ANNA ADARSH COLLEGE FOR WOMEN

DEPARTMENT OF BCA-SHIFT-II

COLLEGE CODE:1353

TEAM ID : SWTID1741175901150308

TEAM MEMBERS NAME :

MONISHA . D - monisha030905@gmail.com

HARINI . D - dharini0910@gmail.com

DEEPIKA . N - deepikan0305@gmail.com

DHANALAKSHMI .V - dhanalakshmivijayakumar200518@gmail.com

DEVADHARSHINI . J - devadharshini22112004@gmail.com

CookBook: Your Virtual Kitchen Assistant

(React Application) Introduction:

CookBook is a revolutionary web application designed to change the way you discover, organize, and create recipes. It caters to both novice and professional chefs, offering a user-friendly interface, robust features, and a vast collection of inspiring recipes.

Team Members:

D. Monisha – Coding, Documentation, Video Editing & Voice Over
Harini. D - Documentation
Deepika. N - Coding

Dhanalakshmi. V -Documentation
DevaDharshini. J -Coding

Project Overview:

Purpose:

The purpose of a cookbook application is to provide users with a collection of recipes, meal planning tools, and cooking guidance, making it easier to discover, plan, and cook meals.

Features:

1. Recipe search and discovery
2. Meal planning and organization
3. Step-by-step cooking instructions
4. Video tutorials and demos
5. Nutritional information and analysis
6. Grocery lists and shopping planning
7. Recipe saving and bookmarking
8. Customizable cookbooks and collections
9. Social sharing and community features
10. Kitchen management and pantry tracking
11. Dietary restriction and preference filtering
12. Measurement conversion and unit tracking
13. Cooking timers and reminders
14. Recipe rating and review system
15. Offline access and syncing capabilities

Architecture:

Component Structure:

State Management Approach: Flux/Redux-inspired architecture.

State Management Flow:

1. User interacts with the app (e.g., searches for recipes).
2. An action is dispatched to the store (e.g., "SEARCH_RECIPES").
3. The reducer updates the state accordingly (e.g., updates the search results).
4. The updated state is reflected in the app's UI.

Routing Structure:

1. Home Route: / (renders the homepage with featured recipes)
2. Recipe List Route: /recipes (renders a list of all recipes)
3. Recipe Detail Route: /recipes/:id (renders a single recipe's details)
4. Search Route: /search (renders search results)
5. Category Route: /categories/:category (renders recipes by category)
6. Tag Route: /tags/:tag (renders recipes by tag)
7. User Profile Route: /profile (renders the user's profile and saved recipes) 8. Login/Register Route: /login and /register (renders login and registration forms)

Setup Instructions:

PRE-REQUISITES:

Here are the key prerequisites for developing a frontend application using React.js:

✓ **Node.js and npm**:

Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the local environment. It provides a scalable and efficient platform for building network applications.

Install Node.js and npm on your development machine, as they are required to run JavaScript on the server-side.

 React.js:

React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications.

Install React.js, a JavaScript library for building user interfaces.

- Create a new React app:

  npx create-react-app my-react-app
  Replace my-react-app with your preferred project name.

- Navigate to the project directory: cd my-react-app

- Running the React App:

  With the React app created, we can now start the development server and see your React application in action.

- Start the development server:

  npm start

  This command launches the development server, and we can access the React app at http://localhost:3000 in the web browser.

✓ **HTML, CSS, and JavaScript**: Basic knowledge of HTML for creating the structure of our app, CSS for styling, and JavaScript for client-side interactivity is essential.

✓ **Development Environment**: Choose a code editor or Integrated Development

Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

- Visual Studio Code: Download from Download visualcode
- Sublime Text: Download from Download sublime text
- WebStorm: Download from Download webstrom

To clone and run the Application project from Google

drive: Follow below steps:

Install Dependencies:

- Navigate into the cloned repository directory and install libraries:

cd recipe-app-react

npm install

✓ **Start the Development Server**:

- To start the development server, execute the following command:

npm start

Access the App:

- Open the web browser and navigate to http://localhost:3000.
- we should see the recipe app's homepage, indicating that the installation and setup were successful.

we have successfully installed and set up the application on the local machine. We can now proceed with further customization, development, and testing as needed.

