**1. INTRODUCTION SPRING MVC**  \_**The Last Shinobi ϟ\_**

- **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.

(**framework** ràng buộc, ép người dùng phải theo đúng luật chơi của nó, nó nắm quyền điều khiển nên phải code sao cho nó hiểu, ví dụ phải đứng ở trạm xe buýt thì mới xe mới đón khách,…).

- **Phân biệt Framework/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**.

|  |  |
| --- | --- |
| **Library** | **Farmework** |
| - Ứng dụng nắm quyền điều khiển (**code** của mình sẽ gọi **code** của thư viện).  - Là tập hợp các tính năng, các lớp.  - Hoạt động bị động.  - Sử dụng trực tiếp mà không cần thay đổi cấu trúc dự án.  - Không có cơ chế **IoC**. | - **Framework** nắm quyền điều khiển ứng dụng (code của **framework** sẽ gọi code của mình).  - Tập hợp các thư viện.  - Hoạt động chủ động.  - Sử dụng **framework** phải thay đổi cấu trúc dự án.  - Có cơ chế **IoC**. |

- **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).

- **DI** (**Dependency Injection**) – là **design pattern** cho phép xóa bỏ sự phụ thuộc, là một cơ chế để triển khai **IoC**, trong đó thao tác tiêm (**inject**) vào các đối tượng khác được thực hiện bởi **container** hoặc **framework**.

(Giống với Nguyên tắc **D** trong **S.O.L.I.D** là **Dependency inversion principle** - **Nguyên lý đảo ngược phụ thuộc**: các **module** cấp cao không nên phụ thuộc vào các **module** cấp thấp, cả 2 nên phụ thuộc vào **abstraction**).

=> Giảm sự phụ thuộc giữa các **module**, **object** giúp cho việc mở rộng **code** sau này được dễ dàng.

=> Giúp cho việc quản lý và kiểm thử ứng dụng dễ dàng hơn.

=> Giúp chúng ta tập trung vào việc viết **logic** nghiệp vụ của ứng dụng, còn việc tạo và quản lý các đối tượng đc giao cho **framework** lo.

🡺 **Có 3 cách triển khai:**

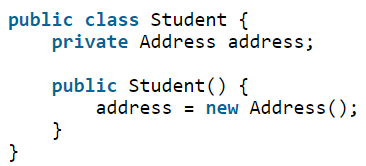
+ **Constructor** - Dùng **inject** (tiêm) các **module** bắt buộc. Các **module** được tiêm nằm trong **constructor**, và được gán lần lượt vào các **field**.

+ **Setter** method - 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.

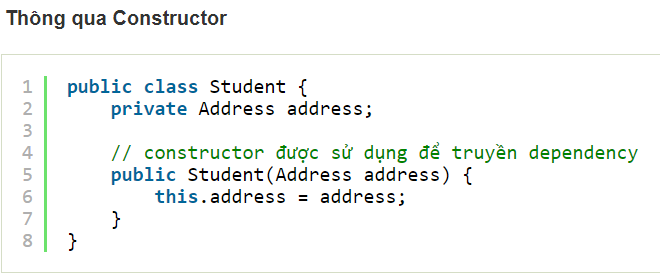
+ **Interface** (field) - Các **client** phải triển khai một **Interface** mà có một **setter** method dành cho việc nhận **dependency**.

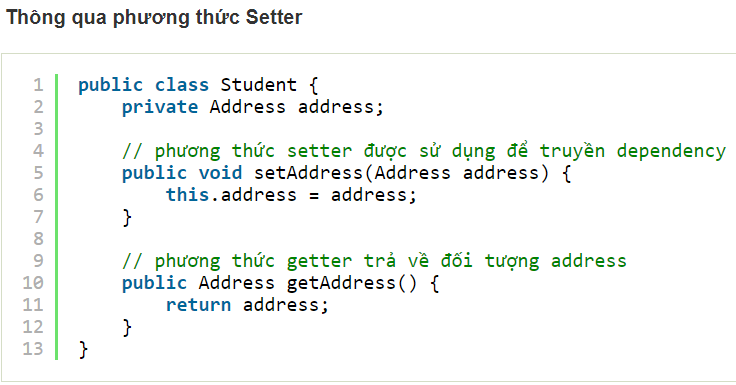
🡺 **Ví dụ:**

+ Khi không sử dụng **DI**: đối tượng **address** sẽ bị phụ thuộc vào **Student**. Mỗi khi muốn thay đổi đối tượng này thì phải thay đổi đối tượng **student**.



+ Khi sử dụng **DI**: để **student** và **address** giảm phụ thuộc vào nhau, chúng ta có thể khởi tạo đối tượng **address** ở bên ngoài lớp **Student** và truyền vào thông qua **Constructor** hoặc phương thức **Setter**.





