**Let’s go step by step on how to work with MongoDB and Java.**

**1. Add MongoDB Java Driver**

To connect Java with MongoDB, you need the official driver. If using **Maven**, add this to your pom.xml:

<dependency>

<groupId>org.mongodb</groupId>

<artifactId>mongodb-driver-sync</artifactId>

<version>5.1.0</version> <!-- check latest -->

</dependency>

For **Gradle**:

implementation 'org.mongodb:mongodb-driver-sync:5.1.0'

**2. Connect to MongoDB**

import com.mongodb.client.MongoClients;

import com.mongodb.client.MongoClient;

import com.mongodb.client.MongoDatabase;

public class MongoConnect {

public static void main(String[] args) {

// connect to local MongoDB

try (MongoClient mongoClient = MongoClients.create("mongodb://localhost:27017")) {

MongoDatabase db = mongoClient.getDatabase("testdb");

System.out.println("Connected to database: " + db.getName());

}

}

}

**3. Insert Documents**

import com.mongodb.client.MongoCollection;

import com.mongodb.client.MongoDatabase;

import org.bson.Document;

import com.mongodb.client.MongoClients;

import com.mongodb.client.MongoClient;

public class InsertExample {

public static void main(String[] args) {

try (MongoClient client = MongoClients.create("mongodb://localhost:27017")) {

MongoDatabase db = client.getDatabase("testdb");

MongoCollection<Document> collection = db.getCollection("users");

Document user = new Document("name", "Alice")

.append("age", 25)

.append("email", "alice@example.com");

collection.insertOne(user);

System.out.println("Document inserted!");

}

}

}

**4. Query Documents**

import com.mongodb.client.MongoCursor;

public class QueryExample {

public static void main(String[] args) {

try (MongoClient client = MongoClients.create("mongodb://localhost:27017")) {

MongoDatabase db = client.getDatabase("testdb");

MongoCollection<Document> collection = db.getCollection("users");

// find all users

MongoCursor<Document> cursor = collection.find().iterator();

while (cursor.hasNext()) {

System.out.println(cursor.next().toJson());

}

// find with filter

Document query = new Document("name", "Alice");

Document result = collection.find(query).first();

System.out.println("Found: " + result.toJson());

}

}

}

**5. Update and Delete**

import com.mongodb.client.model.Updates;

import com.mongodb.client.model.Filters;

public class UpdateDeleteExample {

public static void main(String[] args) {

try (MongoClient client = MongoClients.create("mongodb://localhost:27017")) {

MongoDatabase db = client.getDatabase("testdb");

MongoCollection<Document> collection = db.getCollection("users");

// Update

collection.updateOne(Filters.eq("name", "Alice"),

Updates.set("age", 26));

// Delete

collection.deleteOne(Filters.eq("name", "Alice"));

System.out.println("Update/Delete complete");

}

}

}

**6. POJO Mapping (Optional, Cleaner)**

MongoDB supports mapping Java classes to documents with mongodb-driver-sync + PojoCodecProvider.

import org.bson.codecs.configuration.CodecRegistry;

import org.bson.codecs.pojo.PojoCodecProvider;

import static org.bson.codecs.configuration.CodecRegistries.fromProviders;

import static org.bson.codecs.configuration.CodecRegistries.fromRegistries;

CodecRegistry pojoCodecRegistry = fromRegistries(

MongoClientSettings.getDefaultCodecRegistry(),

fromProviders(PojoCodecProvider.builder().automatic(true).build())

);

MongoDatabase db = client.getDatabase("testdb").withCodecRegistry(pojoCodecRegistry);

Then you can use plain Java classes (User class with fields) instead of Document.

So in summary:

1. Add MongoDB driver.
2. Connect with MongoClients.
3. Perform CRUD using MongoCollection<Document>.
4. Optionally use POJO mapping.

Here’s a compact, **runnable core-Java** example that shows **CRUD + searching + sorting + pagination** on an Employee entity using the MongoDB **sync** driver with POJO mapping.

**What you get**

* POJO Employee
* Connection + POJO codec setup
* Create/Insert (single & bulk)
* Read (by id, all)
* Search: equality, range, regex “contains”, membership (skills)
* Sorting (multi-field), Pagination
* Update (single & bulk)
* Delete (single & bulk)
* Helpful indexes

### **MongoEmployeeDemo.java**

import com.mongodb.MongoClientSettings;

import com.mongodb.client.\*;

import com.mongodb.client.model.\*;

import org.bson.Document;

import org.bson.codecs.configuration.CodecRegistry;

import org.bson.codecs.pojo.PojoCodecProvider;

import org.bson.types.ObjectId;

import org.bson.conversions.Bson;

import java.time.Instant;

import java.util.\*;

import static com.mongodb.client.model.Filters.\*;

import static com.mongodb.client.model.Updates.\*;

import static com.mongodb.client.model.Sorts.\*;

import static org.bson.codecs.configuration.CodecRegistries.fromProviders;

import static org.bson.codecs.configuration.CodecRegistries.fromRegistries;

public class MongoEmployeeDemo {

// ----- POJO -----

public static class Employee {

private ObjectId id; // \_id

private String name;

private String department;

private double salary;

private Instant hiredAt; // requires recent driver; use java.util.Date if you prefer

private List<String> skills;

private boolean active;

public Employee() {} // REQUIRED by POJO codec

public Employee(String name, String department, double salary,

Instant hiredAt, List<String> skills, boolean active) {

this.name = name;

this.department = department;

this.salary = salary;

this.hiredAt = hiredAt;

this.skills = skills;

this.active = active;

}

// Getters & setters (required)

public ObjectId getId() { return id; }

public void setId(ObjectId id) { this.id = id; }

public String getName() { return name; }

public void setName(String name) { this.name = name; }

public String getDepartment() { return department; }

public void setDepartment(String department) { this.department = department; }

public double getSalary() { return salary; }

public void setSalary(double salary) { this.salary = salary; }

public Instant getHiredAt() { return hiredAt; }

public void setHiredAt(Instant hiredAt) { this.hiredAt = hiredAt; }

public List<String> getSkills() { return skills; }

public void setSkills(List<String> skills) { this.skills = skills; }

public boolean isActive() { return active; }

public void setActive(boolean active) { this.active = active; }

@Override public String toString() {

return "Employee{" +

"id=" + id +

", name='" + name + '\'' +

", department='" + department + '\'' +

", salary=" + salary +

", hiredAt=" + hiredAt +

", skills=" + skills +

", active=" + active +

'}';

}

}

public static void main(String[] args) {

// 1) POJO codec setup

CodecRegistry pojoRegistry = fromRegistries(

MongoClientSettings.getDefaultCodecRegistry(),

fromProviders(PojoCodecProvider.builder().automatic(true).build())

);

// 2) Connect

try (MongoClient client = MongoClients.create("mongodb://localhost:27017")) {

MongoDatabase db = client.getDatabase("company").withCodecRegistry(pojoRegistry);

MongoCollection<Employee> employees = db.getCollection("employees", Employee.class);

// Clean slate for demo

employees.drop();

// 3) Indexes (help searches)

employees.createIndex(Indexes.ascending("name"));

employees.createIndex(Indexes.ascending("department", "salary"));

employees.createIndex(Indexes.descending("hiredAt"));

employees.createIndex(Indexes.ascending("skills"));

// 4) Seed data (CREATE - insertMany)

seedData(employees);

// 5) CREATE - insertOne

Employee e = new Employee("Karthik", "Sales", 6.2, Instant.now(),

List.of("negotiation", "excel"), true);

employees.insertOne(e);

System.out.println("Inserted one: " + e.getId());

// 6) READ - find all

System.out.println("\nAll employees:");

employees.find().forEach(System.out::println);

// 7) READ - find by id

Employee foundById = employees.find(eq("\_id", e.getId())).first();

System.out.println("\nFound by \_id: " + foundById);

// 8) SEARCH - by equality

System.out.println("\nSearch: department == 'Engineering'");

employees.find(eq("department", "Engineering")).forEach(System.out::println);

// 9) SEARCH - regex contains (case-insensitive)

System.out.println("\nSearch: name contains 'an' (case-insensitive)");

employees.find(regex("name", "an", "i")).forEach(System.out::println);

// 10) SEARCH - range + AND

System.out.println("\nSearch: dept=Engineering AND 7.0 <= salary <= 10.0");

employees.find(and(eq("department", "Engineering"), gte("salary", 7.0), lte("salary", 10.0)))

.forEach(System.out::println);

// 11) SEARCH - set membership

System.out.println("\nSearch: skills contains 'react'");

employees.find(eq("skills", "react")).forEach(System.out::println);

// 12) SORTING (multi-field)

System.out.println("\nSorted: by salary DESC, then name ASC");

employees.find()

.sort(orderBy(descending("salary"), ascending("name")))

.forEach(System.out::println);

// 13) PAGINATION (page=2, size=3)

int page = 2, size = 3;

System.out.println("\nPagination: page " + page + ", size " + size);

employees.find()

.sort(ascending("name"))

.skip((page - 1) \* size)

.limit(size)

.forEach(System.out::println);

// 14) UPDATE - single

System.out.println("\nUpdate: set Karthik's salary = 6.8 and add skill 'crm'");

UpdateResult ur1 = employees.updateOne(eq("\_id", e.getId()),

combine(set("salary", 6.8), addToSet("skills", "crm")));

System.out.println("Matched: " + ur1.getMatchedCount() + ", Modified: " + ur1.getModifiedCount());

System.out.println("After update: " + employees.find(eq("\_id", e.getId())).first());

// 15) UPDATE - bulk (give 10% raise to Engineering)

System.out.println("\nBulk Update: +10% salary for Engineering");

UpdateResult ur2 = employees.updateMany(eq("department", "Engineering"),

mul("salary", 1.10));

System.out.println("Matched: " + ur2.getMatchedCount() + ", Modified: " + ur2.getModifiedCount());

// 16) DELETE - single

System.out.println("\nDelete: remove Karthik by \_id");

DeleteResult dr1 = employees.deleteOne(eq("\_id", e.getId()));

System.out.println("Deleted: " + dr1.getDeletedCount());

// 17) DELETE - bulk (inactive)

System.out.println("\nBulk Delete: remove inactive employees");

DeleteResult dr2 = employees.deleteMany(eq("active", false));

System.out.println("Deleted: " + dr2.getDeletedCount());

