Spring Data – Mongo Template

Examples of using Spring Boot Data MongoTemplate for CRUD operations on a simple User collection.

The User class is structured as follows:

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "users")

public class User {

@Id

private String id;

private String name;

private String email;

private int age;

// Getters and setters

}

**Create Operation**

To create a new user, you can use the save method of MongoTemplate.

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.mongodb.core.MongoTemplate;

import org.springframework.stereotype.Service;

@Service

public class UserService {

@Autowired

private MongoTemplate mongoTemplate;

public User createUser(User user) {

return mongoTemplate.save(user);

}

}

**Read Operations**

To read or retrieve users, you can use the findById, findAll, and find methods.

**Find User by ID**

import org.springframework.data.mongodb.core.query.Criteria;

import org.springframework.data.mongodb.core.query.Query;

public User getUserById(String id) {

return mongoTemplate.findById(id, User.class);

}

**Find All Users**

public List<User> getAllUsers() {

return mongoTemplate.findAll(User.class);

}

**Find Users by Age**

public List<User> getUsersByAge(int age) {

Query query = new Query();

query.addCriteria(Criteria.where("age").is(age));

return mongoTemplate.find(query, User.class);

}

**Update Operation**

To update an existing user, you can use the findAndModify method with update operations.

import org.springframework.data.mongodb.core.query.Update;

public User updateUser(String id, User user) {

Query query = new Query(Criteria.where("id").is(id));

Update update = new Update()

.set("name", user.getName())

.set("email", user.getEmail())

.set("age", user.getAge());

return mongoTemplate.findAndModify(query, update, User.class);

}

**Delete Operation**

To delete a user, you can use the remove method.

public void deleteUser(String id) {

Query query = new Query(Criteria.where("id").is(id));

mongoTemplate.remove(query, User.class);

}

**Complete Service Class**

Here is a complete UserService class with all CRUD operations:

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.mongodb.core.MongoTemplate;

import org.springframework.data.mongodb.core.query.Criteria;

import org.springframework.data.mongodb.core.query.Query;

import org.springframework.data.mongodb.core.query.Update;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class UserService {

@Autowired

private MongoTemplate mongoTemplate;

// Create a new user

public User createUser(User user) {

return mongoTemplate.save(user);

}

// Get a user by ID

public User getUserById(String id) {

return mongoTemplate.findById(id, User.class);

}

// Get all users

public List<User> getAllUsers() {

return mongoTemplate.findAll(User.class);

}

// Get users by age

public List<User> getUsersByAge(int age) {

Query query = new Query();

query.addCriteria(Criteria.where("age").is(age));

return mongoTemplate.find(query, User.class);

}

// Update an existing user

public User updateUser(String id, User user) {

Query query = new Query(Criteria.where("id").is(id));

Update update = new Update()

.set("name", user.getName())

.set("email", user.getEmail())

.set("age", user.getAge());

return mongoTemplate.findAndModify(query, update, User.class);

}

// Delete a user

public void deleteUser(String id) {

Query query = new Query(Criteria.where("id").is(id));

mongoTemplate.remove(query, User.class);

}

}

**Controller Class**

Here is a simple UserController class to expose these operations via REST API:

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/users")

public class UserController {

@Autowired

private UserService userService;

@PostMapping

public ResponseEntity<User> createUser(@RequestBody User user) {

User createdUser = userService.createUser(user);

return ResponseEntity.ok(createdUser);

}

@GetMapping("/{id}")

public ResponseEntity<User> getUserById(@PathVariable String id) {

User user = userService.getUserById(id);

return ResponseEntity.ok(user);

}

@GetMapping

public ResponseEntity<List<User>> getAllUsers() {

List<User> users = userService.getAllUsers();

return ResponseEntity.ok(users);

}

@GetMapping("/age/{age}")

public ResponseEntity<List<User>> getUsersByAge(@PathVariable int age){ List<User> users = userService.getUsersByAge(age);

return ResponseEntity.ok(users);

}

@PutMapping("/{id}")

public ResponseEntity<User> updateUser(@PathVariable String id, @RequestBody User user) {

User updatedUser = userService.updateUser(id, user);

return ResponseEntity.ok(updatedUser);

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteUser(@PathVariable String id) {

userService.deleteUser(id);

return ResponseEntity.noContent().build();

}

}

**Main Class**

This is the entry point of the Spring Boot application.

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class UserManagementApplication {

public static void main(String[] args) {

SpringApplication.run(UserManagementApplication.class, args);

}

}

**application.properties**

The application.properties file configures the connection to the MongoDB database.

properties

Copy code

spring.data.mongodb.uri=mongodb://localhost:27017/userdb

spring.data.mongodb.database=userdb

server.port=8080

**Directory Structure**

The directory structure of the project should look like this:

user-management-backend

├── src

│ ├── main

│ │ ├── java

│ │ │ └── com

│ │ │ └── example

│ │ │ └── usermanagement

│ │ │ ├── UserManagementApplication.java

│ │ │ ├── User.java

│ │ │ ├── UserService.java

│ │ │ └── UserController.java

│ │ └── resources

│ │ └── application.properties

├── mvnw

├── mvnw.cmd

├── pom.xml

└── README.md

Here is a sample pom.xml file for this project.

