Detailed Summary of Work Done for the MongoDB Java Project

ABOU ORM Daniel: *Git Repo*

January 19, 2025

Contents

1 Introduction **2**
- 1.1 Environment Setup . . . 2
- 1.2 Edit Template Configurations in IntelliJ . . . 2

2 Work on *Sample Training* **2**
- 2.1 Classes and Their Functionality . . . 3
  - 2.1.1 Create . . . 3
  - 2.1.2 Read . . . 3
  - 2.1.3 Update . . . 3
  - 2.1.4 Delete . . . 3

3 Personalized Fitness Project **4**
- 3.1 CreatePersonalizedFitness Class . . . 4
- 3.2 ReadPersonalizedFitness Class . . . 5
- 3.3 UpdatePersonalizedFitness Class . . . 7
- 3.4 DeletePersonalizedFitness Class . . . 8
- 3.5 DeleteEntireDatabase Class . . . 9
- 3.6 Main Class . . . 10

4 Execution Logs **12**

5 Conclusion **14**

1 Introduction

This document summarizes all the tasks and operations we completed during the MongoDB Java project. The goal of the project was to demonstrate how to use MongoDB with Java to perform CRUD (Create, Read, Update, Delete) operations while connecting to a MongoDB Atlas cluster. Below is a breakdown of our activities.

1.1 Environment Setup

Tools and Libraries Used

- **IntelliJ IDEA**: Used as the primary IDE for writing and testing the Java code.
- **Maven**: Dependency management tool used to include the MongoDB Java Driver in the project.
- **MongoDB Atlas**: Cloud database service used to host the MongoDB database.

MongoDB Java Driver Integration

- Added the MongoDB Java Driver to the Maven `pom.xml` to enable database connectivity.
- Configured IntelliJ IDEA with the correct SDK and Maven dependencies to compile and run the project.

1.2 Edit Template Configurations in IntelliJ

- Configured the JVM argument:

```
-Dmongodb.uri=mongodb+srv://USR:PASS@cluster0.rqswf.mongodb.
   net/<sample_training OR PersonalizedFitness>
```

- This allowed IntelliJ to pass the MongoDB URI to the Java programs as a system property.

2 Work on *Sample Training*

In this section, we summarize the work conducted on the `sampleTraining` module, which was designed for learning and testing purposes as part of our MongoDB Java project. The classes created in this module helped us practice fundamental CRUD (Create, Read, Update, Delete) operations on a MongoDB database. These exercises laid the groundwork for the larger project discussed in the following sections.

2.1 Classes and Their Functionality

- **2.1.1 Create**

  This class focuses on inserting documents into the `grades` collection. Two main methods were implemented:

  - `insertOneDocument`: Adds a single document with details such as *student_id*, *class_id*, and a list of scores.
  - `insertManyDocuments`: Adds multiple documents at once by using a loop to generate random data.

- **2.1.2 Read**

  This class explores various ways to read data from the collection. The functionalities include:

  - Retrieving a single document using filters (`Filters.eq`, `Filters.gte`).
  - Iterating through results using a cursor or a list.
  - Sorting, projecting, skipping, and limiting results.

- **2.1.3 Update**

  This class implements multiple approaches for updating existing documents:

  - Adding or modifying specific fields (`updateOne`, `updateMany`).
  - Using the `upsert` option to insert a document if it does not exist.
  - Advanced methods like `findOneAndUpdate` to retrieve the version before or after an update.

- **2.1.4 Delete**

  This class demonstrates how to delete documents in various ways:

  - Deleting a single document matching a specific filter (`deleteOne`).
  - Deleting multiple documents (`deleteMany`).
  - Dropping the entire collection, including its metadata, using the `drop` method.

Purpose of the Work

The `sampleTraining` module served as a practical way to understand how to perform essential MongoDB operations using Java. Each class provided hands-on experience with a specific aspect of working with data, such as creating documents, querying the database, applying updates, and removing data. These exercises were not just isolated tests but also helped us prepare for more advanced use cases in the larger project, which we will detail in the next sections.

