**🔹 Two ways to work with MongoDB in Spring Data**

Spring Data gives you **two main approaches** to talk to MongoDB:

1. **MongoRepository (Recommended)**
   * This is the easy and common way.
   * You don’t have to write queries for basic tasks (insert, read, update, delete).
   * Spring does the heavy lifting and gives you ready-made methods like save(), findById(), findAll(), deleteById(), etc.
2. **MongoTemplate**
   * This gives you more control.
   * You write queries yourself using MongoDB syntax.
   * Used when you need complex queries or custom behavior that MongoRepository cannot provide easily.

Think of it like this:  
👉 MongoRepository = automatic, less code, faster setup.  
👉 MongoTemplate = manual, more code, but more flexibility.

**🔹 Repository Hierarchy in Spring Data**

Here’s how the **interfaces are built step by step** (like a ladder):

Repository (Interface)

↑ extends

CrudRepository (Interface)

↑ extends

PagingAndSortingRepository (Interface)

↑ extends

MongoRepository (Interface)

* **Repository** → The most basic interface, a marker (doesn’t have methods).
* **CrudRepository** → Adds simple methods: save(), findById(), findAll(), delete().
* **PagingAndSortingRepository** → Adds methods to sort results and fetch in pages.
* **MongoRepository** → Adds MongoDB-specific features on top of all the above.

👉 This means when you use MongoRepository, you automatically get all methods from the other three.

# 📘 Notes on MongoRepository Example (Spring Boot + MongoDB)

## 1. **Project Setup**

* We used **Spring Boot Starter Project** with these dependencies:
  + spring-boot-starter-data-mongodb → to connect with MongoDB.
  + spring-boot-starter-web → optional, for REST APIs if needed.
  + lombok → to avoid writing boilerplate code (getters, setters, constructors).
* application.properties contains MongoDB connection details:
* spring.application.name=MongoDbSaveOperation
* spring.data.mongodb.database=fsDB
* spring.data.mongodb.port=27017
* spring.data.mongodb.host=localhost

👉 If the database or collection does not exist, MongoDB will create them automatically when we insert the first document.

## 2. **Model Class (**Student**)**

@Document(collection = "students")

public class Student {

@Id

private String id;

private String name;

private int age;

private String email;

}

### Key Points:

* @Document(collection = "students")  
  → Tells Spring Data MongoDB that this class maps to the **students collection** in MongoDB.  
  → Every object of this class becomes a **document** inside that collection.
* @Id  
  → Marks the field id as the **primary key** for the document.  
  → MongoDB will use \_id internally (if not provided, it auto-generates an ObjectId).
* Lombok annotations (@Data, @AllArgsConstructor, @NoArgsConstructor)  
  → Generate constructors, getters, setters, toString, etc., so we don’t have to write them manually.

## 3. **Repository Layer**

public interface StudentRepository extends MongoRepository<Student, String> {

}

### Key Points:

* MongoRepository<Student, String>  
  → First parameter: model class (Student).  
  → Second parameter: type of the ID field (String).
* Provides ready-made CRUD methods like:
  + save(student)
  + findById(id)
  + findAll()
  + deleteById(id)

4. **Service Layer**

Student student = repository.findById(id).orElse(null);

step by step in simple words:

1. repository.findById(id)

findById(id) is a method from MongoRepository.

It tries to find a document (record) in the collection with the given id.

But instead of returning the Student directly, it returns an \*\*Optional<Student>`.

Example:

If a student with that id exists → Optional<Student> containing the student.

If no student is found → an empty Optional (no value inside).

2. .orElse(null)

Optional provides a method called orElse(defaultValue).

This means:

If the Optional has a value → return that value.

If the Optional is empty → return the value you passed inside orElse().

Here we wrote .orElse(null) → so if the student is not found, it will return null instead of crashing.

3. Full Meaning

So this whole line says:

👉 "Try to find a student with the given id.

If found → store the student in the variable.

If not found → set the variable to null."

4. Why do we need this?

Without orElse(null), you would have to manually check if the Optional has a value:

Optional<Student> optionalStudent = repository.findById(id);

if (optionalStudent.isPresent()) {

student = optionalStudent.get();

} else {

student = null;

}

Using .orElse(null) is just a shortcut for the above code.

