

Tytuł projektu:

System rezerwacji seansów w kinie studyjnym Cinematrix

Krótki opis działania projektu:

Aplikacja umożliwia użytkownikom kina przeglądanie dostępnych seansów, rezerwowanie miejsc na seanse oraz przeglądanie aktywnych rezerwacji. Administratorzy mogą zarządzać harmonogramem seansów, dodawać i usuwać seanse, a także monitorować obłożenie. System posiada podział na role użytkowników np. obserwator, użytkownik standardowy, administrator.

Autorzy projektu:

**Kacper Białowås
Kamil Markiewicz**

1) Specyfikacja wykorzystanych technologii:

W projekcie zastosowano technologię C# .NET 8 w połączeniu z bazą danych SQL Server, co zapewnia wydajną i bezpieczną obsługę danych.

Do komunikacji z bazą danych użyliśmy technologii ORM Entity Framework, co umożliwia mapowanie obiektów na tabele bazy danych i upraszcza pracę z danymi dzięki operacjom CRUD realizowanym bezpośrednio na modelach aplikacji.

W projekcie zastosowano Entity Framework w celu zwiększenia efektywności pracy z bazą danych i zmniejszenia ilości ręcznie pisanych zapytań SQL.

W projekcie zastosowano podejście *code-first* Entity Framework, które umożliwia definiowanie struktury bazy danych bezpośrednio na podstawie klas modelowych w kodzie aplikacji.

Podejście *code-first* pozwala na automatyczne generowanie schematu bazy danych z poziomu kodu dzięki migracjom, które są uruchamiane za pomocą komendy `update-database` w Menedżerze NuGet lub `dotnet ef database update` w terminalu.

Zastosowanie podejścia *code-first* upraszcza proces tworzenia i utrzymywania struktury bazy danych, umożliwiając łatwe aktualizacje poprzez dodawanie zmian w kodzie oraz ich synchronizację z bazą danych.

Mechanizm migracji Entity Framework pozwala na śledzenie zmian w modelach danych i aktualizowanie struktury bazy danych bez konieczności ręcznego tworzenia skryptów SQL.

2) Instrukcja pierwszego uruchomienia:

1. Wybrany folder na komputerze należy otworzyć w terminalu.
2. W terminalu należy wpisać komendę:
`git clone https://github.com/melonkamil/cinematix.git`
3. W powstałym folderze należy uruchomić plik *Cinematix.sln*.
4. Po uruchomieniu Visual Studio należy w *Package Manager Console* wykonać komendę *update-database*.
5. Po wykonaniu powyższych kroków można już skompilować projekt.
6. W oknie przeglądarki otworzy się aplikacja do przeglądania, zarządzania i rezerwacji seansów w kinie Cinematix.
7. Baza danych jest pusta, należy zalogować się kontem administratora
`admin@cinematix.com`
`P@ssw0rd`
celem dodania nowych filmów i seansów.
8. Dodając film należy podać tytuł, opis oraz link URL do plakatu.
9. Dodając seans należy wybrać film, podać datę oraz godzinę zgodnie z podanym schematem w odpowiedzi:
-data RRRR-MM-DD
-godzina GG:MM

3) Opis struktury projektu:

a) Modele w projekcie:

1. ApplicationUser

- Pełni rolę rozszerzenia domyślnego modelu użytkownika w systemie uwierzytelniania. Jest częścią mechanizmu ASP.NET Core Identity, który zapewnia gotowe narzędzia do obsługi logowania, rejestracji, zarządzania rolami użytkowników itp.

- Pola:

`public virtual TKey Id { get; set; } = default!;`

unikalny identyfikator użytkownika

`public virtual string? Email { get; set; }`

adres e-mail

`public virtual string? PasswordHash { get; set; }`

hasz hasła

2. Movie

- Reprezentuje model danych dla filmów w aplikacji. Jest to klasa, która definiuje strukturę informacji, jakie aplikacja przechowuje na temat filmów, np. w bazie danych. Służy ona do przechowywania i zarządzania

danymi o filmach, które mogą być wyświetlane użytkownikom.

