



Département Télécommunications, Réseaux & Informatique

Ecole Nationale des Sciences Appliquées d’El Jadida

Université Chouaib Doukkali

**RAPPORT JEE**

Filière :

**Ingénierie des Systèmes d'Information et de Communication (ISIC)**

2

ème

année Cycle Ingénieur

**TP 3-2** *« TP 2 Hibernate (maven) »*

*Réalisé le*

13 / 10 / 2024

***Réalisé par :***

**ALIOUI** Jaâfar

***Encadré par :***

Prof. **LACHGAR** Mohammed

Année universitaire : 2024/2025

Remerciements

Je tiens à exprimer ma sincère gratitude envers le professeur M.LACHGAR pour l'opportunité qui ma a été offerte de travailler sur cet intéressant projet JEE. Sa confiance en ma capacité à relever ce défi a été une source d'inspiration et de motivation tout au long du processus.

Résumé

Dans ce projet, nous avons développé trois exercices distincts dans le cadre de la gestion des données en utilisant JPA/Hibernate. Chaque exercice abordait une problématique spécifique :

1. **Exercice 1** portait sur la gestion basique des produits, avec des opérations CRUD simples sur la classe Produit.
2. **Exercice 2** ajoutait des relations entre les classes Produit, Categorie, et Commande, avec une relation ManyToMany gérée à travers la classe LigneCommandeProduit.
3. **Exercice 3 (Travail à rendre)** consistait à développer une application de gestion de l'état civil avec des relations complexes ManyToMany entre les entités Homme, Femme, et Mariage, et incluait des requêtes avancées avec des opérations de filtrage et d'analyse des données.

Lien du repo Github : [https://github.com/Jaafarhh/TPs-JEE/tree/main/TP3](https://github.com/Jaafarhh/TPs-JEE/tree/main/TP2)

Abstract

This project involves the development of three distinct exercises within the scope of data management using JPA/Hibernate:

1. **Exercise 1** focused on basic product management, with simple CRUD operations on the Produit class.
2. **Exercise 2** introduced relationships between the Produit, Categorie, and Commande classes, with a ManyToMany relationship managed through the LigneCommandeProduit association class.
3. **Exercise 3 (Final Assignment)** consisted of developing a civil registry management application, with complex ManyToMany relationships between the Homme, Femme, and Mariage entities, and advanced queries for filtering and analyzing data.

Github repo link : <https://github.com/Jaafarhh/TPs-JEE>

Table des matières

[Remerciements ii](#_Toc179977999)

[Résumé iii](#_Toc179978000)

[Abstract iv](#_Toc179978001)

[Table des matières v](#_Toc179978002)

[Liste des figures vii](#_Toc179978003)

[1 TP3 : Exercice 1 9](#_Toc179978004)

[Introduction 9](#_Toc179978005)

[1.1 Création du projet 9](#_Toc179978006)

[1.2 Étapes de développement 9](#_Toc179978007)

[1.2.1 Création de l’entité Produit 9](#_Toc179978008)

[1.2.2 Création de la Base de Données 10](#_Toc179978009)

[1.2.3 Fichier de Configuration Hibernate 10](#_Toc179978010)

[1.2.4 Classe HibernateUtil 11](#_Toc179978011)

[1.2.5 Interface IDao 11](#_Toc179978012)

[1.2.6 Classe ProduitService 12](#_Toc179978013)

[1.2.7 Classe de Test 13](#_Toc179978014)

[1.3 Conclusion 14](#_Toc179978015)

[2 TP3 : Exercice 2 15](#_Toc179978016)

[Introduction 15](#_Toc179978017)

[2.1 Création du projet 15](#_Toc179978018)

[2.2 Etapes de développement 15](#_Toc179978019)

[2.2.1 Partie A : Couche persistance 15](#_Toc179978020)

[2.2.2 Partie B : Couche Service 20](#_Toc179978021)

[2.2.3 Tests 23](#_Toc179978022)

[2.3 Conclusion 24](#_Toc179978023)

[3 TP3 : Travail à Rendre 25](#_Toc179978024)

[Introduction 25](#_Toc179978025)

[3.1 Création du projet 25](#_Toc179978026)

[3.2 Etapes de développement 25](#_Toc179978027)

[3.2.1 Partie A : Couche persistance 25](#_Toc179978028)

