Frontend Development with React.js Project

Documentation format

1. **Introduction** ○ **Project Title**: COOKBOOK

Team Members: List team members and their roles.

(Team leader)PAVITHRA S –pavithra041116@gmail.com

(Team members)SANDHIYA S- sandhiyashankar721@gmail.com **(Team members)NANDHINI B-**nandhininandhini42499@gmail.com **(Team members) SELVI R-**rselvibts2932004@gmail.com

2. **Project Overview** ○ **Purpose**: The Frontend Cookbook serves as a structured guide to streamline frontend development by providing best practices, reusable components, and standardized workflows. Its purpose is to enhance efficiency, maintainability, and collaboration among developers **Goals:**

- Establish consistent coding standards across projects.
- Provide reusable UI components to speed up development.
- Ensure accessibility, performance, and scalability in frontend applications.
  - Facilitate knowledge sharing and onboarding for new developers.
- Improve workflow efficiency through optimized tooling and best practices.
- **Features**:

Reusable UI Component Library

- Prebuilt, customizable UI components (buttons, forms, modals, etc.). • Theming and styling guidelines for consistency.
- Accessibility (ARIA, WCAG compliance) built-in.

Code Standards & Best Practices

- Naming conventions, folder structures, and modular architecture.
- Performance optimization strategies (lazy loading, code splitting).
- Best practices for maintainable and scalable code.

Framework & Library Guidelines

- Setup and usage instructions for React, Vue, Angular, etc. • State management techniques (Redux, Zustand, Vuex).
- API integration patterns and data-fetching best practices.

Development Workflow & Tooling

- Build tools and bundlers (Webpack, Vite, Parcel).
- Version control (Git workflows, branching strategies).
- CI/CD pipeline integration for automated deployments.

Testing & Debugging

- Unit, integration, and end-to-end testing (Jest, Cypress, Playwright).
- Debugging strategies with browser dev tools.
- Performance audits using Lighthouse and Core Web Vitals monitoring.

Performance & Optimization

- Asset optimization (image compression, caching).
- Lazy loading, server-side rendering (SSR), and static site generation (SSG).
- Efficient state management and code-splitting techniques.

Documentation & Knowledge Sharing

- Style guides and component documentation with Storybook.
- Tutorials and onboarding guides for new developers.
- Markdown-based documentation using Docusaurus or similar tools. 3. **Architecture** ○ **Component Structure**:Component Structure in React applications follow a component-based architecture, ensuring modularity and reusability.

Component Types & Structure

- Pages – High-level views (e.g., Home.js, Dashboard.js).
- Layouts – Wrappers for pages (Header.js, Sidebar.js).
- UI Components – Reusable elements (Button.js, Card.js).
- Containers (Smart Components) – Handle state and logic, passing data to UI components.
- Hooks & Utilities – Shared logic (useAuth.js, useFetch.js) **Component Interaction**
- Props – Passes data from parent to child.
- State & Hooks – Manages local and shared state (useState, useEffect).
- Context API – Provides global state without prop drilling.
- State Management (Redux/Zustand) – Manages complex global state.
- **State Management**:Effective state management ensures smooth data flow and maintainability in React applications. This cookbook outlines the primary state management approaches used **Local State (useState, useReducer)**
- Used for component-specific state (e.g., form inputs, toggles). **Context API (Global State)**

- Ideal for app-wide settings (e.g., theme, authentication).
- Avoids prop drilling but can cause unnecessary re-renders.
- **Routing**: The Frontend Cookbook uses React Router for client-side routing, enabling seamless navigation between pages in a React application.
- **Installation**
- Ensure react-router-dom is installed
- **Basic Routing Setup** • Define routes in App.js

4. **Setup Instructions** : **Prerequisites**: **`Ensure the following dependencies are installed before setting up the project`**

- Node.js (v16+ recommended) – Download Here
- Npm (Comes with Node.js) or Yarn – Install Yarn (optional)
- Git – Download Here
- Code Editor – VS Code recommended Download Here
- Browser – Chrome or Edge for best developer tools support **Installation**:
- Clone the Repository • Open a terminal and run:

  Git clone https://github.com/your-repo/frontend-cookbook.git Cd frontend-cookbook
- Configure Environment Variables
- Create a .env file in the project root:

