--
--
========== SPRING MVC ==========

********** BÀI 1 **********

1. KHÁI NIỆM VÀ Ý NGHĨA CỦA FRAMEWORK:
- Framework: là các ứng dụng phần mềm có tính trừu tượng(abstraction), cung cấp các tính năng chung và thông dụng có thể tùy biến để tạo nên các ứng dụng cụ thể khác nhau.
--> Mỗi framework sẽ cung cấp 1 phương pháp riêng biệt để phát triển và xây dựng ứng dụng.
(Spring framework, Java Collection, Laravel, .NET,...)
- Framework bao gồm môi trường tổng thể, tái sử dụng được, nhằm cung cấp các chức năng, công cụ để hỗ trợ quá trình phát triển ứng dụng.
- Điểm khác biệt lớn nhất giữa library và framework là IOC - Inversion of Control.
Your code call library.
Framework call your code.
Trong Framework có thể chứa các Library.

2. CÁC THÀNH PHẦN CỦA SPRING FRAMEWORK:

- Spring là gì? Là 1 Framework phát triển các ứng dụng java. Là 1 mã nguồn mở, được support bởi 1 cộng đồng lớn.

- CÁC THÀNH PHẦN CỦA SPRING:
+) Core Container: Beans(đối tượng do IoC Container tạo và quản lý, có thể dùng để thiết kế Design Pattern Singleton), Core, Context(kết nối các thành phần với nhau), SpEL(Spring Expression Language - giúp ta viết các file cấu hình đơn giản hơn).
+) AOP, Aspects,...
+) JDBC, ORM,...
+) Web, Servlet, WebSocket,...

- Tổng quan về Spring MVC: là 1 nền tảng mã nguồn mở để phát triển ứng dụng java, được cài đặt các tính năng đầy đủ của MVC pattern.

Nó khác với MVC truyền thống là nó cung cấp 1 Fron Controller để xử lý và lắng nghe mỗi khi có request đến ứng dụng.
--> Lợi ích của Front Controller: bảo mật, hỗ trợ I18N(đa ngôn ngữ),...

- CÁC FILE CONFIG CHÍNH CHO PROJECT SPRING MVC:
web.xml : khai báo dispatcher servlet(front controller) và file context
applicationContext.xml: khai báo các bean.
dispatcher-servlet.xml: config cho Spring biết được là tạo bean ở các package nào(scan, annotation)).
Và config đường dẫn của thư mục chứa view.

- ViewResolver: là cơ chế để xử lý tầng view của Spring MVC. Có nhiệm vụ ánh xạ tên của view sang đối tượng view tương ứng.

********** BÀI 2: SPRING CONTROLLER **********

1. Khái niệm về controller.
- Là 1 thành phần trong mô hình MVC có nhiệm vụ điều hướng request(chú ý không code nhiều logic ở controller, logic ở service giải quyết), thường gọi model để nhận dữ liệu, sau khi dữ liệu được xử lý thì trả về lại cho view.
- Để tạo Controller, ta phải tạo class và implements interface Controller.
Nhưng từ Spring2.5 ta sd annotation @Controller
hoặc @RestController(dùng cho web service) để khai báo 1 controller cho Class.

2. @Controller và @RestController
@Controller: khai báo controller trong ứng dụng website truyền thống.
@RestController: sử dụng trong việc khai báo controller trong ứng dụng webservice.

- Để tự động phát hiện Controller trong Spring MVC thì ta khai báo giống:
<context:component-scan base-package="DiamonShop" />
Hoặc có thể config bằng file .java: @ConponentScan("DiamonShop")

3. @RequestMapping: được sd để ánh xạ các request đến các action tương ứng của controller.
VD: @RequestMapping(value = "/login", method = RequestMethod.POST)
NÓi chung nó chứa các thông tin của:

+ Http method -> method=...
+ url -> value
+ Các tham số -> params
+ header của request -> headers
+ MediaTypes -> consumes

Ngắn gọn hơn => dùng các biến thể dành cho HTTP method:
@PostMapping(value = "/login")
@GetMapping
@PatchMapping
@PutMapping
@DeleteMapping

4. Các thuộc tính của @RequestMapping.
- value, method
- consumes: ánh xạ đến Content-Type của request -> quy định loại dữ liệu
mà client gửi đến server để xử lí.
- produces: ánh xạ đến thuộc tính Accept của request -> quy định loại dữ liệu
mà server được phép trả về client.
- params: ánh xạ tới tham số trên url
- headers: ánh xạ đến header của request.

5. @RequestParam và @PathVariable

- Với @PathVariable:
url request có dạng: ..../2 -> 2 là id

@GetMapping(value = "detail/{id}")
public String getDetailStudent(@PathVariable("id") int id){...}; -> ưu tiên dùng.
Nếu đặt tên tham số của hàm giống tên tham số trên url thì chỉ cần:
public String getDetailStudent(@PathVariable int id){...};

- Với @RequestParam:
url request có dạng: ....?id=2 -> 2 là id
@GetMapping(value = "detail")
public String getDetailStudent(@RequestParam("id") int id){...};

===> So sánh @RequestParam và @PathVariable
- Giống: đều để lấy dữ liệu từ client gửi lên.
- Khác:
+ @RequestParam: gửi giá trị theo cặp key-value -> thường sd trong form
+ @PathVariable: lấy dữ liệu theo dạng value -> thường được sd lấy dữ liệu từ url.

