

TP N°3

Accès à une base de données SQL SERVER
Avec Entity Framework Core (Approche Code First)
GESTION DES ETUDIANTS

Enseignant : Malek Zribi

1

PROJET BASÉ SUR L'APPROCHE CODE FIRST

- Ouvrez Visual Studio 2022 et cliquez sur **Créer un nouveau projet** , de type **Application web ASP.Net Core (Modèle-Vue-Contrôleur)**.
- Sous Visual Studio, Click droit sur le nom du projet → Gérer les packages Nuget
- Installer les packages NuGet suivants pour utiliser EF Core dans votre application :

The screenshot displays the Visual Studio NuGet Package Manager interface. On the left, a list of search results is shown, with three packages highlighted by red boxes: **Microsoft.EntityFrameworkCore.SqlServer** (version 6.0.3), **Microsoft.EntityFrameworkCore** (version 6.0.3), and **Microsoft.EntityFrameworkCore.Tools** (version 6.0.3). The right pane shows the details for **Microsoft.EntityFrameworkCore.SqlServer**, including its version (6.0.3), author (Microsoft), license (MIT), and publication date (mardi 8 mars 2022).

Gestionnaire de package NuGet : TP2

Source de package : nuget.org

Recherche (Ctrl+L) ☐ Inclure la version préliminaire

| Package | Version |
|---|---------|
| ★ Microsoft.EntityFrameworkCore.SqlServer par Microsoft Microsoft SQL Server database provider for Entity Framework Core. | 6.0.3 |
| ★ Microsoft.EntityFrameworkCore par Microsoft Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Azure Cosmos DB, MySQL, PostgreSQL, and other databases through a provider plugin API. | 6.0.3 |
| ★ Newtonsoft.Json par James Newton-King Json.NET is a popular high-performance JSON framework for .NET | 13.0.1 |
| ★ Microsoft.EntityFrameworkCore.Tools par Microsoft Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio. | 6.0.3 |

Microsoft.EntityFrameworkCore.SqlServer nuget.org

Version : Dernière version stable 6.0.3

Options

Description
Microsoft SQL Server database provider for Entity Framework Core.

Version : 6.0.3
Auteur(s) : Microsoft
Licence : MIT
Date de publication : mardi 8 mars 2022 (08/03/2022)

CRÉATION DES CLASSES DU DOMAINE

- Dans le cadre du développement d'une application web ASP.NET Core MVC avec C# pour gérer les étudiants et leurs institutions universitaires, on vous donne les classes suivantes :

```
public class School {  
    public int SchoolID { get; set; }  
    public string SchoolName { get; set; }  
    public string SchoolAdress { get; set; }  
    public ICollection<Student> Students { get; set; }  
}
```

```
public class Student{  
    public int StudentId { get; set; }  
    [Required]  
    public string StudentName { get; set; }  
    [Range(1,100)]  
    public int Age { get; set; }  
    [DataType(DataType.Date)]  
    public DateTime BirthDate { get; set; }  
    public int SchoolID { get; set; }  
    public School School { get; set; }  
}
```

- L'application web est structurée selon l'architecture MVC, ajouter les deux classes dans le dossier **Models**.

CRÉATION DE LA CLASSE DE CONTEXTE

- Ajouter les packages **EF Core** puis créer la classe de contexte **StudentContext** :

```
public class StudentContext : DbContext
{
    public StudentContext(DbContextOptions<StudentContext> options) : base(options)
    {
    }
    public DbSet<Student> Students { get; set; }
    public DbSet<School> Schools { get; set; }
}
```

- La chaîne de connexion à la base de données à ajouter dans le fichier de configuration **appsettings.json** est la suivante :

```
"ConnectionStrings": {
    "StudentDBConnection": "server=(localdb)\\MSSQLLocalDB;database=SchoolDB;
Trusted_Connection=true"
}
```

