

ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej

Aplikacje webowe na platformę .NET

W12 – Pakiet Identity, kontroler API, AJAX

Syllabus

- Autentykacja, autoryzacja
- Uwierzytelnianie w ASP .NET Core
- ASP .Net Core **Identity**
- Autoryzacja oparta na rolach
- Polityki autoryzacji

- Kontroler API
 - ReST , ReSTful
 - Metody HTTP i adresy w typie ReSTful
 - Kontroler API
 - curl
 - Narzędzie Postman
 - Narzędzie Swagger

- Ajax - kod do asynchronicznego ładowania danych z kontrolera API:
 - JavaScript
 - jQuery

Autentykacja, Autoryzacja

- Autentykacja/Uwierzytelnienie: To jest weryfikacja czy użytkownik **jest tym, za którego się podaje**.
 - Czyli problemy tutaj to "Kto to jest?" i "Jak sprawdzić, że to ta osoba?,,
 - Rozwiązywana najczęściej przez parę użytkownik-hasło oraz sesja (ciasteczko dla sesji)
- Autoryzacja: To jest weryfikacja czy użytkownik **ma prawo dostępu do konkretnych usług / zasobów**.
 - Pytania to "Czy userX może przeczytać Y?", "Czy userX może zmienić Z?,,
 - Najczęściej przez nadanie roli użytkownikowi (admin, magazynier, kierownik itp.) i sprawdzenie dla operacji, czy dana rola ma do tej operacji dostęp.

ASP .Net Core Identity

Uwierzytelnianie w ASP .NET Core

- Za pomocą kont zapamiętanych w bazie danych aplikacji
- Za pomocą kont zapamiętanych w chmurze
- Za pomocą kont służbowych:
 - W usłudze Active Directory
 - W usłudze MS Azure Active Directory
 - W usłudze Office 365
- Za pomocą uwierzytelniania Windows.
- Za pomocą uwierzytelniania open source
- Za pomocą kont Facebook, Google i in.
- Za pomocą każdego dowolnego z powyższych.

Można też napisać wszystko od początku, jednak zastosowanie powyższych sposobów pozwala na użycie:

- Gotowych klas do dostępu do konta, roli itp.
- Gotowych klas do zarządzania kontami, rolami itd.
- Adnotacji ułatwiających autoryzację
- Gotowych stron w Razorze do zakładania konta, logowania, modyfikacji danych, odzyskiwania hasła itp.

Informacje dodatkowe

Aplikacja internetowa ASP.NET Core (Model-View-Controller)

Platforma ⓘ

.NET 9.0 (Pomoc techniczna objęta standardowym terminem)

Typ uwierzytelniania ⓘ

Pojedyncze konta

Brak

Pojedyncze konta

Platforma tożsamości firmy Microsoft
Windows

Linux

Typ kompilacji kontenera ⓘ

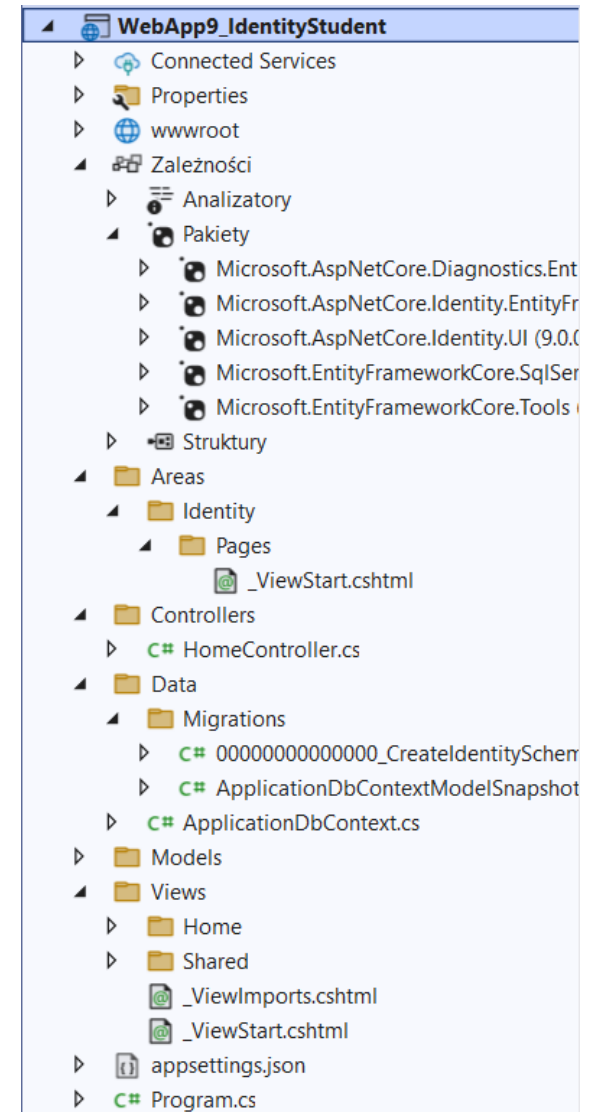
Dockerfile

☒ Nie używaj instrukcji najwyższego poziomu ⓘ

☐ Rejestracja w orkiestracji .NET Aspire ⓘ

Aplikacja szkieletowa z kontami w bazie danych

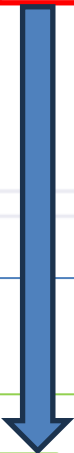
- Dodane zostały pakiety dla EntityFramework oraz pakiet Identity Core, służący do identyfikacji i autoryzacji.
- Oprócz tego przygotowana została **pierwsza migracja**, której zadaniem jest stworzenie tabel potrzebnych do działania Identity Core.
- Ponieważ pakiet IC ma również przygotowane strony razorowe (szkielety) do operacji związanych z logowaniem itp., powstaje też obszar `Identity` (w folderze `Areas`) z jednym widokiem częściowym.
 - Pozostałe strony/widoki są zaszyte w kodzie
 - Jeśli domyślne strony/widoki chcemy zmienić, można je wyekstrahować za pomocą specjalnych generatorów kodu.



Plik appsetting.json

- Generator kodu tworzy connection string, ale z bazą danych o dość losowej nazwie, więc zanim uruchomi się update bazy danych, lepiej tą nazwę zmienić.

```
{
  {...
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=aspnet-WebApp9_IdentityStudent-52208e42-fecb-41a8-97b1-861e701",
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```



```
{
  "ConnectionStrings": {
    "UnivDb": "Server=(localdb)\\mssqllocaldb;Database=UnivWithIdentity;Trusted_Connection=True;MultipleActiveResultSets=true",
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

Kontekst bazy danych

- Kontekst bazy danych dziedziczy po `IdentityDbContext`, w którym jest dostęp do tabel związanych z **identyfikacją i autoryzacją**.
- Oczywiście trzeba go uzupełnić o własne `DbSet`-y.

```
1  using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
2  using Microsoft.EntityFrameworkCore;
3
4  namespace WebApp9_IdentityStudent.Data
5  {
6      Odwołania: 6
7      public class ApplicationDbContext : IdentityDbContext
8      {
9          Odwołania: 0
10         public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
11             : base(options)
12         {
13         }
```


Plik Program.cs 1/2

- Podczas budowania aplikacji w `Main()` dodatkowo pojawia się nowy serwis związany z Identity Core.
- W opcjach jego tworzenia następuje połączenie z kontekstem bazy danych `ApplicationDbContext`.
 - Teoretycznie można te dane pamiętać w pamięci komputera, w innym kontekście itp.
 - Opcje te **należy** potem **uzupełnić** o wiele parametrów zależnych od przyjętej polityki autoryzacji i autentykacji

```
public static void Main(string[] args)
{
    var builder = WebApplication.CreateBuilder(args);

    // Add services to the container.
    var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new InvalidOp
    builder.Services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(connectionString));
    builder.Services.AddDatabaseDeveloperPageExceptionFilter();

    builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = true)
        .AddEntityFrameworkStores<ApplicationDbContext>();
    builder.Services.AddControllersWithViews();

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

Plik Program.cs 2/2

- Powiązanie ciasteczka sesji ze stanem autentykacji i autoryzacji dla konkretnego użytkownika następuje w oprogramowaniu pośredniczącym:
 - czy jest zalogowany,
 - jako kto
 - jakie ma role/uprawnienia
 - itp.
- Odpowiednie serwisy komunikują się z kontekstem (bazą danych) podczas logowania/wylogowania i odczytują jakie dane użytkownik ma uprawnienia.
- Metoda rozszerzająca `UseAuthorization()` musi być wstawiona pomiędzy metody `UseRouting()` i pierwszą metodą związaną z definicjami endpointów (`Map...`, `UseEndpoints()` itp.)

```
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseMigrationsEndPoint();
}
else
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want
    app.UseHsts();
}

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

app.UseAuthorization();

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

app.Run();
```

ASP .Net Core Identity

- ASP .Net Core Identity:
 - Jest interfejsem API obsługującym funkcję logowania w interfejsie użytkownika (UI).
 - Zarządza użytkownikami, hasłami, danymi profilu, rolami, poświadczeniami, tokenami, potwierdzeniami e-mail i nie tylko.
- Użytkownicy mogą utworzyć konto z danymi logowania przechowywanymi w Identity lub mogą korzystać z zewnętrznego dostawcy logowania. Obsługiwani zewnętrzni dostawcy logowania obejmują Facebook, Google, konto Microsoft i Twitter.
- Założenie: klasa `Student` oraz projekt stworzony od początku z możliwością uwierzytelniania (oparty o bazę z EntityFramework)
 - Z wszystkimi krokami do działania z EntityFramework
 - Z kontrolerem i widokami za pomocą generatora kodów dla klasy `Student` w kontekście bazy danych EntityFramework
 - Główna różnica – kontekst bazy danych dziedziczy po `IdentityDbContext` zamiast po `DbContext`

```
public class UniversityDbContext : IdentityDbContext
{
    Odwołania: 7
    public DbSet<Student> Student { get; set; }

    Odwołania: 0
    public UniversityDbContext(DbContextOptions<UniversityDbContext> options)
        : base(options) { }

