**AUDIT MODULE 4**

**Câu 1. Trình bày cách Spring MVC xử lý request?**

- Khi người dùng gửi request, đối tượng đầu tiên nhận request là dispatcherServlet (front-controller của Spring mvc).

- DispatcherServlet sẽ tìm và điều hướng request đến handler phù hợp (handler ở đây các controller mà LTV tạo ra).

- Handler sẽ nhận các request và gọi phương thức phù hợp để xử lý.

- Sau khi có kết quả, handler thiết lập các model dữ liệu và trả về tên view cho dispatcherServlet.

**-** DispatcherServlet sẽ được viewResolver hỗ trợ để chọn ra view template tương ứng để render dữ liệu và gửi về trình duyệt cho người dùng.

**Câu 2: Trình bày cơ chế Dependency Injection trong Spring?**

- **IoC framework** sẽ có các thành phần có sẵn làm nhiệm vụ tạo, quản lý các đối tượng trong chương trình. IoC sẽ quản lý, phân tích các mối phụ thuộc, tạo các đối tượng theo thứ tự phù hợp nhất và liên kết chúng lại với nhau, theo cách lập trình viên mô tả.

- **Spring Boot** sử dụng các annotation dạng như **@Component** để đánh dấu lên class, chỉ ra rằng class đó cần tạo một module. Ngoài @Component, còn có các annotation khác như **@Repository**, **@Controlller**, **@Service**,... cũng được đánh dấu là module

--------------------------------------------------------------------------------------

- **Dependency Injection:** Là một dạng thực hiện của IoC, bằng cách tiêm (inject) module vào một module khác cần nó.

- Mọi module trong IoC đều được gọi là Dependency. Khi chương trình chạy, IoC sẽ quét tất cả các class đánh dấu dependency, tạo một đối tượng duy nhất (**singleton**), bỏ vào túi gọi là **IoC container**, trong Spring gọi là **Application context**. Các module chứa trong Application context gọi là các **Bean**.

- Nếu khi tạo 1 module, mà module đó cần một module khác phụ thuộc, thì IoC sẽ tìm xem trong IoC Container có không, nếu có thì inject vào, nếu chưa thì tạo mới, thêm vào IoC, sau đó inject vào. Việc tự động inject các dependency (module) như thế gọi là Dependency Injection.

- Khi ứng dụng **Spring Boot** chạy, thì IoC container sẽ thực hiện quá trình như sau:

+ Quét tìm (scan) các class được đánh dấu là Bean, và tạo một object singleton, bỏ vào IoC container.

+ Khi có một Bean phụ thuộc vào Bean khác, thì IoC sẽ tìm trong container, nếu chưa có thì tạo, nếu đã có thì lấy ra và inject vào bean cần nó.

**Câu 3: Có bao nhiêu cách để thực hiện Dependency Injection? Trình bày?**

**- Constructor-based injection:** Dùng inject các module bắt buộc. Các module được inject nằm trong constructor, và được gán lần lượt vào các field. Vd: Module Xe, bắt buộc cần có module Động Cơ.

- **Setter-based injection:** Dùng inject các module tùy chọn. Mỗi module sẽ được inject thông qua setter, nằm ở tham số và cũng gán cho field nào đó. Vd: Module Xe, khi tạo thì ko cần thiết phải có module Chủ Xe.

- **Interface Injection**: Class cần inject sẽ implement 1 interface. Interface này chứa 1 hàm tên Inject. Container sẽ injection dependency vào 1 class thông qua việc gọi hàm Inject của interface đó.

**Câu 4: Framework là gì ? Framework khác Library chỗ nào ?**

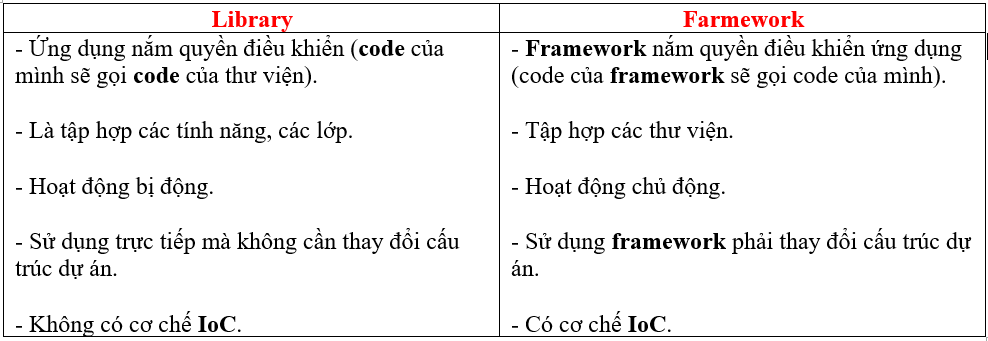
- **Library** - Là một tập hợp các chức năng (functions), các lớp (class) được viết sẳn để có thể tái sử dụng. Mỗi function hoặc class phục vụ cho một công việc cụ thể nào đó.

