University of Calgary

CPSC 471 - Winter 2022
Data Base Management Systems

Paul Serafini (UCID# 30077288)
Zachary Adolphe (UCID# 10135943)
Tait Wiley (UCID# 10064664)

Group#62 Project Report

April 16th, 2022

Abstract

Often, people find themselves not just looking for a random recipe to make, but wondering what kind of recipe they can make (or nearly make) based on the leftover ingredients they currently possess. Our project, dubbed 'Tailor Recipes', reflects a robust design that tackles solving this real-life problem, incorporating a database referentially structured to permit such holistic ingredient-recipe querying, allowing in implementation the ability for general users to interact with stored recipes both extant and of their own design.

This work demonstrates a strong, functional understanding of the database management principles and mechanics taught in CPSC 471, notably featuring a powerful, capable design which incorporates modern middleware structures bridging a sharply streamlined database and an accessible front-end application.

Although some desired functionalities were unable to be implemented in full by the deadline attached to the projects of our course, the overarching form and resources employed by Tailor Recipes clearly allows for an effective future capture of our objective functionalities without requiring significant divergence from the present composition of our project.

1. INTRODUCTION

As described in the project proposal, the problem identified is a simple, if time consuming, problem which almost every household faces on a daily basis: how do I use those ingredients I already have to make enjoyable meals? The database was designed to solve this problem by providing an ingredient-based querying for recipes that contain specific ingredients searched by the user. This system contains recipes in the database posted publicly by the product users where those recipes contain some of those ingredients.

By working to allow users to making informed decisions about their recipes chosen via easy access to recipe, ingredient, and related grocery store information, this project attacks this problem in a highly effective way.

2. PROJECT DESIGN

2.1 USERS

Our project was designed with two user types in mind, one being a 'General User' and the other an 'Administrator'. The primary functionality of the former was designed with respect to allowing the viewing, creation, and favouriting of recipes, such users additionally able to maintain a basket of ingredients such that they could - largely hands-off - execute searches returning recipes consistent with some or all of those items which are present in their basket (and so ostensibly utilizing captured ingredients which are already immediately in their real-life possession). The primary functionality of the latter group, our administrators, is to have moderating control over both the

data posted by and the existence of our aforementioned general users.

Our implemented SQL database structure supports each class of user, with uniquely identifying values automatically assigned on account creation (and stored in a 'user' table) enabling individualized user referencing over a broad range of tables within the database, permitting a truly tailored user experience; in the case of general users, such account creation may be accomplished manually over the web API, whereas an administrative user would be utilizing a pre-existent account (i.e., instantiated by a back-end actor).

With respect to the eventual web-based implementation, time restrictions forced the paring down of our demonstration scope to reflect solely the first class of users. A strong majority of usage cases were covered (and subsequently exhibited in our live demonstration) by our submitted project, captured through transactions which will see described in detail shortly.

2.2 TRANSACTION COLLECTION

With respect to featured transactions, we have the following, implemented capabilities available to general users through our web-based application:

- Login Transactions:
  - Creation of a user, whereby a coupled username and password is added to the database which allow users to then log in
  - Logging in to the database, whereby a general user is able to gain access to primary functionality of the API
- Recipe Transactions:
  - Creation of a recipe, such that it is stored to the database and then able to be viewed (and reviewed!) by other users
  - Searching for a recipe or recipes, with users able to produce all database recipes by name-keyword association
  - Viewing and posting reviews (being a numerical score attached to a comment) for a recipe
  - Deleting a selected recipe, removing the recipe from the database
- Ingredient Transactions
  - Searching for an ingredient or ingredients, which returns a bevy of information relating to possible brands, prices, and grocery stores (which stock the) item

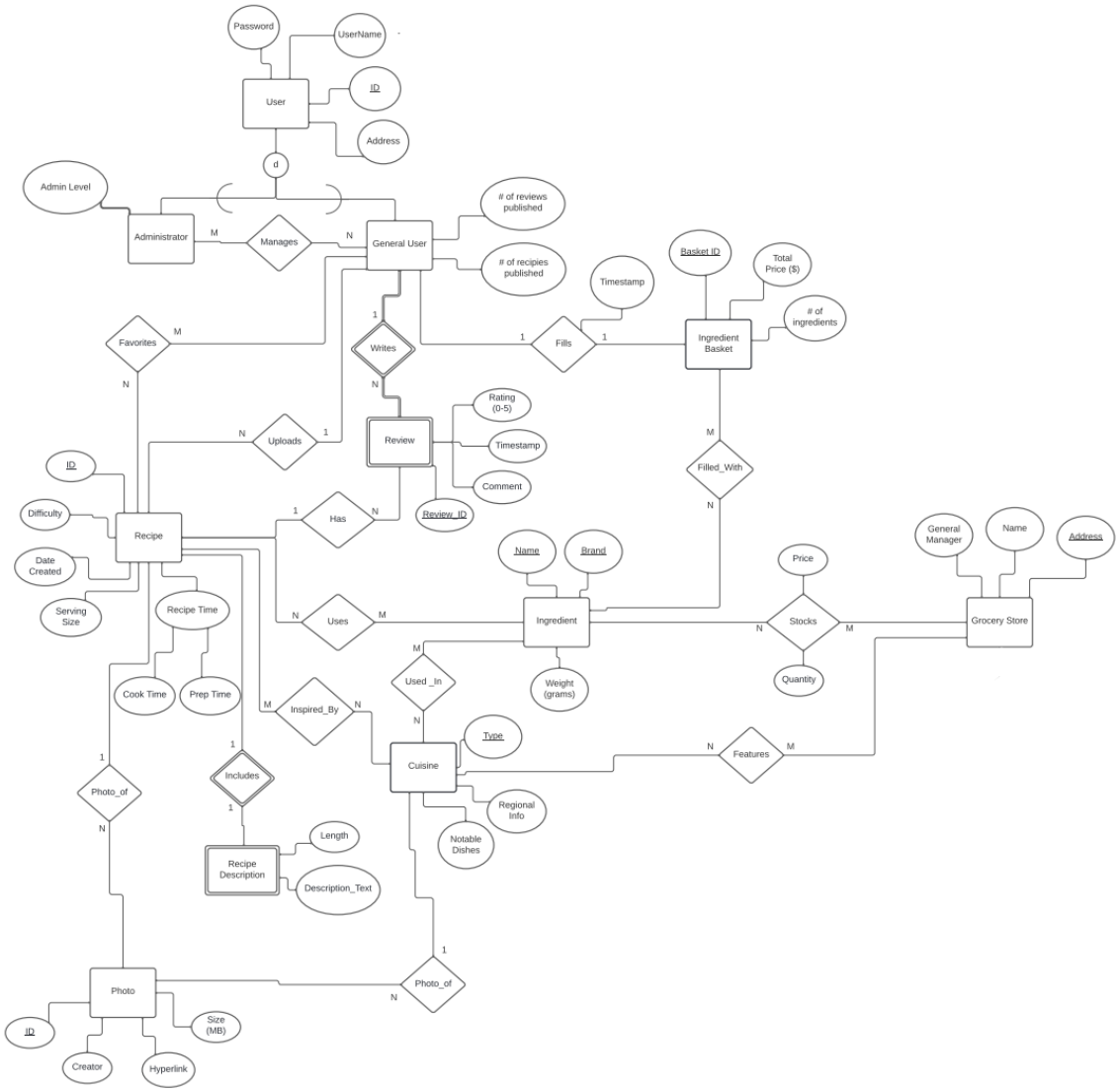
Note that detailed examples of each (with specific usage case information) is available in section 4.2.

