

Sezione 20

Autorizzazione

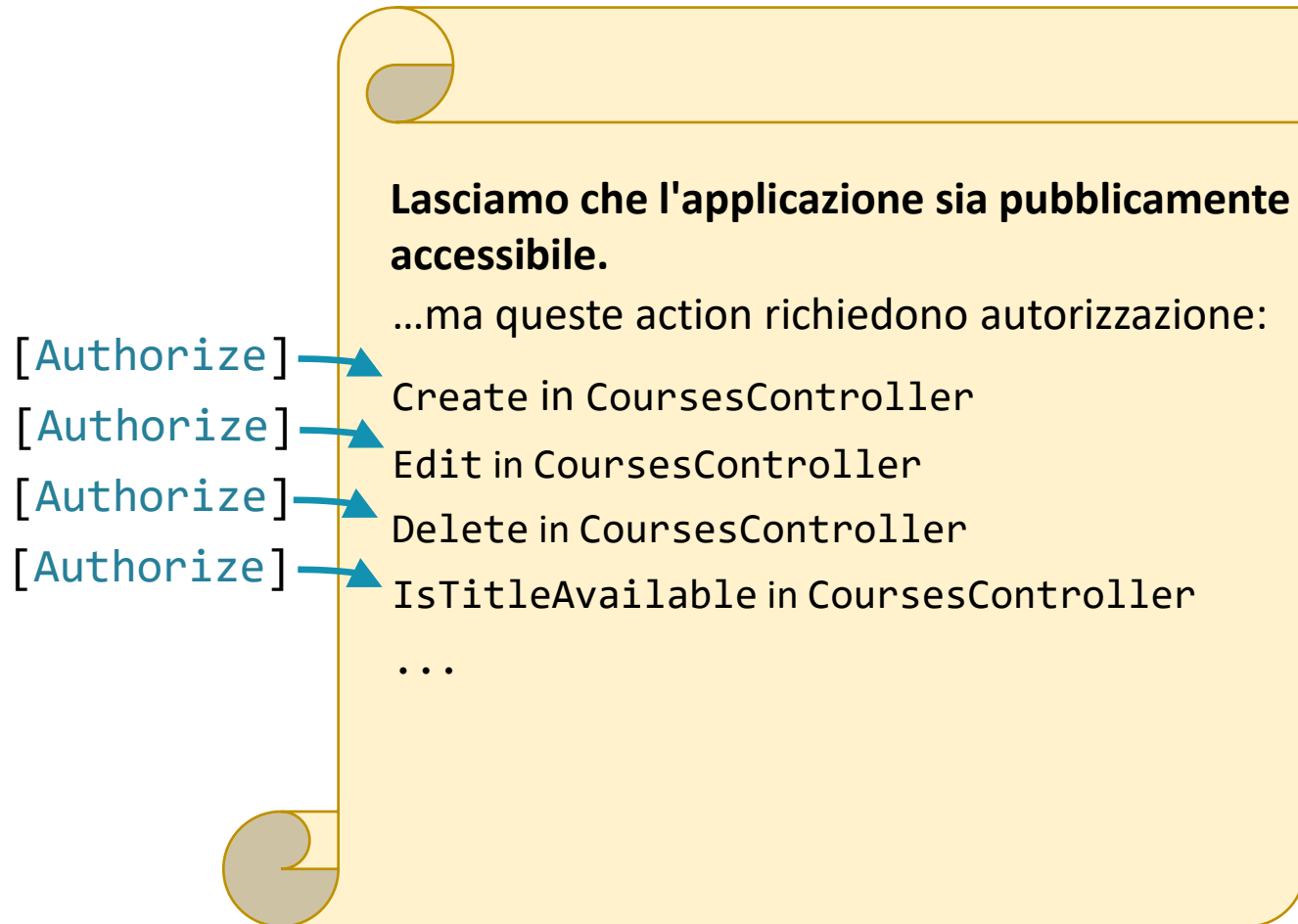
L'attributo Authorize

Lo possiamo porre in corrispondenza di un'action MVC

```
public class CoursesController : Controller {  
    [Authorize]  
    public IActionResult Create() {  
        //...  
    }  
}
```

Così consentiamo l'accesso solo agli utenti "autenticati", cioè a color che hanno fatto il login. In questo modo è protetta da accessi anonimi

L'attributo Authorize: approccio opt-in



Attenzione: dovremo ricordarci di porre l'attributo Authorize su ogni nuova action che intendiamo proteggere da accessi anonimi

L'attributo Authorize

In alternativa, lo posizioniamo sul controller;

Tutte le sue action sono sottoposte ad autorizzazione.

```
[Authorize]
public class CoursesController : Controller {

    public IActionResult Index() { ← Protetta
        //...
    }

    public IActionResult Create() { ← Protetta
        //...
    }
}
```

L'attributo AllowAnonymous: approccio opt-out

Tutte le action del controller richiedono autorizzazione

...tranne queste che sono pubblicamente accessibili:

[AllowAnonymous]



Index in CoursesController

...

Regolare l'accesso alle action MVC

Se poniamo l'attributo `Authorize` sul controller, poi possiamo riabilitare l'accesso anonimo ad una singola action con l'attributo `AllowAnonymous`

```
[Authorize]
public class CoursesController : Controller {
    [AllowAnonymous]
    public IActionResult Create() {           ← Accessibile anonimamente
        //...
    }
    // Le altre action saranno comunque protette
}
```

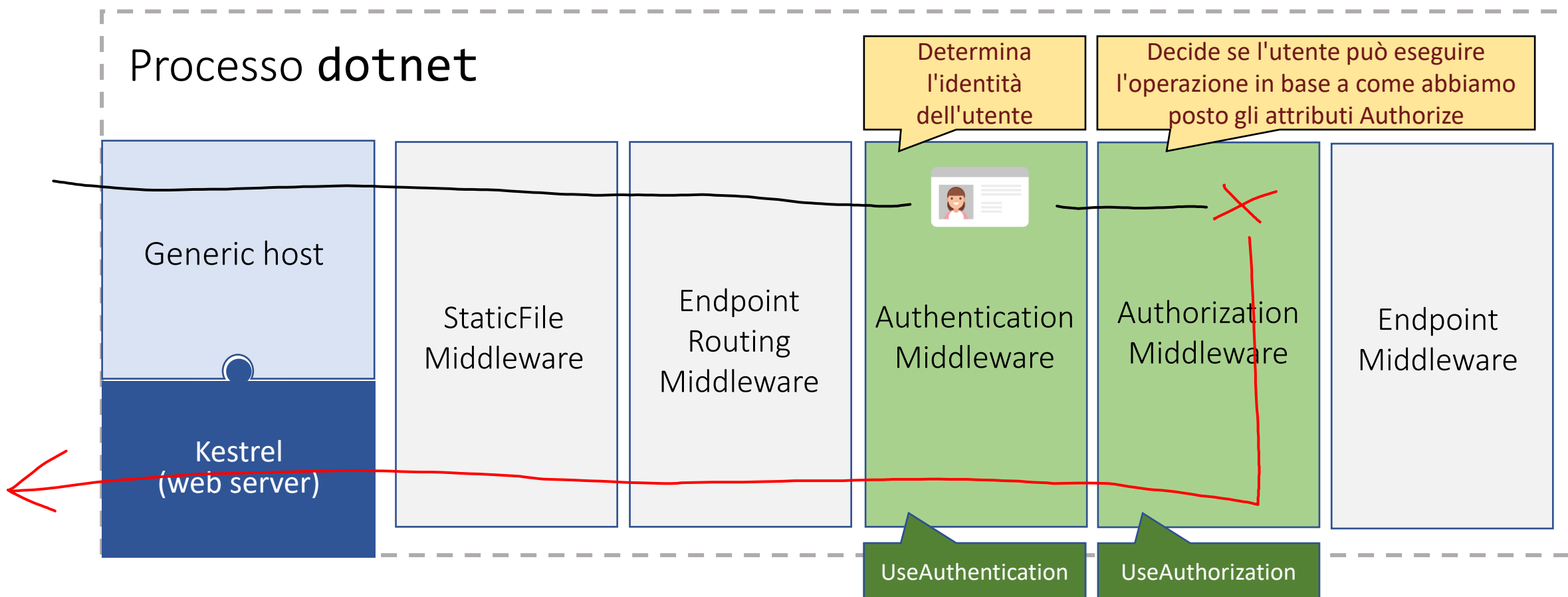
L'attributo Authorize è solo un "marcatore"

La funzionalità di autorizzazione è svolta dall'AuthorizationMiddleware

Nel metodo Configure della classe Startup:

```
app.UseAuthentication();  
app.UseAuthorization();
```

L'attributo Authorize è solo un "marcatore"



AuthorizeFilter

Lo possiamo configurare globalmente;

Così **tutte** le action di **tutti** i controller saranno protette da accessi anonimi.

Nel metodo `ConfigureServices` della classe `Startup`

```
services.AddMvc(options => {  
  
    AuthorizationPolicyBuilder policyBuilder = new();  
    AuthorizationPolicy policy = policyBuilder.RequireAuthenticatedUser().Build();  
    AuthorizeFilter filter = new(policy);  
    options.Filters.Add(filter);  
  
});
```

Alternativa (consigliata) ad AuthorizeFilter


L'AuthorizeFilter esiste ancora in ASP.NET Core per motivi di retrocompatibilità ma viene eseguito esternamente all'AuthorizationMiddleware e questo potrebbe produrre effetti collaterali (es. logica di autorizzazione eseguita di nuovo, inutilmente). Dunque, da .NET Core 3.1 è preferibile usare **RequireAuthorization()**.


