NOTE: Một số trang tài liệu:

* **Spring, maven**
* spring.io/projects
* Maven chỉ là công cụ giúp build dự án

<https://maven.apache.org/what-is-maven.html#mavens-objectives>

* <https://docs.spring.io/spring-framework/docs/3.0.x/spring-framework-reference/html/overview.html>

Bài 1: Getting started:

1. **Spring**

* **?**
* Spring Boot: giúp set up project
* Spring Boot, Spring MVC, Spring REST,… là hệ sinh thái của spring
* SB là móng, còn các cái khác là gạch, đá, ….

1. **Maven**

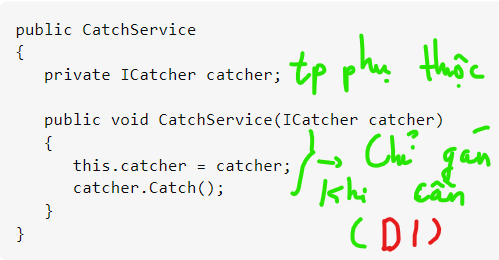
* **?**
* Maven chỉ là công cụ giúp build dự án
* Ko cần thiết phải cài đặt maven cho dự án, nếu muốn dùng maven ở mức global thì phải cài đặt
* Nhờ có 2 file mvnw, mvnw.cmd giúp sử dụng các câu lệnh của maven k cần cài thủ công
* ./mvnw spring-boot:run : build dự án

1. **The POM file**

* **?**
* <https://maven.apache.org/guides/introduction/introduction-to-the-pom.html>
* **Spring beans**
* An instance of a class managed by the Spring container
* **Inversion control và Dependency injection**
* Inversion control (IoC): quản lý đảo ngược tiến trình
* Dependency injection (DI): công cụ để thực hiện IoC
* Vd A->B->C:

A phụ thuộc B và C, B thay đổi thì A và C cũng thay đổi

Spring sẽ khởi tạo A và C trước (Đây gọi là inversion control tức là đảo ngược cái thứ tự), B khi nào cần sẽ được đưa vào sau – cái công việc khi nào cần mới đưa B vào này gọi là DI (tiếng việt là tiêm cái thành phần phụ thuộc vào, nôm na là đưa cái thành phần phụ thuộc vào )



Trong ví dụ trên catcher là tphần phụ thuộc, nó vẫn được khai báo nhưng chỉ khi cần thì mới được gán vào ở method CatchService

* **Injector**
* Việc này do spring làm hết
* **Spring Boot dev tools**
* xml

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

</dependencies>

* **Set up MySQL với Spring**
* https://spring.io/guides/gs/accessing-data-mysql
* **Dependencies:**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.33</version>  
</dependency>

* **Properties:**

spring.jpa.hibernate.ddl-auto=update  
spring.datasource.url=jdbc:mysql://${MYSQL\_HOST:localhost}:3306/db\_example  
spring.datasource.username=springuser  
spring.datasource.password=ThePassword  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
#spring.jpa.show-sql: true

* **Viết code tightly coupled và loosely coupled**
* <https://stackoverflow.com/questions/226977/what-is-loose-coupling-please-provideexamples>
* Tightly coupled:
* các module liên quan, ràng buộc chặt chẽ với nhau, khi thay đổi một module thì các module còn lại cũng phải thay đổi theo
* Đặc điểm: khó thay đổi và khó bảo trì
* Loosely coupled:
* các module độc lập hơn
* Đặc điểm: dễ dàng thay đổi và mở rộng, ko ảnh hưởng các module còn lại
* Các module thường ít biết chi tiết về nhau, thường tương tác với nhau thông qua interface hoặc abstract class

Bài 2: Spring Security

* **Tổng quan:**
* Các khái niệm liên quan
* Authentication
* Authorization
* Spring Security Filter Chain
* Các dạng security
* Form authentication
* Basic authentication
* jwt authentication
* Các lỗi security thường gặp
* CFRS, CORS...
* **Authentication (xác thực)**
* Trả lời câu hỏi who are you ? (quá trình ktra, xác nhân danh tính của user/device trước khi cho phép truy cập vào resources)
* Gắn liền với việc đăng nhập vào hệ thống
* Cách thức hoạt động
* Nhập thông tin xác thực (username + password(đăng nhập)
* Kiểm tra tt (hệ thống)
* Cấp quyền (hệ thống)
* Các cách: sử dụng username/password hoặc biometric
* **Password storage:**
* Spring Security’s PasswordEncoder interface let pw be stored securely
* Rainbow table: kỹ thuật tấn công bẻ khóa các hàm băm
* Giải pháp: sử dụng salted password (thêm 1 chuỗi ngẫu nhiên vào trc khi băm)
* Recommend
* Use adaptive one-way functions: that should be used include bcypt, PBKDF2, scrypt and argon2
* Exchange long term credentials for short term credential
* DelegatingPasswordEncoder
* Default password storage configuration
* Solve all problem of BcryptPasswordEncoder (algo encode pw use hash function prevent bruteforce attack)
* Create a default DelegatingPwEncoder