- Pola:

```
public string Id { get; set; }
```

to unikalny identyfikator filmu w bazie danych

```
public string Title { get; set; }
```

to unikatowa nazwa filmu w systemie, posiada ograniczenie do 200 znaków

```
public string Description { get; set; }
```

służy do przechowywania dodatkowych informacji opisowych o danym filmie, posiada ograniczenie do 1000 znaków

```
public string ImageUrl { get; set; }
```

służy do załączenia obrazu plakatu filmu z sieci, dzięki czemu nie ma potrzeby przechowywania dużego pliku w bazie danych, pole posiada ograniczenie do 500 znaków

```
public bool IsDeleted { get; set; }
```

znacznik logiczny na potrzeby miękkiego usunięcia filmu z bazy danych, a dokładnie do usuwania z widoku na stronie rezerwacji

```
public DateTime CreatedAt { get; set; }
```

znacznik czasowy na potrzeby przechowania informacji o dacie i godzinie utworzenia wpisu o filmie

3. Reservation

- Służy do reprezentowania rezerwacji miejsca na seans filmowy. Jest to model danych opisujący szczegóły rezerwacji dokonanej przez użytkownika, takie jak użytkownik, wybrany seans, koszt rezerwacji oraz czas jej utworzenia.

- Pola:

```
public string Id { get; set; }
```

to unikalny identyfikator rezerwacji w bazie danych

```
public string UserId { get; set; }
```

to unikalny identyfikator użytkownika w bazie danych

```
public string ShowtimeId { get; set; }
```

to unikalny identyfikator seansu w bazie danych

```
public int Cost { get; set; }
```

służy do wyliczania sumarycznego kosztu rezerwacji

```
public DateTime CreatedAt { get; set; }
```

znacznik czasowy na potrzeby przechowania informacji o dacie i godzinie utworzenia rezerwacji

```
public Showtime Showtime { get; set; }
```

służy do przechowania informacji o seansie, którego dotyczy rezerwacja

4. ReservedSeat

- Służy do reprezentowania informacji o konkretnym zarezerwowanym miejscu na dany seans filmowy. W aplikacji jest ona wykorzystywana do przechowywania danych dotyczących tego, które miejsca na danym seansie zostały zarezerwowane przez użytkowników.

- Pola:

```
public string Id { get; set; }
```

to unikalny identyfikator miejsca w bazie danych

```
public string ShowtimeId { get; set; }
```

to unikalny identyfikator seansu w bazie danych

```
public string ReservationId { get; set; }
```

to unikalny identyfikator rezerwacji w bazie danych

```
public string SeatNumber { get; set; }
```

to numer miejsca wskazywanego przez użytkownika

```
public DateTime CreatedAt { get; set; }
```

znacznik czasowy na potrzeby przechowania informacji o dacie i godzinie

5. Showtime

- Służy do reprezentowania konkretnego seansu filmowego w systemie rezerwacji biletów. W aplikacji jest używana do przechowywania informacji o dacie, godzinie oraz filmie wyświetlanym na danym seansie. Dzięki tej klasie można zarządzać harmonogramem wyświetlania filmów.

- Pola:

```
public string Id { get; set; }
```

to unikalny identyfikator seansu w bazie danych

```
public string MovieId { get; set; }
```

to unikalny identyfikator filmu w bazie danych

```
public string Date { get; set; }
```

służy do przechowania informacji o dacie seansu

```
public string Time { get; set; }
```

służy do przechowania informacji o godzinie rozpoczęcia seansu

```
public bool IsDeleted { get; set; }
```

znacznik logiczny na potrzeby miękkiego usunięcia seansu z bazy danych, a dokładnie do usuwania z widoku na stronie rezerwacji

```
public DateTime CreatedAt { get; set; }
```

znacznik czasowy na potrzeby przechowania informacji o dacie i godzinie utworzenia seansu

```
public Movie Movie { get; set; }
```

służy do przechowania informacji o filmie jakiego dotyczy seans

6. ReservationView

- Pełni rolę modelu widoku (ViewModel) w aplikacji. Służy do przekazywania danych dotyczących rezerwacji z warstwy serwisowej lub kontrolera do widoku (interfejsu użytkownika). Umożliwia prezentację informacji o rezerwacji w formacie przyjaznym dla użytkownika, łącząc dane z różnych tabel/modeli.

