**Jour 01**

**Question :** Qu'est-ce que Spring Initializr et comment peut-il faciliter la création d'un nouveau projet Spring ?

Spring Initializr permet de démarrer un projet avec le Framework Spring avec les dossiers bien organisés, déjà préparés avec la bonne terminaison des fichiers.

**Question :** Pourquoi le fichier pom.xml est-il crucial dans un projet Maven ?

Il permet de loader Maven avec la bonne configuration, de charger les parents, les licences, les propriétés de la version java, et toutes les dépendances.

**Question :** Qu'est-ce qu'un contrôleur dans le contexte de Spring MVC ?

Dans Spring MVC, un contrôleur est une classe Java qui joue le rôle d’intermédiaire entre l’utilisateur (le client) et la logique de l’application.

**Question :** Comment Spring permet-il l’injection de propriétés ?

Spring Boot utilise le mécanisme d’Environment et le PropertySource pour gérer la configuration.

En gros : Spring Boot fait correspondre les clés définies dans application.yml avec les placeholders @Value("${...}") et les injecte au démarrage.

**Question :** Pourquoi avoir différents profils ?

Avoir des profils permet de changer facilement le comportement d’une application selon l’environnement :

* Dev → logs détaillés, base de données de test, fonctionnalités expérimentales.
* Prod → logs réduits, base de données réelle, configuration sécurisée.
* Test → base de données en mémoire, mocks, etc.

Avantages :

* Evite de modifier le code pour passer d’un environnement à un autre.
* Permet de centraliser les configurations spécifiques à chaque environnement.
* Réduit les risques d’erreurs (ex : ne pas utiliser la DB de prod en dev).

**Question :** Pourquoi DevTools est bénéfique ?

* Redémarrage automatique : plus besoin de relancer manuellement votre serveur à chaque modification.
* Rechargement rapide des ressources et templates (HTML, YAML, propriétés, fichiers statiques).
* Mode développement : active des fonctionnalités spécifiques pour dev, comme le LiveReload.
* Gain de temps énorme lors du développement, surtout pour tester rapidement les changements.