  REACT_APP_API_URL=https://api.example.com
  REACT_APP_API_KEY=your-api-key

5. **Folder Structure** :

Client:A well-organized React project ensures maintainability, scalability, and efficiency. Below is the recommended folder structure for the Frontend Cookbook:

```
/frontend-cookbook
├── /public          # Static assets (index.html, icons, etc.)
├── /src             # Main application source code
│   ├── /assets      # Images, fonts, and static files
│   ├── /components  # Reusable UI components (Button, Modal, Card, etc.)
│   ├── /layouts     # Layout wrappers (Navbar, Sidebar, Footer)
│   ├── /pages       # Route-specific components (Home, Dashboard, Profile)
│   ├── /features    # Feature-based modules (Auth, Notifications, etc.) │
├── /hooks       # Custom hooks (useAuth, useFetch, useTheme)
│   ├── /context     # Context providers (AuthContext, ThemeContext)
│   ├── /store       # State management (Redux/Zustand slices)
```

```
│   ├── /routes       # Application routing (React Router setup)
│   ├── /services     # API calls and external services (fetchUser, authService)
│   ├── /utils        # Helper functions (formatDate, debounce, validators)
│   ├── App.js        # Root component
│   ├── index.js      # Entry point
│   ├── .env          # Environment variables
├── package.json      # Dependencies & scripts
├── README.md         # Project documentation
```

Folder Breakdown & Purpose

- Public/ – Contains static assets like index.html, images, and icons.
- Src/ – Main application code.
- Assets/ – Stores images, fonts, and stylesheets.
- Components/ – Houses reusable UI components (buttons, modals, inputs).
- Layouts/ – Defines common layout structures like navigation bars and sidebars.
- Pages/ – Contains route-based components (e.g., Home.js, Dashboard.js).
- Features/ – Groups related functionalities (e.g., authentication, notifications).
- Hooks/ – Stores reusable custom hooks (useAuth.js, useFetch.js).
- Context/ – Holds React Context for global state management.
- Store/ – Manages state using Redux, Zustand, or other state management tools.
- Routes/ – Manages application routing setup using React Router.
- Services/ – API interaction and business logic (authService.js, api.js).
- Utils/ – Stores helper functions like formatDate.js and debounce.js.
- App.js – Main application wrapper.
- Index.js – Entry point that renders <App />. • .env – Stores environment variables like API keys.

- **Utilities**:The Frontend Cookbook includes various utility functions and custom hooks to improve code reusability, maintainability, and performance.
- **Utility Functions (/src/utils/)**

  Helper functions that handle common tasks like formatting, debouncing, and validation.

  ```
  Export const formatDate = (date) => {
    Return new Date(date).toLocaleDateString("en-US", {
      Year: "numeric",
      Month: "short",
      Day: "numeric",
  ```

});
};

6. **Running the Application** :`Navigate to the Project Directory` If you haven't already, open a terminal and move into the project folder

Cd frontend-cookbook

Optional: Run with Environment Variables
If the project requires environment variables, ensure a .env file exists before running:

REACT_APP_API_URL=https://api.example.com
REACT_APP_API_KEY=your-api-key

Then, restart the server:

Npm start

Frontend:If your project structure includes a client/ directory for the frontend, follow these steps to start the server **Navigate to the Client Directory:**

Cd frontend-cookbook/client

Install Dependencies

Ensure all required dependencies are installed: Npm install

7. **Component Documentation** :

Key Components:This section outlines major React components used in the Frontend Cookbook, their purpose, and the props they receive. **1.Button Component (components/Button.js)** Purpose:

A reusable button component with support for different styles and click events.

```
Const Button = ({ label, onClick, variant = "primary", disabled = false }) =>
{  Return (     <button
    className={`btn ${variant}`}
onClick={onClick}
    disabled={disabled}
  >
    {label}
  </button>
 );
};
```

Export default Button;

Navbar Component (components/Navbar.js) Purpose:

Displays the navigation menu for the application.

```
Const Navbar = ({ user, onLogout }) => {
 Return (
  <nav className=”navbar”>
   <h1>Cookbook</h1>
   <div>
    {user ? (
     <>
      <span>Welcome, {user.name}</span>
      <button onClick={onLogout}>Logout</button>
     </>
    ) : (
     <a href=”/login”>Login</a>
    )}
   </div>
  </nav>
 );
};
```

Export default Navbar;

- **Reusable Components**: Reusable components help maintain a clean, modular, and efficient codebase. Below are some key reusable components in the Frontend Cookbook, along with their configurations and usage.
- **Button Component (components/Button.js)** • A customizable button used throughout the app.

```
Const Button = ({ label, onClick, variant = “primary”, size = “medium”, disabled = false }) => {
 Return (  <button
className={`btn ${variant} ${size}`}
onClick={onClick}
    disabled={disabled}
  >
    {label}
```

```
    </button>
  );
};
```

Export default Button; **Usage:**

<Button label=”Submit” onClick={handleSubmit} variant=”primary” size=”large” />

8. **State Management**:

Global State: Managing global state effectively ensures smooth data flow and consistency across the Frontend Cookbook application. This section describes the global state management approach, how state flows across components, and best practices.

- **State Management Approach**

The application uses Context API with React Reducer to manage global state efficiently. This approach eliminates unnecessary prop drilling and ensures a single source of truth for important data such as user authentication, recipes, and theme settings.

Other possible state management options include Redux or Zustand, but Context API is chosen for its simplicity and built-in support in React.

- **How State Flows Across the Application Authentication Flow**

User logs in → AuthContext updates user state → Components like Navbar reflect login state.

Recipe Flow

Recipes are fetched → RecipeContext updates global state → Components like RecipeList display recipes.

Theme Flow

User toggles theme → ThemeContext updates the global state → The UI updates instantly.

- **Local State**: Local state is essential for managing data that is relevant only to a specific component. The Frontend Cookbook primarily uses React’s useState hook for local state management within components. In some cases, useEffect and useRef are also used for handling side effects and managing component lifecycles.
- **Handling Local State in Components**

  A. Managing Input Fields with useState

For handling form inputs, useState is commonly used.

Example: Form Input State (components/LoginForm.js)

```
Import { useState } from “react”;

Const LoginForm = () => {
  Const [email, setEmail] = useState(“”);
Const [password, setPassword] = useState(“”);

  Const handleSubmit = € => {
    E.preventDefault();
    Console.log(“Logging in with:”, email, password);
  };
  Return (
    <form onSubmit={handleSubmit}>
      <input type=”email” value={email} onChange={€ =>
setEmail(e.target.value)} placeholder=”Email” />
      <input type=”password” value={password} onChange={€ =>
setPassword(e.target.value)} placeholder=”Password” />
      <button type=”submit”>Login</button>
    </form>
  );
};
```

B. Managing Toggle States (e.g., Theme Switcher, Modal Popups)

Local state is useful for controlling UI elements such as modals, dropdowns, or dark/light mode toggles.

Example: Toggle Modal (components/ModalToggle.js)

```
Import { useState } from “react”;
Import Modal from “./Modal”;

Const ModalToggle = () => {
  Const [isOpen, setIsOpen] = useState(false);

  Return (
    <div>
      <button onClick={() => setIsOpen(true)}>Open Modal</button>
```

```
<Modal isOpen={isOpen} onClose={() => setIsOpen(false)} title="Recipe Details">
      <p>This is a recipe description.</p>
    </Modal>
  </div>
 );
};
```

C. Managing Local Side Effects with useEffect

Sometimes, a component needs to react to state changes or perform actions like fetching data when it mounts.

Example: Fetching Data on Mount (components/RecipeList.js)

```
Import { useState, useEffect } from "react";

Const RecipeList = () => {
  Const [recipes, setRecipes] = useState([]);

  useEffect(() => {
Fetch("/api/recipes")
     .then((res) => res.json())
     .then((data) => setRecipes(data));
}, []); // Runs only on mount

  Return (
   <div>
    {recipes.map((recipe) => (
     <div key={recipe.id}>{recipe.title}</div>
    ))}
   </div>
 );
};
```

D. Using useRef for Uncontrolled Inputs

Sometimes, local state isn't needed. Instead, useRef can be used to reference DOM elements directly.

