ZAKARIA KASSEMI
MOHAMED ICHOU
2 A GI
LE 25 /12/2024

SUP MTI
École Supérieure de Management, de Télécommunication et d'Informatique
RECONNUE PAR L'ÉTAT

RAPPORT

SYSTEME DE GESTION D'INVENTAIRE AVEC ACCES DISTANT DEVELOPPEMENT DISTRIBUE (J2EE)

TABLE DES MATIERES

Table Des Figures

I. Introduction

Ce projet consiste à développer une application Java permettant à une petite entreprise de gérer son inventaire de produits. L'application est divisée en deux parties : un serveur et un client, communiquant via RMI ou sockets. L'objectif est de fournir un système de gestion de l'inventaire avec des fonctionnalités CRUD et un accès distant sécurisé.

II. Objectifs

- Implémenter une base de données pour stocker les informations sur les produits.
- Créer un serveur pour gérer la connexion à la base et exposer les fonctionnalités aux clients.
- Concevoir un client permettant aux employés d'interagir avec le serveur.
- Assurer la synchronisation distante entre les clients et le serveur.

III. Partie Conception (Uml)

1. diagramme de cas d'utilisation

Ce diagramme est un diagramme de cas d'utilisation (Use Case Diagram) qui illustre les interactions entre les acteurs (utilisateurs) et les fonctionnalités d'un système.

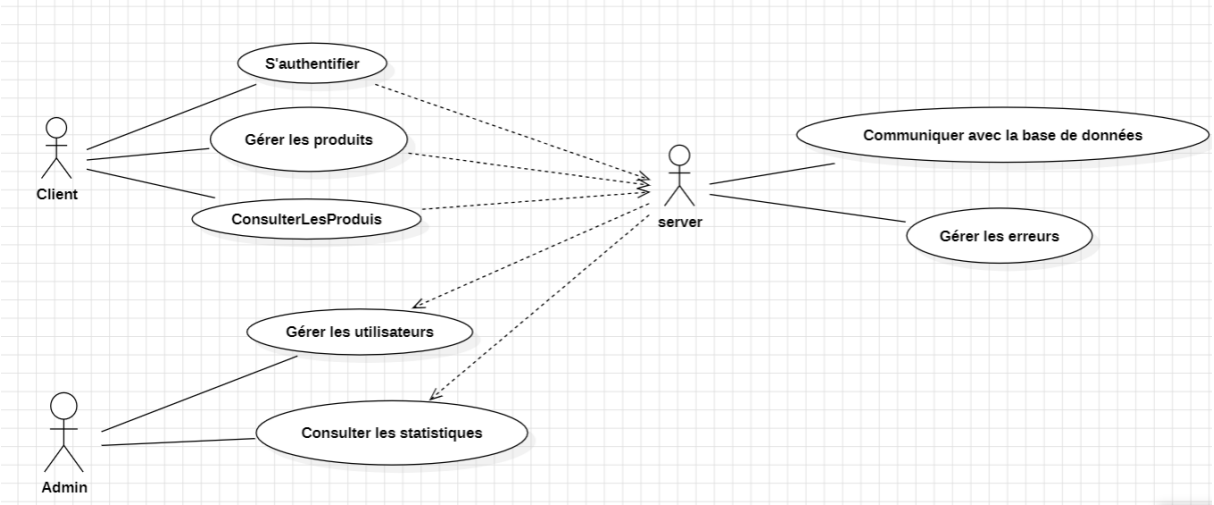


Figure 1: diagramme de cas d'utilisation

Acteurs :

- **Client** – Représente un utilisateur standard du système avec des permissions limitées.
- **Admin** – Représente un administrateur avec des privilèges étendus.
- **Server** – Représente le serveur qui gère la logique métier et les interactions avec la base de données

Cas d'utilisation pour le Client :

- **S'authentifier** – Le client peut se connecter au système en fournissant ses identifiants.
- **Gérer les produits** – Le client peut ajouter, modifier ou supprimer des produits.
- **ConsulterLesProduis** – Le client peut visualiser les informations des produits disponibles.

Cas d'utilisation pour l'Admin :

- **Gérer les utilisateurs** – L'admin peut gérer les informations des utilisateurs, y compris leur création, suppression ou modification.
- **Consulter les statistiques** – L'admin peut consulter des statistiques sur les ventes, les stocks ou d'autres informations pertinentes.

Cas d'utilisation pour le Serveur :

- **Communiquer avec la base de données** – Le serveur se charge d'interagir avec la base de données pour exécuter des opérations comme la récupération, la mise à jour ou la suppression des données
- **Gérer les erreurs** – Le serveur traite les erreurs potentielles et les exceptions qui

peuvent survenir pendant les transactions.

Relations :

- **Flèches pleines** – Indiquent une relation directe entre un acteur et un cas d'utilisation.

- **Flèches en pointillés** – Indiquent une relation indirecte où un acteur dépend d'une interaction avec un autre acteur (ici, le serveur) pour exécuter ses actions.

Ce diagramme montre comment les différents utilisateurs (clients et administrateurs) interagissent avec le serveur pour effectuer diverses opérations telles que la gestion des produits, l'authentification, la gestion des utilisateurs et la consultation des statistiques. Le serveur joue un rôle central en facilitant la communication avec la base de données et en gérant les erreurs.
Ce modèle aide à comprendre les responsabilités et les interactions entre les différentes entités du système.

