**Mini-Projet**

Rihab Touil

Groupe 2

Travail présenté à

Pr. *El Mokhtar En-Naimi*

Pr. *El Aachak Lotfi*

Programmation Orientée Objet en JAVA

Département Génie Informatique

Faculté des sciences et techniques de Tanger

**Préparation d’environnement de travaille :**

Le devoir est travaillé en utilisant l’éditeur de texte Netbeans .

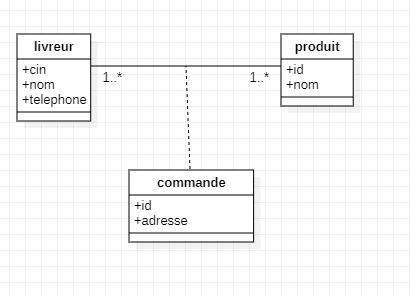
J’ai installé javaFX , l’ajouté dans libraries de Netbeans puis ajouté le path d’exécution de bibliothèque javaFX dans la configuration de la VM , puis j’ai installé Scene Builder le lié avec mon projet pour la création graphique des fichier fxml.

Pour la liaison avec la base de donné SQL , j’ai ajouté dependency de sql version 5 dans le fichier pom.xml de mon projet Maven .

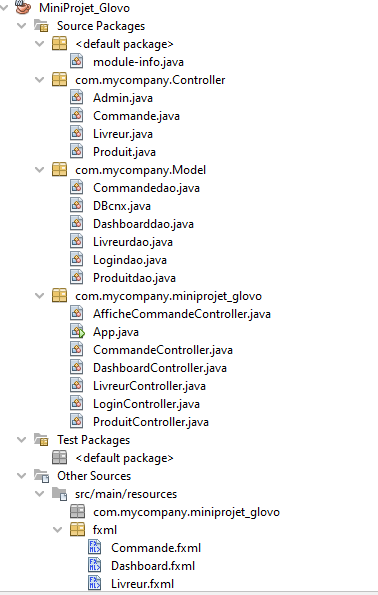




**Diagramme de class :**

****

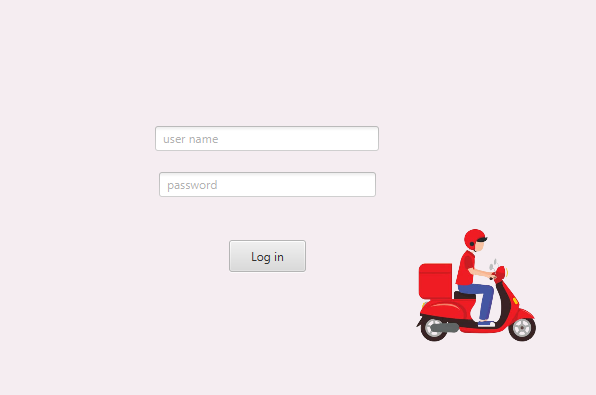
**Plan de travaille :**



dans le packages Controller j’ai crée les class , Model pour les requête SQL , miniprojet\_glovo pour les contrôleurs des view . on a aussi un package fxml pour les view et un autre image pour les images utilisé

Pour le projet Tracking livreur j’ai essayé de créer des interfaces de CRUD pour les produits et les livreurs ainsi que pour passer une commande en ajoutant son adresse le livreur et les produits .

J’ai commencé par une interface de login



Pour accéder à Dashboard on doit effectuer une connexion avec la base de donné pour vérifier le nom d’utilisateur et mot de passe entrer

Admin admin = new Admin() ;

Logindao lg = new Logindao();

public void signin(Event e) throws SQLException, IOException

{

admin.setUsername(txtuser.getText());

admin.setPassword(txtpassword.getText());

if(lg.isLogin(admin))

{

Scene scene;

Node node = (Node)e.getSource(); // emplcement dyal button

Stage stage = (Stage)node.getScene().getWindow(); //stage

Parent root = FXMLLoader.load(getClass().getResource("/fxml/Dashboard.fxml"));

scene = new Scene(root);

stage.setScene(scene);

stage.show();

}

else

{

error.setText("username or password is incorrect");

