Programação Web

Aulas Teóricas – Capítulo 2 – 2.5

1º Semestre - 2023/2024

Departamento de Engenharia Informática e de Sistemas
Instituto Superior de Engenharia de Coimbra/Instituto Politécnico de Coimbra

Programação Web

Segurança em Aplicações Web

Departamento de Engenharia Informática e de Sistemas
Instituto Superior de Engenharia de Coimbra/Instituto Politécnico de Coimbra

Conceitos Gerais

- Autenticação
  - Processo de verificação de identidade de um utilizador ou computador
    - Quem é? Como o provar?
    - Verifica se utilizador é realmente quem diz ser
    - Credenciais podem ser especificadas através de um código de acesso (*password*), um *token* externo, um smart card, …

Conceitos Gerais

- Autorização
  - Processo para determinar o que é que o utilizador pode aceder num determinado computador, rede, …
    - Quais as permissões?
    - Tem acesso a este conteúdo?
    - Tem permissões para ver o conteúdo desta página?

Evolução: Identidade em ASP.NET

- 2005: *ASP.NET Membership*
  - Armazenamento e gestão de dados da identidade
  - *SQL Server*
- 2011: *Simple Membership*
  - Sistema de *Membership* para modelo *ASP.NET Web Pages*
  - Não é possível usar com o *OWIN* (*Open Web Interface for .NET*)

Evolução: Identidade em ASP.NET

- 2012: *Universal Providers Membership*
  - Passou a permitir armazenamento de dados adicionais
  - Suporte à associação de fornecedores externos
  - Permitiu armazenamento da informação *membership* no *Windows Azure SQL Database*
  - Permitiu o uso do *Membership* com a *EF*
- Após 2013: ***ASP.NET Identity***
- **Actualmente: *ASP.Net Core Identity*** *Framework*

ASP.Net Core Identity Framework

ASP.NET Core Identity: Características

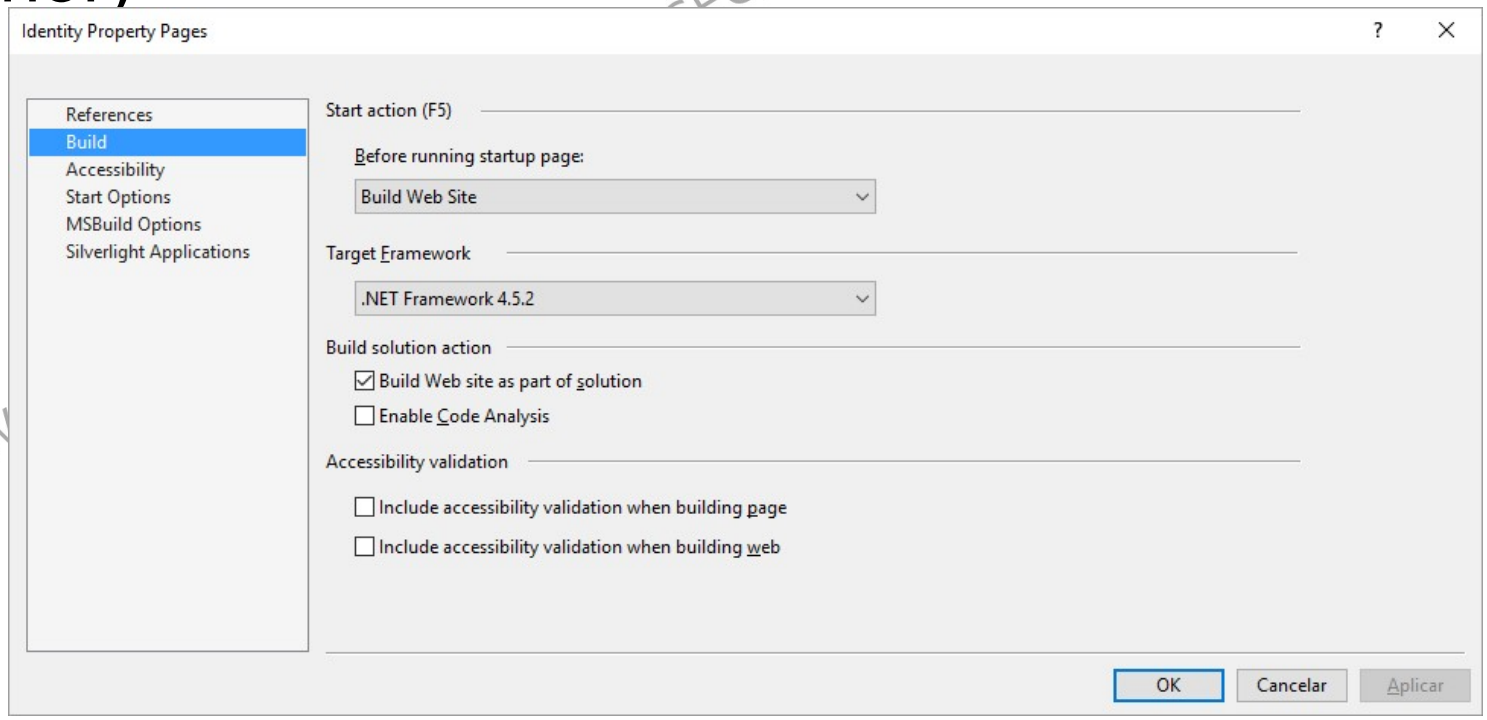
- **Autenticação** e **Autorização** para aplicações *ASP.NET Core*
  - Suporta *ASP.NET Core, MVC, Web API, Web Forms, SignalR, Web Pages*
- Permite efetuar a manipulação de utilizadores, *profiles*, processo de login/logout, criação de perfis, *claims*, etc…
  - Permite manter as contas do utilizador numa base de dados local ou num armazenamento de dados externo

