Applications Web MVC

- Présentation de ASP .NET Core MVC
- Controlleurs & Routes
- Vues
- Soumission de formulaires
- Session Http

Présentation de ASP .NET Core MVC

Le pattern MVC

Comme son nom l'indique, ce framework se base sur le pattern MVC, qui divise l'architecture de l'application en trois parties :

- Modèle : les données de l'application
- Vue : les pages web de l'application
- Controlleur : gestion des actions des utilisateurs, coordonne données et pages

Créer un Project

Pour créer un nouveau projet d'application web :

```
dotnet new mvc -o "mon-projet"
```

Explication du layout du projet

- Dossier `Controllers` : Contient les classes controlleurs de votre application. Vous pouvez supprimer le fichier d'exemple créé par le projet
- Dossier `Models` : Contient les classes de logique et de données de votre application. Vous pouvez supprimer le fichier d'exemple créé par le projet
- Dossier `Views` : Contient les templates HTML du projet. Vous pouvez supprimer le dossier `Home` d'exemple créé par le projet
  - Dossier `Shared` : Contient les éléments de template ré-utilisables :
    - `_Layout.cshtml` : Squelette de toutes vos pages, l'appel `@RenderBody()` rend le contenu de la page. Vous pouvez vider les `<header>` et `<footer>` d'exemple créés par le projet, et les remplacer par votre propre `<header>` et `<footer>`. Tout ce que vous mettrez dans ce template sera rendu sur toutes les pages de votre application
    - `_Layout.css` : le CSS propre à votre squelette de pages
    - `_ValidationScriptsPartial.cshtml` et `Erros.cshtml` : vous pouvez les supprimer, ce sont des exemples du projet

    - `_ViewImports.cshtml` : Imports des namespaces dans les templates. Comme on a vider le namespace `Models`, son import ne compile plus, vous pouvez le commenter pour l'instant (la syntaxe pour les commenaire ici est `@* commenaire *@`
    - `_ViewStart.cshtml` est le point d'entrée de vos template
- Dossier `wwwroot` : les fichiers statiquement servis par votre application
    - `css` : le CSS de votre application
    - `js` : le JS de votre application
    - `lib` : les libraires JS et CSS téléchargées localement
    - `favicon.ico` : l'icone de votre site
- `Program.cs` : le point d'entré de votre application

Controlleurs & Routes

Premier controlleur

Pour créer un controlleur, créez une classe dont le nom finit par "Controller" dans le dossier `Controllers`. Par exemple, "HelloWorldController", qui étend la classe `Controller`, par exemple :

```
namespace net_web_tuto_2.Controllers ;
public class HelloWorldController : Controller
{

}
```

Pour ajouter un préfixe de route à votre controleur, mettez lui l'attribut [Route("maroute")], par exemple :

```
    [Route("hello-world")]
    public class HelloWorldController : Controller
    {

    }
```

Ensuite, vous pouvez créer une méthode endpoint, avec l'annotation `[HttpGet]` :

```
        [HttpGet]
        public string HelloWorld()
        {
            return "Hello world !";
        }
```

La route de ce endpoint est dont `GET /hello-world`. On peut constater en lançant l'application avec un :

```
dotnet run
```

Et en en allant à https://localhost:7229/hello-world

On peut donner des suites de route particulières aux méthodes endpoint :

```
    [HttpGet("fr")]
    public string HelloWorldFr()
    {
        return "Bonjour, le monde";
    }

    [HttpGet("en")]
    public string HelloWorldEn()
    {
        return "Hello, world";
    }
```

On peut ensuite constater le résultat en redémarrant l'app (CTRL + C dans la console puis `dotnet run`). Et en allant à :

- https://localhost:7229/hello-world/fr
- https://localhost:7229/hello-world/en

Paramètres de requête et endpoints

Paramètre de requête

Pour récupérer une paramètre de requête, il suffit de mettre un paramètre avec le même nom à la méthode endpoint :

```
    [HttpGet]
    public string HelloWorld(string name)
    {
        return $"Hello, {name}";
    }
```

On peut ensuite constater le résultat en redémarrant l'app (CTRL + C dans la console puis `dotnet run`). Et en allant à https://localhost:7229/hello-world?name=Arsène.

Paramètre de chemin

Pour récupérer un paramètre de chemin, il suffit de mettre ce paramètre entre accolades dans la route du endpoint, puis ajouter n paramètre avec le même nom à la méthode endpoint :

```
    [HttpGet("{id}")]
    public string HelloWorld(int id)
    {
        return $"Id : {id}";
    }
```

On peut ensuite constater le résultat en redémarrant l'app (CTRL + C dans la console puis `dotnet run`). Et en allant à https://localhost:7229/hello-world/1.

Vues

Retourner une vue

Avant de retourner une vue, il faut d'abord créer un dossier dans `Views` pour notre controleur qui correspond à son nom. Ici notre controleur s'appelle `HelloWorldController`, notre dossier va donc s'appeller `HelloWorld`.

