LAB 7 - WEB API

1 - LIBRARY

Depois de criar o projeto **Class Library (.Net Core)**, num pasta de solução, e juntar os modelos e classes. É necessário instalar os seguintes packages que utilizamos no projeto anterior:

- **Install-Package Microsoft.EntityFrameworkCore.SqlServer**
- **Install-Package Microsoft.EntityFrameworkCore.Tools**
- **Install-Package Microsoft.EntityFrameworkCore.Design**
- **Install-Package Microsoft.AspNetCore.Identity.EntityFrameworkCore**

Depois compilar, tudo OK?

Adicionar à solução **ASP.NET Core Web Application** com nome ProjApi, com o template **API** e **no authentication**. **Set as StartUp Project**

No projeto **Web Api** adicionar **dependency -> reference** a **Library**.

2 - API

Na classe **`Startup,`** dentro do método **`ConfigureServices,`** imediatamente antes de **`services.AddControllers(),`** adicione a linha:

```
services.AddDbContext<ApplicationDbContext>(options =>
options.UseSqlServer(
Configuration.GetConnectionString("DefaultConnection")));
```

E depois no appSettings.json, a ligação à base de dados

```
"ConnectionStrings": {
"DefaultConnection":
"Server=(localdb)\\mssqllocaldb;Database=EsteCarIIIDb;Trusted_Connection=True;MultipleActiveResultSets=true"
},
```

--

Na classe Program: `DbInitializer.Initialize(context);`

Depois, é preciso instalar tb neste projeto (API) as mesmas bibliotecas através do Nuget:

- **Install-Package Microsoft.EntityFrameworkCore.SqlServer**
- **Install-Package Microsoft.EntityFrameworkCore.Tools**
- **Install-Package Microsoft.EntityFrameworkCore.Design**
- **Install-Package Microsoft.AspNetCore.Identity.EntityFrameworkCore**

+ add-migration Inicial e updata-database

--

Adicionar Controller para Marcas

Meter anotação na class
[Route(“Api/[Controller]”)]
[ApiController]

3 – CLIENTcar

Adicionar à solução Console App (.NET Core) nome EstCarIIIClient

Adicionar dependências -> reference para Library

- **Install-Package Microsoft.AspNet.WebApi.Client**

```
// coloque o porto utilizado no seu projecto
// ver na barra de endereço do browser ou dentro de launchSettings.json
cliente.BaseAddress = new Uri("http://localhost:62083/");
```

Métodos de Extensão

1-

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EsteCarIIILibrary
{
    public static class MetodosDeExtensao
    {

        public static String ListaMarcas(this List<Marca> Marcas)
        {
            StringBuilder stb = new StringBuilder("{");
            String virgula="";
            foreach (Marca marca in Marcas) {
                stb.Append(virgula + marca);
                virgula = (virgula == "")?  ", ": virgula; // ou mais si
            }
            stb.Append("}");
            return stb.ToString();
        }
    }
}
```

2 -

```
namespace ExtensionMethods
{
    public static class MyExtensions
    {
        public static int WordCount(this String str)
        {
            return str.Split(new char[] { ' ', '.', '?' },
                             StringSplitOptions.RemoveEmptyEntries).Length;
        }
    }
}
```

METODOS ASSINCRONOS

1-

```
public static async Task<Marca> GetMarcaAsync(string path)
{
    Marca marca = null;
    HttpResponseMessage response = await cliente.GetAsync(path);
    if (response.IsSuccessStatusCode)
    {
        marca = await response.Content.ReadAsAsync<Marca>();
    }
    return marca;
}
```

```
marca = GetMarcaAsync($"api/Marcas/{id}").GetAwaiter().GetResult();

if (marca != null)
    Console.WriteLine("marca: {0}", marca.Designacao);
```

2 -

```
static async Task<HttpStatusCode> UpdateMarcaAsync(Marca marca)
{
    HttpResponseMessage response = await cliente.PutAsJsonAsync(
        $"api/marcas/{marca.MarcaId}", marca);
    response.EnsureSuccessStatusCode();
    return response.StatusCode;
}
```