ASP.NET Core Identity: Características

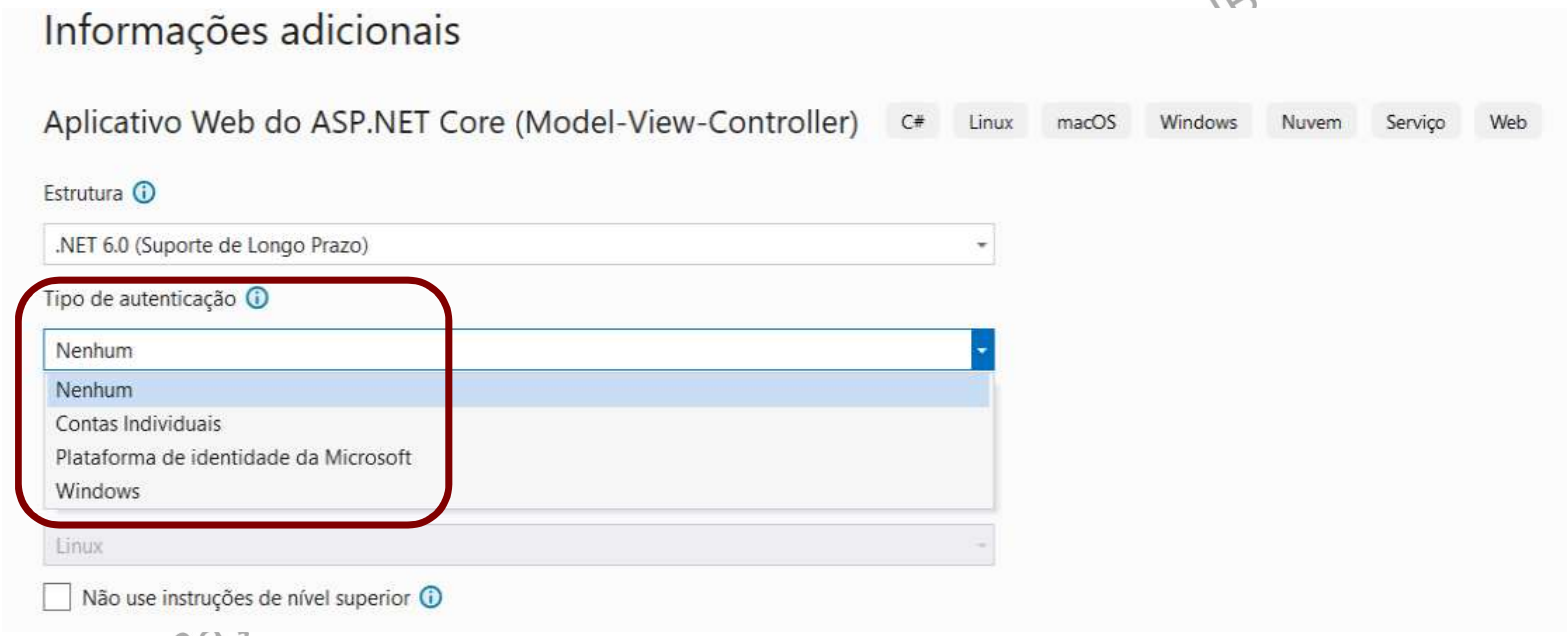
- Facilita escrita de testes unitários
- Permite login externo (através do ***OAuth 2.0***)
  - Suporta *Facebook*, *Google*, *Microsoft*, *Twitter*, *accounts*
- Suporta o *middleware* ***OWIN***
  - Pode executar num servidor que não o *IIS* da *Microsoft*
- Disponível através do gestor de bibliotecas ***NuGet***

ASP.NET Core Identity

- O *ASP.NET Core Identity* recorre à framework 6 (ou versão superior)



