

ASP.NET MVC

Introduction et vue d'ensemble

Objectifs du cours

Dans ce cours, vous allez

- **Construire et déployer des applications sécurisées et évolutives avec ASP.NET MVC**
- **Générer du HTML dynamique avec des vues Razor, des vues partielles et des aides de vue**
- **Créer un modèle faiblement couplé avec Entity Framework et l'injection de dépendances**
- **Développer des contrôleurs d'application et des filtres d'action**
- **Structurer les applications avec des zones et les URL avec le routage**
- **Créer et valider des formulaires avec des aides HTML standard et personnalisées**
- **Concevoir des clients riches et des IU mobiles avec Ajax, jQuery et jQueryUI**
- **Sécuriser les applications avec l'authentification et des fournisseurs tiers**
- **Créer des services RESTful avec Web API**

API = application programming interface
HTML = hypertext markup language

REST = representational
state transfer

UI = user interface
URL = uniform resource locator

Contenu du cours

Introduction et vue d'ensemble

- I. Présentation de ASP.NET MVC**
- II. Razor et les vues**
- III. Le modèle**
- IV. Le contrôleur**
- V. Formulaires et saisie utilisateur**
- VI. Fonctionnalités côté client**
- VII. Créer des applications d'entreprise**
- VIII. Déploiement des applications**

Prérequis

- **Ce cours est un cours avancé qui suppose une connaissance de**
 - Développement d'applications .NET en C# ou Visual Basic
 - Principes du développement orienté objet
 - HTML de base
 - Comprendre les éléments tels que
 - `<a>`
 - `<div>`
 - `<table>`
 - ``
- **Avertissez votre formateur si vous n'avez pas ces prérequis**

I. (5 à 31)

Présentation de ASP.NET MVC

Objectifs du chapitre

Dans ce chapitre, nous allons

- Explorer le modèle de programmation MVC
- Créer une application MVC simple
- Examiner les interactions entre le contrôleur, la vue et le modèle
- Présenter les principaux verbes HTTP
- Comparer les formulaires Web ASP.NET et MVC

HTTP = Hypertext Transfer Protocol

MVC = Model View Controller

Présentation de ASP.NET MVC

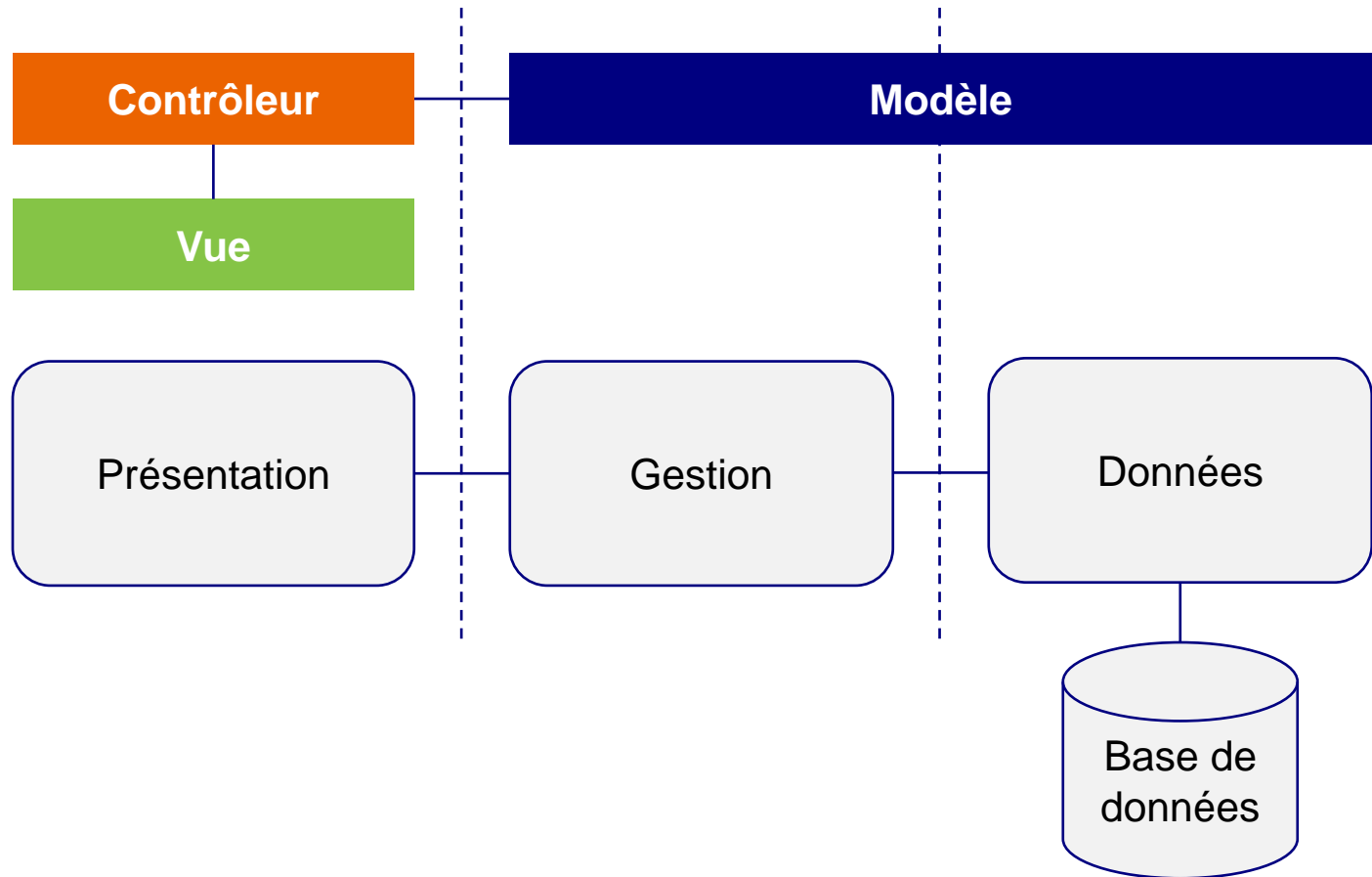
- ➡ **Le design pattern MVC**
 - **Les composants et projets de MVC**
 - **Le protocole HTTP**
 - **Les formulaires Web et MVC**

MVC et les applications multiniveau

- **Les applications sont souvent structurées en trois niveaux ou couches**
 - Présentation : gestion de l'interface utilisateur
 - Gestion : le cœur de l'application comprenant la logique métier
 - Données : stockage permanent, généralement dans une base de données relationnelle
- **MVC est un design pattern appliqué à des applications Web**
 - Modèle : représente les niveaux gestion et données
 - Vue : affiche les données et gère la saisie de l'utilisateur
 - Contrôleur : logique qui traite le flux des requêtes et réponses entre le navigateur et le serveur Web
- **ASP.NET MVC est l'implémentation du design pattern MVC par Microsoft**
 - Comprend des vues et des contrôleurs pour le niveau présentation
 - Le modèle est le reste de l'application et ne fait pas partie de ASP.NET MVC
 - Il peut être utilisé par d'autres couches de présentation telles que WPF ou des applications mobiles

WPF = Windows Presentation Foundation

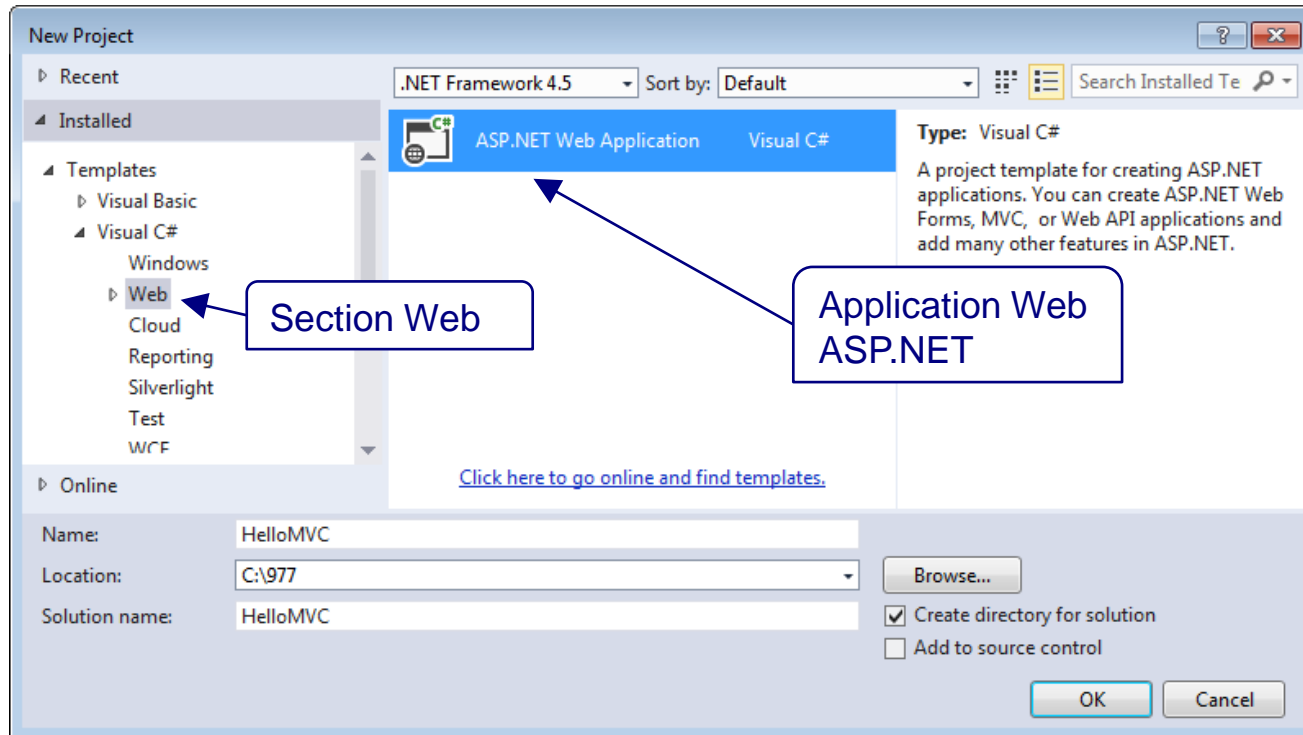
MVC et les applications multiniveau



Créer un projet ASP.NET MVC

➤ Dans la section Templates, cliquer sur Visual C#, puis sur Web

- Sélectionner ASP.NET Web Application
- Choisir le type d'application Web sur l'écran suivant



Modèles de projet ASP.NET

Projet ASP.NET vide

Application ASP.NET MVC avec quelques vues

Sélection des frameworks à configurer

Ajout de tests unitaires

New ASP.NET Project - ListPlus

Select a template:

Empty Web Forms MVC Web API

Single Page Application Facebook

Add folders and core references for:

☐ Web Forms ☒ MVC ☐ Web API

☐ Add unit tests

Test project name: ListPlus.Tests

A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

[Learn more](#)

Change Authentication

Authentication: **Individual User Accounts**

OK Cancel

Exercice 1.1 : Créer une application MVC simple

Présentation de ASP.NET MVC

- Le design pattern MVC
- ➡ **Les composants et projets de MVC**
- Le protocole HTTP
- Les formulaires Web et MVC

Le contrôleur est le chef d'orchestre

➤ Le client fait une requête en

- Cliquant sur un lien
- Postant un formulaire
- Faisant un appel Ajax

Poster et Ajax sont présentés
dans d'autres chapitres

➤ La requête est toujours routée vers une méthode d'une classe contrôleur

- Appelée *méthode d'action*

➤ Le contrôleur appelle le modèle

- Le modèle implémente la logique métier
- Le contrôleur sélectionne la vue selon ce que renvoie le modèle

➤ La vue prépare la sortie

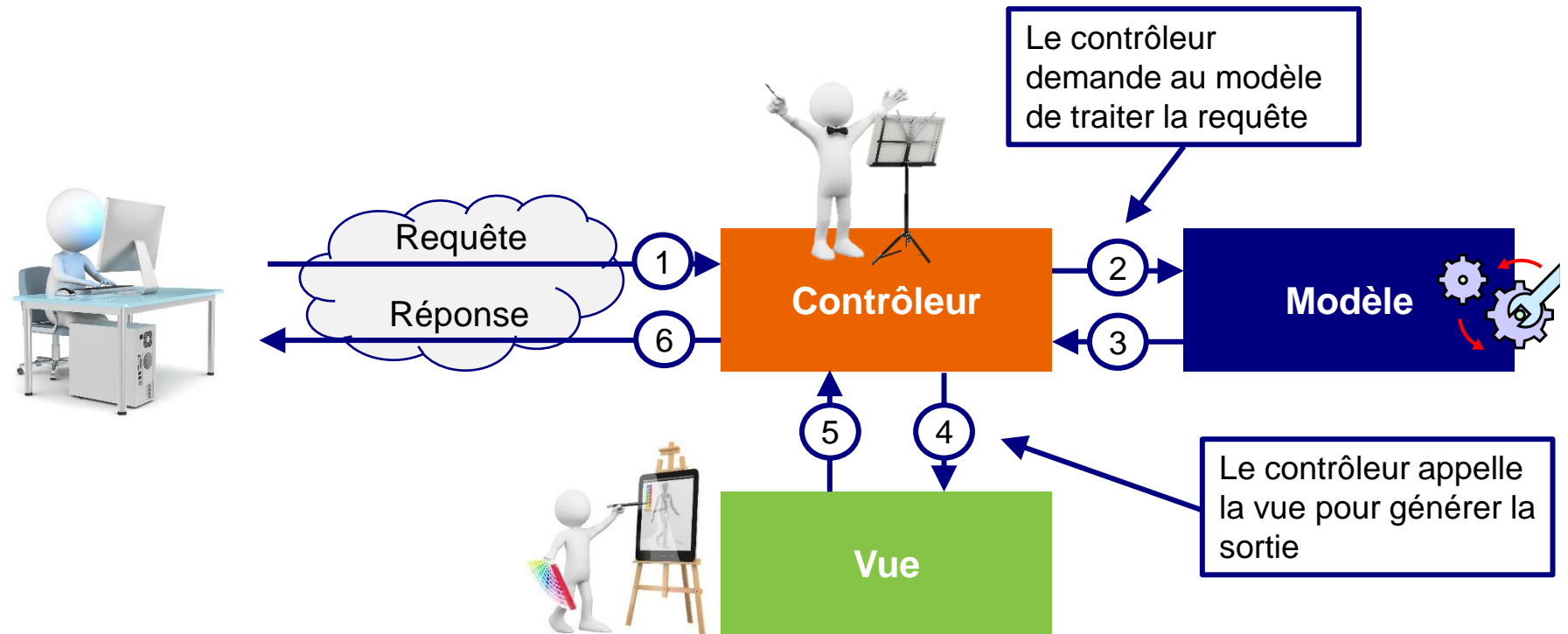
- En mélangeant du HTML statique et dynamique

➤ Le contrôleur renvoie les données générées au client

- Il peut aussi renvoyer d'autres types de contenus, tels que des images ou du code JSON

JSON = JavaScript Object Notation

Demande et réponse MVC



Contrôleur et méthodes d'action

➤ Chaque contrôleur est une classe

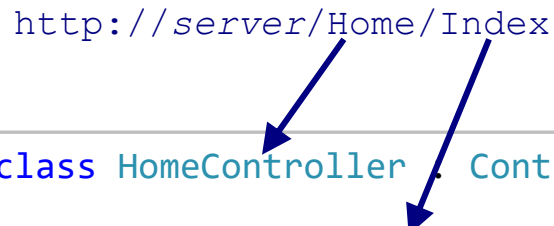
- Dérive de `System.Web.Mvc.Controller`
- Les méthodes d'action renvoient des objets `ActionResult`

➤ L'URL comprend le nom du contrôleur et la méthode d'action


- Le nom du contrôleur sans le suffixe `Controller`

`http://server/Home/Index`

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



Par défaut, le nom de la vue renvoyée est le même que celui de la méthode d'action



Vues et moteurs de rendu

➤ Les vues contiennent du HTML

- Envoyé tel quel au navigateur
- Peuvent aussi inclure du JavaScript qui est exécuté sur le client, dans le navigateur
- Comprennent généralement aussi du code exécuté sur le serveur pour générer du HTML

➤ Un moteur de rendu transforme le code côté serveur en HTML

- Le moteur de rendu initial était ASPX, dont la syntaxe est semblable à celle des formulaires Web
 - Le code se trouvait dans des blocs `<% . . . %>`
- MVC 3 a ajouté le moteur de rendu Razor
 - Razor est utilisé dans tous les exercices
- Il existe des alternatives non Microsoft

ASPX = fichier Active Server Page Extended

Les projets MVC dans Visual Studio

➤ ASP.NET MVC utilise des conventions de nommage

- La plupart peuvent être changées ou adaptées
- Il est plus facile de garder les conventions par défaut

➤ Le dossier **Controllers** contient tous les contrôleurs

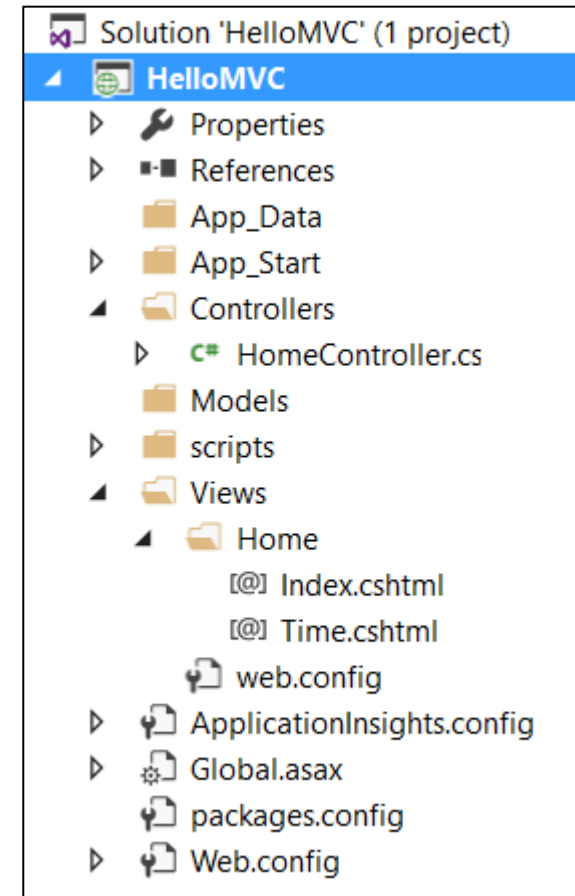
- Les noms des contrôleurs ont le suffixe `Controller`

➤ Le dossier **Models** n'est qu'un emplacement suggéré

- On peut y placer les classes métier
- Qui sont souvent dans un projet bibliothèque de classes séparé

➤ Le dossier **Views** a un sous-dossier pour chaque contrôleur

- Les vues sont dans un dossier nommé d'après le contrôleur



Présentation de ASP.NET MVC

- Le design pattern MVC
- Les composants et projets de MVC
- ➡ **Le protocole HTTP**
- Les formulaires Web et MVC

Le protocole HTTP

➤ Protocole simple de requête – réponse

- Le client fait une requête
- Le serveur Web traite la requête et envoie la réponse au client

➤ Le client est généralement un navigateur Web

- Microsoft Edge
- Internet Explorer
- Chrome
- Firefox
- Safari ou autres

➤ Le serveur de Microsoft est IIS (Internet Information Services)

- Sur les systèmes serveur Windows
- Également sur les postes Windows (7, 8, 10...), avec un nombre de clients limité
- Il existe des alternatives open source
 - Telles que Mono (www.mono-project.com)

Les verbes HTTP

➤ Le protocole HTTP est basé sur du texte

- Les requêtes et les réponses comprennent un en-tête et un corps
- L'en-tête ne comprend que du texte
- Le corps est généralement du texte, mais peut avoir d'autres formats

➤ Un en-tête de requête commence par un verbe

- GET et POST demandent des données au serveur
 - GET n'a pas de corps : les données envoyées sont dans l'URL
 - POST est souvent utilisé avec les formulaires : les données sont dans le corps de la requête

➤ Le corps de la réponse est généralement du HTML

- Peut aussi être des données XML ou JSON
- Ou une image, une vidéo, un son...

➤ Les navigateurs ont un outil pour les développeurs qui peut afficher du HTTP

- Appuyer sur <F12>
- Possibilité également d'utiliser Fiddler, un outil gratuit pour tracer les requêtes et les réponses HTTP

À vous 1a : Explorer le code HTTP

À vous

1. **Démarrez Fiddler à l'aide de son raccourci dans la barre des tâches**
2. **Ouvrez le point de départ Do Now 1a**
 - Il s'agit de la solution terminée HelloMVC du dernier exercice
3. **Lancez votre application en appuyant sur la touche <F5>**
4. **Double-cliquez sur la dernière requête HTTP dans le volet de gauche de la fenêtre de Fiddler**
5. **Cliquez sur l'onglet Raw dans les deux volets. Dans le volet du bas, cliquez sur la barre jaune pour décoder la réponse**
6. **Explorez l'en-tête de la requête (elle n'a pas de corps) et l'en-tête et le corps de la réponse**
7. **S'il vous reste du temps, appuyez sur <F12> dans le navigateur Edge ou sélectionnez « F12 Developer Tools » dans le menu**
8. **Cliquez sur l'onglet Network dans la fenêtre des outils de développement**
9. **Appuyez sur <F5> pour actualiser la page**
10. **Cliquez sur la requête HTTP**
 - Les en-têtes HTTP s'affichent sur la droite

Explorer le code HTTP avec Fiddler

À vous

The screenshot displays the Fiddler Web Debugger interface. The top menu bar includes File, Edit, Rules, Tools, View, Help, GET /book, and GeoEdge. The toolbar contains various icons for WinConfig, Replay, Go, Stream, Decode, Keep: All sessions, Any Process, Find, Save, Browse, Clear Cache, TextWizard, and Tearoff. The main window is divided into several panes. On the left, a list of sessions shows a single entry with Result 200, Protocol HTTP, Host localhost:49914, and URL /. The right pane is split into two sections. The top section, labeled 'Inspectors', shows the raw HTTP request: GET http://localhost:49914/ HTTP/1.1, with headers including Accept, Accept-Language, User-Agent, Accept-Encoding, Host, Connection, and Pragma. The bottom section, labeled 'Raw', shows the HTTP response: HTTP/1.1 200 OK, with headers including Cache-Control, Content-Type, Vary, Server, X-AspNetMvc-Version, X-AspNet-Version, X-SourceFiles, X-Powered-By, Date, and Content-Length. The response body is an HTML document with a title 'Index' and a message 'Hello ASP.NET MVC World!'. The status bar at the bottom indicates 'Capturing' is active, 'All Processes' are being monitored, and the current session is 1/1 for http://localhost:49914/.

Fiddler Web Debugger

File Edit Rules Tools View Help GET /book GeoEdge

WinConfig Replay Go Stream Decode Keep: All sessions Any Process Find Save Browse Clear Cache TextWizard Tearoff

#	Result	Protocol	Host	URL
2	200	HTTP	localhost:49914	/

Statistics Inspectors AutoResponder Composer Log Filters Timeline

Headers TextView WebForms HexView Auth Cookies Raw JSON XML

```
GET http://localhost:49914/ HTTP/1.1
Accept: text/html, application/xhtml+xml, image/jxr, */*
Accept-Language: en-US
User-Agent: Mozilla/5.0 (Windows NT 10.0; win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0
Accept-Encoding: gzip, deflate
Host: localhost:49914
Connection: Keep-Alive
Pragma: no-cache
```

Find... (press Ctrl+Enter to highlight all) View in Notepad

Get SyntaxView Transformer Headers TextView ImageView HexView WebView Auth Caching Cookies Raw

JSON XML

```
HTTP/1.1 200 OK
Cache-Control: private
Content-Type: text/html; charset=utf-8
Vary: Accept-Encoding
Server: Microsoft-IIS/10.0
X-AspNetMvc-Version: 5.2
X-AspNet-Version: 4.0.30319
X-SourceFiles: =?UTF-8?B?QzpcOTc3XERvIESvd3NCRG8gTm93IDFhLVNOYXJ0aW5nIFBvaW50XEH1bGxvTVZDXEH1bGxvTVZD?=?
X-Powered-By: ASP.NET
Date: Sat, 04 Jun 2016 14:08:34 GMT
Content-Length: 299
```

```
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>Index</title>
</head>
<body>
  <div>
    Hello ASP.NET MVC World!<br/>
    <a href="/Home/Time">Display current time</a>
  </div>
</body>
</html>
```

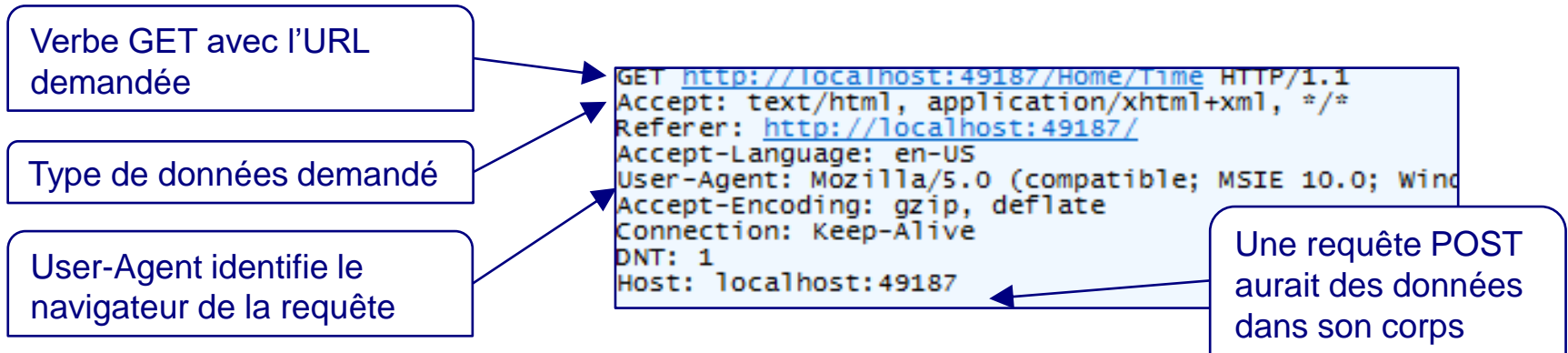
Find... (press Ctrl+Enter to highlight all) View in Notepad

[QuickExec] ALT+Q > type HELP to learn more

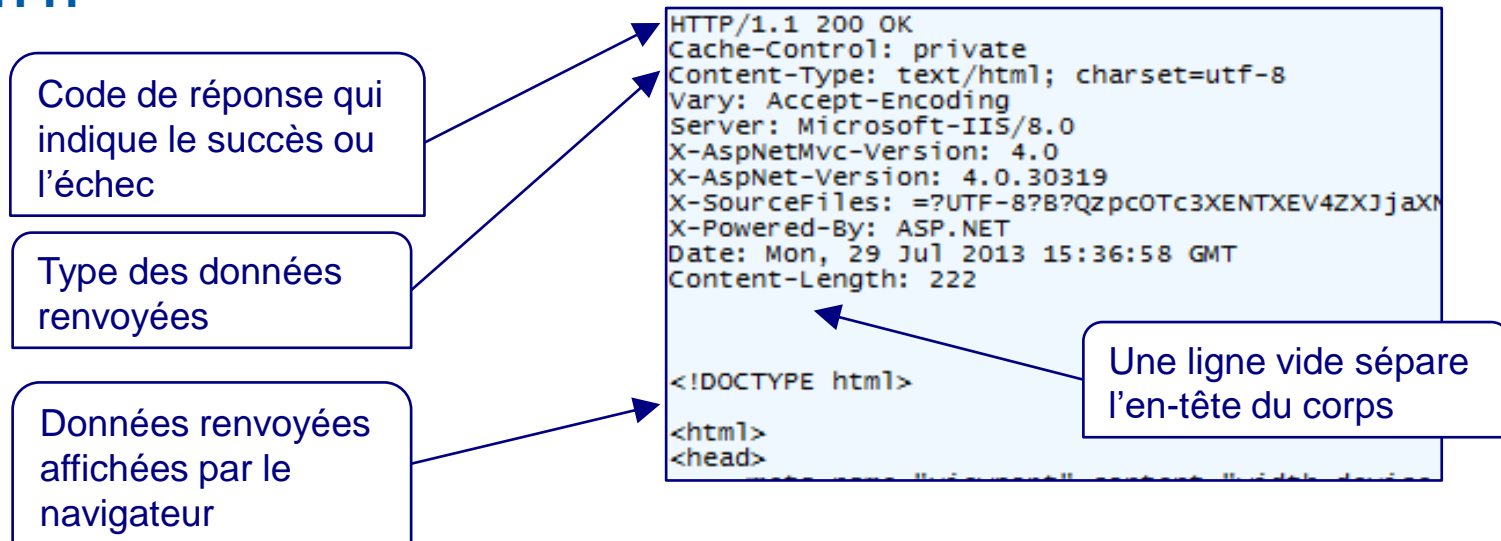
Capturing All Processes 1 / 1 http://localhost:49914/

Requêtes et réponses HTTP

➤ Requête HTTP



➤ Réponse HTTP



Présentation de ASP.NET MVC

- Le design pattern MVC
- Les composants et projets de MVC
- Le protocole HTTP
- ➔ **Les formulaires Web et MVC**

Technologies Microsoft pour les applications Web

- **Deux technologies ASP.NET pour développer des applications Web**
 - Formulaires Web ASP.NET
 - ASP.NET MVC
- **Les formulaires Web ASP.NET existent depuis la première version de .NET**
 - Améliorés dans chaque version de .NET
 - Chaque page HTML (`.aspx`) a un fichier de code associé
 - Avec des gestionnaires d'événements écrits en C# ou Visual Basic
- **ASP.NET MVC est diffusé parallèlement à Visual Studio**
 - Première version en 2009
 - MVC est open source
 - Le code source peut être téléchargé depuis www.codeplex.com

Formulaires Web ASP.NET et MVC

- **Les formulaires Web ont une architecture basée sur des événements**
 - Une action de l'utilisateur sur une page génère un événement sur le serveur
 - Le code associé intercepte et traite les événements
 - Semblable aux applications formulaires Windows ou WPF
 - Un concepteur visuel permet de dessiner les pages ASPX
- **Il n'y a pas d'événements dans ASP.NET MVC**
 - Une action de l'utilisateur est routée vers une méthode d'action de contrôleur
 - Une table de routage définit les règles
- **Les plus et les moins de ASP.NET MVC**
 - Les plus
 - Mieux structuré
 - Contrôle total du HTML généré
 - Intégration avec une infrastructure de TDD plus facile
 - Le moins
 - Pas de concepteur visuel

TDD : test-driven development – développement piloté par les tests

Versions d'ASP.NET et de MVC

- **ASP.NET est une plateforme de développement et d'exécution de services et d'applications Web**
 - Mise à disposition en 2002 avec .NET Framework
 - La dernière version est ASP.NET 4.6, elle fait partie de .NET Framework 4
 - ASP.NET a été entièrement révisé et s'appelle désormais ASP.NET Core (ASP.NET 5)
- **MVC s'appuie sur ASP.NET**
 - Le moteur de vue Razor est apparu dans la version 3
 - Dernière version : MVC 5
 - ASP.NET Core renferme une nouvelle version de MVC (qui à l'origine s'appelait MVC 6)
- **Avec Visual Studio 2015, les développeurs peuvent créer**
 - Des applications ASP.NET 4 – MVC 5
 - Ou des applications ASP.NET Core MVC
- **Cette formation est axée sur les applications ASP.NET 4 – MVC 5**
 - Compatibles avec les anciennes versions de Visual Studio
 - ASP.NET Core MVC fonctionne avec Visual Studio 2015

- **La majorité des exercices du cours sont articulés autour d'une étude de cas**
 - Un système de gestion de listes (to-do lists)
 - Les listes peuvent se classer dans différentes catégories
 - Chaque liste contient des éléments de liste (to-do items)
- **Démonstration : l'application Todo**

Résumé du chapitre

Dans ce chapitre, nous avons

- **Exploré le modèle de programmation MVC**
- **Créé une application MVC simple**
- **Examiné les interactions entre le contrôleur, la vue et le modèle**
- **Présenté les principaux verbes HTTP**
- **Comparé les formulaires Web ASP.NET et MVC**

Questions de révision

À quel niveau appartiennent la vue et le contrôleur ?

Comment s'appelle le moteur de vue utilisé par les applications MVC ?

Quel est le rôle du contrôleur dans une application MVC ?

Comment s'appellent les méthodes des contrôleurs ?

Citez quelques verbes HTTP

Quels sont les avantages de ASP.NET MVC sur les formulaires Web ?

II. (32 à 68)

Razor et les vues

Objectifs du chapitre

Dans ce chapitre, nous allons

- Créer des vues pour afficher des données
- Générer du HTML de manière dynamique avec du code C# et Razor
- Passer des données du contrôleur à la vue avec le `ViewBag` et le modèle
- Structurer les vues avec des vues dispositions et des sections
- Créer un contenu réutilisable avec des vues partielles et des aides de vues

Razor et les vues

- ➡ **Vues, ViewBag et modèles**
 - **La syntaxe Razor**
 - **Structurer les vues avec des vues de disposition**
 - **Vues partielles et aides de vues**

Les vues

➤ Fichiers HTML

- L'extension est `cshtml`
- Combinaison d'éléments HTML et de code C#, avec la syntaxe Razor

➤ Peuvent être des vues complètes ou partielles

- Une vue complète est une page HTML affichée dans le navigateur
- Une vue partielle peut être incluse dans une vue complète ou une autre vue partielle

➤ Peuvent être partagées entre plusieurs contrôleurs

- Placées dans le dossier `Views\Shared`

➤ Les pages de disposition (layout) contiennent des éléments communs à plusieurs vues

- En-tête, pied de page, menu...