Although *not* incorporated into the final version of the API, additional (functioning) endpoints were formulated towards the end of a more comprehensive solution for our proposal motive, a number of them serving as POC for a more ingredient-oriented implementation (an example below):

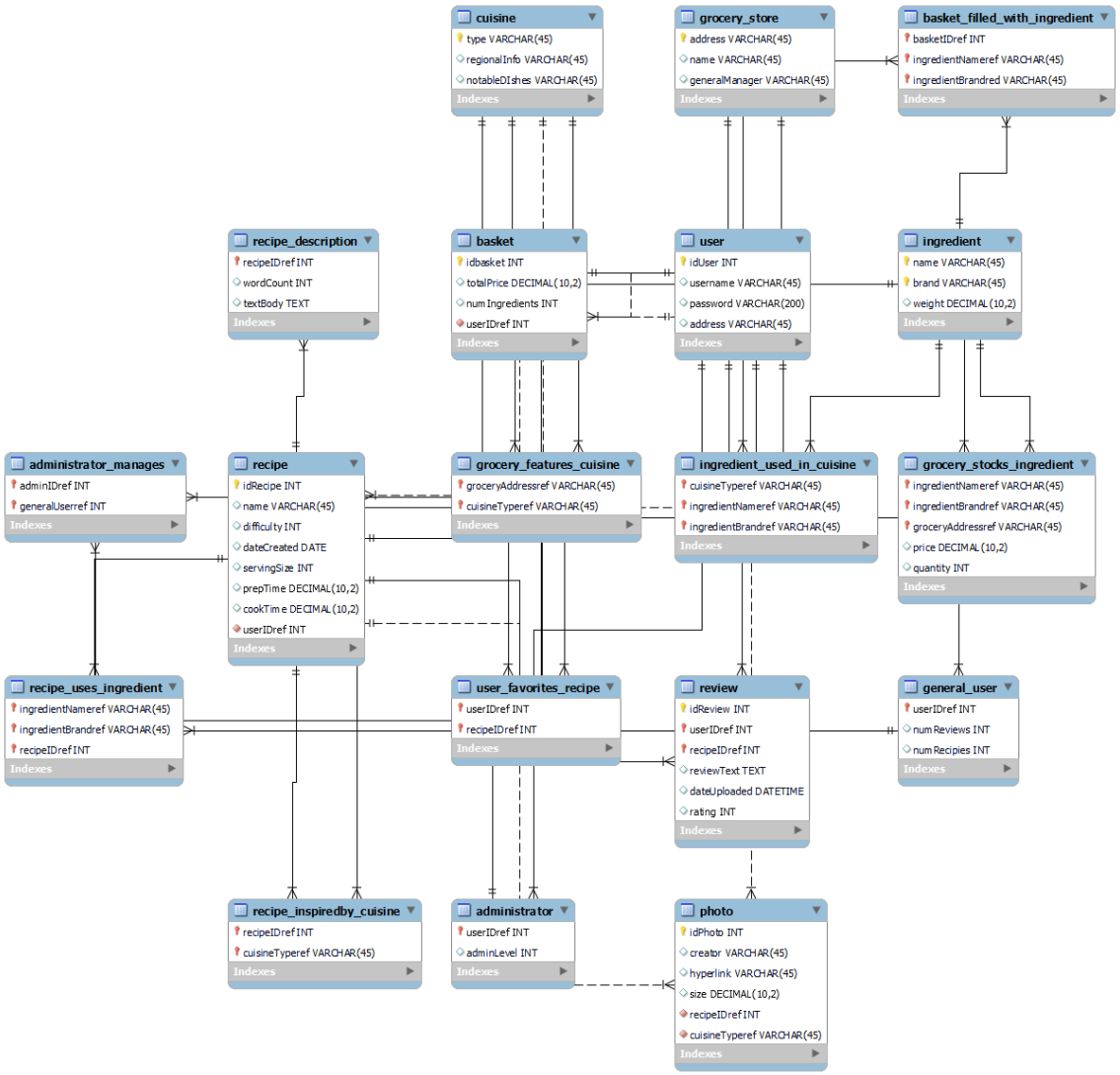
- A transaction which took the ID of a user as input and returned the ID of any recipes which incorporated ingredients from their ingredient basket

The SQL query regarding immediately above will be included near the end of section 3.2.

2.3 EERD


Password
UserName
User
ID
Address
d
Admin Level
Administrator
M
Manages
N
General User
# of reviews published
# of recipies published
Timestamp
Basket ID
Total Price ($)
# of ingredients
Ingredient Basket
Favorites
M
N
1
Writes
N
1
Fills
1
Rating (0-5)
Review
Timestamp
Comment
Review_ID
N
Uploads
1
ID
Difficulty
Recipe
Date Created
Serving Size
Recipe Time
Cook Time
Prep Time
1
Has
N
M
Filled_With
N
Name
Brand
Ingredient
Weight (grams)
N
Uses
M
Price
Stocks
Quantity
N
M
General Manager
Name
Address
Grocery Store
M
Used_In
N
M
Inspired_By
N
Cuisine
Type
Regional Info
Notable Dishes
N
Features
M
1
Photo_of
N
1
Includes
1
Recipe Description
Length
Description_Text
1
Photo_of
N
Photo
ID
Size (MB)
Creator
Hyperlink

Immediately below is our database EERD generated by MySQL Workbench:

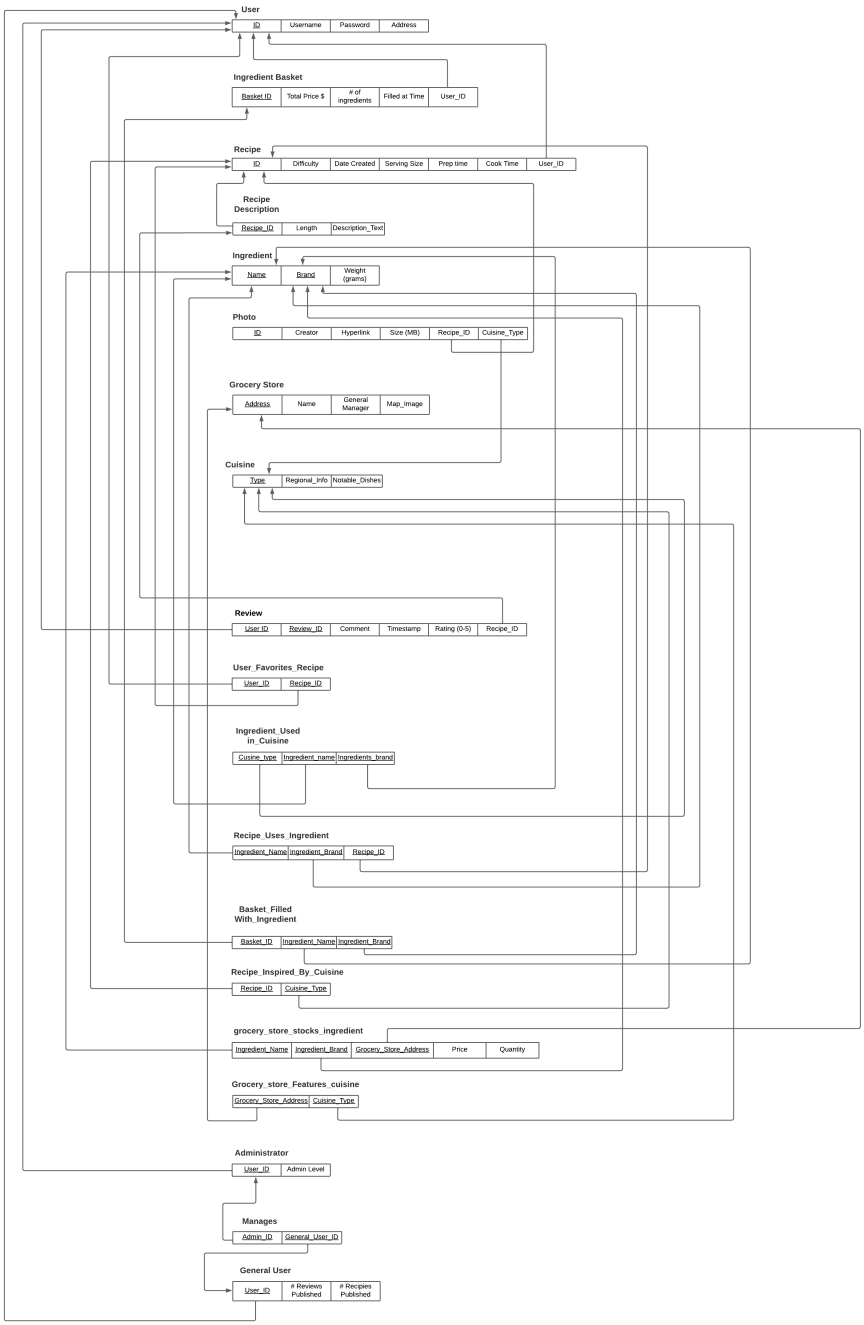


3. IMPLEMENTATION

3.1 RELATIONAL MODEL

Author's Note: our RM laptop died, so here is our RM from a prior submission.

Tait Wiley - 10064664
Paul Serafini - 30077288
Zach Adolphe - 10135943



3.2 DATABASE MANAGEMENT SYSTEM

The Database Management System (DBMS) that was chosen for this task and project was MySQL. MySQL is known for its high performance but also relatively simple ease of use database system for administrating relational data models and larger scale data systems. MySQL as its name suggests recognizes (Structured Query Language (SQL) for querying relational data which is very common in industry practices and is known for being reliable as a free open source DBMS.