## 5. **Main Application**

Student student = new Student(null, "John Doe", 22, "john@example.com");

studentService.createStudent(student);

### Key Points:

* Passing null as id → lets MongoDB generate a new ID.
* Once saved, we get a Student object with the new id.
* Then we can use that id to update or fetch the student later.

## 6. **Behavior of MongoDB with Spring Data**

* No need to **manually create database or collection** → they are created automatically when you save data.
* Every Java object gets stored as a **document** (in JSON-like format) in the specified collection.

# 📝 Summary

* @Document → Maps Java class to MongoDB collection.
* @Id → Marks primary key (\_id).
* findById(id).orElse(null) → Either get the student or null if not found.
* MongoRepository gives ready-made CRUD methods.
* MongoDB auto-creates DB and collection on first insert.
* Lombok helps reduce boilerplate code.

Eg: MongoDbSaveOperation

# 📘 Notes on Delete Operations with Spring Data MongoDB

## 1. @Document in Model

* @Document(collection = "students") → tells Spring this class maps to **MongoDB collection** students.
* Every object of this class = **one document** inside that collection.
* @Id → marks the field as the **primary key** (MongoDB \_id).

## 2. Repository Interface

* MongoRepository<Student, String> → gives ready-made CRUD methods for Student objects.
* Important delete methods:
  + deleteById(id) → deletes one document with matching id.
  + deleteAll() → deletes **all documents** from the collection.
* No need to write queries manually.

## 3. Service Layer

### deleteStudentById(String id)

if (repository.existsById(id)) {

repository.deleteById(id);

}

* existsById(id) checks if record is present.
* If present → deletes it.
* If not present → prints "No student found".
* Good practice → prevents silent failures.

### deleteAllStudents()

repository.deleteAll();

* Removes all records from collection.
* Use carefully → it clears the entire collection.

## 4. Main Class

* We call the service methods from main().
* Example:
* studentService.deleteStudentById("66d4f7c8b12a4a2d9e4f1234");
* studentService.deleteAllStudents();
* This runs the delete operations when app starts.

## 5. Key Points to Remember

* deleteById(id) → only removes **one specific student**.
* deleteAll() → clears **entire collection** (all students).
* Always check existsById(id) before deleteById (optional but safe).
* Service layer keeps code clean and separates **business logic** from **database operations**.
* Repository handles all the database queries internally → you don’t need to write custom queries.

Eg: MongoDbDeleteOperation

## 📒 Notes for count(), findAll(Pageable), findAll(Sort)

### 1. @Document in Model

* We use @Document(collection = "students") in the **Student class**.
* This tells Spring Data MongoDB that this class represents a **MongoDB collection named students**.
* Each object of Student class will be stored as a **document** in that collection.

### 2. count()

public long countStudents() {

return repository.count();

}

* repository.count() returns the **total number of documents** present in the collection.
* Example: If MongoDB has 5 students, it returns 5.
* Very useful when we want to know how many records exist without fetching all of them.

### 3. findAll(Pageable pageable) → ****Pagination****

Page<Student> page = studentService.getStudentsByPage(PageRequest.of(0, 2));

* Pageable is used to fetch **data in chunks (pages)**.
* PageRequest.of(0, 2) → means page number 0 and page size 2.
* It returns at most **2 students** from the first page.
* Page numbers start from **0** (not 1).
* Useful when working with large datasets → we avoid loading everything at once.

### 4. findAll(Sort sort) → ****Sorting****

studentService.getStudentsSorted(Sort.by("name").ascending());

* Sort is used to order the documents.
* Sort.by("name").ascending() → sorts all students by their name field in ascending order.
* We can also use .descending() for reverse order.
* Sorting helps when we want to display ordered data in UI or reports.

### 5. Difference between List and Page

* List<Student> → gives **all records** directly in a list.
* Page<Student> → contains:
  + getContent() → actual list of records for that page
  + getTotalPages() → how many pages exist
  + getTotalElements() → how many total documents
  + hasNext() / hasPrevious() → page navigation info

✅ **In short:**

* count() → total number of records.
* findAll(Pageable) → fetch data in pages (chunks).
* findAll(Sort) → fetch data in sorted order.

Eg: MongoDbCountFindAllPaegableSort