PasswordEncoder passwordEncoder =

PasswordEncoderFactories.createDelegatingPasswordEncoder();

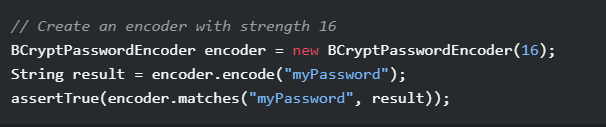
* DelegatingPasswordEncoder Storage Format

{id}encodedPassword

id: bộ mã hóa(PwEncoder) nào đã được sử dụng để encode pw vd(bcrypt, noop, ..)

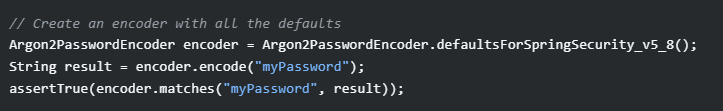
encodedPw: bộ mã đã đc mã hóa

* BcryptPasswordEncoder



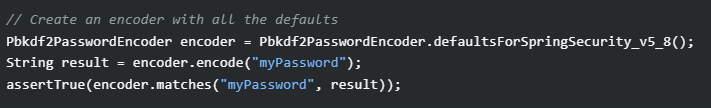
* Argon2PasswordEncoder

Require a large amount of memory



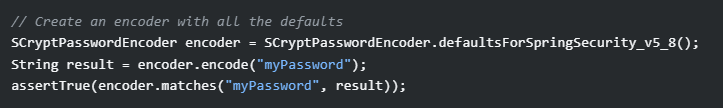
* Pbkdf2PasswordEncoder

good choice when FIPS certification is required.



* ScryptPasswordEncoder

requires large amounts of memory



* **Authorization (phân quyền)**
* Trả lời câu hỏi What can I do ? (là bước sau khi authentication)
* Phương pháp phân quyền:
* Phân quyền dựa trên vai trò (Role-Based Access Control – RBAC)
* Phân quyền dựa trên thuộc tính (Attribute-Based Access Control - ABAC)
* Phân quyền theo chính sách (Policy-Based Access Control)
* **Nguyên tắc thiết kế Security cho hệ thống Web**
* Trust nothing
* Validate every request
* Luôn xác thức “data” gửi vào hệ thống
* Xác định quyền hạn người dùng cho hệ thống
* Nguyên tắc (khi đi làm)
* Nhiều lớp security: firewall, middleware
* Kiến trúc đơn giản, dễ maintain
* **Protection against exploits**
* **CSRF**
* No way to reject evil website’s request or only allow bank’s web request
* To do:

Ensure there is something in request that evil site is unable to provide -> differentiate

* Spring provides two mechanism to protect against CSRF
* The Synchronizer Token Pattern (mô hình token đồng bộ)
* Specifying the SameSite Attribute on your session cookie
* Both protections require Safe Method be Read-only
* Synchronizer Token Pattern
* in addition to our session cookie, a secure random generated value called a CSRF token be present in HTTP request
* CSRF token not automatically included by the browser
* Requiring the actual CSRF token in a cookie does not work because cookies are automatically included in the HTTP request by the browser.
* CSRF token in an HTTP parameter or an HTTP header will work
* **Set up Spring security:**
* <https://docs.spring.io/spring-security/reference/getting-spring-security.html>

<dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-security</artifactId>

</dependency>

* **Spring MVC + Spring security**
* Spring MVC hoạt động:
* Request => Dispatcher Servlet => Controllers... => Response (client)
* Dispatcher Servlet : là 1 lớp "đứng giữa", phụ trách nhiệm vụ trỏ tới đúng  
  controller/method để xử lý request
* Spring MVC + Spring Security
* Request => Spring Security => Dispatcher Servlet => Controllers... => Response
* Tất cả request cần đi qua Spring Security đầu tiên

Bài: Spring MVC

* **Annotaion**
* Note or comment to explain
* Link: <https://stackoverflow.com/questions/2798181/what-are-annotations-and-how-do-they-actually-work-for-frameworks-like-spring>
* Config username, password
* <https://docs.spring.io/spring-boot/appendix/application-properties/index.html#appendix.application-properties.security>
* Ignore random generate pw when access

#config spring security

spring.security.user.name=admin

spring.security.user.password=123

* Disable security
* Temporary disable

@SpringBootApplication(exclude = org.springframework.boot.autoconfigure.security.servlet.SecurityAutoConfiguration.class)

