



# COURS .NET ASP.NET MVC

Lemettre Arnaud

[Arnaud.lemettre@gmail.com](mailto:Arnaud.lemettre@gmail.com)

# SOMMAIRE



- Concept
- Asp.Net MVC
- Intégration JS (MVVM, Framework, ...)
- HTML 5 (Intégration windows 7)

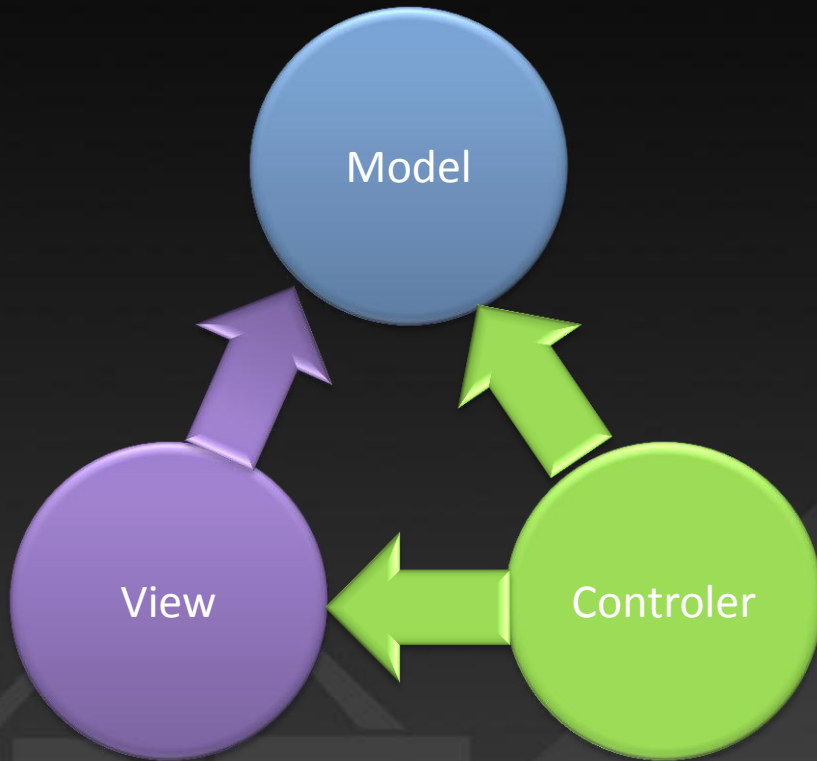


# CONCEPT

# CONCEPT



- MVC pour Model – View – Controller



Model : données + opérations

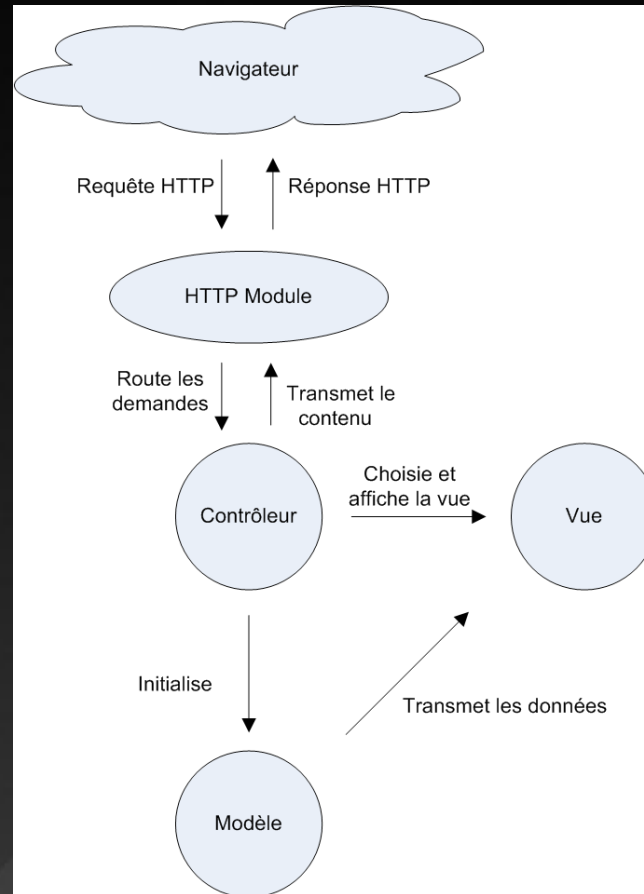
Controller : gère les événements  
et la synchronisation entre vue et modèle

Vue : Présentation des données à  
l'utilisateur

# CONCEPT



- MVC pour Model – View – Controller



*Cycle d'une requête*

# CONCEPT



- Les grandes différences entre Asp.Net et Asp.Net MVC

Asp.net	Asp.net MVC
Contrôleur de page	MVC
View state	binding
PostBack	Contrôle du HTTP
contrôles serveurs	Facilite les tests unitaires

# CONCEPT



- Un peu plus de détails:
  - Le controller, contient nos fonctions (~ actions) et retourne un ActionResult
  - Une Action = point d'entrée dans ASP.net Mvc, soit renvoi une page / des données / ou rien

# CONCEPT



Les avantages :



- Support de l'injection de dépendances
- HTML 5
- Meilleur contrôle du rendu HTML
- Intégration du montage HTML (Designer) plus facile
- Page HTML plus légère





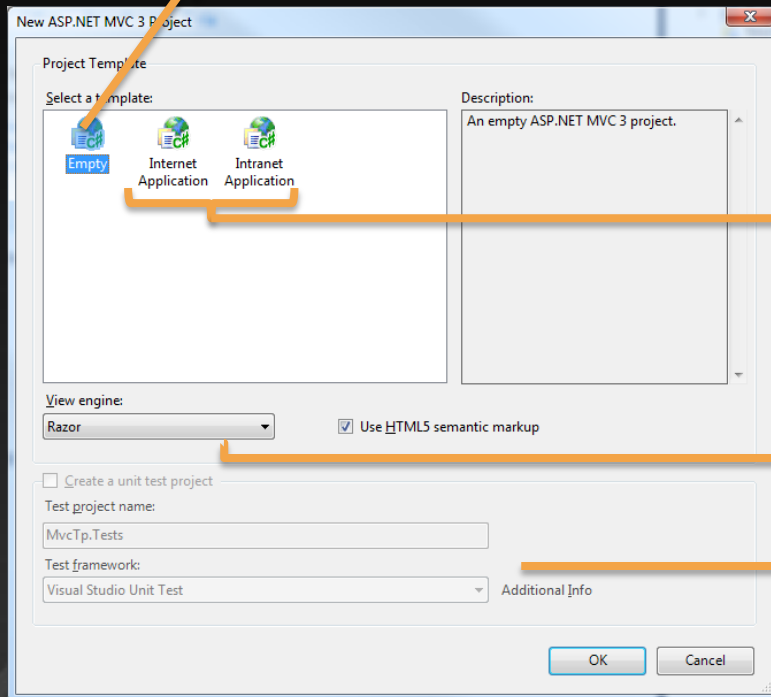
# ASP.NET MVC 3

# ASP.NET MVC 3



- Création d'un projet

Projet vide



Crée un projet avec vues et contrôleurs par défaut, ainsi qu'un système de sécurité.

