

Введение в Blazor

Введение в Blazor

ПРИНЦИП РАБОТЫ

Blazor - это платформа для создания интерактивного клиентского веб-интерфейса с помощью .NET

Blazor

- ❑ Создавать многофункциональные интерактивные пользовательские интерфейсы, используя C # вместо JavaScript.
- ❑ Совместное использование серверной и клиентской логики приложений, написанных на .NET.
- ❑ Визуализировать пользовательский интерфейс как HTML и CSS для широкой поддержки браузеров, включая мобильные браузеры.
- ❑ Интегрироваться с современными хостинговыми платформами, такими как Docker.

Blazor WebAssembly

Blazor WebAssembly - это платформа одностраничных приложений (SPA) для создания интерактивных клиентских веб-приложений с помощью .NET.

Blazor WebAssembly

Blazor WebAssembly использует открытые веб-стандарты без плагинов или перекомпиляции кода на другие языки. Blazor WebAssembly работает во всех современных веб-браузерах, включая мобильные браузеры.

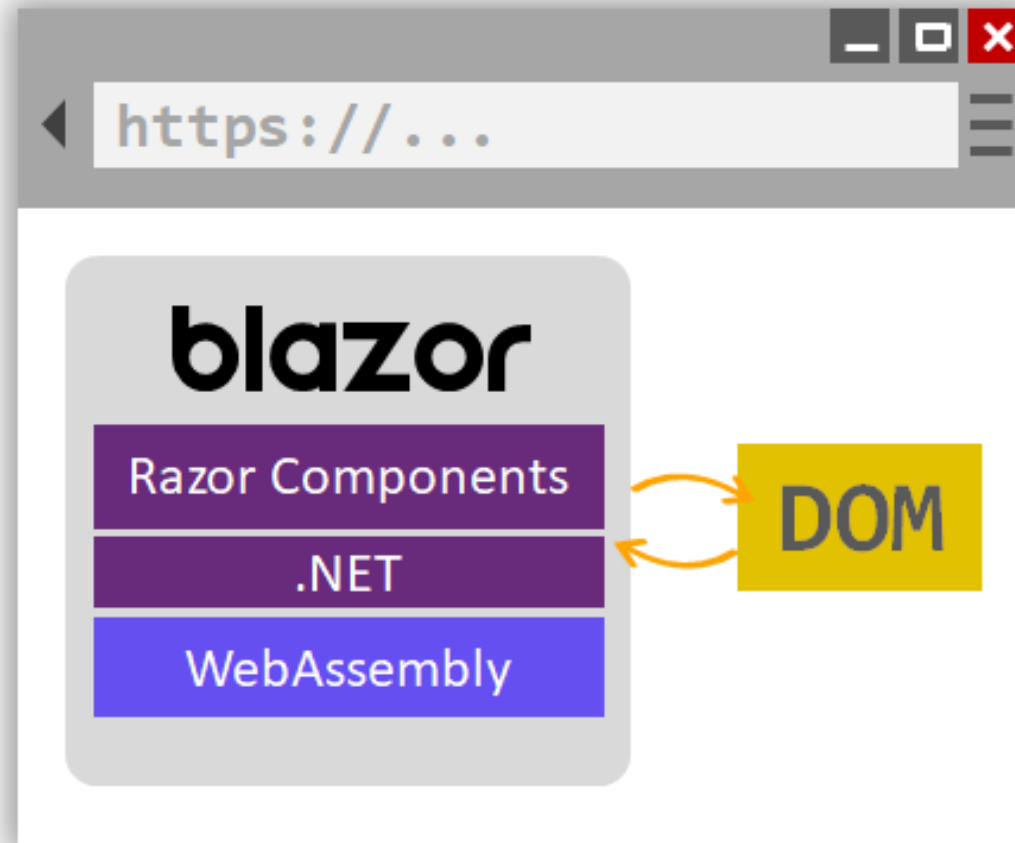
Blazor WebAssembly

Запуск кода .NET внутри веб-браузеров стал возможным благодаря WebAssembly (сокращенно wasm). WebAssembly - это компактный формат байт-кода, оптимизированный для быстрой загрузки и максимальной скорости выполнения.

Blazor WebAssembly

WebAssembly - это открытый веб-стандарт, который поддерживается в веб-браузерах без подключаемых модулей.

Blazor WebAssembly

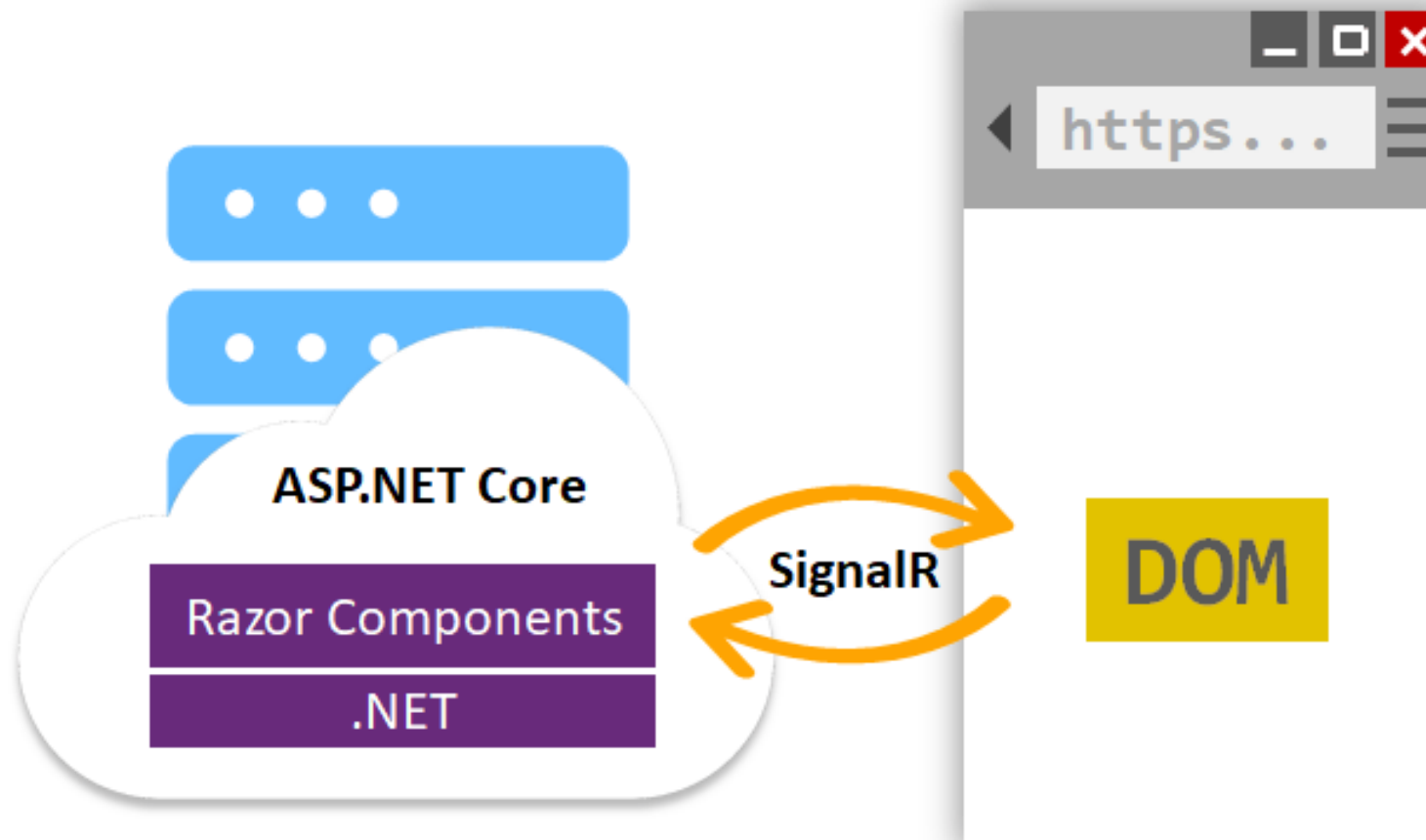


<https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-5.0#blazor-webassembly>

