Assignment Twelve

Please read:

- This is your twelfth assignment which you need to submit on the LMS. This is an assignment on Backend. The requirements are same as assignment 11.
- Work on the questions given below and be ready with your solution. You have to submit your published API link for your backend below on this page.
- Late Submission: The deadline to submit this assignment is 19th February 2024, 6:00PM IST.

Important Instructions:

1. Make sure your POSTMAN published link is working properly.
2. Do not copy from someone else as that would be cheating.

challenge

You can watch the video for reference:

Case Study: Building a Zomato-like Restaurant API

You are tasked with building the backend API for a restaurant discovery platform similar to Zomato. The goal is to create a set of API endpoints that allow users to perform various actions related to restaurants, such as creating, reading, updating, and deleting restaurants, managing menus, adding user reviews, and more.

You are provided with a MongoDB database and the Mongoose library to interact with it. The database schema includes collections for restaurants, user reviews, and user profiles. You are required to implement the following features:

1. Restaurant Management:
    - Create a new restaurant.
    - Retrieve restaurant details by name.
    - Retrieve a list of all restaurants.
    - Retrieve restaurants by cuisine type.
    - Update restaurant information (e.g., rating, address) by ID.
    - Delete a restaurant by ID.
    - Search for restaurants by location (city or address).
2. Menu Management:
    - Add a dish to a restaurant's menu.
    - Remove a dish from a restaurant's menu.
3. User Reviews:
    - Allow users to add reviews and ratings for a restaurant.
    - Retrieve user reviews for a specific restaurant.
    - Calculate and update the average ratings for all restaurants based on user reviews.

mongoDB & mongoose

Step 1: Create a Restaurant Model

You are tasked with creating a model for a restaurant in your database. Use Mongoose to create the `Restaurant` model with the following fields:

1. name: The name of the restaurant.
2. cuisine: The type of cuisine offered by the restaurant.
3. address: The address of the restaurant.
4. city: The city where the restaurant is located.
5. rating: The rating should be on a scale of 0 to 5, where 0 is the lowest and 5 is the highest. The default value is 0.
6. menu: An array of menu items offered by the restaurant.

    Each menu item should include the following fields:

- name: The name of the menu item.
- price: The price of the menu item.
- description: The description for the menu item.
- isVeg: veg or non-veg

7. averageRating: The default value is 0.

Step 2: Create a Restaurant

Create a function `createRestaurant` that accepts an object containing restaurant data and adds a new restaurant to the database.

Step 3: Read a Restaurant

Create a function `readRestaurant` that accepts the restaurant name and retrieves the restaurant details from the database.

Step 4: Read All Restaurants

Create a function `readAllRestaurants` that retrieves all restaurants from the database.

Step 5: Read Restaurants by Cuisine

Create a function `readRestaurantsByCuisine` that accepts a cuisine type and retrieves all restaurants with that cuisine.

Step 6: Update a Restaurant by ID

Create a function `updateRestaurant` that accepts a restaurant ID and an object with updated data, and updates the restaurant with the provided ID.

Step 7: Delete a Restaurant by ID

Create a function `deleteRestaurant` that accepts a restaurant ID and deletes the restaurant with the provided ID.

Step 8: Search Restaurants by Location

Create a function `searchRestaurantsByLocation` that accepts a location and retrieves all restaurants located in that area.

Step 9: Filter Restaurants by Rating

Create a function `filterRestaurantsByRating` that accepts a minimum rating and retrieves all restaurants with ratings greater than or equal to the specified value.

Documentation: https://mongoosejs.com/docs/2.7.x/docs/query.html#gte

`gte` example: `await Users.find({ age: { $gte: 18 } })`

This query finds and retrieves user documents from the "Users" collection where the "age" field is greater than or equal to 18.

Step 10: Add a Dish to a Restaurant's Menu

Create a function `addDishToMenu` that accepts a restaurant ID and a new dish object and adds the dish to the specified restaurant's menu.

Step 11: Remove a Dish from a Restaurant's Menu

Create a function `removeDishFromMenu` that accepts a restaurant ID and the name of a dish to remove from the restaurant's menu.

Step 12: Add a User Review and Rating for a Restaurant

Create a function `addRestaurantReviewAndRating` that allows a user to add a review and rating for a restaurant. Make sure to add the reviews array of object with rating, text & user reference in the `restaurant` model.

Step 13: Retrieve User Reviews for a Restaurant

Create a function `getUserReviewsForRestaurant` that retrieves all user reviews for a specific restaurant.

express

Restaurant API Exercises

Exercise 1: Creating a Restaurant API

Challenge:

Create a POST route at `/restaurants` that accepts JSON data with restaurant details. Use the `createRestaurant` function to save the restaurant data. Handle errors as well.

