**CÁC ANNOTATION**

**1. JPA Annotations (Mapping Entity)**

**Primary Key & Generation**

* @Id: Đánh dấu primary key cho một entity.
* @GeneratedValue: Chỉ định cách primary key được sinh ra.
  + **Strategies**:
    - GenerationType.IDENTITY: Tự động tăng trong cơ sở dữ liệu.
    - GenerationType.SEQUENCE: Dựa vào sequence của cơ sở dữ liệu.
    - GenerationType.TABLE: Sử dụng bảng riêng để tạo giá trị.
    - GenerationType.AUTO: Hibernate tự chọn chiến lược.

**Column Mapping**

* @Column: Tuỳ chỉnh cột trong bảng.
  + **Attributes**:
    - name: Tên cột.
    - nullable: Cho phép null hay không.
    - unique: Đảm bảo giá trị duy nhất.
    - length: Độ dài (cho chuỗi).
    - precision & scale: Định nghĩa độ chính xác số thực.

**Temporal Types**

* @Temporal: Chỉ định kiểu dữ liệu ngày tháng.
  + **Types**:
    - TemporalType.DATE: Chỉ lưu ngày.
    - TemporalType.TIME: Chỉ lưu thời gian.
    - TemporalType.TIMESTAMP: Lưu cả ngày và giờ.

**Relationship Mapping**

* @OneToOne, @ManyToOne, @OneToMany, @ManyToMany: Định nghĩa quan hệ giữa các bảng.
  + **Attributes**:
    - mappedBy: Định nghĩa mối quan hệ ngược.
    - cascade: Cách thức xử lý cascading (PERSIST, REMOVE, etc.).
    - fetch: Chỉ định cách dữ liệu được nạp (EAGER, LAZY).

**Inheritance Mapping**

* @Inheritance: Định nghĩa kiểu kế thừa.
  + **Strategies**:
    - InheritanceType.SINGLE\_TABLE: Tất cả các lớp con lưu cùng bảng.
    - InheritanceType.JOINED: Mỗi lớp con có bảng riêng.
    - InheritanceType.TABLE\_PER\_CLASS: Tạo bảng riêng biệt cho từng lớp.

**2. Hibernate-Specific Annotations**

* @CreationTimestamp & @UpdateTimestamp: Ghi nhận thời gian tạo và cập nhật tự động.
* @Formula: Tạo cột ảo tính toán dựa trên SQL. Không cần lưu trữ cột này trong cơ sở dữ liệu.Ví dụ : @Formula("unit\_price \* quantity")
* @Type: Định nghĩa kiểu dữ liệu Hibernate tuỳ chỉnh.Thường dùng khi làm với enum. @Type(type = "org.hibernate.type.TimestampType")

**3. Validation Annotations (Kiểm tra dữ liệu phía Server)**

**Basic Validations**

* @NotNull: Đảm bảo giá trị không null. Không kiểm tra tính rỗng hay chuỗi khoảng trắng, chỉ cần giá trị không phải null.
* @NotEmpty: Đảm bảo không null và không rỗng (String, Collection như danh sách , mảng). Nhưng nếu là chuỗi chứa khoảng trắng vẫn đúng “ “ .Còn “” hoặc null là sai.
* @NotBlank: Đảm bảo String không null, không rỗng và không chỉ chứa khoảng trắng.
* @Size: Kiểm tra độ dài hoặc kích thước. Áp dụng cho chuỗi là kiểm tra số kí tự, mảng là kiểm tra số phần tử, bộ sưu tập (Collection, List, Set, Map,...) kiểm tra số phần tử.

🡪Ngoại trừ @NotNull các cái còn lại không dùng cho kiểu số

**Numeric Validations**

* @Min / @Max: Giá trị tối thiểu và tối đa.Ví dụ @Min(value = 10, message = "Giá trị phải lớn hơn hoặc bằng 10")
* @Positive / @PositiveOrZero: Kiểm tra số nguyên giá trị số dương.
* @DecimalMin(value = "0.0", inclusive = false, message = "Price phải lớn hơn 0")
* @Negative / @NegativeOrZero: Kiểm tra giá trị số âm.
* @Digits: Đảm bảo giá trị số thực với giới hạn chữ số. Ví dụ : @Digits(integer = 5, fraction = 2) 🡪 Tức là 5 số nguyên và chỉ có 2 số thập phân XXXXX,YY
* @Digits không tự động làm tròn số. Nếu cần làm tròn, bạn nên sử dụng thêm các phương pháp xử lý trong Java (BigDecimal với setScale,...).

**String Validations**

* @Pattern: Kiểm tra theo biểu thức chính quy (Regex).

**Date Validations**

* @Past / @PastOrPresent: Ngày phải ở quá khứ hoặc hiện tại.
* @Future / @FutureOrPresent: Ngày phải ở tương lai hoặc hiện tại.

**4. Spring-Specific Annotations**

* @Transient: Bỏ qua trường không muốn lưu vào cơ sở dữ liệu.
* @JsonIgnore: Bỏ qua trường khi serialize/deserialize JSON (nếu dùng Jackson).
* @Value: Gán giá trị mặc định từ file cấu hình.