- **Spring MVC framework** – là 1 **Framework** đc thực hiện trong mô hình **MVC** của các ứng dụng **web**.

+ **Front controller** – xử lý, lắng nghe toàn bộ **request** từ bên ngoài ứng dụng, thực hiện điều hướng cho **controller** xử lý, trả về **data**, hiển thị lên cho người dùng.

+ **DispatcherServlet** (trình điều phối) – đây chính là **front** **controller**.

**Dispatcher** là thành phần quan trọng nhất trong **SpringMVC**. Nó sẽ là nơi đầu tiên nhận **request** từ **client** sau đó sẽ chuyển **request** đó tới các **controller** tương ứng, đồng thời sẽ là chốt chặn cuối cùng trả về kết quả cho **client**.

+ **Controller** – là **file class**, xử lý **logic** nghiệp vụ.

+ **ViewResolver** (trình phân giải **view**) – là cơ chế ánh xạ “**tên**” của **view** sang đối tượng **view** tương ứng.

+ Luồng thực thi:

🡪 Người dùng điền tên **website** cần truy cập trên **brower** 🡪 nhấn **Enter**

🡪 gửi **request** lên **server** (nơi triển khai ứng dụng **Spring**)

🡪 **DispatcherServlet** tiếp nhận **request**

🡪 chuyển tới **controller** tương ứng(thông qua các cơ chế **mapping –** ánh xạ đường dẫn)

🡪 **controller** nhận yêu cầu và gọi phương thức thích hợp (**GET** / **POST**)

🡪 **controller** tiến hành **render** dữ liệu và trả về tên **view** cho **DispatcherServlet**.

🡪 **DispatcherServlet** sẽ nhận trợ giúp từ **ViewResolver** để xác định **view** cho các yêu cầu.

🡪 **DispatcherServlet** chuyển dữ liệu tới **view** đó 🡪 chuyển tới trình duyệt.

- **Cách gửi dữ liệu từ controller 🡪 view:**

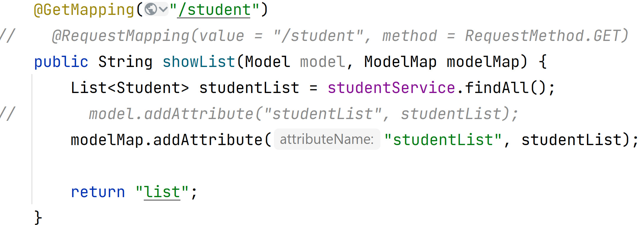
**+ Model** – là 1 **interface**.

**+ ModelMap** – là đối tượng (1 **class**) implement **Map** **interface** trong **Java**, đc **collection** tích hợp thêm các phương thức của **map**.

=> **Model** và **ModelMap** chỉ gửi **dữ liệu**.

**+ ModelAndView** – Là 1 **interface**, là sự kết hợp của 2 khía cạnh truyền **dữ liệu** và **view**.

Ví dụ:





**@GetMapping**(“/hello”) – ánh xạ đường dẫn /hello đến phương thức đằng sau nó (phương thức này sẽ đc gọi khi có 1 truy cập đến đường dẫn /hello).

**@RequestMapping**(…) – ánh xạ các request tới các **action** tương ứng của **controller**.

- **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.

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.

- **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.

**2. SPRING CONTROLLER**

- **Ý nghĩa của Controller:**

Trong mô hình **MVC**, **controller** là nơi nhận **request** từ người dùng, xử lý **request**, xây dựng dữ liệu cho **view** (**model**) và chọn **view** để trả lại kết quả của cho người dùng.

- **@Controller** – dùng để đánh dấu 1 **class** là 1 **Spring MVC controller** (dùng cho **web application**).

- **@RestController** – dùng đánh dấu 1 **class** là **resfull controller** (dùng cho ứng dụng **web service**).

- **@Autowired** – tự động nhúng các **bean** được **Spring Container** sinh ra vào **Class** có khai báo **@Autowire**.

- **@RequestMapping** – đc dùng để ánh xạ các **request** tới các xử lý ở **controller** tương ứng.

Thuộc tính:

+ **value** (URL pattern).

+ **method**: GET, POST, PUT, PATCH, DELETE,…

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

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

- **@RequestParam** – lấy giá trị theo định dạng **key-value** (**param** động – có hay không có **key** cũng đc).

Thuộc tính:

+ **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**.

- **Handle method** – là method dùng để nhận request của người dùng, xử lý và trả về kết quả.

- **Các biến thể của @RequestMapping**:

+ **@GetMapping**.

+ **@PostMapping**.

+ **@DeleteMapping**.

+ **@PutMapping**.

+ **@PathMapping**.

- **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).

