**CÂU HỎI AUDIT MODULE 2**

**BOOTCAMP PREPARATION 2.0**

|  |  |  |
| --- | --- | --- |
| **STT** | | **Câu hỏi** |
| 1 | **Trình bày cách Spring MVC xử lý request?**   * Khi có 1 request đi vào Spring MVC thì:   + Front Controller (dispatcher Servlet) sẽ đón nhận và sử dụng cơ chế URL Handler Mapping để ánh xạ đến Controller tương ứng.   + Controller xử lý request, nếu cần sẽ xuống Model để xử lý nghiệp vụ và lấy dữ liệu dưới DB. Kết quả sẽ được gửi lại cho Front Controller.   + Front Controller sẽ render dữ liệu lên View bởi cơ chế ViewResolver   + Kết quả sẽ được gửi lại cho Front Controller và trả về cho người dùng. | | |
| 2 | **Trình bày cơ chế Dependency Injection (DI) trong Spring?**   * Dependency Injection (tiêm phụ thuộc) là cơ chế giảm sự phụ thuộc giữa 2 class, tức class cấp cao hơn không phụ thuộc trực tiếp vào class cấp thấp hơn, mà thông qua 1 abstraction. | | |
| 3 | **Có bao nhiêu cách để thực hiện Dependency Injection? Trình bày?**   * Các cách DI: Sử dụng @Autowired   + Interface (field)   + Constructor   + Setter | | |
| 4 | **Framework là gì ? Framework khác Library chỗ nào ?**   * Framework là 1 ứng dụng trựu tượng có sẵn các bộ khung chương trình và flow code để tạo ra ứng dụng 1 cách nhanh chóng.  |  |  | | --- | --- | | Library | Framework | | * Chỉ cung cấp các method tiện ích hay các class để sử dụng trong quá trình xây dựng ứng dụng | * Là một khung chương trình, người dùng bổ sung code và tuân theo quy tắc để tạo ra ứng dụng | | * Chúng ta có thể sử dụng các functions của Library một cách trực tiếp mà không cần thay đổi cấu trúc code của dự án. | * Sử dụng Framework bạn phải thay đổi cấu trúc code của dự án (project’s structure) theo các quy tắc của framework đó để có thể sử dụng được các functions mà framework đó cung cấp. | | * Ứng dụng nắm quyền điều khiển | * Có cơ chế IOC (inversion of control) => nguyên lý thiết kế mà Framework nắm quyền điều khiển. |  |  |  | | --- | --- | |  | * Các đối tượng sẽ do IOC Container tạo, láp ráp, quản lý, không còn do ứng dụng quản lý như trước | | | |
| 5 | **Spring Framework là gì?**   * Spring Framework là một thư viện mã nguồn mở, sử dụng để phát triển các ứng dụng doanh nghiệp. Nó bao gồm khoảng hai mươi module khác nhau. | | |
| 6 | **Lợi ích của Spring Framework ?**   * Spring là một framework giúp các nhà phát triển xây dựng những hệ thống và ứng dụng chạy trên JVM một cách đơn giản, tiện gọn, nhanh chóng và mềm dẻo. | | |
| 7 | **Nguyên lý đảo ngược quyền điều khiển (Inversion of Control) là gì ?**   * Với mô hình IoC thì class A không tự khởi tạo cũng không chịu trách nhiệm quản lý Class B, Class C. Nó nhận được hai dependency này từ bên ngoài thông qua một cơ chế nào đó (như bằng setter, bằng tham số hàm tạo lớp A, bằng gán thuộc tính ...) | | |
| 8 | **Bean là gì?**   * Bean: là các đối tượng được tạo, lắp ráp và chịu sự quản lý bởi Spring Container (IOC Container). | | |
| 9 | **Trong Spring có bao nhiêu 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. Khi sử dụng scope này cần chắc chắn rằng các bean không có các biến/thuộc tính được share. * Prototype: Một 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, một 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.   Trong 5 scope trên thì 3 scope cuối chỉ dùng trong ứng dụng web. | | |
| 10 | **@Autowire là gì?**   * @Autowired là Annotation đánh dấu cho Spring biết rằng sẽ tự động inject bean tương ứng vào vị trí được đánh dấu. | | |
| 11 | **@Component có ý nghĩa gì?**   * @Component được sử dụng để định nghĩa một khuôn mẫu chung cho toàn bộ dự án. * Class được đánh dấu với annotation @Component thì sẽ được tạo thành 1 bean. Khi một class được đánh dấu với annotation với @Component thì khi dự án Spring được chạy nó sẽ quét qua và tạo 1 bean cho class đó. * @Service, @Repository, @Controller là một trường hợp đặc biệt của @Component và chúng được sử dụng với những mục đích khác nhau. | | |
| 12 | **Trình bày 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. | | |
| 13 | **Trình bày ý nghĩa của ModelAndView Interface?**   * ModelAndView là 1 interface, giúp truyền dữ liệu từ controller đến View, cần khai báo đối tượng khi sử dụng, cần phải khai báo thêm tên View. | | |
