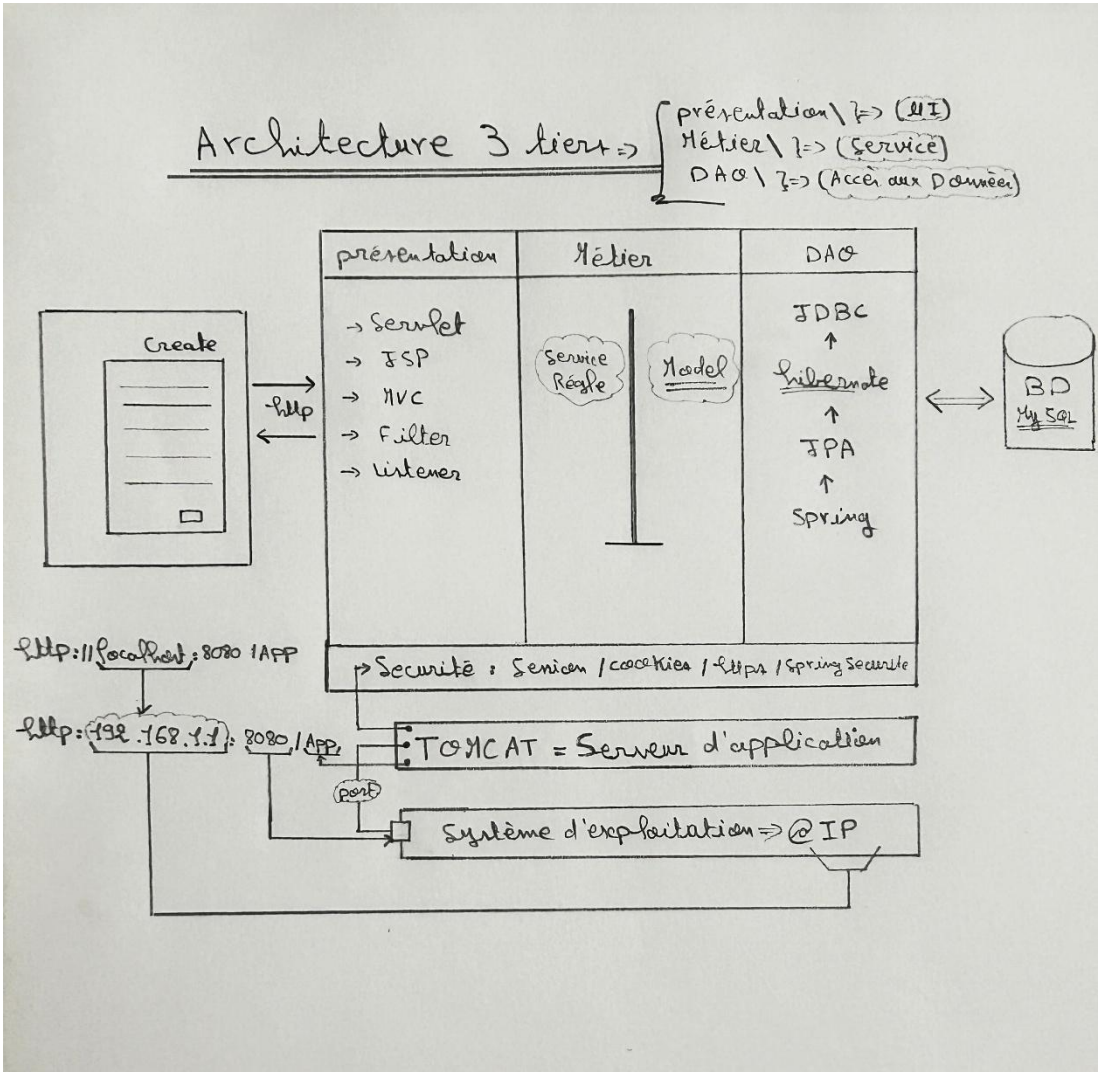
JEE – Java Enterprise Edition

Définition : Plateforme Java conçue pour développer des applications d'entreprise, principalement orientées web.Version (1->7) JAVA ,(8->10) JAKARTA

Desktop (JSE) vs Web (JEE)

Application	Format	Cible	Architecture
Desktop (JSE)	.JAR	Application locale (desktop)	2 tiers (User +BD)
Web (JEE)	.WAR	Application web (serveur web, navigateur)	3 tiers (User+3Couches+BD)

Architecture Web 3 Tiers – JEE (By khayati)



1. Couche Présentation(Interface utilisateur & contrôle)

⇨ Gère l'interface utilisateur. Reçoit les requêtes (Servlet), affiche les réponses (JSP).

