Wyższa Szkoła Bankowa w Poznaniu
Wydział Finansów i Bankowości
Studia stacjonarne I stopnia – Informatyka

Dokumentacja Techniczna Projektu

strony internetowej wydarzeń kulturalnych

Przedmiot: **Programowanie zaawansowane**

Grupa: K-17

Prowadzący: dr Paweł PŁACZEK

Zespół projektowy:

Student: Paweł KOSTECKI

Student: Mateusz TRUSIAK

1. Informacja ogólna o projekcie.

Przedmiotem niniejszego projektu jest zaprojektowanie i wykonanie aplikacji internetowej ASP.NET w architekturze klient-serwer. Implementacja kodu odbywa się w oparciu o język C# i platformę .NET Framework wersja 6.0. Aplikacja zapewnia użytkownikowi dostęp do informacji o wydarzeniach kulturalnych w Polsce, takich jak koncerty, festiwale, zakup biletów oraz założenie konta.

2. Elementy aplikacji.

2.1. Belka nawigacyjna – składa się z następujących elementów:

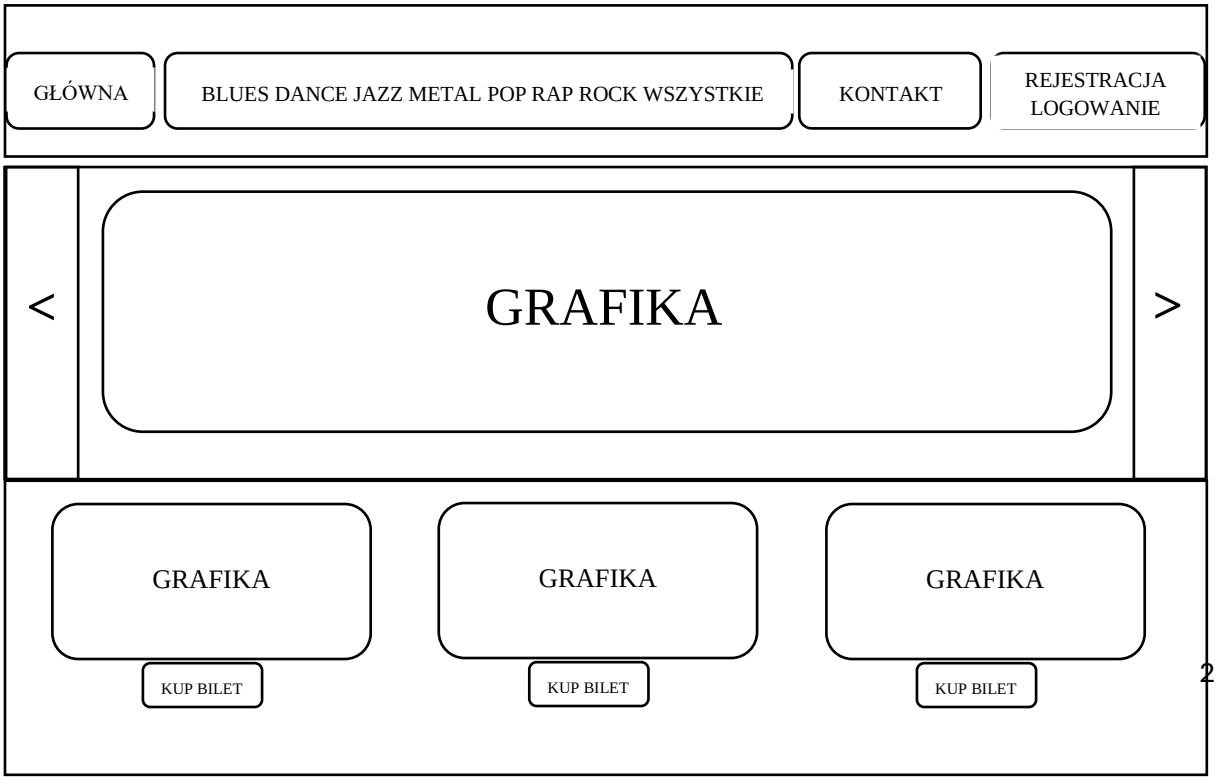
2.1.1. GŁÓWNA – przycisk kieruje na stronę główną;
2.1.2. GATUNKI MUZYCZNE - pod każdym przyciskiem znajdują się wydarzenia z konkretnym tagiem;
2.1.3. KONTAKT – odsyła do formularza kontaktowego;
2.1.4. REJESTRACJA/LOGOWANIE – odsyła do podstrony z opcjami: ZALOGUJ SIĘ i ZAREJESTRUJ SIĘ. Zalogowany użytkownik będzie mógł złożyć zamówienie, dla użytkownika niezalogowanego opcja ta jest niedostępna.

