

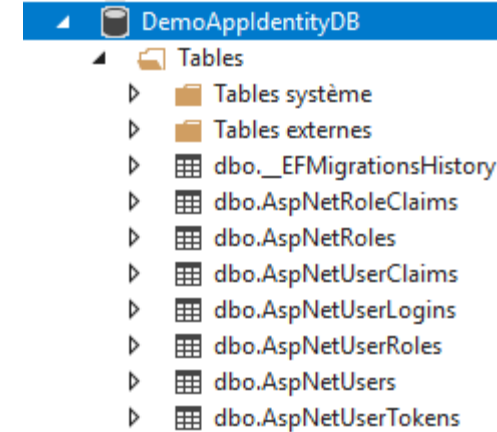
ASP.NET Core Identity

MOHAMED AIROUCHE

AUTOMNE 2020

ASP.NET Core Identity

- ASP.NET Core Identity est un système d'appartenance qui ajoute une fonctionnalité de connexion aux applications ASP.NET Core. Les utilisateurs peuvent créer un compte avec les informations de connexion stockées dans Identity ou utiliser un fournisseur de connexion externe. Les fournisseurs de connexion externes pris en charge incluent Facebook, Google, Microsoft et Twitter.
- Identity peut être configuré à l'aide d'une base de données SQL Server pour stocker les noms d'utilisateur, les mots de passe et les données de profil. Sinon, une autre source de stockage persistant peut être utilisée.



<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-2.1&tabs=visual-studio>

ASP.NET Core Identity

Configurer les services d'Identity

Le code suivant configure Identity avec les valeurs d'option par défaut. Les services sont mis à disposition de l'application via l'injection de dépendances.

```
services.AddDefaultIdentity<IdentityUser>().AddEntityFrameworkStores<ApplicationDbContext>();
```

Modification des options par défaut pour le mot de passe d'Identity

Le code suivant permet de modifier les options de mot de passe par défaut :

```
services.Configure<IdentityOptions>(options =>
{
    // Password settings.
    options.Password.RequireDigit = true;
    options.Password.RequireLowercase = true;
    options.Password.RequireNonAlphanumeric = true;
    options.Password.RequireUppercase = true;
    options.Password.RequiredLength = 6;
    options.Password.RequiredUniqueChars = 1;
});
```

<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-2.1&tabs=visual-studio>

ASP.NET Core Identity

Activation d'Identity pour l'authentification

L'Identity est activée en appelant `UseAuthentication`. `UseAuthentication` ajoute un middleware d'authentification au pipeline de requêtes.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage(); app.UseDatabaseErrorPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error"); app.UseHsts();
    }
    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseCookiePolicy();
    app.UseAuthentication();

    app.UseMvc(routes =>
    {
        routes.MapRoute( name: "default",
                        template: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

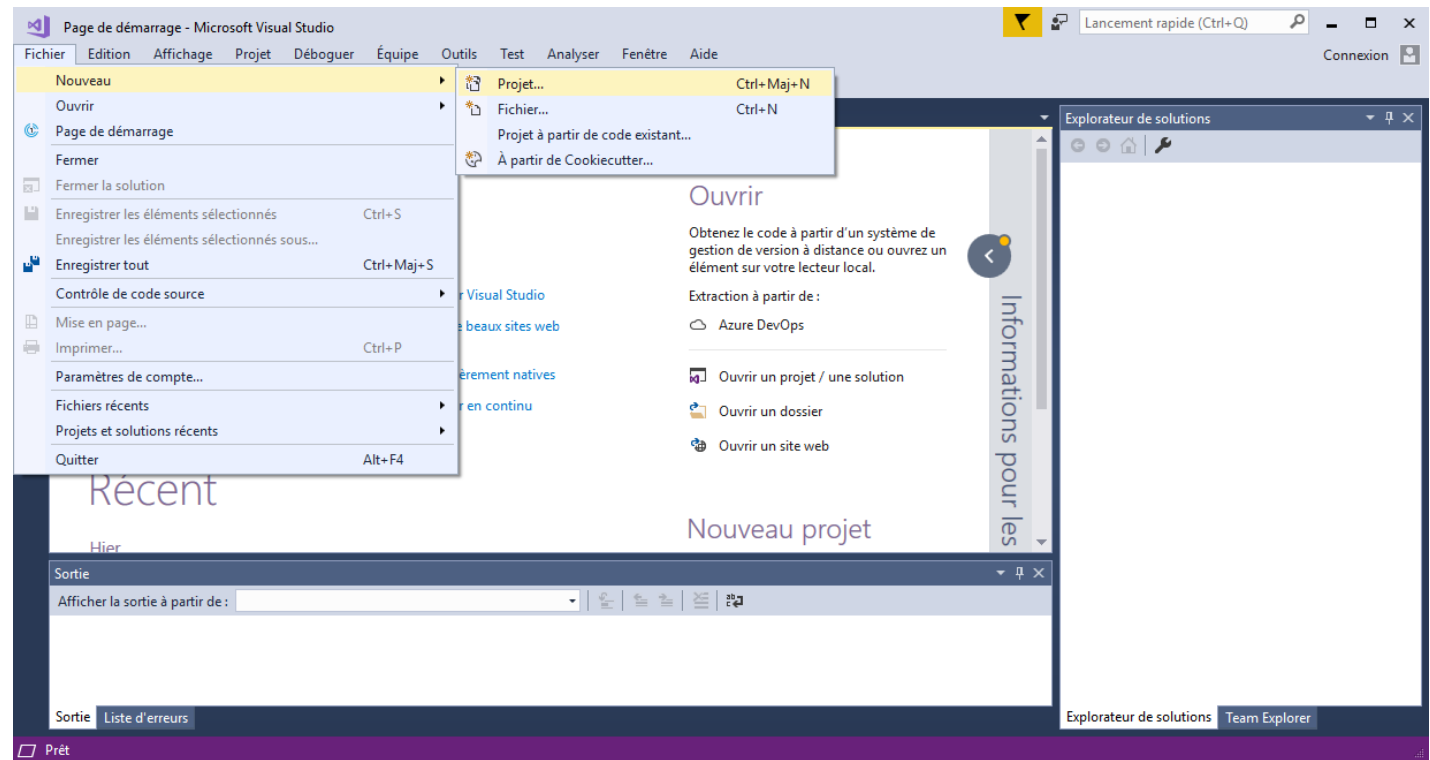
<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-2.1&tabs=visual-studio>

Exemple 1

Développement d'une application qui utilise une authentification :

