Résumé Complet - .NET Framework

ASP.NET Core, Entity Framework Core, LINQ

AmineGR03

28 janvier 2026

Table des matières

Introduction à .NET

Qu'est-ce que .NET ?

.NET est :

— Un standard proposé par Microsoft en 2002

— Une plateforme de développement, gratuite et open source qui supporte un grand nombre de langages de programmation

— Un cadre de travail formé d'outils, de modules actifs au runtime et de classes formant une API très étendue

— Un environnement d'exécution sécurisé

— Le remplaçant de l'architecture 3-tiers DNA (Distributed interNet Architecture)

Architecture .NET

L'architecture .NET est composée de plusieurs couches :

1. **CLR (Common Language Runtime)** : Il représente la machine virtuelle de la plateforme (un peu comme la JVM pour Java). Il est responsable de l'exécution des applications et gère tous les aspects de sécurité.
2. **Un ensemble de librairie de classes** : Située au-dessus du CLR, cette couche offre une API pour la gestion des données. On y retrouve :
   — Windows Forms (WinForms) : ensemble de classes permettant la conception d'IHM pour Windows
   — ADO.NET (Data and XML) : une nouvelle génération de composants d'accès aux bases de données
   — ASP.NET : fournit un ensemble de classes pour la conception de sites dynamiques, la création d'IHM pour le web, les WebForms et la conception de services web
3. **Les langages supportés** : Au sommet de la pile, nous avons les langages supportés par .NET tels que C#, VB.NET, C++, F#, qui sont des produits de Microsoft, et d'autres tels que Cobol, Delphi, etc.

Les bases .NET

Le CLR

Ce runtime est le moteur de .NET, c'est lui qui est en charge de l'exécution des logiciels écrits pour .NET compilés en CIL (Common Intermediate Language) avec un système appelé JIT (Just in Time).

Le CLR gère :

— La compilation en code natif et l'exécution de code IL (Intermediate Language)

— La gestion de la sécurité

— La gestion de la mémoire

— La gestion des processus

— La gestion des threads (tâches)

La BCL ou FCL

La Base Class Library (BCL) ou Framework Class Library (FCL) :

— Se place directement au-dessus du CLR

— Offre de nombreuses classes, interfaces et types qui forment le socle des développements sous .NET

— Permet d'unifier les développements puisqu'elle fournit l'ensemble des outils de base aux applications

Les espaces de noms

— Une sorte de « boîte » dans laquelle des classes ayant (plus ou moins) rapport entre elles sont regroupées

— Exemples : `System.Data.SqlClient`, `System.Web`

— Une **Assembly** (assemblée en français) est un fichier DLL dans lequel les classes sont stockées

Du code source au binaire exécuté

Le processus de compilation en .NET :

1. Code source (C#, VB.NET, etc.)
2. Compilation en CIL (Common Intermediate Language)
3. Exécution par le CLR avec compilation JIT en code natif
4. Exécution du code natif

ASP.NET et ASP.NET Core

Qu'est-ce qu'ASP.NET ?

ASP.NET est :

— Une plateforme de développement d'applications web sous Windows

— Utilise, par défaut, le serveur Web de référence de Microsoft : IIS (Internet Information Services)

— Repose sur le .NET Framework

— Une plateforme Web unifiée fournissant les services nécessaires à la création des applications

— Ne dépend ni du langage de programmation ni du navigateur

ASP.NET vs ASP.NET Core

ASP.NET	**ASP.NET Core**
Pour Windows	Pour Windows, Mac et Linux
Utilise le runtime .NET Framework	Utilise le runtime .NET Core
Bonnes performances	Plus performant qu'ASP.NET
Supporte C#, F# et VB	Supporte C# et F#
Disponible sous Visual Studio	Disponible sous VS, VSCode

Table 1 – Comparaison ASP.NET et ASP.NET Core

Modèle MVC

Introduction

Le modèle MVC (Model-View-Controller) :

— Introduit par Trygve Reenskaug en 1978

— Permet de bien organiser le code source

— Conçu, initialement, pour les applications client-lourd et ensuite généralisé aux applications client-léger (Web)

— Une approche consistant à séparer l'affichage des informations, les actions de l'utilisateur et l'accès aux données

— Chacun de ces composants est construit pour manipuler un aspect particulier de développement de l'application

— Indispensable pour des applications dynamiques et de taille importante

Les trois composants

1. Modèle

La partie qui concerne les données et l'état de notre application. Pour le cas d'une base de données, on peut utiliser un ORM (Object-Relational Mapping) comme Entity Framework.

2. Vue

La partie qui concerne l'affichage : l'interface avec laquelle l'utilisateur interagit (HTML + CSS...)

3. Contrôleur

C'est l'intermédiaire entre le modèle et la vue. Il reçoit la requête de l'utilisateur. Il demande les données au modèle, les analyse et renvoie le résultat à afficher à la vue.

