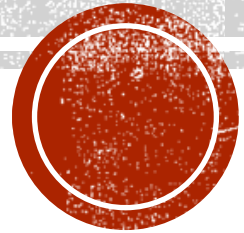
TP N°2

Accès à une base de données SQL SERVER

Avec Entity Framework Core (Approche Code First)

GESTION DES EMPLOYÉS

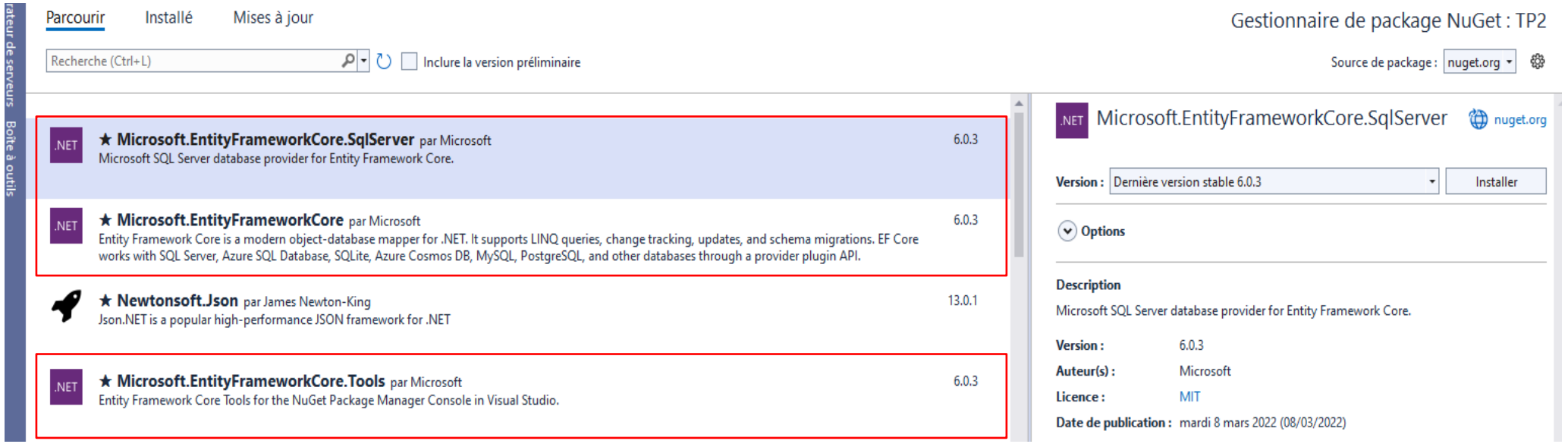


Enseignant :

Malek Zribi

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 :



CRÉATION DES CLASSES DU DOMAINE

- Commençons par créer la classe suivante dans le dossier **Models** :

```
public class Employe
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Departement { get; set; }
        public int Salary { get; set; }

    }
```

- Ensuite, il faut créer la classe de contexte **AppDbContext** qui hérite de **DBContext** pour pouvoir communiquer avec une base de données.

```
using Microsoft.EntityFrameworkCore;
namespace TP2.Models {
    public class AppDbContext : DbContext
    {
        public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)
        {
        }
        public DbSet<Employe> Employees { get; set; }
    }
}
```

CONFIGURATION DU FOURNISSEUR DE BD

- Nous pouvons utiliser la *méthode AddDbContext ()* ou *AddDbContextPool ()* pour inscrire notre classe DbContext spécifique à l'application avec le système d'injection de dépendances ASP.NET Core au niveau de la classe **Program.cs :**

```
builder.Services.AddDbContextPool<AppDbContext>(options =>
options.UseSqlServer(builder.Configuration.GetConnectionString("EmployeeDBConnection
")));
```

- La chaîne de connexion nommée "EmployeeDBConnection" doit être définit dans le fichier de configuration du projet ***appsettings.json*** plutôt que dans le code. Pour cela, il faut ajouter cette clé dans le fichier **appsettings.json :**

```
"ConnectionStrings": {
    "EmployeeDBConnection":
"server=(localdb)\\MSSQLLocalDB;database=EmployeeDB;Trusted_Connection=true"

  }
```