2. diagramme de classe

A partir le Diagramme de classes ci-dessous on modélise une application client-serveur pour la gestion des produits avec des fonctionnalités d'authentification et de gestion des inventaires.

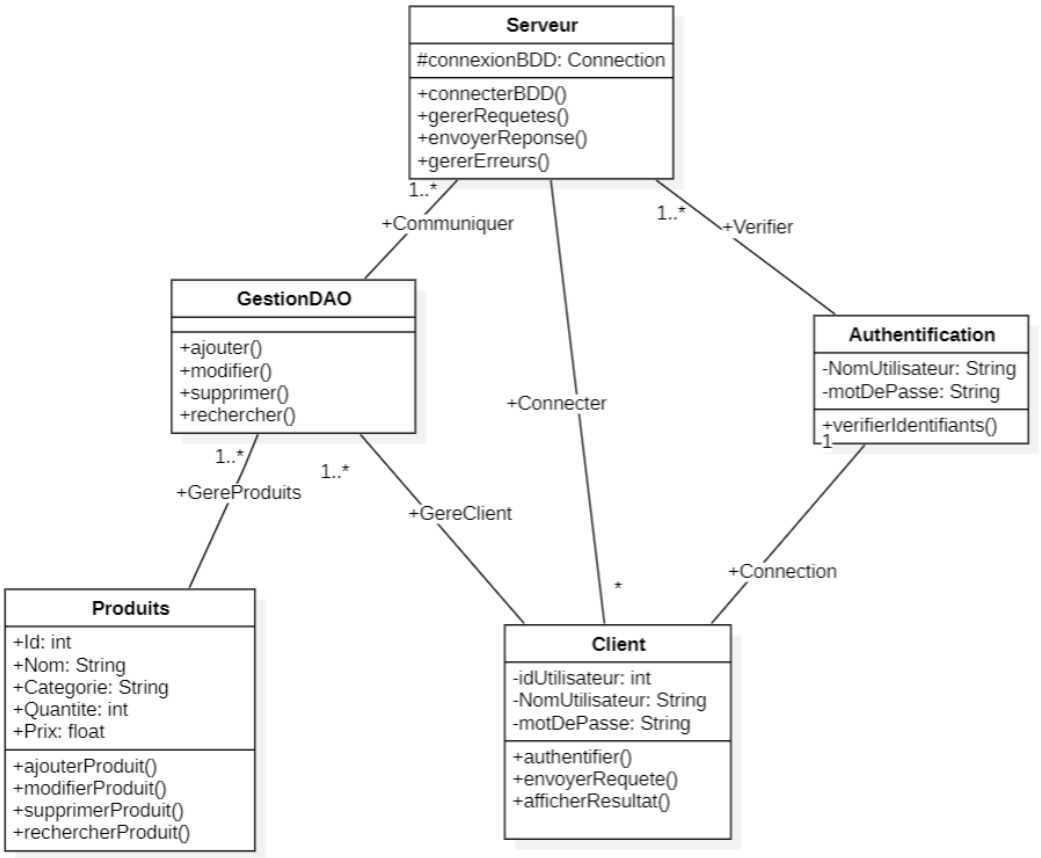


Figure 2: Diagramme de classes

1.1.1. Classe Serveur

Cette classe gère la communication avec la base de données et traite les requêtes des clients.

Attribut :

- #connexionBDD : Connection – Gère la connexion à la base de données.

Méthodes :

- connecterBDD() – Établit une connexion à la base de données.
- gererRequetes() – Traite les requêtes reçues des clients.
- envoyerReponse() – Envoie une réponse au client après traitement de la requête.
- gererErreurs() – Gère les erreurs éventuelles.

Relations :

- Relation avec **GestionDAO** (1..*) : Pour gérer les produits.
- Relation avec **Authentification** (1..*) : Pour vérifier les identifiants des utilisateurs.
- Relation avec **Client** (1..*) : Pour se connecter et répondre aux requêtes des clients.

1.1.2. Classe Client

Cette classe représente les utilisateurs (clients) qui envoient des requêtes au serveur.

Attributs :

- idUtilisateur : int – Identifiant unique de l'utilisateur.
- NomUtilisateur : String – Nom d'utilisateur.
- motDePasse : String – Mot de passe pour l'authentification.

Méthodes :

- authentifier() – Authentifie le client à l'aide de ses identifiants.
- envoyerRequete() – Envoie une requête au serveur.
- afficherResultat() – Affiche les résultats retournés par le serveur.

Relations :

- Relation avec **Serveur** (1..*) : Se connecte au serveur.
- Relation avec **GestionDAO** (1..*) : Gère les produits via les services DAO.
- Relation avec **Authentification** (1) : Pour vérifier les identifiants avant d'envoyer une requête.

1.1.3. Classe Authentification

Cette classe gère l'authentification des utilisateurs.

Attributs :

- NomUtilisateur : String – Nom d'utilisateur pour l'authentification.
- motDePasse : String – Mot de passe pour l'authentification.

Méthode :

- verifierIdentifiants() – Vérifie si les identifiants fournis sont valides.

