

INTRODUÇÃO AO DESENVOLVIMENTO WEB COM BLAZOR (.NET CORE 5)

INTRODUÇÃO

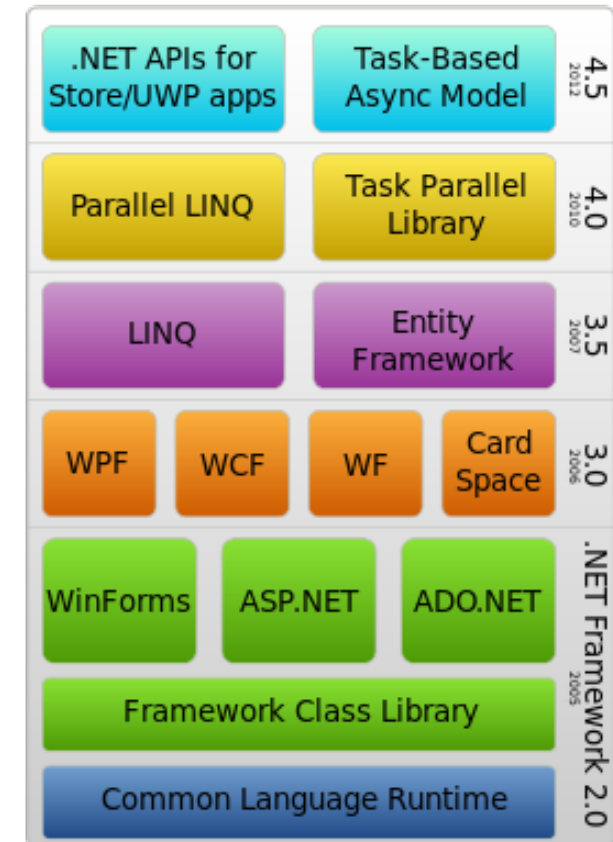


AGENDA

- Introdução à framework .NET;
- Desenvolvimento WEB utilizando ASP.NET Core 5.0
- Introdução a Blazor
- Blazor Server VS WebAssembly
- Ferramentas/tecnologias front-end

FRAMEWORK .NET

- .NET Framework é um ambiente de execução que fornece uma [biblioteca](#) de classes abrangente;
- permite aos programadores desenvolver aplicações robustas com código confiável para todas as principais áreas de desenvolvimento;



FRAMEWORK .NET CORE

- A framework .NET Core é versão **modular** da .NET Framework;
- Uma das características principais desta biblioteca passa pela capacidade de instalar os componentes que são **necessários** para a aplicação;
- Permite que diferentes **versões** de uma aplicação possam coexistir na mesma máquina sem problemas de compatibilidade da framework .NET;
- A ASP.NET Core foi completamente **reescrita** a partir da framework ASP.NET de forma a ser *cross-platform*, *open source* e **sem limitações** de **compatibilidade**;

