

# RAPPORT DU MINI-PROJET E-SALAF POUR LE MODULE : POO EN Java

## Réalisé par :

ZAHTI Soukaina

LST-GI

Groupe : 01

## Encadré par :

Elaachak Lotfi

EN-NAIMI El Mokhtar

# **REMERCIEMENT**

*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 :

	#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
<input type="checkbox"/>	1	id_client 	int(11)			Non	Aucun(e)		AUTO_INCREMENT
<input type="checkbox"/>	2	nom	varchar(255)	utf8mb4_general_ci		Oui	NULL		
<input type="checkbox"/>	3	telephone	varchar(255)	utf8mb4_general_ci		Oui	NULL		

### Table produits :

	#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
<input type="checkbox"/>	1	prod_id 	int(11)			Non	Aucun(e)		AUTO_INCREMENT
<input type="checkbox"/>	2	name	varchar(255)	utf8mb4_general_ci		Oui	NULL		
<input type="checkbox"/>	3	description	text	utf8mb4_general_ci		Oui	NULL		
<input type="checkbox"/>	4	price	float			Oui	NULL		

### Table crédits :

	#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
<input type="checkbox"/>	1	credit_id 	int(11)			Non	Aucun(e)		AUTO_INCREMENT
<input type="checkbox"/>	2	quantity	int(11)			Oui	1		
<input type="checkbox"/>	3	credit_date	timestamp			Oui	current_timestamp()		
<input type="checkbox"/>	4	client_id 	int(11)			Oui	NULL		
<input type="checkbox"/>	5	product_id 	int(11)			Oui	NULL		