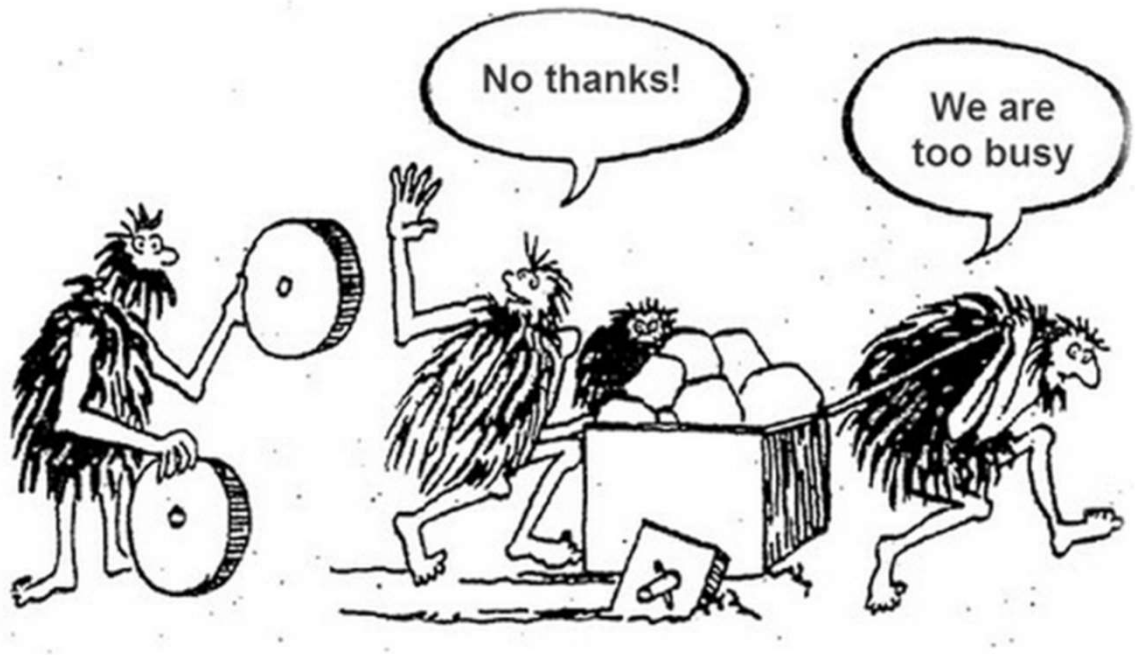
Formas de Autenticação

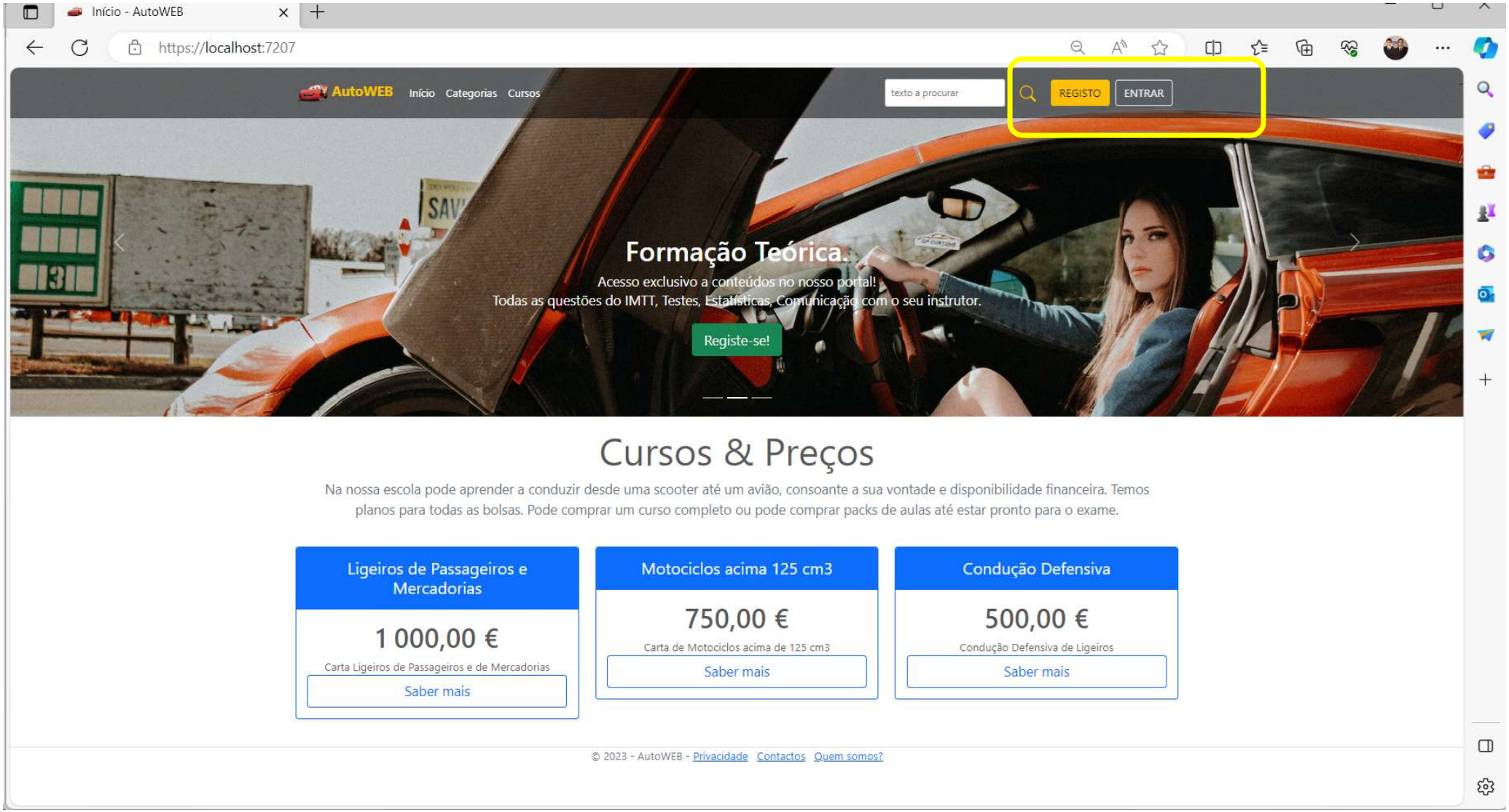


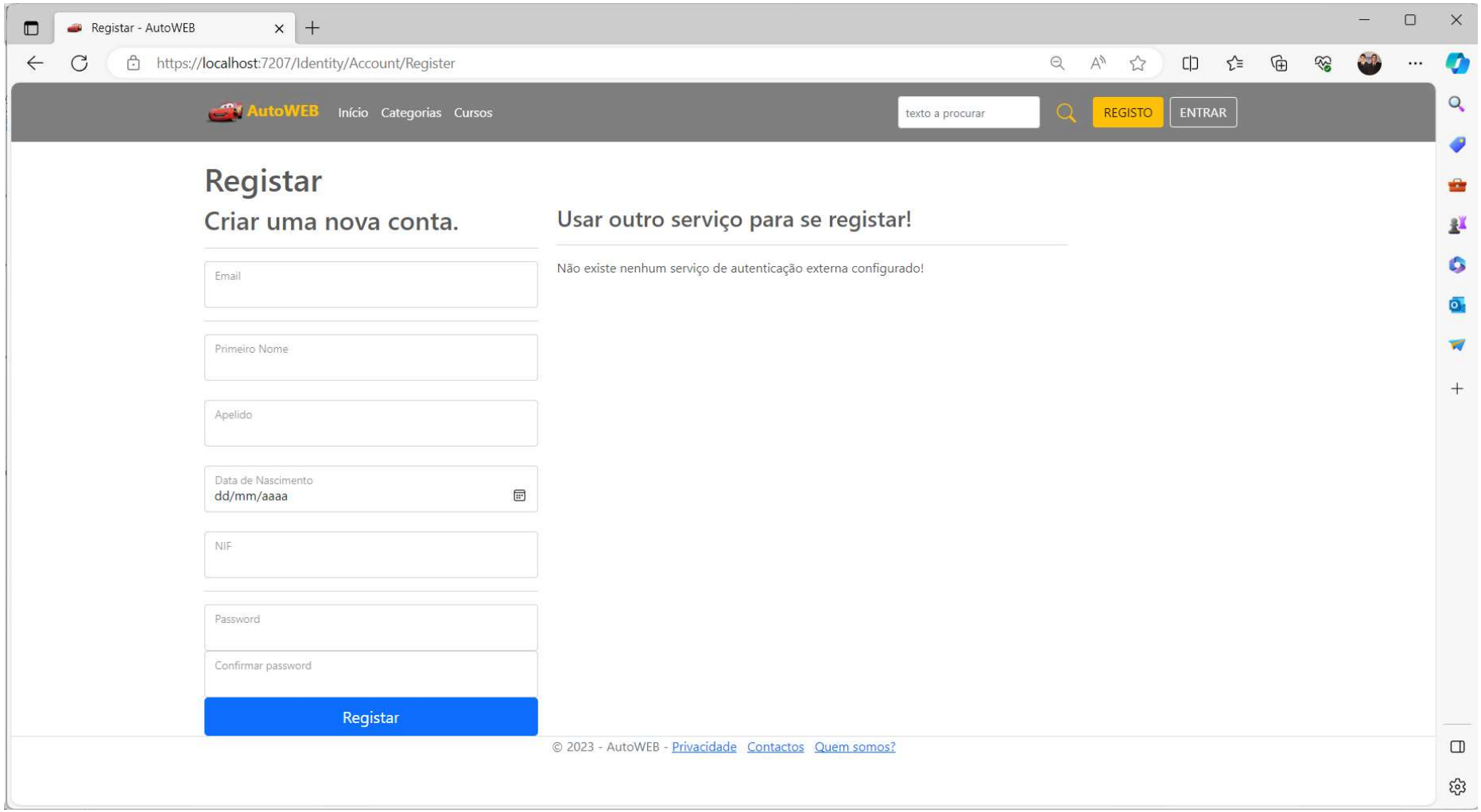
Formas de Autenticação

- **No Authentication**
  - Sem autenticação
- ***Individual User Accounts***
  - Tipicamente utilizado para *websites*
  - Permite a criação de utilizadores locais
  - Permite activação de logins com contas sociais como *Facebook, Google, Twitter*,…

- Para gerir perfis, acessos diferenciados…devo criar uma tabela de utilizadores e perfis?