Example: Focus Input Field on Render (components/SearchBar.js)

```
Import { useRef, useEffect } from “react”;
Const SearchBar = () => {
Const inputRef = useRef(null);

 useEffect(() => {      inputRef.current.focus(); // Auto-
focus the input field
 }, []);

 Return <input ref={inputRef} type=”text” placeholder=”Search
recipes…” />; };

Export default SearchBar;
```

9. **User Interfac:**

How to Capture Screenshots

Windows

- Press Win + Shift + S → Select the area → Paste (Ctrl + V) into an image editor. • Use Win + Print Screen to save the screenshot in Pictures > Screenshots. **Mac**
- Press Cmd + Shift + 4 → Select the area → Screenshot saves to Desktop.
- Press Cmd + Shift + 5 for more capture options.

Browser Tools

- Open DevTools (F12 or Cmd + Option + I) → Right-click an element → Capture Screenshot.

How to Record GIFs for UI Interactions

Windows & Mac

- ScreenToGif (Windows) – Free tool for recording interactions.
- LiceCap (Windows/Mac) – Simple GIF recorder.
- ShareX (Windows) – Powerful screenshot and GIF recorder.
- Giphy Capture (Mac) – Easy-to-use GIF creator.

Steps to Record

- Open the Cookbook project in your browser.
- Start the GIF recording tool and select the area.
- Perform the interaction (e.g., clicking a button, submitting a form).
- Save and upload the GIF.

10. **Styling**

- **CSS Frameworks/Libraries**:.
- A. Tailwind CSS (Primary Framework)

- The project uses Tailwind CSS, a utility-first CSS framework, for rapid UI development.

```
<button className=”bg-blue-500 text-white px-4 py-2 rounded-lg hover:bg-blue-600”>
Click Me
</button>
```

CSS Modules (Component-Specific Styles)

For styles that are specific to a component, CSS Modules are used.

```
.button {
  Background-color: #ff6600;
  Color: white;
  Padding: 10px 20px;
  Border-radius: 5px;
  Transition: background-color 0.3s;
}

.button:hover {
  Background-color: #e65c00;
}
```

- **Theming**:. Dark Mode & Light Mode
- The project includes theme switching using Tailwind CSS and Context API
- Example: ThemeContext.js

```
Import { createContext, useState, useEffect } from “react”;

Export const ThemeContext = createContext();

Const ThemeProvider = ({ children }) => {
  Const [theme, setTheme] = useState(localStorage.getItem(“theme”) || “light”);

  useEffect(() => {
    Document.documentElement.classList.toggle(“dark”, theme === “dark”);
localStorage.setItem(“theme”, theme);
  }, [theme]);

  Const toggleTheme = () => setTheme(theme === “light” ? “dark” : “light”);

  Return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
```

```
      {children}
    </ThemeContext.Provider>
  );
};

Export default ThemeProvider;
```

•

11. **Testing** 1. Testing Approach

Unit Testing (Jest & React Testing Library)

Unit tests focus on testing individual components in isolation. Jest and React Testing Library (RTL) are used for unit testing.

Example: Testing a Button Component (Button.test.js)

```
Import { render, screen, fireEvent } from “@testing-library/react”;

Import Button from “../components/Button”;


Test(“renders button with correct text”, () => {

  Render(<Button text=”Click Me” />);

  Expect(screen.getByText(“Click Me”)).toBeInTheDocument();

});


Test(“calls onClick function when clicked”, () => {

  Const handleClick = jest.fn();
```

Render(<Button text=”Click Me” onClick={handleClick} />);

fireEvent.click(screen.getByText(“Click Me”));

expect(handleClick).toHaveBeenCalledTimes(1);

});

