AKADEMIA NAUK STOSOWANYCH W TARNOWIE

Wydział Politechniczny

Kierunek: Informatyka

Specjalność: Informatyka stosowana

Specjalizacja: Inżynieria oprogramowania

2022/2023

Oliwer Kucharzyk

Praca inżynierska

Kino samoobsługowe

Promotor pracy:

mgr inż. Tomasz Gądek

Tarnów, 2023

Spis treści

Składam serdeczne podziękowania

Panu mgr inż. Tomaszowi Gądkowi

za merytoryczne uwagi udzielane

podczas pisania tej pracy.

1. Wstęp

Pierwszy kinematograf[1] opracowali we Francji bracia Lumiere niespełna 130 lat temu[2]. Publiczne projekcje szybko stały się popularne, chociaż sami twórcy początkowo nie spodziewali się, że ich wynalazek zostanie powszechnie przyjęty[3]. Mimo, że sama kinematografia[4] zmieniała się na przestrzeni lat, koncept kina jako wyrobu komercyjnego pozostał taki sam. Współczesny widz stawia oczekiwania nie tylko oglądanej produkcji, ale również procesowi, który zapewnia mu dostęp do tej rozrywki. W tym celu zaimplementowano system, który usprawnia proces zakupu biletu na seans pozwalając klientowi wygenerować oraz skasować swój bilet za pomocą aplikacji mobilnej - bez konieczności udziału obsługi kina.

1.1. Cel pracy

Przedmiotem pracy jest projekt oraz implementacja systemu usprawniającego proces zakupu biletów do kina poprzez generowanie biletu w formie kodu QR, który następnie jest skanowany na automatycznych bramkach w kinie. System zostanie zintegrowany z platformą testową jednego z dostępnych dostawców usług płatniczych na rynku.

1.2. Zakres pracy

W ramach projektu wdrożono system składający się z następujących modułów:

- aplikacji serwerowej,
- aplikacji klienckiej dla użytkownika,
- skryptu obsługującego kasowanie biletów,
- bazy danych,
- dostawcy usług płatniczych.

[1] aparat służący do projekcji ruchomych obrazów

[2] (“Jak bracia Lumiére opatentowali kinematograf | Portal historyczny Histmag.org - historia dla każdego!”)

[3] (“Kinematograf, czyli początki kina. Krótka historia wynalazku i ciekawostki”)

[4] dziedzina związana z całokształtem wytwarzania filmów

2. Stos technologiczny

Wybór technologii do przygotowania systemu samoobsługowego kina podyktowany był popularnością, łatwością integracji oraz przede wszystkim znajomością poszczególnych narzędzi. Wyjątkiem jest tutaj .NET MAUI[5], czyli technologia wykorzystana do opracowania aplikacji klienckiej, która została oficjalnie wydana w maju 2022 roku i jest wciąż rozwijana przez firmę Microsoft, przez co nie zyskała jeszcze reprezentatywnego grona odbiorców.

2.1. Aplikacja serwerowa

Aplikacja serwerowa została przygotowana w formie WebAPI, na ogół z wykorzystaniem technologii oferowanych lub natywnie wspieranych przez firmę Microsoft. Funkcjonalności, które oferuje aplikacja to głównie operacje CRUD (ang. create, read, update, delete, tłum. utwórz, odczytaj, aktualizuj, usuń), czyli cztery podstawowe operacje wykorzystywane do zarządzania przechowywanymi danymi. Całość została opublikowana na platformie Azure w celu umożliwienia komunikacji z platformą PayU, która wymaga publicznego adresu IP dla hostowanego API.

2.1.1. Azure

Microsoft Azure to platforma chmurowa, której architektura opiera się na modelu PaaS (ang. Platform-as-a-Service, tłum. Platforma jako usługa), która udostępnia ponad 200 produktów i usług do tworzenia i hostowania aplikacji. W swojej ofercie posiada między innymi mechanizmy pozwalające przetwarzać oraz składować dane, usługi sztucznej inteligencji i uczenia maszynowego, narzędzia do zarządzania siecią oraz wiele innych rozwiązań.

2.1.2. C#

Aplikacja serwerowa została napisana głównie z wykorzystaniem języka C#. Jest to nowoczesny, zorientowany obiektowo język programowania, mający korzenie w rodzinie języków C. Pierwsze wydanie języka C# w wersji 1.0 zostało zaprezentowane przez firmę Microsoft w styczniu 2002 roku. Początkowo język ten przypominał składnią wydany przez

[5] opisywana w podrozdziale 2.2.1

firmę Sun Microsystems w 1996 roku język Java[6]. Najnowsze wydanie języka C# w wersji 11 miało miejsce w listopadzie 2022 roku[7]. Język C# pozostaje jednym z pięciu najpopularniejszych języków programowania według statystyk serwisu GitHub z 2022 roku[8].

2.1.3. ASP.NET

ASP.NET jest open-source'owym frameworkiem do tworzenia witryn i aplikacji internetowych[9]. Jest częścią .NET, czyli opracowanej i ustandaryzowanej przez firmę Microsoft platformy deweloperskiej zbudowanej z narzędzi, języków i bibliotek do opracowywania różnorodnych typów aplikacji[10]. Technologia ASP.NET została wykorzystana do utworzenia API aplikacji serwerowej.

2.1.4. Entity Framework

Entity Framework jest obiektowo-relacyjnym mapperem, który pozwala twórcom pracować na relacyjnych bazach danych przy użyciu paradygmatu obiektowego[11], przez co ułatwia on wykonywanie operacji na bazie danych oraz redukuje ilość kodu w aplikacji. Jest dobrym rozwiązaniem przy założeniu, że baza danych oraz zapytania są stosunkowo mało skomplikowane. W przeciwnym razie może być odczuwalny spadek wydajności zapytań w porównaniu do innych metod dostępu do bazy danych. Entity Framework przechowuje informacje o tym, jak mapować obiekty oraz ich właściwości na tabele i kolumny w bazie danych. Oprócz tego pozwala na zautomatyzowane tworzenie baz danych na podstawie zdefiniowanych w programie modeli[12]. Takie podejście zostało wykorzystane przy projektowaniu warstwy dostępu do bazy danych.

2.1.5. JSON Web Token

JWT (ang. JSON Web Token) jest otwartym standardem, który definiuje sposób bezpiecznego przesyłania informacji pomiędzy modułami systemu z wykorzystaniem obiektów

---

[6] ("C# Version History: Examining the Language Past and Present")

[7] ("The history of C# - C# Guide")

[8] ("The top programming languages | The State of the Octoverse")

[9] ("ASP.NET overview")

[10] ("What is .NET? An open-source developer platform.")

[11] ("Entity Framework")

[12] ("Code First to a New Database - EF6")

JSON[13] (ang. JavaScript Object Notation). JWT może być również użyty do autoryzacji. Jego struktura składa się z trzech części rozdzielonych kropkami:

- nagłówka - zawiera informacje o typie tokena oraz rodzaj użytego szyfrowania,
- danych - dowolnie zdefiniowane informacje, np. data ważności tokena, rola użytkownika,
- podpisu - generowany na podstawie nagłówka, danych oraz tajnego klucza, używany do zweryfikowania, czy token nie został zmodyfikowany.

W przypadku autoryzacji po udanym logowaniu do użytkownika zwracany jest token JWT. Zazwyczaj, aby użytkownik mógł uzyskać dostęp do chronionych ścieżek lub zasobów, zwrócony przy logowaniu token musi zostać zawarty w nagłówku zapytania HTTP. Tokeny JWT zostały użyte w aplikacji do zabezpieczenia dostępu do endpointów API.

2.1.6. xUnit

xUnit jest narzędziem do testowania dla platformy .NET. Posiada szereg funkcji przydatnych podczas projektowania testów. Jedna z nich to możliwość dodania do testu atrybutu *[Fact]* lub *[Theory]*. Pierwszy z nich określa, że test nie przyjmuje żadnych parametrów wejściowych i powinien zawsze zwrócić prawdę. Z kolei drugi atrybut pozwala przetestować funkcjonalność dla różnych zestawów danych określających parametry wejściowe i oczekiwane wyniki[14]. Narzędzie xUnit zostało użyte do przygotowania testów aplikacji serwerowej.

2.1.7. Swagger

Specyfikacja OpenAPI, formalnie znana jako specyfikacja Swagger to światowy standard dokumentowania interfejsów REST API oraz narzędzie do wizualnego przedstawiania sporządzonej dokumentacji. Pozwala na przegląd kontrolerów, metod oraz modeli dostępnych w API[15].

[13] ("JSON Web Token Introduction - jwt.io")
[14] ("xUnit.net")
[15] ("Get Started With The OpenAPI Specification")

2.2. Aplikacja użytkownika

Aplikacja użytkownika, podobnie jak aplikacja serwerowa, została zaimplementowana z wykorzystaniem technologii oferowanych przez firmę Microsoft.

2.2.1. .NET MAUI

MAUI, czyli Multi-platform App User Interface to stosunkowo młoda technologia opracowana przez Microsoft do tworzenia multiplatformowych aplikacji z użyciem języka C# oraz XAML. MAUI umożliwia tworzenie aplikacji na systemy iOS, Android, macOS oraz Windows jednocześnie poprzez zunifikowanie kodu do jednego API. Mimo tej integracji technologia MAUI nie pozostaje hermetyczna i pozwala deweloperom na dostęp do wybranych aspektów każdej platformy z osobna. Przydatną funkcjonalnością, którą wspiera MAUI jest tzw. *hot-reload*, czyli aplikowanie zmian kodu aplikacji podczas jej pracy, bez konieczności rekompilacji[16]. Aplikacja kliencka użytkownika została przygotowana z wykorzystaniem MAUI.

2.3. Skrypt obsługujący kasowanie biletów

System symulujący kasowanie biletów z uwagi na posiadane cechy systemu wbudowanego[17] oraz konieczności zastosowania kamery jako urządzenia peryferyjnego do skanowania kodów QR, został zaimplementowana w formie skryptu napisanego w języku Python oraz biblioteki OpenCV. Skrypt został przetestowany na laptopie z wbudowaną kamerą, ale z powodzeniem może zostać wdrożony na urządzeniach typu RaspberryPi.

2.3.1. Python

Python jest zorientowanym obiektowo, interpretowanym językiem programowania wysokiego poziomu. Umożliwia dynamiczne typowanie zmiennych oraz posiada rozbudowany pakiet bibliotek standardowych, przez co jest chętnie wykorzystywany do tworzenia różnorodnego oprogramowania[18]. Został wykorzystany do obsługi aplikacji klienckiej po stronie kina, z uwagi na łatwą integrację z systemami linuxowymi oraz biblioteką OpenCV.

---

[16] ("What is .NET MAUI? - .NET MAUI")

[17] system przeznaczony do wykonywania ograniczonej liczby zadań

[18] ("The Python Tutorial — Python 3.11.1 documentation")

2.3.2. Kod QR

Kody QR (ang. quick response code) są jednym z typów dwuwymiarowego kwadratowego kodu matrycowego. Zostały opracowane w 1994 roku przez firmę Denso Wave. Oprócz możliwości kodowania znaków w modułach[19] o wielkości od 21x21 do 177x177 specyfikacja dopuszcza łączenie do 16 kodów razem w celu zapisania większej ilości informacji[20]. Jednym z ciekawych mechanizmów wykorzystywanych w budowie kodu QR jest mechanizm korekcji błędów na bazie algorytmu Reeda-Solomona, który pozwala odczytać kod QR pomimo jego częściowej degradacji. Kody QR zostały wykorzystane jako medium biletów wygenerowanych przez WebAPI.

2.3.3. OpenCV

OpenCV jest biblioteką udostępniającą setki zaawansowanych zoptymalizowanych algorytmów przetwarzania obrazów[21]. W projekcie została użyta do analizy zdjęć pod kątem odnajdowania i odczytywania kodów QR.

