

Strona internetowa „Elektroniczny Dziennik Szkolny”

Autor projektu: Marcin Stefaniak nr albumu: 138424







1. Krótki opis działania:

Projekt to strona internetowa pełniąca rolę elektronicznego dziennika szkolnego dla uczniów, nauczycieli i administratorów zrealizowaną w architekturze MVC. Umożliwia rejestrację nowych użytkowników, logowanie, przeglądanie i wpisywanie ocen, odnotowywanie obecności na zajęciach oraz zarządzanie przedmiotami. W systemie funkcjonują trzy role: „Administrator”, „Nauczyciel”, „Uczeń” z których każdy posiada odpowiedni poziom dostępu. Uczniowie mogą jedynie przeglądać swoje oceny z poszczególnych przedmiotów oraz aktualny stan ich obecności na wszystkich zajęciach, nauczyciele mają możliwość wprowadzania i modyfikowania ocen, przedmiotów oraz obecności uczniów na zajęciach, a administratorzy posiadają pełne uprawnienia do zarządzania wszystkimi danymi w dzienniku.

2. Specyfikacja wykorzystanych technologii:

- A) Język programowania: C#
- B) Platforma: .NET 8.0
- C) Framework: ASP.NET Core 8.10
- D) Framework ORM: Entity Framework Core 8.11
- E) Baza danych: SQLite Entity Framework
- F) IDE: Microsoft Visual Studio 2022

Wymagane rozszerzenia w Visual Studio 2022 aby aplikacja działa poprawnie !

Pakiety najwyższego poziomu (6)			
	Microsoft.AspNetCore.Identity.EntityFrameworkCore przez: Microsoft	8.0.11	
	ASP.NET Core Identity provider that uses Entity Framework Core.	9.0.0	
	Microsoft.AspNetCore.Identity.UI przez: Microsoft	8.0.11	
	ASP.NET Core Identity UI is the default Razor Pages built-in UI for the ASP.NET Core Identity framework.	9.0.0	
	Microsoft.EntityFrameworkCore.Sqlite przez: Microsoft	8.0.11	
	SQLite database provider for Entity Framework Core.	9.0.0	
	Microsoft.EntityFrameworkCore.Tools przez: Microsoft	8.0.11	
	Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.	9.0.0	
	Microsoft.VisualStudio.Web.CodeGeneration.Design przez: Microsoft	8.0.7	
	Code Generation tool for ASP.NET Core. Contains the dotnet-aspnet-codegenerator command used for generating controllers and views.	9.0.0	
	System.Text.Json przez: Microsoft	8.0.5	
	Provides high-performance and low-allocating types that serialize objects to JavaScript Object Notation (JSON) text and deserialize JSON text to objects, with UTF-8 support built-in. Also provides types to read and write JSON text encoded as UTF-8, and to create an in-memory docu...	9.0.0	

3. Instrukcja pierwszego uruchomienia.

A) Otwórz projekt w Visual Studio 2022 (plik .sln).

B) Sprawdź połączenie z bazą danych w appsettings.json.

C) Otwórz Package Manager Console (Tools -> NuGet Package Manager -> Package Manager Console) – **zainstaluj wszystkie wymagane rozszerzenia patrz pkt. 2**

D) Wykonaj polecenie:

Add-Migration
np. InitialCreate

E) Zaktualizuj bazę danych:

Update-Database

F) Uruchom projekt (F5)

4. Opis struktury projektu wraz z opisem modeli i kontrolerów.

Dziennik Elektroniczny to aplikacja webowa stworzona w technologii ASP.NET Core z wykorzystaniem wzorca MVC (Model-View-Controller).

Modele:

Modele znajdują się w folderze Models. Odpowiadają za mapowanie danych aplikacji na obiekty C#. Spis modeli:

„Administrator”

Model odpowiadający za przechowywanie danych administratorów systemu, takich jak uprawnienia i informacje identyfikacyjne.

```

1 odwołanie
public class Administrator
{
    Odwołania: 0
    public int Id { get; set; }

    [Required(ErrorMessage = "Imię jest wymagane.")]
    [MaxLength(50, ErrorMessage = "Imię nie może mieć więcej niż 50 znaków.")]
    [RegularExpression(@"^[a-zA-Z\s]+$", ErrorMessage = "Imię może zawierać tylko litery i spacje.")]
    Odwołania: 0
    public string? Imie { get; set; }

    [Required(ErrorMessage = "Nazwisko jest wymagane.")]
    [MaxLength(50, ErrorMessage = "Nazwisko nie może mieć więcej niż 50 znaków.")]
    [RegularExpression(@"^[a-zA-Z\s]+$", ErrorMessage = "Nazwisko może zawierać tylko litery i spacje.")]
    Odwołania: 0
    public string? Nazwisko { get; set; }

    [Required(ErrorMessage = "Login jest wymagany.")]
    [MaxLength(50, ErrorMessage = "Login nie może mieć więcej niż 50 znaków.")]
    Odwołania: 0
    public string? Login { get; set; }

    [Required(ErrorMessage = "Hasło jest wymagane.")]
    [MinLength(8, ErrorMessage = "Hasło musi mieć co najmniej 8 znaków.")]
    Odwołania: 0
    public string? Haslo { get; set; }
}

```

Identyfikator administratora. Generowany automatycznie.