## 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 correspondante.

```
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.

A screenshot of a software window titled "Hello E-salaf". The window has a light gray background. At the top, there are two labels: "Nom :" and "Telephone :". Below "Nom :" is a text input field with a blue border. Below "Telephone :" is a text input field with a gray border. Below these fields is a button labeled "Save!". At the bottom of the window is a table with 7 columns: "ID", "Nom", "Tele", "delete", "update", "new credit", and "check credit". The table contains two rows of data and one empty row at the bottom.

ID	Nom	Tele	delete	update	new credit	check credit
34	omar	012356789	Delete	Update	add credit	show credit
35	ahmed	012356758	Delete	Update	add credit	show credit

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(01 , 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 .

ID	Nom	Tele	delete	update	new credit	check credit
34	omni	012356789	Delete	Update	add credit	show credit
35	salma	012356789	Delete	Update	add credit	show credit
37	soukaina	012356789	Delete	Update	add credit	show credit



Nom :

soukaina

Telephone :

012356789

Save!

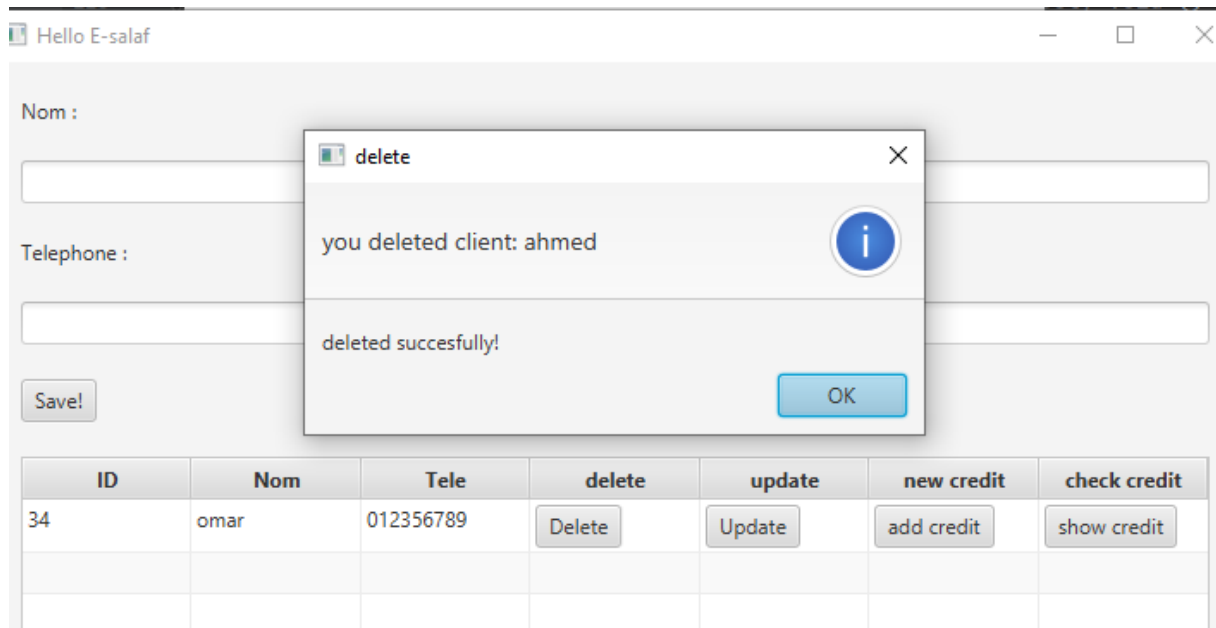
ID	Nom	Tele	delete	update	new credit	check credit
35	salma	012356789	Delete	Update	add credit	show credit
37	soukaina	012356789	Delete	Update	add credit	show credit

### 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.

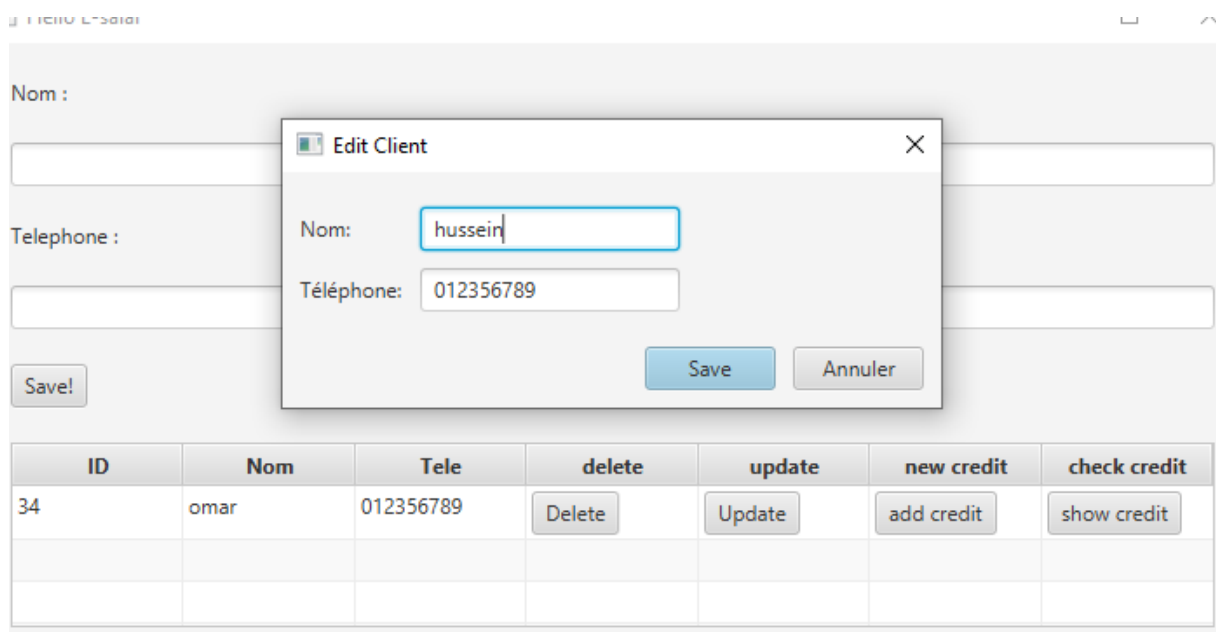
La fonction "UpdateTable" utilise quatre boutons différents pour effectuer des opérations spécifiques sur les données clients affichées dans le tableau. Voici comment fonctionne chaque bouton :

- **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 :

The screenshot shows a window titled "Hello E-salaf" with a form for editing a client. The form has two input fields: "Nom :" and "Telephone :". Below the "Telephone :" field is a "Save!" button. Below the form is a table with 7 columns: "ID", "Nom", "Tele", "delete", "update", "new credit", and "check credit". The first row of the table contains the data for client ID 34, named "hussein", with telephone "012356789". The "delete" column has a "Delete" button, the "update" column has an "Update" button, the "new credit" column has an "add credit" button, and the "check credit" column has a "show credit" button.

ID	Nom	Tele	delete	update	new credit	check credit
34	hussein	012356789	Delete	Update	add credit	show credit

- **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.

The screenshot shows a window titled "Hello E-salaf" with a form titled "Ajouter Crédit". The form has four input fields: "Nom" (containing "ahmed"), "Téléphone" (containing "0555555"), "Produit" (a dropdown menu with "choisir produit" selected), and "Quantité" (an empty field). To the right of the "Nom" field, it says "client ID : 31". At the bottom of the form is a "save" button.

- **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

Hello E-salaf

## Ajouter Crédit

Nom  client ID : 31

Téléphone

Produit

Quantité

tonic  
afia  
indomie  
asiri

save

Hello E-salaf

## Ajouter Crédit

Nom  client ID : 31

Téléphone

Produit

Quantité

save

## 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 ;  
  
}
```

