
RAPPORT DU PROJET

Application desktop java : Tracking des Livreurs

Réalisé par :

BOULBEN Firdaous
(Groupe 2)

Encadré par :

- Prof. E. M. EN-NAIMI
- Prof. L. ELAACHAK

Sommaire

Sommaire	1
Liste des figures.....	2
Introduction	3
Fonctionnalités.....	3
Outils	3
Présentation de l'application.....	4
I. La classe BaseDAO :	4
II. L'interface d'authentification :	5
III. L'interface de Menu Principal :	8
IV. L'interface de Gestion des livreurs :	9
V. L'interface de Gestion des produits :	14
VI. L'interface de Gestion des commandes :	15
VII. L'interface de Produits - Commandes :	19
Base de données	23
I. La table admin :	23
II. La table livreur :	23
III. La table produit :	24
IV. La table commande :	24
V. La table commande_produit :	25
Annexes.....	26
Code source de l'application.....	26
Vidéo démonstrative de toute l'application	26

Liste des figures

Figure 1: Interface d'authentification	5
Figure 2: Erreur lors de l'authentification	7
Figure 3: Interface de Menu Principal	8
Figure 4: Interface de Gestion des livreurs	9
Figure 5: Remplissage des champs par les valeurs de l'élément sélectionné	13
Figure 6: Interface de Gestion des produits	14
Figure 7: Interface de Gestion des commandes	15
Figure 8: Chargement des noms des livreurs dans ComboBox d'après bdd.....	16
Figure 9: Interface de Produits - Comandes	19
Figure 10: Chargement des id des commandes dans ComboBox d'après bdd	20
Figure 11: Alert informant le succès de l'opération de création d'une commande avec ses produits.....	22
Figure 12: Base de données "tracking"	23
Figure 13: Table "admin"	23
Figure 14: Table "livreur"	24
Figure 15: Table "produit"	24
Figure 16: Table "commande"	25
Figure 17: Table "commande_produit"	25

Introduction

L'objectif principal de ce projet est de mettre en place une application desktop java, basée sur JDBC et JavaFX. L'application permet de gérer les livreurs, les commandes, et les produits.

L'application offre une interface d'authentification qui donne, après vérification des données, accès à l'utilisateur au menu principale qui lui permet de naviguer facilement entre les différentes fonctionnalités.

Les utilisateurs de l'application peuvent ajouter, modifier et supprimer des livreurs, des commandes et des produits et d'assigner également les commandes aux livreurs et les produits aux commandes.

Fonctionnalités

- Système d'authentification avec login et mot de passe.
- Menu principale permettant l'accès aux différents espaces de l'application et la déconnexion.
- Gestion des Livreurs : Ajouter, afficher, modifier et supprimer (CRUD).
- Gestion des Produits : Ajouter, afficher, modifier et supprimer (CRUD).
- Gestion des Commandes : Ajouter, afficher, modifier et supprimer (CRUD).
- Affectation des commandes aux livreurs.
- Affectation des produits aux commandes.

Outils

- Java
- JavaFX
- JDBC
- Mysql
- Scene Builder
- IntelliJ IDEA

Présentation de l'application

L'application a été conçue en utilisant une approche MVC pour assurer une séparation claire entre les différentes couches logiques, avec des modèles implémentés en utilisant JDBC pour assurer la connexion avec la base de données, et des interfaces utilisateur développées en utilisant JavaFX.

I. La classe BaseDAO :

On a créé tout d'abord une classe abstraite BaseDAO qui fournit une base pour les opérations de base de données dans le cadre d'un modèle DAO (Data Access Object) pour une entité donnée. Elle établit la connexion à la base de données et fournit des méthodes abstraites à implémenter.

```
public abstract class BaseDAO <T>{

    protected Connection connection ;
    protected Statement statement ;
    protected PreparedStatement preparedStatement;
    protected ResultSet resultSet ;

    // connexion avec bdd
    private String url = "jdbc:mysql://127.0.0.1:3306/tracking";
    private String login = "root";
    private String password = "";

    BaseDAO() throws SQLException {
        this.connection = DriverManager.getConnection(url , login
,password );
    }

    public abstract void save( T object ) throws SQLException;
    public abstract void update( T object, Long id) throws SQLException
;

    public abstract void delete( T object, Long id) throws SQLException
;

    public abstract List<T> getAll( ) throws SQLException ;

}
```