Blazor Server

Blazor отделяет логику отрисовки компонентов от того, как применяются обновления пользовательского интерфейса. Blazor Server обеспечивает поддержку размещения компонентов Razor на сервере в приложении ASP.NET Core. Обновления пользовательского интерфейса обрабатываются через соединение SignalR.

Blazor Server



<https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-5.0#blazor-server>

Введение в Blazor

СТРУКТУРА ПРОЕКТА

Структура проекта

BLAZOR SERVER

Класс Startup

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddRazorPages();
    services.AddServerSideBlazor();
    . . .
}
```

Класс Startup

```
public void Configure(IApplicationBuilder app,  
                     IWebHostEnvironment env)  
{  
    . . .  
    app.UseHttpsRedirection();  
    app.UseStaticFiles();  
    app.UseRouting();  
    app.UseEndpoints(endpoints =>  
    {  
        endpoints.MapBlazorHub();  
        endpoints.MapFallbackToPage("/_Host");  
    });  
}
```

MapBlazorHub - вызывается для установки конечной точки для соединения с браузером в реальном времени. Соединение создается с помощью SignalR.

MapFallbackToPage ("/_Host") - вызывается для настройки корневой страницы приложения (Pages/_Host.cshtml) и включения навигации.

App.razor

App.razor: корневой компонент приложения, который настраивает маршрутизацию на стороне клиента с использованием компонента Router.

Компонент Router перехватывает навигацию браузера и отображает страницу, соответствующую запрошенному адресу.

App.razor

```
<Router AppAssembly="@typeof(Program).Assembly">
  <Found Context="routeData">
    <RouteView RouteData="@routeData"
      DefaultLayout="@typeof(MainLayout)" />
  </Found>
  <NotFound>
    <LayoutView Layout="@typeof(MainLayout)">
      <p>Sorry, there's nothing at this address.</p>
    </LayoutView>
  </NotFound>
</Router>
```

App.razor

Добавление библиотечных компонентов

```
<Router AppAssembly="@typeof(Program).Assembly"  
AdditionalAssemblies =  
"new[] {typeof(StarRating).Assembly }">
```

Папка Pages

Папка Pages: содержит маршрутизируемые компоненты/страницы (.razor), которые составляют приложение Blazor, и корневую страницу Razor приложения Blazor Server (_Host.cshtml).

Маршрут для каждой страницы указывается с помощью директивы @page.

_Host.cshtml

_Host.cshtml (Blazor Server): корневая страница приложения, реализованная как страница Razor:

- Когда первоначально запрашивается какая-либо страница приложения, эта страница создается и возвращается в ответе.
- Загружает файл JavaScript `_framework / blazor.server.js`, который устанавливает соединение SignalR в реальном времени между браузером и сервером.
- На странице «_Host.cshtml» указывается, где будет отображаться корневой компонент приложения (App.razor).

_Host.cshtml

```
<body>  
  <app>  
    <component type="typeof(App)" render-mode="ServerPrerendered" />  
  </app>  
  
  <div id="blazor-error-ui">  
    <environment include="Staging,Production">  
      An error has occurred. This application may no longer respond until reloaded.  
    </environment>  
    <environment include="Development">  
      An unhandled exception has occurred. See browser dev tools for details.  
    </environment>  
    <a href="" class="reload">Reload</a>  
    <a class="dismiss">✕</a>  
  </div>  
  
  <script src="_framework/blazor.server.js"></script>  
</body>
```

MainLayout.razor

@inherits LayoutComponentBase

<div class="sidebar">

<NavMenu />

Компонент Blazor

</div>

<div class="main">

<div class="top-row px-4">

About

</div>

<div class="content px-4">

@Body

</div>

</div>

_Imports.razor

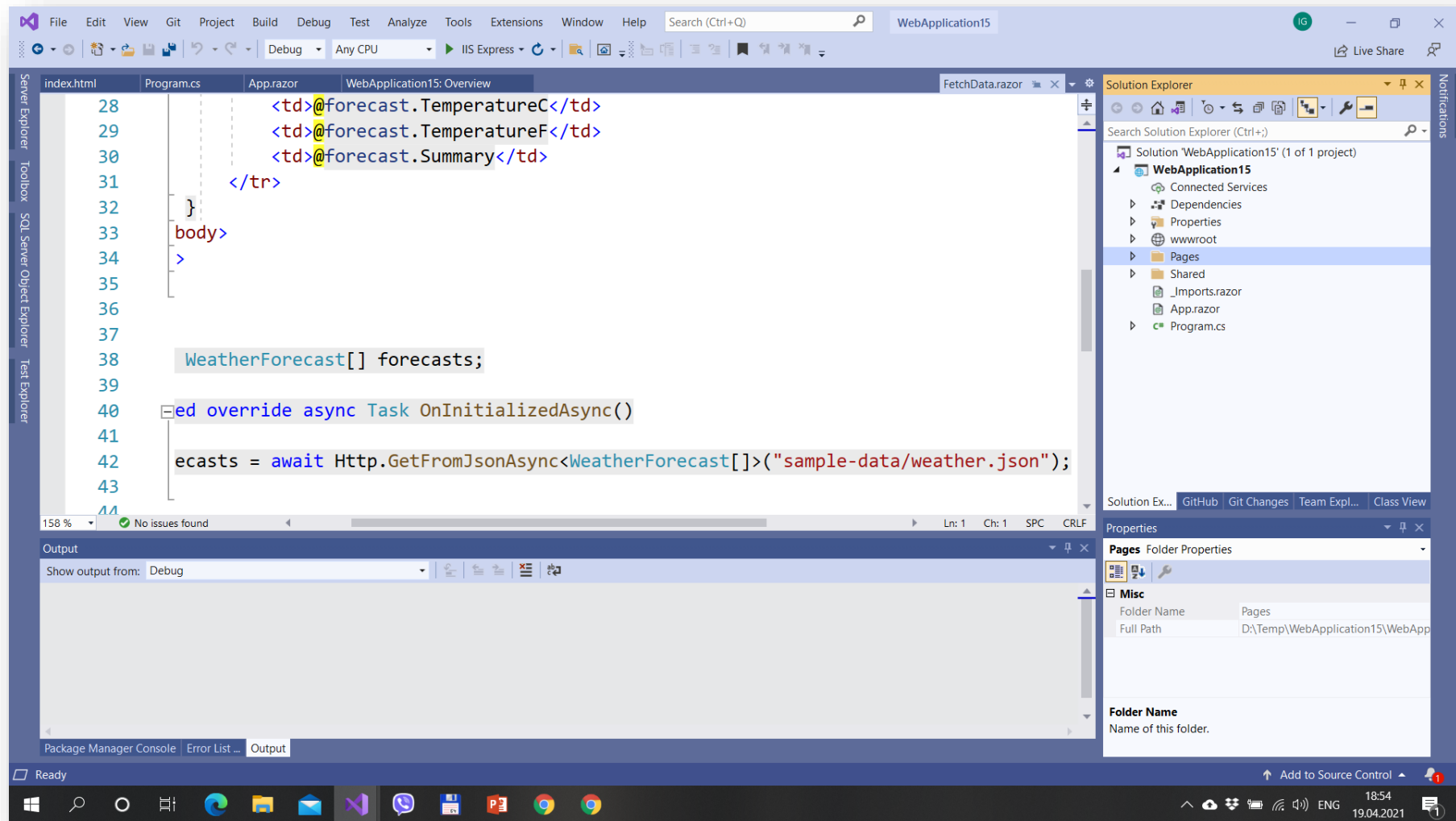
_Imports.razor: включает общие директивы Razor для включения в компоненты приложения (.razor), такие как директивы @using для пространств имен.

_Imports.razor