}

}

Parent root = FXMLLoader.load(getClass().getResource("/fxml/Dashboard.fxml"));

scene = new Scene(root);

stage.setScene(scene);

stage.show();

}

else

{

error.setText("username or password is incorrect");

}

La méthode signin vérifie si les données sont correctes en vérifiant le boolean retourné par isLogin() de la class DBcnx qui se charge de la connexion avec base de donné , qui est static pour éviter d’instancier un objet à chaque on veux se connecter

public class DBcnx {

private static final String HOST ="127.0.0.1";

private static final int PORT = 3306;

private static final String DB\_NAME = "beispiel" ;

private static final String USERNAME= "root" ;

private static final String PASSWORD ="";

private static Connection connection ;

public DBcnx() {

}

public static Connection getConnection() throws SQLException{

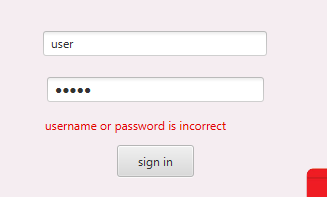
connection = DriverManager.getConnection(String.format("jdbc:mysql://%s:%d/%s", HOST, PORT, DB\_NAME), USERNAME, PASSWORD);

return connection ;}

la méthode isLogin() renvoie true, cela signifie que l'utilisateur a entré des informations de connexion valides. La méthode crée ensuite une nouvelle scène en chargeant le fichier FXML correspondant au tableau de bord (Dashboard.fxml). La scène est affichée dans la fenêtre principale de l'application , sinon un message d'erreur est affiché .

La méthode closeConnection() est utilisée pour fermer la connexion à la base de données une fois qu'elle n'est plus nécessaire. Cette méthode vérifie d'abord si la connexion est ouverte, puis la ferme si c'est le cas.

Le résultat comme suit :



Puis on accède au Dashboard



On peut d’ici accéder au d’autre view des produits , livreurs et commandes

La partie dernier commande pour afficher 3 dernières commandes , les revenues pour les revenues des commandes .

On a par exemple cette fonction lors ce qu’on clique sur livreur

public void toLivreur(Event e) throws IOException

{

Scene scene;

Node node = (Node)e.getSource();

Stage stage = (Stage)node.getScene().getWindow();

Parent root = FXMLLoader.load(getClass().getResource("/fxml/Livreur.fxml"));

scene = new Scene(root);

stage.setScene(scene);

stage.show();

}

La fonction pour calculer les revenues

public double calculer\_rev() throws SQLException {

Statement state = DBcnx.getConnection().createStatement();

ResultSet result = state.executeQuery("SELECT \* FROM commande\_produit");

double revenue =0 ;

while (result.next()) {

int idProd = result.getInt("idProd");

revenue += recPrix(idProd);

}

DBcnx.closeConnection();

return revenue;

}

La méthode itère ensuite sur le résultat de la requête en utilisant une boucle while et récupère l'ID du produit et appelle la méthode recPrix() pour récupérer le prix de chaque produit. La méthode ajoute ensuite le prix de chaque produit au total des revenus.

la méthode ferme la connexion à la base de données et renvoie le montant total des revenus générés par la vente des produits de commande.

public ObservableList<Commande> get3Cmd() throws SQLException {

ObservableList<Commande> commande = FXCollections.observableArrayList();

Statement state = DBcnx.getConnection().createStatement();

ResultSet result = state.executeQuery("SELECT \* FROM commande ORDER BY num\_cmd DESC LIMIT 3");

while(result.next()) {

Commande cmd = new Commande();

cmd.setId\_commande(result.getInt(1));

cmd.setAdresse\_commande(result.getString(2));

cmd.setCin\_livreur(result.getString(3));

commande.add(cmd);

}

DBcnx.closeConnection();

return commande;

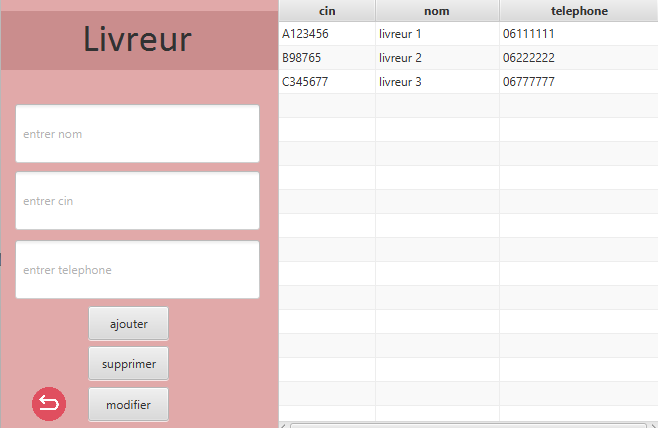
}

la requête SQL sélectionne toutes les entrées de la table "commande" triées par numéro de commande en ordre décroissant et limite les résultats aux trois premiers.