- Pola:

```
public string ReservationId { get; set; }
```

służy do wyświetlenia identyfikatora rezerwacji

```
public string MovieTitle { get; set; }
```

służy do wyświetlenia tytułu filmu rezerwacji

```
public string Date { get; set; }
```

służy do wyświetlenia daty rezerwacji

```
public string Time { get; set; }
```

służy do wyświetlenia godziny rezerwacji

```
public int TotalCost { get; set; }
```

służy do wyświetlenia sumarycznego kosztu rezerwacji

```
public string[] Seats { get; set; }
```

służy do wyświetlenia zarezerwowanych miejsc

b) Strony: (URL)

1. AddMovie ("/add-movie")

Strona dostępna tylko dla administratora dzięki

```
@attribute [Authorize(Policy = "ManageMoviesPolicy")]
```

Służy do dodawania nowego filmu w aplikacji. Komponent obsługuje formularz zawierający pola: tytuł filmu, opis oraz adres URL obrazu, a także zapewnia walidację danych i wyświetlanie komunikatów o błędach. Dodatkowo, po poprawnym wypełnieniu formularza, użytkownik musi potwierdzić dodanie filmu w oknie pop-up.

Metody:

- Metoda: `HandleSubmit`

- HTTP: POST

- Parametry: brak (dane pobierane z formularza `form`)

- Opis: Obsługuje przesłanie formularza dodawania filmu.

Waliduje dane wejściowe i otwiera okno potwierdzenia, jeśli dane są poprawne.

- Zwracane dane: brak odpowiedzi bezpośredniej — wyświetla pop-up.

- Metoda: `ConfirmAdd`

- HTTP: POST

- Parametry:

- `form.Title` (string) – tytuł filmu.

- `form.Description` (string) – opis filmu.

- `form.ImageUrl` (string) – URL plakatu filmu.

- Opis: Potwierdza dodanie filmu. Wysyła dane do serwisu `MovieService.AddMovie`, obsługuje błędy i resetuje formularz po dodaniu.

- Zwracane dane: przekierowanie na stronę `/error/{ErrorCode}` w przypadku błędu.

- Metoda: ``CancelAdd``
 - HTTP: brak (wywołanie lokalne)
 - Parametry: brak
 - Opis: Anuluje proces dodawania filmu, zamyka okno potwierdzenia.
 - Zwracane dane: brak.
-
- Metoda: ``ResetForm``
 - HTTP: brak (wywołanie lokalne)
 - Parametry: brak
 - Opis: Resetuje formularz i czyści błędy walidacji.
 - Zwracane dane: brak.
-
- Metoda: ``ValidateForm``
 - HTTP: brak (wywołanie lokalne)
 - Parametry:
 - ``MovieForm form`` (obiekt) – zawiera dane tytułu, opisu i URL plakatu.
 - Opis: Sprawdza poprawność danych wejściowych, np. długość tytułu, format URL.
 - Zwracane dane: obiekt ``ValidationErrors`` z błędami walidacji, jeśli wystąpią.

2. AddShowtime ("/add-showtime")

Strona dostępna tylko dla administratora dzięki

`@attribute [Authorize(Policy = "ManageMoviesPolicy")]`

Służy do dodawania nowego seansu filmowego. Pozwala na wybranie filmu z listy dostępnych tytułów, a następnie podanie daty i godziny seansu. Komponent obsługuje walidację danych, wyświetlanie komunikatów o błędach oraz potwierdzenie operacji przed dodaniem seansu do systemu.

Metody:

- Metoda: ``HandleSubmit``

- HTTP: POST

- Parametry: brak (dane pobierane z formularza `form`)

- Opis: Obsługuje przesłanie formularza dodawania seansu. Waliduje dane i wyświetla okno potwierdzenia, jeśli dane są poprawne.

- Zwracane dane: brak odpowiedzi bezpośredniej — wyświetla pop-up.

- Metoda: `ConfirmAdd`

- HTTP: POST

- Parametry:

- `form.MovieId` (string) – identyfikator wybranego filmu.

- `form.Date` (string) – data seansu w formacie `yyyy-MM-dd`.

- `form.Time` (string) – godzina seansu w formacie `HH:mm`.

- Opis: Potwierdza dodanie seansu. Wysyła dane do serwisu `MovieService.AddShowtime`, obsługuje błędy i resetuje formularz po dodaniu.

- Zwracane dane: przekierowanie na stronę `/error/{ErrorCode}` w przypadku błędu.

- Metoda: `CancelAdd`

- HTTP: brak (wywołanie lokalne)

- Parametry: brak

- Opis: Anuluje proces dodawania seansu, zamyka okno potwierdzenia.

- Zwracane dane: brak.

