Chapitre 2:

Framework de développement Web **ASP.Net Core MVC**

Hajer TAKTAK
Maître Assistante - INSAT
Ingénieur en Génie Logiciel - INSAT
Docteur à LIPAH-LR11ES14
Faculté des Sciences de Tunis
Département des Sciences de l'Informatique
Université Tunis el Manar

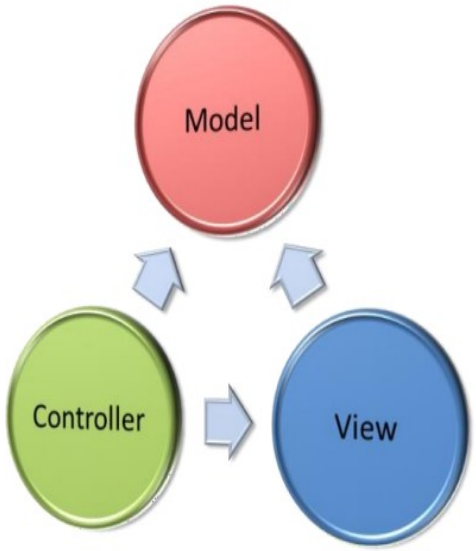
Plan du cours

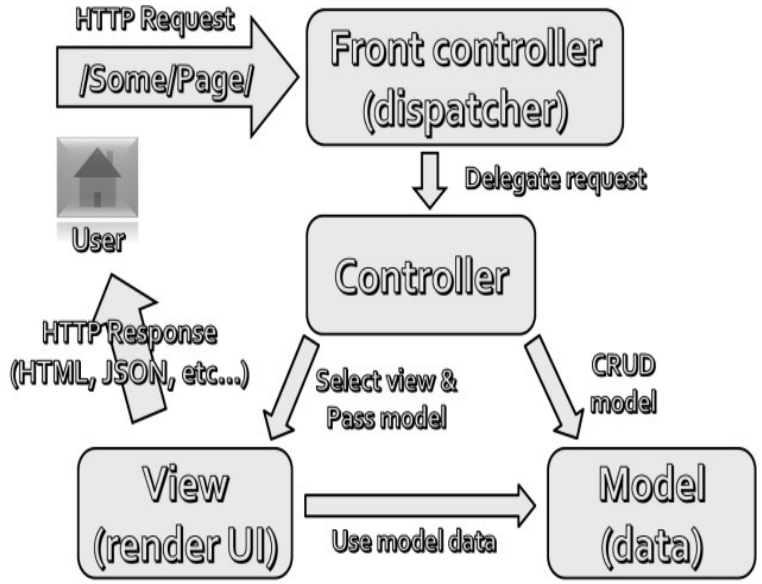
1. Le Patron MVC
2. De ASP vers ASP.net Core
3. Définitions
4. Avantages
5. ASP.NEt Core MVC : Architecture, Arborescence et Cycle de vie des Services

Le patron MVC
MODÈLE- VUE - CONTRÔLEUR

- Modèle MVC: modèle de conception imposant la séparation des préoccupations entre les 3 couches modèle-vue et contrôleur.

- Éviter de mélanger la logique de présentation, la logique métier et la logique d'accès aux données.





MVC Pattern

Model-View-Controller

Le patron MVC
La couche Modèle

- Encapsulation des données sous forme de classes.
- Règles sur la façon dont les données peuvent être modifiées et manipulées.
- Peut contenir des règles de validation des données.
- Très probablement une couche d'accès aux données.
- Classes : Gère le comportement et les données
- Fournit au contrôleur des listes d'objets

Le patron MVC
La couche vue

- Gère l'affichage des données.
- Code HTML CSS.
- Obtenir généralement les données à partir du contrôleur.
- Utiliser le moteur de vue Razor pour incorporer du code .NET dans le balisage HTML (.csHTML)
- Définit comment l'interface utilisateur (UI) de l'application sera affichée.
- Peut prendre en charge les vues principales (mise en page).
- Peut prendre en charge les sous-vues (vues partielles ou contrôles).
- modèle pour générer dynamiquement du HTML

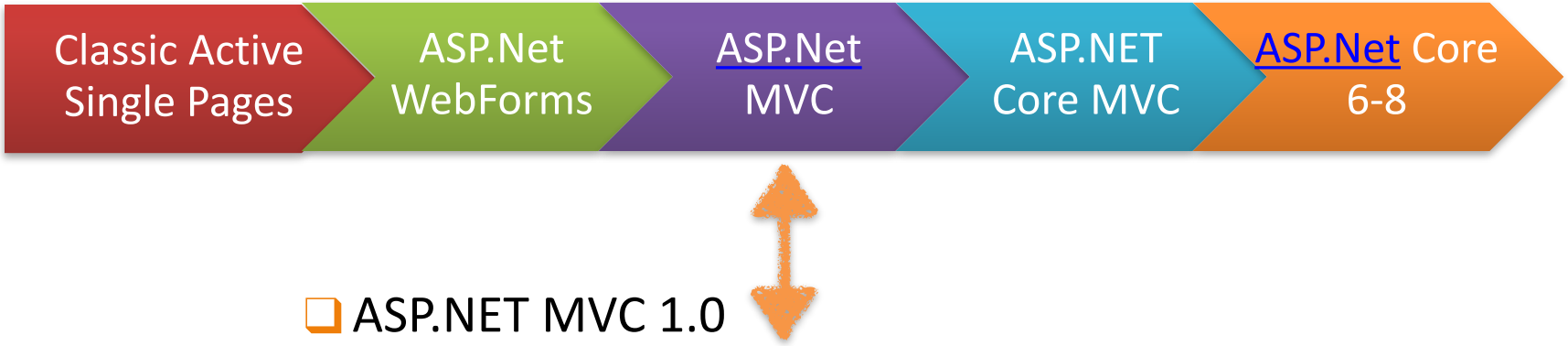
Le patron MVC
La couche contrôleur

- Connecter le modèle, le métier et la page web
- Gérer les événements de page et la navigation entre les pages
- Traiter les requêtes à l'aide de vues et de modèles
- Chaque contrôleur a une ou plusieurs « actions »
- Classes qui gèrent les demandes entrantes de navigateur, récupérer des données de modèle et ensuite spécifier des modèles de vue qui retournent une réponse au navigateur.

Frameworks MVC

- PHP: CakePHP, CodeIgniter, Laravel
- Java: Spring
- Python: Django, Flask, Grok
- Ruby: Ruby on Rails, Camping, Nitro, Sinatra
- JavaScript: AngularJS, JavaScriptMVC
- **ASP.NET Core MVC (.NET 6-7) : Framework de développement Web**

