

Séance 03 : Initiation à l'ASP.NET – Partie I

☒ Les fondamentaux

INF27507 – Technologies du commerce électronique

Prof. Yacine YADDADEN, Ph. D.

Plan

1. Introduction et Contexte
2. Programmation Web
3. Introduction à l'**ASP.NET**
4. Les notions de base d'**ASP.NET**
5. Questions et discussion

Introduction et Contexte

- **Contexte**

- Avènement d'*internet* et utilisation des *dispositifs de communication*,
- De plus en plus de *sites et d'applications Web* sont *déployés* chaque jour.

- **Problématique**

- Applications avec *fonctionnalités de plus en plus complexe*,
- Soucis de la *sécurité des données*,
- Développement rapide sans partir *from scratch* avec les *bonnes pratiques*.

- **Solution**

- Faire appel aux *technologies du Web* en termes de **bibliothèques** et **Framework**.

Plan

1. Introduction et Contexte
2. Développement Web
 - a. Quelques définitions
 - b. Fonctionnement et types d'architectures
 - c. Langages, bibliothèques et Framework
3. Introduction à l'**ASP.NET**
4. Les notions de base d'**ASP.NET**
5. Questions et discussion

Quelques définitions

- **Qu'est-ce que le Web :** « *ou World Wide Web (années 1980) est un système permettant l'organisation visuelle et la publication de documents, leur consultation via navigateur et leur interconnexion à l'aide de liens. Il a été développé par Tim Berners-Lee.* »,
- Trois technologies :
 - **Pages Web**
 - C'est le support du contenu textuel publié sur le Web.
 - **Lien hypertextes**
 - Ou **URL** (Uniform Resource Locator) permet la liaison entre différentes pages Web.
 - **Protocole HTTP** (HyperText Transfer Protocol)
 - Mécanisme permettant aux navigateurs Web d'accéder aux pages Web.

Web ≠ Internet



Quelques définitions (Suite)

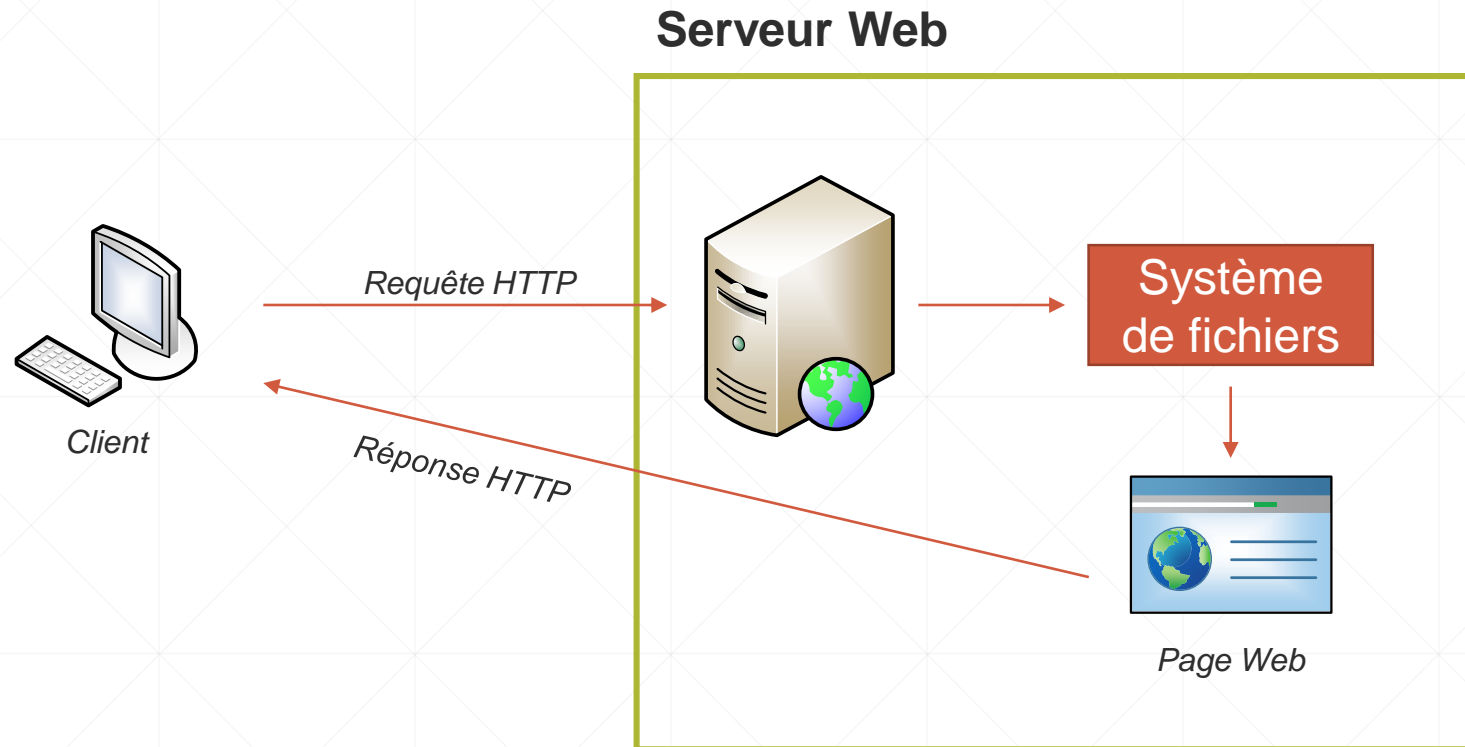
- **Qu'est-ce qu'un site Web :** « *c'est l'ensemble de pages web et de ressources reliées par des hyperliens accessible via une adresse Web.* »,
 - Il y a deux catégories : **Statique & Dynamique.**
- **Qu'est-ce qu'un navigateur Web :** « *c'est un logiciel, programme ou application côté client permettant à l'internaute de communiquer avec des logiciel serveurs afin d'obtenir de l'information (pages Web).* »,
- **Qu'est-ce qu'un serveur Web :** « *c'est un ordinateur robuste et sécurisé sur lequel un logiciel, également appelé serveur web est chargé de répondre aux requêtes des internautes en leur servant les pages voulues ou en renvoyant un message d'erreur.* »,