Le contrôleur et la vue

➤ Une méthode d'action d'un contrôleur sélectionne la vue à renvoyer

- La méthode `View` génère la vue
- Le nom de la vue est celui de la méthode d'action par défaut
 - On peut le définir explicitement en tant que paramètre de la méthode `View`

```
public ActionResult Index()  
{  
    return View();  
}
```

C#

- Le code suivant a le même résultat

```
public ActionResult Index()  
{  
    return View("Index");  
}
```

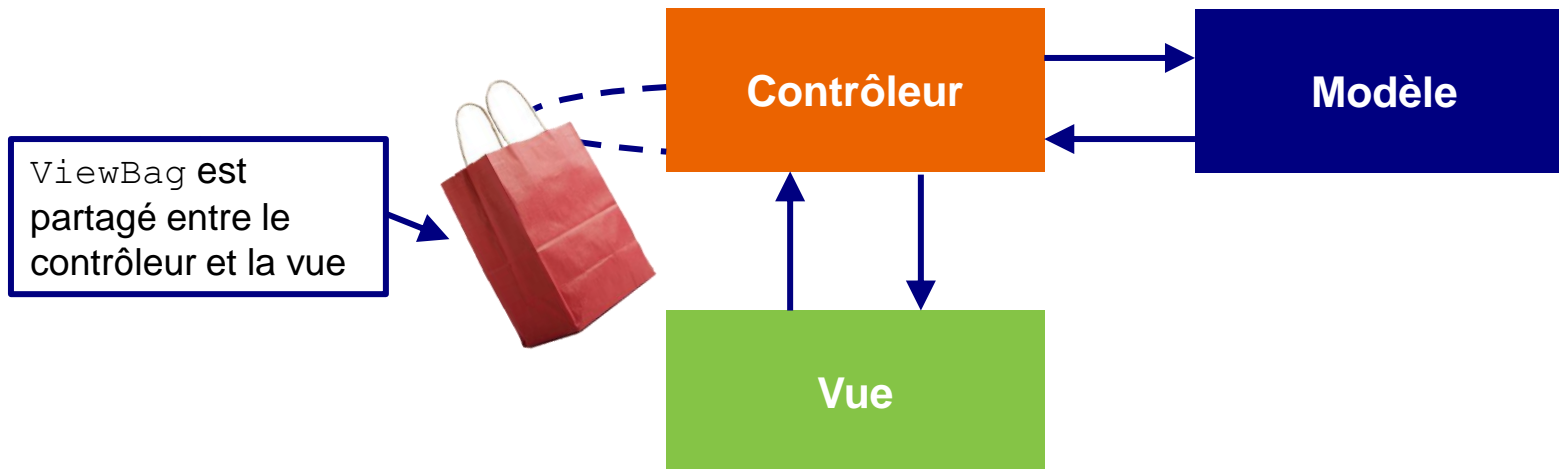
C#

➤ La vue se trouve dans un dossier dont le nom est celui du contrôleur

- `Views\Home` pour le contrôleur `Home`

Passer des données du contrôleur à la vue

- **On peut passer des données de deux façons**
 - Via le `ViewBag`
 - En tant que paramètre de la méthode `View`
- **Les données proviennent généralement du modèle**
 - Utilisées dans la vue pour générer du HTML



Le ViewBag

➤ Dictionnaire de données dynamique

- Les propriétés sont créées à la volée
- Utilise un objet dynamique `Expando`
- Créer ou modifier une propriété appelée `Hello` et lui donner `World` comme valeur :

```
ViewBag>Hello = "World"
```

C#

➤ Code équivalent avec le dictionnaire `ViewData`

- Même résultat qu'avec le code précédent :

```
ViewData["Hello"] = "World";
```

C#

➤ Utilisation du `ViewBag`

- Dans le contrôleur :

```
ViewBag>Hello = "World";  
return View("Index");
```

C#

- Dans la vue :

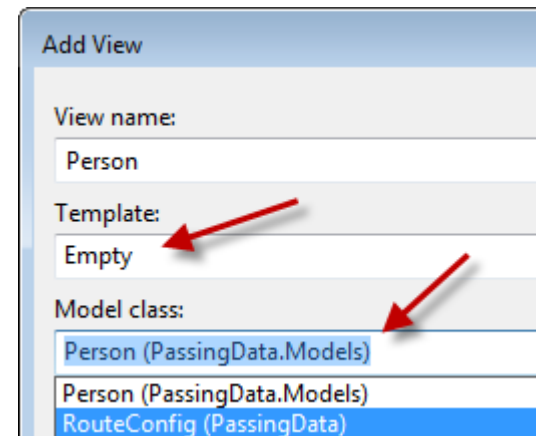
```
Hello @ViewBag>Hello
```

Razor

Affiche Hello World

Le modèle

- **Classe conçue pour partager des données entre contrôleur et vue**
 - Le contrôleur crée un objet modèle, ou l'obtient d'une couche service
 - Il le passe à la vue en tant que second paramètre de la méthode `View`
- **Le modèle est disponible dans la vue à travers sa propriété `Model`**
 - Propriété `dynamic` par défaut
 - Pas typée : IntelliSense n'est d'aucune aide lors de l'édition du code
- **La vue peut être fortement typée**
 - Avec une directive `@model` en haut du fichier
 - La propriété `Model` a le type spécifié dans la directive
- **Visual Studio peut aider à créer une vue typée**
 - Sélectionner « Create a strongly-typed view » lors de l'ajout d'une vue
 - Sélectionner la classe du modèle dans la liste ou taper son nom



Utiliser le modèle dans le contrôleur

- **Le modèle est souvent une classe n'ayant que des propriétés**

```
public class Person
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
}
```

C#

- **La méthode d'action obtient le modèle et le passe à la vue**

- Généralement à partir d'une couche de service

```
var person = new Person { Id = 1, Name = "Andrew", Age = 10 };
return View(person);
```

C#

- **Le modèle est généralement obtenu suite à l'appel d'un couche de service**

- Ou généré par des classes de la couche de service

➤ La vue utilise les propriétés du modèle

```
Name: @Model.Name<br />
Age: @Model.Age
```

➤ Ajouter une directive en haut de la vue pour typer fortement le modèle

```
@model ViewDemo.Models.Person
```

➤ Démonstration : le passage de données entre le contrôleur et la vue

- Ouvrez la solution à Demo\Chapter 2\PassingData

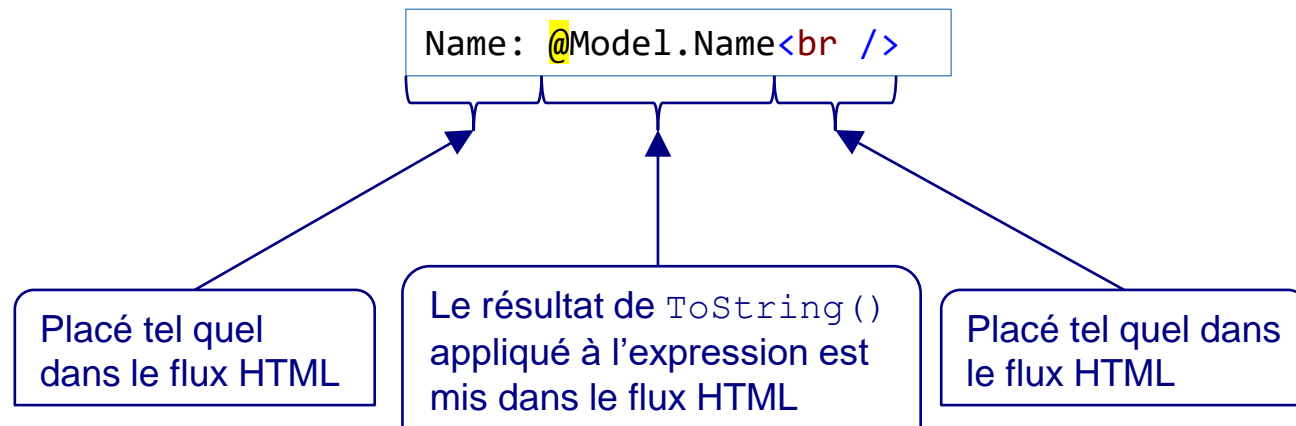
Razor et les vues

- Vues, ViewBag et modèles
- ➡ **La syntaxe Razor**
- Structurer les vues avec des vues de disposition
- Vues partielles et aides de vues

Éléments de syntaxe Razor

➤ Le code est introduit par le caractère @

- Le texte qui suit est interprété en tant que code C# par le moteur Razor
- Il doit renvoyer une valeur qui sera placée dans le flux de sortie
- Un changement de ligne, un caractère < et d'autres caractères spéciaux indiquent au moteur Razor de repasser en mode texte



- Sortie générée si la valeur de la propriété `Name` est Andrew :

```
Name: Andrew<br />
```

Expressions Razor

➤ Les caractères spéciaux peuvent générer des résultats non voulus

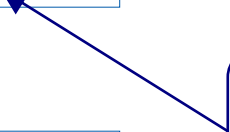
- Par exemple :

Your age next year: @Model.Age+1

- La sortie est :

Your age next year: 10+1

Razor « pense »
que l'expression
est terminée



➤ On peut mettre une expression Razor entre parenthèses

- La syntaxe suivante donne le résultat attendu :

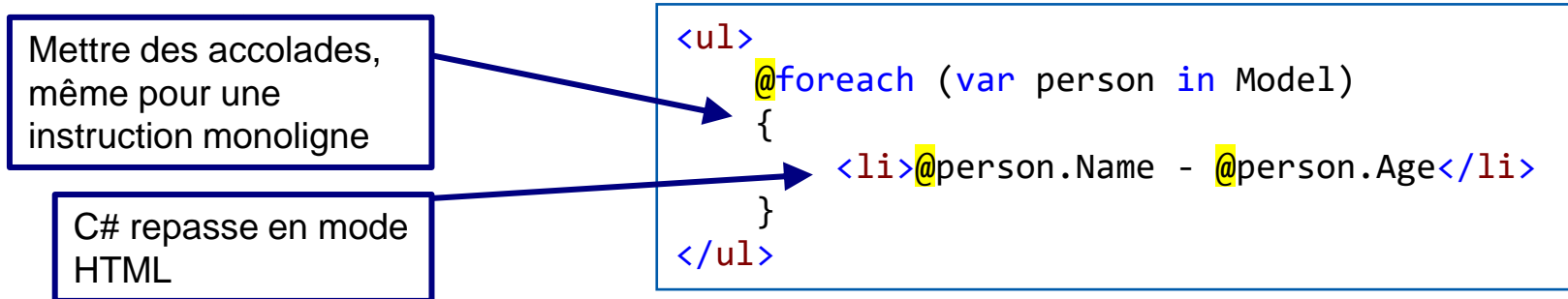
Your age next year: @(Model.Age+1)

- Sortie :

Your age next year: 11

Boucles et tests

- **On peut utiliser des instructions C# avec le caractère @**
 - Boucles `for`, `foreach`, `while`
 - Instructions `if`
 - Expressions LINQ
- **Razor « devine » en général où commence et se termine le code C#**
 - Afficher une liste de personnes, si le type du modèle est `List<Person>` :



LINQ = Language Integrated Query

Blocs de code

➤ On peut écrire des expressions de code multiligne avec des blocs de code

- @ { . . . }
- Code semblable au précédent, avec une variable `persons` pour le modèle :

```
@{  
    var persons = Model;  
}  
<ul>  
    @foreach (var person in persons)  
    {  
        <li>@person.Name - @person.Age</li>  
    }  
</ul>
```

Autres éléments de syntaxe Razor

- **Les commentaires sont dans des blocs `@* ... *`**
 - Ils peuvent inclure du code Razor ou du HTML et être multi line
 - Les blocs de commentaires ne sont pas envoyés dans la sortie
- **Pour inclure du HTML dans le flux de sortie depuis une instruction Razor**
 - Utiliser un élément HTML tel que `` ou ``

```
@foreach (var person in Model) {  
    <span>Person name: @person.Name<br /></span>  
}
```

- Ou utiliser `@:`

```
@foreach (var person in Model) {  
    @:Person name: @person.Name<br />  
}
```

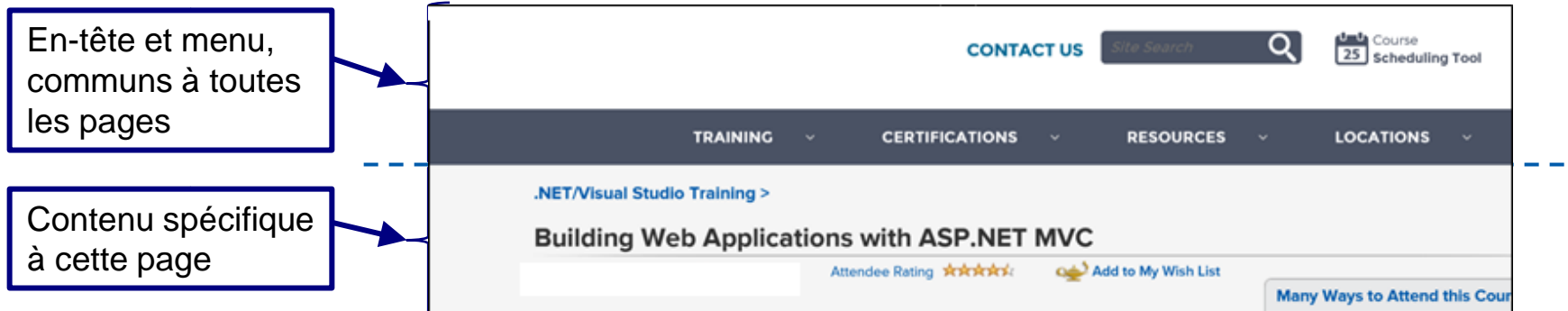
Exercice 2.1 : Créer le projet de l'étude de cas

Razor et les vues

- Vues, ViewBag et modèles
- La syntaxe Razor
- ➡ **Structurer les vues avec des vues de disposition**
- Vues partielles et aides de vues

Pages de disposition

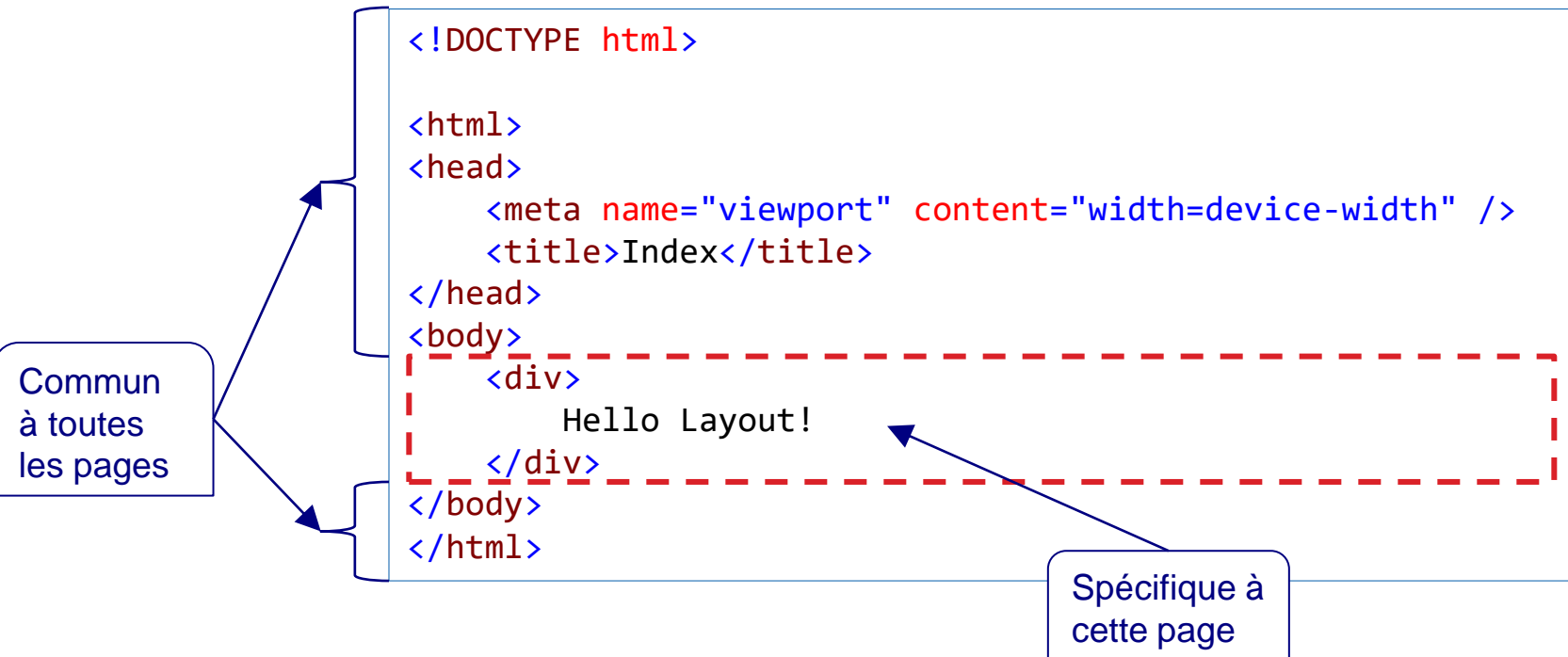
- **Les pages ont une disposition commune dans beaucoup d'applications Web**
 - En-tête
 - Menu
 - Pied de page
- **Différentes zones de l'application peuvent avoir différentes dispositions**
 - Design ou contenu différents pour les zones publiques et privées
- **Les pages de disposition ont du HTML commun à un ensemble de pages**
 - Comme les pages maître des formulaires Web



Disposition = *Layout* en anglais

Balises HTML commun

- **Le balisage commun est placé dans une vue de disposition**
 - Le balisage spécifique reste dans la vue

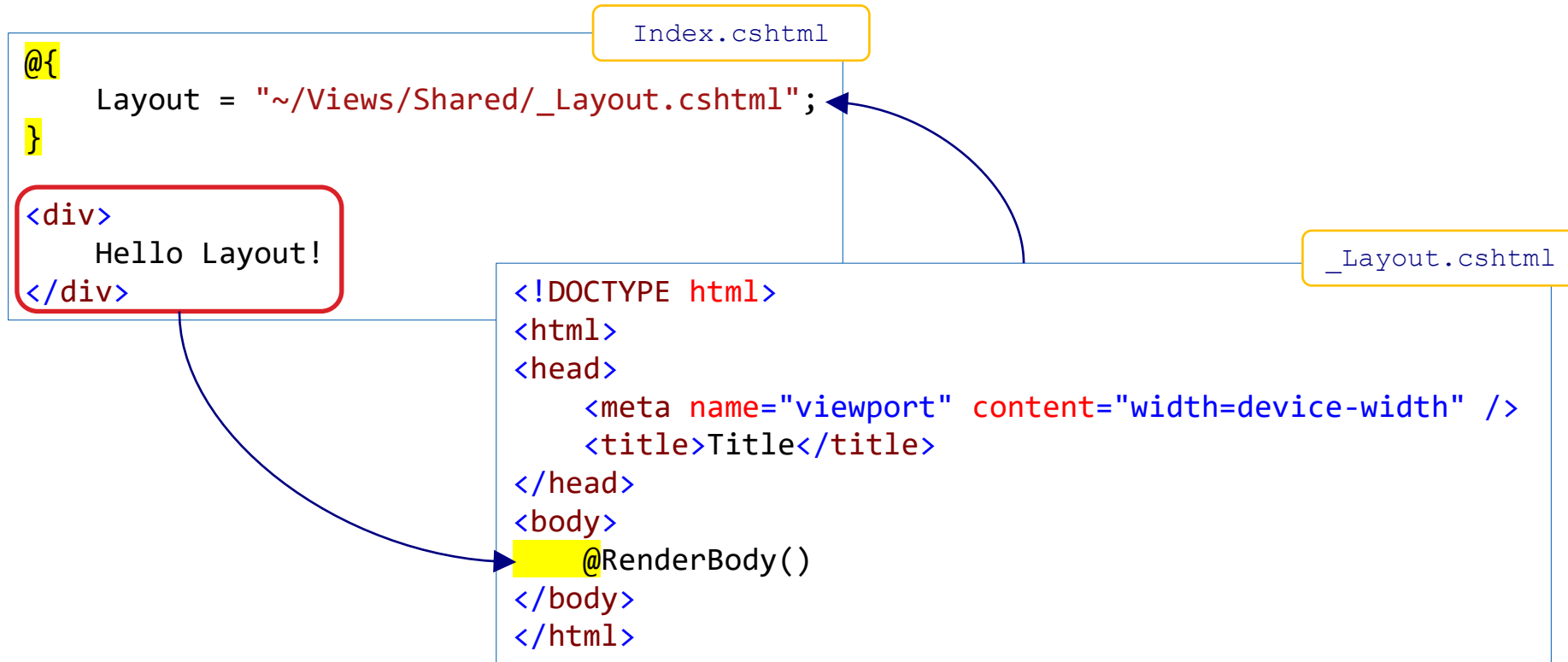


- **Le contenu spécifique à la page est remplacé par un emplacement réservé dans la vue de disposition**

Disposition et vue

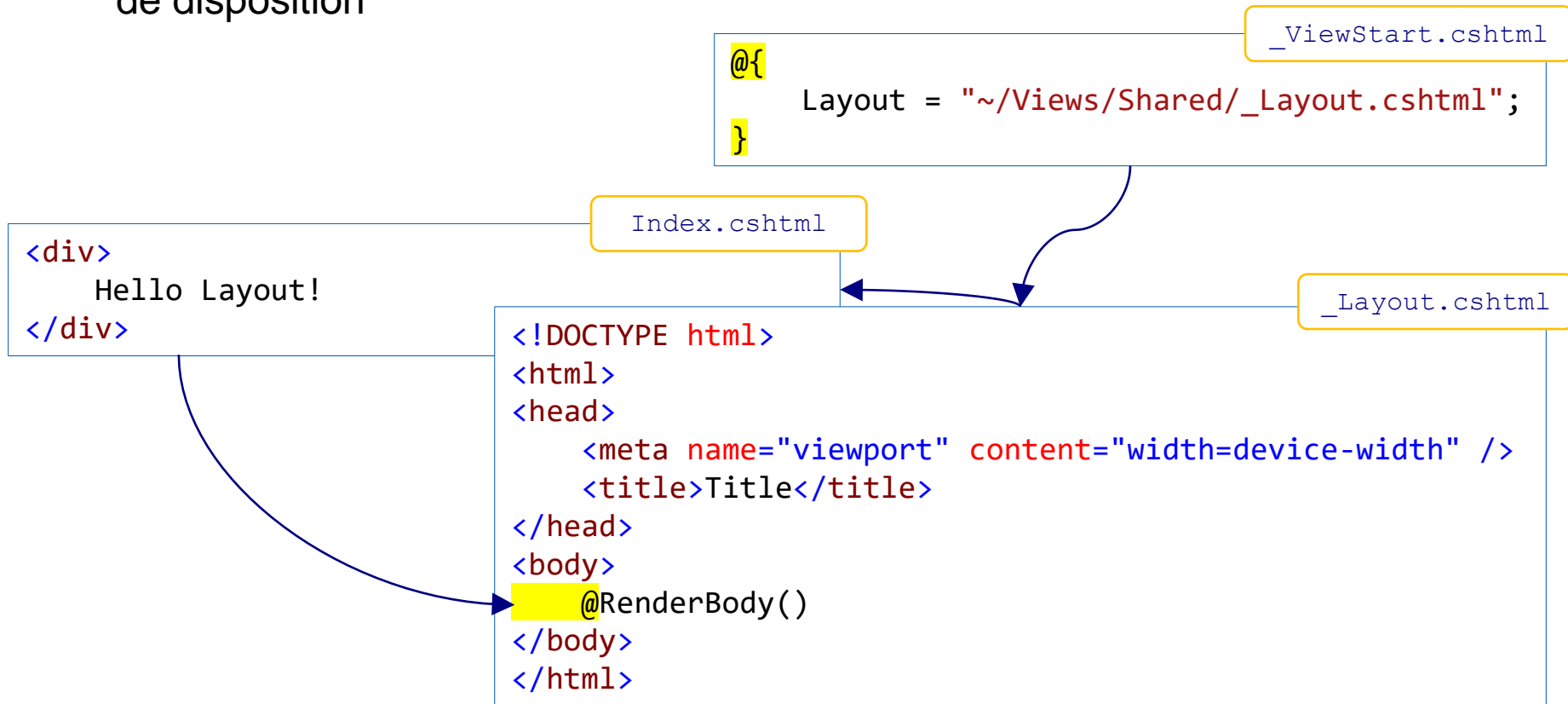
➤ Le client demande une vue spécifique

- Si la vue définit la propriété `layout`, la vue de disposition référencée est générée
- Le contenu de la vue est généré à l'emplacement réservé `@RenderBody`



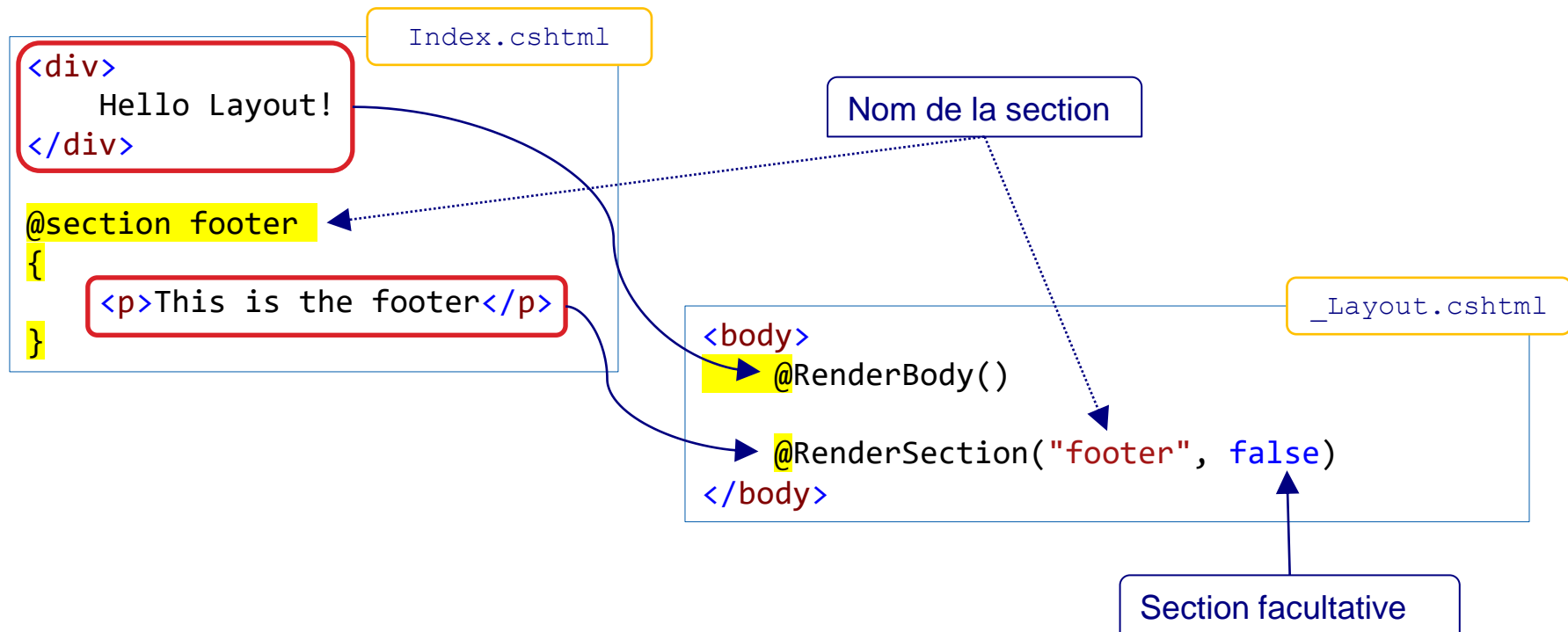
ViewStart

- **Pour éviter d'avoir à définir la propriété `Layout` dans toutes les vues**
 - La définir dans une vue `_ViewStart.cshtml`, dans le dossier `Views`
 - Dans une vue, donner la valeur `null` à `Layout` pour ne pas utiliser de vue de disposition



Sections des vues de disposition

- Une vue de disposition peut avoir plusieurs emplacements réservés
 - Avec une méthode `RenderSection`
 - Les sections sont définies dans la vue avec une directive `@section`
 - La méthode `RenderBody` génère tout le balisage qui n'est pas dans une section



Valeur par défaut d'une section

- Si une section n'est pas définie comme optionnelle, elle doit être présente dans les vues
 - Sinon, une erreur se produit en exécution
- On peut fournir du HTML par défaut dans la vue de disposition pour les sections optionnelles
 - Avec la méthode `IsSectionDefined` pour détecter si la section est définie dans la vue

`_Layout.cshtml`

```
<body>
  @RenderBody()

  @RenderSection("footer", false)
  @if (!IsSectionDefined("footer"))
  {
    @:This is the footer from the layout
  }
</body>
```

Le texte placé après la balise `@:` est envoyé au navigateur

Ne sera généré que si aucune section `footer` n'est définie dans la vue

Exercice 2.2 : Pages de disposition

Razor et les vues

- Vues, ViewBag et modèles
- La syntaxe Razor
- Structurer les vues avec des vues de disposition
- ➔ **Vues partielles et aides de vues**

Factoriser des parties de vues

- **On peut réutiliser des parties de vues de plusieurs façons**
 - Des méthodes `helper` utilisées dans une seule vue
 - Des vues partielles, fragments pouvant être réutilisés dans différentes vues
 - Des aides HTML, méthodes qui génèrent du HTML

Méthodes de vues helper

➤ Les méthodes `helper` sont définies dans une vue

- Avec un bloc Razor `@helper`
- Définit une méthode locale à la vue
- Peuvent avoir des paramètres

```
@Display("Hello") ASP.NET @Display("MVC")  
  
@helper DisplayBold(string text)  
{  
    <b>@text</b>  
}
```

Hello ASP.NET **MVC**

Vues partielles

➤ Une vue partielle est un fragment de vue

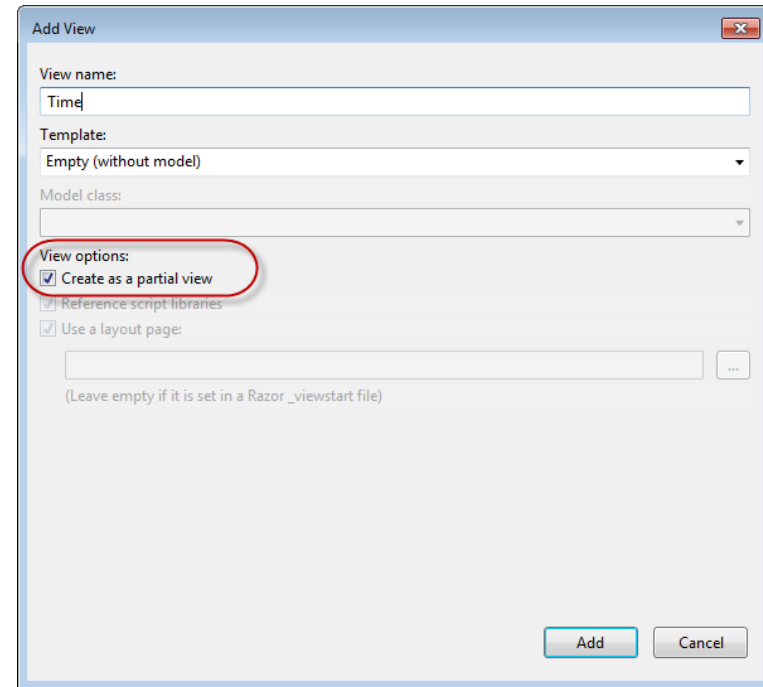
- Peut être utilisée dans plusieurs vues
- A du HTML et du code, comme les vues standard
- Peut être fortement typée, avec son propre modèle

➤ Pour créer une vue partielle

- Cliquer droit sur un dossier de vues et sélectionner Add View
- Cocher la case Create as partial view
- Un modèle peut être défini dans le dialogue ou avec une directive `@model`

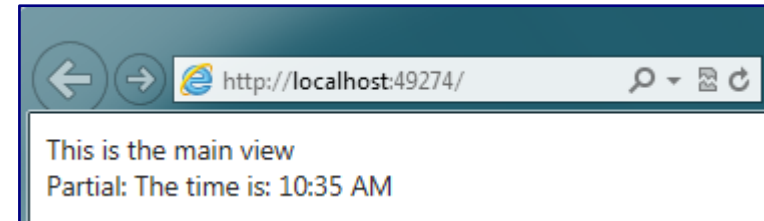
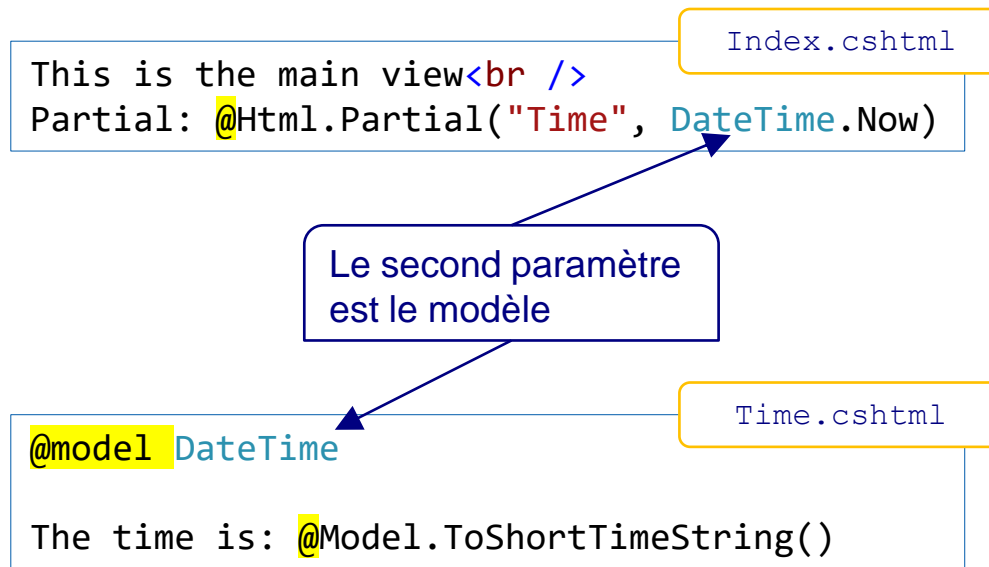
➤ Mettre la vue dans le dossier du contrôleur

- Ou dans le dossier `Views\Shared` afin qu'elle soit disponible pour toutes les vues



Utiliser une vue partielle

- **On peut afficher une vue partielle n'importe où dans une vue**
 - Avec la méthode `Html.Partial`
 - Prend le nom de la vue partielle en paramètre
- **Pour passer des données de la vue principale à la vue partielle**
 - Passer le modèle en tant que second paramètre
 - N'est pas nécessairement le modèle de la vue
 - On peut aussi passer un dictionnaire `ViewData` personnalisé



Aides HTML

- **Une aide HTML est une méthode qui renvoie une chaîne contenant du HTML**
 - Peut être utilisée dans une vue partout où du HTML est autorisé
 - ASP.NET MVC définit de nombreuses aides HTML
 - On peut créer des aides personnalisées
- **Les aides HTML sont définies dans la classe `HtmlHelper`**
 - Accessibles avec la propriété `Html` de la vue
 - L'aide `ActionLink` renvoie un lien
 - La méthode `Encode` renvoie une chaîne où les caractères spéciaux sont encodés
 - `>` est encodé en tant que `>`
 - La méthode `Raw` renvoie le HTML sans transformation
 - Il existe de nombreuses autres méthodes notamment utilisées avec les formulaires

Les formulaires et les aides personnalisées sont présentés dans le chapitre 5

Générer des liens dans une vue

➤ L'aide HTML `ActionLink` génère un lien dans une vue

- Génère un élément `<a>` standard à l'aide des informations de la table de routage
- Il existe plusieurs surcharges
- Dans le contrôleur `Home`

```
@Html.ActionLink("Click here", "Display")
```

→ `Click here`

```
@Html.ActionLink("People list", "List", "People")
```

→ `People list`

- Les paramètres sont passés dans un objet anonyme

```
@Html.ActionLink("Click here", "Display", new { name = "Fred" })
```

→ `Click here`

Le routage est présenté
dans le chapitre 4

Exercice 2.3 : Simplifier la vue et ajouter des styles

L'infrastructure des vues

- **Une vue est une instance de la classe `WebViewPage`**
 - Classe générique dont le type est celui du modèle
 - Le code source de la classe est généré dynamiquement lors de l'appel à la vue
 - Comprend le code qui génère les éléments HTML
 - Et le code Razor qui est inclus dans la page
- **La vue est compilée à la demande**
 - Les erreurs dans la vues n'empêchent pas l'application de s'exécuter
 - Bien qu'elles soient affichées dans la liste des erreurs lors de l'édition
 - Les erreurs de compilation sont affichées par le navigateur en exécution
 - Le développeur doit bien vérifier les vues et les tester avant le déploiement

La classe `WebViewPage`

➤ La classe `WebViewPage` a de nombreuses propriétés

- Documentée à msdn.microsoft.com/library

Propriété	Description
<code>Context</code>	Objet <code>HttpContext</code> associé à la page
<code>App</code>	Objet dynamique disponible dans l'application. On peut y placer des propriétés disponibles dans toutes les pages
<code>Html</code>	Objet <code>HtmlHelper</code> pour afficher des éléments HTML
<code>Model</code>	Le modèle associé à une page
<code>Request</code> , <code>Response</code> , <code>Server</code> , <code>Session</code>	Donnent accès aux objets ASP.NET correspondants
<code>TempData</code>	Dictionnaire pour stocker des données pendant une demande
<code>Url</code>	URL de la page
<code>ViewBag</code> , <code>ViewData</code>	Objet dynamique ou dictionnaire permettant de partager des données entre vue et contrôleur

Résumé du chapitre

Dans ce chapitre, nous avons

Créé des vues pour afficher des données à l'utilisateur

Généré du HTML de manière dynamique avec du code C# et Razor

Passé des données du contrôleur à la vue avec le `ViewBag` et le modèle

Structuré les vues avec des vues de disposition et des sections

Créé un contenu réutilisable avec des vues partielles et des aides de vues

Questions de révision

Citez différentes façons de passer de données du contrôleur à la vue

Écrivez une instruction qui donne le type `Person` au modèle dans une vue

Comment créer un bloc de code avec Razor en C# ?