What follows is a list of the SQL queries utilized by our API, organized by the HTML page which calls it in script and headed by information on the usage case of its containing controller; examples from the API of related results for each process may be found in section 4.2 (below):

```
>>> loginPage.html

    *** POST; AUTHENTICATES USER (username against hashed password)

        "SELECT password FROM user WHERE username = ?;"

    *** POST; RETRIEVES USER ID (to post after successful login)

        "SELECT idUser FROM user WHERE username = ?;",

>>> newUser.html

    *** GET; ENSURE USERNAME UNIQUE (check username)

        "SELECT username FROM user;"

    *** POST; INSERT INTO USER (AFTER HASHING PASSWORD)

        "INSERT INTO user (username, password, address) VALUES (?, ?, ?)"

>>> searchRecipe.html

    *** GET; FIND RECIPES (by name search)

        "SELECT * FROM recipe INNER JOIN recipe_description
        ON recipe.idRecipe = recipe_description.recipeIDref
        AND recipe.name LIKE ?;"

    *** GET; FIND RECIPE (by ID; after clicking on recipe)

        "SELECT * FROM recipe INNER JOIN recipe_description
        ON recipe.idRecipe = recipe_description.recipeIDref
        AND recipe.idRecipe = ?;"
```

```
    *** DELETE; DELETE RECIPE (by ID)

        "DELETE FROM recipe WHERE idRecipe = ?"

    *** GET; OBTAIN REVIEWS (by recipe ID)

        "SELECT reviewText, rating FROM review INNER JOIN
        recipe ON review.recipeIDref = recipe.idRecipe AND
        recipe.idRecipe = ?;"

>>> ingredientBasket.html

    *** GET; OBTAIN INGREDIENT INFO (by name; grocery stock info)

        "SELECT ingredientNameref, ingredientBrandref, name,
        groceryAddressref, price, quantity FROM grocery_stocks_ingredient INNER JOIN grocery_sto
        grocery_stocks_ingredient.groceryAddressref =
        grocery_store.address AND
        grocery_stocks_ingredient.ingredientNameref LIKE ?;";

>>> createRecipe.html

    *** POST; INSERT NEW RECIPE (mostly technical attributes)

        "INSERT INTO recipe (name, difficulty, servingSize,
        prepTime, cookTime, userIDref) VALUES (?, ?, ?, ?, ?, ?)"

    *** POST; ATTACH DESCRIPTION TO RECIPE (namely, its steps)

        "INSERT INTO recipe_description (recipeIDref,
        wordCount, textBody) VALUES (?, ?, ?)"
```

Unable to see final implementation, the following SQL query achieves a primary, driving objective of our project - to derive recipes from the ingredient basket of a user (by *userID* input:

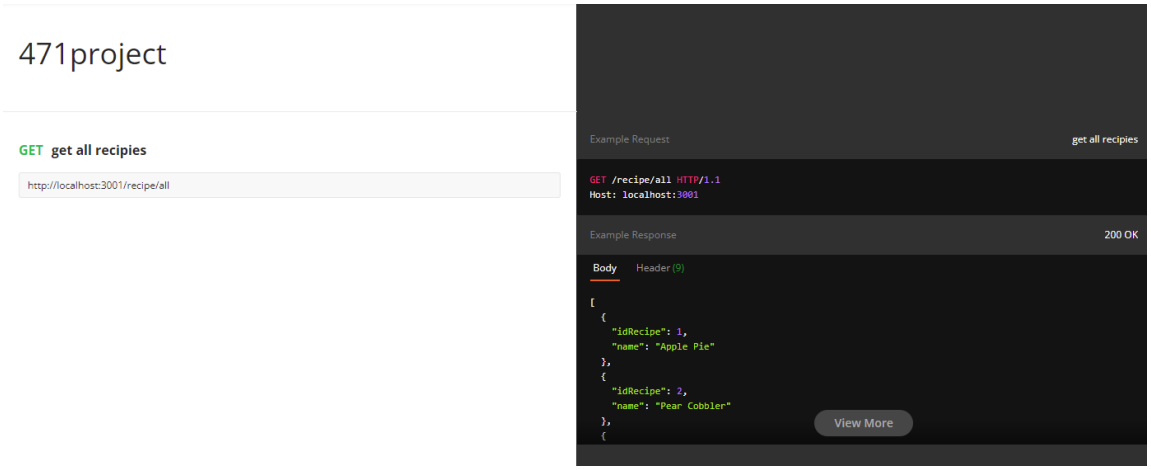
```
"SELECT DISTINCT recipeIDref FROM basket_filled_with_ingredient
INNER JOIN basket ON basket_filled_with_ingredient.basketIDref =
basket.idbasket INNER JOIN recipe_uses_ingredient ON
basket_filled_with_ingredient.ingredientNameref =
recipe_uses_ingredient.ingredientNameref WHERE basket.userIDref = ?";
```

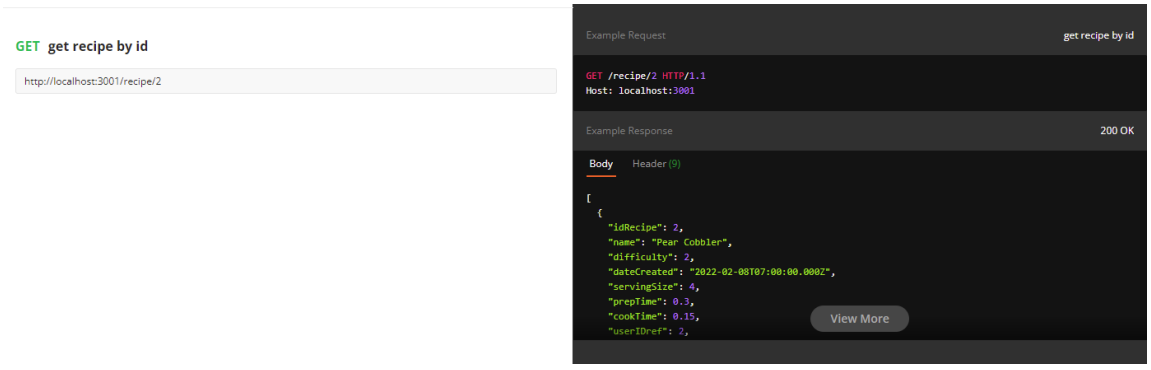
With respect to protecting against SQL injection, we took great care to keep our inputs parameterized, also employing relatively extensive input sanitizing via limiting of special characters, as well as some input validation; evidence of this is near-ubiquitous within our controller code.

3.3 API DOCUMENTATION

The basic structure of our API design is straightforward: we utilize a simple layout for our web interface, developed using the Fetch API[fch] to send requests and get responses and to send and receive JSON files both across the network and locally using the API endpoints; the JSON object returned from fetches is parsed by the JSON method and then displayed front-end for the user so as to allow the app to collect recipe names, ingredients, instructions, etc. from the database.

Below - with expandable contents - may be found published here.



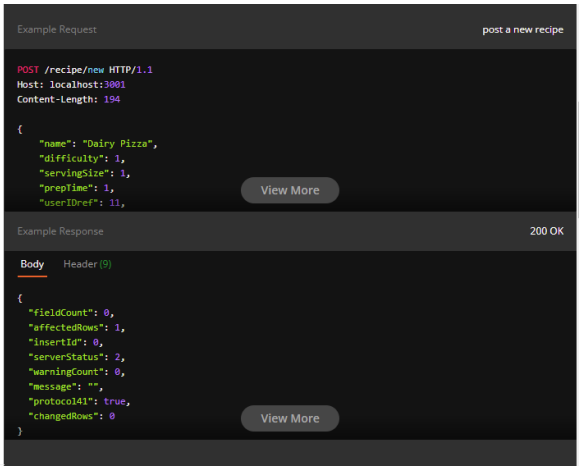


POST post a new recipe

http://localhost:3001/recipe/new

BODY raw


```
{
    "name": "Dairy Pizza",
    "difficulty": 1,
    "servingSize": 1,
    "prepTime": 1,
    "userIDref": 11,
    "wordCount": 120,
    "textBody" : "Make a new pizza mad from milk"
}
```

POST add ingredient to a recipe

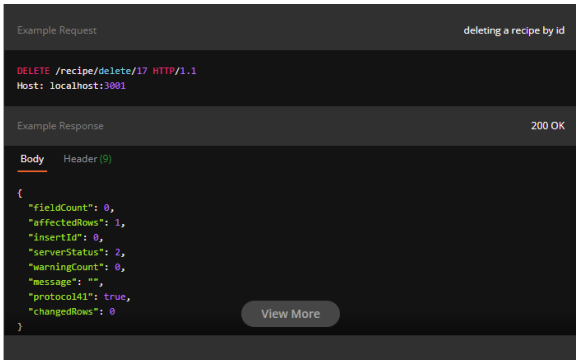
http://localhost:3001/recipe/ingredient

BODY raw


```
{
    "ingredientNameref": "carrot",
    "ingredientBrandref":"orgfarm",
    "recipeIDref": 17
}
```


DEL deleting a recipe by id

http://localhost:3001/recipe/delete/17



PUT Update a recipe description

http://localhost:3001/recipe/edit/description

BODY raw

```
{
    "wordCount": 333,
    "textBody": "THIS IS A NEW DESCRIPTION, UPDATED VIA UPDATE ENDPOINT",
    "recipeIDref": 1
}
```