2.4. Baza danych

Do przechowywania i zarządzania danymi zgromadzonymi na potrzeby działania całego systemu samoobsługowego kina została użyta relacyjna baza danych[22]. Obecnie istnieje szeroki wybór dostawców systemów baz danych. W projekcie został użyty silnik MS SQL Server z uwagi na łatwość integracji z innymi produktami od firmy Microsoft oraz darmową licencję.

2.4.1. Microsoft SQL Server

MS SQL Server to system zarządzania bazą danych. Główny produkt bazodanowy firmy Microsoft, którego językiem zapytań jest Transact-SQL. Oprócz samego silnika bazodanowego, firma Microsoft udostępnia szereg narzędzi do zarządzania i utrzymywania bazami danych[23].

---

[19] kwadratach o jednym z dwóch kolorów - ciemnym lub jasnym

[20] (“Kod QR – Wikipedia, wolna encyklopedia”)

[21] (“OpenCV - Introduction”)

[22] opisany i zorganizowany zbiór tabel połączonych relacjami (związkami między sobą)

[23] (“SQL Server technical documentation”)

2.5. Dostawca usług płatniczych

Aby zagwarantować bezpieczeństwo transakcji, szybkość oraz zgodność z możliwie dużą liczbą systemów płatności, a także w celu symulowania realnych poleceń płatności system został zintegrowany z platformą testową usługodawcy PayU.

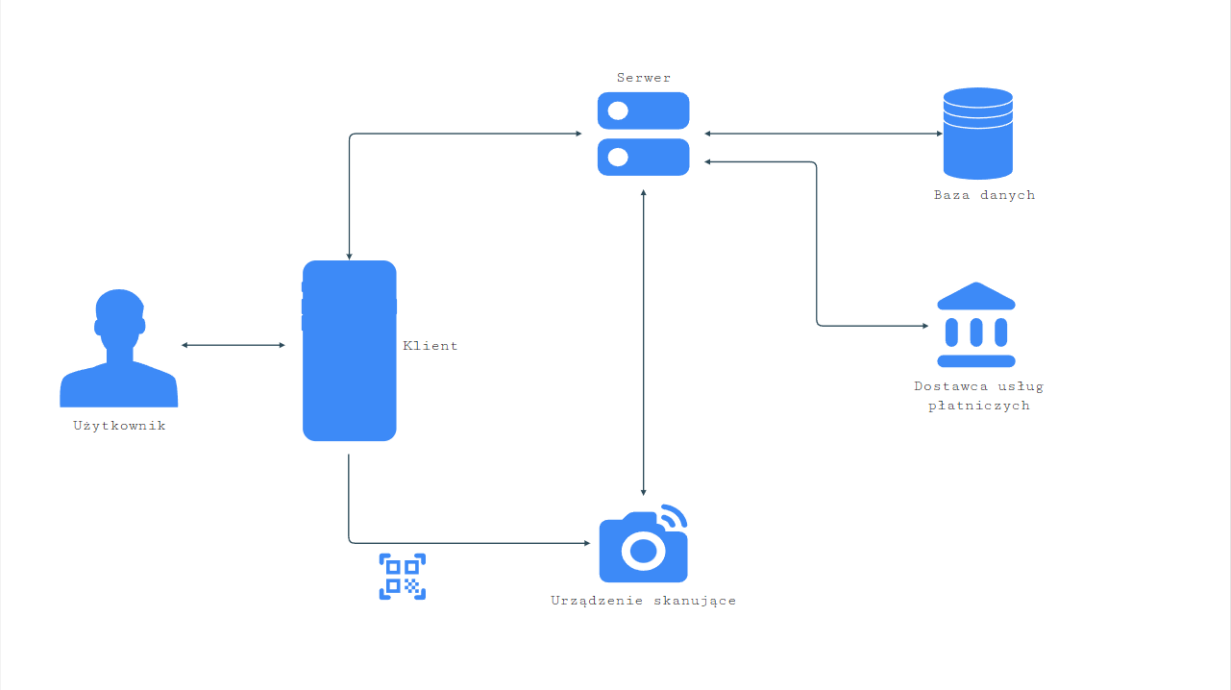
2.5.1. PayU

PayU jest operatorem płatności internetowych, umożliwiającym stosunkowo łatwą integrację swojego systemu z platformami sprzedażowymi. PayU udostępnia darmową platformę testową *Sandbox* oraz pełen zestaw API, które pozwalają tworzyć, rozliczać, anulować, pobierać zamówienia, wykonywać wypłaty i pobierać raporty[24]. PayU zostało użyte w projekcie z uwagi na darmowy dostęp do platformy testowej oraz API.

[24] ("PayU - Overview")

3. Architektura systemu

System został zaimplementowany w architekturze klient-serwer. Oprócz aplikacji klienckiej serwer komunikuje się również z urządzeniem skanującym w kinie oraz dostawcą usług płatniczych (PayU). Aplikacja użytkownika w swojej budowie wykorzystuje wzorzec MVVM (model-view-viewmodel)[25]. Dane aplikacji przechowywane są w relacyjnej bazie danych. Całokształt architektury systemu przedstawiono na rysunku 1.

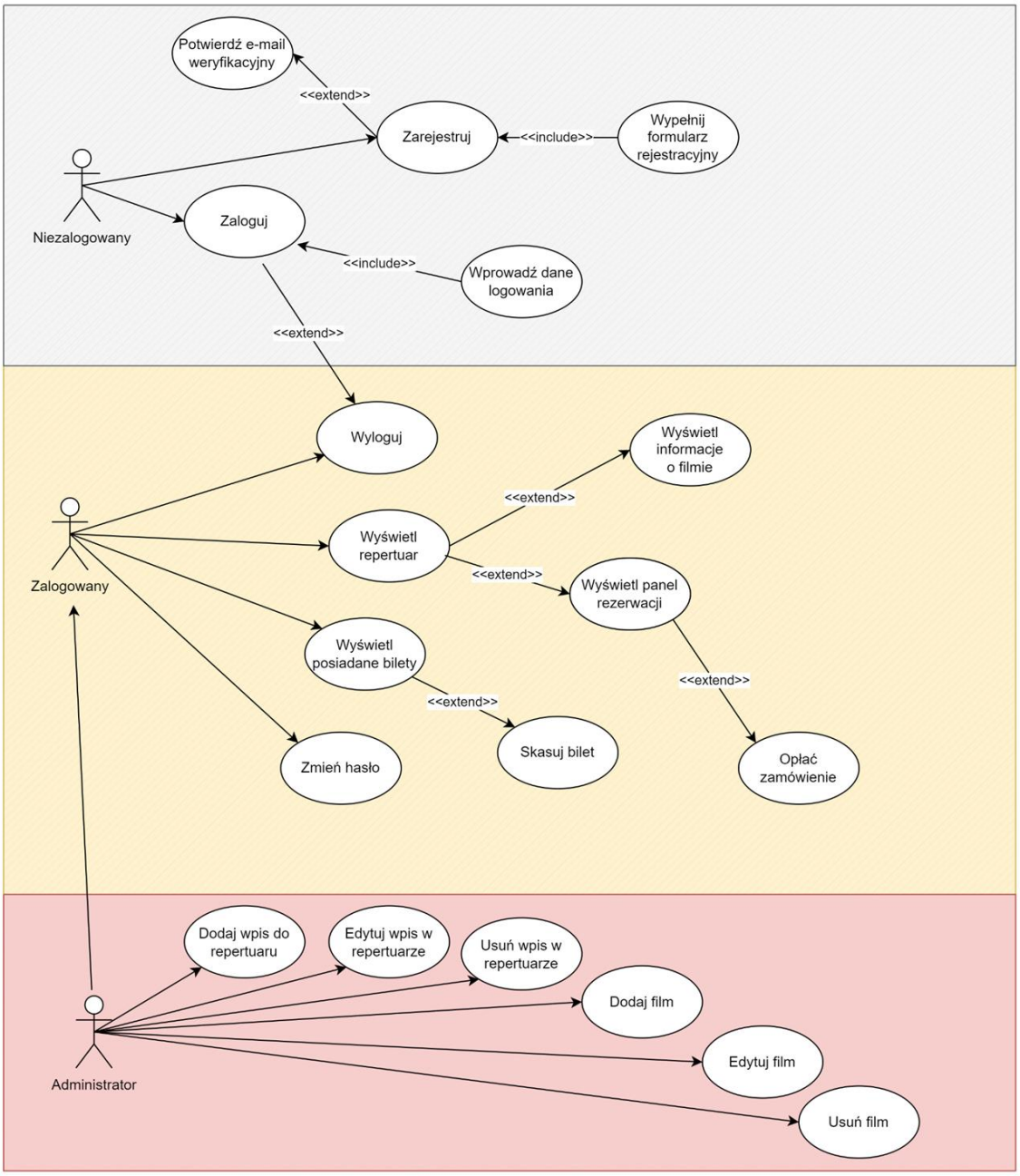


Rys. 1 - Architektura systemu
[opracowanie własne]

3.1. Diagram przypadków użycia

Diagram przypadków użycia prezentuje dostępne funkcjonalności dla poszczególnych aktorów systemu oraz relacje pomiędzy nimi. Dla omawianej aplikacji przewidziano trzech aktorów: użytkownika niezalogowanego, użytkownika zalogowanego oraz administratora, który oprócz właściwych dla siebie akcji, dziedziczy akcje użytkownika zalogowanego.

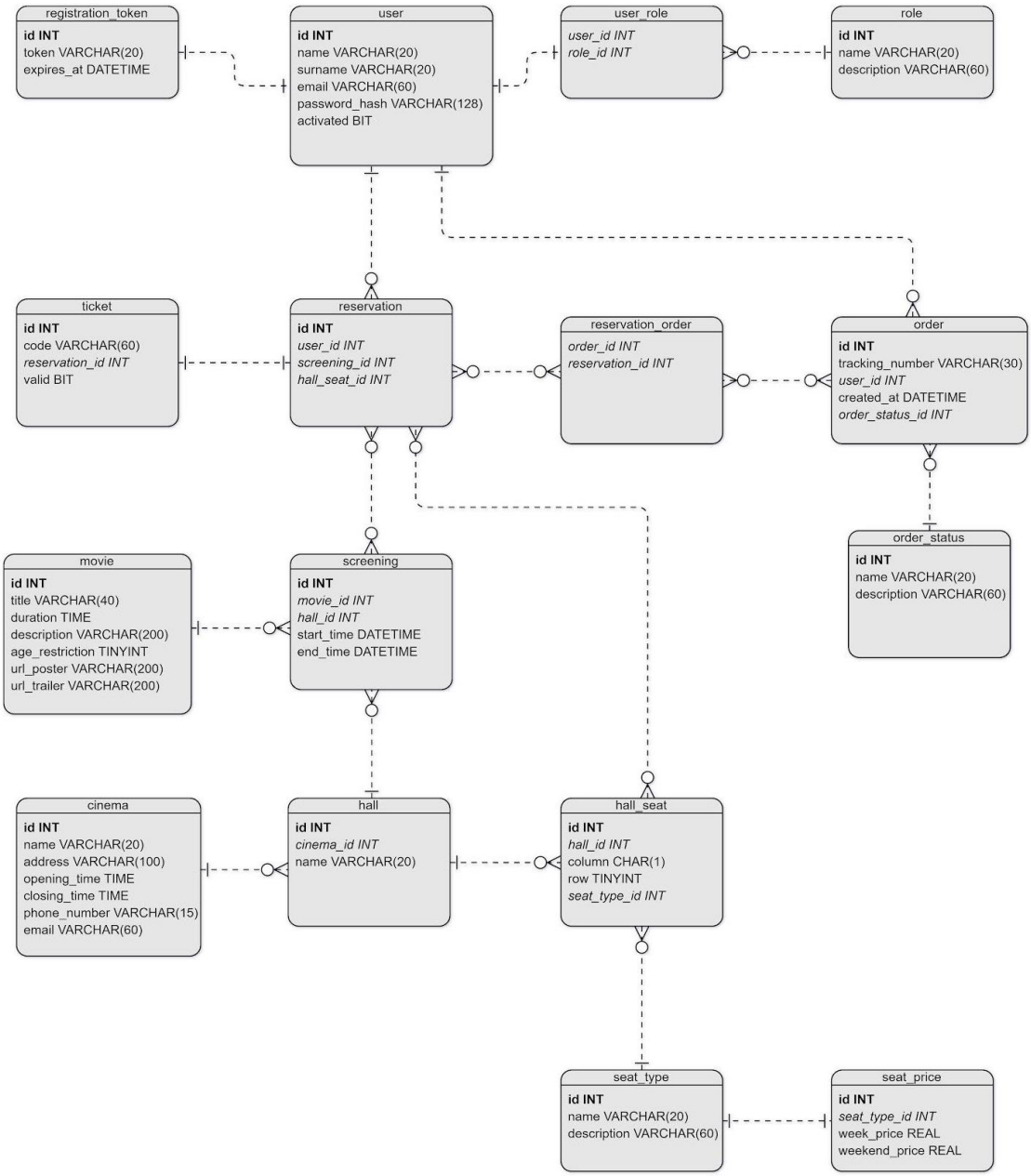
[25] ("Model-View-ViewModel (MVVM)")



