**CookBook: Your Virtual Kitchen Assistant**

**1.Introduction:**

**1.1 project title:**

CookBook: Your Virtual Kitchen Assistant

**1.2 Team Id:**

NM2025TMID37126

**1.3Team leader:**

Mahalakshmi.R

[tamilmudharpavalar@gmail.com](mailto:tamilmudharpavalar@gmail.com)

1.4 Team members:

Yogalakshmi.R

[yogarajkumar@gmail.com](mailto:yogarajkumar@gmail.com)

Mavitha.R

[Mavithar22012007@gmail.com](mailto:Mavithar22012007@gmail.com)

Suriyapriya.

**2.Project Overview**:

**The platform streamlines the freelance job lifecycle by offering features such as:**

* Project posting and discovery: Clients post projects and freelancers search or filter projects matching their expertise.
* Proposal submission and bidding: Freelancers submit tailored proposals outlining their approach, skills, and pricing to win projects.
* Contract and agreement management: Formalizes work arrangements between clients and freelancers for clear expectations on roles and payments.
* Real-time communication: Supports instant messaging for negotiations, clarifications, and collaboration.
* Secure payment processing: Ensures safe transactions, protecting both clients and freelancers.
* Dispute resolution mechanisms to manage conflicts if they arise.

**Purpose**:

SB Works is a freelance platform designed to connect clients and freelancers by facilitating project postings, bidding by freelancers, and real-time communication between parties. This platform aims to streamline the process of finding freelance work and managing projects by offering features that allow clients to post their project requirements, receive bids or proposals from interested freelancers, and maintain direct, real-time communication throughout the project's lifecycle. This approach helps clients find the best suited freelancers for their projects, encourages competitive bidding, and supports efficient collaboration and project management. SB Works typifies freelance platforms that bridge the gap between talent supply and demand with a focus on ease of use for both clients and freelancers.

**2.The Features :**

Project Posting and Bidding: Clients can post detailed project descriptions, including requirements, budgets, and deadlines. Freelancers can browse these projects and submit competitive bids or proposals.

* **Secure Chat System**: Allows real-time, secure communication between clients and freelancers for negotiation, clarifications, and collaboration.
* **Feedback and Review System**: After project completion, both clients and freelancers can leave ratings and reviews to build trust and reputation.
* **Admin Control Panel**: An interface for platform administrators to manage users, projects, disputes, payments, and platform operations.

**Additional typical features often present in such platforms are:**

* **Profile Creation**: Users create detailed profiles showcasing skills, work history, and portfolios to attract suitable projects or freelancers.
* **Contract and Agreement** Management: Formalizes work agreements with clear roles, expectations, and payment terms.
* **Dispute Resolution**: Mechanisms to handle conflicts through mediation or arbitration.
* **Secure Payment Processing**: Ensures safe and reliable transactions between clients and freelancers.
* **Search and Discovery Tools**: Advanced filters and search options to find projects or freelancers efficiently.

**The Project posting and bidding**:

* Clients create detailed project postings including description, deadlines, and budget.
* Freelancers browse available projects and submit bids with proposed price, delivery timeline, and a custom proposal.
* Clients review multiple bids, consider freelancers’ profiles, reviews, and portfolios, and award the project to the best match.
* The system may use escrow to hold payments securely until milestone or project completion.
* Bidding encourages competitive pricing and fast delivery offers.
* Real-time bidding notifications and updates improve responsiveness during the bidding phase.

**The Secure chat system**:

* Private, real-time messaging between clients and freelancers.
* Communication for discussing project scope, clarifications, and negotiations.
* A secure environment ensuring confidentiality of conversations.
* Easy sharing of files, links, and multimedia to assist project discussions.

**The Feedback and review system:**

* A reliable rating system where clients and freelancers can leave honest feedback and reviews after project completion.
* Reviews provide valuable information about work quality, professionalism, and reliability to future clients and freelancers.
* Both parties contribute to a fair marketplace by sharing experiences, which fosters trust and accountability.
* Some platforms allow clients to edit reviews within limited periods and provide mechanisms to detect and prevent fake or manipulated reviews.
* This system helps freelancers improve their services and helps clients make informed hiring decisions.

**The Admin control panel**:

A dashboard for platform administrators to manage users, projects, reviews, disputes, and payments.