De ASP à ASP.Net Core 6

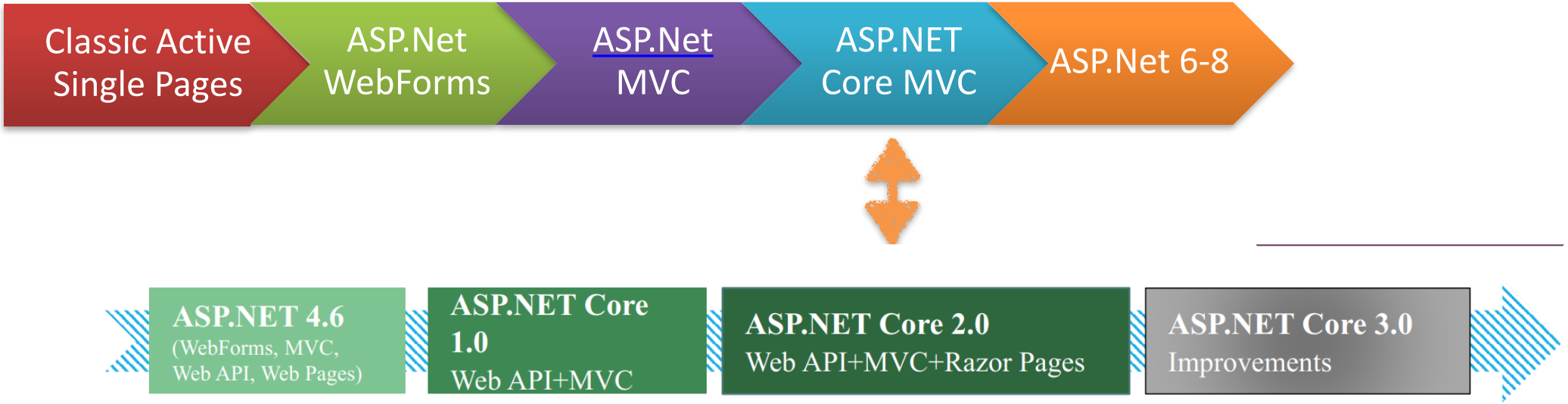


- ASP.NET MVC 1.0

Publié en 13 Mars 2009

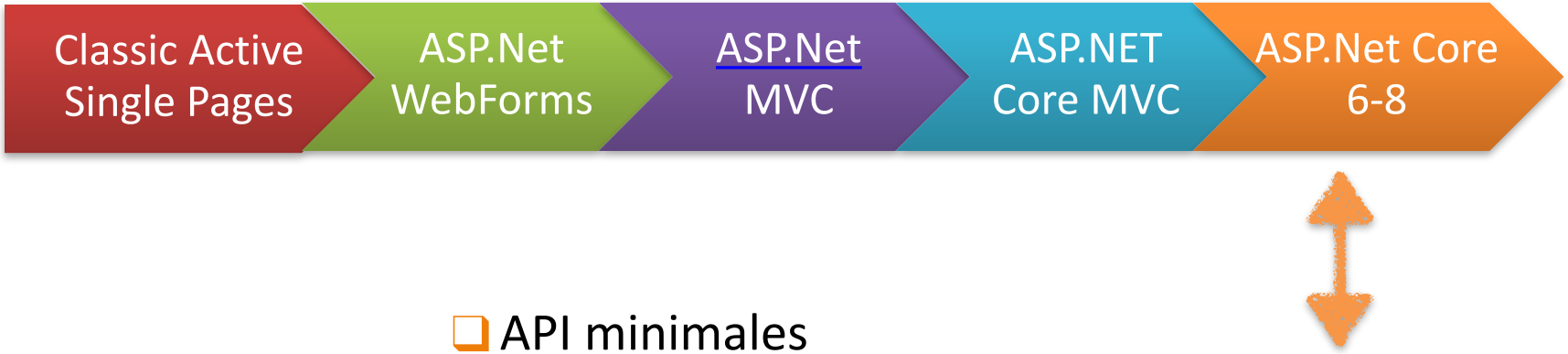
- ASP.NET MVC 2.0 Publié en 10 Mars 2010.
- ASP.NET MVC 3.0 (Razor) – 13 Janvier 2011.
- ASP.NET MVC 4.0 (Web API) – 15 Aout 2012.
- ASP.NET MVC 5.0 (Identity) – 17 Octobre 2013.
- ASP.NET 5 MVC, Web API et Web Pages sont combinés en un seul cadre pour former MVC 6.

De ASP à ASP.Net Core 6



- Besoin multiplateformes pour les applications web
- Architecture micro Services
- Travailler avec des centenaires Docker
- Side By Side Versioning
- Contrôle de l'interface de ligne de commande

De ASP à ASP.Net Core 6



- API minimales
- Signal R
- Compilateur Razor (pas de séparation des views Assembly)
- Blazor…..

Définitions

- **ASP.NET Framework:**
  - ✓ framework permettant de générer à la demande des pages web
  - ✓ utilisé pour mettre en œuvre des applications web
  - ✓ évolution d'Active Server Pages (premier langage et moteur de script côté serveur de Microsoft pour les pages Web dynamiques)
  - ✓ ASP.NET peut être utilisé avec n'importe quel langage de programmation pour la plateforme .NET (Visual Basic .NET, C#, JScript4…).

- **ASP.Net Core 6/7/8**
  - ✓ ASP.NET Core est un framework open source multiplateforme
  - ✓ ASP.NET Core offre les avantages suivants:

Un scénario unifié pour créer une interface utilisateur web et des API web.

Architecture pour la testabilité.

Razor Pages rend le codage des scénarios axés sur la page plus facile et plus productif.

Avantages ASP.Net Core

- Flexibilité: peut être combiné avec les services frontend
  Populaires:
  - React
  - Angular
  - …

ASP.NET Core : EcoSysteme

Cross platform
• *Windows, Linux, Mac, FreeBSD*
Portable
• *Can be ~/bin deployed*
• *Can be user or machine installed as well*
Open source
• *https://github.com/dotnet/coreclr*
• *Contains core runtime and mscorlib (e.g. GC, JIT, BCL)*

EcoSystem

SDK
• *Command-line tooling (dotnet)*
Project system
• *File-system based project system (project.json) Runtime, libraries, and packaging*
• *NuGet-focused*
Editors/IDEs
• *Any text editor (VS Code, Emacs, Sublime, etc) and OmniSharp (OSS)*

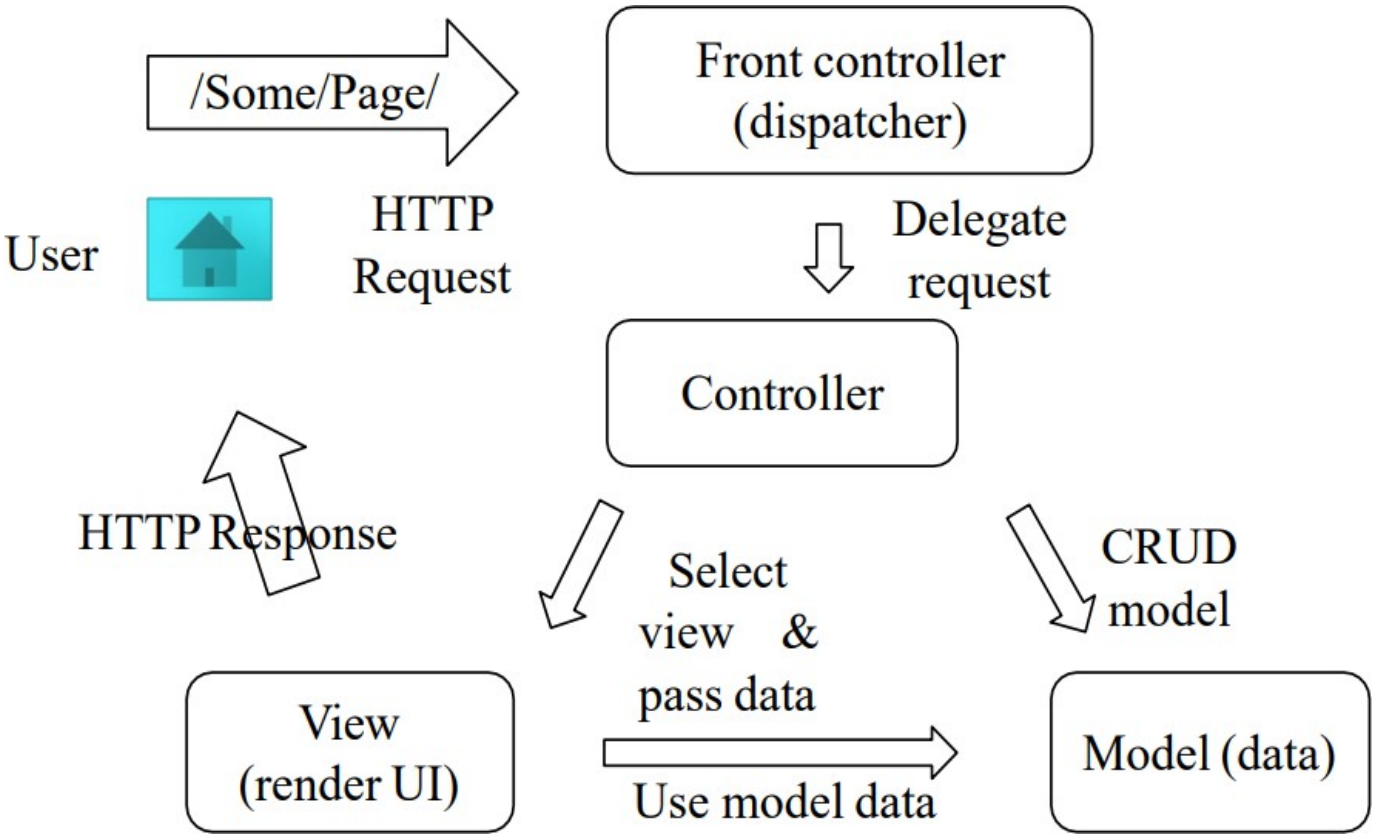
Outils

- Les outils dont on a besoin:

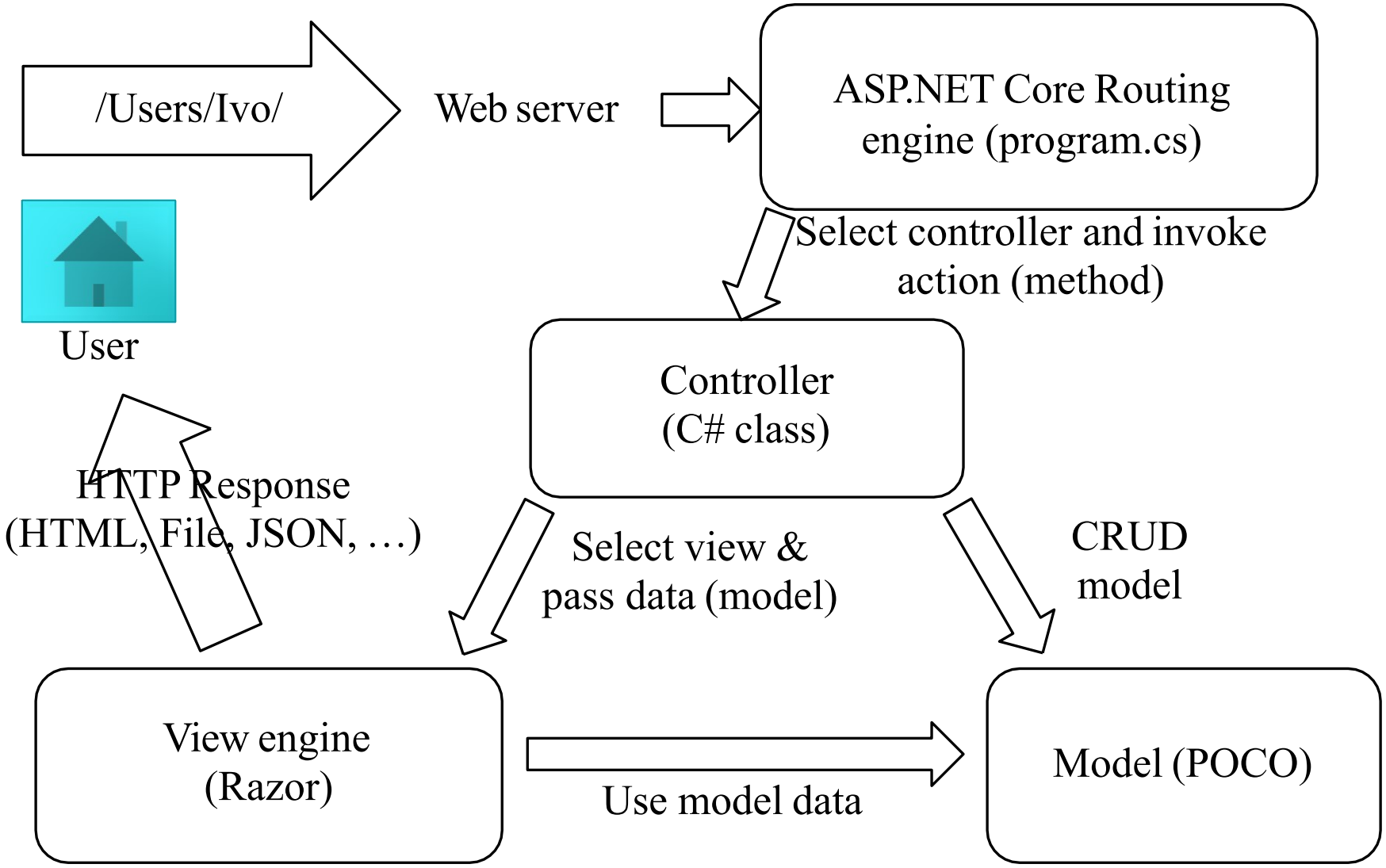
  - ✓ IDE : Visual Studio 2022
  - ✓ Framework: ASP.NET Core MVC
  - ✓ Serveur Web : IIS(Express) ou autre
  - ✓ Données : Microsoft SQL Server (ou autre moteur de BD)

Le patron MVC en ASP.Net Core MVC

- Requête entrante acheminée vers le contrôleur
- Le contrôleur traite la requête et crée le modèle de présentation
- Le contrôleur sélectionne également la vue appropriée
- Le modèle est passé à la vue
- La vue transforme le modèle au format de sortie approprié (HTML)
- La réponse est rendue (réponse HTTP)



Le patron MVC en ASP.Net Core MVC



ASP.NET Core

Middleware

Services

Dependency Injection

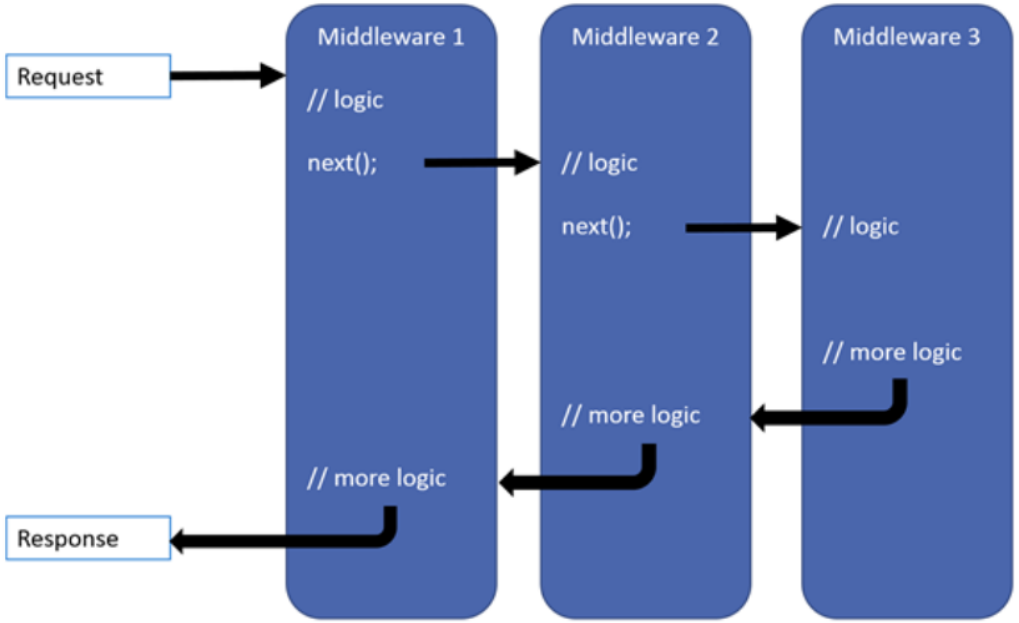
ASP.NET Core

- **Injection de dépendances** intégrée.
- Un **pipeline** des requêtes HTTP léger, à hautes performances et modulaire.
- Possibilité d'héberger sur les éléments suivants :

Kestrel
IIS
HTTP.sys
Nginx
Apache
Docker

- Contrôle de version côte à côte.
- Outils qui simplifient le développement web moderne.

ASP.NET Core : Architecture
Middleware(1/5)



- Un middleware est un logiciel qui est assemblé dans un pipeline d'application pour gérer les requêtes et les réponses.

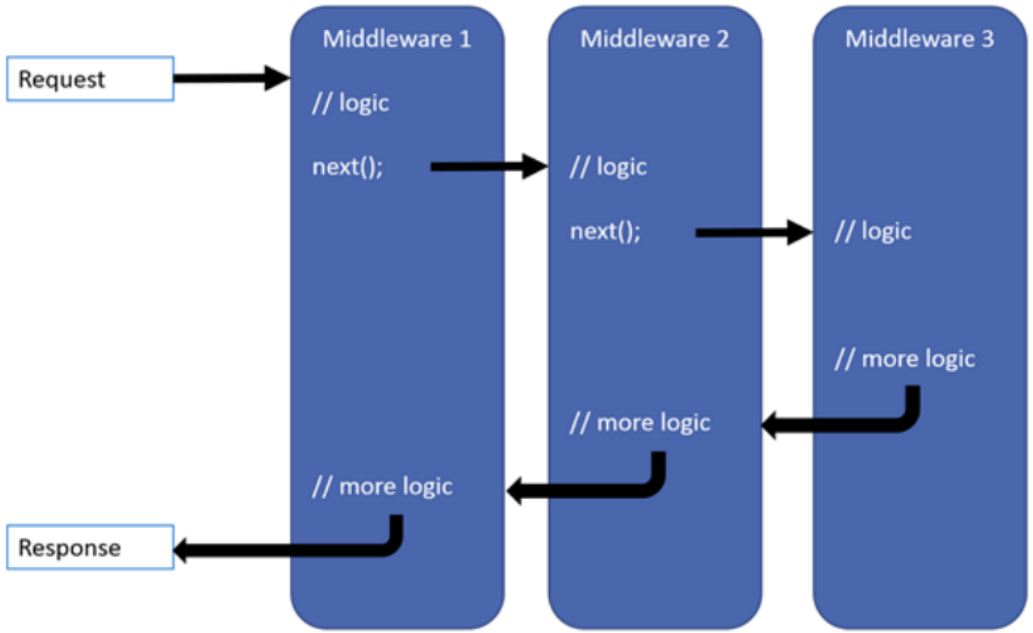
Chaque composant :

- Choisit de passer la requête au composant suivant dans le pipeline.
- Peut travailler avant et après le composant suivant dans le pipeline.

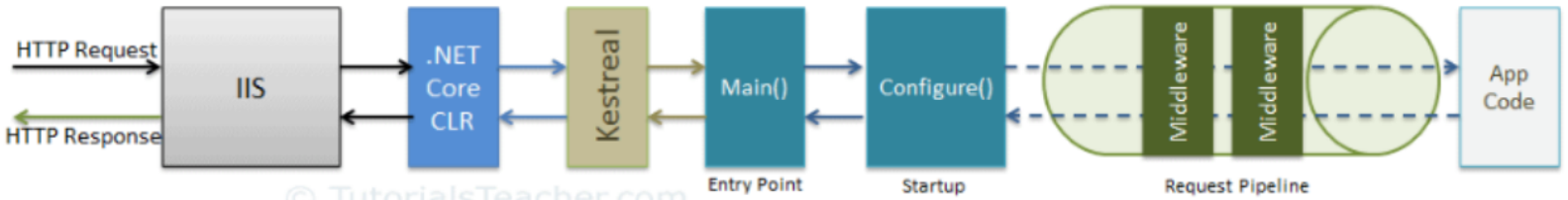
- Les délégués de requête sont utilisés pour créer le pipeline de requête. Les délégués de requête gèrent chaque requête HTTP.

