ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

École Marocaine des Sciences de l'Ingénieur

4ème Année Ingénierie Informatique et Réseaux

Rapport de Projet

Java Avancé

Plateforme Intelligente de Gestion des Tickets

MONDIAL 2030

Réalisé par :

HAITAM SALAH-EDDINE

Encadré par :
Pr. Abderrahim Larhlimi

Année Universitaire : 2025-2026

Remerciements

Nous tenons à exprimer notre profonde gratitude à toutes les personnes qui ont contribué à la réalisation de ce projet.

Nous remercions tout d'abord notre encadrant, **Pr. Abderrahim Larhlimi**, pour son accompagnement, ses conseils précieux et sa disponibilité tout au long de ce projet.

Nous adressons également nos remerciements à l'administration de l'**EMSI** pour nous avoir fourni les ressources nécessaires à la réalisation de ce travail.

Enfin, nous remercions nos camarades de promotion pour leur soutien et les échanges enrichissants que nous avons pu avoir.

L'équipe du projet Mondial 2030

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction Générale

1.1 Contexte du Projet

Dans le cadre de notre formation en 4ème année Ingénierie Informatique et Réseaux à l'EMSI, nous sommes amenés à réaliser un projet de développement Java avancé mettant en œuvre les concepts de programmation orientée objet, les design patterns et les frameworks modernes.

Le Maroc, l'Espagne et le Portugal ont été désignés pour organiser conjointement la **Coupe du Monde de Football 2030**. Cet événement d'envergure mondiale nécessite une infrastructure technologique robuste pour gérer la billetterie des matchs.

1.2 Problématique

La gestion manuelle ou semi-automatisée des tickets pour un événement de cette ampleur pose plusieurs défis :

- **Volume important** : Des millions de tickets à gérer
- **Sécurité** : Prévention de la fraude et des contrefaçons
- **Traçabilité** : Suivi des transactions et des transferts
- **Temps réel** : Gestion des flux de spectateurs
- **Accessibilité** : Interface intuitive pour tous les utilisateurs

1.3 Objectifs du Projet

Notre projet vise à développer une **Plateforme Intelligente de Gestion des Tickets** avec les fonctionnalités suivantes :

Objectifs Principaux

- ✓ Système d'authentification sécurisé (inscription/connexion)
- ✓ Gestion complète des matchs et équipes
- ✓ Achat, transfert et annulation de tickets
- ✓ Génération de QR codes uniques
- ✓ Tableau de bord administrateur avec statistiques
- ✓ Système d'alertes en temps réel
- ✓ Génération de rapports PDF
- ✓ Suivi des flux de spectateurs

Chapitre 2

Analyse et Conception

2.1 Spécification des Besoins

2.1.1 Besoins Fonctionnels

TABLE 2.1 – Besoins fonctionnels - Spectateurs

ID	Description
BF01	Le système doit permettre l'inscription d'un nouveau spectateur
BF02	Le système doit permettre l'authentification sécurisée
BF03	Le système doit afficher la liste des matchs disponibles
BF04	Le système doit permettre l'achat de tickets avec choix de catégorie
BF05	Le système doit générer un QR code unique pour chaque ticket
BF06	Le système doit permettre le transfert de tickets
BF07	Le système doit permettre l'annulation avec remboursement
BF08	Le système doit afficher l'historique des transactions

TABLE 2.2 – Besoins fonctionnels - Administrateurs

ID	Description
BF09	Le système doit fournir un dashboard avec statistiques
BF10	Le système doit permettre la gestion CRUD des matchs
BF11	Le système doit permettre la gestion des utilisateurs
BF12	Le système doit afficher toutes les ventes de tickets
BF13	Le système doit gérer les alertes (création/résolution)
BF14	Le système doit générer et exporter des rapports PDF
BF15	Le système doit visualiser les flux de spectateurs

2.1.2 Besoins Non-Fonctionnels

TABLE 2.3 – Besoins non-fonctionnels

Catégorie	Exigence
Sécurité	Mots de passe hachés avec BCrypt
Performance	Temps de réponse $<$ 2 secondes
Ergonomie	Interface intuitive et moderne
Maintenabilité	Architecture en couches, code documenté
Portabilité	Application desktop multi-plateforme
Fiabilité	Gestion des erreurs et transactions