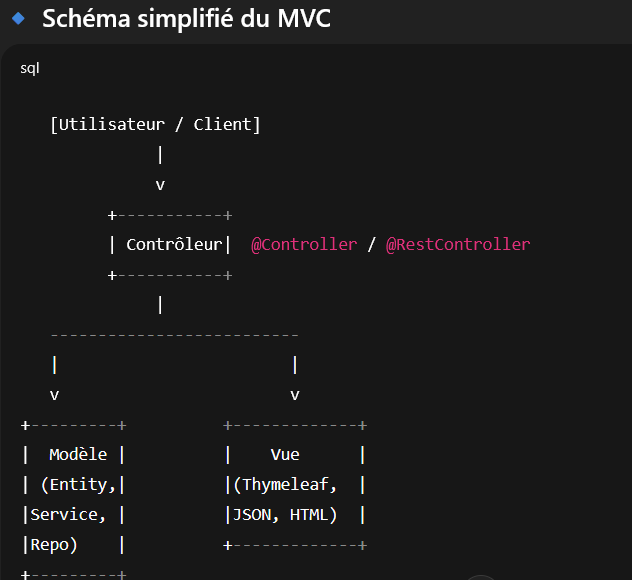
**Jour 02 :**

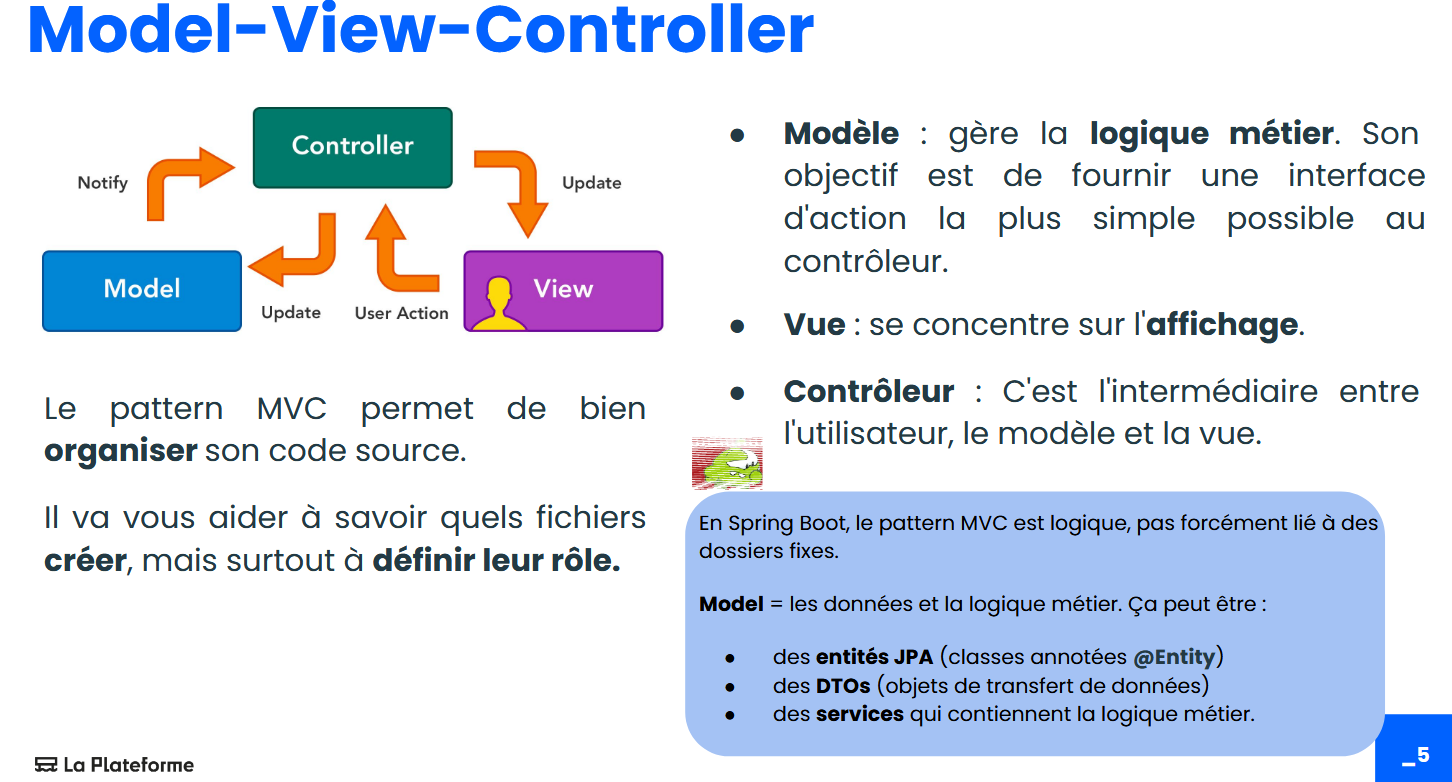
**Question :** Comment le modèle MVC aide-t-il à structurer une application web ?

