

Mémento Annotations

Auteur: François-Xavier COTE (fxcote@clelia.fr)

Révision: 1.20.6

JPA, Hibernate

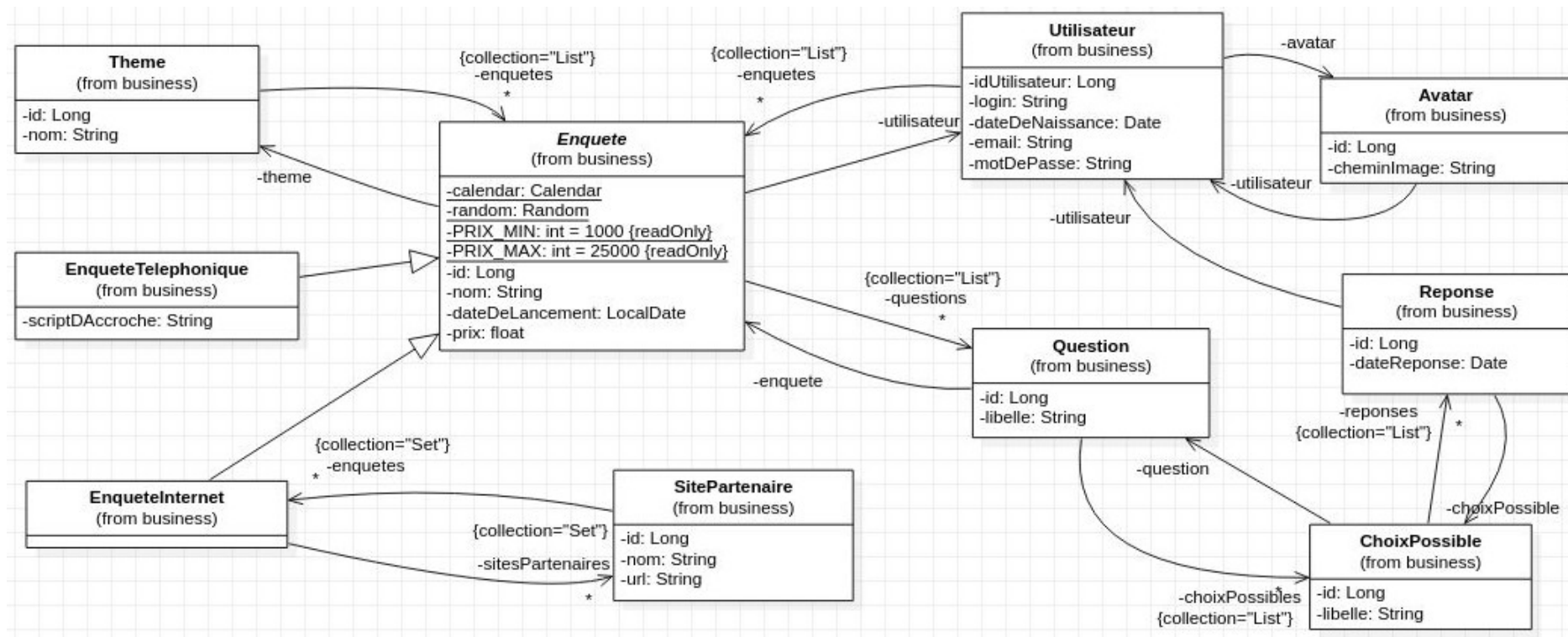
Validation

Spring

Lombok

Jackson

Pour ce mémento on considère une application Web de gestion d'enquêtes dont voici le diagramme de classes métier :

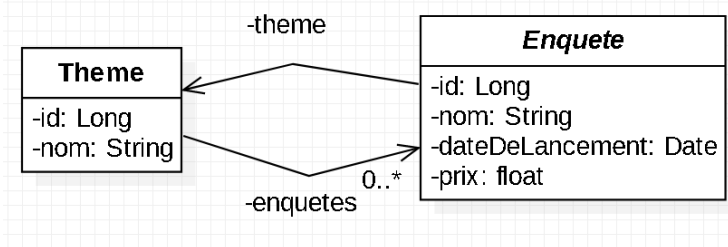


Contexte	Annotation	Description
<p>JPA, Hibernate ORM</p> <p>(à utiliser dans les classes métier en important les annotations du package <code>javax.persistence</code>)</p>	@Entity	<p>Définit une classe POJO (Plain Old Java Object) persistante</p> <pre>@Entity public class Theme { @Id private Long id; private String nom; }</pre> <p>Face à cette classe métier Theme, Hibernate va créer une table theme.</p> <div data-bbox="927 742 1146 911"> <p>Theme</p> <p>-id: Long -nom: String</p> </div> <p style="text-align: center;">↔</p> <div data-bbox="1655 684 2036 970"> <p>theme</p> <p>💡 id BIGINT(20) 💎 nom VARCHAR(255)</p> <p>Indexes ▶</p> </div> <p>Chaque objet de type Theme aura une correspondance avec un enregistrement de la table Theme.</p> <p>En oubliant cette annotation sur une classe métier on obtient une exception <code>IllegalArgumentException :Not a managed type: class fr.clelia.fx.enquetes.business.Theme</code></p> <p>Les entités JPA ont un cycle de vie présenté sur le diagramme suivant :</p>

Contexte	Annotation	Description
		<pre> graph TD dne[does not exist] -- "POJO construction Object obj = new Object();" --> new[new] new -- "1. EntityManager.persist(obj) 2. @PrePersist 3. Database insert 4. @PostPersist" --> managed[managed] managed -- "1. EntityManager.remove(obj) 2. @PreRemove 3. Pending removal in database" --> removed[removed] removed -- "1. Database remove 2. @PostRemove" --> dne managed -- "serialized to another tier" --> detached[detached] detached -- "EntityManager.merge(obj)" --> managed </pre> <p>Source : Oracle</p>
	@Table	<p>Définit le nom de la table associée à la classe POJO persistante</p> <pre> @Entity @Table(name="utilisateurs") public class Utilisateur {} </pre>
	@Id	<p>Précise que l'attribut annoté va donner lieu à une colonne qui sera la clé primaire de la table associée</p> <pre> @Id private Long id; </pre>
	@GeneratedValue	<p>Demande à JPA de choisir une valeur unique pour chaque enregistrement. Il est possible de préciser</p>

Contexte	Annotation	Description
		<p>une stratégie de génération de l'id grâce à un attribut strategy.</p> <pre>@Id @GeneratedValue(strategy=GenerationType.IDENTITY) private Long id;</pre> <p>strategy=GenerationType.IDENTITY utilise une identité propre au SGBD (auto_increment de MySQL, séquence d'Oracle) strategy=GenerationType.AUTO: Hibernate se charge de la génération en créant une séquence unique au schéma strategy=GenerationType.SEQUENCE: La génération de la clé primaire se base sur une séquence strategy=GenerationType.TABLE: La génération de la clé primaire utilise des valeurs stockées dans une table qui se nomme par défaut hibernate_sequences</p>
	@SequenceGenerator	<p>Déclare la séquence que JPA devra utiliser pour attribuer un id à une nouvelle instance</p> <pre>@Id @GeneratedValue(strategy=GenerationType.SEQUENCE, generator="enquete_sequence") @SequenceGenerator(name="enquete_sequence") private Long id;</pre>
	@TableGenerator	<p>Déclare la table que JPA devra utiliser pour attribuer un id à une nouvelle instance</p> <pre>@Id @GeneratedValue(strategy=GenerationType.TABLE, generator="enquete_id_broker") @TableGenerator(name="enquete_id_broker", valueColumnName="next_id") private Long id;</pre>
	@Column	<p>Détaille la manière de créer la colonne en base Les principaux attributs de l'annotation @Column sont:</p> <ul style="list-style-type: none"> - name: nom de la colonne dans la table associée - nullable: permet d'autoriser ou d'interdire null dans la colonne, pour autoriser null: nullable=true - unique: ajoute une contrainte d'unicité sur la colonne - length: précise la longueur de la colonne (pour les attributs de type String), par défaut: 255