ASP.NET Core Identity com Razor Pages

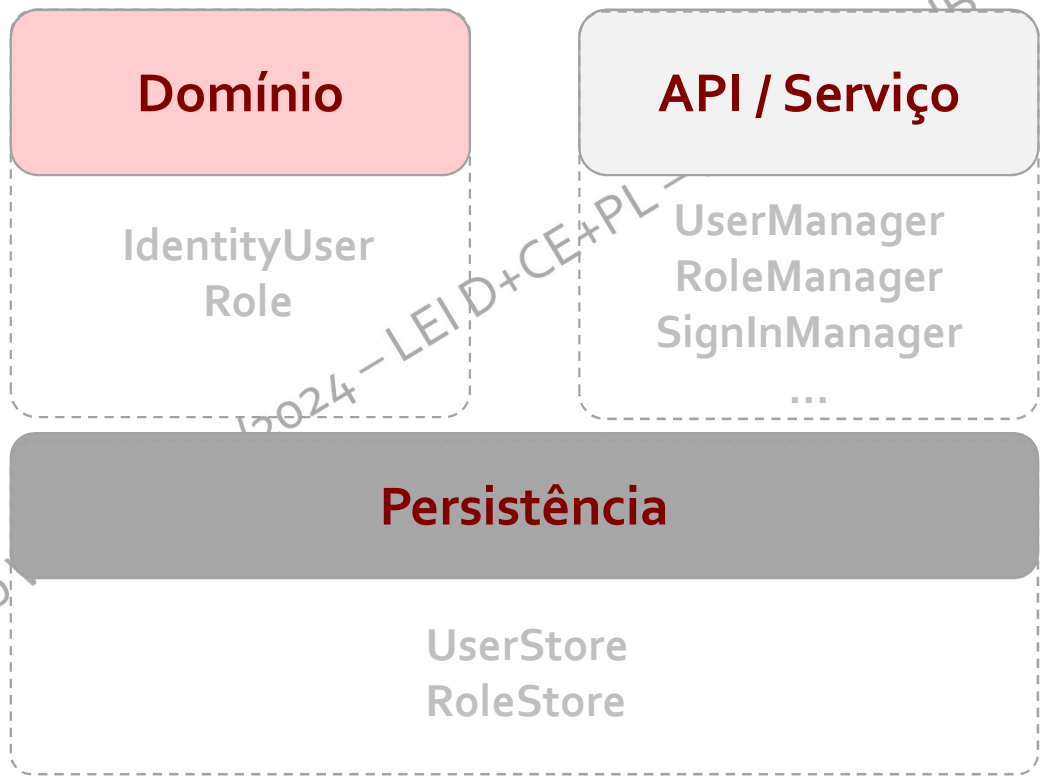
```
// Licensed to the .NET Foundation under one or more agreements.
// The .NET Foundation licenses this file to you under the MIT license.
#nullable disable

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Text.Encodings.Web;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI.Services;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.AspNetCore.WebUtilities;
using Microsoft.Extensions.Logging;
using AutoWEB.Models;

namespace AutoWEB.Areas.Identity.Pages.Account
{
    8 referências
    public class RegisterModel : PageModel
    {
        private readonly SignInManager<ApplicationUser> _signInManager;
        private readonly UserManager<ApplicationUser> _userManager;
        private readonly IUserStore<ApplicationUser> _userStore;
        private readonly IUserEmailStore<ApplicationUser> _emailStore;
        private readonly ILogger<RegisterModel> _logger;
        private readonly IEmailSender _emailSender;

        0 referências
        public RegisterModel(
            UserManager<ApplicationUser> userManager,
            IUserStore<ApplicationUser> userStore,
            SignInManager<ApplicationUser> signInManager,
            ILogger<RegisterModel> logger,
            IEmailSender emailSender)
        {
            _userManager = userManager;
            _userStore = userStore;
            _emailStore = GetEmailStore();
            _signInManager = signInManager;
            _logger = logger;
            _emailSender = emailSender;
        }
```

