COOKBOOK-YOUR VIRTUAL KITCHEN ASSISTANT

Project Documentation

1. Introduction

Project Title: CookBook – Your Virtual Kitchen Assistant

Team ID: NM2025TMID38O650

Team Leader: Name : MUGILAN D & Mail ID : mgc8MUGILAN24@gmail.com

Team Members:

- Name : MANIKANDAN S & Mail ID : mgc8manikandan24@gmail.com
- Name : STEVE JONATHAN R & Mail ID: mgc8stevejonathan24@gamil.com
- Name: MIDHUNA K & Mail ID : mgc8midhuna24@gmail.com

2. Project Overview

The primary goal of CookBook is to provide a user-friendly platform that caters to individuals passionate about cooking, baking, and exploring new culinary horizons.
Our objectives include:

- **User-Friendly Experience:** Create an interface that is easy to navigate, ensuring users can effortlessly discover, save, and share their favourite recipes.
- **Comprehensive Recipe Management:** Offer robust features for organizing and managing recipes, including advanced search options.
- **Technology Stack:** Leverage modern web development technologies, including React.js, to ensure an efficient, and enjoyable user experience.

3. Architecture

Frontend: React.js with Bootstrap and Material UI

Backend: Node.js and Express.js managing server logic and API endpoints

Database: MongoDB stores user data, recipes, ingredients, reviews, and favorites

4. Setup Instructions

Prerequisites:

Node.Js

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.

- Download: https://nodejs.org/en/download/
- Installation instructions: https://nodejs.org/en/download/package-manager/

MongoDB

Git

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, you 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 you can access your React app at http://localhost:3000 in your web browser.

- **HTML, CSS, and JavaScript**: Basic knowledge of HTML for creating the structure of your 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 https://code.visualstudio.com/download
  - Sublime Text: Download from https://www.sublimetext.com/download
  - WebStorm: Download from https://www.jetbrains.com/webstorm/download

To get the Application project from drive:

Follow below steps:

Install Dependencies:

• Navigate into the cloned repository directory and install libraries:

```
cd code
npm install
```

- **Start the Development Server**:
    - To start the development server, execute the following command:

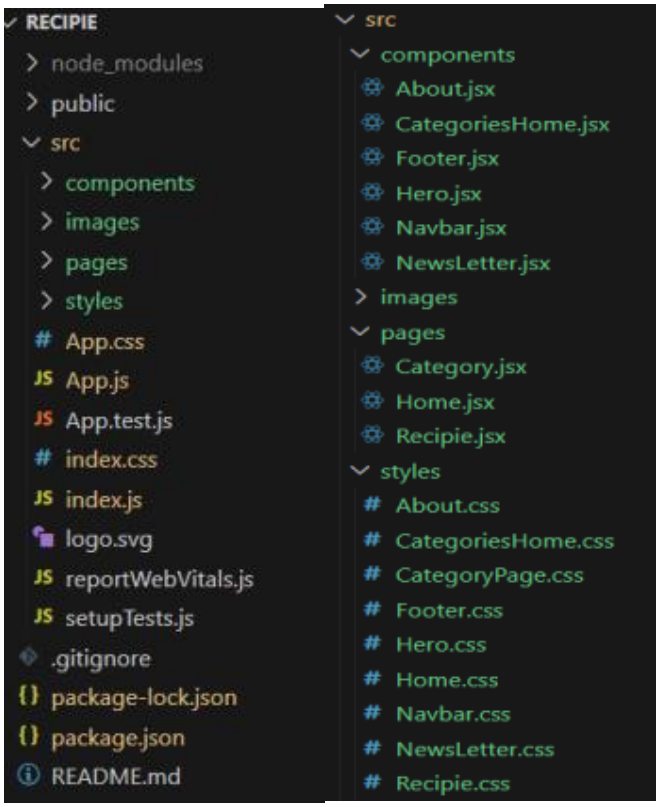
```
npm start
```

Access the App:

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

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

Project structure



In this project, we've split the files into 3 major folders, *Components, Pages and Styles.* In the pages folder, we store the files that acts as pages at different url's in the application. The components folder stores all the files, that returns the small components in the application. All the styling css files will be stored in the styles folder.

Installation Steps:

To build CookBook, we'll need a developer's toolkit. We'll use React.js for the interactive interface, React Router Dom for seamless navigation, and Axios to fetch news data. For visual design, we'll choose either Bootstrap or Tailwind CSS for pre-built styles and icons.