Contexte	Annotation	Description
		<pre>@Column(unique=true, nullable=false, length=150) private String nom;</pre>
	@Basic	<p>Déclare de la manière la plus simple une correspondance entre un attribut Java et une colonne éponyme dans la table associée. Cette annotation est rarement utilisée aujourd'hui. C'est la pièce de musée de ce memento.</p> <pre>@Basic private String libelle;</pre>
	@Lob	<p>Indique que la colonne doit contenir un texte long (permet aussi de stocker des données binaires)</p> <pre>@Lob private String scriptDAccroche;</pre>
	@Temporal	<p>@Temporal(TemporalType.DATE) : crée dans la table une colonne de type date @Temporal(TemporalType.TIME) : crée dans la table une colonne de type time @Temporal(TemporalType.TIMESTAMP) : crée dans la table une colonne de type datetime</p> <p>Important :</p> <ul style="list-style-type: none"> - il est conseillé d'utiliser les classes LocalDate, LocalTime et LocalDateTime. En utilisant ces classes l'annotation Temporal n'a pas d'utilité - il ne faut pas utiliser la classe java.sql.Date dans les classes métier <p>Sans préciser @Temporal, Hibernate ajoute une colonne datetime pour un attribut de type java.util.Date</p> <pre>@Temporal(TemporalType.DATE) private Date dateDeLancement;</pre>

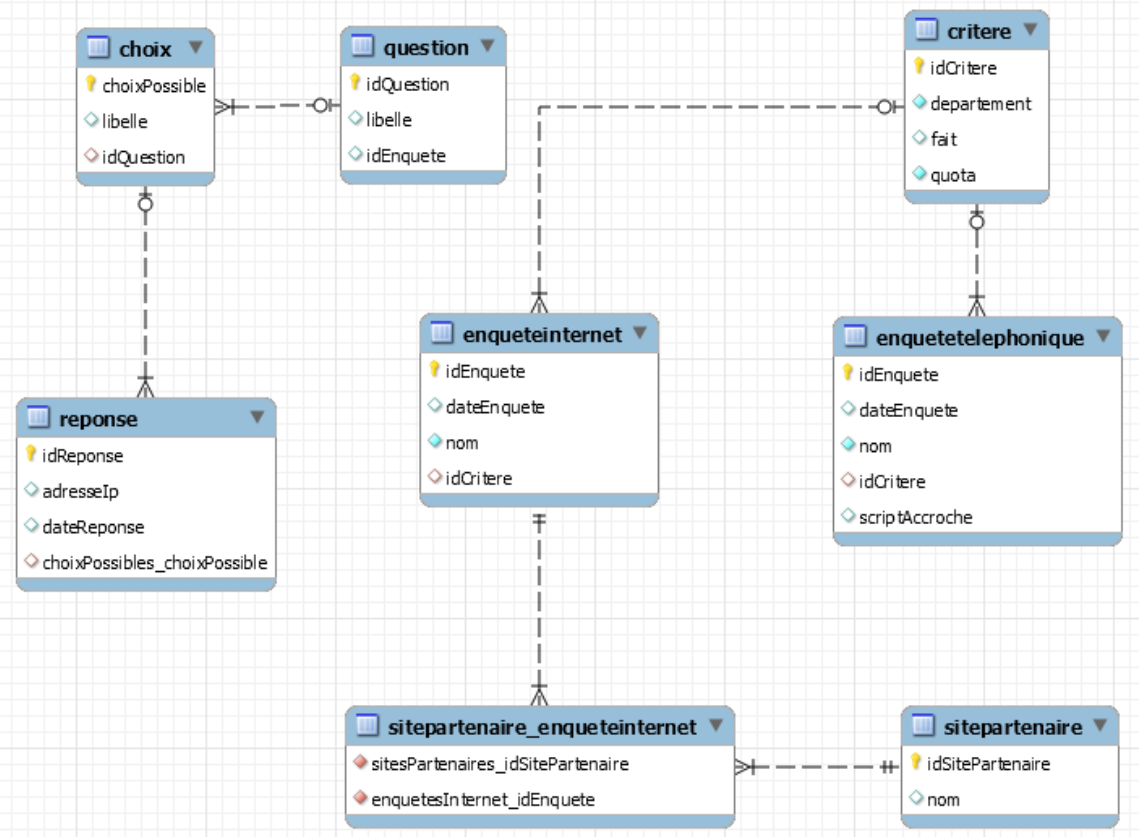
Contexte	Annotation	Description
	@OneToOne	<p>Indique une bijection avec l'autre classe. Entre les deux classes il y a deux associations dirigées (allant dans des directions opposées) avec une multiplicité 0..1 pour chaque association.</p> <p>Exemple: un utilisateur a un seul avatar. Un avatar correspond à un seul utilisateur</p> <pre>@OneToOne private Avatar avatar;</pre>
	@ManyToOne	<p>Indique que plusieurs objets de la classe vont être associés à un seul et même objet de l'autre classe</p> <p>Dans notre exemple plusieurs enquêtes font référence à un même thème.</p>  <pre>@Entity public abstract class Enquete { @Id @GeneratedValue(strategy=GenerationType.IDENTITY) private Long id; @Column(unique=true, nullable=false, length=150) private String nom; private Date dateDeLancement; private float prix; @ManyToOne private Theme theme; }</pre> <p>En base, dans la table Enquete, une colonne theme_id sera ajoutée, c'est une clé étrangère vers la table Theme.</p>

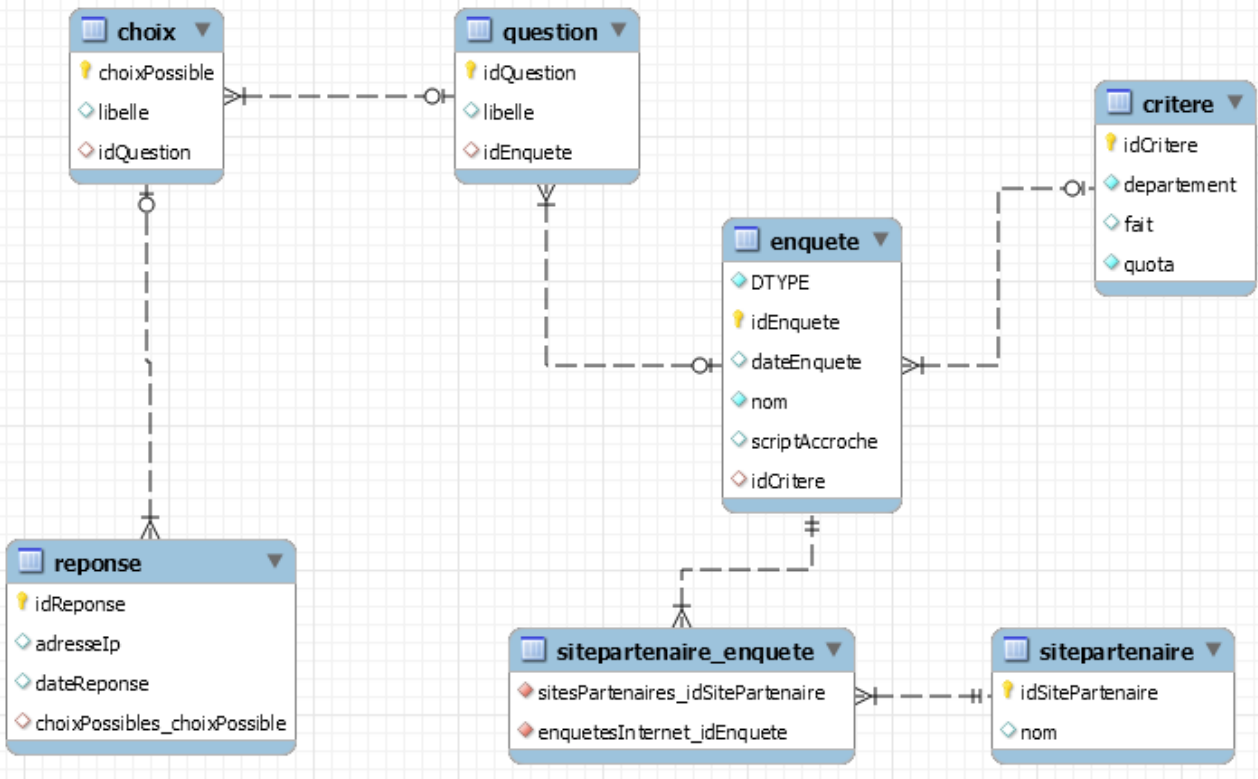
Contexte	Annotation	Description
		<p>En oubliant l'annotation @ManyToOne sur un attribut métier, on obtient l'erreur suivante :</p> <p>Caused by: org.hibernate.MappingException: Could not determine type for: fr.clelia.fx.enquetes.business.Theme, at table: Enquete, for columns: [org.hibernate.mapping.Column(theme)]</p>
	@OneToMany	<p>Indique qu'un objet de la classe va être associé à plusieurs objets de l'autre classe. Pour l'attribut mappedBy, on précise le nom de l'objet dans l'autre classe.</p> <p>Dans l'exemple de la page précédente: à un thème correspond une liste d'enquêtes. Une enquête est associée à un thème, il y a donc un attribut nommé theme dans la classe Enquete. C'est le nom de cet attribut Java que l'on écrit dans l'attribut mappedBy de l'annotation OneToMany:</p> <pre>@OneToMany(mappedBy="theme") private List<Enquete> enquetes;</pre> <p>En règle générale, dès qu'une classe comporte une liste d'objets métier, elle sera annotée @OneToMany. Idem pour un objet de type Set.</p> <p>Bien penser à préciser le type de récupération en utilisant l'attribut fetch, par défaut le fetch est FetchType.LAZY.</p> <p>Exemple: si l'on souhaite récupérer d'emblée toutes les enquêtes d'un thème, on choisira le type de fetch EAGER:</p> <pre>@OneToMany(mappedBy="theme", fetch=FetchType.EAGER) private List<Enquete> enquetes;</pre> <p>En essayant de récupérer les enquêtes sans avoir au préalable utilisé le fetch type EAGER on obtient l'exception suivante:</p> <p>Caused by: org.hibernate.LazyInitializationException: failed to lazily initialize a collection of role: fr.clelia.fx.enquetes.business.Theme.enquetes, could not initialize proxy - no Session</p>

