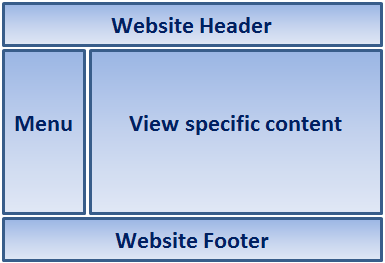
**Chapitre07**

1. **Les Layouts (MasterPage) dans ASP.NET MVC**
   1. **L'utilisation des layouts**

Les Layouts sont utilisés dans MVC pour fournir un aspect cohérent sur toutes les pages de notre application Web. Il est le même que la définition des pages maîtres, mais MVC fournit des fonctionnalités supplémentaires avec les layouts. Un layout typique est composée des éléments suivants:

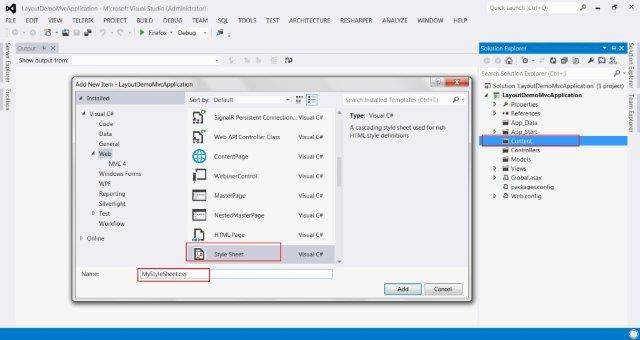
**1.** En-tête de page  
**2.** Pieds de page   
**3.** Menu de Navigation  
**4.** Contenu de la vue



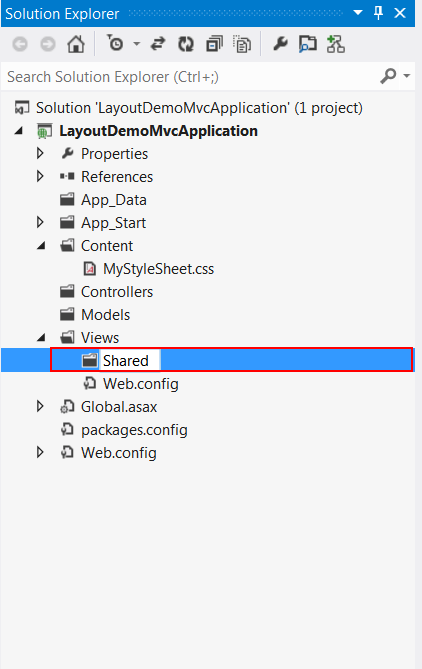
Au lieu d'avoir toutes ces sections dans chaque vue, on peut les définir dans un fichier layout et la faire hériter dans toutes les autres vues.

**Étape 1:** Créer une application asp.net mvc 4 vide.

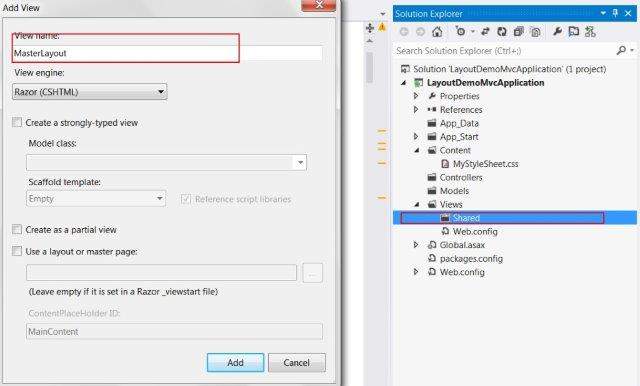
**Étape 2 :** Si vous avez des fichiers css, ajouter les dans le dossier "**Content**" de votre application**.**



**Étape 3:** Cliquer droit sur le dossier "Views" et ajouter un nouveau dossier nommé "**Shared**" s'il n'est pas déjà créé.



**Étape 4:** Cliquer droit sur le dossier "Shared" et ajouter une nouvelle vue avec le nom **"MasterLayout.cshtml" ou "\_Layout.cshtml" si elle n'existe pas**. C'est cette vue qui représente notre page maitresse.



La vue "MasterLayout" peut avoir n'importe quel nom avec l'extension ".cshtml".

