Système de Réservation de Terrains de Football

Application Web ASP.NET Core MVC

Avec Intégration Stripe et API REST

Aboubacar Tounkara - Développement Backend
Eli Daniel Senyo - Développement Frontend
Projet de Technologies du Commerce Électronique

2 novembre 2025

Table des matières

1 Introduction

1.1 Contexte du Projet

Ce projet consiste en la conception et le développement d'une application web complète de réservation de terrains de football. L'application permet aux utilisateurs de consulter des terrains disponibles, de réserver des créneaux horaires, et d'effectuer des paiements en ligne de manière sécurisée.

1.2 Objectifs

Les principaux objectifs de ce projet sont :

— Développer une application web moderne utilisant le framework ASP.NET Core MVC

— Implémenter un système d'authentification multi-rôles (Admin, Fournisseur, Client)

— Intégrer un système de paiement sécurisé via Stripe

— Consommer une API REST externe pour démontrer l'intégration de services tiers

— Créer une interface utilisateur responsive et intuitive

— Gérer les réservations et les disponibilités en temps réel

1.3 Technologies Utilisées

Catégorie	**Technologie**
Framework Backend	ASP.NET Core 8.0 MVC
Langage	C# 12
Base de Données	SQL Server (LocalDB)
ORM	Entity Framework Core 9.0
Authentification	ASP.NET Core Identity
Paiement en Ligne	Stripe API
API Externe	DummyJSON REST API
Frontend	Razor Views, Bootstrap 5
Icônes	Font Awesome 6
Gestion de Projet	Git
IDE	Visual Studio / VS Code

Table 1 – Stack Technique du Projet

1.4 Répartition des Tâches

Le développement de ce projet a été réalisé en collaboration par deux membres de l'équipe, chacun spécialisé dans un domaine spécifique.

Membre	Responsabilités
Aboubacar Tounkara	— Architecture et modèle de données (Models) — Logique métier et contrôleurs (Controllers) — Entity Framework Core et migrations — Services backend (Paiement, Creneaux, Factures) — Intégration Stripe API — API DummyJSON et consommation HTTP — Système d'authentification et autorisation — Gestion des réservations et annulations — Configuration de la base de données — Déploiement et tests
Eli Daniel Senyo	— Interface utilisateur et design (Views) — Intégration Bootstrap et CSS personnalisé — Pages Razor et composants visuels — Formulaires de paiement Stripe Elements — Design responsive et mobile-first — JavaScript côté client — Icônes Font Awesome et animations — Expérience utilisateur (UX/UI) — Pages d'erreur personnalisées — Optimisation de l'interface

TABLE 2 – Répartition des Tâches entre les Membres

1.4.1 Collaboration

La collaboration entre le backend et le frontend a été essentielle pour assurer la cohérence de l'application :

— **Communication continue** : Coordination sur les modèles de données et les ViewModels

— **Revues de code** : Validation mutuelle du code pour garantir la qualité

— **Tests intégrés** : Tests des flux complets de l'application ensemble

— **Résolution de problèmes** : Travail conjoint sur les bugs et les optimisations

2 Architecture de l'Application

2.1 Architecture MVC

L'application suit le pattern architectural Model-View-Controller (MVC) qui sépare les responsabilités :

— **Models** : Représentent les entités métier et la structure de données (Aboubacar)

— **Views** : Gèrent l'affichage et l'interface utilisateur (Eli)

— **Controllers** : Coordonnent les interactions entre les modèles et les vues (Aboubacar)

2.2 Structure du Projet

```
TP1/
          Controllers/          # Contr leurs MVC (Aboubacar)
          Models/              # Mod les de donn es (Aboubacar)
          Views/               # Vues Razor (Eli)
          Data/                # Contexte et initialisation DB (Aboubacar
   )
          Services/            # Services m tier (Aboubacar)
          Migrations/          # Migrations Entity Framework (Aboubacar)
          wwwroot/            # Fichiers statiques CSS, JS (Eli)
```

Listing 1 – Organisation des Dossiers

2.3 Modèle de Données

2.3.1 Entités Principales

1. **Utilisateur** : Hérite d'IdentityUser avec des propriétés personnalisées
2. **Terrain** : Représente un terrain de football avec ses caractéristiques
3. **Creneau** : Définit les plages horaires disponibles pour chaque terrain
4. **Reservation** : Enregistre les réservations effectuées
5. **PanierItem** : Gère le panier d'achat temporaire
6. **Paiement** : Stocke les informations de paiement Stripe
7. **Facture** : Génère les factures pour les réservations payées