Contexte	Annotation	Description
		<p>A noter: on ne peut pas avoir plus d'un FetchType à EAGER par classe. En utilisant deux fois le FetchType EAGER dans une même classe on obtient l'exception suivante: Caused by: org.hibernate.loader.MultipleBagFetchException: cannot simultaneously fetch multiple bags Notion de cascade :</p> <p>- On ajoute en mémoire un objet de type Enquete et on lui associe un ensemble de questions :</p> <pre>Enquete enquete1 = new Enquete("Enquete 1", theme1); enquete1.getQuestions().add(new Question("Question 1", enquete1)); enquete1.getQuestions().add(new Question("Question 2", enquete1));</pre> <p>Pour qu'Hibernate enregistre l'enquete en base ainsi que toutes ses questions, il faut définir la liste de questions comme suit :</p> <pre>@OneToMany(mappedBy="enquete", cascade = CascadeType.PERSIST) private List<Question> questions;</pre> <p>- Pour que les objets questions soit mis à jour en base (lorsqu'ils évoluent en mémoire), il faut utiliser la valeur CascadeType.MERGE</p> <pre>@OneToMany(mappedBy="enquete", cascade = CascadeType.MERGE) private List<Question> questions;</pre> <p>- Pour effacer automatiquement tous les questions associées à une enquête qui doit être effacée, l'attribut cascade avec la valeur CascadeType.REMOVE doit être précisé:</p> <pre>@OneToMany(mappedBy="enquete", cascade = CascadeType.REMOVE) private List<Question> enquetes;</pre> <p>- Pour effacer automatiquement tous les questions associées à une enquête qui doit être effacée et les détacher de l'entity manager, l'attribut cascade avec la valeur CascadeType.DETACH doit être précisé:</p>

Contexte	Annotation	Description
		<pre>@OneToMany(mappedBy="enquete", cascade = CascadeType.DETACH) private List<Question> enquetes;</pre> <p>- Pour garantir toutes les opérations présentées ci-dessus, l'attribut cascade doit avoir la valeur CascadeType.ALL</p> <pre>@OneToMany(mappedBy="enquete", cascade = CascadeType.ALL) private List<Question> questions;</pre>
	@ManyToMany	<p>Indique une liste dans chaque classe. Entre les deux classes il y a deux associations dirigées (allant dans des directions opposées) avec une multiplicité 0..* pour chaque association. Exemple: à une enquête correspond plusieurs sites internet. Un site internet est utilisé par plusieurs enquête.</p> <p>En base une table de mapping sera créée.</p> <p>L'attribut mappedBy doit figurer uniquement sur une des deux annotations sinon l'exception AnnotationException est levée avec le message suivant: Illegal use of mappedBy on both sides of the relationship</p> <p>On choisit de mettre l'attribut mappedBy sur la classe satellite, dans ce projet la classe centrale est Enquete, les classes autour sont les classes satellites.</p> <p>Dans l'exemple ci-dessous, l'enregistrement d'un objet de type Enquete "alimentera" bien la table de mapping car dans la classe SitePartenaire on a le code suivant:</p> <pre>@ManyToMany(mappedBy="sitesPartenaires") private Set<EnqueteInternet> enquetes;</pre> <p>Hibernate va remplir la liste sitesPartenaires avec ce qui a été choisi dans la liste multiple (select)</p>

Contexte	Annotation	Description
		<h2>Enquete Internet</h2> <div> <input type="text" value="Nom"/> <input type="text" value="22/05/2019"/> <input type="text" value="2000.0"/> <div> Veuillez sélectionner un theme ▼ </div> <div> Veuillez sélectionner un ou plusieurs sites partenaires www.clelia.fr www.lemonde.fr </div> <div> <input type="button" value="Enregistrer"/> <input type="button" value="Annuler"/> </div> </div>
	@JoinColumn	Redéfinit la colonne sur laquelle il existe une contrainte de clé étrangère <pre>@ManyToOne @JoinColumn(name = "idEnquete") private Enquete enquete;</pre>
	@JoinTable	Renomme la table de mapping (créée par un ManyToMany) <pre>@ManyToMany @JoinTable(name = "enquetesInternet_SitesPartenaires") private List<SitePartenaire> sitesPartenaires;</pre>
	@Inheritance	La classe Enquete est la classe mère des classes EnqueteTelephonique et EnqueteInternet. Cette annotation permet de préciser la stratégie dans la manière de créer les tables liées à la notion d'héritage <ul style="list-style-type: none"> strategy=InheritanceType.TABLE_PER_CLASS : duplique les données pour éviter les opérations de jointure

Contexte	Annotation	Description
		 <pre> classDiagram class choix { choixPossible libelle idQuestion } class question { idQuestion libelle idEnquete } class critere { idCritere departement fait quota } class reponse { idReponse adresseIp dateReponse choixPossibles_choixPossible } class enqueteinternet { idEnquete dateEnquete nom idCritere } class enquetetelephonique { idEnquete dateEnquete nom idCritere scriptAccroche } class sitepartenaire_enqueteinternet { sitesPartenaires_idSitePartenaire enquetesInternet_idEnquete } class sitepartenaire { idSitePartenaire nom } choix "1" -- "0..1" question question "1" -- "0..1" critere reponse "1" -- "0..1" choix enqueteinternet "1" -- "0..1" question enquetetelephonique "1" -- "0..1" critere sitepartenaire_enqueteinternet "1" -- "0..1" enqueteinternet sitepartenaire_enqueteinternet "1" -- "0..1" sitepartenaire </pre> <p>NB : en choisissant cette stratégie d'héritage, la stratégie de génération de l'id dans la classe mère Enquete ne peut pas être IDENTITY. Il faudra choisir entre AUTO, SEQUENCE et TABLE.</p> <ul style="list-style-type: none"> strategy=InheritanceType.SINGLE_TABLE : classe mère et classes filles sont représentées par une table unique.

