Wydział
Informatyki
POLITECHNIKA BIAŁOSTOCKA

Programowanie aplikacji WWW w technologii .NET

Razor Pages – wiązanie i walidacja danych

Ireneusz Mrozek

Materiały przygotowane w ramach projektu „PB 5.0 – dostosowanie oferty dydaktycznej Politechniki Białostockiej do potrzeb nowoczesnej gospodarki oraz zielonej i cyfrowej transformacji” (nr umowy FERS.01.05-IP.08-0327/23-00) w ramach programu Fundusze Europejskie dla Rozwoju Społecznego 2021-2027 współfinansowanego ze środków Europejskiego Funduszu Społecznego Plus.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską
NCBR Narodowe Centrum Badań i Rozwoju

Wiązanie danych (data binding)

Wiązanie modelu w Razor Pages to proces, który pobiera wartości z żądań HTTP i mapuje je na parametry metod obsługi lub właściwości zdefiniowane wewnątrz żądanej strony (`PageModel`).

Mechanizm redukuje potrzebę, ręcznego „wydobywania” przez dewelopera danych ze źródła żądania i przypisywania ich wartości do zmiennych lub właściwości. Tym samym mechanizm redukuje możliwość popełnienia pomyłki.

```
public class ContactModel : PageModel
{
    [BindProperty]
    public string Name { get; set; }
    [BindProperty]
    public string Email { get; set; }
    public void OnPost()    {
        // Kod przetwarzający formularz
    }
}
```

W przykładzie, pola formularza HTML odpowiadające właściwościom `Name` i `Email` będą automatycznie przekazane do PageModel po zatwierdzeniu formularza.

Wprowadzenie danych

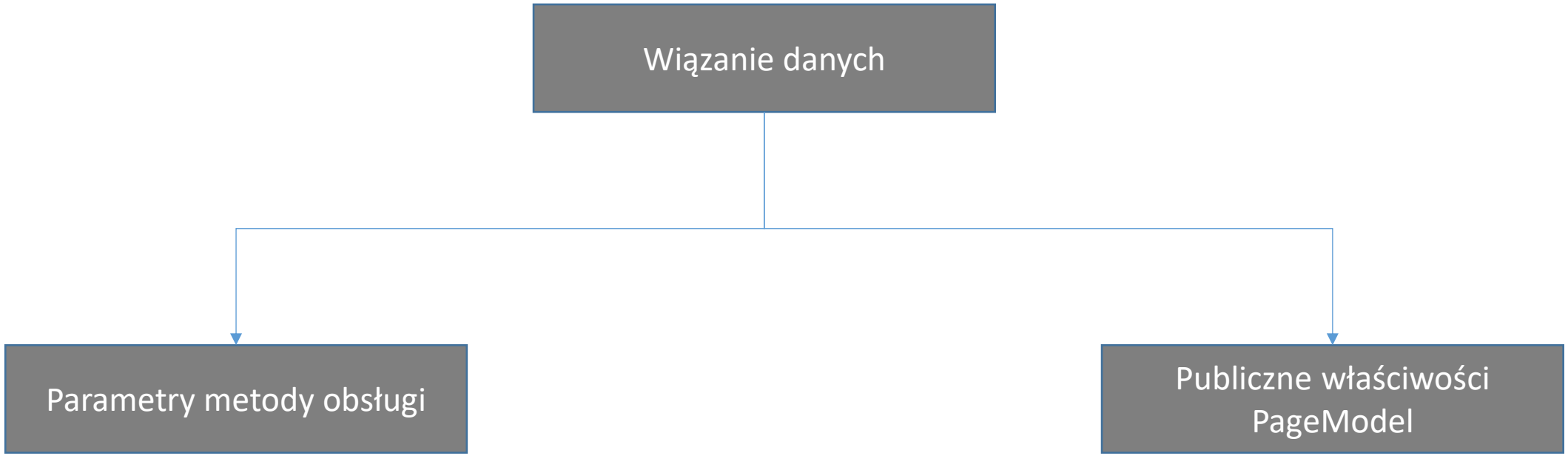
```
@page
@model ModelBidingModel
@{
}
<h3>@(ViewData["welcome"])</h3>
<form method="post">
    <label for="Name">Name:</label>
    <input type="text" id="Name" name="Name">

    <label for="Email">Email:</label>
    <input type="text" id="Email" name="Email">

    <button type="submit">Register</button>
</form>
```

Klasyczne odczytanie danych

```
public void OnPost()
{
    var title = Request.Form["Title"];
    var rating = Request.Form["Rating"];
    ViewData["info"] = $"Movie: {title}, rating: {rating}";
}
```

Dane z formularza można odczytać bezpośrednio przez Request.Form, jednak brak typowania i wsparcia IntelliSense sprawia, że rozwiązanie to jest mało bezpieczne w większych projektach.

