[**https://stackoverflow.com/questions/71179660/spring-boot-3-jakarta-and-javax**](https://stackoverflow.com/questions/71179660/spring-boot-3-jakarta-and-javax)

**Từ Spring Boot ver 3.x trở đi, sẽ dùng thư viện jakarta của Jakarta EE 9 thay vì javax. Và sẽ dùng Java 17.**

**-> khi import các thư viện vẫn xài thư viện của javax, thì down parent version của Spring Boot xuống ver 2.x.**

**Hoặc đơn giản hơn là cứ vào luôn solution copy file pom.xml cho nhanh.**

1. **JPA/Hibernate**
   * **Java Persistence API** là cơ sở để quản lí dữ liệu quan hệ trong app Java. JPA ko phải là tool hoặc framework, JPA là 1 tập hợp các interface để truy cập, duy trì và quản lí data giữa object Java và CSDL
     + JPA chỉ là specification (định nghĩa), chứ ko phải implementation
     + Nó là một tập hợp các quy tắc và hướng dấn để set các interface để triển khai ORM
   * **Hiểu hôm na JPA là một tập hợp các interface chứa những method để có thể thao tác với data giữa app Java và database**
   * JPA định nghĩa những API để thực hiện những việc : data conversion, querying, validation, schema generation

**=> vì JPA là tập hợp các interface, nên sẽ cần implementation để triển khai ORM (JPA providers/ORM frameworks).**

* + **Persistence data (dữ liệu bền vững)** là một khái niệm nói đến vấn đề rằng nếu dữ liệu không được lưu trữ khi hệ thống ngưng hoạt động cho đến khi nó hoạt động lại thì tất cả các dữ liệu phải được khởi tạo và xử lý lại từ đầu. Vậy nên ta cần đảm bảo rằng dữ liệu vẫn có thể sống kể cả khi ứng dụng ngưng hoạt động và có thể tải lên để xử lý tiếp khi hệ thống hoạt động trở lại. **Hiểu nôm na là cứ dữ liệu gì cần lưu trữ lâu dài thì phải lưu vào database.**
  + Entity design
    - Values
    - Entity types
  + Primary key:
    - Identifier: @Id
    - Genterated Strategy:
    - Compose identifier: @EmbbedId
  + Relationship
    - OneToOne
    - OneToMany
    - ManyToMany
  + Inheritance:
    - Single table
    - Joined
    - Table per class
    - Mapped superclass:
  + Data Conversion:
    - DTO
    - JSONView
  + <https://techmaster.vn/posts/36269/trang-thai-cua-entity-thuc-the-trong-hibernate-va-jpa>
  + JPA/Hibernate thay đổi tư duy lập trình viên từ câu lệnh SQL sang chuyển đổi trạng thái entity. Khi entity được quản lý bằng JPA/Hibernate thì mọi thay đổi của entity sẽ được tự động truyền vào trong cơ sở dữ liệu (database). Hiểu đơn giản là nếu dùng Hibernate thì ta sẽ ko phải xài các câu lệnh SQL nữa mà ta sẽ thực hiện những thay đổi trực tiếp object được ORM ánh xạ sang record trong table.
  + Thao tác với các entity (cùng với liên kết của chúng) dễ dàng hơn nhiều so với việc viết bằng câu lệnh SQL. Không có thư viện ORM thì mỗi khi thêm một cột mới vào sẽ đòi hỏi sửa đổi tất cả các câu lệnh INSERT/UPDATE liên quan.
  + Nhưng đi kèm theo đó là vì Hibernate sẽ tự động làm hết mọi thứ cho chúng ta nên chúng ta sẽ khó kiểm soát các câu lệnh SQL được thực thi khi nào và như thế nào. Có nghĩa là sau khi ta code xong ta vẫn phải debug để xem câu lệnh SQL được hibernate sinh ra có đúng theo ý ta không.
  + Muốn biết sự thay đổi entity đã được apply vào database chưa, ta cần phải biết mối quan hệ giữa entity và table đang lưu trữ nó là gì. **Pesistence context** sẽ lưu những thông tin đó.
  + **Persitence context** mô tả mỗi quan hệ giữa tất cả các entity instance trong app và mối quan hệ của chúng ở dưới database.
  + Có 4 state của entity instance trong persistence context:
    - **Transient/New** (tạm thời) là một object vừa mới được tạo ra mà chưa được kết nối với persistence context và cũng chưa được ánh xạ vào bất cứ bảng nào trong database. Vậy nên persistence context sẽ ko biết sự tồn tại của object này.
    - **Persistent/Managed** (bền vững) là một object được tạo ra và đã được kết nối với persistence context. Bất kì thay đổi nào lên object ở state này sẽ đều tự động truyền xuống database mà ko cần gọi persist, merge, remove.
    - **Detached/Unmanaged** (đã bị tách riêng ra) là một object đã bị tách khỏi persistence context khi persistence context đang chạy đó bị đóng lại. Những thay đổi này lên object sẽ ko còn tự động truyền xuống database nữa.
    - **Removed** (đã bị xoá): là một object đã bị detached nhưng record ứng với object này đã bị xoá khỏi database.
  + Để ta kiểm soát được việc modify object đó có được truyền xuống database không, ta cần thay đổi state của các entity đó. Khi đó ta sẽ dùng **EntityManager**.
  + **EntityManager** có một số method:
    - **Persist**: save entity ở state transient xuống database và chuyển state entity thành persistent.