2.2 Conception UML

2.2.1 Diagramme de Classes

Le diagramme de classes complet du projet comprend 21 entités. Voici les classes principales :

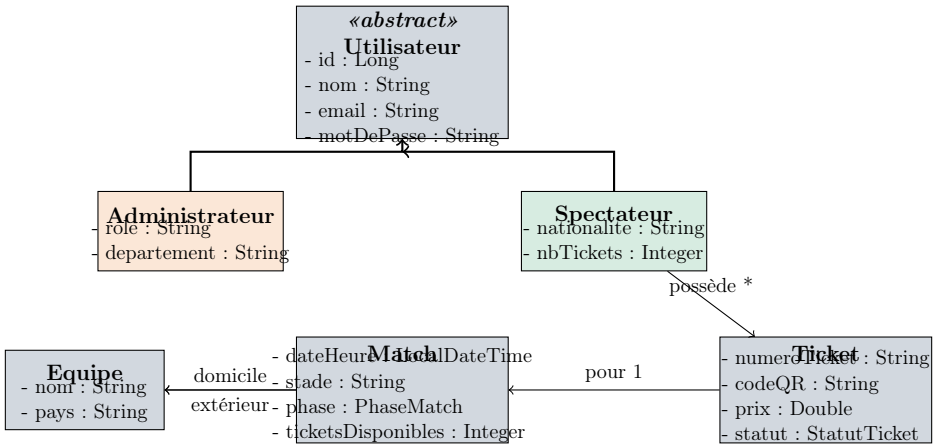

FIGURE 2.1 – Diagramme de classes simplifié

2.3 Conception de la Base de Données

2.3.1 Modèle Logique de Données

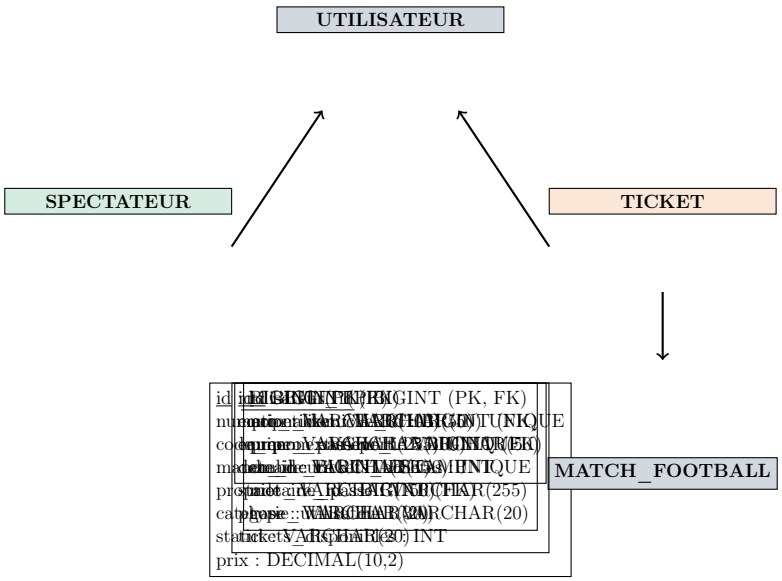

FIGURE 2.2 – Modèle logique de données (extrait)

2.3.2 Dictionnaire de Données

TABLE 2.4 – Dictionnaire de données - Table TICKET

Champ	Type	Taille	Contrainte
id	BIGINT	-	PK, AUTO_INCREMENT
numero_ticket	VARCHAR	50	UNIQUE, NOT NULL
code_qr	VARCHAR	255	UNIQUE
match_id	BIGINT	-	FK, NOT NULL
proprietaire_id	BIGINT	-	FK
siege_id	BIGINT	-	FK
categorie	VARCHAR	20	ENUM
statut	VARCHAR	20	ENUM
prix	DECIMAL	10,2	-
date_achat	TIMESTAMP	-	-
transferable	BOOLEAN	-	DEFAULT TRUE
nombre_transferts	INT	-	DEFAULT 0