Example Request — Update a recipe description

```
PUT /recipe/edit/description HTTP/1.1
Host: localhost:3001
Content-Length: 124

{
    "wordCount": 333,
    "textBody": "THIS IS A NEW DESCRIPTION, UPDATED VIA UPDATE ENDPOINT",
    "recipeIDref": 1
}
```

Example Response — 200 OK

Body Header (9)

```
{
  "fieldCount": 0,
  "affectedRows": 1,
  "insertId": 0,
  "serverStatus": 2,
  "warningCount": 0,
  "message": "(Rows matched: 1  Changed: 1  Warnings: 0",
  "protocol41": true,
  "changedRows": 1
}
```

View More

GET Get all brands of an ingredient

http://localhost:3001/ingredient/brands/apple

Example Request — Get all brands of an ingredient

```
GET /ingredient/brands/apple HTTP/1.1
Host: localhost:3001
```

Example Response — 200 OK

Body Header (9)

```
[
  {
    "brand": "orgfarm"
  }
]
```

POST make a new user (check if username unique as well)

http://localhost:3001/auth/newuser

BODY raw

```
{

    "username" : "testuser2",
    "password" : "hottomali",
    "address" : "tempaddress"

}
```

Example Request — make a new user (check if username unique as well)

```
POST /auth/newuser HTTP/1.1
Host: localhost:3001
Content-Length: 101

{

    "username" : "testuser2",
    "password" : "hottomali",
    "address" : "tempaddress"
```

Example Response — 400 Bad Request

Body Header (9)

```
{
  "fieldCount": 0,
  "affectedRows": 1,
  "insertId": 15,
  "serverStatus": 2,
  "warningCount": 0,
  "message": "",
  "protocol41": true,
  "changedRows": 0
}
```

View More

POST authenticate a user

http://localhost:3001/auth/authuser

BODY raw

```
{
    "username" : "user123",
    "password" : "hottomali"
}
```

Example Request — authenticate a user

```
POST /auth/authuser HTTP/1.1
Host: localhost:3001
Content-Length: 63

{
    "username" : "user123",
    "password" : "hottomali"
}
```

Example Response — 200 OK

Body Header (9)

```
"login successful"
```

GET getting all reviews of a recipe

http://localhost:3001/recipe/reviews/1

Example Request — getting all reviews of a recipe

```
GET /recipe/reviews/1 HTTP/1.1
Host: localhost:3001
```

Example Response — 200 OK

Body Header (9)

```
[
  {
    "reviewText": "Dis pie is bomb!",
    "rating": 5
  }
]
```

POST posting a review for a recipe

http://localhost:3001/recipe/review/new

BODY raw

```
{
    "userIDref" : 1,
    "recipeIDref": 30,
    "reviewText": "This pizza is not great",
    "rating": 5
}
```

Example Request — posting a review for a recipe

```
POST /recipe/review/new HTTP/1.1
Host: localhost:3001
Content-Length: 116

{
    "userIDref" : 1,
    "recipeIDref": 30,
    "reviewText": "This pizza is not great",
    "rating": 5
}
```

Example Response — 200 OK

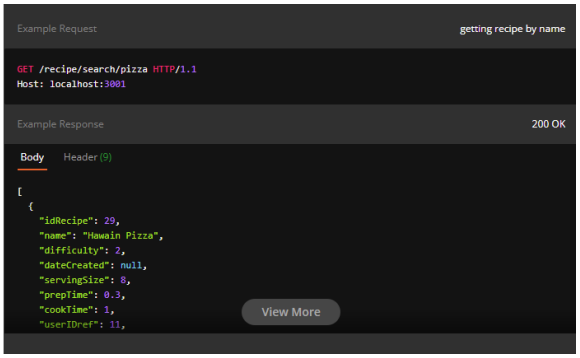
Body Header (9)

```
{
  "fieldCount": 0,
  "affectedRows": 1,
  "insertId": 29,
  "serverStatus": 2,
  "warningCount": 0,
  "message": "",
  "protocol41": true,
  "changedRows": 0
}
```

View More

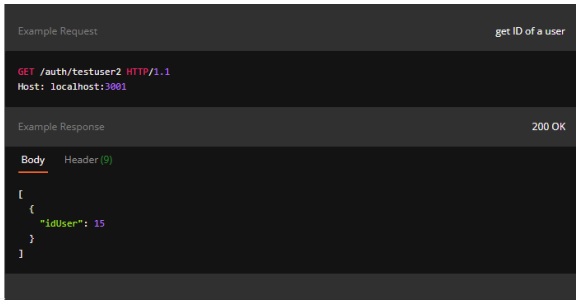
GET getting recipe by name

http://localhost:3001/recipe/search/pizza



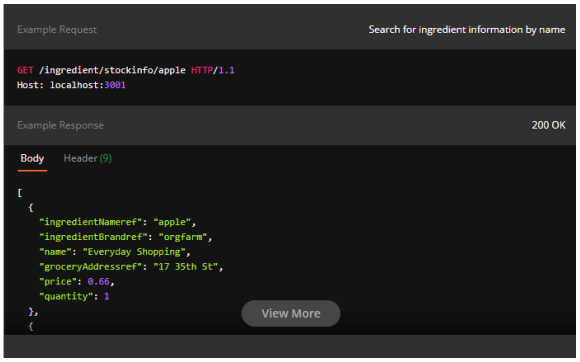
GET get ID of a user

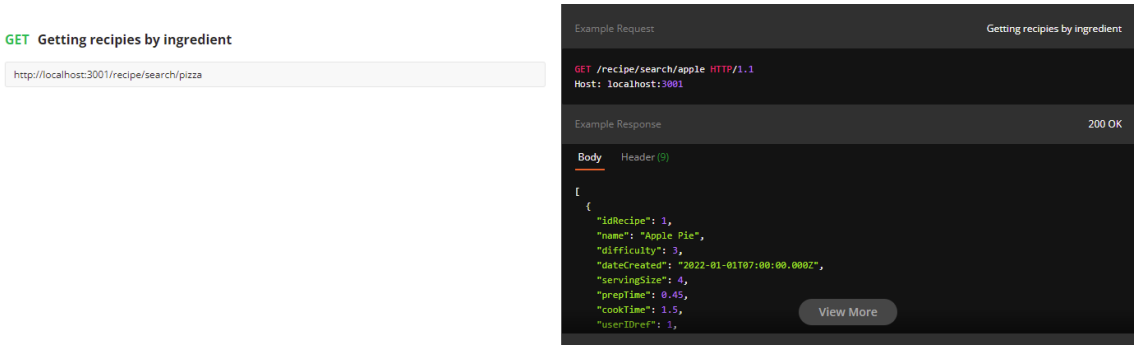
http://localhost:3001/auth/testuser



GET Search for ingredient information by name

http://localhost:3001/ingredient/stockinfo/apple





4. USER GUIDE

4.1 SETUP

Our project utilizes the following programs and tools, which *must* be installed prior to API use:

- MySQL
- Node.js
- npm (typically coupled with the Node.js install)

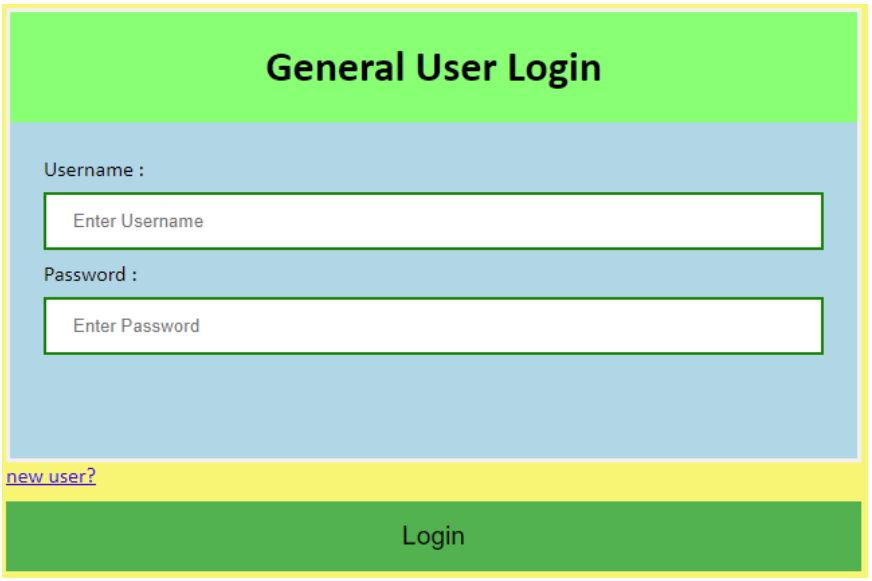
To set up our database, we employ the following protocol:

```
    1) open your MYSQL server
    2) ensure user is 'root' and host is 'localhost'
        (otherwise change these fields in database.js later)
    3) run the SQL query script 'recipe_db.sql' to get the DB
    4) run the SQL query script 'db_initial_data.sql
        to fill the DB with some initial data
    5) unzip 'api.zip'
    6) open the folder in your code editor
    7) open 'database.js' and change the 'password' field to your MYSQL server password
    8) navigate to .../api in your comman prompt/terminal
    9) run command 'node -v' to ensure you have an updated
        version of node; if not, get it!
    10) next, run command 'npm install' on 'body-parser',
        'cors', 'express', 'mysql', 'router', and finally
        'nodemon' (for easy app relaunching if desired)
    11) run 'node app.js' to launch the api server
    12) next, navigate to the 'public' folder in 'api'
-------- To Begin API Interaction --------
    13) right click on loginPage.html, then open with
        your web browser (was tested on firefox)
```