Comment s'appelle l'emplacement réservé pour le contenu du corps dans une page de disposition ?

Où définir le nom de la page de disposition dans un projet ?

Quelles techniques permettent de factoriser des parties de vues ?

III. (69 à 106)

Le modèle

Objectifs du chapitre

Dans ce chapitre, nous allons

- **Construire une architecture solide pour le modèle**
- **Implémenter une couche référentiel avec Entity Framework**
- **Ajouter une couche de service pour implémenter la logique métier**
- **Rendre le couplage entre couches plus faible avec l'injection de dépendances**

- ➡ **Architecture du niveau métier**
 - **Accès aux données avec Entity Framework**
 - **Référentiel et requêtes**
 - **Logiques de service et métier**
 - **Découpler les couches avec l'injection de dépendance**

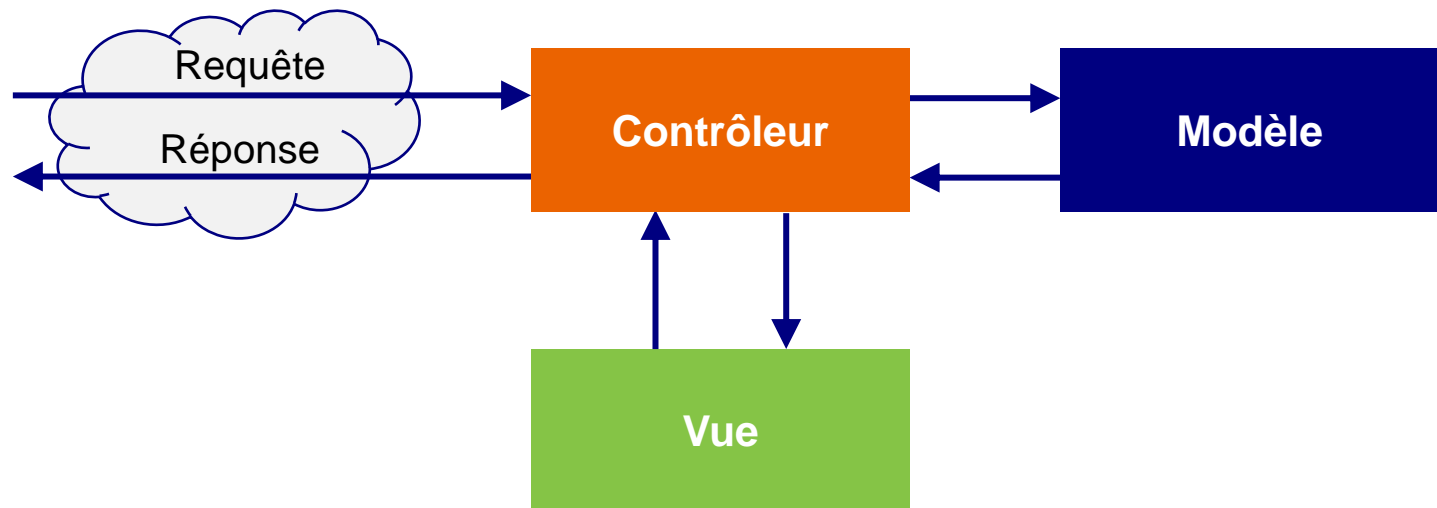
Le modèle dans l'architecture de ASP.NET MVC

➤ Quand l'utilisateur fait une demande

- Le contrôleur appelle une ou plusieurs méthodes du modèle
- Le modèle renvoie des données
- Le contrôleur passe les données à la vue
- La vue construit la sortie qui sera affichée dans le navigateur

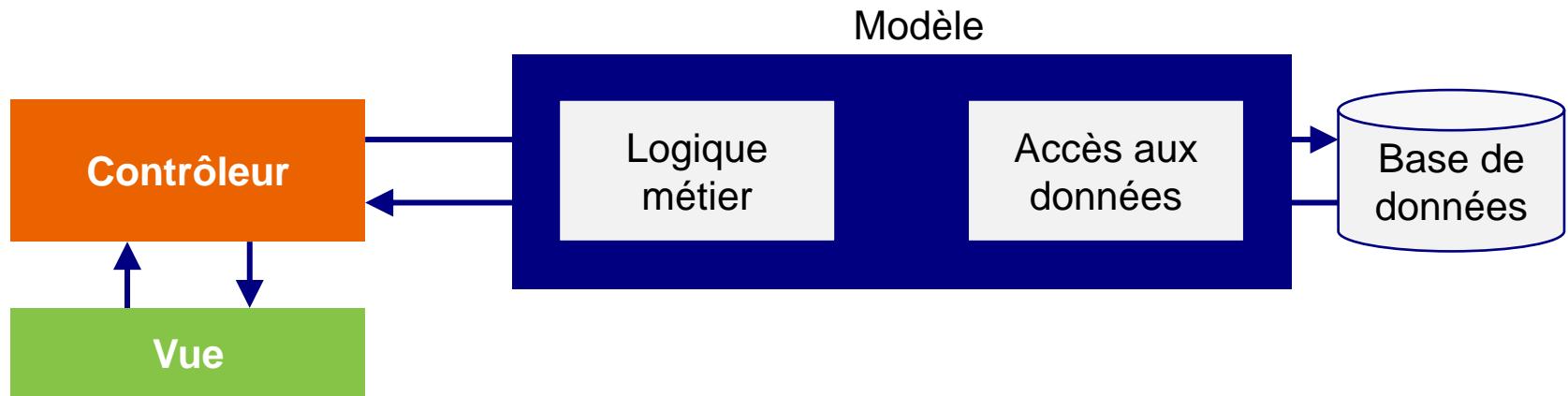
➤ Il n'y a pas de prise en charge du modèle dans ASP.NET MVC

- Juste un dossier vide dans le modèle du projet !



Du modèle à la base de données

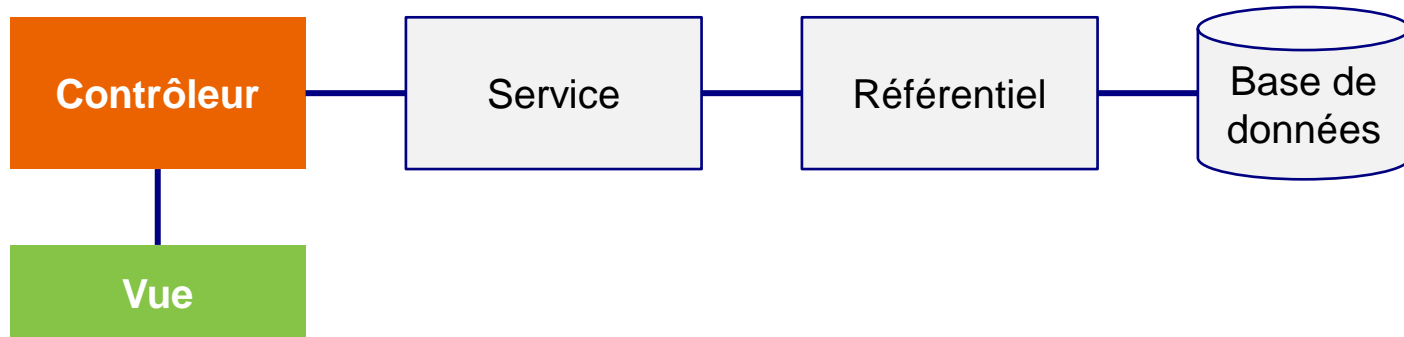
- **Le modèle est le passage vers les niveaux métier et données**
 - Comprend la logique métier et les règles de gestion
 - La logique métier utilise une couche données pour stocker et extraire les données
- **Le modèle comprend une ou plusieurs classes**
 - Peut être dans le projet MVC ou une bibliothèque de classe distincte
 - Tout ou partie du modèle peut être sur une machine différente
 - Accédée à travers des services Web tels que WCF ou Web API



WCF = Windows Communication Foundation

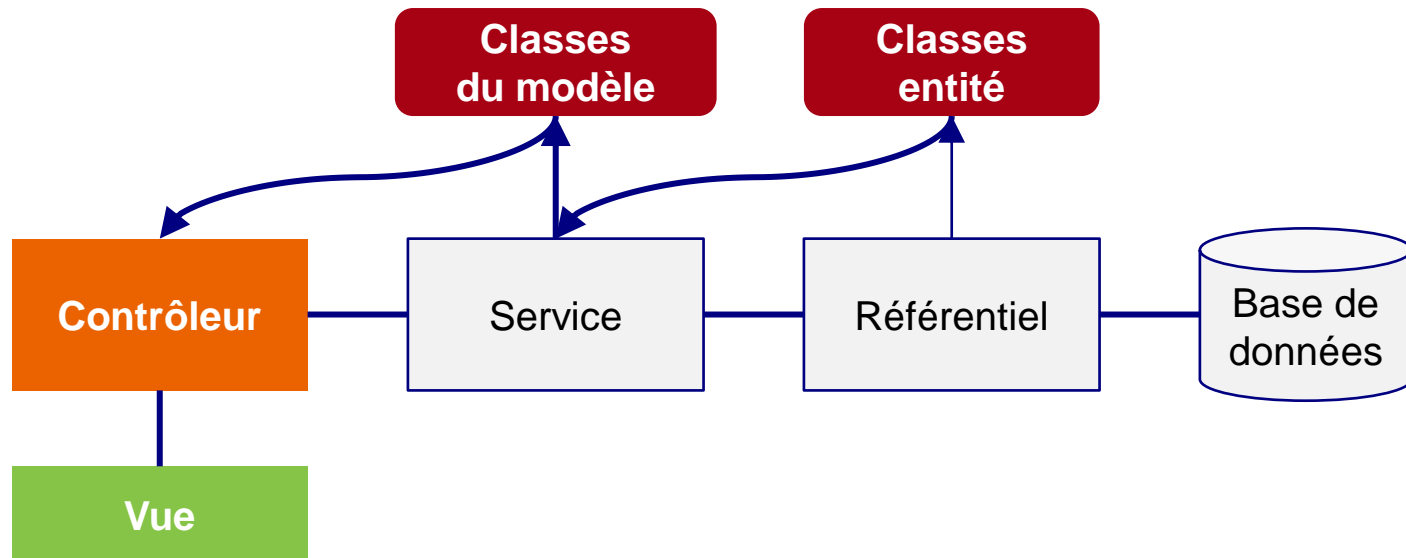
Les couches du modèle

- **Chaque couche du modèle doit faire une et une seule chose**
 - Le principe de responsabilité unique des design patterns
 - Chaque couche doit être aussi indépendante que possible des autres
 - Pour l'extensibilité et les évolutions
- **Les référentiels donnent accès à la base de données**
 - Aussi appelés couche d'accès aux données (DAL, Data Access Layer) ou objets d'accès aux données (DAO, Data Access Objects)
- **Les services ont des méthodes appelées par le contrôleur**
 - Obtiennent les données du référentiel et les transmettent au contrôleur
 - Le contrôleur est ainsi isolé des techniques d'accès aux données utilisées



Les classes de données

- **Les données vont de la base au contrôleur avec deux jeux de classes**
 - Des classes entité générées par le référentiel
 - Des classes modèle remplies par le service et utilisées par le contrôleur et la vue
 - Aussi appelées classes vue modèle
 - Le service transforme les entités en modèles et vice-versa
- **Cette architecture a beaucoup d'avantages**
 - Elle peut être trop complexe pour des petits projets



Le modèle

- **Architecture du niveau métier**
- ➡ **Accès aux données avec Entity Framework**
- **Référentiel et requêtes**
- **Logiques de service et métier**
- **Découpler les couches avec l'injection de dépendance**

Techniques d'accès aux données

➤ Plusieurs techniques permettent l'accès aux données

- Le code ADO.NET pur est la technique de bas niveau de .NET
- Le Dataset a été la première tentative d'ORM de Microsoft pour .NET 1 and 2

➤ Les ORM (Object-Relational Mapping) sont une technique souvent utilisée

- Des bibliothèques qui mappent les données relationnelles à des classes
- Il existe plusieurs ORM
 - Entity Framework fait partie de Visual Studio
 - Alternatives open source telles que nHibernate

➤ Mappage entre code .NET et bases de données relationnelles

Code .NET		Base relationnelle
Classe	↔	Table
Propriété	↔	Colonne
Objet	↔	Ligne
Collection	↔	Relation

Entity Framework

- **Entity Framework (EF) est l'ORM principal de Microsoft**
 - Maintenant open source, disponible sur CodePlex
 - Ne fait plus partie du noyau du Framework .NET
 - Peut être ajouté à un projet avec NuGet
 - Les modèles de MVC (sauf Empty) ajoutent Entity Framework au projet
- **Entity Framework peut se connecter à diverses bases de données**
 - Les différentes éditions de SQL Server
 - Oracle ou d'autres SGBD avec des pilotes adaptés

Édition de SQL Server	Description
Standard, BI, Web, Enterprise	Dans l'entreprise
Express	Édition gratuite limitée
Azure	SQL Server dans le nuage
LocalDB	Inclus dans VS pour le développement
Compact	Embarquée sur des matériels

SGBD = système de gestion de base de données

BI = Business Intelligence

Modes de Entity Framework

➤ Entity Framework peut s'utiliser selon trois modes principaux

- *Base d'abord* : Le designer EF de Visual Studio génère des classes à partir du schéma de la base de données
- *Modèle d'abord* : Le schéma est créé par le développeur dans le designer
- *Code d'abord* : Des classes existantes sont mappées à la base à l'aide d'attributs ou de classes de métadonnées
 - Au cours de cette formation, nous utiliserons le mode code d'abord, qui constituera l'essentiel des prochaines versions d'EF

➤ Code d'abord

- Possibilité de mapper les classes à la base de données
 - Ou d'utiliser Visual Studio pour générer des classes à partir de tables et de procédures stockées sélectionnées
- Il ajoute également des propriétés pour les clés primaires et les relations de clés étrangères

Exercice 3.1 : Ajouter Entity Framework à l'étude de cas

Fichiers générés par Entity Framework

➤ L'assistant Entity Framework lit le schéma de la base depuis SQL Server

- Il génère une classe d'entité pour chaque table sélectionnée
- Ainsi qu'une classe dérivée de `DbContext`
 - Cette classe est utilisée par l'application pour accéder aux données de la base

```
[Table("TodoItem")]
public partial class TodoItem
{
    public int Id { get; set; }
    public int CategoryId { get; set; }
    [Required]
    public string Title { get; set; }
    public bool Done { get; set; }
    (code omitted)
    public virtual Category Category { get; set; }
    public virtual Style Style { get; set; }
}
```

C#

Classe dérivée de DbContext

C#

```
public partial class TodoModel : DbContext
{
    public TodoModel()
        : base("name=TodoModel")
    {
    }

    public virtual DbSet<Category> Categories { get; set; }
    public virtual DbSet<Style> Styles { get; set; }
    public virtual DbSet<TodoItem> TodoItems { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Category>()
            .HasMany(e => e.TodoItems)
            .WithRequired(e => e.Category)
            .WillCascadeOnDelete(false);
    }
}
```

Le modèle

- **Architecture du niveau métier**
- **Accès aux données avec Entity Framework**
- ➡ **Référentiel et requêtes**
- **Logiques de service et métier**
- **Découpler les couches avec l'injection de dépendance**

Opérations CRUD dans le référentiel

➤ Les opérations de manipulation des la base se font avec la classe dérivée de DbContext

- La classe qui dérive de DbContext a une propriété collection DbSet pour chaque entité
- On peut les interroger avec LINQ

➤ Pour afficher la liste de toutes les catégories :

C#

```
public List<Category> List()
{
    using (var db = new TodoModel())
    {
        return db.Categories.ToList();
    }
}
```

La classe dérivée combine le nom de la base et les entités

Le bloc using appelle la méthode Dispose de DbContext, qui se connecte ensuite à la base de données

CRUD = Create, Retrieve, Update, Delete – Créer, Extraire, Modifier, Supprimer

Lire une seule entité

- La méthode `SingleOrDefault` renvoie une seule entité
 - Ou `null` si elle ne trouve rien

C#

```
public Category Read(int id)
{
    using (var db = new TodoModel())
    {
        var q = from cat in db.Categories
                where cat.Id == id select cat;
        return q.SingleOrDefault();
    }
}
```

- Si le critère est la clé primaire, le code peut être simplifié :

C#

```
return db.Categories.Find(id);
```

Ajouter une entité

- **Les modifications sont faites sur les collections DbSet**
 - Et répercutées dans la base avec la méthode `SaveChanges`
- **Pour ajouter un nouvel élément**

```
public void Add(Category category)
{
    using (var db = new TodoModel())
    {
        db.Categories.Add(category);
        db.SaveChanges();
    }
}
```

C#

Modifier une entité

➤ Pour modifier une entité existante

- Extraire l'entité
- Changer les propriétés modifiées
- Enregistrer les changements

➤ Ou rattacher l'entité

- Et indiquer que son état est modifié

C#

```
public void Update(Category category)
{
    using (var db = new TodoModel())
    {
        db.Categories.Attach(category);
        db.Entry(category).State = EntityState.Modified;
        db.SaveChanges();
    }
}
```

Supprimer une entité

➤ Pour supprimer une entité existante

- Extraire l'entité
- La retirer de la collection des entités
- Enregistrer les changements

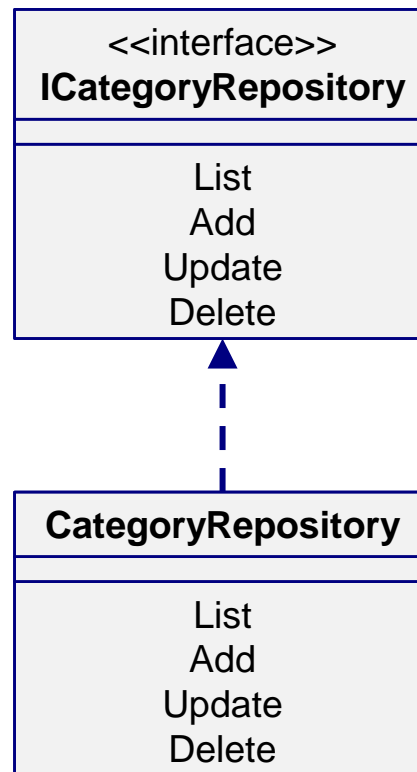
```
public void Delete(Category category)
{
    using (var db = new TodoModel())
    {
        var cat = db.Categories.Find(category.Id);
        db.Categories.Remove(cat);
        db.SaveChanges();
    }
}
```

C#

Exercice 3.2 : Programmer le référentiel

Applications faiblement couplées

- **Une couche devrait exposer ses fonctionnalités à travers des interfaces**
 - Si les classes sont exposées, la couche appelante dépend de l'implémentation
 - Les interfaces permettent de séparer les fonctionnalités de l'implémentation
 - Facilite le passage à une nouvelle implémentation
 - Par exemple, nHibernate en remplacement de Entity Framework



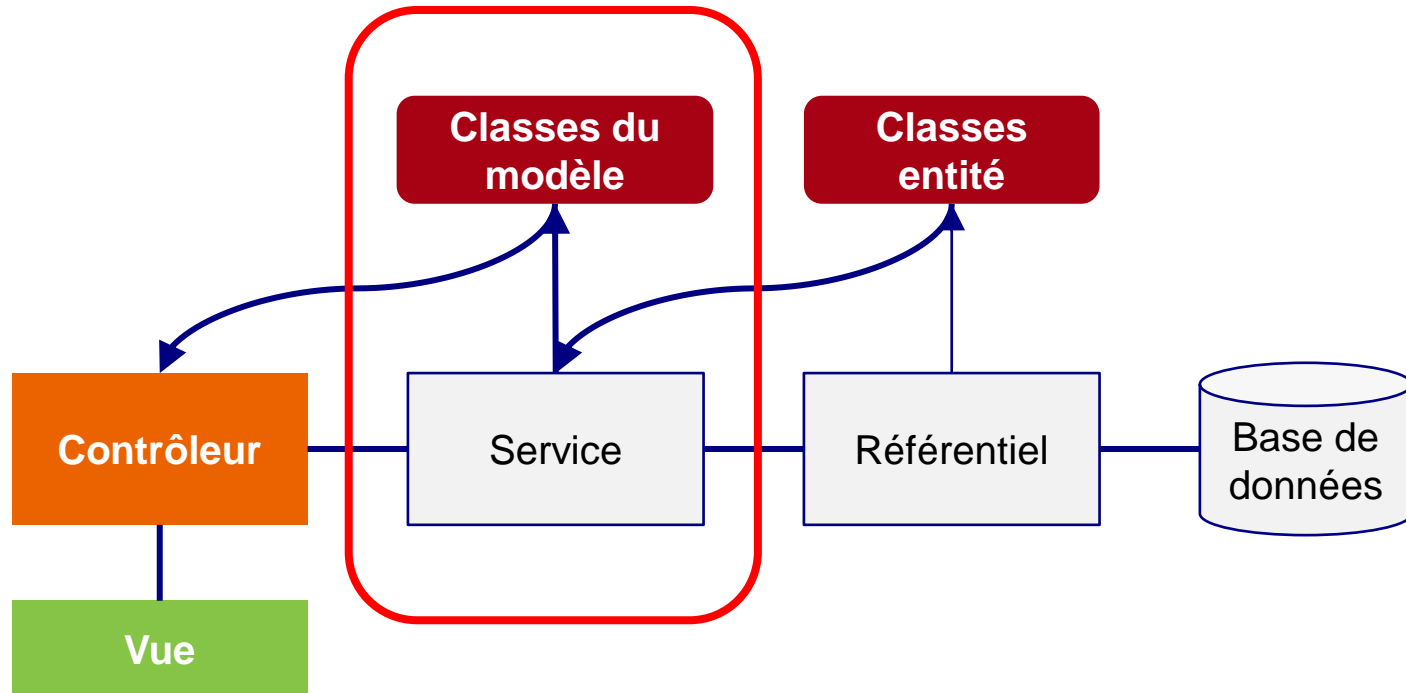
Le modèle

- **Architecture du niveau métier**
- **Accès aux données avec Entity Framework**
- **Référentiel et requêtes**
- ➡ **Logiques de service et métier**
- **Découpler les couches avec l'injection de dépendance**

La couche de service

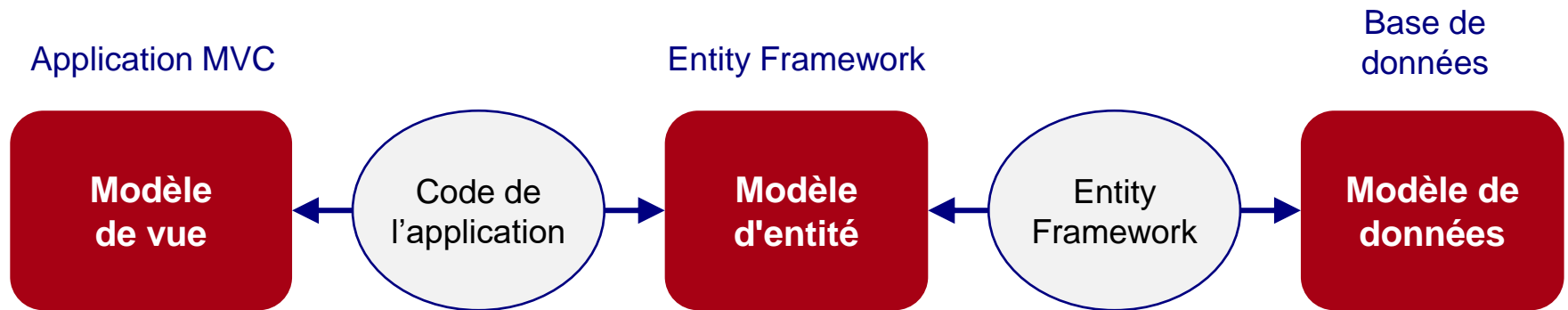
➤ La couche de service est le cœur de l'application

- Elle a des méthodes appelées par les contrôleurs
- Remplit les classes du modèle utilisées par les contrôleurs et vues pour afficher les données
- Implémente la logique métier de l'application



Les modèles de donnée, d'entité et de vue

- **Le modèle de donnée est défini dans la base avec du code SQL**
 - Entity Framework gère le mappage entre les modèles de donnée et d'entité
- **Le modèle d'entité est généré par Entity Framework**
 - Ou écrit par le développeur en mode code d'abord
- **Le modèle de vue est défini dans l'application MVC**
 - Le code de l'application doit mapper le modèle d'entité au modèle de vue
 - Peut être automatisé par des outils logiciels tels qu'AutoMapper



Les classes du modèle

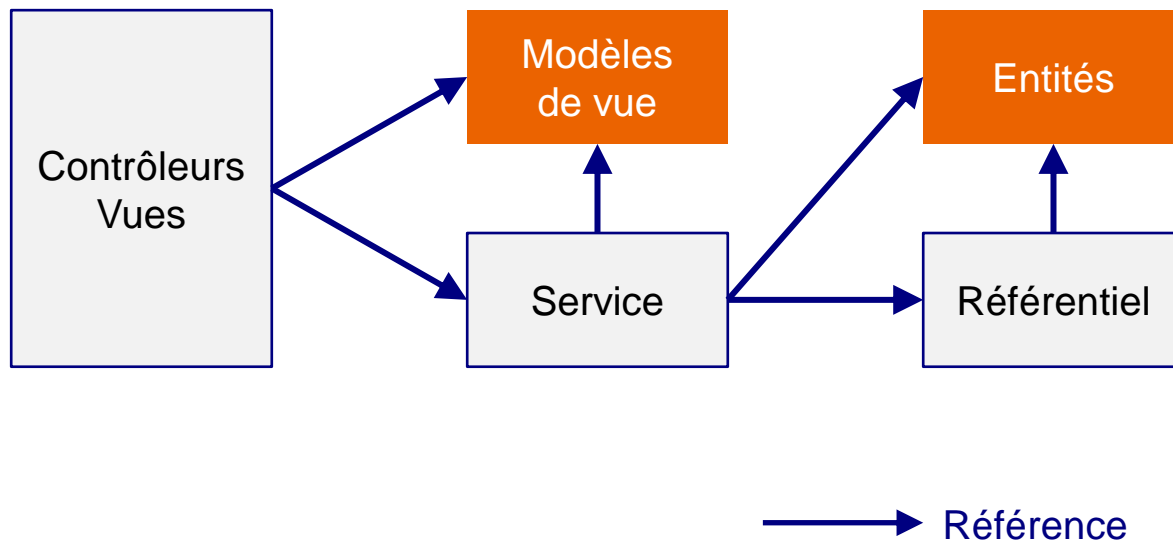
- **Les classes du modèle sont souvent semblables aux classes entité**
 - Chaque classe a les mêmes propriétés que l'entité
 - Peut avoir des propriétés complémentaires
 - Peut combiner des données de plusieurs entités
- **Le développeur MVC personnalise les classes du modèle selon les besoins**
 - Elles ne sont utilisées que par les contrôleurs et les vues
 - Les classes entité sont souvent définies dans des bibliothèques de classes non modifiables
- **On peut automatiser le mappage entre classes modèle et entité**
 - Avec une bibliothèque telle que AutoMapper
 - Gratuit, ajouté à un projet en tant que package NuGet
 - Par défaut, les propriétés sont mappées par nom
 - Le mappage peut être personnalisé pour chaque propriété
 - Voir `automapper.org` pour plus d'informations

Classes de service et interfaces

- **Le niveau de service devrait exposer ses fonctionnalités à travers des interfaces**
 - Comme la couche référentiel
 - Nécessaire pour implémenter l'injection de dépendance (sera vu plus tard)
- **Étapes pour l'extraction de données**
 1. Le contrôleur appelle une méthode du service
 2. Le service appelle une ou plusieurs méthodes du référentiel
 3. Les méthodes du référentiel obtiennent les données de la base grâce à Entity Framework
 4. Le référentiel renvoie des entités au service
 5. Le service implémente les règles métier et traite les entités renvoyées
 6. Le service construit un modèle et le renvoie au contrôleur
 7. Le contrôleur sélectionne la vue et lui passe le modèle
 8. La vue utilise le modèle pour générer le HTML

Références de projets

- **La couche du service « connaît » les modèles et les entités**
 - Les contrôleurs et les vues ne « connaissent » que les modèles
 - Le référentiel ne « connaît » que les entités



Exercice 3.3 : Ajouter le niveau de service

Le modèle

- **Architecture du niveau métier**
- **Accès aux données avec Entity Framework**
- **Référentiel et requêtes**
- **Logiques de service et métier**
- ➔ **Découpler les couches avec l'injection de dépendance**

Couplage fort

➤ Les couches de l'étude de cas sont fortement couplées

- Le contrôleur crée une instance du service et doit connaître la classe concrète qui implémente le service

```
_categoryService = new CategoryService();
```

C#

- De même, le service crée une instance du référentiel

```
_categoryRepository = new CategoryRepository();
```

C#

➤ Il faudrait modifier toute l'application pour remplacer une couche

- Par exemple, remplacer le référentiel par une implémentation avec nHibernate

Couplage faible avec l'injection de dépendances

- **L'injection de dépendance (DI, Dependency Injection) ou IoC (Inversion of Control) est un design pattern**
 - Couplage faible des composants
 - On fait référence à un composant à l'aide de son interface
 - L'implémentation concrète est injectée lors de l'exécution
- **Un conteneur DI est un intermédiaire entre les composants dépendants**
 - Les interfaces et les implémentations sont inscrites dans le conteneur
 - Quand on demande une interface, le conteneur cherche et fournit une implémentation concrète, un objet
 - L'implémentation est souvent injectée dans les constructeurs
 - La configuration peut se faire en XML ou par code
- **Il existe de nombreux conteneur DI, dont**
 - Ninject
 - Unity
 - StructureMap
 - Spring.NET
 - Etc.