- Metoda: `ResetForm`

- HTTP: brak (wywołanie lokalne)

- Parametry: brak

- Opis: Resetuje formularz i czyści błędy walidacji.

- Zwracane dane: brak.

- Metoda: ``ValidateForm``

- HTTP: brak (wywołanie lokalne)

- Parametry:

- ``ShowtimeForm form`` (obiekt) – zawiera dane identyfikatora filmu, daty i godziny seansu.

- Opis: Sprawdza poprawność danych wejściowych, np. format daty, format godziny i wybór filmu.

- Zwracane dane: obiekt ``ValidationErrors`` z błędami walidacji, jeśli wystąpią.

- Metoda: ``OnInitializedAsync``

- HTTP: GET

- Parametry: brak

- Opis: Inicjalizuje komponent, pobierając listę dostępnych filmów z serwisu ``MovieService.GetMovies``.

- Zwracane dane: lista filmów do wyboru w formularzu.

3. DeleteMovie ("/delete-movie")

Strona dostępna tylko dla administratora dzięki

`@attribute [Authorize(Policy = "ManageMoviesPolicy")]`

Umożliwia usuwanie filmu wraz z jego seansami z bazy danych.

Użytkownik może wybrać film z listy, a następnie po zatwierdzeniu usunąć go z systemu. Komponent zapewnia walidację danych, wyświetlanie komunikatów o błędach oraz okno potwierdzenia przed usunięciem filmu.

Metody:

- Metoda: ``HandleSubmit``

- HTTP: POST

- Parametry: brak (dane pobierane z ``movieId`` wybranego w formularzu)

- Opis: Obsługuje przesłanie formularza usuwania filmu. Waliduje dane wejściowe i wyświetla okno potwierdzenia, jeśli wybrano film.

- Zwracane dane: brak odpowiedzi bezpośredniej — wyświetla pop-up.

- Metoda: ``ConfirmAdd``

- HTTP: DELETE

- Parametry:

- ``movieId`` (string) – identyfikator filmu do usunięcia.

- Opis: Potwierdza usunięcie filmu. Wywołuje metodę ``DeleteMovieAndShowtimes`` w serwisie ``MovieService``, która usuwa film oraz powiązane seanse. Obsługuje błędy i odświeża listę filmów po usunięciu.

- Zwracane dane: przekierowanie na stronę ``/error/{ErrorCode}`` w przypadku błędu.

- Metoda: ``CancelAdd``

- HTTP: brak (wywołanie lokalne)

- Parametry: brak

- Opis: Anuluje proces usuwania filmu, zamyka okno potwierdzenia.

- Zwracane dane: brak.

- Metoda: ``ResetForm``

- HTTP: brak (wywołanie lokalne)

- Parametry: brak

- Opis: Resetuje formularz i czyści błędy walidacji.

- Zwracane dane: brak.

- Metoda: ``ValidateForm``

- HTTP: brak (wywołanie lokalne)
- Parametry:
 - `movieId` (string) – identyfikator filmu.
 - Opis: Sprawdza, czy użytkownik wybrał film do usunięcia oraz czy `movieId` ma poprawny format GUID.
 - Zwracane dane: obiekt `ValidationErrors` zawierający błędy walidacji.
- Metoda: `OnInitializedAsync`
- HTTP: GET
- Parametry: brak
- Opis: Inicjalizuje komponent i pobiera listę dostępnych filmów poprzez metodę `LoadMovies`.
- Zwracane dane: lista filmów do wyświetlenia w formularzu.
- Metoda: `LoadMovies`
- HTTP: GET
- Parametry: brak
- Opis: Pobiera listę wszystkich filmów za pomocą `MovieService.GetMovies`, aby umożliwić wybór filmu do usunięcia.
- Zwracane dane: lista filmów.

4. Error ("/Error")

Strona dostępna dla każdego, również niezalogowanego.

Domyślnie utworzona w projekcie strona, która pełni rolę strony błędu, jest wyświetlana, gdy w aplikacji wystąpi nieoczekiwany błąd. Jest to strona obsługi błędów, która przekazuje użytkownikowi podstawowe informacje o błędzie w bezpieczny i czytelny sposób, unikając wyświetlania szczegółów technicznych, które mogłyby ujawniać wrażliwe informacje.