NGINX

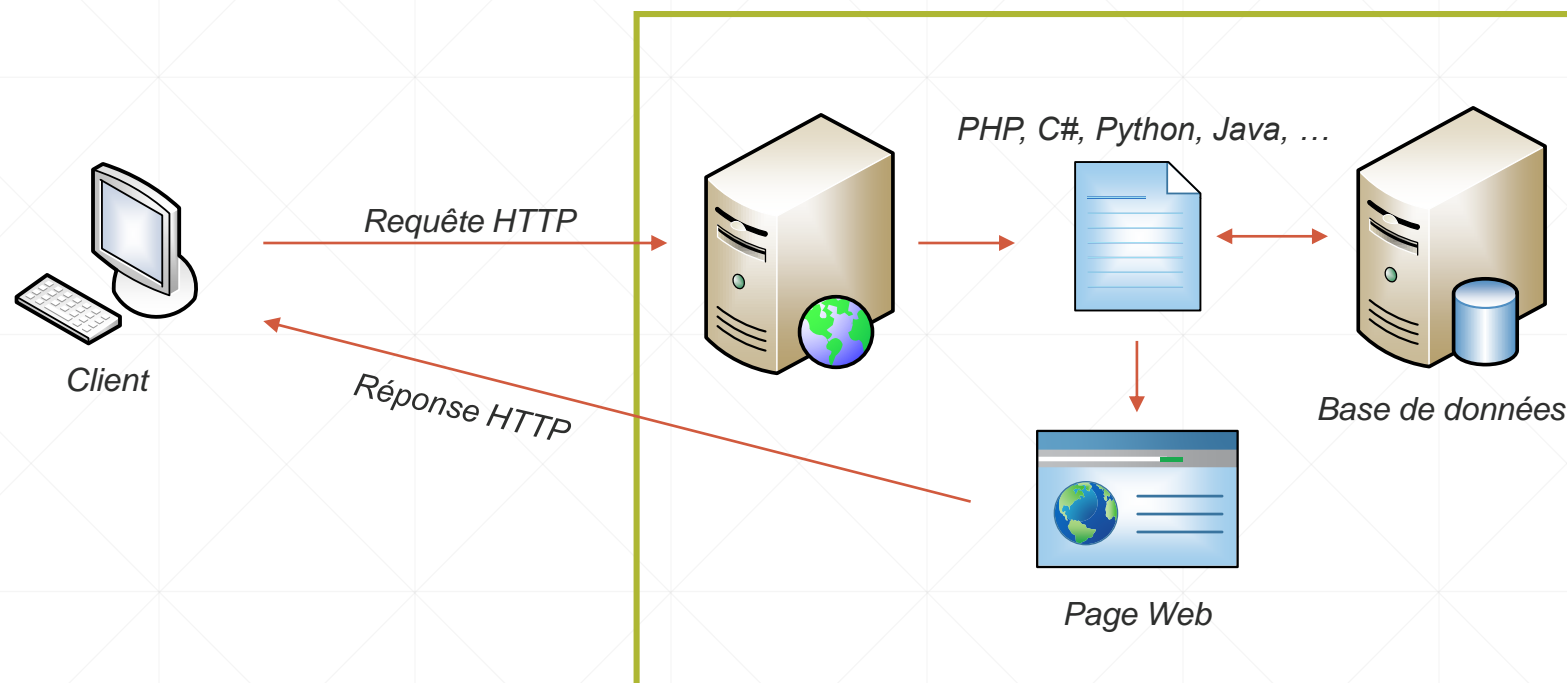


Fonctionnement et types d'architectures



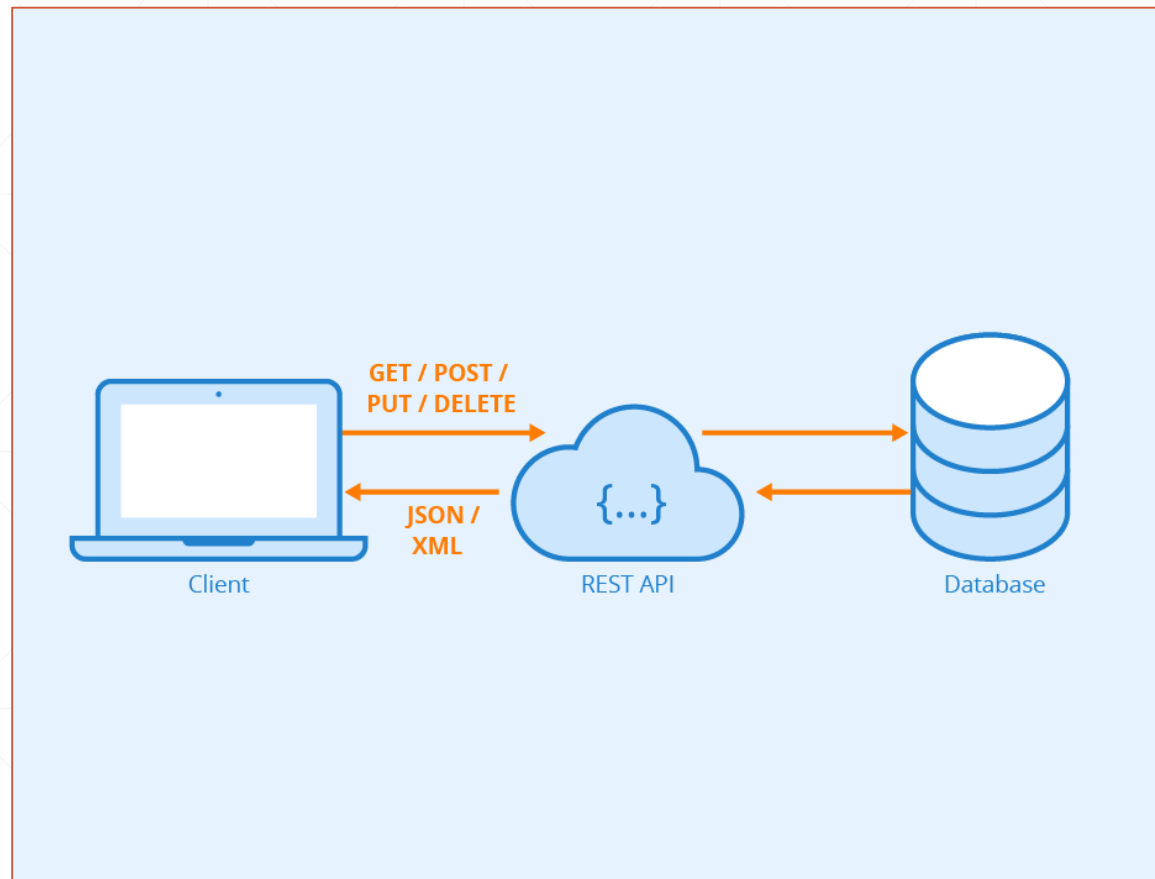
Site Web Statique

Fonctionnement et types d'architectures



Site Web Dynamique

Fonctionnement et types d'architectures



API REST

Langages, bibliothèques et Framework

Le développement Web se divise en deux catégories principales :