```
@using System.Net.Http
@using Microsoft.AspNetCore.Authorization
@using Microsoft.AspNetCore.Components.Authorization
@using Microsoft.AspNetCore.Components.Forms
@using Microsoft.AspNetCore.Components.Routing
@using Microsoft.AspNetCore.Components.Web
@using Microsoft.JSInterop
@using BlazorDemoUI
@using BlazorDemoUI.Shared
@using BlazorDemoUI.Data
```

Структура проекта

BLAZOR WEBASSEMBLY



Введение в Blazor

КОМПОНЕНТЫ RAZOR

Компоненты Razor

Приложения Blazor создаются с использованием компонентов.

Компонент - это автономный фрагмент пользовательского интерфейса (например, страница, диалог или форма).

Компоненты Razor

Компоненты реализованы в файлах компонентов Razor (.razor) с использованием комбинации разметки C # и HTML.

Компонент в Blazor формально называется компонентом Razor.

Компоненты Razor

Имя компонента должно начинаться с заглавной буквы.

Например:

MainMenuComponent.razor - правильно,
а mainMenuComponent.razor - неправильно.

Компоненты Razor

Члены класса компонента определены в блоке `@code`.

В блоке `@code` состояние компонента (свойства, поля) задается методами обработки событий или определения логики других компонентов.

Допускается более одного блока `@code`.

Компоненты Razor

```
@page "/"
@inject IRecipesStore RecipesStore
<SearchBox placeholder="Search recipes..." SearchQueryChanged="Search" />
```

```
...
@code {
    IEnumerable<Recipe> recipes;
    protected override async Task OnInitializedAsync()
    {
        recipes = await RecipesStore.GetRecipes();
    }
    async Task Search(string query)
    {
        recipes = await RecipesStore.GetRecipes(query);
    }
}
```

Компоненты Razor

Компоненты, которые создают веб-страницы, обычно находятся в папке «Pages».

Компоненты, не являющиеся страницами, часто помещаются в папку «Shared» или пользовательскую папку, добавляемую в проект.

Компоненты Razor

Для пользовательских папок, содержащих компоненты, добавьте оператор использования в родительский компонент или в файл `_Imports.razor` приложения.

В следующем примере доступны компоненты в папке «Компоненты»:

```
@using BlazorApp.Components
```

Введение в Blazor

ЯЗЫК RAZOR

Язык Razor (Статический контент)

Blazor следует соглашению о том, что приложения ASP.NET Core размещают статические ресурсы в корневой папке проекта (wwwroot).

Язык Razor (Статический контент)

Используйте базовый относительный путь (/) для ссылки на Web-root статического ресурса. В следующем примере logo.png физически находится в папке wwwroot / images:

```

```


Язык Razor (Статический контент)

Компоненты Razor не поддерживают нотацию тильды (~ /).

Язык Razor (Tag-helpers)

Tag-helpers не поддерживаются
в компонентах Razor (файлы .razor).

Чтобы обеспечить функциональность, подобную TagHelper, в Blazor, создается компонент с той же функциональностью, что и TagHelper.

Маршрутизация

Blazor Server интегрирован в ASP.NET Core Endpoint Routing. Приложение ASP.NET Core настроено на прием входящих соединений для интерактивных компонентов в методе `Startup.Configure`

Startup.Configure

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapBlazorHub();
    endpoints.MapFallbackToPage("/_Host");
});
```

Наиболее типичная конфигурация - перенаправлять все запросы на страницу Razor, которая выступает в качестве хоста для серверной части приложения Blazor Server.

По соглашению, страница хоста обычно называется **_Host.cshtml**.

Компонент Router обеспечивает маршрутизацию к каждому компоненту с указанным маршрутом.
Компонент Router появляется в файле App.razor

App.razor

```
<Router AppAssembly="@typeof(Program).Assembly">
  <Found Context="routeData">
    <RouteView RouteData="@routeData"
      DefaultLayout="@typeof(MainLayout)" />
  </Found>
  <NotFound>
    <LayoutView Layout="@typeof(MainLayout)">
      <p>Sorry, there's nothing at this address.</p>
    </LayoutView>
  </NotFound>
</Router>
```


Если используются компоненты из другой сборки (библиотеки компонентов), необходимо указать наличие такой сборки с помощью параметра `AdditionalAssemblies`

AdditionalAssemblies

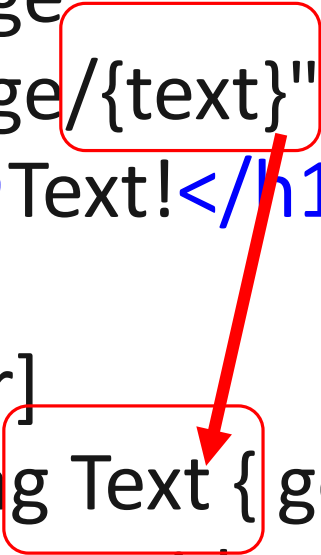
```
<Router AppAssembly="typeof(Program).Assembly"  
AdditionalAssemblies="new[] { typeof(Component1).Assembly }"> ...  
</Router>
```

Параметры маршрута

Маршрутизатор использует параметры маршрута для заполнения соответствующих параметров компонента с тем же именем (без учета регистра)

Параметры маршрута

```
@page "/Manage"  
@page "/Manage/{text}"  
<h1>Blazor is @Text!</h1>  
@code {  
    [Parameter]  
    public string Text { get; set; }  
    protected override void OnInitialized()  
    { Text = Text ?? "fantastic"; }  
}
```



A red arrow points from the `{text}` placeholder in the second `@page` attribute to the `Text` parameter in the `[Parameter]` attribute of the `@code` block, illustrating how the route parameter is mapped to the component parameter.