2.3.2 Relations Entre Entités

— Un **Terrain** appartient à un **Fournisseur** (1-N)

— Un **Terrain** possède plusieurs **Créneaux** (1-N)

— Un **Créneau** peut avoir une **Réservation** (1-1)

— Une **Réservation** est liée à un **Utilisateur** (N-1)

— Une **Réservation** génère un **Paiement** et une **Facture** (1-1)

3 Fonctionnalités Principales

3.1 Système d'Authentification et d'Autorisation

3.1.1 Gestion des Rôles

L'application implémente trois rôles distincts avec des permissions spécifiques :

Rôle	Permissions
Admin	— Accès au tableau de bord administrateur — Gestion des utilisateurs — Consultation des réservations globales — Visualisation des statistiques complètes — Accès aux utilisateurs fictifs via API
Fournisseur	— Gestion de ses propres terrains (CRUD) — Visualisation de ses gains cumulés — Consultation des réservations de ses terrains — Création automatique de créneaux
Client	— Consultation des terrains disponibles — Recherche et filtrage de terrains — Ajout de créneaux au panier — Réservation et paiement en ligne — Annulation de réservations (sous conditions) — Consultation de l'historique et des factures

TABLE 3 – Permissions par Rôle

3.1.2 Comptes de Test

```
# Administrateur
Email: admin@foot.com
Mot de passe: Admin123

# Fournisseur
Email: fournisseur@foot.com
Mot de passe: Fournisseur123

# Client
Email: client@foot.com
Mot de passe: Client123
```

Listing 2 – Identifiants de Connexion

3.2 Gestion des Terrains

3.2.1 Pour les Fournisseurs

Les fournisseurs peuvent gérer leurs terrains de manière autonome :

— **Création de terrain** : Ajout avec nom, type, localisation, description
— **Types disponibles** : 5-a-side, 7-a-side, 11-a-side
— **Génération automatique** : 14 jours de créneaux créés automatiquement
— **Horaires prédéfinis** : 7 créneaux par jour (8h-21h30) avec 30 min de pause entre chaque
— **Durée des créneaux** : 1h30 de jeu + 30 min de pause
— **Tarification** : Prix adapté selon le type de terrain

Type de Terrain	Prix par Créneau
5-a-side	$35.00
7-a-side	$55.00
11-a-side	$90.00

Table 4 – Grille Tarifaire

3.2.2 Pour les Clients

— Consultation de tous les terrains disponibles
— Filtrage par type, localisation et date
— Visualisation des créneaux disponibles
— Détails complets pour chaque terrain

3.3 Système de Réservation

3.3.1 Processus de Réservation

Le processus de réservation suit un flux structuré :

1. **Sélection** : Le client choisit un terrain et un créneau
2. **Panier** : Ajout au panier (le créneau devient indisponible)
3. **Récapitulatif** : Vérification des informations
4. **Paiement** : Saisie des informations bancaires via Stripe
5. **Confirmation** : Génération de la facture et email de confirmation

3.3.2 Gestion du Panier

— Icône flottante affichant le nombre d'articles
— Ajout/Suppression de créneaux
— Calcul automatique du total
— Réservation des créneaux (non disponibles pour les autres)

3.3.3 Annulation de Réservation

Les clients peuvent annuler leurs réservations sous deux conditions **cumulatives** :

1. Moins de 24 heures depuis la réservation
2. Plus de 24 heures avant le début du créneau

Lors de l'annulation :

— Le statut passe à "Annulée"

— Le créneau redevient disponible

— Notification de confirmation

4 Intégration du Paiement Stripe

4.1 Configuration

L'application utilise l'API Stripe en mode test pour les paiements sécurisés.