**- Framework** – là 1 **ứng dụng phần mềm** có tính **trừu tượng** cao, cung cấp các tính năng chung thông dụng nhất và có khả năng tùy biến.

- Phân biệt Framework và Library

+ Giống: Đều cung cấp tính năng viết sẵn để có thể tái sử dụng.

+ Khác: Điểm khác biệt chính giữa framework và library chính là cơ chế IoC



**Câu 5: Spring Framework là gì ?**

- **Spring Framework** - là một trong những **khung ứng dụng** dựa trên **Java** phổ biến nhất (đc **Rod Johnson** phát triển vào năm **2003**).

- Là một **khung mã nguồn mở** có thể được sử dụng để phát triển các ứng dụng **Java** một cách dễ dàng và với tốc độ nhanh chóng.

**-** Spring Framework được xây dựng dựa trên 2 nguyên tắc design chính là: Dependency Injection và Aspect Oriented Programming.

- Nó là một khung công tác nhẹ cũng cung cấp hỗ trợ cơ sở hạ tầng được xác định rõ ràng để phát triển các ứng dụng trong **Java**. Nói cách khác, có thể nói rằng **Spring** xử lý cơ sở hạ tầng để LTV có thể tập trung hơn vào việc phát triển ứng dụng của mình.

**Câu 6: Lợi ích của Spring Framework?**

+ Tính đơn giản và kiểm thử tốt.

+ Đảo ngược quyền điều khiển (**IoC**).

+ Trọng lượng nhẹ (khung cơ bản **Spring** chỉ có kích thước khoảng **1MB** và chi phí xử lý của **Spring** không quá nhiều).

+ Hỗ trợ lập trình hướng khía cạnh (**AOP**).

+ Xây dựng ứng dụng **web** theo mô hình **MVC**.

+ Cấu hình và vòng đời của các đối tượng của ứng dụng được quản lý bởi **Spring framework**.

+ Xử lý ngoại lệ tốt.

**Câu 7: Nguyên lý đảo ngược quyền điều khiển (Inversion of Control) là gì ?**

- **IoC (Inversion of Controller)** – “nguyên lý đảo ngược quyền điều khiển”, là nguyên lý trong phát triển phần mềm, trong đó việc điểu khiển các đối tượng hoặc thành phần của hệ thống đc thực hiện bởi framework hoặc các container (LTV không cần tạo đối tượng thay vào đó họ chỉ cần mô tả rằng nó sẽ được tạo như thế nào, framework sẽ lo phần còn lại).

**Câu 8: Bean là gì?**

- **Bean** – là những **module** **chính** của chương trình, là 1 đối tượng đc quản lý bởi **Spring IoC container** (do **framework** tự khởi tạo, tự quản lý vòng đời của đối tượng).

**Câu 9: Trong Spring có bao nhiêu Bean Scope?**

- **Singleton** (mặc định): IoC container chỉ tạo đúng duy nhất 1 object từ class bean này.

- **Prototype**: return một bean object riêng biệt cho mỗi lần sử dụng.

- **Request**: giống với prototype scope, tuy nhiên nó dùng cho ứng dụng web, 1 thể hiện của bean sẽ được tạo cho mỗi HTTP request

- **Session**: mỗi thể hiện của bean sẽ được tạo cho mỗi HTTP Session

- **Global-Session** – được sử dụng để tạo global sesion bean cho các ứng dụng Portlet.

**Câu 10: @Autowire là gì?**

- @Autowired đánh dấu cho Spring biết rằng sẽ tự động inject bean tương ứng vào vị trí được đánh dấu.

- Sau khi tìm thấy một class đánh dấu @Component. thì quá trình inject Bean xảy ra theo cách như sau:

+ Nếu Class không có hàm Constructor hay Setter. Thì sẽ sử dụng Java Reflection để đưa đối tượng vào thuộc tính có đánh dấu @Autowired.

+ Nếu có hàm Constructor thì sẽ inject Bean vào bởi tham số của hàm

+ Nếu có hàm Setter thì sẽ inject Bean vào bởi tham số của hàm

**Câu 11: @Component có ý nghĩa gì?**

- @Component dùng để đánh dấu một class Java là một Bean.

**Câu 12: Trình bày ý nghĩa của Controller?**

- Spring controller nhận ủy quyền xử lý các request từ phía front controller. Công việc cơ bản của nó là nhận dữ liệu cần thiết từ front controller, xác định view cần sử dụng, triệu gọi lớp xử lý nghiệp vụ cần thiết, sắp xếp dữ liệu mà view cần đến vào model, gửi thông tin về view cùng model lại cho front controller.