// 18) Final list

System.out.println("\nRemaining employees:");

employees.find().forEach(System.out::println);

}

}

private static void seedData(MongoCollection<Employee> employees) {

List<Employee> batch = List.of(

new Employee("Ananya", "Engineering", 9.5, Instant.parse("2022-03-10T00:00:00Z"),

List.of("java", "spring", "mongodb"), true),

new Employee("Rohan", "Engineering", 7.2, Instant.parse("2023-07-01T00:00:00Z"),

List.of("python", "flask"), true),

new Employee("Meera", "HR", 5.1, Instant.parse("2021-11-20T00:00:00Z"),

List.of("recruiting", "communication"), true),

new Employee("Vikram", "Engineering", 10.4, Instant.parse("2020-01-15T00:00:00Z"),

List.of("react", "node", "docker"), true),

new Employee("Sana", "Design", 6.8, Instant.parse("2024-02-05T00:00:00Z"),

List.of("figma", "ux", "html"), false),

new Employee("Nandini", "Finance", 8.0, Instant.parse("2019-06-25T00:00:00Z"),

List.of("excel", "tally"), true),

new Employee("Arun", "Engineering", 7.9, Instant.parse("2023-01-03T00:00:00Z"),

List.of("java", "spring", "kafka"), true),

new Employee("Ishaan", "Engineering", 8.7, Instant.parse("2022-08-19T00:00:00Z"),

List.of("go", "kubernetes"), true)

);

employees.insertMany(batch);

System.out.println("Seeded " + batch.size() + " employees.");

}

}

### How to run

1. Add dependency (Maven):

<dependency>

<groupId>org.mongodb</groupId>

<artifactId>mongodb-driver-sync</artifactId>

<version>5.1.0</version> <!-- or latest -->

</dependency>

1. Ensure MongoDB is running locally at mongodb://localhost:27017.
2. Compile & run:
3. javac -cp .:mongo.jar MongoEmployeeDemo.java
4. java -cp .:mongo.jar MongoEmployeeDemo

### Notes / Tweaks

* If Instant causes issues on older driver versions, switch hiredAt to java.util.Date.
* In real apps, avoid dropping collections; use migrations/seeders.
* Consider **validation rules** and **unique indexes** (e.g., unique email/empCode).
* For large lists, prefer **server-side pagination** using a stable sort + skip/limit or **range pagination** with filters.

**Here’s a clean, runnable Spring Boot + Spring Data MongoDB example that does CRUD + search + sort + pagination on an Employee collection.**

# 0) Prereqs

* **Java 17+**
* **MongoDB** running locally on mongodb://localhost:27017
* Build with **Maven** (Gradle variant included)

# 1) pom.xml

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>net.sivaacademy</groupId>

<artifactId>spring-mongo-demo</artifactId>

<version>0.0.1</version>

<properties>

<java.version>17</java.version>

<spring-boot.version>3.3.3</spring-boot.version>

</properties>

<dependencyManagement>

<dependencies>

<dependency>

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

<artifactId>spring-boot-dependencies</artifactId>

<version>${spring-boot.version}</version>

<type>pom</type><scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<!-- optional: bean validation for request DTOs -->

<dependency>

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

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

**Gradle (Kotlin) quick alt**

dependencies {

implementation("org.springframework.boot:spring-boot-starter-web")

implementation("org.springframework.boot:spring-boot-starter-data-mongodb")

implementation("org.springframework.boot:spring-boot-starter-validation")

testImplementation("org.springframework.boot:spring-boot-starter-test")

}

# 2) application.yml

spring:

application:

name: spring-mongo-demo

data:

mongodb:

uri: mongodb://localhost:27017/companydb

jackson:

serialization:

WRITE\_DATES\_AS\_TIMESTAMPS: false

server:

port: 8080

# **3) Domain Model — Employee.java**

package net.sivaacademy.demo.employee;

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.index.Indexed;

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

import java.time.Instant;

import java.util.List;

@Document("employees")

public class Employee {

@Id

private String id;

@Indexed // speeds up name queries

private String name;

@Indexed

private String department;

private double salary;

private Instant hiredAt;

private List<String> skills;

private boolean active = true;

public Employee() {}

public Employee(String name, String department, double salary,

Instant hiredAt, List<String> skills, boolean active) {

this.name = name;

this.department = department;

this.salary = salary;

this.hiredAt = hiredAt;

this.skills = skills;

this.active = active;

}

// getters/setters …

// (Generate with IDE to keep this short)

}

# **4) Repository — EmployeeRepository.java**

package net.sivaacademy.demo.employee;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.mongodb.repository.MongoRepository;

import org.springframework.data.mongodb.repository.Query;

import java.util.List;

public interface EmployeeRepository extends MongoRepository<Employee, String> {

// Derived queries

List<Employee> findByDepartment(String department);

Page<Employee> findByActive(boolean active, Pageable pageable);

// Contains (regex, case-insensitive) search on name

@Query("{ 'name': { $regex: ?0, $options: 'i' } }")

Page<Employee> searchByName(String namePart, Pageable pageable);

// Range + AND example

Page<Employee> findByDepartmentAndSalaryBetween(String department, double min, double max, Pageable pageable);

// Skills contains an item

Page<Employee> findBySkillsContaining(String skill, Pageable pageable);

}

# **5) Service (optional but neat) — EmployeeService.java**

package net.sivaacademy.demo.employee;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.stereotype.Service;

import java.time.Instant;

import java.util.List;

import java.util.Optional;

@Service

public class EmployeeService {

private final EmployeeRepository repo;

public EmployeeService(EmployeeRepository repo) { this.repo = repo; }

public Employee create(Employee e) {

if (e.getHiredAt() == null) e.setHiredAt(Instant.now());

return repo.save(e);

}

public Optional<Employee> get(String id) { return repo.findById(id); }

public Page<Employee> list(Pageable pageable) { return repo.findAll(pageable); }

public Page<Employee> searchByName(String q, Pageable p) { return repo.searchByName(q, p); }

public Page<Employee> byDeptAndSalary(String dept, double min, double max, Pageable p) {

return repo.findByDepartmentAndSalaryBetween(dept, min, max, p);

}

public Page<Employee> bySkill(String skill, Pageable p) { return repo.findBySkillsContaining(skill, p); }

public Employee update(String id, Employee patch) {

Employee e = repo.findById(id).orElseThrow();

if (patch.getName() != null) e.setName(patch.getName());

if (patch.getDepartment() != null) e.setDepartment(patch.getDepartment());

if (patch.getSkills() != null) e.setSkills(patch.getSkills());

if (patch.getHiredAt() != null) e.setHiredAt(patch.getHiredAt());

if (patch.getSalary() != 0) e.setSalary(patch.getSalary());

e.setActive(patch.isActive() || e.isActive()); // simple example

return repo.save(e);

}

public void delete(String id) { repo.deleteById(id); }

public long deactivateInactive() { // example bulk op

var all = repo.findAll();

long count = 0;

for (var e : all) {

if (!e.isActive()) continue;

if (e.getSalary() < 5.0) { e.setActive(false); repo.save(e); count++; }

}

return count;

}

}

# **6) Controller — EmployeeController.java**

package net.sivaacademy.demo.employee;

import jakarta.validation.Valid;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Sort;

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

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

private final EmployeeService svc;

public EmployeeController(EmployeeService svc) { this.svc = svc; }

// CREATE

@PostMapping

public Employee create(@Valid @RequestBody Employee e) {

return svc.create(e);

}

// READ one

@GetMapping("/{id}")

public Employee get(@PathVariable String id) {

return svc.get(id).orElseThrow();

}

// READ many with pagination + sorting

@GetMapping

public Page<Employee> list(@RequestParam(defaultValue = "0") int page,

@RequestParam(defaultValue = "10") int size,

@RequestParam(defaultValue = "name,asc") String sort) {

String[] s = sort.split(",");

var pageable = PageRequest.of(page, size, Sort.by(

"desc".equalsIgnoreCase(s.length > 1 ? s[1] : "asc")

? Sort.Order.desc(s[0]) : Sort.Order.asc(s[0])));

return svc.list(pageable);

}

// SEARCH name contains (case-insensitive) + pagination/sort

@GetMapping("/search")

public Page<Employee> search(@RequestParam String q,

@RequestParam(defaultValue = "0") int page,

@RequestParam(defaultValue = "10") int size) {

var pageable = PageRequest.of(page, size, Sort.by("name").ascending());

return svc.searchByName(q, pageable);

}

// FILTER complex: dept + salary range

@GetMapping("/filter")

public Page<Employee> filter(@RequestParam String dept,

@RequestParam double min,

@RequestParam double max,

@RequestParam(defaultValue = "0") int page,

@RequestParam(defaultValue = "10") int size) {

return svc.byDeptAndSalary(dept, min, max, PageRequest.of(page, size, Sort.by("salary").descending()));

}

// FILTER by skill

@GetMapping("/skill")

public Page<Employee> bySkill(@RequestParam String skill,

@RequestParam(defaultValue = "0") int page,

@RequestParam(defaultValue = "10") int size) {

return svc.bySkill(skill, PageRequest.of(page, size));

}

// UPDATE (partial)

@PatchMapping("/{id}")

public Employee update(@PathVariable String id, @RequestBody Employee patch) {

return svc.update(id, patch);

}

// DELETE

@DeleteMapping("/{id}")

public void delete(@PathVariable String id) { svc.delete(id); }

}

# **7) Bootstrap & Sample Data — DemoApplication.java**

package net.sivaacademy.demo;

import net.sivaacademy.demo.employee.Employee;

import net.sivaacademy.demo.employee.EmployeeRepository;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import java.time.Instant;

import java.util.List;

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) { SpringApplication.run(DemoApplication.class, args); }

// seed data for quick testing

CommandLineRunner seed(EmployeeRepository repo) {

return args -> {

if (repo.count() == 0) {

repo.saveAll(List.of(

new Employee("Ananya", "Engineering", 9.5, Instant.parse("2022-03-10T00:00:00Z"), List.of("java","spring","mongodb"), true),

new Employee("Rohan", "Engineering", 7.2, Instant.parse("2023-07-01T00:00:00Z"), List.of("python","flask"), true),

new Employee("Meera", "HR", 5.1, Instant.parse("2021-11-20T00:00:00Z"), List.of("recruiting","communication"), true),

new Employee("Vikram", "Engineering", 10.4, Instant.parse("2020-01-15T00:00:00Z"), List.of("react","node","docker"), true),

new Employee("Sana", "Design", 6.8, Instant.parse("2024-02-05T00:00:00Z"), List.of("figma","ux","html"), false)