Folder Structure:

Client:

Root Directory:

- public/: Publicly accessible files (e.g., index.html, favicon.ico) - src/:

Source code directory

Source Code Directory (src/):

- components/: Reusable React components (e.g., RecipeCard, SearchBar)
- pages/: Top-level React components representing individual pages (e.g., Homepage, RecipeDetail)
- assets/: Static assets (e.g., images, fonts, stylesheets)
- api/: API client code for interacting with backend services
- utils/: Utility functions and helpers
- styles/: Global styles and theme configuration
- routes/: Route configuration and routing logic
- store/: State management store (e.g., Redux, MobX)
- index.js: Application entry point

Components Directory (src/components/):

- RecipeCard/: Recipe card component
- SearchBar/: Search bar component
- Navigation/: Navigation component
- Footer/: Footer component

Pages Directory (src/pages/):

- Homepage/: Homepage component

- RecipeDetail/: Recipe detail component
- SearchResults/: Search results component
- UserProfile/: User profile component

Utilities:

Helper Functions:

1. formatRecipeData(): Formats recipe data from the API into a usable format for the application.
2. convertUnits(): Converts units of measurement (e.g., teaspoons to tablespoons).
3. filterRecipes(): Filters recipes based on user input (e.g., dietary restrictions, ingredients).

Utility Classes:

1. RecipeUtils: Provides utility methods for working with recipes (e.g., calculating cooking time, generating recipe summaries).
2. IngredientUtils: Provides utility methods for working with ingredients (e.g., converting units, formatting ingredient lists).

Custom Hooks:

1. useRecipe(): Fetches and returns recipe data from the API.
2. useSearch(): Handles search functionality, including filtering and sorting recipes.
3. useAuth(): Handles user authentication and authorization.

Other Utilities:

1. constants.js: Defines constants used throughout the application (e.g., API endpoints, default values).

2. enums.js: Defines enums used throughout the application (e.g., recipe categories, cooking methods).

Running the Application:

Frontend:

cd recipe-app-react

npm install

npm start

Component Documentation:

Key Components:

1. RecipeCard

- Purpose: Displays a single recipe's information, including title, image, ingredients, and cooking instructions.

- Props:

- recipe: The recipe object containing title, image, ingredients, and cookings instructions.

- onClick: A callback function to handle clicks on the recipe card.

2. RecipeList

- Purpose: Displays a list of recipes, allowing users to browse and select recipes.

- Props:

- recipes: An array of recipe objects.
- onRecipeSelect: A callback function to handle recipe selection.

3. SearchBar

- Purpose: Allows users to search for recipes by keyword, ingredient, or category.
- Props:
- onSearch: A callback function to handle search queries.

4. RecipeDetail

- Purpose: Displays detailed information about a selected recipe, including cooking instructions and nutritional information.
- Props:
- recipe: The selected recipe object.
- onBack: A callback function to handle back navigation.

5. Navigation

- Purpose: Provides navigation links to different sections of the application, such as recipe list, search, and user profile.
- Props:
- activeSection: The currently active section.

6. UserProfile

- Purpose: Displays user profile information, including saved recipes and cooking preferences.
- Props:

- user: The user object containing profile information.
- onSaveRecipe: A callback function to handle saving recipes.

7. Footer

- Purpose: Displays application metadata, such as copyright information and social media links.
- Props:
- socialMediaLinks: An array of social media links.

Reusable Components:

1. Button

- Purpose: A reusable button component for various actions.
- Configurations:
- variant: primary, secondary, or tertiary
- size: small, medium, or large
- onClick: callback function
- disabled: boolean

2. InputField

- Purpose: A reusable input field component for user input.
- Configurations:
- type: text, email, password, or search
- placeholder: string
- onChange: callback function
- value: string

3. ImageCard

- Purpose: A reusable image card component for displaying recipe images.

- Configurations:

- image: image URL

- altText: string

- onClick: callback function

4. RecipeTag

- Purpose: A reusable recipe tag component for displaying recipe categories.

- Configurations:

- tag: string

- color: string

- onClick: callback function

5. Rating

- Purpose: A reusable rating component for displaying recipe ratings.

- Configurations:

- rating: number (1-5)

- size: small, medium, or large

- color: string

6. Loader

- Purpose: A reusable loader component for displaying loading states.

