

# Programowanie wizualne

opracował: Wojciech Frohmberg

## Lab 9

---

Zagadnienia do opanowania:

- Asp.net Identity
  - Web API
- 

Zarys problemu:

Kontynuując projekt z naszego cyklu zajęciowego chcielibyśmy zabezpieczyć go przed nieautoryzowanym użytkowaniem. Do tego celu wykorzystamy Asp.net Identity. Celem prostego zarządzania elementami tożsamości (użytkownikami, tj. w naszym przypadku deweloperami, oraz rolami) użyjemy Web API. Niestety przez użycie klasy dewelopera jako użytkownika aplikacji będziemy musieli zrezygnować z wygenerowanego sposobu zarządzania deweloperami przez kontroler `DeveloperController`. Będzie nam go musiało zastąpić utworzone Web API.

Dołączenie i podpięcie biblioteki do projektu:

1. W ramach projektu `AssignmentSharing` zainstaluj pakiety:
  - `Microsoft.AspNetCore.Identity.UI`
  - `Microsoft.AspNetCore.Identity.EntityFrameworkCore`

np. przy użyciu komend (w przypadku VS2022):

```
Install-Package Microsoft.AspNetCore.Identity.UI
Install-Package Microsoft.AspNetCore.Identity.EntityFrameworkCore
```

lub (w przypadku dotnet cli):

```
dotnet add package Microsoft.AspNetCore.Identity.UI
dotnet add package Microsoft.AspNetCore.Identity.EntityFrameworkCore
```

(przy czym warto skorzystać tu z dodatkowego przełącznika `--version` określając wersję zgodną z pozostałymi zainstalowanymi nugetami np. 7.0.13)

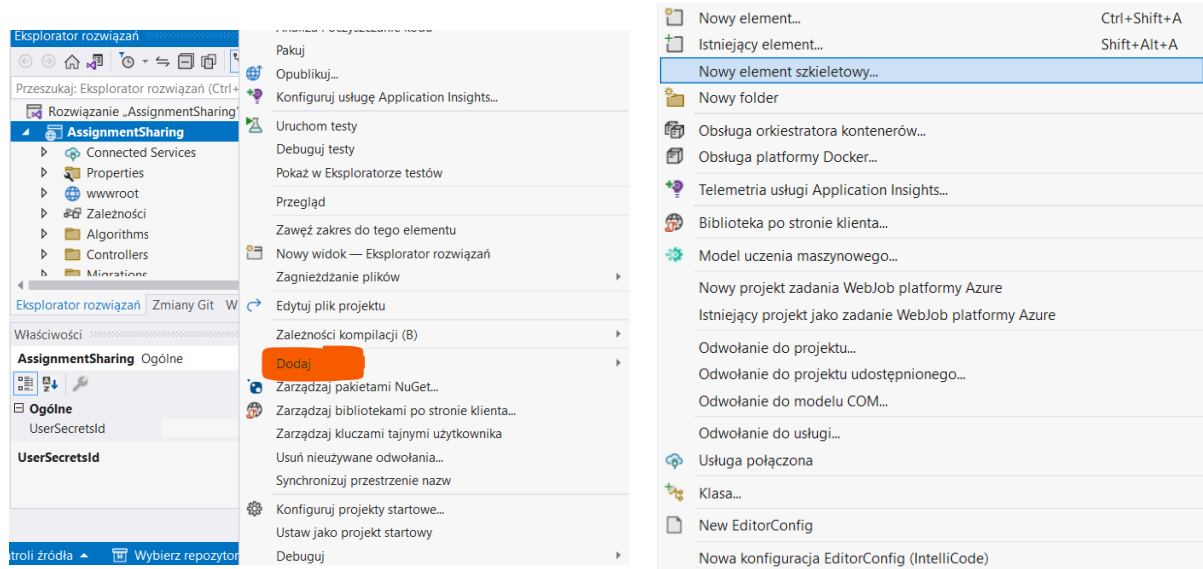
Jeśli nie masz jeszcze zainstalowanego narzędzia służącego do generowania kodu skorzystaj z następującej komendy:

```
dotnet tool install dotnet-aspnet-codegenerator --version 7.0.11
```