Relations :

- Relation avec **Serveur** (1..*) : Vérifie les identifiants au niveau du serveur.
- Relation avec **Client** (1) : Gère la connexion des clients.

1.1.4. Classe GestionDAO

Cette classe implémente les opérations CRUD pour les produits (ajouter, modifier, supprimer et rechercher).

Méthodes :

- ajouter() – Ajoute un produit.
- modifier() – Modifie un produit existant.
- supprimer() – Supprime un produit.
- rechercher() – Recherche un produit dans la base de données.
-

Relations :

- Relation avec **Produits** (1..*) : Pour manipuler les objets Produits.
- Relation avec **Client** (1..*) : Pour permettre aux clients de gérer les produits.
- Relation avec **Serveur** (1..*) : Pour exécuter les opérations via le serveur.

1.1.5. Classe Produits

Cette classe représente un produit dans la base de données.

Attributs :

- id : int – Identifiant unique du produit.
- Nom : String – Nom du produit.
- Categorie : String – Catégorie à laquelle appartient le produit.
- Quantite : int – Quantité disponible en stock.
- Prix : float – Prix du produit.

Méthodes :

- ajouterProduit() – Ajoute un nouveau produit.
- modifierProduit() – Modifie un produit existant.
- supprimerProduit() – Supprime un produit.
- rechercherProduit() – Recherche un produit par critères.

Relations :

- Relation avec **GestionDAO** (1..*) : Pour effectuer des opérations CRUD sur les produits.

IV. Explication technique :

1. Fonctionnement des principales classes

1. DAO (Data Access Object)

Le DAO est un patron de conception qui permet d'abstraire l'accès aux données d'une application. Il est utilisé pour gérer les opérations CRUD (Create, Read, Update, Delete) sur une base de données, tout en offrant une séparation nette entre la logique métier et la persistance des données.
DAO permet :

- Fournir une interface pour accéder aux données.
- Effectuer des opérations de lecture, écriture et mise à jour sur la base de données.
- Manipuler les entités de la base de données sous forme d'objets Java (POJOs).
- Gérer les transactions et assurer la gestion des exceptions liées à la base de données.

2. Serveur

Le serveur est responsable de la gestion des requêtes du client, du traitement des demandes et de la gestion des ressources nécessaires. Il peut être un serveur web, un serveur d'application ou tout autre type de serveur fonctionnant sur un modèle client-serveur.

- **Fonctionnement** : Le serveur reçoit les requêtes du client, effectue le traitement nécessaire (par exemple, en appelant des méthodes sur des objets DAO ou d'autres composants métiers), et renvoie une réponse appropriée.

3. Client

Le client est l'application ou l'utilisateur qui envoie des demandes au serveur. Dans une architecture client-serveur, il peut s'agir d'une application desktop, d'une interface web ou d'une application mobile.

- **Fonctionnement** : Le client envoie une requête au serveur, souvent via HTTP dans le cas des applications web. Il attend ensuite une réponse (par exemple, des données, une confirmation, etc.). Le client peut communiquer avec le serveur en utilisant des protocoles comme HTTP, TCP, etc.

2. Justification des choix technologiques

1. JDBC (Java Database Connectivity) :

JDBC est une API standard en Java qui permet de se connecter et d'interagir avec une base de données relationnelle. Elle permet de gérer les connexions, d'exécuter des requêtes SQL, et de récupérer les résultats sous forme de ResultSet. Le choix de JDBC est pertinent dans un contexte où l'application nécessite une interaction directe avec une base de données relationnelle, comme MySQL, PostgreSQL ou Oracle. En plus, JDBC permet de gérer les transactions, de préparer des instructions SQL et d'exécuter des requêtes paramétrées.

2. RMI (Remote Method Invocation) :

RMI permet d'exécuter des méthodes à distance sur des objets Java situés sur un serveur. Il est utilisé dans une architecture distribuée où le client et le serveur peuvent

être sur des machines différentes. Le choix de RMI est justifié lorsque l'application nécessite une communication à distance entre les composants de l'application (par exemple, entre un client et un serveur). RMI est particulièrement utile dans les systèmes distribués où la transparence et la facilité d'appel des méthodes distantes sont des priorités.

