# EMSI

École Marocaine des Sciences de l'Ingénieur

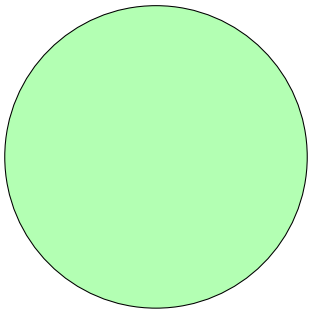
---

# Rapport de Projet Java Avancé

---

## World Cup 2030 Morocco
## Ticket Management System

Application de Gestion de Billetterie



**Morocco 2030**

**Module :**
Java Avancé / Programmation Orientée Objet

**Filière :**
4ème Année Ingénierie Informatique et Réseaux (4IIR)

**Réalisé par :**
Abderrahman

**Encadré par :**
Pr. Abderrahim LARHLIMI

**Année Universitaire : 2025-2026**

# Remerciements

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

Tout d'abord, nous remercions sincèrement notre encadrant,
textbfPr. Abderrahim LARHLIMI, pour ses conseils précieux, sa disponibilité et son accompagnement tout au long de ce projet. Ses orientations nous ont permis d'approfondir nos connaissances en Java avancé et en architecture logicielle.

Nous adressons également nos remerciements à l'administration de l'EMSI pour les moyens mis à notre disposition et pour la qualité de la formation dispensée.

Enfin, nous remercions nos camarades de promotion pour les échanges enrichissants et l'entraide qui ont marqué cette période de travail.

# Table des matières

# Chapitre 1

# Introduction Générale

## 1.1 Contexte du Projet

La Coupe du Monde de Football 2030 représente un événement sportif majeur qui sera co-organisé par le Maroc, l'Espagne et le Portugal. Cette compétition d'envergure mondiale nécessite une infrastructure technologique robuste pour gérer la billetterie, les réservations et l'expérience des spectateurs.

Dans le cadre de notre formation en Java Avancé à l'EMSI, nous avons développé une application de gestion de billetterie moderne et professionnelle, capable de répondre aux exigences d'un tel événement. Ce projet nous permet d'appliquer les concepts avancés de la programmation orientée objet, de l'ORM Hibernate, et des architectures en couches.

## 1.2 Problématique

La gestion de la billetterie pour un événement de l'ampleur de la Coupe du Monde pose plusieurs défis :

- **Volume de données :** Des millions de billets à gérer pour 48 équipes et 104 matchs
- **Concurrence d'accès :** Gestion simultanée de milliers de réservations
- **Intégrité des données :** Éviter la survente de billets (overbooking)
- **Performance :** Temps de réponse optimal malgré la charge
- **Sécurité :** Protection des données des utilisateurs et des transactions

## 1.3 Objectifs du Projet

Notre application vise à fournir les fonctionnalités suivantes :

1. **Gestion des utilisateurs :** Inscription, authentification, profils utilisateurs
2. **Catalogue des matchs :** Consultation des matchs par stade, date, phase de compétition
3. **Système de billetterie :** Recherche, réservation et achat de billets
4. **Gestion des réservations :** Suivi des commandes, confirmation, annulation
5. **Interface Web :** API REST pour l'intégration avec des applications front-end
6. **Interface Console :** Application interactive en ligne de commande

# Chapitre 2

# Analyse et Conception

## 2.1 Spécification des Besoins

### 2.1.1 Besoins Fonctionnels

| **Fonctionnalité** | **Description** |
|---|---|
| Inscription | Le système doit permettre à un utilisateur de créer un compte avec ses informations personnelles (nom, prénom, email, téléphone, pays) |
| Authentification | Le système doit permettre la connexion sécurisée via email et mot de passe |
| Consultation matchs | L'utilisateur peut consulter la liste des matchs avec filtrage par stade et phase |
| Recherche billets | Le système affiche les billets disponibles par match et catégorie |
| Réservation | L'utilisateur peut sélectionner et réserver des billets (max 4 par commande) |
| Confirmation | Le système génère une référence unique et confirme la réservation |
| Annulation | L'utilisateur peut annuler une réservation non confirmée |
| Historique | L'utilisateur peut consulter l'historique de ses réservations |