CRÉATION DES INTERFACES

- Créer un dossier **Repositories** et créer les interfaces **ISchoolRepository** et **IStudentRepository** avec les méthodes nécessaires :

```
public interface ISchoolRepository
{
    IList<School> GetAll();
    School GetById(int id);
    void Add(School s);
    void Edit(School s);
    void Delete(School s);
    double StudentAgeAverage(int schoolId);
    int StudentCount(int schoolId);
}
```

```
public interface IStudentRepository
{
    IList<Student> GetAll();
    Student GetById(int id);
    void Add(Student s);
    void Edit(Student s);
    void Delete(Student s);
    IList<Student> GetStudentsBySchoolID(int? schoolId);
    IList<Student> FindByName(string name);
}
```

CRÉATION DES CLASSES DE SERVICE

- Créer maintenant les classes de services **SchoolRepository** et **StudentRepository** en utilisant le langage de requêtes Linq To Entity pour l'accès à la base de données SQLSERVER dans le code des méthodes.

```
public class SchoolRepository : ISchoolRepository
{
    readonly StudentContext context;
    public SchoolRepository(StudentContext context) {
        this.context = context;
    }
    public IList<School> GetAll() {
        return context.Schools.OrderBy(s => s.SchoolName).ToList();
    }
    public School GetById(int id) {
        return context.Schools.Find(id);
    }
    public void Add(School s) {
        context.Schools.Add(s);
        context.SaveChanges();
    }
    public void Edit(School s) {
        School s1 = context.Schools.Find(s.SchoolID);
        if (s1 != null)
        {
            s1.SchoolName = s.SchoolName;
            s1.SchoolAddress = s.SchoolAddress;
            context.SaveChanges();
        }
    }
}
```

```
public void Delete(School s)
{
    School s1 = context.Schools.Find(s.SchoolID);
    if (s1 != null)
    {
        context.Schools.Remove(s1);
        context.SaveChanges();
    }
}
public double StudentAgeAverage(int schoolId)
{
    if (StudentCount(schoolId) == 0)
        return 0;
    else
        return context.Students.Where(s => s.SchoolID ==
            schoolId).Average(e => e.Age);
}
public int StudentCount(int schoolId)
{
    return context.Students.Where(s => s.SchoolID ==
        schoolId).Count();
}
}
```

CRÉATION DES CLASSES DE SERVICE

- Créer maintenant les classes de services **SchoolRepository** et **StudentRepository** en utilisant le langage de requêtes Linq To Entity pour l'accès à la base de données SQLSERVER dans le code des méthodes.

```
public class StudentRepository : IStudentRepository
{
    readonly StudentContext context;
    public StudentRepository(StudentContext context) {
        this.context = context;
    }
    public IList<Student> GetAll() {
        return context.Students.OrderBy(x => x.StudentName).Include(x
=> x.School).ToList();
    }
    public Student GetById(int id) {
        return context.Students.Where(x => x.StudentId ==
id).Include(x => x.School).SingleOrDefault();
    }
    public void Add(Student s) {
        context.Students.Add(s);
        context.SaveChanges();
    }
    public void Edit(Student s) {
        Student s1 = context.Students.Find(s.StudentId);
        if (s1 != null) {
            s1.StudentName = s.StudentName;
            s1.Age = s.Age;
            s1.BirthDate = s.BirthDate;
            s1.SchoolID = s.SchoolID;
            context.SaveChanges();
        }
    }
}
```

```
public void Delete(Student s)
{
    Student s1 = context.Students.Find(s.StudentId);
    if (s1 != null)
    {
        context.Students.Remove(s1);
        context.SaveChanges();
    }
}
public IList<Student> GetStudentsBySchoolID(int? schoolId)
{
    return context.Students.Where(s =>
s.SchoolID.Equals(schoolId))
.OrderBy(s => s.StudentName)
.Include(std => std.School).ToList();
}
public IList<Student> FindByName(string name)
{
    return context.Students.Where(s =>
s.StudentName.Contains(name)).Include(std =>
std.School).ToList();
}
}
```