Nel metodo Configure della classe Startup

```
app.UseEndpoints(routeBuilder => {  
    routeBuilder.MapControllerRoute("default", "{controller=Home}/{action=Index}/{id?}")  
        .RequireAuthorization();  
    routeBuilder.MapRazorPages()  
        .RequireAuthorization();  
});
```

Autorizzare l'accesso alle Razor Pages

Porre gli attributi [Authorize] o [AllowAnonymous] sul PageModel (infatti sui page handler non funzionano)

```
[Authorize]
public class ContactModel : PageModel
{
    
    public async Task<IActionResult> OnGetAsync(int id) {
        // ...
    }

    
    public async Task<IActionResult> OnPostAsync(int id) {
        // ...
    }
}
```

Fai una domanda al docente

 Invia la domanda

Punto n° 7

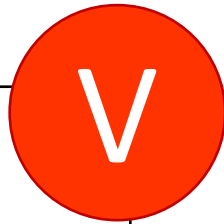
Realizzare una pagina di contatto da cui lo studente potrà inviare domande al docente.

Un form di contatto con Razor Pages

```
@page
@model ContactModel

<h1>@Model.Course.Title</h1>

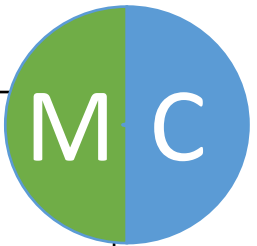
<form method="POST">
  <textarea asp-for="Question">
</textarea>
  <button type="submit">Invia
</button>
</form>
```



```
public class ContactModel : PageModel

    public CourseDetailViewModel Course
        { get; private set; }

    [Required]
    [BindProperty]
    public string Question { get; set; }
}
```



Attributo BindProperty

Da porre sulle proprietà che vogliamo siano valorizzate dal model binder.

```
[BindProperty]
public UserRoleInputModel Input { get; set; }
```

*Valori forniti con richieste POST
(invio di un form con method="post")*

```
[BindProperty(SupportsGet = true)]
public Role InRole { get; set; }
```

*Valori forniti con richieste GET
(link o form con method="get")*

Funziona sia nel PageModel di una RazorPage che in un Controller MVC

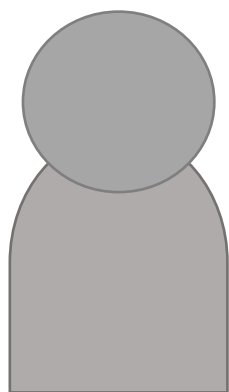
Autorizzare l'accesso alle Razor Pages

In alternativa, possiamo agire dal metodo **AddRazorPages**.

Nel metodo `ConfigureServices` della classe `Startup`

```
services.AddRazorPages(options => {  
  
    options.Conventions.AllowAnonymousToPage("/Privacy");  
    options.Conventions.AllowAnonymousToFolder("/Public");  
    options.Conventions.AuthorizePage("/Contact");  
    options.Conventions.AuthorizeFolder("/Admin");  
  
    options.Conventions.AuthorizeAreaFolder("Identity", "/Account/Manage");  
    options.Conventions.AllowAnonymousToAreaPage("Identity", "/Account/Login");  
  
});
```

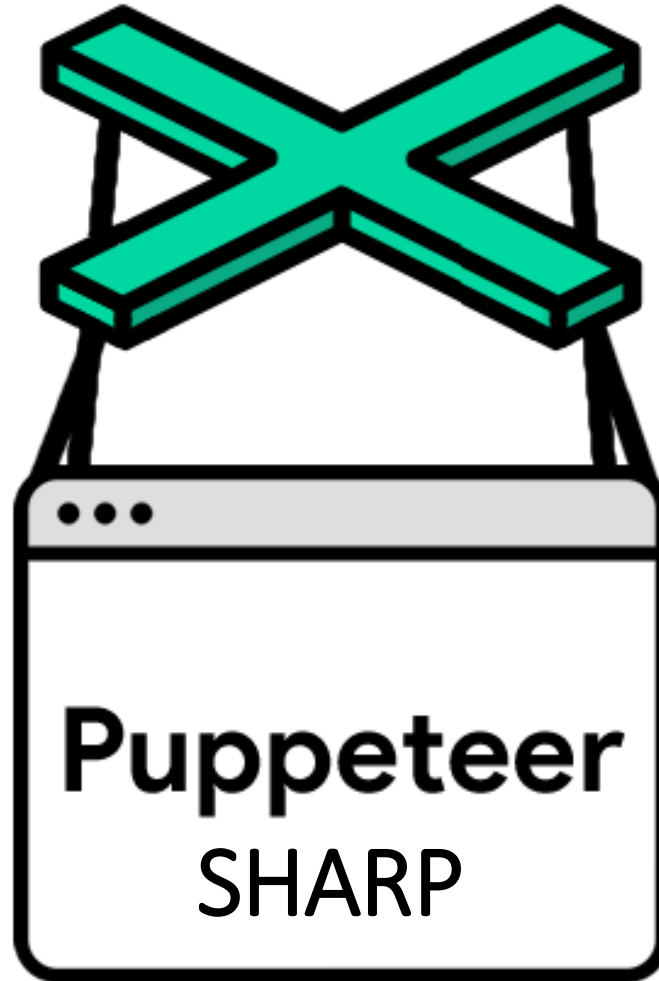
Autorizzare solo gli utenti "umani"



Umano

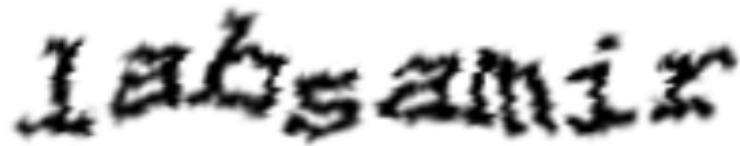


La navigazione in un sito è facilmente automatizzabile con strumenti come Puppeteer



Distinguere umani dai bot con un "CAPTCHA"

CAPTCHA Security check



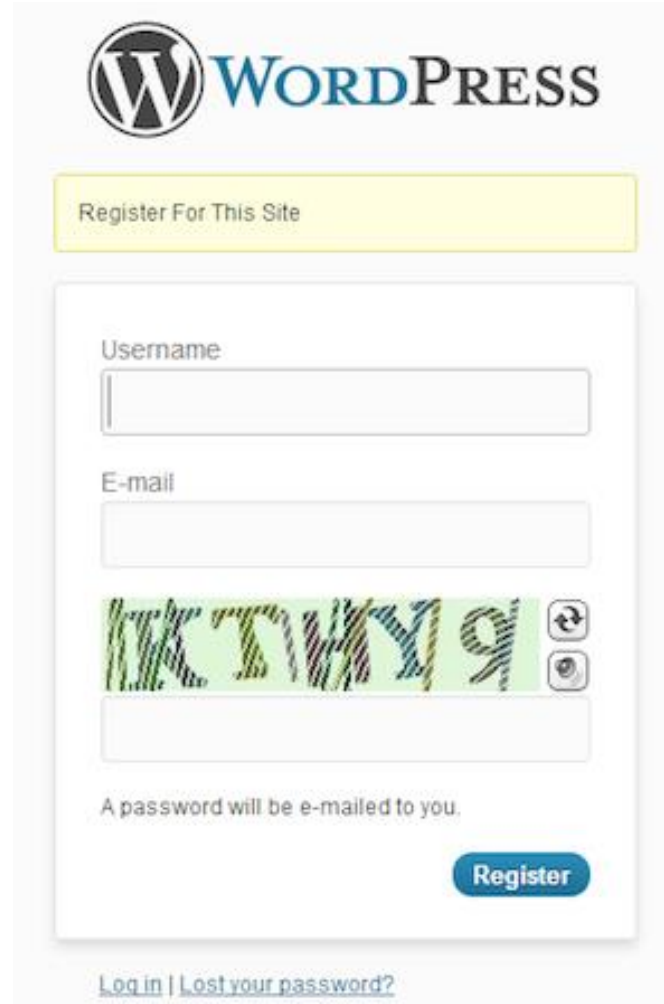
labsamix

[↻ Refresh](#)

Enter the text you see on the image

Can't see the image? [Request an account](#)

Distinguere umani dai bot con un "CAPTCHA"



The image shows a WordPress registration form. At the top is the WordPress logo. Below it is a yellow button labeled "Register For This Site". The form itself is a white box with a light gray border. It contains three input fields: "Username", "E-mail", and a CAPTCHA field. The CAPTCHA field displays a colorful, abstract image with the text "KTYH9" overlaid. To the right of the CAPTCHA image are two small icons: a refresh icon and a volume icon. Below the CAPTCHA field is a text label "A password will be e-mailed to you." and a blue "Register" button. At the bottom of the form are two links: "Log in" and "Lost your password?".

WordPress

Register For This Site

Username

E-mail

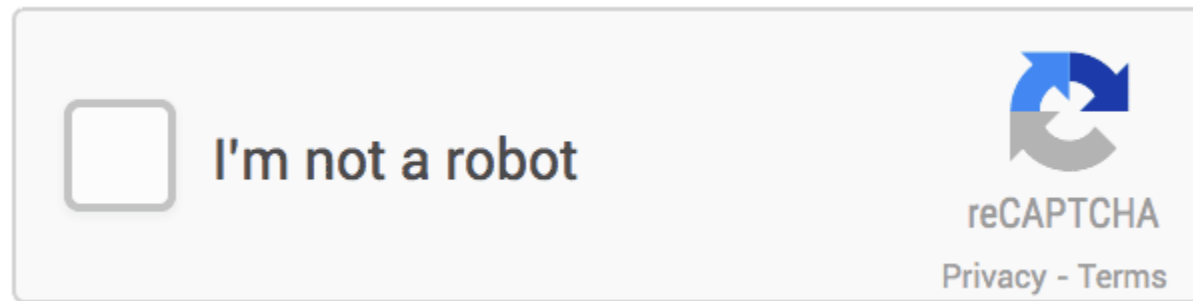
KTYH9

A password will be e-mailed to you.

