

**RAPPORT DU MINI-PROJET E-SALAF**

**POUR LE MODULE : POO EN Java**

**Rapport de mini-Projet**

**Réalisé par :**

ZAHTI Soukaina

LST-GI

Groupe : 01

**Encadré par :**

Elaachak Lotfi

EN-NAIMI El Mokhtar

**REMERCIMENT**

*Avant tout développement, nous souhaitons exprimer nos sincères remerciements à notre Professeur Lotfi EL AACHAK et à Monsieur Ennaimi Mokhtar, qui nous ont donné l’opportunité de réaliser ce projet pratique en JavaFX. Leurs conseils, leur expertise et leur accompagnement ont été précieux tout au long de ce travail. Nous avons été très motivés pour atteindre le niveau d'excellence attendu de notre part et nous espérons que ce projet sera à la hauteur de leurs attentes. Encore une fois, nous tenons à remercier chaleureusement notre professeur et notre encadrant pour leur soutien et leur confiance*

**TABLE DE CONTENU**

* INTRODUCTION
* Création de la base de données
* Interface d’accueil
* Interface de gestion clients
  + - Afficher les clients
    - Ajouter client
    - Supprimer client
    - Modifier client
    - Ajouter crédit
    - Afficher crédit pour un client
* Interface de gestion de produits
  + - Afficher les produits
    - Ajouter produit
    - Supprimer produit
    - Modifier client
* Interface de gestion de crédit

**Introduction :**

Le présent rapport concerne le développement d'un mini projet E-salaf qui permet à son utilisateur d'enregistrer les clients, les produits et les crédits en utilisant les technologies JavaFX, MySQL ,Java et IDE Intellij IDEA . Ce projet est destiné à faciliter le processus de gestion des crédits en permettant à l'utilisateur d'effectuer des opérations CRUD sur les clients, les produits et les crédits.

La réalisation de ce projet implique l'utilisation de différentes technologies et compétences telles que la programmation orientée objet, la gestion de base de données, la conception d'interface utilisateur, et la manipulation de l'architecture du logiciel.

Dans ce rapport, nous allons décrire les différentes étapes de développement du projet, allant de l'analyse des besoins à la conception et la mise en œuvre des différentes fonctionnalités. Nous allons également discuter des différents problèmes rencontrés lors du développement et des solutions adoptées pour y remédier.

En fin de compte, nous espérons que ce rapport fournira une vue d'ensemble détaillée de la manière dont ce projet a été conçu et développé, ainsi que des avantages qu'il peut apporter pour améliorer le processus de gestion des crédits

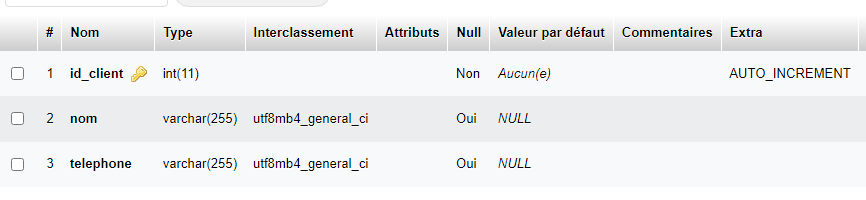
**Structure de la base de données :**

Avant de commencer à coder, on a d’abord créer une base de données contenant trois table : client, credits et produits.

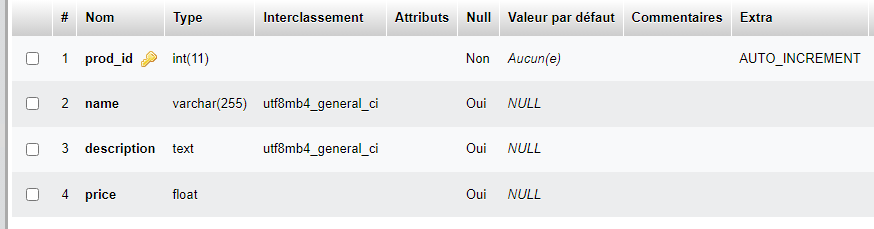
La table credits est liée à les deux autres tables par des clés étrangères (produit\_id et client\_id)

Voici la structure de chacune des tables :

