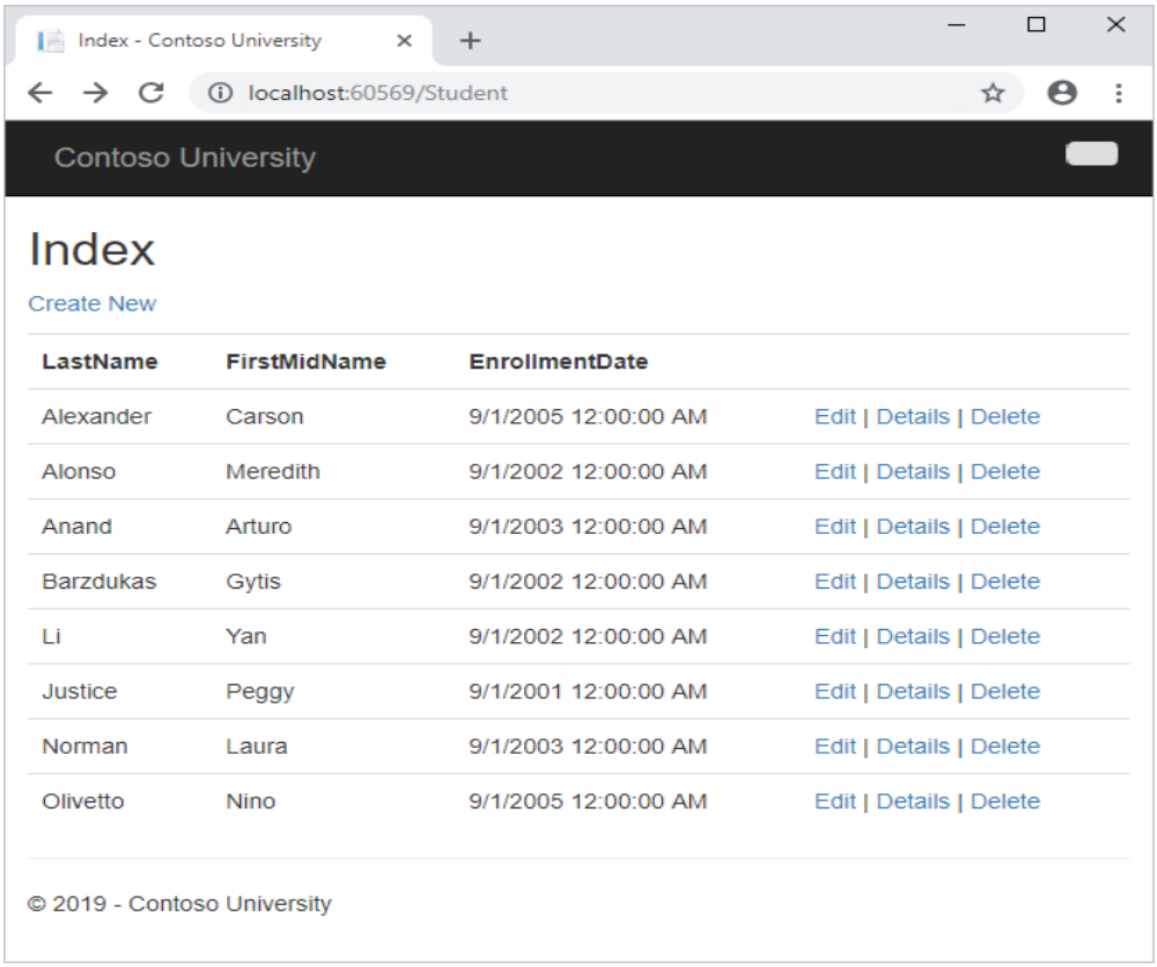
Approche « Code First » avec Entity Framework 6 en utilisant MVC 5