| 14 | **Trình bày ý nghĩa của ModelMap?**   * ModelMap là 1 class, giúp truyền dữ liệu từ controller đến View, được khai báo như là 1 tham số ở method trong Controller, có thêm 1 số method của Map như put(),putAll()…. | | |
| 15 | **Trình bày ý nghĩa của ViewResolver Interface?**   * ViewResolver là cơ chế xử lý tầng View trong Spring MVC * ViewResolver giúp ánh xạ tên View sang đối tượng View tương ứng. | | |
| 16 | **Phân biệt POST với PUT thường sử dụng để làm gì?** | | |
| 17 | **@RequestMapping làm gì?**   * @RequestMapping là 1 Annotation dùng để ánh xạ các request đến Method trong Controller tương ứng. | | |
| 18 | **Trình bày cơ chế Data Binding trong Spring?**   * Data Binding là kỷ thuật liên kết dữ liệu đầu vào (đầu ra) với các Object method, thông qua cơ chế Data binder. * Với 3 nhiệm vụ sau:   + Property Editor: Liên kết với các thuộc tính   + Validator: Kiểm tra dữ liệu có hợp lệ hay không   + Formater / Convester: Chuyển đổi dữ liệu | | |
| 19 | **Thuộc tính consumes trong các Request Mapping là gì ?** | | |
| 20 | **Thymeleaf là gì?**   * Thymeleaf laf Java Template Engine (bộ công cụ xử lý tầng view) dùng để xử lý và tạo HTML, CSS, JS, XML, TEX | | |
| 21 | **Sử dụng lặp trong Thymeleaf như thế nào? Sử dụng điều kiện trong Thymeleaf như thế nào?**   * Thymeleaf cung cấp cho ta vòng lặp 'each', và bạn có thể sử dụng nó thông qua thuộc tính (attribue) th:each. Đây là vòng lặp duy nhất được hỗ trợ trong Thymeleaf. * Vòng lặp này chấp nhận một vài loại dữ liệu như:   + Các đối tượng thực hiện (implements) interface java.util.Iterable.   + Các đối tượng thực hiện (implements) interface java.util.Map.   + Các mảng (Arrays) * Th:if, th:unless, th:switch, th:case | | |
| 22 | **Formatter là gì ? Converter là gì ?**   * 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. | | |
| 23 | **ORM là gì ?**   * ORM (Object-Relational Mapping) là kỹ thuật liên kết giữa các đối tượng trong lập trình với các đối tượng trong CSDL. Khi sử dụng lập trình hướng đối tượng và CSDL quan hệ, ORM sẽ giúp ánh xạ các Class đến các Table và các Object đến các Record trong các Table. | | |
| 24 | **JPA là gì? Spring Data JPA là gì ?**   * JPA là bộ đặc tả của Java bao gồm các interface được thiết kế từ ý tưởng ORM. * Spring Data JPA là 1 module của Spring hỗ trợ triển khai JPA dễ dàng hơn ở tầng Repository. | | |
| 25 | **Hibernate là gì ? Hibernate làm gì với database?**   * Hibernate là 1 ORM Framework để triển khai JPA | | |
| 26 | **Liệt kê một số annotation của hibernate ?**   * **@Entity** được sử dụng để chú thích một class là một Entity * **@Table** cho phép chú thích tên bảng thông qua thuộc tính name (thuộc tính này không bắt buộc). * **@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. * **@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). * **@OneToOne** mô tả quan hệ 1-1 (Một – Một). * **@ManyToMany** mô tả quan hệ N-N (Nhiều – Nhiều). | | |
| 27 | **Entity là gì?**   * Entity là đối tượng đại diện cho dự liệu trong ứng dụng. * Entity thường là lớp đơn giản, chỉ mô tả dữ liệu, không có xử lý logic nghiệp vụ. * Mỗi **Entity** được **mapping** với 1 table trong CSDL (**mapping** 1-1). * Mỗi **Entity** cần phải tuân thủ:    + Được gắn **annotation** **@Entity**, **@Id**.   + Có **Constructor** phải là public không tham số.   + Các **class** hay **method** không được khai báo final.   + Thuộc tính có **access modifier** là **private**, **protected**, hoặc ở mức **package private** (tức là không phải public). | | |
| 28 | **Persistence Context & Entity Manager là gì?**   * **Persistence Context**: là 1 tập các thể hiện của **Entity**, được dùng để quản lý **Entity** trong 1 kho dữ liệu. * **Entity Manager:** là 1 **interface** cung cấp các **method** để thao tác với **entity**:    + **Persist():** Thêm 1 entity.   + **Find():** Tìm kiếm.   + **Merge():** Cập nhật.   + **Remove():** Xóa. | | |
| 29 | **Câu lệnh truy vấn động là gì? Câu lệnh truy vấn tĩnh là gì?**   * **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. | | |
| 30 | **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. | | |
| 31 | **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. | | |