CRÉATION DES CLASSES DU REPOSITORY

- Créer un dossier **Repositories** sous le dossier **Models**.
- Créer l'interface **IEmployeeRepository** suivante :

```
public interface IEmployeeRepository
    {
        Employee GetEmployee(int Id);
        IEnumerable<Employee> GetAllEmployee();
        Employee Add(Employee employee);
        Employee Update(Employee employeeChanges);
        Employee Delete(int Id);
    }
```

- Créer maintenant la classe **SqlEmployeeRepository** dans laquelle on va ajouter toutes les méthodes d'accès à la base de donnée. Cette classe implémente l'interface IEmployeeRepository :

CRÉATION DES CLASSES DU REPOSITORY

```
using Microsoft.EntityFrameworkCore;
…
public class SQLEmployeeRepository : IEmployeeRepository
    {
        private readonly AppDbContext context;
        public SQLEmployeeRepository(AppDbContext context)
        {
            this.context = context;
        }
        public Employee Add(Employee employee)
        {
            context.Employees.Add(employee);
            context.SaveChanges();
            return employee;
        }
        public Employee Delete(int Id)
        {
            Employee employee = context.Employees.Find(Id);
            if (employee != null)
            {
                context.Employees.Remove(employee);
                context.SaveChanges();
            }
            return employee;
        }
```

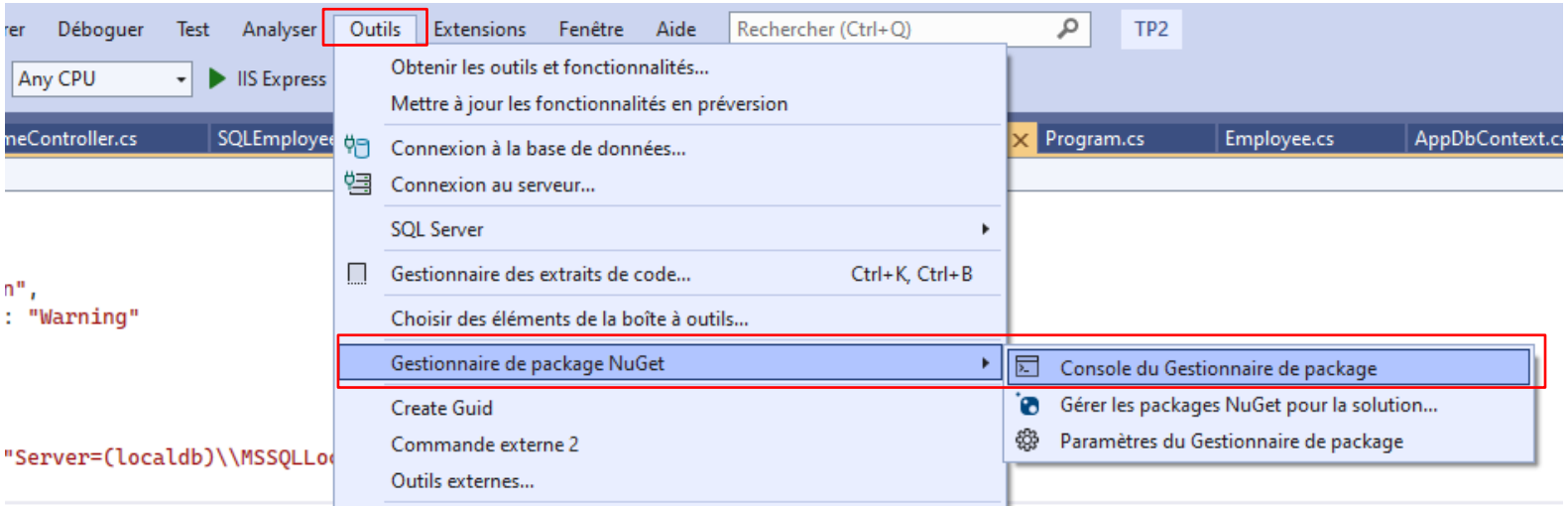
```
public IEnumerable<Employee> GetAllEmployee()
{
        return context.Employees;
}
public Employee GetEmployee(int Id)
{
        return context.Employees.Find(Id);
}
```

INJECTION DES DÉPENDANCES ET CRÉATION DE LA BD

- N'oubliez pas d'injecter les dépendances dans le code de la classe **Program.cs** comme suit :

```
builder.Services.AddScoped<IEmployeeRepository, SQLEmployeeRepository>();
```

- Pour créer la base de données, il faut appliquer la migration comme suit dans la **console du gestionnaire de package NuGet** à partir du **menu** Options :



CRÉATION DE LA BD (SUITE)

- La commande suivante crée la migration initiale. InitialCreate est le nom de la migration.