NavLink

Используйте компонент NavLink вместо элементов гиперссылки HTML (<a>) при создании ссылок навигации.

NavLink

Компонент NavLink ведет себя как элемент `<a>`, за исключением того, что он переключает активный класс CSS в зависимости от того, соответствует ли его `href` текущему URL-адресу.

NavLink

```
<div class="@NavMenuCssClass" @onclick="@ToggleNavMenu">
  <ul class="nav flex-column">
    <li class="nav-item px-3">
      <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
        <span class="oi oi-home" aria-hidden="true"></span> Home
      </NavLink>
    </li>
    <li class="nav-item px-3">
      <NavLink class="nav-link" href="MyComponent" Match="NavLinkMatch.Prefix">
        <span class="oi oi-plus" aria-hidden="true"></span> My Component
      </NavLink>
    </li>
  </ul>
</div>
```

NavigationManager

NavigationManager используется для работы с URI и навигацией в коде C #.

NavigationManager

```
@page "/demo"
```

```
@inject NavigationManager NavigationManager
```

```
<h1>Navigate in Code Example</h1>
```

```
<button class="btn btn-primary" @onclick="NavigateToAdminComponent">
```

```
    Go to admin component
```

```
</button>
```

```
@code {
```

```
    private void NavigateToAdminComponent()
```

```
    {
```

```
        NavigationManager.NavigateTo("admin");
```

```
    }
```

```
}
```

Компоненты

ПАРАМЕТРЫ КОМПОНЕНТОВ

Параметры маршрута

Компоненты могут получать параметры маршрута из шаблона маршрута, указанного в директиве @page.

Параметры маршрута

```
@page "/counter"
```

```
@page "/counter/{CurrentCount:int}"
```

```
<h1>Counter</h1>
```

```
<p>Current count: @currentCount</p>
```

```
<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
```

```
@code {
```

```
[Parameter]
```

```
public int CurrentCount { get; set; } = 0;
```

```
private int currentCount;
```

```
protected override void OnParameterSet()
```

```
{
```

```
    currentCount = CurrentCount;
```

```
}
```

```
...
```

```
}
```

Параметры маршрута

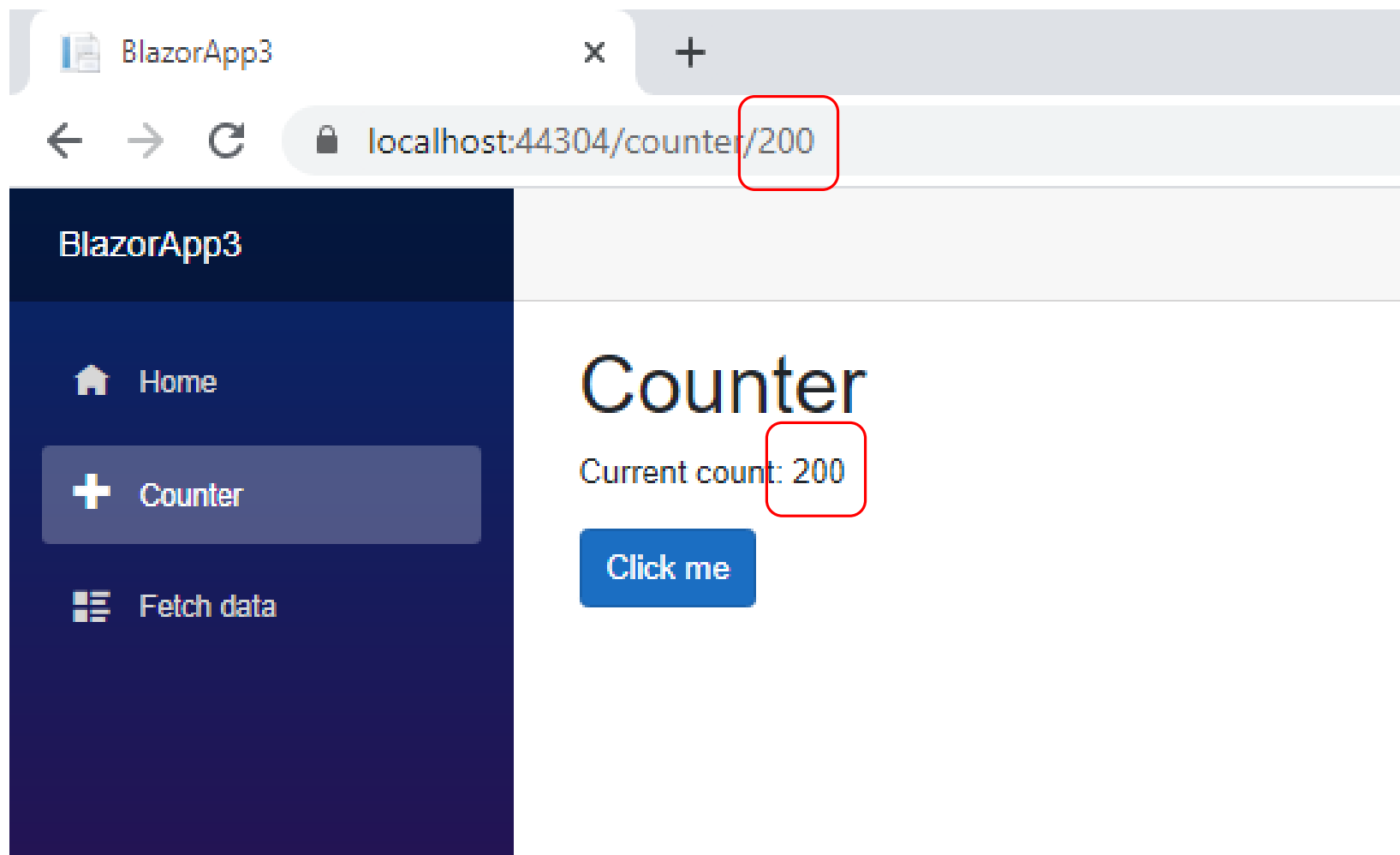
Необязательные параметры ***не поддерживаются***, поэтому в предыдущем примере применяются две директивы @page.

Первая разрешает навигацию к компоненту без параметра. Вторая директива @page получает параметр маршрута {CurrentCount:int} и присваивает значение свойству CurrentCount.

Параметры маршрута

Не создавайте компоненты, которые перезаписывают свои собственные параметры компонентов, вместо этого используйте приватное поле.

Параметры маршрута



Параметры компонента

Компоненты могут иметь параметры, которые определяются с помощью открытых свойств класса компонентов с атрибутом [Parameter].

