Day-41: What Are Spring Boot Actuator Endpoints?

A comprehensive guide to understanding and implementing production-ready monitoring in your Spring Boot applications.

Understanding Spring Boot Actuator

Spring Boot Actuator provides a powerful suite of built-in endpoints that enable you to **monitor, manage, and analyze** your application in real-time. This production-grade feature gives you instant visibility into your application's health, performance metrics, environment configuration, registered beans, API mappings, and much more—all without writing a single line of additional code.

Actuator is absolutely essential for modern cloud deployments, DevOps workflows, and production support teams. It transforms your application into an observable system that can be easily monitored, debugged, and maintained in production environments.

Getting Started with Actuator

Maven Dependency

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-
actuator</artifactId>
</dependency>
```

Simply add this dependency to your `pom.xml` file, and Spring Boot will automatically configure all actuator endpoints for you. No additional setup required!

Essential Actuator Endpoints

/actuator/health

Provides instant health status of your application and its dependencies. Perfect for load balancer health checks.

/actuator/info

Displays custom application information including version, build details, and team contact information.

/actuator/metrics

Exposes comprehensive metrics including CPU usage, memory consumption, JVM statistics, and custom metrics.

/actuator/beans

Lists all Spring Beans registered in your application context—invaluable for debugging dependency injection issues.

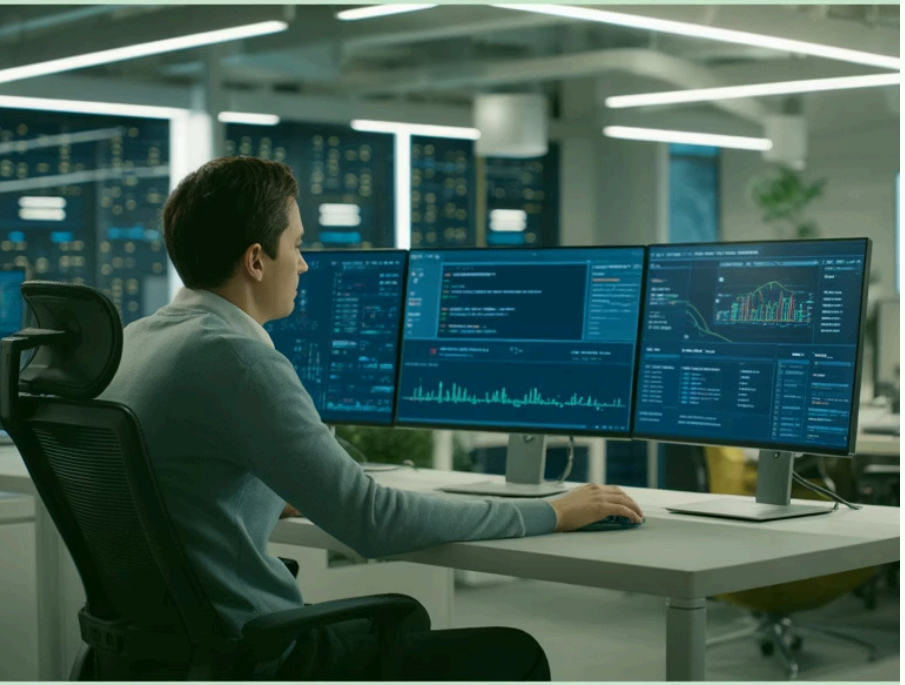
/actuator/mappings

Shows all REST endpoints, HTTP methods, and controller mappings in your application.

Securing Actuator Endpoints

Production applications should always secure Actuator endpoints using Spring Security. You can configure role-based access control to ensure that only authorized personnel can access sensitive monitoring data.

Best Practice: In production, expose only the `/health` endpoint publicly and secure all other endpoints behind authentication and authorization.



"Great developers don't just build apps—they monitor them."

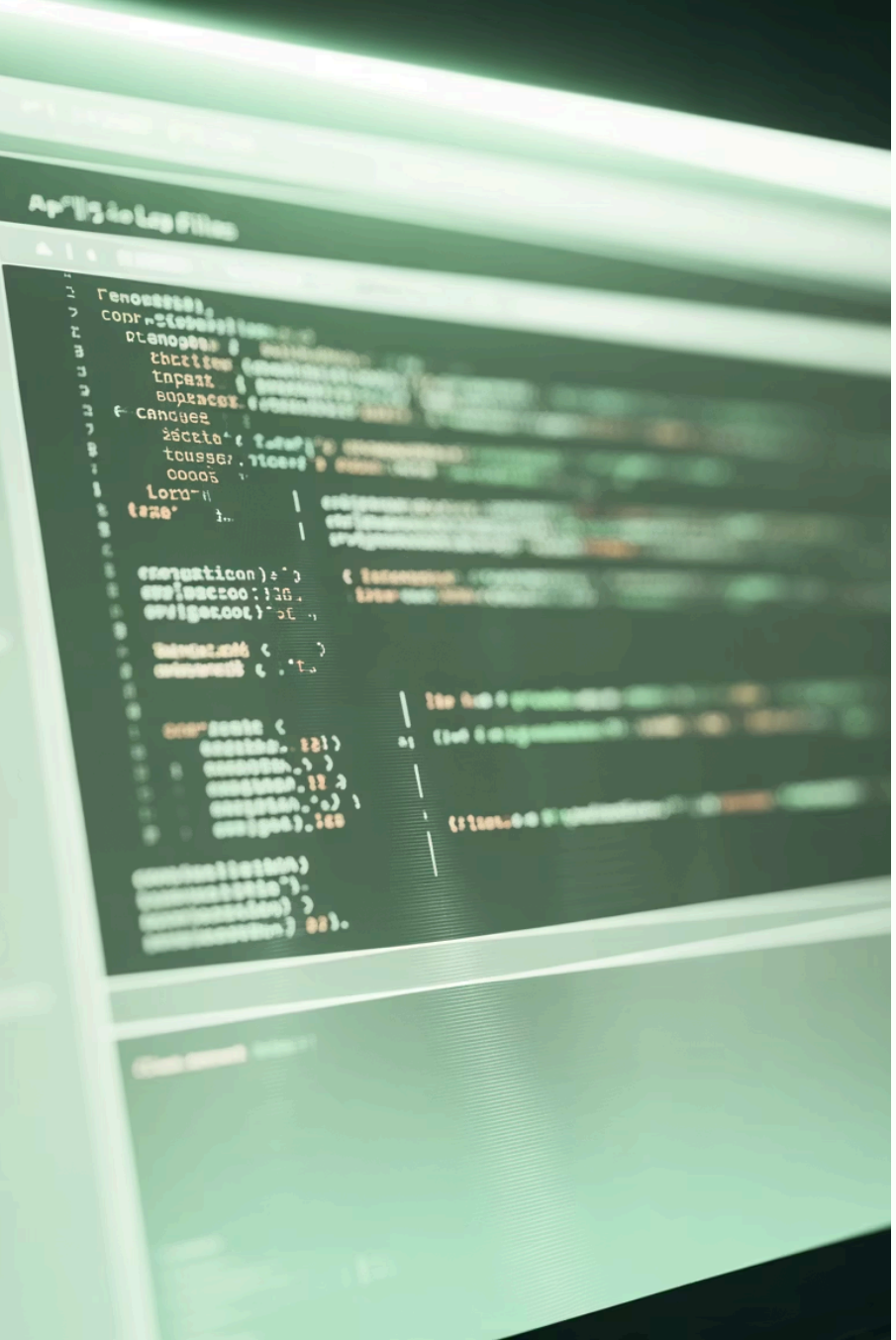
Join DURGASOFT for Expert Spring Boot Training

If you want comprehensive online training in SPRING Framework with SPRING BOOT and Cloud technologies, join DURGASOFT–India's most trusted training institute for Java and Spring ecosystem.

📞 **Contact:** 9246212143, 8885252627

🌐 **Website:** www.durgasoftonline.com

📩 **Email:** durgasoftonlinetraining@gmail.com



Day-42: What Is Spring Boot Logging?

SLF4J + Logback

Master the art of application logging for better debugging, monitoring, and production support.

The Power of Logging

Spring Boot uses **SLF4J** (Simple Logging Facade for Java) as the logging API and **Logback** as the default logging implementation. Logging is your window into understanding what your application is doing at any given moment—it helps you track errors, warnings, application flow, performance bottlenecks, and debugging information.

Spring Boot provides excellent built-in logging support, which means you can start logging meaningful information with just one line of code. No complex configuration files or third-party dependencies needed to get started.

Implementing Logging in Your Application

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@RestController
public class MyController {

 private static final Logger logger =
 LoggerFactory.getLogger(MyController.class);

 @GetMapping("/home")
 public String home() {
 logger.info("Home API called");
 return "Welcome!";
 }
}
```