- **Servlet** : Contrôleur qui reçoit les requêtes, appelle la logique métier, et redirige vers la vue (JSP).
- **JSP :** Vue qui génère le HTML affiché au client, souvent utilisée pour présenter les données.<%-------code java --------%>

- **JSTL :** Bibliothèque de balises pour JSP, permet l'affichage dynamique avec ${} sans code Java. .<C :-------code java -------->
- **MVC** : Architecture qui sépare les responsabilités :
  - ✓ Model = données (POJO),
  - ✓ View = JSP,
  - ✓ Controller = Servlet.
- **Filter :** Intercepte les requêtes pour faire des vérifications (authentification, logs) avant la Servlet.
- **Listener :** Écoute les événements (démarrage appli, session, requêtes) pour exécuter du code automatiquement.

2. Couche Métier(Logique métier)

⇨ Contient la logique de l'application (Services). Applique les règles, traite les données.

- **Service =** Implémente la logique métier (vérifications, règles, appels DAO…)
- **Modele =** Représente les objets métier (ex : Article, Utilisateur, etc.)

3. Couche DAO(Accès aux données)

⇨ Gère l'accès à la base de données. Effectue les opérations SQL via JDBC, JPA, Hibernate,

JDBC – Communication manuelle avec la base de données

1. Connexion à la base
   - On fournit l'URL de la base, l'utilisateur et le mot de passe pour établir la connexion.
2. Écriture manuelle des requêtes SQL
   - Le développeur écrit lui-même les requêtes SQL (ex : SELECT, INSERT, etc.).
3. Exécution de la requête
   - Utilisation de PreparedStatement pour définir les paramètres .Résultats récupérés via un ResultSet.
4. Fermeture des ressources
   - Obligatoire de fermer : ResultSet, PreparedStatement, puis Connection, afin d'éviter les fuites mémoire**.**

Avantage : Donne un contrôle total sur les requêtes SQL et la gestion fine de la base de données.

Inconvénient : Nécessite beaucoup de code répétitif et est source d'erreurs.

Hibernate – Implémentation ORM de JPA

Fonctionnement :

1. Mapping entre les objets Java et la base de données
   - Utilise des annotations comme @Entity, @Table, @Column, etc., pour faire correspondre les classes Java aux tables SQL.
2. Génération automatique des requêtes SQL
   - Lorsque vous appelez save(), get(), persist(), etc., Hibernate génère automatiquement les requêtes SQL nécessaires.
3. Cache intégré
   - Hibernate utilise un système de cache (1er et 2e niveau) pour optimiser les performances et éviter des requêtes inutiles.
4. Gestion de la session Hibernate
   - SessionFactory est créée une fois au démarrage de l'application.
   - Session est utilisée pour interagir avec la base (analogue à EntityManager).
5. Connexion à la base via JDBC
   - Hibernate utilise en interne JDBC pour se connecter à la base, exécuter les requêtes et retourner les résultats.

Avantage : Implémentation JPA puissante avec support avancé des relations et du cache.
Inconvénient : Complexe à configurer et parfois difficile à déboguer à cause des automatismes.

JPA – Java Persistence API

Fonctionnement :

1. Mapping objet ↔ base de données via annotations(Ex :@Entity indiquer une table.)
2. Gestion automatique des requêtes
    - Pas de SQL à écrire : on utilise des méthodes comme persist(), find(), remove().
    - L'implémentation JPA (comme Hibernate) Exécuter les vraies requêtes SQL via JDBC.
3. Configuration via **persistence.xml**
    - Contient :
        - Le nom de la persistence-unit.
        - **Le dialecte** SQL utilisé (ex : MySQL, PostgreSQL).
        - Les infos de connexion JDBC.
        - La liste des classes entités (<class>MonEntite</class>).
4. Gestionnaires importants :

- EntityManagerFactory : Créé une seule fois au démarrage à partir de persistence.xml. Il fabrique les EntityManager.
- EntityManager : Utilisé pour appeler persist(), find(), etc. — il est l'interface principale pour les opérations sur la base.outil qui communique avec BD

Avantage : Simplifie la gestion des entités et des opérations CRUD avec des annotations.
Inconvénient : Moins de contrôle pour les requêtes complexes et dépendant du framework sous-jacent.

Spring Data + JPA + Hibernate + JDBC – Communication intégrée

Étapes de fonctionnement en cascade :

1. Appel par la couche métier
    - Le service appelle les méthodes Spring Data (save(), findAll(), deleteById(), etc.).
2. Spring Data délègue à JPA
    - Ne contient pas de logique SQL, mais transmet les appels à l'API JPA.
3. JPA utilise Hibernate
    - Hibernate fait le lien entre les objets Java et les tables SQL (ORM).
    - Génère auto les requêtes SQL selon les annotations ou fichiers de mapping.
4. Hibernate utilise JDBC
    - JDBC est responsable d'établir la connexion et d'exécuter techniquement les requêtes SQL dans la base.
5. Retour des données
    - Les résultats de la base suivent le chemin inverse :
        - JDBC → Hibernate → JPA → Spring Data → Service métier
    - Le service traite éventuellement les données et les retourne au contrôleur

Avantage : Gère automatiquement la couche DAO avec peu de code, via des interfaces.
Inconvénient : Moins flexible pour des traitements personnalisés ou non standards.