Rys. 2 - Diagram przypadków użycia
[opracowanie własne]

3.2. Diagram relacji encji

Diagram relacji encji przedstawia strukturę bazy danych w formie graficznej. Informacje zawarte na diagramie to występujące tabele, ich kolumny, oraz powiązania pomiędzy kolumnami. Relacje, które zachodzą w przygotowanej bazie danych to: jeden-do-jeden, jeden-do-wielu, wiele-do-jednego oraz wiele-do-wielu.

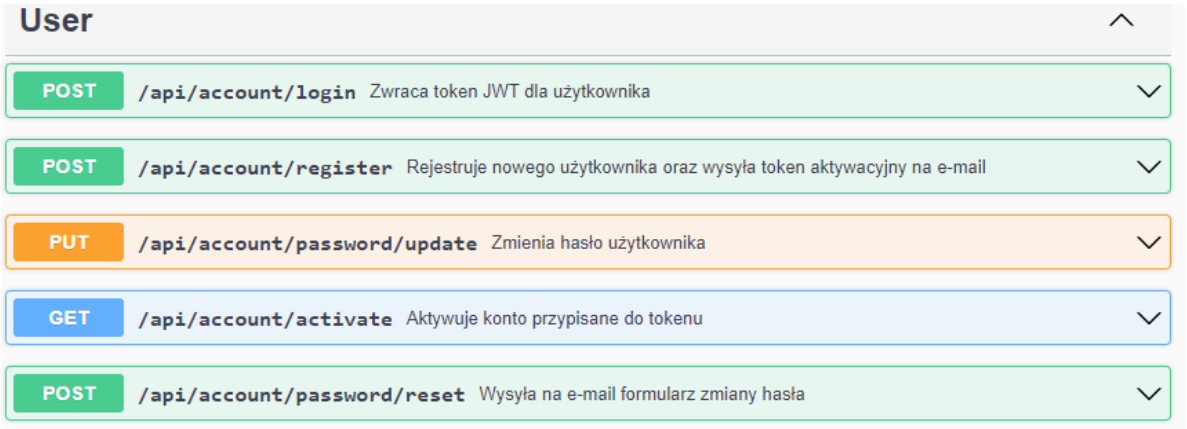


Rys. 3 - Diagram relacji encji [opracowanie własne]

3.3. Dokumentacja API

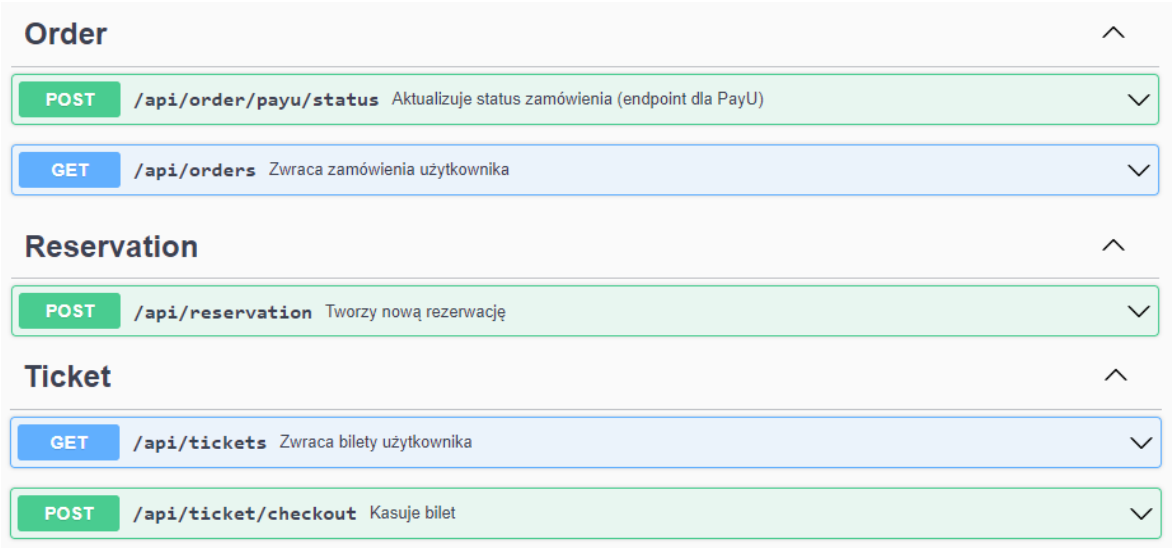
Do przygotowania dokumentacji WebAPI użyto narzędzia Swagger oraz wbudowanych w IDE mechanizmów, które automatycznie generują dokumentację w standardzie OpenApi. Udokumentowane moduły WebAPI przedstawiono na rysunkach 4,5,6.

Moduł *"User"* jest odpowiedzialny za funkcjonalności związane bezpośrednio z użytkownikiem. Należą do nich logowanie, rejestracja, zmiana hasła, aktywacja konta czy wysłanie prośby o zresetowanie hasła.



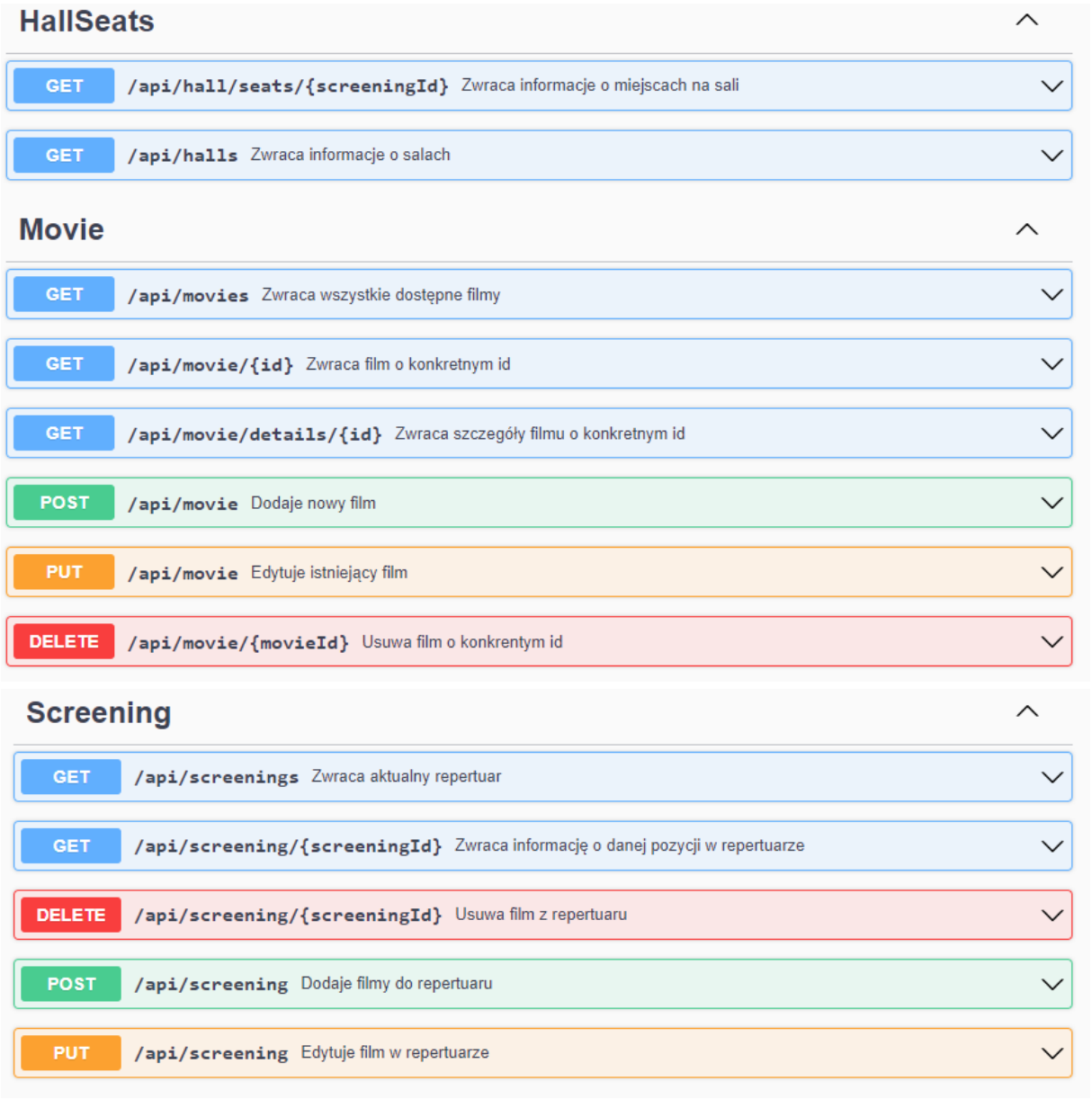
Rys. 4 - Moduł User [dokumentacja Swagger]

Moduły *"Order", "Reservation" oraz "Ticket"* są powiązane z przetwarzaniem oraz realizacją zamówień użytkownika. Moduł *"Reservation"* udostępnia opcję dodania nowej rezerwacji, która na czas realizacji zamówienia blokuje możliwość zakupu wybranych miejsc innym użytkownikom. Wraz z rezerwacją w systemie tworzone jest wirtualne zamówienie, którego następnie przekazywane jest do systemu płatności. Informacje na temat statusu płatności oraz jego aktualizacje przez system PayU są dostępne z poziomu modułu *"Orders"*. Za obsługę wygenerowanych biletów odpowiada moduł *"Tickets"*, gdzie użytkownik może pobrać swoje bilety, a skrypt obsługujący kasowanie biletów może potwierdzić ich ważność.



Rys. 5 - Moduły Order, Reservation, Ticket [dokumentacja Swagger]

Moduły *"Movie", "Screening"* oraz *"HallSeats"* umożliwiają pobieranie informacji o repertuarze, filmach w nim zawartych oraz dostępnych miejscach na sali. Administrator dodatkowo ma opcję dodawania, edycji oraz usuwania pozycji w repertuarze oraz filmów.



Rys. 6 - Moduły HallSeats, Movie, Screening [dokumentacja Swagger]

3.4. Usługi płatnicze

Do utworzenia punktu płatności (ang. POS - point of sale) zostało wykorzystane PayU REST API, czyli domyślny sposób integracji, umożliwiający obsługę wszystkich dostępnych kanałów płatności dostawcy.

