**Service Layer Design Document**

**1. Overview**

The Recip-Ease application utilizes RESTful api to allow the web pages to communicate with the database. Frankly it is about the simplest way to build the UI/database infrastructure, and it was simple to implement the necessary CRUD operations required. Also, with JSON responses, the data is structured on return from the database and easy to handle programmatically. The set up is highly scalable, though in the case of this project that is of less consequence. RESTful processes do mesh well with MongoDBs document-based data model.

Database structure and endpoints were easily adaptable to the needs of Recip-Ease. Javascript is simple to develop, and flexible enough to handle the needs of the project. As can be seen from the screenshots, the database and the RESTful api are fairly well developed. While we are early in the Capstone project lifecycle, I would remind you that I have had this project in mind since about 2020. My vision is solid, and I have done a lot of research on how it would work, and how I would make it work functionally, have been researched and settled a long time ago.

For the purpose of this document, we will progress through the workflow indicating which endpoints come into play. Graphical examples of the appropriate workflows will appear at the end of the discussion of each portion of the website. Some CRUD operations like DELETE for users and some others like PUT for the ingredients collections have been back-burnered as they are not necessary for the functionality of the MVP.

**2. Service Endpoints Summary**

| **HTTP Method** | **Endpoint** | **Page(s)** | **Purpose** |
| --- | --- | --- | --- |
| **POST** | /api/users/register | User Login/Create Account Page | Register a new user |
| **POST** | /api/users/login | User Login/Create Account Page | Authenticate a user |
| **PUT** | /api/users/:id | User Login/Create Account Page | Update user email or password |
| **GET** | /api/users/:id | User Login/Create Account Page | Retrieve user details |
| **POST** | /api/recipes/basicinfo | Recipe Entry Page | Create a new recipe |
| **GET** | /api/recipes | Recipe Entry Page  Recipe Search Page | Retrieve all recipes for the logged-in user |
| **GET** | /api/recipes/:id | Home Page  Recipe Entry Page  Recipe Search Page | Retrieve a specific recipe by ID |
| **PUT** | /api/recipes/:id | Recipe Entry Page | Update an existing recipe |
| **DELETE** | /api/recipes/:id | Recipe Entry Page  Recipe Search Page | Delete a recipe |
| **GET** | /api/recipes/random/4 | Home Page | Retrieve 4 random recipes for homepage slideshow |
| **PUT** | /api/recipes/:id/ingredients | Recipe Entry Page | Add/update an ingredient to a recipe |
| **PUT** | /api/recipes/:id/instructions | Recipe Entry Page | Add/update recipe instructions |
| **DELETE** | /api/recipes/:id | Recipe Entry Page  Recipe Search Page | Delete a specific recipe |
| **DELETE** | /api/recipes/:id/instructions | Recipe Entry Page | Delete an instruction from a recipe |
| **DELETE** | /api/recipes/:id/ingredients | Recipe Entry Page | Delete an ingredient from a recipe |
| **GET** | /api/:collectionName | Recipe Entry Page | Retrieve common ingredients for type of recipe (like “cake” in the dessert\_ingredient collection) |
| **PUT** | /api/:collection/:ingredient/increment | Recipe Entry Page | When an ingredient is added to a recipe, the “usage\_count” is incremented to increase the ingredient’s popularity for later lists |

**3. Endpoint Descriptions & Example Requests**

**User Registration**

Note: While I believe the website and functionality have been well thought out, as development proceeds, no doubt further functionality and endpoints will likely be added. However, I believe the current state of the website and these endpoints will satisfy the requirements of the MVP. One area which would concern me greatly were this website made publicly accessible, would be security. Currently, the API is open and public-facing without authentication and authorization. If I should plan to take this project beyond the MVP, security would be my first implementation. Being a non-SQL database, I at least don’t have to worry about injection attacks so much, though I understand MongoDB isn’t impervious to such things.

**Home Page Functionality**

Other than navigating to other pages, the home page doesn’t have much to do with our persona’s workflow. A couple things I will be implementing are the photo display at the top, and the “Featured Recipes” at the bottom. Both utilize the same endpoint, GET /api/recipes/random/4, which selects four of the recipes at random from the database and displays photos and descriptions of the recipes.