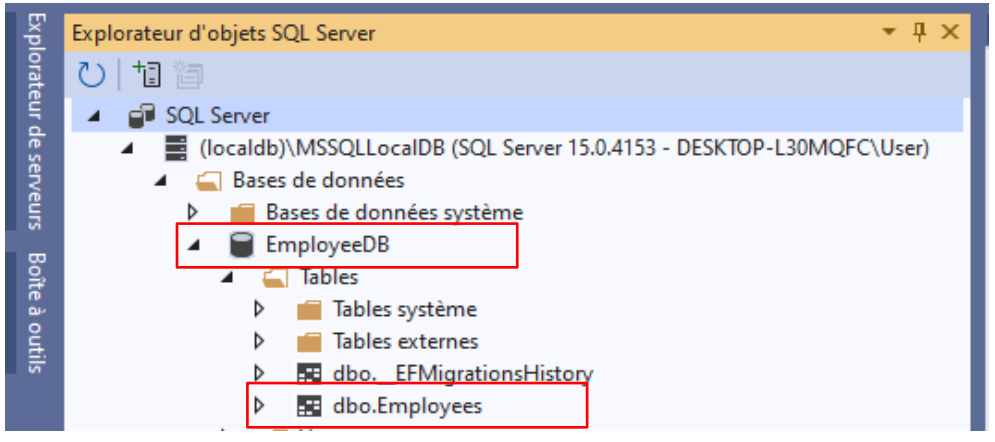
```
PM> Add-Migration InitialCreate
```

- Lorsque la commande ci-dessus se termine, vous verrez un fichier dans le dossier "Migrations" qui contient le nom InitialCreate.cs . Ce fichier contient le code requis pour créer les tables de la base de données.

- Pour mettre à jour la base de données, nous utilisons la commande *Update-Database*.

```
PM> Update-Database
```

- Vous pouvez maintenant aller à l'explorateur d'objets SQL SERVER, et vérifier si la base de données a été bien créée avec la table Employee.