Wiązanie danych



Parametry metody obsługi

```
public void OnPost(string Name, string Email)
{
     ViewData["confirmation"] = $"{name}, information will be sent to {email}";
}
```

Parametry są nazwane zgodnie z polami formularza i mają odpowiedni typ dla oczekiwanych danych.

Po otrzymaniu danych framework dopasowuje wartości z formularza do nazw parametrów i automatycznie je przypisuje, jeśli mogą zostać przekonwertowane na odpowiedni typ.

UWAGA:
Binding działa case-insensitive. ASP.NET Core automatycznie mapuje nazwy pól formularza do właściwości klasy, ignorując wielkość liter.

Publiczne właściwości PageModel

```
public class FeedbackModel : PageModel
{
    [BindProperty] public string User { get; set; }
    [BindProperty] public string Comment { get; set; }

    public void OnPost() =>
        ViewData["msg"] = $"Thanks {User} for your comment!";
}
```

Atrybut `[BindProperty]` pozwala automatycznie przypisać wartości pól formularza do publicznych właściwości modelu strony.

Publiczne właściwości PageModel

```
[BindProperties]   // wiąże wszystkie publiczne properties tej klasy
public class ContactModel : PageModel
{
    public string FullName { get; set; }
    public string Message { get; set; }

    public void OnGet()
    {
    }

    public void OnPost()
    {
        ViewData["info"] = $"Thank you {FullName}, we received your message: \"{Message}\"";
    }
}
```

Żądanie GET

```
public class ModelBindingModel : PageModel
{
    [BindProperty(SupportsGet = true)]
    public string Email { get; set; }
    [BindProperty(SupportsGet = true)]
    public int Id{ get; set; }
    public void OnGet()
    {
        ViewData["welcome"] = $"Welcome {Email}";
    }
}
```

Właściwość `SupportsGet = true`

Atrybuty routingu

```
@page "{productName}"
@model ProductBindingModel
@{
}
<h3>Product name: @ViewData["productName"]</h3>
```

```
public class ProductBindingModel : PageModel
{
    [BindProperty(SupportsGet = true)]
    public string ProductName { get; set; } =
string.Empty;

    public void OnGet()
    {
        ViewData["productName"] = ProductName;
    }

    public void OnPost()
    {
        // Example placeholder for POST logic
    }
}
```

Dane złożone

```
@page
@model ModelBindingModel

<input name="Contact.Name" />
<input name="Contact.Message" />
<h4>@ViewData["msg"]</h4>
```

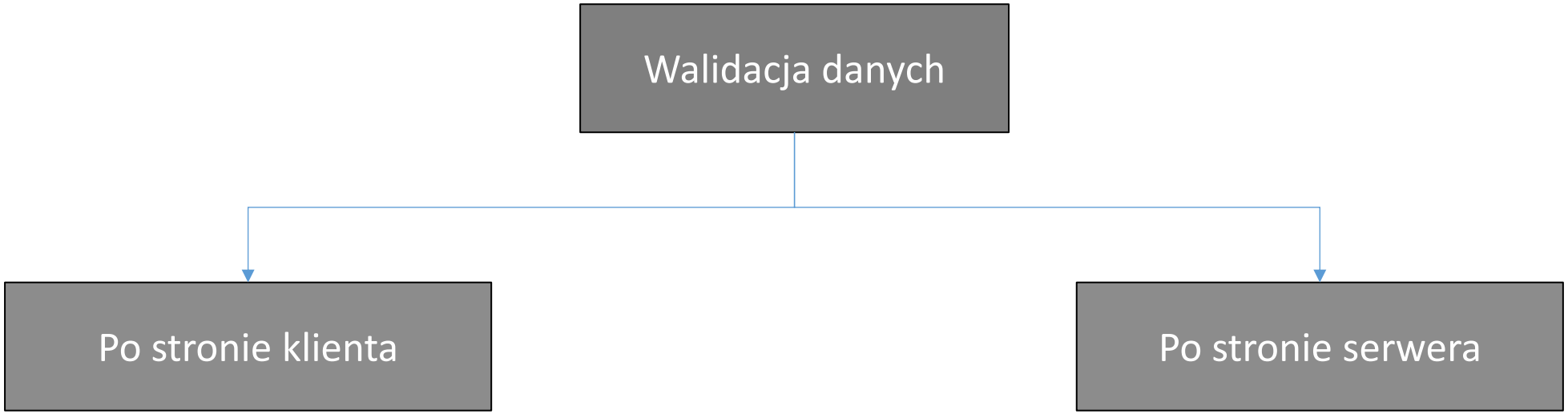
```
public class Contact
{
    public string Name { get; set; }
    public string Message { get; set; }
}

public class ModelBindingModel : PageModel
{
    [BindProperty]
    public Contact Contact { get; set; }

    public void OnPost() =>
        ViewData["msg"] = $"Hi {Contact.Name}";
}
```

Wiązanie danych działa również dla obiektów złożonych — wystarczy, że nazwy pól formularza odpowiadają strukturze właściwości (np. `Contact.Name`).

