****

**Système de gestion d'inventaire avec accès distant**

## Développement Distribué (J2EE)

Bas du formulaire

**HAMZA SEMMAK – YACINE ABANAI**

**Introduction et Objectifs**

**1- Introduction :**

Ce projet est une application de gestion des produits utilisant les technologies Socket pour une architecture client-serveur. Il fournit une plateforme sécurisée pour gérer les produits stockés dans une base de données distante.

**2- Les Objectifs :**

 **Architecture client-serveur** :

* Développer un serveur pour centraliser les opérations d’inventaire.
* Créer un client pour permettre un accès distant aux fonctionnalités.

 **Gestion des produits** :

* Ajouter, modifier, rechercher et supprimer des produits dans la base de données.

 **Accès distant** :

* Implémenter une communication via sockets pour synchroniser les opérations à distance.

 **Base de** données :

* Utiliser MySQL pour stocker et gérer les informations.

 Options supplémentaires :

* Intégrer un système d’authentification pour sécuriser l’accès.
* Ajouter des logs pour suivre les actions effectuées sur l’inventaire.

**Architecture :**

**1- Diagrammes UML :**

Les diagrammes UML fournissent une représentation visuelle de la structure et du comportement du système. Le diagramme de classes illustre les relations entre les différentes entités, tandis que le diagramme de séquence décrit les échanges entre le client et le serveur lors des opérations principales

Diagramme de class :

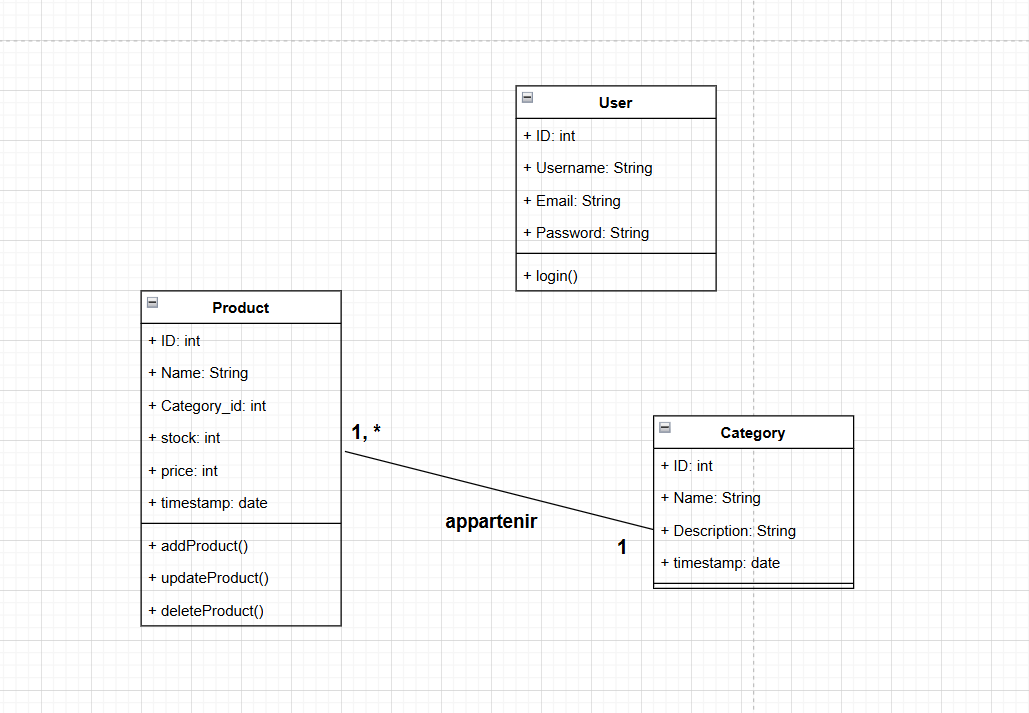
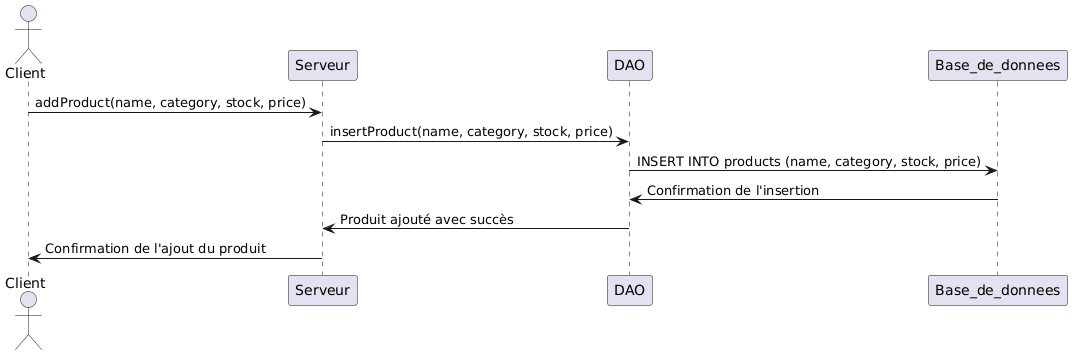
****