Contexte	Annotation	Description
		 <p>La colonne DTYPE contiendra en toutes lettres le nom de la classe fille dont l'objet est une instance. Dans notre exemple : EnqueteTelephonique ou EnqueteInternet</p> <p>La table Enquete va contenir des enregistrements correspondants à des enquêtes téléphoniques et des enregistrements correspondants à des enquêtes Internet</p> <p>Dans la table représentant la classe mère, on trouvera les attributs communs à toutes les classes filles. Dans la table représentant la classe fille, on trouvera les informations propres à la classe fille.</p>

Contexte	Annotation	Description
		<ul style="list-style-type: none"> strategy=InheritanceType.JOINED : classe mère et classes filles sont représentées chacune par une table. <p>Si la stratégie n'est pas précisée, la stratégie SINGLE_TABLE est utilisée. L'annotation @Inheritance se place sur la classe mère.</p> <p>@Entity</p>

Contexte	Annotation	Description
		<pre>@Inheritance(strategy=InheritanceType.SINGLE_TABLE) public abstract class Enquete {}</pre> <p>La classe fille EnqueteTelephonique est déclare ainsi:</p> <pre>@Entity public class EnqueteTelephonique extends Enquete {}</pre>
	@DiscriminatorValue	Cette annotation sert uniquement dans les classes filles et avec la stratégie d'héritage SINGLE_TABLE. Elle permet de redéfinir la valeur qui sera ajoutée dans la colonne discriminante (par défaut nommée DTYPE)
	@DiscriminatorColumn	<p>Définit le nom de la colonne discriminante dans la table représentant la classe mère. En l'absence de cette annotation, la colonne discriminante se nomme DTYPE. Cette annotation sert uniquement lorsque la stratégie d'héritage est SINGLE_TABLE.</p> <pre>@Entity @Inheritance(strategy=InheritanceType.SINGLE_TABLE) @DiscriminatorColumn(name="TypeEnquete") public abstract class Enquete {}</pre>
	@NamedQuery	<p>Déclare une requête HQL qui sera accessible par toutes les classes du projet</p> <pre>@Entity @NamedQuery(name="Joueur.findAll", query="SELECT j FROM Joueur j") public class Joueur {}</pre>
	@Transient	<p>Permet de définir un attribut dans la classe sans qu'une colonne soit ajoutée dans la table associée</p> <pre>@Transient private float prixTTC;</pre>
	@Fetch	<p>Demande à Hibernate d'alimenter automatiquement les listes d'objets. Il existe trois modes de récupération : JOIN, SELECT et SUBSELECT</p> <pre>@Fetch(value = FetchType.SELECT) ManyToMany private List<Question> questions;</pre>

Contexte	Annotation	Description
Validation <i>(à utiliser dans les classes métier en important les annotations du package javax.validation.constraints ou org.hibernate.validator.constraints)</i>	@NotBlank	Garantit que la valeur de l'attribut (de type String) ne contient pas une chaîne vide <pre>@NotBlank(message="Merci de donner un nom à l'enquête") @Column(unique = true, nullable = false, length = 150) private String nom;</pre>
	@NotNull	Garantit que la valeur de l'attribut contient bien une référence vers un autre objet <pre>@NotNull(message="Merci de préciser le thème de l'enquête") private Theme theme;</pre>
	@NotEmpty	Garantit que la chaîne de caractère ou la liste n'est pas vide <pre>@NotEmpty(message="La liste de critères ne peut être vide") private List<Critere> criteres;</pre>
	@Digits	Impose un nombre de chiffres avant la virgule et après la virgule. Cette annotation fonctionne sur un attribut de type int, long, byte, short, Integer, Long, Byte, Short, CharSequence, BigInteger ou BigDecimal <pre>@Digits(integer = 4, fraction = 2, message="Le budget alloué doit contenir 4 chiffres avant la virgule et 2 chiffres après la virgule") private BigDecimal budgetAlloue;</pre>
	@Min	Garantit que la valeur de l'attribut est supérieure ou égale à une valeur Min <pre>@Min(value=100, message="Le prix ne peut pas être inférieur à 100 euros") private float prix;</pre>
	@DecimalMin	Garantit que la valeur de l'attribut est supérieure ou égale à une valeur décimale Min <pre>@DecimalMin(value="100.5", message="Le prix ne peut pas être inférieur à 100.50 euros") private BigDecimal prix;</pre>
	@Max	Garantit que la valeur de l'attribut est inférieure ou égale à une valeur Max <pre>@Max(value=1000, message="Le prix ne peut pas être supérieur à 1000 euros") private float prix;</pre>

Contexte	Annotation	Description
	@DecimalMax	Garantit que la valeur de l'attribut est inférieure ou égale à une valeur décimale Max <pre>@DecimalMax(value="1000.99", message="Le prix ne peut pas être supérieur à 1000.99 euros") private BigDecimal prix;</pre>
	@Range	Garantit que la valeur de l'attribut est comprise entre la borne min et la borne max <pre>@Range(min=100, max=1000, message="Merci de préciser un prix compris entre 100 et 1000 euros") private Float prix;</pre>
	@Positive	Garantit que la valeur de l'attribut est strictement positive <pre>@Positive(message="Merci de préciser un nombre strictement positif") private Float prix;</pre>
	@PositiveOrZero	Garantit que la valeur de l'attribut est positive ou égale à zéro <pre>@PositiveOrZero(message="Merci de préciser un nombre supérieur ou égal à zéro") private Float prix;</pre>
	@Negative	Garantit que la valeur de l'attribut est négative <pre>@Negative(message="Merci de préciser un nombre strictement négatif") private Float valeur;</pre>
	@NegativeOrZero	Garantit que la valeur de l'attribut est négative ou égale à zéro <pre>@NegativeOrZero(message="Merci de préciser un nombre négatif ou égal à zéro") private Float valeur;</pre>
	@CreditCardNumber	Garantit que la valeur de l'attribut contient bien un numéro de carte de crédit valide. Pour ce faire Hibernate utilise l'algorithme de Luhn : https://fr.wikipedia.org/wiki/Formule_de_Luhn <pre>@NotNull(message="Merci de renseigner votre numéro de carte") @CreditCardNumber(message="Le numéro de la carte n'est pas valide")</pre>

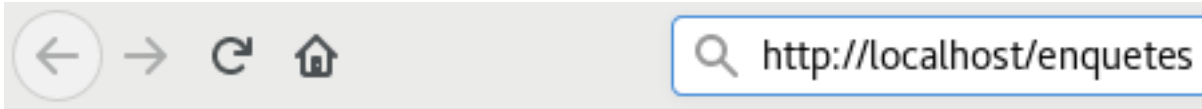
Contexte	Annotation	Description
		<p>private String numero;</p> <p>Exemples de numéro de carte valide : 371449635398431, 4111111111111111</p>
	@URL	<p>Garantit que la valeur de l'attribut correspond à une URL valide</p> <p>@URL(message="Merci de préciser une URL valide") private String url;</p>
	@Past	<p>Garantit que la valeur de l'attribut (de type Date) contient une date dans le passé</p> <p>@Past(message="La date de naissance doit être dans le passé") private Date dateDeNaissance;</p>
	@PastOrPresent	<p>Garantit que la valeur de l'attribut (de type Date) contient une date dans le passé ou aujourd'hui</p> <p>@PastOrPresent(message="La date de naissance doit être aujourd'hui ou dans le passé") private Date dateDeNaissance;</p>
	@Future	<p>Garantit que l'attribut (de type Date) contient une date dans le futur</p> <p>@Future(message="La date de lancement doit être dans le futur") private Date dateDeLancement;</p>
	@FutureOrPresent	<p>Garantit que l'attribut (de type Date) contient une date dans le futur ou la date actuelle</p> <p>@FutureOrPresent(message="La date de lancement doit être aujourd'hui ou dans le futur") private Date dateDeLancement;</p>
	@Pattern	<p>Garantit que la valeur de l'attribut de type String respecte une expression régulière précisée dans l'attribut regexp</p> <p>@Pattern(regexp="^[A-Za-z]+\$", message="La référence doit contenir uniquement des lettres") private String reference;</p> <p>Pour vous aider à écrire vos regex: http://jkorpela.fi/perl/regexp.html et https://regex101.com/</p>
	@Email	<p>Garantit que la valeur de l'attribut de type String contient une adresse email valide</p>