Imię administratora. Pole wymagane, maksymalnie 50 znaków. Tylko litery i spacje.

Nazwisko administratora. Pole wymagane, maksymalnie 50 znaków. Może zawierać tylko litery i spacje.

Login administratora. Pole wymagane, maksymalnie 50 znaków.

Hasło administratora. Pole wymagane, minimalnie 8 znaków.

„ErrorViewModel”

Model używany do przekazywania szczegółów o błędach występujących w aplikacji, takich jak komunikaty i identyfikatory błędów.

```

1 namespace Dziennik_elektroniczny.Models
2 {
3     Odwołania: 2
4     public class ErrorViewModel
5     {
6         Odwołania: 4
7         public string? RequestId { get; set; }
8
9         Odwołania: 0
10        public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);
11    }
12 }

```

Identyfikator żądania, który pozwala na śledzenie błędu.

Może być null, jeśli brak szczegółowych danych o żądaniu.

„Klasa”

Model reprezentujący dane dotyczące klas szkolnych, w tym ich nazwy oraz powiązania z uczniami i zajęciami.

```
Dziennik_elektroniczny
1  using System.Collections.Generic;
2  using System.ComponentModel.DataAnnotations;
3
4  namespace Dziennik_elektroniczny.Models
5  {
6      Odwołania: 17
7      public class Klasa
8      {
9          Odwołania: 8
10         public int Id { get; set; }
11
12         [Required(ErrorMessage = "Numer klasy jest wymagany.")]
13         [Range(1, 8, ErrorMessage = "Numer klasy musi być od 1 do 8.")]
14         Odwołania: 7
15         public int Numer { get; set; }
16
17         [Required(ErrorMessage = "Litera klasy jest wymagana.")]
18         [RegularExpression(@"^[A-E]$", ErrorMessage = "Litera klasy musi być jedną z liter: A, B, C, D, E.")]
19         Odwołania: 7
20         public string? Litera { get; set; }
21
22         Odwołania: 4
23         public ICollection<Uczen> Uczniowie { get; set; } = new List<Uczen>();
24
25         Odwołania: 7
26         public string Nazwa => $"{Numer}{Litera}";
27     }
28 }
```

Numer klasy, wymagany w przedziale od 1 do 8.

Litera klasy, wymagana i ograniczona do liter od A do E.

Automatycznie generowana nazwa klasy w formacie "Numer + Litera".

„LoginViewModel”

Model używany podczas logowania użytkowników, przechowujący dane uwierzytelniające, takie jak login i hasło.

```
using System.ComponentModel.DataAnnotations;
1
2  Odwołania: 3
3  public class LoginViewModel
4  {
5      [Required(ErrorMessage = "Nazwa użytkownika jest wymagana.")]
6      Odwołania: 3
7      public string? Username { get; set; }
8
9      [Required(ErrorMessage = "Hasło jest wymagane.")]
10     [DataType(DataType.Password)]
11     Odwołania: 3
12     public string? Password { get; set; }
13
14     Odwołania: 2
15     public bool RememberMe { get; set; }
16 }
17
```

Nazwa użytkownika, wymagana do zalogowania się do systemu.

Hasło, wymagane i wprowadzane w formacie zabezpieczonym.

Opcja zapamiętania użytkownika, pozwala na pozostanie zalogowanym po zamknięciu przeglądarki.

„Nauczyciel”

Model przechowujący dane nauczycieli, w tym ich dane personalne oraz powiązania z przedmiotami i zajęciami.

```
1 using System.ComponentModel.DataAnnotations;
2 namespace Dziennik_elektroniczny.Models;
3
4 Odwołania: 16
5 public class Nauczyciel
6 {
7     Odwołania: 16
8     public int Id { get; set; }
9
10    [Required(ErrorMessage = "Imię jest wymagane.")]
11    [MaxLength(50, ErrorMessage = "Imię nie może mieć więcej niż 50 znaków.")]
12    [RegularExpression(@"^[a-zA-Z\s]+$", ErrorMessage = "Imię może zawierać tylko litery i spacje.")]
13    Odwołania: 19
14    public string? Imie { get; set; }
15
16    [Required(ErrorMessage = "Nazwisko jest wymagane.")]
17    [MaxLength(50, ErrorMessage = "Nazwisko nie może mieć więcej niż 50 znaków.")]
18    [RegularExpression(@"^[a-zA-Z\s]+$", ErrorMessage = "Nazwisko może zawierać tylko litery i spacje.")]
19    Odwołania: 19
20    public string? Nazwisko { get; set; }
21
22    [Required(ErrorMessage = "Przedmiot jest wymagany.")]
23    [MaxLength(10, ErrorMessage = "Przedmiot nie może mieć więcej niż 50 znaków.")]
24    Odwołania: 8
25    public string? Przedmiot { get; set; }
26 }
```

Identyfikator nauczyciela, unikalny dla każdego rekordu w systemie.