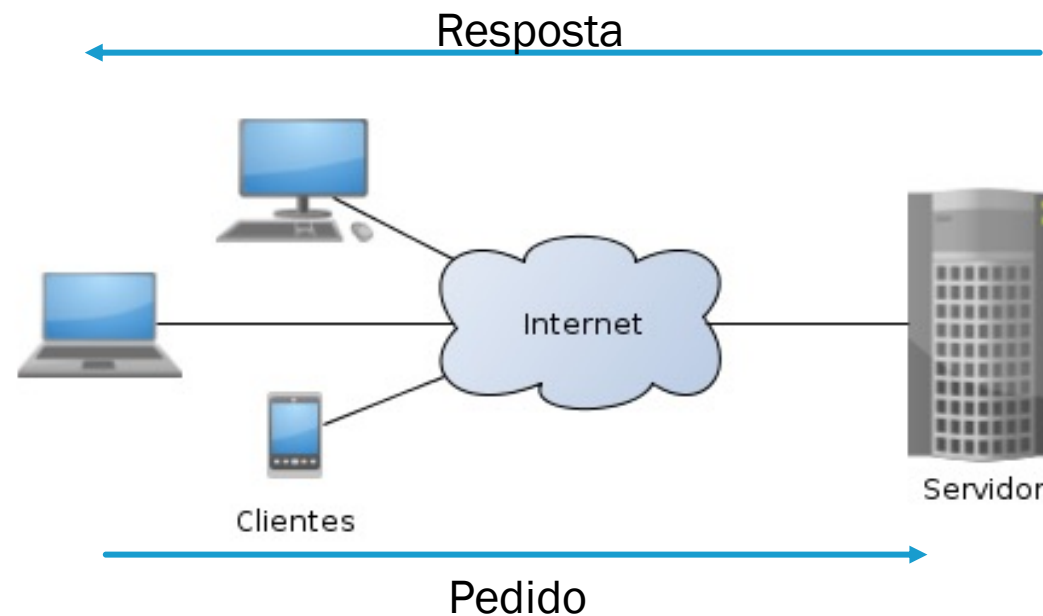
MICROSOFT WEB STACK

■ Modelos de desenvolvimento:

- **Web Forms (2002)**: Permite a construção de websites dinâmicos utilizando uma interface *drag-and-drop* baseado num modelo orientado a eventos a eventos; Permite a criação de aplicações de forma rápida e simples;
- **ASP.NET MVC (2009)**: A framework **MVC** (MVC5) da Microsoft aplica o padrão MVC (Model-View-Controller) sobre o ASP.NET, permitindo o desenvolvimento de aplicações web que promovem a reutilização, organização e desempenho do código;
- **ASP.NET Core** é **diferente** das versões anteriores.

NÃO ESQUECER O PROTOCOLO HTTP

- Na arquitetura **Cliente/Servidor** os **clientes** (por exemplo web browser) realizam **pedidos** de forma a requisitar recursos disponibilizados por **servidores web**;
- A comunicação cliente/servidor é realizada através do **protocolo** de aplicação: HTTP (*HyperText Transport Protocol*):
- Para dois computadores comunicarem é necessário que ambos conheçam o protocolo em termos de **sintaxe**, **semântica** e o **timing**.





DEMO: CRIAR PROJETO BLAZOR DE EXEMPLO

APP STARTUP: ASP.NET CORE

- A classe startup:
 - Utiliza o método `ConfigureServices` para configurar os serviços da app;
 - Um serviço é um componente reutilizável que disponibiliza uma funcionalidade;
 - Os serviços são consumidos através do mecanismo de `dependency injection` ou `Application services`;
 - O método `Configure` cria um pipeline de processamento do pedido HTTP por parte da app;
 - No método `configure` podemos configurar, por exemplo, páginas específicas de desenvolvimento, gestão de exceções, redireccionamento, disponibilização de ficheiros estáticos ou páginas HTML/Razor;

Documentação adicional:
<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/startup?view=aspnetcore-5.0>

INJEÇÃO DE DEPENDÊNCIAS

- O ASP.NET Core possui um mecanismo de gestão de dependências;
- Ao invés de instanciar classes específicas para gerir dependências de um projeto, as classes recebem as instâncias (tipicamente) como parâmetros do constructor;
- O ficheiro [startup.cs](#) permite configurar/registar as dependências do projeto;



BLAZOR

WEB APPS

- With an ever-expanding sea of frameworks, **tools and ecosystems** from which to choose, it can seem harder than ever to figure out the best option for your business when it comes to **building modern web applications**;
- Until recently JavaScript was generally considered the **go-to** option for building “modern” web applications with Blazor;
- Microsoft has create **Blazor**,
- Like JavaScript, **runs in the browser**, but your applications are written **using C#**;
- Your application and a **small version of the .NET runtime** are then shipped to the browser where everything runs via WebAssembly;

WEB APPS

- If the **learning curve** meant you couldn't do anything meaningful with it for the first three months, it probably **wouldn't be your first choice!**
- **Leverage Your Existing Skills:** For many C# developers, JavaScript is either a complete mystery, or at least yet another thing to learn.
- If C# is the **primary language of your enterprise**, then Blazor may be the best choice.
- It also avoids the **pitfalls of having multiple languages** to maintain.

