

ASP.NET Core Razor Pages et Entity Framework Core

MOHAMED AIROUCHE

AUTOMNE 2020

ASP.NET Core Razor Pages

Pour les développeurs qui veulent mettre en place de petites applications Web sans avoir à faire face aux contraintes architecturales qu'impose MVC, Razor Pages est désormais offert avec ASP.NET Core 2.0 depuis l'été 2017. Razor pages peuvent rendre le codage des scénarios orientés page plus facile et plus productif que l'utilisation de contrôleurs et de vues.

Description d'une page Razor

Une page Razor est un fichier .cshtml pouvant contenir du code HTML, C#, JavaScript et CSS. La première instruction dans un fichier Razor Page est :

@Page

La directive @page transforme le fichier en une action MVC - ce qui signifie qu'il gère les requêtes directement, sans passer par un contrôleur. La directive @page doit être la première instruction dans un Razor page.

<https://docs.microsoft.com/fr-ca/aspnet/core/razor-pages/?view=aspnetcore-2.1&tabs=visual-studio>

<https://rdonfack.developpez.com/tutoriels/dotnet/nouveautes-aspnet-core-2/>

<https://rdonfack.developpez.com/tutoriels/asp-net-core-razor-pages/>

Razor Pages en ASP.NET Core

Routage pour Razor Pages

- ✓ Dans une application ASP.NET Core Razor Pages, les pages Web sont contenues dans un dossier Pages. La convention de routage qui est implémentée par ASP.NET Core va permettre de matcher une URL avec le fichier .cshtml correspondant dans le dossier Pages :
- ✓ Les associations de chemins d'URL vers les pages sont déterminées par l'emplacement de la page dans le système de fichiers. Le tableau suivant montre les chemins d'accès à la page Razor et les URL correspondantes :

Nom de fichier et chemin	URL correspondante
<i>/Pages/Index.cshtml</i>	<i>/ or /Index</i>
<i>/Pages/Contact.cshtml</i>	<i>/Contact</i>
<i>/Pages/Store/Contact.cshtml</i>	<i>/Store/Contact</i>
<i>/Pages/Store/Index.cshtml</i>	<i>/Store or /Store/Index</i>

<https://docs.microsoft.com/fr-ca/aspnet/core/razor-pages/?view=aspnetcore-2.1&tabs=visual-studio>

<https://rdonfack.developpez.com/tutoriels/asp-net-core-razor-pages/>

Razor Pages en ASP.NET Core

Une convention a été définie par Razor Pages pour identifier le code qui doit être exécuté au chargement d'une page, en fonction de la requête HTTP :

- pour une requête HTTP Get, la méthode `OnGet()` dans le code métier sera exécutée. Si cette méthode contient du code asynchrone, elle doit avoir pour nom `OnGetAsync()`;
- pour une requête HTTP Post, la méthode `OnPost()` dans le code métier sera exécutée. Si cette méthode contient du code asynchrone, elle doit avoir pour nom `OnPostAsync()`.

<https://docs.microsoft.com/fr-ca/aspnet/core/razor-pages/?view=aspnetcore-2.1&tabs=visual-studio>

<https://rdonfack.developpez.com/tutoriels/asp-net-core-razor-pages/>

Razor Pages en ASP.NET Core

Pour séparer le contenu de présentation de la logique métier dans le même fichier, Razor Pages offre l'instruction `@functions`

Exemple :

```
@page
@functions{
    public string Message { get; set; }

    public void OnGet()
    {
        Message = "Your application description page.";
    }
}

@{
    ViewData["Title"] = "About";
}
<h2>@ViewData["Title"].</h2>
<h3>@Message</h3>

<p>Use this area to provide additional information.</p>
```

<https://rdonfack.developpez.com/tutoriels/asp-net-core-razor-pages/>

Razor Pages en ASP.NET Core

Pour permettre une meilleure séparation du code, Razor Pages introduit la notion de PageModel. Au lieu d'avoir une seule page, chaque page Razor peut être une paire de fichiers:

- Un fichier .cshtml qui contient un balisage HTML avec du code C # utilisant la syntaxe Razor.
- Un fichier .cshtml.cs qui contient du code C # qui gère les événements de page.

Exemple :

✓ About.cshtml

```
@page
@model AboutModel
@{
    ViewData["Title"] = "About";
}
<h2>@ViewData["Title"]</h2>
<h3>@Model.Message</h3>

<p>Use this area to provide additional information.</p>
```

✓ About.cshtml.cs

```
public class AboutModel : PageModel
{
    public string Message { get; set; }

    public void OnGet()
    {
        Message = "Your application description page.";
    }
}
```

<https://rdonfack.developpez.com/tutoriels/asp-net-core-razor-pages/>

ASP.NET Core Razor Pages

Les services de l'application Pages Razor sont activés dans la classe Startup comme pour les applications MVC. La classe Startup est exécuté au démarrage de l'application :

- Les services nécessaires à l'application sont configurés avec la méthode **ConfigureServices**.
- Le pipeline de traitement des demandes de l'application est défini comme une série de composants d'intergiciel (middleware) avec la méthode **Configure**.

```
// Méthode pour ajouter les services à l'application
// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        // This lambda determines whether user consent for non-essential cookies is needed for a given request.
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
}
```

<https://docs.microsoft.com/fr-ca/aspnet/core/razor-pages/?view=aspnetcore-2.1&tabs=visual-studio>

ASP.NET Core Razor Pages

```
// Méthode appelée pour configurer les services de l'applications
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
    }

    app.UseStaticFiles();
    app.UseCookiePolicy();

    app.UseMvc();
}
```


Exemple

Développement d'une application qui permet de gérer des employés et des départements :

On souhaite créer une application qui permet de gérer des employés et les départements de ces employés. Un département est défini par son identifiant et son nom. Chaque employé est défini par son identifiant, son nom, son prénom, son sexe, son identifiant de département et sa ville.