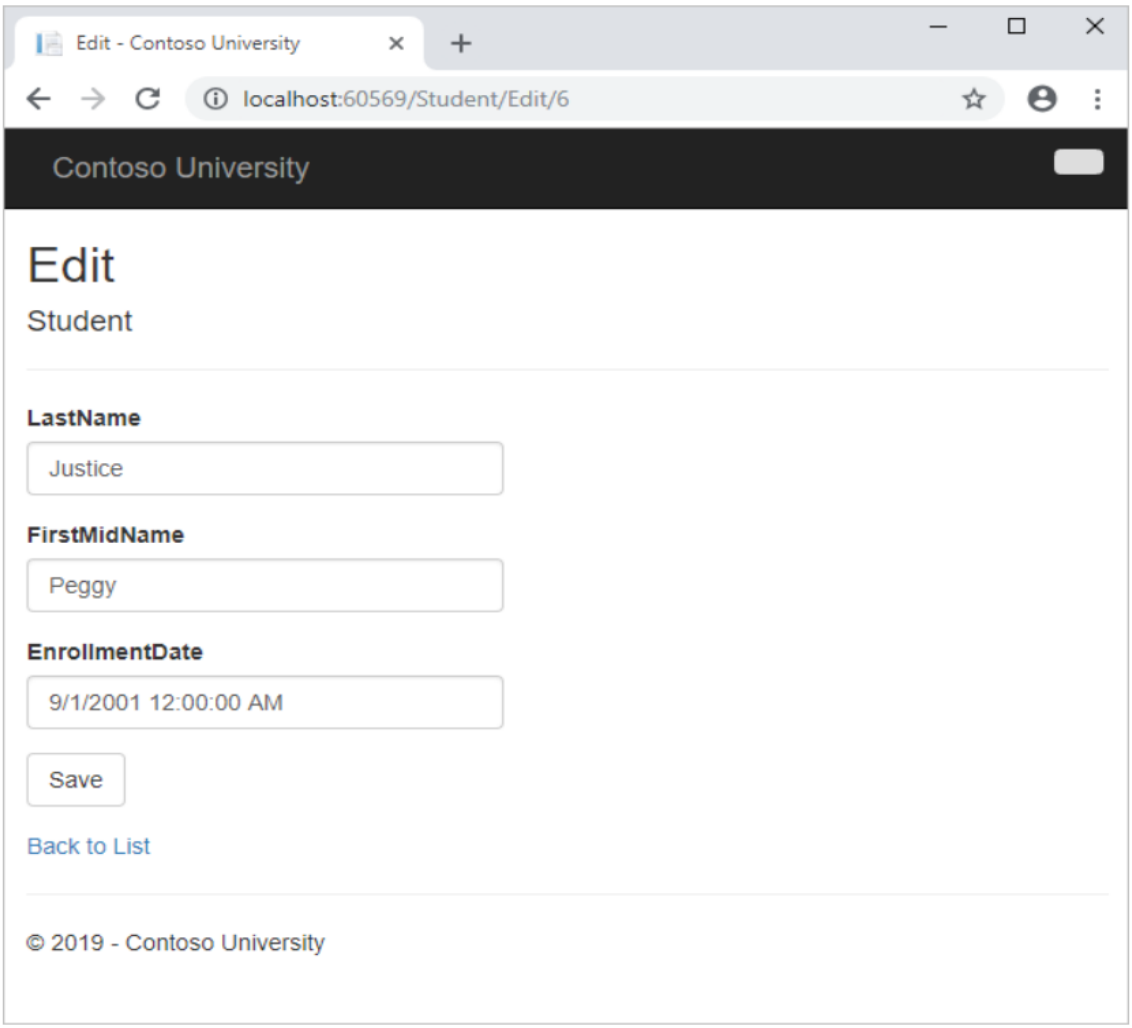
Dans ce TP, on apprend à créer une application ASP.NET MVC 5 qui utilise Entity Framework 6 pour l'accès aux données. Ce TP utilise le workflow Code First. On compte à travers ce dernier créer l'exemple d'application « Contoso University ». L'exemple d'application est un simple site Web universitaire. Avec lui, on peut afficher et mettre à jour les informations sur les étudiants, les cours et les instructeurs.

A travers ce TP on compte :

- ✓ Créer une application Web MVC
- ✓ Configurer le style du site
- ✓ Installer Entity Framework 6
- ✓ Créer le modèle de données
- ✓ Créer le contexte de la base de données
- ✓ Initialiser la base de données avec les données de test
- ✓ Configurer EF 6 pour utiliser LocalDB
- ✓ Créer un contrôleur et des vues
- ✓ Consulter la base de données

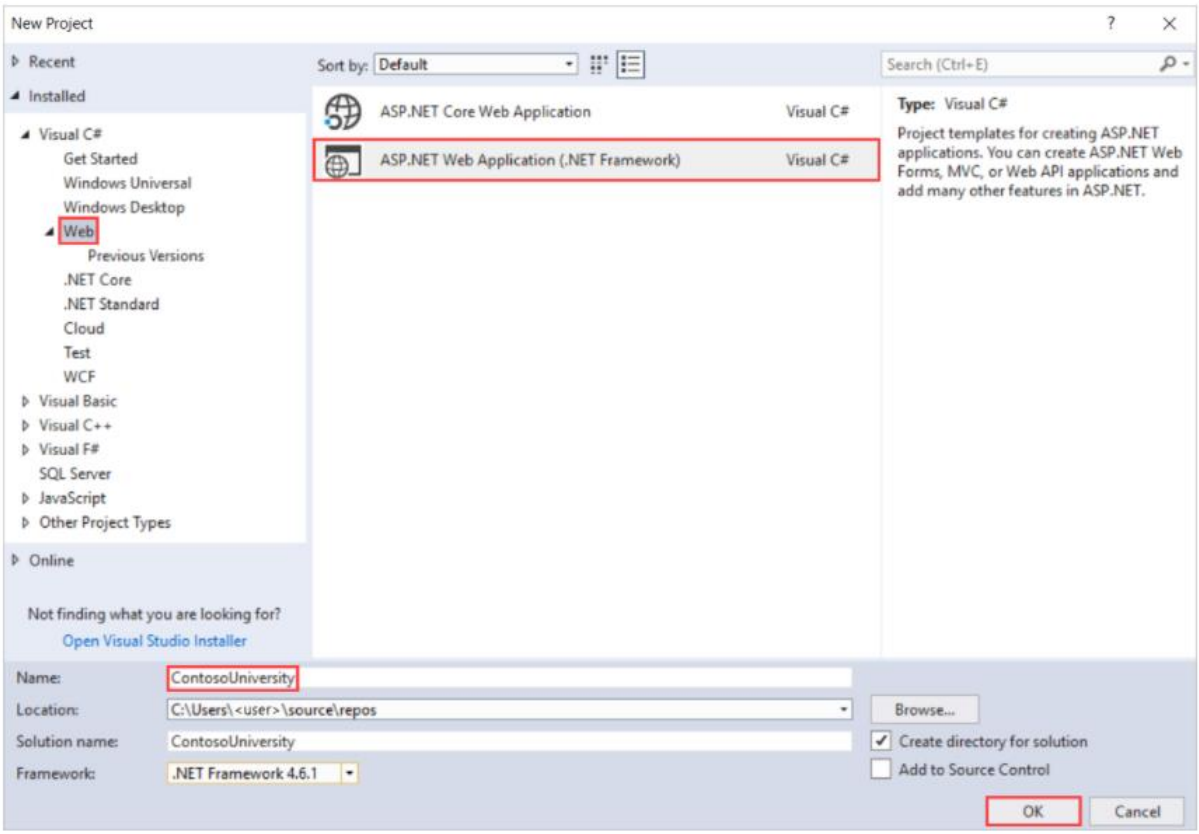
Voici deux captures d'écran de ce qu'on va créer :