Imię nauczyciela, wymagane pole o maksymalnej długości 50 znaków, zawierające wyłącznie litery i spacje.

Nazwisko nauczyciela, wymagane pole o maksymalnej długości 50 znaków, zawierające wyłącznie litery i spacje.

Przedmiot przypisany nauczycielowi, wymagane pole o maksymalnej długości 50 znaków.

„Obecnosc”

Model zapisujący informacje o obecnościach uczniów na zajęciach, w tym datę i status obecności.

```
Odwolania: 29
public class Obecnosc
{
    Odwołania: 14
    public int Id { get; set; }

    [Required(ErrorMessage = "Informacja o obecności jest wymagana.")]
    Odwołania: 14
    public bool Obecny { get; set; }

    [Required(ErrorMessage = "Uczeń jest wymagany.")]
    Odwołania: 20
    public int UczeńId { get; set; }

    Odwołania: 9
    public Uczeń? Uczeń { get; set; }

    [Required(ErrorMessage = "Data zajęć jest wymagana.")]
    Odwołania: 17
    public DateTime DataZajec { get; set; }
}
```

Identyfikator obecności, unikalny dla każdego rekordu w systemie.

Informacja o obecności ucznia, wymagane pole przechowujące wartość logiczną (true/false).

Identyfikator ucznia, wymagane pole wskazujące ucznia przypisanego do danego wpisu obecności.

Obiekt ucznia, opcjonalne pole reprezentujące powiązanego ucznia.

Data zajęć, wymagane pole określające datę, dla której odnotowano obecność.

„Oceny”

Model reprezentujący oceny przypisane uczniom, zawierający informacje o przedmiocie, wartości oceny i uczniu.

```
Odwołania: 36
public class Oceny
{
    Odwołania: 14
    public int Id { get; set; }

    [Required(ErrorMessage = "Ocena jest wymagana.")]
    [Range(1, 6, ErrorMessage = "Ocena musi być w zakresie od 1 do 6.")]
    Odwołania: 20
    public decimal Ocena { get; set; }

    [Required(ErrorMessage = "Data oceny jest wymagana.")]
    Odwołania: 18
    public DateTime DataOceny { get; set; }

    [Required(ErrorMessage = "Uczeń jest wymagany.")]
    Odwołania: 24
    public int UczeńId { get; set; }

    Odwołania: 13
    public Uczeń? Uczeń { get; set; }

    [Required(ErrorMessage = "Przedmiot jest wymagany.")]
    Odwołania: 23
    public int PrzedmiotId { get; set; }

    Odwołania: 10
    public Przedmiot? Przedmiot { get; set; }
}
```

Identyfikator oceny, unikalny dla każdego wpisu w systemie.

Ocena ucznia, wymagane pole z wartością od 1 do 6.

Data wystawienia oceny, wymagane pole określające dzień, w którym została wystawiona ocena.

Identyfikator ucznia, wymagane pole wskazujące ucznia, któremu przypisano ocenę.

Obiekt ucznia, opcjonalne pole reprezentujące powiązanego ucznia.

Identyfikator przedmiotu, wymagane pole określające przedmiot, do którego przypisana jest ocena.

Obiekt przedmiotu, opcjonalne pole reprezentujące powiązany przedmiot.

„Przedmiot”

Model przechowujący dane o przedmiotach nauczanych w szkole oraz ich powiązania z nauczycielami i zajęciami.

```
1 using System.ComponentModel.DataAnnotations;
2 namespace Dziennik_elektroniczny.Models;
3
4 public class Przedmiot
5 {
6     public int Id { get; set; }
7
8     [Required(ErrorMessage = "Nazwa przedmiotu jest wymagana.")]
9     [StringLength(50, ErrorMessage = "Nazwa przedmiotu może mieć maksymalnie 50 znaków.")]
10    public string? Nazwa { get; set; }
11
12    [Required(ErrorMessage = "Nauczyciel jest wymagany.")]
13    public int NauczycielId { get; set; }
14
15    public Nauczyciel? Nauczyciel { get; set; }
16 }
17
```

Identyfikator przedmiotu, unikalny dla każdego wpisu w systemie.

Nazwa przedmiotu, wymagane pole z maksymalną długością 50 znaków.

Identyfikator nauczyciela, wymagane pole wskazujące nauczyciela prowadzącego przedmiot.

