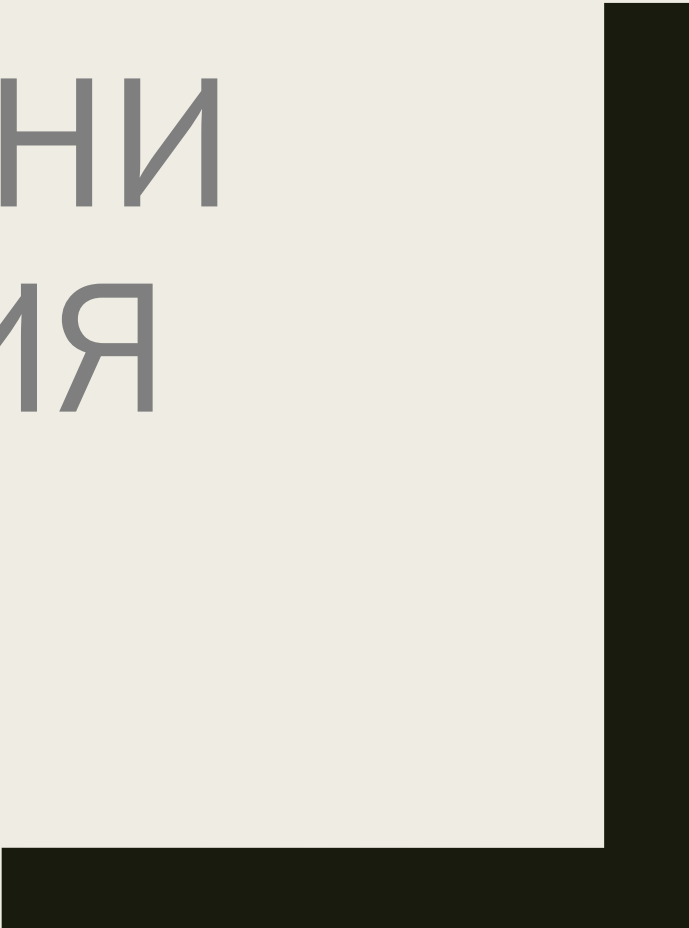




# УЕБ-БАЗИРАНО ПРОГРАМИРАНЕ

Павел Кюркчиев  
Ас. към ПУ „Паисий Хилендарски“  
@pkyurkchiev



# ASP.NET CORE BLAZOR



# Какво представлява .NET Core Blazor?

- Blazor е уеб рамка създадена да изпълнява client-side приложение в браузъра на основата на WebAssembly .NET runtime (Blazor client-side) или server-side на ASP.NET Core (Blazor server-side). Независимо от модела на хостване, приложението и компонентните модели са едни и същи.

# Преимущества на .Net Core Blazor

- Предоставя възможност за писане на код на C# вместо на JavaScript;
- Предоставя възможност за лесна интеграция със съществуващата .Net еко система .Net libraries;
- Споделя логиката на приложението между сървъра и клиента;

- Възползва се от всички предимства предоставени от платформата ASP.Net Core като подобрява производителността, надеждността и сигурността;
- Възможност за разработка както под Windows, macOS и Linux.(Visual Studio и Visual Studio Code);
- Разработен е върху общ набор от езици, рамки и инструменти, които са стабилни, богати на функции и лесни за използване.

# Основа на .Net Core Blazor

- Blazor приложенията представляват съвкупност от компоненти. Компонента в Blazor представлява елемент от UI, като страница, диалогов прозорец или форма за информация.

## Компонентите са .Net класове, вградени в .Net assemblies, които:

- Дефинират гъвкава UI логика за представяне (rendering);
- Обработват потребителски събития;
- Могат да бъдат вложени и да се използват повторно;
- Могат да бъдат разпространявани чрез Razor class libraries или NuGet packages.

# Класове компоненти (components)

- Класове компонентите обикновено са написани във форма от Razor страници с разширение на файловете .razor.
- Компонентите в Blazor доста често реферират към Razor компонентите.