Understanding Log Levels

01

ERROR

Critical issues that need immediate attention

02

WARN

Potential problems that should be reviewed

03

INFO

Important application events and milestones

04

DEBUG

Detailed information for debugging purposes

05

TRACE

Very detailed diagnostic information

Configuring Log Levels

Application Properties

```
logging.level.root=INFO
logging.level.com.durgasoft=DEBUG
logging.file.name=application.log
logging.pattern.console=%d{yyyy-MM-dd HH:mm:ss} - %msg%n
```

You can easily control logging behavior through `application.properties`. Set different log levels for different packages to get exactly the information you need.

In production environments, proper logging configuration is crucial for troubleshooting issues and monitoring application health.

"Good logging is like having X-ray vision inside your application."

Join DURGASOFT for Expert Spring Boot Training

If you want comprehensive online training in SPRING Framework with SPRING BOOT and Cloud technologies, join DURGASOFT–India's most trusted training institute for Java and Spring ecosystem.

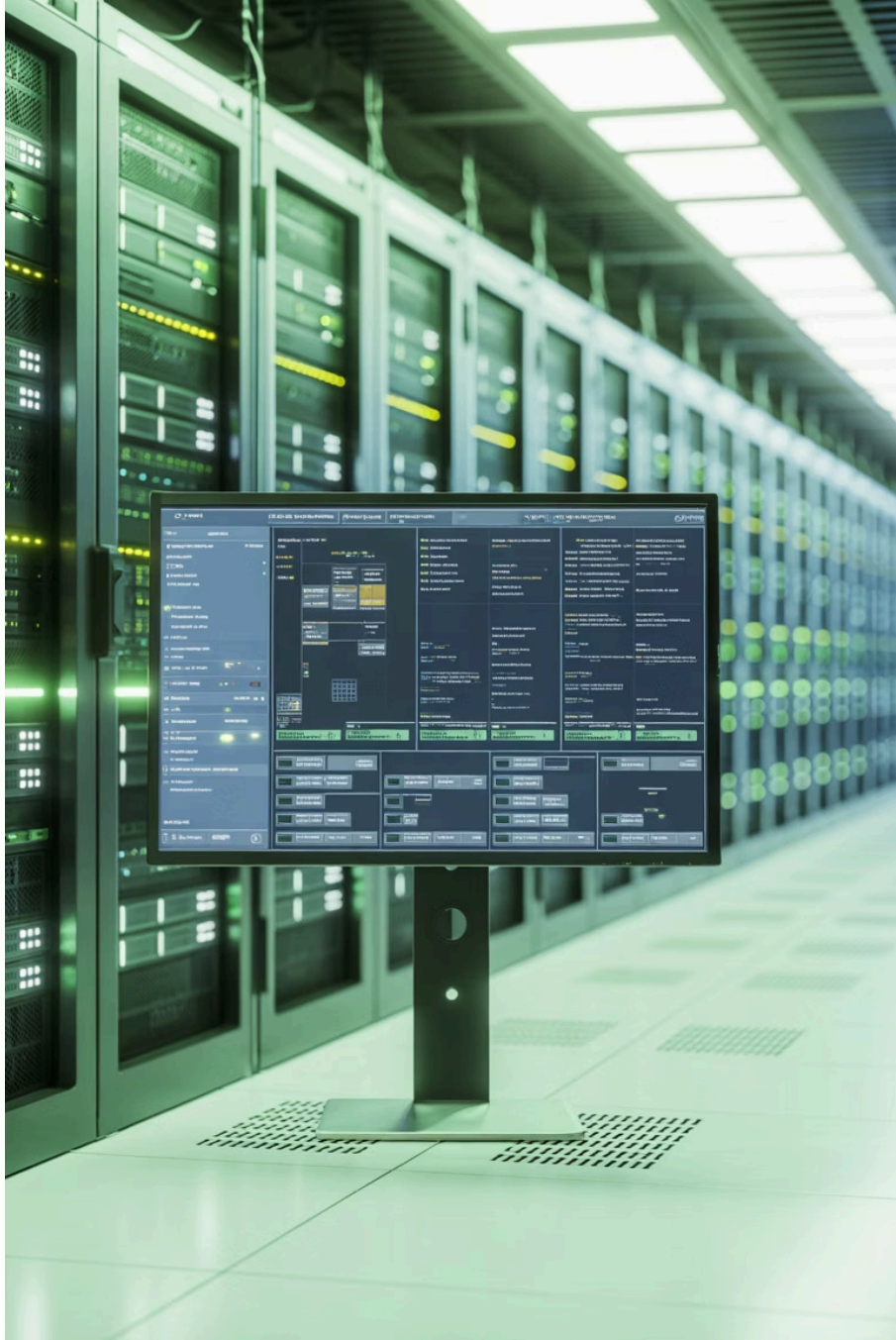
📞 **Contact:** 9246212143, 8885252627

🌐 **Website:** www.durgasoftonline.com

📩 **Email:** durgasoftonlinetraining@gmail.com

Day-43: What Is @ConfigurationProperties in Spring Boot?

Learn how to manage application configuration elegantly using type-safe property binding.



Type-Safe Configuration Management

The `@ConfigurationProperties` annotation allows you to **map an entire block of properties** from `application.properties` or `application.yml` into a strongly-typed Java class. This approach is significantly cleaner and more maintainable than using multiple `@Value` annotations scattered throughout your code.