Obiekt nauczyciela, opcjonalne pole reprezentujące powiązanego nauczyciela.

„RegisterViewModel”

Model używany podczas rejestracji nowych użytkowników, zawierający dane potrzebne do utworzenia konta, takie jak login, hasło i e-mail.

```
{
    Odwołania: 2
    public class RegisterViewModel
    {
        [Required(ErrorMessage = "Nazwa użytkownika jest wymagana.")]
        [StringLength(30, MinimumLength = 3, ErrorMessage = "Nazwa użytkownika musi mieć od 3 do 30 znaków.")]
        [RegularExpression(@"^[a-zA-Z0-9]+$", ErrorMessage = "Nazwa użytkownika może zawierać tylko litery i cyfry.")]
        Odwołania: 4
        public string? Username { get; set; }

        [Required(ErrorMessage = "Adres e-mail jest wymagany.")]
        [EmailAddress(ErrorMessage = "Niepoprawny format adresu e-mail.")]
        Odwołania: 4
        public string? Email { get; set; }

        [Required(ErrorMessage = "Hasło jest wymagane.")]
        [StringLength(100, MinimumLength = 8, ErrorMessage = "Hasło musi mieć co najmniej 8 znaków.")]
        [DataType(DataType.Password)]
        Odwołania: 5
        public string? Password { get; set; }

        [Required(ErrorMessage = "Potwierdzenie hasła jest wymagane.")]
        [Compare("Password", ErrorMessage = "Hasła muszą być takie same.")]
        [DataType(DataType.Password)]
        Odwołania: 3
        public string? ConfirmPassword { get; set; }

        [Required(ErrorMessage = "Rola użytkownika jest wymagana.")]
        [RegularExpression(@"^(Uczen|Nauczyciel|Administrator)$", ErrorMessage = "Nieprawidłowa rola użytkownika.")]
        Odwołania: 7
        public string? Role { get; set; }
    }
}
```

Nazwa użytkownika, wymagane pole o długości od 3 do 30 znaków, akceptujące tylko litery i cyfry.

Adres e-mail, wymagane pole w formacie zgodnym z adresem e-mail.

Hasło, wymagane pole o minimalnej długości 8 znaków, zabezpieczone przed jawnym wyświetlaniem.

Potwierdzenie hasła, wymagane pole, które musi być identyczne jak hasło.

Rola użytkownika, wymagane pole akceptujące jedną z wartości: Uczeń, Nauczyciel lub Administrator.

„Uczeń”

Model reprezentujący uczniów, przechowujący ich dane personalne, powiązania z klasami, ocenami oraz obecnościami.

```
namespace Dziennik_elektroniczny.Models
{
    public class Uczeń
    {
        public int Id { get; set; }

        [Required(ErrorMessage = "Imię ucznia jest wymagane.")]
        [MaxLength(50, ErrorMessage = "Imię ucznia nie może mieć więcej niż 50 znaków.")]
        public string? Imie { get; set; }

        [Required(ErrorMessage = "Nazwisko ucznia jest wymagane.")]
        [MaxLength(50, ErrorMessage = "Nazwisko ucznia nie może mieć więcej niż 50 znaków.")]
        public string? Nazwisko { get; set; }

        [Required]
        [ForeignKey("Klasa")]
        public int KlasaId { get; set; }

        public Klasa? Klasa { get; set; }
    }
}
```

Imię ucznia, wymagane pole o maksymalnej długości 50 znaków.

Nazwisko ucznia, wymagane pole o maksymalnej długości 50 znaków.

Identyfikator klasy, wymagane pole wskazujące przynależność ucznia do konkretnej klasy.

„ViewModel”

Ogólny model bazowy używany do agregacji danych na potrzeby widoków, aby ułatwić zarządzanie danymi przekazywanymi do widoków.

```
using Microsoft.AspNetCore.Mvc.Rendering;

namespace Dziennik_elektroniczny.Models.ViewModels
{
    Odwołania: 0
    public class DodajPrzedmiotViewModel
    {
        Odwołania: 0
        public int? PrzedmiotId { get; set; }
        Odwołania: 0
        public IEnumerable<SelectListItem>? Przedmioty { get; set; }
    }
}
```

Identyfikator przedmiotu, opcjonalne pole przechowujące wybrany przedmiot.
Lista przedmiotów, opcjonalne pole zawierające kolekcję możliwych do wyboru przedmiotów.