- Configurations:

- size: small, medium, or large

- color: string

7. Modal

- Purpose: A reusable modal component for displaying overlays.

- Configurations:

- isOpen: boolean

- onClose: callback function

- children: React node

State Management:

Global State:

Global State:

- Recipes: An array of recipe objects

- SelectedRecipe: The currently selected recipe object

- SearchQuery: The current search query string

- User: The logged-in user object (if applicable)

State Flow:

1. Action Dispatch: Components dispatch actions to the Redux store, triggering state updates.

2. Reducer Functions: Reducer functions handle actions and update the global state accordingly.

3. Store Updates: The Redux store updates the global state with the new values.

4. Component Re-renders: Connected components re-render with the updated state.

Example State Flow:

1. User searches for "chicken recipes" in the search bar.

2. The search bar component dispatches a SEARCH_RECIPES action with the search query.

3. The reducer function updates the SearchQuery state with the new value.

4. The recipe list component re-renders with the updated search query, displaying relevant recipes.

Redux Store Structure:

- recipesReducer: Handles recipe-related state updates

- searchReducer: Handles search query state updates

- userReducer: Handles user-related state updates (if applicable)

Local State:

Local State Handling:

1. useState Hook: Components use the useState hook to create and manage local state.

2. State Initialization: Components initialize their local state with default values.

3. State Updates: Components update their local state using the setState function returned by useState.

4. State Usage: Components use their local state to render dynamic content.

User Interface:

➢ **Hero components**

The hero component of the application provides a brief description

about our application and a button to view more recipes.

➢ **Popular categories**

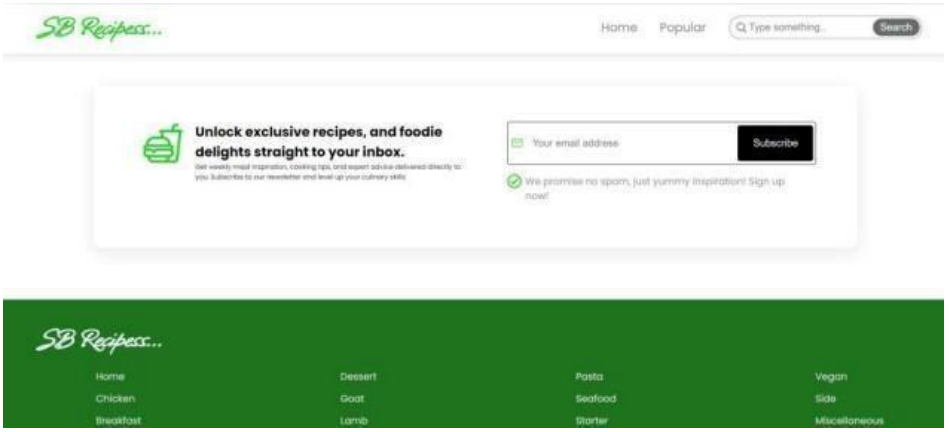
This component contains all the popular categories of recipes..

Trending Dishes

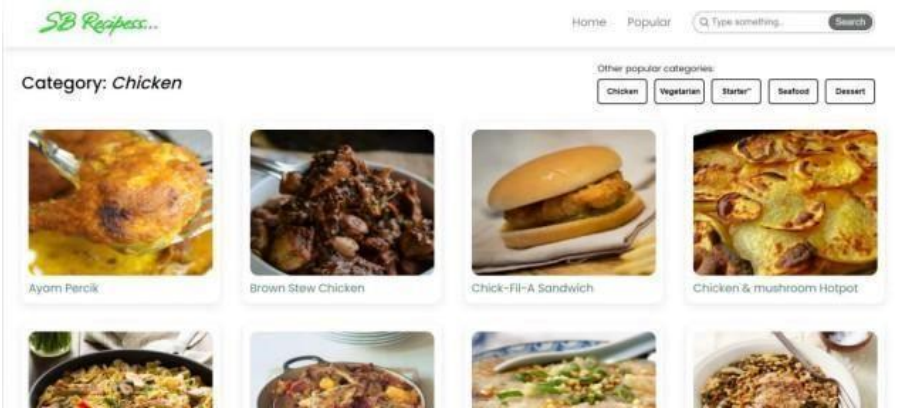
This component contains some of the trending dishes in this application.

