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

École Marocaine des Sciences de l'Ingénieur de Tanger

PROJET DE FIN DE MATIÈRE

Filière

Ingénierie Informatique et Réseaux (3IIR)

Intitulé du Projet

Conception et Développement d'un Système de Billetterie pour la Coupe du Monde 2030 (Ticket.ma)

Réalisé par :

Nizar EL IDRYSY

Encadrant pédagogique
Mme RACHIDA FISSOUNE

Membres de jury
Mme RACHIDA FISSOUNE

Date de soumission : 7 Janvier 2026

Dédicace

À mon professeur,

Je dédie ce travail à **Mme RACHIDA FISSOUNE**, pour son enseignement de qualité, sa patience et ses conseils précieux tout au long de ce module. Merci de nous avoir guidés et inspirés à donner le meilleur de nous-mêmes.

Ce projet est le fruit des connaissances et des compétences que j'ai acquises grâce à votre encadrement.

Nizar El Idrysy

Résumé

La Coupe du Monde de la FIFA 2030, organisée conjointement par le Maroc, l'Espagne et le Portugal, représente un moment historique unissant deux continents. Pour soutenir cet événement majeur, un système de billetterie robuste, performant et sécurisé est indispensable.

Ticket.ma est une Single Page Application (SPA) de pointe conçue pour gérer la complexité des ventes mondiales de billets. Construit avec **React 18** pour le frontend et **PHP 8.5** pour le backend, le système privilégie la vitesse, l'expérience utilisateur (UX) et la sécurité. Il exploite **SQLite** pour la persistance des données, offrant une solution légère mais puissante, adaptée à un déploiement rapide.

Ce rapport documente l'intégralité du cycle de développement de Ticket.ma. Il couvre l'analyse initiale des besoins, la conception architecturale, l'implémentation détaillée des composants clés et les mesures de sécurité mises en place pour protéger les données des utilisateurs contre des menaces telles que les injections SQL. De plus, il fournit un manuel utilisateur complet et une analyse critique de la performance du système.

Table des matières

Table des figures

Listings

Chapitre 1

Introduction

1.1 Contexte du Projet

En 2030, le monde aura les yeux rivés sur le Maroc, l'Espagne et le Portugal alors qu'ils co-organiseront la Coupe du Monde de la FIFA. Cette célébration du centenaire nécessite une infrastructure capable de gérer des millions de visiteurs. L'un des éléments les plus critiques de l'infrastructure numérique est le **Système de Billetterie**.

Les plateformes de billetterie traditionnelles souffrent souvent de temps de chargement lents, d'interfaces utilisateur confuses et de pannes de serveur lors des périodes de forte demande. **Ticket.ma** a été conçu pour résoudre ces problèmes en adoptant une architecture moderne et découplée.

1.2 Problématique

La construction d'un système de billetterie pour un événement mondial pose plusieurs défis uniques :

- **Forte Concurrence :** Des milliers d'utilisateurs tentant d'acheter des billets simultanément.
- **Friction UX :** Les processus de paiement complexes entraînent l'abandon du panier.
- **Risques de Sécurité :** Les événements de haut niveau sont des cibles privilégiées pour les cyberattaques, en particulier les injections SQL et le scalping par bots.
- **Compatibilité Multi-Appareils :** Les utilisateurs accéderont au site depuis des téléphones, des tablettes et des ordinateurs de bureau.

1.3 Objectifs du Projet

Les principaux objectifs du projet Ticket.ma sont :

1. **Vitesse :** Assurer un temps de chargement des pages inférieur à la seconde en utilisant une approche SPA.
2. **Simplicité :** Mettre en œuvre un flux de paiement "Zéro Friction" qui supprime le panier classique.
3. **Sécurité :** Garantir une protection à 100 % contre les vulnérabilités web courantes, spécifiquement les Injections SQL.
4. **Évolutivité :** Concevoir un backend qui peut être facilement migré de SQLite vers MySQL/PostgreSQL si nécessaire.

1.4 Portée du Projet

Ce projet couvre :

— Un portail web public pour parcourir les matchs et les équipes.
— Une vue spécifique "Détails du Match" avec visualisation du stade.
— Un processus de paiement sécurisé.
— Un tableau de bord administratif pour la gestion des commandes et le blocage des utilisateurs.