Contexte	Annotation	Description
		<pre>@Email(message="L'adresse email renseignée n'est pas valide") private String email;</pre>
	@Size	<p>Garantit que le nombre de caractères de l'attribut respecte les contraintes données en paramètre</p> <pre>@Size(min=5, message="Le mot de passe doit contenir au minimum 5 caractères") private String motDePasse;</pre>
	@Length	<p>Garantit que le nombre de caractères de l'attribut respecte les contraintes données en paramètre. @Length appartient à Hibernate Validator alors que @Size appartient à JPA</p> <pre>@Length(min=5, max=100, message="La description doit contenir entre 5 et 100 caractères") private String description;</pre>
	@Valid	<p>Demande la validation des données de l'objet vis-à-vis des contraintes exprimées dans la classe de l'objet.</p> <p>L'annotation @Valid s'utilise dans la couche contrôleur Spring.</p> <p>Exemple : on demande au moment de l'invocation de la méthode la validation des données de l'objet enquete :</p> <pre>public ModelAndView enregistrerEnquetePost(@Valid @ModelAttribute("enquete") Enquete enquete, BindingResult result) {}</pre>
Spring	@Autowired	<p>Demande à Spring d'injecter une dépendance dans l'objet de la classe considérée.</p> <p>Exemple : demande d'injection de la dépendance enqueteDao dans une classe de service :</p> <pre>@Autowired private final EnqueteDao enqueteDao;</pre> <p>Cette annotation est de moins en moins utilisée car la communauté Spring suggère de demander l'injection de dépendances via le constructeur de la classe. En suivant cette recommandation,</p>

Contexte	Annotation	Description
		<p>l'exemple ci-dessus sera ré-écrit :</p> <pre>private final EnqueteDao enqueteDao; public EnqueteServiceImpl(EnqueteDao enqueteDao) { super(); this.enqueteDao = enqueteDao; }</pre>
	@Component	<p>Laisse à Spring la responsabilité de déterminer le stéréotype ou le rôle de la classe, autrement dit à quelle couche logicielle appartient la classe annotée. Cette annotation fonctionne aussi sur des filtres (classes implémentant l'interface javax.servlet.Filter)</p> <p>Il existe cinq types de stéréotypes : https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/stereotype/package-summary.html</p> <pre>@Component public class EnqueteController {}</pre>
	@Controller	<p>Déclare une classe qui va traiter les requêtes HTTP. Cette classe est considérée comme le contrôleur dans l'architecture MVC</p> <pre>@Controller public class EnqueteController {}</pre>
	@RestController	<p>Déclare une classe qui va traiter des requêtes HTTP de type REST</p> <pre>@RestController public class EnqueteControllerWS {}</pre>
	@Service	<p>Déclare une classe de service</p> <pre>@Service public class EnqueteServiceImpl implements EnqueteService {}</pre> <p>En oubliant l'annotation @Service sur la classe de service on obtient l'erreur : Parameter 2 of constructor in</p>

Contexte	Annotation	Description
		fr.clelia.fx.enquete.controller.EnqueteController required a bean of type 'fr.clelia.fx.enquetes.service.EnqueteService' that could not be found.
	@Repository	<p>Déclare une classe DAO (Data Access Object : classe capable de communiquer avec la base de données)</p> <pre>@Repository public class JoueurDaoImpl implements JoueurDao {}</pre> <p>Grâce à Spring Data, cette annotation n'est quasiment plus utilisée car pour mettre en œuvre une DAO il suffit de déclarer une interface qui hérite de JpaRepository, PagingAndSortingRepository ou CrudRepository</p> <p>Spring Data implémente les DAO à notre place, voir la classe SimpleJpaRepository :</p> <p>https://github.com/spring-projects/spring-data-jpa/blob/main/src/main/java/org/springframework/data/jpa/repository/support/SimpleJpaRepository.java</p>
	@Query	<p>Déclare la requête HQL associée à la méthode de l'interface DAO</p> <pre>@Query("FROM Theme t ORDER BY size(t.enquetes) DESC") List<Theme> findThemesSortedByNbEnquetes();</pre> <p>Il est possible de fournir une requête SQL native en utilisant l'attribut nativeQuery=true</p> <p>A noter: sans l'annotation @Query, Spring Data génère à la volée une requête HQL en analysant le nom de la méthode (on parle alors de requête par dérivation) :</p> <p>https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods.query-creation</p> <p>Pour résumer, on peut déclarer des méthodes dans une interface DAO de trois manières différentes :</p>

Contexte	Annotation	Description
		<pre>// Méthode annotée avec @Query portant une requête HQL @Query("FROM Theme ORDER BY id DESC") List<Theme> findThemes(); // Méthode annotée avec @Query portant une requête SQL native @Query(value="SELECT id, nom FROM theme ORDER BY id DESC", nativeQuery=true) List<Theme> findThemes2(); // Méthode dont le nom sera traduit par Spring Data pour produire une requête HQL // Cette requête HQL sera transmise à Hibernate qui la traduira en requête SQL List<Theme> findByNomContainingOrderByDesc(String nom);</pre>
	@Param	<p>Déclare les paramètres de la méthode Java devant être utilisés comme des paramètres HQL</p> <pre>@Query("from Question where enquete.id=:eid") public List<Question> findByIdEnquete(@Param("eid") Long idEnquete);</pre>
	@Modifying	<p>Annote une méthode d'une interface de la DAO qui exécute une requête de type INSERT, UPDATE ou DELETE</p> <pre>@Modifying @Transactional @Query("UPDATE Enquete SET nom=upper(nom)") void updateNomEnquetes();</pre>
	@Transactional	<p>Définit un contexte transactionnel (typiquement sur les méthodes des classes de service ou sur la classe de service)</p> <p>Si une annotation @Transactional est présente sur la classe et sur une méthode, c'est l'annotation de la méthode qui l'emporte.</p> <p>Pour définir une transaction en lecture seule (ce qui est très fréquent pour les méthodes de récupération dans les classes de services) : @Transactional(readOnly = true)</p> <pre>@Service @Transactional public class EnqueteServiceImpl implements EnqueteService {}</pre>

Contexte	Annotation	Description
	@RequestMapping	<p>Définit une correspondance entre une (ou plusieurs) URL et une méthode du contrôleur. En d'autres termes, cette annotation déclare la correspondance entre une méthode du contrôleur et la ou les URL qu'elle prend en charge</p> <pre>@RequestMapping(value = { "index", "/"}, method = RequestMethod.GET) public ModelAndView accueil() {}</pre>
	@GetMapping	<p>Définit une correspondance entre une (ou plusieurs) URL et une méthode du contrôleur Spring. Cette correspondance s'applique uniquement si la méthode HTTP de la requête est Get.</p> <p>La méthode enqueteGet ci-dessous sera invoquée :</p> <ul style="list-style-type: none"> - lorsque l'internaute saisit l'adresse http://localhost/enquetes dans la barre d'adresse de son navigateur internet puis tape Entrée :  <ul style="list-style-type: none"> - ou lorsqu'un hyperlien avec un attribut href égal à "http://localhost/enquetes" est cliqué <pre>@GetMapping("enquetes") public ModelAndView enquetesGet() {}</pre>
	@PostMapping	<p>Définit une correspondance entre une (ou plusieurs) URL et une méthode du contrôleur Spring. Cette correspondance s'applique uniquement si la méthode de la requête HTTP est Post.</p> <pre>@PostMapping("filtrerLesEnquetes") public ModelAndView filtrerEnquetes(@RequestParam Map<String, String> map) {}</pre> <p>A noter: La méthode ci-dessous est invoquée lorsque le bouton submit du formulaire ci-dessous est cliqué:</p> <pre><form action="filtrerLesEnquetes" method="post"> <input type="text" name="NOM"></pre>

