# Utilizzare il framework FluentValidation con ASP.NET Core

Prima di essere utilizzati dall'applicazione, i dati devono essere **convalidati e controllati** per escludere potenziali minacce alla sicurezza. È anche necessario verificare che il tipo e le dimensioni dei dati siano conformi alle regole.  
  
Nonostante l'implementazione della convalida sia un'attività ridondante e noiosa, si tratta comunque di un processo necessario. In ASP.NET MVC essa viene eseguita tipicamente sia nel client sia nel server.  
  
La **validazione lato client** consente all'utente di ridurre i tempi di risposta e di ottimizzare le performance del server.  
  
La **validazione lato server** invece è necessaria per tutte le applicazioni che richiedo i dati a un servizio Web, sia esso WebApi che MVC.  
  
In ASP.NET e in ASP.NET Core è possibile validare i propri dati con delle semplici **DataAnnotation** da posizionare sulle proprietà delle classi del modello; è anche possibile aggiungere logica personalizzate andando ad estendere e arricchire le regole base che il sistema fornisce.  
  
Nello script di oggi presentiamo un nuovo componente che può intervenire in aggiunta o al posto della validazione ASP.NET o ASP.NET Core che siamo abituati ad utilizzare.  
  
Il componente si chiama **FluentValidation** e lo si può integrare in qualsiasi tipo di progetto, sia esso web che desktop. Il framework fornisce delle regole di validazione standard alle quali si possono aggiungere regole personalizzate da applicare a seconda delle esigenze.  
  
Oltre al concetto di regola personalizzata è possibile creare dei **RuleSet** che aggregano più regole semplici e applicare questo set al momento del bisogno.

#### Installare il componente in ASP.NET Core

Come già anticipato il componente può essere utilizzato **in sostituzione** dell'engine ASP.NET oppure in **associazione** con esso; in questo secondo caso verrà prima eseguita la validazione di FluentValidation e in seguito quella di default di ASP.NET.  
  
L'installazione del componente è molto semplice. Dopo aver aggiunto un riferimento al package NuGet

Install-Package FluentValidation.AspNetCore

non dobbiamo far altro che inserire poche righe nella classe **Startup**:

public void ConfigureServices(IServiceCollection services)

{

services.AddMvc()

.SetCompatibilityVersion(CompatibilityVersion.Version\_2\_1)

.AddFluentValidation(fv =>

{

fv.RegisterValidatorsFromAssemblyContaining<PersonValidator>();

fv.RunDefaultMvcValidationAfterFluentValidationExecutes = false;

});

*// .. altro codice qui ..*

}

Il metodo **AddFluentValidation** ci consente di specificare le varie impostazioni per la configuzione del framework. Nel codice in alto, **RegisterValidatorsFromAssemblyContaining** registra tutte le classi di validazione presenti nell'assembly in cui è contenuta la classe **PersonValidator**.

#### Scrivere un validatore

La definizione di un validatore avviene all'interno di una classe che necessariamente deve implementare la classe **AbstractValidator<T>** dove T è la classe di **ViewModel** che vogliamo validare.  
  
La classe **AbstractValidator** ci fornisce una serie di metodi che possiamo utilizzare per validare le singole proprietà.  
  
Le regole di validazione possono essere di due tipi: **RuleFor** e **RuleSet**. Mentre la prima viene utilizzata per validare una singola proprietà, la seconda ci consente di validare un gruppo di proprietà. E' possibile richiamare l'insieme di regole in vari punti dell'applicazione. In questo script ci focalizzeremo sui primi.  
  
L'espressione **RuleFor** richiede la proprietà da validare e successivamente sarà necessario definire i requisiti da associare ad essa.  
  
Nell'esempio seguente:

* la proprietà **Birthday** verifica che la data immessa sia compresa tra la data di oggi e gli ultimi 150 anni;
* la proprietà **Name** invece è richiesta non vuota;
* la proprietà **Surname** viene verificata attraverso un servizio iniettato da ASP.NET Core e che verifica, tramite **accesso al database**, la presenza del cognome a sistema.

public class PersonValidator : AbstractValidator<PersonViewModel>

{

public const string ValidaMail = "ValidaMail";

public const string ValidaNumeriTelefono = "ValidaNumeriTelefono";

private readonly IMyService \_myService;

public PersonValidator(IMyService myService)

{

\_myService = myService;

*// Common rules*

RuleFor(x => x.Birthday)

.ExclusiveBetween(DateTime.Now.AddYears(-150), DateTime.Now);

RuleFor(x => x.Name).NotEmpty();

RuleFor(x => x.Surname).NotEmpty().Custom((x,y) => {

if (\_myService.UserExists(x))

{

y.AddFailure(new ValidationFailure("Surname", "User does not exist"));

}

});

*// .. altro codice qui ..*

}

#### Applicazione e verifica della validazione

FluentValidator agisce come il componente di validazione ASP .NET e l'esito della validazione viene inserito nel **ModelState**, con tutte le informazioni riguardanti ai motivi di scarto del modello.  
  
Pertanto, in condizioni normali, non abbiamo bisogno di effettuare alcuna modifica sul codice dei nostri controller, come si evince dall'esempio seguente.

[HttpPost("Person")]

public IActionResult Post([FromBody] PersonViewModel value)

{

if (ModelState.IsValid)

return Ok();

return BadRequest(ModelState);

}

#### Conclusioni

Oltre a quanto descritto, il framework introduce anche funzionalità avanzate che permettono di verificare relazioni tra diverse proprietà, facilitazioni sul testing, ecc., di cui parleremo nei prossimi script.  
  
Potete trovare il codice esempio che abbiamo descritto in questo script al seguente indirizzo: <https://github.com/andreatosato/fluentvalidationsample>