));

}

};

}

}

# 8) Run it

# in the project folder

./mvnw spring-boot:run

# or

mvn spring-boot:run

# 9) Try it (sample requests)

# List (page=0,size=3, sorted by name ASC)

curl "http://localhost:8080/api/employees?page=0&size=3&sort=name,asc"

# Search by name contains "an"

curl "http://localhost:8080/api/employees/search?q=an"

# Filter dept + salary range

curl "http://localhost:8080/api/employees/filter?dept=Engineering&min=7&max=11"

# Filter by skill

curl "http://localhost:8080/api/employees/skill?skill=react"

# Create

curl -X POST "http://localhost:8080/api/employees" -H "Content-Type: application/json" -d '{

"name":"Karthik","department":"Sales","salary":6.2,

"skills":["negotiation","excel"],"active":true

}'

# Update (partial)

curl -X PATCH "http://localhost:8080/api/employees/{id}" -H "Content-Type: application/json" -d '{

"salary": 6.8, "skills":["excel","crm"]

}'

# Delete

curl -X DELETE "http://localhost:8080/api/employees/{id}"

## Extras you can add

* **Compound index** on {department: 1, salary: -1} using @Indexed on fields or @CompoundIndex at class level.
* **DTOs + Validation** for request payloads.
* **Soft delete**: add deletedAt field and filter globally using MongoTemplate/custom repo.
* **Transactions**: inside Mongo only (replica set required), use @EnableMongoAuditing for created/updated timestamps.

**Here’s a minimal, runnable Spring Boot project that exposes MongoDB collections for Products, Customers, and Orders automatically via Spring Data REST (HAL/JSON with paging, sorting, and search).**

It uses **ID references** (not DBRef) and an embedded OrderItem array inside Order.

# 0) Prereqs

* Java 17+
* MongoDB running at mongodb://localhost:27017
* Maven (Gradle alt included)

# 1) pom.xml

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>net.sivaacademy</groupId>

<artifactId>spring-data-rest-mongo</artifactId>

<version>0.0.1</version>

<properties>

<java.version>17</java.version>

<spring-boot.version>3.3.3</spring-boot.version>

</properties>

<dependencyManagement>

<dependencies>

<dependency>

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

<artifactId>spring-boot-dependencies</artifactId>

<version>${spring-boot.version}</version>

<type>pom</type><scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<!-- Spring Data MongoDB -->

<dependency>

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

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

</dependency>

<!-- Spring Data REST (auto-exports repositories as HAL endpoints) -->

<dependency>

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

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

</dependency>

<!-- Optional: validation + tests -->

<dependency>

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

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

**Gradle (Kotlin) alt**

dependencies {

implementation("org.springframework.boot:spring-boot-starter-data-mongodb")

implementation("org.springframework.boot:spring-boot-starter-data-rest")

implementation("org.springframework.boot:spring-boot-starter-validation")

testImplementation("org.springframework.boot:spring-boot-starter-test")

}

# 2) application.yml

spring:

application.name: sdr-mongo-demo

data:

mongodb:

uri: mongodb://localhost:27017/sdr\_demo

# Optional: expose repositories under a base path

spring.data.rest:

base-path: /api

default-page-size: 10

max-page-size: 100

server:

port: 8080

# 3) Domain models

## Product.java

package net.sivaacademy.sdr.product;

import jakarta.validation.constraints.Min;

import jakarta.validation.constraints.NotBlank;

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.index.Indexed;

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

@Document("products")

public class Product {

@Id

private String id;

@NotBlank @Indexed

private String name;

private String category;

@Min(0)

private double price;

private boolean active = true;

// getters/setters/ctors

public Product() {}

public Product(String name, String category, double price, boolean active) {

this.name = name; this.category = category; this.price = price; this.active = active;

}

// getters & setters omitted for brevity

}

## Customer.java

package net.sivaacademy.sdr.customer;

import jakarta.validation.constraints.Email;

import jakarta.validation.constraints.NotBlank;

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.index.Indexed;

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

@Document("customers")

public class Customer {

@Id

private String id;

@NotBlank

private String fullName;

@Email @Indexed(unique = true)

private String email;

private String phone;

// ctors/getters/setters

public Customer() {}

public Customer(String fullName, String email, String phone) {

this.fullName = fullName; this.email = email; this.phone = phone;

}

}

## Order.java

package net.sivaacademy.sdr.order;

import jakarta.validation.constraints.Min;

import jakarta.validation.constraints.NotBlank;

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.index.Indexed;

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

import java.time.Instant;

import java.util.List;

@Document("orders")

public class Order {

@Id

private String id;

@Indexed

private String customerId; // Reference by ID (no DBRef)

private List<OrderItem> items; // Embedded items

@Indexed

private String status; // NEW, PAID, SHIPPED, CANCELLED

private Instant createdAt = Instant.now();

private double total;

public Order() {}

public Order(String customerId, List<OrderItem> items, String status, double total) {

this.customerId = customerId; this.items = items; this.status = status; this.total = total;

}

public static class OrderItem {

@NotBlank

private String productId; // Reference by ID

@Min(1)

private int quantity;

@Min(0)

private double unitPrice;

public OrderItem() {}

public OrderItem(String productId, int quantity, double unitPrice) {

this.productId = productId; this.quantity = quantity; this.unitPrice = unitPrice;

}

// getters/setters

}

// getters/setters

}

# 4) Repositories (auto-exposed by Spring Data REST)

## ProductRepository.java

package net.sivaacademy.sdr.product;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.mongodb.repository.MongoRepository;

import org.springframework.data.rest.core.annotation.RepositoryRestResource;

import org.springframework.data.rest.core.annotation.RestResource;

@RepositoryRestResource(path = "products", collectionResourceRel = "products")

public interface ProductRepository extends MongoRepository<Product, String> {

// `GET /api/products/search/findByNameContainingIgnoreCase?name=lap`

@RestResource(path = "findByNameContainingIgnoreCase")

Page<Product> findByNameContainingIgnoreCase(String name, Pageable pageable);

// `GET /api/products/search/findByCategory?category=Electronics`

Page<Product> findByCategory(String category, Pageable pageable);

// `GET /api/products/search/findByActive?active=true`

Page<Product> findByActive(boolean active, Pageable pageable);

}

## CustomerRepository.java

package net.sivaacademy.sdr.customer;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.mongodb.repository.MongoRepository;

import org.springframework.data.rest.core.annotation.RepositoryRestResource;

@RepositoryRestResource(path = "customers", collectionResourceRel = "customers")

public interface CustomerRepository extends MongoRepository<Customer, String> {

// `GET /api/customers/search/findByEmail?email=foo@bar.com`

Customer findByEmail(String email);

// `GET /api/customers/search/findByFullNameContainingIgnoreCase?fullName=an`

Page<Customer> findByFullNameContainingIgnoreCase(String fullName, Pageable pageable);

}

## OrderRepository.java

package net.sivaacademy.sdr.order;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.mongodb.repository.MongoRepository;

import org.springframework.data.rest.core.annotation.RepositoryRestResource;

@RepositoryRestResource(path = "orders", collectionResourceRel = "orders")

public interface OrderRepository extends MongoRepository<Order, String> {

// `GET /api/orders/search/findByCustomerId?customerId=...`

Page<Order> findByCustomerId(String customerId, Pageable pageable);

// `GET /api/orders/search/findByStatus?status=PAID`

Page<Order> findByStatus(String status, Pageable pageable);

}

# 5) Projections (optional, for concise “excerpt” views)

## Projections.java

package net.sivaacademy.sdr.view;

import net.sivaacademy.sdr.product.Product;

import net.sivaacademy.sdr.customer.Customer;

import net.sivaacademy.sdr.order.Order;

import org.springframework.data.rest.core.config.Projection;

import java.time.Instant;

import java.util.List;

@Projection(name = "productExcerpt", types = Product.class)

public interface ProductExcerpt {

String getId();

String getName();

String getCategory();

double getPrice();

}

@Projection(name = "customerExcerpt", types = Customer.class)

public interface CustomerExcerpt {

String getId();

String getFullName();

String getEmail();

}

@Projection(name = "orderExcerpt", types = Order.class)

public interface OrderExcerpt {

String getId();

String getCustomerId();

String getStatus();

Instant getCreatedAt();

double getTotal();

List<Order.OrderItem> getItems();

}

Use with ?projection=productExcerpt etc.

# 6) App & Seed Data

## SdrMongoDemoApplication.java

package net.sivaacademy.sdr;

import net.sivaacademy.sdr.customer.Customer;

import net.sivaacademy.sdr.customer.CustomerRepository;

import net.sivaacademy.sdr.order.Order;

import net.sivaacademy.sdr.order.OrderRepository;

import net.sivaacademy.sdr.product.Product;

import net.sivaacademy.sdr.product.ProductRepository;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

import java.util.List;

@SpringBootApplication

public class SdrMongoDemoApplication {

public static void main(String[] args) {

SpringApplication.run(SdrMongoDemoApplication.class, args);

}

@Bean

CommandLineRunner seed(ProductRepository products, CustomerRepository customers, OrderRepository orders) {

return args -> {

if (products.count() == 0 && customers.count() == 0 && orders.count() == 0) {

// Products

var p1 = products.save(new Product("Laptop Air 14", "Electronics", 79999, true));

var p2 = products.save(new Product("NoiseCancel Pro Headphones", "Electronics", 12999, true));

var p3 = products.save(new Product("Ergo Chair", "Furniture", 10999, true));

// Customers

var c1 = customers.save(new Customer("Ananya Sharma", "ananya@sivaacademy.net", "9876543210"));

var c2 = customers.save(new Customer("Rohan Mehta", "rohan@sivaacademy.net", "9123456780"));

// Orders

orders.save(new Order(

c1.getId(),

List.of(new Order.OrderItem(p1.getId(), 1, p1.getPrice()),

new Order.OrderItem(p2.getId(), 1, p2.getPrice())),

"PAID",

p1.getPrice() + p2.getPrice()

));

orders.save(new Order(

c2.getId(),

List.of(new Order.OrderItem(p3.getId(), 2, p3.getPrice())),

"NEW",

2 \* p3.getPrice()

));

}

};

}

}

# 7) Run

mvn spring-boot:run

# 8) Explore the REST API (HAL/JSON)

Base path is /api (per application.yml). Spring Data REST auto-exposes CRUD with paging & sorting.