II. L'interface d'authentification :

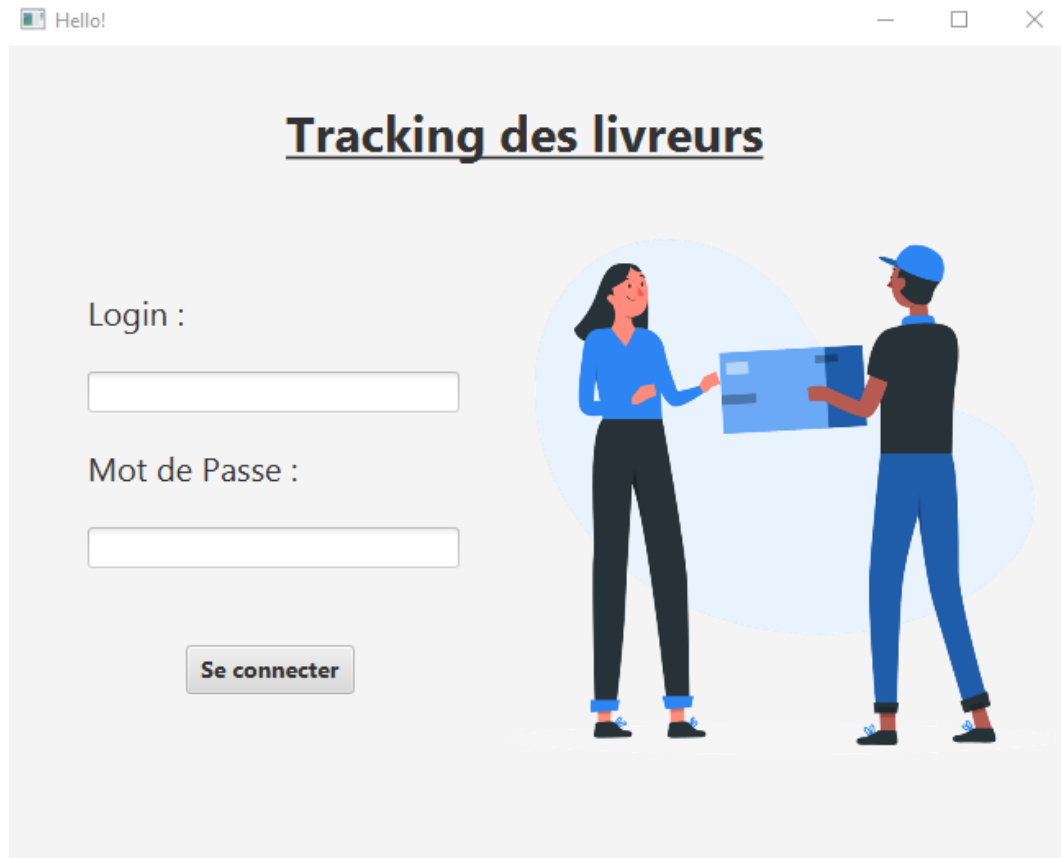


Figure 1: Interface d'authentification

On a créé tout d'abord le fichier FXML (**login-view.fxml**) avec un Label pour le nom de l'application, une ImageView et deux champs à remplir qui correspondent respectivement au login et le mot de passe, ainsi qu'on a ajouté un bouton qui permet, suite à son clic, la vérification de la correspondance entre les données fournis par l'utilisateur et celles qui existent dans notre base de données. Et ceci en utilisant la méthode suivante :

```
public static void login(ActionEvent event, String login, String password){
    Connection connection = null;
    PreparedStatement preparedStatement = null;
    ResultSet resultSet = null;

    try {
        connection =
        DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/tracking", "root",
        "");
        preparedStatement = connection.prepareStatement("select
        password from admin where login = ?");
        preparedStatement.setString(1, login);
        resultSet = preparedStatement.executeQuery();

        if (login.isEmpty() || password.isEmpty()){
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("Login Erreur");
            alert.setContentText("Veuillez remplir tous les champs.");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

```

        alert.show();
    } else if (!resultSet.isBeforeFirst()) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Login Erreur");
        alert.setContentText("Utilisateur introuvable.");
        alert.show();
    } else {
        while (resultSet.next()) {
            String retrievedPassword =
resultSet.getString("password");
            if (retrievedPassword.equals(password)) {
                switchToHome(event);
            } else {
                Alert alert = new Alert(Alert.AlertType.ERROR);
                alert.setTitle("Login Erreur");
                alert.setContentText("Mot de passe incorrecte.");
                alert.show();
            }
        }
    }

} catch (SQLException e) {
    throw new RuntimeException(e);
}

}
}

```

Si les données sont erronées, on affiche un alert, sinon on bascule sur l'interface de menu principale (**hello-view.fxml**) grâce à la méthode suivante :

```

public static void switchToHome(ActionEvent event) {
    try {
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(new
File("C:\\Users\\asus\\Desktop\\TrackingL\\src\\main\\resources\\ma\\fstt\\
trackingl\\hello-view.fxml").toURI().toURL());
        Parent homeParent = loader.load();
        Scene homeScene = new Scene(homeParent);
        Stage window = (Stage) ((Node)
event.getSource()).getScene().getWindow();
        window.setScene(homeScene);
        window.show();
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}

```

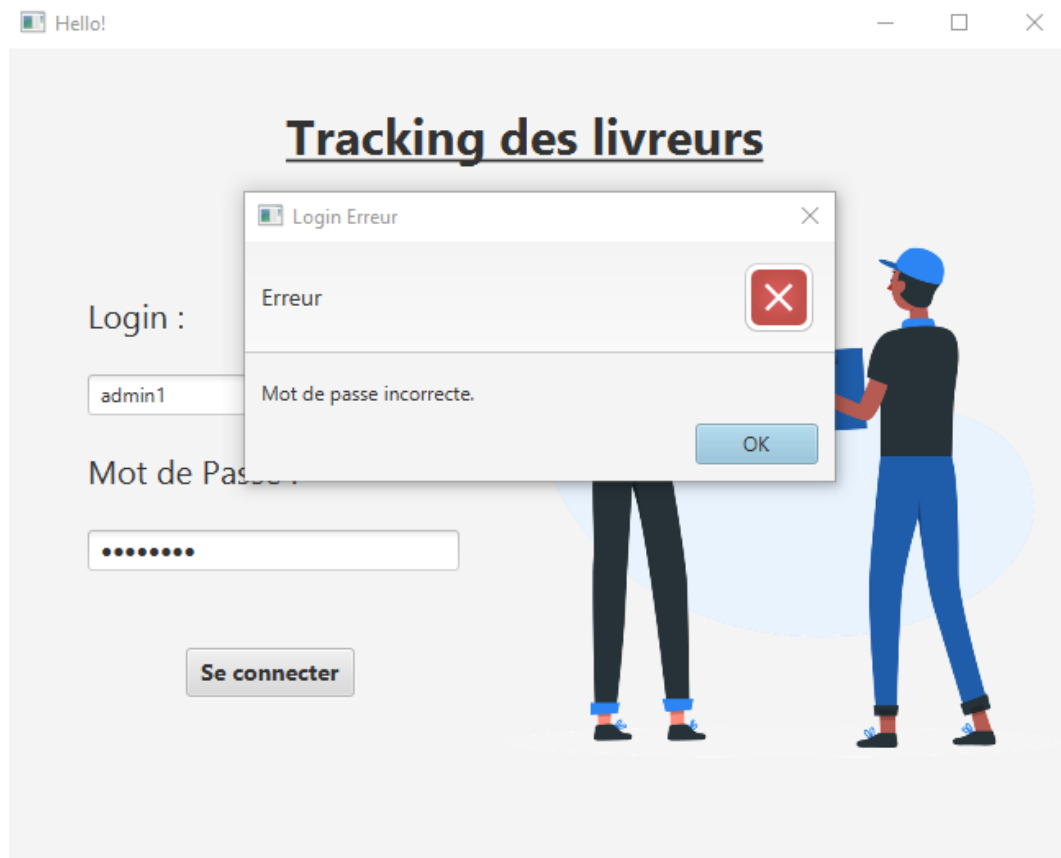


Figure 2: Erreur lors de l'authentification

On crée une classe **LoginApplication** qui étend la classe abstraite `Application` de JavaFX et qui permet grâce à la méthode `start()` de charger le fichier FXML et gérer la création de la fenêtre et son affichage à l'écran, et de lancer finalement l'application JavaFX en utilisant la méthode `main()`.