Chapitre 3

Environnement Technique

3.1 Stack Technologique

TABLE 3.1 – Technologies utilisées

Technologie	Version	Usage
Java	17 (LTS)	Langage de programmation principal
JavaFX	21.0.1	Framework d'interface graphique
Maven	3.x	Gestion des dépendances et build
Hibernate	6.4.1.Final	ORM (Object-Relational Mapping)
SQLite	3.44.1.0	Base de données embarquée
BCrypt	0.4	Hachage sécurisé des mots de passe
ZXing	3.5.2	Génération de QR codes
iText	8.0.2	Génération de documents PDF
SLF4J	2.0.9	Framework de logging
CSSFX	11.5.1	Rechargement CSS en temps réel

3.2 Configuration Maven

Extrait du fichier pom.xml

```
<properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <javafx.version>21.0.1</javafx.version>
    <hibernate.version>6.4.1.Final</hibernate.version>
</properties>

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

    <!-- Hibernate Core -->
    <dependency>
        <groupId>org.hibernate.orm</groupId>
        <artifactId>hibernate-core</artifactId>
```

```
        <version>${hibernate.version}</version>
    </dependency>

    <!-- BCrypt -->
    <dependency>
        <groupId>org.mindrot</groupId>
        <artifactId>jbcrypt</artifactId>
        <version>0.4</version>
    </dependency>
</dependencies>
```

3.3 Configuration Hibernate

```
<hibernate-configuration>
    <session-factory>
        <!-- SQLite Configuration -->
        <property name="hibernate.connection.driver_class">
            org.sqlite.JDBC
        </property>
        <property name="hibernate.connection.url">
            jdbc:sqlite:mondial2030.db
        </property>
        <property name="hibernate.dialect">
            org.hibernate.community.dialect.SQLiteDialect
        </property>
        <property
           name="hibernate.hbm2ddl.auto">update</property>
        <property name="hibernate.show_sql">true</property>

        <!-- Entity Mappings -->
        <mapping class="com.mondial2030.entity.Utilisateur"/>
        <mapping class="com.mondial2030.entity.Ticket"/>
        <mapping class="com.mondial2030.entity.Match"/>
        <!-- ... autres mappings -->
    </session-factory>
</hibernate-configuration>
```

Listing 3.1 – Configuration Hibernate (hibernate.cfg.xml)

3.4 Outils de Développement

- **IDE** : IntelliJ IDEA / VS Code avec extensions Java
- **Modélisation UML** : StarUML, Draw.io
- **Base de données** : DB Browser for SQLite
- **Versioning** : Git
- **Documentation** : LaTeX (Overleaf)

Chapitre 4

Architecture et Implémentation

4.1 Architecture Logicielle

Le projet suit une **architecture en couches** (Layer Architecture) combinée avec le pattern **MVC** (Model-View-Controller).