- Les délégués de requête sont configurés à l'aide des méthodes d'extension **Run, Map et Use.**

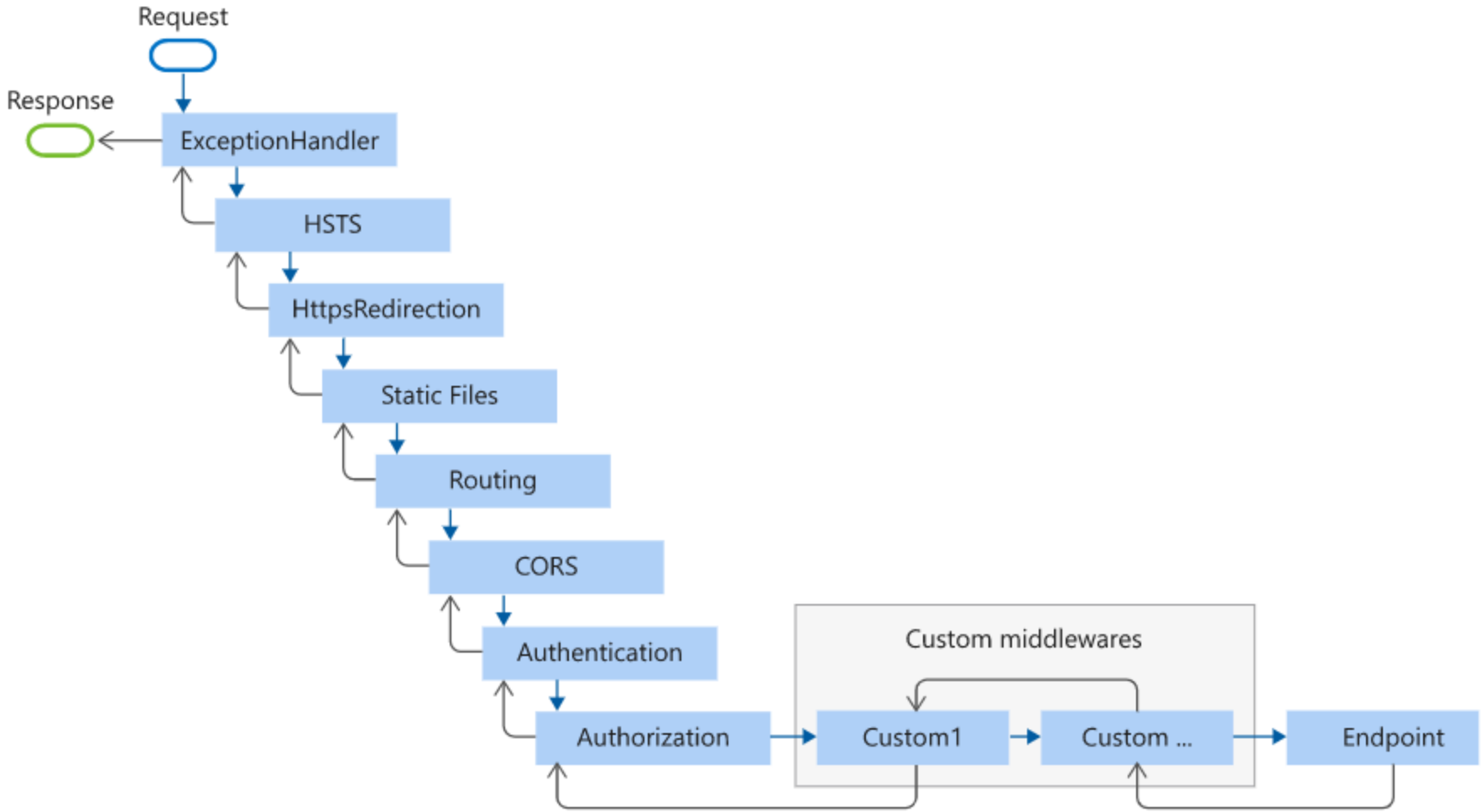
ASP.NET Core : Architecture
Middleware(2/5)




```
public class Startup
{
    public void Configure(IApplicationBuilder app)
    {
        app.Run(async context =>
        {
            await context.Response.WriteAsync("Hello, World!");
        });
    }
}
```



ASP.NET Core : Architecture
Middleware(3/5)



ASP.NET Core : Architecture
Middleware(4/5)

- **Gestion des erreurs/exceptions**

Quand l'application s'exécute dans l'environnement de développement :

* Le middleware Page d'exceptions du développeur (**UseDeveloperExceptionPage**) signale des erreurs de runtime de l'application.

* Le middleware Page d'erreur de base de données (**UseDatabaseErrorPage**) signale des erreurs de runtime de la base de données.

Quand l'application s'exécute dans l'environnement de production :

* Le middleware Gestionnaire d'exceptions (**UseExceptionHandler**) intercepte des exceptions levées dans les middlewares suivants.

ASP.NET Core : Architecture
Middleware(5/5)

- Le middleware Protocole HSTS (HTTP Strict Transport Security) (UseHsts) ajoute l'en-tête Strict-Transport-Security.
- Le middleware Redirection HTTPS (UseHttpsRedirection) redirige les requêtes HTTP vers HTTPS.
- Le middleware Fichier statique (UseStaticFiles) retourne des fichiers statiques et court-circuite tout traitement supplémentaire de la requête.
- Le middleware Cookie Policy (UseCookiePolicy) met l'application en conformité avec les réglementations du RGPD (Règlement général sur la protection des données).
- Middleware Routing (UseRouting) pour router les requêtes.

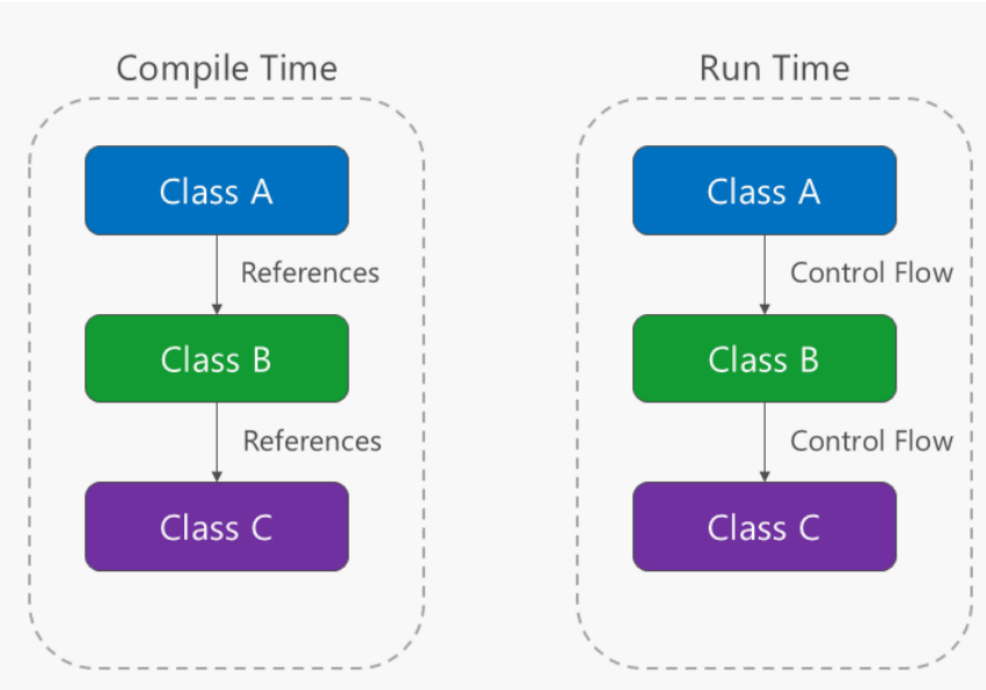
ASP.NET Core : Architecture
Injection de dépendances (1/4)

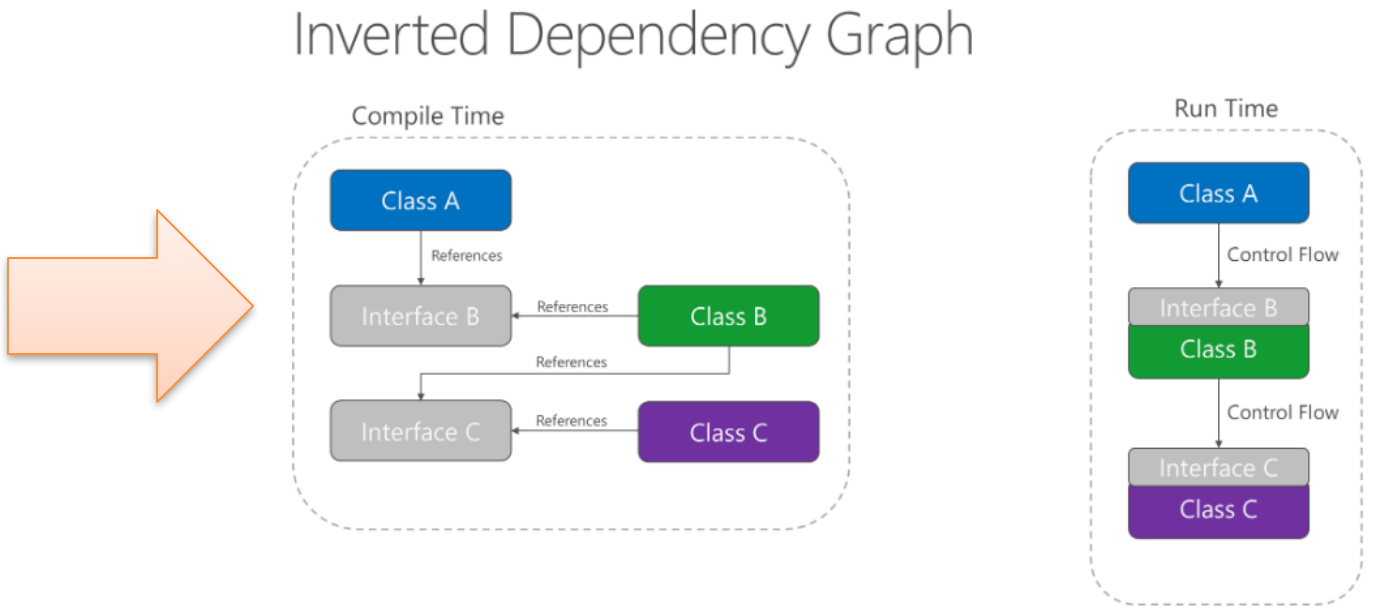
- ASP.NET Core prend en charge le modèle de conception logicielle d'injection de dépendances, technique qui permet d'obtenir une inversion de contrôle entre les classes et leurs dépendances.

