Lecture 51

Database Integrations Using MongoDB

Spring Boot and Spring Data MongoDB

Today’s objective and agenda

Objective

- Connect the Student management project to MongoDB using Spring Data MongoDB.
- Complete Student CRUD backed by MongoDB persistence.
- Verify each milestone using API calls and MongoDB Compass.

Agenda

- Codebase readiness warmups and fixes in the existing project.
- Mongo integration concepts and configuration.
- Live coding milestones to complete Mongo backed CRUD.

Warmups

Improving the Student management project

- You will work on three small tasks on the existing codebase.
- Multiple learners will work on the same task in parallel so we can compare approaches.
- Your output is the fixed code in the ongoing Student management project.
- When you are done, share the file names & your code on Github Discussions

Task 1: Remove Optional from API responses

Observed issue

- Some endpoints return Optional in the response body.

Why is this not a good API contract?

- Optional is a Java helper type, not an API response type.
- API clients should receive a real resource or a clear HTTP outcome.

Where to inspect?

- StudentController. Look for endpoints that return Optional in ResponseEntity.

Definition of done

- GET by id returns Student on success and returns 404 when not found.
- POST returns Student with status 201.
- No endpoint returns Optional in the response body.

Task 2: Fix update endpoint correctness

Observed issue

- Update writes the wrong values into the wrong fields.

Symptom example

```
{
  "name": "Rahul Kumar",
  "email": "rahul.k@gmail.com"
}
```

- Current behavior to look for: name and email appear swapped, or one field overwrites the other.

Where to inspect

- StudentController update endpoint.
- StudentService updateStudent. Check parameter order and the setters being used.

Definition of done

- name updates name and email updates email.
- Updating to an email used by another student returns 409.

Task 3: Fix pagination metadata and total semantics

Observed issue

- page, size, and total in the list response do not represent what the caller asked for or what the filters produced.

Symptom example

```
GET /students?page=1&size=2&active=true
```

- page returns 0 even when page 1 is requested.
- size returns item count, not the requested size.
- total represents all students, not only active students.

Correct behavior

- page equals the requested page, 0-indexed
- size equals the requested size, and
- total equals the number of matching students after filters are applied
- items contains only the slice for that page and size.
- Inspect StudentService searchStudents. Focus on response construction and pagination slicing.

Why we fix these before integrating with MongoDB?

- These changes make the API predictable before we change the persistence layer.
- These changes remove correctness issues that become harder to debug after wiring MongoDB.
- These changes ensure the list endpoint stays accurate when data moves from memory to MongoDB.

MongoDB integration

From last lecture to today’s implementation

- We discussed how Spring Data MongoDB fits into our application structure.
- Today we wire MongoDB into the Student management project and enable persistence.
- Controller behavior and service rules stay consistent while we replace the repository layer.
- Key: MongoDB has ObjectId type, which isn’t supported by Java’s data types
    - We lock the id strategy now so the rest of the integration stays clean.
- Our verification loop:
    - Call the API (Postman or cURL)
    - Confirm the document state in Compass

MongoDB connection strings

What a connection string tells the app

- Where MongoDB is running.
- Which database to use.
- How to authenticate when using a remote cluster.

Parts to recognize

- Scheme, host, and port for local, or cluster host for remote.
- Database name.
- Credentials for a remote database (eg: username, password)
- Options for retry and other behaviors.

```
Format: <scheme>://<host>:<port>/<database>
```

Examples

```
mongodb://localhost:27017/studentdb
```

```
mongodb+srv://demouser:demo123@cluster0.abcd1.mongodb.net/studentdb?retryWrites=true
```

MongoDB Compass: A recap

- We will continue, but to verify the data saved by our Spring Boot application
- After configuring the MongoDB URI in Spring Boot, use the same value to connect in Compass.
- Three areas we will use repeatedly: databases navigation, Documents view, and the filter bar.

Example: filtering by email

```
{ "email": "rahul.k@gmail.com" }
```

Checks when something looks off:

- Refresh Documents after an API call.
- Confirm the database name matches the one in the URI.
- Confirm you are looking at the appropriate collection (eg: students).

Spring Data MongoDB: what it gives us today

- Maps Java objects to MongoDB documents.
- MongoRepository gives CRUD methods without writing implementation code.
- Spring Boot auto configures the Mongo client when the starter is present and a URI is provided
    - In Spring Boot, a *starter* means a dependency like spring-boot-*starter*-web or spring-boot-*starter*-data-mongodb

Code touch points:

```
@Document(collection = "students")
public class Student {
  @Id
  private String id;
}
```

```
public interface StudentRepository
  extends MongoRepository<Student, String> {
}
```

Spring Boot configuration: local MongoDB

- Add the MongoDB URI in application properties.
- This is the minimum required for local connectivity.

```
spring.data.mongodb.uri=mongodb://localhost:27017/studentdb
```

- Pick a database name that matches our project context.
- If the database does not exist yet, MongoDB will create it on first write.