Параметры компонента

```
<div class="panel panel-default">  
  <div class="panel-heading">@Title</div>  
  <div class="panel-body">@ChildContent</div>  
  <button class="btn btn-primary" @onclick="OnClickCallback">  
    Trigger a Parent component method  
  </button>  
</div>
```

```
@code {  
  [Parameter]  
  public string Title { get; set; }  
  [Parameter]  
  public RenderFragment ChildContent { get; set; }  
  [Parameter]  
  public EventCallback<MouseEventArgs> OnClickCallback { get; set; }  
}
```

Параметры компонента

```
@page "/"
```

```
@inject IJSRuntime jsRuntime
```

```
<ParamsDemo Title="Заголовок из родительского компонента"
```

```
    OnClickCallback="@ShowMessage">
```

Содержимое передано из родительского компонента.

```
</ParamsDemo>
```

```
@code{
```

```
    private async Task ShowMessage()
```

```
    {
```

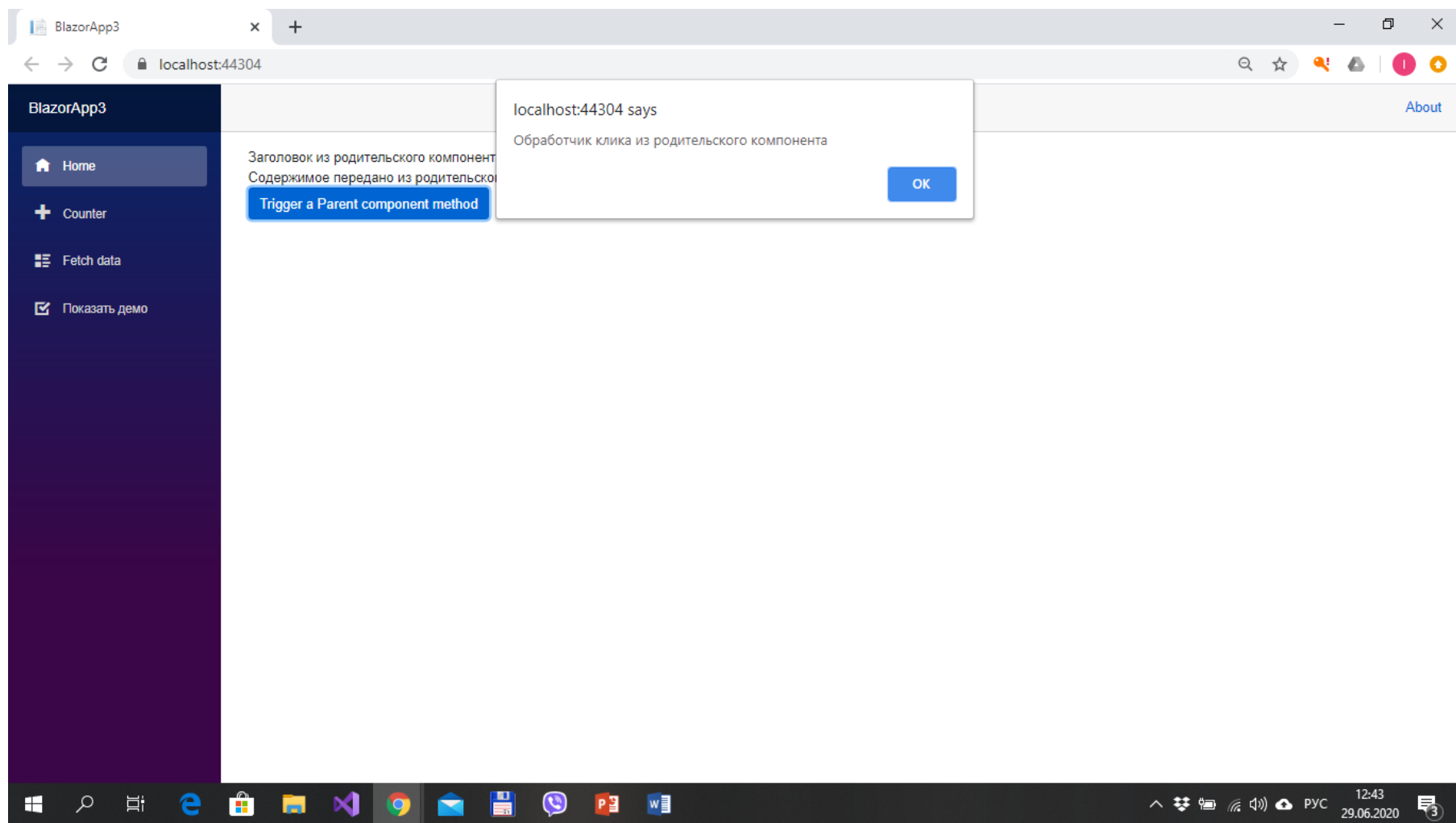
```
        var msg = "Обработчик клика из родительского компонента";
```

```
        await jsRuntime.InvokeAsync<string>("alert", msg);
```

```
    }
```

```
}
```

Параметры компонента



Встроенные компоненты

EDITFORM

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

Form

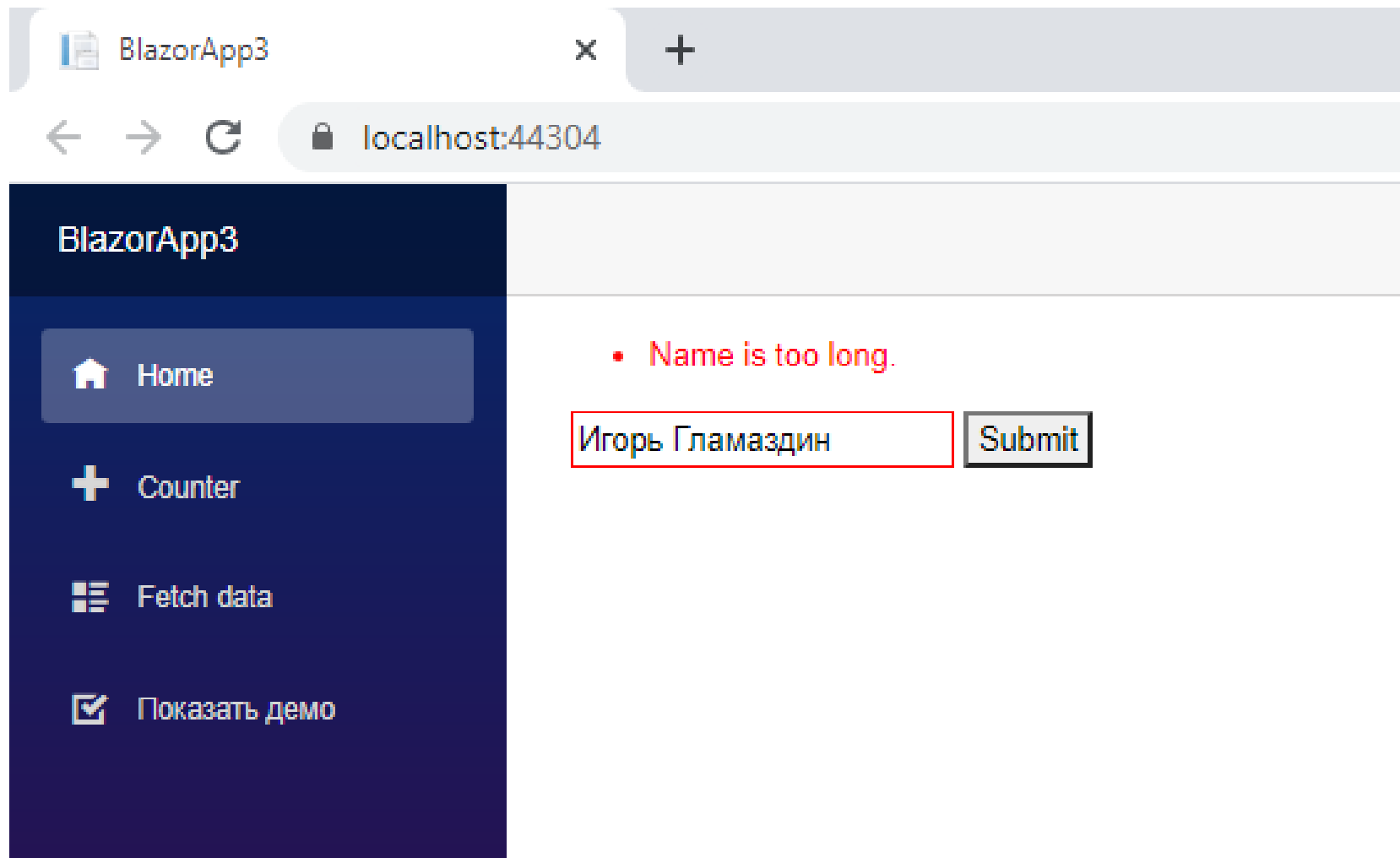
```
<EditForm Model="@exampleModel" OnValidSubmit="HandleValidSubmit">  
    <DataAnnotationsValidator />  
    <ValidationSummary />  
  
    <InputText id="name" @bind-Value="exampleModel.Name" />  
  
    <button type="submit">Submit</button>  
</EditForm>
```

```
@code {  
    private ExampleModel exampleModel = new ExampleModel();
```

```
    private void HandleValidSubmit()
```

```
{  
    ...  
}
```