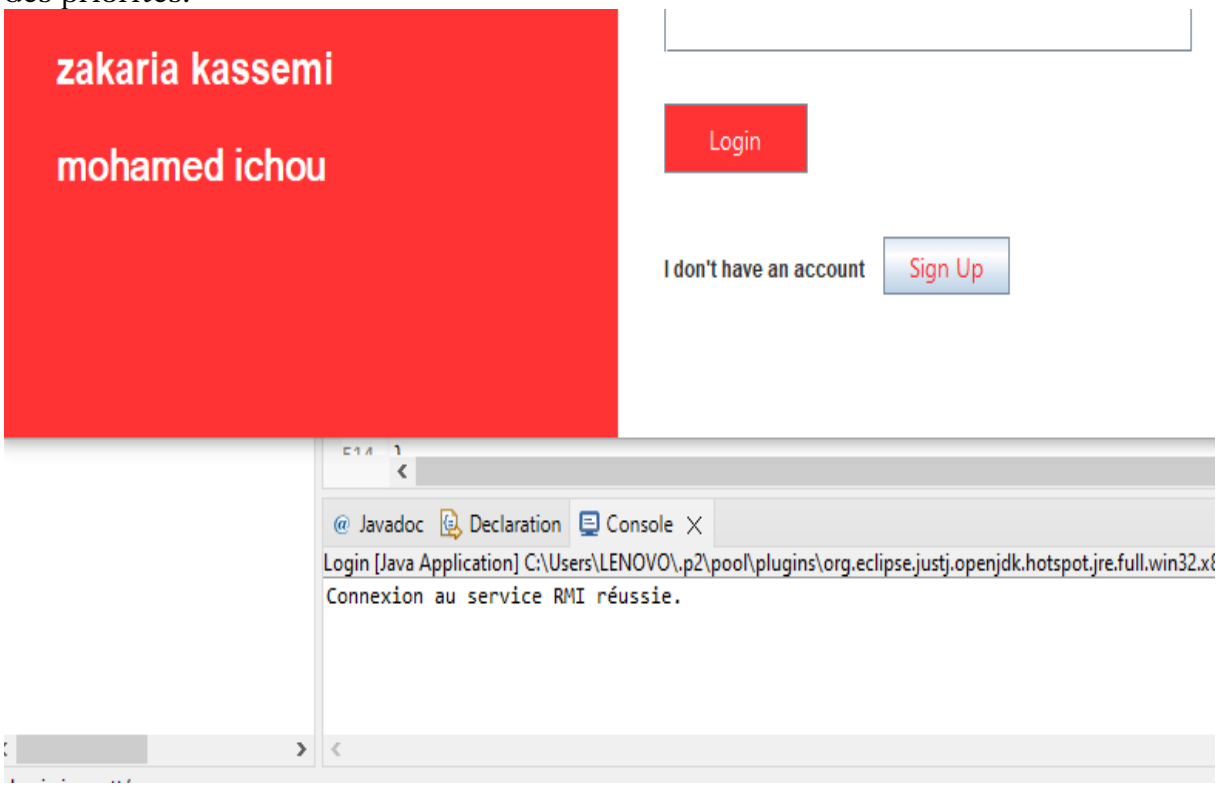


Figure 3: Démarrage de Serveur RMI

3. MySQL

Le choix de MySQL peut être justifié dans une application nécessitant une gestion efficace des données, une gestion des utilisateurs, des performances élevées et une compatibilité avec différents systèmes d'exploitation. De plus, MySQL bénéficie d'une large communauté, de nombreux outils et d'une bonne intégration avec Java via JDBC.

En résumé, JDBC est utilisé pour accéder à la base de données MySQL, RMI est utilisé pour la communication distante entre le client et le serveur dans un système distribué, et MySQL est choisi comme base de données en raison de sa stabilité, de ses performances et de son écosystème riche pour les applications Java.

V. Realisation et Test

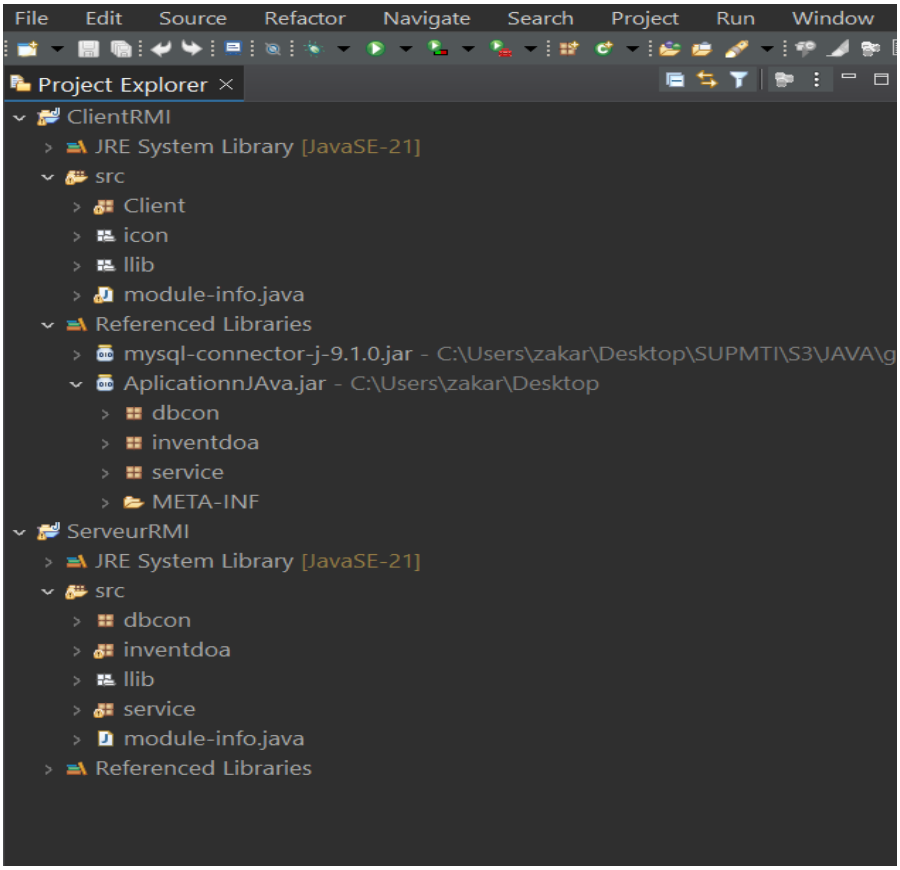
1. Création de la base de données

a conception de la base de données en créant un modèle relationnel adapté à nos besoins spécifiques. Pour cela, nous utiliserons le Système de Gestion de Base de Données (SGBD) phpMyAdmin pour la gestion de la base de données, et le serveur XAMPP pour héberger et gérer l'environnement de développement local.

