**Chapitre10**

**Ajax en MVC et Vues Partielles**

1. **Fonctionnalités du rendu de page partielle**

Une vue partielle est un fichier (.cshtml ) qui peut s'insérer dans un autre fichier de balisage (vue).

Le terme "vue partielle" est utilisé lors du développement d'une application MVC, dans laquelle les fichiers de marquage sont appelés des vues, ou d'une application Razor Pages, dans laquelle les fichiers de marquage sont appelés des pages.

**Quand utiliser des vues partielles**

Les vues partielles sont un moyen efficace pour:

* Découpez les gros fichiers de marquage en composants plus petits.
* Dans un fichier de balisage volumineux et complexe composé de plusieurs éléments logiques, il est avantageux de travailler avec chaque élément isolé dans une vue partielle.
* Réduisez la duplication du contenu du balisage commun entre les vues.
* Lorsque les mêmes éléments sont utilisés dans des fichiers de balisage "vue", une vue partielle supprime la duplication du contenu.

Remarque :

* Les vues partielles ne doivent pas être utilisées pour conserver des éléments de présentation communs. Les éléments de présentation communs doivent être placés dans les fichiers \_Layout.cshtml .
* Contrairement à la vue, une vue partielle n'exécute pas le fichier "\_ViewStart.cshtml".
* Les noms de fichier de vue partielle commencent souvent par un trait de soulignement ( \_).

**Référencer une vue partielle**

Remarque : Pour rendre une vue partielle dans une vue parent, vous pouvez utiliser l’une des deux fonctions**:**

* **@await Html.PartialAsync(..)**
* **@ { await Html.RenderPartialAsync(…) ;}**
* **<partial ../>**

**Exemple 1:**

1. Créer une base de données nommée "DbRoles" ;
2. Ajouter À cette base de données une table nommée "Roles"`;

CREATE TABLE [dbo].[Roles](

[Id] [nvarchar](10) NOT NULL,

[RoleName] [nvarchar](10) NULL,

[Description] [nvarchar](50) NULL,

[CreatedDate] [datetime] NULL,

CONSTRAINT [PK\_Roles] PRIMARY KEY ([Id] )

GO

* Insérer quelques rôles dans la tables "Roles"

1. Créer un nouveau projet core mvc nommé "TestAjaxPartialView";
2. Dans le fichier "appsettings.json, ajouter la chaine de connexion

"ConnectionStrings": {

"DbRoleConnection": "Data Source=localhost;Initial Catalog=DbRoles;User ID=sa; Password=sql" },

1. En utilisant la commande "scaffold-dbcontext", générer le contexte de donné nommé "ApplicationDbContext" ;
2. Dans le fichier configuration "Startup.cs", configurer votre DbContext dans le but de l'injection ;

services.AddDbContext<ApplicationDbContext>(options => options.UseSqlServer(Configuration.GetConnectionString("DbRoleConnection")));

1. Dans le dossier "Models", ajouter une classe nommée "RoleViewModel" ;

public class RoleViewModel

{

public string Id { get; set; }

public string RoleName { get; set; }

public string Description { get; set; }

}

1. Dans le dossier "Controllers", ajouter un contrôleur nommé "RoleController" ;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using TestAjaxCoreMVC.Models;

namespace TestAjaxCoreMVC.Controllers

{

public class RoleController : Controller

{

private readonly ApplicationDbContext \_dbcontext;

public RoleController(ApplicationDbContext dbcontext)

{

this.\_dbcontext = dbcontext;

}

public IActionResult Index()

{

List<RoleViewModel> modellisterole = new List<RoleViewModel>();

modellisterole = this.\_dbcontext.Roles.Select(r => new RoleViewModel

{

RoleName = r.RoleName,

Id = r.Id,

Description = r.Description

}).ToList();

ViewBag.ListRole = modellisterole;

ViewBag.Role = new RoleViewModel();

return View();

}

public IActionResult Add(RoleViewModel model)

{

if (ModelState.IsValid)

{

this.\_dbcontext.Roles.Add(new ApplicationRole()

{

Id =model.Id,

RoleName=model.RoleName,

Description=model.Description,

CreatedDate=DateTime.Now

});

this.\_dbcontext.SaveChanges();

}

return RedirectToAction(nameof(Index));

}

public IActionResult Delete(string id)

{

ApplicationRole role = this.\_dbcontext.Roles.Find(id);

if (role!=null)

{

this.\_dbcontext.Roles.Remove(role);

this.\_dbcontext.SaveChanges();

}

return RedirectToAction(nameof(Index));

}

}

}

1. Ajouter une vue partielle nommée "\_ListPartial.cshtml" basée sur le modèle "RoleViewModel";

@model IEnumerable<TestAjaxCoreMVC.Models.RoleViewModel>

<div class="top-buffer"></div>

<div class="panel panel-primary">

<div class="panel-heading panel-head">Gestion des Roles</div>

<div class="panel-body">

<div class="btn-group">

<a id="createRoleModal" href="#" class="btn btn-primary">

<i class="glyphicon glyphicon-plus"></i> Add Role

</a>

</div>

<div class="top-buffer"></div>

<table class="table table-bordered table-striped table-condensed">

<thead>

<tr>

<th>Id</th>

<th>Name</th>

<th>Description</th>

<th>Action</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>@item.Id</td>

<td>@item.RoleName</td>

<td>@item.Description</td>

<td>

<**a** id="addModal" **asp-action**="Add" **asp-route-id**="@item.Id" data-target="#modal-action"

class="btn btn-info">

<i class="glyphicon glyphicon-pencil"></i> Edit

</**a**>

<**a** id="deleteModal" **asp-action**="Delete" **asp-route-id**="@item.Id" data-target="#modal-action" class="btn btn-danger">