La méthode utilise ensuite une boucle while pour parcourir chaque ligne du résultat de la requête, créer un objet Commande, y stocker les données de chaque colonne de la table, puis ajouter l'objet Commande créé à une liste ObservableList.

Enfin, la méthode ferme la connexion à la base de données et renvoie la liste ObservableList contenant les trois dernières commandes.

Pour la partie livreur



Une a une partie CRUD pour ajouter , supprimer et modifier les livreurs le tableau à coté pour afficher les livreurs

public void ajouter\_liv(Event e) throws SQLException

{

livreur.setNom\_livreur(txtNom\_liv.getText());

livreur.setCin\_livreur(txtCin\_liv.getText());

livreur.setTelephone(txtTele\_liv.getText());

lv.ajouter(livreur);

UpdateTable();

cor.setText("enregistrer avec succees !");

}

la méthode récupère les informations du nouveau livreur à partir des champs de texte du formulaire et les enregistre dans l'objet Livreur correspondant.

utilise ensuite la méthode ajouter de l'objet Livreurdao pour ajouter le nouvel objet Livreur dans la base de données, la méthode UpdateTable pour mettre à jour l'affichage de la table des livreurs.

public void ajouter(Livreur l) throws SQLException

{

String request = "insert into livreur (nom , cin ,telephone) values (? , ? ,?)";

Connection connection= DBcnx.getConnection();

preparedStatement = connection.prepareStatement(request);

preparedStatement.setString(1 , l.getNom\_livreur());

preparedStatement.setString(2 , l.getCin\_livreur());

preparedStatement.setString(3 , l.getTelephone() );

preparedStatement.execute();

}

la méthode ajouter prépare la requête SQL en utilisant la méthode "prepareStatement" de l'objet Connection, qui compile la requête SQL , elle affecte ensuite les valeurs des attributs de l'objet Livreur aux paramètres de la requête SQL à l'aide des méthodes "setString" dans les colonnes corespondantes de la requette .

pour supprimer

public void supprimer\_liv(Event e) {

btnDelete\_liv.setOnMouseClicked(event -> {

if (event.getClickCount() == 1) {

Livreur selectedLivreur = tableLiv.getSelectionModel().getSelectedItem();

try {

lv.supprimer(selectedLivreur);

tableLiv.getItems().remove(selectedLivreur); // Supprime la ligne correspondante de la table

} catch (SQLException ex) {

Logger.getLogger(LivreurController.class.getName()).log(Level.SEVERE, null, ex);

}

}

});

cor.setText("Supprimé avec succès !");

}

la méthode utilise l'événement "setOnMouseClicked" pour détecter le nombre de clics sur le bouton et récupérer le livreur sélectionné dans la table. Ensuite, elle appelle la méthode "supprimer" de la classe "Livreurdao" pour supprimer le livreur de la base de données et remove(selectedLivreur) pour supprimer également la ligne correspondante de la table

public void supprimer(Livreur l) throws SQLException

{

String request = "delete from livreur where cin = ? ";

Connection connection= DBcnx.getConnection();

preparedStatement = connection.prepareStatement(request);

preparedStatement.setString(1 , l.getCin\_livreur());

preparedStatement.execute();

}

Supprimer prépare une requête SQL de suppression avec delete et le critère de sélection where cin = ?. Ensuite, elle associe le paramètre cin de la requête avec le cin de l'objet Livreur en utilisant la méthode setString(), puis elle exécute la requête à l'aide de la méthode execute().