Voici le code de la vue "MasterLayout.cshtml"

<html>  
<head>  
    <title>@ViewBag.Title</title>  
    @\*tous les fichiers javascript et css nécessaires pour votre application \*@

</head>  
<body>  
    <table border="1" style="width:800px; font-family:Arial">  
        <tr>  
            <td colspan="2" style="text-align:center">  
                <h3>Website Header</h3>  
            </td>  
        </tr>  
        <tr>  
            <td style="width:200px">  
               <h3>Menu</h3>  
            </td>  
            <td style="width:600px">  
                @**RenderBody**() // l'endroit réservé pour la vue de contenu  
            </td>  
        </tr>  
        <tr>  
            <td colspan="2" style="text-align:center; font-size:x-small">  
                <h3>Website Footer</h3>  
            </td>  
        </tr>  
    </table>  
</body>  
</html>

**Remarque**: Pour insérer un fichier css dans votre vue, utiliser le code suivant:

<Link rel = "stylesheet" href = "@ Url.Content (" ~ / Content / MyStyleSheet.css ")" />

**Remarque:** La vue de contenu doit être insérée dans l'endroit où la méthode @**RenderBody**() est appelée.  
  
**Étape 5:** Maintenant, on doit modifier la vue index pour qu'elle utilise le fichier layout "**MasterLayout.cshtml"**. Ajouter la ligne suivante juste après la déclaration du modèle "@model"

@{ **Layout = "~/Views/Shared/MasterLayout.cshtml";**}  
  
**Étape** 6: Tester la vue.

1.2 **Utilisation du fichier " \_ViewStart"**

Le fichier "\_ViewStart.cshtml" s'exécute au début de rendu de chaque vue. Le contenu de ce fichier sera exécuté avant l'exécution du code de toute vue. Ce fichier doit définir le layout utilisée par chaque vue dans votre application.

@{

Layout = "~/Views/Shared/\_Layout.cshtml";

}

**Remarque**:

Dans une application MVC on peut avoir plusieurs fichiers "\_ViewStart.cshtml". L'exécution du fichier dépond de sa position dans la hiérarchie des dossiers de votre application.

Exemple:

1) Views\\_ViewStart.cshtml

2) Views\Home\\_ViewStart.cshtml

3) Views\Products\\_ViewStart.cshtml

**1-3) Les sections nommées dans le fichier layout dans mvc**

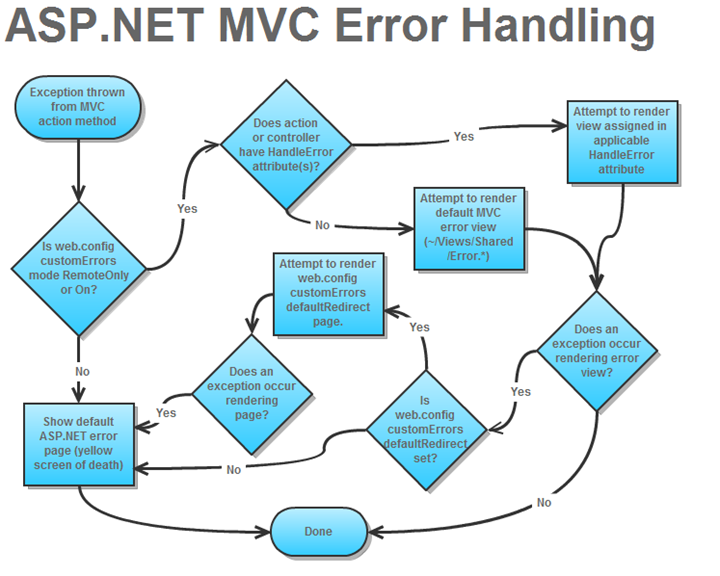
Si par exemple on veut avoir un menu par vue, dans ce cas-là on doit **définir une section** **dans chaque vue**. Dans cette section on définit le menu.

**Étape 1:** Définissez une section "Menu" dans une vue. Pour définir une section dans une vue, on doit utiliser l'instruction "@section" suivie du nom de la section.

Exemple:

@section Menu  
{  
    @Html.ActionLink("List", "Index") <br />  
    @Html.ActionLink("Details", "Details", new { id = Model.Id }) <br />  
    @Html.ActionLink("Delete", "Delete", new { id = Model.Id })  
}  
  
**Étape 2**: Dans le fichier **Layout**, on doit spécifier l'endroit du rendu de la section qu'on a définie en utilisant la fonction @**IsSectionDefined et la méthode @RenderSection.**.  
 

**B) La gestion des exceptions**



ASP.NET MVC est livré avec un support intégré pour la gestion des exceptions en utilisant une fonctionnalité appelée filtres d'exception. Nous allons apprendre deux approches ici: une avec l'utilisation des try catch et une autre en utilisant une classe qui implémente l'interface "**IExceptionFilter" ou qui dérive de la classe "ExceptionFilterAttribute**".

