Project Report: Backend for "Coffee Corner" Web Application Using Nest.js and MongoDB

Project Contributors:

- **Likhita Sai Matta (ID: 101293458)**
- **Jaywant Vijaykumar Patel (ID: 101321483)**

Repository Link

GitHub Repository: LikhitaSaiMatta/Web-and-Mobile-Development-Assignments

---

Project Overview

The "Coffee Corner" web application is designed for users in Ontario, Canada, allowing them to discover and review local cafes. We developed a backend using **NestJS** to provide RESTful API services for managing cafe and review data. Initially, the backend stored data in memory, but as the application expanded, we transitioned to a **MongoDB** database. This shift ensured data persistence, scalability, and efficiency in managing larger datasets, which are essential for future growth.

---

2. Objectives

The objectives for developing the backend and transitioning it to MongoDB were:

1. **Establish RESTful API Services**: Create a flexible and functional API for basic CRUD operations.
2. **Data Persistence and Scalability**: Implement MongoDB to store data permanently and handle larger datasets.
3. **Improve Performance and Efficiency**: Utilize MongoDB for optimized data retrieval and storage, which is essential as user demand grows.
4. **Maintain Code Modularity and Organization**: Structure the application using NestJS modules, controllers, and services for easier future maintenance and scalability.

---

3. API Endpoints

The backend API provides endpoints to manage cafes and their reviews, allowing CRUD operations and data retrieval as required by the front end.

Core Endpoints

Endpoint	Method	Description
`/cafes`	GET	Retrieve a list of all cafes
`/cafes/:id`	GET	Retrieve information on a single cafe
`/cafes`	POST	Add a new cafe
`/cafes/:cafeId/reviews`	GET	Retrieve all reviews for a cafe
`/cafes/:cafeId/reviews`	POST	Submit a review for a cafe

Each endpoint was tested for functionality, data persistence, and correct responses after integrating MongoDB. We confirmed the endpoints accurately support the required operations for "Coffee Corner" users.

---

4. Implementation Summary

4.1 Initial Setup with In-Memory Storage

We started the backend development using NestJS's modular architecture. The in-memory storage setup allowed us to quickly create basic CRUD functionalities for cafes and reviews. This approach was useful for testing and verifying initial API structures, data models, and controller functions.

4.2 Transition to MongoDB for Data Persistence

MongoDB Setup

After setting up MongoDB locally and creating the `coffee_corner` database, we connected it to the NestJS project using Mongoose. MongoDB was chosen for its flexibility with unstructured data and scalability, which fit well with our application's needs.

Database Schemas

Schemas for `Cafe` and `Review` were defined to structure data in MongoDB. Each `Cafe` document includes details such as name, location, rating, and associated reviews, while each `Review` document contains fields linking it to a specific cafe. These schemas allowed consistent data organization and ensured compatibility with MongoDB's document-oriented model.

Updating Services and Controllers

With MongoDB in place, we refactored the services to use MongoDB's Mongoose queries for data manipulation instead of in-memory arrays. This transition enabled persistent storage and efficient data retrieval, which is critical for supporting real-world application demands.

---

5. Testing and Validation

We performed extensive testing of each endpoint using **Postman**. The testing ensured that:

1. **Data Persistence**: Data added or modified through the API remained accessible across server restarts.
2. **Correctness of Data**: Data fetched from MongoDB matched the requests, providing accurate responses.
3. **Functionality**: Each endpoint performed the intended CRUD operations as expected.

Through testing, we confirmed that the MongoDB transition was successful, and the backend maintained functionality and efficiency.

---

6. Challenges and Solutions

1. **Schema Design**: Structuring relationships between cafes and reviews was crucial. We implemented a reference system linking reviews to cafes, ensuring data consistency.
2. **Service Transition**: Adapting existing service logic to MongoDB required replacing in-memory operations with Mongoose queries. This change preserved the functionality of the original services while supporting data persistence.
3. **Ensuring Data Persistence**: Testing with MongoDB Compass and Postman allowed us to verify data accuracy and persistence, giving confidence that data remained stable across sessions.

---

7. Lessons Learned

1. **Benefits of MongoDB**: MongoDB’s scalability and document-based structure provide a robust solution for applications needing flexible data handling.
2. **Type Safety with Mongoose**: Using TypeScript with Mongoose improves reliability and reduces runtime errors, particularly when interacting with complex data structures.
3. **Modular Development in NestJS**: NestJS’s modular structure simplifies the codebase, allowing easier maintenance and enabling future feature additions.

---

8. Future Improvements

1. **Role-Based Access Control**: Implementing authentication and authorization to control access to certain endpoints (e.g., only managers can add cafes).
2. **Advanced Data Querying**: Enhancing search and filter functionalities to improve user experience.
3. **Error Handling**: Adding comprehensive error handling and validation to improve robustness and provide meaningful feedback to users.

---

9. Conclusion

Transitioning the "Coffee Corner" backend to MongoDB has significantly improved the stability and scalability of the application. The NestJS and MongoDB integration provides a flexible, maintainable backend structure, essential for the application’s growth. With this setup, "Coffee Corner" is well-prepared to handle increasing user demand and additional features in the future.