**Cookbook: Your Virtual Kitchen Assistant — Frontend Documentation**

**1. Introduction**

**Project Title:** Cookbook: Your Virtual Kitchen Assistant  
**Team Members:**

**Team ID :** NM2025TMID32561

**Team Leader :** Jothimani D

**Team member :** BRABHA R

**Team member :** DIVYA K

**Team member :** VARSHA K

**Team member :** Suguna V

**2. Project Overview**

**Purpose**

Cookbook is a single-page React application that helps users discover, store, and prepare recipes. It aims to provide an intuitive virtual kitchen assistant with search, personalized recipe collections, step-by-step cooking guides, and grocery list generation.

**Key Features**

* Recipe search and filters (ingredients, cuisine, time, difficulty)
* Recipe detail pages with step-by-step instructions and timers
* User authentication (signup, login, profile)
* Personal recipe collections (favorites, meal plans)
* Grocery list creation and export
* Responsive UI with accessibility best practices
* Offline-friendly caching for recently viewed recipes

**3. Architecture**

**Component Structure (high-level)**

* App — Root app, provides routing and global providers
  + AuthProvider (Context) — authentication state
  + ThemeProvider (Styled/Context) — theme tokens
  + Routes (react-router v6)
    - HomePage — search bar, featured recipes
    - SearchResults — list of RecipeCard components
    - RecipePage — RecipeHeader, IngredientsList, Steps, Timer
    - Profile — user info and collections
    - CollectionPage — collection CRUD
    - NotFound — 404

**Interaction Flow**

* Parent pages fetch data via useEffect and call helper API utilities.
* Data flows down as props to presentational components (RecipeCard, IngredientItem).
* Child components emit events (callbacks) to update local or global state (e.g., add to favorites).

**State Management**

* **Global:** React Context + useReducer for auth, user collections, and theme. Chosen for simplicity and easier testing in a medium-sized app.
* **Server-state:** React Query (or SWR) for fetching and caching recipe data, with background refetch and stale-while-revalidate behavior.
* **Local:** useState and useReducer for component-specific state (form inputs, timers, modal open/close).

**Routing**

* Uses **react-router v6** with route-based code-splitting (React lazy + Suspense).
* Example route tree:
  + / → HomePage
  + /search → SearchResults (?q=, ?cuisine=, ?time=)
  + /recipe/:id → RecipePage
  + /profile → Profile (protected)
  + /collections/:id → CollectionPage (protected)
  + \* → NotFound

**4. Setup Instructions**

**Prerequisites**

* Node.js LTS (≥ 18.x) and npm (≥ 9.x) or Yarn
* Git
* (Optional) .env values for connecting to backend/API keys

**Installation**

1. Clone repository:

git clone https://github.com/your-org/cookbook-frontend.git

cd cookbook-frontend

1. Install dependencies (npm):

cd client

npm install

1. Create environment config:

* Copy .env.example to .env.local and update values:

REACT\_APP\_API\_URL=https://api.example.com

REACT\_APP\_GOOGLE\_CLIENT\_ID=your-google-client-id

REACT\_APP\_ENV=development

1. Start the dev server:

npm start

**5. Folder Structure**

client/

├─ public/

├─ src/

│ ├─ assets/ # images, icons, fonts

│ ├─ components/ # reusable UI components

│ │ ├─ ui/ # Button, Input, Modal, Avatar

│ │ ├─ recipe/ # RecipeCard, IngredientsList, Steps

│ ├─ pages/ # route-level components (Home, Recipe, Profile)

│ ├─ hooks/ # custom hooks (useAuth, useTimer, useDebounce)

│ ├─ context/ # Context providers and reducers

│ ├─ services/ # API clients (axios), auth service

│ ├─ styles/ # global CSS, theme tokens

│ ├─ utils/ # helpers (formatters, validators)

│ ├─ tests/ # unit & integration tests

│ ├─ App.tsx

│ └─ index.tsx

**Utilities**

* services/api.ts — centralized axios instance, request/response interceptors (token refresh).
* hooks/useAuth.ts — auth logic and convenience methods (login, logout, getUser).
* hooks/useLocalStorage.ts — persistent state in localStorage.
* utils/format.ts — value formatters (time, servings).
* utils/validators.ts — form validation helpers.
* Custom hooks are small and single-responsibility to ease testing.