[3.2.2 Partie B : Couche Service 29](#_Toc179978029)

[3.2.3 Tests 32](#_Toc179978030)

[3.3 Conclusion 33](#_Toc179978031)

[4 Conclusion générale 34](#_Toc179978032)

Liste des figures

[Figure 1 : *Librairies importées* 9](#_Toc179977962)

[Figure 2 : *Classe Produit* 10](#_Toc179977963)

[Figure 3 : *fichier de configuration hibernate.cfg* 11](#_Toc179977964)

[Figure 4 : *classe HibernateUtil* 11](#_Toc179977965)

[Figure 5 : *Interface IDao* 11](#_Toc179977966)

[Figure 6 : *Classe ProduitService* 12](#_Toc179977967)

[Figure 7 : *Classe test (1)* 13](#_Toc179977968)

[Figure 8 : *Methode FindByDate implementee dans la classe ProduitService* 14](#_Toc179977969)

[Figure 9 : *classe Test (2)* 14](#_Toc179977970)

[Figure 10 : *Classe ma.projet.classes.Commande* 16](#_Toc179977971)

[Figure 11 : *Classe ma.projet.classes.Categorie* 16](#_Toc179977972)

[Figure 12 : *Classe ma.projet.classes.Produit* 17](#_Toc179977973)

[Figure 13 : *Classe ma.projet.classes.LigneCommandeProduit* 18](#_Toc179977974)

[Figure 14 : *Classe ma.projet.classes.CommandeProduitPK* 18](#_Toc179977975)

[Figure 15 : *Le fichier de configuration hibernate.cfg.xml* 19](#_Toc179977976)

[Figure 16 : *Classe ma.projet.util.HibernateUtil* 19](#_Toc179977977)

[Figure 17 : *Interface ma.projet.dao.IDao* 20](#_Toc179977978)

[Figure 18 : *Classe ma.projet.service.CategorieService* 20](#_Toc179977979)

[Figure 19 : *Méthode findByCategorie* 21](#_Toc179977980)

[Figure 20 : *Méthode findByDate* 21](#_Toc179977981)

[Figure 21 : *Méthode afficherProduitsCommande* 22](#_Toc179977982)

[Figure 22 : *Méthode produitsPrixSuperieur* 22](#_Toc179977983)

[Figure 23 : *Classe ma.projet.test.Test* 23](#_Toc179977984)

[Figure 24 : *Classe ma.projet.beans.Personne* 26](#_Toc179977985)

[Figure 25 : *Classe ma.projet.beans.Femme* 26](#_Toc179977986)

[Figure 26 : *Classe ma.projet.beans.Homme* 26](#_Toc179977987)

[Figure 27 : *Classe ma.projet.beans.Mariage* 27](#_Toc179977988)

[Figure 28 : *Classe ma.projet.beans.HommeFemmePK* 27](#_Toc179977989)

[Figure 29 : *Le fichier de configuration hibernate.cfg.xml* 28](#_Toc179977990)

[Figure 30 : *Classe ma.projet.util.HibernateUtil* 28](#_Toc179977991)

[Figure 31 : *Interface ma.projet.dao.IDao* 29](#_Toc179977992)

[Figure 32 : *Classe ma.projet.service.Mariage* 29](#_Toc179977993)

[Figure 33 : *Methode afficherEpouses* 30](#_Toc179977994)

[Figure 34 : *Methode getChildrenCount* 30](#_Toc179977995)

[Figure 35 : *Methode getWomenMarriedTwiceOrMore* 31](#_Toc179977996)

[Figure 36 : *Methode afficherMariages* 31](#_Toc179977997)

[Figure 37 : *Classe ma.projet.test.Test* 32](#_Toc179977998)

1. TP3-2 : Hibernate ( Maven )
   1. Introduction

Le projet vise à illustrer l'utilisation d'Hibernate, un framework de persistance en Java, pour la gestion d'une base de données relationnelle MySQL. L'objectif est de simplifier l'intégration entre l'application Java et la base de données en fournissant une couche d'abstraction qui atténue la complexité des opérations CRUD. L'application développée dans ce projet implémente un système de gestion simplifié où les entités de la base de données sont associées aux classes Java grâce aux annotations fournies par Hibernate.