| 32 | **SessionFactory là gì?**  Session Factory là một interface giúp tạo ra session kết nối đến database bằng cách đọc các cấu hình trong Hibernate configuration. | | |
| 33 | **Validation dữ liệu là gì?**  Validation là thao tác ràng buộc dữ liệu đầu vào để đảm bảo tính đúng đắn và chọn vẹn của dữ liệu | | |
| 34 | **Trình bày cách triển khai validate dữ liệu trong Spring?**  **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**. | | |
| 35 | **Binding Result là gì?**   * BindingResult là nơi Spring chứa kết quả của việc xác thực dữ liệu. | | |
| 36 | **AOP là gì? AOP gồm những yếu tố nào?**   * AOP (Aspect Oriented Programming): là 1 kỹ thuật lập trình hướng khía cạnh, nhằm tách các xử lý lặp đi lặp lại, không liên quan đến nhiệm vụ chính thành 1 module riêng lẻ, 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).     * Các yếu tố:   + Aspect: là 1 khối độc lập với OOP, dùng để bổ trợ cho OOP   + Advice: là các hành động mà Aspect dùng để quan sát OOP   + Pointcut: là tập hợp các điểm có khả năng xảy ra advice   + Join point: là các điểm xảy ra advice. | | |
| 37 | **Các loại Advice?**   * Before: Được thực thi trước khi method được thực thi * After: được thực thi sau khi method được thực thi * After-returning: được thực thi sau khi method được thực thi thành công * After-throwing: được thực thi sau khi method được thực thi không thành công, nghĩa là có ngoại lệ được throw * Around: Được thực thi cả trước và sau khi method được thực thi | | |
| 38 | **JoinPoint là gì?**    -Join Point là là 1 vị trí nào đó trong luồng thực thi của chương trình mà advice sẽ đc đan (weave) vào chương trình chính. | | |
| 39 | **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. | | |
| 40 | **Xử lý ngoại lệ trong Spring Web MVC như thế nào?**  + 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**). | | |
| 41 | **Cookie là gì?**   * Cookie là 1 đoạn văn bản ghi thông tin được Server lưu lại trong ổ nhớ của người dùng. Những thông tin đó sẽ được gửi lại cho Server khi người dùng request lại | | |
| 42 | **Phân biệt Session và Cookie**   |  |  | | --- | --- | | **Session** | **Cookie** | | Kiểu dữ liệu bất kỳ | Kiểu dữ liệu dạng text | | Thời gian sống:   * Tắt trình duyệt * Tắt server * Hết thời gian timeOut | Thời gian sống:   * Cố tính xóa trên trình duyệt * Hết thời gian sống (setMaxAge) | | Lưu:   * Lưu ở server (Đồng thời tạo cookie tên là JSESSION ID để phân biệt với các user khác) | Lưu:   * Client | | Bảo mật:   * Dữ liệu session không dễ dàng sửa đổi vì chúng được lưu trữ ở phía máy chủ (bảo mật hơn). | Bảo mật:   * Dữ liệu cookie dễ dàng sửa đổi khi chúng được lưu trữ ở phía người dùng (kém bảo mật). | | | |
| 43 | **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.  (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**. | | |
| 44 | **Web Service là gì? Lấy ví dụ về web service**   * Web service là 1 thành phần bao gồm tiêu chuẩn và giao thức để giúp trao đổi dữ liệu giữa các ứng dựng với nhau. | | |
| 45 | **Phân biệt Web Service và Web truyền thông?** | | |
| 46 | **SOAP là gì? RESTful là gì?** | | |
| 47 | **jQuery là gì? Ajax là gì?** | | |
| 48 | **Nêu ví dụ về cách bạn triển khai Ajax trong dự án của bạn?** | | |
| 49 | **@RequestHeader và @ResponseHeader có ý nghĩa gì?** | | |
| 50 | **I18n và L10n là gì?** | | |
| 51 | **Nêu cách triển khai I18N và I10N trong dự án Spring?** | | |
| 52 | **Interceptor là gì?** | | |
| 53 | **Bảo mật là gì? Cơ chế bảo mật trong Spring như thế nào?** | | |
| 54 | **Authentication là gì ? Authorization là gì ?** | | |
| 55 | **Liệt kê các cơ chế xác thực** | | |
| 56 | **CSRF là gì?** | | |
| 57 | **CORS là gì** | | |
| 58 | **Spring Boot là gì?** | | |
| 59 | **Sự khác nhau giữa SpringBoot và SpringMVC là gì?** | | |
| 60 | **Cấu hình cho Spring Boot Tìm kiếm các Bean ở nhiều package khác nhau bằng cách nào?** | | |
| 61 | **Trình bày Spring Security? @EnableWebSecurity làm gì?** | | |
| 62 | **Trình bày Spring Boot JPA? Cài đặt?** | | |
| 63 | **Để sử dụng interface JpaRepository cần cung cấp những thông tin nào?** | | |
| 64 | **Bạn có biết annotation @ConfigurationProperties? Trình bày?** | | |