- **Code Coverage**: 1. Code Coverage Tools
- To ensure adequate test coverage, the Cookbook frontend uses Jest with Istanbul for unit and integration test coverage. Cypress also provides coverage reports for endtoend (E2E) testing.
-
- A. Jest with Istanbul
- Jest comes with built-in support for code coverage using Istanbul.
- It tracks the percentage of statements, branches, functions, and lines covered in the tests.
- Generating a Coverage Report
- Run the following command:

  npm test -- --coverage

  ----------|---------|----------|---------|---------|-------------------

  File | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s

  ----------|---------|----------|---------|---------|-------------------

  All files | 85.4% | 76.9% | 90.0% | 87.2% |

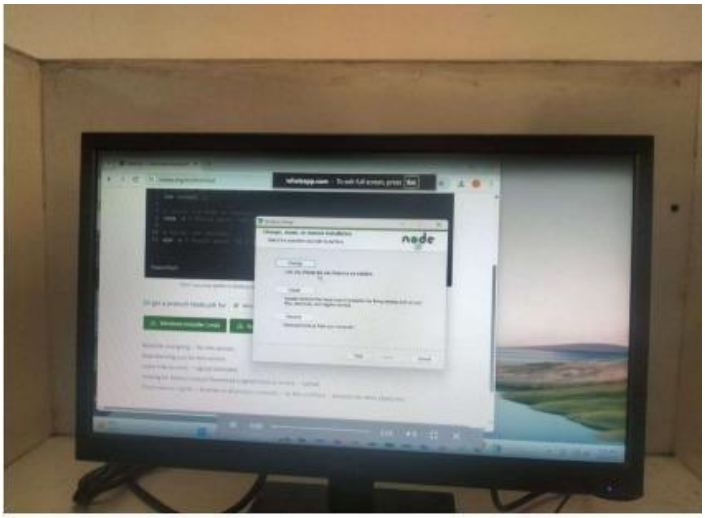
  Src/components | 92.0% | 85.0% | 100.0% | 95.0% |

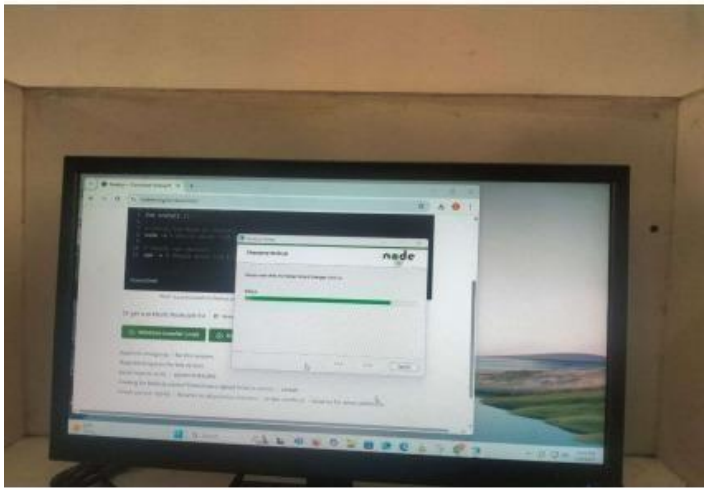
  Src/pages | 78.0% | 65.0% | 82.0% | 80.0% | 12, 45, 89

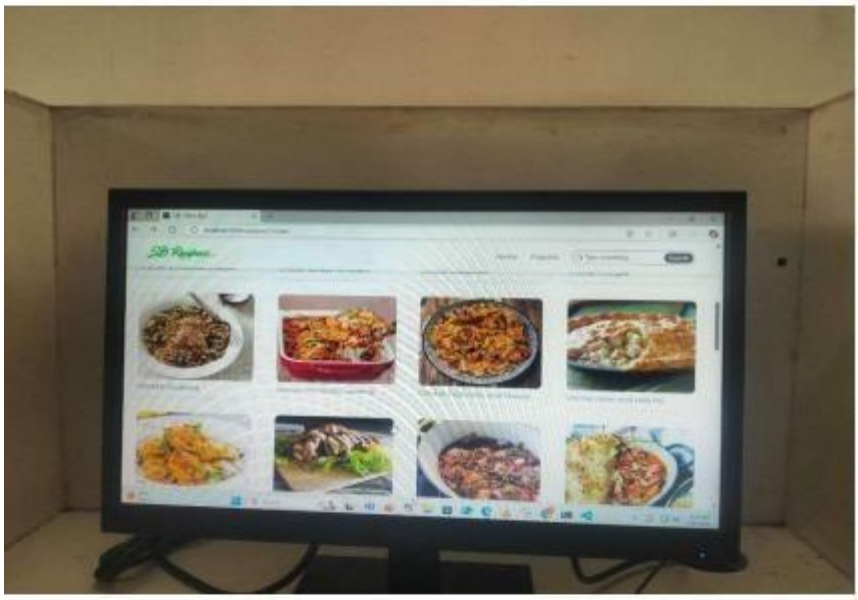
  ----------|---------|----------|---------|---------|-------------------