2.2. Suwak – miejsce przeznaczone do promowania najważniejszych wydarzeń. Użytkownik ma możliwość przechodzenia pomiędzy slajdami używając strzałek znajdujących się z lewej oraz prawej strony wyświetlanego zdjęcia.

2.3. Wyszukiwarka – umożliwi użytkownikowi wyszukanie wydarzenia po nazwie artysty, miejscu wydarzenia (np. klub), mieście.

2.4. Sekcje – każda z poszczególnych sekcji wyświetla maksymalnie trzy wydarzenia, w równych odstępach od siebie. Użytkownik zainteresowany kupnem biletu zostanie przekierowany do strony wybranego wydarzenia, gdzie będzie mógł dokonać wyboru ilości biletów oraz finalizacji zakupu.

3. Wizualizacja układu strony:



4. Opis ogólny

Interfejs graficzny niniejszej aplikacji internetowej został podzielony tak aby każdy z elementów umożliwił użytkownikowi wykonanie innej „akcji".

Na samej górze strony umiejscowiono belkę nawigacyjną, która ułatwi poruszanie się oraz wybór sekcji będącej przedmiotem zainteresowania osoby odwiedzającej. Każdy z przycisków przeniesie użytkownika w odpowiednie miejsce. Na przykład użytkownik po wybraniu i naciśnięciu na „METAL" będzie mógł dokonać wyboru wydarzenia na podstawie dostępnych w danym momencie wydarzeń – do bazy danych dodane są trzy wydarzenia tej kategorii. Taką samą funkcjonalność będą posiadały pozostałe przyciski. Strona kontaktowa wyświetli się po wybraniu „KONTAKT". Przycisk „REJESTRACJA" przekieruje użytkownika do strony rejestracji, gdzie należy podać adres email oraz hasło. Po rejestracji należy potwierdzić założenie konta klikając w link wyświetlający się na stronie. Tylko zarejestrowany i zalogowany użytkownik będzie mógł dokonać zakupu biletu.

Poniżej belki nawigacyjnej umieszczony zostanie suwak wyświetlający wydarzenia wyróżnione przez właściciela strony. Zdjęcia zostaną umieszczone „w karuzeli", co umożliwi wyświetlanie ich w pętli i przesuwanie dedykowanymi przyciskami w lewo bądź prawo, w zależności jaką akcję podejmie użytkownik.

Sekcja wydarzeń ma charakter informacyjny oraz zachęcający użytkownika do zakupu biletów. Wyświetlone zostaną tu zdjęcia promocyjne, data, miejsce, a także krótki opis i opcja zakupu. Użytkownik po wybraniu oraz kliknięciu w odnośnik zostanie przeniesiony do podstrony dotyczącej konkretnego wydarzenia.

5. Użytkownicy aplikacji

5.1.Użytkownik indywidualny

Poszukuje informacji o wydarzeniach i kupuje bilety online.

Przypadki użycia:

- Rejestracja - Użytkownik może założyć nowe konto, aby uzyskać dostęp do zaawansowanych funkcji;
- Logowanie - Użytkownik loguje się na swoje konto, aby zarządzać swoimi danymi i zakupami;
- Wyszukiwanie wydarzeń - Użytkownik może wyszukiwać wydarzenia według różnych kryteriów;
- Przeglądanie szczegółów wydarzeń - Użytkownik może zobaczyć szczegółowe informacje o wydarzeniach;
- Zakup biletów - Użytkownik może kupować bilety na wydarzenia;
- Kontakt z organizatorem - Użytkownik może wysłać zapytanie do organizatora wydarzenia.

