

Sezione 18

Autenticazione con ASP.NET Core Identity

Benvenuto su MyCourse!

Impara in maniera facile e divertente con i nostri corsi online.
Scegli da un vasto catalogo sempre a tua disposizione.

[Sfoglia il catalogo dei corsi ➤](#)



Punto n° 8

Nella pagina di dettaglio del corso deve trovarsi un bottone per far registrare lo studente.

Autenticazione

Determinare l'identità dell'utente in base a un documento valido, emesso da un'autorità affidabile ed esibito dall'utente stesso.



Foto di "Noi cambiamo" <https://www.noicambiamo.it/news/2016/09/11/nuova-proposta-del-m5s-per-la-selezione-degli-scrutatori-a-marino/seggio-elettorale/>

Autenticazione

In un'applicazione ASP.NET, un "documento valido" è un cookie di autenticazione che reca alcune informazioni (cifrate) dell'utente.

The screenshot shows the Network tab of the F12 developer tools in a browser. The 'Cookie' tab is selected. A red box highlights the 'Nome' (Name) and 'Valore' (Value) columns of a table below, which lists the cookie information. Another red box highlights the 'Cookie' tab itself.

Nome	Valore
.AspNetCore.Identity.Application	CfDJ8FVKpLztR-Zmi_gki5KOtfIB51z...

Nome	Intestazioni	Anteprima	Risposta	Iniziatore	Tempo	Cookie
localhost						<input checked="" type="checkbox"/> Cookie di richiesta <input type="checkbox"/> mostra cookie delle richieste escluse dal filtro

Registrazione

Rendersi noti al sistema fornendo i propri dati.



Register

Create a new account.

Email

Password

Confirm password

Register

Login

Autenticarsi per ottenere il documento di autenticazione.



Man photo created by drobotdean - <https://www.freepik.com/free-photos-vectors/people>
https://it.wikipedia.org/wiki/File:Sportello_per_il_pubblico_in_Villa_Carrara.JPG

Log in
Use a local account to log in.

Email

Password

Remember me?

Log in

[Forgot your password?](#)

[Register as a new user](#)



I vantaggi dei cookie

- Memorizzare localmente un cookie è più sicuro che memorizzare una password, perché ha una durata limitata nel tempo ed è inaccessibile da codice JavaScript (se il cookie è "HttpOnly");
- L'applicazione ha più controllo: può reimpostare un cookie per invalidarlo prima della sua normale scadenza;
- Hanno un contenuto cifrato: possiamo inserirci dentro piccole informazioni che ci evitano di interrogare il database.

Autorizzazione

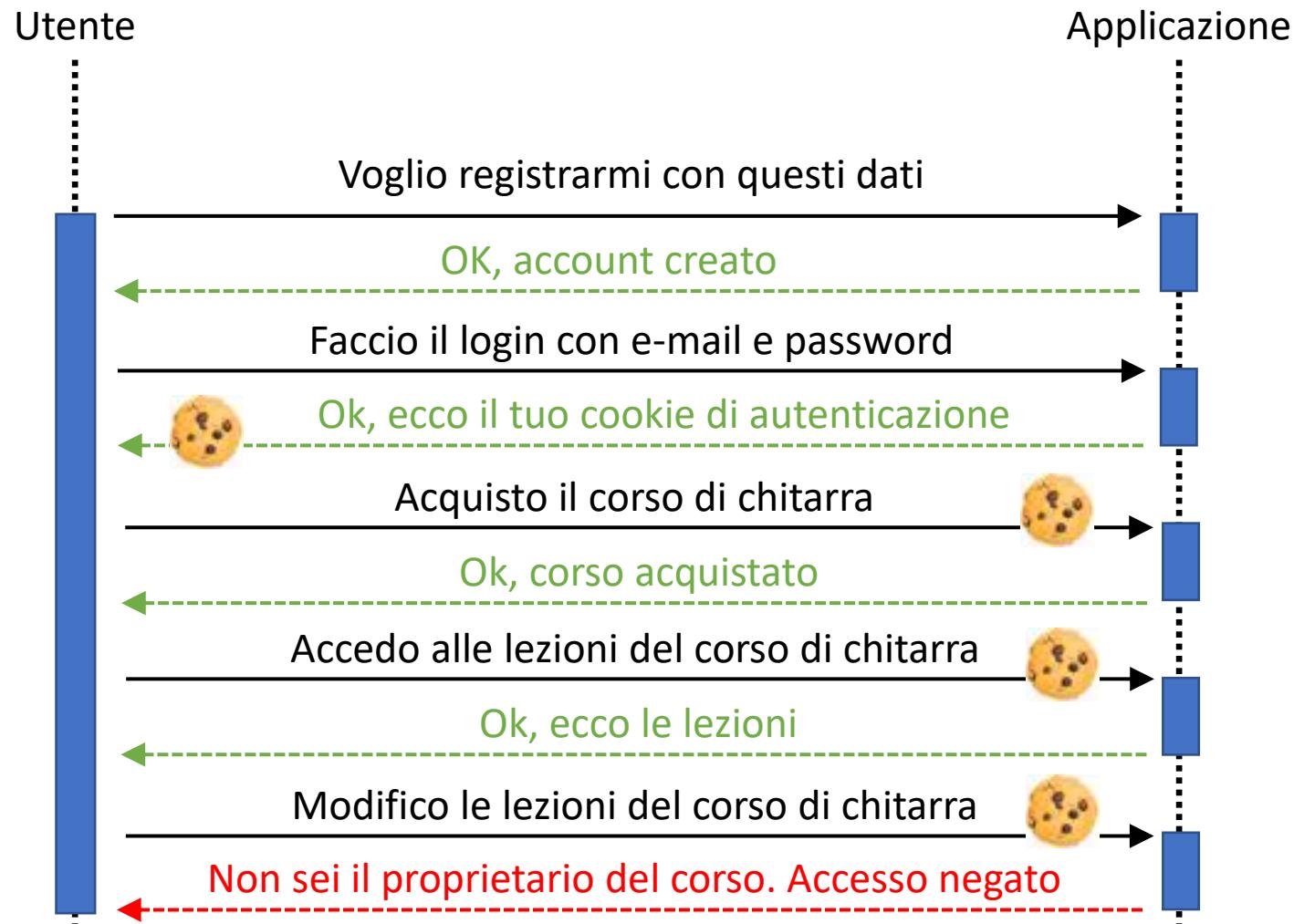
Determinare se l'utente, di cui ormai conosciamo l'identità, ha diritto a compiere un'operazione, in base a policy ben precise.



Foto di "Noi cambiamo" <https://www.noicambiamo.it/news/2016/09/11/nuova-proposta-del-m5s-per-la-selezione-degli-scrutatori-a-marino/seggio-elettorale/>

A screenshot of a web page titled "MyCourse". The main content area is blue and displays the text "Accesso negato" in red. Below it, a message in red says "Non sei autorizzato a modificare questo corso." At the bottom left, there is a link "← Torna all'elenco dei corsi".

Interazione-tipo dell'utente



ASP.NET Core Identity

È la soluzione consigliata per realizzare l'autenticazione.

- Si compone di tante parti: è estremamente modulare;
- Complicato come il meccanismo di un orologio;
 - Migliore di una soluzione personalizzata;
- Asseconda le nostre esigenze:
 - Che database vogliamo usare?
 - ADO.NET o Entity Framework Core?
 - Database di utenti locali o autenticazione da provider esterni?



ASP.NET Core Identity



Microsoft.AspNetCore.Identity.UI

È il pacchetto che contiene interfacce web integrabili nell'applicazione.

Register

Create a new account.

Email

Password

Confirm password

[Register](#)

Log in

Use a local account to log in.

Email

Password

Remember me?

[Log in](#)

[Forgot your password?](#)

[Register as a new user](#)

Microsoft.AspNetCore.Identity.EntityFrameworkCore

È il pacchetto che contiene l'IdentityDbContext, che contiene la logica di mapping per la persistenza degli utenti su tabelle del database.

```
public partial class MyCourseDbContext : DbContext IdentityDbContext
{
    //...
}
```

Sorgente dell'IdentityDbContext nel repository di ASP.NET Core

<https://github.com/dotnet/aspnetcore/blob/release/3.1/src/Identity.EntityFrameworkCore/src/IdentityDbContext.cs>

Microsoft.Extensions.Identity.Stores

È il pacchetto che contiene l'IdentityUser, una classe base che rappresenta un utente.

```
public class IdentityUser<TKey> {  
  
    public virtual TKey Id { get; set; }  
  
    [ProtectedPersonalData]  
    public virtual string Email { get; set; }  
  
    public virtual string PasswordHash { get; set; }  
  
    //...  
}
```

Sorgente dell'IdentityUser nel repository di ASP.NET Core

<https://github.com/dotnet/aspnetcore/blob/release/3.1/src/Identity/Extensions.Stores/src/IdentityUser.cs>

Microsoft.Extensions.Identity.Core

È il pacchetto che contiene i servizi applicativi, come lo UserManager che permette un accesso programmatico in lettura e scrittura agli utenti.

```
public class UserManager<TUser> {

    public virtual Task<TUser> FindByEmailAsync(string email);
    public virtual IQueryable<TUser> Users { get; }

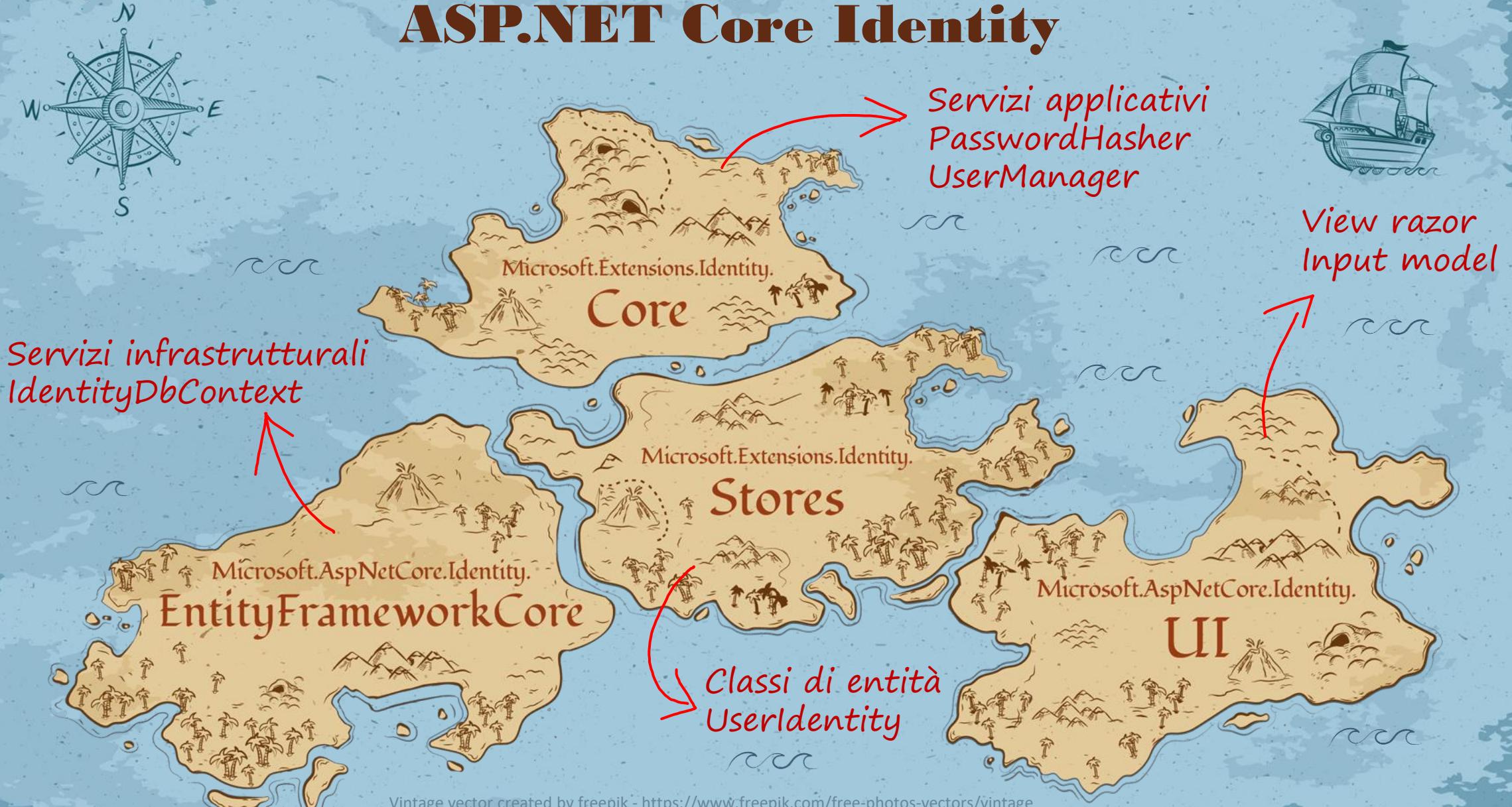
    public virtual Task<IdentityResult> CreateAsync(TUser user);
    public virtual Task<IdentityResult> UpdateAsync(TUser user);
    public virtual Task<IdentityResult> DeleteAsync(TUser user);

    public virtual Task<bool> CheckPasswordAsync(TUser user, string password);

}
```

<https://github.com/dotnet/aspnetcore/blob/release/3.1/src/Identity/Extensions.Core/src/UserManager.cs>

ASP.NET Core Identity



Integrare Identity in un nuovo progetto

Con Sqlite

```
dotnet new mvc --auth Individual
```

Con Sql Server

```
dotnet new mvc --auth Individual --use-local-db
```

Per altre opzioni...