12.**Screenshots or Demo**

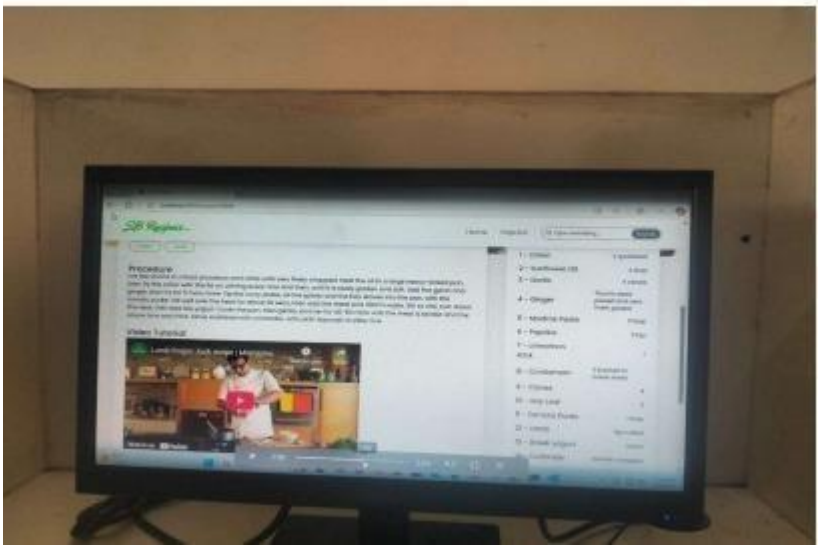
Demolink:
https://drive.google.com/file/d/1ySNIILL9CAQFhugY3nd80H1eTsnr3nhp/view?usp=sharing








Discover delicious recipes
for every occasion..



13. **Known Issues**

While the Cookbook application is designed for a seamless user experience, there are a few known issues that users and developers should be aware of. One notable issue is a brief flash of light mode when the page reloads, even if dark mode is enabled. This occurs due to how theme preferences are applied dynamically. Additionally, some recipe images fail to load if they contain incorrect URLs, though a fallback image has been implemented as a temporary fix.

Form validation also experiences a slight delay in displaying error messages, particularly in cases where users submit incomplete forms. To improve responsiveness, validation

should be triggered on field blur instead of form submission. Another issue involves slow initial loading, particularly on the homepage, due to API calls fetching a large amount of recipe data. Implementing lazy loading and caching techniques can help optimize performance.

On the browser compatibility side, Safari users may notice minor misalignments in grid and flex layouts, which require additional styling adjustments with -webkit- prefixes. Additionally, Internet Explorer 11 is not supported due to the application's reliance on modern JavaScript features.

14. **Future Enhancements** :

To improve the user experience and functionality of the Cookbook application, several enhancements and new features are planned for future updates.

1. New Features & Components

- Advanced Search & Filtering – Implement enhanced filtering options, such as dietary preferences (vegan, gluten-free), cooking time, and ingredient exclusions.
- User Profiles & Favorites – Allow users to create accounts, save their favorite recipes, and track cooking history.
- Recipe Ratings & Reviews – Enable users to rate and review recipes, providing feedback to improve community engagement.
- Shopping List Integration – Add a feature that generates a shopping list based on selected recipes.
- **UI/UX Enhancements**
- Smooth Animations – Implement Framer Motion for fluid transitions and hover effects, enhancing interactivity.
- Dark Mode Improvements – Optimize theme transitions to remove the initial light mode flash.
- Enhanced Recipe Display – Improve recipe layouts with better typography, image galleries, and step-by-step instructions.
- **Performance & Optimization**
- Lazy Loading & Caching – Optimize API calls and images to improve page load speeds.
- Progressive Web App (PWA) Support – Enable offline access and installable app features.

- Accessibility Improvements – Ensure compliance with WCAG standards for better screen reader support and keyboard navigation.