4.1.1 Clés API

```
{
  "Stripe": {
    "PublishableKey": "pk_test_...",
    "SecretKey": "sk_test_..."
  }
}
```

Listing 3 – Configuration dans appsettings.json

4.2 Architecture du Paiement

4.2.1 Stripe Elements

L'implémentation utilise Stripe Elements (méthode classique) :

```
var stripe = Stripe(publishableKey);
var elements = stripe.elements();

var cardElement = elements.create('card', {
    hidePostalCode: true,
    disableLink: true
});

cardElement.mount('#payment-element');
```

Listing 4 – Initialisation Stripe Elements

4.2.2 Flux de Paiement

1. **Création du PaymentIntent** : Le serveur crée un PaymentIntent avec le montant
2. **Saisie des informations** : Le client remplit le formulaire
3. **Confirmation** : Stripe valide et confirme le paiement
4. **Enregistrement** : Le serveur enregistre la réservation et génère la facture

4.3 Informations Requises

Le formulaire de paiement demande :

— Nom du titulaire de la carte

— Numéro de carte bancaire

— Date d'expiration

— Code CVC

— Code postal

4.4 Cartes de Test

Numéro	Type	Résultat
4242 4242 4242 4242	Visa	Succès
4000 0025 0000 3155	Visa (3D Secure)	Succès après auth
4000 0000 0000 9995	Visa	Décliné

Table 5 – Cartes Bancaires de Test Stripe

4.5 Sécurité

— Aucune donnée bancaire stockée sur le serveur
— Tokenisation via Stripe
— Connexion HTTPS obligatoire
— Validation côté client et serveur

5 Intégration API REST Externe

5.1 DummyJSON API

Pour démontrer la consommation d'une API REST externe, l'application intègre l'API DummyJSON.

5.1.1 Objectif

- Démontrer l'utilisation de HttpClient
- Consommer une API REST tierce
- Désérialiser des données JSON
- Afficher des données externes dans l'interface

5.2 Implémentation

5.2.1 Service HTTP

```
public class DummyJsonService : IDummyJsonService
{
    private readonly HttpClient _httpClient;
    private const string BaseUrl = "https://dummyjson.com";

    public async Task<List<DummyJsonUser>> GetUsersAsync(int limit = 30)
    {
        var response = await _httpClient.GetAsync($"/users?limit={limit}
    ");
        var json = await response.Content.ReadAsStringAsync();
        var result = JsonSerializer.Deserialize<DummyJsonUsersResponse>(
    json);
        return result?.Users ?? new List<DummyJsonUser>();
    }
}
```

Listing 5 – Service DummyJsonService.cs

5.3 Affichage

Les 30 utilisateurs fictifs sont affichés dans la page "Gestion des Utilisateurs" (accessible uniquement à l'Admin) :

- Intégrés dans le même tableau que les utilisateurs réels
- Badge "Client" pour uniformité
- 0 réservation et $0.00 de dépenses
- Affichage : Nom, Email, Rôle, Réservations, Total

6 Interface Utilisateur

6.1 Design et Ergonomie

6.1.1 Thème Visuel

L'interface utilise un design moderne et épuré :

— Palette de couleurs cohérente (bleu/vert)
— Icônes Font Awesome pour une meilleure lisibilité
— Cards avec ombres pour la profondeur
— Espacement généreux pour la clarté

6.1.2 Navigation

La barre de navigation s'adapte selon le rôle de l'utilisateur :

— Menu contextuel selon les permissions
— Icône de panier flottante pour les clients
— Dropdown utilisateur avec accès au profil
— Responsive sur tous les écrans

6.2 Pages Principales

6.2.1 Page d'Accueil

— Section héros avec image d'accroche
— Présentation des terrains disponibles
— Boutons d'action contextuels par rôle
— Section avantages et témoignages

6.2.2 Page Réserver

— Filtres dynamiques (Type, Localisation, Date)
— Grille de terrains avec images
— Affichage des créneaux disponibles
— Action rapide "Réserver le terrain"

6.2.3 Tableau de Bord Admin

— Statistiques en temps réel
— Graphiques de revenus
— Top clients et terrains populaires
— Liste des réservations récentes

6.2.4 Gestion des Gains (Fournisseur)

— Total des gains cumulés
— Nombre de réservations payées
— Gain moyen par réservation
— Détail par terrain
— Historique complet

7 Fonctionnalités Techniques

7.1 Entity Framework Core

7.1.1 Migrations

L'application utilise EF Core Migrations pour la gestion du schéma de base de données :

```
# Cr er une migration
dotnet ef migrations add NomDeLaMigration

# Appliquer les migrations
dotnet ef database update

# R initialiser la base
dotnet ef database drop --force
dotnet ef database update
```

Listing 6 – Commandes de Migration

7.1.2 Historique des Migrations

1. `InitialCreate` : Création du schéma initial
2. `SimplifierReservationsSansPlaces` : Suppression des quantités
3. `AjouterFournisseurAuxTerrains` : Lien terrain-fournisseur