```
if (marca != null)
{
    marca.Designacao = nomeNovo;
    var status2 = UpdateMarcaAsync(marca).GetAwaiter().GetResult();
    Console.WriteLine("status: {0}", status2);
}
```

View Components

```
[ViewComponent(Name="EmployeeList1")]       -> Optional : Para mudar o nome do componente
public class TestViewComponent : ViewComponent
{
   ....
   ....
}
```

```
public async Task<IViewComponentResult> InvokeAsync()
{
    List<string> cenas = new List<string>() { "a", "b", "c" };

    return View(cenas);
}
```

NA VIEW, INCLUIR O COMPONENTE:

```
<div>
   @await Component.InvokeAsync("AlugueresHoje")
</div>
```

A view do componente por defeito chama-se default.cshtml. Deve ser colocada em **Views\Shared\Components\NOME_COMPONENTE**

Podemos passar o nome da View assim: return View("nome_view", cenas)

Criar um componente:

1 - Create new folder named **ViewComponents**. In this folder, create new class named **CategoryViewComponent.cs**

2 **-** Create new folder named **Views**. In **Views** folder, create new folders with path **Views\Shared\Components\Category**. In **Category** folder, create new file named **default.cshtml**:

Para usar Dependecy injection é necessário:

1 - criar na Pasta Services:
Interface e Classe com o mesmo nome (sem o i) que a Implemente.

2- registar o serviço no startup.cs como Scoped:

```
services.AddScoped<IAlugueresHoje, AlugueresHoje>();
```

TESTS

```
using Xunit;

namespace EsteCarIITest
{
    0 references
    public class HomeControllerTest
    {
        [Fact]
        0 references
        public void Index_ReturnsViewResult()
        {
            var controller = new HomeController(null);
            var result = controller.Index();
            var viewResult = Assert.IsType<ViewResult>(result);
        }

        [Fact]
        0 references
        public void Privacy_ReturnsViewResult()
        {
            var controller = new HomeController(null);
            var result = controller.Privacy();
            var viewResult = Assert.IsType<ViewResult>(result);
        }
    }
}
```

Não esquecer das dependências do projeto que estamos a testar

Teste com contexto

SQLITE

```
public class MarcasControllerTest
{
    [Fact]
    public async Task Index_CanLoadFromContext()
    {
```

```
var connection = new SqliteConnection("DataSource=:memory:");
connection.Open();
var options = new DbContextOptionsBuilder<ApplicationDbContext>()
    .UseSqlite(connection)
    .Options;

using (var context = new ApplicationDbContext(options))
{
    context.Database.EnsureCreated();
    context.Marca.AddRange(
        new Marca { Designacao = "Ferrary" },
        new Marca { Designacao = "Porche" },
        new Marca { Designacao = "BMW" });
    context.SaveChanges();
}
```

+ teste

```
using (var context = new ApplicationDbContext(options))
{
    var controller = new MarcasController(context);

    var result = await controller.Index();

    var viewResult = Assert.IsType<ViewResult>(result);
    var model = Assert.IsAssignableFrom<IEnumerable<Marca>>(
        viewResult.ViewData.Model);
    Assert.Equal(3, model.Count());
}
```

}

TESTs

```
[Fact]
public async Task Details_ReturnsNotFoundResult_WhenMarcaDoesntExist()
{
    var connection = new SqliteConnection("DataSource=:memory:");
    connection.Open();
    var options = new DbContextOptionsBuilder<ApplicationDbContext>()
        .UseSqlite(connection)
        .Options;

    using (var context = new ApplicationDbContext(options))
    {
        context.Database.EnsureCreated();
        context.SaveChanges();
    }
    using (var context = new ApplicationDbContext(options))
    {
        var controller = new MarcasController(context);

        var result = await controller.Details(1);

        Assert.IsType<NotFoundResult>(result);
    }
}
```

```
[Fact]
public async Task Details_RetunrsViewResult_WhenMarcaExists()
{
    var connection = new SqliteConnection("DataSource=:memory:");
    connection.Open();
    var options = new DbContextOptionsBuilder<ApplicationDbContext>()
        .UseSqlite(connection)
        .Options;

    using (var context = new ApplicationDbContext(options))
    {
        context.Database.EnsureCreated();
        context.Marca.Add(new Marca { Designacao = "Ferrary" });
        context.SaveChanges();
    }
    using (var context = new ApplicationDbContext(options))
    {
        var controller = new MarcasController(context);

        var result = await controller.Details(1);

        var viewResult = Assert.IsType<ViewResult>(result);
        var model = Assert.IsAssignableFrom<Marca>(viewResult.ViewData.Model);
        Assert.Equal(1, model.MarcaId);
    }
}
```

PARA USAR O MESMO OBJETO EM VÁRIOS TESTES -> CLASS Fixture