1. Front-end (*côté client*) :

- **Langages utilisés** : **HTML** (*structure*), **CSS** (*esthétique*) et **JavaScript** (*interaction*),
- **Outils JavaScript** : *jQuery, ReactJS, VueJS, AngularJS, ...*
- **Outils CSS** : *Twitter Bootstrap, Semantic UI, Material UI, ...*

2. Back-end (*côté serveur*) :

- **Langages utilisés** :
 - **Python** (*Django et Flask*), **Java** (*Java EE*), **C#** (*ASP.NET*), **PHP** (*Laravel*), **Ruby** (*Ruby on Rails*), ...
- **Persistance de données** :
 - Bases de données **relationnelles** (*MySQL et PostgreSQL*) et **non relationnelles** (*MongoDB*).

Plan

1. Introduction et Contexte
2. Programmation Web
3. Introduction à **ASP.NET**
 - a. Qu'est-ce que **ASP.NET** ?
 - b. Environnement de développement
 - c. Le paradigme **MVC**
4. Les notions de base d'**ASP.NET**
5. Questions et discussion

Qu'est-ce que ASP.NET MVC ?

- **Définition** : « *c'est un Framework destiné à la programmation Web développé et maintenu par Microsoft.* »,
- Initialement créé en **2007** par **Scott Guthrie**,
- Le paradigme de programmation **MVC** lui a été intégré en **2009**,
- La génération de pages Web dynamique se fait à travers la syntaxe **Razor**,
- Il est devenu *Open Source*,
- Exemple de site : **Stack Overflow** → <https://stackoverflow.com/>

Environnement de développement

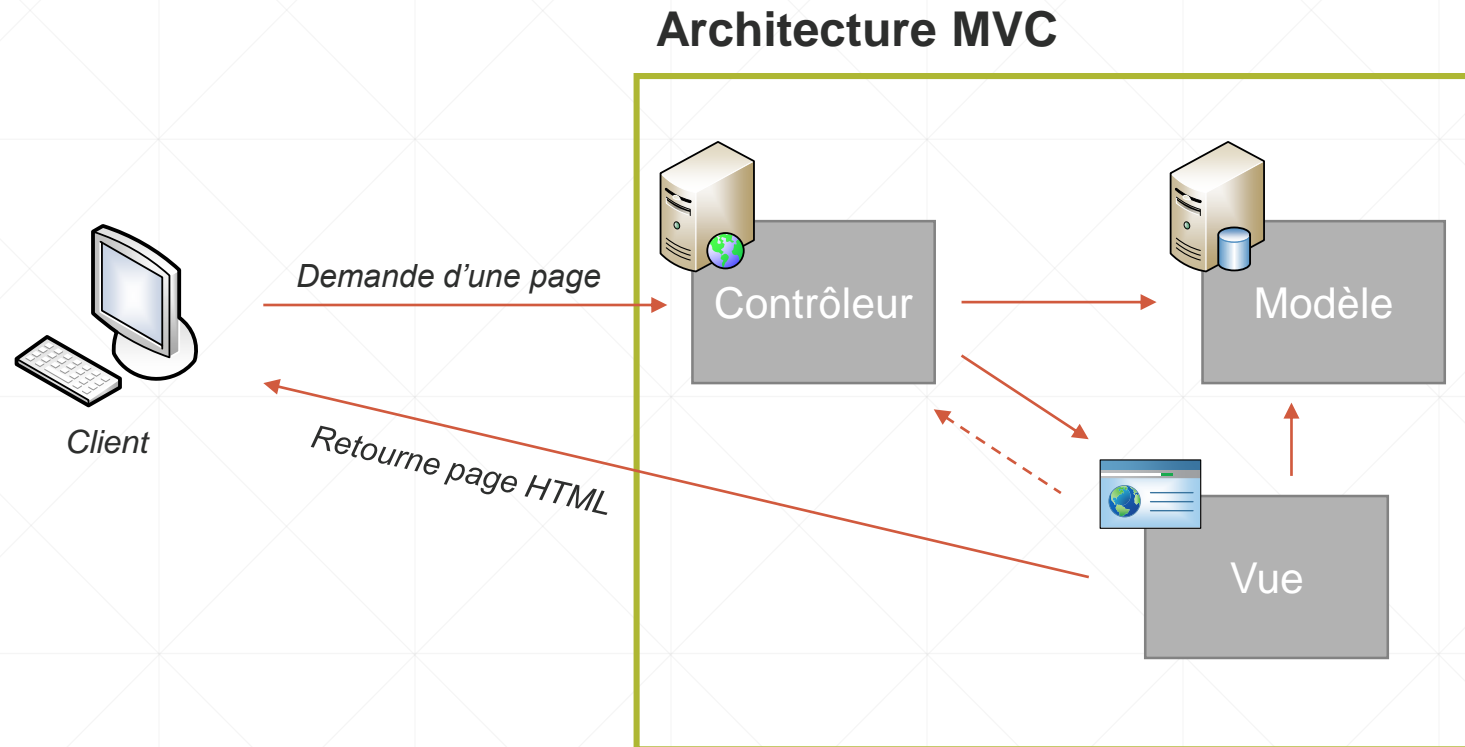
- Afin de développer des sites et application Web avec **ASP.NET**, il faut :
 - L'Environnement de Développement Intégré **Microsoft Visual Studio Community**
 - Lien : <https://visualstudio.microsoft.com/fr/vs/community/>
- Il faut s'assurer d'installer les paquets nécessaires :