* Controls to approve or reject user-submitted reviews to maintain authenticity.
* Monitoring tools for detecting fraudulent behavior and protecting platform integrity.
* Administrative privileges to moderate content, handle complaints, and ensure adherence to platform policies.
* Tools to oversee the smooth functioning of the marketplace, enable reports and analytics, and maintain overall security.

Together, these features ensure quality control, user trust, and efficient management of the freelance platform environment.

Related

Draft admin panel controls for managing reviews and flags

Define review submission workflow and visibility rules

Design database schema for reviews, responses, and audits

Specify moderation tools and automated fraud detection

Create UI components for review display and analytics.

**3. Architecture:**

* **Code Organization and Design**: Creating a coherent design system or UI component library, commonly using React.js with styling frameworks like Bootstrap and Material UI. This ensures consistency, reusability, and maintainability of frontend code.
* **Process and Tooling**: Implementing modern development workflows including version control, task runners, linters, automated testing, and documentation tools. This improves developer efficiency and reduces errors.
* **Testing and Stability**: Establishing robust frontend testing practices to create a stable, scalable product that can evolve with the platform's needs.
* **Documentation**: Thoroughly documenting components, APIs, and workflows to support ongoing development and onboarding.

**Frontend:**

It Built with React.js for a dynamic, component-based UI, combined with Bootstrap and Material UI. Bootstrap provides a responsive grid system and easy styling utilities, while Material UI offers React-specific components implementing Google's Material Design for a modern, cohesive, and accessible user experience.

**Backend:**

Backend Developed with Node.js and Express.js to manage server -side logic, handle API endpoints for client-server communication, and process core business functions such as authentication, project management, bidding, and messaging.

**Database:**

DatabaseMongoDB is used to store all persistent data, including user profiles, project details, applications, chat messages, and reviews. Its NoSQL document-oriented structure is well suited for handling flexible, evolving data models typical of freelance platforms.

**Setup Instructions:**

**Prerequisites:**

**Nodes.js:**

The Node.js cookbook provides practical recipes and detailed instructions for setting up, developing, and managing scalable Node.js applications. Key pointers from the cookbook include:

* **Installation and Configuration**: Installing Node.js on various platforms and configuring it for development and production environments.
* **Core Concepts**: Understanding asynchronous programming, event-driven architecture, file handling, and streams that form the backbone of Node.js development.
* **Building Web Servers**: Using frameworks like Express.js to create RESTful APIs and handle HTTP requests efficiently.
* **Working with Databases**: Integrating with databases such as MongoDB using native drivers or ODMs like Mongoose for CRUD operations.
* **Middleware and Routing**: Implementing middleware functions to handle cross-cutting concerns and defining clean routing structures.
* **Error Handling and Debugging**: Writing robust error handling debugging tools to troubleshoot issues.
* **Advanced Topics**: Covering real-time applications with WebSocket, creating scalable and maintainable codebases, deployment strategies, and performance optimization.

**Mongo db:**

**Installation and Configuration**:

Support for single MongoDB instances, replication setups, sharding, and combined replication with sharding.Configuration settings such as data directory paths, log file paths, port settings, and authentication options can be customized.The cookbook supports MongoDB package installation and management via repositories.

**User Management and Security**:

Supports enabling authentication and creating MongoDB users with defined roles and permissions.Handles user addition, modification, and deletion with considerations for security, including encrypted data management in production.

**Replication and Sharding:**

Recipes for setting up replica sets for high availability and sharded clusters for scalability.Ensures consistent configuration across nodes with shared key files for authentication.

**Monitoring:**

Integration with MongoDB Monitoring System (MMS) agent to gather and visualize performance metrics, aiding in operational monitoring and troubleshooting.

**Administration:**

Manage configuration file changes with automatic restarts or reloads.Provides resources for seamless upgrades and backup management.This cookbook serves as an essential resource for robust MongoDB deployment and management, adapted for scenarios like backend data storage in applications such as freelance platforms.

**GIT:**

**Instalization and Setup:**

* + - Installing Git via package managers or source on various platforms including Linux, macOS, and Windows.
    - Configuring Git client settings such as user name, email, default editor, alias commands, and credential helpers using the git config command.

**Common Operations**:

* + - Initializing repositories (git init), cloning remote repositories (git clone), staging (git add), committing (git commit), branching, merging, and rebasing.
    - Managing remote repositories (git remote, git fetch, git pull, git push).
    - Undoing changes through reset, revert, clean, and stash.

**Advanced Usage:**

* + - Using Git hooks for automation of tasks like pre-commit checks and post-merge actions.
    - Resolving merge conflicts, cherry-picking commits, and managing tags.
    - Techniques for repository maintenance, including pruning, garbage collection, and recovering lost commits.
    - Integrating Git with continuous integration/deployment workflows.

**Example Recipes:**

* + - Setting up a centralized Git server.
    - Creating highly customized Git workflows for collaborative teams.

**React.js:**

**Project Setup and Environment:**

* Integrating libraries like Redux for state management and React Router for navigation.
* Using styling frameworks such as Bootstrap and Material UI for polished UI components.

**Component Design:**

* Building reusable, composable components using functional and class-based approaches.
* Handling component lifecycle, hooks (useState, useEffect), and context API.

**Data Fetching and API Integration**:

* Fetching data asynchronously using fetch or axios and managing loading/error states.
* Using GraphQL and REST APIs effectively within React components.

**Form Handling and Validation:**

* Managing controlled and uncontrolled form components.
* Implementing validation logic and user input sanitization.

**Testing and Debugging:**

* Writing unit tests and integration tests using Jest and React Testing Library.
* Debugging techniques and performance optimization.

**Express.js – Mongoose – Visual Studio Code**

**1.project Setup in Visual Studio Code**:

* + Open VS Code and create a new Node.js project directory.
  + Initialize npm with npm init -y.
  + Install required packages:

text

npm install express mongoose dotenv

* + Use VS Code terminal and editor for coding, debugging, and version control.

**2.Express.js Server Setup:**

* + Create an app.js or server.js file.
  + Import Express and create an app instance:

javascript

**const** express = require('express');

**const** app = express();

app.use(express.json());

* + Set up basic routes and middleware.

**3.Mongoose Connection and Schema Definitions:**

* + Create a separate config file to set up MongoDB connection:

javascript

**const** mongoose = require('mongoose');

**const** connectDB = **async** () => {

**try** {

**await** mongoose.connect(process.env.MONGO\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true

}); javascript

**const** mongoose = require('mongoose');

**const** connectDB = **async** () => {

**try** {

**await** mongoose.connect(process.env.MONGO\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true

});

console.log('MongoDB connected');

} **catch** (err) {

console.error(err);

process.exit(1);

}

};

module.exports = connectDB;

console.log('MongoDB connected');

} **catch** (err) {

console.error(err);

process.exit(1); javascript

**const** mongoose = require('mongoose');

**const** connectDB = **async** () => {

**try** {

**await** mongoose.connect(process.env.MONGO\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true

});

console.log('MongoDB connected');

} **catch** (err) {

console.error(err);

process.exit(1);

}

};

module.exports = connectDB;

}

};

module.exports = connectDB;

* + Define schemas and models in separate files using Mongoose:

javascript

**const** mongoose = require('mongoose');

**const** RecipeSchema = **new** mongoose.Schema({

title: { type: String, required: true },

ingredients: [String],

instructions: { type: String, required: true },

author: { type: mongoose.Schema.Types.ObjectId, ref: 'User' }

});

**const** Recipe = mongoose.model('Recipe', RecipeSchema);

**4.Using Controllers and Routes:**

* + Organize logic into controllers.
  + Create Express routes importing controller functions to handle CRUD operations for recipes or other data.

**5.Running and Debugging in Visual Studio Code:**

* + Use VS Code debugging tools to run the Node.js server.
  + Set breakpoints and inspect variables.
  + Configure environment variables (.env file) to keep sensitive data like database URI.

**Installation Steps:**

**#Clone the repository git clone**:

bash

git clone <repository-url>

**#Install client dependencies cd client npm instal:**

bash

cd client

npm install

**#Install server dependencies cd ../server npm install :**

bash

cd ../server

npm install

**Running the Application:**

* **Navigate to the client directory of the project**:

text

cd client

* **Run the React development server using npm:**

text

npm start

**Access: Visit** [**http://localhost:3000**](http://localhost:3000)

**API Documentation:**

**User Registration**: /api/user/register

* Create an Express route to handle POST requests to /api/user/register.
* Extract user details like username, email, and password from the request body.
* Check if the user already exists by searching the database.
* Hash the password using bcrypt (or a similar library).
* Save the new user to the MongoDB database using Mongoose.
* Send a success response or error messages accordingly.

**User Login: /api/user/login**

* Handle POST requests to /api/user/login.
* Extract email and password from request.
* Find the user by email and verify the password.
* Generate a JSON Web Token (JWT) for authenticated sessions.
* Return the token to the client or error messages if login fails.

**Projects:**

**API: /api/projects/create (Create Project**)

* Method: POST
* Receives project details in request body.
* Validates and saves the project to MongoDB using Mongoose.
* Responds with the created project or error message.

**API: /api/projects/:id (Get Project by ID)**

* Method: GET
* Extracts project ID from URL params.
* Queries MongoDB for the project using the ID.
* Responds with project details or not found message

**Chats:**

**API: /api/chat/send (Send a Chat Message)**

* Method: POST
* Receives a chat message payload in the request body including sender, recipient, message text, timestamp, and optionally chat room ID.
* Saves the message to the MongoDB collection (ChatMessages).
* Emits a real-time event if using WebSocket or Socket.IO to update clients.
* Returns a success response or error message.

**/api/chat/:userId:**

API: /api/chat/:userId (Get User Chats)

* Method: GET
* Extract userId from request parameters.
* Query MongoDB for chat messages or chat rooms involving the user.
* Return the list of messages or chat rooms as JSON.

**User Interface:**

**Landing Page:**

* Visually appealing hero section with clear call-to-action buttons for clients and freelancers.
* Overview of platform features and benefits.
* Client and freelancer testimonials or success stories.
* Easy navigation to sign-up/login and project browsing.

**Freelancer Dashboard:**

* Personalized greeting and overview of active projects/applications.
* Tabs or sections for bidding on projects, messages, and notifications.
* Quick access to profile editing and review history.
* Clean layout with project cards or list views showing bid status.

**Admin Panel:**

* User management with search and filter options (freelancers, clients).
* Project oversight for approving, deleting, or managing posted projects.
* Monitoring of disputes, reviews, and site analytics.
* Role-based access controls with clear navigation and dashboard widgets.

**Project Details Page:**

* Detailed project description with budget, timeline, and client info.
* Apply button or bidding interface for freelancers.
* Section for client reviews and freelancer proposals.
* Interactive comments or chat component.
* Responsive and uncluttered design focusing on key project info.

Testing:

**Manual Testing During Milestones**

* Perform manual testing at key development milestones (e.g., after completing login, project posting, bidding modules).
* Validate core workflows end-to-end to catch logical or UI issues early.
* Ensure critical API endpoints behave as expected under various input conditions.
* Manually verify UI responsiveness, navigation, forms, and interactive elements.

**Tools for API and Frontend Testing**

**Postman**

* Use Postman for manual functional testing of RESTful APIs.
* Create and organize collections for endpoints such as user auth (/api/user/login, /register), projects (/api/projects), chat (/api/chat).
* Validate request payloads, responses, HTTP status codes, headers, and JSON schemas.
* Test both positive scenarios (valid data) and negative cases (invalid data, edge cases).
* Save and reuse requests to speed up regression testing.

