Laboratorium 4 - Powtórka

Blazor - Aplikacje w .NET 8

Materiały na kolokwium

7 grudnia 2025

Spis treści

1 Teoria

1.1 Dlaczego Blazor (.NET 8 "Blazor Web App")?

Kluczowe zalety

Jeden szablon = wiele trybów renderowania
Blazor .NET 8 wprowadza koncepcję *progressive enhancement*, która pozwala na:

- Używanie jednego szablonu projektu dla różnych scenariuszy
- Elastyczny wybór trybu renderowania na poziomie komponentu
- Stopniowe dodawanie interaktywności tam, gdzie jest potrzebna

1.1.1 Zalety podejścia

1. **SEO + szybkość pierwszego ładowania (SSR)**
   Server-Side Rendering zapewnia:
   - Szybkie wyświetlenie treści (First Contentful Paint)
   - Indeksowanie przez wyszukiwarki bez JavaScript
   - Lepsze wskaźniki Core Web Vitals
2. **Możliwość przejścia do interaktywności**
   Nie trzeba przepisywać całej aplikacji - można:
   - Zacząć od statycznego SSR
   - Dodać interaktywność tylko tam, gdzie potrzebna
   - Mieszać różne tryby w jednej aplikacji
3. **Wspólny kod .NET**
   Model, walidacja, logika biznesowa:
   - Dzielone między serwerem a klientem
   - Jedna baza kodu w C#
   - Możliwość reużycia bibliotek . NET

1.2 Tryby renderowania (@rendermode)

Cztery tryby renderowania

Blazor . NET 8 oferuje cztery główne tryby renderowania, które można mieszać w jednej aplikacji.

1.2.1 1. Static SSR (brak interaktywności)

Oznaczenie: brak atrybutu `@rendermode` lub `@rendermode="InteractiveServer"@(false)`

Charakterystyka:

- Renderowanie tylko po stronie serwera
- Brak możliwości obsługi zdarzeń (onClick, onChange, etc.)
- Minimalny rozmiar przesyłanych danych
- Idealny dla stron statycznych, treści, formularzy bez walidacji

Przykład:

```
@page "/about"

<h3>O nas</h3>
<p>Strona statyczna bez interaktywno ci</p>
```

1.2.2 2. Interactive Server

Oznaczenie: `@rendermode="InteractiveServer"`

Charakterystyka:

- Logika wykonywana po stronie serwera
- Komunikacja przez SignalR/WebSocket
- Mały rozmiar początkowego ładowania
- Każde zdarzenie wymaga round-trip do serwera
- Wymaga stałego połączenia

Zalety:

- Dostęp do zasobów serwera (baza danych, pliki)
- Bezpieczny kod (nie widoczny dla klienta)
- Szybkie uruchomienie aplikacji

Wady:

- Latencja sieci przy każdej interakcji
- Wymagane stałe połączenie
- Większe obciążenie serwera
- Nie działa offline

Przykład:

```razor
@page "/counter"
@rendermode InteractiveServer

<h3>Counter</h3>
<p>Current count: @currentCount</p>
<button @onclick="IncrementCount">Click me</button>

@code {
    private int currentCount = 0;

    private void IncrementCount()
    {
        currentCount++;
    }
}
```

1.2.3 3. Interactive WebAssembly

Oznaczenie: `@rendermode="InteractiveWebAssembly"`

Charakterystyka:

- Logika wykonywana w przeglądarce
- Aplikacja działa po stronie klienta
- Większy rozmiar początkowego pobierania
- Brak latencji przy interakcjach
- Możliwość pracy offline (PWA)

Zalety:

- Natychmiastowa reakcja na interakcje
- Lepsza skalowalność (mniej obciążenia serwera)
- Możliwość pracy offline
- Możliwość tworzenia PWA

Wady:

- Większy rozmiar początkowego pobierania (2-3 MB)
- Dłuższy czas pierwszego uruchomienia
- Kod widoczny dla klienta
- Ograniczony dostęp do zasobów systemowych

Przykład:

```
@page "/wasm-counter"
@rendermode InteractiveWebAssembly

<h3>WebAssembly Counter</h3>
<p>Current count: @currentCount</p>
<button @onclick="IncrementCount">Click me</button>

@code {
    private int currentCount = 0;

    private void IncrementCount()
    {
        currentCount++;
    }
}
```

1.2.4 4. Interactive Auto

Oznaczenie: `@rendermode="InteractiveAuto"`

Charakterystyka:

- Pierwszy raz: Interactive Server
- W tle pobiera WebAssembly
- Kolejne wizyty: Interactive WebAssembly
- Best of both worlds

