PROJECT DOCUMENTATION

COOKBOOK: YOUR VIRTUAL KITCHEN ASSISTANT



CookBook: Your Virtual Kitchen Assistant

Team ID: NM2025TMID36534

Team Leader:

* Gayathri .S-24csgayathri029@gmail.com

Team Member:

* Durga Devi .B- 24csdurgadevib027@gmail.com
* Durga Devi .K- 24csdurgadevi028@gmail.com
* Durga .B- 24csdurga026@gmail.com

**1. Introduction**

This document descry.bes the Cookbook project in a clear step-by-step format: goals, features, architecture, data models, APIs, UI flows, implementation milestones, testing, deployment, and deliverables. It is written so your team can follow the sequence of tasks from planning to launch and future improvements.

**2. Project Overview**

**Purpose**

Cookbook is a web/mobile application where users can create, share, discover, and manage recipes. It supports rich recipe content (ingredients, steps, photos, nutrition), user interaction (comments, ratings, favorites), planning (meal planner & shopping lists), and admin moderation.

**High-level Goals**

* Make recipe creation and discovery simple and delightful.
* Support social feedback (comments, ratings) and quality control via admin moderation.
* Provide tools that help users plan meals and build shopping lists from recipes.
* Support media (images, short videos) and structured ingredient/nutritional data.

**Core Features (mapped to the structure you provided)**

* **Recipe posting & management** (maps to “project posting”): create, edit, draft, publish recipes with images and structured ingredient lists.
* **Search & discovery (no bidding required)**: curated feeds, categories, trending recipes, tags, and advanced filters (dietary restrictions, time, difficulty).
* **Real-time collaboration & sharing (maps to “secure chat system” concept)**: private recipe sharing, comments in real-time, optional group cook sessions (optional Socket.io feature).
* **Feedback & review system**: star ratings, upvoting, comments, and moderation tools.
* **Admin control panel**: user management, recipe moderation, content analytics, and site settings.
* **User accounts & profiles**: personal recipe books, favorites, following other users, basic social features.

## 3. Architecture (use your provided stack)

Include a short diagram (you can paste a block diagram image) and describe each layer.

**Text to include**

* **Frontend:** React.js using Bootstrap + Material UI for components. Responsible for UI, client routing, form validation, and calling backend APIs.
* **Backend:** Node.js + Express — REST API endpoints, auth, business logic, file handling (images), validation.
* **Database:** MongoDB — stores users, recipes, comments, favourites, shopping lists.

**4. Setup instructions**

**Prerequisites**

Install these on your machine before starting (recommended minimums):

* Node.js (LTS, e.g. v18+)
* npm (comes with Node)
* MongoDB (local) **or** a MongoDB Atlas cluster
* Git
* Visual Studio Code (or your editor of choice)

**Installation Steps**

Follow these steps to set up and run the **Cookbook Project** on your local machine:

1. **Clone the Repository**  
   Open a terminal and run the following command to clone the project from GitHub:
2. git clone <your-repository-link>
3. **Navigate to the Client Folder**  
   Move into the client directory where the frontend code is located:
4. cd client
5. **Install Client Dependencies**  
   Install all required frontend packages using npm:
6. npm install
7. **Navigate to the Server Folder**  
   Go to the server directory where the backend code is located:
8. cd ../server
9. **Install Server Dependencies**  
   Install all required backend packages using npm:
10. npm install
11. **Start the Client (Frontend)**  
    From inside the **client** folder, run:
12. npm start

This will start the frontend on your local development server

1. **Start the Server (Backend)**  
   From inside the **server** folder, run:
2. npm start

This will start the backend server

1. **Access the Application**  
   Open your browser and go to:
2. http://localhost:3000

The Cookbook app should now be running successful

**5. Folder Structure**

Cookbook app/

│── src/

│ ├── App.js → Main application logic (state + functions + layout)

│ ├── components/ → Reusable components folder

│ │ ├── cook recieps Form.js → Form to add new items (name + picture)