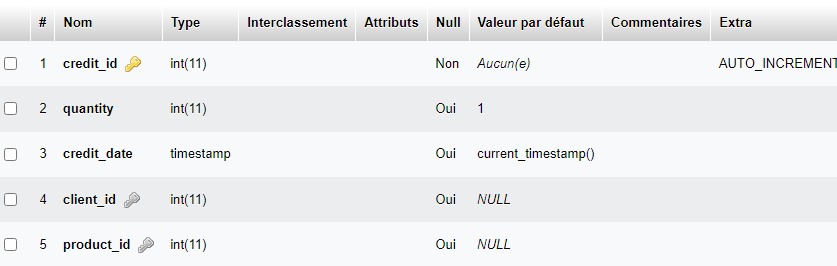
**Table client :**

****

**Table produits :**

****

**Table crédits :**

****

**L’interface d’accueil :**

Au démarrage de l’application, on présente à l’utilisateur une interface qui lui permet de basculer entre les différentes autres interfaces.

On lui permet de faire ça en utilisant les boutons **« view clients »,** **« view products** » et **« view credit ».**

Pour chacun de ces bouton on a attribuer une fonction **#onAction**, le rôle de laquelle est d’ouvrir l’interface correspendante.

public void onCliButtonClick() {

    try {

        // Load the FXML file for the new interface

        FXMLLoader loader = new FXMLLoader(getClass().getResource("hello-view.fxml"));

        Parent root = loader.load();

        // Create a new stage to display the new interface

        Stage stage = new Stage();

        stage.setScene(new Scene(root));

        stage.setTitle("Hello E-salaf");

        // Show the new interface

        stage.show();

    } catch (IOException e) {

        e.printStackTrace();

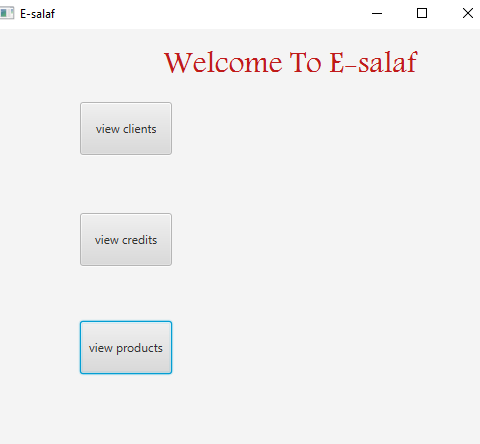
    }

}

En cliquant sur le bouton **« view clients »** on ouvre l’interface **« hello-view.fxml »**

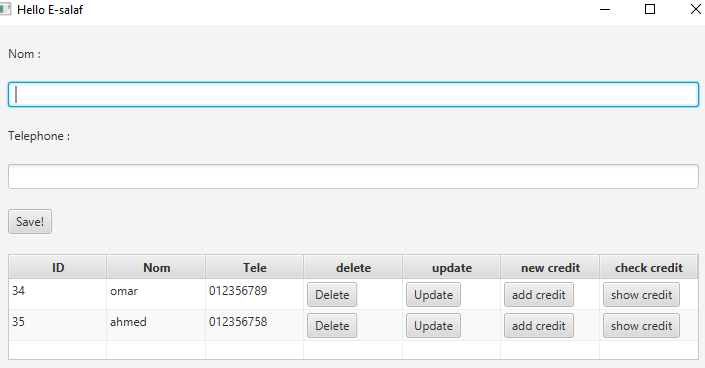
Qui contient la table des clients.

Même chose pour les boutons **« view credit »** et **« view product »** qui ouvrent respectivement les interface **« showCredit.fxml »** et **« showProduct.fxml »**

****

**Interface Gestion Des Clients :**

Cette interface contient deux **TextFields** pour entrer les noms et numéros de téléphone des clients, un bouton **« save »** pour enregistrer les informations et un tableau qui affiche ces informations.



En cliquant sur le bouton « save » on appelle la fonction **onSaveButtonClick(),** qui crée une instance du classe client et lui attribut les informations entré dans les TextFields en utilisant le constructeur.