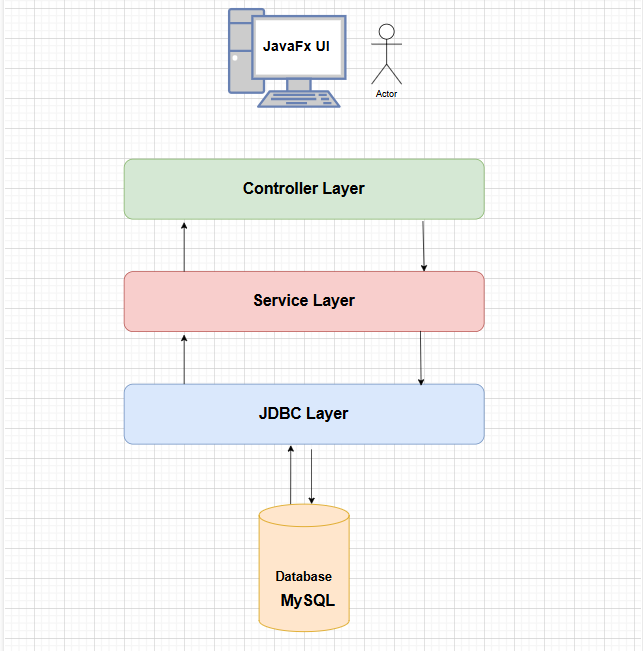
Diagramme de Séquence



**2- Description de l’architecture client-serveur**

L'architecture client-serveur permet au client d'envoyer des requêtes au serveur, qui gère les actions et interagit avec la base de données pour manipuler les produits. Dans ce projet, nous utilisons des **sockets** pour la communication entre les composants, assurant une séparation des responsabilités et une gestion efficace des données.

**Architecture :**



### **Utilisation de Sockets :**

Dans cette application, les **sockets** sont utilisés pour établir une communication fiable entre le client et le serveur, permettant l'échange de données en temps réel. Le serveur écoute sur un port spécifique et attend les connexions des clients, qui envoient des requêtes pour effectuer des opérations sur l'inventaire.

Le serveur reçoit les requêtes des clients, les traite et interagit avec la base de données pour exécuter les actions demandées. Par exemple, lorsqu'un client souhaite ajouter un produit, la requête est transmise via un socket au serveur, qui utilise les classes DAO (Data Access Object) pour insérer les données dans la base MySQL.

Ce choix garantit une communication efficace et maintient une séparation claire entre les couches de service et de persistance.

### **Structure de la Base de Données :**

La structure de la base de données a été conçue pour assurer une organisation efficace et une gestion optimale des données nécessaires à l'application. Voici une description des principales tables :

1. **Table categories** :
   * Contient les informations sur les catégories de produits, telles que leur nom et une description.
   * La clé primaire est id, qui s'incrémente automatiquement.
2. **Table products** :
   * Stocke les détails des produits, y compris leur nom, catégorie associée, stock disponible, et prix.
   * La relation entre les produits et leurs catégories est définie par une clé étrangère category\_id, qui fait référence à la table categories. La contrainte ON DELETE CASCADE garantit que les produits associés à une catégorie sont supprimés automatiquement si la catégorie est supprimée.
3. **Table users** :
   * Gère les informations des utilisateurs, telles que leur nom d'utilisateur, adresse email, et mot de passe.
   * Chaque utilisateur est identifié de manière unique par la colonne id, et les colonnes username et email sont également définies comme uniques pour éviter les doublons.