1. **La méthode simple**
2. Dans le dossier "Models", ajouter un modèle nommé "**ErrorInfoViewModel**"

public class ErrorInfoViewModel

{

public string ActionName{get; set;}

public string ControllerName{get; set;}

public string Message{get; set;}

}

1. Dans le dossier "**Views/Shared"**, ajouter une nouvelle vue nommée "**Error**" fortement typée du modèle "**ErrorInfoViewModel**"

Copiez le code suivant à l'intérieur de la vue "**Error.cshtml**" nouvellement créé :

@model Models.ErrorInfoViewModel

@{

Layout = null;

}

<!DOCTYPE html>

<html>

<head>

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

<title>Error</title>

</head>

<body>

<div>

<h1>Désolé, une erreur s'est produite dans le programme</h1>

<h2>Le contrôleur: @Model.ControllerName</h2>

<h2>L'action: @Model.ActionName</h2>

<h2>Le message: @Model.Message</h2>

<h3> @Html.ActionLink("Retourner à la page d'accueil","Index")</h3>

</div>

</body>

</html>

1. Dans les méthodes du contrôleur, ajouter les blocs **try** et **catch**. Quand une exception est levée le bloc catch est exécuté et il nous redirige vers la vue d'erreur en lui envoyant un objet de type "**ErrorInfoViewModel**" comme paramètre.

**Exemple**:

public ActionResult SomeError()

{

try

{

Votre code;

}

catch(Exception ex)

{

RouteData route = ((MvcHandler)HttpContext.Handler).RequestContext.RouteData;

string controllerName = route.Values["controller"].ToString();

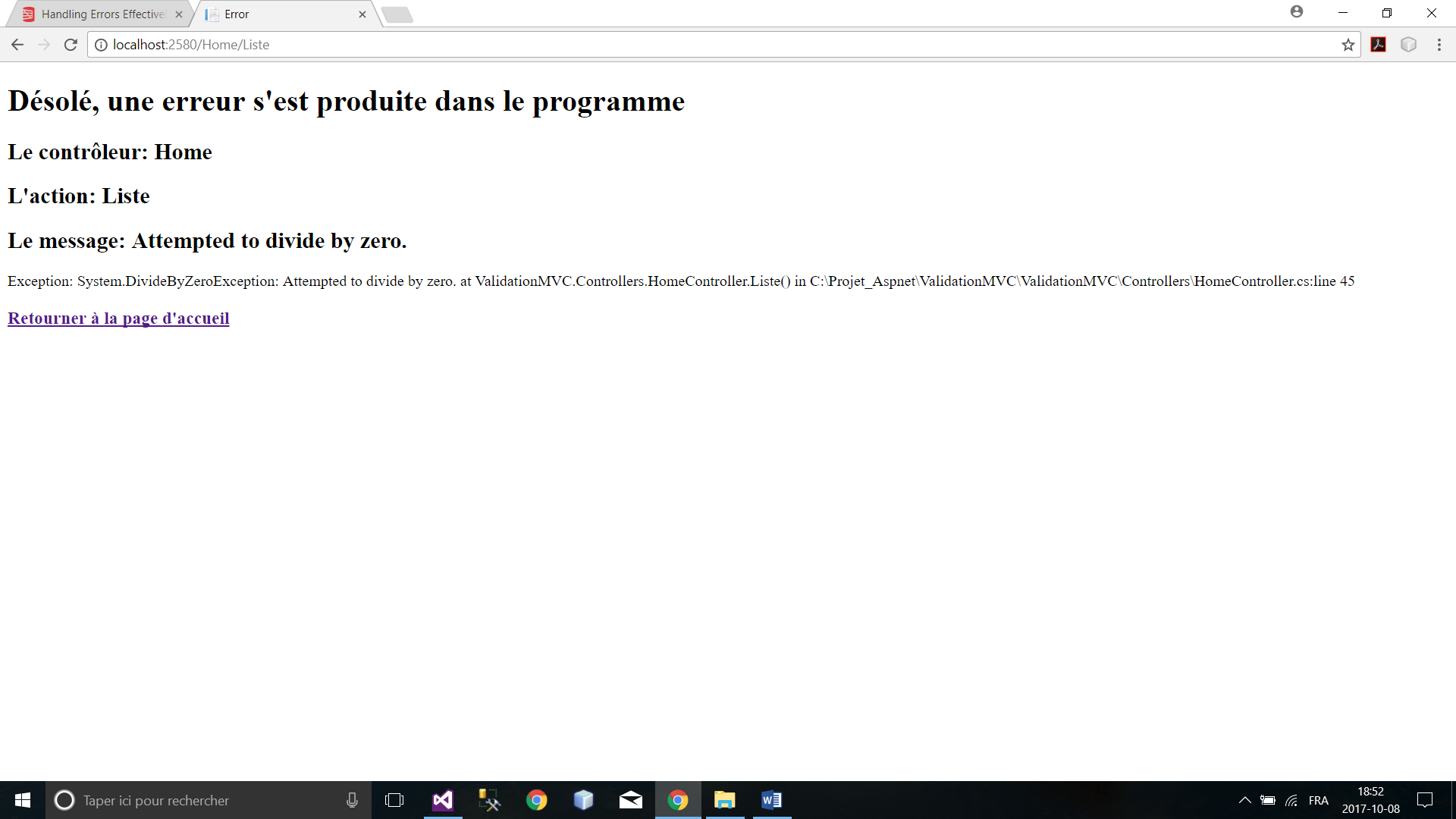
string actionName = route.Values["action"].ToString();

return **View**("Error", new ErrorInfoViewModel(ex.Message, controllerName, actionName));

