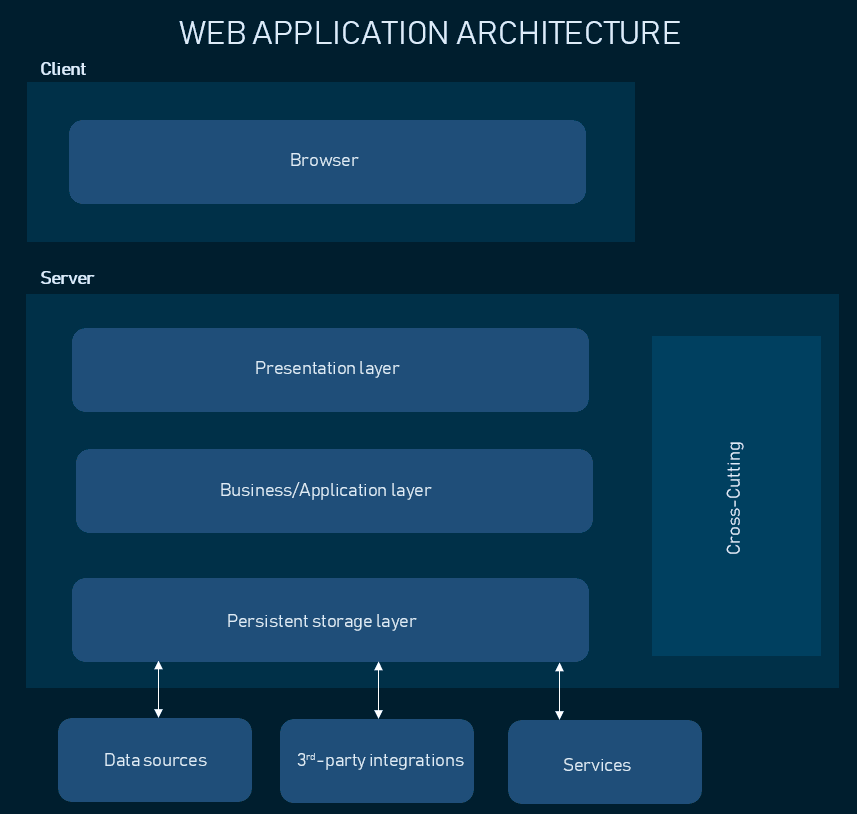
# Spring introduction

## Spring framework



Diagram, table

Description automatically generated

## Spring products

# Spring core framework

## DI and IoC

Ioc: (Inversion of Control): Đảo ngược điều khiển, nó giúp thay đổi luồng điều khiển của chương trình một cách linh hoạt

* IoC means that objects **do not create** other objects on which they rely to do their work. Instead, they get the objects that they need **from an outside source**

**DI:** (Dependency Injection): is a form of IoC, where implementations are passed into an object through constructors/setters/service lookups( namespace, configuration)

* **Transferring the task of creating the object to someone else** and directly using the dependency is called dependency injection.

\*Note: **IoC Container (**tạo ra các object – Spring Bean**)** is a specific kind of **DI**. **DI** is a specific kind of **IoC**

## POJO and Contructors

### Constructors and Setter Injections

**Contructors Injection**

**Text

Description automatically generated**

**🡪**Hạn chế tính extendable and flexible 🡪 **POJO**

**Setter injection**

**Graphical user interface, application

Description automatically generated**

**Trong java spring có 3 cách để thực hiện DI:**

* **XML configuration:** dùng
  + thẻ **property** (setter injection)
  + **constructor-arg** (constructor injection)
  + thuộc tính **autowire** 
    - **default**-contructor injection(no parameter)
    - **constructor**- constructor injection (have parameter)
    - **byName** gọi constructor no parameter and then call setter
    - **byType** tương tự **byName** nhưng nếu có 2 thẻ cùng type sẽ lỗi
* **Annotation based configuration:** @Service, @Repository, @Controller
  + **@Autowired:** contructor injecttion, setter injection
* **Java based configuration:** @Configuration, @Bean (thêm @ComponentScan)
  + **@Autowired**

### POJO

* Plain Old Java Object
* Là 1 class mà không chịu sự chi phối của bất kỳ framework cụ thể nào
* POJO is a Java object **not bound by any restriction** other than those forced by the Java language specification