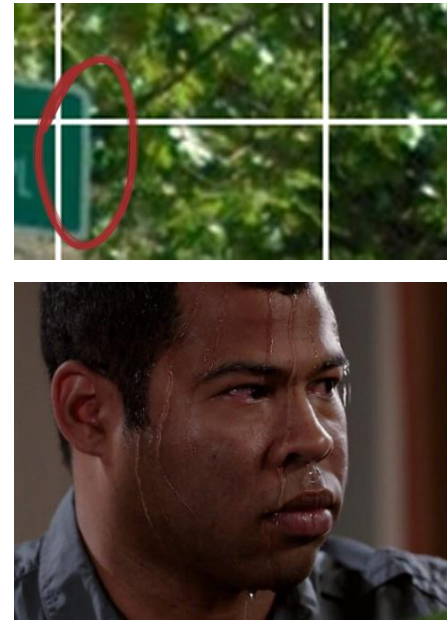
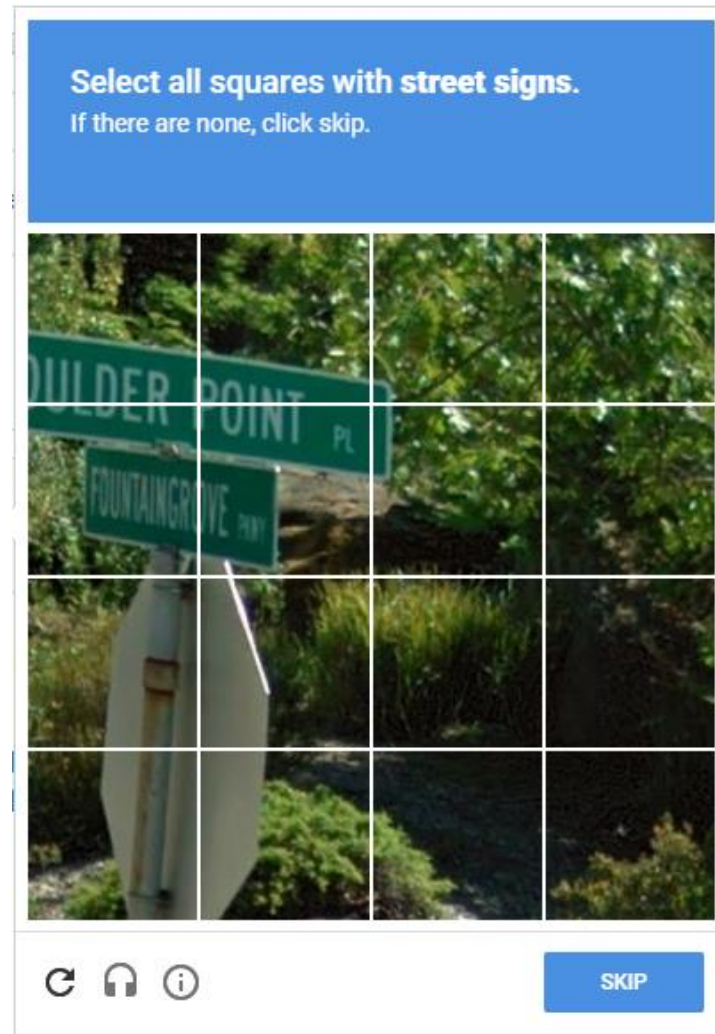
Register

[Log in](#) | [Lost your password?](#)

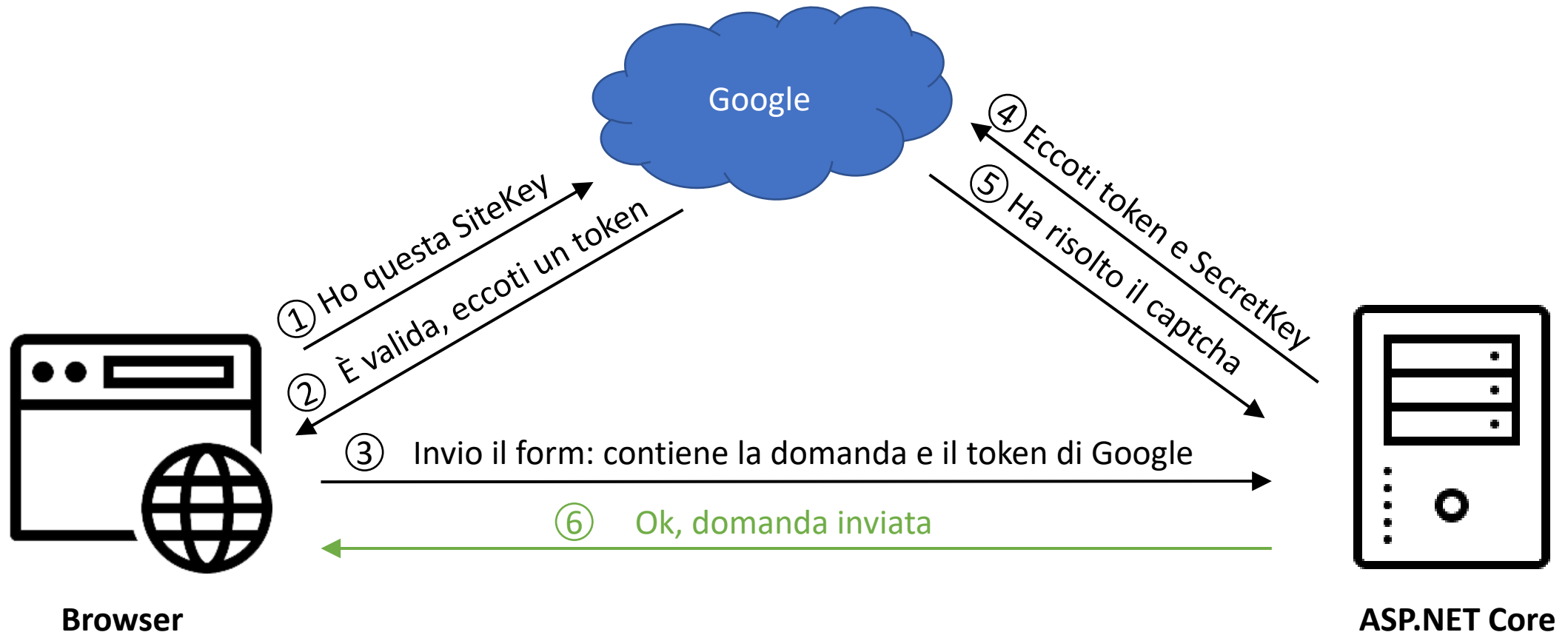
Distinguere umani dai bot con un "CAPTCHA"



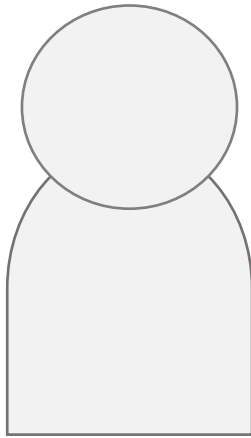
Distinguere umani dai bot con un "CAPTCHA"



Funzionamento di reCaptcha v2



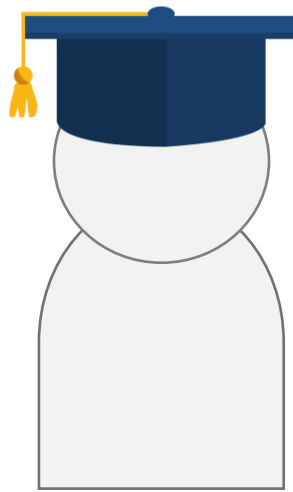
Ruoli dell'applicazione MyCourse



Utente "base"

Privo di ruolo

Inviare domande, iscriversi ai corsi, vedere le lezioni ed esprimere una valutazione.



Docente

Teacher

Crea, modifica ed elimina un corso, pubblica le lezioni, ...



Amministratore

Administrator

Assegna ruoli agli altri utenti

Modifica corso

Salva



Titolo

Descrizione

B *I*

Lorem ipsum dolor sit amet consectetur adipisicing elit. Et minima sunt quia nulla voluptate, illum eum incidunt repudiandae beatae, vero accusantium minus hic eveniet omnis laborum architecto inventore dolores molestias? Placeat sequi sapiente hic culpa optio quisquam est fugiat dolorem itaque non, quasi cum voluptates quidem repudiandae doloribus? Autem mollitia esse odio nihil atque non ea quisquam consequuntur exercitationem? Amet! Non ut itaque qui tempore illum! Amet, accusamus minima. Ut rerum praesentium obcaecati sint, accusantium maxime odio voluptatibus quaerat repudiandae corrupti magnam, non perferendis. Officia

Email di contatto

Prezzo intero

Prezzo corrente

Immagine rappresentativa



Punto n° 22

Le pagine di inserimento, modifica ed eliminazione dei corsi devono essere accessibili solo ai docenti.

Gestione utenti

Email dell'utente

Ruolo assegnato

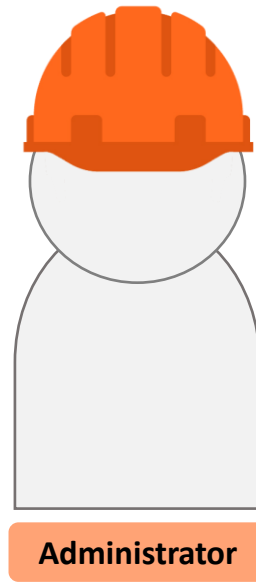


Imposta ruolo

Punto n° 23

Solo un utente amministratore può assegnare i ruoli agli utenti.