    Odwołania: 0
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder); // create tables for Identity
        modelBuilder.Seed();
    }
}
```

Nowe tabele w bazie

- Po stworzeniu (update-database) bazy powstaną odpowiednie tabele
- Tabela `AspNetUsers` zawiera dane o użytkownikach (w tym identyfikator i hasło).
- Dwie tabele służą do autoryzacji oparte na rolach (ang. **Role-Based Security**)
 - `AspNetRoles` – role jakie może posiadać użytkownik
 - `AspNetUserRoles` – którzy użytkownicy jakie mają przydzielone role
- Tabele ze słowem `Claim` służą do autoryzacji bazującej na roszczeniach/oświadczeniach (ang. **Claims-Based Security**). Aby ich użyć trzeba stworzyć polityki (policy) autoryzacji umieszczone w kodzie C#
 - Bardziej elastyczne
 - Bardziej czasochłonne
- Pozostałe tabele są do pamiętania danych do autoryzacji z innych źródeł (`AspNetUserLogins`) lub autoryzacją oparta na tokenach (`AspNetUserTokens`)

| dbo.AspNetUsers | |
|------------------------------------------|--|
| Columns | |
| Id (PK, nvarchar(450), not null) | |
| UserName (nvarchar(256), null) | |
| NormalizedUserName (nvarchar(256), null) | |
| Email (nvarchar(256), null) | |
| NormalizedEmail (nvarchar(256), null) | |
| EmailConfirmed (bit, not null) | |
| PasswordHash (nvarchar(max), null) | |
| SecurityStamp (nvarchar(max), null) | |
| ConcurrencyStamp (nvarchar(max), null) | |
| PhoneNumber (nvarchar(max), null) | |
| PhoneNumberConfirmed (bit, not null) | |
| TwoFactorEnabled (bit, not null) | |
| LockoutEnd (datetimeoffset(7), null) | |
| LockoutEnabled (bit, not null) | |
| AccessFailedCount (int, not null) | |

| dbo.AspNetRoles | |
|----------------------------------------|--|
| Columns | |
| Id (PK, nvarchar(450), not null) | |
| Name (nvarchar(256), null) | |
| NormalizedName (nvarchar(256), null) | |
| ConcurrencyStamp (nvarchar(max), null) | |

| dbo.AspNetUserRoles | |
|------------------------------------------|--|
| Columns | |
| UserId (PK, FK, nvarchar(450), not null) | |
| RoleId (PK, FK, nvarchar(450), not null) | |

| dbo.AspNetRoleClaims | |
|--------------------------------------|--|
| Columns | |
| Id (PK, int, not null) | |
| RoleId (FK, nvarchar(450), not null) | |
| ClaimType (nvarchar(max), null) | |
| ClaimValue (nvarchar(max), null) | |

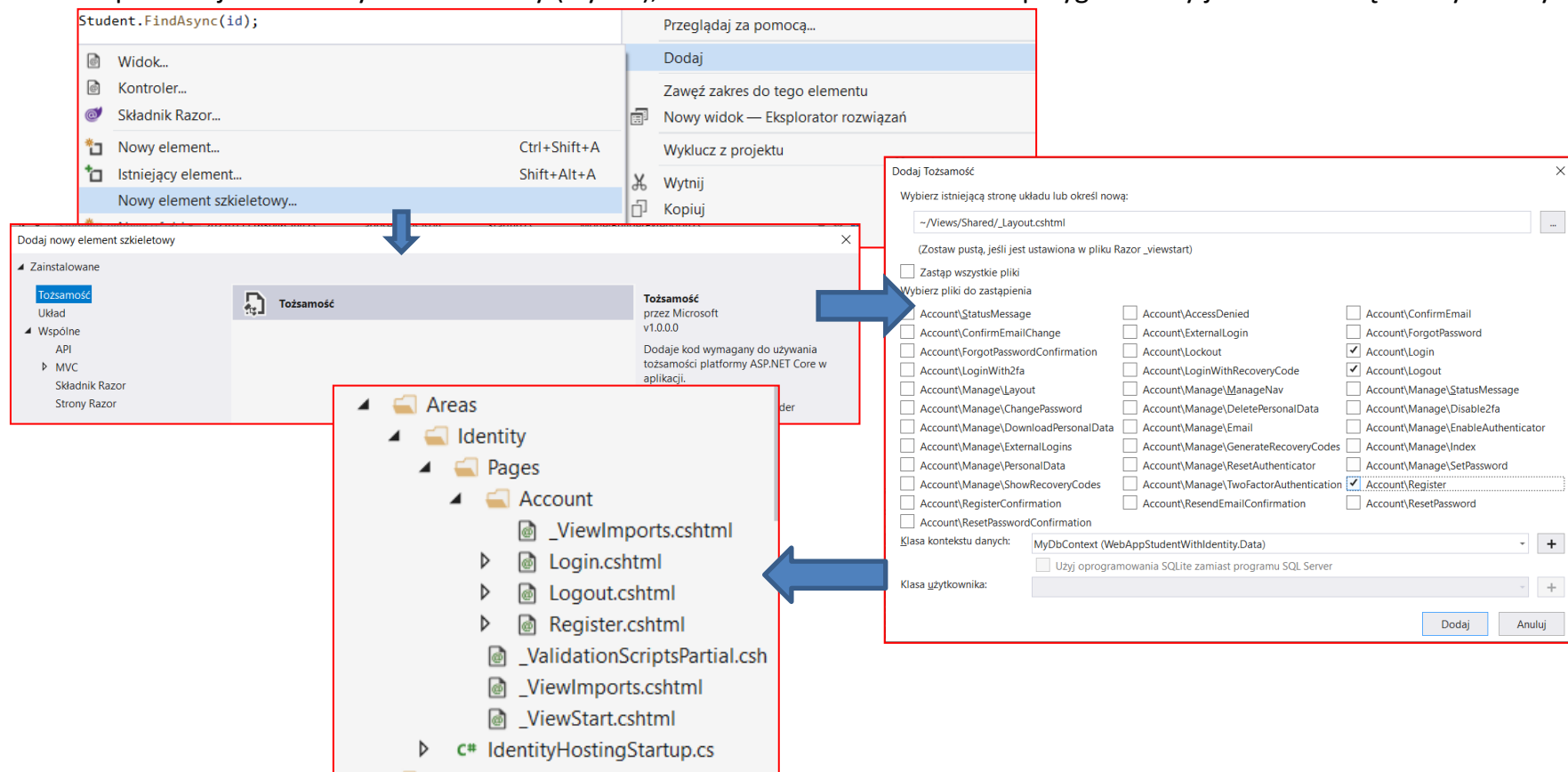
| dbo.AspNetUserClaims | |
|--------------------------------------|--|
| Columns | |
| Id (PK, int, not null) | |
| UserId (FK, nvarchar(450), not null) | |
| ClaimType (nvarchar(max), null) | |
| ClaimValue (nvarchar(max), null) | |

| dbo.AspNetUserLogins | |
|---------------------------------------------|--|
| Columns | |
| LoginProvider (PK, nvarchar(128), not null) | |
| ProviderKey (PK, nvarchar(128), not null) | |
| ProviderDisplayName (nvarchar(max), null) | |
| UserId (FK, nvarchar(450), not null) | |

| dbo.AspNetUserTokens | |
|---------------------------------------------|--|
| Columns | |
| UserId (PK, FK, nvarchar(450), not null) | |
| LoginProvider (PK, nvarchar(128), not null) | |
| Name (PK, nvarchar(128), not null) | |
| Value (nvarchar(max), null) | |

Strony Razora dla Identity

- Strony do operacji związanych z zarządzaniem kontami, rolami itd. są to **strony** Razora. Nie są one tworzone samoistnie, gdyż można je też napisać od zera.
- Aby skorzystać z gotowych **wzorców stron** należy:
 - PPM na projekcie -> Dodaj -> Dodaj nowy element szkieletowy
 - Następnie: Tożsamość -> Tożsamość -> Dodaj
 - Jeśli chcemy dodać do istniejącego layout-u to wpisujemy np. ~/Views/Shared/_Layout.cshtml
 - Jako kontekst można stworzyć nowy lub podać używany np. UniversityDbContext
- Pojawi się nowy folder Area/Identity z wybranymi stronami
- Nie powoduje to zmiany układu strony (layout), chociaż w folderze Shared przygotowany jest widok częściowy do użycia.



Zmiany w plikach

- Dodać w serwisach, że identyfikacja będzie na podstawie wytworzonej bazy EntityFramework i będziemy **używać ról** (menadżera ról).
- W domyślnym layout-ie jest już włączenie widoku częściowego do logowania.

```
// Add services to the container.
var connectionString = builder.Configuration.GetConnectionString("UnivDb") ?? throw new InvalidOperationException();
builder.Services.AddDbContext<UniversityDbContext>(options =>
    options.UseSqlServer(connectionString));
builder.Services.AddDatabaseDeveloperPageExceptionFilter();

builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = false)
    .AddRoles<IdentityRole>() //add the role service.
    .AddEntityFrameworkStores<UniversityDbContext>();
builder.Services.AddControllersWithViews();

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

```
<div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
    <ul class="navbar-nav flex-grow-1">
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
        </li>
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Students" asp-action="Index">Students</a>
        </li>
    </ul>
    <partial name="_LoginPartial" />
</div>
```

Działanie

- Użytkownicy mogą się zarejestrować i logować
 - Ale domyślnie jest potrzeba potwierdzenia zakładania konta przez podany email (wyłączona opcją na poprzednim slajdzie)
 - Są wymagania na poprawne, trudne hasło (też można zmienić)
 - Użytkownik nie ma roli
- Po zalogowaniu działa panel użytkownika (PPM na nazwę użytkownika):
 - Istnieją domyślne strony Razora (mimo, że ich nie dodaliśmy do obszaru)

The diagram illustrates the user registration and login process in the WebApp9_IdentityStudent application. It consists of three main components:

- Register Page:** A form titled "Register" with the subtitle "Create a new account." It includes fields for Email (placeholder: name@example.com), Password (placeholder: password), and Confirm Password (placeholder: password). A blue "Register" button is at the bottom.
- Log in Page:** A form titled "Log in" with the subtitle "Use a local account to log in." It includes fields for Email (placeholder: normaluser@localhost) and Password (placeholder: password). A checkbox for "Remember me?" is present. A blue "Log in" button is at the bottom. A link "Forgot your password?" is also visible.
- Manage your account Page:** A page titled "Manage your account" with the subtitle "Change your account settings." It features a "Profile" section with a "Save" button. The profile information includes Username (normaluser@localhost), Password, Two-factor authentication, and Personal data (Phone number). A "Logout" button is also present.

Arrows indicate the flow of the process: from the Register and Log in pages to the Manage your account page.

Autoryzacja oparta na rolach

Autoryzacja oparta na rolach

- Ustalić nazwy ról jakie mogą występować w aplikacji.
- Ustalić związane z nimi uprawnienia.
- Oznaczyć kontrolery lub poszczególne akcje adnotacjami pozwalającymi/zabraniającymi korzystania z danej akcji przez określone role/użytkowników.
- Użyć adnotacji dla akcji dopuszczalnych dla nie/zalogowanych użytkowników.
- Role można sprawdzić w ramach kody C#/Razor i na ich podstawie zaprezentować inny widok (widok częściowy) lub jego fragment.
- Warto na początku „zasiać” bazę danych rolami oraz kontem administracyjnym.
- Domyślny szkielet aplikacji z autoryzacją pozwala każdemu założyć konto (w niektórych zastosowaniach nie jest to pożądana funkcjonalność).
- Należy w wielu miejscach kodu dodać zespół `Microsoft.AspNetCore.Authorization`;
- Zmienić kod w `Program.cs`, aby używać ról.
- Analogicznie narzędzia (adnotacje/klasa i metody) dostępne są również dla **autoryzacji opartej o politykach** (ang. policy).

Autoryzacja – zasianie danych

- Zasianie danymi
 - Można uruchomić w kontekście b.d. w ramach metody `OnModelCreating()`
 - Wtedy jednak trzeba samemu szyfrować hasła, wpisać „ręcznie” role i przypisać je do użytkowników itd..
 - Lepiej w `Program.cs` z wykorzystaniem odpowiednich manadżerów.
- W przypadku pracy na użytkownikach i rolach warto użyć odpowiednich manadżerów
 - Stworzą identyfikatory
 - Sprawdzą zgodność z regułami hasła
 - I in.
- Ponieważ w nowszych wersjach .Net cały kod startowy jest w metodzie `Main()`, to nie można do niej wstrzyknąć czegokolwiek. Zamiast tego należy wprost pobrać z kontenera serwisów odpowiednie managery.

```
app.UseAuthorization();
```

```
using (var scope = app.Services.CreateScope())  
{  
    var userManager = scope.ServiceProvider.GetRequiredService<userManager<IdentityUser>>();  
    var roleManager = scope.ServiceProvider.GetRequiredService<RoleManager<IdentityRole>>();  
    MyIdentityDataInitializer.SeedData(userManager, roleManager);  
}
```

```
app.MapStaticAssets();
```

```
app.MapControllerRoute(
```

Inicjalizacja danych do autoryzacji 1/2

Data/MyIdentityDataInitializer

```
public class MyIdentityDataInitializer
{
    public static void SeedData(UserManager<IdentityUser> userManager,
                                RoleManager<IdentityRole> roleManager)
    {
        SeedRoles(roleManager);
        SeedUsers(userManager);
    }

    // name - poprawny adres email
    // password - min 8 znaków, mała i duża litera, cyfra i znak specjalny
    public static void SeedRoles(RoleManager<IdentityRole> roleManager)
    {
        if (!roleManager.RoleExistsAsync("Admin").Result)
        {
            IdentityRole role = new IdentityRole
            {
                Name = "Admin",
            };
            IdentityResult roleResult = roleManager.CreateAsync(role).Result;
        }
        if (!roleManager.RoleExistsAsync("Dean").Result)
        {
            IdentityRole role = new IdentityRole
            {
                Name = "Dean",
            };
            IdentityResult roleResult = roleManager.CreateAsync(role).Result;
        }
    }
}
```

Inicjalizacja danych do autoryzacji 2/2

Data/MyIdentityDataInitializer

```
public static void SeedOneUser(UserManager<IdentityUser> userManager,
    string name, string password, string role = null)
{
    if (userManager.FindByNameAsync(name).Result == null)
    {
        IdentityUser user = new IdentityUser
        {
            UserName = name, // musi być taki sam jak email, inaczej nie zadziała
            Email = name
        };
        IdentityResult result = userManager.CreateAsync(user, password).Result;
        if (result.Succeeded && role != null)
        {
            userManager.AddToRoleAsync(user, role).Wait();
        }
    }
}

public static void SeedUsers(UserManager<IdentityUser> userManager)
{
    SeedOneUser(userManager, "normaluser@localhost", "nUpass1!");
    SeedOneUser(userManager, "adminuser@localhost", "aUpass1!", "Admin");
    SeedOneUser(userManager, „deanuser@localhost", „dUpass1!", „Dean");
}
}
```

Adnotacje dla autoryzacji

- Adnotacje mogą się odnosić do **kontrolera** lub do **akcji**.
- Adnotacja przy akcji **dokładą** regułę do adnotacji przy kontrolerze.
 - Wyjątek: `[AllowAnonymous]`
- Najczęstsze adnotacje:
 - `[AllowAnonymous]`
 - Dostęp możliwy dla wszystkich. Domyślny, jeśli nie ma żadnych innych adnotacji dla autoryzacji. Przy akcji odwołuje ograniczenia nałożone przez adnotacje przy kontrolerze/akcji.
 - `[Authorize]`
 - Dostęp dla zalogowanych użytkowników
 - `[Authorize(Roles = "Admin, PowerUser")]`
 - jeśli chcemy, aby użytkownik musiał mieć dowolną z ról, oddzielamy je przecinkiem
 - `[Authorize(Roles = "Admin")]`
`[Authorize(Roles = "PowerUser")]`
 - Jeśli musi posiadać kilka ról, aby móc wykonać akcję.
- Podczas wykorzystywania polityk autoryzacji stosuje się
 - `[Authorize(Policy = "RequireRoleForTurnOnOff")]`

Adnotacje dla autoryzacja – przykład 1

- Adnotacje dla tylko dla akcji w kontrolerze HomeController.

```
{
  Odwołania: 0
  public class HomeController : Controller
  {
    Odwołania: 0
    public IActionResult Index()
    {
      return View();
    }

    [Authorize(Roles = "Admin")]
    Odwołania: 0
    public IActionResult ForAdmin()
    {
      ViewData["Info"] = "For Admin";
      return View("Info");
    }

    [AllowAnonymous]
    Odwołania: 0
    public IActionResult ForAll()
    {
      ViewData["Info"] = "For All";
      return View("Info");
    }

    [Authorize]
    Odwołania: 0
    public IActionResult ForLogIn()
    {
      ViewData["Info"] = "For Log In";
      return View("Info");
    }

    [Authorize(Roles = "Admin, Dean")]
    Odwołania: 0
    public IActionResult ForAdminOrDean()
    {
      ViewData["Info"] = "For Admin or Dean";
      return View("Info");
    }
  }
}
```

Adnotacje dla autoryzacja – przykład 2

```
[Authorize(Roles = "Admin")]
Odwołania: 0
public class AdminController : Controller
{
    [AllowAnonymous]
    Odwołania: 0
    public IActionResult Index() // for all
    {
        ViewData["Info"] = "AdminController -> For All";
        return View("Info");
    }

    [Authorize(Roles = "Dean")]
    Odwołania: 0
    public IActionResult ForDean() // for (Admin and Dean) role
    {
        ViewData["Info"] = "AdminController -> For (Admin and Dean)";
        return View("Info");
    }

    Odwołania: 0
    public IActionResult ForAdmin() // for Admin role
    {
        ViewData["Info"] = "AdminController -> For Admin";
        return View("Info");
    }
}
```

- Przykładowe adnotacje dla kontrolera AdminController i jego akcji.