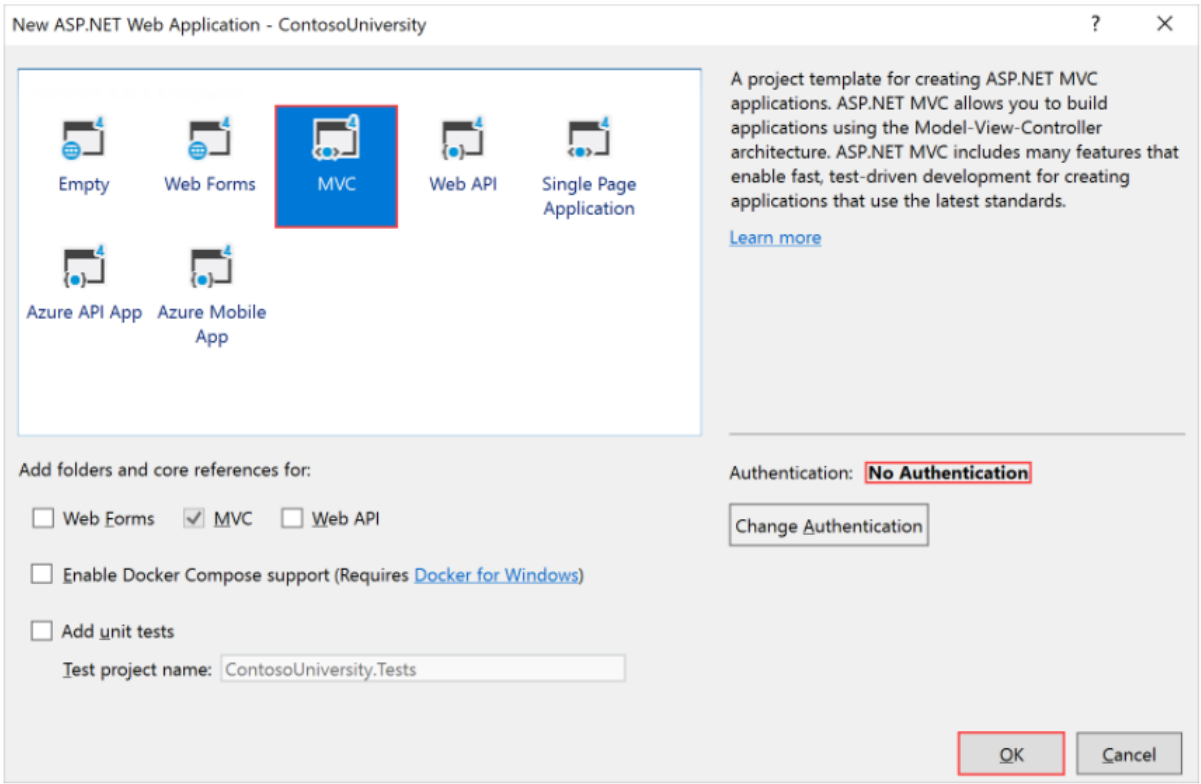


A. Créer une application Web MVC

1. Ouvrez Visual Studio et créez un projet Web C# à l'aide du modèle d'application Web ASP.NET (.NET Framework). Nommez le projet ContosoUniversity et sélectionnez OK.



2. Dans Nouvelle application Web ASP.NET - ContosoUniversity, sélectionnez MVC. Puis, Sélectionnez OK pour créer le projet.



B. Configurer le style du site

Quelques modifications simples pour configurer le menu du site, la mise en page et la page d'accueil.

1. Ouvrez *Views\Shared_Layout.cshtml* et apportez les modifications suivantes :
   - ✓ Remplacez chaque occurrence de « Mon application ASP.NET » et « Nom de l'application » par « Contoso University ».
   - ✓ Ajoutez des entrées de menu pour les étudiants, les cours, Contact et About.