6. Redirect và forward
- Redirect : yêu cầu 1 chuyến khứ hồi tới server
- Forward : trả về reponse luôn
--> Forward sẽ nhanh hơn redirect.
--> Tuy nhiên, forward có nhược điểm: không thể chuyến hướng trang web ra bên ngoài của web hiện tại.
- Dùng redirect giúp người dùng tránh việc gọi lại cùng 1 URL khi người dùng reload lại trang web.
---> Redirect thích hợp khi bạn muốn người dùng thấy một URL mới và tạo ra một request mới từ trình duyệt của họ. Điều này thường xuyên đ ược sử dụng cho việc điều hướng giữa các trang và khi bạn muốn chia sẻ URL có thể bookmark được.

Forward thích hợp khi bạn muốn chuyển tiếp người dùng ngay trong cùng một request và không muốn thay đổi URL trên trình duyệt của họ. Forward thường được sử dụng khi bạn muốn thực hiện các xử lý nội bộ và chuyển tiếp kết quả sang một controller hoặc view khác.

Redirect tạo ra 2 request:
Khi sử dụng redirect, trình duyệt của người dùng sẽ gửi một request đến URL mới. Điều này có thể dẫn đến việc tạo ra hai request HTTP: một request để xử lý chuyển hướng và một request mới từ trình duyệt của người dùng đến URL mới.

Forward tạo ra 1 request:
Trong trường hợp của forward, không có request mới từ trình duyệt. Thay vào đó, controller hoặc servlet hiện tại sẽ trực tiếp chuyển tiếp kết quả của request hiện tại đến một endpoint hoặc view khác trong cùng một request.

- Trong trường hợp để url hiện thị đúng ý nghĩa của trang thì nên dùng redirect.

Ví dụ: đang ở url /create. Nếu ở đây ta tạo 1 đối tượng mới và forward đến trang jsp list.jsp thì trước khi ta forward ta phải add model cho nó và khi chuyển trang list.jsp thì url vẫn hiện là /create -> ko tường mình.

Thay vì đó, ở url /create ta redirect:/list -> nó sẽ tìm đến @GetMapping("/list") và phương thức dưới @GetMapping("/list") sẽ sữ lý add model và forward cho ta. Giúp ta có url /list tường mình hơn.

7. Handler Method và các kiểu trả về

- Handler Method: trong class Controller, mỗi phương thức xử lý 1 action cho 1 request -> gọi là Handler Method.
- Có các tham số: @PathVariable, @RequestParam, Model,...
- Trả về: String, ModelAndView, void, @ResponseBody, @ModelAttribute,...

****** BÀI 3: SPRING DATABINDING VÀ FORM ******

1. Khái niệm về data binding:

- Là cơ chế liên kết dữ liệu đầu vào, đầu ra với model.
- Tự động chuyển đổi các dữ liệu trên form thành các thuộc tính của các đối tượng liên kết với nó.

-> giúp ta tương tác dễ dàng hơn vì các form đều sẽ liên kết với dữ liệu.
Hỗ trợ chuyển đổi dữ liệu + validate dữ liệu.

2. Cơ chế hoạt động của Data Binding trong Spring.

Text Input(.jsp,.html)
=> DataBinder(Fomatters, ConversionService, PropertyEditors, Validators)
=> Bean(đối tượng liên kết với form), BindingResult(thành công or th bại).

3. Spring form: hỗ trợ tạo form trong form.

- Ngoài các thuộc tính bình thường, nó còn hỗ trợ 1 thuộc tính quan trọng là modelAttribute -> giúp ta thực hiện cơ chế binding.
- Trong <form:form....>:

+) modelAttribute = "student": đại diện cho model được binding vào.
+) path: liên kết tên thuộc tính của model với trường hiện tại(input, checkbox,...)
- Để binding model giữa view và controller: Bên controller ta sd:
@ModelAttribute("student") Student student;

- Hiện thị danh sách với select, checkbox:

```
<form:select path="classRoom">
        <form:option value="" label="-- Select Class Room --" />
        <form:options items="${classRooms}" itemValue="id" itemLabel="name" />
</form:select>
```

==> public class ClassRoom {
    private Long id;
    private String name;

    // constructor, getters, setters
}
==> itemValue="id":
Chọn thuộc tính id của mỗi đối tượng ClassRoom làm giá trị của mỗi option.
itemLabel="name":
Chọn thuộc tính name của mỗi đối tượng ClassRoom làm nhãn để hiện thị ra cho mỗi option.

********** BÀI 4: THYMELEAF **********

1. Khái niệm về Thymeleaf: là bộ xử lý view, sd trong các ứng dụng web và các ứng dụng độc lập. Được xây dựng phù hợp với tiêu chuẩn của web, đặc biệt là html (html 5)
- Cho phép xử lý các loại template: html, xml, text, javascript, css, raw.

2. Các biểu thức của Thymeleaf:

- Biểu thức với biến: ${}
- Biểu thức với thuộc tính: *{}
- Biểu thức với message: #{}

- Biểu thức với URL: @{}
- Biểu thức phân đoạn: ~{}

3. Vòng lặp trong THYMELEAF

Vd: <tr th:each="student,loopStatus :${studentList}">

<td th:text="${loopStatus.count}"></td>

<td th:text="${student.id}>
<td th:switch="${student.gender">

<span th:default th:text="LGBT">
</td>

<td th:each="language :${student.languageList}" th:text="${language} + ',' "></td>
</tr>

4. form với Thymeleaf

<form th:object="${student}" th:action="@{/student/create}" method="post">
<input th:field="*{id}">
<input th:field="*{name}">
<input type="radio" th:field="*{gender}" th:value="1" th:text="Male">
<input type="radio" th:field="*{gender}" th:value="0" th:text="Female">
<input type="radio" th:field="*{gender}" th:value="2" th:text="LGBT">
<input type="checkbox" th:field="*{languages}"
th:each="language:${languageList}" th:text="${language} th:value="${language}"'/>

***Chú ý: languages trong field là thuộc tính của student, chứa danh sách các ngôn ngữ lập trình.

Khi ta check vào các box để chọn các ngôn ngữ thì nó được ánh xạ luôn vào thuộc tính này.

