

Programmation avancée en c#.NET

Ing.Meryem OUARRACHI



Plan du module

Programmation WEB

- ☐ Généralités outils Web
- ☐ ASP MVC
- ☐ **ASP.Net core**
- ☐ Angular en ASP.Net Core

Génie logiciel en .Net

BI en Self Service

Programmation distribuée avancée

- ☐ Web API
- ☐ GraphQL

CHAPITRE 7:

.Net Core

Plan du chapitre

☐ .Net Core

☐ Docker

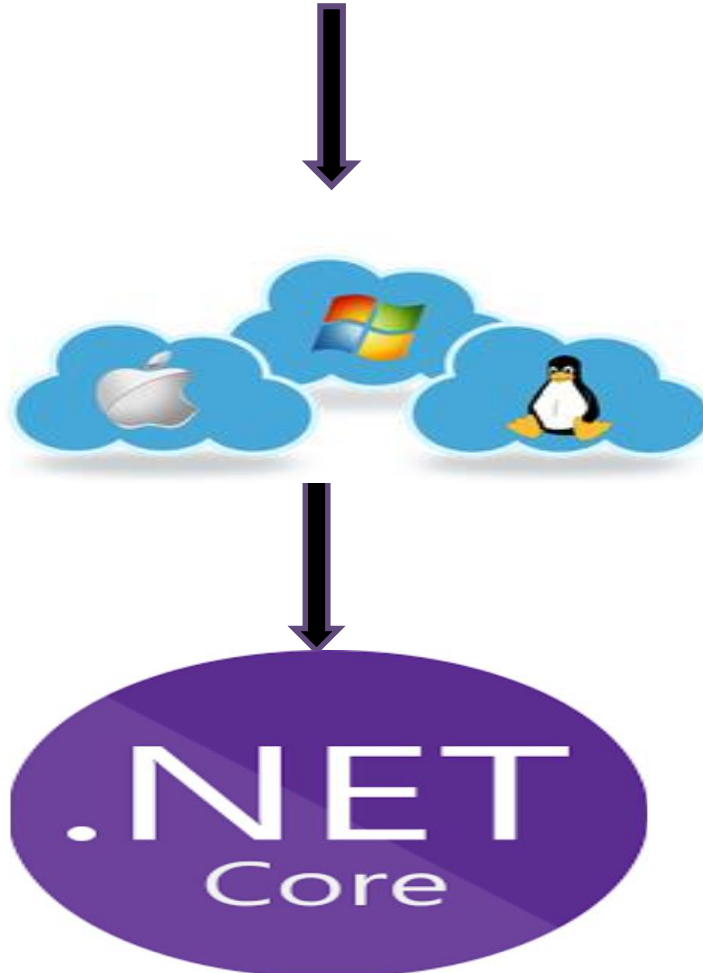
☐ ASP .Net Core

- Développement sous linux
- Entity Framework
- Inversion du contrôle
- Les sessions et les cookies
- JWT



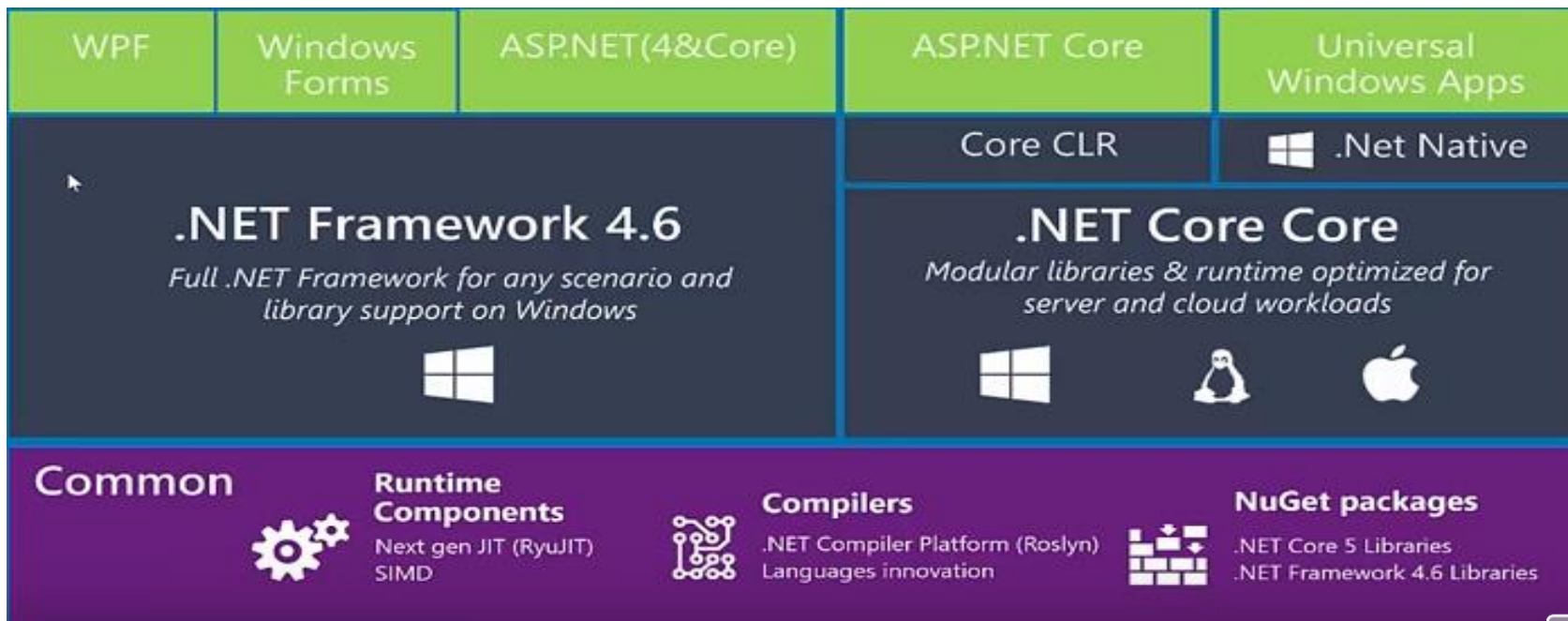
C'est quoi le .Net Core?

