TP : Gestion des Clients et Commandes avec Validation - ASP.NET Core

Objectifs

- Concevoir une API REST avec **ASP.NET Core** et **Entity Framework Core**.
- Manipuler une base de données **SQL Server** avec **Entity Framework Core**.
- Comprendre et gérer une relation **OneToMany** entre entités.
- Valider les entrées avec **Data Annotations** et **FluentValidation**.

Contexte

Vous devez développer une application de gestion des clients et de leurs commandes. Un **client** peut passer plusieurs **commandes**, mais une **commande** appartient à un seul **client**.

L'application devra permettre :

- D'ajouter, lister et récupérer un **client**.
- D'ajouter, lister et récupérer une **commande**.
- D'associer une commande à un client.
- **Valider les données entrées** lors de la création d'un client ou d'une commande.

Questions

1. Modélisation des entités

- Créez les classes `Client` et `Commande` avec la relation OneToMany appropriée.
- Quelles propriétés de navigation devez-vous ajouter ?

Modèle Client

```
public class Client
{
    public int Id { get; set; }
    public string Nom { get; set; }
    public string Prenom { get; set; }
    public string Email { get; set; }
    public string Telephone { get; set; }
    public string Adresse { get; set; }
    public DateTime DateCreation { get; set; }
    public ICollection<Commande> Commandes { get; set; } = new List<Commande>();
}
```

Modèle Commande

```
public class Commande
{
    public int Id { get; set; }
    public string NumeroCommande { get; set; }
    public DateTime DateCommande { get; set; }
    public decimal MontantTotal { get; set; }
    public string Statut { get; set; }
    public int ClientId { get; set; }
    public Client Client { get; set; }
}
```

2. Configuration d'Entity Framework Core

- Configurez le DbContext pour gérer les entités `Client` et `Commande`.
- Créez la migration initiale et mettez à jour la base de données SQL Server.

3. Validation des données

- Ajoutez les validations nécessaires sur les propriétés des classes (e.g., Required, StringLength, Range).
- Utilisez FluentValidation pour créer des validateurs personnalisés pour `Client` et `Commande`.

4. Contrôleurs et endpoints

- Créez un `ClientController` avec les endpoints : GET (tous/un), POST, PUT, DELETE.
- Créez un `CommandeController` avec les endpoints : GET (tous/un), POST, PUT, DELETE.
- Comment allez-vous associer une commande à un client via l'API ?

5. Gestion des erreurs

- Comment gérez-vous les erreurs de validation dans vos contrôleurs ?
- Créez un middleware ou un filtre d'exception personnalisé.

6. Relations et cascade

- Configurez le comportement de suppression en cascade (quand un client est supprimé, ses commandes le sont aussi).
- Testez ce comportement.

7. Tests

- Écrivez des tests unitaires pour les services métier.
- Écrivez des tests d'intégration pour les endpoints API.