

**ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej**

# Aplikacje webowe na platformę .NET

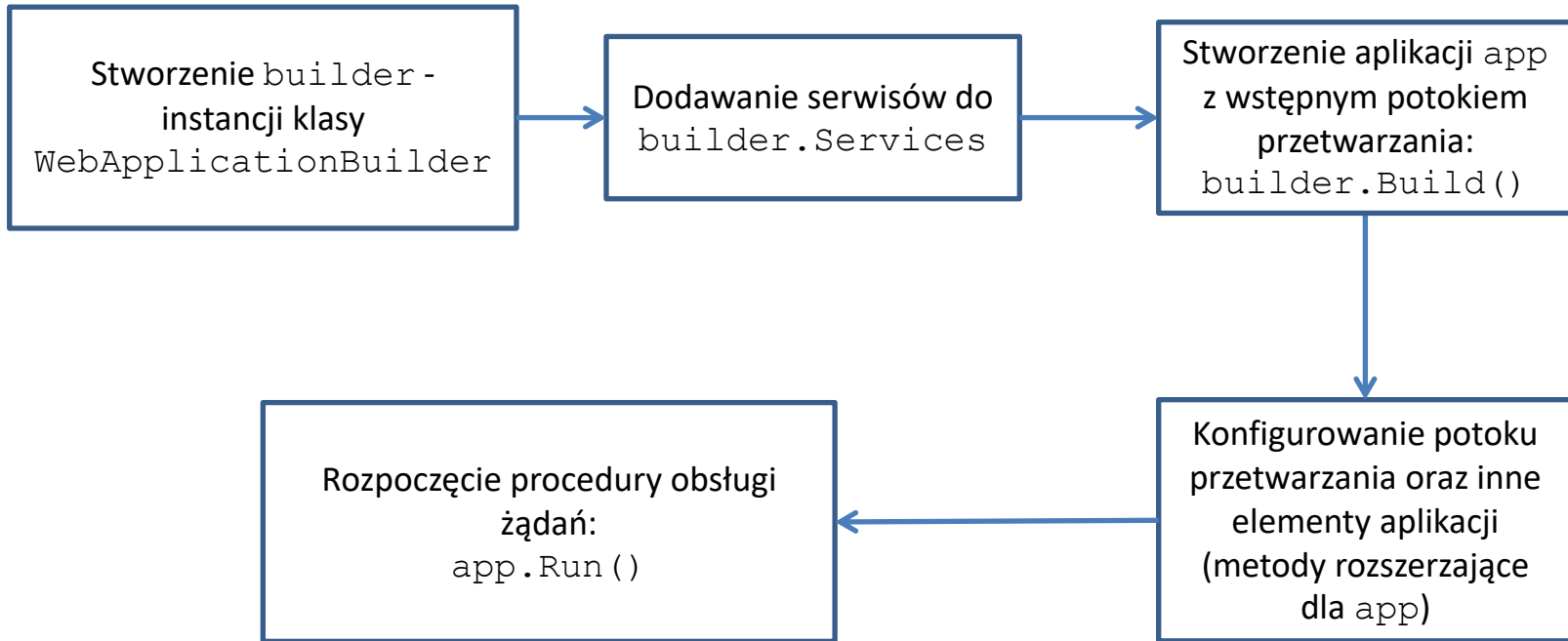
W11 – Potok przetwarzania ciasteczka,  
sesja, strony Razor

# Syllabus

- Potok przetwarzania
- Oprogramowanie pośredniczące
- Bezstanowość protokołu HTTP a ciasteczka
- Działanie ciasteczek
- Ciasteczka w ASP .NET
- Sesja
- TempData
- Atak CSRF
  - Zabezpieczenie przeciw atakowi
- Strony Razora
  - Idea
  - Struktura powiązań
  - Generacja kodu stron i modeli (z wykorzystaniem Entity Framework)
  - Inne opcje routingu i data bindingu
  - Handler (wiele formularzy na jednej stronie)
  - Adnotacja [TempData]
  - Tag-helper `partial`
  - Komponent widoku
  - Obszary – Areas
  - Porównanie do MVC

# **Potok przetwarzania, oprogramowanie pośredniczące**

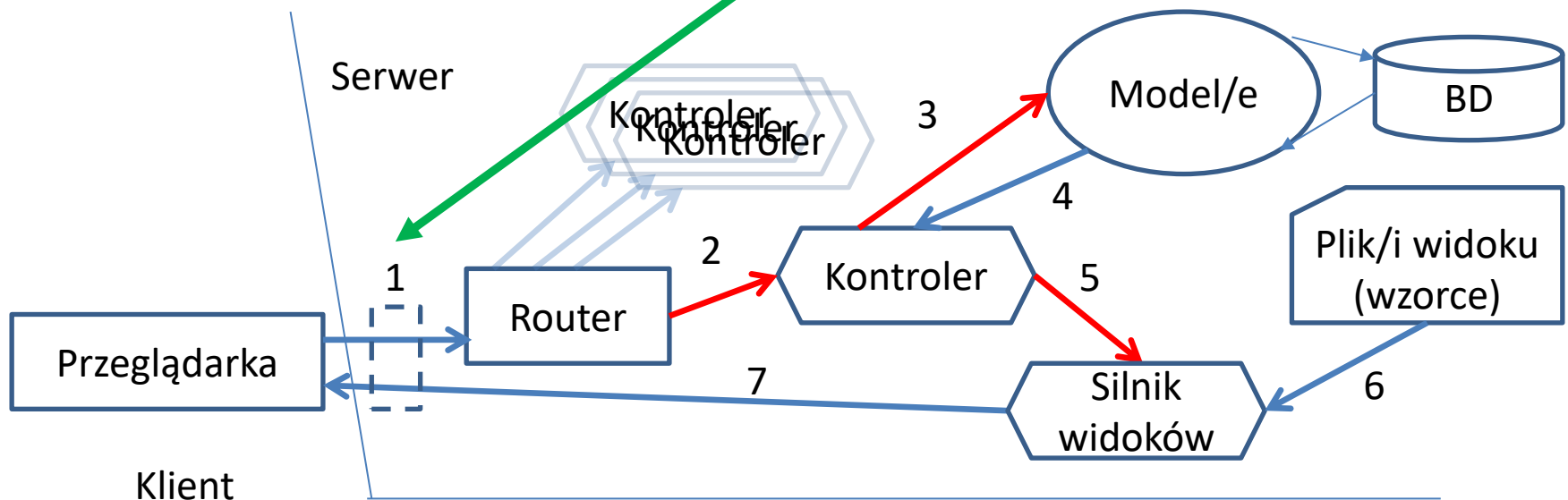
# Sposób przytowania potoku przetwarzania



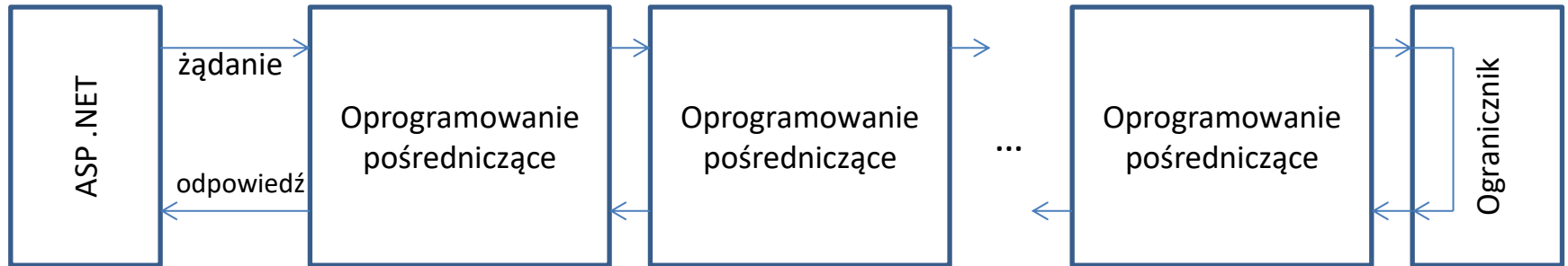
- Konfigurowanie aplikacji umożliwia podanie zależności, które nie można wyrazić za pomocą wstrzykiwania.
  - W tym potok przetwarzania.

# MVC (i nie tylko) w kontekście aplikacji webowych (wykład W08)

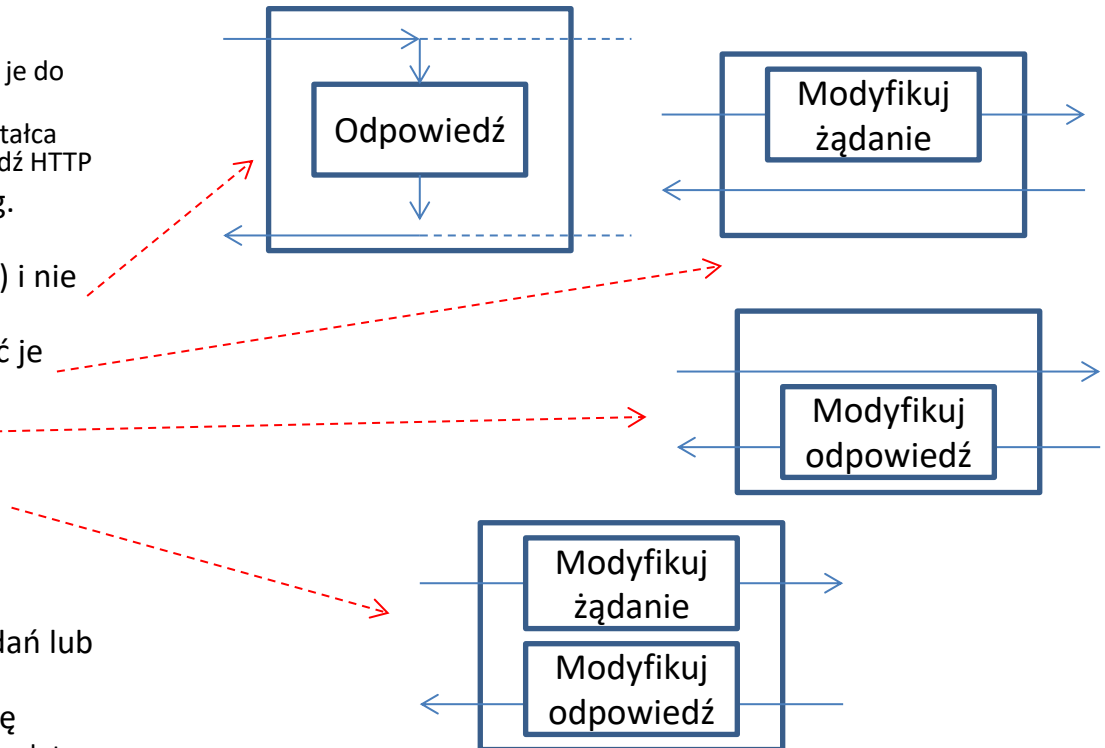
- Przebieg obsługi żądania HTTP w ASP:
  - Żądanie HTTP (1), przetworzone przez serwer, tworzy obiekt `HttpContext` (m. in. z właściwością `Request`) zawierający wszystkie informacje z żądania przetworzone na odpowiednie właściwości (ścieżka URL, parametry zapytania POST/GET/inne itp.)
  - W większość przypadków obiekt ten nie będzie używany wprost (starsze podejście), ale informacje w nim zawarte będą używane wraz z mechanizmem odbicia do kolejnych kroków.
  - Na drodze (1) działa jeszcze tzw. **oprogramowanie pośredniczące**, które może zmodyfikować obiekt `HttpContext`.
    - W tym miejscu jest też obsługa ciasteczek, sesji, autoryzacji itd. (elementy kolejnych wykładów)



# Potok oprogramowania pośredniczącego (OP)



- Blok ASP.NET
  - odczytuje żądanie HTTP i „przekopakuje” je do `HttpContext.Request`
  - Odbiera wytworzoną odpowiedź i przekształca `HttpContext.Response` w odpowiedź HTTP
- Oprogramowania pośredniczące (OP, ang. middleware) ułożone są w ciąg.
- OP może wygenerować treść (odpowieź) i nie przesłać dalej żądania
- OP może zmodyfikować żądanie i przesłać je dalej
- OP może zmodyfikować odpowiedź
- OP teoretycznie może równocześnie zmodyfikować żądanie, a po powrocie sterowania na strumieniu powrotnych zmodyfikować odpowiedź
- Wszystkie powyższe działania mogą być warunkowe, czyli tylko dla wybranych żądań lub odpowiedzi
- Ogranicznik zwraca potok w drugą stronę
  - Najczęściej poprawne żądanie do niego nie dotrze



# Włączenie podstawowej usługi ASP

- Projekt WebApp9\_Middleware, MVC
  - z usunięciem większości oprogramowania pośredniczącego

```
public static void Main(string[] args)
{
    var builder = WebApplication.CreateBuilder(args);
    // Add services to the container.
    builder.Services.AddControllersWithViews();
    var app = builder.Build();

    #region my middlewares
    #endregion
    app.UseRouting();
    app.UseStaticFiles();
    app.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}")
        .WithStaticAssets();

    app.Run();
}
```

- Stwórzmy serwis ( w folderze /Services) pokazujący czas działania aplikacji (zostanie dodany do kontenera serwisów)

```
public class UptimeService
{
    private Stopwatch timer;

    public UptimeService()
    {
        timer = Stopwatch.StartNew();
    }

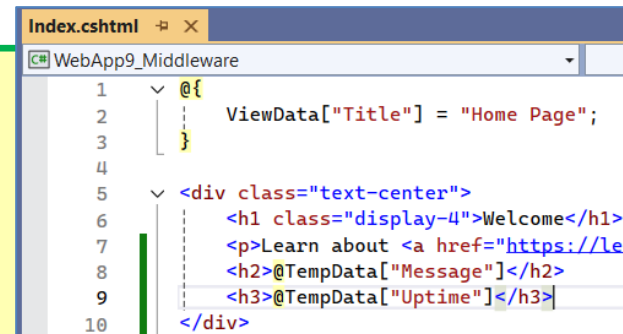
    public long Uptime => timer.ElapsedMilliseconds;
}
```

