Lecture 52

Database Integrations Using MongoDB

Spring Boot CRUD Operations

Today’s objective and agenda

Objective

Complete student CRUD endpoints using Spring Data MongoDB.
Improve the API behavior student for exception scenarios.
Verify milestones using API calls and MongoDB Compass.

Agenda

Warm-up quiz.
Concept walkthrough.
Live coding milestones.
Activity and wrap-up.

Warm-up quiz

Warm-up quiz

Answer five multiple choice questions.
Reply format: 1 / 2 / 3 / 4
Time: 7 minutes

Topics

MongoDB mapping annotations.
MongoRepository and derived query methods.
Validation and error outcomes.

Question 1

In Spring Data MongoDB, what is the main purpose of @Document(collection = "students")?

1) Marks the class as a MongoDB-mapped entity and tells Spring which collection name to use
2) Automatically creates REST endpoints for the class
3) Makes the fields in the class immutable
4) Forces MongoDB to embed related documents automatically

Reply format: 1 / 2 / 3 / 4

Answer 1

In Spring Data MongoDB, what is the main purpose of @Document(collection = "students")?

1) Marks the class as a MongoDB-mapped entity and tells Spring which collection name to use
2) Automatically creates REST endpoints for the class
3) Makes the fields in the class immutable
4) Forces MongoDB to embed related documents automatically

Correct answer: 1
Why: @Document tells Spring Data that this class is stored in MongoDB and provides the collection name so documents go into the intended collection

Question 2

In Spring Data MongoDB, what does @Id do inside a document class?

1) It maps the field to MongoDB _id, which Spring uses as the document identifier
2) It automatically creates a REST API for the class
3) It converts the field into a unique index automatically
4) It forces the field type to be ObjectId

Reply format: 1 / 2 / 3 / 4

Answer 2

In Spring Data MongoDB, what does @Id do inside a document class?

1) It maps the field to MongoDB _id, which Spring uses as the document identifier
2) It automatically creates a REST API for the class
3) It converts the field into a unique index automatically
4) It forces the field type to be ObjectId

Correct answer: 1
Why: MongoDB stores the primary key as _id, and @Id marks the Java field that represents it.

Question 3

What is the main benefit of extending MongoRepository<Student, String>?

1) It stores all students in memory for fast reads
2) It disables the need for MongoDB Compass
3) It prevents validation errors automatically
4) It provides a ready implementation for CRUD operations without writing repository code

Reply format: 1 / 2 / 3 / 4

Answer 3

What is the main benefit of extending MongoRepository<Student, String>?

1) It stores all students in memory for fast reads
2) It disables the need for MongoDB Compass
3) It prevents validation errors automatically
4) It provides a ready implementation for CRUD operations without writing repository code

Correct answer: 4
Why: Spring Data generates the repository implementation at runtime, so common CRUD methods work without boilerplate.

Question 4

Which method name follows derived query naming rules for case-insensitive email lookup?

1) getStudentByEmailIgnoreCase(String email)
2) findByEmailIgnoreCase(String email)
3) searchStudentEmailIgnoreCase(String email)
4) findIgnoreCaseEmail(String email)

Reply format: 1 / 2 / 3 / 4

Answer 4

Which method name follows derived query naming rules for case-insensitive email lookup?

1) getStudentByEmailIgnoreCase(String email)
2) findByEmailIgnoreCase(String email)
3) searchStudentEmailIgnoreCase(String email)
4) findIgnoreCaseEmail(String email)

Correct answer: 2
Why: Derived queries follow the pattern prefix plus field name plus operator, and findByEmailIgnoreCase matches that pattern.

Question 5

A student update request is sent with an invalid email format, and the controller method uses @Valid. What happens?

1) Spring rejects the request before the service runs and returns a 400 error
2) The controller method runs, and the service fixes the email automatically
3) MongoDB rejects the request and returns a 404 error
4) The repository returns Optional.empty

Reply format: 1 / 2 / 3 / 4

Answer 5

A student update request is sent with an invalid email format, and the controller method uses @Valid. What happens?

1) Spring rejects the request before the service runs and returns a 400 error
2) The controller method runs, and the service fixes the email automatically
3) MongoDB rejects the request and returns a 404 error
4) The repository returns Optional.empty

Correct answer: 1
Why: With @Valid, validation runs before the method body, and invalid input is rejected as a bad request.