**Retrieve Random Recipes for Homepage**

**Endpoint:** GET /api/recipes/random/4

**Example Request:**

A screenshot of a computer

AI-generated content may be incorrect.

**Error Response (500 Internal Server Error):**

{

"error": "Database error while fetching random recipes"

}

The most common navigation workflow for most users will be to navigate to the User Sign-In/Create User page. Recipes can be identified by the user creating them, and the user can list and view their recipes. Custom cookbooks, or collections of recipes will be a stretch feature. Currently, when a user either registers or logs in, the endpoint returns a success response along with the user’s \_id, which can be stored in a variable for including in new recipe documents or for searches. The first endpoint is for registering a new user. It accepts a username, email, and password, which it hashes when storing. The API also checks for duplicate usernames or emails.

**New User Registration Endpoint:** POST /api/users/register

**Example Request/Response:**

A screenshot of a computer

AI-generated content may be incorrect.

**Error Response (400 Bad Request - Duplicate User):**

A screenshot of a computer

AI-generated content may be incorrect.

Once a user account has been created, users can log in with their username and password. The API returns the user’s document \_id.

**User Login**

**Endpoint:** POST /api/users/login

**Example Request:**

A screenshot of a computer

AI-generated content may be incorrect.

**Error Response (401 Unauthorized):**

**A screenshot of a computer

AI-generated content may be incorrect.  
Error Response (400 Bad Request):**

A screenshot of a computer

AI-generated content may be incorrect.

Users will have the ability to change either their email address or password. The first step in that process is to retrieve the user details. Once logged in and the userId stored, the system retrieves the user’s information with a GET endpoint for the user, then they can change either their email address or password.

**Retrieve User Details**

**Endpoint:** GET /api/users/:id

**Example Request (Successful):**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Error Response (404 Not Found):**

A screenshot of a computer

AI-generated content may be incorrect.

Once the user details are displayed, the user can update their email or password.

**Update User Details (change email or password)**

**Endpoint:** PUT /api/users/:id

**Example Request:  
A screenshot of a computer

AI-generated content may be incorrect.**

**Database record updated:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Error Response (404 Not Found):**

**A screenshot of a computer

AI-generated content may be incorrect.**

Once logged in, most persona will likely go to the recipe Search page. There, they will have the several options for searching for and displaying their and others, recipes. There are several endpoints for these functions and others on the page; the first is to retrieve all recipes in the database, the second is to retrieve just the user’s recipes, and the last is to retrieve recipes using either category and type or with a search term. The same endpoint works for either of the last options.

**Retrieve All Recipes Endpoint:** GET /api/recipes

**Example Request:  
A screenshot of a computer

AI-generated content may be incorrect.**

**Retrieve Recipe By ID Endpoint:** GET /api/recipes/:id

**Example Request:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Error Response (404 Not Found):**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Recipe Search Endpoint:** GET /api/recipes/search?query=<search\_term>

**Example Request:**

A screenshot of a computer

AI-generated content may be incorrect.

**Error Response (404 Not Found):**

A screenshot of a computer

AI-generated content may be incorrect.

The final set of endpoints as far as the MVP is concerned is the creation, modification, and deletion of recipes by the user. To recap, to create a recipe, the user will enter some details like recipe name, as well as category and type. I will likely add “description” to this basic information in the MVP as well. Initially, given the workflow of the recipe entry page, the recipe document will be created at this point, the recipe \_id stored, and as the user enters the ingredients and instructions, a PUT endpoint will add them to the recipe document. The returned “recipe\_id” field is loaded into a variable and utilized for later PUT endpoints to add ingredients and instructions to the recipe.

**Create New Recipe Endpoint:** POST /api/recipes/basicinfo

**Example Request:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Malformed Request (400 Bad Request):**

A screenshot of a computer program

AI-generated content may be incorrect.

The user can update the basic recipe information such as name, category, type. Those are updated with this endpoint. Ingredients and recipe instructions are handled by other endpoints for a variety of reasons, most of which have to do with the workflow design.