L'application doit permettre de :

➤ Pour les employés :

- Afficher tous les employés.
- Afficher les détails d'un employé.
- Saisir et ajouter un nouvel employé.
- Éditer et modifier un employé.
- Supprimer un employé.

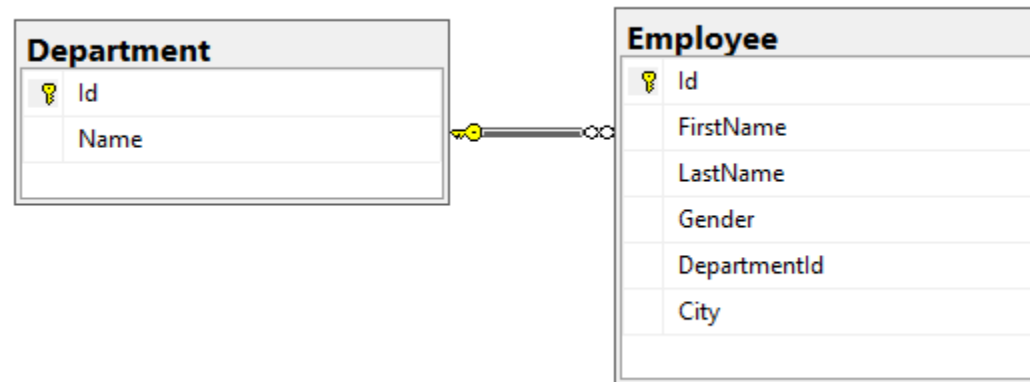
➤ Pour les départements :

- Afficher tous les départements.
- Afficher les détails d'un département.
- Saisir et ajouter un nouveau département.
- Éditer et modifier un département.
- Supprimer un département.

Exemple

Développement d'une application qui permet de gérer des employés et des départements :

Les employés et les départements seront stockés dans une base de données locale SQL Server Express. Cette base de données doit contenir deux tables (table Employee et table Department).

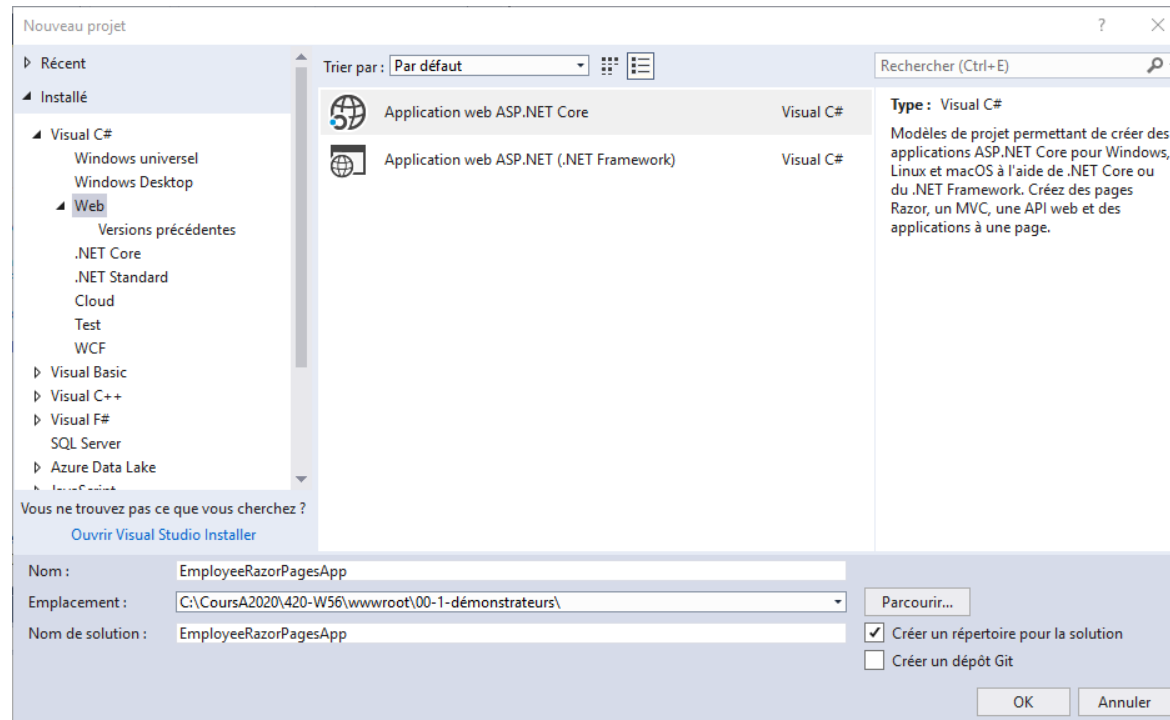


Dans cet exemple, l'application utilisera l'approche Code First de l'ORM Entity Framework Core pour l'accès aux données.