* 1. Outils Utilisés

Pour ce projet, plusieurs outils et technologies ont été employés afin de garantir une gestion efficace de la base de données ainsi que le développement de l'application Java :

1. IntelliJ IDEA : Environnement de développement intégré (IDE) qui s'intègre facilement avec Maven, Java et Hibernate, offrant un cadre robuste pour écrire, déboguer et exécuter du code.
2. JDK 1.8 (Java 8) : Cette version stable et largement adoptée est utilisée pour le développement d'applications d'entreprise.
3. MySQL : Système de gestion de base de données relationnelle servant à stocker les données de l'application. L'interface phpMyAdmin facilite la gestion des tables et des données à travers une interface web.
4. Maven : Outil crucial pour la gestion de projet et des dépendances, permettant l'intégration de bibliothèques tierces comme Hibernate et le connecteur MySQL.
   1. Configuration du fichier pom.xml

Maven simplifie l'intégration des bibliothèques requises pour Hibernate et MySQL à travers le fichier pom.xml, qui définit les dépendances à télécharger et intégrer automatiquement dans le projet :

* Hibernate : Essentiel pour mapper des objets Java aux tables de la base de données, permettant ainsi d'effectuer des opérations SQL.
* MySQL Connector : Pour assurer la communication entre l'application Java et la base de données MySQL via le protocole JDBC.
* Javax.persistence-api : Un API standard JPA qui aidera a implémenter les annotations.
* JUnit : Bibliothèque utilisée pour rédiger et exécuter des tests unitaires afin de valider le comportement de l'application, garantissant ainsi le bon fonctionnement du code.
  1. Création de l’Interface IDao

Nous avons créer une interface générique IDao<T> dans le package dao qui définit les méthodes nécessaires pour réaliser les opérations (Create, Read, Update, Delete) :

* create(T o)
* delete(T o)
* update(T o)
* findById(int id)
* findAll()
  1. Création des Entités
* **Machine** : La classe Machine est initialisée avec des attributs : ID, la référence (ref) et la date d'achat (dateAchat). Elle est associée à l'entité Salle par une relation ManyToOne, indiquant que plusieurs machines peuvent appartenir à une salle. Puis deux requêtes nommées ont été définies : une en HQL et l’autre en native SQL.
* **Salle** : La classe Salle modélise une salle, comprenant les attributs id et code. Elle est liée à plusieurs machines via une relation OneToMany, signifiant qu'une salle peut contenir plusieurs machines. Cette relation est gérée par l'attribut machines, qui utilise le type de chargement EAGER pour récupérer les machines associées à chaque salle.
* **Étape 4 : Configuration de Hibernate**

Le fichier hibernate.cfg.xml configure la connexion JDBC pour Hibernate avec MySQL. Il définit le dialecte MySQL5Dialect, le driver com.mysql.cj.jdbc.Driver, et l'URL de connexion pour la base de données gtsalle sur localhost. Les identifiants de connexion sont configurés avec root comme nom d'utilisateur. Le fichier permet également d'activer l'affichage et le formatage des requêtes SQL et de gérer la mise à jour automatique du schéma de la base.

* **Étape 5 : Configuration d’une Classe Utilitaire pour Hibernate**

La classe HibernateUtil centralise la gestion des sessions Hibernate, initialisant une instance de SessionFactory à partir du fichier de configuration hibernate.cfg.xml. En cas d'erreur lors de la création de la session, une exception est enregistrée et renvoyée. La méthode getSessionFactory() fournit une instance unique de session, garantissant une connexion efficace à la base de données tout au long du cycle de vie de l'application.

* **Étape 6 : Création de la Couche Service**
* **Classe MachineService** : Cette classe implémente l'interface générique IDao<Machine>, permettant la gestion des entités Machine avec des opérations CRUD et une méthode personnalisée pour rechercher des machines entre deux dates.

**Méthodes Principales :**

1. create(Machine o) : Ouvre une session Hibernate pour sauvegarder une entité Machine. En cas d’erreur, une rollback est effectuée.
2. delete(Machine o) : Supprime une entité Machine en utilisant son ID, avec rollback en cas d’échec.
3. update(Machine o) : Met à jour les informations de la machine en base.
4. findById(int id) : Recherche et retourne une machine spécifique par son ID.
5. findAll() : Retourne toutes les machines disponibles dans la base.

