Spring Boot Deep-Dive: Beans, DI, REST APIs, MVC, Tomcat, Hibernate, DTOs & Complete TODO Project

This version is *significantly more detailed, intense, and REST API–focused*. It is designed as a full learning manual.

---

1. What is Spring Boot (in depth)

Spring Boot is a modern, opinionated framework built on top of Spring to simplify:

- manual configuration
- dependency management
- server deployment

It provides:

✔️Opinionated autoconfiguration

Spring Boot scans project classpath and auto-configures:

- DataSource
- JPA/Hibernate
- Tomcat server
- Spring MVC
- Jackson JSON Mapper

✔️Embedded server

You do **not** deploy WARs. Spring Boot includes **embedded Tomcat/Jett/Netty**.

✔️Starter dependencies

Instead of manually adding 20 JARs, you add:

```
spring-boot-starter-web
spring-boot-starter-data-jpa
spring-boot-starter-validation
spring-boot-starter-test
```

---

2. Beans, IOC Container & Dependency Injection (Deep Explanation)

2.1 What is a Bean?

A **bean** is an object created, managed, and destroyed by Spring's **IoC container**.

Examples of beans:

- Controller classes (`@RestController`)
- Configuration classes
- Repositories (`@Repository`)
- Services (`@Service`)

Spring manages their:

- lifecycle
- instantiation
- injection

2.2 IoC Principle

Instead of your code creating objects, the IoC container creates them.

Traditional Java:

```
Car car = new Car();
Driver d = new Driver(car);
```

Spring IoC:

```
@RestController
class DriverController {
    private final Car car;

    public DriverController(Car car) {  // Spring injects
        this.car = car;
    }
}
```

Spring decides:

- **Which object to create**
- **When to inject it**
- **How long it should live (scope)**

---

3. Dependency Injection Types (Full Detail)

✔️Constructor Injection (recommended)

```
@Service
class TodoService {
    private final TodoRepository repo;

    public TodoService(TodoRepository repo) {
        this.repo = repo;
    }
}
```

Spring auto-injects because there is only **one constructor**.

Advantages:

- immutable dependencies
- test-friendly
- best practice in Spring Boot

✔️Field Injection (Not recommended for production)

```
@Autowired
private TodoRepository repo;
```

Downside:

- cannot be mocked easily
- hard to test
- prevents immutability

✔️Setter Injection

```
private TodoRepository repo;

@Autowired
public void setRepo(TodoRepository repo) {
    this.repo = repo;
}
```

Used when dependencies are optional.

---

4. Spring MVC Deep Explanation

Spring MVC is a web framework based on the **front controller pattern**.

Request Life Cycle:

```
Client → DispatcherServlet → HandlerMapping → Controller → Service →
Repository → DB
```

DispatcherServlet

A special servlet that:

- receives all HTTP requests
- forwards request to appropriate controller
- manages exception handling & JSON conversion

Controllers

REST controllers return JSON.

```
@RestController
@RequestMapping("/api/todos")
class TodoController {}
```

---

5. Tomcat Internal Explanation

Spring Boot embeds Tomcat. No need to install Tomcat separately.

Tomcat Responsibilities:

- Start HTTP server on port 8080
- Manage request threads
- Create servlet containers
- Load `DispatcherServlet`

Why embedded Tomcat?

- No WAR deployment
- Portable app
- Simply run:

```
java -jar app.jar
```

---

6. REST API — Deep Explanation

A Spring Boot REST API typically has:

- **Controller** → Handles HTTP requests/responses
- **Service** → Business logic
- **Repository** → Database interaction
- **DTOs** → Request/Response mapping
- **Entity** → JPA/Hibernate managed table mapping

Each layer has a purpose.

---

7. DTOs (Request & Response) + ENTITY Mapping

Why DTOs?

Entities should NOT be returned directly because:

- Entities represent DB tables
- Entities may contain sensitive fields
- Entities may expose internal DB structure

Thus:
Controller ↔ DTO ↔ Service ↔ Entity ↔ DB

Example TODO DTOs

Request DTO

```
public class TodoRequest {
    @NotBlank
    private String title;

    private boolean completed;

    // getters setters
}
```

Response DTO

```
public class TodoResponse {
    private Long id;
    private String title;
```

```
    private boolean completed;
}
```

Entity

```
@Entity
@Table(name = "todos")
public class TodoEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private boolean completed;
}
```

---

8. Manual Mapping: toEntity() & toDTO()

Convert Request → Entity

```
public TodoEntity toEntity(TodoRequest dto) {
    TodoEntity entity = new TodoEntity();
    entity.setTitle(dto.getTitle());
    entity.setCompleted(dto.isCompleted());
    return entity;
}
```

Convert Entity → Response

```
public TodoResponse toDTO(TodoEntity entity) {
    TodoResponse dto = new TodoResponse();
    dto.setId(entity.getId());
    dto.setTitle(entity.getTitle());
    dto.setCompleted(entity.isCompleted());
    return dto;
}
```