Các cách khai báo:

+ Dùng annotation: **@Controller**, **@Service**, **@Repository**, **@Component**.

+ Dùng kết hợp **@Configuration** + **@Bean**.

+ Khai báo **Bean** bằng **XML**.

- **Có 5 Bean Scope:**

+ **Singleton** – chỉ duy nhất một thể hiện của **bean** sẽ được tạo cho mỗi **container**. Đây là **scope** mặc định cho **spring bean**.

+ **Prototype** – khác với **Singleton**, 1 thể hiện của **bean** sẽ được tạo cho mỗi lần được yêu cầu (**request**).

+ **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**.

- **@Component** – là một **annotation** của **class**. Nó dùng để đánh dấu **class Java** là một **bean**. Khi **Spring start** thì nó quét qua các **annotation** có đánh dấu là **@Component** thì nó sẽ tạo **bean** cho **class** đó.

- **Cách cấu hình cho Spring Boot tìm kiếm các Bean ở nhiều package khác nhau**:

+ Cách 1: Sử dụng **@ComponentScan** – là **annotation** khai báo ở cấp độ **class**, nó thông báo **Spring** **Container** biết phải vào **package** nào trong dự án để quét các **Annotation** và tạo **Bean**., ngoài ra chúng ta cũng có thể tùy chỉnh **package** cần **scan**.

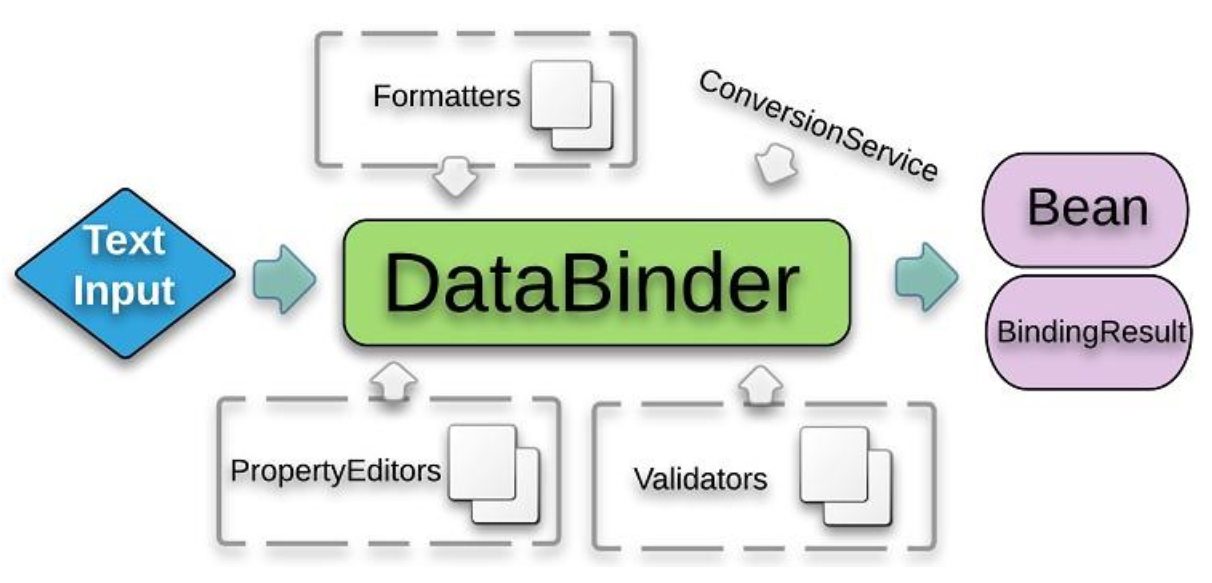
+ Cách 2: Sử dụng **scanBasePackages** tromg **@SpringBootApplication**.

**3. DATA BINDING & FORM**

- **Data binding** – là cơ chế liên kết dữ liệu đầu vào / đầu ra với đối tượng **model**.

+ Các **form** đều đc liên kết với 1 đối tượng biểu diễn dữ liệu phía sau 🡪 dữ liệu trên **form** sẽ đc tự động chuyển đổi thành các thuộc tính của đối tượng liên kết **form**.

+ **Data binding** hỗ trợ chuyển đổi dữ liệu và **validate**, hoạt động dựa trên **data binder**.



- **Data binder:**

+ Mỗi đối tượng như là 1 **form**, mỗi thuộc tính tương ứng với 1 ô **input**.

+ **Text input** sẽ đc **mapping** với **data binder** thông qua **propertyEditors** (thông qua **getter/setter**).