**Root & profile**

curl http://localhost:8080/api

curl http://localhost:8080/api/profile

**Products**

# List (paged)

curl "http://localhost:8080/api/products?page=0&size=5&sort=price,desc"

# Search by name contains (case-insensitive)

curl "http://localhost:8080/api/products/search/findByNameContainingIgnoreCase?name=lap"

# Projection

curl "http://localhost:8080/api/products?projection=productExcerpt"

# Create

curl -X POST http://localhost:8080/api/products \

-H "Content-Type: application/json" \

-d '{"name":"USB-C Dock","category":"Electronics","price":4999,"active":true}'

# Update (PATCH)

curl -X PATCH http://localhost:8080/api/products/{id} \

-H "Content-Type: application/json" \

-d '{"price":4599}'

**Customers**

# Find by email

curl "http://localhost:8080/api/customers/search/findByEmail?email=ananya@sivaacademy.net"

# Search by name contains

curl "http://localhost:8080/api/customers/search/findByFullNameContainingIgnoreCase?fullName=an"

# Create

curl -X POST http://localhost:8080/api/customers \

-H "Content-Type: application/json" \

-d '{"fullName":"Sana Kapoor","email":"sana@sivaacademy.net","phone":"9000000000"}'

**Orders**

# List orders (sorted by createdAt desc)

curl "http://localhost:8080/api/orders?sort=createdAt,desc"

# Search orders by customer

curl "http://localhost:8080/api/orders/search/findByCustomerId?customerId={customerId}"

# Search by status

curl "http://localhost:8080/api/orders/search/findByStatus?status=PAID"

# Create an order (use valid product/customer IDs)

curl -X POST http://localhost:8080/api/orders \

-H "Content-Type: application/json" \

-d '{

"customerId":"{customerId}",

"status":"NEW",

"total":14998,

"items":[{"productId":"{productId}","quantity":2,"unitPrice":7499}]

}'

## Notes & Options

* **No controllers required**—Spring Data REST publishes repository endpoints automatically.
* **Sorting & paging** are handled with ?page=...&size=...&sort=field,asc|desc.
* Prefer **ID references** between collections; avoid @DBRef unless you specifically need DBRef semantics.
* Add **indexes** with @Indexed or @CompoundIndexes in entities for production.
* Use **projections** (shown above) or **excerpt projections** to tailor responses.
* If you want **CORS** or to hide certain repository methods, we can add a RepositoryRestConfigurer.

**Here’s an “upgrade pack” to your Spring Data REST + Mongo project adding:**

* **Analytics (MongoDB aggregations)**
* **Soft delete (hide DELETE; add custom “soft-delete” endpoints)**
* **Input validation + better error messages**
* **CORS + repository exposure tweaks**
* **A small “API gateway” view that joins Orders with Product/Customer names**

**You can copy these files into the project you just created.**

# 1) Add MongoTemplate (already present via starter)

No pom.xml change needed—spring-boot-starter-data-mongodb already brings MongoTemplate.

# 2) Global config (CORS, base path, ID exposure)

## RestConfig.java

package net.sivaacademy.sdr.config;

import org.springframework.context.annotation.Configuration;

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

import org.springframework.data.rest.core.config.RepositoryRestConfiguration;

import org.springframework.data.rest.webmvc.config.RepositoryRestConfigurer;

import org.springframework.web.servlet.config.annotation.CorsRegistry;

@Configuration

public class RestConfig implements RepositoryRestConfigurer {

@Override

public void configureRepositoryRestConfiguration(RepositoryRestConfiguration config, CorsRegistry cors) {

config.setBasePath("/api"); // mirrors your application.yml

// Expose IDs automatically for all Mongo entities

config.exposeIdsFor(

net.sivaacademy.sdr.product.Product.class,

net.sivaacademy.sdr.customer.Customer.class,

net.sivaacademy.sdr.order.Order.class

);

cors.addMapping("/api/\*\*")

.allowedOrigins("http://localhost:3000", "http://127.0.0.1:3000")

.allowedMethods("GET","POST","PATCH","PUT","DELETE","OPTIONS")

.allowCredentials(true);

}

}

# 3) Soft delete

### 3.1 Add deletedAt fields

Update your entities to include a nullable timestamp. Example for **Product** (repeat similarly for Customer/Order):

// in Product.java

import java.time.Instant;

// ...

private Instant deletedAt; // null = not deleted

public Instant getDeletedAt() { return deletedAt; }

public void setDeletedAt(Instant deletedAt) { this.deletedAt = deletedAt; }

### 3.2 Hide repository DELETE endpoints

// ProductRepository.java

import org.springframework.data.rest.core.annotation.RestResource;

@RestResource(exported = true)

public interface ProductRepository extends MongoRepository<Product, String> {

// ...

@Override @RestResource(exported = false) void deleteById(String id);

@Override @RestResource(exported = false) void delete(Product entity);

@Override @RestResource(exported = false) void deleteAll(Iterable<? extends Product> entities);

@Override @RestResource(exported = false) void deleteAll();

}

(Do the same in CustomerRepository and OrderRepository.)

### 3.3 Only return non-deleted by default

Change your listing/search to **exclude** soft-deleted rows. For Spring Data REST derived methods, just add deletedAtIsNull to signatures:

// ProductRepository.java

Page<Product> findByDeletedAtIsNull(Pageable pageable);

Page<Product> findByNameContainingIgnoreCaseAndDeletedAtIsNull(String name, Pageable p);

Page<Product> findByCategoryAndDeletedAtIsNull(String category, Pageable p);

Page<Product> findByActiveAndDeletedAtIsNull(boolean active, Pageable p);

(Adjust controllers/clients to use these search paths. For the “default list”, Spring Data REST’s /api/products still exposes findAll(). If you want that to also exclude soft-deleted, create a **projection** or use a custom controller endpoint for “public list”.)

### 3.4 Custom soft-delete endpoints

## AdminController.java

package net.sivaacademy.sdr.admin;

import net.sivaacademy.sdr.customer.CustomerRepository;

import net.sivaacademy.sdr.order.OrderRepository;

import net.sivaacademy.sdr.product.ProductRepository;

import org.springframework.http.ResponseEntity;

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

import java.time.Instant;

@RestController

@RequestMapping("/api/admin")

public class AdminController {

private final ProductRepository products;

private final CustomerRepository customers;

private final OrderRepository orders;

public AdminController(ProductRepository p, CustomerRepository c, OrderRepository o) {

this.products = p; this.customers = c; this.orders = o;

}

@PatchMapping("/products/{id}/soft-delete")

public ResponseEntity<?> softDeleteProduct(@PathVariable String id) {

return products.findById(id).map(p -> {

p.setDeletedAt(Instant.now());

products.save(p);

return ResponseEntity.noContent().build();

}).orElse(ResponseEntity.notFound().build());

}

@PatchMapping("/customers/{id}/soft-delete")

public ResponseEntity<?> softDeleteCustomer(@PathVariable String id) {

return customers.findById(id).map(c -> {

c.setDeletedAt(Instant.now());

customers.save(c);

return ResponseEntity.noContent().build();

}).orElse(ResponseEntity.notFound().build());

}

@PatchMapping("/orders/{id}/soft-delete")

public ResponseEntity<?> softDeleteOrder(@PathVariable String id) {

return orders.findById(id).map(o -> {

o.setDeletedAt(Instant.now());

orders.save(o);

return ResponseEntity.noContent().build();

}).orElse(ResponseEntity.notFound().build());

}

}

# 4) Validation (clear messages)

You already have @NotBlank, @Min, @Email. Add a simple handler to make errors tidy:

## ValidationAdvice.java

package net.sivaacademy.sdr.config;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.validation.FieldError;

import org.springframework.web.bind.MethodArgumentNotValidException;

import org.springframework.web.bind.annotation.ControllerAdvice;

import org.springframework.web.bind.annotation.ExceptionHandler;

import java.util.LinkedHashMap;

import java.util.Map;

@ControllerAdvice

public class ValidationAdvice {

@ExceptionHandler(MethodArgumentNotValidException.class)

public ResponseEntity<?> handle(MethodArgumentNotValidException ex) {

Map<String, Object> body = new LinkedHashMap<>();

body.put("message", "Validation failed");

Map<String, String> errors = new LinkedHashMap<>();

for (FieldError fe : ex.getBindingResult().getFieldErrors()) {

errors.put(fe.getField(), fe.getDefaultMessage());

}

body.put("errors", errors);

return ResponseEntity.status(HttpStatus.BAD\_REQUEST).body(body);

}

}

(For Spring Data REST POST/PUT/PATCH on repositories, these messages will surface as a clean JSON payload.)

# 5) Analytics (MongoDB Aggregations)

Create read-only endpoints under /api/analytics/\*\*. We’ll compute:

* **Order totals by customer**
* **Top products by revenue** (sum of quantity \* unitPrice)

## AnalyticsController.java

package net.sivaacademy.sdr.analytics;

import net.sivaacademy.sdr.order.Order;

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

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

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

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

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

import java.util.List;

import java.util.Map;

import static org.springframework.data.mongodb.core.aggregation.Aggregation.\*;

@RestController

@RequestMapping("/api/analytics")

public class AnalyticsController {

private final MongoTemplate mongo;

public AnalyticsController(MongoTemplate mongo) { this.mongo = mongo; }

@GetMapping("/orders/total-by-customer")

public List<Map<String,Object>> orderTotalsByCustomer() {

var agg = newAggregation(

match(where("deletedAt").exists(false)), // exclude soft-deleted

group("customerId").sum("total").as("totalAmount"),

sort(org.springframework.data.domain.Sort.by("totalAmount").descending())

);

AggregationResults<Map> res = mongo.aggregate(agg, "orders", Map.class);

return (List) res.getMappedResults();

}

@GetMapping("/products/top-by-revenue")

public List<Map<String,Object>> topProductsByRevenue(@RequestParam(defaultValue = "10") int limit) {

var itemsField = "items";

var agg = newAggregation(

match(where("deletedAt").exists(false)),

unwind(itemsField),

project()

.andExpression("items.productId").as("productId")

.andExpression("items.quantity \* items.unitPrice").as("lineAmount"),

group("productId").sum("lineAmount").as("revenue"),

sort(org.springframework.data.domain.Sort.by("revenue").descending()),

limit(limit)

);

AggregationResults<Map> res = mongo.aggregate(agg, "orders", Map.class);

return (List) res.getMappedResults();

}

}

**Try:**

curl http://localhost:8080/api/analytics/orders/total-by-customer

curl "http://localhost:8080/api/products?projection=productExcerpt"

curl "http://localhost:8080/api/analytics/products/top-by-revenue?limit=5"

# 6) “API Gateway” view for Orders (joined details)

A convenience view that returns an order with **customer name/email** and **expanded product line items**.