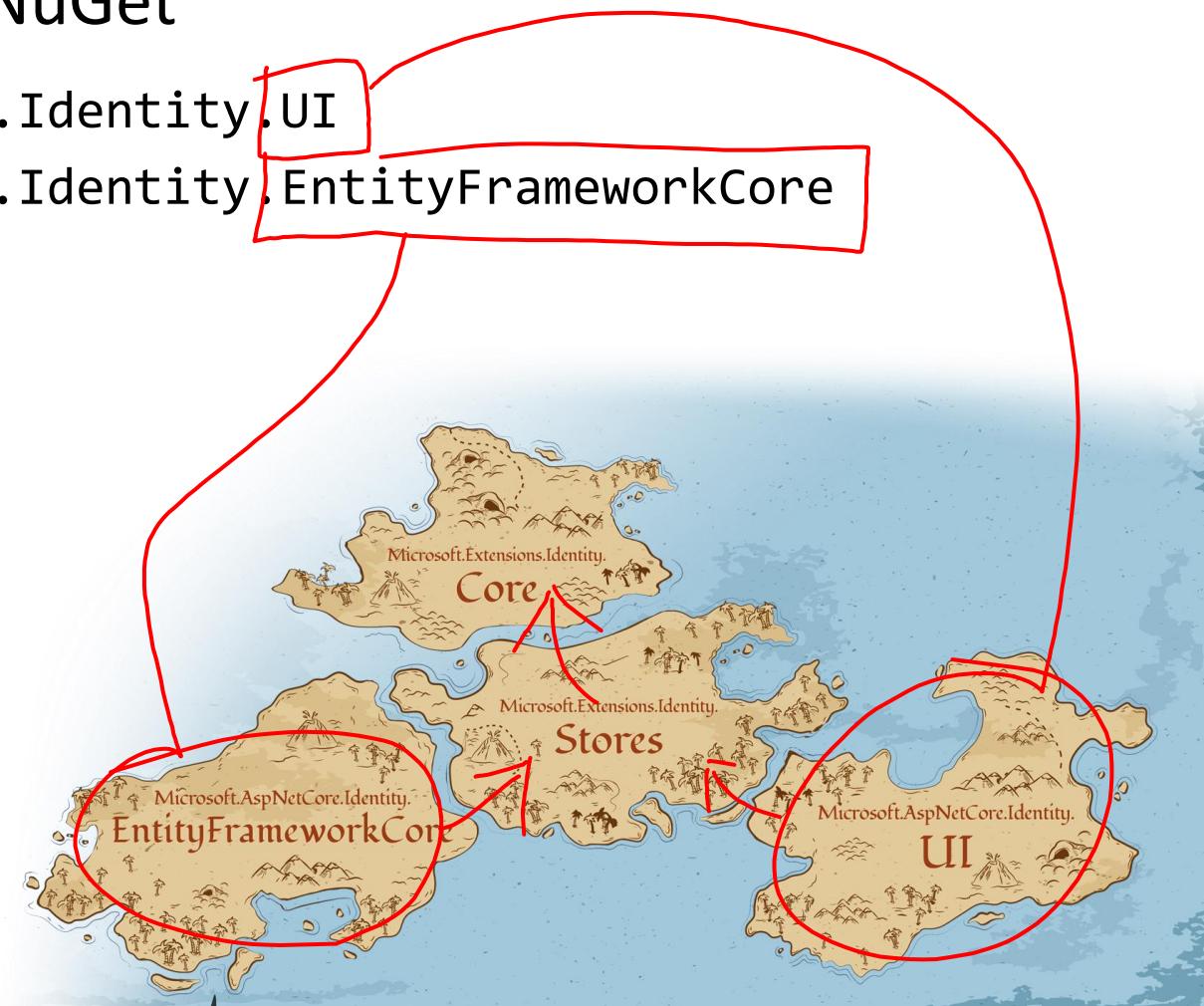
```
dotnet new mvc --help
```

Integrare Identity in un progetto esistente (1/7)

Aggiungere il riferimento ai pacchetti NuGet

```
dotnet add package Microsoft.AspNetCore.Identity.UI
```

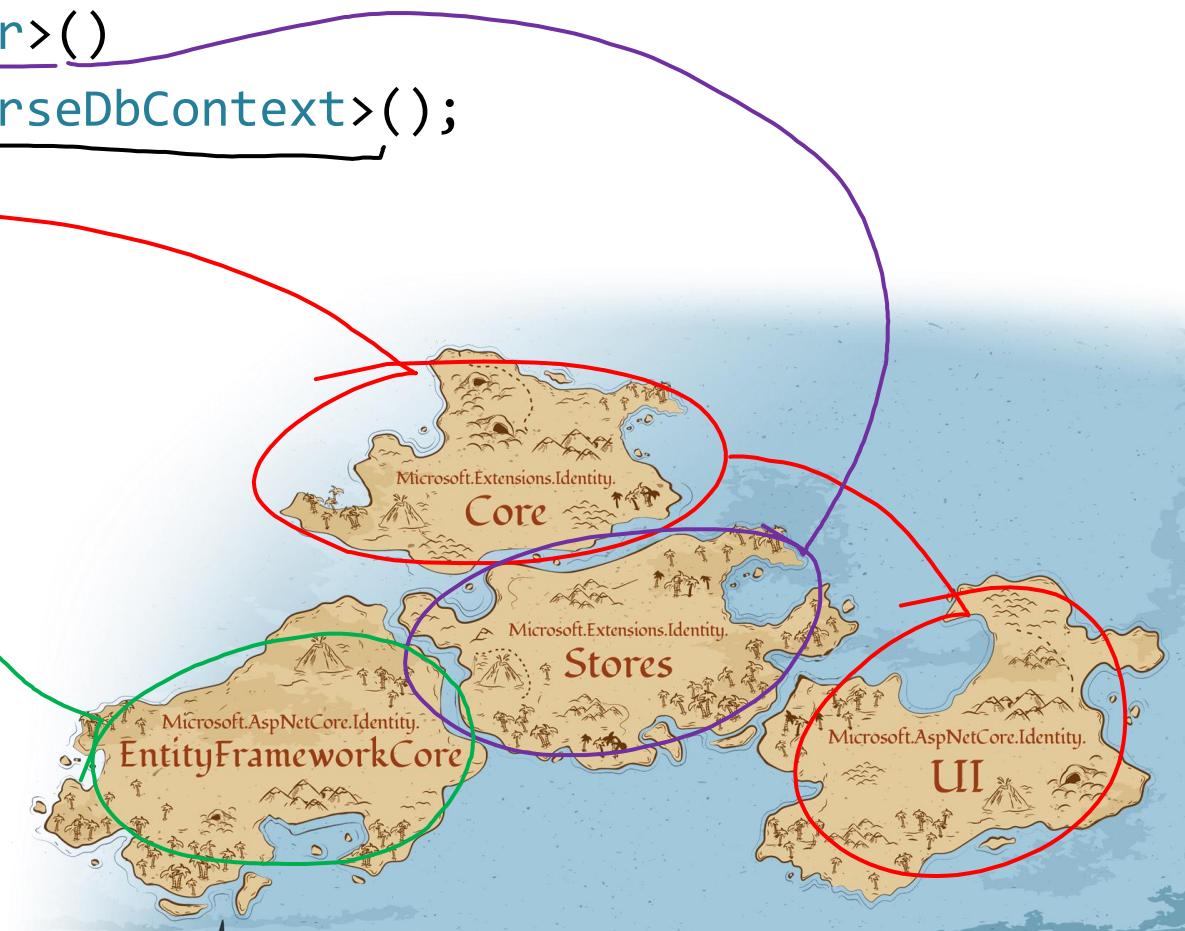
```
dotnet add package Microsoft.AspNetCore.Identity.EntityFrameworkCore
```



Integrare Identity in un progetto esistente (2/7)

Configurare i servizi nella classe Startup, metodo ConfigureServices

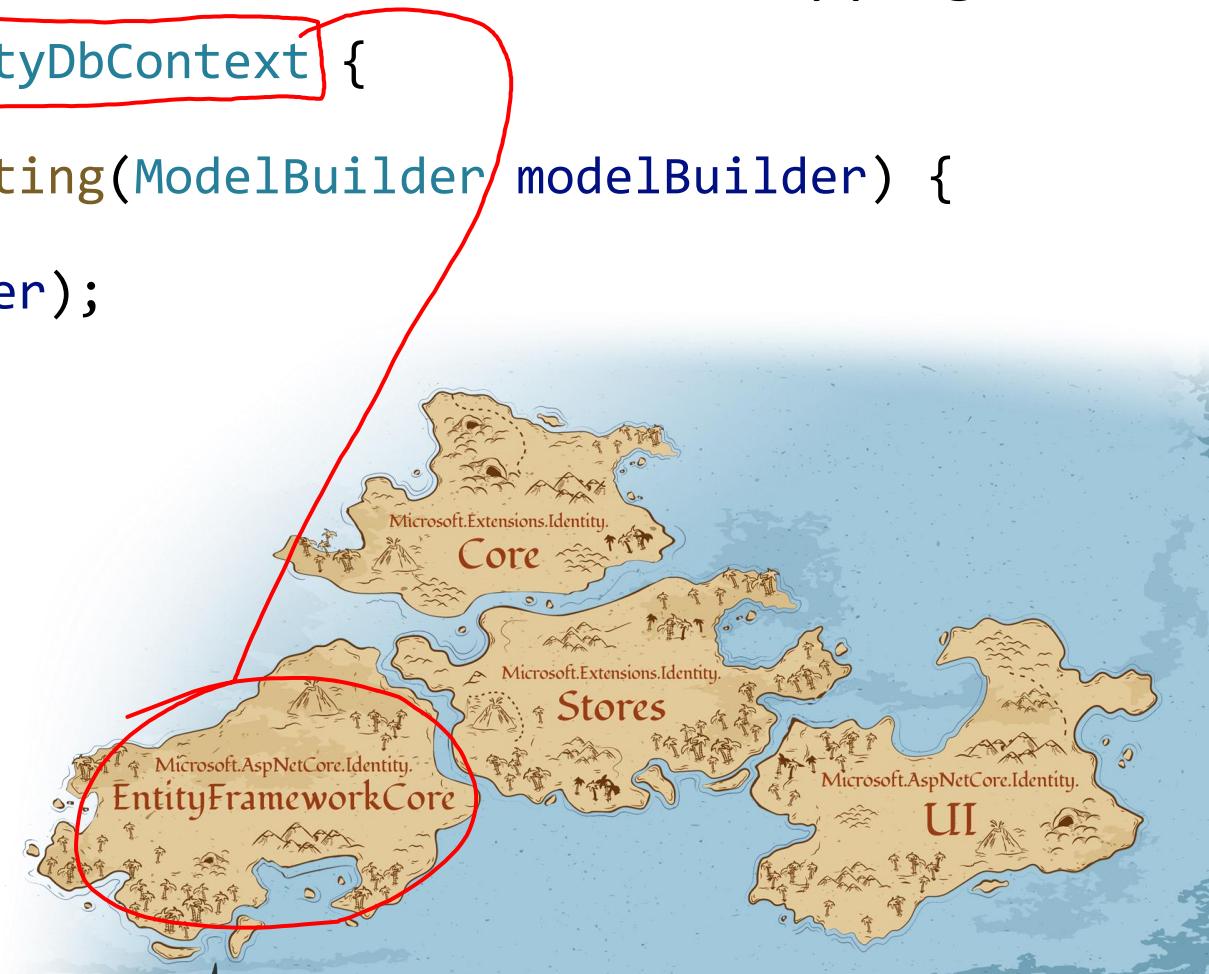
```
services.AddDefaultIdentity<IdentityUser>()  
    .AddEntityFrameworkStores<MyCourseDbContext>();
```



