COOK BOOK: YOUR VIRTUAL KITCHEN ASSISTANT



**1 INTRODUCTION**

**Project Title:**

Cook Book: Your virtual Kitchen Assistant

Team ID:

NM2025TMID35188

Team Leader:

Name: MOHAMED ABDUL KALAM M

Mail ID: [mohamedabdulkalam@gmail.com](mailto:mohamedabdulkalam@gmail.com)

Team Members:

Name: KABILNATH U

Mail ID: [kuttykabil9965@gmail.com](mailto:kuttykabil9965@gmail.com)

Name: KARTHICK S

Mail ID: [samikarthick869@gmail.com](mailto:samikarthick869@gmail.com)

Name: IMTHIYAS AHAMED S

Mail ID: [mrimthiyasking@gmail.com](mailto:mrimthiyasking@gmail.com)

**2 Project Overview**

The “CookBook: Your Virtual Kitchen Assistant” application is designed to revolutionize the cooking experience by providing a comprehensive, all-in-one digital solution for home chefs. It acts as a personal kitchen assistant, simplifying meal planning, recipe management, and the cooking process from start to finish.

Purpose

The primary purpose of the CookBook application is to make cooking more accessible, efficient, and enjoyable for everyone. It aims to reduce the common frustrations of finding recipes, creating shopping lists, and following complex instructions, empowering users to confidently prepare meals.

Features

The application is built around several key features that work together to create a seamless user experience.

\*Recipe Browsing and Management:

\*Search and Filter: Users can search for recipes by keyword, ingredients, cuisine type, dietary restrictions, and more.

\* Personal Cookbook: Users can save their favorite recipes, categorize them, and add personal notes or modifications.

\*Ingredient Scanner: A smart feature to identify ingredients and suggest recipes based on what’s available.

\* Guided Cooking:

\* Step-by-Step Instructions: Provides clear, easy-to-follow, and interactive cooking instructions with timers and multimedia support.

\* Smart Assistant: A real-time assistant to answer questions about substitutions, measurements, or cooking techniques during the process.

\* Community and Collaboration:

\* Feedback and Reviews: Users can rate and review recipes, share their own photos, and provide helpful tips for others.

\* Sharing: The ability to share recipes with friends and family via a secure chat system or a direct link.

\* Administrative Control Panel:

\* A secure control panel for administrators to manage users, approve new recipes, moderate comments and reviews, and monitor overall platform activity.

3 Architecture:

Frontend Architecture

The frontend is the client-side component that users interact with directly. It is responsible for the user interface (UI), user experience (UX), and handling all interactions within the browser.

\* Core Library: React.js

\* React is used to build the single-page application (SPA). Its component-based architecture allows for the creation of reusable UI elements, making the codebase modular and scalable. Each feature, such as a recipe card, a search bar, or a user profile, can be developed as an independent component.

\* UI Frameworks: Bootstrap and Material U

\* Bootstrap is a powerful, mobile-first CSS framework used for creating responsive layouts and grid systems. It ensures the application’s design adapts seamlessly to various screen sizes, from mobile phones to desktops.

\* Material UI is a React component library that implements Google’s Material Design principles. It provides a set of pre-built, aesthetically pleasing components (like buttons, forms, and cards) that offer a consistent look and feel across the application, reducing development time and ensuring a polished user interface.

2. Backend Architecture

The backend is the server-side component that handles all the business logic, data processing, and communication with the database. It is the central hub for the application’s core functionality.

\*Runtime Environment: Node.js

\* Node.js is an open-source, JavaScript runtime environment that allows for the execution of server-side code. Its event-driven, non-blocking I/O model makes it highly efficient and suitable for building fast and scalable network applications.

\* Web Framework: Express.js

\* Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features for building web and mobile applications. It simplifies the process of creating API endpoints (routes), handling HTTP requests, and managing server logic. For the “Cook Book” app, Express.js will manage endpoints for fetching recipes, user authentication, and storing user-generated content.

3. Database

The database is the data persistence layer where all application data is stored, organized, and retrieved.

\*Database System: MongoDB

\* MongoDB is a NoSQL, document-oriented database. Unlike traditional relational databases, MongoDB stores data in flexible, JSON-like documents. This is a perfect fit for a “Cook Book” application, as it allows for dynamic and nested data structures (e.g., a recipe document can contain an array of ingredients, and each ingredient can have its own properties like amount and unit) without a fixed schema. This flexibility is ideal for handling user data, project information, and chat messages.