WEB APPS

- Possibilities for front-end development
 - React (<https://reactjs.org/>) ships with a **component model** and the ability to bind components to data but very little else.
 - This is a deliberate choice by the React team, leaving it up **to other parties** to provide additional concerns like **routing, support for forms, etc.**
 - Because of this, React itself is **fairly straightforward** to learn (due to its relatively **limited scope**).

WEB APPS

- Possibilities for front-end development
 - Angular (<https://angular.io/>), on the other hand, includes routing, forms and virtually anything else you can think of.
 - The **upside** of this is you **don't have to jump off to various third-party providers** to plug the gaps, but it also makes the **surface area** of Angular much, much bigger, meaning **there's a lot to learn**.

WEB APPS

- Blazor (<https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor>) occupies a sort of middle ground here, shipping with a **solid component model**, plus **routing and forms**, and an **HTTP client** for making **network requests**.
- Overall Blazor is “**lighter**” than Angular in this regard.
- It also builds on established Microsoft technologies, so if your **team is familiar with ASP.NET** routing and writing markup using Razor, much of Blazor’s functionality will come naturally to them.
- Blazor, despite being the newest kid in town has an ecosystem with rich tools, such as Telerik UI (<https://www.telerik.com/blazor-ui>) or Syncfusion (<https://www.syncfusion.com/blazor-components>), providing tools to speed up your development.

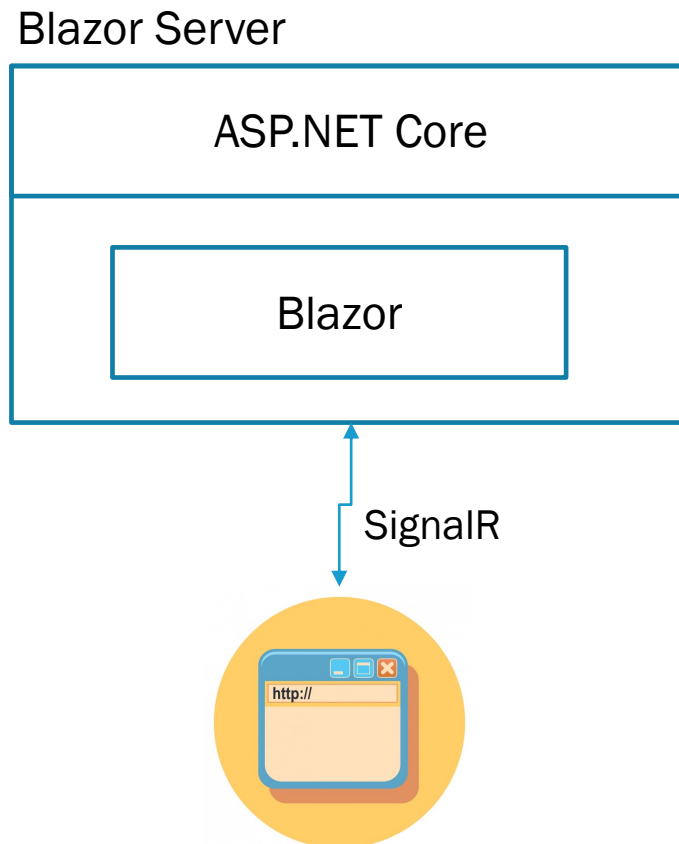
BLAZOR

- **Blazor applications** are composed of components that are constructed using C#, HTML-based **Razor syntax**, and CSS.
- Blazor has two different **runtime modes**: **server-side Blazor** and **client-side Blazor**, also known as **Blazor WebAssembly**.
- Blazor brings a modern component-based version of **Razor**, making it possible to break your features down into **tiny components** which are small, focused and therefore quicker to build.
- Once you have these components you can easily compose them together to **make a bigger feature**, or an entire application.