Utiliser Ninject

➤ Le projet de l'étude de cas utilise Ninject

- Gratuit
- Simple à configurer
- Simple à utiliser

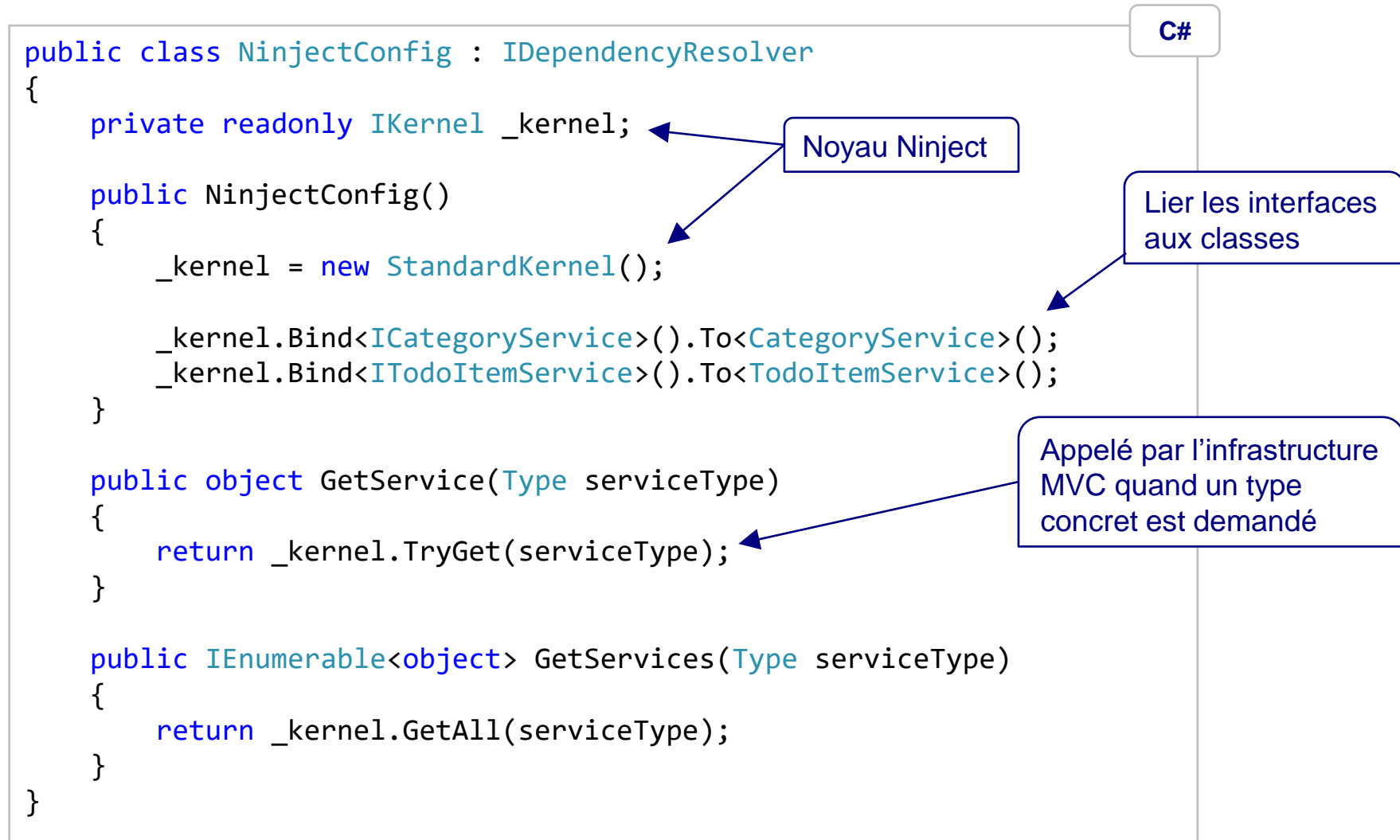
➤ Il faut installer Ninject pour pouvoir l'utiliser dans un projet

- Facilement fait avec NuGet
- Ajoute une référence à la DLL Ninject

➤ Pour configurer Ninject

- Définir une classe résolveur de dépendances qui implémente l'interface `IDependencyResolver`
 - La plupart du code est ajouté automatiquement au projet
 - Seules les liaisons doivent être personnalisées
- Inscrire ce résolveur de dépendances auprès de MVC

Le résolveur de dépendances



Inscrire le résolveur de dépendances

➤ Généralement fait dans `global.asax`

- La méthode `Application_Start` est appelée au démarrage de l'application
- Appeler la méthode statique `SetResolver` de la classe `MVC DependencyResolver`

```
DependencyResolver.SetResolver(new NinjectConfig());
```

C#

➤ Quand un contrôleur est demandé

- MVC appelle la méthode `GetService` du résolveur de dépendances
- Le noyau Ninject obtient le contrôleur à partir de son nom et cherche ses constructeurs
- Ninject instancie les classes concrètes liées aux interfaces demandées
 - Les passe en tant que paramètres au constructeur du contrôleur

Exercice facultatif 3.4 : Ajouter le conteneur DI Ninject

Résumé du chapitre

Dans ce chapitre, nous avons

- **Construit une architecture solide pour le modèle**
- **Implémenté une couche référentiel avec Entity Framework**
- **Ajouté une couche de service pour implémenter la logique métier**
- **Rendu le couplage entre couches plus faible avec l'injection de dépendances**

Questions de révision

Quel est le rôle du modèle dans l'architecture MVC ?

Citez les couches qui aident à structurer le modèle

Par quelle classe accède-t-on aux données avec Entity Framework ?

Comment casser les dépendances entre les couches du modèle ?

À quel élément de code sont mappés ces éléments de base de données avec un ORM ?

Table	
Colonne	
Ligne	
Relation	

IV. (107 à 154)

Le contrôleur

Objectifs du chapitre

Dans ce chapitre, nous allons

- Créer des contrôleurs avec la classe `Controller`
- Écrire des méthodes d'action avec des paramètres
- Examiner les différents type de résultats d'actions
- Écrire et utiliser des filtre
- Personnaliser la table de routage
- Structurer les applications avec des zones

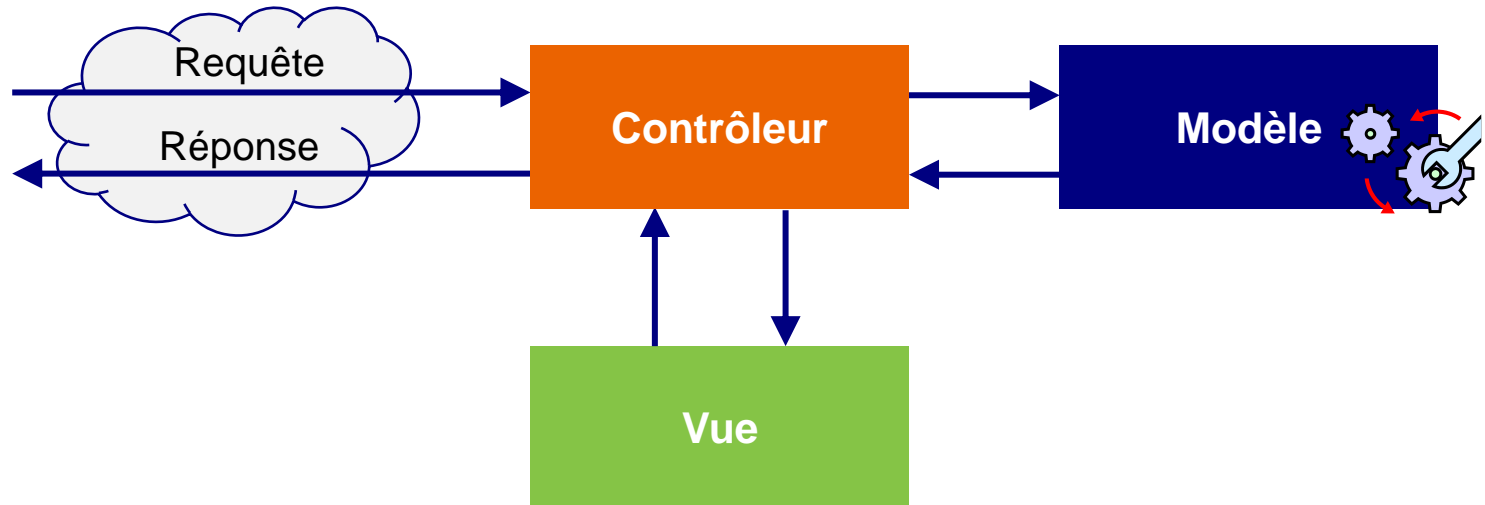


Le contrôleur

- ➡ **La classe Controller**
 - **Méthodes d'action et ActionResult**
 - **Les filtres**
 - **Le routage**
 - **Les zones**

Le contrôleur est le chef d'orchestre

- **Le contrôleur reçoit les requêtes du client**
 - Il demande au modèle de faire le traitement
 - Il sélectionne la vue pour générer la réponse



La classe Controller

- **Elle implémente l'interface `Controller`**
 - Les contrôleurs personnalisés dérivent généralement de `Controller`
- **La classe `Controller` a de nombreuses propriétés**
 - Dont certaines figurent dans le tableau ci-dessous

Propriété	Description
<code>HttpContext</code>	Informations HTTP sur la requête HTTP
<code>Request</code>	Contient des données sur la requête
<code>Response</code>	Permet d'ajouter des données à la réponse
<code>RouteData</code>	Données de routage pour la requête courante
<code>Session</code>	Objet session de ASP.NET
<code>TempData</code>	Dictionnaire permettant de stocker des données pendant une requête

- **Les classes dérivant de `Controller` contiennent des méthodes d'action**
 - Appelées quand le navigateur fait des requêtes

Stocker des données sur le serveur

- **ViewData, TempData et Session sont des dictionnaires**
 - La clé est une chaîne, la valeur un objet
 - Ils peuvent stocker n'importe quel type de donnée
- **ViewData ne dure que le temps d'une requête**
 - On peut également y accéder par le ViewBag
- **TempData permet de passer des données d'une requête à la suivante**
 - La lecture d'une donnée la retire du dictionnaire
 - La première requête définit la donnée, la suivante la lit, ce qui la détruit
- **Les données de Session durent aussi longtemps qu'une session**
 - Expire au bout de 20 minutes sans requête
 - Cela peut être changé dans l'élément `<sessionState>` de `Web.config`
 - On peut aussi fermer une session par code avec `Session.Abandon()`

Stocker des données sur le serveur – Exemple

➤ Définir des données dans TempData OU Session

```
TempData["name"] = "foo";  
or  
Session["name"] = "foo";
```

C#

➤ Obtenir des données de TempData OU Session

```
var nameTemp = TempData["name"];  
or  
var nameSession = Session["name"];
```

C#

Quelles sont les valeurs des variables `nameTemp` et `nameSession` ?

Le contrôleur

- La classe `Controller`
- ➡ **Méthodes d'action et `ActionResult`**
- Les filtres
- Le routage
- Les zones

Méthodes d'action

➤ Un contrôleur peut avoir plusieurs méthodes d'action

- Chacune correspond à une URL spécifique
- Selon les règles de routage de l'application

Le routage est couvert plus loin dans ce chapitre

`http://localhost:1234/Category/CategoryList`

```
public class CategoryController : Controller
{
    public ActionResult CategoryList()
    {
        return View();
    }
}
```

➤ Une méthode d'action renvoie un ActionResult

- Classe de base n'ayant qu'une seule méthode : `ExecuteResult`
- `ExecuteResult` génère la sortie et la met dans la réponse

Paramètres des méthodes d'action

➤ Une méthode d'action peut avoir des paramètres

- Les données peuvent provenir de la chaîne de requête (paramètres URL) ou de données de formulaire
- La chaîne de requête et les données de formulaire sont mappés aux paramètres par nom
- La liaison peut se faire sur des types individuels, des collections ou classes

`http://localhost:1234/ToDoItem/ItemList?id=12`

The diagram illustrates the mapping of a URL parameter to a C# method parameter. A red bracket underlines the `id=12` portion of the URL `http://localhost:1234/ToDoItem/ItemList?id=12`. A red arrow points from this bracket to a red bracket underlining the `int id` parameter in the C# method signature `public ActionResult ItemList(int id)`. A box labeled `C#` is connected to the code block by a line.

```
public ActionResult ItemList(int id)
{
    return View();
}
```

➤ Si un paramètre n'est pas dans la chaîne de requête ou le formulaire, une valeur nulle est passée

- Lève une exception pour les types valeur (nombres, dates...)
- On peut rendre les paramètres nullable ou définir une valeur par défaut

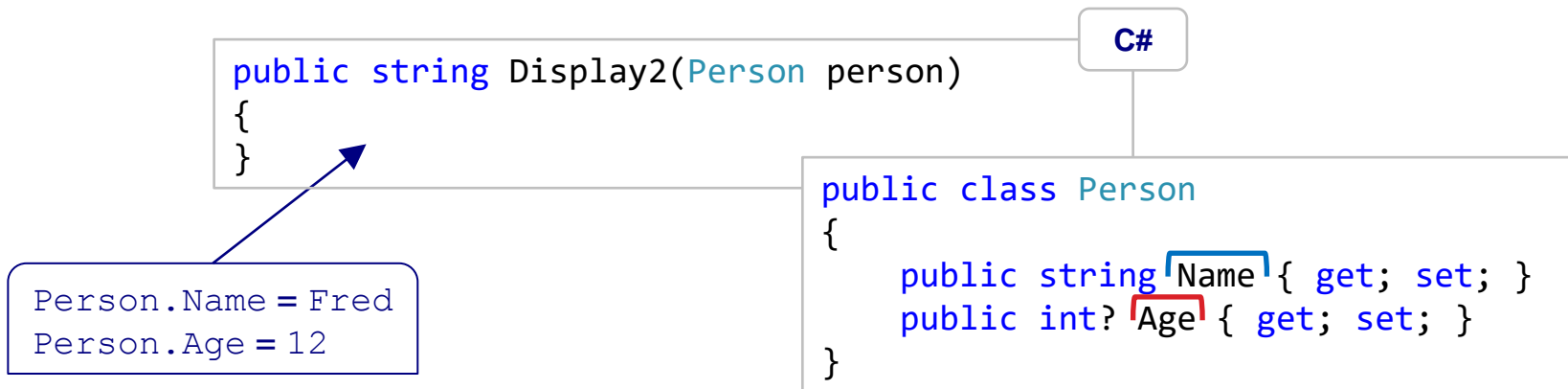
Nous étudierons les formulaires
et leurs données dans le chapitre 5

Mapper des paramètres à des données complexes

➤ Un paramètre de méthode d'action peut être une classe ou structure

- Le mappage est fait des données source vers chaque propriété
- Non sensible à la casse
- Les propriétés qui n'ont pas de données source seront nulles

`http://localhost:1234/Simple/Display2?Name=Fred&Age=12`



1. Ouvrez le point de départ Do Now 4a et exécutez l'application
 - Les noms du contrôleur et de la méthode d'action sont affichés
 2. Ajoutez `?name=Hello` à la fin de l'URL, après le caractère /
 - Qu'affiche le navigateur ?
-

3. Testez avec `?name=Hello&age=10` à la fin de l'URL
 4. Ajoutez un second paramètre entier `age` à la méthode `Index` du contrôleur `Home`, et modifiez le code afin d'afficher l'âge
 - Que se passe-t-il si vous retirez le paramètre `age` de l'URL ?
-

5. Corrigez le problème en rendant le paramètre `age` nullable, ou en lui donnant une valeur par défaut, puis testez
6. Utilisez la classe `Person` comme unique paramètre de la méthode `Display`, modifiez le code afin d'utiliser la classe et testez à nouveau

ActionResult

- **Une méthode d'action renvoie un objet `ActionResult`**
 - Ou une valeur nulle
 - Après le retour de la méthode d'action, le framework ASP.NET MVC appelle `ExecuteResult` sur l'objet `ActionResult`
- **`ActionResult` est une classe de base abstraite**
 - MVC comprend plusieurs classes concrètes dérivées
 - On peut créer des classes personnalisées dérivées de `ActionResult`
- **Chaque classe dérivée de `ActionResult` peut choisir comment traiter le résultat**
 - Généralement en mettant des données dans la réponse HTTP
- **La classe contrôleur a des méthodes d'aide pour générer des résultats d'action spécifiques**
 - Par exemple, `View` pour renvoyer un objet `ViewResult`

Les classes ActionResult

Classe	Méthode du contrôleur	Description
ViewResult	View	Renvoie le HTML d'une vue cshtml
PartialViewResult	PartialView	Renvoie le HTML défini dans une vue partielle
RedirectResult	Redirect	Redirige vers une URL
RedirectToRouteResult	RedirectToRoute	Redirige vers une URL selon les données de routage
FileResult	File	Classe de base pour renvoyer des données de fichiers
ContentResult	Content	Met une chaîne dans la réponse
JsonResult	Json	Renvoie des données JSON
EmptyResult	null	Ne renvoie rien

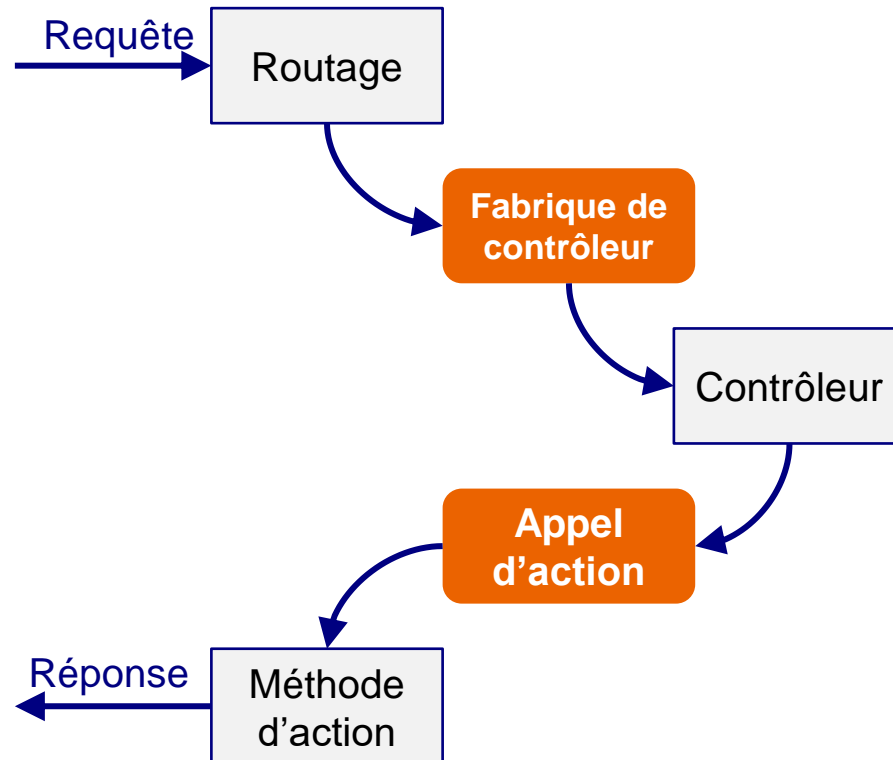
Le contrôleur

- **La classe Controller**
- **Méthodes d'action et ActionResult**
- ➡ **Les filtres**
- **Le routage**
- **Les zones**

Traiter une requête dans une application MVC

- 1. Le module de routage analyse l'URL et construit un `RouteData`**
 - Avec le nom du contrôleur, celui de l'action et les paramètres
 - Mappe des éléments de l'URL aux membres de `RouteData` avec la table de routage
- 2. L'objet `RouteData` est passé à la fabrique de contrôleur**
 - La fabrique de contrôleur recherche la classe du contrôleur en utilisant le nom dans l'objet `RouteData`, puis l'instancie
- 3. Le contrôleur appelle l'invocateur d'action, en lui passant le nom de l'action**
 - L'invocateur d'action recherche la méthode d'action et l'invoque
- 4. La méthode d'action demande au modèle de faire le traitement et prépare un objet `ActionResult`**
 - Généralement une vue
 - Peut aussi être du texte, une image, des données JSON, une instruction de redirection...
- 5. La méthode `ExecuteResult` du `ActionResult` est appelée**
 - Pour une vue, elle construit la réponse renvoyée au client

De la requête à la réponse



Attributs et filtres MVC

- **Les attributs sont des classes .NET qui dérivent de la classe `Attribute`**
 - Le nom se termine généralement par `Attribute`
 - Ils peuvent être utilisés avec ou sans le suffixe `Attribute`
 - Appliqués à des classes, interfaces, propriétés, méthodes, paramètres...
 - Le compilateur, le framework MVC, une application peuvent les interroger
- **Les filtres sont des attributs pouvant être mis sur des méthodes d'action ou des contrôleurs**
 - Si un attribut est appliqué à un contrôleur, il s'applique à toutes ses méthodes d'action
 - On peut aussi les appliquer à tous les contrôleurs dans `global.asax`
- **Les filtres MVC dérivent de la classe `FilterAttribute`**
 - MVC définit des filtres intrinsèques
 - Les filtres d'action personnalisés dérivent de la classe `ActionFilterAttribute`

Filtres prédéfinis

➤ Principaux filtres prédéfinis

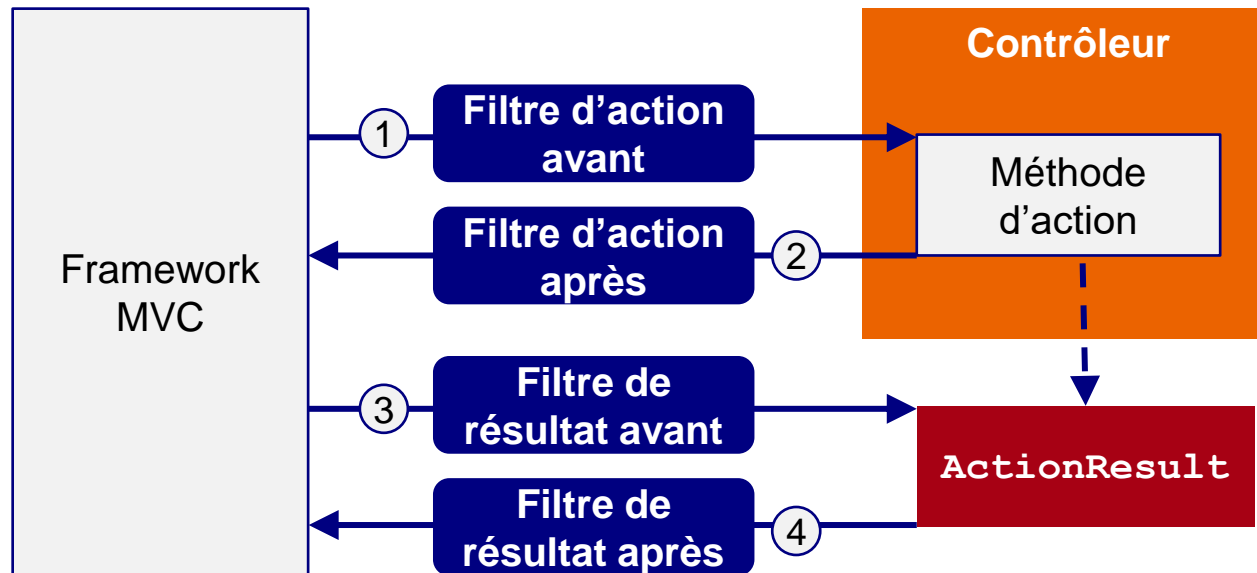
Attribut de filtre	Description
<code>Authorize</code>	Autorise l'appelant à exécuter la méthode d'action
<code>HandleError</code>	Traitement des exceptions non gérées
<code>RequireHttps</code>	Requiert ou redirige vers HTTPS pour l'appel des actions
<code>ValidateInput</code> <code>ValidateAntiForgeryToken</code>	Sécurise la saisie de l'utilisateur
<code>ChildActionOnly</code>	L'action ne peut être appelée que depuis les méthodes d'aide <code>Action</code> ou <code>RenderAction</code>
<code>OutputCache</code>	Met la sortie de la méthode d'action en cache

La plupart des filtres prédéfinis sont couverts dans d'autres chapitres

Filtres d'action et de résultat

➤ MVC définit deux types de filtres

- Les filtres d'action sont exécutés avant et après une méthode d'action
- Les filtres de résultat sont exécutés avant et après le traitement du résultat



➤ On peut aussi définir des filtres d'action personnalisés

- En implémentant les interfaces `IActionFilter` et / ou `IResultFilter`
- Ou dans une classe qui dérive de `ActionFilterAttribute`

Filtre d'action personnalisé

- **ActionFilterAttribute** implémente les interfaces **IActionFilter** et **IResultFilter**
 - Pour l'utiliser, définir une classe qui dérive de `ActionFilterAttribute`
 - A quatre méthodes pouvant être redéfinies

Méthode	Description
<code>OnActionExecuting</code>	① Appelée avant l'exécution d'une méthode d'action
<code>OnActionExecuted</code>	② Appelée après l'exécution d'une méthode d'action
<code>OnResultExecuting</code>	③ Appelée avant l'exécution de la méthode <code>ExecuteResult</code> sur un résultat d'action
<code>OnResultExecuted</code>	④ Appelée après l'exécution de la méthode <code>ExecuteResult</code> sur un résultat d'action

1. Ouvrez le point de départ Do Now 4b
2. Ouvrez la classe `ProfileFilterAttribute` dans le dossier `Filters`
 - Elle démarre un chronomètre quand l'action commence et l'arrête à la fin
3. Ouvrez le contrôleur `Home`. Ajoutez un attribut `ProfileFilter` à la méthode `Index`
 - Code C# : `[ProfileFilter]`
 - Ajoutez l'instruction `using` manquante à `ActionFilters.Filters`
4. Appelez `System.Threading.Thread.Sleep` avant de renvoyer la vue, avec une durée de 500 millisecondes
5. Exécutez l'application avec <F5> et examinez la fenêtre Output lors de l'actualisation de la page `Index`
 - La valeur affichée devrait être proche de 500

Attribut OutputCache

- **Met la sortie d'une méthode d'action en cache**
 - À l'aide de l'infrastructure de cache de ASP.NET
 - Lors du premier appel de la méthode d'action, le code s'exécute normalement
 - Les appels suivants renvoient les données en cache
- **L'utilisation d'un cache de sortie peut améliorer les performances**
 - Mais la sortie ne correspond pas à l'état courant des données
 - Pas approprié quand les données sont souvent modifiées
- **La propriété `Duration` indique la durée du cache en secondes**
 - Le code de la méthode est à nouveau exécuté lors de la requête suivant l'expiration de la durée
 - D'autres propriétés permettent de régler le comportement du cache

Propriétés de l'attribut OutputCache

➤ Principales propriétés de l'attribut OutputCache

Propriété	Description
Duration	Durée en secondes avant l'expiration du cache
VaryByParam	Liste séparée par des point-virgule de valeurs GET ou POST associées à des copies distinctes en cache. Si *, toute variation est mise en cache séparément.
Location	Emplacement de stockage du cache. Peut être Server, Client, Downstream (client ou proxy), ServerAndClient, Any, ou None
CacheProfile	Utiliser les paramètres de OutputCacheSettings dans Web.config
SqlDependency	Utiliser la fonctionnalité de dépendance de cache de ASP.NET / SQL Server

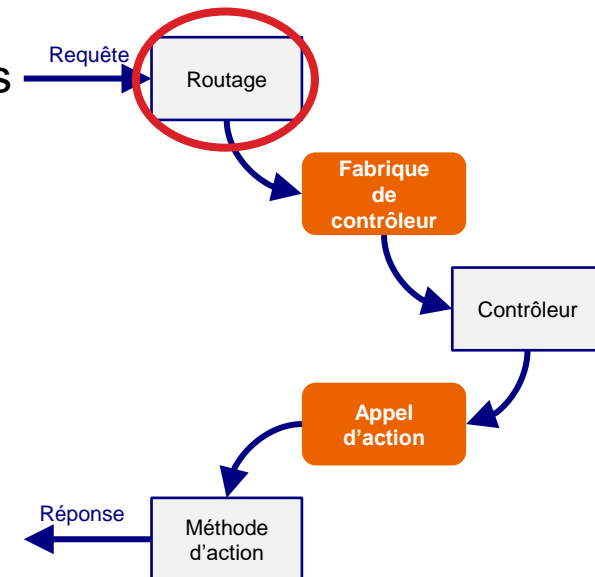
Exercice 4.1 : Mettre en cache la sortie d'une action

Le contrôleur

- **La classe Controller**
- **Méthodes d'action et ActionResult**
- **Les filtres**
- ➡ **Le routage**
- **Les zones**

Le routage

- **Mappe les URL entrantes à des méthodes d'action dans des contrôleurs**
 - Traite également les paramètres
 - Appelé routage *entrant*
- **Génère également des URL à partir des noms de contrôleur et d'action**
 - Y compris les paramètres
 - Appelé routage *sortant*
- **Le routage entrant est le premier traitement fait sur une requête**
 - L'URL demandée est analysée
 - Les noms du contrôleur et de l'action en sont extraits
- **Deux stratégies pour définir les routes**
 - Routage basé sur des conventions
 - Utilise une table de routage initialisée au démarrage de l'application
 - Routage par attributs (nouveau dans MVC 5)
 - On peut utiliser les deux dans une même application



Pourquoi le routage ?

➤ Les URL sont souvent mappées directement aux dossiers du site Web

`http://www.server.com/admin/pages/index.html`

- Pourrait être mappé à

`C:\wwwroot\accounting\admin\pages\index.html`

- Les URL peuvent aussi avoir des paramètres

`http://www.server.com/admin/pages/index.html?id=567h98gf-66&opt=password`

➤ Horribles URL

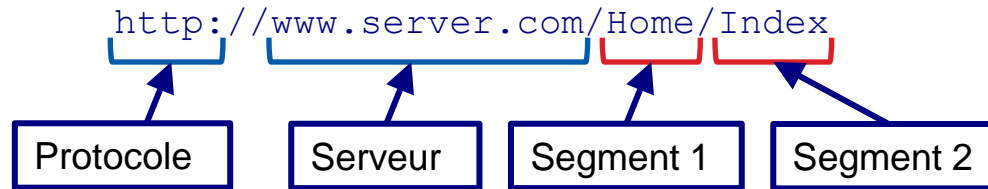
- Difficiles à mémoriser
- Peuvent être modifiées, ce qui rend les signets peu fiables
- Peuvent révéler l'architecture sous-jacente

➤ Le routage supprime le lien direct entre l'URL et la structure physique

- Donne au développeur un contrôle complet de l'URL
- Des URL courtes sont plus faciles à taper
- Moins sensibles aux modifications
- Optimisées pour les moteurs de recherche (SEO, Search Engine Optimization)
 - Le poids des mots de l'URL est plus élevé pour les moteurs de recherche

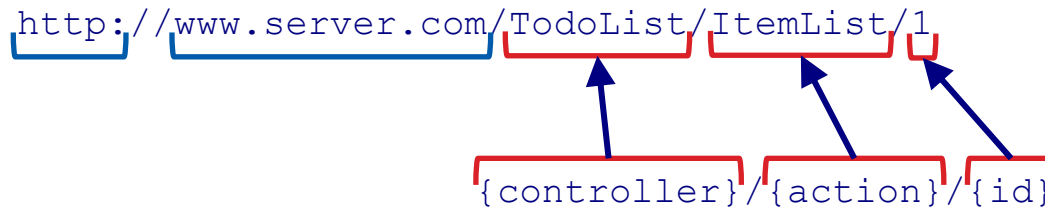
Structure de l'URL et modèles de route

- Les règles de routage s'appliquent aux segments de l'URL suivant le nom du serveur



- Une table de routage définit un modèle d'URL

- Mappé aux segments d'URL séparés par des barres obliques
- Les segments nommés sont entre accolades



- Avec cette URL et ce modèle
 - La valeur de route `controller` est `ToDoList`
 - La valeur de route `action` est `Index`
 - La valeur de route `id` est `1`

Utiliser les valeurs de route

➤ Les segments de route sont mises dans la propriété `Values` de la classe `RouteData`

- `Values` est un dictionnaire dont les clés sont les noms des parties de route
- MVC utilise la valeur de `controller` pour trouver le contrôleur, et celle de `action` pour la méthode d'action
- Également disponibles dans le contrôleur à travers la propriété `RouteData`

```
RouteData.Values["action"]
```

C#

- On peut aussi obtenir les parties personnalisées en tant que paramètres des méthodes d'action :

```
public ActionResult List(int id)
```

C#

Mappé depuis une valeur
de route appelée `id`

Inscrire les routes

- **Les routes sont inscrites dans une table de routage**
 - Typiquement au démarrage de l'application
 - Les modèles de projet de Visual Studio définissent une classe `RouteConfig` dans le dossier `App_Start`
 - Appelée depuis `Application_Start` dans `global.asax`
- **Table de routage par défaut**

C#

```
routes.MapRoute(  
    name: "Default",  
    url: "{controller}/{action}/{id}",  
    defaults: new  
        { controller = "Home", action = "Index", id = UrlParameter.Optional }  
);
```

Contrôleur par défaut `Home`

Action par défaut `Index`

Paramètre `id` optionnel

Ajouter une nouvelle route

➤ Une table de routage a souvent plusieurs routes

- Les routes sont analysées successivement lors de l'arrivée d'une requête
- Dans l'ordre d'ajout dans la table
- La première qui correspond est utilisée

➤ Mettre les routes les plus restrictives en premier

- La première route intercepte une URL telle que `/List/2`
- Le contrôleur et l'action sont codés en dur dans la route

```
routes.MapRoute("List", "List/{id}",  
    new { controller = "ToDoList", action = "ItemList" });  
  
routes.MapRoute("Default", // Default route omise (voir précédent slide)
```

C#

Routage par attributs

➤ On peut encore personnaliser la table de routage

- Mais cela est plus facile avec le routage par attributs
- Nouveau dans MVC 5
- Peut être installé avec un package NuGet dans les versions précédentes

➤ Pour utiliser le routage par attributs

- Ajouter une ligne d'initialisation dans la méthode `RouteConfig.RegisterRoutes`

```
routes.MapMvcAttributeRoutes();
```

C#

- Ajouter des attributs `Route` sur les méthodes d'action et/ou les classes de contrôleur

```
[Route("List/{id}")]  
public ActionResult ItemList(int id)
```

C#

Une URL telle que `/List/2` sera mappée à la méthode d'action `ItemList`

Valeurs par défaut

➤ On peut définir des valeurs par défaut pour les segments de route

- Pour que l'URL `/List` soit équivalente à `/List/1` :

```
Route("List/{id=1}")
```

➤ On peut aussi appliquer des valeurs par défaut aux méthodes d'action

- Pour que `Index` soit la méthode d'action par défaut, appliquer l'attribut au contrôleur :

```
Route("{action=Index}")
```

➤ Un paramètre peut être optionnel avec un suffixe ?

