

Platformy programistyczne .Net i Java

Projekt aplikacji webowej w technologii ASP.NET Core

Mateusz Marciak 272599

Opis Projektu

W tym projekcie, aplikacja webowa w technologii ASP .NET Core została podzielona na dwie części: **Webapp** oraz **Web_database_app**.

- **Webapp** jest prostą modyfikacją podstawowego szablonu utworzonego przy pomocy frameworku **Blazor**.
- **Web_database_app** jest również utworzony przy pomocy szablonu **Blazor**, aczkolwiek jest rozwinęty o interakcje z utworzoną przez nas bazą danych. W tym przypadku aplikacja prezentuje się jako prosty system dodawania i oceniania filmów.

Ostatecznie utworzoną przez nas aplikację z bazą danych, należało wdrożyć w chmurze Azure.

Webapp

Początkowo utworzyliśmy projekt przy pomocy Blazor Web App oraz dobraliśmy odpowiednią konfigurację. Następnie dodaliśmy atrybut do odpowiedniej autentykacji: **builder . Services . AddAuthentication (**

CertificateAuthenticationDefaults . AuthenticationScheme).AddCertificate () ;

Następnie zmodyfikowaliśmy utworzoną przez template stronę **Weather**, która wyświetla dane o pogodzie. Modyfikacja polegała na dodaniu kolejnych 5 danych, a następnie posortowaniu ich po temperaturze większej niż 15°C. Dodatkowo dodaliśmy pasek wyszukiwania, do filtrowania po nazwie.

```
private void WarmDaysFilter()
{
    forecasts = originalForecasts
        ?.Where(f => f.TemperatureC > 15)
        .ToArray();
}

private void Restore()
{
    forecasts = originalForecasts?.ToArray();
}

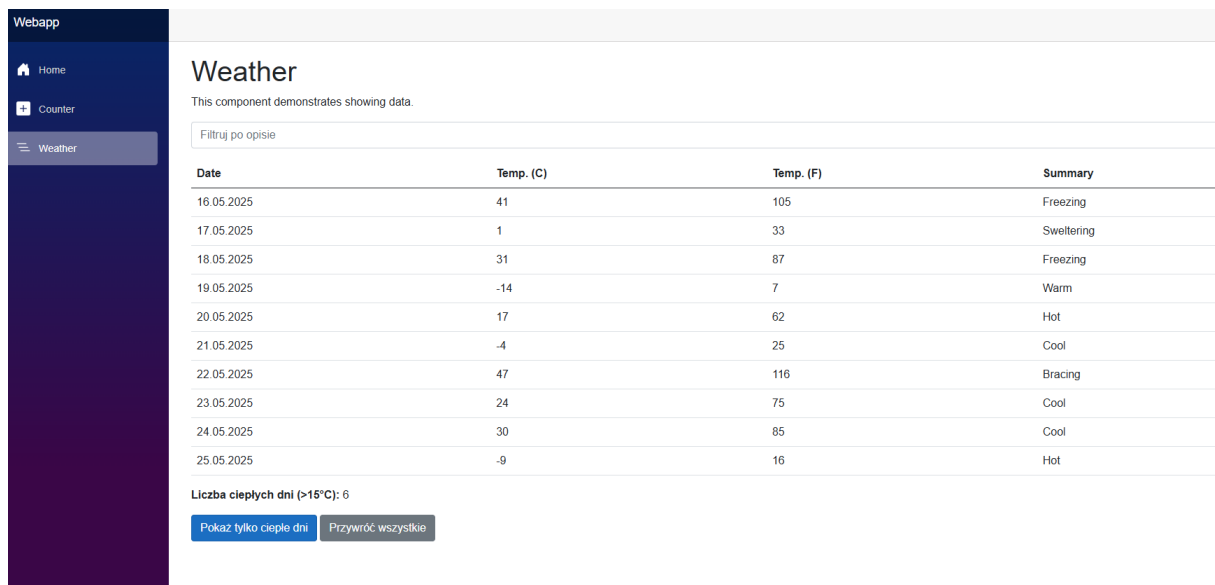
private string filterText = "";

private void Input(ChangeEventArgs e)
{
    filterText = e?.Value?.ToString() ?? "";
    forecasts = originalForecasts
        ?.Where(f => f.Summary != null && f.Summary.Contains(filterText, StringComparison.OrdinalIgnoreCase))
        .ToArray();
}

private class WeatherForecast
{
    public DateOnly Date { get; set; }
    public int TemperatureC { get; set; }
    public string? Summary { get; set; }
    public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);
}
```