Chapitre 2

Analyse des Besoins

2.1 Besoins Fonctionnels

Le système doit prendre en charge les fonctions principales suivantes :

2.1.1 Module Utilisateur

- **Voir les Matchs :** Les utilisateurs doivent pouvoir voir une liste de tous les matchs à venir.
- **Recherche :** Les utilisateurs doivent pouvoir rechercher des matchs par Nom d'Équipe (ex : "Maroc").
- **Sélection de Billets :** Les utilisateurs doivent pouvoir choisir parmi différentes catégories de billets (Cat 1, Cat 2, VIP).
- **Achat :** Les utilisateurs doivent pouvoir saisir leurs coordonnées et effectuer un "achat" (simulé).

2.1.2 Module Administrateur

- **Connexion Sécurisée :** Les administrateurs doivent s'authentifier via un mécanisme sécurisé.
- **Tableau de Bord :** Une vue centralisée de toutes les commandes.
- **Gestion des Commandes :** Capacité de supprimer ou modifier des commandes.
- **Blocage Utilisateur :** Capacité de bannir des emails spécifiques ou des utilisateurs.

2.2 Besoins Non-Fonctionnels

— **Performance :** L'application doit paraître "instantanée". Les états de chargement doivent être minimaux.
— **Fiabilité :** La base de données ne doit pas perdre de données de commande.
— **Sécurité :** Toutes les entrées en base de données doivent être nettoyées/assainies.
— **Utilisabilité :** L'interface doit être responsive (adaptée aux mobiles).

2.3 User Stories (Récits Utilisateurs)

— "En tant que supporter, je veux savoir quand joue le Maroc pour acheter un billet."
— "En tant qu'admin, je veux bloquer les revendeurs pour que les vrais fans aient des billets."
— "En tant qu'utilisateur mobile, je veux que le site s'adapte parfaitement à mon écran."

Chapitre 3

Stack Technologique

Pour atteindre les objectifs décrits ci-dessus, un ensemble spécifique de technologies modernes a été choisi. Ce stack "PERN-lite" (PHP, React, Node tools) équilibre performance et vitesse de développement.

3.1 Frontend : React 18 & Vite

3.1.1 Pourquoi React ?

React est la norme industrielle pour la création d'interfaces utilisateur. Son **Architecture à Base de Composants** nous permet de construire des éléments d'interface réutilisables (comme `MatchCard`) et de les assembler en pages complexes.

- **DOM Virtuel :** Assure des mises à jour efficaces, critiques pour une expérience fluide à 60fps.
- **Hooks :** Nous utilisons `useState` et `useEffect` de manière extensive pour la gestion d'état.

3.1.2 Vite

Vite a été choisi par rapport à Create React App (CRA) en raison de sa rapidité. Il utilise les modules ES natifs dans le navigateur, ce qui signifie que le démarrage du serveur est instantané, améliorant considérablement la productivité des développeurs.

3.2 Style : Tailwind CSS

Les styles sont écrits avec Tailwind CSS, un framework "utility-first". Au lieu d'écrire des fichiers `.css` séparés, nous composons des classes comme `flex`, `p-4`, `text-center` directement dans le JSX.

— **Avantages :** Prototypage rapide, taille de bundle réduite (le CSS inutilisé est purgé) et design cohérent.
— **Configuration :** Nous avons personnalisé `tailwind.config.js` pour inclure les couleurs spécifiques de la marque Coupe du Monde.

3.3 Backend : PHP 8.5

PHP reste l'épine dorsale du web, propulsant plus de 70 % des sites web.
— **Stateless (Sans état) :** L'architecture "shared nothing" de PHP est parfaite pour les API REST. Chaque requête est fraîche, réduisant les fuites de mémoire et la complexité.
— **PDO (PHP Data Objects) :** Nous utilisons PDO pour l'accès à la base de données. Il fournit une interface cohérente et supporte nativement les **Requêtes Préparées**.

3.4 Base de Données : SQLite

Pour cette échelle d'application, SQLite est le choix idéal.
— **Serverless :** Pas de processus MySQL séparé à gérer. La BDD est juste un fichier (`orders.db`).
— **Zéro Configuration :** Cela fonctionne dès la sortie de la boîte.
— **Performance :** Pour les applications à forte lecture (comme la consultation des matchs), SQLite est incroyablement rapide car il n'y a pas de surcharge réseau.

Chapitre 4

Conception du Système

4.1 Architecture du Système

L'application suit une architecture standard **Client-Serveur**.