│ │ ├── foodList.js → Shows list of all inventory items

│ │ └──foodstyle Item.js → Handles single item (pictures+ food style)

│ ├── index.js → Entry point of React app (renders <App />)

│ └── App.css → Styling for your app

**6. Running the Application**

# 1) Prerequisties

* Node.js (v16+ recommended)
* npm or yarn
* MongoDB running locally (or a MongoDB Atlas connection string)
* Git (to clone if needed)
* Optional: nodemon (dev), concurrently (to run both at once), Postman / Insomnia

# 2) Clone & install

# clone (if not already)

git clone <your-repo-url>

cd <your-repo-folder>

# install server deps

cd server

npm install

# install client deps

cd ../client

npm install

# go back to repo root

cd ..

# 3) Environment variables (example .env for server)

Create server/.env:

PORT=5000

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

JWT\_SECRET=your\_jwt\_secret\_here

# any other keys (e.g. CLOUDINARY\_URL, EMAIL creds)...

If using MongoDB Atlas replace MONGO\_URI with the connection string.

Frontend .env (create client/.env — React only exposes REACT\_APP\_ prefixed vars):

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

# REACT\_APP\_MAP\_KEY=...

# 4) Start MongoDB

* **Local MongoDB**:
  + Linux/macOS: sudo systemctl start mongod or mongod --config /etc/mongod.conf
  + Windows: start MongoDB service or run mongod from the MongoDB installation bin folder
* **Atlas**: ensure MONGO\_URI is the Atlas connection string and IP whitelist is configured.

# 5) Run in development (separate terminals)

Open two terminals.

Terminal A — Backend:

cd server

# optional: install nodemon globally: npm i -g nodemon

npm run dev # typical script runs nodemon (see package.json). otherwise: node index.js

# or

nodemon index.js

Terminal B — Frontend:

cd client

npm start

# opens http://localhost:3000 by default

**Typical default ports**

* Backend: http://localhost:5000
* Frontend: <http://localhost:3000>

**7. API Documentation**

Users

**GET /v1/users/:id** — Get a public user profile (name, avatar, bio, projectCount)  
**GET /v1/users/me** — Get current user profile (protected)  
**PUT /v1/users/me** — Update profile (protected)  
**DELETE /v1/users/me** — Delete account (protected)

Projects

**GET /v1/projects** — List all projects (public)  
Query params: page, limit, q (search by title/description), sort.

**POST /v1/projects** — Create new project (protected)  
Example body:

{

"title": "Recipe Collaboration Platform",

"description": "A web app where users share and collaborate on recipes",

"tags": ["nodejs", "react"],

"visibility": "public"

}

**GET /v1/projects/:id** — Get project details (includes owner, members, createdAt)  
**PUT /v1/projects/:id** — Update project (protected — only owner/admin)  
**DELETE /v1/projects/:id** — Delete project (protected)

**POST /v1/projects/:id/members** — Add member to project (protected)  
**DELETE /v1/projects/:id/members/:userId** — Remove member from project

Chats

**GET /v1/projects/:id/chats** — Get chat messages for a project (protected)  
Query params: page, limit for pagination.

**POST /v1/projects/:id/chats** — Send a new message (protected)  
Body:

{

"text": "Hello team, let's start working on the recipe!"

}

**DELETE /v1/chats/:chatId** — Delete a message (protected — only sender or admin)

## 8.Authentication

* **JWT-based Authentication**
  + User logs in → server issues a **JWT**.
  + JWT is sent with each request in header:
  + Authorization: Bearer <token>
* **Middleware Protection**
  + Middleware checks the token before giving access to **private routes**.
  + Invalid/missing token → 401 Unauthorized.
* **Access & Refresh Tokens**
  + **Access Token** → short-lived (for API requests).
  + **Refresh Token** → longer-lived (to get new access tokens).
* **Security**
  + Passwords hashed with **bcrypt**.
  + All requests use **HTTPS**.
  + Tokens signed with a **secret key**.