```
public class LoginApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(LoginApplication.class.getResource("login-view.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 650, 500);
        stage.setTitle("Hello!");
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch();
    }
}
```

On crée également la classe **LoginController** qui implémente l'interface `Initializable` de JavaFX et qui fournit une méthode `initialize()` appelée automatiquement après que le fichier FXML a été chargé.

```
public class LoginController implements Initializable {
    @FXML
```



```

private Button btnLogin;

@FXML
private TextField tf_login ;

@FXML
private PasswordField tf_mdp ;

@Override
public void initialize(URL url, ResourceBundle resourceBundle) {
    btnLogin.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) {
            Login.logIn(event, tf_login.getText(), tf_mdp.getText());
        }
    });
}
}

```

III. L'interface de Menu Principal :

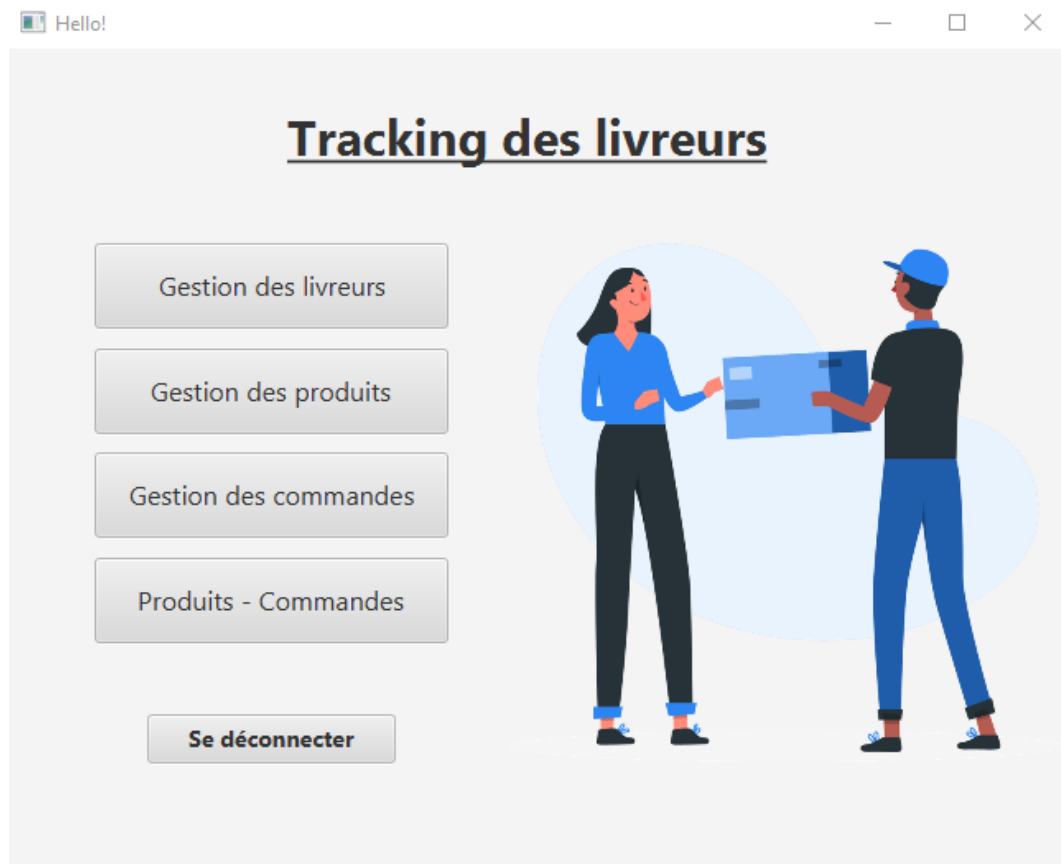


Figure 3: Interface de Menu Principal

De même pour cette interface, on a créé tout d'abord le fichier FXML (**hello-view.fxml**) avec un Label, une ImageView et trois boutons permettant de naviguer entre les différentes interfaces de l'application : Gestion des livreurs, Gestion des produits et Gestion des commandes et Produits-Commandes. Ainsi qu'on a ajouté un bouton qui permet de se déconnecter et basculer alors de nouveau vers l'interface de

l'authentification. Le suivant est un exemple de la méthode permettant cette navigation entre les scènes :