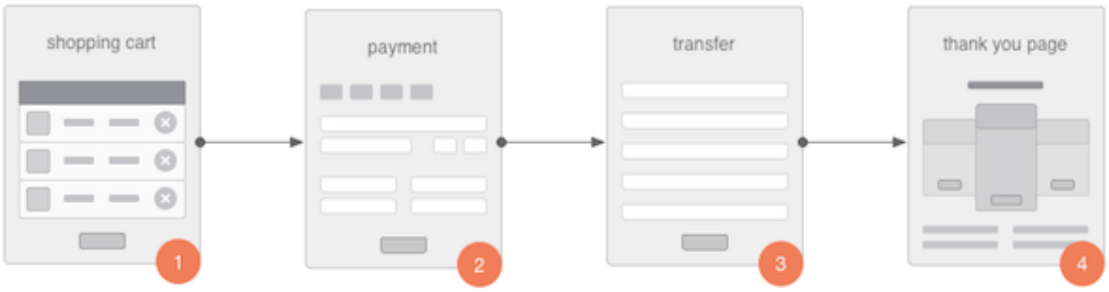
Konfiguracja POS wymaga utworzenia testowego sklepu na platformie PayU Sandbox oraz dodania wybranego typu punktu płatności. Dla tak utworzonego POS zostały wygenerowane klucze konfiguracyjne, które kolejno są dołączane do nagłówków zapytań podczas tworzenia nowego zamówienia w systemie PayU. Na rysunku 8 przedstawiono widok na panel użytkownika platformy PayU Sandbox.

Proces złożenia nowego zamówienia od strony biznesowej przebiega w następujący sposób:

1. Kupujący klika w przycisk reprezentujący usługę płatności PayU.
2. System PayU prezentuje stronę z podsumowaniem zamówienia, na której kupujący potwierdza płatność. System PayU przekierowuje kupującego na stronę banku.
3. Kupujący akceptuje płatność na stronie Banku. Następuje przekierowanie kupującego ponownie na stronę systemu PayU.
4. System Sprzedawcy prezentuje podziękowanie za transakcje.

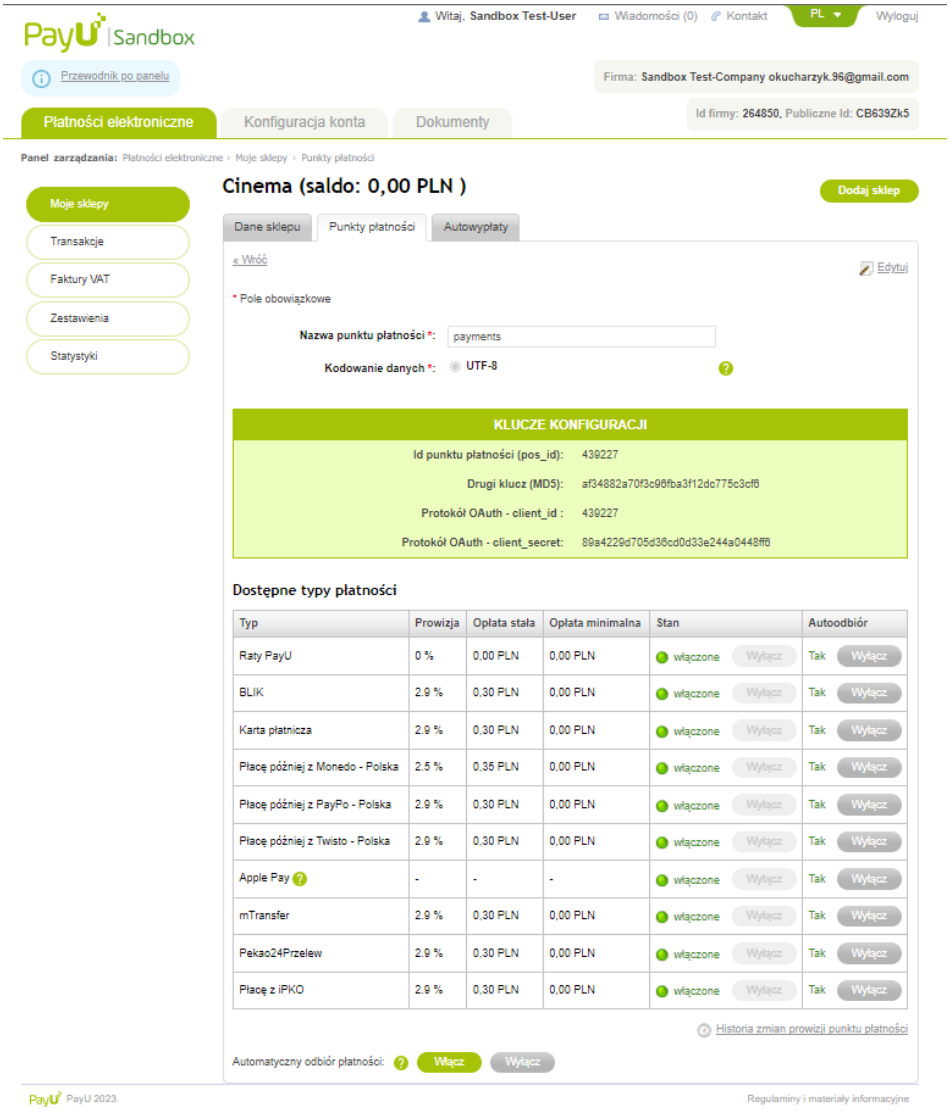
Proces przedstawiono graficznie na rysunku 7[26].

Do testowania różnych scenariuszy przebiegu płatności zamówienia PayU udostępnia testowe karty, które powodują określone zachowania autoryzacji. Przykładowe dane kart oraz ich zachowania przedstawiono na rysunku 9.



Rys. 7 - Proces złożenia zamówienia w PayU [dokumentacja PayU]

[26] ("PayU - Overview")



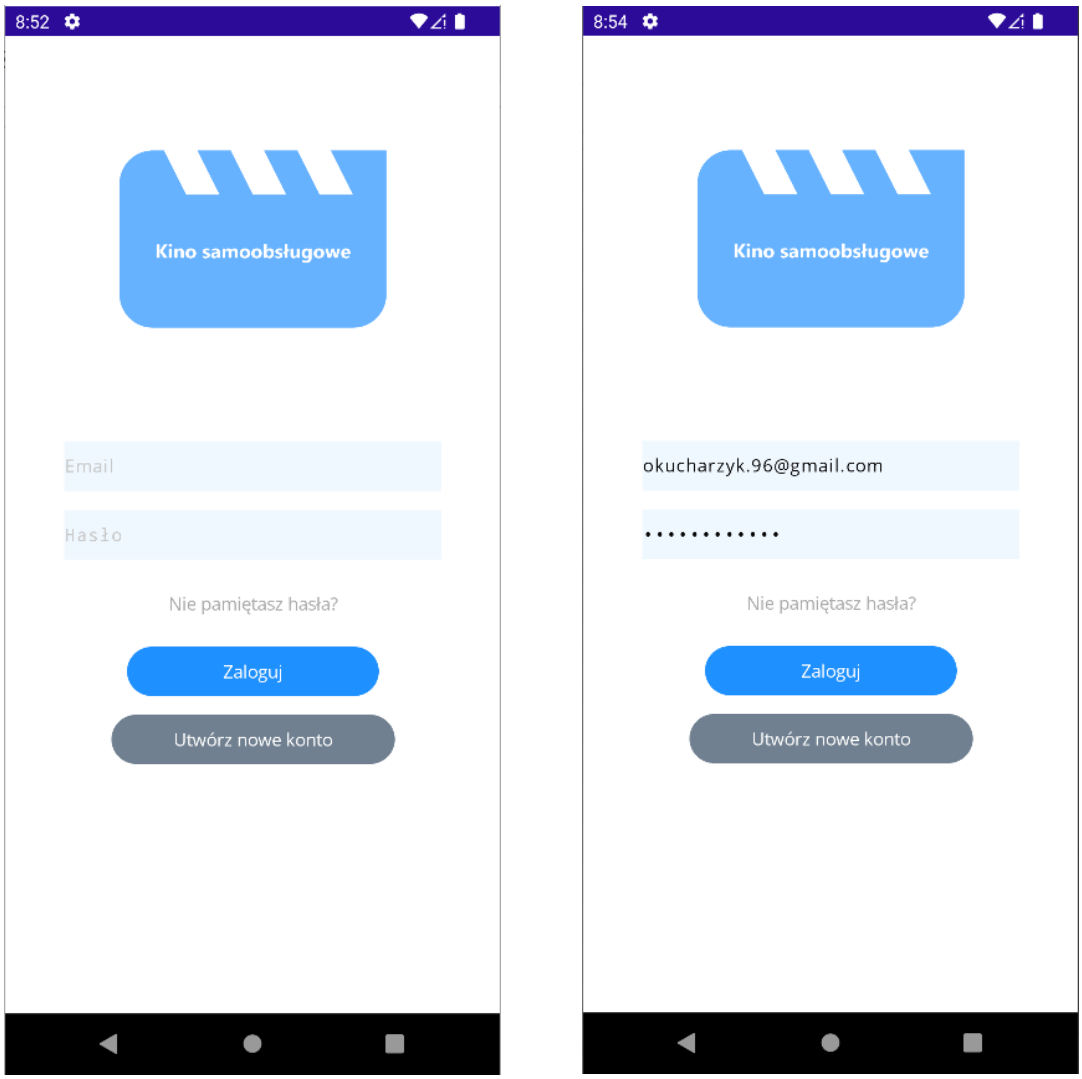
Rys. 8 - Panel managera [platforma PayU Sandbox]

Numer	Miesiąc	Rok	CVV	Wynik 3DS	Raty Mastercard	Zachowanie
5100052384536826	12	29	123	domyślny	nie	Autoryzacja pozytywna
5521455186577727	12	29	123	nieudane uwierzytelnienie typu frictionless	nie	brak autoryzacji
5405860937270285	12	29	123	domyślny	nie	50% szans na udaną autoryzację
4532598021104999	12	29	123	domyślny	nie	udana autoryzacja tylko na kwotę poniżej 1000 groszy

Rys. 9 - Dane oraz zachowania testowych kart płatniczych [dokumentacja PayU REST API]