7.2 Services Métier

L'application utilise l'injection de dépendances pour les services :

Service	Responsabilité
IPaiementService	Gestion des paiements Stripe
IFactureService	Génération des factures
ICreneauService	Gestion des créneaux
IDummyJsonService	Consommation API externe

Table 6 – Services de l'Application

7.3 Sécurité

7.3.1 Authentification

— ASP.NET Core Identity pour la gestion des utilisateurs
— Hachage des mots de passe avec PBKDF2
— Cookies sécurisés HttpOnly
— Protection CSRF avec AntiForgeryToken

7.3.2 Autorisation

— Attribut `[Authorize(Roles = "...")]` sur les contrôleurs
— Vérification de propriété pour les ressources
— Redirection vers AccessDenied si non autorisé

7.4 Validation

7.4.1 Validation Côté Serveur

```
[Required(ErrorMessage = "Le nom est requis")]
[StringLength(100)]
public string Nom { get; set; }

[Required]
[EmailAddress]
public string Email { get; set; }

[Range(0, double.MaxValue)]
public decimal Prix { get; set; }
```

Listing 7 – Exemple de Validation

7.4.2 Validation Côté Client

— jQuery Validation Unobtrusive
— Validation en temps réel des formulaires
— Messages d'erreur contextuels

8 Tests et Qualité

8.1 Tests Manuels

L'application a été testée manuellement pour tous les scénarios utilisateurs :

8.1.1 Scénarios Testés

1. Inscription et connexion avec chaque rôle
2. Création et gestion de terrains par fournisseur
3. Recherche et filtrage de terrains par client
4. Ajout au panier et gestion du panier
5. Processus complet de paiement avec Stripe
6. Annulation de réservations
7. Consultation des factures
8. Visualisation des gains pour fournisseurs
9. Dashboard administrateur
10. Intégration API DummyJSON

8.2 Gestion des Erreurs

— Messages d'erreur clairs et explicites
— Page d'erreur personnalisée
— Logging avec ILogger
— Gestion des exceptions dans les contrôleurs

9 Difficultés Rencontrées et Solutions

9.1 Disponibilité des Créneaux

Problème : Les créneaux n'étaient pas marqués comme indisponibles après ajout au panier.

Solution :

— Ajout du champ `EstDisponible` au modèle Creneau

— Mise à jour lors de l'ajout au panier

— Remise à disponible lors de l'annulation

9.2 Relation Terrain-Fournisseur

Problème : Les terrains créés par les fournisseurs n'étaient pas liés correctement.

Solution :

— Migration pour ajouter `FournisseurId`

— Relation de clé étrangère vers AspNetUsers

— Filtrage par fournisseur dans les requêtes

10 Conclusion

10.1 Bilan du Projet

Ce projet de système de réservation de terrains de football a permis de mettre en pratique de nombreux concepts du développement web moderne avec ASP.NET Core MVC :

— Architecture MVC et séparation des responsabilités
— Entity Framework Core et gestion de base de données
— Authentification et autorisation multi-rôles
— Intégration de services de paiement (Stripe)
— Consommation d'APIs REST externes
— Design responsive et expérience utilisateur
— Gestion d'état et flux de données

10.2 Compétences Acquises

La réalisation de ce projet a permis de développer des compétences dans :

1. **Backend** : Développement avec C# et ASP.NET Core
2. **Base de données** : Conception et gestion avec EF Core
3. **Sécurité** : Authentification, autorisation et protection des données
4. **Paiement en ligne** : Intégration Stripe et gestion des transactions
5. **APIs** : Consommation d'APIs REST tierces
6. **Frontend** : Interfaces utilisateur avec Razor et Bootstrap
7. **Architecture** : Conception d'applications web structurées

10.3 Travail d'Équipe

La collaboration entre Aboubacar Tounkara (Backend) et Eli Daniel Senyo (Frontend) a été exemplaire :

— **Complémentarité** : Les compétences backend et frontend se sont parfaitement complétées
— **Communication** : Échanges réguliers pour assurer la cohérence de l'application
— **Qualité** : Respect des standards de développement et des bonnes pratiques
— **Résultats** : Application fonctionnelle, moderne et professionnelle

10.4 Perspectives

L'application constitue une base solide pour un système de réservation complet. Les compétences acquises en backend et frontend permettent d'envisager des projets encore plus ambitieux à l'avenir.