Le paradigme MVC

- **Définition :** *« c'est un motif d'architecture logicielle ou design pattern destiné à la conception d'interface graphique lancé en 1978. Il est souvent utilisé pour la conception de sites et applications Web. »*,
- **Objectif :**
 - *Permettre une séparation entre les données et le rendu graphique,*
 - *Meilleure organisation et maintenabilité du code.*
- Il est composé de trois éléments principaux :
 1. Le **modèle** (Model) : *responsable de représentation et de l'accès aux données,*
 2. La **vue** (View) : *responsable du rendu graphique ou partie apparence,*
 3. Le **contrôleur** (Controller) : *permet d'ordonner les différentes actions.*

Aperçu du MVC



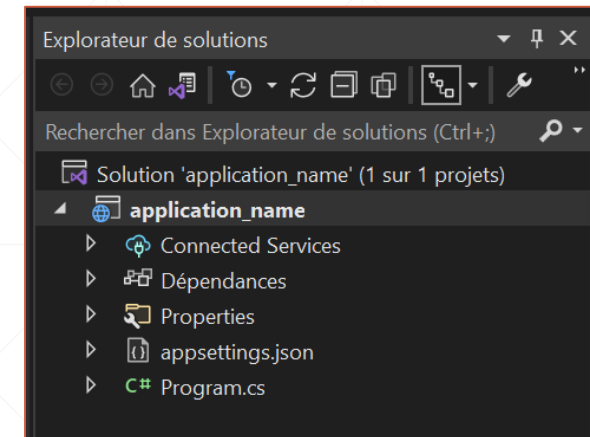
Plan

1. Introduction et Contexte
2. Programmation Web
3. Introduction à l'**ASP.NET**
4. Les notions de base d'**ASP.NET**
 - a. Création d'un projet **ASP.NET**
 - b. Traitement des requêtes **HTTP**
 - c. Gestion des dépendances (*back-end*)
 - d. Les fichiers statiques
 - e. Gestion des dépendances (*front-end*)
 - f. Quelques optimisations (*front-end*)
 - g. Utilisation du **MVC**
 - h. Les contrôleurs
 - i. Les vues avec **RAZOR**
 - j. Échange de données *vue ↔ contrôleur*
 - k. Validation des données
5. Mise en application
6. Questions et discussion

Création d'un premier projet ASP.NET MVC

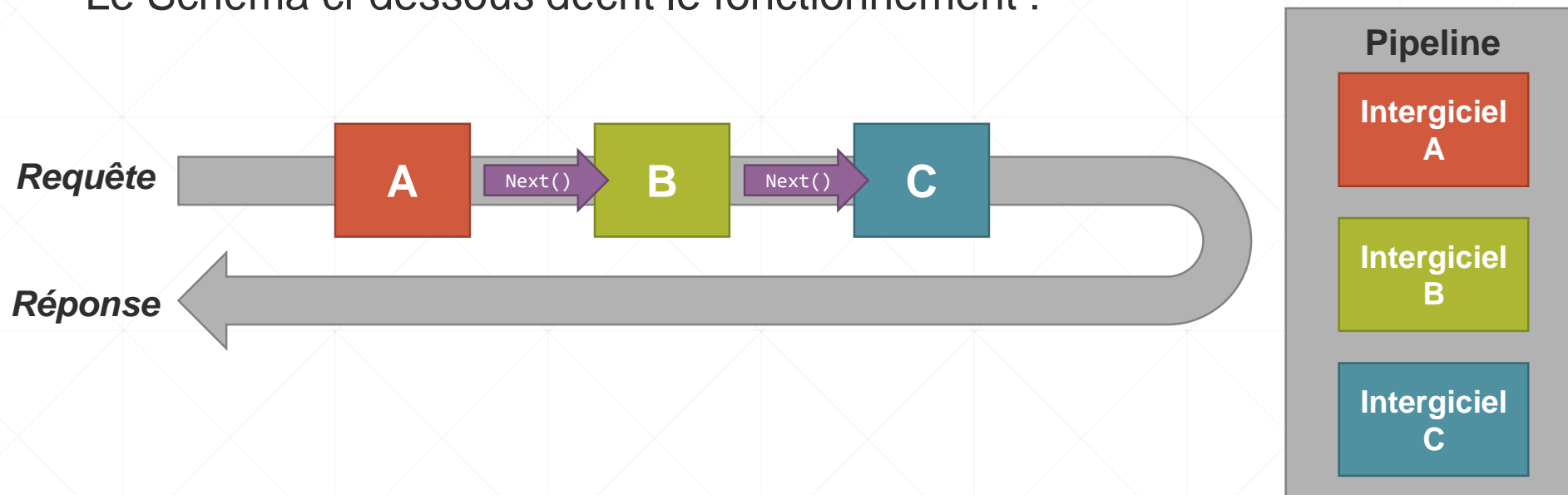
Afin de créer un premier projet en **ASP.NET**, il faut :

1. Lancer **Microsoft Visual Studio** puis *Create a new project*,
2. Il faut en suite sélectionner *ASP.NET Core vide*,
3. Donner un nom à votre projet, exemple : **application_name**,
4. Appuyer sur *Next*, puis :
 - Décocher *Configure for HTTPS*,
5. Appuyer sur *Create* et voilà !



Traitement des requêtes HTTP

- Les **requêtes HTTP** en **ASP.NET** sont gérées suivant un *principe de pipeline*,
- Il y a plusieurs *middlewares* (ou *intergiciels*) effectuant des tâches spécifiques,
- Le Schéma ci-dessous décrit le fonctionnement :



Traitement des requêtes HTTP

- Dans le fichier **Program.cs**, mettre le code suivant :

Middleware A



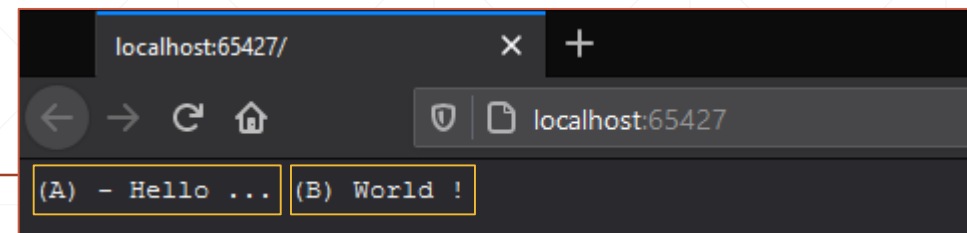
```
app.Use(async (context, next) =>
{
    await context.Response.WriteAsync("(A) - Hello ... ");
    await next();
});
```