## OrderViewController.java

package net.sivaacademy.sdr.view;

import net.sivaacademy.sdr.customer.CustomerRepository;

import net.sivaacademy.sdr.order.Order;

import net.sivaacademy.sdr.order.OrderRepository;

import net.sivaacademy.sdr.product.ProductRepository;

import org.springframework.http.ResponseEntity;

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

import java.util.\*;

@RestController

@RequestMapping("/api/orders/view")

public class OrderViewController {

private final OrderRepository orders;

private final CustomerRepository customers;

private final ProductRepository products;

public OrderViewController(OrderRepository o, CustomerRepository c, ProductRepository p) {

this.orders = o; this.customers = c; this.products = p;

}

@GetMapping("/{id}")

public ResponseEntity<?> view(@PathVariable String id) {

return orders.findById(id).map(order -> {

if (order.getDeletedAt() != null) return ResponseEntity.notFound().build();

var cust = customers.findById(order.getCustomerId()).orElse(null);

var itemsView = new ArrayList<Map<String,Object>>();

if (order.getItems() != null) {

for (var it : order.getItems()) {

var p = products.findById(it.getProductId()).orElse(null);

Map<String,Object> line = new LinkedHashMap<>();

line.put("productId", it.getProductId());

line.put("productName", p != null ? p.getName() : "(missing)");

line.put("quantity", it.getQuantity());

line.put("unitPrice", it.getUnitPrice());

line.put("lineTotal", it.getQuantity() \* it.getUnitPrice());

itemsView.add(line);

}

}

Map<String,Object> view = new LinkedHashMap<>();

view.put("orderId", order.getId());

view.put("status", order.getStatus());

view.put("createdAt", order.getCreatedAt());

view.put("total", order.getTotal());

view.put("customer", cust == null ? Map.of() : Map.of(

"id", cust.getId(),

"fullName", cust.getFullName(),

"email", cust.getEmail()

));

view.put("items", itemsView);

return ResponseEntity.ok(view);

}).orElse(ResponseEntity.notFound().build());

}

}

**Try:**

curl http://localhost:8080/api/orders

# take one order ID:

curl http://localhost:8080/api/orders/view/{orderId}

# 7) Helpful indexes

Add indexes for performance:

// Product.java

import org.springframework.data.mongodb.core.index.Indexed;

@Indexed private String name;

@Indexed private String category;

@Indexed private Boolean active;

@Indexed private java.time.Instant deletedAt;

// Customer.java

@Indexed(unique = true) private String email;

@Indexed private String fullName;

@Indexed private java.time.Instant deletedAt;

// Order.java

@Indexed private String customerId;

@Indexed private String status;

@Indexed private java.time.Instant createdAt;

@Indexed private java.time.Instant deletedAt;

# 8) Quick test script

# seed happens automatically on startup (from your previous runner)

# Soft delete a product

curl -X PATCH http://localhost:8080/api/admin/products/{productId}/soft-delete

# Confirm it's hidden by “non-deleted” search

curl "http://localhost:8080/api/products/search/findByNameContainingIgnoreCaseAndDeletedAtIsNull?name=laptop"

# Analytics

curl http://localhost:8080/api/analytics/orders/total-by-customer

curl "http://localhost:8080/api/analytics/products/top-by-revenue?limit=3"

# Joined order view

curl http://localhost:8080/api/orders/view/{orderId}

## That’s it!

You now have **Spring Data REST** for standard CRUD, plus:

* custom **analytics**,
* **soft deletes** with hidden DELETE,
* **clean validation errors**, **CORS**,
* and an **order view** endpoint that composes related documents.

**Adding the three extras:**

* **A) Analytics: avg order value per day & per ISO week**
* **B) Dynamic filtering (build Criteria from query params) for Products & Orders**
* **C) Full-text search (MongoDB text index) for Products & Customers**

**Drop these files into your existing Spring Data REST + Mongo project.**

# A) Analytics — Avg order value per day / week

## AnalyticsController.java (add endpoints)

// ... keep your existing imports + class

@GetMapping("/orders/avg-per-day")

public List<Map<String, Object>> avgOrderValuePerDay() {

var agg = newAggregation(

match(where("deletedAt").exists(false)),

project().andExpression("{$dateToString: {format: '%Y-%m-%d', date: '$createdAt'}}").as("day")

.and("total").as("total"),

group("day").avg("total").as("avg").sum("total").as("sum").count().as("count"),

sort(org.springframework.data.domain.Sort.by("day").ascending())

);

return (List) mongo.aggregate(agg, "orders", Map.class).getMappedResults();

}

@GetMapping("/orders/avg-per-week")

public List<Map<String, Object>> avgOrderValuePerWeek() {

var agg = newAggregation(

match(where("deletedAt").exists(false)),

project()

.andExpression("{$isoWeekYear: '$createdAt'}").as("year")

.andExpression("{$isoWeek: '$createdAt'}").as("week")

.and("total").as("total"),

group(fields().and("year").and("week")).avg("total").as("avg").sum("total").as("sum").count().as("count"),

sort(org.springframework.data.domain.Sort.by("year","week").ascending())

);

return (List) mongo.aggregate(agg, "orders", Map.class).getMappedResults();

}

**Try**

curl http://localhost:8080/api/analytics/orders/avg-per-day

curl http://localhost:8080/api/analytics/orders/avg-per-week

# B) Dynamic filtering (Criteria from query params)

## ProductSearchController.java

package net.sivaacademy.sdr.search;

import net.sivaacademy.sdr.product.Product;

import org.springframework.data.domain.\*;

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

import org.springframework.data.mongodb.core.query.\*;

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

import java.util.\*;

@RestController

@RequestMapping("/api/search/products")

public class ProductSearchController {

private final MongoTemplate mongo;

public ProductSearchController(MongoTemplate mongo) { this.mongo = mongo; }

@GetMapping

public Page<Product> search(

@RequestParam(required = false) String name,

@RequestParam(required = false) String category,

@RequestParam(required = false) Double minPrice,

@RequestParam(required = false) Double maxPrice,

@RequestParam(required = false) Boolean active,

@RequestParam(defaultValue = "true") boolean excludeDeleted,

@RequestParam(defaultValue = "0") int page,

@RequestParam(defaultValue = "10") int size,

@RequestParam(defaultValue = "name,asc") String sort

) {

List<Criteria> and = new ArrayList<>();

if (excludeDeleted) and.add(Criteria.where("deletedAt").is(null));

if (name != null && !name.isBlank()) and.add(Criteria.where("name").regex(name, "i"));

if (category != null && !category.isBlank()) and.add(Criteria.where("category").is(category));

if (minPrice != null) and.add(Criteria.where("price").gte(minPrice));

if (maxPrice != null) and.add(Criteria.where("price").lte(maxPrice));

if (active != null) and.add(Criteria.where("active").is(active));

Query q = new Query(and.isEmpty() ? new Criteria() : new Criteria().andOperator(and));

// sorting

String[] s = sort.split(",");

Sort by = ("desc".equalsIgnoreCase(s.length > 1 ? s[1] : "asc"))

? Sort.by(Sort.Order.desc(s[0])) : Sort.by(Sort.Order.asc(s[0]));

long total = mongo.count(q, Product.class);

q.with(PageRequest.of(page, size, by));

List<Product> content = mongo.find(q, Product.class);

return new PageImpl<>(content, PageRequest.of(page, size, by), total);

}

}

## OrderSearchController.java

package net.sivaacademy.sdr.search;

import net.sivaacademy.sdr.order.Order;

import org.springframework.data.domain.\*;

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

import org.springframework.data.mongodb.core.query.\*;

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

import java.time.Instant;

import java.util.\*;

@RestController

@RequestMapping("/api/search/orders")

public class OrderSearchController {

private final MongoTemplate mongo;

public OrderSearchController(MongoTemplate mongo) { this.mongo = mongo; }

@GetMapping

public Page<Order> search(

@RequestParam(required = false) String customerId,

@RequestParam(required = false) String status,

@RequestParam(required = false) Instant dateFrom,

@RequestParam(required = false) Instant dateTo,

@RequestParam(required = false) Double minTotal,

@RequestParam(required = false) Double maxTotal,

@RequestParam(defaultValue = "true") boolean excludeDeleted,

@RequestParam(defaultValue = "0") int page,

@RequestParam(defaultValue = "10") int size,

@RequestParam(defaultValue = "createdAt,desc") String sort

) {

List<Criteria> and = new ArrayList<>();

if (excludeDeleted) and.add(Criteria.where("deletedAt").is(null));

if (customerId != null && !customerId.isBlank()) and.add(Criteria.where("customerId").is(customerId));

if (status != null && !status.isBlank()) and.add(Criteria.where("status").is(status));

if (dateFrom != null) and.add(Criteria.where("createdAt").gte(dateFrom));

if (dateTo != null) and.add(Criteria.where("createdAt").lte(dateTo));

if (minTotal != null) and.add(Criteria.where("total").gte(minTotal));

if (maxTotal != null) and.add(Criteria.where("total").lte(maxTotal));

Query q = new Query(and.isEmpty() ? new Criteria() : new Criteria().andOperator(and));

String[] s = sort.split(",");

Sort by = ("desc".equalsIgnoreCase(s.length > 1 ? s[1] : "asc"))

? Sort.by(Sort.Order.desc(s[0])) : Sort.by(Sort.Order.asc(s[0]));

long total = mongo.count(q, Order.class);

q.with(PageRequest.of(page, size, by));

List<Order> content = mongo.find(q, Order.class);

return new PageImpl<>(content, PageRequest.of(page, size, by), total);

}

}

**Try**

# Products: category Electronics, price 3000..50000, active, sorted by price desc

curl "http://localhost:8080/api/search/products?category=Electronics&minPrice=3000&maxPrice=50000&active=true&sort=price,desc"

# Orders: for a customer in July–Aug 2024, totals >= 10k

curl "http://localhost:8080/api/search/orders?customerId={cid}&dateFrom=2024-07-01T00:00:00Z&dateTo=2024-08-31T23:59:59Z&minTotal=10000"

# C) Full-text search (MongoDB text index)

## 1) Add text annotations to entities

### Product.java (add)

import org.springframework.data.mongodb.core.index.TextIndexed;

import org.springframework.data.mongodb.core.index.TextScore;

// ...