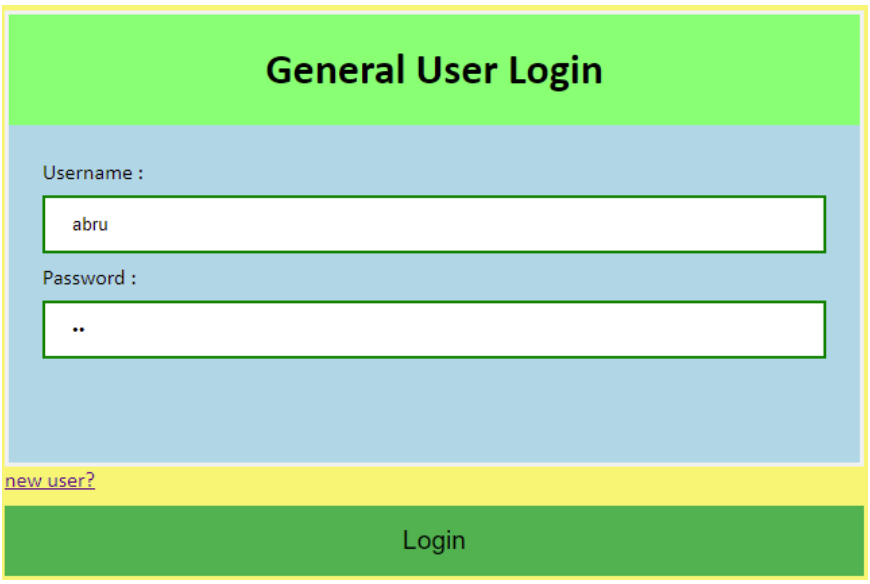
4.2 WEB INTERFACE FUNCTIONALITIES

What follows is a snapshot of each of the web-based application instances of our project, coupled with explanation or functional description as required; note that - for the sake of minimizing redundancy - in cases where a dark, rectangular button labelled *Back* exists (examples of this clearly visible below), clicking said button will return users to the page which would have previously redirected them to their current page in the course of normal usage.

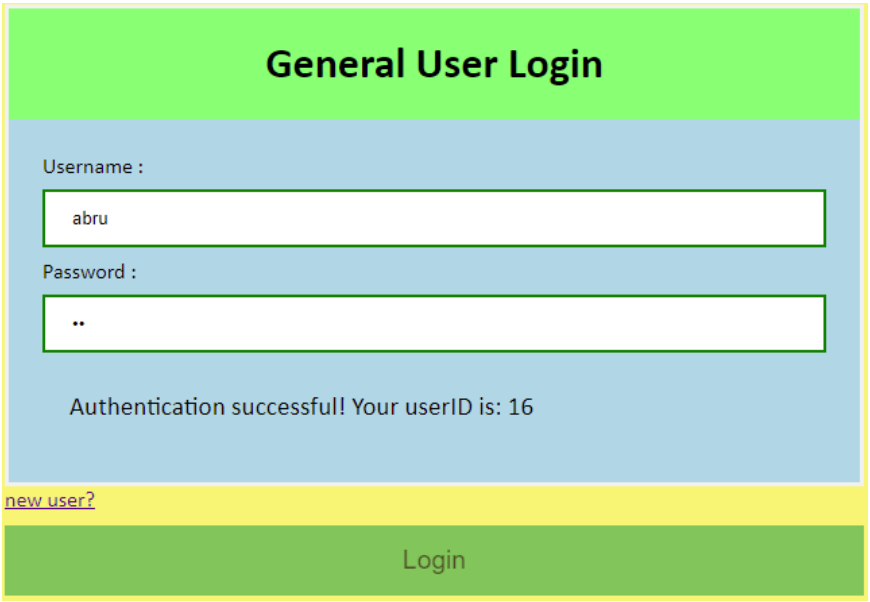
- **Logging In** (LOGINPAGE.HTML)
  - Upon initially loading the Tailor Recipes web application, one is greeted with a friendly, largely blue-green login screen, shown below:



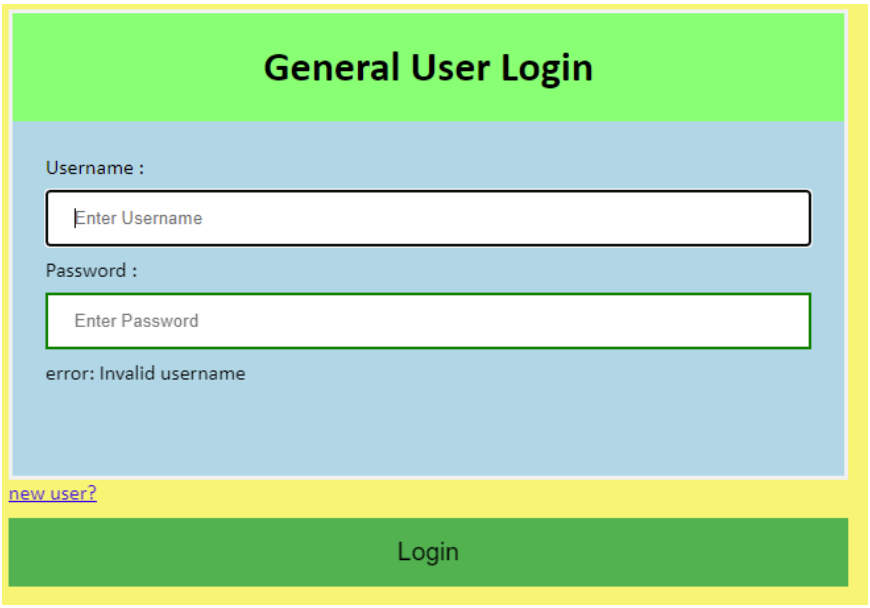
– Users are given the option to either sign in using pre-existing credentials, or to create a new user; in the case that they have a valid login already, they may enter their username in the *Username* field and their password in the *Password* field, then clicking *Login*:



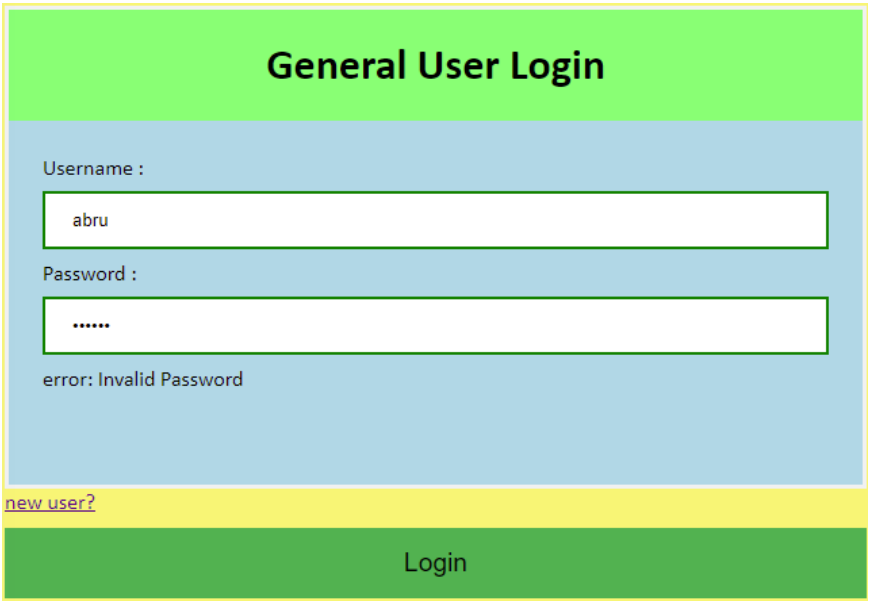
– If their credentials matches what is stored in our database (passwords are hashed!), they are informed of their successful authentication and provided their unique *userID* (which should be jotted down), noting users are after quickly redirected to HOMEPAGE.HTML):



– In the case that an invalid (not-stored) username is provided, an error statement will be returned visibly to that effect:

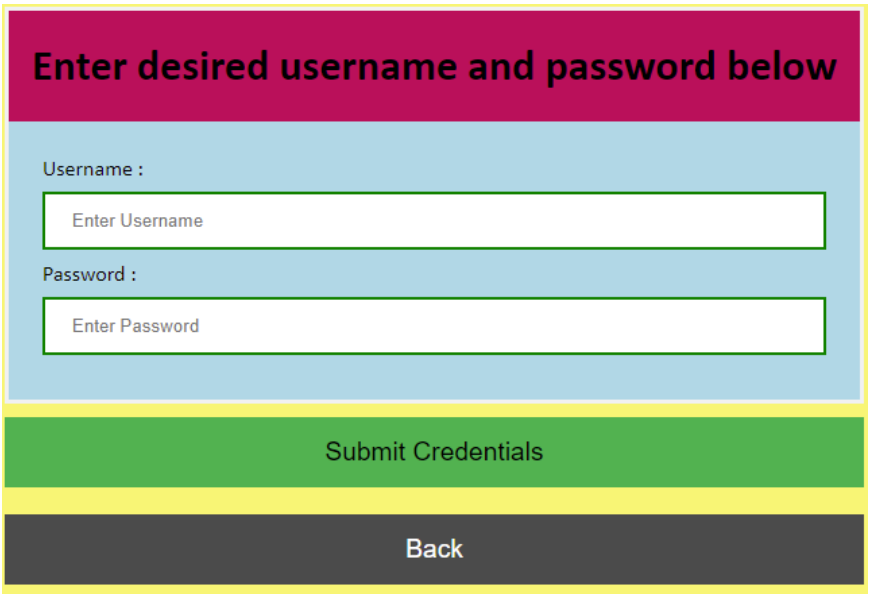


– In the case that the username exists but the password does not match the stored hash, an error statement is returned visibly to that effect as well:

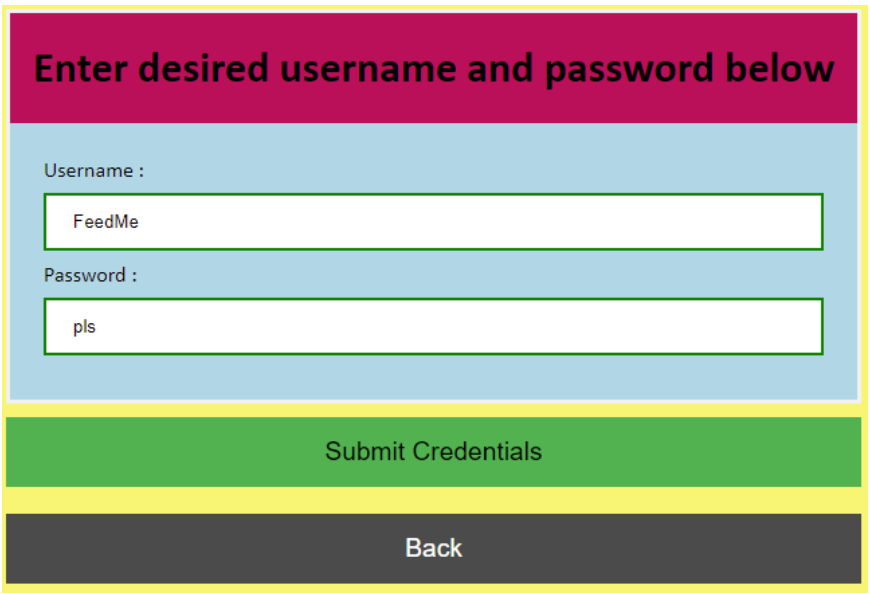


– As alluded to prior, users without valid login credentials may click *new user?* instead.

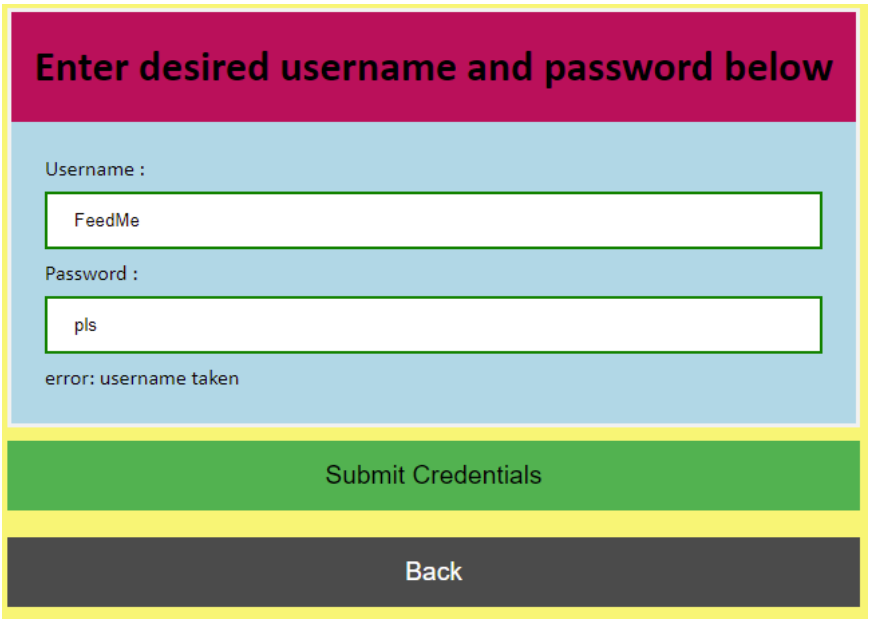
- **User Creation** (NEWUSER.HTML)
  - Upon clicking *new user?* at the start page, a user is redirected to the following:



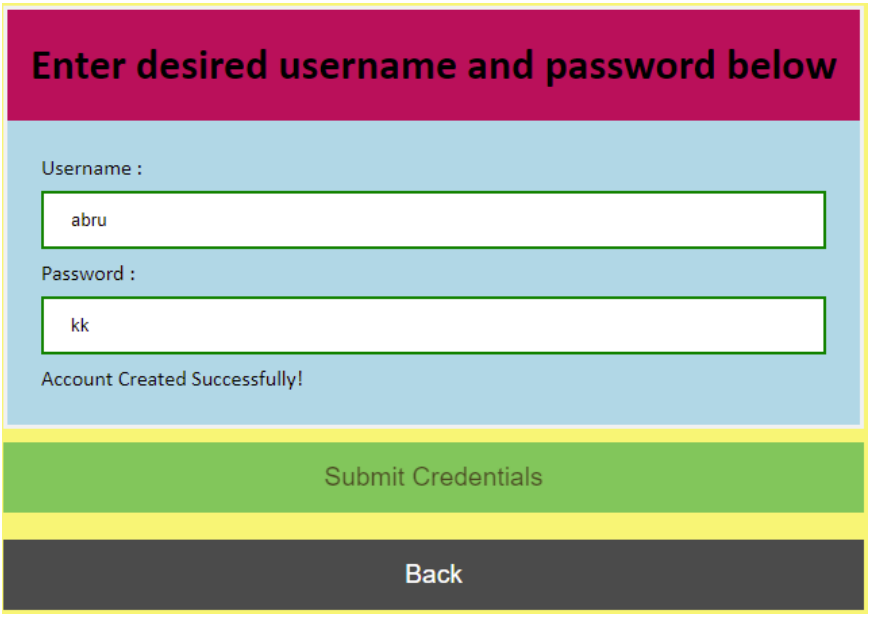
  - To create a new account, a user may enter a new username in the *Enter Username* field and a password in the *Enter Password* field, then click *Submit Credentials*:



– If the username is already taken, an error to that effect will be presented:

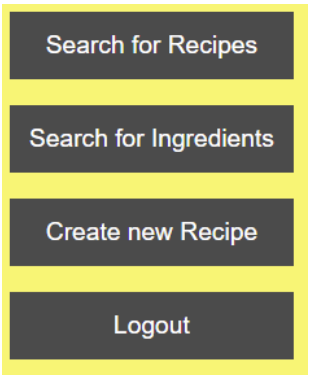


– However, if the username is new, then users will be notified of such in cheery terms:

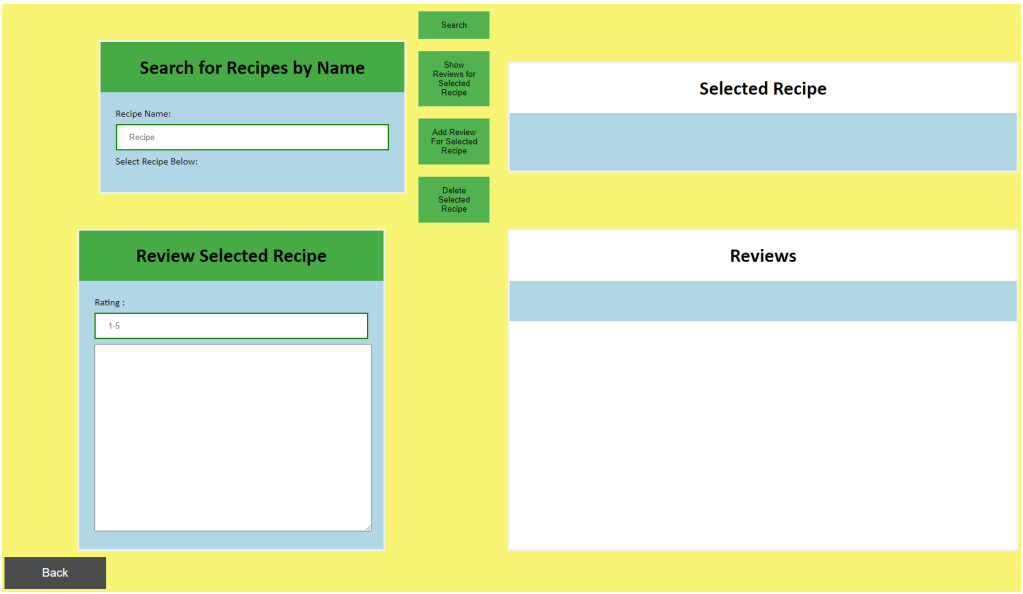


– Upon successful account creation, users will be redirected to the starting page again, whereupon they may now submit their newly-stored credentials!