```
public void switchToLogin(ActionEvent event) throws IOException {  
    FXMLLoader loader = new FXMLLoader(getClass().getResource("login-  
view.fxml"));  
    Parent commandeParent = loader.load();  
    Scene commandeScene = new Scene(commandeParent);  
    Stage window = (Stage) root.getScene().getWindow();  
    window.setTitle("Connexion");  
    window.setScene(commandeScene);  
    window.show();  
}
```

On crée également, une classe **HelloApplication** qui étend la classe abstraite Application de JavaFX et qui contient les deux méthodes start() et main() qui chargent le fichier FXML, créent la fenêtre et lancent l'application JavaFX. Et une classe **HelloController** qui est un contrôleur qui gère les actions principales de cette interface graphique et qui définit les quatre méthodes switchToLivreur(), switchToProduit(), switchToCommande(), switchToCmdProd() et switchToLogin() qui sont appelées lorsqu'on clique sur les boutons correspondants.

IV. L'interface de Gestion des livreurs :

Id	Nom	Téléphone	Nombre de commandes	
1	Livreur 1	0611223344	2	
2	Livreur 2	0622334455	0	
3	Livreur 3	0633445566	1	
4	Livreur 4	0644556677	1	

Figure 4: Interface de Gestion des livreurs

On a créé tout d'abord le fichier FXML (**liv-view.fxml**) avec un Label indiquant le titre et un espace CRUD qui contient un formulaire d'ajout des livreurs permettant l'enregistrement des données : Nom, Téléphone et le Nombre de commandes associé, et des boutons permettant la réalisation des différents opérations : Ajouter, Modifier, Supprimer et Effacer. Ainsi qu'un TableView qui affiche la liste de tous les livreurs et un bouton qui permet de retourner vers le Menu Principal.

On crée ensuite deux classe **Livreur** qui définit notre classe avec ses attributs : id_livreur, nom, téléphone et nb_commande, les constructeurs, les accesseurs, les mutateurs et la méthode toString, et la classe **LivreurDAO** qui extends la classe abstraite BaseDAO et qui implémente ses méthodes abstraites :

- **save** qui ajoute un nouvel enregistrement :

```
public void save(Livreur object) throws SQLException {
    String request = "insert into livreur (nom , telephone) values (? , ?)";

    // mapping objet table
    this.preparedStatement = this.connection.prepareStatement(request);

    // mapping
    this.preparedStatement.setString(1 , object.getNom());
    this.preparedStatement.setString(2 , object.getTelephone());

    this.preparedStatement.execute();
}
```

- **update** qui modifie l'enregistrement correspondant à l'identifiant passé en paramètre:

```
public void update(Livreur object, Long id) throws SQLException {
    String request = "update livreur set nom = ? , telephone = ? where id_livreur = ?";

    // mapping objet table
    this.preparedStatement = this.connection.prepareStatement(request);

    // mapping
    this.preparedStatement.setString(1 , object.getNom());
    this.preparedStatement.setString(2 , object.getTelephone());
    this.preparedStatement.setLong(3 , id);

    this.preparedStatement.execute();
}
```

- **delete** qui supprime l'enregistrement correspondant à l'identifiant passé en paramètre:

```
public void delete(Livreur object, Long id) throws SQLException {
    String request = "delete from livreur where id_livreur = ?";

    // mapping objet table
    this.preparedStatement = this.connection.prepareStatement(request);
}
```

```

// mapping
this.preparedStatement.setLong(1 , id);

this.preparedStatement.execute();
}

```

- **getAll** qui retourne une liste de tous les enregistrements de la table "livreur" sous forme d'une liste d'objets Livreur :

```

public List<Livreur> getAll() throws SQLException {
    List<Livreur> mylist = new ArrayList<Livreur>();

    String request = "select * from livreur ";

    this.statement = this.connection.createStatement();

    this.resultSet = this.statement.executeQuery(request);

    // parcours de la table
    while ( this.resultSet.next()){
        // mapping table objet
        mylist.add(new Livreur(this.resultSet.getLong(1) ,
                                this.resultSet.getString(2) ,
                                this.resultSet.getString(3),
                                this.resultSet.getInt(4)));
    }

    return mylist;
}

```

On crée également la classe **LivApplication** qui lance cette interface graphique d'après le fichier FXML et la classe **LivController** qui est un contrôleur qui gère les événements. Elle appelle la méthode save() pour ajouter un nouvel enregistrement en utilisant la méthode suivante :

```

protected void onSaveButtonClick() {
    // access a la bdd
    try {
        LivreurDAO livreurDAO = new LivreurDAO();
        Livreur liv = new Livreur(01 , nom.getText() , tele.getText() ,
        Integer.parseInt(cmd.getText()));
        livreurDAO.save(liv);

        UpdateTable();
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}

```

Elle récupère les informations du livreur sélectionné dans la table et les affiche dans les champs de texte en utilisant la méthode suivant :

```

public void getData(javafx.scene.input.MouseEvent mouseEvent) {
    Livreur livreur = mytable.getSelectionModel().getSelectedItem();
}

```

```

        id = livreur.getId_livreur();
        nom.setText(livreur.getNom());
        tele.setText(livreur.getTelephone());
        cmd.setText(String.valueOf(livreur.getNb_commande()));

        btnSave.setDisable(true);
    }

```

afin de le modifier ou le supprimer en utilisant les méthodes suivantes :

```

protected void onUpdateButtonClick() {
    // access a la bdd
    try {
        LivreurDAO livreurDAO = new LivreurDAO();
        Livreur liv = new Livreur(01 , nom.getText() , tele.getText() ,
Integer.parseInt(cmd.getText()));
        livreurDAO.update(liv, id);

        UpdateTable();

    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}

```

```

protected void onDeleteButtonClick() {
    // access a la bdd
    try {
        LivreurDAO livreurDAO = new LivreurDAO();
        Livreur liv = new Livreur(01 , nom.getText() , tele.getText() ,
Integer.parseInt(cmd.getText()));
        livreurDAO.delete(liv, id);

        UpdateTable();

    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}

```

Figure 5: Remplissage des champs par les valeurs de l'élément sélectionné

On utilise la méthode suivante pour effacer les champs de texte et réactiver le bouton d'ajout afin de pouvoir ajouter des nouveaux enregistrements une autre fois :

