

Below is the **short, precise call flow** of your program. This is the **mental model** you must have when working with Spring Boot. No code, no verbosity—only *who calls whom*.

1. Application Startup (Once)

Spring Boot → Spring Container

1. `main()` in `EmsApplication`
2. Spring scans packages:
 - `@Controller`
 - `@Service`
 - `@Repository`
3. Spring **creates objects (beans)**:
 - `EmployeeController`
 - `EmployeeServiceImpl`
 - `EmployeeRepository`
4. Spring **injects dependencies** via constructors

➡ You do NOT create objects. Spring does.

2. HTTP Request Flow (Runtime)

Example: POST `/api/employees`

```
Browser / React
    ↓
EmployeeController
    ↓
EmployeeServiceImpl
    ↓
EmployeeRepository
    ↓
PostgreSQL
```

3. Exact “Who Calls Who”

Step-by-step:

1. Client (React / Postman)

- Sends HTTP request

2. Spring MVC

- Finds matching controller method

3. EmployeeController

- Calls `employeeService.createEmployee(dto)`

4. EmployeeServiceImpl

- Calls `employeeRepository.save(entity)`

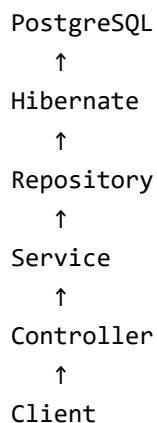
5. EmployeeRepository (JPA)

- Calls Hibernate

6. Hibernate

- Executes SQL on PostgreSQL

7. Response flows back up



4. Important Rule (Core Concept)

- **Controller never talks to database**
- **Repository never handles HTTP**
- **Service contains business logic**
- **Spring wires everything**

5. One-Line Summary (Remember This)

Client → **Controller** → **Service** → **Repository** → **Database** → **back**

If you understand this line, you understand **Spring Boot architecture**.

6. Why Your Error Happened (Linking Back)

You broke the chain by:

- Manually creating a class (new)
- Or Spring not managing a bean

Result:

Controller → Service → NULL Repository ❌

7. Interview-Level Answer (Use This)

“Spring MVC routes the HTTP request to a controller, the controller delegates to a service, the service uses a repository abstraction backed by JPA/Hibernate to interact with the database, and the response propagates back through the same layers.”

If you want next:

- Diagram (text-based)
- Lifecycle explanation
- How Spring decides *which method* to call

Say **diagram** or **lifecycle**.