„Zajecia”

Model przechowujący dane o zajęciach, w tym ich harmonogram, przypisane klasy i przedmioty.

```
Odwołania: 26
public class Zajecia
{
    Odwołania: 13
    public int Id { get; set; }

    [Required(ErrorMessage = "Data zajęć jest wymagana.")]
    Odwołania: 15
    public DateTime DataZajec { get; set; }

    [Required(ErrorMessage = "Klasa jest wymagana.")]
    Odwołania: 18
    public int KlasaId { get; set; }

    Odwołania: 6
    public Klasa? Klasa { get; set; }

    [Required(ErrorMessage = "Przedmiot jest wymagany.")]
    Odwołania: 18
    public int PrzedmiotId { get; set; }

    Odwołania: 6
    public Przedmiot? Przedmiot { get; set; }
}
```

Data zajęć, pole wymagane przechowujące datę zajęć.

Identyfikator klasy, pole wymagane wskazujące klasę, która uczestniczy w zajęciach.

Obiekt klasy, opcjonalne pole reprezentujące klasę uczestniczącą w zajęciach.

Identyfikator przedmiotu, pole wymagane wskazujące przedmiot realizowany podczas zajęć.

Obiekt przedmiotu, opcjonalne pole reprezentujące przedmiot realizowany podczas zajęć.

B) Kontrolery:

Kontrolery znajdują się w folderze Controllers. Każdy kontroler odpowiada za logikę obsługi żądań użytkowników i przekazywanie danych do odpowiednich widoków. Spis kontrolerów:

„AccountController”

AccountController zarządza procesami uwierzytelniania i autoryzacji użytkowników w systemie. Obejmuje funkcje rejestracji, logowania, wylogowywania oraz potwierdzania kont za pomocą tokenów. Dostosowuje zachowanie aplikacji w zależności od ról przypisanych użytkownikowi (Uczeń, Nauczyciel, Administrator).

Metoda **Register()** obsługuje żądania GET i POST. Dla żądań GET wyświetla formularz rejestracji użytkownika z możliwością wyboru roli (Uczeń, Nauczyciel, Administrator). Dla żądań POST waliduje dane formularza, tworzy konto użytkownika, przypisuje odpowiednią rolę oraz generuje link aktywacyjny. W przypadku błędów zwraca widok rejestracji z komunikatami.

Metoda **ConfirmEmail()** obsługuje żądania GET i służy do potwierdzenia adresu e-mail użytkownika na podstawie tokenu i identyfikatora użytkownika. Po pomyślnym potwierdzeniu aktywuje konto i zwraca widok potwierdzenia. W przypadku błędów wyświetla odpowiedni komunikat.

Metoda **Login()** obsługuje żądania GET i POST. Dla żądań GET wyświetla formularz logowania użytkownika. Dla żądań POST weryfikuje dane logowania i po sukcesie przekierowuje użytkownika do odpowiedniego panelu w zależności od przypisanej roli (Administrator, Nauczyciel, Uczeń). Przy błędach zwraca widok logowania z komunikatem.

Metoda **Logout()** obsługuje żądania POST i realizuje proces wylogowania użytkownika z systemu. Po zakończeniu procesu przekierowuje użytkownika na stronę główną.

„AdministratorController”

AdministratorController jest odpowiedzialny za zarządzanie systemem z perspektywy administratora. Umożliwia zarządzanie danymi uczniów, klas, nauczycieli, przedmiotów, zajęć, obecności oraz ocen. Kontroler udostępnia akcje do dodawania, edycji oraz usuwania rekordów w różnych częściach systemu.

Metoda **Dashboard()** obsługuje żądania GET i wyświetla główny widok panelu administratora.

Metoda **Uczniowie()** zwraca widok zawierający listę wszystkich uczniów, uwzględniając ich przynależność do klas. Żądania GET pobierają dane z bazy i przekazują je do widoku.

Metoda **DodajUcznia()** obsługuje formularz dodawania nowego ucznia. W przypadku żądania GET wyświetlany jest formularz z listą dostępnych klas, natomiast żądanie POST zapisuje nowego ucznia w bazie danych.

Metoda **EdytujUcznia()** działa podobnie jak DodajUcznia, ale dotyczy edycji istniejącego ucznia. Żądanie GET ładuje dane ucznia do formularza, a POST zapisuje zmiany w bazie danych.

Metoda **UsunUcznia()** umożliwia usunięcie ucznia. Żądanie GET wyświetla stronę potwierdzenia, a POST usuwa ucznia z bazy danych, jeśli istnieje.

Metoda **Klasy()** wyświetla listę wszystkich klas w systemie, uwzględniając liczbę przypisanych uczniów. Żądania GET pobierają dane i renderują widok.

Metoda **DodajKlase()** obsługuje formularz dodawania nowych klas. W żądaniu GET wyświetla pusty formularz, a w żądaniu POST zapisuje nową klasę w bazie danych.

Metoda **EdytujKlase()** umożliwia edycję danych istniejącej klasy. GET ładuje dane klasy do formularza, a POST zapisuje zmiany w bazie danych.

Metoda **UsunKlase()** pozwala na usunięcie klasy, o ile nie ma przypisanych uczniów. GET wyświetla stronę potwierdzenia, a POST usuwa klasę z bazy danych.