Metody:

- Metoda: `OnInitialized`

- HTTP: GET

- Parametry: brak

- Opis: Inicjalizuje komponent i przypisuje identyfikator żądania ``RequestId``. Jeśli ``Activity.Current?.Id`` jest dostępne, używa tego identyfikatora; w przeciwnym razie korzysta z ``HttpContext?.TraceIdentifier``.

- Zwracane dane: identyfikator żądania ``RequestId``, który jest używany do wyświetlenia informacji o żądaniu w widoku.

- Właściwość: ``ShowRequestId``

- HTTP: brak (właściwość lokalna)

- Parametry: brak

- Opis: Określa, czy identyfikator ``RequestId`` jest ustawiony (czyli czy powinien zostać wyświetlony w widoku).

- Zwracane dane: ``true``, jeśli ``RequestId`` nie jest ``null`` ani pusty; ``false`` w przeciwnym razie.

5. Index ("/")

Strona dostępna dla każdego, również niezalogowanego.

Pełni rolę strony głównej aplikacji wyświetlającej listę dostępnych filmów oraz godzin seansów dla wybranego dnia. Użytkownik może przetaczać dni tygodnia, aby zobaczyć repertuar na wybrany dzień, kliknąć na film, aby zobaczyć szczegóły, lub wybrać godzinę seansu, aby przejść do rezerwacji biletu.

Metody:

- Metoda: ``OnInitializedAsync``

- HTTP: GET

- Parametry: brak

- Opis: Inicjalizuje stronę główną. Pobiera listę dni tygodnia (od dzisiaj do 7 dni do przodu) oraz listę filmów na dzisiejszy dzień. Pobiera stan uwierzytelnienia użytkownika.

- Zwracane dane: lista dostępnych dni oraz lista filmów z pokazami na dzisiaj.

- Metoda: ``ShowMovies``

- HTTP: GET

- Parametry:

- `Day day` (obiekt) – wybrany dzień z listy dni tygodnia.
- Opis: Ustawia wybrany dzień jako aktywny i ładuje listę filmów dla tego dnia.
- Zwracane dane: lista filmów z pokazami dla wybranego dnia.

- Metoda: `GetMoviesForDay`
 - HTTP: GET
 - Parametry:
 - `string date` – data w formacie `yyyy-MM-dd`.
 - Opis: Pobiera z serwisu listę seansów dla podanej daty i grupuje je według filmów.
 - Zwracane dane: lista filmów zawierających szczegóły pokazów (tytuł, opis, godziny seansów).

- Metoda: `FetchMoviesForDay`
 - HTTP: GET
 - Parametry:
 - `string date` – data w formacie `yyyy-MM-dd`.
 - Opis: Wywołuje `GetMoviesForDay` i przypisuje wyniki do listy `movieList`.
 - Zwracane dane: lista filmów dla danej daty.

- Metoda: `ReserveShowtime`
 - HTTP: GET
 - Parametry:
 - `ShowtimeHour showtimeHour` (obiekt) – szczegóły seansu (identyfikator i godzina).
 - Opis: Przekierowuje użytkownika na stronę rezerwacji seansu, jeśli jest zalogowany. W przeciwnym razie przekierowuje na stronę logowania.
 - Zwracane dane: brak.

- Metoda: `GoToMovieDetails`
 - HTTP: GET
 - Parametry:
 - `string movieId` – identyfikator wybranego filmu.
 - Opis: Przekierowuje użytkownika na stronę szczegółów wybranego filmu.
 - Zwracane dane: brak.

- Metoda: `GetDayButtonLabel`

- HTTP: brak (metoda lokalna)
 - Parametry:
 - `Day day` (obiekt) – zawiera datę i nazwę dnia tygodnia.
 - Opis: Formatuje etykietę przycisku dnia (np. „Mon 4.12”).
 - Zwracane dane: sformatowany tekst etykiety dnia.
-
- Metoda: `GroupByMovie`
 - HTTP: brak (metoda lokalna)
 - Parametry:
 - `List<Data.Showtime> showtimes` – lista seansów dla danego dnia.
 - Opis: Grupuje seanse według filmów i tworzy listę obiektów `Movie`, zawierających godziny seansów.
 - Zwracane dane: lista obiektów `Movie` z pokazami.

6. Issue ("/error/{ErrorCode}")

Strona dostępna dla każdego, również niezalogowanego.

Utworzona na rzecz projektu strona, podobna do „Error”, która pełni rolę wyświetlania znanych, spodziewanych błędów z logiki biznesowej.