<i class="glyphicon glyphicon-trash"></i> Delete

</**a**>

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

1. Ajouter une vue partielle nommée "\_AddPartial.cshtml" basée sur le modèle "RoleViewModel";

@model RoleViewModel

@using TestAjaxCoreMVC.Models

<div class="panel panel-primary">

<div class="panel-heading panel-head">Ajouter des Roles</div>

<**form** id= "addForm" **asp-action**="Add" role="form" method="post">

<div class="modal-body form-horizontal">

<div class="row">

<div class="form-group">

<**label** **asp-for**="Id" class="col-lg-3 col-sm-3 control-label"></**label**>

<div class="col-lg-6">

<**input** **asp-for**="Id" class="form-control" />

</div>

</div>

<div class="form-group">

<**label** **asp-for**="RoleName" class="col-lg-3 col-sm-3 control-label"></**label**>

<div class="col-lg-6">

<**input** **asp-for**="RoleName" class="form-control" />

</div>

</div>

<div class="form-group">

<**label** **asp-for**="Description" class="col-lg-3 col-sm-3 control-label"></**label**>

<div class="col-lg-6">

<**input** **asp-for**="Description" class="form-control" />

</div>

</div>

<div class="form-group">

<button id="btCancel"type="reset" class="btn btn-primary">Cancel</button>

<button id ="btSubmit" type="submit" class="btn btn-primary">Submit</button>

<div class="col-lg-6">

</div>

</div>

</div>

</div>

</**form**>

</div>

1. Ajouter une vue nommée "Index.cshtml" vide ;

<div id="divliste">

<**partial** **name**="\_ListPartial" **model**="ViewBag.ListRole" />

</div>

<div id="divaddmodal">

<**partial** **name**="\_AddPartial" **model**="ViewBag.Role" />

</div>

@section Scripts

{

<script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>

<**script** **src**="~/js/site.js" **asp-append-version**="true"></**script**>

}

1. Dans le fichier "site.js", ajouter le code JavaScript suivant :

$(function () {

var formModal = fndefinirModal();

$("#createRoleModal").on("click", function () { $(formModal).dialog("open"); });

$("#btSubmit").on("click", function () { $(formModal).dialog("close"); });

$("#btCancel").on("click", function () { $(formModal).dialog("close"); });

});

function fndefinirModal() {

var formModal = $("#divaddmodal").dialog({

autoOpen: false,

height: 400,

width: 350,

modal: true,

show: "blind",

hide: "drop",

close: function () {

formModal.dialog("close");

}

});

return formModal;

}

**Exemple 2 :**

**Le contrôleur :**

public ActionResult Index()

{

NorthwindEntities db = new NorthwindEntities();

ViewBag.Customers = db.Customers.ToList();

ViewBag.Orders = db.Orders.ToList().OrderBy(c=>c.CustomerID).ToList();

return View();

}

**La vue Index :**

@using WebApplication2.Models

<!DOCTYPE html>

<html>

<head>

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

<title>Index</title>

</head>

<body>

<div>

<h2>La liste des clients</h2>

<div id="listeclients" style="height:500px; overflow:scroll">

@{await Html.RenderPartialAsync("Customers",(List<Customers>) ViewBag.Customers); }

</div>

<h2>La liste des factures</h2>

<div id="listefactures" style="height:400px; overflow:scroll">

@await Html.PartialAsync("Orders",(List<Orders>) ViewBag.Orders)

</div>

</div>

</body>

</html>

**La vue partielle liste des clients**

@model IEnumerable<WebApplication2.Models.Customers>

<table class="table">

<tr>

<th>

@Html.DisplayNameFor(model => model.CustomerID)

</th>

<th>

@Html.DisplayNameFor(model => model.CompanyName)

</th>

<th>

@Html.DisplayNameFor(model => model.ContactName)

</th>

<th>Opération</th>

</tr>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.CustomerID)

</td>

<td>

@Html.DisplayFor(modelItem => item.CompanyName)

</td>

<td>

@Html.DisplayFor(modelItem => item.ContactName)

</td>

<th></th>

</tr>

}

</table>

**La vue partielle la liste des commandes**

@model IEnumerable<WebApplication2.Models.Orders>

<table class="table">

<tr>

<th>

@Html.DisplayNameFor(model => model.Customers.CompanyName)

</th>

<th>

@Html.DisplayNameFor(model => model.EmployeeID)

</th>

<th>

@Html.DisplayNameFor(model => model.OrderDate)

</th>

<th>

@Html.DisplayNameFor(model => model.RequiredDate)

</th>

</tr>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.Customers.CompanyName)

</td>

<td>

@Html.DisplayFor(modelItem => item.EmployeeID)

</td>

<td>

@Html.DisplayFor(modelItem => item.OrderDate)

</td>

<td>

@Html.DisplayFor(modelItem => item.RequiredDate)

</td>

</tr>

}

1. **Utilisation d’Ajax avec les vues partielles**

Ajax est un raccourci pour Asynchronous JavaScript et XML.

**Remarque :**

Utilisez des fonctionnalités AJAX lorsque vous souhaitez effectuer les opérations suivantes :

* Améliorer l'expérience utilisateur avec des pages Web qui sont plus riches, qui sont plus réactives aux actions utilisateur et qui se comportent comme des applications clientes traditionnelles ;
* Limiter l'actualisation de pages entières et éviter le scintillement des pages ;
* Assurer la compatibilité entre les navigateurs sans écrire de script client ;
* Établir une communication client/serveur de style AJAX sans écrire de script client ;
* Utiliser des composants d'ASP.NET MVC AJAX ;
* Utilisation d'AJAX de JQuery.