6. Specyfikacja wykorzystanych technologii

Projekt "Aplikacja Koncertowa" został zrealizowany przy użyciu następujących technologii:

6.1.Języki programowania:

- SCSS – odpowiedzialny za takie funkcje jak zmienne, zagnieżdżanie selektorów. Według GitHub zajmuje 47.4% całkowitego kodu projektu, co świadczy o intensywnym wykorzystaniu jego możliwości w stylizacji.
- JavaScript – jest językiem skryptowym stosowanym do tworzenia dynamicznych i interaktywnych elementów na stronach internetowych. Udział tego języka wynosi 34.9%.
- C# – język programowania stworzony przez Microsoft. Udział w projekcie wynosi 14.2% w kodzie projektu. Odpowiada za implementację logiki biznesowej i części back-endowej aplikacji.
- HTML – podstawowy język używany do prezentacji treści na stronach internetowych. Jego udział w projekcie to 3.5% według statystyki GitHub.

6.2. Platforma:

- .NET Framework 6.0 – podstawowa platforma, na której oparta jest aplikacja. Zapewnia wsparcie dla budowy aplikacji oraz dostęp do bazy.

6.3. Framework:

- ASP.NET Core MVC: Framework MVC (Model-View-Controller) użyty do zarządzania strukturą aplikacji, obsługi żądań HTTP, renderowania widoków oraz kontrolowania logiki aplikacji.

6.4. Baza danych:

- MS SQL Server: Relacyjny system zarządzania bazą danych używany do przechowywania danych aplikacji, takich jak informacje o koncertach, użytkownikach, biletach, itp. Używany w połączeniu z Entity Framework Core.

6.5. ORM (Object-Relational Mapping):

- Entity Framework Core: ORM użyty do mapowania obiektów .NET na tabele bazy danych. Zapewnia prostotę w zarządzaniu danymi oraz migracje bazy danych.

6.6. Zarządzanie użytkownikami i uwierzytelnianie:

- ASP.NET Identity: Framework użyty do zarządzania użytkownikami, uwierzytelniania oraz autoryzacji. Umożliwia łatwe tworzenie kont użytkowników, logowanie, zarządzanie hasłami, role i wiele innych.

6.7. Frontend:

- HTML5: Używany do strukturyzowania treści na stronie internetowej.
- CSS3: Używany do stylizacji strony, zapewniając atrakcyjny wygląd.
- Bootstrap: Framework CSS użyty do responsywnego projektowania i stylizacji interfejsu użytkownika. Umożliwia szybkie tworzenie atrakcyjnych i responsywnych stron internetowych.
- JavaScript: Używany do interaktywności na stronie oraz manipulacji elementami DOM.

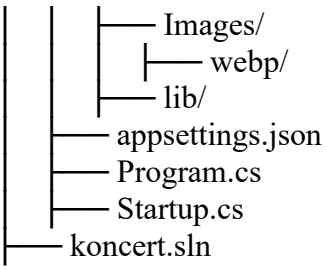
6.8. System kontroli wersji:

- GitHub – platforma do hostowania kodu źródłowego, współpracy z zespołem oraz zarządzania wersjami projektu.

7. Opis struktury projektu

Struktura projektu przedstawia się następująco:

```
Aplikacja_Koncert/
├── koncert/
│   ├── Areas/
│   │   ├── Identity/
│   │   │   ├── Pages/
│   │   │   │   ├── Account/
│   │   │   │   │   ├── Register.cshtml
│   │   │   │   │   ├── Login.cshtml
│   ├── Controllers/
│   │   ├── ConcertController.cs
│   │   ├── CategoryController.cs
│   ├── Models/
│   │   ├── Entities/
│   │   │   ├── Concert.cs
│   │   │   ├── Category.cs
│   │   │   ├── Description.cs
│   │   │   ├── NoteOneDescription.cs
│   │   │   ├── NoteTwoSell.cs
│   │   │   ├── NoteThreeOther.cs
│   │   │   ├── Content.cs
│   │   │   ├── Chapter.cs
│   │   │   ├── Place.cs
│   │   ├── Repositories/
│   │   │   ├── IConcertRepository.cs
│   │   │   ├── ICategoryRepository.cs
│   │   ├── koncertDbContext.cs
│   ├── Properties/
│   ├── TagHelpers/
│   ├── ViewModels/
│   ├── Views/
│   │   ├── Concert/
│   │   │   ├── List.cshtml
│   │   │   ├── Details.cshtml
│   │   ├── Contact/
│   │   ├── Home/
│   │   │   ├── Index.cshtml
│   │   ├── Order/
│   │   ├── Shared/
│   │   │   ├── _Layout.cshtml
│   │   │   ├── _LoginPartial.cshtml
│   │   ├── ShoppingCart/
│   ├── wwwroot/
```



8. Kluczowe elementy:

8.1.Areas/Identity/Pages/Account:
- Zawiera strony odpowiedzialne za rejestrację (Register.cshtml) i logowanie (Login.cshtml).