Na tę stronę zostaje przeniesiony użytkownik w sytuacji wystąpienia błędu poziomu biznesowego, np.: chcąc dodać film, który już istnieje lub w momencie wystąpienia błędów logiki biznesowej, np.: kiedy dodając seans zostanie użyty nieprawidłowy pattern daty lub godziny.

Metody:

- Metoda: `OnInitializedAsync`
- HTTP: GET
- Parametry:
 - `string ErrorCode` (parametr URL) – kod błędu przekazany w adresie URL.
- Opis: Inicjalizuje stronę błędu. Na podstawie kodu błędu (`ErrorCode`) pobiera komunikat błędu z rejestru `ErrorCodeRegistry`. Jeśli kod błędu nie jest obsługiwany, wyświetla domyślny komunikat "An unhandled error has occurred".
- Zwracane dane: komunikat błędu `errorMessage`, wyświetlany na stronie.

7. MakeReservation ("/make-reservation/{ShowtimeId}")

Strona dostępna tylko dla zalogowanego użytkownika dzięki

@attribute [Authorize]

Umożliwia użytkownikowi rezerwację miejsc na wybrany seans filmowy. Po wejściu na stronę /make-reservation/{ShowtimeId} użytkownik widzi układ sali kinowej, wybiera miejsca, a następnie potwierdza rezerwację. Komponent obsługuje walidację wybranych miejsc, wyświetla łączny koszt biletów i umożliwia potwierdzenie lub anulowanie zakupu.

Metody:

- Metoda: ``OnInitializedAsync``
 - HTTP: GET
 - Parametry:
 - ``string ShowtimeId`` (parametr URL) – identyfikator seansu.
 - Opis: Inicjalizuje stronę rezerwacji. Pobiera listę już zarezerwowanych miejsc na dany seans oraz cenę biletu.
 - Zwracane dane: lista zarezerwowanych miejsc ``reservedSeats``, cena biletu ``ticketPrice``.

- Metoda: ``GetSeat``
 - HTTP: brak (metoda lokalna)
 - Parametry:
 - ``int col`` – kolumna miejsca.
 - ``int row`` – rząd miejsca.
 - Opis: Generuje oznaczenie miejsca w formacie "rząd + numer", np. ``A5``.
 - Zwracane dane: sformatowany string oznaczenia miejsca.

- Metoda: ``GetSeatClass``
 - HTTP: brak (metoda lokalna)
 - Parametry:
 - ``string seat`` – oznaczenie miejsca.
 - Opis: Zwraca klasę CSS miejsca w zależności od jego statusu (zajęte, wybrane, dostępne).

- Zwracane dane: string z klasą CSS (` reserved` , ` selected` , ` available`).

- Metoda: ` SelectSeat`

- HTTP: brak (metoda lokalna)

- Parametry:

- ` string seat` – oznaczenie miejsca.

- Opis: Dodaje miejsce do wybranych lub usuwa je z listy wybranych, jeśli użytkownik kliknął na nie ponownie.

- Zwracane dane: brak.

- Metoda: ` SubmitPurchase`

- HTTP: brak (metoda lokalna)

- Parametry: brak

- Opis: Wywołuje okno potwierdzenia zakupu miejsc.

- Zwracane dane: brak.

- Metoda: ` ConfirmPurchase`

- HTTP: POST

- Parametry:

- ` string Showtimeld` – identyfikator seansu.

- ` string[] selectedSeats` – tablica wybranych miejsc.

- ` string user.Identity.Name` – nazwa użytkownika (login).

- Opis: Potwierdza rezerwację. Wysyła dane do serwisu ` MovieService.MakeReservation` , obsługuje błędy i przekierowuje na stronę z rezerwacjami użytkownika.

- Zwracane dane: przekierowanie na ` /my-reservations` lub ` /error/{ErrorCode}` w przypadku błędu.

- Metoda: ` CancelPurchase`

- HTTP: brak (metoda lokalna)
- Parametry: brak
- Opis: Anuluje proces zakupu, zamykając okno potwierdzenia.
- Zwracane dane: brak.

- Metoda: ``FormatPrice``

- HTTP: brak (metoda lokalna)

- Parametry:

- ``int price`` – cena w groszach.

- Opis: Formatuje cenę na dwie cyfry po przecinku i zamienia kropkę na przecinek.