-La tendance actuelle de Microsoft: ouverture à un nouveau type de client.



C'est quoi le .Net Core?

-C'est une nouvelle implémentation open source de .NET qui va nous permettre d'écrire du code multiplateforme pour les charges de travail optimisé pour le cloud.



Caractéristiques de .Net Core

- **Multiplateforme:** càd cross plateforme
- **Open source:** Le Framework .Net Core est open source et disponible sur GitHub.



Caractéristiques de .Net Core

- **Modulaire:** .NET Core est modulaire dans son architecture car on ajoute au **project.json** uniquement les dépendances dont on a besoin. Par défaut, il n'y en a pas, et Visual Studio et Nuget se chargent de télécharger les fichiers correspondant aux dépendances supplémentaires.
- Dans le Framework .Net classique, il est nécessaire d'installer l'ensemble du Framework. Les applications se basent ensuite sur le GAC pour trouver les librairies à utiliser.
- Dans le .Net Core l'ensemble des librairies utilisées par l'application sont fournies à l'application, c'est-à-dire chaque application vivra avec ses versions de librairies. Il n'y aura plus de dépendances entre les applications, cela via l'outil DNU (DN Update) de **DNX** (Dot Net eXecution)
- Le DNX est un sdk open source ainsi qu'un environnement d'exécution qui permet le développement d'applications .Net pouvant fonctionner sur windows, OSX ou linux.



Caractéristiques de .Net Core

- Côté exécution:

- .Net Core est plus rapide dans l'exécution
- .Net Core est folder base et non project base càd si on supprime/ajoute quelque chose dans le dossier sera directement supprimé/ajouté dans le projet en visual studio



Caractéristiques de .Net Core

- Côté Déploiement: (projet ASP.net core)

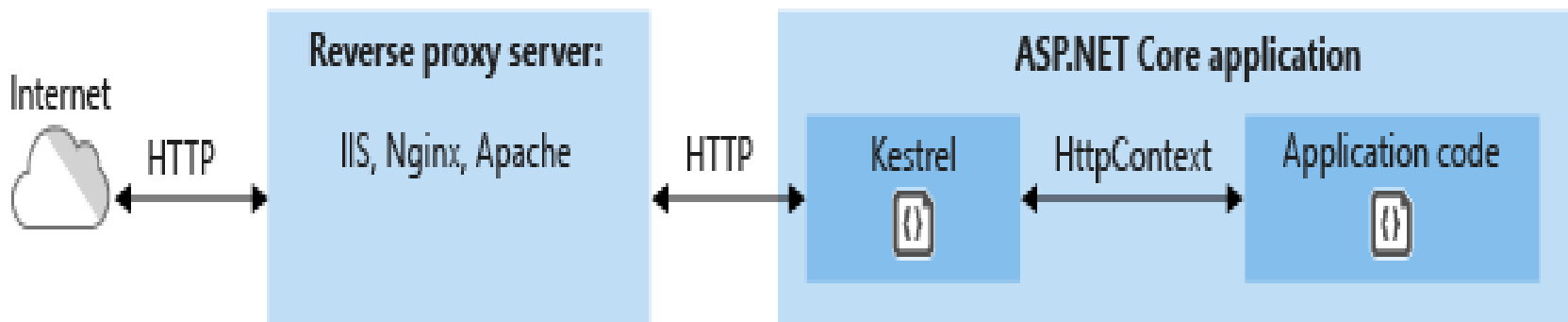
-On reste pas limiter à IIS puisque la configuration de projet se fait dans le projet.json et le fichier startup càd pas de web.config qui est spécifique à IIS.

-Pour l'exécution microsoft fournit un serveur, **Kestrel**. Ce serveur est également développé avec .NET Core, et lancé avec DNX. c'est un serveur multiplateforme.

Caractéristiques de .Net Core

- Côté Déploiement: (projet ASP.net core)

- Kestrel est pris en charge sur toutes les plates-formes
- On peut utiliser Kestrel seul ou avec un *serveur proxy inverse* , tel que IIS, Apache ou Nginx. Un serveur proxy inverse reçoit des requêtes HTTP provenant d'Internet et les transmet à Kestrel après un traitement préliminaire.