1. Create a POST Route: Set up a POST route at `/restaurants`.
2. Handle the Request: In the route handler, use the `createRestaurant` function to save the restaurant data received in the request body.
3. Success Response: If the restaurant is successfully added, respond with a success message and the saved restaurant details.
4. Error Handling: If an error occurs during the process, respond with an error message.

Example:

```
POST /restaurants

Request Body:
{
  "name": "New Restaurant",
  "cuisine": "Italian",
  "address": "123 Main St",
  "city": "New York",
  "rating": 4.5,
  "menu": [
    { "name": "Pizza Margherita", "price": 12.99 },
    { "name": "Pasta Carbonara", "price": 15.99 }
  ]
}
```

COPY

Exercise 2: Reading a Restaurant API

Challenge:

Set up a GET route at `/restaurants/:name` that allows users to retrieve restaurant details by providing the restaurant name as a route parameter. Utilize the `readRestaurant` function to fetch the restaurant data. Don't forget error handling!

1. Create a GET Route: Define a GET route at `/restaurants/:name` to handle requests for specific restaurants.

2. Retrieve Restaurant Data: In the route handler, use the `readRestaurant` function to find the restaurant with the provided name.
3. Success Response: If the restaurant is found, respond with the restaurant details.
4. Error Handling: If an error occurs during the process, respond with an error message.

Example:

```
GET /restaurants/New%20Restaurant
```

COPY

Exercise 3: Reading All Restaurants API

Challenge:

Create a GET route at `/restaurants` to fetch all restaurant details. Utilize the `readAllRestaurants` function to retrieve the restaurant data. Handle any potential errors gracefully.

1. Create a GET Route: Define a GET route at `/restaurants` to handle requests for all restaurants.
2. Retrieve All Restaurants: In the route handler, use the `readAllRestaurants` function to retrieve all restaurant data.
3. Success Response: If restaurants are found, respond with an array containing all restaurant details.
4. Error Handling: If an error occurs during the process, respond with an error message.

Example:

```
GET /restaurants
```

COPY

Exercise 4: Reading Restaurants by Cuisine

Challenge:

Create a GET route at `/restaurants/cuisine/:cuisineType` that allows users to retrieve restaurants by cuisine type. Utilize the `readRestaurantsByCuisine` function to fetch the restaurant data. Handle errors gracefully.

1. Create a GET Route: Set up a GET route at `/restaurants/cuisine/:cuisineType` to handle requests for retrieving restaurants by cuisine type.
2. Retrieve Restaurants: In the route handler, use the `readRestaurantsByCuisine` function to find restaurants with the provided cuisine type.
3. Success Response: If restaurants are found, respond with an array containing the details of those restaurants.
4. Error Handling: If an error occurs during the process, respond with an error message.

Example:

```
GET /restaurants/cuisine/Italian
```

COPY

Exercise 5: Updating a Restaurant API

Challenge:

Develop a POST route at `/restaurants/:restaurantId` that allows users to update a restaurant's details. Utilize the `updateRestaurant` function to make the necessary changes. Handle errors effectively.

1. Create a POST Route: Design a POST route at `/restaurants/:restaurantId` to handle requests for updating restaurant details.
2. Update Restaurant: In the route handler, use the `updateRestaurant` function to modify the specified restaurant using its ID and the updated data.
3. Success Response: If the restaurant is successfully updated, respond with the updated restaurant details.
4. Error Handling: If an error arises during the process, respond with an error message.

Example:

```
POST /restaurants/617a2ed9466e8c00160192e0

Request Body:
{
  "name": "Updated Restaurant Name",
  "cuisine": "Mexican"
}
```

COPY

Exercise 6: Deleting a Restaurant API

Challenge:

Craft a DELETE route at `/restaurants/:restaurantId` that enables users to delete a restaurant. Utilize the `deleteRestaurant` function to perform the deletion. Handle errors gracefully.

1. Create a DELETE Route: Establish a DELETE route at `/restaurants/:restaurantId` to manage requests for deleting a restaurant.
2. Delete Restaurant: In the route handler, use the `deleteRestaurant` function to remove the specified restaurant using its ID.
3. Success Response: If the restaurant is successfully deleted, respond with a success message.
4. Error Handling: If an error arises during the process, respond with an error message.

Example:

```
DELETE /restaurants/617a2ed9466e8c00160192e0
```

COPY

Exercise 7: Searching for Restaurants by Location

Challenge:

Create a GET route at `/restaurants/search` that allows users to search for restaurants by location (city or address). Utilize the `searchRestaurantsByLocation` function to retrieve the restaurant data based on the location provided in the query parameter. Handle errors gracefully.