Rysunek 1 Dodane zmiany

Rezultat ostateczny prezentuje się w taki sposób



The screenshot shows a web application with a dark blue sidebar containing navigation links: Home, Counter, and Weather. The main content area is titled "Weather" and includes a subtitle "This component demonstrates showing data." Below this is a search bar labeled "Filtruj po opisie". A table displays weather data for dates from 16.05.2025 to 25.05.2025, with columns for Date, Temp. (C), Temp. (F), and Summary. At the bottom, it states "Liczba ciepłych dni (>15°C): 6" and provides two buttons: "Pokaż tylko ciepłe dni" and "Przywróć wszystkie".

Date	Temp. (C)	Temp. (F)	Summary
16.05.2025	41	105	Freezing
17.05.2025	1	33	Sweltering
18.05.2025	31	87	Freezing
19.05.2025	-14	7	Warm
20.05.2025	17	62	Hot
21.05.2025	-4	25	Cool
22.05.2025	47	116	Bracing
23.05.2025	24	75	Cool
24.05.2025	30	85	Cool
25.05.2025	-9	16	Hot

Rysunek 2 Rezultat aplikacji Webapp

Web_database_app

Pracę nad tym etapem zaczęliśmy od utworzenia bazy danych przy użyciu EntityFramework. Nasz model danych miał za zadanie przetrzymywać informację o:

- tytule filmu
- opisu filmu
- dacie premiery
- ocenie filmu (zmieniającej się na bieżąco)
- zdjęciu/url zdjęcia dla filmu

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Web_database_app.Models
{
    [6 references]
    public class Movie
    {
        [11 references]
        public int Id { get; set; }
        [13 references]
        public string? Title { get; set; }
        [13 references]
        public string? Description { get; set; }

        [DataType(DataType.Date)]
        [13 references]
        public DateTime? ReleaseDate { get; set; }

        [3 references]
        public int VoteCount { get; set; } = 0;
        [2 references]
        public float TotalScore { get; set; } = 0;
        [NotMapped]
        [4 references]
        public float? Rate => VoteCount == 0 ? null : TotalScore / VoteCount;

        [12 references]
        public string? ImageUrl { get; set; }

        [6 references]
        public string? ImageFileName { get; set; }

        [NotMapped]
        [0 references]
        public IFormFile? UploadedImage { get; set; }
    }
}
```

Rysunek 3 Model danych

Dane opisane jako **[NotMapped]** służyły tylko do odczytu i nie znajdowały się w realnej bazie.

Następnie korzystając z gotowych narzędzi **Razor Components using Entity Framework (CRUD)**, wygenerowaliśmy podstrony:

Create, opisaliśmy całą funkcjonalność dodawania filmu

Edit, opcja edytowania wprowadzonych danych

Delete, usuwanie danych z rekordów

Details, podejrzanie detali filmu

Index, główna zawartość strony

Ważną rzeczą było dodanie na początku każdej strony opcji autentykacji oraz trybu odświeżania na bieżąco.