- Les contrôleurs ASP.NET Core MVC demandent les dépendances explicitement via des constructeurs.

- ASP.NET Core offre une prise en charge intégrée de l'injection de dépendances.

ASP.NET Core : Architecture
Injection de dépendances (2/4)





ASP.NET Core : Architecture
Injection de dépendances (3/4)

```
public interface IDateTime
{
    DateTime Now { get; }
}
```

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IDateTime, SystemDateTime>();

    services.AddControllersWithViews();
}
```

```
public class SystemDateTime : IDateTime
{
    public DateTime Now
    {
        get { return DateTime.Now; }
    }
}
```

```
public class HomeController : Controller
{ private readonly IDateTime _dateTime;

    public HomeController(IDateTime dateTime)
    {
        _dateTime = dateTime;
    }

    public IActionResult Index()
    {
        var serverTime = _dateTime.Now;}
```

ASP.NET Core : Architecture
Injection de dépendances (4/4)

- Injection par Constructeur : Une des techniques DI les plus populaires. Avec cette approche vous créez une instance de votre dépendance et vous la passez en tant qu'argument au constructeur de la classe dépendante

- Injection par Méthode : Dans ce cas vous créez une instance de la dépendance et vous la passez à une méthode spécifique de la classe dépendante

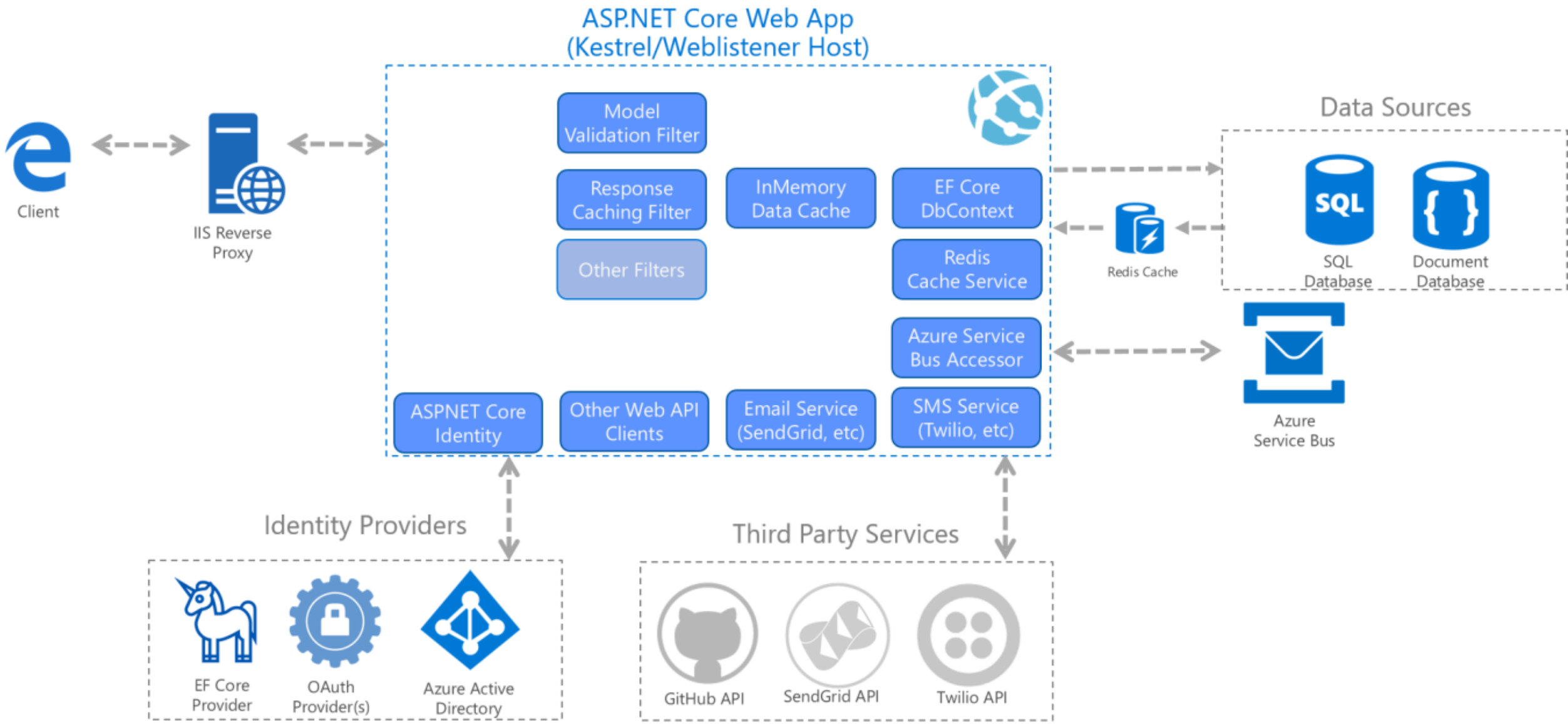
- Injection par Propriété : Cette approche vous permet d'assigner l'instance de votre dépendance à une propriété spécifique de la classe dépendante

ASP.NET Core : Architecture

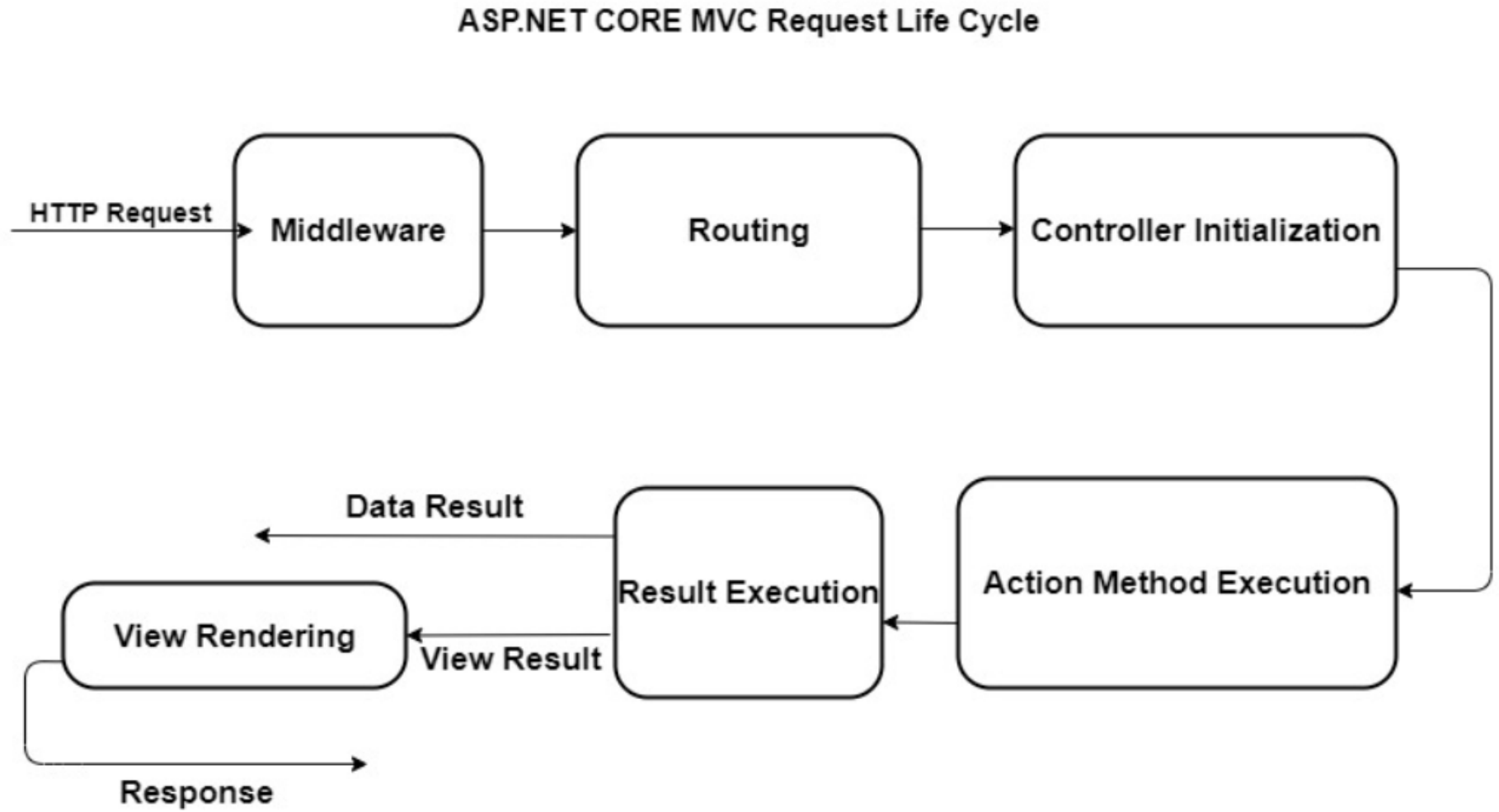
Durée de vie des services dans Centenaire DI

