Spring Boot MVC with Dependency Injection

1. Presentation Layer (Controller)

Handles HTTP requests and sends responses.

```
@RestController
@RequestMapping("/products")
public class ProductController {

    private final ProductService productService;

    public ProductController(ProductService productService) {
        this.productService = productService;
    }

    @GetMapping
    public List<Product> getAllProducts() {
        return productService.getAllProducts();
    }

    @GetMapping("/{id}")
    public ResponseEntity<Product> getProductById(@PathVariable Long id) {
        Product product = productService.getProductById(id);
        return ResponseEntity.ok(product);
    }

    @PostMapping
    public ResponseEntity<Product> createProduct(@RequestBody Product product) {
        Product savedProduct = productService.createProduct(product);
        return ResponseEntity.ok(savedProduct);
    }
}
```

2. Service Layer (Business Logic)

Encapsulates rules, validations, and business operations.

```
@Service
public class ProductService {

    private final ProductRepository productRepository;

    public ProductService(ProductRepository productRepository) {
        this.productRepository = productRepository;
    }

    public List<Product> getAllProducts() {
        return productRepository.findAll();
    }

    public Product getProductById(Long id) {
        return productRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Product not found"));
    }

    public Product createProduct(Product product) {
        if (product.getPrice() <= 0) {
            throw new IllegalArgumentException("Price must be positive");
        }
```

```
        return productRepository.save(product);
    }
}
```

3. Persistence Layer (Repository)

Responsible for database operations only.

```
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {
    List<Product> findByCategory(String category);
}
```

4. Model (Entity)

Represents database tables as Java objects.

```
@Entity
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String category;
    private double price;

    // Getters & Setters
}
```

How Dependency Injection Wires It All Together

1. Spring creates beans for @Controller, @Service, @Repository, and @Entity classes. 2. Dependencies are injected automatically (Controller → Service → Repository). 3. Example Flow (GET /products): - User → Controller → Service → Repository → Database → Response.