}

}

Si vous essayez d'exécuter l'application maintenant, il donnera le résultat suivant. Le code ci-dessus rend l'erreur. Voir lorsqu'une exception se produit dans l'une des méthodes d'action dans ce contrôleur.



**Remarque** : L'avantage de cette approche est que plusieurs actions dans le même contrôleur peuvent partager cette logique de gestion des erreurs. Cependant, l'inconvénient est que nous ne pouvons pas utiliser la même logique de gestion des erreurs sur plusieurs contrôleurs.

1. **L'implémentation d'un filtre de type "ExceptionFilterAttribute"**

Cette approche est utilisée quand on veut gérer toutes les exceptions à travers les méthodes d'action au niveau **d'un contrôleur**.

Pour comprendre cette approche, créer une application core MVC (suivez les étapes couvertes dans les chapitres précédents).

1. Dans le dossier "Models", ajouter une classe nommée "MyGestException" qui dérive de la classe "**ExceptionFilterAttribute**". Dont voici le code :

public class MyGestException : **ExceptionFilterAttribute**

{

public override void **OnException**(ExceptionContext context)

{

var result = new ViewResult { ViewName = "CustomError" };

result.ViewData = new ViewDataDictionary(

new EmptyModelMetadataProvider(), context.ModelState);

result.ViewData["message"] = context.Exception.Message;

// ou on peut utiliser l'objet request pour envoyer des informations à la vue

//context.HttpContext.Items.Add("Source", context.Exception.Source);

context.Result = result;

context.ExceptionHandled = true;

}

}

1. Dans le contrôleur, ajouter à vos méthodes d'action l'attribut d'annotation avec votre gestionnaire d'exception.

[**MyGestException**]

public IActionResult Index()

{

throw new DivideByZeroException();

//return View();

}

1. La vue "CustomError.cshtml"

@{

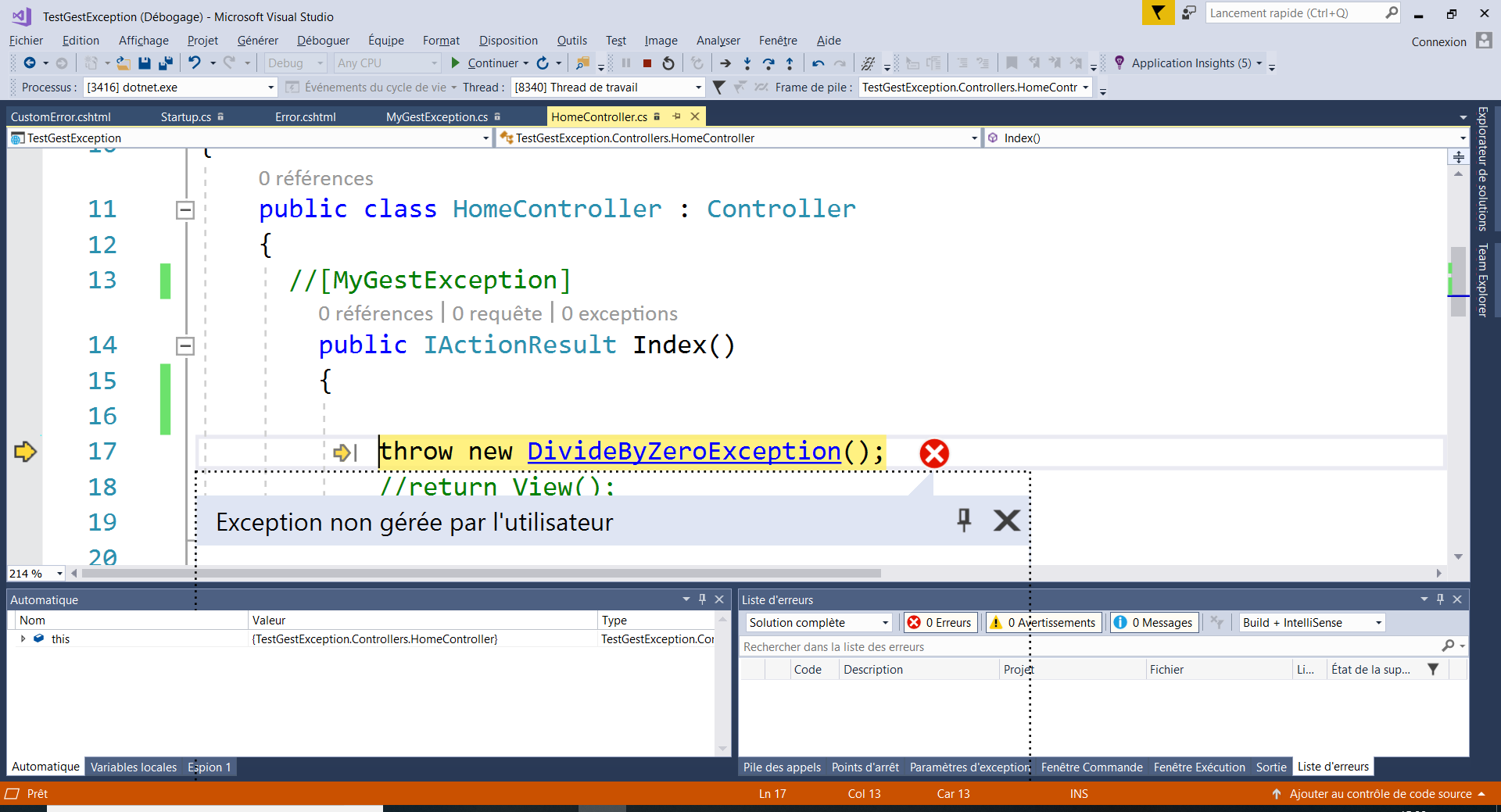
ViewData["Title"] = "Error";

