École Marocaine des Sciences de l'Ingénieur

Système de Billetterie et d'Accréditation Biométrique pour le Mondial 2030

Intitulé du module :
Java Avancé

Filière :
4IIR

Réalisé par :
Dounia Chraoua

Encadré par :
Abderrahim Larhlimi

Année Universitaire 2025/2026

11 janvier 2026

Remerciements

Je tiens à exprimer ma sincère gratitude envers toutes les personnes qui ont contribué de près ou de loin à la réalisation de ce projet.

Je remercie tout particulièrement M. Abderrahim Larhlimi, mon encadrant, pour son encadrement, ses précieuses orientations et sa disponibilité tout au long de la réalisation de ce projet.

Je remercie également l'administration de l'EMSI pour avoir mis à disposition les ressources nécessaires à la réalisation de ce travail.

Enfin, je remercie tous ceux qui m'ont soutenue et encouragée durant cette période.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction Générale

1.1 Contexte du projet

Ce projet s'inscrit dans le cadre du module *Java Avancé* de la filière 4IIR à l'École Marocaine des Sciences de l'Ingénieur (EMSI) pour l'année universitaire 2025/2026.

Le projet consiste en le développement d'une application desktop de gestion de billetterie et d'accréditation biométrique pour le Mondial 2030, un événement sportif majeur nécessitant une gestion efficace des accès et une sécurité renforcée.

Le système doit permettre la gestion complète du cycle de vie des tickets, depuis leur création jusqu'à leur utilisation, en intégrant des mécanismes de sécurité avancés incluant la biométrie faciale et la génération de codes QR.

1.2 Problématique

Avec l'organisation d'un événement de grande envergure comme le Mondial 2030, plusieurs défis doivent être relevés :

- **Gestion des accès :** Comment garantir que seules les personnes autorisées accèdent aux installations ?
- **Sécurisation des tickets :** Comment éviter la contrefaçon et la duplication des tickets ?
- **Gestion des matchs :** Comment permettre aux administrateurs de créer et gérer facilement les événements sportifs ?
- **Vérification rapide :** Comment permettre au personnel de sécurité de vérifier rapidement et efficacement les accès ?
- **Traçabilité :** Comment conserver un historique de tous les accès pour des raisons de sécurité et d'analyse ?

1.3 Objectifs

Le présent projet vise à développer un système complet permettant :

1.3.1 Objectifs fonctionnels

1. Gérer les utilisateurs avec des rôles distincts (STAFF et SUPPORTER)

2. Permettre l'achat de tickets pour les supporters
3. Générer automatiquement des QR Codes pour chaque ticket
4. Intégrer un système biométrique (empreinte faciale)
5. Contrôler l'accès via vérification combinée (ticket + QR Code + biométrie)
6. Permettre au STAFF de créer et gérer les matchs
7. Scanner et valider les QR Codes pour le contrôle d'accès
8. Conserver un journal complet de tous les accès
9. Enregistrer les tickets validés avec photo et QR Code

1.3.2 Objectifs techniques

1. Utiliser JavaFX pour une interface utilisateur moderne
2. Implémenter une architecture MVC propre
3. Utiliser le pattern DAO pour l'accès aux données
4. Assurer la sécurité avec le hashage BCrypt des mots de passe
5. Utiliser MySQL pour la persistance des données
6. Appliquer les bonnes pratiques de développement Java

Première partie

Analyse et Conception

Chapitre 2

Spécification des besoins

2.1 Besoins fonctionnels

2.1.1 Gestion des utilisateurs

— Le système doit permettre l'inscription de nouveaux utilisateurs
— Le système doit permettre la connexion avec authentification sécurisée
— Le système doit afficher un tableau de bord personnalisé selon le rôle
— Le système doit permettre la déconnexion

2.1.2 Gestion des tickets (SUPPORTER)

— Le système doit permettre la consultation de tous les tickets achetés par l'utilisateur
— Le système doit permettre l'achat de tickets pour un match sélectionné
— Le système doit générer automatiquement un code de ticket unique
— Le système doit générer automatiquement un QR Code pour chaque ticket
— Le système doit afficher les détails et le QR Code d'un ticket sélectionné
— Le système doit permettre l'enregistrement du ticket validé en image PNG

2.1.3 Gestion biométrique (SUPPORTER)

— Le système doit permettre le téléchargement d'une photo faciale
— Le système doit créer une empreinte faciale à partir de l'image
— Le système doit assigner la biométrie au compte utilisateur
— Le système doit générer un QR Code personnel pour l'utilisateur
— Le système doit vérifier l'accès en combinant ticket + QR Code + biométrie
— Le système doit afficher le ticket validé avec photo et QR Code

2.1.4 Gestion des matchs (STAFF)

— Le système doit permettre la création de nouveaux matchs avec toutes les informations nécessaires
— Le système doit permettre la consultation de tous les matchs créés
— Le système doit gérer la disponibilité des tickets pour chaque match

2.1.5 Contrôle d'accès (STAFF)

— Le système doit permettre le scan/saisie de QR Codes de tickets
— Le système doit vérifier la validité du ticket
— Le système doit autoriser ou refuser l'accès avec messages explicites
— Le système doit marquer automatiquement le ticket comme utilisé après validation
— Le système doit enregistrer toutes les tentatives d'accès dans un journal

2.1.6 Journal des accès (STAFF)

— Le système doit conserver un historique de tous les accès
— Le système doit afficher les informations : utilisateur, type d'accès, résultat, date/-heure
— Le système doit inclure les scans effectués par les SUPPORTER

2.2 Besoins non-fonctionnels

2.2.1 Sécurité

— Les mots de passe doivent être hashés avec BCrypt (10 rounds minimum)
— Les requêtes SQL doivent utiliser PreparedStatement pour éviter les injections SQL
— Le contrôle d'accès doit être basé sur les rôles
— Les comptes inactifs doivent être protégés contre la connexion

2.2.2 Performance

— Le temps de réponse des opérations de base de données doit être ¡ 1 seconde
— L'interface utilisateur doit rester réactive pendant les opérations
— Les requêtes doivent être optimisées avec des index appropriés

2.2.3 Ergonomie

— L'interface doit être intuitive et facile à utiliser
— Les messages d'erreur doivent être clairs et explicites
— La navigation doit être fluide avec des boutons de retour appropriés
— L'affichage doit s'adapter selon le rôle de l'utilisateur

2.2.4 Fiabilité

— Le système doit gérer correctement les erreurs sans planter
— Les données doivent être validées avant enregistrement
— Les transactions doivent garantir l'intégrité des données

Chapitre 3

Conception UML

3.1 Diagramme de classes

Le diagramme de classes présente l'architecture objet du système avec les principales entités et leurs relations.

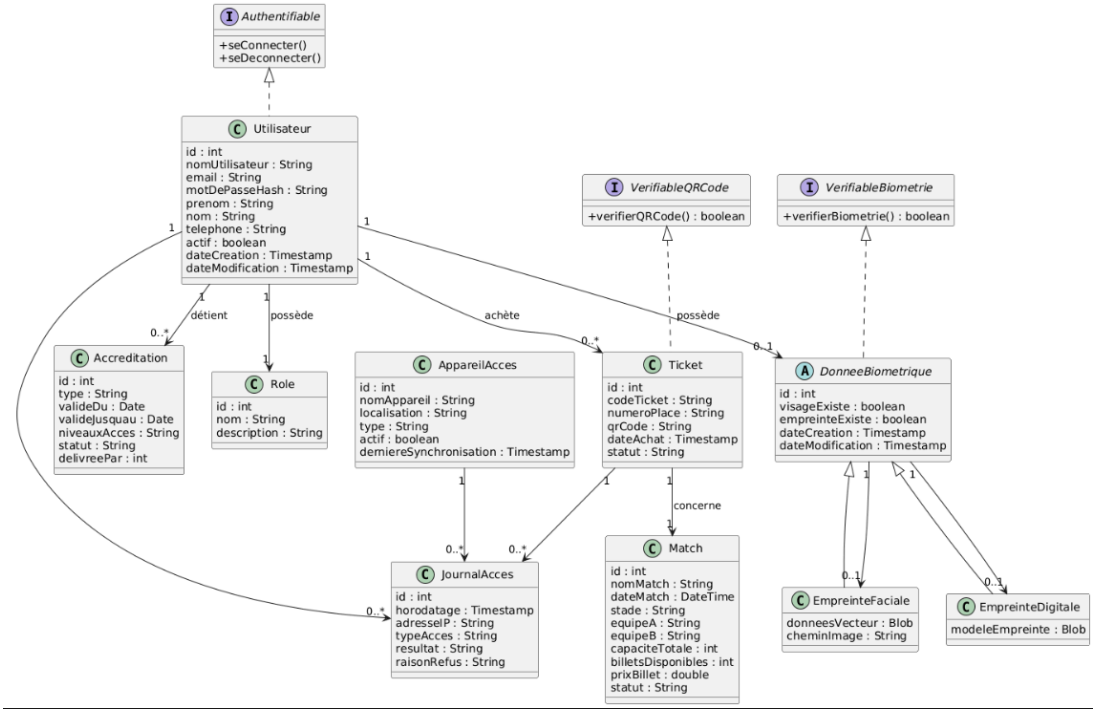

FIGURE 3.1 – Diagramme de classes du système

3.1.1 Classes principales

Package model

Les classes du modèle de données représentent les entités métier :

— **User** : Représente un utilisateur du système avec ses attributs (username, email, password_hash, etc.)

