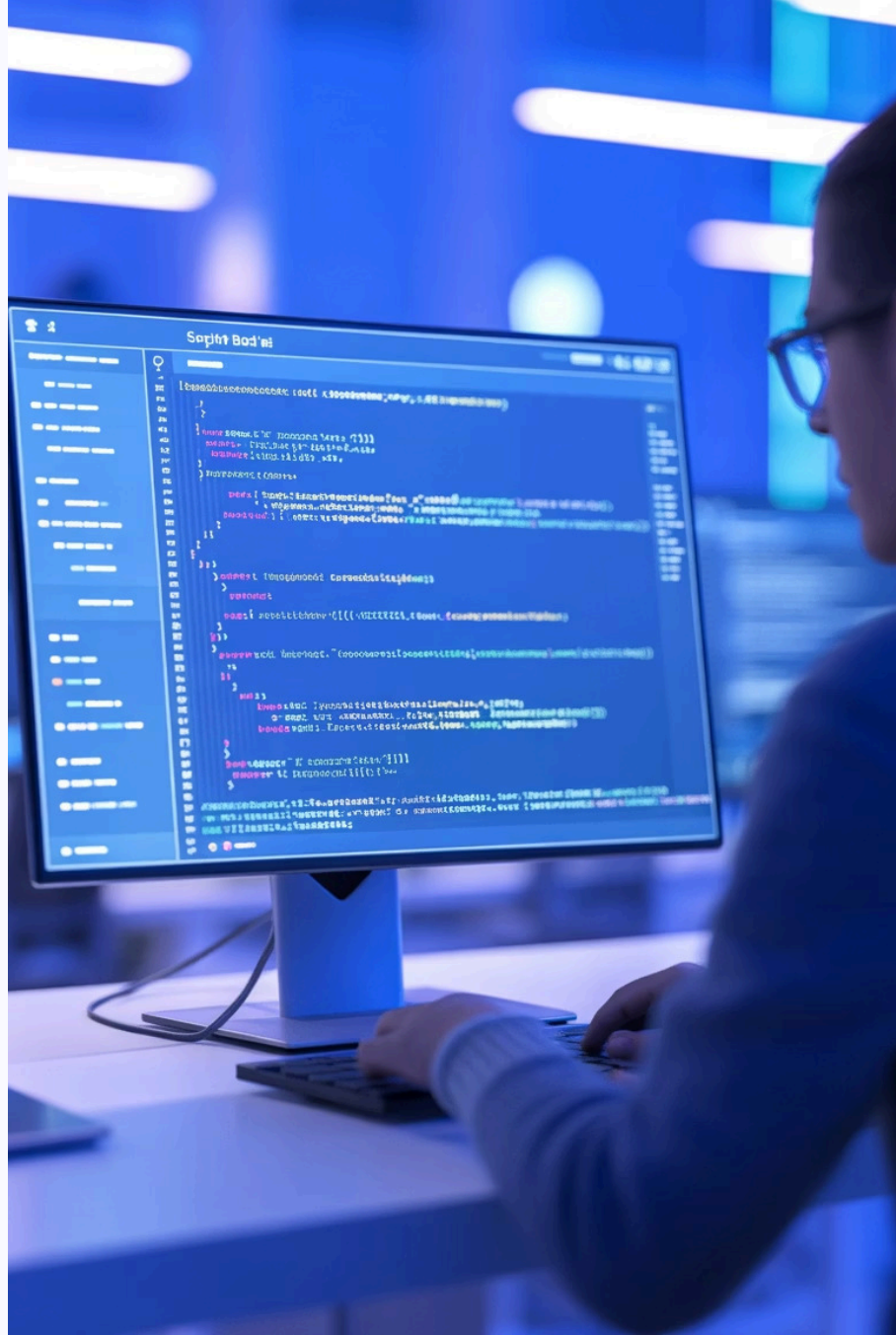
Day-61: What Is @RestController in Spring Boot?

Learn how to build powerful REST APIs with Spring Boot's most essential annotation for web development.



Understanding @RestController

What Is It?

@RestController is a specialized annotation that combines @Controller and @ResponseBody. It tells Spring Boot that your class will return data directly in JSON or text format, rather than rendering JSP or HTML pages.

Why It Matters

This annotation is the foundation for building REST APIs in Spring Boot. It simplifies your code by eliminating the need to add @ResponseBody on every method, making API development faster and cleaner.

@RestController in Action

No @ResponseBody Needed

Every method automatically returns JSON without extra annotations

Perfect for APIs

Built specifically for RESTful web service development

Auto Conversion

Automatically converts Java objects to JSON format

Code Example

```
@RestController
public class WelcomeController {

    @GetMapping("/hello")
    public String sayHello() {
        return "Welcome to Spring Boot!";
    }
}
```

This simple controller creates a REST endpoint at `/hello` that returns a welcome message. Notice how clean and readable the code is - no complex configuration needed!

Where @RestController Shines

Microservices

Build scalable, independent services that communicate via REST APIs

Mobile App APIs

Create robust backends for Android and iOS applications

Cloud Services

Develop cloud-native applications with RESTful interfaces

"When your backend speaks JSON, the world of APIs becomes easy."

Join DURGASOFT for Complete Training

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

📞 **Contact:** 9246212143, 8885252627

🌐 **www.durgasoftonline.com**

📩 durgasoftonlinetraining@gmail.com

Day-62: What Is @Service in Spring Boot?

Discover how to organize your application's business logic with clean, maintainable code using Spring Boot's @Service annotation.

The Business Logic Layer

@Service is used to mark a class as the **business logic layer** in a Spring application. It sits between the Controller and Repository layers, acting as the brain of your application where all business rules and calculations happen.

This annotation helps you write clean, modular code by separating business rules from API endpoints and database operations. Following this pattern makes your code easier to test, maintain, and scale.