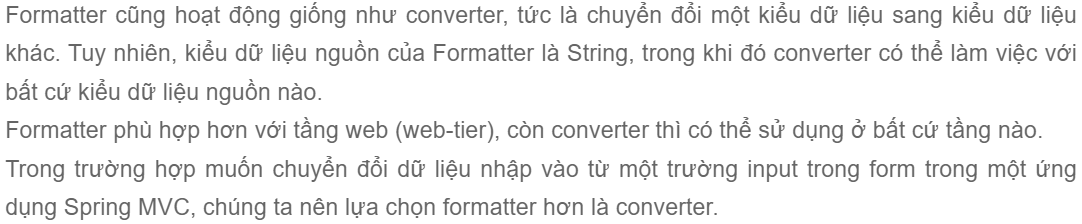
+ **ConversionService** giúp **convert** (chuyển đổi) dữ liệu 1 cách tự động.

+ **Formatters** giúp can thiệp sâu về cấu hình (ví dụ như **validate** dữ liệu ngày, giờ).

+ **Validators** giúp **validate** trực tiếp thuộc tính của đối tượng.

+ Đầu ra sẽ là 1 **Bean** 🡪 gán giá trị vào đối tượng (bắt đối tượng thông qua **@ModelAttribute**).

- **Formatter & Converter:**



- **Giới thiệu về form và các thẻ của thư viện Spring-form-tld:**

+ Thư viện **Spring-form-tld** hỗ trợ tạo **form** trong **spring**, bổ sung thêm các thuộc tính **ModelAttribute** của thẻ **form**.

+ **Spring-form** hỗ trợ các thẻ **input**, **checkbox**, **radio**,...

+ Một số thuộc tính quan trọng:

- **path** – tên thuộc tính của đối tượng liên kết với **form**.

- **item** – danh sách các hạng mục của trường (thường dùng với **checkboxs**, **radio buttons**,...).

**4. VIEW THYMELEAF**

- **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 đểm:

+ 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, th:href, th:fragment,...) 🡪 dễ sử dụng.

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

+ **Thymealeaf** sẽ tham gia vào **renderd** 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.

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

**+ th:if** trả về **true** trong các trường hợp:

• 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.

+ **th:each** cung cấp các trạng thá isau:

• **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 trị1

• **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.

- **Các biểu thức:**

+ Biểu thức với biến **${...}**.

+ Biểu thức với thuộc tính **\*{...}**.

+ Biểu thức với message **#{...}**.

+ Biểu thức với URL **@{...}**.

+ Biểu thức với phân đoạn **~{...}**.

- **@ModelAttribute** – như là một cầu nối giữa **Controller** và **View**. Từ **Controller** chúng ta truyền các dữ liệu qua cho **View** thông qua **ModelAttribute**. Từ **View** chúng ta sẽ sử dụng **Thymeleaf** để đọc các dữ liệu từ **model** và hiển thị ra cho người dùng.

**5. ORM & SPRING DATA JPA**

- **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 kích thước 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**.

- **Hibernate** - là một thư viện **ORM** 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**.

- **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 được tầng được xây dự 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,…

🡺 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.

🡺 Có thể hiểu nôm na, **Hibernate** làm việc tay chân như những anh công nhân chăm chỉ, chịu khó, kham khổ, và để tăng năng suất thì những anh công nhân này cần đến sự chỉ đạo từ những anh tiến sĩ Việt Nam yêu nước (**JPA**). Những anh tiến sĩ này sẽ cũng cấp các bản vẽ, các phương pháp để tăng hiệu năng sản xuất. Đắng lòng thay, không phải cái gì các anh tiến sĩ Việt Nam cũng làm đc, vậy nên mới cần đến sự hỗ trợ của những con người có trình độ cao hơn, đó là các tiến sĩ nước ngoài, với những bản “luận án chiến sĩ” tối ưu hơn, (**Spring Data Jpa**).

- **Interface trong Spring Data JPA:**

+ **Repository** – quản lý **entity**, kiểu dữ liệu khóa chính **entity**.

+ **CrudRepository** – định nghĩa các phương thức **CRUD**.

+ **PagingAndSortingRepository** – phân trang và sắp xếp.

+ **JpaRepository** – tích hợp và cải tiến thêm chức năng của các **interface** trên.

- **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**).

- **Persistence Context** – là một tập các thể hiện của **Entity**, được dùng quản lý **Entity** trong một kho dữ liệu.

- **EntityManager** – là 1 **interface** cung cấp các phương thức để thao tác với **entity**:

+ **persit**( ) – thêm 1 entity.

+ **merge**( ) – chỉnh sửa, cập nhật.

+ **remove**( ) – xóa.

+ **find**( ) – tìm kiếm.

- **Một số annotation của Hibernate:** @Entity, @Table, @JoinTable, @Id, @GenerateValue, @Column, @JoinColumn, @OneToOne, @OneToMany, @ManyToOne, @ManyToMany,…

- **Trạng thái của các Entity** (4 trạng thái):

+ **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**.