Open the project folder to install necessary tools, In this project, we use:

- React Js
- React Router Dom
- React Icons
- Bootstrap/tailwind css
- Axios

• For further reference, use the following resources

- https://react.dev/learn/installation
- https://react-bootstrap-v4.netlify.app/getting-started/introduction/
- https://axios-http.com/docs/intro
- https://reactrouter.com/en/main/start/tutorial

project developer

- ? Setup the Routing paths

  Setup the clear routing paths to access various files in the application.

```
<Routes>

 <Route path="/" element={<Home />} />
 <Route path="/category/:id" element={<Category />} />
 <Route path="/recipie/:id" element={<Recipie />} />
</Routes>
```

? Develop the Navbar and Hero components

? Code the popular categories components and fetch the categories from ***themealsdb*** Api.

? Also, add the trending dishes in the home page.

? Now, develop the category page to display various dishes under the category.

? Finally, code the recipe page, where the ingredients, instructions and a demo video will be integrated to make cooking much easier.

Important Code snips:

? **Fetching all the available categories**

Here, with the API request to Rapid API, we fetch all the available categories.

```
const [categories, setCategories] = React.useState([])

useEffect(() => {
  fetchCategories()
}, [])

const fetchCategories = async () => {
  await axios.get('https://www.themealdb.com/api/json/v1/1/categories.php')
    .then(response => {
      setCategories(response.data.categories)
      console.log(response.data.categories)
    })
    .catch(error => console.error(error));

}
```

This code snippet demonstrates how to fetch data from an API and manage it within a React component. It leverages two key functionalities: state management and side effects.

State Management with useState Hook:

The code utilizes the useState hook to create a state variable named categories. This variable acts as a container to hold the fetched data, which in this case is a list of meal categories. Initially, the categories state variable is set to an empty array [].

Fetching Data with useEffect Hook:

The useEffect hook is employed to execute a side effect, in this instance, fetching data from an API. The hook takes a callback function (fetchCategories in this case) and an optional dependency array. The callback function is invoked after the component renders and whenever the dependencies in the array change. Here, the dependency array is left empty [], signifying that the data fetching should occur only once after the component mounts.

Fetching Data with fetchCategories Function:

An asynchronous function named fetchCategories is defined to handle the API interaction. This function utilizes the axios.get method to make a GET request to a specified API endpoint (https://www.themealdb.com/api/json/vi/1/categories.php in this example). This particular endpoint presumably returns a JSON response containing a list of meal categories.

Processing API Response:

The .then method is chained to the axios.get call to handle a successful response from the API. Inside the .then block, the code retrieves the categories data from the response and updates the React component's state using the setCategories function. This function, associated with the useState hook, allows for modification of the categories state variable. By calling setCategories(response.data.categories), the component's state is updated with the fetched list of meal categories.

? **Fetching the food items under a particular category**

Now, with the API request, we fetch all the available food items under the certain category.

```
const {id} = useParams();

const [items, setItems] = React.useState([])

useEffect(() => {
  fetchItems(id)
}, [window.location.href])

const fetchItems = async (idd) => {
  await axios.get(`https://www.themealdb.com/api/json/v1/1/filter.php?c=${idd}`)
    .then(response => {
      setItems(response.data.meals)
      console.log(response.data.meals)
    })
    .catch(error => console.error(error));

}
```

This React code snippet manages data fetching from an API.

- It leverages the useState hook to establish a state variable named categories. This variable acts as a container to hold the fetched data, which is initially set to an empty array [].
- The useEffect hook comes into play to execute a side effect, in this instance, fetching data from an API endpoint. The hook takes a callback function (fetchCategories in this case) and an optional dependency array. The callback function is invoked after the component renders and whenever the dependencies in the array change. Here, the dependency array is left empty [], signifying that the data fetching should occur only once after the component mounts.
- The fetchCategories function is an asynchronous function responsible for handling the API interaction. This function utilizes the axios.get method to make a GET request to a predetermined API endpoint (https://www.themealdb.com/api/json/vi/1/categories.php in this

example). This particular endpoint presumably returns a JSON response containing a list of meal categories.

- The code snippet employs the .then method, which is chained to the axios.get call, to handle a successful response from the API. Inside the .then block, the code retrieves the categories data from the response and updates the React component's state using the setCategories function. This function, associated with the useState hook, allows for modification of the categories state variable. By calling setCategories(response.data.categories), the component's state is updated with the fetched list of meal categories.
- An optional error handling mechanism is incorporated using the .catch block. This block is designed to manage any errors that might arise during the API request. If an error occurs, the .catch block logs the error details to the console using the console.error method. This rudimentary error handling mechanism provides a way to identify and address potential issues during the data fetching process.