Zalety:

- Szybkie pierwsze uruchomienie
- Późniejsza praca bez latencji
- Automatyczne przełączanie

Wady:

- Większa złożoność
- Trudniejszy debugging
- Kod musi być kompatybilny z oboma trybami

1.2.5 Wybór trybu renderowania

Tryb można wybrać na trzech poziomach:

1. **Globalnie** - w `App.razor`:

```
<Routes @rendermode="InteractiveServer" />
```

2. **Per strona** - w komponencie strony:

```
@page "/mypage"
@rendermode InteractiveServer
```

3. **Per komponent** - przy użyciu komponentu:

```
<MyComponent @rendermode="InteractiveServer" />
```

1.3 Koszt i trade-offy

Tryb	Zalety (+)	Wady (-)
Static SSR	Minimalny rozmiar Szybkie ładowanie Dobre SEO	Brak interaktywności Tylko proste formularze
Server	Mały rozmiar startowy Dostęp do serwera Bezpieczny kod	Stałe połączenia Latency roundtrip Nie działa offline
WebAssembly	Skalowalność Praca offline PWA możliwe Brak latencji	Większy download Dłuższe uruchomienie Kod widoczny
Auto	Szybki start Potem bez latencji Lepsze UX	Większa złożoność Trudniejszy debug

Tabela 1: Porównanie trybów renderowania

1.4 Stream Rendering & Enhanced Navigation

1.4.1 Stream Rendering

Stream Rendering pozwala na wyświetlenie części strony natychmiast, a następnie zaktualizowanie jej po załadowaniu wolnych danych.

Przykład:

```
@attribute [StreamRendering(true)]

@if (forecasts == null)
{
    <p>Loading...</p>
}
else
{
    <table>
        @foreach (var forecast in forecasts)
        {
            <tr><td>@forecast.Date</td></tr>
        }
    </table>
}

@code {
    private WeatherForecast[]? forecasts;

    protected override async Task OnInitializedAsync()
    {
        // Symulacja wolnego zapytania
        await Task.Delay(2000);
        forecasts = await GetForecasts();
    }
}
```

Działanie:

1. Serwer natychmiast zwraca HTML z "Loading..."
2. W tle wykonuje się `OnInitializedAsync()`
3. Po zakończeniu serwer streamuje zaktualizowany HTML
4. Przeglądarka podmienia zawartość

1.4.2 Enhanced Navigation

Enhanced Navigation przyspiesza nawigację między stronami statycznymi SSR poprzez:

- Pobieranie tylko zmienionych części strony (nie cały dokument)
- Zachowanie JavaScript state
- Szybsze przejścia (podobne do SPA)

Włączane domyślnie w `App.razor`:

```
<head>
    <HeadOutlet @rendermode="InteractiveServer" />
</head>
```

1.5 Struktura projektu "Blazor Web App"

Typowy projekt Blazor Web App (. NET 8) ma następującą strukturę:

```
MyBlazorApp/
 MyBlazorApp/              # Projekt główny (Server)
    Components/
       Layout/
       Pages/
       App.razor
    wwwroot/
    appsettings.json
    Program.cs

 MyBlazorApp.Client/       # Projekt WebAssembly
     Pages/
     wwwroot/
     Program.cs
```

Wyjaśnienie:

- **MyBlazorApp** - projekt główny, renderuje SSR i Server
- **MyBlazorApp.Client** - projekt dla WebAssembly
- Komponenty mogą być współdzielone przez oba projekty

1.6 Formularze w Blazor

1.6.1 EditForm

Podstawowy komponent do tworzenia formularzy z walidacją:

```
<EditForm Model="@model" OnValidSubmit="HandleValidSubmit">
    <DataAnnotationsValidator />
    <ValidationSummary />

    <div>
        <label>Nazwa:</label>
        <InputText @bind-Value="model. Name" />
        <ValidationMessage For="@(() => model.Name)" />
    </div>

    <button type="submit">Zapisz</button>
</EditForm>

@code {
    private PersonModel model = new();

    private void HandleValidSubmit()
    {
        // Zapisz dane
    }
}
```

1.6.2 Model z walidacją

```
using System
```

2 Materiały

2.1 Filmy instruktażowe

Materiały wideo

- Blazor SSR vs Server vs WebAssembly vs Auto
  Omówienie różnic między trybami renderowania w Blazor . NET 8
- CRUD z wszystkimi trybami renderowania (. NET 8)
  Praktyczne przykłady implementacji operacji CRUD
- CRUD z trybem WebAssembly (.NET 8)
  Szczegółowa implementacja w trybie WebAssembly

