**Project Documentation**

CookBook: Your Virtual Kitchen Assistant

**1.Introduction**

Team ID : NM2025TMID33709

Team Size : 5

Team Leader : AKASH D (akashakash8838362413@gmail.com)

Team member : Aravinth (aravintharavinth5144@gmail.com)

Team member : Amudhan (amudhansrinivasano02@gmail.com)

Team member : BALAMURUGAN.D (bala100106@gmail.com)

Team member : Baranidharan (barathibharani@gmail.com)

**Project Overview**

Tech stack: React.js (v18+), React Router, TypeScript (optional), Context API / Redux Toolkit, Tailwind CSS (or CSS Modules), Axios (for API calls), Jest + React Testing Library.

**Primary features:**

Recipe search and filtering

Recipe detail pages with ingredients & steps

User bookmarks/favorites

Meal planner (calendar-like UI)

Shopping list generation and export (CSV)

Basic user preferences (dietary restrictions, measurement units)

Audience: Home cooks, meal planners, recipe enthusiasts.

Architecture

High-level layers:

1. UI Layer (Components): Presentational components, form controls, lists, modals.

2. State Layer: Global app state (Auth, Recipes, Planner, UI flags) via Context or Redux Toolkit.

3. Data Layer: API client (Axios) + caching strategy (in-memory + localStorage).

4. Routing Layer: React Router for page navigation (/, /recipes, /recipe/:id, /planner, /shopping).

Folder-level architecture (conceptual):

src/

├─ api/ # API client & endpoints

├─ assets/ # images, icons

├─ components/ # reusable UI components

├─ features/ # domain features (recipes/, planner/, shopping/)

├─ hooks/ # custom hooks

├─ pages/ # route-level components

├─ store/ # redux slices or context providers

├─ styles/ # Tailwind config or global CSS

├─ utils/ # helpers, formatters

└─ App.tsx

**Setup Instructions**

1. Prerequisites: Node.js 18+, npm or yarn installed.

2. Clone:

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

cd cookbook-frontend

3. Install dependencies:

npm install

# or

yarn

4. Environment variables: Copy .env.example to .env and fill in values (e.g. REACT\_APP\_API\_URL, REACT\_APP\_FEATURE\_FLAGS).

5. Run locally:

npm start

# or

yarn start

6. Build for production:

npm run build

Folder Structure (detailed)

Explain each top-level folder and example files.

src/api/ — client.ts (axios instance), recipes.ts (endpoints)

src/components/ — Button/, Card/, Modal/, SearchInput/

src/features/recipes/ — RecipeList.tsx, RecipeCard.tsx, recipesSlice.ts (if using Redux)

src/pages/ — HomePage.tsx, RecipesPage.tsx, RecipePage.tsx, PlannerPage.tsx, ShoppingPage.tsx

src/hooks/ — useDebounce.ts, useLocalStorage.ts, useAuth.ts

src/store/ — index.ts (store configuration), rootReducer.ts

src/styles/ — tailwind.css or global.css

Running the Application

Development: npm start — opens http://localhost:3000 by default.

Environment options:

PORT — change dev server port

REACT\_APP\_API\_URL — point to staging or mock server

Working with mocks:

Use msw (Mock Service Worker) in src/mocks/ to run without a backend.

Component Documentation

For each major component include:

Name: RecipeCard

Location: src/features/recipes/RecipeCard.tsx

Description: Displays a recipe thumbnail, title, short meta (time, servings), and action buttons (save/share)

Props:

id: string — recipe id (required)

title: string — displayed recipe title

image?: string — optional image URL

onSave?: (id: string) => void — save callback

Accessibility notes: Ensure images have alt, use semantic buttons, keyboard focus states.

Example usage:

<RecipeCard

id={recipe.id}

title={recipe.title}

image={recipe.image}

onSave={(id)=>handleSave(id)}

/>

Repeat the pattern for SearchBar, RecipeList, MealPlanner, ShoppingList, Navbar, Footer, Modal.

State Management

Recommended approach:

Use Redux Toolkit for app-wide state (recipes cache, user settings, planner). Use slices per domain: recipesSlice, plannerSlice, shoppingSlice, authSlice.

Use React Query as an alternative for server-state (caching, background refresh) and keep Redux for local-only state (planner UI, modal states).

Example store setup (Redux Toolkit):

// src/store/index.ts

import { configureStore } from '@reduxjs/toolkit'

import recipesReducer from '../features/recipes/recipesSlice'

export const store = configureStore({

reducer: {

recipes: recipesReducer,

// other slices

},

})

Local persistence: Use redux-persist or custom useLocalStorage hooks to save favorites & preferences.

User Interface

Routing map:

/ — Home / Featured recipes

/search?q= — Search results

/recipes/:id — Recipe detail

/planner — Meal planner

/shopping — Shopping list

UX considerations:

Responsive layout (mobile-first)

Offline-friendly shopping list (saved in localStorage)

Immediate feedback on actions (optimistic UI for favorites)

Styling

Option A — Tailwind CSS (recommended):

Rapid utility classes, consistent spacing scale, dark mode support.

Add tailwind.config.js with design tokens (colors, spacing, fonts).

Option B — CSS Modules / Styled Components:

Use CSS Modules for component encapsulation.

Design tokens:

Define a small token set: --color-primary, --space-4, --radius-lg in :root or Tailwind theme.

Testing

Unit tests: Jest + React Testing Library for components and hooks.

Example test: RecipeCard.test.tsx — renders title, triggers onSave when the save button is clicked.

E2E tests: Cypress (basic flows: search -> open recipe -> add to planner -> generate shopping list).

Test commands:

npm test # run unit tests

npm run test:watch

npm run cypress # open cypress runner

**Screenshots or Demo**

https://drive.google.com/file/d/1ILMFiBHLkcDJcZjqz7iMJ3RUkakMWLQ1/view

How to capture: npm start -> take screenshots at common breakpoints (mobile 375px, tablet 768px, desktop 1280px). Embed images into this repo README.md and into this docs section.

Known Issues

Search debounce must be tuned to avoid excessive API calls on mobile networks.

Some recipes have inconsistent ingredient measurement formatting — needs normalization.

Planner timezone handling: dates are stored in local timezone and may shift for cross-timezone users.

Future Enhancements

User authentication + cloud sync for saved recipes and planners.

Recipe import (upload from URL) and OCR from images.

Collaborative meal planning (share a planner with family).

Nutrition estimation per recipe (calorie/macro breakdown).

PWA support for offline use and installability.

---

Contribution Guidelines (optional)

Fork -> feature branch named feat/<short-desc> -> PR to develop.

Linting: eslint + prettier. Run npm run lint before PR.

Appendix: Useful commands

npm start

npm run build

npm test

npm run lint

npm run storybook # if storybook is used to showcase components