**5.**

**@Configuration  
@EnableWebSecurity**

**@Service**

Là một annotation trong Spring, dùng để đánh dấu một class là một "Service" component.

Nó giúp Spring biết rằng class này đảm nhận logic xử lý nghiệp vụ (business logic) và Spring sẽ quản lý vòng đời của nó.

**Khi nào sử dụng @Service:**

Khi bạn có một class chịu trách nhiệm xử lý logic nghiệp vụ (ví dụ: tính toán, kết hợp dữ liệu từ các nguồn, điều phối giữa các tầng như DAO và Controller).

Thường dùng để định nghĩa các service layer trong kiến trúc **Layered Architecture**:

* **Controller Layer**: Xử lý yêu cầu HTTP, trả kết quả cho client.
* **Service Layer**: Xử lý logic nghiệp vụ.
* **DAO Layer**: Tương tác với cơ sở dữ liệu.

**@Autowired : phải có @Service mói dùng được**

Tiêm các dependency (AppUserDao, AppRoleDao) để sử dụng trong lớp. Điều này giúp loại bỏ việc khởi tạo thủ công và tận dụng Spring IoC (Inversion of Control).

Được sử dụng để **tiêm (inject)** các dependency (phụ thuộc) vào một class.

Nó cho phép Spring tự động tìm và khởi tạo các bean cần thiết, dựa vào container của Spring. Hiểu đơn giản, @Autowired giúp bạn **nhúng** một class vào một class khác mà **không cần khởi tạo thủ công**.

**@Configuration**

Cho biết đây là class cấu hình dùng trong Spring security

**@EnableWebSecurity**

Bật tính năng bảo mật web của Spring Security.

**@Repository**

@Repository là một annotation của Spring để đánh dấu một lớp là repository, nơi chứa các thao tác truy vấn dữ liệu. Annotation này giúp Spring tự động xử lý các exception liên quan đến cơ sở dữ liệu (ví dụ: DataAccessException).**Tóm @Repository không tạo ra API REST, nó chỉ đánh dấu lớp này là nơi tương tác với cơ sở dữ liệu.**

**@RepositoryRestResource**

Mục đích chính của @RepositoryRestResource là để tạo ra một API REST tự động cho repository của bạn mà không cần phải viết controller thủ công.

**Spring Data REST** sẽ tự động tạo các endpoint cho tất cả các phương thức CRUD của repository, ví dụ như:

* GET để lấy danh sách hoặc chi tiết dữ liệu.
* POST để tạo mới.
* PUT để cập nhật.
* DELETE để xóa.

Nếu **exported = false**, nghĩa là bạn chỉ sử dụng repository này trong code nội bộ của mình, không cung cấp API REST ra ngoài.

**@ControllerAdvice**

Annotation này đánh dấu class là một global exception handler.

Nó sẽ áp dụng cho tất cả các controller trong ứng dụng.

Bất kỳ exception nào ném ra từ controller sẽ được xử lý bởi class này, thay vì trả về lỗi mặc định của Spring.

**@RestController:**

Biến lớp này thành một **REST API controller**.

Các phương thức trong lớp này sẽ trả về **dữ liệu JSON** hoặc **XML** (thường là JSON).

# **Các loại lỗi**

## **Lỗi 404 (Not Found)**

**Ý nghĩa**:

Lỗi này xảy ra khi server không tìm thấy tài nguyên (resource) mà client yêu cầu.

Tài nguyên có thể là một trang web, một API endpoint, hoặc một file cụ thể.

**Nguyên nhân phổ biến**:

* **URL không tồn tại**:

Client gửi một request đến URL không được định nghĩa trên server.

Ví dụ: /api/addresses/123 nhưng không có route nào xử lý 123.

* **Tài nguyên bị xóa hoặc di chuyển**:

Ví dụ: Một bài viết hoặc sản phẩm đã bị xóa khỏi cơ sở dữ liệu.

* **Sai lỗi đánh máy (typo) trong URL**:

Ví dụ: Client nhập /adress thay vì /address.

**Khi nào xảy ra?**

Trong Spring Boot, nếu không có controller hoặc endpoint nào khớp với URL, hoặc khi dữ liệu cần tìm không tồn tại, lỗi 404 sẽ được trả về.

## **Lỗi 500 (Internal Server Error)**

**Ý nghĩa**:

Đây là lỗi **phía server** (server-side), cho biết server gặp vấn đề khi xử lý request.

Nó không chỉ rõ lỗi là gì, chỉ biết rằng có một lỗi **bên trong server**.

**Nguyên nhân phổ biến**:

* **Lỗi lập trình**:

Exception không được xử lý, như NullPointerException hoặc ArrayIndexOutOf Bounds Exception.

* **Sai cấu hình server**: Ví dụ: Kết nối với cơ sở dữ liệu không thành công.
* **Lỗi logic trong ứng dụng**: Ví dụ: Một method tính toán trả về giá trị không hợp lệ.
* **Kết nối với service khác bị lỗi**: Ví dụ: Kết nối tới API của bên thứ ba thất bại.

**Khi nào xảy ra?**

Trong Spring Boot, lỗi 500 thường xảy ra khi một exception không được xử lý hoặc có lỗi nghiêm trọng trong logic code.