Walidacja danych

Walidacja danych to proces sprawdzania, czy dane wprowadzone przez użytkownika spełniają określone kryteria, takie jak format, zakres wartości czy zgodność z wymaganiami systemu. Służy do zapewnienia bezpieczeństwa, poprawności oraz prawidłowego działania aplikacji, eliminując błędne lub potencjalnie szkodliwe dane.

Walidacja danych



Atrybuty Data annotations

Data Annotation to wbudowane atrybuty w C#, które umożliwiają:

- Walidację danych wejściowych
- Definiowanie reguł modelu
- Ułatwienie pracy z Entity Framework i formularzami

```
public class User
{
    [Required]
    public string Name { get; set; }
    [EmailAddress]
    public string Email { get; set; }
}
```

Atrybuty Data annotations - walidacja

Atrybuty pól modelu (Data annotations) mogą również wspomagać walidację danych.

Atrybuty wspomagające walidację:

- Compare
- MaxLength
- MinLength
- Range
- Required
- StringLength

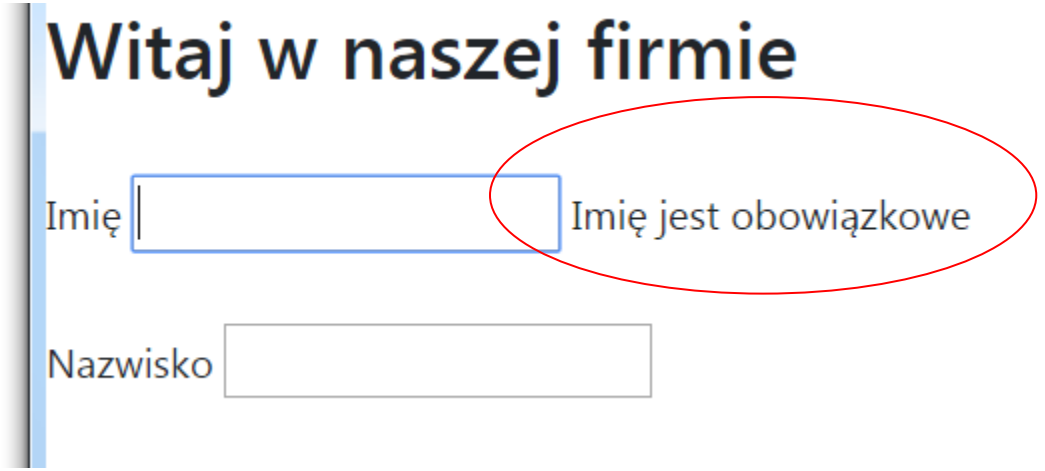
UWAGA:
Atrybuty, które nie mogą przyjmować wartości NULL uznawane są jako obowiązkowe (Required)

Atrybuty Data annotations

Atrybut	Znaczenie
Compare	Pozwala zdefiniować inny atrybut, z którym bieżący atrybut ma zostać porównany w celu stwierdzenia zgodności wartości
MaxLength/MinLength	Pozwala zdefiniować maksymalna/minimalną liczbę znaków/bajtów/elementów, które mogą być zaakceptowane.
Range	Pozwala zdefiniować zakres wartości jakie może dany element przyjąć.
RegularExpression	Pozwala sprawdzić wprowadzoną wartość pod kątem zgodności ze zdefiniowanym wyrażeniem regularnym.

Walidacja po stronie klienta

```
<label asp-for="person.firstName"></label>
 <input asp-for="person.firstName" />
 <span asp-validation-for="person.firstName"></span><br /><br />
```



Za wyświetlenie komunikatu o błędzie odpowiada element z atrybutem: `asp-validation-for=` .

Walidacja po stronie klienta

Index.cshtml.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using System.ComponentModel.DataAnnotations;

namespace WebApplication3.Pages
{
    public class IndexModel : PageModel
    {
        [BindProperty]
        [Required(ErrorMessage = "Imię jest
obowiązkowe")]
        [Display(Name = "Imię")]
        public string Name { get; set; }
    }
}
```

Index.cshtml

```
@page
@model IndexModel
<div style="margin-top:30px;">
    <form method="post">
        <label asp-for="Name"> </label>
        <div><input asp-for="Name" /></div>
        <span asp-validation-for="Name">
</span>
        <div><input type="submit" /></div>
    </form>
    @if (!string.IsNullOrEmpty(Model.Name))
    {
        <p>Hello @Model.Name!</p>
    }
</div>
```

Walidacja po stronie serwera

Niezależnie od walidacji po stronie klienta zawsze musi odbywać się walidacja po stronie serwera

Standardowa walidacja z wykorzystaniem mechanizmu `ModelState`

```
 public IActionResult OnPost(Person person)
{
    if (!ModelState.IsValid)
    {
        return Page();
    }
    return RedirectToPage("Test");
}
```

Dane z formularza można odczytać wykorzystując atrybut `[BindProperty]`

```
public class IndexModel : PageModel
{
    [BindProperty]
    public Person person { get; set; }
…
```