Caractéristiques de .Net Core

- Côté Déploiement: (projet ASP.net core)

-Kestrel avec IIS: Lorsque on utilise IIS ou IIS Express en tant que proxy inverse pour ASP.NET Core, l'application ASP.NET Core s'exécute dans un processus distinct du processus de travail IIS. Dans le processus IIS, un module IIS spécial s'exécute pour coordonner la relation de proxy inverse. C'est le *module de base ASP.NET*. Les principales fonctions du module principal ASP.NET sont de démarrer l'application ASP.NET Core, de la redémarrer lorsqu'elle se bloque et de lui transférer le trafic HTTP au serveur kestrel

-Kestrel avec Nginx:

Caractéristiques de .Net Core

- Côté Déploiement: (projet ASP.net core)

- Possibilité de déployer sous **Doker** , afin de nettement faire baisser les coûts des licences liés aux applications .NET.

- Docker est une application open source permettant d'automatiser le déploiement d'applications dans des conteneurs logiciels.

- Docker s'intègre dans les outils de la mise en place de **Devops**

DevOps

- C'est une philosophie de développement qui vise à créer une collaboration entre l'équipe de développement et les opérationnelles afin de développer;
- Outils qui aident à arriver à ce but
- Les outils de développement continue comme **Git/SVN**
- Les outils d'automatisation des tests unitaires et les tests d'intégration comme **Jenkins**
- Outils d'automatisation de déploiement comme :**Docker**

Docker

Avant docker:

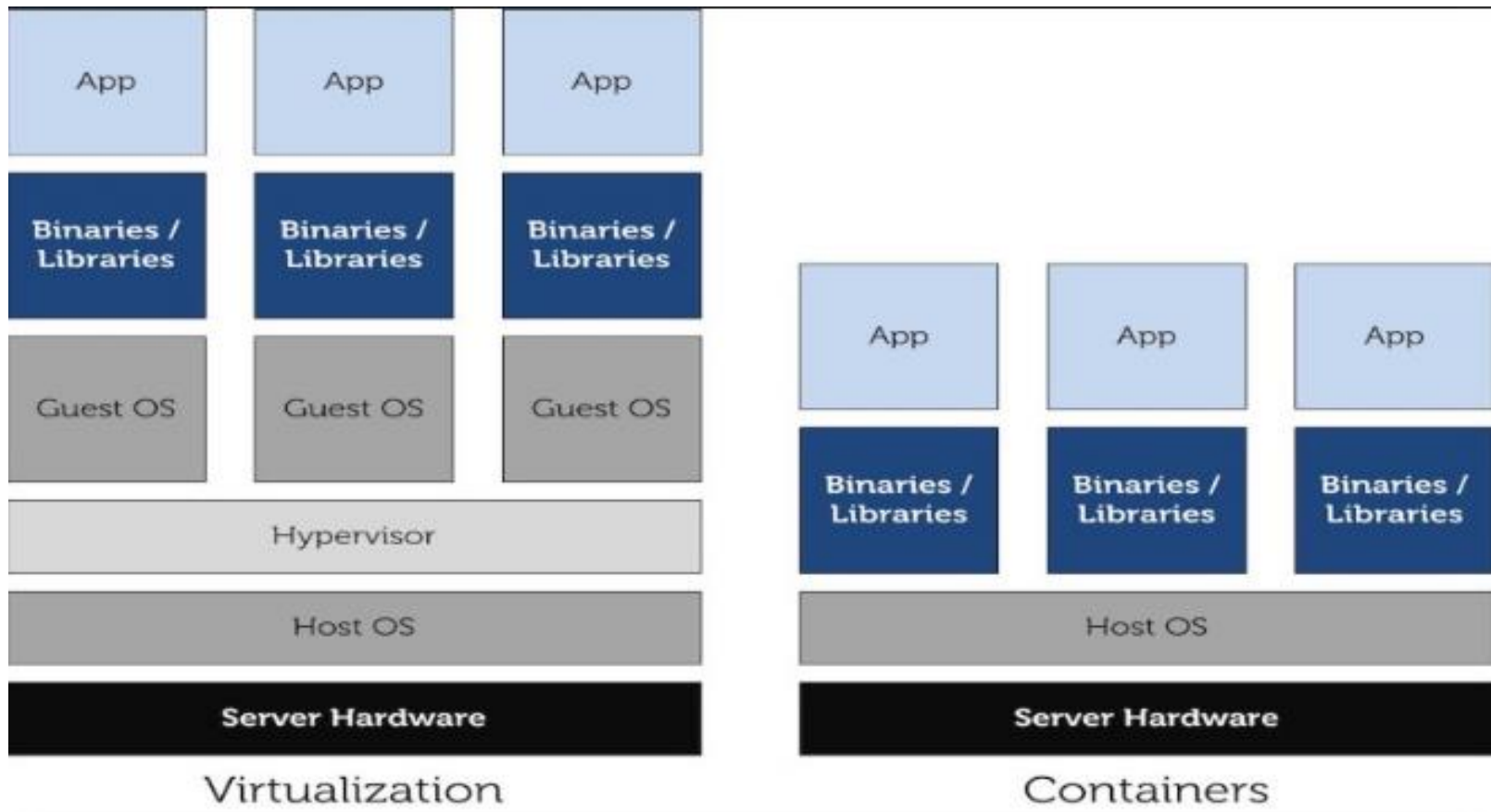
1. le développeur : développe l'application puis génère le package de l'application à déployer: l'ensemble de fichiers cs .Net ou le fichier war en java.
2. Il doit déterminer l'ensemble des dépendances et des configurations à mettre en place pour que l'application fonctionne correctement
3. L'opérationnel : doit installer et configurer tout ce qui est demandé par le développeur
4. Si le développeur effectue des mises à jour, l'opérationnel doit refaire l'étape précédente--> la vie sera difficile pour l'administrateur système--> Avoir des conflits avec le développeur + beaucoup de temps pour le déploiement