   Les modifications sont mises en évidence dans l'extrait de code suivant :

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - Contoso University</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse"
data-target=".navbar-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                @Html.ActionLink("Contoso University", "Index", "Movie", new { area
= "" }, new { @class = "navbar-brand" })
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li>@Html.ActionLink("Accueil", "Index", "Home")</li>
                     <li>@Html.ActionLink("Students", "Index", "Student")</li>
                    <li>@Html.ActionLink("Courses", "Index", "Course")</li>
                    <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
                    <li>@Html.ActionLink("About", "About", "Home")</li>
                </ul>
            </div>
        </div>
    </div>



    <div class="container">
        @RenderBody()
        <hr />
        <footer>
            <p>&copy; @DateTime.Now.Year - Contoso University</p>
        </footer>
    </div>
    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/bootstrap")
    @RenderSection("scripts", required: false)
</body>
</html>
```

2. Dans *Views\Home\Index.cshtml* , remplacez le contenu du fichier par le code suivant pour remplacer le texte sur ASP.NET et MVC par du texte sur cette application :

```
@{
    ViewBag.Title = "Home Page";
}
<div class="jumbotron">
    <h1>Contoso University</h1>
</div>
<div class="row">
    <div class="col-md-4">
        <h2>Welcome to Contoso University</h2>
        <p>
            Contoso University is a sample application that
            demonstrates how to use Entity Framework 6 in an
            ASP.NET MVC 5 web application.
        </p>
    </div>
    <div class="col-md-4">
        <h2>Build it from scratch</h2>
        <p>
            You can build the application by following the steps in the tutorial
            series on the ASP.NET site.
        </p>
    </div>
</div>
```

3. Appuyez sur Ctrl+F5 pour exécuter le site Web. Vous voyez la page d'accueil avec le menu principal.

C. Installer Entity Framework 6

1. Dans le menu Outils, choisissez NuGet Package Manager , puis choisissez Package Manager Console .
2. Dans la fenêtre Package Manager Console , saisissez la commande suivante :

```
Install-Package EntityFramework
```

Cette étape peut être effectuée automatiquement à travers le NuGet package manager.

D. Créer le modèle de données

Ensuite, vous allez créer des classes d'entités pour l'application « Contoso University ». Vous commencerez avec les trois entités suivantes :

Course <-> Enrollment <-> Student

Entities	Relationship
Course to Enrollment	One-to-many
Student to Enrollment	One-to-many

Il existe une relation One-to-many entre les entités Student et Enrollment, et il existe une relation One-to-many entre les entités Course et Enrollment. En d'autres termes, un étudiant peut être inscrit à n'importe quel nombre de cours, et un cours peut avoir n'importe quel nombre d'étudiants inscrits. Passons maintenant à créer la classe pour chacune de ces entités.

- **Classe Student :**

Dans le dossier *Modèles*, créez un fichier de classe nommé *Student.cs* en cliquant avec le bouton droit sur le dossier dans l'Explorateur de solutions et en choisissant Ajouter -> Classe. Remplacez le code du modèle par le code suivant :

```
using System;
using System.Collections.Generic;

namespace ContosoUniversity.Models
{
    public class Student
    {
        public int ID { get; set; }
        public string LastName { get; set; }
        public string FirstMidName { get; set; }
        public DateTime EnrollmentDate { get; set; }

        public virtual ICollection<Enrollment> Enrollments { get; set; }
```

```
    }
}
```

La propriété ID deviendra la colonne de clé primaire de la table de base de données qui correspond à cette classe. Par défaut, Entity Framework interprète une propriété nommée ID ou *classname*ID comme clé primaire.

La propriété Enrollments est une propriété de navigation. Les propriétés de navigation contiennent d'autres entités liées à cette entité. Dans ce cas, la propriété Enrollments de l'entité Student contiendra toutes les entités Enrollment propres à cet étudiant. En d'autres termes, si une ligne étudiant possède deux inscriptions dans la base de données (lignes qui contiennent la valeur de clé primaire de cet étudiant dans leur colonne StudentID de clé étrangère), la propriété de navigation Enrollments de cet objet étudiant contiendra ces deux entités Enrollment.

Les propriétés de navigation sont généralement définies public virtual pour pouvoir tirer parti de certaines fonctionnalités d'Entity Framework telles que le lazy loading. Le lazy loading retarde le chargement des données associées, jusqu'à ce que vous le demandiez spécifiquement. Par exemple, l'entité Student contient l'entité StudentAddress. Dans le lazy loading, le contexte charge d'abord les données d'entité Student à partir de la base de données, puis il chargera l'entité StudentAddress lorsque nous accéderons à la propriété StudentAddress.

Généralement, on utilise le lazy loading si on a un objet qui est coûteux à créer, et on souhaite tarder sa création jusqu'à ce qu'il soit demandé. Pour cela, il doit être déclaré virtual.

Si une propriété de navigation peut contenir plusieurs entités (comme dans les relations One-to-many ou Many-to-many), son type doit être une liste dans laquelle des entrées peuvent être ajoutées, supprimées et mises à jour, comme ICollection.

- **Classe Enrollment :**

Dans le dossier Models, créez la classe Enrollment.cs et remplacez le code existant par le code suivant :