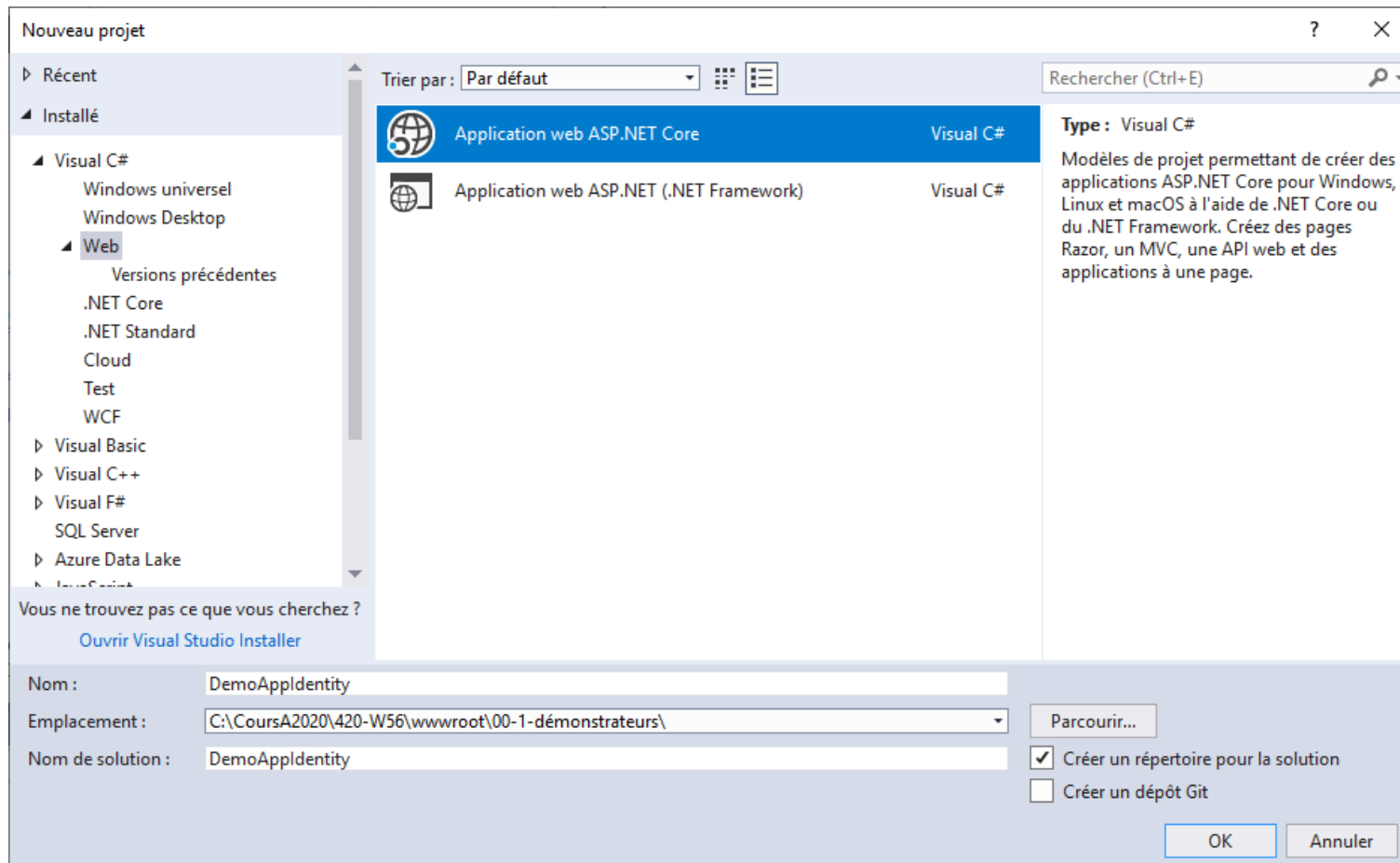
On souhaite créer une application MVC avec une authentification.

La création d'un projet web MVC utilisant .NET Core s'effectue via le menu **Fichier-Nouveau Projet**



Exemple 1- Création d'un projet

Sélectionnez (Application web ASP.NETCore) et nommez le projet : **DemoApplidentity**



Exemple 1-Création d'un projet

Sélectionnez le modèle Application web (Model-View-Controller) et Cliquez sur le bouton *Modifier l'authentification*

Nouvelle application web ASP.NET Core - DemoApplIdentity

.NET Core ASP.NET Core 2.1 [En savoir plus](#)

Vide API Application web **Application web (Model-View-Controller)** Bibliothèque de classes Razor

Angular React.js React.js et Redux

Modèle de projet permettant de créer une application ASP.NET Core avec des exemples de vues et de contrôleurs ASP.NET Core MVC. Vous pouvez également utiliser ce modèle pour les services HTTP RESTful.

[En savoir plus](#)

Auteur : Microsoft
Source : SDK 2.1.507

Authentification : **Aucune authentification**

[Get additional project templates](#)

☐ Activer le support de Docker (Requies [Docker pour Windows](#))

SE : Windows

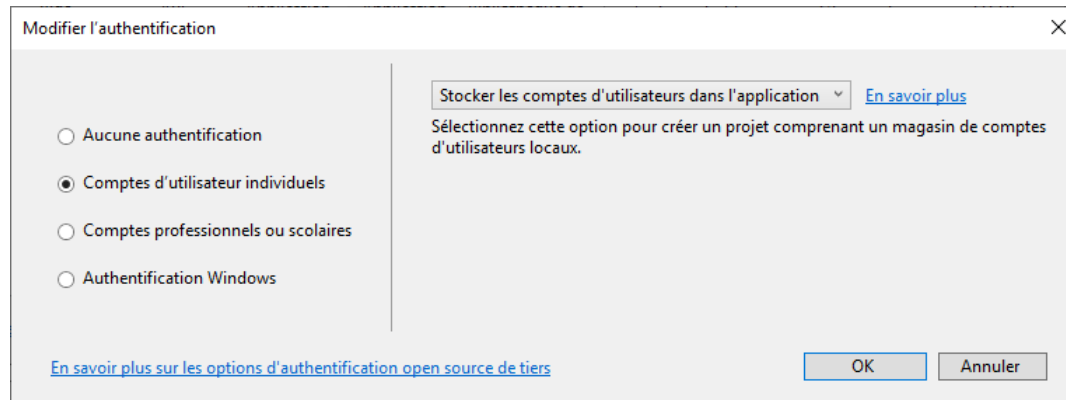
☒ Configurer pour HTTPS

[Modifier l'authentification](#)

OK Annuler

Exemple 1- Création d'un projet

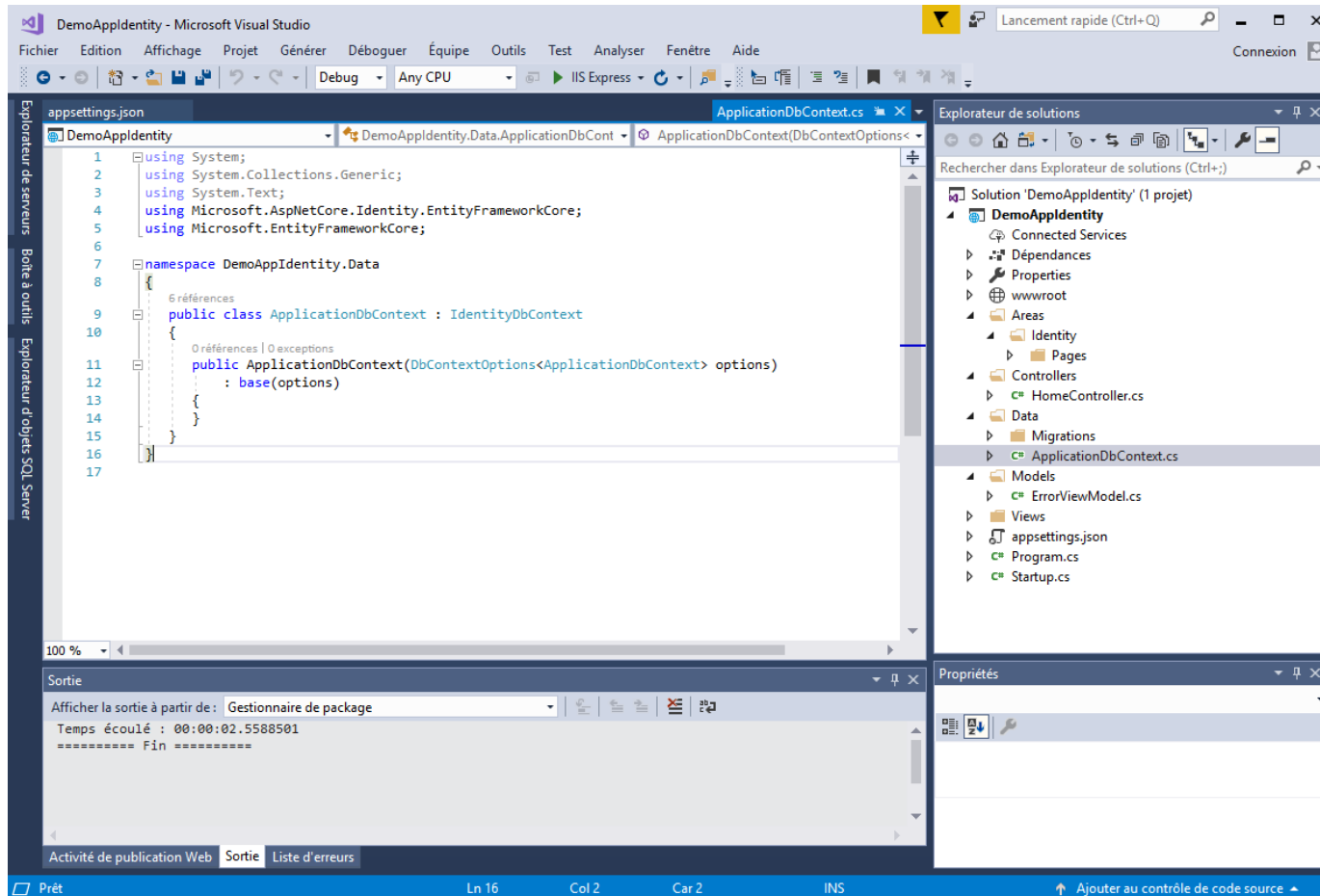
Sélectionnez l'authentification : Comptes d'utilisateur individuels et cliquez sur Ok.