- **Singleton** : Crée une instance du service disponible tant que l'application restera en exécution.
- **Transient** : votre service sera créé à chaque fois qu'il sera demandé. Cela veut dire, par exemple, qu'un service injecté dans le constructeur d'une classe durera "autant de temps" que l'instance de cette classe.
- **Scoped** : créer une nouvelle instance d'un service par chaque demande d'un client. Ceci est particulièrement utile dans le contexte d'ASP.NET car cela vous permet de partager la même instance de service pour la durée de traitement de la demande http.

ASP.NET Core Architecture

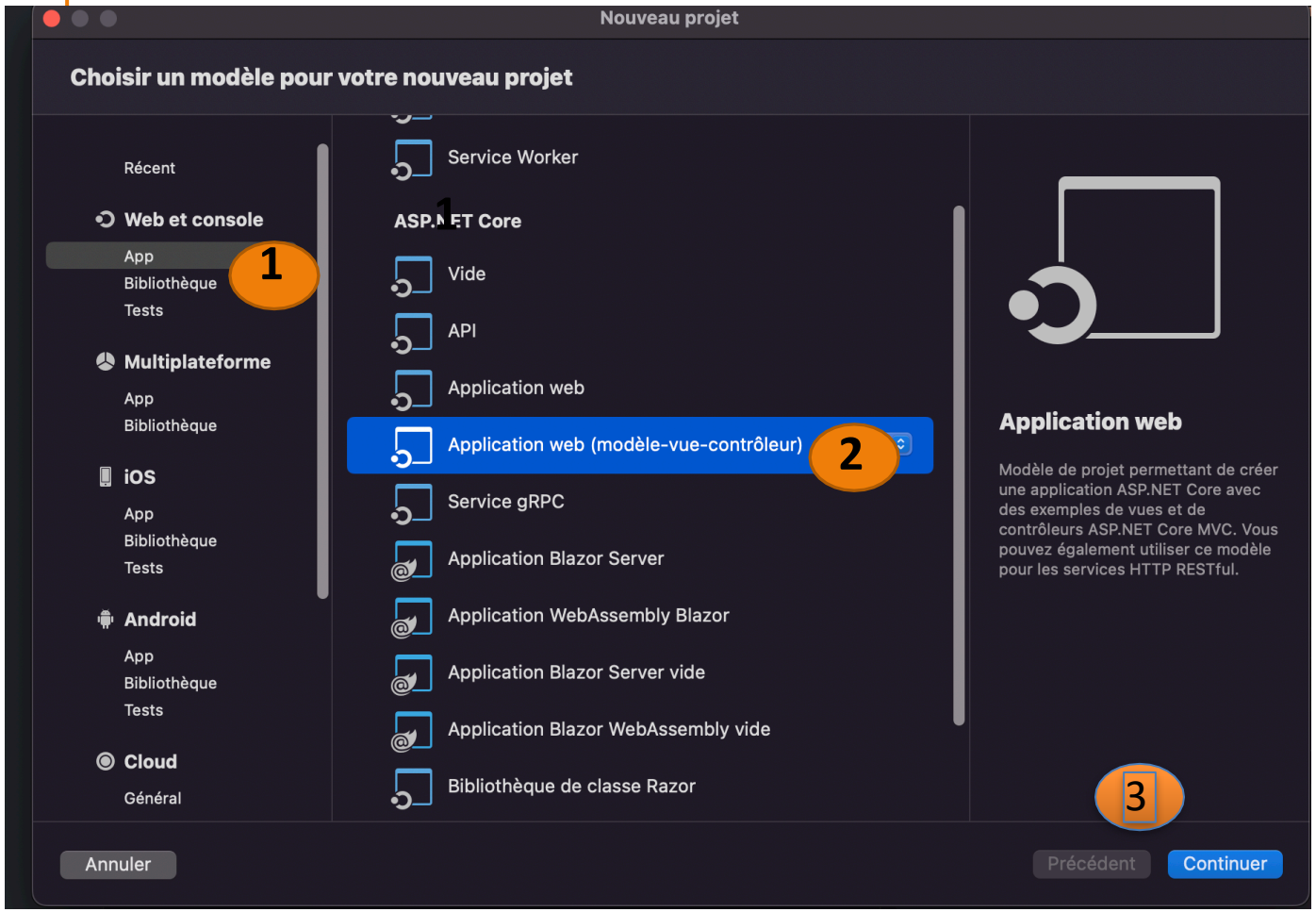


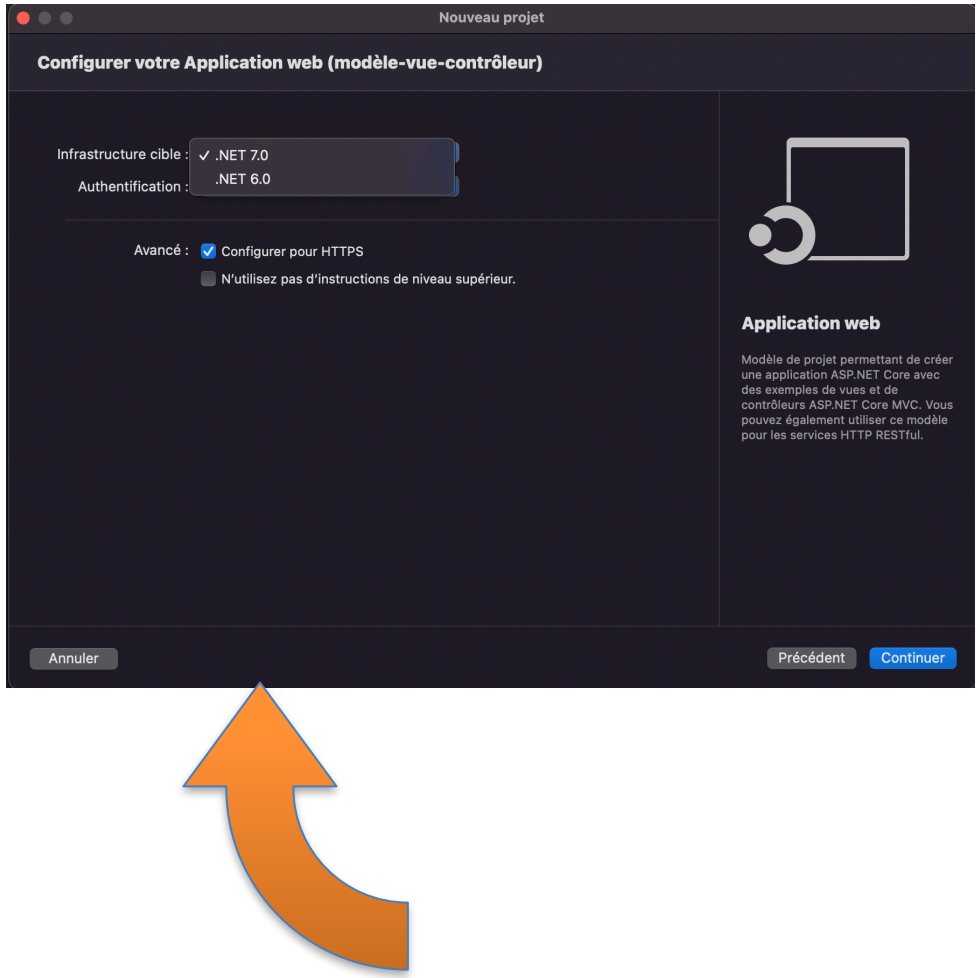
Asp.net Core MVC Cycle de Vie



ASP.NET Core MVC en pratique

Créer un nouveau projet ASP.net Core avec Template MVC

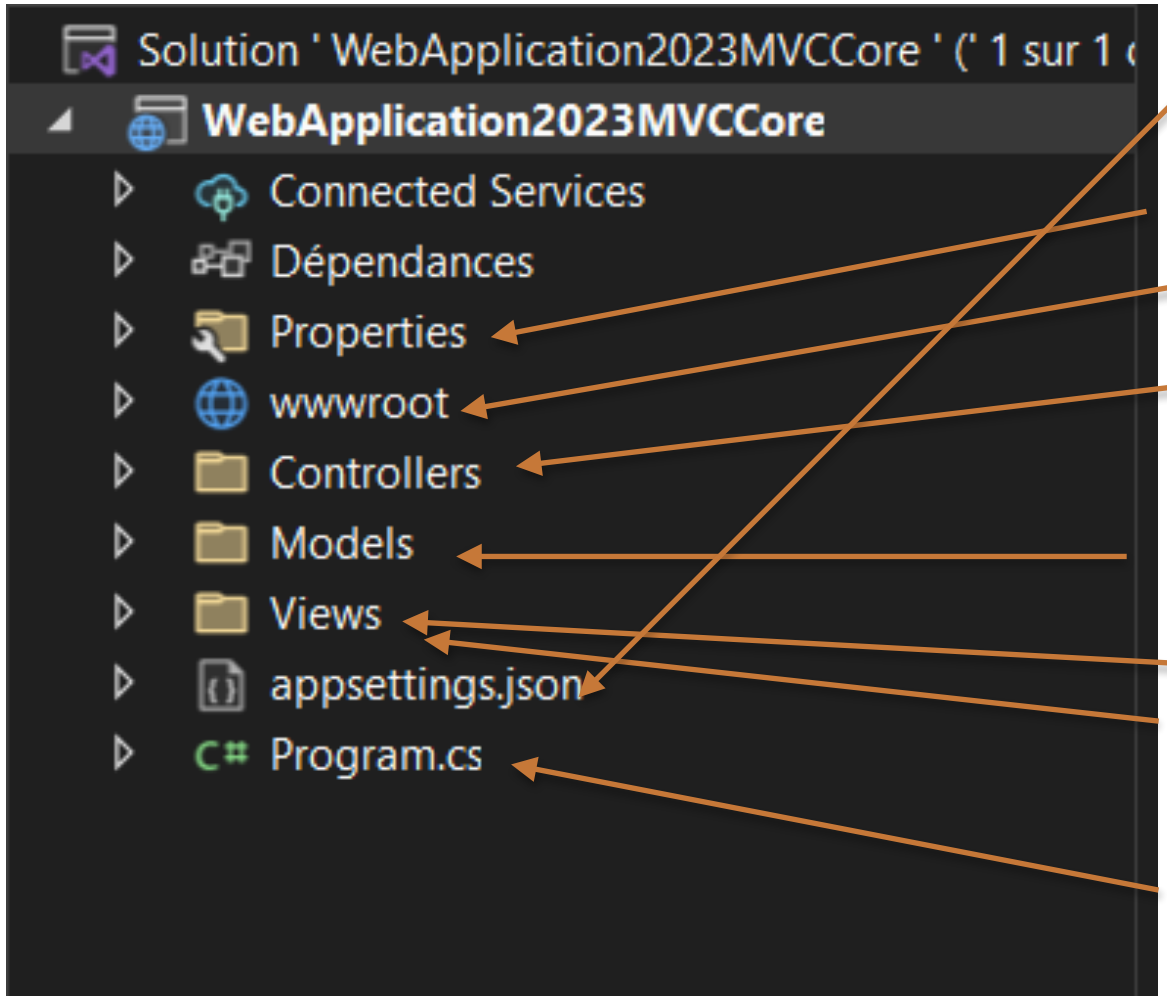




ASP.NET Core MVC : Arborescence

- **web.config is no more**
- **Global.asax is no more**
- **New configuration system**

command line
environment variables
JSON files



Le fichier de configuration

Les propriétés du projet

Fichiers CSS

Les contrôleurs (y incluses les actions) liant les modèles avec les vues

Les classes qui définissent les données

Les interfaces utilisateurs

Le template principal duquel hérite toutes les vues

Point d'entrée au programme

Configuration des packages installés à partir du Nuget package manager

Fichier de configuration du projet:
sécurité, connexion,…

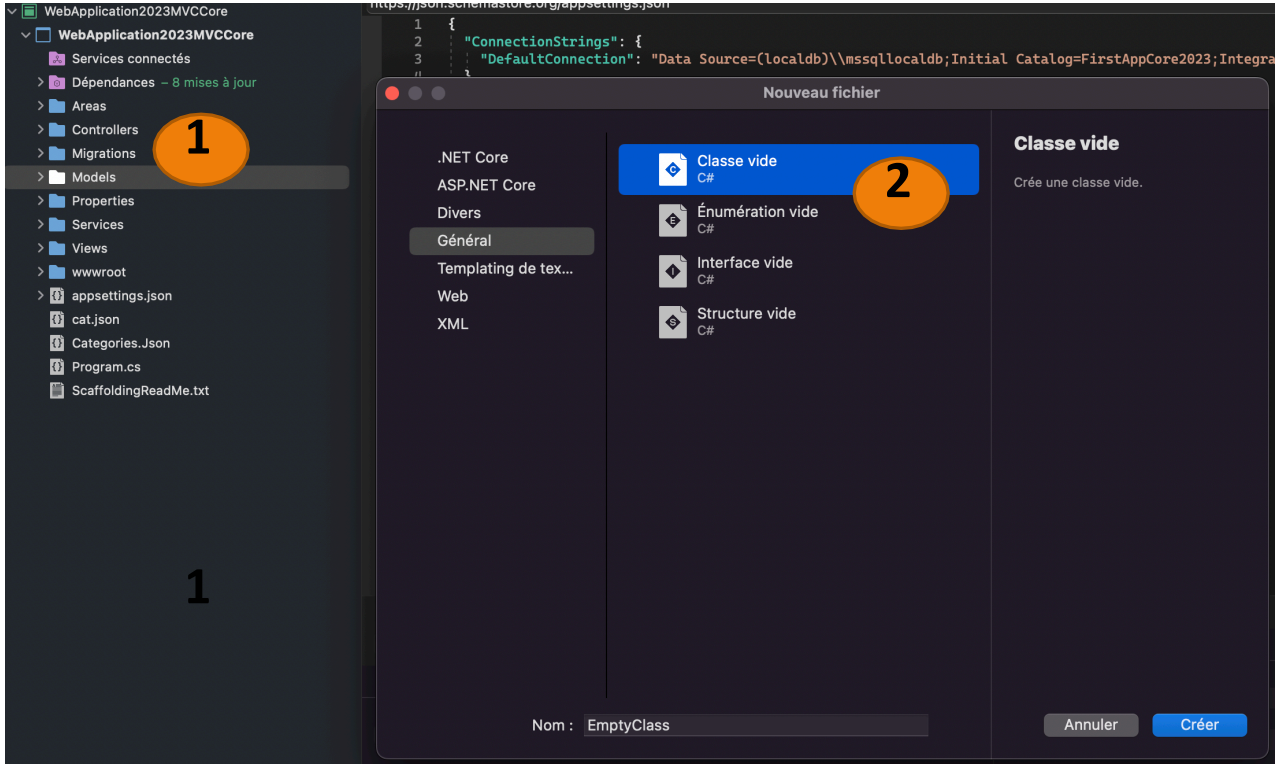
```
namespace ASP_APP1.Models
{
    1 référence
    public class Personne
    {
        int cin;
        string nom;
        string prenom;