***Chú ý: languageList là danh sách các ngôn ngữ ta truyền từ controller sang

để nó hiện thị lên form cho người dùng chọn.

</form>

5. Câu điều kiện trong Thymeleaf

th:if="dieu_kien"

th:if trả về true trong các trường hợp sau:
+ dieu_kien: là 1 giá trị boolean true
+ dieu_kien: là 1 giá trị khác 0
+ dieu_kien: là 1 chuỗi ký tự khác "0"
+ dieu_kien: là 1 chuỗi khác các chuỗi "false", "no", "off"
+ dieu_kien: là 1 giá trị không phải boolean, số, ký tự hoặc chuỗi.

********* BÀI 5: ORM **********

1. Khái niệm về ORM
- Object Relational Mapping, là 1 kỹ thuật liên kết giữa các đối tượng trong lập trình với các đối tượng trong CSDL.
--> ORM cho phép ta truy xuất dễ dàng đến dữ liệu thông qua các đối tượng lập trình mà khôg cần quá quan tâm đến CSDL.

Object(OOP) -> ORM <- Table(RDB)

Giúp cho việc thay đổi csdl cũng đơn giản hơn. Ví dụ đổi từ mysql -> sql server.
Ta chỉ cần sửa lại phần config là ok. Các câu query ko ảnh hưởng nhiều.

2. Ưu điểm của ORM:
- Quản lý dữ liệu tập trung trong code.
- Tránh được các lỗi liên quan đến SQL.
- Hỗ trợ giao dịch(transaction)
- Hỗ trợ bộ nhớ đệm cache -> lần truy xuất sau sẽ truy xuất nhanh hơn.

3. Nhược điểm của ORM:
- Lập trình viên dễ rơi vào bẫy truy xuất dữ liệu(vì quá dễ truy xuất do sd hàm hỗ trợ có sẵn -> truy xuất nhiều hơn mức cần thiết)-> ảnh hưởng đến

hiệu năng của chương trình.
- Đối với các thao tác phức tạp, có thể cần đến việc sử dụng sql thuần(
thuần nhưng ko phải sd JDBC, mà là tự viết câu lệnh query và ORM sẽ
thực hiện câu lệnh query đó giúp ta)

4. Khái niệm về JPA và Entity
- Java Persistence API - JPA cung cấp các đặc tả(interface) để duy trì, đọc,
quản lý dữ liệu từ đối tượng java sang các quan hệ trong CSDL.
- JPA chỉ chứa các interface.
- Hibernate sẽ triển khai các abstract method của JPA.
- JPA cung cấp 1 cái mô hình POJO persistence - cho phép ánh xạ các
table/các mối quan hệ trong Database sang các class/mối quan hệ giữa các Object.
- JPA quản lý các Entity:
+ Là đối tượng đại diện cho dữ liệu trong ứng dụng.
+ Entity thường là POJO.
+ Entity sẽ được ánh xạ tới 1 bảng trong csdl.
+ Để ánh xạ được, 1 entity cần tuân thủ các điều sau:
- Khai báo @Entity(name="ten_bang") trên class của entity đó.
- Phải có 1 constructor không tham số, để access modifire là public.
- Không được khai báo final.
- Các thuộc tính không khai báo public.
- Đối với các thuộc tính của entit thì nên:
@Column(name="date_of_birth", columnDefinition="DATE")
private String dateOfBirth;
- Với khóa chính:
@Id
@GeneratedValue(stratery = GenerationType.IDENTITY)
private Integer id;
- Với khóa chính là chuỗi ta có thể Custom GeneratedValue.

5. Persistence Context và Entity Manager trong Spring.
- Persistence Context là 1 tập các thể hiện của Entity được quản lý,
tồn tại trong 1 kho dữ liệu.
Persistence Context là nơi mà JPA quản lý các đối tượng của bạn khi
chúng được đọc từ cơ sở dữ liệu hoặc khi bạn lưu các thay đổi vào cơ sở
dữ liệu.

Khi một đối tượng được đọc từ cơ sở dữ liệu, nó trở thành một phần của Persistence Context.

Mục tiêu của Persistence Context là đảm bảo sự nhất quán giữa dữ liệu
trong cơ sở dữ liệu và các đối tượng trong ứng dụng.

Persistence Context cũng hoạt động như một cache đối tượng.
Khi bạn truy vấn cơ sở dữ liệu để đọc một đối tượng,
JPA có thể kiểm tra xem đối tượng đã được tải vào Persistence Context chưa.
Nếu đã tồn tại, nó có thể trả về đối tượng từ Persistence Context
thay vì thực hiện một truy vấn mới.

- Entity Manager: cung cấp các phương thức để tương tác với Persistence Context.

6. HQL - Hibernate Query Language
VD: "select s from Student s"

7. Câu lệnh truy vấn động và truy vấn tĩnh.
- Truy vấn động: có thể truyền tham số trong quá trình runtime.
--> tránh được lỗ hổng bảo mật sql injection
- Truy vấn tĩnh: Truy vấn tĩnh là loại truy vấn mà bạn xác định
và xây dựng tại thời điểm biên dịch
-> truyền tham số cố định tại quá trình compile. -> dễ bị sql injection

- createQuery() --> hàm giúp ta truy vấn động.
- createNamedQuery --> truy vấn tĩnh.