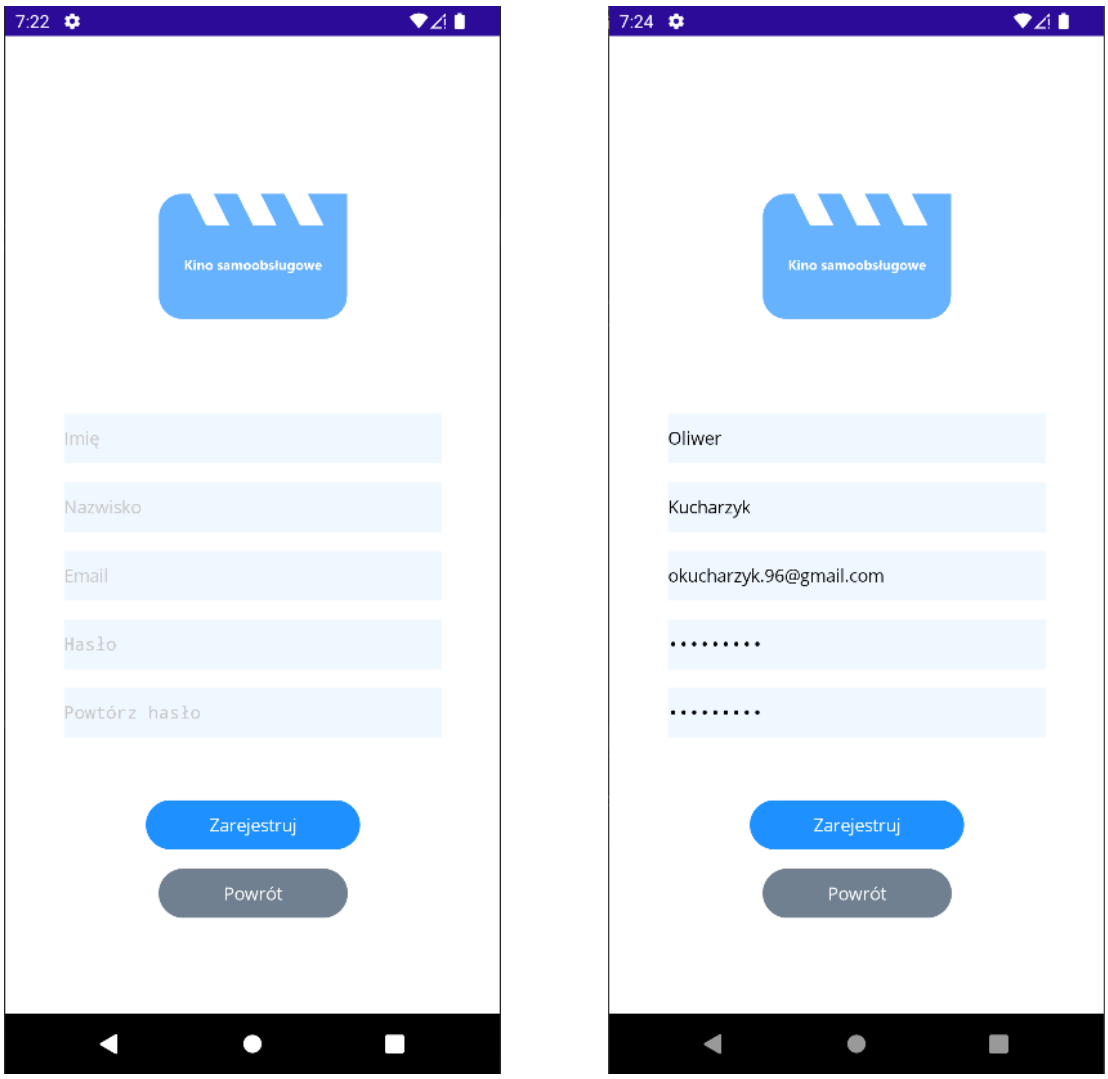
4. Interfejsy użytkownika

Aplikacja kliencka została przygotowana w formie aplikacji mobilnej, której główne graficzne interfejsy użytkownika zostały omówione w dalszej części rozdziału.



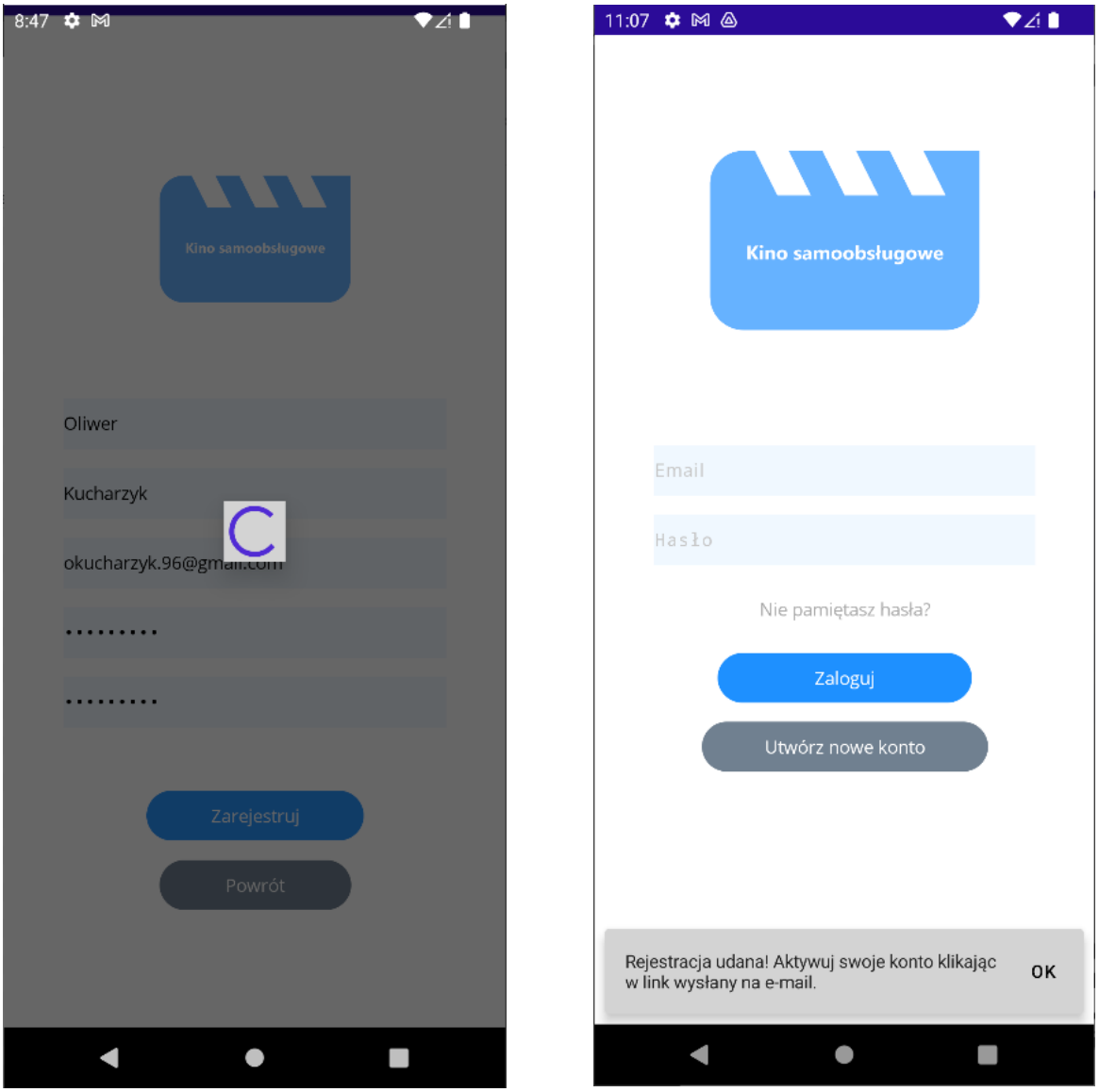
Rys. 10 - Ekran logowania [opracowanie własne]

Na rysunku 10 został przedstawiony panel logowania. Jest to pierwszy prezentowany widok po otwarciu aplikacji, w przypadku kiedy użytkownik nie jest jeszcze zalogowany. Z poziomu ekranu logowania użytkownik ma możliwość utworzyć nowe konto, zalogować się na wcześniej utworzone konto lub poprosić o utworzenia nowego hasła poprzez formularz wysyłany na e mail powiązany z kontem.



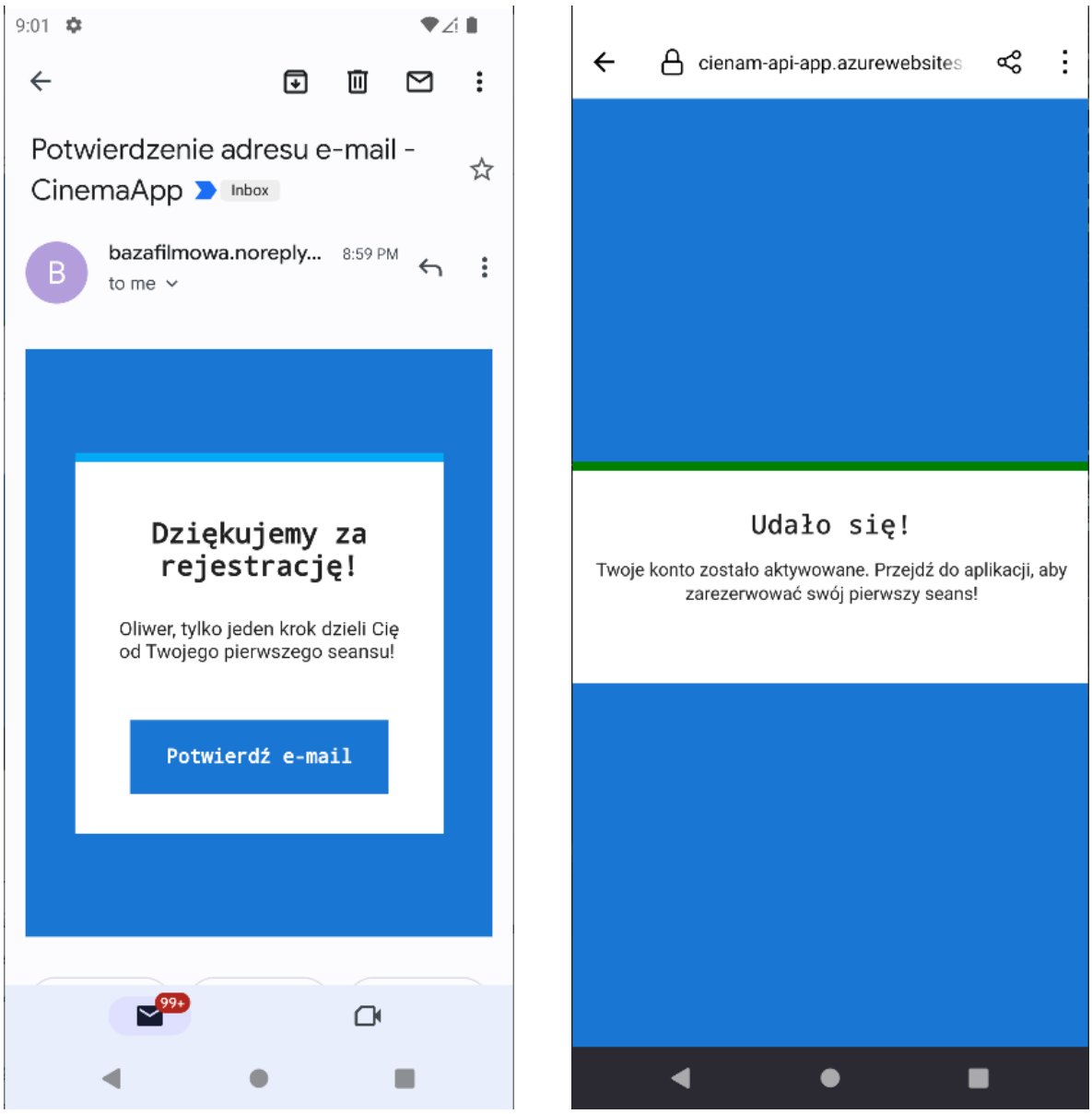
Rys. 11 - Ekran rejestracji[opracowanie własne]

Klikając w przycisk *"Utwórz nowe konto"* użytkownik jest przenoszony do widoku rejestracji, przedstawionego na rysunku 11. Do założenia konta wymagane jest podanie podstawowych informacji osobowych - imię, nazwisko, adres e-mail - oraz utworzenie hasła składającego się z przynajmniej ośmiu znaków. Użytkownik ma możliwość powrotu do widoku logowania klikając w przycisk *"Powrót"*. Kliknięcie przycisku *"Zarejestruj"* będzie skutkować wysłaniem na serwer zapytania o rejestrację nowego użytkownika z dołączonymi danymi wprowadzonymi w formularzu. Po pozytywnej walidacji zapytania, tzn. jeśli wprowadzone dane są poprawne oraz w bazie danych nie istnieje użytkownik powiązany z danym adresem e-mail, na elektroniczną skrzynkę pocztową wysyłany jest ważny 24 godziny link aktywujący konto. Informacja o wysłanym mailu jest przedstawiana przez komponent *"SnackBar"* na ekranie logowania przedstawiony na rysunku 12.