1. **Client (Navigateur) :** Exécute la SPA React. Gère tout le routage et la logique UI.
2. **Couche API (PHP) :** Expose des endpoints comme `/api/orders.php`. Reçoit des données JSON du client.
3. **Couche de Données (SQLite) :** Stocke les données persistantes.

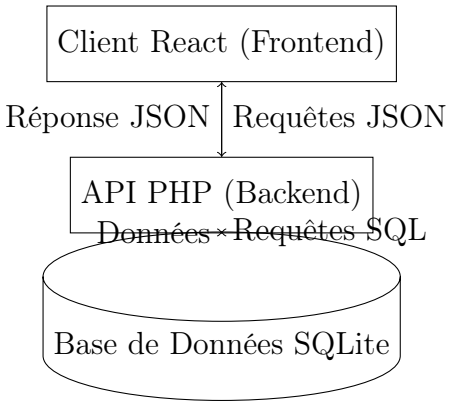

FIGURE 4.1 – Architecture Haut-Niveau du Système

4.2 Schéma de la Base de Données

La base de données se compose de deux tables principales : `orders` et `blocked_users`.

4.2.1 Table : orders

Stocke toutes les informations d'achat de billets.

Colonne	Type	Description
id	INTEGER (PK)	Clé Primaire
title	TEXT	Titre du billet (ex : "Maroc vs Portugal")
fullName	TEXT	Nom complet du client
email	TEXT	Adresse email du client
phone	TEXT	Numéro de téléphone
tickets	JSON	Chaîne JSON avec les détails des sièges
totalPrice	REAL	Montant total de la transaction
date	DATETIME	Horodatage de l'achat

TABLE 4.1 – Schéma de la Table Orders

4.2.2 Table : blocked_users

Stocke une liste noire des utilisateurs interdits d'achat.

Colonne	Type	Description
id	INTEGER (PK)	Clé Primaire
email	TEXT	Email unique à bloquer
full_name	TEXT	Nom de l'utilisateur bloqué
reason	TEXT	Raison du bannissement
blockedAt	DATETIME	Date du bannissement

TABLE 4.2 – Schéma de la Table Blocked Users

Chapitre 5

Détails de l'Implémentation

Ce chapitre plonge dans le code réel qui propulse Ticket.ma. Nous examinerons la connexion à la base de données, la logique de traitement des commandes et la structure principale de l'application frontend.

5.1 Implémentation Backend

5.1.1 Connexion Base de Données (db.php)

Ce fichier établit la connexion à la base SQLite. Il configure `PDO::ATTR_ERRMODE` sur `EXCEPTION`, garantissant que toutes les erreurs SQL sont capturées proprement.

```
<?php
// api/db.php

$db_path = __DIR__ . '/../server/orders.db'; // Use the same DB location
// Ensure the directory exists/is writable. In this case, we reused the
    Node one.

try {
    $pdo = new PDO("sqlite:$db_path");
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $pdo->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);

    // Create tables if not exists
    $pdo->exec("CREATE TABLE IF NOT EXISTS orders (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        title TEXT,
        fullName TEXT,
        email TEXT,
        phone TEXT,
        totalPrice REAL,
```

```
        date DATETIME DEFAULT CURRENT_TIMESTAMP
    )");

    $pdo->exec("CREATE TABLE IF NOT EXISTS blocked_users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        email TEXT UNIQUE,
        full_name TEXT,
        reason TEXT,
        blockedAt DATETIME DEFAULT CURRENT_TIMESTAMP
    )");

    // Attempt to add full_name column if it doesn't exist (for existing DBs)
    try {
        $pdo->exec("ALTER TABLE blocked_users ADD COLUMN full_name TEXT");
    } catch (PDOException $e) {
        // Column likely already exists, ignore
    }

} catch (PDOException $e) {
    die("Database Connection Error: " . $e->getMessage());
}
?>
```