- Có 2 cách để truyền tham số vào truy vấn động
(Trong JPQL (Java Persistence Query Language) hoặc HQL (Hibernate Query Language)):
+) Sử dụng nameParam và phương thức setParameter("nameParam", value)

String jpqlQuery = "SELECT e FROM Employee e WHERE e.department = :dept AND e.salary > :minSalary";
TypedQuery<Employee> query = entityManager.createQuery(jpqlQuery, Employee.class);
query.setParameter("dept", someDepartment);
query.setParameter("minSalary", minSalary);

```
List<Employee> result = query.getResultList();
```

+) Sử dung ?vịtrí tham số và phương thức setParameter(vị trí, vlaue)

```
String jpqlQuery = "SELECT e FROM Employee e WHERE e.department = ?1 AND e.salary > ?2";
TypedQuery<Employee> query = entityManager.createQuery(jpqlQuery, Employee.class);
query.setParameter(1, someDepartment);
query.setParameter(2, minSalary);
List<Employee> result = query.getResultList();
```

8. entityManagerFactory và entityManager

- EntityManagerFactory:

EntityManagerFactory là một đối tượng tạo ra các đối tượng EntityManager.
Nó đại diện cho một "container" cho các đối tượng quản lý.
Thường thì chỉ cần tạo một đối tượng EntityManagerFactory trong suốt thời gian chạy của ứng dụng. Điều này được thực hiện bằng cách sử dụng một cơ chế quản lý lifecycle như Dependency Injection hoặc Singleton pattern.
EntityManagerFactory chịu trách nhiệm về việc quản lý các thông số cấu hình, đối tượng kết nối cơ sở dữ liệu, và các thiết lập khác cần thiết để tạo và quản lý EntityManager.

```
EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory("myPersistenceUnit");
```

- EntityManager:

EntityManager là một giao diện cho phép tương tác với cơ sở dữ liệu thông qua JPA. Mỗi EntityManager liên kết với một "đơn vị kiểm soát (persistence unit)" cụ thể được định nghĩa trong tệp persistence.xml.
Thường thì mỗi truy vấn hoặc thao tác với cơ sở dữ liệu sẽ sử dụng một EntityManager mới.
EntityManager được tạo ra và đóng khi cần thiết.
EntityManager thực hiện các phương thức như persist, merge, remove, và find để quản lý đối tượng trong cơ sở dữ liệu.

```
EntityManager entityManager = entityManagerFactory.createEntityManager();

// Example of using EntityManager
entityManager.getTransaction().begin();
MyEntity entity = new MyEntity();
entityManager.persist(entity);
entityManager.getTransaction().commit();
```

********** BÀI 6 ORM JPA HIBERNATE, Spring Data JPA **********

1. Phân biệt ORM, JPA, Hibernate
- ORM, JPA
- Hibernate: là 1 ORM Framework. Hibernate implements các interface của JPA.

2. Spring Data JPA
- Giúp ta triển khai JPA trong ứng dụng Spring 1 cách dễ dàng hơn.
(không cần phải cấu hình EntityManagerFactory, EntityManager như JPA thuần bài 5)
-> cải tiến JPA tiêu chuẩn -> đơn giản hóa tầng truy xuất -> tự tạo ra repository,
tạo ra các câu lệnh truy vấn thông qua tên phương thức. Ngoài ra, giúp ta ghi log,
hay dễ dàng hơn khi thực hiện phân trang.
- Spring Data là 1 module của dự án Spring.
  Data JPA là 1 phần của Spring Data.

3. Các interface trong Spring Data Repository.
- JpaRepository(mới nhất, có all chức năng của 3 thằng kia)
- PagingAndSortingRepository
- CrudRepository
- Repository(cũ nhất)

pulic interface IStudentRepository extends JpaRepository<Student, Integer>{}
-> Student là entity muốn tương tác với database
-> Integer là kiểu dữ liệu của khóa chính(thường là id của entity)

4. Các phương thức có sẵn

- findById(idStudent).orElse(null) : nếu ko thấy trả về null.
- save(student) : save dùng cho cả cập nhật(khi có id) và
cả thêm mới(khi ko có id trong database)
- delete(student)
- delete(id)

5. Tự tạo phương thức

```
pulic interface IStudentRepository extends JpaRepository<Student, Integer>{

        // cách 1:
        List<Student> findAllByNameContaining(String name);

        // cách 2:
        @Query(câu truy vấn)
        List<Student> searchByName(String name);
```

CHÚ Ý: @Query(...) -> giúp ta viết theo 2 ngôn ngữ là SQL và HQL

- Nếu @Query(value="...", nativeQuery = true) -> viết truy vấn theo ngng SQL
- Nếu @Query(...) or @Query(value="...", nativeQuery = false) -> theo ngng HQL

- Nếu có tham số trong câu truy vấn có thể sd 2 cách:

```
"SELECT e FROM Employee e WHERE e.department = ?1 AND e.salary > ?2"
List<Employee> searchByName(String department, double salary);
```

hoặc

```
@Query("SELECT e FROM Employee e WHERE e.department = :dept AND e.salary > :minSalary")
List<Employee> searchByName(@Param("dept") String department, @Param("minSalary") double minSalary);
```

--> sử dụng @Param để ánh xạ giá trị từ các tham số của phương thức vào các tham số trong câu truy vấn.

```
}
```

6. @ManyToOne và @OneToMany.

Lưu ý: Trong 2 trường hợp sau, không nên/không cần dùng @OneToMany,
dùng mỗi bên @ManyToOne cũng OK:

- Không dùng đến list.
- Không sử dụng casade.

("Cascade" là một thuật ngữ thường được sử dụng trong ngữ cảnh của ORM (Object-Relational Mapping)
để mô tả cách quản lý và tự động lan truyền các thay đổi từ một đối tượng đến
các đối tượng liên quan.)