- La route suivante sera traitée pour `/List/Food` ou `/List`

```
Route("List/{name?}")
```

Contraintes de route

➤ Une contrainte de route restreint une route à des types de données ou des valeurs spécifiques

- Appliquer la contrainte à un segment de route

```
Route("List/{id:int}")
```

- Cette route ne s'appliquera que si le segment `id` de l'URL est un entier

➤ Il existe de nombreux types de contraintes

- `alpha`, `bool`, `datetime`, `decimal`, `double`, `float`, `guid`, `int`, `length`, `long`, `max`, `maxlength`, `min`, `minlength`, `range`, `regex`
- On peut aussi les chaîner

```
Route("List/{id:int:min(0):max(10)}")
```

- Cette route ne s'appliquera que si le segment `id` de l'URL est un entier entre 0 et 10

Factoriser des segments de route

➤ Plusieurs méthodes d'action d'un contrôleur peuvent avoir un segment commun

- Généralement le nom du contrôleur

```
[Route("Home/Index")]  
public ActionResult Index()
```

C#

- Peut n'être défini qu'une fois avec le préfixe `RoutePrefix` sur le contrôleur

```
[RoutePrefix("Home")]  
public class HomeController : Controller  
{  
    [Route("Index")]  
    public ActionResult Index()  
}
```

C#

- **Votre instructeur va vous montrer la factorisation de segments de route**
 - Ouvrez la solution à `Demo\Chapter 4\AttributeRouting`
 - Notez que la route par défaut a été supprimée dans `RouteConfig`
 - Ouvrez `HomeController`. Notez que l'attribut `Route` a été défini. Notez aussi l'utilisation d'un modèle de route vide pour la route par défaut
- **Factoriser le segment Home des routes**
 - Ajoutez un `RoutePrefix` au contrôleur `Home`
 - Supprimez le segment `Home` des méthodes d'action
- **Redéfinir le préfixe de route**
 - Changez la route vide de `Index` en `~/`
 - Ajoutez une route `~/Show` à la méthode d'action `Display`

La démo terminée est à `Demo\Chapter 4\AttributeRouting - Completed`

Exercice 4.2 : Personnaliser des routes

Contourner le système de routage

- **Certaines ressources ne doivent pas être traitées par la table de routage**
 - Les ressources `*.axd` sont traitées par des gestionnaires HTTP de ASP.NET
 - Peuvent comprendre des groupes de JavaScript compressés
- **IgnoreRoute interrompt le traitement pour le chemin mappé**
 - Les modèles de Visual Studio ajoutent cette ligne à la méthode `RegisterRoutes` :

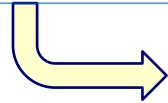
```
routes.IgnoreRoute("{resource}.axd/{*pathInfo}")
```
 - Toute requête d'URL ayant une extension `axd` est ignorée
 - `resource` est le nom de la ressource dans l'URL
 - `pathInfo` contient les parties restantes de l'URL (attrape-tout)
- **L'URL est passée à ASP.NET**
 - L'extension `axd` redirige vers le gestionnaire HTTP correspondant

Routage sortant

➤ Le routage sortant construit des URL en utilisant la table de routage table

- L'aide HTML `ActionLink` utilise la table de routage
- Avec la route par défaut :

```
@Html.ActionLink("About", "About", new { id = 15 })
```



```
<a href="/Home/About/15">About</a>
```

➤ `RouteLink` est semblable

- Mais peut aussi créer un lien pour d'autres ressources, comme des fichiers
- Ce lien est le même que le précédent :

```
@Html.RouteLink("About", new { action = "About", id = 15 })
```

➤ La classe d'aide `Url` crée des URL selon la table de routage

```
var url = Url.RouteUrl(new { action = "About", id = 15 });
```

C#

`url` contient `/Home/About/15`

Routage sortant et routage par attributs

- **ActionLink utilise les attributs Route définis pour créer un lien**
 - On peut le voir dans la démo et l'exercice précédents
- **Des surcharges de `Html.RouteLink` et `Url.RouteUrl` prennent un nom de route**
 - Peut être utilisé avec le routage par attributs
 - Il faut donner un nom à l'attribut `Route`

```
[Route("~/Show", Name="ShowRoute")]
```

- Peut être utilisé avec

```
@Html.RouteLink("Display", "ShowRoute")
```

- Ou

```
<a href="@Url.RouteUrl("ShowRoute")">Display</a>
```

Code à `Demo\Chapter 4\Outgoing AttributeRouting`

Le contrôleur

- **La classe Controller**
- **Méthodes d'action et ActionResult**
- **Les filtres**
- **Le routage**
- ➡ **Les zones**

De la nécessité des zones

➤ Les applications ASP.NET MVC peuvent grossir

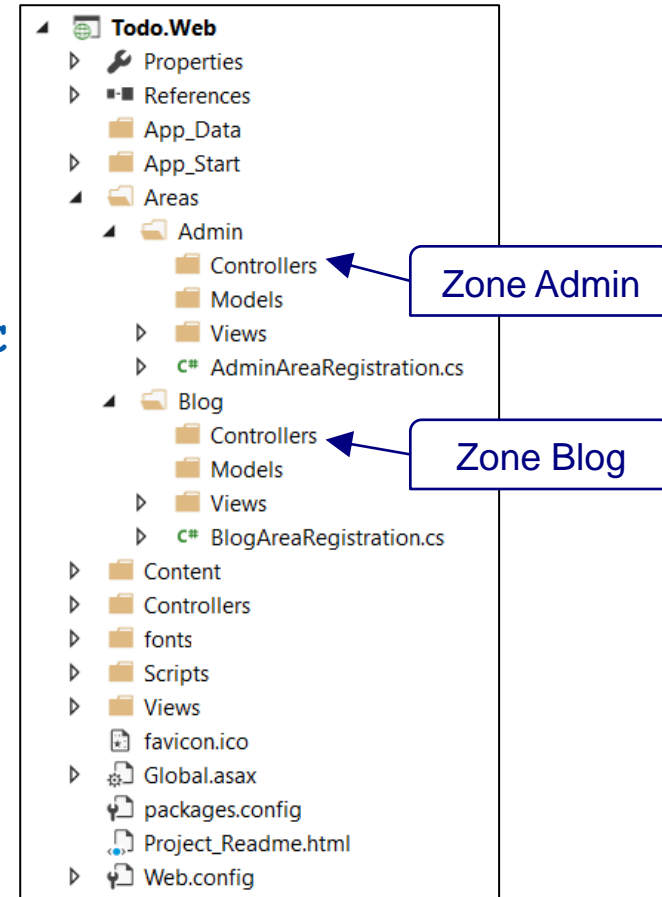
- Avec des centaines de contrôleurs et des milliers de vues
- Gérer et nommer les vues et contrôleurs peut devenir complexe

➤ Les zones ajoutent une dimension au projet

- Un dossier Areas dans lequel la structure Modèle-Vue-Contrôleur est répliquée
- Il peut y avoir autant de zones que nécessaire

➤ Pour ajouter une zone à un projet ASP.NET MVC

- Cliquer droit sur le projet, sélectionner **Add | Area** et donner un nom à la zone
- Le dossier `Areas` est créé la première fois
- Des dossiers `Controllers`, `Models` et `Views` sont ajoutés à chaque zone



Routage et zones

➤ Chaque zone ajoute des éléments dans la table de routage

- Dans une classe ajoutée à la racine de la zone
- Toutes les classes qui dérivent de `AreaRegistration` sont appelées depuis `global.asax`

```
AreaRegistration.RegisterAllAreas();
```

➤ Pour utiliser le routage par attributs avec des zones

- Supprimer la classe qui dérive de `AreaRegistration`
- Ajouter un attribut `RouteArea` aux contrôleurs de la zone
- Définir `RoutePrefix` et l'action par défaut

```
[RouteArea("Admin")]  
[RoutePrefix("Users")]  
[Route("{action}")]  
public class UsersController : Controller
```

C#

Liens sortants pour les zones

➤ **ActionLink peut générer des liens vers des zones**

- Ajouter une propriété `area` au paramètre `RouteValues`

```
@(Html.ActionLink("Administration", "Index",  
    new { area = "Admin", controller = "Users" })))
```

C#

➤ **Les vues d'une zone ont leur propre page de disposition par défaut**

- On peut réutiliser la vue `layout` principale avec un `_ViewStart.xxhtml` dans la zone

Exercice facultatif 4.3 : Structurer une application avec des zones

Résumé du chapitre

Dans ce chapitre, nous avons

- Créé des contrôleurs avec la classe `Controller`
- Écrit des méthodes d'action avec des paramètres
- Examiné les différents type de résultats d'actions
- Écrit et utilisé des filtre
- Personnalisé la table de routage
- Structuré les applications avec des zones

Questions de révision

Quelle interface doit implémenter un contrôleur ?

Quel est le type de retour d'une méthode d'action ?

Comment MVC trouve-t-il le contrôleur à instancier à partir de son nom ?

Citez des filtres intrinsèques

Quelles sont les deux responsabilités principales du système de routage ?

Comment structurer une application MVC qui a beaucoup de contrôleurs ?

V. (155 à 215)

Formulaires et saisie utilisateur

Objectifs du chapitre

Dans ce chapitre, nous allons

- Créer des formulaires pour la saisie utilisateur
- Simplifier le codage du HTML avec des aides HTML
- Développer des aides HTML personnalisées
- Valider la saisie et appliquer les règles métier avec des annotations de données

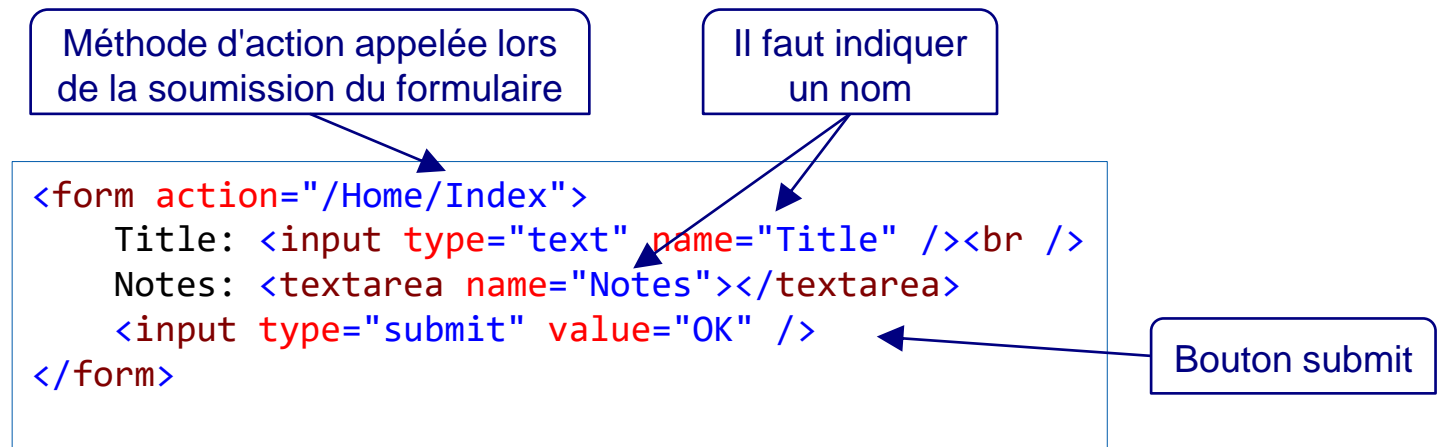
- ➡ **Formulaires HTML et MVC**
 - **Simplifier la génération du HTML avec des aides**
 - **Développer des aides HTML personnalisées**
 - **Méthodes d'aide pour l'affichage et l'édition**
 - **Validation des données**

Saisir les données utilisateur

- **Peut se faire avec**
 - Des formulaires HTML standard
 - Du HTML avec JavaScript et Ajax pour un contrôle plus précis
- **Dans tous les cas, le développeur contrôle complètement le HTML**
 - Une page peut avoir plusieurs formulaires
 - Les Web Forms ASP.NET ne peuvent avoir qu'un formulaire côté serveur
- **On peut créer des formulaires avec des éléments de saisie HTML standard**
 - Peuvent être simplifiés et optimisés avec des aides HTML
 - Peuvent même n'avoir qu'une ligne de code Razor
 - Tout est alors piloté par le modèle
- **Visual Studio peut aussi construire des formulaires à partir de modèle**
 - Le code Razor et HTML est généré par Visual Studio
 - Peut être personnalisé

Formulaires HTML

- **La saisie utilisateur se fait à l'aide d'éléments de formulaire HTML**
 - Des éléments – `input`, `textarea`, `select` – sont placés dans le formulaire
 - Le navigateur construit une chaîne avec les noms et valeurs de ces éléments
 - Celle-ci est envoyée au serveur avec la requête lors de l'envoi du formulaire
- **Avec MVC, l'attribut `action` du formulaire est le chemin d'une méthode d'action**
 - Chaque élément de saisie doit avoir un attribut `name`
 - Le formulaire est envoyé avec un élément `input` de type `submit`



Les éléments HTML peuvent aussi avoir un attribut `id`, utilisé côté client par les CSS et JavaScript. `name` et `id` ont souvent la même valeur, mais peuvent être différents

CSS = Cascading Style Sheets

Donner des valeurs par défaut

➤ On peut définir des valeurs par défaut dans le modèle

- Ou dans des propriétés du ViewBag
- Les valeurs sont définies dans le contrôleur
- Ou mieux, avec un appel au niveau service

```
var model = new TodoItem { Title = "Meeting", Notes = "At 5pm" };  
return View(model);
```

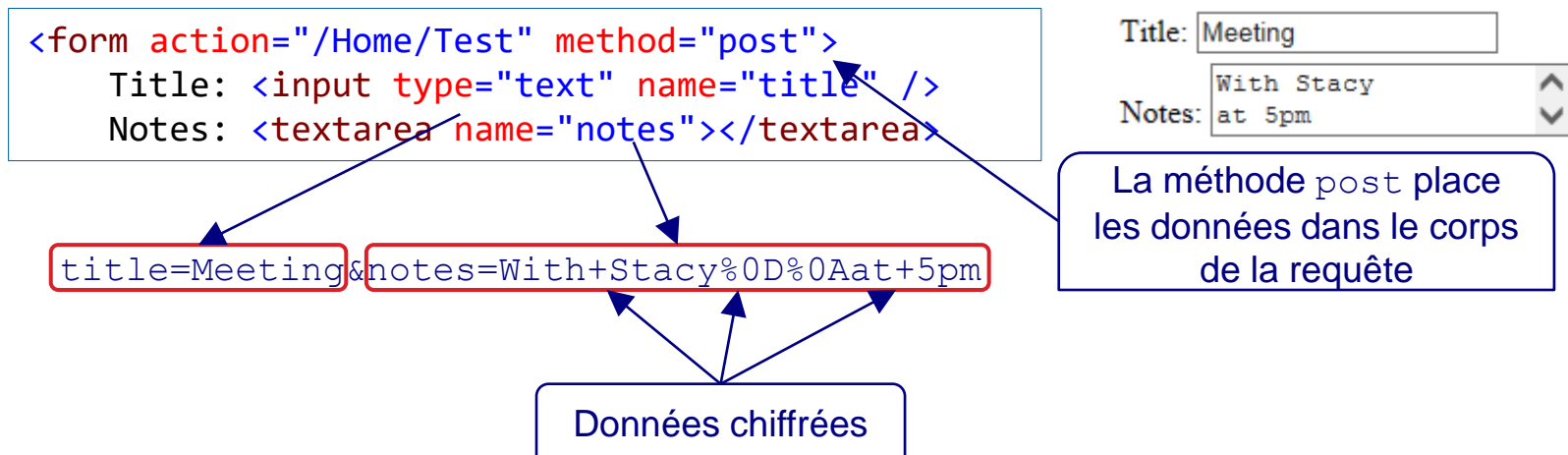
C#

Valeurs par défaut

```
<form action="/Home/Index">  
  Title: <input type="text" name="Title" value="@Model.Title" /><br />  
  Notes: <textarea name="Notes">@Model.Notes</textarea>  
  <input type="submit" value="OK" />  
</form>
```


Données du formulaire

- **Les données du formulaire sont envoyées avec la requête**
 - Avec la syntaxe suivante : `name1=value1&name2=value2`
 - Si l'élément `form` a un attribut de méthode `get`, les données sont placées dans l'URL
 - Avec un attribut de méthode `post`, elles sont placées dans le corps de la requête HTTP
- **Sur le serveur, l'application peut extraire les données de la requête**
 - Le dictionnaire `Form` contient les données saisies
 - Le HTML est encodé pour les caractères spéciaux tels que les espaces



Traiter les données du formulaire dans une méthode d'action

➤ L'obtention des données est simplifiée avec MVC

- Inutile d'explorer le HTTP ou de décoder des données HTML
- MVC mappe les éléments de saisie aux paramètres de la méthode d'action
 - Le nom d'un élément est mappé au nom d'un paramètre
- Peut aussi mapper des objets complexes comme on le verra plus tard

```
<form action="/Home/Index" method="post">  
  Title: <input type="text" name="title" />  
  Notes: <textarea name="notes"></textarea>
```

Title:

Notes:

```
[HttpPost]  
public ActionResult Index(string title, string notes)  
{  
  
}
```

C#

Un formulaire, deux méthodes d'action

➤ Deux méthodes d'action sont généralement associées à un formulaire

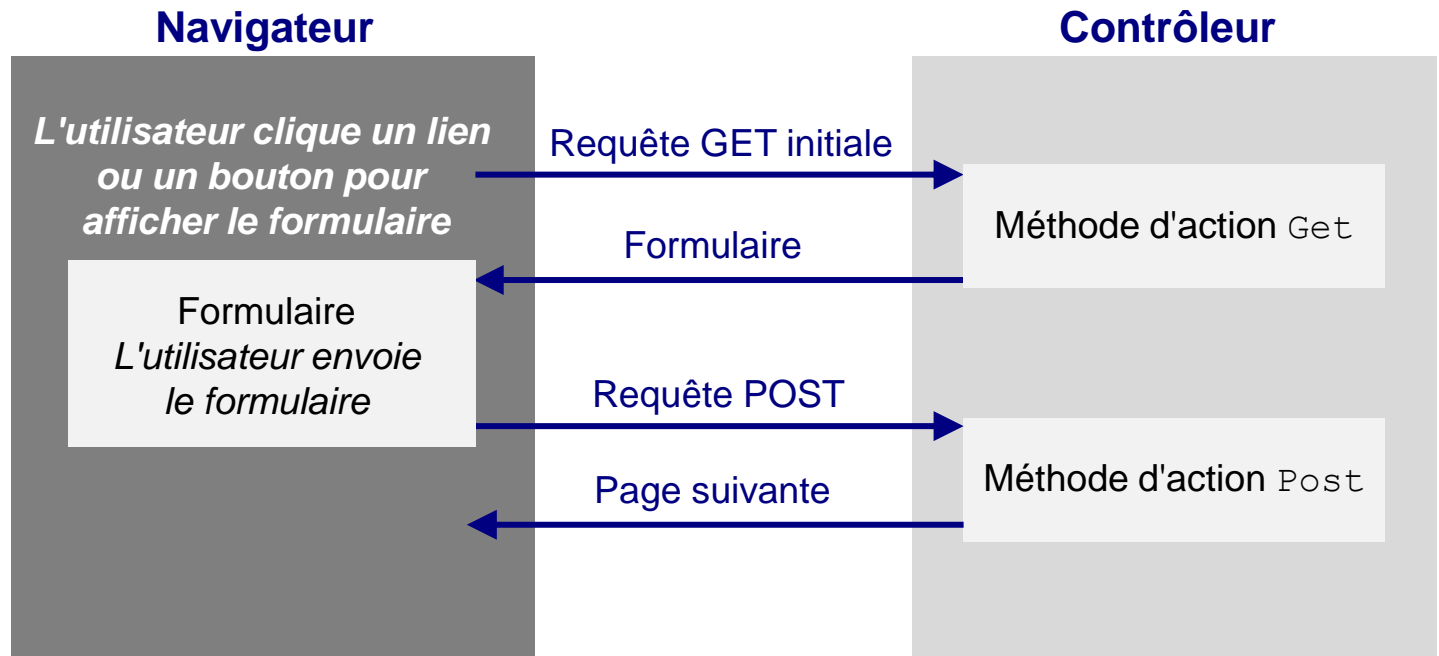
- La première traite une requête GET afin d'afficher les données
- La seconde traite la requête POST après la soumission du formulaire
- On peut utiliser le même nom pour les deux
 - Le compilateur distingue les méthodes par leurs signatures
 - Le routage les distingue à l'aide de l'attribut `HttpPost` de la seconde méthode

```
public ActionResult Index()  
{  
    var model = ...;  
    return View(model);  
}  
  
[HttpPost]  
public ActionResult Index(ListingItemModel model)  
{  
    return View("Confirm", model);  
}
```

C#

Naviguer vers une autre vue
une fois le traitement terminé

Un formulaire, deux méthodes d'action – Exemple



➤ Démo : un formulaire HTML

- Solution dans `Demo\Chapter 5\HTMLForm`
- Le contrôleur `Home` a deux méthodes d'action `Index`
 - La vue `Index` contient un formulaire HTML
 - La vue `Confirm` affiche les données saisies
- Exécutez l'application et cliquez sur **OK**
 - Les données du formulaire sont affichées dans la fenêtre de sortie de Visual Studio
- Changez la valeur de `method` dans l'élément `form` en `get`. Exécutez et cliquez sur **OK**
 - La vue `Index` est affichée à nouveau
 - Les données du formulaire sont dans l'URL

1. Ouvrez le point de départ Do Now 5a dans Do Nows\Do Now 5A-Starting Point et générez la solution
 - Solution de l'exercice précédent avec des fonctionnalités d'édition
2. Ouvrez `TodoListController`
 - Il a deux méthodes d'action `ItemEdit`

Quel est le type du paramètre de la seconde méthode `ItemEdit` ?

3. Ouvrez la vue `TodoList\ItemEdit`
 - Elle implémente un formulaire avec plusieurs champs du modèle

Quel est le type de modèle de la vue ? _____

4. Exécutez l'application et allez jusqu'au formulaire `ItemEdit`
 - Un formulaire est affiché, mais les données ne sont pas encore enregistrées

Formulaires et saisie utilisateur

- **Formulaires HTML et MVC**
- ➡ **Simplifier la génération du HTML avec des aides**
- **Développer des aides HTML personnalisées**
- **Méthodes d'aide pour l'affichage et l'édition**
- **Validation des données**

Aides HTML

- **Une aide HTML est une méthode qui génère du HTML**
 - Comme l'aide `ActionLink`
 - Contribue à la génération de HTML propre
 - Les attributs `name` et les propriétés du modèle sont synchronisés
- **`Html.BeginForm` génère l'élément `form`**
 - A plusieurs surcharges
 - Avec un bloc `using`, elle génère aussi l'élément de fermeture du `form`
 - Permet d'éviter l'utilisation de la méthode `EndForm`

```
@using (Html.BeginForm())  
{  
    Title: <input type="text" ...  
}
```

Génère

```
<form action="/Home/Index" method="post">
```


Surcharges de BeginForm

- **BeginForm sans paramètre génère un formulaire par défaut**
 - La méthode est POST
 - L'action est la méthode d'action courante
 - Des surcharges permettent de définir le contrôleur, l'action, ou d'ajouter des attributs HTML

```
Html.BeginForm("Edit", "Category", FormMethod.Get,  
    new { id = "myForm", @class = "formClass" })
```

Génère

Notez l'utilisation de @class, car
class est un mot réservé en C#

```
<form action="/Category/Edit" class="formClass" id="myForm" method="get">
```

Aides pour les éléments de saisie

➤ Plusieurs aides HTML génèrent des éléments de saisie

- Ont des surcharges pour ajouter des attributs ou définir une valeur par défaut
- Par exemple

```
<input type="text" id="title" name="title" value="@Model.Title" />
```

peut être remplacé par

```
@Html.TextBox("title", Model.Title)
```

➤ Le premier paramètre est la valeur des propriétés id et name

- Peuvent être redéfinies avec un objet anonyme
- Le deuxième paramètre est généralement la valeur par défaut

```
@Html.TextBox("title", Model.Title, new { id = "idTitle" })
```

C#



```
<input id="idTitle" name="title" type="text" value="Meeting" />
```

Aides HTML pour la saisie

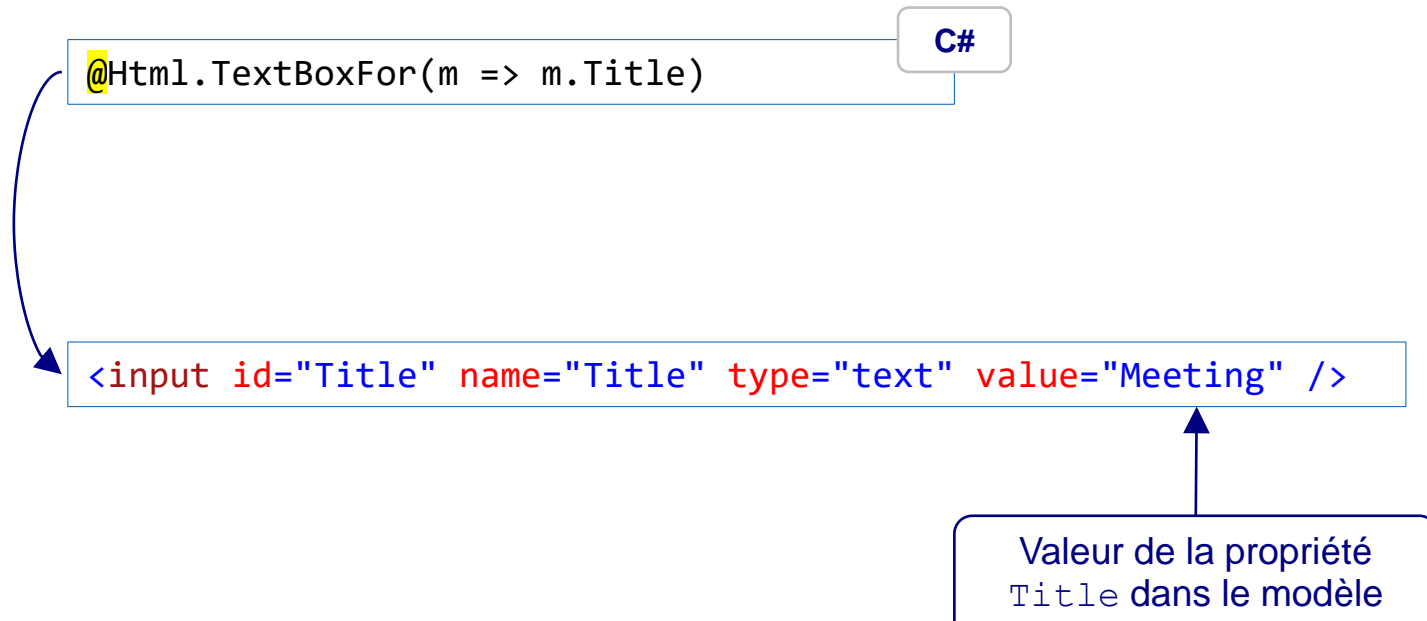
➤ Principales aides HTML pour la saisie

- Les attributs `name` et `id` sont omis pour simplifier

Méthode d'aide	Élément HTML
TextBox	<code><input type="text" /></code>
TextArea	<code><textarea></textarea></code>
Password	<code><input type="password" /></code>
CheckBox	<code><input type="checkbox" /></code>
RadioButton	<code><input type="radio" /></code>
Hidden	<code><input type="hidden" /></code>

Aides HTML fortement typées

- **Les aides HTML pour la saisie ont des versions fortement typées**
 - Le nom du champ est indiqué avec une expression lambda
 - IntelliSense affiche le nom du champ, ce qui évite les erreurs de frappe
 - Même noms que les versions non typées, avec un suffixe `For`
 - Utiliser la version fortement typée autant que possible



Aides HTML pour les labels

➤ Label et LabelFor affichent un élément HTML label

- Affiche le nom de la propriété associée du modèle par défaut
- On peut le modifier avec l'attribut `Display` dans le modèle
 - Défini dans `System.ComponentModel.DataAnnotations`
 - Peut aussi être utilisé avec d'autres aides

```
[Display(Name="Please enter title:")]  
public string Title { get; set; }
```

C#

```
@Html.LabelFor(m => m.Title)
```

```
<label for="Title">Please enter title:</label>
```

for associe le label
au textbox

➤ DisplayName et DisplayNameFor n'affichent que le texte, sans élément HTML Label

Listes et listes déroulantes HTML

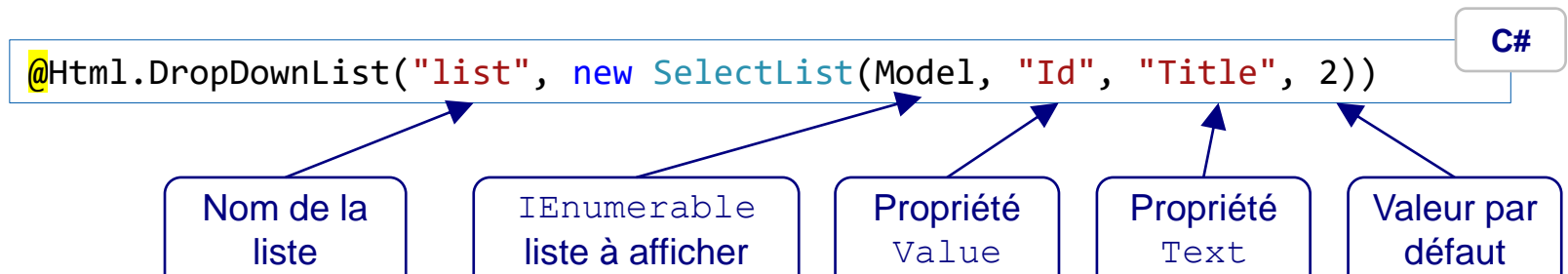
- **Les listes HTML utilisent un élément `select`**
 - Contient une liste d'éléments `option`
 - Chaque `option` a un attribut `value` et le texte affiché
 - Une liste (listbox) est affichée si l'attribut `size` est supérieur à 1, une liste déroulante (drop-down list) sinon
- **On peut construire une liste manuellement dans une vue Razor avec une boucle `foreach`**

```
<select>
    @foreach (var item in Model)
    {
        <option value="@item.Id">@item.Title (@item.Notes)</option>
    }
</select>
```

Aides HTML de liste

➤ Deux aides HTML génèrent des listes HTML

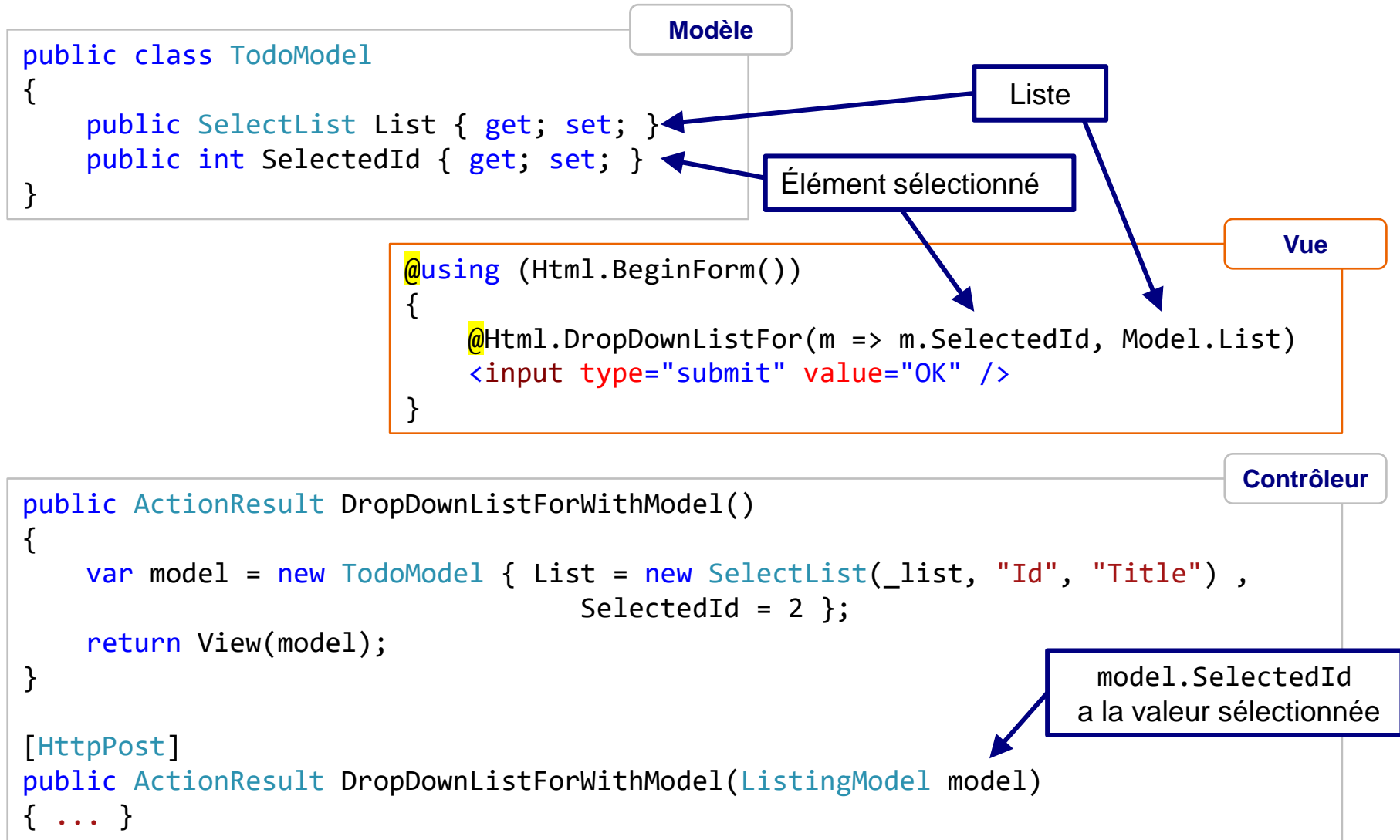
- DropDownList et ListBox ou leurs versions fortement typées
- Prennent un objet SelectList comme paramètre
- SelectList a plusieurs surcharges et encapsule la liste à afficher