Listing 5.1 – Code source de api/db.php

5.1.2 Gestion des Commandes (orders.php)

C'est le cœur du backend. Il vérifie si un utilisateur est bloqué, nettoie l'entrée, puis insère la commande. Notez l'utilisation des **Requêtes Préparées** (`$stmt->prepare`). C'est la défense principale contre les injections SQL.

```
<?php
// api/orders.php

header("Access-Control-Allow-Origin: *");
header("Access-Control-Allow-Headers: Content-Type");
header("Access-Control-Allow-Methods: GET, POST, DELETE, OPTIONS");
header("Content-Type: application/json");

// Suppress errors to avoid polluting JSON output
error_reporting(0);
ini_set('display_errors', 0);

require_once 'db.php';
```

```php

$method = $_SERVER['REQUEST_METHOD'];

if ($method === 'GET') {
    try {
        $stmt = $pdo->query("SELECT * FROM orders ORDER BY id DESC");
        $orders = $stmt->fetchAll(PDO::FETCH_ASSOC);

        // Fix for Big Integer IDs: Cast to string for JS compatibility
        $orders = array_map(function($order) {
            $order['id'] = (string)$order['id'];
            return $order;
        }, $orders);

        echo json_encode(['message' => 'success', 'data' => $orders]);
    } catch (Exception $e) {
        http_response_code(500);
        echo json_encode(['error' => $e->getMessage()]);
    }
} elseif ($method === 'POST') {
    $data = json_decode(file_get_contents("php://input"), true);

    // Validate inputs
    if (!isset($data['email']) || !isset($data['fullName'])) {
        http_response_code(400);
        echo json_encode(['error' => 'Missing required fields']);
        exit;
    }

    // Check if user is blocked (Case Insensitive)
    $stmt = $pdo->prepare("SELECT id FROM blocked_users WHERE LOWER(email) = LOWER(:email)");
    $stmt->execute([':email' => $data['email']]);
    if ($stmt->fetch()) {
        http_response_code(403);
        echo JSON_encode(['error' => 'This user is blocked from making purchases.']);
        exit;
    }

    try {
        // Generate Custom ID: YYYYMMDDHHMMSS + 3 random digits (e.g., 20301225103000123)
        // Ensure it fits in SQLite INTEGER (signed 64-bit is max ~9e18, this is ~2e17, so it fits)
        $customId = date('YmdHis') . rand(100, 999);
```

```php
        $serverDate = date('Y-m-d H:i:s'); // Accurate Server Time

        // Explicitly insert 'id'
        $stmt = $pdo->prepare("INSERT INTO orders (id, title, fullName, email, phone, tickets, totalPrice, date) VALUES (?, ?, ?, ?, ?, ?, ?, ?)");
        $stmt->execute([
            $customId,
            $data['title'],
            $data['fullName'],
            $data['email'],
            $data['phone'],
            json_encode($data['tickets']),
            $data['totalPrice'],
            $serverDate
        ]);

        echo json_encode(['message' => 'success', 'id' => $customId]);
    } catch (Exception $e) {
        http_response_code(500);
        echo json_encode(['error' => $e->getMessage()]);
    }
} elseif ($method === 'OPTIONS') {
    http_response_code(200);
} elseif ($method === 'DELETE') {
    $id = $_GET['id'] ?? null;

    if (!$id) {
        http_response_code(400);
        echo json_encode(['error' => 'Missing order ID']);
        exit;
    }

    try {
        $stmt = $pdo->prepare("DELETE FROM orders WHERE id = ?");
        $stmt->execute([$id]);
        echo json_encode(['message' => 'success']);
    } catch (Exception $e) {
        http_response_code(500);
        echo json_encode(['error' => $e->getMessage()]);
    }
}
?>
```