    - **Find**: lấy 1 object ở dưới database, và entity trả về sẽ có sẵn state là persistent. Vì vậy những thay đổi lên object đó sẽ được truyền xuống database mà ko cần gọi lại method persist.
    - **Merge**: save entity ở state detached xuống database nhưng entity đó vẫn sẽ ở state detached.
    - **Remove**: detach một entity ở state persistent và xoá khỏi database.
  + **Note:** tất cả những thay đổi này chỉ áp dụng trên persistence context và sẽ chỉ được truyền xuống database khi Persistent context đó kết thúc.
  + **EntityManager** là của JPA. Hibernate thì có **Session**, là implementation của **EntityManager**, vây nên **Session** của hibernate sẽ có nhiều method hơn.
  + Khi có những entity associations nằm trong entity. Việc lúc nào cũng lấy những entity con nằm trong entity cha ta cần lấy sẽ hơi thừa thãi và tiêu tốn tài nguyên cho database. **Fetch type** của Hibernate sẽ xử lí việc này:
    - **EAGER**: lấy toàn bộ value mà nằm trong entity mà ta fetch. Hibernate mặc định sẽ dùng fetch type này cho quan hệ **OneToOne** và **ManyToOne**.
    - **LAZY**: chờ việc lấy value nằm trong entity cho tới khi chúng được refer. Tức là khi chưa được refer, entity type này sẽ không có giá trị, sau khi được refer thì chúng sẽ lấy dữ liệu dưới database và gán vào nên sẽ có giá trị. Hibernate mặc định sẽ dùng fetch type này cho quan hệ **OneToMany** và **ManyToMany**.
  + Entity relationship thường phụ thuộc vào sự tồn tại của entity khác, ví dụ Person – Address. Nếu ko có person, address entity sẽ ko có ý nghĩa. Vậy nên khi ta xoá Person, Address nên bị xoá theo. **Cascade** là cách để đạt được điều này.
  + **CascadeType** cho phép ta thực hiện hành động persistence nào đó trên Entity thì những entity assiociations cũng sẽ được thực hiện theo.
    - **ALL:** tương ứng với tất cả các loại cascade. cascade={DETACH, MERGE, PERSIST, REFRESH, REMOVE}
    - **PERSIST:** Nếu đối tượng cha được persisted vào persistence context, thì các đối tượng tham chiếu tới nó cũng được persisted.
    - **DETACH:** tương tự
    - **MERGE:** tương tự
    - **PERSIST:** tương tự
    - **REMOVE:** tương tự
  + **JPQL (Java Persistence Query Language)** là query được viết cho Entity manager để có thể trả về entity. Syntax khá giống SQL và sẽ tự động convert sang câu SQL phù hợp với database mà app đang dùng. (trông có vẻ ít dùng nên ko cần học)
    - Typed query
    - Named query
  + **Hibernate** là một trong những JPA Providers/ORM framework (**Object Relational Mapping**) phổ biến giúp mapping giữa record dưới database và object trong Java. Hiểu đơn giản đây là một implementation của JPA API
  + **Spring Data JPA** cung cấp code và code generation tools để ta có thể dễ dàng sử dụng JPA. Nó cung cấp cho ta những đoạn code đơn giản như find, save, remove… cho những thao tác select, insert, delete đơn giản và có thể tạo ra những method cho những thao tác phức tạp hơn (bằng cách tự động generate JPQL). Việc này sẽ do **Spring Data common** làm. => cái này sinh ra để giảm boilerplate code cho những việc persist, merge,… lặp đi lặp lại.
  + Khi ta khai báo repository, chỉ cần viết interface **JpaRepository**, khi run app Spring Boot, implementation của nó sẽ tự động dc tạo ra. Class **SimpleJpaRepository** là implementation của interface **JpaRepository,** đã có implementation của toàn bộ các hàm cơ bản như save, delete,… sử dụng **EntityManager** => annotation **@Repository** có hay ko cũng được vì implementation của nó đã có annotation **@Repository** rồi
  + Interface **JpaRepository** thuộc library **Spring Data JPA**, extends từ interface **CrudRepository** thuộc library **Spring Data Common => Spring Data JPA** ko liên quan gì tới **Hibernate**
  + Những thay đổi của entity trên Persistence context thực sự chưa được lưu xuống database ngay lập tức, chúng phải thực hiện thêm 1 việc nữa đó là **flushing:** là quá trình synchronize state của entity giữa persistence context với database.
  + **Flushing** là việc biến những thay đổi trên entity thành thao tác insert, update, delete,… xuống database. Việc này cho phép Hibernate gom những thay đổi nó vào một **transaction,** điều này sẽ giúp hibernate có thể rollback lại về state ban đầu nếu như xảy ra lỗi. (hoặc là thực hiện toàn bộ hành động, hoặc là ko làm gì cả)
  + Khai báo annotation **@Transactional** ở trên method hoặc trên class (áp dụng cho toàn bộ method trong class). Khi đặt annotation này, mọi xử lí trong method sẽ được nhét vào trong 1 **transaction**.
  + **‘**