DEVELOPPEZ UNE APPLICATION ASP NET CORE AVEC LE MODELE MVC

PARTIE 1 : Comprenez le fonctionnement d'ASP.NET Core MVC

1. Découvrez .NET MVC

✅ Qu'est-ce que .NET Core MVC ?

- **MVC** = **Model - View - Controller**, un **design pattern** utilisé pour **organiser le code** :
  - **Model (Modèle)** : représente les **données** et la **logique métier**.
  - **View (Vue)** : gère l'**affichage à l'utilisateur** (HTML, CSS...).
  - **Controller (Contrôleur)** : gère les **interactions utilisateur** (clics, requêtes...) et **coordonne Model ↔ View**.
- **ASP.NET MVC** est un **framework Web** de **Microsoft** basé sur le pattern MVC, utilisé pour développer des applications **web performantes** avec **ASP.NET**.
- **ASP.NET Core MVC** est l'**évolution moderne** :
  - Multiplateforme (**Windows, Linux, macOS**).
  - Permet de créer **des applications Web, APIs RESTful, etc.**
  - Plus rapide, plus léger et **plus adapté aux projets modernes**.

✅ Pourquoi .NET Core MVC est plus qu'un simple pattern ?

- C'est un **framework complet**, pas juste un modèle de structure de code.
- Il fait partie de l'**écosystème .NET** moderne.
- Il **remplace les anciens WebForms** (jugés lents et dépassés).
- Il est **flexible**, **performant** et **adapté à l'architecture moderne** (REST API, microservices, etc.).

✅ Pourquoi apprendre .NET Core MVC ?

- .NET Core MVC est **au cœur du développement Web en .NET**.
- Très **demandé sur le marché de l'emploi**, surtout avec **C#**.
- Permet de **devenir un développeur .NET compétitif**.
- Utile pour construire **des applications robustes, évolutives et modernes**.

✅ Préparation au développement .NET MVC :

1. **Volonté d'apprendre sérieusement le framework**.
2. **Installer Visual Studio** (version **Community gratuite**).
3. **Configurer Visual Studio pour le développement ASP.NET Core MVC**.

✅ Premier projet : Watchlist App

- **But de l'application :**
  - Lister et noter les films que tu as vus ou que tu possèdes.
  - Système de **notation (1 à 5 étoiles)**.
  - **Stockage des données dans une base relationnelle** (probablement via Entité Framework).
  - Possibilité de **rechercher, trier et filtrer** les films.

✅ Et ensuite ?

- Créer l'application Watchlist.
- Configurer un **compte Azure** pour **publier l'application en ligne**.
- Tu vas apprendre **pas à pas** à construire une **application MVC réelle**.

📌 Pour bien répondre au quiz :

- Retiens bien que **.NET MVC est un framework complet** basé sur **le pattern MVC**, intégré dans **.NET Core**, et **essentiel dans le développement moderne**.
- Ne confonds pas **pattern MVC** (le concept) avec **ASP.NET MVC** (le framework).
- Souviens-toi que **Visual Studio + volonté d'apprendre** suffisent pour démarrer.

2. Exécutez votre premier projet .NET MVC

✅ Créer un compte Azure : https://azure.microsoft.com/

✅ Créez un nouveau projet ASP.NET Core dans Visual Studio

1. *Lancement de Visual Studio*

- Ouvrir **Visual Studio** (menu Démarrer, icône sur le bureau ou barre rapide).
- Cliquer sur **"Créer un projet"**.

2. *Choix du type de projet*

- Dans la fenêtre suivante :
  - Filtrer avec :
    - **Langage : C#**
    - **Plateforme : Tous**
    - **Type de projet : Web**
- Sélectionner **"Application Web ASP.NET Core"**, puis cliquer sur **Suivant**.

3. *Configuration du projet*

- Donner un **nom au projet** : `Watchlist`.
- Choisir **l'emplacement du projet**, puis cliquer sur **Suivant**.

4. *Authentification sécurisée*

- Dans les paramètres d'authentification :

- Choisir **"Comptes individuels"**.
- Sélectionner **"Stocker les comptes dans l'application"**, puis cliquer sur **Créer**.

✓□ *Résultat :*

- Un projet de base est généré avec **ASP.NET Core Identity** intégré.
- Il gère l'**inscription**, la **connexion** et la **déconnexion** des utilisateurs.