### 2.1.2 Besoins Non-Fonctionnels

— **Performance :** Temps de réponse inférieur à 2 secondes pour toute opération
— **Sécurité :** Hashage des mots de passe, validation des entrées utilisateur
— **Disponibilité :** Architecture permettant une haute disponibilité (Docker)
— **Maintenabilité :** Code structuré, commenté et suivant les bonnes pratiques
— **Extensibilité :** Architecture modulaire permettant l'ajout de fonctionnalités
— **Portabilité :** Conteneurisation Docker pour le déploiement multi-plateforme

## 2.2 Conception UML

### 2.2.1 Diagramme de Cas d'Utilisation

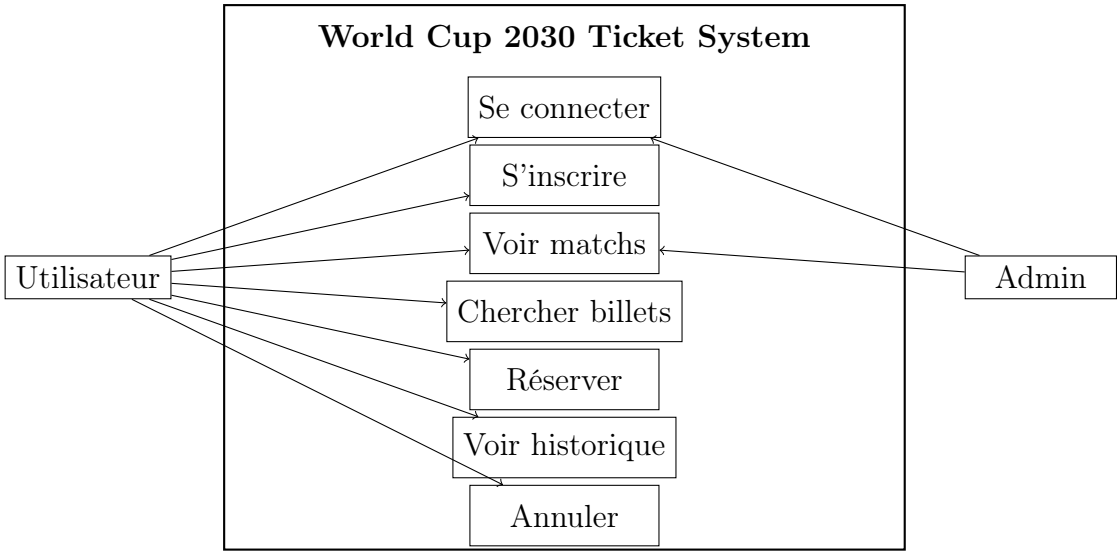



FIGURE 2.1 – Diagramme de Cas d'Utilisation

### 2.2.2 Diagramme de Classes

Le système est composé de 6 entités principales avec leurs relations :