ASP.NET MVC

ASP.NET MVC :

— Créé en 2007 par Scott Guthrie et intégré dans ASP.NET depuis 2009

— Framework de développement d'applications web selon le design pattern : MVC

— Par rapport à ASP.NET : ASP.NET MVC permet de structurer davantage l'application, en créant des composants avec des rôles bien identifiés

— Indispensable pour des applications dynamiques et de taille importante

Déroulement d'une requête MVC

1. Le client envoie une requête depuis la Vue
2. Le Contrôleur intercepte et analyse la requête du client
3. Le Contrôleur détermine quelle partie du Modèle est concernée afin d'effectuer les traitements nécessaires
4. Le Modèle s'occupe de l'interaction avec les données, applique les règles métier et renvoie les données au Contrôleur
5. Le Contrôleur sélectionne la Vue correspondante et lui injecte les données
6. La Vue affiche les données au client

Structure d'un projet ASP.NET MVC

- **Controllers/** : Logique métier, actions (C#)
- **Models/** : Données, objets, classes (C#)
- **Views/** : Pages HTML dynamiques (.cshtml)
  - **Shared/** : Layout général (_Layout.cshtml)
  - **Home/** : Pages spécifiques au contrôleur Home
- **App_Start/** : Fichiers de config (RouteConfig.cs)
- **Global.asax** : Point de départ de l'app, appelle RouteConfig
- **packages.config** : Packages NuGet installés
- **Web.config** : Config de l'application (base de données, etc.)

Entity Framework Core

Introduction à ORM

ORM (Object-Relational Mapping) : Mapping Objet-Relationnel

- Programme informatique jouant le rôle de traducteur entre le modèle relationnel et le modèle objet
- Permet d'interroger et manipuler les données à partir d'une base de données à l'aide d'un paradigme orienté objet
- Deux composants dans les ORM :
  - **Entités** : Instanciation d'une classe (étudiant, enseignant, cours, projet, etc.)
  - **Gestionnaire d'entités** : à utiliser pour persister les entités dans la base de données

Entity Framework

Entity Framework est :

- Un framework ORM open source pour les applications .NET prises en charge par Microsoft
- Permet de créer une couche d'accès aux données liées à une base de données relationnelle
- Permet aux développeurs de manipuler des données à l'aide d'objets de classes C# sans se concentrer sur les tables et colonnes d'une base de données (où ces données sont stockées)
- Permet aux développeurs de créer et maintenir des applications orientées données avec moins de code, par rapport aux applications traditionnelles, grâce à LINQ vers les entités

Fonctionnement d'Entity Framework

Composants principaux

- **Entité** : Une classe qui correspond à une table de base de données
- **Le gestionnaire d'entités** : LINQ to Entities

EDM : Entity Data Model

EDM est une représentation en mémoire de l'ensemble des métadonnées. EDM consiste en trois parties :

1. **Conceptuelle** : contient les classes du modèle et ses relations
2. **Stockage** : contient le modèle physique de la base : tables, vues, procédures stockées, les relations et les clés
3. **Mapping** : définit les mécanismes de passage du modèle conceptuel au stockage

Fonctionnalités d'EF

À l'aide d'EDM, EF peut :

— Effectuer des opérations CRUD (Create, Read, Update and Delete)

— Créer des requêtes SQL à partir de requêtes LINQ, créer des commandes INSERT, UPDATE et DELETE et transformer le résultat de la base de données en objets d'entité

— Traduire les requêtes LINQ-to-Entities en requêtes SQL pour les bases de données relationnelles

— Reconvertir les résultats (requêtes SQL) en objets d'entité

— Suivre et sauvegarder les changements dans la base de données

Architecture d'Entity Framework

— **Object Service** : est le point d'entrée pour l'accès aux données d'une base de données. Il convertit les résultats SQL en objets C#

— **Entity Client Data Provider** : la tâche principale est de transformer une expression de LINQ à une requête SQL

— **ADO.NET Data Provider** : Gère la communication technique avec la BD

Les différentes approches

1. Database First

On crée la base de données (ou on a une base de données qui existe déjà) et Entity Framework génère nos entités à partir de cette base de données.

2. Code First

On crée les entités puis Entity Framework génère la base de données.

3. Model First

On crée notre modèle (de classe) et Entity Framework génère la base de données et les entités correspondantes.