```
public class ApplicationDbContextFixture
{
    public ApplicationDbContext DbContext { get; private set; }

    public ApplicationDbContextFixture()
    {
        var connection = new SqliteConnection("DataSource=:memory:");
        connection.Open();
        var options = new DbContextOptionsBuilder<ApplicationDbContext>()
                .UseSqlite(connection)
                .Options;
        DbContext = new ApplicationDbContext(options);

        DbContext.Database.EnsureCreated();

        // Adicionar Marcas para testes
        DbContext.Marca.AddRange(
            new Marca { Designacao = "Ferrary" },
            new Marca { Designacao = "Porsche" },
            new Marca { Designacao = "BMW" });
        DbContext.SaveChanges();

        DbContext.Carro.AddRange(
            new Carro { MarcaId = (DbContext.Marca.First(m => m.Designacao.Contains
            new Carro { MarcaId = (DbContext.Marca.First(m => m.Designacao.Contains
        DbContext.SaveChanges();
    }
}
```

!! Para usar a fixture definida tem de implementar a interface IClassFixture<> que não tem métodos

```
public class CarrosControllerTest : IClassFixture<ApplicationDbContextFixture>
{
    private ApplicationDbContext _context;

    public CarrosControllerTest(ApplicationDbContextFixture contextFixture)
    {
        _context = contextFixture.DbContext;
    }

    [Fact]
    public async Task Index_CanLoadFromContext()
    {
        var controller = new CarrosController(_context);

        var result = await controller.Index();

        var viewResult = Assert.IsType<ViewResult>(result);
        var model = Assert.IsAssignableFrom<IEnumerable<Carro>>(
            viewResult.ViewData.Model);
    }

    [Fact]
    public async Task Index_MarcaIsIncluded_WhenLoadsFromContext()
    {
        var controller = new CarrosController(_context);

        var result = await controller.Index();

        var viewResult = Assert.IsType<ViewResult>(result);
        var model = Assert.IsAssignableFrom<IEnumerable<Carro>>(
            viewResult.ViewData.Model);
        Assert.NotNull(model.FirstOrDefault().Marca);
    }
```

(…)

Mais um teste, CarroIsIncluded

```
[Fact]
public async Task Index_CarroIsIncluded_WhenLoadsFromContext()
{
    var controller = new AlugueresController(_context);

    var result = await controller.Index();

    var viewResult = Assert.IsType<ViewResult>(result);
    var model = Assert.IsAssignableFrom<IEnumerable<Aluguer>>(
        viewResult.ViewData.Model);
    Assert.NotNull(model.FirstOrDefault().Carro);
}
```

Mais exemplos Assert.True(…), False.., Not Found

```
[Fact]
public void AluguerExists_ReturnsTrue_WhenItExists()
{
    var controller = new AlugueresController(_context);

    var result = controller.AluguerExists(1);

    Assert.True(result);
}

[Fact]
public void AluguerExists_ReturnsFalse_WhenItDoesntExist()
{
    var controller = new AlugueresController(_context);

    var result = controller.AluguerExists(0);

    Assert.False(result);
}

[Fact]
public async Task Delete_ReturnsNotFoundResult_WhenIdIsNull()
{
    var controller = new AlugueresController(_context);

    var result = await controller.Delete(null);

    var viewResult = Assert.IsType<NotFoundResult>(result);
}
```

INTEGRATION TESTES

```
namespace AspNetCoreTodo.IntegrationTests
{
    public class TestFixture : IDisposable
    {
        private readonly TestServer _server;
        public HttpClient Client { get; }
        public TestFixture()
        {
            var builder = new WebHostBuilder()
                .UseStartup<AspNetCoreTodo.Startup>()
                .ConfigureAppConfiguration((context, config) =>
                {
                    config.SetBasePath(Path.Combine(
                        Directory.GetCurrentDirectory(),
                        "..\\..\\..\\..\\AspNetCoreTodo"));

                    config.AddJsonFile("appsettings.json");
                });