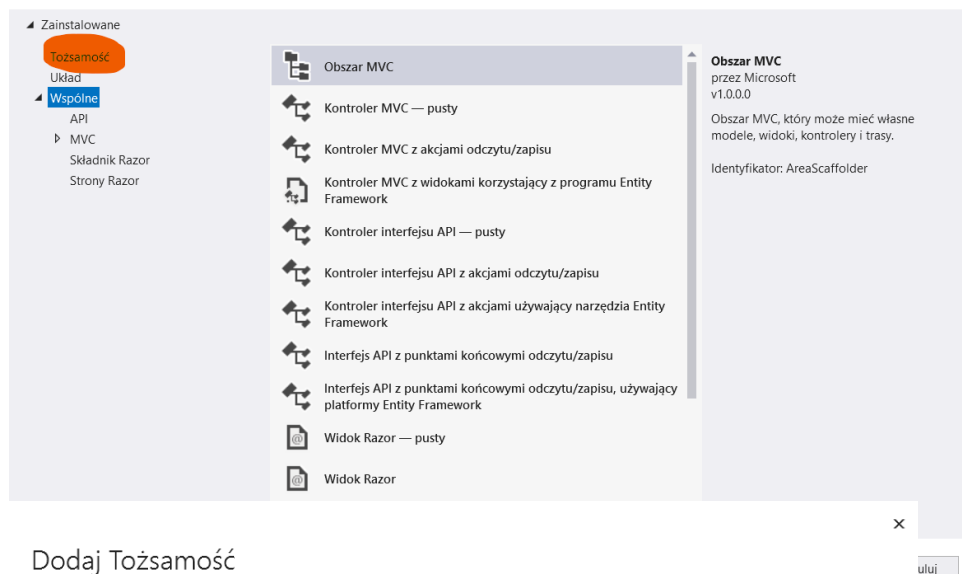
Pamiętaj że komenda ta wymaga posiadania utworzonego manifestu (`dotnet new tool-manifest`)

2. Klasę modelu `Models.Developer` wydziedzicz z klasy `IdentityUser<Guid>` i zadбай o to żeby wszystkie elementy klasy typu string były nullable (zmieniając ich typ na „string?”).
3. Właściwość `Id` zmień tak by była nadpisaniem właściwości `Id` rodzica tj. klasy `IdentityUser` poprzez dodanie słówka `override` po modyfikatorze dostępu. Nad właściwością dodaj atrybut `[DatabaseGenerated(DatabaseGeneratedOption.Identity)]`

- Klasę kontekstu DataContext zamiast z DbContext wydziedzicz z klasy IdentityDbContext<Developer, IdentityRole<Guid>, Guid>.
- Do projektu AssignmentSharing dodaj nowy element szkieletowy. Jeśli używasz VS2022 zrobisz to z menu kontekstowego projektu. W ramach otwartego dialogu wybierz tu element typu Tożsamość. Zaznacz checkbox „Zastąp wszystkie pliki”, ale nie generuj nowej klasy kontekstu. Zamiast tego wybierz istniejącą klasę DataContext jako klasę kontekstu dla tożsamości.



Dodaj nowy element szkieletowy



Dodaj Tożsamość

Wybierz istniejącą stronę układu lub określ nową:  
/Areas/Identity/Pages/Account/Manage/\_Layout.cshtml  
(Zostaw pustą, jeśli jest ustawiona w pliku Razor \_viewstart)

