PROGRAMOWANIE ZAAWANSOWANE

Dokumentacja techniczna projektu zaliczeniowego

Tytuł projektu: System oceny filmów

Autorzy projektu:

- Maria Blandzi 138451
- Daniel Urbanowicz 138434

Opis projektu

Proponowana aplikacja umożliwi jej użytkownikom zarządzanie personalną listą filmów oraz dokonywanie ich oceny.

Ocena filmu możliwa będzie w skali od 1 (nie polecam) do 5 (polecam).
Oprócz oceny liczbowej możliwe będzie również pozostawienie tekstowego opisu (zwanego dalej recenzją), stanowiącego z założenia opis wrażeń po obejrzeniu filmu.

Niezalogowani użytkownicy będą mogli przeglądać listę filmów oraz zapoznać się z ocenami innych osób.
Zalogowani użytkownicy mogą dodawać filmy do własnej listy filmów „Do obejrzenia” oraz oznaczać je jako „Obejrzane”.

Dla filmów oznaczonych jako “Obejrzane” dostępna będzie funkcjonalność zarządzania opiniami i recenzjami, w ramach której możliwe będzie pozostawienie oceny i opcjonalnej recenzji, ich edycja oraz usuwanie.

1. Specyfikacja wykorzystanych technologii

1) Wersja platformy

Aplikacja została zbudowana w oparciu o C# i .NET 8.0, co zapewnia najnowsze funkcje i wysoką wydajność w kontekście aplikacji webowych.

2) Wykorzystane biblioteki NuGet

W projekcie wykorzystano następujące biblioteki:

a) **Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore** (8.0.8)
Obsługuje diagnostykę błędów w kontekście bazy danych.
b) **Microsoft.AspNetCore.Identity.EntityFrameworkCore** (8.0.8)
Odpowiada za integrację systemu tożsamości (Identity) z bazą danych, zarządzaną przez Entity Framework Core.
c) **Microsoft.AspNetCore.Identity.UI** (8.0.8)
Dostarcza gotowe komponenty interfejsu użytkownika dla systemu tożsamości, takie jak formularze logowania czy rejestracji.
d) **Microsoft.EntityFrameworkCore.SqlServer** (8.0.8)
Umożliwia korzystanie z SQL Server jako bazy danych dla aplikacji.
e) **Microsoft.EntityFrameworkCore.Tools** (8.0.11)
Zapewnia narzędzia wspierające zarządzanie migracjami i bazą danych w Entity Framework Core.
f) **Microsoft.VisualStudio.Web.CodeGeneration.Design** (8.0.7)
Narzędzia do generowania kodu, w tym automatycznego tworzenia kontrolerów i widoków w Visual Studio.

3) Baza danych

Aplikacja korzysta z **SQL Server** [SQL Server Express LocalDB] jako systemu zarządzania bazą danych.
Połączenie z bazą danych definiowane jest w pliku konfiguracyjnym aplikacji (appsettings.json), a migracje i zarządzanie schematem danych odbywają się za pomocą **Entity Framework Core**.

2. Instrukcja pierwszego uruchomienia projektu

1) Uruchom Visual Studio.
2) Wybierz opcję “Klonuj repozytorium”.
3) W polu adresu URL repozytorium wklej link: http://github.com/Basilisk-404/MoviesReviewer
4) Wybierz pusty folder w którym projekt zostanie zapisany i potwierdź.
5) Po zakończeniu klonowanie, otwórz konsolę menedżera pakietów NuGet (z narzędzi) i wykonaj polecenie ‘Update-Database’
6) Uruchom aplikację, klikając zielony przycisk uruchom na pasku narzędzi.

3. Struktura projektu

Projekt **MoviesReviewer** to aplikacja webowa napisana w ASP.NET Core MVC. Umożliwia użytkownikom zarządzanie filmami, oznaczanie ich preferencji oraz tworzenie opinii. Struktura projektu opiera się na:

1) **Modelach**: Reprezentują dane aplikacji oraz zawierają walidację.
2) **Kontrolerach**: Realizują logikę biznesową i odpowiadają za obsługę żądań HTTP.
3) **Widokach**: Odpowiadają za generowanie interfejsu użytkownika.
4) **Baza danych**: Obsługiwana przez **Entity Framework**, definiująca relacje między tabelami.