**Câu 13: Trình bày ý nghĩa của ModelAndView Interface?**

**-** ModelAndView là sự kết hợp của 2 mục đích ModelMap and View. Nó cho phép controller trả về 1 giá trị bao gồm Model và View.

**Câu 14: Trình bày ý nghĩa của ModelMap?**

- ModelMap cho phép truyền dữ liệu từ Controller sang View để hiển thị. ModelMap có các tính chất của Map.

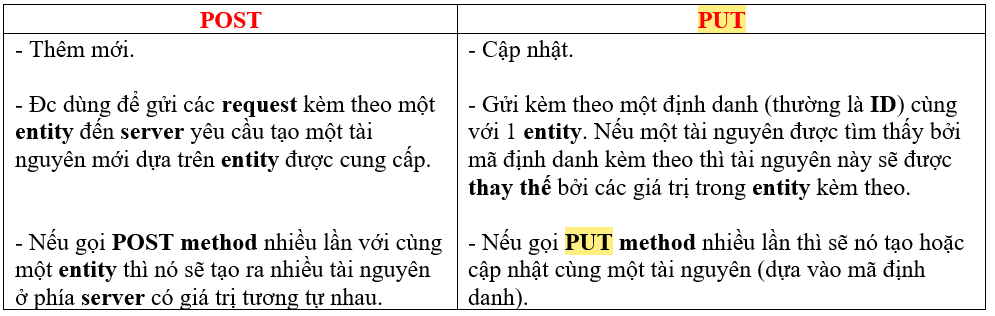
**Câu 15: Trình bày ý nghĩa của ViewResolver Interface?**

**-** ViewResolver là interface được implement bởi các đối tượng có thể phân giải view thông qua tên view.

**Câu 16: Phân biệt POST với PUT thường sử dụng để làm gì?**

**- Giống:** Đều có sự thay đổi về dữ liệu

**- Khác:**



**Câu 17: @RequestMapping làm gì?**

- @RequestMapping là một trong những annoation sử dụng nhiều nhất trong Spring MVC. Annotation @RequestMapping được sử dụng để map request với class hoặc method xử lý request đó. @RequestMapping có thể được áp dụng với controller class hoặc method trong controller class.

**Câu 18: Thuộc tính consumes trong các Request Mapping là gì ?**

**- Consumes**: Định dạng kiểu dữ liệu khi **client request** tới **server** (tương ứng với cài đặt “**content - type**” của **request header**).

**Câu 19: Thuộc tính produces trong các Request Mapping là gì ?**

**- Produces**: định nghĩa kiểu dữ liệu trả về cho client (tương ứng với cài đặt “accept” của request header).

**Câu 20: Trình bày cơ chế Data Binding trong Spring?**

- Data Binding là cơ chế liên kết dữ liệu đầu vào và đầu ra với một đối tượng Model. .

+ Các form đều được liên kết với một đối tượng biểu diễn dữ liệu phía sau.

=> Dữ liệu trên form sẽ được tự động chuyển đổi thành các thuộc tính của đối tượng liên kết với form.

+ Data binding hỗ trợ chuyển đổi dữ liệu và validate.

+ Data binding hoạt động dựa trên binder.

**Câu 21: So sánh RequestParam và PathVariable?**

**- Giống nhau:** Đều thực hiện chung một nhiệm vụ là lấy các tham số từ người dùng truyền lên.

**- Khác nhau:**

**+ @RequestParam** – lấy giá trị theo định dạng **key-value** (**param** động – có hay không có **key** cũng đc), có các thuộc tính sau:

. **value**: tên của **parameter** (**key**) gửi lên **server**.

. **required**: quy định **param** có bắt buộc hay không (mặc định là **true**).

. **default**: giá trị mặc định của **param**.

+ **@PathVariable** – lấy giá trị tham số tĩnh trên **URL** (chỉ lấy **value**), sử dụng đc **regex**.

**Câu 22: Thymeleaf là gì?**

- **Thymeleaf** – là 1 bộ xử lý **view**, đc sử dụng cho các ứng dụng **web** và các ứng dụng độc lập.

+ Được sử dụng cho phù hợp với tiêu chuẩn **web**, hỗ trợ **HTML 5**.

+ **Thymeleaf** cho phép hỗ trợ các **template**: HTML, XML, JS, CSS, TEXT,...

**- Ưu điểm:**

+ Thân thiện, gần gũi vì sử dụng cú pháp **HTML**.

+ Cú pháp của **thymeleaf** là 1 **attribute** (thuộc tính) của 1 thẻ **HTML** và bắt đầu bằng **th:** (th:each, th:text, th:switch, th:if, th:unless, th:field,...) 🡪 dễ sử dụng.

+ Vì là **HTML** nên ta có thể xem các file **view** mà không cần khởi chạy **server**.

+ Chỉ cần sử dụng file **HTML** là có thể hiển thị tất cả mọi thứ (không cần **JSP**,...).