☒ Zastąp wszystkie pliki

Wybierz pliki do zastąpienia

<input checked="" type="checkbox"/> Account\StatusMessage	<input checked="" type="checkbox"/> Account\AccessDenied	<input checked="" type="checkbox"/> Account\ConfirmEmail
<input checked="" type="checkbox"/> Account\ConfirmEmailChange	<input checked="" type="checkbox"/> Account\ExternalLogin	<input checked="" type="checkbox"/> Account\ForgotPassword
<input checked="" type="checkbox"/> Account\ForgotPasswordConfirmation	<input checked="" type="checkbox"/> Account\Lockout	<input checked="" type="checkbox"/> Account>Login
<input checked="" type="checkbox"/> Account>LoginWith2fa	<input checked="" type="checkbox"/> Account>LoginWithRecoveryCode	<input checked="" type="checkbox"/> Account\Logout
<input checked="" type="checkbox"/> Account\Manage\Layout	<input checked="" type="checkbox"/> Account\Manage\ManageNav	<input checked="" type="checkbox"/> Account\Manage\StatusMessage
<input checked="" type="checkbox"/> Account\Manage\ChangePassword	<input checked="" type="checkbox"/> Account\Manage\DeletePersonalData	<input checked="" type="checkbox"/> Account\Manage\Disable2fa
<input checked="" type="checkbox"/> Account\Manage\DownloadPersonalData	<input checked="" type="checkbox"/> Account\Manage\Email	<input checked="" type="checkbox"/> Account\Manage\EnableAuthenticator
<input checked="" type="checkbox"/> Account\Manage\ExternalLogins	<input checked="" type="checkbox"/> Account\Manage\GenerateRecoveryCodes	<input checked="" type="checkbox"/> Account\Manage\Index
<input checked="" type="checkbox"/> Account\Manage\PersonalData	<input checked="" type="checkbox"/> Account\Manage\ResetAuthenticator	<input checked="" type="checkbox"/> Account\Manage\SetPassword
<input checked="" type="checkbox"/> Account\Manage\ShowRecoveryCodes	<input checked="" type="checkbox"/> Account\Manage\TwoFactorAuthentication	<input checked="" type="checkbox"/> Account\Register
<input checked="" type="checkbox"/> Account\RegisterConfirmation	<input checked="" type="checkbox"/> Account\ResendEmailConfirmation	<input checked="" type="checkbox"/> Account\ResetPassword
<input checked="" type="checkbox"/> Account\ResetPasswordConfirmation		

Klasa DbContext:

Dostawca bazy danych:

Klasa użytkownika:

Jeśli z kolei korzystasz z dotnet cli do utworzenia

dotnet aspnet-codegenerator identity --dbContext DataContext

6. W głównym folderze projektu pojawił się folder Areas, z szablonem poszczególnych składowych, które możemy wykorzystywać do wizualizowania tożsamości.
7. W ramach pliku Views → Shared → Layout.cshtml na koniec głównego elementu div navbaru dodaj:  
<partial name="\_LoginPartial"/>
8. Z pliku Program.cs zakomentuj/usuń wygenerowaną linię:  
var connectionString = ...  
Dodaj odpowiedni using wymagany przez linię:  
builder.Services.AddDefaultIdentity<Developer>(...);  
a przed linią app.Run(); dodaj:  
app.MapRazorPages();
9. Dodaj migrację i odśwież bazę danych

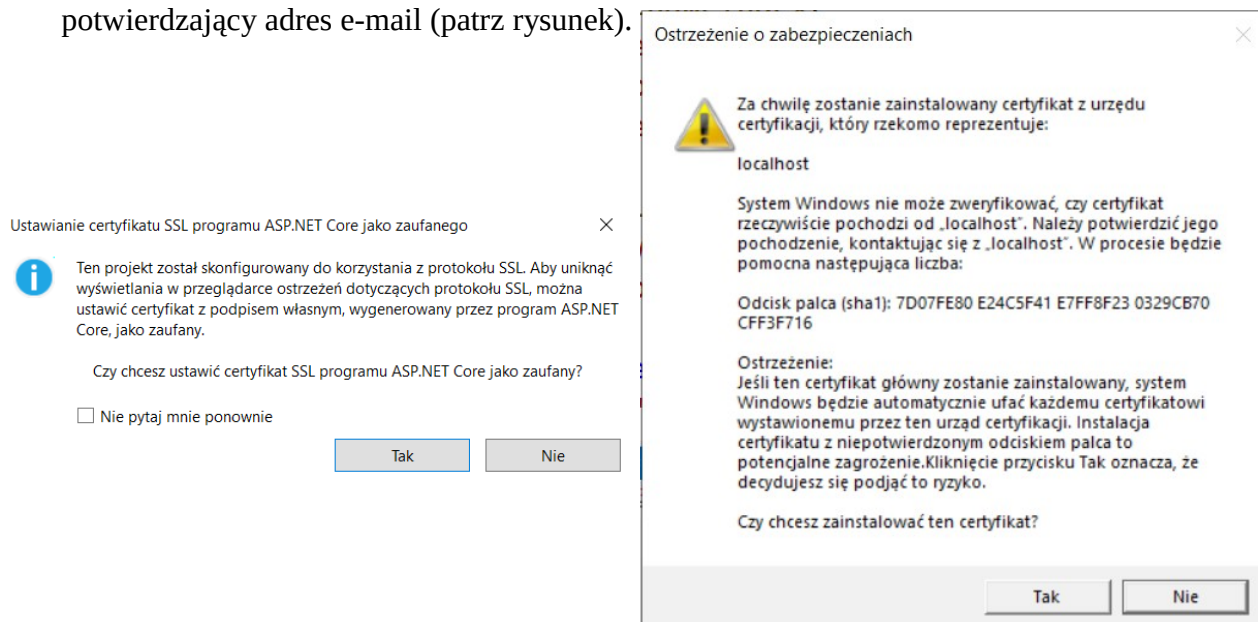
W przypadku VS2022:

