



ÉCOLE NORMALE SUPÉRIEURE DE L'ENSEIGNEMENT  
TECHNIQUE DE MOHAMMEDIA  
UNIVERSITÉ HASSAN II DE CASABLANCA

# ÉCOLE NORMALE SUPÉRIEURE DE L'ENSEIGNEMENT TECHNIQUE - MOHAMMEDIA

MODULE : CONCEPTION ET PROGRAMMATION POO

---

## Application de Gestion de Livraison de Colis

---

*Élèves :*  
Yasser NAMEZ

*Enseignant :*  
M. Abdelmajid  
BOUSSELHAM

3 juin 2025

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Objectif du Projet . . . . .	3
1.2	Technologies Utilisées . . . . .	3
<b>2</b>	<b>Architecture du Système</b>	<b>4</b>
2.1	Diagramme de Cas d'Utilisation . . . . .	4
2.2	Diagramme de Classes . . . . .	5
2.3	Modèle Logique de Données (MLD) . . . . .	6
<b>3</b>	<b>Architecture du Code</b>	<b>6</b>
3.1	Package Structure . . . . .	6
3.2	Couche Entités (Entities) . . . . .	6
3.2.1	Classe Colis . . . . .	7
3.2.2	Classe Livreur . . . . .	7
3.3	Couche DAO (Data Access Objects) . . . . .	8
3.3.1	Interface IColisDAO . . . . .	8
3.3.2	Implémentation ColisDAOImpl Complète . . . . .	8
3.4	Couche Service . . . . .	13
3.4.1	Interface IColisService . . . . .	13
3.4.2	Implémentation ColisServiceImpl . . . . .	13
3.5	Couche Contrôleur . . . . .	15
3.5.1	ColisController . . . . .	15
<b>4</b>	<b>Interfaces Utilisateur</b>	<b>17</b>
4.1	Interface Principale . . . . .	17
4.2	Interface de Gestion des Livreurs . . . . .	18
<b>5</b>	<b>Fonctionnalités Principales</b>	<b>18</b>
5.1	Gestion des Colis . . . . .	18
5.1.1	Ajout de Colis . . . . .	18
5.1.2	Assignation aux Livreurs . . . . .	19
5.1.3	Marquage comme Livré . . . . .	19
5.2	Gestion des Livreurs . . . . .	20
5.2.1	CRUD des Livreurs . . . . .	20
5.3	Export CSV . . . . .	21
<b>6</b>	<b>Gestion des Erreurs et Validation</b>	<b>22</b>
6.1	Validation Côté Client . . . . .	22
6.2	Gestion des Exceptions . . . . .	22
<b>7</b>	<b>Configuration et Déploiement</b>	<b>23</b>
7.1	Configuration Maven . . . . .	23
7.2	Structure des Ressources . . . . .	23

<b>8</b>	<b>Code Complet des Classes Principales</b>	<b>23</b>
8.1	Entités Métier . . . . .	23
8.1.1	Classe Colis Complète . . . . .	23
8.1.2	Classe Livreur Complète . . . . .	25
8.2	Interfaces DAO . . . . .	27
8.2.1	Interface IColisDAO Complète . . . . .	27
8.2.2	Interface ILivreurDAO Complète . . . . .	28
8.3	Implémentations DAO Complètes . . . . .	29
8.3.1	Classe LivreurDAOImpl Complète . . . . .	29
8.4	Services Complètes . . . . .	32
8.4.1	Interface ILivreurService Complète . . . . .	32
8.4.2	Classe LivreurServiceImpl Complète . . . . .	33
8.5	Classe Utilitaire DatabaseManager . . . . .	35
8.6	Classe Utilitaire CSVExporter . . . . .	37
8.7	Classe MainApp . . . . .	39
<b>9</b>	<b>Tests</b>	<b>40</b>
<b>10</b>	<b>Conclusion</b>	<b>41</b>
10.1	Résumé des Réalisations . . . . .	41
10.2	Points Forts . . . . .	42
10.3	Améliorations Possibles . . . . .	42

# 1 Introduction

Ce rapport présente une application de gestion de livraison de colis développée en Java avec JavaFX. L'application permet de gérer les colis, les livreurs et les opérations de livraison dans un système centralisé.

## 1.1 Objectif du Projet

L'objectif principal est de créer une application desktop permettant de :

- Gérer les colis (ajout, modification, suppression)
- Gérer les livreurs (ajout, modification, suppression)
- Assigner des colis aux livreurs
- Suivre le statut des livraisons
- Exporter les données de livraison au format CSV

## 1.2 Technologies Utilisées

- **Java** : Langage de programmation principal
- **JavaFX** : Framework pour l'interface utilisateur
- **FXML** : Format XML pour la définition des interfaces
- **JDBC** : Accès aux données
- **CSS** : Stylisation de l'interface
- **Maven** : Gestionnaire de dépendances

## 2 Architecture du Système

### 2.1 Diagramme de Cas d'Utilisation

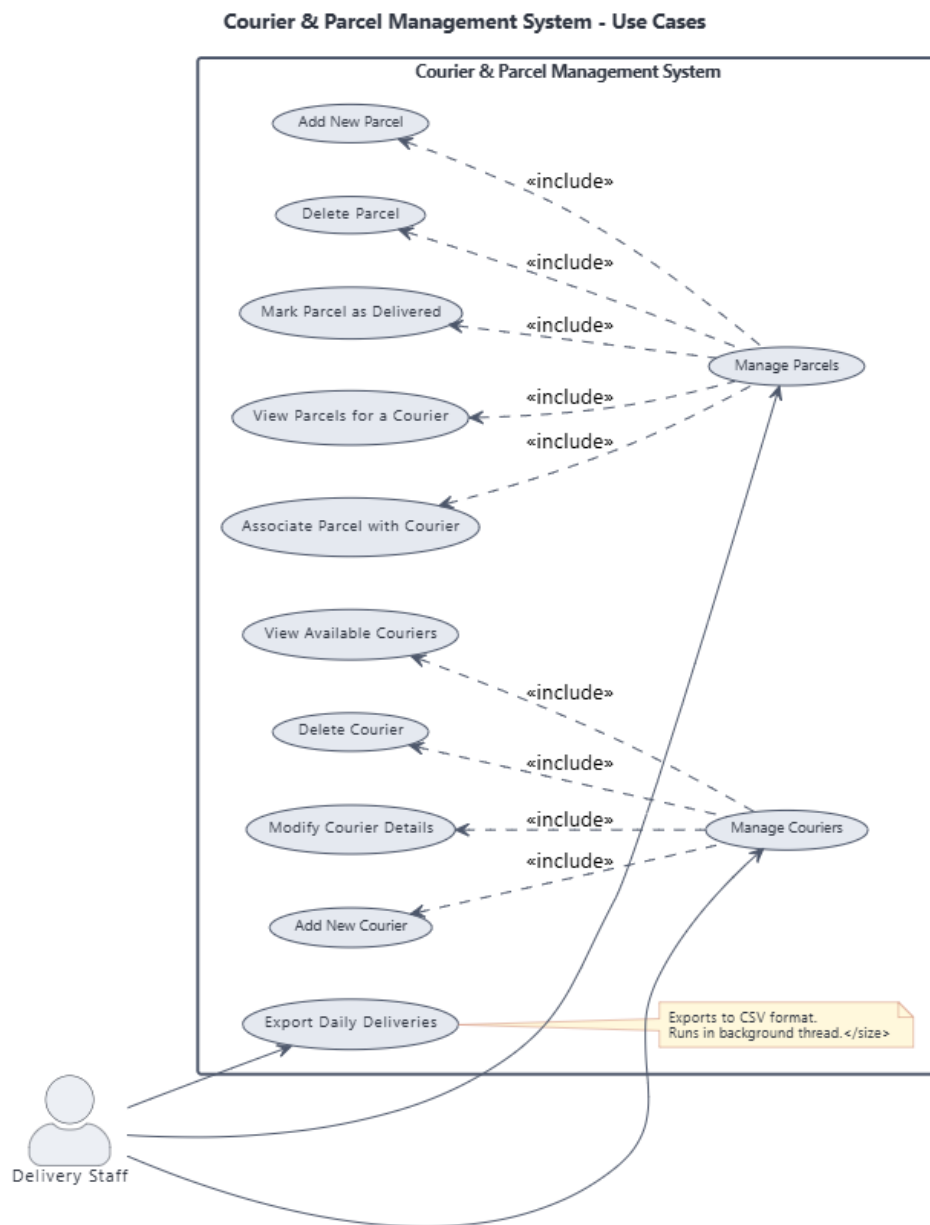


FIGURE 1 – Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation illustre les principales fonctionnalités de l'application et les interactions entre les utilisateurs et le système.