Autoryzacja – strony Razora

- Korzystanie z ról w kodzie Razor-a:
 - strona częściowa `_LoginPartial.cshtml`
 - zmiana menu w `_Layout.cshtml`

```
@using Microsoft.AspNetCore.Identity
@inject SignInManager<IdentityUser> SignInManager
@inject UserManager<IdentityUser> UserManager

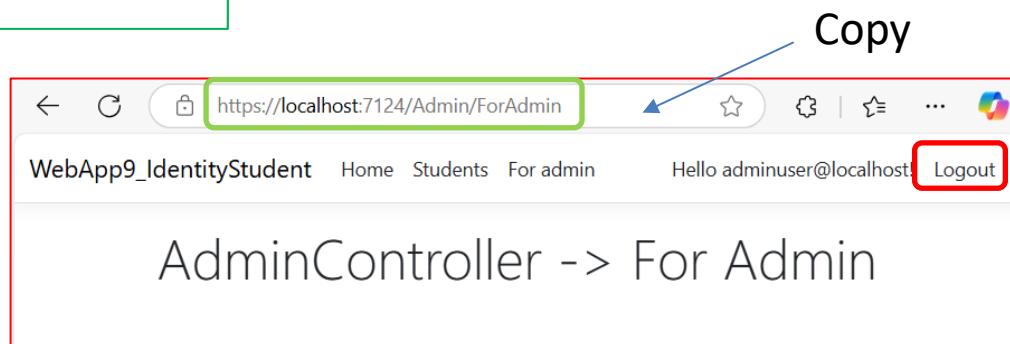
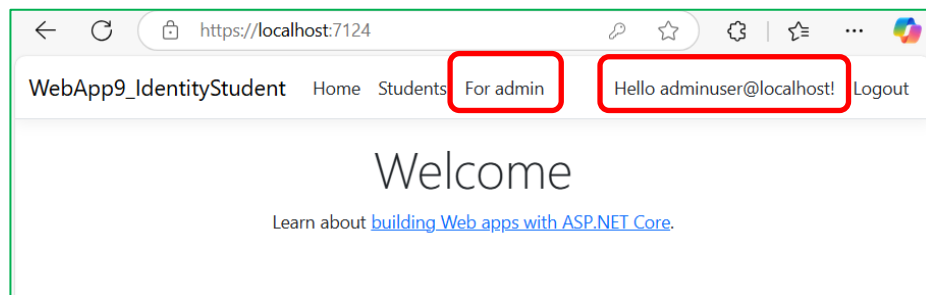
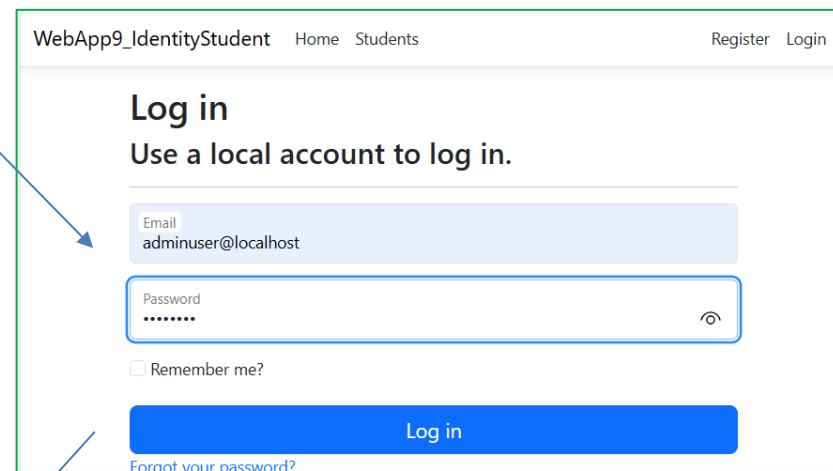
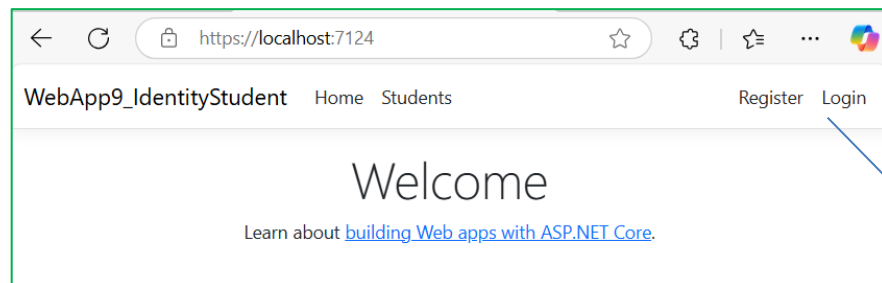
<ul class="navbar-nav">
    @if (SignInManager.IsSignedIn(User))
    {
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="Identity" asp-page="/Account/Manage/Index" title="Manage">Hello @User.Identity.Name!</a>
        </li>
        <li class="nav-item">
            <form class="form-inline" asp-area="Identity" asp-page="/Account/Logout" asp-route-returnUrl="@Url.Action("Index", "Home", new { area = "" })">
                <button type="submit" class="nav-link btn btn-link text-dark">Logout</button>
            </form>
        </li>
    }
    else
    { ... }
```

```
<li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="Students" asp-action="Index">Students</a>
</li>
@if (User.IsInRole("Admin"))
{
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Admin" asp-action="ForAdmin">For admin</a>
    </li>
}
```


Autoryzacja – scenariusze użycia

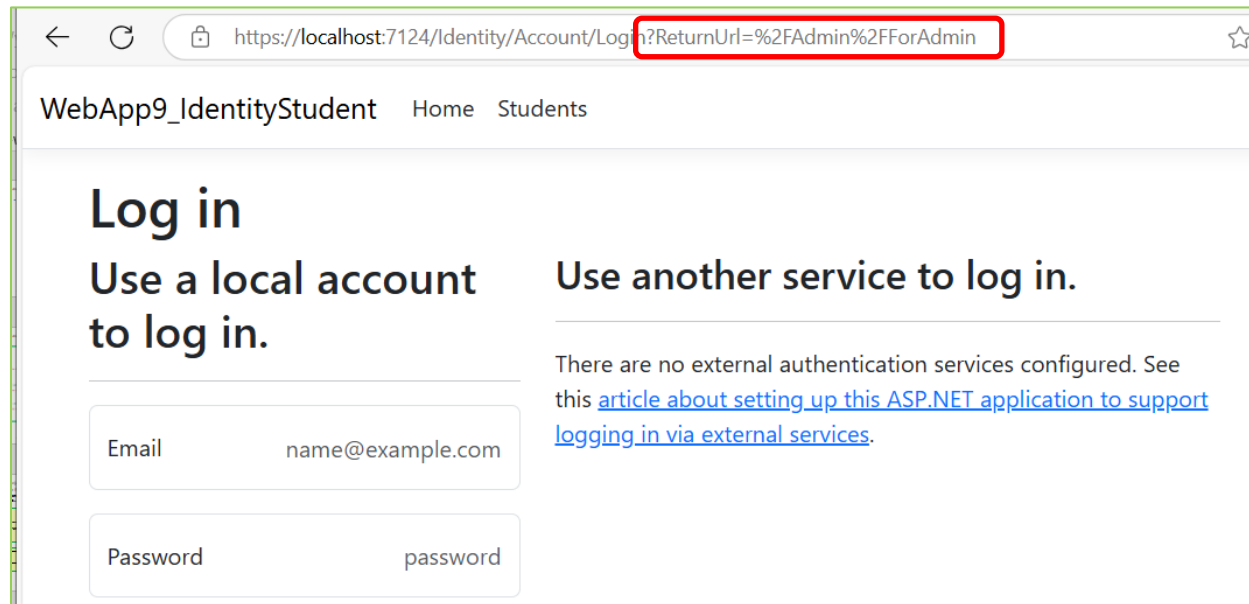
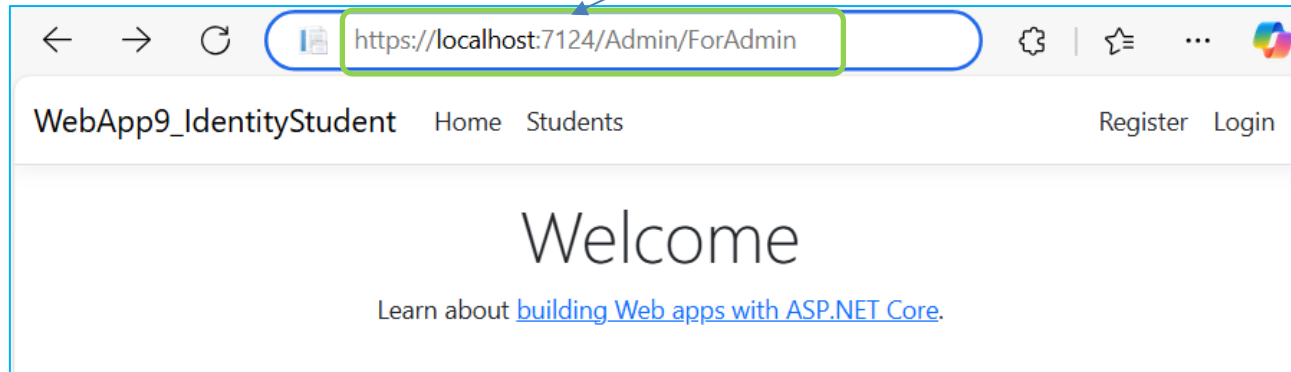
- Zalogowanie na zwykłego użytkownika
- Zalogowanie z rolą dziekana („Dean”)
- Zalogowanie z rolą administrator
- Obserwacja zmiany menu
- Dostęp do stron bez zalogowania
 - Próba dostępu poprzez wpisanie nazwy akcji w pasku adresu – przekierowanie do logowania

Logowanie z rolą „Admin”



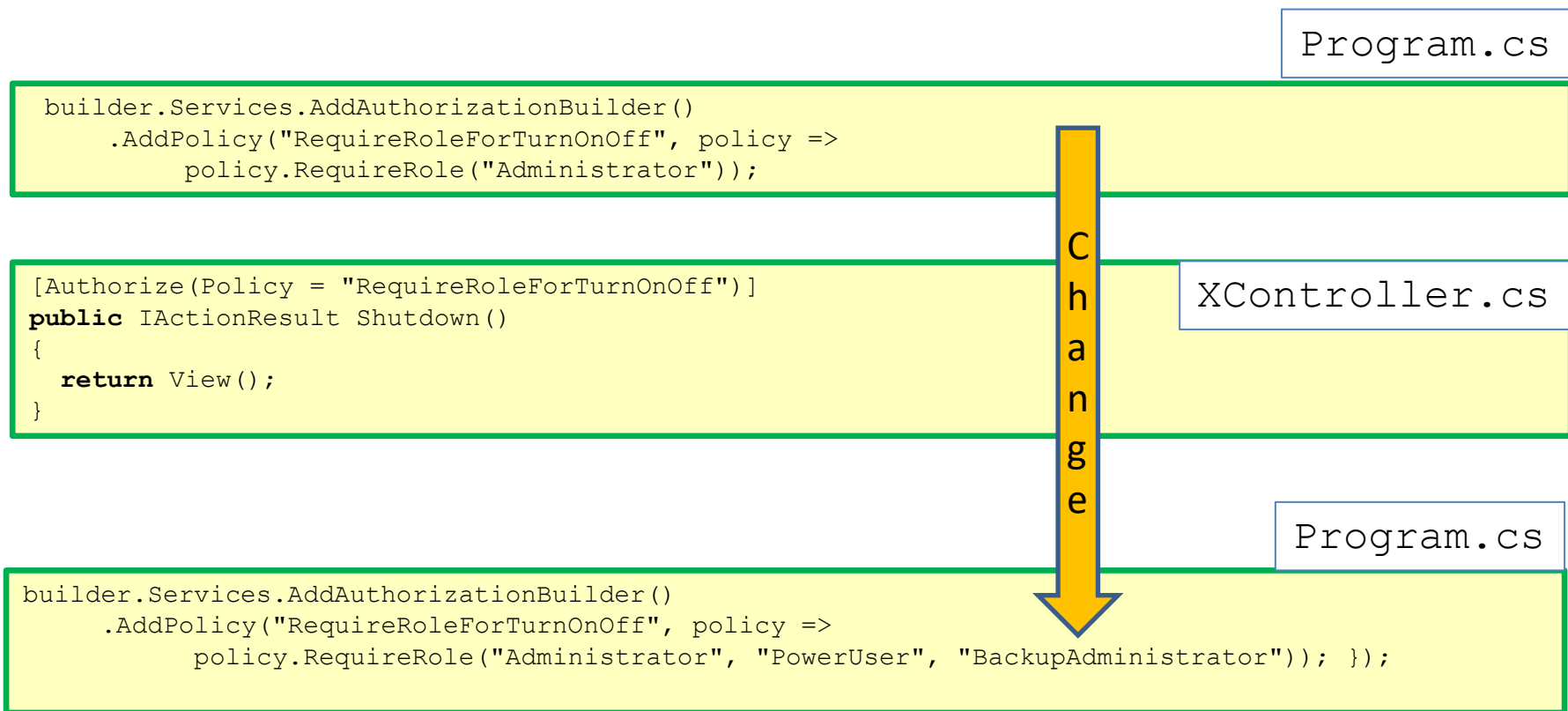
Przekierowanie po zalogowaniu

Paste+<Enter>



Przykład tworzenia polityk autoryzacji

- Tworzenie polityk autoryzacji jest bardziej elastyczne – można tworzyć reguły z użycie ról, roszczeń (Claim), tokenów itd.
- Pozwalają też dodać kolejny poziom elastyczności podczas programowania lub projektowania.
- Przykład (bez rzeczywistej implementacji) poniżej:
 - Zmiana definicji polityki bez potrzeby zmian adnotacji



Dodawanie Identity do istniejącego projektu z EF

Dodawanie identyfikacji **do istniejącego** projektu:

- Instalacja pakietu Identity
- Zmiana dziedziczenia kontekstu
 - `public class GameDbContext : IdentityDbContext`
- Dodanie migracji w konsoli pakietów:
 - `add-migration AddIdentity`
- Uaktualnienie bazy danych
- Dodawanie stron Razora:
 - PPM na projekcie -> Dodaj -> Dodaj nowy element szkieletowy
 - Tożsamość -> Tożsamość -> Dodaj
 - Jeśli chcemy dodać do istniejącego layout-u to wpisujemy np. `~/Views/Shared/_Layout.cshtml`
- Stworzenie (skopiowanie z innego projektu?) `~/Views/Shared/_LoginPartial.cshtml`
- Zmodyfikowanie `~/Views/Shared/_Layout.cshtml`, aby korzystał z powyższej strony częściowej.
- Dodanie (gdzie potrzebujemy) korzystania z zespołów `Microsoft.AspNetCore.Identity`, `Microsoft.AspNetCore.Identity.EntityFrameworkCore` itp.
- Dodanie w serwisach użycie identyfikacji i autoryzacji:
 - `services.AddDefaultIdentity<IdentityUser>()`
 - `.AddRoles<IdentityRole>()`
 - `.AddEntityFrameworkStores<GameDbContext>();`
- Dodanie w konfiguracji potoku oprogramowania pośredniczącego:
 - `app.UseAuthorization();`
- Dodanie mapowania dla stron Razora

Zawartość pakietu Identity

- Wybrane dodatkowe funkcjonalności:
 - Operacje CRUD na kontach użytkowników
 - Potwierdzanie poprawności konta
 - Odzyskiwanie hasła
 - Dwuetapowa autentykacja SMS-em
 - itd.
- Bardzo elastyczne rozwiązanie. Można:
 - Stworzyć własną klasę użytkownika dziedzicząc po `IdentityUser` (można też bez dziedziczenia)
 - Stworzyć własną klasę ról dziedzicząc po `IdentityRole` (można też bez dziedziczenia)
 - Podać typ klucza użytkownika
 - Stworzyć własne menadżery użytkowników i ról



```
2 references
public class IdentityAppContext : IdentityDbContext<AppUser, AppRole, int>
{
    0 references | 0 exceptions
    public IdentityAppContext(DbContextOptions<IdentityAppContext> options) : base(options)
    {
    }
}
```

Claim and Token

Krótko-idea:

- Inne sposoby podejścia do autoryzacji
- Rola to tylko nazwa (string)
 - Użytkownik posiada rolę lub nie, rodzaj wartości logicznej.
- Claims to jakby słownik z cechami dla użytkownika
 - Cechy mogą mieć wartość (np. stan konta w banku, poziom zaufania, wiek itp.)
- Token – jakby ciasteczka dla autoryzacji
 - Np. JWT - JSON Web Token

Dodatek

- Jak skonfigurować opcje aby nie wprowadzać za dużo wymagań na hasło

```
builder.Services.AddDefaultIdentity<IdentityUser>(options => {  
    options.SignIn.RequireConfirmedAccount = false;  
    options.Password.RequireLowercase = false;  
    options.Password.RequireUppercase = false;  
})  
    .AddRoles<IdentityRole>() //add the role service.  
    .AddEntityFrameworkStores<UniversityDbContext>();
```


Kontrolery API

Podejście ReST i RESTful

- REST, ReST - **R**epresentational **S**tate **T**ransfer
 - Zaprojektowane wokół zasobów (a nie operacji)
 - URI (**U**niform **R**esource **I**dentifier) definiujące konkretny zasób (zob. następny slajd)
 - Klient komunikuje się z serwerem wymieniając *reprezentację* danych (JSON).
 - Zunifikowany interfejs używający czasowników z metod HTTP: GET, POST, PUT, PATCH oraz DELETE.
 - Bezstanowość jak w HTTP
 - Jest sterowane przez hiperlinki w reprezentacji (poniżej)
- Uznaje się 4 stopnie we wprowadzaniu REST API:
 - 1) zdefiniuj jedno URI i użyj do wszystkiego żądań POST do tego URI.
 - 2) Stwórz oddzielne URI dla każdego zasobu.
 - 3) Użyj metod HTTP do zdefiniowania operacji na zasobach.
 - 4) użyj hipermediów HATEOAS (Hypertext as the Engine of Application State)
- RESTful oznacza użycie 4 stopnia.

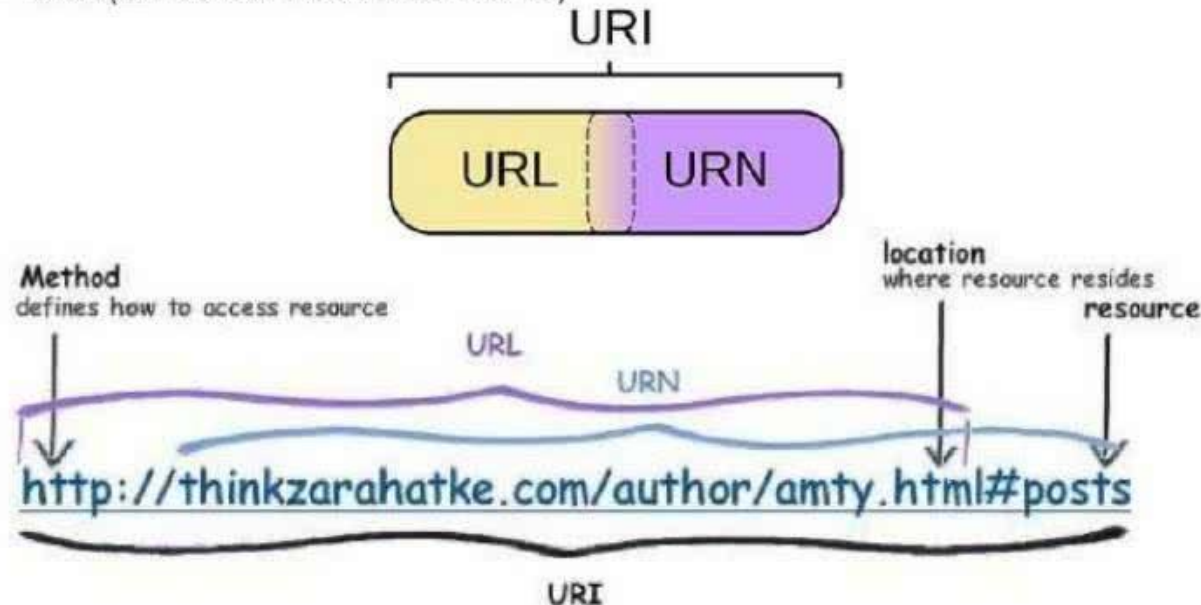
```
{
  "account": {
    "account_number": 12345,
    "balance": {
      "currency": "usd",
      "value": 100.00
    },
    "links": {
      "deposits": "/accounts/12345/deposits",
      "withdrawals": "/accounts/12345/withdrawals",
      "transfers": "/accounts/12345/transfers",
      "close-requests": "/accounts/12345/close-requests"
    }
  }
}
```

URI, URL, URN

- Różnice między URI, URL, URN
- Źródło: <https://ep.com.pl/novosci/koktajl-newsow/13819-adres-strony-internetowej-to-nie-url-glupcze>

URI(2)

- URL(Universal Resource Locator)
- URN(Universal Resource Name)



© Copyright 2013 iSecure Co., Ltd. The information contained herein is subject to change without notice.

Metody HTTP i adresy w typie RESTful

| Metoda HTTP | Adres URL | Opis |
|-------------|----------------|-------------------------------------------|
| GET | /api/student | Pobranie wszystkich obiektów Student |
| GET | /api/student/1 | Pobranie obiektu Student z id równym 1 |
| POST | /api/student | Utworzenie nowego obiektu Student |
| PUT | /api/student | Zastąpienie istniejącego obiektu |
| PATCH | /api/student/1 | Modyfikacja obiektu Student o id równym 1 |
| DELETE | /api/student/1 | Usunięcie obiektu Student o id równym 1 |

Ogólna zasada adresacji

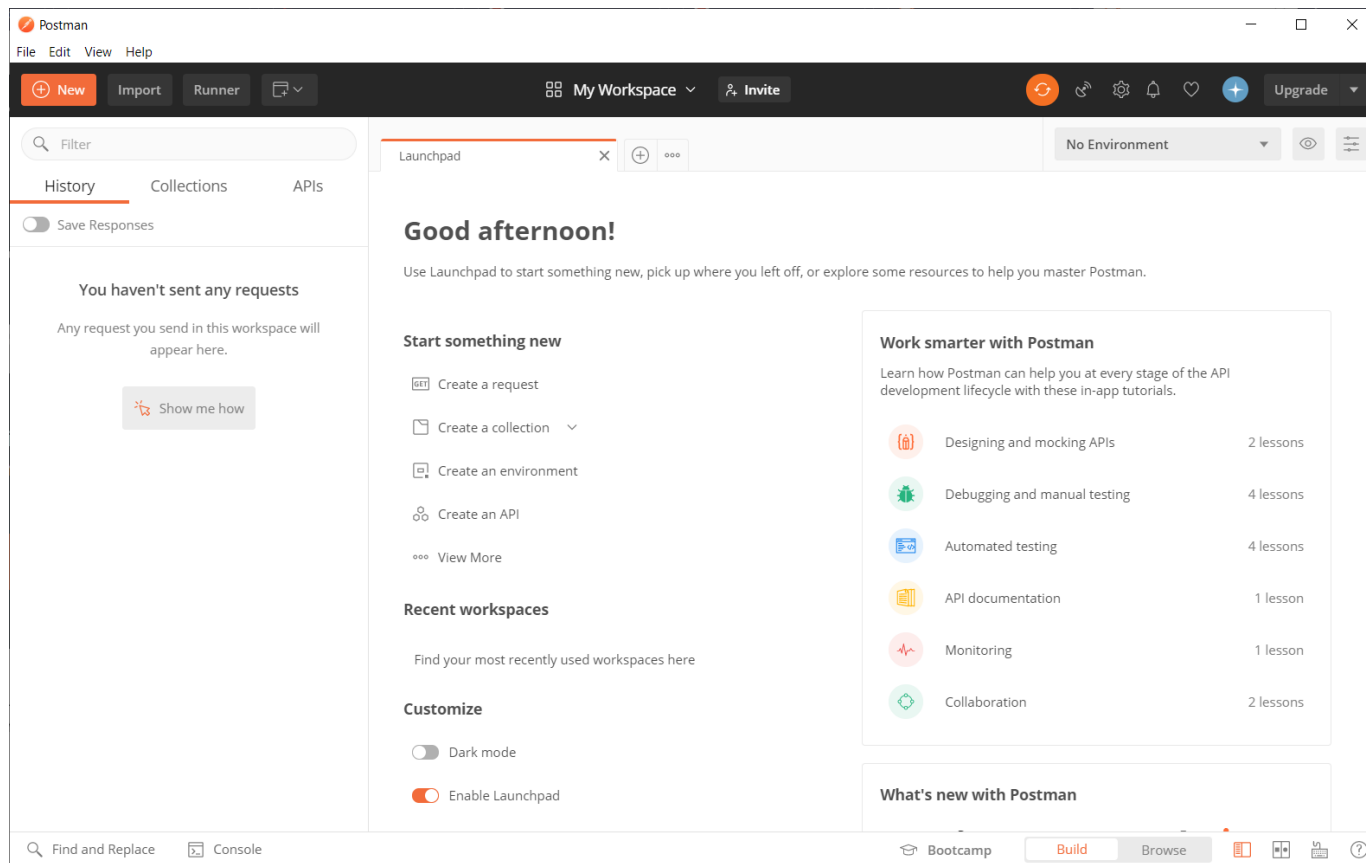
- Na początku: `/api/` lub `/api/v1/`, `api/v2/` itp.
- Potem nazwa zasobu lub zbioru funkcjonalności
 - Podzielona znakami `/`: `studia/grupy/zapisy`,
- Ewentualne oczywiste parametry: `{id}`
- Ewentualne opcjonalne lub dodatkowe parametry jako query string:
`?pos_from={from}&pos_to={to}`

Ładunek w żądaniach API

- Ładunek (ang. payload) to zawartość ciała (ang. body) w żądaniu i odpowiedzi
- Może być w różnych formatach
 - W wykładzie użyty zostanie JSON
 - Kiedyś był to XML, ale jest zbyt rozwlekły format
- Obecność ładunku wynika dość logicznie z operacji:
 - GET – brak w żądaniu, **w odpowiedzi** jeden lub wszystkie zasoby
 - POST – **w żądaniu**: minimum wymagane pola zasobu, **w odpowiedzi**: cały utworzony zasób z magazynu danych (ew. brak ładunku)
 - PUT: **w żądaniu**: zasób do utworzenia/zastąpienia , **odpowieź** jak w POST
 - PATCH: **w żądaniu**: pola wymagające zmiany, odpowiedź: potwierdzenie lub nie zmian
 - DELETE: brak ładunku w żądaniu i odpowiedzi

Postman – narzędzie do testowania

- <https://www.postman.com/downloads/>
 - Windows
 - MacOS
 - Linux
- Należy stworzyć konto lub użyć konta Google



Postman – użycie 1/2

- Wysyłanie żądania HTTP w wybranym formacie
- Odbiór wyniku żądania
- Wybrać „Create a request”