— **Role** : Représente un rôle (STAFF, SUPPORTER)

— **Ticket** : Représente un ticket acheté avec son statut

— **MatchEvent** : Représente un match/événement sportif
— **AccessLog** : Représente une entrée du journal d'accès
— **BiometricData** : Représente les données biométriques d'un utilisateur
— **GateDevice** : Représente un dispositif de contrôle d'accès

Package dao

Les classes DAO (Data Access Object) gèrent l'accès aux données :
— **UserDAO** : Accès aux données des utilisateurs
— **TicketDAO** : Accès aux données des tickets
— **MatchEventDAO** : Accès aux données des matchs
— **AccessLogDAO** : Accès aux logs d'accès
— **BiometricDataDAO** : Accès aux données biométriques
— **RoleDAO** : Accès aux rôles
— **GateDeviceDAO** : Accès aux dispositifs

Package service

Les services contiennent la logique métier :
— **AuthenticationService** : Gestion de l'authentification
— **TicketService** : Logique métier des tickets
— **MatchEventService** : Logique métier des matchs
— **BiometricService** : Logique métier de la biométrie
— **AccessControlService** : Contrôle d'accès et validation
— **UserService** : Gestion des utilisateurs

Package controller

Les contrôleurs gèrent l'interface utilisateur :
— **LoginController** : Contrôleur de l'écran de connexion
— **RegisterController** : Contrôleur de l'écran d'inscription
— **DashboardController** : Contrôleur du tableau de bord
— **TicketsController** : Contrôleur de la gestion des tickets
— **BiometricController** : Contrôleur de la biométrie
— **ScanController** : Contrôleur du scan QR Code
— **AdminController** : Contrôleur de l'administration

3.1.2 Relations entre classes

— **User** → **Role** : Association (un utilisateur a un rôle)
— **Ticket** → **User** : Association (un ticket appartient à un utilisateur)
— **Ticket** → **MatchEvent** : Association (un ticket correspond à un match)
— **AccessLog** → **User** : Association (un log est lié à un utilisateur)
— **AccessLog** → **Ticket** : Association (un log peut être lié à un ticket)
— **BiometricData** → **User** : Association un-à-un (une biométrie par utilisateur)

3.1.3 Schéma relationnel

1. **roles** (id : PK, name, description, created_at)

2. **users** (id : PK, username, email, password_hash, first_name, last_name, phone, role_id : FK → roles.id, is_active, created_at, updated_at)
3. **match_events** (id : PK, match_name, match_date, venue, team_a, team_b, total_capacity, available_tickets, ticket_price, status, created_at, updated_at)
4. **tickets** (id : PK, ticket_code, user_id : FK → users.id, match_event_id : FK → match_events.id, seat_number, qr_code_data, purchase_date, status)
5. **biometric_data** (id : PK, user_id : FK → users.id, has_face_data, has_fingerprint_data, created_at, updated_at)
6. **face_embeddings** (id : PK, biometric_data_id : FK → biometric_data.id, embedding_data, image_path, created_at)
7. **gate_devices** (id : PK, device_name, device_location, device_type, is_active, last_sync, created_at)
8. **access_logs** (id : PK, user_id : FK → users.id, ticket_id : FK → tickets.id, gate_device_id : FK → gate_devices.id, access_type, access_result, denial_reason, access_timestamp, ip_address)
9. **accreditations** (id : PK, user_id : FK → users.id, accreditation_type, valid_from, valid_until, access_levels, status, issued_by : FK → users.id, created_at, updated_at)

3.2 Dictionnaire de données

3.2.1 Table : users

Champ	Type	Taille	Description / Contraintes
id	INT	-	Clé primaire, auto-incrémentée
username	VARCHAR	100	Nom d'utilisateur unique, non nul
email	VARCHAR	255	Email unique, non nul
password_hash	VARCHAR	255	Hash BCrypt du mot de passe, non nul
first_name	VARCHAR	100	Prénom, non nul
last_name	VARCHAR	100	Nom, non nul
phone	VARCHAR	20	Numéro de téléphone (optionnel)
role_id	INT	-	Clé étrangère vers roles.id, non nul
is_active	BOOLEAN	-	Statut actif/inactif, défaut TRUE
created_at	TIMESTAMP	-	Date de création, automatique
updated_at	TIMESTAMP	-	Date de mise à jour, automatique

3.2.2 Table : tickets

Champ	Type	Taille	Description / Contraintes
id	INT	-	Clé primaire, auto-incrémentée

ticket_code	VARCHAR	50	Code unique du ticket (format TKT-YYYYMMDDHHmmss-UUID), non nul, unique
user_id	INT	-	Clé étrangère vers users.id, non nul, CASCADE DELETE
match_event_id	INT	-	Clé étrangère vers match_events.id, non nul, RESTRICT DELETE
seat_number	VARCHAR	20	Numéro de siège (optionnel)
qr_code_data	TEXT	-	Données du QR Code encodées (optionnel)
purchase_date	TIMESTAMP		Date d'achat, automatique
status	ENUM	-	VALID, USED, CANCELLED, EXPIRED, défaut VALID

3.2.3 Table : match_events

Champ	**Type**	**Taille**	**Description / Contraintes**
id	INT	-	Clé primaire, auto-incrémentée
match_name	VARCHAR	255	Nom du match, non nul
match_date	DATETIME	-	Date et heure du match, non nul
venue	VARCHAR	255	Lieu du match, non nul
team_a	VARCHAR	100	Équipe A, non nul
team_b	VARCHAR	100	Équipe B, non nul
total_capacity	INT	-	Capacité totale du stade, non nul
available_tickets	INT	-	Nombre de tickets disponibles, non nul
ticket_price	DECIMAL	10,2	Prix du ticket, non nul
status	ENUM	-	UPCOMING, ONGOING, COMPLETED, CANCELLED, défaut UPCOMING
created_at	TIMESTAMP		Date de création, automatique
updated_at	TIMESTAMP		Date de mise à jour, automatique

Deuxième partie

Environnement Technique

Chapitre 4

Environnement de Développement

4.1 Langage de programmation

Le projet est développé en **Java** version **21 (LTS)**, avec l'utilisation de **JavaFX 21** pour l'interface graphique.

4.1.1 Justification du choix de Java

— Langage orienté objet mature et robuste
— Grande communauté et documentation abondante
— Compatibilité multiplateforme (Windows, Linux, macOS)
— Gestion mémoire automatique avec le Garbage Collector
— Support natif des threads pour des applications réactives

4.1.2 Justification du choix de JavaFX

— Framework moderne pour applications desktop
— Séparation claire entre logique et présentation (FXML)
— Possibilité de créer des interfaces modernes et attrayantes
— Support intégré des événements et binding de propriétés
— Intégration facile avec CSS pour le styling

4.2 Environnement de développement (IDE)

IntelliJ IDEA est utilisé comme environnement de développement intégré.

4.2.1 Fonctionnalités utilisées

— Autocomplétion intelligente
— Debugging intégré
— Gestion des projets Maven
— Refactoring automatique
— Intégration Git
— Analyse de code en temps réel

4.3 Gestion de projet et Build

Apache Maven version 3.9+ est utilisé pour la gestion des dépendances et la compilation du projet.

4.3.1 Dépendances principales (extrait du pom.xml)

Listing 4.1 – Extrait du pom.xml - Dépendances principales

```
<properties>
    <javafx.version>21</javafx.version>
    <mysql.version>8.0.33</mysql.version>
    <bcrypt.version>0.4</bcrypt.version>
    <zxing.version>3.5.2</zxing.version>
</properties>

<dependencies>
    <!-- JavaFX -->
    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-controls</artifactId>
        <version>${javafx.version}</version>
    </dependency>
    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-fxml</artifactId>
        <version>${javafx.version}</version>
    </dependency>

    <!-- MySQL Connector -->
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <version>${mysql.version}</version>
    </dependency>

    <!-- BCrypt pour le hashage -->
    <dependency>
        <groupId>org.mindrot</groupId>
        <artifactId>jbcrypt</artifactId>
        <version>${bcrypt.version}</version>
    </dependency>

    <!-- ZXing pour QR Codes -->
    <dependency>
        <groupId>com.google.zxing</groupId>
        <artifactId>core</artifactId>
        <version>${zxing.version}</version>
    </dependency>
</dependencies>
```