**6. Running the Application**

**Common scripts (in client/)**

# Start dev server

npm start

# Build for production

npm run build

# Run unit tests

npm test

# Run lint

npm run lint

# Run end-to-end tests (Cypress)

npm run cypress:open

(Primary local start command: npm start in the client directory.)

**7. Component Documentation**

**Key Components**

**RecipeCard**

* **Purpose:** Display compact recipe info in lists and search results.
* **Props:**
  + id: string
  + title: string
  + imageUrl?: string
  + timeMinutes?: number
  + onAddToFavorites?: (id: string) => void
* **Notes:** Memoized (React.memo) and accessible (proper alt text and keyboard focus).

**RecipePage**

* **Purpose:** Full recipe detail including ingredients, steps, and timers.
* **Subcomponents:** RecipeHeader, IngredientsList, Steps, NutritionInfo
* **Props:** fetched via router loader or react-query; expects Recipe object.

**SearchBar**

* **Purpose:** Instant search with debounce; handles query suggestions.
* **Props:**
  + onSearch: (query: string) => void
  + initialValue?: string

**AuthForm**

* **Purpose:** Unified sign-in / sign-up form component with validation.
* **Props:**
  + mode: 'login' | 'signup'
  + onSuccess: (user) => void

**Reusable Components**

* Button — supports variants: primary, secondary, ghost.
* Modal — portal-based, focus-trap enabled.
* Input, Select, Checkbox — standardized form controls with validation hooks.

**8. State Management**

**Global State**

* **Auth Context:** Stores user, token, and status (loading/idle). Reducer actions: LOGIN\_SUCCESS, LOGOUT, REFRESH\_TOKEN.
* **Collections Context:** Keeps user collections (favorites, meal plans). Actions support optimistic updates for UX.
* **Theme Context:** Stores theme (light | dark) with persisted preference in localStorage.

State flow: server-state (react-query) is the source for recipe data; global contexts manage user-specific state and local UI preferences. Components read contexts with selectors to minimize re-renders.

**Local State**

* Component-level: form inputs, input validation errors, open/closed modals, timer counts.
* Prefer useReducer in complex local forms (e.g., recipe editor) to keep updates predictable.

**9. User Interface**

Include screenshots or GIFs in docs/assets/screenshots/. Example images:

* docs/assets/screenshots/homepage.png
* docs/assets/screenshots/recipe\_page.png
* docs/assets/screenshots/profile\_collections.png

(Replace with real screenshots or a hosted demo link. If you want, attach images and I will embed them into documentation.)

**10. Styling**

**CSS Frameworks / Libraries**

* **Tailwind CSS** for utility-first rapid UI development.
* **Styled-Components** for component-scoped theming and dynamic styles (used minimally for theme tokens).
* PostCSS for autoprefixing.

**Theming / Design System**

* Theme tokens (colors, spacing, font-sizes) defined in src/styles/theme.ts and exposed via ThemeProvider.
* Supports light/dark modes and accessibility-friendly contrast levels.
* Common components adhere to a small design system (Button variants, spacing scale, typography scale).

**11. Testing**

**Testing Strategy**

* **Unit tests:** Jest + React Testing Library for component behavior, props, render states.
* **Integration tests:** Component + hook interactions (e.g., SearchBar + useDebounce).
* **E2E tests:** Cypress for critical user flows (signup/login, search → open recipe → add to favorites → create grocery list).
* **Mocking:** MSW (Mock Service Worker) for API mocking during unit/integration tests.

**Code Coverage**

* Coverage enforced in CI (minimum 80% for components folder). Use jest --coverage and CI gate to prevent regressions. Focus on critical flows for higher coverage.

**12. Demo / Screenshots**

* **Local demo:** run npm start and open http://localhost:3000.
* **Hosted demo:** [React App](http://localhost:3000/recipie/52922)

**13. Known Issues**

* Image-heavy pages may cause layout shift on slow networks — pending lazy-loading enhancements.
* Some recipes from legacy backend lack normalized nutrition fields (frontend handles missing fields but UI can appear sparse).
* Minor accessibility issue: keyboard users may need improved skip links on nested modals (tracked in issue #42).

**14. Future Enhancements**

* Improved offline mode via service worker caching strategies for full recipe browsing offline.
* Add natural language recipe search (ingredient-based prompts).
* Collaborative meal planning (shareable collections).
* Step-by-step guided cooking mode with voice instructions and auto timers.
* Animation polish for step transitions and ingredient state changes.
* Internationalization (i18n) support — start with en/hi.