**JavaBean** về cơ bản vẫn là POJO nhưng cần thỏa mãn các yêu cầu sau:

* Access levels: các thuộc tính trong class đều được đặt private và chỉ được truy suất thông qua getter và setter
* Impliments Serializable interface (giúp truyền qua mạng or đọc ghi file)
* Default constructor: chứa 1 constructor không tham số

## Spring IoC Container

* Spring IoC container là trái tim của Spring

Diagram

Description automatically generated

* Quản lý vòng đời của bean, khởi tạo, cấu hình và tương tác giữa các bean
* Có thể cấu hình Spring IoC bằng
  + XML
  + Java code
  + Java annotation
* Có 2 loại IoC Container:
  + BeanFactory (các bean được tạo ra khi gọi getBean())
  + ApplicationContext (khi container được start thì bean cũng được tạo 🡪 không cần chờ gọi phương thức getBean() )

## Spring Beans

Là những Java Object mà từ đó tạo nên khung sườn của 1 Spring application

Khởi tạo object thông qua Spring container được gọi là Spring Bean

## Beans life cycle

Diagram

Description automatically generated

Graphical user interface, diagram

Description automatically generated with medium confidence

Ways to implement the life cycle of a bean:

* XML: in bean tag **init-method**="method\_name" **destroy-method**="method name"
* Implement *InitializingBean, DisposableBean* interface and override afterPropertiesSet() and destroy() method
* Using annotation: @PostConstruct, @PreDestroy and in xml file

<**bean** class=*"org.springframework.context.annotation.CommonAnnotationBeanPostProcessor"*/>

# SPRING WITH DATABASE

## ORM, JPA

**O**bject **R**elational **M**apping: kỹ thuật cho phép chuyển đổi từ các object trong ngôn ngữ lập trình 🡪 database và ngược lại

* Trong java nó được thực hiện với sự trợ giúp của **Reflection and JDBC**

Diagram

Description automatically generatedJPA (Java Persistence API): là một đặc Java cho việc ánh xạ giữa các đối tượng java với relational database sử dụng công nghệ (hướng tiếp cận) **ORM**

* JPA cung cấp đầy đủ các công cụ cho phép thao tác với database **nhanh chóng và đơn giản**
* JPA dùng để **persist** POJO 🡪 database và get data from database and maps to POJO

### JPA Structure:

Graphical user interface

Description automatically generated

* EntityManagerFactory: khởi tạo và quản lý EntityManager
* EntityManager: (interface) cung cấp các API cho việc tương tác với Entity
  + Persist, Merge, Remove
  + Tạo EntityTransaction
* Persistence: định nghĩa một tập các Entity class được quản lý bởi 1 instance của EntityManager
  + Javax.persistence.Persistence: bao gồm các phương thức static để get instance của EntityManagerFactory
* EntityTransaction: có quan hệ 1-1 với EntityManager
  + Bất kỳ thao tác nào được bắt đầu thông qua đối tượng EntityManager đều được đặt trong Transaction
* Query

Diagram

Description automatically generated

### Mapping Directions:

* **Unidirectional relationship**
* **Bidirectional relationship**

### Types of mapping:

* One-to-one
* One-to-many
* Many-to-one
* Many-to-many

### JPA Entity Manager

Important roles of an entity manager:

* The entity manager implements the API and encapsulates all of them within a single interface
* Used to read, delete, write an entity
* An object referenced by an entity (@Entity) is **managed by entity manager**

# SPRING BOOT

## Goals:

Provide common non-functional features:

* Embedded servers
* Metrics
* Health checks
* Externalized configuration

🡪bổ sung cho Spring MVC, là phiên bản nâng cấp của Spring MVC 🡪 giúp cấu hình nhanh hơn

🡪là cơ sở cho phát triển spring cloud - microservice

# Example

## Spring Bean, Spring Component: cần phân biệt được @Bean & @Component

|  |  |
| --- | --- |
| @Bean | @Component |
| **Explicitly** declare a single bean | It will automatically detect by using classpath scan |
| Method level | Class level |
| Specific implementation depending on dynamic state. We can control instance creation logic. | Not impossible |
| @Configuration needed |  |
| For 3rd party libraries, we don’t have source code. There @Bean is not possible | Preferable for component scanning and auto-wiring |

## Spring property, Spring config: biết cách get value từ file config, cần biết cách sử dụng nhiều file config (dev, alpha, beta, stg, prod)