+ exclude: loại đi >< include

* **Áp dụng mô hình MVC**
* <https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html>
* https://gitlab.com/public-starter-projects1/000-java/01-java-spring-mvc/02-java-react-with-jhipster
* Common anotation in MVC:
* <https://www.baeldung.com/spring-mvc-annotations>
* @Controller, @Service
* Render html
* <https://docs.spring.io/spring-framework/reference/web/webmvc/mvc-config/static-resources.html>
* Static file
* Đc lưu trong hệ thống, ít thay đổi, đc public để bên ngoài truy cập vào
* Sử dụng:

B1: @Controller để báo hiệu đây là 1 class trong MVC

B2: tạo view

* View là html
* src/java/resources/static

B3: controller return ra file html muốn hiển thị

* Dynamic file
* **View engine**
* Vđề:
* Vs static file, nội dung ko đổi => cần code từ a-z
* Dữ liệu trang html cần “dynamic” vì đôi khi data sẽ thay đổi
* Định nghĩa:
* Công cụ giúp code ít đi, hiệu quả hơn so với code html truyền thống
* Giải quyết bài toán dynamic

Truyền data từ controller sang view(html) để render

Sử dụng if else, for,.. trong view

* Với Spring, ta có: Spring view Technologies:

<https://docs.spring.io/spring-framework/reference/web/webmvc-view.html>



* **Set up JSP**
* https://howtodoinjava.com/spring-boot/spring-boot-jsp-view-example/
* Setup dependencies

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

artifactId>tomcat-embed-jasper</artifactId>

</dependency>

<dependency>

<groupId>jakarta.servlet.jsp.jstl</groupId>

<artifactId>jakarta.servlet.jsp.jstl-api</artifactId>

</dependency>

<dependency>

<groupId>org.glassfish.web</groupId>

<artifactId>jakarta.servlet.jsp.jstl</artifactId>

</dependency>

* Config view folder

spring.mvc.view.prefix=/WEB-INF/view/  
spring.mvc.view.suffix=.jsp

ý 2: tạo folder **config**

@Configuration

@EnableWebMvc

**public** **class** WebMvcConfig **implements** WebMvcConfigurer {

@Bean

**public** ViewResolver viewResolver() {

**final** InternalResourceViewResolver bean = **new** InternalResourceViewResolver();

bean.setViewClass(JstlView.**class**);

bean.setPrefix("/WEB-INF/jsp/");

bean.setSuffix(".jsp");

**return** bean;

}

@Override

**public** **void** configureViewResolvers(ViewResolverRegistry registry) {

registry.viewResolver(viewResolver());

}

}

* **JSP và JSTL**
* <https://docs.spring.io/spring-framework/reference/web/webmvc-view/mvc-jsp.html>
* Cú pháp JSP
* <https://www.tutorialspoint.com/jsp/jsp_syntax.htm>
* Định nghĩa model: <https://stackoverflow.com/questions/18486660/what-are-the-differences-betweenmodel-modelmap-and-modelandview>
* Different Model, ModelMap, ModelAndView

Model is interface, truyền data từ controller->view

ModelMap is class, a glorified Map

ModelAndView is a container for both a ModelMap and a view object

* Display a variable: ${variableName}

* **JSTL – JSP Standard Tag Library**
* **<%@page contentType="text/html" pageEncoding="UTF-8"%>**

**<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>**

* A page directive (<%@page ... %>) sets the content type returned by the page.
* Tag library directives (<%@taglib ... %>) import custom tag libraries.

Bài: JSP và JSTL

* **View resolvers**
* placing JSP files in a directory under the 'WEB-INF' directory so there can be no direct access by clients.
* JSP vs JSTL: special class JstlView
* **Spring’s form tag library**
* Configuration:
* The library descriptor is called spring-form.tld

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

form is the tag name prefix you want to use for the tags from this library.

* Form tag
* This tag render a HTML form element, expose binding path to inner tag for binding
* Input tag
* Render HTML input element with bound value and type = ‘text’ by default