- **Truy vấn động và truy vấn tĩnh:**

+ **Câu truy vấn tĩnh** không thể truyền tham số trong lúc **complie-time**, **câu truy vấn động** có thể truyền tham số và thay đổi linh động tùy thuộc vào giá trị tham số truyền vào.

+ 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.

- **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.

- **Session Factory** – là 1 **interface** giúp tạo ra **session** kết nối đến **database** bằng cách đọc các cấu hình trong **Hibernate configuration**. Mỗi một **database** phải có một **session factory**.

- **@Configuration** – là 1 **annotation** của **class**. **Class** được đánh dấu annotaiton này được **Spring Container** sử dụng làm nguồn định nghĩa **bean**.

- **@EnableAutoConfiguration** – tự động cấu hình **bean** có trong **classpath** và cấu hình nó để chạy các phương thức.

- **@SpringBootApplication** – là sự kết hợp của **@EnableAutoConfiguration**, **@ComponentScan** và **@Configuration**.

**6. SPRING BOOT**

- **Spring boot** – là 1 trong số các module của Spring Framework, đc sử dụng để tạo ra các ứng dụng độc lập có thể chạy ngay với rất ít cấu hình thêm.

Ưu điểm:

+ Tạo ứng dụng độc lập, chạy ngay bằng cách sử dụng lệnh **Java**.

+ Nhúng sẵn server **Tomcat**, **Jetty**.

+ Cung cấp sẵn các thư viện “**starter**”.

+ Tự động cấu hình **Spring**.

+ Không yêu cầu cấu hình **XML**.

- **So sánh Spring MVC và Spring Boot:**

|  |  |
| --- | --- |
| **Spring Boot** | **Spring MVC** |
| - Là một **module** của **Spring** để đóng gói ứng dụng dựa trên **Spring** với các giá trị mặc định hợp lý.  - Cung cấp các cấu hình mặc định để xây dựng khung công tác được cung cấp bởi **Spring**.  - Không cần xây dựng cấu hình theo cách thủ công.  - Tránh mã viết sẵn và kết hợp các phần phụ thuộc lại với nhau trong một đơn vị duy nhất.  - Làm giảm thời gian phát triển và tăng năng suất. | - **Spring MVC** là **model-view-controller** web framework trong khuôn khổ **Spring framework**.  - Cung cấp các tính năng sẵn sàng sử dụng để xây dựng một ứng dụng **web**.  - Yêu cầu cấu hình xây dựng theo cách thủ công.  - Nó chỉ định từng phụ thuộc riêng biệt.  - Cần nhiều thời gian hơn để phát triển. |

**7. SPRING DATA REPOSITORY**

- **Package** – là đối tượng hỗ trợ phân trang.

- **Các thuộc tính / phương thức thông dụng:**

+ **number** – trang hiện tại (bắt đầu từ 0).

+ **totalPage** – tổng số trang.

+ **hasNext( )** – kiểm tra có trang tiếp theo hay không.

+ **hasPrevious( )** – kiểm tra có trang trước đó hay không.

- **Cách triển khai Phân trang & Sắp xếp:**

+ Ở **controller** sử dụng annotation **@PageableDefault** để khai báo tham số truyền vào **handle method** của đối tượng **pageable**.

🡪 Thuộc tính **value** = số record / 1 trang, (mặc định value = 10).

🡪 Thuộc tính **sort** = "tên trường muốn sắp xếp theo", **direction** = **Sort.Direction.DESC** (**DESC** là sắp xếp giảm dần, **ASC** là sắp xếp tăng dần).

**8. VALIDATION**

- **Validation** – thực hiện đánh giá / xác minh tính hợp lệ của dự 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:**

|  |  |  |
| --- | --- | --- |
| **UI layer** | **Business layer** | **Data layer** |
| - Phía **client**.  - Dùng ngôn ngữ **front-end** (JS, HTML,…).  - **Validate** trực tiếp, tốc độ phản hồi nhanh, mượt, cân bằng tải (giảm tải cho **server**).  - Bảo mật không cao, nhiều dữ liệu không xử lý đc (ví dụ như xử lý trùng lặp, xử lý ngày, giờ,…). | - Phía **server**.  - Dùng ngôn ngữ **server** (Java, C#, PHP,…).  - Ràng buộc chặt chẽ, độ chính xác cao, xử lý đa dạng.  - Nhiều xử lý khi có nhiều truy cập 🡪 **server** quá tải. | - Phía **DB server**.  - Dùng **SQL** để tạo ràng buộc (not null, check, primary,…).  - Cú pháp đơn giản, ngắn gọn, dễ sử dụng.  - Khó bắt lỗi nhập liệu, tốc độ rất chậm (còn phải xét đến sự liên kết giữa các **table** – **CSDL** quan hệ). |

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

- **Các phương pháp xác thực:**

**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).

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

🡪 Tại **thymeleaf** dùng **th:errors** để hiển thị lỗi.