➤ Deux options pour construire la SelectList

- Dans la vue, comme dans l'exemple ci-dessus
- Dans le contrôleur ou le niveau service, dans une propriété du modèle ou dans le ViewBag

DropDownListFor avec SelectList dans un modèle



➤ **Démo : des aides de liste HTML**

- Solution dans `Demo\Chapter 5>ListHelpers`
- Cliquez sur **HTML DropDown** pour afficher un élément `select` généré manuellement
- Cliquez sur **DropDownList** pour afficher une liste déroulante générée avec une aide `DropDownList`
- Cliquez sur **DropDownListFor with model** pour afficher une liste déroulante avec une aide `DropDownListFor`

Télécharger des fichiers

- **Le téléchargement de fichiers se fait avec un élément `input type="file"`**
 - Doit être dans un formulaire utilisant la méthode POST et un attribut `enctype`

```
@(using (Html.BeginForm("AddPhoto", "Home", FormMethod.Post,
    new { enctype = "multipart/form-data" })))
{
    <input type="file" name="file" />
    <input type="submit" value="Upload" />
}
```

- **La méthode d'action reçoit le fichier dans un paramètre**
 - De type `HttpPostedFileBase`
 - Également disponible dans la propriété `Request.Files`

```
[HttpPost]
public ActionResult AddPhoto(HttpPostedFileBase file)
{
}
```

Traiter des fichiers téléchargés

➤ Propriétés et méthodes de `HttpPostedFileBase`

Propriété ou méthode	Description
<code>ContentLength</code>	Taille du fichier en octets
<code>ContentType</code>	Type MIME du fichier
<code>FileName</code>	Nom complet du fichier sur le client
<code>InputStream</code>	Flux ayant vers le contenu du fichier
<code>SaveAs</code>	Méthode pour enregistrer le fichier sur le serveur

➤ L'objet `Server` du contrôleur a une méthode `MapPath`

- Renvoie le chemin physique sur le serveur d'un chemin logique

```
Server.MapPath("~/Photos");
```

➤ Démonstration : le téléchargement de photos vers un site Web

- Solution dans `Demos\Chapter 5\Photos`
- Exécutez l'application. Cliquez sur **Browse** pour choisir une photo
 - Il existe des photos dans `Database\Photos`
- Cliquez sur **Upload** pour télécharger la photo sélectionnée
- Cliquez sur **Show All Files** dans Solution Explorer
 - Les photos téléchargées se trouvent dans le dossier `Photos` du projet
- Examinez la classe `HomeController` et la vue `Index`

Exercice 5.1 : Remplacer des éléments HTML par des aides HTML

Formulaires et saisie utilisateur

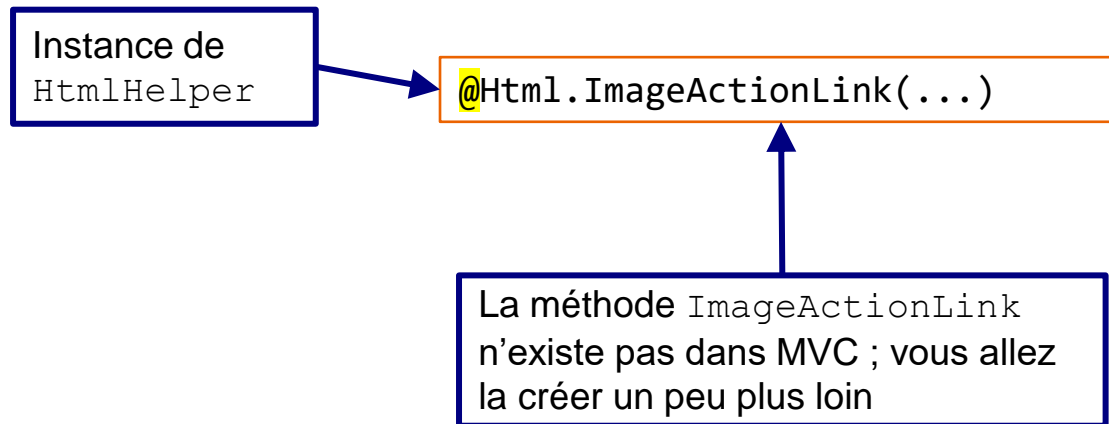
- **Formulaires HTML et MVC**
- **Simplifier la génération du HTML avec des aides**
- ➡ **Développer des aides HTML personnalisées**
- **Méthodes d'aide pour l'affichage et l'édition**
- **Validation des données**

Méthodes d'extension

- **La plupart des aides HTML MVC sont des méthodes d'extension**
 - Des méthodes qui étendent la classe `HtmlHelper`
 - On peut écrire des aides HTML personnalisées
- **Une méthode d'extension semble faire partie d'une classe**
 - On peut l'ajouter à des classes existantes
 - Particulièrement utile si le code source de la classe n'est pas disponible
 - Par exemple, les classes du framework .NET
- **Il faut la définir dans une classe `static`**
 - Le premier paramètre de la méthode est l'objet auquel elle s'applique
 - Dans le client, ajouter un `using` pour l'espace de noms où elle est définie
- **Une aide HTML personnalisée doit renvoyer une chaîne de caractères**
 - Du HTML inséré dans la vue
 - Renvoyer un `HtmlString` afin que Razor n'encode pas la chaîne
 - On peut utiliser la classe `TagBuilder` pour construire le HTML

Une méthode d'extension pour une aide HTML

- **Méthode qui construit un lien constitué d'un texte et d'une image**
 - La méthode standard `ActionLink` génère un élément `<a>` avec du texte
 - L'aide personnalisée `ImageActionLink` générera un élément `<a>` avec une image et du texte
- **Une méthode d'extension s'utilise comme si elle faisait partie de la classe qu'elle étend**
 - La propriété `Html` des vues est de type `HtmlHelper`



Une méthode d'extension pour une aide HTML (suite)

La classe et la méthode
doivent être `static`

```
public static class MyExtensions
{
    public static HtmlString ImageActionLink(this HtmlHelper html, ...
    {
    }
}
```

C#

Le premier paramètre
est décoré avec `this`

Classe TagBuilder

➤ Classe simple pour construire des éléments HTML

- Le constructeur reçoit le nom de l'élément

```
var link = new TagBuilder("a");
```

C#

- Méthodes et propriétés pour personnaliser l'élément

Méthode / Propriété	Description
AddCssClass	Ajouter des valeurs à l'attribut <code>class</code>
MergeAttribute	Ajouter un attribut à l'élément
SetInnerText	Texte entre les éléments d'ouverture et de fermeture (encodé)
Attributes	Dictionnaire des attributs
InnerHtml	Texte entre les éléments d'ouverture et de fermeture (non encodé)
ToString	Renvoie la chaîne HTML de l'élément

Classe UrlHelper

➤ Construite une URL à partir de valeurs de route

- On peut y accéder avec la propriété `Url` des vues
- Le constructeur reçoit une contexte de requête

```
var urlHelper = new UrlHelper(html.ViewContext.RequestContext);
```

C#

Le fait d'étendre la classe `HtmlHelper` donne accès à ses membres

- Les méthodes renvoient l'URL

Méthode	Description
Action	Plusieurs surcharges pour générer une URL à partir de l'action, du contrôleur, des valeurs de routage
RouteUrl	Génère une URL à partir d'un nom de route
Content	Convertit un chemin virtuel en chemin absolu

Construire un lien

```
var link = new TagBuilder("a");  
link.Attributes.Add("href",  
    urlHelper.Action(actionName, controllerName, routeValues));  
link.SetInnerText(linkText);  
// Use with: link.ToString()
```

Paramètres passés à la
méthode d'extension

```
graph TD
    A[Paramètres passés à la méthode d'extension] --> B["@Html.MyActionLink(listing.Title, 'List', 'ListItem', new { id = listing.Id })"]
    B --> C["<a href='/ListItem/List/1'>Food</a>"]
```

```
@Html.MyActionLink(listing.Title,  
    "List", "ListItem", new { id = listing.Id })
```

Vue

```
<a href="/ListItem/List/1">Food</a>
```

1. Ouvrez le point de départ Do Now 5b à
`Do Nows\Do Now 5b-Starting Point`
2. Exécutez l'application et cliquez sur le lien `Categories`. Notez que le lien fonctionne comme prévu
3. Ouvrez la classe `MyExtensions` dans le dossier `Extensions` du projet web
4. Essayez de comprendre le code. Demandez à votre formateur le cas échéant
5. Allez à la vue `CategoryList` du dossier `Views\Category`
6. Supprimez les lignes de code qui génèrent le lien et l'image
7. Dé-commentez les deux dernières lignes qui appellent `ImageActionLink`
8. De retour dans l'application en exécution, allez à la page `My Lists`. Elle doit fonctionner comme précédemment
9. Regardez le code source de la page pour voir les liens HTML générés

Classes d'aide

- **Étendre la classe `HtmlHelper` permet d'avoir accès à ses propriétés et méthodes**
 - Telles que `ViewData`, `ViewBag`, ou `ViewContext`
 - Et aux propriétés de `ViewContext` , comme `HttpContext` ou `RequestContext`
- **MVC a aussi quelques classes d'aide utiles**
 - `ExpressionHelper` obtient le texte d'une expression
 - `ModelMetadata` donne accès aux métadonnées du modèle, telles que `DisplayName`
 - Utilisées dans la section bonus du prochain exercice
- **La classe `HtmlString` encapsule une chaîne**
 - Indique au moteur Razor de ne pas encoder la chaîne
 - Depuis .NET 4.0
 - `MvcHtmlString` est équivalent, à utiliser avec les application avant .NET 4.0

Exercice 5.2 : Développer une aide HTML `LabelTextBox`

Formulaires et saisie utilisateur

- **Formulaires HTML et MVC**
- **Simplifier la génération du HTML avec des aides**
- **Développer des aides HTML personnalisées**
- ➡ **Méthodes d'aide pour l'affichage et l'édition**
- **Validation des données**

Interface utilisateur pilotée par le modèle

- **MVC peut générer la vue et les éléments de saisie à partir du modèle**
 - Sélectionne les éléments HTML selon le type de donnée des propriétés
 - Peut être personnalisé avec des attributs
 - Utilisé de cette façon, un modèle peut être appelé un modèle de vue
- **Utilise les méthodes d'aide `Editor` et `Display`**
 - Ou leurs versions fortement typées `EditorFor` et `DisplayFor`
 - L'élément HTML généré dépend du type de la propriété
 - `EditorForModel` et `DisplayForModel` génèrent des éléments pour tout le modèle
- **`Editor` et `EditorFor` génèrent des éléments de saisie HTML**
 - Le type est `text` par défaut
 - Peut être `checkbox` pour les propriétés booléennes
 - Peut être `number` ou `datetime` sur les navigateurs compatibles HTML5
- **`Display` et `DisplayFor` génèrent du texte**
 - Ou des case à cocher inopérantes pour les propriétés booléennes
 - Tout peut être personnalisé

Éditer et afficher l'ensemble du modèle

- **EditorForModel** génère un champ de saisie pour chaque propriété du modèle
 - `DisplayForModel` affiche chaque propriété du modèle
 - On peut cacher ou mettre en forme les propriétés avec des attributs
- **Des attributs du modèle définissent l'affichage de chaque propriété**
 - Définis dans les espaces de noms `System.ComponentModel.DataAnnotations` et `System.Web.Mvc`

Attribut	Description
<code>ScaffoldColumn</code>	Pas de HTML généré pour la propriété si le constructeur reçoit la valeur <code>false</code>
<code>HiddenInput</code>	Génère un élément <code>input type="hidden"</code> pour les méthodes <code>Editor</code> . Aucun HTML généré avec les méthodes <code>Display</code> si <code>DisplayValue</code> vaut <code>false</code>
<code>DisplayFormat</code>	Définit la mise en forme de la sortie. <code>NullDisplayText</code> est le texte affiché si la valeur est nulle. <code>DataFormatString</code> est une chaîne de mise en forme

Utiliser les attributs Editor et Display

C#

```
[HiddenInput(DisplayValue = false)]
public int Id { get; set; }

[DisplayFormat(NullDisplayText = "-", DataFormatString="{0:D}")]
[Display(Name = "Due Date")]
public DateTime? DueDate { get; set; }

[ScaffoldColumn(false)]
public int? LookAndFeelId { get; set; }
```

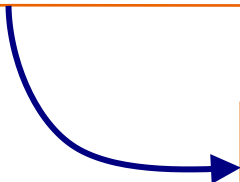
```
@Html.DisplayForModel()
```

Editor et EditorFor

➤ EditorFor remplace des méthodes de saisie HTML plus spécifiques

- Le HTML généré est piloté par le type de donnée
- Peut être personnalisé avec des attributs

```
@Html.TextBoxFor(m => m.Title)  
@Html.CheckBoxFor(m => m.Done)
```



```
@Html.EditorFor(m => m.Title)  
@Html.EditorFor(m => m.Done)
```

Personnaliser le HTML généré

- **Editor et EditorFor génèrent du HTML selon le type de donnée de la propriété**
 - La plupart des types génèrent des `input` de type `text`
 - Peut être personnalisé avec un attribut `DataType`
 - Ou un attribut `UIHint`
- **L'attribut `DataType` définit le modèle utilisé pour générer le HTML**
 - MVC a des modèles d'affichage et d'édition pour chaque type de donnée
 - On peut aussi créer des modèles personnalisés
 - `DataType` a des classes dérivées
 - Les deux attributs suivants sont identiques

```
[DataType(DataType.EmailAddress)]
```

– et

```
[EmailAddress]
```

Attribut DataType

➤ Principales valeurs de l'énumération DataType

Valeur de DataType	Description
MultilineText	Génère un élément HTML <code>textarea</code>
Password	Génère un élément <code>input</code> de type <code>password</code>
Date, Time	Affiche la partie date ou heure d'une propriété <code>DateTime</code>
EmailAddress	Génère un élément lien <code>mailto:</code>
Url	Génère un élément lien

➤ D'autres valeurs génèrent du code de validation

- `CreditCard`, `PhoneNumber`, `PostalCode`...
- Valident le format, pas la valeur

La validation est présentée
dans la section suivante

Modèles d'affichage et d'édition personnalisés

➤ On peut définir des modèles d'affichage et d'édition personnalisés

- Vues partielles situées dans les dossiers `DisplayTemplates` ou `EditorTemplates` du dossier `Shared`
- Le type du modèle est le type de la propriété
- Associé implicitement à la propriété grâce au nom du type
 - `Boolean.cshtml` traite toutes les propriétés `bool/Boolean`
 - Remplace le modèle intrinsèque `Boolean`
- Ou explicitement avec un attribut `UIHint` ou `DataType`

Exercice 5.3 : Utiliser les modèles d'édition et d'affichage

Formulaires et saisie utilisateur

- **Formulaires HTML et MVC**
- **Simplifier la génération du HTML avec des aides**
- **Développer des aides HTML personnalisées**
- **Méthodes d'aide pour l'affichage et l'édition**
- ➔ **Validation des données**

Validation client et serveur

- **Il faut vérifier la saisie utilisateur avant qu'elle n'atteigne la base de données**
 - Doit être fait sur le client et sur le serveur
 - Avec du code JavaScript sur le client
 - Sur le serveur, la validation est faite dans le contrôleur
 - Le niveau métier peut ajouter des vérifications supplémentaires en appliquant des règles métier
- **ASP.NET MVC fait la validation côté serveur pendant la liaison au modèle**
 - Ajoute des attributs aux éléments pour faire la validation sur le client

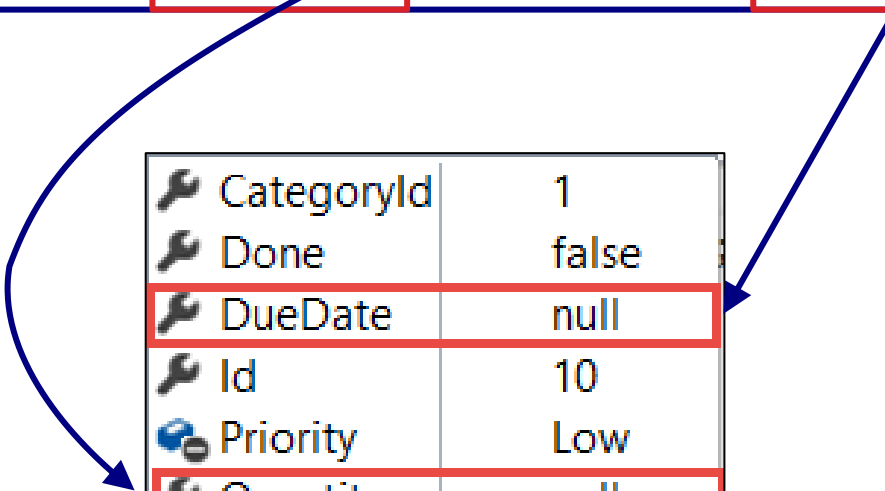
Validation pendant la liaison au modèle

➤ Comparer les données postées et celles du modèle

- Les données qui ne peuvent être converties dans le type cible sont ignorées
- Données envoyées – dans le corps de la requête :

ListingId=1&Title=Cookies&Done=false&Notes=&Quantity=Ten&Priority=Normal&DueDate=today

- Modèle – en paramètre de la méthode d'action :



CategoryId	1
Done	false
DueDate	null
Id	10
Priority	Low
Quantity	null
Style	null
StyleId	null
Title	🔍 "Cakes"

ModelState

➤ ModelState est un dictionnaire

- Keys contient les noms des champs
- Values a une collection Errors
- La propriété IsValid est fausse si au moins une collection Errors n'est pas vide

IsValid	false
Keys	Count = 8
Values	Count = 8
[0]	{System.Web.Mvc.ModelState}
[1]	{System.Web.Mvc.ModelState}
[2]	{System.Web.Mvc.ModelState}
[3]	{System.Web.Mvc.ModelState}
[4]	{System.Web.Mvc.ModelState}
[5]	{System.Web.Mvc.ModelState}
Errors	Count = 1
[0]	{System.Web.Mvc.ModelError}
ErrorMessage	"The value 'Ten' is not valid for Quantity."

➤ ModelState est rempli pendant la liaison du modèle dans la méthode d'action

- Une erreur est ajoutée à chaque échec de liaison
 - Exemple : saisie de texte dans une propriété numérique
 - Ou si la valeur saisie ne correspond pas à certains attributs d'annotation des données

Valider le modèle dans une méthode d'action

- **La validation personnalisée du modèle se fait dans la méthode d'action**
 - La méthode `View` renvoyée ajoute une classe `input-validation-error` à tous les champs en erreur

```
[HttpPost]
public ActionResult ItemEdit(TodoItemModel model)
{
    if (!ModelState.IsValid)
        return View(model);

    _todoItemService.Update(model);
    return RedirectToAction("ItemList", new { id = model.CategoryId });
}
```

Affiche la vue en cas d'erreur

Pas d'erreur

Ajouter un système de validation personnalisé

- **Possibilité d'ajouter un système de validation du modèle personnalisé dans la méthode d'action**
 - `IsValidField` teste la validité des champs
 - L'application peut ajouter des éléments dans le dictionnaire avec la méthode `AddModelError`

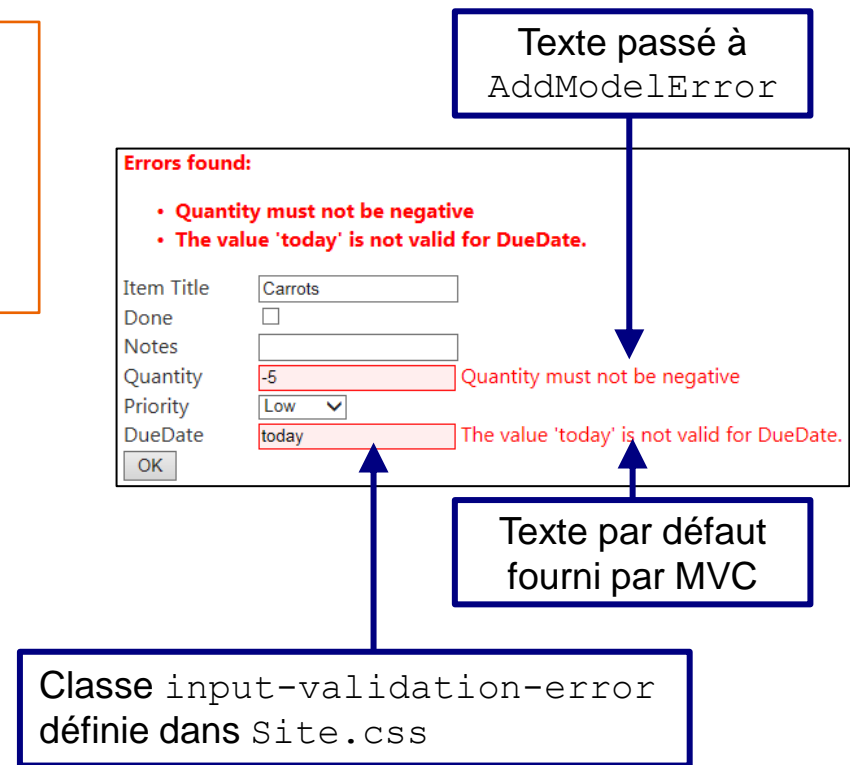
```
if (ModelState.IsValidField("Quantity") &&  
    model.Quantity.HasValue && model.Quantity < 0)  
    ModelState.AddModelError("Quantity", "Quantity must not be negative");
```

Si `Quantity` est validé, vérifier
que la valeur n'est pas négative

Afficher des message d'erreur

- **L'aide HTML `ValidationMessageFor` affiche un message d'erreur**
 - Avec la classe CSS `field-validation-error`
 - En mettre un pour chaque champ susceptible d'avoir une erreur
 - `ValidationSummary` affiche une liste de toutes les erreurs de la page

```
@Html.ValidationSummary("Errors found:")  
@Html.LabelFor(m => m.Quantity)  
@Html.EditorFor(m => m.Quantity)  
@Html.ValidationMessageFor(m => m.Quantity)  
(remaining lines omitted)
```



Contrôler la validation avec des attributs

- **Beaucoup de règles de validation sont encapsulées dans des attributs**
 - Définis dans l'espace de noms `System.ComponentModel.DataAnnotations`
 - La propriété `ErrorMessage` de chaque attribut définit le message d'erreur

Attribut	Description
<code>Required</code>	Le champ ne doit pas être vide
<code>StringLength</code>	Indique les longueur maximum et minimum
<code>Compare</code>	La valeur doit être égale à une autre propriété
<code>Range</code>	Valeur entre deux nombres ou autres types
<code>CreditCard</code>	Format de carte de crédit valide
<code>EmailAddress</code>	Format d'adresse email valide
<code>Phone</code>	Format de numéro de téléphone valide
<code>RegularExpression</code>	Valide un texte avec une expression régulière

Définir un attribut de validation personnalisé

➤ Dans une classe qui dérive de la classe abstraite `ValidationAttribute`

- Implémenter sa méthode `IsValid`
- Reçoit la valeur de la propriété en tant qu'objet
- Renvoyer `true` si la validation réussit, ou `false` si elle échoue

```
public class FutureDateValidationAttribute : ValidationAttribute
{
    public override bool IsValid(object value)
    {
        var date = value as DateTime?;
        return date == null || date > DateTime.Today;
    }
}
```

Valider sur le client

- **La validation MVC se fait par défaut sur le serveur**
 - Le formulaire doit être posté pour que la validation se fasse
 - Pas très réactif
- **ASP.NET MVC s'intègre bien avec la validation jQuery**
 - Ajoute des attributs de validation commençant par `data-val` aux éléments
 - Tels que `data-val-required` ou `data-val-length`
- **Pour autoriser la validation sur le client**
 - Ajouter le bundle `jqueryval` à la page ou la page de disposition
 - Déjà défini dans la classe `BundleConfig`

```
@Scripts.Render("~/bundles/jqueryval")
```

Les bundles sont
présentés dans
le chapitre suivant

Exercice 5.4 : Ajouter la validation à l'étude de cas

Faire construire des vues par Visual Studio

- **Visual Studio peut construire des vues à partir d'un modèle**
 - Crée la vue à partir des propriétés du modèle
 - Sélectionner le modèle dans le dialogue Add View
 - Permet l'ajout de vues pour Create, Delete, Details, Edit, List
- **Visual Studio peut aussi construire un contrôleur à partir d'un modèle**
 - Ajoute des méthodes pour les opérations CRUD
 - Sélectionner le type de construction dans le dialogue Add Controller

Modèle à utiliser

Classe du modèle

View name: ScaffoldDisplay

Template: Details

Model class: TodoItemModel (Todo.Common.Models)

Options:

☐ Create as a partial view

☐ Reference script libraries

☒ Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)

Add Cancel

En anglais,
scaffold = échafaudage

- 1. Ouvrez le point de départ Do Now 5c dans**
`Do Nows\Do Now 5c-Starting Point`
- 2. Ouvrez `TodoListController` dans le projet `Todo.Web`**
 - Une nouvelle méthode d'action a été ajoutée à la fin : `ScaffoldDisplay`
 - Copie de la méthode `ItemDisplay` avec un nom différent
- 3. Cliquez droit sur la méthode `ScaffoldDisplay` et sélectionnez Add View**
- 4. Sélectionnez Details pour Template et `TodoItemModel` dans Model class, puis cliquez sur Add**
- 5. Dans la vue `ItemList`, modifier le `ActionLink` en remplaçant `ItemDisplay` par `ScaffoldDisplay`**
- 6. Exécutez l'application**
 - La vue construite doit s'afficher lors de la sélection d'un élément
- 7. S'il vous reste du temps, ajustez les liens générés dans la vue construite afin qu'ils fonctionnent correctement**
 - Changez aussi le nom de l'action dans la méthode `ItemEdit`

Résumé du chapitre

Dans ce chapitre, nous avons

- Créé des formulaires pour la saisie utilisateur
- Simplifié le codage du HTML avec des aides HTML
- Développé des aides HTML personnalisées
- Validé la saisie et appliqué les règles métier avec des annotations de données

Questions de révision

Écrivez le code Razor qui génère un formulaire dans une vue

Citez quelques aides HTML pour des champs de saisie

Quelle classe utiliser pour générer une liste déroulante ?

Quelle fonctionnalité du langage est utilisée pour créer une aide HTML personnalisée ?

Comment savoir si la liaison au modèle a réussi ?

Écrivez le code qui limite la saisie d'une propriété `Name` à 20 caractères

VI. (216 à 261)

Fonctionnalités côté client

Objectifs du chapitre

Dans ce chapitre, nous allons

- Construire des application hautement interactives avec jQuery
- Développer des éléments d'interface utilisateur côté client avec jQueryUI
- Optimiser le chargement des pages avec le regroupement et la compression
- Établir une communication entre le client et le server avec Ajax
- Rendre une application mobile
- Préparer les applications aux marchés internationaux
- Afficher des données à l'aide de grilles et de graphiques

Fonctionnalités côté client

➡ **Présentation de jQuery**

- **Ajax**
- **Applications mobiles**
- **Applications internationales**
- **WebGrid et Chart**

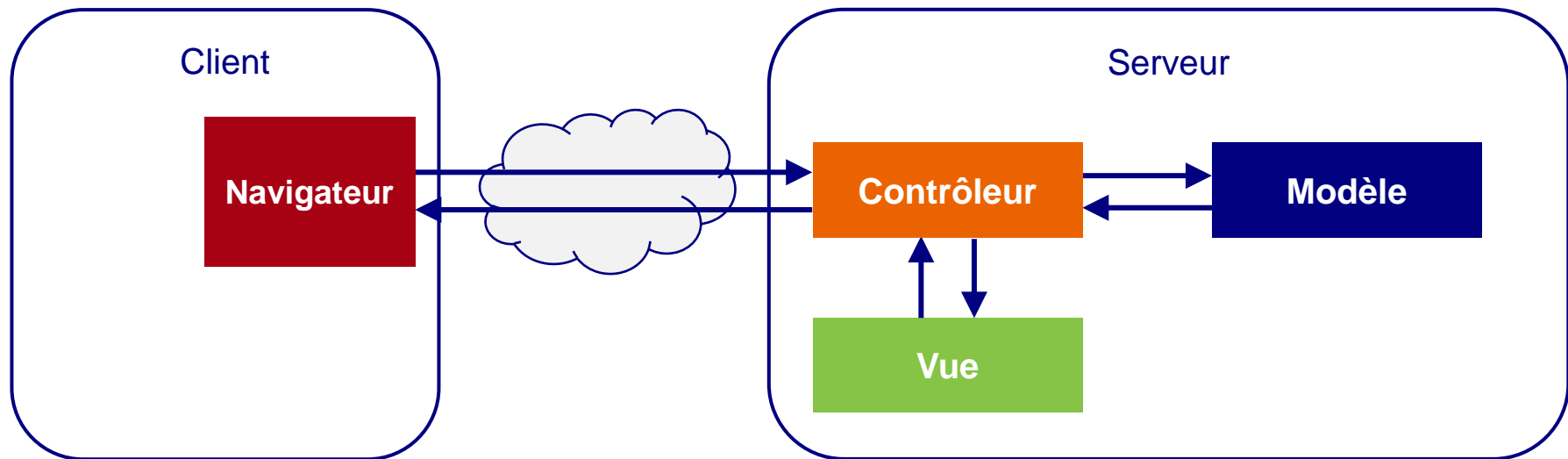
Technologies côté serveur et côté client

➤ ASP.NET MVC est une technologie côté serveur

- HTML, CSS, images sont envoyés par le serveur
- Le navigateur affiche les pages
- Un clic sur un lien ou un bouton de formulaire envoie une requête au serveur

➤ Il faut développer côté client pour que l'application soit plus réactive

- IU dynamique : menus, animations, sélection de dates...
- Obtenir des données du serveur sans actualiser toute la page
- Fait dans le navigateur, avec du code JavaScript



JavaScript et jQuery

➤ **JavaScript est un langage dynamique de type C**

- Implémenté dans tous les navigateurs modernes
- Le code JavaScript est inclus dans les pages HTML
- S'exécute dans le navigateur
- A accès au DOM (Document Object Model)
 - Représentation hiérarchique du document HTML

➤ **jQuery masque la plupart des spécificités des navigateurs**

- Bibliothèque JavaScript légère, open-source incluse dans Visual Studio
- Téléchargement, mises à jour et documentation à `jquery.com`
 - Peut aussi être mis à jour via NuGet

➤ **Principales fonctionnalités de jQuery**

- Manipuler le HTML du DOM : éléments, attributs, classes, etc.
- Traiter le JSON – syntaxe de description de données compacte, alternative à XML
- Séparer le comportement du contenu avec du JavaScript non intrusif
- Des milliers de plug-ins sont disponibles pour ajouter des fonctionnalités

Présentation de jQuery

- **jQuery est publié sous la forme d'un fichier .js**
 - Il faut l'inclure sur une page pour l'utiliser
 - Généralement fait dans une vue de disposition
 - Ajouté via des regroupements dans la plupart des modèles de projets MVC
- **La fonction `jQuery` est au cœur de l'API de jQuery**
 - `jQuery` passe un sélecteur à une expression
 - Retourne un objet `jQuery`, ensemble des éléments correspondants
 - Écrit sous la forme `jQuery()` ou `$()`
- **Le code jQuery ne doit s'exécuter que quand tout le HTML est chargé**
 - L'événement `jQuery.ready()` s'exécute quand le DOM est chargé
 - Écrit sous la forme `$(function () { ... });`



Votre code
jQuery ici

Sélecteurs jQuery

- **Un sélecteur jQuery fait référence à un ou plusieurs éléments du DOM**
 - Basé sur les sélecteurs des CSS
 - La syntaxe est `$ ('selector')`
- **Il existe de nombreux types de sélecteurs, en voici quelques-uns**

Sélecteur	Description
<code>\$ ('button')</code>	Tous les éléments <code>button</code> de la page
<code>\$ ('#btn')</code>	L'élément dont l'attribut <code>id</code> vaut <code>btn</code>
<code>\$ ('.cancel')</code>	Tous les éléments ayant un nom de classe <code>cancel</code>
<code>\$ ('li.opt input')</code>	Tous les éléments <code>input</code> situés dans un élément <code>li</code> dont le nom de classe est <code>opt</code>

