

# Curso Java COMPLETO

[devsuperior.com.br](https://devsuperior.com.br)

Dr. Nelio Alves

---

## Projeto web services com Spring Boot e JPA / Hibernate

### Objetivos

- Criar projeto Spring Boot Java
- Implementar modelo de domínio
- Estruturar camadas lógicas: resource, service, repository
- Configurar banco de dados de teste (H2)
- Povoar o banco de dados
- CRUD - Create, Retrieve, Update, Delete
- Tratamento de exceções

### Github:

<https://github.com/acenelio/workshop-springboot2-jpa>

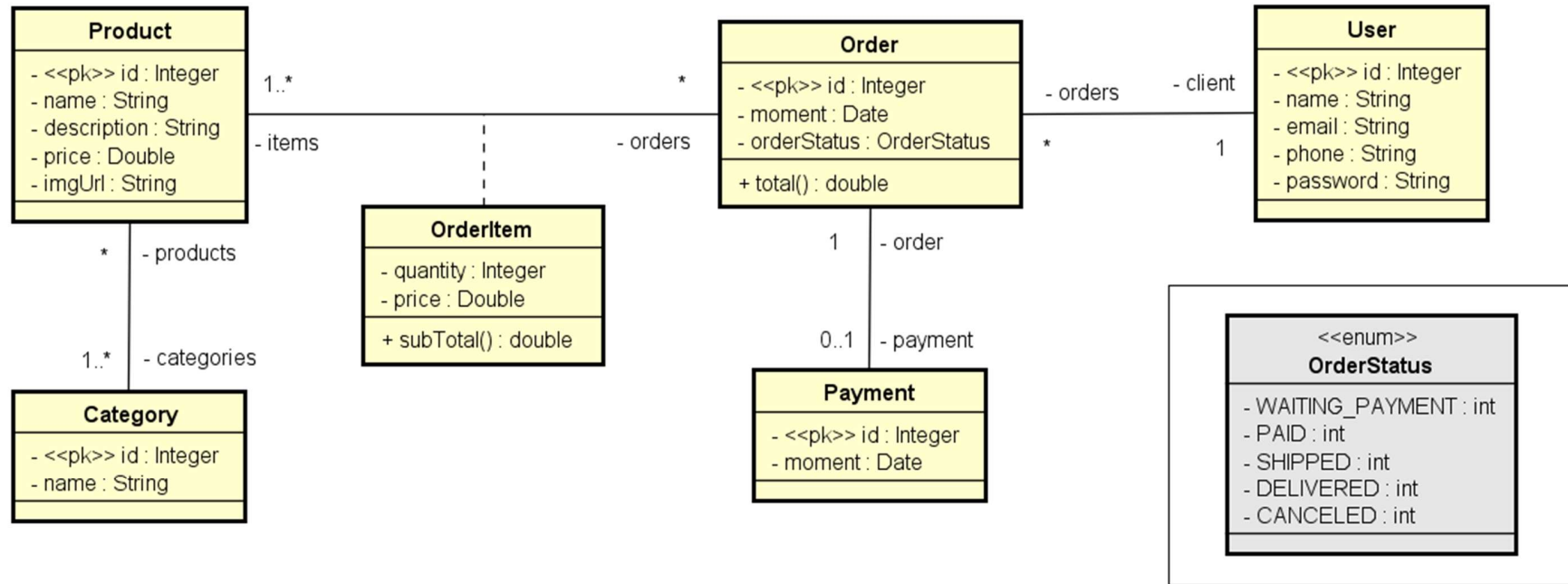
<https://github.com/acenelio/workshop-springboot3-jpa>



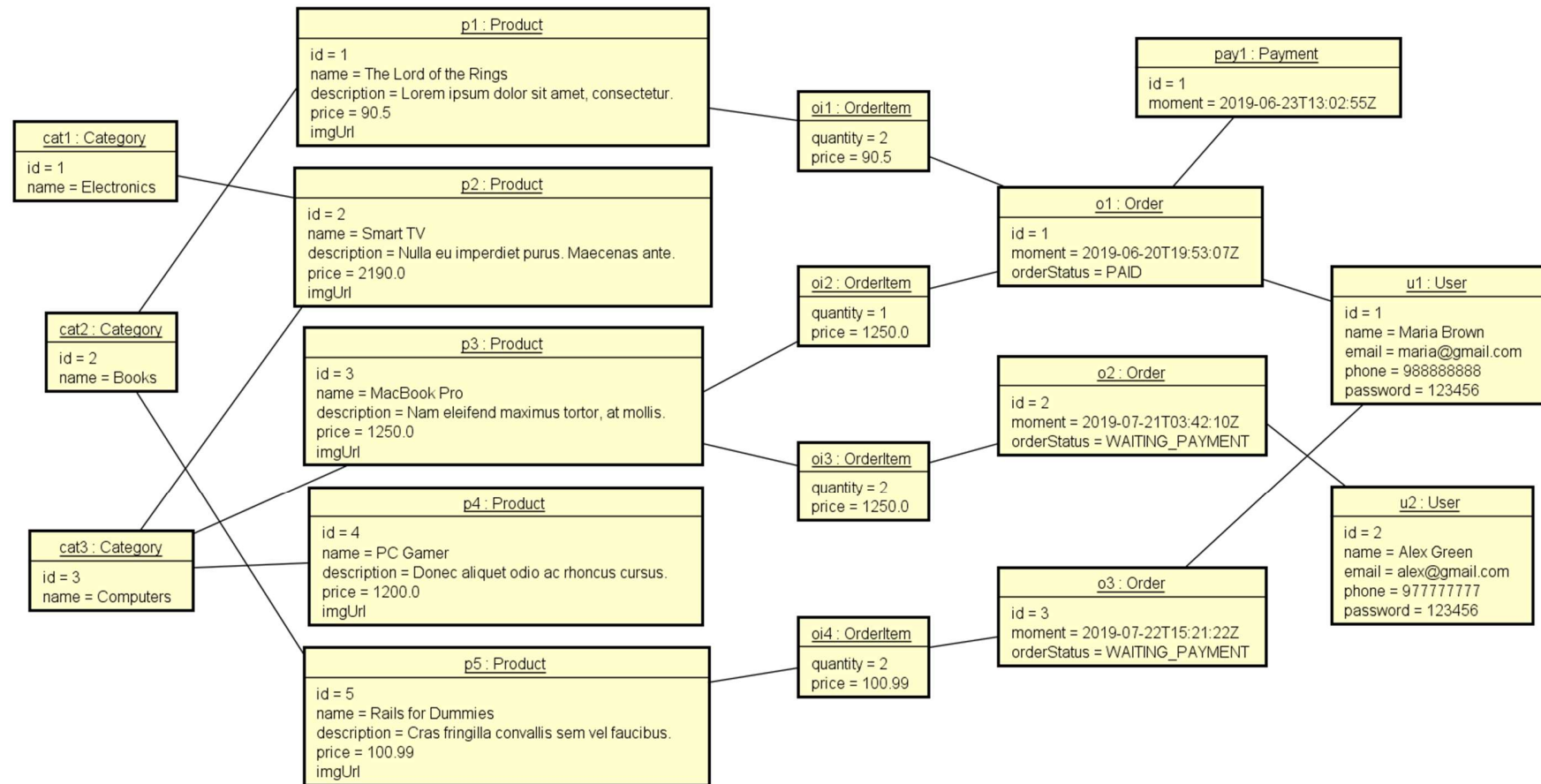
PostgreSQL



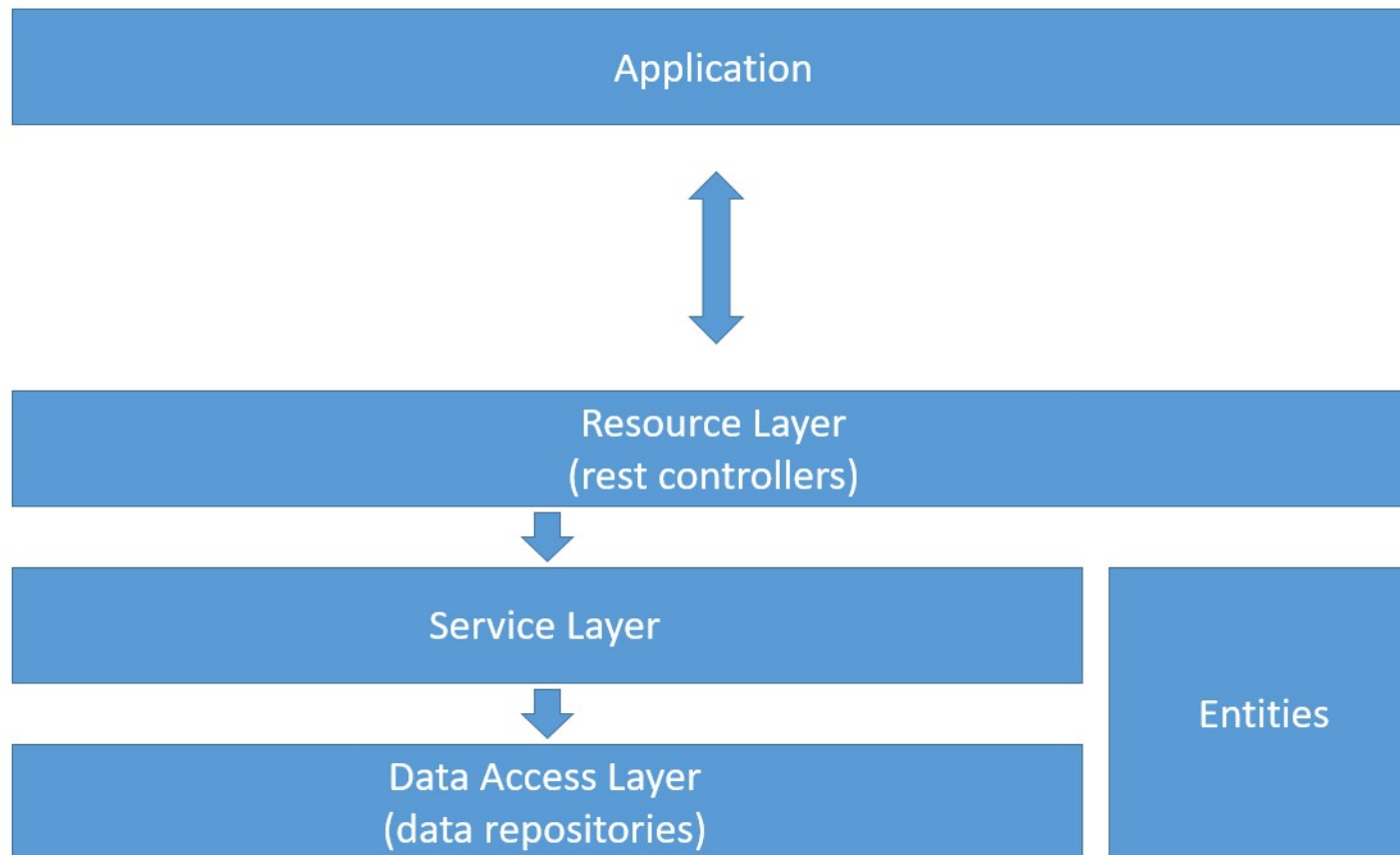
## Domain Model



## Domain Instance



## Logical Layers



## Project created

### Checklist:

- Spring Initializr
  - Maven
  - Java 17
  - Packing JAR
  - Dependencies: Spring Web

## User entity and resource

### Basic entity checklist:

- Basic attributes
- Associations (instantiate collections)
- Constructors
- Getters & Setters (collections: only get)
- hashCode & equals
- Serializable

## H2 database, test profile, JPA

### Checklist:

- JPA & H2 dependencies
- application.properties
- application-test.properties
- Entity: JPA mapping

### Dependencies:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>
```

### application.properties:

```
spring.profiles.active=test

spring.jpa.open-in-view=true
```

### application-test.properties:

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=

spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

## JPA repository, dependency injection, database seeding

### Checklist:

- UserRepository extends JpaRepository<User, Long>
- Configuration class for "test" profile
- @Autowired UserRepository
- Instantiate objects in memory
- Persist objects

### Objects:

```
User u1 = new User(null, "Maria Brown", "maria@gmail.com", "988888888", "123456");
User u2 = new User(null, "Alex Green", "alex@gmail.com", "977777777", "123456");
```

## Service layer, component registration

## Order, Instant, ISO 8601

### Basic new entity checklist:

- Entity
  - "To many" association, lazy loading, JsonIgnore
- Repository
- Seed
- Service
- Resource

### Objects:

```
Order o1 = new Order(null, Instant.parse("2019-06-20T19:53:07Z"), u1);
Order o2 = new Order(null, Instant.parse("2019-07-21T03:42:10Z"), u2);
Order o3 = new Order(null, Instant.parse("2019-07-22T15:21:22Z"), u1);
```

## OrderStatus enum

## Category

### Objects:

```
Category cat1 = new Category(null, "Electronics");  
Category cat2 = new Category(null, "Books");  
Category cat3 = new Category(null, "Computers");
```

## Product

### Objects:

```
Product p1 = new Product(null, "The Lord of the Rings", "Lorem ipsum dolor sit amet, consectetur.", 90.5, "");  
Product p2 = new Product(null, "Smart TV", "Nulla eu imperdiet purus. Maecenas ante.", 2190.0, "");  
Product p3 = new Product(null, "Macbook Pro", "Nam eleifend maximus tortor, at mollis.", 1250.0, "");  
Product p4 = new Product(null, "PC Gamer", "Donec aliquet odio ac rhoncus cursus.", 1200.0, "");  
Product p5 = new Product(null, "Rails for Dummies", "Cras fringilla convallis sem vel faucibus.", 100.99, "");
```

## Many-to-many association with JoinTable

## OrderItem, many-to-many association with extra attributes

### Checklist:

- OrderItemPK
- OrderItem
- Order one-to-many association
- Seed

### Objects:

```
OrderItem oi1 = new OrderItem(o1, p1, 2, p1.getPrice());  
OrderItem oi2 = new OrderItem(o1, p3, 1, p3.getPrice());  
OrderItem oi3 = new OrderItem(o2, p3, 2, p3.getPrice());  
OrderItem oi4 = new OrderItem(o3, p5, 2, p5.getPrice());
```

## Product-OrderItem one-to-many association

## Payment, one-to-one association

## Subtotal & Total methods

## User insert

### Checklist:

- UserService
- UserResource

### Test:

```
{  
  "name": "Bob Brown",  
  "email": "bob@gmail.com",  
  "phone": "977557755",  
  "password": "123456"  
}
```

## User delete

### Checklist:

- UserService
- UserResource

## User update

### Checklist:

- UserService
- UserResource

### Test:

```
{  
  "name": "Bob Brown",  
  "email": "bob@gmail.com",  
  "phone": "977557755"  
}
```

## Exception handling - findById

### Checklist:

- NEW CLASS: services.exceptions.ResourceNotFoundException
- NEW CLASS: resources.exceptions.StandardError
- NEW CLASS: resources.exceptions.ResourceExceptionHandler
- UserService



## Exception handling - delete

### Checklist:

- NEW CLASS: services.exceptions.DatabaseException
- ResourceExceptionHandler
- UserService
  - EmptyResultDataAccessException
  - DataIntegrityViolationException

## Exception handling - update

### Checklist:

- UserService
  - EntityNotFoundException

## Deploy Heroku

**ATENÇÃO: OPCIONAL / PAGO (CARTÃO DE CRÉDITO)**

### Create Heroku app & provision PostgreSQL

#### Checklist:

- Heroku Sign Up
- Create app
- Provision PostgreSQL
  - App dashboard -> Resources
  - Search "postgres" -> select "Heroku Postgres"

### Install local PostgreSQL

#### Checklist:

- Download and install: <https://www.postgresql.org/download/>
  - Super user: postgres
  - Password: 1234567
  - Port: 5432
- Start/stop service: Task manager -> Services
- Check instalation
  - Start pgAdmin
  - Databases -> Create -> Database
    - Encoding: UTF8

### Dev profile

#### Checklist:

- PgAdmin: create local database: create database springboot\_course
- Add PostgreSQL Maven dependency

```
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <scope>runtime</scope>
</dependency>
```

- Create file: application-dev.properties

```
spring.datasource.url=jdbc:postgresql://localhost:5432/springboot_course
spring.datasource.username=postgres
spring.datasource.password=1234567
```

```
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

```
jwt.secret=MYJWTSECRET
jwt.expiration=3600000
```

- Update application.properties: spring.profiles.active=dev
- Run application

## Get SQL script from local PostgreSQL

- PgAdmin: get SQL script:
  - Select database
  - Tools -> Backup
    - Format: Plain
    - Encoding: UTF8
    - Dump options:
      - Only schema: YES
      - Blobs: NO
      - Do not save: (ALL)
      - Verbose messages: NO
- Delete instructions before CREATE statements

## Run SQL Script

### Checklist:

- App dashboard -> Settings - > Config Vars

EXAMPLE:

postgres://wavglvupbdad:358f443aafe452eca4c58fbc15d02e50b08130c7aaea3aff6c4f59c13f9abb@ec2-23-21-106-266.compute-1.amazonaws.com:5432/d7u9ub86cdsu

**user:** wavglvupbdad

**password:** 358f443aafe452eca4c58fbc15d02e50b08130c7aaea3aff6c4f59c13f9abb

**server:** ec2-23-21-106-266.compute-1.amazonaws.com

**port:** 5432

**database:** d7u9ub86cdsu

- PgAdmin: Servers -> Create -> Server
  - Advanced -> DB rescription: (database)
- Database -> Query Tool
  - Load and run SQL Script

## Heroku CLI

Google: Heroku CLI

### Terminal

```
heroku login
```

```
winpty heroku.cmd login
```

## Deploy app to Heroku

- Heroku app dashboard -> Deploy

```
heroku git:remote -a myapp
git remote -v
```
- Setup Heroku app Config Vars
  - DATABASE\_URL
  - JWT\_EXPIRATION
  - JWT\_SECRET
- Create: application-prod.properties

```
spring.datasource.url=${DATABASE_URL}
```

```
spring.jpa.hibernate.ddl-auto=none
```

```
spring.jpa.show-sql=false
```

```
spring.jpa.properties.hibernate.format_sql=false
```

```
jwt.secret=${JWT_SECRET}
```

```
jwt.expiration=${JWT_EXPIRATION}
```

- Update application.properties: `spring.profiles.active=prod`
- Create files: system.properties

```
java.runtime.version=17
```

- Send to Heroku:  
    `git add .`  
    `git commit -m "Deploy app to Heroku"`  
    `git push heroku main`