Spring Boot configuration: remote MongoDB

Remote example:

```
mongodb+srv://demouser:demo123@cluster0.abcd1.mongodb.net/studentdb?retryWrites=true
```

If setting environment variables:

```
spring.data.mongodb.uri=${MONGODB_URI}
```

- Do not commit real credentials. Use environment variables for real setups.
- If a remote URI fails, check network access rules and credentials first.

What stays the same and what changes today

Stays the same

- Endpoints and request bodies.
- Validation rules and error behavior.
- Service level logic and constraints.

Changes today

- Student becomes a Mongo document model.
- Repository becomes Mongo backed using MongoRepository.
- Id handling shifts to String ids aligned with Mongo document ids.

Id strategy for Mongo integration

Why this matters

- MongoDB identifies each document using an id value that you see in Compass.
- Using String ids in the API keeps URLs simple and avoids conversion issues in clients.
- When the id looks the same in the API and in Compass, debugging becomes faster and less confusing.

What changes in our project

- Controller path variables for id become String.
- Service methods that accept id become String.
- Student id field becomes String and is treated as the document id.

Example id value

```
65f2c8a9e8d4b3c2a1f0d9e7 (For APIs) <-> ObjectId(65f2c8a9e8d4b3c2a1f0d9e7) (For MongoDB)
```

Live Coding

Milestone A: Connect the app to MongoDB

Code changes

- Add Spring Data MongoDB dependency.
- Add MongoDB URI configuration in application properties.

Proof check

- Application starts cleanly.
- MongoDB connection is established with the configured URI.

If it fails, check these first

- MongoDB is not running.
- URI is incorrect.
- Database name in the URI is not what you expect.

Milestone B: Persistence

Code changes

- Map Student as a Mongo document.
- Replace the in memory repository with MongoRepository.

Proof check

- POST /students returns 201 and a Student response body.
- A student document appears in the students collection in Compass.

Compass check

- Filter by email and confirm the saved fields.

Milestone C: Read by id works and surviving restarts

Proof check

- GET /students/{id} returns the Student from MongoDB.
- Restart the app. GET /students/{id} still returns the Student.

What this proves

- The API is reading from MongoDB, not from in memory storage.

If it fails, check these first

- Controller is still calling the old repository.
- Id type mismatch between controller, model, and repository.
- Database name differs between the app and Compass.

Milestone D: Update and delete work end to end

Goal: Model an Enrollment document to link students and courses

Implement these files

- PUT updates name and email correctly.
- Compass shows the updated values.

Proof check for delete

- DELETE removes the document.
- GET by id returns 404.
- Compass no longer shows the document.

If it fails, check these first

- Update logic still swaps fields.
- Delete is not removing by the correct id.
- Controller still returns Optional or the wrong status for not found.

Your turn!

Activity A: Build the courses collection

Goal

- Add a courses collection where each document represents one course (course catalog).

Target collection shape

```
{
  "_id": "65b7e1d2c4f3a12b9e8d77b9",
  "title": "Spring Boot Fundamentals",
  "level": "BEGINNER"
}
```

Steps

- Create model: src/main/java/<your_package>/model/Course.java
- Create repository: src/main/java/<your_package>/repository/CourseRepository.java
- Seed 2-3 courses. In our existing seeder, also insert courses:
    - “Spring Boot Fundamentals” level “BEGINNER”
    - “MongoDB Basics” level “BEGINNER”
    - “REST APIs with Spring” level “INTERMEDIATE”
- Verify in Compass

Activity B: Build the enrollments collection

Goal

- Add an enrollments collection where each document represents one enrollment (relationship between a student and a course)

Steps

- Create model: src/main/java/<your_package>/model/Enrollment.java
- Create repository: src/main/java/<your_package>/repository/EnrollmentRepository.java
- Seed 2-3 enrollments. In our existing seeder, also insert courses
- Verify in Compass

Target collection shape

```
{
  "_id": "65b7f9a0d4a8c91f2a3b1111",
  "studentId": "65b7e1d2c4f3a12b9e8d77a1",
  "courseId": "65b7e1d2c4f3a12b9e8d77b9",
  "status": "ACTIVE",
  "enrolledAt": "2026-01-25"
}
```

That’s a wrap!