4.3.2 Configuration du compilateur

Listing 4.2 – Configuration Maven Compiler

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.11.0</version>
    <configuration>
        <source>21</source>
        <target>21</target>
        <release>21</release>
    </configuration>
</plugin>
```

4.4 Système de Gestion de Base de Données

MySQL 8.0+ est utilisé comme SGBD relationnel.

4.4.1 Justification

— Open-source et gratuit
— Performance élevée pour les applications web et desktop
— Support complet du SQL standard
— Outils d'administration riches (phpMyAdmin, MySQL Workbench)
— Compatibilité avec JDBC

4.4.2 Driver JDBC

Le driver **MySQL Connector/J 8.0.33** est utilisé pour la connexion Java-MySQL.

4.5 Outils de modélisation

— **MySQL Workbench** : Pour la conception et la modélisation de la base de données
— **StarUML** : Pour la création des diagrammes UML (classes, cas d'utilisation)
— **IntelliJ IDEA** : Pour le développement et le refactoring

Troisième partie

Architecture et Implémentation

Chapitre 5

Architecture Logicielle

5.1 Organisation des packages

Le projet suit une architecture en couches (layered architecture) avec séparation des responsabilités selon le pattern MVC.

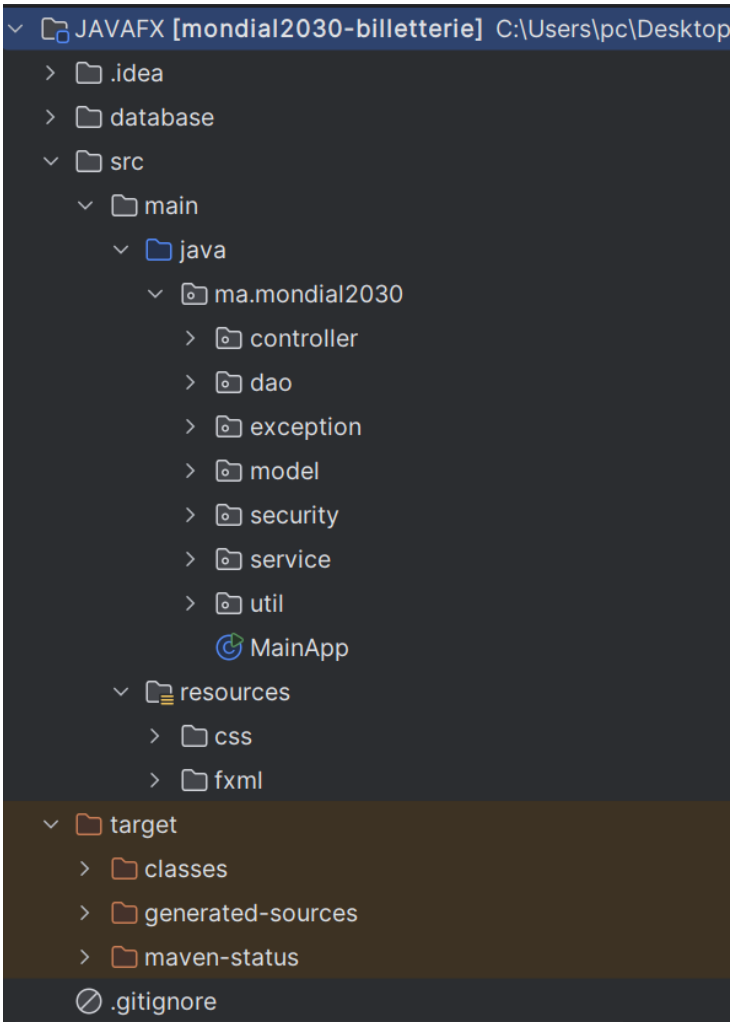

FIGURE 5.1 – Organisation des packages

5.1.1 Structure des packages

Listing 5.1 – Structure des packages

```
ma.mondial2030/
        MainApp.java                # Point d'entr e de
   l'application
        controller/                 # Couche pr sentation (Vue)
            LoginController.java
            RegisterController.java
            DashboardController.java
            TicketsController.java
            BiometricController.java
            ScanController.java
            AdminController.java
        service/                    # Couche logique m tier
            AuthenticationService.java
            TicketService.java
            MatchEventService.java
            BiometricService.java
            AccessControlService.java
            UserService.java
        dao/                        # Couche acc s aux donn es
            UserDAO.java
            TicketDAO.java
            MatchEventDAO.java
            AccessLogDAO.java
            BiometricDataDAO.java
            RoleDAO.java
            GateDeviceDAO.java
        model/                      # Mod les de donn es
            User.java
            Role.java
            Ticket.java
            MatchEvent.java
            AccessLog.java
            BiometricData.java
            GateDevice.java
        util/                       # Utilitaires
            DatabaseConnection.java
            QRCodeGenerator.java
            QRCodeReader.java
            PasswordHasher.java
        exception/                  # Exceptions personnalis es
            AuthenticationException.java
            DatabaseException.java
        security/                   # S curit
            PasswordHasher.java
```

5.2 Séparation des couches

5.2.1 Couche Vue (View)

Responsable de l'affichage et de l'interaction utilisateur.
— Fichiers FXML dans `resources/fxml/`
— Contrôleurs JavaFX dans `controller/`
— Styles CSS dans `resources/css/`

5.2.2 Couche Contrôleur (Controller)

Gère les événements utilisateur et coordonne avec la couche service.
— Traite les actions des utilisateurs
— Appelle les services appropriés
— Met à jour l'interface utilisateur

5.2.3 Couche Service

Contient la logique métier de l'application.
— Valide les données
— Applique les règles métier
— Coordonne les appels aux DAO
— Gère les transactions

5.2.4 Couche DAO

Abstrait l'accès à la base de données.
— Encapsule les requêtes SQL
— Mappe les résultats aux objets Java
— Gère les erreurs de base de données

5.2.5 Couche Model

Représente les entités métier.
— Classes POJO (Plain Old Java Objects)
— Getters et Setters
— Méthodes utilitaires

Chapitre 6

Design Patterns (Patrons de Conception)

6.1 Pattern Singleton

Le pattern Singleton est utilisé pour la connexion à la base de données afin de garantir une unique instance de connexion dans toute l'application.

Listing 6.1 – DatabaseConnection.java - Pattern Singleton

```
public class DatabaseConnection {
    private static DatabaseConnection instance;
    private Connection connection;

    private DatabaseConnection() {
        // Constructeur priv  pour emp cher l'instanciation
    }

    public static DatabaseConnection getInstance() {
        if (instance == null) {
            synchronized (DatabaseConnection.class) {
                if (instance == null) {
                    instance = new DatabaseConnection();
                }
            }
        }
        return instance;
    }

    public Connection getConnection() throws SQLException {
        if (connection == null || connection.isClosed()) {
            String url =
               "jdbc:mysql://localhost:3306/mondial2030_db";
            connection = DriverManager.getConnection(url,
               username, password);
        }
        return connection;
    }
}
```

Justification : Une seule instance de connexion évite la surcharge de connexions multiples et garantit une gestion efficace des ressources.

6.2 Pattern DAO (Data Access Object)

Le pattern DAO sépare la logique d'accès aux données de la logique métier, permettant une meilleure maintenabilité et testabilité.

Listing 6.2 – UserDAO.java - Exemple du pattern DAO

```
public class UserDAO {
    public User findByUsername(String username) {
        String sql = "SELECT * FROM users WHERE username = ?";
        try (Connection conn = DatabaseConnection.getInstance()
                .getConnection();
             PreparedStatement stmt =
                conn.prepareStatement(sql)) {

            stmt.setString(1, username);
            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    return mapResultSetToUser(rs);
                }
            }
        } catch (SQLException e) {
            logger.error("Erreur lors de la recherche
               utilisateur", e);
        }
        return null;
    }

    private User mapResultSetToUser(ResultSet rs) throws
       SQLException {
        // Mapping des donn es du ResultSet vers l'objet User
        User user = new User();
        user.setId(rs.getInt("id"));
        user.setUsername(rs.getString("username"));
        // ... autres attributs
        return user;
    }
}
```

Avantages :

- Isolation de la logique SQL
- Facilite les changements de SGBD
- Réutilisabilité du code
- Testabilité améliorée

6.3 Pattern MVC (Model-View-Controller)

Le pattern MVC est appliqué à travers toute l'application pour séparer les préoccupations.

— **Model :** Classes dans `model/` (User, Ticket, MatchEvent, etc.)
— **View :** Fichiers FXML dans `resources/fxml/`
— **Controller :** Classes dans `controller/`

6.3.1 Exemple de flux MVC

1. L'utilisateur clique sur un bouton dans la Vue (FXML)
2. L'événement est capturé par le Controller
3. Le Controller appelle le Service approprié
4. Le Service utilise les DAO pour accéder aux données
5. Les données sont retournées au Controller
6. Le Controller met à jour la Vue

Chapitre 7

Extraits de Code Clés

7.1 Gestion de la connexion à la base de données

Listing 7.1 – DatabaseConnection.java - Gestion sécurisée de la connexion