Concept walkthrough

What this project is building

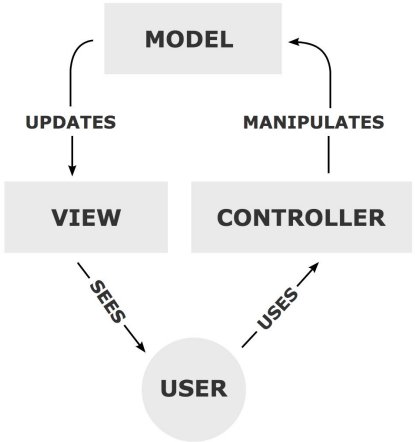
Our application exposes a student API backed by MongoDB for persistence.
The code uses a layered structure so each class has a clear job.
Controller handles HTTP input and output.
Service applies rules and makes decisions, including error outcomes.
Repository performs database operations through Spring Data.
The API is designed to return predictable status codes for common cases.

- Not found returns 404, invalid input returns 400, and duplicate email returns 409.

Spring MVC request flow

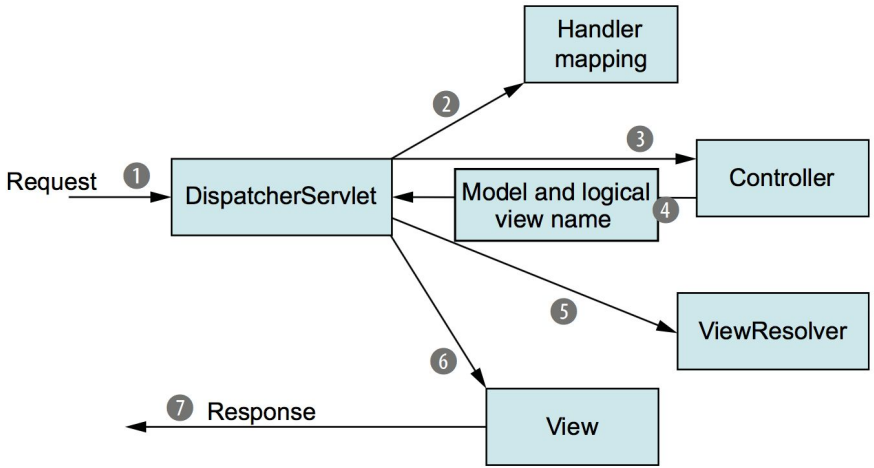
Recap: Spring MVC

Spring MVC is the web layer used to build HTTP APIs in Spring Boot.
Purpose: routes requests to controller methods and converts data between JSON and Java.
In this API context, controller and model are the focus.
The model is the Java data used by the API, such as Student.
The controller defines endpoints such as GET, POST, PUT, and DELETE under /students.
Traditionally, view is the user interface (think: webpage). In our app, view is the JSON response from the server.



DispatcherServlet

Spring MVC routes every request through a single front component called DispatcherServlet.
It selects the correct controller method using the URL path and HTTP method.
It prepares method inputs so controller code stays clean.
It binds path variables, query parameters, and request bodies to Java values.
It coordinates the response by converting returned objects into JSON.



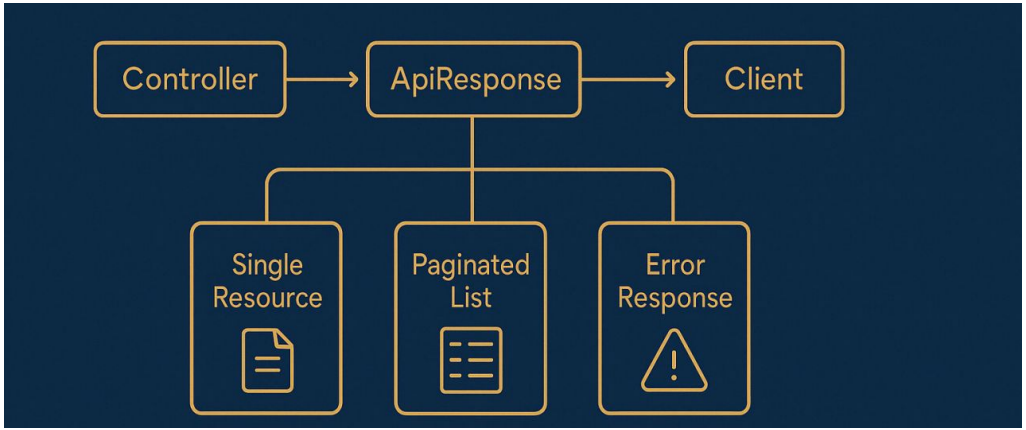
Request handling flow

An HTTP request reaches the embedded server and is handed to Spring MVC.
DispatcherServlet routes the request to the correct controller method.
Spring prepares inputs before the controller method runs.
JSON request bodies are converted into Java objects.
Validation runs automatically when @Valid is present.
Controllers delegate to services, and services delegate to repositories.

```
@GetMapping("/students/{id}")
public ResponseEntity<Student> getStudentById(@PathVariable String id) {
    return ResponseEntity.ok(studentService.getStudentById(id));
}
```