Docker

Avec docker:

1. Le développeur développe l'application et construit une image docker contenant toutes les dépendances nécessaires pour son exécution.
2. L'opérationnel instancie le conteneur et fait un « run » de l'image précédente

Docker



C'est quoi docker

- C'est un conteneur permettant de créer un environnement d'isolation d'une application
- Un conteneur effectue un regroupement de tous les fichiers nécessaires à l'exécution d'un programme(code, runtime, outils de SE, bibliothèque..)

Docker

le développeur va développer l'application et la mettre dans un conteneur et par la suite avec des simples commandes on va arriver à déployer et démarrer l'application → on aura pas des conflits entre les versions et les différences dépendances utilisées pour chaque service de l'application

Mise en place de docker

Pour commencer à utiliser docker il faut avoir

- le moteur docker:Exécute des commandes nécessaires pour créer et exécuter les images docker
- docker client:pour commencer à interroger avec le moteur docker

Remarque: On a aussi Docker hub c'est une plateforme dans laquelle on peut publier les conteneurs

Les principales commandes de docker

- `docker build`: pour créer une image
- `docker push image`: publier une image sur docker hub
- `docker pull image`: pour télécharger une image
- `docker run image`: exécuter une image

Implémentation de docker

1. Le développeur crée l'application
2. Le développeur crée un dockerfile: un fichier qui contient quelques lignes décrivant les dépendances que l'application aura besoin
3. Puis il exécute la commande docker build qui permet de créer une image de projet
4. Une fois l'image créée il peut la publier sur Docker hub via la commande docker push

Implémentation de docker

5. L'administrateur va appeler la commande 'Docker run':pour exécuter une image sur une machine càd démarrer l'application sur un conteneur→l'administrateur système n'a pas besoin de plusieurs éléments pour qu'il puisse déployer l'application il va rien installer chez lui

Docker en .Net

Nouvelle application web ASP.NET Core - WebApplication7

.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 2.1.402

Authentification : **Aucune authentification**

[Modifier l'authentification](#)

☒ Activer le support de Docker

SE : Windows

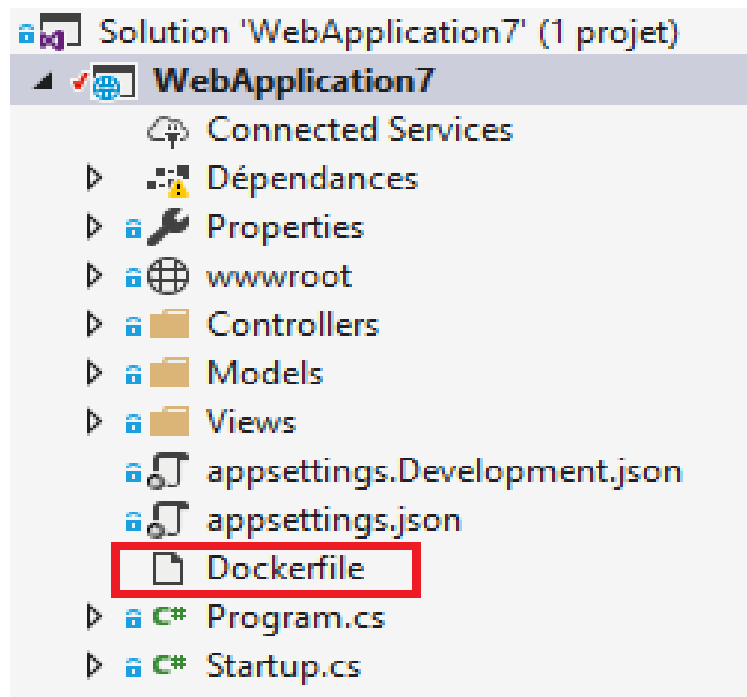
Nécessite [Docker pour Windows](#)

Vous pouvez aussi activer le support Docker plus tard [En savoir plus](#)

☒ Configurer pour HTTPS

OK Annuler

Docker en .Net



ASP.Net Core

Environnement du developpement .Net Core

- Il faut avoir DNX qui représente l'environnement d'exécution
- Il faut avoir Kestrel pour faire fonctionner une application
- Coupler l'application avec un serveur proxy

Windows  IIS

Linux  Nginx

Environnement du développement. Net Core

- **Sous Windows**

Etape1: Installation de visual studio community 2017

Etape2: Installation .Net core (cela s'installe implicitement avec VS2017)