## 2.2 Diagramme de Classes

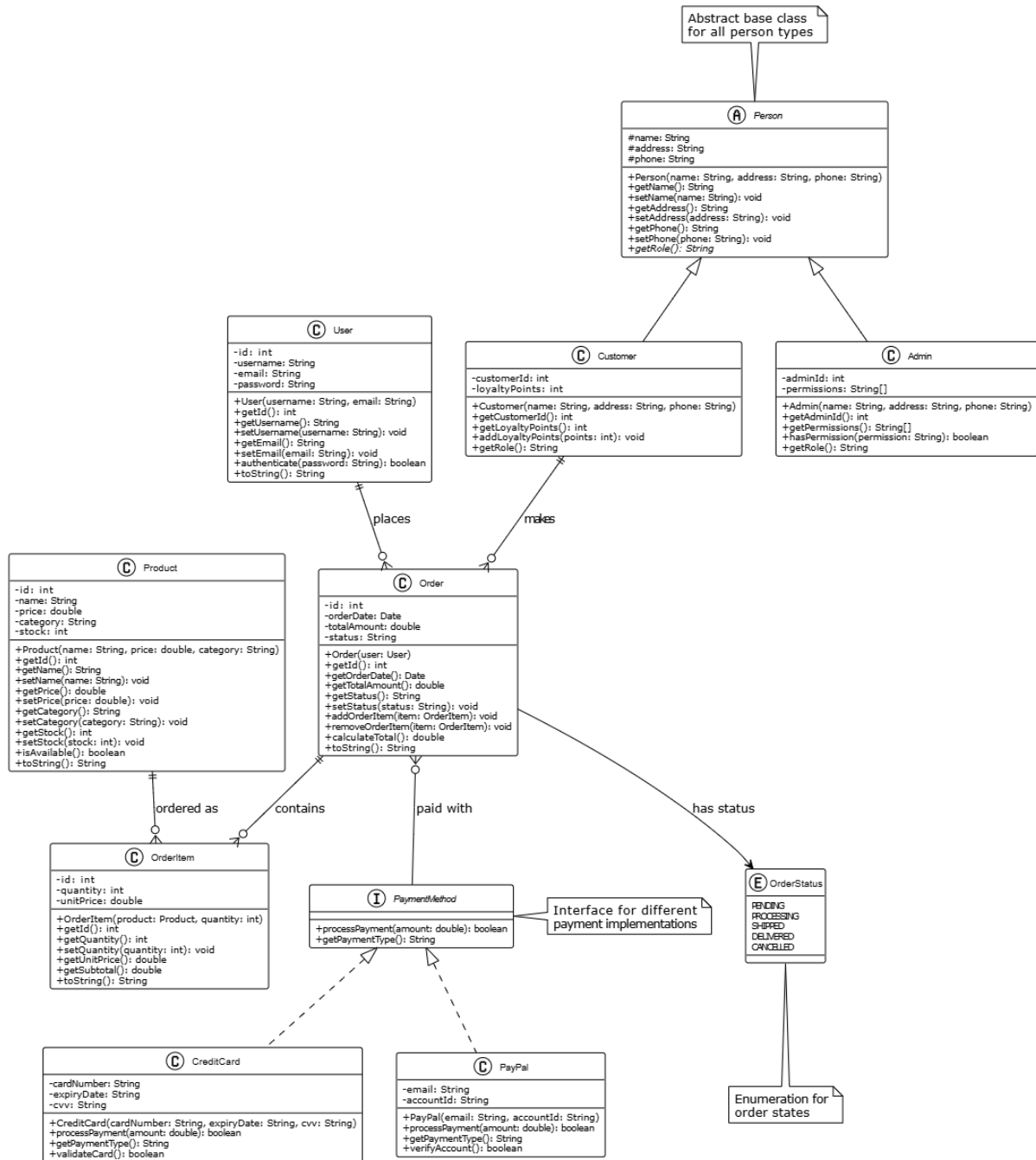


FIGURE 2 – Diagramme de classes

Le diagramme de classes montre la structure orientée objet de l'application avec les principales entités, leurs attributs et relations.

## 2.3 Modèle Logique de Données (MLD)

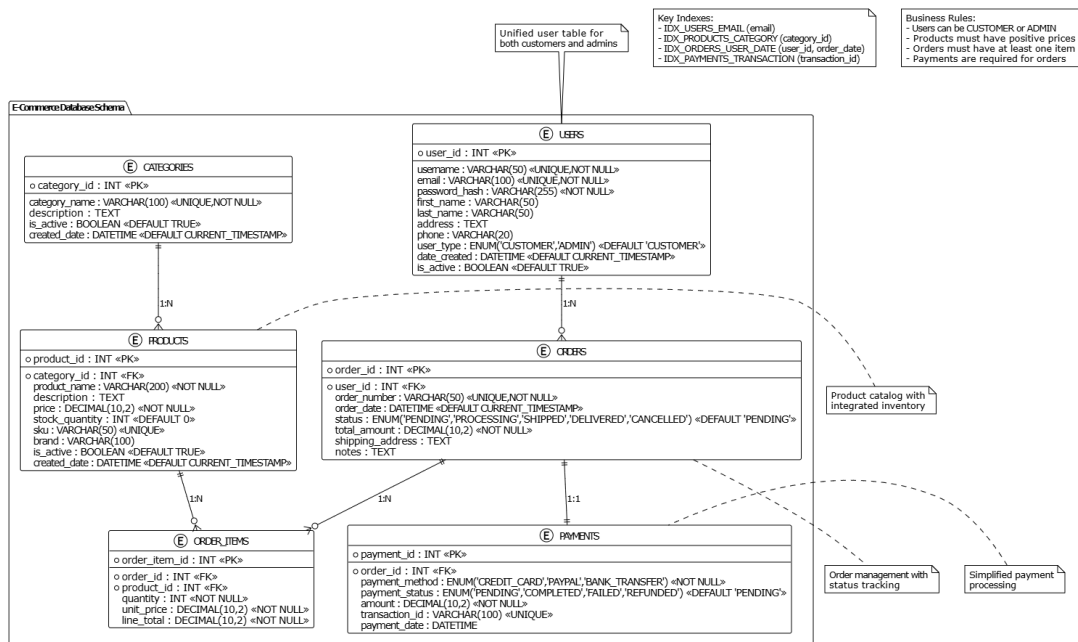


FIGURE 3 – Modèle Logique de Données

Le MLD présente la structure de la base de données avec les tables et leurs relations.

## 3 Architecture du Code

L'application suit une architecture en couches (layered architecture) organisée comme suit :

### 3.1 Package Structure

```

1  com.example.deliveryapp/
2      app/                // Point d'entr e de l'application
3      controller/         // Contr leurs JavaFX (MVC)
4      dao/                // Data Access Objects
5      entities/           // Entit s m tier
6      service/            // Logique m tier
7      util/               // Utilitaires
    
```

Listing 1 – Structure des packages

### 3.2 Couche Entités (Entities)

Les entités représentent les objets métier principaux :

### 3.2.1 Classe Colis

```
1 public class Colis {
2     private int id;
3     private String destinataire;
4     private String adresse;
5     private LocalDate dateEnvoi;
6     private boolean livre;
7     private LocalDate dateLivraison;
8     private Integer livreurId;
9     private Livreur livreur;
10
11     // Constructeurs
12     public Colis() {}
13
14     public Colis(String destinataire, String adresse, LocalDate
15         dateEnvoi) {
16         this.destinataire = destinataire;
17         this.adresse = adresse;
18         this.dateEnvoi = dateEnvoi;
19         this.livre = false;
20     }
21
22     // Getters et Setters...
23
24     public String getStatutText() {
25         return livre ? "Livré" : "Non livré";
26     }
27
28     public String getLivreurName() {
29         return livreur != null ? livreur.getFullName() : "";
30     }
31 }
```

Listing 2 – Entité Colis

### 3.2.2 Classe Livreur

```
1 public class Livreur {
2     private int id;
3     private String nom;
4     private String prenom;
5     private String telephone;
6
7     // Constructeurs
8     public Livreur() {}
9
10    public Livreur(String nom, String prenom, String telephone) {
11        this.nom = nom;
12        this.prenom = prenom;
13    }
14 }
```



```
13         this.telephone = telephone;
14     }
15
16     // Getters et Setters...
17
18     @Override
19     public String toString() {
20         return prenom + " " + nom;
21     }
22
23     public String getFullName() {
24         return prenom + " " + nom;
25     }
26 }
```

Listing 3 – Entité Livreur

### 3.3 Couche DAO (Data Access Objects)

La couche DAO gère l'accès aux données.

