

# Cours n°10

## Programme du cours

<i>Activité</i>	<i>Durée</i>
<b>Explication MVC</b>	15m
<b>Liste de courses – Création du site</b>	10m
<b>Liste de courses – Création du formulaire</b>	45h
<b>Liste de courses – Ajout d'un contrôleur</b>	10m
<b>Liste de courses – Création de la page de réception</b>	15m
<b>Liste de courses – Passage des informations</b>	15m
<b>Liste de courses – Évolutions</b>	1h

## Explication du model MVC

MVC est l'acronyme de Modèle-Vue-Contrôleur. C'est un motif de conception (design pattern) couramment utilisé dans le développement d'applications Web. Il est conçu pour séparer la logique de l'application en trois composants interconnectés, ce qui permet de structurer le code de manière plus propre et plus facile à maintenir.

### Voici une description des trois composants :

**Modèle (Model) :** Le modèle représente les données et la logique métier de l'application. Il interagit avec la base de données et effectue les opérations nécessaires pour stocker, récupérer et traiter les données. En C#, les classes du modèle sont généralement définies comme des classes C# avec des propriétés pour représenter les attributs de l'entité.

**Vue (View) :** La vue est responsable de la présentation des données à l'utilisateur. Elle définit la structure, la mise en page et l'apparence de l'interface utilisateur. Dans une application ASP.NET Core MVC, les vues sont généralement créées en utilisant le langage de balisage Razor, qui permet d'intégrer du code C# dans des fichiers HTML.

**Contrôleur (Controller) :** Le contrôleur sert de pont entre le modèle et la vue. Il reçoit les entrées de l'utilisateur à travers la vue, traite ces entrées en utilisant le modèle, puis renvoie les données mises à jour à la vue pour être affichées. En C#, les contrôleurs sont définis comme des classes C# avec des méthodes pour gérer les actions de l'utilisateur.

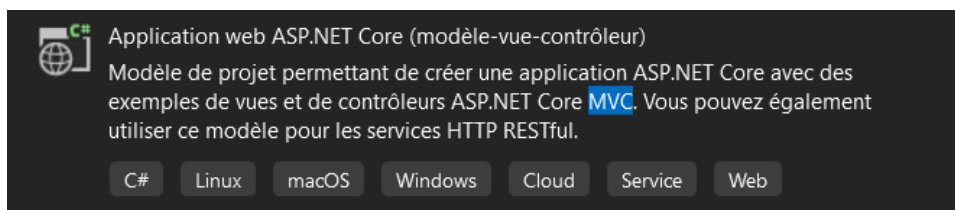
Le motif MVC facilite la répartition des responsabilités et la réutilisation du code dans une application. Il permet également de simplifier les tests, puisque chaque composant peut être testé indépendamment des autres.

# Liste de courses

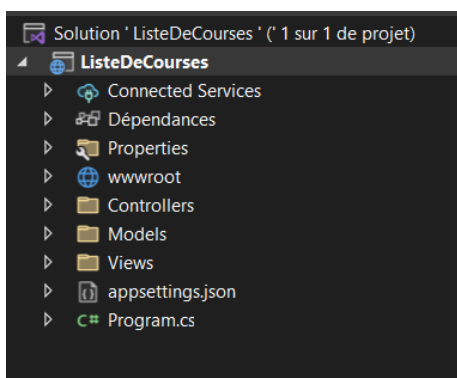
## Création du site

L'objectif est de créer un formulaire permettant de cocher une liste d'article. Les articles sélectionnés sont ensuite affichés dans une autre page.

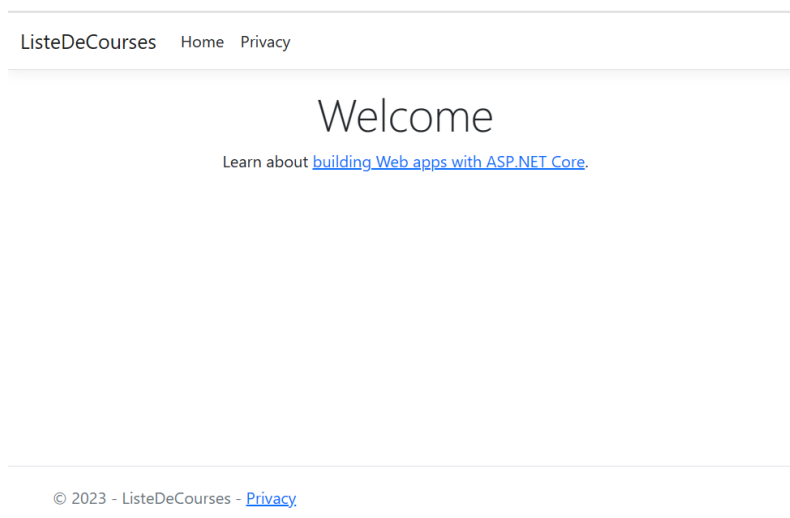
Créer une application ASP.NET Core MVC nommé **ListeDeCourses**



La structure de la solution doit ressembler à ça :



Dans cette structure, nous voyons les dossiers concernant les vues, les modèles et les contrôleurs. Ils seront utilisés comme dans l'explication définie plus haut. Commençons par lancer l'application en l'état. L'application doit ressembler à ceci :



Pour des raisons de simplifications nous allons travailler directement dans le fichier *View/Home/Index.cshtml*. On supprime les lignes 5 à 8 avant de commencer. Lançons le projet, nous voyons que la partie Welcome a disparue.