```
public class DatabaseConnection {
    private static final String DB_URL =
        "jdbc:mysql://localhost:3306/mondial2030_db?useSSL=false&serverTimez
    private static final String DB_USER = "root";
    private static final String DB_PASSWORD =
       "votre_mot_de_passe";

    private static DatabaseConnection instance;
    private Connection connection;

    private DatabaseConnection() {
        // Constructeur priv
    }

    public static synchronized DatabaseConnection getInstance() {
        if (instance == null) {
            instance = new DatabaseConnection();
        }
        return instance;
    }

    public Connection getConnection() throws SQLException {
        if (connection == null || connection.isClosed()) {
            try {
                connection = DriverManager.getConnection(DB_URL,
                   DB_USER, DB_PASSWORD);
                logger.info("Connexion    la base de donn es
                    tablie   avec succ s");
            } catch (SQLException e) {
                logger.error("Erreur lors de la connexion    la
                   base de donn es", e);
                throw e;
            }
```

```
30          }
31          return connection;
32      }
33
34      public void closeConnection() {
35          try {
36              if (connection != null && !connection.isClosed()) {
37                  connection.close();
38                  logger.info("Connexion ferm e");
39              }
40          } catch (SQLException e) {
41              logger.error("Erreur lors de la fermeture de la
                    connexion", e);
42          }
43      }
44  }
```

7.2 Requête PreparedStatement avec jointure

Listing 7.2 – TicketDAO.java - Requête avec jointure et gestion ResultSet

```
1  public List<Ticket> findByUserId(int userId) {
2      List<Ticket> tickets = new ArrayList<>();
3      String sql = "SELECT * FROM tickets WHERE user_id = ? ORDER
           BY purchase_date DESC";
4
5      try (Connection conn =
           DatabaseConnection.getInstance().getConnection();
6            PreparedStatement stmt = conn.prepareStatement(sql)) {
7
8          stmt.setInt(1, userId);
9          try (ResultSet rs = stmt.executeQuery()) {
10             // Lire toutes les donn es primitives d'abord
11             List<TicketData> ticketDataList = new ArrayList<>();
12             while (rs.next()) {
13                 int id = rs.getInt("id");
14                 String ticketCode = rs.getString("ticket_code");
15                 String seatNumber = rs.getString("seat_number");
16                 String qrCodeData = rs.getString("qr_code_data");
17                 String statusStr = rs.getString("status");
18                 int ticketUserId = rs.getInt("user_id");
19                 int matchEventId = rs.getInt("match_event_id");
20                 Timestamp purchaseDate =
                       rs.getTimestamp("purchase_date");
21
22                 ticketDataList.add(new TicketData(id,
                       ticketCode, seatNumber,
23                         qrCodeData, statusStr, ticketUserId,
                               matchEventId, purchaseDate));
24             }
```

```
25
26              // Construire les objets Ticket apr s fermeture du
                   ResultSet
27              for (TicketData data : ticketDataList) {
28                  User user = userDAO.findById(data.userId);
29                  MatchEvent matchEvent =
                       matchEventDAO.findById(data.matchEventId);
30
31                  Ticket ticket = new Ticket();
32                  ticket.setId(data.id);
33                  ticket.setTicketCode(data.ticketCode);
34                  ticket.setSeatNumber(data.seatNumber);
35                  ticket.setQrCodeData(data.qrCodeData);
36                  ticket.setStatus(Ticket.Status.valueOf(data.statusStr));
37                  ticket.setUser(user);
38                  ticket.setMatchEvent(matchEvent);
39                  if (data.purchaseDate != null) {
40                      ticket.setPurchaseDate(data.purchaseDate.toLocalDateTime
41                  }
42
43                  tickets.add(ticket);
44              }
45          }
46      } catch (SQLException e) {
47          logger.error("Erreur lors de la r cup ration des
                tickets", e);
48      }
49      return tickets;
50  }
```

Points clés :

— Utilisation de `try-with-resources` pour la gestion automatique des ressources

— Extraction des données primitives avant les appels DAO pour éviter les problèmes de ResultSet fermé

— Utilisation de PreparedStatement pour éviter les injections SQL

7.3 Gestion des exceptions avec try-with-resources

Listing 7.3 – Exemple de gestion sécurisée des ressources

```
1  public boolean create(Ticket ticket) {
2      String sql = "INSERT INTO tickets (ticket_code, user_id,
           match_event_id, " +
3                   "seat_number, qr_code_data, status) VALUES (?,
                      ?, ?, ?, ?, ?)";
4
5      try (Connection conn =
           DatabaseConnection.getInstance().getConnection();
6            PreparedStatement stmt = conn.prepareStatement(sql,
7                Statement.RETURN_GENERATED_KEYS)) {
8
```

```
9           stmt.setString(1, ticket.getTicketCode());
10          stmt.setInt(2, ticket.getUser().getId());
11          stmt.setInt(3, ticket.getMatchEvent().getId());
12          stmt.setString(4, ticket.getSeatNumber());
13          stmt.setString(5, ticket.getQrCodeData());
14          stmt.setString(6, ticket.getStatus().name());
15
16          int affectedRows = stmt.executeUpdate();
17          if (affectedRows > 0) {
18              try (ResultSet generatedKeys =
                    stmt.getGeneratedKeys()) {
19                  if (generatedKeys.next()) {
20                      ticket.setId(generatedKeys.getInt(1));
21                      return true;
22                  }
23              }
24          }
25      } catch (SQLException e) {
26          logger.error("Erreur lors de la cr ation du ticket", e);
27      }
28      return false;
29  }
```

Avantages du try-with-resources :

— Fermeture automatique des ressources (Connection, PreparedStatement, ResultSet)
— Code plus propre et moins sujet aux erreurs
— Gestion automatique des exceptions

7.4 Génération et lecture de QR Codes

Listing 7.4 – QRCodeGenerator.java - Génération de QR Code

```
1   public class QRCodeGenerator {
2       public static Image generateQRCode(String data) {
3           try {
4               QRCodeWriter qrCodeWriter = new QRCodeWriter();
5               BitMatrix bitMatrix = qrCodeWriter.encode(data,
6                   BarcodeFormat.QR_CODE, 200, 200);
7
8               int width = bitMatrix.getWidth();
9               int height = bitMatrix.getHeight();
10              BufferedImage image = new BufferedImage(width,
                    height,
11                  BufferedImage.TYPE_INT_RGB);
12
13              for (int x = 0; x < width; x++) {
14                  for (int y = 0; y < height; y++) {
15                      image.setRGB(x, y, bitMatrix.get(x, y) ?
16                          0xFF000000 : 0xFFFFFFFF);
17                  }
```

```
18              }
19
20              return SwingFXUtils.toFXImage(image, null);
21          } catch (Exception e) {
22              logger.error("Erreur lors de la g n ration du QR
                   Code", e);
23              return null;
24          }
25      }
26
27      public static String generateTicketQRData(String ticketCode,
28              int userId, int matchEventId) {
29          return String.format("TICKET:%s:USER:%d:MATCH:%d",
30              ticketCode, userId, matchEventId);
31      }
32  }
```

7.5 Hashage des mots de passe avec BCrypt

Listing 7.5 – PasswordHasher.java - Sécurité des mots de passe