```
protected void onClearButtonClick() {
    nom.setText(null);
    tele.setText(null);
    cmd.setText(null);
    btnSave.setDisable(false);
}
```

Enfin, pour afficher la liste des enregistrements dans la liste, on utilise les méthodes suivantes :

```
public static ObservableList<Livreur> getDataLivreurs() {
    LivreurDAO livreurDAO = null;

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

    try {
        livreurDAO = new LivreurDAO();
        for (Livreur ettemp : livreurDAO.getAll())
            listfx.add(ettemp);
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}
```

```

        return listfx ;
    }

```

```

public void UpdateTable() {
    col_id.setCellValueFactory(new
PropertyVFactry<Livreur, Long> ("id_livreur"));
    col_nom.setCellValueFactory(new
PropertyVFactry<Livreur, String> ("nom"));
    col_tele.setCellValueFactory(new
PropertyVFactry<Livreur, String> ("telephone"));
    col_cmd.setCellValueFactory(new
PropertyVFactry<Livreur, Integer> ("nb_commande"));

    mytable.setItems(this.getDataLivreurs());
}

```

```

public void initialize(URL location, ResourceBundle resources) {
    UpdateTable();
}

```

V. L'interface de Gestion des produits :

Id	Nom	Prix	Description
1	Huile de table Lesieur	91.8	Huile de table équilibrée 5L
2	Spaghetti Panzani	18.5	Spaghetti n°7 500g
3	Farine fleur Kenz	31.8	Farine fleur pâtisseries de blé tendre 5Kg
4	Couscous Dari	20.9	Couscous Fin 1kg
5	Sucre Saint Louis	52.0	Sucre petits morceaux spécial café 1Kg

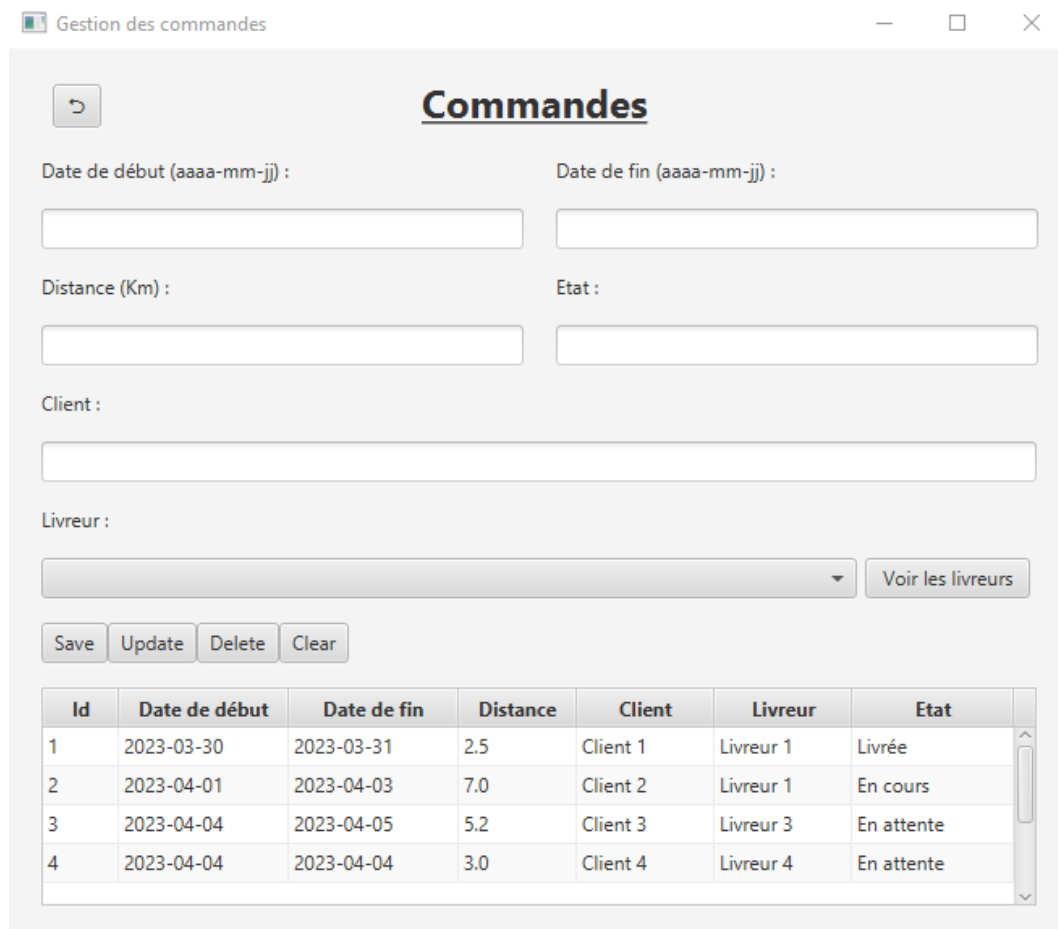
Figure 6: Interface de Gestion des produits

De même que la gestion des livreurs, on a créé tout d'abord le fichier FXML (**prod-view.fxml**) avec un Label indiquant le titre et un espace CRUD qui contient un formulaire d'ajout des produits permettant l'enregistrement des données : Nom, Prix et une Description du produit, et des boutons permettant la réalisation des différents

opérations de CRUD. Ainsi qu'un TableView qui affiche la liste de tous les produits et un bouton qui permet de retourner vers le Menu Principal.

On crée ensuite de même la classe **Produit** qui définit notre classe avec ses attributs, les constructeurs, les accesseurs, les mutateurs et la méthode toString, la classe **ProduitDAO** qui étend la classe abstraite BaseDAO et qui implémente ses méthodes abstraites, la classe **ProdApplication** qui lance cette interface graphique et le contrôleur **ProdController** qui gère les événements de clic des boutons et définit autres méthodes pour l'affichage de la liste des enregistrements.

VI. L'interface de Gestion des commandes :



Id	Date de début	Date de fin	Distance	Client	Livreur	Etat
1	2023-03-30	2023-03-31	2.5	Client 1	Livreur 1	Livrée
2	2023-04-01	2023-04-03	7.0	Client 2	Livreur 1	En cours
3	2023-04-04	2023-04-05	5.2	Client 3	Livreur 3	En attente
4	2023-04-04	2023-04-04	3.0	Client 4	Livreur 4	En attente

Figure 7: Interface de Gestion des commandes

De même, on a créé tout d'abord le fichier FXML (**cmd-view.fxml**) avec un Label indiquant le titre et un espace CRUD qui contient un formulaire d'ajout des produits permettant l'enregistrement des données : Date de début, Date de fin, Distance, Etat, Client et un ComboBox permettant de choisir le Livreur de cette commande parmi les livreurs existant dans notre base de données, et des boutons permettant la réalisation des différents opérations de CRUD. Ainsi qu'un TableView qui affiche la liste de tous les commandes et un bouton qui permet de retourner vers le Menu Principal.

On crée ensuite de même la classe **Commande** qui définit notre classe avec ses attributs, les constructeurs, les accesseurs, les mutateurs et la méthode toString, ainsi que la classe **CommandeDAO** permettant de définir les différentes méthodes qu'on va appeler éventuellement dans la classe de contrôleur :

- **load** qui prend un objet ComboBox en paramètre et le remplit avec les livreurs provenant de la base de données, en ajoutant le nom de chaque livreur en tant que nouvel élément dans le ComboBox :