Integrare Identity in un progetto esistente (3/7)

Far derivare il DbContext da IdentityDbContext ed ereditare il mapping

```
public class MyCourseDbContext : IdentityDbContext {  
    protected override void OnModelCreating(ModelBuilder modelBuilder) {  
        base.OnModelCreating(modelBuilder);  
        //Qui mapping esistente  
    }  
}
```



Integrare Identity in un progetto esistente (4/7)

Aggiungere e applicare una migration

```
dotnet ef migrations add IdentityModel
```

```
dotnet ef database update
```



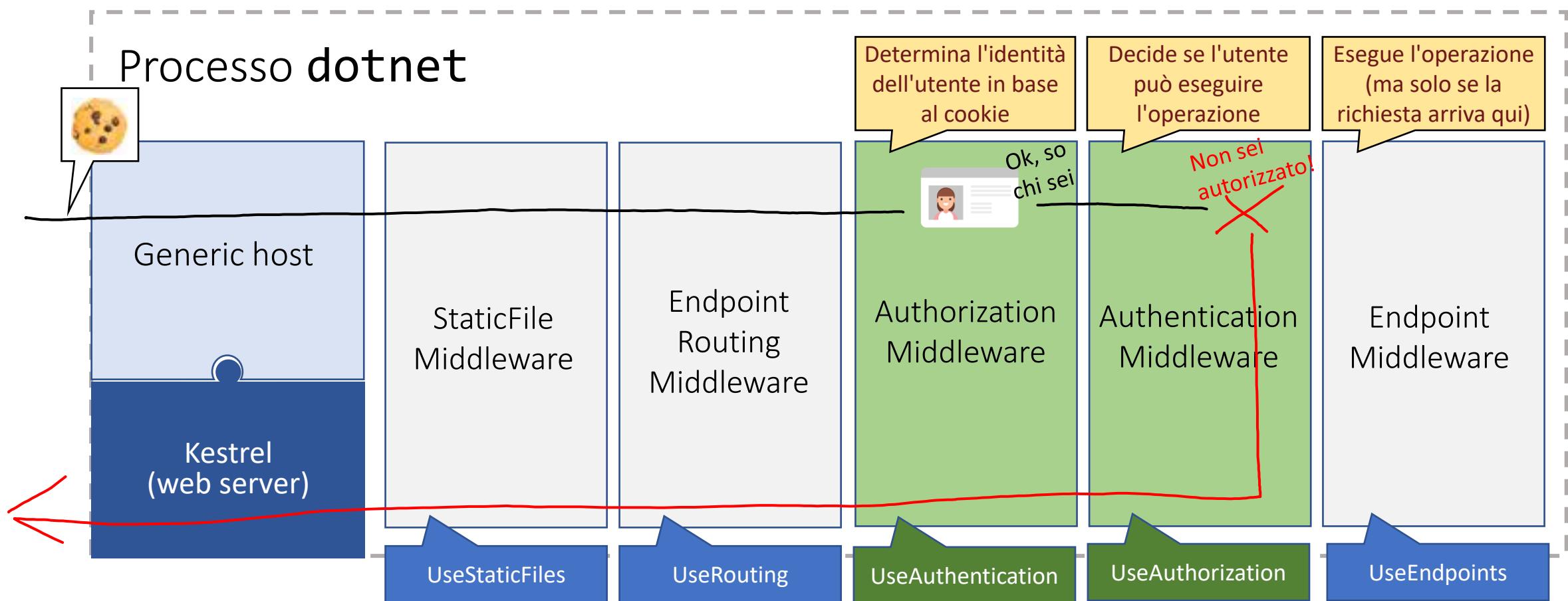
Integrare Identity in un progetto esistente (5/7)

Usare i middleware nella classe Startup, metodo Configure

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



Middleware di autenticazione e autorizzazione



Integrare Identity in un progetto esistente (6/7)

Abilitare Razor Pages

Classe Startup, metodo ConfigureServices

```
services.AddRazorPages();
```

Classe Startup, metodo Configure

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

Integrare Identity in un progetto esistente (7/7)

Aggiungere la partial view `_LoginPartial.cshtml`

`/Views/Shared/_LoginPartial.cshtml`

oppure

`/Pages/_LoginPartial.cshtml`

Impostare la view di layout per Identity

 Areas

 Identity

 Pages

 _ViewStart.cshtml

```
@{  
    Layout = "/Views/Shared/_Layout.cshtml";  
}
```

Criteri di complessità della password

ASP.NET Core Identity li mette a disposizione come opzioni.

IdentityOptions

```
services.AddDefaultIdentity<IdentityUser>(options => {
```

 RequireDigit	Richiede numeri?
 RequiredLength	Lunghezza minima?
 RequiredUniqueChars	Quanti caratteri unici?
 RequireLowercase	Richiede minuscole?
 RequireNonAlphanumeric	Richiede simboli?
 RequireUppercase	Richiede maiuscole?

```
)
```

```
...
```

Criteri di complessità della password

Da soli, non ci garantiscono che l'utente digiterà una password sicura.

```
services.AddDefaultIdentity<IdentityUser>(options => {
    options.Password.RequireDigit = true;
    options.Password.RequiredLength = 8;
    options.Password.RequireUppercase = true;
    options.Password.RequireLowercase = true;
    options.Password.RequireNonAlphanumeric = true;
    options.Password.RequiredUniqueChars = 4;
})  
...
```



Ci interessa se un account viene rubato?

Sì, perché:

- Se un account viene rubato, dobbiamo per forza di cose fornire un'assistenza puntuale all'utente (e ciò comporta tempo);
- Può sfociare in un danno d'immagine per il committente;
- In base al tipo di applicazione che stiamo sviluppando e ai termini di utilizzo, potrebbe essere necessario riparare il danno economico.

Alcuni utenti scelgono password facili da indovinare

President Trump's Twitter accessed by security expert who guessed password 'maga2020!'

Zack Whittaker @zackwhittaker / 12:37 AM GMT+2 • October 23, 2020

 Comment



Alcuni utenti scelgono password facili da indovinare

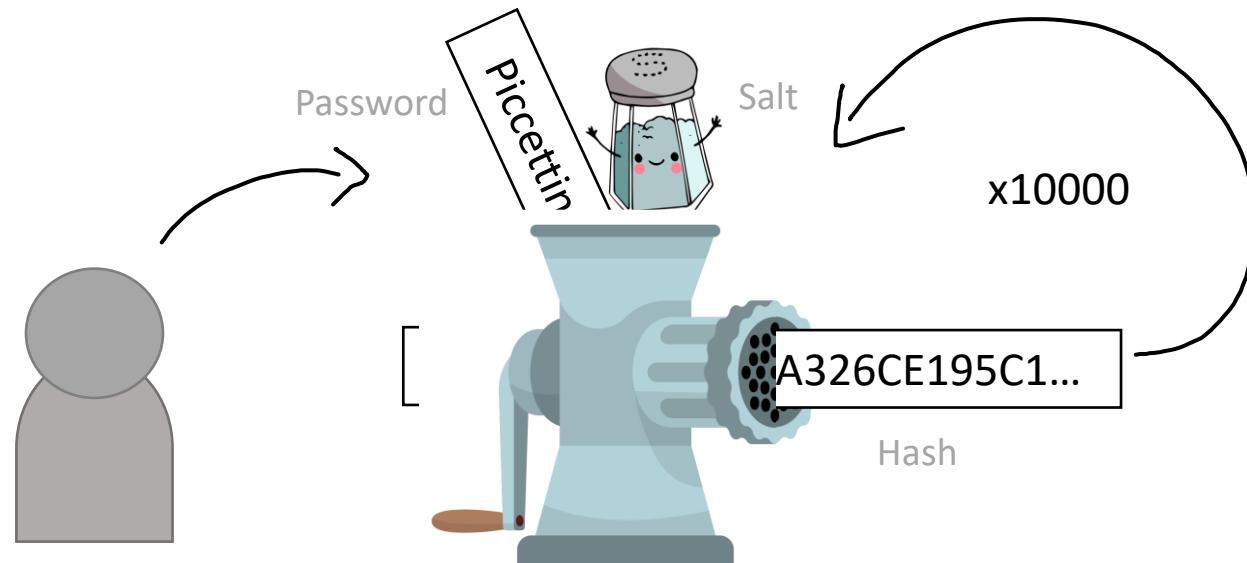
È responsabilità dell'utente scegliere una password sicura ma ci sono comunque alcune cose che possiamo fare noi come sviluppatori.

Microsoft ha pubblicato dei consigli per mitigare questo fenomeno.

https://www.microsoft.com/en-us/research/wp-content/uploads/2016/06/Microsoft_Password_Guidance-1.pdf

PasswordHasher<TUser>

Usa un algoritmo di derivazione della password chiamato PBKDF2 che applica HMAC-SHA256 per 10.000 volte. Scoraggia attacchi brute-force.



<https://en.wikipedia.org/wiki/PBKDF2>

Immagine vettoriale di macrovector - <https://www.freepik.com/free-photos-vectors/food>



Validare la password con logica personalizzata

```
public class CommonPasswordValidator<TUser> :  
    IPasswordValidator<TUser> where TUser : class  
{  
    public async Task<IdentityResult> ValidateAsync(  
        UserManager<TUser> manager, TUser user, string password)  
    {  
        if (password == "password") {  
            var error = new IdentityError { Description = "Troppo comune" };  
            return IdentityResult.Failed(error);  
        }  
        return IdentityResult.Success;  
    }  
}
```



Validare la password con logica personalizzata

```
services.AddDefaultIdentity<IdentityUser>(options => {
    //...
})
    .AddPasswordValidator<CommonPasswordValidator<IdentityUser>>()
    .AddEntityFrameworkStores<MyCourseDbContext>();
```



User

- È una proprietà disponibile sia nelle action che nelle view Razor;
- È sempre accessibile, anche quando la richiesta è anonima (cioè se l'utente non ha fatto il login).



```
User.Identity.Name      = "email@dominio.com"  
User.Claims              = "Nome" "Mario"  
                           "Età"  "18"
```

User

- È una proprietà disponibile sia nelle action che nelle view Razor;
- È sempre accessibile, anche quando la richiesta è anonima (cioè se l'utente non ha fatto il login).



`User.Identity.Name` = null

`User.Claims` = vuoto

SignInManager<TUser>

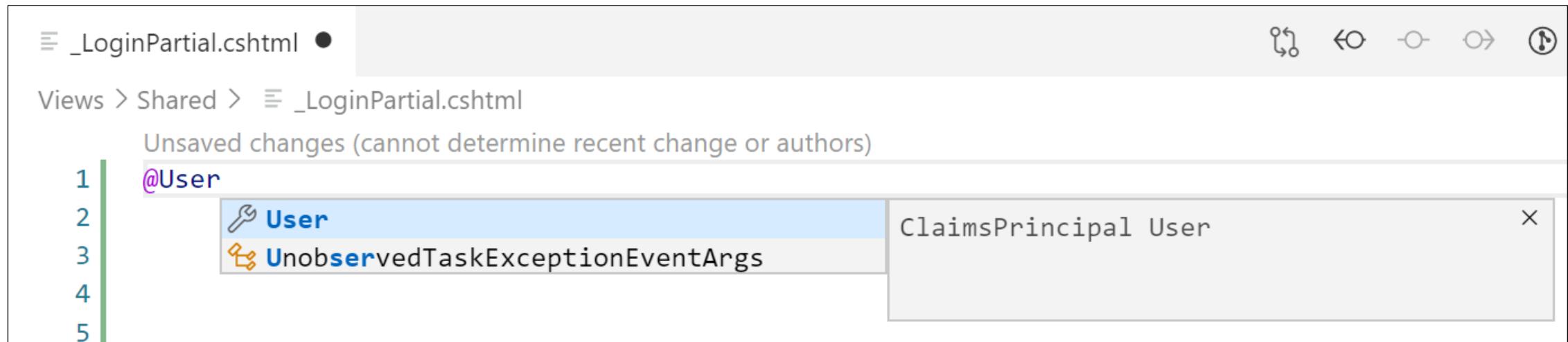
È il componente che materialmente esegue il login e logout.

```
@inject SignInManager<IdentityUser> SignInManager
@if (SignInManager.IsSignedIn(User))
{
    //L'utente ha fatto il login
}
else
{
    //L'utente non ha fatto il login
}
```



User

Disponibile nelle view Razor...