The screenshot shows a dialog box titled "Modifier l'authentification" with a close button (X) in the top right corner. On the left side, there are four radio button options: "Aucune authentification", "Comptes d'utilisateur individuels" (which is selected), "Comptes professionnels ou scolaires", and "Authentification Windows". To the right of these options, there is a dropdown menu labeled "Stocker les comptes d'utilisateurs dans l'application" with a downward arrow, followed by a blue link "En savoir plus". Below this, a text instruction reads: "Sélectionnez cette option pour créer un projet comprenant un magasin de comptes d'utilisateurs locaux." At the bottom left, there is another blue link: "En savoir plus sur les options d'authentification open source de tiers". At the bottom right, there are two buttons: "OK" and "Annuler".

Exemple 1- Création d'un projet

Visual Studio a crée un projet.



Exemple 1- Classe de contexte

Ajout de la classe de contexte :

- ✓ Visual studio à crée le projet avec la classe de contexte **ApplicationDbContext** dans le dossier Data qui hérite de la classe **IdentityDbContext** afin d'ajouter les entités utilisés par Identity.

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

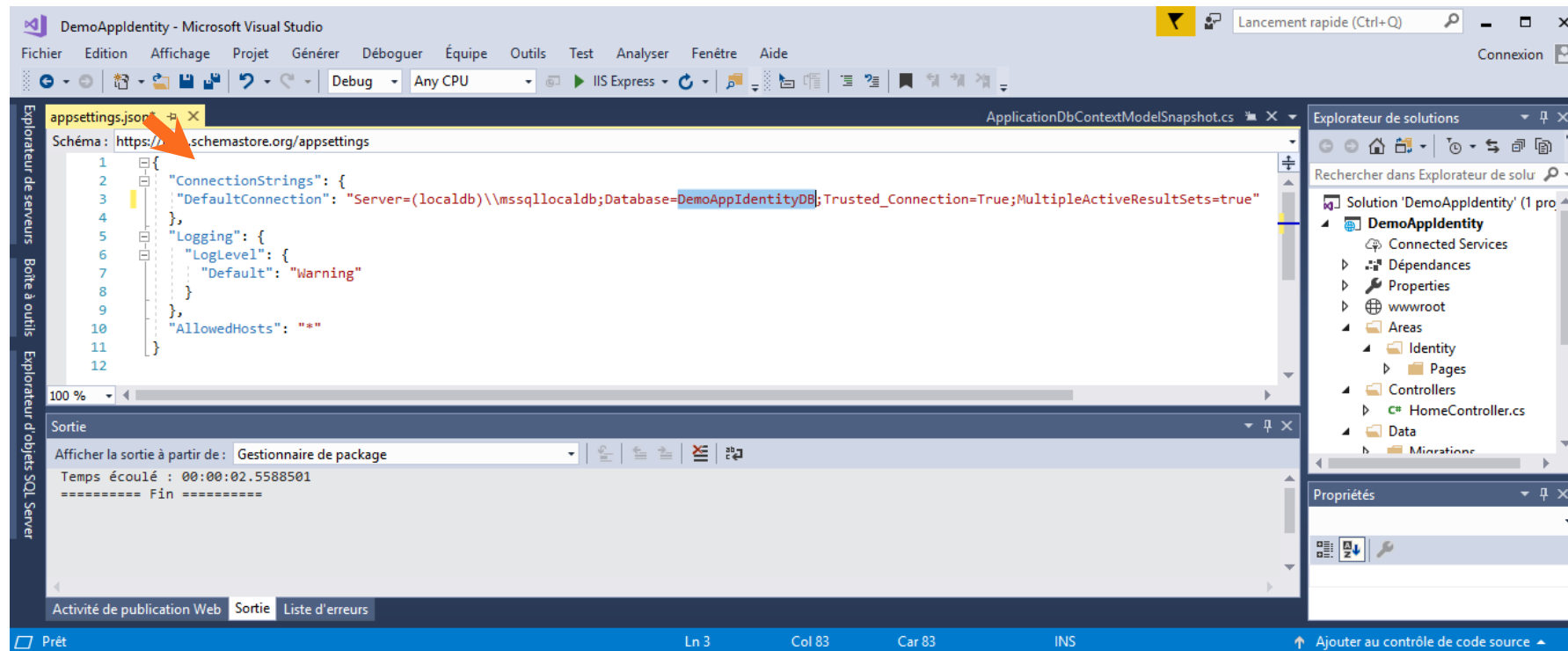
namespace DemoAppIdentity.Data
{
    public class ApplicationDbContext : IdentityDbContext
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
        }
    }
}
```

- ✓ Visual studio a ajouté également un dossier Migrations qui contient un fichier ApplicationDbContextModelSnapshot.cs.

Exemple 1- Chaîne de connexion

Ajout d'une chaîne de connexion à la base de données

✓ Visual studio a ajouté une chaîne de connexion dans le fichier *appsettings.json* :

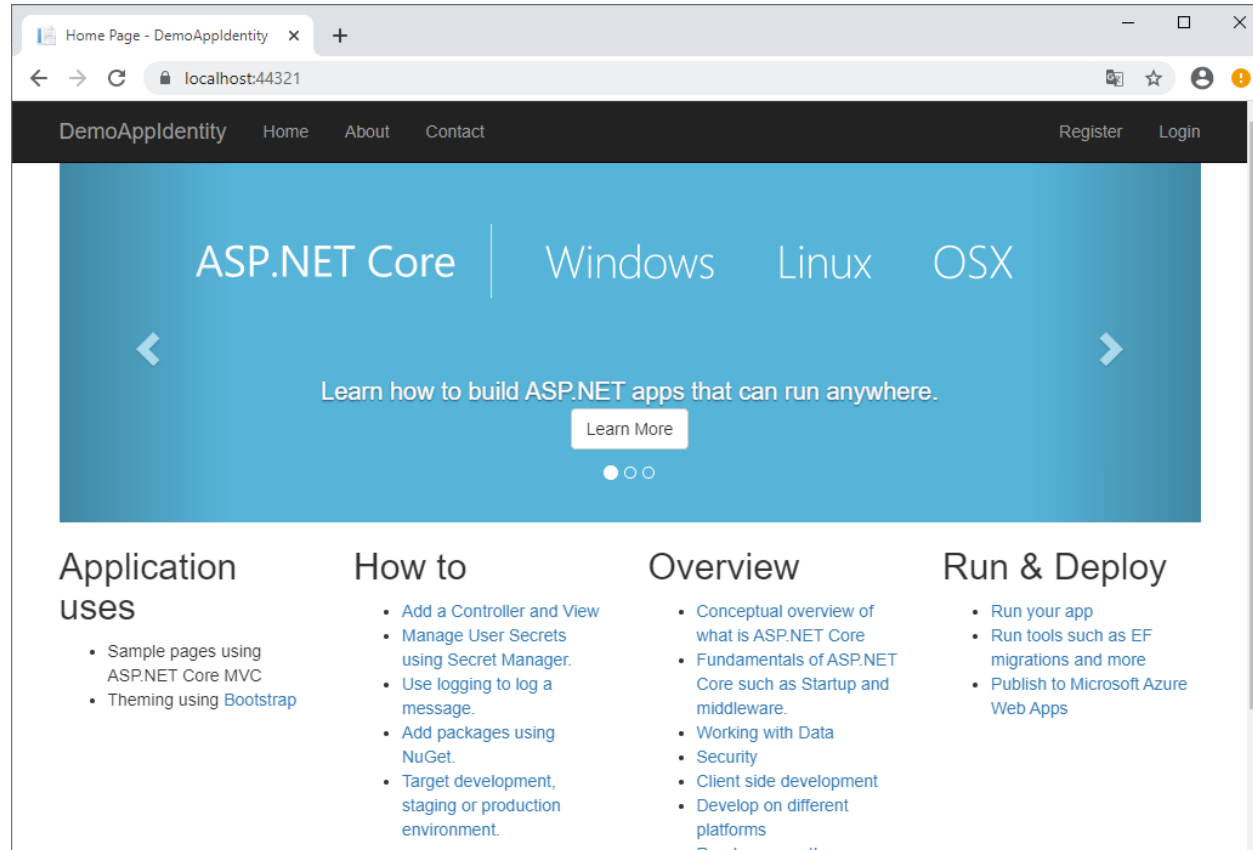


Générez le projet pour la vérification des erreurs du compilateur.

<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-2.1&tabs=visual-studio>

Exemple 1- Exécution

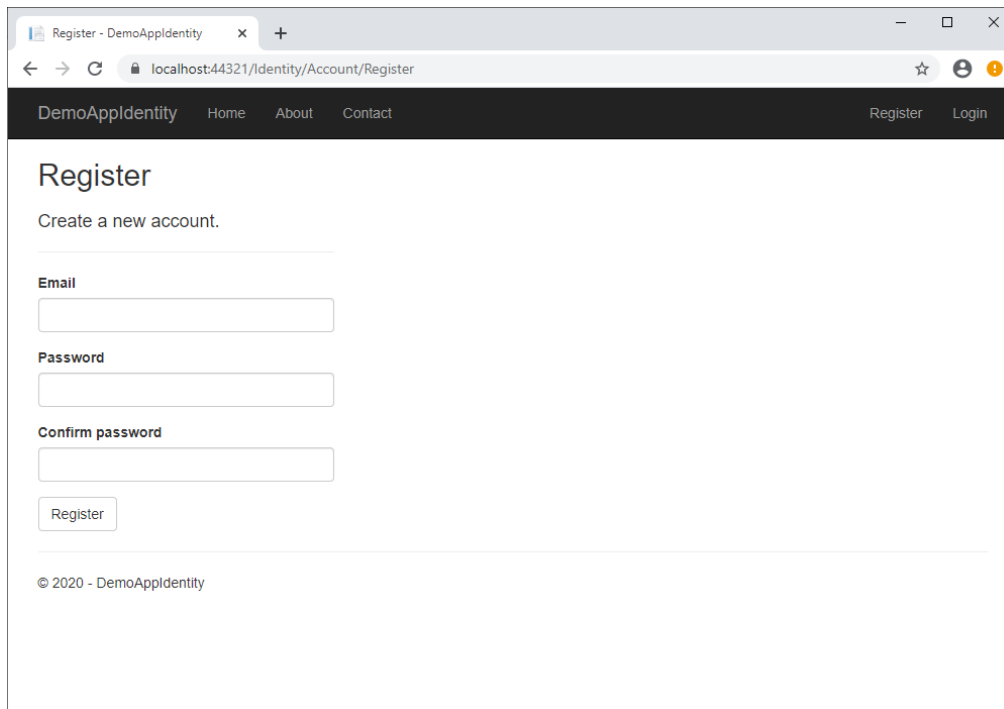
La figure suivante montre la page affichée après l'exécution de l'application :



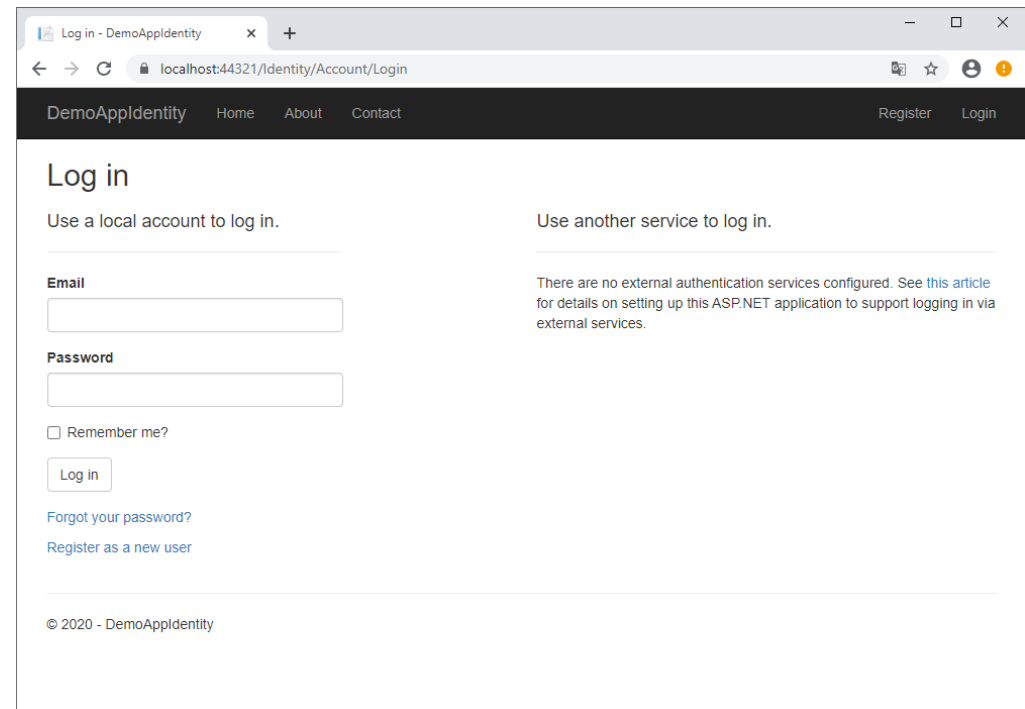
Exemple 1- Authentication

Les figures suivantes montrent les pages affichées après l'exécution de l'application :

- Page d'enregistrement d'un nouveau compte
- Page de connexion d'un utilisateur



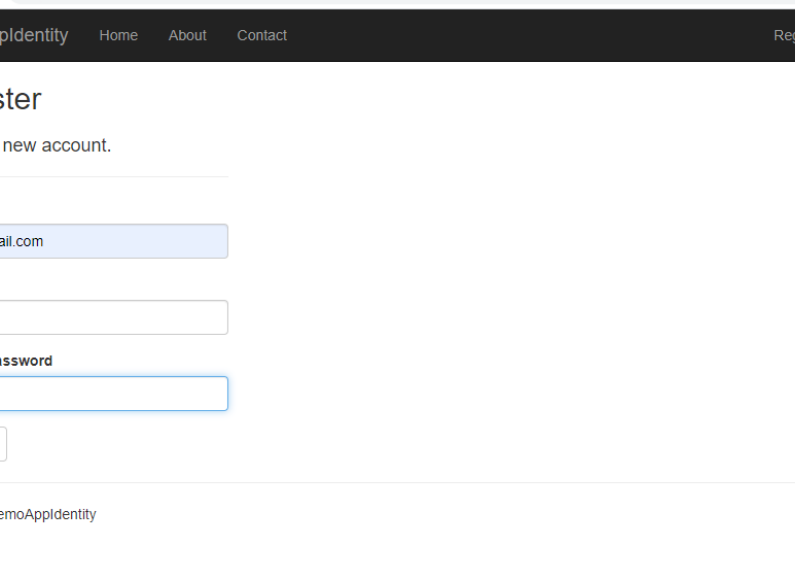