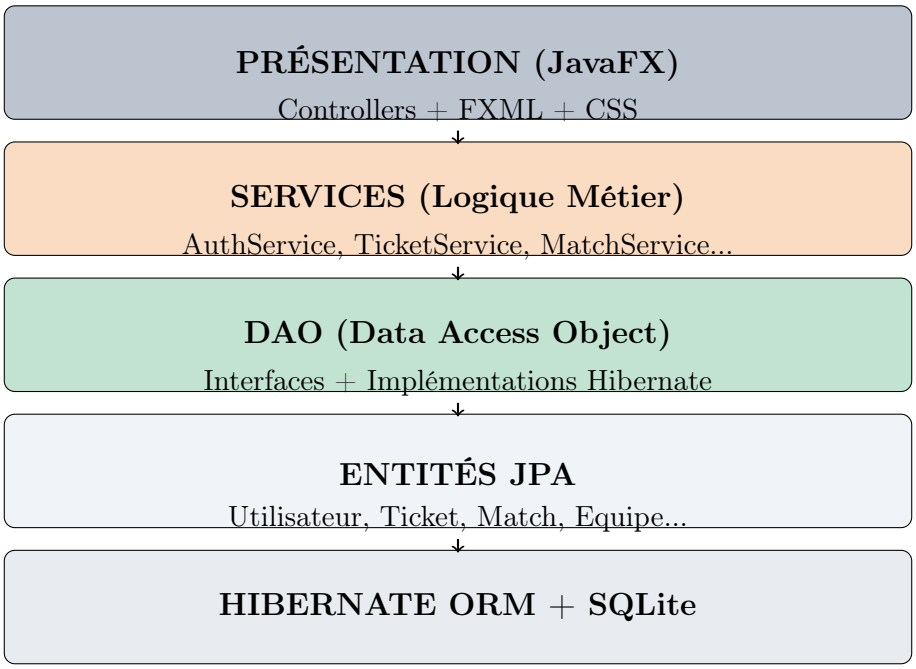

FIGURE 4.1 – Architecture en couches du projet

4.1.1 Organisation des Packages

```
com.mondial2030/
|-- MainApp.java                # Point d'entree JavaFX
|-- controller/                 # Controleurs MVC
|   |-- BaseController.java
|   |-- LoginController.java
|   |-- AdminDashboardController.java
|   +-- SpectateurDashboardController.java
|-- dao/
|   |-- interfaces/             # 14 interfaces DAO
|   +-- impl/                   # Implementations
|-- entity/                     # 21 entites JPA
|-- service/                    # 6 services metier
+-- util/                       # Utilitaires
```

Listing 4.1 – Structure des packages

4.2 Design Patterns Utilisés

4.2.1 Pattern Singleton

Justification

Le pattern Singleton est utilisé pour les services métier afin de garantir une **instance unique** et un **point d'accès global**. Cela permet d'économiser les ressources et d'assurer la cohérence des données.

```
1  public class AuthenticationService {
2      private static AuthenticationService instance;
3      private Utilisateur utilisateurConnecte;
4
5      // Constructeur prive
6      private AuthenticationService() {
7          this.adminDAO = new AdministrateurDAOImpl();
8          this.spectateurDAO = new SpectateurDAOImpl();
9      }
10
11     // Acces synchronise a l'instance
12     public static synchronized AuthenticationService
           getInstance() {
13         if (instance == null) {
14             instance = new AuthenticationService();
15         }
16         return instance;
17     }
18
19     public Optional<Utilisateur> authentifier(String email,
           String mdp) {
20         // Logique d'authentification
21     }
22 }
```

Listing 4.2 – Implémentation du Singleton - AuthenticationService

4.2.2 Pattern DAO (Data Access Object)

Justification

Le pattern DAO permet d'**isoler la logique d'accès aux données** du reste de l'application. Cela facilite la maintenance, les tests et un éventuel changement de base de données.

```java
public interface GenericDAO<T, ID> {
    void save(T entity);
    void update(T entity);
    void delete(ID id);
    Optional<T> findById(ID id);
    List<T> findAll();
    long count();
    boolean existsById(ID id);
}
```

Listing 4.3 – Interface générique DAO

```java
public abstract class GenericDAOImpl<T, ID extends Serializable>
        implements GenericDAO<T, ID> {

    protected final Class<T> entityClass;

    protected Session getSession() {
        return HibernateUtil.getSessionFactory().openSession();
    }

    @Override
    public void save(T entity) {
        Transaction tx = null;
        try (Session session = getSession()) {
            tx = session.beginTransaction();
            session.persist(entity);
            tx.commit();
        } catch (Exception e) {
            if (tx != null) tx.rollback();
            throw new RuntimeException(e);
        }
    }

    @Override
    public Optional<T> findById(ID id) {
        try (Session session = getSession()) {
            T entity = session.get(entityClass, id);
            return Optional.ofNullable(entity);
        }
    }
}
```

Listing 4.4 – Implémentation générique avec Hibernate

4.2.3 Pattern MVC

TABLE 4.1 – Composants MVC du projet

Composant	Implémentation	Description
Model	Entités JPA	Classes métier persistantes
View	FXML + CSS	Interfaces déclaratives
Controller	*Controller.java	Logique de présentation