# Wstrzyknięcie serwisu w kontrolerze HomeController

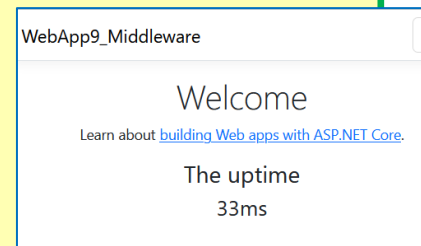
- W tym przypadku warto dodać serwis jako AddSingleton
  - Z jednym parametrem generycznym
  - Możemy stworzyć **tylko** akcję Index w kontrolerze HomeController
  - Brak akcji Privacy()
- Scenariusz użycia: uruchomienie, po pewnym czasie odświeżenie strony WWW.

```
// Add services to the container.  
builder.Services.AddControllersWithViews();  
builder.Services.AddSingleton<UptimeService>();
```

```
public class HomeController : Controller  
{  
    private UptimeService uptime;  
    public HomeController(UptimeService up)  
    {  
        uptime = up;  
    }  
  
    public IActionResult Index()  
    {  
        TempData["Message"] = "The uptime";  
        TempData["Uptime"] = $"{uptime.Uptime}ms";  
        return View();  
    }  
}
```



```
Index.cshtml  
WebApp9_Middleware  
1 @section  
2     ViewData["Title"] = "Home Page";  
3 }  
4  
5 <div class="text-center">  
6     <h1 class="display-4">Welcome</h1>  
7     <p>Learn about <a href="https://learn.microsoft.com/aspnet/core">building Web apps with ASP.NET Core.</a>  
8     <h2>@TempData["Message"]</h2>  
9     <h3>@TempData["Uptime"]</h3>  
10 </div>
```

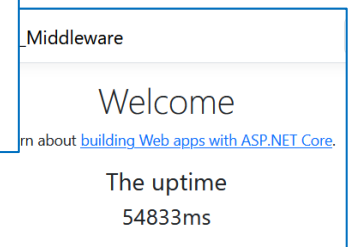


WebApp9\_Middleware

Welcome

Learn about [building Web apps with ASP.NET Core.](#)

The uptime  
33ms



Middleware

Welcome

Learn about [building Web apps with ASP.NET Core.](#)

The uptime  
54833ms



## OP generujące odpowiedź

- Tworzenie OP, które przechwyci żądanie ze ścieżką „/middleware”, zanim uruchomi się domyślna reguła routingu
- Dodatkowo wstrzyknięcie stworzonego serwisu

```
public class CatchMiddleware
{
    private RequestDelegate nextDelegate;
    private UptimeService uptime;

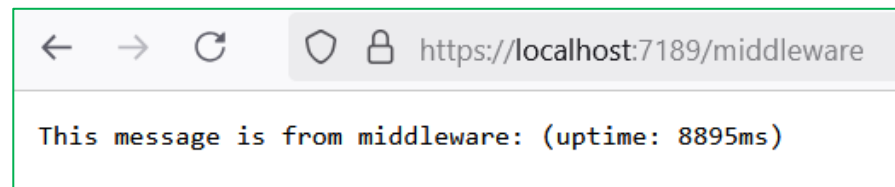
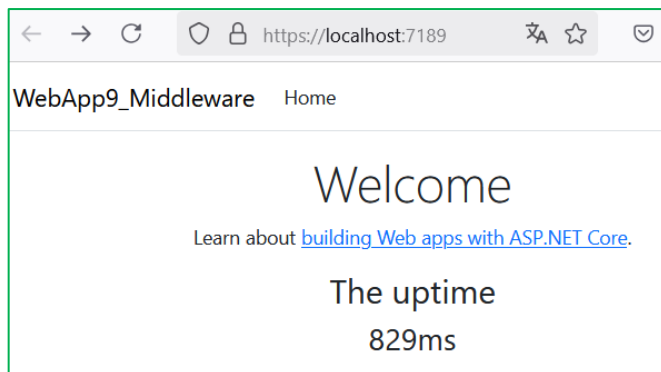
    public CatchMiddleware(RequestDelegate next, UptimeService up)
    {
        nextDelegate = next;
        uptime = up;
    }

    public async Task Invoke(HttpContext httpContext)
    {
        if (httpContext.Request.Path.ToString().ToLower() == "/middleware")
        {
            await httpContext.Response.WriteAsync(
                " This message is from middleware: " + $"(uptime: {uptime.Uptime}ms)", Encoding.UTF8);
            // do NOT call nextDelegate.Invoke
        }
        else
        {
            await nextDelegate.Invoke(httpContext);
        }
    }
}
```

# Użycie w konfiguracji

```
...  
#region my middlewares  
app.UseMiddleware<CatchMiddleware>();  
#endregion  
...
```

- Scenariusz użycia: adres URL z `/middleware` i bez
- Dlaczego nie w ramach zwykłych reguł routingu?
  - Pozwala w konfiguracji developerskiej dodać odpowiedzi na pewne żądania, których nie będzie w wersji użytkowej, a dodanie/usunięcie to tylko dodanie/usunięcie jednej linijki z procedury `Configure()`
  - Można wręcz napisać inne konfiguracje potoków oprogramowania pośredniczącego w zależności od środowiska (developerskie/klienckie)



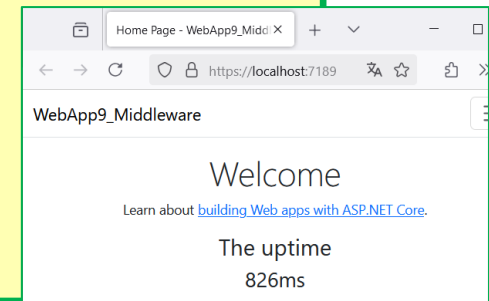
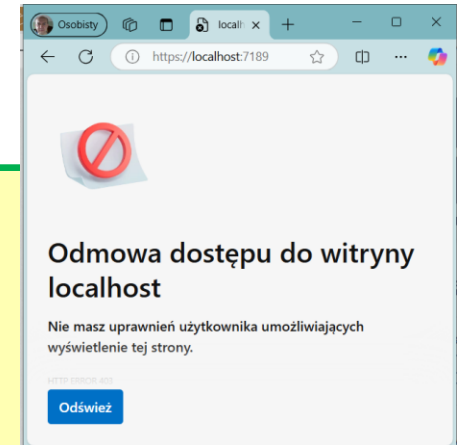
# OP skracające potok żądań

- Założmy, że obecne rozwiązanie w pewnej przeglądarce (np. MS Edge) działa niepoprawnie, czy wręcz myląco.
- OP reagujące na takie żądanie i natychmiast zwracające błąd.
- Scenariusz użycia: w przeglądarce MS Edge i innej

```
public class BlockMsEdge
{
    private RequestDelegate nextDelegate;

    public BlockMsEdge(RequestDelegate next)
        => nextDelegate = next;

    public async Task Invoke(HttpContext httpContext)
    {
        if (httpContext.Request.Headers["User-Agent"].Any(v => v.ToLower().Contains("edg")))
        {
            httpContext.Response.StatusCode = 403;
        }
        else
        {
            await nextDelegate.Invoke(httpContext);
        }
    }
}
```



```
...
#region my middlewares
app.UseMiddleware<CatchMiddleware>();
app.UseMiddleware<BlockMsEdge>();
#endregion
...
```

## OP modyfikujące żądanie 1/2

- HttpContext posiada słownik pod właściwością Items.
- Rozbijamy poprzednie OP na dwa:
  - OP sprawdzające typ przeglądarki i jeśli jest to MS Edge, to ustawia wybrany klucz w słowniku Items (np. Item["EdgeBrowser"]) na **true** oraz wykonuje kolejne OP w ciągu
  - OP sprawdzające ten klucz i generujące ewentualnie kod błędu.
- Pokazuje to możliwość przekazywania danych między OP, ale też danych do kontrolerów.

```
public class BrowserTypeMiddleware
{
    private RequestDelegate nextDelegate;

    public BrowserTypeMiddleware(RequestDelegate next)
        => nextDelegate = next;

    public async Task Invoke(HttpContext httpContext)
    {
        httpContext.Items["EdgeBrowser"]
            = httpContext.Request.Headers["User-Agent"].Any(v => v.ToLower().Contains("edg"));
        await nextDelegate.Invoke(httpContext);
    }
}
```

## OP modyfikujące żądanie 2/2

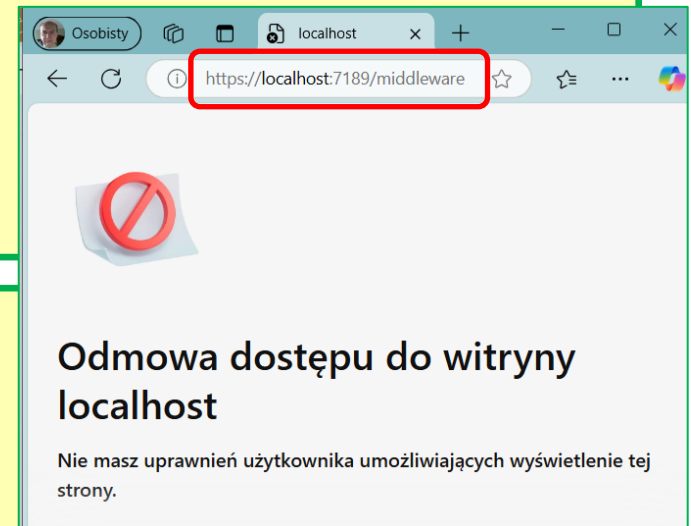
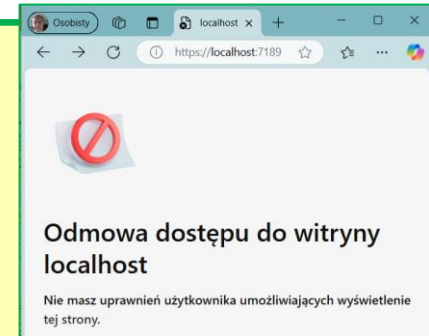
- Zmodyfikujemy `BlockMsEdge` tak, aby korzystało z kolekcji `Items`

```
public class BlockMsEdge
{
    private RequestDelegate nextDelegate;

    public BlockMsEdge(RequestDelegate next)
        => nextDelegate = next;

    public async Task Invoke(HttpContext httpContext)
    {
        if (httpContext.Items["EdgeBrowser"] as bool? == true)
        {
            httpContext.Response.StatusCode = 403; // do NOT call nextDelegate.Invoke()
        }
        else
        {
            await nextDelegate.Invoke(httpContext);
        }
    }
}
```

```
...
#region my middlewares
app.UseMiddleware<BrowserTypeMiddleware>();
app.UseMiddleware<BlockMsEdge>();
app.UseMiddleware<CatchMiddleware>();
#endregion
...
```



# OP modyfikujące odpowiedź

- OP, które powiadomi (w czytelny dla użytkownika sposób), że:
  - przeglądarka MS Edge nie jest obsługiwana.
  - Lub jakiś inny błąd 403 i 404.

```
public class ErrorPageMiddleware
{
    private RequestDelegate nextDelegate;

    public ErrorPageMiddleware(RequestDelegate next)
    {
        nextDelegate = next;
    }

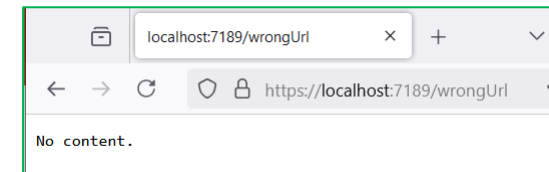
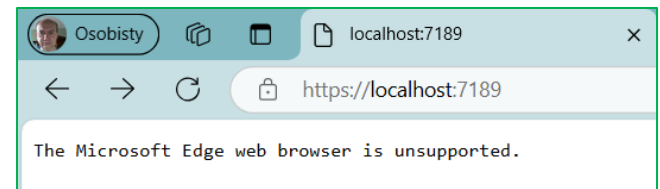
    public async Task Invoke(HttpContext httpContext)
    {
        await nextDelegate.Invoke(httpContext); // forward without changes

        // on return path modify response
        if (httpContext.Response.StatusCode == 403 &&
            (httpContext.Items["EdgeBrowser"] as bool? == true)) // one condition is enough
        {
            await httpContext.Response
                .WriteAsync("The Microsoft Edge web browser is unsupported.", Encoding.UTF8);
        }
        else if (httpContext.Response.StatusCode == 404)
        {
            await httpContext.Response
                .WriteAsync("No content.", Encoding.UTF8);
        }
    }
}
```

# Cały potok OP

```
...
#region my middleware
app.UseMiddleware<ErrorPageMiddleware>();
app.UseMiddleware<BrowserTypeMiddleware>();
app.UseMiddleware<BlockMsEdge>();
app.UseMiddleware<CatchMiddleware>();
#endregion
...
```

- Analiza potoku do przodu i od tyłu.
- Scenariusz użycia:
  - Przez MS Edge
  - URL z „/middleware”
  - URL „/” lub „/Home” lub „/Home/Index”
  - URL „/wrongUrl”
  - Analogicznie dla np. przeglądarki Firefox



# Potok przetwarzania - podsumowanie

- Ważna jest kolejność bloków w potoku
  - W przykładzie wstawienie oprogramowanie pośredniczącego `app.UseMiddleware<CatchMiddleware>()` jako pierwsze spowoduje, że adres `.../middleware` będzie działał tak samo dla każdej przeglądarki
  - Część bloków zależy od przekształceń wcześniejszych lub zakładają odrzucenie (filtr) niepoprawnych żądań (i już tego nie sprawdzają)
- Większość potoku działa na **nagłówku** żądania/odpowiedzi (czyli używa odpowiednie pola `HttpContext`), dopiero na końcu potoku analizowane jest **ciało** żądania (dzięki OP dodawanemu poprzez `app.UseRouting()` i `app.MapControllerRoute()`). Czyli dopiero finalnie działa:
  - Mechanizm data binding
  - Wywołanie akcji kontrolera
- Istnieje wiele pakietów z oprogramowaniem pośredniczącym wewnątrz konkretnego zestawu.
- Wiele pakietów, aby poprawnie działać, wymaga wstawienia oprogramowania pośredniczącego (i to w odpowiednim miejscu) jak również, oczywiście, rejestracji serwisu/ów w kontenerze.
  - Np. `UseSession()`,
- Pobieranie/ustawianie ciasteczek, zmiennych sesyjnych, metody autoryzacji itd. są oparte o wstawienie oprogramowania pośredniczącego w odpowiednie miejsce.