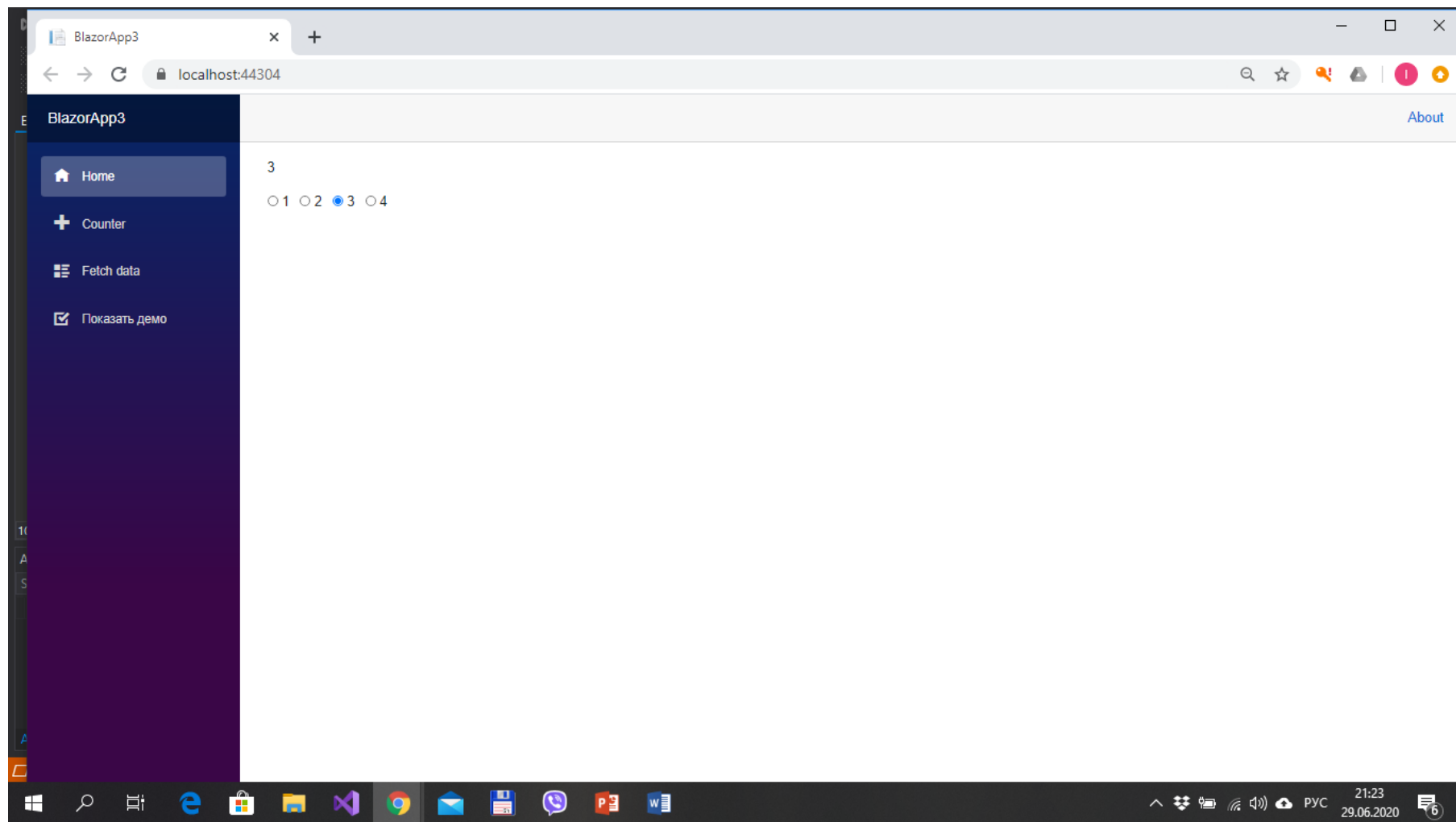
```
public class ExampleModel
{
    [Required]
    [StringLength(10, ErrorMessage = "Name is too long.")]
    public string Name { get; set; }
}
```



Радио - кнопки

Радио-кнопки

```
<p>@RadioValue</p>
  @for(var i=1; i<5;i++)
  {
    <label class="mr-2">
      <input type="radio" name="rButton" value="@i" @onchange="OnChange" /> @i
    </label>
  }
@code {
private int RadioValue=1;
private void OnChange(ChangeEventArgs args)
{
  BindConverter.TryConvertToInt(args.Value,
                                System.Globalization.CultureInfo.CurrentCulture,
                                out RadioValue);
}
```



Привязка данных

Привязка данных

Компоненты Razor предоставляют функции привязки данных через атрибут HTML-элемента `@bind` со значением поля, свойства или выражения Razor.

Привязка данных

```
<input type="text" @bind="TextToInput"/>
```

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

Привязка данных

Значение свойства `TextInput` обновляется по событию **`onchange`**.

Для обновления по другому событию используется атрибут **`@bind:event`** с параметром события.