Cette structure relationnelle permet une intégrité des données tout en facilitant les opérations CRUD (Create, Read, Update, Delete) dans l'application. Elle constitue la base pour les échanges entre les différentes couches de l'architecture client-serveur.

**Explication Technique :**

### **1- Fonctionnement des Principales Classes :**

1. **Controller**
   * Le *Controller* est responsable de traiter les requêtes entrantes provenant de l'interface utilisateur développée en JavaFX.
   * Il applique des conditions et des validations sur les données reçues, puis encapsule ces données dans des objets DTO (*Data Transfer Objects*).
   * Ces DTO sont ensuite transmis au niveau *Service* pour le traitement ultérieur.
2. **Service**
   * Le *Service* agit comme une couche intermédiaire entre le *Controller* et la base de données.
   * Il contient un *Mapper* qui convertit les objets DTO en entités adaptées pour les opérations sur la base de données.
   * Cette couche délègue ensuite les tâches de persistance au *Layer JDBC*.
3. **Layer JDBC**
   * Le *Layer JDBC* contient les classes DAO (*Data Access Objects*) qui interagissent directement avec la base de données.
   * Il implémente les requêtes SQL nécessaires pour insérer, mettre à jour, supprimer ou lire les données.
   * Cette architecture suit le modèle *n-tier*, séparant les responsabilités en couches bien définies pour une meilleure maintenabilité et évolutivité.
4. **Classes Utilitaires**
   * Des classes utilitaires telles que StringUtils, HashUtils, et DateUtils sont utilisées pour des tâches communes, comme la manipulation de chaînes, le hachage de données sensibles (ex. mots de passe), et la gestion des dates.
   * Ces classes aident à simplifier et centraliser les fonctions répétitives, améliorant ainsi la lisibilité et la modularité du code.
5. **Système de Logs**
   * Une classe Logger personnalisée, basée sur la classe BufferedWriter de Java, a été développée pour gérer les journaux.
   * Elle crée automatiquement un fichier journal quotidien et enregistre divers types d'informations, notamment :
     + Les traces des actions des utilisateurs.
     + Les erreurs.
     + Les messages d'avertissement et d'information.
6. **Partie Serveur**
   * La connexion entre le client et le serveur via *Sockets* est initiée dès que l'utilisateur interagit avec l'application.
   * Cette communication bidirectionnelle permet de transmettre des requêtes et des réponses entre le client et le serveur en temps réel.
   * Le serveur gère également la logique métier, les validations et l'interaction avec la base de données à partir des requêtes reçues.

**2- Justification des choix technologiques :**

* **JDBC** : Utilisé pour établir une connexion fluide entre l'application Java et la base de données MySQL. Il offre une API standardisée et fiable pour exécuter des requêtes SQL et manipuler les données.
* **Sockets** : Choisi pour assurer une communication bidirectionnelle en temps réel entre le client et le serveur. Les sockets permettent une transmission rapide et efficace des données dans une architecture client-serveur.
* **MySQL** : Sélectionné pour sa robustesse, sa performance et sa compatibilité avec JDBC. MySQL garantit une gestion efficace des données relationnelles et est largement utilisé dans les applications professionnelles.

### **3- Instructions d'Installation et d'Exécution**

Pour exécuter l'application, assurez-vous d'avoir Java 17 et MySQL installés, de préférence via Laragon Server pour simplifier la gestion. Configurez la base de données en exécutant les scripts SQL fournis. Ensuite, démarrez le serveur en exécutant la classe principale correspondante, puis lancez le client pour établir la connexion via sockets. Vérifiez que les ports nécessaires sont ouverts pour la communication.