# Ciasteczka

# Ciasteczka a bezstanowość HTTP

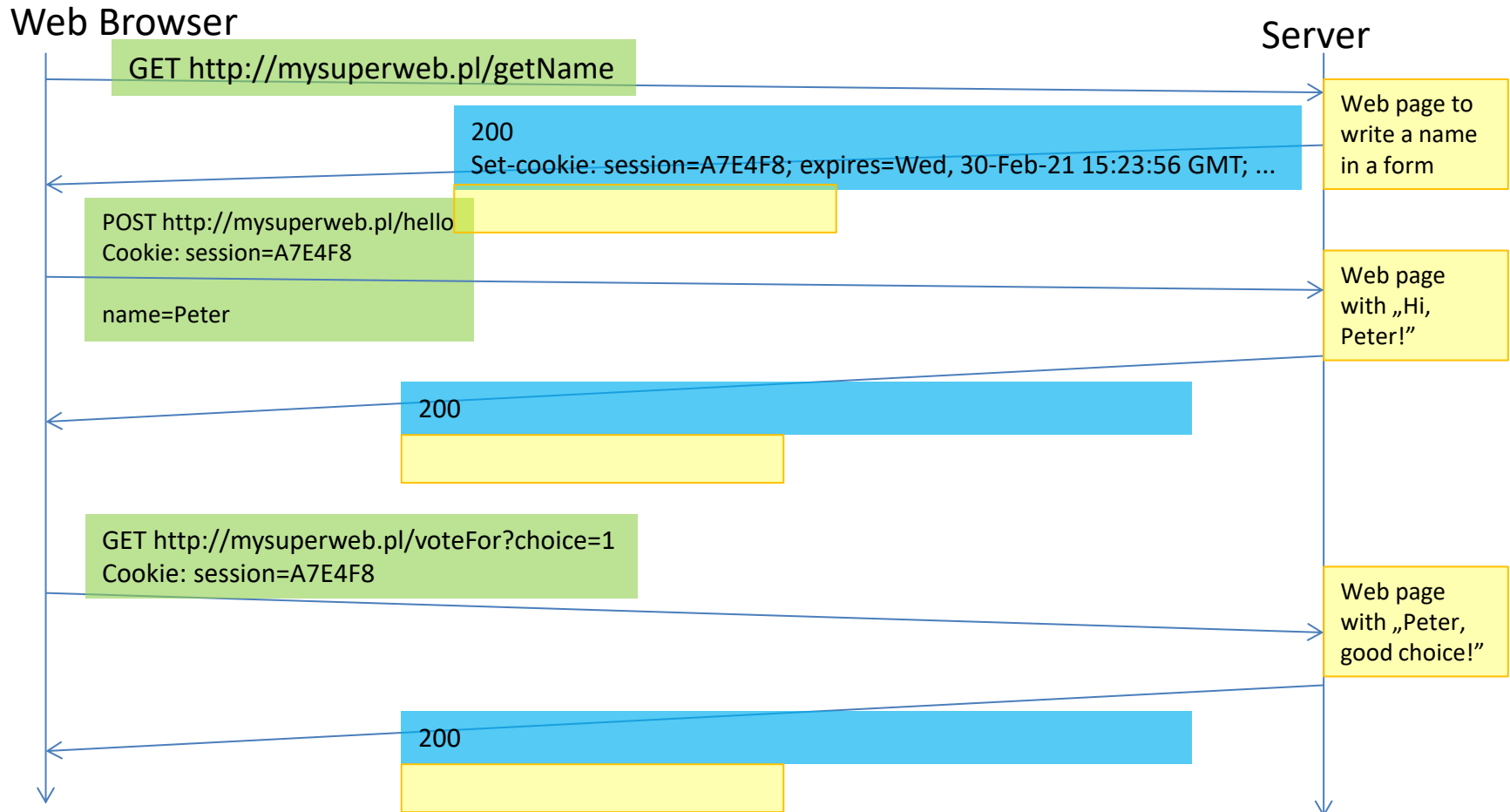
- Protokół HTTP jest bezstanowy
- Potrzeba **łączenia kolejnych żądań**, aby zapamiętać, że są od **tego samego użytkownika**
- Mechanizm – **ciasteczka** (ang. cookies)
  - Przekazywane są **w nagłówku**: żądania lub odpowiedzi
    - Niewidoczne dla zwykłego użytkownika
  - Ciasteczko posiada **klucz** (tekst) i **wartość** (tekst)
  - Mają ustalony termin ważności, przekazywany w **ciasteczkach odpowiedzi**:
    - punkt w czasie
    - bez podania czasu ważności – na czas sesji, rozumianej jako czas działania przeglądarki
  - Ciasteczko odpowiedzi posiada też inne cechy (czy dostęp tylko dla źródłowej domeny, czy dla wszystkich domen itp.)
- W ciasteczka można trzymać bezpośrednio pewne informacje od użytkownika:
  - preferencje kolorystyczne,
  - kolejność bloków z informacjami, które preferuje
  - koszyk zakupowy
  - które reklamy kliknął użytkownik
  - itp. w tym **sesję**

## Ciasteczka a przeglądarka

- Przeglądarka działa wg prostego algorytmu:
  - Dla ciasteczka w odpowiedzi:
    - Zapamiętuje wszystkie ciasteczka z domeny skąd przyszła odpowiedź
    - Jeśli ciasteczko o podanym kluczu **już było**, wcześniej **usuwa poprzednie**
  - Przy wysyłaniu żądania do określonej domeny wysyła **wszystkie nieprzeterminowane** ciasteczka (również z **innej karty** lub **innej instancji** tej samej przeglądarki)
    - Przeterminowane może usunąć z pamięci przeglądarki
  - Gdy przeglądarka **kończy (zaczyna)** działanie **usuwa** wszystkie **sesyjne** i ewentualnie **przeterminowane ciasteczka**
    - Jednak pozostałe zachowuje w pliku do kolejnego włączenia przeglądarki!
- Ciasteczka powinny być **nieduże** (jedno do 4096 bajtów) a ich **liczba** dla jednej domeny też jest często **ograniczona**
- **Użytkownik** przeglądarki może **manipulować** ciasteczkami (**usunąć**) lub wręcz **zabronić** ich używania dla określonej domeny

# Ciasteczka - sesja

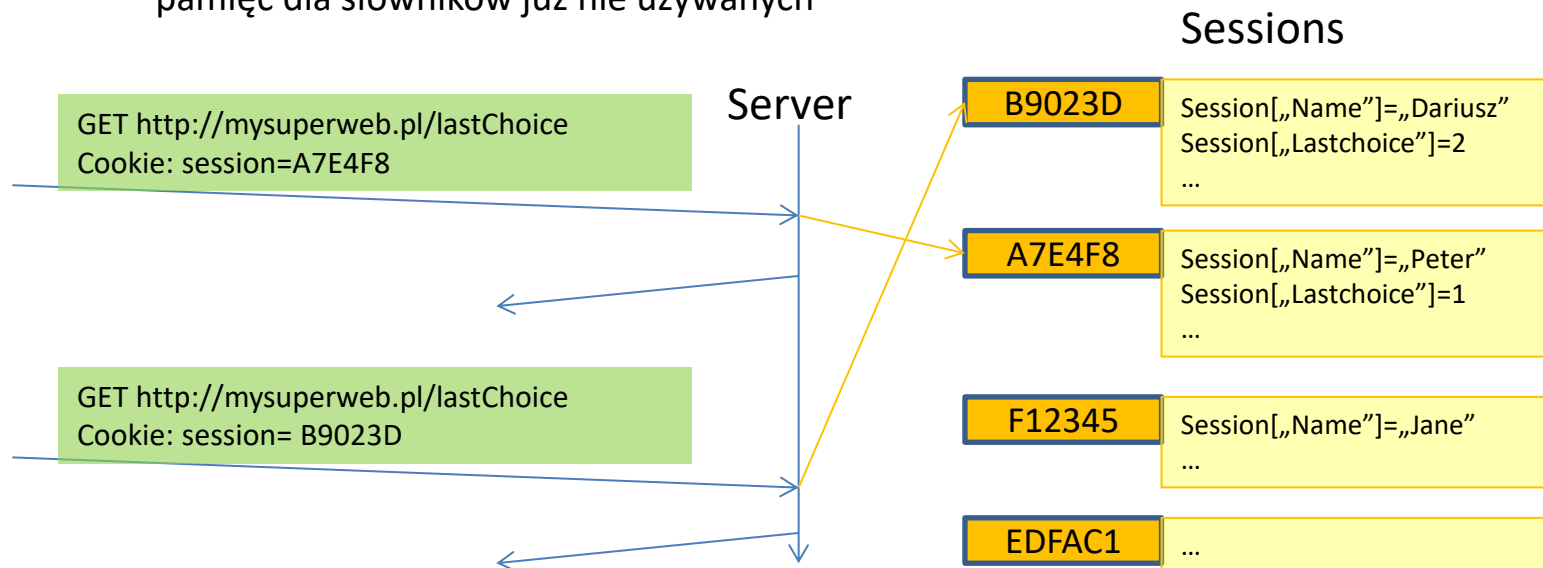
- Najczęściej używane są ciasteczka do zapamiętania sesji z użytkownikiem oraz danych z sesją związanych.
- Sesja** jest rozumiana jako ciąg **kolejnych żądań** od **tego samego użytkownika**.



# Cookie dla sesji – działanie serwera

Serwer:

- Tworzy ciasteczka i dodaje do odpowiedzi na żądanie
  - Nowe ciasteczko lub modyfikacja wcześniej wysłanego (np. podając termin ważności, który już minął spowoduje usunięcie ciasteczka z pamięci przeglądarki)
- Odczytuje ciasteczko z żądania i na jego podstawie decyduje o podjęciu odpowiedniej akcji.
- W przypadku **sesji** trzymane wartości w ciasteczku to **losowy string**, dla którego po stronie serwera zapamiętywany jest słownik klucz-wartość (same ciasteczka to klucze słownika wyższego poziomu).
  - Serwer odczytuje z żądania ciasteczko z sesją, znajduje powiązany z nim słownik i korzysta z kluczy tego słownika przygotowując stronę z odpowiedzią.
  - Serwer również zapamiętuje dane ciasteczka (np. o terminie ważności), np. żeby zwolnić pamięć dla słowników już nie używanych



## Ciasteczka w ramach ASP .Net Core

Ogólnie (są w różnych właściwościach)

- Realizowane są przez **oprogramowanie pośredniczące** (wykład W12)
- Dostęp bezpośredni do ciasteczek za pomocą obiektu typu kolekcja ciasteczek (jest wiele, w nazwie `Cookies`)
- Sesja - automatyczne powiązanie jednego **ciasteczka sesyjnego** ze słownikiem dla sesji (jak na poprzednim slajdzie) – interfejs `ISession` najczęściej we właściwości o nazwie `Session`.
- Dane dla przekierowań (`RedirectToAction/RedirectToPage`) w słowniku `TempData` są przechowywane w podobnych słownikach jak dla sesji (kolejne slajdy)
- Wszystkie te słowniki są (mogą być) jako **właściwości** (dla MVC) kontrolera lub (dla stron Razora) modelu strony
  - W przypadku sesji należy ją skonfigurować
- Dane dla przekierowań (`TempData`) standardowo są połączone ze specjalnym ciasteczkiem do tego celu, ale można tak skonfigurować aplikację, aby połączyć je z sesją (czyli ciasteczkiem dla sesji).