4 SETUP INSTRUCTIONS:

​4. Setup and Prerequisites

​ This section outlines the necessary tools and steps to get the “Cook Book” application running on your local machine.

​

4.1. Prerequisites

​ Before you begin, ensure you have the following software installed on your system:

​Node.js and npm: The backend is built on Node.js, and npm (Node Package Manager) is used to install all dependencies for both the frontend and backend. You can download the latest version from the official Node.js website.

​MongoDB: The application uses a MongoDB database. You can either install MongoDB locally on your machine or use a cloud-based service like MongoDB Atlas.

​Git: Git is essential for cloning the project’s source code from a repository.

​Code Editor: A text editor or Integrated Development Environment (IDE) like Visual Studio Code, which is highly recommended for JavaScript development.

​Web Browser: A modern web browser like Google Chrome, Mozilla Firefox, or Microsoft Edge.

​4.2. Installation Steps

​Follow these steps to set up and run the application:

​Clone the Repository: Open your terminal or command prompt and clone the project repository using Git.

​Navigate to the Project Directory: Change your current directory to the cloned project folder.

​Install Dependencies: The project has separate package.json files for the frontend and backend.

​Navigate to the backend directory (cd backend) and run npm install.

​Navigate to the frontend directory (cd frontend) and run npm install.

​Configure Environment Variables: Create a .env file in the backend directory. This file will store sensitive information like your MongoDB connection string.

​Start the Backend Server: In the backend directory, run the command npm start. This will start the Express.js server and connect to your MongoDB database.

​Start the Frontend Server: In the frontend directory, run the command npm start. This will launch the React development server.

​Once both servers are running, the application should be accessible in your web browser at <http://localhost:3000>.

5 Folder Structure

Top-Level Directory

The root directory of your project will contain two main folders: client for the React application and server for the Node.js/Express backend. This monorepo-style structure keeps everything in a single repository while maintaining a clear division.

Cookbook-app/

├── client/ (Frontend React app)

├── server/ (Backend Node.js/Express API)

├── .gitignore (Files to be ignored by Git)

├── package.json (Project dependencies and scripts for the monorepo)

├── README.md (Project documentation)

└── …

Client (Frontend) Folder Structure

The client directory will contain all the code for your React application. The src (source) directory is where most of your development will happen.

Client/

└── src/

├── api/ (Handles all API requests to the backend)

├── assets/ (Static files like images, fonts, and icons)

├── components/ (Reusable UI components, e.g., Card, Button, Navbar)

├── context/ (For Context API-based global state management)

├── pages/ (Components that represent a full page/view of the application, e.g., HomePage, RecipeDetail)

├── styles/ (Global CSS files or configuration for styling)

├── utils/ (Utility functions, e.g., date formatting, input validation)

├── App.jsx (The main application component)

└── index.js (Entry point of the React app)

Description of folders:

\* api/: This is where you’ll define functions to handle communication with your backend. For example, api/recipes.js could have a function getRecipe(id) that fetches a recipe from your API.

\* assets/: A centralized place for all static assets.

\* components/: This is the core of your React application. It contains small, reusable, and “dumb” components that don’t know about the application’s state or business logic.

\* context/: If you’re using React’s Context API for state management, this is where you’ll define your contexts and providers.

\* pages/: These components combine smaller components to form the full pages of your application. They typically manage state and handle business logic.

\* styles/: Used for global styles, theme configuration, or CSS-in-JS setup.

\* utils/: A folder for all helper functions and non-component-related logic that can be reused across the application.

Server (Backend) Folder Structure

The server directory will contain all the code for your Node.js/Express backend. It follows the common Model-View-Controller (MVC) pattern.

Server/

├── config/ (Configuration files, e.g., database connection string)

├── controllers/ (Request handlers and business logic)

├── middleware/ (Middleware functions, e.g., authentication, error handling)

├── models/ (Mongoose schemas for MongoDB)

├── routes/ (Defines API endpoints)

└── server.js (The main entry point for the backend server)

6 Running the Application

1 Frontend

The frontend is the user-facing part of the application, built with React. It handles the user interface and communicates with the backend to retrieve and display data. 🍽️

Prerequisites

\* Node.js: v18 or higher

\* npm or yarn

Installation and Running

* Clone the Repository: Get the frontend code from its repository.

Git clone <https://github.com/your-repo/cookbook-frontend.git>

Cd cookbook-frontend

* Install Dependencies: Use npm or yarn to install all the required packages.

Npm install

* Configure Environment Variables: Create a .env file in the root directory. This file will hold configurations like the backend API endpoint.

