Day-51: What Is @ControllerAdvice in Spring Boot?

A comprehensive guide to global exception handling in Spring Boot applications

Global Exception Handling Made Simple

@ControllerAdvice is a powerful annotation used to handle exceptions **globally** across all controllers in your Spring Boot application. Instead of writing @ExceptionHandler methods repeatedly in every single controller class, you write it just once in a dedicated @ControllerAdvice class and apply it everywhere automatically.

This approach keeps your codebase clean, centralized, and incredibly easy to maintain. When an exception occurs anywhere in your application, Spring Boot automatically routes it to your global handler. This means consistent error responses, reduced code duplication, and a single source of truth for exception management.

Think of @ControllerAdvice as your application's central error management station—one place where all exceptions are caught, processed, and handled uniformly across your entire application.

Practical Implementation Example

The Global Handler Class

Creating a centralized exception handler is straightforward. You define a class annotated with @ControllerAdvice and add methods annotated with @ExceptionHandler to catch specific exception types.

This handler becomes active across your entire application automatically. Spring Boot manages the routing and ensures every exception is directed to the appropriate handler method.

```
@ControllerAdvice
public class GlobalExceptionHandler {

  @ExceptionHandler(Exception.class)
  public String handleAll(Exception ex) {
    return "Error: " + ex.getMessage();
  }
}
```

Centralized Logic

Write exception handling code once and apply it everywhere

Flexible Responses

Return JSON objects, custom status codes, or structured error messages

Clean Controllers

Keep controller classes focused on business logic without error handling clutter

"Centralized error handling leads to cleaner, safer applications."

Join DURGASOFT for Expert Training

> ✨ *If you want online training in SPRING Framework with SPRING BOOT Cloud, join DURGASOFT – India's trusted training institute.*

Contact Numbers

📞 9246212143
📞 8885252627

Website

🌐 www.durgasoftonline.com

Email Address

📩
durgasoftonlinetraining@gmail.com

Day-52: What Is @Valid in Spring Boot?

Mastering automatic request validation for REST APIs

Automatic Input Data Validation

@Valid is a powerful annotation used to **validate input data** automatically in REST APIs before processing begins. When a client sends incorrect, incomplete, or malformed data to your API, Spring Boot will detect the issues immediately and return meaningful validation error messages without executing your business logic.

This annotation saves enormous amounts of development time by eliminating manual validation checks. It prevents bad data from entering your application, protects your database from invalid records, and ensures data integrity throughout your system.

By catching validation errors at the entry point of your API, you create a robust first line of defense that makes your application safer, more reliable, and easier to maintain. Your business logic remains clean because it only processes data that has already passed validation.

Complete Validation Example

DTO Class with Constraints

```
public class UserRequest {

  @NotEmpty
  private String name;

  @Min(18)
  private int age;
}
```

The DTO class defines validation rules using annotations like @NotEmpty, @Min, @Max, @Email, and many others from the javax.validation package.

Controller with Validation

```
@PostMapping("/save")
public String saveUser(
  @Valid @RequestBody UserRequest req) {
  return "Valid Data Received";
}
```

The @Valid annotation triggers automatic validation. If the data is invalid, Spring throws a MethodArgumentNotValidException before your method executes.

01

Define Constraints

Add validation annotations to your DTO class fields

02

Apply @Valid

Use @Valid annotation in your controller method parameter

03

Handle Errors

Combine with @ControllerAdvice for custom error messages

"Validate early, avoid problems later — build safe APIs."

Transform Your Spring Boot Skills

✨ *If you want online training in SPRING Framework with SPRING BOOT Cloud, join DURGASOFT – India's trusted training institute.*

Call Us Today

📞 9246212143
📞 8885252627

Visit Our Website

🌐 www.durgasoftonline.com

Email Your Queries

📩 durgasoftonlinetraining@gmail.com

Day-53: What Is @Entity in Spring Boot (JPA)?

Converting Java classes into database tables with JPA

Mapping Java Classes to Database Tables

@Entity is a fundamental JPA annotation used to mark a Java class as a **database table** mapping. Each instance (object) of this class represents a **row** in the corresponding database table. This powerful feature is at the heart of Object-Relational Mapping (ORM) in Spring Boot.