```
using Microsoft.AspNetCore.Authorization
@attribute [Authorize]
@page "/movies/edit"
@rendermode InteractiveServer
using Microsoft.EntityFrameworkCore
using Web_database_app.Models
@Inject IDbContextFactory<Web_database_app.Data.ApplicationDbContext> DbFactory
@Inject NavigationManager NavigationManager
```

Rysunek 4 Dodanie odpowiednich atrybutów

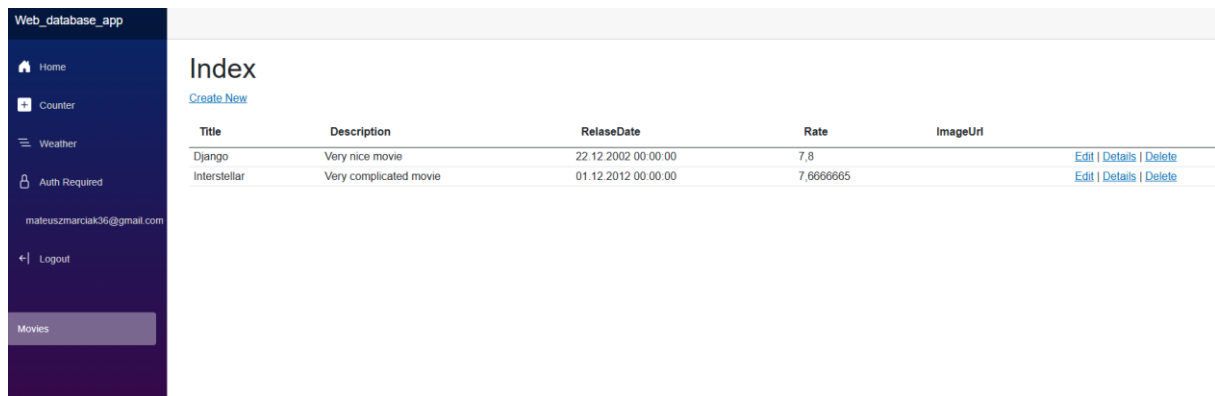
Dodatkowo wykonano możliwość logowania przy pomocy Google. Wykonanie tego kroku polegało na utworzeniu projektu w serwisie od **console.cloud.google.com**, który wygenerował nam specjalny token do logowania (unikalny dla jednego konta).