The screenshot shows a web browser window with the title "Register - DemoAppIdentity". The address bar shows the URL "localhost:44321/Identity/Account/Register". The page has a dark header with the site name "DemoAppIdentity" and navigation links "Home", "About", "Contact", "Register", and "Login". The main content area is titled "Register" and contains the text "Create a new account." Below this are three input fields labeled "Email", "Password", and "Confirm password". A "Register" button is at the bottom of the form. The footer shows "© 2020 - DemoAppIdentity".



The screenshot shows a web browser window with the title "Log in - DemoAppIdentity". The address bar shows the URL "localhost:44321/Identity/Account/Login". The page has a dark header with the site name "DemoAppIdentity" and navigation links "Home", "About", "Contact", "Register", and "Login". The main content area is titled "Log in" and contains two sections: "Use a local account to log in." and "Use another service to log in." The local account section has "Email" and "Password" input fields, a "Remember me?" checkbox, and a "Log in" button. Below these are links for "Forgot your password?" and "Register as a new user". The "Use another service to log in." section has a message stating "There are no external authentication services configured. See [this article](#) for details on setting up this ASP.NET application to support logging in via external services." The footer shows "© 2020 - DemoAppIdentity".

Example 1- Authentication

Les figures suivantes montrent les pages affichée si on essaye d'enregistrer un nouveau utilisateur:



The screenshot shows a web browser window with the address bar displaying "localhost:44321/Identity/Account/Register". The page title is "Register - DemoAppIdentity". The browser's address bar shows the URL "localhost:44321/Identity/Account/Register". The page has a dark header bar with the text "DemoAppIdentity" on the left and "Register" and "Login" on the right. The main content area has a light gray background. The heading "Register" is in a large, bold, black font. Below it, the text "Create a new account." is in a smaller, regular black font. There are three form fields: "Email" with the value "test@gmail.com", "Password" with masked characters "*****", and "Confirm password" with masked characters "*****". Each field has a corresponding label above it. Below the "Confirm password" field is a "Register" button. At the bottom of the page, there is a footer with the text "© 2020 - DemoAppIdentity".

Register - DemoAppIdentity

localhost:44321/Identity/Account/Register

DemoAppIdentity Home About Contact Register Login