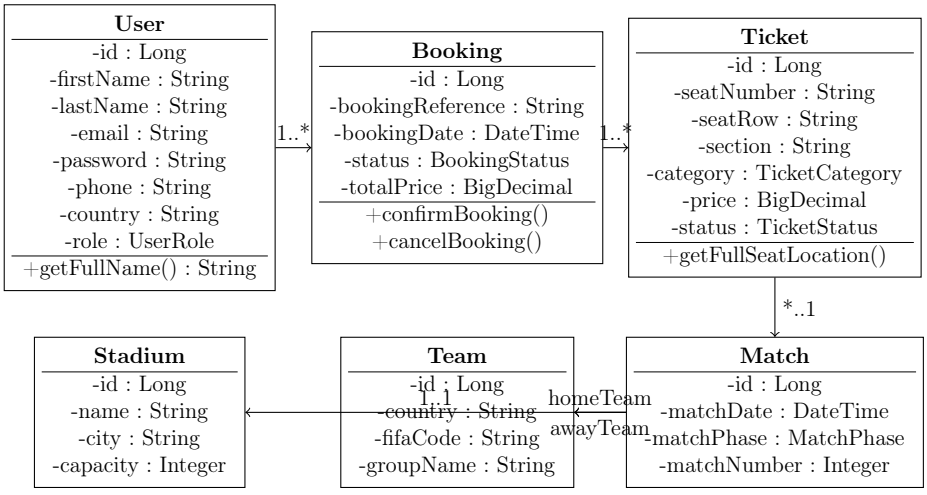



FIGURE 2.2 – Diagramme de Classes Simplifié

### 2.2.3 Diagramme de Séquence - Réservation de Billets

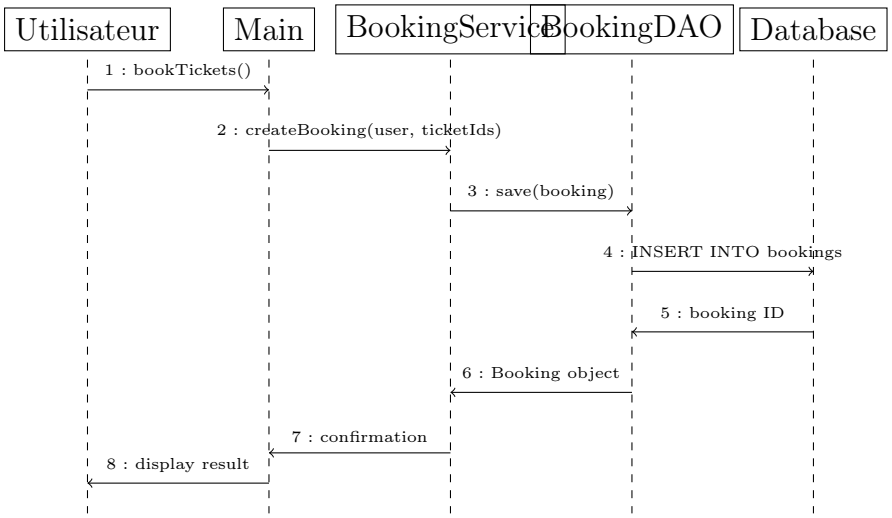



FIGURE 2.3 – Diagramme de Séquence - Réservation

## 2.3 Conception de la Base de Données

### 2.3.1 Modèle Logique de Données (MLD)

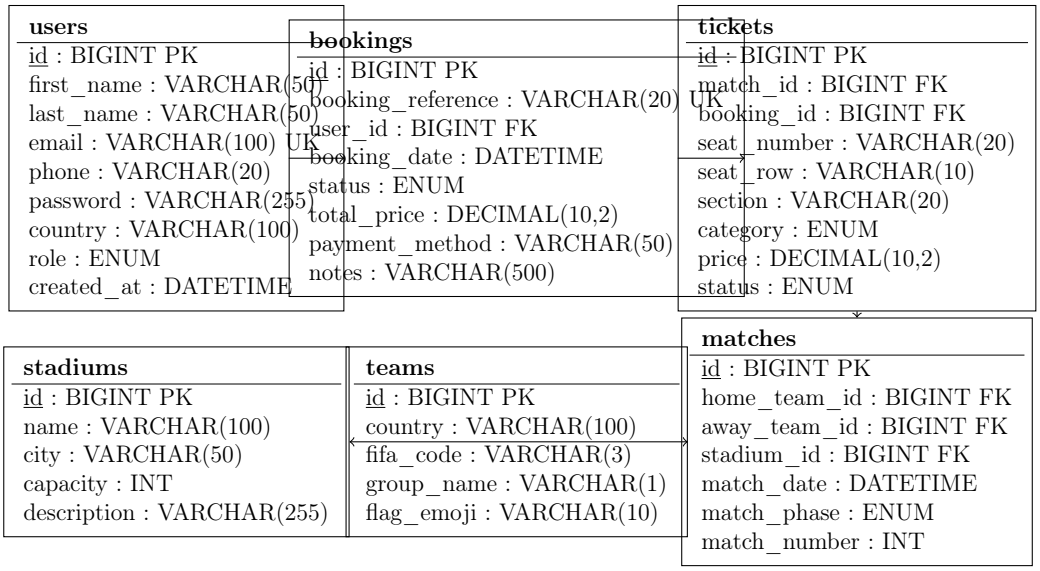



FIGURE 2.4 – Modèle Logique de Données

### 2.3.2 Dictionnaire de Données

| Champ | Type | Contrainte | Description |
|---|---|---|---|
| **Table : users** | | | |