2.2 Dokumentacja Microsoft

Dokumentacja oficjalna

- Blazor (ASP.NET Core)
  Główna dokumentacja frameworka Blazor
- Render modes (.NET 8)
  Szczegółowy opis trybów renderowania
- Komponenty i routing
  Tworzenie i zarządzanie komponentami
- Forms & EditForm
  Praca z formularzami w Blazor
- Stream rendering / Enhanced navigation
  Zaawansowane techniki renderowania
- SignalR (Blazor Server)
  Komunikacja w czasie rzeczywistym
- Entity Framework Core
  ORM do pracy z bazami danych
- Hosting i publikacja
  Wdrażanie aplikacji Blazor
- PWA (Blazor WebAssembly)
  Tworzenie Progressive Web Apps

2.3 Repozytoria przykładowe

Przykładowy kod źródłowy

- GitHub: dotnet/blazor-samples
  Oficjalne przykłady od Microsoft
- GitHub: AspNetCore.Docs.Samples
  Przykłady z dokumentacji

2.4 Dodatkowe zasoby

- Strona główna Blazor
- Blazor University - nieoficjalny przewodnik
- Biblioteka komponentów Blazor

3 Cheat Sheet - Ściąga na kolokwium

3.1 Tryby renderowania - składnia

NAJWAŻNIEJSZE!

```
// 1. Brak interaktywno ci
@page "/static"
<h3>Static SSR</h3>

// 2. Interactive Server
@page "/server"
@rendermode InteractiveServer

// 3. Interactive WebAssembly
@page "/wasm"
@rendermode InteractiveWebAssembly

// 4. Interactive Auto
@page "/auto"
@rendermode InteractiveAuto
```

3.2 Stream Rendering

```
@attribute [StreamRendering(true)]

@if (data == null)
{
    <p>Loading...</p>
}
else
{
    <!-- Wy wietl dane -->
}

@code {
    private Data?  data;

    protected override async Task OnInitializedAsync()
    {
        data = await FetchDataAsync();
    }
}
```

3.3 Formularze

```
<EditForm Model="@model" OnValidSubmit="HandleSubmit">
    <DataAnnotationsValidator />
    <ValidationSummary />

    <InputText @bind-Value="model. Name" />
    <ValidationMessage For="@(() => model. Name)" />

```

```
    <button type="submit">Zapisz</button>
</EditForm>

@code {
    private MyModel model = new();

    private void HandleSubmit()
    {
        // Zapisz
    }
}
```

3.4 Walidacja modelu

```
using System.ComponentModel.DataAnnotations;

public class MyModel
{
    [Required(ErrorMessage = "Pole wymagane")]
    public string Name { get; set; } = "";

    [Range(1, 100)]
    public int Age { get; set; }

    [EmailAddress]
    public string Email { get; set; } = "";

    [Compare("Password")]
    public string ConfirmPassword { get; set; } = "";
}
```

3.5 Entity Framework

```
// DbContext
public class AppDbContext : DbContext
{
    public AppDbContext(DbContextOptions<AppDbContext> options)
        : base(options) { }

    public DbSet<Entity> Entities { get; set; }
}

// Program. cs
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlite("Data Source=app.db"));

// U ycie
@inject AppDbContext Db

@code {
    private List<Entity> items = new();

    protected override async Task OnInitializedAsync()
    {
        items = await Db.Entities.ToListAsync();
```

```
    }

    private async Task Add(Entity entity)
    {
        Db.Entities.Add(entity);
        await Db.SaveChangesAsync();
    }

    private async Task Delete(int id)
    {
        var item = await Db.Entities. FindAsync(id);
        if (item != null)
        {
            Db.Entities. Remove(item);
            await Db.SaveChangesAsync();
        }
    }
}
```

3.6 Dependency Injection

```
// Program.cs - rejestracja
builder.Services.AddScoped<IMyService, MyService>();
builder.Services.AddSingleton<ICacheService, CacheService>();
builder.Services.AddTransient<IEmailService, EmailService>();

// Komponent - u ycie
@inject IMyService MyService

@code {
    protected override async Task OnInitializedAsync()
    {
        var data = await MyService.GetDataAsync();
    }
}
```

3.7 HTTP w WebAssembly

```
@inject HttpClient Http

@code {
    // GET
    var items = await Http.GetFromJsonAsync<List<Item>>(
        "api/items");

    // POST
    await Http.PostAsJsonAsync("api/items", newItem);

    // PUT
    await Http.PutAsJsonAsync($"api/items/{id}", item);

    // DELETE
    await Http.DeleteAsync($"api/items/{id}");
}
```