4.3 Extraits de Code Clés

4.3.1 Achat de Ticket avec Transaction

```
public Optional<Ticket> acheterTicket(Spectateur spectateur,
        Match match, CategorieTicket categorie, String
           methodePaiement) {
    try {
        // 1. Verifier la disponibilite
        Optional<Match> matchFrais =
           matchDAO.findById(match.getId());
        if (matchFrais.isEmpty() ||
            matchFrais.get().getTicketsDisponibles() <= 0) {
            return Optional.empty();
        }

        // 2. Calculer le prix selon la categorie
        double prix = calculerPrix(matchFrais.get(), categorie);

        // 3. Creer le ticket avec QR code unique
        Ticket ticket = new Ticket(matchFrais.get(), spectateur,
                                   categorie, prix);
        ticket.setDateAchat(LocalDateTime.now());
        ticket.setStatut(StatutTicket.RESERVE);

        // 4. Creer et valider la transaction
        Transaction transaction = new Transaction(spectateur,
           ticket,
                                     TypeTransaction.ACHAT,
                                        prix);

        if (simulerPaiement(transaction)) {
            ticket.valider();
            ticketDAO.save(ticket);
            transaction.valider();
            transactionDAO.save(transaction);
            matchDAO.decrementerTicketsDisponibles(match.getId());
            return Optional.of(ticket);
        }

```

```
33          return Optional.empty();
34      } catch (Exception e) {
35          logger.error("Erreur achat ticket", e);
36          return Optional.empty();
37      }
38  }
```

Listing 4.5 – Méthode d'achat de ticket

4.3.2 Authentification Sécurisée avec BCrypt

```
1  public Optional<Spectateur> authentifier(String email, String
       motDePasse) {
2      try (Session session = getSession()) {
3          String hql = "FROM Spectateur s WHERE s.email = :email
               AND s.actif = true";
4          Query<Spectateur> query = session.createQuery(hql,
               Spectateur.class);
5          query.setParameter("email", email);
6
7          Optional<Spectateur> spectateur =
               query.uniqueResultOptional();
8
9          // Verification du mot de passe avec BCrypt
10         if (spectateur.isPresent() &&
11             BCrypt.checkpw(motDePasse,
                   spectateur.get().getMotDePasse())) {
12             return spectateur;
13         }
14
15         return Optional.empty();
16     }
17 }
```

Listing 4.6 – Authentification avec vérification BCrypt

4.3.3 Génération de QR Code

```
1  public class QRCodeGenerator {
2
3      public static Image generateQRCode(String content, int
           width, int height) {
4          try {
5              QRCodeWriter writer = new QRCodeWriter();
6              BitMatrix matrix = writer.encode(content,
7                  BarcodeFormat.QR_CODE, width, height);
8
9              BufferedImage bufferedImage = MatrixToImageWriter
10                 .toBufferedImage(matrix);
11
```

```
12                return SwingFXUtils.toFXImage(bufferedImage, null);
13            } catch (WriterException e) {
14                throw new RuntimeException("Erreur generation QR",
                      e);
15            }
16        }
17    }
```

Listing 4.7 – Génération de QR Code avec ZXing

Chapitre 5

Interface Utilisateur et Tests

5.1 Présentation des Interfaces

5.1.1 Page de Connexion

FIGURE 5.1 – Maquette de la page de connexion

Caractéristiques :

— Design moderne avec gradient bleu
— Carte de connexion centrée avec ombrage
— Champs de saisie stylisés
— Switch entre connexion et inscription
— Messages d'erreur contextuels

5.1.2 Dashboard Administrateur

Le tableau de bord administrateur comprend :

— **Statistiques globales** : Nombre de matchs, tickets vendus, spectateurs, alertes
— **Graphiques** :
  — PieChart : Répartition des tickets par catégorie
  — BarChart : Ventes par match
  — LineChart : Tendance des ventes
— **Tableaux de gestion** : Matchs, Utilisateurs, Tickets, Alertes
— **Filtres et recherche** : ComboBox et TextField

5.1.3 Dashboard Spectateur

— Liste des matchs disponibles avec filtres
— Mes tickets avec QR codes
— Formulaire d'achat de ticket
— Historique des transactions
— Option de transfert de ticket

5.2 Styles CSS

```css
/* Conteneur principal avec gradient */
.login-container {
    -fx-background-color: linear-gradient(
        to bottom right, #0f2847, #1e3a5f, #2c5282);
}

/* Carte de login avec ombre */
.login-card {
    -fx-background-color: linear-gradient(
        to bottom, #ffffff, #fafbfc);
    -fx-background-radius: 20;
    -fx-effect: dropshadow(gaussian, rgba(0,0,0,0.25),
                           30, 0, 0, 10);
}

/* Bouton primaire avec animation */
.btn-primary {
    -fx-background-color: linear-gradient(
        to right, #ed8936, #f6ad55);
    -fx-text-fill: white;
    -fx-background-radius: 10;
    -fx-cursor: hand;
}

.btn-primary:hover {
    -fx-scale-x: 1.02;
    -fx-scale-y: 1.02;
}
```