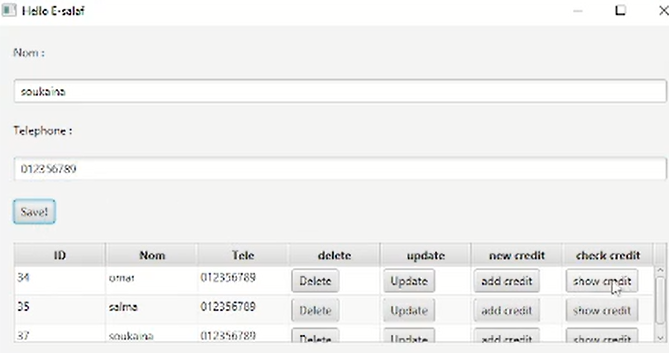
  Client cli = new Client(0l , nom.getText() , tele.getText());

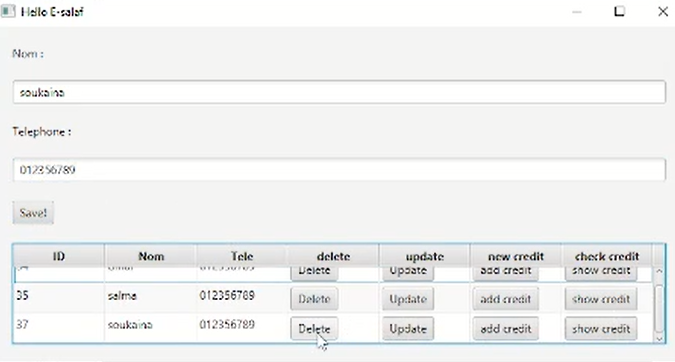
Après elle crée une instance de classe clientDAO contenant les fonctions en relations avec la base de données et faire appel à la fonction **save().**

 ClientDAO clidao = new ClientDAO();

        clidao.save(cli);

et finalement la fonction **onSaveButtonClick()** faire appel à la fonction updateTable() pour afficher ces informations enregistrés dans la table de l’interface graphique**.**

****

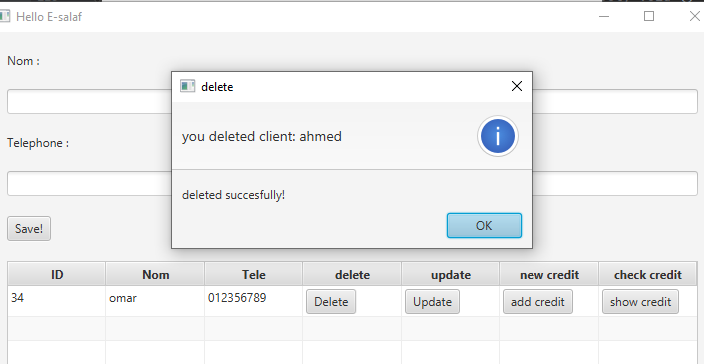
****

**La fonction updateTable()**

La fonction "**UpdateTable"** est utilisée pour mettre à jour le contenu d'un tableau qui affiche des données clients. Elle utilise des **"CellValueFactory**" pour lier les propriétés des objets **"Client"** aux colonnes du tableau. Elle ajoute également des boutons pour effectuer des opérations telles que la suppression, la mise à jour et l'ajout de crédit pour chaque client. Chaque bouton est configuré pour effectuer une action spécifique lorsqu'il est cliqué en fonction des données du client associé à la ligne du tableau. Enfin, la fonction appelle la méthode "**setDataClients"** pour mettre à jour le contenu du tableau avec les données clients les plus récentes.

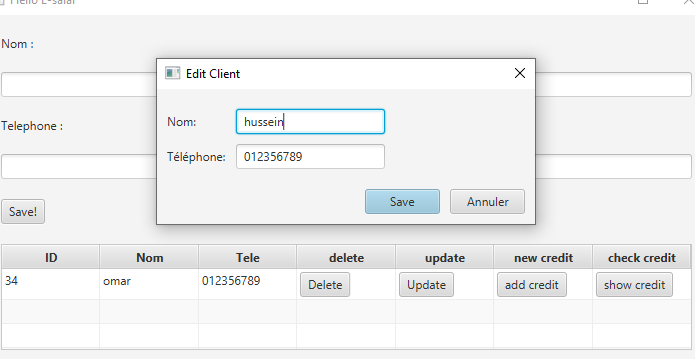
La fonction "UpdateTable" utilise quatre boutons différents pour effectuer des opérations spécifiques sur les données clients affichées dans le tableau. Voici comment fonctionne chaque bouton :

* **Bouton "Delete"** : Ce bouton est créé pour chaque ligne du tableau et permet de supprimer le client associé à cette ligne. Lorsque l'utilisateur clique sur le bouton "Delete", la méthode "delete" du DAO associé est appelée pour supprimer le client correspondant de la base de données. Ensuite, la méthode "UpdateTable" est appelée pour mettre à jour le contenu du tableau avec les données clients les plus récentes.