Główne foldery:

- Models: Zawiera klasy reprezentujące strukturę danych w aplikacji.
- Controllers: Zawiera logikę odpowiadającą za obsługę zapytań i interakcje użytkownika.
- Views: Przechowuje pliki Razor odpowiadające za interfejs użytkownika.
- Enums: Przechowuje klasę PreferenceType, zawierającą możliwe typy preferencji
- Dtos: Przechowuje klasę PreferenceTypeDto służącą do przekazywania danych dot. preferencji pomiędzy kontrolerem a widokami

Głównym elementem projektu są filmy (Movie). Na ich podstawie tworzona jest lista preferencji (Preference). Dzięki współistnieniu filmów i preferencji możliwe jest tworzenie opinii (Reviews).

4. Modele

1) ErrorViewModel.cs

CEL: Model używany do wyświetlania szczegółów błędów w aplikacji.
POLA:

a) RequestId: *(string?)* — ID zapytania, które spowodowało błąd.
b) ShowRequestId: *(bool)* — Wskazuje, czy RequestId powinno być wyświetlone (gdy nie jest null ani puste).

2) Movie.cs

CEL: Reprezentuje film w aplikacji.
POLA:

a) Id: *(int)* — Klucz główny - ID filmu.
b) Year: *(int)* — Rok produkcji filmu; walidacja: zakres od 1888 do 2100.
c) Title: *(string)* — Tytuł filmu; walidacja: minimalna długość 2, maksymalna 300 znaków.
d) Author: *(string)* — Reżyser filmu; walidacja: minimalna długość 2, maksymalna 300 znaków.
e) UserId: *(string?)* — ID użytkownika, który dodał film.
f) User: *(IdentityUser?)* — Powiązanie z użytkownikiem.

3) Preference.cs

CEL: Przechowuje preferencje użytkownika wobec filmów.
POLA:

a) Id: *(int)* — Klucz główny - ID preferencji.
b) Type: *(string)* — Typ preferencji - możliwy spośród dostępnych w PreferenceType.cs
Sprawdzanie poprawności wyboru odbywa się na etapie tworzenia lub edycji preferencji
c) UserId: *(string?)* — ID użytkownika.
d) User: *(IdentityUser?)* — Powiązanie z użytkownikiem.
e) MovieId: *(int?)* — ID filmu.
f) Movie: *(Movie?)* — Powiązanie z filmem.

4) **Review.cs**

CEL: Reprezentuje opinię o filmie stworzoną przez użytkownika.
POLA:

a) Id: *(int)* — Klucz główny - ID opinii.
b) Value: *(int)* — Ocena filmu (1–5); walidacja: zakres od 1 do 5.
c) Comment: *(string?)* — Komentarz do opinii (tzw. “recenzja”); walidacja: długość od 3 do 1000 znaków.
d) CreatedAt: *(DateTime?)* — Data utworzenia recenzji.
e) UserId: *(string?)* — ID użytkownika tworzącego recenzję.
f) User: *(IdentityUser?)* — Powiązanie z użytkownikiem.
g) MovieId: *(int?)* — ID recenzowanego filmu.
h) Movie: *(Movie?)* — Powiązanie z filmem.

5. Kontrolery

1) **ReviewsController**
CEL → Zarządza recenzjami użytkowników
   a) **Index**
      i) Metoda HTTP: GET
      ii) Parametry: Brak
      iii) Opis
      Wyświetlenie listy istniejących opinii
      iv) Zwracane dane
      Widok z listą recenzji (View)
   b) **Details**
      i) Metoda HTTP: GET
      ii) Parametry
         (1) int? id
      iii) Opis
      Wyświetla szczegóły opinii o podanym ID. Jeśli brak ID lub recenzji, zwraca NotFound.
      iv) Zwracane dane
      Szczegóły recenzji (View)
   c) **My**
      i) Metoda HTTP: GET
      ii) Parametry: Brak
      iii) Opis
      Wyświetlenie listy wszystkich istniejących opinii należących do zalogowanego użytkownika, wraz z informacją o filmie, którego dotyczą.
      iv) Zwracane dane
      Widok z listą opinii użytkownika (View)

d) Create

i) Metoda HTTP: GET

ii) Parametry

(1) int? mov

iii) Opis

Metoda odpowiedzialna za wyświetlanie formularza tworzenia opinii dla filmu o podanym ID.
W jej ramach następuje sprawdzanie czy film o podanym ID istnieje, czy wybrany film został obejrzany oraz czy nie została już utworzona opinia.

iv) Zwracane dane

(1) ALBO - Formularz recenzji (View)