| Champ | Type | Contrainte | Description |
|---|---|---|---|
| id | BIGINT | PK, AUTO | Identifiant unique |
| email | VARCHAR(100) | UK, NOT NULL | Adresse email unique |
| password | VARCHAR(255) | NOT NULL | Mot de passe hashé |
| role | ENUM | DEFAULT USER | USER ou ADMIN |
| **Table : tickets** | | | |
| id | BIGINT | PK, AUTO | Identifiant unique |
| category | ENUM | NOT NULL | CAT1, CAT2, CAT3 |
| price | DECIMAL(10,2) | NOT NULL | Prix en dollars |
| status | ENUM | NOT NULL | AVAILABLE, RESERVED, SOLD |
| **Table : matches** | | | |
| match_phase | ENUM | NOT NULL | GROUP_STAGE, ROUND_OF_16, etc. |

# Chapitre 3

# Environnement Technique

## 3.1 Technologies Utilisées

### 3.1.1 Langage de Programmation

— **Java 17 (LTS)** : Choisi pour sa stabilité, ses performances et le support des fonctionnalités modernes comme les switch expressions, les records et les text blocks.

### 3.1.2 Environnement de Développement

— **IDE :** IntelliJ IDEA Ultimate / Eclipse

— **JDK :** OpenJDK 17

— **Système de Build :** Apache Maven 3.x

### 3.1.3 Gestion de Projet - Maven

Maven a été utilisé pour **automatiser la gestion des dépendances, le cycle de build et faciliter l'intégration des librairies tierces** comme le driver MySQL et Hibernate.

```
<dependencies>
    <!-- Hibernate Core - ORM pour mapping objet-relationnel -->
    <dependency>
        <groupId>org.hibernate.orm</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>6.4.1.Final</version>
    </dependency>

    <!-- MySQL Connector - Driver JDBC pour MySQL -->
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <version>8.0.33</version>
    </dependency>

    <!-- Jakarta Persistence API -->
    <dependency>
        <groupId>jakarta.persistence</groupId>
        <artifactId>jakarta.persistence-api</artifactId>
        <version>3.1.0</version>
```

```
    </dependency>

    <!-- Gson - Serialisation JSON pour l'API REST -->
    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.10.1</version>
    </dependency>
</dependencies>
```

Listing 3.1 – Extrait du pom.xml - Dépendances clés

### 3.1.4 Base de Données

— **SGBD :** MySQL 8.0

— **Conteneurisation :** Docker & Docker Compose pour le déploiement reproductible

— **ORM :** Hibernate 6.4.1 avec JPA

```
services:
  mysql:
    image: mysql:8.0
    container_name: worldcup2030-mysql
    environment:
      MYSQL_ROOT_PASSWORD: root123
      MYSQL_DATABASE: worldcup2030
      MYSQL_USER: worldcup
      MYSQL_PASSWORD: worldcup2030
    ports:
      - "3306:3306"
```

Listing 3.2 – docker-compose.yml

### 3.1.5 Bibliothèques Tierces

| **Bibliothèque** | **Version** | **Utilisation** |
|---|---|---|
| Hibernate ORM | 6.4.1 | Mapping objet-relationnel, gestion du cycle de vie des entités |
| MySQL Connector/J | 8.0.33 | Driver JDBC pour la connexion à MySQL |
| Jakarta Persistence | 3.1.0 | API standard pour la persistance JPA |
| Gson | 2.10.1 | Sérialisation/Désérialisation JSON pour l'API REST |
| SLF4J Simple | 2.0.9 | Logging des opérations Hibernate |