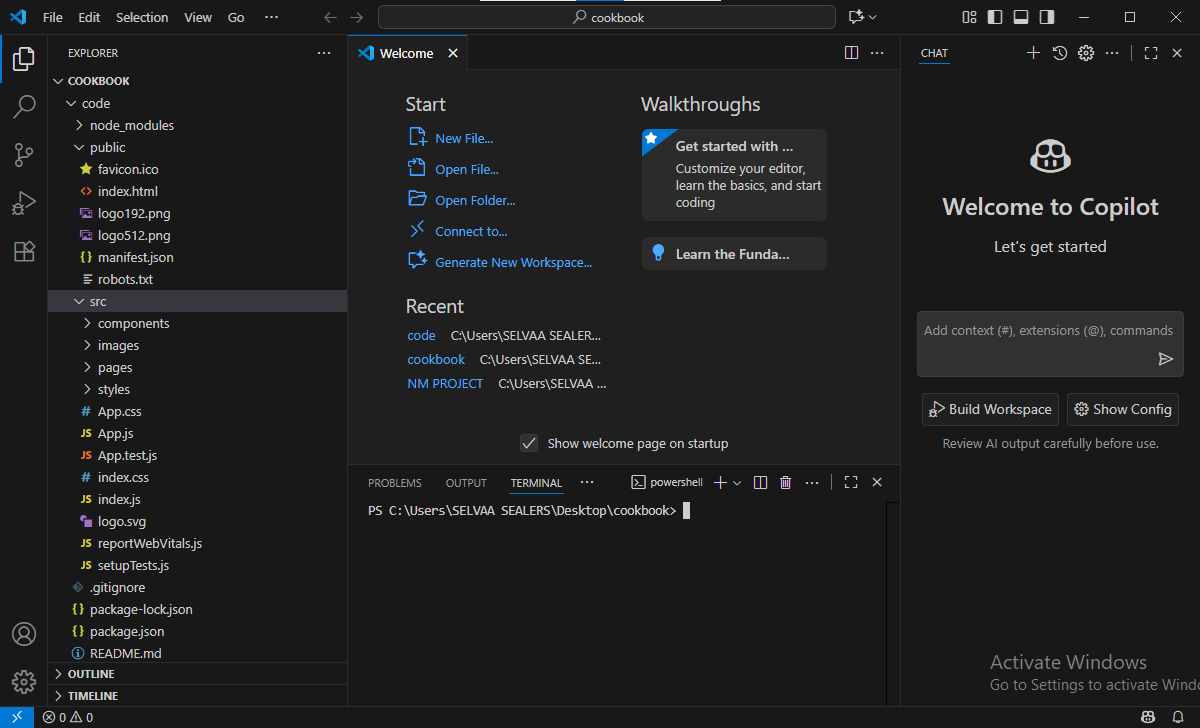
## 9. User Interface

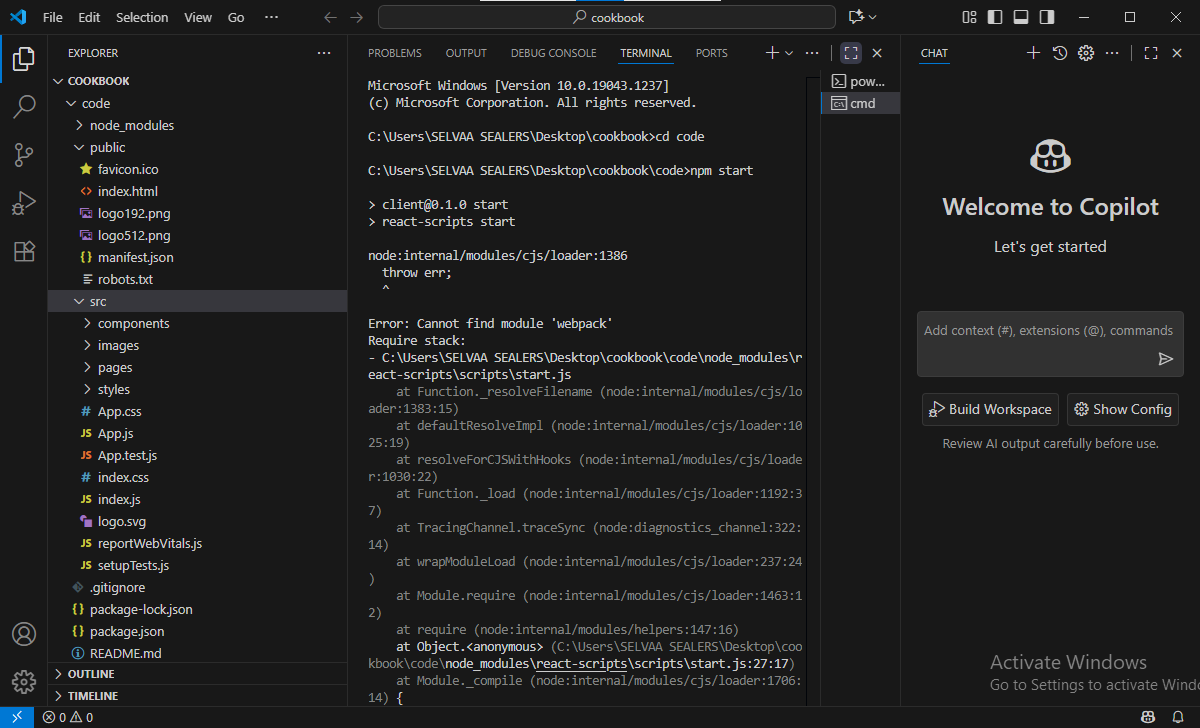
* **Landing Page**
  + Intro, features, and popular projects.
  + Options: **Login / Sign Up** and **Post a Project**.
* **Freelancer Dashboard**
  + Shows **bids, recommended projects, messages, and earnings**.
  + Quick actions: **Apply, View Stats, Manage Profile**.
* **Admin Panel**
  + Manage **users, projects, disputes, and settings**.
  + Tools: **tables, filters, analytics, approve/suspend actions**.
* **Project Details Page**
  + Displays **title, budget, skills, description, client info**.
  + Actions: **Freelancers → Bid/Edit Bid** | **Clients → Manage Proposals & Milestones**.