Utiliser jQuery

➤ Afficher un message quand l'utilisateur clique sur un bouton

- Avec jQuery, le code est dans une section différente de la page HTML
- Ou mieux, dans un fichier `.js` référencé par la page
- jQuery est non intrusif

```
<input type="button" value="Click me" id="button" />
```

Section `script`
définie dans la vue
de disposition

Appelée quand le
DOM est chargé

```
@section scripts {
```

```
<script>
```

```
$(function () {
```

```
    $('#button').click(function () {  
        alert('Thanks!');
```

```
    });
```

```
});
```

```
</script>
```

```
}
```

Ajoute un gestionnaire
d'événement `click`

Le paramètre est une
fonction appelée quand
l'événement est généré

jQuery en action

➤ Démo : une page avec du code jQuery

- Démarrez l'application
- Sélectionnez jQuery ou Animation
- Cliquez sur une couleur

```
<ul id="colors">
  <li><a data-color="red" href="#">Red</a></li>
  <li><a data-color="green" href="#">Green</a></li>
  <li><a data-color="blue" href="#">Blue</a></li>
</ul>
```

```
<div id="divDemo" />
```

```
$(function () {
  $('#colors a').click(function () {
    var color = $(this).data('color');
    $('#divDemo').removeClass().addClass(color);
  });
});
```

Notez l'API fluide : chaque méthode renvoie l'objet sur lequel elle est appelée

Regroupement et compression

- **Les pages Web nécessitent souvent plusieurs fichiers JavaScript et CSS**
 - Chacun est inclus dans un élément `<script>` ou `<link>`
 - Une requête HTTP est faite pour chacun
 - Ralentit le chargement de la page, surtout si le réseau est lent
- **Le regroupement et la compression sont nouveaux dans .NET 4.5**
 - Il fallait précédemment utiliser des compléments tels que Combres
 - Le regroupement combine plusieurs fichiers `.js` ou `.css` en un seul
 - La compression retire les espaces, tabulations, changements de ligne et les commentaires
 - Minimise aussi les noms de variables
- **Ne se produisent que dans l'environnement de production**
 - Si l'attribut `debug` de l'élément `compilation` de `Web.config` est `false`

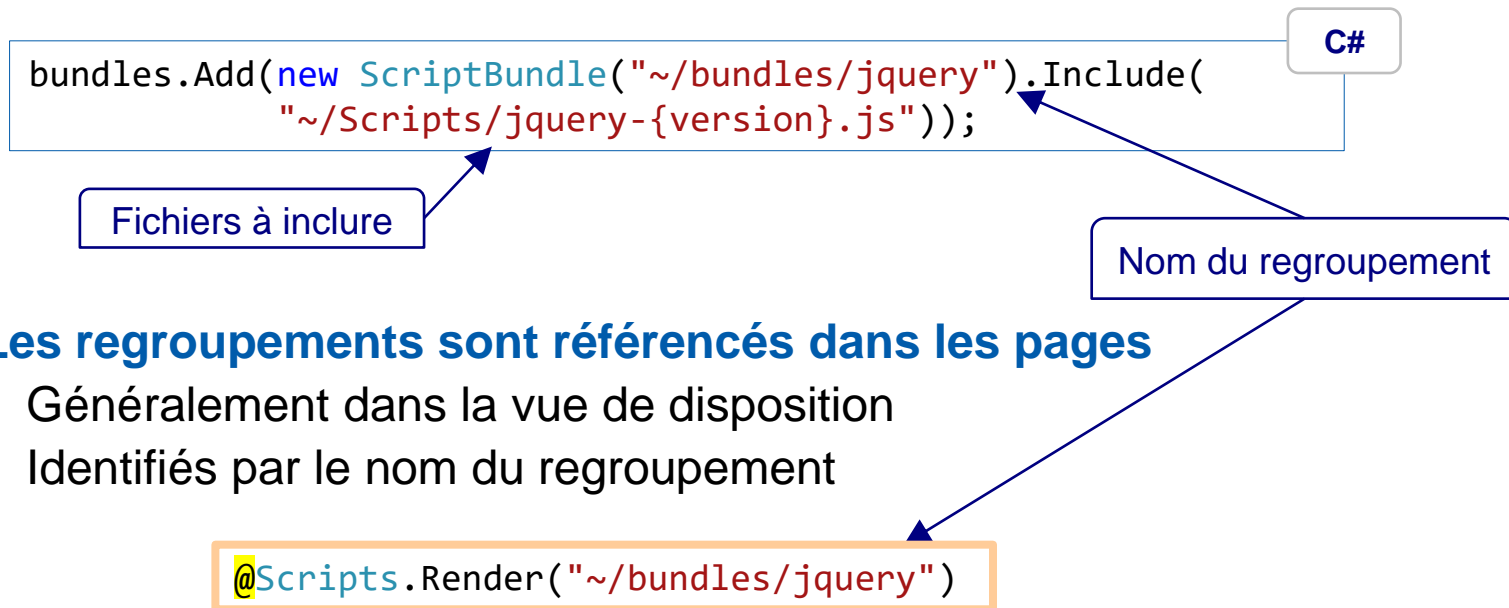
Regroupement = bundling – Compression = minification

Configurer le regroupement

➤ Le regroupement est défini dans l'espace de noms

`System.Web.Optimization`

- La classe `BundleTable` a une collection statique `Bundles`
- Contient une collection d'objets `ScriptBundle` et `StyleBundle`
- La classe `BundleConfig` dans le dossier `App_Start` inscrit les regroupements
- Appelée depuis `Application_Start`, dans `Global.asax`



➤ Les regroupements sont référencés dans les pages

- Généralement dans la vue de disposition
- Identifiés par le nom du regroupement

1. **Ouvrez la solution à Demo\Chapter 6\Bundling**
 - Solution de l'exercice précédent, légèrement modifiée
 2. **Ouvrez la vue `_ViewStart` dans le dossier Views**
 - Elle définit `_LayoutWithoutBundling` comme vue par défaut
 3. **Ouvrez `_LayoutWithoutBundling` dans le dossier Views\Shared**
 - N'utilise pas le regroupement, mais des éléments `link` et `script` classiques
 4. **Exécutez-le dans Chrome avec `<Ctrl><F5>` et appuyez sur `<F12>` pour afficher les outils du développeur**
 5. **Cliquez sur l'onglet Network, puis cochez Disable cache in the toolbar**
 6. **Dans votre application, cliquez sur l'élément de menu Home**
- **Quelle est la taille du fichier de script jQuery ? _____**

7. Changez la disposition dans `_ViewStart` :

```
Layout = "~/Views/Shared/_LayoutWithBundling.cshtml"
```

8. Ouvrez `_LayoutWithBundling` et examinez les différences avec la vue `_LayoutWithoutBundling`

9. Dans `Web.config`, changez `debug` en `false` dans l'élément `compilation`

- Prenez bien le `Web.config` situé à la racine de l'application

10. Enregistrez tous les fichiers et actualisez l'affichage dans le navigateur

Quelle est la taille du regroupement jQuery ? _____

11. Cliquez sur l'onglet Sources dans la fenêtre des outils de développement

12. Développez les regroupements et cliquez sur jQuery pour voir la source compressée

jQuery UI

➤ Il existe de nombreux plug-ins jQuery

- `plugins.jquery.com` en a une liste partielle
- jQuery UI est très populaire et fait partie de Visual Studio
- Bootstrap a également de nombreuses fonctionnalités et plug-ins
 - Fonctionne mieux avec les navigateurs récents compatibles CSS3

➤ jQuery UI permet d'enrichir l'interface utilisateur

- Glisser et déposer des éléments sur une page
- Construire des animations
- Nombreux widgets tels que Dialog, Menu, Datepicker, Tabs ...

➤ Démo : jQuery UI

- Si une connexion Internet est disponible dans la salle
- À `jqueryui.com/demos`

1. Ouvrez le point de départ Do Now 6a à Do Nows\Do Now 6a-Starting Point

- Exercice précédent auquel jQuery UI a été ajouté en tant que package NuGet



Dans les étapes suivantes, pour dé-commenter des lignes, sélectionnez les lignes commentées et cliquez sur le bouton Uncomment de la barre d'outils, ou appuyez sur <Ctrl><K> suivi de <Ctrl><U>

2. Ouvrez la classe `BundleConfig` dans le dossier `App_Start`. Dé-commentez les deux blocs de lignes qui définissent les regroupements pour jQuery UI
3. Dans la vue `_Layout.xxhtml`, supprimez les commentaires des deux lignes commentées, près du haut et du bas du fichier
4. Dé-commentez la section `scripts` en bas de `ItemEdit.cshtml` dans le dossier `ToDoList`
5. Exécutez l'application dans Internet Explorer, allez à la page edit et cliquez sur le champ `DueDate`
 - Un datepicker permet de sélectionner la date

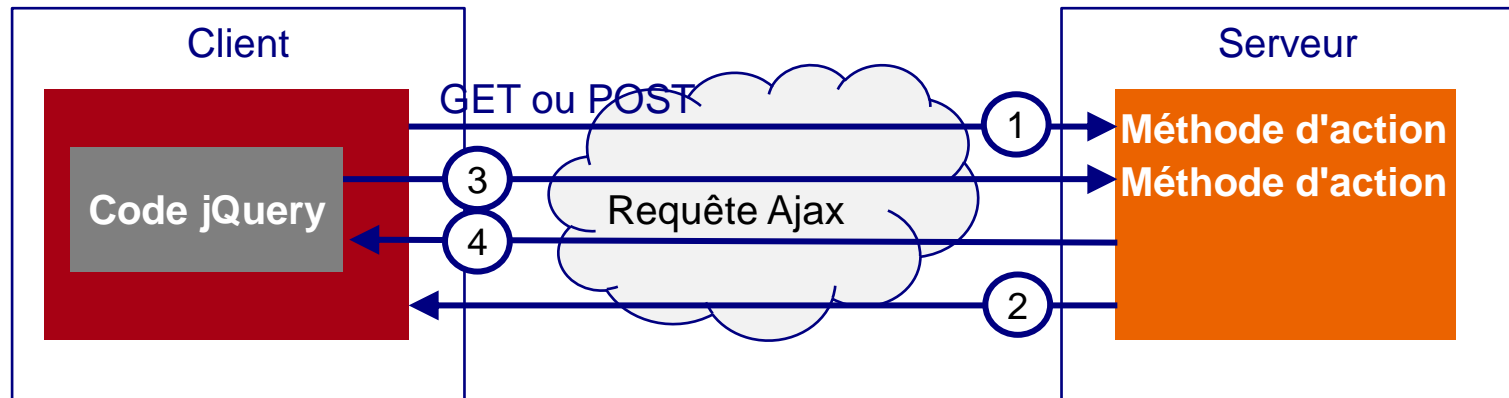
Fonctionnalités côté client

- **Présentation de jQuery**
- ➡ **Ajax**
- **Applications mobiles**
- **Applications internationales**
- **WebGrid et Chart**

Ajax

➤ Combine JavaScript, le DOM, CSS, et XMLHttpRequest

- Permet le rafraichissement partiel d'une page
 - Les requêtes HTTP GET ou POST normales actualisent toute la page
- La requête appelle une méthode d'action
- Peut renvoyer du HTML, XML, JSON, texte
- La réponse est traitée par du code JavaScript ou jQuery
- Le navigateur n'actualise pas toute la page



Ajax et les méthodes d'action

- **Les requêtes Ajax sont des requêtes HTTP normales**
 - Généralement GET ou POST
 - Traitées par une méthode d'action dans une application MVC
 - Asynchrones : l'appelant n'est pas bloqué pendant l'exécution de la requête
- **La méthode d'action peut renvoyer n'importe quel type de données, dont**
 - HTML : avec `return PartialView` pour renvoyer une vue partielle
 - JSON : avec `return Json` pour renvoyer des données JSON
 - Structure de données représentant des combinaisons d'objets, de tableaux, de collections
- **Du code jQuery traite les données renvoyées**
 - Il faut indiquer une méthode de rappel, appelée quand la réponse est reçue
 - Peut mettre à jour le DOM pour modifier la disposition de la page

JSON

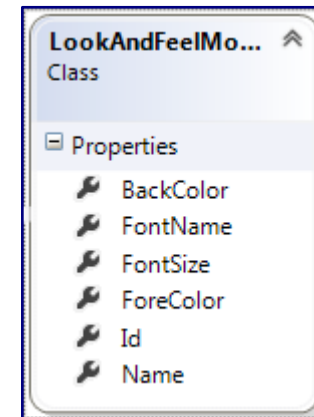
➤ JSON est la représentation native des objets en JavaScript

- Les propriétés des objets sont dynamiques, créées à la volée
- Le type de donnée est implicite

➤ Représentation JSON d'un objet LookAndFeelModel

```
{  
  "Id": 5,  
  "Name": "Pink",  
  "FontName": "Times New Roman",  
  "FontSize": 12,  
  "ForeColor": "Pink",  
  "BackColor": "LightGreen"  
}
```

JSON



➤ Très semblable aux littéraux objet de JavaScript

- Un tableau est déclaré avec des crochets

```
var person = { Name: "Fred", Scores: [ 10, 12, 8 ] };
```

jQuery

MVC et JSON

➤ Une méthode d'action peut renvoyer des données JSON

- Avec la classe `JsonResult` qui dérive de `ActionResult`
- Ou la méthode d'aide `Json`

```
var item = _lookAndFeelService.Read(id);  
return Json(item, JsonRequestBehavior.AllowGet);
```

Passe un objet

Obligatoire avec la
méthode d'action `GET`

➤ Le code jQuery accède aux propriétés avec leurs noms .NET

➤ On peut aussi utiliser la bibliothèque `Json.NET`

- Gratuite, peut être téléchargée avec NuGet
- Peut convertir des objets .NET en JSON : sérialisation
- Ou de JSON en .NET : désérialisation
- Peut aussi construire et manipuler des objets JSON depuis du code .NET

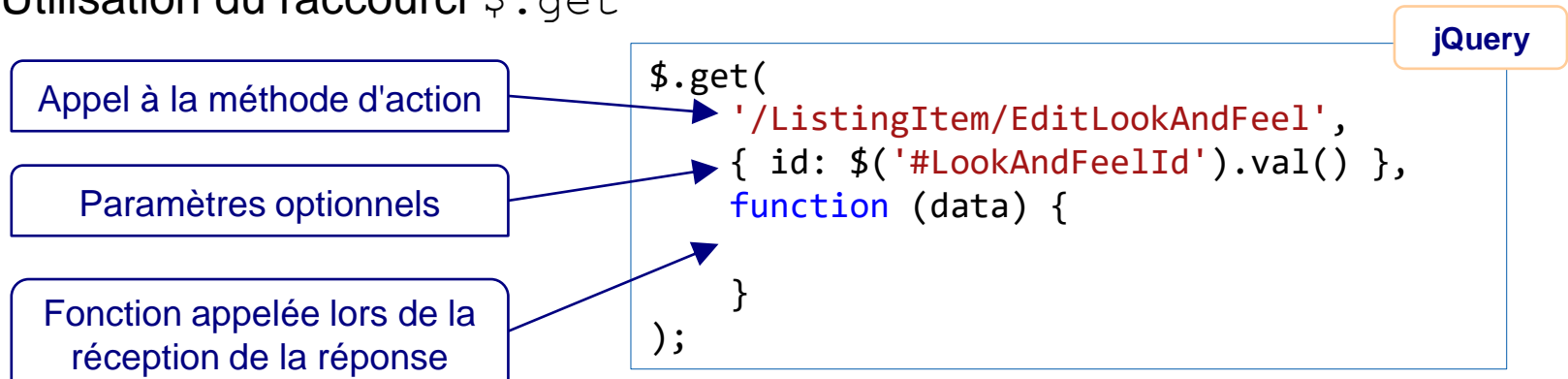
Méthodes Ajax de jQuery

➤ jQuery a plusieurs méthodes pour faire des appels Ajax au serveur

- La méthode `$.ajax` fait une requête Ajax au serveur
- Prend un objet littéral en paramètre, de nombreuses options sont disponibles
- Plusieurs raccourcis sont plus faciles à utiliser

Ajax	Description
<code>\$.get</code>	Charger des données du serveur avec une requête HTTP GET
<code>\$.post</code>	Charger des données du serveur avec une requête HTTP POST
<code>\$.getJSON</code>	Charger du JSON depuis le server avec une requête HTTP GET

- Utilisation du raccourci `$.get`



Afficher un dialogue modal avec Ajax et jQueryUI

➤ Définir une vue partielle et une méthode d'action qui la renvoie

- Vues partielle appelée `EditLookAndFeel`

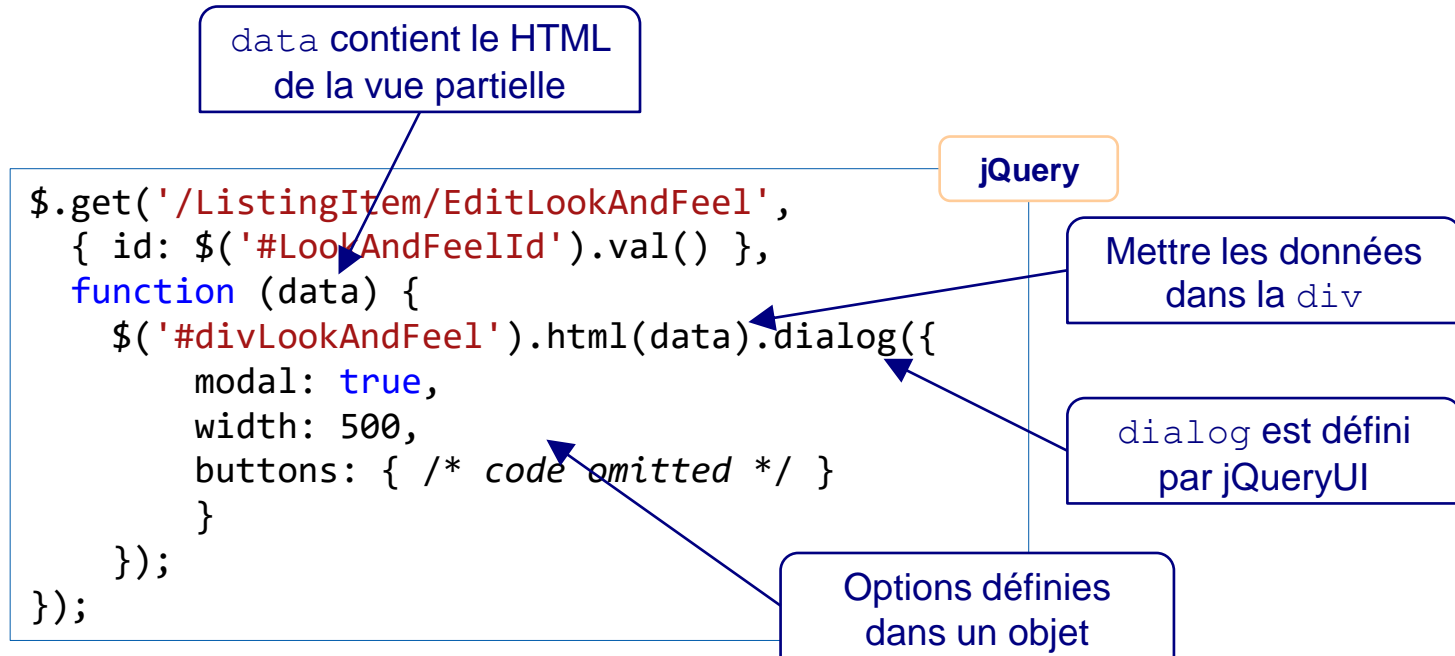
Please select the Look and Feel: `@Html.DropDownList("LookAndFeelList")`

- Méthode d'action

```
public ActionResult EditLookAndFeel(int id)
{
    var list = _lookAndFeelService.List();
    ViewBag.LookAndFeelList = new SelectList(list, "Id", "Name", id);
    return PartialView();
}
```

Afficher un dialogue modal avec Ajax et jQueryUI (suite)

- Appeler la méthode d'action et traiter les données renvoyées dans le code jQuery



Exercice 6.1 : Afficher un dialogue avec jQuery et Ajax

Fonctionnalités côté client

- **Présentation de jQuery**
- **Ajax**
- ➡ **Applications mobiles**
- **Applications internationales**
- **WebGrid et Chart**

Applications Mobile

➤ Deux types d'applications mobiles

- Applications clientes développées spécifiquement pour l'OS mobile
 - iOS, Android, Windows Phone ...
 - Pas le sujet de ce cours
- Applications Web accédées depuis un navigateur mobile

➤ Un navigateur mobile peut accéder à tous les sites Web

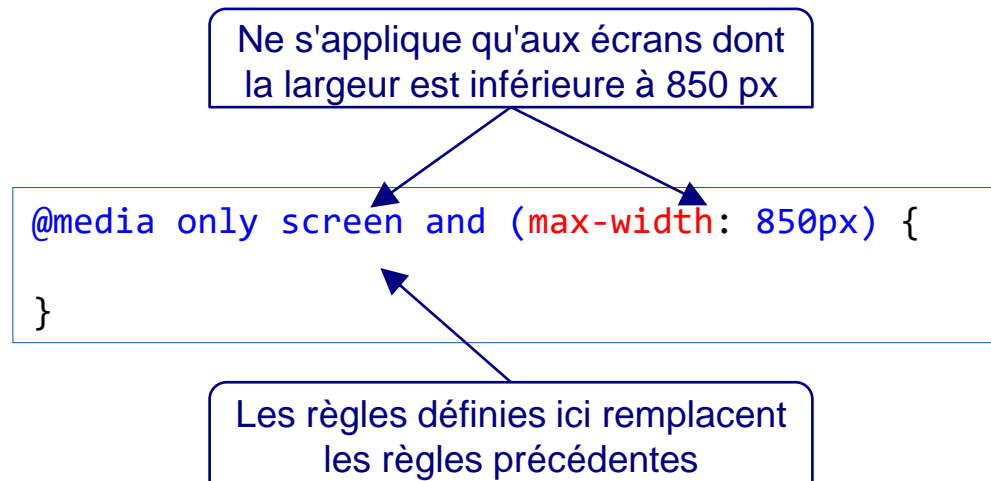
- Si le site n'est pas adapté pour les mobiles, la page peut être difficile à lire
- Les défis pour le développeur sont la taille de l'écran et le toucher



iOS = iPhone OS
OS = operating system

Adapter une application pour les mobiles

- **Deux stratégies principales pour adapter une application pour les mobiles**
 - Définir des règles CSS pour les écrans plus petits
 - Créer des vues spécifiques pour le mobiles
- **Des règles CSS spécifiques sont définies dans une section CSS @media**
 - Appliquées à tous les petits écrans, même sur un matériel non mobile

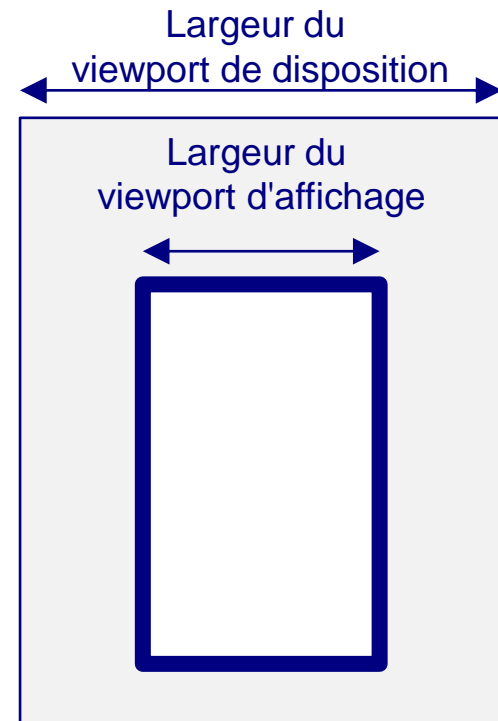


- **Bootstrap fait partie de Visual Studio et définit de nombreuses règles CSS**
 - Utilisé dans l'application Todo

Les fenêtres d'affichage des mobiles

- **Deux fenêtres d'affichage (viewport) sont définies sur les mobiles**
 - Le viewport de disposition, virtuel, à la largeur duquel la page est ajustée
 - Le viewport d'affichage, dont la largeur est celle du périphérique
- **La largeur du viewport de disposition est définie par le navigateur**
 - Valeur par défaut entre 800 et 980 pixels selon le navigateur
- **Quand la page est affichée**
 - Sa largeur est ajustée à celle du viewport de disposition
 - La page est zoomée afin que le viewport de disposition s'ajuste au viewport d'affichage
 - Conséquence : Le contenu de la page est trop petit
- **Le viewport de disposition peut être défini dans une page**
 - Typiquement dans `_Layout.cshtml`
 - Pour ajuster sa largeur à celle de l'écran

```
<meta name="viewport" content="width=device-width" />
```



Tester les applications mobiles

- **On peut tester les applications mobiles avec divers outils**
 - Sur un matériel mobile s'il peut accéder au site Web
 - Avec un émulateur, tel qu'Opera Mobile Emulator
 - Ou Windows Phone SDK
 - En manipulant la chaîne de l'agent utilisateur
 - Les navigateurs sont parfois équipés d'un émulateur d'appareils mobiles
 - C'est le cas pour Chrome, que nous utiliserons dans le prochain exercice
- **Chaque requête HTTP contient une chaîne de l'agent utilisateur dans son en-tête**
 - Définie par le navigateur, elle identifie la source de la requête
 - On peut la changer dans la plupart des navigateurs pour simuler un autre navigateur
 - Dans les outils du développeur dans la plupart des navigateurs
 - En saisissant `about:config` dans la barre d'adresse de Firefox ou avec l'extension User Agent Switcher

Chaîne de l'agent utilisateur = user agent string

Vues mobiles

- **On peut définir des vues spécifiques aux mobiles**
 - Généralement, des clones de vues existantes, avec quelques modifications
 - Ajouter `Mobile` dans le nom : `ItemDisplay.Mobile.cshtml`
 - Peut s'appliquer aux vues, aux vues de disposition, aux vues partielles
 - Si la requête provient d'un mobile, la version mobile est utilisée
- **ASP.NET utilise des fichiers de définition XML pour déterminer l'origine d'une requête**
 - Il compare la chaîne de l'agent utilisateur à la définition
 - `C:\Windows\Microsoft.NET\Framework\v[n]\Config\Browsers`
- **Une vue peut aussi tester si une requête provient d'un mobile**

```
if (HttpContext.GetOverriddenBrowser().IsMobileDevice)
{
    //...
}
```

- **On peut définir des vues spécifiques pour un matériel particulier**
 - En ajoutant un élément dans la table `Modes` de la classe `DisplayModeProvider`
 - Généralement fait au démarrage de l'application dans `Global.asax`

Ajouter une nouvelle condition de mode d'affichage

- Dans la méthode `Application_Start` de `Global.asax`
 - `Index.iPhone.cshtml` est utilisé si la condition est vraie

Insérer en première position

```
DisplayModeProvider.Instance.Modes.Insert(0, new DefaultDisplayMode("iPhone")
{
    ContextCondition = (context => context.GetOverriddenUserAgent().IndexOf
        ("iPhone", StringComparison.OrdinalIgnoreCase) >= 0)
});
```

Condition

Partie du nom de fichier

Exercice 6.2 : Adapter Todo pour les appareils mobiles

Fonctionnalités côté client

- **Présentation de jQuery**
- **Ajax**
- **Applications mobiles**
- ➡ **Applications internationales**
- **WebGrid et Chart**

Globalisation des applications

- **Rendre une application internationale implique plusieurs modifications**
 - Afficher les vues dans différentes langues
 - Modifier l'affichage et la saisie des nombres et des dates
 - Changer d'autres éléments d'IU, comme des images ou des pages entières
- **Plusieurs techniques peuvent être appliquées aux applications MVC**
 - Définir plusieurs ensembles de vues pour chaque langue
 - Placer les textes traduits dans des ressources
 - Développer un système personnalisé
 - Utile quand il existe déjà une base de données des traductions
- **L'utilisation de ressources est la plus simple**
 - Basée sur des techniques .NET standard
 - Facile à implémenter
 - Des traducteurs externes peuvent tirer parti de projets de ressources séparés
- **Le navigateur envoie le nom de sa culture avec chaque requête**
 - ASP.NET MVC peut l'utiliser pour déterminer la vue à afficher

Bases de la culture

➤ .NET définit la classe `CultureInfo` pour la globalisation

- Dans l'espace de noms `System.Globalization`
- On peut définir la culture par son nom, selon la région (neutre) ou indépendamment

Nom de la culture	Description
en	Anglais (indépendant de la région)
en-US	Anglais américain
en-UK	Anglais britannique
fr-FR	Français de France

- msdn.microsoft.com/en-us/globalization/bb896001.aspx pour une liste complète

➤ Chaque thread a deux propriétés pour les infos de culture

- `CurrentCulture` pour les dates et heures, les nombres, les unités monétaires, le tri, la casse...
- `CurrentUICulture` pour trouver les ressources spécifiques d'une culture
- Propriétés statiques de la classe `System.Threading.Thread`

Ressources

- **Les ressources .NET sont définies dans des fichiers XML .resx**
 - Peuvent contenir des chaînes, images, ou d'autres types de données
 - Visual Studio peut générer des classes pour en faciliter l'accès
- **Pour ajouter un fichier de ressources à un projet**
 - Sélectionner **Add | New item**
 - Sélectionner le modèle de fichier **Resource**
 - Nommer le fichier de ressources selon sa culture
 - `MyResource.resx` pour un fichier de culture invariante
 - `MyResource.fr.resx` pour des ressources en français (indépendantes de la région)
 - `MyResource.fr-CH.resx` pour des ressources français de Suisse
- **On utilise les ressources dans le code avec la classe générée**
 - Chaque ressource est une propriété

```
var msg = Resources.Models.ListingItemModel.PriorityQuantityError;
```

Exercice facultatif 6.3 : Internationalisation de l'application Todo

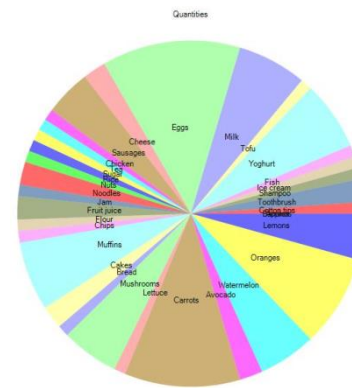
Fonctionnalités côté client

- **Présentation de jQuery**
- **Ajax**
- **Applications mobiles**
- **Applications internationales**
- ➡ **WebGrid et Chart**

Grilles et graphiques

- **Composants pour afficher grilles et graphiques sur des pages HTML**
 - Entièrement côté client avec jQuery, comme jqGrid
 - Ou avec une prise en charge côté serveur, comme Kendo UI for ASP.NET MVC de Telerik
 - Certains sont gratuits, d'autres non
- **ASP.NET MVC comprend les aides WebGrid et Chart**
 - Affichent des grilles et des graphiques
 - Pas de saisie, bien que la grille puisse être personnalisée avec du HTML
 - Les grilles peuvent être paginées, stylée, autoriser le tri des colonnes
 - Plusieurs types de graphiques : colonnes, camemberts, lignes...

Title	Quantity	Done
Apples		No
Bananas		No
Lemons	4	No
Oranges	8	No
Watermelon	5	No
Avocado	2	No
Carrots	10	No
Lettuce	1	No
Mushrooms	5	No
Bread	1	No
1 2 3 4 >		



Créer un WebGrid

➤ La création d'un WebGrid se fait en deux étapes

- Instancier un objet `WebGrid` en passant une collection à son constructeur
- Obtenir le HTML de la grille

➤ Le constructeur du WebGrid a de nombreux paramètres optionnels

- Le plus courant est la collection à afficher dans la grille

```
@{  
    var grid = new WebGrid(Model, canPage: false);  
}
```

C#

➤ La grille est affichée avec la méthode `GetHtml`

- Renvoie le HTML de la grille
- A de nombreux paramètres optionnels

```
@grid.GetHtml()
```

Personnaliser un WebGrid

➤ Options par défaut du WebGrid

- Affiche une colonne pour chaque propriété de la source
- Les colonnes peuvent être triées
- La pagination est opérationnelle, avec une taille de page de 10

➤ Toutes les options sont personnalisables

- Des parties de la grille peuvent aussi être stylées avec des classes CSS

```
@(grid.GetHtml(columns: new List<WebGridColumn>
{
    new WebGridColumn { ColumnName = "Title", CanSort = true },
    new WebGridColumn { ColumnName = "Quantity" },
    new WebGridColumn { ColumnName = "Done" }
}))
```


Générer un graphique dans un contrôleur

➤ La classe `Chart` crée un graphique

- Les dimensions sont définies dans le constructeur
- La méthode `AddSeries` ajoute des séries pour les axes x et l'axe y
- Généralement dans une méthode d'action du contrôleur

```
var chart = new Chart(600, 600);  
chart.AddSeries("Quantity", xValue: titles, yValues: quantities);
```

Collection des
valeurs de l'axe x



Collection des
valeurs de l'axe y



➤ `Chart` crée une image en mémoire

- Renvoyée au client en tant qu'image avec l'aide `File`

```
return File(chart.GetBytes(), "image/jpeg");
```

Affichage d'un graphique sur une page

- **Pour afficher un graphique sur une page, utiliser l'élément HTML ``**
 - Donner la méthode d'action comme valeur de l'attribut `src`
 - En passant les paramètres nécessaires

```

```

- **Démo : le fonctionnement des grilles et des graphiques dans l'application Todo**
 - Dossier `Demo\Chapter 6\Grid and Chart`
 - Lancer l'application et afficher une liste
 - Cliquer sur le nouveau lien **Display in grid**
 - Deux nouvelles méthodes d'action ont été ajoutées à `TodoListController`
 - `GridList` pour afficher les éléments dans une grille
 - `Chart` pour les afficher dans un graphique
 - La vue `GridList.cshtml` renferme le code permettant de contrôler la grille et le graphique

Résumé du chapitre

Dans ce chapitre, nous avons

- Construit des application hautement interactives avec jQuery
- Développé des éléments d'interface utilisateur côté client avec jQueryUI
- Optimisé le chargement des pages avec le regroupement et la compression
- Établi une communication entre le client et le server avec Ajax
- Rendu une application mobile
- Préparé les applications aux marchés internationaux
- Affiché des données à l'aide de grilles et de graphiques

Questions de révision

Quelles techniques minimisent la taille des pages envoyées au navigateur ?