Pour modifier :

public void modifier\_liv(Event e)

{

btnModify\_liv.setOnMouseClicked(event->{

if(event.getClickCount()==1)

{

Livreur selectedLivreur = tableLiv.getSelectionModel().getSelectedItem() ;

Livreur newLivreur = new Livreur();

newLivreur.setCin\_livreur(txtCin\_liv.getText());

newLivreur.setNom\_livreur(txtNom\_liv.getText());

newLivreur.setTelephone(txtTele\_liv.getText());

try {

//cor.setText( selectedLivreur.getNom\_livreur());

lv.modifier(selectedLivreur, newLivreur);

UpdateTable();

} catch (SQLException ex) {

Logger.getLogger(LivreurController.class.getName()).log(Level.SEVERE, null, ex);

}

}

});

cor.setText("modifier avec succees !");

}

La méthode commence par récupérer l'objet Livreur sélectionné dans une TableView et créer un nouvel objet Livreur avec les nouvelles informations entrées dans des champs de texte (nom, cin, téléphone).

Ensuite, la méthode appelle la méthode modifier() de la classe Livreurdao en lui passant l'objet Livreur sélectionné et le nouvel objet Livreur comme paramètres.

Enfin, la méthode appelle la méthode UpdateTable() pour mettre à jour la TableView affichée à l'écran avec les nouvelles informations de la base de données

public void modifier(Livreur l,Livreur lNew) throws SQLException

{

String request = "UPDATE livreur SET nom = ? , cin = ? , telephone=? WHERE cin = ? ";

Connection connection= DBcnx.getConnection();

preparedStatement = connection.prepareStatement(request);

preparedStatement.setString(1 , lNew.getNom\_livreur());

preparedStatement.setString(2 , lNew.getCin\_livreur());

preparedStatement.setString(3 , lNew.getTelephone() );

preparedStatement.setString(4 , l.getCin\_livreur());

preparedStatement.execute();

}

Cette méthode permet de modifier un livreur existant dans la base de données. Elle prend deux objets Livreur en entrée : le premier représente le livreur existant que l'on souhaite modifier, et le deuxième représente le nouveau livreur avec les informations mises à jour. La méthode utilise une requête SQL de type UPDATE pour mettre à jour les informations du livreur dans la base de données en utilisant son identifiant (cin)

Pour afficher les livreurs

@Override

public void initialize(URL url, ResourceBundle rb) {

Livreurdao livreurDAO = new Livreurdao();

// récupérer tous les livreurs de la base de données

ObservableList<Livreur> livreurList = null;

try {

livreurList = livreurDAO.getAll();

} catch (SQLException ex) {

Logger.getLogger(LivreurController.class.getName()).log(Level.SEVERE, null, ex);

}

tabCin.setCellValueFactory(new PropertyValueFactory<>("cin\_livreur"));

tabNom.setCellValueFactory(new PropertyValueFactory<>("nom\_livreur"));

tabTele.setCellValueFactory(new PropertyValueFactory<>("telephone"));

tableLiv.setItems(livreurList);

}

Dans cette méthode, la classe Livreurdao est instanciée pour permettre de récupérer tous les livreurs de la base de données avec la méthode getAll(). Ces données sont ensuite associées aux différentes colonnes de la TableView à l'aide de la méthode setCellValueFactory(). Enfin, les données sont ajoutées à la TableView à l'aide de la méthode setItems().

public ObservableList<Livreur> getAll() throws SQLException

{

ObservableList livreur = FXCollections.observableArrayList();

state = DBcnx.getConnection().createStatement() ;

ResultSet result = state.executeQuery("select \* from livreur");

while(result.next())

{

Livreur liv = new Livreur();

liv.setNom\_livreur(result.getString(1));

liv.setCin\_livreur(result.getString(2));

liv.setTelephone(result.getString(3));

livreur.add(liv);

}

DBcnx.closeConnection();

return livreur ;

}

La méthode getAll() permet de récupérer tous les livreurs stockés dans la base de données et de les stocker dans une liste observable, en utilisant la requette select \*

private void UpdateTable() {

Livreurdao livreurDAO = new Livreurdao();

ObservableList<Livreur> livreurList = null;

try {

livreurList = livreurDAO.getAll();