Approche Database First : Étapes pratiques

1. Création de la base de données

```sql
CREATE DATABASE BDVENTE;
USE BDVENTE;

CREATE TABLE PRODUITS (
    ID INT PRIMARY KEY IDENTITY(1,1),
    NOM VARCHAR(100) NOT NULL,
    DESCRIPTION VARCHAR(255),
    PRIX DECIMAL(10,2) NOT NULL,
    QUANTITE INT NOT NULL,
    DATE_AJOUT DATE DEFAULT GETDATE(),
    DISPONIBLE BIT DEFAULT 1
);
```

2. Création d'une application ASP.NET Core MVC

1. Créer un nouveau projet sous Visual Studio : `Fichier` → `Nouveau` → `Projet`
2. Sélectionner `Application web ASP.NET Core MVC`
3. Remplir le champ Nom par : `App_ACHAT`
4. Valider et attendre la fin de création du projet

3. Installation des packages NuGet

Méthode 1 - Interface graphique :

— `Outils` > `Gestionnaire de packages NuGet` > `Gérer les packages NuGet pour la solution`

— Chercher et installer ces packages :

— `Microsoft.EntityFrameworkCore`

— `Microsoft.EntityFrameworkCore.SqlServer`

— `Microsoft.EntityFrameworkCore.Tools`

Méthode 2 - Console :

```
Install-Package Microsoft.EntityFrameworkCore
Install-Package Microsoft.EntityFrameworkCore.SqlServer
Install-Package Microsoft.EntityFrameworkCore.Tools
```

4. Génération des modèles (Database First)

Exécuter la commande suivante dans la console NuGet :

```
Scaffold-DbContext "Server=...;Database=BDVENTE;..."
    Microsoft.EntityFrameworkCore.SqlServer
    -OutputDir Models
```

Cela crée : `VenteContext.cs` (classe DbContext) et `Produit.cs` (classe Model)

5. Configuration de la chaîne de connexion

Dans `appsettings.json` :

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=...;Database=BDVENTE;..."
  }
}
```

6. Ajout du DbContext dans Program.cs

```
builder.Services.AddDbContext<BdVenteContext>(options =>
    options.UseSqlServer(builder.Configuration
        .GetConnectionString("DefaultConnection")));
```

DbContext

Définition

Une classe héritant de la classe `DbContext` est appelée **classe de contexte** (context class) dans le framework d'entité.

Rôles du DbContext

`DbContext` est la classe principale responsable de l'interaction avec la base de données. Il est responsable de plusieurs activités :

- **Querying** : convertit les requêtes LINQ-to-Entities en requêtes SQL et les envoie à la base de données
- **Change Tracking** : assure le suivi des modifications apportées aux entités après une requête à partir de la base de données
- **Persisting Data** : effectue les opérations d'insertion, de mise à jour et de suppression dans la base de données, en fonction des états de l'entité
- **Object Materialization** : convertit les données brutes de la base de données en objets d'entité

DbSet

- La classe de contexte (`BdVenteContext`) inclut l'ensemble d'entités de type `DbSet<TEntity>`
- `DbSet` est considérée comme une propriété de la classe de contexte et liée à une table de la base de données
- `DbSet` contient un ensemble de méthodes permettant d'effectuer les opérations CRUD nécessaires

Opérations CRUD avec Entity Framework

Read (Lire)

```
public async Task<IActionResult> Index()
{
    return View(await _context.Produits.ToListAsync());
}
```

- Récupère la liste de tous les produits depuis la base de données
- Retourne la vue `Index.cshtml` avec les données

Create (Créer)

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind(...)] Produit produit)
{
    if (ModelState.IsValid)
    {
        _context.Add(produit);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(produit);
}
```

- Récupère les données saisies dans le formulaire
- Les ajoute à la base de données avec `_context.Add(...)`
- Enregistre avec `SaveChangesAsync()`
- Redirige vers `Index()` après création

Update (Mettre à jour)

```
[HttpPost]
public async Task<IActionResult> Edit(int id, [Bind(...)] Produit
    produit)
{
    if (id != produit.Id)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(produit);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            // Gestion des erreurs de concurrence
        }
        return RedirectToAction(nameof(Index));
    }
    return View(produit);
}
```

— Vérifie que l'id correspond bien à l'objet envoyé

— Met à jour le produit dans la base avec `_context.Update(...)`

— Gère les erreurs de concurrence avec `try...catch`

Delete (Supprimer)

```
[HttpPost, ActionName("Delete")]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var produit = await _context.Produits.FindAsync(id);
    if (produit != null)
    {
        _context.Produits.Remove(produit);
        await _context.SaveChangesAsync();
    }
    return RedirectToAction(nameof(Index));
}
```

— Supprime réellement le produit avec `_context.Produits.Remove(...)`

— Enregistre les changements

LINQ to Entities

Introduction

LINQ to Entities :

— Assure une prise en charge de la technologie LINQ (Language Integrated Query), ce qui permet aux développeurs d'écrire des requêtes par rapport au modèle conceptuel Entity Framework à l'aide du langage C#