Automatyczna zamiana w obydwie strony

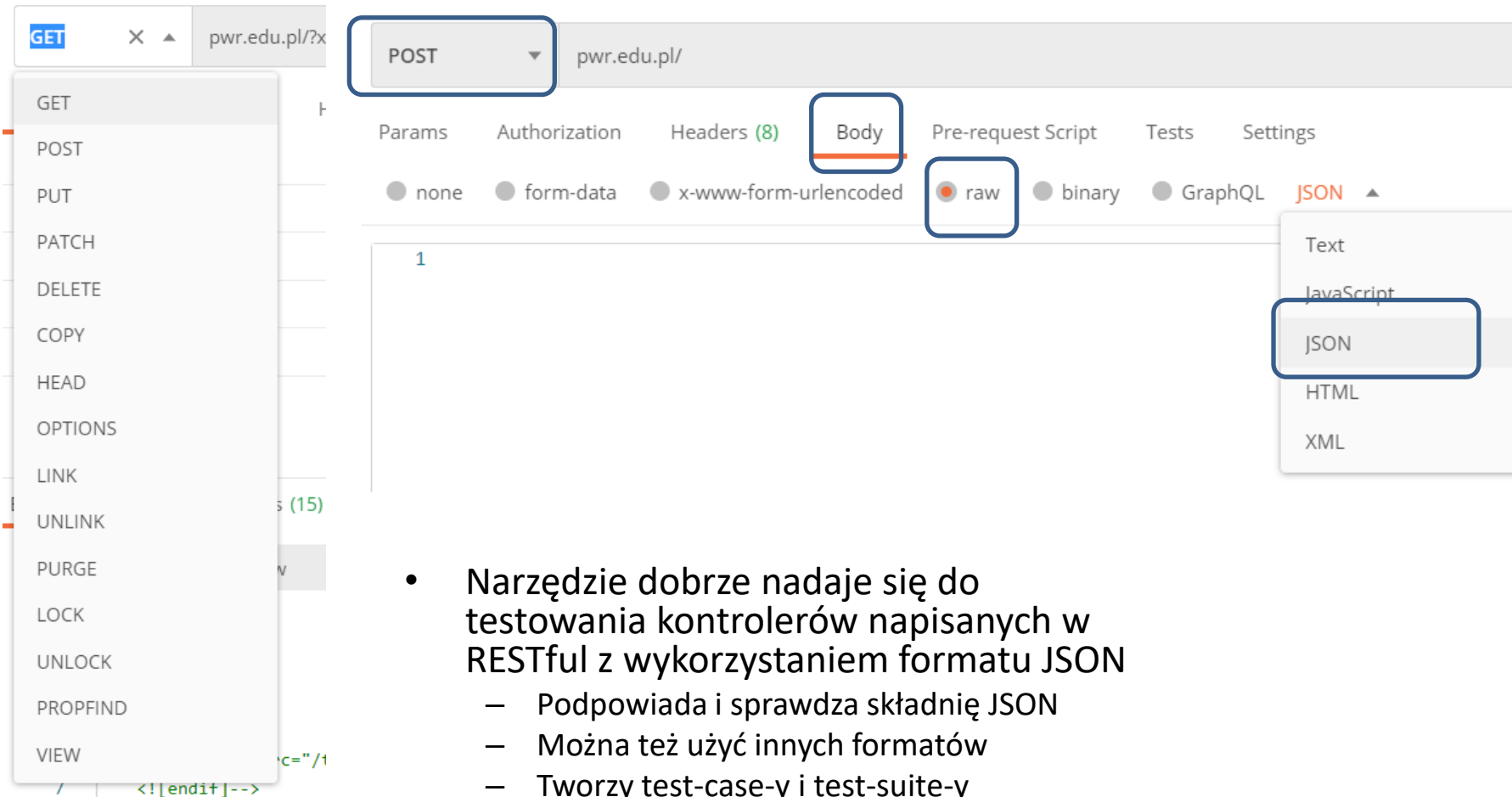
The screenshot shows the Postman interface with a GET request to `pwr.edu.pl/?x=4&Name=Dariusz`. The `Params` tab is selected, displaying the following query parameters:

| KEY | VALUE | DESCRIPTION |
|------|---------|-------------|
| x | 4 | |
| Name | Dariusz | |

The response is shown in the bottom panel, indicating a `Status: 200 OK` with a time of `3.40 s` and a size of `192.18 KB`. The response body is displayed in the `HTML` view, showing the following code:

```
1 <!DOCTYPE html>
2 <html lang="pl">
3
4 <head>
5   <!--[if IE]>
6     <script src="/themes/_system/js/jquery-1.12.3.min.js"></script>
7   <![endif]>
8   <title>Politechnika Wrocławska</title>
9   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```


Postman – użycie 2/2



- Narzędzie dobrze nadaje się do testowania kontrolerów napisanych w RESTful z wykorzystaniem formatu JSON
 - Podpowiada i sprawdza składnię JSON
 - Można też użyć innych formatów
 - Tworzy test-case-y i test-suite-y

Kontroler RESTful – przygotowanie modelu danych

- Klasy dla danych:
 - Klasa `Student`
 - Interfejs `IRepository`
 - Dodanie interfejsu do serwisu
- W projekcie MVC, z wykorzystaniem tylko strony `Index`

```
public class Student
{
    public int Id { get; set; }
    public int Index { get; set; }
    public string? Name { get; set; }
}
```

`Student.cs`

```
public interface IRepository
{
    IEnumerable<Student> Students { get; }
    Student? this[int id] { get; }
    Student AddStudent(Student student);
    Student UpdateStudent(Student student);
    void DeleteStudent(int id);
    Student? GetNextStudent(int id);
    Student? GetPreviousStudent(int id);
}
```

`IRepository.cs`

Później (dla AJAX-a)

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

`Program.cs`

Klasa MemoryRepository

- Implementacja interfejsu, MemoryRepository

```
public class MemoryRepository : IRepository    {
    private readonly Dictionary<int, Student> items; // po co musi być readonly?!
    public MemoryRepository()
    {
        items = new Dictionary<int, Student>();
        new List<Student>
        {
            new Student{ Index=11111, Name="Smith"},
            new Student{ Index=22222, Name="Kowal"},
            new Student{ Index=33333, Name="Schneider"}
        }.ForEach(s => AddStudent(s));
    }
    public Student this[int id] => items.ContainsKey(id)?items[id]:null;

    public IEnumerable<Student> Students => items.Values;

    public Student AddStudent(Student student) {
        if (student.Id == 0)
        {
            int key = items.Count;
            while (items.ContainsKey(key)) { key++; };
            student.Id = key;
        }
        items[student.Id] = student;
        return student;
    }

    public void DeleteStudent(int id)=>items.Remove(id);

    public Student UpdateStudent(Student student) => AddStudent(student);
    ...// next lines for AJAX
}
```

MemoryRepository.cs

Sprawdzenie poprzez MVC – Kontroler Home

```
public class HomeController : Controller
{
    private IRepository Repository { get; set; }

    public HomeController(IRepository repo) => Repository = repo;

    public ActionResult Index() => View(Repository.Students);

    [HttpPost]
    public IActionResult AddStudent(Student student)
    {
        Repository.AddStudent(student);
        return RedirectToAction("Index");
    }
}
```

HomeController.cs

- Utworzenie widoku o poniższym wyglądzie:

WebApiController Home Privacy

Index:

Name:

| ID | Index | Name |
|----|-------|-----------|
| 0 | 11111 | Smith |
| 1 | 22222 | Kowal |
| 2 | 33333 | Schneider |

Sprawdzenie poprzez MVC – widok Index.cshtml

```
@model IEnumerable<Student>
@{ Layout = "_Layout"; }

<form id="addform" asp-action="AddStudent" method="post">
    <div class="form-group">
        <label for="Index">Index:</label>
        <input class="form-control" name="Index" />
    </div>
    <div class="form-group">
        <label for="Name">Name:</label>
        <input class="form-control" name="Name" />
    </div>
    <div class="text-center panel-body">
        <button type="submit" class="btn btn-sm btn-primary">Dodaj</button>
    </div>
</form>

<table class="table table-sm table-striped table-bordered m-2">
    <thead><tr><th>ID</th><th>Index</th><th>Name</th></tr></thead>
    <tbody>
        @foreach (var r in Model)
        {
            <tr>
                <td>@r.Id</td>
                <td>@r.Index</td>
                <td>@r.Name</td>
            </tr>
        }
    </tbody>
</table>
```

Index.cshtml

- Scenariusz użycia: uruchomienie i dodanie studenta

Kontroler API dla klasy Student 1/2

- Standardowo routing dla kontrolerów Api zaczynający się od segmentu „/api”
- Od Core 2.1 istnieje atrybut ApiControllerAttribute
- Kontroler API dziedziczy po ControllerBase (a nie po Controller)

StudentApiController.cs

```
[EnableCors]
[Route("api/student")]
[ApiController]
public class StudentApiController : ControllerBase
{
    private IRepository repository;

    public StudentController(IRepository repo) => repository = repo;

    [HttpGet]
    public IEnumerable<Student> Get() => repository.Students;

    [HttpGet("{id}")]
    public Student? Get(int id) => repository[id];

    [HttpPost]
    public Student Post([FromBody] Student res) =>
        repository.AddStudent(new Student
        {
            Index = res.Index,
            Name = res.Name
        });
}
```

Kontroler API dla klasy Student 2/2

- Dla operacji PATCH potrzeba doinstalować `Microsoft.AspNetCore.JsonPatch`
 - Po wpisaniu „`[FromBody] JsonPatchDocument<Student> patch`” VS 2019 sam zaproponuje instalację.

StudentApiController.cs

```
[HttpPut]
public Student Put([FromBody] Student res) =>
    repository.UpdateStudent(res);

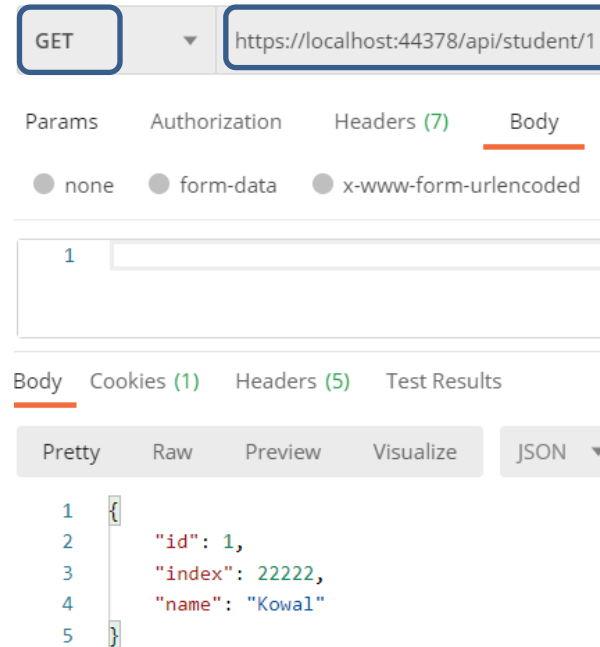
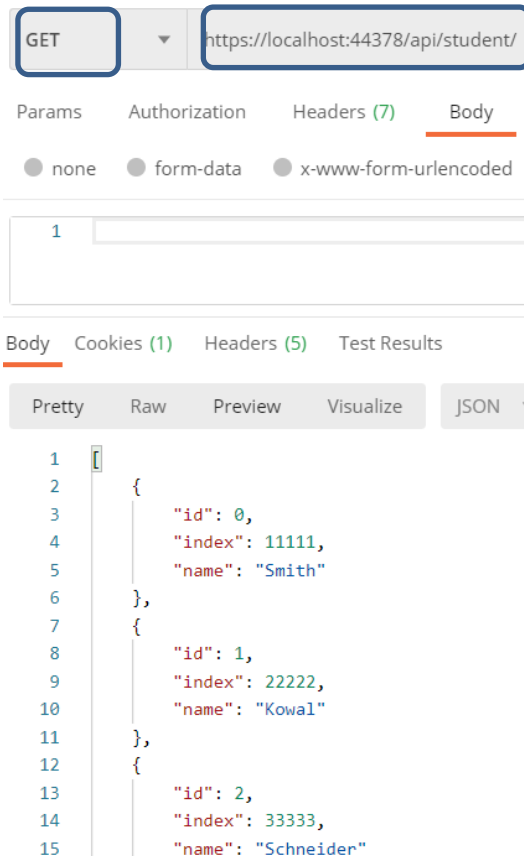
[HttpPatch("{id}")]
public StatusCodeResult Patch(int id,
    [FromBody] JsonPatchDocument<Student> patch)
{
    Student res = Get(id);
    if (res != null)
    {
        patch.ApplyTo(res);
        return Ok();
    }
    return NotFound();
}

[HttpDelete("{id}")]
public void Delete(int id) => repository.DeleteStudent(id);