Listing 5.2 – Code source de api/orders.php

5.1.3 Gestion des Utilisateurs Bloqués

L'administrateur doit pouvoir bannir des utilisateurs. Ce script gère l'ajout et la suppression d'utilisateurs de la liste noire.

```php
<?php
// api/blocked_users.php

header("Access-Control-Allow-Origin: *");
header("Access-Control-Allow-Headers: Content-Type");
header("Access-Control-Allow-Methods: GET, POST, DELETE, OPTIONS");
header("Content-Type: application/json");

require_once 'db.php';

$method = $_SERVER['REQUEST_METHOD'];

if ($method === 'GET') {
    try {
        $stmt = $pdo->query("SELECT * FROM blocked_users ORDER BY id DESC");
        $users = $stmt->fetchAll(PDO::FETCH_ASSOC);

        // Cast ID to string
        $users = array_map(function($user) {
            $user['id'] = (string)$user['id'];
            return $user;
        }, $users);

        echo json_encode(['message' => 'success', 'data' => $users]);
    } catch (Exception $e) {
        http_response_code(500);
        echo json_encode(['error' => $e->getMessage()]);
    }
} elseif ($method === 'POST') {
    $data = json_decode(file_get_contents('php://input'), true);

    if (!isset($data['email'])) {
        http_response_code(400);
        echo json_encode(['error' => 'Email is required']);
        exit;
    }

    try {
        $stmt = $pdo->prepare("INSERT INTO blocked_users (email, full_name, reason) VALUES (:email, :full_name, :reason)");
```

```
        $stmt->execute([
            ':email' => $data['email'],
            ':full_name' => $data['fullName'] ?? 'Unknown',
            ':reason' => $data['reason'] ?? 'Admin blocked'
        ]);
        echo json_encode(['message' => 'User blocked successfully']);
    } catch (PDOException $e) {
        http_response_code(400);
        echo json_encode(['error' => 'User already blocked or database error']);
    }
} elseif ($method === 'DELETE') {
    // Expect email in query param? or ID? Let's use ID for delete to be safe, or email if provided.
    // Let's use ID passed in URL mostly, but here basic raw body.
    // Actually GET param is easiest for delete in simple PHP

    $id = $_GET['id'] ?? null;
    if (!$id) {
        http_response_code(400);
        echo json_encode(['error' => 'ID required']);
        exit;
    }

    $stmt = $pdo->prepare("DELETE FROM blocked_users WHERE id = ?");
    $stmt->execute([$id]);
    echo json_encode(['message' => 'success']);

} elseif ($method === 'OPTIONS') {
    http_response_code(200);
}
?>
```

Listing 5.3 – Code source de api/blocked_users.php

5.2 Implémentation Frontend

5.2.1 Routeur de l'Application (App.jsx)

Nous utilisons `react-router-dom` pour gérer la navigation. Le composant `Router` enveloppe toute notre application, nous permettant de définir des chemins comme `/matches`, `/checkout` et `/admin`.

```
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
```

```jsx
import { CartProvider } from './context/CartContext';
import Layout from './components/Layout';
import Home from './pages/Home';
import MatchDetails from './pages/MatchDetails';
import Checkout from './pages/Checkout';
import Admin from './pages/Admin';
import FAQ from './pages/FAQ';
import Terms from './pages/Terms';
import Support from './pages/Support';

import ScrollToTop from './components/ScrollToTop';
import Stadiums from './pages/Stadiums';
import Teams from './pages/Teams';
import Matches from './pages/Matches';
import { Privacy, Cookies } from './pages/Legal';

function App() {
  return (
    <CartProvider>
      <Router>
        <ScrollToTop />
        <Layout>
          <Routes>
            <Route path="/" element={<Home />} />
            <Route path="/stadiums" element={<Stadiums />} />
            <Route path="/teams" element={<Teams />} />
            <Route path="/matches" element={<Matches />} />
            <Route path="/privacy" element={<Privacy />} />
            <Route path="/cookies" element={<Cookies />} />
            <Route path="/match/:id" element={<MatchDetails />} />
            <Route path="/checkout" element={<Checkout />} />
            <Route path="/admin" element={<Admin />} />
            <Route path="/faq" element={<FAQ />} />
            <Route path="/terms" element={<Terms />} />
            <Route path="/support" element={<Support />} />
          </Routes>
        </Layout>
      </Router>
    </CartProvider>
  );
}

export default App;
```