This feature becomes extremely helpful in microservices architectures and cloud applications where configuration files tend to grow large and complex. It provides compile-time safety, auto-completion in IDEs, and easier refactoring.

YAML Configuration Example

application.yml

```
app:
  title: DURGASOFT Training
  version: 1.0
  server:
    url: https://durgasoftonline.com
    timeout: 5000
  features:
    - REST APIs
    - Microservices
    - Cloud Deployment
```

YAML format provides a clean, hierarchical structure for configuration data. It's more readable than properties files and supports complex nested structures naturally.

Java Configuration Class

```
@Component
@ConfigurationProperties(prefix = "app")
public class AppConfig {

 private String title;
 private String version;
 private Server server;
 private List<String> features;

 public static class Server {
 private String url;
 private int timeout;

 // getters and setters
 }

 // getters and setters
}
```

Benefits of @ConfigurationProperties

Type Safety

Configuration errors are caught at startup rather than runtime, preventing production issues.

Organization

Related properties are grouped together in a single class, making configuration management easier.

IDE Support

Get auto-completion and validation support in your IDE when working with configuration properties.

"Organized configuration leads to organized development."

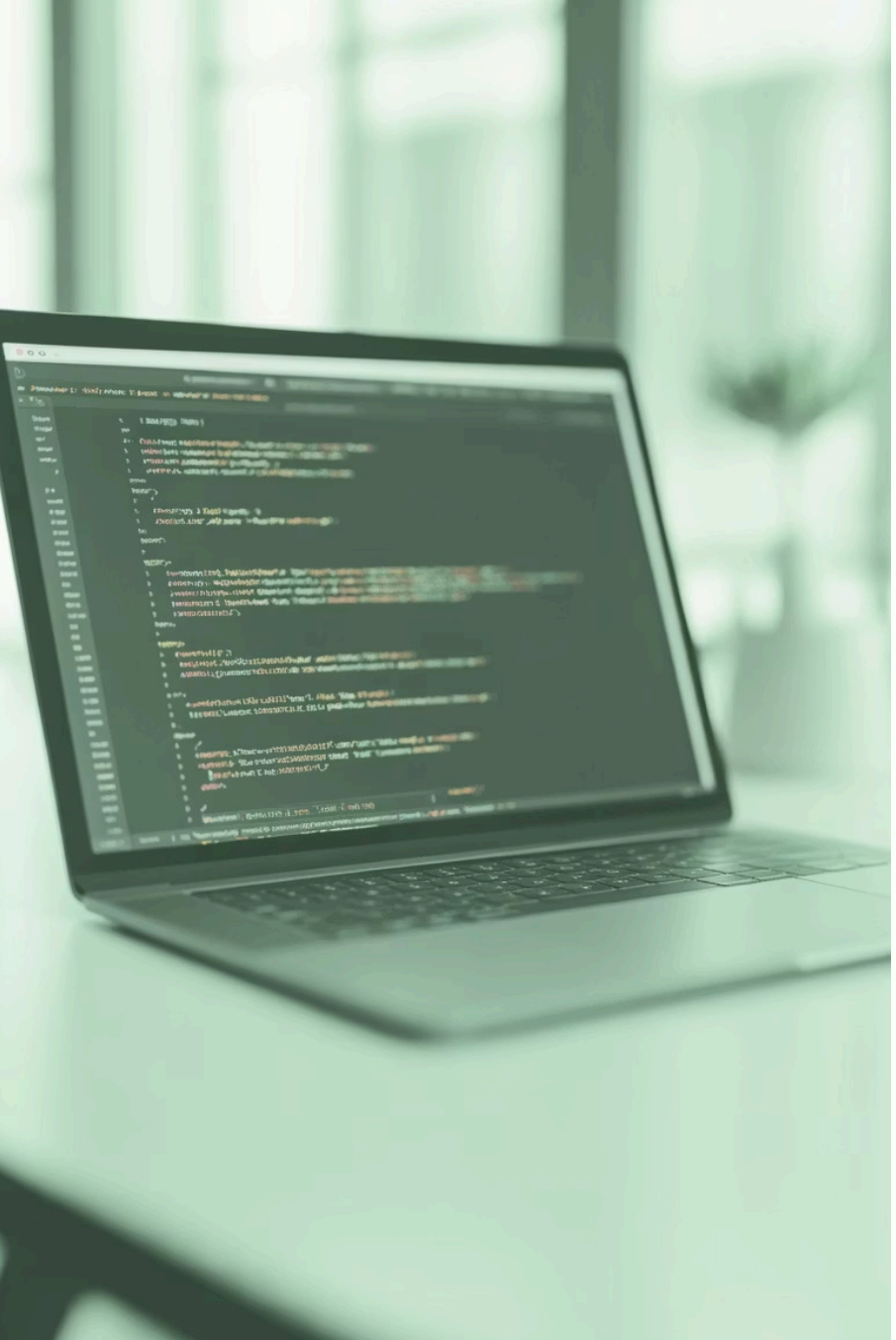
Join DURGASOFT for Expert Spring Boot Training