Exemple

Création d'une application ASP.NET Core Razor Pages :

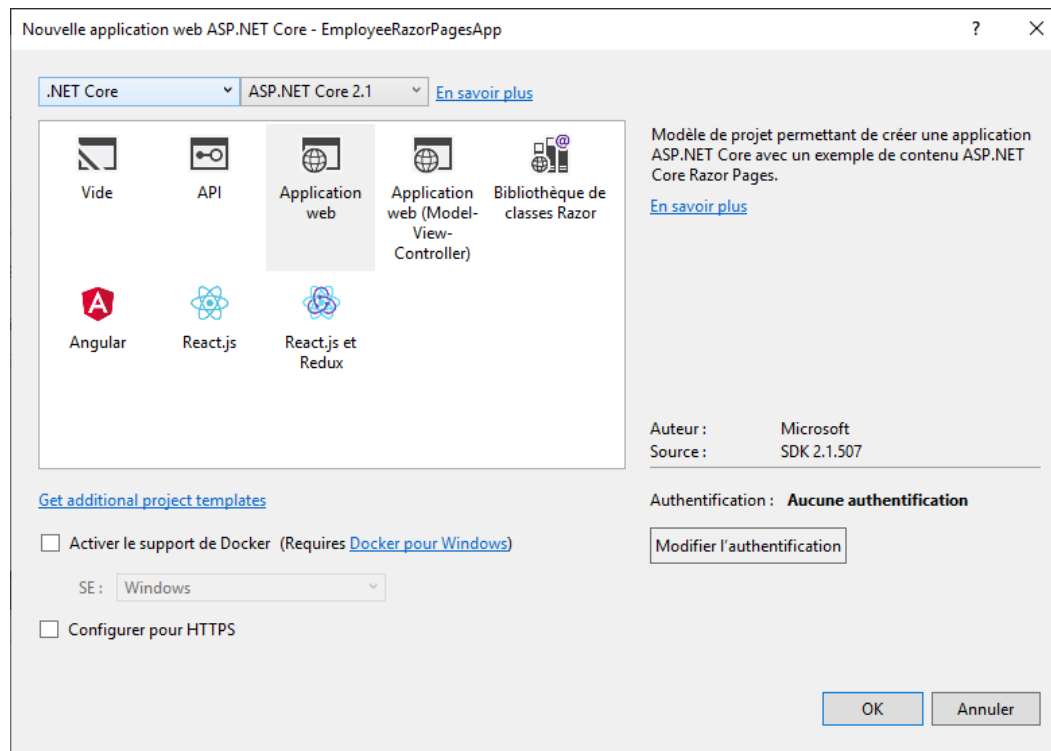
- Créez une application web ASP.NET Core Razor Pages : **EmployeeRazorPagesApp**.



Exemple

Création d'une application ASP.NET Core Razor Pages :

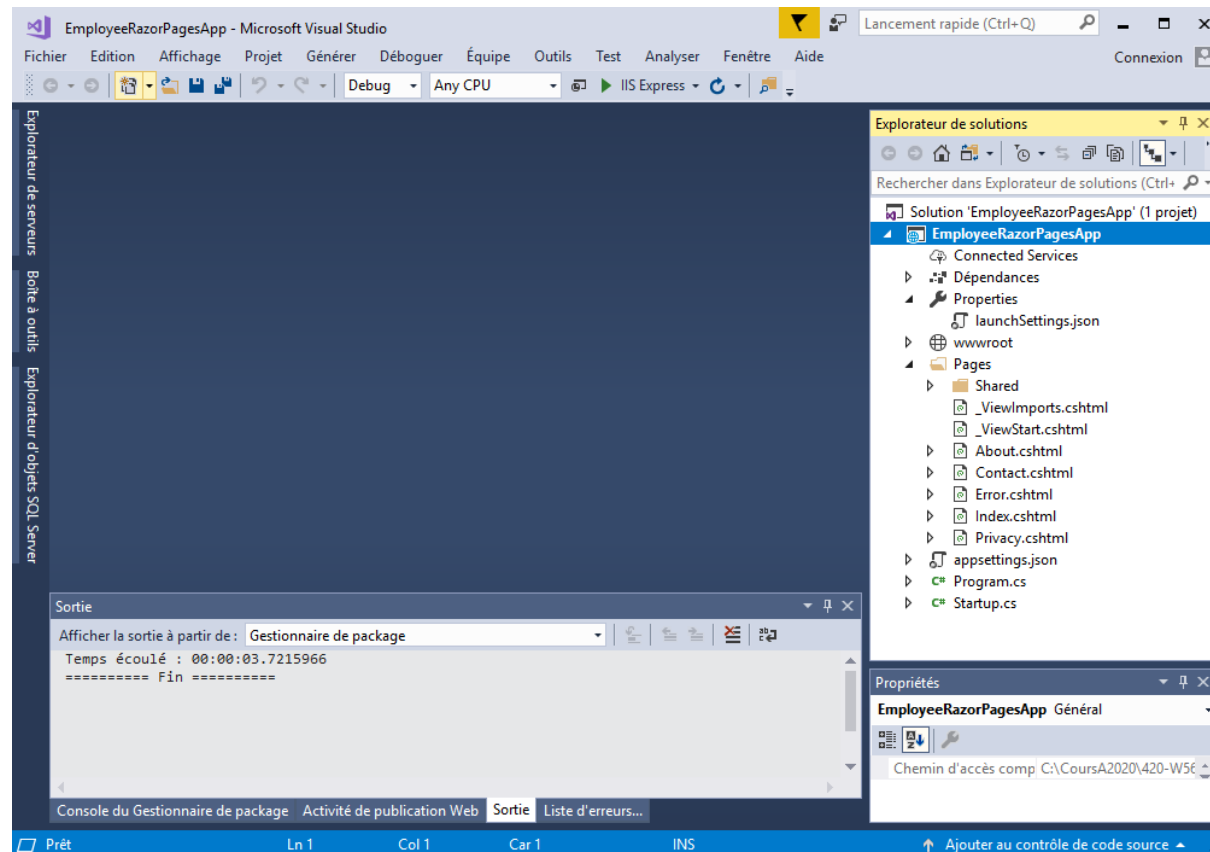
- Sélectionnez le modèle Application web.