How @Service Works

Controller

Receives HTTP requests from clients

Service

Processes business logic and rules

Repository

Handles database operations

Database

Stores application data

@Service Example

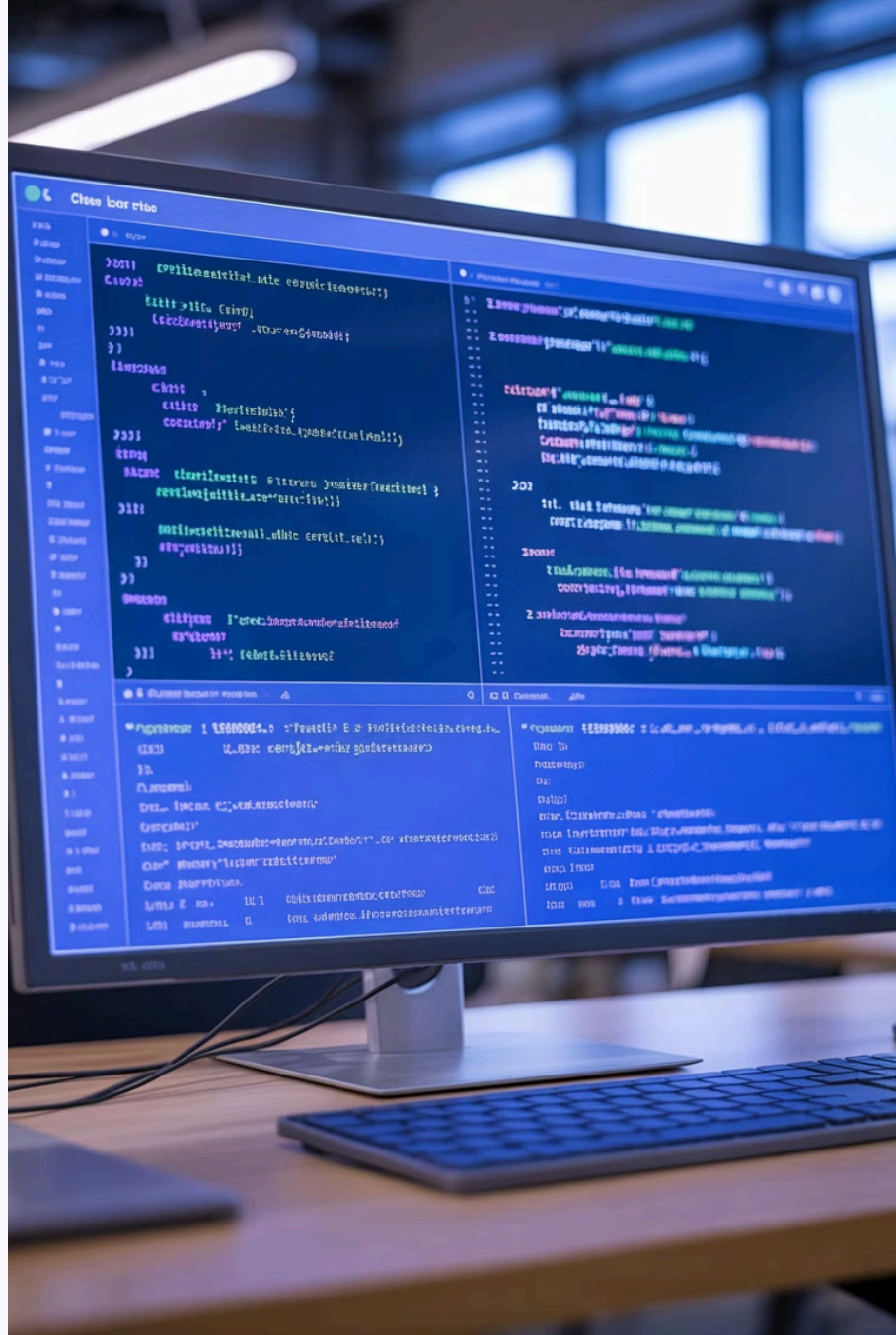
```
@Service
public class UserService {

    public double calculateDiscount(
        double amount) {
        return amount * 0.10;
    }
}
```

Key Benefits

- Keeps your business logic organized in one place
- Improves code reusability across controllers
- Helps with transaction management
- Makes testing easier with mock services
- Keeps Controller clean and focused on HTTP handling

"Clean business logic is the heart of every strong application."



