AJAX

- AJAX sta per Asynchronous JavaScript and XML (ovvero JavaScript asincrono e XML).
- Si tratta di uno script lato client che comunica da e verso un server/database senza la necessità di un aggiornamento completo della pagina.
- L'Ajax accelera i tempi di risposta.

AJAX

- In quali casi si utilizza?
- Come metodo per scambiare dati con un server e aggiornare parti di una pagina web, senza ricaricare l'intera pagina.

- Dove avviene la chiamata alla funzione?
  - O all’interno del file .cshtml
  - O nello specifico file di js/site.js

Gli URL con MVC

Classe ProductsController

I metodi diventano Action

- Action Categories()
  - http://www.miosito/Products/Categories
- Action List(string category)
  - http://www.miosito/Products/List/?category=ASP.NET
- Action Details(int id)
  - http://www.miosito.it/Products/Details/15

Da ASP.NET MVC…

La magia avviene nella classe *RouteConfig*, richiamata allo startup dal *GlobaAsax.cs*

Viene sfruttato il *ModelBinding* per associare i parametri della route

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

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

…ad ASP.NET Core

La magia avviene nella classe *Startup* all'interno del metodo *Configure*

Il metodo *UseMvc* crea una nuova istanza di *RouteMiddleware* e la aggiunge alla pipeline di esecuzione

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)
{
    app.UseMvc(routes =>
    {
        routes.MapRoute(name: "default",
                        template: "{controller}/{action}/{id?}",
                        defaults: new { controller = "Home", action = "Index" });
    });

    // equivalente
    app.UseMvcWithDefaultRoute();
}
```

Personalizzazione

È possibile personalizzare il comportamento di routing semplicemente aggiungendo constraint sulle proprietà o eventuali token per gestire, ad esempio, la localizzazione

```
routes.MapRoute(
    name: "us_english_products",
    template: "en-US/Products/{id}",
    defaults: new { area = "Backoffice", controller = "Products",
                    action = "Details" },
    constraints: new { id = new IntRouteConstraint() },
    dataTokens: new { locale = "en-US" });
```

Constraint per route

I parametri inclusi tra parentesi graffe devono fare match altrimenti la route viene ignorata

Il carattere «?» rappresenta un valore opzionale

Il carattere «.» per le estensioni dei file è comunque un valore opzionale

Le constraint sono classi che implementano *IRouteConstraint* e possono forzare il tipo di un parametro

```
{id:int}
{active:bool}
{name:lenght(5,20)}
```

```
namespace Microsoft.AspNetCore.Routing.Constraints
{
    public class IntRouteConstraint : IRouteConstraint
    {
        public IntRouteConstraint();
    }
}
```

icubed

Attribute routing

Il routing può essere definito anche come attributo nella decorazione del metodo corrispondente

In questo caso non viene calcolato il nome del controller perché verrà mappato sempre su Home

```
[Route("")]
[Route("Home")]
[Route("Home/Index")]
public IActionResult MyIndex()
{
    return View("Index");
}
```

icubed

Gestione dei conflitti

Può succedere che ci siano due o più metodi corrispondenti alla stessa route: in questo caso viene generato un conflitto

In caso di conflitti, se MVC non è in grado di trovare una «best» route, verrà generata una *AmbiguousActionException*

Per evitarli, è bene:

- Specificare dettagliatamente il routing tramite attribute routing
- Impostare HTTP Verb specifici come decorazione dei metodi
- Considerare una route di fallback e di default per gestire un eventuale redirect

Ottenere l'URL

E' possibile recuperare l'url da una action attraverso la classe *Url* che analizzerà la route corrispondente

Viene utilizzato anche con i tag helper per la gestione delle form

```
public IActionResult Source()
{
    // Genera /custom/url/to/destination
    var url = Url.Action(nameof(Destination));
    return Content($"Url generato: {url}");
}