È possibile assegnare/rimuovere un ruolo con la stessa facilità con cui si mette e toglie un cappello



Un utente può assumere più ruoli



Il ruolo è assegnato all'utente come claim

ClaimsIdentity



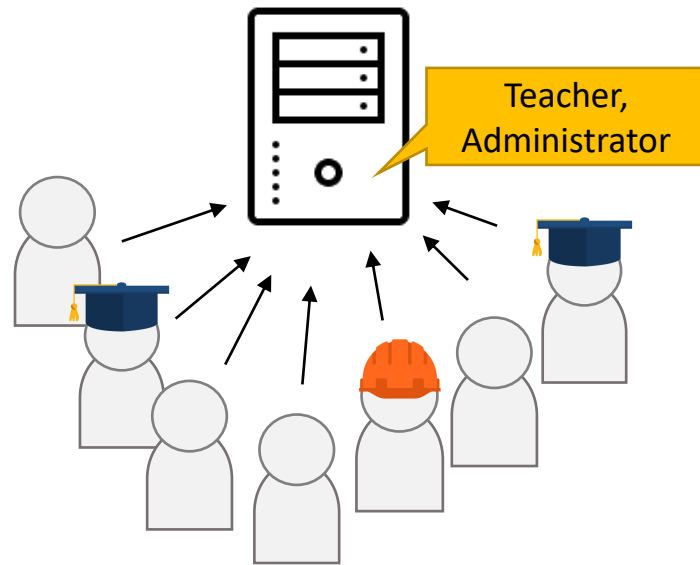
FullName	Laura Bianchi
Email	lb@email.com
Role	Administrator
Role	Teacher

Assegnare un ruolo a un utente

Soluzione #1

`userManager<ApplicationUser>`

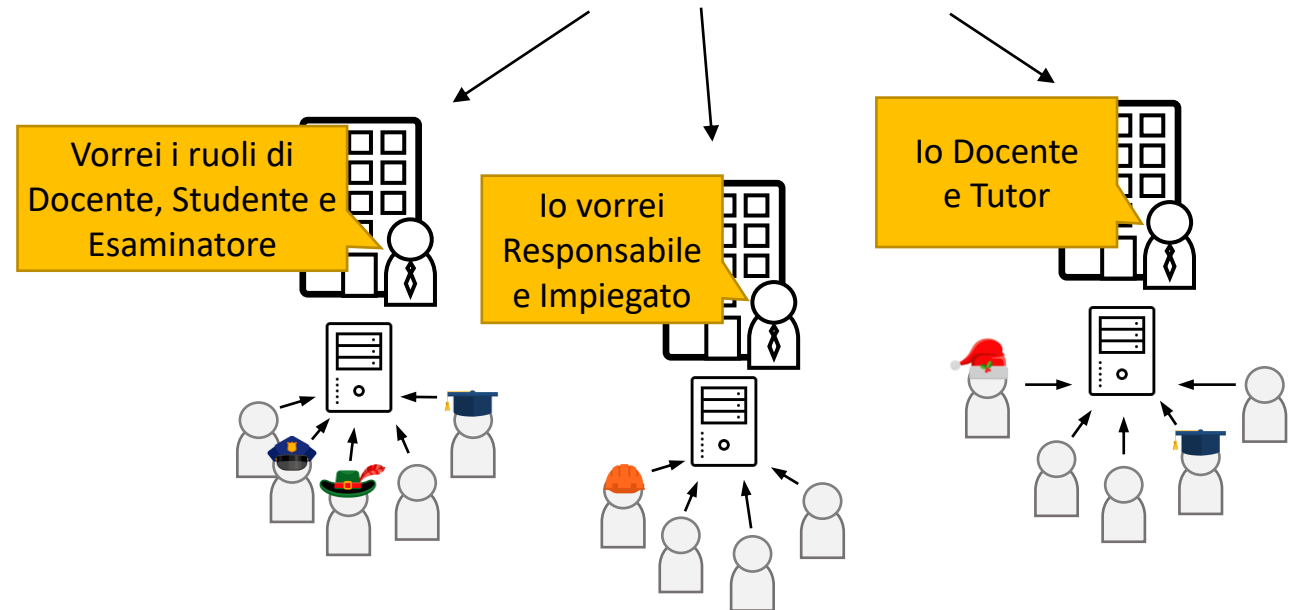
Servizi (es. MyCourse)



Soluzione #2

`RoleManager<IdentityRole>`

Prodotti (es. e-learning)



Servizi e prodotti

Soluzione #1

userManager<ApplicationUser>

Servizi (es. MyCourse)

Solo se ha il ruolo Teacher

```
[Authorize( )]  
public IActionResult Create()  
{  
  
}
```

Soluzione #2

RoleManager<IdentityRole>

Prodotti (es. e-learning)

Solo se ha il claim con claim type "Permission"
e claim value "courses.create"

```
[Authorize( )]  
public IActionResult Create()  
{  
  
}
```

Assegnare un ruolo a un utente

Soluzione #1

UserManager<ApplicationUser>

Assegnare il claim del ruolo all'utente

Soluzione #1

```
UserManager<ApplicationUser>
```

```
ApplicationUser user = await userManager.FindByEmailAsync("account@example.com");
```

<http://schemas.microsoft.com/ws/2008/06/identity/claims/role>

```
Claim claim = new Claim(ClaimTypes.Role, "Administrator");
```

```
IdentityResult result = await userManager.AddClaimAsync(user, claim);
```

```
if (result.Succeeded)
```

```
{
```

```
    // OK, il claim è stato assegnato
```

```
}
```


Assegnare un ruolo a un utente

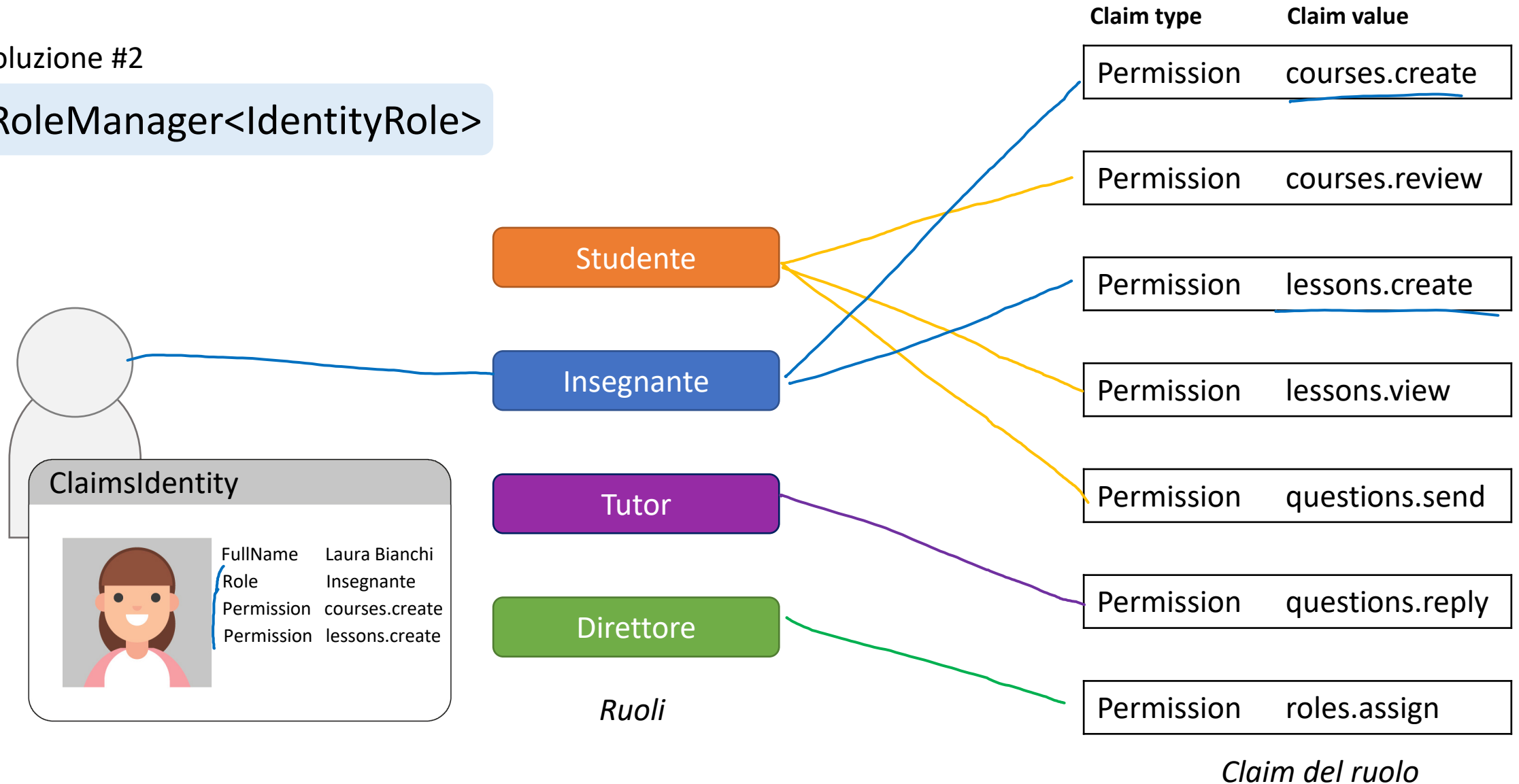
Soluzione #2

```
RoleManager<IdentityRole>
```

Il RoleManager fornire una soluzione più strutturata

Soluzione #2

RoleManager<IdentityRole>



Registrazione il RoleManager<TRole> (1/4)

Soluzione #2

RoleManager<IdentityRole>

Nel metodo ConfigureServices della classe Startup

```
services.AddDefaultIdentity<ApplicationUser>()  
    .AddRoles<IdentityRole>()  
    .AddRoleManager<RoleManager<IdentityRole>>();
```

Aggiungere un ruolo in anagrafica (2/4)

Soluzione #2

```
RoleManager<IdentityRole>
```

```
IdentityRole role = new IdentityRole(roleName);  
IdentityResult result = await roleManager.CreateAsync(role);  
if (result.Succeeded)  
{  
    // Tutto ok  
}
```

Assegnare un claim al ruolo (3/4)

Soluzione #2

```
RoleManager<IdentityRole>
```

```
IdentityRole role = await roleManager.FindByNameAsync(roleName);  
Claim claim = new Claim("Permission", claimValue);  
IdentityResult result = await roleManager.AddClaimAsync(role, claim);
```

Assegnare il ruolo a un utente (4/4)

Soluzione #2

```
RoleManager<IdentityRole>
```

```
ApplicationUser user = await userManager.FindByNameAsync(userName);  
IdentityResult result = await userManager.AddToRoleAsync(user, roleName);  
if (result.Succeeded)  
{  
    // Tutto ok  
}
```

Decidere come associare un'informazione all'utente

Come Claim	Come proprietà di ApplicationUser
Persistito nella tabella AspNetUserClaims	Persistito nella tabella AspNetUsers
In formato chiave-valore (stringhe)	Valori di vario tipo e oggetti complessi
Viene automaticamente aggiunto alla ClaimsIdentity	Può essere aggiunto alla ClaimsIdentity con una UserClaimsPrincipalFactory
Utile se abbiamo bisogno del valore ad ogni richiesta	Utile per informazioni anagrafiche che usiamo in pagine specifiche

Come faccio a sapere chi ha il ruolo di Docente e Amministratore in questo momento?