#### 3.3.1 Interface IColisDAO

```
1 public interface IColisDAO {
2     void addColis(Colis colis) throws Exception;
3     void updateColis(Colis colis) throws Exception;
4     void deleteColis(int id) throws Exception;
5     Colis getColisById(int id) throws Exception;
6     List<Colis> getAllColis() throws Exception;
7     List<Colis> getColisByLivreurId(int livreurId) throws Exception
8     ;
9     List<Colis> getColisDeliveredOnDate(LocalDate date) throws
10     Exception;
11 }
```

Listing 4 – Interface IColisDAO

#### 3.3.2 Implémentation ColisDAOImpl Complète

```
1 package com.example.deliveryapp.dao;
2
3 import com.example.deliveryapp.entities.Colis;
4 import com.example.deliveryapp.entities.Livreur;
5 import com.example.deliveryapp.util.DatabaseManager;
6
7 import java.sql.*;
8 import java.time.LocalDate;
9 import java.util.ArrayList;
10 import java.util.List;
```

```

11
12 /**
13  * Implementation of IColisDAO using JDBC
14  */
15 public class ColisDAOImpl implements IColisDAO {
16
17     @Override
18     public void addColis(Colis colis) throws Exception {
19         String sql = "INSERT INTO colis (destinataire, adresse,
20             dateEnvoi, livre, dateLivraison, livreur_id) VALUES (?,
21             ?, ?, ?, ?, ?)";
22
23         try (Connection conn = DatabaseManager.getConnection();
24             PreparedStatement stmt = conn.prepareStatement(sql,
25                 Statement.RETURN_GENERATED_KEYS)) {
26
27             stmt.setString(1, colis.getDestinataire());
28             stmt.setString(2, colis.getAdresse());
29             stmt.setDate(3, Date.valueOf(colis.getDateEnvoi()));
30             stmt.setBoolean(4, colis.isLivre());
31             stmt.setDate(5, colis.getDateLivraison() != null ? Date
32                 .valueOf(colis.getDateLivraison()) : null);
33             stmt.setObject(6, colis.getLivreurId());
34
35             int affectedRows = stmt.executeUpdate();
36             if (affectedRows == 0) {
37                 throw new SQLException("Creating colis failed, no
38                     rows affected.");
39             }
40
41             try (ResultSet generatedKeys = stmt.getGeneratedKeys())
42             {
43                 if (generatedKeys.next()) {
44                     colis.setId(generatedKeys.getInt(1));
45                 } else {
46                     throw new SQLException("Creating colis failed,
47                         no ID obtained.");
48                 }
49             }
50         }
51
52     @Override
53     public void updateColis(Colis colis) throws Exception {
54         String sql = "UPDATE colis SET destinataire = ?, adresse =
55             ?, dateEnvoi = ?, livre = ?, dateLivraison = ?,
56             livreur_id = ? WHERE id = ?";
57
58         try (Connection conn = DatabaseManager.getConnection();
59             PreparedStatement stmt = conn.prepareStatement(sql)) {
60
61         }
62     }
63 }

```

```

52         stmt.setString(1, colis.getDestinataire());
53         stmt.setString(2, colis.getAdresse());
54         stmt.setDate(3, Date.valueOf(colis.getDateEnvoi()));
55         stmt.setBoolean(4, colis.isLivre());
56         stmt.setDate(5, colis.getDateLivraison() != null ? Date
57             .valueOf(colis.getDateLivraison()) : null);
58         stmt.setObject(6, colis.getLivreurId());
59         stmt.setInt(7, colis.getId());
60
61         int affectedRows = stmt.executeUpdate();
62         if (affectedRows == 0) {
63             throw new SQLException("Updating colis failed, no
64                 rows affected.");
65         }
66     }
67
68     @Override
69     public void deleteColis(int id) throws Exception {
70         String sql = "DELETE FROM colis WHERE id = ?";
71
72         try (Connection conn = DatabaseManager.getConnection();
73             PreparedStatement stmt = conn.prepareStatement(sql)) {
74
75             stmt.setInt(1, id);
76
77             int affectedRows = stmt.executeUpdate();
78             if (affectedRows == 0) {
79                 throw new SQLException("Deleting colis failed, no
80                     rows affected.");
81             }
82         }
83
84         @Override
85         public Colis getColisById(int id) throws Exception {
86             String sql = "SELECT c.*, l.nom as livreur_nom, l.prenom as
87                 livreur_prenom, l.telephone as livreur_telephone " +
88                 "FROM colis c LEFT JOIN livreurs l ON c.
89                 livreur_id = l.id WHERE c.id = ?";
89
90             try (Connection conn = DatabaseManager.getConnection();
91                 PreparedStatement stmt = conn.prepareStatement(sql)) {
92
93                 stmt.setInt(1, id);
94
95                 try (ResultSet rs = stmt.executeQuery()) {
96                     if (rs.next()) {
97                         return mapResultSetToColis(rs);
98                     }
99                 }
100             }
101         }
    
```

```

97         }
98     }
99 }
100     return null;
101 }
102
103 @Override
104 public List<Colis> getAllColis() throws Exception {
105     String sql = "SELECT c.*, l.nom as livreur_nom, l.prenom as
106         livreur_prenom, l.telephone as livreur_telephone " +
107         "FROM colis c LEFT JOIN livreurs l ON c.
108         livreur_id = l.id ORDER BY c.dateEnvoi DESC"
109     ;
110     List<Colis> colisList = new ArrayList<>();
111
112     try (Connection conn = DatabaseManager.getConnection();
113         PreparedStatement stmt = conn.prepareStatement(sql);
114         ResultSet rs = stmt.executeQuery()) {
115
116         while (rs.next()) {
117             colisList.add(mapResultSetToColis(rs));
118         }
119     }
120
121     return colisList;
122 }
123
124 @Override
125 public List<Colis> getColisByLivreurId(int livreurId) throws
126 Exception {
127     String sql = "SELECT c.*, l.nom as livreur_nom, l.prenom as
128         livreur_prenom, l.telephone as livreur_telephone " +
129         "FROM colis c LEFT JOIN livreurs l ON c.
130         livreur_id = l.id WHERE c.livreur_id = ?
131         ORDER BY c.dateEnvoi DESC";
132     List<Colis> colisList = new ArrayList<>();
133
134     try (Connection conn = DatabaseManager.getConnection();
135         PreparedStatement stmt = conn.prepareStatement(sql)) {
136
137         stmt.setInt(1, livreurId);
138
139         try (ResultSet rs = stmt.executeQuery()) {
140             while (rs.next()) {
141                 colisList.add(mapResultSetToColis(rs));
142             }
143         }
144     }
145
146     return colisList;
147 }
    
```

```

140     }
141
142     @Override
143     public List<Colis> getColisDeliveredOnDate(LocalDate date)
144         throws Exception {
145         String sql = "SELECT c.*, l.nom as livreur_nom, l.prenom as
146             livreur_prenom, l.telephone as livreur_telephone " +
147             "FROM colis c LEFT JOIN livreurs l ON c.
148                 livreur_id = l.id WHERE c.dateLivraison = ?
149                 ORDER BY c.id";
150
151         List<Colis> colisList = new ArrayList<>();
152
153         try (Connection conn = DriverManager.getConnection();
154             PreparedStatement stmt = conn.prepareStatement(sql)) {
155
156             stmt.setDate(1, Date.valueOf(date));
157
158             try (ResultSet rs = stmt.executeQuery()) {
159                 while (rs.next()) {
160                     colisList.add(mapResultSetToColis(rs));
161                 }
162             }
163         }
164
165         return colisList;
166     }
167
168     /**
169     * Helper method to map ResultSet to Colis object
170     */
171     private Colis mapResultSetToColis(ResultSet rs) throws
172         SQLException {
173         Colis colis = new Colis();
174         colis.setId(rs.getInt("id"));
175         colis.setDestinataire(rs.getString("destinataire"));
176         colis.setAdresse(rs.getString("adresse"));
177
178         Date dateEnvoi = rs.getDate("dateEnvoi");
179         if (dateEnvoi != null) {
180             colis.setDateEnvoi(dateEnvoi.toLocalDate());
181         }
182
183         colis.setLivre(rs.getBoolean("livre"));
184
185         Date dateLivraison = rs.getDate("dateLivraison");
186         if (dateLivraison != null) {
187             colis.setDateLivraison(dateLivraison.toLocalDate());
188         }
189
190         Object livreurIdObj = rs.getObject("livreur_id");
    
```

```

185         if (livreurIdObj != null) {
186             colis.setLivreurId((Integer) livreurIdObj);
187
188             // Create Livreur object if data is available
189             String livreurNom = rs.getString("livreur_nom");
190             if (livreurNom != null) {
191                 Livreur livreur = new Livreur(
192                     colis.getLivreurId(),
193                     livreurNom,
194                     rs.getString("livreur_prenom"),
195                     rs.getString("livreur_telephone")
196                 );
197                 colis.setLivreur(livreur);
198             }
199         }
200
201         return colis;
202     }
203 }
    
```

Listing 5 – Implémentation ColisDAOImpl complète

## 3.4 Couche Service

La couche service contient la logique métier.