NB :La base de donnée est jointe

2. Realisation de Ficher Jar

le fichier JAR permettent D'assurer la communication entre le serveur RMI et le client via des objets distants et De simplifier le partage et le déploiement des modules (comme les connexions et services de base de données)



3. Authentification

Cette première capture présente l'interface d'authentification dans laquelle l'administrateur doit saisir son nom d'utilisateur et son mot de passe pour commencer à utiliser notre application. Cette interface constitue la fenêtre d'accueil de notre application.

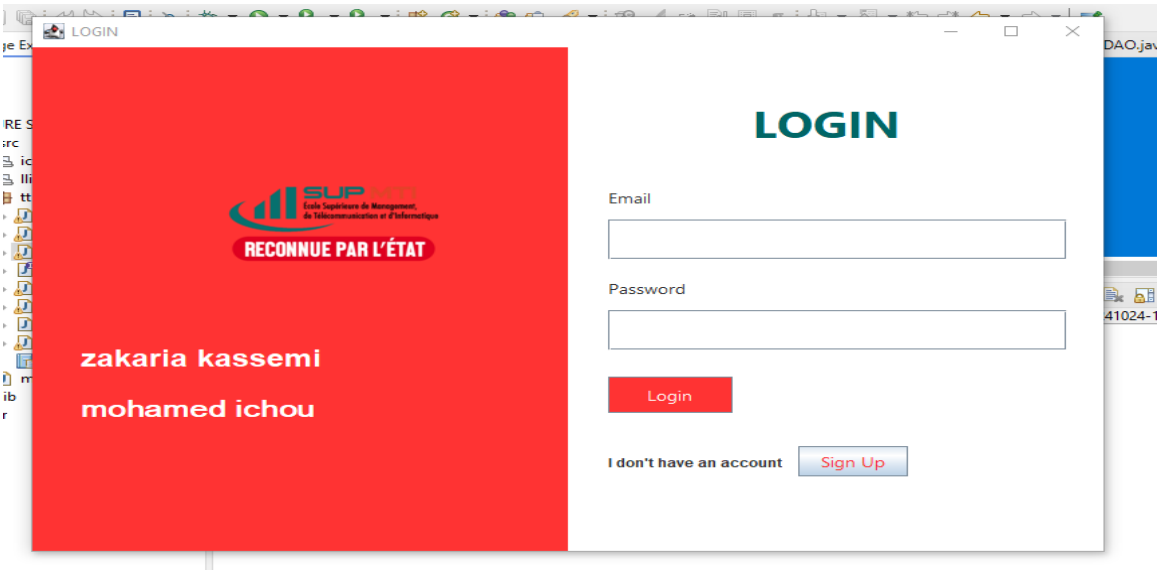


Figure 4:login

4. Creation d'un nouveau Compte

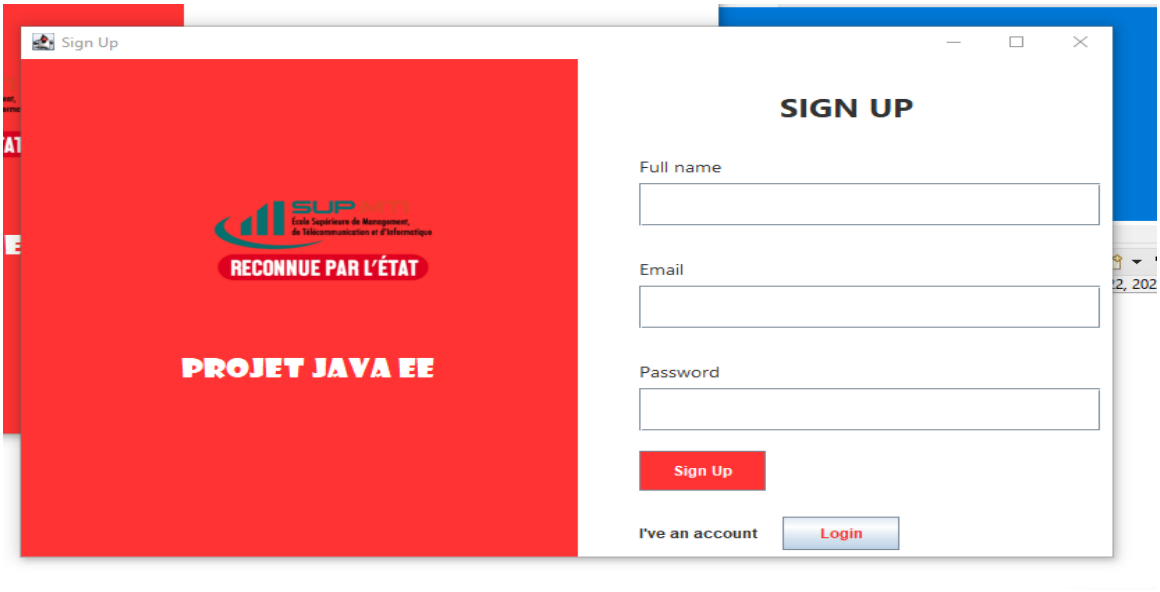


Figure 5: Creation d'un nouveau Compte

Cette interface est un bon point de départ pour un projet Java EE. Elle présente une organisation claire et un design attractif. Quelques améliorations en matière de validation, sécurité et feedback utilisateur peuvent être apportées pour renforcer son efficacité et sa convivialité.

- **Les utilisateurs peuvent s'inscrire en fournissant leur nom complet, email, et mot de passe.**
- **Les informations saisies sont probablement enregistrées dans une base de données via une connexion Java EE.**

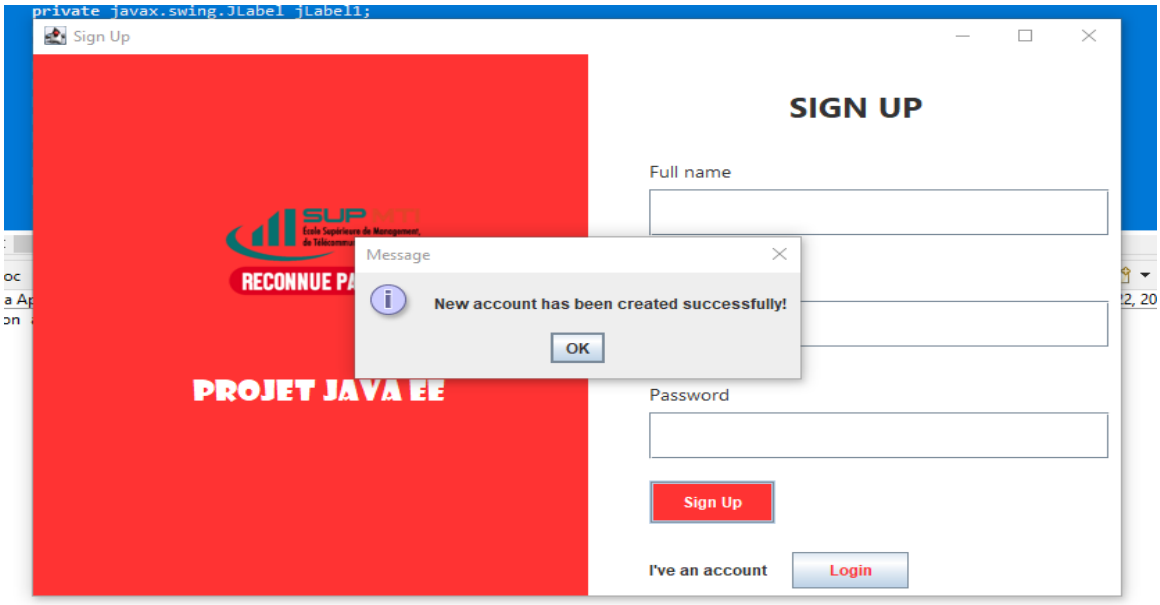


Figure 6 :Créer un compte utilisateur

Une fois le compte créé avec succès, un message de confirmation apparaît ("New account has been created successfully!").

Connexion pour les utilisateurs existants :

- **Si un utilisateur a déjà un compte, il peut cliquer sur le bouton Login pour accéder à la page de connexion.**
- **Cela permet une transition rapide entre l'inscription et la connexion.**

Validation des données :

- **Vérification que l'email est valide et que le mot de passe respecte certaines règles de sécurité.**

Sécurité et gestion des comptes :

- **Stockage sécurisé des mots de passe (par hachage ou chiffrement).**
- **Prévention des doublons d'inscription (email unique).**

Apres l'inscription et la Confirmation Une interface principale va être afficher pour les opérations de gestion

Figure 7 : L'interface principale

- L'interface comporte une disposition en deux parties principales :
  - **Zone supérieure** : Formulaire d'entrée de données comprenant des champs pour le nom, la catégorie, la quantité et le prix des articles.
  - **Zone inférieure** : Tableau listant les données des articles existants, incluant les colonnes ID, nom, catégorie, quantité, et prix.

5. Les operation CRUD

1. Ajouter

- **Ajouter** : Pour insérer de nouveaux articles dans le système.