**2. Dùng custom validation:**

🡪 Tại **dto**, cần **implement** interface **Validator**.

🡪 Tại **controller**, gọi phương thức **validate** bởi **dto**.

**3. Custom annotation:** tự tạo **annotation**.

**9. AOP**

- **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**).

+ Ví dụ:

**Chức năng login**: cho phép người dùng đăng nhập, hiển thị lỗi khi nhập sai.

**Chức năng phụ**: ghi số lần đăng nhập thành công/thất bại, bộ lọc/**security**, **code** liên quan đến xử lý **exception**.

+ Những tính năng phụ trên đc gọi là **Aspect** (khía cạnh).

- **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.

🡺 **Aspect** tương tự như 1 **Java class**, nó đóng gói toàn bộ **cross-cutting concern**.

- **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**.

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

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

+ **After returning 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**.

- **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,…

🡺 Có thể tạm hiểu là, có 1 anh chàng tên **AOP**, sinh năm 2k1, quê ở bản LakKonKu – Gia Lai thầm crush cô nàng tên **OOP** sinh năm 2k, quê Núi Thành, Quảng Nôm. Dù làm gì, chàng trai trẻ cũng luôn nhìn về phía cô nàng này, điều này như là 1 mối quan tâm xuyên xuốt. Đơn phương hoài cũng chạnh lòng, anh chàng này đã mạnh dạn hẹn gặp cô nàng đó. Họ đã gặp nhau (**weave**) tại điểm hẹn (**join point**). Chàng trai tân đã đưa ra những lời bộc bạch, lời thả thính, lời khuyên (**advice**)… Và, nếu cuộc hẹn suông sẻ, thì sẽ có những cuộc hẹn tiếp theo, chuỗi hẹn hò của câu chuyện tình cảm lạnh này chính là **point cut**. Suốt nửa năm hẹn hò, ăn uống, xem phim, ngắm bờ kè cùng nhau,… bỗng dưng 1 hôm cô gái hỏi chàng trai: “Anh tốt như vậy mà đến giờ vẫn chưa có người yêu à?”

- **Cách triển khai AOP:**

🡪 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.

- **Exception handle:**

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

+ Bảo mật, tránh bị kẻ xấu xâm nhập.

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

- **Các phương thức xử lí ngoại lệ:**

+ 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.

(có thể sử dụng **@RestControllerAdvice** hoặc **@ControllerAdvice** để định nghĩa 1 lớp nào đó để đón và xử lý ngoại lệ, thay vì phải xử lý trực tiếp ở **controller**).

**10. SESSION & COOKIE**

- **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,…).

- **Cookie** – 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).

- **So sánh cookie và session:**

+ **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.

- **Cách triển khai Session:**

🡪 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.

(Giống như ứng dụng **shopping cart**. Khi người dùng chọn 1 sản phẩm thì ta dùng **session** để lưu lại. Khi khách hàng thanh toán giỏ hàng thì ta lấy hết tất cả các mặt hàng chứa trong **session** ra và tính toán).

Nếu muốn triển khai thêm **cookie**, thì:

🡪 Sử dụng **annotation** **@CookieValue** để ràng buộc giá trị của **cookie HTTP** với tham số phương thức trong **Controller**.

🡪 Tạo **Cookie** và trả về cho **Client**.

**11. WEBSERVICE**

- **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** (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).

- **REST** (Representation 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. Giao thức sử dụng **XML** hoặc **JSON** (tốc độ nhanh hơn nhưng độ bảo mật và an toàn thấp hơn **SOAP**).

+ 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,…

- **So sánh Web application và Web service:**

|  |  |
| --- | --- |
| **Web application** | **Web service** |
| - Có giao diện.  - Tương tác giữa người dùng và ứng dụng, thông qua **form**, **link**, **image**, **button**,…  - Kết quả trả về cho người dùng là **view** hình ảnh, âm thanh, video,… | - Không có giao diện.  - Tương tác giữa các ứng dụng với nhau, thông qua các phương thức **HTTP** (GET, POST, PUT, DELETE,…) và **HTTP status code**.  - Kết quả trả về cho người dùng chỉ là data **XML** hoặc **JSON** (khó đọc đối với người dùng). |

- **Rest** là kiểu cấu trúc cung cấp các quy tắc để xây dựng **Webservice**.

Các quy tắc cơ bản của cấu trúc **REST**:

+ Truy cập tài nguyên – **GET**.

+ Tạo tài nguyên trên máy chủ – **POST**.

+ Xóa tài nguyên – **DELETE**.

+ Chỉnh sửa tài nguyên – **PUT** (cập nhật toàn phần) hoặc **PATCH** (cập nhật 1 phần).