REACT\_APP\_API\_URL=http://localhost:5000/api

Note: Replace <http://localhost:5000/api> with the actual backend URL if it’s hosted elsewhere.

* Run the Application: Start the development server. The frontend will typically be accessible on <http://localhost:3000>.

Npm start

2 Backend

The backend is the server-side component of the application. It manages data, handles business logic, and provides an API for the frontend. 💾

Prerequisites

\* Node.js: v18 or higher

\* npm or yarn

\* Database: MongoDB (or another specified database)

Installation and Running

* Clone the Repository: Get the backend code from its repository.

Git clone <https://github.com/your-repo/cookbook-backend.git>

Cd cookbook-backend

* Install Dependencies: Install the server-side packages.

Npm install

* Configure Environment Variables: Create a .env file in the root directory for database connection strings, API keys, and other secrets.

PORT=5000

MONGO\_URI=mongodb://localhost:27017/cookbook

API\_KEY=your\_secret\_api\_key

Note: Adjust the MONGO\_URI to your specific MongoDB connection string.

* Run the Application: Start the backend server. It will typically run on the port specified in your .env file (e.g., <http://localhost:5000>).

Npm start

1. Access

To use the CookBook application, you’ll need both the frontend and backend running simultaneously. The frontend makes API calls to the backend to function correctly. 🔗

Local Access

\*Ensure both the frontend and backend applications are running in their respective terminals as per the instructions above.

\*Open your web browser and navigate to the frontend’s URL, which is <http://localhost:3000>.

\*The application should load, and you can begin using your virtual kitchen assistant!

Remote Access (if hosted)

\* Frontend: Access the application via the domain name or IP address where the frontend is deployed (e.g., <https://cookbook.yourdomain.com>).

\* Backend: The backend API will be running on a server. The frontend configuration (REACT\_APP\_API\_URL) must point to this server’s URL. The API is not meant for direct user access.

\* Ensure CORS (Cross-Origin Resource Sharing) is correctly configured on the backend server to allow requests from the frontend’s domain.

For any issues, check the terminal logs for error messages. Common issues include incorrect environment variables or services not running on the correct ports.

7. API Documentation

This document provides a detailed overview of the RESTful API for the CookBook application, a virtual kitchen assistant. The API is designed to manage users, recipes (referred to as “projects”), and real-time chat functionality.

All API endpoints are prefixed with /api.

1. User Endpoints

Manage user registration, authentication, and profile information.

\* POST /api/users/register

\* Description: Creates a new user account.

\* Request Body:

{

“username”: “string”,

“email”: “string (email format)”,

“password”: “string (min. 8 characters)”

}

* Success Response (201 Created):

{

“message”: “User registered successfully.”,

“user”: {

“id”: “string”,

“username”: “string”,

“email”: “string”

}

}

* Error Response (400 Bad Request):

{

“message”: “Email already in use.”

}

\* POST /api/users/login

\* Description: Authenticates a user and returns an authentication token.

\* Request Body:

{

“email”: “string”,

“password”: “string”

}

* Success Response (200 OK):

{

“message”: “Login successful.”,

“token”: “JWT\_token\_string”

}

* Error Response (401 Unauthorized):

{

“message”: “Invalid email or password.”

}

1. Projects (Recipes) Endpoints

Handle the creation, retrieval, updating, and deletion of recipes.

\* GET /api/projects

\* Description: Retrieves a list of all public recipes.

\* Success Response (200 OK):

[

{

“id”: “string”,

“title”: “string”,

“authorId”: “string”,

“description”: “string”,

“ingredients”: [“string”],

“instructions”: [“string”],

“rating”: “number”

}

]

\* POST /api/projects

\* Description: Creates a new recipe. Requires authentication.

\* Request Body:

{

“title”: “string”,

“description”: “string”,

“ingredients”: [“string”],

“instructions”: [“string”]

}

\* Success Response (201 Created): Returns the newly created recipe object.

\* GET /api/projects/{projectId}

\* Description: Retrieves a single recipe by its ID.

\* Success Response (200 OK): Returns a single recipe object.

\* PUT /api/projects/{projectId}

\* Description: Updates a specific recipe. Requires authentication and ownership.

\* Request Body: Same as the POST request, but only include fields to be updated.

\* Success Response (200 OK): Returns the updated recipe object.

\* DELETE /api/projects/{projectId}

\* Description: Deletes a recipe. Requires authentication and ownership.

\* Success Response (204 No Content):

3. Chats Endpoints

Manages the real-time chat system for recipe discussions and collaboration.