**Méthode Personnalisée :**

* findBetweenDate(Date d1, Date d2) : Utilise une requête nommée pour rechercher les machines dont la date d'achat est comprise entre deux dates spécifiques.
* **Classe SalleService** : Suivant une structure similaire à MachineService, cette classe implémente également l'interface générique IDao<Salle> pour fournir des opérations CRUD sur les entités Salle.
* **Étape 7 : Classe de Test pour Générer la Base de Données**

L'objectif de ce test est de vérifier la persistance correcte des entités dans la base de données ainsi que leur récupération, tout en s'assurant que la configuration d'Hibernate fonctionne de manière appropriée.

* **Création et Insertion de Salles** : Deux instances de la classe Salle sont créées avec les codes "A1" et "B2" et sont persistées à l'aide de la méthode create() de la classe SalleService.
* **Création et Insertion de Machines** : Deux instances de la classe Machine sont créées avec les références "M123" et "M124", chacune associée à une salle, puis elles sont également persistées via la méthode create() de MachineService.
* **Récupération et Affichage des Salles et Machines Associées** : Toutes les instances de Salle sont récupérées, et pour chaque salle, les machines associées sont affichées, permettant ainsi de vérifier que la relation "un à plusieurs" fonctionne correctement.
* **Utilisation de la Méthode findBetweenDate** : Cette méthode est utilisée pour récupérer les machines achetées entre deux dates précises, avec affichage des références et des dates d'achat des machines correspondantes.
* **Étape 8 : Mise en Place des Tests Unitaires avec JUnit**

Dans cette étape, des tests unitaires ont été réalisés pour vérifier les différentes méthodes des services liés aux entités Salle et Machine, garantissant le bon fonctionnement des opérations CRUD ainsi que la méthode personnalisée findBetweenDate.

* **Classe MachineServiceTest** : Comprend des méthodes pour préparer et nettoyer les tests, ainsi que pour tester les opérations CRUD.
* **Classe SalleServiceTest** : Suivant la même approche, cette classe valide les opérations CRUD liées à l'entité Salle.

Après l'exécution des tests, les rapports indiquent que tous les tests ont réussi, confirmant le bon fonctionnement des services MachineService et SalleService.

* **Conclusion**

Ce projet a permis d'explorer Hibernate, un framework ORM crucial pour la gestion des bases de données en Java. Grâce à l

4o mini

* **You said:**

continue

* **ChatGPT said:**

ChatGPT

l'implémentation d'une architecture bien définie, comprenant des entités, des services, et des tests unitaires, nous avons pu établir une base solide pour la gestion d'une application de salle et de machine.

Les résultats obtenus témoignent de la capacité d'Hibernate à gérer efficacement les opérations de base de données, en simplifiant les interactions entre le code Java et les tables SQL. En outre, l'utilisation de Maven pour la gestion des dépendances a facilité l'intégration des bibliothèques nécessaires, ce qui a contribué à un développement fluide et structuré.

Les tests unitaires réalisés garantissent non seulement la robustesse de notre application, mais ils offrent également une assurance de qualité, essentielle pour toute application de production. Cette approche axée sur les tests est cruciale pour assurer la maintenabilité et l'évolutivité du code à long terme.

Dans un contexte professionnel, la maîtrise d'Hibernate et des bonnes pratiques de développement, telles que l'utilisation de tests unitaires, constitue un atout considérable. Ce projet a donc été non seulement une opportunité d'apprentissage technique, mais également un pas en avant vers une meilleure compréhension des systèmes de gestion de bases de données dans le développement d'applications Java.

1. Conclusion générale

Ce projet nous a permis d’explorer les bases de la persistance des données à travers trois exercices de difficulté croissante :

1. **Exercice 1** a introduit les opérations basiques de gestion de produits.
2. **Exercice 2** nous a appris à modéliser des relations entre plusieurs entités, avec une implémentation de jointures complexes comme ManyToMany.
3. **Exercice 3** a approfondi l'utilisation des requêtes avancées et des filtres complexes dans une application de gestion de l’état civil. Nous avons mis en place des méthodes pour gérer des opérations plus complexes tout en respectant les normes de performances et d'efficacité. Le projet dans son ensemble a renforcé notre compréhension des meilleures pratiques en matière de gestion des données relationnelles dans une application Java.