[HttpGet("custom/url/to/destination")]
public IActionResult Destination()
{
    return View();
}
```

icubed

Configurazione avanzata

Basta agire sul Startup.cs

```
services.AddRouting(x =>
            {
                x.AppendTrailingSlash = true;
                x.LowercaseUrls = true;
            });
```

icubed

Demo

Esercitazione n. 2

- Completare le operazioni CRUD sull'entità ***Prodotto*** utilizzando:
  - Un'azione nel controller, accessibile direttamente dalla tabella dei prodotti;
  - Un'azione che viene lanciata tramite comando Ajax nella pagina di dettaglio.

Aggiungere un Tag Helper

Si possono aggiungere i tag helper sfruttando la direttiva *@addTagHelper*, di solito all'interno del file *_ViewImports*

Allo stesso modo è possibile rimuovere uno specifico tag helper all'interno di una pagina specificando la direttiva *@removeTagHandler*

```
@addTagHelper "*, TagHelperNamespace"
@removeTagHelper "MyTagHelper, TagHelperNamespace"
```

Disabilitare un tag helper

E possibile specificare l'utilizzo dei tag helper su tutti i controlli tranne che su un attributo specifico

E' sufficiente aggiungere all'apertura del tag HTML il carattere «!»

```
<!span asp-validation-for="Email"
      class="text-danger"></!span>
```

Disabilitare un tag helper

AGGIUNGERE SLIDE SULLA CLASSE DI TAG HELPER

```
<!span asp-validation-for="Email"
      class="text-danger"></!span>
```

Tag Helper custom

Un TagHelper è una classe che implementa dall'interfaccia *ITagHelper*, ma per comodità si eredita direttamente da TagHelper per avere accesso al metodo Process

Per convenzione è meglio utilizzare il nome del tag seguito da TagHelper, ma non è obbligatorio.

```
public class EmailTagHelper : TagHelper
{
    public override void Process(TagHelperContext context, TagHelperOutput output)
    {
        output.TagName = "a"; }
}
```

icubed

Tag Helper custom

Context contiene le informazioni associate all'esecuzione del tag corrispondente

Output contiene invece le informazioni relative all'elemento HTML corrispondente al tag sorgente

```
public class EmailTagHelper : TagHelper
{
    public override void Process(TagHelperContext context, TagHelperOutput output)
    {
        output.TagName = "a";
    }
}
```

Tag Helper custom

Per rendere il tag helper custom raggiungibile ed utilizzabile, è necessario registrarlo con la direttiva *@addTagHelper* (come wildcard, oppure tramite nome esplicito)

```
@addTagHelper *, Assembly
```

Tag Helper custom

E’ possibile aggiungere attributi personalizzati all’elemento HTML renderizzato

```
public class EmailTagHelper : TagHelper
{
    public string MailTo { get; set; }

    public override void Process(TagHelperContext context, TagHelperOutput output)
    {
        output.TagName = "a";    // Replaces <email> with <a> tag

        var address = MailTo + "@" + EmailDomain;
        output.Attributes.SetAttribute("href", "mailto:" + address);
        output.Content.SetContent(address);
    }
}
```

Tag Helper custom

Se le proprietà vengono create con sintassi Pascal-case, allora gli attributi vengono analizzati con sintassi lower-kebab case

```
<email mail-to="" />
```

E’ anche possibile accettare proprietà complesse, in questo caso si deve passare l’intero oggetto così come verrà interpretato da C#

Tag Helper custom

E’ possibile decorare i tag helper custom con degli attributi per ridefinire il comportamento