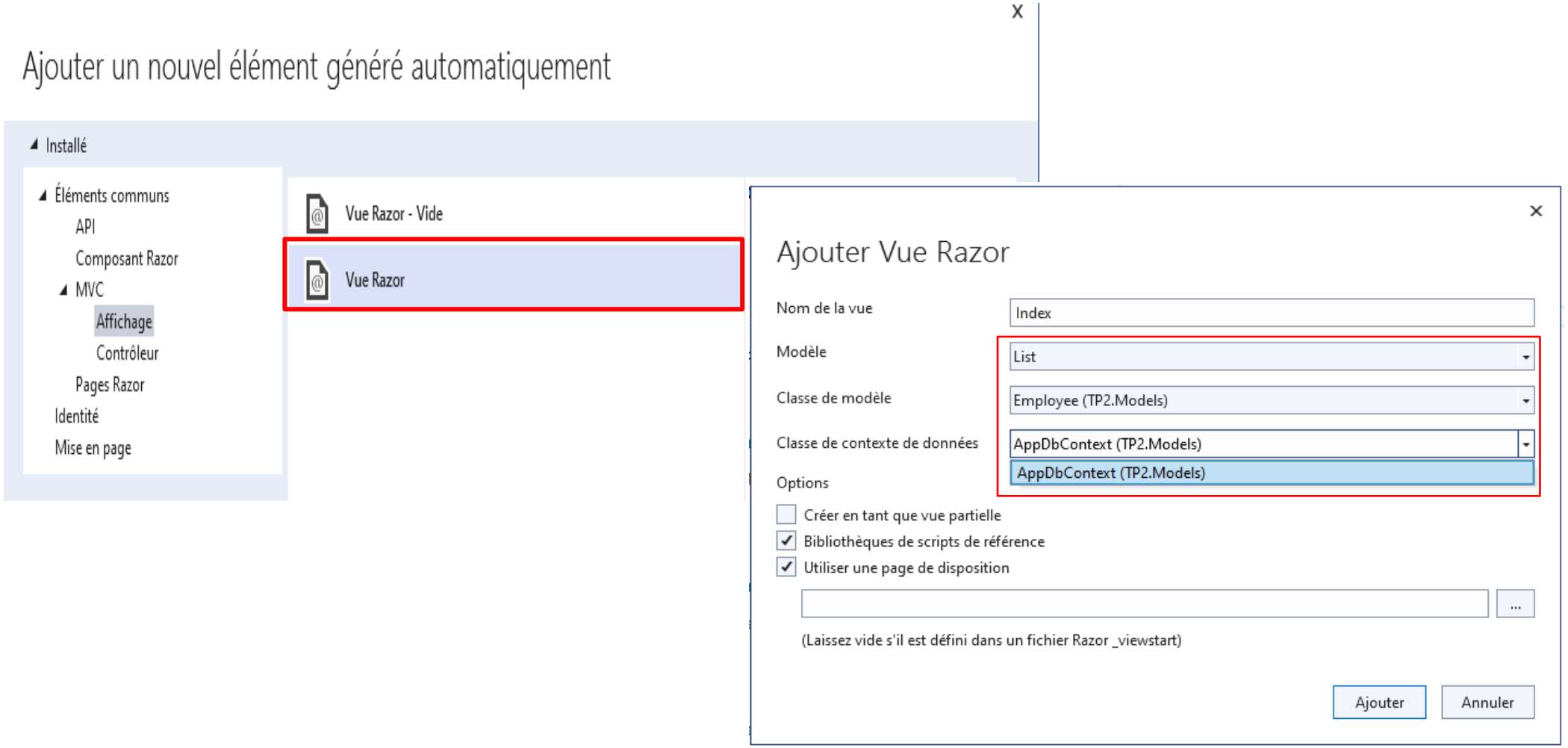
CRÉER UN CONTRÔLEUR

- Dans le dossier Controllers, Créer un nouveau contrôleur nommé **EmployeeController** (avec read/write actions) permettant de gérer les opérations sur les employés. Compléter le code des méthodes d'action en se basant sur le TP N°1.

```
1 référence
public class EmployeeController : Controller
{
    private IEmployeeRepository employeeRepository;

    0 références
    public EmployeeController(IEmployeeRepository employeeRepository)
    {
        this.employeeRepository = employeeRepository;
    }

    3 références
    public ViewResult Index()
    {
        var model = employeeRepository.GetAllEmployee();
        return View(model);
    }
    //Complèter le code ....
```

CRÉER LES VUES

- Passons maintenant à la création des vues de notre application, et commençons par la vue de la méthode Index, puis compléter les vues Create, Edit, Delete et Details :



AJOUTER UNE MIGRATION

- La classe Employee a la structure suivante :

```
public class Employee
{
    8 références
    public int Id { get; set; }
    12 références
    public string Name { get; set; }
    12 références
    public string Departement { get; set; }
    12 références
    public int Salary { get; set; }

}
```

- Si on veut changer la classe en ajoutant une propriété **PhotoPath** *:*

```
33 références
public class Employee
{
    8 références
    public int Id { get; set; }
    12 références
    public string Name { get; set; }
    12 références
    public string Departement { get; set; }
    12 références
    public int Salary { get; set; }
    0 références
    public string PhotoPath { get; set; }

}
```

AJOUTER UNE MIGRATION

- Pour grader la synchronisation entre la base de données et les classes de modèle, il faut lancer une Migration via la commande Add-Migration.