## IOC Containers, DI: cần phân biệt được các cách Inject/Autowired

## Bean Scopes, Bean Life cycle

### Bean Scopes

* singleton: a single instance per Spring IoC Container
* prototype: a single bean to have a number of object instannces
* request: dùng cho ứng dụng web, một instance của bean sẽ được tạo cho mỗi HTTP Request
* session: mỗi instance của bean sẽ được tạo cho mỗi HTTP Session
* global-session: được sử dụng để tạo global session bean cho các ứng dụng Portlet

### Bean life cycle:

* Khi IoC Container (ApplicationContext) tìm thấy một Bean cần quản lý 🡪 **khởi tạo bằng constuctor (không tham số)**
* Inject dependencies vào Bean bằng **setter** và thực hiện một số quá trình cài đặt khác vào Bean như setBeanName, setBeanClassLoader, …
* Hàm **afterPropertiesSet() (@PostConstruct)** được gọi
* Some code
* Bean **sẵn sàng** để hoạt động
* Nếu IoC Container không quản lý Bean nữa hoặc bị shutdown sẽ gọi hàm **destroy() (@PreDestroy)** trong Bean
* **Xóa** Bean

## Entity, ORM, Relationship, Fetch Type (Lazy vs Eager) : cần phân biệt được các loại Cascade, các FetchType, relationship

Vấn đề khi dùng thư viện JDBC để kết nối tới cơ sở dữ liệu: phải tương tác thông qua ResultSet cho mỗi lần thực hiện truy vấn từ java 🡪 hướng tiếp cận **ORM** (Object Relational Mapping)

Java Persistence API (**JPA**)

ORM Framework: (các triển khai của **JPA**)

* Spring Data JPA (JPA)
* Hibernate
* EclipseLink
* Ibatis (MyBatis)

Một số annotation:

* @Entity: Entities in JPA are nothing but POJOs representing data (ensure that the entity has a **no-arg constructor and a primary key**)
  + The entity name defaults to the name of the class
* @Id: define the primary key
  + Generated the id by the @GeneratedValue (**AUTO/TABLE/SEQUENCE/IDENTITY**)
* @Table
* @Column: name=””, length=, nullable=,unique=
* @Transient: make **a field non-persistent** (Eg: age)
* @Temporal: in some cases, we may have to save temporal values in our table
* @Enumerated: to specify whether the *enum* should be persisted by name or original

### Các trạng thái của Entity

Diagram

Description automatically generated

1. **New (Transient)** state

* An object that is newly created and has never been associated with JPA Persistence Context
* Is not stored in the database

1. **Persistent (JPA managed)** state

* An object that is associated with *persistence context*
* Any changes made to objects in this state are **automatically propagated** to databases

1. **Detached (unmanaged)** state

* An object becomes detached when the currently running Persistence Context is closed🡪 any changes made to detached object **are no longer automatically propagated** to the database
* Use detach() method

1. **Removed** state

* Removed object are deleted from the database

### Cascade

* ALL: bao gồm tất cả các Cascade (bao gồm cả các Cascade được Hibernate hỗ trợ như REPLICATE, SAVE\_UPDATE, LOCK)
* PERSIST: làn truyền hoạt động này một cách tự động từ persistent entity này 🡪 các related entity (Eg: thêm, xóa)
* MERGE: giúp hợp nhất detached entity với persistence entity cùng id (Eg: update)
* REMOVE: khi một entity bị xóa trong database 🡪 related entity cũng bị xóa
* REFRESH:
* DETACH: xóa một entity ra khỏi persistenct context🡪 related entity cũng sẽ bị loại bỏ khỏi persistence context
* REPLICATE: dùng trong trường hợp có nhiều data source và đồng bộ dữ liệu
* SAVE\_UPDATE: giống PERSIST và MERGE
* LOCK: dùng để chuyển detached và related entity vào persistence context

### FetchType

|  |  |
| --- | --- |
| LAZY | EAGER |
| Chỉ tải dữ liệu được gọi. Không gọi các dữ liệu liên quan | Tải tất cả các dữ liệu liên quan |
| Default cho quan hệ many-to-many và many-to-one | Default cho quan hệ one-to-one |

### Quan hệ 1-1, 1-N, N-N (@OneToOne, @OneToMany,@ManyToOne,@ManyToMany)

|  |  |
| --- | --- |
| Bidirectional | Unidirectional |
|  |  |
|  |  |
|  |  |
|  |  |