Register

Create a new account.

Email

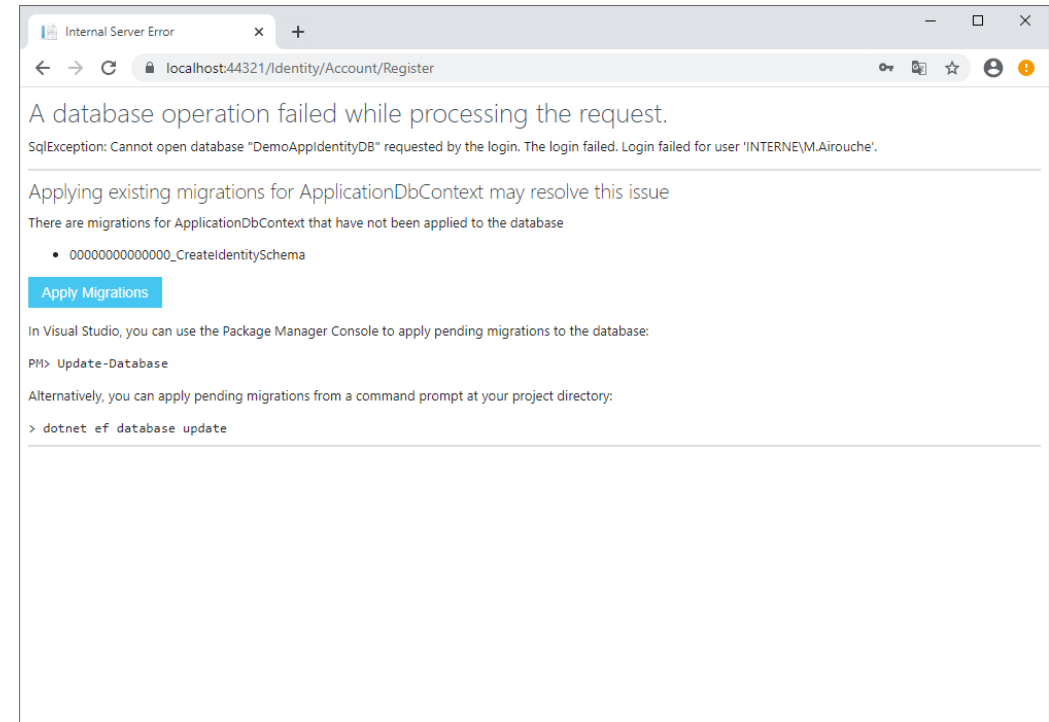
test@gmail.com

Password

Confirm password

Register

© 2020 - DemoAppIdentity



Une erreur sera affichée sur le navigateur pour indiquer qu'il faut appliquer la migration pour créer la base de données. Cliquer sur le bouton Apply Migrations ou dans la console du gestionnaire de package, entrez la commande suivante : [Update-Database](#)

Exemple 2

Développement d'une application de gestion d'employés avec authentification :

On souhaite créer une application qui permet de gérer des employés et les départements de ces employés avec authentification. 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.

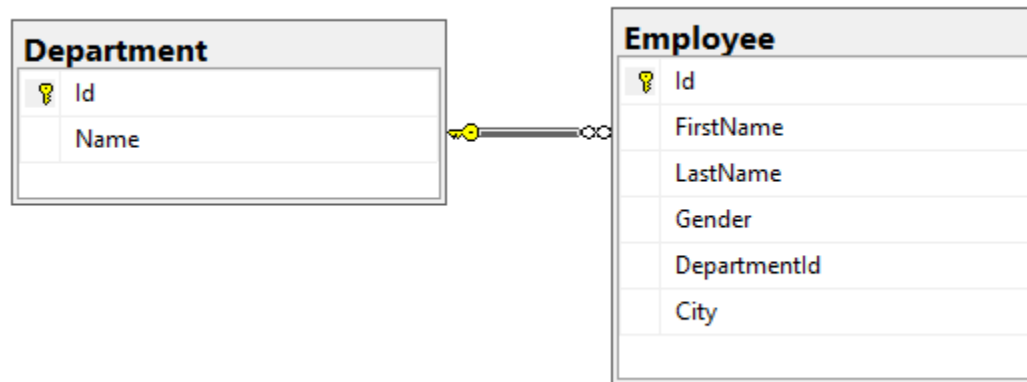
➤ Pour les départements :

- 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 2

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

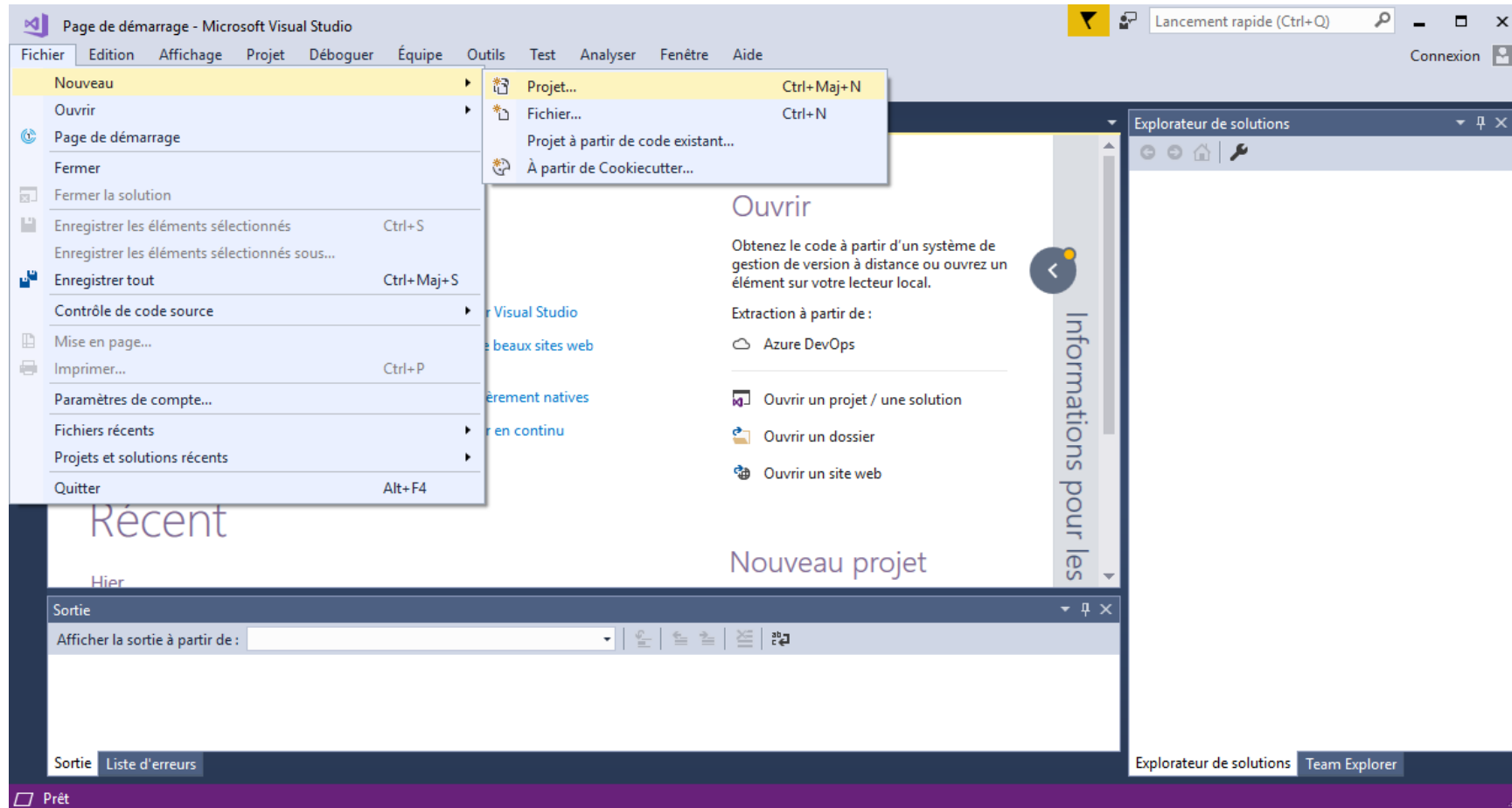
Les employés et les départements seront stockés dans une base de données locale SQL Server Express. Cette base de données doit contenir deux tables (table Employee et table Department).



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

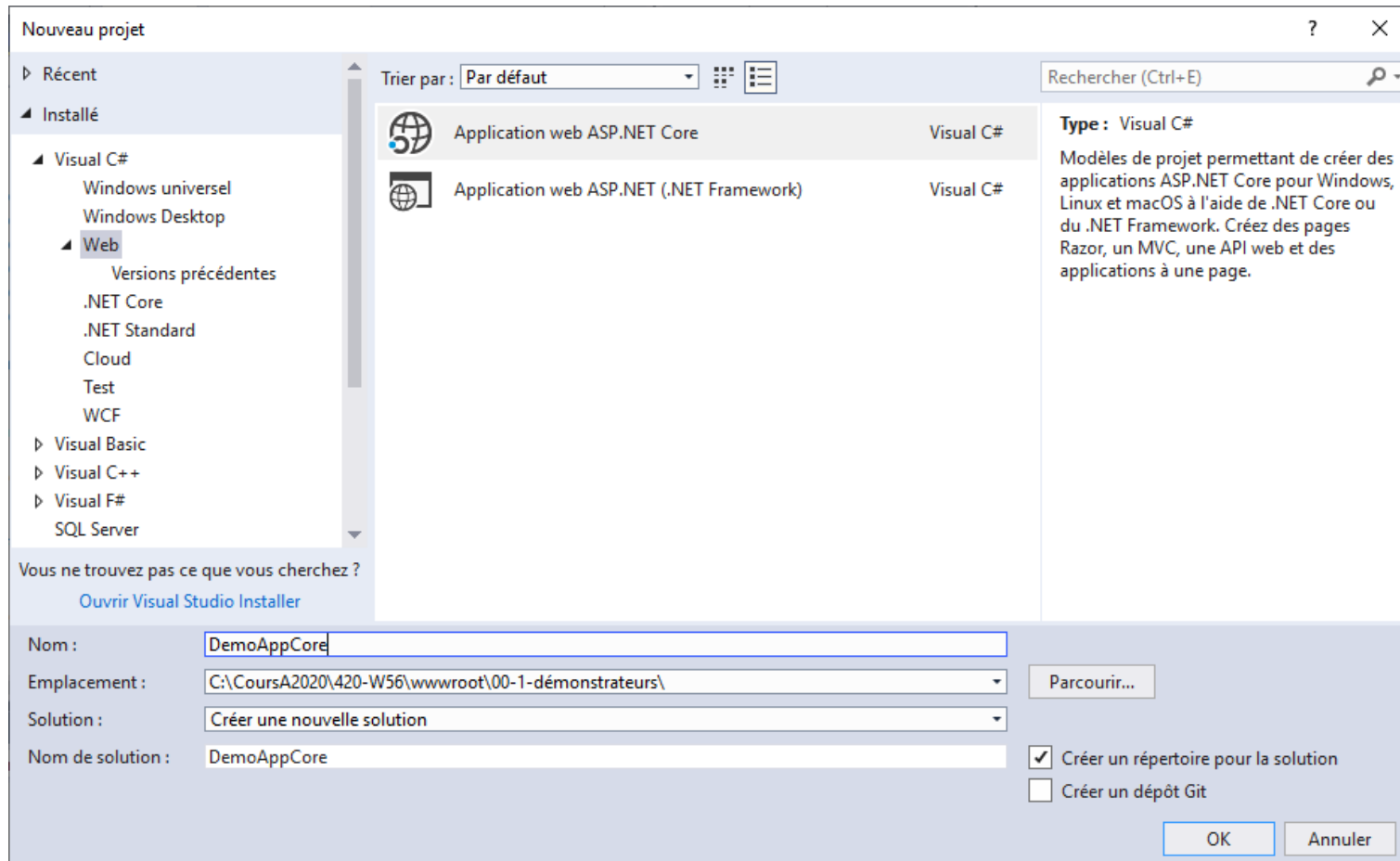
Exemple 2- Création d'un projet

La création d'un projet web MVC utilisant .NET Core s'effectue via le menu **Fichier-Nouveau Projet**



Exemple 2- Création d'un projet

Sélectionnez (Application web ASP.NETCore) et nommez le projet : **DemoAppCore**





Exemple 2- Création d'un projet


Sélectionnez le modèle Application web (Model-View-Controller)


Nouvelle application web ASP.NET Core - - DemoAppCore


.NET Core ASP.NET Core 2.1 [En savoir plus](#)



Vide



API



Application web


Application web (Model-View-Controller)


Bibliothèque de classes Razor


Angular


React.js


React.js et Redux

Modèle de projet permettant de créer une application ASP.NET Core avec des exemples de vues et de contrôleurs ASP.NET Core MVC. Vous pouvez également utiliser ce modèle pour les services HTTP RESTful.

[En savoir plus](#)

Auteur : Microsoft
Source : SDK 2.1.507

Authentification : **Aucune authentification**

[Get additional project templates](#)

☐ Activer le support de Docker (Requires [Docker pour Windows](#))

SE : Windows

☐ Configurer pour HTTPS

[Modifier l'authentification](#)

OK Annuler

Exemple 2- Structure du projet – Classe Startup

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/fundamentals/?view=aspnetcore-3.1&tabs=windows>

Exemple 2- Structure du projet – Classe Startup