If you want comprehensive online training in SPRING Framework with SPRING BOOT and Cloud technologies, join DURGASOFT–India's most trusted training institute for Java and Spring ecosystem.

📞 **Contact:** 9246212143, 8885252627

🌐 **Website:** www.durgasoftonline.com

📩 **Email:** durgasoftonlinetraining@gmail.com



Day-44: What Is Thymeleaf in Spring Boot?

Discover the power of server-side template rendering for building dynamic web applications.

Modern Server-Side Template Engine

Thymeleaf is a modern, elegant server-side Java template engine used in Spring Boot to build dynamic HTML pages. It allows you to write natural, valid HTML templates and enhance them with special attributes like `th:text`, `th:each`, and `th:if` to bind data directly from your Spring controllers.

Thymeleaf is perfect for building admin dashboards, content management systems, internal tools, and simple web applications. It seamlessly integrates with Spring Boot and provides excellent support for internationalization, form handling, and layout management.

Thymeleaf in Action

Spring Controller

```
@Controller
public class WelcomeController {

    @GetMapping("/welcome")
    public String welcome(Model model) {
        model.addAttribute("name",
            "DurgaSoft Students");
        model.addAttribute("course",
            "Spring Boot");
        return "welcome";
    }
}
```

Thymeleaf Template

```
<html xmlns:th="http://www.thymeleaf.org">
<body>
 <h1 th:text="'Welcome, ' + ${name}"></h1>
 <p th:text="'You are learning ' + ${course}"></p>
</body>
</html>
```

Powerful Thymeleaf Features

Iteration

Use th:each to loop through collections and display lists of data dynamically.

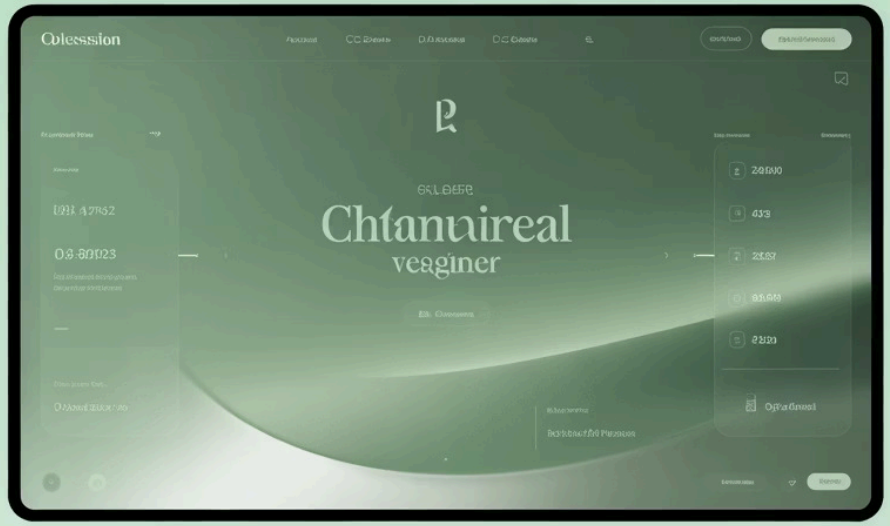
Conditionals

Apply th:if and th:unless to show or hide content based on conditions.

Form Binding

Bind form inputs to Java objects using th:object and th:field attributes.

"Beautiful UI with simple HTML—that's the power of Thymeleaf."



Join DURGASOFT for Expert Spring Boot Training

If you want comprehensive online training in SPRING Framework with SPRING BOOT and Cloud technologies, join DURGASOFT–India's most trusted training institute for Java and Spring ecosystem.