Add-Migration AssignmentSharingIdentity  
Update-Database

W przypadku dotnet cli:

dotnet ef migrations add AssignmentSharingIdentity  
dotnet ef database update

10. Przetestuj aplikację poprzez zarejestrowanie użytkownika przyciskiem dostępnym z paska nawigacyjnego (pamiętaj by zaakceptować dodanie certyfikatu SSL, wymaganego przez aplikację do korzystania z protokołu https). Po zarejestrowaniu naciśnij przycisk potwierdzający adres e-mail (patrz rysunek).



## Register confirmation

This app does not currently have a real email sender registered, see [these docs](#) for how to configure a real email sender. Normally this would be emailed to [your account](#). [Click here to confirm your account](#)

11. Uzależnij możliwość skorzystania z kontrolera Assignments od tego czy użytkownik jest zalogowany – do tego celu przed deklaracją klasy skorzystaj z atrybutu [Authorize]. Następnie przetestuj czy da się wejść na adres `https://localhost:<port>/Assignments/` bez uprzedniego zalogowania.

Póki co wszyscy deweloperzy po zarejestrowaniu mają takie same uprawnienia i możemy nieautoryzować do oglądania jakichś treści na naszej stronie jeśli użytkownik nie przeszedł poprawnie czynności logowania. Żeby przydzielić niejednorodne uprawnienia deweloperom możemy przypiąć pod nich odpowiednie role. Poniżej przedstawiono jeden z możliwych sposobów zarządzania rolami.

Tworzenie API służącego zarządzaniu rolami:

1. W pliku Program.cs dodaj do ciągu instrukcji rozpoczynających się od:  
`builder.Services.AddDefaultIdentity<Developer>(...)`  
metodę wpinającą usługi Managera kont użytkowników i ról oraz zadeklaruj typ roli:  
`.AddRoles<IdentityRole<Guid>>()`  
`.AddUserManager<UserManager<Developer>>()`  
`.AddRoleManager<RoleManager<IdentityRole<Guid>>>()`  
(Upewnij się że elementy będą dodane przed wywołaniem metody  
`.AddEntityFrameworkStores<DataContext>()`).
2. Do struktury folderów projektu dodaj folder ApiControllers.
3. Do folderu dodaj nowy element kontrolera API. Jeśli korzystasz z VS2022 z menu kontekstowego folderu ApiController wybierz opcję „Dodaj” -> „Kontroler interfejsu API z akcjami odczytu/zapisu”  
(Zainstalowane → Wspólne → API). Jeśli przy dodawaniu nowego elementu uruchomił się tryb kompaktowy skorzystaj z przycisku „Pokaż wszystkie szablony”. Kontroler nazwij RolesController.

Jeśli korzystasz z dotnet cli skorzystaj z następującej komendy:

```
dotnet aspnet-codegenerator controller -name RolesController -actions -api  
-outDir ApiControllers
```

4. Do kontrolera wstrzyknij (poprzez mechanizm Dependency Injection) managera ról oraz managera użytkowników np. poprzez dodanie do kontrolera kodu:

```
private readonly UserManager<Developer> _userManager;  
private readonly RoleManager<IdentityRole<Guid>> _roleManager;  
  
public RolesController(UserManager<Developer> userManager,  
                       RoleManager<IdentityRole<Guid>> roleManager)  
{  
    _userManager = userManager;  
    _roleManager = roleManager;  
}
```