# TempData - szczegóły

- TempData to specjalny słownik gdzie każda para klucz-wartość posiada **znacznik**, czy po wysłaniu odpowiedzi na żądanie ma dana para zostać usunięta ze słownika.
- Standardowo odczyt spod klucza oznacza usunięcie go ze słownika
- TempData najczęściej się stosuje przy przekierowywaniu strony (kod odpowiedzi 302: przekieruj żądanie na URL podany w nagłówku pod 'location')
- Czyli przeglądarka generuje **nowe** żądanie
- Rozwiązane to jest przez specjalne ciasteczko (jakby sesja na jedno, kolejne żądanie).
  - Najpierw ciasteczko bez terminu ważności (ciasteczko sesyjne przeglądarki)
  - Kolejne ciasteczko z terminem ważności w **przeszłości** (usunięcie z pamięci przeglądarki)

**Code Snippet:**

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> CreateEnroll([Bind("StudentId,CourseId")] CourseStudent courseStudent)
{
    try
    {
        if (ModelState.IsValid)
        {
            _context.CourseStudent.Add(courseStudent);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        return View(nameof(Index));
    }
    catch (DbUpdateException)
    {
        TempData["Error"] = "Duplicate enrollment";
        return RedirectToAction(nameof(CreateEnroll));
    }
}
```

**Browser Response (302 Found):**

- Method: POST
- URL: https://localhost:44398/Students/CreateEnroll
- Location: /Students/CreateEnroll

**Cookies:**

- TempData:** expires: "1970-01-01T00:00:00.000Z", httpOnly: true, path: "/", samesite: "Lax", value: ""

## TempData – specjalny słownik

- Standardowo odczyt spod klucza oznacza **ustawienie znacznika**, że para **ma być usunięta** ze słownika przy wysyłaniu **odpowiedzi na żądanie HTTP**, ale...
- Można odczytać wartość ze słownika TempData bez usuwania dla kolejnego żądania HTTP
  - Dla obiektu TempData użyć metody `Peek(key)` zamiast odczytu poprzez indeksator `[]`.
- Lub **nawet po odczycie** oznaczyć, aby zostawić podaną parę w słowniku
  - Metoda `Keep(key)`
- Dlatego ciasteczko związane z TempData jest **bez daty ważności**, natomiast „użycie” wszystkich wartości TempData bez wywoływania `Peek()` lub `Keep()` spowoduje wstawienie do nagłówka odpowiedzi klucz powiązanego ciasteczko z czasem wartości **w przeszłości**, celem jego **usunięcia** z przeglądarki
- Oczywiście **same wartości** słownika TempData są pamiętane tylko **po stronie serwera** (dość podobnie do słownika sesji)



# Cookie - korzystanie bezpośrednio

- Dostęp do ciasteczki można uzyskać z obiektu klasy (nazwijmy go `Accessor`) implementującej interfejs `IHttpContextAccessor`, który można wstrzyknąć np. w konstruktor wybranego obiektu (np. kontrolera). Ten `Accessor` posiada dostęp poprzez dwa słowniki `Accessor.HttpContext.Request.Cookies` oraz `Accessor.HttpContext.Response.Cookies`
  - Operuje się wtedy na obiektach klasy `HttpCookie`
- Dla MVC (i stron Razora) w ramach kontrolera lub modelu strony istnieją właściwości `Request` i `Response` z dostępem (właściwością) do kolekcji ciasteczek `Cookies`. Mimo takiej samej nazwy są to kolekcje różnych typów z operacjami zależnymi od tego, czy są to ciasteczka żądania, czy też ciasteczka odpowiedzi.
  - `Request.Cookies` jest kolekcją typu `HttpRequestCookieCollection`
  - `Response.Cookies` jest kolekcją typu `IResponseCookies`

```
public IActionResult SetCookies()
{
    SetCookie("short", "Short Cookie is active", 10);
    SetCookie("long", "Long Cookie is active", 60);
    SetCookie("veryLong", "Very long Cookie is active", 600);
    SetCookie("forSession", "Cookie for session is active");
    return View();
}

Odwolania: 0
public IActionResult ShowCookies()
{
    ViewData["short"] = Request.Cookies["short"];
    ViewData["long"] = Request.Cookies["long"];
    ViewData["veryLong"] = Request.Cookies["veryLong"];
    ViewData["forSession"] = Request.Cookies["forSession"];
    return View();
}

Odwolania: 4
public void SetCookie(string key, string value, int? numberOfSeconds = null)
{
    CookieOptions option = new CookieOptions();
    if (numberOfSeconds.HasValue)
        option.Expires = DateTime.Now.AddSeconds(numberOfSeconds.Value);
    Response.Cookies.Append(key, value, option);
}
```

```
@{
    ViewData["Title"] = "Set cookie";
}

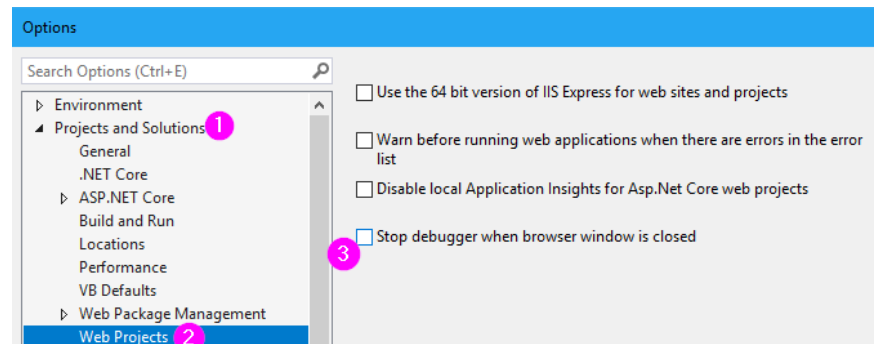
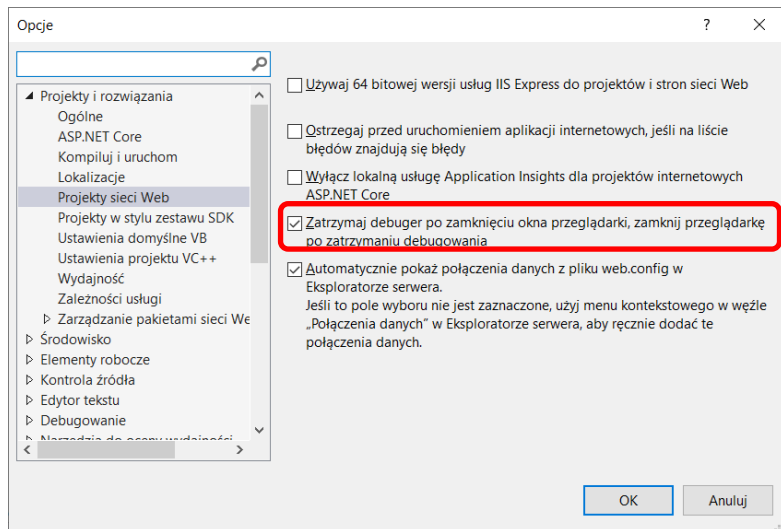
<div class="text-center">
    <h1 class="display-4">Set cookies</h1>
    <h2>Done.</h2>
</div>
```

```
@{
    ViewData["Title"] = "Show Cookies";
}

<div class="text-center">
    <h1 class="display-4">Show cookies</h1>
    <ul>
        <li>Short : @ViewData["short"]</li>
        <li>Long : @ViewData["long"]</li>
        <li>Very Long : @ViewData["veryLong"]</li>
        <li>For session: @ViewData["forSession"]</li>
    </ul>
</div>
```

# Uwaga techniczna - debugowanie

- Domyślne ustawienie Visual Studio powoduje, że zatrzymanie serwera powoduje zamknięcie przeglądarki, natomiast zamknięcie przeglądarki – wyłączenie serwera.
- Można tą opcję wyłączyć w Narzędzia->Opcje a dalej jak na poniższych zrzutach (angielska wersja z <https://stackoverflow.com/questions/40729535/how-to-stop-browser-closing-automatically-when-you-stop-debugging-on-vs-2017>)
  - ang. Tools -> Options
- Pozwoli to zaobserwować różnicę w działaniu ciasteczka sesyjnego i długoterminowego



# Obserwacja działania

- 1) Show Cookies
- 2) Set Cookies
- 3) Show Cookies
- 4) Odczekanie 15 sekund
- 5) Show Cookies
- 6) Odczekanie 90 sekund
- 7) Show Cookies
- 8) Zamknięcie przeglądarki (bez wyłączenia serwera)
- 9) Otwarcie przeglądarki i wejście na Show Cookies

1 Show cookies

Short :  
Long :  
Very Long :  
For session:

2 Set cookies  
**Done.**

3 Show cookies

Short : Short Cookie is active  
Long : Long Cookie is active  
Very Long : Very long Cookie is active  
For session: Cookie for session is active

5 Show cookies

Short :  
Long : Long Cookie is active  
Very Long : Very long Cookie is active  
For session: Cookie for session is active

7 Show cookies

Short :  
Long :  
Very Long : Very long Cookie is active  
For session: Cookie for session is active

9 Show cookies

Short :  
Long :  
Very Long : Very long Cookie is active  
For session:

# Konfigurowanie sesji na serwerze

- Aby skorzystać z sesji należy ją skonfigurować w `Program.cs`
- Dane powiązane z sesją danego użytkownika (słowniki) są najczęściej przechowywane w pamięci ulotnej (RAM), ale mogą również być przechowywane w bazie danych, czy w chmurze. Serwer działa wielowątkowo, więc dostęp do tego typu danych musi być przygotowany na to. Dlatego aby wszystko dobrze działało trzeba również zarejestrować w kontenerze serwisów obsługę pamięci współbieżnej (`builder.Services.AddDistributedMemoryCache()`).
- Dodając mechanizm sesji do kontenera serwisów można skonfigurować opcje sesji, czyli ciasteczka dla sesji (domyślnie sesja trwa 20 minut).
- Podczas metody definiowania potoku przetwarzania należy użyć sesji w odpowiednim miejscu:
  - po `UseRouting()`
  - przed `UseEndpoints()`

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

builder.Services.AddDistributedMemoryCache();
builder.Services.AddSession(options =>
{
    options.IdleTimeout = TimeSpan.FromMinutes(5);
    options.Cookie.HttpOnly = true;
    options.Cookie.IsEssential = true;
});

var app = builder.Build();
```

```
app.UseHttpsRedirection();
app.UseRouting();

app.UseAuthorization();

app.UseSession();

app.MapStaticAssets();
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=CookieSession}/{action=Index}/{id?}"
).WithStaticAssets();

app.Run();
```

# Sesja

- Zmienne sesyjne w kontrolerze dostępne są poprzez `HttpContext.Session`
- Sesja ogólnie:
  - pozwala wymieniać dane między np. różnymi kontrolerami
  - pozwala zapamiętać dane zalogowanego użytkownika
  - Zapamiętać dane, które są **częściowe** skompletowane:
    - koszyk w sklepie internetowym
    - dane wyprawy kosmicznej w grze zanim skompletujemy całą załogę i sprzęt
    - itp.
- Teoretycznie jedyne dane jakie mogą być zapamiętane muszą być typu `int` lub `string`.
- Aby zapamiętać cokolwiek innego należy dokonać serializacji danej (czyli zamiany na `string`)
  - Serializacja pamięta również obiekty wewnętrzne itd. rekurencyjnie, ale bez zapętleń
  - Można to wykonać na wiele sposobów
  - Obecnie używanym standardem w ASP .Net jest używanie formatu JSON za pomocą pakietu `Newtonsoft.Json`
  - Aby skorzystać z obiektu należy go zdeserializować

**Newtonsoft.Json** przez: James Newton-King  
13.0.3

```
public IActionResult SetSession()
{
    HttpContext.Session.SetString("name", "Jane");
    HttpContext.Session.SetInt32("age", 25);
    HttpContext.Session.SetString("point", JsonConvert.SerializeObject( new Point(2,3)));
    return View();
}

Odwolania: 0
public IActionResult ShowSession()
{
    ViewData["name"] = HttpContext.Session.GetString("name");
    ViewData["age"] = HttpContext.Session.GetInt32("age");
    byte[] arr;
    HttpContext.Session.TryGetValue("point", out arr); // Point is a struct not an object
    if (arr != null)
    {
        Point p = JsonConvert.DeserializeObject<Point>(HttpContext.Session.GetString("point"));
        ViewData["x"] = p.X;
        ViewData["y"] = p.Y;
    }
    return View();
}
```

```
@{
    ViewData["Title"] = "Set session";
}

<div class="text-center">
    <h1 class="display-4">Set session</h1>
    <h2>Done.</h2>
</div>
```

```
@using System.Drawing;
@{
    ViewData["Title"] = "Show Session";
}

<div class="text-center">
    <h1 class="display-4">Show session</h1>
    <ul>
        <li>
            Name: @ViewData["name"]
        </li>
        <li>
            Age: @ViewData["age"]
        </li>
        <li>
            Point.X: @ViewData["x"]
        </li>
        <li>
            Point.Y: @ViewData["y"]
        </li>
    </ul>
</div>
```

# Scenariusz użycia

- 1) Wybranie „Show session”
- 2) Wybranie „Set session”
- 3) Wybranie „Show session”
- 4) Wybranie „Set Cookies”

1

Show session

Name:  
Age:  
Point.X:  
Point.Y:

No cookies in request header  
and response header

4

Set cookies  
Done.

2

Set session  
Done.