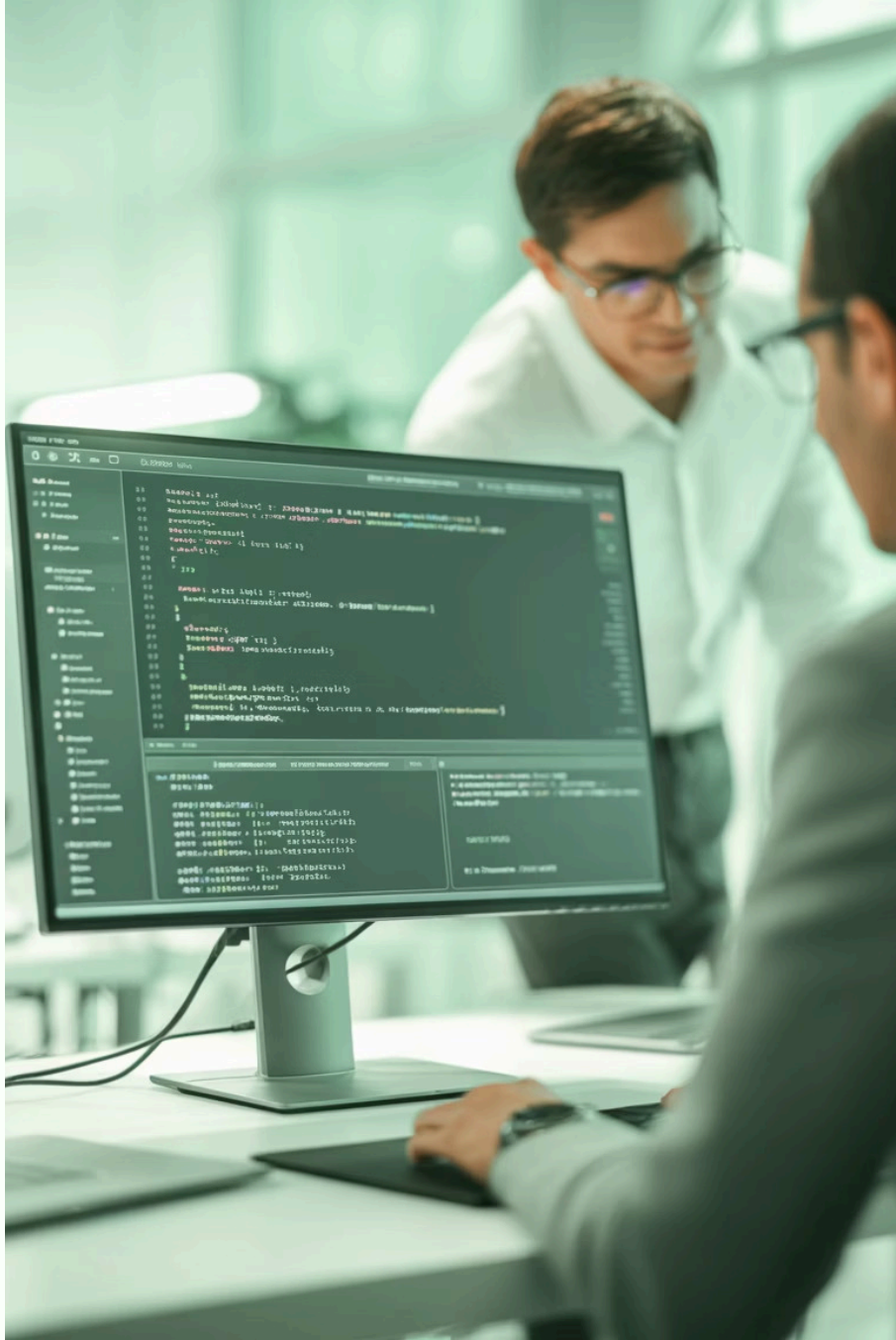
📞 **Contact:** 9246212143, 8885252627

🌐 **Website:** www.durgasoftonline.com

📩 **Email:** durgasoftonlinetraining@gmail.com

Day-45: What Is @RestController in Spring Boot?

Understanding the foundation of modern REST API development in Spring Boot.



The Backbone of REST APIs

`@RestController` is a specialized Spring annotation that combines **@Controller + @ResponseBody**. This powerful combination means that method return values are automatically written **directly into the HTTP response** as JSON, rather than being resolved as view names.

This annotation is the absolute backbone of REST API development in Spring Boot and is used extensively in mobile applications, microservices architectures, and cloud-based systems. It eliminates the need for view templates and focuses purely on data exchange.

Creating a REST API

```
@RestController
@RequestMapping("/api")
public class UserController {

 @GetMapping("/user")
 public User getUser() {
 return new User("Durga", "Hyderabad");
 }

 @GetMapping("/users")
 public List<User> getAllUsers() {
 return Arrays.asList(
 new User("Durga", "Hyderabad"),
 new User("Ravi", "Bangalore"),
 new User("Priya", "Chennai")
 );
 }
}
```

Automatic JSON Conversion

Spring Boot automatically converts Java objects to JSON using the **Jackson** library. You don't need to write any serialization code or configure JSON converters–everything works out of the box.

This makes REST API development incredibly fast and productive. You focus on business logic while Spring Boot handles all the technical details of HTTP communication and data serialization.

"JSON is the language of modern apps—master it through REST APIs."

Join DURGASOFT for Expert Spring Boot Training

If you want comprehensive online training in SPRING Framework with SPRING BOOT and Cloud technologies, join DURGASOFT–India's most trusted training institute for Java and Spring ecosystem.

📞 **Contact:** 9246212143, 8885252627

🌐 **Website:** www.durgasoftonline.com

📩 **Email:** durgasoftonlinetraining@gmail.com

Day-46: What Is @PathVariable in Spring Boot REST APIs?

Learn how to capture dynamic values from URLs for building flexible REST APIs.

Dynamic URL Path Handling

`@PathVariable` enables you to capture dynamic values directly from the URL path. This annotation is commonly used for fetching specific records using identifiers like userId, productId, orderId, transactionId, and more.

Using path variables makes your APIs cleaner, more RESTful, and more user-friendly. Instead of using query parameters for identifiers, you embed them directly in the URL path, following REST best practices and making your API endpoints more intuitive and readable.