When you annotate a class with @Entity, Spring Boot and Hibernate work together to automatically create the corresponding database table structure. The class fields become table columns, the data types are mapped appropriately, and relationships between entities are managed seamlessly.

This approach eliminates the need to write SQL CREATE TABLE statements manually. Spring Boot handles table creation, updates, and management automatically based on your entity definitions. It also provides sophisticated CRUD operations, transaction management, and relationship handling—all through simple Java code.

Entity Class Structure

```
@Entity
public class User {

  @Id
  @GeneratedValue
  private Long id;

  private String name;
  private String city;
}
```

Understanding the Annotations

- **@Entity** – Marks this class as a database table mapping
- **@Id** – Designates the primary key field
- **@GeneratedValue** – Enables auto-increment for the primary key
- **Private fields** – Become table columns automatically

Spring Boot + JPA creates a powerful combination that accelerates database development dramatically while maintaining clean, readable code.

Java Class

Define your entity with @Entity annotation

ORM Processing

Hibernate maps the class to database structure

Database Table

Table automatically created with proper schema

"Your Java classes can become real database tables — that's the magic of JPA."

Learn Spring Boot at DURGASOFT

✨ *If you want online training in SPRING Framework with SPRING BOOT Cloud, join DURGASOFT – India's trusted training institute.*

Phone Numbers

9246212143
8885252627

Online Portal

www.durgasoftonline.com

Email Support

durgasoftonlinetraining@gmail.com

Day-54: What Is @Repository in Spring Boot?

Building robust data access layers with Spring annotations

Data Access Object Layer Component

@Repository is a specialized Spring annotation used specifically for **DAO (Data Access Object)** classes that interact with databases. It serves multiple critical purposes in your application architecture and is an essential component of well-structured Spring Boot applications.

This annotation helps Spring identify and register database-related components in the application context. More importantly, it provides automatic exception translation—converting low-level database exceptions (like SQLException) into Spring's unified **DataAccessException** hierarchy. This makes error handling cleaner, more consistent, and framework-independent.

@Repository is commonly used with various persistence technologies including JPA, Hibernate, JDBC Template, and other ORM frameworks. It represents the data layer in the standard three-tier architecture: Controller → Service → Repository. This separation of concerns creates maintainable, testable, and scalable applications.

Repository Implementation Pattern

```
@Repository
public class UserRepository {

  @Autowired
  private JdbcTemplate jdbc;

  public List getAllUsers() {
    return jdbc.queryForList(
      "SELECT name FROM users",
      String.class
    );
  }
}
```

Component Identification

Marks the class as a database layer component, making it discoverable by Spring's component scanning mechanism

Exception Translation

Automatically converts database-specific exceptions into Spring's consistent DataAccessException types

Layered Architecture

Promotes clean separation between Controller, Service, and Repository layers for better code organization

"Strong applications are built on strong data layers — start with clean repositories."

Expert Spring Boot Training

✨ *If you want online training in SPRING Framework with SPRING BOOT Cloud, join DURGASOFT – India's trusted training institute.*