The screenshot shows a code editor window for a file named '_LoginPartial.cshtml' located in the 'Views > Shared' folder. The file contains the following code:

```
1 @User
2
3
```

A tooltip is displayed over the '@User' placeholder, showing the type 'ClaimsPrincipal User'. The tooltip includes a close button 'X'.

User

...nei controller...

The screenshot shows a code editor window for a file named 'CoursesController.cs'. The code is as follows:

```
11 public class CoursesController : Controller
12 {
13     public async Task<IActionResult> Index(CourseListInputModel input)
14     {
15         User
```

A tooltip is displayed over the word 'User' at line 15. The tooltip contains the following information:

ClaimsPrincipal User

Gets the System.Security.Claims.ClaimsPrincipal for user associated with the executing action.

User

...e in qualsiasi altro componente grazie al servizio `IHttpContextAccessor`



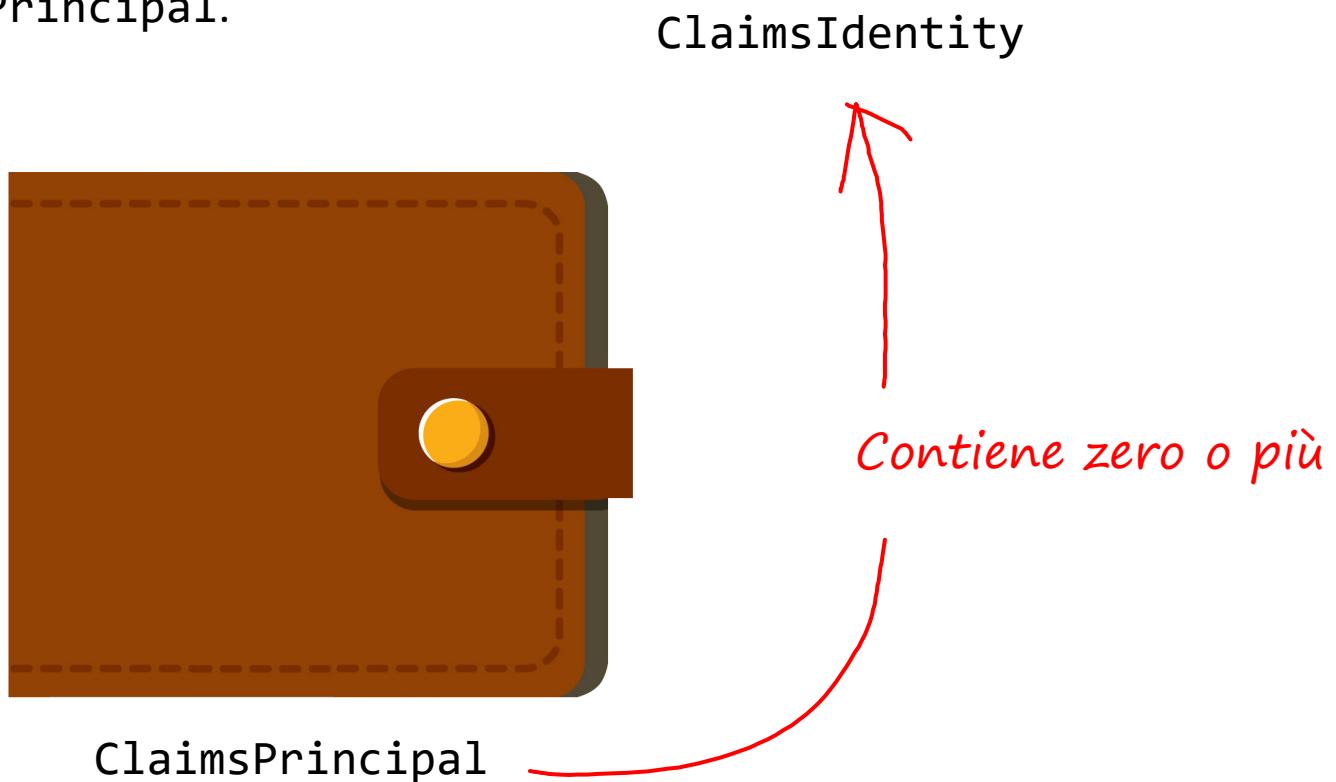
```
C# EfCoreCourseService.cs •
Models > Services > Application > Courses > C# EfCoreCourseService.cs
1 reference | Unsaved changes (cannot determine recent change or authors)
21     public class EfCoreCourseService : ICourseService
22     {
23         0 references
24         public EfCoreCourseService(IHttpContextAccessor httpContextAccessor)
25         {
26             httpContextAccessor.HttpContext.User
27         }
28     }
29 
```

ClaimsPrincipal User

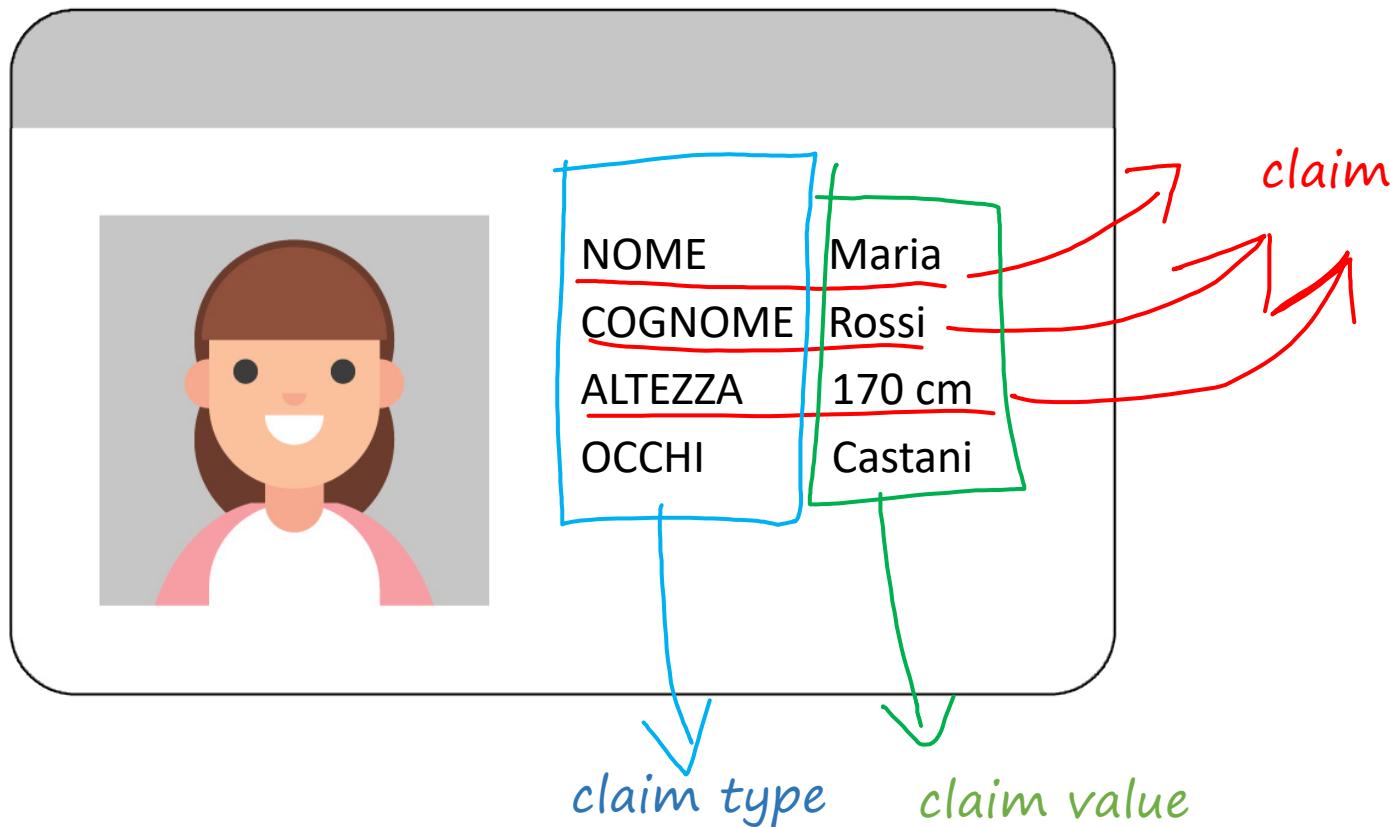
Gets or sets the user for this request.

User è un contenitore di identità

E' un oggetto di tipo `ClaimsPrincipal`.



Una ClaimsIdentity contiene dei *claim*



SignInManager<TUser>

Ci dice se un utente è autenticato (con Identity)

```
@inject SignInManager<IdentityUser> signInManager  
  
@if(signInManager.IsSignedIn(User))  
{  
    <!-- Autenticato -->  
}  
else  
{  
    <!-- Non autenticato -->  
}
```

Nota: serve uno using nel file /Views/_ViewImports.cshtml
@using Microsoft.AspNetCore.Identity



Scaffolding della UI di Identity

Installiamo il tool globale

```
dotnet tool install -g dotnet-aspnet-codegenerator
```

Installiamo il pacchetto

```
dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design
```

Scaffolding della UI di Identity

 Areas

 Identity

 Pages

 Account

 Register.cshtml

 Login.cshtml

 Logout.cshtml

Scaffolding della UI di Identity

Elencare tutte le Razor Page disponibili

```
dotnet aspnet-codegenerator identity --listFiles
```

Fare lo scaffolding di una o più Razor Page

```
dotnet aspnet-codegenerator identity  
--files "Account.Register;Account.Manage.Index"  
--dbContext MyCourse.Models.Services.Infrastructure.MyCourseDbContext  
--useSqlLite
```

Uno o più file, separati da punto e virgola e inclusi nei doppi apici

Scaffolding di altri elementi MVC / Razor Pages

Generare un controller

```
dotnet aspnet-codegenerator controller --help
```

Generare una view

```
dotnet aspnet-codegenerator view --help
```

Generare una razor page

```
dotnet aspnet-codegenerator razorpage --help
```

Aree

Catalogo dei corsi

Titolo ▾



Web marketing facile
di Adelinda Greco



Spagnolo per principianti
di Antonella Zetticci



Sommelier in 15 giorni
di Angelo Endrizzi



Siti responsive con HTML5
di Penelope Lucchesi



Pilota il tuo drone
di Davide Fosca



Pianoforte: livello intermedio
di Rodrigo Bellucci



Modellare la ceramica
di Severino Padovano



Le basi del Cake design
di Elsa De Luca

Manage your account

Change your account settings

Profile

Email

Password

Two-factor
authentication

Personal data



Admin

Calendar

2

Gallery

Mailbox

Pages

Extras

MISCELLANEOUS

Documentation

Archivio blog

2020



2019



gennaio

febbraio

marzo

aprile

maggio

Catalogo corsi

/

Gestione utenti

/Identity

Area Riservata

/Admin

Blog

/Blog

Area /Identity

/Account/Register /Account/RegisterConfirmation	Registrazione nuovo profilo utente
/Account/Login	Autenticazione con email e password
/Account/LoginWith2fa /Account/LoginWithRecoveryCode	Autenticazione a due fattori
/Account/ExternalLogin	Autenticazione social
/Account/ForgotPassword /Account/ForgotPasswordConfirmation /Account/ResetPassword /Account/ResetPasswordConfirmation	Recupero password dimenticata
/Account/Lockout	Blocco del profilo utente
/Account/ConfirmEmail /Account/ConfirmEmailChange	Conferma dell'indirizzo e-mail
/Account/Logout	Disconnessione