Exemple

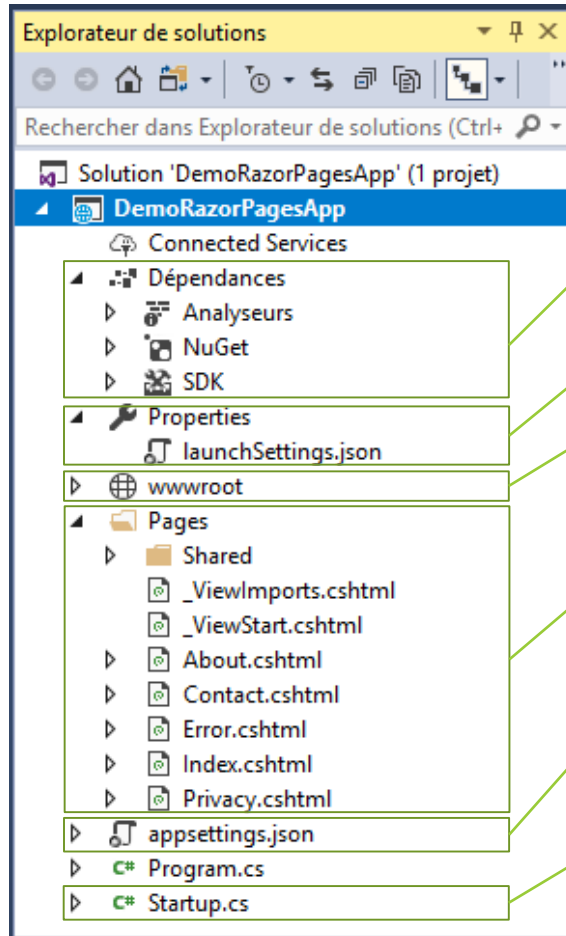
Création d'une application ASP.NET Core Razor Pages :

- L'application ASP.NET Core Razor Pages **EmployeeRazorPagesApp** est créée.



Exemple

Structure d'une application ASP.NET Core Razor Pages :



Dépendances du projet : contiennent tous les packages NuGet côté serveur installés et toutes les bibliothèques et Framework utilisés dans le projet.

launchSettings.json est un fichier qui inclut les profils Visual Studio des paramètres de débogage.

wwwroot contient les ressources statiques de l'application web : HTML, CSS, JavaScript, Images, etc...

Pages : contient toutes les pages Razor de votre application. Par défaut, ce dossier contient les pages Index, Contact, About et Error.

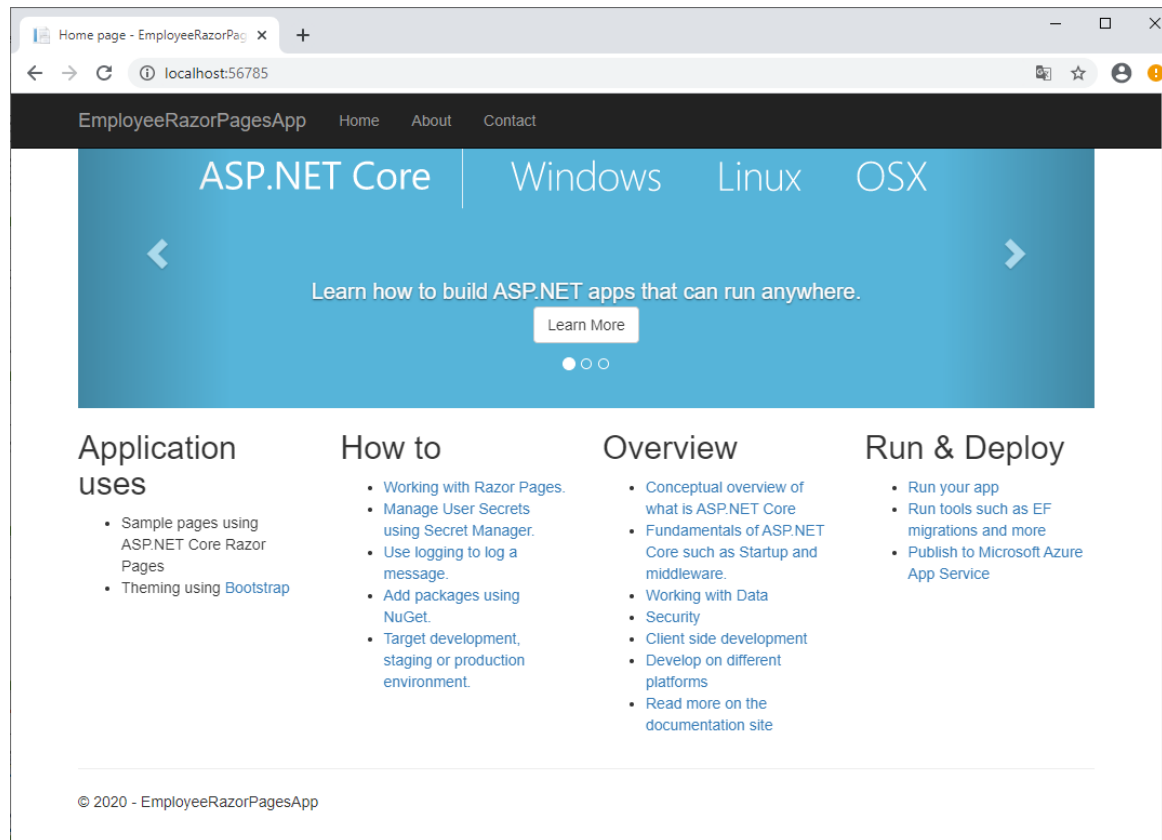
appsettings.json est un fichier qui peut être utilisé pour configurer l'application.

