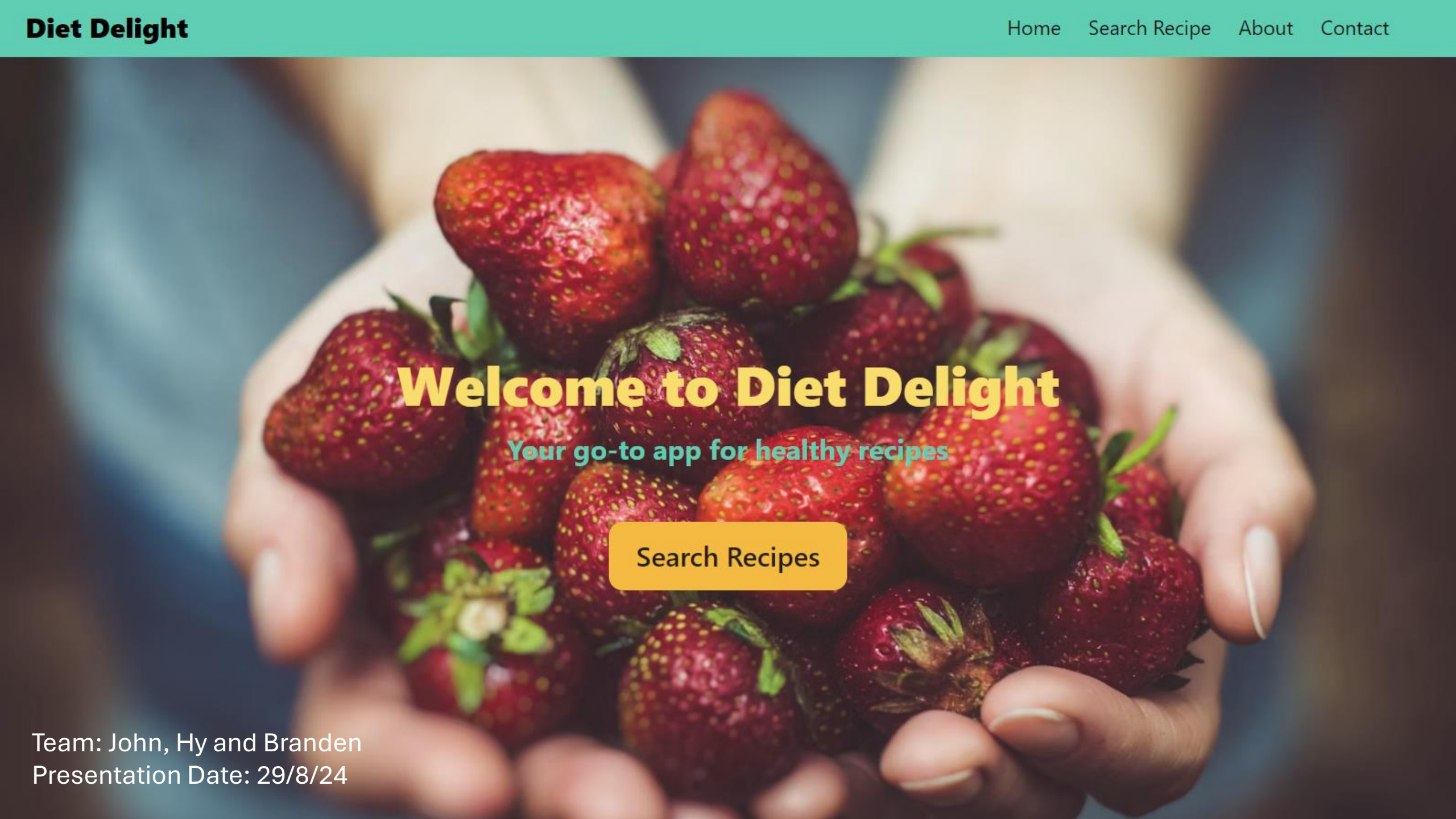


Team: John, Hy and Branden
Presentation Date: 29/8/24

Description

Fit Life Gym (hypothetical client) wants to offer additional value to its members by providing personalized meal planning and healthy eating guidance. Gym instructors need **a tool to help gym members select meals that complement their fitness goals**, such as muscle gain, weight loss, or improved endurance.

Functionality and Features

1. **Search Recipes (MVP)**

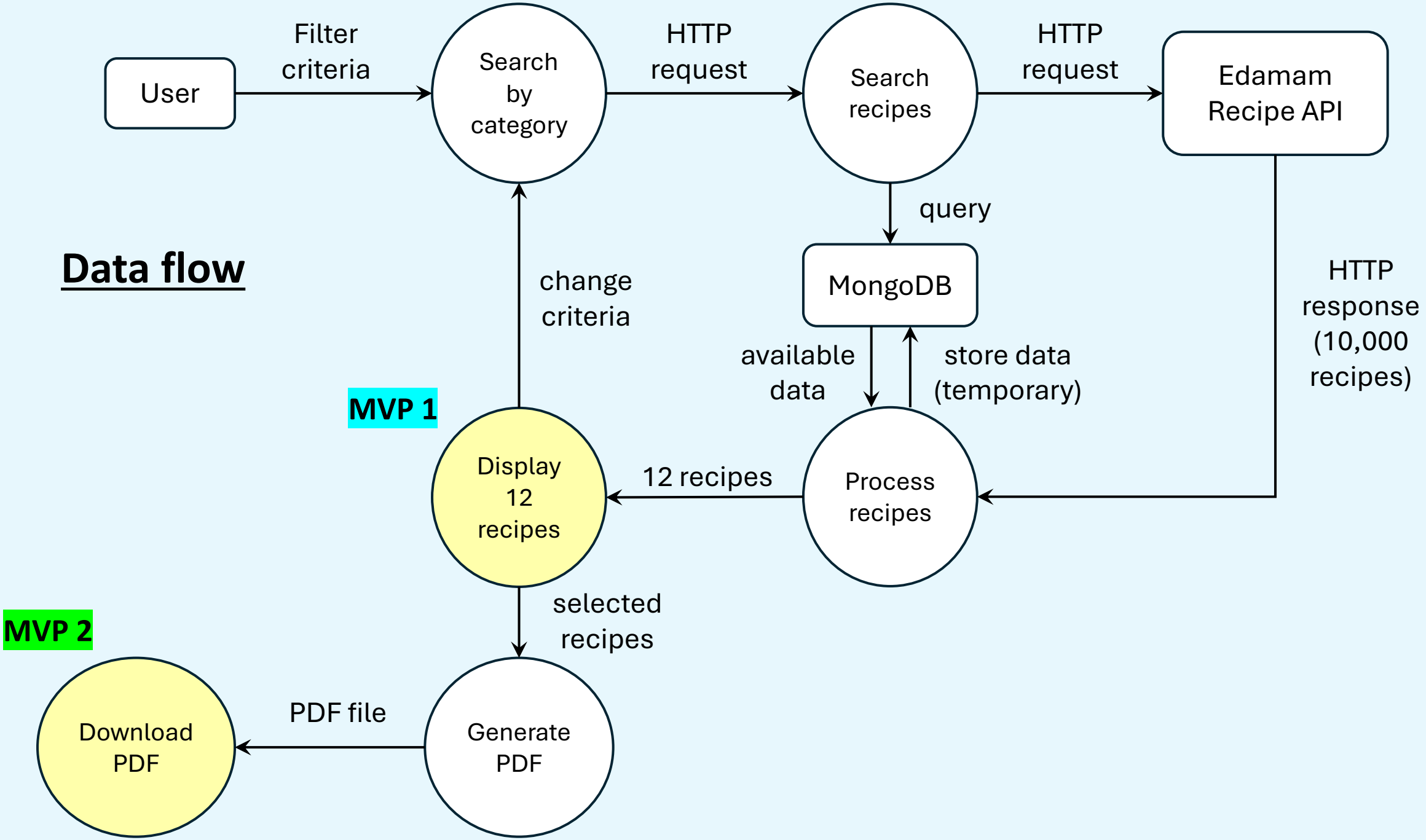
   Users can search for recipes by selecting dietary categories aligned with their fitness goals (e.g., high-protein for muscle gain, low-carb for weight loss).