3 Personalized Fitness Project

The `PersonalizedFitness` module is the primary focus of our MongoDB Java project. This module extends the concepts learned in the `sampleTraining` phase and applies them to a real-world scenario. The goal of the `PersonalizedFitness` project is to manage and analyze fitness-related data, showcasing advanced MongoDB operations combined with Java to create a functional application.

This part of the project emphasizes:

- Designing a database schema tailored for fitness data, including user profiles, workout plans, and progress tracking.
- Implementing complex CRUD operations to handle fitness data efficiently.
- Showcasing advanced MongoDB features such as aggregations, indexing, and performance optimization.

The `PersonalizedFitness` project serves as a practical demonstration of building a Java application with MongoDB, aimed at providing personalized insights and features to users based on their fitness data.

3.1 CreatePersonalizedFitness Class

The `CreatePersonalizedFitness` class is a critical component of the `PersonalizedFitness` module, responsible for initializing and populating the MongoDB database with user and training plan data. Below is a detailed explanation of its components and functionality:

Overview

The class includes methods for creating users and training plans dynamically or through a static main method. It interacts with two primary collections in the `PersonalizedFitness` database:

- **Users:** Contains user information such as `UserID`, `Name`, `Age`, and `TrainingPlanID`.
- **TrainingPlans:** Stores predefined workout plans, each associated with a unique `PlanID`, `Goal`, and a list of exercises.

Class Components

- **Constructor:** Initializes the connection to the `PersonalizedFitness` database and sets up references to the `Users` and `TrainingPlans` collections. The database URI is dynamically passed as an argument, allowing flexibility for different environments.
- **addUser Method:** Dynamically adds a single user to the `Users` collection if the user does not already exist. The method ensures no duplicate entries are added by checking the `UserID`.
- **Static Main Method:** Demonstrates the process of creating and populating the database:

- **Training Plans Creation:** Defines five unique training plans, each with a specific `Goal` and associated exercises. These plans are inserted into the `TrainingPlans` collection if they do not already exist.
- **Users Creation:** Inserts 15 unique users into the `Users` collection. User details, including `Name`, `Age`, and `TrainingPlanID`, are dynamically generated using a combination of predefined names and random values.

Key Functionalities in the Main Method

- Ensures the `TrainingPlans` collection contains five distinct entries by checking for existing `PlanID`s before inserting new plans.
- Cycles through a list of predefined names and randomly assigns ages (between 18 and 50) and valid training plan IDs (1 to 5) for 15 unique users.
- Provides detailed console output to track the progress of data insertion, highlighting skipped entries if a duplicate is detected.

Significance

The `CreatePersonalizedFitness` class exemplifies the integration of Java with MongoDB to manage dynamic data creation. Its functionality ensures the database is preloaded with relevant data, forming the foundation for subsequent operations such as querying, updating, and analyzing fitness-related information in the `PersonalizedFitness` project.

3.2 ReadPersonalizedFitness Class

The `ReadPersonalizedFitness` class is responsible for retrieving data from the `PersonalizedFitness` database. It includes various methods to query user and training plan information, either individually or in bulk. Below is a detailed explanation of its components and functionality:

Overview

The class provides methods to:

- Retrieve a specific user or training plan by their unique identifier.
- Retrieve and display all users and training plans.
- Handle both successful queries and cases where no matching documents are found.

Class Components

- **Constructor:**
  - Accepts a MongoDB URI as a parameter and establishes a connection to the `PersonalizedFitness` database.
  - Initializes references to the `Users` and `TrainingPlans` collections.
  - Includes validation to ensure the URI is not null or empty, throwing an exception if invalid.