No cookies in request header.

```
▼ Response Headers
cache-control: no-cache,no-store
content-encoding: gzip
content-type: text/html; charset=utf-8
date: Tue, 05 Jan 2021 09:56:12 GMT
expires: -1
pragma: no-cache
server: Microsoft-IIS/10.0
set-cookie: .AspNetCore.Session=CfDJ8E6%2BExTVJv1McmDPud4dtM2edbCY1bRptA6V6yc%2B7keh9tfbYG!8Ttb0aP%2B37v4TTIdfJ0eRGtuKn1diMcuDYVUTb6U4I:Qvnqa%2BehYnFmoZMDjQeauqNb0LrM78QNAH6uULG5B3iogiuEfgXiXtexaslxl8arsA0dY; path=/; samesite=x; httponly
vary: Accept-Encoding
x-powered-by: ASP.NET
```

▼ Request Headers

3

Show session

Name: Jane  
Age: 25  
Point.X: 2  
Point.Y: 3

No cookies in response header.

```
▼ Request Headers
:authority: localhost:44387
:method: GET
:path: /Home/ShowSession
:scheme: https
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
accept-encoding: gzip, deflate, br
accept-language: pl,en;q=0.9,en-GB;q=0.8,en-US;q=0.7
cookie: .AspNetCore.Session=CfDJ8E6%2BExTVJv1McmDPud4dtM2edbCY1bRptA6V6yc%2B7keh9tfbYG56f8Ttb0aP%2B37v4TTIdfJ0eRGtuKn1diMcuDYVUTb6U4I8idQvnnqa%2BehYnFmoZMDjQeauqNb0LrM78QNAH6uULG5B3wqog1uEfgXiXtexaslxl8arsA0dY
referer: https://localhost:44387/Home/SetSession
```

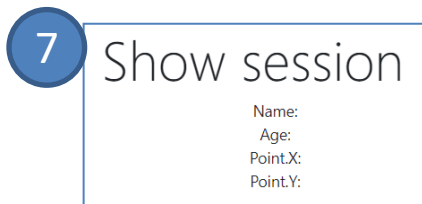
▼ Request Headers

```
:authority: localhost:44387
:method: GET
:path: /Home/SetCookies
:scheme: https
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
accept-encoding: gzip, deflate, br
accept-language: pl,en;q=0.9,en-GB;q=0.8,en-US;q=0.7
cookie: .AspNetCore.Session=CfDJ8E6%2BExTVJv1McmDPud4dtM2edbCY1bRptA6V6yc%2B7keh9tfbYG56f8Ttb0aP%2B37v4TTIdfJ0eRGtuKn1diMcuDYVUTb6U4I8idQvnnqa%2BehYnFmoZMDjQeauqNb0LrM78QNAH6uULG5B3wqog1uEfgXiXtexaslxl8arsA0dY
referer: https://localhost:44387/Home/ShowSession
```

▼ Response Headers

```
content-encoding: gzip
content-type: text/html; charset=utf-8
date: Tue, 05 Jan 2021 10:03:02 GMT
server: Microsoft-IIS/10.0
set-cookie: short=Short%20Cookie%20is%20active; expires=Tue, 05 Jan 2021 10:03:12 GMT; path=/
set-cookie: long=Long%20Cookie%20is%20active; expires=Tue, 05 Jan 2021 10:04:02 GMT; path=/
set-cookie: veryLong=Very%20long%20Cookie%20is%20active; expires=Tue, 05 Jan 2021 10:13:02 GMT; path=/
set-cookie: forSession=Cookie%20for%20session%20is%20active; path=/
vary: Accept-Encoding
```

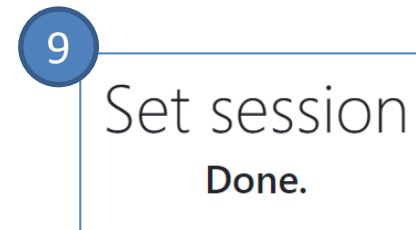
- 5) Odczekanie 10s
- 6) Zamknięcie przeglądarki
- 7) Uruchomienie przeglądarki, wybranie „Show session”
- 8) odczekanie około 50s
- 9) Wybranie „Set session”



No cookies in response header.

▼ Request Headers

```
:authority: localhost:44387
:method: GET
:path: /Home/ShowSession
:scheme: https
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
accept-encoding: gzip, deflate, br
accept-language: pl,en;q=0.9,en-GB;q=0.8,en-US;q=0.7
cookie: veryLong=Very%20long%20Cookie%20is%20active; long=Long%20Cookie%20is%20active
:referrer: https://localhost:44387/Home/ShowSession
```



▼ Request Headers

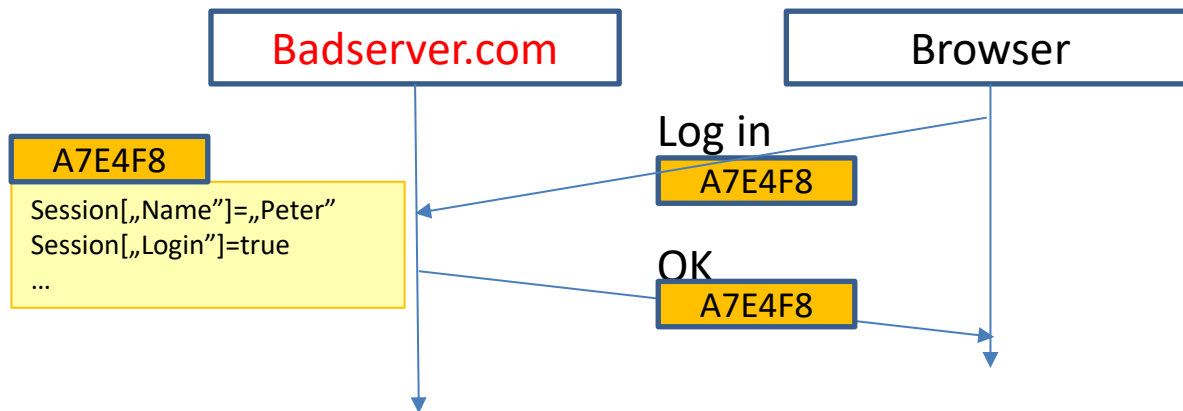
```
:authority: localhost:44387
:method: GET
:path: /Home/SetSession
:scheme: https
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
accept-encoding: gzip, deflate, br
accept-language: pl,en;q=0.9,en-GB;q=0.8,en-US;q=0.7
cookie: veryLong=Very%20long%20Cookie%20is%20active
:referrer: https://localhost:44387/Home/ShowSession
```

▼ Response Headers

```
cache-control: no-cache,no-store
content-encoding: gzip
content-type: text/html; charset=utf-8
date: Tue, 05 Jan 2021 10:13:54 GMT
expires: -1
pragma: no-cache
server: Microsoft-IIS/10.0
set-cookie: .AspNetCore.Session=CfDJ8E6%2BExTVAdJv1McmDPud4ePZ%2B%2BL52hdnRycWEqFsj9ABs4DNWcgAEr61UJtoHBh%2B10hOhEiUjjxRruZy%2BGOHpssTXha%2BFxocLFaE76jj%2FL0SX0XW6nzagiMbflUo1XCYS18a%2BCmXpSsc0pw0Tqbb65nYCu0p0Jg7YkTUNTbtift; path=/; same-site=lax; httponly
varv: Accept-Encoding
```

# CSRF (Cross-Site Request Forgery) 1/3

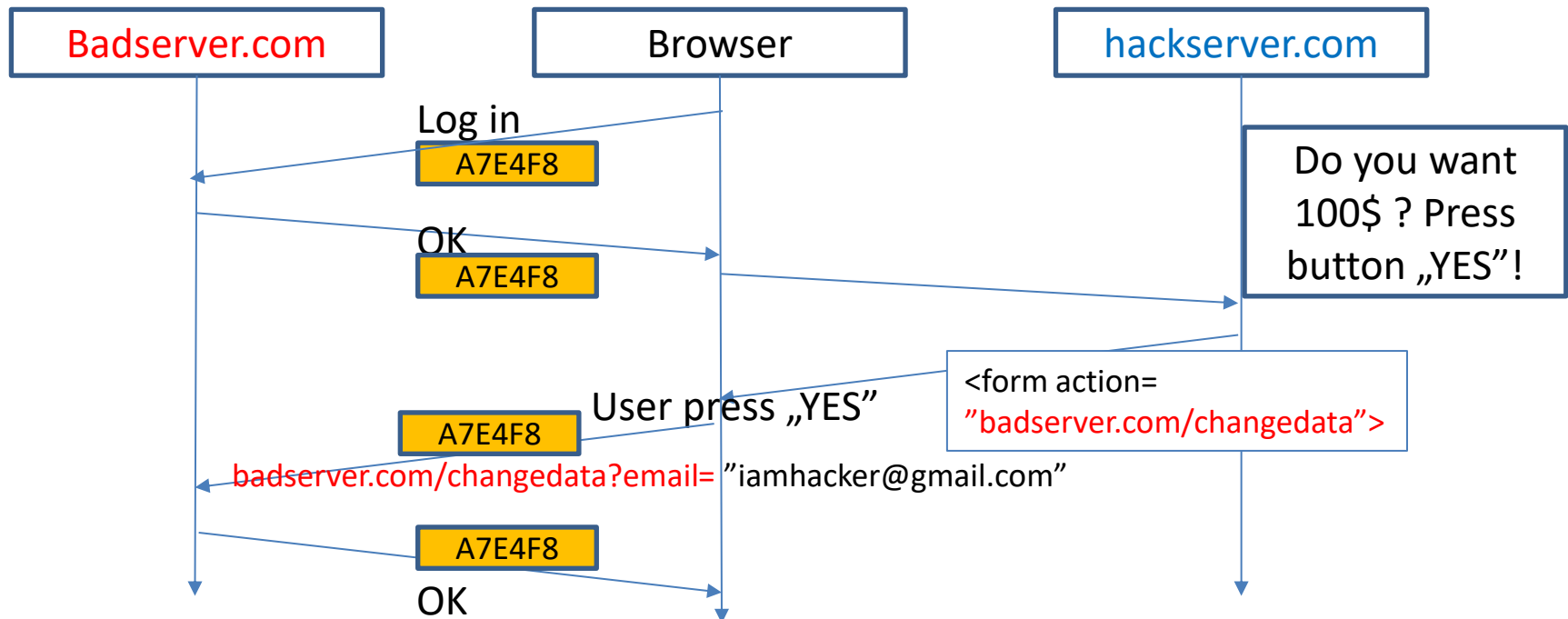
- Cross-site request – żądanie do innej domeny niż strona, z której chcemy go wykonać (np. zwykły link)
- Mechanizm ciasteczek umożliwia atak CSRF. Jeśli przed nim nie zabezpieczymy stron z formularzami możliwy jest poniższy scenariusz (np. na domenie **badserver.com**).
- Chcemy, żeby nieświadomy użytkownik zmodyfikował swoje dane, np. adres email, który służy często do odzyskiwania hasła i logowania. Zmiana adresu polega na wypełnieniu formularza (strona przesłana przez żądanie GET) na stronie **badserver.com/changedata** i wysłania go na ten sam adres (ale przez żądanie POST), ale oczywiście tylko jeśli zmienna sesyjna wskazuje na właściwego użytkownika. Wiedząc to wszystko przygotowany został atak na stronie **hackserver.com**.
  - Użytkownik loguje się na swoje konto na **badserver.com**
  - Przeglądarka zapamiętuje zmienną sesyjną oraz informację, który użytkownik jest zalogowany.





## CSRF (Cross-Site Request Forgery) 2/3

- W tym samym czasie w tej samej przeglądarce użytkownik otwiera stronę [hackserver.com](http://hackserver.com), przygotowaną do ataku na jego konto.
- Znajduje się tam prosty formularz z przyciskiem „Chcesz wygrać 100\$ ?”, jednak w polach ukrytych ma parametry wymagane do formularza zmiany danych przez stronę [badserver.com/changedata](http://badserver.com/changedata) (np. pole „email” jest ustawione na „iamhacker@gmail.com”) i tam **przekierowuje** wysłanie formularza (**cross-site request**).
- Przeglądarka widzi, że będzie żądanie do domeny [badserver.com](http://badserver.com), zatem dodaje w nagłówku **wszystkie** ciasteczka dla tej domeny, w tym to związane z sesją.
- Zmiana danych na serwerze [badserver.com](http://badserver.com) zostanie **dokonana**, bo ciasteczka się zgadzają!
- Użytkownik może zobaczyć informację o zmianie danych (albo kompletnie nic) i wróci zdziwiony do domeny [badserver.com](http://badserver.com).



## CSRF (Cross-Site Request Forgery) 3/3

- Twórca [hackserver.com](http://hackserver.com) nie wie, czy atak mu się udał, ale co jakiś czas próbuje „odzyskiwać” hasło do konta `iamhacker@gmail.com`
- Zabezpieczenie:
  - Dla każdego formularza wysłanego przez GET dodawanie **ukrytego pola (tokena)** np. `antyForgery` z losową wartością, które musi się zgadzać, gdy dane z formularza wrócą z żądaniem POST.
  - Dla pamiętania wartości tokena można wykorzystać ciasteczko
  - Powyższy atak się nie powiedzie, bo karta/JavaScript ma dostęp **tylko do danych swojej karty** oraz **ciasteczek** (odpowiednio skonfigurowanych) z **oryginalnej domeny** danej strony WWW.
- Atak CSRF ma wiele odmian: z użyciem skryptu Javy, ze spreparowanym obrazkiem itp. W każdym przypadku powyższe zabezpieczenie jest wystarczające

# Token przeciw atakowi CSRF w ASP .Net

- Dla tokena przygotowane jest też odpowiednie ciasteczko bez terminu ważności
  - Wystarczy jeden token i jedno ciasteczko na całą sesję

```
// POST: Students/Create
// To protect from overposting attacks, enable the specific properties you want to bind to, for
// more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task Create(
    [Bind("Id,Index,Name,Gender,Active,DepartmentId,Year,Month,Day")] StudentCreateViewModel studentView)
{
    if (ModelState.IsValid)
    {
        // ...
    }
}
```

https://localhost:44320/Students/Create

WebAppForEntityFrameworkDemo

## Create Student

Index

Last Name

```
▼ Response Headers
cache-control: no-cache, no-store
content-encoding: gzip
content-type: text/html; charset=utf-8
date: Tue, 05 Jan 2021 10:34:41 GMT
pragma: no-cache
server: Microsoft-IIS/10.0
set-cookie: .AspNetCore.Antiforgery.2dSpfZQiGQQ=CfDJ8E6-ExTVAadJv1MCmDPud4d3zdIOWKISm2NZL_071m4W
Ej1HD6ntbP4f9gNQNN-jgswWwennbMPmyxenXY2sHIdAsF
T85J8b14XaXJ220X7MBPdhDHLDr-hWmPBwKqkPRiCvOYQ
GBeZ8Zhniuo-Q; path=/; samesite=strict; httpOnly
```

```
<div class="form-group">
  <label class="control-label" for="Day">Day</label>
  <input class="form-control" type="number" data-val="true" data-val-range="The field Day must be
  <span class="text-danger field-validation-valid" data-valmsg-for="Day" data-valmsg-replace="tru
  </div>
  <div class="form-group">
    <input type="submit" value="Create" class="btn btn-primary" />
  </div>
  <input name="__RequestVerificationToken" type="hidden" value="CfDJ8E6-ExTVAadJv1MCmDPud4fqfnDIRHwburUgc
</div>
</div>
<div>
  <a href="/Students">Back to List</a>
</div>
```

## Inne wybrane elementy ciasteczka

- Właściwość `max-age=`*max-age-in-seconds*, zamiast `expires`
- Właściwość `samesite`:
  - `lax`: pozwala blokować większość ataków CSRF, natomiast dla prostych linków (`href=„http://mysuperweb.com/Student/Create”`) w dokumentach na stronach nie z domeny ciasteczka będą wysyłane przez przeglądarkę
  - `strict`: nawet dla prostych linków (jak dla opcji `lax`) żadne ciasteczka nie zostaną dołączone przez przeglądarkę dla żądań cross-site.
  - `none`: brak ograniczeń na wysyłanie ciasteczek dla cross-site requests
- Właściwość `HttpOnly`:
  - `true`: ciasteczka można używać tylko przy generowaniu żądania HTTP, brak dostępu ze skryptu/CSS itp.
  - `false`: JavaScript może mieć dostęp
    - łatwo spreparować „obrazek”, który będzie skryptem np. PHP odbierającym ciasteczko w query string-u.