/Account/Manage/Index	Pagina principale della gestione account
/Account/Manage/Email	Cambio e-mail
/Account/Manage/ChangePassword /Account/Manage/SetPassword	Cambio password
/Account/Manage/ExternalLogins	Gestione autenticazione social
/Account/Manage/TwoFactorAuthentication /Account/Manage/EnableAuthenticator /Account/Manage>ShowRecoveryCodes /Account/Manage/GenerateRecoveryCodes /Account/Manage/ResetAuthenticator /Account/Manage/Disable2fa	Gestione autenticazione a due fattori
/Account/Manage/PersonalData /Account/Manage/DeletePersonalData /Account/Manage/DownloadPersonalData	Download ed eliminazione dati personali

Riepilogo dei percorsi alla UI di Identity

REGISTRAZIONE

```
<a asp-page="/Account/Register" asp-area="Identity">Registrati</a>
```

LOGIN

```
<a asp-page="/Account/Login" asp-area="Identity">Accedi</a>
```

MODIFICA DEL PROFILO

```
<a asp-page="/Account/Manage/Index" asp-area="Identity">Ciao @User.Identity.Name</a>
```

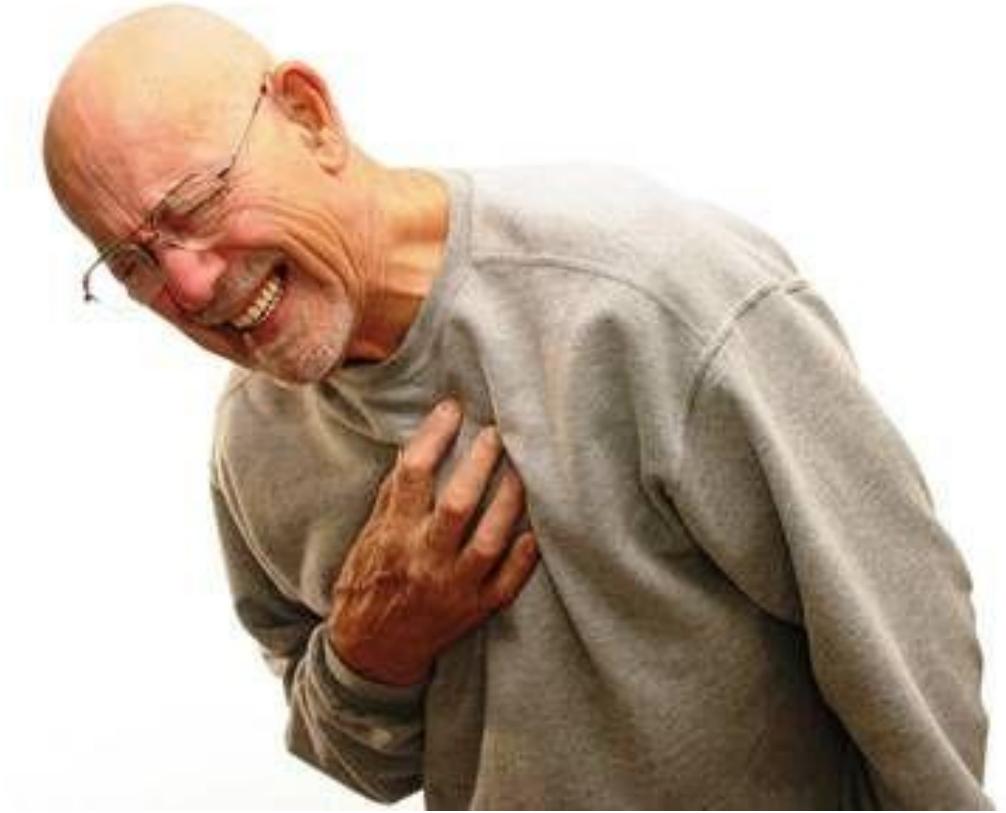
LOGOUT

```
<form method="post" asp-page="/Account/Logout" asp-area="Identity">
    <button type="submit">Esci</button>
</form>
```

Localizzazione della UI di Identity

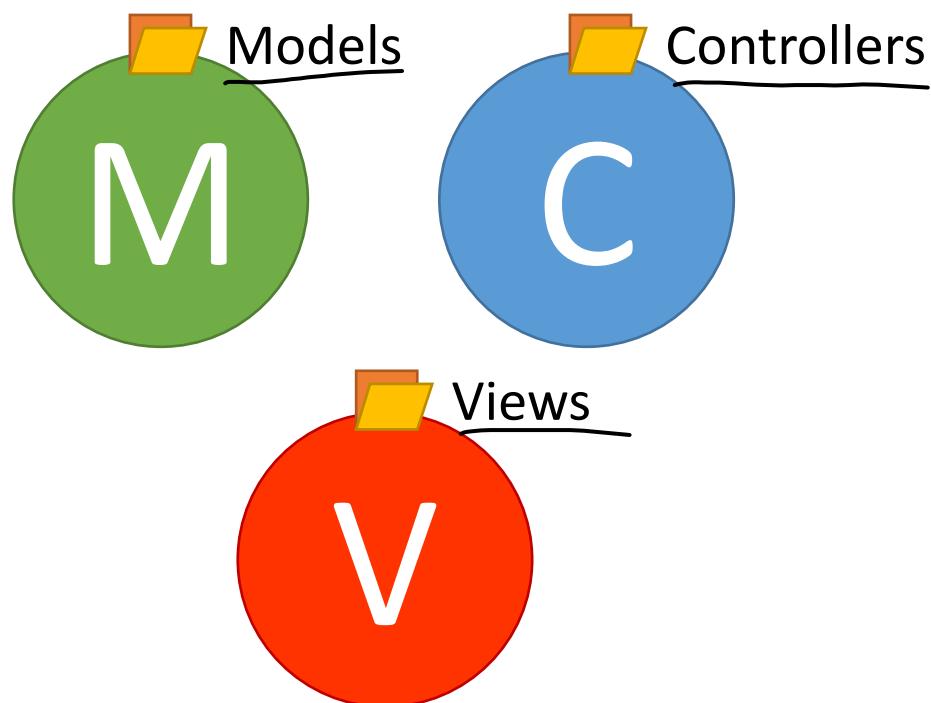
Va fatto a mano modificando tutte le razor page

<https://github.com/dotnet/AspNetCore.Docs/issues/11038>

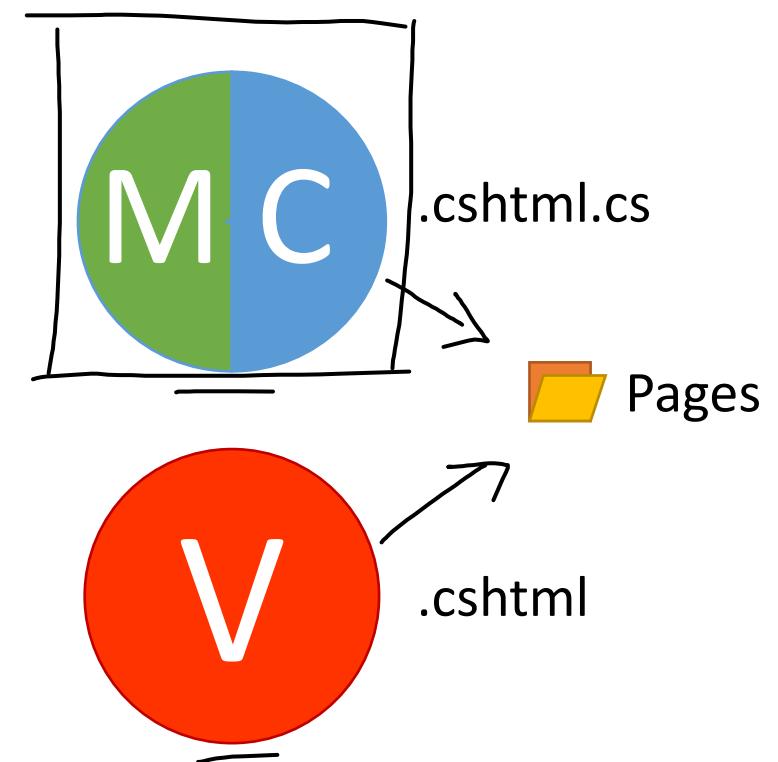


MVC e Razor Pages

Organizzazione dei file in MVC



Organizzazione dei file in Razor Pages



Razor Pages: la direttiva @page

Rende la Razor Page raggiungibile da un URL, in base alla directory in cui è.

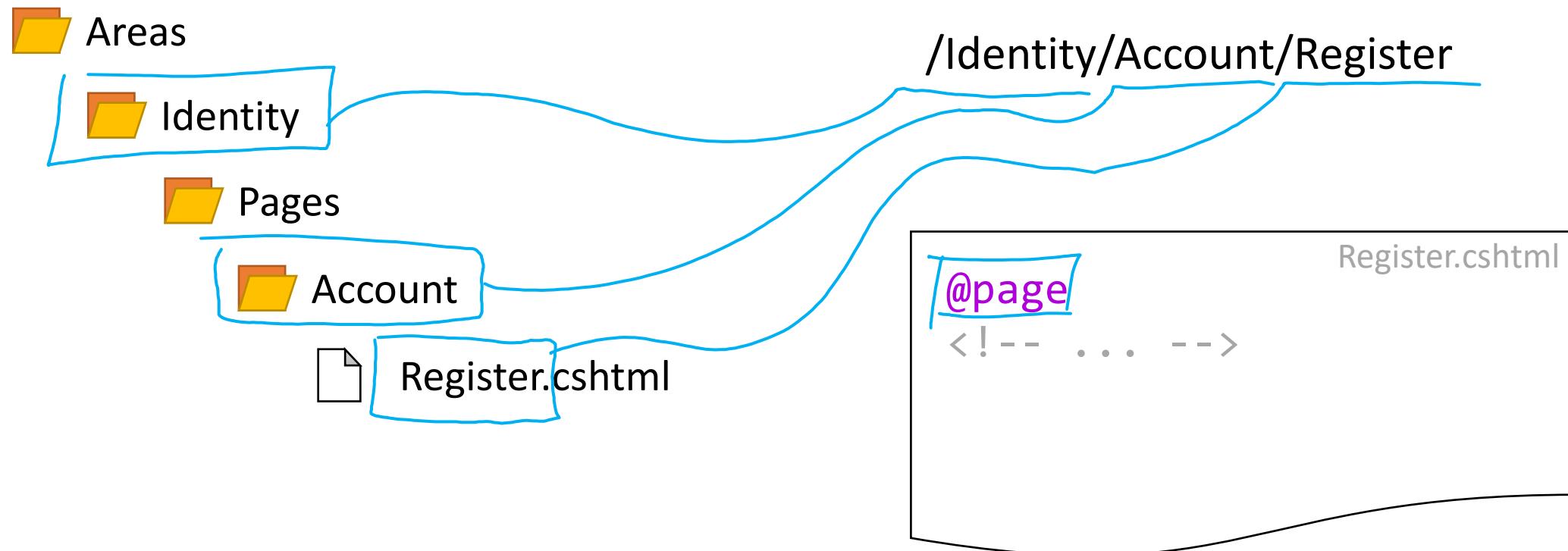
```
@page
@model RegisterModel
@{
    ViewData["Title"] = "Registrati";
}

<h1>@ViewData["Title"]</h1>

<div class="row">
    <!-- ... -->
</div>
```