BLAZOR

- Both modes run in all modern web browsers, including web browsers on mobile phones
- Client-side Blazor is composed of the same code as server-side Blazor; however, it runs entirely in the web browser using a technology known as WebAssembly (<https://webassembly.org/>)
- The primary difference in Blazor applications that are created in server-side Blazor versus client-side Blazor is that the client-side Blazor applications need to make web calls to access server data, whereas the server-side Blazor applications can omit this step as all their code is executed on the server.

BLAZOR SERVER VS BLAZOR WEBASSEMBLY



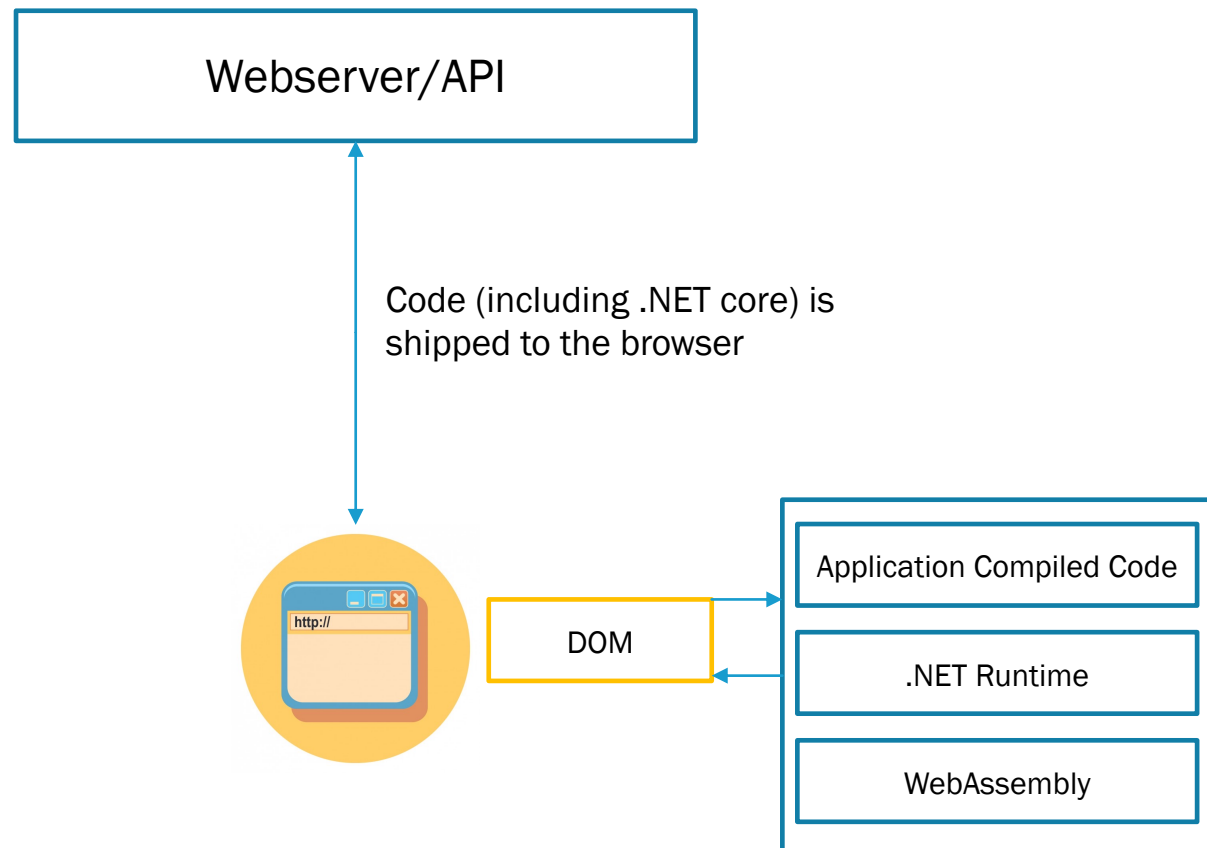
Application runs on the server

SignalR component enables the communication

Easier to develop since integrates all components for web app development

BLAZOR SERVER VS BLAZOR WEBASSEMBLY

Blazor WebAssembly



Webserver/API need to be developed separately (it is useful if you already have an existing API)