8.2.Controllers:
- ConcertController.cs: Obsługuje żądania związane z koncertami, takie jak wyświetlanie listy koncertów i szczegółów konkretnego koncertu.
- CategoryController.cs: Obsługuje żądania związane z kategoriami koncertów, takie jak wyświetlanie listy kategorii.

8.3.Models/Entities:
- Concert.cs: Model danych reprezentujący wydarzenie koncertowe.
- Category.cs: Model danych reprezentujący kategorię koncertów.
- Description.cs: Model danych reprezentujący szczegółowy opis koncertu.
- NoteOneDescription.cs: Model danych dla dodatkowych informacji o koncercie.
- NoteTwoSell.cs: Model danych dla informacji o sprzedaży biletów.
- NoteThreeOther.cs: Model danych dla innych informacji związanych z koncertem.
- Content.cs: Model danych dla zawartości koncertu.
- Chapter.cs: Model danych dla rozdziałów zawartości koncertu.
- Place.cs: Model danych dla miejsc związanych z koncertem.

8.4.Models/Repositories:
- IConcertRepository.cs: Interfejs repozytorium dla modelu Concert.
- ICategoryRepository.cs: Interfejs repozytorium dla modelu Category.
- koncertDbContext.cs:
- Klasa kontekstu bazy danych używana do komunikacji z bazą danych za pomocą Entity Framework Core.

8.5.Views:
- Concert: Zawiera widoki związane z koncertami (List.cshtml, Details.cshtml).
- Home: Zawiera stronę główną (Index.cshtml).
- Shared: Zawiera wspólne widoki i komponenty (_Layout.cshtml, _LoginPartial.cshtml).

8.6.wwwroot:
- Images: Zawiera obrazy używane w aplikacji, takie jak miniatury koncertów.
- lib: Zawiera biblioteki JavaScript i CSS używane w aplikacji.

8.7.appsettings.json:
- Plik konfiguracyjny zawierający ustawienia aplikacji, takie jak connection string do bazy danych.

8.8.Program.cs:
- Klasa główna aplikacji odpowiedzialna za uruchomienie aplikacji.

8.9.koncert.sln:
- Plik rozwiązania Visual Studio, który zawiera informacje o projekcie i jego strukturze.

9. Kontrolery wraz z metodami

Kontrolery znajdują się w folderze Controllers i obsługują żądania HTTP, przetwarzają dane oraz zwracają odpowiedzi do widoków. Każdy kontroler jest odpowiedzialny za różne funkcje aplikacji. Poniżej znajduje się opis kontrolerów wraz z metodami:

9.1. ConcertController

Kontroler obsługujący żądania związane z koncertami.

```
using koncert.Models.Repositories;

using Microsoft.AspNetCore.Mvc;

namespace koncert.Controllers
{
   public class ConcertController : Controller
   {
      private readonly IConcertRepository _concertRepository;
      private readonly ICategoryRepository _categoryRepository;

      public ConcertController(IConcertRepository concertRepository, ICategoryRepository
categoryRepository)
      {
         _concertRepository = concertRepository;
         _categoryRepository = categoryRepository;
      }

      // Wyświetla listę wszystkich koncertów
      public IActionResult List()
      {
         var concerts = _concertRepository.AllConcerts;
         return View(concerts);
      }

      // Wyświetla szczegóły konkretnego koncertu
      public IActionResult Details(int id)
      {
         var concert = _concertRepository.GetConcertById(id);
         if (concert == null)
            return NotFound();

         return View(concert);
      }

      // Wyświetla listę koncertów z wybranej kategorii
      public IActionResult ByCategory(string category)
      {
         var concerts = _concertRepository.GetConcertsByCategory(category);
         return View(concerts);
```

```
        }

        // Metoda dodawania nowego koncertu (dla organizatora)
        [HttpPost]
        public IActionResult AddConcert(Concert concert)
        {
            if (ModelState.IsValid)
            {
                _concertRepository.AddConcert(concert);
                return RedirectToAction("List");
            }
            return View(concert);
        }

        // Metoda edycji koncertu (dla organizatora)
        [HttpPost]
        public IActionResult EditConcert(Concert concert)
        {
            if (ModelState.IsValid)
            {
                _concertRepository.UpdateConcert(concert);
                return RedirectToAction("Details", new { id = concert.ConcertId });
            }
            return View(concert);
        }
    }
}
```