Startup.cs représente la classe de démarrage. C'est la première classe qui sera instanciée par le Framework. Les contenus des deux méthodes **ConfigureServices** et **Configure** seront exécutés au démarrage de l'application.

<https://rdonack.developpez.com/tutoriels/asp-net-core-razor-pages/>

Exemple

La figure suivante montre la page affichée après l'exécution de l'application :



Exemple

Ajout de modèle à utiliser par Entity Framework Core:

- ✓ Pour utiliser l'approche Code First d'ORM Entity Framework Core, il faut ajouter les classes des entités et la classe de contexte qui hérite de la classe DbContext.
- ✓ La classe de contexte de la base de données : `EmployeeDbContext`

```
using Microsoft.EntityFrameworkCore;

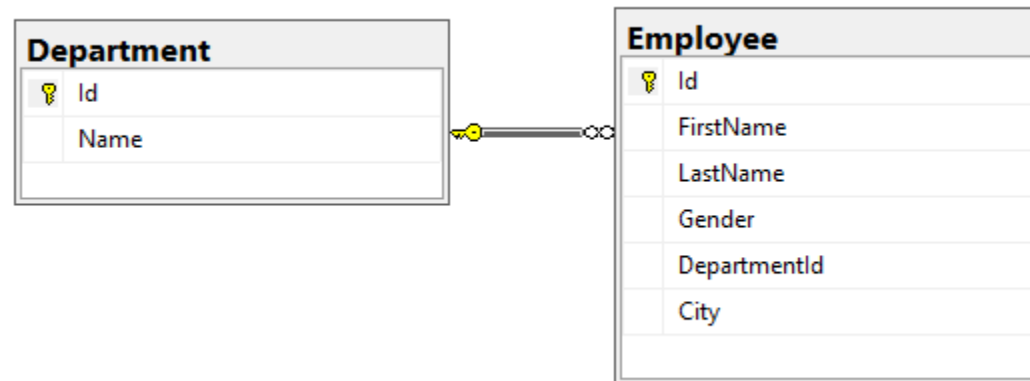
namespace EmployeeAppCore.Models
{
    public class EmployeeDbContext : DbContext
    {
        public EmployeeDbContext(DbContextOptions<EmployeeDbContext> options)
            : base()
        {
        }
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer("Server =(localdb)\\mssqllocaldb;Database=EmployeeDbMVC;Trusted_Connection=True;");
        }
        public DbSet<Employee> Employees { get; set; }

        public DbSet<Department> Departments { get; set; }
    }
}
```


Exemple

L'association : One To Many ----- ManyToOne :

La base de données doit contenir deux tables reliées (table Employee et table Department).



- ✓ Prenons le cas où chaque employé appartient à un département et un département contient plusieurs employés.
- ✓ Dans ce cas, nous avons deux entités : Employee et Department.
- ✓ La relation qui lie ces deux entités est de type OneToMany \longleftrightarrow ManyToOne

Exemple

✓ La classe d'entité : **Employee**

```
using System.ComponentModel.DataAnnotations;

namespace _09_DemoEmployeeDepartementEF.Models
{
    public class Employee
    {
        public int Id { get; set; }
        [Required]
        public string FirstName { get; set; }
        [Required]
        public string LastName { get; set; }
        [Required]
        public string Gender { get; set; }
        [Required]
        public int DepartmentId { get; set; }
        [Required]
        public string City { get; set; }
        public virtual Department Department { get; set; }
    }
}
```

✓ La classe d'entité : **Department**

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

namespace _09_DemoEmployeeDepartementEF.Models
{
    public class Department
    {
        public int Id { get; set; }
        [Required]
        public string Name { get; set; }
        public virtual ICollection<Employee> Employees { get; set; }
    }
}
```

Propriété de clé


Propriétés de navigation

Exemple


Inscrire le contexte de base de données

ASP.NET Core comprend l'injection de dépendances. Les services (tels que le contexte de base de données EF Core) doivent être inscrits auprès de l'injection de dépendances au démarrage de l'application. Ces services sont fournis par les composants qui requièrent ces services par le biais de paramètres de constructeur. Vous allez inscrire le contexte de base de données auprès du conteneur d'injection de dépendances.

- ✓ En tête du fichier *Startup.cs*, ajoutez l'instruction using suivante :

 `using Microsoft.EntityFrameworkCore;`

- ✓ Ajoutez le code en surbrillance suivant dans la méthode `Startup.ConfigureServices` :

```
public void ConfigureServices(IServiceCollection services)
{
     services.AddDbContext<EmployeeDbContext>(options =>
    options.UseSqlServer(Configuration.GetConnectionString("EmployeeConnection")));
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
}
```


<https://docs.microsoft.com/En-us/aspnet/core/razor-pages/?view=aspnetcore-2.1&tabs=visual-studio>

Exemple

Ajouter une chaîne de connexion de base de données

Ajoutez une chaîne de connexion dans le fichier *appsettings.json* :

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "EmployeeConnection":
      "Server=(localdb)\\mssqllocaldb;Database=EmployeeDbMVC;Trusted_Connection=True;MultipleActiveResultSets=true"
  }
}
```