```
1   public class PasswordHasher {
2       private static final int BCRYPT_ROUNDS = 10;
3
4       public static String hashPassword(String plainPassword) {
5           return BCrypt.hashpw(plainPassword,
                BCrypt.gensalt(BCRYPT_ROUNDS));
6       }
7
8       public static boolean verifyPassword(String plainPassword,
9               String hashedPassword) {
10          try {
11              return BCrypt.checkpw(plainPassword, hashedPassword);
12          } catch (Exception e) {
13              logger.error("Erreur lors de la v rification du mot
                   de passe", e);
14              return false;
15          }
16      }
17  }
```

Sécurité :

- BCrypt avec 10 rounds pour un bon compromis sécurité/performance
- Chaque hash est unique même pour le même mot de passe
- Résistant aux attaques par force brute

Quatrième partie

Interface Utilisateur et Tests

Chapitre 8

Présentation des Interfaces

8.1 Écran de Connexion

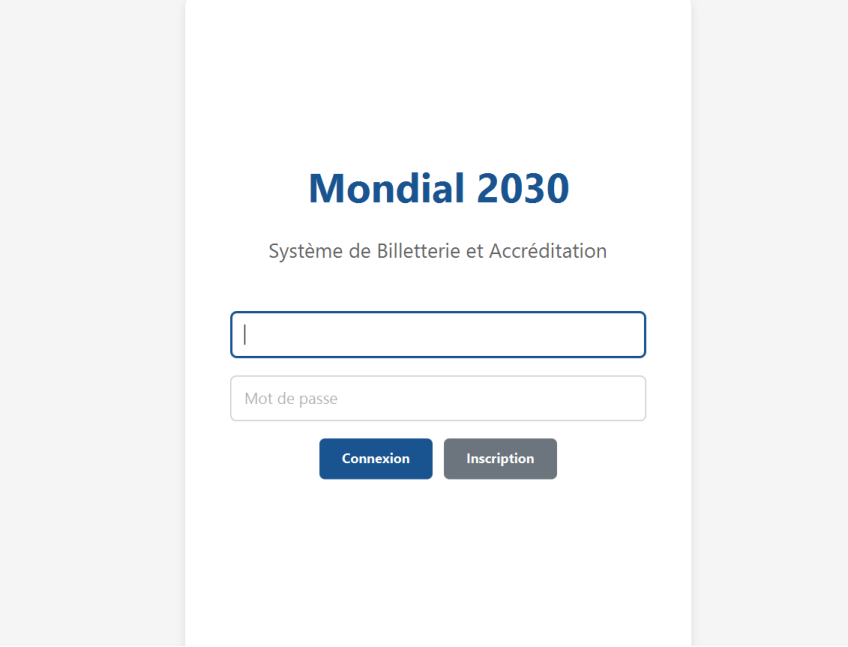

FIGURE 8.1 – Écran de connexion

Description :
— Champs : Nom d'utilisateur et Mot de passe
— Bouton "Se connecter" pour l'authentification
— Lien "S'inscrire" pour créer un nouveau compte
— Messages d'erreur en cas d'échec d'authentification

8.2 Écran d'Inscription

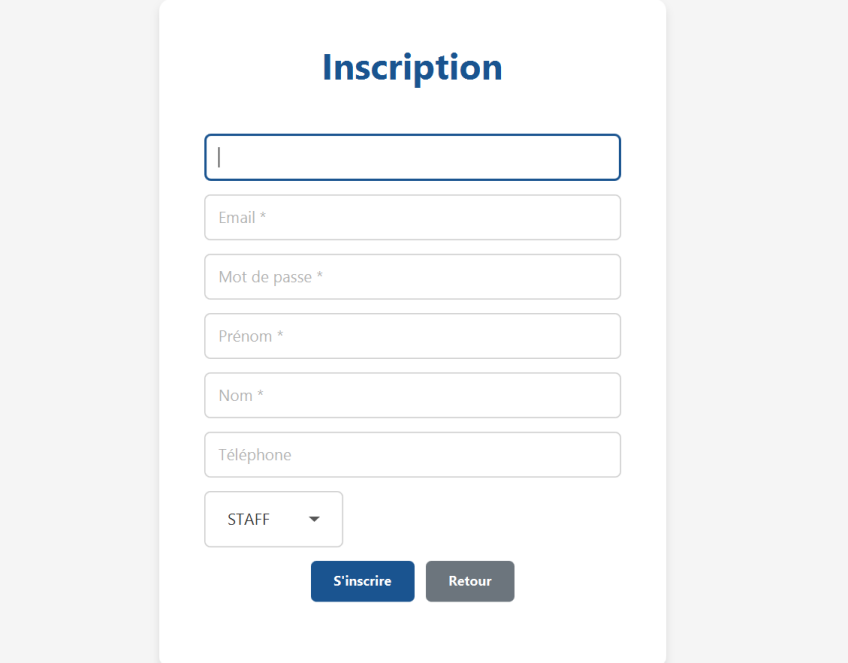

FIGURE 8.2 – Écran d'inscription

Description :
— Formulaire complet : Prénom, Nom, Email, Téléphone, Username, Password
— Choix du rôle : STAFF ou SUPPORTER
— Validation des champs obligatoires
— Vérification de l'unicité de l'email et du username
— Bouton "S'inscrire" pour créer le compte

8.3 Tableau de bord

FIGURE 8.3 – Tableau de bord personnalisé selon le rôle

Description :
- Message de bienvenue avec le nom complet de l'utilisateur
- Affichage du rôle
- Boutons de navigation selon le rôle :
  - **Tous :** Mes Tickets
  - **STAFF :** Scan QR Code, Gestion Tickets
  - **SUPPORTER :** Scan QR Code, Biométrie
- Bouton de déconnexion

8.4 Écran Mes Tickets

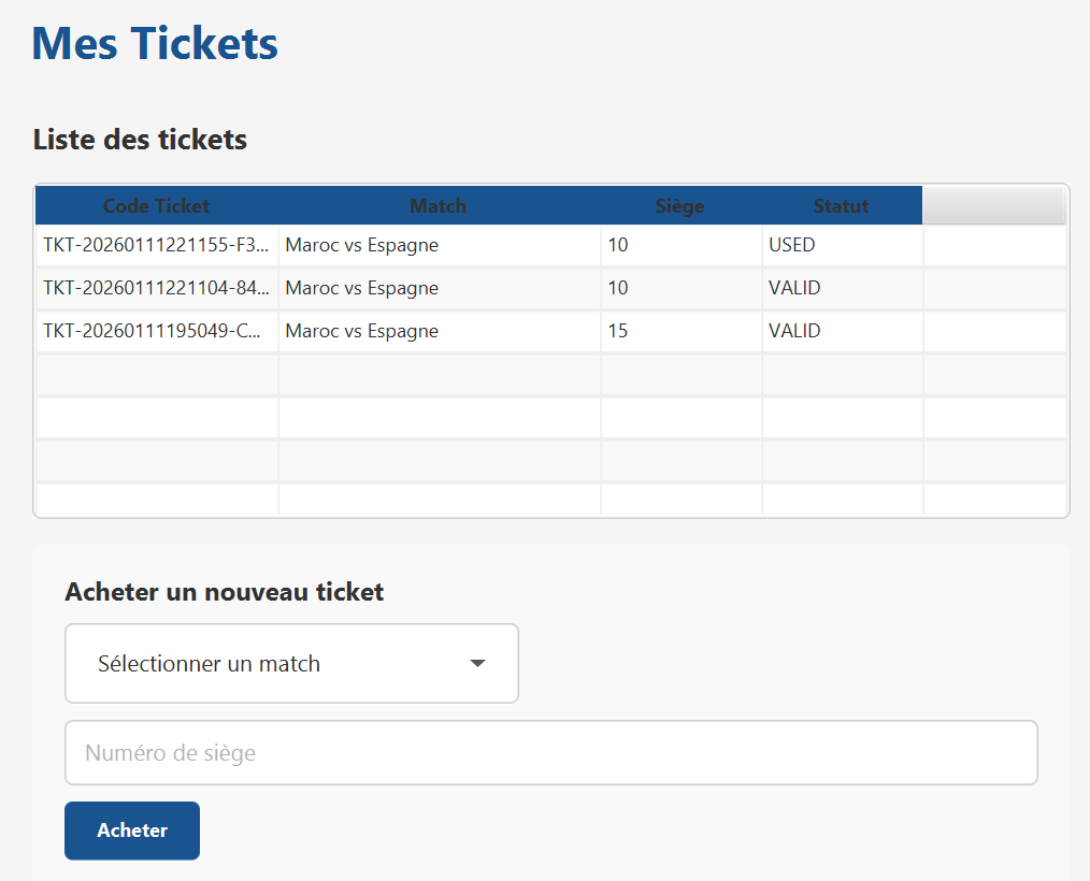

FIGURE 8.4 – Écran de gestion des tickets

Description :
- Tableau affichant tous les tickets de l'utilisateur
- Formulaire d'achat :
  - ComboBox avec tous les matchs disponibles (ajoutés par STAFF)
  - Champ pour le numéro de siège
  - Bouton "Acheter"
- Zone d'affichage du QR Code du ticket sélectionné
- Détails du ticket : code, match, siège, statut, date d'achat

8.5 Écran Biométrie

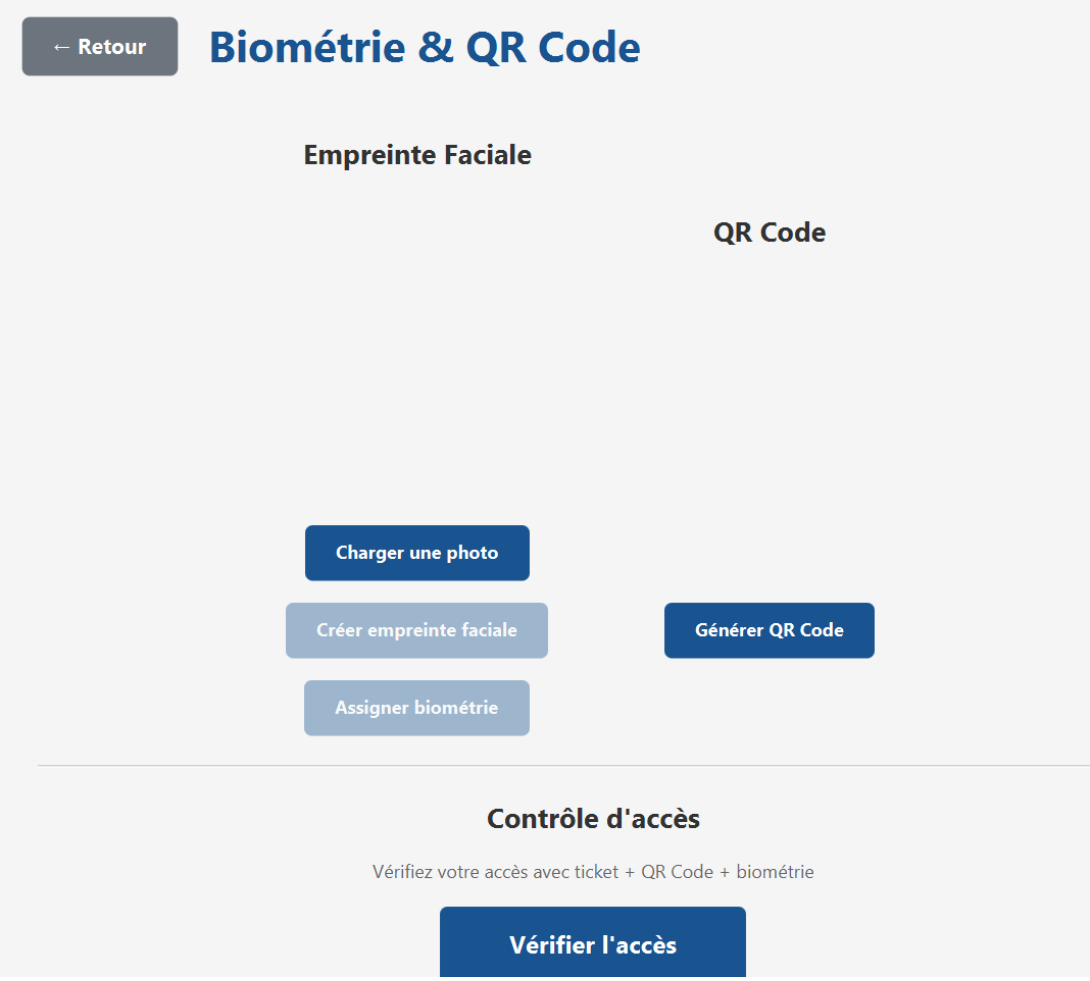

FIGURE 8.5 – Écran de gestion biométrique

Description :
- Section "Empreinte Faciale" :
  - Zone de prévisualisation de la photo
  - Bouton "Charger une photo"
  - Bouton "Créer empreinte faciale"
  - Bouton "Assigner biométrie"
- Section "QR Code" :
  - Zone d'affichage du QR Code personnel
  - Bouton "Générer QR Code"
- Section "Contrôle d'accès" :
  - Bouton "Vérifier l'accès"
  - Affichage du résultat de vérification
- Bouton retour (haut et bas)

8.6 Dialogue Ticket Validé

FIGURE 8.6 – Dialogue affichant le ticket validé avec photo et QR Code

Description :
— Affichage des informations du ticket : code, match, siège, utilisateur
— Photo du supporter
— QR Code du ticket
— Bouton "Enregistrer" pour sauvegarder en PNG
— Bouton "OK" pour fermer

8.7 Écran Scan QR Code

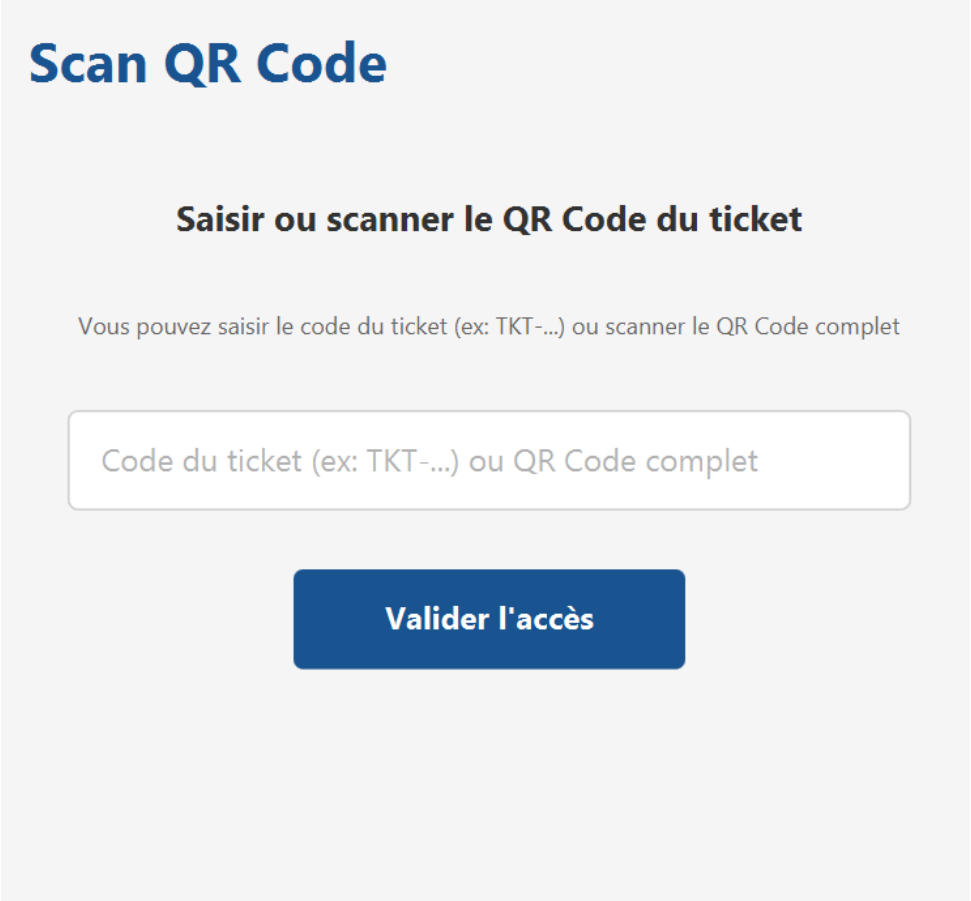

FIGURE 8.7 – Écran de scan QR Code

Description :
— Champ de saisie pour le QR Code
— Accepte le format complet ou juste le code ticket
— Indication claire pour l'utilisateur
— Bouton de validation
— Affichage du résultat (vert = autorisé, rouge = refusé)
— Messages d'erreur détaillés

8.8 Écran Gestion Tickets & Annonces (STAFF)

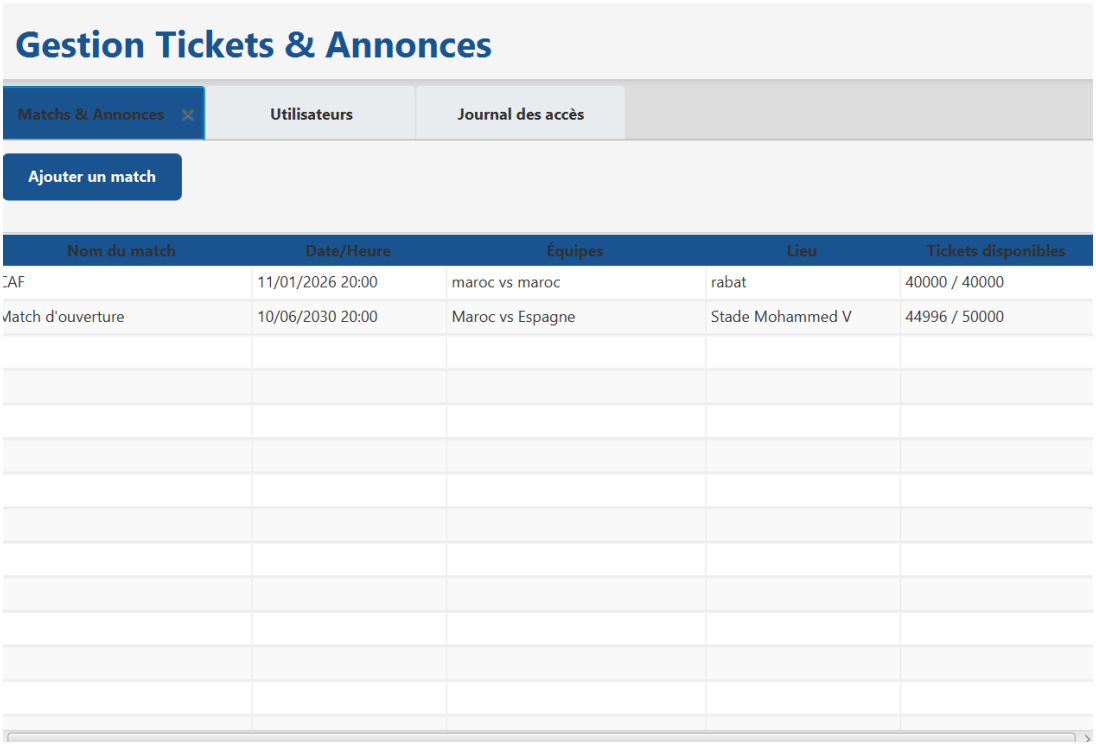

FIGURE 8.8 – Écran d'administration pour STAFF

Description :

- Onglet "Matchs & Annonces" :
  - Bouton "Ajouter un match"
  - Tableau des matchs avec toutes les informations
- Onglet "Utilisateurs" :
  - Tableau de tous les utilisateurs avec leurs informations
- Onglet "Journal des accès" :
  - Tableau de tous les logs d'accès
  - Informations : utilisateur, type, résultat, date/heure

Chapitre 9

Scénarios de Test

9.1 Tests nominaux (Cas de succès)

9.1.1 Connexion réussie

1. Ouvrir l'application
2. Entrer un username valide (ex : `supporter1`)
3. Entrer le mot de passe correct (`password123`)
4. Cliquer sur "Se connecter"
5. **Résultat attendu :** Redirection vers le dashboard avec message de bienvenue

9.1.2 Achat de ticket

1. Se connecter en tant que SUPPORTER
2. Aller dans "Mes Tickets"
3. Sélectionner un match dans le ComboBox
4. Saisir un numéro de siège (ex : "A-101")
5. Cliquer sur "Acheter"
6. **Résultat attendu :**
   — Message de succès affiché
   — Nouveau ticket ajouté dans le tableau
   — QR Code généré automatiquement
   — Tickets disponibles du match décrémentés

9.1.3 Création d'un match (STAFF)

1. Se connecter en tant que STAFF
2. Aller dans "Gestion Tickets"
3. Cliquer sur "Ajouter un match"
4. Remplir le formulaire :
   — Nom : "Quart de finale"
   — Date : 20/06/2030
   — Heure : 18 :00

- Lieu : "Stade Mohammed V"
- Équipe A : "Maroc"
- Équipe B : "France"
- Capacité : 60000
- Prix : 800.00

5. Cliquer sur "Ajouter"
6. **Résultat attendu :**
   - Message de succès
   - Match ajouté dans le tableau
   - Match visible dans la liste d'achat de tickets