Quels sont les avantages de JSON par rapport à XML ?

Citez des méthodes jQuery pour faire des appels Ajax au serveur

Comment transformer une vue en vue spécifique à un mobile ?

Quelles propriétés définissent les options internationales d'un thread ?

Listez des aides HTML de MVC HTML pour afficher des données structurées

VII. (262 à 300)

Créer des applications d'entreprise

Objectifs du chapitre

Dans ce chapitre, nous allons

- Authentifier les utilisateurs et contrôler l'accès aux méthodes d'action
- Traiter les exceptions non gérées avec la gestion des erreurs
- Ajouter des tests unitaires automatiques aux applications MVC
- Créer des services RESTful avec web API

Créer des applications d'entreprise

- ➔ **Contrôler l'accès utilisateur**
 - **Gestion des erreurs et débogage**
 - **Tests unitaires**
 - **Web API et applications distribuées**

Authentification et autorisation

- **Il est souvent nécessaire de restreindre l'accès à tout ou partie d'un site à certains utilisateurs**
 - Ou des catégories d'utilisateurs appelées rôles
 - *L'authentification* permet d'assurer que l'utilisateur est celui qu'il dit être
 - *L'autorisation* vérifie que l'utilisateur a accès aux ressources demandées
- **L'autorisation utilise l'attribut `Authorize` dans MVC**
 - Peut être défini pour une méthode d'action ou un contrôleur
 - Si appliqué à un contrôleur, il s'applique à toutes les méthodes d'action
 - On peut aussi le définir au niveau de l'application
 - L'utilisateur doit être authentifié pour accéder à une ressource `Authorize`
- **L'authentification peut utiliser plusieurs mécanismes**
 - Des comptes individuels, dans une base de données personnalisée, ou des comptes existants de Facebook, Twitter, Google, Microsoft ...
 - Des comptes d'entreprise avec Active Directory ou Office 365
 - L'authentification Windows

L'attribut Authorize

➤ L'attribut Authorize est un filtre

- Peut s'appliquer à une méthode

```
[Authorize]  
public ActionResult Index()
```

C#

- Ou à un contrôleur

```
[Authorize]  
public class AdminController : Controller
```

C#

- Ou comme filtre global dans la classe `FilterConfig` de `App_Start`

```
filters.Add(new AuthorizeAttribute());
```

C#

Attributs Authorize et AllowAnonymous

➤ L'autorisation peut se limiter à un ensemble d'utilisateurs ou de rôles

- Avec les propriétés `Users` et `Roles`
- Ont une liste d'utilisateurs ou de rôles séparés par des virgules

```
[Authorize(Roles="Admin,SuperAdmin")]
```

C#

➤ Quand `Authorize` est appliqué à toute l'application ou un contrôleur, aucune méthode d'action ne peut être appelée de façon anonyme

- `AllowAnonymous` permet un accès anonyme à une action ou un contrôleur
- Par exemple, pour la vue de connexion !

```
[AllowAnonymous]  
public class AccountController : Controller
```

C#

Identité ASP.NET

➤ L'authentification dans ASP.NET repose sur ASP.NET Identity

- Remplace l'appartenance (membership) des versions précédentes
- S'applique à MVC, Web Forms, Web API et plus

➤ ASP.NET Identity

- Extensible : On peut définir des données de profil utilisateur (ville d'origine, par exemple)
- Persistant : SQL Server par défaut, mais peut être personnalisé
- Utilisation de fournisseurs de connexion sociaux comme Facebook ou Google
- Prend en charge l'authentification à base de revendication (claims)
- Basé sur OWIN (Open Web Interface for .NET)
 - Standard pour découpler le serveur de l'application

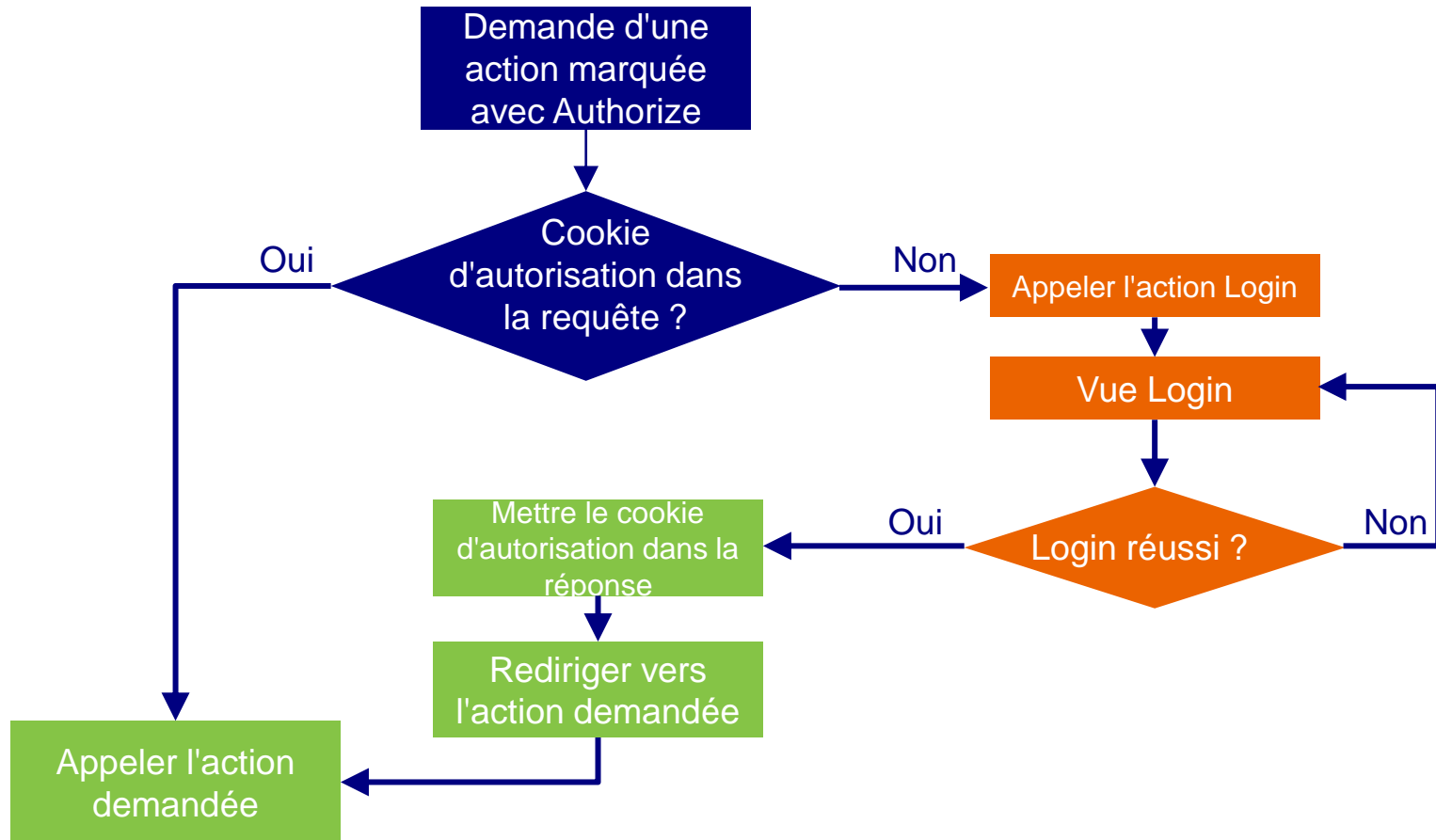
Comptes utilisateur individuels

- **Peuvent utiliser leur propre stockage, ou se connecter à des fournisseurs de connexion sociaux**
- **Basés sur deux classes**
 - `IdentityUser` : L'utilisateur, essentiellement un nom
 - `userManager`: Définit les opérations pouvant être faites sur l'utilisateur
- **Par défaut, les données sont stockées dans une base SQL Server**
 - Avec le code d'abord de Entity Framework
 - Cela peut être entièrement personnalisé
 - En ajoutant des données utilisateur spécifiques
 - En changeant l'emplacement ou le fournisseur du stockage

Le cookie d'autorisation

➤ La requête initiale n'a pas de cookie d'autorisation

- Après la connexion, les demandes et réponses ont un cookie d'autorisation



OAuth et OpenID

➤ Standards ouverts pour l'autorisation

- Connexion avec un compte existant sur un site de confiance (fournisseur)
- Facebook, Twitter, LinkedIn, Windows Live implémentent OAuth
- Google, Yahoo implémentent OpenID

➤ La classe `OAuthWebSecurity` cache la plupart des différences entre les protocoles et les fournisseurs

- Interagit avec la bibliothèque open-source `DotNetOpenAuth`
- A des méthodes pour s'inscrire auprès des fournisseurs
- OAuth nécessite un ID d'autorisation du fournisseur
- Pour inscrire un client Facebook

```
OAuthWebSecurity.RegisterFacebookClient(  
    appId: "xxxxxxxxxxxxxxxx",  
    appSecret: "yyyyyyyyyyyyyy");
```

C#

- **Démonstration : le modèle standard pour l'authentification par compte individuel**
 - Ouvrez `IdentityModel` dans le dossier `Models`
 - Notez les classes dérivées de `IdentityUser` et `IdentityDbContext`
 - Ouvrez `AccountController`
 - Notez la propriété `userManager` et son utilisation dans le constructeur
 - Elle est utilisée pour faire des opérations sur l'utilisateur
 - Exécutez et inscrivez un nouveau compte
 - Examinez la base de données créée par défaut dans le dossier `App_Data`
 - La chaîne de connexion `DefaultConnection` définie dans `Web.config`
 - Ouvrez `Startup.Auth` dans `App_Start`
 - Notez le `LoginPath`
 - Observez les instructions en commentaires concernant les comptes de fournisseurs tiers
 - Pour les utiliser, vous devez vous enregistrer sur le site des développeurs de chaque fournisseur

Application terminée à `Demo\Chapter 7\IndividualAccountAuthentication`

Exercice 7.1 : Ajouter des méthodes d'authentification et d'autorisation

Autres options d'authentification

- **Autres options d'authentification pour la création d'un nouveau projet**
 - Authentification Windows
 - Comptes d'entreprise ou d'établissement scolaire
- **Authentification Windows**
 - Les utilisateurs sont authentifiés avec leur login Windows
 - Pas de contrôleur de compte
 - Option ajoutée dans `Web.config` : `<authentication mode="Windows" />`
- **Comptes d'entreprise ou d'établissement scolaire**
 - L'authentification est faite par Active Directory (cloud ou local) ou Office 365

Change Authentication

☐ No Authentication

☐ Individual User Accounts

☒ Work And School Accounts

☐ Windows Authentication

For applications that authenticate users with Active Directory, Microsoft Azure Active Directory, or Office 365.

[Learn more](#)

On-Premises

On-Premises Authority:

Enter metadata document URL

App ID URI:

Default value will be automatically populated

OK Cancel

Autres options de sécurité

- **Le mot de passe est envoyé en clair dans la page de connexion**
 - Il faut utiliser SSL pour crypter les demandes et les réponses
 - Fait dans le bonus du dernier exercice
 - Ajouter l'attribut `RequireHttps` au contrôleur `Account`
 - Ainsi que sur tous les méthodes d'actions et contrôleurs protégés
- **Prévenir les attaques potentielles sur le site Web**
 - Installer le package NuGet AntiXSS pour prévenir les attaques intersites
 - Encode tout le HTML et JavaScript
 - Prévenir les falsification intersites
 - Ajouter un jeton caché dans les pages avec `@Html.AntiForgeryToken()`
 - Ajouter l'attribut `ValidateAntiForgeryToken` aux méthodes d'action POST
 - Fait par défaut par les modèles de Visual Studio

SSL = Secure Sockets Layer

Créer des applications d'entreprise

- Contrôler l'accès utilisateur
- ➡ **Gestion des erreurs et débogage**
- Tests unitaires
- Web API et applications distribuées

Gérer les erreurs avec l'attribut `HandleError`

- **Un filtre d'exception implémente l'interface `ExceptionHandler`**
 - Définit une seule méthode, `OnException`
 - S'exécute quand une exception non gérée se produit où l'attribut est appliqué
 - Une exception non gérée est une exception non traitée par un `try-catch`
- **MVC fournit une implémentation intrinsèque, `HandleErrorAttribute`**
 - La propriété `ExceptionType` indique l'exception à traiter
 - La propriété `View` indique la vue à afficher, `Error.cshtml` par défaut
 - La vue reçoit un modèle de type `HandleErrorInfo`
- **Comme les autres filtres, un filtre d'exception peut être appliqué à une méthode d'action**
 - Souvent défini dans `global.asax` afin de s'appliquer à toutes les méthodes d'action de tous les contrôleurs
 - Les modèles de VS définissent un filtre global `HandleError` et une vue `Error.cshtml`

Utiliser des erreurs personnalisées

➤ Les filtres globaux sont dans la classe `FilterConfig` du dossier `App_Start`

- `RegisterGlobalFilters` est appelé depuis `global.asax`
- D'autres filtres peuvent y être inscrits

```
filters.Add(new HandleErrorAttribute())
```

➤ Pour que l'attribut `HandleError` fonctionne, il faut définir `customErrors`

- Dans le fichier `Web.config`

```
<system.web>  
  <customErrors mode="On"/>
```

- Valeurs de l'attribut `mode` :
 - `Off` : Une page d'erreur standard est affichée (« page jaune »)
 - `On` : Des pages d'erreur personnalisées sont affichées
 - `RemoteOnly` : Des pages d'erreur personnalisées sont affichées si le navigateur n'est pas sur le serveur Web, sinon une page d'erreur standard est affichée

Définir la valeur dans le fichier `Web.config` situé à la racine du site Web, pas dans le dossier `Views`

Classe HandleErrorInfo

➤ **La classe `HandleErrorInfo` est passée en tant que modèle à la vue d'erreur**

- Propriétés `ActionName` et `ControllerName`
- La propriété `Exception` donne accès aux détails de l'exception

```
Exception <b>@Model.Exception.Message</b> <br />  
In action method <b>@Model.ActionName</b>  
of <b>@Model.ControllerName</b> controller
```

➤ **Il faut soigneusement tester le code de la vue de gestion des erreurs**

- Si une exception se produit, la page jaune par défaut est affichée

1. Ouvrez et générez le point de départ Do Now 7a dans `Do Nows\Do Now 7a-Start Point`
2. Ouvrez la méthode d'action `Index` du contrôleur `Home`
 - Elle fait une division par zéro qui lève une exception
3. Exécutez l'application en dehors de Visual Studio avec `<Ctrl><F5>`
 - Une page jaune est affichée
4. Ouvrez le fichier `Web.config` situé à la racine de l'application. Ajoutez la ligne suivante sous la balise `<system.web>`

```
<customErrors mode="On"/>
```

5. Ouvrez la vue `Error.cshtml` dans le dossier `Shared`
6. Ajoutez une ligne `model` pour `HandleErrorInfo`
7. Ajoutez la ligne suivante après l'élément `h2`

```
@Html.Raw(Model.Exception.StackTrace.Replace("\n", "<br/>"))
```

8. Exécutez avec `<Ctrl><F5>` et examinez la page d'erreur personnalisée

Créer des applications d'entreprise

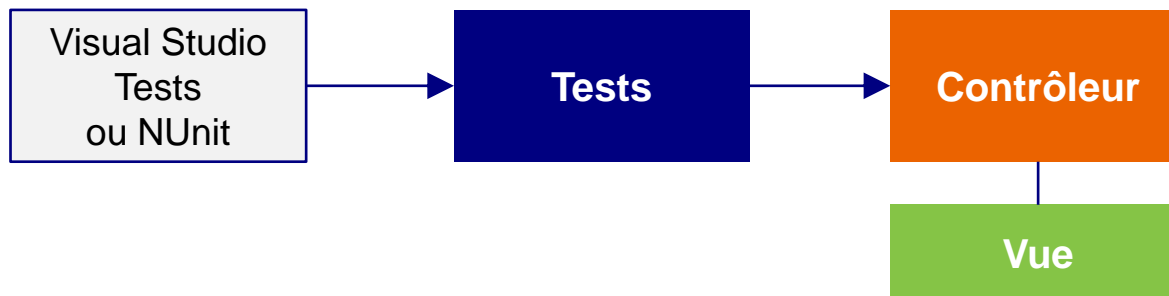
- Contrôler l'accès utilisateur
- Gestion des erreurs et débogage
- ➡ **Tests unitaires**
- Web API et applications distribuées

Tests unitaires et développement piloté par les tests

- **Avec les tests unitaires, on teste des unités individuelles de code**
 - Généralement des classes ou des méthodes individuelles
- **Le développement piloté par les tests (TDD, Test-driven development)**
 - Fonctionne sur de petites unités logicielles
 - Commencer par écrire un test qui définit un comportement attendu
 - Le test doit échouer, car l'implémentation n'est pas encore faite
 - Écrire l'implémentation jusqu'à ce que le test réussisse
 - Refactoriser le code, en vérifiant que le test réussit toujours
 - Répéter sur chaque unité logicielle jusqu'à ce que l'application soit terminée
- **L'automatisation des tests est un des points forts de MVC**
 - Prise en charge du TDD dans les applications ASP.NET MVC
 - Ainsi que des tests unitaires standard
 - Cocher la case **Create a unit test project** lors de la création d'un nouveau projet MVC

Test unitaires appliqué aux applications MVC

- **Les tests unitaires sont faits sur des bibliothèques de classes**
 - Dans MVC, les tests se focalisent sur les contrôleurs et les modèles
 - Le code de test prend la place des vues
- **Une application MVC ne doit pas « savoir » qu'elle est testée**
 - Les tests sont écrits dans un projet bibliothèque de classes
 - Fait référence à l'application MVC
 - Instancie les contrôleurs et appelle les méthodes d'action
- **Un projet de test est exécuté par une application de test**
 - Nous utiliserons Visual Studio Tests
 - Il existe des alternatives, par exemple NUnit



Lancer les tests

➤ Visual Studio Tests charge la bibliothèque de classes des tests

- Recherche des attributs spécifiques sur les classes et les méthodes
- Instancie les classes et appelle les méthodes selon ces attributs
- NUnit fonctionne de la même façon, seuls les noms des attributs diffèrent

➤ Principaux attributs de Visual Studio Tests

Attribut	Description
TestClass	La classe fait partie des tests
TestMethod	La méthode est appelée lors des tests
ClassInitialize	La méthode est appelée au chargement de la classe
ClassCleanup	La méthode est appelée au déchargement de la classe
TestInitialize	La méthode est appelée avant chaque test de la classe
TestCleanup	La méthode est appelée après chaque test de la classe

Méthodes de test

➤ Une méthode de test a trois phases

- *Arrange (préparer)* : Instancier le contrôleur et les autres ressources
- *Act (agir)* : Appeler la méthode à tester
- *Assert (affirmer)* : Tester les valeurs résultat

```
[TestClass]
public class HomeControllerTest
{
    [TestMethod]
    public void Index()
    {
        // Arrange
        var controller = new HomeController();

        // Act
        var result = controller.Index() as ViewResult;

        // Assert
        Assert.AreEqual(result.ViewName, "Index");
    }
}
```

C#

Caster le résultat pour
avoir accès à la propriété
ViewName

Assert

➤ La classe `Assert` a de nombreuses méthodes statiques pour tester le résultat

- Si un `Assert` échoue, la méthode est marquée comme « échouée »
- Signaux rouge-vert

Méthode <code>Assert</code>	Description
<code>AreEqual/AreNotEqual</code>	Valeurs égales / différentes
<code>AreSame/AreNotSame</code>	Deux variables référencent le même objet / des objets différents
<code>IsTrue/IsFalse</code>	La condition est vraie / fausse
<code>IsInstanceOfType/ IsNotInstanceOfType</code>	L'objet est / n'est pas une instance du type
<code>IsNull/IsNotNull</code>	L'objet est / n'est pas nul

Visual Studio et les tests

➤ Les tests sont effectués depuis Studio

The screenshot shows the Test Explorer window in Visual Studio. It displays a list of tests under the heading 'Test Explorer'. The 'Failed Tests' section shows one failed test, 'List', with a red 'X' icon and a duration of 515 ms. The 'Passed Tests' section shows one passed test, 'Index', with a green checkmark icon and a duration of 163 ms. Below the list, the details for the failed 'List' test are shown, including the source file 'CategoryControllerTest.cs line 26', the error message 'Test Failed - List', the assertion failure 'Message: Assert.AreEqual failed. Expected:<3>. Actual:<2>.', the elapsed time '515 ms', and the stack trace 'CategoryControllerTest.List()'. Three annotations with arrows point to specific elements: 'Signal rouge' points to the red 'X' icon, 'Signal vert' points to the green checkmark icon, and 'Détails du résultat du test' points to the detailed error message for the failed test.

Signal rouge

Signal vert

Détails du résultat du test

Test Explorer

Run All | Run... | Playlist : All Tests

Failed Tests (1)

List 515 ms

Passed Tests (1)

Index 163 ms

List

Source: CategoryControllerTest.cs line 26

Test Failed - List

Message: Assert.AreEqual failed. Expected:<3>. Actual:<2>.

Elapsed time: 515 ms

StackTrace:

CategoryControllerTest.List()

À vous 7b : Ajouter des tests unitaires à Todo

À vous

1. Ouvrez et générez le point de départ de l'exercice Do Now 7b qui se trouve dans `Do Nows\Do Now 7b-Start Point`
2. Ouvrez la classe `HomeControllerTest` qui se trouve dans le dossier `Controllers` du projet `Todo.Web.Test` et essayez de comprendre à quoi sert le code
3. Cliquez sur **Test | Run | All Tests** et regardez le résultat dans **Test Explorer**
 - Un test a échoué
4. Cliquez sur **Test | Debug | All Tests** et essayez de comprendre pourquoi le test a échoué
 - Le code est interrompu avant l'appel à la méthode `Assert`
5. Corrigez le code dans la méthode `Index` de la classe `HomeController` du projet `Todo.Web`
 - Passez le nom de la vue à la méthode `View`
6. Relancez tous les tests
 - Il n'y a plus d'erreurs

Test d'applications complexes

➤ Les tests peuvent être beaucoup plus complexes

- Les méthodes du contrôleur appellent des méthodes de service
- Les méthodes de service appellent le référentiel qui accèdent à la base de données

➤ Une bonne pratique consiste à tester les composants individuellement

- Tests *unitaires*
- Le niveau service peut avoir son propre projet de test
- Les tests MVC se focalisent sur le test des contrôleurs

➤ Le pattern Mock remplace un composant par un composant factice

- Conçu spécifiquement pour les tests
- On peut avoir recours à des outils logiciels tels que Moq
- Installé avec NuGet

➤ **Démo : une application de test avec Moq**

- Ouvrez la solution du dernier exercice À vous 7b qui se trouve dans `Do Nows\Do Now 7b-Completed`
- Moq a été ajouté au projet `Todo.Web.Test`
- Ouvrez la classe `CategoryControllerTest`
- Notez les points suivants :
 - Une variable de classe appelée `categories` renferme les données de test
 - Un objet `Mock` est initialisé avec `ICategoryService` en type générique
 - L'objet `Mock` est passé au `CategoryController`, prenant la place du paramètre `ICategoryService` attendu

Créer des applications d'entreprise

- Contrôler l'accès utilisateur
- Gestion des erreurs et débogage
- Tests unitaires
- ➔ **Web API et applications distribuées**

Applications distribuées

- **Une application distribuée a des composants situés sur plusieurs ordinateurs**
 - Communiquent à travers le réseau
 - Typiquement Internet ou un intranet
- **Les services Web permettent aux composants de communiquer sur Internet**
 - Les clients appellent des méthodes sur les serveurs
 - Plusieurs protocoles existent, dont SOAP, WSDL
 - Web API est une alternative légère
- **Web API fait maintenant partie de ASP.NET**
 - Utilise REST : L'URL définit la ressource demandée
 - Exploite les possibilités de HTTP
 - Facile à utiliser par tout client qui peut faire une requête HTTP
 - Les données de la réponse peuvent être du XML ou du JSON

SOAP = Simple Object Access Protocol

WSDL = Web Services Description Language

Web API

➤ Web API utilise également des contrôleurs

- Les classes dérivent d'une autre classe : `ApiController`
- Pas limité aux applications MVC, fonctionne aussi avec les Web Forms
- Les méthodes d'action renvoient un objet, pas un `ActionResult`
- L'objet renvoyé est sérialisé en XML ou JSON et passé au client

➤ Chaque méthode d'action est mappée à un verbe HTTP

- Par nom
 - Le nom de la méthode d'action inclut le nom du verbe
 - `Get`, `GetList`, ou `PleaseGetTheList` sont mappés au verbe GET
- Ou avec un attribut tel que `HttpGet`

➤ Les verbes HTTP sont mappés à des actions logiques

- GET pour lire des données
- POST pour ajouter des données
- PUT pour modifier des données
- DELETE pour supprimer des données

Créer un contrôleur Web API

- Dans le dialogue Add controller, sélectionner Web API 2 Controller – Empty
 - D'autres modèles ajoutent des méthodes d'action dans le contrôleur
- Ajouter des méthodes mappées aux verbes HTTP

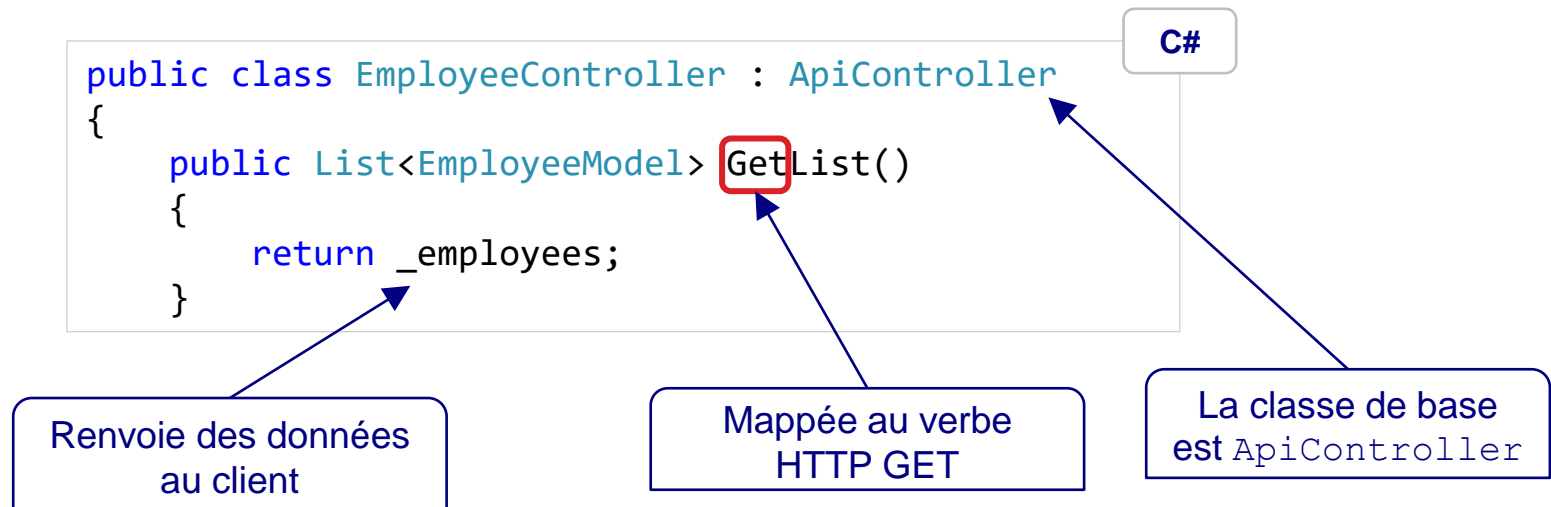


Table de routage de Web API

➤ Web API utilise une table de routage spécifique

- Initialisée dans `WebApiConfig` du dossier `App_Start`
- On peut ajouter et personnaliser des routes comme pour les routes MVC
- Pas de nom d'action dans les routes
 - La méthode est déterminée en combinant le nom du contrôleur au verbe HTTP

```
config.Routes.MapHttpRoute(  
    name: "DefaultApi",  
    routeTemplate: "api/{controller}/{id}",  
    defaults: new { id = RouteParameter.Optional }  
);
```

C#

Préfixe par défaut des URL

`http://foo.com/api/Employee/2`

Clients Web API

- **Les clients Web API peuvent être n'importe que type d'application**
 - Doivent pouvoir faire une requête HTTP et traiter le XML ou JSON renvoyé
 - .NET ou non .NET
 - Le format des données est défini dans la requête du client
 - Avec le paramètre `Accept` de l'en-tête de la requête HTTP
- **Les clients Web peuvent faire des appels avec Ajax**
 - Avec jQuery et la méthode générale `$.ajax`
 - Ou avec des aides plus spécifiques : `$.get`, `$.post`, ou `$.getJSON`
- **Les clients Windows peuvent utiliser la classe `WebClient`**
 - Ou `HttpClient` de ASP.NET Web API Client Libraries, téléchargeable depuis NuGet

➤ **Démo : une application serveur Web API**

- Dans `Demo\Chapter 7\WebAPI`
- `EmployeeApiController` a des méthodes Web API pour les opérations CRUD
- La vues `Index` a du code jQuery pour faire un appel Ajax à la méthode `GetList`
- Exécuter avec `<Ctrl><F5>` et cliquer sur le bouton **List Employees**
- Examiner les requêtes ou en faire avec Fiddler

Exercice 7.2 : Créer un contrôleur Web API

Résumé du chapitre

Dans ce chapitre, nous avons

- **Authentifié les utilisateurs et contrôlé l'accès aux méthodes d'action**
- **Traité les exceptions non gérées avec la gestion des erreurs**
- **Ajouté des tests unitaires automatiques aux applications MVC**
- **Créé des services RESTful avec web API**

Questions de révision

Quel attribut limite l'accès à une méthode d'action aux utilisateurs authentifiés ?

Comment gérer les exceptions dans une application ASP.NET MVC ?

Quels attributs sont appliqués à une classe ou une méthode pour les inclure dans des tests ?

Citez les verbes HTTP permettant d'implémenter des opérations CRUD

VIII. (301 à 309)

Déploiement des applications

Objectifs du chapitre

Dans ce chapitre, vous allez

- Préparer le déploiement d'une application
- Déployer l'application ASP.NET MVC sur un serveur Web IIS

Déploiement des applications

➔ **Préparer et déployer dans IIS**

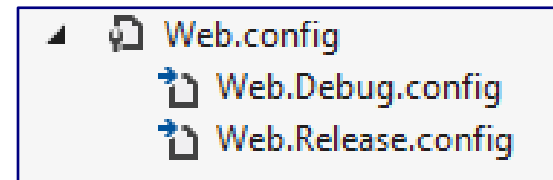
Préparer le déploiement

- **Le déploiement d'une application doit être préparé**
 - Compiler toutes les vues afin de détecter les erreurs
 - Retirer les informations de débogage
 - Nettoyer `Web.config`
- **On peut déployer sur un serveur IIS**
 - Ou dans le nuage (cloud), Windows Azure par exemple
 - Cibler un serveur de pré-production ou de production
 - Peut être fait avec l'assistant de publication de Visual Studio

Modifier Web.config

- **Visual Studio définit par défaut deux configurations**
 - Debug et Release
 - On peut en ajouter d'autres avec BUILD | Configuration Manager
- **Chaque configuration a un fichier de transformation Web.config**

- Nommé d'après la configuration
- A des instructions pour modifier les paramètres de Web.config
- Traité lors de la génération du projet



- **Dans Web.Release.config**

- Pour retirer l'attribut debug de l'élément compilation

```
<system.web>  
  <compilation xdt:Transform="RemoveAttributes(debug)" />
```

Publier l'application

➤ Utiliser BUILD | Publish dans le projet Web

- On peut définir plusieurs profiles
- Chacun définit la destination

Méthode de publication	Description
Web Deploy	Publie directement dans IIS. Il faut avoir des droits d'administrateur
Web Deploy Package	Crée un package qui peut être importé dans IIS
FTP	Publie dans un dossier FTP
File System	Publie dans un dossier local ou un partage réseau
FPSE	FrontPage Server Extensions (ancien)

➤ Web Deploy comprend des instructions pour la configuration de IIS

- Les bases de données peuvent aussi être incluses dans le package de déploiement

FTP = File Transfer Protocol

IIS et les applications ASP.NET MVC

- **IIS 7 et les versions ultérieures sont compatibles avec ASP.NET MVC en natif**
 - Sauf si le pool d'applications est configuré en mode classique, pour la compatibilité avec IIS 6
- **IIS 6 et IIS 7 en mode classique nécessitent de la configuration**
 - Utilisent l'extension de fichier pour traiter les URL demandées
 - Les fichiers Web Forms `.aspx` sont routés à ASP.NET
 - Mais MVC n'utilise pas d'extension de fichier
- **Deux solutions principales**
 - Ajouter une pseudo extension de fichier aux contrôleurs dans la table de routage, et mapper cette extension à ASP.NET dans IIS
 - Par exemple, `Todo/Category.mvc/List`
 - Utiliser un mappage générique à ASP.NET dans IIS
 - Toutes les demandes, y compris les fichiers `.html`, passent par ASP.NET
 - Voir <http://www.asp.net/learn/mvc/tutorial-08-cs.aspx> pour plus de détails

Exercice 8.1 : Déployer l'application

Veillez vous reporter au manuel d'exercices

Résumé du chapitre

Dans ce chapitre, vous avez

- **Préparé le déploiement d'une application**
- **Déployé l'application ASP.NET MVC sur un serveur Web IIS**