ORM – Fonctionnement général

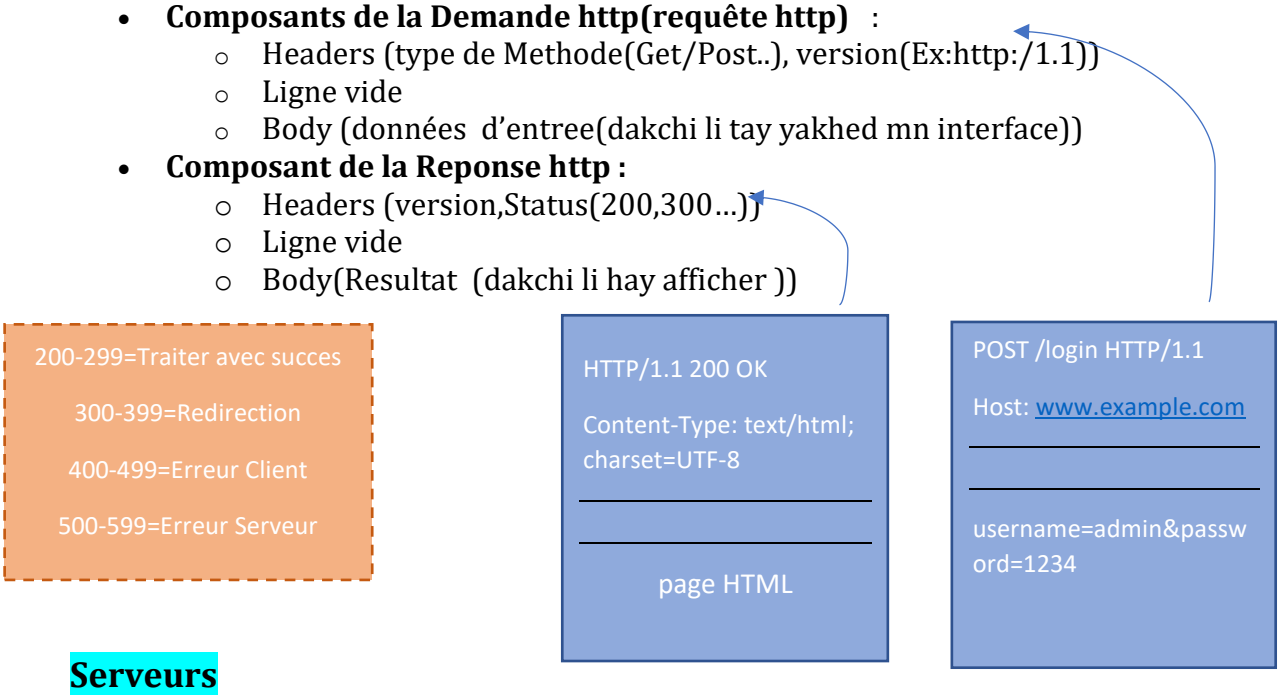
1. Correspondance automatique entre classes Java et tables
   - Fait via annotations comme @Entity, @Table, @Column (ou via un fichier orm.xml).
   - Correspondance :
     - Classe Java ↔ Table
     - Attribut Java ↔ Colonne SQL
2. Génération automatique des requêtes
   - Ex. : entityManager.persist(monObjet) génère automatiquement :
   - Plus besoin d'écrire du SQL manuellement — l'ORM le fait à ta place.

Avantage : Automatise le mapping objet-relationnel, réduisant le besoin d'écrire du SQL.
Inconvénient : Moins performant ou optimisé pour les requêtes très spécifiques.

– Détails

Protocole HTTP

- **GET** : lecture ,**POST** : envoi (formulaire) ,**PUT/DELETE** : modification/suppression
- **Composants de la Demande http(requête http)** :
  - Headers (type de Methode(Get/Post..), version(Ex:http:/1.1))
  - Ligne vide
  - Body (données d'entree(dakchi li tay yakhed mn interface))
- **Composant de la Reponse http :**
  - Headers (version,Status(200,300…))
  - Ligne vide
  - Body(Resultat (dakchi li hay afficher))



Serveurs

- **Tomcat** : léger, le plus courant
- **WildFly / JBoss** : complet
- **GlassFish** : JEE officiel
- **Jetty** : embedded, très léger

Cycle de vie d'une Servlet

1. **Chargement :** Le conteneur charge la classe servlet
2. **Instanciation :** Cr´eation d'une instance de la servlet
3. **Initialisation :** Appel de la m´ethode init()
4. . **Traitement des requêtes (service())** : Pour chaque requête, la méthode service() est appelée, qui dispatch vers doGet(), doPost(), etc.
5. **Destruction :** Quand le serveur arrête la servlet, il appelle destroy() pour libérer les ressources.

