Recipe Search Engine

By: CS320 Group 2

Names: Daniel Kim, Aayush Koirala, Eric Gu, Bradly Pacheco

Group #: 2

Manager: Bhoomika

Frameworks

Fronted: React

Backend: Express

User Story: Inclusion of Ingredients

The user wants to find a recipe with the ingredients included in the search. The user selects the ingredients that they want to be included in the search and selects a search button. The system will search for any recipe that includes the ingredients and displays it to the user.

User Story: Inclusion of ONLY Ingredients

The user wants to find recipes which only include the ingredients that they enter to the search bar. The user enters said ingredients into the search bar. The user clicks the search buttons. The ingredients are queried to the database and the recipes that match are displayed to the user. The displayed recipes should only include the ingredients that the user entered. If none of the recipes exist, then nothing is displayed.

User Story: Exclusion of Ingredients

The user wants to find a recipe without specific ingredients when searching for a recipe. The user selects the ingredients that they want to be excluded in the search and selects a search button. The system will search for any recipe that does not include the ingredients and displays it to the user.

FetchRecipes(string ingredients, boolean only, boolean include)

Note: If a user wants multiple ingredients, we will call this function with the different ingredients multiple times.

Function/Output: Calls to the MealDB-API and fetches recipes using .fetch() function.

Parameters:

- The “only” flag should be set to true iff the user ONLY wants the ingredients provided, otherwise false.
- The “ingredient” string is the ingredient that the user wants. If a user enters more than one string, we would take more than one.
- The “include” flag should be set to true iff the user wants to use those ingredients, if they want to exclude the ingredients, it should be false.

Output: A JSON file with a key “meals” which has a list of objects. Each object has "strMeal" (name of meal), "strMealThumb” (URL to thumbnail of meal) and, "idMeal" (meal id)

Testing: For testing, we can hardcode the backend to test that the frontend is calling the backend correctly.

renderFetchRecipes(data)

Parameters: Takes in a parameter called data, which is a JSON object. It has a key “meals” which has a list of objects. Each object has "strMeal" (name of meal), "strMealThumb” (URL to thumbnail of meal) and, "idMeal" (meal id)

Output: Returns HTML that renders the recipes in a list format

Testing: For testing, we can hardcode the backend to mimic the frontend retrieving the data and check if the frontend renders the correct data.

User Story: Save recipe

The user finds a recipe that they like or want to use later. The user selects a favorite icon next to the recipe that they want to save for later. The system will save the recipe to the user’s profile, so the user can come back to it again.

saveRecipe(number userID, string recipeName) - Frontend

Function/Output: Sends userID and recipeName data to the the ExpressJS backend API using .fetch() and POST method.

Parameters:

- The userID should be the id of the current user.
- recipeName should come from the recipe that was indicated to be favorited on the frontend.

Output: Returns nothing, should send the data to the backend

Testing: To test if the frontend is sending data to the backend, we can select a new favorite recipe and check if the new favorite recipe appears in the user profile.

saveRecipe(number userID, string recipeName) - Backend

Function/Output: Sends userID and recipeName data to the Favorites database.

Parameters:

- The userID should be the id of the current user.
- recipeName should come from the recipe that was indicated to be favorited on the frontend.

Output: Returns nothing, should save recipe to SQL database

Testing: To test if the frontend is sending data to the backend, we can select a new favorite recipe and check if the new favorite recipe appears in the user profile.

fetchSaveRecipe(number userID) - Backend

Function/Output: Requests userID from oracle database, getting the result.

Parameters: Takes in the current userID as a parameter.

Output: Returns a JSON file containing all the favorite recipes from that userID, ideally by sending a SQL query to the backend

Testing: To test if the backend is sending data to the frontend, we can hardcode the backend to send a user profile with some favorited recipes and check if those favorites show up in the frontend.

fetchSaveRecipe(number userID) - Frontend

Function/Output: Frontend Function: Requests userID from oracle database/backend and returns HTML to frontend.

Parameters: Takes in the current userID as a parameter.

Output: Returns a JSON file containing all the favorite recipes from that userID, ideally displaying them to the user.

Testing: To test if the backend is sending data to the frontend, we can hardcode the backend to send a user profile with some favorited recipes and check if those favorites show up in the frontend.

User Story: Report recipe

The user finds a recipe that is “undesirable” (incorrect or doesn’t make any sense) and wants to report it. The user clicks a flag icon next to the recipe, and the user can type in a reason, if they want to. The system sends a report to the developers about the flagged recipe.

ReportRecipe(string recipeName)

Function/Output: Sends an email/notification to the dev email, **devforrecisearch@gmail.com**, using .fetch() POST method

Parameters: recipeName should come from the recipe that was indicated to be reported on the frontend.

Output: Nothing, Ideally sends an email to the email address given.

Testing: For testing, we can do a test report of a recipe and see if the correct recipe with any additional information is sent to the dev email.

Thank you