- **Restful** là **phi trạng thái** (ứng dụng **Restfull** không lưu trữ thông tin, phiên làm việc của **client**).

- **@JsonManagerReference** – ngăn đệ quy.

- **@JsonBackReference** – ngăn ánh xạ ngược.

- **@JsonIgnore** – không xuất hiện trường **Collection**.

**11. JQUERY & AJAX**

- **DOM** (**Document Object Modal**) – là mô hình các đối tượng trong tài liệu **HTML**.

+ Cung cấp cách truy cập hoặc thay đổi nội dung cải thiện **HTML**.

+ **JS** truy cập phần tử của **DOM** thông qua đối tượng có sẵn là **document**.

- **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.

- **Gọi hàm trong Jquery:**

+ Có 2 cách: sử dụng **$(…)** hoặc **Jquery(…)**.

+ Bộ chọn (**selector**) trong **Jquery**:

- Univesal (chọn tất cả các phần tử có sẵn trong **DOM**): **$(\*)**.

- Tag/ element: **$(“**tên thẻ**”).**

- Bộ chọn id: **$(“#**id**”).**

- Bộ chọn class: **$(“.**class**”).**

- Truy cập vào thẻ HTML hiện tại: **$(this).**

- …

- **Đồng bộ** – là xử lý mà chương trình chạy theo từng bước, nghĩa là thực hiện xong đoạn **code** này thì đoạn **code** kế tiếp mới đc thực hiện.

- **Bất đồng bộ** – chương trình có thể hoạt động nhảy cóc, đoạn **code** phía dưới có thể thực hiện xong mà không cần chờ **function**/đoạn **code** nào đó phía trên thực hiện xong.

- **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; **}**

**})**

- **Cách triển khai Ajax:**

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

🡪 Có thể dùng **jQuery** để bắt sự kiện.

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

**12. SPRING SECURITY**

- **Security** (**bảo mật**) – là sự hạn chế khả năng lạm dụng tài nguyên và tài sản. Gồm cơ chế bảo mật và phân quyền.

- **Spring Security** – là một **Framework** của **Spring**, dành riêng cho việc thiết lập bảo mật ứng dụng bao gồm **Authentication** và **Authorization**.

- **Authentication** (**xác thực**) – là 1 cơ chế / hành động nhằm thiết lập hoặc chứng thực 1 thông điệp / đối tượng là đáng tin cậy.

- **Authorization** (**phân quyền** / **cấp quyền**) – là quá trình xác định xem 1 đối tượng / người dùng có quyền truy cập vào 1 tài nguyên cụ thể nào đó để thực hiện 1 số hành động **CRUD**.

- **Các cơ chế xác thực:**

+ **HTTP basic**: yêu cầu client cung cấp username và password để thực hiện xác thực.

🡪 **username** và **password** đc gửi cùng với mọi yêu cầu, có khả năng hiển thị chúng, ngay cả khi đc gửi qua kết nối an toàn.

🡪 yêu cầu thay đổi mật khẩu thường xuyên.

+ **Cookies**.

🡪 cần phải nỗ lực nhiều hơn để giảm thiểu các cuộc tấn công **CSRF**.

🡪 không tương thích với **REST**, vì nó đưa vào một trạng thái thành một giao thức không trạng thái

+ **Oauth 2** (Open Authorization): xác thực thông qua ứng dụng thứ 3.

+ **Token**: JWT (Json Web Token).

🡪 là 1 tiêu chuẩn mở định nghĩa cách thức truyền tin an toàn giữa các thành viên bằng 1 đối tượng **JSON**. Thông tin này có thể được xác thực và đánh dấu tin cậy nhờ vào "chữ ký" đã đc mã hóa.

+ **Signatures**.

🡪 là mô hình sử dụng các kỹ thuật mật mã để gắn với mỗi người sử dụng 1 cặp khóa công khai – bí mật và qua đó có thể ký các văn bản điện tử cũng như trao đổi các thông tin mật.

+ **One time password** (OTP).

🡪 người dùng nhập **username** và **password**, sau đó cả máy chủ lẫn máy khách đều tạo mật khẩu một lần.

+ **SSO** (Single sign-on).

🡪 đăng nhập 1 lần có thể vào đc nhiều trang khác nhau (ví dụ như đăng nhập vào trang **james.codegym.vn** thì có thể vào đc trang **ken.codegym.vn** và **bob.codegym.vn**,…).

+ Ngoài ra, còn nhiều cơ chế khác như **Sich trắc học** (vân tay, nhận diện khuôn mặt,…),…

- **Các thành phần cốt lõi của Spring Security:**

+ **SecurityContext** – là **interface** lưu trữ tất cả thông tin liên quan đến bảo mật ứng dụng.