Methode Servlet

Par Annotation => @WebServlet("/creer")=> Moderne, rapide

Par web.xml => via <servlet> + <mapping> => Ancienne, config centralisée

AV (++)Les servlets sont rapides, portables et offrent un contrôle précis sur les requêtes HTTP.
INC (---)Elles sont verbeuses, peu adaptées à la vue et difficiles à maintenir dans les grandes applications.

Dependences

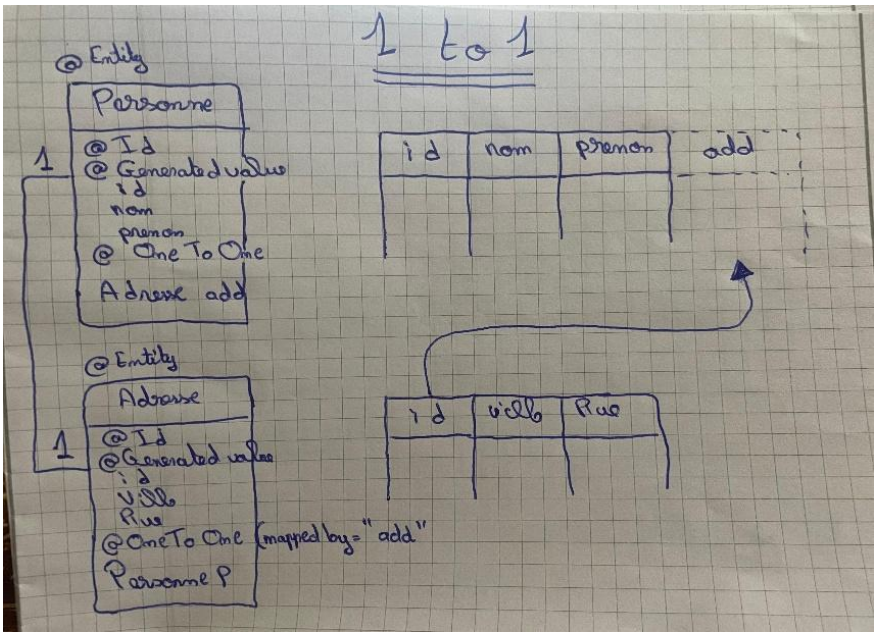
- **Maven** = On ajoute manuellement les dépendances dans le fichier pom.xml
- **SpringBoot =** On ajoute les starters Spring dans pom.xml (ou build.gradle), et Spring Boot gère automatiquement les versions.

Dépendance	Utilité
Spring Web	Créer des applications web REST ou MVC (contrôleurs, routes HTTP, etc.).
Spring Validation	Valider les données (formulaires, objets) avec des annotations (@NotNull, etc.).
Thymeleaf	Moteur de template HTML pour générer des vues dynamiques côté serveur.
Lombok	Réduit le code boilerplate (getters, setters, constructeurs, etc.) via annotations.
MySQL Driver	Permet à l'application Java de se connecter à une base de données MySQL.
Spring Data JPA	Simplifie l'accès aux données avec des interfaces sans écrire de SQL.