# Компоненти за четене на информация

Input component	Rendered as...
InputText	<input>
InputTextArea	<textarea>
InputSelect	<select>
InputNumber	<input type="number">
InputCheckbox	<input type="checkbox">
InputDate	<input type="date">

# Blazor синтаксис

```
<div>  
  <h1>@Title</h1>  
  
  @ChildContent  
  
  <button @onclick="OnYes">Yes!</button>  
</div>
```

Компонент - *Dialog.razor*

```
@code {  
    [Parameter]  
    public string Title { get; set; }  
  
    [Parameter]  
    public RenderFragment ChildContent { get; set; }  
  
    private void OnYes()  
    {  
        Console.WriteLine("Write to the console in C#! 'Yes' button was selected.");  
    }  
}
```

Компонент - *Dialog.razor*

```
@page "/"
```

```
<h1>Hello, world!</h1>
```

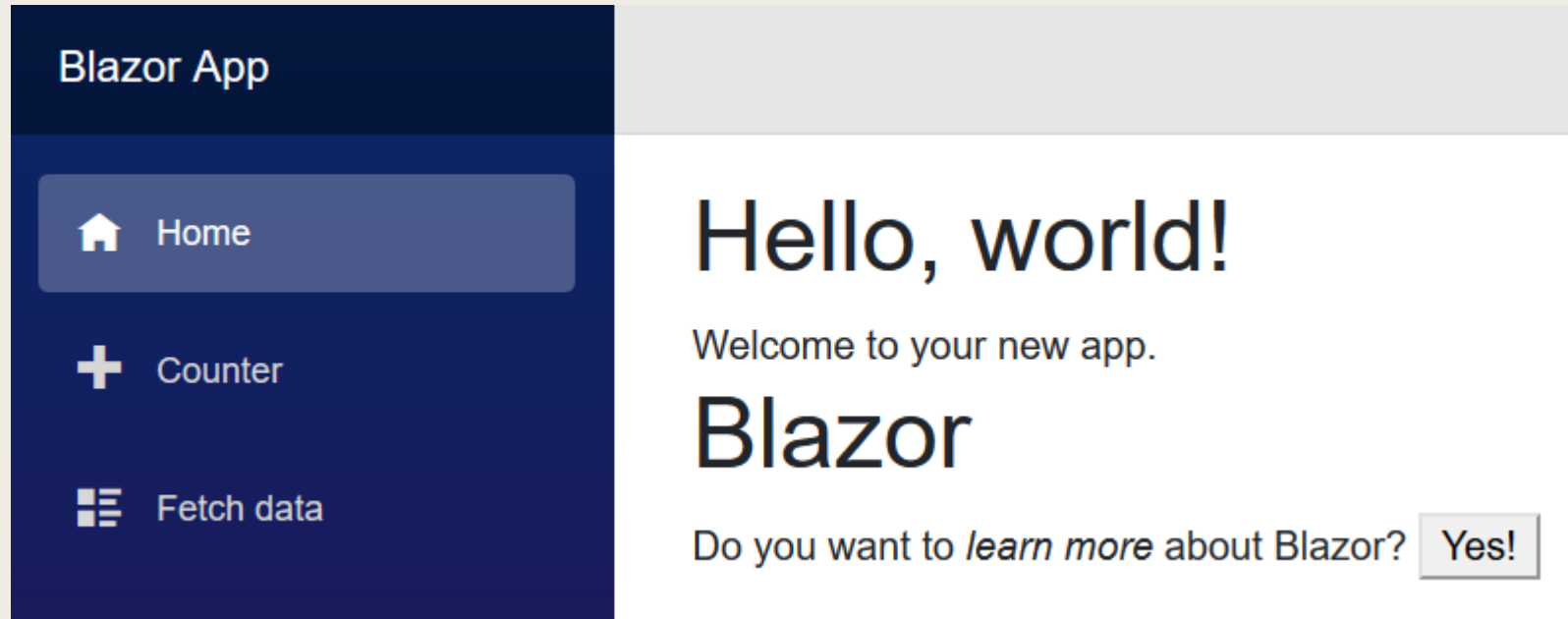
```
Welcome to your new app.
```

```
<Dialog Title="Blazor">
```

```
    Do you want to <i>learn more</i> about Blazor?  
</Dialog>
```