\* POST /api/chats/start

\* Description: Starts a new chat session for a specific recipe.

\* Request Body:

{

“projectId”: “string”,

“participants”: [“string”, “string”]

}

* Success Response (201 Created):

{

“chatId”: “string”,

“projectId”: “string”,

“messages”: []

}

\* GET /api/chats/{chatId}/messages

\* Description: Retrieves the message history for a given chat session.

\* Success Response (200 OK):

[

{

“senderId”: “string”,

“text”: “string”,

“timestamp”: “string (ISO 8601 format)”

}

]

\* POST /api/chats/{chatId}/messages

\* Description: Sends a new message to a chat session. This is typically handled via WebSockets for real-time functionality.

\* Request Body:

{

“senderId”: “string”,

“text”: “string”

}

* Success Response (201 Created): Returns the new message object.

8 Authentication & Security

​ The application will use JWT (JSON Web Token) authentication to ensure secure user access.

​JWT-based authentication will be implemented for all user logins, providing a stateless and secure method for verifying user identity. Upon successful login, the server will issue a JWT that the client will store and send with subsequent requests.

​Middleware will be used to protect private routes. This middleware will intercept requests to protected endpoints, verify the presence and validity of the JWT, and grant or deny access accordingly. This ensures that only authenticated and authorized users can access sensitive information or perform specific actions.

9 User Interface

The application’s user interface will be designed to be intuitive and easy to navigate, with distinct pages for different user roles and functions.

​Landing Page: This will be the public-facing entry point. It will feature a brief introduction to the platform, a search bar for discovering public recipes, and clear calls-to-action for user sign-up and login.

​Freelancer Dashboard: Once logged in, content creators will be directed here. It will display a list of assigned recipe projects, their current status, and quick links to edit or submit their work. It will also include a section for managing their personal profile and tracking their progress.

​Admin Panel: This restricted area is for administrators. It will provide tools to manage the entire platform, including user accounts, pending recipe submissions, content moderation, and platform analytics.

​Project Details Page: This page will display the full details of a single recipe project. It will include all necessary fields such as ingredients, step-by-step instructions, preparation time, and any collaborative notes.

10. Testing & Quality Assurance

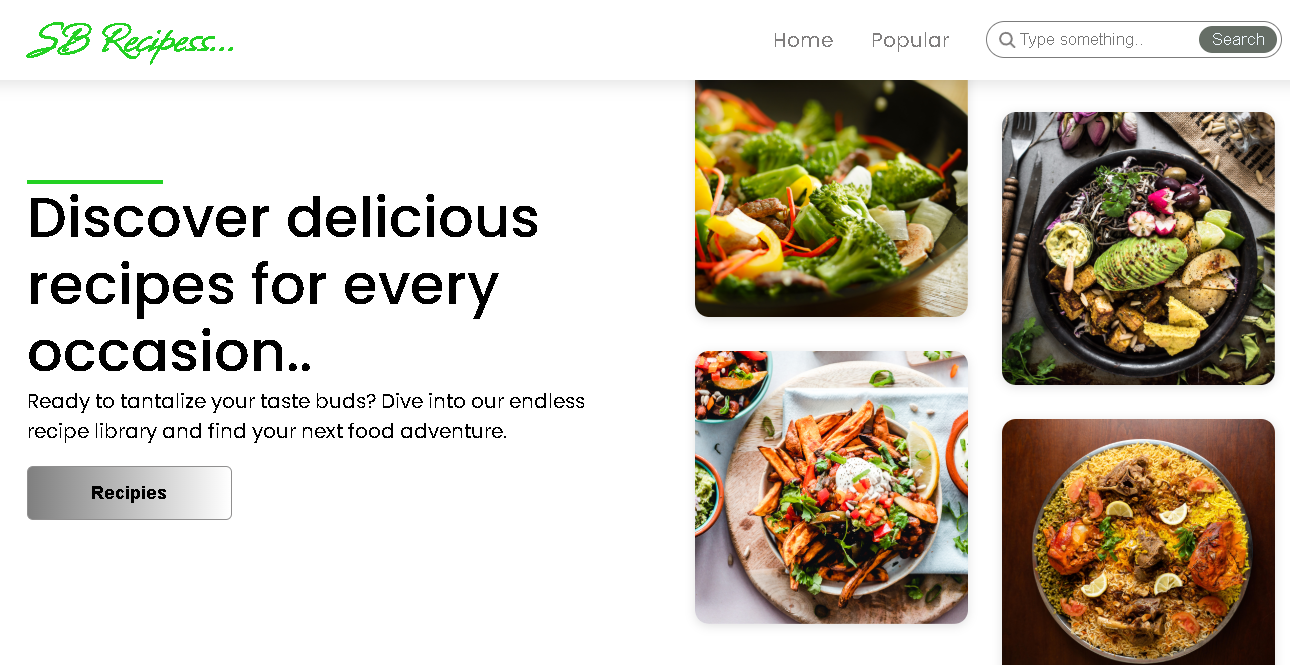
A strategic approach to testing will be followed to ensure the application’s reliability and stability.