INJECTION DES DÉPENDANCES ET CRÉATION DE LA BD

- Dans le fichier Program.cs, inscrire notre classe StudentContext spécifique à l'application et ajouter les injections de dépendances nécessaires pour les classes **SchoolRepository** et **StudentRepository** :

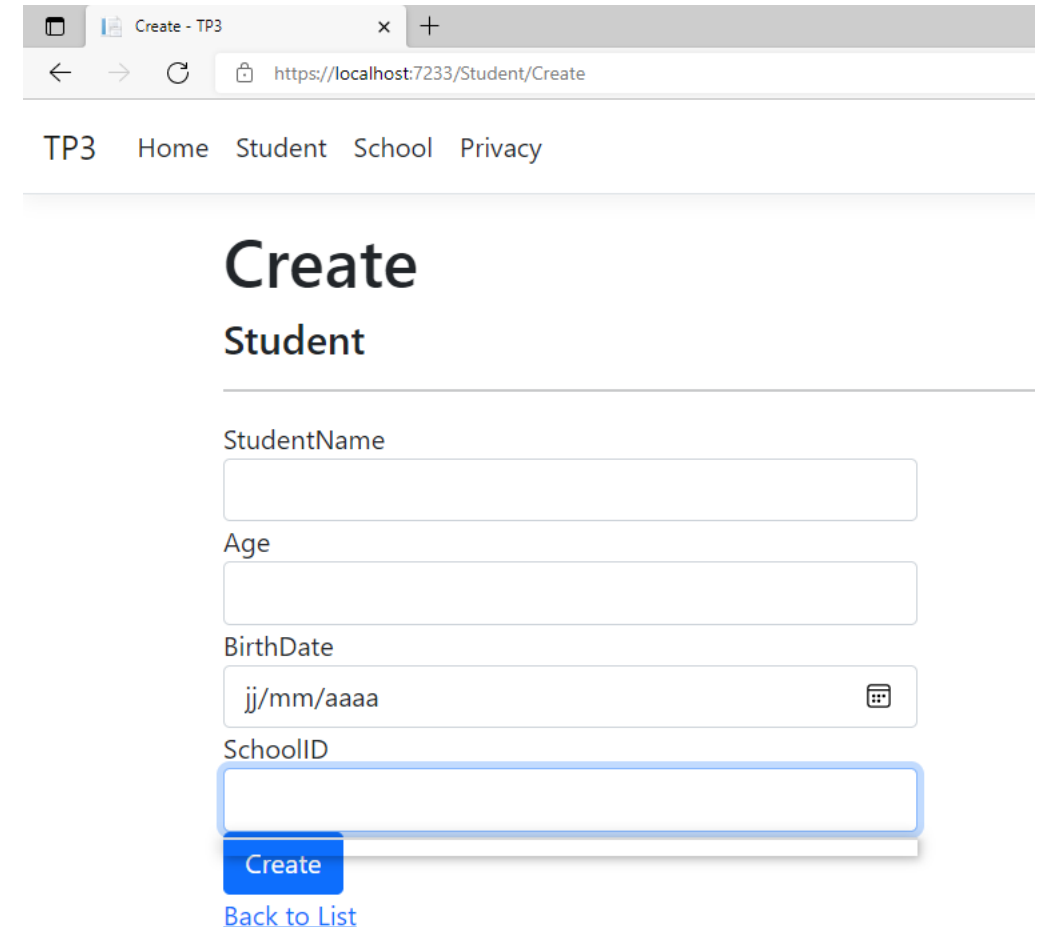
```
builder.Services.AddDbContextPool<StudentContext>(options =>  
options.UseSqlServer(builder.Configuration.GetConnectionString("StudentDBConnection")));  
builder.Services.AddScoped<ISchoolRepository, SchoolRepository>();  
builder.Services.AddScoped<IStudentRepository, StudentRepository>();
```

- Créer maintenant votre base de données avec une migration Entity Framework Core.
- Créer et compléter le code des méthodes d'actions des contrôleurs StudentController et SchoolController
- Générer les vues pour les méthodes d'action des deux contrôleurs.