# Chapitre 4

# Architecture et Implémentation

## 4.1 Architecture Logicielle

L'application suit une **architecture en couches** (Layered Architecture) qui permet une séparation claire des responsabilités :

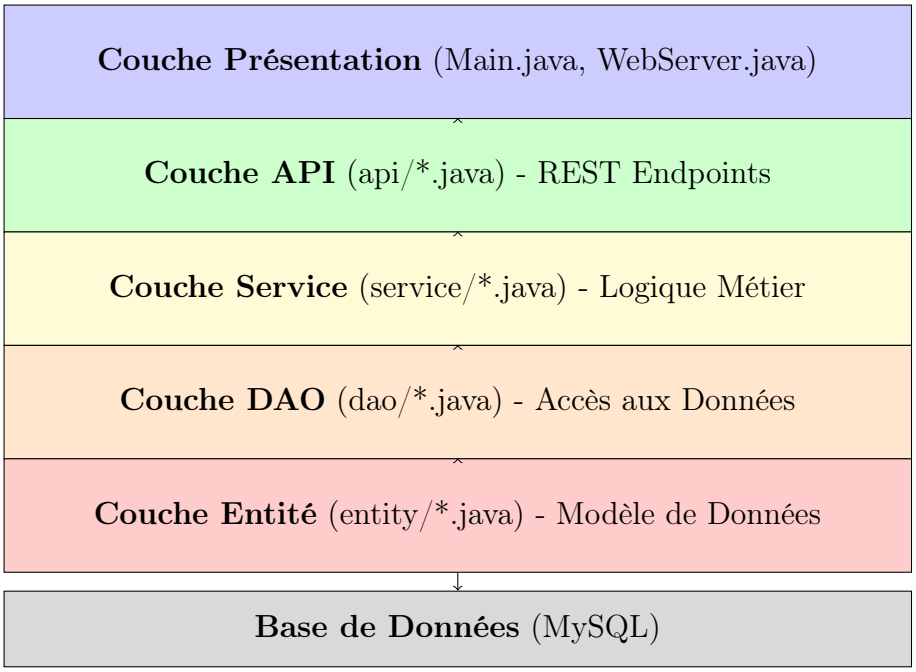



FIGURE 4.1 – Architecture en Couches

### 4.1.1 Structure des Packages

```
com.worldcup2030/
 Main.java              # Application console interactive
 WebServer.java         # Serveur HTTP intégré (API REST)
 entity/                # Entités JPA (POJO)
    User.java
    Booking.java
    Ticket.java
    Match.java
```

```
    Team.java
    Stadium.java
 dao/                    # Data Access Objects
    GenericDAO.java     # DAO générique abstrait
    UserDAO.java
    BookingDAO.java
    TicketDAO.java
    MatchDAO.java
    TeamDAO.java
    StadiumDAO.java
 service/                # Services métier
    UserService.java
    BookingService.java
    TicketService.java
    MatchService.java
 api/                    # API REST
    ApiHandler.java
    UserApi.java
    BookingApi.java
    TicketApi.java
    MatchApi.java
    TeamApi.java
    StadiumApi.java
 util/                   # Utilitaires
     HibernateUtil.java    # Singleton SessionFactory
     DataInitializer.java  # Initialisation données
```

## 4.2 Design Patterns Utilisés

### 4.2.1 Pattern Singleton - HibernateUtil

Le pattern Singleton est utilisé pour **garantir une instance unique de la SessionFactory Hibernate**, évitant ainsi la création multiple de connexions coûteuses à la base de données.

```java
public class HibernateUtil {

    private static SessionFactory sessionFactory;

    static {
        try {
            sessionFactory = new Configuration()
                    .configure("hibernate.cfg.xml")
                    .buildSessionFactory();
        } catch (Exception e) {
            System.err.println("Error initializing Hibernate: " + e.
   getMessage());
            throw new ExceptionInInitializerError(e);
        }
    }

```

```
    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }

    public static void shutdown() {
        if (sessionFactory != null) {
            sessionFactory.close();
        }
    }
}
```

Listing 4.1 – HibernateUtil.java - Pattern Singleton

**Justification :** Ce pattern assure qu'une seule connexion pool est créée pour toute l'application, optimisant les ressources et garantissant la cohérence des transactions.

