# Système de Gestion de Location de Voitures

*CarRental*

## Rapport de Projet

Développement d'Applications avec .NET


**Équipe de développement :**

Taha AIT BAISSI
Hossam NAZIH
Mouad AGDOUZ
Taha RAMI


Semestre 1 - 2026

# Table des matières

# 1 Introduction

## 1.1 Contexte du projet

Dans le cadre de notre formation en développement .NET, nous avons conçu et développé un système complet de gestion de location de voitures. Ce projet vise à moderniser et automatiser les processus de location de véhicules en offrant une solution logicielle robuste, scalable et maintenable.

## 1.2 Objectifs

Les principaux objectifs de ce projet sont :

— Développer une application multi-plateforme (Web, API, Desktop) pour la gestion de locations

— Implémenter une architecture propre et modulaire suivant les principes SOLID

— Mettre en place un système d'authentification et d'autorisation sécurisé

— Créer une interface utilisateur intuitive pour les clients et les employés

— Garantir la qualité du code et la maintenabilité du système

## 1.3 Portée du système

Le système CarRental couvre l'ensemble du cycle de vie d'une location de véhicule, incluant :

— La recherche et la réservation de véhicules

— La gestion des clients et des employés

— Le traitement des paiements

— La maintenance des véhicules

— La génération de rapports et de documents

# 2 Architecture du Système

## 2.1 Architecture en couches (Clean Architecture)

Notre projet adopte le pattern **Clean Architecture**, qui organise le code en couches concentriques avec des dépendances unidirectionnelles vers le centre. Cette architecture garantit une séparation claire des responsabilités et facilite les tests unitaires.

### 2.1.1 Couche Domain (Core)

La couche `CarRental.Core` constitue le cœur du système et contient :
**Entités principales :**

— `Vehicle` : Représente un véhicule avec ses attributs (VIN, marque, modèle, année, statut)

— `Booking` : Gère les réservations avec calcul de durée et détection de chevauchement

— `Client` : Profil client avec informations de permis de conduire

— `User` : Authentification et gestion des utilisateurs
— `Payment` : Traitement des paiements et transactions
— `Maintenance` : Historique de maintenance des véhicules

**Value Objects :**
— `Money` : Représentation immuable de valeurs monétaires avec validation de devise
— `DateRange` : Gestion des plages de dates avec détection de chevauchement

**Enumerations :**
— `VehicleStatus` : Available, Rented, InMaintenance, OutOfService
— `BookingStatus` : Confirmed, Pending, Cancelled, Completed, NoShow
— `PaymentStatus` : Pending, Completed, Failed, Refunded

#### 2.1.2 Couche Application

La couche `CarRental.Application` contient la logique métier et orchestre les cas d'utilisation :

**Services applicatifs :**
— `BookingService` : Création, annulation et gestion des réservations
— `VehicleService` : CRUD des véhicules, recherche de disponibilité
— `AuthService` : Authentification, inscription, réinitialisation de mot de passe
— `PaymentService` : Traitement des paiements
— `DashboardService` : Agrégation de statistiques

**DTOs (Data Transfer Objects) :** Plus de 25 DTOs pour la communication entre couches, incluant `CreateBookingDto`, `VehicleSearchDto`, `AuthResponseDto`, etc.

#### 2.1.3 Couche Infrastructure

La couche `CarRental.Infrastructure` fournit les implémentations concrètes des services :

**Services techniques :**
— `EmailService` : Envoi d'emails avec MailKit (mode dev et production)
— `PdfService` : Génération de PDF avec QuestPDF (confirmations, factures)
— `QrCodeService` : Génération de codes QR pour vérification
— `CsvImportService/CsvExportService` : Import/export de données
— `PasswordHasher` : Hachage sécurisé avec BCrypt
— `TokenService` : Génération et validation de tokens JWT

#### 2.1.4 Couche Persistence

La couche `CarRental.Persistence` gère l'accès aux données :

**Technologies :**
— Entity Framework Core 9.0
— MySQL comme SGBD
— Pattern Repository pour l'abstraction de l'accès aux données
— Pattern Unit of Work pour les transactions

**Configurations Fluent API :** Chaque entité possède sa configuration (BookingConfiguration, VehicleConfiguration, etc.) définissant les contraintes, relations et index.

## 2.2 Patterns de conception utilisés

### 2.2.1 Repository Pattern

Abstraction de la couche d'accès aux données permettant de découpler la logique métier de la persistance.

### 2.2.2 Unit of Work Pattern

Coordination des modifications sur plusieurs repositories dans une transaction unique.

### 2.2.3 Dependency Injection

Injection de dépendances native de .NET Core pour gérer le cycle de vie des services et faciliter les tests.

### 2.2.4 Result Pattern

Type générique `Result<T>` pour encapsuler les succès et les erreurs de manière explicite sans exceptions.