9.2. CategoryController

Kontroler obsługujący żądania związane z kategoriami koncertów.

```
using koncert.Models.Repositories;
using Microsoft.AspNetCore.Mvc;

namespace koncert.Controllers
{
    public class CategoryController : Controller
    {
        private readonly ICategoryRepository _categoryRepository;

        public CategoryController(ICategoryRepository categoryRepository)
        {
            _categoryRepository = categoryRepository;
        }

        // Wyświetla listę wszystkich kategorii koncertów
        public IActionResult List()
        {
            var categories = _categoryRepository.AllCategories;
```

```
            return View(categories);
        }
    }
}
```

9.3. AccountController

Kontroler obsługujący żądania związane z kontem użytkownika, takie jak rejestracja, logowanie i zarządzanie profilem.

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using koncert.Models.ViewModels;

namespace koncert.Controllers
{
    public class AccountController : Controller
    {
        private readonly UserManager<IdentityUser> _userManager;
        private readonly SignInManager<IdentityUser> _signInManager;

        public AccountController(UserManager<IdentityUser> userManager,
SignInManager<IdentityUser> signInManager)
        {
            _userManager = userManager;
            _signInManager = signInManager;
        }

        // Wyświetla formularz rejestracji
        [HttpGet]
        public IActionResult Register()
        {
            return View();
        }

        // Obsługuje przesłanie formularza rejestracji
        [HttpPost]
        public async Task<IActionResult> Register(RegisterViewModel model)
        {
            if (ModelState.IsValid)
            {
                var user = new IdentityUser { UserName = model.Email, Email = model.Email };
                var result = await _userManager.CreateAsync(user, model.Password);

                if (result.Succeeded)
                {
                    await _signInManager.SignInAsync(user, isPersistent: false);
                    return RedirectToAction("Index", "Home");
                }
```

```
            foreach (var error in result.Errors)
            {
                ModelState.AddModelError(string.Empty, error.Description);
            }
        }

        return View(model);
    }

    // Wyświetla formularz logowania
    [HttpGet]
    public IActionResult Login()
    {
        return View();
    }

    // Obsługuje przesłanie formularza logowania
    [HttpPost]
    public async Task<IActionResult> Login(LoginViewModel model)
    {
        if (ModelState.IsValid)
        {
            var result = await _signInManager.PasswordSignInAsync(model.Email,
model.Password, model.RememberMe, lockoutOnFailure: false);

            if (result.Succeeded)
            {
                return RedirectToAction("Index", "Home");
            }

            ModelState.AddModelError(string.Empty, "Invalid login attempt.");
        }

        return View(model);
    }

    // Wylogowuje użytkownika
    [HttpPost]
    public async Task<IActionResult> Logout()
    {
        await _signInManager.SignOutAsync();
        return RedirectToAction("Index", "Home");
    }
  }
}
```

9.4. HomeController

Kontroler obsługujący żądania dotyczące strony głównej.

```
using Microsoft.AspNetCore.Mvc;

namespace koncert.Controllers
{
   public class HomeController : Controller
   {
      // Wyświetla stronę główną
      public IActionResult Index()
      {
         return View();
      }

      // Wyświetla stronę kontaktową
      public IActionResult Contact()
      {
         return View();
      }
   }
}
```

9.5. OrderController

Kontroler obsługujący żądania związane z zamówieniami biletów.

```
using koncert.Models.Repositories;
using Microsoft.AspNetCore.Mvc;

namespace koncert.Controllers
{
   public class OrderController : Controller
   {
      private readonly IOrderRepository _orderRepository;

      public OrderController(IOrderRepository orderRepository)
      {
         _orderRepository = orderRepository;
      }

      // Wyświetla formularz zamówienia
      public IActionResult Checkout()
      {
         return View();
      }

      // Obsługuje przesłanie formularza zamówienia
      [HttpPost]
      public IActionResult Checkout(Order order)
      {
         if (ModelState.IsValid)
         {
```

```
            _orderRepository.CreateOrder(order);
            return RedirectToAction("CheckoutComplete");
        }

        return View(order);
    }

    // Wyświetla stronę potwierdzenia zamówienia
    public IActionResult CheckoutComplete()
    {
        return View();
    }
  }
}
```

10. Modele danych

Modele danych znajdują się w folderze Models/Entities i reprezentują strukturę danych aplikacji.

10.1. Koncert – reprezentuje wydarzenie koncertowe.

