**Project Documentation**

CookBook: Your Virtual Kitchen Assistant

# 1. Introduction

* **Project Title:** **CookBook; Your Virtual Kitchen Assistant**
* **Team ID : NM2025TMID36627**
* **Team Leader: R.Rakesh/** [rvanitha208@gmail.com](mailto:rvanitha208@gmail.com)
* **Team Members:** **K.Ravi Ragavendra** / [ravivirat2111@gmail.com](mailto:ravivirat2111@gmail.com)

**K.Rishikesh**/ [rishikesh7383@gmail.com](mailto:rishikesh7383@gmail.com)

**R.Dhivya Sri**/ [ss9838701@gmail.com](mailto:ss9838701@gmail.com)

# 2. Project Overview

# Purpose:

# Cookbook is a smart virtual kitchen assistant designed to help users explore, organize, and manage recipes while offering personalized recommendations. It allows users to discover new dishes, plan meals, and maintain digital cookbooks with real-time updates.

# Features:

# \* Recipe search and recommendations

# \* Meal planner and shopping list generator

# \* Personalized cookbook management

# \* User feedback and rating system

# \* Admin control panel for recipe moderation

# 3. Architecture

# Frontend: React.js with Bootstrap and Material UI

# Backend: Node.js and Express.js (API and server logic)

# Database: MongoDB (user data, recipes, reviews, shopping lists, etc.)

# 4. Setup Instructions

# Prerequisites:

**\*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.Download: https://nodejs.org/en/download/Installation instructions: <https://nodejs.org/en/download/package-manager/>

**\*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.Version Control: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

**\*Git**: Download and installation instructions can be found at: https://git-scm.com/downloads Development Environment: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

# 5. Floder Structure:

# Cookbook/

# |-- client/ # React frontend

# | |-- components/

# | |-- pages/

# |

# |-- server/ # Node.js backend

# | |-- routes/

# | |-- models/

# | |-- controllers/

# 6. Running the Application

To run the Cookbook: Your Virtual Kitchen Assistant project, you need to start both the frontend (React.js) and backend (Node.js + Express.js) servers.

**1. Frontend (React.js):**

# Navigate to client folder

cd client

# Start the React development server

npm start

By default, the frontend runs on: http://localhost:3000

**2. Backend(Node.js + Express.js):**

# Navigate to server folder

cd server

# Start the Node.js backend server

npm start

By default, the backend runs on: http://localhost:5000 (or the port defined in .env)

**3. Accessing the Application:**

Open your browser and go to: http://localhost:3000

The frontend will communicate with the backend APIs (http://localhost:5000).

* **Access:** Visit <http://localhost:3000>

# 7. API Documentation

**User:**

POST /api/user/register

POST /api/user/login

**Recipes:**

POST /api/recipes/create

GET /api/recipes/:id

**Meal Planner:**

POST /api/mealplan

**Shopping List:**

GET /api/shoppinglist/:userId

# 8. Authentication

# 8.1 Authentication Method:

JWT (JSON Web Token) Based Authentication

Password Hashing: User passwords are hashed using bcrypt before being stored in MongoDB.

**8.2 Workflow:**

1. User Registration (POST /api/user/register)
2. A new user signs up with name, email, and password.

Password is hashed and stored in MongoDB.

**2. User Login (POST /api/user/login):**

User provides email and password.

Server verifies credentials.

If valid, server generates a JWT token containing user ID and role.

**3. Token Handling:**

The token is sent to the client.

It is stored securely on the client side (in localStorage or as an httpOnly cookie).

**4. Accessing Protected Routes:**

For requests like saving a recipe, generating a shopping list, or creating a meal plan, the client must send the token in the Authorization header:

Authorization: Bearer <token>

A middleware function on the server verifies the token before granting access.

**5. Role-Based Access**

User: Can create meal plans, manage cookbooks, and write reviews.

Admin: Can manage users, approve or reject recipes, and moderate reviews.

**8.3 Middleware Example:**

const jwt = require("jsonwebtoken");

function authMiddleware(req, res, next) {

const token = req.header("Authorization")?.replace("Bearer ", "");

if (!token) return res.status(401).json({ error: "Access denied. No token provided." });

try {

const verified = jwt.verify(token, process.env.JWT\_SECRET);

req.user = verified; // attach user info to request

next();

} catch (err) {

res.status(400).json({ error: "Invalid token" });

}

}

module.exports = authMiddleware;

**8.4 Security Features:**

JWT Expiry: Tokens expire after a set time (e.g., 1h).

Password Hashing: All passwords are encrypted before storage.

Protected Routes: Only logged-in users can access recipe saving, shopping lists, and meal plans.

Admin Privileges: Special endpoints are restricted to admin tokens.

# 9. User Interface

**\*Landing Page**

Highlights featured and trending recipes.

Search bar for quick recipe lookup (by name, ingredient, or cuisine).

Call-to-action buttons for Sign Up / Login.

Category sections (Breakfast, Lunch, Dinner, Desserts, Healthy, etc.).

**\* User Dashboard**

Personalized homepage after login.