+ **Thymealeaf** sẽ tham gia vào **render** các file **HTML** dưới dạng các thuộc tính trong các thẻ **HTML** 🡪 do đó ta không cần phải thêm bất kỳ thẻ **non-HTML** nào.

**Câu 23: Sử dụng lặp trong Thymeleaf như thế nào? Sử dụng điều kiện trong Thymeleaf như thế nào?**

**- Vòng lặp:** th:each="loop : ${loopList}";

+ index: chỉ số của lần lặp hiện tại, bắt đầu từ giá trị 0.

+ count: chỉ số của lần lặp hiện tại, bắt đầu từ giá trị 1.

+ size: tổng số các phần tử.

+ current: phần tử của vòng lặp hiện tại.

+ even / odd: vòng lặp chẵn / lẻ.

+ first / last: vòng lặp đầu tiên / cuối cùng.

- **Điều kiện**: th:if="${condition}";

\* th:if trả về true trong các trường hợp sau:

+ Một giá trị boolean là true.

+ Một số có giá trị là non-zero (khác 0).

+ Một ký tự có giá trị là non-zero (khác 0).

+ Một chuỗi có giá trị khác với “false”, “off” hoặc “no”.

+ Nếu một giá trị không phải là boolean , số, ký tự hoặc chuỗi.

**Câu 24: Formatter là gì ? Converter là gì ?**

- **Spring Converter** là một đối tượng được dùng để chuyển đổi kiểu dữ liệu này sang kiểu dữ liệu khác. Để tạo một converter, chúng ta viết một lớp Java triển khai interface **org.springframework.core.convert.converter.Converter.**

- **Formatter** cũng hoạt động giống như converter, tức là chuyển đổi một kiểu dữ liệu sang kiểu dữ liệu khác. Tuy nhiên, kiểu dữ liệu nguồn của Formatter là String, trong khi đó converter có thể làm việc với bất cứ kiểu dữ liệu nguồn nào. Formatter phù hợp hơn với tầng web (web-tier), còn converter thì có thể sử dụng ở bất cứ tầng nào. Trong trường hợp muốn chuyển đổi dữ liệu nhập vào từ một trường input trong form trong một ứng dụng Spring MVC, chúng ta nên lựa chọn formatter hơn là converter. Để tạo một formatter, chúng ta viết một lớp Java triển khai interface **org.springframework.format.Formatter**

**Câu 25: ORM là gì ?**

- ORM (Object-Relational Mapping) – là kỹ thuật liên kết giữa đối tượng trong lập trình với đối tượng trong CSDL (là quá trình biến đổi giữa 1 record trong DB và 1 object trong ngôn ngữ lập trình OOP).

+ ORM ra đời để khắc phục nhược điểm của JDBC, như:

. Cấu hình thủ công (cần phải chuẩn bị DB trước).

. Có kiến thức nền tảng về SQL.

. Phải biết kiểu dữ liệu đầu ra / đầu vào của từng column.

+ ORM có rất nhiều triển khai (Hibernate, Eclipselink, Ibatis,…), trong đó Hibernate đc xem là ORM Framework nổi tiếng nhất và sử dụng rộng rãi trong Java.

**Câu 26: JPA là gì? Spring Data JPA là gì ?**

- **JPA (Java Persistence API)** – là 1 bộ đặc tả tiêu chuẩn của Java, trong đó mô tả các thao tác quản lý dữ liệu trong Java cung cấp các mô hình cho phép ánh xạ các bảng trong CSDL với các class. (JPA chỉ chứa interface hướng dẫn triển khai ORM, việc implement sẽ do ORM framework thực hiện).

**- Spring Data Jpa** – là 1 phần của dự án Spring Data, có thể sử dụng Spring Data JPA với các ORM framework như Hibernate, Open JPA,…

Là tầng được xây dựng phía trên tầng JPA. Có nghĩa là nó sẽ sử dụng tất cả các chức năng của JPA như entity mapping, JPA query, xây dựng một số tính năng như khởi tạo query thông qua tên hàm,…

Spring Data Jpa giúp cải tiến **JPA** tiêu chuẩn, đơn giản hóa tầng truy xuất dữ liệu, tự tạo **repository**, tạo các truy vấn **JPA** thông quá tên phương thức.

**Câu 27: Hibernate là gì ? Hibernate làm gì với database?**

- **Hibernate** - là một ORM Framework mã nguồn mở giúp LTV viết ứng dụng Java có thể mapping các Object với hệ CSDL, và hỗ trợ thực hiện các khái niệm OOP với CSDL.

- Hibernate có thể hỗ trợ tạo bảng và các dữ liệu thông qua ngôn ngữ Java.

**Câu 28: Liệt kê một số annotation của hibernate?**

**- @Entity** được sử dụng để chú thích một class là một Entity.

- **@Id** được sử dụng để mô tả đây là Id (Identity) của Entity, nó tương đương với cột đó là khóa chính (Primary Key) của table trong database.

