

FUTURE DECODED

6-7 OTT '16 / MILANO

IN PARTNERSHIP WITH:



CommunityDays.it

www.futuredecoded.it



#FutureDecoded

NET05 – ASP.NET MVC Core 1.0

Andrea Saltarello

Solution Architect @ Managed Designs S.r.l. | Presidente UGIdotNET

Microsoft Regional Director

 @andysal74

www.futuredecoded.it

 #FutureDecoded

Me.About();

- Solution Architect @ [Managed Designs](#)
- Microsoft MVP dal 2003
- Microsoft Regional Director
- Co-autore di [.NET: Architecting Applications for the Enterprise](#), by Microsoft Press
- Sostanzialmente, un ~~software architect~~ developer che non vede l'ora di scrivere codice 😊

Talk.About();

- Routing
 - Navigation Flow
- MVC
 - Controller
 - Razor
- Dependency Injection

P.S.: demo scaricabili qui: <http://nsk.codeplex.com>

Anatomia di un URL

Dato uno URL, ad es:

`http://www.sito.net/Customer/Detail/1`

Il *routing engine* di ASP.NET:

- mappa le route verso action/file
- effettua il parsing dello URL e recupera i parametri
- Può essere configurato per supportare route personalizzate

Introduzione a ASP.NET MVC (1/4)

<http://www.sito.net/Customer/Detail/1>

```
namespace ManagedDesigns.MvcShowdown.Controllers
{
    public class CustomerController : Controller
    {
        public ActionResult Index()
        {
            throw new NotImplementedException();
        }

        public ActionResult Detail(int id)
        {
            throw new NotImplementedException();
        }
    }
}
```

Introduzione a ASP.NET MVC (2/4)

http://www.sito.net/**Customer**/Detail/1

```
namespace ManagedDesigns.MvcShowdown.Controllers
{
    public class CustomerController : Controller
    {
        public ActionResult Index()
        {
            throw new NotImplementedException();
        }

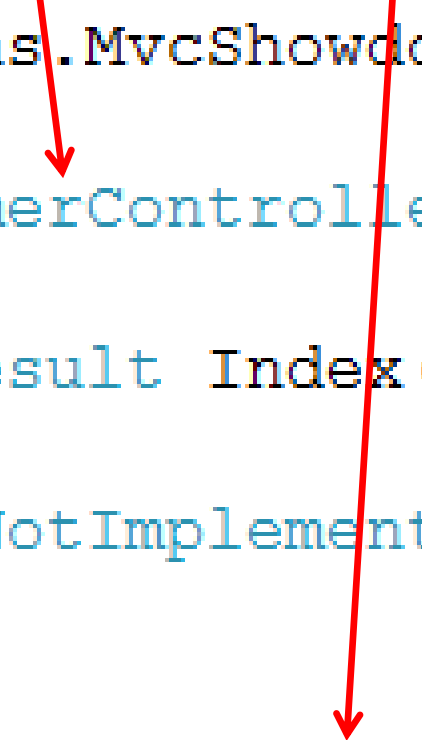
        public ActionResult Detail(int id)
        {
            throw new NotImplementedException();
        }
    }
}
```

Introduzione a ASP.NET MVC (3/4)

http://www.sito.net/**Customer/Detail/1**

```
namespace ManagedDesigns.MvcShowdown.Controllers
{
    public class CustomerController : Controller
    {
        public ActionResult Index()
        {
            throw new NotImplementedException();
        }

        public ActionResult Detail(int id)
        {
            throw new NotImplementedException();
        }
    }
}
```

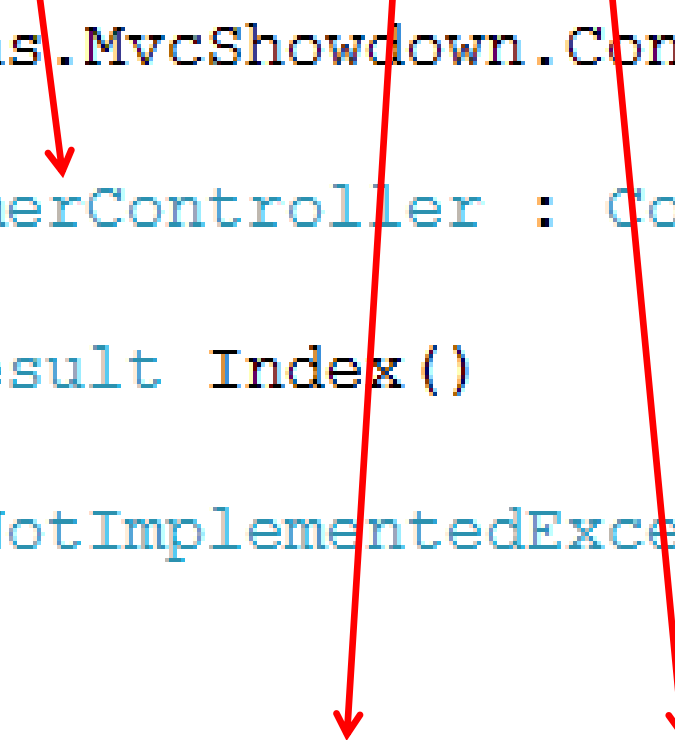
Two red arrows originate from the URL 'http://www.sito.net/Customer/Detail/1'. The first arrow points from the 'Customer' segment to the 'CustomerController' class definition in the code. The second arrow points from the 'Detail' segment to the 'Detail(int id)' method definition in the code.