Runs on the client (displaying data)

Obviously, you cannot connect directly to the database

It is useful for front-end

Supports progressive web app (specially used if you want to ship web app to mobile phones)



BLAZOR - COMPONENTS AND ROUTING

- A Blazor application is composed of components;
- A component is a **chunk of code** consisting of a user interface and processing logic;
- A Blazor component is also called a **Razor component**;
- Blazor features routing, where you can **provide navigation to your controls** using the **@page** directive followed by a unique route in quotes preceded by a slash;
- A Razor component is contained in a **.razor** file and can be **nested inside** of other components.

BLAZOR - COMPONENTS AND ROUTING

- For example, we can create a component named **ComponentOne.razor** using the following code.

```
<h4 style="background-color:goldenrod">
  This is ComponentOne
</h4>
```

```
@code { }
```

→
We can alter
ComponentExample.razor to
contain ComponentOne.razor.

```
@page "/componentexample"
<h3>This is Component Example</h3>
```

```
<ComponentOne />
```

```
@code {
}
```

BLAZOR - PARAMETERS

- Razor components can pass values to other components using **parameters**.
- Component parameters are defined using the **[Parameter]** attribute, which must be declared as **public**.

```
<h4>Parameter Example Component</h4>
```

```
<h5 style="color:red">@Title</h5>  
@code {  
    [Parameter]  
    public string Title { get; set; }  
}
```



```
@page "/parameterexample"  
<h4>Parameter Example</h4>  
<ParameterExampleComponent  
    Title="Passed from Parent" />  
@code {  
}
```

BLAZOR - DATA BINDING

- Simple, **one-way binding** in Blazor is achieved by declaring a parameter and referencing it using the @ symbol. An example of this is shown in the following code:

```
<b>BoundValue:</b> @BoundValue
@code {
    private string BoundValue { get; set; }

    protected override void OnInitialized() {
        BoundValue = "Initial Value";
    }
}
```

BLAZOR - DATA BINDING

- **Two-way**, dynamic data binding in Razor components is implemented using the **@bind** attribute.

```
<input @bind="BoundValue" @bind:event="oninput" />
<p>Display CurrentValue: @BoundValue</p>
@code {
    private string BoundValue { get; set; }
}
```


BLAZOR - EVENT

- Raising events in Razor components is straightforward. The following example demonstrates using the **@onclick** event handler to execute the method `IncrementCount` when the button is clicked

```
<p>Current count: @currentCount</p>  
<button class="btn btn-primary" @onclick="IncrementCount"> Click me </button>
```

```
@code {  
    private int currentCount = 0;  
    private void IncrementCount() {  
        currentCount++;  
    }  
}
```