Използване на *Dialog.razor* - *Index.razor*

# Index.razor



# Елементи изграждащи ASP.NET Core

## Blazor

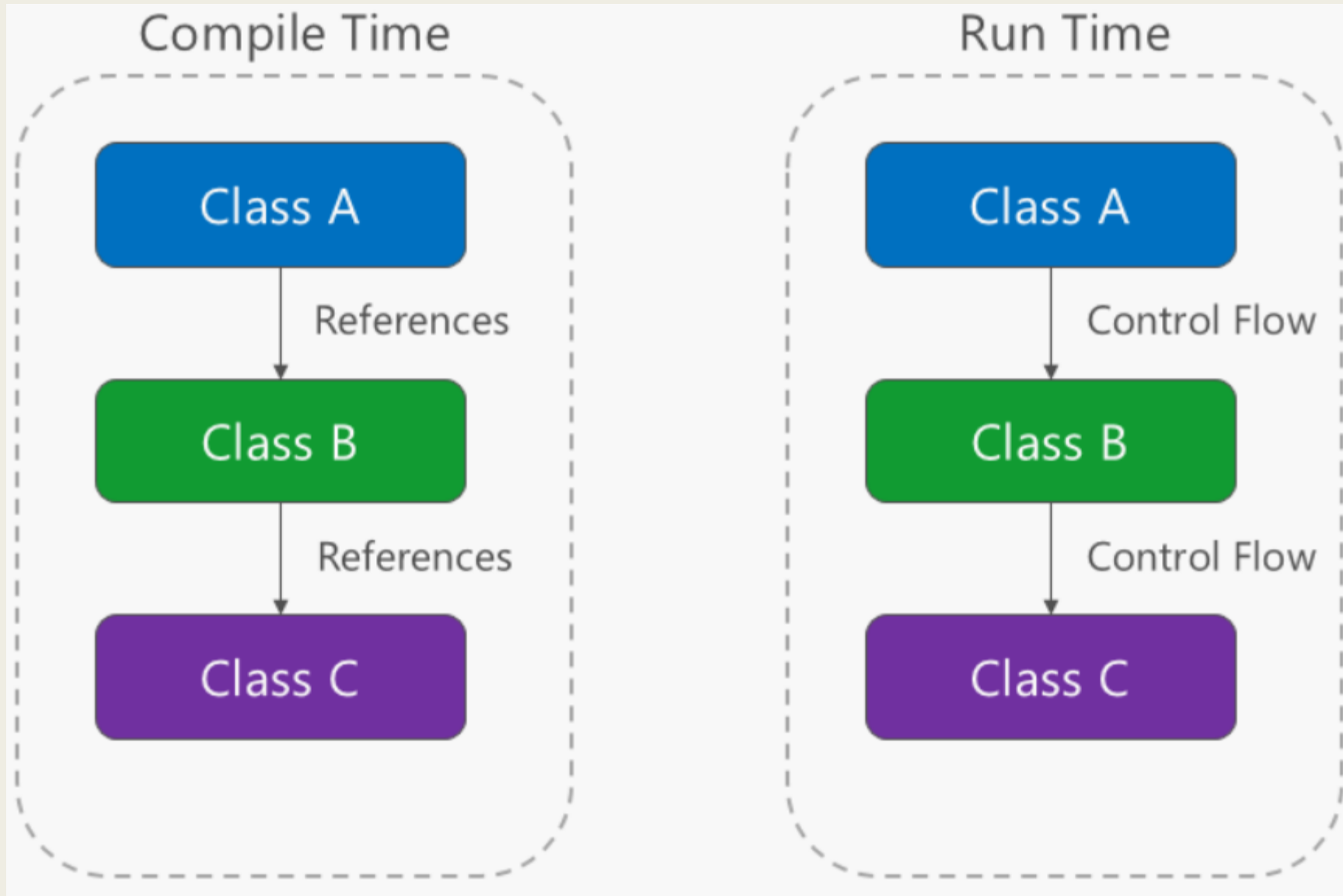
- Razor syntax – беше разгледан в лекция 2
- Dependency injection
- Data binding
- Forms and Validation