```
public void load(ComboBox livreurs) throws SQLException {  
    String request = "select nom from livreur ";  
    this.statement = this.connection.createStatement();  
    this.resultSet = this.statement.executeQuery(request);  
    while (this.resultSet.next()) {  
        livreurs.getItems().add(resultSet.getString("nom"));  
    }  
}
```

The screenshot shows a Java Swing window titled "Gestion des commandes". The main content area is titled "Commandes". It contains several input fields: "Date de début (aaaa-mm-jj) :", "Date de fin (aaaa-mm-jj) :", "Distance (Km) :", "Etat :", and "Client :". Below these is a "Livreur :" label and a dropdown menu. The dropdown menu is open, showing a list of 10 couriers: "Livreur 1", "Livreur 2", "Livreur 3", "Livreur 4", "Livreur 5", "Livreur 6", "Livreur 7", "Livreur 8", "Livreur 9", and "Livreur 10". To the right of the dropdown is a button labeled "Voir les livreurs". Below the dropdown is another dropdown menu labeled "Etat" with options: "Livrée", "En cours", "En attente", and "En attente".

Figure 8: Chargement des noms des livreurs dans ComboBox d'après bdd

- **saveCmd** qui ajoute une nouvelle commande tout en modifiant le nombre de commandes associé aux livreurs:

```
public void saveCmd(Commande object, ComboBox livreurs) throws
SQLException {
    //Récupérer la valeur sélectionnée
    String selectedValue = (String) livreurs.getValue();
    object.setLivreur(selectedValue);

    PreparedStatement insertStmt = null;
    PreparedStatement updateStmt = null;

    try {
        this.connection.setAutoCommit(false);

        //Ajouter une nouvelle commande
        insertStmt = this.connection.prepareStatement("insert into
commande (date_debut, date_fin , distance, client, livreur, etat) values
(? , ? , ? , ? , ? , ?)");
        insertStmt.setString(1 , object.getDate_debut());
        insertStmt.setString(2 , object.getDate_fin());
        insertStmt.setFloat(3 , object.getDistance());
        insertStmt.setString(4 , object.getClient());
        insertStmt.setString(5 , object.getLivreur());
        insertStmt.setString(6 , object.getEtat());
        insertStmt.executeUpdate();

        //Mettre à jour le nombre de commandes du livreur sélectionné
        updateStmt = this.connection.prepareStatement("UPDATE livreur
SET nombre_commande = nombre_commande + 1 WHERE nom = ?");
        updateStmt.setString(1, selectedValue);
        updateStmt.executeUpdate();

        this.connection.commit();

    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

- **updateCmd** qui modifie la commande correspondante à l'identifiant passé en paramètre avec la mise à jour du nombre de commandes associé aux livreurs:

```
public void updateCmd(Commande object, Long id, ComboBox livreurs)
throws SQLException {
    //Récupérer la valeur sélectionnée
    String selectedValue = (String) livreurs.getValue();
    object.setLivreur(selectedValue);

    PreparedStatement updateCmdStmt = null;
    PreparedStatement updateLivStmt = null;

    //Modifier une commande
    try {
        updateCmdStmt = this.connection.prepareStatement("update
commande set date_debut = ? , date_fin = ?, distance = ?, client = ?,
livreur = ?, etat = ? where id_commande = ?");
        updateCmdStmt.setString(1 , object.getDate_debut());
        updateCmdStmt.setString(2 , object.getDate_fin());
```

```

        updateCmdStmt.setFloat(3 , object.getDistance());
        updateCmdStmt.setString(4 , object.getClient());
        updateCmdStmt.setString(5 , object.getLivreur());
        updateCmdStmt.setString(6 , object.getEtat());
        updateCmdStmt.setLong(7 , id);
        updateCmdStmt.executeUpdate();

    } catch (SQLException e) {
        e.printStackTrace();
    }

    //Mettre à jour le nombre de commande associé à chaque livreur
    updateLivStmt = this.connection.prepareStatement("UPDATE livreur l
SET l.nombre_commande = (SELECT COUNT(*) FROM commande c WHERE c.livreur
= l.nom )");
    updateLivStmt.executeUpdate();
}

```

Et les méthodes **delete** pour supprimer une commande et **getAll** pour retourner la liste de toutes les commandes de même que celles déclarées auparavant.