```
public class Concert

{

   public int ConcertId { get; set; }

   public string Name { get; set; } = string.Empty;

   public string Artist { get; set; } = string.Empty;

   public decimal Price { get; set; }

   public string? ImageUrl { get; set; }

   public string? ImageThumbnailUrl { get; set; }

   public bool IsConcertOfTheMonth { get; set; }

   public bool IsRecommended { get; set; }

   public Description Description { get; set; } = default!;

   public ICollection<Content> ConcertContent { get; set; } = new List<Content>();

   public ICollection<Faq>? Faq { get; set; }

   public ICollection<Announcement>? Announcements { get; set; }
```

```
    public ICollection<Opinion>? Opinions { get; set; }

    public int CategoryId { get; set; }

    public Category Category { get; set; } = default!;

}
```

Podczas tworzenia tej klasy zdefiniowano identyfikator kursu (ConcertId), jego wyświetlaną nazwę (Name) oraz artystę (Artist) i cenę biletów (Price). Zawarto również informację o tym czy dany koncert jest wydarzeniem miesiąca czy polecanym przez innych użytkowników strony. Kolejnym elementem jest opis. Dodatkowo można odszukać deklarację adresu obrazka wydarzenia oraz jego miniatury, wyświetlanych użytkownikowi.

10.2. Kategorie:

```
namespace KONCERT.Models.Entities
{
    public class Category
    {
        public int CategoryId { get; set; }
        public string Name { get; set; } = string.Empty;
        public ICollection<Concert>? Concerts { get; set; }
    }
}
```

Klasa ta zawiera identyfikator kategorii, następnie jej nazwę oraz listę koncertów.

10.3. Opis wydarzenia:

```
namespace KONCERT.Models.Entities
{
    public class Description
    {
        public int DescriptionId { get; set; }
        public string Name { get; set; } = string.Empty;
        public ICollection<NoteOneDescription>? NoteOneDescription { get; set; }
        public ICollection<NoteTwoSell>? NoteTwoSell { get; set; }
        public ICollection<NoteThreeOther>? NoteThreeOther { get; set; }
        public string Content { get; set; } = string.Empty;

        public int ConcertId { get; set; }

    }
}
```

Podczas implementacji tej klasy poza numerem Id oraz nazwy Name wydarzenia zdefiniowano trzy kolekcje:

- NoteOneDescription – dla informacji odnośnie wydarzenia;
- NoteTwoSell – dla informacji związanych ze sprzedażą;

- NoteThreeOther – dla innych informacji związanych z wydarzeniem;

Zauważyć można również umieszczenie właściwości nawigacyjnej (ConcertId), która łączy utworzony opis z konkretnym wydarzeniem.

10.4. NoteOneDescription:

```
namespace KONCERT.Models.Entities
{
   public class NoteOneDescription
   {
      public int NoteOneDescriptionId { get; set; }
      public string Content { get; set; } = string.Empty;
      public int DescriptionId { get; set; }

   }
}
```

Klasa przygotowana dla opisu. Połączona z konkretnym opisem wydarzenia.

10.5. NoteTwoSell:

```
namespace KONCERT.Models.Entities
{
   public class NoteTwoSell
   {
      public int NoteTwoSellId { get; set; }
      public string Content { get; set; } = string.Empty;
      public int DescriptionId { get; set; }
   }
}
```

Klasa przygotowana dla opisu dotyczącego sprzedaży biletów. Połączona z konkretnym opisem wydarzenia.

10.6. NoteThreeOther:

```
namespace KONCERT.Models.Entities
{
   public class NoteThreeOther
   {
      public int NoteThreeOtherId { get; set; }
      public string Content { get; set; } = string.Empty;
      public int DescriptionId { get; set; }
   }
}
```

Klasa przygotowana dla umieszczenia innych informacji związanych z koncertem. Połączona z konkretnym opisem wydarzenia.

10.7. Content:

```
namespace KONCERT.Models.Entities
{
```

```
    public class Content
    {
        public int ContentId { get; set; }
        public ICollection<Chapter> chapters { get; set; }
        public int ConcertId { get; set; }

    }
}
```

10.8. Chapter:

```
namespace KONCERT.Models.Entities
{
    public class Chapter
    {
        public int ChapterId { get; set; }
        public string Name { get; set; } = string.Empty;
        public ICollection<Place> Places { get; set; } = new List<Place>();
        public int ConcertId { get; set; }     }
}
```

10.9. Place:

```
namespace koncert.Properties.Models.Entities
{
    public class Place
    {
        public int PlaceId { get; set; }
        public string Name { get; set; } = string.Empty;
        public string Duration { get; set; } = string.Empty;
        public int ChapterId { get; set; }

    }
}
```

11. Repozytoria

11.1. Koncert:

```
using koncert.Properties.Models.Entities;

namespace koncert.Properties.Models.Repositories
{
    public interface IConcertRepository
    {
        IEnumerable<Concert> AllConcerts { get; } //wszystkie koncerty jakie mamy
        IEnumerable<Concert> ConcertOfTheMonths { get; } //wszystkie koncerty miesiąca
        IEnumerable<Concert> Recommended {  get; } //wszystkie polecane koncerty
        Concert? GetConcertById(int concertid);
    }
}
```

Repozytorium do obsługi modelu koncert.

11.2. Kategorie:

```
using koncert.Properties.Models.Entities;

namespace koncert.Properties.Models.Repositories
{
   public interface ICategoryRepository
   {
      IEnumerable<Category> AllCategories { get; }
   }
}
```

Repozytorium do obsługi modelu kategorie koncertów.

12. Kontrolery

12.1. Kontroler koncert:

```
using koncert.Models.Repositories;
using Microsoft.AspNetCore.Mvc;

namespace koncert.Controllers
{
   public class ConcertController : Controller
   {
      private readonly IConcertRepository _concertRepository;
      public ConcertController(IConcertRepository concertRepository)
      {
         _concertRepository = concertRepository;
      }

      public IActionResult List()
      {
         var result = _concertRepository.AllConcerts;
         return View(result);
      }
   }
}
```

Entity Framework Core 6

1. Microsoft.EntityFrameworkCore.SqlServer v. 6.0.31
2. Microsoft.EntityFrameworkCore.Tools v. 6.0.31

13. Instrukcja pierwszego uruchomienia projektu

Aplikacja znajduje się pod linkiem: https://github.com/kostadojo/Aplikacja_Koncert.git

1. Wyszukaj <> Code i rozwiń listę
2. Wybierz Open with Visual Studio lub Download ZIP

Opcja Open with Visual Studio:

1. Otwórz Wybierz aplikację
2. Klonuj repozytorium – wprowadź ścieżkę
3. Przycisk klonuj
4. Po poprawnym klonowaniu uruchom: Konsola menadżera pakietów (Lewy dolny róg lub Narzędzia>Menadżer pakietów NuGet> Konsola menadżera pakietów)
5. Wpisz polecenie: update-database
6. Po aktualizacji bazy danych aplikacja jest gotowa do użycia
7. Uruchom

Opcja Download with ZIP:

1. Pobierz plik i rozpakuj
2. Otwórz projekt w Visual Studio 2022
3. Wybierz opcję "Open a project or solution" i przejdź do katalogu, w którym znajduje się plik koncert.sln, a następnie otwórz go
4. Uruchom : Konsola menadżera pakietów (Lewy dolny róg lub Narzędzia>Menadżer pakietów NuGet> Konsola menadżera pakietów)
5. Wpisz polecenie: update-database
6. Po aktualizacji bazy danych aplikacja jest gotowa do użycia
7. Uruchom

Opcjonalnie sprawdź czy są zainstalowane pakiety NuGet w wersji 6.0:

1. Microsoft.AspNetCore.Identity.EntityFrameworkCore
2. Microsoft.AspNetCore.Identity.UI
3. Microsoft.EntityFrameworkCore.Sqlite
4. Microsoft.EntityFrameworkCore.SqlServer
5. Microsoft.EntityFrameworkCore.Tools
6. Microsoft.VisualStudio.Web.CodeGeneration.Design

Pakiety znajdują się: Narzędzia>Menadżer pakietów NuGet> Zarządzań pakietami NuGet rozwiązania…