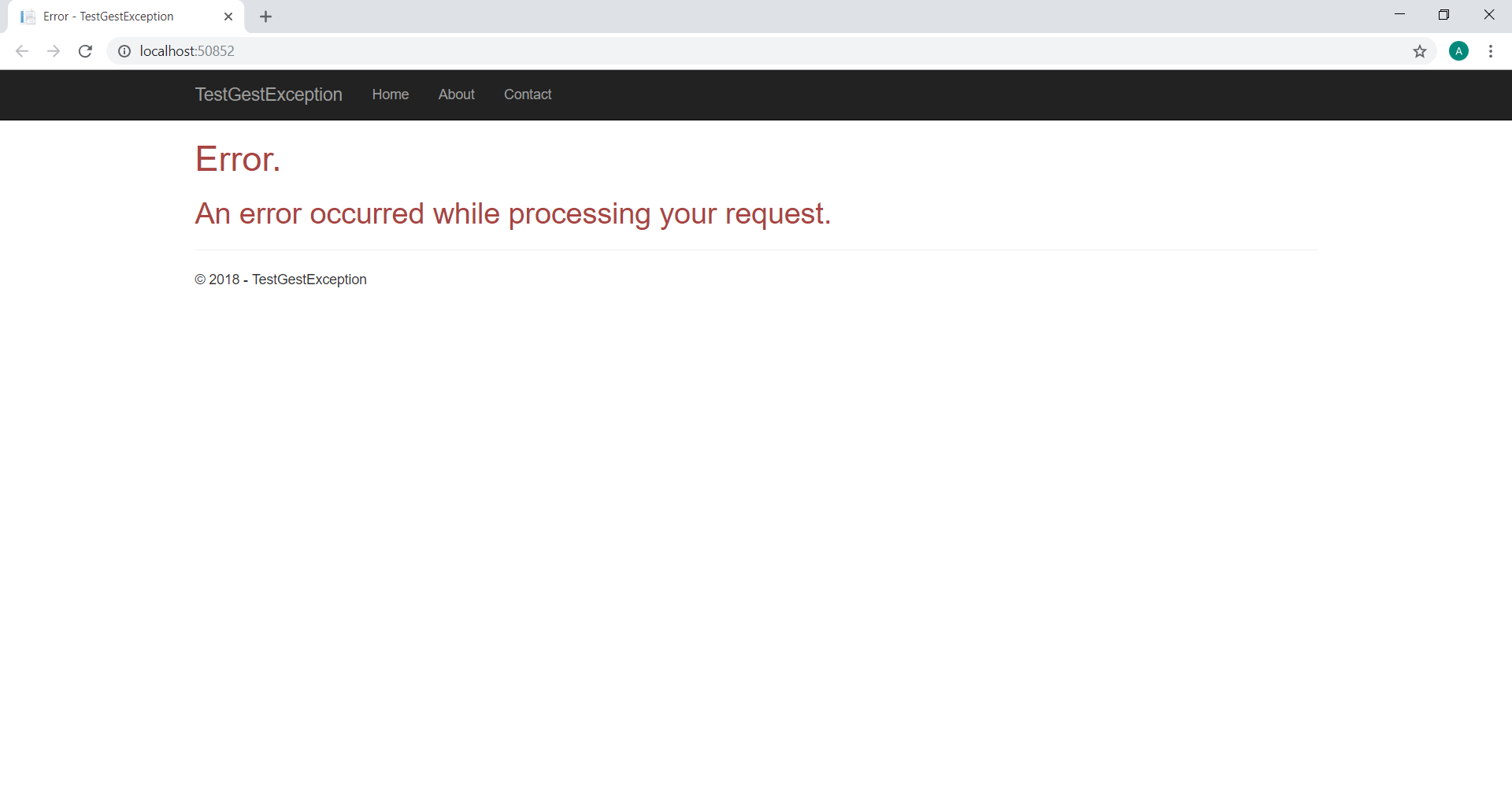
}

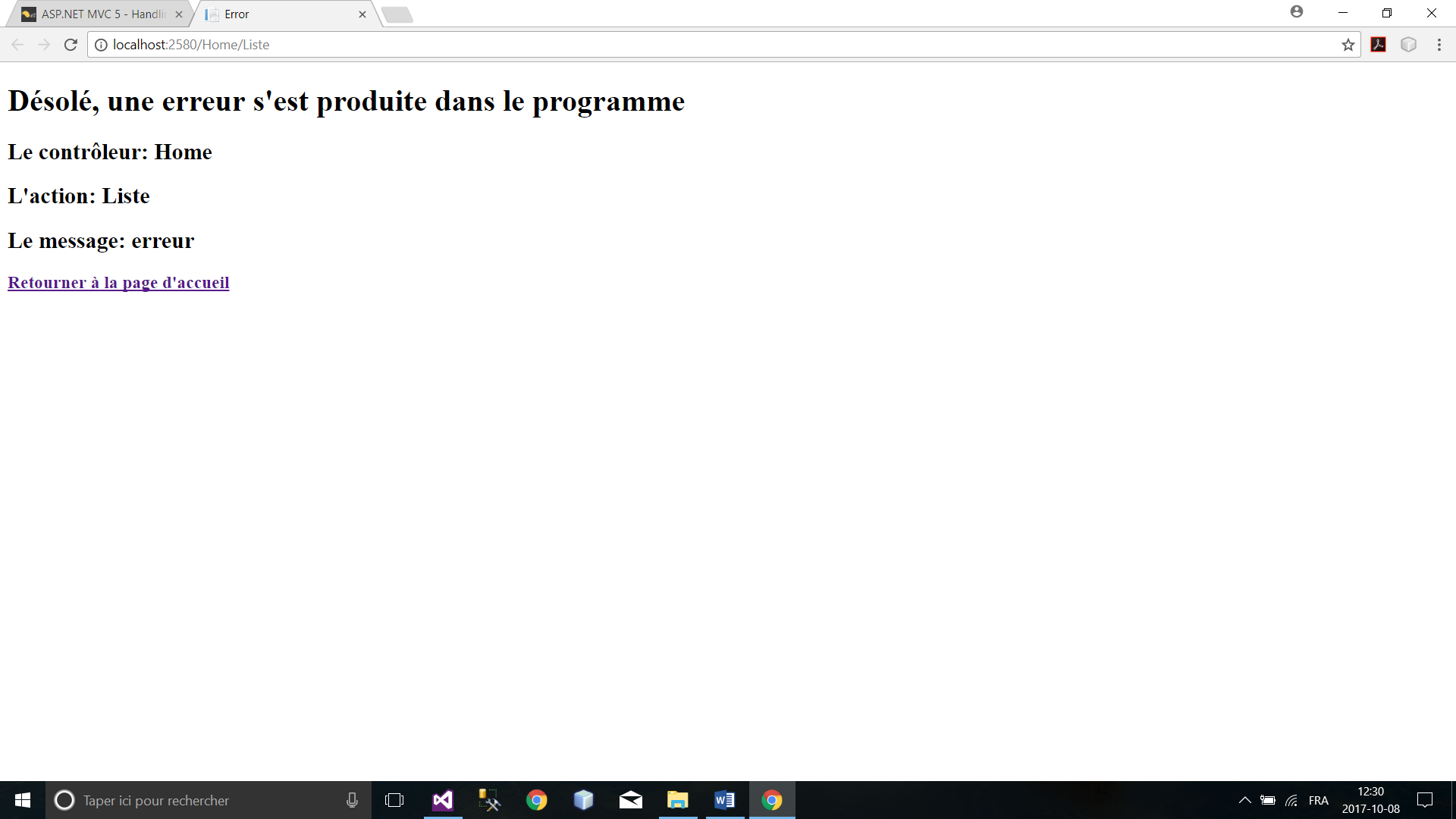
<h2 class="text-danger">Désolé un erreur c'est produite durant l'exécution de votre requete.</h2>

<h2 class="text-danger">Message :@ViewData["message"]</h2>

**Vous allez voir une erreur. Cliquez sur la touche "f5" pour afficher la page d'erreur**







1. **3) L'implémentation d'un filtre de type "IExceptionFilter"**

Cette approche est utilisée quand on veut gérer toutes les exceptions de façon globale à travers notre **application**.