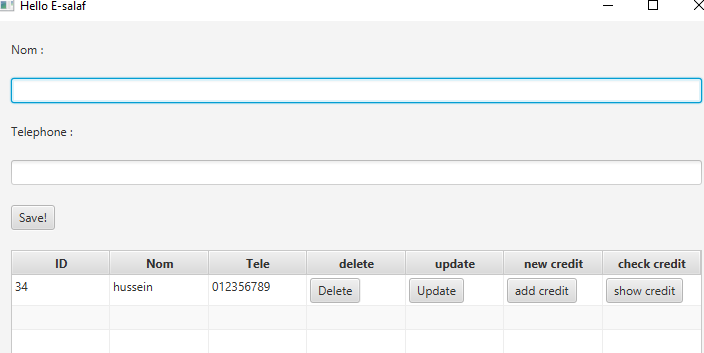


* **Bouton "Update"** : Ce bouton est également créé pour chaque ligne du tableau et permet de mettre à jour les informations du client associé à cette ligne. Lorsque l'utilisateur clique sur le bouton "Update", la méthode "editClient" est appelée pour ouvrir une nouvelle interface utilisateur permettant à l'utilisateur de modifier les informations du client. Cette méthode prend en paramètre l'objet "Client" associé à la ligne du tableau.

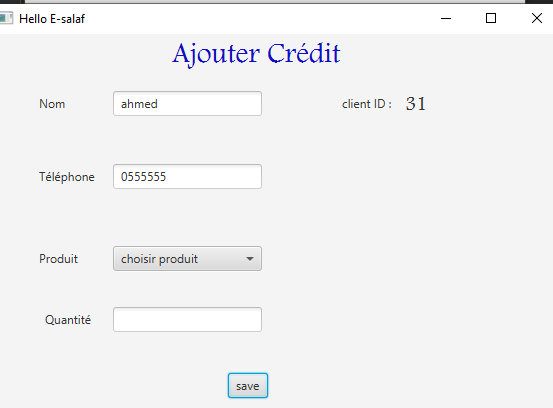
AVANT MODIFICATION



Après modification :



* **Bouton "add credit"** : Ce bouton est également créé pour chaque ligne du tableau et permet d'ajouter du crédit au compte du client associé à cette ligne. Lorsque l'utilisateur clique sur le bouton "add credit", la méthode "getTableRow().getItem()" est appelée pour récupérer l'objet "Client" associé à la ligne du tableau. Ensuite, une nouvelle interface utilisateur est ouverte pour permettre à l'utilisateur d'ajouter du crédit au compte du client. Les informations du client sélectionné sont également passées à cette nouvelle interface.



* **Bouton "show credit"** : Ce bouton est également créé pour chaque ligne du tableau et permet d'afficher le crédit disponible sur le compte du client associé à cette ligne. Lorsque l'utilisateur clique sur le bouton "show credit", la méthode "getTableRow().getItem()" est appelée pour récupérer l'objet "Client" associé à la ligne du tableau. Ensuite, une nouvelle interface utilisateur est ouverte pour afficher le crédit disponible sur le compte du client sélectionné. Les informations du client sélectionné sont également passées à cette nouvelle interface.

En résumé, chaque bouton permet d'effectuer une opération spécifique sur les données clients affichées dans le tableau. Ces boutons utilisent les propriétés des objets "Client" associés aux lignes du tableau pour effectuer les opérations de manière ciblée sur les clients sélectionnés

**Interface Ajouter Crédit :**

Cette interface permet d'ajouter une nouvelle entrée de crédit pour un client et un produit sélectionné.

Le code comporte des éléments d'interface tels que des champs de texte et des boutons pour l'entrée des données, ainsi qu'un menu déroulant pour la sélection des produits.