            _server = new TestServer(builder);

            Client = _server.CreateClient();
            Client.BaseAddress = new Uri("http://localhost:8888");
        }

        public void Dispose()
        {
            Client.Dispose();
            _server.Dispose();
        }
    }
}
```

```
namespace AspNetCoreTodo.IntegrationTests
{
    public class TodoRouteShould : IClassFixture<TestFixture>
    {
        private readonly HttpClient _client;

        public TodoRouteShould(TestFixture fixture)
        {
            _client = fixture.Client;
        }

        [Fact]
        public async Task ChallengeAnonymousUser()
        {
            // Arrange
            var request = new HttpRequestMessage(
                HttpMethod.Get, "/todo");

            // Act: request the /todo route
            var response = await _client.SendAsync(request);

            // Assert: the user is sent to the login page
            Assert.Equal(
                HttpStatusCode.Redirect,
                response.StatusCode);

            Assert.Equal(
                "http://localhost:8888/Account" +
                "/Login?ReturnUrl=%2Ftodo",
                response.Headers.Location.ToString());
        }
    }
}
```

This test makes an anonymous (not-logged-in) request to the `/todo` route and verifies that the browser is redirected to the login page.

This scenario is a good candidate for an integration test, because it involves multiple components of the application: the routing system, the controller, the fact that the controller is marked with `[Authorize]`, and so on. It's also a good test because it ensures you won't ever accidentally remove the `[Authorize]` attribute and make the to-do view accessible to everyone.

Run the test

Run the test in the terminal with **`dotnet test`**. If everything's working right, you'll see a success message:

```
Starting test execution, please wait...
 Discovering: AspNetCoreTodo.IntegrationTests
 Discovered:  AspNetCoreTodo.IntegrationTests
 Starting:    AspNetCoreTodo.IntegrationTests
 Finished:    AspNetCoreTodo.IntegrationTests

Total tests: 1. Passed: 1. Failed: 0. Skipped: 0.
Test Run Successful.
Test execution time: 2.0588 Seconds
```

ANGULAR

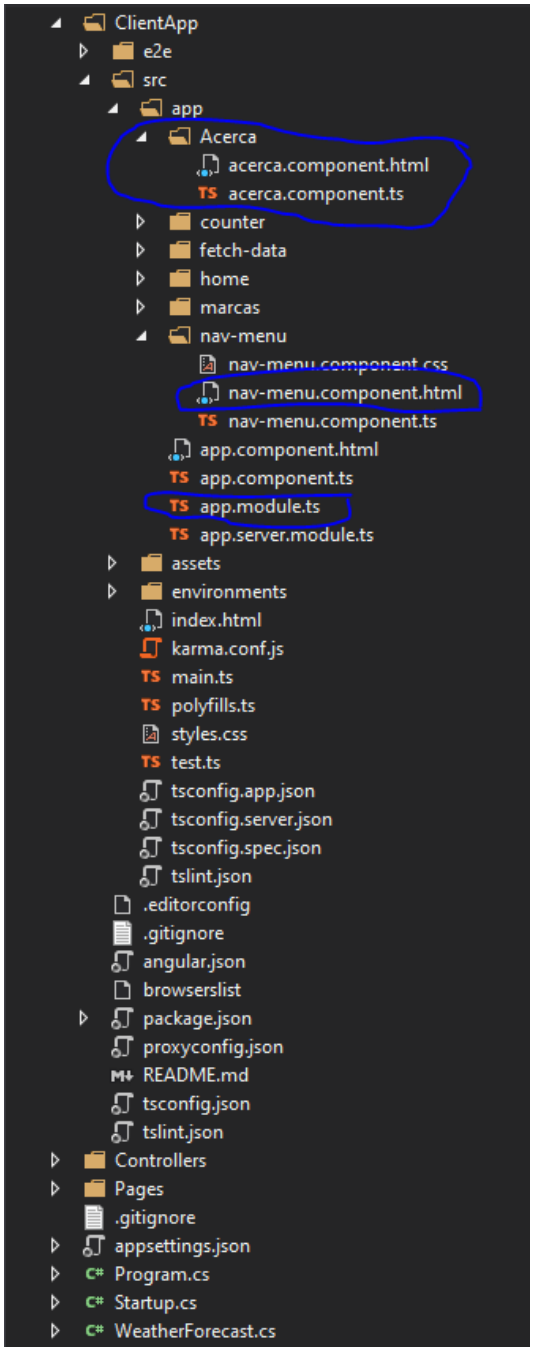
Usar API

proxyconfig.json na pasta onde estiver o ficheiro **package.json**