```
[HtmlTargetElement(TagStructure = TagStructure.WithoutEndTag)]
```

L’ordine di valutazione di questi attributi può alterare il risultato.

Due proprietà separate da virgola all’interno dello stesso attributo vengono valutate come AND, se fanno parte di attributi differenti sono valutate come OR.

Demo

Esercitazione n. 3

- Creare un Tag Helper custom con le seguenti caratteristiche:
  - <e-mail to-user=“m.bluth@example.com”/>
- che sia la sostituzione del tag ancora:
  - Email

icubed

Filters

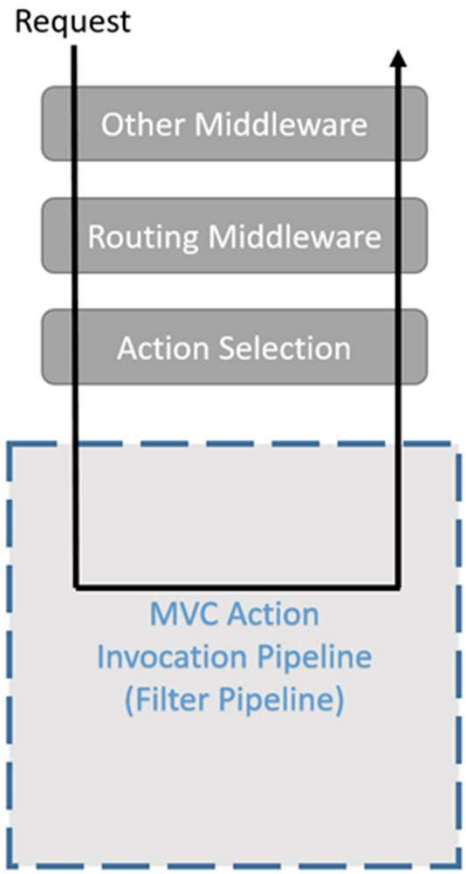
Consentono di aggiungere comportamenti a controller ed action

Ad esempio:

- [Authorize]
- [HttpPost]

Globali (*Startup.cs*)

Locali (su controller o action)



L'infrastruttura dei filtri

Simili agli *HttpModule* di ASP.NET

Sono pensati per ASP.NET Core MVC

Dal punto di vista strettamente pratico, si tratta di classi che implementano una serie di interfacce, ognuna specifica per una particolare funzionalità.

Tipologie di filtri

Action filter: implementano l'interfaccia IActionFilter e ci consentono di gestire le fasi immediatamente precedenti e successive all'esecuzione della action

Result filter: simili ai precedenti, implementano l'interfaccia IResponseFilter e ci danno la possibilità di iniettare codice nelle fasi immediatamente precedenti e successive all'esecuzione del result di una action

Exception filter: implementano IExceptionFilter e contengono codice che gestisce situazioni di errore, permettendo di intercettare un'eccezione non gestita sul server

Authorization filter: implementano IAuthorizationFilter e ci permettono di gestire le policy secondo cui consentire l'esecuzione di una determinata action

Ordine d’esecuzione dei filtri

Controller: *OnActionExecuting*
Global: *OnActionExecuting*
Class: *OnActionExecuting*
Method: *OnActionExecuting*
Method: *OnActionExecuted*
Class: *OnActionExecuted*
Global: *OnActionExecuted*
Controller: *OnActionExecuted*

Costruire un custom filter

Classe che eredita da *ActionFilterAttribute*

OnAuthorization
- prima dell'esecuzione della action per verificare l'accesso

OnActionExecuting
- subito prima dell'esecuzione del codice della action

OnActionExecuted
- appena dopo l'esecuzione della action

OnResultExecuting
- subito prima dell'esecuzione del risultato ritornato dalla action

OnResultExecuted
- appena dopo il termine dell'elaborazione del risultato

OnException
- nel caso in cui il flusso di gestione della richiesta abbia sollevato un'eccezione

icubed

Registazione globale dei filtri

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        ...
        services.AddMvc(option =>
        {
                // istanza condivisa per ogni richiesta
                option.Filters.Add(new MyFilterAttribute());
                // gestione fatta da DI
                option.Filters.Add(typeof(MyFilterAttribute));
        });
    }
}
```