### 4.2.2 Pattern DAO (Data Access Object)

Le pattern DAO **isole le code d'accès aux données du reste de l'application**, facilitant la maintenance et permettant de changer de SGBD sans modifier la logique métier.

```
public abstract class GenericDAO<T> {

    private final Class<T> entityClass;

    public GenericDAO(Class<T> entityClass) {
        this.entityClass = entityClass;
    }

    public T save(T entity) {
        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().
   openSession()) {
            transaction = session.beginTransaction();
            session.persist(entity);
            transaction.commit();
            return entity;
        } catch (Exception e) {
            if (transaction != null) transaction.rollback();
            throw e;
        }
    }

    public Optional<T> findById(Long id) {
        try (Session session = HibernateUtil.getSessionFactory().
   openSession()) {
            return Optional.ofNullable(session.get(entityClass, id));
        }
    }

    public List<T> findAll() {
        try (Session session = HibernateUtil.getSessionFactory().
   openSession()) {
            return session.createQuery("FROM " + entityClass.
   getSimpleName(),
                                       entityClass).list();
        }
```

```
    }
}
```

Listing 4.2 – GenericDAO.java - DAO Générique avec Generics

**Justification :** L'utilisation des Generics Java permet de créer un DAO réutilisable pour toutes les entités, réduisant la duplication de code.

### 4.2.3 Pattern Template Method

Les DAOs spécifiques héritent du GenericDAO et ajoutent des méthodes spécifiques :

```
public class TicketDAO extends GenericDAO<Ticket> {

    public TicketDAO() {
        super(Ticket.class);
    }

    public List<Ticket> findAvailableByMatch(Long matchId) {
        try (Session session = getSession()) {
            return session.createQuery(
                "FROM Ticket t WHERE t.match.id = :matchId " +
                "AND t.status = :status", Ticket.class)
                .setParameter("matchId", matchId)
                .setParameter("status", Ticket.TicketStatus.AVAILABLE)
                .list();
        }
    }

    public List<Ticket> findByCategory(Long matchId, Ticket.
   TicketCategory cat) {
        try (Session session = getSession()) {
            return session.createQuery(
                "FROM Ticket t WHERE t.match.id = :matchId " +
                "AND t.category = :cat AND t.status = :status", Ticket.
   class)
                .setParameter("matchId", matchId)
                .setParameter("cat", cat)
                .setParameter("status", Ticket.TicketStatus.AVAILABLE)
                .list();
        }
    }
}
```

Listing 4.3 – TicketDAO.java - Extension du GenericDAO

## 4.3 Extraits de Code Clés

### 4.3.1 Entité avec Annotations JPA

```
@Entity
@Table(name = "bookings")
public class Booking {

    public enum BookingStatus {
```

```
        PENDING, CONFIRMED, CANCELLED, COMPLETED
    }

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "booking_reference", unique = true, nullable = false)
    private String bookingReference;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @OneToMany(mappedBy = "booking", cascade = CascadeType.ALL, fetch =
    FetchType.EAGER)
    private List<Ticket> tickets = new ArrayList<>();

    @Enumerated(EnumType.STRING)
    @Column(nullable = false)
    private BookingStatus status = BookingStatus.PENDING;

    @Column(name = "total_price", precision = 10, scale = 2)
    private BigDecimal totalPrice = BigDecimal.ZERO;

    public void confirmBooking() {
        this.status = BookingStatus.CONFIRMED;
        for (Ticket ticket : tickets) {
            ticket.setStatus(Ticket.TicketStatus.SOLD);
        }
    }

    public void cancelBooking() {
        this.status = BookingStatus.CANCELLED;
        for (Ticket ticket : tickets) {
            ticket.setStatus(Ticket.TicketStatus.AVAILABLE);
            ticket.setBooking(null);
        }
        tickets.clear();
    }
}
```