9.1.4 Vérification d'accès biométrique

1. Se connecter en tant que SUPPORTER
2. Aller dans "Biométrie"
3. Charger une photo
4. Créer empreinte faciale
5. Assigner biométrie
6. Générer QR Code personnel
7. Avoir un ticket valide
8. Cliquer sur "Vérifier l'accès"
9. **Résultat attendu :**
   - Dialogue s'affiche avec "ACCÈS AUTORISÉ"
   - Ticket validé affiché avec photo et QR Code
   - Possibilité d'enregistrer le ticket

9.1.5 Scan QR Code (STAFF)

1. Se connecter en tant que STAFF
2. Aller dans "Scan QR Code"
3. Saisir un QR Code valide (format complet ou code seul)
4. Cliquer sur "Valider l'accès"
5. **Résultat attendu :**
   - Message "ACCÈS AUTORISÉ" en vert
   - Ticket marqué comme "USED"
   - Entrée ajoutée dans le journal des accès

9.2 Tests d'erreurs (Cas limites)

9.2.1 Connexion avec mot de passe incorrect

1. Ouvrir l'application
2. Entrer un username valide
3. Entrer un mot de passe incorrect

4. Cliquer sur "Se connecter"
5. **Résultat attendu :**
   — Message d'erreur : "Nom d'utilisateur ou mot de passe incorrect"
   — Reste sur l'écran de connexion
   — Logs enregistrés pour sécurité

9.2.2 Achat de ticket sans sélection de match

1. Se connecter en tant que SUPPORTER
2. Aller dans "Mes Tickets"
3. Ne pas sélectionner de match
4. Saisir un numéro de siège
5. Cliquer sur "Acheter"
6. **Résultat attendu :**
   — Message d'alerte : "Veuillez sélectionner un match"
   — Achat non effectué

9.2.3 Achat de ticket pour un match sans disponibilité

1. Se connecter en tant que SUPPORTER
2. Sélectionner un match avec `available_tickets = 0`
3. Essayer d'acheter un ticket
4. **Résultat attendu :**
   — Message d'erreur : "Il n'y a plus de tickets disponibles pour ce match"
   — Achat refusé

9.2.4 Scan d'un QR Code invalide

1. Se connecter en tant que STAFF
2. Aller dans "Scan QR Code"
3. Saisir un code inexistant ou déjà utilisé
4. Cliquer sur "Valider"
5. **Résultat attendu :**
   — Message "ACCÈS REFUSÉ" en rouge
   — Raison du refus affichée (ex : "Ticket non trouvé", "Ticket déjà utilisé")
   — Entrée dans le journal avec statut DENIED

9.2.5 Vérification d'accès sans biométrie complète

1. Se connecter en tant que SUPPORTER
2. Aller dans "Biométrie"
3. Ne pas charger de photo ou ne pas assigner biométrie
4. Cliquer sur "Vérifier l'accès"
5. **Résultat attendu :**
   — Message d'avertissement avec statut de chaque vérification
   — Résultat final : "ACCÈS REFUSÉ" si une condition manque
   — Indication claire de ce qui manque

9.2.6 Création de match avec données invalides

1. Se connecter en tant que STAFF
2. Aller dans "Gestion Tickets" → "Ajouter un match"
3. Laisser des champs obligatoires vides
4. Ou entrer une date dans le passé
5. Cliquer sur "Ajouter"
6. **Résultat attendu :**
   — Message d'erreur de validation
   — Match non créé
   — Formulaire reste ouvert pour correction

Chapitre 10

Conclusion et Perspectives

10.1 Bilan technique

10.1.1 Respect du cahier des charges

Le système développé répond aux objectifs fixés dans le cahier des charges :

- **Gestion des utilisateurs :** Inscription, connexion avec deux rôles (STAFF et SUPPORTER)
- **Gestion des tickets :** Achat, consultation, génération automatique de QR Codes
- **Gestion des matchs :** Création et consultation par le STAFF
- **Biométrie :** Upload d'image, création d'empreinte, assignation
- **Contrôle d'accès :** Vérification combinée (ticket + QR Code + biométrie)
- **Scan QR Code :** Validation des tickets par le STAFF et scan personnel par SUPPORTER
- **Journal des accès :** Historique complet de tous les accès
- **Sécurité :** Hashage BCrypt, PreparedStatement, contrôle d'accès par rôles
- **Interface moderne :** JavaFX avec navigation intuitive

10.1.2 Technologies maîtrisées

- **Java 21** : Langage orienté objet, gestion des exceptions, collections
- **JavaFX 21** : Interface graphique moderne, FXML, événements
- **MySQL 8.0** : Modélisation de base de données, requêtes SQL, jointures
- **JDBC** : Connexion à la base de données, PreparedStatement, gestion des transactions
- **Maven** : Gestion des dépendances, compilation, packaging
- **BCrypt** : Sécurité des mots de passe
- **ZXing** : Génération et lecture de QR Codes
- **Architecture MVC** : Séparation des responsabilités
- **Design Patterns** : Singleton, DAO, MVC

10.2 Bilan personnel

10.2.1 Compétences acquises

— **Développement Java avancé :** Maîtrise des concepts orientés objet, gestion des exceptions, collections Java
— **Architecture logicielle :** Compréhension et application du pattern MVC, organisation en packages
— **Base de données :** Conception de schéma relationnel, écriture de requêtes SQL complexes, optimisation
— **JDBC :** Gestion des connexions, utilisation de PreparedStatement, gestion des ResultSet
— **JavaFX :** Création d'interfaces modernes, gestion des événements, binding de propriétés
— **Maven :** Gestion des dépendances, configuration de build, résolution de conflits
— **Sécurité :** Hashage des mots de passe, prévention des injections SQL, contrôle d'accès
— **Design Patterns :** Application pratique de patterns (Singleton, DAO)
— **Débogage :** Résolution de problèmes complexes (versions Java, ResultSet, etc.)
— **Gestion de projet :** Organisation du code, documentation, tests

10.2.2 Difficultés rencontrées et solutions