? **Fetching Recipe details**

With the recipe id, we fetch the details of a certain recipe.

```
const {id} = useParams();

const [recipie, setRecipie] = React.useState()


useEffect(() => {
  fetchRecipie()
}, [])

const fetchRecipie = async () => {
  await axios.get(`https://www.themealdb.com/api/json/v1/1/lookup.php?i=${id}`)
    .then(response => {
      setRecipie(response.data.meals[0])
      console.log(response.data.meals[0])
    })
    .catch(error => console.error(error));
}
```

This React code manages fetching recipe data from an API and storing it within a state variable.

- It leverages the useState hook to establish a state variable named recipie (which is initially empty). This variable acts as a container to hold the fetched recipe data.
- The useEffect hook comes into play to execute a side effect, in this instance, fetching data from an API endpoint. The hook takes a callback function (fetchRecipie in this case) and an optional dependency array. The callback function is invoked after the component renders and

whenever the dependencies in the array change. Here, the dependency array is left empty [], signifying that the data fetching should occur only once after the component mounts.

- The fetchRecipie function is an asynchronous function responsible for handling the API interaction. This function likely utilizes the axios.get method to make a GET request to a predetermined API endpoint, the exact URL construction of which depends on a recipeId retrieved from somewhere else in the code (not shown in the snippet).
- The code snippet employs the .then method, which is chained to the axios.get call, to handle a successful response from the API. Inside the .then block, the code retrieves the first recipe from the data.meals array in the response and updates the React component's state using the setRecipie function. This function, associated with the useState hook, allows for modification of the recipie state variable. By calling setRecipie(response.data.meals[0]), the component's state is updated with the fetched recipe data, effectively making it available for use throughout the component.
- An optional error handling mechanism is incorporated using the .catch block. This block is designed to manage any errors that might arise during the API request. If an error occurs, the .catch block logs the error details to the console using the console.error method. This rudimentary error handling mechanism provides a way to identify and address potential issues during the data fetching process.

5. Folder Structure

```
CookBook/
|-- code/        # React frontend
|  |-- components/
|  |-- pages/
|
|-- server/      # Node.js backend
   |-- routes/
   |-- models/
   |-- controllers/
```

6. Running the Application

Frontend:

```
cd code
npm i
npm start
```

Backend:

cd server
npm start

Access:

Visit: http://localhost:3000

7. API Documentation

User

POST /api/user/register – Register new user

POST /api/user/login – User login

Recipes

POST /api/recipes/create – Add new recipe

GET /api/recipes/:id – Get recipe by ID

GET /api/recipes – Fetch all recipes

POST /api/recipes/:id/review – Add review for recipe

Ingredients

POST /api/ingredients/add – Add ingredients

GET /api/ingredients/:id – Fetch ingredient details

8. Authentication

JWT-based authentication for secure login

Middleware protects private routes

9. User Interface

Landing Page (discover recipes)

Recipe Dashboard (explore recipes)