<form:input path="lastName"/>

* Checkbox tag

<form:checkboxes path="preferences.interests" items="${interestList}"/>

* Radiobuttons tag

<form:radiobutton path="sex" value="F"/>

<form:radiobuttons path="sex" items="${sexOptions}"/>

* Password tag

<form:password path="password"/>

<form:password path="password" value="^76525bvHGq" showPassword="true"/>

* Select tag

<form:select path="skills" items="${skills}"/>

* Textarea tag

Submit text

<form:textarea path="notes" rows="3" cols="20"/></td>

* Hidden tag

 renders an HTML input tag with the type set to hidden

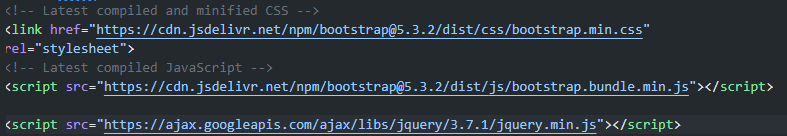
<form:hidden path="house"/>

* Errors tag

 renders field errors in an HTML span element

<form:errors path="lastName"/>

* **Tích hợp jQuery và Boostrap**
* Online



* Sử dụng css, js
* <https://stackoverflow.com/questions/25098413/whats-the-diff-btween-src-mainresources-and-src-main-webapp-resources>
* Tạo webapp/resources

Muốn viết view tĩnh thì viết vào resources bên trong, view động thì resources bên ngoài

* Cấu hình

<https://www.baeldung.com/spring-mvc-static-resources>

* **Send data with HTML Form**
* Create form having 2 attributes: method and action
* Solve method POST

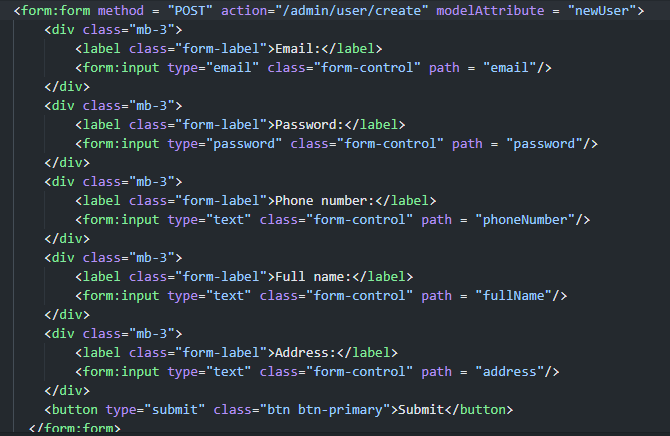
@RequestMapping(value = "/admin/user", method = RequestMethod.POST)

* Create form with JSTL

<%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>

<https://www.baeldung.com/spring-mvc-form-tutorial>

* **@ModelAttribute** Annotation : là cách chúng ta convert dữ liệu từ **View** trả cho **controller** xử lý
* Forms in Spring MVC



* modelAttribute = “newUser”: newUser sẽ có các thuộc tính là các input trong form

Bài: Spring Data với JPA và Hibernate

* **Spring Data**
* <https://spring.io/projects/spring-data>
* Cung cấp đối tượng repository để truy cập database
* **Spring Data với JPA**
* JPA (Java Persistence API) : <https://docs.spring.io/springdata/jpa/reference/index.html>
* Cài đặt:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>

<https://github.com/spring-projects/spring-boot/blob/main/spring-boot-project/spring-boot-starters/spring-boot-starter-data-jpa/build.gradle>

* **Cách Read/Write Data**
* Với database chúng ta có 2 thao tác: READ(đọc), Write(update/insert/delete)
* Có 2 cách:

Sử dụng JDBC

Sử dụng ORM/ODM

* ORM: object Relational Mapping

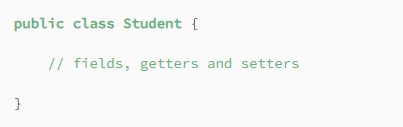
Truy cập database thông qua các object định nghĩa trong code

Vd: all Users = findAll()

Ưu: ko cần biết truy vấn SQL, gần với natural language

Nhược: hiệu suất k cao, muốn kiểm soát tốt hiệu năng cần dùng SQL

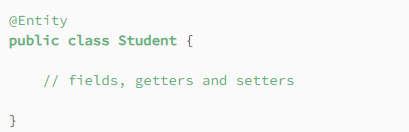
* **Hibernate vs Spring Data JPA**
* <https://www.youtube.com/watch?app=desktop&v=4Py9RTVWyvE>
* <https://docs.jboss.org/hibernate/orm/6.4/introduction/html_single/Hibernate_Introduction.html#introduction>
* **Entity**
* <https://www.baeldung.com/learn-jpa-hibernate>
* <https://spring.io/guides/gs/accessing-data-jpa>
* POJO (Plain Old Java Objects): class java thuần túy k có framework gì
* Represent a table stored in db, instance of an entity reprensenta row in table
* ***Entity* anotation**
* We have a POJO called *Student* and would like to store in db

**

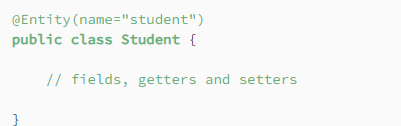
* Todo this, we should define an entity so that JPA is aware of it