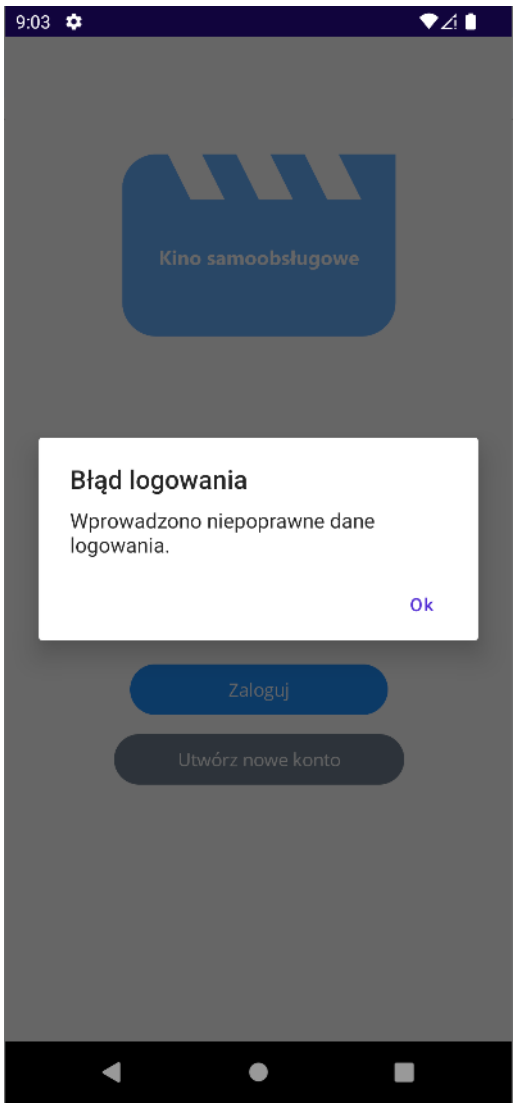
Rys. 12 - Ekran logowania , potwierdzenie pozytywnego przebiegu rejestracji [opracowanie własne]

Na rysunku 13 przedstawiono przykładową spersonalizowaną wiadomość e-mail z linkiem aktywacyjnym. Po kliknięciu przycisku *"Potwierdź e-mail"* na serwer wysyłane jest zapytanie zawierające unikalny token aktywacyjny powiązany z zarejestrowanym kontem. Jeśli aktywacja przebiegła pomyślnie - tzn. token nie stracił ważności, ani konto nie zostało już wcześniej aktywowane - serwer w odpowiedzi przekieruje użytkownika na stronę podsumowującą proces aktywacji konta.

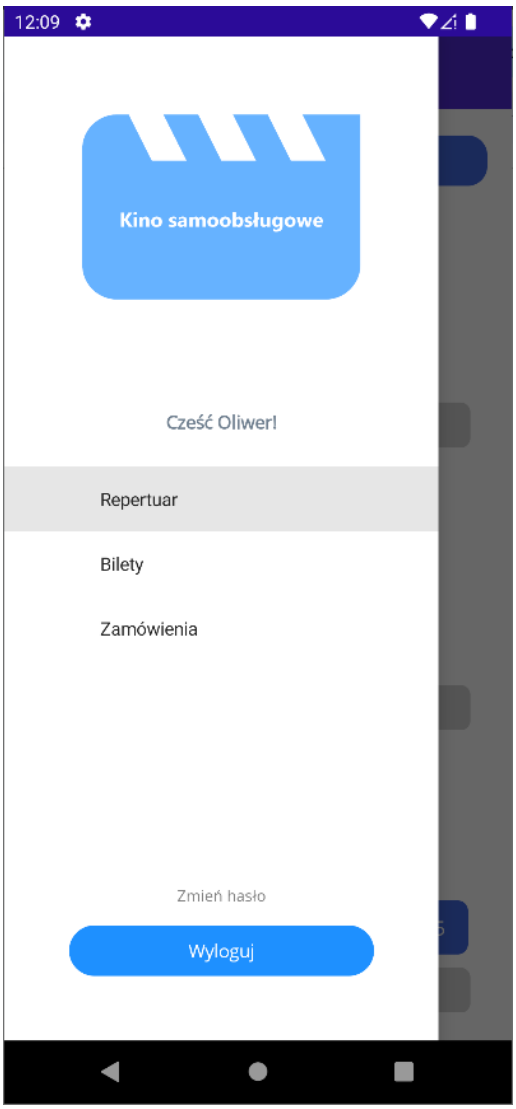


Rys. 13 - E-mail aktywacyjny oraz strona przekierowania po pozytywnej aktywacji [opracowanie własne]

Jeśli proces rejestracji oraz aktywacji konta przebiegł pomyślnie, użytkownik ma możliwość zalogowania się do swojego konta wprowadzając w widoku logowania powiązany adres e-mail i hasło oraz klikając w przycisk *"Zaloguj"*. Po pozytywnym uwierzytelnieniu użytkownik jest przenoszony do widoku repertuaru (rysunek 16), a w *"Secure Storage"* urządzenia zapisywany jest otrzymany od serwera token JWT. Token JWT od tej pory jest dołączony do każdego zapytania wysłanego na serwer i zawiera informacje personalne o użytkowniku oraz przypisaną mu rolę. W przypadku braku uwierzytelnienia na ekranie logowania wyświetlany jest odpowiedni alert (rysunek 14).

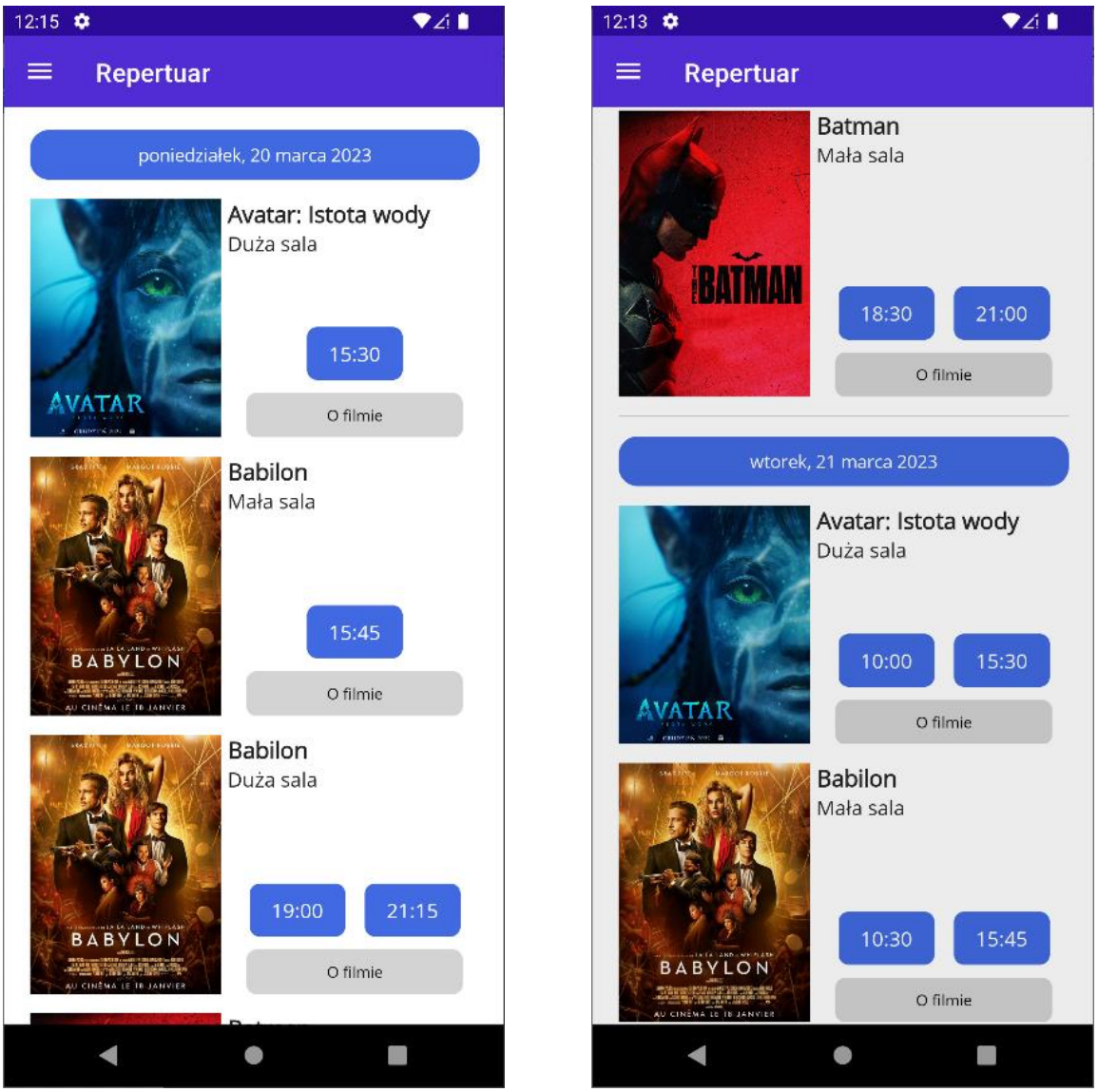


Rys. 14 - Alert niepowodzenia autentykacji - widok logowania [opracowanie własne]



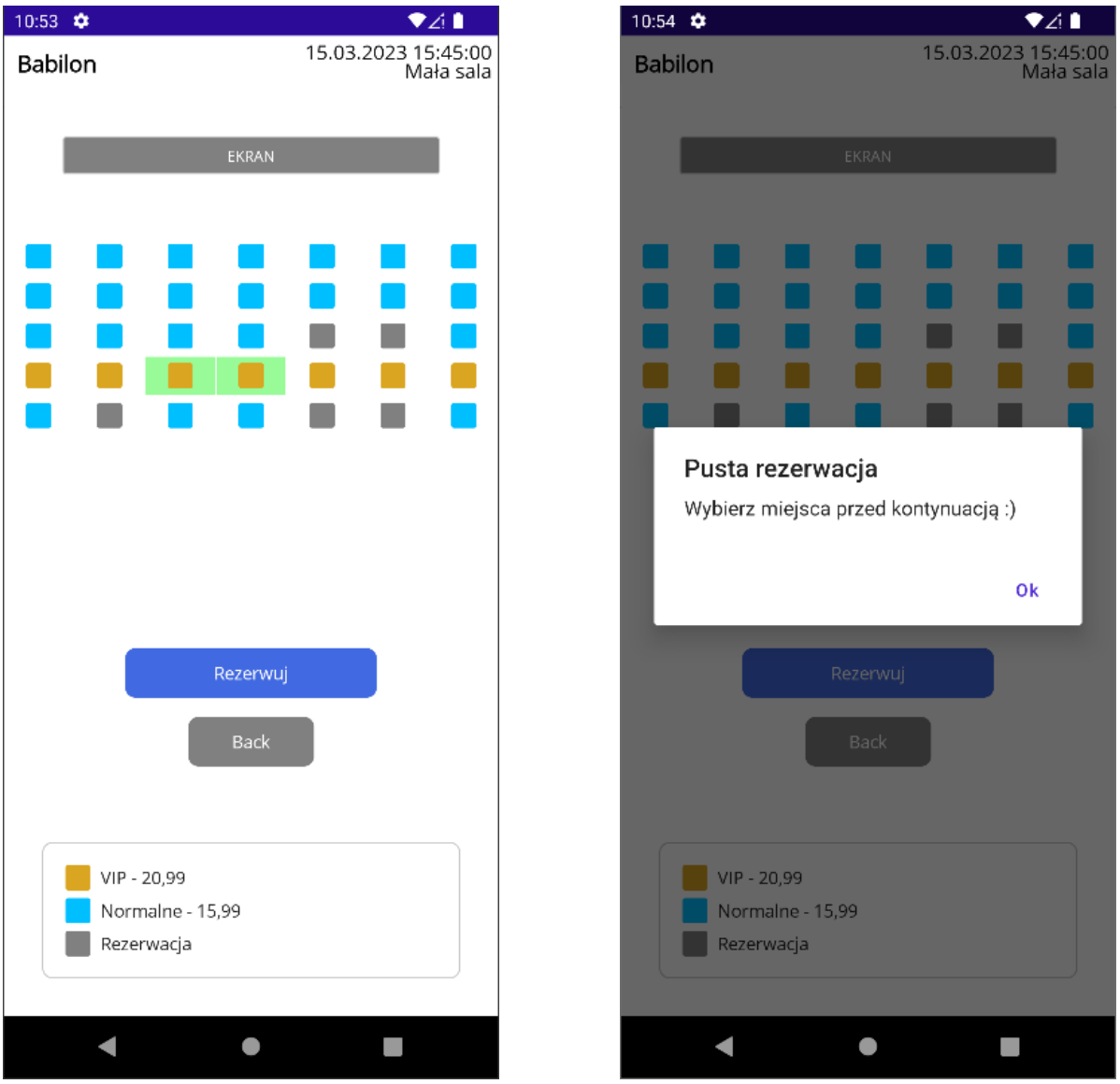
Rys. 15 - Widok menu po zalogowaniu [opracowanie własne]

Po zalogowaniu użytkownik może korzystać z bocznego menu otwieranego przyciskiem znajdującym się na górnym pasku nawigacyjnym lub gestem przeciągnięcia do środka ekranu. Menu, przedstawione na rysunku 15, pozwala na nawigowanie pomiędzy widokami repertuaru, biletów oraz zamówień przypisanych do konta. Przycisk *"Wyloguj"* powoduje przejście do ekranu logowania oraz usunięcie tokena JWT z *"Secure Storage"* urządzenia. Klikając w tekst *"Zmiana hasła"* użytkownik jest przenoszony do widoku aktualizacji hasła przypisanego do konta. W górnej części menu wyświetlane są dane aktualnie zalogowanego użytkownika.