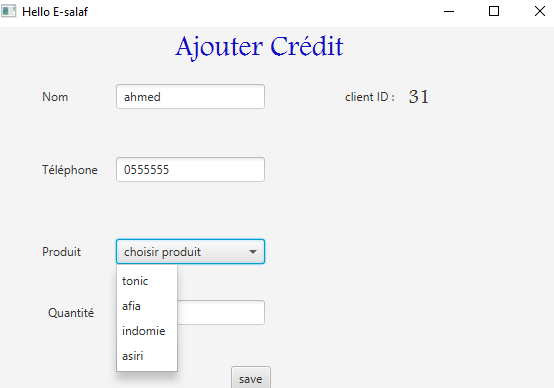
Le code utilise également des classes et des méthodes de la couche de persistance pour enregistrer les données entrées dans la base de données.

La méthode initialize() est appelée lors du chargement de l'interface utilisateur et permet de charger les produits à partir de la base de données dans le menu déroulant.

La méthode addProductsToMenu() est utilisée pour ajouter chaque produit en tant qu'élément de menu avec l'ID du produit en tant que propriété d'ID.

La méthode handleProductSelection() est appelée lorsque l'utilisateur sélectionne un produit dans le menu déroulant et récupère l'ID du produit correspondant.

La méthode onSave() est appelée lorsqu'un utilisateur appuie sur le bouton de sauvegarde et récupère les valeurs entrées pour créer une nouvelle entrée de crédit, qu'elle enregistre ensuite dans la base de données en utilisant la classe creditDAO





**Interface Afficher Crédit :**

La classe "showCreditById" est une classe qui implémente l'interface Initializable de JavaFX. Elle contient des méthodes pour afficher les détails d'un crédit donné en utilisant une table view. Voici le fonctionnement de chaque méthode :

* **setSelecetdClient() :**

public void setSelecetdClient(String name, String Tele , String id) throws SQLException {

    this.nom.setText(name);

    this.tele.setText(Tele);

    long id\_client=Long.parseLong(id);

    id\_cli=id\_client;

    getProductIDs();

    List<credit> credits = new ArrayList<>(); // create a list of credit objects

// add credit objects to the list

    populateTableData(credits); // call the function with the list of credit objects

    creditDAO credao = null;

    credao = new creditDAO();

    calculateTotal(credao.getRowsByClientId(id\_cli));

}

Cette méthode est appelée lorsqu'un client est sélectionné pour afficher ses crédits. Elle prend en entrée le nom, le téléphone et l'identifiant du client. Elle appelle ensuite la méthode "getProductIDs()" pour récupérer la liste des identifiants de produits associés aux crédits du client sélectionné. Elle crée ensuite une liste de crédits et appelle la méthode "populateTableData()" avec cette liste pour remplir la table view avec les produits correspondants. Elle utilise également la méthode "calculateTotal()" pour calculer le total des crédits associés au client et l'afficher.

* **populateTableData()** :

Cette méthode prend en entrée une liste de crédits et utilise la méthode "getProductIDs()" pour récupérer les identifiants de produits correspondants. Elle utilise ensuite ces identifiants pour récupérer les produits correspondants à partir de la base de données en utilisant la classe "productDAO". Elle ajoute ensuite les produits récupérés à une liste observable pour les afficher dans la table view. Elle ajoute également une colonne pour afficher les quantités de crédits associés à chaque produit et une autre colonne pour afficher les dates des crédits.

* **getProductIDs() :**

public  ObservableList<Long> getProductIDs(){

    creditDAO credao = null;

    ObservableList<Long> listfx = FXCollections.observableArrayList();

    try {

        credao = new creditDAO();

        for(Long ettemp : credao.getProductIdsByClientId(id\_cli))

            listfx.add(ettemp);

    } catch (SQLException e) {

        throw new RuntimeException(e);

    }

    return listfx ;

}

Cette méthode utilise la classe "creditDAO" pour récupérer la liste des identifiants de produits associés aux crédits du client sélectionné. Elle retourne une liste observable de ces identifiants.