# Какво е Dependency injection (DI)?

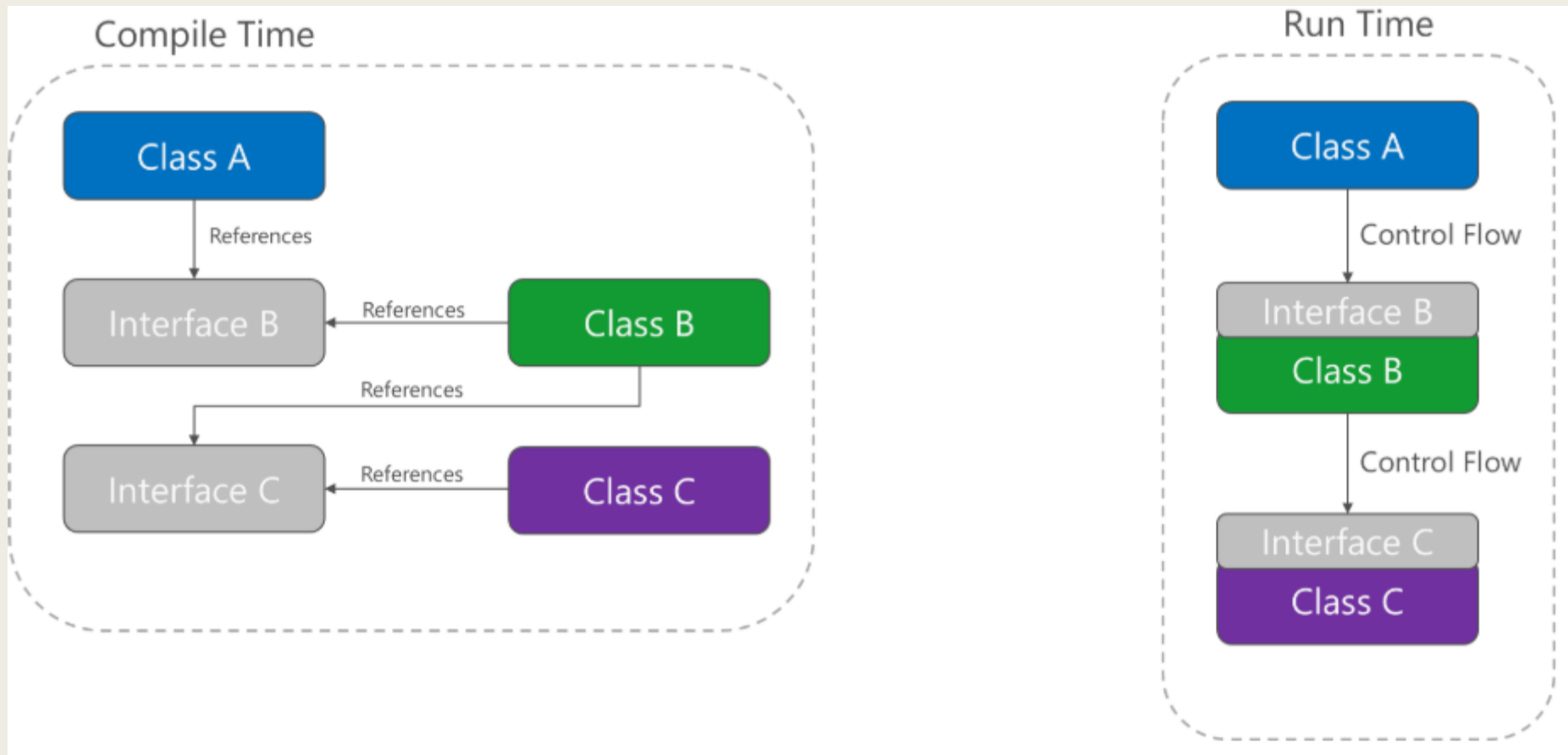
- Dependency injection е техника, при която един обект доставя всички необходими зависимости (необходима бизнес логика) на друг обект. “Dependency” е обект, който може да се използва. Пример за това е услуга. “Injection” се отнася до предаването на зависимост (услуга) към обект (клиент), който би я използвал.
- Dependency injection е възможно само чрез следването на практиките на dependency inversion.



# Direct Dependency Graph



# Inverted Dependency Graph



# Описание на DI

- Описанието на DI се случва в `Startup.ConfigureServices`.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IDataAccess, DataAccess>();
}
```

## Конфигурация на DI

# Жизнен цикъл на Контейнера за услуги

Lifetime	Description
Scoped	Blazor hosting model поддържа Scoped lifetime. Предпочитано е за услуги, които трябва да бъдат изпълнявани сами и ограничени само до текущия потребител.
Singleton	DI създава единична инстанция от услугата. Всички компоненти, изискващи услуга Singleton, получават една и съща инстанция от същата услуга.
Transient	Всеки път, когато компонент получи инстанция от Transient услуга от сервизния контейнер, той получава нов екземпляр от услугата.

# Услуги по подразбиране

Service	Lifetime	Description
HttpClient	Singleton	Предоставя методи за изпращане на HTTP заявки и получаване на HTTP отговори от ресурс, идентифициран от URI. Обърнете внимание, че този екземпляр на HttpClient използва браузъра за обработка на HTTP трафика във фонов режим. HttpClient.BaseAddress автоматично се задава основния URI префикс на приложението.
IJSRuntime	Singleton	Представява инстанция на JavaScript runtime.
NavigationManager	Singleton	Съдържа инструменти за работа с URI и навигация.

Инжектиране на услуги