Привязка данных

```
<input type="text" @bind="TextToInput" @bind:event="oninput"/>
```

```
@code{  
    string textToInput;  
    private string TextToInput {  
        get { return textToInput; }  
        set {  
            textToInput = value;  
            var query = @$"SELECT * FROM Cars WHERE Name LIKE {textToInput}";  
            ...  
        }  
    }  
}
```


Привязка данных

Если нужно привязать атрибут элемента, отличный от value, используется синтаксис

@bind-{ATTRIBUTE} с @bind-{ATTRIBUTE}:event

bind:after

```
[HttpPost]
public IEnumerable<WeatherForecast> Post([FromBody] string searchSummary)
{
    var forecasts = Enumerable.Range(1, 50).Select(index => new
WeatherForecast
    {
        Date = DateOnly.FromDateTime(DateTime.Now.AddDays(index)),
        TemperatureC = Random.Shared.Next(-20, 55),
        Summary = Summaries[Random.Shared.Next(Summaries.Length)]
    });

    if(!string.IsNullOrEmpty(searchSummary))
    {
        return forecasts.Where(
            f=>f.Summary!.ToLower().Contains(searchSummary.ToLower()))
            .ToArray();
    }

    return forecasts.ToArray();
}
```

bind:after

```
<input @bind="@searchSummary" />
<button @onclick="Search" >Search</button>
```

...

```
@code {
    private WeatherForecast[]? forecasts;
    private string searchSummary = string.Empty;

    . . .

    private async Task Search()
    {
        var result = await Http.PostAsJsonAsync("WeatherForecast", searchSummary);
        forecasts = await result
                                .Content
                                .ReadFromJsonAsync<WeatherForecast[]>();
    }
}
```

bind:after

```
<input @bind="@searchSummary" @bind:after="Search" />...
```

```
@code {  
    private WeatherForecast[]? forecasts;  
    private string searchSummary = string.Empty;  
  
    . . .  
  
    private async Task Search()  
    {  
        var result = await Http.PostAsJsonAsync("WeatherForecast", searchSummary);  
        forecasts = await result  
            .Content  
            .ReadFromJsonAsync<WeatherForecast[]>();  
    }  
}
```

bind:after

```
<input @bind="@searchSummary"  
      @bind:event="oninput"  
      @bind:after="Search" />
```

Привязка параметров от родительского к дочернему компоненту

Дочерний компонент:

```
<p>Age: @Age</p>
```

```
@code{
```

```
[Parameter]
```

```
public int Age { get; set; }
```

```
[Parameter]
```

```
public EventCallback<int> AgeChanged { get; set; }
```

```
...
```

```
}
```

Привязка параметров от родительского к дочернему компоненту

Родительский компонент:

```
<ChildComponent @bind-Age="ParentAge" />
```

```
@code{
```

```
[Parameter]
```

```
public int ParentAge { get; set; }
```

```
}
```

Привязка параметров от дочернего к родительскому компоненту

Password:

```
<input @oninput="OnPasswordChanged"
      required
      type="password"
      value="@Password" />
```

```
@code {
  [Parameter]
  public string Password { get; set; }

  [Parameter]
  public EventCallback<string> PasswordChanged { get; set; }

  private Task OnPasswordChanged(ChangeEventArgs e)
  {
    Password = e.Value.ToString();
    return PasswordChanged.InvokeAsync(Password);
  }
}
```


Привязка параметров от дочернего к родительскому компоненту

Родительский компонент

```
<ChildComp @bind-Password="password" />
```

```
@code {  
    private string password;  
}
```

bind:set, bind:get

Компонент SearchSummary

```
<input @bind:event="oninput"
      @bind:set="ValueSet" @bind:get="SearchValue" />

@code {
    [Parameter]
    public string SearchValue { get; set; } = String.Empty;
    [Parameter]
    public EventCallback<string> SearchValueChanged { get; set; }

    public async Task ValueSet(string data)
    {
        await SearchValueChanged.InvokeAsync(data);
    }
}
```

bind:set, bind:get

```
<SearchSummary  
  @bind-SearchValue="searchSummary"  
  @bind-SearchValue:after="Search" />
```

Бизнес-логика в отдельном сервисе

```
public class CalcService
{
    public int Result{ get; set;} =0;
    public void Increment()
    {
        Result++;
    }
}
```

```
builder.Services.AddScoped<CalcService>();
```

Компонент CounterDisplay

```
@inject CalcService Calc
```

```
<p role="status">Current count: @Calc.Result</p>
```

```
@code {
```

```
}
```

```
@page "/counter"
@Inject CalcService Calc

<PageTitle>Counter</PageTitle>

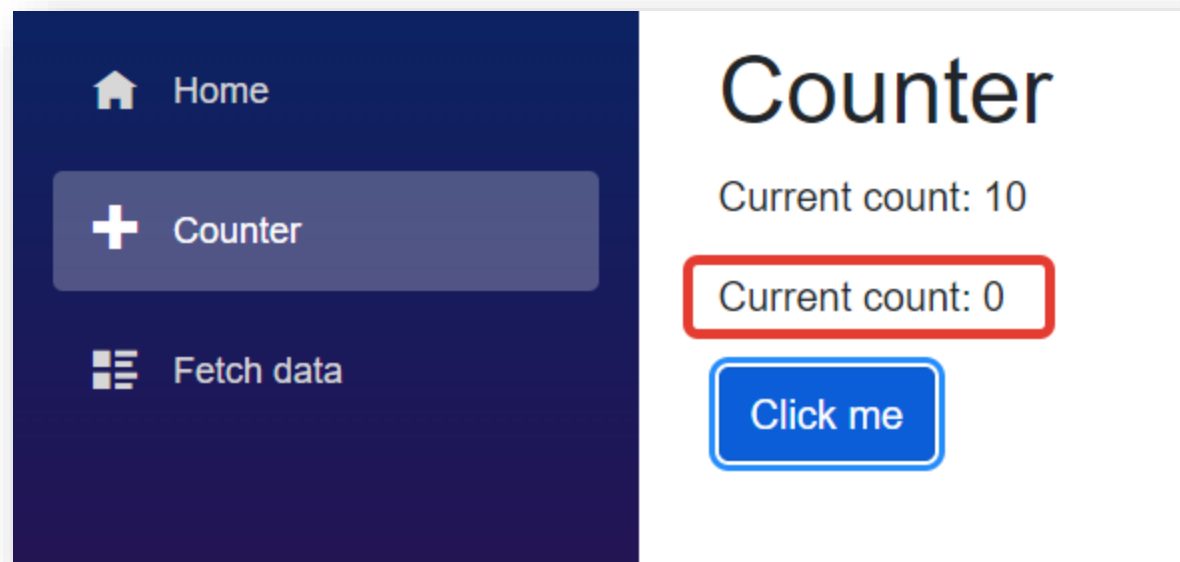
<h1>Counter</h1>

<p role="status">Current count: @Calc.Result</p>
<CounterDisplay />

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>

@code {
    private int currentCount = 0;

    private void IncrementCount()
    {
        Calc.Increment();
    }
}
```

```
public class CalcService
{
    int _result = 0;
    public int Result
    {
        get => _result;
        set
        {
            if (_result == value) { return; }
            _result = value;
            ResultChanged?.Invoke(this, null);
        }
    }
    public void Increment()
    {
        Result++;
    }

    public event EventHandler? ResultChanged;
}
```

```
@inject CalcService Calc
```

```
<p role="status">Current count: @Calc.Result</p>
```

```
@code {  
    protected override void OnInitialized()  
    {  
        Calc.ResultChanged += (o,e)=>StateHasChanged();  
    }  
}
```

Обработка событий

Обработка событий

Компоненты Razor обеспечивают функции обработки событий.