Contexte	Annotation	Description
		<pre><input type="submit" value="Filtrer"> </form></pre>
	@PutMapping	<p>Définit une correspondance entre une (ou plusieurs) URL et une méthode du contrôleur Spring. Cette correspondance s'applique uniquement si la méthode de la requête HTTP est Put.</p> <pre>@PutMapping(value="mettreAJourEnquete") public Enquete majEnquete(@RequestBody Enquete enquete) {}</pre>
	@PatchMapping	<p>Définit une correspondance entre une (ou plusieurs) URL et une méthode du contrôleur Spring. Cette correspondance s'applique uniquement si la méthode de la requête HTTP est Patch.</p> <pre>@PatchMapping(value="mettreAJourPrixEnquete") public Enquete majPrixEnquete(@RequestParam Map<String, String> map) {}</pre>
	@DeleteMapping	<p>Définit une correspondance entre une (ou plusieurs) URL et une méthode du contrôleur Spring. Cette correspondance s'applique uniquement si la méthode de la requête HTTP est Delete.</p> <pre>@DeleteMapping(value="supprimerEnquete") public boolean supprimerEnquetes(@RequestParam Long idEnquete) {}</pre>
	@ResponseStatus	<p>Annotation qui impose le code retour de la réponse à une requête HTTP</p> <p>Il est conseillé d'utiliser l'énumération HttpStatus : https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/HttpStatus.html</p> <p>Exemples de codes retour usuels :</p> <p>200 OK 201 CREATED 204 NO CONTENT 302 FOUND 400 BAD REQUEST 404 NOT FOUND 409 CONFLICT</p>

Contexte	Annotation	Description
		<p>422 UNPROCESSABLE ENTITY</p> <p><code>@ResponseStatus (code=HttpStatus.OK)</code></p>
	@ExceptionHandler	<p>Annotation à placer au dessus du méthode qui traite une exception</p> <pre> @ExceptionHandler(fr.clelia.fx.enquetes.exception.ThemeExistantException) @ResponseStatus (code=HttpStatus.CONFLICT) public String traiterThemeDejaExistant(Exception exception) { return exception.getMessage(); } </pre>
	@RequestParam	<p>Indique que le paramètre de la méthode (du contrôleur) provient de l'objet request. En d'autres termes, on demande à Spring de lire un paramètre existant dans la requête HTTP.</p> <p>Spring convertira l'objet dans le type attendu.</p> <p>Exemple 1 : Lecture des données dans l'URL On souhaite lire le paramètre ID fourni dans l'URL suivante : http://localhost:8080/enquete?ID=2</p> <p>Pour cela il faut déclarer un paramètre annoté @RequestParam comme suit :</p> <pre> @GetMapping(value="enquete") public ModelAndView enqueteGet(@RequestParam(name="ID") Long id) {} </pre> <p>Exemple 2 : Lecture des données saisies sur un formulaire HTML On considère le formulaire HTML suivant :</p> <pre> <form action="enregistrerQestion" method="post"> <input type="text" name="LIBELLE"> <input type="hidden" name="ID_ENQUETE"> <input type="submit" value="Enregistrer"> </form> </pre> <p>La méthode du contrôleur doit être annotée PostMapping car l'attribut method de la balise form est post.</p>

Contexte	Annotation	Description
		<p>L'attribut value de l'annotation PostMapping doit correspondre à la valeur de l'attribut action de la balise form</p> <pre>@PostMapping(value = "enregistrerQuestion") public ModelAndView questionPost(@RequestParam(name="ID_ENQUETE") Long idEnquete, @RequestParam(name="LIBELLE") String libelle) {}</pre> <p>Il est possible de rassembler tous les paramètres dans un dictionnaire comme suit :</p> <pre>@RequestMapping(value = { "index", "/"}, method = RequestMethod.GET) public ModelAndView accueil(@RequestParam Map<String, String> map) {}</pre> <p>Pour ne pas obliger la présence d'un paramètre (et éviter un code retour 400 si le paramètre est absent), l'attribut required sera affecté à false :</p> <pre>public ModelAndView questionPost(@RequestParam(name="ID_ENQUETE") Long idEnquete, @RequestParam(name="LIBELLE", required=false) String libelle) {}</pre> <p>Il est possible de donner une valeur par défaut dans le cas où le paramètre n'est pas présent dans la requête HTTP :</p> <pre>public ModelAndView questionPost(@RequestParam(name="ID_ENQUETE") Long idEnquete, @RequestParam(name="LIBELLE", required=false, defaultValue="Indéfini") String libelle) {}</pre>
	@ModelAttribute	<p>Annotation utilisée sur un paramètre d'une méthode du contrôleur. Elle accompagne un objet d'une classe métier retourné par une vue (Thymeleaf ou JSP)</p> <pre>@PostMapping("inscription") public ModelAndView inscriptionPost(@Valid @ModelAttribute("utilisateur") Utilisateur utilisateur, BindingResult result) { if (result.hasErrors()) { // La validation de l'objet utilisateur par rapport aux contraintes // de validation définies dans la classe métier Utilisateur a produit // des erreurs ModelAndView mav = inscriptionGet();</pre>

Contexte	Annotation	Description
		<pre> mav.addObject("utilisateur", utilisateur); return mav; } else { utilisateurService.enregistrerUtilisateur(utilisateur); return new ModelAndView("redirect:merciInscription"); } } </pre> <p>A noter: la vue (la JSP inscription.jsp) doit contenir une balise form:form comme suit: <code><form:form action="inscription" method="post" modelAttribute="utilisateur"></code></p> <p>Pour rappel: l'annotation @Valid permet de déléguer à Spring le travail de validation sur l'objet annoté. En d'autres termes, toutes les contraintes de validation exprimées dans les classes métier (présentes dans le package business) seront vérifiées par Spring.</p>
	@SessionAttributes	<p>Demande à Spring de stocker en session les attributs d'un objet envoyé à un formulaire HTML</p> <pre> @Controller @SessionAttributes("enquete") public class EnqueteController {} </pre>
	@DateTimeFormat (pattern = "dd/MM/yyyy")	<p>Permet de définir un attribut de type Date, LocalDate, LocalTime ou LocalDateTime avec un format précisé en attribut. L'utilisation de cette annotation dispense d'écrire du code dans la méthode annotée initBinder</p> <pre> @DateTimeFormat(pattern = "dd/MM/yyyy") private Date dateDeLancement; </pre> <p>A la place d'un attribut pattern, on peut se reposer sur l'attribut iso qui fera appel à une des valeurs de l'énumération DateTimeFormat.ISO :</p> <p>https://docs.spring.io/spring-framework/docs/current/javadoc-api/org.springframework.format/annotation/DateTimeFormat.ISO.html</p> <pre> @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate dateDeNaissance private LocalDate dateDeNaissance; </pre>
	@PageableDefault	Définit les paramètres de pagination et de tri par défaut

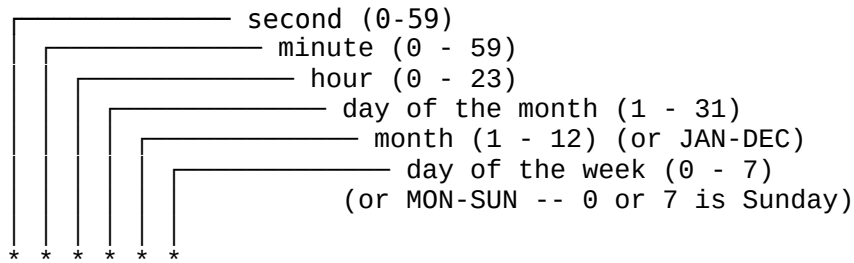
Contexte	Annotation	Description
		<p>public ModelAndView enquetesGet(@PageableDefault(size = 10, sort = "dateDeLancement") Pageable pageable) {}</p> <p>A noter : dans les services et les DAO, toutes les méthodes ayant un paramètre de type Pageable doivent renvoyer une page</p>
	@SortDefault	<p>Définit les paramètres de tri par défaut</p> <p>public ModelAndView enquetesGet(@PageableDefault(size = 10) @SortDefault(sort = "dateDeLancement", direction = Sort.Direction.DESC) Pageable pageable) {}</p>
	@InitBinder	<p>Annotation d'une méthode "montrant" à Spring comment obtenir un objet métier à partir de son id ou comment transformer un objet en un autre objet. La conversion de données (en anglais binding) est réalisée par Spring grâce aux méthodes annotées @InitBinder</p> <pre> @InitBinder public void initBinder(WebDataBinder binder) { // Apprend à Spring à convertir un String en Date SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd"); dateFormat.setLenient(false); binder.registerCustomEditor(Date.class, new CustomDateEditor(dateFormat, true)); // Apprend à Spring à convertir un id de Theme en objet de type Theme binder.registerCustomEditor(Theme.class, "theme", new PropertyEditorSupport() { @Override public void setAsText(String id) { setValue((id.equals("")) ? null : themeService.recupererTheme(Long.parseLong((String) id)); } }); // Apprend à Spring à convertir une liste d'id en liste d'intérêts binder.registerCustomEditor(List.class, "interets", new CustomCollectionEditor(List.class) { @Override public Object convertElement(Object objet) { Long id = Long.parseLong((String) objet); return interetService.recupererInteret(id); } }); </pre>