```
// 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("/Home/Error");
    }

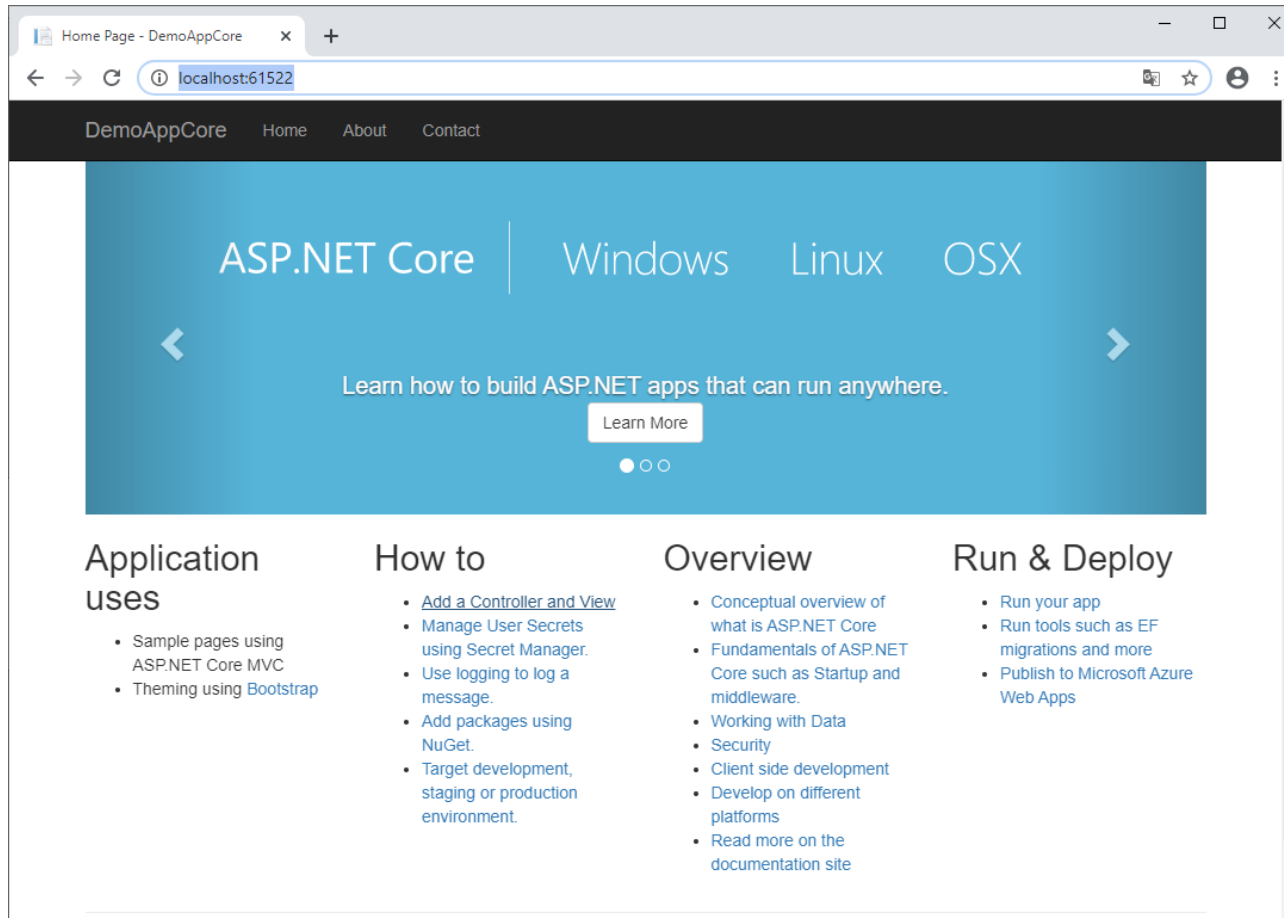
    app.UseStaticFiles();
    app.UseCookiePolicy();