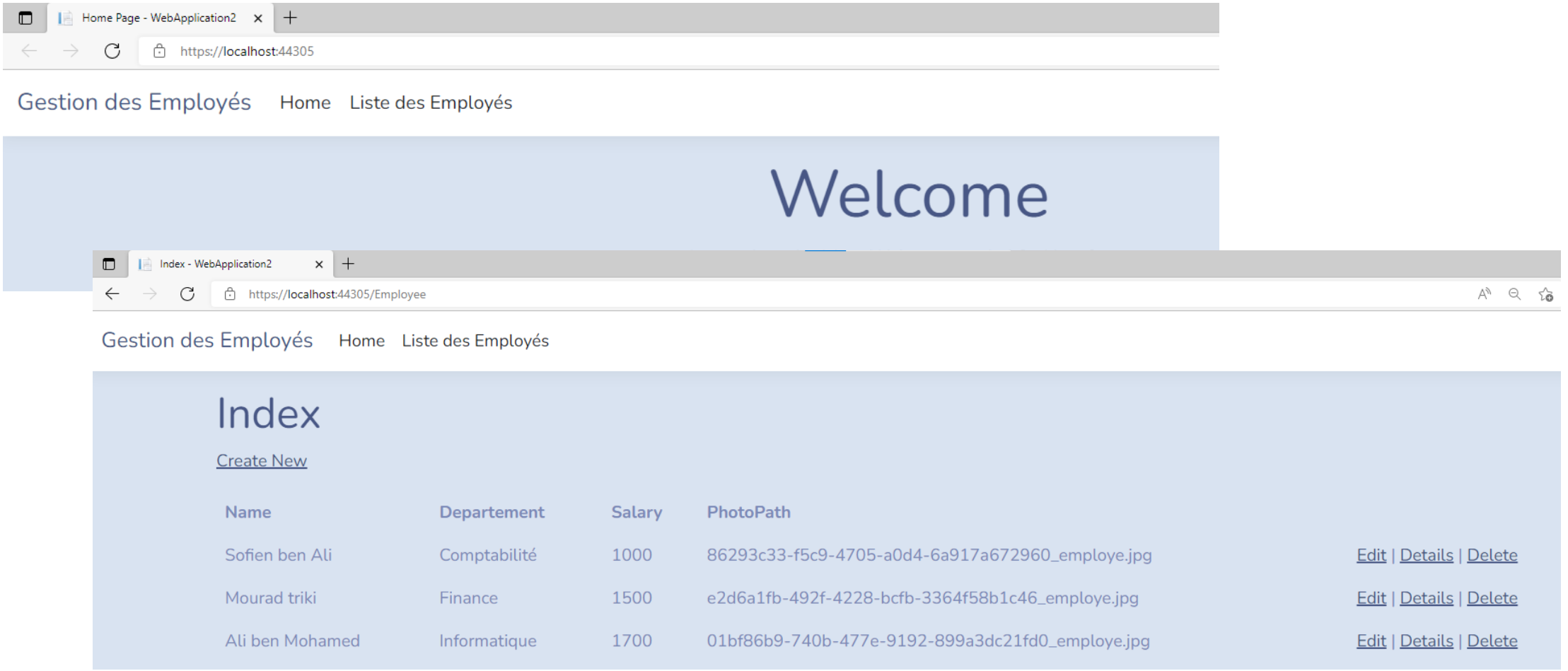
```
PM> Add-Migration AddPhotoPathToEmployees
```

- Pour appliquer la migration et mettre à jour la base de données, utilisez la commande suivante :

```
PM> Update-Database
```

CRÉER UN CONTRÔLEUR

- L'exécution donne comme résultat les pages suivantes :



UPLOAD FILE

- Pour pouvoir faire un upload il faut créer un attribut de type **IFormFile**. Comme il n'est pas pratique de déclarer cet attribut dans notre classe de modèle Employee, on a besoin alors de créer une classe ViewModel.
- Commençons alors par créer un dossier **ViewModels**.
- Ensuite, créer la classe **CreateViewModel** suivante :

```
using System.ComponentModel.DataAnnotations;
namespace TP2.ViewModels
{
    public class CreateViewModel
    {
        [Required]
        [MaxLength(20, ErrorMessage = "Taille Max 50 cc")]
        public string Name { get; set; }
        public string Department { get; set; }
        [Range(300, 5000, ErrorMessage = "Doit être entre 300 et 5000")]
        public int Salary { get; set; }
        public IFormFile Photo { get; set; }
    }
}
```