Etape3: Avoir power shell3.0 pour qu'on puisse installer les dépendances

Environnement du développement. Net Core

- **Sous linux**

Etape1: Installation des dépendances .NET Core

Cela dépend de la version de linux voir le lien

<https://www.microsoft.com/net/learn/get-started/linux/rhel>

Etape2: Installation de Visual studio code

<https://www.microsoft.com/net/learn/get-started/linux/rhel#editor>

Remarque: Microsoft a produit une version sql server pour linux

<https://www.windows8facile.fr/installer-sql-server-linux-ubuntu/>

Environnement du développement. Net Core

- **Sous linux**

-Installation de NPM: package manager qui facilite l'installation des packages à partir d'internet

sudo apt-get install npm

//Mettre à jour la version de NPM

sudo npm install -g npm

//InstallerNode.js

sudo apt-get install nodejs-legacy

Environnement du développement. Net Core

- **Sous linux**

-Installation Yeoman: permet d'avoir des templates des projets de .Net

```
sudo npm install -g yo
```

-Installer ASP.Net Generator

```
sudo npm install -g generator-aspnet
```

-Installation de Bower: package manager pour le web

```
sudo npm install -g bower
```

Environnement du développement. Net Core

- **Sous linux**

Remarques:

-Certains modèles de Yeoman utilisent des versions "anciennes" .Net Core. Exécuter cette commande pour corriger ces dépendances en installant toutes les versions .Net Core Framework.

`sudo apt-get install dotnet-dev*`

Environnement du développement. Net Core

- **Sous linux**

- Création du projet

```
mkdir Projet1
```

```
cd Projet1
```

```
yo aspnet
```

Remarque: si vous recevez des erreurs concernant le droit d'accès exécuter la commande suivante

```
sudo chmod -R 777 ~/.config/".
```

Environnement du développement. Net Core

- **Sous linux**



Environnement du développement. Net Core

- **Sous linux**

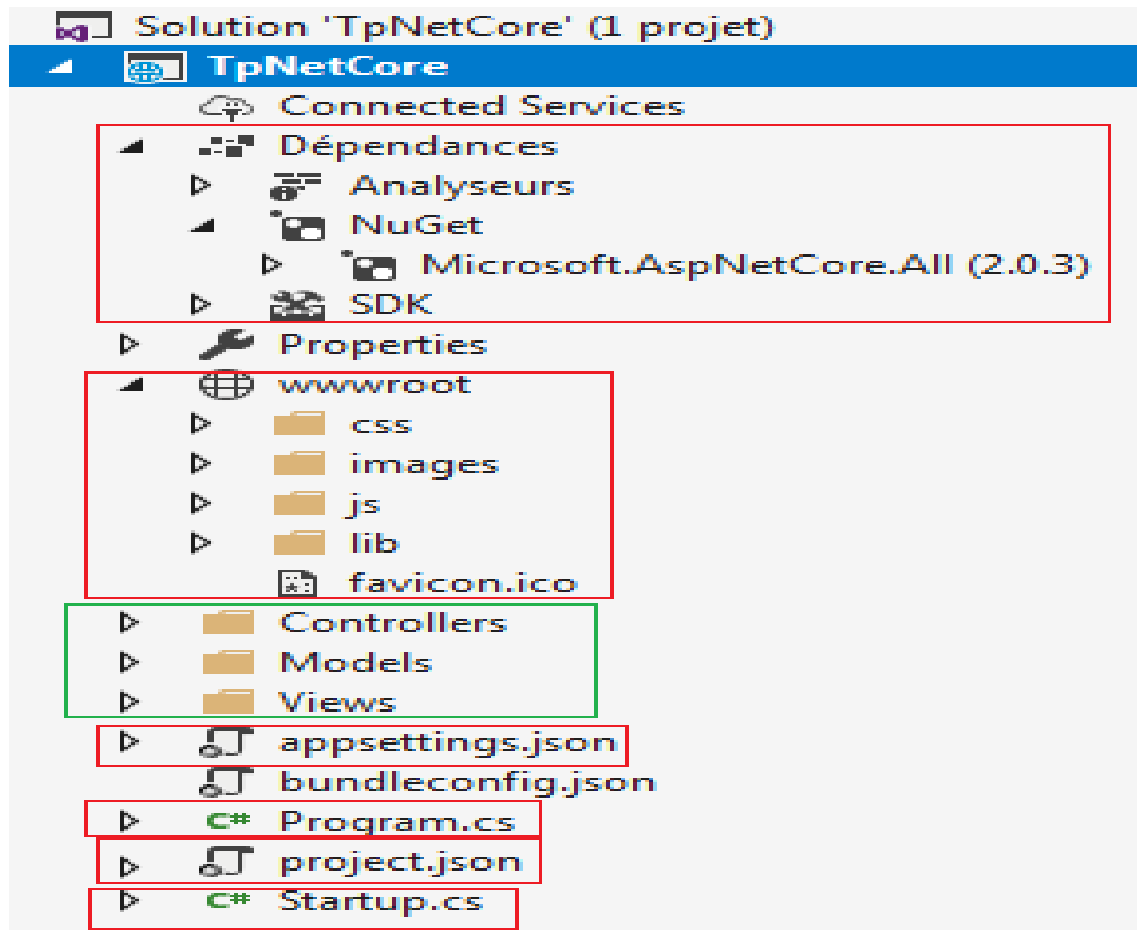
Pour exécuter l'application, exécuter les deux commandes suivantes :

dotnet restore

dotnet run

Architecture d'une application ASP.Net Core

Ajouter nouveau projet → ASP.Net Core



Architecture d'une application ASP.Net Core

- **Dépendances:** contient toutes les librairies utilisées dans le projet. Ces librairies vont être déclarées dans le fichier `project.json`.
- **project.json:** un fichier qui déclare les dépendances du projet. lorsque nous ajouterons de nouvelles dépendances via NuGet, elles seront ajoutées ici.
- **Appsetting.json:** fichier de configuration de l'application : paramètres de chaîne de connexion
- **wwwroot:** contient les fichiers statiques de l'application: `html,css,images,javascripts...`
- **Startup.cs:** représente la classe du démarrage. Il dispose de deux méthodes qui sont invoquées à chaque démarrage du projet Web