(2) ALBO - CustomErrorView - Widok z informacją o błędzie. Zwracany gdy któryś z warunków powyżej nie jest spełniony

e) Create

i) Metoda HTTP: POST

ii) Parametry

(1) Review review

iii) Opis

Metoda odpowiedzialna za utworzenie opinii dla filmu o podanym ID.
W jej ramach następuje sprawdzanie czy film o podanym ID istnieje, czy wybrany film został obejrzany oraz czy nie została już utworzona opinia. W przypadku spełnienia tych warunków tworzona jest nowa opinia

iv) Zwracane dane

(1) ALBO - Przekierowanie do listy moich opinii

(2) ALBO - CustomErrorView - Widok z informacją o błędzie. Zwracany gdy któryś z warunków powyżej nie jest spełniony

(3) ALBO Widok tworzenia - gdy nie udało się dodać opinii

f) Edit

i) Metoda HTTP: GET

ii) Parametry:

(1) int ?id

iii) Opis

Funkcja odpowiedzialna za wyświetlenie formularza edycji opinii

Sprawdza czy opinia istnieje, czy należy do zalogowanego użytkownika oraz czy istnieje film, którego dotyczy recenzja

iv) Zwracane dane

(1) Formularz edycji (View)

(2) ALBO CustomErrorView (View) - gdy film nie istnieje

(3) ALBO NotFound - gdy opinia nie istnieje albo nie należy do zalogowanego użytkownika

g) Edit

i) Metoda HTTP: POST

ii) Parametry:

(1) int id

(2) Review r

iii) Opis

Funkcja odpowiedzialna za edycję opinii

Sprawdza czy opinia istnieje, czy należy do zalogowanego użytkownika oraz czy istnieje film, którego dotyczy recenzja

iv) Zwracane dane

(1) Przekierowanie do listy opinii

(2) ALBO CustomErrorView (View) - gdy film nie istnieje

(3) ALBO NotFound - gdy opinia nie istnieje albo nie należy do zalogowanego użytkownika

(4) ALBO widok edycji, gdy nie udało się zapisać zmian

h) Delete

i) Metoda HTTP: GET

ii) Parametry:

(1) int? id

iii) Opis

Funkcja odpowiedzialna za wyświetlanie widoku potwierdzenia usunięcia opinii

Sprawdza czy opinia istnieje oraz czy należy do zalogowanego użytkownika

iv) Zwracane dane

(1) Widok potwierdzenia usunięcia (View)

(2) ALBO NotFound - gdy opinia nie istnieje albo nie należy do zalogowanego użytkownika

i) **DeleteConfirmed**
   i) Metoda HTTP: POST
   ii) Parametry:
      (1) int id
   iii) Opis
   Funkcja odpowiedzialna za usunięcie opinii jeśli zalogowany użytkownik jest jej autorem
   iv) Zwracane dane
      (1) Przekierowanie do My

j) **ReviewExists**
   i) Metoda HTTP: Nie dotyczy
   ii) Parametry:
      (1) int id
   iii) Opis
   Funkcja odpowiedzialna za sprawdzenie czy istnieje opinia o podanym ID
   iv) Zwracane dane
   Wartość logiczna true/false

2) **MoviesController**
CEL → Zarządza filmami w aplikacji

a) **Index**
   i) Metoda HTTP: GET
   ii) Parametry: Brak
   iii) Opis
   Wyświetlenie listy wszystkich filmów w aplikacji
   iv) Zwracane dane
   Widok z listą filmów (View)

b) **Details**
   i) Metoda HTTP: GET
   ii) Parametry
      (1) int? id
   iii) Opis
   Wyświetlenie szczegółów filmu o podanym ID
   iv) Zwracane dane
      (1) Szczegóły filmu (View) - jeśli istnieje
      (2) NotFound - gdy film nie istnieje

c) Create

i) Metoda HTTP: GET

ii) Parametry: Brak

iii) Opis

Wyświetlenie formularza dodawania filmu. Umożliwia wprowadzenie danych dot. tytułu, reżysera i roku produkcji

iv) Zwracane dane

Formularz dodawania filmu (View)

d) Create

i) Metoda HTTP: POST

ii) Parametry:

(1) Movie movie

iii) Opis

Odpowiada zapisanie nowego filmu do bazy danych
Białe znaki z początku i końca tytułu i autora są usuwane.
Następuje tutaj sprawdzenie, czy podobny film już istnieje w bazie.
W przypadku udanego dodania nowego filmu następuje automatyczne dodanie go do listy preferencji zalogowanego użytkownika

iv) Zwracane dane

(1) Przekierowanie do Index - w przypadku powodzenia operacji
(2) CustomErrorView (View) - gdy film już istnieje
(3) Widok tworzenia - w przypadku niepowodzenia akcji

e) ReviewExists

i) Metoda HTTP: Nie dotyczy

ii) Parametry:

(1) int id

iii) Opis

Funkcja odpowiedzialna za sprawdzenie czy istnieje film o podanym ID

iv) Zwracane dane

Wartość logiczna true/false

3) **PreferencesController**

CEL → Zarządza preferencjami użytkowników względem filmów

a) **Index**

i) Metoda HTTP: GET

ii) Parametry: Brak

iii) Opis

Wyświetlenie listy wszystkich preferencji zalogowanego użytkownika

iv) Zwracane dane

Widok z listą preferencji (View)

b) **Details**

i) Metoda HTTP: GET

ii) Parametry:

(1) int? id

iii) Opis

Wyświetlenie szczegółów preferencji.

Następuje sprawdzenie czy preferencja istnieje oraz czy należy do zalogowanego użytkownika

iv) Zwracane dane

(1) Widok z informacjami o preferencji (View)

(2) ALBO NotFound - gdy preferencja nie istnieje albo nie należy do użytkownika, który chce je wyświetlić

c) **Edit**

i) Metoda HTTP: GET

ii) Parametry:

(1) int? id

iii) Opis

Wyświetlenie formularza edycji danej preferencji

Następuje sprawdzenie czy preferencja istnieje, czy należy do zalogowanego użytkownika oraz czy istnieje film, do którego odnosi się preferencja

iv) Zwracane dane

(1) Widok z formularzem edycji preferencji (View)

(2) ALBO NotFound - gdy preferencja nie istnieje albo nie należy do użytkownika, który chce je wyświetlić

(3) ALBO CustomErrorView (View) - gdy film nie istnieje

d) Edit

i) Metoda HTTP: GET

ii) Parametry:

(1) int? id

iii) Opis

Wyświetlenie formularza edycji danej preferencji
Następuje sprawdzenie czy preferencja istnieje, czy należy do zalogowanego użytkownika oraz czy istnieje film, do którego odnosi się preferencja

iv) Zwracane dane

(1) Widok z formularzem edycji preferencji (View)
(2) ALBO NotFound - gdy preferencja nie istnieje albo nie należy do użytkownika, który chce ją edytować
(3) ALBO CustomErrorView (View) - gdy film nie istnieje

e) Edit

i) Metoda HTTP: POST

ii) Parametry:

(1) int id
(2) Preference preference

iii) Opis

Funkcja odpowiedzialna za zapisanie zmian w danej preferencji
Następuje sprawdzenie czy preferencja istnieje, czy należy do zalogowanego użytkownika oraz czy istnieje film, do którego odnosi się preferencja i czy podany typ preferencji jest dozwolony

iv) Zwracane dane

(1) Przekierowanie do Index - gdy zmiany zostały zapisane
(2) ALBO NotFound - gdy preferencja nie istnieje albo nie należy do użytkownika, który chce ją edytować
(3) ALBO CustomErrorView (View) - gdy film, którego dotyczy preferencja, nie istnieje lub gdy podano niedozwolony typ preferencji
(4) ALBO Widok edycji - gdy nie udało się zapisać zmian

f) Delete

i) Metoda HTTP: GET

ii) Parametry:

(1) int? id

iii) Opis

Wyświetlenie formularza usuwania danej preferencji

Następuje sprawdzenie czy preferencja istnieje oraz czy należy do zalogowanego użytkownika

iv) Zwracane dane

(1) Widok z formularzem potwierdzenia usuwania preferencji (View)

(2) ALBO NotFound - gdy preferencja nie istnieje albo nie należy do użytkownika, który chce ją edytować

g) DeleteConfirmed

i) Metoda HTTP: POST

ii) Parametry:

(1) int id

iii) Opis

Usunięcie danej preferencji

Następuje sprawdzenie czy preferencja istnieje oraz czy należy do zalogowanego użytkownika

iv) Zwracane dane

(1) Przekierowanie do Index

h) AddPreference

i) Metoda HTTP: POST

ii) Parametry:

(1) Preference preference

iii) Opis

Dodanie nowej preferencji

Następuje sprawdzenie czy podany typ preferencji jest dozwolony oraz czy użytkownik nie ma już dodanej preferencji w odniesieniu do danego filmu

iv) Zwracane dane

(1) Przekierowanie do Index - w przypadku udanego dodania preferencji

(2) ALBO CustomErrorView (View) - gdy typ preferencji jest niedozwolony, użytkownik posiada już preferencję powiązaną z danym filmem lub gdy wystąpił błąd i nie udało się dodać preferencji

i) PreferenceExists

i) Metoda HTTP: Nie dotyczy
ii) Parametry:
    (1) int id
iii) Opis
Funkcja odpowiedzialna za sprawdzenie czy istnieje preferencja o podanym ID
iv) Zwracane dane
Wartość logiczna true/false

4) HomeController

CEL → Obsługuje stronę główną aplikacji oraz strony błędów

a) Index

i) Metoda HTTP: GET
ii) Parametry: Brak
iii) Opis
Metoda odpowiedzialna za wyświetlenie strony głównej aplikacji - ogólnych informacji oraz przydatnych linków
iv) Zwracane dane
Widok strony głównej (View)

b) Privacy

i) Metoda HTTP: GET
ii) Parametry: Brak
iii) Opis
Metoda odpowiedzialna za wyświetlenie strony z polityką prywatności
iv) Zwracane dane
Widok polityki prywatności (View)

c) Error

i) Metoda HTTP: GET
ii) Parametry: Brak
iii) Opis
Metoda odpowiedzialna za wyświetlenie szczegółów błędów aplikacji
iv) Zwracane dane
Widokbłędówi (View)

6. System Użytkowników

1) Role w systemie

W systemie funkcjonuje podział na dwie grupy użytkowników:

a) Zalogowani użytkownicy
b) Niezalogowani użytkownicy (goście)

Nie zdefiniowano bardziej szczegółowych ról w systemie, takich jak administrator czy moderator. Funkcjonalność zależy jedynie od stanu logowania użytkownika.

2) Możliwości użytkowników zalogowanych w odróżnieniu od gości

→ Niezalogowani użytkownicy mogą:

a) Przeglądać listę dostępnych filmów.
b) Wyświetlać szczegóły dot. filmów
c) Wyświetlać listę opinii
d) Wyświetlać szczegóły opinii

→ Zalogowani użytkownicy mogą:

a) Wykonywać wszystkie czynności dozwolone dla gości
b) Dodawać nowe filmy do publicznej bazy
c) Zarządzać listą swoich preferencji dot. filmów
    i) Wyświetlać listę
    ii) Wyświetlać szczegóły konkretnych preferencji
    iii) Dodawać filmy do swojej listy preferencji, wraz z oznaczeniem ich jako *„do obejrzenia”* lub *„obejrzane”*.
    iv) Edytować swoje preferencje zmieniając ich oznaczenie
    v) Usuwać swoje preferencje
d) Wyświetlać listę swoich opinii
e) Dodawać opinie do filmów, zawierające zarówno ocenę (w skali od 1 do 5), jak i treść tekstową (recenzję).
f) Edytować swoje opinie dot. filmów oraz dokonywać ich usunięcia

3) Ograniczenia

a) Próba dostępu niezalogowanego użytkownika do funkcji zarezerwowanych dla użytkowników zalogowanych przekierowuje go na stronę logowania.

4) **Powiązane informacje**
    a) Dane użytkownika są obsługiwane przez bibliotekę ASP.NET Identity, co zapewnia gotowe mechanizmy rejestracji, logowania i zarządzania tożsamością.
    b) Do użytkowników przypisane są:
        i) Filmy - każdy użytkownik dodając film do publicznej bazy staje się autorem konkretnego wpisu
        ii) Preferencje - każdy użytkownik tworzy swoją listę preferencji na podstawie wszystkich dostępnych filmów.
        iii) Opinie - informacje o recenzjach i ocenach są również przypisane do konkretnego zalogowanego użytkownika, co pozwala na ich późniejszą modyfikację lub usunięcie.

7) Krótka charakterystyka najciekawszych funkcjonalności

1) **Spersonalizowana strona z błędami**
   W ramach aplikacji dodana została obsługa błędów w ramach spersonalizowanej strony błędu, wyświetlającej stosowny komunikat, informację o przyczynie niepowodzenia akcji oraz umożliwiającej powrót w odpowiednie miejsce w aplikacji.
   Mimo możliwości zablokowania przeniesienia na tę stronę poprzez zastosowanie odpowiednich filtrowań, niektóre funkcjonalności zostały celowo na nią skierowane, w celu wykorzystania i zaobserwowania w prosty sposób.