Middleware B



```
app.Run(async (context) =>
{
    await context.Response.WriteAsync("(B) World !");
});
```

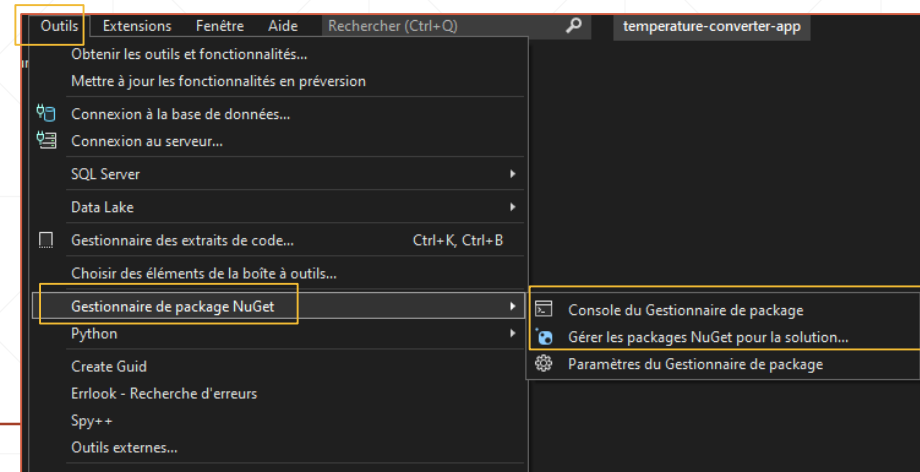
- Lors de l'exécution (serveur **IIS**), sur le navigateur :



Gestion des dépendances (*back-end*)

Afin d'installer des dépendances côté *back-end*, on passera par :

- Console de **gestion de paquets NuGet** :
 - **Aller** : *Outils > gestionnaire de packages NuGet > Console du Gestionnaire de package*
- Interface graphique pour la **gestion de paquets NuGet** :
 - **Aller** : *Outils > gestionnaire de packages NuGet > Gérer Les packages NuGet pour la solution ...*
- Commandes :
 - **Install-Package** *nom_du_package*
 - **Uninstall-Package** *nom_du_package*



Les fichiers statiques

- Il faut ajouter un nouveau fichier dans **wwwroot** de type **HTML** :
 - **Aller** : *Ajouter > Nouvel élément... > HTML page*
 - Lui donner un nom, exemple : **index.html**
- Dans le fichier **Program.cs**, mettre le code suivant :

```
app.UseFileServer();
```

- Lors de l'exécution :



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title></title>
  </head>
  <body>
    <h1>Hello World !</h1>
  </body>
</html>
```

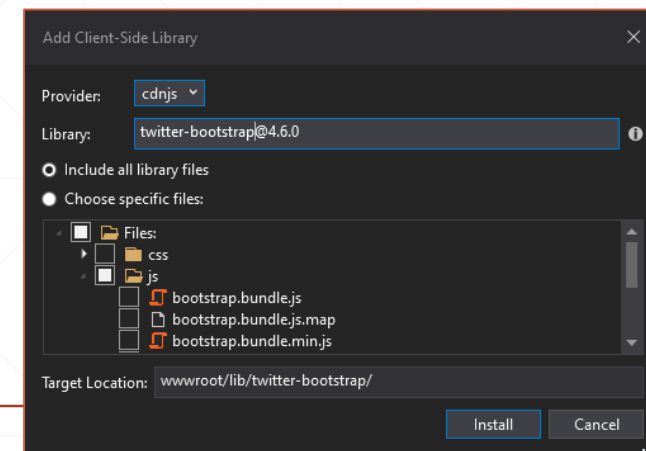
Gestion des dépendances (*front-end*)

Afin de gérer les dépendances côté *front-end*, on passera par :

- Habituellement, c'est **Bower**¹ qui était utilisé, mais Microsoft l'a changé,
- Maintenant, c'est **LibMan** qui est utilisé :
 - **Aller** : *Ajouter > Bibliothèque côté client...*
- Chercher la bibliothèque à installer, exemple : **twitter-bootstrap**,
- Ça va créer un fichier **libman.json**.

¹ <https://bower.io/>

```
{
  "version": "1.0",
  "defaultProvider": "cdnjs",
  "libraries": [
    {
      "library": "jquery@3.6.0",
      "destination": "wwwroot/lib/jquery/"
    },
    {
      "library": "twitter-bootstrap@4.6.0",
      "destination": "wwwroot/lib/twitter-bootstrap/"
    }
  ]
}
```



Quelques optimisations (*front-end*)


- Afin d'optimiser l'application Web au niveau des fichiers statiques, il y a :
 - **Regroupement** (*bundling*) :
 - *Permet de combiner plusieurs fichiers CSS, JavaScript, etc. L'objectif est d'accélérer le chargement de la page Web.*
 - **Minimisation** (*minification*) :
 - *Réduire la taille des fichiers CSS, JavaScript et Image en supprimant les caractères inutiles. Généralement, c'est les commentaires, les espaces, ...*
- Habituellement, c'est **Gulp**¹ qui était utilisé, mais Microsoft l'a changé,
- Il est nécessaire d'installer un paquet avec la commande :
 - **Install-Package BuildBundlerMinifier**
 - Ou utiliser le lien sur **Visual Studio Marketplace**²

¹ <https://gulpjs.com/>

² <https://marketplace.visualstudio.com/items?itemName=Failwyn.BundlerMinifier64>

Quelques optimisations (*front-end*)

- Afin d'ajouter les deux optimisations, il faut :
 - **Aller** : *Ajouter > Nouvel élément... > Fichier JSON*
 - Lui donner le nom de : **bundleconfig.json**
 - Ensuite, y mettre le contenu suivant :
- À chaque compilation, il y aura :
 - **Regroupement** des fichiers CSS et JavaScript,
 - **Minimisation** des fichiers générés.
- Il faut au préalable créer le dossier **wwwroot**,
- Il faut englober la configuration dans [].