### 2.2.5 MVVM (Model-View-ViewModel)

Utilisé dans l'application Desktop WPF avec séparation claire entre la logique de présentation et l'interface utilisateur.

# 3 Fonctionnalités Implémentées

## 3.1 Gestion des utilisateurs et authentification

### 3.1.1 Inscription et authentification

— Inscription avec validation d'email (code à 6 chiffres)
— Connexion avec JWT (JSON Web Tokens)
— Réinitialisation de mot de passe par email
— Hachage sécurisé des mots de passe avec BCrypt
— Gestion des rôles (Client, Employee)

### 3.1.2 Profils utilisateurs

— Profil client avec informations de permis de conduire
— Profil employé avec département et poste
— Mise à jour des informations personnelles

## 3.2 Gestion des véhicules

### 3.2.1 CRUD des véhicules

— Création de véhicules avec types (Sedan, SUV, Truck, etc.)
— Modification des informations et du statut
— Suppression logique des véhicules
— Upload d'images multiples par véhicule
— Gestion des tarifs par véhicule ou type

### 3.2.2 Recherche et disponibilité

— Recherche de véhicules disponibles par plage de dates
— Filtrage par type de véhicule et prix maximum
— Détection automatique des conflits de réservation
— Calcul de disponibilité en temps réel

## 3.3 Système de réservation

### 3.3.1 Processus de réservation

— Sélection de véhicule avec dates de début et fin
— Validation de la disponibilité
— Calcul automatique du montant total
— Confirmation par email avec PDF
— Génération de code QR pour vérification

### 3.3.2 Gestion des réservations

— Annulation avec politique (minimum 24h avant le début)
— Modification des dates (sous conditions)
— Suivi du statut (Pending, Confirmed, Completed, Cancelled)
— Calcul de frais de retard automatique
— Historique complet des réservations

## 3.4 Gestion des paiements

— Traitement des paiements avec statut
— Support de multiples méthodes de paiement
— Génération de factures PDF
— Référence de transaction unique
— Historique des paiements par client

### 3.5 Maintenance des véhicules

— Planification de maintenance préventive
— Suivi des interventions avec coûts
— Historique de maintenance par véhicule
— Mise à jour automatique du statut du véhicule
— Calcul du kilométrage depuis dernière maintenance

### 3.6 Tableau de bord et statistiques

— Vue d'ensemble en temps réel :
  — Total de véhicules et répartition par statut
  — Nombre de clients et réservations actives
  — Revenus total et mensuel
— Dernières activités et réservations récentes
— Statistiques de performance

### 3.7 Génération de documents

— Confirmations de réservation en PDF avec QR code
— Factures détaillées avec calcul de taxes
— Export CSV des véhicules et réservations
— Import de véhicules en masse via CSV
— Emails HTML formatés avec templates

## 4 Technologies et Outils Utilisés

### 4.1 Framework et langages

| Technologie | Version/Description |
|---|---|
| .NET | 9.0 |
| C# | 12.0 |
| ASP.NET Core | 9.0 (Web MVC et Web API) |
| WPF | .NET 9.0 (Application Desktop) |
| Entity Framework Core | 9.0 |

Table 1 – Technologies principales

### 4.2 Packages NuGet importants

**Couche Infrastructure :**

— `QuestPDF` : Génération de PDF professionnels
— `QRCoder` : Création de codes QR

— `MailKit` : Envoi d'emails SMTP
— `BCrypt.Net-Next` : Hachage de mots de passe

**Couche API :**

— `Microsoft.AspNetCore.Authentication.JwtBearer` : Authentification JWT
— `Scalar.AspNetCore` : Documentation API interactive
— `Swashbuckle/OpenAPI` : Spécification OpenAPI

**Couche Persistence :**

— `Pomelo.EntityFrameworkCore.MySql` : Provider MySQL pour EF Core
— `Microsoft.EntityFrameworkCore.Tools` : Migrations

## 4.3 Base de données

**MySQL** avec les caractéristiques suivantes :

— 14 tables principales
— Relations many-to-many et one-to-many bien définies
— Index sur colonnes fréquemment interrogées
— Contraintes de clés étrangères avec comportement CASCADE/RESTRICT
— Support des audits avec `CreatedAt` et `UpdatedAt`

## 4.4 Architecture de présentation

### 4.4.1 Web API (Backend)

— RESTful API avec 12 contrôleurs
— Documentation Scalar avec thème BluePlanet
— Middleware d'exception globale
— CORS configuré pour frontend
— Authentification JWT avec expiration

### 4.4.2 Web MVC (Frontend Client)

— ASP.NET Core MVC
— Authentification par cookies
— Localisation en français (fr-FR)
— Communication avec API via HttpClient
— Session management pour l'état utilisateur

### 4.4.3 Application Desktop (WPF)

— Interface riche pour employés
— Pattern MVVM avec `ViewModelBase`
— Navigation entre vues avec `NavigationService`
— Communication API avec `ApiClient`
— Gestion d'état avec `AppState` et `SessionManager`