➢ **News Letter**

The news letter component provides an email input to subscribe for the recipe newsletters.



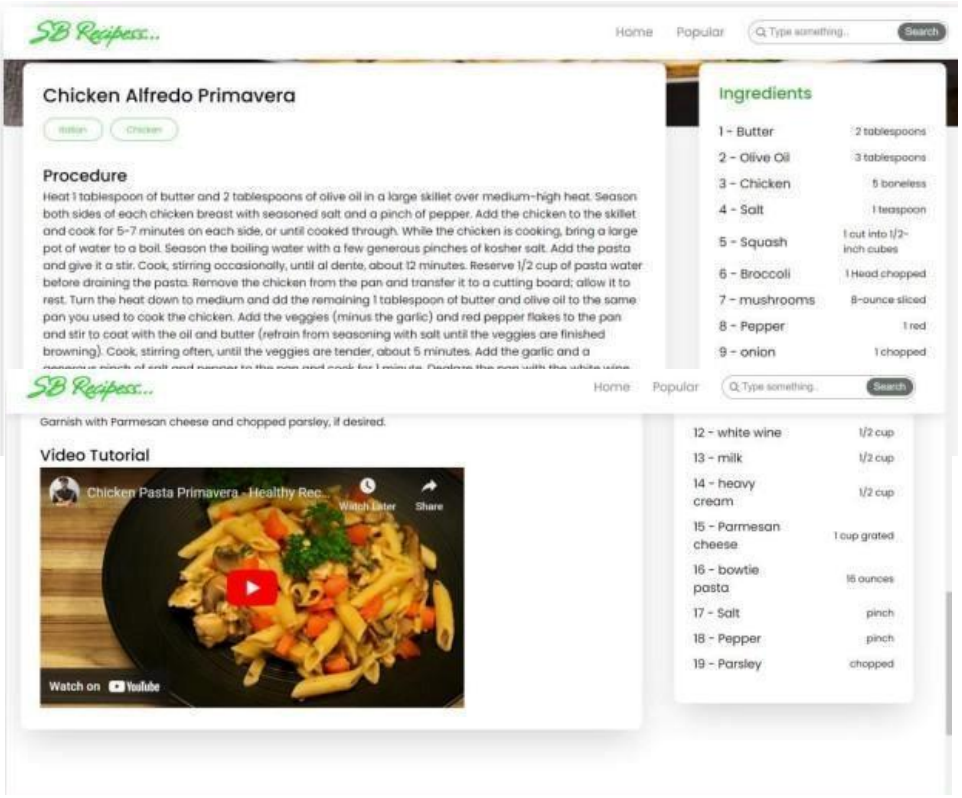
➢ **Category dishes page**



The category page contains the list of dishes under a certain category.

➢ **Recipe page**

The images provided below shows the recipe page, that includes images, recipe instructions, ingredients and even a tutorial video.



Styling:

CSS Framework:

1. Tailwind CSS: A utility-first CSS framework for building custom user interfaces.

CSS Pre-processor:

1. Sass: A CSS pre-processor that allows for more efficient and modular CSS development.

CSS-in-JS Library:

1. Styled Components: A library that allows for writing CSS code in JavaScript files, enabling a more component-driven development approach.

Theming:

Theming System:

1. Color Palette: A predefined color palette is used throughout the application, with a focus on accessibility and readability.

2. Typography: A custom typography system is implemented, with a focus on clear headings, readable body text, and accessible font sizes.

3. Spacing: A consistent spacing system is used throughout the application, with a focus on creating a clear and organized layout.

Testing:

Blackbox testing:

Blackbox testing is a software testing technique that involves testing a software application without knowing the internal structure or implementation details of the application.

Known Issues:

1. Recipe search functionality: The search functionality is case-sensitive, which may lead to incorrect search results.

2. Recipe image upload: The image upload feature is not working correctly in Safari browsers.

3. User profile editing: The user profile editing feature is not saving changes correctly.

Future Enhancements:

1. Meal planning feature: A component that allows users to plan and organize their meals for the week.

2. Recipe ratings and reviews: A component that enables users to rate and review recipes.

3. Nutrition information: A component that displays detailed nutrition information for each recipe.

4. Recipe variations: A component that suggests variations of a recipe based on user preferences.

: