# Recipe Sharing Platform

## Recipe Sharing App – Frontend Documentation

The frontend of the Recipe Sharing App is developed using **React.js**, a modern and component-driven JavaScript library that ensures a dynamic and responsive user interface. This documentation outlines the architectural flow, major components, navigation, and additional features like search functionality, giving a complete overview of how the frontend interacts with the backend and renders the user experience.

The React application is structured under the /frontend directory. The main.jsx file serves as the starting point, where the application is rendered into the DOM. Routing is handled within App.jsx, connecting different pages such as home, add recipe, and edit recipe using react-router-dom. Core components and reusable logic are organized into subdirectories like /components and /pages, promoting modularity and reusability.

### Application Flow

The application initiates from main.jsx, which renders the App component. Inside App.jsx, routing is configured to dynamically load page components based on the URL path. Home fetches and displays all recipes, while pages like AddFoodRecipe and Edit allow authenticated users to manage recipe content. UI navigation is maintained through MainNav, and user interactions are handled through modal-based login systems.

### Component Breakdown

**App.jsx**

- **Role:** Central routing hub of the application.
- **Details:** Configures paths for /, /add, and /edit/:id using Routes and Route components. Provides consistent page rendering based on route.

**main.jsx**

- **Role:** Entry point of the React application.
- **Details:** Attaches the App component to the root DOM node using ReactDOM.createRoot.

**/components**

**MainNav.jsx**

- **Role:** Site-wide navigation bar.
- **Details:** Provides links to core routes (Home, Add Recipe) and manages login/logout state. Ensures accessible site-wide navigation.

**LoginModal.jsx**

- **Role:** Handles user login in a modal interface.
- **Details:** Accepts credentials, makes API call to login endpoint, stores JWT locally upon success, and closes the modal post-authentication.

**Items.jsx**

- **Role:** Displays recipe cards.
- **Details:** Iterates over recipe data to display each item. Supports edit, delete, and view buttons for each recipe. Accepts filtered data from Home.jsx.

**InputForm.jsx**

- **Role:** Reusable form component for both adding and editing recipes.
- **Details:** Manages input fields for title, ingredients, steps, time, and photo. Uses controlled inputs and validation before submission.

**RecipeDetails.jsx**

- **Role:** Displays full details of a single recipe.
- **Details:** Fetches recipe by ID using axios, and renders complete content including image, ingredients, and steps.

**Footer.jsx**

- **Role:** Static site footer.
- **Details:** Provides consistent branding or messaging across pages.

**Navigation.jsx**

- **Role:** (Optional/Legacy) Alternate navigation component.
- **Details:** Similar to MainNav, may be unused or under consideration for future replacement.

**/pages**

**Home.jsx**

- **Role:** Main landing page listing all recipes.
- **Details:** Fetches all recipes on load via axios. Displays them through Items.jsx. Also implements real-time **search functionality**.

**AddFoodRecipe.jsx**

- **Role:** Page for submitting a new recipe.
- **Details:** Uses InputForm to gather recipe details. On submit, sends a POST request to backend with recipe data and image.

**Edit.jsx**

- **Role:** Page for editing existing recipes.
- **Details:** Fetches the recipe by ID using URL params. Prefills InputForm with existing data, allowing updates via PUT request.

## Search Functionality

The application includes a real-time client-side search feature implemented in Home.jsx.

- **Search UI:** An input field at the top of the recipe list captures user queries.
- **Logic:** Filters recipes in memory by comparing the lowercase title with the search input string.

This feature improves user experience by allowing instant search without backend queries.

## Authentication and Authorization (Frontend Perspective)

- **Login Workflow:** Handled via LoginModal, which sends credentials to /api/user/login. On success, JWT is stored in local storage.
- **Protected Routes:** Pages like Add/Edit conditionally render forms based on login state. JWT is attached to requests where needed (e.g., add/edit/delete).
- **Logout:** Clears the token and resets state via the navigation bar.

## UI Component Interaction

- Navigation (MainNav) → Routing (App.jsx) → Pages (Home, Add, Edit)
- Pages → Components (InputForm, Items, RecipeDetails)
- State & Props → Manage user input, token state, and recipe data

**Summary**

This frontend architecture emphasizes a clear separation of UI, logic, and routing. Using React’s component-driven model, each feature is encapsulated and easily extendable. The app handles authentication, CRUD operations, and search smoothly, interacting seamlessly with the backend. Combined with modular structure and route-aware pages, the frontend ensures scalability, user experience, and maintainability in a modern web environment.

# Recipe Sharing App - Backend Documentation

The backend of the Recipe Sharing App is built using a **modern MERN stack** approach, where the server logic is handled through Node.js and Express.js. It enables seamless interactions between the frontend and the database, supporting various user and recipe-related functionalities. This document provides a detailed yet easy-to-understand walkthrough of the backend implementation, highlighting the architecture, API routes, logic, and security practices.

The backend server is structured in a modular format under the /backend directory. The index.js file serves as the entry point, initializing the server and connecting to MongoDB through the configuration defined in config/mongoDB.js. Data models are defined using Mongoose schemas in the models directory, which include userModel.js for handling user-related information and recipeModel.js for storing recipe data. Business logic is encapsulated within the controller directory, which contains userController.js for managing user actions like signup and login, and recipeController.js for creating, retrieving, updating, and deleting recipe entries.

The application uses a combination of robust libraries and tools to enhance security and maintainability. Node.js provides a powerful asynchronous runtime, while Express.js simplifies routing and middleware management. MongoDB, coupled with Mongoose, facilitates dynamic data handling with schema validation. Authentication is securely managed using JSON Web Tokens (JWT), allowing user sessions to be validated efficiently. Passwords are encrypted using bcrypt before storage, ensuring that sensitive data is never exposed directly. Environment variables are managed with dotenv, keeping critical secrets like the JWT secret key outside the source code.

## User Signup:

The signup API route is defined at POST /api/user/signup. When a new user attempts to register, the server first validates that both email and password fields are provided. It then checks if the user already exists in the database. If not, it hashes the password using bcrypt with a defined salt round and stores the new user in the database. A JWT is generated upon successful signup, which is returned to the client for maintaining the session.

## User Login:

The login route, defined at POST /api/user/login, accepts email and password credentials. It searches the database for a matching user email. The password entered is then compared to the hashed password stored in the database using bcrypt.compare, which is awaited to ensure correct validation. If the credentials are valid, a JWT is signed and returned. This token authenticates subsequent requests to protected routes.

## Get User by ID:

Accessible via GET /api/user/:id, this route allows the frontend to retrieve user information such as email by providing a valid user ID. This can be useful for displaying user-specific content or settings.

## Create Recipe:

Recipes can be added through the POST /api/recipes route. This is a protected route, meaning it requires a valid JWT to access. Users can submit new recipes by providing relevant details such as the title, ingredients, and preparation steps. The recipe is saved in the MongoDB collection associated with the authenticated user.

**Retrieve All Recipes:**

The GET /api/recipes route is used to fetch all recipes from the database. This route does not require authentication, making it accessible to all users for browsing and exploring community-shared recipes.

**Update a Recipe:**

Users can update existing recipes using the PUT /api/recipes/:id endpoint. It requires authentication and the recipe ID to locate and modify the correct entry. This is useful when users want to revise or enhance their previously submitted recipes.

**Delete a Recipe:**

The DELETE /api/recipes/:id route enables the deletion of recipes. It is also a protected route and ensures that only authenticated users can delete their own entries. The server locates the recipe by ID and removes it from the database.

**Authentication Middleware:**

The authentication middleware (/middleware/auth.js) is a crucial component that ensures only authorized users can access certain routes. It works by checking for a valid JWT in the request headers. If the token is verified successfully, the user's information is attached to the request object, allowing access to the protected routes. If the token is missing or invalid, the middleware sends a 401 Unauthorized response.

**Application Flow:**

The application begins by allowing users to either sign up or log in. Upon successful authentication, a token is issued and stored on the client side. This token is then used to make authenticated requests to protected routes like adding, updating, or deleting recipes. Recipes can be viewed by all users, but modifications are restricted to authenticated users. This flow ensures a secure and user-specific experience across the platform.

**Security Practices:**

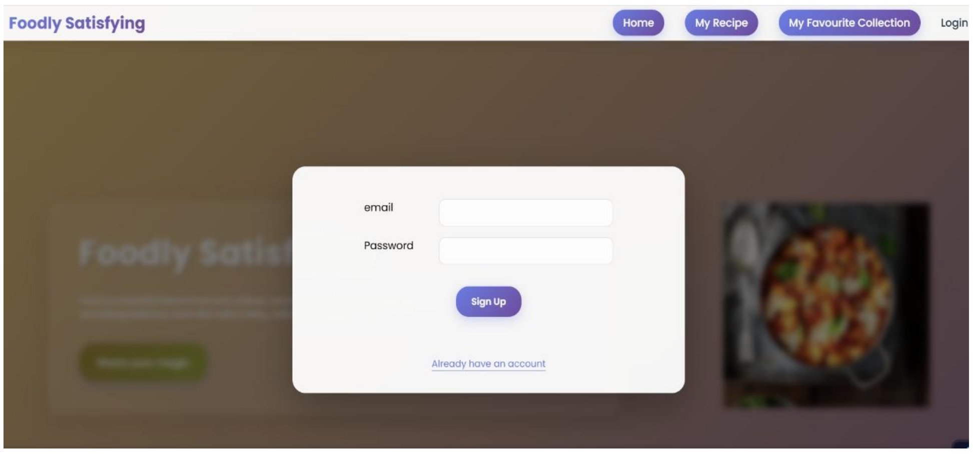
Passwords are hashed before being stored, ensuring no raw passwords are saved in the database. JWTs are signed using a secret stored securely in an .env file, preventing exposure of sensitive credentials in the codebase. The system is designed to verify and authorize requests appropriately, enhancing data privacy and user security.

**Summary:**

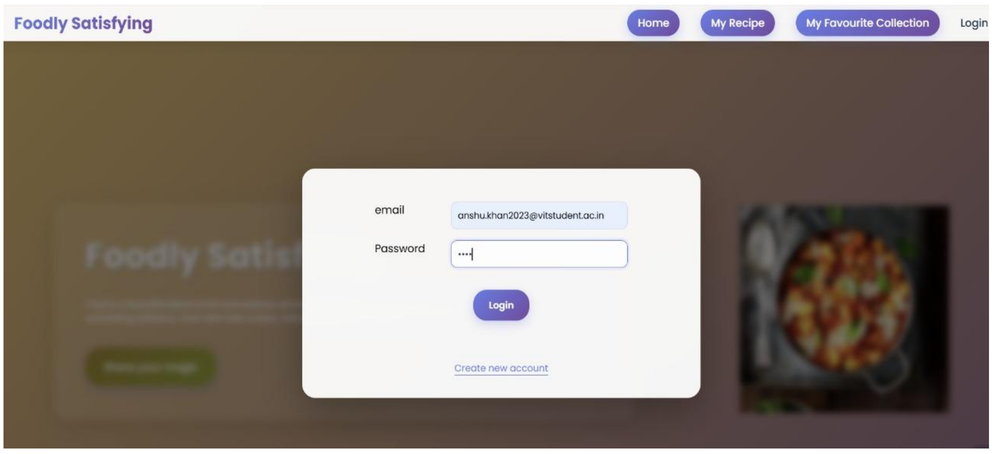
This backend implementation offers a well-organized, scalable, and secure system to manage users and recipes. The modular structure with clear separation of concerns facilitates easy maintenance and expansion. With strong authentication, route protection, and encrypted data handling, the app is well-suited for deployment in a modern web environment.

**<u>Screenshots of the website</u>**

1. Sign Up

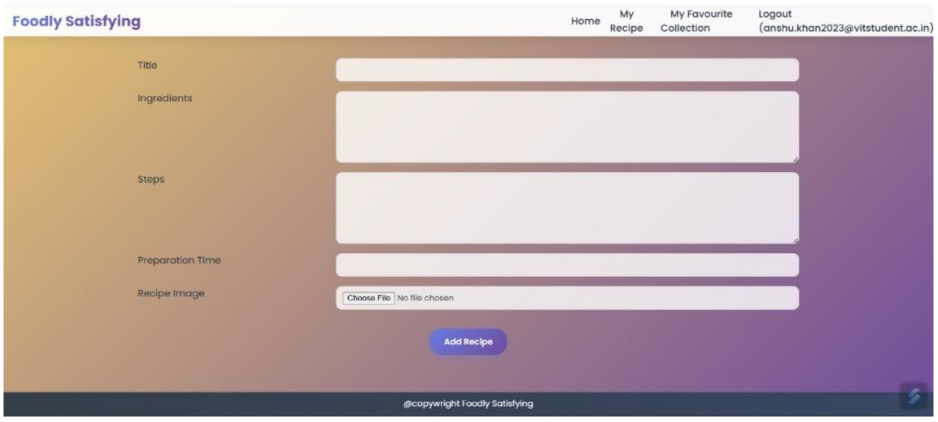


2. Login

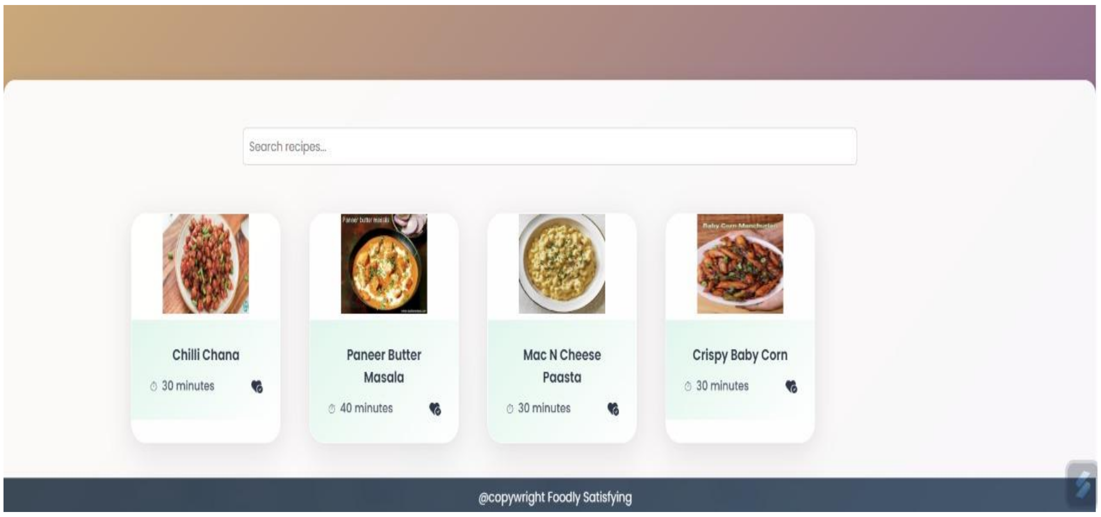


email: anshu.khan2023@vitstudent.ac.in

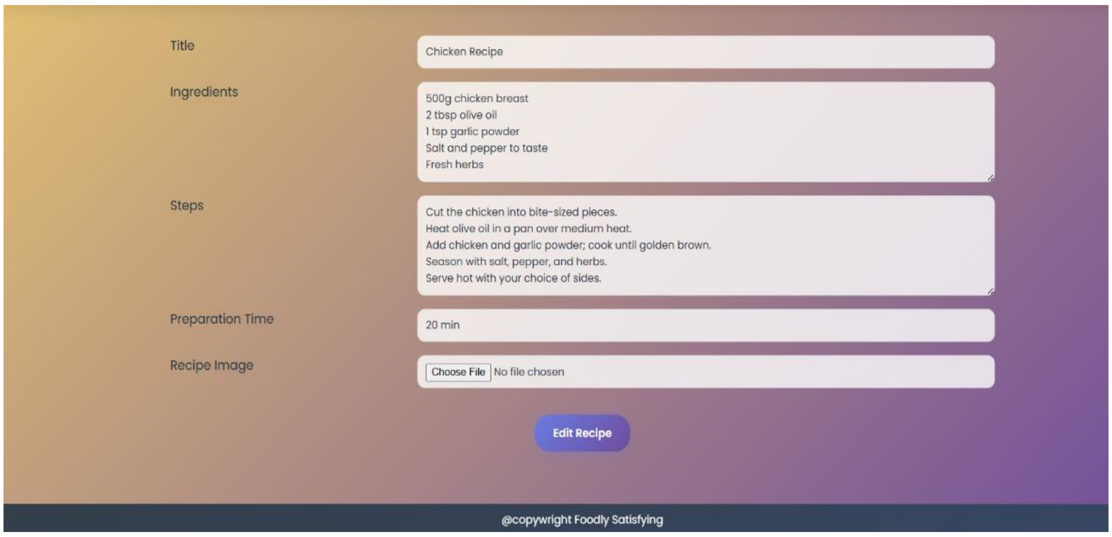
3. Add Recipe



4. All Recipes



5. Edit Recipe



Title: Chicken Recipe

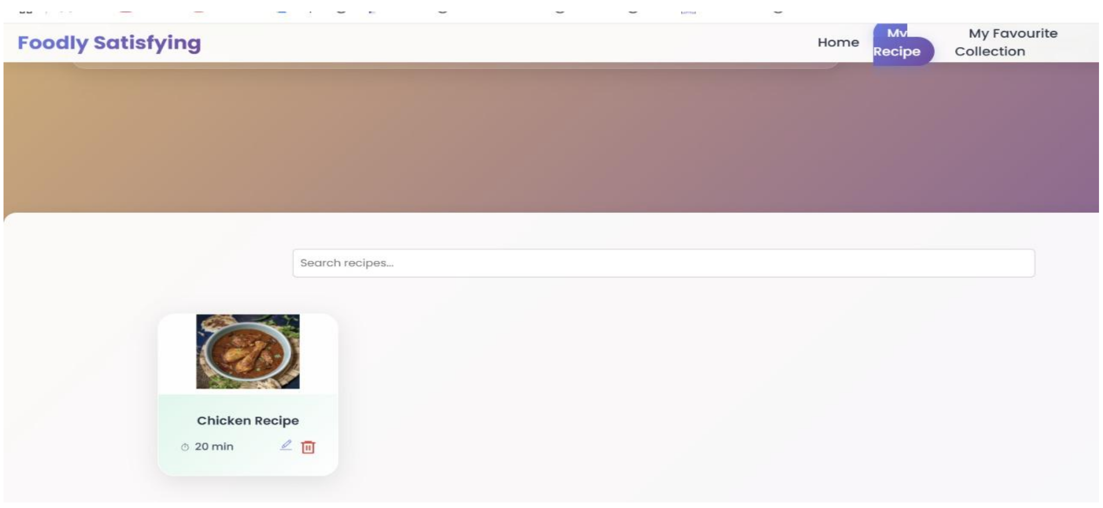
Ingredients:
500g chicken breast
2 tbsp olive oil
1 tsp garlic powder
Salt and pepper to taste
Fresh herbs

Steps:
Cut the chicken into bite-sized pieces.
Heat olive oil in a pan over medium heat.
Add chicken and garlic powder; cook until golden brown.
Season with salt, pepper, and herbs.
Serve hot with your choice of sides.

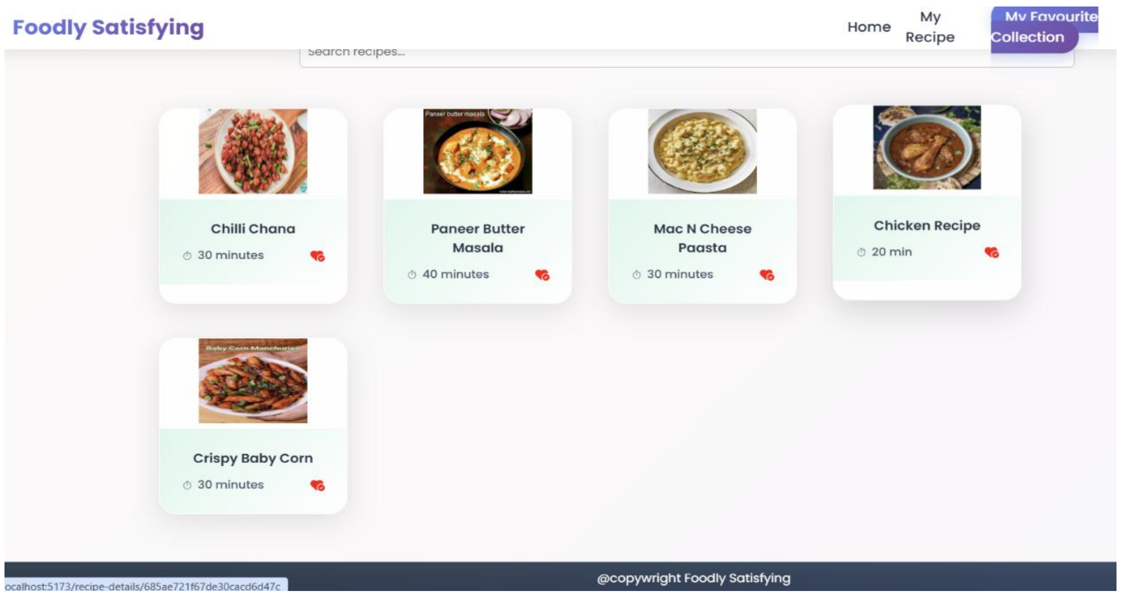
Preparation Time: 20 min

Recipe Image: Choose File No file chosen

6. My Recipe

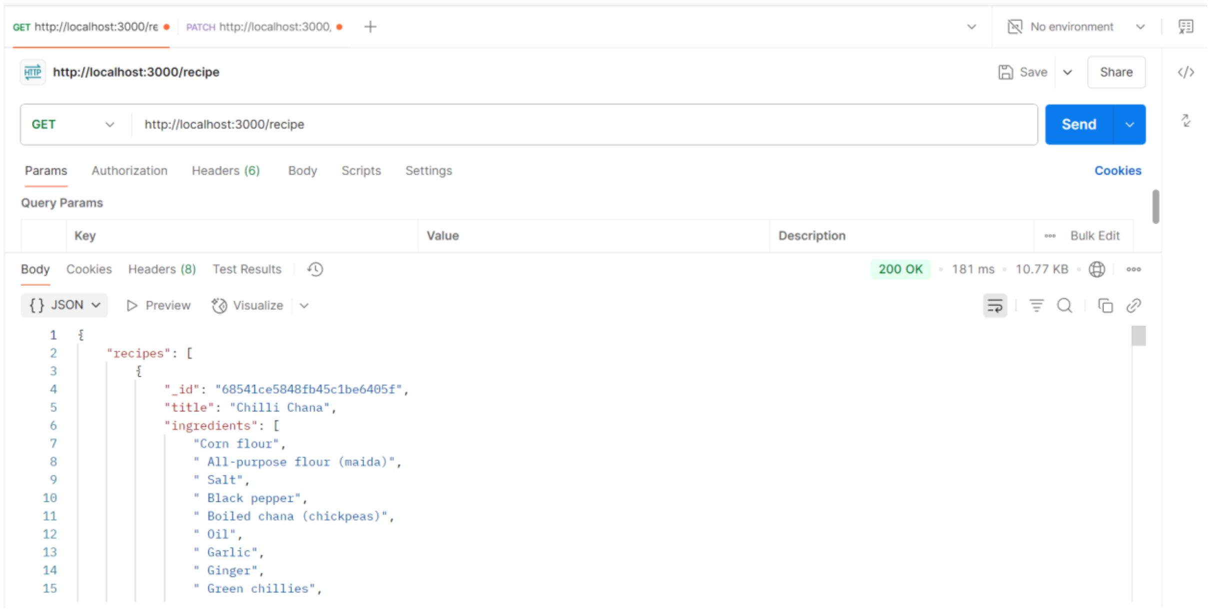


7. Favourite Recipe



## Screenshots of postman (backend)

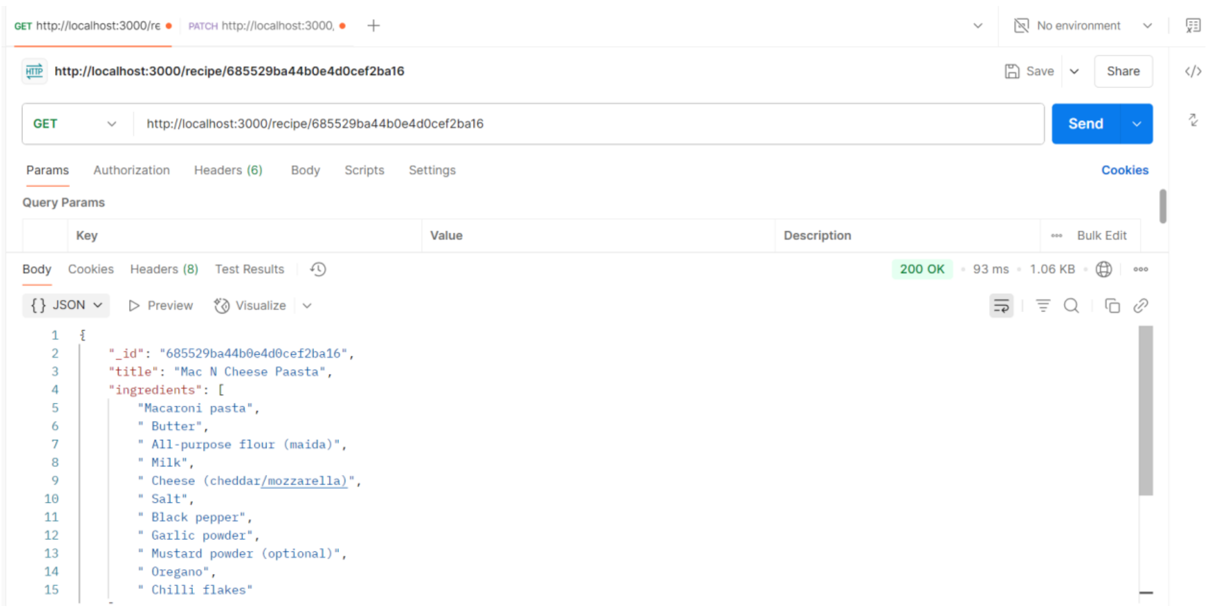
1. Get All Recipes



GET http://localhost:3000/recipe

```
{
    "recipes": [
        {
            "_id": "68541ce5848fb45c1be6405f",
            "title": "Chilli Chana",
            "ingredients": [
                "Corn flour",
                " All-purpose flour (maida)",
                " Salt",
                " Black pepper",
                " Boiled chana (chickpeas)",
                " Oil",
                " Garlic",
                " Ginger",
                " Green chillies",
```

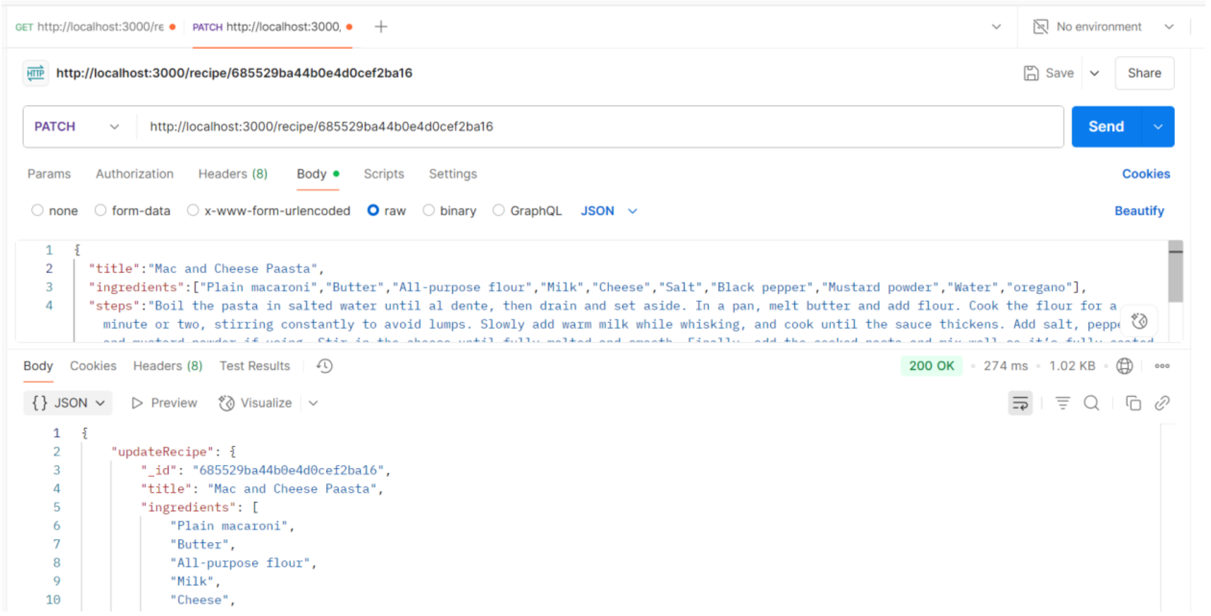
2. Get Recipe By ID


GET http://localhost:3000/recipe/685529ba44b0e4d0cef2ba16

```
{
  "_id": "685529ba44b0e4d0cef2ba16",
  "title": "Mac N Cheese Paasta",
  "ingredients": [
    "Macaroni pasta",
    " Butter",
    " All-purpose flour (maida)",
    " Milk",
    " Cheese (cheddar/mozzarella)",
    " Salt",
    " Black pepper",
    " Garlic powder",
    " Mustard powder (optional)",
    " Oregano",
    " Chilli flakes"
```

Status: 200 OK · 93 ms · 1.06 KB

3. Update Recipe By ID

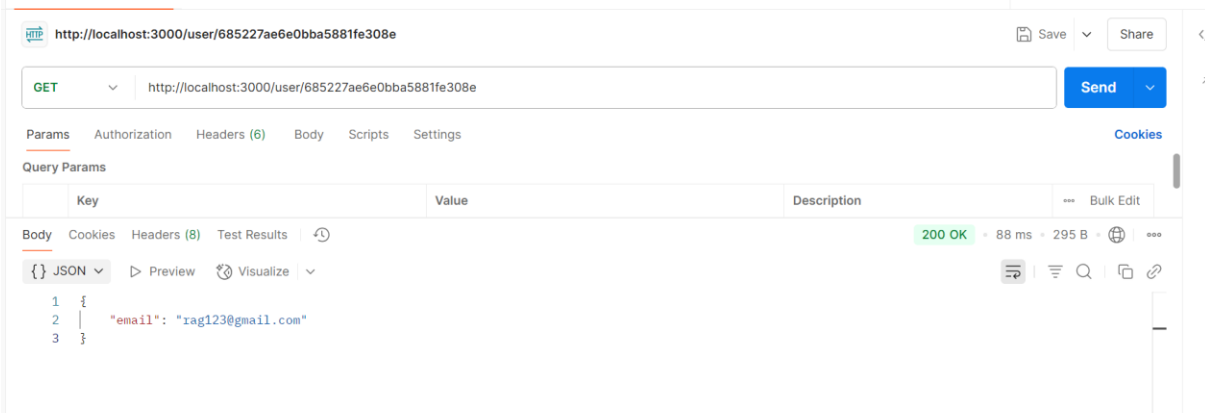

PATCH http://localhost:3000/recipe/685529ba44b0e4d0cef2ba16

```
{
"title":"Mac and Cheese Paasta",
"ingredients":["Plain macaroni","Butter","All-purpose flour","Milk","Cheese","Salt","Black pepper","Mustard powder","Water","oregano"],
"steps":"Boil the pasta in salted water until al dente, then drain and set aside. In a pan, melt butter and add flour. Cook the flour for a
  minute or two, stirring constantly to avoid lumps. Slowly add warm milk while whisking, and cook until the sauce thickens. Add salt, peppe
```

Status: 200 OK · 274 ms · 1.02 KB

```
{
  "updateRecipe": {
    "_id": "685529ba44b0e4d0cef2ba16",
    "title": "Mac and Cheese Paasta",
    "ingredients": [
      "Plain macaroni",
      "Butter",
      "All-purpose flour",
      "Milk",
      "Cheese",
```