Pour comprendre cette approche, créer une application core MVC (suivez les étapes couvertes dans les chapitres précédents).

1. Dans le dossier "Models", ajouter une classe nommée "**MyGestException**" qui implémentre l'interface "I**ExceptionFilter**". Dont voici le code :

public class MyGestException : I**ExceptionFilter**

{

public void **OnException**(ExceptionContext context)

{

var result = new ViewResult { ViewName = "CustomError" };

result.ViewData = new ViewDataDictionary(

new EmptyModelMetadataProvider(), context.ModelState);

result.ViewData["message"] = context.Exception.Message;

context.Result = result;

context.ExceptionHandled = true;

}

}

1. Configurer votre filtre globalement dans le fichier de configuration "statup.cs"

services.AddMvc(options =>

{

options.Filters.Add(new MyGestException());

});

**Remarque** : vous n'aurez pas besoin de garnir votre méthode avec l'attribut d'annotation de gestion d'exception [MyGestException]

**C) L'implémentation de la pagination dans asp.net MVC en utilisant la balise d'aide (TagHelper) "<pager>"**

**La classe ProduitDbContext :**

public static class ProduitDbContext

{

public static List<ProduitViewModel> ListeProduits = new List<ProduitViewModel>

{

new ProduitViewModel{Code=1, Description="Stylo",Prix=12},

new ProduitViewModel{Code=2, Description="Crayon",Prix=5},

new ProduitViewModel{Code=3, Description="Efface",Prix=3},

new ProduitViewModel{Code=4, Description="Cartable",Prix=25},

new ProduitViewModel{Code=5, Description="Papier",Prix=8},

new ProduitViewModel{Code=6, Description="Tablette",Prix=285}

};