Direct Contact

📞 9246212143, 8885252627

Website

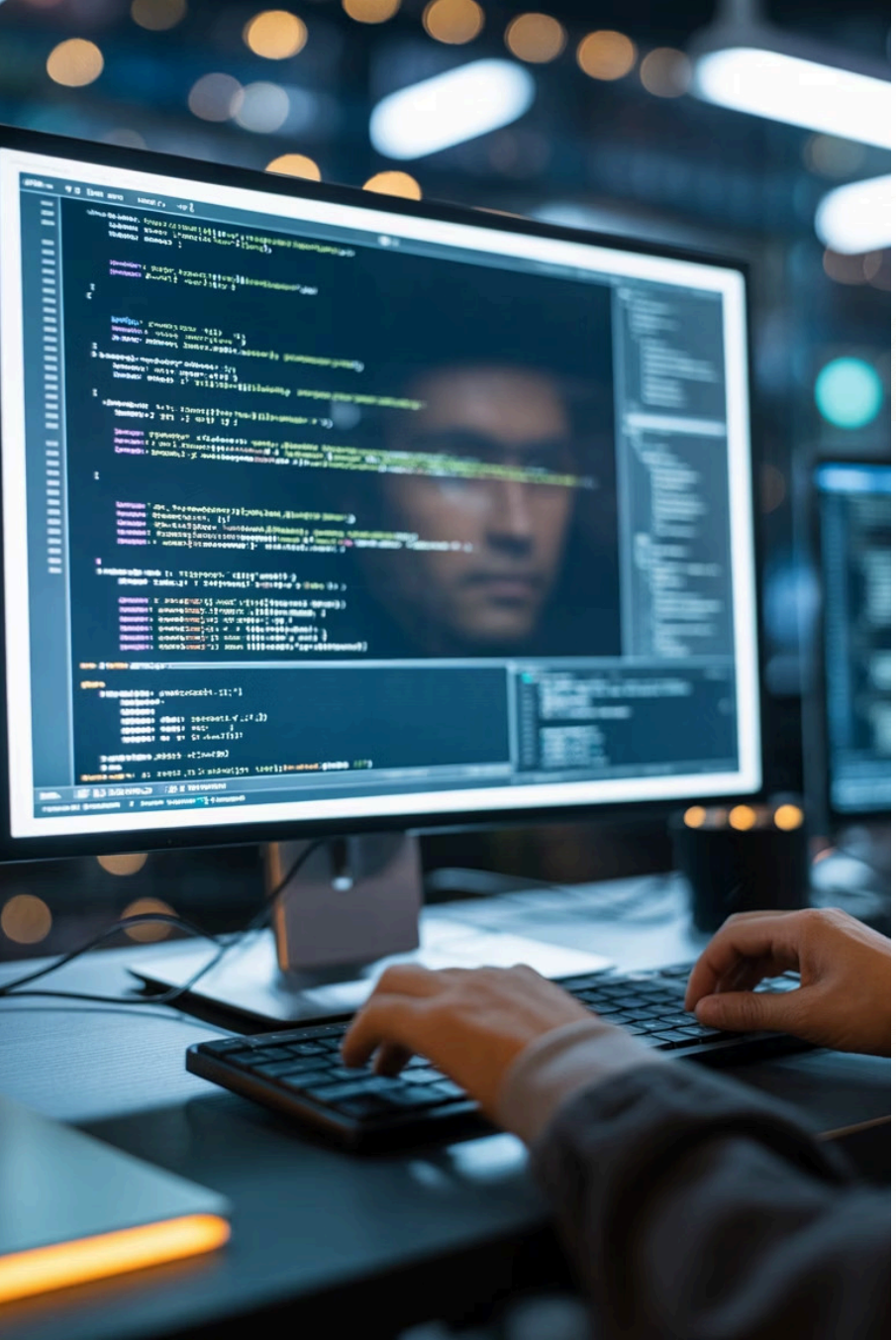
🌐 www.durgasoftonline.com

Email

📩 durgasoftonlinetraining@gmail.com

Day-55: What Is CrudRepository in Spring Data JPA?

Zero-code database operations with Spring Data JPA



Ready-Made CRUD Operations Interface

CrudRepository is a powerful Spring Data interface that provides **complete CRUD (Create, Read, Update, Delete) operations** without requiring you to write any SQL queries or implementation code. This revolutionary approach dramatically accelerates database development.

When you extend CrudRepository, you immediately gain access to essential methods like save(), findById(), findAll(), deleteById(), count(), and existsById(). These methods work out of the box—Spring Data JPA automatically generates the implementation at runtime based on your entity structure.

This interface represents the foundation of Spring Data's repository abstraction. It eliminates boilerplate code, reduces development time, minimizes bugs, and creates consistent data access patterns across your entire application. Perfect for rapid application development, microservices architectures, and prototyping.

CrudRepository Interface Usage

```
public interface UserRepository
  extends CrudRepository {

}
```

This simple interface declaration gives you instant access to all CRUD operations. No implementation class needed—Spring creates everything automatically.

Built-In Methods You Get

- **save(entity)** – Insert or update an entity
- **findById(id)** – Retrieve entity by primary key
- **findAll()** – Get all entities from the table
- **deleteById(id)** – Remove entity by primary key
- **count()** – Get total number of entities
- **existsById(id)** – Check if entity exists

Perfect for microservices and quick development cycles where speed and simplicity matter most.

"Write less, achieve more — let Spring Data do the heavy lifting."