Basic PathVariable Example

```
@RestController
@RequestMapping("/api")
public class UserController {

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

 @GetMapping("/user/{id}/orders/{orderId}")
 public String getUserOrder(
 @PathVariable int id,
 @PathVariable int orderId) {
 return "User: " + id + ", Order: " + orderId;
 }
}
```

PathVariable Use Cases

User Management

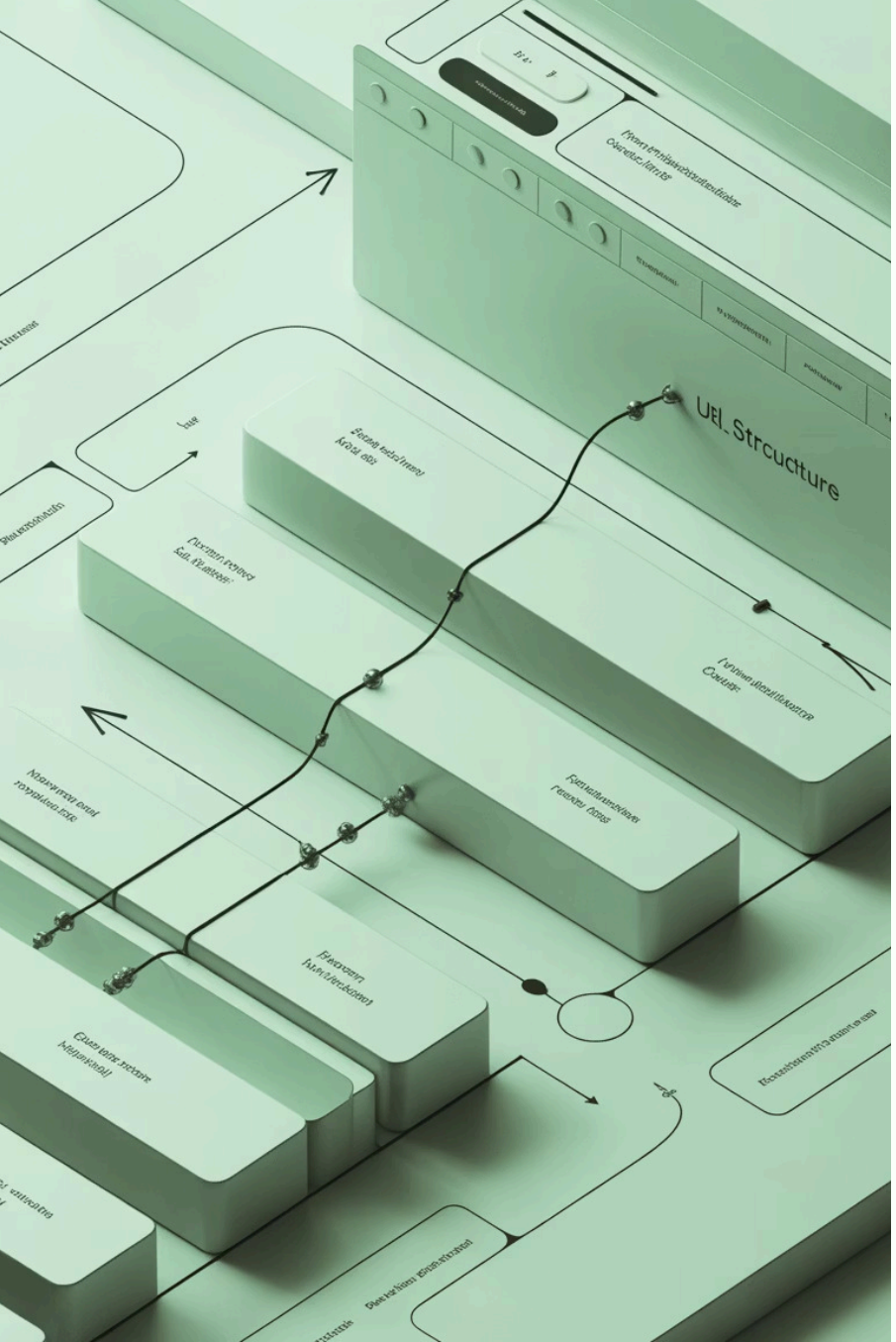
`/api/users/123` - Fetch user with ID 123

E-commerce

`/api/products/456/reviews` - Get reviews for product 456

Document Management

`/api/documents/789/download` - Download document 789



"Smart URL design creates smart APIs—keep them clean and meaningful."

Join DURGASOFT for Expert Spring Boot Training

If you want comprehensive online training in SPRING Framework with SPRING BOOT and Cloud technologies, join DURGASOFT–India's most trusted training institute for Java and Spring ecosystem.