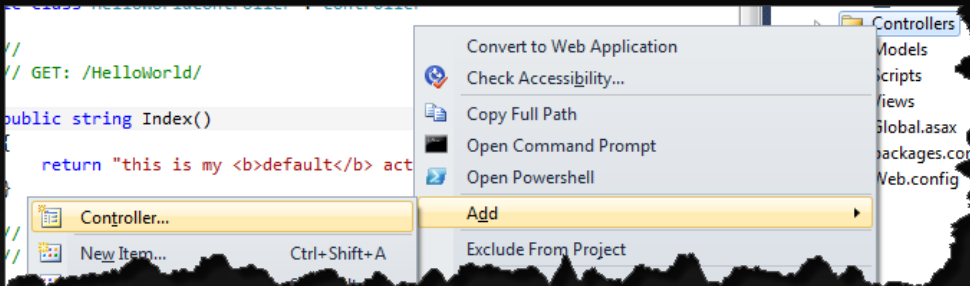
Moteur de vue Razor ou Aspx

Possibilité de créer les tests unitaires associés.

# ASP.NET MVC 3

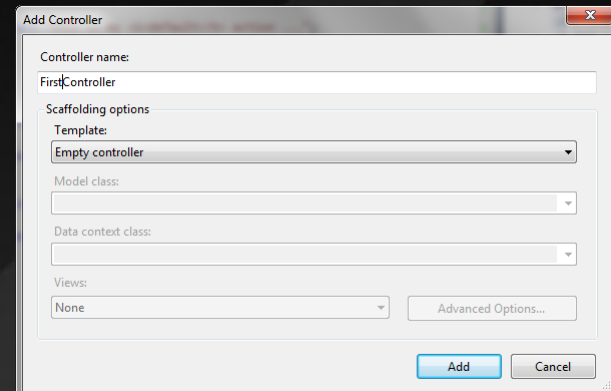


- La couche controller



Dans les templates, vous pouvez les associer :

- Controller vide
- Binder sur un Entity Framework
- Binder sur son propre système de persistance



Pas deux méthodes avec le même nom.

# ASP.NET MVC 3



- La couche controller

Pour comprendre le principe du controller nous sélectionnerons pour commencer un controller vide.

Par convention les controllers se nomment :

*NomDuControleur***Controller**

Le mot controller est obligatoire à la fin pour asp.net MVC

```
namespace MvcTest.Controllers
{
    public class FirstController : Controller
    {
        public string Index()
        {
            return "<b>action</b> par default...";
        }

        public string Welcome()
        {
            return "Welcome methode ...";
        }

        public string WelcomeName(string name)
        {
            return "Welcome ... " + name;
        }
    }
}
```

Index() est la méthode par défaut du Controller

Dans notre cas nous renvoyons uniquement des strings mais nous pourrions renvoyer également des vues.

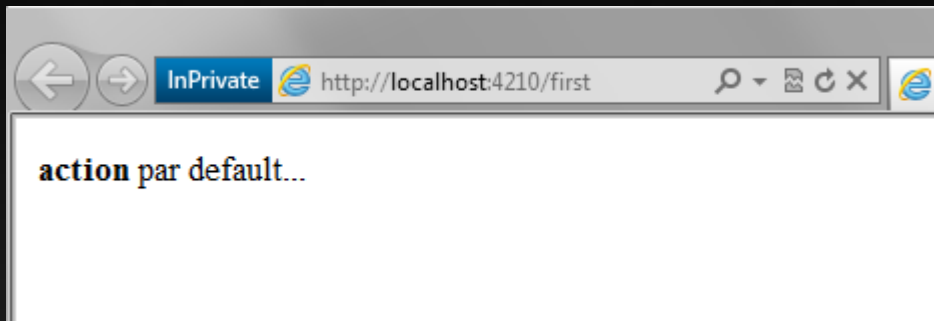
# ASP.NET MVC 3



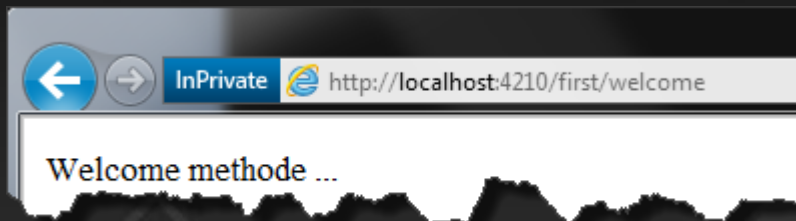
- La couche controller

L'invocation des controllers se fait de cette manière :

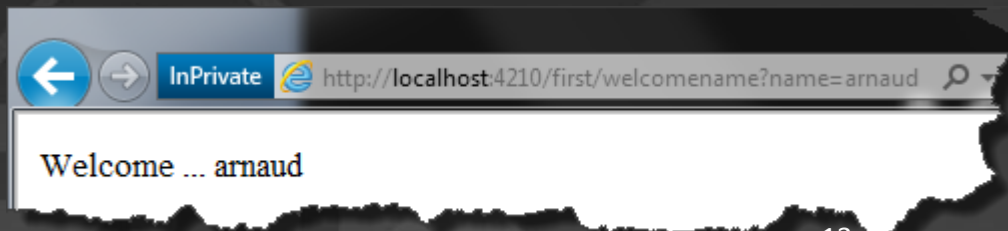
**`/[Controller]/[ActionName][?Parameter=value]`**



Appel sans méthode (méthode index par défaut)



Appel avec méthode



Appel avec méthode + paramètre

# ASP.NET MVC 3



- La couche controller

Retourner des chaînes de caractères c'est bien mais pas très évolué, passons à la création des vues.

Pour ça nous allons créer un nouveau controller.

```
namespace MvcTest.Controllers
{
    public class SecondController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

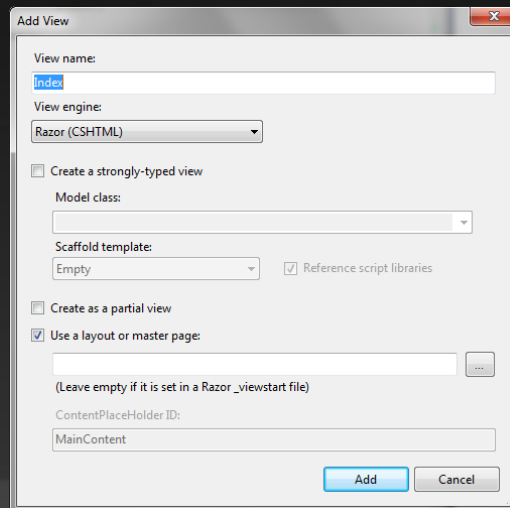
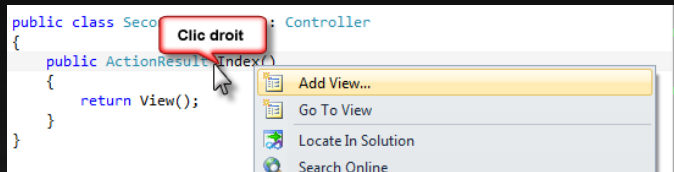
Nous utilisons ici directement ActionResult pour retourner une vue

# ASP.NET MVC 3



- La couche View

Nous pouvons générer notre vue à partir du controller



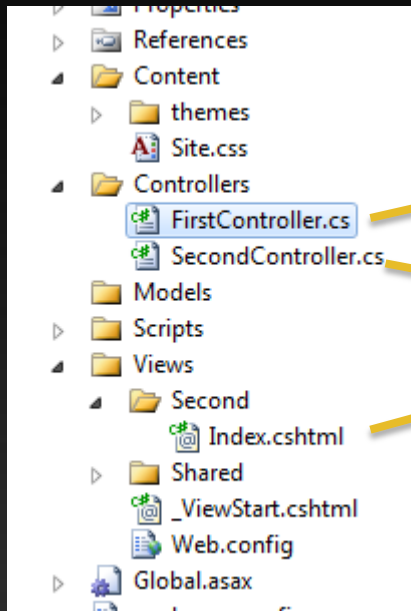
Nous utiliserons ici le moteur de vue Razor, que nous introduirons un peu plus loin

# ASP.NET MVC 3



- La couche View

Nous avons donc l'architecture suivante :



Notre 1<sup>er</sup> controller à base de string

Notre 2<sup>nd</sup> controller basé sur une vue

La vue peut être appelée par `http://server/Second`



# ASP.NET MVC 3



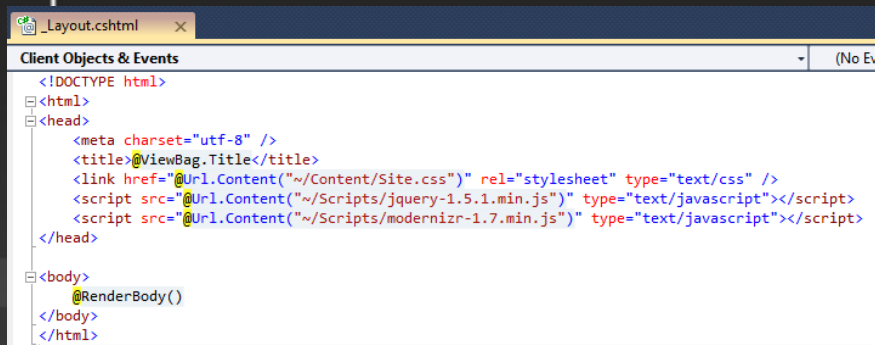
- La couche View

Notre première vue ressemble à ceci :



Notre vue est affichée mais pas toute seule elle dépend d'un conteneur se situant :  
/views/Shared/\_layout.cshtml

Ce fichier nous permettra de faire notre « shell » de notre site web (en incluant tous les éléments communs de celui-ci).



La méthode @RenderBody() permet de définir l'endroit où sera insérer le contenu de notre vue

# ASP.NET MVC 3



- La couche View

Communication entre le controller et la view

Un moyen simple : ViewBag => Objet dynamic