```
public class ComponentBase : IComponent
{
    // DI works even if using the InjectAttribute in a
    component's base class.
    [Inject]
    protected IDataAccess DataRepository { get; set; }
    ...
}
```

## Инжектиране в Property



```
@page "/customer-list"  
@using Services  
@inject IDataAccess DataRepository
```

## Инжектиране в View

```
public class DataAccess : IDataAccess
{
    // The constructor receives an HttpClient via dependency
    // injection. HttpClient is a default service.
    public DataAccess(HttpClient client)
    {
        ...
    }
}
```

## Инжектиране в услуги

# Предаване на данни (Data biding)

- Предаването на данни е възможно чрез:
  - *Data binding*
  - *Event handling*
  - *Chained bind*

```
<input @bind="CurrentValue" />
```

```
@code {  
    private string CurrentValue { get; set; }  
}
```

## Data binding

```
<button class="btn btn-primary" @onclick="UpdateHeading">  
    Update heading  
</button>
```

```
@code {  
    private void UpdateHeading(MouseEventArgs e)  
    {  
        ...  
    }  
}
```

## Event handling

# Форма и валидиране на данни (Forms and Validation)

- Валидацията на формите се осъществява чрез проверка на ModelState.

# Пълен списък на Data Annotations за .Net Core

- <https://learn.microsoft.com/en-us/dotnet/api/system.componentmodel.dataannotations?view=net-6.0>

```
public class Movie
{
    public int Id { get; set; }

    [Required]
    [StringLength(100)]
    public string Title { get; set; }

    [DataType(DataType.Date)]
    public DateTime ReleaseDate { get; set; }

    [Range(0, 999.99)]
    public decimal Price { get; set; }
}
```

Model



```
public async Task<IActionResult> OnPostAsync()
{
    if (!ModelState.IsValid)
    {
        return Page();
    }

    _context.Movie.Add(Movie);
    await _context.SaveChangesAsync();