********** BÀI 7 SPRING BOOT **********

1. Tổng quan về Spring boot
- Spring boot là 1 trong các module của Spring, cung cấp tính năng RAD.
RAD - Rapid Application Development.
-> Tạo 1 ứng dụng độc lập dựa trên Spring mà có thể chạy ngay với ít cấu hình.
-> Không cần dùng xml để cấu hình mà thay vào đó nó sử dụng quy ước để cấu hình.
-> giảm tải công việc cho lập trình viên.

2. Ưu điểm của Spring BOOT
- Cũng cấp sẵn các thư viện "starter" để đơn giản hóa cấu hình maven hoặc gradle
- Tự động cấu hình spring khi có thể -> ko yêu cầu cấu hình bằng xml, mà cấu hình bằng file java.
- Được nhúng sẵn Tomcat, Jetty hoặc Undertow.-> ko cần triển khai các tệp war.
- Tạo ứng dụng Spring độc lập có thể được chạy bằng cách sử dụng lệnh java.
- Cung cấp các tính năng "ăn liền": đo đếm số liệu, kiểm tra cấu hình của ứng dụng.

3. Phân trang bằng pageable và slide.
- Interface PagingAndSortRepository: có thêm chức năng phân trang và sắp xếp.
Cho phép truyền các tham số đặc biệt: Pageable, Slice, Sort.
- Pageable và Slice đều hỗ trợ phân trang nhưng:
+) Pageable biết được tổng số trang -> Slice thì không.
+) Slice có hiệu năng tốt hơn -> do ko cần đếm tổng số trang -> thường dùng cho trường hợp dữ liệu lớn.

4. Converter và Formatter
- Chuyển đổi dữ liệu nhập vào thành kiểu dữ liệu phù hợp.
- Converter có thể sử dụng chung cho toàn bộ các tầng của ứng dụng.
- Formatter chỉ sử dụng được tầng web(web tier)

********** BÀI 8 VALIDATION **********

1. Khái niệm và ý nghĩa của Validation.
- Validation thực hiện đánh giá/xác minh tính hợp lệ của dữ liệu đầu vào.

2. Các tầng có thể validate
- view/UI
- controller, business layer
- data layer.
---> Spring hỗ trợ validation ở tầng Business layer.

3. Các annotation dùng để validate trong Spring.
- @NotNull (bắt khi giá trị là null),
  @NotEmpty (bắt khi giá trị ko null, nhưng rỗng),
  @NotBlank (bắt khi giá trị ko null,ko empty nhưng toàn dấu cách)
- @Size(min=, max=) -> độ dài của chuỗi, số
  ví dụ: @Size(min=2, max=2) -> chạy từ 10 -> 99
- @Max, @Min -> chỉ áp dụng cho kiểu số.
- @Pattern
- @Email -> của spring không chặt chẽ lắm, nó chỉ bắt có @ hay không.
- @Range(min = 1, max = 100) -> value chỉ được trong khoảng 1 -> 100

4. @Valid và @Validated
- @Valid : validate cho toàn bộ thuộc tính của model
- @Validated : validate 1 phần/nhóm của model.
- Lưu ý: BindingResult phải luôn luôn nằm ngay sau model.
  Nếu có 2 model thì cần 2 BindingResult.
VD:

```
public String saveStudent(@Valid @ModelAndAttribute("student") Student student,
                                                    BindingResult bindingResult{
        if(bindingResult.hasError()){
                /// return lại chính trang nhập dữ liệu đó.

                //Ưu điểm của bindingResult :
```

```
                // Nếu có lỗi thì BindingResult sẽ trả lại html nguyên si dữ liệu
                // mà trước đó ta đã nhập.
        }
        ///
}
```

5. Các cách khác để validate

- Tự Custom validation: implements Validator -> interface của Spring FRAMEWORK
- Tự tạo annotation

********** BÀI 9 : AOP **********

1. KHái niệm AOP - Aspect Oriented Programming - Lập trình hướng khía cạnh.
 - Là 1 kỹ thuật lập trình, nhằm phân tách chương trình thành các module riêng lẻ, phân biệt và không phụ thuộc lẫn nhau. -> phù hợp với dự án lớn.
 - AOP không phải để thay thế OOP, mà là bổ sung cho OOP.

2. Khái niệm về Aspect(khía cạnh).
 - Là những mối quan tâm xuyên suốt AOP.
 - Lập trình AOP -> phân tách các module -> mỗi module sẽ thực hiện 1 nhiệm vụ thì aspect là 1 class thực hiện nhiệm vụ này: advice và join point.
 Ví dụ facebook sẽ 1 aspect chuyên để ghi log theo dõi màn hình người dùng, khi ta chuyển sang màn hình shoppe search sản phẩm, nó sẽ ghi lại, và đặt quảng cáo về sản phẩm đó trên trang facebook của ta.
-> đó cũng được coi là 1 aspect, nó ko làm ảnh hưởng đến luồng chạy chính của facebook.

3. Các thuật ngữ quan trọng trong AOP.
- 2 loại concern:
+) Core concern/primary concern: là requirement, logic xử lý chính của chương trình(CRUD, tìm kiếm,...)
+) Cross-cutting concern: là logic xử lý phụ: bảo mật, ghi log, theo dõi, giám sát.