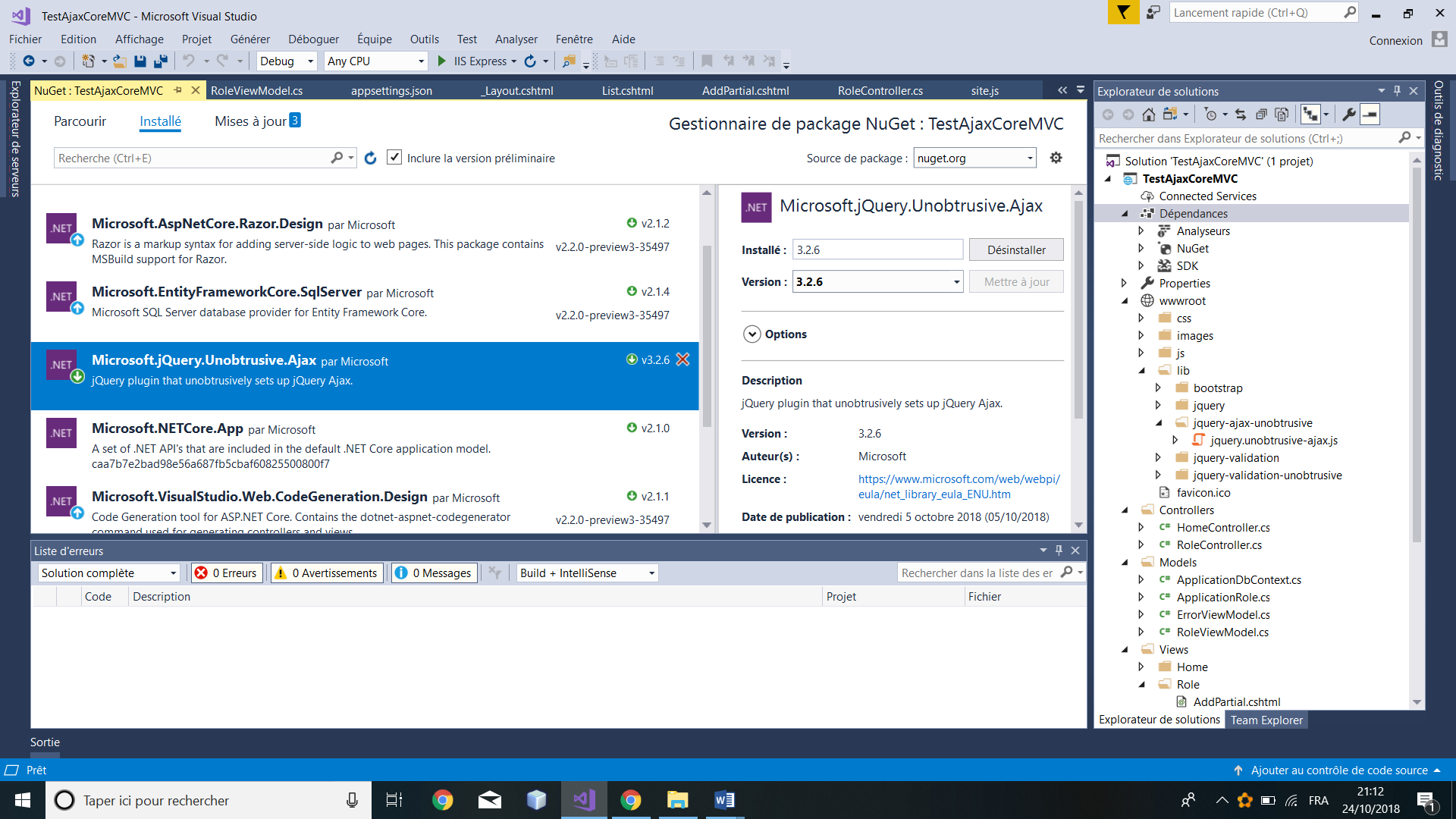
1. **Utilisation d'Ajax du CORE MVC**

Le MVC Framework contient un support intégré pour Ajax par lequel vous pouvez utiliser les méthodes d'assistance pour définir vos fonctions Ajax sans ajout de code dans toutes les vues. Cette fonctionnalité dans MVC est basée sur les caractéristiques jQuery.