Metoda **Nauczyciele()** zwraca widok listy nauczycieli w systemie. Żądania GET pobierają dane z bazy i prezentują je w widoku.

Metoda **DodajNauczyciela()** obsługuje dodawanie nowych nauczycieli. GET wyświetla formularz, a POST zapisuje nauczyciela w bazie danych.

Metoda **EdytujNauczyciela()** pozwala na edycję istniejącego nauczyciela. GET ładuje dane do formularza, a POST zapisuje zmiany.

Metoda **UsunNauczyciela()** umożliwia usunięcie nauczyciela. GET wyświetla stronę potwierdzenia, a POST usuwa nauczyciela z bazy danych.

Metoda **Przedmioty()** wyświetla listę wszystkich przedmiotów z przypisanymi nauczycielami. Żądania GET renderują widok na podstawie danych z bazy.

Metoda **DodajPrzedmiot()** obsługuje dodawanie nowych przedmiotów. GET wyświetla formularz z listą nauczycieli, a POST zapisuje nowy przedmiot w bazie.

Metoda **EdytujPrzedmiot()** umożliwia edycję danych przedmiotu. GET ładuje dane przedmiotu do formularza, a POST zapisuje zmiany w bazie.

Metoda **UsunPrzedmiot()** pozwala na usunięcie przedmiotu. GET wyświetla stronę potwierdzenia, a POST usuwa przedmiot z bazy danych.

Metoda **Zajecia()** wyświetla listę wszystkich zajęć w systemie. GET renderuje widok z zajęciami, uwzględniając przypisane klasy i przedmioty.

Metoda **DodajZajecia()** obsługuje dodawanie nowych zajęć. GET wyświetla formularz z listą klas i przedmiotów, a POST zapisuje nowe zajęcia w bazie.

Metoda **EdytujZajecia()** umożliwia edycję danych zajęć. GET ładuje dane zajęć do formularza, a POST zapisuje zmiany w bazie.

Metoda **UsunZajecia()** pozwala na usunięcie zajęć. GET wyświetla stronę potwierdzenia, a POST usuwa zajęcia z bazy danych.

Metoda **Obecnosci()** wyświetla listę obecności uczniów na zajęciach. Żądania GET renderują widok na podstawie danych z bazy.

Metoda **DodajObecnosc()** obsługuje dodawanie nowych rekordów obecności. GET wyświetla formularz z listą uczniów, a POST zapisuje obecność w bazie.

Metoda **EdytujObecnosc()** umożliwia edycję danych obecności. GET ładuje dane obecności do formularza, a POST zapisuje zmiany.

Metoda **UsunObecnosc()** pozwala na usunięcie rekordu obecności. GET wyświetla stronę potwierdzenia, a POST usuwa obecność z bazy danych.

Metoda **Oceny()** wyświetla listę ocen uczniów. Żądania GET renderują widok na podstawie danych z bazy.

Metoda **DodajOcene()** obsługuje dodawanie nowych ocen. GET wyświetla formularz z listą uczniów i przedmiotów, a POST zapisuje ocenę w bazie.

Metoda **EdytujOcene()** umożliwia edycję danych oceny. GET ładuje dane oceny do formularza, a POST zapisuje zmiany.

Metoda **UsunOcene()** pozwala na usunięcie oceny. GET wyświetla stronę potwierdzenia, a POST usuwa ocenę z bazy danych.

„NauczycielController”

NauczycieleController odpowiada za funkcjonalności związane z zarządzaniem przez nauczycieli uczniami, przedmiotami, zajęciami, ocenami oraz obecnościami. Kontroler wymaga autoryzacji dla roli "Nauczyciel".

Metoda **Dashboard()** obsługuje żądania GET i wyświetla panel główny nauczyciela z informacją powitalną. Przekazuje komunikat do widoku za pomocą ViewData.

Metoda **Uczniowie()** obsługuje żądania GET i wyświetla listę wszystkich uczniów z przypisanymi klasami. Pobiera dane z bazy przy użyciu metody Include dla powiązanych klas.

Metoda **DodajUcznia()** obsługuje żądania GET i POST. Dla żądań GET wyświetla formularz do dodania nowego ucznia oraz przekazuje listę klas jako SelectList. Dla żądań POST zapisuje nowego ucznia w bazie danych po weryfikacji danych z formularza.

Metoda **EdytujUcznia()** obsługuje żądania GET i POST. Dla żądań GET pobiera dane istniejącego ucznia na podstawie identyfikatora i wyświetla formularz edycji z listą klas. Dla żądań POST aktualizuje dane ucznia w bazie danych po walidacji.

Metoda **Przedmioty()** obsługuje żądania GET i wyświetla listę wszystkich przedmiotów wraz z przypisanymi nauczycielami. Pobiera dane z bazy przy użyciu metody Include dla powiązanych nauczycieli.