Registazione locale dei filtri

```
// verrà utilizzata la DI
// dobbiamo ricordarci di registrare MyFilterAttribute
[ServiceFilter(typeof(MyFilterAttribute))]
public IActionResult Index()
{
    return View();
}

// non necessita di registrazione via DI
[TypeFilter(typeof(MyFilterAttribute),
     Arguments = new object[] { "Param1", "Value" })]
public IActionResult Index()
{
    return View();
}
```

icubed

Utilizzo dei filtri su Controller/Action

I filtri possono essere applicati più volte a:

- Controller
- Action

```
[HandleError(ExceptionType = typeof(DbException), View = "DatabaseError")]
[HandleError(ExceptionType = typeof(AppException), View = "ApplicationError")]
public class ProductController
{
}
```

Filtri custom

```
public class MyActionFilter : ActionFilterAttribute
{
}
```

I filtri estendono una delle classi base a seconda del tipo di filtro da implementare
È possibile controllare l'ordine d'esecuzione dei filtri appartenente al solito tipo impostando la proprietà *Order*

icubed

Esempi di filtri

Action Filter

- Per gestire il ciclo di vita della action/view

Resource Filter

- Per gestire un output sotto forma di risorsa

Exception Filter

- Per gestire un errore custom (es: risposta in JSON per WebAPI, anziché pagina di errori in HTML)

Result Filter

- Per gestire direttamente il risultato

Filtri o Middleware

Sono simili in scope

Filtri = MVC

MiddleWare = ASP.NET Core

I filtri hanno il vantaggio di stare più vicino ad MVC, quindi accedo alle sue caratteristiche

- Es: model validation
- Es: ViewData/ViewBag

Demo

Identità e sicurezza

ASP.NET Identity

CookieAuthentication

Data Protection

icubed

ASP.NET Identity

Nuovo sistema di gestione dell'identity basato sul concetto di claim

- Un claim è una dichiarazione: es "username" = "Daniele"

Basato su provider

- modello estendibile facilmente con provider, Entity Framework già incluso

Supporta provider multipli per il login

- Cookie, social, Active Directory, Office 365, Federation, etc

ASP.NET Identity

Ha già tutta la logica necessaria a gestire two-factor authentication, reset dei PIN via SMS/mail

- Il template di default di VS si occupa di creare un po' tutto

Si integra alla perfezione con Controller/View

Cookie Middleware

Se non vogliamo gestire con gli automatismi di ASP.NET Identity, possiamo gestire con Cookie Middleware

Saremo responsabili di gestire gli utenti e sfrutteremo un cookie per assicurarci che il ticket di authentication sia giusto

In *startup.cs*:

```
Services
.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
.AddCookie(options =>
{
   options.LoginPath = new PathString("/Account/Unauthorized/");
   options.AccessDeniedPath = new PathString("/Account/Forbidden/");
   options.Cookie.Name ="MyApp"
});
```

Gestire il cookie di autenticazione

È sufficiente, dopo aver controllato il database, fare una chiamata come la seguente

```
await HttpContext.Authentication.SignInAsync("MyCookieMiddlewareInstance",
    principal,
    new AuthenticationProperties
    {
      ExpiresUtc = DateTime.UtcNow.AddMinutes(20),
      IsPersistent = true
    });
```

Per fare logout

```
await HttpContext.Authentication.SignOutAsync("MyCookieMiddlewareInstance");
```

icubed

Autenticazione

Con ASP.NET Core 1.x l'autenticazione è configurata con Middleware

Con ASP.NET Core 2 l'autenticazione è configurata con i Services

- Supporto a Cookie, OpenID Connect, JWT Bearer, Facebook, Twitter, Google, Microsoft Account

Nuove estensioni ad *HttpContext* anziché utilizzare *IAuthenticationManager*

https://docs.microsoft.com/en-us/aspnet/core/migration/1x-to-2x/identity-2x

Demo