// next lines for AJAX
...
}
```

Testowanie za pomocą Postmana 1/2

- Scenariusz użycia: wszystkie operacje z kontrolera z użyciem Postman-a



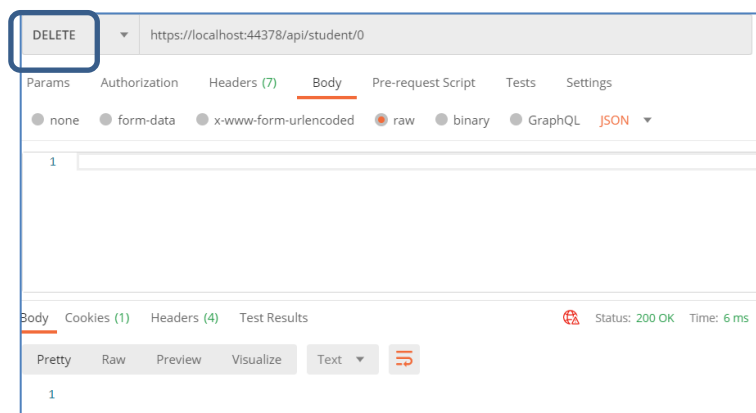
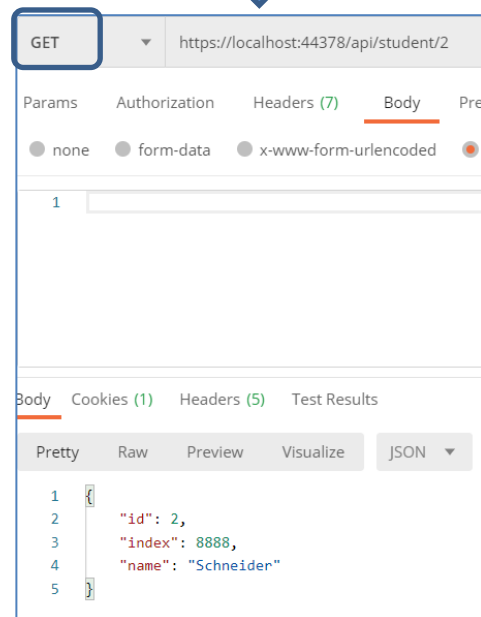
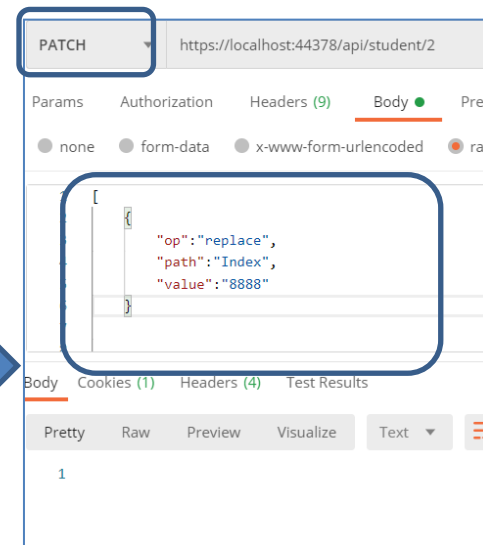
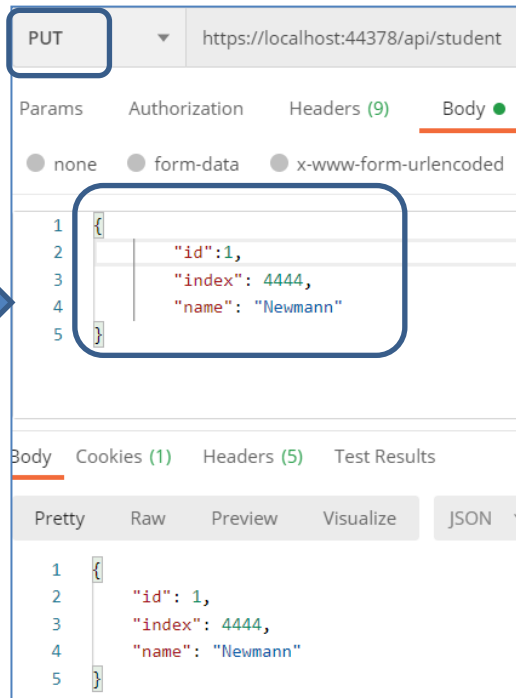
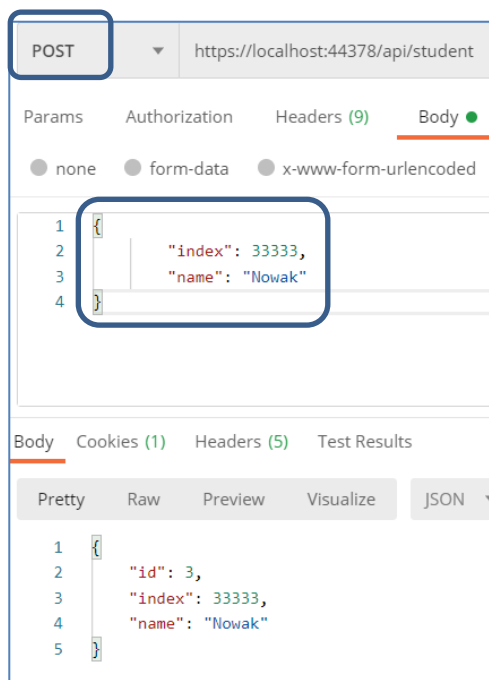
Operacja PATCH i Core 3.x

- W Core operacja PATCH nie działa poprawnie. Aby to poprawić należy doinstalować pakiet `Microsoft.AspNetCore.Mvc.NewtonsoftJson` i `Microsoft.AspNetCore.Mvc.NewtonsoftJson` oraz odpowiednio skonfigurować serwisy (np. jak poniżej).
- Powoduje to zapis i odczyt wszystkich zawartości z JSON przez ten pakiet, również podczas obsługi związanej z pakietem `System.Text.Json`.
 - Jak ograniczyć to tylko do dla operacji Patch w RESTfull:
<https://stackoverflow.com/questions/57914725/how-to-implement-jsonpatch-in-net-core-3-0-preview-9-correctly>
- **Często** operacja Patch **nie jest implementowana** w kontrolerach API.
 - Dla niezbyt rozbudowanych obiektów operacja PUT jest wystarczająca i bardziej przejrzysta

Program.cs

```
builder.Services.AddControllersWithViews()  
    .AddNewtonsoftJson();
```

Testowanie za pomocą Postmana 2/2



Komenda curl

Testowanie z linii komend: `curl`

- Bardzo dużo opcji
- Jeśli jest ładunek w zapytaniu najlepiej przygotować plik tekstowy i dołączyć odpowiednią opcją
- Można ustawić/zobaczyć nagłówek zapytania/odpowiedzi

```
C:\Wiersz polecenia

C:\Users\dariu>curl https://localhost:44378/api/student
[{"id":0,"index":11111,"name":"Smith"}, {"id":1,"index":22222,"name":"Kowal"}, {"id":2,"index":33333,"name":"Schneider"}]

C:\Users\dariu>curl https://localhost:44378/api/student/2
{"id":2,"index":33333,"name":"Schneider"}

C:\Users\dariu>curl -h
Usage: curl [options...] <url>
  --abstract-unix-socket <path> Connect via abstract Unix domain socket
  --anyauth             Pick any authentication method
-a, --append           Append to target file when uploading
  --basic              Use HTTP Basic Authentication
  --cacert <CA certificate> CA certificate to verify peer against
  --capath <dir>       CA directory to verify peer against
-E, --cert <certificate[:password]> Client certificate file and password
  --cert-status        Verify the status of the server certificate
  --cert-type <type>   Certificate file type (DER/PEM/ENG)
  --ciphers <list of ciphers> SSL ciphers to use
  --compressed         Request compressed response
-K, --config <file>    Read config from a file
  --connect-timeout <seconds> Maximum time allowed for connection
  --connect-to <HOST1:PORT1:HOST2:PORT2> Connect to host
-C, --continue-at <offset> Resumed transfer offset
-b, --cookie <data>    Send cookies from string/file
-c, --cookie-jar <filename> Write cookies to <filename> after operation
  --create-dirs         Create necessary local directory hierarchy
  --crlf               Convert LF to CRLF in upload
  --crlfile <file>     Get a CRL list in PEM format from the given file
-d, --data <data>      HTTP POST data
  --data-ascii <data>  HTTP POST ASCII data
```

Narzędzie Swagger

- Narzędzie Swagger
 - Pakiet `Swashbuckle.AspNetCore`
- Po zainstalowaniu wystarczy uruchomić dodawanie serwisów i wstawić do strumienia przetwarzania żądania (kody poniżej)

```
// can be even without options
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo { Title = "WebApp9_ApiStudent Test", Version = "v1" });
});
```

Program.cs

```
app.UseSwagger();
app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "WebApp9_ApiStudent Test v1"));
```

Program.cs

- Narzędzie rozbudowane o wiele opcji pomocniczych
 - Można dostosować sobie interfejs graficzny tego narzędzia
 - API kontrolery mogą udostępniać dodatkowy opis operacji
 - itd.

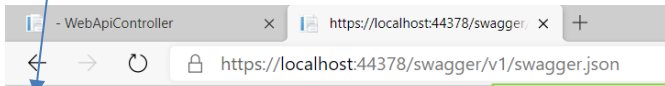
Zastosowanie Swagger-a

Dokument JSON opisujący endpoint-y wg specyfikacji OpenApi

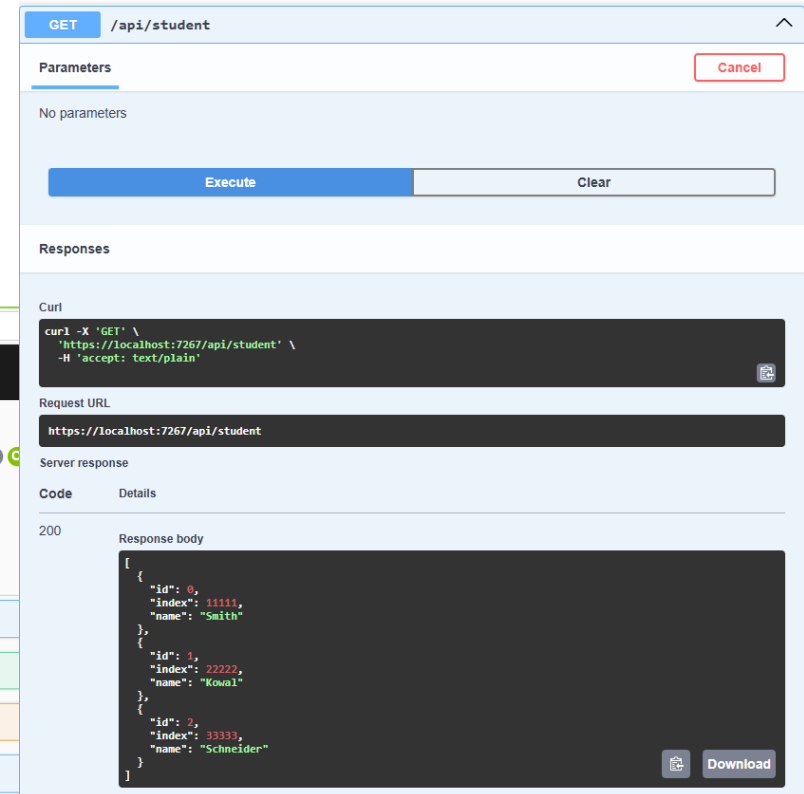
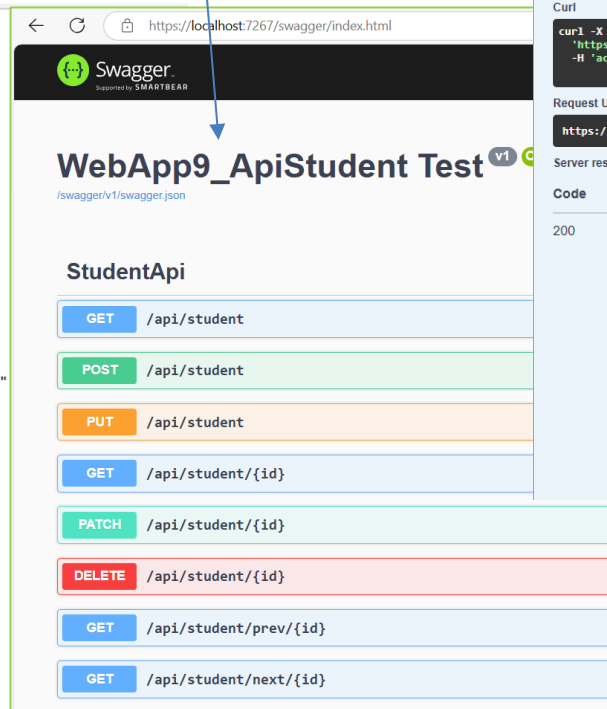
- <http://localhost:<port>/swagger/v1/swagger.json>

Testowanie działania endpointów:

- <http://localhost:<port>/swagger/>
- Np.. Wybrać GET, „try it out”, „Execute” →



```
{
  "openapi": "3.0.1",
  "info": {
    "title": "WebAppApiTest",
    "version": "v1"
  },
  "paths": {
    "/api/Student": {
      "get": {
        "tags": [
          "Student"
        ],
        "responses": {
          "200": {
            "description": "Success",
            "content": {
              "text/plain": {
                "schema": {
                  "type": "array",
                  "items": {
                    "$ref": "#/components/schemas/Student"
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```



Ajax

Problem – przechodzenie między danymi studentów

- Chcemy na stronie `Details` z danymi studenta mieć możliwość przesunięcia się od razu do kolejnego/poprzedniego studenta, aby zobaczyć jego dane w tym samym widoku `Details`.
 - Kolejność mają wyznaczać identyfikatory `Id`. Jeśli nie posortujemy danych, trudno będzie określić, który jest następny.
- Założenie: do **dodawania** danych `Student`-a mamy akcje w kontrolerze `HomeController`, wykonane dla demonstracji kontrolera API (wcześniejsza część tego wykładu). Kolejne kody rozwiązują tylko problem przechodzenia między studentami (odczytu danych).
- W interfejsie `IRepository` i jego implementacji `MemoryRepository` tworzymy metody zwracające następnego/poprzedniego studenta po podanym indeksie.

Znajdowanie poprzedniego/następnego studenta.

- Jeśli poprzedniego/kolejnego elementu nie ma, zwrócona będzie wartość `null`.

```
public interface IRepository
{
    ...
    // next lines for AJAX presentation
    Student? GetNextStudent(int id);
    Student? GetPreviousStudent(int id);
}
```

IRepository.cs

```
public class MemoryRepository : IRepository
{
    ...
    // next lines for AJAX presentation
    public Student? GetNextStudent(int id)
    {
        return items
            .Select(s => s.Value)
            .Where(s => s.Id > id)
            .OrderBy(s => s.Id)
            .FirstOrDefault();
    }
    public Student? GetPreviousStudent(int id)
    {
        return items
            .Select(s => s.Value)
            .Where(s => s.Id < id)
            .OrderByDescending(s => s.Id)
            .FirstOrDefault();
    }
}
```

MemoryRepository.cs

Klasyczne rozwiązanie

- Rozwiązanie pierwsze – klasyczne:
 - Tworzymy StudentController kontroler dla Studenta.
 - Zostawiamy (tworzymy) akcje i widoki Index i Details
 - Wystarczające dla demonstracji działania
 - Do późniejszej modyfikacji
 - W akcji Index studentów sortujemy wg id.
 - W widoku Indeks przy każdym studencie link do jego szczegółów (akcji Details)

```
public ActionResult Index()
{
    return View(repository.Students.OrderBy(s=>s.Id));
}
```

StudentController.cs

```
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Id)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Index)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Name)
        </td>
        <td>
            @Html.ActionLink("Details", "Details", new { id=item.Id })
        </td>
    </tr>
}
```

Index.cshtml

Nowe akcje

- Tworzymy nowe akcji do przechodzenia na poprzedniego/kolejnego studenta
 - Przekierowanie na akcję `Details` z nowym `Id` lub pozostanie na bieżącym jeśli nie ma poprzedniego/kolejnego studenta.

```
...  
  
<dt class="col-sm-2">  
    @Html.DisplayNameFor(model => model.Name)  
</dt>  
<dd id="name" class="col-sm-10">  
    @Html.DisplayFor(model => model.Name)  
</dd>  
</dl>  
</div>  
<div>  
    <a asp-action="Index">Back to List</a>  
</div>  
<div>  
    StudentController:  
    @Html.ActionLink("Prev", "Previous", new { id = Model.Id }) |  
    @Html.ActionLink("Next", "Next", new { id = Model.Id })  
</div>  
<div>  
    JavaScript (AJAX) :  
    <button onclick="jsPrev()"> Previous </button> |  
    <button onclick="jsNext()"> Next </button>  
</div>  
<div>  
    jQuery (AJAX) :  
    <button onclick="jqPrev()"> Previous </button> |  
    <button onclick="jqNext()"> Next </button>  
</div>
```

Details.cshtml

Widok Details

WebApiController Home Privacy Students

Index

[Create New](#)

| Id | Index | Name | |
|----|-------|-----------|-------------------------|
| 0 | 11111 | Smith | Details |
| 1 | 22222 | Kowal | Details |
| 2 | 33333 | Schneider | Details |

© 2020 - WebApiController - [Privacy](#)

WebApiController Home Privacy Students

Details

Student

| | |
|--------------|-----------|
| Id | 2 |
| Index | 33333 |
| Name | Schneider |

[Back to List](#)

StudentController: [Prev](#) | [Next](#)

JavaScript(AJAX): [Previous](#) | [Next](#)

jQuery(AJAX): [Previous](#) | [Next](#)

Nowe akcje kontrolera

- Ostatecznie należy stworzyć nowe akcje w kontrolerze: `Previous()` i `Next()`
 - Gdy repozytorium zwróci `null`, strona z `Details` ma pozostać na tych samych danych studenta.

```
public ActionResult Next(int id)
{
    Student stud = repository.GetNextStudent(id);
    if(stud==null)
        return RedirectToAction(nameof(Details), new { id });
    else
        return RedirectToAction(nameof(Details), new { id = stud.Id });
}
public ActionResult Previous(int id)
{
    Student stud = repository.GetPreviousStudent(id);
    if (stud == null)
        return RedirectToAction(nameof(Details), new { id });
    else
        return RedirectToAction(nameof(Details), new { id = stud.Id });
}
```

StudentController.cshtml

Działanie 1/3

- Akcja Index.