Listing 4.4 – Booking.java - Entité avec relations JPA

### 4.3.2 Gestion des Transactions avec try-with-resources

```
public Booking createBooking(User user, List<Long> ticketIds) {
    Transaction transaction = null;
    try (Session session = HibernateUtil.getSessionFactory().openSession
    ()) {
        transaction = session.beginTransaction();

        Booking booking = new Booking(user);

        for (Long ticketId : ticketIds) {
            Ticket ticket = session.get(Ticket.class, ticketId);
```

```
            if (ticket != null && ticket.getStatus() == TicketStatus.
    AVAILABLE) {
                booking.addTicket(ticket);
            } else {
                throw new RuntimeException("Ticket not available: " +
    ticketId);
            }
        }

        booking.calculateTotalPrice();
        session.persist(booking);
        transaction.commit();

        return booking;
    } catch (Exception e) {
        if (transaction != null) transaction.rollback();
        throw e;
    }
}
```

Listing 4.5 – Gestion robuste des transactions

### 4.3.3 Menu Interactif avec Switch Expression (Java 17)

```
private static boolean showMainMenu() {
    System.out.println("1. View All Matches");
    System.out.println("2. Search Available Tickets");
    System.out.println("3. Book Tickets");
    System.out.println("4. View My Bookings");
    System.out.println("0. Exit");

    int choice = readInt();

    switch (choice) {
        case 1 -> viewAllMatches();
        case 2 -> searchAvailableTickets();
        case 3 -> bookTickets();
        case 4 -> viewMyBookings();
        case 0 -> { return false; }
        default -> System.out.println("Invalid option.");
    }
    return true;
}
```

Listing 4.6 – Switch Expression moderne

# Chapitre 5

# Interface Utilisateur et Tests

## 5.1 Présentation des Interfaces

### 5.1.1 Interface Console

L'application propose une interface console interactive avec des menus clairs et des emojis pour améliorer l'expérience utilisateur.

```
============================================================
         WORLD CUP 2030 MOROCCO - TICKET SYSTEM
============================================================
System ready! Welcome to the World Cup 2030 Ticket System.

----------------------------------------
       LOGIN / REGISTER
----------------------------------------
1. Login
2. Register
3. Quick Demo Login
0. Exit

Choose an option: _
```

FIGURE 5.1 – Menu de Connexion

```
============================================================
         WORLD CUP 2030 MOROCCO - MAIN MENU
   Logged in as: Demo User
============================================================
1. View All Matches
2. View Matches by Stadium
3. Search Available Tickets
4. Book Tickets
5. View My Bookings
6. Cancel Booking
7. Logout
0. Exit

Choose an option: _
```

FIGURE 5.2 – Menu Principal

```
----------------------------------------
       SEARCH TICKETS
----------------------------------------
Match #1: MAR vs ESP | GROUP STAGE | Casablanca | 2030-06-15

Available Tickets:
CATEGORY 1 - Premium ($500):
   Ticket #1 | Section A - Row 1 - Seat 1 | CAT1 | $500.00 | AVAILABLE
   Ticket #2 | Section A - Row 1 - Seat 2 | CAT1 | $500.00 | AVAILABLE

CATEGORY 2 - Standard ($300):
   Ticket #15 | Section B - Row 5 - Seat 1 | CAT2 | $300.00 | AVAILABLE

CATEGORY 3 - Economy ($150):
   Ticket #30 | Section C - Row 10 - Seat 1 | CAT3 | $150.00 | AVAILABLE

Total available: 45 tickets
```

FIGURE 5.3 – Recherche de Billets

### 5.1.2 Interface Web (API REST)

L'application expose également une API REST accessible à `http://localhost:8080` :

- `GET /api/matches` - Liste des matchs
- `GET /api/tickets?matchId=1` - Billets disponibles
- `POST /api/bookings` - Créer une réservation
- `GET /api/users/id/bookings` - Historique utilisateur