```
namespace ContosoUniversity.Models
{
    public enum Grade
    {
        A, B, C, D, F
    }

    public class Enrollment
    {
        public int EnrollmentID { get; set; }
        public int CourseID { get; set; }
        public int StudentID { get; set; }
        public Grade? Grade { get; set; }

        public virtual Course Course { get; set; }
        public virtual Student Student { get; set; }
    }
}
```

La propriété EnrollmentID sera la clé primaire de la table Enrollment.

La propriété Grade est une enum. Le point d'interrogation après la déclaration de type Grade indique que la propriété Grade est nullable. Un grade nul est différent d'un grade zéro — nul signifie qu'un grade n'est pas connu ou n'a pas encore été attribué.

La propriété StudentID est une clé étrangère et la propriété de navigation correspondante est Student. Une entité Enrollment est associée à une entité Student, donc la propriété ne peut contenir qu'une seule entité Student (contrairement à la propriété de navigation Student.Enrollments que vous avez vue précédemment, qui peut contenir plusieurs entités Enrollment).

La propriété CourseID est une clé étrangère et la propriété de navigation correspondante est Course. Une entité Enrollment est associée à une seule entité Course.

Entity Framework interprète une propriété comme une propriété de clé étrangère si elle est nommée <nom de propriété de navigation><nom de propriété de clé primaire> (par exemple, StudentID pour la propriété de navigation Student puisque la clé primaire de l'entité Student est ID). Les propriétés de clé étrangère peuvent également être nommées de la même manière simplement <nom de propriété de clé primaire> (par exemple, CourseID puisque la clé primaire de l' entité Course est CourseID).

- **Classe Course :**

Dans le dossier Models, créez la classe Course.cs, en remplaçant le code du modèle par le code suivant :

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class Course
    {
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int CourseID { get; set; }
        public string Title { get; set; }
        public int Credits { get; set; }
        public virtual ICollection<Enrollment> Enrollments { get; set; }
    }
}
```

La propriété Enrollments est une propriété de navigation. Une entité Course peut être liée à un nombre quelconque d'entités Enrollment.

L'attribut DatabaseGeneratedAttribute vous permet de saisir la clé primaire du cours plutôt que de la générer par la base de données.

E. Créer le contexte de la base de données

La classe principale qui coordonne la fonctionnalité Entity Framework pour un modèle de données donné est la classe de contexte de base de données. Vous créez cette classe en la dérivant de la classe System.Data.Entity.DbContext. Dans votre code, vous spécifiez quelles entités sont incluses dans le modèle de données. Vous pouvez également personnaliser certains comportements d'Entity Framework. Dans ce projet, la classe est nommée SchoolContext.

- Pour créer un dossier dans le projet ContosoUniversity, cliquez avec le bouton droit sur le projet dans l'Explorateur de solutions et cliquez sur Ajouter, puis sur Nouveau dossier. Nommez le nouveau dossier DAL (pour Data Access Layer). Dans ce dossier,

créez un nouveau fichier de classe nommé SchoolContext.cs et remplacez le code du modèle par le code suivant :

```
using ContosoUniversity.Models;
using System.Data.Entity;
using System.Data.Entity.ModelConfiguration.Conventions;

namespace ContosoUniversity.DAL
{
    public class SchoolContext : DbContext
    {

        public SchoolContext() : base("SchoolContext")
        {
        }

        public DbSet<Student> Students { get; set; }
        public DbSet<Enrollment> Enrollments { get; set; }
        public DbSet<Course> Courses { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
          {
   modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
        }
    }

}
```

Spécifier des ensembles d'entités

Ce code crée une propriété DbSet pour chaque ensemble d'entités. Dans la terminologie Entity Framework, un *ensemble d'entités* correspond généralement à une table de base de données et une *entité* correspond à une ligne de la table.

Spécifier la chaine de connexion

Le nom de la chaîne de connexion (que vous ajouterez ultérieurement au fichier Web.config) est transmis au constructeur.

```
public SchoolContext() : base("SchoolContext")
{
}
```

Si vous ne spécifiez pas de chaîne de connexion ou le nom d'une de manière explicite, Entity Framework suppose que le nom de la chaîne de connexion est le même que le nom de la classe. Le nom de chaîne de connexion par défaut dans cet exemple serait alors SchoolContext, le même que celui que vous spécifiez explicitement.

Spécifier des noms de table singuliers

L'instruction modelBuilder.Conventions.Remove dans la méthode OnModelCreating empêche les noms de table d'être pluralisés. Si vous ne le faisiez pas, les tables générées dans la base de données

seraient nommées Students, Courses, et Enrollments. Au lieu de cela, les noms de table seront Student, Course, et Enrollment.

F. Initialiser la base de données avec les données de test

Entity Framework peut automatiquement créer (ou supprimer et recréer) une base de données pour vous lorsque l'application s'exécute. Vous pouvez spécifier que cela doit être fait à chaque exécution de votre application ou uniquement lorsque le modèle n'est pas synchronisé avec la base de données existante. Vous pouvez également écrire une méthode Seed qu'Entity Framework appelle automatiquement après la création de la base de données afin de la remplir avec des données de test.