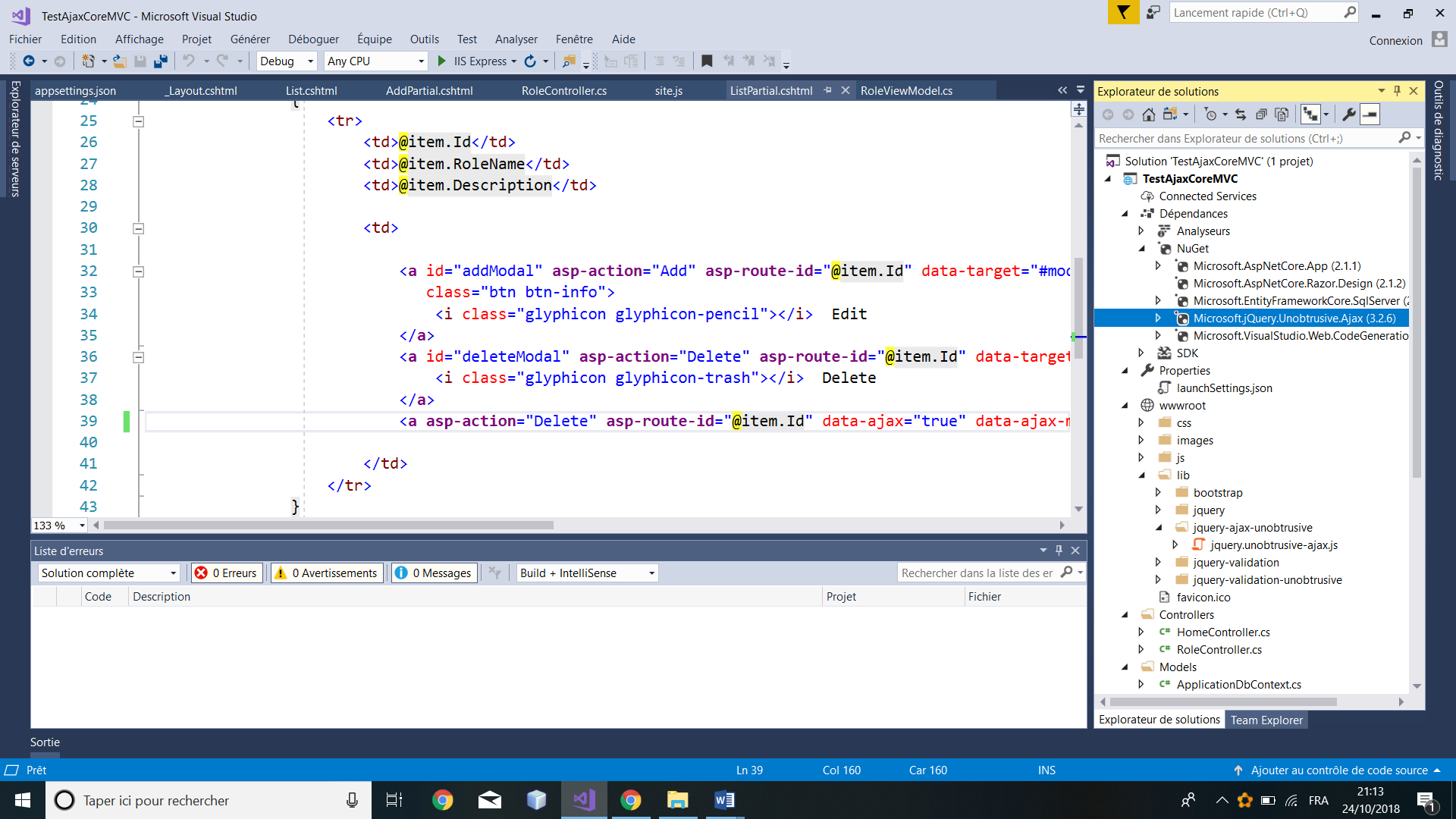
**Remarque** : Pour activer le support AJAX dans l'application Core MVC suivez les étapes suivantes :

1. Dans la console de gestion des package NuGet, installer le fichier :

"Microsoft.JQuery.Unobtrusive.Ajax". Cliquer droit sur "Dependances" puis "Gérer les packages NuGet". Et puis dans le textbox de recherche, taper le texte "Microsoft.JQuery.Unobtrusive.Ajax" puis cliquer sur installer.



Il ajoute le package "Microsoft.jQuery.Unobtrusive.Ajax" au projet



Remarque : Pour utiliser les attributs "data-ajax-\*", vous aurez besoin du script "jquery.unobtrusive-ajax.js". Vous pouvez installer ce script en utilisant le gestionnaire de package nommé "bower" que vous devez l'installer s'il n'est pas déjà installé. Voici la commande à taper sur la ligne de commande:

* Pour installer bower : install -g bower
* Pour installer le script : bower install Microsoft.jQuery.Unobtrusive.Ajax.

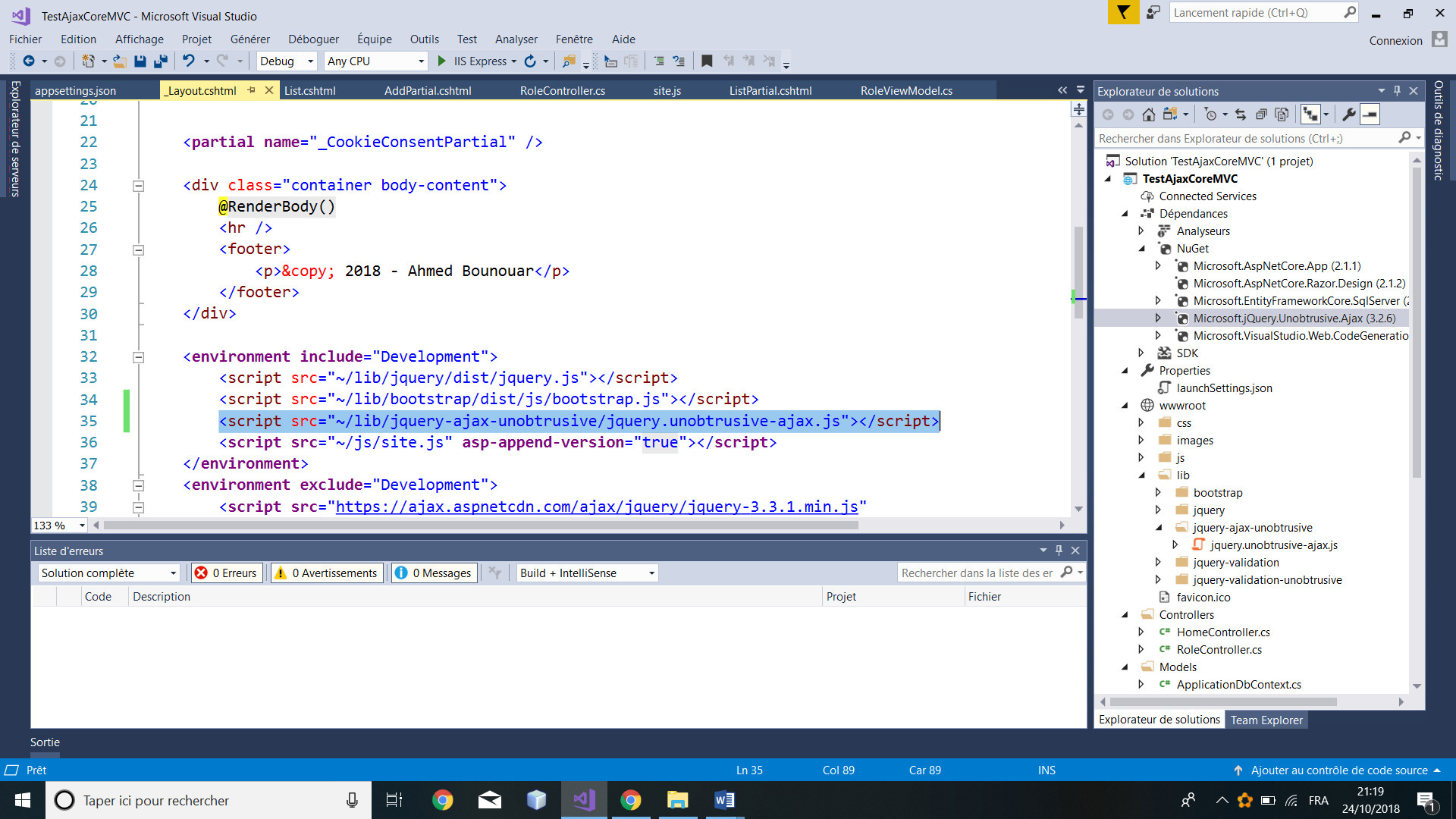
C:\Users\info1\Source\Repos\TestVSTS\TestAjaxMVC\packages\Microsoft.jQuery.Unobtrusive.Ajax.3.2.3\Content\Scripts\