Define by making use of the ***@Entity* annotation,** specify this anotation at the class level

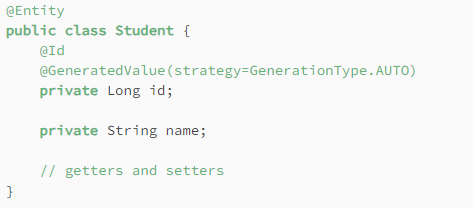
* Must ensure that the entity has a o-arg contructor and a primary key



* The entity name defaults to the name of class, change it by using the *name* element

****

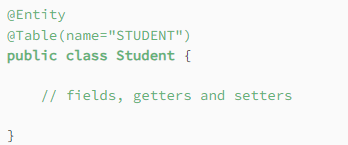
* Various JPA implement will try subclassing our entity to provide their functionality, **entity classes must not be declared *final***
* ***Id* anotation**
* Each JPA entity must have a primary key uniquely identifies it, the   
  ***@Id*** annotation defines the primary key, we can generate identifiers in different ways, which are specified by the ***@GeneratedValue*** annotation
* Can choose from four id generation strategies with *strategy* element. The value: AUTO, TABLE, SEQUENCE, or IDENTITY



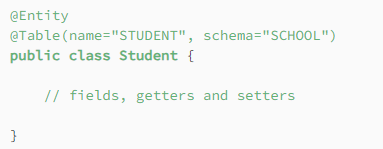
GenerationType.AUTO, JPA provider use any strategy it wants to generate identifiers

if we anotate the enity’s fields, JPA provider use these fields to get and set the entity’s state

* ***Table* Annotation**
* In most cases, the name of the table in the db and the name of the entity won’t be the same
* In these cases, can specify the table name using the ***@Table*** annotation

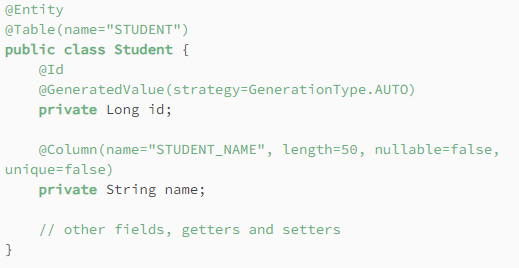


Can also mention the schema using the *schema* student



If we don’t use the ***@Table*** *annotation, the name table will be the name of entity*

* ***Column* Annotation**
* Mention the details of a column in table
* Has many elements such as *name, length, nullable and unique*.



*name:* specifies the name of the column in the table

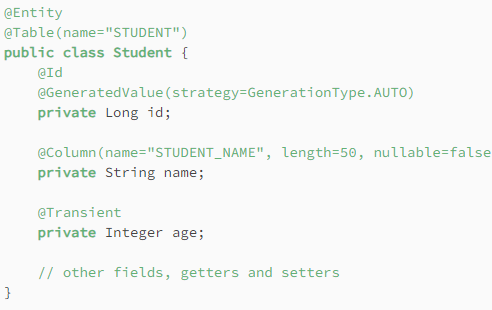
*length:* its length

*nullable:* nullable or not

*unique:* column is unique

if we don’t specify this annotation, the name of the column in the table will be the name of the field

* ***Transient* Annotation**
* Want to make a field non-persistent (k lưu trữ), use ***@Transient*** *annotation*, the field won’t be persisted

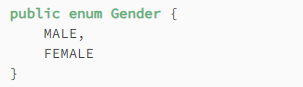


The field *age* won’t be persisted in the table

* ***Temporal* Annotation**
* In some case, we may have to save temporal values(giá trị thời gian) in our table
* Support for: *java.util.Date, java.util.Calendar, java.time.LocalDate, java.time.LocalTime, java.time.LocalDateTime*, java.time.OffsetTime and java.time.OffsetDateTime,
* Three values: TemporalType.DATE, TIME, TIMESTAMP



* ***Enumerated Annotation***
* Persist a Java *enum* type
* ***@Enumerated*** annotation specify whether the enum should be persisted by name (string) or by ordinal(default – số nguyên)

****

****

Values: EnumType.STRING, EnumType.ORDINAL

* **Mô hình MVC áp dụng với Spring Data**
* Domain driven-design:
* <https://tech.cybozu.vn/trai-nghiem-chuyen-sau-domain-driven-design-6eef4/>
* Là một cách tiếp cận để phát triển phần mềm
* Một project trên 30 use cases thì mới nên apply DDD.
* Chia làm 2 phần to lớn: Strategic Design and Tactical Desgin,

Strategic : answering question “what” and “why”

Tactical: answering question “how”

Strategic Design sẽ làm trước Tactical Design, cái trước thường là nghiệp vụ, cái sau sẽ sẽ việc code logic thực hiện

* Domain là nghiệp vụ hay là các class, hay entity
* Business rule hay business logic:

Vd: các requirement: mỗi user chỉ có 1 email, admin có thể tạo role, total price >= 0, trạng thái khi đang tạo đơn hàng là pending, ….