# 5 Structure du Projet

## 5.1 Organisation des dossiers

```
CarRental/
|-- CarRental.Core/              # Domain Layer
|   |-- Entities/                # Entites metier
|   |-- ValueObjects/            # Value Objects
|   |-- Enums/                   # Enumerations
|   |-- Interfaces/              # Interfaces abstraites
|
|-- CarRental.Application/       # Application Layer
|   |-- Services/                # Services applicatifs
|   |-- Interfaces/              # Interfaces de services
|   |-- DTOs/                    # Data Transfer Objects
|   |-- Common/                  # Modeles communs (Result)
|
|-- CarRental.Infrastructure/    # Infrastructure Layer
|   |-- Services/                # Services techniques
|   |-- Security/                # Securite (JWT, Hash)
|   |-- Settings/                # Configuration
|
|-- CarRental.Persistence/       # Data Access Layer
|   |-- Configurations/          # EF Configurations
|   |-- Repositories/            # Implementations Repository
|   |-- Migrations/              # Migrations EF Core
|   |-- UnitOfWork/              # Pattern UnitOfWork
|   |-- Seed/                    # Donnees initiales
|
|-- CarRental.WebApi/            # API REST
|   |-- Controllers/             # Controleurs API
|   |-- Middleware/              # Middleware personnalises
|
|-- CarRental.Web/               # Application Web MVC
|   |-- Controllers/             # Controleurs MVC
|   |-- Views/                   # Vues Razor
|   |-- Models/                  # ViewModels
|   |-- wwwroot/                 # Ressources statiques
|
|-- CarRental.Desktop/           # Application WPF
    |-- Views/                   # Vues XAML
    |-- ViewModels/              # ViewModels MVVM
    |-- Services/                # Services client API
    |-- Controls/                # Controles personnalises
```

Listing 1 – Structure du projet

## 5.2 Flux de dépendances

Les dépendances suivent strictement les règles de Clean Architecture :

`Presentation →Application →Infrastructure/Persistence →Core`

Seule la couche Core n'a aucune dépendance externe. Les couches externes dépendent toujours des couches internes via des interfaces.

# 6 Sécurité

## 6.1 Authentification et autorisation

— **JWT (JSON Web Tokens)** pour l'authentification stateless
— Configuration avec Issuer, Audience et SecretKey
— Expiration des tokens après 1 heure
— Claims-based authorization avec rôles
— Refresh token non implémenté (amélioration future)

## 6.2 Protection des données

— Hachage BCrypt avec salt automatique
— Aucun mot de passe stocké en clair
— Validation des entrées côté serveur
— Protection CSRF dans Web MVC
— Tokens de réinitialisation avec expiration

## 6.3 Bonnes pratiques

— HTTPS obligatoire en production
— CORS configuré pour origines autorisées uniquement
— Logging des tentatives d'authentification
— Rate limiting (à améliorer)
— Validation des DTOs avec Data Annotations

# 7 Conclusion

Ce projet de système de gestion de location de voitures démontre une maîtrise complète de l'écosystème .NET et des bonnes pratiques de développement logiciel. L'architecture Clean Architecture adoptée garantit la maintenabilité et l'évolutivité du système.

## 7.1 Compétences acquises

Au cours de ce projet, notre équipe a développé et renforcé les compétences suivantes :

— **Architecture logicielle** : Conception et implémentation d'une Clean Architecture
— **Entity Framework Core** : Modélisation de données, migrations, relations complexes
— **ASP.NET Core** : Développement d'API RESTful et applications Web MVC
— **WPF et MVVM** : Création d'applications Desktop avec séparation UI/logique
— **Sécurité** : Authentification JWT, hachage de mots de passe, autorisation
— **Patterns de conception** : Repository, Unit of Work, Dependency Injection, Result
— **Travail en équipe** : Collaboration, versioning avec Git, répartition des tâches

## 7.2 Résultats obtenus

Nous avons livré un système fonctionnel comprenant :

— Une API REST complète avec 12 contrôleurs et documentation Scalar

— Une application Web pour les clients avec 5 contrôleurs MVC

— Une application Desktop pour les employés avec 15+ vues

— 16 entités métier avec relations et validations

— Plus de 25 DTOs pour la communication inter-couches

— Services complets (Email, PDF, QR, Import/Export)

## 7.3 Perspectives

Ce projet constitue une base solide pour une application de production. Les améliorations futures identifiées (tests automatisés, intégrations de paiement, application mobile) permettraient de transformer ce prototype en une solution commerciale viable.

Notre équipe est fière du résultat obtenu et reconnaît que ce projet a été une expérience d'apprentissage enrichissante, nous préparant efficacement aux défis du développement logiciel professionnel.

*Semestre 1 - 2026*