    return RedirectToPage("./Index");
}
```

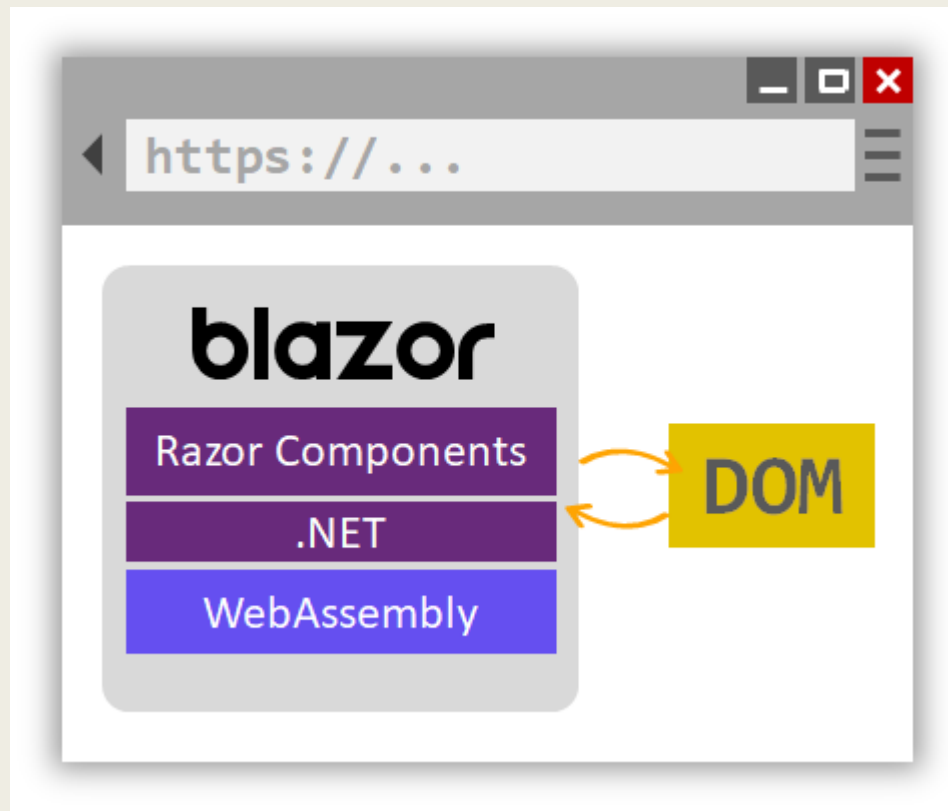
## Проверка на модела

Два режима на работа

# Blazor client-side

- Blazor client-side е single page application(SPA) рамка за разработка на client-side уеб приложения с .Net Core. Blazor client-side е open web standart, който без никакви плъгини или допълнителни код транслации работи с всички съвременни браузъри. Стартирането на .Net код в браузъра е възможно благодарение на WebAssembly.

# Blazor client-side



# Blazor client-side начин на работа

- C# код файл и Razor файл се компилират до .NET assemblies.
- Assemblies и .NET runtime се смъкват в уеб браузъра.
- Blazor client-side бутстрапва .NET runtime и конфигурира runtime-ма да зареди assemblies-та на приложението. Blazor client-side runtime използва JavaScript interop за да манипулира DOM и браузър API заявките.

## Blazor client-side оптимизации

- Не използвания код се премахва от приложението при неговото публикуване от Intermediate Language (IL) Linker.