5. Implementację metody Get (nie przyjmującej w parametrze id) zastąp kodem:

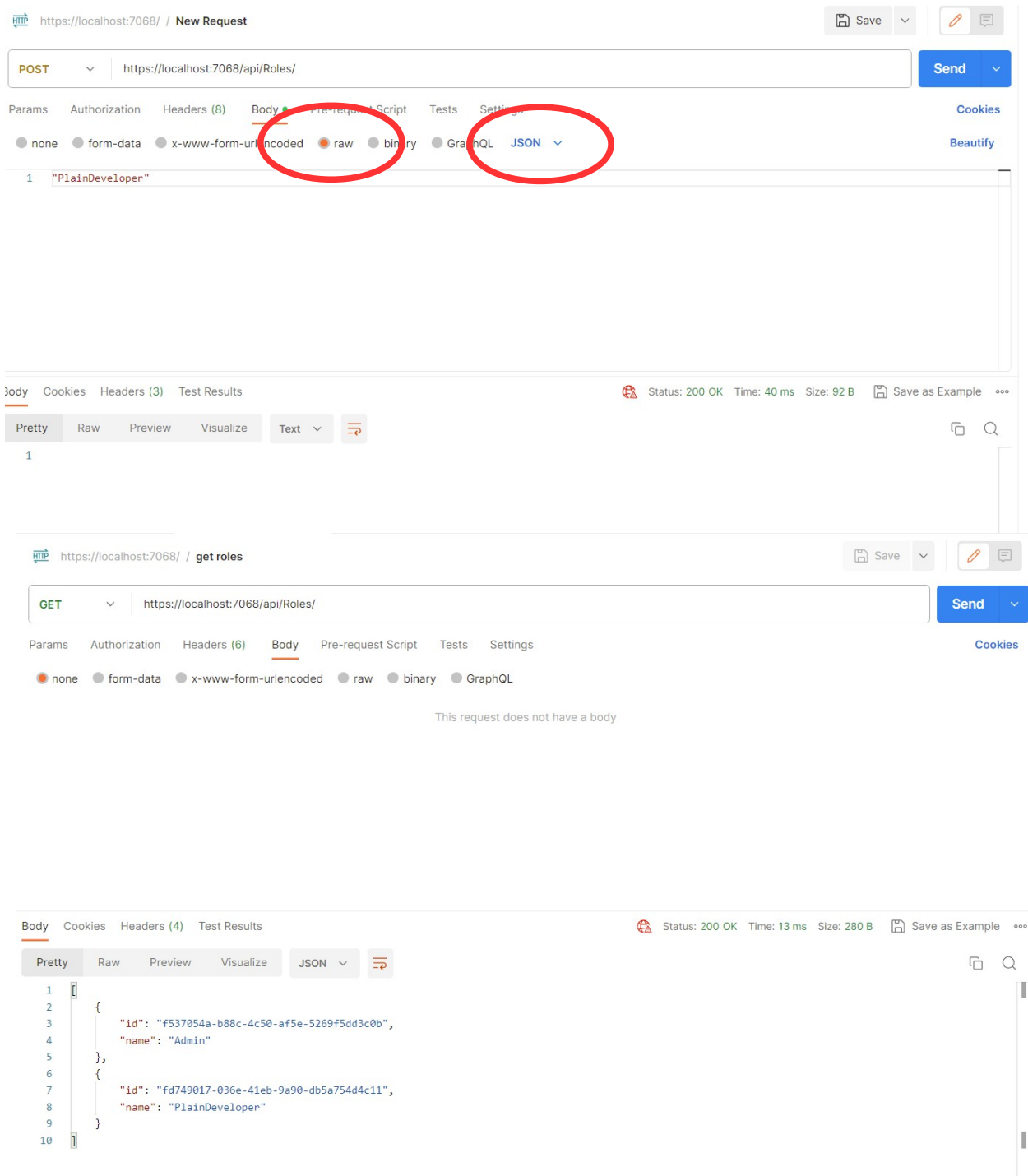
```
[HttpGet]  
public IEnumerable<dynamic> Get()  
{  
    return _roleManager.Roles.Select(r => new { r.Id, r.Name }).ToList();  
}
```

6. Implementację metody Post zastąp kodem:

```
[HttpPost]
```

```
public void Post([FromBody] string value)
{
    _roleManager.CreateAsync(new IdentityRole<Guid>(value)).Wait();
}
```

## 7. Przetestuj działanie api np. przy użyciu narzędzia Postman



The top screenshot shows a POST request in Postman to the endpoint `https://localhost:7068/api/Roles/`. The request type is set to `POST`. The body is set to `JSON` and contains the string `"PlainDeveloper"`. The status of the request is `200 OK` with a response time of `40 ms` and a size of `92 B`.