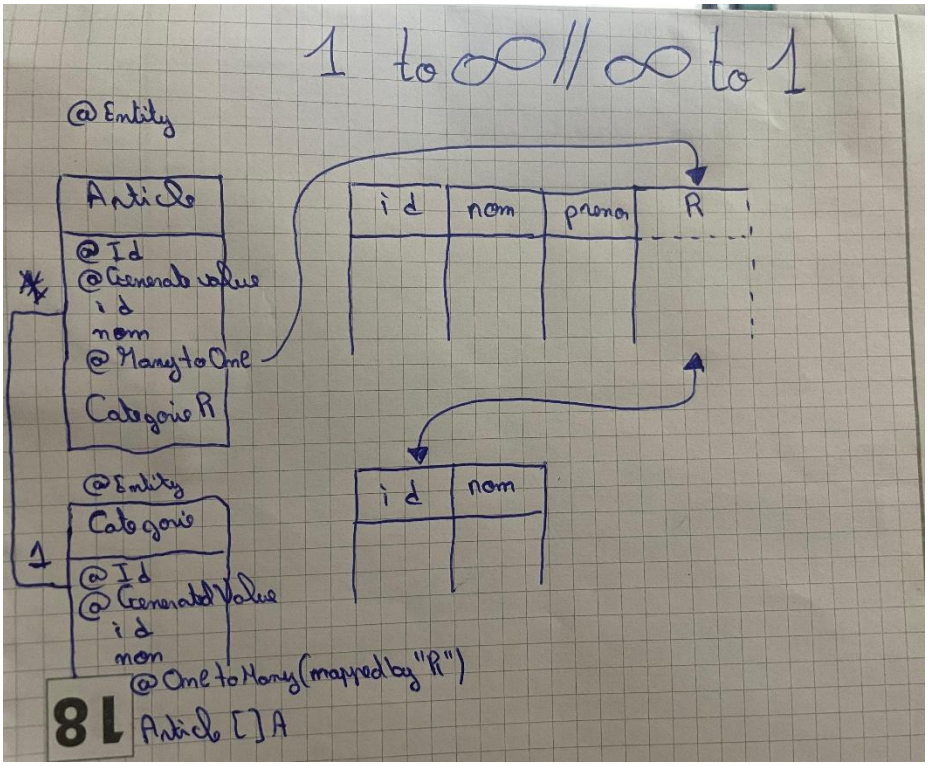
ORM – Détails

Mapping ORM

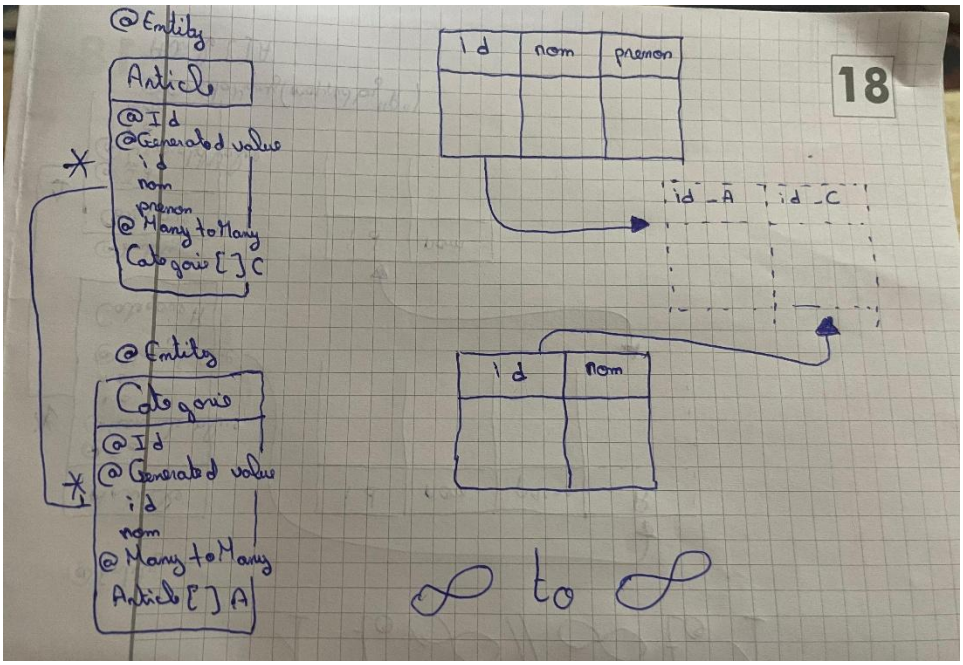
- **OneToOne :** Lie deux entités ayant une relation exclusive (une instance ↔ une seule autre),avec une clé étrangère.



- **ManyToOne :** Plusieurs entités enfant sont reliées à une seule entité parent via une clé étrangère.
- **OneToMany :** Une entité parent possède une collection d'entités enfants (inverse de ManyToOne).



- **ManyToMany :** Deux entités sont liées via une table d'association intermédiaire contenant leurs identifiants respectifs.