}

**La classe ProduitViewModel :**

public class ProduitViewModel

{

public int Code { get; set; }

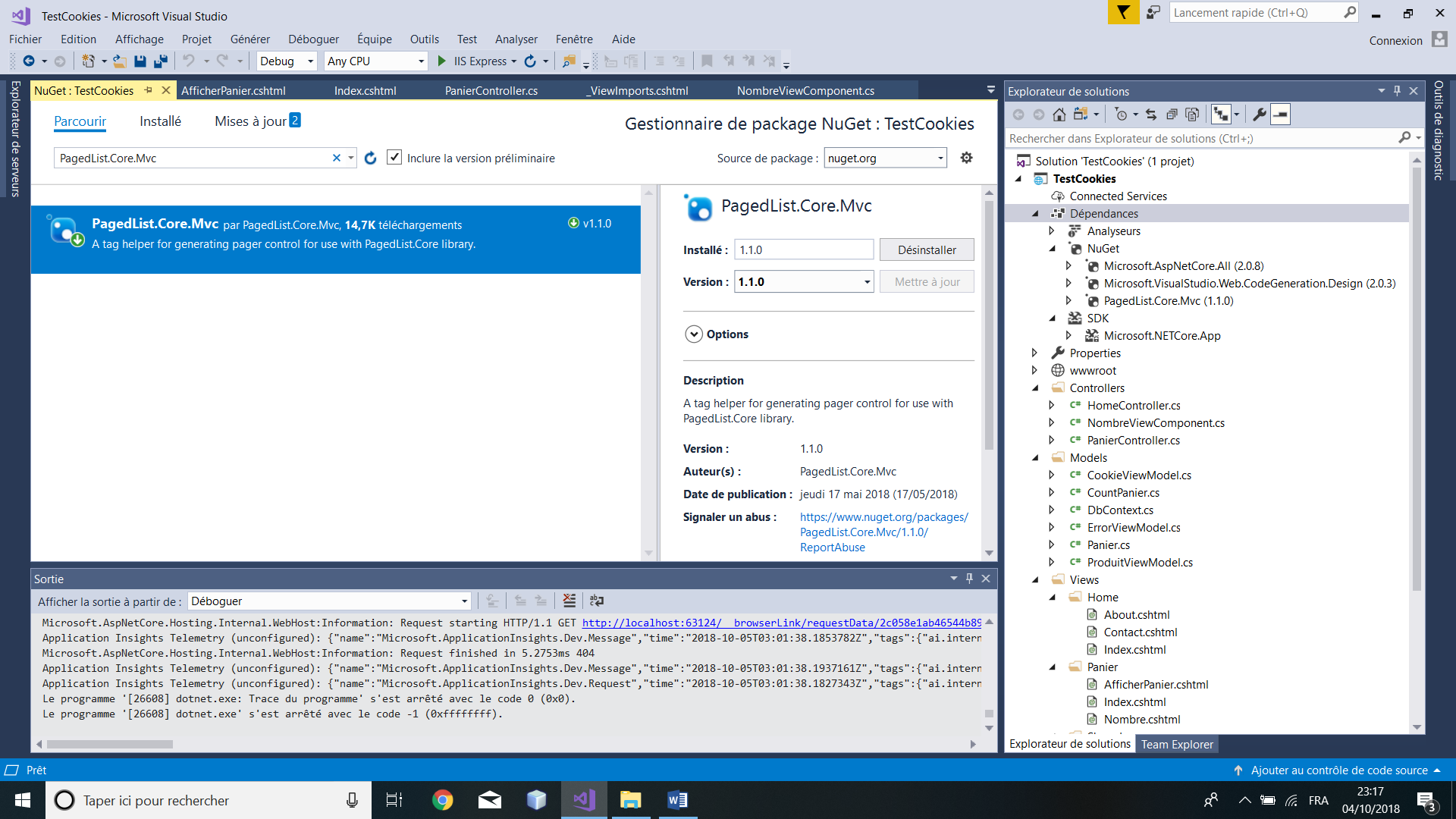
public string Description { get; set; }

public int Prix { get; set; }

}

**Étape 1:** installer "**PagedList.Core.Mvc" en utilisant le gestionnaire des packages** NuGet.

Pour le faire, cliquer droit sur le dossier "**Dépendances**" de votre projet MVC et sélectionner "Manage NuGet Packages". Dans le textbox de recherche, taper "**PagedList**.Core.Mvc" et cliquer sur installer.



**Étape 2 :** Ajouter le service de pagination en ajoutant le code suivant dans le fichier de configuration "Setup.cs" :

services.AddSingleton<IActionContextAccessor, ActionContextAccessor>();

**Étape 3** : Ajouter le TagHelper "PagedList.Core.MVc dans le fichier "ViewImports.cshtml".

@addTagHelper \*, PagedList.Core.Mvc

**Étape 4** : Ajouter les instructions suivantes à votre contrôleur "ProduitController.cs"

using PagedList.Core;  
  
Modifier le contenu de la méthode d'action "**Index**" comme il est montré ci-dessous:

* On ajoute un paramètre optionnel nommé "**page**" qui est utilisé pour spécifier le numéro de la page à afficher. Ce paramètre peut être null.
* Il faut noter que si le paramètre est null c'est la valeur 1 qui est passée à la méthode ToPagedList, sinon c'est la valeur du paramètre qui est passée.
* La méthode index doit envoyer une liste de type "**IPagedList**" à la vue. Donc on doit convertir la liste en un objet de type "**IQueryable**" puis à un objet de type "**ToPagedList**()"

Voici le code de la méthode "Index" du contrôleur.

public ActionResult Index(int? page)

{

var maliste = ProduitDbContext.ListeProduits.AsQueryable();

return View(maliste.ToPagedList(page ?? 1, 5));

}

**Étape 5 :** faites les changements suivant dans la vue "Index.cshtml"

1. Ajouter les deux lignes suivantes à votre vue.  
   @using PagedList.Core.Mvc;

@using PagedList.Core;

1. Le model de la vue doit être de type "**IPagedList**<Type>".

@model IPagedList<ProduitViewModel>

1. Changer la section qui affiche l'en-tête de la table comme montré ci-dessous  
   <tr>  
       <th>  
           @Html.DisplayNameFor(model => model.First().Name)  
       </th>  
       <th>  
           @Html.DisplayNameFor(model => model.First().Gender)  
       </th>  
       <th>  
           @Html.DisplayNameFor(model => model.First().Email)  
       </th>  
       <th>Action</th>  
   </tr>

**g) Insérer le contrôle de pagination à votre vue**

Ajouter la tag d'assistance de pagination nommé "**pager**" à la vue Index.

<**pager** class="pager-container" **list**="@Model" **options**="@PagedListRenderOptions.TwitterBootstrapPager" **asp-action**="Index" **asp-controller**="Panier" />

1. Utilisation de la dll JQuery "DataTable"

Insérer les fichiers suivants dans votre vue :

Pour les css : https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css

Pour les scripts :

https://code.jquery.com/jquery-3.3.1.js

https://cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js

Pour applique la méthode "DataTable" à votre balise "table" dont la propriété id=maTable, ajouter le script suivant :

$(document).ready( function () {

$('#maTable').DataTable();

} );