## 5.2 Scénarios de Test

### 5.2.1 Tests Nominaux

| **Scénario** | **Actions** | **Résultat Attendu** |
|---|---|---|
| Inscription réussie | Remplir tous les champs du formulaire avec des données valides | Compte créé, message de succès |
| Connexion valide | Email : demo@worldcup2030.com, Password : demo123 | Accès au menu principal |
| Réservation complète | Sélectionner match, catégorie, quantité puis confirmer | Référence booking générée |
| Consultation historique | Menu "View My Bookings" | Liste des réservations affichée |

### 5.2.2 Tests d'Erreurs

| **Scénario** | **Actions** | **Résultat Attendu** |
|---|---|---|
| Mauvais mot de passe | Connexion avec password incorrect | Message "Invalid email or password" |
| Billet indisponible | Tenter de réserver un billet déjà vendu | Message "Ticket not available" |
| Quantité excessive | Demander plus de 4 billets | Message "Invalid quantity" |
| Option invalide | Entrer une option hors menu | Message "Invalid option" |

# Chapitre 6

# Conclusion et Perspectives

## 6.1 Bilan Technique

Le projet **World Cup 2030 Morocco Ticket System** a été réalisé avec succès, respectant l'ensemble du cahier des charges initial :

**Gestion des utilisateurs :** Inscription, authentification et gestion des rôles

**Catalogue des matchs :** Navigation par stade, date et phase de compétition

**Système de billetterie :** Réservation avec 3 catégories de prix

**Gestion des réservations :** Création, confirmation et annulation

**Persistance des données :** Base MySQL avec Hibernate ORM

**Architecture robuste :** Séparation en couches (Entity, DAO, Service, API)

## 6.2 Bilan Personnel

Ce projet nous a permis d'acquérir et de consolider de nombreuses compétences :

- **Maîtrise de Hibernate/JPA :** Configuration, annotations, relations entre entités
- **Patterns de conception :** Singleton, DAO, Template Method
- **Gestion de projet Maven :** Dépendances, cycle de build, plugins
- **Java moderne :** Switch expressions, try-with-resources, Optional, Streams
- **Docker :** Conteneurisation de bases de données
- **Architecture logicielle :** Conception en couches, séparation des responsabilités

## 6.3 Difficultés Rencontrées

1. **Configuration Hibernate :** Résolution des problèmes de mapping et de lazy loading
2. **Gestion des transactions :** Implémentation correcte du rollback en cas d'erreur
3. **Relations bidirectionnelles :** Synchronisation des associations OneToMany/ManyToOne
4. **Encodage des caractères :** Support des emojis en base de données

Ces difficultés ont été surmontées grâce à la documentation officielle, Stack Overflow et les conseils de notre encadrant.

## 6.4 Perspectives et Améliorations Futures

Avec plus de temps, plusieurs améliorations pourraient être apportées :

1. **Interface Web moderne :** Développement d'un frontend React/Vue.js
2. **Application mobile :** Version Android/iOS pour la réservation mobile
3. **Paiement en ligne :** Intégration Stripe/PayPal pour les transactions
4. **Génération PDF :** Billets électroniques avec QR codes
5. **Notifications :** Emails de confirmation et rappels via JavaMail
6. **Sécurité avancée :** JWT, OAuth2, hashage bcrypt des mots de passe
7. **Tests automatisés :** JUnit 5, Mockito pour les tests unitaires et d'intégration
8. **CI/CD :** Pipeline Jenkins/GitHub Actions pour le déploiement continu

# Webographie / Bibliographie


1. **Documentation Oracle Java 17**
   https://docs.oracle.com/en/java/javase/17/
2. **Hibernate ORM Documentation**
   https://hibernate.org/orm/documentation/6.4/
3. **Jakarta Persistence (JPA) Specification**
   https://jakarta.ee/specifications/persistence/3.1/
4. **Maven Documentation**
   https://maven.apache.org/guides/
5. **Docker Documentation**
   https://docs.docker.com/
6. **MySQL 8.0 Reference Manual**
   https://dev.mysql.com/doc/refman/8.0/en/
7. **Baeldung - Java Tutorials**
   https://www.baeldung.com/
8. **Stack Overflow**
   https://stackoverflow.com/questions/tagged/java
9. **GitHub - Hibernate Examples**
   https://github.com/hibernate/hibernate-orm