## Création du formulaire

Nous allons tout d'abord créer une liste d'éléments dans un formulaire. La liste contiendra les articles suivants :

- Lait
- Pain
- Œufs
- Fromage
- Fruits

Chaque article devra avoir une case à cocher sur la gauche de celui-ci. Il y a un bouton envoyé à la fin du formulaire

Pour créer la liste, vous pouvez utiliser des balises HTML `<ol>` et `<li>`. Pour chaque article de la liste, vous devez utiliser une balise `<input>` de type checkbox pour la case à cocher chaque balise `<input>` doit avoir un attribut « **value** » et une balise `<label>` pour le nom de l'article. Un formulaire est entouré d'une balise `<form>`. Ajouter un bouton envoyé à la fin du formulaire.

Une fois terminée la page doit être dans ce format :

ListeDeCourses   Home   Privacy

1. ☐ Lait

2. ☐ Pain

3. ☐ Oeufs

4. ☐ Fromage

5. ☐ Fruits

Envoyer

## Ajout du contrôleur

Maintenant nous allons vouloir récupérer les éléments cochés dans le contrôleur pour cela, nous allons ajouter une nouvelle méthode nommée **ListeCourses** dans le contrôleur avec un paramètre de type `List<string>` nommé `articles`.

```
public List<string> ListeCourses(List<string> articles)
{
    return articles;
}
```

- Pour que ASP.NET Core puisse relier les entrées du formulaire à cette variable, il faut ajouter un attribut « **name="articles"** ».
- Pour que ASP.NET Core puisse relier le formulaire au contrôleur, il faut mettre l'attribut « **asp-action="ListeCourses"** » au niveau de la balise `<form>`.

En lançant l'application maintenant, en envoyant le formulaire avec les cases cochées la page suivante doit s'afficher.

```
[
    "Lait",
    "Pain"
]
```

## Création de la page de réception

Nous allons créer une page de réception qui permettra d'afficher les éléments cochés dans une nouvelle page. Pour cela, dans le dossier *Views/Home* faite clique droite, ajouter, Vue, Vue Razor – Vide. Nommé le fichier **ListeCourses** et supprimer ensuite le contenu du fichier.

Pour que notre contrôleur nous envoie vers la page que nous venons de créer ; Il faut ajouter modifier le contrôleur comme ceci :

```
0 références
public IActionResult ListeCourses(List<string> articles)
{
    return View();
}
```

La fonction **View()** va permettre au contrôleur de rediriger l'utilisateur sur une page qui a le même nom que la méthode (ici **ListeCourses**).

En lançant l'application, on voit que si on envoie le formulaire nous sommes redirigés vers la page que nous venons de créer.

## Passage des informations

Nous voulons maintenant afficher les éléments cochés dans la page de réception. Pour cela nous allons simplement passer la variable **articles** comme argument de la méthode **View()**.

```
public IActionResult ListeCourses(List<string> articles)
{
    return View(articles);
}
```

Maintenant dans la page **ListeCourses**, nos articles sont stockés dans une variable **@Model**. Cette variable est automatiquement créée et elle contient l'argument passé dans la méthode **View()**. Nous allons dire à la vue que la variable passée en tant que **@Model** est de type *List<string>*. Pour ça on ajoute cette ligne au début de notre fichier *ListeCourses.cshtml* :

```
@model List<string>
```

Pour exécuter du code C# dans la partie HTML, il faut ajouter le symbole **@** avant l'instruction. Ici, nous allons simplement itérer sur la variable **Model**.

```
@model List<string>

@foreach (var article in Model)
{
    <li>@article</li>
}
```

## Évolutions

1. Ajouter un champ de type texte permettant d'ajouter un autre article.

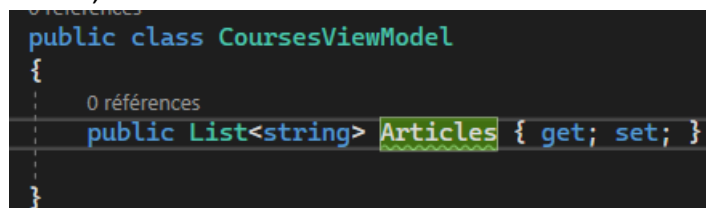


A screenshot of a web form. It contains a list of six items, each with a radio button and a text label: '1. ☐ Lait', '2. ☐ Pain', '3. ☐ Oeufs', '4. ☐ Fromage', '5. ☐ Fruits', and '6. Autre'. To the right of the 'Autre' label is a text input field. Below the list is a button labeled 'Envoyer'.

2. C'est une mauvaise pratique de directement passé la liste de string directement à la vue. Nous devrions passer par une ViewModel.

Un ViewModel est une classe qui représente les données à afficher dans une vue spécifique. Il contient toutes les données nécessaires pour rendre la vue. Les ViewModels sont spécifiques à chaque vue et peuvent être utilisés pour encapsuler les données nécessaires à la vue. Ils peuvent être utilisés pour rassembler des données de plusieurs modèles et pour les formater ou les transformer pour les rendre plus facilement utilisables dans une vue.

Dans le dossier *Model*, créer une classe comme ci-dessous :



```
public class CoursesViewModel
{
    0 références
    public List<string> Articles { get; set; }
}
```

Adapter le contrôleur et la page *ListeCourses.cshtml* pour utiliser ce ViewModel

3. Le projet ASP.NET Core utilise la librairie visuelle **Bootstrap**. Essayer d'utiliser la librairie **Bootstrap** pour rendre votre formulaire plus sympathique.

documentation: [Checks and radios · Bootstrap v5.3 \(getbootstrap.com\)](https://getbootstrap.com/docs/5.3/forms/checks-and-radios/)