1. Create a GET Route: Define a GET route at `/restaurants/search` to handle requests for searching restaurants by location.
2. Retrieve Restaurants: In the route handler, use the `searchRestaurantsByLocation` function to find restaurants based on the provided location in the query parameter.
3. Success Response: If restaurants are found, respond with an array containing the details of those restaurants.
4. Error Handling: If an error occurs during the process, respond with an error message.

📌 Make sure to check the order in which you've defined your routes.

Example:

```
GET /restaurants/search?location=New%20York
```

COPY

Exercise 8: Filtering Restaurants by Rating

Challenge:

Create a GET route at `/restaurants/rating/:minRating` that allows users to filter restaurants by a minimum rating. Utilize the `filterRestaurantsByRating` function to retrieve the restaurant data. Handle errors effectively.

1. Create a GET Route: Define a GET route at `/restaurants/rating/:minRating` to handle requests for filtering restaurants by a minimum rating.
2. Retrieve Restaurants: In the route handler, use the `filterRestaurantsByRating` function to find restaurants with a rating equal to or higher than the provided minimum rating.
3. Success Response: If restaurants are found, respond with an array containing the details of those restaurants.
4. Error Handling: If an error occurs during the process, respond with an error message.

Example:

```
GET /restaurants/rating/4.0
```

COPY

Menu Management API Exercises

Exercise 9: Adding a Dish to a Restaurant's Menu

Challenge:

Create a POST route at `/restaurants/:restaurantId/menu` that allows users to add a new dish to a restaurant's menu. Utilize the `addDishToMenu` function to add the dish. Handle errors effectively.

1. Create a POST Route: Design a POST route at `/restaurants/:restaurantId/menu` to handle requests for adding a dish to a restaurant's menu.
2. Add Dish: In the route handler, use the `addDishToMenu` function to add the new dish to the specified restaurant's menu.
3. Success Response: If the dish is successfully added, respond with a success message and the updated menu.
4. Error Handling: If an error arises during the process, respond with an error message.

Example:

```
POST /restaurants/617a2ed9466e8c00160192e0/menu

Request Body:
{
  "name": "New Dish",
  "price": 9.99
}
```

COPY

Exercise 10: Removing a Dish from a Restaurant's Menu

Challenge:

Create a DELETE route at `/restaurants/:restaurantId/menu/:dishName` that allows users to remove a dish from a restaurant's menu. Utilize the `removeDishFromMenu` function to delete the dish. Handle errors gracefully.

1. Create a DELETE Route: Establish a DELETE route at `/restaurants/:restaurantId/menu/:dishName` to manage requests for removing a dish from a restaurant's menu.
2. Remove Dish: In the route handler, use the `removeDishFromMenu` function to delete the specified dish from the restaurant's menu.
3. Success Response: If the dish is successfully removed, respond with a success message.
4. Error Handling: If an error occurs during the process, respond with an error message.

Example:

```
DELETE /restaurants/617a2ed9466e8c00160192e0/menu/:dishName
```

COPY

User Reviews API Exercises

Exercise 11: Allowing Users to Add Reviews and Ratings for a Restaurant

Challenge:

Create a POST route at `/restaurants/:restaurantId/reviews` that allows users to add reviews and ratings for a restaurant. Utilize the `addRestaurantReviewAndRating` function to add the review. Handle errors effectively.

1. Create a POST Route: Set up a POST route at `/restaurants/:restaurantId/reviews` to handle requests for adding reviews and ratings for a restaurant.
2. Add Review: In the route handler, use the `addRestaurantReviewAndRating` function to add the review and rating to the specified restaurant.
3. Success Response: If the review is successfully added, respond with a success message and the updated restaurant details.
4. Error Handling: If an error arises during the process, respond with an error message.

Example:

```
POST /restaurants/617a2ed9466e8c00160192e0/reviews

Request Body:
{
  "userId": "617a2ed9466e8c00160192e1",
  "rating": 4.5,
  "reviewText": "Great food and excellent service!"
}
```

COPY

Exercise 12: Retrieving User Reviews for a Specific Restaurant

Challenge:

Create a GET route at `/restaurants/:restaurantId/reviews` that allows users to retrieve user reviews for a specific restaurant. Utilize the `getUserReviewsForRestaurant` function to fetch the reviews. Handle errors gracefully.

1. Create a GET Route: Set up a GET route at `/restaurants/:restaurantId/reviews` to handle requests for retrieving user reviews for a specific restaurant.
2. Retrieve Reviews: In the route handler, use the `getUserReviewsForRestaurant` function to fetch the user reviews for the specified restaurant.
3. Success Response: If reviews are found, respond with an array containing the user reviews.
4. Error Handling: If an error occurs during the process, respond with an error message.

Example:

```
GET /restaurants/64f579d188c2b014b0c0a687/reviews
```