2. **Print Recipes (MVP)**

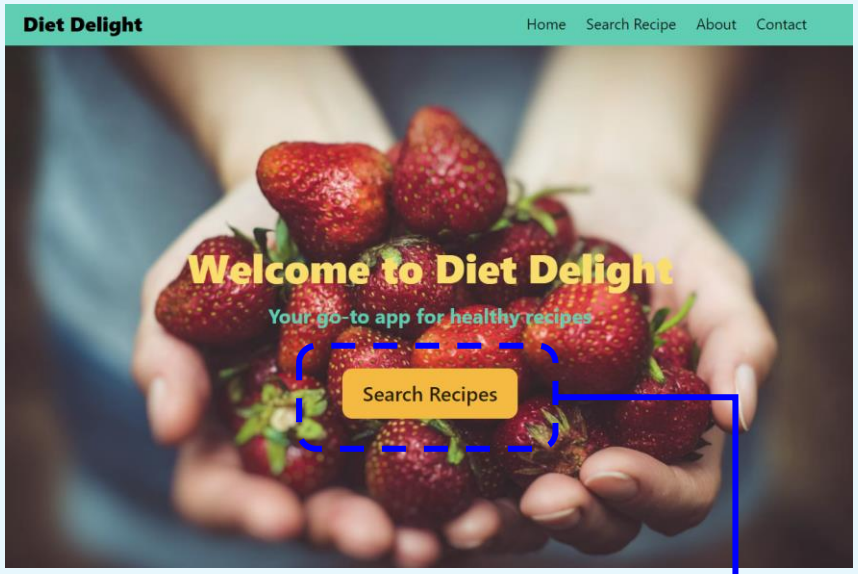
   Users can select and print recipes in PDF format, which provides a handy grocery list, or save the file for easy access.

3. **Featured Recipes (MVP)**

   Access to a curated list of recommended recipes to explore and choose appealing options.


Data flow
User
Filter criteria
Search by category
HTTP request
Search recipes
HTTP request
Edamam Recipe API
query
MongoDB
HTTP response (10,000 recipes)
change criteria
available data
store data (temporary)
MVP 1
Display 12 recipes
12 recipes
Process recipes
selected recipes
MVP 2
Download PDF
PDF file
Generate PDF

MVP 1



Landing page

Dietary Selection page

User can select one or more criteria

Frontend

Dietary Selection component

```
<div className="columns mt-5 is-flex is-flex-direction-column is-justify-content-center is-align-items-center has-text-centered">
    <CheckboxGroup
        title="Dietary Preferences:"
        options={dietOptions}
        selectedOptions={dietCriteria}
        onChange={(event) => handleCheckboxChange(event, 'diet')}
    />
    <CheckboxGroup
        title="Health Considerations:"
        options={healthOptions}
        selectedOptions={healthCriteria}
        onChange={(event) => handleCheckboxChange(event, 'health')}
    />
</div>
```

This section of the component creates two groups of checkboxes using the CheckboxGroup component: one for dietary preferences (dietOptions) and one for health considerations (healthOptions). **It allows users to select multiple options** from each group. **When a checkbox is checked or unchecked, the handleCheckboxChange function updates the state** (dietCriteria or healthCriteria) to reflect the current selections.

Search Recipe Button component

```
return (
    <div className="is-flex is-justify-content-center">
        {/* Button to trigger search */}
        <button className="button is-warning is-medium mt-5 mb-5" id="btn-search" onClick={handleSearch}>Search</button>
    </div>
)
```

This section of the component **renders a button labeled "Search".** The button has an **onClick event handler (handleSearch) that triggers a search function when clicked,** allowing users to fetch recipes based on the selected dietary and health criteria.

Exported function

Frontend

```
export const fetchRecipes = async (dietCriteria, healthCriteria, setIsLoading, setRecipes) => {
    // Start the loading before fetching
    setIsLoading(true);

    try {
        // Construct URL with query parameters
        const queryParams = new URLSearchParams();

        // Append dietary criteria to query parameters
        dietCriteria.forEach(diet => queryParams.append('diet', diet));
        // Append health criteria to query parameters
        healthCriteria.forEach(health => queryParams.append('health', health));

        // Fetch recipes from the API with the constructed query parameters
        const response = await fetch(`https://diet-delight-backend.onrender.com/recipes?${queryParams.toString()}`);

        if (!response.ok) {
            // Throw an error if response is not OK
            throw new Error('Network response was not ok');
        }

        const data = await response.json();
        // Update the state with the fetched recipes
        setRecipes(data.recipes);

        // Simulate a delay (for development or demonstration purposes)
        await new Promise(resolve => setTimeout(resolve, 800));
    } catch (error) {
        console.error('Error fetching recipes:', error);
        // Update the state with an empty array in case of an error
        setRecipes([]);
    } finally {
        // Ensure loading state is turned off regardless of success or failure
        setIsLoading(false); // End loading after a delay
    }
};
```

The main purpose of the fetchRecipes function is **to asynchronously fetch a list of recipes from an API based on selected dietary and health criteria.** It builds a query string using the provided criteria, makes a network request to retrieve the data, updates the state with the fetched recipes, and manages the isLoading state (loading spinner) to provide feedback to the user while the data is being fetched. If an error occurs during the fetch process, it logs the error and resets the recipes state to an empty array.