Metoda **DodajPrzedmiot()** obsługuje żądania GET i POST. Dla żądań GET wyświetla formularz do dodania nowego przedmiotu oraz przekazuje listę nauczycieli jako SelectList. Dla żądań POST zapisuje nowy przedmiot w bazie danych po weryfikacji danych z formularza.

Metoda **EdytujPrzedmiot()** obsługuje żądania GET i POST. Dla żądań GET pobiera dane istniejącego przedmiotu na podstawie identyfikatora i wyświetla formularz edycji z listą nauczycieli. Dla żądań POST aktualizuje dane przedmiotu w bazie danych po walidacji.

Metoda **Zajecia()** obsługuje żądania GET i wyświetla listę wszystkich zajęć z przypisanymi klasami i przedmiotami. Pobiera dane z bazy przy użyciu metody Include.

Metoda **DodajZajecia()** obsługuje żądania GET i POST. Dla żądań GET wyświetla formularz do dodania nowych zajęć oraz przekazuje listy klas i przedmiotów jako SelectList. Dla żądań POST zapisuje nowe zajęcia w bazie danych po weryfikacji danych z formularza.

Metoda **Oceny()** obsługuje żądania GET i wyświetla listę wszystkich ocen uczniów z przypisanymi przedmiotami. Pobiera dane z bazy przy użyciu metody Include.

Metoda **DodajOcene()** obsługuje żądania GET i POST. Dla żądań GET wyświetla formularz do wystawienia nowej oceny oraz przekazuje listy uczniów i przedmiotów jako SelectList. Dla żądań POST zapisuje nową ocenę w bazie danych po weryfikacji danych z formularza.

Metoda **Obecnosci()** obsługuje żądania GET i wyświetla listę wszystkich obecności uczniów. Pobiera dane z bazy przy użyciu metody Include dla powiązanych uczniów.

Metoda **DodajObecnosc()** obsługuje żądania GET i POST. Dla żądań GET wyświetla formularz do dodania obecności oraz przekazuje listę uczniów jako SelectList. Dla żądań POST zapisuje nową obecność w bazie danych po weryfikacji danych z formularza.

Metoda **Wystawienie()** obsługuje żądania GET i wyświetla listę ocen uczniów w kontekście przypisanych przedmiotów i klas. Pobiera dane z bazy przy użyciu metod Include.

Metoda **WystawOcene()** obsługuje żądania GET i POST. Dla żądań GET wyświetla formularz do wystawienia oceny z możliwością wyboru ucznia, przedmiotu i klasy jako SelectList. Dla żądań POST zapisuje ocenę oraz dodatkowe informacje, takie jak data zajęć i obecność, w bazie danych po weryfikacji danych z formularza.

„HomeController”

HomeController to kontroler odpowiedzialny za zarządzanie głównymi widokami aplikacji, takimi jak strona główna oraz panele dla użytkowników zalogowanych i administratorów. Wykorzystuje mechanizm logowania informacji o działaniach użytkowników za pomocą ILogger. Kontroler zawiera również obsługę błędów.

Metoda **Index()** obsługuje żądania GET i wyświetla stronę główną aplikacji. Loguje informację o jej wyświetleniu.

Metoda **About()** obsługuje żądania GET i wyświetla stronę informacyjną "O nas". Loguje informację o odwiedzeniu tej strony.

Metoda **Dashboard()** dostępna wyłącznie dla zalogowanych użytkowników (zabezpieczona atrybutem [Authorize]). Obsługuje żądania GET i wyświetla panel użytkownika. Loguje informację o dostępie do tego widoku.

Metoda **AdminPanel()** przeznaczona wyłącznie dla użytkowników z rolą "Administrator" (zabezpieczona atrybutem [Authorize(Roles = "Administrator")]). Obsługuje żądania GET i wyświetla panel administracyjny. Loguje informację o dostępie administratora do panelu.

Metoda **Error()** obsługuje błędy aplikacji. Generuje widok błędu z modelem `ErrorViewModel`, zawierającym identyfikator żądania (`RequestId`) oraz inne szczegóły błędu. Loguje informację o wystąpieniu błędu.

„UczenController”

`UczenController` odpowiada za obsługę widoku i funkcjonalności dostępnych dla zalogowanego użytkownika z rolą "Uczeń". Kontroler umożliwia przeglądanie panelu ucznia, ocen oraz listy obecności. Wszystkie akcje są zabezpieczone atrybutem `[Authorize(Roles = "Uczen")]`, co zapewnia, że dostęp mają jedynie użytkownicy z odpowiednią rolą.

Metoda **Dashboard()** obsługuje żądania GET i wyświetla panel ucznia. Przekazuje użytkownikowi komunikat powitalny w `ViewData`.