UPLOAD FILE

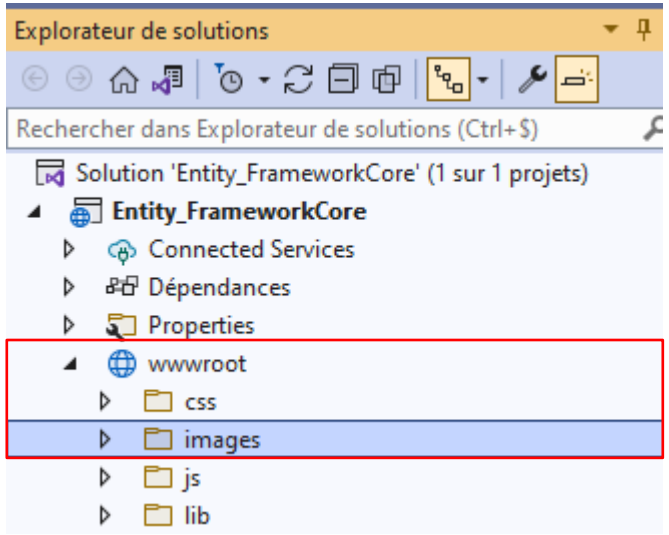
- Nous allons passer à modifier la méthode d'action Create du contrôlleur.

```
private IEmployeeRepository employeeRepository;
private readonly IWebHostEnvironment hostingEnvironment;
public EmployeeController(IEmployeeRepository employeeRepository, IWebHostEnvironment hostingEnvironment) {
            this.employeeRepository = employeeRepository;
            this.hostingEnvironment = hostingEnvironment;

}
//Reste du code ...
public ActionResult Create(CreateViewModel model) {
            if (ModelState.IsValid)
            {
                string uniqueFileName = null;

                // If the Photo property on the incoming model object is not null, then the user has selected an image to upload.
                if (model.Photo != null)
                {
                    // The image must be uploaded to the images folder in wwwroot
                    // To get the path of the wwwroot folder we are using the inject
                    // HostingEnvironment service provided by ASP.NET Core

                    string uploadsFolder = Path.Combine(hostingEnvironment.WebRootPath, "images");

                    // To make sure the file name is unique we are appending a new
                    // GUID value and an underscore to the file name

                    uniqueFileName = Guid.NewGuid().ToString() + "_" + model.Photo.FileName;
                    string filePath = Path.Combine(uploadsFolder, uniqueFileName);

                    // Use CopyTo() method provided by IFormFile interface to
                    // copy the file to wwwroot/images folder

                    model.Photo.CopyTo(new FileStream(filePath, FileMode.Create));
                }
```

UPLOAD FILE

```
Employee newEmployee = new Employee
                {
                    Name = model.Name,
                    Salary = model.Salary,
                    Departement = model.Department,
                    // Store the file name in PhotoPath property of the employee object
                    // which gets saved to the Employees database table
                    PhotoPath = uniqueFileName
                };

                employeeRepository.Add(newEmployee);
                return RedirectToAction("details", new { id = newEmployee.Id });
            }
            return View();
}
```

- Ajouter un nouveau sous dossier nommé **images** dans le dossier **wwwroot** .



UPLOAD FILE

- Maintenant, nous allons remplacer le code de la vue par le code suivant :

```
@model TP2.ViewModels.CreateViewModel

@{
    ViewData["Title"] = "Create";
}

<h1>Create</h1>

<h4>Employee</h4>
<hr />
<div class="row">
    <div class="col-md-4">
         @*To support file upload set the form element enctype="multipart/form-data" *@
        <form enctype="multipart/form-data" asp-action="Create">
            <div asp-validation-summary="All" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Name" class="control-label"></label>
                <input asp-for="Name" class="form-control" />
                <span asp-validation-for="Name" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Department" class="control-label"></label>
                <input asp-for="Department" class="form-control" />
                <span asp-validation-for="Department" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Salary" class="control-label"></label>
                <input asp-for="Salary" class="form-control" />
                <span asp-validation-for="Salary" class="text-danger"></span>
            </div>
           @*  asp-for tag helper is set to "Photo" property. "Photo" property type is IFormFile so at runtime asp.net core generates file
upload control (input type=file)*@
```

UPLOAD FILE

```
 <div class="form-group row">
                <label asp-for="Photo" class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <div class="custom-file">
                        <input asp-for="Photo" class="form-control custom-file-input">
                        <label class="custom-file-label">Choose File...</label>
                    </div>
                </div>
            </div>
            <div class="form-group">
                <input type="submit" value="Create" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>

<div>
    <a asp-action="Index">Back to List</a>
</div>

@*@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}*@

@*This script is required to display the selected file in the file upload element*@

@section Scripts {
    <script>
            $(document).ready(function () {
                $('.custom-file-input').on("change", function () {
                    var fileName = $(this).val().split("\\").pop();
                    $(this).next('.custom-file-label').html(fileName);
                });
            });
    </script>
}
```

