

# Platformy programistyczne .Net i Java

Projekt aplikacji bazodanowej

**Mateusz Marciak 272599**

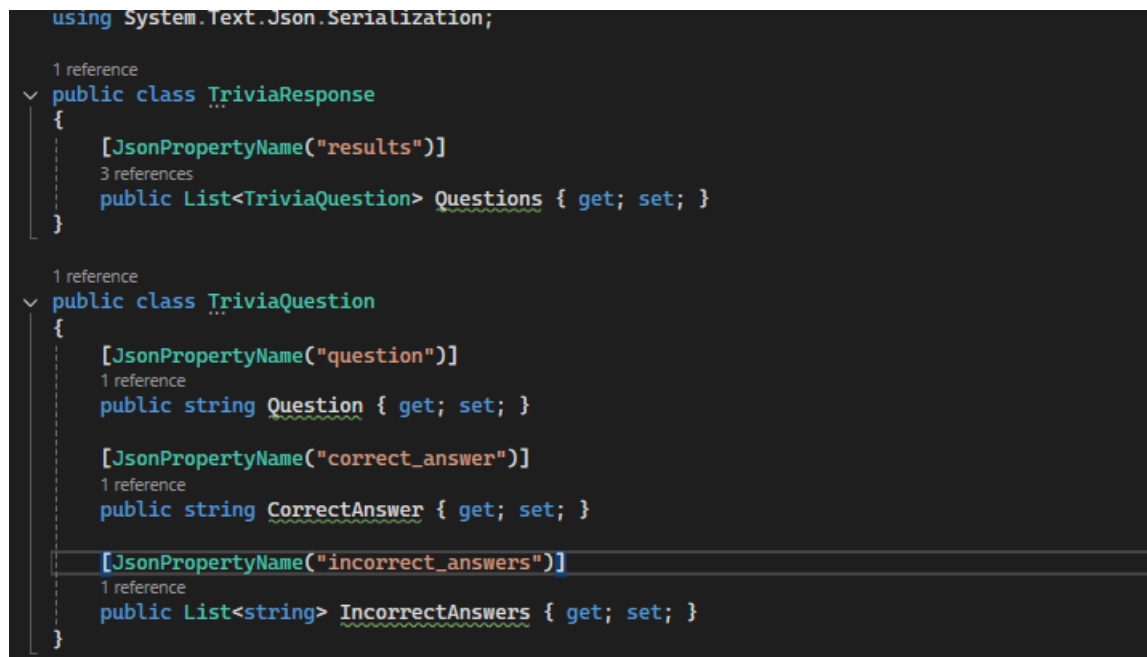
# Opis Projektu

Aplikacja w C# opiera się na połączeniu z API udostępnionego przez stronę Open Trivia Db, utworzeniu zapytania, pobrania pytań quizowych wraz z odpowiedziami, deserializacji i zapisaniu ich w bazie. Następnie utworzono aplikację GUI do przeglądania pytań oraz ich odpowiedzi. W projekcie użyto:

- platformę .NET 8.0
- Entity Framework 8.0.3
- MAUI.

## API oraz deserializacja

Deserializacja została przeprowadzona przy użyciu **System.Text.Json.Serialization**, a następnie podzielona na pytania, poprawne odpowiedzi oraz niepoprawne odpowiedzi.



```
using System.Text.Json.Serialization;

1 reference
public class TriviaResponse
{
    [JsonPropertyName("results")]
    3 references
    public List<TriviaQuestion> Questions { get; set; }
}

1 reference
public class TriviaQuestion
{
    [JsonPropertyName("question")]
    1 reference
    public string Question { get; set; }

    [JsonPropertyName("correct_answer")]
    1 reference
    public string CorrectAnswer { get; set; }

    [JsonPropertyName("incorrect_answers")]
    1 reference
    public List<string> IncorrectAnswers { get; set; }
}
```

Rysunek 1 Deserializacja

## Baza Danych

Następnym krokiem, było utworzenie bazy danych przy pomocy Entity Framework. W bazie znajdują się 2 tabele TriviaQuestions oraz Anwsers, połączone relacją jeden do wielu (jedno pytania może mieć wiele odpowiedzi, odpowiedzi mogą tylko do jednego pytania).

```

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

6 references
public class AnswerDb
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    0 references
    public int Id { get; set; }

    2 references
    public string AnswerText { get; set; }
    2 references
    public bool IsCorrect { get; set; }

    // Klucz obcy
    2 references
    public int TriviaQuestionDbId { get; set; }
    1 reference
    public TriviaQuestionDb TriviaQuestion { get; set; }
}

```

Rysunek 2 Tabela Answers

```

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

11 references
public class TriviaQuestionDb
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    1 reference
    public int Id { get; set; }

    2 references
    public string Question { get; set; }

    3 references
    public List<AnswerDb> Answers { get; set; } = new List<AnswerDb>();
}

