**Frontend Development with React.js**

**Project Documentation format**

1. **Introduction** **Project Title**: Cook Book: Your Virtual Kitchen **Team Members**:

**KAVIPRIYA.L**

**KEERTHIGA.R**

**KAYALVIZHI.K**

**KAVIYA SHREE.S**

**KAVYA.A**

1. **Project Overview** **Purpose**: The purpose of the Cook Book – Virtual Kitchen Assistant project is to create a smart and interactive platform that helps users discover, learn, and prepare recipes easily.

**Goals:**

1) Provide step-by-step cooking instructions.

2) Suggest recipes based on available ingredients. 3) Offer nutritional details for healthy eating.

4) Save time by acting as a quick cooking reference guide.

5)Enhance user experience through a virtual assistant approach.

**Features**: The frontend of the Cook Book application includes the following key features:

1) Recipe Search & Filters: Users can search recipes by dish name, ingredients, cuisine, or cooking time.

2)Personalized Suggestions: Recommends recipes based on ingredients entered by the user.

3) Step-by-Step Instructions: Interactive and easy to follow cooking steps.

4) Nutritional Information: Displays calories, protein, carbs, and fat for each recipe.

5)Favourites & Bookmarking: Option to save favourite recipes for quick access

1. **Architecture** **Component Structure**: The application follows a modular component-based architecture in React.

Major components include: App.js – Root component that integrates routing and state management.

* + Header.js – Displays logo, navigation bar, and search bar.
  + Home.js – Landing page with featured recipes and categories.
  + SearchBar.js – Allows users to search recipes by name or ingredients.
  + RecipeList.js – Displays recipe cards fetched from API or database. o RecipeCard.js – Individual card showing recipe image, title, and short description.
  + RecipeDetails.js – Detailed view with ingredients, step-by-step instructions, and nutrition info. o Favorites.js – Lists all bookmarked recipes. oVoiceAssistant.js (optional) – Provides speech output for step-by-step cooking guidance.

Footer.js – Contains links, credits, and contact details.

Interaction Flow:

* + Header/ Search Bar → updates search query → Recipe List updates results.
  + Recipe Card (click) → navigates to Recipe Details.
  + Recipe Details → option to bookmark → updates Favourites.
  + **State Management**: The project uses React Context API for centralized state management. oGlobal States Managed: Current user search query. List of fetched recipes. Bookmarked/favourite recipes (stored in local storage for persistence).

Why Context API?

* + Light weight Easy to manage for a project of this size. Reduces prop drilling by sharing state across multiple components.
  + **Routing**: The app uses react-router for client navigation.
  + Routing Structure:

/ → Home (featured recipes + categories).

/recipes → Recipe List (search results & filters).

/recipes/:id → Recipe Details (full recipe with steps & nutrition).

/favourites → Favourite (bookmarked recipes).

\* → 404 Page (fallback for undefined routes). o

1. **Setup Instructions**
   * + Prerequisites: Before setting up the project, ensure the following software is installed:
     + Nodes’ (v16 or above) – Download here
     + Git – for cloning the repository
     + Code Editor (e.g., Visual Studio Code)
     + Browser: Google Chrome / Firefox for testing
     + Postman: For testing API endpoints (if using backend integration)

**Installation**: 1. Clone the Repository

* + - Git clone [https://github.com/yourusername/cookbook-virtualassistant.git](https://github.com/your-username/cook-book-virtual-assistant.git)
    - Cd cook-book-virtual-assistant 2.

3. Configure Environment Variables

oCreate a .env file in the project root and add the following (example):

(Replace with your recipe/nutrition API keys if applicable.)

4. Start Development Server start This runs the app in development mode.

* + Open [http://localhost:3000 i](http://localhost:3000/)n your browser to view it.

1. Build for Production run build This creates an optimized production build inside the build/ directory.
2. Deployment (Optional)Deploy using hosting platforms like: Netflix, GitHub Pages.

5. **Folder Structure** o **Client**: The project follows a clean and modular folder structure to improve maintainability and scalability.

* A. Client (React Application) o Cook-book virtual-assistant/ o │── public/ # Static files (index.html, favicon,

**Utilities**: The project includes helper functions, utility classes, and custom hooks for reusability: o Custom Hooks:

1)useFetch.js → Handles API requests for fetching recipes/nutrition data.

2)useLocalStorage.js → Stores/retrieves favourite recipes in local storage. Utility Functions (helpers.js): Format recipe titles and instructions. Convert cooking time into readable format Validate user input (e.g., search queries). API Integration (api.js):

1.Functions to call recipe/nutrition APIs.

2.Centralized error handling.

3.Constants (constants.js):

4.API base URLs.

5.Routing paths (/recipes, /favourites).

6. **Running the Application: Provide commands to start the frontend to run the Cook Book – Virtual Kitchen Assistant frontend locally, follow these steps:**

Step 1: Navigate to Project Directory If you are not already inside the project folder, move into it: Cd cook-book virtual assistant.

Step 2: Install Dependencies (Only needed the first time you set up the project.)

Step 3: NPM install and start the fronted server

Step 4: Run the following command to start the React development server:

Step 5: NPM start. This will launch the application on [http://localhost:3000 i](http://localhost:3000/)n your default browser. The app will automatically reload if you make changes in the source code.