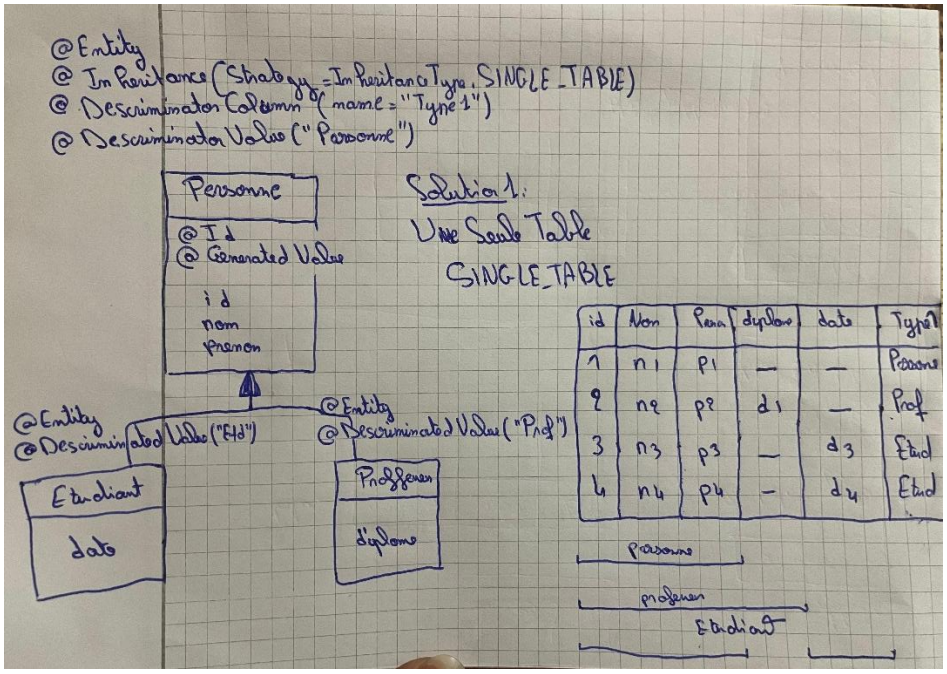


Héritage :

1. Single Table (@Inheritance(strategy = SINGLE_TABLE))

Toutes les classes de l'héritage sont stockées dans une seule table, avec une colonne discriminante.

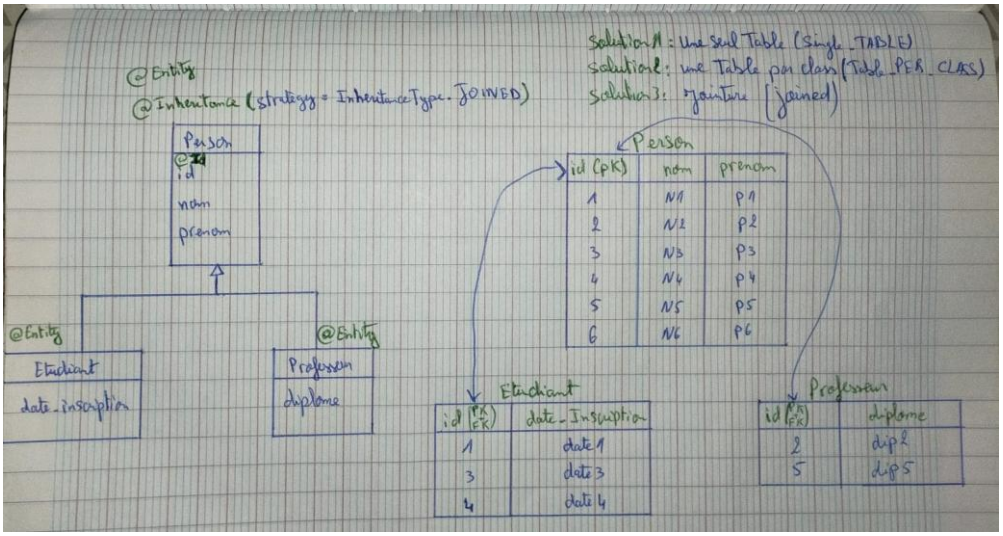
- **Avantages :** Performant (une seule table, pas de jointure) et simple à requêter.
- **Inconvénients :** Beaucoup de colonnes nulles si les sous-classes ont des attributs très différents.



2. Joined (@Inheritance(strategy = JOINED))(By MDD)

Chaque classe a sa propre table, liées par jointure sur la clé primaire.

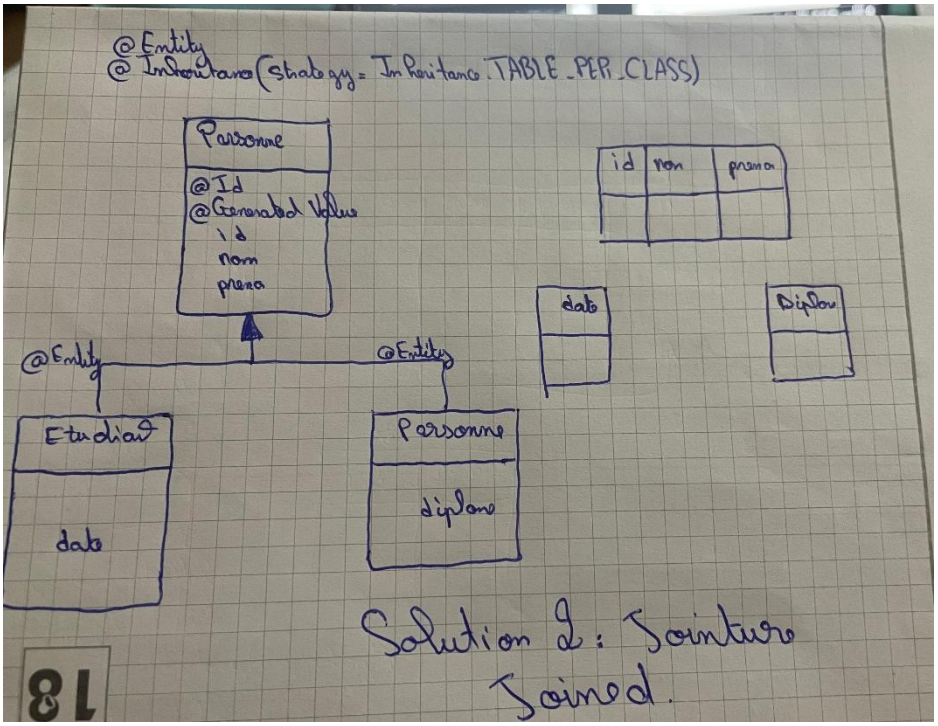
- **Avantages :** Structure de base propre, pas de colonnes inutilisées.
- **Inconvénients :** Requêtes plus lentes à cause des jointures.