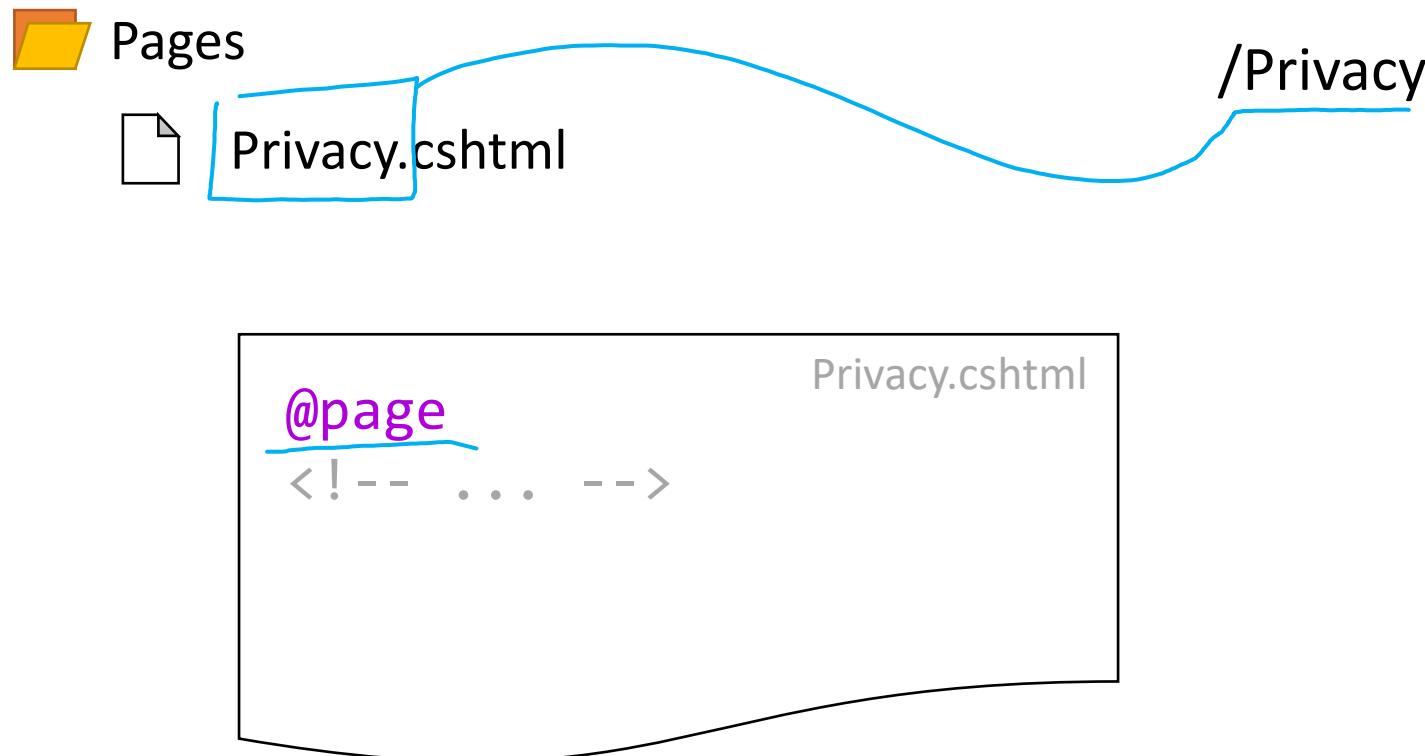
Razor Pages: la direttiva @page

Rende la Razor Page raggiungibile via web a un certo indirizzo.



Razor Pages: la direttiva @page

Rende la Razor Page raggiungibile via web a un certo indirizzo.



Page model di una Razor Page

Si trova nella stessa directory della view



Areas



Identity



Pages



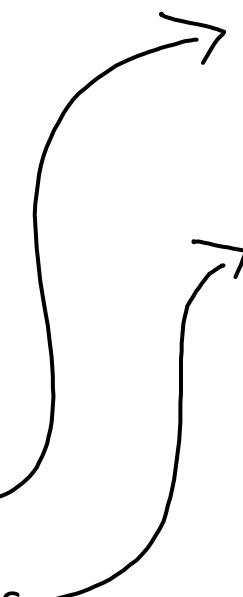
Account



Register.cshtml



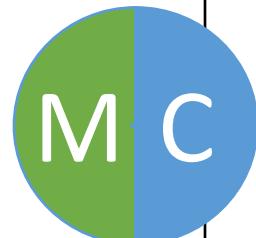
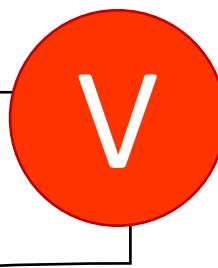
Register.cshtml.cs



```
@page  
@model RegisterModel
```

```
public class RegisterModel : PageModel {
```

```
}
```



Razor Page

- È composta di due file: `view` (.cshtml) e `page model` (.cshtml.cs);
- I file si trovano nella cartella `Pages` (o `Areas/NomeArea/Pages`);
- Nella view si trova la direttiva `@page`
- Il page model è una classe che deriva dal tipo base `PageModel`
- Il page model contiene delle proprietà e ponendo l'attributo `[BindProperty]`, il model binder le valorizzerà con l'input dell'utente
- Il page model contiene i metodi `OnGetAsync` e `OnPostAsync` per gestire le richieste di tipo GET e POST
- Una Razor Page supporta `model binding` e `dependency injection`

Estendere IdentityUser

Ci permette di creare una nostra classe che rappresenta l'utente (es. ApplicationUser), in modo che possiamo aggiungere altre proprietà, oltre a quelle ereditate dalla classe base IdentityUser.

```
public class ApplicationUser : IdentityUser {  
  
    public virtual string FullName { get; set; }  
  
    //...  
}
```



Accedere al profilo con lo UserManager

Riceviamo lo **UserManager** grazie alla **dependency injection**

```
@inject UserManager< ApplicationUser > userManager
```

Lo **UserManager** interroga il db e restituisce l'**ApplicationUser**

```
@{  
    ApplicationUser applicationUser = await userManager.GetUserAsync(User);  
}
```

Visualizziamo i valori

```
<div>@applicationUser.FullName</div>
```



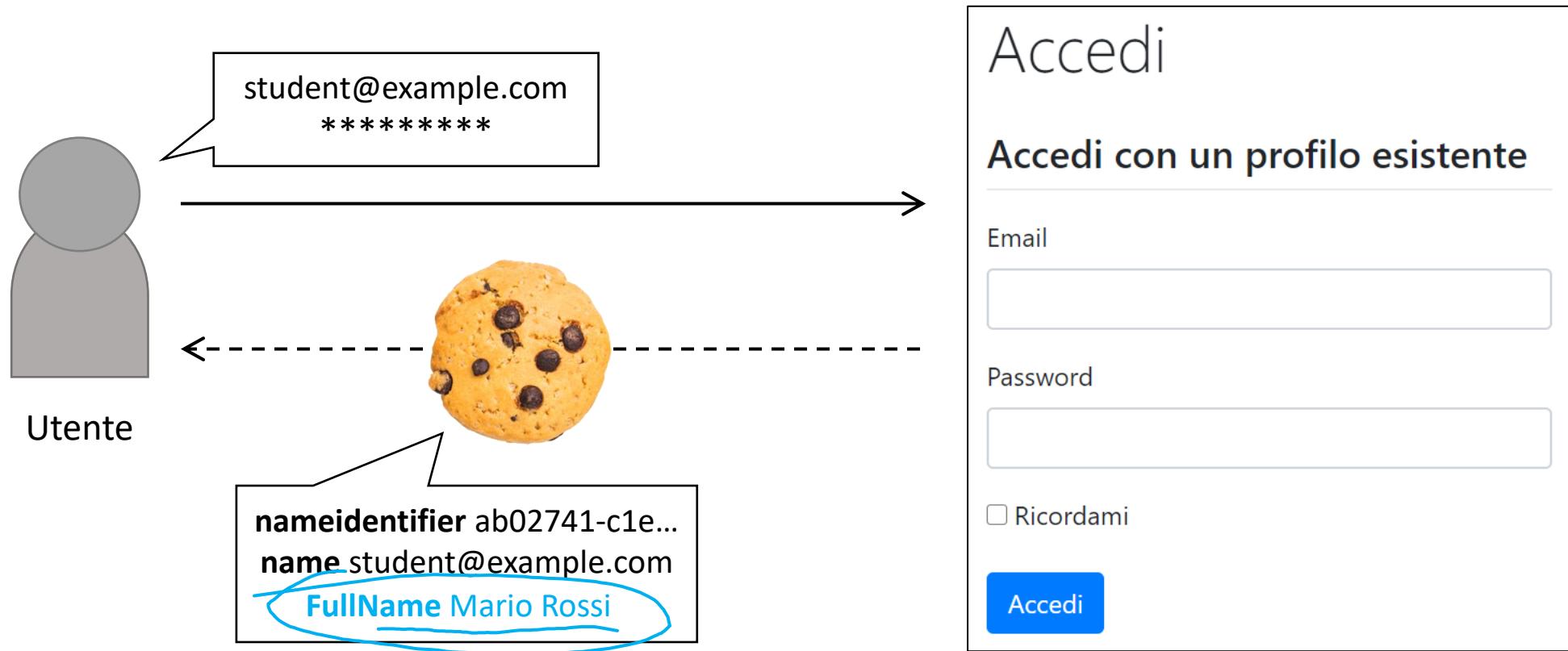
Accedere al profilo con lo UserManager

```
 ApplicationUser applicationUser = await userManager.GetUserAsync(User);  
  
 ApplicationUser applicationUser = await userManager.FindByIdAsync("ab382c4...");  
 ApplicationUser applicationUser = await userManager.FindByEmailAsync("student@...");  
 ApplicationUser applicationUser = await userManager.FindByNameAsync("Joshua");  
  
 List<ApplicationUser> applicationUsers = await userManager.Users  
     .Where(user => user.FullName.StartsWith("Bob"))  
     .ToListAsync();
```

Persistere il profilo con lo UserManager

```
 ApplicationUser applicationUser = await userManager.GetUserAsync(User);
applicationUser.FullName = "Luigi Verdi";
IdentityResult result = await userManager.UpdateAsync(applicationUser);
if (result.Succeeded)
{
    // Mostra conferma all'utente
}
else
{
    // Mostra result.Errors
}
```

Aggiungere un claim personalizzato al cookie



I cookie hanno un limite di circa 4KB

The screenshot shows a web browser window for 'MyCourse' with a blue header bar. The main content area displays the text 'Benvenuto su MyCourse!' and some descriptive text about learning online and choosing from a catalog. A blue button at the bottom left says 'Sfoglia il catalogo dei corsi >'. To the right of the browser is the developer tools interface.

The developer tools have several tabs at the top: Elementi, Console, Origini, Rete (selected), Prestazioni, Memoria, Applicazione, and others. There are also various icons and checkboxes for network monitoring.

The Network tab shows a timeline with a single request highlighted in blue, spanning from 50 ms to 500 ms. Below the timeline is a table titled 'Nome' (Name) which lists various files like bootstrap.min.css, style.css, etc.

On the right side of the developer tools, there are two tables under the 'Cookie' tab:

- Cookie di richiesta**: Shows a list of request cookies. Several cookies are highlighted in yellow:
 - .AspNetCore.Antiforgery.Hnu_PNmjemE
 - AspNetCore.Identity.Application
 - AspNetCore.Identity.ApplicationC1
 - AspNetCore.Identity.ApplicationC2
 - AspNetCore.Identity.ApplicationC3
 - AspNetCore.Identity.ApplicationC4These highlighted cookies all have a value length of 4008, which is likely the maximum size of a cookie (4KB). Other cookies listed include .AspNetCore.Mvc.CookieTempDataPro... with a value length of 110.
- Cookie di risposta**: Shows a list of response cookies, including .AspNetCore.Mvc.CookieTempDataPro... with a value length of 110.

UserClaimsPrincipalFactory<TUser>

```
public class CustomClaimsPrincipalFactory : UserClaimsPrincipalFactory<ApplicationUser>
{
    public CustomClaimsPrincipalFactory(
        UserManager<ApplicationUser> userManager, IOptions<IdentityOptions> optionsAccessor)
        : base(userManager, optionsAccessor)
    {
    }

    protected async override Task<ClaimsIdentity> GenerateClaimsAsync(ApplicationUser user)
    {
        ClaimsIdentity identity = await base.GenerateClaimsAsync(user);
        identity.AddClaim(new Claim("FullName", user.FullName));
        return identity;
    }
}
```

Qui aggiungiamo il claim personalizzato al cookie di autenticazione. Ce lo ritroveremo nell'oggetto User ad ogni successiva richiesta dell'utente. Ci eviterà una chiamata al database.

Registrare la claims principal factory personalizzata

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

```
services
    .AddDefaultIdentity< ApplicationUser >(options => {
        // options
    })
    .AddClaimsPrincipalFactory< CustomClaimsPrincipalFactory >();
```

Accedere ai claim dell'oggetto User

Usare il metodo `FindFirst` per recuperare il primo claim che reca un certo claim type

```
<div>@User.FindFirst("FullName").Value</div>
<div>@User.FindFirst(ClaimTypes.NameIdentifier).Value</div>
```

using System.Security.Claims

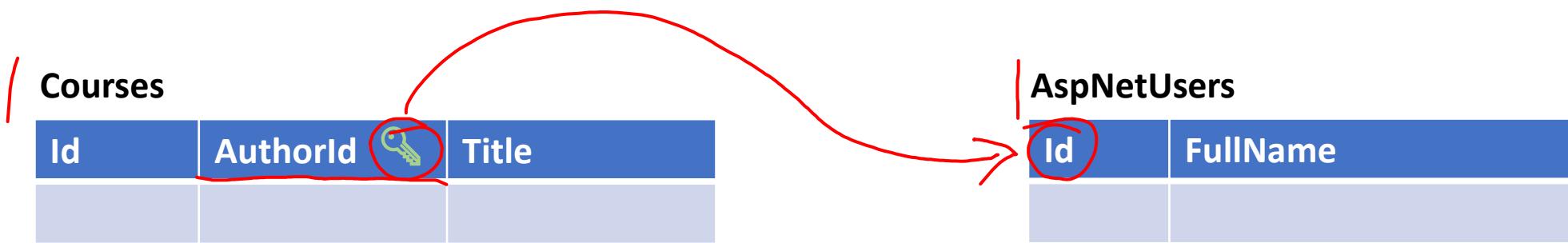
In una `ClaimsIdentity` possono coesistere più claim con lo stesso claim type.

