* **4. Styled-Components – A CSS-in-JS library to style components dynamically.**

**Example usage:**

**import styled from "styled-components";**

**const Button = styled.button`**

**background-color: #ff5722;**

**color: white;**

**padding: 10px 20px;**

**border-radius: 5px;**

**`;**

**function App() {**

**return <Button>Try Recipe</Button>;**

**}**

* **5. Sass (SCSS) – A CSS pre-processor to use variables, nesting, and mixins for better styling.**

**Example usage (SCSS file):**

**$primary-color: #ff5722;**

**button {**

**background-color: $primary-color;**

**color: white;**

**padding: 10px;**

**}**

**Theming & Custom Design System:**

**To implement custom theming in your Cookbook website, you can use:**

* **1. CSS Variables (Native Theming)**

**:root {**

**--primary-color: #ff5722;**

**--secondary-color: #f4a261;**

**}**

**button {**

**background-color: var(--primary-color);**

**}**

* **2. Theme Provider (Styled-Components)**

**import { ThemeProvider } from "styled-components";**

**const theme = {**

**colors: {**

**primary: "#ff5722",**

**secondary: "#f4a261",**

**},**

**};**

**function App() {**

**return (**

**<ThemeProvider theme={theme}>**

**<RecipePage />**

**</ThemeProvider>**

**);**

**}**

* **3. Chakra UI Theming**

**import { extendTheme, ChakraProvider } from "@chakra-ui/react";**

**const theme = extendTheme({**

**colors: {**

**brand: {**

**500: "#ff5722",**

**},**

**},**

**});**

**function App() {**

**return (**

**<ChakraProvider theme={theme}>**

**<RecipePage />**

**</ChakraProvider>**

**);**

**}**

1. **Testing**

* To ensure the reliability and functionality of the CookBook website, we employ a structured testing approach that includes Unit Testing, Integration Testing, and End-to-End (E2E) Testing.
* 1. Unit Testing
* Purpose: Validate individual components and functions.
* Tools: Jest & React Testing Library.
* What We Test:
* UI Components (e.g., Recipe Card, Search Bar, Favorites List).
* Utility functions (e.g., ingredient formatting, unit conversions).
* State management (Redux or Context API logic).
* Example:
* import { render, screen } from "@testing-library/react";
* import RecipeCard from "../components/RecipeCard";
* test("renders recipe title correctly", () => {
* render(<RecipeCard title="Chocolate Cake" />);
* expect(screen.getByText(/Chocolate Cake/i)).toBeInTheDocument();
* });
* 2. Integration Testing
* Purpose: Ensure that multiple components work together as expected.
* Tools: React Testing Library, Mock Service Worker (MSW) for API mocking.

What We Test:

Fetching and displaying recipes from API.

User interactions across multiple components (e.g., searching recipes, adding favorites).

Example:

import { render, screen, fireEvent } from "@testing-library/react";

import SearchBar from "../components/SearchBar";

test("updates input value on change", () => {

render(<SearchBar />);

const input = screen.getByPlaceholderText("Search recipes...");

fireEvent.change(input, { target: { value: "Pasta" } });

expect(input.value).toBe("Pasta");

});

* 3. End-to-End (E2E) Testing

Purpose: Test the entire application flow from the user’s perspective.

Tools: Cypress or Playwright.

What We Test:

User login, searching, and adding recipes to favorites.

Navigating between pages (e.g., Home → Recipe Detail → Favorites).

API interactions and error handling.

Example (Cypress):

describe("CookBook Website E2E Tests", () => {

it("searches for a recipe and views details", () => {

cy.visit("/");

cy.get("input[placeholder='Search recipes...']").type("Pasta");

cy.contains("Pasta").click();

cy.url().should("include", "/recipe/");

});

});

Code Coverage

Ensuring Adequate Test Coverage

To measure and ensure thorough test coverage, we use:

Jest Coverage Reports: Helps track coverage of statements, branches, and functions.