Metoda **Oceny()** obsługuje żądania GET i wyświetla listę ocen przypisanych zalogowanemu uczniowi. Wykorzystuje identyfikator użytkownika z kontekstu uwierzytelnienia, aby pobrać dane ucznia z bazy. Jeśli dane ucznia nie zostaną znalezione, metoda przekierowuje użytkownika na stronę Dashboard z komunikatem o błędzie. Pobiera oceny ucznia z bazy danych i dołącza powiązane przedmioty.

Metoda **Obecnosci()** obsługuje żądania GET i wyświetla listę obecności zalogowanego ucznia. Wykorzystuje identyfikator użytkownika z kontekstu uwierzytelnienia, aby znaleźć dane ucznia w bazie. W przypadku braku danych ucznia przekierowuje użytkownika na stronę Dashboard z komunikatem o błędzie. Pobiera listę obecności ucznia z bazy danych i przekazuje ją do widoku.

C) Widoki – bez szczegółowego opisu.

Widoki znajdują się w folderze Views. Odpowiadają za prezentację danych użytkownikowi i są podzielone według kontrolerów. Widoki są również zorganizowane w subfoldery przypisane do ról:

Account

Administrator

Nauczyciele

Shared

Uczen

D) Baza danych – EntityFrameworkSQLite

Klasa DziennikContext jest centralnym punktem zarządzania danymi w projekcie Dziennik Elektroniczny. Dziedziczy ona po IdentityDbContext, co pozwala na korzystanie z mechanizmów ASP.NET Identity, takich jak zarządzanie użytkownikami, rolami czy logowaniem.

Klasa DziennikContext zapewnia mapowanie obiektów aplikacji na tabele w bazie danych za pomocą Entity Framework Core. Dzięki niej możliwe jest łatwe zarządzanie danymi oraz operacjami CRUD .

DbSet<Uczen> Uczniowie: Reprezentuje tabelę uczniów, przechowującą dane, takie jak imię, nazwisko, klasa czy oceny przypisane do konkretnego ucznia.

DbSet<Nauczyciel> Nauczyciele: Odpowiada za dane nauczycieli, w tym ich przypisane przedmioty oraz zajęcia.

DbSet<Administrator> Administratorzy: Tabela przechowująca dane administratorów systemu.

DbSet<Klasa> Klasy: Zawiera informacje o klasach, takich jak numer, litera oraz powiązani uczniowie.

DbSet<Przedmiot> Przedmioty: Obejmuje dane o przedmiotach prowadzonych w systemie, takie jak nazwa przedmiotu czy nauczyciel odpowiedzialny za jego realizację.

DbSet<Zajecia> Zajecia: Przechowuje dane dotyczące zajęć, takie jak data, klasa oraz przypisany przedmiot.

DbSet<Obecnosc> Obecności: Odpowiada za rejestrację obecności uczniów na zajęciach.

DbSet<Oceny> Oceny: Reprezentuje tabele ocen, przypisanych do uczniów w kontekście konkretnych przedmiotów i dat.

5. Opis systemu użytkowników:

System użytkowników w moim projekcie Dziennika Elektronicznego jest oparty na trzech rolach: Uczeń, Nauczyciel i Administrator. Każda z tych ról ma przypisane różne uprawnienia i dostęp do odpowiednich funkcjonalności.

Role w systemie:

Uczeń:

Uczniowie mają możliwość zalogowania się do panelu ucznia, gdzie mogą przeglądać swoje oceny oraz obecności. Dane takie jak lista ocen czy obecności są ściśle powiązane z kontem ucznia. Uczniowie nie mają możliwości edycji danych tylko przegląd.

Nauczyciel:

Panel nauczyciela umożliwia zarządzanie uczniami, przedmiotami, zajęciami, ocenami, obecnościami oraz wystawianie ocen. Nauczyciel widzi wszystkie dane tak jak administrator, ale ma ograniczony zakres może tylko dodawać nowe dane lub je edytować, **tylko** administrator ma prawo usunięcia.

Administrator:

Administrator ma pełny dostęp do systemu i może zarządzać uczniami, nauczycielami, klasami, przedmiotami, zajęciami, ocenami i obecnościami itd. Rola administratora nie jest związana z żadnymi danymi osobistymi, ponieważ jej zadaniem jest zarządzanie aplikacją jako całością i dodatkowo ma prawo usuwania danych.

Różnice między użytkownikami zalogowanymi a gośćmi.

Goście, czyli osoby niezalogowane, mają dostęp jedynie do głównej strony powitalnej. Zalogowani użytkownicy, w zależności od swojej roli, otrzymują dostęp do dedykowanych paneli oraz funkcji. System został zaprojektowany tak, aby każda rola miała jasno określone uprawnienia i zakres działań, co jest zabezpieczone przez atrybuty [Authorize] oraz [Authorize(Roles = "...")] w kontrolerach.