- **Method:** `readUserById`
  - Queries the `Users` collection using the `UserID`.
  - Prints the user's information in JSON format if found.
  - Displays an appropriate message if no user matches the specified `UserID`.
- **Method:** `readTrainingPlanById`
  - Queries the `TrainingPlans` collection using the `PlanID`.
  - Prints the training plan's details in JSON format if found.
  - Displays an appropriate message if no training plan matches the specified `PlanID`.
- **Method:** `readAllUsers`
  - Retrieves and prints all documents in the `Users` collection.
  - Iterates through the results using a loop, displaying each user's details in JSON format.
- **Method:** `readAllTrainingPlans`
  - Retrieves and prints all documents in the `TrainingPlans` collection.
  - Iterates through the results using a loop, displaying each training plan's details in JSON format.

Main Method for Testing

The `main` method provides a standalone testing mechanism for the class's functionality:

- Retrieves the MongoDB URI from system properties and validates its presence.
- Tests the following methods:
  - `readUserById`: Demonstrates both successful and unsuccessful queries with a valid and a non-existent `UserID`.
  - `readTrainingPlanById`: Demonstrates both successful and unsuccessful queries with a valid and a non-existent `PlanID`.
  - `readAllUsers`: Displays all user documents in the collection.
  - `readAllTrainingPlans`: Displays all training plan documents in the collection.
- Outputs detailed console logs for each operation, making it easy to track results and errors during execution.

Significance

The `ReadPersonalizedFitness` class showcases effective querying techniques using MongoDB's Java Driver. It ensures efficient retrieval of both individual and bulk data, facilitating the development of data-driven applications within the `PersonalizedFitness` project.

3.3 UpdatePersonalizedFitness Class

The `UpdatePersonalizedFitness` class is designed to modify existing records in the `PersonalizedFitness` database. It provides methods to update user details and training plan attributes. This class ensures efficient updates by targeting specific fields and documents within the collections.

Overview

The class includes methods to:

- Update a user's name based on their `UserID`.
- Modify a training plan's goal using its `PlanID`.
- Change the assigned training plan for a specific user.

Class Components

- **Constructor:**
  - Accepts a MongoDB URI as a parameter to establish a connection with the database.
  - Initializes references to the `Users` and `TrainingPlans` collections.
  - Validates the URI and outputs debug messages to confirm successful connection.
- **Method:** `updateUserName`
  - Updates the name of a user identified by their `UserID`.
  - Uses the `updateOne` method to modify the `Name` field.
  - Outputs a confirmation message if the user is found and updated.
  - Displays a message if no matching user is found.
- **Method:** `updateTrainingPlanGoal`
  - Modifies the `Goal` field of a training plan identified by its `PlanID`.
  - Utilizes the `updateOne` method for efficient updates.
  - Outputs success or failure messages based on the operation's result.
- **Method:** `updateUserTrainingPlan`
  - Updates the `TrainingPlanID` for a user based on their `UserID`.
  - Ensures that the update applies only to existing users.
  - Displays confirmation or failure messages after attempting the update.

Main Method for Testing

The `main` method provides a comprehensive testing setup:

- Retrieves the MongoDB URI from system properties and validates it.
- Demonstrates the functionality of each update method:
  - `updateUserName`: Updates both an existing and a non-existent `UserID`.
  - `updateTrainingPlanGoal`: Tests updates for a valid and an invalid `PlanID`.
  - `updateUserTrainingPlan`: Modifies the training plan for an existing user and attempts the same for a non-existent user.
- Outputs detailed logs to track the results of each test case.

Significance

The `UpdatePersonalizedFitness` class demonstrates the capability to efficiently modify specific fields within MongoDB documents. By handling edge cases such as non-existent records, it ensures robustness and reliability in the update operations, making it a key component of the `PersonalizedFitness` project.

3.4 DeletePersonalizedFitness Class

The `DeletePersonalizedFitness` class handles the removal of data from the `PersonalizedFitness` database. It extends the `UpdatePersonalizedFitness` class, inheriting its methods and collections. This class focuses on deleting users and training plans, while managing the reassignment of users affected by the removal of a training plan.