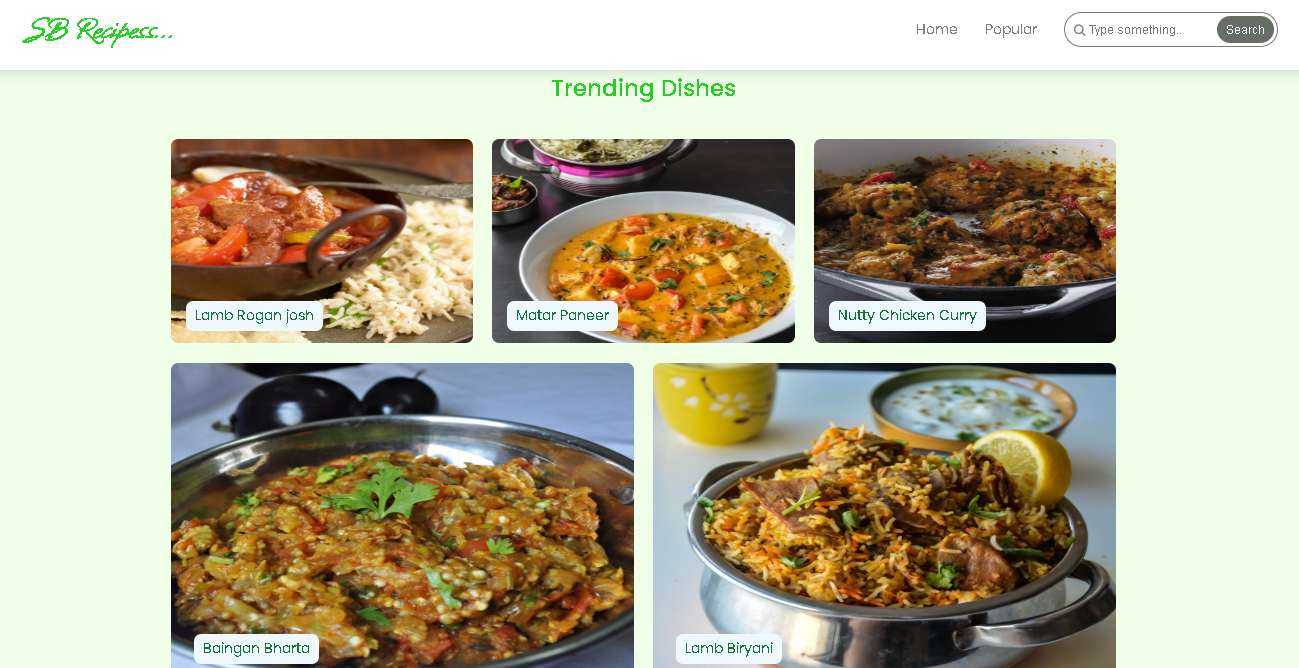
​Manual testing will be conducted at major development milestones to verify core functionality and user flows. This includes testing the entire user journey from registration to recipe submission.