Master Spring Boot with DURGASOFT

✨ *If you want online training in SPRING Framework with SPRING BOOT Cloud, join DURGASOFT – India's trusted training institute.*

Contact Details

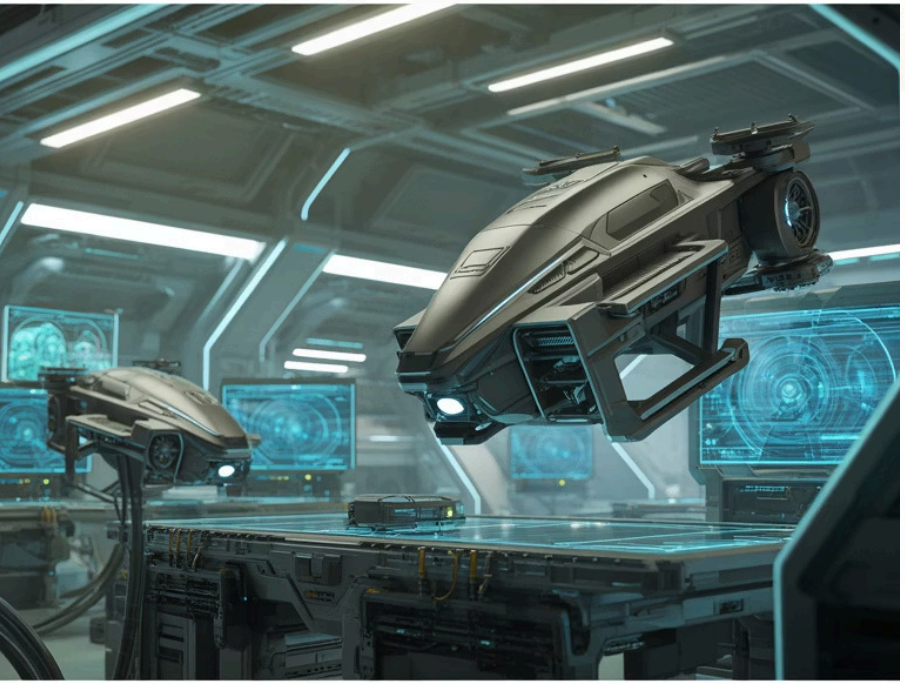
📞 9246212143
📞 8885252627

Visit Online

🌐 www.durgasoftonline.com

Email Us

📩 durgasoftonlinetraining@gmail.com



Day-56: What Is JpaRepository in Spring Boot?

Advanced repository features for enterprise applications

Enterprise-Grade Repository Interface

JpaRepository is an advanced Spring Data interface that extends both CrudRepository and PagingAndSortingRepository, combining their capabilities into one powerful interface. It provides **complete CRUD operations**, **pagination support**, **sorting capabilities**, and many specialized JPA features—all without writing a single SQL query.

This interface represents the most commonly used repository type in real-world Spring Boot projects and production applications. It offers everything from basic CRUD to advanced batch operations, custom query methods, and sophisticated data retrieval strategies.

JpaRepository is the go-to choice for enterprise applications, microservices architectures, and any system requiring robust, scalable data access. It provides the perfect balance between ease of use and powerful functionality, making it ideal for both small projects and large-scale production systems handling millions of records.

JpaRepository Complete Feature Set

```
public interface UserRepository
  extends JpaRepository {

}
```

All CRUD Methods

Inherits save, findById, findAll, deleteById, and more from CrudRepository

Pagination & Sorting

Built-in findAll(Pageable) and findAll(Sort) for handling large datasets efficiently

Batch Operations

Methods like saveAll(), deleteAllInBatch() for processing multiple records at once

JPA Specific Features

flush(), getOne(), and other specialized JPA operations for fine-grained control

Plus automatic query derivation from method names—write `findByName(String name)` and Spring generates the query automatically!

"Level up your data layer — use JpaRepository for maximum power."

Professional Training at DURGASOFT

✨ *If you want online training in SPRING Framework with SPRING BOOT Cloud, join DURGASOFT – India's trusted training institute.*

Call Anytime

📞 9246212143
📞 8885252627

Explore Online

🌐 www.durgasoftonline.com

Send Email

📩 durgasoftonlinetraining@gmail.com

Day-57: What Is @PathVariable in Spring Boot?