```
{  
  "outputFileName": "wwwroot/css/site.min.css",  
  "inputFiles": [  
    "wwwroot/lib/twitter-bootstrap/css/bootstrap.css"  
  ],  
  "minify": { "enabled": true }  
},  
{  
  "outputFileName": "wwwroot/js/site.min.js",  
  "inputFiles": [  
    "wwwroot/lib/jquery/jquery.js",  
    "wwwroot/lib/twitter-bootstrap/js/bootstrap.js"  
  ],  
  "minify": { "enabled": true }  
}  
}
```

Utilisation du MVC

Afin de pouvoir exploiter le *paradigme MVC* dans une application **ASP.NET**, il faut :

- Définir dans le fichier **Program.cs** le service adéquat :

```
builder.Services.AddMvc(option => option.EnableEndpointRouting = false);
```

- Déclarer les différentes *routes* :

```
app.UseMvc(routes => routes.MapRoute("Default", "{controller=Home}/{action=Index}"));
```

nom

Contrôleur

Méthode

Les contrôleurs

- Le *contrôleur* est un des trois éléments principaux du *paradigme MVC*,
- Il faut commencer par ajouter une nouveau dossier **Controllers** au projet :
 - **Aller : Ajouter > Nouveau dossier**
- Ensuite, au niveau du dossier créé, il faut ajouter une *classe contrôleur* :
 - **Aller : Ajouter > Contrôleur > Contrôleur MVC – Vide**
 - Lui donner un nom suivant la diapositive précédente : **HomeController.cs**
 - nom*
 - partie commune*

Les contrôleurs

- Un code par défaut se trouve au niveau de la *classe contrôleur* créée,
- Pour faire un test simple, il faut changer le code qui s'y trouve :

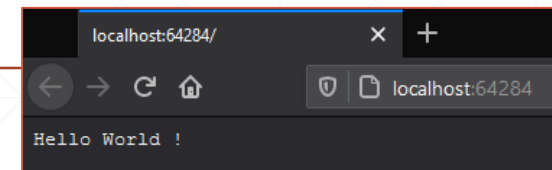
Dépendance

`using Microsoft.AspNetCore.Mvc;`

Code retournant
une chaîne de
caractères

```
public class HomeController : Controller
{
    public string Index()
    {
        return "Hello World !";
    }
}
```

Résultat



Les vues avec RAZOR

- **Définition :** « *c'est la syntaxe utilisée pour la création de page Web dynamiques avec le langage C#. Il a été introduit en 2011 et permet la génération des vues dans **ASP.NET MVC**.* »,
- La *vue* est un des trois éléments principaux du *paradigme MVC*,
- Il faut commencer par ajouter une nouveau dossier **Views** au projet :
 - **Aller :** *Ajouter > Nouveau dossier*
- Il faut commencer par ajouter une nouveau *sous-dossier* **Home** au projet :
 - **Aller :** *Ajouter > Nouveau dossier*
- Ensuite, au niveau du *sous-dossier* créé, il faut ajouter une nouvelle *vue* :
 - **Aller :** *Ajouter > Vue... > Vue Razor – Vide → Index.cshtml*

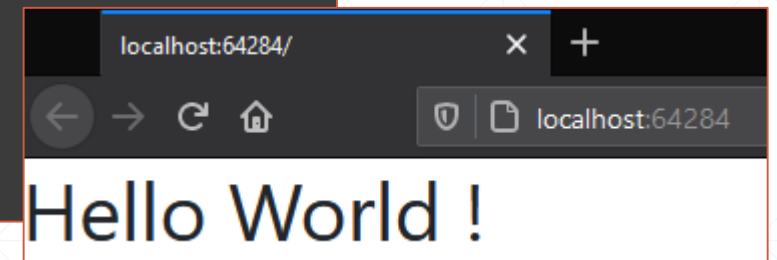
Les vues avec RAZOR – Première page

- Au niveau du fichier de vue créé, il faut mettre le code suivant :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title></title>
    <script src="~/js/site.min.js"></script>
    <link href="~/css/site.min.css" rel="stylesheet" />
  </head>
  <body>
    <h1>Hello World !</h1>
  </body>
</html>
```

Inclure les fichiers
CSS et JavaScript

Résultat



- Dans le fichier **Program.cs**, il faut mettre :

```
app.UseFileServer();
```

HomeController.cs

```
public class HomeController : Controller
{
    public IActionResult Index()
    {
        return View();
    }
}
```

Les vues avec RAZOR – Modèle

- Il est possible d'utiliser des *modèles* afin de réduire le code à écrire,
- Il faut commencer par ajouter une nouveau *sous-dossier* **Shared** au projet :
 - **Aller : Ajouter > Nouveau dossier**
- Ensuite, au niveau du *sous-dossier* créé, il faut ajouter une nouvelle *vue* :
 - **Aller : Ajouter > vue... > Vue Razor – Vide → `_Layout.cshtml`**

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title></title>
    <script src="~/js/site.min.js"></script>
    <link href="~/css/site.min.css" rel="stylesheet" />
  </head>
  <body>
    @RenderBody()
  </body>
</html>
```

```
@{
    Layout = "_Layout";
}

<h1>Hello World !</h1>
```

Index.cshtml



Les vues avec RAZOR – Les vues *partielles*

- Il est possible de réutiliser des *bouts* de vues,
- Ensuite, au niveau du sous-dossier **Shared**, il faut ajouter une nouvelle *vue* :
 - **Aller : Ajouter > Vue... > Vue Razor – Vide** → **_Title.cshtml**
- Y mettre le code suivant : `<h1>Hello World !</h1>`
- Au niveau de la page **Index.cshtml** on aura le code :

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@{
    Layout = "_Layout";
}

<partial name="_Title" />
```

← Permettre l'utilisation
de balises spéciales

Échange de données : *vue* ← *contrôleur*

Il y a principalement deux moyens de transmettre des données à partir du *contrôleur* vers la *vue* :

- **Méthode *brute*** : *c'est la plus facile à utiliser, on utilisera le conteneur ViewBag pour y insérer des données :*

```
public class HomeController : Controller
{
    public IActionResult Index()
    {
        ViewBag.Title = "This is my title !";
        return View();
    }
}
```