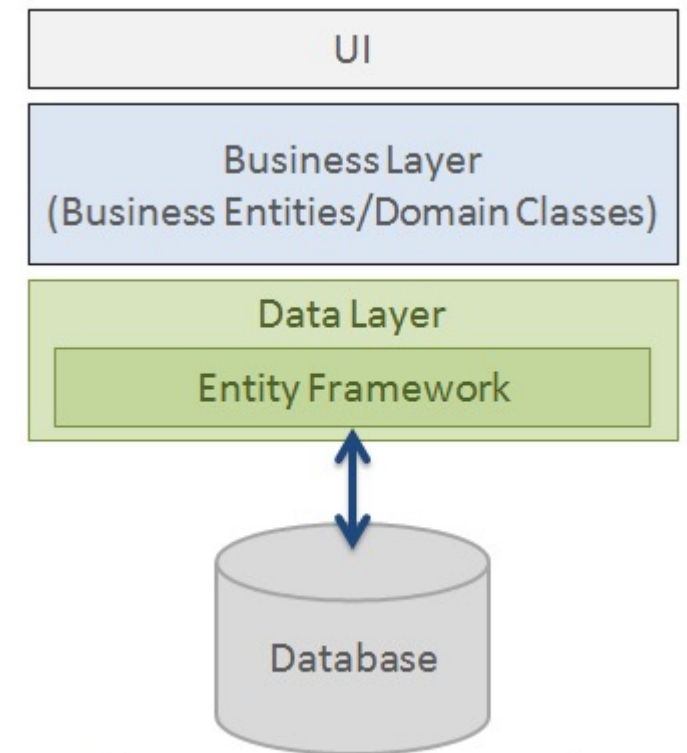


DATABASE MANAGEMENT

- Entity Framework: <https://docs.microsoft.com/en-us/ef/>
- Dapper: <https://dapper-tutorial.net/dapper>

ENTITY FRAMEWORK

- Entity Framework is an open-source ORM framework for .NET applications supported by Microsoft.
- It enables developers to work with data using objects of domain specific classes without focusing on the underlying database tables and columns where this data is stored.
- With the Entity Framework, developers can work at a higher level of abstraction when they deal with data, and can create and maintain data-oriented applications with less code compared with traditional applications.



© EntityFrameworkTutorial.net



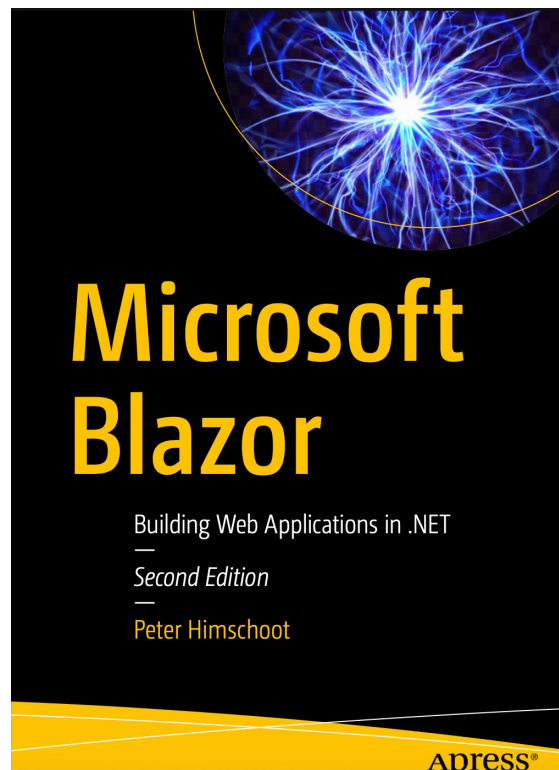
DAPPER

- Dapper is a simple object mapper for .NET and owns the title of **King of Micro ORM** in terms of speed and is virtually as fast as using a raw ADO.NET data reader. An ORM is an Object Relational Mapper, which is responsible for mapping between database and programming language.


DEMO (STARTUP)

- Using Visual Studio Code
 - `dotnet new blazorserver -o BlazorApp --no-https`
 - `cd BlazorApp`
 - `dotnet watch run`
 - Wait for the command to display that it's listening on `http://localhost:5000` and then, open a browser and navigate to that address.
 - https://github.com/brunobmo/Blazor_Course

RECURSOS



- <https://dotnet.microsoft.com/learn/aspnet/blazor-tutorial/intro>
- <https://docs.microsoft.com/en-us/aspnet/core/tutorials/build-a-blazor-app?view=aspnetcore-5.0>
- <https://www.youtube.com/watch?v=8DNgdphLvag&t=2142s>



INTRODUÇÃO AO DESENVOLVIMENTO WEB COM BLAZOR (.NET CORE 5)

INTRODUÇÃO