Architecture d'une application ASP.Net Core

- **Program.cs**: contient la méthode Main c'est lui qui démarre le serveur kestrel

Remarque: En VS2017 le projet.json est remplacé par le project.xml. Pour l'afficher click droit sur le projet → Modifier NotreApplication.csproj.

-Il y'a la possibilité de créer un projet ASP.NET core(.Net Framework) càd un projet qui prend les privilèges de .Net core(structure du projet/multiplateforme) mais il prend toutes les dépendances de .Net framework

Développement en ASP.Net core

- Tous ce qui est vu en ASP MVC est valable en ASP.Net core
- Les points de différence vont se traiter dans les prochaines diapositives.

Entity framewok.net core

Pour utiliser entity framework dans les projets.Net Core, on doit l'installer dans le projet concerné via Nuget.

Install-Package Microsoft.AspNetCore.All -Version 2.0.3

Install-Package Microsoft.EntityFrameworkCore.SqlServer -Version 2.0.1

Install-Package Microsoft.EntityFrameworkCore.Tools -Version 2.0.1

Install-Package Microsoft.EntityFrameworkCore.SqlServer.Design -Version 1.1.5

Install-Package Microsoft.VisualStudio.Web.CodeGeneration.Design -Version2.0.1



Entity framewok.net core

- **Code First:**

- **Etape 1:** Model → ajouter une classe (dans laquelle on va définir la structure de notre table)

```
public class User
{
    public int Id { get; set; }

    [StringLength(50)]
    public string Nom { get; set; }

    [Required]
    [StringLength(50)]
    public string Prenom { get; set; }

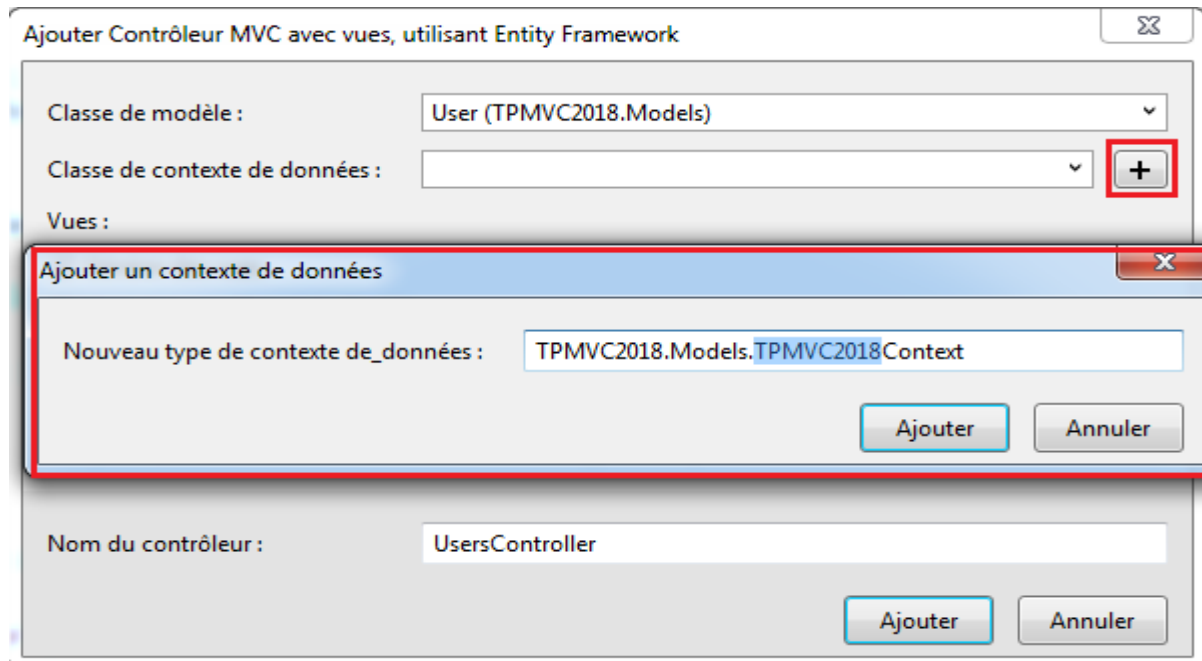
    [Required]
    [DataType(DataType.EmailAddress)]
    [StringLength(50)]
    public string Email { get; set; }
}
```



Entity framewok.net core

- **Code First:**

- **Etape2:** Contrôler → ajouter « Contrôleur MVC avec vues, utilisant Entity Framework »



- Le dataContext va être générer dans un Dossier appelé « Data »

- La chaine de connexion s'ajoute dans le fichier « appsettings.json »

Entity framework.net core

- Code First:**

- Etape3:** génération de la base de donnée

Add-Migration Initial -Context TPMVC2018.Models.TPMVC2018Context

→La commande Add-Migration va créer dans notre projet un dossier “Migrations”, avec du code représentant le schéma de notre base de données.