Response flow

Service methods contain the business decisions and call repository methods.
Repository methods perform reads and writes against MongoDB through Spring Data.
The controller returns an object or throws an exception.
Spring converts the result into a JSON response body.
Global exception handling can convert exceptions into consistent HTTP responses.



Spring Boot wiring & Mongo connectivity

Recap: Spring managed objects

Spring Boot creates and manages key application objects at startup.
These managed objects are called beans and can be injected into each other.
Annotations mark common bean types used in this project.

- @RestController for controllers, @Service for services, and @Component for general components.

Recall: Constructor injection makes dependencies explicit and easy to test.

```
public StudentController(StudentService studentService) {
    this.studentService = studentService;
}
```

Repository wiring

Spring Data scans for repository interfaces and registers them as beans.
It generates a runtime implementation, so application code uses the interface only.
Services call repository methods to persist and fetch data.
Custom methods can be added using *derived query naming* rules.

- For example, findByEmailIgnoreCase becomes a real query without writing query code.

```
public interface StudentRepository extends MongoRepository<Student, String> {
    Optional<Student> findByEmailIgnoreCase(String email);
}
```

Auto-configuration

Auto-configuration is how Spring Boot sets up infrastructure based on dependencies and properties. For MongoDB, two inputs determine the setup.

- The MongoDB starter dependency in pom.xml enables MongoDB support.
- The MongoDB URI property in application.properties tells Spring how to connect and which database to use.

Spring Boot then creates a MongoClient and connects repositories to it.

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

Mongo connectivity layers

Your repository methods don’t connect to MongoDB by themselves.
Spring Data sits in the middle and turns those method calls into real database queries.
Those queries are executed by the MongoDB Java Driver (the official MongoDB library).
The driver uses MongoClient to open and reuse connections, then sends commands to the MongoDB server.
So if the URI is wrong, the server is down, or the DB name is off, this “connection layer” is where it fails.

Mapping Java to MongoDB documents

Recap: Collections and documents

MongoDB stores data as documents inside collections.
A collection is a named group of documents, and a document is a JSON-like record.
Every document has a unique primary key field called _id.
If _id is not provided, MongoDB generates one automatically.
In out project, each student is stored as one document in the students collection.

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

Recap: Document mapping with @Document

@Document marks a class as a MongoDB entity managed by Spring Data.

- links the class to a MongoDB collection (where documents for that class are stored).

You can set the exact collection name with @Document(collection = "...") to avoid name-guessing.
Repositories use this mapping so reads/writes always hit the intended collection.
In this project, Student is stored in the students collection.

```
@Document(collection = "students")
public class Student { }
```

Id mapping with @Id

MongoDB stores the document identifier in the _id field.
@Id marks the Java field that represents that identifier.
A Java field named id is stored as _id in MongoDB when it is annotated with @Id.

- Out project uses an id of type String, and Compass displays the stored value under _id.
- If the stored value is an ObjectId, it can still be carried as a string in the API.

```
@Id
private String id;
```

Unique email and indexes

A unique index is a database rule that prevents duplicate values for a field.
In this project, email is unique to prevent two student records from sharing the same email.
The database enforces the rule during writes (POST, PUT).
Spring Data can create indexes from annotations when auto index creation is enabled in application.properties.

```
@Indexed(unique = true)
private String email;

spring.data.mongodb.auto-index-creation=true
```

Spring Data repositories and queries

Role of MongoRepository

A Repository is your app's database API for one document type (example: Student).
It uses your @Document mapping, so reads/writes go to the correct collection.
What does the developer write?

- only an interface and Spring Data creates the implementation at runtime (as a Spring Bean).

What does Spring Data generate:

- You instantly get common CRUD operations: save, findById, findAll, deleteById.