3. Table Per Class (@Inheritance(strategy = TABLE_PER_CLASS))

Chaque classe a sa propre table complète, sans jointure.

- **Avantages** : Pas de jointures, structure claire pour chaque type.
- **Inconvénients :** Redondance de colonnes, difficile à requêter sur la classe parente.



Annotation

Annotations JPA / Hibernate

Annotation	Rôle
@Entity	Déclare la classe comme une entité persistante (mappée à une table).
@Table(name="...")	Spécifie le nom de la table en base de données.
@Id	Désigne un champ comme clé primaire.
@GeneratedValue	Configure la génération automatique de l'ID (auto-incrément, séquence).
@Column(name="...")	Définit le nom et éventuellement des contraintes pour une colonne.
@OneToOne,@OneToMany, @ManyToOne, @ManyToMany	Définit les relations entre entités.
@JoinColumn	Spécifie la colonne utilisée comme clé étrangère dans une relation.

Annotations Spring Framework

Annotation	Rôle
@Component	Marque une classe comme composant géré par Spring (scope singleton).
@Service	Spécialisation de @Component pour la couche métier.
@Repository	Spécialisation de @Component pour la couche DAO (gestion des exceptions).
@Controller	Classe contrôleur pour les applications web MVC.
@RestController	Combine @Controller + @ResponseBody (retourne des données JSON/XML).
@GetMapping, @PostMapping	Mappe des requêtes HTTP (GET, POST, etc.) vers des méthodes.
@SpringBootApplication	Active l'auto-configuration Spring Boot (classe principale).

Annotations Servlet / JEE Web

Annotation	Rôle
@WebServlet("/chemin")	Déclare un servlet sans configuration web.xml.
@WebFilter	Crée un filtre pour intercepter les requêtes/réponses HTTP.
@WebListener	Permet d'écouter les événements (démarrage, arrêt, etc.) de l'application.

Services Web SOAP dans JEE (Java EE)

⇨ SOAP (*Simple Object Access Protocol*) est un protocole basé sur XML pour échanger des données structurées dans des services web. Il fonctionne généralement sur HTTP/HTTPS et est utilisé dans les architectures orientées services (SOA).

Composants clés :