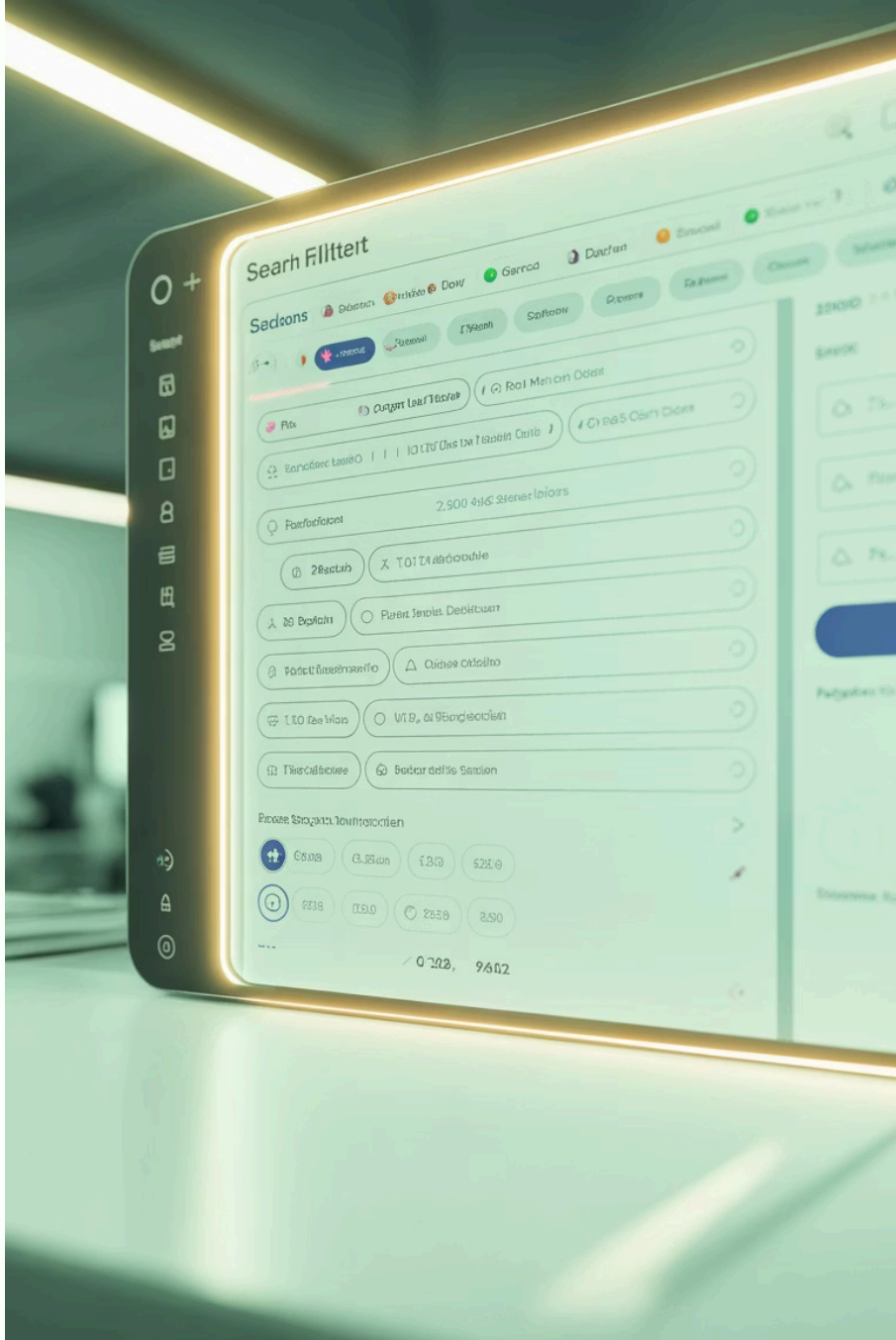
📞 **Contact:** 9246212143, 8885252627

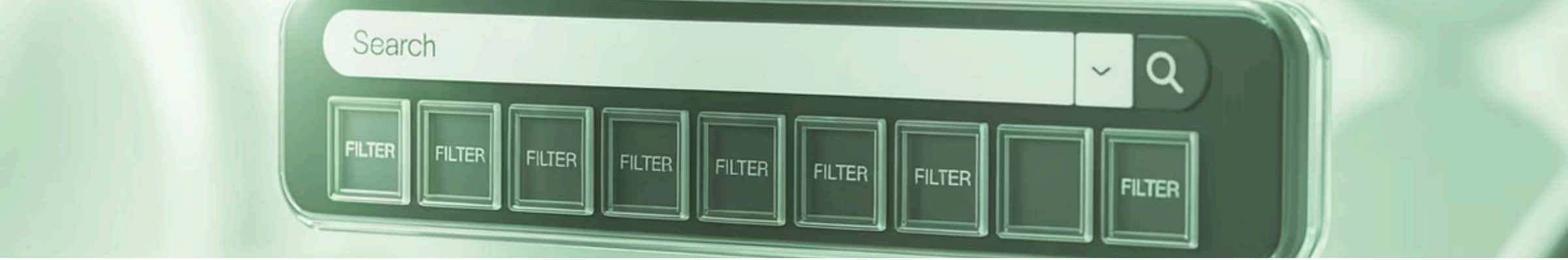
🌐 **Website:** www.durgasoftonline.com

📩 **Email:** durgasoftonlinetraining@gmail.com

Day-47: What Is @RequestParam in Spring Boot?

Master the art of handling query parameters for filtering, searching, and pagination.





Query Parameter Handling

`@RequestParam` is used to read **query parameters** from the URL. This annotation is incredibly helpful for implementing filters, search functionality, pagination, sorting options, and optional inputs in your REST APIs.

When a parameter is missing from the request, you can gracefully handle it by setting a default value using the `defaultValue` attribute. This makes your APIs more robust and user-friendly, preventing errors when clients don't provide all expected parameters.

RequestParam Examples

```
@RestController
@RequestMapping("/api")
public class SearchController {

 @GetMapping("/search")
 public String search(
 @RequestParam String keyword,
 @RequestParam(defaultValue = "1") int page,
 @RequestParam(defaultValue = "10") int size,
 @RequestParam(required = false) String sortBy) {

 return "Searching for: " + keyword +
 ", Page: " + page +
 ", Size: " + size +
 ", Sort: " + sortBy;
 }
}
```

Query Parameter Patterns

1

Search

`/search?keyword=spring&category=backend`

2

Pagination

`/products?page=2&size=20`

3

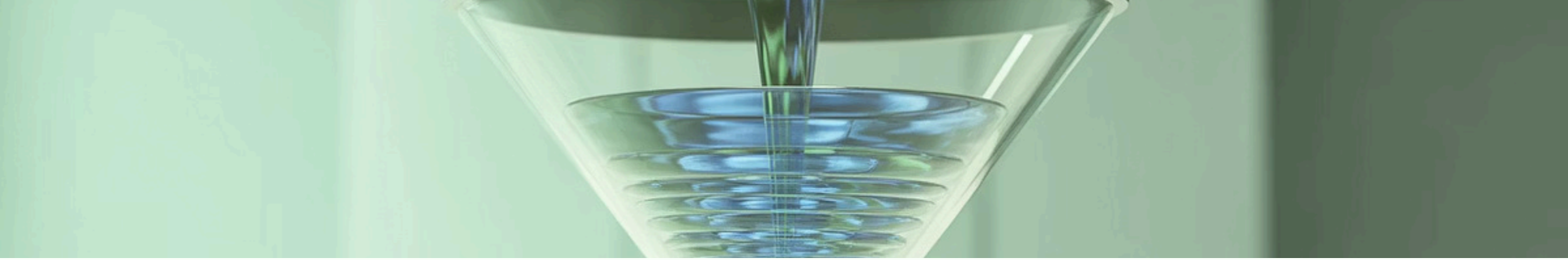
Filtering

`/users?status=active&role=admin`

4

Sorting

`/orders?sortBy=date&order=desc`



"Small parameters create big possibilities in your APIs."