1. **Problème :** Incompatibilité de version Java
   — *Description :* Classes compilées avec Java 25 alors que le runtime utilisait Java 21
   — *Solution :* Configuration explicite de Maven pour compiler avec Java 21 et utilisation de `<release>21</release>`
2. **Problème :** ResultSet fermé avant extraction complète des données
   — *Description :* Erreur "Operation not allowed after ResultSet closed" lors de l'appel de méthodes DAO imbriquées
   — *Solution :* Extraction de toutes les données primitives dans une liste temporaire avant les appels aux autres DAO
3. **Problème :** Hashs BCrypt non valides lors de la connexion
   — *Description :* Les hashs statiques dans seed_data.sql ne fonctionnaient pas car BCrypt génère des hashs uniques
   — *Solution :* Création d'un utilitaire `FixAllPasswords` pour générer dynamiquement des hashs valides
4. **Problème :** Erreurs de contraintes de clés étrangères
   — *Description :* IDs hardcodés dans seed_data.sql ne correspondaient pas aux IDs auto-incrémentés
   — *Solution :* Utilisation de sous-requêtes SQL dynamiques : `SELECT id FROM users WHERE username = '...'`
5. **Problème :** Erreurs de parsing FXML
   — *Description :* Caractères spéciaux (&) non échappés dans les fichiers FXML
   — *Solution :* Remplacement de `&` par `&` dans les labels XML
6. **Problème :** Types JavaFX non reconnus dans FXML

— *Description :* `ImageView` non trouvé dans tickets.fxml
— *Solution :* Ajout de l'import `<?import javafx.scene.image.ImageView?>`

10.3 Perspectives et améliorations futures

10.3.1 Améliorations fonctionnelles

1. **Intégration webcam**
   — Scanner les QR Codes directement via une webcam
   — Capture automatique de photos pour la biométrie
2. **Reconnaissance faciale complète**
   — Intégration d'une bibliothèque de reconnaissance faciale (OpenCV, Face Recognition)
   — Comparaison faciale en temps réel lors du contrôle d'accès
   — Détection de tentatives de fraude
3. **Export PDF des tickets**
   — Génération de tickets au format PDF
   — Impression directe
   — Envoi par email
4. **Système de paiement**
   — Intégration d'une passerelle de paiement
   — Gestion des remboursements
   — Historique des transactions
5. **Gestion des annonces**
   — Création d'annonces pour les matchs
   — Notification des supporters
   — Calendrier des événements
6. **Réservation de sièges interactifs**
   — Plan de stade interactif
   — Sélection visuelle des sièges
   — Vérification de disponibilité en temps réel
7. **Statistiques et rapports**
   — Tableaux de bord analytiques
   — Graphiques de fréquentation
   — Rapports d'accès exportables

10.3.2 Améliorations techniques

1. **Version Web**
   — Développement d'une version web avec Spring Boot
   — Interface responsive
   — Accessible depuis navigateur
2. **API REST**
   — Exposition d'APIs REST pour intégration mobile
   — Authentification JWT
   — Documentation avec Swagger

3. **Application Mobile**
   — Application Android/iOS pour les supporters
   — QR Codes dans le portefeuille mobile
   — Notifications push
4. **Base de données distribuée**
   — Réplication pour haute disponibilité
   — Sauvegarde automatique
   — Restauration en cas de panne
5. **Sécurité avancée**
   — Authentification à deux facteurs (2FA)
   — Chiffrement des données sensibles
   — Audit logs détaillés
   — Détection d'intrusions
6. **Tests automatisés**
   — Tests unitaires avec JUnit
   — Tests d'intégration
   — Tests d'interface avec TestFX
7. **Déploiement en production**
   — Dockerisation de l'application
   — CI/CD avec GitHub Actions
   — Monitoring et logging avancés

10.3.3 Évolutions métier

1. **Gestion multi-événements**
   — Support de plusieurs événements simultanés
   — Séparation des données par événement
2. **Système de fidélité**
   — Points de fidélité pour les supporters
   — Réductions pour les clients réguliers
3. **Gestion VIP**
   — Zones VIP avec accès spéciaux
   — Services premium
4. **Intégration avec réseaux sociaux**
   — Partage de tickets
   — Authentification sociale

10.4 Conclusion

Ce projet a permis de développer un système complet de billetterie et d'accréditation biométrique pour le Mondial 2030, répondant aux besoins fonctionnels et techniques spécifiés. L'application intègre des fonctionnalités avancées de sécurité, de biométrie et de gestion d'accès, tout en offrant une interface utilisateur moderne et intuitive.

L'utilisation d'une architecture MVC propre, de design patterns appropriés, et de bonnes pratiques de développement garantit la maintenabilité et l'évolutivité du système.

Le projet démontre une bonne maîtrise des technologies Java, JavaFX, MySQL et des concepts de développement logiciel.

Les perspectives d'amélioration ouvrent la voie à une application encore plus complète, pouvant évoluer vers une solution web et mobile pour une accessibilité maximale.

"Un code propre est comme une œuvre d'art : fonctionnel, élégant et compréhensible."

Annexe A

Annexes

A.1 Annexe A : Schéma de la base de données complet

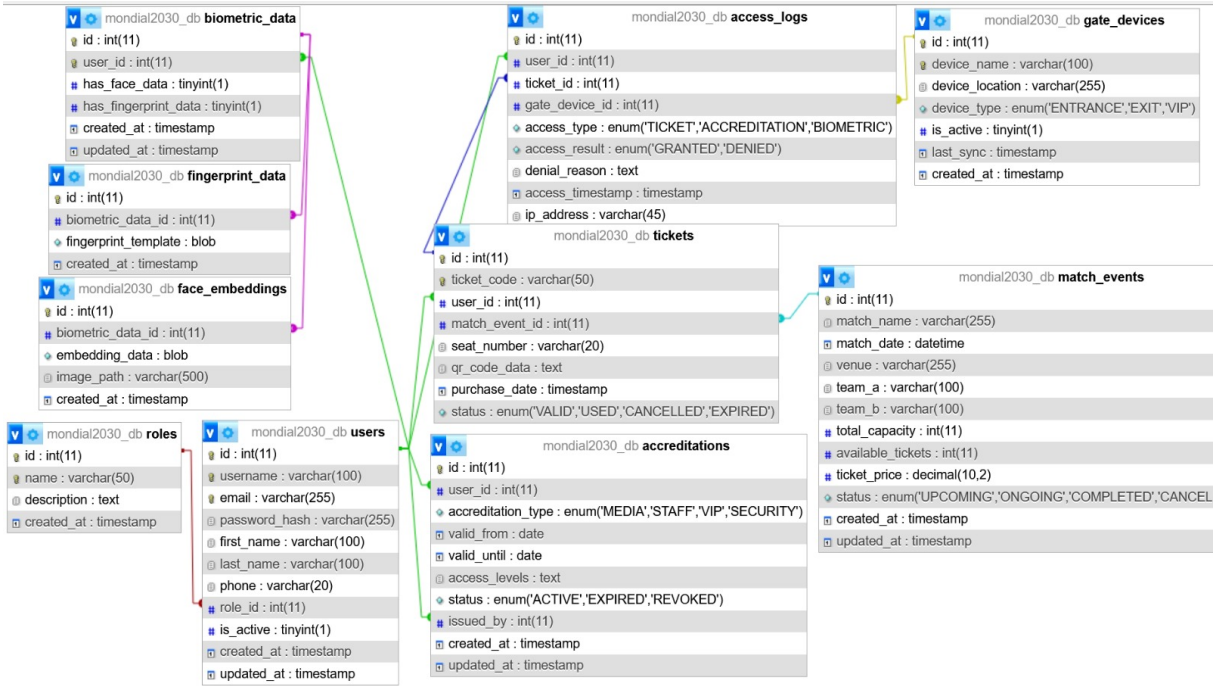

FIGURE A.1 – Schéma complet de la base de données

A.2 Annexe B : Extraits de code supplémentaires

A.2.1 AuthenticationService.java

Listing A.1 – Service d'authentification complet