Contexte	Annotation	Description
		<pre> } }); } </pre> <p>Pour la classe PropertyEditorSupport, se référer à la javadoc : https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/java/beans/PropertyEditorSupport.html</p> <p>L'annotation InitBinder peut accueillir un attribut value dont la valeur contient le nom de l'objet à convertir :</p> <pre> @InitBinder(value="theme.nom") public void initBinder(WebDataBinder binder) { // Apprend à Spring à convertir un nom de Theme en objet de type Theme binder.registerCustomEditor(Theme.class, new PropertyEditorSupport() { @Override public void setAsText(String nom) { setValue((nom.equals("")) ? null : themeService.recupererTheme(nom)); } }); } </pre>
	@PostConstruct	<p>Annotation sur une méthode que Spring va invoquer juste après avoir instancié la classe et injecté toutes les dépendances dans l'objet instancié</p> <p>Exemple : la méthode ajouterDonneesInitiales() est présente dans le contrôleur EnqueteController. Une fois que Spring a instancié un objet de cette classe et injecté toutes les dépendances (notamment les services), la méthode ajouterDonneesInitiales() sera invoquée :</p> <pre> @PostConstruct public void ajouterDonneesInitiales() {} </pre>
	@PreDestroy	<p>Annotation sur une méthode qui sera invoquée automatiquement juste avant que Spring retire le bean de son contexte applicatif</p>

Contexte	Annotation	Description
		<pre>@PreDestroy public void retirerDonneesInitiales() {}</pre>
	@PathVariable	<p>Désigne une variable qui se trouve dans le chemin de l'URL (et non en paramètre (pour rappel les paramètres se trouvent après le point de l'interrogation d'une URL))</p> <p>Exemple d'URL: http://localhost:8080/enquete/4</p> <p>Le code ci-dessous récupère l'id de l'enquête à partir de l'URL. (idEnquete aura dans l'URL précédente la valeur 4)</p> <pre>@GetMapping(value="/enquete/{idEnquete}", produces="MediaType.APPLICATION_JSON_VALUE") public Enquete enqueteGet(@PathVariable Long idEnquete) {}</pre>
	@Bean	<p>Déclare une méthode dont l'objet retourné sera géré par le conteneur Spring</p> <pre>@Bean public EmbeddedServletContainerFactory servletContainer() {}</pre>
	@Scope	<p>Précise la portée du bean. Par défaut Spring crée une seule instance d'un bean (singleton). Pour que Spring crée une nouvelle instance à chaque fois que la méthode est invoquée, il faut utiliser la valeur @Scope("prototype")</p> <pre>/** * Cette méthode affiche en console tous les objets présents dans le * conteneur Spring * @param applicationContext * @return */ @Bean @Scope(ConfigurableBeanFactory.SCOPE_SINGLETON) public CommandLineRunner commandLineRunner(ApplicationContext applicationContext) { return args -> { String[] noms = applicationContext.getBeanDefinitionNames();</pre>

Contexte	Annotation	Description
		<pre> for (String nom : noms) { System.out.println(nom + " : " + applicationContext.getBean(nom).getClass().getSimpleName()); } }; } </pre>
	@Configuration	<p>Déclare une classe de configuration (un objet de cette classe remplace le fichier xml de configuration de Spring (souvent appelé spring-servlet.xml))</p> <pre> @Configuration public class EnquetesConfiguration {} </pre> <p>A noter : les classes de configuration peuvent être remplacées par des lignes dans le fichier texte src/main/resources/application.properties systématiquement lu par Spring Boot</p>
	@Value	<p>Utilise la valeur d'une variable définie dans le fichier src/main/resources/application.properties:</p> <pre> @Value("\${google.client.client-secret}") private String clientSecret; @Value("#{ systemProperties['user.region'] }") private String locale; </pre>
	@RequestBody	<p>Récupère les données dans le corps de la requête HTTP</p> <pre> @PostMapping("/theme") public Theme ajouterTheme(@RequestBody Theme theme) { return themeService.enregistrerTheme(theme); } </pre>
	@ResponseBody	<p>Précise que le retour de la méthode correspond à la réponse qui va être envoyée au client HTTP</p> <pre> @GetMapping(value="/fichierExcel", produces="application/vnd.ms-excel") public @ResponseBody byte[] fichierExcelGet(@RequestParam(name="ID") Long idFichierExcel) throws IOException {} </pre> <p>Cela évite, entre autres, d'écrire une méthode ayant un paramètre de type HttpServletResponse</p>

Contexte	Annotation	Description
		<p>comme suit :</p> <pre>public void fichierExcel(HttpServletResponse response, @RequestParam(name="ID") Long idFichierExcel) {}</pre>
	@Secured	<p>Restreint l'accès aux utilisateurs ayant le rôle précisé en paramètre</p> <pre>@Secured("ROLE_ADMIN") @GetMapping("/enquetes") public ModelAndView enquetesGet() {}</pre>
	@PreAuthorize	<p>Restreint l'exécution d'une méthode au(x) rôles précises en argument. Cette annotation accompagne souvent les méthodes d'une DAO</p> <pre>@PreAuthorize("hasRole('ADMIN') or hasRole('MODERATEUR')")</pre>
	@Scheduled	<p>Programme l'invocation de la méthode de manière automatique.</p> <pre>@Scheduled(fixedRate = 1000, initialDelay = 5000) public void notifierUtilisateurs() {}</pre> <p>Alternativement , cette annotation accepte un attribut cron.</p> <p>L'exemple ci-dessous invoque la méthode envoyerEmails() tous les jeudis à 17h:</p> <pre>@Scheduled(cron="00 00 17 * * THU") public void envoyerEmails() {}</pre> <p>L'exemple ci-dessous invoque la méthode envoyerSMS() toutes les secondes:</p> <pre>@Scheduled(cron="* * * * *") public void envoyerSMS() {}</pre> <p>L'exemple ci-dessous invoque la méthode envoyerMails() toutes les 5 minutes:</p> <pre>@Scheduled(cron="0 */5 * * * *") public void envoyerSMS() {}</pre>

Contexte	Annotation	Description
		 <p>Source : https://spring.io/blog/2020/11/10/new-in-spring-5-3-improved-cron-expressions</p>
	@Primary	<p>Indique la classe d'implémentation que Spring devra utiliser face à un objet déclaré avec une interface (à utiliser lorsque plusieurs classes implémentent une même interface)</p> <pre>@Primary @Service public class EnqueteServiceImpl implements EnqueteService {}</pre>
	@Qualifier	<p>Permet de distinguer deux paramètres de même type dans une méthode</p> <pre>@RequestMapping(value = { "/index", "/"}, method = RequestMethod.GET) public ModelAndView accueil(@Qualifier("enquete") @PageableDefault(value = 10, sort = "nom") Pageable pageableEnquete, @Qualifier("utilisateur") @PageableDefault(value = 4, sort = "login") Pageable pageableUtilisateur) { ModelAndView mav = new ModelAndView("index"); mav.addObject("pageDEnquetes", enqueteService.recupererEnquetes(pageableEnquete)); mav.addObject("pageDUtilisateurs", utilisateurService.recupererUtilisateurs(pageableUtilisateur)); return mav; }</pre>