```
public class SecondController : Controller
{
    public ActionResult Index()...

    public ActionResult PushName(string name)
    {
        ViewBag.Name = name;
        // On utilise la vue créée précédemment
        return View("Index");
    }
}
```

```
Client Objects & Events
@{
    ViewBag.Title = "Index";
}
<h2>Index</h2>
Name : @ViewBag.Name
```

Sinon nous aurions dû créer  
une vue ayant pour nom  
PushName

Utilisation du ViewBag avec une propriété  
dynamic

Appel de la page :

<http://server/Second/PushName?name=arnaud>

# ASP.NET MVC 3



- La couche Model

Création d'une classe dans le dossier Model :

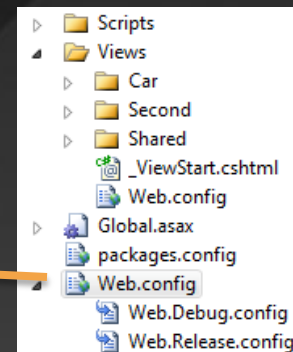
```
namespace MvcTest.Models
{
    public class Car
    {
        public int Id { get; set; }
        public int Speed { get; set; }
    }
}
```

Nous pouvons ici utiliser la méthode du code First afin de générer notre base de données, pour cela il faut créer le contexte :

```
namespace MvcTest.Models
{
    public class CarDbContext : DbContext
    {
        public DbSet<Car> Movies { get; set; }
    }
}
```

Chaine de connexion pour la BDD,  
Ici nous utiliserons une compact Base

```
<connectionStrings>
  <add name="CarDbContext"
        connectionString="Data Source=|DataDirectory|cars.sdf"
        providerName="System.Data.SqlServerCe.4.0"/>
</connectionStrings>
```

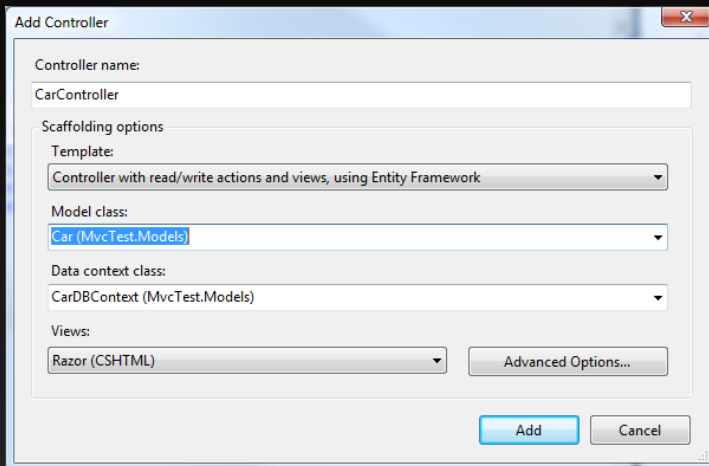


La chaine de connexion se crée à partir du web.config général et pas de celui des vues.

# ASP.NET MVC 3



- Association du Model et du controller



Etant sur un model entity code First, nous sélectionnons directement le template correspondant et on indique le nom de la classe ainsi que du data context à utiliser

Ensemble des méthodes générées, avec les vues associées

```
namespace MvcTest.Controllers
{
    public class CarController : Controller
    {
        private CarDbContext db = new CarDbContext();

        public ActionResult Index()...

        public ActionResult Details(int id)...

        public ActionResult Create()...

        [HttpPost]
        public ActionResult Create(Car car)...

        public ActionResult Edit(int id)...

        [HttpPost]
        public ActionResult Edit(Car car)...

        public ActionResult Delete(int id)...

        [HttpPost, ActionName("Delete")]
        public ActionResult DeleteConfirmed(int id)...

        protected override void Dispose(bool disposing)...
    }
}
```

# ASP.NET MVC 3



- Liaison Controller - View

*/\* Création d'une méthode de recherche \*/*

Dans CarController

```
public ActionResult Search(int searchPattern)
{
    var movies = from m in db.Movies
                 select m;

    if (searchPattern != null)
    {
        movies = movies.Where(s => s.Speed == searchPattern);
    }
    return View(movies);
}
```

Typage fort de la vue

Rajout d'action

Affichage des résultats

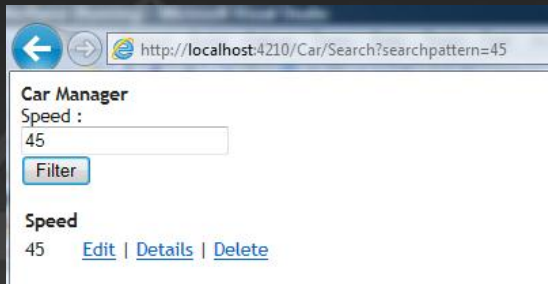
Vue :

```
@model IEnumerable<MvcTest.Models.Car>

<table>
<tr>
<th>
Speed
</th>
</tr>
<tr>
<td>
@using (Html.BeginForm()){
<p> Speed : @Html.TextBox("searchPattern")
<input type="submit" value="Filter" /></p>
}
</td>
</tr>
<tr>
<td>
@foreach (var item in Model) {
<tr>
<td>
@Html.DisplayFor(modelItem => item.Speed)
</td>
<td>
@Html.ActionLink("Edit", "Edit", new { id=item.Id }) |
@Html.ActionLink("Details", "Details", new { id=item.Id }) |
@Html.ActionLink("Delete", "Delete", new { id=item.Id })
</td>
</tr>
}
</td>
</tr>
</table>
```

L'appel à l'action se fera par :

http://server/Car/Search?SearchPattern=50



# ASP.NET MVC 3



- Le routing

L'URL routing est une partie importante de MVC. Ceci nous permet de trouver les ressources que le navigateur demande au travers des requêtes.

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute(
        "Default", // Route name
        "{controller}/{action}/{id}", // URL with parameters
        new { controller = "First", action = "Index", id = UrlParameter.Optional } //
    );
}
```