1. 1-1

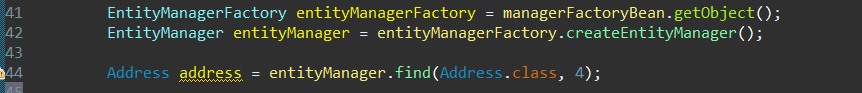
Text

Description automatically generated

Text

Description automatically generated

Test





Với mỗi managed entity, the Persistence Context requires entity type and id, so the **child id must be known when loading parent entity** 🡪 **only way to find the associated Student primary key is to execute a secondary**

🡪 để tối ưu 🡪 dùng shared key với @MapsId (do mỗi student chỉ có 1 address)🡪 can fetch Student using Address id

Student.java

Text

Description automatically generated

Text

Description automatically generated

The best way to map a @OneToOne relationship is to use @MapsId (can fetch the Address by using Student id)

* By specifying the annotation it can populate the PK of Address from Student
* **@OneToOne(optional=false):** if the association is optional 🡪 Hibernate has no way of knowing if an Address exists for a given Student without issuing a query🡪 cant populate the Address filed with a proxy (no Address referencing the Student and cant populate it with null)🡪 optional=false: it trusts you and assumes that an Address exists and it populate the Address field with a proxy

mappedBy: once we have defined the owing side of the relationship 🡪 Hibernate already has all the information it needs to map the relationship in our database 🡪 to make this **association bidirectional** 🡪 we’ll have to do is to define the referencing 🡪 use *mappedBy* attribute

**@JoinColumn**: chỉ ra FK của Address tham chiếu đến PK của Student

**@MapsId:** indicates that the **primary key values will copied from Student entity**

1. 1-N

**Simple**

Sử dụng @ManyToOne ở bảng children 🡪 to take advantage of entity state transitions and the dirty checking mechanism 🡪 map the child entities as a collection in the parent object 🡪 JPA offers the @OneToMany

1. Unidirectional @OneToMany

By default, that’s how the unidirectional @OneToMany association works

Graphical user interface, text, application

Description automatically generated

🡪use @JoinColumn (in parent entity) to figure out that there is a post\_id FK

**Best approach**

**Post\_Comment**

Text

Description automatically generated

Text

Description automatically generated

* The parent entity, Post, features a utility method (addComment) which are used to **synchronize both side of bidirectional** association. You should always provide that method
* The child entity, PostComment, implement the *equals* and *hashCode*. Since we cannot rely on a natural id for equality checks, we need to use the entity id instead for the equals method
* *@OneToMany(*orphanRemoval=true*)*: đảm bảo không có Comment nào mà không tham chiếu đến Post nào

<https://vladmihalcea.com/the-best-way-to-map-a-onetomany-association-with-jpa-and-hibernate/>

1. N-N

Graphical user interface

Description automatically generated with medium confidence

Text

Description automatically generated

Text

Description automatically generated

<https://vladmihalcea.com/the-best-way-to-use-the-manytomany-annotation-with-jpa-and-hibernate/>

## Controller, HTTP Methods (POST, GET, PUT, DELETE) : cần biết các thành phần request, response, các annotation của request

1. Http Request

Diagram

Description automatically generated with medium confidence

1. Http Response

Diagram

Description automatically generated

1. Anntation
   * @RequestBody: lấy dữ liệu trong request body. Khi dùng @RestController trong springboot data trong request body sẽ được chuyển dữ liệu JSON, XML, … 🡪 object (use http message converter)
   * @ReponseBody: lấy dữ liệu trong response body. Khi dùng @RestController trong springboot object 🡪 data trong request body sẽ được chuyển dữ liệu JSON, XML, … (use http message converter)
   * @RestController= @Controller+ @ResponseBody
     + Chỉ có thể trả về data trong response body

## Authentication vs Authorization : tìm hiểu về tầng filter, generate JWT, validate JWT, secure API

## Spring Security, User service

## Service, Scheduler/Cronjob, Transactional

## Email service, Thymeleaf, multilanguage: hiểu các cách config email

### Annotation @Async

## Repository, Spring JPA, Native query, HQL: hiểu các cách query data

### Projection, DTO, native query join table: hiểu cách mapping result ra class ko phải entity có sẵn Paging

## DTO

### Mapstruct, Mapper

## JMS (optional)

## Mock, Unit test, Integration test

## Spring resttemplate vs Webclient