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

<https://docs.microsoft.com/fr-ca/aspnet/core/fundamentals/?view=aspnetcore-3.1&tabs=windows>

Exemple 2- Exécution de l'application

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



Exemple 2

Ajout de modèle à utiliser par Entity Framework Core incluant les entités d'Identity :

- ✓ Il faut ajouter la classe de contexte `EmployeeDbContext` qui étends la classe `DbContextIdentity` afin d'ajouter les entités utilisés par Identity :

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace EmployeeAppCore.Models
{
    public class EmployeeDbContext: IdentityDbContext
    {
        public EmployeeDbContext(DbContextOptions<EmployeeDbContext> options): base(options)
        {
        }
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            base.OnConfiguring(optionsBuilder);
        }
        public DbSet<Employee> Employees { get; set; }

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

Exemple 2

✓ 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 2




Inscrire le contexte de base de données et ajout des services de ASP.NET Core Identity

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 et les services Identity 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("EmployeeConnectionIdentity")));
     services.AddIdentity<IdentityUser, IdentityRole>().AddEntityFrameworkStores<EmployeeDbContext>();


    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
}
```

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

Exemple 2

Ajout de la méthode d'authentification dans le pipeline de requêtes de middlewares

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseStaticFiles();
    app.UseCookiePolicy();
     app.UseAuthentication();
    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}
```


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

Exemple 2

Ajout d'une chaîne de connexion à la base de données

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

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



Générez le projet pour la vérification des erreurs du compilateur.

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

Exemple 2

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 :

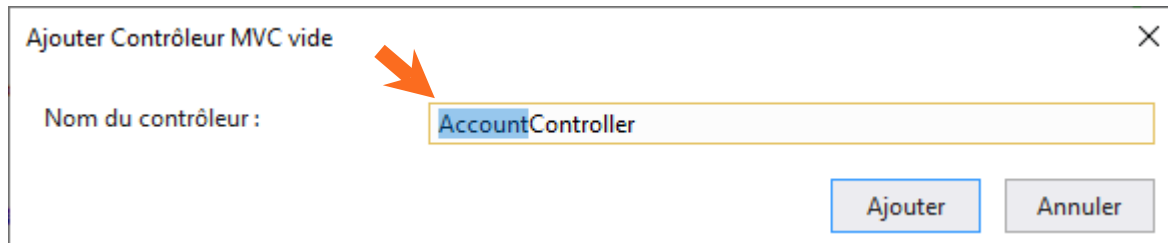
 **Add-Migration** InitialCreate
Update-Database

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

Exemple 2

Ajout d'un Contrôleur Vide

Ajoutez un contrôleur pour la gestion de comptes et la connexion : **Account**



Ajouter Contrôleur MVC vide

Nom du contrôleur :

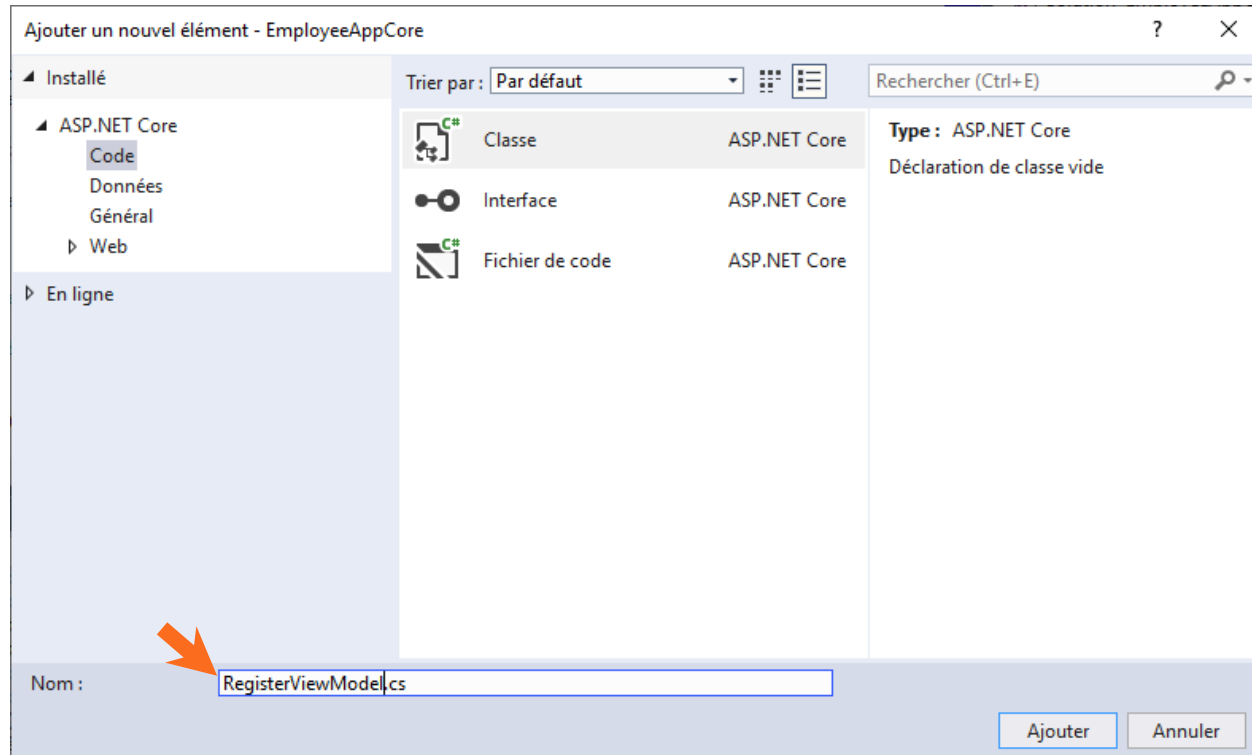
AccountController

Ajouter Annuler

Exemple 2

Ajout d'une Classe pour la page d'enregistrement

Ajoutez une Classe pour la page d'enregistrement : RegisterViewModel



```
namespace EmployeeAppCore.ViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }

        [DataType(DataType.Password)]
        [Display(Name = "Confirm password")]
        [Compare("Password", ErrorMessage = "Password
            and confirmation password do not
            match.")]
        public string ConfirmPassword { get; set; }
    }
}
```