— Convertit les requêtes Language-Integrated en requêtes sous forme d'arbre de commande, exécute les requêtes dans le Framework Entity et retourne des objets qui peuvent être utilisés à la fois par le Framework Entity et LINQ

— **LINQ to Entities** = utilisation de LINQ sur Entity Framework Core

Types d'opérateurs LINQ

Type	Opérateurs LINQ
Projection	Select, SelectMany
Filtrage	Where, OfType
Tri	OrderBy, ThenBy, Reverse, OrderByDescending
Jointures	Join, GroupJoin, Include
Agrégation	Count, Sum, Min, Max, Average
Groupement	GroupBy
Quantification	Any, All, Contains

Table 2 – Opérateurs LINQ principaux

Syntaxe de requête

Sélection : Select, SelectMany

Syntaxe de requête :

```
var produits = from p in _context.Produits
               select p;
```

Syntaxe Lambda :

```
var produits = _context.Produits.Select(p => p.Nom);
```

Filtrage : Where

Syntaxe de requête :

```
var produits = from p in Produits
               where p.Prix > 50
               select p;
```

Syntaxe Lambda :

```
var produits = Produits.Where(p => p.Prix > 50);
```

Jointure simple : Join

Syntaxe de requête :

```
var resultat = from p in Produits
               join c in Categories
               on p.CategorieId equals c.Id
               select p;
```

Syntaxe Lambda :

```
var resultat = Produits.Join(
    Categories,
    p => p.CategorieId,
    c => c.Id,
    (p, c) => new {
        ProduitNom = p.Nom,
        Prix = p.Prix,
        CategorieNom = c.Nom
    }
);
```

Tri : OrderBy, OrderByDescending

Syntaxe de requête :

```
var produits = from p in Produits
               orderby p.Nom
               select p;
```

Syntaxe Lambda :

```
var produits = Produits.OrderBy(p => p.Nom);
var produitsDesc = Produits.OrderByDescending(p => p.Prix);
```

Agrégation

```
// Compter
int nombre = Produits.Count();

// Somme
decimal total = Produits.Sum(p => p.Prix);

// Minimum
decimal min = Produits.Min(p => p.Prix);

// Maximum
decimal max = Produits.Max(p => p.Prix);

// Moyenne
decimal moyenne = Produits.Average(p => p.Prix);
```

Groupement : GroupBy

```
var groupes = from p in Produits
              group p by p.CategorieId into g
              select new {
                  CategorieId = g.Key,
                  Produits = g
              };
```

Quantification : Any, All, Contains

```
// V rifier si au moins un    lment    existe
bool existe = Produits.Any(p => p.Prix > 100);

// V rifier si tous les    lments    satisfont une condition
bool tous = Produits.All(p => p.Prix > 0);

// V rifier si un    lment    est contenu
bool contient = Produits.Any(p => p.Nom.Contains("Ordinateur"));
```

Migrations Entity Framework

Introduction aux migrations

Les migrations permettent de :

— Créer et mettre à jour le schéma de la base de données

— Suivre les changements du modèle de données

— Appliquer les modifications de manière contrôlée

Commandes de migration

```
// Cr er une migration
dotnet ef migrations add NomMigration

// Appliquer les migrations
dotnet ef database update

// Supprimer la derni re migration
dotnet ef migrations remove
```

ASP.NET Core Identity

Introduction

ASP.NET Core Identity est un système d'authentification et d'autorisation intégré à ASP.NET Core qui permet de :

— Gérer les utilisateurs et leurs rôles

— Gérer l'authentification (connexion/déconnexion)

— Gérer l'autorisation (accès aux ressources)

— Gérer les mots de passe (hachage, réinitialisation)

Concepts Clés à Retenir

Architecture .NET

— **CLR** : Machine virtuelle pour l'exécution

— **BCL/FCL** : Bibliothèque de classes de base

— **Assemblies** : Fichiers DLL contenant les classes

ASP.NET MVC

— **Modèle** : Données et logique métier
— **Vue** : Interface utilisateur (HTML/CSS)
— **Contrôleur** : Intermédiaire entre Modèle et Vue

Entity Framework Core

— **ORM** : Mapping Objet-Relationnel
— **DbContext** : Classe de contexte pour l'accès aux données
— **DbSet** : Représentation d'une table
— **Approches** : Database First, Code First, Model First

LINQ to Entities

— **Syntaxe de requête** : `from...where...select`
— **Syntaxe Lambda** : Méthodes d'extension (`Where`, `Select`, etc.)
— **Opérateurs** : Projection, Filtrage, Tri, Jointures, Agrégation

Conclusion

Le framework .NET offre une plateforme complète pour le développement d'applications web modernes. La combinaison d'ASP.NET Core MVC, Entity Framework Core et LINQ permet de créer des applications robustes, maintenables et performantes avec une séparation claire des responsabilités grâce au pattern MVC.