WebApiController Home Privacy Students

Index

[Create New](#)

| Id | Index | Name | |
|----|-------|-----------|-------------------------|
| 0 | 11111 | Smith | Details |
| 1 | 22222 | Kowal | Details |
| 2 | 33333 | Schneider | Details |

Network tab: 1 request. Filter: All. Has blocked cookies: ☐ Blocked Requests: ☐ 3rd-party requests: ☐. Waterfall view shows requests for Student, bootstrap.min.css, site.css, jquery.min.js, bootstrap.bundle..., site.js?v=4q1jwFh..., and data:image/svg+... with status 200 and size 3.6 kB.

Działanie 2/3

- Akcja Details (0).

The screenshot displays a web browser window with the URL `https://localhost:5001/Student/Details/0`. The page title is "Details Student". The student information is as follows:

| Property | Value |
|----------|-------|
| Id | 0 |
| Index | 11111 |
| Name | Smith |

Navigation links include "Back to List", "StudentController: Prev", and "Next". Below these are "JavaScript(AJAX): Previous | Next" and "jQuery(AJAX): Previous | Next". A blue arrow points from the "Next" link in the jQuery(AJAX) section to the bottom right of the image.

The browser's developer tools are open to the "Network" tab, showing a list of requests. The first request, labeled "0", is highlighted with a red box. Its details are as follows:

| Name | Stat.. | Type | Initiator | Size | T. | Waterfall |
|----------------------|--------|---------|-----------|----------|------|-------------------|
| 0 | 200 | doc... | Other | 5.5 kB | 3... | [Waterfall chart] |
| bootstrap.min.css | 200 | styl... | 0 | (memo... | 0... | [Waterfall chart] |
| site.css | 200 | styl... | 0 | (memo... | 0... | [Waterfall chart] |
| jquery.min.js | 200 | script | 0 | (memo... | 0... | [Waterfall chart] |
| bootstrap.bundle.... | 200 | script | 0 | (memo... | 0... | [Waterfall chart] |
| site.js?v=4q1jwFh... | 200 | script | 0 | (memo... | 0... | [Waterfall chart] |

Działanie 3/3

- Pierwsze żądanie to przekierowanie
- Drugie to cała strona HTML z nowym studentem.
 - Jednak w zasadzie zmieniły się tylko dane (pomarańczowa ramka)
 - Pojawia się opóźnienie związane z ładowaniem, renderowaniem całej strony HTML

The screenshot shows a web browser at the URL `https://localhost:5001/Student/Details/1`. The page displays student details for ID 1, Index 22222, and Name Kowal. A yellow box highlights the data fields. The browser's developer tools are open to the Network tab, showing a list of requests. A red box highlights the first request, which is a 70 B document from the browser's cache. The second request is a 5.4 kB document from the server at `https://localhost:5001/Student/Details/1`.

WebApiController Home Privacy Students

Details

Student

| | |
|-------|-------|
| Id | 1 |
| Index | 22222 |
| Name | Kowal |

[Back to List](#)

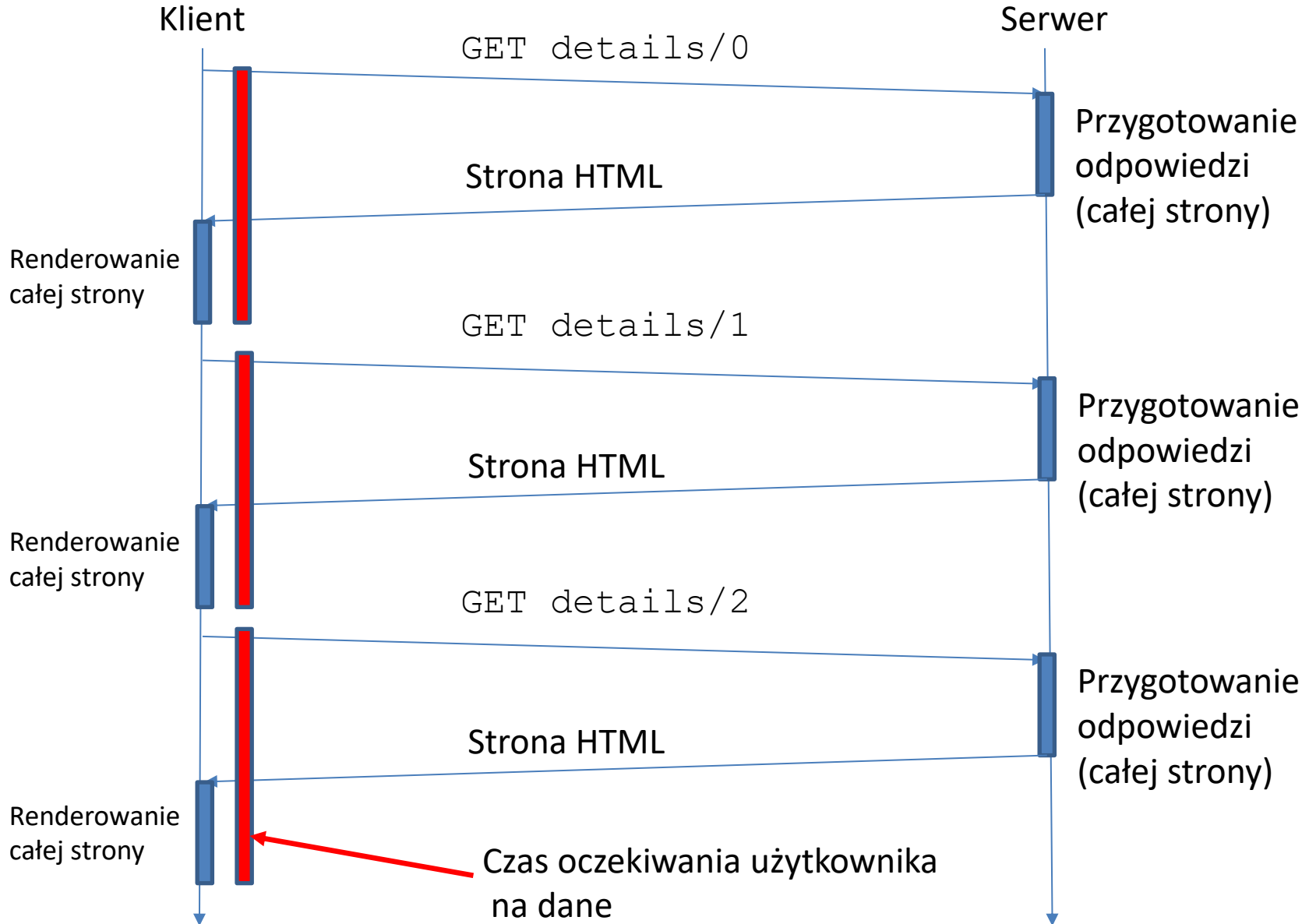
StudentController: [Prev](#) | [Next](#)

JavaScript(AJAX): [Previous](#) | [Next](#)

jQuery(AJAX): [Previous](#) | [Next](#)

| Name | Stat.. | Type | Initiator | Size | T. | Waterfall |
|----------------------|--------|---------|------------------------------------------|------------|------|-----------|
| 0 | 302 | doc... | Other | 70 B | 3... | |
| 1 | 200 | doc... | https://localhost:5001/Student/Details/1 | 5.4 kB | 5... | |
| bootstrap.min.css | 200 | styl... | 1 | (disk c... | 2... | |
| jquery.min.js | 200 | script | 1 | (memo... | 0... | |
| site.css | 200 | styl... | 1 | (disk c... | 1... | |
| bootstrap.bundle.... | 200 | script | 1 | (memo... | 0... | |
| site.js?v=4q1jwFh... | 200 | script | 1 | (memo... | 0... | |

Klasyczna komunikacja



AJAX

- AJAX – (ang. Asynchronous JavaScript and XML) asynchroniczny JavaScript i XML
 - Czyli kod (funkcja) JavaScript, która w sposób asynchroniczny wyśle żądanie do serwera i wykona działanie po otrzymaniu odpowiedzi w czasie, gdy przeglądarka cały czas prezentuje bieżącą stronę WWW użytkownikowi i reaguje na jego działania.
- Używany protokół HTTP(S)
- Możliwość uruchomienia połączenia z klienta do serwera w sposób asynchroniczny:
 - Nawet wiele zapytań jednocześnie
 - Pytania mogą być do różnych domen WWW
 - Odpowiedzią nie musi być strona HTML
 - Może to być strumień w formacie JSON, np. z kontrolera API
- To kod w JavaScriptcie decyduje jak zinterpretować otrzymany strumień danych.

Rozwiązanie z AJAX-em

- Skorzystamy z wcześniej napisanego kontrolera API dla studenta
- **Dodamy do niego dwie akcje** analogiczne jak dla StudentController-a z innymi endpointami:
 - /api/student/prev/{id}
 - /api/student/next/{id}
- W widoku Details należy dołączyć np. przyciski, które zostaną obsłużone w JavaScriptcie (kod HTML być wcześniej, str 26) w funkcjach jsPrev() i jsNext().
 - Aby mogły dostać się do elementów DOM najlepiej nadać im identyfikatory.

StudentApiController.cshtml

```
[HttpGet("prev/{id}")]
public Student? GetPrev(int id) =>
    repository.GetPreviousStudent(id);
[HttpGet("next/{id}")]
public Student? GetNext(int id) =>
    repository.GetNextStudent(id);
```

Details.cshtml

```
<div>
    JavaScript(AJAX):
    <button onclick="jsPrev()"> Previous </button> |
    <button onclick="jsNext()"> Next </button>
</div>
```

Details.cshtml

```
<dl class="row">
    <dt class="col-sm-2">
        @Html.DisplayNameFor(model => model.Id)
    </dt>
    <dd id="id" class="col-sm-10">
        @Html.DisplayNameFor(model => model.Id)
    </dd>
    <dt class="col-sm-2">
        @Html.DisplayNameFor(model => model.Index)
    </dt>
    <dd id="index" class="col-sm-10">
        @Html.DisplayNameFor(model => model.Index)
    </dd>
    <dt class="col-sm-2">
        @Html.DisplayNameFor(model => model.Name)
    </dt>
    <dd id="name" class="col-sm-10">
        @Html.DisplayNameFor(model => model.Name)
    </dd>
</dl>
```

JavaScript z Ajax-em - kod

- Używamy obiektu klasy XMLHttpRequest.

```
@section Scripts{
<script>
    function jsAjax(word, suffix) {
        const xhr = new XMLHttpRequest();
        xhr.onload = function () {
            if (this.status === 200) {
                try {
                    const stud = JSON.parse(this.responseText);
                    //console.log(stud);
                    document.getElementById("id").innerHTML = stud.id;
                    document.getElementById("index").innerHTML = stud.index;
                    document.getElementById("name").innerHTML = stud.name;
                } catch (e) {
                    console.warn('There was an error in JSON. Could not parse.');
```

Details.cshtml

JavaScript z Ajax-em - opis

- Tworzymy obiekt `xhr` typu `XMLHttpRequest`.
 - XML w nazwie oznacza, że powinien być to dokument w formacie XML
 - W praktyce może być dowolny strumień danych
 - Obecnie w przypadku obiektów jest używany format JSON
 - Do prostych obiektów (cały graf byłoby trudno przesłać)
- W polu `onload` tworzymy funkcję, która zostanie wywołana, jeśli komunikacja zakończy się (poprawnie lub niepoprawnie).
 - W tej funkcji `this` oznacza obiekt `xhr`.
 - Pole `this.status` oznacza kod odpowiedzi
 - Pole `this.responseText` oznacza tekst **ładunku** (payload).
 - Kod 204 oznacza pusty ładunek, czyli kontroler API ASP .Net tak koduje wartość `null`.
 - Wówczas pokazujemy alert z odpowiednim napisem.
 - Odpowiedź JSON zamienia jest na **obiekt** JavaScript za pomocą metody `JSON.parse()`.
 - Dane z obiektu przepiszemy do odpowiednich obiektów DOM.
 - W przypadku niepowodzenia komunikacji na konsolę Javascriptu wypisujemy odpowiedni komunikat
- Przygotowujemy dane do żądania korzystając z obecnej zawartości elementu o identyfikatorze „id”, jednak może on mieć różne białe znaki przed i za wartością liczbową więc usuwamy je metodą `trim()`.
- Metoda `xhr.open()` przygotowuje strumień żądania.
 - Podajemy w niej typ żądania oraz URL.
 - Można jeszcze nim manipulować np. dodając zawartość lub parametry w nagłówkach żądania.
- Ostateczne wysłanie żądania w sposób asynchroniczny następuje po wywołaniu metody `xhr.send()`.
 - Przeglądarka działa niezależnie
 - Po odebraniu odpowiedzi wywołana zostanie metoda zapamiętana w `xhr.onload`.
- Ogólna metoda `jsAjax(word, suffix)` wymaga podania słowa `word`, które pojawi się w alercie, gdy nie ma poprzedniego/następnego elementu oraz końcówki `suffix` adresu URL na który ma być wysłane żądanie.