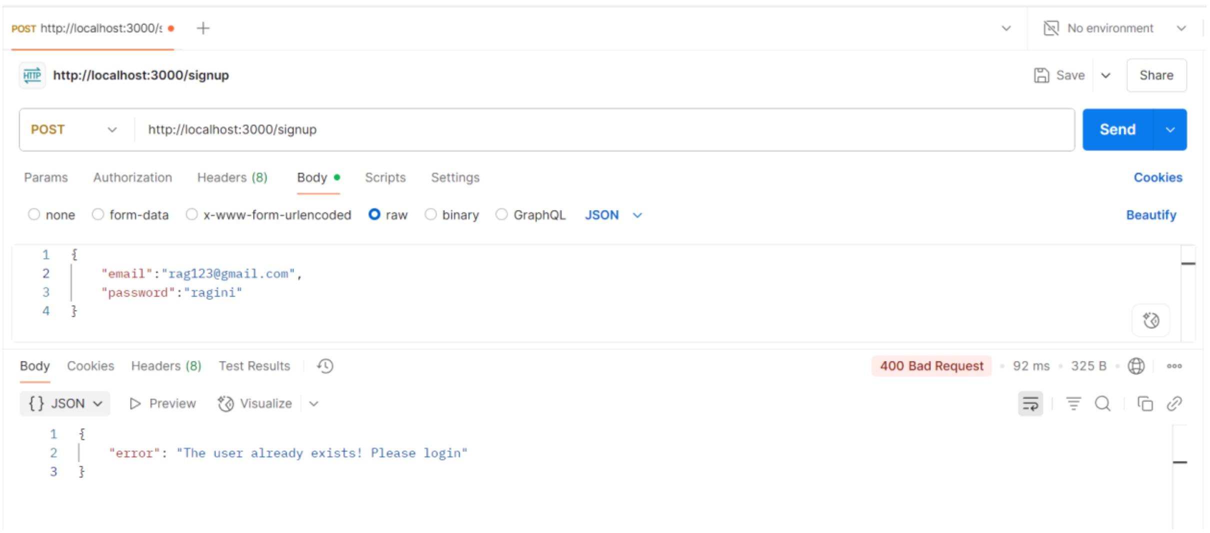
4. Get User by ID


GET http://localhost:3000/user/685227ae6e0bba5881fe308e

Status: 200 OK · 88 ms · 295 B

```
{
  "email": "rag123@gmail.com"
}
```

5. New Sign Up (User already exisis)



http://localhost:3000/signup

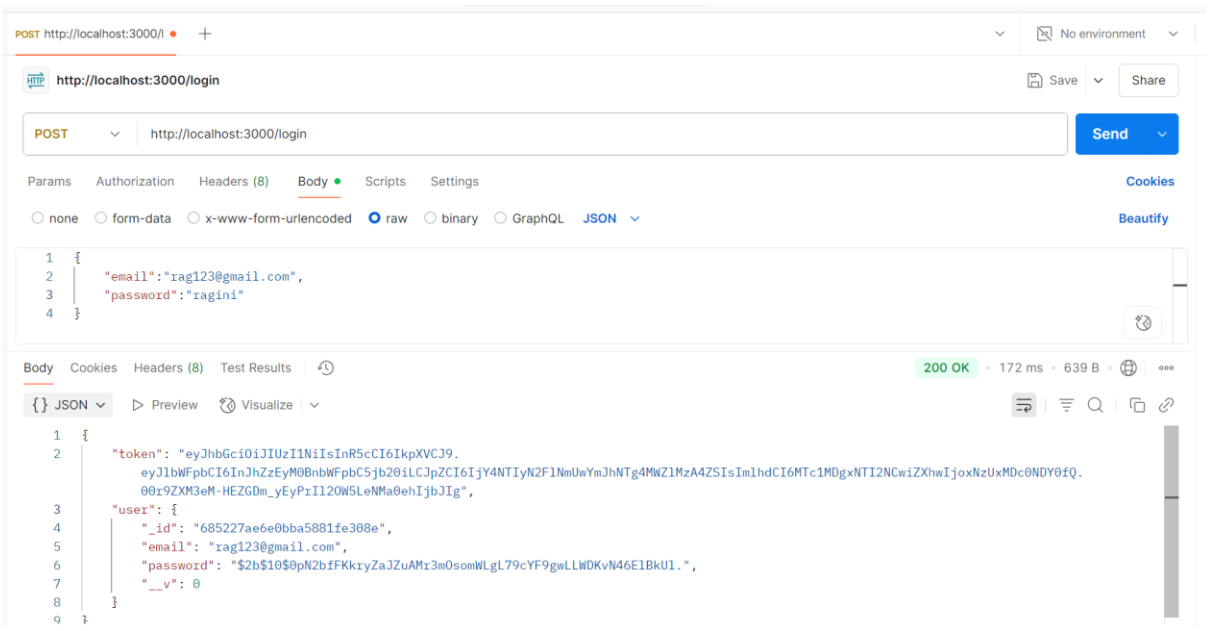
POST http://localhost:3000/signup

```
{
    "email":"rag123@gmail.com",
    "password":"ragini"
}
```

400 Bad Request · 92 ms · 325 B

```
{
    "error": "The user already exists! Please login"
}
```

6. Login



http://localhost:3000/login

POST http://localhost:3000/login

```
{
    "email":"rag123@gmail.com",
    "password":"ragini"
}
```

200 OK · 172 ms · 639 B

```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJlbWFpbCI6InJhZzEyM0BnbWFpbC5jb20iLCJpZCI6IjY4NTIyN2FlNmUwYmJhNTg4MWZlMzA4ZSIsImlhdCI6MTc1MDgxNTI2NCwiZXhwIjoxNzUxMDc0NDY0fQ.
        00r9ZXM3eM-HEZGDm_yEyPrIl20W5LeNMa0ehIjbJIg",
    "user": {
        "_id": "685227ae6e0bba5881fe308e",
        "email": "rag123@gmail.com",
        "password": "$2b$10$0pN2bfFKkryZaJZuAMr3mOsomWLgL79cYF9gwLLWDKvN46E1BkUl.",
        "__v": 0
    }
}
```