### 3.4.1 Interface IColisService

```

1 public interface IColisService {
2     void addColis(Colis colis) throws Exception;
3     void updateColis(Colis colis) throws Exception;
4     void deleteColis(int id) throws Exception;
5     Colis getColisById(int id);
6     List<Colis> getAllColis();
7     void assignColisToLivreur(int colisId, int livreurId) throws
8         Exception;
9     List<Colis> getColisByLivreur(int livreurId);
10    void markColisAsDelivered(int colisId) throws Exception;
11    List<Colis> getColisDeliveredToday();
12    String exportDeliveredTodayToCSV() throws Exception;
13 }
    
```

Listing 6 – Interface IColisService

### 3.4.2 Implémentation ColisServiceImpl

```

1 public class ColisServiceImpl implements IColisService {
2     private final IColisDAO colisDAO;
3     private final CSVExporter csvExporter;
    
```

```

4
5     public ColisServiceImpl() {
6         this.colisDAO = new ColisDAOImpl();
7         this.csvExporter = new CSVExporter();
8     }
9
10    @Override
11    public void addColis(Colis colis) throws Exception {
12        validateColis(colis);
13        colisDAO.addColis(colis);
14    }
15
16    @Override
17    public void markColisAsDelivered(int colisId) throws Exception
18    {
19        if (colisId <= 0) {
20            throw new Exception("ID du colis invalide");
21        }
22
23        Colis colis = colisDAO.getColisById(colisId);
24        if (colis == null) {
25            throw new Exception("Colis non trouv ");
26        }
27
28        if (colis.isLivre()) {
29            throw new Exception("Ce colis est d j   marqu   comme
30                livr ");
31        }
32
33        colis.setLivre(true);
34        colis.setDateLivraison(LocalDate.now());
35        colisDAO.updateColis(colis);
36    }
37
38    private void validateColis(Colis colis) throws Exception {
39        if (colis == null) {
40            throw new Exception("Les donn es du colis ne peuvent
41                pas tre nulles");
42        }
43
44        if (colis.getDestinataire() == null || colis.
45            getDestinataire().trim().isEmpty()) {
46            throw new Exception("Le destinataire du colis est
47                obligatoire");
48        }
49
50        // Autres validations...
51    }
52 }

```

Listing 7 – Service Colis avec validation

## 3.5 Couche Contrôleur

Les contrôleurs gèrent l'interaction entre l'interface utilisateur et la logique métier.

### 3.5.1 ColisController

```

1  public class ColisController implements Initializable {
2      @FXML private TextField destinataireField;
3      @FXML private TextArea adresseField;
4      @FXML private DatePicker dateEnvoiPicker;
5      @FXML private ComboBox<Livreur> livreurComboBox;
6      @FXML private TableView<Colis> colisTable;
7
8      private IColisService colisService;
9      private ILivreurService livreurService;
10     private ObservableList<Colis> colisList;
11     private Colis selectedColis;
12
13     @Override
14     public void initialize(URL url, ResourceBundle resourceBundle)
15     {
16         colisService = new ColisServiceImpl();
17         livreurService = new LivreurServiceImpl();
18
19         setupTable();
20         setupForm();
21         loadData();
22         setupEventHandlers();
23     }
24
25     @FXML
26     private void handleAddColis() {
27         if (!validateForm()) return;
28
29         try {
30             Colis newColis = createColisFromForm();
31             colisService.addColis(newColis);
32
33             AlertUtils.showSuccess("Succ s", "Colis ajout avec succ s.");
34             loadColis();
35             clearForm();
36
37         } catch (Exception e) {
38             AlertUtils.showError("Erreur d'Ajout",
39                 "Impossible d'ajouter le colis", e.getMessage());
40         }
41     }
42
43     @FXML
    
```

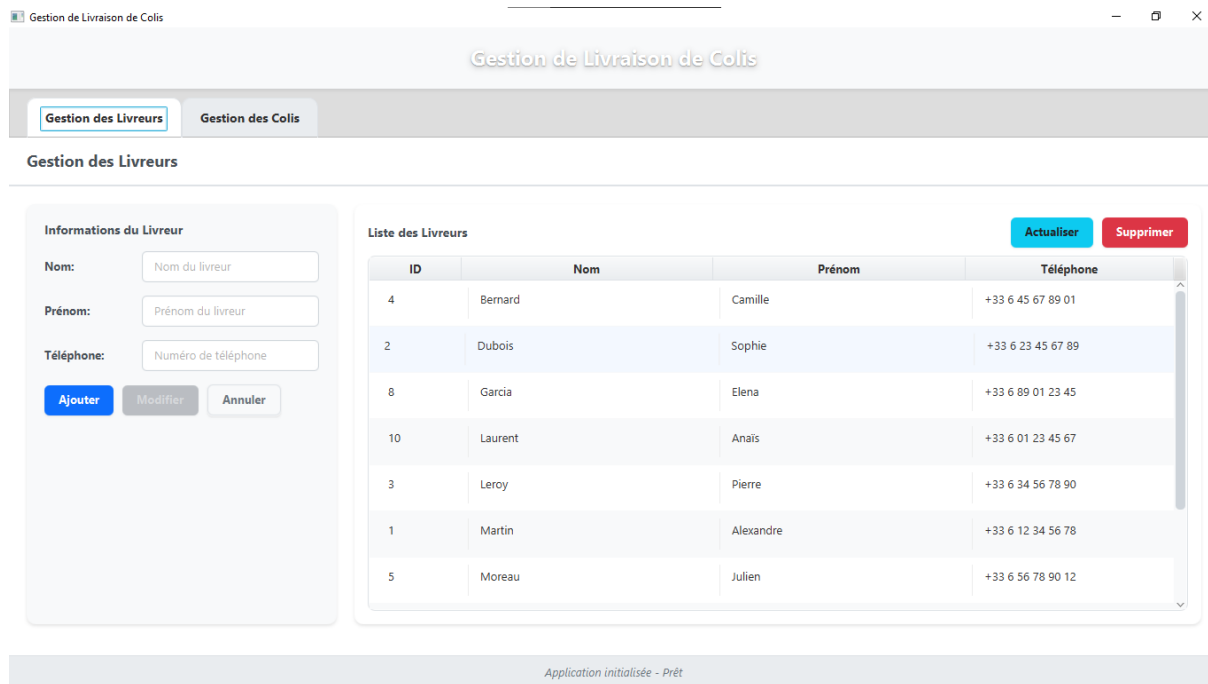


```
43     private void handleAssignToDriver() {  
44         if (selectedColis == null || selectedColis.isLivre())  
45             return;  
46  
47         Livreur selectedLivreur = livreurComboBox.getValue();  
48         if (selectedLivreur == null) {  
49             AlertUtils.showWarning("S lection", "Aucun livreur  
50                 s lectionn ",  
51                 "Veuillez s lectionner un livreur.");  
52             return;  
53         }  
54  
55         try {  
56             colisService.assignColisToLivreur(selectedColis.getId()  
57                 ,  
58                 selectedLivreur.getId());  
59             AlertUtils.showSuccess("Succ s",  
60                 "Colis assign avec succ s au livreur " +  
61                 selectedLivreur.getFullName() + ".");  
62             loadColis();  
63         } catch (Exception e) {  
64             AlertUtils.showError("Erreur d'Assignment",  
65                 "Impossible d'assigner le colis", e.getMessage());  
66         }  
67     }  
68 }
```

Listing 8 – Contrôleur Colis (extrait)

## 4 Interfaces Utilisateur

### 4.1 Interface Principale



**Gestion de Livraison de Colis**

**Gestion des Livreurs**

**Informations du Livreur**

Nom:

Prénom:

Téléphone:

**Ajouter** **Modifier** **Annuler**

**Liste des Livreurs** **Actualiser** **Supprimer**

ID	Nom	Prénom	Téléphone
4	Bernard	Camille	+33 6 45 67 89 01
2	Dubois	Sophie	+33 6 23 45 67 89
8	Garcia	Elena	+33 6 89 01 23 45
10	Laurent	Anais	+33 6 01 23 45 67
3	Leroy	Pierre	+33 6 34 56 78 90
1	Martin	Alexandre	+33 6 12 34 56 78
5	Moreau	Julien	+33 6 56 78 90 12

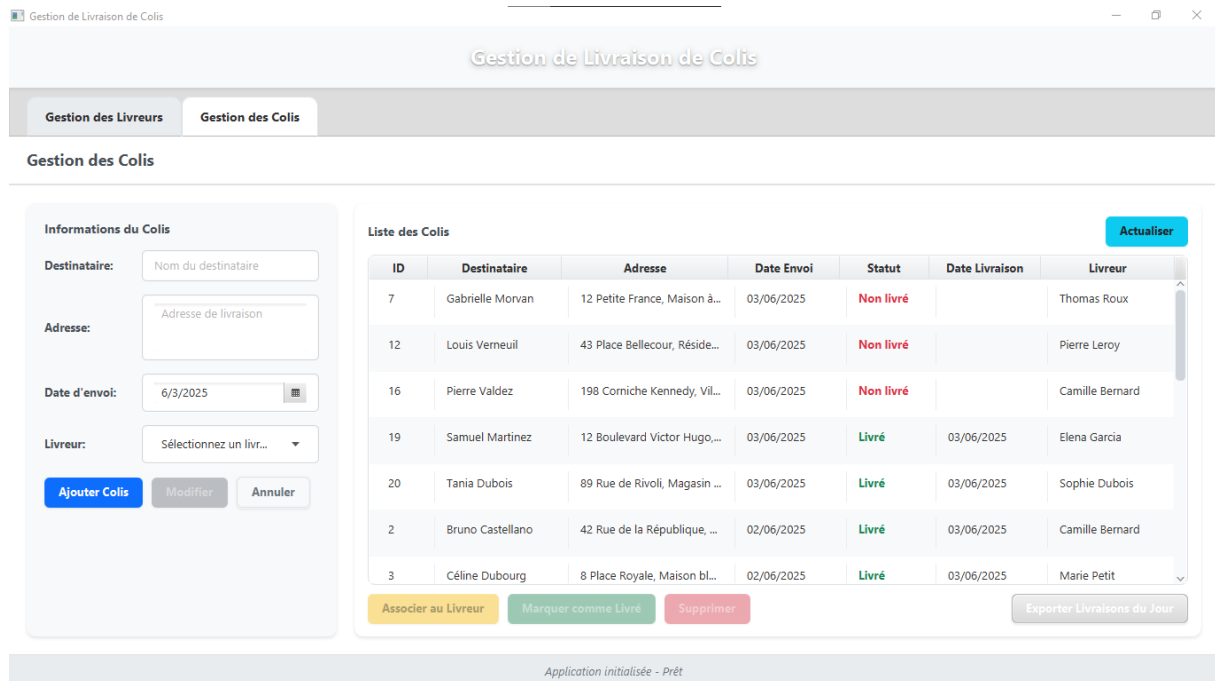
Application initialisée - Prêt

FIGURE 4 – Interface principale - Gestion des colis

L'interface principale permet de :