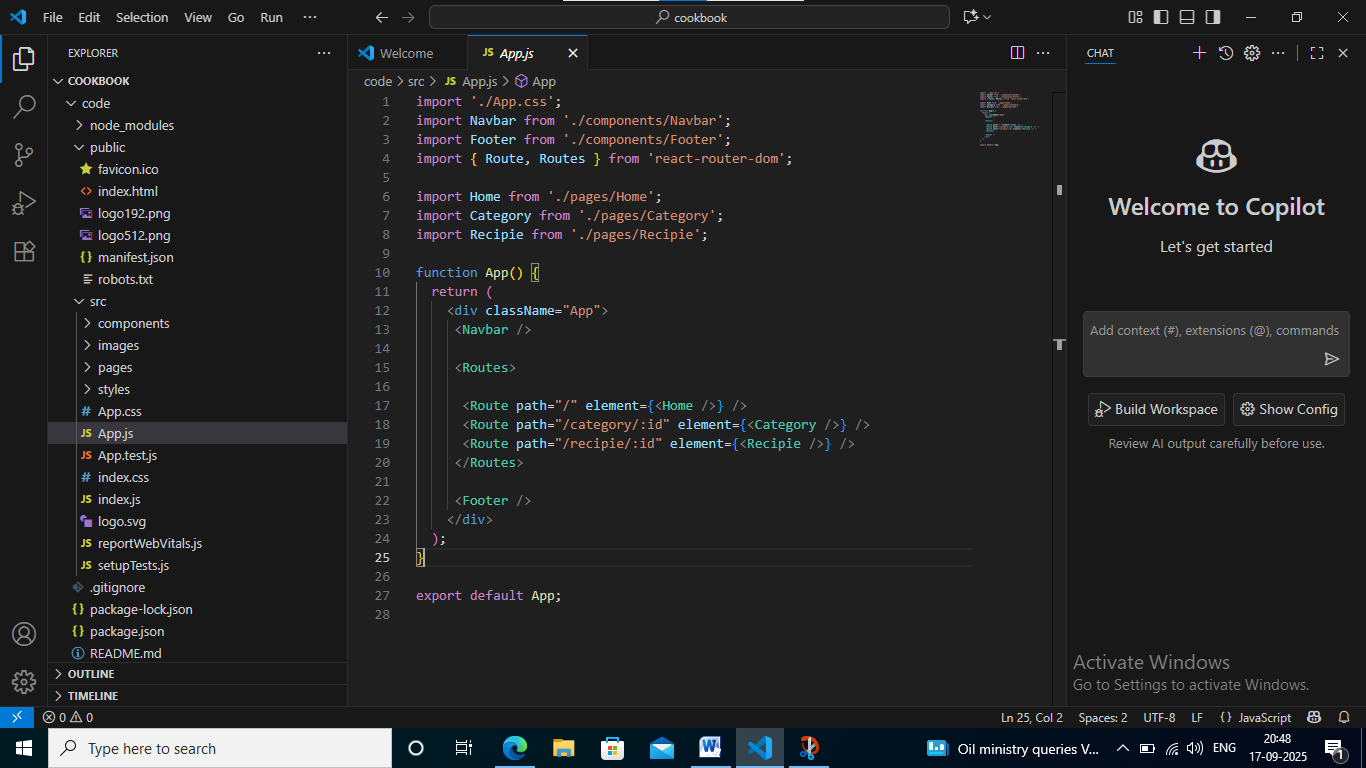
## 10. Testing

* **Manual Testing (Milestones)**
  + After each milestone, test completed features step by step.
  + Check main flows: Login → Post Project → Bid → Logout.
  + Report bugs with steps + screenshots.
* **Tools**
  + **Postman** → test APIs (status codes, responses, auth headers).
  + **Chrome DevTools** → check UI (console err
  + ors, responsive view, network requests).
* **Acceptance**
  + All smoke tests pass.
  + No critical bugs open.
  + ch milestone, test completed features step by step.
  + Check main flows: Login → Post Project → Bid → Logout.
  + Report bugs with steps + screenshots.
* **Tools**
  + **Postman** → test APIs (status codes, responses, auth headers).
  + **Chrome DevTools** → check UI (console errors, responsive view, network requests).
* **Acceance**
  + All smoke tests pass.
  + No critical bugs open.

**11.Screenshots or demo**







## 12. Known Issues

* **Authentication Errors**  
  • Token expiration may cause users to be logged out unexpectedly.  
  • JWT refresh mechanism not fully implemented.
* **UI/UX Limitations**  
  • Some pages are not fully responsive on smaller devices.  
  • Limited accessibility support (e.g., screen reader compatibility).
* **Performance Constraints**  
  • Slow load times for pages with large datasets (e.g., project listings).  
  • Search and filter functions may lag under heavy load.
* **Chat System**  
  • Occasional message delivery delays.  
  • No offline message caching.
* **Error Handling**  
  • Generic error messages displayed instead of descriptive ones.  
  • Inconsistent validation feedback across forms.
* **Deployment Issues**  
  • Environment configuration varies between local and production.  
  • Limited logging for debugging in production mode.
* **Testing Gaps**  
  • Automated testing not fully integrated.  
  • Manual testing needed for edge cases and high-traffic scenarios.

## 13. Future Enhancements

* Token refresh & MFA for authentication
* Fully responsive UI + dark mode
* Faster performance with caching & optimized queries
* Real-time chat notifications + offline support
* Smart project recommendations & advanced search
* Analytics & role-based access in admin panel
* Automated testing + CI/CD pipeline
* Stronger security with scans & encryption