HomeController.cs

```
@{
    Layout = "_Layout";
}
<h1>@ViewBag.Title</h1>
```

Index.cshtml

Échange de données : *vue* ← *contrôleur*

- **Méthode typée (recommandé)** : *elle est différente de la première méthode dans le sens où l'on fait passer un **objet** qu'on aura créé au préalable dans le **modèle** :*
 - Il faut commencer par ajouter une nouveau dossier **Models** au projet :
 - **Aller** : *Ajouter* > *Nouveau dossier*
 - Ensuite, il faut créer un nouveau *modèle* **PageInfo.cs** en passant par :
 - **Aller** : *Ajouter* > *Classe...*
 - Avec le code suivant :

PageInfo.cs

```
public class PageInfo
{
    public string Title { get; set; }
    public int PageNumber { get; set; }
}
```

Échange de données : *vue* ← *contrôleur*

Si on veut afficher une petite liste transmise du *contrôleur* vers la *vue* :

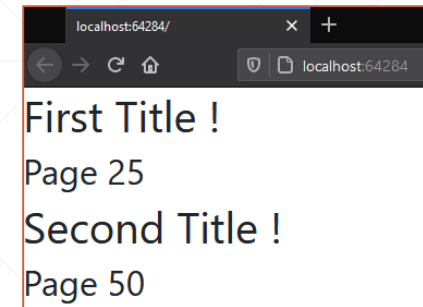
```
public IActionResult Index()
{
    List<Models.PageInfo> pageInfo = new List<Models.PageInfo>() {
        new Models.PageInfo() { Title = "First Title !", PageNumber = 25 },
        new Models.PageInfo() { Title = "Second Title !", PageNumber = 50 }
    };

    return View(pageInfo);
}
```

HomeController.cs

_Title.cshtml

```
@model application_name.Models.PageInfo
<h1>@Model.Title</h1>
<h2>Page @Model.PageNumber</h2>
```



Index.cshtml

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@{
    Layout = "_Layout";
}
@model List<application_name.Models.PageInfo>
@foreach (var pageInfo in Model)
{
    <partial name="_Title" model=pageInfo />
}
```

Échange de données : *vue* → *contrôleur*

- On utilisera pour cela des *formulaire*s **HTML**,
- Il y a deux principales approches pour faire des *formulaire*s en **ASP.NET** :
 - **La syntaxe RAZOR** : *elle permet d'écrire moins et de bénéficier des fonctionnalités de RAZOR, par contre on a moins de contrôle sur les éléments du formulaire.*
 - **Les balises spéciales** :
 - Créer trois **actions** (*méthodes*) dans le **contrôleur** :
 - **Add()** en mode **GET** pour l'affichage du formulaire,
 - **Add()** en mode **POST** pour traiter les données,
 - **Result()** en mode **GET** pour l'affichage des résultats.
 - Il faut également créer deux **vues** :
 - **Add.cshtml**
 - **Result.cshtml**

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

[HttpPost]
public IActionResult Add(models.PageInfo pageInfo)
{
    return RedirectToAction("Result", pageInfo);
}

[HttpGet]
public IActionResult Result(models.PageInfo pageInfo)
{
    return View(pageInfo);
}
```

HomeController.cs

Échange de données : *vue* → *contrôleur*

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@{
    Layout = "_Layout";
}

@model application_name.Models.PageInfo

<form asp-action="Add">
    <div class="form-group">
        <label asp-for="Title">Titre</label>
        <input type="text" asp-for="Title" class="form-control" />
    </div>

    <div class="form-group">
        <label asp-for="PageNumber">Numéro de Page</label>
        <input type="number" asp-for="PageNumber" class="form-control" />
    </div>

    <button type="submit" class="btn btn-primary">Ajouter une page</button>
</form>
```

Add.cshtml

Résultat

Result.cshtml

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@{
    Layout = "_Layout";
}

@model application_name.Models.PageInfo

<partial name="_Title" model=Model />
```


Validation des données

La validation des données issues des formulaires se fait comme suit :

- Au niveau du *modèle* ou le fichier suivant :

Dépendance →

Champs requis →

Message à afficher →

```
PageInfo.cs
using System.ComponentModel.DataAnnotations;

namespace to_do_list_app.Models
{
    public class PageInfo
    {
        [Required(ErrorMessage = "Champs requis !")]
        public string Title { get; set; }

        [Required(ErrorMessage = "Champs requis !")]
        public int? PageNumber { get; set; }
    }
}
```

Validation des données

- Au niveau du *contrôleur* ou le fichier suivant : HomeController.cs

Validation

```
[HttpPost]
public IActionResult Add(models.PageInfo pageInfo)
{
    if(ModelState.IsValid)
        return RedirectToAction("Result", pageInfo);

    return View();
}
```

- Au niveau de la *vue* ou le fichier suivant :