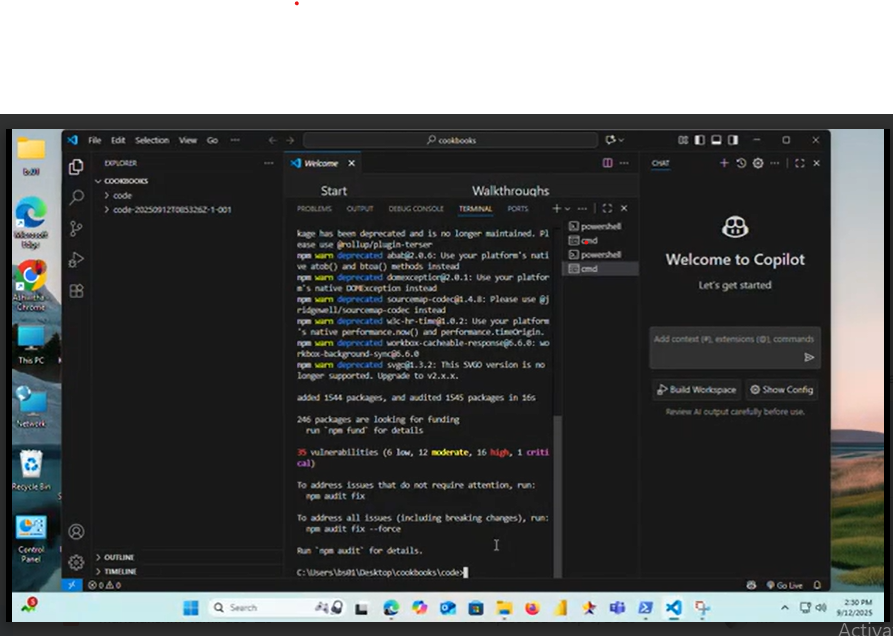
**Chrome Dev Tools**

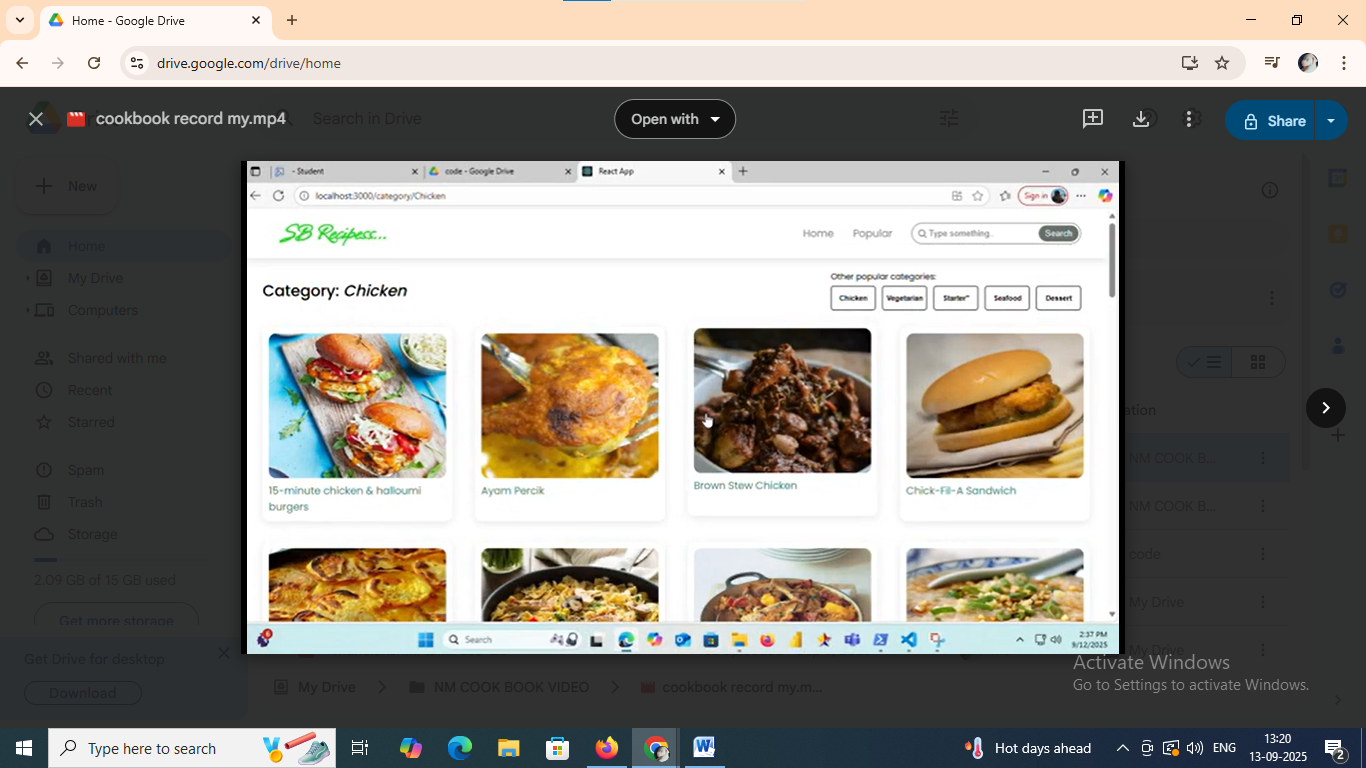
* Inspect network calls to verify API requests and responses during frontend interactions.
* Debug JavaScript errors.
* Test UI responsiveness and simulate various device screen sizes.
* Utilize console logs and breakpoints for detailed debugging.

**Best Practices**

* Maintain clear and updated API documentation for precise manual test cases.
* Combine manual tests with automated scripts for regression coverage.
* Test errors and boundary conditions explicitly.
* Track issues and retest after fixes.
* Ensure security testing by verifying authentication and protected routes during manual tests.

**Screenshot:**

****

****

**Demo link:**

**https://drive.google.com/drive/folders/18kA6njt0ViJv2i-ENu9N3BpWXbeaTWMf?usp=drive\_link**