You can add *derived queries* by naming methods (example: findByEmailIgnoreCase).

```
public interface StudentRepository extends MongoRepository<Student, String> {
  Optional<Student> findByEmailIgnoreCase(String email);
}
```

Built-in repository methods

save persists a document and can be used for create and update.
findById fetches a document by id and returns Optional.
findAll returns documents, and findAll(Pageable) returns a Page with paging metadata.
deleteById deletes a document by id.

```
Pageable pageable = PageRequest.of(0, 10);       // page 0, size 10
Page<Student> page = repo.findAll(pageable);
List<Student> items = page.getContent();
long total = page.getTotalElements();
```

Save semantics

save is a single method that supports both create and update.

- When id is missing, MongoDB generates _id and a new document is created.
- When id is present, the document with that id is written as an update.

save writes a document representation, so controlled updates usually follow fetch, modify, save.

```
Student existing = repo.findById(id).orElseThrow(...);
existing.setName(body.getName());
return repo.save(existing);
```

Derived query methods

Spring Data can generate MongoDB queries from repository method names

The naming pattern is prefix plus field name plus operator.
Pattern: Prefix + By + Field(s) + Operator(s)

- Example: findByEmailIgnoreCase - “find student where email matches (case-insensitive)”

Rule: Field names must match your Java properties (not Mongo column names).
Combine: conditions using And / Or (example: findByCityAndActive).
Supports: sorting and paging via Sort / Pageable parameters (we’ll see these in the upcoming slide)
Outcome: Spring generates the query from these method definitions at runtime.

```
Optional<Student> findByEmailIgnoreCase(String email);
boolean existsByEmailIgnoreCase(String email);
```

Examples: Derived query methods

Common prefixes

- findBy (return docs), existsBy (true/false), countBy (number), deleteBy (remove)

Common operators

- IgnoreCase, Containing, StartingWith, EndingWith, In, Between, LessThan, GreaterThan

```
Optional<Student> findByEmailIgnoreCase(String email);
boolean existsByEmailIgnoreCase(String email);

List<Student> findByCityAndActive(String city, boolean active);
List<Student> findByNameContainingIgnoreCase(String namePart);

Page<Student> findByDepartment(String dept, Pageable pageable);
List<Student> findByAgeGreaterThan(int age, Sort sort);
long countByActive(boolean active);
```

Pagination with Pageable

Recall: Pagination returns results in chunks (eg: 10 at a time) instead of loading everything.
Pageable is the request: which page? how many items? what sort?
Page<T> is the response: items + metadata (eg: totalElements, totalPages).
Getting totals usually triggers an extra query to fetch the count from MongoDB.

- This is helpful for UI, but adds a cost to your API layer.

If you don't need totals, prefer Slice<T> (next/previous paging with no count).

```
// page is 0, size is 10
Pageable pageable = PageRequest.of(0, 10, Sort.by("createdAt").descending());
Page<Student> page = repo.findAll(pageable);

Slice<Student> slice = repo.findByCity("Bangalore", PageRequest.of(0, 10));
boolean hasNext = slice.hasNext();
```

Error behavior and edge cases

Missing documents and 404

A query may return nothing because the document does not exist. Eg: find student with ID 9999
How can your application handle it?

- Repository methods express absence using Optional for findById.
- Service code converts absence into a not found decision by throwing a not found exception.
- Global exception handling maps that exception into a 404 response.
- The HTTP response is returned to the client.

```
return repo.findById(id)
    .orElseThrow(() -> new StudentNotFoundException(id));
```

400 Bad Request: Invalid Input

When input is invalid, fail fast at the API boundary.
Request body validation (@Valid) runs before your controller/service logic.

- Use the @Email or appropriate annotations in the DTO
- Result: Bad email / blank name can return a validation exception (400 Bad Request)

Query parameter validation protects list endpoints (paging/sorting/filtering).

- Eg: if sortBy, sortDir, negative page/size is part of the query parameters, throw IllegalArgumentException (400 Bad Request)

Global exception handling turns these failures into a 400 response & returns the HTTP response

409 Conflict: Duplicate Data

Duplicate email is not “invalid input,” but a conflict with existing data.
Our service can check first and return a friendly message early, but if two requests hit at the same time, only the database can reliably prevent duplicates.

Enforce uniqueness with a unique index in MongoDB (this is the real guarantee).
When the DB rejects the write, Spring Data surfaces a duplicate-key exception

- This can map to 409 Conflict with a friendly HTTP response message.

That’s a wrap!