In tal caso, usare `FindAll` per estrarli tutti:

```
@foreach (var claim in User.FindAll("FullName"))
{
    <li>@claim.Value</li>
}
```

Relazionare corsi e autori

Relazione uno-a-molti



Relazionare corsi e autori

Relazione uno-a-molti

```
public class Course {
    public string AuthorId { get; set; }
    public virtual ApplicationUser AuthorUser { get; set; }
}

public class ApplicationUser : IdentityUser {
    public virtual ICollection<Course> AuthoredCourses { get; set; }
}

modelBuilder.Entity<Course>(entity => {
    entity.HasOne(course => course.AuthorUser)
        .WithMany(user => user.AuthoredCourses)
        .HasForeignKey(course => course.AuthorId);
}
```

Sopprimere la transazione nelle migration

I comandi SQL vengono eseguiti nel contesto di una transazione;
... ma **PRAGMA foreign_keys = 0;** non funziona in una transazione;

Allora facciamo in modo che la transazione venga soppressa.

```
protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.Sql(@"PRAGMA foreign_keys = 0;
    ...QUI COMANDI SQL...
    PRAGMA foreign_keys = 1;",
    suppressTransaction: true);
}
```

Conferma di registrazione



Registrati

Crea un nuovo profilo

Nome completo

Email

Password

Conferma password

Registrati

Abilitare la conferma di registrazione

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

Se **ASP.NET Core 3.x**

```
services.AddDefaultIdentity<ApplicationUser>(options => {
    options.SignIn.RequireConfirmedAccount = true;
});
```

Se **ASP.NET Core 2.x**

```
services.AddDefaultIdentity<ApplicationUser>(options => {
    options.SignIn.RequireConfirmedEmail = true;
});
```

Inviare l'email di conferma registrazione

```
public class MailKitEmailSender : IEmailSender {  
    public async Task SendEmailAsync(string email, string subject, string htmlMessage) {  
        using var client = new SmtpClient();  
        await client.ConnectAsync("smtp.example.org", 25, SecureSocketOptions.StartTls);  
        await client.AuthenticateAsync("myuser", "mypassword");  
  
        var message = new MimeMessage();  
        message.From.Add(MailboxAddress.Parse("MyCourse <noreply@mycourse.com>"));  
        message.To.Add(MailboxAddress.Parse(email));  
        message.Subject = subject;  
        message.Body = new TextPart("html") { Text = htmlMessage };  
        await client.SendAsync(message);  
        await client.DisconnectAsync(true);  
    }  
}
```

MailKit.Net.Smtp

Installarlo con dotnet add package MailKit

Registrare l'implementazione di IEmailSender

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

```
services.AddSingleton<IEmailSender, MailKitEmailSender>();
```

Blocco automatico dell'account

Dopo alcuni tentativi di accesso falliti, il login è inibito per un certo tempo.

- Scoraggia i tentativi di accesso con la forza bruta.

Nel metodo `ConfigureServices` della classe `Startup`

```
services.AddDefaultIdentity<ApplicationUser>(options => {
    options.Lockout.AllowedForNewUsers = true;
    options.Lockout.MaxFailedAccessAttempts = 5;
    options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(5);
});
```

Blocco automatico dell'account

Dopo alcuni tentativi di accesso falliti, il login è inibito per un certo tempo.

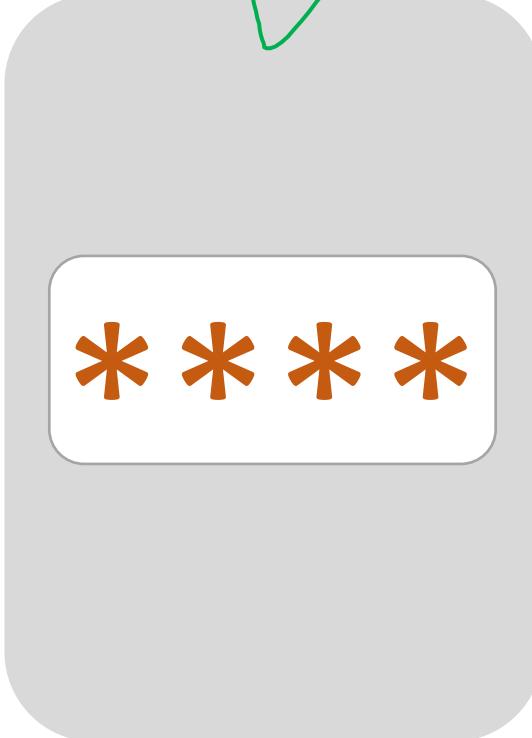
- Scoraggia i tentativi di accesso con la forza bruta.

Nella Razor Page `/Areas/Identity/Pages/Account/Login.cshtml.cs`

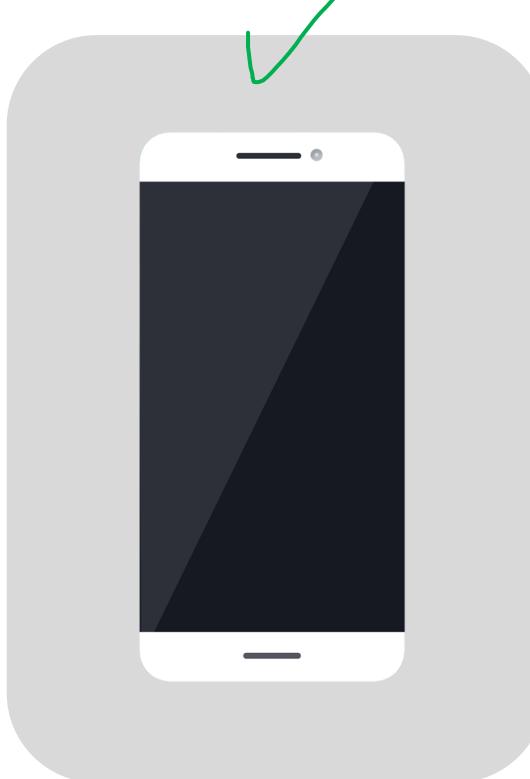
```
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    //...
    await _signInManager.PasswordSignInAsync(
        Input.Email, Input.Password, Input.RememberMe,
        lockoutOnFailure: true
    );
    //...
}
```

Autenticazione a due fattori (2FA)

L'utente deve dimostrare di essere il legittimo proprietario dell'account in due diversi modi.



Qualcosa che conosce

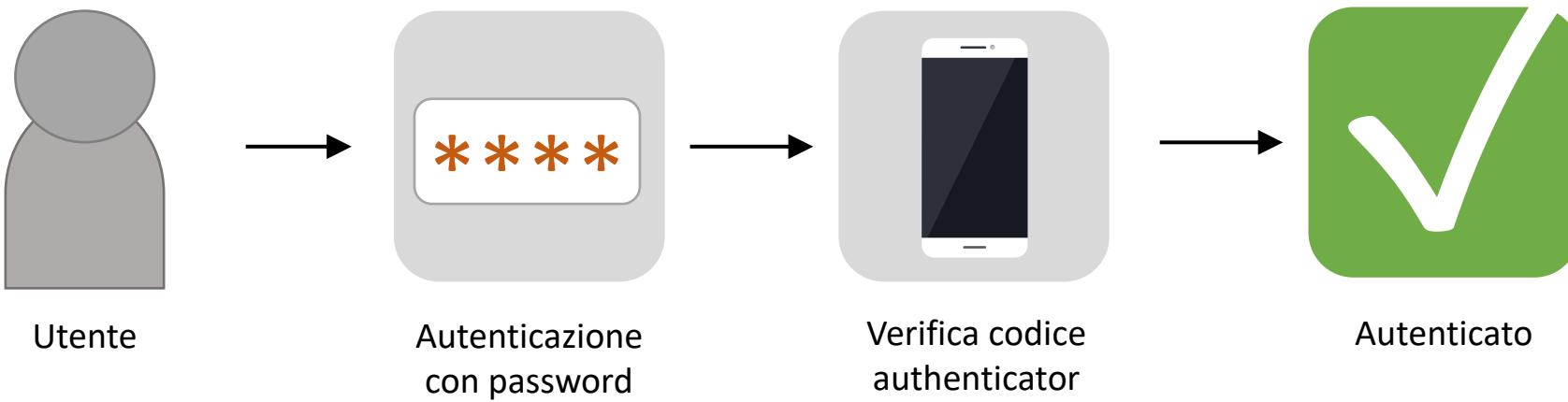


Qualcosa che possiede



Qualcosa che è

Autenticazione a due fattori (2FA)



Visualizzare il QRCode

1. Procurarci una libreria JavaScript in grado di generare il QRCode (es. qrcode.js)

```
libman install qrcodejs@1.0.0 --files qrcode.min.js
```

2. Incollare il codice nella Razor Page /Areas/Identity/Pages/Account/Manage/EnableAuthenticator.cshtml

```
<script type="text/javascript" src="~/lib/qrcodejs/qrcode.min.js"></script>
<script type="text/javascript">
    new QRCode(document.getElementById("qrCode"), {
        text: "@Html.Raw(Model.AuthenticatorUri)",
        width: 200,
        height: 200
    });
</script>
```

Ora che l'accesso è sicuro, attenzione a non introdurre punti deboli.





Dammi la password
del database di produzione



Tu vuoi cosa? Per fare che?

Davide Bianchi su "Storie dalla sala macchine"

