

Présentée par :

*Widad Essetti*

*Groupe : 1*

*Encadrée par :*

*M.En-naimi*

*M.Elachak*

Compte rendu de l’application ‘Esalaf’





**NOM : ESSETTI PRENOM : WIDAD GROUPE : 1**

**Compte Rendu :**

Introduction :

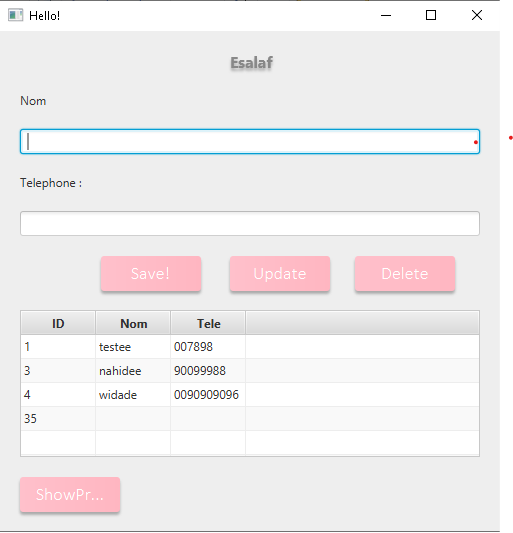
JavaFX est une bibliothèque logicielle développée par Oracle pour la création d'interfaces utilisateur graphiques (GUI) pour les applications Java. Il permet aux développeurs de créer des applications de bureau avec des interfaces utilisateur riches et interactives, en utilisant des composants graphiques tels que des boutons, des zones de texte, des listes et des tableaux. JavaFX utilise le langage de programmation Java, ce qui signifie que les développeurs peuvent profiter des avantages de Java, tels que la portabilité et la robustesse, tout en créant des interfaces utilisateur graphiques avancées pour leurs applications. JavaFX offre également une prise en charge intégrée de l'animation, de la 3D et de l'audio, ce qui en fait une solution complète pour la création d'applications de bureau riches en fonctionnalités.

Objectif du projet :

Mise en place d'une application desktop java , basée sur JDBC et JavaFX, l'application doit gérer les credit, les commandes , les produits, les clients, ainsi elle donne un tableau de board générique. En utilisant les outils suivantes : JavaFX, JDBC , Mysql , Java

Interface Client :

Cette interface nous permet d’enregistrer les clients dans un tableau , ces derniers sont enregistrés aussi dans la base de donnée .On peut aussi modifier les informations du client ou le supprimer définitivement.



Pour le bouton Update on a utilisé la fonction suivante :

void onUpdateButtonClick() {  
 Client selectedclient = mytab.getSelectionModel().getSelectedItem();  
 if(selectedclient != null){  
 try {  
 ClientDAO clidao = new ClientDAO();  
 selectedclient.setNom(nom.getText());  
 selectedclient.setTelepehone(tele.getText());  
 boolean update1;  
 if (clidao.update(selectedclient)) {  
 update1 = true;  
 } else {  
 update1 = false;  
 }  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
 UpdateTable();  
 }  
}

Pour le bouton Delete on a utilisé la fonction suivante :

void onDeleteButtonClick() {  
 Client selectedClient = mytab.getSelectionModel().getSelectedItem();  
 if (selectedClient != null) {  
 try {  
 ClientDAO clidao = new ClientDAO();  
 clidao.delete(selectedClient);  
 UpdateTable();  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
 }  
}

Pour le bouton ShowProduit , c’est un bouton qui permet de passer à l’interface produit , pour faire cela , j’ai créé une classe ’ ProduitController  ‘ et une nouvelle interface ‘ShowProduit.fxml’ .

Voici la fonction que j’ai utilisé dans ‘HelloController’ pour passer de l’interface Client à l’interface Produit.

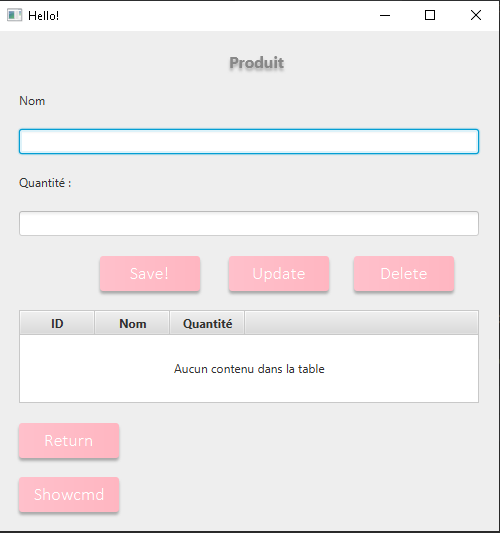
public void handlebtn1() throws Exception{  
 FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.getResource("showproduit.fxml"));  
 Stage Window = (Stage) showproduit.getScene().getWindow();  
 Window.setScene(new Scene(fxmlLoader.load() , 500,500));  
}