```

Rysunek 3 Tabela TriviaQuestions

Relacja zdefiniowana została przy pomocy metody z użytego frameworku **modelBuilder**.

```

using Microsoft.EntityFrameworkCore;
using System.Runtime.CompilerServices;

[assembly: InternalsVisibleTo("Gui")]

6 references
internal class TriviaDbContext : DbContext
{
    4 references
    public DbSet<TriviaQuestionDb> Questions { get; set; }
    3 references
    public DbSet<AnswerDb> Answers { get; set; }

    0 references
    protected override void OnConfiguring(DbContextOptionsBuilder options)
    {
        options.UseSqlite("Data Source=trivia.db");
    }

    0 references
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        // Relacja 1 do wielu między TriviaQuestionDb a AnswerDb
        modelBuilder.Entity<AnswerDb>()
            .HasOne(a => a.TriviaQuestion)
            .WithMany(q => q.Answers)
            .HasForeignKey(a => a.TriviaQuestionDbId);
    }
}

```

Rysunek 4 Utworzenie relacji

## GUI

Ostatnim krokiem było utworzenie aplikacji GUI w MAUI (Multi-platform App UI). Celem tego etapu było udostępnienie funkcjonalności dla użytkownika takich jak:

- wyświetlenie rekordów znajdujących się w bazie danych,
- wprowadzenie parametrów API, pozwalających na pobranie interesujących danych,
- ręczne usunięcie rekordu do bazy danych

Utworzenie interfejsu opierało się na rozmieszczeniu layout w języku XAML oraz odpowiedniego zdefiniowania funkcjonalności w C#.

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Gui.MainPage">

    <StackLayout Padding="10" Spacing="10">

        <!-- Lista pytań -->
        <ListView x:Name="QuestionsListView"
                 ItemSelected="OnQuestionSelected">
            <ListView.ItemTemplate>
                <DataTemplate>
                    <ViewCell>
                        <StackLayout Orientation="Horizontal" Padding="5">
                            <Label Text="{Binding Question}"
                                   VerticalOptions="CenterAndExpand"
                                   FontSize="16"
                                   HorizontalOptions="StartAndExpand"/>
                            <Label Text=">"
                                   FontSize="16"
                                   VerticalOptions="Center"
                                   TextColor="Gray"/>
                        </StackLayout>
                    </ViewCell>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>

        <Label x:Name="StatusLabel" Text="Kliknij aby pobrać pytania" />
        <!-- Przyciski -->
        <Button Text="Pobierz pytania z API"
                Clicked="FetchQuestionsFromApi"/>

        <Button Text="Wyczyść bazę"
                Clicked="ClearDatabase"
                BackgroundColor="Red"
                TextColor="White"/>
    </StackLayout>
</ContentPage>
```

Rysunek 5 Layout strony głównej

Od strony backendowej musieliśmy przekazać bazę danych utworzoną wcześniej, sposób deserializacji, utworzyć funkcjonalność odświeżania oraz obsługi umieszczonych przycisków.

```

public partial class MainPage : ContentPage
{
    private readonly TriviaDbContext _dbContext;
    4 references
    public ObservableCollection<TriviaQuestionDb> Questions { get; set; } = new ObservableCollection<TriviaQuestionDb>();

    0 references
    public MainPage()
    {
        InitializeComponent();
        _dbContext = new TriviaDbContext();
        LoadQuestions();
        BindingContext = this;
    }

    2 references
    private void LoadQuestions()
    {
        Questions.Clear();
        var questionsFromDb = _dbContext.Questions.Include(q => q.Answers).ToList();
        foreach (var question in questionsFromDb)
        {
            Questions.Add(question);
        }
        QuestionsListView.ItemsSource = Questions;
    }
}

```

Rysunek 6 Fragment funkcjonalności w C#

Powyżej przedstawiona metoda **LoadQuestion()** służy do ciągłego odświeżania rekordów z bazy.

```

0 references
private async void FetchQuestionsFromApi(object sender, EventArgs e)
{
    StatusLabel.Text = "Pobieranie danych...";
    var newQuestions = await _FetchFromApi();
    if (newQuestions.Any())
    {
        _dbContext.Questions.AddRange(newQuestions);
        await _dbContext.SaveChangesAsync();
        StatusLabel.Text = "Dane pobrane! Kliknij ponownie aby pobrać więcej";

        LoadQuestions();
    }
}

```

Rysunek 7 Obsługa pobrania pytań z API

```

0 references
private async void ClearDatabase(object sender, EventArgs e)
{
    _dbContext.Answers.RemoveRange(_dbContext.Answers);
    _dbContext.Questions.RemoveRange(_dbContext.Questions);
    await _dbContext.SaveChangesAsync();