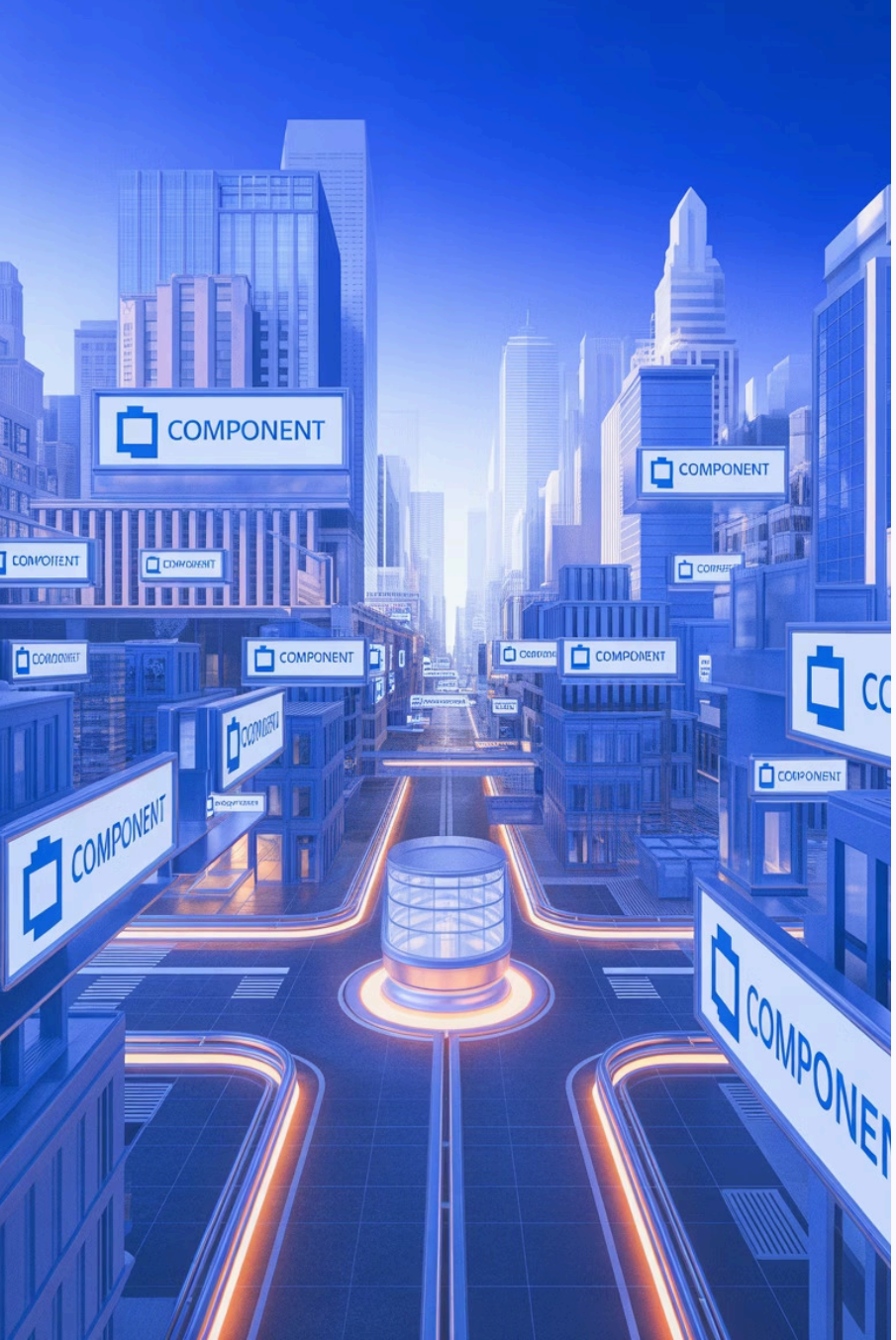
Join DURGASOFT for Complete Training

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

📞 **Contact:** 9246212143, 8885252627

🌐 **www.durgasoftonline.com**

📩 durgasoftonlinetraining@gmail.com



Day-63: What Is @Component in Spring Boot?

Learn about the most fundamental Spring annotation that enables automatic bean management and dependency injection.

Understanding @Component

@Component is the **most basic Spring annotation** used to mark a class as a Spring-managed bean. When you add this annotation to a class, Spring automatically detects it during component scanning and creates an object for you.

It's incredibly useful for utility classes, helpers, custom logic, or any general-purpose bean that doesn't fit into more specific categories like Service or Repository.

@Component in Action

```
@Component
public class AppLogger {

    public void log(String message) {
        System.out.println("LOG: " + message);
    }
}
```

With this simple annotation, Spring automatically creates an AppLogger object at startup. You can then inject it anywhere in your application using @Autowired - no need to manually create objects with the `new` keyword!

Why Use @Component?

Automatic Object Creation

Spring creates the object for you - no need to use `new` keyword

Dependency Injection

Enables seamless injection into other components

Base Annotation

Foundation for @Service, @Repository, and @Controller

Utility Classes

Perfect for helper classes and common functionality

"Even the smallest component can make a big impact in your application."

Join DURGASOFT for Complete Training

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

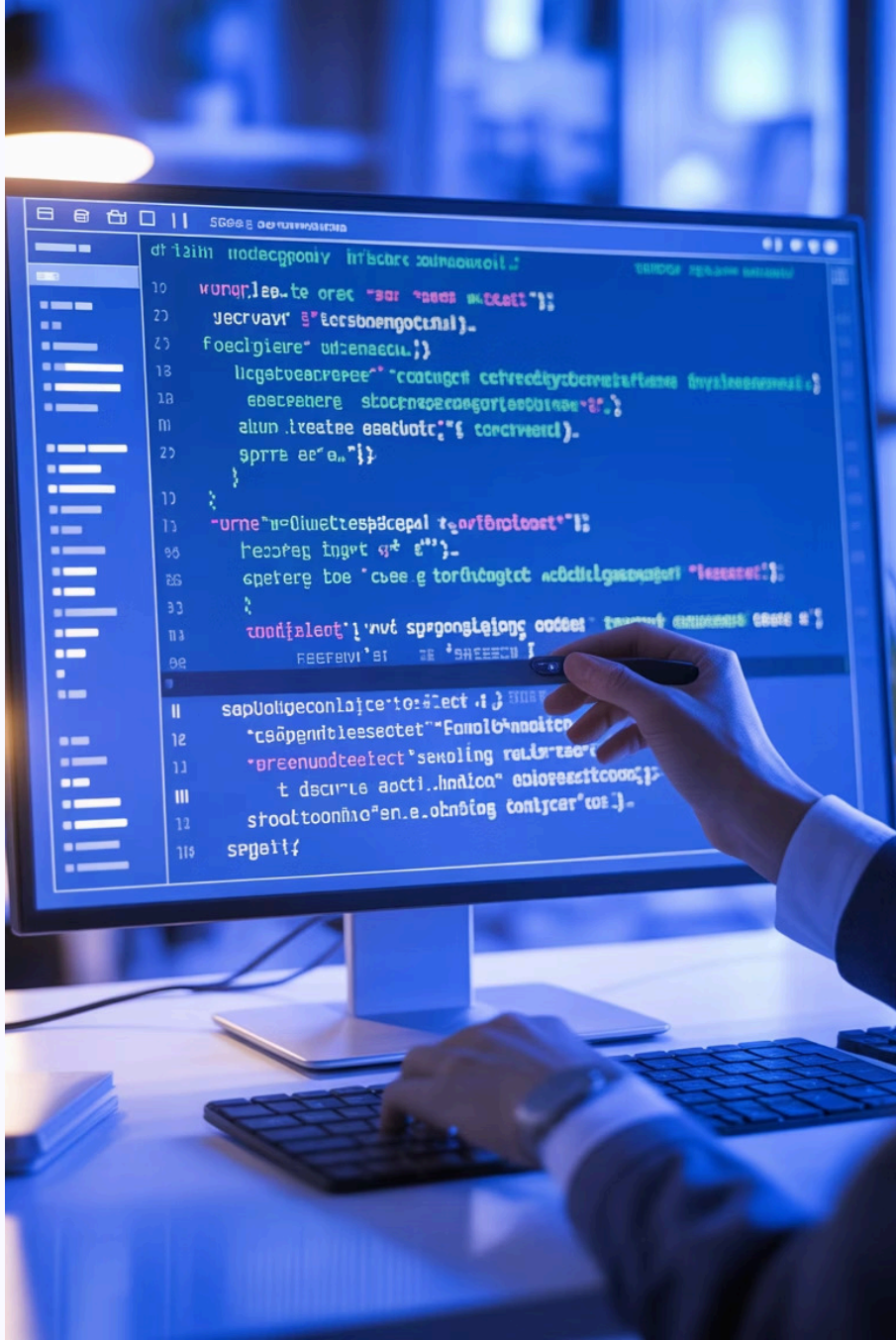
📞 **Contact:** 9246212143, 8885252627

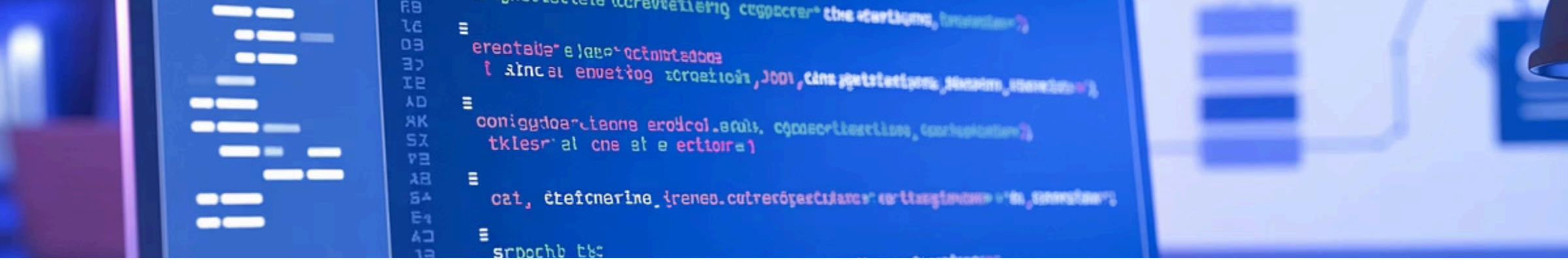
🌐 **www.durgasoftonline.com**

📩 durgasoftonlinetraining@gmail.com

Day-64: What Is @Configuration in Spring Boot?