Contexte	Annotation	Description
		<p>Exemple d'URL: http://localhost:8080/index?utilisateur_page=4&enquete_page=1</p> <p>Exemple d'URL avec tri : http://localhost:8080/index?utilisateur_page=4&enquete_page=1&enquete_sort=nom</p>
	@RepositoryRestResource	<p>Génère automatiquement une API pour REST pour l'entité associée (les méthodes disponibles sont GET, POST, PUT, PATCH et DELETE). L'attribut exported a la valeur true par défaut</p> <p>https://docs.spring.io/spring-data/rest/docs/current/reference/html/#reference</p> <pre>@RepositoryRestResource(exported = true) public interface EnqueteDao extends JpaRepository<Enquete, Long> {}</pre>
	@PersistenceContext	<p>Récupère l'entity manager (aka persistence manager) associé au projet</p> <pre>@PersistenceContext private EntityManager entityManager;</pre> <p>Un avantage est de récupérer une session de la SessionFactory grâce à la méthode unwrap :</p> <pre>Session session = entityManager.unwrap(Session.class); CriteriaBuilder criteriabuilder = session.getCriteriaBuilder();</pre>
	@EntityScan	<p>Active l'analyse des classes présentes dans le package de la classe annotée ainsi que des classes présentes dans les sous-packages. Spring va tester l'existence de classes annotées @Entity</p> <pre>@EntityScan("fr.clelia.fx.domain")</pre>
	@ComponentScan	<p>Active l'analyse des classes présentes dans le package de la classe annotée ainsi que des classes présentes dans les sous-packages. Spring va tester l'existence de classes annotées @Component, @Service, @Repository, etc.</p> <p>Avec cette annotation, le nom d'un package peut être indiqué :</p> <pre>@ComponentScan("fr.clelia.fx.un_autre_package_a_scanner")</pre>

Contexte	Annotation	Description
	@SpringBootApplication	Annotation qui remplace les trois annotations suivantes : @Configuration, @EnableAutoConfiguration et @ComponentScan. Cette annotation se trouve par défaut sur la classe exécutable du package racine : @SpringBootApplication public class EnquetesApplication {}
	@EnableJpaRepositories	Annotation activant les JPA repositories dans les packages déclarés en attribut
	@EnableAutoConfiguration	Laisse au framework la configuration des beans qui seront très probablement nécessaires pour l'application
	@EnableScheduling	Annotation nécessaire dans la classe exécutable pour que les méthodes annotées @Scheduled se lancent comme programmées https://spring.io/guides/gs/scheduling-tasks/ @SpringBootApplication @EnableScheduling public class EnquetesApplication {}
Spring Test	@SpringBootTest	Annotation à placer au dessus d'une classe de test pour indiquer à Spring qu'il s'agit d'une classe de test. @SpringBootTest @TestMethodOrder(MethodOrderer.OrderAnnotation.class) public class EnqueteServiceImplTest { @Autowired private EnqueteService enqueteService; @Test public void testerAjouterEnquete() { String nom = "Enquete de test"; float prix = 10f; Enquete enquete = enqueteService.ajouterEnquete(nom, prix); assertNotNull(enquete); assertNotNull(enquete.getNom()); } }

Contexte	Annotation	Description
		<pre> assertEquals(enquete.getNom(), nom); assertEquals(enquete.getPrix(), prix); } </pre> <p>Il est possible d'utiliser des mockMvc :</p> <pre> @WebMvcTest(EnqueteController.class) public class EnqueteControllerTest { private MockMvc mockMvc; @Autowired private EnqueteController enqueteController; @BeforeEach public void setup() { mockMvc = MockMvcBuilders.standaloneSetup(enqueteController).build(); } @Test public void testerConnexionAvecSucces() throws Exception { this.mockMvc .perform(MockMvcRequestBuilders.post ("/connexion").accept(MediaType.TEXT_HTML) .param("email", "fxcote@clelia.fr").param("password", "12345")) .andExpect(view().name("enquetes")).andExpect(status().isOk()); } } </pre>
	@DataJpaTest	<p>Annotation destinée à une classe testant une DAO</p> <p>Les tests dans cette classe utilisent par défaut une base de données stockée en mémoire</p>
	@Test	Annotation à utiliser sur chaque méthode que la phase de test doit invoquer
	@MockBean	Annotation à placer sur l'objet que la classe de test doit simuler

Contexte	Annotation	Description
	@AutoConfigureMockMvc	Annotation à placer sur la classe de test pour que Spring configure automatiquement les Mocks Mvc
	@Sql	Annotation à placer sur la classe de test pour que Spring exécute le ou les scripts SQL donnés en paramètre <pre>@Sql({"drop.sql", "create.sql"}) public class EnqueteControllerTest {}</pre>
Lombok	@NoArgsConstructor	Annotation ajoutant un constructeur par défaut, c'est-à-dire sans paramètre <pre>@NoArgsConstructor public class Enquete { }</pre>
	@AllArgsConstructor	Annotation ajoutant un constructeur contenant un paramètre pour chaque attribut de la classe <pre>@AllArgsConstructor public class Enquete { }</pre>
	@Getter	Annotation remplaçant les accesseurs <pre>@Getter public class Enquete { }</pre>
	@Setter	Annotation remplaçant les mutateurs <pre>@Setter public class Enquete { }</pre>
	@EqualsAndHashCode	Annotation remplaçant les méthodes equals() et hashCode() <pre>@EqualsAndHashCode public class Enquete { }</pre>
	@ToString	Annotation remplaçant la méthode toString <pre>@ToString public class Enquete { }</pre> <p>La collection d'attributs peut être renseignée dans l'attribut of :</p>

Contexte	Annotation	Description
		<pre>@ToString(of= {"id", "nom", "prix"}) public class Enquete { }</pre> <p>Il est cependant recommandé d'exclure un attribut de la méthode toString comme suit :</p> <pre>@ToString.Exclude private List<Question> questions;</pre>
	@Data	<p>Annotation qui remplace les annotations @Getter @Setter @EqualsAndHashCode et @RequiredArgsConstructor</p> <pre>@Data public class Enquete { }</pre>
	@Log	<p>Annotation qui ajoute à la classe annotée un attribut nommé log de type java.util.logging.Logger. Le niveau de log est défini dans le fichier application.properties ou dans le fichier application.yml : logging.level.root=INFO</p> <pre>@Controller @Log public class EnqueteController { public EnqueteController() { log.info("Invocation du constructeur par défaut du contrôleur EnqueteController"); } }</pre>
	@Log4j2	<p>Annotation qui ajoute à la classe annotée un attribut nommé log de type org.apache.logging.log4j.Logger</p> <pre>@Controller @Log4j2 public class EnqueteController { public EnqueteController() { log.info("Invocation du constructeur par défaut du contrôleur EnqueteController"); } }</pre>

Contexte	Annotation	Description
		<pre> } } </pre>
	@Slf4j	<p>Annotation qui ajoute à la classe annotée un attribut nommé log de type org.slf4j.Logger</p> <pre> @Controller @Slf4j public class EnqueteController { public EnqueteController() { log.info("Invocation du constructeur par défaut du contrôleur EnqueteController"); } } </pre>
	@FieldDefaults	<p>Annotation expérimentale utilisée pour définir, entre autres, la visibilité des attributs</p> <pre> @FieldDefaults(level= AccessLevel.PRIVATE) public class Enquete { } </pre>
Jackson	@JsonProperty	<p>Annotation servant à indiquer le nom de l'attribut tel qui doit apparaître dans la version sérialisée en JSON de l'objet Java</p> <pre> @JsonProperty(value="prix") private float prixTTC; </pre> <p>L'ordre de sérialisation est imposé avec l'attribut index : Value="", index=2</p> <pre> @JsonProperty(value="prix", index=2) private float prixTTC; </pre>
	@JsonIgnore	<p>Annotation servant à ne pas inclure l'attribut annoté dans l'expression JSON de l'objet Java</p> <pre> @JsonIgnore private float prixTTC; </pre>

Références :

- Jakarta Persistence : <https://jakarta.ee/specifications/persistence/>
- Lombok : <https://projectlombok.org/features/all>