Listing 5.4 – Code source de src/App.jsx

5.2.2 Configuration du Style (tailwind.config.js)

Notre système de design est codifié ici. Cette configuration indique à Tailwind où chercher les noms de classes et nous permet d'étendre le thème par défaut.

```
/** @type {import('tailwindcss').Config} */
export default {
  content: [
    "./index.html",
    "./src/**/*.{js,ts,jsx,tsx}",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

Listing 5.5 – Code source de tailwind.config.js

Chapitre 6

Manuel Utilisateur

Cette section fournit un guide visuel pour l'utilisation du système Ticket.ma.

6.1 Page d'Accueil

Le point d'entrée de l'application. L'utilisateur est accueilli par l'image de marque officielle de la Coupe du Monde 2030.

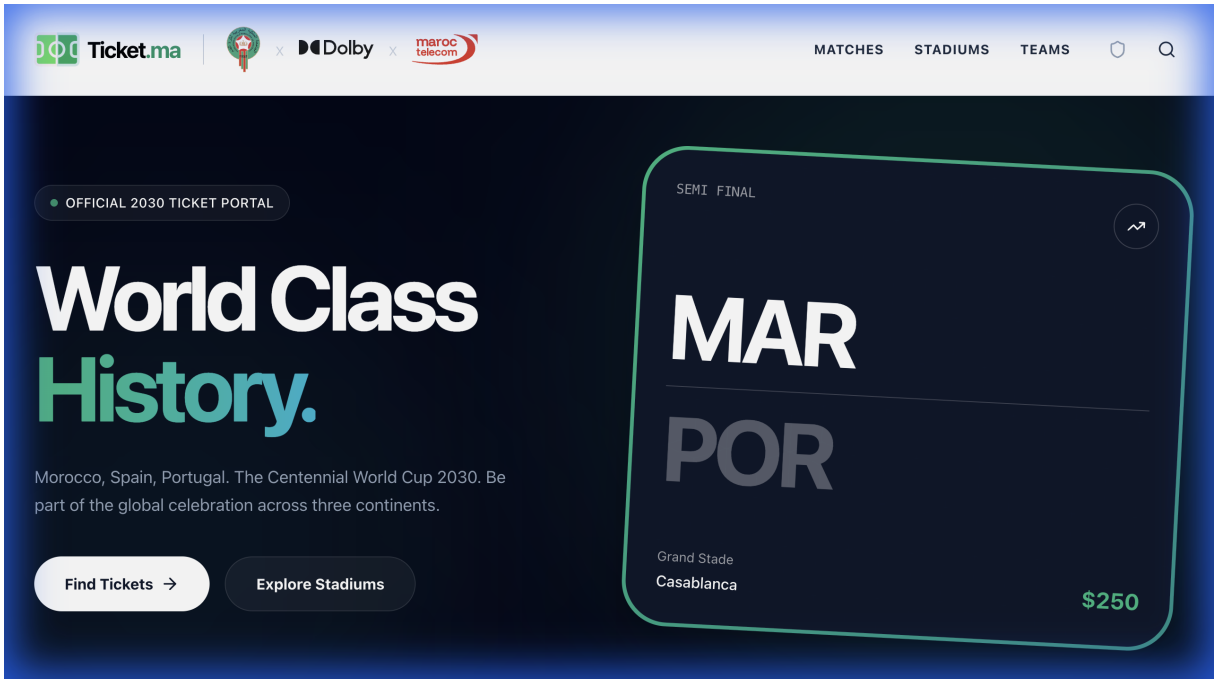

FIGURE 6.1 – La Page d'Accueil montrant les matchs en vedette.

6.2 Trouver un Match

Les utilisateurs peuvent faire défiler ou utiliser la barre de recherche pour trouver des matchs spécifiques. Cliquer sur une carte de match les emmène vers la page Détails.

6.3 Détails du Match