3.8 Nawigacja

```
@inject NavigationManager Navigation

<button @onclick="Navigate">Przejd </button>

@code {
    private void Navigate()
    {
        Navigation.NavigateTo("/other-page");
    }
}
```

3.9 Parametry komponentu

```
// Definicja
@code {
    [Parameter]
    public string Title { get; set; } = "";

    [Parameter]
    public EventCallback OnClick { get; set; }
}

// U ycie
<MyComponent Title="Hello" OnClick="HandleClick" />
```

3.10 Lifecycle hooks

```
protected override void OnInitialized()
{
    // Synchroniczny, pierwszy
}

protected override async Task OnInitializedAsync()
{
    // Asynchroniczny, do API/DB
}

protected override void OnParametersSet()
{
    // Po ka dej zmianie parametr w
}

protected override async Task OnAfterRenderAsync(bool firstRender)
{
    if (firstRender)
    {
        // JS interop, focus, etc.
    }
}

public void Dispose()
{
    // Cleanup
}
```

3.11 Binding

```
// One-way
<p>@text</p>

// Two-way
<input @bind="text" />
<input @bind="text" @bind:event="oninput" />

// Checkbox
<input type="checkbox" @bind="isChecked" />

// Select
<select @bind="selectedValue">
    <option value="1">Opcja 1</option>
    <option value="2">Opcja 2</option>
</select>
```

3.12 Conditional rendering

```
@if (condition)
{
    <p>True</p>
}
else
{
    <p>False</p>
}

@switch (value)
{
    case 1:
        <p>Jeden</p>
        break;
    case 2:
        <p>Dwa</p>
        break;
    default:
        <p>Inne</p>
        break;
}
```

3.13 Loops

```
@foreach (var item in items)
{
    <div>@item.Name</div>
}

@for (int i = 0; i < 10; i++)
{
    <p>@i</p>
}
```

3.14 Event handling

```
<button @onclick="HandleClick">Click</button>
<input @onchange="HandleChange" />
<input @oninput="HandleInput" />
<form @onsubmit="HandleSubmit">

@code {
    private void HandleClick()
    {
        // ...
    }

    private void HandleClick(MouseEventArgs e)
    {
        // Z argumentami
    }

    private async Task HandleClickAsync()
    {
        // Asynchroniczny
    }
}
```

3.15 Kiedy którego trybu użyć - tabela decyzyjna

Scenariusz	**Tryb**
Strona "O nas", blog	Static SSR
Dashboard firmowy	Interactive Server
Edytor graficzny	Interactive WebAssembly
PWA, praca offline	Interactive WebAssembly
Aplikacja SaaS	Interactive Auto
Prototyp	Interactive Server
Wysoki ruch, publiczna	Interactive WebAssembly
Wymaga DB access	Interactive Server

Tabela 2: Decyzja o trybie renderowania

3.16 Najczęstsze błędy

UWAGA!

1. Brak @rendermode przy interaktywności

```
//  LE  - @onclick nie zadzia a
@page "/counter"
<button @onclick="Increment">Count</button>

// DOBRZE
@page "/counter"
@rendermode InteractiveServer
<button @onclick="Increment">Count</button>
```

2. Używanie DbContext w WebAssembly

```
//  LE  - WebAssembly nie ma dost pu do DB
@rendermode InteractiveWebAssembly
@inject AppDbContext Db

// DOBRZE - u yj HTTP API
@rendermode InteractiveWebAssembly
@inject HttpClient Http
```

3. Zapomnienie await

```
//  LE
var data = Db.Items.ToListAsync();

// DOBRZE
var data = await Db. Items.ToListAsync();
```

4. Brak DataAnnotationsValidator

```
//  LE  - walidacja nie dzia a
<EditForm Model="@model">
    <InputText @bind-Value="model. Name" />
</EditForm>

// DOBRZE
<EditForm Model="@model">
    <DataAnnotationsValidator />
    <InputText @bind-Value="model. Name" />
</EditForm>
```

3.17 Polecenia CLI

Przydatne komendy

```
# Nowy projekt
dotnet new blazor -o MyApp -int Server

# Dodanie pakietu
dotnet add package Microsoft.EntityFrameworkCore. Sqlite

# Migracje EF Core
dotnet ef migrations add InitialCreate
dotnet ef database update

# Uruchomienie
dotnet run

# Publikacja
dotnet publish -c Release
```