Ok!

Mostriamo un elenco subito sotto:

Gestione utenti

Indirizzo email

severino.padovano@example.com

Ruolo

Docente

Operazione

Assegna

Revoca

{ _____

Elencare gli utenti in un ruolo

Soluzione #1

```
UserManager<ApplicationUser>
```

```
Claim claim = new (ClaimTypes.Role, "Administrator");  
IList<ApplicationUser> usersInRole = await userManager.GetUsersForClaimAsync(claim);
```

Soluzione #2

```
RoleManager<IdentityRole>
```

```
string roleName = "Administrator";  
IList<ApplicationUser> usersInRole = await userManager.GetUsersInRoleAsync(roleName);
```

Filtrare, paginare e ordinare gli utenti

`IQueryable<ApplicationUser>`

```
IList<ApplicationUser> users = await userManager.Users
    .Where(user => user.UserClaims.Any(claim =>
        claim.ClaimType == ClaimTypes.Role &&
        claim.ClaimValue == "Administrator"))
    .OrderBy(user => user.FullName)
    .Skip(0)
    .Take(20)
    .ToListAsync();
```

*Solo per EFCore, perché con ADO.NET è
difficile implementare
`IQueryable<ApplicationUser>`*

*Se proviamo ad usare la proprietà User
con ADO.NET, otterremmo un'eccezione*

An unhandled exception occurred while processing the request.

NotSupportedException: Store does not implement IQueryableUserStore<TUser>.

Microsoft.AspNetCore.Identity.UserManager<TUser>.get_Users()

Stack

Query

Cookies

Headers

Routing

NotSupportedException: Store does not implement IQueryableUserStore<TUser>.

Microsoft.AspNetCore.Identity.UserManager<TUser>.get_Users()

MyCourse.Pages.Admin.UsersModel.OnPostAssignAsync() in **Users.cshtml.cs**

+ 49. **var a = userManager.Users.ToList();**

Microsoft.AspNetCore.Mvc.RazorPages.Infrastructure.ExecutorFactory+GenericTaskHandlerMethod.Convert<T>(object taskAsObject)

Microsoft.AspNetCore.Mvc.RazorPages.Infrastructure.ExecutorFactory+GenericTaskHandlerMethod.Execute(object receiver, object[] arguments)

Microsoft.AspNetCore.Mvc.RazorPages.Infrastructure.PageActionInvoker.InvokeHandlerMethodAsync()

Microsoft.AspNetCore.Mvc.RazorPages.Infrastructure.PageActionInvoker.InvokeNextPageFilterAsync()

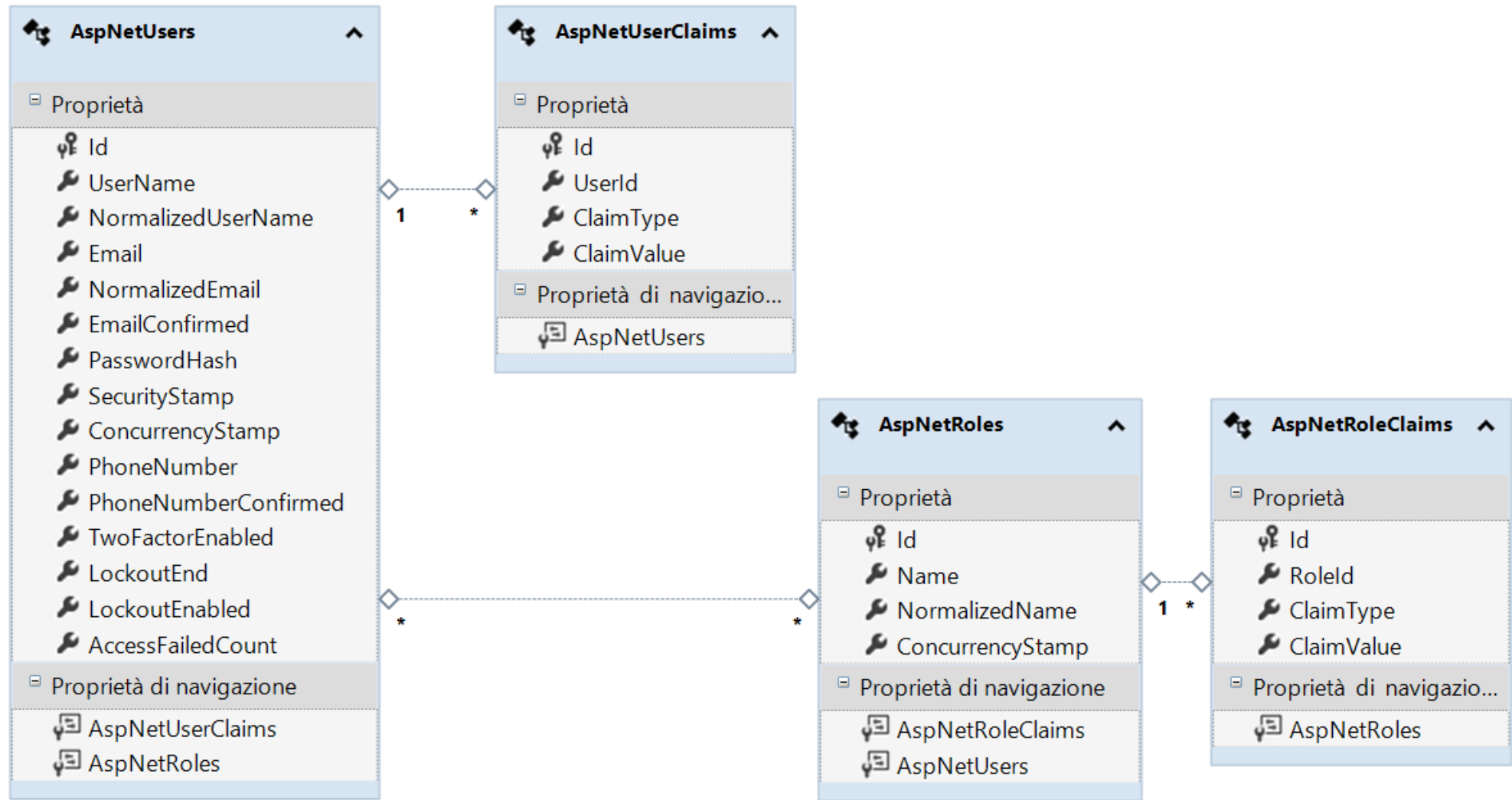
Microsoft.AspNetCore.Mvc.RazorPages.Infrastructure.PageActionInvoker.Rethrow(PageHandlerExecutedContext context)

Pazienza, se lo UserManager non fa per noi...

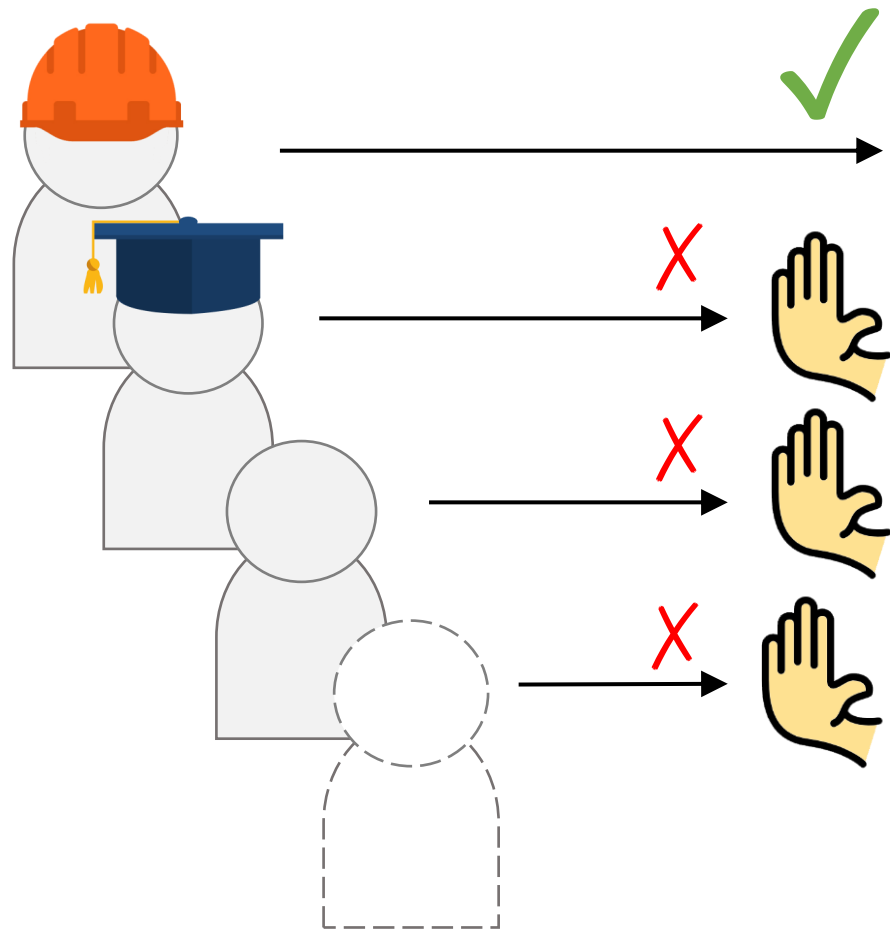
...possiamo pur sempre creare un nostro servizio applicativo.

```
public interface ICourseService
{
    Task<ListViewModel<CourseViewModel>> GetCoursesAsync(CourseListInputModel model);
    // ...
}
```

```
public interface IUserService
{
    Task<ListViewModel<UserViewModel>> GetUsersAsync(UserListInputModel model);
    // ...
}
```