La configuration des routes se fait dans le fichier global.asax

Par exemple nous pouvons avoir une URL plus compréhensible pour la recherche

Avant modification :

<http://server/car/Search?searchPattern=55>

Après modification :

<http://server/car/Search/45>

```
routes.MapRoute(
    "Car", // Route name
    "Car/Search/{searchPattern}", // URL
    new
    {
        controller = "Car",
        action = "Search",
        searchPattern = "searchPattern"
    } // Parameter defaults
);
```

# ASP.NET MVC 3



- Validation de données

Sur la classe Model

```
using System.Web;
using System.Data.Entity;
using System.ComponentModel.DataAnnotations;

namespace MvcTest.Models
{
    public class Car
    {
        public int Id { get; set; }
        [Range(0, 130, ErrorMessage="Vitesse limite dépassée")]
        public int Speed { get; set; }
    }
}
```

Classe pour les tests :

- Required
- Range
- StringLength

Coté vue :

```
@Html.LabelFor(model => model.Speed)
</div>
<div class="editor-field">
    @Html.EditorFor(model => model.Speed)
    @Html.ValidationMessageFor(model => model.Speed)
</div>
```

Permet d'afficher le message d'erreur

Controller

```
[HttpPost]
public ActionResult Create(Car car)
{
    if (ModelState.IsValid)
    {
        db.Movies.Add(car);
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(car);
}
```

Méthode permettant de vérifier la validité des objets

<http://server/Car/create>

# ASP.NET MVC 3



- Le moteur de vue

Détails de Razor

Les structures de contrôles :

Exemple du FOREACH

```
@foreach (var item in Model)
{
    <b>car</b>
}
```

Exemple du IF

```
@if (Model.Any())
{
    <p>no car</p>
}
else
{
    <p>we have cars</p>
}
```



# ASP.NET MVC 3



- Le moteur de vue

Détails de Razor

Affichage

```
speed : @Html.DisplayFor(modelItem => item.Speed)  
speed : @Html.EditorFor(x => item.Speed)
```



```
speed : 45  
speed : 45
```

*DisplayFor* génère du texte

*EditorFor* génère une textbox

Bien entendu tous les autres contrôles existent également. Sur les contrôles HTML nous pouvons également rajouter tous les attributs

```
speed : @Html.TextBoxFor(x => item.Speed, new { @class="classCss" })
```

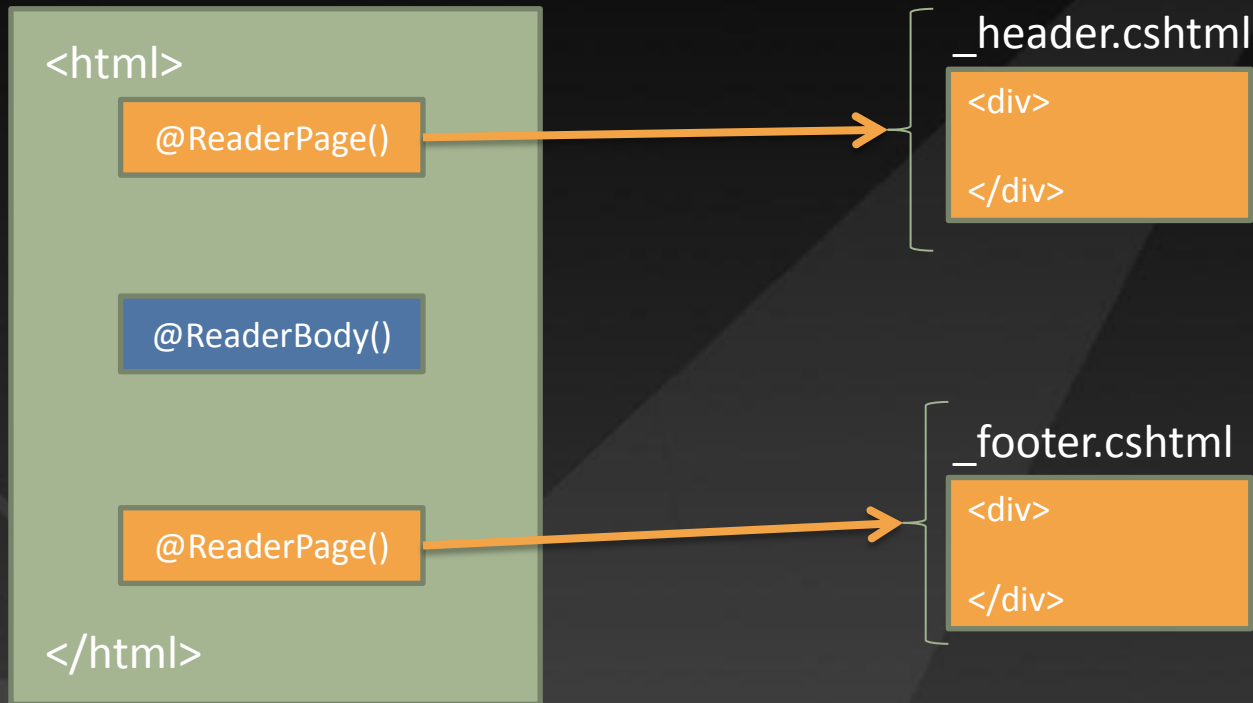
# ASP.NET MVC 3



- Le moteur de vue

Détails de Razor

Construire une page avec Razor, éviter de reproduire du code :



# ASP.NET MVC 3



- Le moteur de vue

En concret :

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>@ViewBag.Title</title>
  <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet" type="text/css" />
  <script src="@Url.Content("~/Scripts/jquery.js")" type="text/javascript" />
  <script src="@Url.Content("~/Scripts/modernizr.js")" type="text/javascript" />
</head>
<body>
  @RenderPage("../_header.cshtml")
  @RenderBody()
  @RenderPage("../_footer.cshtml")
</body>
</html>
```

## Client Objects & Events

```
<div>
  <b>Car Manager Header</b>
</div>
```

```
<div>
  <i>footer - copyright 2011</i>
</div>
```

```
@model IEnumerable<MvcTest.Models.Car>

@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>

<p>
    @Html.ActionLink("Create New", "Create")
</p>

<table>
<tr>
<th>...</th>
<th></th>
</tr>

@foreach (var item in Model) {
    <tr>...</tr>
}

</table>
```

# ASP.NET MVC 3



- Le moteur de vue

Un peu d'HTML 5



Le cshtml pour le header :

```
<div id="header">  
  <b>Car Manager Header</b>  
</div>
```



En HTML5 :

```
<header>  
  <b>Car Manager Header</b>  
</header>
```

Le cshtml pour le footer :

```
<div id="footer">  
  <i>footer - copyright 2011</i>  
</div>
```



En HTML5 :