---

9. Full TODO API (Deep Version)

9.1 Controller Layer

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

    private final TodoService service;

    public TodoController(TodoService service) {
        this.service = service;
    }

    @PostMapping
    public ResponseEntity<TodoResponse> create(@RequestBody @Valid
TodoRequest request) {
        return ResponseEntity.ok(service.create(request));
    }

    @GetMapping
    public List<TodoResponse> getAll() {
        return service.getAll();
    }

    @GetMapping("/{id}")
    public TodoResponse getOne(@PathVariable Long id) {
        return service.getOne(id);
    }

    @DeleteMapping("/{id}")
    public void delete(@PathVariable Long id) {
        service.delete(id);
    }
}
```

9.2 Service Layer

```
@Service
public class TodoService {

    private final TodoRepository repo;

    public TodoService(TodoRepository repo) {
        this.repo = repo;
    }
```

```
    public TodoResponse create(TodoRequest request) {
        TodoEntity entity = toEntity(request);
        repo.save(entity);
        return toDTO(entity);
    }

    public List<TodoResponse> getAll() {
        return repo.findAll().stream().map(this::toDTO).toList();
    }

    public TodoResponse getOne(Long id) {
        TodoEntity entity = repo.findById(id)
                .orElseThrow(() -> new RuntimeException("Todo Not Found"));
        return toDTO(entity);
    }

    public void delete(Long id) {
        repo.deleteById(id);
    }

    private TodoEntity toEntity(TodoRequest dto) {
        TodoEntity e = new TodoEntity();
        e.setTitle(dto.getTitle());
        e.setCompleted(dto.isCompleted());
        return e;
    }

    private TodoResponse toDTO(TodoEntity e) {
        TodoResponse dto = new TodoResponse();
        dto.setId(e.getId());
        dto.setTitle(e.getTitle());
        dto.setCompleted(e.isCompleted());
        return dto;
    }
}
```

9.3 Repository Layer

```
@Repository
public interface TodoRepository extends JpaRepository<TodoEntity, Long> {}
```

---

10. Hibernate Deep Explanation

Hibernate is Spring Boot's default ORM.

Responsibilities:

- map entities to database tables
- handle CRUD queries
- manage caching
- handle transactions

JPA → Hibernate

All Spring Boot apps use JPA interfaces:

```
JpaRepository
Entity
Table
Id
```

Hibernate is the **implementation**.

---

11. Application Properties

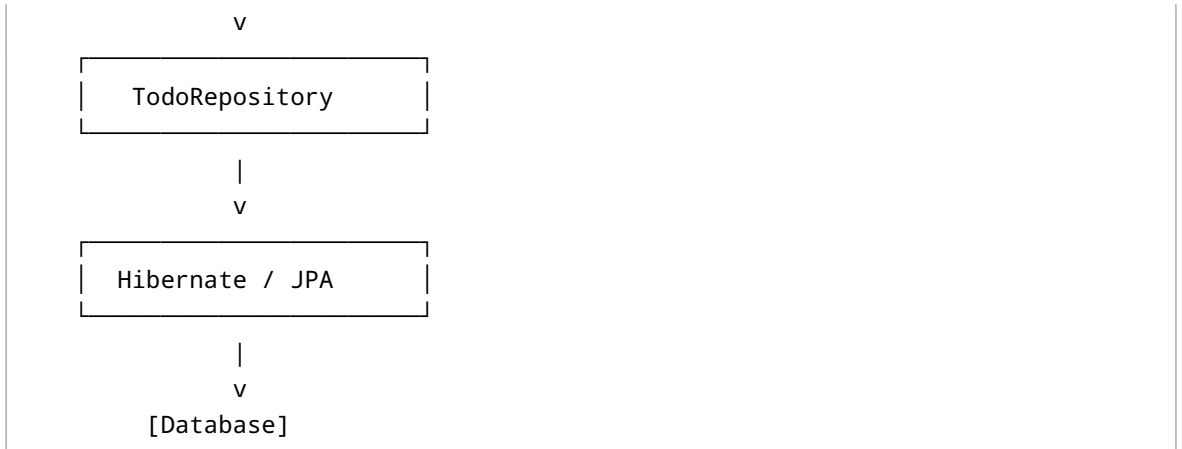
```
spring.datasource.url=jdbc:mysql://localhost:3306/todo_db
spring.datasource.username=root
spring.datasource.password=pass

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

---

12. Complete Flow (Architecture Diagram)

```
      [Client/Postman]
              |
              v
      /api/todos (JSON)
              |
              v
   ┌──────────────────────┐
   │   TodoController     │
   └──────────────────────┘
              |
              v
   ┌──────────────────────┐
   │    TodoService       │
   └──────────────────────┘
              |
```



Want me to add more sections?

I can add:

- Swagger documentation
- Global exception handling
- MapStruct mapping version
- JWT authentication
- Service layer interfaces
- Unit tests (Mockito, WebMvcTest)
- More advanced REST patterns (PATCH, filtering, pagination)

Just tell me!