Đống này gọi là business logic

Business logic dễ thay đổi và mở rộng vì sản phẩm phải đáp ứng nhu cầu của khách hàng

Business logic sẽ được đặt trong core domain layer. Cụ thể là trong các value object, entity, aggregate, domain service

* DDD tập trung vào domain
* Bouned context (ngữ cảnh giới hạn tức là mỗi ngữ cảnh giới hạn một phần mềm nghiệp vụ với một mô hình riêng, ranh giới rõ ràng)

Vd: user bounded context: chứa logic, nghiệp vụ liên quan tới user, từ ngữ liên quan user, roles

Post bounded context: chứa logic nghiệp vụ liên quan tới các bài post

Khi nói số lượng user xem bài post abc này trong 1 tháng là 100 users. Thì ý nghĩa của user trong post context sẽ khác với user trong user context.

User trong user context đề cập tới user, admin, author, subcriber,.. hay role.

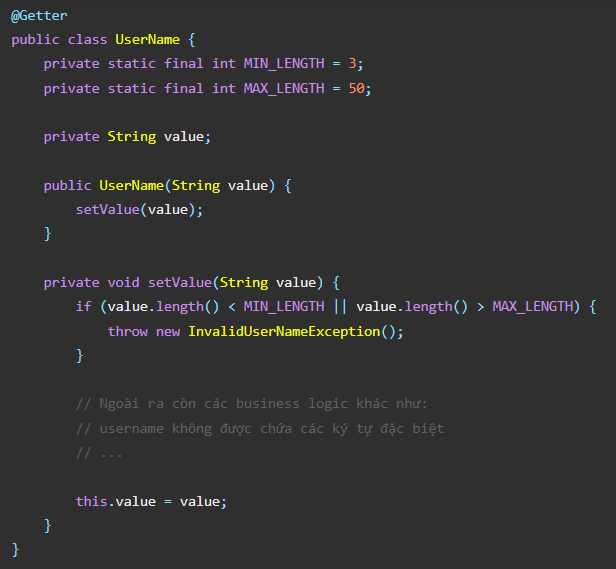
Còn user trong post context đơn giản là user đã xem bài post, k quan tâm tới chi tiết, vai trò của user đó

* Layer architecture: hexagonal/ clean architecture
* Data transfer object (DTO): là object chuyển data đi qua các layer trong 1 vòng đời business flow
* Value object: vùng chứa dữ liệu, immutable tức k có setter

Hiểu đơn giản thì 1 object chứa các value object, value object là các attribute của object

Ko có danh tính riêng, hai value object có các giá trị thuộc tính giống hệt nhau được coi là bằng nhau

Vd: về value object UserName



* Entity

Object giống i hệt value object, khác là có định danh (ID), thường dùng để chứa dữ liệu lưu xuống db, business logic liên quan được đặt trong entity

* Repository

Pattern đẻ giao tiếp với db

* Domain service

Chứa các logic để phục vụ cho layer domain đó. Logic ở đây chính là business logic

khi ko biết đặt business logic ở đâu, tức là k đặt đc trong entity, value object hay aggregate thì hãy nghĩ đến domain service

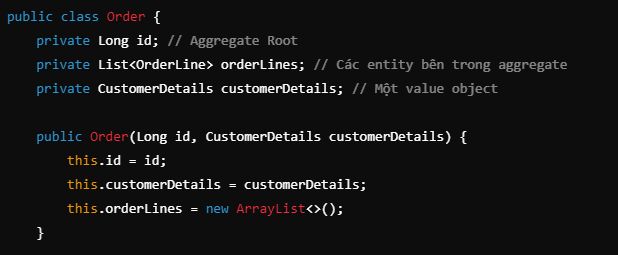
* Aggregate

Giải quyết inconsistency data trong business flow

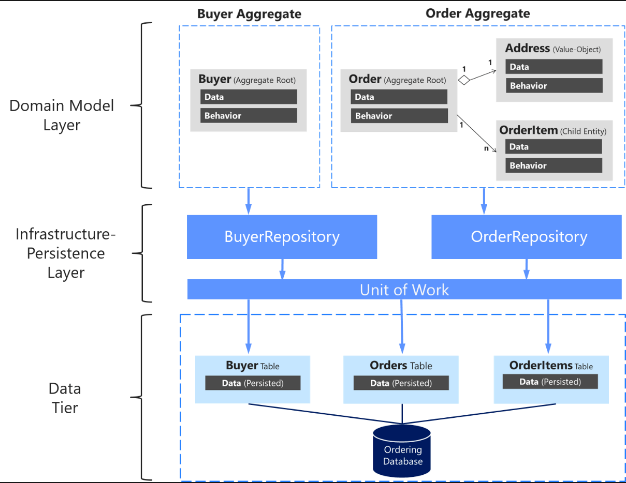
Là tập hợp nhiều value object, entity khác nhau