Autorizzare l'accesso in base al ruolo



Gestione utenti

Indirizzo email	Ruolo	Operazione
<input type="text" value="Email"/>	<div>Amministratore ▼</div>	<div>Assegna Revoca</div>
<div><div>Amministratore</div><div>Docente</div></div>		
<hr/>		
Severino Padovano	severino.padovano@example.com	
<hr/>		
Guglielma Rivo	guglielma.rivo@example.com	
<hr/>		
Anastasia Trentini	anastasia.trentini@example.com	
<hr/>		
Vanna Palermo	vanna.palermo@example.com	
<hr/>		
Adelinda Greco	adelinda.greco@example.com	
<hr/>		

Autorizzare l'accesso in base al ruolo

Usiamo l'attributo `Authorize` sul Page Model della Razor Page, o su action o controller MVC.

```
[Authorize(Roles = "Administrator")]  
public class UsersModel : PageModel  
{  
  
}
```

*Ricorda: nelle Razor Page,
l'attributo `Authorize` funziona
solo sul `PageModel`!*

Autorizzare l'accesso in base al ruolo (1/2)

Usiamo l'attributo `Authorize` sul Page Model della Razor Page

```
[Authorize(Roles = nameof(Role.Administrator))]
```

```
public class UsersModel : PageModel
```

```
{
```

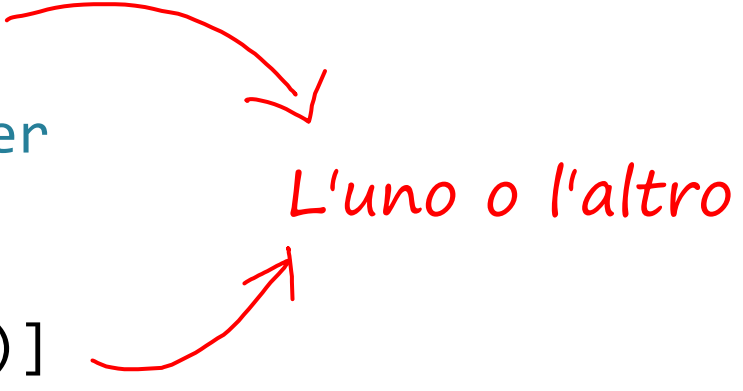
```
}
```

*Ricorda: nelle Razor Page,
l'attributo `Authorize` funziona
solo sul `PageModel`!*

Autorizzare l'accesso in base al ruolo (2/2)

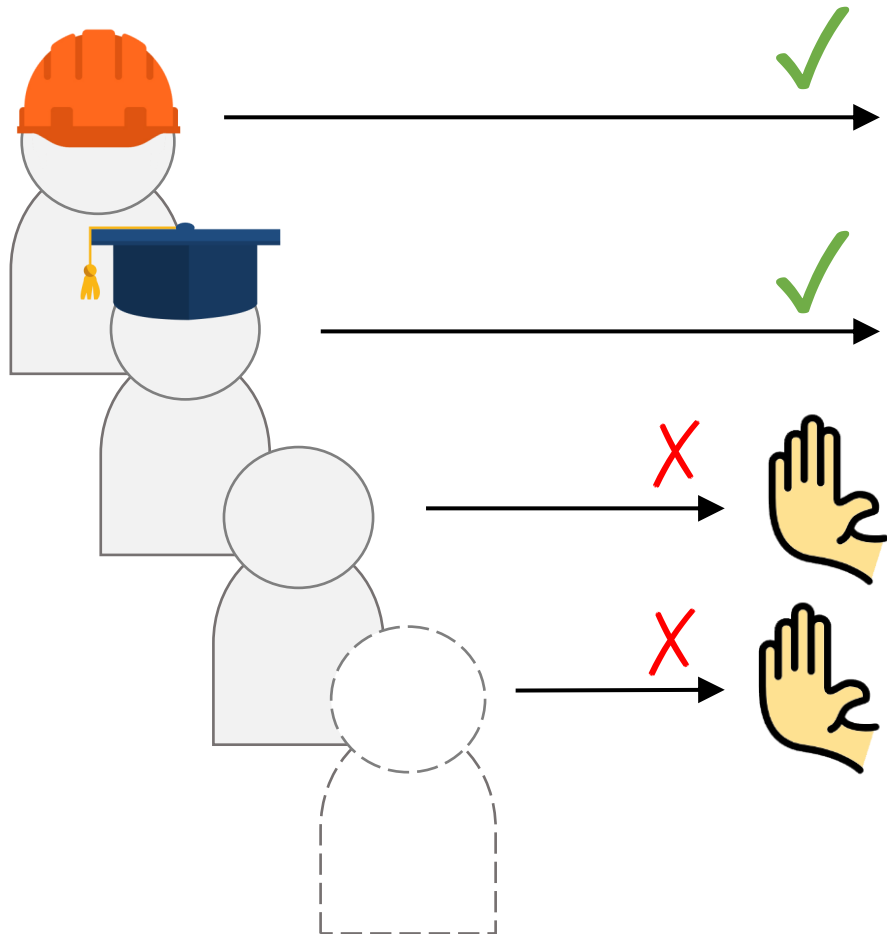
Usiamo l'attributo `Authorize` su action o controller MVC

```
[Authorize(Roles = nameof(Role.Teacher))]  
public class CoursesController : Controller  
{  
    [Authorize(Roles = nameof(Role.Teacher))]  
    public IActionResult Edit()  
    {  
    }  
}
```



L'uno o l'altro

Autorizzare l'accesso a due o più ruoli

[illegible]

Autorizzare l'accesso a due o più ruoli (OR)

Usiamo l'attributo `Authorize` indicando i ruoli separati da virgola

```
[Authorize(Roles = "Administrator,Teacher")]  
public class UsersModel : PageModel  
{  
  
}
```

Autorizzare l'accesso a due o più ruoli (OR)

Usiamo l'attributo `Authorize` indicando i ruoli separati da virgola

```
[Authorize(Roles = nameof(Role.Administrator) + "," + nameof(Role.Teacher))]  
public class UsersModel : PageModel  
{  
  
}
```

Autorizzare l'accesso in base al ruolo (AND)

Gli attributi Authorize sono componibili

```
[Authorize(Roles = nameof(Role.Teacher))]  
public class CoursesController : Controller  
{  
    [Authorize(Roles = nameof(Role.Administrator))]  
    public IActionResult Report ()  
    {  
    }  
}
```

In questo caso l'utente dovrà possedere sia il ruolo Teacher che Administrator per riuscire ad accedere all'action Report