```
<footer>  
  <i>footer - copyright 2011</i>  
</footer>
```

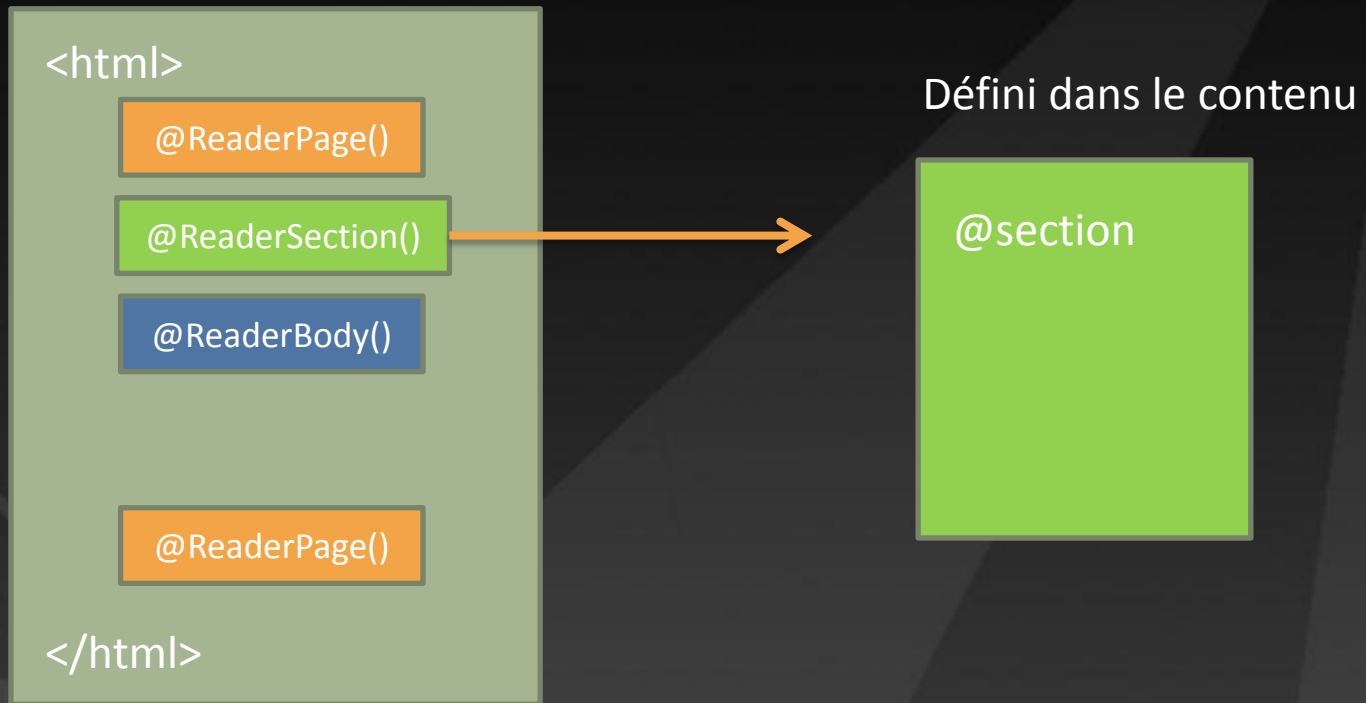
# ASP.NET MVC 3



- Le moteur de vue

Détails de Razor

Rajout de zone : (une zone peut être optionnelle)



# ASP.NET MVC 3



- Le moteur de vue

En concret

\_Layout.cshtml

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>@ViewBag.Title</title>
  <link href="@Url.Content("~/Content/Site.css")" />
  <script src="@Url.Content("~/Scripts/jquery-1.4.1.min.js")" />
  <script src="@Url.Content("~/Scripts/modernizr-1.2.6.min.js")" />
</head>
<body>
  @RenderPage("../_header.cshtml")

  <div style="float: left">
    @RenderSection("List")
  </div>

  @RenderBody()
  @RenderPage("../_footer.cshtml")
</body>
</html>
```

Search.cshtml

```
@model IEnumerable<MvcTest.Models.Car>

@{
    ViewBag.Title = "Search";
}

@section List{
    <ul>
        <li>item 1</li>
        <li>item 2</li>
    </ul>
}

<table>
<tr>
<th>
        Speed
    </th>
<th>
    </th>
</tr>
<tr>
<td>
        @using (Html.BeginForm())
        {
            <p>
                Speed : @Html.TextBox("searchPat")
                <input type="submit" value="Filter" />
            </p>
        }
    </td>
<td>
        @foreach (var item in Model)
        {
            <tr>
                <td>@item.Speed</td>
                <td>@item.Name</td>
            </tr>
        }
    </td>
</tr>
</table>
```

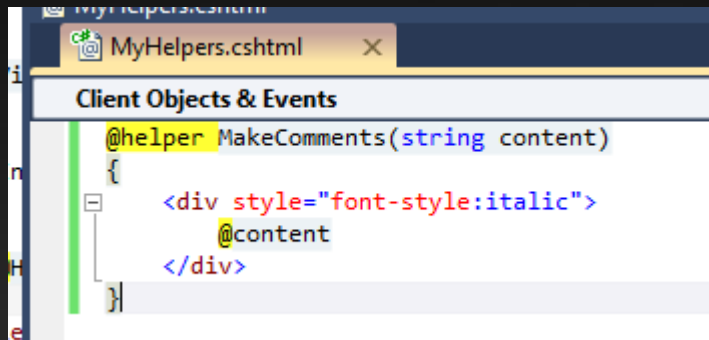
# ASP.NET MVC 3



- Le moteur de vue

Dans le but de réutiliser votre code, vous pouvez également créer vos propres helpers. Pour cela rajouter un dossier à la racine nommé App\_Code.

Puis ajouter un fichier dans celui-ci nommé par exemple : MyHelpers.cshtml



```
@helper MakeComments(string content)
{
    <div style="font-style:italic;">
        @content
    </div>
}
```

N'oubliez pas de compiler pour le voir apparaître !

Utilisation :



```
</td>
<td>
    @Html.ActionLink("Edit", "Edit", new { id=item.Id }) |
    @Html.ActionLink("Details", "Details", new { id=item.Id }) |
    @Html.ActionLink("Delete", "Delete", new { id=item.Id })
</td>
<td>
    @MyHelpers.MakeComments("ma voiture")
</td>
</tr>
```

# ASP.NET MVC 3



- Les ActionFilter vous permette d'agir sur les différentes étapes du cycle de vie d'une page, mais permet également de gérer des autorisations.