- Visualiser la liste de tous les colis
- Ajouter, modifier et supprimer des colis
- Assigner des colis aux livreurs
- Marquer les colis comme livrés
- Exporter les livraisons du jour en CSV

## 4.2 Interface de Gestion des Livreurs



**Gestion de Livraison de Colis**

**Gestion des Colis**

**Informations du Colis**

Destinataire:

Adresse:

Date d'envoi:

Livreur:

**Ajouter Colis** **Modifier** **Annuler**

**Liste des Colis**

ID	Destinataire	Adresse	Date Envoi	Statut	Date Livraison	Livreur
7	Gabrielle Morvan	12 Petite France, Maison à...	03/06/2025	Non livré		Thomas Roux
12	Louis Verneuil	43 Place Bellecour, Réside...	03/06/2025	Non livré		Pierre Leroy
16	Pierre Valdez	198 Corniche Kennedy, Vil...	03/06/2025	Non livré		Camille Bernard
19	Samuel Martinez	12 Boulevard Victor Hugo, ...	03/06/2025	Livré	03/06/2025	Elena Garcia
20	Tania Dubois	89 Rue de Rivoli, Magasin ...	03/06/2025	Livré	03/06/2025	Sophie Dubois
2	Bruno Castellano	42 Rue de la République, ...	02/06/2025	Livré	03/06/2025	Camille Bernard
3	Céline Dubourg	8 Place Royale, Maison bl...	02/06/2025	Livré	03/06/2025	Marie Petit

**Associer au Livreur** **Marquer comme Livré** **Supprimer** **Exporter Livraisons du Jour**

Application initialisée - Prêt

FIGURE 5 – Interface de gestion des livreurs

Cette interface permet de :

- Visualiser la liste des livreurs
- Ajouter de nouveaux livreurs
- Modifier les informations des livreurs existants
- Supprimer des livreurs

## 5 Fonctionnalités Principales

### 5.1 Gestion des Colis

#### 5.1.1 Ajout de Colis

L'application permet d'ajouter de nouveaux colis avec validation des données :

```

1 private boolean validateForm() {
2     if (destinataireField.getText().trim().isEmpty()) {
3         AlertUtils.showWarning("Validation", "Champ requis",
4             "Le destinataire est obligatoire.");
5         destinataireField.requestFocus();
6         return false;
7     }
8
9     if (adresseField.getText().trim().isEmpty()) {
10        AlertUtils.showWarning("Validation", "Champ requis",
11            "L'adresse est obligatoire.");
  
```

```

12     adresseField.requestFocus();
13     return false;
14 }
15
16     if (dateEnvoiPicker.getValue() == null) {
17         AlertUtils.showWarning("Validation", "Champ requis",
18             "La date d'envoi est obligatoire.");
19         dateEnvoiPicker.requestFocus();
20         return false;
21     }
22
23     return true;
24 }
    
```

Listing 9 – Validation des colis

### 5.1.2 Assignation aux Livreurs

Les colis peuvent être assignés aux livreurs disponibles :

```

1  @Override
2  public void assignColisToLivreur(int colisId, int livreurId) throws
    Exception {
3      if (colisId <= 0) {
4          throw new Exception("ID du colis invalide");
5      }
6      if (livreurId <= 0) {
7          throw new Exception("ID du livreur invalide");
8      }
9
10     Colis colis = colisDAO.getColisById(colisId);
11     if (colis == null) {
12         throw new Exception("Colis non trouv ");
13     }
14
15     colis.setLivreurId(livreurId);
16     colisDAO.updateColis(colis);
17 }
    
```

Listing 10 – Assignation de colis

### 5.1.3 Marquage comme Livré

Les colis peuvent être marqués comme livrés avec mise à jour automatique de la date :

```

1  @Override
2  public void markColisAsDelivered(int colisId) throws Exception {
3      Colis colis = colisDAO.getColisById(colisId);
4      if (colis == null) {
5          throw new Exception("Colis non trouv ");
6      }
    
```

```

7
8     if (colis.isLivre()) {
9         throw new Exception("Ce colis est d j  marqu  comme
10             livr ");
11     }
12     colis.setLivre(true);
13     colis.setDateLivraison(LocalDate.now());
14     colisDAO.updateColis(colis);
15 }
    
```

Listing 11 – Marquage comme livré

## 5.2 Gestion des Livreurs

### 5.2.1 CRUD des Livreurs

L'application fournit toutes les opérations CRUD pour les livreurs :

```

1 public class LivreurServiceImpl implements ILivreurService {
2     private final ILivreurDAO livreurDAO;
3
4     @Override
5     public void addLivreur(Livreur livreur) throws Exception {
6         validateLivreur(livreur);
7         livreurDAO.addLivreur(livreur);
8     }
9
10    @Override
11    public void updateLivreur(Livreur livreur) throws Exception {
12        validateLivreur(livreur);
13        if (livreur.getId() <= 0) {
14            throw new Exception("ID du livreur invalide pour la
15                mise jour");
16        }
17        livreurDAO.updateLivreur(livreur);
18    }
19
20    private void validateLivreur(Livreur livreur) throws Exception
21    {
22        if (livreur == null) {
23            throw new Exception("Les donn es du livreur ne peuvent
24                pas tre nulles");
25        }
26
27        if (livreur.getNom() == null || livreur.getNom().trim().
28            isEmpty()) {
29            throw new Exception("Le nom du livreur est obligatoire"
30                );
31        }
32    }
33 }
    
```

```

28         if (livreur.getPrenom() == null || livreur.getPrenom().trim
29             ().isEmpty()) {
30             throw new Exception("Le pr nom du livreur est
31                 obligatoire");
32         }
33     }
34 }
    
```

Listing 12 – Service Livreur

### 5.3 Export CSV

L'application permet d'exporter les livraisons du jour au format CSV :

```

1  @FXML
2  private void handleExportCSV() {
3      if (!AlertUtils.showConfirmation("Exporter les livraisons",
4          "Exporter les livraisons du jour",
5          "Voulez-vous exporter les livraisons d'aujourd'hui au
6              format CSV ?")) {
7          return;
8      }
9
10     Task<String> exportTask = new Task<String>() {
11         @Override
12         protected String call() throws Exception {
13             var deliveredToday = colisService.
14                 getColisDeliveredToday();
15
16             if (deliveredToday.isEmpty()) {
17                 throw new Exception("Aucune livraison trouv e pour
18                     aujourd'hui.");
19             }
20
21             String csvContent = colisService.
22                 exportDeliveredTodayToCSV();
23             String filename = "livraisons_" +
24                 LocalDate.now().format(DateTimeFormatter.ofPattern(
25                     "yyyy-MM-dd")) + ".csv";
26
27             try (FileWriter writer = new FileWriter(filename)) {
28                 writer.write(csvContent);
29             }
30
31             return filename;
32         }
33     };
34
35     // Gestion des vnements de succ s et d' chec ...
36 }
    
```

Listing 13 – Export CSV

## 6 Gestion des Erreurs et Validation

### 6.1 Validation Côté Client

L'application implémente une validation robuste côté client avec des messages d'erreur explicites.

### 6.2 Gestion des Exceptions

Toutes les opérations critiques sont protégées par une gestion d'exceptions appropriée :

```

1 public class AlertUtils {
2     public static void showError(String title, String header,
3         String content) {
4         Alert alert = new Alert(Alert.AlertType.ERROR);
5         alert.setTitle(title);
6         alert.setHeaderText(header);
7         alert.setContentText(content);
8         alert.showAndWait();
9     }
10
11     public static void showSuccess(String title, String content) {
12         Alert alert = new Alert(Alert.AlertType.INFORMATION);
13         alert.setTitle(title);
14         alert.setHeaderText("Op ration r ussie");
15         alert.setContentText(content);
16         alert.showAndWait();
17     }
18
19     public static boolean showDeleteConfirmation(String itemName) {
20         Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
21         alert.setTitle("Confirmation de suppression");
22         alert.setHeaderText(" tes -vous s r ?");
23         alert.setContentText("Voulez-vous vraiment supprimer " +
24             itemName + " ?");
25
26         Optional<ButtonType> result = alert.showAndWait();
27         return result.isPresent() && result.get() == ButtonType.OK;
28     }
29 }

```

Listing 14 – Utilitaire d'alertes

## 7 Configuration et Déploiement

### 7.1 Configuration Maven

Le projet utilise Maven pour la gestion des dépendances. Le fichier `pom.xml` inclut les dépendances nécessaires pour JavaFX et JDBC.