+ **SecurityContextHolder** – cách truy cập vào **SecurityContext**.

+ **UserDetails** – là **interface** đại diện cho **principal** (principal có thể hiểu nôm na là 1 con người, 1 thiết bị hoặc 1 hệ thống nào đó có thể thực hiện 1 hành động nào đó trong ứng dụng **Spring**).

+ **GrantedAuthority** – là quyền đc cấp cho **principal** (các quyền đều có tiền tố là “**Role\_xxxxx**”).

- **@EnableWebSecurity** - kích hoạt tính năng **Spring Security** trên ứng dụng Web của mình.

- **@ConfigurationProperties** – là **annotation** cho phép truy xuất các thuộc tính được đặt trong các tệp cấu hình một cách tự động và mạnh mẽ.

- **I18n** (**Internationalization**) – là quá trình thiết kế một ứng dụng phần mềm để nó có thể đáp ứng được nhiều ngôn ngữ và khu vực khác nhau mà không cần thay đổi kỹ thuật.

- **L10n** (**Localization**) – là quá trình điều chỉnh phần mềm đã được quốc tế hóa cho một ngôn ngữ hoặc một khu vực cụ thể bằng cách chỉ định ngôn ngữ hoặc khu vực sau đó dịch văn bản.

- **Cách triển khai I18N và I10N trong dự án Spring:**

🡪 Cấu hình **Java**: LocalResolver, LocalChangeInterceptor, InterceptorRegistry, MessageSource.

🡪 Tạo file **Resource Bundle** để khai báo **message** bằng các ngôn ngữ khác nhau.

🡪 Sử dụng **Thymeleaf** để gán các **message** vào **HTML**.

- **Interceptor (chốt chặn):**

// Khi bạn tới công ty và muốn gặp sếp của công ty đó. Bạn cần phải đi qua các chốt chặn (**Interceptor**), các chốt chặn ở đây có thể là người bảo vệ cổng, nhân viên lễ tân,..

+ Trong **Spring**, khi một **request** được gửi đến **controller**, trước khi **request** được xử lý bởi **Controller**, nó phải vượt qua các **Interceptor** (0 hoặc nhiều).

+ **Spring Interceptor** là một khái niệm khá giống với **Servlet Filter**.

+ **Spring Interceptor** chỉ áp dụng đối với các **request** đang được gửi đến một **Controller**.

- **CSRF** – hay còn gọi là kỹ thuật tấn công “**Cross-site Request Forgery**“, nghĩa là kỹ thuật tấn công giả mạo chính chủ thể của nó.

**CSRF** nói đến việc tấn công vào chứng thực **request** trên **web** thông qua việc sử dụng **Cookies**. Đây là nơi mà các **hacker** có khả năng sử dụng thủ thuật để tạo **request** mà bạn không hề biết. Vì vậy, một **CSRF** là **hacker** lạm dụng sự tin tưởng của một ứng dụng **web** trên trình duyệt của nạn nhân.

**🡺 Ví dụ:**

Người dùng đã đăng nhập trên **web** của bạn, **cookie** sẽ được tạo và lưu trữ dưới trình duyệt, khi bạn vào **website** lần sau bạn không cần phải đăng nhập lại. Giả sử bạn chưa đăng xuất, lúc này **cookies** của bạn vẫn còn hạn trong phiên làm việc.

Lúc này nếu **website** của bạn mắc lỗi **CSRF**, người dùng vô tình vào một trang **hacker** giả mạo với mục đích lấy tài khoản từ ứng dụng **web** của bạn. Trong trang giả mạo **hacker** sẽ dùng **script** để chạy một **url** để cố ý **reset** mật khẩu người dùng trên trang của bạn.

- **CORS** (**Cross-origin resource sharing**) – là một cơ chế cho phép nhiều tài nguyên khác nhau (**fonts**, **Javascript**, v.v…) của một trang **web** có thể được truy vấn từ **domain** khác với **domain** của trang đó.

**CORS** được sinh ra là vì **same-origin policy**, là một chính sách liên quan đến bảo mật được cài đặt vào toàn bộ các trình duyệt hiện nay. Chính sách này ngăn chặn việc truy cập tài nguyên của các **domain** khác một cách vô tội vạ.

**🡺 Ví dụ:**

Bạn truy cập một trang **web** có mã độc. Trang **web** đó sử dụng **Javascript** để truy cập tin nhắn **Facebook** của bạn ở địa chỉ **https://facebook.com/messages**.

Nếu bạn đã đăng nhập **Facebook** từ trước rồi. Nếu không có **same-origin policy**, trang web độc hại kia có thể thoải mái lấy dữ liệu của bạn và bất cứ điều gì chúng muốn.

\_**The Last Shinobi ϟ\_**