- **@GeneratedValue** được sử dụng để Hibernate tự động tạo ra giá trị và gán vào cho một cột trong trường hợp insert mới một Entity vào database. Nó có thể gắn trên cột Id hoặc một cột nào đó.

- **@ManyToOne** mô tả một quan hệ  **N-1** (Nhiều – Một), nó thường được sử dụng cùng với **@JoinColumn**.

- **@OneToMany** mô tả quan hệ **1-N** (Một – Nhiều). Nó là đảo ngược của @ManyToOne, và vì vậy nó dựa vào @ManyToOne để định nghĩa ra @OneToMany.

- **@OneToOne** mô tả quan hệ **1-1** (Một – Một).

- **@ManyToMany** mô tả quan hệ **N-N** (Nhiều – Nhiều).

- **@Table** cho phép chú thích tên bảng thông qua thuộc tính name (thuộc tính này không bắt buộc). Nếu không chỉ rõ tên bảng trong phần tử name, Hibernate sẽ dựa vào phần tử name của **@Entity** sau đó mới tới tên của class.

- @Column được sử dụng để chỉ định thông tin chi tiết của cột mà một field của entity sẽ được ánh xạ với một column trong database.

**Câu 29: Entity là gì?**

**- Entity** – là đối tượng đại diện cho dữ liệu trong ứng dụng.

+ **Entity** thường là **POJO** (Plain Old Java Object), nghĩa là lớp đơn giản, chỉ mô tả dữ liệu, không có xử lý logic nghiệp vụ.

+ Mỗi **Entity** đc **mapping** với 1 **table** trong **CSDL** (**mapping** 1-1).

+ Mỗi **Entity** cần phải tuân thủ:

. Được gắn **annotation** **@Entity**, **@Id**.

. Có **Constructor** là **public** không tham số.

. Không được khai báo **final**.

- Thuộc tính có **access modifier** là **private**, **protected**, hoặc ở mức **package private** (tức là không phải **public**).

**Câu 30: Persistence Context & Entity Manager là gì?**

**- Persistence Context** là tập các thể hiện của entity được quản lý, tồn tại trong một kho dữ liệu.

**- Entity Manager** là đối tượng quản lý các entity. Interface EntityManager thực hiện các công việc sau:

+ **persit**( ) – thêm 1 entity.

+ **merge**( ) – chỉnh sửa, cập nhật.

+ **remove**( ) – xóa.

+ **find**( ) – tìm kiếm.

**Câu 31: Câu lệnh truy vấn động là gì? Câu lệnh truy vấn tĩnh là gì?**

- **Câu truy vấn động** có thể truyền tham số và thay đổi linh động tùy thuộc vào giá trị tham số truyền vào.

- **Câu truy vấn tĩnh** không thể truyền tham số trong lúc **complie-time .**

- Dùng phương thức **createQuery()** của **EntityManager** để tạo truy vấn động.

- Dùng phương thức **createNameQuery()** để tạo truy vấn tĩnh.

- Dùng phương thức **createNativeQuery()** để tạo truy vấn thuần.

**Câu 32: Trạng thái của entity bao gồm những gì?**

- **Transient** (Tạm thời): Đối tượng không có quan hệ với **Session** hiện tại của **Hibernate**. Đối tượng ở trạng thái này chưa từng gắn vào **context**, nó không có bản ghi tương ứng trong **CSDL**.

- **Persistent** (Bền vững): Đối tượng đang liên hệ với một **context**, tức là với một đối tượng **Session** và trạng thái của nó được đồng bộ với cơ sở dữ liệu khi mà ta **commit** cái **Session**.

- **Detached** (Đã bị tách riêng ra): Đối tượng đã từng có trạng thái **persistent** nhưng hiện tại đã không còn giữ quan hệ với **Session**. Nếu nó không được **attached** trở lại, nó sẽ bị bộ gom rác của **Java** quét đi theo cơ chế thông thường.

- **Removed** (Đã bị xóa): tương tự như **detached** nhưng bản ghi tương ứng với đối tượng này trước đó đã bị xóa khỏi **database**.

**Câu 33: Có bao nhiêu loại mapping trong Hibernate?**

- **Association Mappings (hay Entity Mappings)** – là ánh xạ các liên kết giữa các lớp entity và các mối quan hệ giữa các bảng chính, gồm 4 kiểu liên kết:

+ Many-to-One.

+ One-to-Many.

+ One-to-One.

+ Many-to-Many.

- **Collections Mappings** – nếu một entity hoặc lớp có một tập các giá trị trong một trường Collection cụ thể, thì chúng ta có thể ánh xạ các giá trị đó bằng bất kỳ một trong các interface collections có sẵn trong java. 1 số kiểu collection: Set, SortedSet, List, Collection, Map, SortedMap.