### 7.2 Structure des Ressources

- **FXML** : Fichiers de définition d'interface dans `src/main/resources/fxml/`
- **CSS** : Feuilles de style dans `src/main/resources/css/`
- **Configuration** : Fichier de configuration de base de données

## 8 Code Complet des Classes Principales

Cette section présente le code complet de toutes les classes principales de l'application.

### 8.1 Entités Métier

#### 8.1.1 Classe Colis Complète

```
1 package com.example.deliveryapp.entities;
2
3 import java.time.LocalDate;
4
5 /**
6  * Entity class representing a package (Colis)
7  */
8 public class Colis {
9     private int id;
10    private String destinataire;
11    private String adresse;
12    private LocalDate dateEnvoi;
13    private boolean livre;
14    private LocalDate dateLivraison;
15    private Integer livreurId;
16    private Livreur livreur; // For convenience, populated by
17                               service layer
18
19    // Default constructor
20    public Colis() {}
21
22    // Constructor without ID (for new packages)
23    public Colis(String destinataire, String adresse, LocalDate
24    dateEnvoi) {
25        this.destinataire = destinataire;
26        this.adresse = adresse;
27        this.dateEnvoi = dateEnvoi;
28        this.livre = false;
```



```
27     }
28
29     // Full constructor
30     public Colis(int id, String destinataire, String adresse,
31                 LocalDate dateEnvoi,
32                 boolean livre, LocalDate dateLivraison, Integer
33                 livreurId) {
34         this.id = id;
35         this.destinataire = destinataire;
36         this.adresse = adresse;
37         this.dateEnvoi = dateEnvoi;
38         this.livre = livre;
39         this.dateLivraison = dateLivraison;
40         this.livreurId = livreurId;
41     }
42
43     // Getters and Setters
44     public int getId() {
45         return id;
46     }
47
48     public void setId(int id) {
49         this.id = id;
50     }
51
52     public String getDestinataire() {
53         return destinataire;
54     }
55
56     public void setDestinataire(String destinataire) {
57         this.destinataire = destinataire;
58     }
59
60     public String getAdresse() {
61         return adresse;
62     }
63
64     public void setAdresse(String adresse) {
65         this.adresse = adresse;
66     }
67
68     public LocalDate getDateEnvoi() {
69         return dateEnvoi;
70     }
71
72     public void setDateEnvoi(LocalDate dateEnvoi) {
73         this.dateEnvoi = dateEnvoi;
74     }
75
76     public boolean isLivre() {
```

```

75         return livre;
76     }
77
78     public void setLivre(boolean livre) {
79         this.livre = livre;
80     }
81
82     public LocalDate getDateLivraison() {
83         return dateLivraison;
84     }
85
86     public void setDateLivraison(LocalDate dateLivraison) {
87         this.dateLivraison = dateLivraison;
88     }
89
90     public Integer getLivreurId() {
91         return livreurId;
92     }
93
94     public void setLivreurId(Integer livreurId) {
95         this.livreurId = livreurId;
96     }
97
98     public Livreur getLivreur() {
99         return livreur;
100    }
101
102    public void setLivreur(Livreur livreur) {
103        this.livreur = livreur;
104        if (livreur != null) {
105            this.livreurId = livreur.getId();
106        }
107    }
108
109    public String getStatutText() {
110        return livre ? "Livré" : "Non livré";
111    }
112
113    public String getLivreurName() {
114        return livreur != null ? livreur.getFullName() : "";
115    }
116 }
    
```

Listing 15 – Classe Colis complète

### 8.1.2 Classe Livreur Complète

```

1 package com.example.deliveryapp.entities;
2
3 /**
    
```

```

4  * Entity class representing a delivery driver (Livreur)
5  */
6  public class Livreur {
7      private int id;
8      private String nom;
9      private String prenom;
10     private String telephone;
11
12     // Default constructor
13     public Livreur() {}
14
15     // Constructor without ID (for new drivers)
16     public Livreur(String nom, String prenom, String telephone) {
17         this.nom = nom;
18         this.prenom = prenom;
19         this.telephone = telephone;
20     }
21
22     // Constructor with ID (for existing drivers)
23     public Livreur(int id, String nom, String prenom, String
24         telephone) {
25         this.id = id;
26         this.nom = nom;
27         this.prenom = prenom;
28         this.telephone = telephone;
29     }
30
31     // Getters and Setters
32     public int getId() {
33         return id;
34     }
35
36     public void setId(int id) {
37         this.id = id;
38     }
39
40     public String getNom() {
41         return nom;
42     }
43
44     public void setNom(String nom) {
45         this.nom = nom;
46     }
47
48     public String getPrenom() {
49         return prenom;
50     }
51
52     public void setPrenom(String prenom) {
53         this.prenom = prenom;

```

```
53     }
54
55     public String getTelephone() {
56         return telephone;
57     }
58
59     public void setTelephone(String telephone) {
60         this.telephone = telephone;
61     }
62
63     @Override
64     public String toString() {
65         return prenom + " " + nom;
66     }
67
68     public String getFullName() {
69         return prenom + " " + nom;
70     }
71 }
```

Listing 16 – Classe Livreur complète

## 8.2 Interfaces DAO

### 8.2.1 Interface IColisDAO Complète

```
1 package com.example.deliveryapp.dao;
2
3 import com.example.deliveryapp.entities.Colis;
4 import java.time.LocalDate;
5 import java.util.List;
6
7 /**
8  * Data Access Object interface for Colis entities
9  */
10 public interface IColisDAO {
11     /**
12      * Add a new package to the database
13      * @param colis The package to add
14      * @throws Exception if operation fails
15      */
16     void addColis(Colis colis) throws Exception;
17
18     /**
19      * Update an existing package in the database
20      * @param colis The package to update
21      * @throws Exception if operation fails
22      */
23     void updateColis(Colis colis) throws Exception;
24 }
```

```

25     /**
26      * Delete a package from the database
27      * @param id The package ID to delete
28      * @throws Exception if operation fails
29      */
30     void deleteColis(int id) throws Exception;
31
32     /**
33      * Get a package by its ID
34      * @param id The package ID
35      * @return The package or null if not found
36      * @throws Exception if operation fails
37      */
38     Colis getColisById(int id) throws Exception;
39
40     /**
41      * Get all packages from the database
42      * @return List of all packages
43      * @throws Exception if operation fails
44      */
45     List<Colis> getAllColis() throws Exception;
46
47     /**
48      * Get all packages assigned to a specific driver
49      * @param livreurId The driver's ID
50      * @return List of packages assigned to the driver
51      * @throws Exception if operation fails
52      */
53     List<Colis> getColisByLivreurId(int livreurId) throws Exception
54         ;
55
56     /**
57      * Get all packages delivered on a specific date
58      * @param date The delivery date
59      * @return List of packages delivered on the specified date
60      * @throws Exception if operation fails
61      */
62     List<Colis> getColisDeliveredOnDate(LocalDate date) throws
        Exception;
    }
    
```

Listing 17 – Interface IColisDAO complète

### 8.2.2 Interface ILivreurDAO Complète

```

1 package com.example.deliveryapp.dao;
2
3 import com.example.deliveryapp.entities.Livreur;
4 import java.util.List;
5
    
```

```

6  /**
7   * Data Access Object interface for Livreur entities
8   */
9  public interface ILivreurDAO {
10     /**
11      * Add a new driver to the database
12      * @param livreur The driver to add
13      * @throws Exception if operation fails
14      */
15     void addLivreur(Livreur livreur) throws Exception;
16
17     /**
18      * Update an existing driver in the database
19      * @param livreur The driver to update
20      * @throws Exception if operation fails
21      */
22     void updateLivreur(Livreur livreur) throws Exception;
23
24     /**
25      * Delete a driver from the database
26      * @param id The driver ID to delete
27      * @throws Exception if operation fails
28      */
29     void deleteLivreur(int id) throws Exception;
30
31     /**
32      * Get a driver by ID
33      * @param id The driver ID
34      * @return The driver or null if not found
35      * @throws Exception if operation fails
36      */
37     Livreur getLivreurById(int id) throws Exception;
38
39     /**
40      * Get all drivers from the database
41      * @return List of all drivers
42      * @throws Exception if operation fails
43      */
44     List<Livreur> getAllLivresurs() throws Exception;
45 }
    
```