ASP.NET Core Identity: Arquitetura



ASP.NET *Identity*

UserManager, RoleManager

UserStore, RoleStore

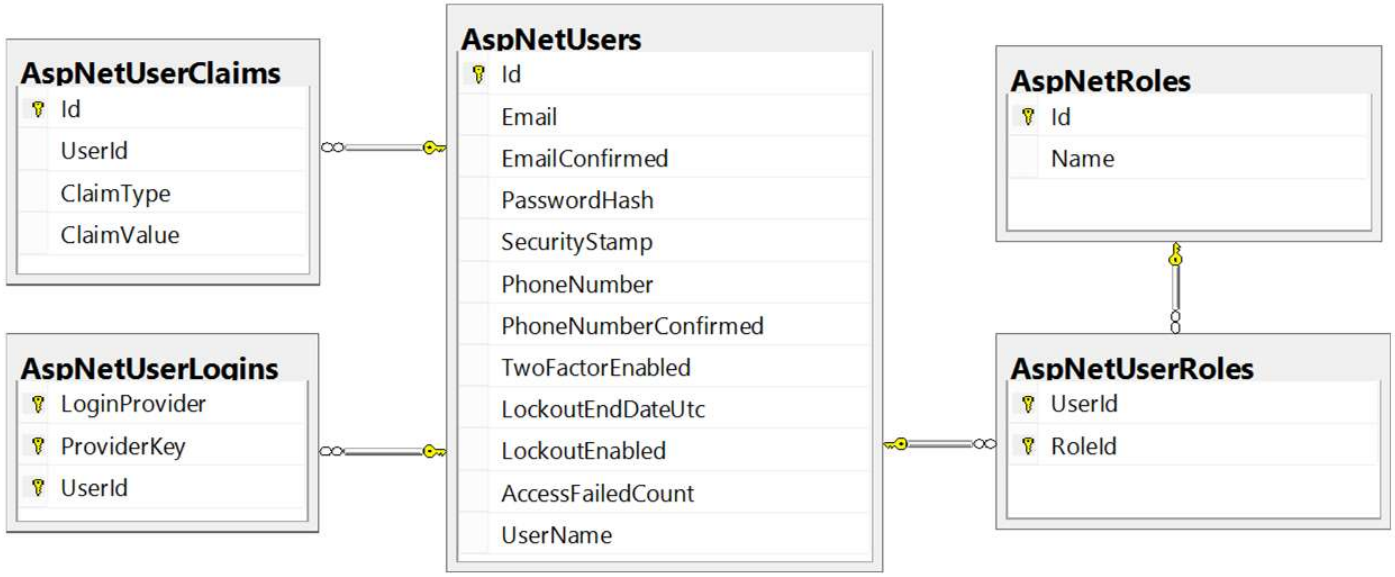
SQL Server, MySql, Windows Azure Table Storage

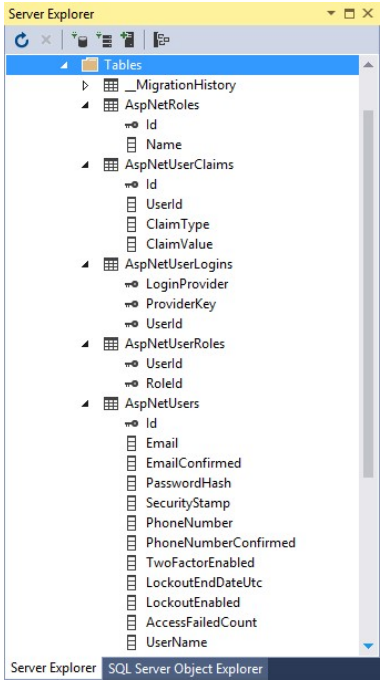
ASP.NET *Identity: Classes e Interfaces*

- As classes e interfaces mais importantes no *ASP.NET Identity* são as seguintes:
  - ***Iuser***
  - ***IdentityUser***
  - ***IdentityDbContext***
  - ***UserManager<Tuser>***
  - ***IAuthenticationManager***