# FRAMEWORK JS

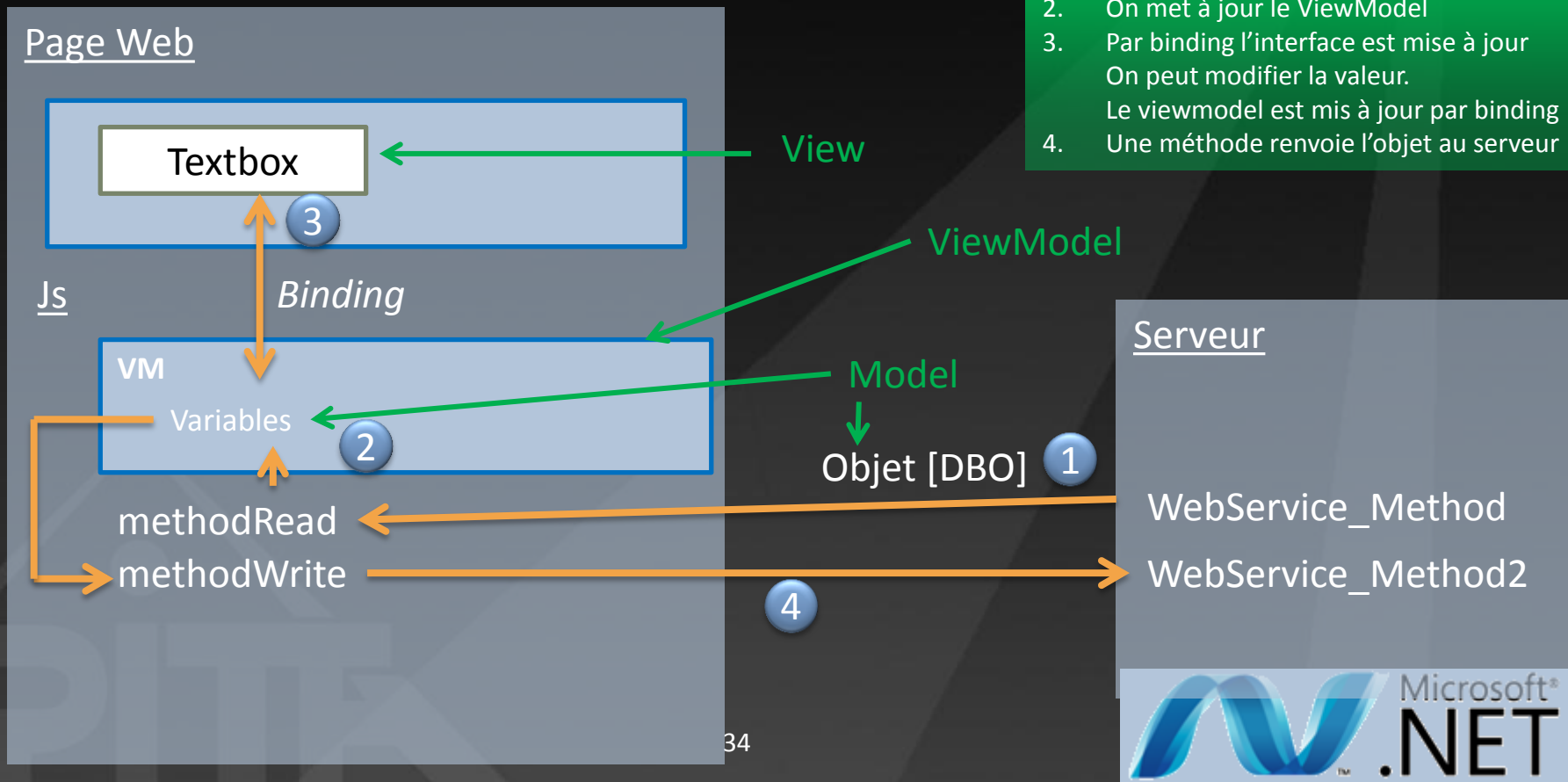


# KNOCKOUT



- Permet d'introduire le design Pattern MVVM = *Model-View-ViewModel*

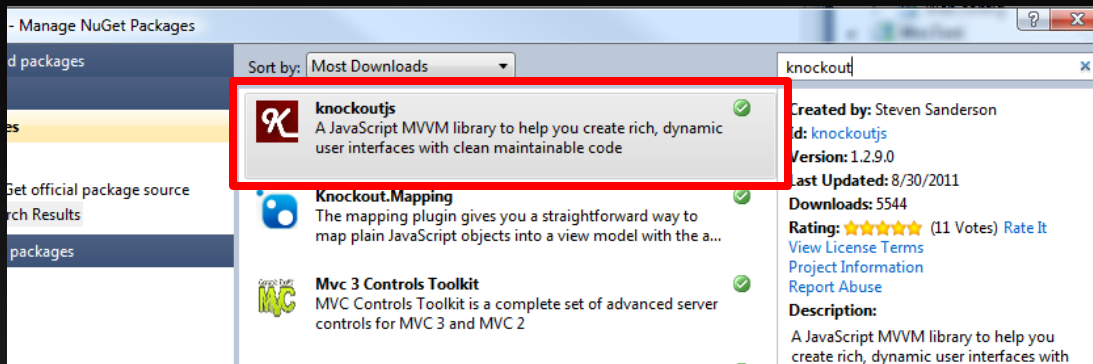
1. On récupère un objet du serveur
2. On met à jour le ViewModel
3. Par binding l'interface est mise à jour  
On peut modifier la valeur.  
Le viewmodel est mis à jour par binding
4. Une méthode renvoie l'objet au serveur



# KNOCKOUT



- Installation => nuget



Ou sur le site officiel : <http://knockoutjs.com/>  
Mise en place de la bibliothèque :

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>@ViewBag.Title</title>
  <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet" type="text/css" />
  <script src="@Url.Content("~/Scripts/jquery-1.5.1.min.js")" type="text/javascript"></script>
  <script src="@Url.Content("~/Scripts/modernizr-1.7.min.js")" type="text/javascript"></script>
  <script src="@Url.Content("~/Scripts/knockout-1.3.0beta.js")" type="text/javascript"></script>
</head>
<body>
  @RenderPage("../_header.cshtml")
  <div style="float: left">
```

# KNOCKOUT



- Pour utiliser knockout avec l'intelligence il faut ré-inclure les scripts dans le cshtml

```
Sample.cshtml x
Client Objects & Events
@{
    ViewBag.Title = "Sample";
}
@if (false)
{
    <script src="/Scripts/knockout-1.3.0beta.debug.js" type="text/javascript"></script>
    <script src="/Scripts/jquery-1.5.1-vsdoc.js" type="text/javascript"></script>
    <script src="/Scripts/jquery-ui-1.8.11.js" type="text/javascript"></script>
    <script src="/Scripts/knockout-1.3.0beta.debug.js" type="text/javascript"></script>
}
<h2>Sample</h2>
the person is
```

Permettra d'avoir l'auto-complétion mais les scripts ne seront pas ré-inclus

- Comprendre le M-V-VM

# KNOCKOUT



- Comprendre le M-V-VM

```
}  
  
<h2>Sample</h2>  
  
the person is <span data-bind="text: personName"></span>  
  
<script type="text/javascript">  
    var viewModel =  
        {  
            personName: 'lemettre',  
            personFirstname: 'arnaud'  
        };  
  
    ko.applyBindings(viewModel);  
</script>
```

Permet d'établir la propriété qui va être bindée

Le script décrivant le ViewModel doit se trouver à la fin de la page pour pouvoir s'appliquer

L'object view Model contient deux propriétés [personName, personFirstname], ces propriétés seront bindées sur les éléments *data-bind*

Pour appliquer le binding, nous devons appliquer le binding grâce à l'objet *ko*.

```
ko.applyBindings(viewModel);
```

# KNOCKOUT



- Un peu de dynamisme

Modification du viewModel pour pouvoir observer les changements

```
<script type="text/javascript">
  var viewModel = {
    personName: ko.observable('lemettre'),
    personFirstname: ko.observable('arnaud')
  };

  ko.applyBindings(viewModel);

  function SetValue() {
    viewModel.personName('bill');
  }
</script>
```

Pour modifier la valeur on rajoute un bouton qui vient modifier la valeur du viewModel par binding le texte change sur l'interface

```
Sample</h2>
the person is <span data-bind="text: personName"></span>
<br />
<button onclick="SetValue()">
  set value
</button>
<br />
```

# KNOCKOUT



- Intégration dans un projet MVC.  
La communication devra se faire au travers du JS nous utiliserons donc du Json pour communiquer entre notre serveur et notre Model JS

Model que nous utiliserons pour communiquer :

```
namespace MvcTest.Models
{
    public class RealTimeInformation
    {
        public int Speed { get; set; }
        public string IdCar { get; set; }
    }
}
```

## Sample

the person is lemettre

set value

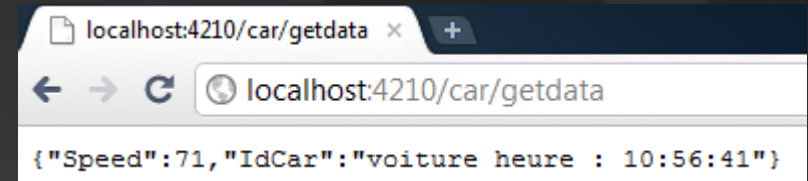
id car is voiture heure : 01:52:08

speed is 185

footer - copyright 2011

Au niveau du contrôleur nous rajouterons une nouvelle action :