JavaScript z Ajax-em – działanie 1/4

- Akcja Index.

The screenshot shows a web browser at the URL `https://localhost:5001/Student`. The page displays a table with student information and a 'Details' link for each row. The 'Details' link for the first row (Id: 0, Name: Smith) is highlighted with a green box. A blue arrow points from this link to the Network tab in the browser's developer tools. The Network tab shows a list of requests, with the first request being a GET request to `Student` (3.6 kB).

Index
[Create New](#)

| Id | Index | Name | Details |
|----|-------|-----------|-------------------------|
| 0 | 11111 | Smith | Details |
| 1 | 22222 | Kowal | Details |
| 2 | 33333 | Schneider | Details |

Network

| Name | Stat.. | Type | Initiator | Size | T. | Waterfall |
|----------------------|--------|---------|-------------|----------|------|-----------|
| Student | 200 | doc... | Other | 3.6 kB | 1... | |
| bootstrap.min.css | 200 | styl... | Student | (memo... | 0... | |
| site.css | 200 | styl... | Student | (memo... | 0... | |
| jquery.min.js | 200 | script | Student | (memo... | 0... | |
| bootstrap.bundle.... | 200 | script | Student | (memo... | 0... | |
| site.js?v=4q1jwFh... | 200 | script | Student | (memo... | 0... | |
| data:image/svg+... | 200 | svg... | bootstra... | (memo... | 0... | |

JavaScript z Ajax-em – działanie 2/4

- Akcja Details(0).

The screenshot shows a web browser at `https://localhost:5001/Student/Details/0`. The page displays details for a student with ID 0, Index 11111, and Name Smith. Navigation links include "Back to List", "StudentController: Prev", "Next", "JavaScript(AJAX): Previous", "Next", and "jQuery(AJAX): Previous", "Next". The "Next" link in the JavaScript(AJAX) section is highlighted with a green box, and a blue arrow points from it to the bottom right of the slide.

The browser's Network tab is open, showing a list of requests. The first request, labeled "0", is highlighted with a red box. It is a document request (Type: doc...) initiated by "Other", with a size of 5.5 kB and a status of 200. The Waterfall view shows the request duration.

| Name | Stat.. | Type | Initiator | Size | T. | Waterfall |
|----------------------|--------|---------|-----------|----------|------|-----------------|
| 0 | 200 | doc... | Other | 5.5 kB | 3... | [Waterfall bar] |
| bootstrap.min.css | 200 | styl... | 0 | (memo... | 0... | [Waterfall bar] |
| site.css | 200 | styl... | 0 | (memo... | 0... | [Waterfall bar] |
| jquery.min.js | 200 | script | 0 | (memo... | 0... | [Waterfall bar] |
| bootstrap.bundle... | 200 | script | 0 | (memo... | 0... | [Waterfall bar] |
| site.js?v=4q1jwFh... | 200 | script | 0 | (memo... | 0... | [Waterfall bar] |

JavaScript z Ajax-em – działanie 3/4

- Jesteśmy w ramach jednego głównego żądania Details/0.
- Pojawiło się jedno **dodatkowe żądanie**
`https://localhost:5001/api/student/next/0`
- Liczba przesłanych danych bardzo się zmniejszyła z 5,5 KB do 135 B.
- Przeglądarka nie odświeża całej strony, tylko 3 elementy DOM.

The screenshot shows a web browser at `https://localhost:5001/Student/Details/0`. The page title is "Details Student". It displays the following information:

- Id:** 1
- Index:** 22222
- Name:** Kowal

Navigation links include "Back to List", "StudentController: Prev | Next", "JavaScript(AJAX): Previous | Next", and "jQuery(AJAX): Previous | Next".

The Network tab in the browser's developer tools shows a list of resources. The last resource, `https://localhost:5001/api/student/next/0`, is highlighted. It is an XHR request with a status of 200, a size of 135 B, and a response time of 0:122. The previous resources (bootstrap.min.css, site.css, jquery.min.js, bootstrap.bundle..., site.is?v=4n1wFh) are all 200 status with sizes ranging from 5.4 kB to 0 B.

| Name | Stat | Type | Initiator | Size | T | Waterfall |
|---------------------|------|---------|-----------|----------|------|-------------|
| 0 | 200 | doc... | Other | 5.4 kB | 1... | |
| bootstrap.min.css | 200 | styl... | 0 | (memo... | 0... | |
| site.css | 200 | styl... | 0 | (memo... | 0... | |
| jquery.min.js | 200 | script | 0 | (memo... | 0... | |
| bootstrap.bundle... | 200 | script | 0 | (memo... | 0... | |
| site.is?v=4n1wFh | 200 | script | 0 | (memo... | 0... | |
| 0 | 200 | xhr | 0:122 | 135 B | 5... | kontynuacja |

JavaScript z Ajax-em – działanie 4/4

- Po **trzecim** wciśnięciu klawisza „Next” (na ostatnim studencie w bazie) pojawia się odpowiedni komunikat.

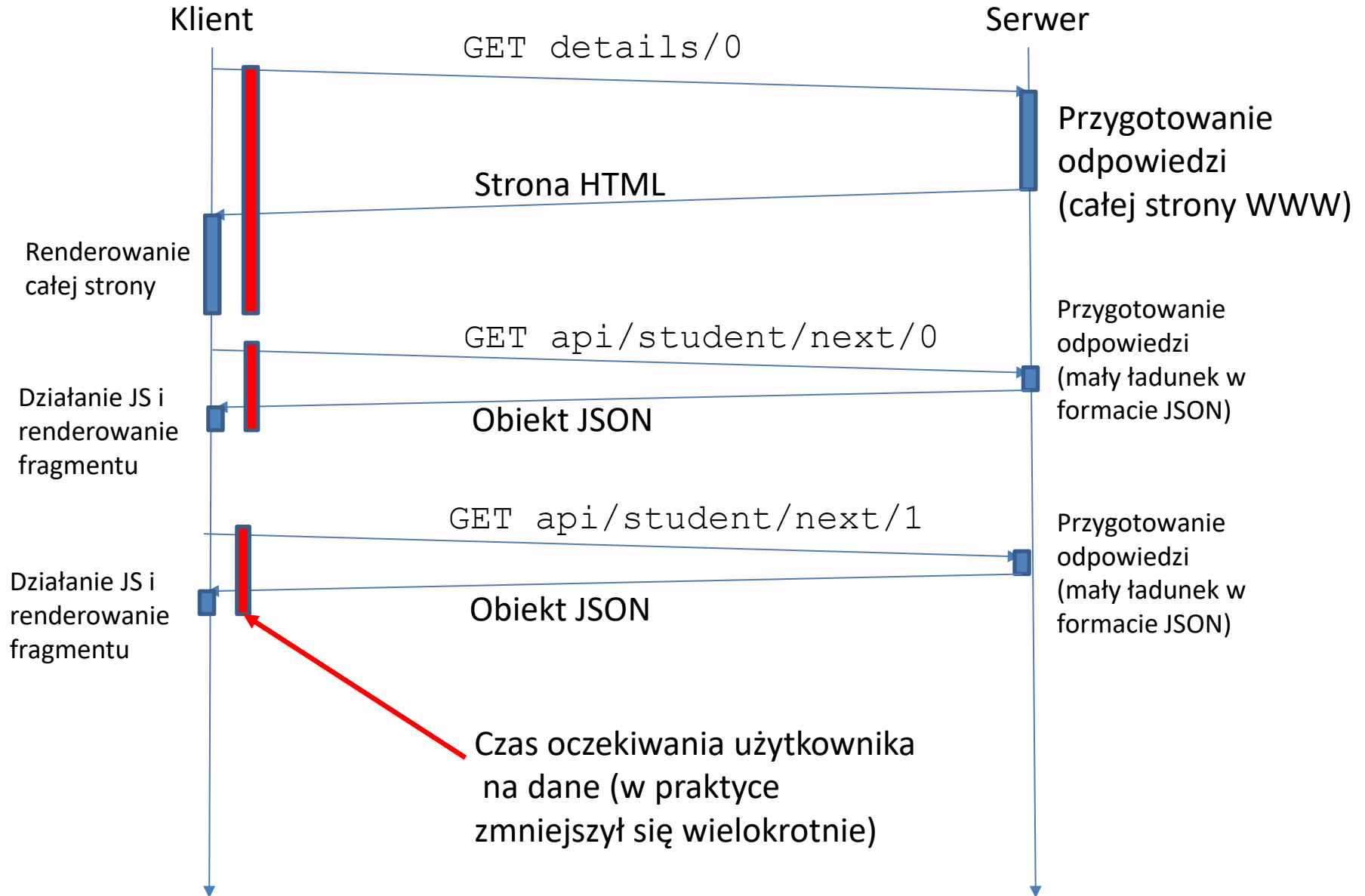
The screenshot shows a web browser at `https://localhost:5001/Student/Details/0`. The page displays student details for Id 2, Index 33333, and Name Schneider. Navigation links include "Back to List", "StudentController: Prev | Next", "JavaScript(AJAX): Previous | Next", and "jQuery(AJAX): Previous | Next". The "Next" button in the JavaScript(AJAX) section is highlighted with a green box. A blue arrow points from this button to a modal dialog box titled "Komunikat z witryny localhost:5001:" with the message "No next element" and an "OK" button.

Below the page, a network log is visible. The log table has columns: Name, Stat., Type, Initiator, Size, T., and Waterfall. The following table represents the data in the log:

| Name | Stat.. | Type | Initiator | Size | T. | Waterfall |
|----------------------|--------|---------|-----------|----------|------|-----------|
| 0 | 200 | doc... | Other | 5.4 kB | 1... | |
| bootstrap.min.css | 200 | styl... | 0 | (memo... | 0... | |
| site.css | 200 | styl... | 0 | (memo... | 0... | |
| jquery.min.js | 200 | script | 0 | (memo... | 0... | |
| bootstrap.bundle... | 200 | script | 0 | (memo... | 0... | |
| site.js?v=4q1jwFh... | 200 | script | 0 | (memo... | 0... | |
| 0 | 200 | xhr | 0:122 | 135 B | 5... | |
| 1 | 200 | xhr | 0:122 | 147 B | 3... | |
| 2 | 204 | xhr | 0:122 | 46 B | P... | |

Green boxes highlight the "Next" button in the UI, the "204" status code in the network log, and the "135 B" and "147 B" sizes in the network log. The word "kontynuacja" is written in green next to the last row of the network log.

Komunikacja z AJAX-em



Dodatek - Ajax w jQuery

- Najniższe przyciski w widoku `Details` przygotowane są dla funkcji napisanych w jQuery.
 - Zapis bardziej skompresowany niż bezpośrednio w Javascriptcie

```
function jqAjax(word, suffix) {  
    $.ajax({  
        type: "GET",  
        url: "/api/student/" + suffix + "/" + $("#id").html().trim(),  
        success: function (stud, textStatus, jqXHR) { // codes 200..299  
            if (jqXHR.status === 204) { // 204 No Content, so NULL  
                window.alert("No " + word + " element");  
                return;  
            }  
            $('#id').html(stud.id);  
            $('#index').html(stud.index);  
            $('#name').html(stud.name);  
        }  
    })  
    .fail(function (jqXHR, textStatus) { // codes 400..499  
        console.warn("Recived " + jqXHR.status + " in response code.");  
    });  
}  
  
function jqPrev(){  
    jqAjax("previous", "prev");  
}  
  
function jqNext() {  
    jqAjax("next", "next");  
}  
</script>
```

Details.cshtml

Visual Studio 2022 – projekt „API kontroler”

- Gotowy szablon dla projektu tylko z kontrolerem API



Internetowy interfejs API platformy ASP.NET Core

Szablon projektu służący do tworzenia aplikacji platformy ASP.NET Core z przykładowym kontrolerem obsługującym usługę HTTP RESTful. Tego szablonu można także użyć dla widoków i kontrolerów platformy ASP.NET Core MVC.

C#

Linux

macOS

Windows

Chmura

Usługa

Internet

WebAPI

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllers(); // enough for API controllers
});
```

Publiczne RESTful kontrolery

- Zamiast parsować dokument HTML i szukać potrzebnych danych, lepiej użyć kontrolerów API (jeśli twórca strony z danymi takie udostępnia)
- Dostęp do kontrolerów RESTfull powinno się połączyć z autoryzacją i autentykacją.
- Część portali posiada dostępne publiczne serwery RESTfull
 - Dla programistów
 - Często wymagają rejestracji
 - Dla dużej liczby zapytań są płatne
 - Głównie operacja GET
- Przykład: Google Maps
 - <https://developers.google.com/maps/documentation>

Wady/zalety kontrolerów API z prostą obsługą Ajax-em

- Zalety:
 - Mniej danych jest przesyłanych
 - Szybsze działanie strony
 - Projekt można łatwiej podzielić na część backend-ową z kontrolerami API oraz część frontend-ową z AJAX-em w JavaScript
- Wady:
 - 1) Potrzebny kod w Javascriptcie
 - 2) trudno automatycznie podlinkować stronę (URL się nie zmienia)
 - 3) mieszanie kodu Razora z kodem Javascript
- Rozwiązanie wad:
 - Ad 1) Istnieją framework-i ułatwiające użycie AJAX-a
 - Ad 1) Albo nawet rozwiązania typu Blazor Web Assembly, gdzie można pisać w C#
 - Ad 2) rozwiązanie mieszane: kontroler MVC i kontrolerAPI, oraz przycisk pobierania linku do strony, który będzie obsługiwany przez kontroler MVC.
 - Ad 2) i 3) Frameworki frontendowe pozwalające pisać całą aplikację jako SPA (Single Page Application).
 - Zamiast używania Razor-a
- W kolejnych wykładach będzie wprowadzenie do framework-a Angular użytego do pisania frontend-u w TypeScriptie.
 - Jako backend posłuży aplikacja .Net jako kontroler API.