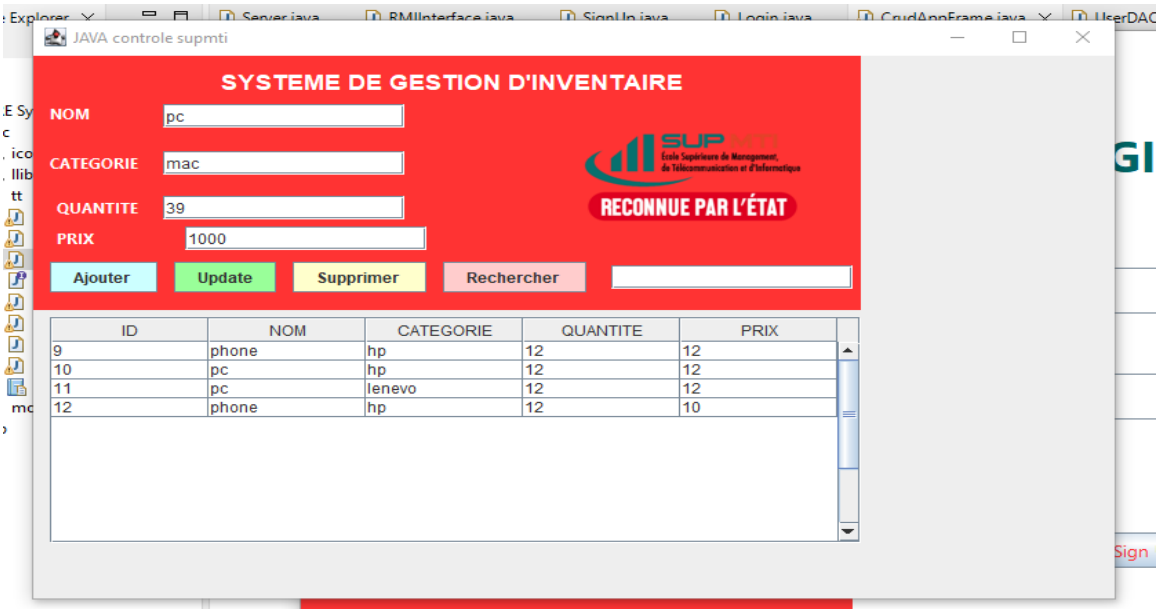


Figure 8:insérer de nouveaux articles

2. Mise a jour des article

- **Update** : Pour mettre à jour les informations existantes.

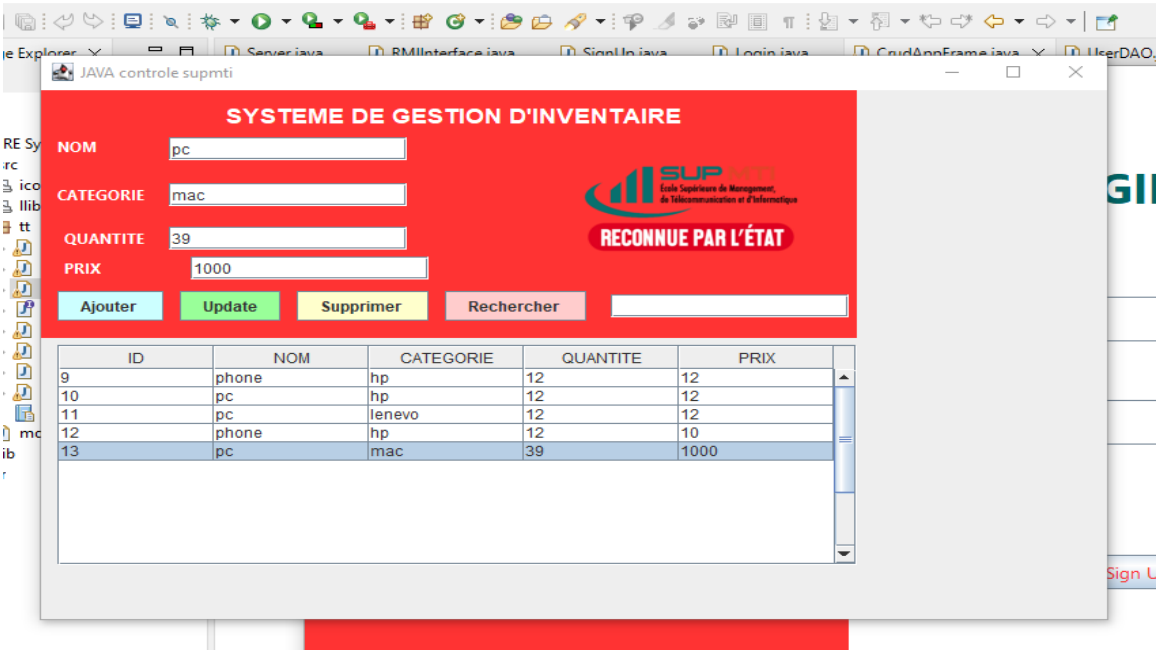


Figure 9: Update

Le bouton « Update » (mettre à jour) permettant de modifier des entrées existantes dans la base de données. L'utilisateur modifie les informations dans les champs du formulaire correspondant.

3. Supprimer

- **Supprimer** : Pour supprimer un article sélectionné.