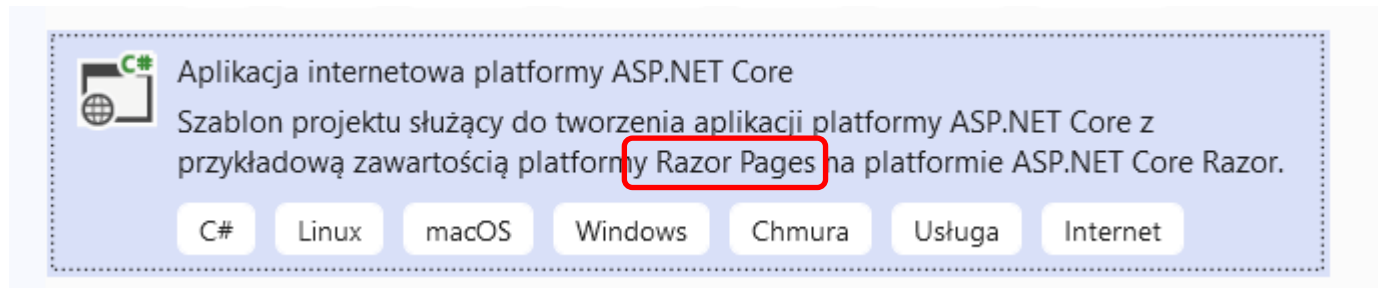
## Podsumowanie

- Nie wszystkie szczegóły nt. ciasteczek czy też ich użycia zostały przedstawione.
- Dużo istotnych szczegółów można znaleźć na stronie: <https://docs.microsoft.com/pl-pl/aspnet/core/fundamentals/app-state?view=aspnetcore-9.0>
- **W prawie europejskim** użytkownik musi wyrazić **zgode** na przechowywanie ciasteczek dla naszej strony WWW.
- Konfiguracja przeglądarki może spowodować, że ciasteczko sesyjne nie zawsze jest usuwane po zamknięciu przeglądarki (Firefox).
  - Celem odtworzenia stanu okienka/zakładek i umożliwienia kontynuacji korzystania z przeglądarki.

# Strony Razor-owe

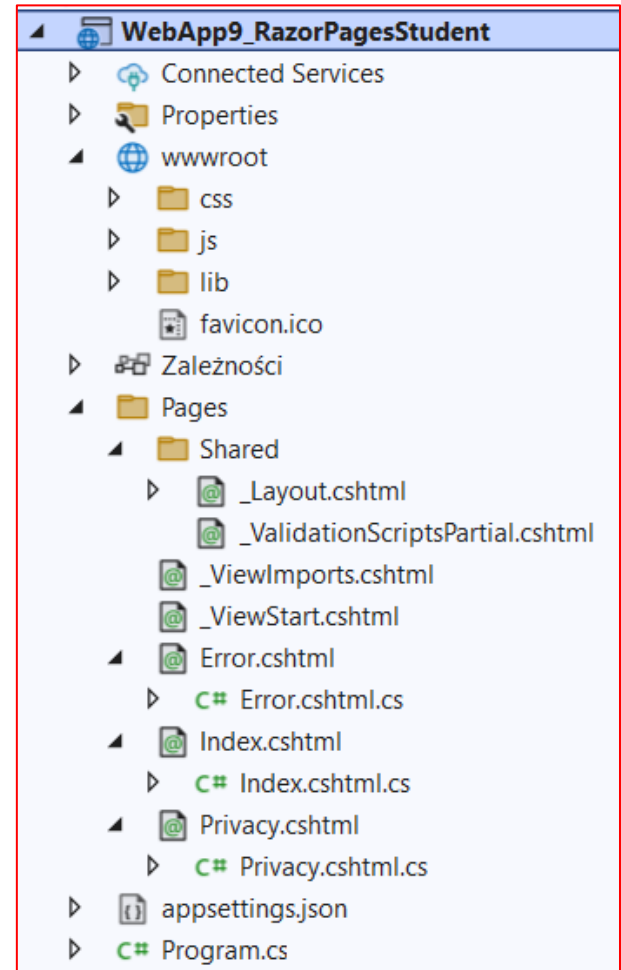
# Strony Razor – ogólna idea

- **Strony** Razor – Razor **Pages** (w MVC są **widoki** Razor = Razor **Views**)
- Zamiast rozbudowanego wzorca MVC
- Dla prostszych (co do struktury) aplikacji webowych
- Minimum dla każdej strony: wzorzec strony i klasa z modelem strony i metodą `OnGet()`
- Brak kontrolera z akcjami, które wybierają sobie **widok** Razor do wyświetlenia
- Uproszczony routing:
  - Brak reguł routingu znanych z MVC
  - Folder `Pages` strony Razor w projekcie to w zasadzie fragment (korzeń) adresu URL
- Występują dodatkowe opcje data bindingu.
- Występują inne opcje routingu.



# Strony Razora – struktura powiązań

- Strona Razora składa się z dwóch plików
  - Wzorzec strony
    - np. plik **Index.cshtml**
  - Model strony
    - Np. plik **Index.cshtml.cs** zawierający klasę **IndexModel** dziedziczącą po **PageModel**
- Domyślnym folderem dla stron Razora jest folder **Pages**. Stąd domyślny routing analizuje strukturę tego folderu. Np.:
  - `https://localhost:12345/Index` oznacza zwrócenie strony Razora z lokalizacji w projekcie: `Pages/Index.cshtml`
  - `https://localhost:12345/Student/Add` oznacza zwrócenie strony Razora z lokalizacji w projekcie: `Pages/Student/Add.cshtml`
  - itd. w głąb podfolderów
- Znaczenie plików `_ViewImports.cshtml`, `_ViewStart.cshtml` oraz folderu `Shared` w zasadzie takie samo jak dla MVC
- Podobnie foldery `Properties`, `wwwroot`, `Zależności`, czy też pliki `appsettings.json`, `Program.cs`.
- Konfiguracja serwisów polega na wykonaniu metody `AddRazorPages()`
- Powyżej opisany sposób mapowania zapewniony jest poprzez metodę `MapRazorPages()`:



```
// Add services to the container.  
builder.Services.AddRazorPages();
```

```
app.MapStaticAssets();  
app.MapRazorPages()  
    .WithStaticAssets();  
  
app.Run();
```



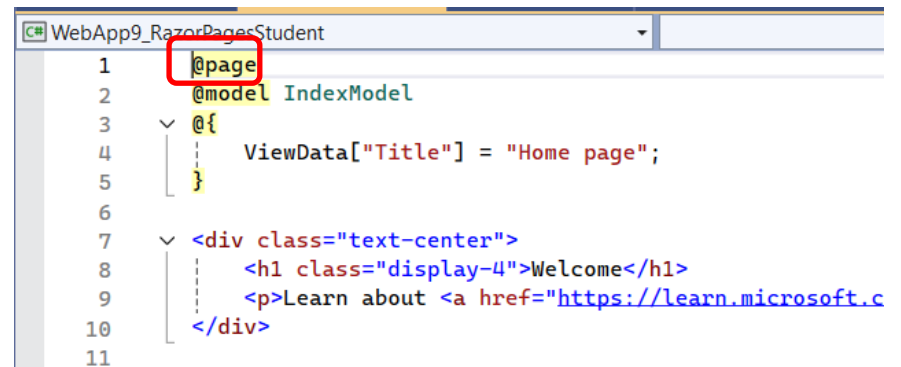
# Pliki wzorca i modelu strony

- Strona wzorca CSHTML zaczyna się od linii z poniższym kodem (który odróżnia **stronę** Razora od **widoku** Razora):  
`@page`
- Poniżej przykład automatycznie wytworzonej klasy modelu strony i wzorca strony dla strony /Index (klasa `IndexModel` w pliku `Index.cshtml.cs`, wzorzec strony w pliku `Index.cshtml`)
- Ogólnie dla Pages/<Path>/<PageName>:
  - Klasa modelu `<PageName>Model` w pliku `<PageName>.cshtml.cs`
  - Wzorzec strony w pliku `<PageName>.cshtml`
- W modelu strony, czyli klasie dziedziczącej po `PageModel` jest metoda **void** `OnGet()`, która jest wywoływana zanim Razor zacznie analizować powiązany wzorzec strony
  - Metoda `OnGet()` zostaje wywołana dla zapytania GET na adres powiązany ze wzorcem strony (np. /Index)
- Metodą `OnGet()` nie musi być typu **void**, może być np. `ActionResult`.

```
public class IndexModel : PageModel
{
    private readonly ILogger<IndexModel> _logger;

    Odwołania: 0
    public IndexModel(ILogger<IndexModel> logger)
    {
        _logger = logger;
    }

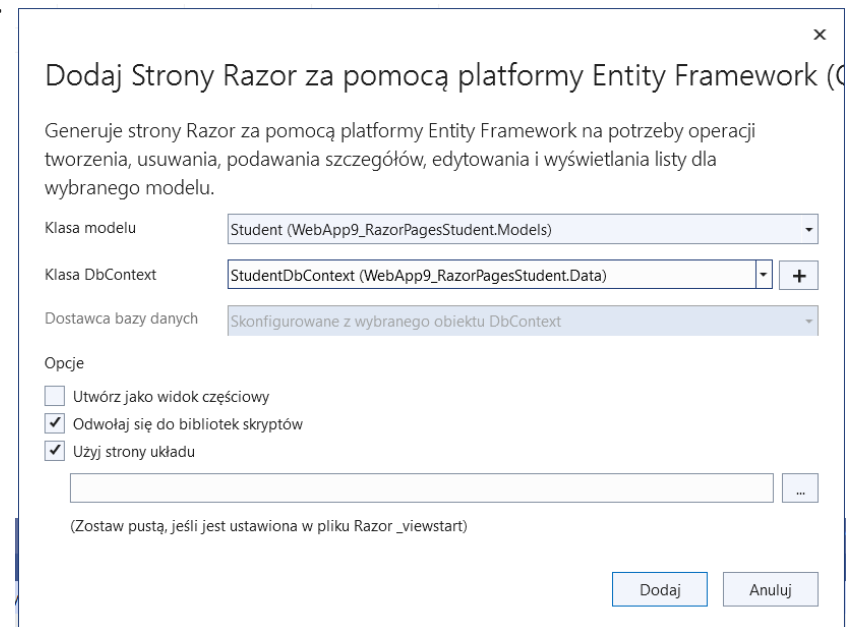
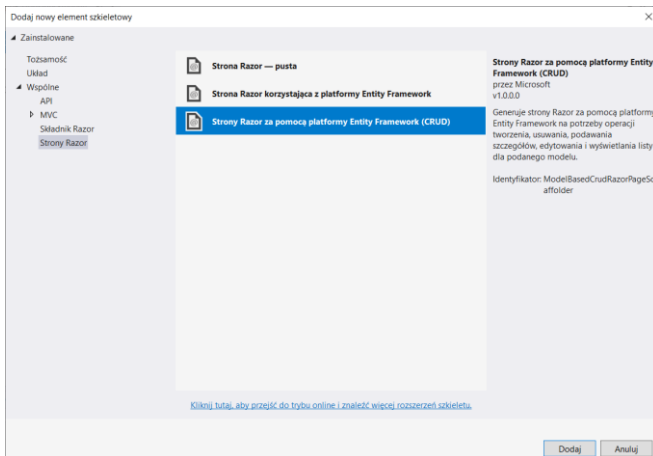
    Odwołania: 0
    public void OnGet()
    {
    }
}
```



```
1 @page
2 @model IndexModel
3 @{
4     ViewData["Title"] = "Home page";
5 }
6
7 <div class="text-center">
8     <h1 class="display-4">Welcome</h1>
9     <p>Learn about <a href="https://learn.microsoft.c
10 </div>
11
```

# Generator kodu dla stron Razor-a

- Stworzyć folder `/Pages/Students`
- Umieszczając kursor myszki na folderze `Students` oraz klikając PPM wybrać Dodaj -> Strona Razor... można, analogicznie do architektury MVC, wygenerować albo stronę dla pojedynczej operacji dla wybranego modelu danych albo wszystkie operacje CRUD.
  - Oczywiście wcześniej trzeba przygotować **model danych** oraz **klasę kontekstu**, dodać **connection string**, wykonać odpowiednie **wstrzyknięcia** (jak również później **migracje i aktualizację** bazy danych)
  - Działania z pakietem Entity Framework przebiegają tak samo jak dla MVC
- **Uwaga:** nazwa folderu powinna być inna niż klasy modelu (`Students`, a nie `Student`), gdyż wytworzona zostaje przestrzeń nazw `WebAppRazorPagesStudents.Pages.Students`, zamiast `WebAppRazorPagesStudents.Pages.Student`, która będzie generować błędy kompilacji spowodowane konfliktem z nazwą klasy `Student`.