Capturing dynamic values from URL paths in REST APIs

Dynamic URL Parameter Extraction

@PathVariable is a fundamental Spring annotation used to capture **dynamic values directly from the URL path**. It transforms your REST APIs from static endpoints into flexible, parameterized routes that can handle variable inputs naturally and intuitively.

This annotation makes your API design cleaner and more RESTful. Instead of passing data through query parameters or request bodies for simple lookups, you embed the values directly in the URL structure. This creates URLs that are intuitive, easy to read, and follow REST best practices.

@PathVariable is perfect for resource identification scenarios like retrieving a specific user (/users/10), viewing a product (/products/5), accessing an order (/orders/123), or any situation where you're working with a specific resource instance. It's the standard approach for GET requests that target individual entities and is widely used in modern microservices and REST API design.

Path Variable Implementation

Single Variable Example

```
@GetMapping("/user/{id}")
public String getUser(
  @PathVariable int id) {
  return "User ID: " + id;
}
```

URL: /user/50
Output: "User ID: 50"

Multiple Variables Example

```
@GetMapping("/order/{userId}/{orderId}")
public String getOrder(
  @PathVariable int userId,
  @PathVariable int orderId) {
  return "User: " + userId +
        ", Order: " + orderId;
}
```

URL: /order/10/250
Output: "User: 10, Order: 250"

Client Makes Request

User visits /user/50 in browser or API client

1

Spring Routes Request

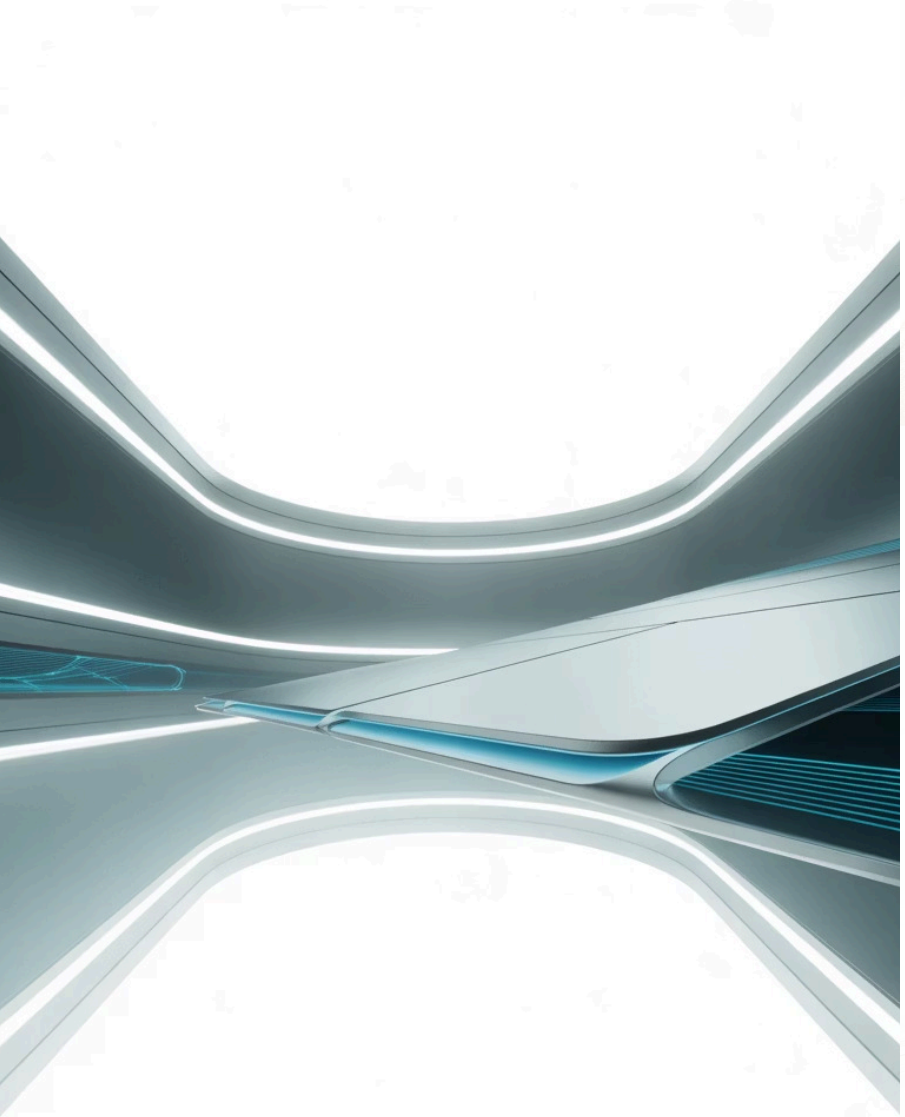
Framework matches URL pattern and extracts value

2

Method Receives Value

Your method gets id=50 as a parameter automatically

3



"Small annotations create big flexibility — APIs become smarter."