Thường được biểu diễn dưới dạng 1 lớp, có aggregate root và các thành phần

Vd:



* Repository pattern
* <https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design>



* **Repository**
* <https://docs.spring.io/spring-data/jpa/reference/jpa/getting-started.html>
* Repository sẽ là interface, chứa các phương thức CRUD
* Controller sẽ nhận request và gọi tới service (hiểu đơn giản nó chỉ là người quản lý chia việc cho ai làm)
* Serivce là nơi xử lý các logic
* **Repository query**
* Derived query method: phương thức truy vấn tự động
* Sử dụng JpaRepository vì nó trả về List, còn CrudRepository trả về các Iterable
* <https://docs.spring.io/spring-data/jpa/reference/repositories/query-methods-details.html>
* Defining Query Methods:
* Repository proxy (lớp trung gian Spring tạo ra cung cấp chức năng truy vấn dữ liệu không cần viết mã thủ công)
* Repository proxy has two ways to derive a store-specific query from the method name:

Deriving the query from the method name directly

Using a maunally defined query

* Query lookup Strategies:

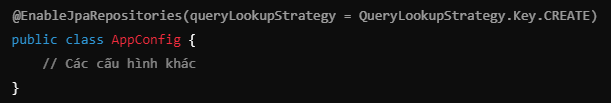
queryLookupStrategy attribute of the EnableJpaRepositories annotation.

CREATE: construct a query from the query method name

USE\_DECLARED\_QUERY: tries to find a declared query, throws an exception if it cannot find one (commonly with @Query annotation)

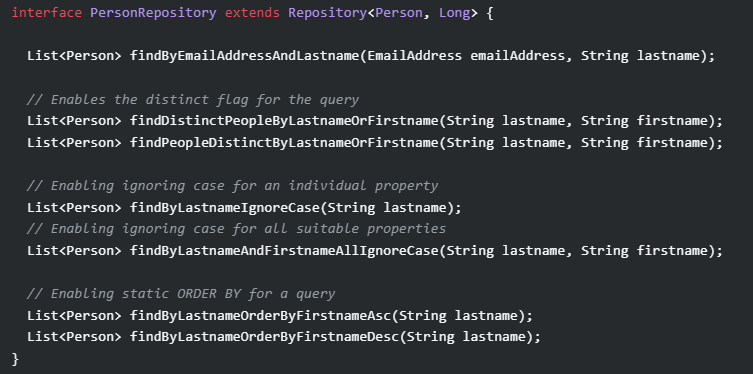
CREATE\_IF\_NOT\_FOUND: combine 2 strategies above

Eg:



* Query creation:

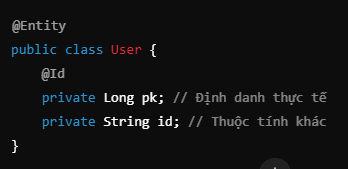
Query cretion from method name:



Parsing query method

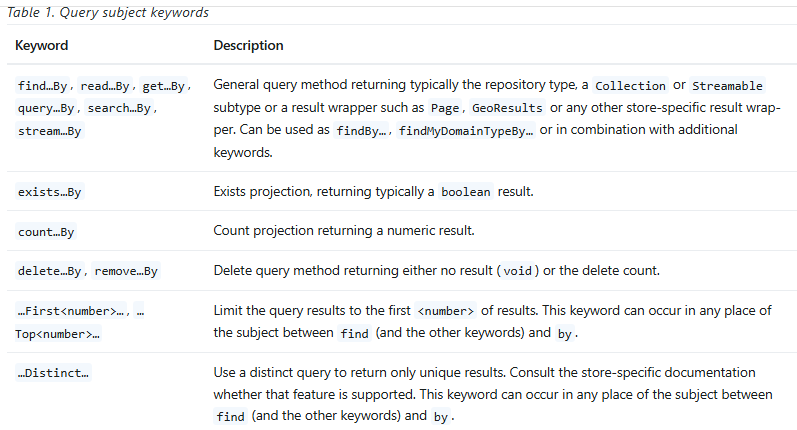
* Name divided into subject and predicate
* First part: find…By, exists…By, define subject, second part forms predicate
* Any text between find and By: such as distinct, …
* Reserved Method Name (dành riêng)

method names inherited from the base repository targeting the identifier property