Overview

The class provides methods to:

- Delete a user by their `UserID`.
- Delete a training plan by its `PlanID` and reassign affected users to other available plans.

Class Components

- **Constructor:**
  - Inherits from `UpdatePersonalizedFitness`, initializing access to the `Users` and `TrainingPlans` collections.
  - Reuses the parent class's connection setup and debugging features.
- **Method:** `deleteUserById`
  - Deletes a user document from the `Users` collection based on their `UserID`.
  - Verifies if a user exists before attempting deletion.
  - Outputs the result of the operation, indicating success or failure.

- **Method:** `deleteTrainingPlanById`
  - Deletes a training plan document from the `TrainingPlans` collection by its `PlanID`.
  - Checks if the plan exists before proceeding.
  - Ensures there are other training plans available to reassign affected users.
  - Reassigns users linked to the deleted plan to a randomly selected alternative plan.
  - Outputs detailed messages about the reassignment process and the deletion result.

Main Method for Testing

The `main` method facilitates testing and debugging:

- Retrieves the MongoDB URI from system properties, validating its presence.
- Demonstrates the functionality of each method:
  - `deleteUserById`: Tests deletion of both an existing and a non-existent `UserID`.
  - `deleteTrainingPlanById`: Tests deletion of a valid and an invalid `PlanID`, ensuring users are reassigned when applicable.
- Provides comprehensive logs for each test case to confirm correct functionality.

Significance

The `DeletePersonalizedFitness` class ensures the integrity of the `PersonalizedFitness` database by handling data deletions responsibly. The reassignment of users affected by training plan deletions highlights its ability to maintain data consistency and continuity within the application.

3.5 DeleteEntireDatabase Class

The `DeleteEntireDatabase` class provides functionality to delete the entire `PersonalizedFitness` database, ensuring that this critical action is performed securely through password protection and user confirmation.

Overview

This class implements:

- A secure process for deleting the entire database.
- Password and confirmation prompts to prevent accidental or unauthorized deletions.

Class Components

- **Constructor:**
  - Initializes a connection to the `PersonalizedFitness` database using the provided MongoDB URI.
  - Verifies the connection and outputs the database name for debugging purposes.
- **Method:** `deleteDatabase`
  - Warns the user about the critical nature of the action.
  - Requests a password input (`Dano2003`) for authentication.
  - Prompts for additional confirmation by requiring the user to type `CONFIRM`.
  - Deletes the entire database (`database.drop()`) if the password and confirmation are correct.
  - Outputs the status of the operation, such as success or cancellation due to incorrect inputs.

Main Method for Testing

The `main` method demonstrates the following:

- Retrieves the MongoDB URI from system properties and validates its presence.
- Provides an optional test case to delete the database without password protection (commented out for safety).

Significance

The `DeleteEntireDatabase` class ensures secure deletion of the `PersonalizedFitness` database. By implementing password protection and confirmation prompts, it mitigates the risk of accidental or unauthorized deletion, making it a robust and responsible implementation for critical database operations.

3.6 Main Class

The `Main` class acts as the entry point for the `PersonalizedFitness` project. It integrates all components—`CreatePersonalizedFitness`, `ReadPersonalizedFitness`, `UpdatePersonalizedFit` `DeletePersonalizedFitness`, and `DeleteEntireDatabase`—to demonstrate their functionalities in a unified workflow.

Overview

The `Main` class performs the following tasks:

- Establishes a connection to the MongoDB Atlas database using a URI retrieved from system properties.
- Demonstrates the creation, reading, updating, and deletion of users and training plans.
- Optionally deletes the entire database with password protection (commented out for safety).