Move from old XML configuration to modern Java-based configuration with Spring Boot's powerful @Configuration annotation.





Configuration Made Simple

@Configuration tells Spring that a class contains **bean definitions**. It's a game-changer that replaces old XML configuration files, allowing you to define beans using clean Java code instead.

This annotation works closely with @Bean methods to create objects that Spring manages throughout your application's lifecycle. It's the modern way to configure Spring applications.

@Configuration Example

```
@Configuration
public class AppConfig {

    @Bean
    public String appMessage() {
        return "Hello from Spring!";
    }
}
```

Key Advantages

- Replaces XML <beans> configuration
- Creates reusable beans easily
- Provides full control over object creation
- Type-safe configuration
- Better IDE support and refactoring

Common Use Cases

Security Configuration

Define authentication and authorization rules for your application using Spring Security beans.

Database Configuration

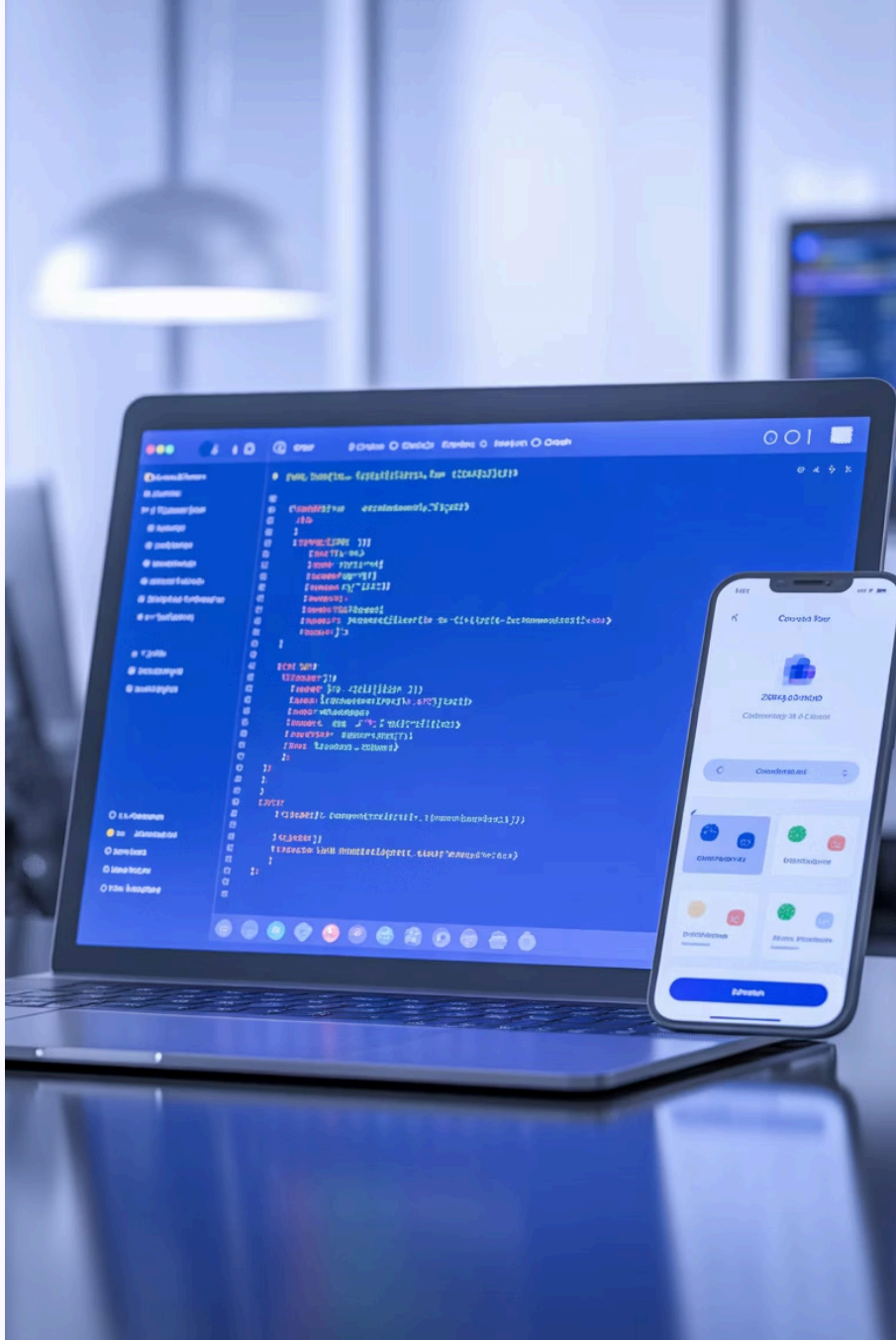
Set up data sources, connection pools, and transaction managers for database operations.

Custom Beans

Create and configure third-party library objects or custom components with specific settings.

Spring automatically loads all @Configuration classes during application startup, making your beans available for dependency injection throughout your application.

"Clean configuration builds clean applications — let Spring manage the setup."