} catch (SQLException ex) {

Logger.getLogger(LivreurController.class.getName()).log(Level.SEVERE, null, ex);

}

tabCin.setCellValueFactory(new PropertyValueFactory<>("cin\_livreur"));

tabNom.setCellValueFactory(new PropertyValueFactory<>("nom\_livreur"));

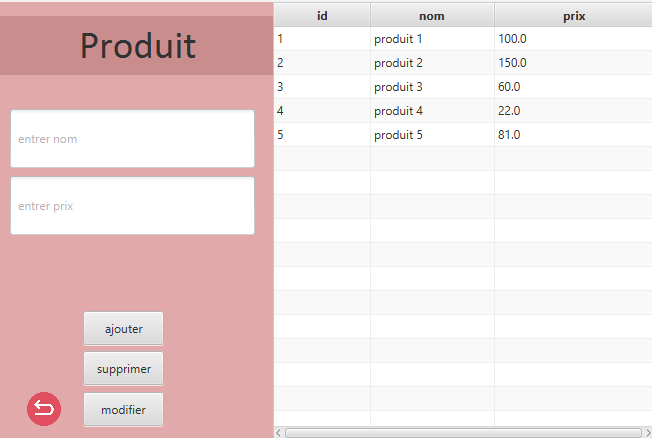
tabTele.setCellValueFactory(new PropertyValueFactory<>("telephone"));

tableLiv.setItems(livreurList);

}

On utilise cette méthode qui récupère tous les livreurs de la base de données pour actualiser la liste si on effectue des modifications

Pour la partie produit



Une a une partie CRUD pour ajouter , supprimer et modifier les livreurs le tableau à coté pour afficher les produits en utilisant les même méthodes utilisé dans livreur

Exemple code d’affichage :

Produitdao livreurDAO = new Produitdao();

ObservableList<Produit> produitList = null;

try {

produitList = livreurDAO.getAll();

} catch (SQLException ex) {

Logger.getLogger(LivreurController.class.getName()).log(Level.SEVERE, null, ex);

}

tabId.setCellValueFactory(new PropertyValueFactory<>("id\_produit"));

tabNom.setCellValueFactory(new PropertyValueFactory<>("nom\_produit"));

tabPrix.setCellValueFactory(new PropertyValueFactory<>("prix\_produit"));

tableprod.setItems(produitList); }

on instansier un nouveau objet de Produitdao, puis la méthode getAll() est appelée pour récupérer la liste de produits à partir de la base de données. Si une exception de type SQLException est levée, elle est gérée et une trace de l'erreur est enregistrée.

Ensuite, les valeurs de chaque propriété de chaque produit de la liste sont associées aux colonnes d'un tableau à l'aide de la méthode setCellValueFactory() de la classe PropertyValueFactory. Enfin, la liste de produits est affichée dans un tableau à l'aide de la méthode setItems() de la classe TableView.

Description de la méthode getAll()

public ObservableList<Produit> getAll() throws SQLException

{

ObservableList produit = FXCollections.observableArrayList();

state = DBcnx.getConnection().createStatement() ;

ResultSet result = state.executeQuery("select \* from produit");

while(result.next())

{

Produit prd = new Produit();

prd.setId\_produit(result.getInt(1));

prd.setNom\_produit(result.getString(2));

prd.setPrix\_produit(result.getFloat(3));

produit.add(prd);

}

DBcnx.closeConnection();

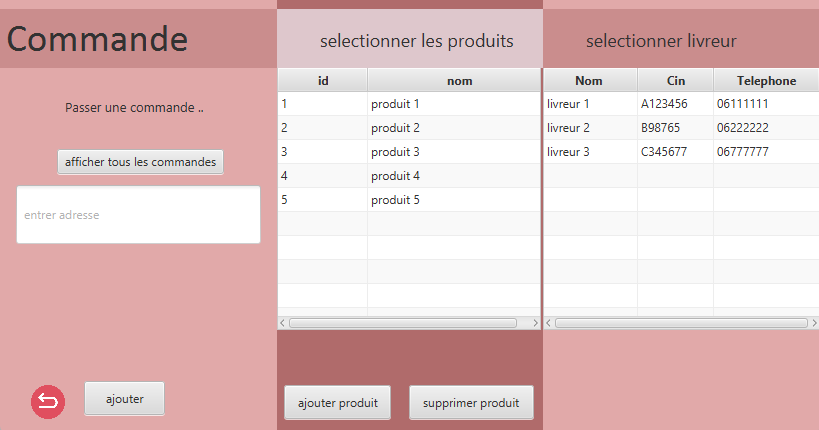
return produit ;