UPLOAD FILE

- Le code de la vue **Details** sera le suivant :

```
@model TP2.Models.Employee
@{
    ViewData["Title"] = "Details";
    var photoPath = "~/images/" + (Model.PhotoPath ?? "noimage.jpg");
}
<h1>Details</h1>
<div>
    <h4>Employee</h4>
    <hr />
    <dl class="row">
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Name)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Name)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Departement)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Departement)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Salary)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Salary)
        </dd>
        <dt class="col-sm-2">
            <label><b>Photo</b> </label>
        </dt>
        <dd class="col-sm-10">
            <div class="card-body text-left">
                <img src="@photoPath" asp-append-version="true" height="100" width="100" />
            </div>
        </dd>
    </dl>
</div>
```

MODIFICATION D'UN EMPLOYÉ

- Pour pouvoir modifier l'image d'un employé, allons suivre les étapes suivantes :
- Créer une classe **EditViewModel** sous le répertoire **ViewModels** dont le code est le suivant :

```
public class EditViewModel : CreateViewModel
    {
        public int Id { get; set; }
        public string ExistingPhotoPath { get; set; }
    }
```

- Remplacer le code de la méthode **Edit** du contrôleur par le code suivant :

```
// GET: EmployeeController/Edit/5
        public ActionResult Edit(int id)
        {
            Employee employee = employeeRepository.GetEmployee(id);
            EditViewModel employeeEditViewModel = new EditViewModel
            {
                Id = employee.Id,
                Name = employee.Name,
                Salary = employee.Salary,
                Department = employee.Departement,
                ExistingPhotoPath = employee.PhotoPath
            };
            return View(employeeEditViewModel);
        }
```

MODIFICATION D'UN EMPLOYÉ

```
// POST: EmployeeController/Edit/5
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Edit(EditViewModel model) {
            // Check if the provided data is valid, if not rerender the edit view
            // so the user can correct and resubmit the edit form
            if (ModelState.IsValid)
            {
                // Retrieve the employee being edited from the database
                Employee employee = employeeRepository.GetEmployee(model.Id);
                // Update the employee object with the data in the model object
                employee.Name = model.Name;
                employee.Salary = model.Salary;
                employee.Departement = model.Department;
                // If the user wants to change the photo, a new photo will be
                // uploaded and the Photo property on the model object receives
                // the uploaded photo. If the Photo property is null, user did
                // not upload a new photo and keeps his existing photo
                if (model.Photo != null)
                {
                    // If a new photo is uploaded, the existing photo must be
                    // deleted. So check if there is an existing photo and delete
                    if (model.ExistingPhotoPath != null)
                    {
                        string filePath = Path.Combine(hostingEnvironment.WebRootPath,"images", model.ExistingPhotoPath);
                        System.IO.File.Delete(filePath);
                    }
                    // Save the new photo in wwwroot/images folder and update
                    // PhotoPath property of the employee object which will be
                    // eventually saved in the database
                    employee.PhotoPath = ProcessUploadedFile(model);
                }
```

MODIFICATION D'UN EMPLOYÉ

```
// Call update method on the repository service passing it the
                // employee object to update the data in the database table
                Employee updatedEmployee = employeeRepository.Update(employee);
                if (updatedEmployee != null)
                    return RedirectToAction("index");
                else
                    return NotFound();
            }

            return View(model);
        }
        [NonAction]
        private string ProcessUploadedFile(EditViewModel model)
        {
            string uniqueFileName = null;

            if (model.Photo != null)
            {
                string uploadsFolder = Path.Combine(hostingEnvironment.WebRootPath, "images");
                uniqueFileName = Guid.NewGuid().ToString() + "_" + model.Photo.FileName;
                string filePath = Path.Combine(uploadsFolder, uniqueFileName);
                using (var fileStream = new FileStream(filePath, FileMode.Create))
                {
                    model.Photo.CopyTo(fileStream);
                }
            }

            return uniqueFileName;
        }
```