Recipe Controller ("/recipes" route)

Backend

```
43      // Check the database for available recipes
44      let recipes = await Recipe.find(query, { _id: 0, __v: 0 }).exec();
45      let dataFetchedFrom;
46
47      if (recipes.length < 12) {
48          // Fetch recipes from the Edamam API
49          const queryString = dietCriteriaArrayLower.map(diet => `diet=${encodeURIComponent(diet)}`).join('&') +
50                                  '&' +
51                                  healthCriteriaArrayLower.map(health => `health=${encodeURIComponent(health)}`).join('&');
52          const data = await fetchRecipes(`&${queryString}`);
53
54          // Check if 'hits' property exists and is not an empty array in the API response
55          if (data && data.hits && data.hits.length > 0) {
56                  console.log('Recipes fetched from Edamam API service provider.');
57                  dataFetchedFrom = 'Edamam API'
58
59              // Slice the first 100 recipes
60              const fetchedRecipes = data.hits.slice(0, 100).map(hit => ({
61                  title: hit.recipe.label, // Recipe title
62                  image: hit.recipe.image, // Recipe image URL
63                  source: hit.recipe.source || 'Unknown', // Source of the recipe
64                  instructionsUrl: hit.recipe.url || 'No URL available', // URL for recipe instructions
65                  dietLabels: hit.recipe.dietLabels.map(label => capitalizeWords(label)) || [], // Normalize to match MongoDB format
66                  healthLabels: hit.recipe.healthLabels.map(label => capitalizeWords(label)) || [], // Normalize to match MongoDB format
67                  ingredients: hit.recipe.ingredientLines || [], // List of ingredients
68                  servingSize: hit.recipe.yield !== undefined ? hit.recipe.yield : null, // Number of servings
69                  caloriesPerServing: hit.recipe.calories !== undefined ? hit.recipe.calories / hit.recipe.yield : null, // Calories per serving (kcal)
70                  totalTime: hit.recipe.totalTime !== undefined ? hit.recipe.totalTime : null, // totalTime = prep time + cooking time (in minutes)
71                  cuisineType: hit.recipe.cuisineType || [], // e.g. Australian, Italian, Japanese
72                  mealType: hit.recipe.mealType || [], // e.g. breakfast, lunch, dinner
73                  dishType: hit.recipe.dishType || [], // The food category (e.g., main course, salad, soup)
74                  totalNutrients: hit.recipe.totalNutrients || {}, // Nutritional information
75              }));
```

This checks the MongoDB database for recipes that match the user's dietary and health criteria. If sufficient recipes are found in the database, it uses those directly and logs the source.

If fewer than 12 matching recipes are found in the database, it fetches additional recipes from Edamam API (thru fetchRecipes function), normalizes the data, saves new recipes to the database, and updates the recipe list to include these fetched results.

Exported function

Backend

```
// Function to fetch recipes based on a query parameter
const fetchRecipes = async (query) => {
    const url = `https://api.edamam.com/api/recipes/v2?type=public&app_id=${process.env.APP_ID}&app_key=${process.env.APP_KEY}${query}`;

    try {
        const response = await fetch(url);
        if (!response.ok) {
            throw new Error(`HTTP error! status: ${response.status}`);
        }
        return await response.json();
    } catch (err) {
        console.log('Error fetching data from Edamam API:', err);
    }
};
```

This function makes an asynchronous GET request to the Edamam API using dynamically constructed query parameters and authentication credentials to retrieve recipe data. It handles potential errors by checking the response status and logging any issues that arise during the fetch operation.