- HTTP responses се компресират.
- .NET runtime and assemblies се кешират в браузъра.

# Браузър изисквания - Blazor client-side

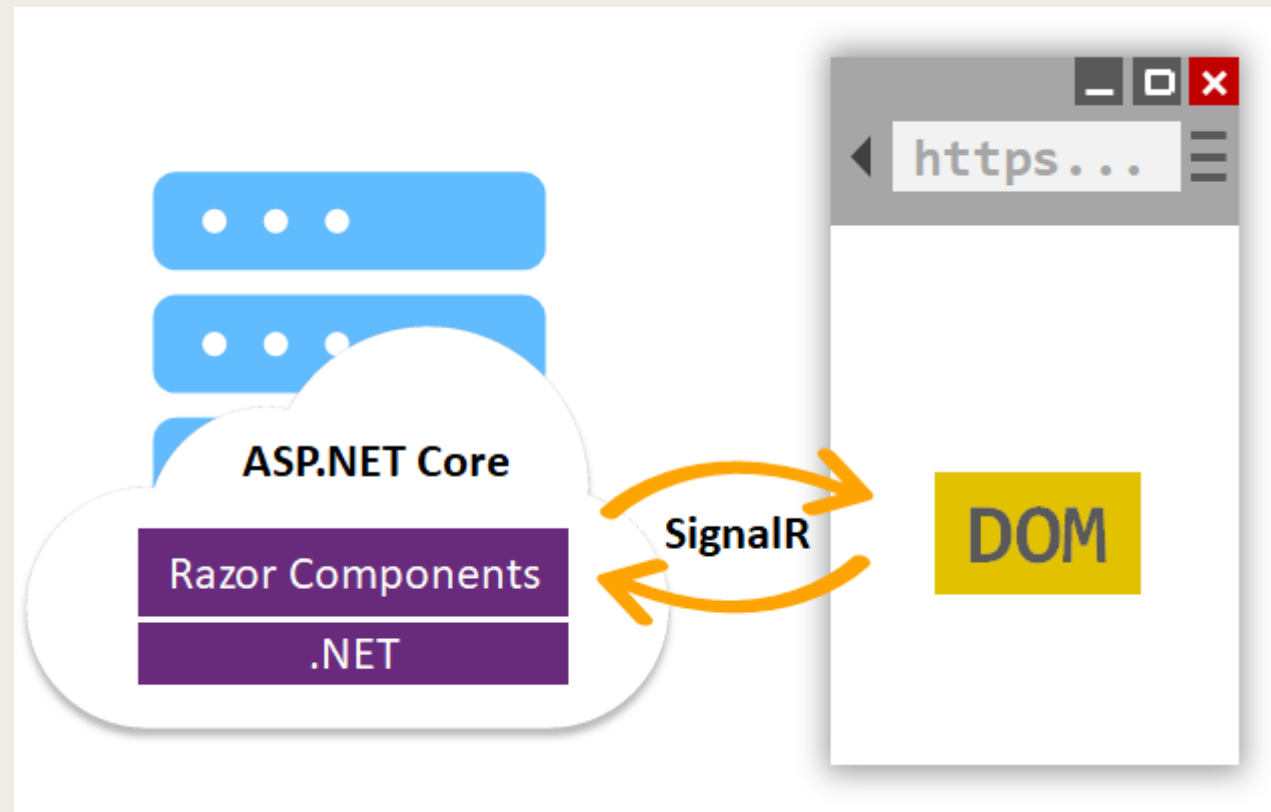
Браузър	Версия
Microsoft Edge	Current
Mozilla Firefox	Current
Google Chrome, including Android	Current
Safari, including iOS	Current
Microsoft Internet Explorer	Not Supported†

# Blazor server-side

- Blazor отделя логиката на изобразяване на компоненти от начина на прилагане на актуализациите на потребителския интерфейс. Blazor server-side предоставя поддръжка за хостинг на Razor компоненти на сървър посредством ASP.NET Core приложение. UI ъпдейтите се осъществяват посредством SignalR връзка.



# Blazor server-side



# Blazor server-side начин на работа

- Runtime-ма обработва изпращането на UI събития от брауъра до сървъра и прилага актуализации на потребителския интерфейс, изпратени от сървъра обратно към брауъра след стартиране на компонентите.