- Jointpoint: là 1 điểm trong chương trình, là những nơi có thể chèn cross-cutting

concern.
1 joinpoint có thể là 1 phương thức đuọc gọi, 1 throw được ném ra hoặc 1 field bị thay đổi.
- Pointcut: có nhiều cách để xác định joinpoint, những cách như vậy được gọi là pointcut -> kiểm tra nó có khớp với joinpoint hay không -> khớp thì advice thực hiện.
- Advice: những xử lý phụ được thêm vào xử lý chính, code đó được gọi là advice.
1:08:43 / 1:31:27

********** BÀI 10 SESSION VÀ COOKIE **********

1. Khái niệm về cookie:
- Là các tệp được trang web người dùng truy cập tạo ra.
-> giúp trải nghiệm người dùng tốt bằng cách lưu thông tin duyệt web.
-> Tác dụng: duy trì trạng thái đăng nhập, ghi nhớ tùy chọn web, cung cấp nội dung phù hợp với ví trí của người dùng.
- tệp tincookie được truyền từ server và lưu tại máy của người dùng.
- Dù có tắt browser cũng không mất đi các giá trị của cookie vì chúng ta đã lưu nó trên máy tính của mình.

2. Cách sử dụng Cookie trong Spring.
- Khởi tạo cookie:
Cookie cookie = new Cookie("nameCookie", value) => lưu ý: "nameCookie" và value phải là String
cookie.setMaxAge(60*60*24) -> 1 ngày, tính theo giây.
response.addCookie(cookie);

- Ta sd annotation để lấy giá trị của cookie:
@CookieValue("nameCookie") DataType nameCookie;
Đặt annotation trong phần tham số của handle method.

- Xóa cookie của client từ phía server:
+B1 Đọc lại cookie đang tồn tại trong client.
+B2 set thời gian sống về 0
+B3 trả về cho client.

3. SESSION

- Session là 1 phiên làm việc giữa client và server.
- 1 phiên làm việc bắt đầu khi client thực hiện request đầu tiên đến server
và kết thúc khi client dừng làm việc với server.
- Các trường hợp kết thúc phiên làm việc:
  +) Đóng tag hoặc đóng browser(tuy nhiên, SESSIONID phía client ko bị xóa).
  +) Khi log out
  +) Khi trong 1 khoảng thời gian client ko tương tác với server.
- Trong 1 phiên làm việc thì biến session sẽ được tạo ra và lưu trữ trên server.
- Session hoạt động dựa trên cookie(JSESSIONID).

4. Cách sử dụng Session trong Spring.
- @SessionAttributes:(chú ý: có "s") để khai báo session
- @ModelAttribute bên ngoài phương thức -> để khởi tạo session.
- @ModelAttribute trong tham số của handle method để tương tác với session
NHƯNG CHỈ SỬ DỤNG ĐƯỢC BÊN TRONG CONTROLLER TẠO RA SESSION ĐÓ.
- @SessionAttribute: (chú ý:không có "s")trong tham số của handle method để tương tác với session
nhưng chỉ sử dụng bên ngoài controller tạo ra session đó.

5. So sánh Session và Cookie.
- Giống: đều tạo ra dữ liệu trong quá trình sử dụng và tương tác.
- Khác:
+) Cookie lưu trữ phía client, session lưu trữ phía server.
+) Có thể set thời gian sống cho cookie. Còn thời gian sống của session chỉ
trong 1 phiên làm việc.
+) Session có thể lưu trữ nhiều kiểu dữ liệu khác nhau.
Còn Cookie chỉ có thể lưu trữ kiểu String.
+) Session hoạt động dự trên cookie.

******* BÀI 11 WEB SERVICE **********

1. Khái niệm về Webservice.

PC,mobile,... <-> API <-> Server

- Web service (dịch vụ web) là các thành phần ứng dụng được hiển thị dưới dạng

các dịch vụ trên www.
- Web service có thể sử dụng để tích hợp với các ứng dụng được viết bằng các ngôn ngữ lập trình khác nhau và chạy trên nhiều nền tảng khác nhau.
-> Vì nó có chuẩn mở và sử dụng các giao thức mở để giao tiếp.
- Web service hoạt động như 1 server(bank-end) trong mô hình client - server.
-> giao tiếp thông qua giao thức HTTP/HTTPS.
- Với dữ liệu đầu vào xác định, web service xử lý và trả về dữ liệu theo 1 chuẩn đảm bảo mọi ứng dụng có thể hiểu và sử dụng mà không cần quan tâm đến loại thiết bị, hệ điều hành, kiến trúc phần mềm hay ngôn ngữ lập trình được sd.
-> nó thường trả về dữ liệu kiểu JSON hoặc XML.

2. So sánh Webservice và Website truyền thống.

- Website:
+) Có giao diện người dùng hoặc GUI.
+) Hiểu là được sử dụng bởi con người.
+) Hoạt động đa nền tảng, vì chúng yêu cầu tinh chỉnh để hoạt động trên các trình duyệt hay hệ điều hành khác nhau.
+) Được truy cập bởi các thành phần trong giao diện người dùng như button, textbox, form,....

-> NẾU VÍ TRONG DỰ ÁN CỦA TA THÌ CONTROLLER SẼ TRẢ VỀ VIEW.

- Web service:
+) Không có giao diện người dùng, chỉ có dữ liệu khô khan. Sau đó Front-End sẽ lấy dữ liệu đó đổ ra giao diện cho người dùng.
+) Hiểu là được sử dụng bởi các ứng dụng tương tác với nhau qua internet.
+) Độc lập về nền tảng và tất cả dạng truyền thống đều sử dụng giao thức chuẩn.
+) Được truy cập bởi phương thức HTTP: PUT, GET, POST, DELETE,....
Ví dụ: Google Maps API

-> NẾU VÍ TRONG DỰ ÁN CỦA TA THÌ CONTROLLER CHỈ TRẢ VỀ DỮ LIỆU THÔI.
Rồi từ dữ liệu đó Front-End sẽ đổ vào view tương ứng.