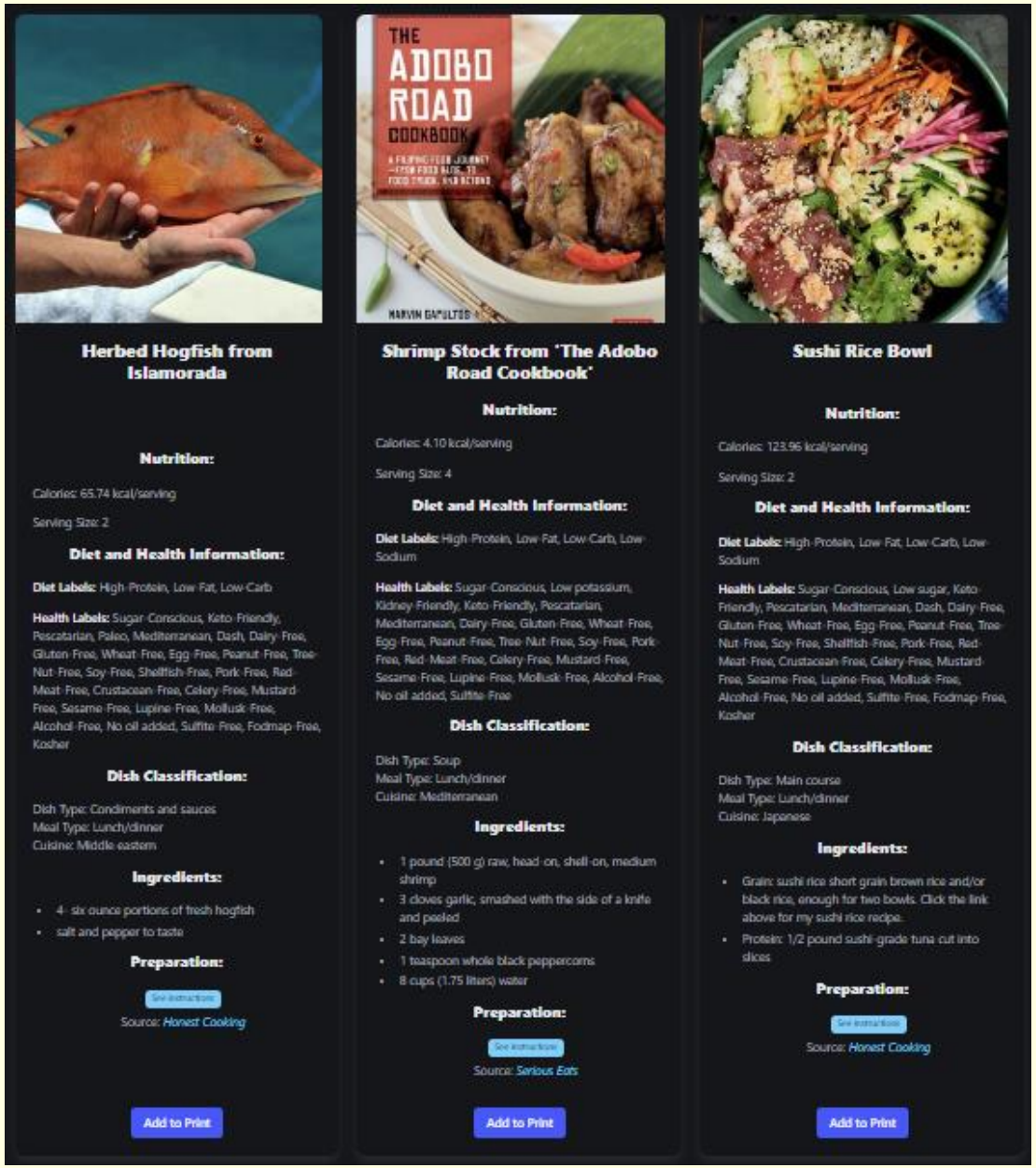
HTTP Response

Frontend

Headers | Payload | Preview | Response | Initiator | Timing


```
▼ [{title: "Pressure-Cooker Octopus Recipe",…}, {title: "Grilled Shrimp Cocktail",…},…]
  ▼ 0: {title: "Pressure-Cooker Octopus Recipe",…}
      caloriesPerServing: 185.73
    ▶ cuisineType: ["caribbean"]
    ▶ dietLabels: ["High-Protein", "Low-Fat", "Low-Carb"]
    ▶ dishType: ["main course"]
    ▶ healthLabels: ["Sugar-Conscious", "Low Sugar", "Keto-Friendly", "Pescatarian", "Paleo", "Mediterranean", "DASH",…]
      image: "https://edamam-product-images.s3.amazonaws.com/web-img/416/416123b29c87f07477df9ddc975f6e18.jpg?X-Amz-Securi
    ▶ ingredients: ["1 (2 1/2-pound; 1kg) whole octopus, rinsed well (including inside head cavity)", "Kosher salt"]
      instructionsUrl: "https://www.seriouseats.com/pressure-cooker-octopus-recipe"
    ▶ mealType: ["lunch/dinner"]
      servingSize: 6
      source: "Serious Eats"
      title: "Pressure-Cooker Octopus Recipe"
    ▶ totalNutrients: {ENERC_KCAL: {label: "Energy", quantity: 1114.3799999999999, unit: "kcal"},…}
      totalTime: 25
  ▶ 1: {title: "Grilled Shrimp Cocktail",…}
  ▶ 2: {title: "Sushi Rice Bowl",…}
  ▶ 3: {title: "Quick, Concentrated Shrimp Stock",…}
  ▶ 4: {title: "Lobster thermidor",…}
  ▶ 5: {title: "Ebi Shio (Shrimp Salt)",…}
  ▶ 6: {title: "Bonito Crudo",…}
  ▶ 7: {title: "Cook the Book: Canned Tuna",…}
  ▶ 8: {title: "Fried Calamari with Curry",…}
  ▶ 9: {title: "Shrimp Cocktail with Three Sauces",…}
  ▶ 10: {title: "Smoked haddock & hollandaise bake with dill & caper fried potatoes",…}
  ▶ 11: {title: "Homemade Salt Cod",…}
```

Search results



Shrimp Stock from 'The Adobo Road Cookbook'

Nutrition:

Calories: 4.10 kcal/serving

Serving Size: 4

Diet and Health Information:

Diet Labels: High-Protein, Low-Fat, Low-Carb, Low-Sodium

Health Labels: Sugar-Conscious, Low potassium, Kidney-Friendly, Keto-Friendly, Pescatarian, Mediterranean, Dairy-Free, Gluten-Free, Wheat-Free, Egg-Free, Peanut-Free, Tree-Nut-Free, Soy-Free, Pork-Free, Red-Meat-Free, Celery-Free, Mustard-Free, Sesame-Free, Lupine-Free, Mollusk-Free, Alcohol-Free, No oil added, Sulfite-Free

Dish Classification:

Dish Type: Soup
Meal Type: Lunch/dinner
Cuisine: Mediterranean

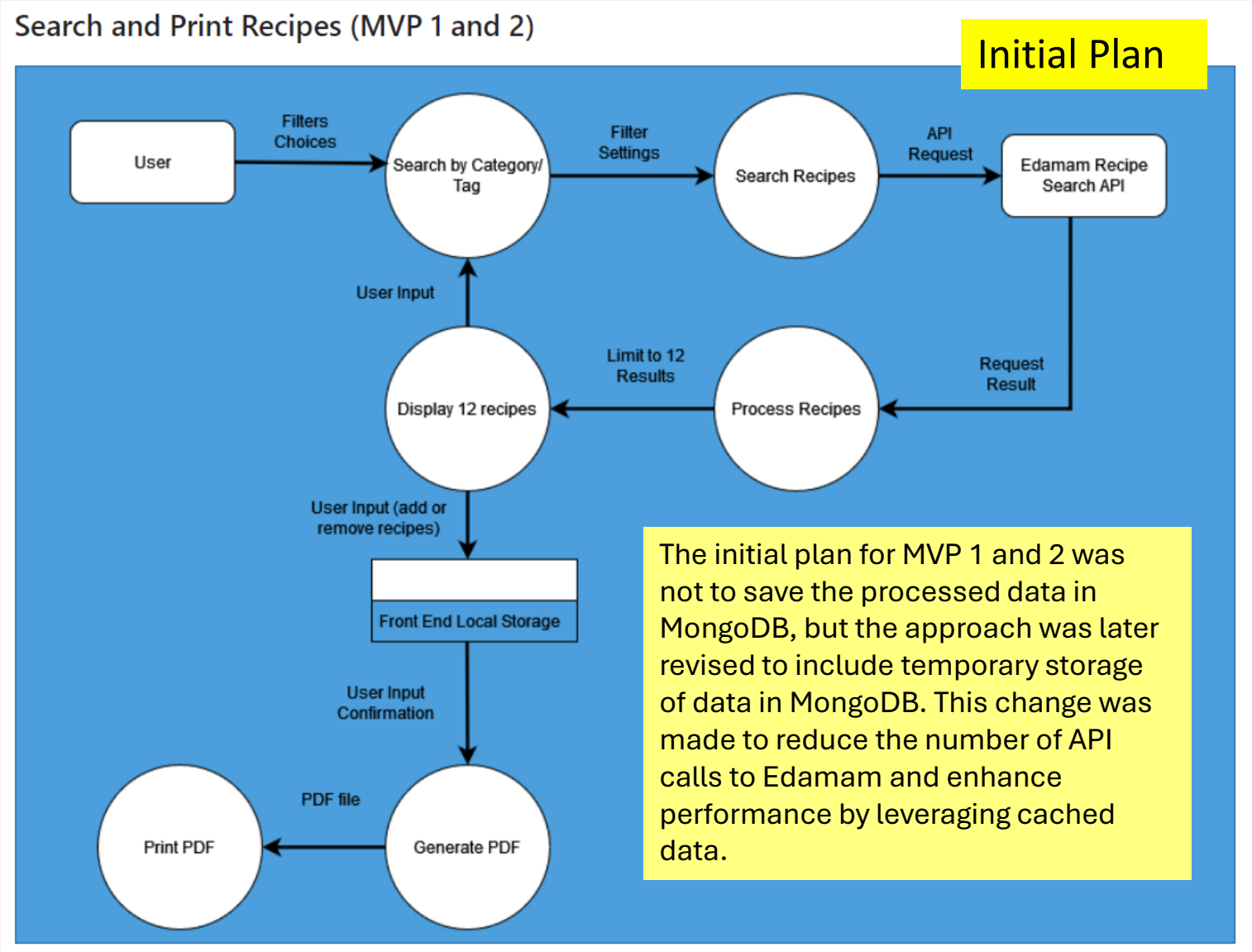
Set up auto delete of data in MongoDB to comply with Edamam's policy

Backend

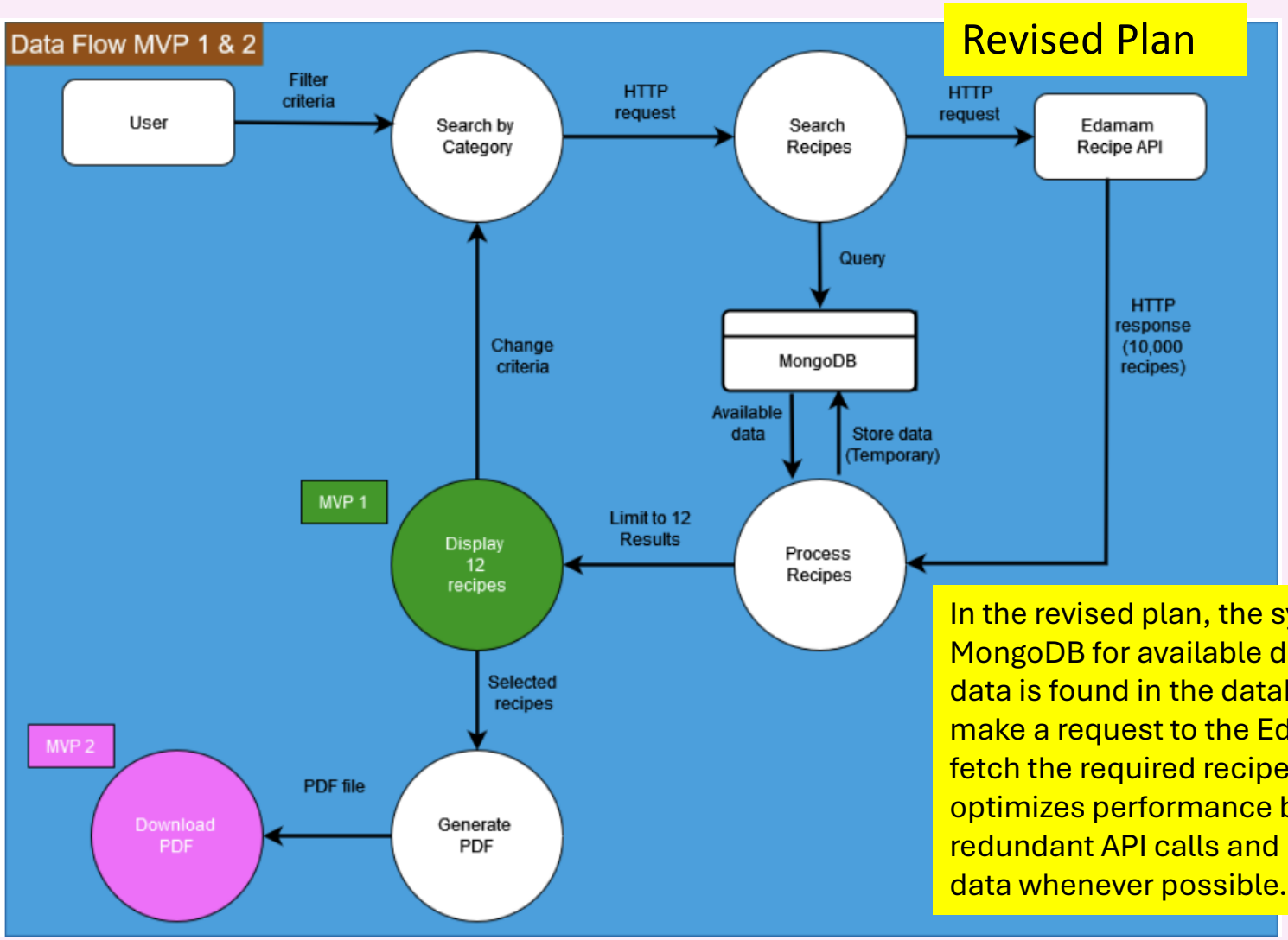
```
// Schema for recipe information
const recipeSchema = new mongoose.Schema({
    title: { type: String, required: true }, // Recipe title
    image: { type: String, required: true }, // Recipe image UR
    source: { type: String, required: true }, // Source of the recipe
    instructionsUrl: { type: String, required: true }, // URL for recipe instructions
    dietLabels: [{ type: String }], // Commonly used nutrient level aspects of the recipe.
    healthLabels: [{ type: String }], // Commonly used ingredient level aspects of the recipe.
    ingredients: [{ type: String }], // List of ingredients
    servingSize: { type: Number }, // Number of servings
    caloriesPerServing: { type: Number }, // Calories per serving (kcal)
    totalTime: { type: Number }, // totalTime = prep time + cooking time (in minutes)
    cuisineType: [{ type: String }], // e.g. Australian, Italian, Japanese
    mealType: [{ type: String }], // e.g. breakfast, lunch, dinner
    dishType: [{ type: String }], // The food category (e.g., main course, salad, soup)
    totalNutrients: [nutrientSchema], // Nutritional information
    createdAt: { type: Date, default: Date.now } // Timestamp when the recipe is created
});

// Create a TTL index to automatically delete documents after 10 minutes (600 seconds)
recipeSchema.index({ createdAt: 1 }, { expireAfterSeconds: 600 });

// Create a model for the Recipe schema
const Recipe = mongoose.model('Recipe', recipeSchema);

export default Recipe;
```