Update-Database Initial -Context TPMVC2018.Models.TPMVC2018Context

→La commande Update-Database va utiliser le code précédent pour mettre à jour la base de données.

Entity framewok.net core

- **Code First:**

Remarque: Au cas des erreurs lors de création de la BD:

- Vérifier qu'il n'y'a pas des problèmes au niveau de chaine de connection dans appsettings.json

Entity framewok.net core

- **DataBase First:**

Etape1: Création le modèle EF basé sur notre base de données

Pour cela exécuter la commande suivante dans la console Nuget

```
Scaffold-DbContext "Data source=user-pc\sqlexpress;Initial  
Catalog=tp1ADO;Integrated Security=true"  
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

Entity framewok.net core

- **DataBase First:**

Etape2: Enregistrez votre contexte avec l'injection de dépendance

- *Supprimer la configuration de contexte en ligne:*

Dans ASP.NET Core, la configuration est généralement effectuée dans Startup.cs. Pour se conformer à ce modèle, nous allons déplacer la configuration du fournisseur de base de données vers Startup.cs.

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.UseSqlServer(@"Server=user-PC\SQLEXPRESS;Database=tp1ADO;Trusted_Connection=True");
    }
}
```

Entity framewok.net core

- **DataBase First:**

Etape2: Enregistrez votre contexte avec l'injection de dépendance

▪ *Ajoutez le constructeur suivant, qui permettra à la configuration d'être passée dans le contexte par injection de dépendance*

```
public tp1ADOContext(DbContextOptions<tp1ADOContext> options) : base(options)
{ }
```



Entity framewok.net core

- **DataBase First:**

Etape2: Enregistrez votre contexte avec l'injection de dépendance

- *Enregistrez et configurez le contexte dans Startup.cs*

Pour que nos contrôleurs MVC puissent utiliser le context de données « tp1ADOContext », nous allons l'enregistrer en tant que service.

```
public void ConfigureServices(IServiceCollection services)
{
    var connection = @"Server=user-PC\SQLEXPRESS;Database=tp1ADO;Trusted_Connection=True;";
    services.AddDbContext<tp1ADOContext>(options => options.UseSqlServer(connection));

    services.AddMvc();
}
```



Inversion du contrôle

Microsoft Dependency Injection

•Installation:

Install-Package NLog.Extensions.Logging -Version 1.0.0-rtm-rc7

Install-Package Microsoft.Extensions.DependencyInjection -Version 2.0.0

```
<ItemGroup>  
  <PackageReference Include="Microsoft.Extensions.DependencyInjection" Version="2.0.0" />  
  <PackageReference Include="NLog.Extensions.Logging" Version="1.0.0-rtm-rc7" />  
</ItemGroup>
```

Microsoft Dependency Injection

- Configuration des métadonnées (Définition de dépendance)

```
var services = new ServiceCollection()  
    .AddSingleton<IDAO, DAOImpl>()  
    .AddSingleton<IMetier, MetierImp>();
```

-ServiceCollection est une collection qui va être responsable d'établir la correspondance entre l'interface et son implémentation(service à utiliser)

-C'est comme ça qu'on ajoute la configuration dans le conteneur

Microsoft Dependency Injection

•Instanciation des objets:

```
// crée un fournisseur de services à partir de la collection de services
var serviceProvider = services.BuildServiceProvider();

// résoudre le graphe de dépendance
var appService = serviceProvider.GetService<IMetier>();

Console.WriteLine("Valeur= " + appService.TraitementDonne());
```

Microsoft Dependency Injection

- **Mode d'instanciation:**

- **AddSingleton:**

Il crée une instance unique dans toute l'application. Il crée l'instance pour la première fois et réutilise le même objet dans tous les appels.

```
services.AddSingleton< IFilieresService , FilieresServiceImpl >();
```

Microsoft Dependency Injection

- **Mode d'instanciation:**

- AddTransient:**

Les objets auront une durée de vie transitoire, ce mode crée une nouvelle instance à chaque fois ces objets seront demandés

```
services.AddTransient< IFilireService , FilireServiceImpl >();
```

Microsoft Dependency Injection

- **Mode d'instanciation:**

- **AddScope:**

il crée une instance pour chaque requête http mais on utilise la même instance dans les autres appels de la même requête Web.

```
services.AddScoped< IFiliereService , FiliereServiceImpl >();
```

Microsoft Dependency Injection

Exemple: On crée une application ASPMVC.Net Core

1.Ajout des services

```
public interface IService1
{
    String GetValeurInstance();
}
```

```
public class Service1Imp:IService1
{
    public String GetValeurInstance()
    {
        return "Bonjour " + GetHashCode();
    }
}
```


Microsoft Dependency Injection

2. Configuration des services

La configuration de Microsoft Dependency Injection en ASP MVC se fait dans le fichier **Startup.cs**

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();

    services.AddSingleton<IService1, Service1Imp>();
}
```

Microsoft Dependency Injection

3.Utilisation des services :Controller