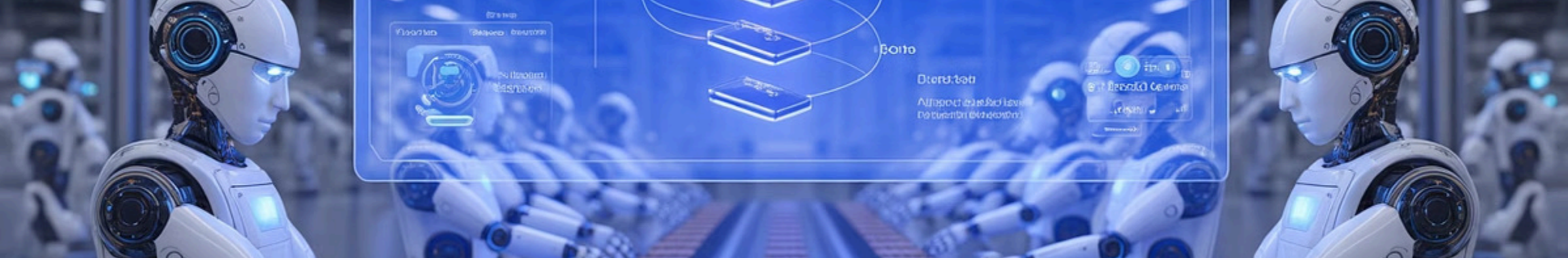
Join DURGASOFT for Complete Training

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

📞 **Contact:** 9246212143, 8885252627

🌐 **www.durgasoftonline.com**

📩 durgasoftonlinetraining@gmail.com



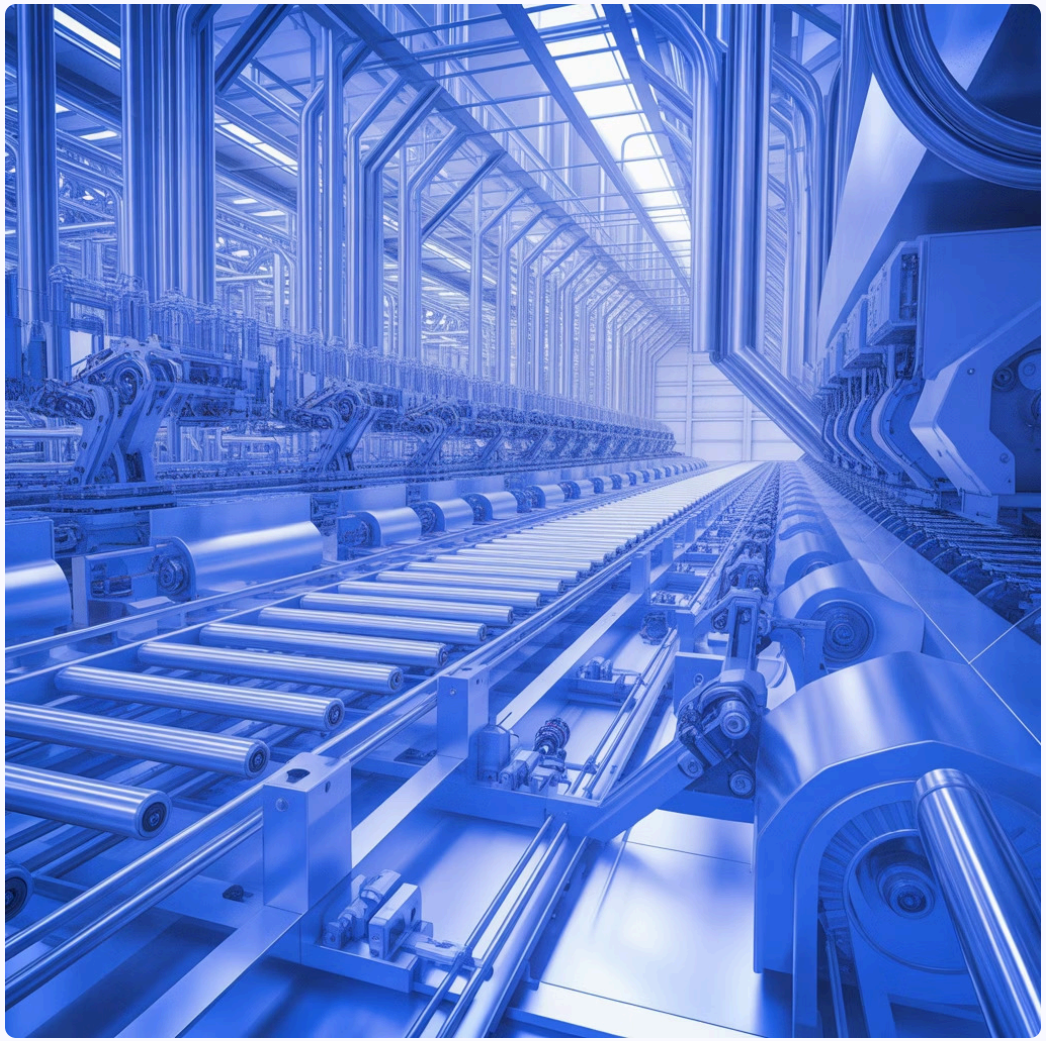
Day-65: What Is @Bean in Spring Boot?

Master the art of creating custom-configured Spring beans with full control over initialization and dependencies.

The Power of @Bean

@Bean is used inside a @Configuration class to tell Spring to create and manage an object. Unlike component scanning, @Bean gives you **full control** over object creation, custom settings, and initialization logic.

It's perfect for creating beans from third-party library classes that you cannot annotate with @Component, or when you need special configuration during object creation.



@Bean in Action

```
@Configuration
public class AppConfig {

    @Bean
    public PasswordEncoder encoder() {
        return new BCryptPasswordEncoder();
    }
}
```

In this example, we're creating a BCryptPasswordEncoder bean for password security. Spring creates this instance once at startup and injects it wherever needed throughout your application.

Why @Bean Is Powerful

Third-Party Classes

Create beans from library classes you don't own

Custom Creation

Full control over how objects are initialized

Replaces XML

Modern alternative to XML <bean> tags

Complex Configuration

Set up beans with multiple dependencies easily

Smooth Injection

Works perfectly with dependency injection

"When you control the beans, you control the application."

Join DURGASOFT for Complete Training

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

📞 **Contact:** 9246212143, 8885252627

🌐 **www.durgasoftonline.com**

📩 durgasoftonlinetraining@gmail.com

Day-66: What Is @Autowired in Spring Boot?

Learn how Spring Boot automatically wires your dependencies together, eliminating boilerplate code and manual object creation.

Automatic Dependency Injection

@Autowired is used for **automatic dependency injection**. When you mark a field, constructor, or setter with @Autowired, Spring automatically creates the required object and injects it into your class without you needing to use the `new` keyword.

This keeps your code clean, reduces manual wiring, and makes your application loosely coupled - a key principle of good software design. Your classes depend on abstractions, not concrete implementations.

@Autowired Example

```
@Service
public class OrderService {

    @Autowired
    private PaymentService paymentService;

    public void processOrder() {
        paymentService.pay();
    }
}
```

What Happens Here?

Spring automatically finds the PaymentService bean and injects it into OrderService. No manual object creation needed!