Step 6: Stop the Server. To stop the development server, press: CTRL + C

Step 7: Build for Production (Optional). To create an optimized production build, run: NPM run build. This generates static files in the build/ directory, ready for deployment.

Frontend: NPM start in the client directory.

7. **Component Documentation** **Key Components** These are the main building blocks of the application:

1. App.js

/Root component of the application.

/Manages routing and provides global context to child components.

/Displays featured recipes and recipe categories.

/Provides entry point for users to search and explore recipes.

1. RecipeList.js

/Displays a grid/list of recipes based on search results or category.

/Fetches data from the API and passes it to Recipe Card.

1. RecipeDetails.js

/Shows detailed recipe info including ingredients, cooking steps, and nutrition.

/Allows user to bookmark recipe into Favourites.

1. Favorites.js

/Displays all bookmarked recipes stored in local storage or global state.

/Provides option to remove recipes from favourites.

1. NotFound.js

/Shown when a user navigates to an undefined route.

/**Reusable Components**: These components are designed to be used across multiple pages for consistency:

1. Header.js

Contains logo, navigation bar, and search bar.

oProps: title, on Search(query)

2. Footer.js

* Contains copyright and links. o Props: links 3. SearchBar.js
* Reusable search input for recipes. oProps: placeholder, on Search(query)

1. RecipeCard.js Displays recipe preview (image, title, short description). Props: title, image, description, on Click ()
2. Button.js (common) Custom styled button used throughout the app.

Props: label, on Click, variant

1. Modal.js (common)

oGeneric popup modal for alerts or additional info. o Props: is Open, on Close, title, children.

1. **State Management**

**Global State**: Global state is used for data that needs to be shared across multiple components. This is managed using React Context API.

oManaged Data in Global State:

1. Search Query → so that both Search Bar and Recipe List stay in sync.
2. Fetched Recipes → allows Recipe List,

Recipe Card, and Favourites to access the same recipe

* + 3. Favourites /Bookmarks → persisted in local storage and accessed by multiple components (Recipe Details, Favourites).
  + 4. Selected Recipe Details → passed to Recipe Details page for detailed view.
  + Flow: App Context provides state values and functions.

Components like Search Bar update the search query → triggers API fetch → updates Recipe List. o When a recipe is bookmarked, Recipe Details updates global favourites → displayed in Favourites.

**Local State**: Local state is used for component-specific data that does not need to be shared globally. Managed using use State book.

* + Examples of Local State:
  + 1. SearchBar.js – stores temporary input text before submitting search
  + 2. RecipeCard.js – handles hover effects or “expanded/collapsed” states.
  + 3. Modal.js – is Open state to toggle visibility.
  + 4. VoiceAssistant.js – controls whether speech is playing or stopped.
  + 5. Loader.js – tracks loading spinner state during

API fetch. o Flow: o User types in Search Bar → local state stores input.

On submit → value is passed to global state

(search Query) Recipe List listens to global state changes and updates accordingly.

1. **Styling**

* **CSS Frameworks/Libraries**: The project uses a combination of custom CSS and a modern CSS framework for faster development and consistent design.
* Framework/Library Options (depending on what you pick):
* Tailwind CSS → Utility-first CSS framework for responsive and modern UI.
* (Alternative: Bootstrap 5 or Material-UI if you prefer prebuilt components.)
* Custom CSS Modules are used for fine-tuned styling of components.
* Responsive Design is ensured using media queries or framework utilities (e.g., SM: md: classes in Tailwind).
* **Theming**: A custom theme has been implemented to maintain brand identity and consistency.

Primary Theme Colours:

* Primary: Tomato Red (#E63946) – buttons, highlights.
* Secondary: Leaf Green (#2A9D8F) – accents and hover states.
* Background: Cream White (#FFF8F0) – clean and food-friendly layout.

* Text: Dark Charcoal (#333333) – for readability.
* Fonts & Typography:
* Google Fonts (e.g., Poppins, Roboto, or Lato) for a modern look.
* Font hierarchy → Headings (bold), Body (regular).

**10.Future Enhancements**:

**To make the application more powerful, user-friendly, and engaging, the following improvements can be considered in future iterations:**

1)New Features & Components: User Authentication → Allow users to sign up, log in, and sync favourites across devices.

Meal Planner Component → Generate weekly meal plans based on dietary preferences.

Shopping List Generator → Automatically create a grocery list from selected recipes.

Ratings & Reviews System → Enable users to rate recipes and share feedback.

Multi-language Support → Provide recipe instructions in different languages.

#) UI/UX Improvements

Animations & Transitions → Smooth page transitions, hover effects on recipe cards, and interactive loading animations.

Dark Mode → Theme toggle for better accessibility and user preference.

Voice Commands → Extend the Virtual Assistant to accept spoken commands (e.g., “Show me vegetarian recipes”).

Accessibility Enhancements → Keyboard navigation and ARIA labels for visually impaired users.

#) Data & Integration Enhancements

Advanced Filtering → Filter recipes by calories, preparation time, cuisine type, and allergens.

Nutrition API Integration → Provide more accurate and detailed nutritional breakdowns.

Offline Mode (PWA) → Allow access to saved recipes without internet.

Cloud Syncing → Store favourites and preferences in cloud storage.

#) Performance & Scalability

Lazy Loading & Code Splitting → Optimize performance for faster load times.

Caching Strategies → Improve API response times with caching.

AI-based Recipe Recommendations → Suggest recipes based on user history and preferences.