✓ Initialiser la base de données

✚ *Migration Code First*

- Ouvrir **Console du Gestionnaire de Package** :
  - Menu : **Affichage > Autres fenêtres > Console du Gestionnaire de package**.
- Taper la commande suivante :

```
update-database
```

☞ Cela crée la base de données à partir du code généré automatiquement.

✓ Tester l'inscription et la connexion

- Lancer l'application : **F5** ou bouton **IIS Express**.
- Cliquer sur **Register** pour s'inscrire avec un email + mot de passe.

⚠□ Règles du mot de passe :

- Min. 6 caractères
- 1 chiffre
- 1 majuscule
- 1 minuscule
- 1 caractère spécial

✓ Après inscription → Message d'accueil + bouton Logout

✓ Tester aussi la connexion après déconnexion

✓ Explorer l'arborescence du projet (MVC)

- **Models** → classes représentant les données
- **Views** → pages HTML (Razor Pages `.cshtml`)
- **Controllers** → logique de traitement des requêtes

🔗 Routage MVC :

```
/[Contrôleur]/[Action]/[id?]
```

Exemple :

```
https://votredomaine.com/films/create
→ FilmsController > Méthode Create > Vue Create.cshtml
```

- Dossier **Views/Shared** :
  - Contient :
    - **Layout (_Layout.cshtml)** → structure commune
    - **Partial Views** → éléments réutilisables (ex : formulaire)

✅ Entity Framework Core & ORM

- Dossier **Data** :
  - **ApplicationDbContext.cs** :
    - Représente la base de données (ORM)
    - Hérite de `IdentityDbContext`
  - **Migrations/** :
    - Contient le code pour créer et gérer la base via EF Core

➡ Le **mapping objet-relationnel (ORM)** permet de manipuler les données SQL via des objets C# (classes).

✅ Bilan du chapitre

✓ Compte Azure configuré
✓ Application ASP.NET Core créée
✓ Authentification sécurisée mise en place
✓ Base de données initialisée
✓ Fonctionnalités Inscription / Connexion testées avec succès

3. QUIZ .NET MVC

1. **Vrai ou faux ? L'édition Professional ou Entreprise de Visual Studio est obligatoire pour créer des applications web MVC prêtes pour la production.**

❌ Faux.

➡ **L'édition Community de Visual Studio** (gratuite) suffit largement pour **créer des applications web MVC prêtes pour la production**.
Elle offre toutes les fonctionnalités nécessaires pour développer, tester et déployer des applications ASP.NET Core MVC.

✅ Les éditions **Professional** ou **Enterprise** apportent des fonctionnalités supplémentaires (tests avancés, outils d'architecture, collaboration en équipe à grande échelle…), mais **elles ne sont pas obligatoires** pour un projet MVC complet et professionnel.

2. **Vrai ou faux ? MVC et .NET Core ne sont rien d'autre que des noms plus sophistiqués donnés au langage de programmation C#.**

❌ **Faux.**

➞ **MVC** (Model-View-Controller) est un **patron de conception (design pattern)** utilisé pour organiser le code d'une application en trois parties distinctes :

- **Model** (les données),
- **View** (l'interface utilisateur),
- **Controller** (la logique de traitement).

➞ **.NET Core** (ou maintenant **.NET 6/7/8**) est un **framework de développement multiplateforme** (pour créer des applications web, desktop, mobiles, etc.).

➞ **C#**, en revanche, est **le langage de programmation** utilisé **au sein du framework .NET**.

☞ Donc, **MVC** et **.NET Core ne sont pas des synonymes de C#**, mais plutôt **des outils et structures** dans lesquels on utilise le **langage C#** pour construire des applications.

3. **Vrai ou faux ? Visual Studio est un outil de développement réservé au Web, car toutes les applications .NET sont des applications web.**

❌ **Faux.**

➞ **Visual Studio** est un **environnement de développement intégré (IDE)** complet, qui permet de développer **tous types d'applications**, pas seulement des applications web.

Avec Visual Studio, tu peux créer :

- ✅ des **applications web** (ASP.NET, Blazor, etc.),
- ✅ des **applications desktop** (Windows Forms, WPF),
- ✅ des **applications mobiles** (avec .NET MAUI ou Xamarin),
- ✅ des **applications console**,
- ✅ des **API REST**, des **services cloud**, des **jeux vidéo** (avec Unity), etc.