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>user-management</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>jar</packaging>

<name>user-management</name>

<description>User Management API</description>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.7.4</version>

<relativePath/>

</parent>

<properties>

<java.version>11</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-mongodb</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**Running the Application**

To run the application, you can use the following Maven command:

./mvnw spring-boot:run

For example, to create a new user, you can send a POST request to http://localhost:8080/users with a JSON body:

{

"name": "Krishna Murthy",

"email": "krishna@example.com",

"age": 50 }

**Find All Users**

**Using find()**

public List<User> findAllUsers() {

return mongoTemplate.findAll(User.class);

}

**Using Aggregation Pipeline**

import org.springframework.data.mongodb.core.aggregation.Aggregation;

public List<User> findAllUsersWithAggregation() {

Aggregation aggregation = Aggregation.newAggregation(Aggregation.match(new Criteria()));

return mongoTemplate.aggregate(aggregation, "users", User.class).getMappedResults();

}

**Find Users by Age**

**Using find()**

public List<User> findUsersByAge(int age) {

Query query = new Query(Criteria.where("age").is(age));

return mongoTemplate.find(query, User.class);

}

**Using Aggregation Pipeline**

import org.springframework.data.mongodb.core.aggregation.Aggregation;

import org.springframework.data.mongodb.core.aggregation.MatchOperation;

import org.springframework.data.mongodb.core.query.Criteria;

public List<User> findUsersByAgeWithAggregation(int age) {

MatchOperation matchOperation = Aggregation.match(Criteria.where("age").is(age));

Aggregation aggregation = Aggregation.newAggregation(matchOperation);

return mongoTemplate.aggregate(aggregation, "users", User.class).getMappedResults();

}

**Find Users by Name**

**Using find()**

public List<User> findUsersByName(String name) {

Query query = new Query(Criteria.where("name").is(name));

return mongoTemplate.find(query, User.class);

}

**Using Aggregation Pipeline**

public List<User> findUsersByNameWithAggregation(String name) {

MatchOperation matchOperation = Aggregation.match(Criteria.where("name").is(name));

Aggregation aggregation = Aggregation.newAggregation(matchOperation);

return mongoTemplate.aggregate(aggregation, "users", User.class).getMappedResults();

}

**Find Users by Email Domain**

**Using find()**

public List<User> findUsersByEmailDomain(String domain) {

Query query = new Query(Criteria.where("email").regex(".\*@" + domain + ".\*"));

return mongoTemplate.find(query, User.class);

}

**Using Aggregation Pipeline**

public List<User> findUsersByEmailDomainWithAggregation(String domain) {

MatchOperation matchOperation = Aggregation.match(Criteria.where("email").regex(".\*@" + domain + ".\*"));

Aggregation aggregation = Aggregation.newAggregation(matchOperation);

return mongoTemplate.aggregate(aggregation, "users", User.class).getMappedResults();

}

**Example Service Class with Queries**

Here is a complete UserService class including these queries:

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.mongodb.core.MongoTemplate;

import org.springframework.data.mongodb.core.aggregation.Aggregation;

import org.springframework.data.mongodb.core.aggregation.MatchOperation;

import org.springframework.data.mongodb.core.query.Criteria;

import org.springframework.data.mongodb.core.query.Query;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class UserService {

@Autowired

private MongoTemplate mongoTemplate;

// Create a new user

public User createUser(User user) {

return mongoTemplate.save(user);

}

// Get a user by ID

public User getUserById(String id) {

return mongoTemplate.findById(id, User.class);

}

// Get all users using find()

public List<User> findAllUsers() {

return mongoTemplate.findAll(User.class);

}

// Get all users using aggregation pipeline

public List<User> findAllUsersWithAggregation() {

Aggregation aggregation = Aggregation.newAggregation(Aggregation.match(new Criteria()));

return mongoTemplate.aggregate(aggregation, "users", User.class).getMappedResults();

}

// Get users by age using find()

public List<User> findUsersByAge(int age) {

Query query = new Query(Criteria.where("age").is(age));

return mongoTemplate.find(query, User.class);

}

// Get users by age using aggregation pipeline

public List<User> findUsersByAgeWithAggregation(int age) {

MatchOperation matchOperation = Aggregation.match(Criteria.where("age").is(age));

Aggregation aggregation = Aggregation.newAggregation(matchOperation);

return mongoTemplate.aggregate(aggregation, "users", User.class).getMappedResults();

}

// Get users by name using find()

public List<User> findUsersByName(String name) {

Query query = new Query(Criteria.where("name").is(name));

return mongoTemplate.find(query, User.class);

}

// Get users by name using aggregation pipeline

public List<User> findUsersByNameWithAggregation(String name) {

MatchOperation matchOperation = Aggregation.match(Criteria.where("name").is(name));

Aggregation aggregation = Aggregation.newAggregation(matchOperation);

return mongoTemplate.aggregate(aggregation, "users", User.class).getMappedResults();

}

// Get users by email domain using find()

public List<User> findUsersByEmailDomain(String domain) {

Query query = new Query(Criteria.where("email").regex(".\*@" + domain + ".\*"));

return mongoTemplate.find(query, User.class);

}

// Get users by email domain using aggregation pipeline

public List<User> findUsersByEmailDomainWithAggregation(String domain) {

MatchOperation matchOperation = Aggregation.match(Criteria.where("email").regex(".\*@" + domain + ".\*"));

Aggregation aggregation = Aggregation.newAggregation(matchOperation);

return mongoTemplate.aggregate(aggregation, "users", User.class).getMappedResults();

}

// Update an existing user

public User updateUser(String id, User user) {

Query query = new Query(Criteria.where("id").is(id));

Update update = new Update()

.set("name", user.getName())

.set("email", user.getEmail())

.set("age", user.getAge());

return mongoTemplate.findAndModify(query, update, User.class);

}

// Delete a user

public void deleteUser(String id) {

Query query = new Query(Criteria.where("id").is(id));

mongoTemplate.remove(query, User.class);

}

}