```
{
"/api": {
"target": "https://localhost:44365",
"secure": false,
"changeOrigin": true
}
```

Coloque o seguinte código no método **`ConfigureServices`** da classe **`Startup`**:

```
services.AddCors(options =>
{
options.AddPolicy("CorsPolicy",
builder => builder.WithOrigins("http://localhost:52250"));
});

app.UseCors("CorsPolicy");
```



Na pasta com o nome do componente cria-se:

Html e ts. Pode tb ter css

No app.modules.ts:

Import …

Declaration [

… ,

AcercaComponent,

…

]

Routermodule.forRoot([

… ,

{ path: ‘acerca’, componente: AcercaComponent },

…

])

Criar Component Acerca

(ng generate componente NomeComponete) ou:

Criar pasta:
ClientApp > src > app > Acerca :

acerca.component.html
acerca.component.ts

No app.module.ts:

```
import { AppComponent } from './app.component';
import { NavMenuComponent } from './nav-menu/nav-menu.component';
import { HomeComponent } from './home/home.component';
import { CounterComponent } from './counter/counter.component';
import { FetchDataComponent } from './fetch-data/fetch-data.component';
import { AcercaComponent } from './acerca/acerca.component';
import { MarcasComponent } from './marcas/marcas.component';

@NgModule({
    declarations: [
        AppComponent,
        NavMenuComponent,
        HomeComponent,
        CounterComponent,
        FetchDataComponent,
        AcercaComponent,
        MarcasComponent
    ],
    imports: [
        BrowserModule.withServerTransition({ appId: 'ng-cli-universal' }),
        HttpClientModule,
        FormsModule,
        RouterModule.forRoot([
            { path: '', component: HomeComponent, pathMatch: 'full' },
            //{ path: 'counter', component: CounterComponent },
            //{ path: 'fetch-data', component: FetchDataComponent },
            { path: 'acerca', component: AcercaComponent },
            { path: 'marcas', component: MarcasComponent },
        ])
    ],
    providers: [],
    bootstrap: [AppComponent]
})
export class AppModule { }
```

Incluir um link. Por exemplo: no nav-menu.component.html

```
            >Marcas</a
          >
          <li class="nav-item" [routerLinkActive]="['link-active']">
            <a class="nav-link text-dark" [routerLink]="['/acerca']"
            >Acerca</a
            >
          </li>
```

Codigo de componente TS e html

marcas.component.html

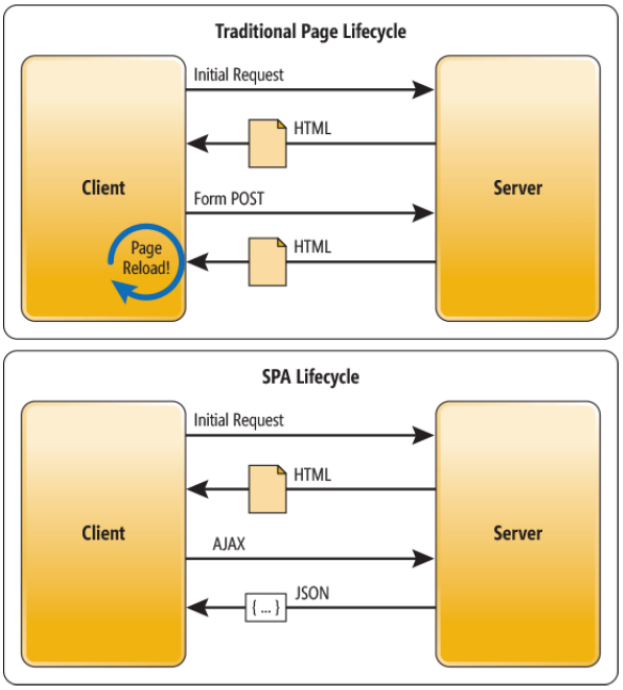
```html
<h1>Marcas</h1>