Advance Your Skills with DURGASOFT

✨ *If you want online training in SPRING Framework with SPRING BOOT Cloud, join DURGASOFT – India's trusted training institute.*

Phone Support

📞 Call us at 9246212143 or 8885252627 for immediate assistance and course details

Online Resources

🌐 Visit www.durgasoftonline.com to explore courses and schedules

Email Inquiries

📩 Write to durgasoftonlinetraining@gmail.com for detailed information

Day-58: What Is @RequestParam in Spring Boot?

Reading query parameters for flexible API inputs

Query Parameter Extraction

@RequestParam is a Spring annotation used to read **query parameters** from the URL—those key-value pairs that appear after the question mark in a URL. It's the perfect solution when you need optional values, search filters, sorting preferences, pagination controls, or any additional input that customizes the API response.

Query parameters are ideal for scenarios like search APIs (/search?name=Ravi&city=Hyderabad), filtering large datasets, providing sort options, controlling pagination (page=1&size=10), or passing optional configuration flags. They make your API flexible without changing the base URL structure.

One of @RequestParam's most powerful features is the ability to set default values. This prevents errors when clients don't provide certain parameters and ensures your API degrades gracefully. You can mark parameters as optional or required, providing fine-grained control over your API's input requirements while maintaining backward compatibility.

Request Parameter Usage Examples

```
@GetMapping("/search")
public String searchUser(
  @RequestParam String name,
  @RequestParam(defaultValue = "India") String country) {
  return name + " from " + country;
}
```

Basic Usage

URL: /search?name=Kiran
Output: "Kiran from India" (uses default value)

Custom Parameters

URL: /search?name=Kiran&country=USA
Output: "Kiran from USA" (overrides default)

Real-World Uses

Perfect for search APIs, data filtering, result sorting, pagination controls, and optional configuration settings

"Flexible inputs give flexible APIs — design with freedom."

Learn Spring Boot at DURGASOFT

✨ *If you want online training in SPRING Framework with SPRING BOOT Cloud, join DURGASOFT – India's trusted training institute.*

Phone

📞 9246212143, 8885252627

Website

🌐 www.durgasoftonline.com

Email

📩 durgasoftonlinetraining@gmail.com

Day-59: What Is @RequestBody in Spring Boot?

Automatic JSON-to-object conversion for REST APIs

JSON Data Processing Made Simple

@RequestBody is an essential Spring annotation used to read **JSON data** sent from client applications—whether that's Postman during testing, frontend web applications, mobile apps, or other backend services. Spring Boot automatically converts the incoming JSON into Java objects using the powerful Jackson library.

This annotation is fundamental for modern REST API development. It's used primarily with POST, PUT, and PATCH HTTP methods where clients send data to create or update resources on the server. The conversion happens automatically—no manual parsing, no error-prone string manipulation, just clean object-oriented code.

@RequestBody eliminates the tedious work of manually parsing JSON strings, extracting values, handling type conversions, and dealing with malformed data. Spring handles all of this behind the scenes, giving you a ready-to-use Java object that you can immediately validate, process, and persist to your database.

JSON to Object Conversion Flow

JSON Input (Client → Server)

```
{
  "name": "Ravi",
  "city": "Hyderabad"
}
```

Controller Method

```
@PostMapping("/save")
public String saveUser(
  @RequestBody User user) {
  return "User: " + user.getName();
}
```

Expected Output

Response: "User: Ravi"

Key Benefits

- **No Manual Parsing** – Spring handles JSON conversion automatically
- **Type Safety** – Get strongly-typed Java objects, not strings
- **Validation Ready** – Combine with @Valid for instant input validation
- **Clean Code** – Focus on business logic, not data transformation

"APIs become powerful when they understand your JSON clearly."