Le comportement par défaut est de créer une base de données uniquement si elle n'existe pas (et de lever une exception si le modèle a changé et que la base de données existe déjà). Dans cette section, vous spécifierez que la base de données doit être supprimée et recréée chaque fois que le modèle change. La suppression de la base de données entraîne la perte de toutes vos données. Cela est généralement correct pendant le développement, car la méthode Seed s'exécutera lorsque la base de données sera recréée et recréera vos données de test. Mais en production, vous ne voulez généralement pas perdre toutes vos données à chaque fois que vous devez modifier le schéma de la base de données. Plus tard, vous verrez comment gérer les modifications de modèle en utilisant Code First Migrations pour modifier le schéma de la base de données au lieu de supprimer et de recréer la base de données.

1. Dans le dossier DAL, créez un nouveau fichier de classe nommé *SchoolInitializer.cs* et remplacez le code du modèle par le code suivant, ce qui provoque la création d'une base de données en cas de besoin et charge les données de test dans la nouvelle base de données.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data.Entity;
using ContosoUniversity.Models;

namespace ContosoUniversity.DAL
{
    public class SchoolInitializer : System.Data.Entity.
DropCreateDatabaseIfModelChanges<SchoolContext>
    {
        protected override void Seed(SchoolContext context)
        {
            var students = new List<Student>
            {
            new
Student{FirstMidName="Carson",LastName="Alexander",EnrollmentDate=DateTime.Pars
e("2005-09-01")},
            new
Student{FirstMidName="Meredith",LastName="Alonso",EnrollmentDate=DateTime.Parse
("2002-09-01")},
            new
Student{FirstMidName="Arturo",LastName="Anand",EnrollmentDate=DateTime.Parse("2
003-09-01")},
```

```
            new
Student{FirstMidName="Gytis",LastName="Barzdukas",EnrollmentDate=DateTime.Parse
("2002-09-01")},
            new
Student{FirstMidName="Yan",LastName="Li",EnrollmentDate=DateTime.Parse("2002-
09-01")},
            new
Student{FirstMidName="Peggy",LastName="Justice",EnrollmentDate=DateTime.Parse("
2001-09-01")},
            new
Student{FirstMidName="Laura",LastName="Norman",EnrollmentDate=DateTime.Parse("2
003-09-01")},
            new
Student{FirstMidName="Nino",LastName="Olivetto",EnrollmentDate=DateTime.Parse("
2005-09-01")}
            };

            students.ForEach(s => context.Students.Add(s));
            context.SaveChanges();
            var courses = new List<Course>
            {
            new Course{CourseID=1050,Title="Chemistry",Credits=3,},
            new Course{CourseID=4022,Title="Microeconomics",Credits=3,},
            new Course{CourseID=4041,Title="Macroeconomics",Credits=3,},
            new Course{CourseID=1045,Title="Calculus",Credits=4,},
            new Course{CourseID=3141,Title="Trigonometry",Credits=4,},
            new Course{CourseID=2021,Title="Composition",Credits=3,},
            new Course{CourseID=2042,Title="Literature",Credits=4,}
            };
            courses.ForEach(s => context.Courses.Add(s));
            context.SaveChanges();
            var enrollments = new List<Enrollment>
            {
            new Enrollment{StudentID=1,CourseID=1050,Grade=Grade.A},
            new Enrollment{StudentID=1,CourseID=4022,Grade=Grade.C},
            new Enrollment{StudentID=1,CourseID=4041,Grade=Grade.B},
            new Enrollment{StudentID=2,CourseID=1045,Grade=Grade.B},
            new Enrollment{StudentID=2,CourseID=3141,Grade=Grade.F},
            new Enrollment{StudentID=2,CourseID=2021,Grade=Grade.F},
            new Enrollment{StudentID=3,CourseID=1050},
            new Enrollment{StudentID=4,CourseID=1050,},
            new Enrollment{StudentID=4,CourseID=4022,Grade=Grade.F},
            new Enrollment{StudentID=5,CourseID=4041,Grade=Grade.C},
            new Enrollment{StudentID=6,CourseID=1045},
            new Enrollment{StudentID=7,CourseID=3141,Grade=Grade.A},
            };
            enrollments.ForEach(s => context.Enrollments.Add(s));
            context.SaveChanges();
        }
    }

}
```

La méthode Seed prend l'objet de contexte de base de données comme paramètre d'entrée et le code de la méthode utilise cet objet pour ajouter de nouvelles entités à la base de données. Pour chaque type d'entité, le code crée une collection de nouvelles entités, les ajoute à la propriété DbSet appropriée, puis enregistre les modifications dans la base de données. Il n'est pas nécessaire d'appeler la méthode SaveChanges après chaque groupe d'entités, comme cela est fait ici, mais cela vous aide à

localiser la source d'un problème si une exception se produit pendant que le code écrit dans la base de données.

2. Pour indiquer à Entity Framework d'utiliser votre classe d'initialisation, ajoutez un élément à l'élément entityFramework dans le fichier Web.config de l'application (celui du dossier racine du projet), comme illustré dans l'exemple suivant :