The bottom screenshot shows a GET request to the same endpoint `https://localhost:7068/api/Roles/`. The request type is set to `GET`. The response body is displayed in JSON format, showing two roles:

```
[
  {
    "id": "f537054a-b88c-4c50-af5e-5269f5dd3c0b",
    "name": "Admin"
  },
  {
    "id": "fd749017-036e-41eb-9a90-db5a754d4c11",
    "name": "PlainDeveloper"
  }
]
```

- Póki co API jest dostępne publicznie bez ograniczeń. Celem zabezpieczenia akcji kontrolera możemy dodać klasę Middleware. W dużym uproszczeniu middleware stanowi warstwę pośredniczącą między naszą aplikacją a naszym API. W ramach tej warstwy możemy przetwarzać zapytania do nas dostarczone fragment po fragmencie. W naszym przypadku możemy posłużyć się Middlewarem do przefiltrowania zapytań które nie są w jakiś sposób autoryzowane. Na nasze potrzeby możemy ustalić sposób autoryzacji do

api poprzez klucz – prosty ciąg znaków, którym użytkownik API będzie musiał się posłużyć żeby skorzystać z określonych funkcjonalności.

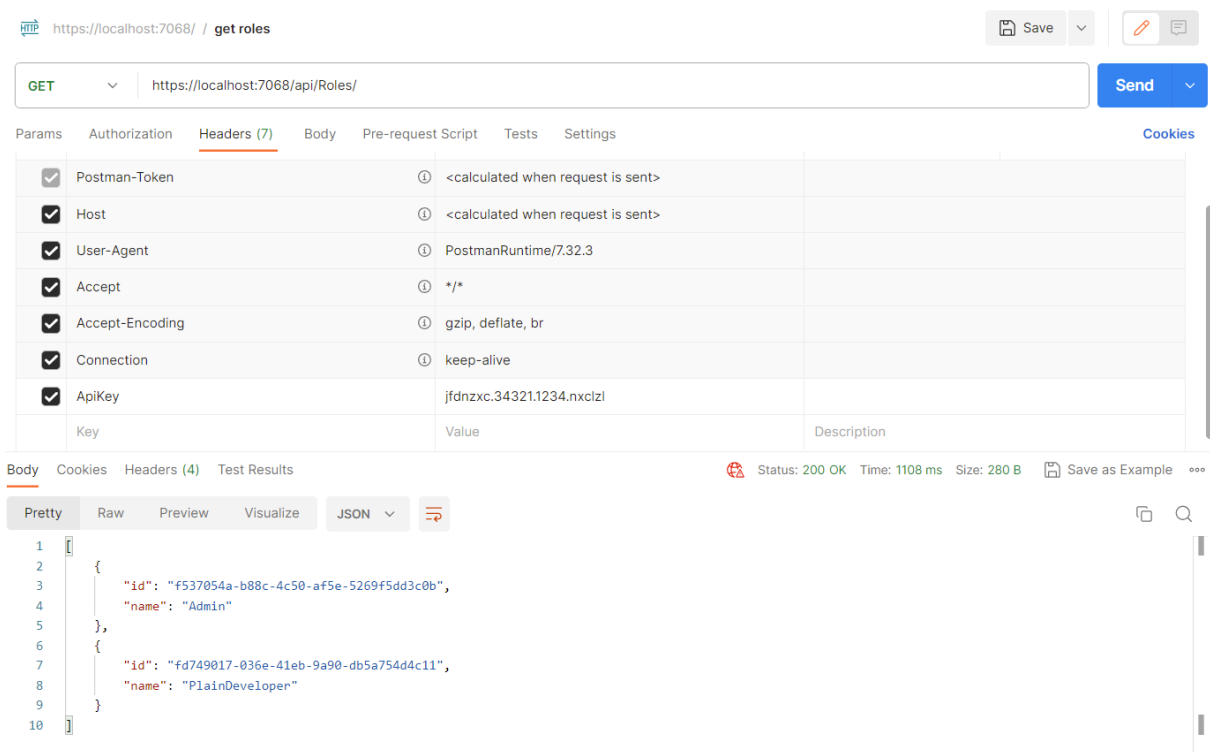
W tym celu do folderu ApiControllers dodaj klasę o nazwie `ApiKeyMiddleware`. Żeby skorzystać z klasy jako z middlewarowego filtra musi ona zawierać metodę `Configure`. Dodaj ją zatem z przykładową implementacją:

```
public void Configure(IApplicationBuilder app)
{
    app.Use(async (context, next) =>
    {
        IConfiguration configuration =
            context.RequestServices.GetRequiredService<IConfiguration>();
        if (context.Request.Headers.TryGetValue("ApiKey", out var apiKey)
            && apiKey == configuration["ApiKey"])
            await next.Invoke();
        else
        {
            context.Response.StatusCode = 401;
            context.Response.WriteAsync("Unauthorized").Wait();
        }
    });
}
```