3. Ưu nhược điểm của webservice.

- Nhược điểm chính:
+) Một khi web service chết hoặc dừng hoạt động -> gây ra lỗi.
Ví dụ facebook dừng cho các website khác sử dụng api login bằng facebook
thì các web đó sẽ lỗi.
+) Cần quan tâm đến vấn đề an toàn và bảo mật nhiều hơn với web service khi
public cho bên thứ 3 sử dụng.
+) Việc có quá nhiều chuẩn cho web service dẫn đến người sử dụng khó nắm bắt.

- Ưu điểm:
+) Hoạt động trên các ứng dụng, nền tảng, hệ điều hành khác nhau.
+) Khả năng tái sử dụng cao -> nên giúp giảm thời gian phát triển và giảm sự
phức tạp của hệ thống.
+) Giúp xây dựng ứng dụng phân tán.

Một ứng dụng phân tán là một hệ thống phần mềm mà các thành phần chức năng của

nó được triển khai trên nhiều máy tính hoặc nhiều nơi khác nhau trong một mạng. Trong môi trường này, các thành phần của ứng dụng hoạt động cùng nhau để thực hiện một tác vụ cụ thể. Các thành phần này có thể giao tiếp với nhau thông qua mạng để chia sẻ dữ liệu, tài nguyên, hoặc để thực hiện các nhiệm vụ phức tạp.

4. Các loại webservice.

Có 2 loại chủ yếu:
- SOAP(Simple Object Access Protocol) là giao thức sử dụng XML để định nghĩa
dữ liệu và truyền dữ liệu thông qua HTTP.
- REST(Representational State Transfer) định nghĩa dữ liệu JSON và XML, truyền
thông qua mạng internet sử dụng giao thức HTTP.
JSON(JavaScript Object Notation)
--> Các web mà phát triển dựa trên REST -> RESTful.
- CÁC phương thức HTTP:
+) GET : lấy tài nguyên từ server.
+) POST : tạo thêm mới tài nguyên.
+) DELETE : xóa tài nguyên.
+) PUT : Cập nhật 1 tài nguyên nào đó.
+) PATCH : Cập nhật 1 phần của tài nguyên nào đó.

TRONG SPRING BOOT: ta dùng @RestController ở controller để xây dựng web service.

Cụ thể, chức năng của @RestController bao gồm:

+) Kết hợp @Controller và @ResponseBody: @RestController tự động thêm @ResponseBody
vào mọi phương thức trong controller. ĐIỀU NÀY CÓ NGHĨA LÀ KẾT QUẢ CỦA MỖI PHƯƠNG
THỨC SẼ ĐƯỢC CHUYỂN ĐỔI THÀNH ĐỊNH DẠNG JSON HOẶC XML VÀ TRẢ VỀ TRỰC TIẾP CHO
CLIENT, thay vì điều hướng đến một trang web cụ thể.

+) Giảm độ phức tạp của việc tạo RESTful web service: Bằng cách sử dụng @RestController, bạn có thể tạo ra các endpoint của API mà không cần sử dụng @ResponseBody cho từng phương thức riêng lẻ.

Ví dụ:

```
@RestController
@RequestMapping("/api")
public class MyRestController {

    @GetMapping("/hello")
    public String sayHello() {
        return "Hello, World!";
    }

    @GetMapping("/user/{id}")
    public ResponseEntity<User> getUser(@PathVariable Long id) {
        // Logic to retrieve user by ID

                try{
                        User user = userRepository.findById(id).orElse(null);

                        if (user != null) {
                                return new ResponseEntity<>(user, HttpStatus.OK);
```

=> ok thì trả về user đó và trạng thái. // 200

```
                } else {
                        return new ResponseEntity<>(HttpStatus.NOT_FOUND); => ko user null thì trả về trạng thái NOT_FOUND //404
                }
            }catch(Exception e){
                return new ResponseEntity<>(HttpStatus.BAD_REQUEST); //400
            }

    }
}
```

********* Bài 12 AJAX **********

1. Ajax
- Asynchronous JavaScript and XML là 1 nhóm các công nghệ phát triển web được sử dụng để tạo các ứng dụng giàu tính tương tác(bình luận, like,...), nhanh hơn, mượt hơn với sự giúp đỡ của XML, HTML, CSS và JavaScript.
-> vì nó sử dụng cơ chế bất đồng bộ để trao đổi 1 lượng dữ liệu nhỏ với server.
- Với Ajax, người dùng có thể tiếp tục sử dụng trang web trong khi chương trình trên client tạo request và lấy thông tin từ server.
- Ajax không phải công nghệ của jQuery, jQuery chỉ hỗ trợ viết Ajax dễ dàng hơn.
- Ta có thể kết hợ web service(api) với ajax.

2. Bất đồng bộ là gì? Cơ chế xử bất đồng bộ trong javascript.
(https://viblo.asia/p/co-che-bat-dong-bo-trong-javascript-jvElaO1zKkw)

3. Các thành phần cơ bản của Ajax.
- HTML&CSS: hiển thị dữ liệu
- Javascript
- DOM
- Đối tượng XMLHttpRequest: trao đổi dữ liệu 1 cách bất đồng bộ với server.
- XML hoặc JSON: định dạng dữ liệu truyền đi.

4. Ưu nhược điểm của AJAX.

- Nhược điểm:
+) Nó không thể lưu lịch sử duyệt web -> đo đó nút back sẽ mất tác dụng.
+) Không thể bookmark trang ajax.
+) Nếu trình duyệt ko hỗ trợ JS hoặc vô hiệu hóa JS -> ko dùng được AJAX.
+) Nếu dev ko kiểm thử hết các trường hợp thì dễ bị tấn cổng bởi các đoạn mã độc.