Key Steps and Operations

1. **Environment Setup:**
   - The MongoDB URI is retrieved using `System.getProperty("mongodb.uri")`.
   - If the URI is not provided, an error message is displayed, and the program exits.
2. **Class Instantiations:**
   - `CreatePersonalizedFitness` for adding users and training plans.
   - `ReadPersonalizedFitness` for retrieving data.
   - `UpdatePersonalizedFitness` for updating user and training plan information.
   - `DeletePersonalizedFitness` for removing specific users or training plans.
   - `DeleteEntireDatabase` for deleting the entire database securely (optional).
3. **Operations:**
   (a) **Creating a User:**
      - A new user is dynamically added with `UserID = 117`, `Name = "Daniel"`, `Age = 21`, and `TrainingPlanID = 5`.

   (b) **Reading Data:**
      - Retrieves a specific user and training plan by their IDs.
      - Reads and displays all users and training plans in the database.

   (c) **Updating Data:**
      - Updates a user's name (e.g., changing `UserID = 101` to `Name = "Alice"`).
      - Modifies a training plan's goal (e.g., updating `PlanID = 3` to `"Updated Endurance Goal"`).
      - Changes a user's assigned training plan (e.g., reassigning `UserID = 104` to `TrainingPlanID = 4`).

   (d) **Deleting Data:**
      - Deletes a user by ID (e.g., `UserID = 102`).
      - Deletes a training plan by ID (e.g., `PlanID = 3`) and reassigns its associated users.

   (e) **Database Deletion:**
      - Optionally deletes the entire database. Password protection and user confirmation are required to prevent accidental execution.
      - This operation is commented out for safety during testing.