Result

Your code is cleaner, more maintainable, and easier to test with mock objects.

Three Ways to Use @Autowired

01

Field Injection

Direct injection into class fields - quick and simple

```
@Autowired
private UserService userService;
```

02

Constructor Injection

Most recommended - makes dependencies explicit and supports immutability

```
@Autowired
public OrderService(PaymentService
payment) {
    this.payment = payment;
}
```

03

Setter Injection

Injection through setter methods - useful for optional dependencies

```
@Autowired
public void setUserService(UserService
service) {
 this.userService = service;
}
```

Key Benefits

No Manual Creation

Spring handles all object creation automatically

Lifecycle Management

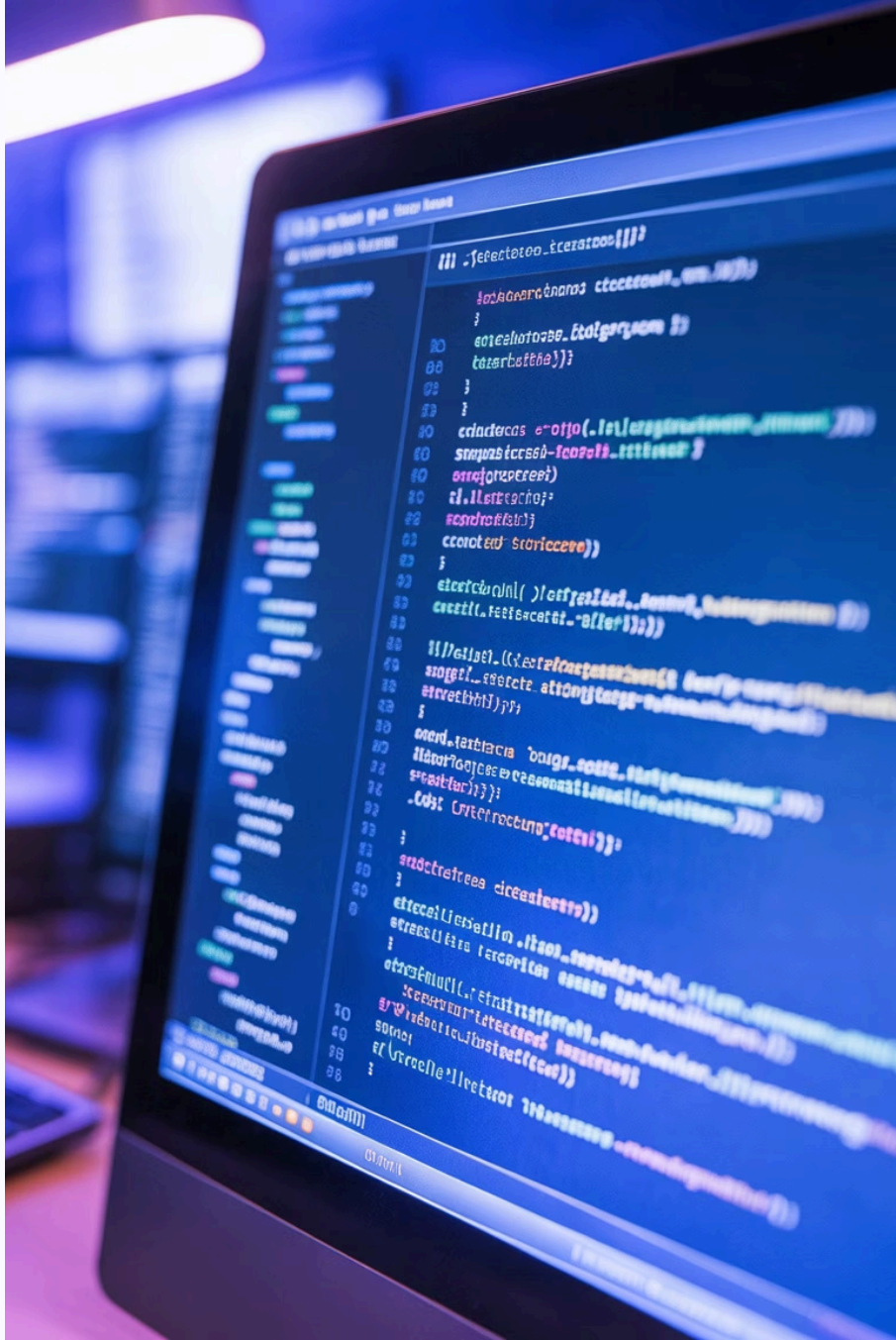
Spring manages the complete bean lifecycle

Less Boilerplate

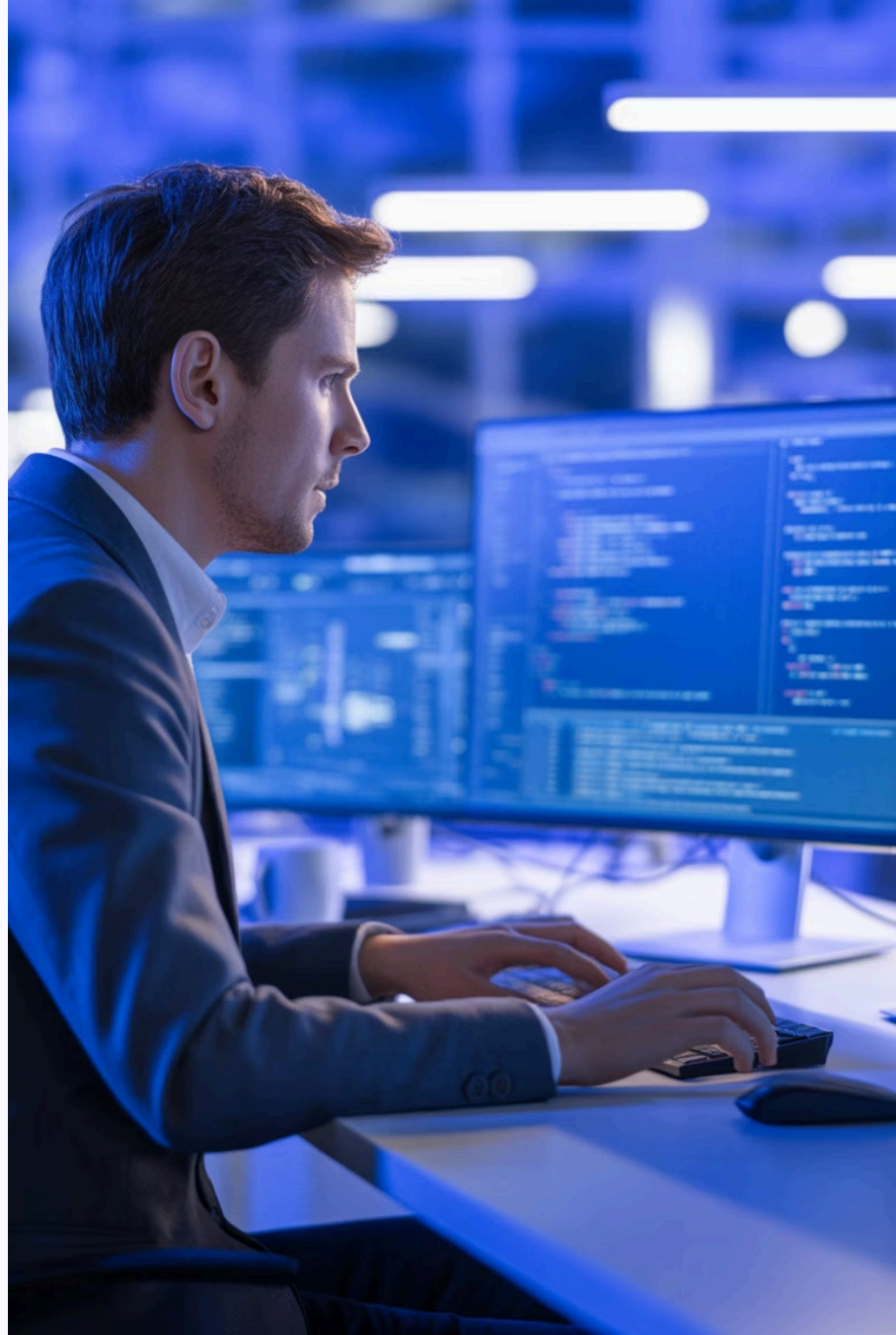
Write less code, focus on business logic