Ce projet démontre la maîtrise des technologies du commerce électronique et du développement web full-stack avec ASP.NET Core.

A Annexes

A.1 A. Structure de la Base de Données

A.1.1 Schéma Relationnel

AspNetUsers

- Id (PK) : nvarchar(450)
- Nom : nvarchar(100)
- Email : nvarchar(256)
- Role : nvarchar(max)
- DateInscription : datetime2

Terrains

- Id (PK) : int
- FournisseurId (FK) : nvarchar(450)
- Nom : nvarchar(100)
- Type : nvarchar(50)
- Localisation : nvarchar(200)
- Description : nvarchar(max)
- ImageUrl : nvarchar(max)

Creneaux

- Id (PK) : int
- TerrainId (FK) : int
- Date : datetime2
- HeureDebut : time
- HeureFin : time
- Prix : decimal(18,2)
- EstDisponible : bit

Reservations

- Id (PK) : int
- UtilisateurId (FK) : nvarchar(450)
- CreneauId (FK) : int
- MontantTotal : decimal(18,2)
- Statut : nvarchar(50)
- DateReservation : datetime2

Paiements

- Id (PK) : int
- ReservationId (FK) : int
- StripePaymentIntentId : nvarchar(max)
- Montant : decimal(18,2)

— Statut : nvarchar(50)
— DatePaiement : datetime2

Factures

— Id (PK) : int
— ReservationId (FK) : int
— NumeroFacture : nvarchar(50)
— Montant : decimal(18,2)
— Date : datetime2

A.2 B. Endpoints API

Méthode	Route	Description
GET	/Home/Index	Page d'accueil
GET	/Home/Reserver	Page de recherche terrains
GET	/Home/DetailsTerrain/{id}	Détails d'un terrain
GET	/Account/Register	Formulaire inscription
POST	/Account/Register	Traitement inscription
GET	/Account/Login	Formulaire connexion
POST	/Account/Login	Traitement connexion
POST	/Account/Logout	Déconnexion
GET	/Cart/Index	Panier du client
POST	/Cart/Add	Ajouter au panier
POST	/Cart/Remove	Retirer du panier
GET	/Checkout/Index	Page de paiement
POST	/Checkout/CreatePaymentIntent	Créer PaymentIntent
POST	/Checkout/ConfirmPayment	Confirmer paiement
GET	/Checkout/Success	Page de succès
GET	/Reservations/MyBookings	Mes réservations
GET	/Reservations/History	Historique factures
GET	/Reservations/Invoice/{id}	Détails facture
POST	/Reservations/Cancel/{id}	Annuler réservation
GET	/FournisseurTerrains/Index	Liste terrains fournisseur
GET	/FournisseurTerrains/Create	Formulaire création
POST	/FournisseurTerrains/Create	Enregistrer terrain
GET	/FournisseurTerrains/Gains	Gains cumulés
GET	/Admin/Dashboard	Dashboard admin
GET	/Admin/Utilisateurs	Liste utilisateurs
GET	/Admin/Reservations	Liste réservations
GET	/Admin/Terrains	Liste terrains

A.3 C. Configuration Requise

A.3.1 Environnement de Développement

— .NET 8.0 SDK ou supérieur
— Visual Studio 2022 ou VS Code
— SQL Server LocalDB ou SQL Server
— Navigateur web moderne (Chrome, Edge, Firefox)

A.3.2 Configuration appsettings.json

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=
    ReservationTerrainsDB;Trusted_Connection=true;
    MultipleActiveResultSets=true"
  },
  "Stripe": {
    "PublishableKey": "pk_test_...",
    "SecretKey": "sk_test_..."
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  }
}
```

Listing 8 – Configuration Minimale

A.4 D. Installation et Démarrage

```
# Cloner ou extraire le projet
cd reservation-terrains-football-main

# Restaurer les packages NuGet
dotnet restore

# Appliquer les migrations
dotnet ef database update

# Lancer l'application
dotnet run

# Ouvrir dans le navigateur
# https://localhost:5001
```

Listing 9 – Commandes de Démarrage

A.5 E. Références

1. Documentation ASP.NET Core : https://docs.microsoft.com/aspnet/core/

2. Entity Framework Core : https://docs.microsoft.com/ef/core/
3. Stripe API : https://stripe.com/docs/api
4. Bootstrap 5 : https://getbootstrap.com/docs/5.0/
5. DummyJSON API : https://dummyjson.com/