- **Component Mappings** – mapping cho một lớp có 1 tham chiếu đến 1 lớp khác như là một biến thành viên. Sử dụng phần tử <component> để thiết lập sự tồn tại của các thuộc tính khác của lớp của biến tham chiếu bên trong lớp sở hữu nó.

**Câu 34: Để sử dụng interface JpaRepository cần cung cấp những thông tin nào?**

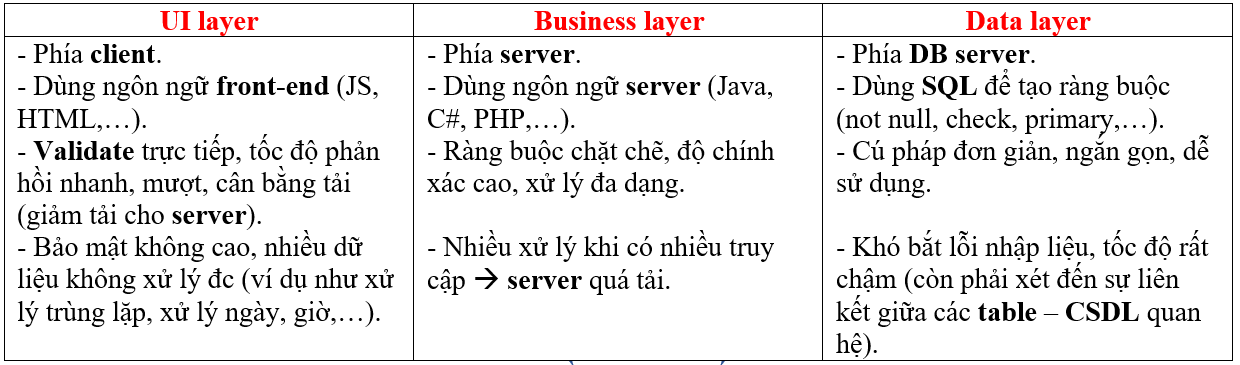
- Cung cấp 1 Entity Class tương ứng với class java đại diện cho 1 bảng trong database

- Cung cấp kiểu dữ liệu của trường là khóa chính trong bảng đó.

**Câu 35: Validation dữ liệu là gì?**

- **Validation** – là việc thực hiện đánh giá / xác minh tính hợp lệ của dữ liệu đầu vào (đối với mọi ứng dụng web, dữ liệu luôn quan trọng hơn công nghệ).

- Có thể thực hiện validation ở 3 tầng:



=> Một ứng dụng web thông thường, chỉ cần validate kết hợp giữa **UI layer** và **Business layer** (Spring hỗ trợ validation ở tầng business).

**Câu 36: Trình bày cách triển khai validate dữ liệu trong Spring?**

**Cách 1: Dùng annotation có sẵn của Spring**

- **Đặt annotation tại tầng cần validate.**

**@NotNull** – không đc phép null, request gửi lên không đc mất trường.

**@NotEmpty** – không đc trống / empty (có thể áp dụng với trường collection).

**@NotBlank** – không đc phép null / khoảng trắng (áp dụng cho chuỗi).

**@Size** – kiểm tra độ dài min / max.

**@Min** / **@Max** – giá trị nhỏ nhất / lớn nhất.

**@Email** – định dạng email.

**@Pattern** – truy định dạng thức của 1 chuỗi.

**@DateTimeFormat** – định dạng ngày, giờ.

- Ở **controller**, khai báo **@Validated** / **@Valid**, **BindingResult** (dùng **hasFieldErrors()** để kiểm tra có tồn tại lỗi hay không).

- Tại **thymeleaf** dùng **th:errors** để hiển thị lỗi

**Cách 2: Dùng custom validation**

- Tại dto, cần implement interface **Validator**

- Tại controller, gọi phương thức validate bởi dto.

**Cách 3: Custom annotation (tự tạo annotation)**

**Câu 37: Binding Result là gì?**

**- BindingResult** là 1 đối tượng, là nơi **Spring** chứa các kết quả (lỗi) trong quá trình **Validate**.

**Câu 38: AOP là gì? AOP gồm những yếu tố nào?**

- **AOP (Aspect Oriented Programming)** – là kỹ thuật lập trình hướng khía cạnh, nhằm tách chương trình chính thành các module riêng lẻ, phân biệt và không phụ thuộc lẫn nhau (nó chỉ hỗ trợ cho OOP, và có phụ thuộc vào OOP).

- AOP bao gồm những yếu tố:

+ **Core concern / Primary concern**: Là những xử lí logic chính của chương trình.

+ **Cross-cutting concern** (mối quan tâm xuyên suốt): Là những **logic xử lí phụ** cần đc thực hiện của chương trình khi **core concern** đc gọi.

+ **Join point** (điểm gia nhập): Là 1 vị trí nào đó trong luồng thực thi của chương trình mà advice sẽ đc đan (**weave**) vào chương trình chính.