**Update Recipe Information Endpoint:** PUT /api/recipes/:id

**Example Request:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Error Message (404 Not Found):**

**A screenshot of a computer

AI-generated content may be incorrect.**

Once the user has entered or updated the basic recipe info, the database will utilize this information to populate a list of the most common ingredients in a scrollable list. The items on the list will be clickable for selection by the user. This endpoint, utilizes the category (which translates into :collectionName) and type to GET the list. (The ingredient collections are preloaded with 100 – 125 common ingredients culled from some 10,000 recipes, and ranked by common use using AI. This is a database administrative function and not a part of the website’s functional capabilities.)

**Retrieve Common Ingredients By Type Endpoint:** GET /api/:collectionName

**Example Request for Ingredients from dessert\_ingredients for recipe type ‘cake’:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Example Error (400 Bad Request):**

**A screenshot of a computer

AI-generated content may be incorrect.**

Once the ingredient list is loaded, the user clicks on the ingredient, then click the quantity buttons provided in the left navigation. As they click these, the quantities are loaded into a variable, and when the user hits the “Submit Ingredient” button, a PUT endpoint adds the ingredient to the recipe.

**Add or Modify Ingredients to Recipe Document Endpoint:** PUT /api/recipes/:id/ingredients

**Example Request:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Example Error (400 Bad Request):  
A screenshot of a computer program

AI-generated content may be incorrect.**

Part of the unique functionality of the website is that it essentially “learns” from the frequency that users use individual ingredients in their recipes. Rather than implementing a “Favorites” functionality, the ingredients documents in those collections keep track of a number called “usage\_count”. When the ingredients are retrieved from the database they are displayed sorted on “usage\_count” descending. The more an ingredient is used, the higher on the list it will appear. Updating this usage\_count field is done with a PUT endpoint which will be called after each of the above PUT endpoints.

**Update Ingredient Document Incrementing “usage\_count” Endpoint**: PUT /api/:collection/:ingredient/increment

**Database Document Before:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Example Request:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Database Document After:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Example Error (404 Not Found):  
A screenshot of a computer

AI-generated content may be incorrect.**

Should a user decide to remove an ingredient from a recipe, they can click on that line in the recipe display and click the “Remove” button. When doing so, a DELETE endpoint removes the ingredient from the recipe.

**Delete Ingredient Endpoint:** DELETE /api/recipes/:id/ingredients

**Database Before:  
A screenshot of a computer code

AI-generated content may be incorrect.**

**Example Request:**

**A screenshot of a recipe

AI-generated content may be incorrect.**

**Database After:  
A screenshot of a computer code

AI-generated content may be incorrect.**

**Example Request (400 Bad Request):**

**A screenshot of a computer

AI-generated content may be incorrect.**

At any time, the user can add instruction steps to the recipe by typing them in the provided text entry box. When they click the “Submit Step” button, a PUT endpoint adds the instruction to the recipe.

**Add or Update Instructions to Recipe Document Endpoint:** PUT /api/recipes/:id/instructions

**Example Request:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Database Updated:  
A screenshot of a recipe

AI-generated content may be incorrect.**

**Example Error (404 Not Found):  
A screenshot of a computer program

AI-generated content may be incorrect.**

Users will also be able to delete lines of instructions in much the same way. This utilizes a DELETE endpoint when the user clicks the “Remove” button in the instruction list.

**Delete Recipe Instruction Endpoint:** DELETE /api/recipes/:id/instructions

**Database Before:  
A screenshot of a computer

AI-generated content may be incorrect.**

**Example Request:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Database After:  
A screenshot of a computer program

AI-generated content may be incorrect.**

**Example Error (400 Bad Request):**

**A screenshot of a computer

AI-generated content may be incorrect.**

When displayed, a “Delete” recipe button will be provided to the user. It will only function on recipes where the userId field matches that of the user.

**Delete Recipe Endpoint:** DELETE /api/recipes/:id

**Example Request:  
A screenshot of a computer

AI-generated content may be incorrect.**

**Example Error (404 Not Found):  
A screenshot of a computer

AI-generated content may be incorrect.**

**6. Conclusion**

This service layer design provides a clear separation between the frontend UI and MongoDB database, ensuring:

* **Consistency**: Defined endpoints for data access.
* **Scalability**: Easy future modifications.
* **Maintainability**: Clean API structure for interacting with recipes, users, and ingredients.

The endpoints follow RESTful principles, ensuring predictable behaviors for retrieving, creating, updating, and deleting data.