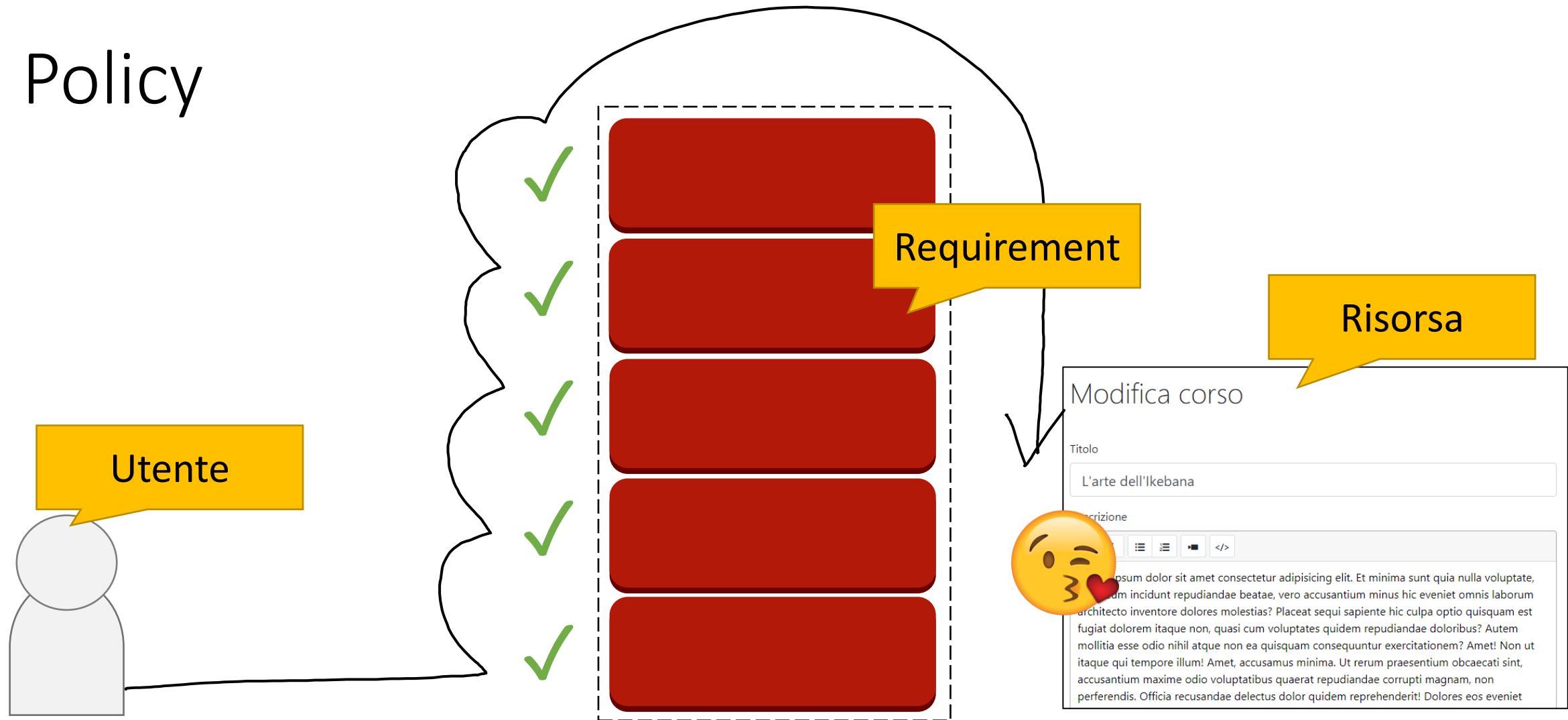
Autorizzare l'accesso in base al ruolo

Gli attributi Authorize sono componibili

```
public class CoursesController : Controller
{
    [Authorize(Roles = nameof(Role.Teacher))]
    [Authorize(Roles = nameof(Role.Administrator))]
    public IActionResult Report ()
    {
    }
}
```

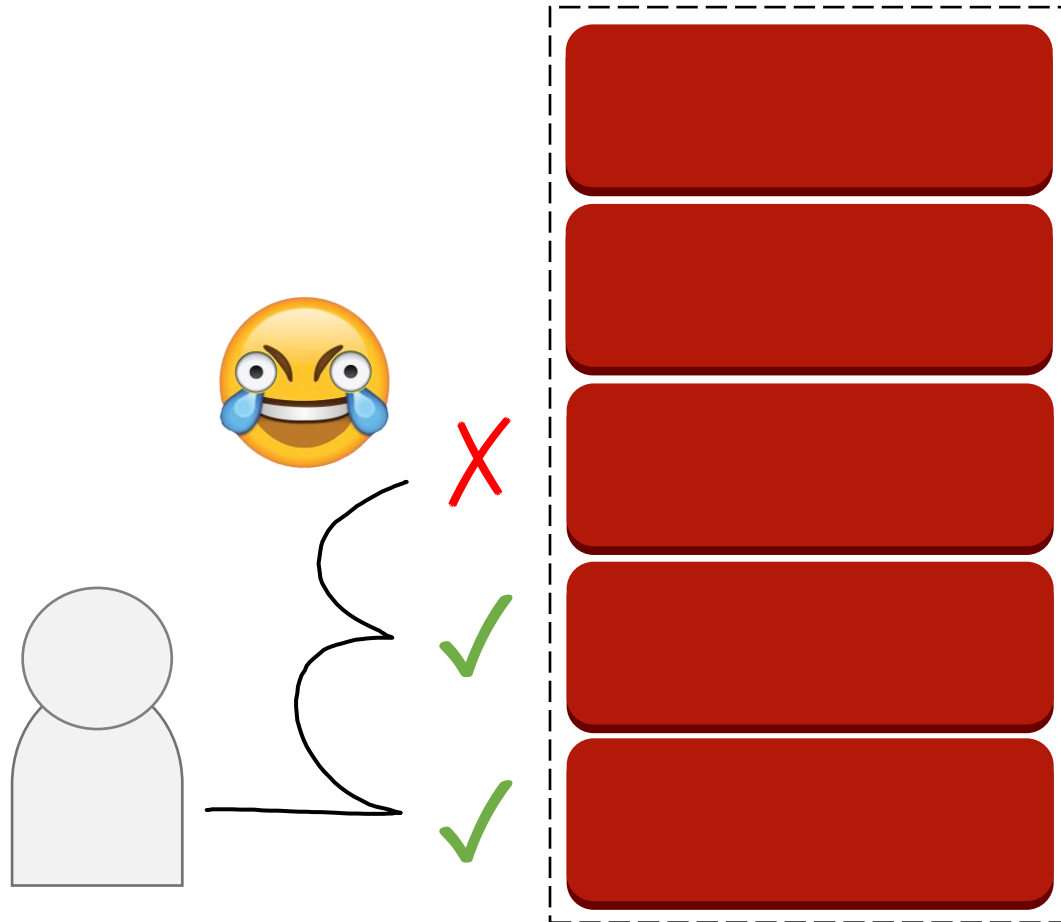
Anche in questo caso l'utente dovrà possedere sia il ruolo Teacher che Administrator per riuscire ad accedere all'action Report

Policy



La policy è come un muro composto di mattoncini (Requirement) che separa l'utente dalla risorsa (pagina)

Policy



Modifica corso

Titolo

L'arte dell'Ikebana

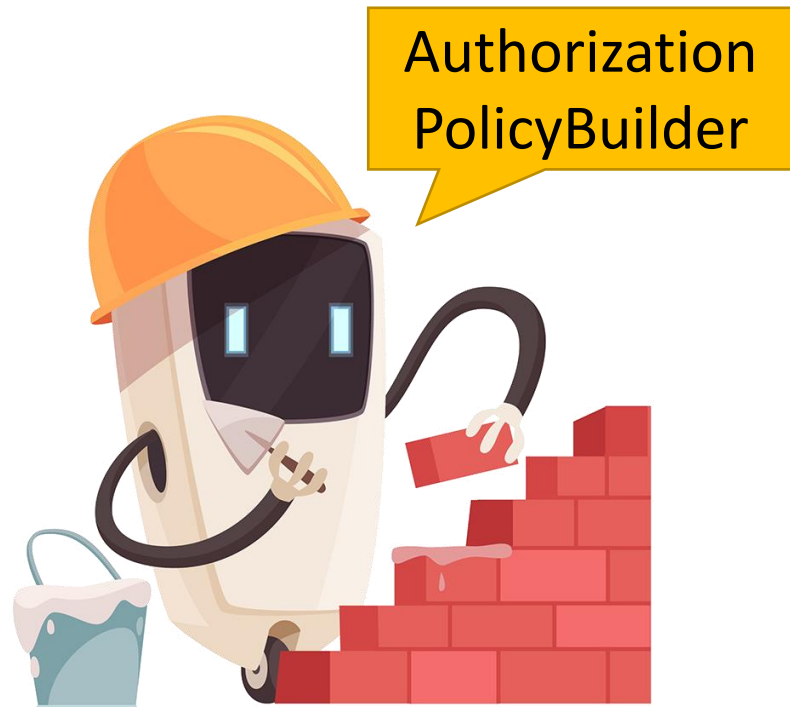
Descrizione

B *I*

Lorem ipsum dolor sit amet consectetur adipisicing elit. Et minima sunt quia nulla voluptate, illum eum incidunt repudiandae beatae, vero accusantium minus hic eveniet omnis laborum architecto inventore dolores molestias? Placeat sequi sapiente hic culpa optio quisquam est fugiat dolore itaque non, quasi cum voluptates quidem repudiandae doloribus? Autem mollitia esse odio nihil atque non ea quisquam consequuntur exercitationem? Amet! Non ut itaque qui tempore illum! Amet, accusamus minima. Ut rerum praesentium obcaecati sint, accusantium maxime odio voluptatibus quaerat repudiandae corrupti magnam, non perferendis. Officia recusandae delectus dolor quidem reprehenderit! Dolores eos eveniet

*Se l'utente NON soddisfa anche un solo requirement,
NON riuscirà ad accedere alla risorsa*

Policy



Un requirement può portare con sé delle informazioni

```
.RequireRole("Teacher", "Administrator")
```

Aggiungere una policy al registro

Nel metodo **ConfigureServices** della classe **Startup**

```
services.AddAuthorization(options => {  
    options.AddPolicy("CourseAuthor", builder => {  
        builder.RequireAuthenticatedUser();  
        builder.RequireRole("Teacher");  
        builder.RequireClaim("Country", "Italy");  
        builder.Requirements.Add(new CourseAuthorRequirement());  
    });  
});
```



Authorization
PolicyBuilder

Creare il requirement

Creiamo una classe che implementa **IAuthorizationRequirement**

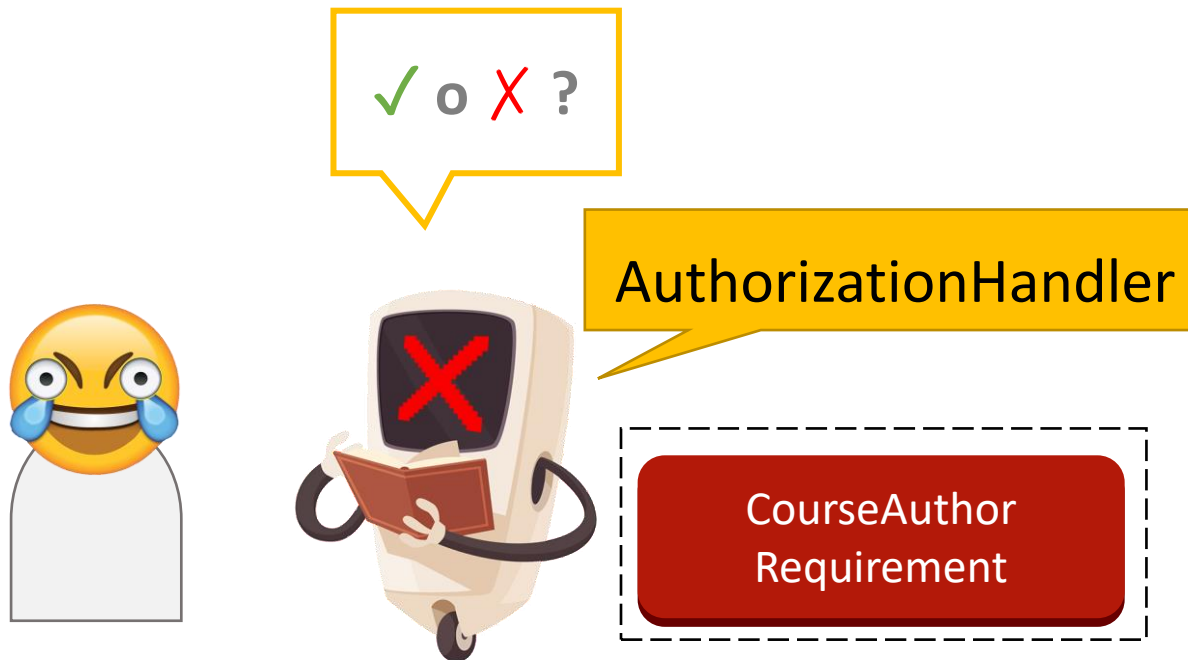
```
public class CourseAuthorRequirement : IAuthorizationRequirement  
{  
  
}
```