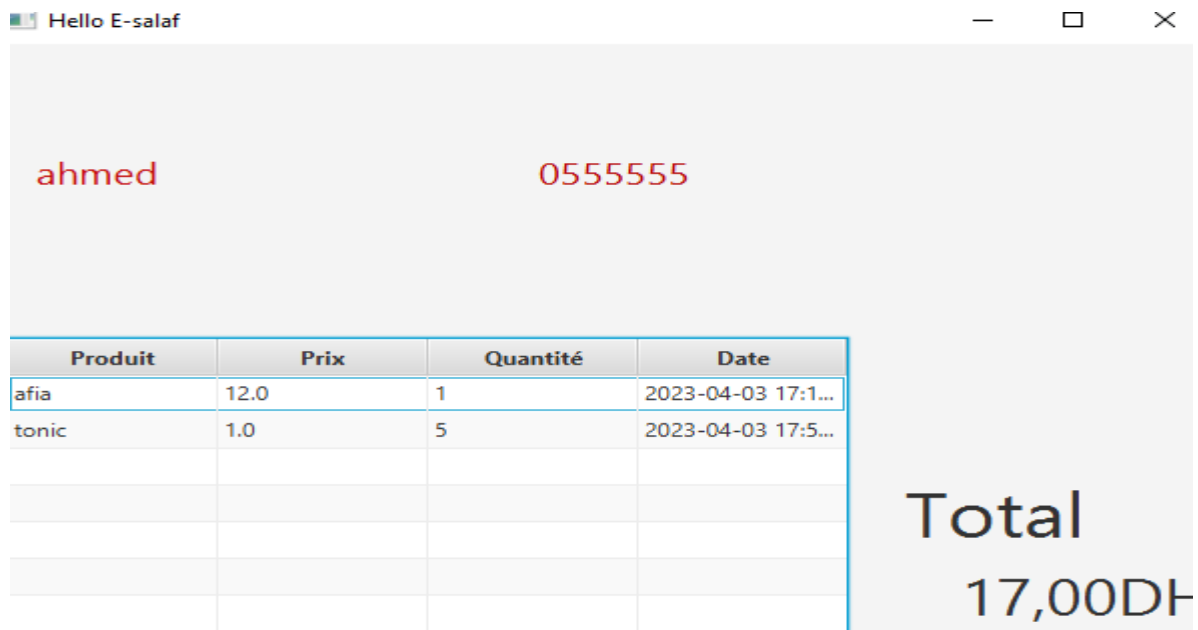
Cette méthode utilise la classe "creditDAO" pour récupérer la liste des identifiants de produits associés aux crédits du client sélectionné. Elle retourne une liste observable de ces identifiants.

➤ **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.



Produit	Prix	Quantité	Date
afia	12.0	1	2023-04-03 17:1...
tonic	1.0	5	2023-04-03 17:5...

Total  
17,00DH



## 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 :

Hello E-salaf

nom

Raibi

description

juice

prix

2

save

produit_id	nom	description	prix	delete	update
14	tonic	biscuit	1.0	Delete	update
15	afia	oil, 1leter	12.0	Delete	update
16	indomie	noodles	3.5	Delete	update
17	asiri	juice	4.5	Delete	update

Hello E-salaf

nom

Raibi

description

juice


prix


2

save

produit_id	nom	description	prix	delete	update
14	tonic	biscuit	1.0	Delete	update
15	afia	oil, 1leter	12.0	Delete	update
16	indomie	noodles	3.5	Delete	update
17	asiri	juice	4.5	Delete	update
18	Raibi	juice	2.0	Delete	update

### Supprimer un produit :


 Delete ×

you deleted this product 

deleted succesfully!

OK

### Modifier un produit :

 Edit Product ×


Nom:

Description:

price:

Save Annuler

### Après modification :

 Hello E-salaf — □ ×

nom

description

prix

produit_id	nom	description	prix	delete	update
14	tonic	biscuit	1.0	<input type="button" value="Delete"/>	<input type="button" value="update"/>
16	indomie	noodles	3.5	<input type="button" value="Delete"/>	<input type="button" value="update"/>
17	asiri	juice	4.5	<input type="button" value="Delete"/>	<input type="button" value="update"/>
18	danon	juice	2.0	<input type="button" value="Delete"/>	<input type="button" value="update"/>