# Powstałe modele i wzorce stron

- Wytworzone zostaną wzorce stron i modele stron dla wszystkich operacji CRUD oraz dla strony Index.
- Jest w nich kod analogiczny jak w akcjach kontrolera w architekturze MVC.
- Nie ma wprost zapisanego zabezpieczenia przed CSRF, ale jest on w tag-helperze dla formularza.
  - <https://docs.microsoft.com/pl-pl/aspnet/core/razor-pages/?view=aspnetcore-9.0&tabs=visual-studio#xsrf>
- Pojawia się adnotacja związana z model bindingiem dla żądań POST - [BindProperty].
  - Ale działa również model binding dla parametrów OnGet, OnPost() itd, czyli zamiast adnotacji [BindProperty] można stworzyć metodę OnPostAsync(Student st), a wewnątrz przepisać st do właściwości Student jeśli jest taka potrzeba.
- Zamiast **return** View() jest **return** Page().
- Zamiast RedirectToAction() używane jest RedirectToPage() z podaniem URL.
- Scenariusz użycia: analiza powstałego kodu

## XSRF/CSRF and Razor Pages

Razor Pages are protected by **Antiforgery validation**. The **FormTagHelper** injects antiforgery tokens into HTML form elements.

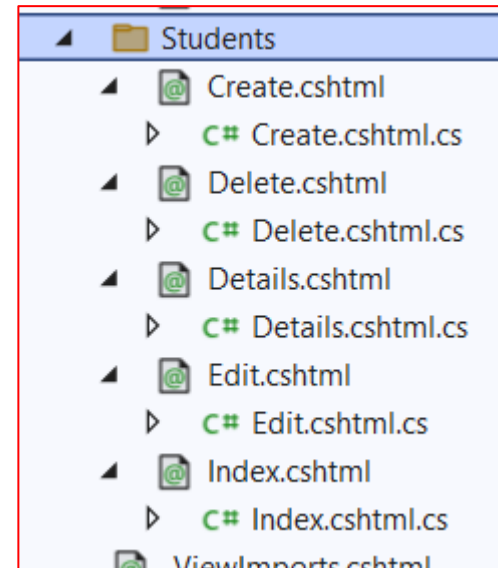
```
Odwołania: 0
public IActionResult OnGet()
{
    return Page();
}

[BindProperty]
Odwołania: 18
public Student Student { get; set; } = default!;
```

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

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

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



# Uzupełnienia kodu

- Wygenerowany kod należy zmodyfikować ze względu na typ wyliczeniowy dla operacji Create i Edit.
- Warto dopisać import w `_ViewImports.cshtml` dla przestrzeni nazw studenta.
- Wytworzona strona html dla powyższych operacji zawiera Token przeciw atakowi CSFR.

```
<div class="form-group">
  <label asp-for="Student.Gender" class="control-label"></label>
  <select asp-for="Student.Gender" class="custom-select" asp-items="Html.GetEnumSelectList<Gender>()">
    <option value="">Please select</option>
  </select>
  <span asp-validation-for="Student.Gender" class="text-danger"></span>
</div>
```

```
@using WebApp9_RazorPagesStudent
@using WebApp9_RazorPagesStudent.Models
@namespace WebApp9_RazorPagesStudent.Pages
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

```
<div class="form-group">
  <label asp-for="Student.Gender" class="control-label"></label>
  <select asp-for="Student.Gender" class="custom-select" asp-items="Html.GetEnumSelectList<Gender>()">
  </select>
  <span asp-validation-for="Student.Gender" class="text-danger"></span>
</div>
```

```
101 <span class="text-danger field-validation-valid" data-valmsg-for="Student.Gender" data-val="true"></span>
102 </div>
103 <div class="form-group">
104   <span class="text-danger"></span>
105   <label class="control-label" for="Student_BirthDate">BirthDate</label>
106   <input class="form-control" type="datetime-local" data-val="true" data-valmsg-for="Student_BirthDate" data-val="true">
107   <span class="text-danger field-validation-valid" data-valmsg-for="Student_BirthDate" data-val="true"></span>
108 </div>
109 <div class="form-group">
110   <input type="submit" value="Save" class="btn btn-primary" />
111 </div>
112 <input name="__RequestVerificationToken" type="hidden" value="CfDJ8J1_zDLUen9J" />
113 </div>
114 </div>
```

Name	Headers	Preview	Response	Initiator	>>
⌵ Edit?id=1					
bootstrap.min.css					
site.css					
jquery.min.js					
bootstrap.bundle.min.js					
site.js?v=4q1jwFhaPaZgr8WA...					
jquery.validate.min.js					
jquery.validate.unobtrusive.mi...					
data:image/svg+xml,...					
:path: /Students/Edit?id=1					
:scheme: https					
accept: text/html,application/xhtml+xml,application/javascript;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9					
accept-encoding: gzip, deflate, br					
accept-language: pl,en;q=0.9,en-GB;q=0.8,en-US;q=0.7					
cookie: .AspNetCore.Antiforgery.JhfGAY6yjjc=CfDJ: xTVAadJv1MCmDPud4fqWkjWIZsn-v3WKjH1LqJnJjaxxnZdlwC0JIcRjqX1uC990iVJyue8JJIQ5pN5pvB6f2121-G_e3nnwA'OH1eyyw7sgy96kZJExwxXhK1QQ5PD-KD1QL4vh4PzI					
referer: https://localhost:44310/Students					
sec-fetch-dest: document					

## Podsumowanie części 1

- Adnotacje walidujące będą działać po zainstalowaniu pakietu

`Microsoft.AspNetCore.Mvc.DataAnnotations`

- Dotyczy to też podsumowania walidacji
- Tak samo działa większość tag-helperów dla stron Razor
- Działa sprawdzanie modelu przez `ModelState.IsValid`

# Routing – elementy konfiguracji

W przypadku stron Razora (ale też innych architektur) mogą się przydać automatyczna zmiana linku URL na wersję z małymi literami. Należy to wykonać w konfiguracji opcji routingu:

```
Services.Configure<RouteOptions>(options =>...)
```

- Używanie małych liter w adresie URL (do znaku ?) przy tworzeniu linku, mimo że w linku na stronie Razora użyto wielkich liter: `options.ToLowerCaseUrls=true;`
- Używanie małych liter w query string (za znakiem ?) przy tworzeniu linku, mimo że w `asp-route-  
<parameter>` użyto dużych liter (np. `asp-route-ID="@item.id"`):  
`options.ToLowerCaseQueryString=true;`
  - Nie działa bez ustawienia poprzedniej opcji na `true`!
- Dodanie końcowego `'/'` w adresie przy tworzenie linku: `options.AppendTrailingSlash=true;`
- Ostatecznie np.:
  - `students/details/?id=1`
- Zamiana na małe litery będzie wykonana nawet, gdy w kodzie będzie adres z dużymi literami.
- W adresach lepiej wyglądają małe litery, ale standard C# zaleca wielkie litery na początku nazw plików, klas, właściwości itp., co powodowałoby możliwość generacji błędów (nie działające linki z powodu niewłaściwej wielkości pierwszej litery)
- Zamiast `asp-controller` i `asp-action` używa się tag-helper `asp-page`.

```
options.UseSqlServer(builder.Configuration.GetConnectionString("MyDB"));
builder.Services.Configure<RouteOptions>(options =>
{
    options.ToLowerCaseUrls = true;
    options.ToLowerCaseQueryString = true;
    options.AppendTrailingSlash = true;
});
var app = builder.Build();
```

```
<li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-page="/Privacy">Privacy</a>
</li>
<li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-page="/Students/Index">Students</a>
</li>
```

<https://localhost:44310/privacy/>

<https://localhost:44310/students/>

# Routing – dodanie parametrów

Przykład dla działania `index` i `details`.

- Nie ma ogólnych reguł routingu, ale można dodać reguły związane z data bindingiem
- Działa większość tag-helperów np. `asp-route-<name>` (np. `asp-route-id`)
- Standardowo działa data binding dla parametrów żądania GET
  - Tworzony URL dla żądania GET wstawia parametry jako query string, czyli np.  
`https://localhost:44310/students/details?id=1`
- Parametr jako fragment adresu URL (parametr routingu), a nie query string
  - `@page {id} //` na początku strony Razor-a
- Ostatecznie np.:
  - `student/details/1`

```
</td>
<td>
    <a asp-page="./Edit" asp-route-id="@item.Id">Edit</a> |
    <a asp-page="./Details" asp-route-id="@item.Id">Details</a> |
    <a asp-page="./Delete" asp-route-id="@item.Id">Delete</a>
</td>
```

```
Details.cshtml
WebApp9_RazorPagesStudent
1 @page
2 @model WebApp9_RazorPagesStudent.Pages.Students.DetailsModel
3
4 @{
5     ViewData["Title"] = "Details";
6 }
7
8 <h1>Details</h1>
```

```
Odwołania: 0
public async Task<IActionResult> OnGetAsync(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    Student = await _context.Student.FirstOrDefaultAsync(m => m.Id == id);
```

## Różne sposoby data binding z żądania

- Aby działał adres `/students/details/` bez parametru należy w `OnGet (int id=1)` dodać wartość domyślną parametru
- Gdy potrzeba użyć parametru z URL w modelu strony należałoby stworzyć właściwość np. jak poniżej i przypisać do niej wartość w metodzie `OnGet ()`.
- Jednak można od razu powiązać właściwość jako fragment URL będący parametrem dla `OnGet`.
- Standardowo adnotacja `BindProperty` wiąże dane z żądania POST. Jeśli jest potrzeba, aby wiązana następowało również w adresie URL żądania GET należy dopisać w adnotacji (`SupportsGet = true`).

```
[BindProperty(SupportsGet = true)]
Odwolania: 2
public int? Id { get; set; }
//      public async Task<IActionResult> OnGetAsync(int? id)
Odwolania: 0
public async Task<IActionResult> OnGetAsync()
{
    if (Id == null)
    {
        return NotFound();
    }

    Student = await _context.Student.FirstOrDefaultAsync(m => m.Id == Id);

    if (Student == null)
    {
        return NotFound();
    }
    return Page();
}
```



# Ograniczenia na parametr routingu

- Przy podawaniu nazwy parametru routingu można narzucić oczekiwany typ (wpisując go za dwukropkiem):
  - **int, decimal, float, double, bool, long, datetime, guid**
- Można dodawać kolejne ograniczenia:
  - Dla liczb: `min(value), max(value), range(min, max)`
  - Dla string: `alpha, minlength(value), maxlength(value), length(min, max), length(value), regex(exp)`
- Ograniczenie łączymy dostawiając kolejny dwukropek za poprzednim ograniczeniem
- Za pomocą nawiasów klamrowych można powiązać dane z wieloma właściwościami
- Pamiętać, aby ewentualny znak zapytania (oznaczenie, że parametr jest opcjonalny) użyć za ostatnim ograniczeniem:
- Rozbudowany przykład ze strony: <https://exceptionnotfound.net/how-to-use-routing-in-asp-net-core-3-0-razor-pages/>
- `{**title}` oznacza, że dowolny ciąg znaków do końca URL to title.

```
cshtml Details.cshtml Program.cs Startup.cs Dele
@page "{id:int?}"
@model WebAppRazorPagesStudents.Pages.Students.DetailsModel
```

```
public class DateArticleModel : PageModel
{
    [BindProperty(SupportsGet = true)]
    public int Year { get; set; }

    [BindProperty(SupportsGet = true)]
    public int Month { get; set; }

    [BindProperty(SupportsGet = true)]
    public int Day { get; set; }

    [BindProperty(SupportsGet = true)]
    public string Title { get; set; }

    public DateTime PublishDate { get; set; }
    public void OnGet()
    {
        PublishDate = new DateTime(Year, Month, Day);
    }
}
```

```
@page "/post/{year:int}/{month:int}/{day:int}/{**title}"
@model RoutingAspNetCoreDemo.RazorPages.Areas.Blog.Pages.DateArticleModel
@{
}

<h1>Date Article</h1>

<p>The publish date is @Model.PublishDate.ToString("yyyy/MM/dd")</p>
```

# Własna reguła ograniczania routingu

- Można utworzyć własną regułę routingu
- Przykład dla parzystych id
- Implementacja interfejsu `IRouteConstraint`
- Nową regułę trzeba zarejestrować w `startup.cs`
  - Jako element słownika
- Jeśli reguła zwraca fałsz, to:
  - We wzorcu strony **nie zostanie** wygenerowany link
  - Po wpisaniu URL nie spełniającego ograniczeń błąd 404, page not found
- Ostatni parametr zwraca informację, czy jest używany podczas odbierania żądania, czy podczas tworzenia linku.
- Przykład z (Pragim/kudvenkat): <https://www.youtube.com/watch?v=5kbF4dAD14I>

```
namespace RazorPagesTutorial
{
    public class EvenConstraint : IRouteConstraint
    {
        public bool Match(HttpContext httpContext, IRouter route, string routeKey,
            RouteValueDictionary values, RouteDirection routeDirection)
        {
            int id;

            if (Int32.TryParse(values["id"].ToString(), out id))
            {
                if (id % 2 == 0)
                {
                    return true;
                }
            }

            return false;
        }
    }
}
```

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<RouteOptions>(options =>
    {
        options.ConstraintMap.Add("even", typeof(EvenConstraint));
    });
}
```

@page "/employees/view/{id:even}"