Coverage Thresholds: Define minimum coverage requirements in package.json.

"jest": {

"collectCoverage": true,

"coverageThreshold": {

"global": {

"statements": 80,

"branches": 75,

"functions": 80,

"lines": 80 }

}

}

* CI/CD Integration: Use GitHub Actions or Jenkins to run tests and fail builds if coverage is insufficient.

1. **Screenshots or Demo**

* https://drive.google.com/file/d/1khMJkccySgKyqRaEZgCpgDACHi572Llj/view?usp=sharing
* link to a demo showcasing the application’s features and design.

1. **Known Issues**

* 1. API & Data Fetching Issues

CORS Errors: If fetching recipes from an external API, ensure the API allows cross-origin requests or use a proxy.

Slow API Responses: Implement loading states to prevent blank screens.

Infinite Loops in useEffect(): Ensure dependencies in useEffect() are correctly set to avoid excessive re-renders.

2. State Management Issues

State Not Updating Immediately: React state updates asynchronously; use useEffect() or functional updates when needed.

Props Drilling: Avoid excessive passing of props by using Context API or Redux.

3. UI/UX Bugs

Recipe Images Not Loading: Ensure image URLs are correct and handle errors with fallback images.

Responsive Design Issues: Use CSS frameworks (Tailwind, Bootstrap) or media queries for proper mobile display.

Form Validation Issues: Validate user input when adding recipes to prevent incorrect data submissions.

4. Performance Issues

Large Recipe Data Causing Lag: Implement pagination or lazy loading to handle large datasets efficiently.

Unnecessary Re-renders: Use React.memo() and useCallback() to optimize performance.

1. **Future Enhancements**

* 1. New Components

Recipe Submission Form – Allow users to submit their own recipes with images and ingredient lists.

Shopping List Generator – Convert recipe ingredients into a downloadable shopping list.

Meal Planner – Let users plan meals for a week and save them.

Nutrition Calculator – Display calories and nutrients based on ingredients.

Video Tutorial Section – Support video uploads or YouTube embeds for step-by-step guidance.

User Profiles & Dashboards – Users can save their favorite recipes and track their cooking history.

Comment & Rating System – Let users rate recipes and leave feedback.

AI-Based Recipe Suggestions – Suggest recipes based on available ingredients.

* 2. Improved Animations & UX Enhancements

Smooth Page Transitions – Use Framer Motion or React Transition Group for seamless navigation.

Hover Effects – Interactive animations on buttons and images using CSS-in-JS (Styled Components or TailwindCSS).

Lazy Loading for Images & Videos – Use React Lazy Load for better performance.

Parallax Scrolling Effects – Enhance visual appeal with a smooth scrolling effect.

Micro-Interactions – Small animations like loading spinners, button clicks, and hover states.

* 3. Enhanced Styling & UI Improvements

Dark Mode Support – Implementlight/dark mode toggling.

Theme Customization – Let users choose between different UI themes.

Modern Card Layouts – Use Material UI or Chakra UI for stylish recipe displays.

Accessibility Improvements – Ensure proper ARIA attributes and keyboard navigation support.

Sticky Sidebar & Navigation – Keep recipe categories or filters easily accessible.

Typography & Color Enhancements Use Google Fonts and well-balanced color schemes.

* 4. Advanced Functionalities

Progressive Web App (PWA) Support Enable offline access to saved recipes.

Voice Command Feature – Let users navigate hands-free while cooking.

Multilingual Support – Provide translations for global audiences.

Ingredient Substitutions – Offer alternative ingredients for dietary restrictions.

Blockchain for Recipe Ownership – Let creators register their original recipes for authentication.

Integration with Grocery Delivery Services – Auto-add ingredients to a shopping cart on platforms like Instacart or Amazon Fresh.

Outline potential future features or improvements, such as new components, animations, or enhanced styling.