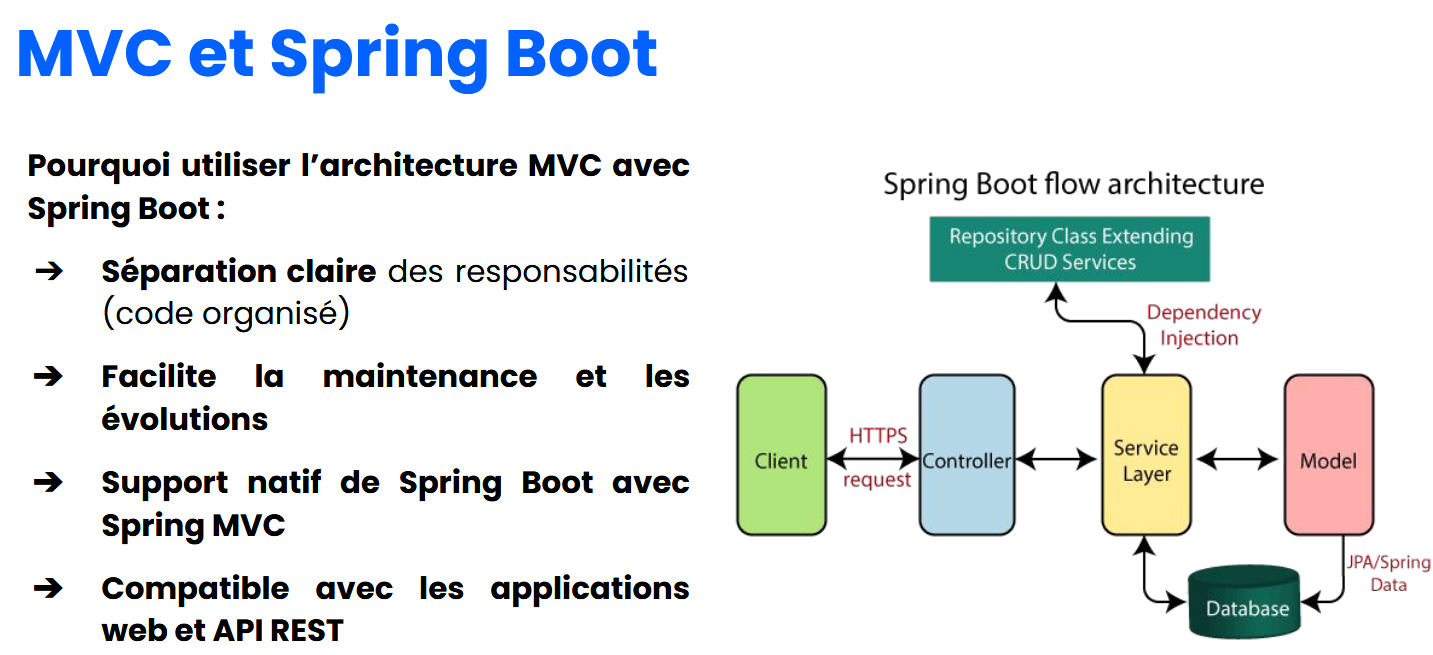
Le modèle MVC aide à structurer une application web en séparant clairement les responsabilités :

* Le Modèle gère les données et la logique métier.
* La Vue s’occupe de la présentation et l’affichage.
* Le Contrôleur fait le lien entre les deux, en orchestrant les flux de données.

Cela rend le code plus clair, modulaire et maintenable, et permet à plusieurs développeurs de travailler en parallèle (ex. un développeur backend sur le modèle, un développeur frontend sur la vue).







**Question :** Quelle est la différence entre Thymeleaf et les templates HTML classiques ?

* HTML classique :
  + Fichier statique.
  + Le contenu est figé (même chose pour tous les utilisateurs).
  + Pas de logique dynamique côté serveur.
* Thymeleaf :
  + Moteur de template dynamique intégré à Spring Boot.
  + Permet d’injecter des données du contrôleur dans la vue (th:text, th:each, etc.).
  + Supporte la logique conditionnelle (th:if, th:unless) et les boucles (th:each).
  + Garde une syntaxe valide HTML (peut être ouvert dans un navigateur même sans serveur).

En résumé :

* HTML classique = statique
* Thymeleaf = HTML enrichi, dynamique, piloté par Spring Boot

**Question**: Comment pouvez-vous passer des données d'un contrôleur à une vue dans Spring ?

Dans Spring Boot, on utilise l’objet **Model** (ou ModelMap, ou ModelAndView).

On ajoute des attributs avec model.addAttribute("nom", objet).