4 Execution Logs

The following logs were produced during the execution of the `Main` class, showcasing the successful operations performed on the `PersonalizedFitness` database:

```
Connected to PersonalizedFitness database.
Connected to database: PersonalizedFitness
Users Collection: PersonalizedFitness.Users
TrainingPlans Collection: PersonalizedFitness.TrainingPlans
Connected to database: PersonalizedFitness
Users Collection: PersonalizedFitness.Users
TrainingPlans Collection: PersonalizedFitness.TrainingPlans
Connected to database: PersonalizedFitness
Users Collection: PersonalizedFitness.Users
TrainingPlans Collection: PersonalizedFitness.TrainingPlans
Connected to database: PersonalizedFitness
User with UserID 117 already exists. Skipping...

Reading User with UserID 117:
User Found: {"_id": {"$oid": "678c354ad1ef3d681783b8c6"}, "UserID
   ": 117, "Name": "Daniel", "Age": 21, "TrainingPlanID": 5}

Reading Training Plan with PlanID 5:
Training Plan Found: {"_id": {"$oid": "678c191907d1422c40bae77b
   "}, "PlanID": 5, "Goal": "General Fitness", "Exercises": [{"
   Name": "Circuit Training", "Duration": 45, "CaloriesBurned":
   350}, {"Name": "Burpees", "Duration": 15, "CaloriesBurned":
   150}]}

Reading all Users:
All Users:
{"_id": {"$oid": "678c191907d1422c40bae77c"}, "UserID": 101, "
   Name": "Alice", "Age": 29, "TrainingPlanID": 2}
{"_id": {"$oid": "678c191907d1422c40bae77e"}, "UserID": 103, "
   Name": "Charlie", "Age": 46, "TrainingPlanID": 4}
{"_id": {"$oid": "678c191907d1422c40bae77f"}, "UserID": 104, "
   Name": "Diana", "Age": 21, "TrainingPlanID": 4}
{"_id": {"$oid": "678c191907d1422c40bae780"}, "UserID": 105, "
   Name": "Eve", "Age": 26, "TrainingPlanID": 1}
{"_id": {"$oid": "678c191907d1422c40bae781"}, "UserID": 106, "
   Name": "Frank", "Age": 35, "TrainingPlanID": 2}
{"_id": {"$oid": "678c191907d1422c40bae782"}, "UserID": 107, "
   Name": "Grace", "Age": 50, "TrainingPlanID": 4}
{"_id": {"$oid": "678c191a07d1422c40bae783"}, "UserID": 108, "
   Name": "Hank", "Age": 24, "TrainingPlanID": 4}
{"_id": {"$oid": "678c191a07d1422c40bae784"}, "UserID": 109, "
   Name": "Ivy", "Age": 34, "TrainingPlanID": 5}
{"_id": {"$oid": "678c191a07d1422c40bae785"}, "UserID": 110, "
   Name": "Jack", "Age": 32, "TrainingPlanID": 1}
{"_id": {"$oid": "678c191a07d1422c40bae786"}, "UserID": 111, "
   Name": "Kate", "Age": 45, "TrainingPlanID": 2}
```

```
Reading all Training Plans:
All Training Plans:
{"_id": {"$oid": "678c191907d1422c40bae777"}, "PlanID": 1, "Goal
   ": "Updated Goal 1", "Exercises": [{"Name": "Cardio Routine",
   "Duration": 45, "CaloriesBurned": 300}, {"Name": "Jumping
   Jacks", "Duration": 15, "CaloriesBurned": 150}]}
{"_id": {"$oid": "678c191907d1422c40bae778"}, "PlanID": 2, "Goal
   ": "Muscle Gain", "Exercises": [{"Name": "Weight Lifting", "
   Duration": 60, "CaloriesBurned": 250}, {"Name": "Push-Ups", "
   Duration": 20, "CaloriesBurned": 100}]}
{"_id": {"$oid": "678c191907d1422c40bae77a"}, "PlanID": 4, "Goal
   ": "Flexibility", "Exercises": [{"Name": "Yoga", "Duration":
   60, "CaloriesBurned": 200}, {"Name": "Stretching", "Duration":
    30, "CaloriesBurned": 100}]}
{"_id": {"$oid": "678c191907d1422c40bae77b"}, "PlanID": 5, "Goal
   ": "General Fitness", "Exercises": [{"Name": "Circuit Training
   ", "Duration": 45, "CaloriesBurned": 350}, {"Name": "Burpees",
    "Duration": 15, "CaloriesBurned": 150}]}
Updated UserID 101 with new Name: Alice
Training Plan with PlanID 3 not found. No update performed.
Updated UserID 104 with new TrainingPlanID: 4
User with UserID 102 not found. No deletion performed.
Training Plan with PlanID 3 not found. No deletion performed.
```

These logs highlight the integration and functioning of various operations within the `Main` class, demonstrating data creation, retrieval, updating, and deletion in the `PersonalizedFitness` database.

Significance

The `Main` class serves as the central hub that integrates and tests all functionalities of the `PersonalizedFitness` project. By demonstrating various operations on users and training plans, it highlights the versatility and robustness of the implemented classes.

5 Conclusion

The MongoDB Java Project provided an extensive exploration of working with MongoDB in a Java environment. Through a series of well-structured modules, we were able to demonstrate the fundamental operations required for efficient data management, including creating, reading, updating, and deleting records.

The project's journey began with the *Sample Training* module, where we developed foundational CRUD skills using MongoDB collections. This experience served as a stepping stone for the more advanced and practical *Personalized Fitness* module. In this phase, we implemented dynamic and reusable components that allowed seamless interactions with the MongoDB database.

Notable achievements of the project include:

- Designing dynamic constructors for modular database operations.
- Implementing real-world use cases such as user and training plan management.
- Incorporating robust update and delete functionalities, ensuring data consistency and user-centric operations.
- Employing security measures, such as password protection for database deletion, to prevent accidental data loss.
- Creating a comprehensive logging system to document execution results and monitor database activities.

Overall, this project not only strengthened our technical expertise in MongoDB and Java integration but also highlighted the importance of structuring code to support real-world applications. The skills and knowledge gained through this endeavor will serve as a foundation for tackling more complex data-driven challenges in the future.