On crée également la classe **CmdApplication** qui lance cette interface graphique d'après le fichier FXML et la classe **CmdController** qui est un contrôleur qui gère les événements de clic pour chaque bouton : `onLoadButtonClick`, `onSaveButtonClick`, `onUpdateButtonClick`, `onDeleteButtonClick` et `onClearButtonClick`, ainsi que les méthodes responsables pour l'affichage de la liste de toutes les commandes dans la `TableView`.

VII. L'interface de Produits - Commandes :

Id	Nom	Prix	Description
1	Huile de table Lesieur	91.8	Huile de table équilibrée 5L
2	Spaghetti Panzani	18.5	Spaghetti n°7 500g
3	Farine fleur Kenz	31.8	Farine fleur pâtissière de blé tendre 5Kg

Figure 9: Interface de Produits - Comandes

On a créé tout d'abord le fichier FXML (**cmd-prod.fxml**) avec un Label indiquant le titre, un ComboBox permettant d'afficher les identifiants de toutes les commandes effectuées, des champs de textes qui seront remplies par les informations relatives au produit qui a été sélectionné de la TableView et cette dernière qui affiche la liste de tous les produits dans le stock, ainsi que les boutons save et clear et un bouton qui permet de retourner vers le Menu Principal.

On crée ensuite la classe **CmdProd** qui définit notre classe avec ses attributs : **id_comande**, **id_produit** et la quantité, les constructeurs, les accesseurs, les mutateurs et la méthode **toString**, ainsi que la classe **CmdProdDAO** permettant de définir les deux méthodes **load** qui remplit le ComboBox avec les identifiants des commandes et **saveCmdProd** qui insère chaque commande avec le produit qu'on lui a associée et son quantité dans notre base de données :

```
public void load(ComboBox commandes) throws SQLException {  
    String request = "select id_comande from commande ";  
    this.statement = this.connection.createStatement();  
    this.resultSet = this.statement.executeQuery(request);  
}
```

```

while (this.resultSet.next()) {
    commandes.getItems().add(resultSet.getString("id_commande"));
}
}

```

Ajout des produits aux commandes

Ajouter des produits aux commandes

Commandes :

1
2
3
4
5
6
7
8

Voir les commandes

Description :

Save Clear

Id	Nom	Prix	Description
1	Huile de table Lesieur	91.8	Huile de table équilibrée 5L
2	Spaghetti Panzani	18.5	Spaghetti n°7 500g
3	Farine fleur Kenz	31.8	Farine fleur pâtisseries de blé tendre 5Kg

Figure 10: Chargement des id des commandes dans ComboBox d'après bdd

```

public void saveCmdProd(CmdProd object, ComboBox commandes, TextField
quant) throws SQLException {
    //Récupérer la valeur sélectionnée
    Long selectedValue = Long.parseLong(commandes.getValue().toString());
    object.setId_commande(selectedValue);
    object.setQuantite(Integer.parseInt(quant.getText()));

    String request = "INSERT INTO commande_produit (id_commande,
id_produit, quantite) VALUES (?, ?, ?)";

    this.preparedStatement = this.connection.prepareStatement(request);

    // mapping
    this.preparedStatement.setString(1 ,
String.valueOf(object.getId_commande()));
    this.preparedStatement.setString(2 ,
String.valueOf(object.getId_produit()));
    this.preparedStatement.setString(3 ,
String.valueOf(object.getQuantite()));

    this.preparedStatement.execute();
}

```

```

        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Commande créée avec succès!");
        alert.setContentText("Vous avez ajouté " + object.getQuantite() + " de
produit id=" + object.getId produit() + " à la commande id=" +
object.getId_commande());
        alert.show();
    }

```

Et on crée enfin la classe **CmdProdApplication** qui lance cette interface graphique d'après le fichier FXML et le contrôleur **CmdProdController** qui définit les méthodes responsables de l'affichage des informations relatives aux produits dans la TableView et de remplissage des champs par les valeurs correspondantes au produit sélectionné dans la liste et qui gère également les événements de clic des boutons en appelant les méthodes déjà définies :

```

protected void onLoadButtonClick() {
    // access a la bdd
    try {
        CmdProdDAO cmdProdDAO = new CmdProdDAO();

        cmdProdDAO.load(commandes);

    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}

```

```

protected void onSaveButtonClick() {
    // access a la bdd
    try {
        CmdProdDAO cmdProdDAO = new CmdProdDAO();

        CmdProd cmdProd = new CmdProd(01 , id,
Integer.parseInt(quant.getText()));

        cmdProdDAO.saveCmdProd(cmdProd, commandes, quant);

        UpdateTable();

    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}

```

Et on affiche éventuellement un alert informant l'utilisateur de l'application que les produits sont affectés avec succès à la commande :

Ajout des produits aux commandes

Ajouter des produits aux commandes

Commandes :

8

Produits :

Nom :

Sucre Saint Louis

Prix (DH) :

52.0

Quantité :

5

Description :

Sucre petits morceaux spécial café 1Kg

Save Clear

Voir les commandes

Commande créée avec succès!

Message

Vous avez ajouté 5 de produit id=5 à la commande id=8

OK

Id	Nom	Prix	Description
3	Farine fleur Kenz	31.8	Farine fleur pâtisserie de blé tendre 5Kg
4	Couscous Dari	20.9	Couscous Fin 1kg
5	Sucre Saint Louis	52.0	Sucre petits morceaux spécial café 1Kg

Figure 11: Alert informant le succès de l'opération de création d'une commande avec ses produits

Base de données

On crée une base de données en Mysql nommée : "**tracking**" et on crée cinq tables : "**admin**", "**livreur**", "**produit**", "**commande**" et "**commande_produit**".