MODIFICATION D'UN EMPLOYÉ

- Le code de la vue **Edit** sera le suivant :

```
@model TP2.ViewModels.EditViewModel
@{
    ViewBag.Title = "Edit Employee";
    // Get the full path of the existing employee photo for display
    var photoPath = "~/images/" + (Model.ExistingPhotoPath ?? "noimage.jpg");
}
<form asp-controller="Employee" asp-action="edit" enctype="multipart/form-data" method="post" class="mt-3">
    <div asp-validation-summary="All" class="text-danger">
    </div>
    @*Use hidden input elements to store employee id and ExistingPhotoPath
        which we need when we submit the form and update data in the database*@
    <input hidden asp-for="Id" />
    <input hidden asp-for="ExistingPhotoPath" />

    @*Bind to the properties of the EmployeeEditViewModel. The asp-for tag helper
        takes care of displaying the existing data in the respective input elements*@
    <div class="form-group row">
        <label asp-for="Name" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <input asp-for="Name" class="form-control" placeholder="Name">
            <span asp-validation-for="Name" class="text-danger"></span>
        </div>
    </div>
    <div class="form-group row">
        <label asp-for="Salary" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <input asp-for="Salary" class="form-control" placeholder="Email">
            <span asp-validation-for="Salary" class="text-danger"></span>
        </div>
    </div>
   <div class="form-group row">
        <label asp-for="Department" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <input asp-for="Department" class="form-control" placeholder="Department">
            <span asp-validation-for="Department" class="text-danger"></span>
        </div>
    </div>
```

MODIFICATION D'UN EMPLOYÉ

```
<div class="form-group row">
        <label asp-for="Photo" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <div class="custom-file">
                <input asp-for="Photo" class="custom-file-input form-control">
                <label class="custom-file-label">cliquer ici pour changer la photo</label>
            </div>
        </div>
    </div>

    @*Display the existing employee photo*@
    <div class="form-group row col-sm-4 offset-4">
        <img src="@photoPath" asp-append-version="true" height="100" width="100"/>
    </div>

    <div class="form-group row">
        <div class="col-sm-10">
            <button type="submit" class="btn btn-primary">Update</button>
            <a asp-action="index" asp-controller="Employee" class="btn btn-primary">Cancel</a>
        </div>
    </div>

    @section Scripts {
        <script>
            $(document).ready(function () {
                $('.custom-file-input').on("change", function () {
                    var fileName = $(this).val().split("\\").pop();
                    $(this).next('.custom-file-label').html(fileName);
                });
            });
        </script>
    }
</form>
```

NOUVEL AFFICHAGE DE LA LISTE

- Pour afficher la liste des employés avec leurs images, nous allons utiliser la classe bootstrap card-group. Le nouveau code de la vue Index est le suivant :

```
@model IEnumerable<Employee>
@{
    ViewBag.Title = "Employee List";
}
<div class="card-group">
    @foreach (var employee in Model)
    {
        var photoPath = "~/images/" + (employee.PhotoPath ?? "noimage.jpg");
        <div class="card m-3" style="min-width: 18rem; max-width:30.5%;">
            <div class="card-header">
                <h5><b>Name : </b> @employee.Name</h5>
                <h5><b>Departement :</b>@employee.Departement</h5>
                <h5><b>Salaire :</b> @employee.Salary</h5>
            </div>

            <img class="card-img-top imageThumbnail" src="@photoPath"
                 asp-append-version="true" />

            <div class="card-footer text-center">
                <a asp-controller="Employee" asp-action="Details" asp-route-id="@employee.Id"
                   class="btn btn-primary m-1">View</a>
                <a asp-action="Edit" asp-controller="Employee" class="btn btn-primary m-1" asp-route-id="@employee.Id">Edit</a>
                <a asp-action="Delete" asp-controller="Employee" class="btn btn-danger m-1" asp-route-id="@employee.Id">Delete</a>
            </div>
        </div>
    }
</div>
```

NOUVEL AFFICHAGE DE LA LISTE

- En exécutant, la liste des employés aura la forme suivante. Ajouter le code nécessaire dans le fichier « **_layout.cshtml** » pour ajouter un élément dans le Navbar « **Nouvel Employé** » permettant d'exécuter l'action **Create** du contrôleur :