Transform Your Career with DURGASOFT

✨ *If you want online training in SPRING Framework with SPRING BOOT Cloud, join DURGASOFT – India's trusted training institute.*

📞 Phone Contact

Call us at 9246212143 or 8885252627

🌐 Website

Visit www.durgasoftonline.com

📩 Email

durgasoftonlinetraining@gmail.com

Day-60: What Is ResponseEntity in Spring Boot?

Complete control over HTTP responses in REST APIs

Professional API Response Management

ResponseEntity is a powerful Spring class that gives you complete control over the **HTTP status code**, **response headers**, and **response body** of your REST API responses. Instead of just returning raw data, you can construct comprehensive, professional HTTP responses that communicate status, metadata, and content clearly.

This class is essential for building production-grade REST APIs and microservices where proper HTTP semantics matter. It allows you to return appropriate status codes (200 for success, 201 for created, 404 for not found, 500 for errors), set custom headers for caching or authentication, and structure your response body in any format.

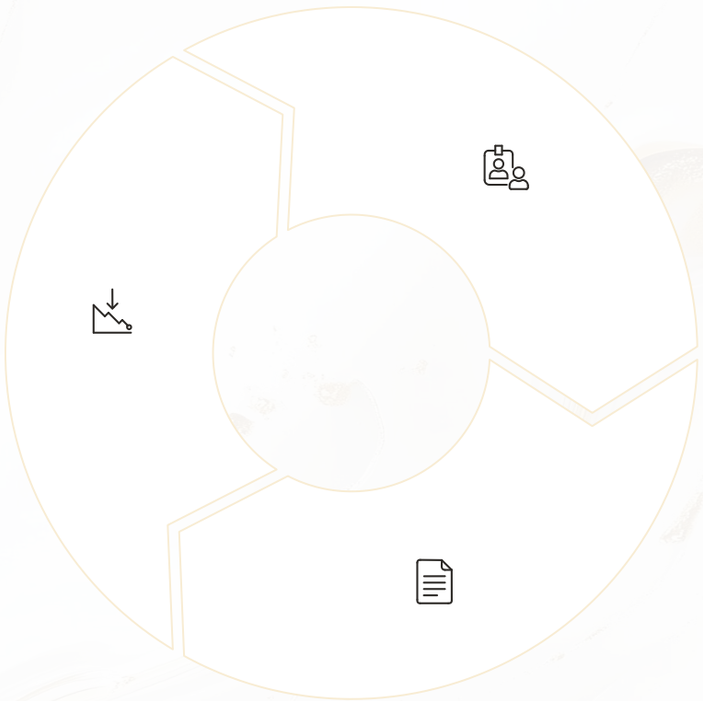
ResponseEntity transforms simple data returns into sophisticated, standards-compliant API responses. It's particularly valuable in error handling scenarios, where you need to communicate not just what went wrong, but also provide proper HTTP status codes and structured error messages. This leads to APIs that are easier to consume, debug, and integrate with other systems.

ResponseEntity Implementation Examples

```
@GetMapping("/status")
public ResponseEntity getStatus() {
  return ResponseEntity
    .status(200)
    .header("App-Version", "1.0")
    .body("Application is Running");
}
```

Set Status Code

Choose appropriate HTTP codes: 200, 201, 400, 404, 500, etc.



Add Custom Headers

Include metadata like versioning, caching directives, authentication tokens

Structure Response Body

Return any type: String, JSON objects, lists, custom DTOs, error structures

Advanced Example:

```
return ResponseEntity.ok(new ApiResponse("Success", data));
```

Perfect for standardizing API responses with consistent structure across your entire application!

"Professional APIs speak clearly — ResponseEntity gives you that control."

Join DURGASOFT Today

✨ *If you want online training in SPRING Framework with SPRING BOOT Cloud, join DURGASOFT – India's trusted training institute.*

1

Top Institute

India's most trusted training provider for Spring Boot and Cloud technologies

100%

Practical Training

Hands-on learning with real-world projects and industry-standard practices

24/7

Support Available

Round-the-clock assistance for all your learning queries and doubts

Contact Information:

📞 **Phone:** 9246212143, 8885252627
🌐 **Website:** www.durgasoftonline.com
📩 **Email:** durgasoftonlinetraining@gmail.com