Listing 18 – Interface ILivreurDAO complète

## 8.3 Implémentations DAO Complètes

### 8.3.1 Classe LivreurDAOImpl Complète

```

1  package com.example.deliveryapp.dao;
2
3  import com.example.deliveryapp.entities.Livreur;
    
```

```

4  import com.example.deliveryapp.util.DatabaseManager;
5
6  import java.sql.*;
7  import java.util.ArrayList;
8  import java.util.List;
9
10 /**
11  * Implementation of ILivreurDAO using JDBC
12  */
13 public class LivreurDAOImpl implements ILivreurDAO {
14
15     @Override
16     public void addLivreur(Livreur livreur) throws Exception {
17         String sql = "INSERT INTO livreurs (nom, prenom, telephone)
18             VALUES (?, ?, ?)";
19
20         try (Connection conn = DatabaseManager.getConnection();
21             PreparedStatement stmt = conn.prepareStatement(sql,
22                 Statement.RETURN_GENERATED_KEYS)) {
23
24             stmt.setString(1, livreur.getNom());
25             stmt.setString(2, livreur.getPrenom());
26             stmt.setString(3, livreur.getTelephone());
27
28             int affectedRows = stmt.executeUpdate();
29
30             if (affectedRows == 0) {
31                 throw new SQLException("Creating livreur failed, no
32                     rows affected.");
33             }
34
35             try (ResultSet generatedKeys = stmt.getGeneratedKeys())
36             {
37                 if (generatedKeys.next()) {
38                     livreur.setId(generatedKeys.getInt(1));
39                 } else {
40                     throw new SQLException("Creating livreur failed
41                         , no ID obtained.");
42                 }
43             }
44         }
45
46     @Override
47     public void updateLivreur(Livreur livreur) throws Exception {
48         String sql = "UPDATE livreurs SET nom = ?, prenom = ?,
49             telephone = ? WHERE id = ?";
50
51         try (Connection conn = DatabaseManager.getConnection();
52             PreparedStatement stmt = conn.prepareStatement(sql)) {
53
54             stmt.setString(1, livreur.getNom());
55             stmt.setString(2, livreur.getPrenom());
56             stmt.setString(3, livreur.getTelephone());
57             stmt.setInt(4, livreur.getId());
58
59             int affectedRows = stmt.executeUpdate();
60
61             if (affectedRows == 0) {
62                 throw new SQLException("Updating livreur failed, no
63                     rows affected.");
64             }
65         }
66     }
67 }

```

```

48         stmt.setString(1, livreur.getNom());
49         stmt.setString(2, livreur.getPrenom());
50         stmt.setString(3, livreur.getTelephone());
51         stmt.setInt(4, livreur.getId());
52
53         int affectedRows = stmt.executeUpdate();
54         if (affectedRows == 0) {
55             throw new SQLException("Updating livreur failed, no
56                 rows affected.");
57         }
58     }
59 }
60
61 @Override
62 public void deleteLivreur(int id) throws Exception {
63     String sql = "DELETE FROM livreurs WHERE id = ?";
64
65     try (Connection conn = DatabaseManager.getConnection();
66         PreparedStatement stmt = conn.prepareStatement(sql)) {
67
68         stmt.setInt(1, id);
69
70         int affectedRows = stmt.executeUpdate();
71         if (affectedRows == 0) {
72             throw new SQLException("Deleting livreur failed, no
73                 rows affected.");
74         }
75     }
76
77 @Override
78 public Livreur getLivreurById(int id) throws Exception {
79     String sql = "SELECT * FROM livreurs WHERE id = ?";
80
81     try (Connection conn = DatabaseManager.getConnection();
82         PreparedStatement stmt = conn.prepareStatement(sql)) {
83
84         stmt.setInt(1, id);
85
86         try (ResultSet rs = stmt.executeQuery()) {
87             if (rs.next()) {
88                 return mapResultSetToLivreur(rs);
89             }
90         }
91     }
92     return null;
93 }
94
95 @Override
    
```



```

96     public List<Livreur> getAllLivres() throws Exception {
97         String sql = "SELECT * FROM livreurs ORDER BY nom, prenom";
98         List<Livreur> livreurs = new ArrayList<>();
99
100        try (Connection conn = DriverManager.getConnection();
101            PreparedStatement stmt = conn.prepareStatement(sql);
102            ResultSet rs = stmt.executeQuery()) {
103
104            while (rs.next()) {
105                livreurs.add(mapResultSetToLivreur(rs));
106            }
107        }
108
109        return livreurs;
110    }
111
112    /**
113     * Helper method to map ResultSet to Livreur object
114     */
115    private Livreur mapResultSetToLivreur(ResultSet rs) throws
116        SQLException {
117        Livreur livreur = new Livreur();
118        livreur.setId(rs.getInt("id"));
119        livreur.setNom(rs.getString("nom"));
120        livreur.setPrenom(rs.getString("prenom"));
121        livreur.setTelephone(rs.getString("telephone"));
122        return livreur;
123    }

```

Listing 19 – Implémentation LivreurDAOImpl complète

## 8.4 Services Complètes

### 8.4.1 Interface ILivreurService Complète

```

1  package com.example.deliveryapp.service;
2
3  import com.example.deliveryapp.entities.Livreur;
4  import java.util.List;
5
6  /**
7   * Service interface for Livreur business logic
8   */
9  public interface ILivreurService {
10     /**
11      * Add a new driver with validation
12      * @param livreur The driver to add
13      * @throws Exception if validation fails or operation fails
14      */

```

```

15     void addLivreur(Livreur livreur) throws Exception;
16
17     /**
18      * Update an existing driver with validation
19      * @param livreur The driver to update
20      * @throws Exception if validation fails or operation fails
21      */
22     void updateLivreur(Livreur livreur) throws Exception;
23
24     /**
25      * Delete a driver by ID
26      * @param id The driver ID to delete
27      * @throws Exception if operation fails
28      */
29     void deleteLivreur(int id) throws Exception;
30
31     /**
32      * Get a driver by ID
33      * @param id The driver ID
34      * @return The driver or null if not found
35      */
36     Livreur getLivreurById(int id);
37
38     /**
39      * Get all drivers
40      * @return List of all drivers
41      */
42     List<Livreur> getAllLivres();
43 }
    
```

Listing 20 – Interface ILivreurService complète

#### 8.4.2 Classe LivreurServiceImpl Complète

```

1  package com.example.deliveryapp.service;
2
3  import com.example.deliveryapp.dao.ILivreurDAO;
4  import com.example.deliveryapp.dao.LivreurDAOImpl;
5  import com.example.deliveryapp.entities.Livreur;
6
7  import java.util.ArrayList;
8  import java.util.List;
9
10 /**
11  * Implementation of ILivreurService
12  */
13 public class LivreurServiceImpl implements ILivreurService {
14     private final ILivreurDAO livreurDAO;
15
16     public LivreurServiceImpl() {
    
```

```

17         this.livreurDAO = new LivreurDAOImpl();
18     }
19
20     @Override
21     public void addLivreur(Livreur livreur) throws Exception {
22         validateLivreur(livreur);
23         livreurDAO.addLivreur(livreur);
24     }
25
26     @Override
27     public void updateLivreur(Livreur livreur) throws Exception {
28         validateLivreur(livreur);
29         if (livreur.getId() <= 0) {
30             throw new Exception("ID du livreur invalide pour la
31                                     mise      jour");
32         }
33         livreurDAO.updateLivreur(livreur);
34     }
35
36     @Override
37     public void deleteLivreur(int id) throws Exception {
38         if (id <= 0) {
39             throw new Exception("ID du livreur invalide");
40         }
41         livreurDAO.deleteLivreur(id);
42     }
43
44     @Override
45     public Livreur getLivreurById(int id) {
46         try {
47             return livreurDAO.getLivreurById(id);
48         } catch (Exception e) {
49             System.err.println("Error getting livreur by ID: " + e.
50                                     getMessage());
51             return null;
52         }
53     }
54
55     @Override
56     public List<Livreur> getAllLivres() {
57         try {
58             return livreurDAO.getAllLivres();
59         } catch (Exception e) {
60             System.err.println("Error getting all livres: " + e.
61                                     getMessage());
62             return new ArrayList<>();
63         }
64     }
65
66     /**

```

```

64     * Validate Livreur data
65     * @param livreur The livreur to validate
66     * @throws Exception if validation fails
67     */
68     private void validateLivreur(Livreur livreur) throws Exception
69     {
70         if (livreur == null) {
71             throw new Exception("Les donn es du livreur ne peuvent
72                 pas tre nulles");
73         }
74         if (livreur.getNom() == null || livreur.getNom().trim().
75             isEmpty()) {
76             throw new Exception("Le nom du livreur est obligatoire"
77                 );
78         }
79         if (livreur.getPrenom() == null || livreur.getPrenom().trim
80             ().isEmpty()) {
81             throw new Exception("Le pr nom du livreur est
82                 obligatoire");
83         }
84         if (livreur.getNom().length() > 255) {
85             throw new Exception("Le nom du livreur ne peut pas
86                 d passer 255 caract res");
87         }
88         if (livreur.getPrenom().length() > 255) {
89             throw new Exception("Le pr nom du livreur ne peut pas
90                 d passer 255 caract res");
91         }
92         if (livreur.getTelephone() != null && livreur.getTelephone
93             ().length() > 20) {
94             throw new Exception("Le num ro de t l phone ne peut
95                 pas d passer 20 caract res");
96         }
97     }
98 }

```

Listing 21 – Implémentation LivreurServiceImpl complète

## 8.5 Classe Utilitaire DatabaseManager

```

1 package com.example.deliveryapp.util;
2
3 import java.io.FileInputStream;
4 import java.io.IOException;
5 import java.sql.Connection;