Introduzione a ASP.NET MVC (4/4)

http://www.sito.net/**Customer/Detail/1**

```
namespace ManagedDesigns.MvcShowdown.Controllers
{
    public class CustomerController : Controller
    {
        public ActionResult Index()
        {
            throw new NotImplementedException();
        }

        public ActionResult Detail(int id)
        {
            throw new NotImplementedException();
        }
    }
}
```



Controller

Una singola, ed opzionale, classe base: **Microsoft.AspNet.Mvc.Controller**

Sono *Action* tutti i metodi pubblici di un controller.

Se restituiscono:

- **IActionResult**, il “giro” è quello di **MVC**
- **!= IActionResult**, il “giro” è quello di **WebAPI** (quindi *content negotiation* o **ProducesAttribute**)

Se costruiamo degli helper, decoriamoli con **NonActionAttribute**

Navigation Flow

Le action restituiscono un valore di tipo IActionResult. “Pragmapolimorficamente” parlando:

- **ViewResult**. Restituito dal metodo **View**
- **PartialViewResult**. Restituito dal metodo **PartialView**
- **ContentResult**. Restituito dal metodo **Content**
- **JsonResult**. Restituito dal metodo **Json**
- **EmptyResult**. Restituito dalle action che vogliano restituire “null”
- **RedirectToRouteResult**. Restituito dai metodi **RedirectToAction** e **RedirectToRoute**
- **RedirectResult**. Restituito dal metodo **Redirect**
- *YourOwnPersonalResult* (semi-cit ☺); esempi [qui](#) e [qui](#)

demo

MVC

WebAPI

JSON Serialization

Razor view engine

Razor è il motore di rendering delle *view*, che:

- Sono file con estensione .cshtml
- Contengono un mix di markup e codice C#
- Possono essere tipizzate

Una view può essere visualizzata:

- indipendentemente
- all'interno di un layout
- all'interno di un'altra view (Partial View, Editor/Display template)

All'interno di una view abbiamo accesso agli HTML helper method

demo

Layout
Tooling

POCO Controller

Nessuna classe base; può essere conveniente dichiarare ed *attivare*:

- `IUrlHelperFactory`
- `IActionContextAccessor`
- `public ActionContext ActionContext { get; set; } *`
- `public ViewDataDictionary ViewData { get; set; } *`

* Solo con container di terze parti:

- <https://github.com/aspnet/Mvc/issues/2151#issuecomment-104541719>
- <https://github.com/aspnet/Announcements/issues/28>

demo

POCO Controller

Tag Helper

Sono la forma “accatiemmellizzata” degli Helper method:

- Package **Microsoft.AspNet.Mvc.TagHelpers** (o cmq contenente classi derivate da **TagHelper**)
- **@addTagHelper** AssemblyName (es: "Microsoft.AspNet.Mvc.TagHelpers")
- **@removeTagHelper**
- **@tagHelperPrefix**
- !

Possiamo costruire Tag Helper custom ereditando la classe **TagHelper**

demo

_ViewImports.cshtml
Custom TagHelper
Environment

Display/Editor templates

I Display/Editor template:

- Permettono di “insegnare” a Razor come renderizzare un Data Type
- Sono partial view il cui nome file coincide con quello del data type (es: DateTime.cshtml)

View Component

I View Component:

- Sostituiscono le *child action* di MVC 5 e sono simili alle *partial view*, ma composti da:
 - Una (o più) view
 - Un “light” controller
- Si usano come gli helper method e nella roadmap di MVC Core è previsto il supporto in futuro ad una sintassi *taghelper like*
- Possono essere distribuiti in una libreria esterna al progetto:
<https://channel9.msdn.com/Series/aspnetmonsters/ASPNET-Monsters-Episode-52-Loading-View-Components-from-a-Class-Library>

demo

View component

Dependency Injection

Supportata in ~~Controller, Filtri, View~~ sostanzialmente *dappertutto*

1. Registrare i tipi in **ConfigureServices**, indicando lo scope:

- **AddInstance**
- **AddSingleton**
- **AddScoped**
- **AddTransient**

2. Esporre le dipendenze

- Ctor (Controller, ViewComponent)
- Parametri action: **FromServicesAttribute**
- View: **@inject**

E' possibile sostituire il container built in con uno di terze parti:

<https://github.com/aspnet/DependencyInjection/blob/dev/README.md>

demo

Controller (Logger)
View (EditorTemplate)
Custom ActionResult



GRAZIE!

www.futuredecoded.it



#FutureDecoded