1. Créer un sous répertoire nommé "jquery-unobtrusive-ajax" dans le dossier "wwwroot/lib/
2. Copier le script "jquery.unobtrusive-ajax.js" dans le dossier "jquery-unobtrusive-ajax". Ce script se trouve dans le dossier :

C:\Users\info1\Source\Repos\TestVSTS\TestAjaxMVC\packages\Microsoft.jQuery.Unobtrusive.Ajax.3.2.3\Content\Scripts\jquery.unobtrusive-ajax.js

1. Ajouter le fichier "" dans le fichier "\_Layout.cshtml" :

<script src="~/lib/jquery-ajax-unobtrusive/jquery.unobtrusive-ajax.js"></script>



**Remarque** : pour utiliser la fonctionnalité d'ajax au balises (tags), core mvc a ajouté d'autres attributs qu'on peut les utiliser. Voici le tableau des attributs :

| **Attribute** | **Description** |
| --- | --- |
| data-ajax | Doit être égale à "true" pour active "unobtrusive Ajax" sur l'élément cible. |
| data-ajax-confirm | Gets or sets the message to display in a confirmation window before a request is submitted. |
| data-ajax-method | Gets or sets the HTTP request method ("Get" or "Post"). |
| data-ajax-mode | Gets or sets the mode that specifies how to insert the response into the target DOM element. Valid values are before, after and replace. Default is replace |
| data-ajax-loading-duration | Gets or sets a value, in milliseconds, that controls the duration of the animation when showing or hiding the loading element. |
| data-ajax-loading | Gets or sets the id attribute of an HTML element that is displayed while the Ajax function is loading. |
| data-ajax-begin | Gets or sets the name of the JavaScript function to call immediately before the page is updated. |
| data-ajax-complete | Gets or sets the JavaScript function to call when response data has been instantiated but before the page is updated. |
| data-ajax-failure | Gets or sets the JavaScript function to call if the page update fails. |
| data-ajax-success | Gets or sets the JavaScript function to call after the page is successfully updated. |
| data-ajax-update | Gets or sets the ID of the DOM element to update by using the response from the server. |
| data-ajax-url | Gets or sets the URL to make the request to. |

Exemple:

* Le cas d'un lien

<a data-ajax-url="Role/Delete?id=@item.Id" data-ajax="true" data-ajax-method="GET" data-ajax-mode="replace" data-ajax-update= "#divliste"> supprimer Ajax2</a>

* Le cas d'un formulaire

<form asp-controller="Home" asp-action="SaveForm"

data-ajax-begin="onBegin" data-ajax-complete="onComplete"

data-ajax-failure="onFailed" data-ajax-success="onSuccess"

data-ajax="true" data-ajax-method="POST">

<input type="submit" value="Save" class="btn btn-primary" />

<div id="Results"></div>

</form>

1. Pour ajouter la fonction Ajax à notre exemple, Faites les modifications suivantes**:**
2. Dans le contrôleur "RoleController", ajouter la méthode "List" définie comme suivant :

public IActionResult List()

{

List<RoleViewModel> modellisterole = new List<RoleViewModel>();

modellisterole = this.\_dbcontext.Roles.Select(r => new RoleViewModel

{

RoleName = r.RoleName,

Id = r.Id,

Description = r.Description

}).ToList();

return PartialView("\_ListPartial", modellisterole);

}

1. Modifier la méthode "Add"

public IActionResult Add(RoleViewModel model)

{

if (ModelState.IsValid)

{

this.\_dbcontext.Roles.Add(new ApplicationRole()

{

Id =model.Id,

RoleName=model.RoleName,

Description=model.Description,

CreatedDate=DateTime.Now

});

this.\_dbcontext.SaveChanges();

}

return RedirectToAction(nameof(List));