W głównym programie dodano odpowiednie komendy do autentykacji

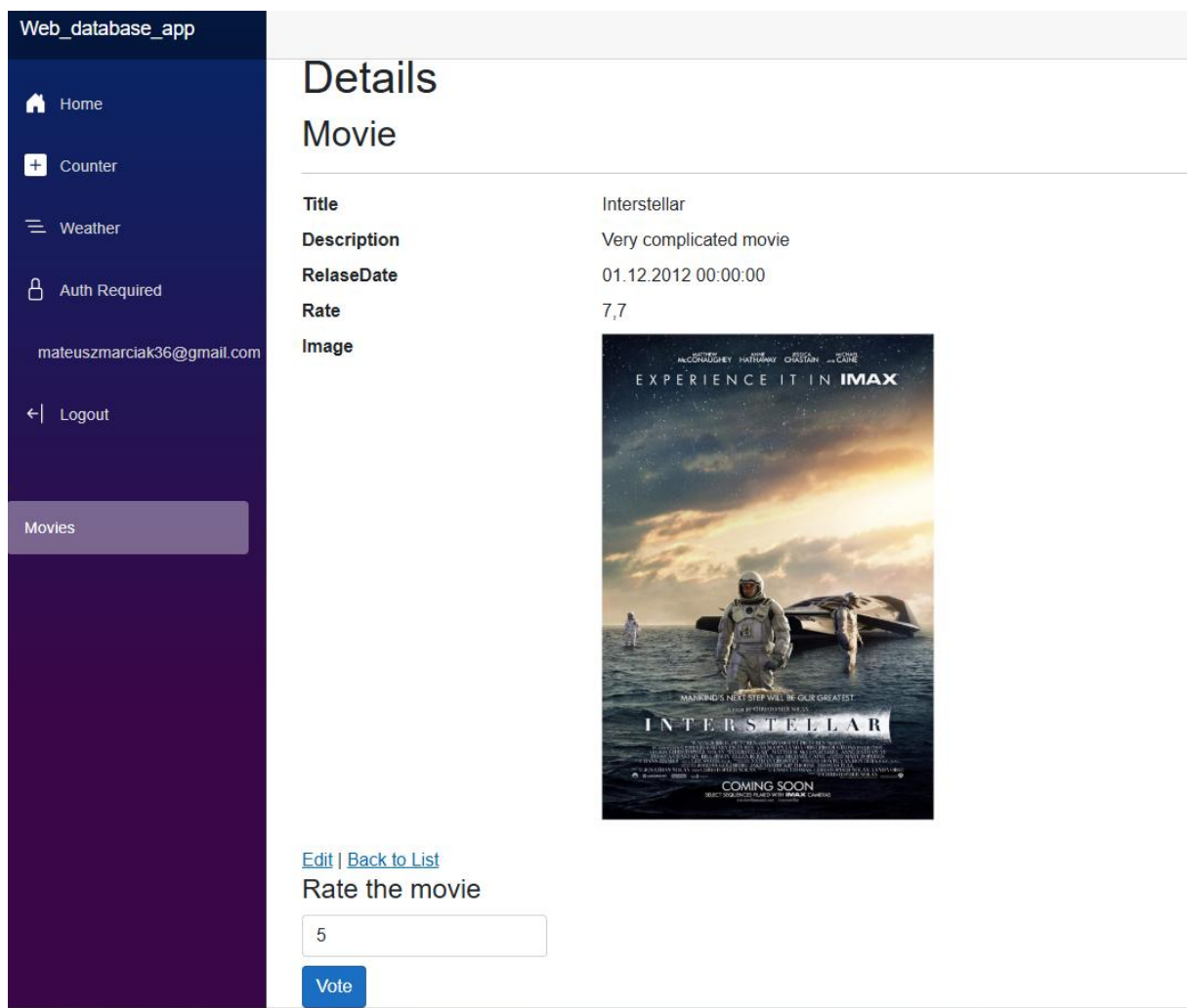
```
builder.Services.AddAuthentication()
    .AddGoogle(options =>
        builder.Configuration.Bind("Authentication:Google", options));
```

Rysunek 5 Logowanie przy pomocy Google

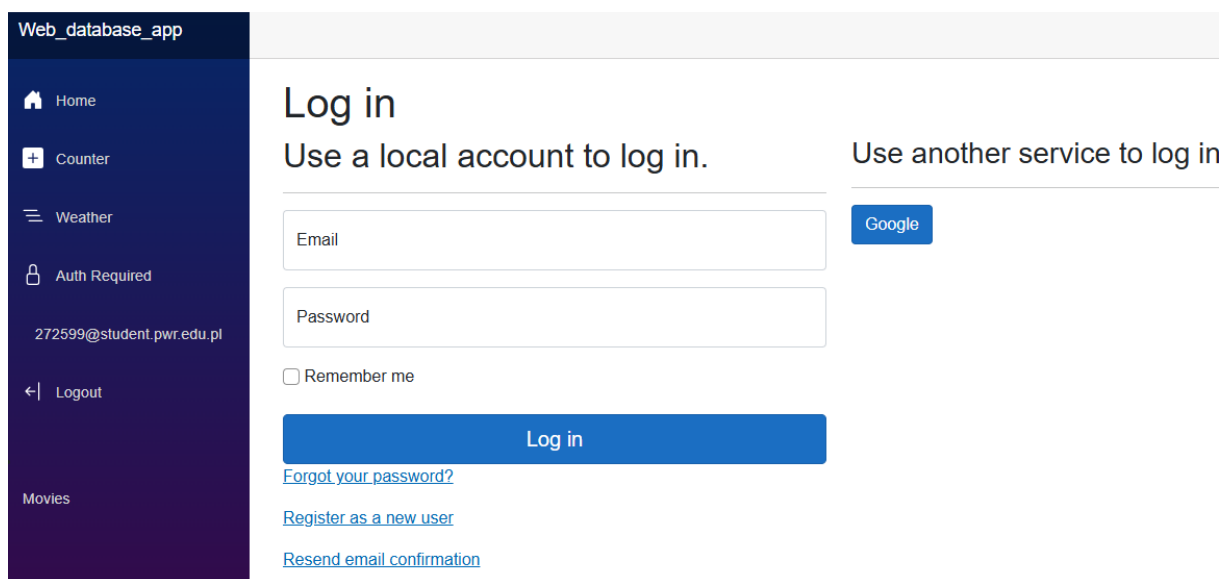
Ostateczne rezultaty prezentują się w taki sposób:



Rysunek 6 Główna strona



Rysunek 7 Szczegóły filmu



Web_database_app

Home

Counter

Weather

Auth Required

272599@student.pwr.edu.pl

Logout

Movies

Log in

Use a local account to log in.

Email

Password

☐ Remember me

Log in

[Forgot your password?](#)

[Register as a new user](#)