@TextIndexed private String name;

@TextIndexed private String category;

private transient @TextScore Float textScore; // returned score (not persisted)

public Float getTextScore() { return textScore; }

public void setTextScore(Float textScore) { this.textScore = textScore; }

### Customer.java (add)

import org.springframework.data.mongodb.core.index.TextIndexed;

import org.springframework.data.mongodb.core.index.TextScore;

// ...

@TextIndexed private String fullName;

@TextIndexed private String email;

private transient @TextScore Float textScore;

public Float getTextScore() { return textScore; }

public void setTextScore(Float textScore) { this.textScore = textScore; }

Alternatively, you can build compound text indexes at startup via MongoTemplate.indexOps(Product.class).ensureIndex(...). The annotations are simplest.

## 2) Text search endpoints

### TextSearchController.java

package net.sivaacademy.sdr.search;

import net.sivaacademy.sdr.product.Product;

import net.sivaacademy.sdr.customer.Customer;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageImpl;

import org.springframework.data.domain.PageRequest;

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

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

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

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

import java.util.List;

@RestController

@RequestMapping("/api/text")

public class TextSearchController {

private final MongoTemplate mongo;

public TextSearchController(MongoTemplate mongo) { this.mongo = mongo; }

@GetMapping("/products")

public Page<Product> searchProducts(@RequestParam String q,

@RequestParam(defaultValue = "0") int page,

@RequestParam(defaultValue = "10") int size) {

var criteria = TextCriteria.forDefaultLanguage().matching(q);

var query = TextQuery.queryText(criteria)

.sortByScore()

.skip((long) page \* size)

.limit(size);

// exclude soft-deleted

query.addCriteria(org.springframework.data.mongodb.core.query.Criteria.where("deletedAt").is(null));

List<Product> list = mongo.find(query, Product.class);

long total = mongo.count(TextQuery.queryText(criteria), Product.class);

return new PageImpl<>(list, PageRequest.of(page, size), total);

}

@GetMapping("/customers")

public Page<Customer> searchCustomers(@RequestParam String q,

@RequestParam(defaultValue = "0") int page,

@RequestParam(defaultValue = "10") int size) {

var criteria = TextCriteria.forDefaultLanguage().matching(q);

var query = TextQuery.queryText(criteria).sortByScore()

.skip((long) page \* size).limit(size);

query.addCriteria(org.springframework.data.mongodb.core.query.Criteria.where("deletedAt").is(null));

List<Customer> list = mongo.find(query, Customer.class);

long total = mongo.count(TextQuery.queryText(criteria), Customer.class);

return new PageImpl<>(list, PageRequest.of(page, size), total);

}

}

**Try**

curl "http://localhost:8080/api/text/products?q=laptop electronics"

curl "http://localhost:8080/api/text/customers?q=ananya sivaacademy"

## Optional: ensure compound text index at startup

If you prefer explicit index creation:

// In your main application runner or a @Configuration class

@Bean

CommandLineRunner ensureTextIndexes(MongoTemplate mongo) {

return args -> {

mongo.indexOps(Product.class).ensureIndex(

new org.springframework.data.mongodb.core.index.TextIndexDefinition.TextIndexDefinitionBuilder()

.onField("name").onField("category").build()

);

mongo.indexOps(Customer.class).ensureIndex(

new org.springframework.data.mongodb.core.index.TextIndexDefinition.TextIndexDefinitionBuilder()

.onField("fullName").onField("email").build()

);

};

}

## That’s all set

You now have:

* /api/analytics/orders/avg-per-day and /avg-per-week
* /api/search/products and /api/search/orders for **dynamic filtering**
* /api/text/products and /api/text/customers for **full-text** search with relevance scores

**Here’s the add-on pack:**

* **(1) CSV/Excel export for Products & Orders**
* **(2) Spec-style dynamic filters like price>5000;category==Electronics;active==true**
* **(3) Async analytics jobs (fire-and-forget + poll status)**

**All snippets drop into your existing Spring Data REST + Mongo project.**

# 1) CSV / Excel export

## 1.1 pom.xml (Excel optional)

<!-- CSV uses core Java (no dep). For Excel, add: -->

<dependency>

<groupId>org.apache.poi</groupId>

<artifactId>poi-ooxml</artifactId>

<version>5.2.5</version>

</dependency>

## 1.2 ExportController.java

package net.sivaacademy.sdr.export;

import jakarta.servlet.http.HttpServletResponse;

import net.sivaacademy.sdr.order.Order;

import net.sivaacademy.sdr.product.Product;

import org.apache.poi.ss.usermodel.\*;

import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import org.springframework.data.domain.Sort;

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

import org.springframework.data.mongodb.core.query.\*;

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

import java.io.PrintWriter;

import java.net.URLEncoder;

import java.nio.charset.StandardCharsets;

import java.time.format.DateTimeFormatter;

import java.util.List;

@RestController

@RequestMapping("/api/export")

public class ExportController {

private final MongoTemplate mongo;

public ExportController(MongoTemplate mongo) { this.mongo = mongo; }

// -------- CSV (Products) --------

@GetMapping("/products.csv")

public void productsCsv(

@RequestParam(defaultValue = "true") boolean excludeDeleted,

@RequestParam(defaultValue = "name,asc") String sort,

HttpServletResponse res

) throws Exception {

Query q = new Query();

if (excludeDeleted) q.addCriteria(Criteria.where("deletedAt").is(null));

String[] s = sort.split(",");

Sort by = "desc".equalsIgnoreCase(s.length>1?s[1]:"asc")

? Sort.by(Sort.Order.desc(s[0])) : Sort.by(Sort.Order.asc(s[0]));

q.with(by);

List<Product> data = mongo.find(q, Product.class);

res.setContentType("text/csv; charset=UTF-8");

res.setHeader("Content-Disposition",

"attachment; filename=\"products.csv\"");

try (PrintWriter w = res.getWriter()) {

w.println("id,name,category,price,active");

for (var p : data) {

w.printf("%s,%s,%s,%.2f,%s%n",

safe(p.getId()), safe(p.getName()), safe(p.getCategory()),

p.getPrice(), p.isActive());

}

}

}

// -------- CSV (Orders) --------

@GetMapping("/orders.csv")

public void ordersCsv(

@RequestParam(defaultValue = "true") boolean excludeDeleted,

@RequestParam(defaultValue = "createdAt,desc") String sort,

HttpServletResponse res

) throws Exception {

Query q = new Query();

if (excludeDeleted) q.addCriteria(Criteria.where("deletedAt").is(null));

String[] s = sort.split(",");

Sort by = "desc".equalsIgnoreCase(s.length>1?s[1]:"asc")

? Sort.by(Sort.Order.desc(s[0])) : Sort.by(Sort.Order.asc(s[0]));

q.with(by);

List<Order> data = mongo.find(q, Order.class);

DateTimeFormatter ISO = DateTimeFormatter.ISO\_INSTANT;

res.setContentType("text/csv; charset=UTF-8");

res.setHeader("Content-Disposition","attachment; filename=\"orders.csv\"");

try (PrintWriter w = res.getWriter()) {

w.println("id,customerId,status,createdAt,total,itemsCount");

for (var o : data) {

int items = o.getItems()==null?0:o.getItems().size();

w.printf("%s,%s,%s,%s,%.2f,%d%n",

safe(o.getId()), safe(o.getCustomerId()), safe(o.getStatus()),

o.getCreatedAt()!=null?ISO.format(o.getCreatedAt()):"",

o.getTotal(), items);

}

}

}

// -------- Excel (Products) --------

@GetMapping("/products.xlsx")

public void productsXlsx(

@RequestParam(defaultValue = "true") boolean excludeDeleted,

HttpServletResponse res

) throws Exception {

Query q = new Query();

if (excludeDeleted) q.addCriteria(Criteria.where("deletedAt").is(null));

List<Product> data = mongo.find(q, Product.class);

try (Workbook wb = new XSSFWorkbook()) {

Sheet sh = wb.createSheet("Products");

int r=0;

Row h = sh.createRow(r++);

h.createCell(0).setCellValue("ID");

h.createCell(1).setCellValue("Name");

h.createCell(2).setCellValue("Category");

h.createCell(3).setCellValue("Price");

h.createCell(4).setCellValue("Active");

for (var p: data) {

Row row = sh.createRow(r++);

row.createCell(0).setCellValue(nz(p.getId()));

row.createCell(1).setCellValue(nz(p.getName()));

row.createCell(2).setCellValue(nz(p.getCategory()));

row.createCell(3).setCellValue(p.getPrice());

row.createCell(4).setCellValue(p.isActive());

}

String fname = URLEncoder.encode("products.xlsx", StandardCharsets.UTF\_8);

res.setHeader("Content-Disposition","attachment; filename\*=UTF-8''"+fname);

res.setContentType("application/vnd.openxmlformats-officedocument.spreadsheetml.sheet");

wb.write(res.getOutputStream());

}

}

private static String safe(String s){

if(s==null) return "";

// very light CSV escaping

if(s.contains(",")||s.contains("\"")||s.contains("\n")){

return "\""+s.replace("\"","\"\"")+"\"";

}

return s;

}

private static String nz(String s){ return s==null?"":s; }

}

# 2) Spec-style dynamic filters

**Grammar (simple):**

* Clauses separated by ;
* Operators: ==, !=, >, >=, <, <=, ~ (contains regex, case-insensitive), in(...)
* Fields: any document field (nested with dot e.g., total>=10000;status==PAID)

## 2.1 SpecFilterParser.java

package net.sivaacademy.sdr.search;

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

import java.util.\*;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class SpecFilterParser {

private static final Pattern IN = Pattern.compile("^(.+?)\\s\*in\\((.+)\\)$");

public static Criteria parse(String spec) {

if (spec == null || spec.isBlank()) return new Criteria();

List<Criteria> and = new ArrayList<>();

for (String c : spec.split(";")) {

String clause = c.trim();

if (clause.isEmpty()) continue;