Table	Action	Rows	Type	Collation	Size	Overhead
<input type="checkbox"/> admin	★ Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb4_general_ci	16.0 KiB	-
<input type="checkbox"/> commande	★ Browse Structure Search Insert Empty Drop	8	InnoDB	utf8mb4_general_ci	16.0 KiB	-
<input type="checkbox"/> commande_produit	★ Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb4_general_ci	16.0 KiB	-
<input type="checkbox"/> livreur	★ Browse Structure Search Insert Empty Drop	11	InnoDB	utf8mb4_general_ci	16.0 KiB	-
<input type="checkbox"/> produit	★ Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_general_ci	16.0 KiB	-
5 tables	Sum	30	InnoDB	utf8mb4_general_ci	80.0 KiB	0 B

Figure 12: Base de données "tracking"

I. La table admin :

C'est la table qui contient les données d'authentification des administrateurs qu'on les vérifie avec celles saisi dans l'interface d'authentification afin de décider à accorder la connexion ou pas. Elle contient 3 colonnes : "**id**", "**login**" et "**password**".

				id	login	password
<input type="checkbox"/>	Edit	Copy	Delete	1	admin1	password1
<input type="checkbox"/>	Edit	Copy	Delete	2	admin2	password2
<input type="checkbox"/>	Edit	Copy	Delete	3	admin3	password3

Figure 13: Table "admin"

II. La table livreur :

C'est la table qui contient les données de tous les livreurs de notre application. Elle contient 4 colonnes : "**id_livreur**", "**nom**", "**telephone**" et "**nombre_commande**" qu'on met à jour chaque fois qu'une commande s'est associée au livreur.




































					id_livreur	nom	telephone	nombre_commande	
<input type="checkbox"/>		Edit		Copy		Delete	1 Livreur 1	0611223344	2
<input type="checkbox"/>		Edit		Copy		Delete	2 Livreur 2	0622334455	0
<input type="checkbox"/>		Edit		Copy		Delete	3 Livreur 3	0633445566	1
<input type="checkbox"/>		Edit		Copy		Delete	4 Livreur 4	0644556677	1
<input type="checkbox"/>		Edit		Copy		Delete	5 Livreur 5	0655667788	1
<input type="checkbox"/>		Edit		Copy		Delete	6 Livreur 6	0666778899	1
<input type="checkbox"/>		Edit		Copy		Delete	7 Livreur 7	0677889911	1
<input type="checkbox"/>		Edit		Copy		Delete	8 Livreur 8	0688991122	1
<input type="checkbox"/>		Edit		Copy		Delete	9 Livreur 9	0699112233	0
<input type="checkbox"/>		Edit		Copy		Delete	10 Livreur 10	0600112233	0
<input type="checkbox"/>		Edit		Copy		Delete	11 Livreur 11	0611998877	0

Figure 14: Table "livreur"

III. La table produit :

C'est la table qui contient les informations relatives à chaque produit de notre application. Elle contient 4 colonnes : "**id_produit**", "**nom_produit**", "**prix**" et "**description**".

<div><div><div></div><div></div><div></div></div><div></div></div>							id_produit	nom_produit	prix	description	
<input type="checkbox"/>		Edit		Copy		Delete	1	Huile de table Lesieur	91.8	Huile de table équilibrée 5L	
<input type="checkbox"/>		Edit		Copy		Delete	2	Spaghetti Panzani	18.5	Spaghetti n°7 500g	
<input type="checkbox"/>		Edit		Copy		Delete	3	Farine fleur Kenz	31.8	Farine fleur pâtissière de blé tendre 5Kg	
<input type="checkbox"/>		Edit		Copy		Delete	4	Couscous Dari	20.9	Couscous Fin 1kg	
<input type="checkbox"/>		Edit		Copy		Delete	5	Sucre Saint Louis	52	Sucre petits morceaux spécial café 1Kg	

Figure 15: Table "produit"

IV. La table commande :

C'est la table qui contient les informations relatives à chaque commande effectuée. Elle contient 7 colonnes : "**id_commande**", "**date_debut**", "**date_fin**", "**distance**", "**client**", "**livreur**" qu'on récupère son nom d'après les valeurs de la colonne nom de la table livreur dans le ComboBox et "**description**".

























←T→		id_commande	date_debut	date_fin	distance	client	livreur	etat
<input type="checkbox"/>	 Edit	 Copy			1	2023-03-30	2023-03-31	2.5 Client 1 Livreur 1 Livrée
<input type="checkbox"/>	 Edit	 Copy			2	2023-04-01	2023-04-03	7 Client 2 Livreur 1 En cours
<input type="checkbox"/>	 Edit	 Copy			3	2023-04-04	2023-04-05	5.2 Client 3 Livreur 3 En attente
<input type="checkbox"/>	 Edit	 Copy			4	2023-04-04	2023-04-04	3 Client 4 Livreur 4 En attente
<input type="checkbox"/>	 Edit	 Copy			5	2023-04-05	2023-04-07	10 Client 5 Livreur 5 En attente
<input type="checkbox"/>	 Edit	 Copy			6	2023-04-06	2023-04-07	8 Client 6 Livreur 6 En attente
<input type="checkbox"/>	 Edit	 Copy			7	2023-04-07	2023-04-07	3.5 Client 7 Livreur 7 En attente
<input type="checkbox"/>	 Edit	 Copy			8	2023-04-07	2023-04-08	5.7 Client 8 Livreur 8 En attente

Figure 16: Table "commande"

V. La table commande produit :

C'est la table qui enregistre les ids des produits affectés à chaque comande avec l'id de cette dernière et la quantité de ce produit. Elle contient colonnes : "**id_commande**", "**id_produit**" et "**quantite**".

id_commande	id_produit	quantite
2	1	4
3	4	9
1	3	5
8	5	5

Figure 17: Table "commande_produit"

Annexes

Code source de l'application

<https://github.com/firdaous-boulben/Tracking-des-livreurs.git>

Vidéo démonstrative de toute l'application

<https://photos.app.goo.gl/iAJYwV9EuaK6DSGM9>