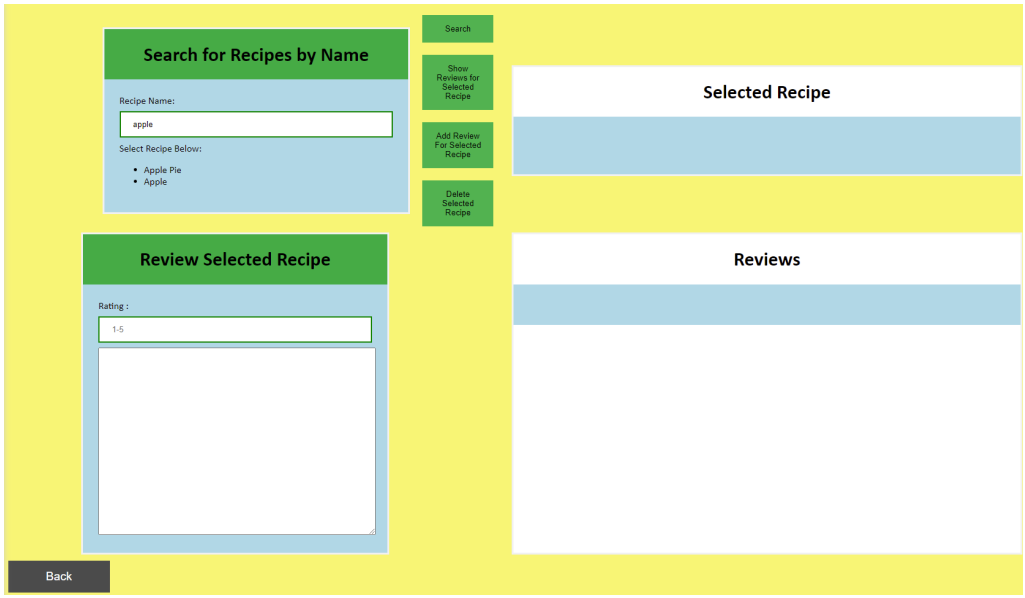
- **Home** (homePage.html)
  - Successfully logging in will redirect users to their custard-hued home page, where they are provided with a veritable smorgasbord of clickable options:



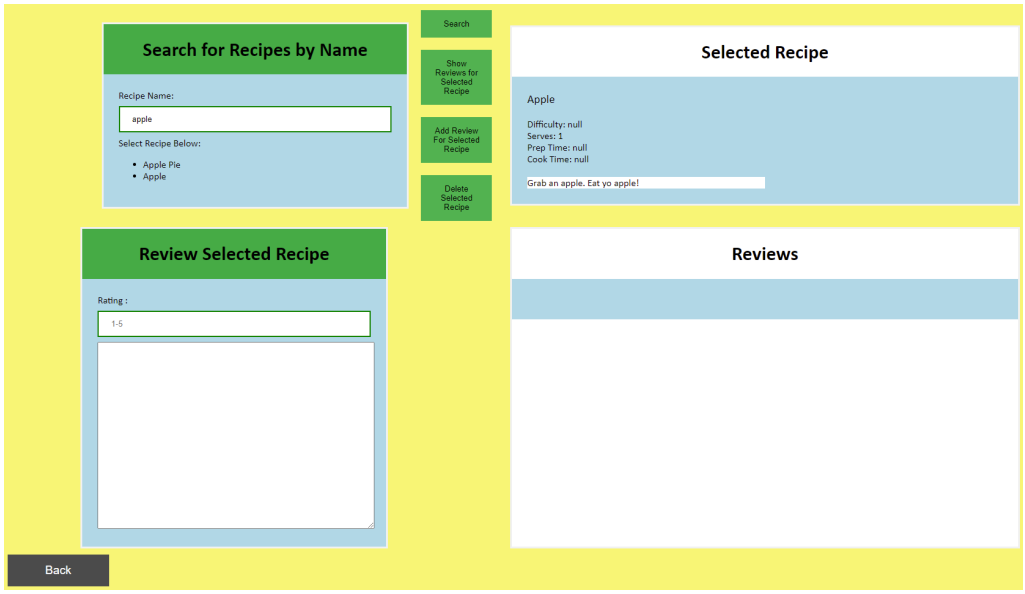
  - As the button names imply, clicking each button will redirect users to, respectively:
    - ***Search for Recipes*:** a page where users may interact with existing recipes;
    - ***Search for Ingredients*:** a page where information on ingredients existing in the database may be found;
    - ***Create new Recipe*:** a page where new recipes may be created and submitted;
    - ***Logout*:** the login page, right back where we very first started!
- **Recipe Search** (searchRecipe.html)
  - Here, we may search up recipes by keyword, show their related reviews, add a review to a selected recipe, and even deleted a selected recipe:



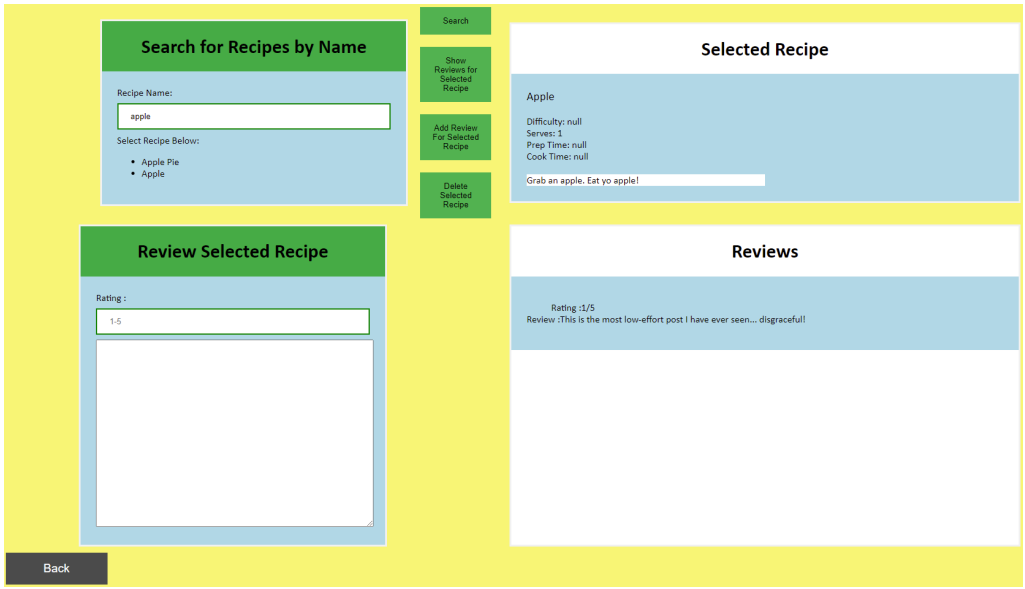
– By inputting a (not case-sensitive) string into the *Recipe* field and then clicking *Search*, users may obtain any recipes whose names contain the search key, as seen below:



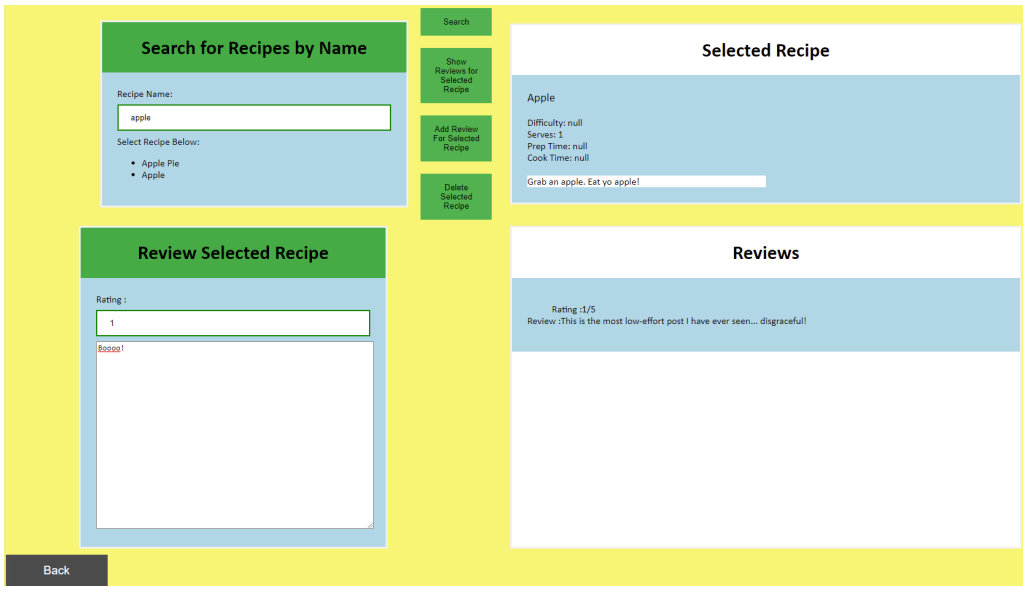
– Clicking one of the returned recipe names will load up its associated recipe, as with below (upon clicking *Apple*):



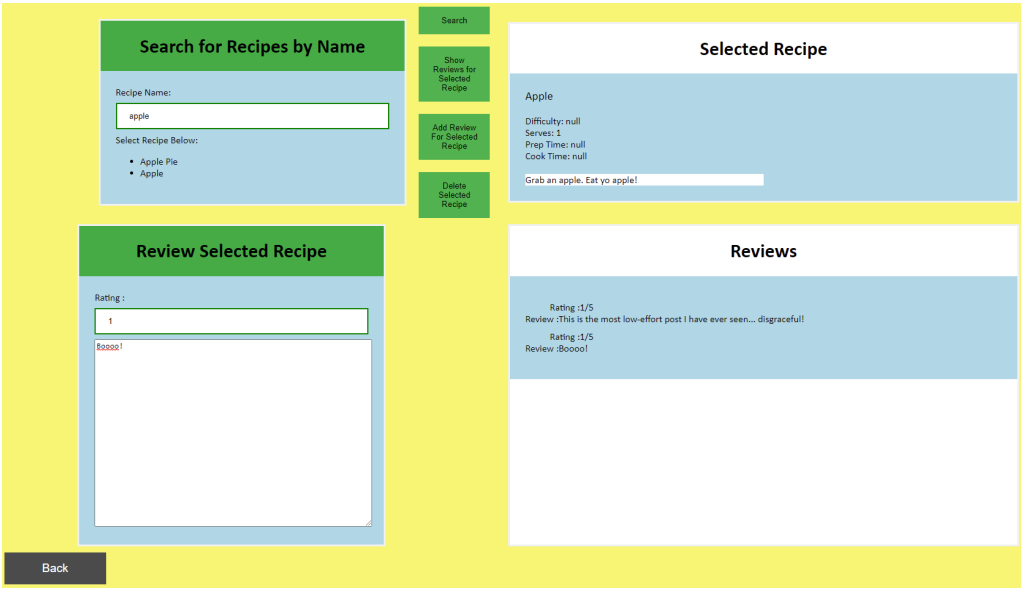
– If a user wishes to see the reviews for the just-selected recipe, they may click the button *Show Reviews for Selected Recipe*, and see:



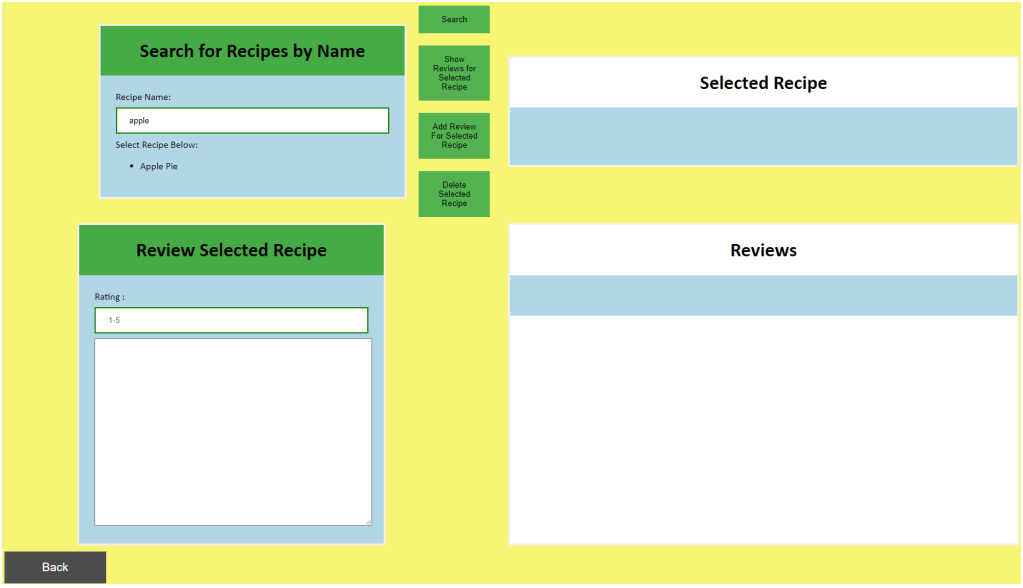
– If, instead, a user wishes to write a review for a selected recipe, they must fill in the *Rating* field with a numerical (integer) score from 1 to 5, as well as write in the field below a textual review of the recipe, finally clicking *Add Review For Selected Recipe*, altogether as such:



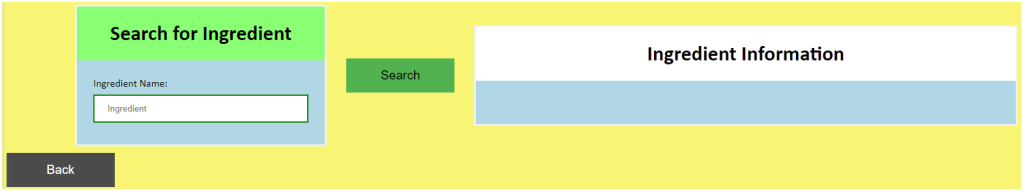
– Notably, these last two steps may occur in any order, although the just-written review will only show up under *Reviews* for the selected recipe if the *Show Reviews for Selected Recipe* button is clicked <u>after</u> review submission; if completed thusly, users would see:



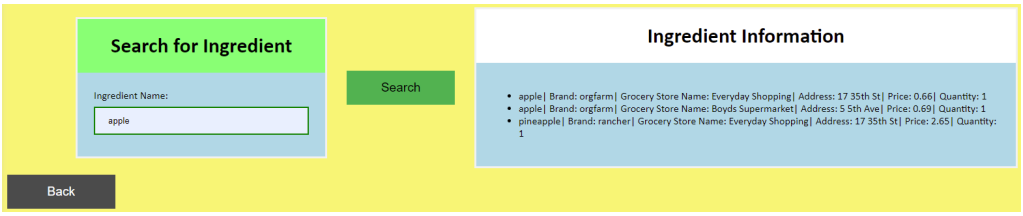
– Finally, by clicking *Delete Selected Recipe* after selecting a recipe, a user deletes the selected recipe from the database, whereupon future searches will not return that recipe:



- **Ingredient Search** (ingredientBasket.html)
  - This page enables users to search for ingredient information via inputting a (not case-sensitive) string into the *Ingredient Name* field and then clicking *Search*, with the initial page seen below:



  - After searching for an ingredient as described above, it will return a list of its database-stored details, each element being uniquely identified by name, brand, and grocery store information:



  - No change will visibly occur (i.e., no results returned) if a searched-for ingredient does not exist in the database.
  - Note also that any returned piece of ingredient information may - at present - only be viewed (whereby it is advised to not try to click the text describing ingredient or store names in the same way one would the recipe names shown earlier oversearchRecipe.html).

- **Recipe Creation** (createRecipe.html)
  - This page allows users to submit a recipe of their creation which shall be stored in the database by filling out each field (utilizing their transcribed *userID* from earlier) and then clicking *Submit Recipe*; we observe the following input for a stellar recipe:

  - Note that any field having been filled out with invalid type will result in an error being thrown and their poor excuse for a recipe not being inserted into the database.

5. CONCLUSION

In all, our project exhibits a strong technical framework for a database and API implementation, its development serving to appreciably (and - we believe - correctly!) deepen our skill-set with respect to future database design and implementation.

Although certain features were unable to make our final product, that we feel confident in implementing these remaining functionalities given a relatively small, additional amount of time speaks volumes to the level of growth this project has nurtured in us; we fully anticipate continuing with this project to allow it to reach its deserving final form.

REFERENCES

[fch] https://developer.mozilla.org (Accessed April 12th 2022).