Matcher m = IN.matcher(clause);

if (m.find()) {

String field = m.group(1).trim();

String[] vals = Arrays.stream(m.group(2).split(","))

.map(String::trim).toArray(String[]::new);

and.add(Criteria.where(field).in((Object[])vals));

continue;

}

String op = findOp(clause);

String[] parts = clause.split(Pattern.quote(op), 2);

String field = parts[0].trim();

String raw = parts[1].trim();

Object val = coerce(raw);

switch (op) {

case "==": and.add(Criteria.where(field).is(val)); break;

case "!=": and.add(Criteria.where(field).ne(val)); break;

case ">": and.add(Criteria.where(field).gt(val)); break;

case ">=": and.add(Criteria.where(field).gte(val)); break;

case "<": and.add(Criteria.where(field).lt(val)); break;

case "<=": and.add(Criteria.where(field).lte(val)); break;

case "~": and.add(Criteria.where(field).regex(val.toString(), "i")); break;

default: throw new IllegalArgumentException("Unknown op: "+op);

}

}

return and.isEmpty()? new Criteria(): new Criteria().andOperator(and);

}

private static String findOp(String s){

for (String op : new String[]{"==","!=",">=","<=","~",">","<"}){

int i = s.indexOf(op);

if (i>0) return op;

}

throw new IllegalArgumentException("No operator in '"+s+"'");

}

private static Object coerce(String raw){

if (raw.equalsIgnoreCase("true")||raw.equalsIgnoreCase("false")) return Boolean.valueOf(raw);

try { if (raw.contains(".")) return Double.valueOf(raw); return Long.valueOf(raw); }

catch(Exception ignore){}

if ((raw.startsWith("'")&&raw.endsWith("'"))||(raw.startsWith("\"")&&raw.endsWith("\"")))

return raw.substring(1, raw.length()-1);

return raw;

}

}

## 2.2 Generic endpoints using the spec

### SpecSearchController.java

package net.sivaacademy.sdr.search;

import net.sivaacademy.sdr.order.Order;

import net.sivaacademy.sdr.product.Product;

import org.springframework.data.domain.\*;

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

import org.springframework.data.mongodb.core.query.\*;

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

import java.util.List;

@RestController

@RequestMapping("/api/spec")

public class SpecSearchController {

private final MongoTemplate mongo;

public SpecSearchController(MongoTemplate mongo){ this.mongo = mongo; }

@GetMapping("/products")

public Page<Product> products(@RequestParam String q,

@RequestParam(defaultValue="0") int page,

@RequestParam(defaultValue="10") int size,

@RequestParam(defaultValue="name,asc") String sort){

Criteria c = SpecFilterParser.parse(q);

Query query = new Query(c);

String[] s = sort.split(",");

Sort by = "desc".equalsIgnoreCase(s.length>1?s[1]:"asc")

? Sort.by(Sort.Order.desc(s[0])) : Sort.by(Sort.Order.asc(s[0]));

long total = mongo.count(query, Product.class);

query.with(PageRequest.of(page,size,by));

List<Product> list = mongo.find(query, Product.class);

return new PageImpl<>(list, PageRequest.of(page,size,by), total);

}

@GetMapping("/orders")

public Page<Order> orders(@RequestParam String q,

@RequestParam(defaultValue="0") int page,

@RequestParam(defaultValue="10") int size,

@RequestParam(defaultValue="createdAt,desc") String sort){

Criteria c = SpecFilterParser.parse(q);

Query query = new Query(c);

String[] s = sort.split(",");

Sort by = "desc".equalsIgnoreCase(s.length>1?s[1]:"asc")

? Sort.by(Sort.Order.desc(s[0])) : Sort.by(Sort.Order.asc(s[0]));

long total = mongo.count(query, Order.class);

query.with(PageRequest.of(page,size,by));

List<Order> list = mongo.find(query, Order.class);

return new PageImpl<>(list, PageRequest.of(page,size,by), total);

}

}

**Examples**

# Products active in Electronics with price 3000..50000 and name contains "lap"

curl "http://localhost:8080/api/spec/products?q=active==true;category==Electronics;price>=3000;price<=50000;name~lap"

# Orders: PAID or SHIPPED with total>10000

curl "http://localhost:8080/api/spec/orders?q=status in(PAID,SHIPPED);total>10000"

# 3) Async analytics jobs

* Fire a job (e.g., recompute “top products by revenue”).
* Poll status/result later.
* Backed by a Mongo collection analytics\_jobs.

## 3.1 Enable async

### AsyncConfig.java

package net.sivaacademy.sdr.config;

import org.springframework.context.annotation.Configuration;

import org.springframework.scheduling.annotation.EnableAsync;

@Configuration

@EnableAsync

public class AsyncConfig {}

## 3.2 Job model & repo

### AnalyticsJob.java

package net.sivaacademy.sdr.analytics;

import org.springframework.data.annotation.Id;

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

import java.time.Instant;

import java.util.Map;

@Document("analytics\_jobs")

public class AnalyticsJob {

@Id private String id;

private String type; // e.g., TOP\_PRODUCTS, TOTALS\_BY\_CUSTOMER

private String status; // QUEUED, RUNNING, DONE, ERROR

private Instant createdAt = Instant.now();

private Instant finishedAt;

private Map<String,Object> params; // input

private Object result; // aggregation result or error

// getters/setters...

}

### AnalyticsJobRepository.java

package net.sivaacademy.sdr.analytics;

import org.springframework.data.mongodb.repository.MongoRepository;

public interface AnalyticsJobRepository extends MongoRepository<AnalyticsJob, String> {}

## 3.3 Service that runs jobs

### AnalyticsJobService.java

package net.sivaacademy.sdr.analytics;

import org.springframework.data.domain.Sort;

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

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

import org.springframework.scheduling.annotation.Async;

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Map;

import static org.springframework.data.mongodb.core.aggregation.Aggregation.\*;

@Service

public class AnalyticsJobService {

private final MongoTemplate mongo;

private final AnalyticsJobRepository repo;

public AnalyticsJobService(MongoTemplate mongo, AnalyticsJobRepository repo) {

this.mongo = mongo; this.repo = repo;

}

@Async

public void run(String jobId) {

var job = repo.findById(jobId).orElseThrow();

try {

job.setStatus("RUNNING"); repo.save(job);

Object res;

switch (job.getType()) {

case "TOP\_PRODUCTS" -> res = topProducts((Integer) job.getParams().getOrDefault("limit", 10));

case "TOTALS\_BY\_CUSTOMER" -> res = totalsByCustomer();

default -> throw new IllegalArgumentException("Unknown job type");

}

job.setResult(res);

job.setStatus("DONE");

job.setFinishedAt(java.time.Instant.now());

repo.save(job);

} catch (Exception e) {

job.setStatus("ERROR");

job.setResult(Map.of("error", e.getMessage()));

job.setFinishedAt(java.time.Instant.now());

repo.save(job);

}

}

private List<Map> topProducts(int limit) {

var agg = newAggregation(

match(where("deletedAt").exists(false)),

unwind("items"),

project()

.andExpression("items.productId").as("productId")

.andExpression("items.quantity \* items.unitPrice").as("lineAmount"),

group("productId").sum("lineAmount").as("revenue"),

sort(Sort.by(Sort.Direction.DESC,"revenue")),

limit(limit)

);

return mongo.aggregate(agg, "orders", Map.class).getMappedResults();

}

private List<Map> totalsByCustomer() {

var agg = newAggregation(

match(where("deletedAt").exists(false)),

group("customerId").sum("total").as("totalAmount"),

sort(Sort.by(Sort.Direction.DESC,"totalAmount"))

);

return mongo.aggregate(agg, "orders", Map.class).getMappedResults();

}

}

## 3.4 Controller to create + poll jobs

### AnalyticsJobController.java

package net.sivaacademy.sdr.analytics;

import org.springframework.http.ResponseEntity;

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

import java.util.Map;

@RestController

@RequestMapping("/api/analytics/jobs")

public class AnalyticsJobController {

private final AnalyticsJobRepository repo;

private final AnalyticsJobService svc;

public AnalyticsJobController(AnalyticsJobRepository repo, AnalyticsJobService svc) {

this.repo = repo; this.svc = svc;

}

@PostMapping("/{type}")

public ResponseEntity<?> create(@PathVariable String type, @RequestBody(required = false) Map<String,Object> params) {

var job = new AnalyticsJob();

job.setType(type.toUpperCase());

job.setStatus("QUEUED");

job.setParams(params==null? Map.of(): params);

job = repo.save(job);

svc.run(job.getId()); // fire & forget

return ResponseEntity.accepted().body(Map.of("jobId", job.getId(), "status", job.getStatus()));

// client polls GET /api/analytics/jobs/{id}

}

@GetMapping("/{id}")

public ResponseEntity<?> get(@PathVariable String id) {

return repo.findById(id)

.<ResponseEntity<?>>map(j -> ResponseEntity.ok(Map.of(

"id", j.getId(),

"type", j.getType(),

"status", j.getStatus(),

"createdAt", j.getCreatedAt(),

"finishedAt", j.getFinishedAt(),

"result", j.getResult()

)))

.orElse(ResponseEntity.notFound().build());

}

}

**Try**

# Start a job

curl -X POST http://localhost:8080/api/analytics/jobs/TOP\_PRODUCTS -H "Content-Type: application/json" -d '{"limit":5}'

# Poll the returned jobId until status == DONE

curl http://localhost:8080/api/analytics/jobs/{jobId}

## You’re set

You now have:

* Downloadable **CSV** and optional **Excel** exports
* Powerful **spec filters** via /api/spec/...
* **Async analytics** jobs with status polling

**Here’s the next pack:**

* **Security: API-Key (for admins) + JWT (for users) with Spring Security 6 / Boot 3.3**
* **Rate limits: Bucket4j on the download/export endpoints**
* **React admin UI: a tiny panel to soft-delete, kick analytics jobs, poll job status, and download exports**

**Copy/paste these into your existing project.**

# 1) Add dependencies

## pom.xml

<dependencies>

<!-- ... your existing deps ... -->

<!-- Security -->

<dependency>

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

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

</dependency>

<!-- JWT validation (Resource Server style) -->

<dependency>

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

<artifactId>spring-boot-starter-oauth2-resource-server</artifactId>

</dependency>

<!-- Rate limiting -->

<dependency>

<groupId>com.github.vladimir-bukhtoyarov</groupId>

<artifactId>bucket4j-core</artifactId>

<version>8.10.1</version>

</dependency>

</dependencies>

# 2) Configuration

## application.yml (add/adjust)

siva:

security:

admin-api-key: CHANGE\_ME\_SUPER\_SECRET # used for X-API-KEY auth on admin-only routes

spring:

security:

oauth2:

resourceserver:

jwt:

# HS256 shared secret for demo. Put in env var in prod.

secret: "REALLY\_LONG\_RANDOM\_256bit\_SECRET\_AT\_LEAST\_32\_CHARS"

server:

port: 8080

**How auth works**

* **Admins:** send header X-API-KEY: <value> → gets role ADMIN (and is treated as authenticated).
* **Users/services:** send Authorization: Bearer <JWT> (HS256) → gets roles from scope/authorities claims.
* You can use **both** in one app. Admin routes require ROLE\_ADMIN.