        0 références
        public Personne(int c, string n, string p)
        {
            this.cin = c;
            this.nom = n;
            this.prenom = p;
        }
    }
}
```

ASP.NET Core MVC en pratique

CRÉER UN MODÈLE

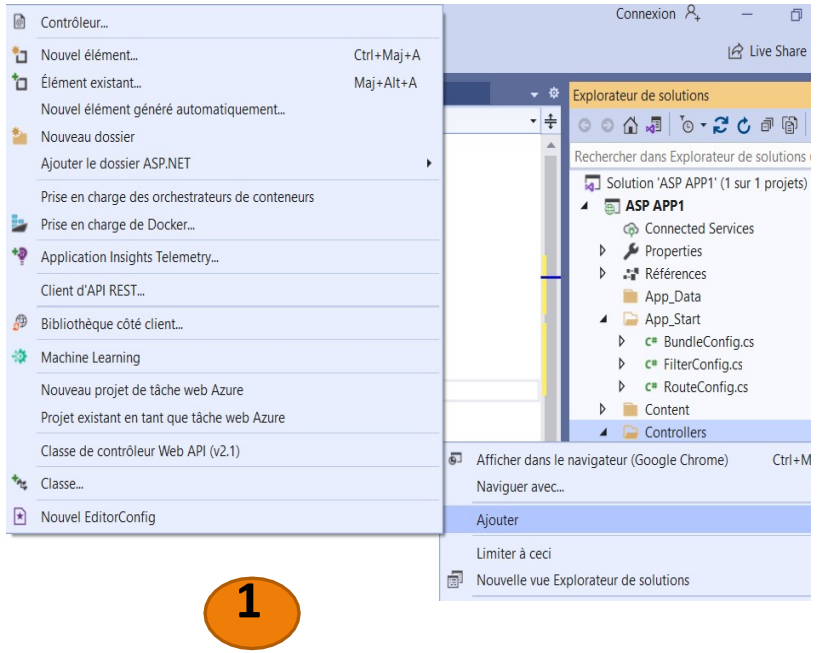
- **La couche modèle** sert à définir les **données** manipulées au niveau de l'application et représentées sous forme de classes.

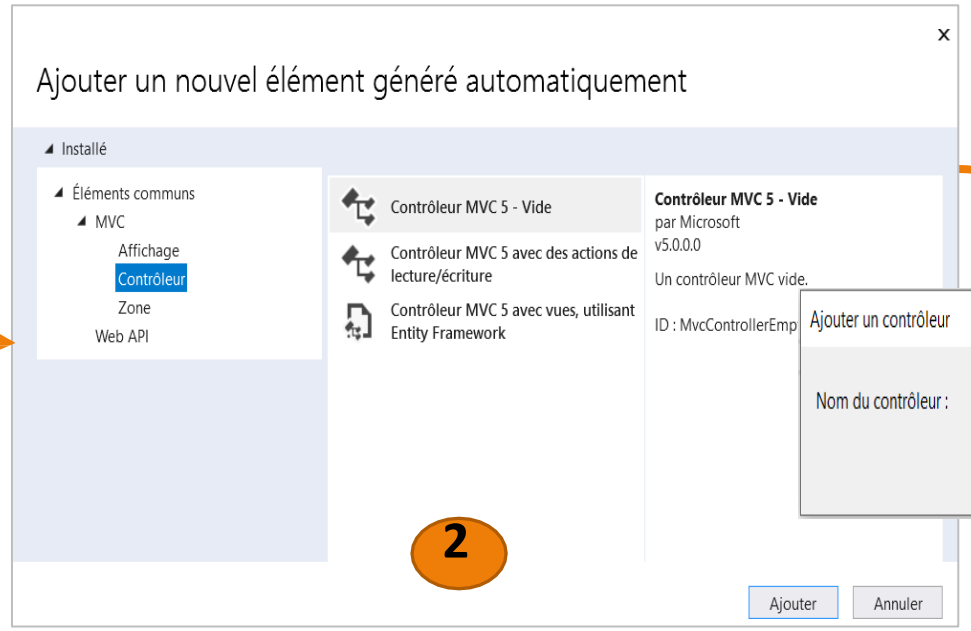


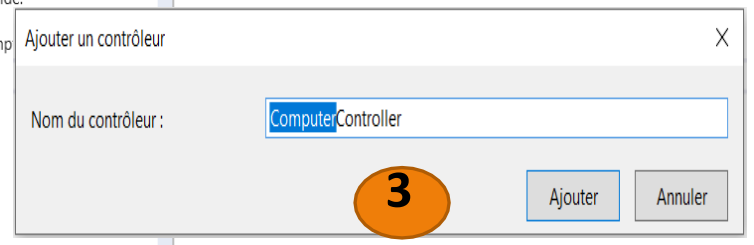