}

on a premièrement une connexion à la base de données en appelant la méthode getConnection() de la classe DBcnx .Ensuite, la méthode executeQuery() est utilisée pour exécuter une requête SQL qui récupère tous les enregistrements de la table "produit". Les résultats de la requête sont stockés dans un objet ResultSet.

la boucle while crée pour chaque enregistrement du ResultSet, un nouvel objet Produit est associe les attributs par sa collone correspondantes Enfin, l'objet Produit est ajouté à la liste ObservableList à l'aide de la méthode add().

Pour la partie commande



Dans la partie commande on peut passer une nouvelle commande en entrant l’adresse et en choisissant un livreur , puis on passe à remplir la commande en ajoutant ou supprimant un produit d’après la liste des produits à coté .

On affiche tous les commandes si on clique sur le bouton afficher tous les commande

public void ajouter\_cmd(Event e) throws SQLException {

Livreur selectedLivreur = tabLiv.getSelectionModel().getSelectedItem();

if (selectedLivreur == null) {

msg.setText("Veuillez sélectionner un livreur.");

} else {

commande.setAdresse\_commande(adresse\_cmd.getText());

commande.setCin\_livreur(selectedLivreur.getCin\_livreur());

cd.ajouter(commande);

cor.setText(Integer.toString(commande.getId\_commande()));

msg.setText("Commande enregistrée avec succès !");

cin\_cmd.setText(selectedLivreur.getCin\_livreur());

}

}

Pour créer une commande on doit sélectionner un livreur sinon un message d’erreur va s’afficher

La méthode "getSelectionModel()" récupère le modèle de sélection de TableView et "getSelectedItem()" récupère l'élément sélectionné dans le tableau. Si aucun élément n'est sélectionné, la méthode affiche un message d'erreur pour demander à l'utilisateur de sélectionner un livreur. Ensuite en utilise la méthode ajouter de la class Commandedao .

public void ajouter(Commande c) throws SQLException

{

String request = "insert into commande (adresse\_cmd, cin\_livreur) values (? , ? )";

Connection connection= DBcnx.getConnection();

PreparedStatement preparedStatement = connection.prepareStatement(request, Statement.RETURN\_GENERATED\_KEYS);

preparedStatement.setString(1 ,c.getAdresse\_commande());

preparedStatement.setString(2 ,c.getCin\_livreur() );

preparedStatement.execute();

ResultSet generatedKeys = preparedStatement.getGeneratedKeys();

if (generatedKeys.next()) {

int id = generatedKeys.getInt(1);

c.setId\_commande(id);

}

}

on établit une connexion à la base de données en appelant la méthode "getConnection()" de la classe "DBcnx", puis une requête SQL est préparé en créant un objet de type "PreparedStatement" on passe également un paramètre "Statement.RETURN\_GENERATED\_KEYS" pour renvoyer les clés auto-incrémente . l’étape suivante consiste à remplir les valeurs de la commande à insérer en utilisant les méthodes getAdresse\_commande() et getCin\_livreur() de l'objet Commande puis la requête SQL est exécutée en appelant la méthode execute() ,La dernière étape consiste à récupérer l'identifiant généré automatiquement pour cette commande en appelant la méthode getGeneratedKeys() de l'objet PreparedStatement. Ensuite, on extrait l'identifiant de la commande à partir du résultat de cette requête SQL en appelant la méthode getInt(1) de l'objet "ResultSet" et définit l'identifiant de la commande nouvellement insérée en appelant la méthode setId\_commande(id) de l'objet Commande.

public void remplir\_prod() {

int num = Integer.parseInt(cor.getText());

btn\_remplir.setOnAction(event -> {

Produit selectedProduit = tableprod.getSelectionModel().getSelectedItem();

try {

cd.remplir(num, selectedProduit.getId\_produit(), selectedProduit.getNom\_produit());