On va ensuite créer dedans un fichier `hello-world.cshtml`, qui sera notre template :

```
<h1>Hello, world</h1>
```

Pour le retourner depuis une méthode endpoint, il faut utiliser la fonction `View()` du controleur :

```
    [HttpGet]
    public IActionResult HelloWorld()
    {
        return View("hello-world");
    }
```

On peut ensuite constater le résultat en redémarrant l'app (CTRL + C dans la console puis `dotnet run`). Et en allant à https://localhost:7229/hello-world.

Templater une vue

Avec une classe modèle

On peut templater une vue avec un modèle de façon fortement typée, ce qui est pratique car cela permet de répérer d'eventuelles erreurs plus vites.

Soit la classe de modèle (dans le dossier `Model`, et n'oublier pas de décomenter l'import du namespace dans `_ViewImports.cshtml`) :

```
namespace net_web_tuto.Models {
    public class Person {
```

```
        public string FirstName { get; set; }
        public string LastName { get; set; }
    }
}
```

Dans la méthode endpoint, on ajouter le modèle dans l'appel à `View` :

```
    [HttpGet]
    public IActionResult HelloWorld()
    {
        var person = new Person(){ FirstName = "John", LastName = "Shepard"};
        return View("hello-world", person);
    }
```

Et pour le template, la syntaxe est la suivante :

```
<h1>Hello, @Model.FirstName @Model.LastName</h1>
```

On peut ensuite constater le résultat en redémarrant l'app (CTRL + C dans la console puis `dotnet run`). Et en allant à https://localhost:7229/hello-world.

Avec d'autres données

Pour templater la vue avec d'autres infos que celles d'un modèle, on peut utiliser le `ViewBag`, exemple :

Dans la méthode endpoint, on ajouter le modèle dans l'appel à `View` :

```
    [HttpGet]
    public IActionResult HelloWorld()
    {
        var person = new Person(){ FirstName = "John", LastName = "Shepard"};
      ViewBag.Today = DateTime.Now;
        return View("hello-world", person);
    }
```

```
<h1>Hello, @Model.FirstName @Model.LastName, The Date is : @ViewBag.Today</h1>
```

On peut ensuite constater le résultat en redémarrant l'app (CTRL + C dans la console puis `dotnet run`). Et en allant à https://localhost:7229/hello-world.

Rendu conditionnel

Pour faire du rendu conditionnel il faut utiliser `@if` :

```
@if((20 % 2) == 0) {
<p> 20 is even </p>
}

@if((21 % 2) == 0) {
<p> 21 is not even </p>
}
```

Seulement "20 is even" sera rendu sur la page.

Boucles

Pour faire du rendu en boucle, il faut utiliser `@foreach`, exemple :

```
@foreach (var person in Model.Persons)

    <div>@item.FirstName @item.LastName</div>
}
```

Soumission de formulaires

Premier Formulaire

Pour gérer une soumission de formulaire, il faut un modèle dont les noms des champs correspondent aux champs du formulaire, puis ajouter ce modèle en paramètre la méthode enpoint. La méthode endpoint doit également utiliser la méhotde HTTP POST, et donc l'attribut `[HttpPost]`. Exemple :

Les deux templates, pour le formulaire et le résultat :

`hello-world-form.cshtml` :

```
<form action="/hello-world/salute" method="post">
    <input name="FirstName" type="text"/>
    <input name="lastName" type="text"/>
    <input type="submit" value="Say hello !"/>
</form>
```

`hello-world.cshtml` :

```
<h1>Hello, @Model.FirstName @Model.LastName</h1>
```

Les méthodes endpoint :

```
    [HttpGet]
    public IActionResult HelloWorldForm()
    {
        return View("hello-world-form");
    }

    [HttpPost("salute")]
    public IActionResult SayHello(Person person)
    {
        return View("hello-world", person);
    }
```

On peut ensuite constater le résultat en redémarrant l'app (CTRL + C dans la console puis `dotnet run`). Et en allant à https://localhost:7229/hello-world.

Session Http

On peut stocker des données dans la session HTTP. Pour cela il faut d'abord la convigurer en rajoutant dans le `Program.cs` (après la ligne `var builder = WebApplication.CreateBuilder(args);`) :

```
builder.Services.AddDistributedMemoryCache();
builder.Services.AddSession(options =>
{
    options.IdleTimeout = TimeSpan.FromSeconds(10);
    options.Cookie.HttpOnly = true;
    options.Cookie.IsEssential = true;
});
```

Et après `app.UseAuthorization();` :

```
app.UseSession();
```

Pour enregistrer des données dans la session :

```
HttpContext.Session.SetString("key", "value");
```

Pour récupérer des données de la session :

```
string value = HttpContext.Session.GetString("key");
```