}

1. Modifier la balise "<form>" de la vue "\_AddPartial"

<**form** **asp-controller**="Role" **asp-action**="Add" data-ajax-mode="replace" data-ajax-update="#divliste" data-ajax="true" data-ajax-method="POST" data-ajax-complete="onComplete" >

1. Modifier la balise "<a>" pour l'élément supprimer de la vue "\_ListPartial"

<**a** id="deleteModal" **asp-action**="Delete" **asp-route-id**="@item.Id" data-ajax="true" data-ajax-method="GET" data-ajax-mode="replace" data-ajax-update="#divliste" class="btn btn-danger">

<i class="glyphicon glyphicon-trash"></i> Delete

</**a**>

1. Dans le fichier "site.js", ajouter le code suite :

function oncomplete() {

var formModal = fndefinirModal();

$("#createRoleModal").on("click", function () { $(formModal).dialog("open"); });

$("form input[type="text"]).prop("value","");

}

1. **Utilisation d’Ajax de JQuery**
2. **Ajax et JQuery**

JQuery fournit un ensemble de méthodes pour interroger le serveur en mode asynchrone et qui sont simples à utiliser.

Syntaxe: **JQuery.ajax**(settings)

**settings**

Type: [PlainObject](http://api.jquery.com/Types/" \l "PlainObject)

C'est un objet composé d'un ensemble d'éléments de type "key/value" paire qui configure la requête d'Ajax. Dont voici quelques-unes :

* **url** (default: la page courante)

Type: [String](http://api.jquery.com/Types/#String)

C'est une chaine de caractères qui contient l'URL de la page à appeler.

* **type** (default: 'GET')

Type: [String](http://api.jquery.com/Types/#String)

Le type de la requête ("POST" or "GET").

* **data**

Type: [PlainObject](http://api.jquery.com/Types/" \l "PlainObject) or [String](http://api.jquery.com/Types/#String)

Les données à envoyer au serveur avec la requête.

Remarque:

Les données peuvent contenir une chaine de caractère sous la forme (key1=value1&key2=value2) en utilisant la méthode "param" de jquery "**jQuery.param"**

ou un objet sous la forme **Json** {key1: 'value1', key2: 'value2'} qui doit être converti en chaine en utilisant la méthode [jSON.stringify()](http://api.jquery.com/jQuery.param/) avant d'être envoyée.

* **contentType** (default: 'application/x-www-form-urlencoded; charset=UTF-8')

Type: [String](http://api.jquery.com/Types/#String)

**error**

Type: [Function](http://api.jquery.com/Types/#Function)( [jqXHR](http://api.jquery.com/Types/#jqXHR) jqXHR, [String](http://api.jquery.com/Types/#String) textStatus, [String](http://api.jquery.com/Types/#String) errorThrown )

C'est la fonction à appeler dans la cas d'un échec

**success**

Type: [Function](http://api.jquery.com/Types/#Function)( [PlainObject](http://api.jquery.com/Types/" \l "PlainObject) data, [String](http://api.jquery.com/Types/#String) textStatus, [jqXHR](http://api.jquery.com/Types/" \l "jqXHR) jqXHR )

C'est la fonction à appeler dans le cas d'un succès.

Exemple

**var** request = $.ajax({

url: "/Home/Index",

type: "POST",

data: { id : menuId },

dataType: "json"

});

Exemple : l'envoi d'un objet en format JSON

function updateajaxJson() {

var cid = $("#DropDownList1").val();

**var client={"Customerid":cid, "Nom":"Ahmed"} // objet JSON**

$.**ajax**({

type: "POST",

url: "/Employe/Add",

**dataType: "**text/plain**",**

**contentType: "application/json; charset=UTF-8"**,

**data**: **JSON.stringify**(client),

async: true,

success: function (data, textStatus) {

if (textStatus == "success") {

$("#divlistview").html(data);

}

},

error: function (data, status, error) {

alert("error");

}

});

}

Exemple : l'envoi d'un objet JSON sous forme d'un QueryString

function updateajaxParam() {

var cid = $("#DropDownList1").val();

var client = { "Customerid": cid, "Nom": "Ahmed" };

$.ajax({

type: "**GET**",

url: "/Customers/Add",

data: **jQuery.param(client)**,

async: true,

success: function (data, textStatus) {

if (textStatus == "success") {

$("#divlistview").html(data);

}

},

error: function (data, status, error) {

alert("error");

}

});

}

La réception de l'objet du côté serveur.

1. La réception de l'objet du coté serveur en utilisant la propriété Query.

string code = Request.**Query**["Customerid"];

string nom = Request.**Query**["Nom"];

1. La lecture des données à partir du corps de la requête http, puis l'utilisation de la méthode "**Deserialize**" de la classe "**JavaScriptSerializer**" ou la méthode "**DeserializeObject**" de la classe "JsonConvert" pour désérialiser l'objet.

StreamReader reader = new StreamReader(Request.InputStream);

string data=reader.ReadToEnd();

JavaScriptSerializer jss = new **JavaScriptSerializer**();

Client user = **jss.Deserialize<Client>(data)**;

ou Client user = **JsonConvert.DeserializeObject**<Client>(data);

1. Utilisation de l'attribut d'annotation [**FromBody**] ou [**FromForm**]

public IActionResult Add**([FromForm]**RoleViewModel model)

public IActionResult Add **([FromBody**]RoleViewModel model)

1. **L'objet JqXHR**

Dans un souci de compatibilité avec l'objet "XMLHttpRequest", l'objet "jqXHR" fournit les propriétés et méthodes suivantes:

* readyState
* status
* statusText
* responseXML et  responseText
* setRequestHeader(name, value)
* getAllResponseHeaders()
* getResponseHeader()
* statusCode()
* abort()

**Les méthodes:**

* **jqXHR.done(function( data, textStatus, jqXHR ) {});**

C'est une alternative à l'option "**success**" de la méthode "**JQuery**. **Ajax**".