9. Do pliku `appsettings.json` dodaj wpis `ApiKey` z wybranym przez Ciebie ciągiem będącym kluczem do API. Np.

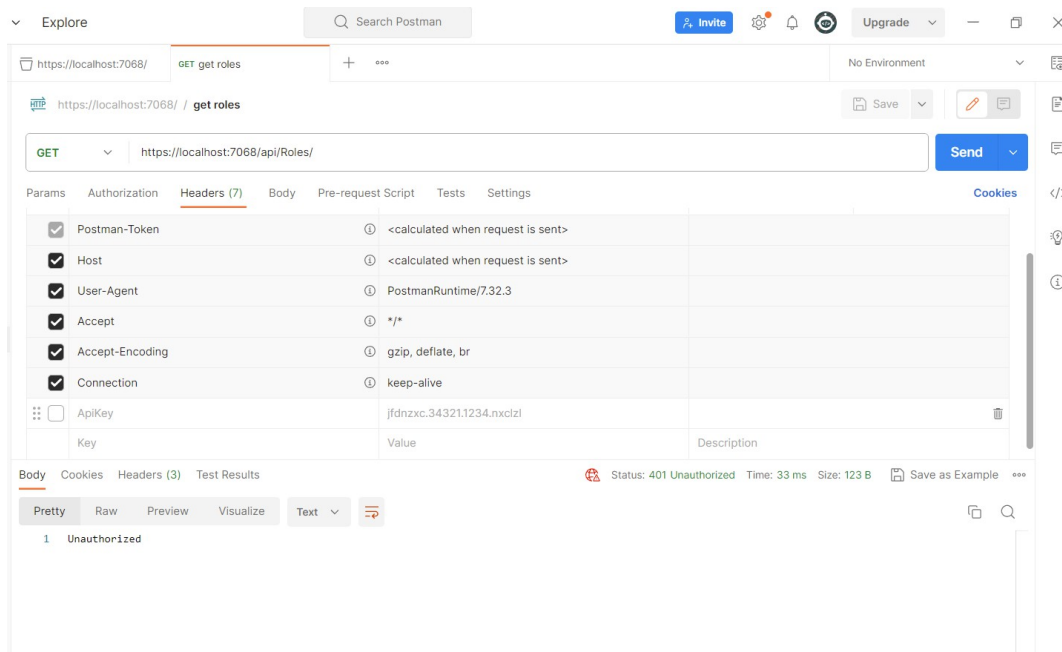
```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "ConnectionStrings": {
    "Sqlite": "Data source=assignments.db"
  },
  "AllowedHosts": "*",
  "ApiKey": "jfdnzxc.34321.1234.nxc1z1"
}
```

10. Klasę `RolesController` oznacz dodatkowo atrybutem: `[MiddlewareFilter(typeof(ApiKeyMiddleware))]`
11. Przetestuj uruchomienie metody `Get` oraz `Post` bez podawania klucza i podając w ramach klucza w ramach nagłówka requestu:



12. Zaproponuj implementację pozostałych metod kontrolera.

13. Dodaj nowy



kontroler API UserRoleBindingsController i wstrzyknij w niego (mechanizmem Dependency Injection) manager ról oraz użytkowników - deweloperów (w razie problemów patrz punkt 4).