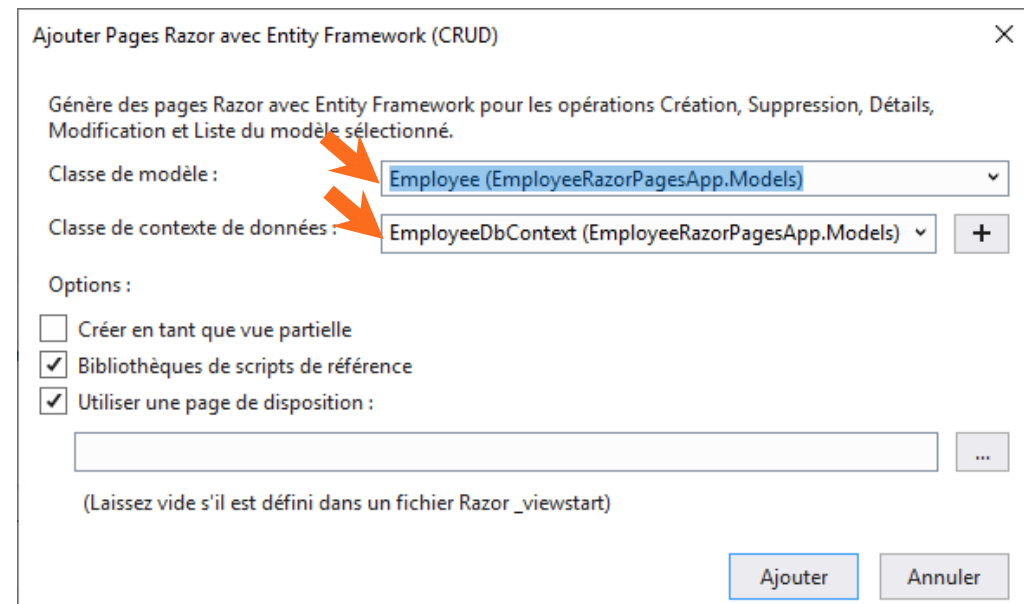
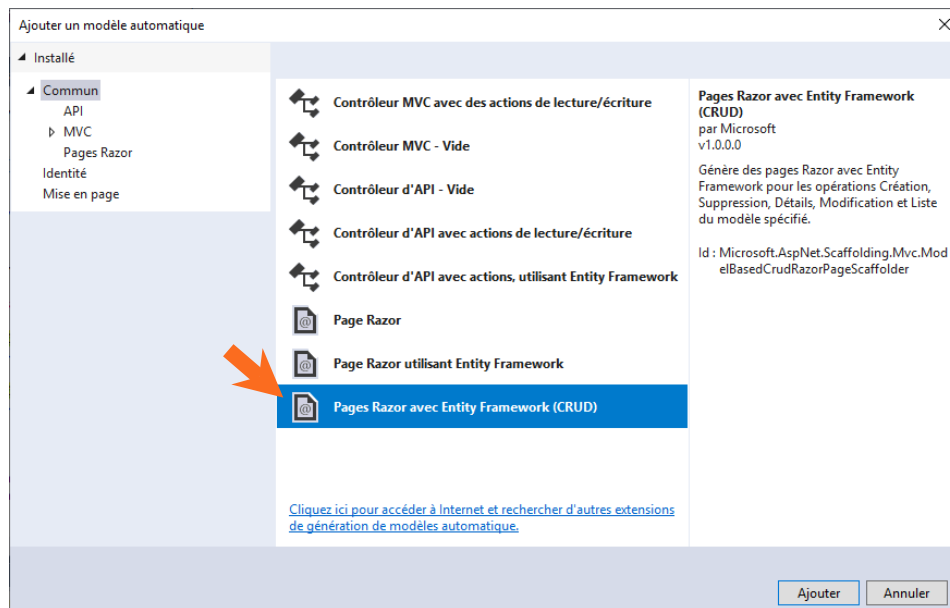
Générez le projet pour la vérification des erreurs du compilateur.

<https://docs.microsoft.com/En-us/aspnet/core/razor-pages/?view=aspnetcore-2.1&tabs=visual-studio>

Exemple

Générer automatiquement des pages pour le modèle Employee

Ajoutez au projet un nouveau dossier **Employees** dans le dossier Pages. Utilisez l'outil de génération de modèles automatique pour créer, lire, mettre à jour et supprimer des pages du modèle **Employee**.

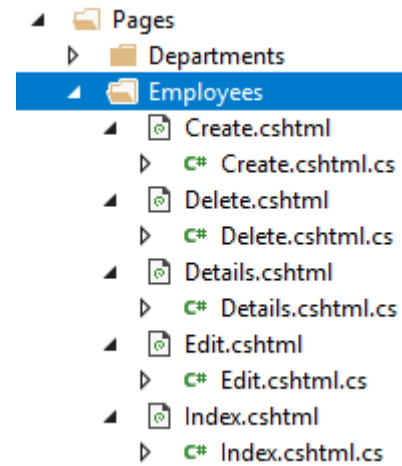
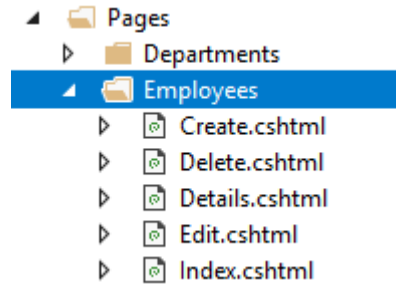


<https://docs.microsoft.com/En-us/aspnet/core/razor-pages/?view=aspnetcore-2.1&tabs=visual-studio>