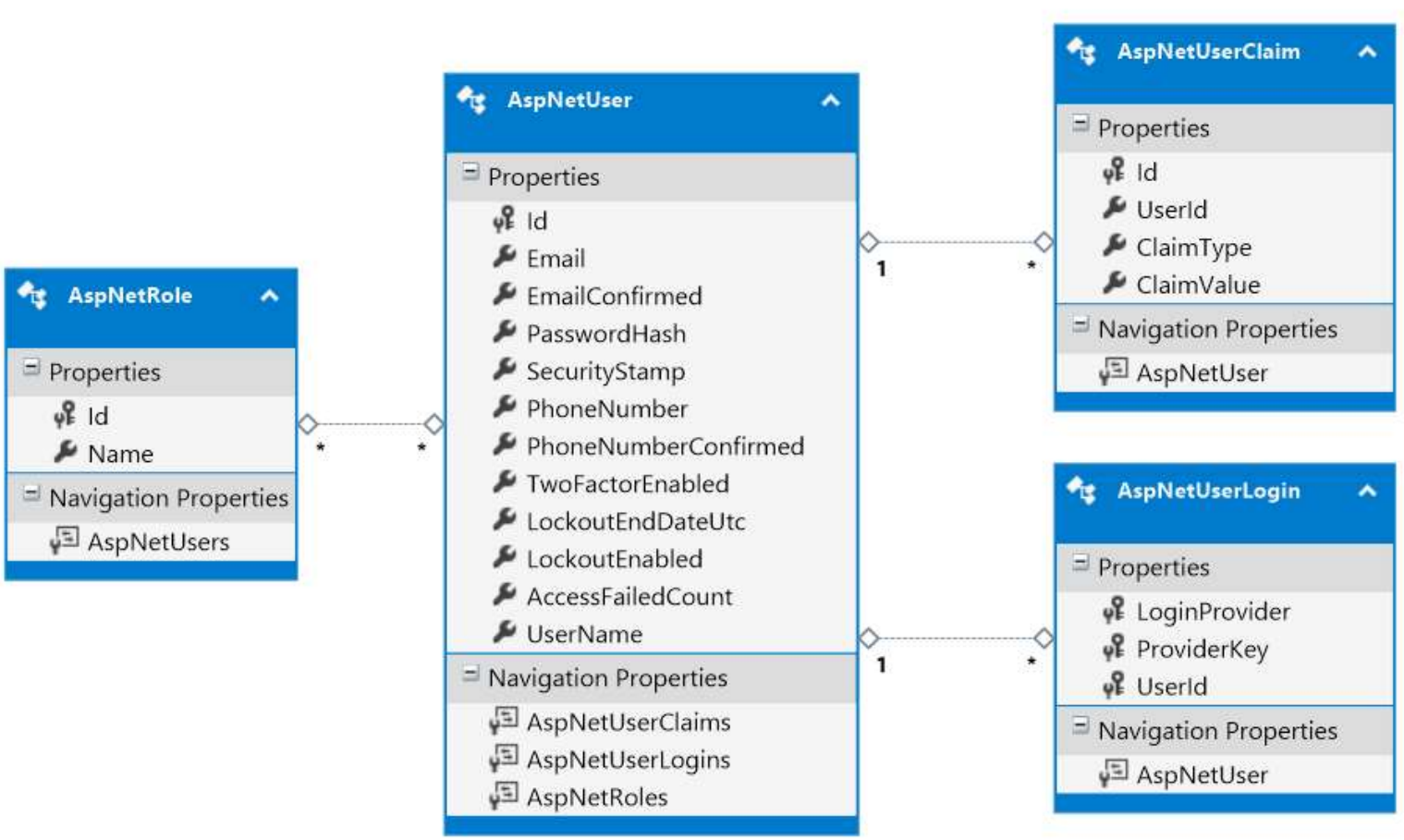
ASP.NET Core Identity: Modelo de Dados

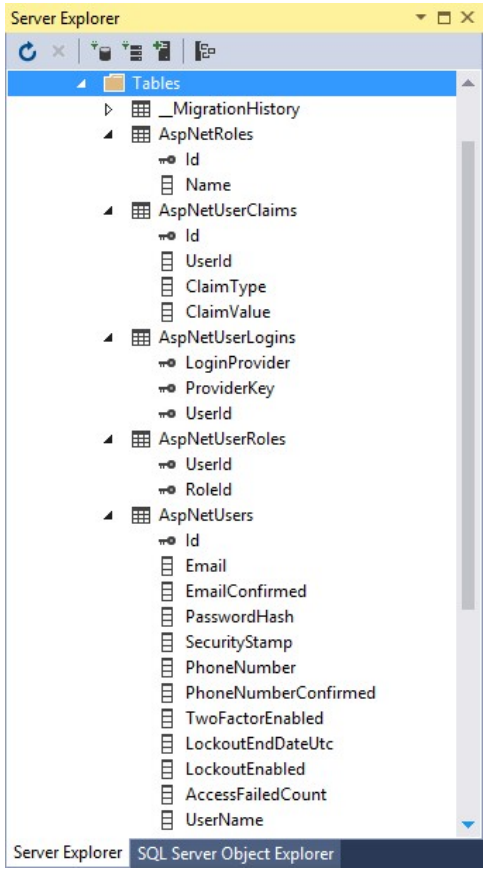
- Geralmente, os dados de identidade do ***ASP.NET Core*** (utilizadores, *passwords*, perfis) são armazenados numa base de dados relacional por meio da ***EF Code First***





ASP.NET Identity: Modelo de Dados





Estender IdentityUser - ApplicationUser

```
using Microsoft.AspNetCore.Identity;
using System.ComponentModel.DataAnnotations;

namespace AutoWEB.Models
{
    40 referências
    public class ApplicationUser : IdentityUser
    {
        [Display(Name = "Primeiro Nome")]
        9 referências
        public string PrimeiroNome { get; set; }

        [Display(Name = "Último Nome")]
        6 referências
        public string UltimoNome { get; set; }

        [Display(Name = "Data de Nascimento")]
        3 referências
        public DateTime DataNascimento { get; set; }
        3 referências
        public int NIF { get; set; }

        [Display(Name = "A minha fotografia")]
        3 referências
        public byte[]? Fotografia { get; set; }
        0 referências
        public ICollection<Agendamento> Agendamentos { get; set; }
    }
}
```

Identity Framework Core com Razor Pages

- **Iremos implementar a Identity com o recurso a Razor Pages**

ASP.NET Identity: Configuração

- Formas de configurar a autenticação baseada no *ASP.NET Core Identity*
  - Recorrer ao *template* do *ASP.NET* Core a partir do *Visual Studio*
  - Ou… Instalar bibliotecas através do *Nuget*
  - Bibliotecas a instalar:
    - `Microsoft.AspNet.Identity.EntityFrameworkCore`