+ **Advice** (lời khuyên): Là 1 hành động cụ thể của Aspect.

+ **Weave** (đan / dệt): Là quá trình liên kết giữa các thành phần aspect và core concern của 1 chương trình để tạo đầu ra mong muốn.

+ **Point cut**: Chỉ ra tập hợp các join point để công cụ AOP đan các advice vào nghiệp vụ chính.

+ **Execution**: áp dụng cho method.

+ **Within**: áp dụng cho package, class,…

**Câu 39: Các loại Advice?**

- **Before advice** – thực thi trước khi Join point.

- **After returning advice** – thực hiện khi Join point hoàn thành bình thường.

- **After throwing advice** – thực hiện khi Join point kết thúc có ngoại lệ.

- **After (finally)** – thực thi sau khi Join point kết thúc, dù thành công hay thất bại.

- **Around advice** – thực thi trước và sau khi join point.

**Câu 40: JoinPoint là gì?**

**- Join point** (điểm gia nhập): Là 1 vị trí nào đó trong luồng thực thi của chương trình mà advice sẽ đc đan (**weave**) vào chương trình chính.

**Câu 41: Nêu cách triển khai AOP mà bạn dùng trong dự án?**

- Khai báo **class AOP** và đánh dấu bằng **annotation** **@Aspect**, **@Component**.

- Xác định điểm gia nhập (**join point**) hoặc tập hợp các điểm gia nhập (**point cut**) thông qua **annotation** **@PointCut** (phạm vi áp dụng là **execution** / **within**).

- Khai báo các **Advice** với các **annotation**: **@Before** / **@AfterReturning** /

**@AfterThrowing** / **@After** / **@Around**.

- Viết phương thức **Advice** (đưa ra các lời khuyên) khi **Joinpoint** xảy ra.

**Câu 42: Mục đích xử lý ngoại lệ trong Spring Web MVC ?**

**-** Bắt đc ngoại lệ và có hướng xử lí phù hợp, đảm bảo chương trình không bị dừng đột ngột.

- Bảo mật, tránh bị kẻ xấu xâm nhập.

- Không gửi thông tin lỗi về client, thân thiện với người dùng.

**Câu 43: Các cách xử lý ngoại lệ trong Spring Web MVC ?**

- Sử dụng **try-catch-finally** (**throw**, **throws**).

- Sử dụng **@ExceptionHandle** để bắt 1 **Exception** nào đó.

Nếu có **exception** ở bất cứ đâu, sẽ được chuyển tới **method** có **@ExceptionHandler** tương ứng. Thứ tự đặt chúng không quan trọng, **Spring** sẽ tự động tìm cái phù hợp nhất, nếu không có thì chuyển dần lên các **exception** class cha (do đó, nên có một **@ExceptionHandler** để bắt **Exception class**, dành cho các **exception** còn lại).

Các **method** này viết tương ứng với **method** của **Controller**, nhưng thay vì trả **data** về thì chúng ta trả về **message** lỗi, hoặc 1 trang thông báo đến người dùng.

**Câu 44: Cookie là gì?**

-Là những trang **web** do người dùng truy cập tạo ra, giúp trãi nghiệm trực tuyến dễ dàng hơn

(ví dụ như ghi nhớ nội dung tùy chọn các dữ liệu đã nhập của trang **web**, thông tin lịch sử truy cập **web.** => giúp cho **website** đề xuất những nội dung phù hợp với người dùng).

**Câu 45: Phân biệt Session và Cookie?**

**- Giống:** Đều dùng để lưu thông tin tạm thời.

- **Khác:**



- 1 **session** bị phụ thuộc vào **cookie** thông qua **session id**, ngược lại thì không.

(Nghĩa là, mỗi **session** sẽ có 1 **SessionID** riêng biệt. Khi kết thúc một **session** và bắt đầu một **session** mới thì ta sẽ có 1 **SessionID** khác. Giá trị của **SessionID** sẽ được lưu vào trong **cookie**, giúp cho **server** biết **session** này của **client** nào).

- **JsessionId** khởi tạo khi vào **session** => phân biệt các **session** ở các **client** khác nhau.

**Câu 46: Session là gì ? Cho ví dụ về session ?**

**- Session** là phiên làm việc giữa **client** và **server**.

+ Thông thường, sử dụng **session** để lưu trữ thông tin đăng nhập, giỏ hàng, hoặc những dữ liệu mang tính chất tạm thời.

+ 1 **session** đc bắt đầu khi **client** thực hiện **request** lần đầu tiên đến **server** và kết thúc khi **client** dừng làm việc với **server** (đóng trình duyệt, hết time **session**, **server** bị tắt,…).