- Zwracane dane: string z ceną w formacie ``xx,xx PLN``.

8. MovieDetails ("/movie-details/{MovieId}")

Strona dostępna dla każdego, również niezalogowanego.

Służy do wyświetlania szczegółów wybranego filmu. Po wejściu na stronę /movie-details/{MovieId}, gdzie MovieId to identyfikator filmu, użytkownik widzi tytuł, plakat oraz opis filmu.

Metody:

- Metoda: ``OnInitializedAsync``

- HTTP: GET

- Parametry:

- ``string MovieId`` (parametr URL) – identyfikator filmu.

- Opis: Inicjalizuje stronę szczegółów filmu. Pobiera dane filmu z serwisu ``MovieService.FindMovie`` na podstawie ``MovieId``. Jeśli film nie zostanie znaleziony, przekierowuje użytkownika na stronę błędu.

- Zwracane dane: obiekt ``Movie`` zawierający tytuł, opis i URL plakatu filmu lub przekierowanie na ``/error/{ErrorCode}`` w przypadku błędu.

9. MyReservations ("/my-reservations")

Strona dostępna tylko dla zalogowanego użytkownika dzięki

`@attribute [Authorize]`

wyświetla listę aktywnych rezerwacji użytkownika. Po zalogowaniu użytkownik może przejść na stronę /my-reservations, aby zobaczyć szczegóły swoich aktualnych rezerwacji, takich jak tytuł filmu, data i

godzina seansu, cena oraz wybrane miejsca.

Metody:

- Metoda: ``OnInitializedAsync``

- HTTP: GET

- Parametry: brak

- Opis: Inicjalizuje stronę rezerwacji użytkownika. Pobiera stan uwierzytelnienia i na jego podstawie pobiera listę aktywnych rezerwacji użytkownika z serwisu

- ``MovieService.GetUserActiveReservations``.

- Zwracane dane: lista obiektów ``ReservationView``, zawierająca informacje o rezerwacjach (tytuł filmu, ID rezerwacji, data, godzina, cena całkowita, miejsca).

- Metoda: ``FormatPrice``

- HTTP: brak (metoda lokalna)

- Parametry:

- ``int price`` – cena w groszach.

- Opis: Formatuje cenę całkowitą rezerwacji na format ``xx,xx PLN``, zamieniając kropkę na przecinek.

- Zwracane dane: string sformatowanej ceny.

4) Opis systemu użytkowników

W naszym projekcie wykorzystaliśmy wbudowany mechanizm uwierzytelniania i autoryzacji oferowany przez platformę Microsoft.

Dzięki użyciu konfiguracji

```
builder.Services.AddIdentityCore<ApplicationUser>()
```

oraz

```
AddRoles<IdentityRole>()
```

zarejestrowaliśmy usługę Identity, która obsługuje rejestrację, logowanie i zarządzanie użytkownikami oraz rolami.

Ustawienie

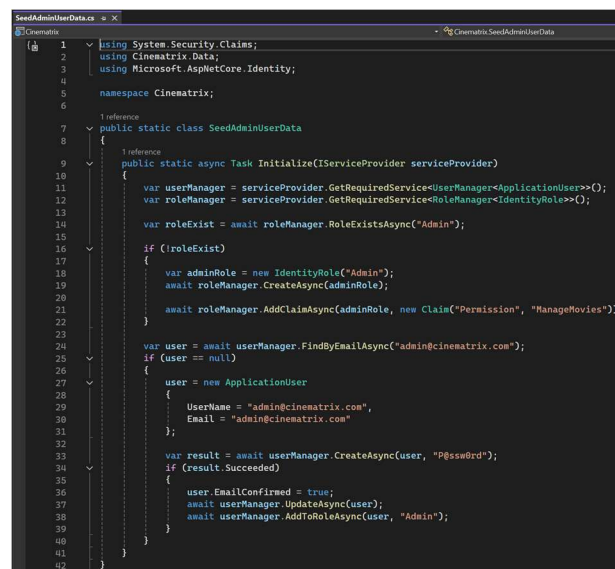
```
options.SignIn.RequireConfirmedAccount = true
```

dodatkowo wymusza, aby użytkownik przed zalogowaniem potwierdził swoje konto, co zwiększa bezpieczeństwo procesu logowania. W efekcie mamy kompleksowe wsparcie dla operacji związanych z tożsamością użytkowników, bez potrzeby implementowania tych mechanizmów od podstaw.