Exemple

Visual studio vas créer automatiquement dans le dossier **Employees** les pages suivantes :

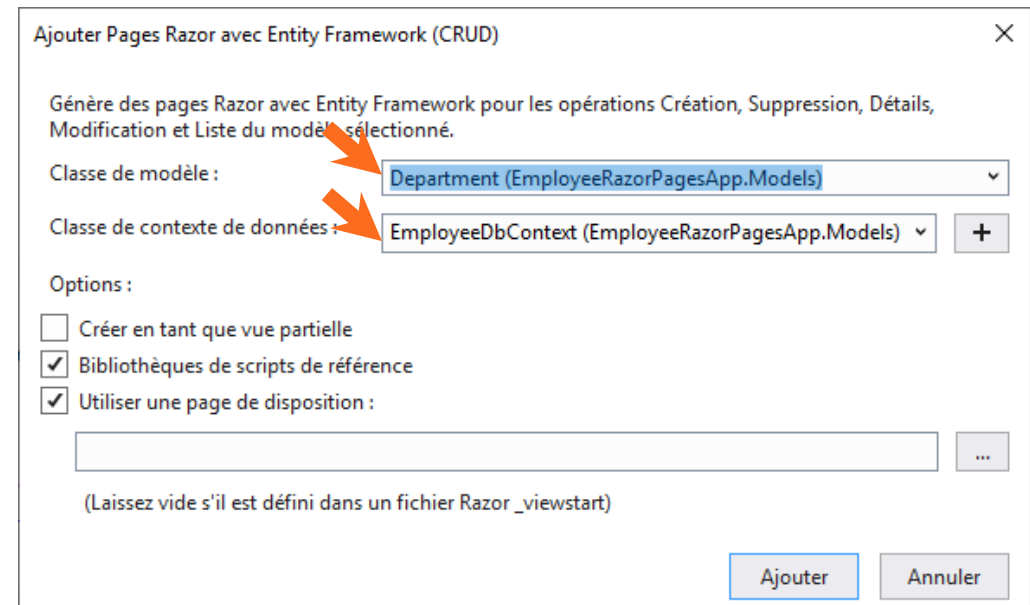
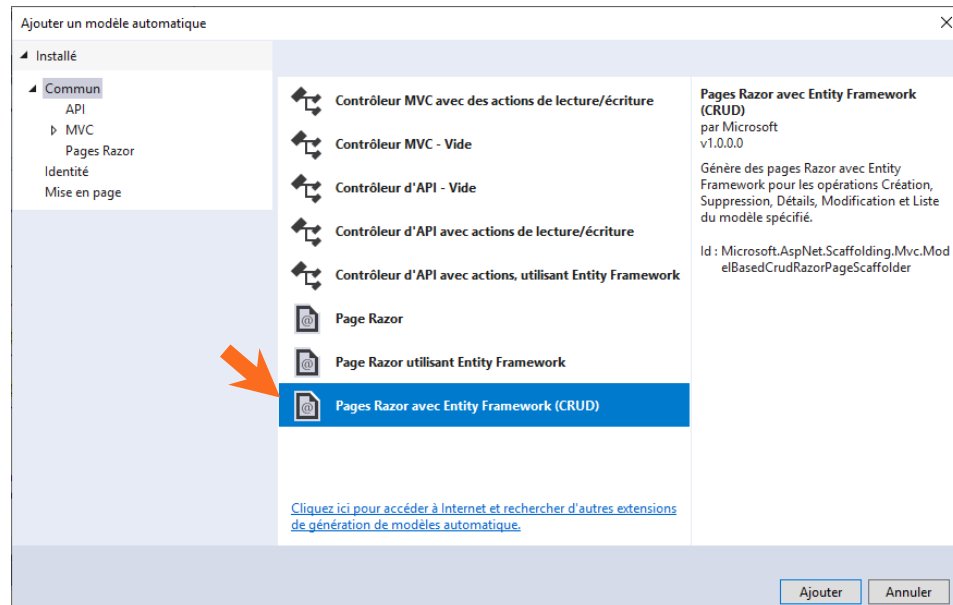
- Index.cshtml
- Create.cshtml
- Edit.cshtml
- Delete.cshtml
- Details.cshtml



Exemple

Générer automatiquement des pages pour le modèle Department

Ajoutez au projet un nouveau dossier **Departments** dans le dossier Pages. Utilisez l'outil de génération de modèles automatique pour créer, lire, mettre à jour et supprimer des pages du modèle **Department**.

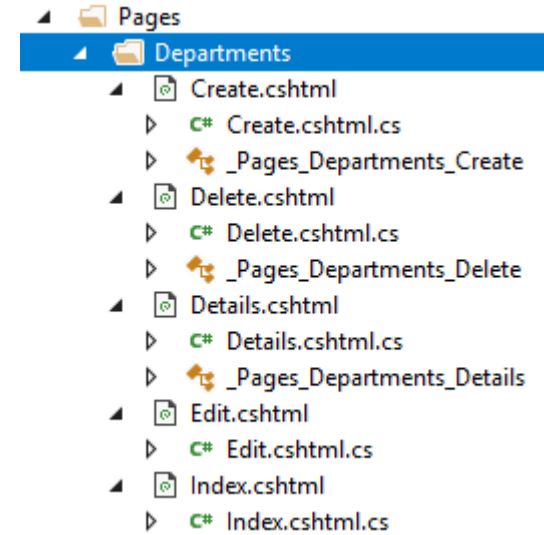
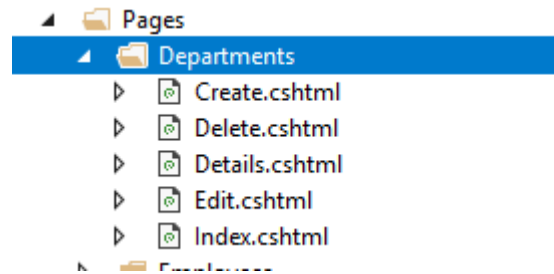