EXÉCUTION

- Une fois les vues créées et on exécute notre projet, on va remarquer que lorsqu'on désire ajouter un étudiant, le champ **SchoolId** est représenté sous forme d'une liste déroulante vide créée sous forme d'un ViewBag nommé : "ViewBag.SchoolID".
- On va alors la relier à la base de données et précisément au champ **NameSchool** de la table **School**.
- Pour cela, ajouter cette instruction dans l'action Get et Post de l'action **Create** et **Edit** du contrôleur **StudentController** :

```
ViewBag.SchoolID = new  
SelectList(schoolrepository.GetAll(), "SchoolID",  
"SchoolName");
```



The screenshot shows a web browser window with the URL `https://localhost:7233/Student/Create`. The page title is "Create Student". The form contains the following fields:

- StudentName**: A text input field.
- Age**: A text input field.
- BirthDate**: A date picker field showing "jj/mm/aaaa".
- SchoolID**: A dropdown menu with a blue border.

At the bottom of the form, there is a blue "Create" button and a "Back to List" link.

AJOUTER DES FILTRES

- On désire ajouter dans la page Index de l'étudiant deux zones de recherche :
 - Une zone de texte pour pouvoir rechercher les étudiants par nom ;
 - Une liste déroulante pour pouvoir rechercher les étudiants par établissement, remplie par le contenu de la table **School**.

TP3 Home Student School Privacy

Index

[Create New](#)

FSEG Recherche

StudentName FSEG ISET School

- On va ajouter le code suivant dans la page **Index** du **Student** :

```
<form class="form-inline my-2 my-lg-0" asp-action="Search" asp-controller="Student">
  <p><input size="30" placeholder="tapez un Nom" type="search" name="name" />
  <select asp-items="ViewBag.SchoolID" name="schoolid" size=1.5 style="width:300px ; height:30px" ></select>
  <button class="btn btn-outline-success my-2 my-sm-0" type="submit">Recherche</button>
</p>
</form>
```

- N'oubliez pas de charger le ViewBag dans l'action Index du contrôleur.

AJOUTER DES FILTRES

- Dans le contrôleur StudentController, ajouter l'action search avec le code suivant :

```
public ActionResult Search(string name, int? schoolid)
{
    var result = studentrepository.GetAll();
    if(!string.IsNullOrEmpty(name))
        result = studentrepository.FindByName(name);
    else
        if (schoolid != null)
            result = studentrepository.GetStudentsBySchoolID(schoolid);
    ViewBag.SchoolID = new SelectList(schoolrepository.GetAll(), "SchoolID", "SchoolName");
    return View("Index", result);
}
```

- Pour rafraîchir la liste et afficher tous les étudiants, ajouter cette ligne dans la page Index :

```
<a asp-action="Index" asp-controller="Student"> All Students</a>
```

AJOUTER DES FILTRES

- Pour rafraîchir la liste et afficher tous les étudiants, ajouter cette ligne dans la page Index :

```
<form class="form-inline my-2 my-lg-0" asp-action="Search" asp-controller="Student">
  <p><input size="30" placeholder="tapez un Nom" type="search" name="name" />
  <select asp-items="ViewBag.SchoolID" name="schoolid" size=1.5 style="width:300px ;
height:30px" ></select>
  <button class="btn btn-outline-success my-2 my-sm-0" type="submit">Search</button>
</p>
</form>
```

- Pour rafraîchir la liste et afficher tous les étudiants, ajouter cette ligne dans la page Index :

```
<a asp-action="Index" asp-controller="Student"> All Students</a>
```