Rys. 16 - Widok repertuaru
[opracowanie własne]

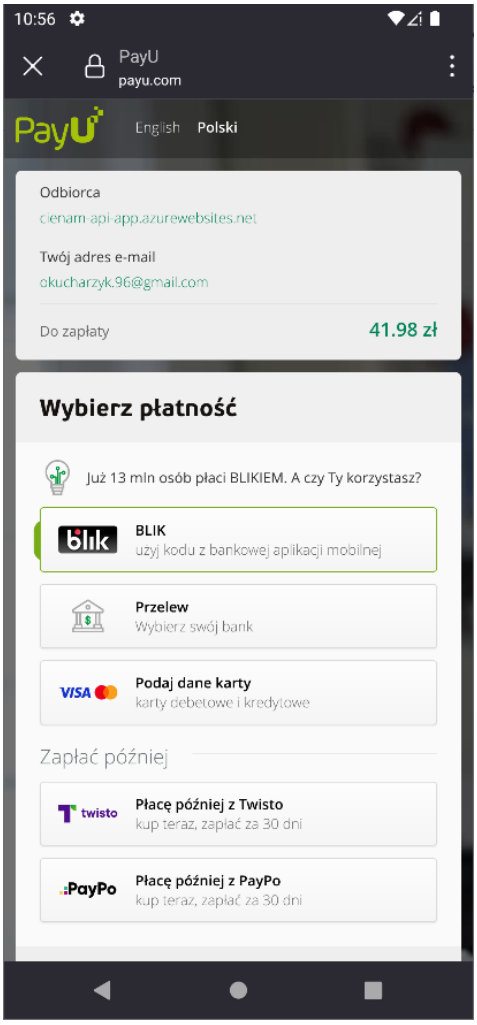
Repertuar kina, przedstawiony na rysunku 16, wyświetlany jest w formie przewijanego widoku. Seanse zostały pogrupowane według daty oraz sali na jakiej są prezentowane. Każda grupa posiada swój nagłówek z datą, pod którą są wyświetlane filmy prezentowane w danym dniu. Kafelki repertuaru zawierają podstawowe informacje o filmie – plakat, tytuł oraz salę, na której seans jest wyświetlany. Do każdej pozycji przypisane zostały przyciski z godzinami, w jakich można obejrzeć konkretny seans. Po kliknięciu na przycisk otwierany jest widok rezerwacji na film o wybranej godzinie. Szczegóły filmu są dostępne po kliknięciu w przycisk *"O filmie"*.



Rys. 17 - Widok rezerwacji
[opracowanie własne]

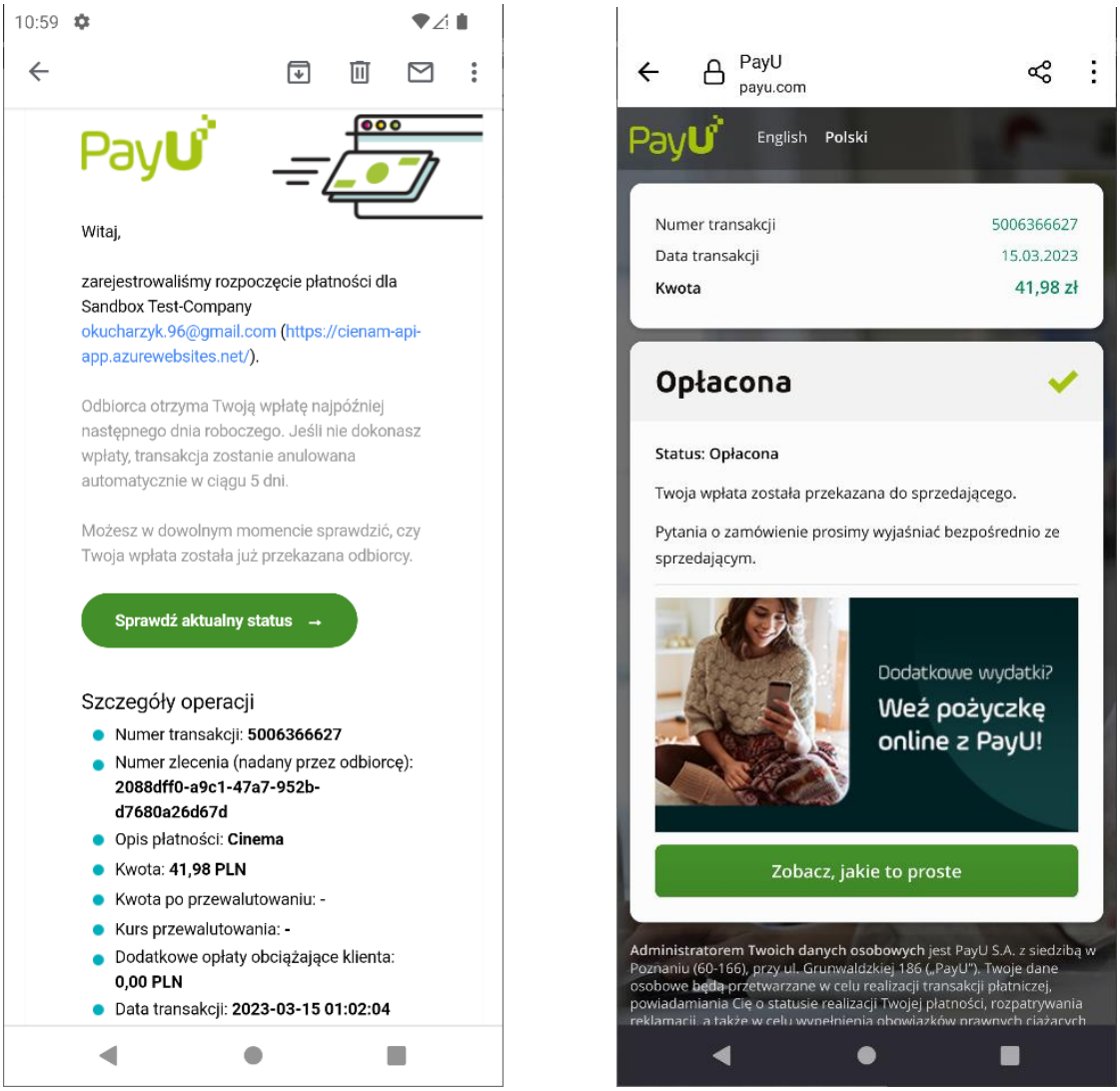
Widok rezerwacji przedstawiono na rysunku 17. W górnej części ekranu wyświetlane są informacje o seansie, którego dotyczy rezerwacja – tytuł filmu, data oraz godzina projekcji, a także sala na której seans jest wyświetlany. W środkowej części widoku prezentowany jest podgląd sali kinowej z zaznaczeniem dostępnych oraz zajętych miejsc, a także ich rodzaju – standardowe lub VIP. Legenda opisująca rodzaj miejsca jest widoczna w dolnej części ekranu i zawiera informację o cenie biletu na dany rodzaj miejsca. Użytkownik dla pojedynczej rezerwacji ma możliwość wyboru maksymalnie sześciu spośród dostępnych miejsc na sali. Wybrane miejsca są podświetlane zielonym kolorem. Użytkownik ma możliwość powrotu do widoku repertuaru klikając przycisk *„Powrót”* lub finalizacji rezerwacji klikając przycisk *„Rezerwuj”*. W drugim przypadku, jeżeli wybrano przynajmniej jedno miejsce, na serwer wysyłane jest zapytanie z rezerwacją. Po otrzymaniu zapytania serwer tworzy nowe

zamówienie i przypisuje do niego przekazaną rezerwację. Następnie do punktu płatności PayU wysyłane jest zapytanie o utworzenie nowej płatności PayU dla utworzonego zamówienia. API PayU w odpowiedzi zwraca informacje o utworzonym punkcie płatności – między innymi adres URL, na który należy przekierować użytkownika w celu finalizacji zamówienia. W odpowiedzi do klienta zwracany jest wspomniany adres przekierowania, który następnie jest otwierany w oknie przeglądarki wewnątrz aplikacji, co przedstawiono na rysunku 18. Po finalizacji transakcji użytkownik jest przekierowywany na stronę podsumowania płatności (rysunek 19).



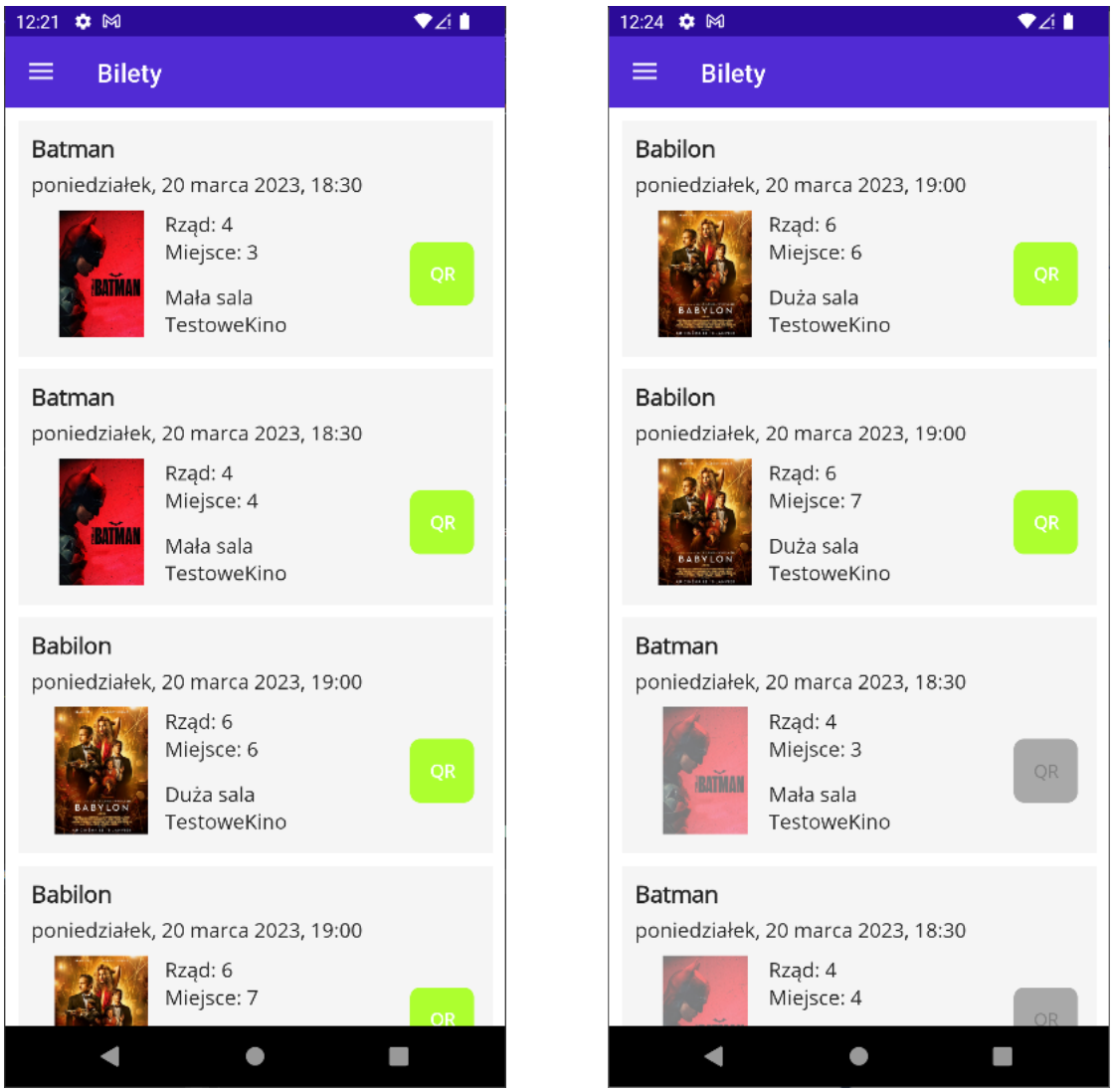
Rys. 18 - Strona finalizacji zamówienia [PayU]