Ces attributs deviennent accessibles dans la **vue Thymeleaf** via ${nom}.

**Question :** Comment Spring facilite-t-il la gestion des formulaires ?

Spring Boot + Spring MVC fournissent une liaison automatique des données de formulaire aux contrôleurs.

* Avec @RequestParam, tu récupères directement la valeur d’un champ.
* Avec @ModelAttribute ou des DTO/objets, on peut lier tout un formulaire à une classe Java automatiquement.
* Thymeleaf s’intègre parfaitement avec Spring MVC :
  + th:action="@{/url}" → définit la route du formulaire.
  + th:field="\*{attribut}" → lie les champs aux propriétés d’un objet Java.

Résultat : pas besoin d’écrire de parsing manuel → Spring fait le lien entre HTML form ↔ Java object.

**Question :** Comment Spring permet-il la validation des données du formulaire ?

1. Bean Validation (javax.validation / jakarta.validation) : on ajoute des annotations comme @NotBlank, @Min, @Max, @Email, etc.
2. @Valid / @Validated dans le contrôleur : déclenche automatiquement la validation de l’objet formulaire.
3. BindingResult : contient les erreurs si la validation échoue.
4. Thymeleaf s’intègre avec Spring MVC pour afficher facilement les erreurs via th:errors.

Résultat : on n’a pas besoin de parser ou valider les champs manuellement → Spring gère tout le cycle validation ↔ retour utilisateur.

**Jour 03 :**

**Question :** Qu'est-ce que JPA et comment facilite-t-il l'accès aux bases de données ?

JPA = une spécification qui définit comment mapper des objets Java à des tables.

Hibernate/EclipseLink = implémentations de JPA.

Spring Data JPA = couche supplémentaire qui automatise le CRUD et réduit le code.

Pourquoi JPA + Spring Data JPA facilite l’accès aux bases de données ?

* Plus besoin d’écrire du SQL manuel pour les opérations simples (CRUD).
* Mapping automatique des objets Java ↔️ tables SQL.
* Gestion simplifiée des transactions et du cycle de vie des entités.
* Extensible : on peut toujours écrire des requêtes JPQL ou SQL natives si besoin.

**Question :** Pourquoi utiliser une base en mémoire comme H2 en développement ?

* Elle est rapide car elle tourne directement en mémoire RAM.
* Elle est jetable : la base est recréée à chaque démarrage → pratique pour tester.
* Elle ne nécessite aucune installation (pas besoin de MySQL/Postgres).
* Elle permet de simuler rapidement une vraie base relationnelle pendant le développement ou les tests.

**Question :** Quelle est l’importance des annotations, telles que @Entity, dans le contexte de JPA ?

* Elles permettent à JPA de comprendre comment une classe Java doit être reliée à une table SQL.
* Sans @Entity, JPA ne sait pas que la classe doit être persistée en base.
* Les annotations définissent aussi les règles de mapping :
  + @Id → clé primaire
  + @Column → personnalisation des colonnes
  + @Table → personnalisation du nom de la table
* Bref, elles permettent de transformer une simple classe Java en une entité persistante utilisable avec Spring Data JPA.

**Question :** Comment Spring Data facilite-t-il la création de requêtes de base de données ?

* Il fournit **JpaRepository**, qui expose déjà toutes les opérations CRUD (save, findAll, deleteById, etc.) → pas besoin de les coder.
* Il analyse automatiquement le **nom des méthodes** (findByName, findByAgeGreaterThan, etc.) et génère la requête correspondante.
* Il permet aussi d’écrire des requêtes personnalisées avec l’annotation @Query si besoin.

**Question :** Comment pouvez-vous créer et lire des entités avec Spring Data JPA ?

Créer : save(entity) ou saveAll(list)

Lire : findAll(), findById(id), ou méthodes personnalisées comme findByName()

Spring Data JPA gère tout le SQL derrière, on travaille uniquement avec des objets Java.