} catch (SQLException ex) {

Logger.getLogger(CommandeController.class.getName()).log(Level.SEVERE, null, ex);

}

});

}

on commence par extraire l'identifiant de la commande à partir du champ cor qui contient l'identifiant de la commande courante sous forme de texte. L'identifiant est converti en entier en utilisant la méthode "Integer.parseInt(cor.getText())".Lorsqu'on sélectionne un produit dans la table "tableprod", le code extrait l'objet "Produit" correspondant en utilisant la méthode "getSelectedItem()" de la table.Ensuite, la méthode "cd.remplir()" est appelée pour ajouter ce produit à la commande

public void remplir(int num ,int id , String nom) throws SQLException

{

String request = "insert into commande\_produit (numCmd, idProd , nomProd) values (? , ?,? )";

Connection connection= DBcnx.getConnection();

PreparedStatement preparedStatement = connection.prepareStatement(request);

preparedStatement.setInt(1, num);

preparedStatement.setInt(2, id);

preparedStatement.setString(3, nom);

preparedStatement.execute();

}

Finalement l’interface d’affichage des commandes et les produits



On a une partie qui affiche toutes les commandes , on peut supprimer une commande la partie à droite pour afficher les produits de la commande sélectionné .

public void supprimer\_cmd(Event e) throws SQLException {

Commande selectedCommande = tableCmd.getSelectionModel().getSelectedItem();

if (selectedCommande != null) {

int selectedCommandeId = selectedCommande.getId\_commande();

try {

cd.supprimer(selectedCommandeId);

tableCmd.getItems().remove(selectedCommande);

} catch (SQLException ex) {

Logger.getLogger(AfficheCommandeController.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

Cette fonction récupère le id de la commande sélectionné puis le renvoie dans la fonction de supprimer()

public void supprimer(int p) throws SQLException {

Connection connection = DBcnx.getConnection();

try {

connection.setAutoCommit(false);

String deleteCommandeProduitQuery = "DELETE FROM commande\_produit WHERE numCmd=?";

PreparedStatement deleteCommandeProduitStatement = connection.prepareStatement(deleteCommandeProduitQuery);

deleteCommandeProduitStatement.setInt(1, p);

deleteCommandeProduitStatement.executeUpdate();

String deleteCommandeQuery = "DELETE FROM commande WHERE num\_cmd=?";

PreparedStatement deleteCommandeStatement = connection.prepareStatement(deleteCommandeQuery);

deleteCommandeStatement.setInt(1, p);

deleteCommandeStatement.executeUpdate();

connection.commit();

} catch (SQLException ex) {

connection.rollback();

throw ex;

} finally {

connection.setAutoCommit(true);

}

La méthode crée deux requêtes DELETE : une pour supprimer les produits associés à la commande, et l'autre pour supprimer la commande elle-même. Ces requêtes sont préparées en utilisant un objet PreparedStatement et sont exécutées avec succès en appelant la méthode executeUpdate().

Si les deux suppressions sont exécutées sans erreur, la méthode valide la transaction en appelant la méthode commit() sur l'objet de connexion. Cependant, si une exception SQLException est levée, la méthode effectue un rollback() de la transaction et propage l'exception vers l'appelant. Enfin, la méthode réactive l'autocommit en appelant setAutoCommit(true) sur l'objet de connexion.

public void toReturn(Event e) throws IOException

{

Scene scene;

Node node = (Node)e.getSource(); // emplcement dyal button

Stage stage = (Stage)node.getScene().getWindow(); //stage

Parent root = FXMLLoader.load(getClass().getResource("/fxml/Commande.fxml"));

scene = new Scene(root);

stage.setScene(scene);

stage.show();

}

Finalement on trouve cette fonction qui nous permet de retourner à l’interface des commandes en cliquant sur la flèche au dessous de la page.



**conclusion :**

En conclusion, ce projet de fin de module sur la création d'une application de suivi des livreurs a été une expérience très enrichissante et interessante. Il m'a permis de développer mes compétences en programmation Java et en gestion de bases de données SQL, ainsi m'a permis de renforcer mes connaissances dans le domaine de la programmation orientée objet et de l'interface utilisateur JavaFX.