```

```

6  import java.sql.DriverManager;
7  import java.sql.SQLException;
8  import java.util.Properties;
9
10 /**
11  * Database connection manager
12  */
13 public class DatabaseManager {
14     private static final String CONFIG_FILE = "config.properties";
15     private static String url;
16     private static String username;
17     private static String password;
18
19     static {
20         loadConfiguration();
21     }
22
23     /**
24     * Load database configuration from properties file
25     */
26     private static void loadConfiguration() {
27         Properties props = new Properties();
28         try (FileInputStream fis = new FileInputStream(CONFIG_FILE)) {
29             props.load(fis);
30             url = props.getProperty("db.url", "jdbc:mysql://localhost:3306/delivery_db");
31             username = props.getProperty("db.username", "root");
32             password = props.getProperty("db.password", "");
33         } catch (IOException e) {
34             // Use default values
35             System.err.println("Could not load configuration file, using defaults: " + e.getMessage());
36             url = "jdbc:mysql://localhost:3306/delivery_db";
37             username = "root";
38             password = "";
39         }
40     }
41
42     /**
43     * Get a database connection
44     * @return Database connection
45     * @throws SQLException if connection fails
46     */
47     public static Connection getConnection() throws SQLException {
48         try {
49             Class.forName("com.mysql.cj.jdbc.Driver");
50             return DriverManager.getConnection(url, username, password);
51         } catch (ClassNotFoundException e) {

```

```

52         throw new SQLException("MySQL JDBC Driver not found", e
53             );
54     }
55 }
56 /**
57  * Test database connection
58  * @return true if connection successful
59  */
60 public static boolean testConnection() {
61     try (Connection conn = getConnection()) {
62         return conn != null && !conn.isClosed();
63     } catch (SQLException e) {
64         System.err.println("Database connection test failed: "
65             + e.getMessage());
66         return false;
67     }
68 }
    
```

Listing 22 – Gestionnaire de base de données

## 8.6 Classe Utilitaire CSVExporter

```

1 package com.example.deliveryapp.util;
2
3 import com.example.deliveryapp.entities.Colis;
4 import java.time.format.DateTimeFormatter;
5 import java.util.List;
6
7 /**
8  * Utility class for CSV export functionality
9  */
10 public class CSVExporter {
11     private static final String CSV_SEPARATOR = ",";
12     private static final String CSV_HEADER = "ID, Destinataire,
13         Adresse, Date Envoi, Statut, Date Livraison, Livreur";
14     private static final DateTimeFormatter DATE_FORMATTER =
15         DateTimeFormatter.ofPattern("dd/MM/yyyy");
16
17     /**
18      * Export a list of Colis to CSV format
19      * @param colisList List of packages to export
20      * @return CSV content as string
21      */
22     public String exportColisToCSV(List<Colis> colisList) {
23         StringBuilder csvContent = new StringBuilder();
24         csvContent.append(CSV_HEADER).append("\n");
25
26         for (Colis colis : colisList) {
27
28         }
29     }
30 }
    
```

```

25         csvContent.append(formatColisToCSVLine(colis)).append("\n");
26     }
27
28     return csvContent.toString();
29 }
30
31 /**
32  * Format a single Colis object to CSV line
33  * @param colis The package to format
34  * @return CSV line string
35  */
36 private String formatColisToCSVLine(Colis colis) {
37     StringBuilder line = new StringBuilder();
38
39     line.append(colis.getId()).append(CSV_SEPARATOR);
40     line.append(escapeCSVField(colis.getDestinataire())).append(
41         CSV_SEPARATOR);
42     line.append(escapeCSVField(colis.getAdresse())).append(
43         CSV_SEPARATOR);
44     line.append(colis.getDateEnvoi().format(DATE_FORMATTER)).
45         append(CSV_SEPARATOR);
46     line.append(colis.getStatutText()).append(CSV_SEPARATOR);
47
48     if (colis.getDateLivraison() != null) {
49         line.append(colis.getDateLivraison().format(
50             DATE_FORMATTER));
51     }
52     line.append(CSV_SEPARATOR);
53
54     line.append(escapeCSVField(colis.getLivreurName()));
55
56     return line.toString();
57 }
58
59 /**
60  * Escape CSV field if it contains special characters
61  * @param field The field to escape
62  * @return Escaped field
63  */
64 private String escapeCSVField(String field) {
65     if (field == null) {
66         return "";
67     }
68
69     if (field.contains(CSV_SEPARATOR) || field.contains("\\"")
70         || field.contains("\n")) {
71         return "\"" + field.replace("\\"", "\\\"") + "\"";
72     }
73 }

```

```

69         return field;
70     }
71 }
    
```

Listing 23 – Exportateur CSV

## 8.7 Classe MainApp

```

1  package com.example.deliveryapp.app;
2
3  import javafx.application.Application;
4  import javafx.fxml.FXMLLoader;
5  import javafx.scene.Scene;
6  import javafx.scene.image.Image;
7  import javafx.stage.Stage;
8
9  import java.io.IOException;
10
11 /**
12  * Main JavaFX Application class
13  */
14 public class MainApp extends Application {
15
16     @Override
17     public void start(Stage stage) throws IOException {
18         try {
19             FXMLLoader fxmlLoader = new FXMLLoader(MainApp.class.
20                 getResource("/fxml/MainView.fxml"));
21             Scene scene = new Scene(fxmlLoader.load(), 1200, 800);
22
23             // Load CSS stylesheet
24             scene.getStylesheets().add(MainApp.class.getResource("/
25                 css/styles.css").toExternalForm());
26
27             stage.setTitle("Gestion de Livraison de Colis");
28             stage.setScene(scene);
29             stage.setMinWidth(1000);
30             stage.setMinHeight(700);
31
32             // Set application icon (optional)
33             try {
34                 Image icon = new Image(MainApp.class.
35                     getResourceAsStream("/images/icon.png"));
36                 stage.getIcons().add(icon);
37             } catch (Exception e) {
38                 // Icon not found, continue without it
39                 System.out.println("Application icon not found,
40                     continuing without it.");
41             }
42         }
43     }
44 }
    
```



```

39         stage.show();
40
41     } catch (Exception e) {
42         e.printStackTrace();
43         System.err.println("Failed to start application: " + e.
44             getMessage());
45     }
46 }
47
48 public static void main(String[] args) {
49     launch();
50 }
    
```

Listing 24 – Classe principale de l'application

## 9 Tests

Le projet inclut des tests unitaires pour valider le fonctionnement des services :

```

1 package com.example.deliveryapp.service;
2
3 import com.example.deliveryapp.entities.Livreur;
4 import org.junit.jupiter.api.BeforeEach;
5 import org.junit.jupiter.api.Test;
6 import static org.junit.jupiter.api.Assertions.*;
7
8 public class LivreurServiceTest {
9     private ILivreurService livreurService;
10
11     @BeforeEach
12     void setUp() {
13         livreurService = new LivreurServiceImpl();
14     }
15
16     @Test
17     void testAddLivreur() {
18         Livreur livreur = new Livreur("Dupont", "Jean", "0123456789
19             ");
20
21         assertDoesNotThrow(() -> {
22             livreurService.addLivreur(livreur);
23         });
24
25     }
26
27     @Test
28     void testValidationLivreurWithEmptyFields() {
29         Livreur livreur = new Livreur("", "", "");
30
31         Exception exception = assertThrows(Exception.class, () -> {
    
```

```

30         livreurService.addLivreur(livreur);
31     });
32
33     assertTrue(exception.getMessage().contains("obligatoire"));
34 }
35
36 @Test
37 void testValidationLivreurWithNullFields() {
38     Livreur livreur = new Livreur(null, null, null);
39
40     Exception exception = assertThrows(Exception.class, () -> {
41         livreurService.addLivreur(livreur);
42     });
43
44     assertTrue(exception.getMessage().contains("obligatoire"));
45 }
46
47 @Test
48 void testValidationLivreurWithLongFields() {
49     String longName = "a".repeat(300); // More than 255
50     // characters
51     Livreur livreur = new Livreur(longName, "Jean", "0123456789
52     ");
53
54     Exception exception = assertThrows(Exception.class, () -> {
55         livreurService.addLivreur(livreur);
56     });
57
58     assertTrue(exception.getMessage().contains("255 caract res
59     "));
60 }
61
62 @Test
63 void testGetAllLivres() {
64     assertDoesNotThrow(() -> {
65         var livres = livreurService.getAllLivres();
66         assertNotNull(livres);
67     });
68 }
69 }
    
```

Listing 25 – Tests unitaires pour LivreurService

## 10 Conclusion

### 10.1 Résumé des Réalisations

Cette application de gestion de livraison de colis représente une solution complète développée en Java avec JavaFX. Elle démontre :

- Une architecture propre et modulaire suivant les principes de la programmation orientée objet
- Une séparation claire des responsabilités avec le pattern MVC
- Une interface utilisateur intuitive et responsive
- Une gestion robuste des erreurs et de la validation
- L'intégration avec une base de données relationnelle
- Des fonctionnalités d'export de données

## 10.2 Points Forts

- **Architecture claire** : Séparation en couches bien définie
- **Code maintenable** : Utilisation d'interfaces et de bonnes pratiques
- **Interface utilisateur moderne** : JavaFX avec CSS pour le styling
- **Gestion des erreurs** : Validation complète et messages explicites
- **Fonctionnalités complètes** : CRUD, assignation, export CSV

## 10.3 Améliorations Possibles

- Intégration d'un système de notifications
- Ajout d'un système d'authentification
- Implémentation d'un système de rapports plus avancé
- Ajout de fonctionnalités de recherche et de filtrage
- Intégration avec des services de géolocalisation