This defines a timestamp field, createdAt in MongoDB schema, which automatically records the creation time of each document with a default value of the current date and time. It also creates a TTL (Time-To-Live) index on this field, ensuring that **documents are automatically deleted from the database 10 minutes** after their creation. This approach helps **to comply with Edamam API's policy on data retention.**

Problem Encountered #1



Problem Encountered #1



In the revised plan, the system first checks MongoDB for available data. If insufficient data is found in the database, it will then make a request to the Edamam API to fetch the required recipes. This approach optimizes performance by reducing redundant API calls and utilizing cached data whenever possible.

Problem Encountered #1

```
// Handler to fetch and process recipes based on selected diet criteria
const getRecipes = async (req, res) => {
    try {
        // Extract diet and health query parameters from the request
        const dietCriteria = req.query.diet || [];
        const healthCriteria = req.query.health || [];

        // Ensure dietCriteria and healthCriteria are arrays
        const dietCriteriaArray = Array.isArray(dietCriteria) ? dietCriteria : [dietCriteria];
        const healthCriteriaArray = Array.isArray(healthCriteria) ? healthCriteria : [healthCriteria];

        // Construct a query object for MongoDB
        const query = {};

        if (dietCriteriaArray.length > 0) {
            query.dietLabels = { $all: dietCriteriaArray };
        }

        if (healthCriteriaArray.length > 0) {
            query.healthLabels = { $all: healthCriteriaArray };
        }

        // Check the database first for available recipes
        let recipes = await Recipe.find(query).exec();

        if (recipes.length < 12) {
            // Fetch recipes from the Edamam API
            const queryString = dietCriteriaArray.map(diet => `diet=${encodeURIComponent(diet)}`).join('&') + '&' +
                healthCriteriaArray.map(health => `health=${encodeURIComponent(health)}`).join('&');
            const data = await fetchRecipes(`&${queryString}`);
```

The added code constructs a MongoDB query object to filter recipes based on specified dietary and health criteria. It first searches the database for matching recipes.

Problem Encountered #1

In this case, if the frontend's HTTP requests always retrieve data from Edamam and never include _id and __v fields, it indicates that the backend might not be querying MongoDB correctly. This discrepancy suggests that the application may not be utilizing the database to retrieve recipes as intended.

Name

recipes?diet=balanced 1

recipes?diet=balanced

Headers Payload Preview


```
[{title: "Easy Mango-Blackberry Smoothie Recipe",…}, {title: "Crispy Frizzled Artichokes recipes",…},…]
  0: {title: "Easy Mango-Blackberry Smoothie Recipe",…}
    caloriesPerServing: 274.59632724687503
    cuisineType: ["american"]
    dietLabels: ["Balanced", "High-Fiber", "Low-Sodium"]
    dishType: ["drinks"]
    healthLabels: ["Vegetarian", "Pescatarian", "Mediterranean", "Gluten-Free", "Wheat-Free", "Egg-Free", "Peanut-Free",…]
    image: "https://edamam-product-images.s3.amazonaws.com/web-img/36b/36be4fd2cbb2dafacf722da53d7de6da.jpg?X-Amz-Security-Token=IQoJ
    ingredients: ["1 large ripe Ataulfo mango, peeled and roughly chopped (about 1 cup)",…]
    instructionsUrl: "http://www.seriouseats.com/recipes/2014/01/easy-mango-blackberry-kefir-smoothie-recipe.html"
    mealType: ["breakfast"]
    servingSize: 1
    source: "Serious Eats"
    title: "Easy Mango-Blackberry Smoothie Recipe"
    totalNutrients: {ENERC_KCAL: {label: "Energy", quantity: 274.59632724687503, unit: "kcal"},…}
    totalTime: 3
```


Name

recipes?diet=balanced

recipes?diet=balanced 2

Headers Payload Preview Response Initiator Timing


```
[{title: "Easy Mango-Blackberry Smoothie Recipe",…}, {title: "Crispy Frizzled Artichokes recipes",…},…]
  0: {title: "Easy Mango-Blackberry Smoothie Recipe",…}
    caloriesPerServing: 274.59632724687503
    cuisineType: ["american"]
    dietLabels: ["Balanced", "High-Fiber", "Low-Sodium"]
    dishType: ["drinks"]
    healthLabels: ["Vegetarian", "Pescatarian", "Mediterranean", "Gluten-Free", "Wheat-Free", "Egg-Free", "Peanut-Free",…]
    image: "https://edamam-product-images.s3.amazonaws.com/web-img/36b/36be4fd2cbb2dafacf722da53d7de6da.jpg?X-Amz-Security-Token=IQoJ
    ingredients: ["1 large ripe Ataulfo mango, peeled and roughly chopped (about 1 cup)",…]
    instructionsUrl: "http://www.seriouseats.com/recipes/2014/01/easy-mango-blackberry-kefir-smoothie-recipe.html"
    mealType: ["breakfast"]
    servingSize: 1
    source: "Serious Eats"
    title: "Easy Mango-Blackberry Smoothie Recipe"
    totalNutrients: {ENERC_KCAL: {label: "Energy", quantity: 274.59632724687503, unit: "kcal"},…}
    totalTime: 3
```


The absence of _id and __v fields in the response indicates that the data was not retrieved from MongoDB.