Для атрибута HTML-элемента с именем `@on {EVENT}` (например, `@onclick`) с типизированным делегатом значением компонент Razor обрабатывает значение атрибута как обработчик события.

Обработка событий

```
<button class="btn btn-primary" @onclick="DoSomethingHandler">  
    Do something  
</button>
```

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

Обработка событий

Обработчик события может быть также асинхронным

```
<button class="btn btn-primary" @onclick="DoSomethingHandler">  
    Do something  
</button>  
@code {  
    private async Task DoSomethingHandler(MouseEventArgs e)  
    {  
        ...  
    }  
}
```

Аргументы событий

Clipboard [ClipboardEventArgs](#) oncut, oncopy, onpaste

Drag [DragEventArgs](#) ondrag, ondragstart, ondragenter, ondragleave, ondragover, ondrop, ondragend

[DataTransfer](#) и [DataTransferItem](#) содержат данные перетаскиваемого элемента.

Error [ErrorEventArgs](#) onerror

Аргументы событий

Event [EventArgs](#) General

onactivate, onbeforeactivate, onbeforedeactivate, ondeactivate, onfullscreenchange, onfullscreenerror, onloadeddata, onloadedmetadata, onpointerlockchange, onpointerlockerror, onreadystatechange, onscroll

Clipboard

onbeforecut, onbeforecopy, onbeforepaste

Input

oninvalid, onreset, onselect, onselectionchange, onselectstart, [OnSubmit](#)

Media

oncanplay, oncanplaythrough, oncuechange, ondurationchange, onemptied, onended, onpause, onplay, onplaying, onratechange, onseeked, onseeking, onstalled, onstop, onsuspend, ontimeupdate, onvolumechange, onwaiting

[EventHandlers](#) holds attributes to configure the mappings between event names and event argument types.

Аргументы событий


Focus	<u>FocusEventArgs</u>	onfocus, onblur, onfocusin, onfocusout Doesn't include support for relatedTarget.
Input	<u>ChangeEventArgs</u>	onchange, oninput
Keyboard	<u>KeyboardEventArgs</u>	onkeydown, onkeypress, onkeyup
Mouse	<u>MouseEventArgs</u>	onclick, oncontextmenu, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout

Аргументы событий

Mouse pointer	PointerEventArgs	onpointerdown, onpointerup, onpointercancel, onpointermove, onpointerover, onpointerout, onpointerenter, onpointerleave, ongotpointercapture, onlostpointercapture
Mouse wheel	WheelEventArgs	onwheel, onmousewheel
Progress	ProgressEventArgs	onabort, onload, onloadend, onloadstart, onprogress, ontimeout
Touch	TouchEventArgs	ontouchstart, ontouchend, ontouchmove, ontouchenter, ontouchleave, ontouchcancel TouchPoint представляет одну контактную точку тач-пад.

Передача дополнительных параметров

```
@for (var i = 1; i < 4; i++)  
{  
    var buttonNumber = i;  
    <button class="btn btn-primary"  
        @onclick="@ (e => DoSomething(e, buttonNumber))">  
        Button @i  
    </button>  
}  
@code {  
    private void DoSomething(MouseEventArgs e, int buttonNumber)  
    {  
        ...  
    }  
}
```



EventCallback

EventCallback

Распространенный сценарий с вложенными компонентами - это возможность запустить метод родительского компонента, когда происходит событие дочернего компонента.

EventCallback

Чаще всего обрабатывается событие **onclick**, происходящее в дочернем компоненте

EventCallback

Для передачи события между компонентами, используется событие EventCallback.

Родительский компонент может назначить метод обратного вызова для EventCallback дочернего компонента.

EventCallback

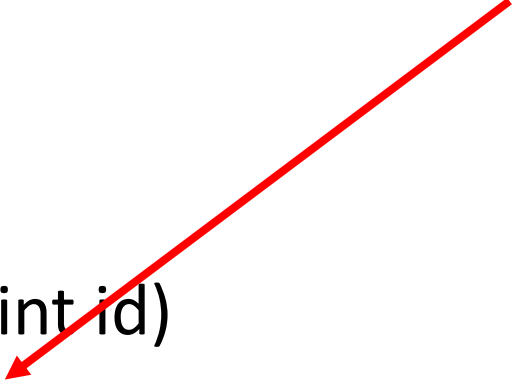
```
@foreach(var dish in Dishes)
{
    <button type="button" class="list-group-item list-group-item-action"
        @onclick="@ (e=>Selected(e, dish.DishId))">
        @dish.DishName
    </button>
}
```

```
[Parameter]
public EventCallback<int> SelectedIdChanged { get; set; }

private void Selected(MouseEventArgs e, int id)
{
    SelectedIdChanged.InvokeAsync(id);
}
```

```
<DishesList @bind-Dishes="dishes"  
             SelectedIdChanged="ShowDetails">  
</DishesList>
```

```
private async Task ShowDetails(int id)  
{  
    ...  
}
```



Запрос к API-сервисам

Запрос к API-сервисам

BLAZOR SERVER

Приложения Blazor Server вызывают веб-интерфейсы API с помощью экземпляров HttpClient, обычно созданных с помощью IHttpClientFactory.

```
...  
builder.services.AddHttpClient();  
...
```

@page "/apidemo"

@inject IHttpConnectionFactory clientFactory


```

string apiBaseAddress = "https://localhost:44310/Api/Dishes";
[Parameter]
public DishViewModel SelectedDish { get; set; }
protected override async Task OnInitializedAsync()
{
    var client = clientFactory.CreateClient();
    var response = await client.GetAsync(apiBaseAddress);
    if (response.IsSuccessStatusCode)
    {
        using var responseStream = await response.Content.ReadAsStreamAsync();
        dishes = await JsonSerializer
            .DeserializeAsync<IEnumerable<DishListViewModel>>(responseStream,
                new JsonSerializerOptions { PropertyNameCaseInsensitive=true
    });
    }
}

```

```
var request = new HttpRequestMessage(HttpMethod.Get,  
                                     apiBaseAddress);  
var client = clientFactory.CreateClient();  
var response = await client.SendAsync(request);
```

Запрос к API-сервисам

BLAZOR WEBASSEMBLY

Приложения Blazor WebAssembly вызывают веб-API с помощью предварительно настроенного сервиса HttpClient.

```

public class Program
{
    public static async Task Main(string[] args)
    {
        var builder = WebAssemblyHostBuilder.CreateDefault(args);
        builder.RootComponents.Add<App>("#app");

        builder.Services.AddScoped(sp =>
            new HttpClient {
                BaseAddress
Uri(builder.HostEnvironment.BaseAddress) }); = new
        await builder.Build().RunAsync();
    }
}

```

@page "/apidemo"

@inject HttpClient client

```
dishes = await client  
    .GetFromJsonAsync<IEnumerable<ListViewModel>>(apiBaseAddress);
```