ASP.NET Core Identity: Autorize

```
[Authorize]

public ActionResult Index()
{
    return View();
}
```

Filtro

Será chamado pela Framework antes e depois da acção ser exectuada

O atributo *Autorize* irá verificar se o utilizador corrente está bloqueado ou não

Se não estiver logado, reencaminha para a página de login

ASP.NET Identity: Autorize

- Aplicar no controlador

```
using System.Data.Entity;
using System.Linq;
using System.Net;
using System.Web.Mvc;
using CodeFirstApp.Models;

namespace CodeFirstApp.Controllers
{
    [Authorize]
    public class AlunosController : Controller
    {
        private EscolaContext db = new EscolaContext();

        // GET: Alunos
        public ActionResult Index()
        {
```

ASP.NET Identity: Autorize + Role

- Aplicar no controlador

```
using System.Data.Entity;
using System.Linq;
using System.Net;
using System.Web.Mvc;
using CodeFirstApp.Models;

namespace CodeFirstApp.Controllers
{
    [Authorize(Roles="Admin, Gestor")]
    public class AlunosController : Controller
    {

        // GET: Alunos
        public ActionResult Index()
        {
```

ASP.NET Identity: AllowAnonymous

```
using CodeFirstApp.Models;
using System;
using System.Linq;
using System.Web.Mvc;

namespace CodeFirstApp.Controllers
{
    [AllowAnonymous]
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
```


Restringir Ações a Utilizadores

```
[Authorize(Users="joao")]
public ActionResult Create()
{
    return View();
}
```

- Como especificar utilizadores / perfis iniciais?

Perfis e Roles Iniciais

```
using Microsoft.AspNetCore.Identity;
using AutoWEB.Models;
using System;

namespace AutoWEB.Data
{
    4 referências
    public enum Roles
    {
        Admin,
        Formador,
        Cliente
    }
    1 referência
    public static class Inicializacao
    {
        1 referência
        public static async Task CriaDadosIniciais(UserManager<ApplicationUser> userManager, RoleManager<IdentityRole> roleManager)
        {
            //Adicionar defaut Roles
            await roleManager.CreateAsync(new IdentityRole(Roles.Admin.ToString()));
            await roleManager.CreateAsync(new IdentityRole(Roles.Formador.ToString()));
            await roleManager.CreateAsync(new IdentityRole(Roles.Cliente.ToString()));

            //Adicionar Default User - Admin
            var defaultUser = new ApplicationUser
            {
                UserName = "admin@localhost.com",
                Email = "admin@localhost.com",
                PrimeiroNome = "Administrador",
                UltimoNome = "Local",
                EmailConfirmed = true,
                PhoneNumberConfirmed = true
            };
            if (userManager.Users.All(u => u.Id != defaultUser.Id))
            {
                var user = await userManager.FindByEmailAsync(defaultUser.Email);
                if (user == null)
                {
                    await userManager.CreateAsync(defaultUser, "Is3C..00");
                    await userManager.AddToRoleAsync(defaultUser, Roles.Admin.ToString());
                }
            }
        }
    }
}
```

Restringir Ações

- Criar novas vistas quando é necessário restringir muitas partes de uma vista

```
public ActionResult Index()
{
    if (User.IsInRole("ManipulaAlunos"))
        return View(db.Alunos.ToList());
    return View("VistaApenasLeitura",db.Alunos.ToList());
}
```

Restrições / Permissão

```
// GET: Cursos/Edit/5
[Authorize(Roles = "Admin")]
public async Task<IActionResult> Edit(int? id)
{
    if (id == null || _context.Cursos == null)
    {
        return NotFound();
    }
...
...
```

```
@if (@User.IsInRole("Admin"))
  {
    <small><a class="btn btn-primary btn-sm" asp-action="Create">Adicionar</a></small>
  }
```

Restrições / Permissão

- As restrições de acesso de funcionalidades a determinados utilizadores devem ser implementadas:
  - No inicio do Controller quando a restrição/permissão seja global a todo o Controller
  - Em determinadas Action dos Controllers quando a restrição/permissão se refira só a essa Action
  - Nas Views, restringido/permitindo o acesso a Links, Botões e etc …
  - Nos menús, nos ficheiros Layout, para restringir/permitir a visualização das opções do menú

Adicionar dados ao *Perfil*

- Adicionar propriedade PrimeiroNome

```
public class ApplicationUser : IdentityUser
    {
        public string PrimeiroNome { get; set;
}
    ...
```

- Criar Migração
- Atualizar a base de dados
- Alterar o Register.cshtml correspondente para aceitar a nova propriedade
- Alterar o Register.cshtml.cs para gravar esta propriedade
  - Razor Page: Register.cshtml + Register.cshtml.cs

Adicionar dados ao *Perfil*