Creare il requirement

Il requirement può contenere delle informazioni

```
public class CourseLimitRequirement : IAuthorizationRequirement {  
    public CourseLimitRequirement(int limit) {  
        Limit = limit;  
    }  
    public int Limit { get; }  
}
```

Policy: verifica dei requirement



Modifica corso

Titolo

L'arte dell'Ikebana

Descrizione

B *I* `</>`

Lorem ipsum dolor sit amet consectetur adipisicing elit. Et minima sunt quia nulla voluptate, illum eum incidunt repudiandae beatae, vero accusantium minus hic eveniet omnis laborum architecto inventore dolores molestias? Placeat sequi sapiente hic culpa optio quisquam est fugiat dolore itaque non, quasi cum voluptates quidem repudiandae doloribus? Autem mollitia esse odio nihil atque non ea quisquam consequuntur exercitationem? Amet! Non ut itaque qui tempore illum! Amet, accusamus minima. Ut rerum praesentium obcaecati sint, accusantium maxime odio voluptatibus quaerat repudiandae corrupti magnam, non perferendis. Officia recusandae delectus dolor quidem reprehenderit! Dolores eos eveniet

Creare un AuthorizationHandler<T>

Verifica se l'utente soddisfa il requirement

```
public class CourseAuthorRequirementHandler : AuthorizationHandler<CourseAuthorRequirement> {  
    protected override async Task HandleRequirementAsync (AuthorizationHandlerContext context,  
                                                         CourseAuthorRequirement requirement) {  
        if (condizione) {  
            context.Succeed(requirement);  
        } else {  
            context.Fail();  
        }  
    }  
}
```

*Gli AuthorizationHandler
supportano la dependency injection*

Registrare l'AuthorizationHandler<T>

Si registrano per la dependency injection usando un opportuno ciclo di vita:

- **Singleton** se si limita a verificare la presenza o il contenuto dei claim nell'identità dell'utente o se sfrutta servizi registrati essi stessi con il ciclo di vita **Singleton**;
- **Scoped** se sfrutta servizi come un DbContext che sono stati essi stessi registrati con il ciclo di vita **Scoped** o **Transient**;

Si registrano nel metodo **ConfigureServices** della classe **Startup**:

```
// Uso Scoped perché questi servizi accedono al database usando un DbContext
services.AddScoped<IAuthorizationHandler, CourseAuthorRequirementHandler>();
services.AddScoped<IAuthorizationHandler, CourseLimitRequirementHandler>();
```

Applicare una policy

Si usa l'attributo **Authorize** su Controller, Action o PageModel

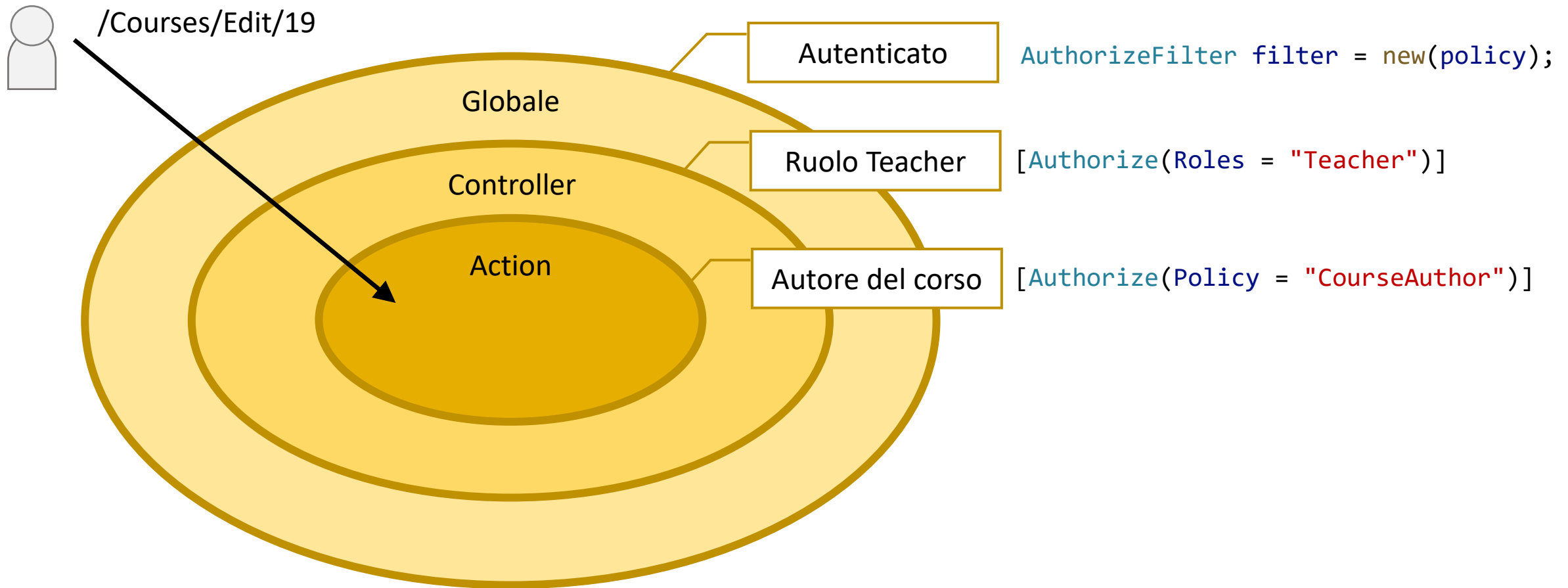
```
[Authorize(Policy = "CourseAuthor")]
```


Applicare una policy

Si usa l'attributo **Authorize** su Controller, Action o PageModel

```
[Authorize(Policy = nameof(Policy.CourseAuthor))]
```

Componibilità dei criteri di autorizzazione



Applicare una policy a livello globale

Usiamo `RequireAuthorization`

Nel metodo `Configure` della classe `Startup`

```
app.UseEndpoints(routeBuilder => {  
    routeBuilder.MapControllerRoute("default", "{controller=Home}/{action=Index}/{id?}")  
        .RequireAuthorization("NomePolicy");  
    routeBuilder.MapRazorPages()  
        .RequireAuthorization("NomePolicy");  
});
```

*RequireAuthorization ha
un overload che accetta il
nome di una policy*

Non c'è un limite al numero di corsi
che il docente può creare, però...

Se ne crea più di 5 vorrei
essere informata

Autorizzazione dichiarativa: attributo Authorize

```
[Authorize(Roles = nameof(Role.Teacher))]  
public class CoursesController : Controller  
{  
    [Authorize(Policy = "CourseLimit")]  
    public IActionResult Create()  
    {  
        // ...  
    }  
}
```

Autorizzazione imperativa: IAuthorizationService

```
[Authorize(Roles = nameof(Role.Teacher))]  
public class CoursesController : Controller {  
    [HttpPost]  
    public IActionResult Create([FromServices] IAuthorizationService authorizationService) {  
        // Persisto il corso nel database  
        AuthorizationResult result = await authorizationService.AuthorizeAsync(User, "CourseLimit");  
        if (!result.Succeeded) {  
            // Invio un'email all'amministratore  
        }  
        // Reindirizzo il docente  
    }  
}
```

Usare IAuthorizationService da una view Razor

Nel file `_LayoutImports.cshtml`

```
@using Microsoft.AspNetCore.Authorization
```

Nella view Razor

```
@model ...
```

```
@inject IAuthorizationService authService
```

```
@{  
    AuthorizationResult courseAuthorResult = await authService.AuthorizeAsync(User, "CourseAuthor");  
}
```


Overload del metodo AuthorizeAsync

1. Fornendo l'utente e il nome della policy

```
await authorizationService.AuthorizeAsync(User, "NomePolicy");
```

2. Fornendo l'utente e un'istanza di AuthorizationPolicy

```
AuthorizationPolicy policy = new AuthorizationPolicyBuilder()  
    .RequireRole("Administrator").Build();  
await authorizationService.AuthorizeAsync(User, policy);
```

3. Fornendo l'utente, la risorsa e la policy

```
await authorizationService.AuthorizeAsync(User, course.Id, "NomePolicy");
```


Account per l'accesso a MyCourse

Utente con ruolo Administrator

Email	Password
admin@example.com	Administrator1!

Utenti con ruolo Teacher

Email	Password
severino.padovano@example.com	Teacher1!
filiberta.castiglione@example.com	Teacher1!
guglielma.rivo@example.com	Teacher1!

Utente privo di ruolo

Email	Password
mario@example.com	Student1!

Progresso nella specifica

- Punto 7: pagina di contatto;
- Punto 22: autorizzazione alla modifica;
- Punto 23: assegnazione ruolo a utente.

Requisiti funzionali

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23		

Requisiti non funzionali

a	b	c	d
---	---	---	---