```
namespace ASP_APP1.Models
{
    0 références
    public class Computer
    {
        0 références
        public int Id { get; set; }
        0 références
        public string Name { get; set; }
        0 références
        public double Price { get; set; }
    }
}
```

ASP.NET Core MVC en pratique
CRÉER UN CONTRÔLEUR (1/3)

- **La couche contrôleur** sert à lier la couche modèle avec la couche vue, c'est une classe fille de la classe de base controller.







Par convention, le contrôleur prend le même nom que le modèle auquel il est lié.

ASP.NET Core MVC en pratique
CRÉER UN CONTRÔLEUR (2/3)

Par convention, le contrôleur prend le même nom que le modèle auquel il est lié.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using ASP_APP1.Models;

namespace ASP_APP1.Controllers
{
    0 références
    public class ComputerController : Controller
    {
        // GET: Computer
        0 références
        public ActionResult Display()
        {
            Computer MyComputer = new Computer();
            MyComputer.Id = 001;
            MyComputer.Name = "Lenovo";
            MyComputer.Price = 500;

            return View(MyComputer);
        }
    }
}
```

Dans ASP.NET MVC, les contrôleurs sont représentés comme des classes qui héritent de la classe de base Controller (a le suffixe « Controller » dans son nom).Les méthodes individuelles (appelées actions) correspondent à des URL individuelles.

indique que la vue doit être rendue

ASP.NET Core MVC en pratique
CRÉER UN CONTRÔLEUR (3/3)

À ce stade, on se demande **comment le framework sait comment mapper les URL à une action de contrôleur particulière?**

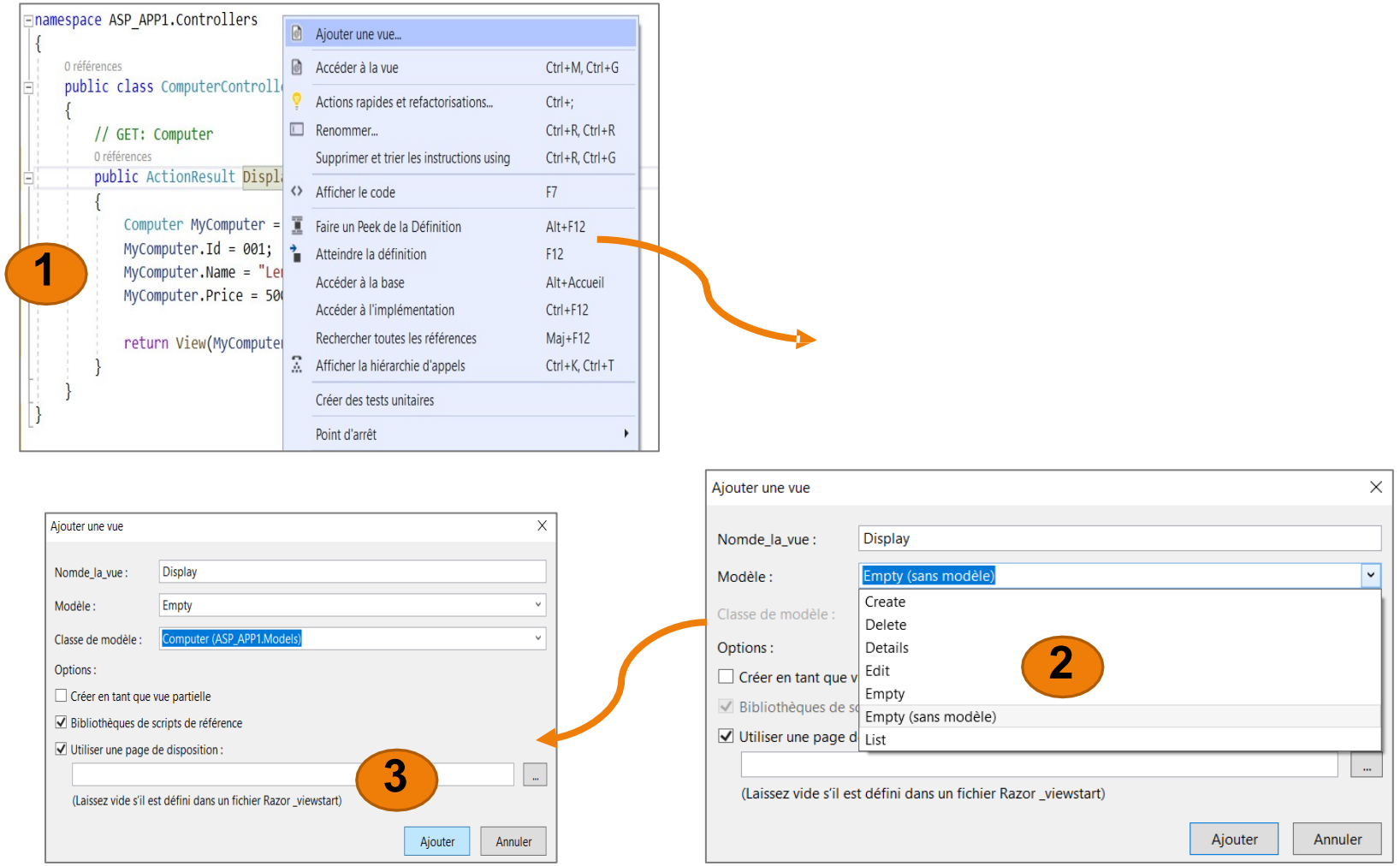
La réponse se trouve dans program.cs et plus particulièrement dans le pipeline. Cette méthode définit des routes par défaut qui mappent un modèle d'URL à un contrôleur ou à une action.

Program.cs

```
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
app.MapRazorPages();
```

ASP.NET Core MVC en pratique

CRÉER UNE VUE (1/3)



ASP.NET Core MVC en pratique
CRÉER UNE VUE (2/3)

Il existe un fichier nommé Index.cshtml qui réside dans le sous-répertoire Views/Home.

Si vous ouvrez ce fichier, vous verrez le balisage suivant dans le cadre d'Index.cshtml

La vue Index contient un mélange de code C# et de balisage HTML.

Le haut du fichier contient un bloc de code qui définit le titre de la page, puis un message s'affiche dans un élément <h2/>.

L'appel à @ViewBag.Message écrit le contenu de la propriété Message de ViewBag qui a été définie dans le manette.

```
@{
    ViewBag.Title = "Home Page";
}

@section featured {
<section class="featured">
    <div class="content-wrapper">
        <hgroup class="title">
            <h1>@ViewBag.Title.</h1>
            <h2>@ViewBag.Message</h2>
        </hgroup>
        <p>The current date is @ViewBag.CurrentDate.ToLongDateString()</p>
        <p>
            To learn more about ASP.NET MVC visit <a href="http://asp.net/
     mvc" title="ASP.NET MVC Website">http://asp.net/mvc</a>.
            ...
        </p>
    </div>
</section>
}
```

ASP.NET Core MVC en pratique
CRÉER UNE VUE (3/3)

Comment asp.Net Core MVC sait-il rendre une vue particulière plutôt que l'une des autres vues de l'application?

L'appel de la méthode View renvoie un objet View qui sait comment rendre une vue particulière.

Lorsque cette méthode est appelée sans arguments, le framework en déduit que le nom de la vue à rendre doit être le même que le nom de l'action .