```
public class Service1Controller : Controller
{
    private IService1 s1;
    private IService1 s2;

    public Service1Controller(IService1 s1, IService1 s2)
    {
        this.s1 = s1;
        this.s2 = s2;
    }

    public IActionResult Index()
    {
        ViewData["message"] = s1.GetValeurInstance() + " " + s2.GetValeurInstance();
        return View();
    }
}
```

Microsoft Dependency Injection

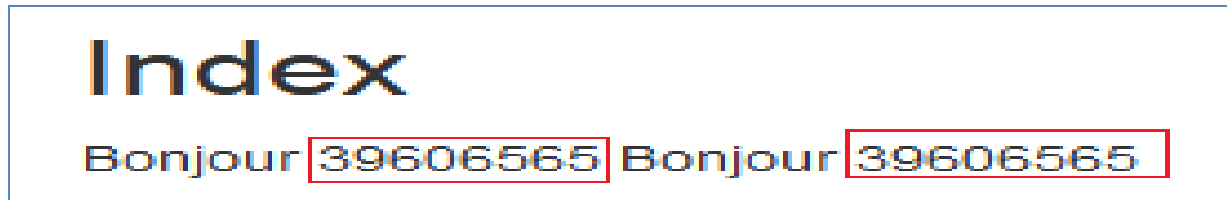
3.Utilisation des services :View

```
<h2>Index</h2>  
  
@ViewData["message"]
```

Microsoft Dependency Injection

3. Résultat d'exécution:

-Mode singleton:

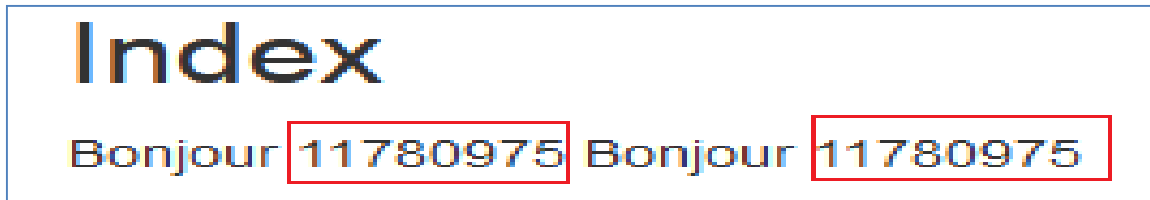


- On a le même résultat pour les deux appels
- Aussi si on fait un refresh/ou on passe à un autre navigateur ,toujours on a le même résultat

Microsoft Dependency Injection

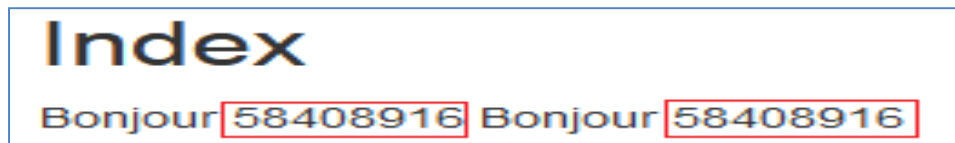
3. Résultat d'exécution:

-Mode scoped:



-On a le même résultat pour les deux appels

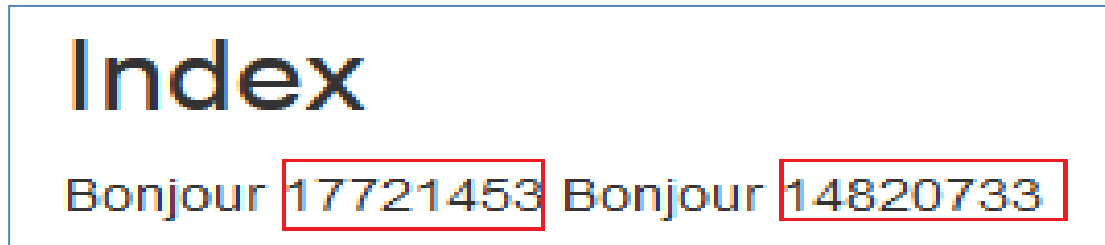
-Mais si on fait un refresh/ou on passe à un autre navigateur on aura un autre résultat



Microsoft Dependency Injection

3. Résultat d'exécution:

-Mode Transient:



Index
Bonjour 17721453 Bonjour 14820733

-On a une valeur pour chaque appel

Microsoft Dependency Injection

Remarque:

-ASP MVC.Net core intègre un support pour l'injection de dépendances.

Microsoft Dependency Injection

- FormService:

```
public class Service1Controller : Controller
{
    private IService1 s1;
    private IService1 s2;

    public Service1Controller(IService1 s1, IService1 s2)
    {
        this.s1 = s1;
        this.s2 = s2;
    }

    public IActionResult Index()
    {
        ViewData["message"] = s1.GetValeurInstance() + " " + s2.GetValeurInstance();
        return View();
    }
}
```

-Ici on doit passer par un constructeur ,sinon s1 et s2 reste null

Microsoft Dependency Injection

- **FromService:**

C'est une annotation utilisé pour injecter l'objet donc ici ce n'est pas la peine d'ajouter un constructeur.

```
public class Service1Controller : Controller
{
    private IService1 s1;
    private IService1 s2;

    public IActionResult Index([FromServices]IService1 s1, [FromServices]IService1 s2)
    {
        ViewData["message"] = s1.GetValeurInstance() + " " + s2.GetValeurInstance();
        return View();
    }
}
```