    Questions.Clear();
}

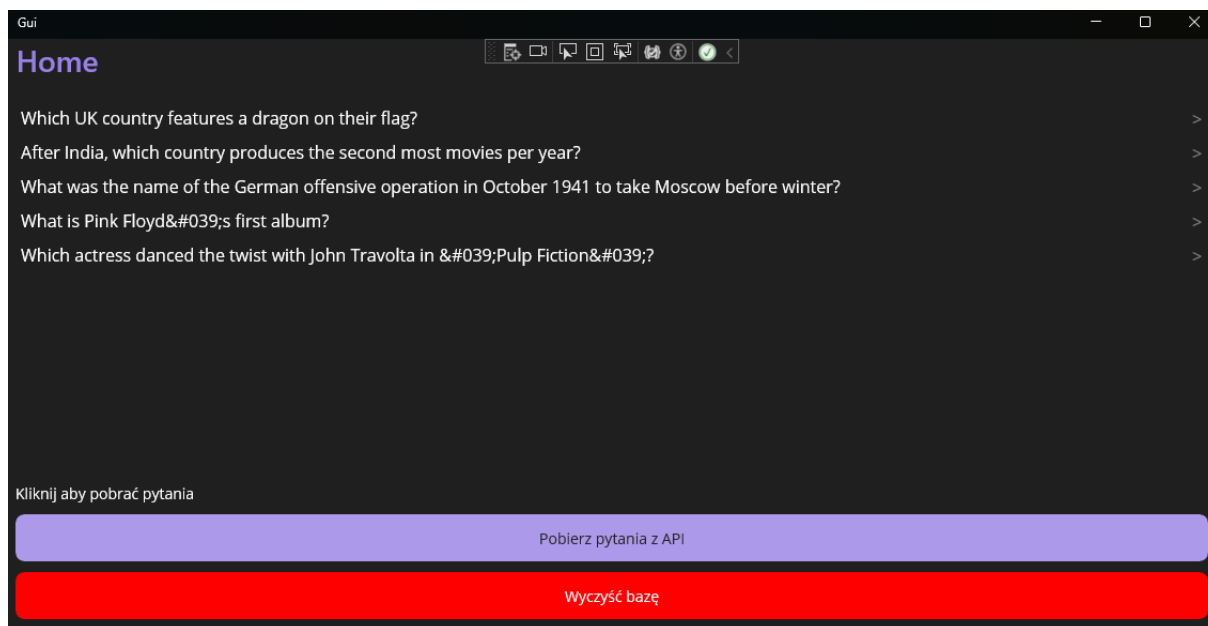
```

Rysunek 8 Obsługa czyszczenia bazy

Przedstawione na Rysunku 7 i 8 metody są wywoływane po naciśnięciu przycisków.

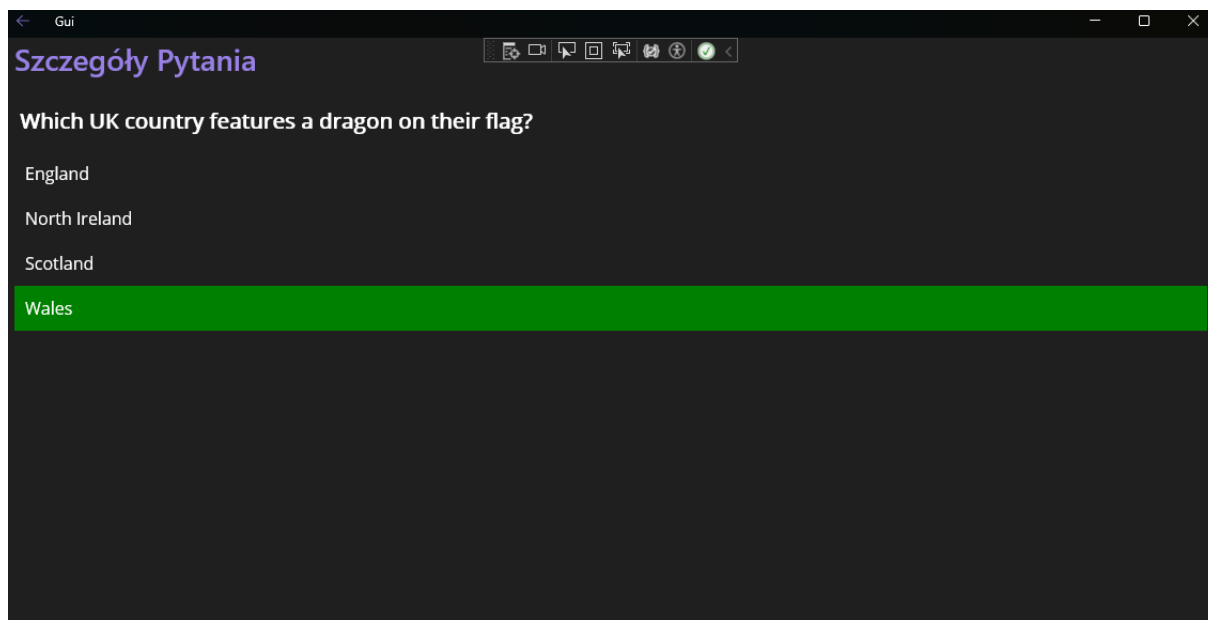
# Rezultat

Ostateczny rezultat wygląda w następujący sposób:



Rysunek 9 Utworzona aplikacja okienkowa

W celu rozwinięcia pytania oraz zobaczenia odpowiedzi należało kliknąć na wybrane pytanie.



Rysunek 10 Rozwinięcie pytania