Easy Testing

Inject mock objects during unit testing



"Let Spring do the wiring — you focus on the logic."



Join DURGASOFT for Complete Training

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

📞 **Contact:** 9246212143, 8885252627

🌐 **www.durgasoftonline.com**

📩 durgasoftonlinetraining@gmail.com

Day-67: What Is @Qualifier in Spring Boot?

Resolve ambiguity when multiple beans of the same type exist - tell Spring exactly which bean you want to inject.

Solving the Multiple Bean Problem

@Qualifier is used when multiple beans of the same type exist and Spring doesn't know which one to inject. Without @Qualifier, Spring throws an error because it cannot decide which bean to use.

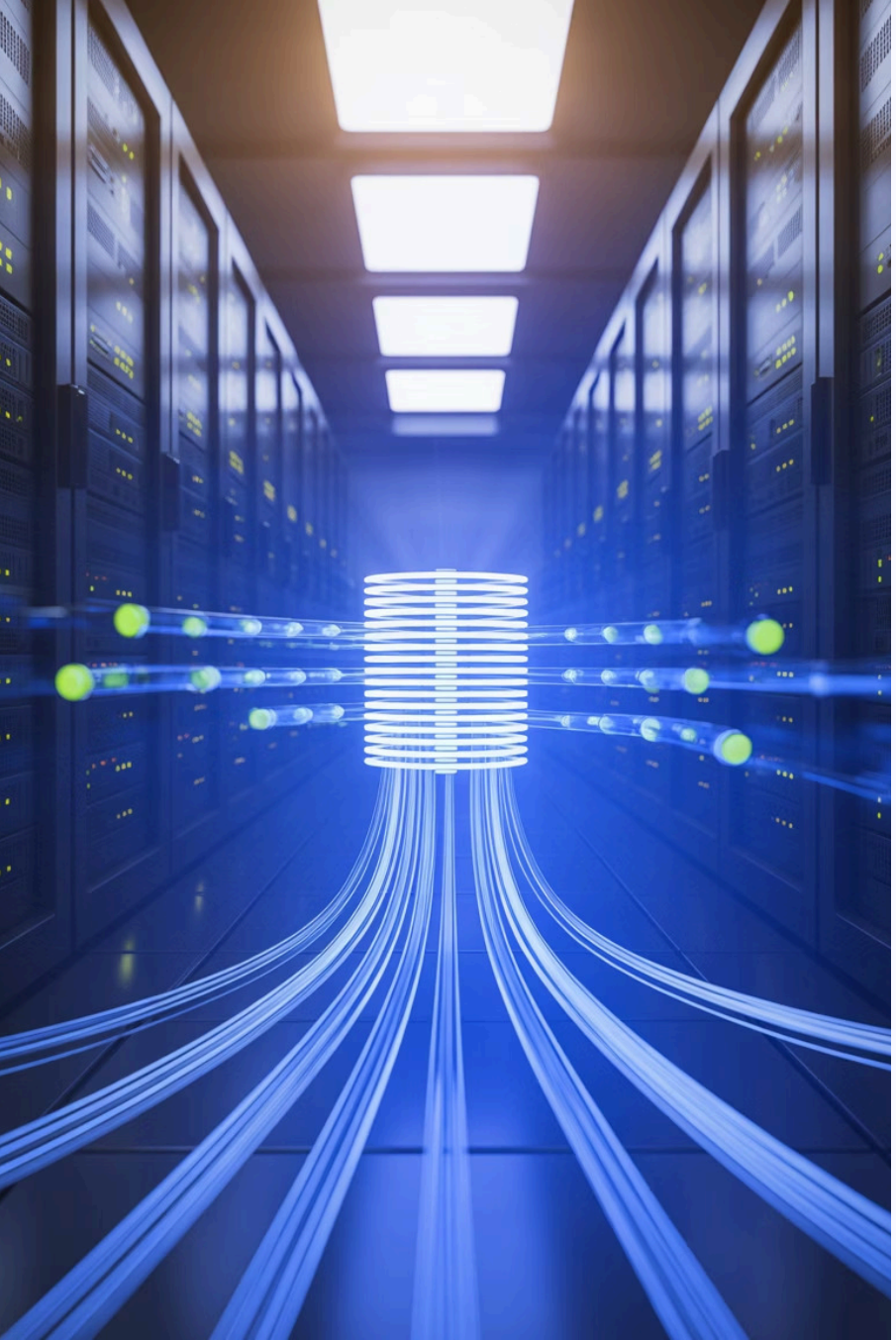
With @Qualifier, you can tell Spring **exactly which bean** should be used by specifying its name. This avoids confusion and ensures proper dependency injection in complex applications.