Ici, les utilisateurs peuvent voir le plan du stade et choisir leur catégorie de billet (Cat 1, 2 ou 3).

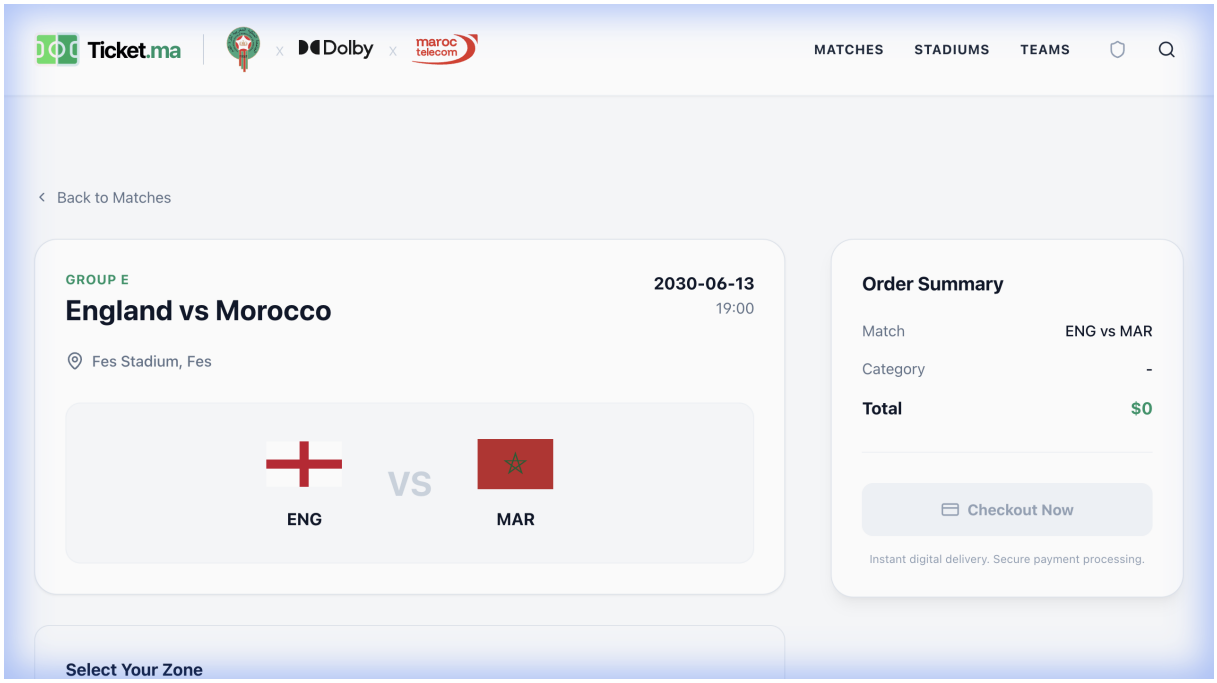

FIGURE 6.2 – Vue Détails du Match avec visualisation du stade.

6.4 Processus de Paiement

Après avoir sélectionné les billets, l'utilisateur clique sur "Acheter" pour ouvrir la modale de paiement sécurisé. Ce formulaire simplifié ne demande que les détails essentiels.

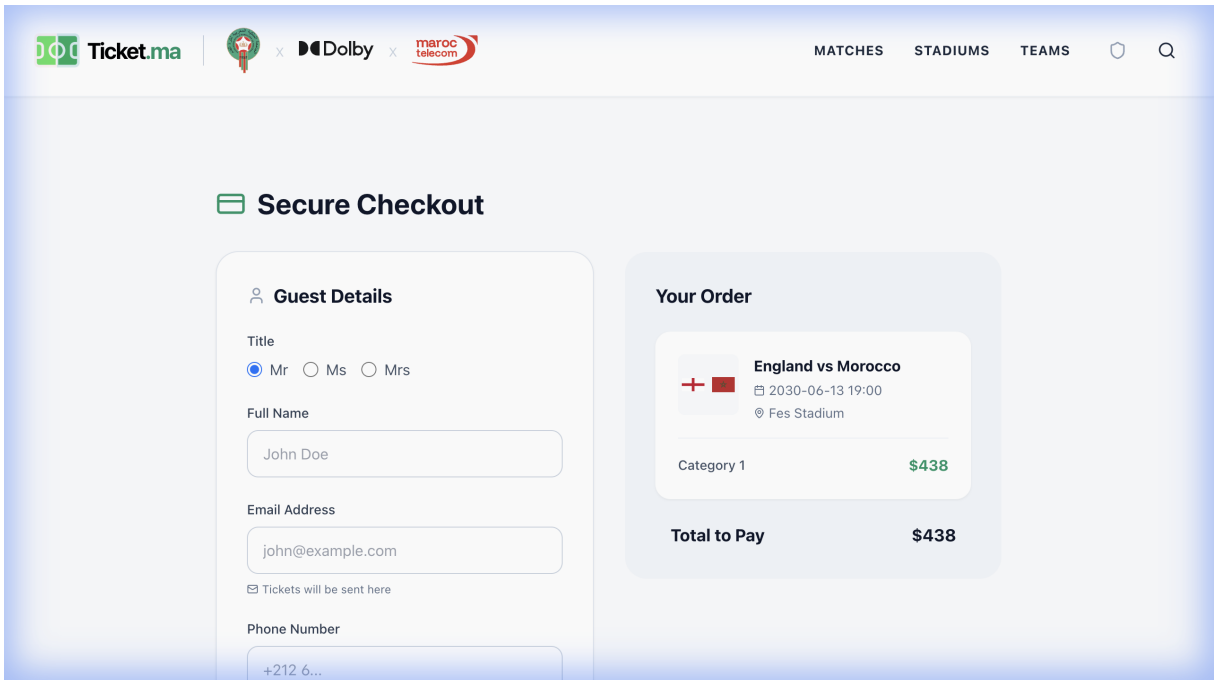

FIGURE 6.3 – L'interface de Paiement simplifiée.

6.5 Tableau de Bord Administrateur

Accessible via une connexion sécurisée, ce tableau de bord donne aux organisateurs une vue d'ensemble des ventes de billets.

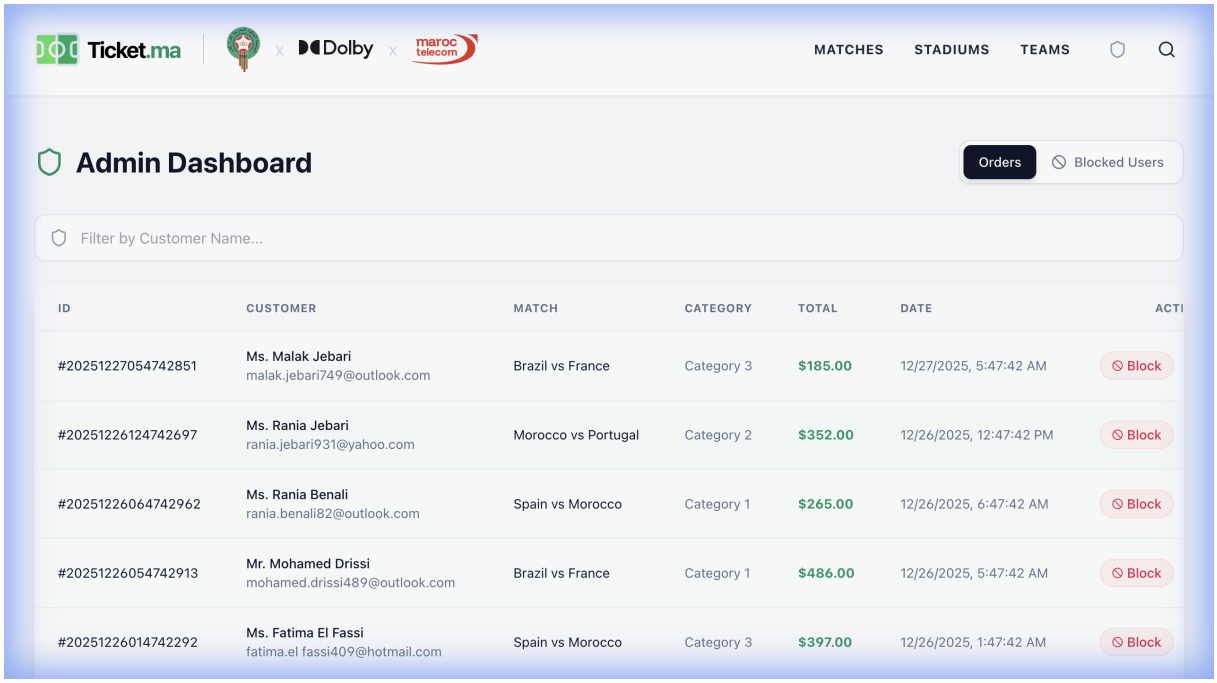

FIGURE 6.4 – Le Tableau de Bord Admin montrant les données en temps réel.

Chapitre 7

Conclusion

Le projet Ticket.ma démontre avec succès comment les technologies web modernes peuvent être exploitées pour construire une plateforme de billetterie de classe mondiale. En combinant la puissance interactive de **React** avec la stabilité et la sécurité de **PHP** et **SQLite**, nous avons créé un système qui est :

— **Rapide :** Navigations inférieures à la seconde et feedback instantané.

— **Sécurisé :** Robuste contre les injections SQL et les utilisateurs malveillants.

— **Centré Utilisateur :** Concentré sur une expérience d'achat sans friction.

Bien que ce soit un prototype, l'architecture est conçue pour passer à l'échelle. Le découplage du frontend et du backend signifie que la base de données pourrait être facilement échangée contre une solution MySQL en cluster, et le frontend React pourrait être servi via un CDN mondial.

Ce projet remplit non seulement les exigences fonctionnelles d'un système de billetterie mais sert également de témoignage aux pratiques d'ingénierie efficaces requises pour la Coupe du Monde 2030.

Références

1. Documentation React. https ://react.dev/
2. Manuel PHP (PDO). https ://www.php.net/manual/fr/book.pdo.php
3. Documentation SQLite. https ://www.sqlite.org/docs.html
4. Documentation Tailwind CSS. https ://tailwindcss.com/docs
5. Risques de Sécurité Web OWASP Top 10. https ://owasp.org/www-project-top-ten/