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddRazorPages();
    services.AddSingleton<IEmployeeRepository, MockEmployeeRepository>();

    services.Configure<RouteOptions>(options =>
    {
        options.LowercaseUrls = true;
        options.LowercaseQueryStrings = true;
        options.AppendTrailingSlash = true;
        options.ConstraintMap.Add("even", typeof(EvenConstraint));
    });
}
```

# Dwa formularze i metody dla nich

- Na stronie może być wiele formularzy
  - Np. mały formularz u góry w rogu celem logowania się do serwisu, lub włączenia filtru
    - Chcemy po wpisaniu danych do tego formularza pozostać na bieżącej stronie
  - Dla MVC Core będą dwie różne akcje
- Dla formularzy na stronach Razora używa się tzw. page-handler, czyli nazwy metody, która ma otrzymać dane z formularza. Domyślnie, dla formularza nazwa ta to `OnPost`, jednak poprzez tag helper `asp-page-handler` można przyporządkować inną metodę.
- Ale jeśli otwiera się potem znowu ten sam wzorec strony, to część danych z model bindingu może nie być właściwie połączonych
- Przykład z (Pragim/kudvenkat): <https://www.youtube.com/watch?v=-6PE4p4gUYQ>

### Handle Multiple Forms in a Razor Page

**Notification Preferences**  
☐ Recieve email notification when my details change  
**Update Notification Preferences**  
**Form 1**  
**OnPostUpdateNotificationPreferences()**

**Edit**  
Name: John  
Email: john@pragimtech.com  
Department: IT  
Photo: Click here to change photo  
**Update** **Cancel**  
**Form 2**  
**OnPost**

```
<form method="post" asp-page-handler="UpdateNotificationPreferences">  
    @*This is Form 1*@  
</form>  
  
<form method="post" asp-page-handler="OnPost">  
    @*This is Form 2*@  
</form>
```

```
public void OnPost(Employee employee)  
{  
    // Code to update employee data  
}  
  
public void OnPostUpdateNotificationPreferences()  
{  
    // Code to update notification preferences  
}
```

# Jak do URL wstawiany jest handler

- Wysyłając formularz z niedomyślnym handlerem nazwa handlera przekazywana jest jako parametr `handler` w query string
- Tworząc odpowiednią regułę routing przy `@page` można użyć parametru `handler`, aby umieścić go jako parametr routingu
- Dla `OnPost` analogiczny URL (ale jeśli jest `redirect`, to włączając śledzenie w przeglądarce dopiero można to zaobserwować)

## How is the Handler Name passed

By default the handler name is passed in the URL as a **query string parameter**



`https://localhost:12345/employees/edit/2/?handler=updatenotificationpreferences`

To pass the handler name as a **route parameter**

`https://localhost:44383/employees/edit/2/updatenotificationpreferences/`

`@page "{id:min(1)}/{handler?}"`

```
Edit.cshtml"
@page "{id:min(1)}/{handler?}"
@model RazorPagesTutorial.Pages.Employees.EditModel
@{
    ViewData["Title"] = "Edit";
    var photoPath = "~/images/" + (Model.Employee.PhotoPath ??
```

Name	Status	Type	Initiator	Size	Waterfall
 employees/	200	document	44383...	4.5 KB	

`https://localhost:44383/employees/edit/2/onpost/`

## Adnotacja [ TempData ] dla stron Razora



- W ramach stron Razora też istnieje słownik TempData z taką samą funkcjonalnością jak dla MVC
- Dane z TempData można powiązać z właściwością poprzez adnotację [ TempData ].
  - Nastąpi data binding dla elementu słownika TempData
- Przykład z (Pragim/kudvenkat): <https://www.youtube.com/watch?v=BZQH9Bg9z6s>

**TempData attribute to access TempData**

```
Details.cshtml.cs
[TempData]
public string Message { get; set; }
```

```
Details.cshtml
@if (Model.Message != null)
{
    <div class="col-sm-8">
        <div class="alert alert-primary">
            @Model.Message
        </div>
    </div>
}
```

**DOWNLOAD** To download all our videos please visit [pragimtech.com/downloadcourses/](http://pragimtech.com/downloadcourses/)

3  facebook.com/pragimtech  twitter.com/kudvenkat

## Widok częściowy - inaczej

- Widok częściowy nie jest stroną Razora, ale w pierwszej linii może oczekiwać modelu. Dodatkowo, poza modelem, może używać słownika ViewData.
- Najlepiej wykorzystać do tego poniższy tag-helper `partial` (od ASP.NET Core 2.1).

```
1 @page
2 @model RazorPagesTutorial.Pages.Employees.IndexModel
3 @{
4     ViewData["Title"] = "Index";
5     ViewData["ShowButtons"] = true;
6 }
7
8 <style>
9     .btn {
10         width: 75px;
11     }
12 </style>
13
14 <h1>Employees</h1>
15
16 <div class="card-deck">
17     @foreach (var employee in Model.Employees)
18     {
19         <partial name="_DisplayEmployeePartial" model="employee" view-data="ViewData" />
20     }
21 </div>
22
23
```

```
@model Employee
@{
    var photoPath = "~/images/" + (Model.PhotoPath ?? "noimage.jpg");
    bool showButtons = (bool)ViewData["ShowButtons"];
}

<div class="card m-3" style="min-width: 18rem; max-width: 30.5%;">
    <div class="card-header">
        <h3>@Model.Name</h3>
    </div>

    @if (showButtons)
    {
        <div class="card-footer text-center">
            <a asp-page="/Employees/Details" asp-route-ID="@Model.Id"
                class="btn btn-primary m-1">View</a>

            <a asp-page="/Employees/Edit" asp-route-ID="@Model.Id"
                class="btn btn-primary m-1">Edit</a>

            <a asp-page="/Employees/Delete" asp-route-ID="@Model.Id"
                class="btn btn-danger m-1">Delete</a>
        </div>
    }
</div>
```



# Komponent widoku ViewComponent

- Klasa modelu dziedziczy po ViewComponent (plik .cs)
  - Powinna mieć końcówkę -ViewComponent
- Posiada widok Razorowy o tej samej nazwie, ale z rozszerzeniem .cshtml
  - Ale może mieć też inne widoki
- Komponent nie odpowiada na żądania HTTP, są stworzone dla innych stron Razor-a (podobnie jak widoki częściowe)
  - Posiada klasę modelu, ale nie z metodami OnGet () itp.
- Klasa modelu obsługuje jedną z metod:
  - public IActionResult Invoke()
  - public Task< IActionResult> InvokeAsync()
- Mimo zastosowania wersji synchronicznej wywołuje się ją asynchronicznie poprzez `@await Component.InvokeAsync("HeadCount")`
  - Oznacza wywołanie Invoke dla HeadCountViewComponent

```
public class HeadCountViewComponent : ViewComponent
{
    private readonly IEmployeeRepository employeeRepository;

    public HeadCountViewComponent(IEmployeeRepository employeeRepository)
    {
        this.employeeRepository = employeeRepository;
    }

    public IActionResult Invoke()
    {
        var result = employeeRepository.EmployeeCountByDept();
        return View(result);
    }
}
```

```
@model IEnumerable<DeptHeadcount>

<h3>Employee Head Count Summary</h3>

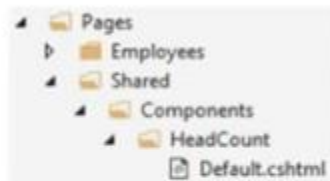
<table class="table table-bordered">
    <thead class="thead-light">
        <tr><th>Department</th><th>Head Count</th></tr>
    </thead>
    <tbody>
        @foreach (var deptHeadCount in Model)
        {
            <tr>
                <td>@deptHeadCount.Department</td>
                <td>@deptHeadCount.Count</td>
            </tr>
        }
    </tbody>
</table>
```

## Miejsce na komponenty widoku

- Można też używać w projekcie wzorca MVC
- Komponenty są poszukiwane w odpowiednich folderach stron Razor lub widoków MVC, schemat poniżej

### Razor pages project

➤ **/Pages/Shared/Components/{View Component Name}/{View Name}**



### MVC project

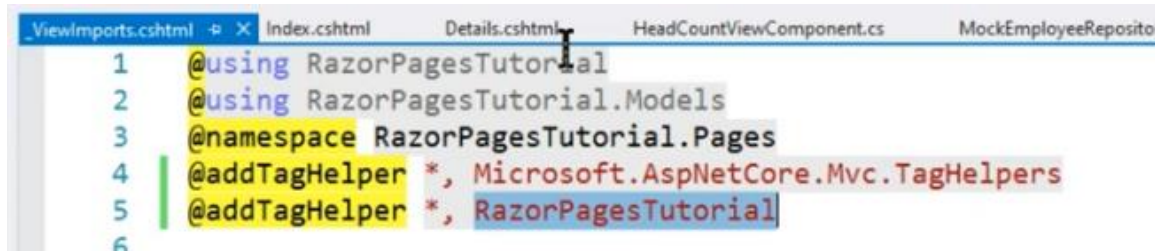
➤ **/Views/{Controller Name}/Components/{View Component Name}/{View Name}**

➤ **/Views/Shared/Components/{View Component Name}/{View Name}**



# Wywołanie tworzenie komponentów widoku

- Wywołanie tworzenia viewComponent za pomocą tag-helperów(jako znacznik) `<vc:...>`
- Trzeba dodać jakby tag helpery z tworzonego projektu (najlepiej w `_ViewImports.cshtml`)
- Użycie Kebab Case (w HTML wielkość liter w znacznikach i atrybutach nie ma znaczenia!):
  - Parametr metody (inne j niż wcześniej) `Invoke (DepartmentName)`



```
1 @using RazorPagesTutorial
2 @using RazorPagesTutorial.Models
3 @namespace RazorPagesTutorial.Pages
4 @addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
5 @addTagHelper *, RazorPagesTutorial
6
```

## Using View Component as a Tag Helper

```
@await Component.InvokeAsync("HeadCount", new { department = Dept.IT })
```

```
<vc:head-count department="Dept.IT"></vc:head-count>
```

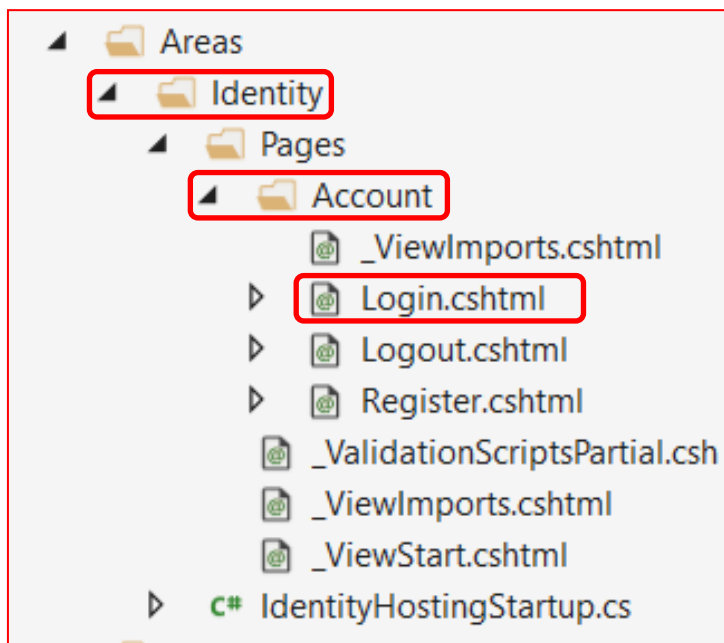
HeadCount	DepartmentName
<pre>&lt;vc:head-count department-name="Dept.IT"&gt;&lt;/vc:head-count&gt;</pre>	

**Kebab Case**

# Obszary - Areas

Folder `Areas` do wydzielania obszarów logicznych dla stron Razora

- Posiadają własne pliki typu `_ViewImports.cshtml` itp., własne `layouts`, komponenty,
- Oprócz folderu `Pages` obszar o danej nazwie może mieć inne foldery: z modelami, kontekstem itp.
- Pozwala odseparować kod zależnie od obszaru, do którego ma służyć: inny do logowania, inny dla administratora do zarządzania itp.
- Mapowanie routingu dla stron Razora obsługuje obszary pomijając nazwy folderów „`Areas`” oraz „`Pages`”
  - dla poniższego przykładu poprawny URL to np. `/identity/account/login`



# Porównanie do MVC

- Nie ma klas kontrolera
- Widok połączony z akcją/akcjami
- Domyślne mapowanie folderów w ramach routing
- Zamiast `return View()`, piszemy `return Page()`.
- Zamiast `RedirectToAction()` używamy `RedirectToPage()`
- Domyślnie do wzorca strony wysyłany model z nim powiązany
- Dodatkowe notacje dla data-bindingu
- Obszary – połączenie logiczne **różnych** funkcjonalności
  - W MVC foldery łączą typy funkcjonalności: wszystkie kontrolery w `Controllers`, wszystkie widoki w `Views`, modele w `Models` itp.

Ale

- Trzeba w **każdym** modelu wstrzykiwać zależności
- Nie można używać `asp-controller`, `asp-action`
  - Za to jest `asp-page-handler`
- Działają inne tag- helpery dla `asp-page`
  - Przekierowanie nie przez nazwę kontrolera, ale link (wada?) np. `RedirectToPage("../Index")` ;
- Działają pozostałe tag-helpery jak `asp-for`, `asp-route-<param>` itd..
- W każdym modelu trzeba tworzyć ewentualne pola, które chcemy przekazać do wzorca strony (np. `Student`).
  - Przez co dostęp we wzorcu strony jest w dłuższym zapisie np. `@Model.Student.Id`
- Oczywiście w ramach jednego projektu można mieszać architektury...
  - Różne foldery przygotowane dla MVC i Razor Pages