Investigation

We did not notice beforehand that the Edamam response uses capitalized initial letters, which led to discrepancies as this data was stored in the database. This mismatch caused issues when querying the database.

Documentation from Edamam API service provider

diet

array[string]

(query)

balanced
high-fiber
high-protein

Bruno HTTP response from Edamam API

GET https://api.edamam.com/api/recipes/v2?type=public&app_id=[redacted]&app_key=[redacted]

Query 4 | Body | Headers | Auth

Vars | Script | Assert | Tests | Docs

Name	Value
type	public
app_id	[redacted]
app_key	[redacted]

Response | Headers | Timeline | Tests

200 OK 1.40s 377

```
069629e1699455/-/balanced",
37        "yield": 1,
38        "dietLabels": [
39          "Balanced",
40          "High-Fiber",
41          "Low-Sodium"
42        ],
43        "healthLabels": [
44          "Vegetarian",
```

Frontend UI

Dietary Preferences:

balanced high-fiber high-protein low-carb low-fat low-sodium

Health Considerations:

dairy-free egg-free gluten-free low-potassium low-sugar mediterranean
mustard-free no-oil-added soy-free sugar-conscious tree-nut-free vegan
vegetarian wheat-free

Backend logic

```
10  const getRecipes = async (req, res) => {
11      try {
12          // Extract diet and health query parameters from the request
13          const dietCriteria = req.query.diet || [];
14          const healthCriteria = req.query.health || [];
```

```
31          // Check the database first for available recipes
32          let recipes = await Recipe.find(query).exec();
33
```

```
60          // Save the fetched recipes to the MongoDB database
61          await Recipe.insertMany(fetchedRecipes);
62
```

In the Edamam documentation, parameters are specified with lowercase initial letters, so we followed this convention on the frontend. Consequently, this format was used for querying the database.

Solution

```
9   const getRecipes = async (req, res) => {
10      try {
11          // Extract diet and health query parameters from the request
12          const dietCriteria = req.query.diet || [];
13          const healthCriteria = req.query.health || [];
14
15          // Function to capitalize the first letter of each word
16          const capitalizeWords = (str) => str
17              .split('-')
18              .map(word => word.charAt(0).toUpperCase() + word.slice(1).toLowerCase())
19              .join('-');
20
21          // Function to convert string to lowercase
22          const toLowerCase = (str) => str.toLowerCase();
23
24          // Normalize query parameters for MongoDB (capitalize first letter of each word)
25          const dietCriteriaArray = Array.isArray(dietCriteria) ? dietCriteria.map(d => capitalizeWords(d)) : [capitalizeWords(dietCriteria)];
26          const healthCriteriaArray = Array.isArray(healthCriteria) ? healthCriteria.map(h => capitalizeWords(h)) : [capitalizeWords(healthCriteria)];
27
28          // Normalize query parameters for Edamam API (all lowercase)
29          const dietCriteriaArrayLower = dietCriteriaArray.map(d => toLowerCase(d));
30          const healthCriteriaArrayLower = healthCriteriaArray.map(h => toLowerCase(h));
31
32          // Construct a query object for MongoDB
33          const query = {};
34
35          if (dietCriteriaArray.length > 0) {
36              query.dietLabels = { $all: dietCriteriaArray };
37          }
38
39          if (healthCriteriaArray.length > 0) {
40              query.healthLabels = { $all: healthCriteriaArray };
41          }
42
43          // Check the database for available recipes
44          let recipes = await Recipe.find(query, { _id: 0, __v: 0 }).exec();
```

Capitalize the HTTP request query parameters before sending them to MongoDB for querying to ensure consistency with the stored data format.

Result

Name | Headers | Payload | Preview | Response | Initiator | Timing

recipes?diet=balanced 1

recipes?diet=balanced


```
{dataFetchedFrom: "Edamam API",…}
  dataFetchedFrom: "Edamam API"
  recipes: [{title: "Meat Lite: Oven Roasted Potato Wedges with Bacon Grease Recipe",…},…]
    0: {title: "Meat Lite: Oven Roasted Potato Wedges with Bacon Grease Recipe",…}
    1: {title: "Serious Salsa: Chile Morita Salsa Recipe",…}
    2: {title: "The Secret Ingredient (Mango Chutney): Chutney and Cheddar Toasties Recipe",…}
    3: {title: "Vietnamese Coffee Ice Cream recipes",…}
    4: {title: "Pasta alla Gricia Recipe",…}
    5: {title: "Grilled Garlic Scapes",…}
    6: {title: "Kefir Date Shake Recipe",…}
    7: {title: "Roasted Brussels Sprouts and Shallots With Balsamic Vinegar Recipe",…}
    8: {title: "Easy Mango-Blackberry Smoothie Recipe",…}
    9: {title: "Roman-Style Fettuccine With Alfredo Sauce Recipe",…}
    10: {title: "Serious Salsa: Chile Morita Salsa Recipe",…}
    11: {title: "Milk Creams (Indian Cashew Sweets) Recipe",…}
```


Name | Headers | Payload | Preview | Response | Initiator | Timing

recipes?diet=balanced

recipes?diet=balanced 2


```
{dataFetchedFrom: "MongoDB",…}
  dataFetchedFrom: "MongoDB"
  recipes: [{title: "Roman-Style Fettuccine With Alfredo Sauce Recipe",…}, {title: "Frothy Indian Coffee",…},…]
    0: {title: "Roman-Style Fettuccine With Alfredo Sauce Recipe",…}
    1: {title: "Frothy Indian Coffee",…}
    2: {title: "Pasta alla Gricia Recipe",…}
    3: {title: "Easy Roasted Cabbage Recipe",…}
    4: {title: "Serious Salsa: Chile Morita Salsa Recipe",…}
    5: {title: "Easy Mango-Blackberry Smoothie Recipe",…}
    6: {title: "Spicy and Salty Mango Lassi Recipe",…}
    7: {title: "Momofuku Milk Bar's Cereal Milk Recipe",…}
    8: {title: "Milk Creams (Indian Cashew Sweets) Recipe",…}
    9: {title: "Kefir Date Shake Recipe",…}
    10: {title: "Chocolate Milk from Scratch Recipe",…}
    11: {title: "Roasted Brussels Sprouts and Shallots With Balsamic Vinegar Recipe",…}
```


As a result, the frontend's subsequent HTTP requests now successfully fetch data from MongoDB.