Add.cshtml

Validation du titre

Validation du numéro

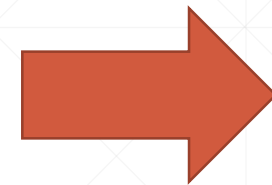
```
<form asp-action="Add">
    <div class="form-group">
        <label asp-for="Title">Titre</label>
        <input type="text" asp-for="Title" class="form-control" />
        <span asp-validation-for="Title" class="text-danger"></span>
    </div>

    <div class="form-group">
        <label asp-for="PageNumber">Numéro de page</label>
        <input type="number" asp-for="PageNumber" class="form-control" />
        <span asp-validation-for="PageNumber" class="text-danger"></span>
    </div>

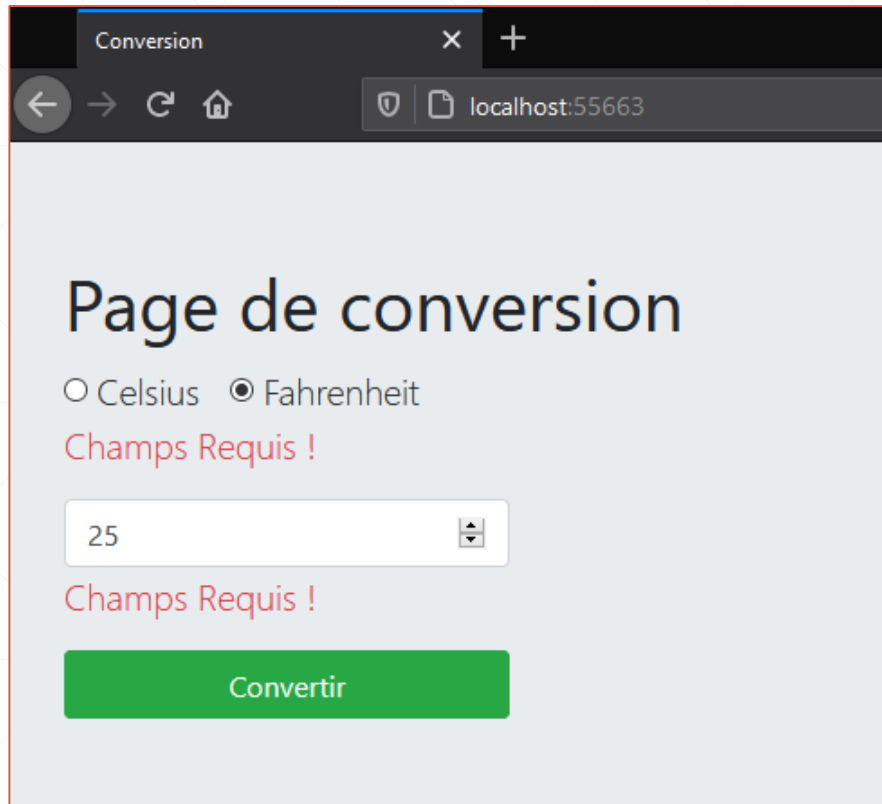
    <button type="submit" class="btn btn-primary">Ajouter une page</button>
</form>
```

Mise en application

- Créer une application permettant une conversion entre :
 - **Degré Celsius °C** et **Degré Fahrenheit °F**
- Pour cela, il faut utiliser :
 - Création d'un *contrôleur*,
 - Création d'un *modèle*,
 - Création des *vues* nécessaires.
- Le résultat attendu est comme suit :



Mise en application – Conversion de température



Conversion

Page de conversion

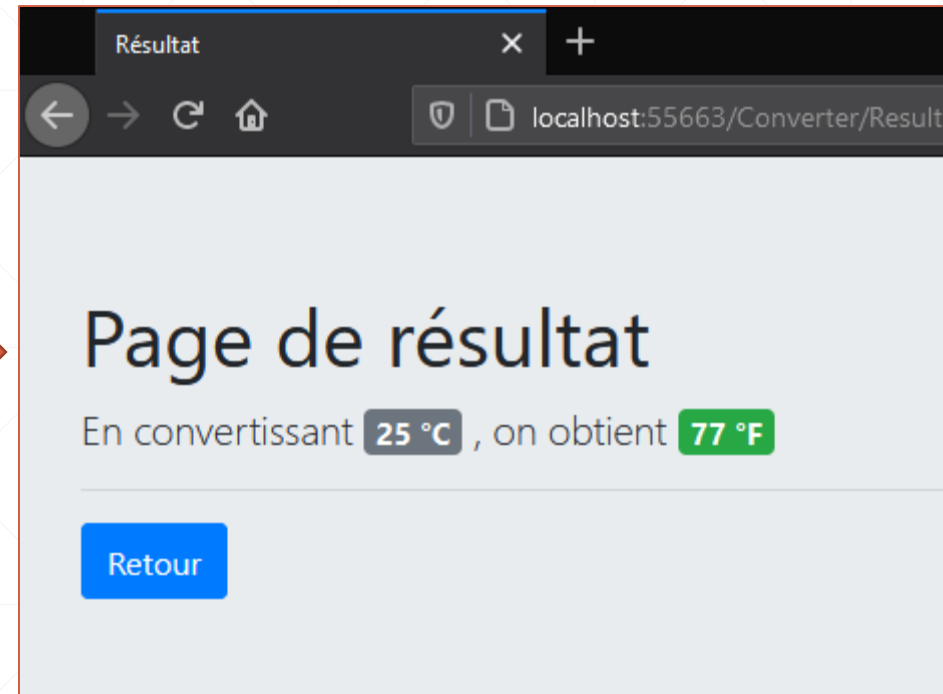
☐ Celsius ☒ Fahrenheit

Champs Requis !

25

Champs Requis !

Convertir



Résultat

Page de résultat

En convertissant 25 °C , on obtient 77 °F

Retour

Dépôt GitHub : <https://github.com/yyaddaden/asp.net-intro-demo>

Exercice à faire – Énoncé

En vous basant sur ce l'exemple fournis, il vous est demandé de :

1. Avoir une liste de livres disponibles (dans une liste),
2. Faire une recherche sur la base de l'*auteur*, *éditeur* ou *titre*,
3. Retourner les livres si disponibles ou un message si ce n'est pas le cas.

Optionnel :

1. Rajouter pour chaque livre sa couverture en image,
2. Ajouter un bouton de retour dans l'affichage des résultats,
3. Utilisation des icônes, voir ceux fournis par Twitter Bootstrap 5 ou Font-Awsome.

Exercice à faire – Liste des livres

- **Victor Hugo**

- Han d'Islande (Plume de Carotte)
- Le dernier Jour d'un Condamné (Arvensa)

- **J.R.R. Tolkien**

- Le Silmarillion (Bourgois)
- Le Seigneur des anneaux : Les Deux Tours (Bourgois)
- Le Seigneur des anneaux : Le Retour du roi (Bourgois)

- **Oscar Wilde**

- Le Portrait de Dorian Gray (Le livre qui parle)

Exercice à faire – Aperçu

Barre de recherche

Message d'erreur quand indisponible

Affichage des résultats quand disponible

Image	Informations :
	Titre : Le Silmarillion Auteur : J.R.R. Tolkien Éditeur : Bourgois
	Titre : Le Seigneur des anneaux : Les Deux Tours Auteur : J.R.R. Tolkien Éditeur : Bourgois
	Titre : Le Seigneur des anneaux : Le Retour du roi Auteur : J.R.R. Tolkien Éditeur : Bourgois

Questions & Discussion

Bibliographie

1. Guérin, B., A. (2016). *ASP.NET avec C# sous Visual Studio 2015 - Conception et développement d'applications Web*. Éditions ENI.
2. Labat, L. (2013). *ASP.NET MVC 4 - Développement d'applications Web en C# - Concepts et bonnes pratique*. Éditions ENI.