- Връзката, използвана от страна на сървъра на Blazor за комуникация с брауъра, се използва и за обработка на извикванията от JavaScript interop.

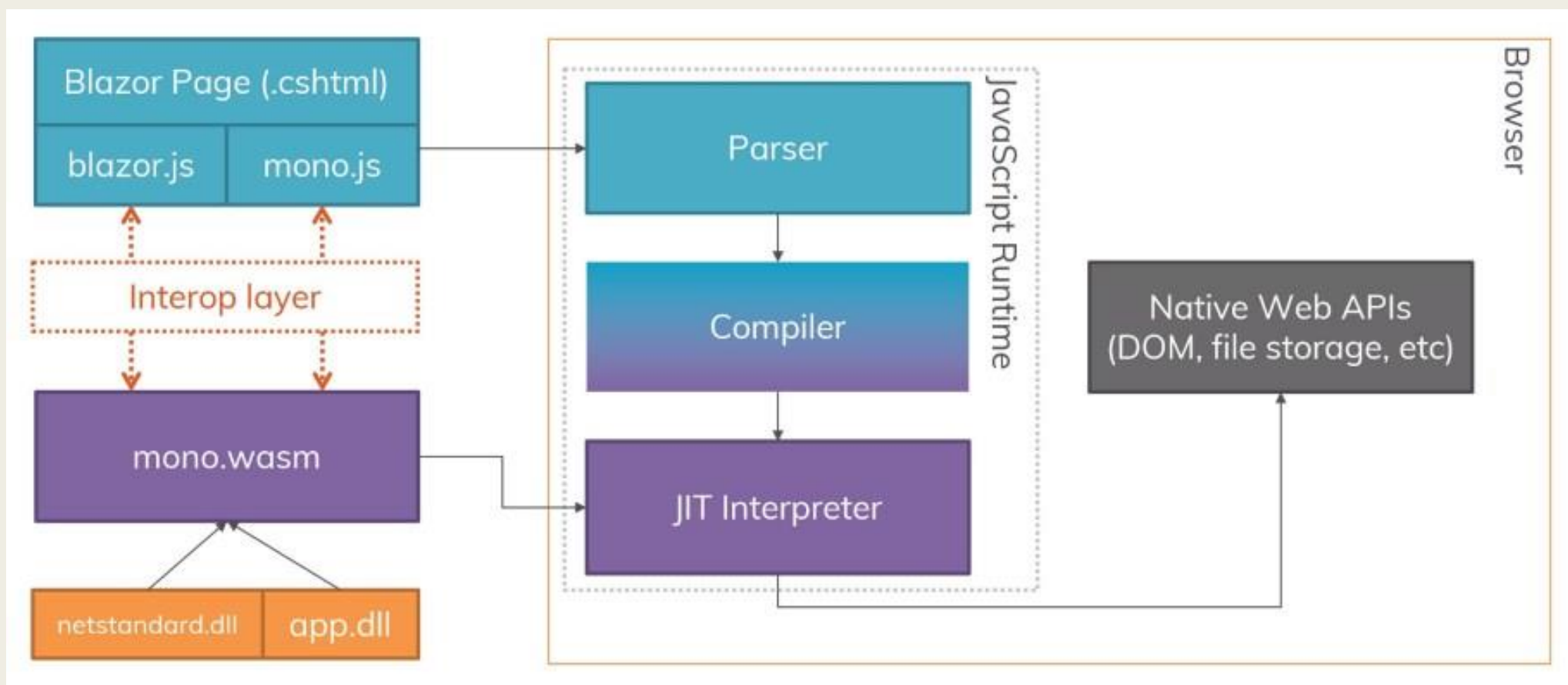
# Браузър изисквания - Blazor server-side

Браузър	Версия
Microsoft Edge	Current
Mozilla Firefox	Current
Google Chrome, including Android	Current
Safari, including iOS	Current
Microsoft Internet Explorer	11†

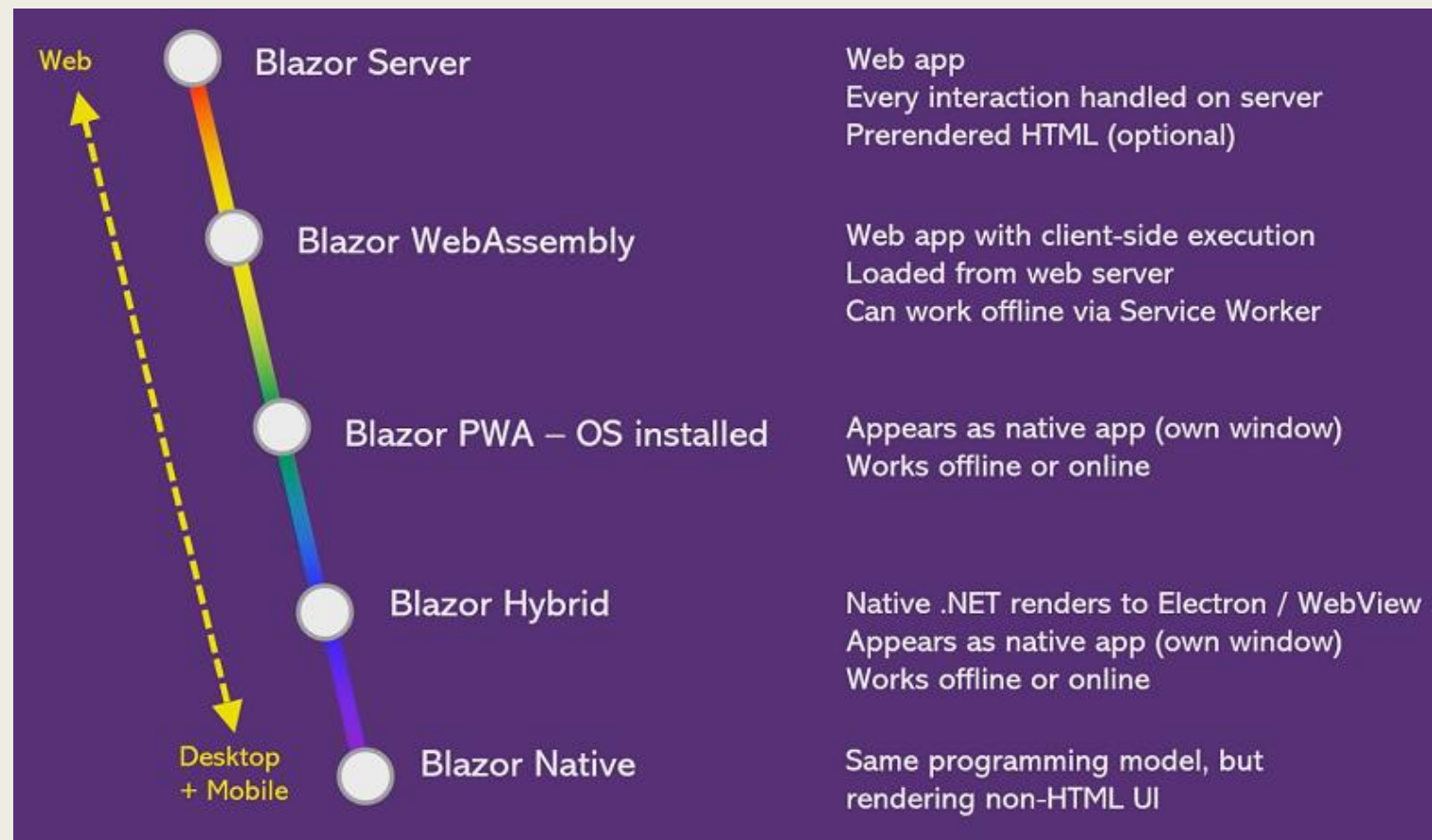
# JavaScript interop

- Тъй като Blazor е създаден на основата на Mono и WebAssembly и не разполага с директен достъп до браузъра и неговото DOM API, то за създаването на заявките се използва JavaScript. Необходимостта от включването на JavaScript в технологичния стек предоставя не само възможност за достъп до DOM, но и подsigурява бъдещата гъвкавост и развитие на продукта.

# Блок диаграма на Blazor & Browser взаимодействието



# Какво да очакваме от Blazor в бъдеще



# DEMO .NET BLAZOR

<https://github.com/pkyurkchiev/web-programming-biel/tree/master/BlazorBaseProject>

ВЪПРОСИ ?