* **jqXHR.fail(function( jqXHR, textStatus, errorThrown ) {});**

C'est une alternative à l'option "**error**" de la méthode "**JQuery**.**ajax**".

1. **Implémentation d'Ajax de JQuery dans notre application :**

**Étape 1 :**

Dans la vue "\_ListePartial", faites les modifications suivantes :

* Pour la balise <table>, ajouter un id :

<table id="listeTable" class="table table-bordered table-striped table-condensed">

* **Pour la balise <tr> jouter un id :**

<tr id="@item.Id">

* **Pour la balise <a>, ajouter un événement click:**

<a class="btn btn-danger" onclick='fndelete("@item.Id")'>

<i class="glyphicon glyphicon-trash"></i> Delete

</a>

Dans la vue "\_AddPartial", modifier la balise form comme : <**form**>

**Étape 2 : Dans le fichier "site.js", ajouter le code suivant:**

var formModal;

$(function () {

formModal = fndefinirModal();

$("#createRoleModal").on("click", function () {

$(formModal).dialog("open");

$("form").prop("title", "Add");

});

$("#btSubmit").on("click", function () {

$(formModal).dialog("close");

$("form").prop("title")==="Add"?fnAddAjax():fneditPost();

});

$("#btCancel").on("click", function () { $(formModal).dialog("close");});

});

function fndefinirModal() {

var formModal = $("#divaddmodal").dialog({

autoOpen: false,

height: 400,

width: 350,

modal: true,

close: function () {

formModal.dialog("close");

}

});

return formModal;

}

function fnAddAjax() {

var url ="/Role/Add";

var data = {

id: $("#Id").val(),

RoleName: $("#RoleName").val(),

Description: $("#Description").val()

};

$.ajax({

data: JSON.stringify(data),

//data: $("form").serialize(),

type: "POST",

url: url,

datatype: "text/plain",

contentType: "application/json; charset=utf-8",

//contentType : "application/x-www-form-urlencoded; charset=utf-8",

success: function (result) {

alert(result);

var tr = "<tr id=" + $("#Id").val()+"><td>" + $("#Id").val() + "</td><td>" + $("#RoleName").val() + "</td><td>" + $("#Description").val() + "</td>" +

"<td>" +

"<a class=\"btn btn-info\" onclick=\"fneditGet("+$("#Id").val()+")\"> <i class=\"glyphicon glyphicon-pencil\"></i> Edit </a>" +

" <a class=\"btn btn-danger\" onclick=\"fndelete("+$("#Id").val()+")\"><i class=\"glyphicon glyphicon-trash\"></i>Delete</a>"+

"</td></tr >";

$("#listeTable").append(tr);

},

error: function (xhr, status) { alert("erreur:" + status); }

});

}

function fndelete(roleId) {

$.ajax({

// data: jQuery.param({ id: roleId }),

type: "GET",

url: "/Role/Delete/"+roleId,

datatype: "text/plain",

contentType: "application/html; charset=utf-8",

success: function (result) {

alert(result);

$("#listeTable tr[id=\""+roleId+"\"]").remove();

},

error: function (xhr, status) { alert("erreur:" + status); }

});

}

function fneditGet(roleId) {

$.ajax({

// data: jQuery.param({ id: roleId }), // envoyer le paramètre dans le querystring

type: "GET",

url: "/Role/Edit/"+roleId, // envoyer le paramètre dans l'uri

contentType: "application/html; charset=utf-8",

datatype: "text/plain",

success: function (result) {

$("form").prop("title", "Edit");

formModal.dialog("open");

$("#Id").val(result.id);

$("#RoleName").val(result.roleName);

$("#Description").val(result.description);

},

error: function (xhr, status) { alert("erreur:" + status); }

});

}

function fneditPost(roleId) {

var data = {

id: $("#Id").val(),

RoleName: $("#RoleName").val(),

Description: $("#Description").val()

};

$.ajax({

data: JSON.stringify(data),

type: "POST",

url: "/Role/Edit/",

contentType: "application/json; charset=utf-8",

datatype: "text/plain",

success: function (result, textStatus) {

alert(textStatus + ":" + result);

var tr = "<td>" + $("#Id").val() + "</td><td>" + $("#RoleName").val() + "</td><td>" + $("#Description").val() + "</td>" +

"<td>" +

"<a class=\"btn btn-info\" onclick=\"fneditGet(" + $("#Id").val() + ")\"> <i class=\"glyphicon glyphicon-pencil\"></i> Edit </a>" +

" <a class=\"btn btn-danger\" onclick=\"fndelete(" + $("#Id").val() + ")\"><i class=\"glyphicon glyphicon-trash\"></i>Delete</a>" +

"</td>";

$("#listeTable tr[id=\"" + $("#Id").val() + "\"]").html(tr);

},

error: function (xhr, status) { alert("erreur:" + status); }

});

}

**Étape** **3** : Dans le contrôleur "RoleController", faites les modifications suivantes:

1. La méthode "Add"

Version 1: La réception des données envoyées, à partir du flux d'entrée de la requête http

[HttpPost]

public IActionResult Add()

{

MemoryStream stream = new MemoryStream();

Request.Body.CopyTo(stream);

stream.Position = 0;

using (StreamReader reader = new StreamReader(stream))

{

string requestBody = reader.ReadToEnd();

if (requestBody.Length > 0)