```
[OutputCache(Duration = 0)]
public JsonResult GetData()
{
    var data = Models.RealTimeInformationManager.GetData();
    return Json(data, JsonRequestBehavior.AllowGet);
}
```



*Retour de la méthode au format Json*

Coté serveur nous avons terminé, il faut maintenant rajouter la logique de communication avec le client



# KNOCKOUT



- Pour la vue :
  - Nous rajouterons un nouveau view Model sur une div permettant d'afficher les informations

```
<div id="secondViewModel">  
  id car is <span data-bind="text: IdCar"></span>  
  <br />  
  speed is <span data-bind="text: Speed"></span>  
</div>
```

Le second view Model :

```
<script type="text/javascript">  
  
  var viewModelRealTime =  
  {  
    Speed: ko.observable(0),  
    IdCar: ko.observable('')  
  };  
  
  ko.applyBindings(viewModelRealTime, document.getElementById('secondViewModel'));  
  
  $(function () {  
    PollGetData();  
  });  
  
  function PollGetData() {  
    $.get('/Car/GetData', function (data) {  
      viewModelRealTime.Speed(data.Speed);  
      viewModelRealTime.IdCar(data.IdCar);  
      setTimeout("PollGetData()", 1000);  
    });  
  }  
</script>
```

Les variables à exposer

Application du binding sur une div en particulier

Fonction appelée au chargement de la page

On appelle l'action qui retourne du Json que l'on passe dans la fonction pour mettre à jour les données.

Puis on initialise le timer pour répéter l'appel.





# AMPLIFY JS



<http://amplifyjs.com/>

- Fourni 3 API :
  - Système de requêtes
  - De sauvegarde
  - Publication / abonnement

L'avantage de ces composants est de permettre de s'abstraire des implémentations des navigateurs pour supporter ces fonctionnalités.

Par exemple : API de stockage IE-6 ne supporte pas le localStorage mais IE-9 oui. Cependant le code reste le même pour sauvegarder et le composant trouve le meilleur moyen de sauvegarder (cookie, local storage).

# AMPLIFY JS



- Scenarios d'utilisation

Système de requête :

Permet de mettre en cache des appels à des requêtes ce qui permet de ne pas trop utiliser de bande passante

Système de publication :

Permet de déclencher des traitements en fonction d'action dans le code.

# AMPLIFY JS



- Amplify pour sauvegarder

Le but est de pouvoir sauvegarder un formulaire et de le recharger à l'affichage de la page

Rajout de la bibliothèque :

```
AmplifySave.cshtml x
Client Objects & Events
@{
    ViewBag.Title = "AmplifySave";
}
<script src="../../Scripts/amplify.js" type="text/javascript"></script>
```

Nos contrôles :

```
<p>
    <label for="name">Nom</label>
    <input id="name" type="text" />
</p>
<p>
    <a onclick="Save()">Enregistrer les valeurs</a>
</p>
```

# AMPLIFY JS



Rajout du script :

```
<script type="text/javascript">
  function Save() {
    amplify.store("name", $('#name').val());
  }

  $(function () {
    $('#name').val(amplify.store("name"));
  });
</script>
```

*Amplify.store*('clef pour retrouver la valeur',  
'la valeur')

Valeur = *Amplify.store*('clef pour retrouver la valeur')

# MODERNIZR



 **MODERNIZR** <http://www.modernizr.com>

- Est une bibliothèque qui permet d'utiliser les nouvelles fonctionnalités HTML5 et CSS3 sans sacrifier l'expérience utilisateur sur les anciens navigateurs

Son fonctionnement :



*Polyfill : fourni un artefact pour simuler l'API sur les vieux navigateur* [Téléchargement de polyfill](#)

# MODERNIZR



- Si on reprend la page de tout à l'heure nous obtenons :

INSCRIPTION À CAR MANAGER

Nom

Mot de passe

[Afficher + d'options](#)

ou [annuler](#) ou [Enregistrer les valeurs](#)

IE - 9

INSCRIPTION À CAR MANAGER

Nom

Mot de passe

[Afficher + d'options](#)

ou [annuler](#) ou [Enregistrer les valeurs](#)

Chrome

La différence graphique est notable : gradient, ....

# MODERNIZR



- Nous permet de tester si le navigateur contient la fonctionnalité :

```
<script type="text/javascript">
```

```
Modernizr.load({  
  test: Modernizr.cssgradients,  
  nope: '/Scripts/PIE.js'  
});
```

Initialisation du loader

On test les différentes fonctionnalités

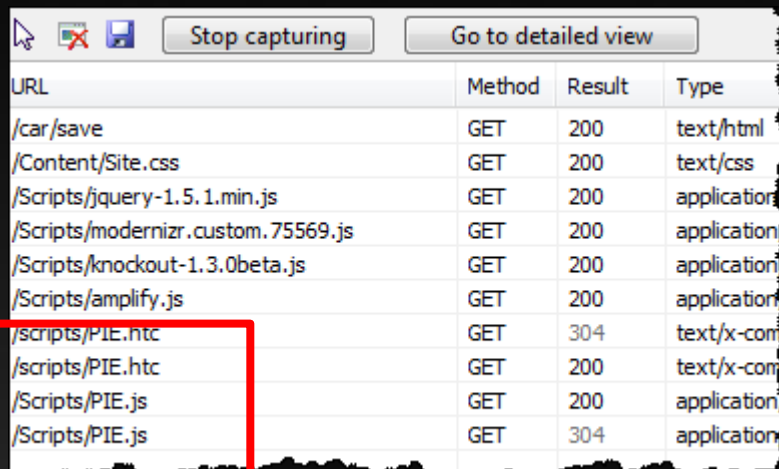
Si elles n'existent pas on charge le polyfill

Ps : Dans notre cas pour rajouter le gradient sur IE nous n'auront pas besoins de rajouter ce JS. Juste de rajouter un fichier dans les scripts et le remplacement d'une partie du CSS

# MODERNIZR

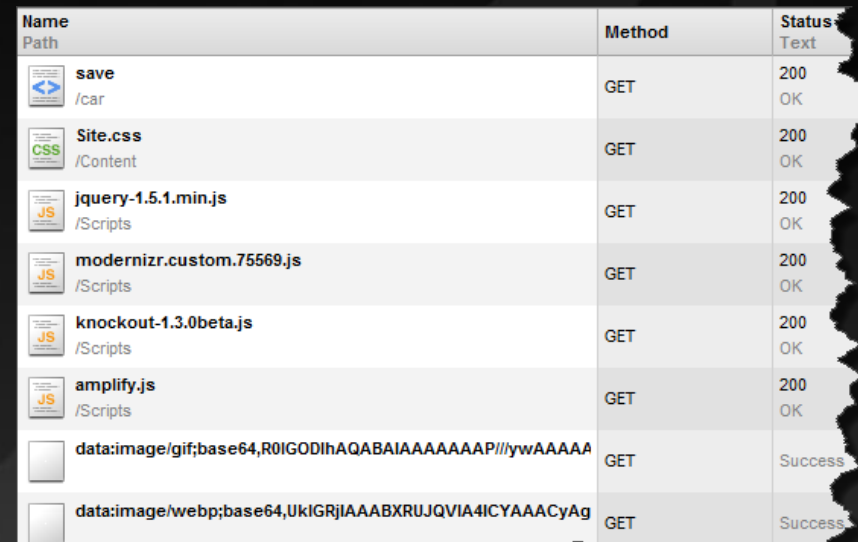


Si on test la fonctionnalité :



URL	Method	Result	Type
/car/save	GET	200	text/html
/Content/Site.css	GET	200	text/css
/Scripts/jquery-1.5.1.min.js	GET	200	application/javascript
/Scripts/modernizr.custom.75569.js	GET	200	application/javascript
/Scripts/knockout-1.3.0beta.js	GET	200	application/javascript
/Scripts/amplify.js	GET	200	application/javascript
/scripts/PIE.htc	GET	304	text/x-com
/scripts/PIE.htc	GET	200	text/x-com
/Scripts/PIE.js	GET	200	application/javascript
/Scripts/PIE.js	GET	304	application/javascript

Sous IE on voit bien le téléchargement des scripts car la fonctionnalité n'est pas présente



Name Path	Method	Status Text
save /car	GET	200 OK
Site.css /Content	GET	200 OK
jquery-1.5.1.min.js /Scripts	GET	200 OK
modernizr.custom.75569.js /Scripts	GET	200 OK
knockout-1.3.0beta.js /Scripts	GET	200 OK
amplify.js /Scripts	GET	200 OK
data:image/gif;base64,R0IGODIhAQABAIAAAAAAAP///ywAAAAA	GET	Success
data:image/webp;base64,UkIGRjIAAABXRJQVIA4ICYAAACyAg	GET	Success

Sous Chrome aucun téléchargement de fichier



# MODERNIZR