<https://docs.microsoft.com/En-us/aspnet/core/razor-pages/?view=aspnetcore-2.1&tabs=visual-studio>

Exemple

Visual studio vas créer automatiquement dans le dossier **Departments** les pages suivantes :

- Index.cshtml
- Create.cshtml
- Edit.cshtml
- Delete.cshtml
- Details.cshtml



Exemple

Vous ne pouvez pas utiliser les pages générées automatiquement si la base de données n'existe pas. Il faut tout d'abord créer la base de données en utilisant la fonctionnalité de migration.

Migration initiale

Utilisez la fonctionnalité Migrations d'EF Core pour créer la base de données. La fonctionnalité Migrations est un ensemble d'outils qui vous permettent de créer et de mettre à jour une base de données pour qu'elle corresponde à votre modèle de données.

Dans le menu **Outils**, sélectionnez **Gestionnaire de package NuGet > Console du gestionnaire de package** (PMC).

Dans la console du gestionnaire de package, entrez les commandes suivantes :

 **Add-Migration** InitialCreate
Update-Database

<https://docs.microsoft.com/fr-ca/aspnet/core/tutorials/first-mvc-app/adding-model?view=aspnetcore-3.1&tabs=visual-studio>

Exemple



Add-Migration InitialCreate
Update-Database

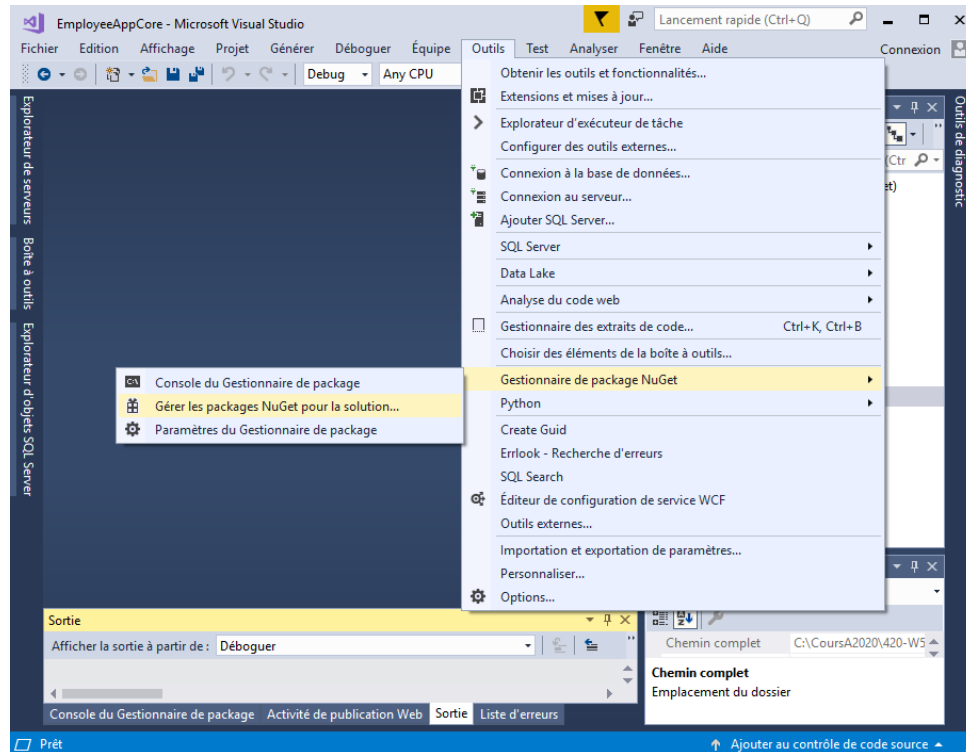
- **Add-Migration** InitialCreate : Génère un fichier de migration *_InitialCreate.cs* *migrations/{timestamp}*. L'argument **InitialCreate** est le nom de la migration. Vous pouvez utiliser n'importe quel nom, mais par convention, un nom décrivant la migration est sélectionné. Étant donné qu'il s'agit de la première migration, la classe générée contient du code permettant de créer le schéma de la base de données. Le schéma de base de données est basé sur le modèle spécifié dans la classe `EmployeeDbContext`.
- **Update-Database** : Met à jour la base de données avec la dernière migration, créée par la commande précédente. La commande exécute la méthode `Up` dans le fichier *Migrations/{horodatage}_InitialCreate.cs*, ce qui entraîne la création de la base de données.

<https://docs.microsoft.com/fr-ca/aspnet/core/tutorials/first-mvc-app/adding-model?view=aspnetcore-3.1&tabs=visual-studio>

Exemple

Ajout d'Entity Framework Core au projet :

- ✓ Utilisez l'outil NuGet en cas où votre projet n'arrive pas à ajouter les dépendances EF:



Les dépendances à ajouter au projet :

Microsoft.EntityFrameworkCore

Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Tools

Exemple

Ajout d'Entity Framework Core au projet :

- ✓ Pour ajouter Entity Framework Core au projet, utilisez l'outil NuGet.
- ✓ Chercher Entity Framework Core, puis cocher la case à cocher pour l'ajouter au projet.