```
1 public class AuthenticationService {
2     private static User currentUser;
3
```

```
    public User login(String username, String password)
            throws AuthenticationException {
        if (username == null || username.trim().isEmpty()) {
            throw new AuthenticationException(
                "Le nom d'utilisateur est requis");
        }
        if (password == null || password.isEmpty()) {
            throw new AuthenticationException(
                "Le mot de passe est requis");
        }

        User user = userDAO.findByUsername(username);
        if (user == null) {
            logger.warn("Utilisateur inexistant: {}", username);
            throw new AuthenticationException(
                "Nom d'utilisateur ou mot de passe incorrect");
        }

        if (!user.isActive()) {
            throw new AuthenticationException(
                "Ce compte est d sactiv ");
        }

        if (!PasswordHasher.verifyPassword(
                password, user.getPasswordHash())) {
            logger.warn("Mot de passe incorrect pour: {}",
               username);
            throw new AuthenticationException(
                "Nom d'utilisateur ou mot de passe incorrect");
        }

        currentUser = user;
        logger.info("Utilisateur connect : {}", username);
        return user;
    }
}
```

A.3 Annexe C : Scripts SQL

A.3.1 Création de la base de données

Listing A.2 – Script de création de la base

```
CREATE DATABASE IF NOT EXISTS mondial2030_db
CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
USE mondial2030_db;

-- Insertion des r les
INSERT INTO roles (name, description) VALUES
('STAFF', 'Personnel autoris  pour le contr le d''acc s ,
```

```
8             gestion des tickets et annonces'),
9 ('SUPPORTER', 'Supporteur avec acc s aux matchs,
10                g n ration QR Codes et biom trie');
```

Bibliographie

[1] Oracle Corporation, *JavaFX Documentation*, https://openjfx.io/

[2] Oracle Corporation, *MySQL 8.0 Reference Manual*, https://dev.mysql.com/doc/refman/8.0/en/

[3] Apache Software Foundation, *Apache Maven Documentation*, https://maven.apache.org/guides/

[4] *BCrypt Java Implementation*, https://github.com/jeremyh/jBCrypt

[5] Google, *ZXing ("Zebra Crossing") barcode scanning library*, https://github.com/zxing/zxing

[6] OMG, *Unified Modeling Language (UML)*, https://www.omg.org/spec/UML/

[7] Oracle Corporation, *JDBC Documentation*, https://docs.oracle.com/javase/tutorial/jdbc/

[8] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley.

[9] Oracle Corporation, *Java Platform, Standard Edition Documentation*, https://docs.oracle.com/javase/