```
form{
  background:#f7f7f7;
  background:-moz-linear-gradient(90deg, #ccc, #fff); /* Fi
  background:-webkit-gradient(linear, left top, left bottom
  border:1px solid #aaa;
  margin:60px auto 0;
  padding:20px;
  width:440px;
  border-radius: 8px;
  box-shadow:0 0 15px #aaa;
  -pie-background:linear-gradient(90deg, #ccc, #fff);
  behavior: url(/scripts/PIE.htc);
}
```

Remplacement de la fonctionnalité  
de gradient pour IE

PIE.HTC est le polyfill

IE - 9

Chrome





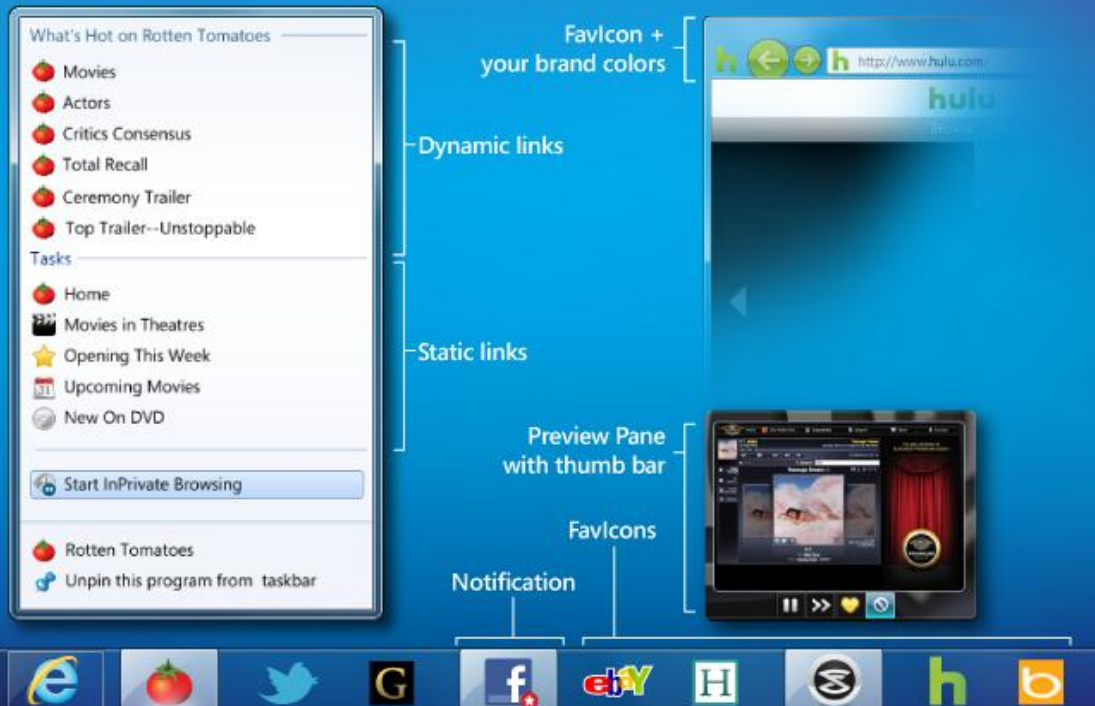
# INTÉGRATION SEVEN

# INTÉGRATION SEVEN



- Permet d'avoir une meilleure expérience utilisateur pour les applications web

<http://buildmypinnedsite.com>



- favicon
- JumpList
- Notification
- Miniature
- Affichage de bannières

# INTÉGRATION SEVEN



- Exemple avec la JumpList

La première des choses rajouter les actions dans les meta :

```
<meta name="msapplication-task" content="name=epita;  
action-uri=http://www.epita.fr;  
icon-uri=/content/favicon.ico" />
```

→ Name permettra de repérer les taches dans le script JQuery

```
<meta name="msapplication-task" content="name=Inscription;  
action-uri=http://localhost:4210/car/save;  
icon-uri=/content/favicon.ico" />
```

```
<script type="text/javascript">
```

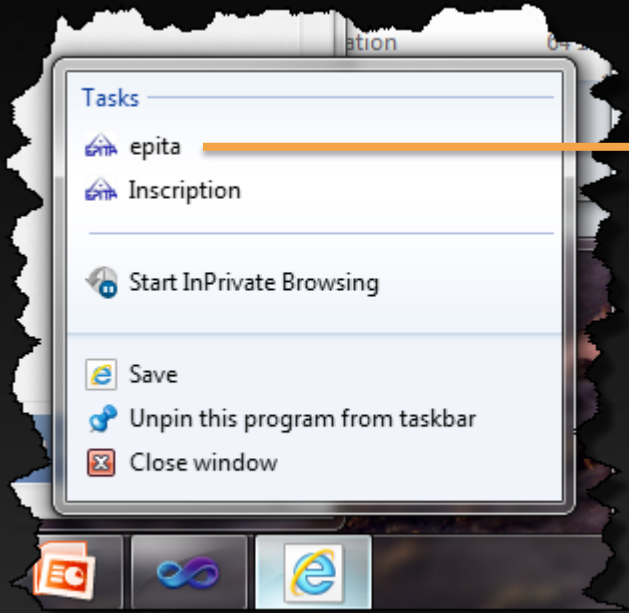
```
$( "head" ).pinify({  
  applicationName: "Build My Pinned Site",  
  favIcon: "/favicon.ico",  
  navColor: "#3480C0",  
  startUrl: "http://buildmypinnedsite.com",  
  tooltip: "Start Build My Pinned Site",  
  window: "width=1024;height=768",  
  tasks: [{  
    'name': "epita",  
    'action': "http://www.epita.fr",  
    'icon': "/favicon.ico"  
  },  
  {  
    'name': "Inscription",  
    'action': "http://localhost:4210/car/save",  
    'icon': "/favicon.ico"  
  }  
]  
});  
</script>
```

→ Script JQuery pour rajouter les taches

# INTÉGRATION SEVEN



- Résultat :



Rajout de liens pour accélérer certaines actions



# QUESTIONS ?