{

var model = JsonConvert.DeserializeObject<RoleViewModel>(requestBody);

this.\_dbcontext.Roles.Add(new ApplicationRole()

{

Id = model.Id,

RoleName = model.RoleName,

Description = model.Description,

CreatedDate = DateTime.Now

});

this.\_dbcontext.SaveChanges();

return Ok("élément ajouté avec succès");

}

return BadRequest("élément inexistant");

}

}

Version 2: La réception des données envoyées sous format JSON

[HttpPost]

public IActionResult Add([FromBody]RoleViewModel model)

{

if(model!=null)

{

this.\_dbcontext.Roles.Add(new ApplicationRole()

{

Id = model.Id,

RoleName = model.RoleName,

Description = model.Description,

CreatedDate = DateTime.Now

});

this.\_dbcontext.SaveChanges();

return Ok("élément ajouté avec succès");

}

return BadRequest("élément non ajouté");

}

Version 3: La réception des données envoyées dans le corps de la requête http

[HttpPost]

public IActionResult Add([FromForm]RoleViewModel model)

{

if(model!=null)

{

this.\_dbcontext.Roles.Add(new ApplicationRole()

{

Id = model.Id,

RoleName = model.RoleName,

Description = model.Description,

CreatedDate = DateTime.Now

});

this.\_dbcontext.SaveChanges();

return Ok("élément ajouté avec succès");

}

return BadRequest("élément non ajouté");

}

1. La méthode "Delete"

[HttpGet]

public IActionResult Delete(string id)

{

ApplicationRole role = this.\_dbcontext.Roles.Find(id);

if (role!=null)

{

this.\_dbcontext.Roles.Remove(role);

this.\_dbcontext.SaveChanges();

return Ok("élément supprimé avec succès");

}

return BadRequest("élément inexistant");

}

[HttpGet]

public IActionResult Edit(string id)

{

ApplicationRole role = this.\_dbcontext.Roles.Find(id);

return Json(role);

}

[HttpPost]

public IActionResult Edit([FromBody]RoleViewModel model) {

if (ModelState.IsValid)

{

ApplicationRole role = this.\_dbcontext.Roles.Find(model.Id);

role.RoleName = model.RoleName;

role.Description = model.Description;

this.\_dbcontext.SaveChanges();

return Ok("élément modifié avec succès");

}

return BadRequest("Erreur de modification");

}

**Exemple 3** : Une application qui met à jour une division (Nombre de produits dans le panier) de façon asynchrone.

Le contenu de la page “\_Layout”

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

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

<title>@ViewBag.Title</title>

@Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr")

<script src="~/scripts/jquery-1.8.2.min.js" type="text/javascript"></script>

<script src="~/scripts/jquery.unobtrusive-ajax.min.js" type="text/javascript" ></script>

</head>

<body>

@RenderBody()

@Scripts.Render("~/bundles/jquery")

@RenderSection("scripts", required: false)

</body>

</html>

Le contenu de La classe “Panier”

public class Panier

{

public List<int> ListeProduits { get; set; }

public Panier()

{

ListeProduits = new List<int>();

}

public int AjouterProduit(int id)

{

this.ListeProduits.Add(id);

return this.ListeProduits.Count;

}

public static Panier GetPanier()

{

Panier p;

if (HttpContext.Current.Session["panier"] != null)

{

p = HttpContext.Current.Session["panier"] as Panier;

}

else

{

p = new Panier();

}

return p;

}

public static void SavePanier(Panier p)

{

HttpContext.Current.Session["panier"]=p;

}

}

Le contenu de La classe Produitcontroller

public class ProduitController : Controller

{

private NORTHWNDEntities db = new NORTHWNDEntities();

//

// GET: /Produit/

public ActionResult Index()

{

var products = db.Products.Include(p => p.Categories);

return View(products.ToList());

}

//

// GET: /Produit/Details/5

public PartialViewResult AjouterProduit(int id)

{

// lire le panier de la variable session

Panier panier = Panier.GetPanier();

// ajouter le produit au panier

int nombreProduits =panier.AjouterProduit(id);

// sauvegarder le panier dans la variable session

Panier.SavePanier(panier);

HtmlString rep = new HtmlString("<h5>Items:" + nombreProduits + "</h5>");

return PartialView(rep);

}

protected override void Dispose(bool disposing)

{

db.Dispose();

base.Dispose(disposing);

}

}

Le contenu de la vue Index

@model IEnumerable<TestMVCAjax.Models.Products>

@{

ViewBag.Title = "Index";

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

}

<h2>Index</h2>

<div id="DivItems"><h5>Items:0</h5></div>

<div>

<table>

<tr>

<th>

@Html.DisplayNameFor(model => model.ProductID)

</th>

<th>

@Html.DisplayNameFor(model => model.ProductName)

</th>

<th>

@Html.DisplayNameFor(model => model.Categories.CategoryName)

</th>

<th>

@Html.DisplayNameFor(model => model.UnitPrice)

</th>

<th></th>

</tr>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.ProductID)

</td>

<td>

@Html.DisplayFor(modelItem => item.ProductName)

</td>

<td>

@Html.DisplayFor(modelItem => item.Categories.CategoryName)

</td>

<td>

@Html.DisplayFor(modelItem => item.UnitPrice)

</td>

<td>

<**a** id="deleteModal" **asp-action**="AjouterProduit" **asp-route-id**="@item.Id" data-ajax="true" data-ajax-method="GET" data-ajax-mode="replace" data-ajax-update="DivItems" class="btn btn-danger">

<i class="glyphicon glyphicon-trash"></i> Acheter

</**a**>

</td>

</tr>

}

</table>

</div>

La vue Partial "\_AjouterProduit"

@model HtmlString

@Model