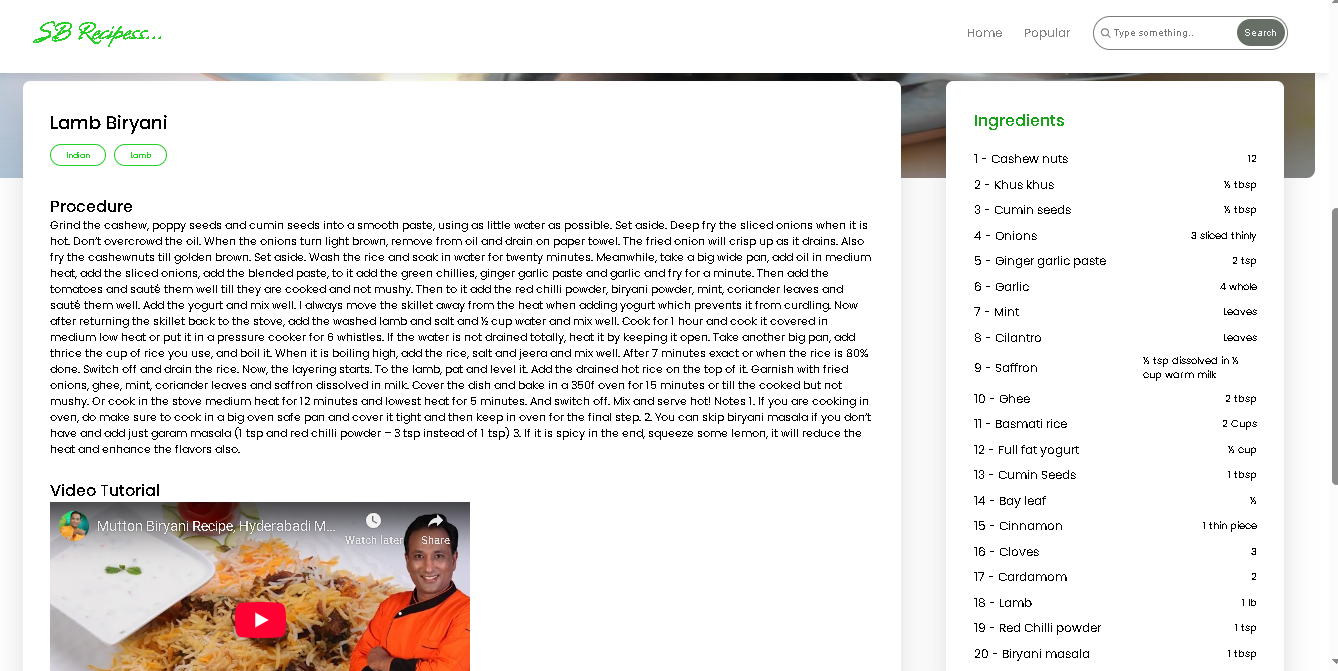
​Postman will be used for API testing to ensure all backend endpoints are functioning correctly and returning the expected data. This allows for early detection of issues with the server-side logic.

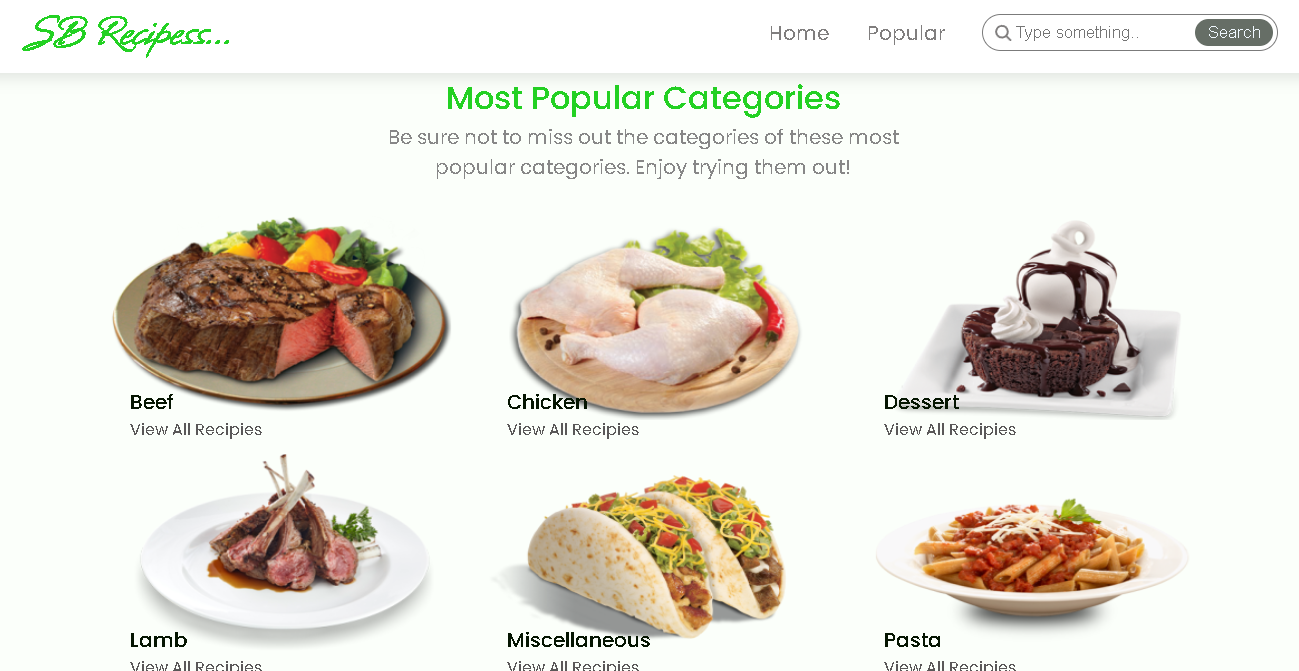
​Chrome Dev Tools will be utilized for front-end debugging. We will use it to inspect the UI, monitor network requests, and debug client-side JavaScript issues, ensuring a smooth user experience.