Join DURGASOFT for Expert Spring Boot Training

If you want comprehensive online training in SPRING Framework with SPRING BOOT and Cloud technologies, join DURGASOFT–India's most trusted training institute for Java and Spring ecosystem.

📞 **Contact:** 9246212143, 8885252627

🌐 **Website:** www.durgasoftonline.com

📩 **Email:** durgasoftonlinetraining@gmail.com

Day-48: What Is @RequestBody in Spring Boot?

Understanding how to receive and process JSON data from client applications.

Receiving JSON Data

`@RequestBody` is used to read **JSON data** sent from client applications (Postman, Angular, React, Vue, Mobile Apps, etc.) and automatically convert it into a Java object. Spring Boot uses the Jackson library to perform this conversion seamlessly.

This makes REST API development incredibly efficient. You don't need to manually parse JSON strings or write complex deserialization code. Spring Boot handles all the technical details, allowing you to focus on business logic and application functionality.

RequestBody in Action

JSON Input

```
{
  "name": "Durga",
  "city": "Hyderabad",
  "email": "durga@example.com",
  "phone": "9876543210"
}
```

Controller Method

```
@PostMapping("/saveUser")
public ResponseEntity<String> saveUser(
 @RequestBody User user) {

 // Save to database
 userService.save(user);

 return ResponseEntity.ok(
 "User saved: " + user.getName()
 );
}
```

Common Use Cases

User Registration

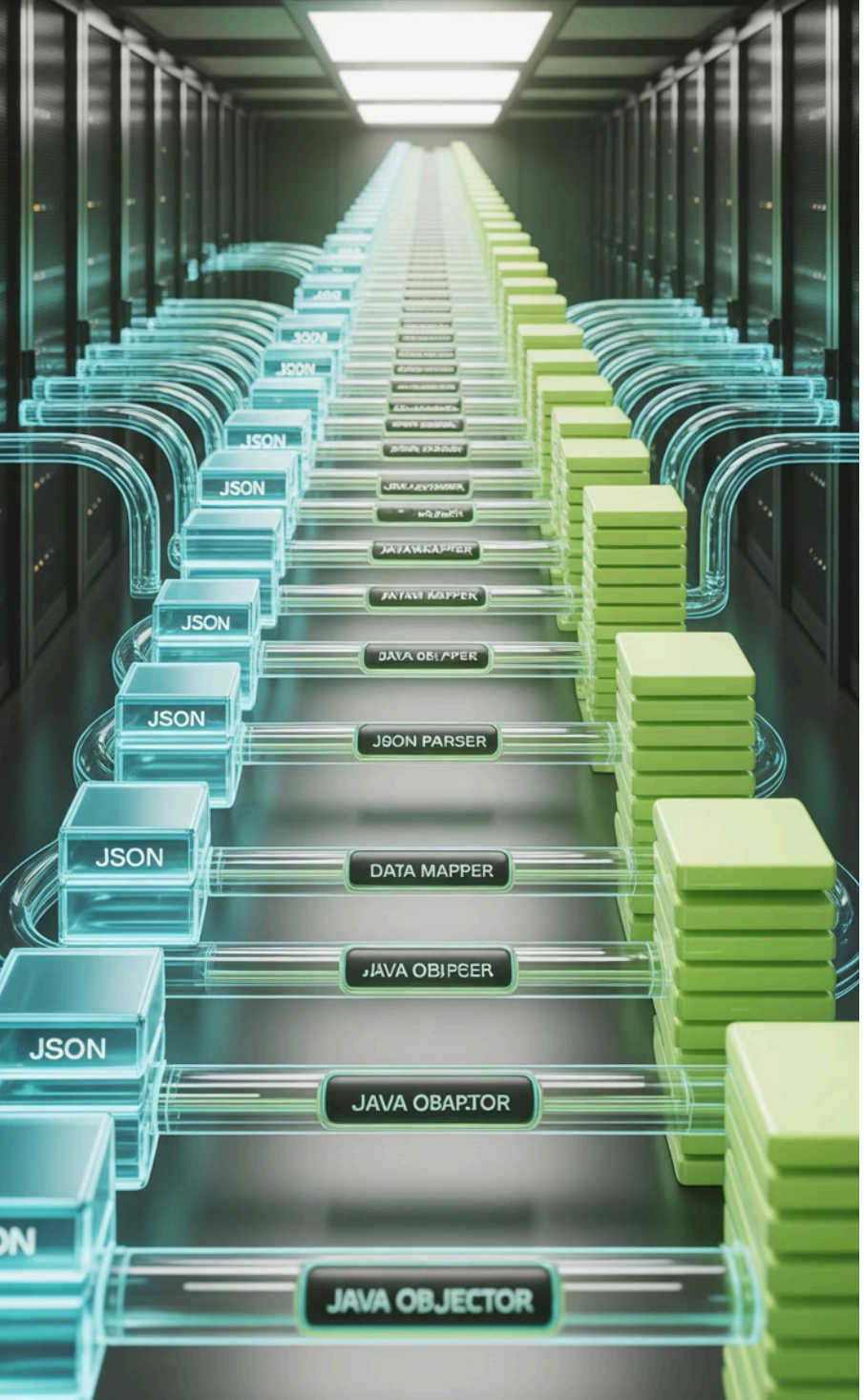
Accept user registration data including personal details, credentials, and preferences.

Form Submissions

Process complex form data with nested objects and multiple fields from web applications.

API Integration

Receive data from external services and third-party APIs in JSON format.



"JSON in, Java object out—that's the power of modern APIs."