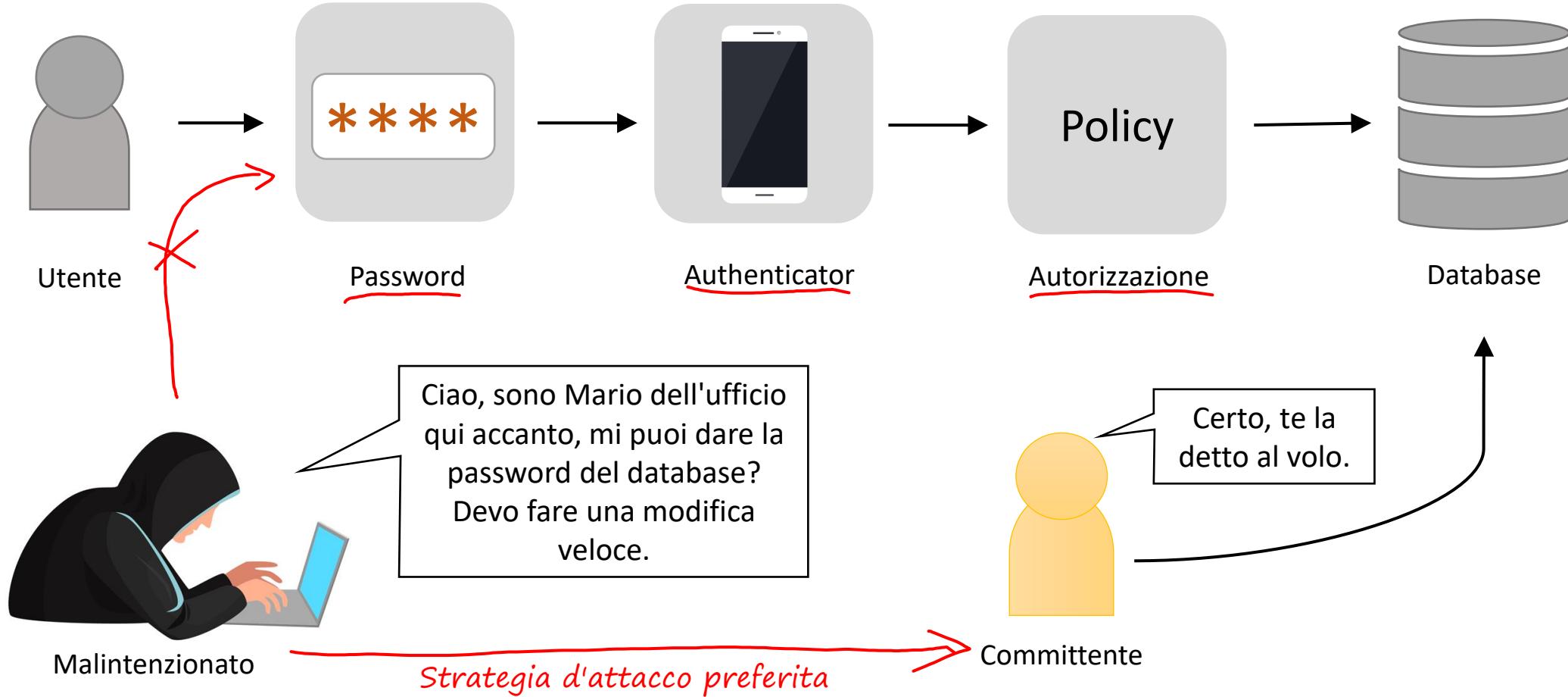
<https://www.soft-land.org/storie>



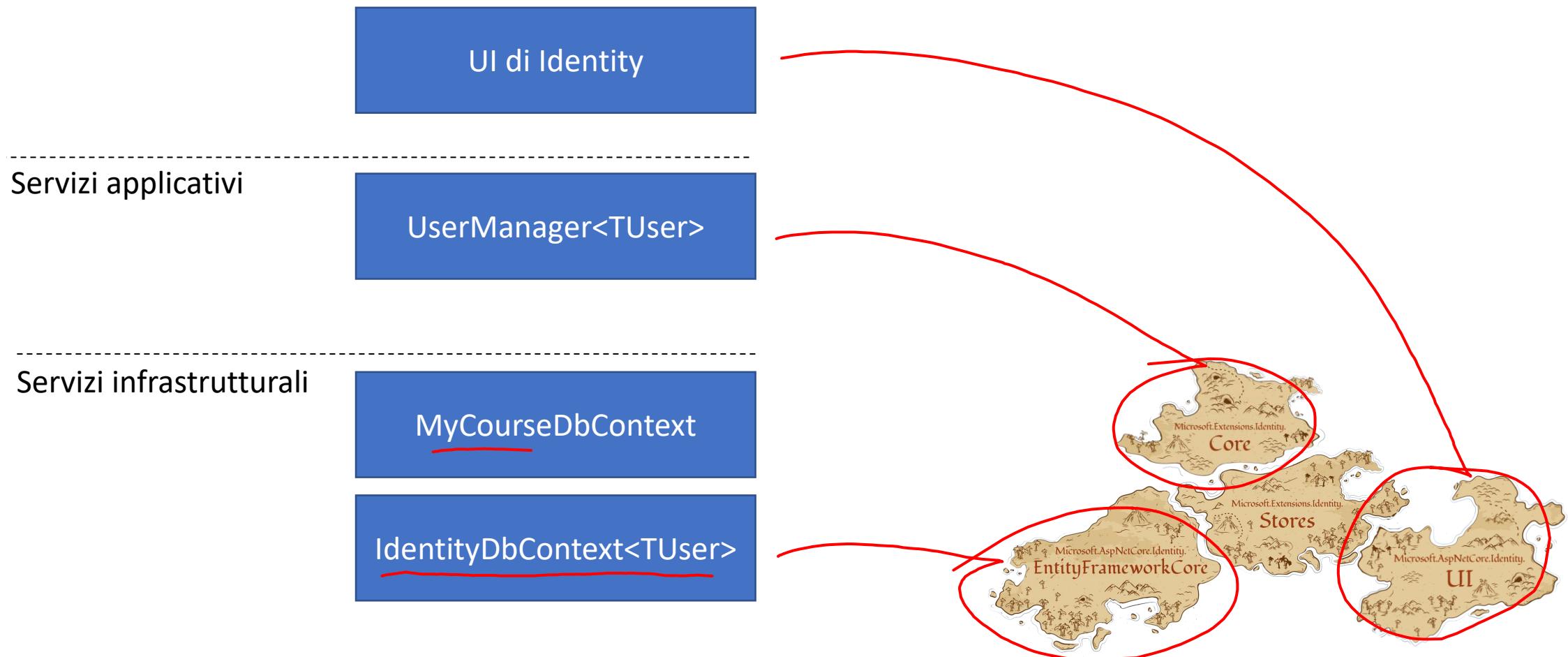
A businesswoman with blonde hair and glasses, wearing a dark blazer over a white shirt, sits at a desk with her arms crossed. A speech bubble graphic points towards her from the bottom left, containing the text "Adesso non ho tempo per spiegare, inviamela e basta".

Adesso non ho tempo per
spiegare, inviamela e basta

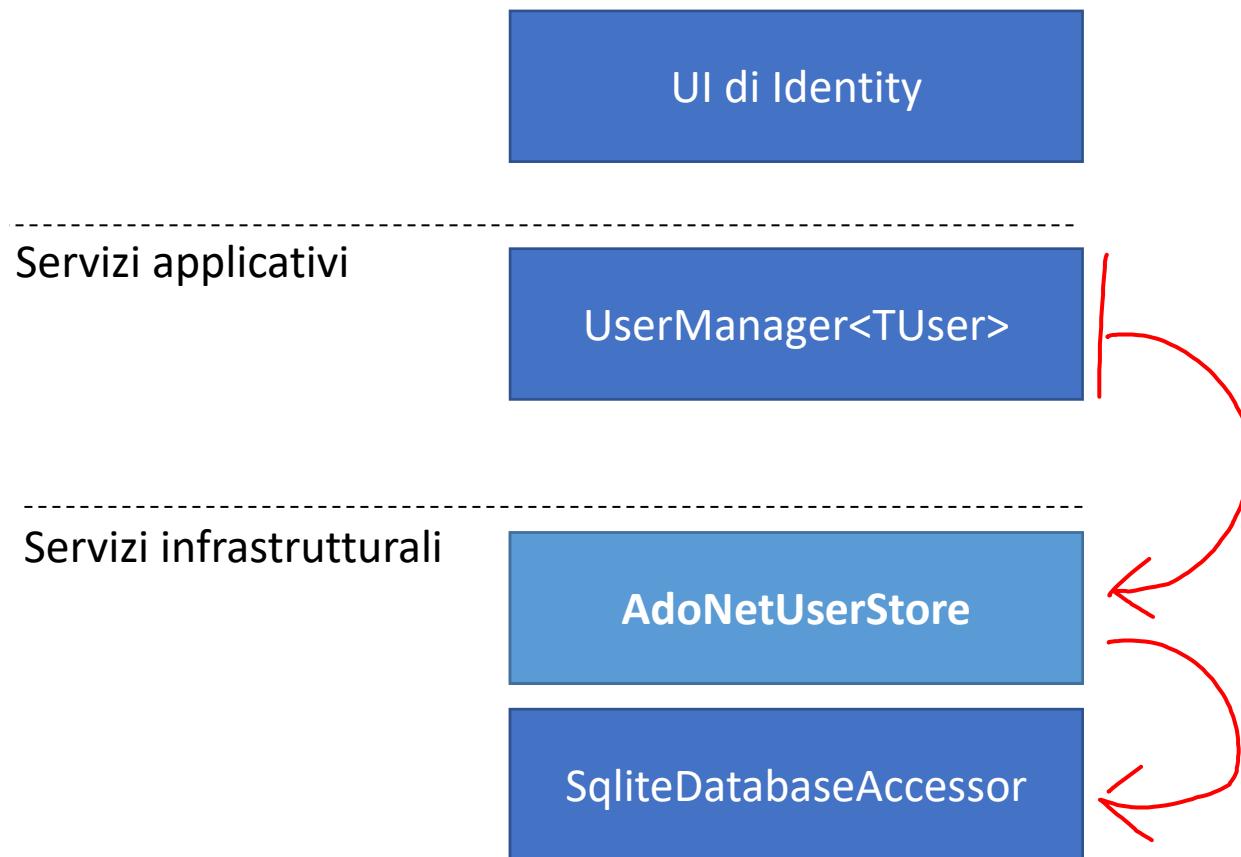
Cerchiamo di non introdurre punti deboli



Lo stack dei servizi di Identity (con EFCore)



Lo stack dei servizi di Identity (con ADO.NET)



AdoNetUserStore : IUserStore<TUser>

userStore.

- > `CreateAsync`
- > `DeleteAsync`
- > `FindByIdAsync`
- > `FindByNameAsync`
- > `GetNormalizedUserNameAsync`
- > `GetUserIdAsync`
- > `GetUserNameAsync`
- > `SetNormalizedUserNameAsync`
- > `SetUserNameAsync`
- > `UpdateAsync`



Altre interfacce opzionali

Interfaccia	Quando usarla
IUserClaimStore<TUser>	Se l'utente ha dei claim (es. il ruolo)
IUserEmailStore<TUser>	Se ha un'email
IUserPasswordStore<TUser>	Se ha una password
IUserConfirmation<TUser>	Se l'account deve essere confermato
IUserPhoneNumberStore<TUser>	Se ha un numero di telefono
IUserLockoutStore<TUser>	Se supportiamo il blocco dell'account
IUserTwoFactorStore<TUser>	Se supportiamo l'autenticazione a due fattori
IUserTwoFactorRecoveryCodeStore<TUser>	Se supportiamo i codici di recupero con la 2FA
IUserAuthenticatorKeyStore<TUser>	Se vogliamo usare un'app authenticator con la 2FA
IUserAuthenticationTokenStore<TUser>	Se, in generale, vogliamo usare dei codici di verifica

Altre interfacce opzionali (2)

Interfaccia	Quando usarla
<code>IUserLoginStore<TUser></code>	Se supportiamo l'accesso con servizi esterni (es. Facebook)
<code>IUserSecurityStampStore<TUser></code>	Se supportiamo il security stamp (es. per il logout remoto)
<code>IProtectedUserStore<TUser></code>	Se il database è in grado di offrire crittografia dei dati
<code>IQueryableUserStore<TUser></code>	Se supportiamo l'interrogazione con query LINQ

CancellationToken

È un oggetto che il chiamante di un metodo può usare per segnalargli che deve interrompere ciò che sta facendo.

- Utile in operazioni di "lunga durata", sia sincrone che asincrone

```
public void CalculatePrimeNumbers(CancellationToken token)
{
    while (!token.IsCancellationRequested)
    {
        Console.WriteLine(primeNumber);
    }
}
```

CancellationToken

È un oggetto che il chiamante di un metodo può usare per segnalargli che deve interrompere ciò che sta facendo.

- Utile in operazioni di "lunga durata", sia sincrone che asincrone

```
public void CalculatePrimeNumbers(CancellationToken token)
{
    while (true)
    {
        token.ThrowIfCancellationRequested();
        Console.WriteLine(primeNumber);
    }
}
```



```
if (token.IsCancellationRequested)
    throw new OperationCanceledException(token);
```

CancellationToken

È un oggetto che il chiamante di un metodo può usare per segnalargli che deve interrompere ciò che sta facendo.

- Utile in operazioni di "lunga durata", sia sincrone che asincrone

```
public async Task<IdentityResult> CreateAsync(ApplicationUser user, CancellationToken token)
{
    int affectedRows = await db.CommandAsync($"INSERT INTO ...", token);
    //...
}
```

Registrare lo user store basato su ADO.NET

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

```
services.AddDefaultIdentity<ApplicationUser>(options => {  
    // Opzioni...  
})  
.AddEntityFrameworkStores<MyCourseDbContext>() ← Non serve più  
.AddUserStore<AdoNetUserStore>();
```

CancellationTokenSource

È l'oggetto usato dal chiamante per ottenere il CancellationToken e cancellare l'operazione.

```
var tokenSource = new CancellationTokenSource();
var token = tokenSource.Token;
// ...
tokenSource.Cancel();
```

Identity funziona con varie tecnologie database

Qui si trova un elenco di user store sviluppati dalla community per varie tecnologie database:

<https://github.com/dotnet/aspnetcore/tree/release/3.1/src/Identity#community-maintained-store-providers>

Progresso nella specifica

- Punto 8: Registrazione degli utenti.

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
---	---	---	---