➞ Et **toutes les applications .NET ne sont pas des applications web**. Le framework **.NET** est **polyvalent**, et il est utilisé dans plusieurs domaines.

☞ Donc non, Visual Studio n'est **pas réservé au web**, et **.NET n'est pas uniquement pour les applications web**.

4. **Vrai ou faux ? Une migration code first est le processus que Visual Studio utilise pour générer le code de base d'une application MVC.**

❌ **Faux.**

☞ Une **migration code first** n'est **pas le processus qui génère le code de base d'une application MVC**, mais plutôt :

➞ **Un processus utilisé pour créer ou mettre à jour une base de données à partir du code C#** (c'est-à-dire les classes modèles).

✅ **En d'autres termes :**

- **Le code de base d'une application MVC** (les dossiers *Models*, *Views*, *Controllers*, etc.) est **généré lors de la création du projet dans Visual Studio**.
- **La migration code first**, elle, **sert à créer la structure de la base de données (tables, colonnes, relations)** à partir du code (modèles C#) grâce à **Entity Framework Core**.

📌 Exemple : Tu crées une classe `Film` dans ton dossier *Models*. Pour créer une table `Films` dans la base de données, tu fais une **migration code first**, puis tu exécutes la commande :

```
update-database
```

5. **Parmi les éléments suivants, lesquels font partie des exigences par défaut concernant les mots de passe dans les applications ASP.NET Core MVC ?**

✓ 2. Au moins 6 caractères
✓ 3. Au moins un caractère majuscule
✓ 4. Au moins un caractère minuscule
✓ 5. Au moins un chiffre
✓ 6. Au moins un caractère spécial (non alphanumérique)

☞ Ce sont les **paramètres de sécurité par défaut** définis dans `IdentityOptions.Password` dans ASP.NET Core.

Exemple dans le code :

```
options.Password.RequireDigit = true;
options.Password.RequiredLength = 6;
options.Password.RequireNonAlphanumeric = true;
options.Password.RequireUppercase = true;
options.Password.RequireLowercase = true;
```

6. **Qu'est-ce que MVC ?**

✓ 2. Un design pattern pour le développement d'applications
✓ 3. L'acronyme de Modèle-Vue-Contrôleur
✓ 4. En ASP.NET, un framework pour la création d'applications web qui inclut le développement d'API RESTful

❌ Faux :

- **1. Un acronyme pour Microsoft Visual C/C++** → Cela n'a rien à voir avec **MVC**, c'est une confusion avec **Visual C++**.

📌 Pour rappel :

- **MVC (Model-View-Controller)** est **un modèle architectural (design pattern)** qui sépare les responsabilités :
  - **Model (Modèle)** : la logique métier et les données.
  - **View (Vue)** : l'interface utilisateur.
  - **Controller (Contrôleur)** : traite les requêtes et lie le modèle à la vue.
- ASP.NET MVC est un **framework Web de Microsoft** basé sur ce pattern.

7. **Qu'est-ce qu'une classe de contexte de base de données ?**

✓ 1. Une classe qui représente le schéma d'une base de données et permet ainsi à un développeur d'interagir avec celle-ci.

✓ **2. Une classe utilisée pour interroger une base de données et/ou écrire dans celle-ci.**
✓ **3. Une classe utilisée pour établir une connexion à une base de données.**

❌ Faux :

- **4. Une classe inutile si vous utilisez un ORM → ❌ FAUX. La classe de contexte est justement essentielle lorsqu'on utilise un ORM comme Entity Framework.**

📌 En résumé : **La** classe de contexte (DbContext) **dans** Entity Framework Core **:**

- **Sert de** passerelle entre votre application et la base de données**.**
- **Permet de** configurer les entités**,** effectuer des requêtes**,** insérer**,** mettre à jour**,** supprimer **des données.**
- **C'est** le cœur d'EF Core **!**

□ Exemple :

```
public class ApplicationDbContext : DbContext
{
    public DbSet<Employee> Employees { get; set; }

    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }
}
```

8. **La classe IdentityDbContext est :**

✓ **2. La classe dont doivent hériter toutes les classes de contexte de base de données dans EF Core si elles utilisent ASP.NET Identity et/ou des rôles.**

📌 Explications :

- `IdentityDbContext` est une **classe spéciale fournie par ASP.NET Core Identity**, qui **hérite de** `DbContext`, et ajoute **les entités nécessaires à la gestion des utilisateurs, rôles, jetons, logins externes, etc.**
- Elle permet de **gérer facilement les utilisateurs, rôles, mots de passe, connexions sociales, etc.** dans les applications ASP.NET MVC / Razor Pages.

