

Below is the **concise, correct, and interview-grade explanation** of how the **Controller, Service, and Repository layers interact** in a Spring Boot application. This is the standard **3-layer architecture** used in industry.

1. Roles (One Line Each)

- **Controller** → Handles HTTP requests and responses
- **Service** → Contains business logic and orchestration
- **Repository** → Handles database persistence

Each layer has **one responsibility** and **talks only to the adjacent layer**.

2. Who Talks to Whom (Rule)

Controller → Service → Repository → Database

Reverse flow for responses.

🚫 **No layer skips another layer**

3. Interaction Flow (Real Example)

Request: POST /api/employees

Step-by-step:

1. Client (React/Postman)

- Sends HTTP request with JSON

2. Controller

- Receives request
- Converts JSON → DTO
- Calls service method

```
employeeService.createEmployee(dto);
```

3. Service

- Applies business rules
- Converts DTO → Entity
- Calls repository

```
employeeRepository.save(employee);
```

4. Repository

- Uses JPA/Hibernate
- Executes SQL on database

5. Database

- Stores data
- Returns result

6. Response travels back

Database → Repository → Service → Controller → Client

4. What Each Layer MUST NOT Do

Controller MUST NOT:

- Access database
- Contain business rules
- Call repository directly

Service MUST NOT:

- Handle HTTP (@RequestBody, @PathVariable)
- Return database entities directly (use DTOs)

Repository MUST NOT:

- Contain business logic
 - Know about HTTP or DTOs
-

5. Dependency Direction (Very Important)

Controller depends on Service
Service depends on Repository

Repository depends on JPA

Dependencies are **one-directional**.

6. How Spring Wires Them (Key Insight)

At startup:

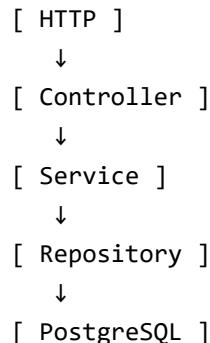
- Spring creates beans for all layers
- Injects dependencies via constructors

You never manually create objects.

```
public EmployeeController(EmployeeService service)
public EmployeeServiceImpl(EmployeeRepository repo)
```

Spring connects them.

7. Visual Mental Model (Text Diagram)



8. Why This Architecture Exists

- Separation of concerns
 - Easier testing
 - Replace database without touching controllers
 - Clean, maintainable code
-

9. Interview-Quality Summary (Memorize This)

"The controller handles HTTP concerns, delegates business processing to the service layer, which in turn interacts with the repository abstraction for persistence. Each

layer has a single responsibility and communicates only with the adjacent layer."

10. One Golden Rule (Remember Forever)

Controller = API Service = Brain Repository = Storage

If you want next:

- Where DTOs fit in this interaction
- How transactions work across layers
- How validation flows before service execution

Say the topic name.