ở ví dụ này: khi ta gọi findById thì spring sẽ hiểu là muốn tìm kiếm dựa trên pk (vì nó là thuộc tính định danh thực sự) chứ k phải là tìm kiếm theo id. Vậy nên muốn tìm kiếm theo id, cần phải đặt tên query rõ ràng

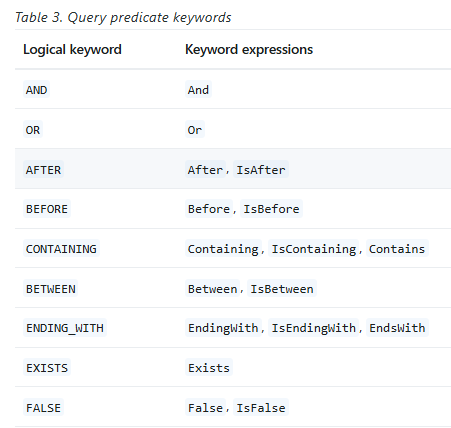
* <https://docs.spring.io/spring-data/jpa/reference/repositories/query-keywords-reference.html>
* **Repository query key word**
* Supported query method subject keywords

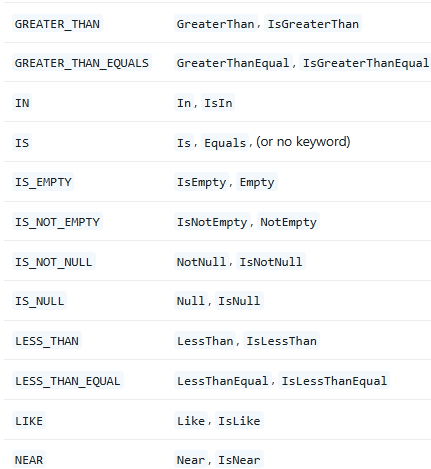


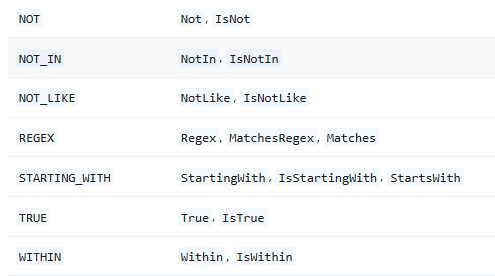
* Reserved method

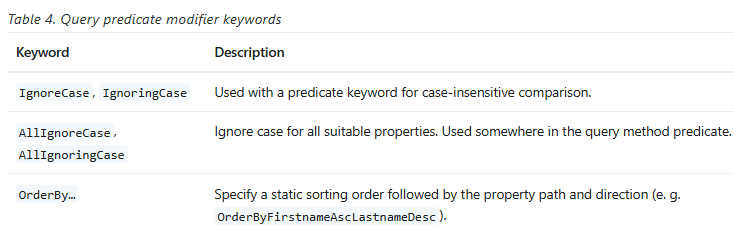


* Trong Spring Data, các từ khóa và các modifier hỗ trợ để xây dựng truy vấn tự động (query derivation)





****

****

* **Nhận kết quả trả về từ repository**
* <https://www.baeldung.com/spring-data-jpa-findby-vs-findoneby>
* Ko có sự khác biệt giữa findBy, findOneBy hay findAxzBy, chúng để có thể trả về 1 object hoặc 1 list object

Vì Spring data sẽ ignore hết các từ ở giữa hai từ find và By

* Vì thế ta có thể thêm các từ vào giữa find và By để làm rõ nghĩa query method
* Spring Data quyết định return 1 hoặc 1 collections dựa trên kiểu trả về của query method
* Tuy nhiên, nếu ta để kiểu trả về của query method là một object, nhưng truy vấn sql lại có thể chọn ra nhiều record với tiêu chí đó thì sẽ sinh ra ngoại lệ IncorrectResultSizeDataAccessException
* findAll luôn return 1 array, nên khi sử dụng kiểu trả về là list thì sẽ bị lỗi => ta cần giới hạn phần tử trả về cho nó bằng cách dùng findFirst/ findTop1

Bài 4: Vòng lặp

* **Lớp StringBuilder**
* Được sử dụng vào việc chèn, xóa, thêm kí tự trong xâu
* Khai báo:
* **StringBuilder sb = StringBuilder(s);**
* Method:
* toString(): chuyển từ stringBuilder về String
* reverse();
* insert(index, <thứ cần chèn>):

thứ cần chèn có thể là số, string, chuỗi kí tự, boolean, kí tự..

* delete(stIndex, endIndex):
* replace(stIndex, endIndex, str):

thay thế đoạn từ (st, end) bằng str

vd: s.replace