# 3) Security config

## SecurityConfig.java

package net.sivaacademy.sdr.config;

import jakarta.servlet.Filter;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.http.HttpMethod;

import org.springframework.security.authentication.AbstractAuthenticationToken;

import org.springframework.security.config.Customizer;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.core.authority.SimpleGrantedAuthority;

import org.springframework.security.web.SecurityFilterChain;

import org.springframework.security.web.authentication.preauth.AbstractPreAuthenticatedProcessingFilter;

import java.util.List;

@Configuration

public class SecurityConfig {

@Bean

SecurityFilterChain api(HttpSecurity http,

@Value("${siva.security.admin-api-key}") String adminApiKey,

Filter apiKeyFilter) throws Exception {

http

.csrf(csrf -> csrf.disable())

.authorizeHttpRequests(auth -> auth

// public: HAL root and profile

.requestMatchers("/api", "/api/profile/\*\*").permitAll()

// exports allowed to authenticated users (JWT or API key)

.requestMatchers("/api/export/\*\*").hasAnyRole("USER","ADMIN")

// admin-only

.requestMatchers("/api/admin/\*\*").hasRole("ADMIN")

.requestMatchers("/api/analytics/jobs/\*\*").hasRole("ADMIN")

// the “view” and “analytics read-only” can be user or admin

.requestMatchers("/api/orders/view/\*\*", "/api/analytics/\*\*").hasAnyRole("USER","ADMIN")

// the repository CRUD endpoints: require at least USER for writes

.requestMatchers(HttpMethod.GET, "/api/\*\*").permitAll() // browse catalog if you want it open

.requestMatchers(HttpMethod.POST, "/api/\*\*").hasAnyRole("USER","ADMIN")

.requestMatchers(HttpMethod.PATCH, "/api/\*\*").hasAnyRole("USER","ADMIN")

.requestMatchers(HttpMethod.PUT, "/api/\*\*").hasAnyRole("USER","ADMIN")

.requestMatchers(HttpMethod.DELETE, "/api/\*\*").hasRole("ADMIN")

.anyRequest().authenticated()

)

// add API-Key filter before standard auth

.addFilterBefore(apiKeyFilter, AbstractPreAuthenticatedProcessingFilter.class)

// enable JWT (resource server)

.oauth2ResourceServer(oauth -> oauth.jwt(Customizer.withDefaults()));

return http.build();

}

@Bean

public Filter apiKeyFilter(@Value("${siva.security.admin-api-key}") String adminApiKey) {

return (request, response, chain) -> {

String key = request.getHeader("X-API-KEY");

if (key != null && !key.isBlank() && key.equals(adminApiKey)) {

var auth = new ApiKeyAuthenticationToken("admin", List.of(

new SimpleGrantedAuthority("ROLE\_ADMIN"),

new SimpleGrantedAuthority("ROLE\_USER")

));

// set into context

var ctx = org.springframework.security.core.context.SecurityContextHolder.createEmptyContext();

ctx.setAuthentication(auth);

org.springframework.security.core.context.SecurityContextHolder.setContext(ctx);

}

chain.doFilter(request, response);

};

}

static class ApiKeyAuthenticationToken extends AbstractAuthenticationToken {

private final String principal;

ApiKeyAuthenticationToken(String principal, List<SimpleGrantedAuthority> authorities) {

super(authorities);

this.principal = principal;

setAuthenticated(true);

}

@Override public Object getCredentials() { return ""; }

@Override public Object getPrincipal() { return principal; }

}

}

To generate **JWTs** for testing, you can use any HS256 tool with the secret above and {"sub":"user1","scope":"ROLE\_USER"} or {"authorities":["ROLE\_USER"]}.

# 4) Rate limiting for exports

## RateLimitConfig.java

package net.sivaacademy.sdr.config;

import com.bucket4j.\*;

import jakarta.servlet.\*;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import java.io.IOException;

import java.time.Duration;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

@Configuration

public class RateLimitConfig {

private final Map<String, Bucket> cache = new ConcurrentHashMap<>();

@Bean

public Filter exportRateLimiter() {

// 20 downloads per minute per client IP (burst 20)

Refill refill = Refill.greedy(20, Duration.ofMinutes(1));

Bandwidth limit = Bandwidth.classic(20, refill);

return new Filter() {

@Override public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)

throws IOException, ServletException {

HttpServletRequest request = (HttpServletRequest) req;

HttpServletResponse response = (HttpServletResponse) res;

if (request.getRequestURI().startsWith("/api/export/")) {

String key = clientKey(request);

Bucket bucket = cache.computeIfAbsent(key, k -> Bucket4j.builder().addLimit(limit).build());

if (!bucket.tryConsume(1)) {

response.setStatus(429);

response.setContentType("application/json");

response.getWriter().write("{\"message\":\"Rate limit exceeded. Try again later.\"}");

return;

}

}

chain.doFilter(req, res);

}

private String clientKey(HttpServletRequest req) {

String xf = req.getHeader("X-Forwarded-For");

return xf != null ? xf.split(",")[0].trim() : req.getRemoteAddr();

}

};

}

}

Spring Boot auto-registers the filter bean. If you want a different policy for admins, you can check authorities from the security context and skip limiting.

# 5) Minimal React Admin UI (single file)

This is a tiny component you can drop into any React app (Vite/CRA). It lets you:

* set **API Key** or **JWT** headers
* soft-delete a product
* start a **TOP\_PRODUCTS** job and poll its status
* download **products.csv**

Uses plain fetch + minimal styling. Replace BASE if your backend host differs.

// AdminPanel.jsx

import React, { useState } from "react";

const BASE = "http://localhost:8080";

export default function AdminPanel() {

const [apiKey, setApiKey] = useState("");

const [jwt, setJwt] = useState("");

const [productId, setProductId] = useState("");

const [jobId, setJobId] = useState("");

const [jobJson, setJobJson] = useState("");

const headers = () => {

const h = { "Content-Type": "application/json" };

if (apiKey) h["X-API-KEY"] = apiKey;

if (jwt) h["Authorization"] = `Bearer ${jwt}`;

return h;

};

async function softDeleteProduct() {

const r = await fetch(`${BASE}/api/admin/products/${productId}/soft-delete`, {

method: "PATCH", headers: headers(),

});

alert(r.ok ? "Soft-deleted" : `Failed (${r.status})`);

}

async function startTopProductsJob() {

const r = await fetch(`${BASE}/api/analytics/jobs/TOP\_PRODUCTS`, {

method: "POST", headers: headers(), body: JSON.stringify({ limit: 5 }),

});

const j = await r.json();

setJobId(j.jobId || "");

setJobJson(JSON.stringify(j, null, 2));

}

async function pollJob() {

if (!jobId) return;

const r = await fetch(`${BASE}/api/analytics/jobs/${jobId}`, { headers: headers() });

const j = await r.json();

setJobJson(JSON.stringify(j, null, 2));

}

function downloadProductsCsv() {

fetch(`${BASE}/api/export/products.csv`, { headers: headers() })

.then(r => r.blob())

.then(b => {

const url = window.URL.createObjectURL(b);

const a = document.createElement("a");

a.href = url;

a.download = "products.csv";

a.click();

URL.revokeObjectURL(url);

});

}

return (

<div style={{maxWidth: 720, margin: "2rem auto", fontFamily: "system-ui"}}>

<h2>Admin Panel</h2>

<div style={{display:"grid", gap:12}}>

<label>API Key (admin) <input value={apiKey} onChange={e=>setApiKey(e.target.value)} style={{width:"100%"}} /></label>

<label>JWT (user/admin) <input value={jwt} onChange={e=>setJwt(e.target.value)} style={{width:"100%"}} /></label>

</div>

<hr style={{margin:"20px 0"}}/>

<h3>Soft delete product</h3>

<input placeholder="Product ID" value={productId} onChange={e=>setProductId(e.target.value)} />

<button onClick={softDeleteProduct} style={{marginLeft:8}}>Soft Delete</button>

<hr style={{margin:"20px 0"}}/>

<h3>Analytics Jobs</h3>

<button onClick={startTopProductsJob}>Start TOP\_PRODUCTS (limit 5)</button>

<button onClick={pollJob} style={{marginLeft:8}}>Poll Job</button>

<pre style={{background:"#f7f7f7", padding:12, marginTop:12, overflowX:"auto"}}>{jobJson}</pre>

<hr style={{margin:"20px 0"}}/>

<h3>Exports</h3>

<button onClick={downloadProductsCsv}>Download products.csv</button>

</div>

);

}

If you prefer a production-ready UI, I can scaffold a **shadcn/ui + Tailwind** dashboard with auth header management and tables.

# 6) Quick test checklist

1. **Start app**: mvn spring-boot:run
2. **Call a public GET** (optional):
   * curl http://localhost:8080/api/products
3. **Try admin with API key**:
   * curl -X PATCH -H "X-API-KEY: CHANGE\_ME\_SUPER\_SECRET" http://localhost:8080/api/admin/products/{id}/soft-delete -i
4. **Try export with JWT** (or API key):
   * curl -H "Authorization: Bearer <JWT\_WITH\_ROLE\_USER>" -OJ http://localhost:8080/api/export/products.csv
5. **Hit rate limit** by calling export >20 times/min → expect **429**.

## Notes / Hardening

* Put secrets in **env vars** or a **vault**.
* Replace HS256 with **RSA** (JWK/JWKS) if you already run an IdP (Keycloak/Okta).
* For per-user rate limits, use a key: userId (from JWT) instead of IP.
* Add @Order(…) to filters if you need stricter ordering with others.