- Ưu điểm:
+) Cập nhật lại trang nếu có sự thay đổi dữ liệu -> giảm băng thông và thời gian load trang.
+) Việc dùng request bất đồng bộ thì giúp trải nghiệm người dùng cải thiện hơn.
+) Làm giảm các kết nối với server, do các mã js và css chỉ yêu cầu 1 lần.

******** BÀI 13: Spring i18n *******

1. Khái niệm và ý nghĩa là i18n(internationalization - quốc tế hóa)
- Là quá trình thiết kế 1 ứng dụng phần mềm đáp ứng được nhiều ngôn ngữ(anh, việt, nhật,..) khác nhau mà không cần thay đổi kỹ thuật.

2. Khái niệm L10N(Localization - địa phương hóa)
- Là quá trình điều chỉnh phần mềm đã được quốc tế hóa cho 1 ngôn ngữ hoặc 1 khu vực cụ thể bằng cách chỉ định ngôn ngữ hoặc khu vực sau khi dịch văn bản.
-> Sự kết hợp của L10N và I18N được gọi là G11N(Globalization)

3. Interceptor và LocaleResolver
- Interceptor: sẽ được đính vào trong 1 vòng đời của request.
Có khả năng xử lý tiền và hậu các request.
Được dùng phổ biến để internationlize.
- RecaleResolver: có thể phân giải message dựa trên thông tin về locale trong request hoặc từ session, cookie.

****** Bài 14: Authentication *******

1. Khái niệm về Authentication
Là 1 hành động nhằm thiết lập hoặc chứng thực 1 thông điệp hoặc đối tượng

nào đó là đáng tin cậy -> hệ thống sẽ tin vào những lời khai báo cũng như thông điệp từ đối tượng đó đưa ra.

2. Các cơ chế xác thực.

Cách 1: HTTP basic: xác thực thông qua username và password.

Cách 2: Cookies. Sau khi đăng nhập bằng username và password thành công, server sẽ trả về cho ta 1 cookie chứa sessionid. Vào lần đăng nhập sau ta chỉ cần dùng cookie là đăng nhập được.

Cách 3: JWT(Json web token): là tiêu chuẩn mở(RFC-7519) định nghĩa cách thức truyền tin an toàn giữa các thành viên bằng 1 JSON.
Xác thực và đánh dấu là tin cậy bởi "chữ ký" của nó.
Phần chữ kỹ của JWT sẽ được mã hóa bằng HMAC và RSA.
- JWT có 3 thành phần chính:
+) Tiêu đề: chứa loại mã thông báo và thuật toán băm.
+) Tải trọng: chứa xác nhận quyền sở hữu(thông tin cá nhân không nhạy cảm : username, địa chỉ,...), vì nếu để thông tin quan trọng vào đây thì hacker có thể giải mã ra được.
+) Chữ ký

===> NÊN TÌM HIỂU VỀ JWT

Cách 4: Signature - chữ ký số.
Sử dụng cặp khóa công khai - bảo mật và qua đó có thể ký các văn bản điện tử cũng như trao đổi thông tin mật.

Cách 5: OTP - One Time Password: khi chuyển tiền ngân hàng.

Cách 6: Oauth 2: Xác thực thông qua các ứng dụng khác: google, facebook,...

Cách 7: Xác thực thông qua mã OR.

3. Spring Security và các thành phần của Spring Security.

- Spring Security là 1 dự án của Spring framework, cung cấp các dịch vụ bảo mật toàn diện cho các ứng dụng Java EE.
- Gồm có Authentication và Authorization.
- Authentication là quá trình thiết lập 1 đối tượng pricipal(là 1 người, 1 thiết bị hoặc 1 ứng dụng khác,...) mà có thể truy cập được vào ứng dụng của chúng ta.
- Authorization: là tiến trình quyết định xem liệu rằng 1 pricipal có thể thực hiện 1 hành động gì đó trong ứng dụng hay không.

- Các thành phần trong ứng dụng Spring Security:
+) Security Context: là 1 interface, lưu trữ tất cả các chi tiết liên quan đến bảo mật. Khi kích hoạt Spring Security thì nó cũng kích hoạt Security Context.
+) SecurityContextHolder: để truy cập vào SecurityContext. Nó gồm chi tiết của pricipal đang tương tác với ứng dụng.
+) UserDetails: là 1 interface đại diện cho 1 principal gồm các phương thức:
getAuthorities() -> trả về danh sách các quyền của người dùng.
getPassword(), getUsername(),
isAccountNonExpired(), isAcountNonLocked(),
isCredentialsNonExpired(), isEnabled()

********** BÀI 15: Authorization **********

1. Khái niệm Authorization
- Hay còn gọi là Access - control, là tiến trình quyết định xem 1 pricipal có được phép thực hiện 1 hành động trong ứng dụng hay không.
Trước khi diễn ra authorization thì pricipal phải được thiết lập bởi authentication.

2. Cách thức tấn công CSRF(Cross-Site Request Forgery)
- Là kỹ thuật tấn công bằng cách sd quyền chứng thực của người dùng đối với 1 website. CSRF là kỹ thuật tấn công vào người dùng, dựa vào đó, hacker có thể thực thi những thao tác phải yêu cầu chứng thực.

3. Cách thức tấn công XSS
- Tiêm 1 đoạn mã độc vào trong trang web. Đây là 1 trong các cách tấn công phổ biến nhất hiện tại.
Khi có cuộc tấn công XSS trên trang web thì mã độc sẽ thực thi trong trình

duyệt của người dùng -> đánh cặp thông tin(cookie) của người dùng, để xem thông tin hoặc giả mạo người dùng.