Problem Encountered #2

Background: During the development of the Dietary Selection component, we faced issues with managing the state of dietary and health criteria selections. The component allowed users to select dietary preferences and health considerations via checkboxes. When the search button was clicked, relevant recipes should be displayed based on these selections.

Issue: The primary issue was with state management for the checkboxes. Users' selections were not being accurately reflected in the state, leading to problems when the search button was clicked. Additionally, the search functionality sometimes failed to display results correctly, and the loading spinner was not appearing as expected.

Solution

1. State Management Fixes

Checkbox Handling: We implemented the handleCheckboxChange function (inside the DietarySelection component) to update the state based on user interactions with the checkboxes. This function ensures that the dietCriteria and healthCriteria arrays are correctly updated.

```
// Function to handle changes in checkbox selections
const handleCheckboxChange = (event, type) => {
    const value = event.target.value;
    if (type === 'diet') {
        // Update diet criteria based on checkbox selection
        setDietCriteria(prev =>
            prev.includes(value) ? prev.filter(item => item !== value) : [...prev, value]
        );
    } else if (type === 'health') {
        // Update health criteria based on checkbox selection
        setHealthCriteria(prev =>
            prev.includes(value) ? prev.filter(item => item !== value) : [...prev, value]
        );
    }
};
```

This code snippet ensures that when a checkbox is checked or unchecked, the state is updated accordingly.

Component Re-rendering: By making sure that the SearchRecipeButton component triggers a re-render with the updated criteria, the application correctly reflects the user’s selections.

Solution

2. Search Functionality

Search Initiation: The SearchRecipeButton component is configured to handle search requests with the selected criteria and update the results.

DietarySelection component

```
{/* Button to initiate recipe search */}
<SearchRecipeButton
    dietCriteria={dietCriteria}
    healthCriteria={healthCriteria}
    setErrorMessage={setErrorMessage}
    setRecipes={setRecipes}
    setIsSearchClicked={setIsSearchClicked}
    setIsLoading={setIsLoading}
/>
```

The SearchRecipeButton component triggers the recipe search process by using the provided dietary and health criteria. It updates the state with search results and controls the loading indicator based on its internal logic.

SearchRecipeButton component

```
5   const SearchRecipeButton = ({ dietCriteria, healthCriteria, setErrorMessage, setRecipes, setIsSearchClicked, setIsLoading }) => {
6       // Function to handle the search button click
7       const handleSearch = () => {
8           // Set search clicked state to true
9           setIsSearchClicked(true)
10
11          // Check if no dietary or health criteria are selected
12          if (dietCriteria.length === 0 && healthCriteria.length === 0) {
13              // Set error message if no criteria are selected
14              setErrorMessage('Please tick at least one dietary or health criterion below')
15              // Clear recipes and return
16              setRecipes([])
17              return;
18          }
19          // Clear error message if criteria are selected
20          setErrorMessage('');
21          // Fetch recipes based on selected criteria
22          fetchRecipes(dietCriteria, healthCriteria, setRecipes, setIsLoading);
23      };
24
25      return (
26          <div className="is-flex is-justify-content-center">
27              {/* Button to trigger search */}
28              <button className="button is-warning is-medium mt-5 mb-5" id="btn-search" onClick={handleSearch}>Search</button>
29          </div>
30      )
31  }
```

Solution

2. Search Functionality

Loading Spinner: To ensure the loading spinner displays correctly, we manage the isLoading state to show the spinner while data is being fetched.

DietarySelection component

```
{/* Conditional rendering based on whether the search button has been clicked */}
{isSearchClicked && (
    // Display loading spinner while recipes are being fetched
    isLoading ? (
        <LoadingSpinner />
    ) : (
    // Display the fetched recipes once they are available
    <ShowRecipes recipes={recipes} />
    )
)}
```

The code snippet manages the user interface based on the state of the search process. When the search button is clicked (isSearchClicked is true), it displays a loading spinner if data is still being fetched (isLoading is true), and once loading is complete, it presents the fetched recipes using the Show Recipes component.

Fetch Recipes function

```
export const fetchRecipes = async (dietCriteria, healthCriteria, setRecipes, setIsLoading) => {
    try {
        // Show the loading spinner by setting its state to true
        setIsLoading(true)
        // Create URLSearchParams object to build query string
        const queryParams = new URLSearchParams();

        // Append dietary criteria to query parameters
        dietCriteria.forEach(diet => queryParams.append('diet', diet));
        // Append health criteria to query parameters
        healthCriteria.forEach(health => queryParams.append('health', health));

        // Fetch recipes from the API with the constructed query parameters
        const response = await fetch(`https://diet-delight-backend.onrender.com/recipes?${queryParams.toString()}`);

        if (!response.ok) {
            // Throw an error if response is not OK
            throw new Error('Network response was not ok');
        }

        const data = await response.json();
        // Update recipes state with fetched data
        setRecipes(data.recipes);

    } catch (error) {
        console.error('Error fetching recipes:', error);
        // Set recipes state to an empty array if there's an error
        setRecipes([]);
    } finally {
        // Set loading spinner state to false regardless of the outcome
        setIsLoading(false)
    }
};
```

Solution

3. Error Handling

Error Messages: We included error handling to display messages if no criteria were selected or if there was an issue with the search.

DietarySelection component

```
{/* Display error message if an error exists */}
<DisplayErrorMessage message={errorMessage} style={'no-option-selected'}/>
```

This component provides feedback to the user if there are issues with the search criteria.

SearchRecipeButton component

```
// Check if no dietary or health criteria are selected
if (dietCriteria.length === 0 && healthCriteria.length === 0) {
    // Set error message if no criteria are selected
    setErrorMessage('Please tick at least one dietary or health criterion below')
    // Clear recipes and return
    setRecipes([])
    return;
}
```

Outcome: After implementing these changes, the DietarySelection component functioned correctly. The checkboxes accurately reflected user selections, the search results were displayed properly, and the loading spinner appeared during data fetching. These fixes resolved the encountered problems and improved the overall functionality and user experience.

Thank you!

Thank you, Matt and the CA team, for your guidance and support. We deeply appreciate the effort you have put into teaching and inspiring us.