- VD: Khi bạn đăng nhập vào một trang web bằng tài khoản email và mật khẩu của bạn được xác minh là đúng, một Session sẽ được tạo trên máy chủ để lưu các thao tác của bạn cho đến khi kết thúc phiên. Các thông tin như dữ liệu đăng nhập, thông tin về sản phẩm trong giỏ hàng,… đều có thể được lưu lại.

**Câu 47: Nêu cách triển khai Session mà bạn dùng trong dự án?**

- Ở **controller**, khai báo **annotation** **@SessionAttributes**.

- Khởi tạo đối tượng **Session** thông qua **annotation** **@ModelAttribute**.

- Sử dụng **@SessionAttribute** để lưu trữ các giá trị trong một phiên làm việc.

**Câu 48: Web Service là gì? Lấy ví dụ về web service?**

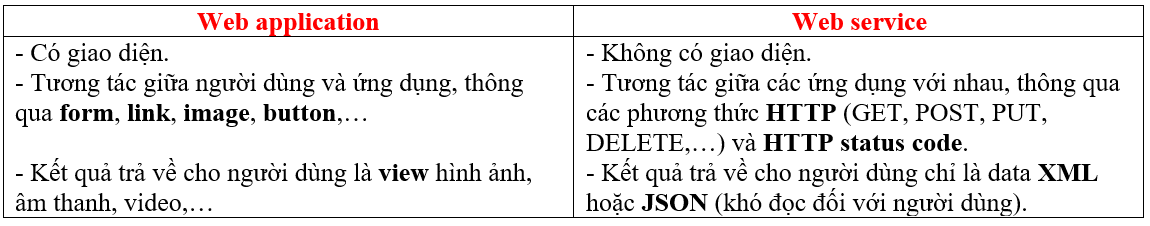
**- Webservice** – là ứng dụng **web** cung cấp nền tảng cho nhiều ứng dụng khác (giá vàng, chứng khoáng, dự báo thời tiết,…), những ứng dụng này có thể đc viết từ nhiều nền tảng công nghệ khác nhau, nhưng có thể giao tiếp với nhau (thông qua **data**).

+ Kiểu dữ liệu dùng để giao tiếp thường là **XML** hoặc **JSON**.

+ Phân loại: SOAP, REST

- Ví dụ: những trang web đc tích hợp bản đồ, giá vàng, chứng khoáng, dự báo thời tiết, biểu đồ thống kê số ca mắc covid,…

**Câu 49: Phân biệt Web Service và Web truyền thống?**



**Câu 50: SOAP là gì? RESTful là gì?**

- **SOAP** (Simple Object Access Protocol) – là giao thức **Internet** cho phép các chương trình trong các hệ điều hành khác nhau hoặc trên các máy chủ riêng biệt liên kết với nhau qua **Internet**. Giao thức sử dụng **XML** (cú pháp nghiêm ngặt, môi trường đóng do dùng trong doanh nghiệp).

- **RESTful (Representational State Transfer)** là một tiêu chuẩn trong thiết kế API cho các ứng dụng web để quản lý resource. Sử dụng phương thức HTTP đơn giản để giao tiếp giữa các máy. RESTful sử dụng giao thức JSON hoặc XML.

**Câu 51: jQuery là gì? Ajax là gì?**

- **Jquery** – Là thư viện đc xây dựng từ **JS**, với câu slogan **“Write less, do more”**.

Với các tính năng nổi bật:

+ Thao tác với **DOM** dễ dàng, nhanh chóng (cách truy cập **DOM** gần giống với cú pháp **CSS**).

+ Xử lý **event** tốt.

+ Xử lý **animation** dùng để tạo hiệu ứng chuyển động.

+ Hỗ trợ **Ajax**.

+ Gọn nhẹ, tích hợp **file JS**.

+ Hỗ trợ hầu hết các loại trình duyệt.

- **Ajax** – Là 1 kỹ thuật giúp phát triển trang **web** động mà không cần tải lại trang.

+ Tạo các trang **web** nhanh hơn, mượt hơn.

+ Cập nhật nội dung của 1 phần trang **web** mà không cần tải lại trang.

=> Giúp trải nghiệm người dùng tốt hơn.

+ Cú pháp:

**$.ajax({**

**url:** "địa chỉ trang web cần tương tác”,

**type:** "phương thức HTTP method",

**dataType:** "kiểu dữ liệu",

**success: function (**response**) {** //xử lý kết quả trả về khi thành công; **}**

**})**

**Câu 52: Nêu ví dụ về cách bạn triển khai Ajax trong dự án của bạn?**

- Ở controller, khai báo annotation @CrossOrigin (cho phép truy cập các đường dẫn khác).

- Sử dụng jQuery $(document).ready để thông báo sử dụng jQuery khi HTML load xong

- Ở view, có thể dùng jQuery để bắt sự kiện.

- Dùng $.ajax để triển khai Ajax thông qua các thuộc tính: url, dataType, type, success.

**Câu 53: @RequestHeader và @ResponseHeader có ý nghĩa gì?**