Stworzyliśmy politykę autoryzacji opartą na claimach, co pozwala na precyzyjne kontrolowanie dostępu do stron i operacji w aplikacji. Dzięki temu tylko użytkownicy posiadający odpowiednie uprawnienia mogą np. zarządzać stroną czy wykonywać działania administracyjne. Mechanizm ten zwiększa bezpieczeństwo i umożliwia przypisywanie ról oraz praw dostępu zgodnie z indywidualnymi uprawnieniami użytkowników.

```
builder.Services.AddAuthorization(options =>
{
options.AddPolicy("ManageMoviesPolicy", policy =>
policy.RequireClaim("Permission", "ManageMovies"));
});
```

Podczas uruchamiania systemu automatycznie tworzy się rola Admin, do której przypisywane jest uprawnienie "manage movies". Następnie to uprawnienie jest powiązane z rolą Admin, a konto użytkownika admin@cinematrix.com jest tworzone i przypisywane do tej roli. Cała ta logika jest zaimplementowana w pliku SeedAdminUserData.cs. Jeśli konto admin@cinematrix.com zostanie usunięte, system przy kolejnym uruchomieniu odtworzy je ponownie.



```
1 using System.Security.Claims;
2 using Cinematrix.Data;
3 using Microsoft.AspNetCore.Identity;
4
5 namespace Cinematrix;
6
7 public static class SeedAdminUserData
8 {
9     public static async Task Initialize(IServiceProvider serviceProvider)
10     {
11         var userManager = serviceProvider.GetRequiredService<UserManager<ApplicationUser>>();
12         var roleManager = serviceProvider.GetRequiredService<RoleManager<IdentityRole>>();
13
14         var roleExist = await roleManager.RoleExistsAsync("Admin");
15
16         if (!roleExist)
17         {
18             var adminRole = new IdentityRole("Admin");
19             await roleManager.CreateAsync(adminRole);
20             await roleManager.AddClaimAsync(adminRole, new Claim("Permission", "ManageMovies"));
21         }
22
23         var user = await userManager.FindByEmailAsync("admin@cinematrix.com");
24         if (user == null)
25         {
26             user = new ApplicationUser
27             {
28                 UserName = "admin@cinematrix.com",
29                 Email = "admin@cinematrix.com"
30             };
31
32             var result = await userManager.CreateAsync(user, "P@ssw0rd");
33             if (result.Succeeded)
34             {
35                 user.EmailConfirmed = true;
36                 await userManager.UpdateAsync(user);
37                 await userManager.AddToRoleAsync(user, "Admin");
38             }
39         }
40     }
41 }
42
```

Każdy nowo zarejestrowany użytkownik otrzymuje domyślnie status zwykłego użytkownika bez dodatkowych uprawnień. Uprawnienia mogą być nadane takim użytkownikom wyłącznie ręcznie, poprzez edycję danych w bazie.

Gość (niezalogowany użytkownik) ma jedynie możliwość przeglądania seansów. W momencie wyboru godziny seansu, aplikacja automatycznie przekierowuje go do strony logowania lub rejestracji. Bez zalogowania się lub założenia konta nie można przejść do rezerwacji ani obejść procesu w celu dokonania rezerwacji anonimowo. Taki mechanizm zapobiega nieautoryzowanym operacjom i zabezpiecza system przed próbami obejścia ścieżki autoryzacji.

5) Krótka charakterystyka najciekawszych funkcjonalności

- a) Make reservation - posiada siatkę przedstawiającą wizualny rozkład siedzeń w kinie, które informują użytkownika o dostępnych miejscach. Wolne miejsca są widoczne w kolorze niebieskim, wybrane przez użytkownika są podświetlone na zielono, a zajęte przez innych widnieją na szaro oraz nie są dostępne do zaznaczenia.
- b) Delete movie and showtimes – funkcja która przy usunięciu wybranego filmu w repertuarze usuwa również wszystkie seanse tego filmu co pozwala na szybkie zarządzanie bez wymogu usuwania każdego seansu z osobna.
- c) Get user active reservation – funkcja która pobiera dane o rezerwacjach zalogowanego użytkownika i wyświetla tylko je w podglądzie rezerwacji przez co każdy użytkownik widzi tylko rezerwacje wykonane przez jego konto i nie może ingerować w inne.