Listing 5.1 – Extrait des styles CSS

5.3 Scénarios de Test

5.3.1 Tests Nominaux

TABLE 5.1 – Scénarios de test - Cas nominaux

ID	Action	Résultat attendu	Statut
TN01	Connexion admin valide	Accès au dashboard admin	✓
TN02	Inscription spectateur	Compte créé, redirection	✓
TN03	Achat d'un ticket	Ticket créé avec QR code	✓
TN04	Transfert de ticket	Nouveau propriétaire	✓
TN05	Génération rapport PDF	Fichier PDF créé	✓

5.3.2 Tests d'Erreurs

TABLE 5.2 – Scénarios de test - Cas d'erreur

ID	Action	Résultat attendu	Statut
TE01	Connexion mot de passe invalide	Message d'erreur affiché	✓
TE02	Email déjà existant	Erreur "Email déjà utilisé"	✓
TE03	Achat sans tickets dispo	Message "Complet"	✓
TE04	Transfert ticket non transférable	Erreur affichée	✓

Chapitre 6

Conclusion et Perspectives

6.1 Bilan Technique

Ce projet a permis de développer une **plateforme complète de gestion de tickets** respectant les exigences du cahier des charges :

Objectifs atteints

- ✓ Architecture en couches claire et maintenable
- ✓ Implémentation des design patterns (Singleton, DAO, MVC)
- ✓ Persistance des données avec Hibernate/JPA
- ✓ Interface utilisateur moderne avec JavaFX
- ✓ Sécurité des mots de passe avec BCrypt
- ✓ Génération de QR codes et rapports PDF

6.2 Bilan Personnel

Ce projet nous a permis d'acquérir et de renforcer plusieurs compétences :

— **Programmation Orientée Objet** : Héritage, polymorphisme, encapsulation
— **Design Patterns** : Singleton, DAO, Factory, MVC
— **Framework Hibernate** : Mapping ORM, sessions, transactions
— **JavaFX** : Interfaces modernes, FXML, CSS
— **Gestion de projet** : Maven, Git, documentation

6.3 Difficultés Rencontrées

Difficultés et solutions

— **Configuration Hibernate/SQLite** : Résolu en utilisant le dialect communautaire
— **Gestion des sessions Hibernate** : Pattern try-with-resources
— **Relations JPA complexes** : Documentation et tests itératifs
— **Styling JavaFX** : Utilisation de CSSFX pour le rechargement temps réel

6.4 Perspectives d'Amélioration

1. **Tests unitaires** : Ajouter JUnit 5 et Mockito
2. **Version Web** : Migration vers Spring Boot + Angular/React
3. **Application Mobile** : Version Android/iOS
4. **Microservices** : Découpage en services indépendants
5. **Cloud** : Déploiement AWS/Azure avec base de données distante
6. **Machine Learning** : Prédiction de la demande de tickets
7. **Internationalisation** : Support multilingue

Webographie / Bibliographie

— **Documentation Oracle Java** : https://docs.oracle.com/en/java/
— **Documentation JavaFX** : https://openjfx.io/
— **Documentation Hibernate** : https://hibernate.org/orm/documentation/
— **Maven Repository** : https://mvnrepository.com/
— **ZXing (QR Code)** : https://github.com/zxing/zxing
— **iText PDF** : https://itextpdf.com/
— **BCrypt** : https://www.mindrot.org/projects/jBCrypt/
— **Stack Overflow** : https://stackoverflow.com/
— **Baeldung Java Tutorials** : https://www.baeldung.com/

Annexe A

Annexe : Guide d'Installation

A.1 Prérequis

— JDK 17 ou supérieur
— Maven 3.6 ou supérieur
— IDE : IntelliJ IDEA ou VS Code

A.2 Installation et Exécution

```
# Cloner le projet
cd E:\javaprojet\ticketing-mondial-2030

# Compiler
mvn clean compile

# Lancer l'application
mvn javafx:run
```

A.3 Comptes de Test

Type	**Email**	**Mot de passe**
Administrateur	admin@mondial2030.com	admin123
Spectateur	spectateur@test.com	test123