[Resend email confirmation](#)

Use another service to log in

Google

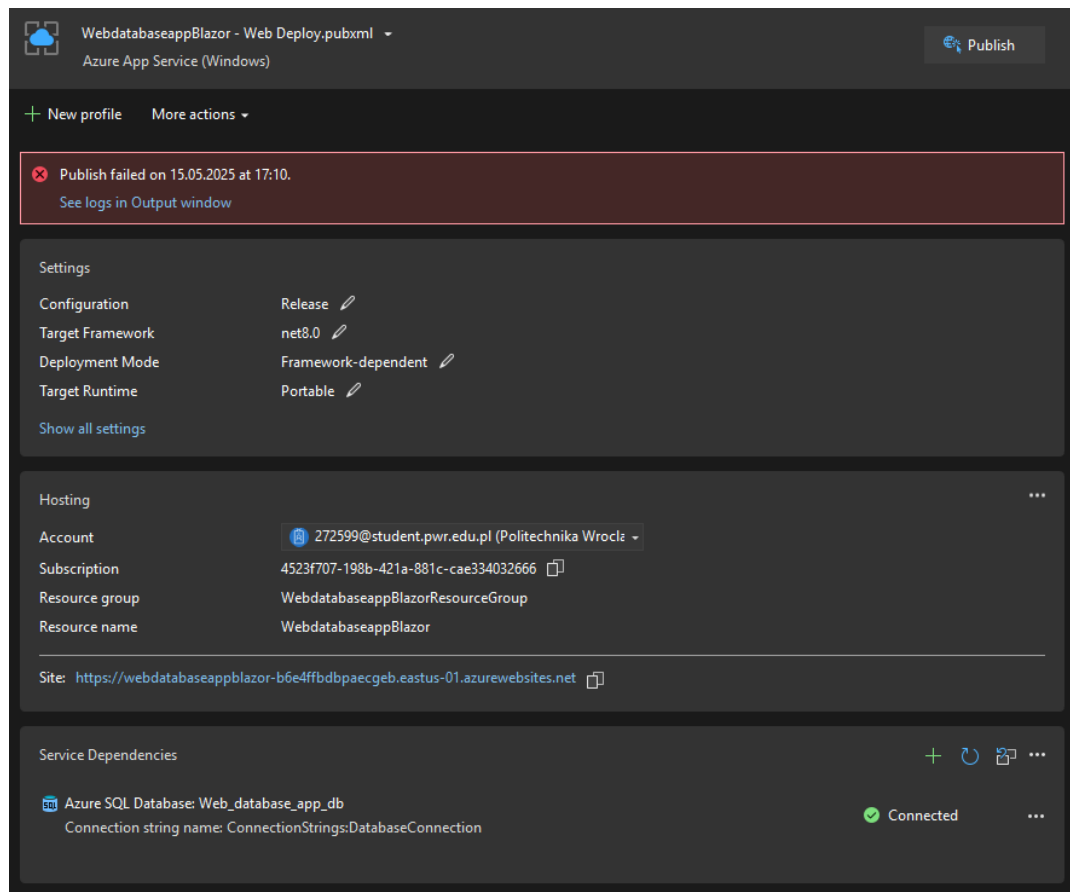
Rysunek 8 Opcja logowanie się przez Google

Publikowanie aplikacji w chmurze

Ostatecznie cały utworzony nasz projekt wraz z bazą danych należało wrzucić do chmury **Azure**. W tym celu utworzono konto na **Azure Portal** oraz podłączono projekt.

Niestety ze względu na zmiany sposobu autentykacji ten etap powiódł się częściowo, mianowicie udało się zrobić hosting dla aplikacji, aczkolwiek dla bazy danych już nie.

Poniżej przedstawiono rezultat konfiguracji publikowania projektu.



Rysunek 9 Konfiguracja publikacji