Frontend Development with React.js

Project Documentation format

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

Team Members:

Janani M
Abinaya R
Loshini S

2. **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 easyto-follow cooking steps.
- 4) Nutritional Information: Displays calories, protein, carbs, and fat for each recipe.
- 5)Favorites & Bookmarking: Option to save favorite recipes for quick access.
- 6) Responsive Design: Accessible on mobile, tablet, and desktop.
- 7) User-friendly Interface: Clean navigation with recipe categories (breakfast, lunch, dinner, snacks, desserts, etc.).

3. **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. ◦ RecipeCard.js – Individual card showing recipe image, title, and short description.
   - RecipeDetails.js – Detailed view with ingredients, step-by-step instructions, and nutrition info. ◦ Favorites.js – Lists all bookmarked recipes.
   - VoiceAssistant.js (optional) – Provides speech output for step-by-step cooking guidance.

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

Interaction Flow:

- Header/SearchBar → updates search query → RecipeList updates results.
- RecipeCard (click) → navigates to RecipeDetails.
- RecipeDetails → option to bookmark → updates Favorites.
- **State Management**: The project uses React Context API for centralized state management.
- Global States Managed:Current user search query.List of fetched recipes. Bookmarked/favorite recipes (stored in local storage for persistence).

Why Context API?

- Lightweight ◦ Easy to manage for a project of this size. ◦ Reduces prop drilling by sharing state across multiple components.
- **Routing**: The app uses react-router-dom for clientside navigation.
- Routing Structure:

  / → Home (featured recipes + categories).

  /recipes → RecipeList (search results & filters).

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

  /favorites → Favorites (bookmarked recipes).

  * → 404 Page (fallback for undefined routes). ◦

4. **Setup Instructions**

- Prerequisites:Before setting up the project, ensure the following software is installed:
- Nodes' (v16 or above) – Download here
- Npm (comes with Node.js) or yarn – for dependency management
- 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/your-username/cookbook-virtual-assistant.git
- Cd cook-book-virtual-assistant 2.

  Install Dependencies

- Using npm:Npm install ◦ Or using yarn:Yarn install

3. Configure Environment Variables

- Create a .env file in the project root and add the following (example):
- REACT_APP_API_KEY=your_api_key_here ◦ REACT_APP_BASE_URL=https://api.spoonacular.com/recipes

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

4. Start Development Server

- Npm start ◦ This runs the app in development mode.
- Open http://localhost:3000 in your browser to view it.

5. Build for Production ◦ Npm run build ◦ This creates an optimized production build inside the build/ directory.

6. Deployment (Optional)

- Deploy using hosting platforms like:Netlifx,Vercel,GitHub Pages.

5. **Folder Structure** ◦ **Client**: The project follows a clean and modular folder structure to improve maintainability and scalability.
   - A. Client (React Application) ◦ Cook-book-virtual-assistant/ ◦ │── public/ # Static files (index.html, favicon, manifest)
   - │
   - │── src/ # Main source code ◦ │
     │── assets/ # Images, icons, fonts, and styling resources
   - │ │ ├── images/ ◦ │ │ ├── icons/ ◦ │ │ └── styles/ # Global CSS or SCSS files ◦ │ ◦ │ │── components/ # Reusable UI components
   - │ │ ├── Header.js ◦ │ │ ├── Footer.js ◦ │ ├── SearchBar.js ◦ │ │ ├──

RecipeCard.js ◦ │ │ ├── VoiceAssistant.js (optional) ◦ │ │ └── common/ # Buttons, Modals, Form inputs, etc.
◦ │ │
◦ │ ├── pages/ # Main application pages
◦ │ │ ├── Home.js ◦ │ │ ├── RecipeList.js ◦ │ │ ├── RecipeDetails.js ◦ │ │ ├── Favorites.js ◦ │ │ └── NotFound.js # 404 Page ◦ │ │ ◦ │ ├── context/ # React Context API for global state
◦ │ │ └── AppContext.js ◦ │ │ ◦ │ ├── hooks/ # Custom React hooks ◦ │ │ ├── useFetch.js # For fetching API data ◦ │ │ └── useLocalStorage.js # For persisting favorites
◦ │ │ ◦ │ ├── utils/ # Helper functions & constants
◦ │ │ ├── api.js # API request functions ◦ │ │ ├── constants.js # Reusable constants (routes, config) ◦ │ │ └── helpers.js # Utility functions (formatting, validation) ◦ │ │ ◦ │ ├── App.js # Root component ◦ │ ├── index.js # Entry point ◦ │ ├── routes.js # Routing configuration (if separated) ◦
│
◦ ├── package.json # Project dependencies and scripts ◦ ├── .env # Environment variables (API keys, URLs)

- ◦ │── README.md # Documentation ◦ **Uilities**: The project includes helper functions, utility classes, and custom hooks for reusability: ◦ Custom Hooks:

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

2)useLocalStorage.js → Stores/retrieves favorite 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, /favorites).

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

Step 1: Navigate to Project DirectoryIf you are not already inside the project folder, move into it:Cd cook-bookvirtual-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 in 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.

2. Home.js

/Displays featured recipes and recipe categories.

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

3. RecipeList.js

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

/Fetches data from the API and passes it to RecipeCard.

4. RecipeDetails.js

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

/Allows user to bookmark recipe into Favorites.

5. Favorites.js

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

/Provides option to remove recipes from favorites.

6. 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.
- Props: title, onSearch(query)

2. Footer.js

- Contains copyright and links.
- Props: links

3. SearchBar.js

- Reusable search input for recipes.
- Props: placeholder, onSearch(query)

4. RecipeCard.js

- Displays recipe preview (image, title, short description).
- Props: title, image, description, onClick()

5. Button.js (common)

- Custom styled button used throughout the app.
- Props: label, onClick, variant

6. Modal.js (common)

- Generic popup modal for alerts or additional info.
- Props: isOpen, onClose, title, children.

8. **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.

- Managed Data in Global State:

- 1. Search Query → so that both SearchBar and RecipeList stay in sync. ◦ 2. Fetched Recipes → allows RecipeList, RecipeCard, and Favorites to access the same recipe data.
- 3. Favorites/Bookmarks → persisted in local storage and accessed by multiple components (RecipeDetails, Favorites).
- 4. Selected Recipe Details → passed to RecipeDetails page for detailed view.
- Flow:
- AppContext provides state values and functions. ◦ Components like SearchBar update the search query → triggers API fetch → updates RecipeList. ◦ When a recipe is bookmarked, RecipeDetails updates global favorites → displayed in Favorites.

Local State: Local state is used for component-specific data that does not need to be shared globally. Managed using React's useState hook.

- 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 – isOpen state to toggle visibility.
- 4. VoiceAssistant.js – controls whether speech is playing or stopped.
- 5. Loader.js – tracks loading spinner state during API fetch. ◦ Flow: ◦ User types in SearchBar → local state stores input.

- On submit → value is passed to global state (searchQuery) ◦ RecipeList listens to global state changes and updates accordingly.

9. **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:, lg: classes in Tailwind).
- **Theming**: A custom theme has been implemented to maintain brand identity and consistency.

  Primary Theme Colors:

- 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 favorites 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 favorites 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.