Exemple 2

Ajout des méthodes d'action pour la gestion d'enregistrement

- Utilisez l'injection par constructeur pour ajouter les services : **userManager**, **signInManager**.
- Ajoutez dans le contrôleur **Account** une méthode d'action **HttpGet** pour l'enregistrement :

```
public class AccountController : Controller
{
    private readonly UserManager<IdentityUser> userManager;
    private readonly SignInManager<IdentityUser> signInManager;

    public AccountController(UserManager<IdentityUser> userManager, SignInManager<IdentityUser> signInManager)
    {
        this.userManager = userManager;
        this.signInManager = signInManager;
    }

    [HttpGet]
    public IActionResult Register()
    {
        return View();
    }
}
```

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

Exemple 2

Ajout des méthodes d'action pour la gestion d'enregistrement

➤ Ajoutez dans le contrôleur Account une méthode d'action Httppost pour l'enregistrement :

```
[HttpPost]
public async Task<IActionResult> Register(RegisterViewModel model)
{
    if (ModelState.IsValid)
    {
        // Copy data from RegisterViewModel to IdentityUser
        var user = new IdentityUser
        {
            UserName = model.Email, Email = model.Email
        };
        // Store user data inAspNetUsers database table
        var result = await userManager.CreateAsync(user, model.Password);
        // If user is successfully created, sign-in the user using. SignInManager and redirect to index action of HomeController
        if (result.Succeeded)
        {
            await signInManager.SignInAsync(user, isPersistent: false);
            return RedirectToAction("index", "Employees");
        }
        // If there are any errors, add them to the ModelState object. which will be displayed by the validation summary tag helper
        foreach (var error in result.Errors)
        {
            ModelState.AddModelError(string.Empty, error.Description);
        }
    }
    return View(model);
}
```


Exemple 2

Ajout d'une vue pour l'enregistrement

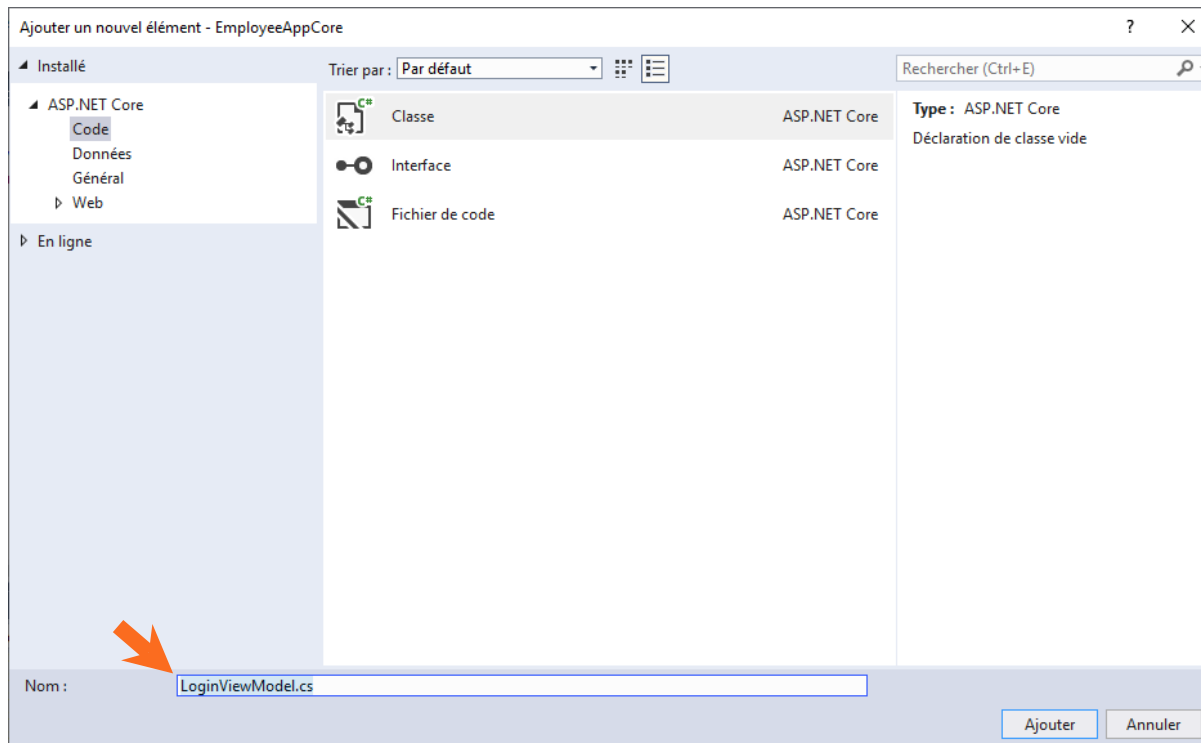
```
@model RegisterViewModel
@{
    ViewBag.Title = "User Registration";
}
<h1>User Registration</h1>

<div class="row">
    <div class="col-md-12">
        <form method="post">
            <div asp-validation-summary="All" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Email"></label>
                <input asp-for="Email" class="form-control" />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Password"></label>
                <input asp-for="Password" class="form-control" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="ConfirmPassword"></label>
                <input asp-for="ConfirmPassword" class="form-control" />
                <span asp-validation-for="ConfirmPassword" class="text-danger"></span>
            </div>
            <button type="submit" class="btn btn-primary">Register</button>
        </form>
    </div>
</div>
```

Exemple 2

Ajout d'une Classe pour la page de connexion

Ajoutez une Classe pour la page de connexion : LoginViewModel



```
namespace EmployeeAppCore.Models
{
    public class LoginViewModel
    {
        [Required]
        [EmailAddress]
        public string Email { get; set; }


        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }


        [Display(Name = "Remember me")]
        public bool RememberMe { get; set; }
    }
}
```

Exemple 2

Ajout des méthodes d'actions pour la gestion de connexion

➤ Ajoutez dans le contrôleur Account les méthodes d'action pour la gestion de connexion :

```
[HttpGet]
public IActionResult Login()
{
    return View();
}

[HttpPost]
public async Task<IActionResult> Login(LoginViewModel model)
{
    if (ModelState.IsValid)
    {
        var result = await signInManager.PasswordSignInAsync(
            model.Email, model.Password, model.RememberMe, false);

        if (result.Succeeded)
        {
            return RedirectToAction("index", "Employees");
        }

        ModelState.AddModelError(string.Empty, "Invalid Login Attempt");
    }

    return View(model);
}
```

Exemple 2


Ajouter d'une vue pour la connexion

```
@model LoginViewModel
@{
    ViewBag.Title = "User Login";
}
<h1>User Login</h1>
<div class="row">
    <div class="col-md-12">
        <form method="post">
            <div asp-validation-summary="All" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Email"></label>
                <input asp-for="Email" class="form-control" />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Password"></label>
                <input asp-for="Password" class="form-control" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>
            <div class="form-group">
                <div class="checkbox">
                    <label asp-for="RememberMe">
                        <input asp-for="RememberMe" />
                        @Html.DisplayNameFor(m => m.RememberMe)
                    </label>
                </div>
            </div>
            <button type="submit" class="btn btn-primary">Login</button>
        </form>
    </div>
</div>
```

Exemple 2

Ajout d'une méthode d'action pour la déconnexion


- Ajoutez dans le contrôleur Account la méthode d'action pour la déconnexion :



```
[HttpPost]
public async Task<IActionResult> Logout()
{
    await signInManager.SignOutAsync();
    return RedirectToAction("index", "home");
}
```

Ajout du code dans la page _Layout pour personnaliser le menu selon la connexion/déconnexion


- Injecter le service **SignInManager** dans la page _Layout :



```
@inject SignInManager<IdentityUser> SignInManager
```

Exemple 2

- Ajoutez de code pour gérer l'affichage de menu dans la page _Layout selon que l'utilisateur est connecté ou non :



```
@if (SignInManager.IsSignedIn(User))
{
    <form method="post" asp-controller="account" asp-action="logout" class="navbar-right">
        <ul class="nav navbar-nav navbar-right">
            <li>
                <button type="submit" class="btn btn-link navbar-btn nav-link ">
                    Logout @User.Identity.Name
                </button>
            </li>
        </ul>
    </form>
}
else
{
    <ul class="nav navbar-nav navbar-right">
        <li>
            <a asp-controller="account" asp-action="register"> Register </a>
        </li>
        <li>
            <a asp-controller="account" asp-action="login"> Login </a>
        </li>
    </ul>
}
```