Exemple :

```
public class ApplicationDbContext : IdentityDbContext
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }

    public DbSet<Product> Products { get; set; }
}
```

Ici, en plus des tables liées à l'authentification (`AspNetUsers`, `AspNetRoles`, etc.), tu peux rajouter d'autres entités comme `Products`.

❌ Les autres réponses sont incorrectes :

- **La classe dont doivent hériter toutes les classes de contexte de base de données dans EF Core. ❌ → Pas toutes les classes DbContext, seulement celles** utilisant ASP.NET Identity**.**

- **La classe qui représente la table Users dans la base de données. ❌ → `IdentityDbContext` ne représente** pas seulement la table Users**, mais** tout le système d'authentification/autorisation**.**
- **La classe qui représente toutes les bibliothèques ASP.NET Identity. ❌ → Ce n'est** pas une bibliothèque**, mais une** classe spécifique à EF Core**.**

9. **Qu'est-ce qu'un ORM ?**

✓ 2. Un outil qui gère le mappage entre un ensemble d'objets et une base de données relationnelle.
✓ 3. Un outil qui convertit des données entre des types incompatibles à l'aide de langages de programmation orientés objet, en créant une « base de données objet virtuelle » qui peut être utilisée dans le langage de programmation.

📌 Explication :

- **ORM** signifie **Object-Relational Mapping** (*Mappage Objet-Relationnel*).
- C'est un **outil** ou une **bibliothèque** qui permet aux développeurs de **travailler avec une base de données en utilisant des objets de leur langage de programmation orienté objet**, sans écrire directement du SQL.

En ASP.NET Core, l'ORM le plus courant est Entity Framework Core (EF Core).

☞ Par exemple : Tu peux manipuler des objets `User`, `Product`, `Order` dans ton code, et EF Core s'occupe de les **traduire en requêtes SQL pour lire/écrire dans la base de données**.

❌ Les mauvaises réponses :

1. L'utilitaire d'ASP.NET pour la gestion de la réputation en ligne. Faux **→ Ce n'est** pas lié à la gestion de réputation en ligne**.**

4. Un outil utilisé pour faire correspondre des classes JavaScript à des modèles de données C#. Faux **→ Un ORM** n'a rien à voir avec JavaScript ou le mappage entre JavaScript et C#**.**

10. **Sélectionnez l'ordre qui se rapproche le plus du processus d'exécution d'une migration code first :**

✓ Bonne réponse :
4. Dans la console du Gestionnaire de package : `Add-Migration name-of-migration` (appuyez sur Entrée), puis `Update-Database` (appuyez sur Entrée).

📌 Explication :

Voici le **processus standard d'une migration Code First dans Entity Framework Core** :

1. **`Add-Migration NomMigration`**
   → Crée un fichier de migration contenant les modifications détectées dans tes classes du modèle (entités).
2. **`Update-Database`**
   → Applique cette migration à la base de données (c'est-à-dire qu'il exécute les commandes SQL correspondantes pour modifier ou créer les tables).

❌ Pourquoi les autres sont incorrectes ?

1. **Dans la console du Gestionnaire de package : `Add-Migration` (appuyez sur `Entrée`), puis `Update-Database` (appuyez sur `Entrée`). Presque correcte, mais il** manque le nom de la migration**, ce qui est** obligatoire **dans `Add-Migration`.**

2. **Dans la console du Gestionnaire de package : `Update-Database -automaticmigration`. La commande `Update-Database -automaticmigration` n'existe pas dans EF Core (elle faisait partie de l'ancien EF6 avec des migrations automatiques).**
3. **Dans la console du Gestionnaire de package : `Update-Database migration=0` (appuyez sur `Entrée`). `Update-Database migration=0` → Ce n'est pas une commande valide pour créer ou appliquer une migration.**