Questions Jour 2

Job 1

Le Modèle MVC permet de structurer le code et de séparer clairement les responsabilités de chaque partie d'une application. Dans Spring Boot, ce modèle est enrichi par une architecture plus modulaire, qui facilite la maintenance et l'évolution de l'application. Il contient :

- View : Génère l’interface graphique (souvent en HTML) en utilisant le moteur de template Thymeleaf, qui permet d’intégrer dynamiquement des données côté serveur dans les pages web.
- Controller : Gère les requêtes HTTP provenant du client (navigateur), traite les actions demandées, et renvoie la vue appropriée. Il est généralement annoté avec @Controller ou @RestController.
- Repository : Représente la couche d’accès aux données. Elle communique directement avec la base de données en utilisant Spring Data JPA, et est souvent annotée avec @Repository.
- Service : Contient la logique métier de l’application. C’est une couche intermédiaire entre le contrôleur et le repository, permettant une meilleure séparation des responsabilités. Annotée avec @Service.
- Model / Entity : Représente les objets métiers (entités) de l’application. Ces classes sont généralement annotées avec @Entity et sont mappées aux tables de la base de données.

Cette structure permet une application plus claire, modulaire et évolutive, facilitant les tests, la réutilisation du code et la collaboration entre développeurs.

```
 [ Navigateur ]
       ↓
 Requête HTTP (GET, POST...)
       ↓
[ Controller (UserController.java) ]
       ↓
 Appelle la couche Service
       ↓
[ Service (UserService.java) ]
       ↓
 Appelle la couche Repository
       ↓
[ Repository (UserRepository.java) ]
       ↓
 Accède aux données via le Model
       ↓
[ Model / Entity (User.java) ]
       ↓
 Retourne les données
       ↓
[ View (users.html avec Thymeleaf) ]
       ↓
 Génère une réponse HTML
       ↓
 Réponse HTTP
       ↓
 [ Navigateur ]
```

```
src/
└── main/
    ├── java/
    │   └── com.example.demo/
    │       ├── controller/
    │       │   └── UserController.java     <-- Gère les requêtes HTTP (@Controller)
    │       ├── model/
    │       │   └── User.java               <-- Entité / Objet métier (@Entity)
    │       ├── repository/
    │       │   └── UserRepository.java     <-- Accès aux données (@Repository)
    │       └── service/
    │           └── UserService.java        <-- Logique métier (@Service)
    └── resources/
        └── templates/
            └── users.html                  <-- Vue générée avec Thymeleaf
```

Job 2

Thymeleaf permet, à la différence des templates HTML de créer des pages HTML qui sont modifiables dynamiquement dans l'application. On affiche le contenu de variables et non seulement le texte brut de l'HTML.

Job 3

Pour passer des données depuis un contrôleur vers une vue HTML, on utilise un objet de type Model. On crée une méthode de type String qui retourne le nom de la vue et qui prend le Model en paramètre. Dans cette méthode, on crée une variable contenant les données à transmettre, puis on l'ajoute au Model avec la méthode addAttribute. Enfin, dans la vue, grâce à Thymeleaf, on récupère et affiche dynamiquement le contenu de cette variable.

Job 4

Spring facilite la gestion des formulaires en permettant de lier automatiquement les données saisies dans un formulaire HTML aux paramètres d'une méthode du contrôleur, grâce à l'annotation @RequestParam. Il permet aussi de rediriger les données vers une vue via un objet Model, rendant le traitement et l'affichage des données simples et structurés. Avec Thymeleaf, Spring offre une intégration fluide entre le backend et le frontend, pour afficher dynamiquement les résultats après soumission.

Job 5

Spring permet de valider les données d'un formulaire en utilisant des annotations (comme @NotBlank, @Min, etc.) dans les classes du modèle. La validation est activée dans le contrôleur grâce à l'annotation @Valid, et les erreurs peuvent être récupérées avec BindingResult.