Displays:

Saved Recipes (user’s favorites).

Meal Planner (weekly calendar view).

Shopping List (auto-generated based on meal plan).

Quick links to “Add New Recipe” or “Plan Meals.”

\*Recipe Details Page

Shows complete recipe information:

Ingredients list

Step-by-step instructions

Cooking time, servings, and nutrition info

Buttons: Save to Cookbook, Add to Meal Plan, Generate Shopping List

Section for user ratings, reviews, and comments.

**\* Shopping List Page**

Auto-generated from selected recipes or meal plan.

Users can check off purchased items.

Editable (add/remove ingredients).

**\* Meal Planner**

Weekly calendar with drag-and-drop recipe placement.

Option to auto-generate a shopping list from planned meals.

**\* Admin Panel**

Available only for admin users.Features:

Manage recipes (approve/reject user submissions).

Manage users (activate/deactivate accounts)

Moderate reviews/feedback.

# 10. Testing

**\* Manual Testing**

Tested each feature step by step (registration, login, recipe search, meal planning, shopping list).

Verified responsiveness on desktop, tablet, and mobile devices.

**\* Functional Testing**

Checked whether each module works as intended:

Can users register/login with valid credentials?

Can recipes be searched, saved, and reviewed?

Does the shopping list update automatically after adding recipes?

Are admin features (approve/reject recipes) restricted only to admins?

**\* API Testing**

Used Postman to test all backend endpoints.

Verified responses for user authentication, recipe CRUD, and meal planner APIs.

Ensured unauthorized requests without JWT tokens were blocked.

**\* Integration Testing**

Checked interaction between frontend (React.js) and backend (Node.js + Express.js).

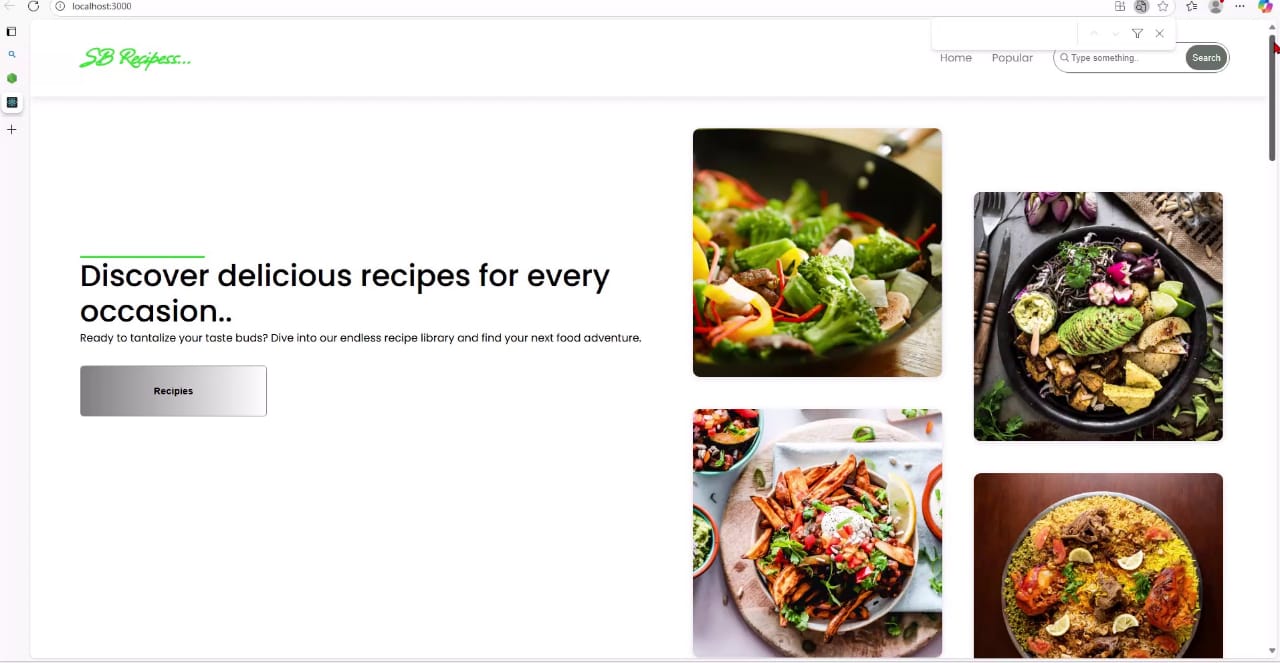
Verified that user actions on the UI triggered the correct API calls and database updates in MongoDB.

**\* Performance Testing**

Simulated multiple users searching and saving recipes.

Checked system response time and stability under load.

**11. Screenshots or Demo**

****

**Or**

**Demo link**

<https://drive.google.com/file/d/14jJyma_aWuAKpsgijBXf7c5Yw6B-lt_h/view?usp=drivesdk>

**12. Known Issues**

Limited recipe dataset in initial version

Some features not optimized for mobile

# 13. Future Enhancements

AI-powered recipe recommendations

Voice assistant for cooking guidance

Multilingual recipe support

Integration with IoT smart kitchen devices