```
<entityFramework>
 <contexts>
<context type="ContosoUniversity.DAL.SchoolContext, ContosoUniversity">
<databaseInitializer type="ContosoUniversity.DAL.SchoolInitializer,
ContosoUniversity" />
</context>
</contexts>
  <defaultConnectionFactory
type="System.Data.Entity.Infrastructure.LocalDbConnectionFactory,
EntityFramework">
    <parameters>
      <parameter value="v11.0" />
    </parameters>
  </defaultConnectionFactory>
  <providers>
    <provider invariantName="System.Data.SqlClient"
type="System.Data.Entity.SqlServer.SqlProviderServices,
EntityFramework.SqlServer" />
  </providers>
</entityFramework>
```

Le context type spécifie le nom complet de la classe de contexte et l'assembly dans lequel elle se trouve, et le databaseinitializer type spécifie le nom complet de la classe d'initialisation et de l'assembly dans lequel elle se trouve. (Lorsque vous ne voulez pas que EF utilise l'initialiseur, vous pouvez définir un attribut sur l'élément context : disableDatabaseInitialization="true").

Une alternative à la définition de l'initialiseur dans le fichier Web.config consiste à le faire dans le code en ajoutant une instruction Database.SetInitializer à la méthode Application_Start dans le fichier Global.asax.cs .

L'application est maintenant configurée de sorte que lorsque vous accédez à la base de données pour la première fois dans une exécution donnée de l'application, Entity Framework compare la base de données au modèle (votre SchoolContextet classes d'entité). S'il y a une différence, l'application supprime et recrée la base de données.

G. Configurer EF 6 pour utiliser LocalDB

LocalDB est une version allégée du moteur de base de données SQL Server Express. Il est facile à installer et à configurer, démarre à la demande et s'exécute en mode utilisateur. LocalDB s'exécute dans un mode d'exécution spécial de SQL Server Express qui vous permet de travailler avec des bases de données en tant que fichiers .mdf. Vous pouvez placer les fichiers de base de données LocalDB dans le dossier App_Data d'un projet Web si vous souhaitez pouvoir copier la base de données avec le projet. LocalDB est installé par défaut avec Visual Studio.

Dans ce TP, vous travaillerez avec LocalDB. Ouvrez le fichier Web.config de l'application et ajoutez un élément connectionStrings avant l'élément appSettings, comme illustré dans l'exemple suivant.

(Assurez-vous de mettre à jour le fichier Web.config dans le dossier racine du projet. Il existe également un fichier Web.config dans le sous-dossier Views que vous n'avez pas besoin de mettre à jour).

```
<connectionStrings> <add name="SchoolContext" connectionString="Data
Source=(LocalDb)\MSSQLLocalDB;Initial Catalog=ContosoUniversity1;Integrated Security=SSPI;"
providerName="System.Data.SqlClient"/> </connectionStrings>
<appSettings>
  <add key="webpages:Version" value="3.0.0.0" />
  <add key="webpages:Enabled" value="false" />
  <add key="ClientValidationEnabled" value="true" />
  <add key="UnobtrusiveJavaScriptEnabled" value="true" />
</appSettings>
```

La chaîne de connexion que vous avez ajoutée spécifie qu'Entity Framework utilisera une base de données *LocalDB* nommée *ContosoUniversity1.mdf* . (La base de données n'existe pas encore mais EF la créera).

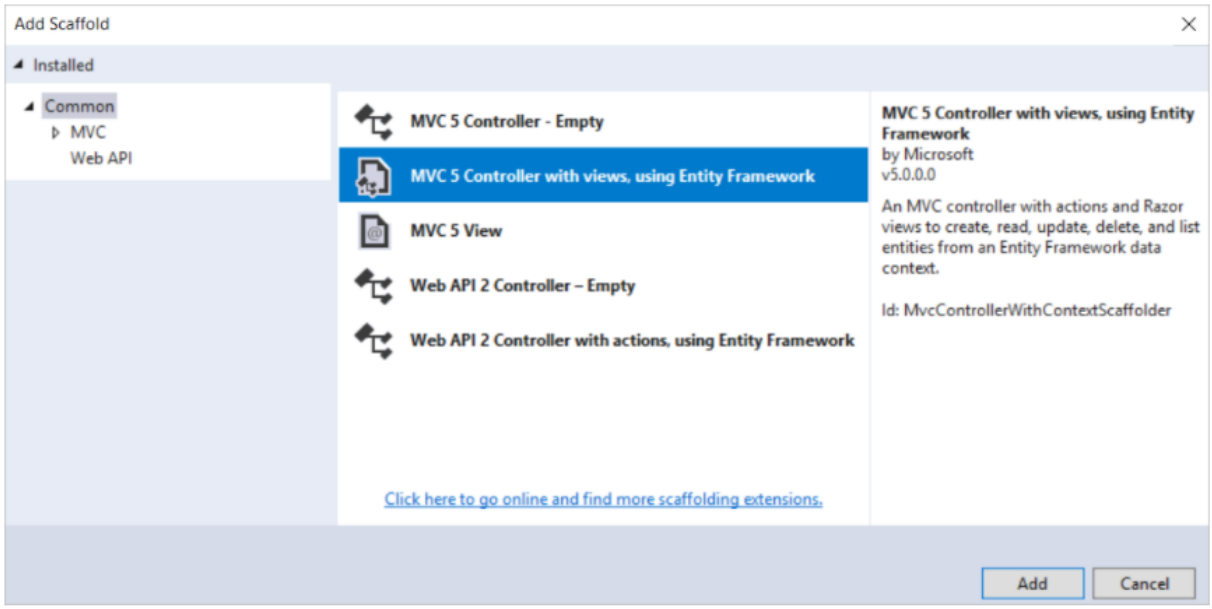
Vous n'avez pas réellement besoin d'une chaîne de connexion dans le fichier *Web.config* . Si vous ne fournissez pas de chaîne de connexion, Entity Framework utilise une chaîne de connexion par défaut basée sur votre classe de contexte.

H. Créer un contrôleur et des vues

Vous allez maintenant créer une page Web pour afficher les données. Le processus de demande des données déclenche automatiquement la création de la base de données. Vous commencerez par créer un nouveau contrôleur. Mais avant de faire cela, générez le projet pour rendre les classes de modèle et de contexte disponibles pour la construction de contrôleur MVC.

1. Cliquez avec le bouton droit sur le dossier Contrôleurs dans l'Explorateur de solutions, sélectionnez Ajouter, puis cliquez sur Nouvel élément construit.

2. Dans la boîte de dialogue d'ajout, sélectionnez MVC 5 Controller with views, using Entity Framework , puis choisissez Add.



3. Dans la boîte de dialogue, effectuez les sélections suivantes, puis choisissez Ajouter :
   - ✓ Classe modèle : Student(ContosoUniversity.Models) . (Si vous ne voyez pas cette option dans la liste déroulante, déboguer le projet et réessayez.)
   - ✓ Classe de contexte de données : SchoolContext (ContosoUniversity.DAL).
   - ✓ Nom du contrôleur : StudentController (pas StudentsController).

Laissez les valeurs par défaut pour les autres champs.

Lorsque vous cliquez sur Ajouter, un fichier StudentController.cs est créé et un ensemble de vues (fichiers.cshtml) qui fonctionnent avec le contrôleur. Vous pourrez également profiter de certaines fonctionnalités supplémentaires du scaffolder : créez votre première classe de modèle, ne créez pas de chaîne de connexion, puis dans la zone Ajouter un contrôleur, spécifiez un nouveau contexte de données. En sélectionnant le bouton + à côté de **Data context class**. Ainsi votre classe DbContext et votre chaîne de connexion ainsi que le contrôleur et les vues sont tous créés.

4. Visual Studio ouvre le fichier Controllers\StudentController.cs. Vous voyez qu'une variable de classe a été créée qui instancie un objet de contexte de base de données :