Dans l’interface Produit j’ai créé un autre bouton ‘return’ qui permet de revenir à l’interface Client , on utilisant la fonction suivante qui se trouve dans ‘ProduitController’ :

public void handlebtn2() throws Exception{  
 FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.getResource("hello-view.fxml"));  
 Stage Window = (Stage) returne.getScene().getWindow();  
 Window.setScene(new Scene(fxmlLoader.load() , 500,500));  
}

Interface Produit :

Cette interface nous permet d’enregistrer les produits dans un tableau , ces derniers sont enregistrés aussi dans la base de donnée .On peut aussi modifier les informations du produit ou le supprimer définitivement.



Pour enregistrer les données dans le tableau j’ai utilisé la fonction suivante :

protected void onSaveButtonClick1(){  
 produit pro = new Produit(0l , nomp.getText() , quantité.getText());  
 try {  
 ProduitDAO clidao = new ProduitDAO();  
 prodao.save(pro);  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
 UpdateTable();  
}

Pour modifier les données dans le tableau j’ai utilisé la fonction suivante :

@FXML  
void onUpdateButtonClick1() {  
 Produit selectedproduit = mytab.getSelectionModel().getSelectedItem();  
 if(selectedproduit != null){  
 try {  
 ProduitDAO prodao = new ProduitDAO();  
 selectedproduit.setNom(nom.getText());  
 selectedproduit.setQuantité(Quantité.getText());  
 boolean update1;  
 if (prodao.update(selectedproduit)) {  
 update1 = true;  
 } else {  
 update1 = false;  
 }  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
 UpdateTable();  
 }  
}

J’ai aussi créé le bouton ‘ ShowCommande’ pour passer à l’interface de la commande on utilisant la fonction suivante :

public void showcmd() throws Exception{  
 FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.getResource("showcommande.fxml"));  
 Stage Window = (Stage) cmd.getScene().getWindow();  
 Window.setScene(new Scene(fxmlLoader.load() , 500,500));  
}

Interface Commande :

Cette interface nous permet d’enregistrer les commandes dans un tableau , ces derniers sont enregistrés aussi dans la base de donnée .On peut aussi modifier les informations de la commande ou le supprimer définitivement.



Pour le bouton ‘save’ j’ai utilisé la fonction suivante :

@FXML  
protected void onSaveButtonClicka(){  
 Commande com = new Commande(0l , idd.getText() , des.getText());  
 try {  
 CommandeDAO comdao = new CommandeDAO();  
 comdao.save(com);  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
 UpdateTable2();  
}

Pour le bouton ‘Update’ j’ai utilisé la fonction suivante :

@FXML  
void onUpdateButtonClickb() {  
 Commande selectedcommande = mytab2.getSelectionModel().getSelectedItem();  
 if(selectedcommande != null){  
 try {  
 CommandeDAO comdao = new CommandeDAO();  
 selectedcommande.setPrix(idd.getText());  
 selectedcommande.setDestinataire(des.getText());  
 boolean update1;  
 if (comdao.update(selectedcommande)) {  
 update1 = true;  
 } else {  
 update1 = false;  
 }  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
 UpdateTable2();  
 }  
}

Pour le bouton ‘Delete’ j’ai utilisé la fonction suivante :

@FXML  
void onDeleteButtonClickc() {  
 Commande selectedCommande = mytab2.getSelectionModel().getSelectedItem();  
 if (selectedCommande != null) {  
 try {  
 CommandeDAO comdao = new CommandeDAO();  
 comdao.delete(selectedCommande);  
 UpdateTable2();  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
 }  
}

Conclusion :

En conclusion, le développement d'une application JavaFX dans IntelliJ peut être une expérience enrichissante, mais peut également présenter certaines difficultés. Lors de ce projet, j’ai rencontré des défis tels que la configuration de l'environnement de développement, la compréhension de la structure du projet, le débogage, la gestion des dépendances et l'optimisation des performances.

Malgré ces difficultés, j’ai pu créer une application fonctionnelle qui répond aux besoins de l'utilisateur. Nous avons utilisé les outils et les fonctionnalités offerts par IntelliJ pour développer une interface utilisateur conviviale, inclure des fonctionnalités utiles et rendre l’application stable

En travaillant sur ce projet, j’ai acquis une expérience précieuse en matière de développement d'applications JavaFX et en utilisant l'environnement de développement IntelliJ. J’ai appris à surmonter les difficultés et à créer une application de qualité qui répond aux besoins de l'utilisateur.

En fin de compte, je suis convaincus que les avantages de développer une application JavaFX dans IntelliJ l'emportent largement sur les défis rencontrés.