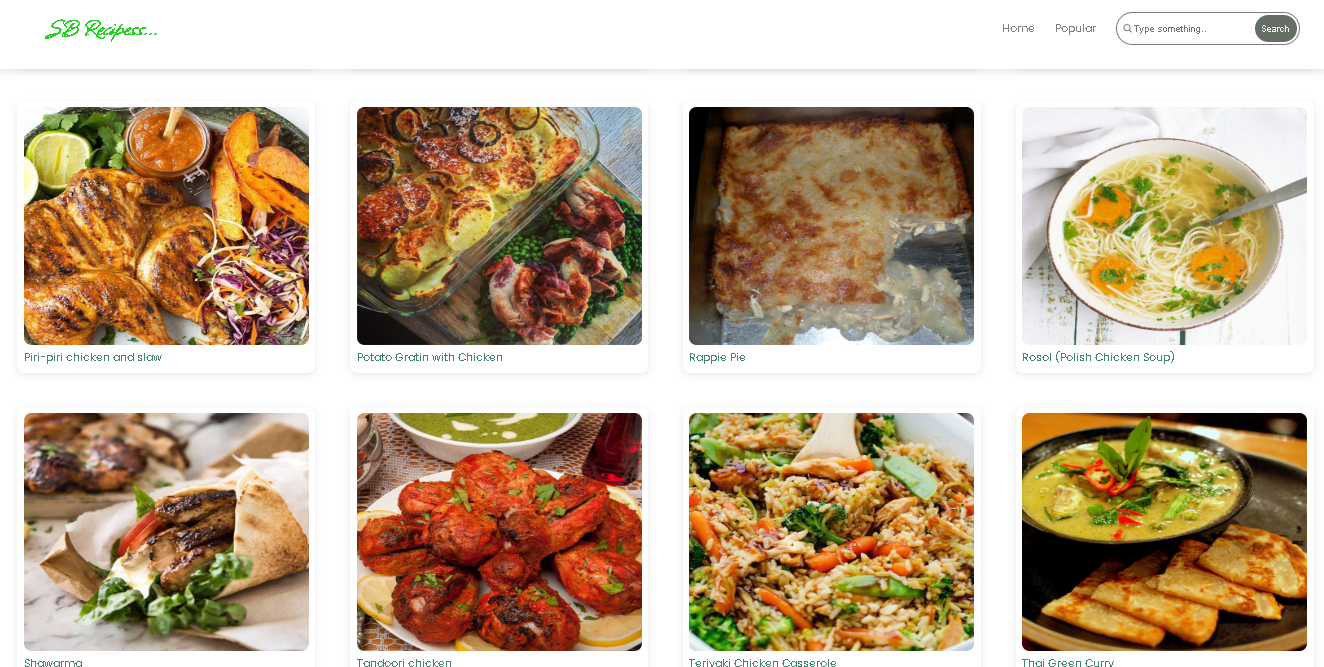
**11 Screenshot And Demo**

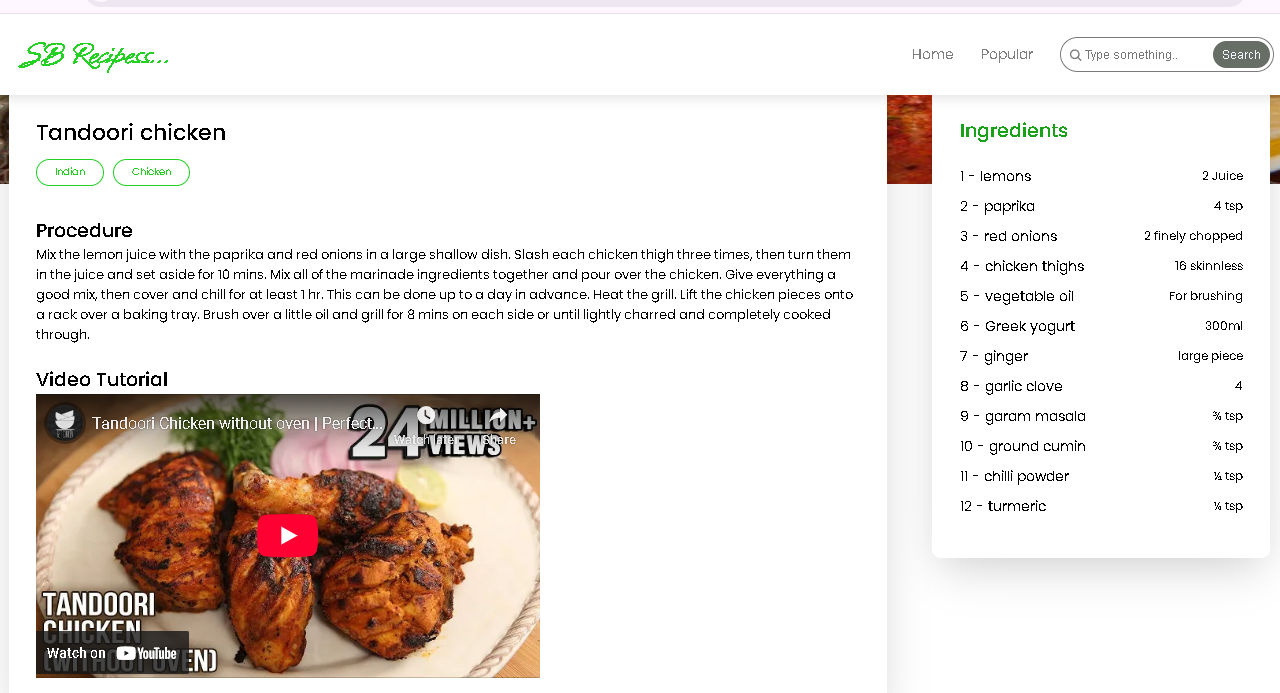
****

****

****

****

****

****

**12 Known Issues**

\* Voice Recognition Inaccuracy: The voice command feature can occasionally misinterpret spoken instructions, especially in noisy environments or when a user has a strong accent. This can lead to frustration when trying to progress through a recipe hands-free. We are currently collecting more voice data to improve the underlying model.

\* Ingredient Substitution Limitations: While the app can suggest basic ingredient substitutions, it lacks a comprehensive understanding of flavor profiles and chemical reactions. This may result in suboptimal or unexpected outcomes for more complex recipes.

\* Inconsistent Recipe Time Estimates: The estimated cooking times for recipes can sometimes be inaccurate, as they do not account for variations in stove power, oven calibration, or user skill level. This can cause timing issues during meal preparation.

\* No Offline Mode: The application requires a consistent internet connection to access the recipe database and AI features. An intermittent or lost connection in the kitchen can interrupt the cooking process.

\* Lack of Integration with Smart Kitchen Appliances: “Cook Book” does not currently connect with or control smart kitchen devices like ovens, scales, or timers. This prevents a truly seamless and automated cooking experience.

\* Minor UI/UX Bugs: Users have reported minor visual glitches and navigation issues, such as buttons that are unresponsive after a certain sequence of actions or text that is cut off on smaller screens. These are being cataloged for a future maintenance release.