<p>Marcas existentes no servidor</p>

<p *ngIf="!marcas"><em>Loading...</em></p>

<table class='table' *ngIf="marcas">
  <thead>
    <tr>
      <th>Id</th>
      <th>Designação</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let marca of marcas">
      <td>{{ marca.marcaId }}</td>
      <td>{{ marca.designacao }}</td>
    </tr>
  </tbody>
</table>

```

marcas.component.ts

```typescript
import { Component, Inject } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Component({
  selector: 'app-marcas',
  templateUrl: './marcas.component.html'
})
export class MarcasComponent {
  public marcas: Marca[];

  constructor(http: HttpClient) {
    http.get<Marca[]>('api/Marcas').subscribe(result => {
      this.marcas = result;
    }, error => console.error(error));
  }
}

interface Marca {
  marcaId: number;
  designacao: string;
}
```

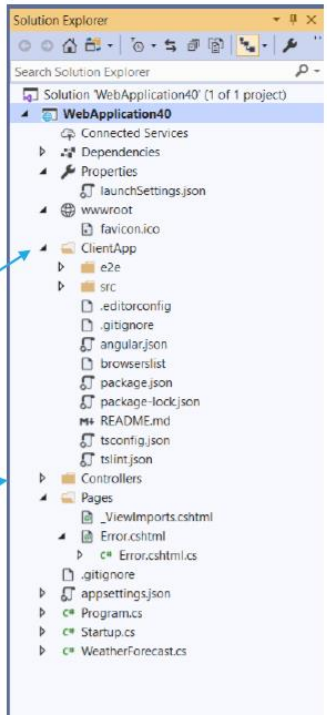
Atenção: interface Marca.

ANGULAR SLIDES



O Template Angular permite juntar 2 projetos num só:

- Um projeto Angular responsável pela UI de *Frontend* (client side)
- Um projeto ASP.NET Core responsável pela API de *Backend* (server side)



O Angular é uma framework JavaScript para a criação de aplicações Web

- Criado originalmente em 2009, por um técnico da Google – Miško Hevery – como um projeto "lateral" para simplificar o desenvolvimento Web.
  - Baseado em javaScript, aplica o padrão MVC.

- Foi reescrito em 2014 pela equipa da Google e lançado oficialmente em Setembro de 2016 como Angular 2.
  - As versões posteriores são referidas como Angular 2+.
  - Atualmente é chamado simplesmente Angular e vai na versão 9.0
    - Usa TypeScript, uma versão tipificada do JavaScript
    - Documentado no site oficial: https://angular.io/

O Angular é uma plataforma para a criação de aplicações Web que combina:

- Modelos declarativos (templates).
- Injeção de dependências (dependency injection).
- Um conjunto de ferramentas completo para todo o processo de desenvolvimento (end-to-end tooling).
- Um conjunto integrado de práticas de testes.

TYPESCRIPT

É um "superconjunto" do JavaScript

- Foi desenvolvido pela Microsoft
  - Um dos autores é o Anders Hejlsberg que esteve na criação do C# e que criou também as linguagens Delphi e Turbo Pascal.
- É uma linguagem fortemente tipificada, com verificação de tipos na altura da compilação.
- Os ficheiros .ts de typescript são compilados para ficheiros .js (javaScript)
  - *Transpilling*
  - O código JavaScript gerado é mais rápido (e menos legível).

- Exemplo

```
class Student {
  fullName: string;
  constructor(public firstName, public middleInitial, public lastName) {
    this.fullName = firstName + " " + middleInitial + " " + lastName;
  }
}

interface Person {
  firstName: string;
  lastName: string;
}

function greeter(person: Person) {
  return "Hello, " + person.firstName + " " + person.lastName;
}

var user = new Student("Jane", "M.", "User");

//document.body.innerHTML = greeter(user);
```