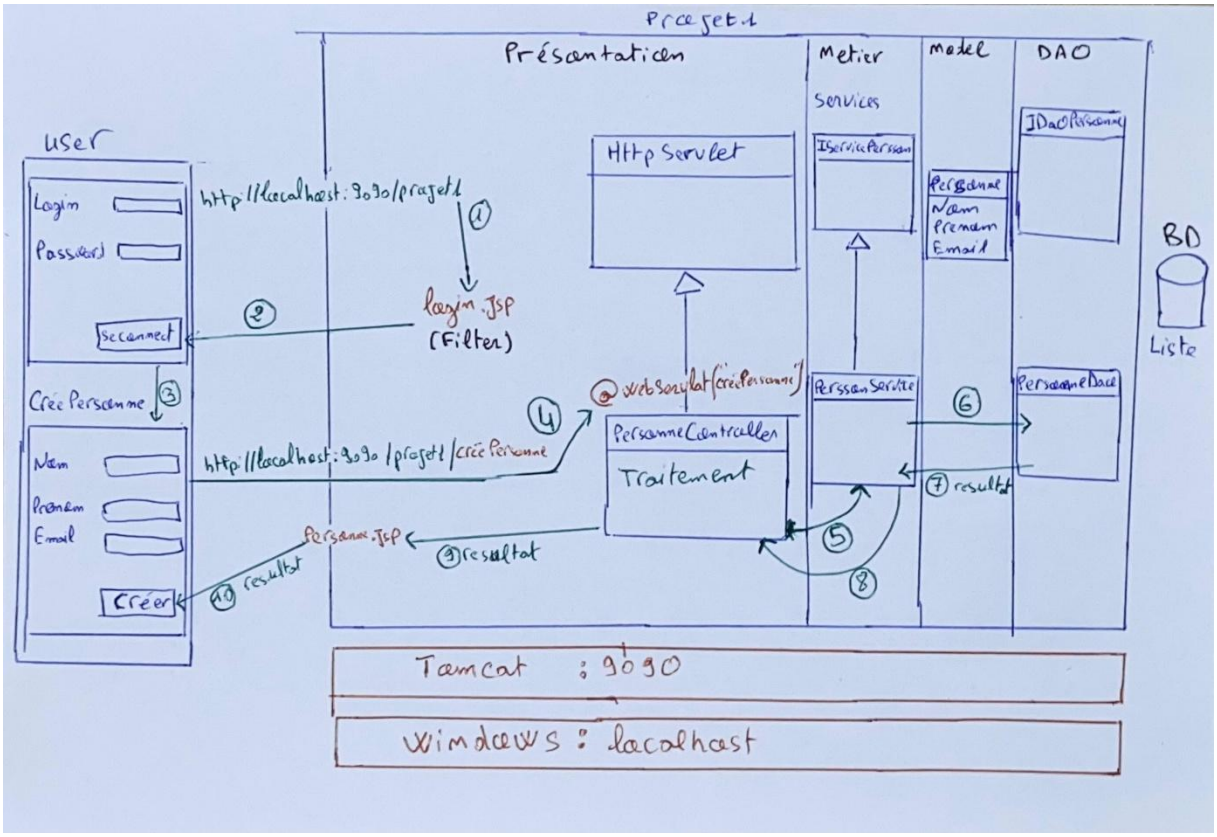
Composant	Rôle
SOAP (*Simple Object Access Protocol*)	Protocole de messagerie XML pour les échanges entre clients et services web. Il définit un format d'enveloppe (*envelope*) pour encapsuler les requêtes/réponses.
WSDL (*Web Services Description Language*)	Fichier XML qui décrit l'interface du service web (méthodes disponibles, types de données, URL d'accès, etc.). C'est le "contrat" entre le client et le serveur.
UDDI (*Universal Description, Discovery, and Integration*)	Annuaire centralisé (comme un registre) permettant de publier et découvrir des services web SOAP. Moins utilisé aujourd'hui au profit de REST.

(+++)Standardisé, sécurisé et fiable (idéal pour les transactions critiques).

(----)Plus lourd et complexe que REST (XML verbeux, configuration WSDL).

Scénario complet : Création d'une Personne

(By ECH-CHAOUI)



Étapes détaillées du scénario :

1. Interface utilisateur (User) – Requête HTTP

- L'utilisateur saisit son login/mot de passe et envoie une requête HTTP via le navigateur :
- http://localhost:9090/projet
- Cette requête est interceptée par un Filter.

2. Filter – Contrôle d'accès

Le Filter vérifie l'authentification :

- S'il n'y a pas de session valide, il renvoie la page login.jsp.
- S'il est authentifié, il autorise l'accès à la suite.

3. Interface – Création d'une personne

L'utilisateur clique sur "Créer Personne", ce qui déclenche une nouvelle requête HTTP :
http://localhost:9090/projet/creerPersonne

4. Servlet (Contrôleur) – PersonneController

- Cette servlet hérite de HttpServlet, elle reçoit les données du formulaire (nom, prénom, email), via doPost().

- Elle transmet ces données à la couche métier.

5. Couche Métier – PersonneService

Le PersonneService applique les règles de gestion :

- Vérifie les champs obligatoires (ex : nom non vide),Valide les formats (ex : email)

S'il n'y a pas d'erreur, il appelle la couche DAO pour enregistrer la personne.

6. Couche DAO – PersonneDao

Le DAO est responsable de la communication directe avec la base de données :

- Il prépare les requêtes SQL (JDBC) ou utilise un ORM (JPA/Hibernate).
- Il enregistre les données dans la base.

7. Base de données – Réponse

- La base confirme l'enregistrement (succès ou échec).
- Le DAO renvoie ce résultat à la couche métier.

8. Couche Métier → Contrôleur

- Le service retourne le résultat (succès ou erreur) au contrôleur PersonneController.

9. Contrôleur → Vue

- Le contrôleur prépare les données à afficher (objet Java ou liste de résultats) et les transmet à une page JSP.

10. Présentation – JSP + JSTL

- La page JSP génère la vue HTML dynamique.
- Les balises JSTL affichent les données (${personne.nom} etc.)

11. Retour HTTP vers l'utilisateur

- La vue est renvoyée au client sous forme de réponse HTTP, avec le rendu HTML dans le navigateur.

Résumé du flux complet