```
private SchoolContext db = new SchoolContext();
```

La méthode Index action obtient une liste d'étudiants à partir de l'ensemble d'entités Students en lisant la propriété Students de l'instance de contexte de base de données :

```
public ViewResult Index()
{
    return View(db.Students.ToList());
}
```

La vue Student\Index.cshtml affiche cette liste dans un tableau :

```
<table>
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.LastName)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.FirstMidName)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.EnrollmentDate)
        </th>
        <th></th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.LastName)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.FirstMidName)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.EnrollmentDate)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.ID }) |
            @Html.ActionLink("Details", "Details", new { id=item.ID }) |
            @Html.ActionLink("Delete", "Delete", new { id=item.ID })
        </td>
    </tr>
}
```

5. Appuyez sur Ctrl+F5 pour exécuter le projet. (Si vous obtenez une erreur "Cannot create Shadow Copy", fermez le navigateur et réessayez).

Cliquez sur l'onglet Students pour voir les données de test insérées par la méthode Seed. Selon la taille de la fenêtre de votre navigateur, vous verrez le lien de l'onglet Students dans la barre d'adresse supérieure ou vous devrez cliquer dans le coin supérieur droit pour voir le lien.

I. Consulter la base de données

Lorsque vous avez exécuté la page Étudiants et que l'application a tenté d'accéder à la base de données, EF a découvert qu'il n'y avait pas de base de données et en a créé une. EF a ensuite exécuté la méthode seed pour remplir la base de données avec des données.

Vous pouvez utiliser l'Explorateur de serveurs ou l'Explorateur d'objets SQL Server (SSOX) pour afficher la base de données dans Visual Studio. Pour ce TP, vous utiliserez l'Explorateur de serveurs.

1. Fermez le navigateur.
2. Dans l'Explorateur de serveurs, développez Connexions de données (vous devrez peut-être d'abord sélectionner le bouton d'actualisation), développez Contexte scolaire

(ContosoUniversity) , puis développez Tables pour voir les tables de votre nouvelle base de données.

3. Cliquez avec le bouton droit sur la table Student et cliquez sur Afficher les données de la table pour voir les colonnes qui ont été créées et les lignes qui ont été insérées dans la table.
4. Fermez la connexion de l'explorateur de serveur.

Étant donné que vous utilisez l'initialiseur DropCreateDatabaseIfModelChanges, vous pouvez maintenant apporter une modification à la classe Student, exécuter à nouveau l'application et la base de données sera automatiquement recréée pour correspondre à votre modification. Par exemple, si vous ajoutez une propriété EmailAddress à la classe Student, réexécutez la page Studentd, puis examinez à nouveau le tableau, vous verrez une nouvelle colonne EmailAddress.