14. Dodaj do kontrolera Middleware filtrujący żądania bez klucza (w razie problemów patrz punkt 10).

15. Zastąp kod metody post następującym kodem:

```

public class PostHelper
{
    public string userName { get; set; }
}

```

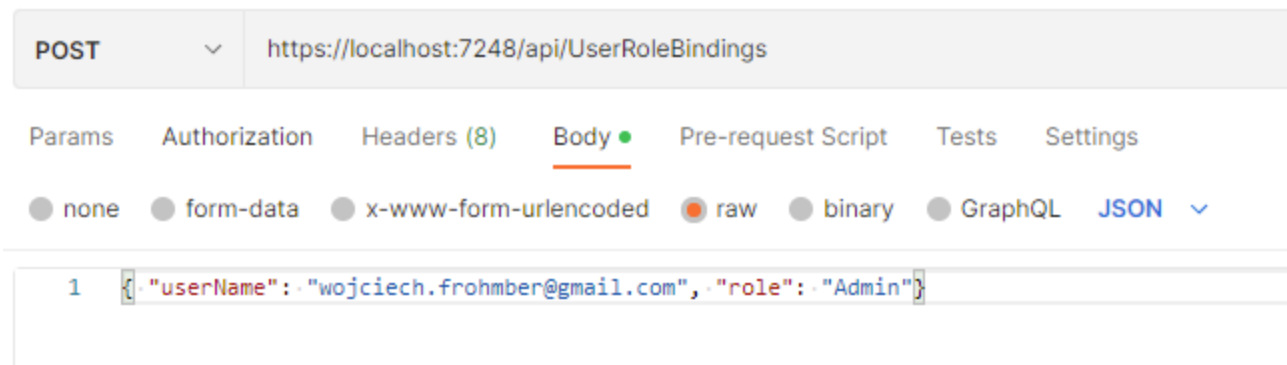
```

        public string role { get; set; }
    }

    // POST api/<UserRoleBindingsController>
    [HttpPost]
    public void Post([FromBody] PostHelper parameters)
    {
        Task<Developer?> task = _userManager.FindByNameAsync(parameters.userName);
        task.Wait();
        var user = task.Result;
        _userManager.AddToRoleAsync(user, parameters.role);
    }

```

16. Przetestuj działanie metody przy użyciu Postmana (pamiętaj o dodaniu klucza API)



17. Dodaj do akcji Privacy kontrolera Home zabezpieczenie przed nieautoryzowanym użyciem wymagające roli Admin użytkownika. (Użyj atrybutu `[Authorize(Roles = \"Admin\")]`).
18. Zaproponuj i przetestuj ograniczenia wykorzystujące niestandardowe zasady (politykę). Przykładowa polityka zdefiniowana w ramach Program.cs:

```

builder.Services.AddAuthorization(options =>
{
    options.AddPolicy("pseudonims_with_ski_ending_only", policy =>
    policy.RequireAssertion(
        context =>
        {
            if (context.Resource is HttpContext httpContext)
            {
                if (httpContext?.User?.Identity?.Name == null)
                {
                    return false;
                }
                var dataContext = httpContext.RequestServices
                    .GetRequiredService<DataContext>();
                var userManager = httpContext.RequestServices
                    .GetRequiredService<UserManager<Developer>>();
                var userTask = userManager
                    .FindByNameAsync(httpContext.User.Identity.Name);
                userTask.Wait();
                var user = userTask.Result;
                var dev = dataContext.Developers.Find(user.Id);
                return dev.Pseudonym.EndsWith("ski");
            }
        }
    );
});

```



```
        return false;
    });
};
```

Zaaplikowanie polityki:

```
[Authorize(Policy = "pseudonims_with_ski_ending_only")]
```

19. Utwórz urząd certyfikujący prawo jazdy poprzez kontroler API przyjmujący w ciele żądania typu post ciąg znaków z nazwą użytkownika, któremu należy nadać uprawnienie do prowadzenia pojazdu. Pamiętaj by każdemu z certyfikowanych użytkowników nadać unikalny numer dokumentu.
20. Utwórz politykę, która zezwala na dostęp tylko użytkownikom z nadanym prawem jazdy.
21. Zabezpiecz akcję Privacy w kontrolerze Home tak by był możliwy do niej dostęp zgodnie z utworzoną w poprzednim kroku polityką.