* **calculateTotal() :**

public void calculateTotal(List<credit> credits) throws SQLException {

    float totalPrice = 0;

    for (credit credit : credits) {

        productDAO prodao = new productDAO();

        long id=credit.getProduit\_id();

        product p=prodao.getOne(id);

        totalPrice += credit.getQuantity() \* p.getPrice();

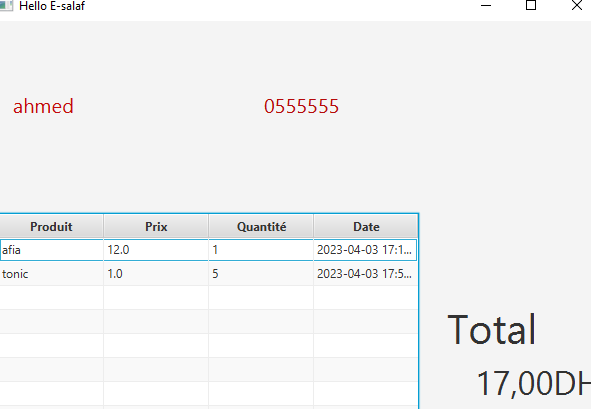
    }

    total.setText(String.format("%.2f", totalPrice) +"DH");

}

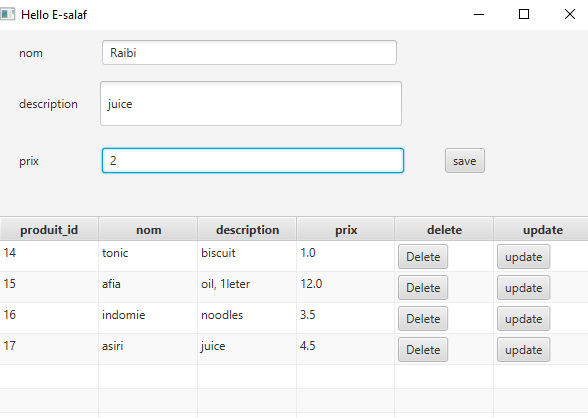
Cette méthode prend en entrée une liste de crédits et utilise la classe "productDAO" pour récupérer les produits correspondants à partir de leurs identifiants. Elle calcule ensuite le total des crédits en multipliant la quantité de chaque crédit par le prix de

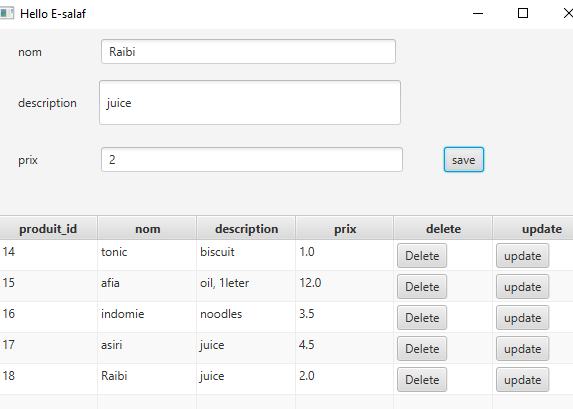
son produit correspondant et en les ajoutant. Elle affiche ensuite le résultat dans la fenêtre.

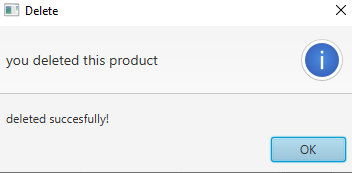


**Interface Gestion Des Produits :**

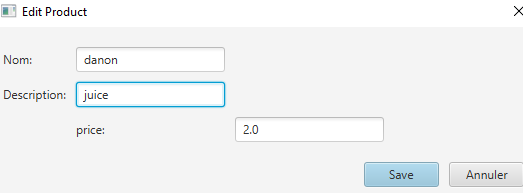
On a utilisé presque les même fonctions pour ajouter, supprimer, éditer et afficher les produits qu’on a utilisé dans l’interface Clients.

**Ajouter un produit :**

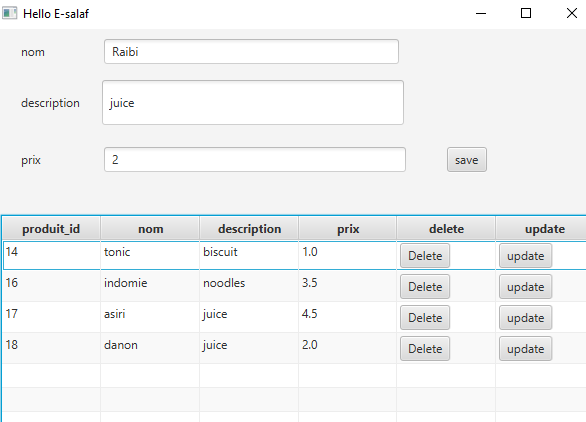


**Supprimer un produit :**

**Modifier un produit :**

****

**Après modification :**

****