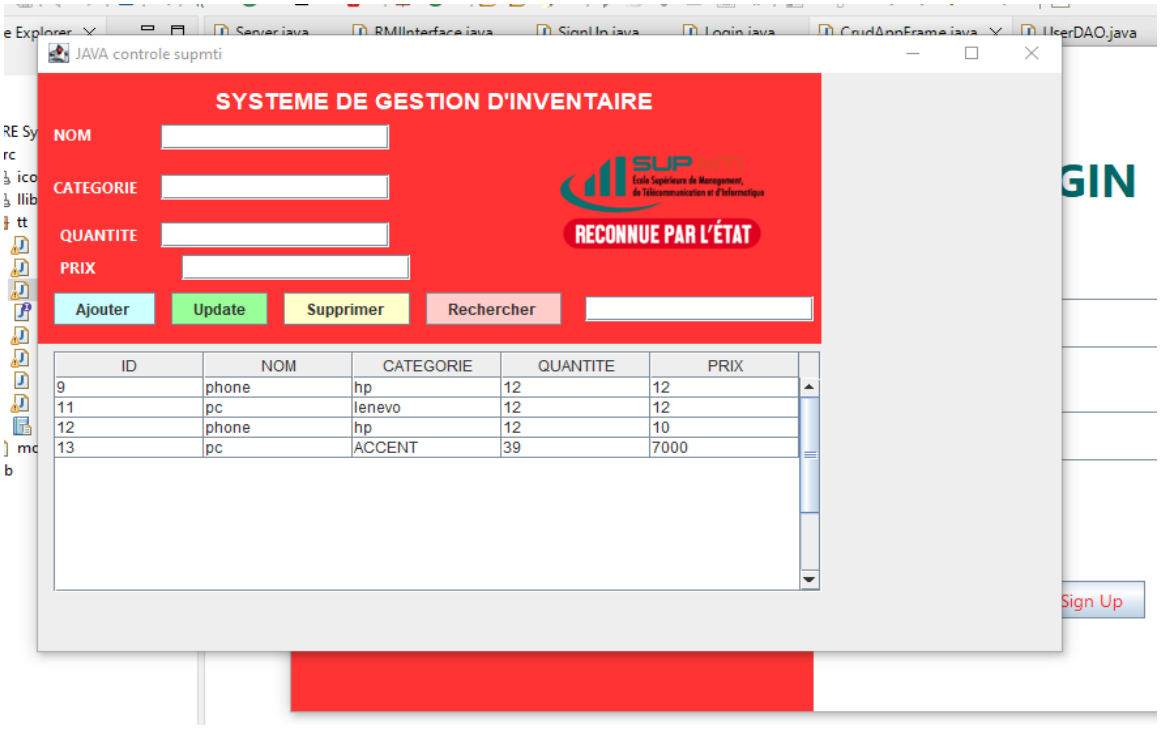


Figure 10: suppression

Une fois la suppression effectuée avec succès, le tableau ou la liste est mis à jour pour refléter les modifications.

4. Rechercher

- **Rechercher** : Pour rechercher un article en fonction d'un critère

Le bouton « Rechercher » permet à l'utilisateur de localiser rapidement des informations spécifiques dans la base de données.

L'utilisateur saisit un critère de recherche (nom) dans les champs appropriés.

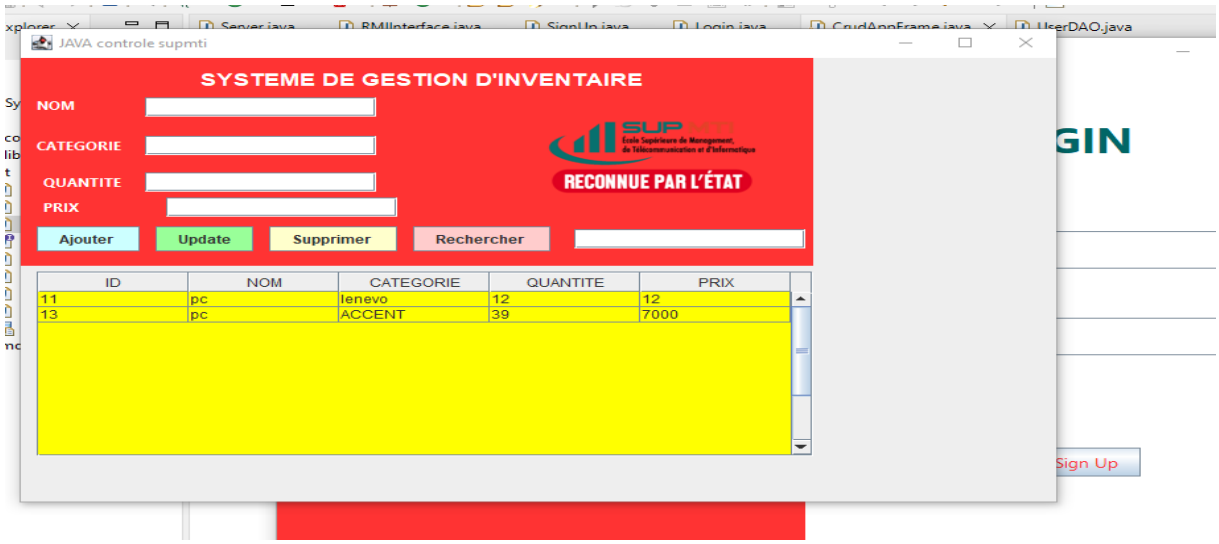


Figure 11 : Rechercher

Lors du clic sur le bouton « Rechercher », une requête est envoyée au backend pour extraire les données correspondant aux critères spécifiés. Les données correspondantes sont affichées dans le tableau

VI. Code Source

Le code source a été publié sur GitHub .

github : https://github.com/zakariakassemi2000/projetJava2.git