**13 Future Enhancements**

\* Advanced Voice-Activated Guidance: We plan to upgrade the voice model to support more natural, conversational language and to understand contextual commands (e.g., “Go back three steps,” “What’s the next ingredient?”).

\* Pantry and Inventory Management: A new feature will allow users to input their pantry items and expiration dates. The app will then suggest recipes based on available ingredients and generate smart shopping lists to reduce food waste.

\* Personalized Nutritional Tracking: Integrate a nutrition database to provide detailed information for each recipe, including calories, macronutrients, and common allergens. The app will also be able to tailor meal plans to specific dietary restrictions (e.g., keto, vegan, gluten-free).

\* Visual Recipe Guidance: Incorporate short video clips or animated GIFs for each step of a recipe to provide clear visual instructions, particularly for complex techniques.

\* Meal Planning and Calendar Integration: A weekly meal planner feature will allow users to schedule recipes and automatically create a shopping list for the entire week. This feature will also sync with the user’s personal calendar.

\* Community and Social Features: A “Social Spice” module will enable users to create their own digital cookbooks, share them with friends, and participate in community cooking challenges. This will foster a vibrant community and encourage user engagement.

\* Smart Appliance Integration (Phase 2): In a later release, we will integrate with common smart kitchen appliances to allow the app to preheat ovens, set timers, and receive real-time temperature updates.