The Problem: Multiple Implementations

```
@Component("paypal")
public class PaypalPayment implements Payment {
    public void pay() {
        System.out.println("Paying via PayPal");
    }
}

@Component("stripe")
public class StripePayment implements Payment {
    public void pay() {
        System.out.println("Paying via Stripe");
    }
}
```

Here we have two Payment implementations. If you try to inject Payment without @Qualifier, Spring will throw an error because it doesn't know which one to choose!



The Solution: Using @Qualifier

```
@Service
public class OrderService {

    @Autowired
    @Qualifier("stripe")
    private Payment payment;

    public void checkout() {
        payment.pay();
    }
}
```

What Happens?

Spring injects the **StripePayment** bean into OrderService, not PaypalPayment.

Output

When `checkout()` is called, you'll see: "Paying via Stripe"

When to Use @Qualifier

Multiple Implementations

When you have several classes implementing the same interface and need to choose one specific implementation

Avoiding Injection Conflicts

Prevent Spring from throwing "no unique bean" errors when multiple candidates exist

Clear Bean Selection

Make your code more explicit and readable by clearly stating which bean you're using

"When choices create confusion, qualifiers create clarity."

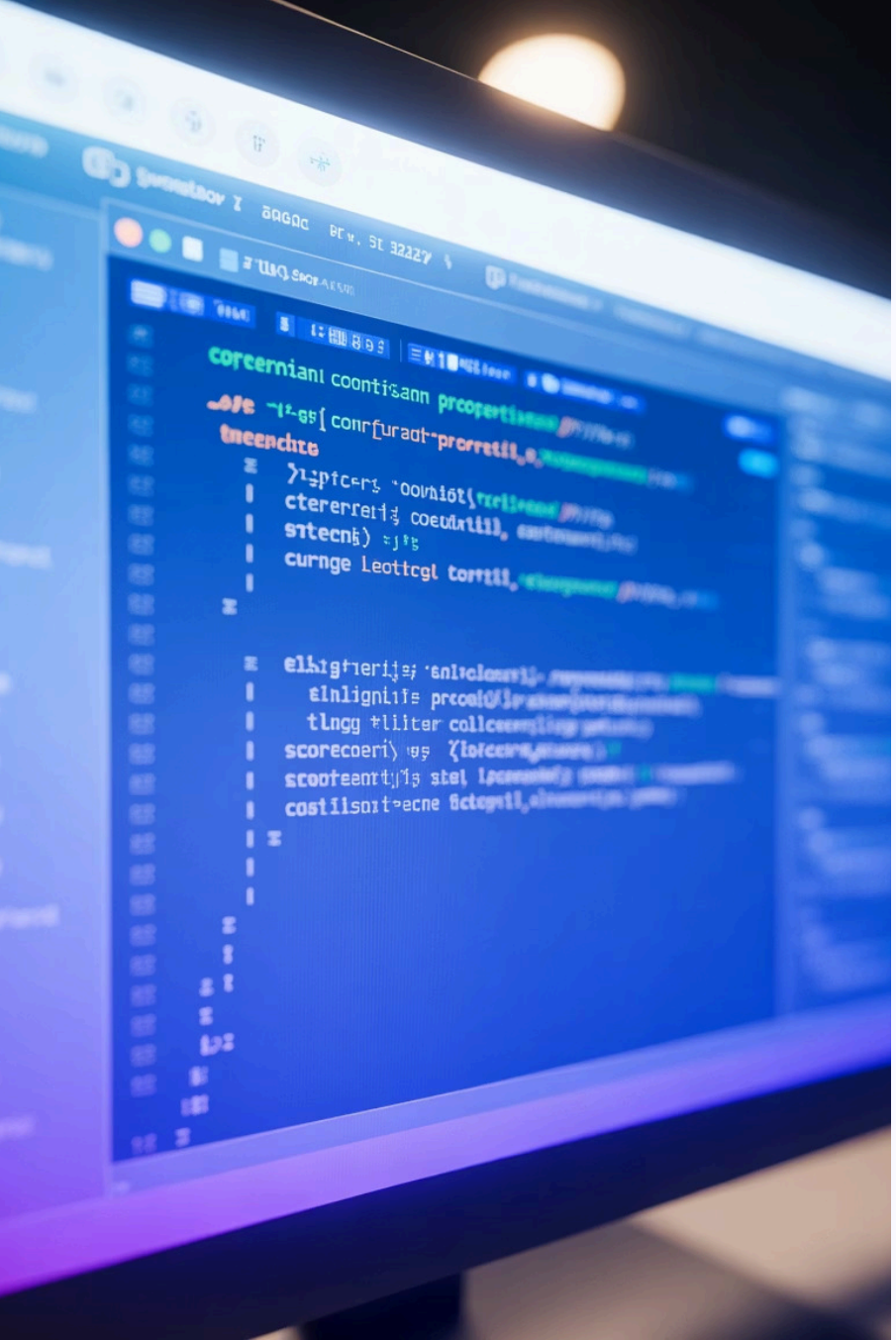
Join DURGASOFT for Complete Training

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

📞 **Contact:** 9246212143, 8885252627

🌐 **www.durgasoftonline.com**

📩 durgasoftonlinetraining@gmail.com



Day-68: What Is @Value in Spring Boot?

Keep your application flexible by externalizing configuration values - learn to inject properties from files, YAML, and environment variables.

Externalizing Configuration

@Value is used to inject values from `application.properties`, **YAML files**, or even **environment variables** directly into your fields. It's perfect for reading configuration values like database URLs, API keys, usernames, secrets, limits, or default values.

This approach helps you avoid hardcoding values in your code, making your application more flexible and easier to configure across different environments without recompiling.

@Value Example

In application.properties

```
app.welcome=Hello Spring Boot Learners!
app.version=2.1.5
app.max.users=1000
```

In Java Class

```
@Component
public class AppInfo {

    @Value("${app.welcome}")
    private String message;

    @Value("${app.version}")
    private String version;
}
```

Spring automatically reads values from your properties file and injects them into your fields at runtime.

Advanced @Value Features

1

Default Values

Provide fallback values if property is not found

```
@Value("${app.mode:DEV}")
private String mode;
```

2

Type Conversion

Automatically converts strings to proper types

```
@Value("${app.max.users}")
private int maxUsers;
```

3

SpEL Expressions

Use Spring Expression Language for complex values

```
@Value("#{systemProperties['user.home']}")
private String homeDir;
```

Benefits of @Value

No Hardcoding

Keep configuration separate from code

External Files

Read from properties, YAML, or environment

Microservices Ready

Perfect for cloud-native configuration

Environment Specific

Different values for dev, test, and production

"Move your configuration out of code — keep your application flexible."