Recipe Details Page

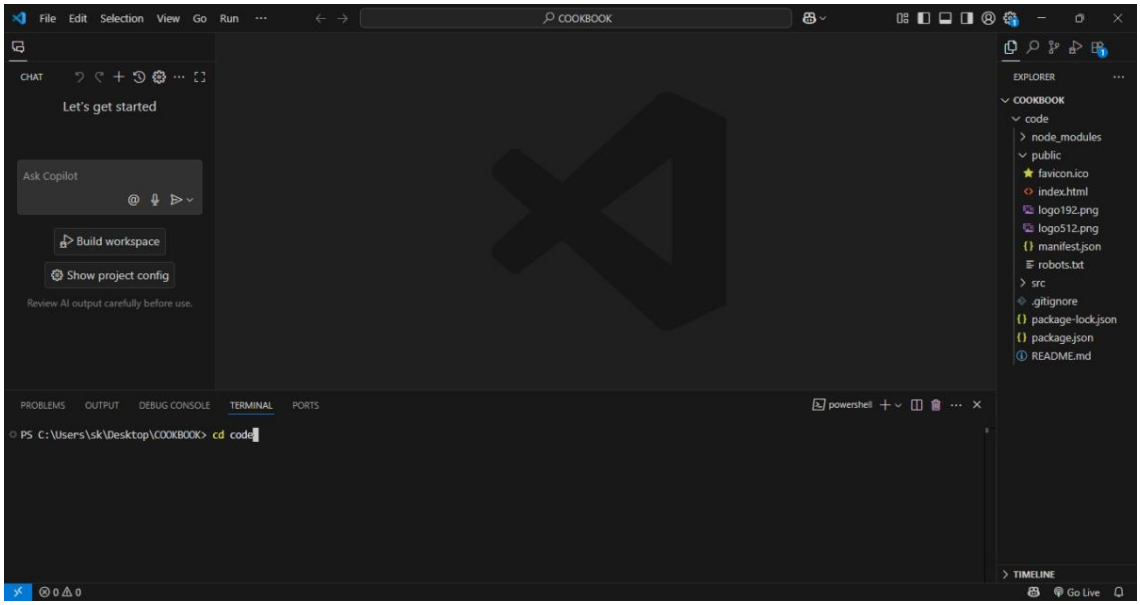
Subscription Panel

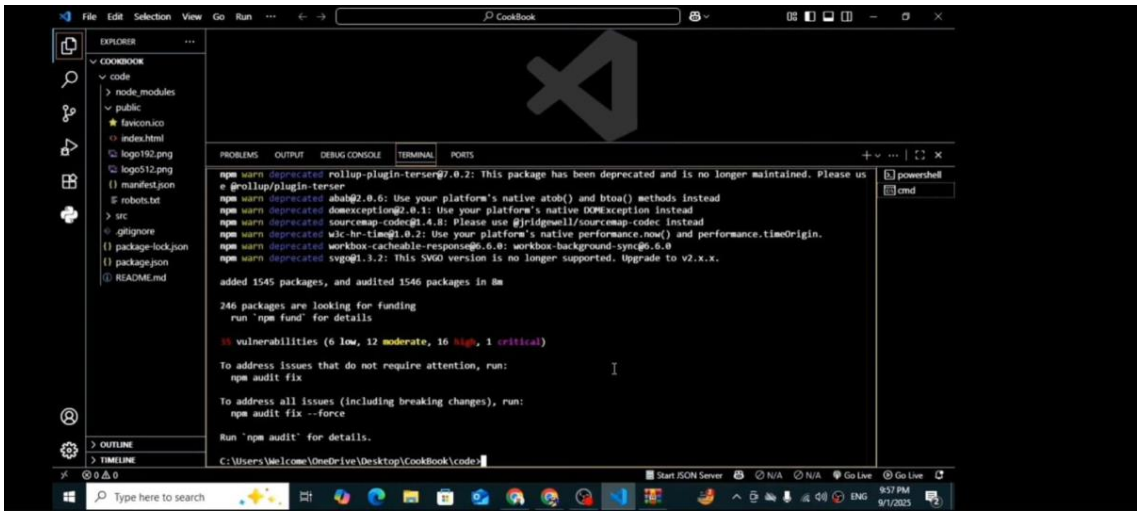
10. Testing

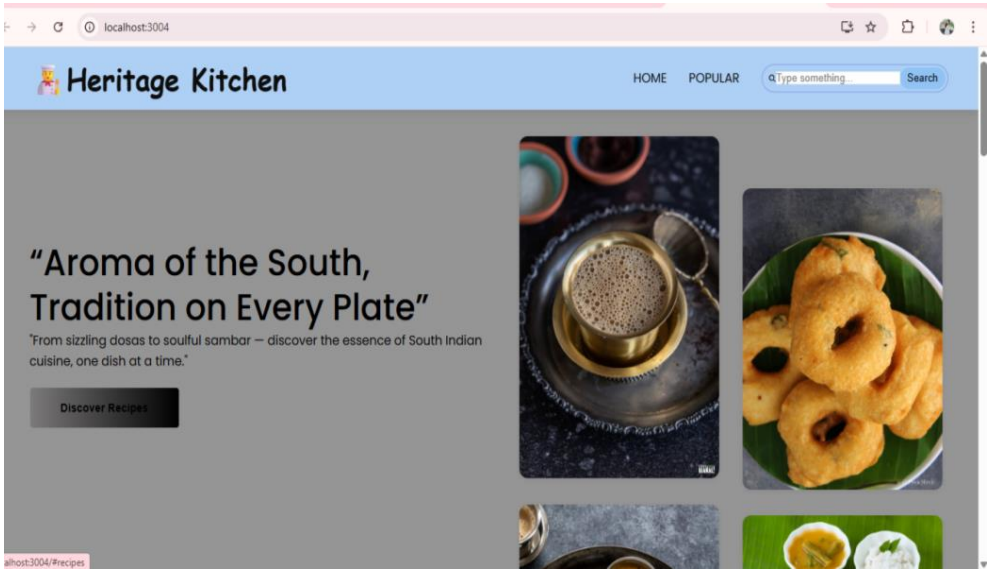
Manual testing during milestones

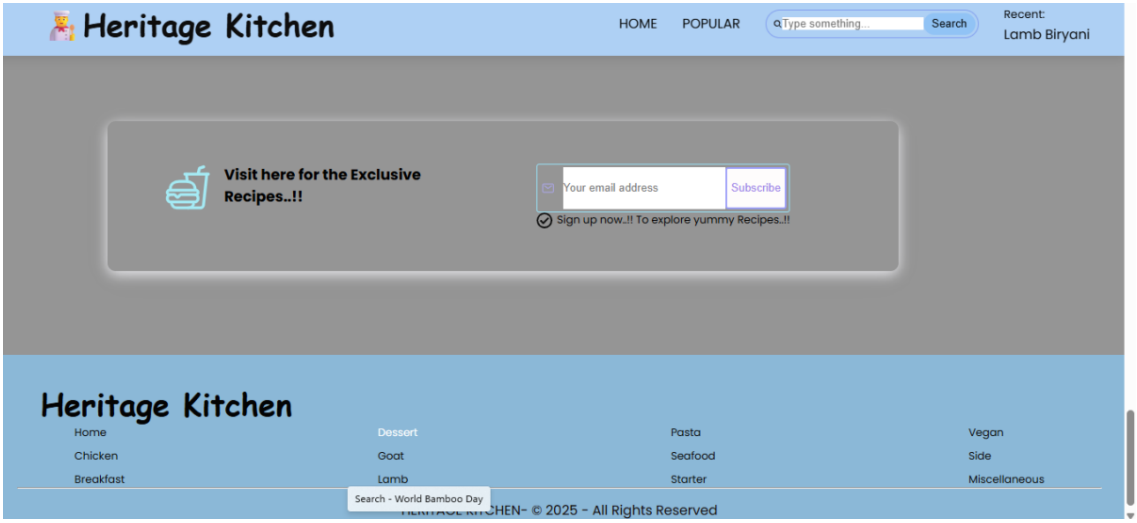
Tools: Chrome Dev Tools

11. Screenshots or Demo




File Edit Selection View Go Run
CookBook
EXPLORER
COOKBOOK
code
node_modules
public
favicon.ico
index.html
logo192.png
logo512.png
manifest.json
robots.txt
src
.gitignore
package-lock.json
package.json
README.md
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
cmd
npm warn deprecated rollup-plugin-terser@7.0.2: This package has been deprecated and is no longer maintained. Please use @rollup/plugin-terser
npm warn deprecated abab@2.0.6: Use your platform's native atob() and btoa() methods instead
npm warn deprecated domexception@2.0.1: Use your platform's native DOMException instead
npm warn deprecated sourcemap-codec@1.4.8: Please use @jridgewell/sourcemap-codec instead
npm warn deprecated w3c-hr-time@1.0.2: Use your platform's native performance.now() and performance.timeOrigin.
npm warn deprecated workbox-cacheable-response@6.6.0: workbox-background-sync@6.6.0
npm warn deprecated svgo@1.3.2: This SVGO version is no longer supported. Upgrade to v2.x.x.
added 1545 packages, and audited 1546 packages in 8m
246 packages are looking for funding
run `npm fund` for details
35 vulnerabilities (6 low, 12 moderate, 16 high, 1 critical)
To address issues that do not require attention, run:
npm audit fix
To address all issues (including breaking changes), run:
npm audit fix --force
Run `npm audit` for details.
C:\Users\Welcome\OneDrive\Desktop\CookBook\code>
OUTLINE
TIMELINE
Start JSON Server
Go Live
Type here to search
ENG
9:57 PM
9/1/2025


localhost:3004
Heritage Kitchen
HOME
POPULAR
Type something...
Search
"Aroma of the South, Tradition on Every Plate"
"From sizzling dosas to soulful sambar — discover the essence of South Indian cuisine, one dish at a time."
Discover Recipes
alhost:3004/#recipes



12. Known Issues

Some UI features under development

Optimization required for large recipe database

Redirection of pages is under development

13. Future Enhancements

Youtube Reference link

AI-based meal planner

Cooking instructions through Voice

Dietary Recipes

Subscription for Premium Features