Rys. 19 - Strona podsumowania finalizacji zamówienia [opracowanie własne]



Rys. 20 - Informacje o statusie płatności [PayU]

Po przejściu na adres finalizacji zamówienia, przez PayU na adres e-mail użytkownika przesyłane jest powiadomienie o rozpoczęciu płatności. Wiadomość zawiera link do śledzenia aktualnego statusu płatności bezpośrednio w usłudze PayU. Powiadomienie oraz przykładowy status płatności wystawiony przez dostawcę usług płatniczych przedstawiono na rysunku 20. W aplikacji klienckiej wszystkie zamówienia przypisane do użytkownika oraz ich statusy dostępne są w widoku zamówień, który przedstawiono na rysunku 23. Aktualizacja statusów płatności w bazie danych aplikacji odbywa się poprzez cykliczne wysyłanie zapytań POST przez API PayU na przygotowany według specyfikacji usługodawcy endpoint.



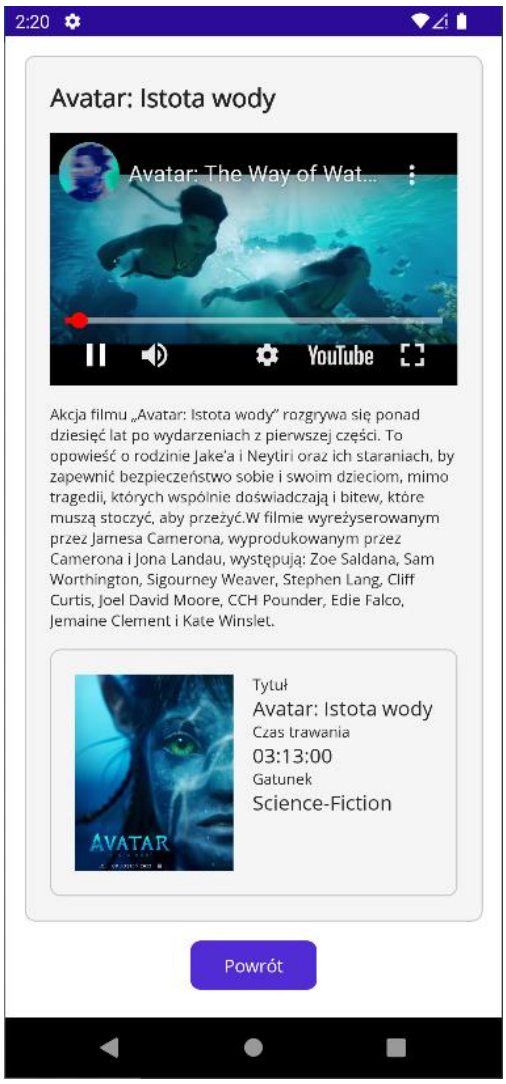
Rys. 21 - Ekran biletów
[opracowanie własne]

Jeżeli finalizacja płatności przebiegła pomyślnie i API aplikacji otrzymało od PayU aktualizację statusu zamówienia, w systemie generowane są bilety powiązane z rezerwacjami przypisanymi do tego zamówienia. Wygenerowane wejściówki są widoczne z poziomu ekranu biletów. Do każdego biletu przypisany jest kod QR, który zawiera numer identyfikacyjny pozwalający na jego późniejsze uwierzytelnienie. Po kliknięciu w zielony przycisk *"QR"* w widoku użytkownika wyświetlany jest kod, który może zostać zeskanowany w kinie (rysunek 22). Bilety, które zostały już skasowane są wyszarzone i nie ma możliwości ponownego wyświetlenia ich kodu QR. Przedawnione wejściówki są przechowywane w bazie danych, ale nie są wyświetlane w aplikacji użytkownika.

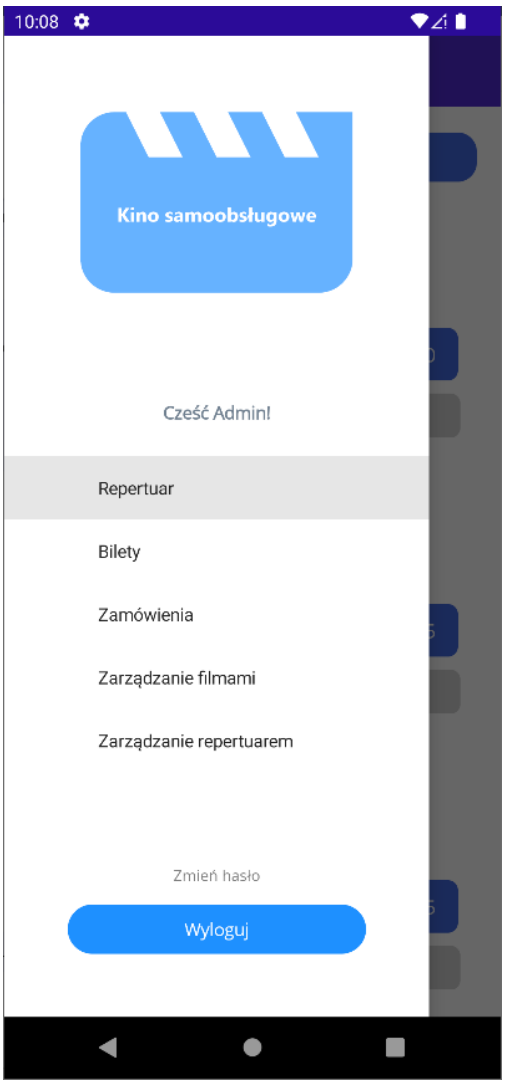
Rys. 22 - Widok kodu QR biletu [biblioteka Syncfusion]

Rys. 23 - Widok zamówień [opracowanie własne]

Na rysunku 24 przedstawiono widok szczegółów filmu. Ekran jest dostępny po kliknięciu w przycisk *"O filmie"* z poziomu widoku repertuaru. Ekran zawiera opis fabuły, okno z odtwarzaczem zwiastuna filmu oraz informacje o czasie trwania i gatunku. Po kliknięciu przycisku *"Powrót"* użytkownik jest przenoszony z powrotem do widoku repertuaru.

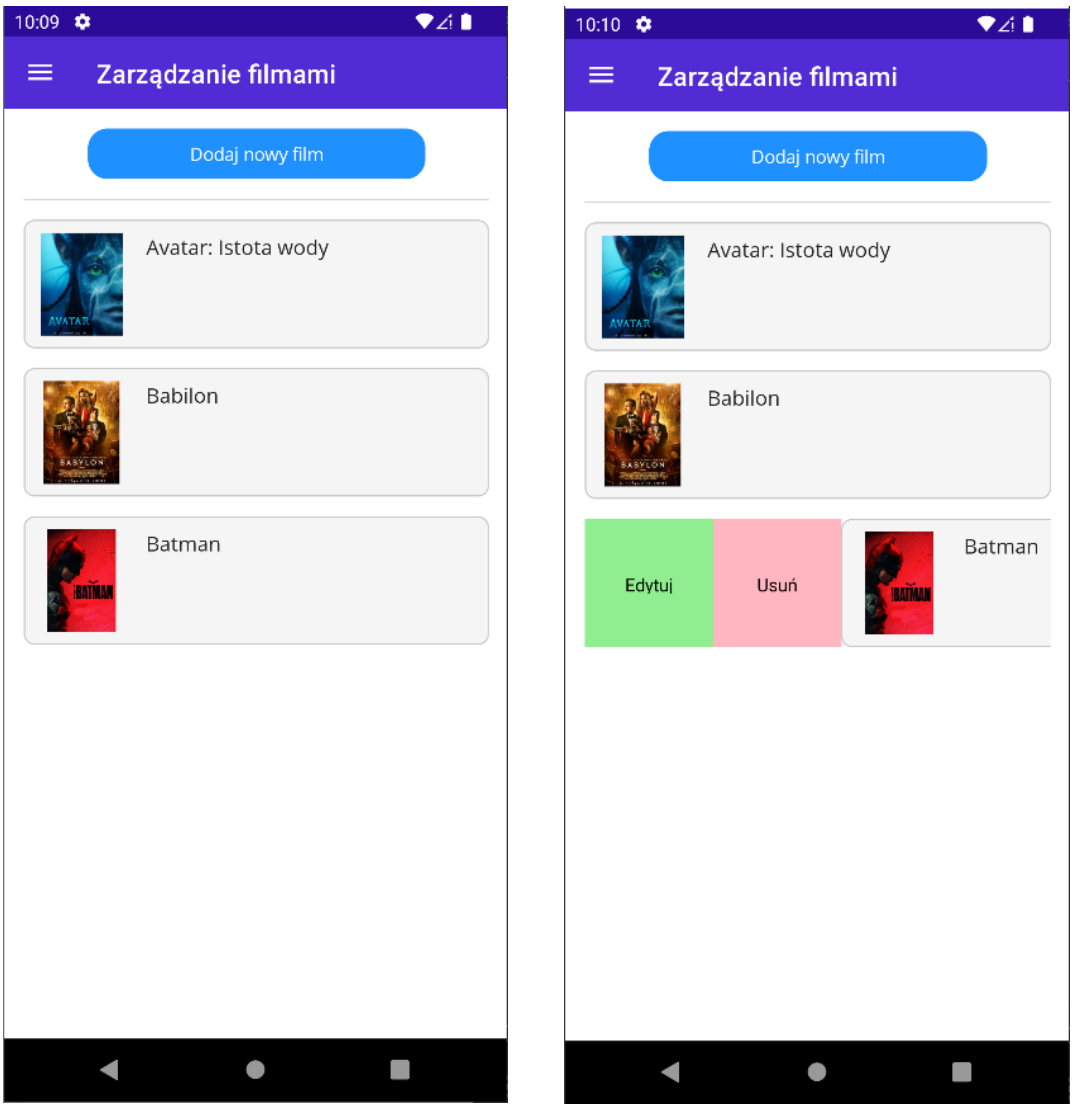


Rys. 24 - Widok szczegółów filmu [opracowanie własne]



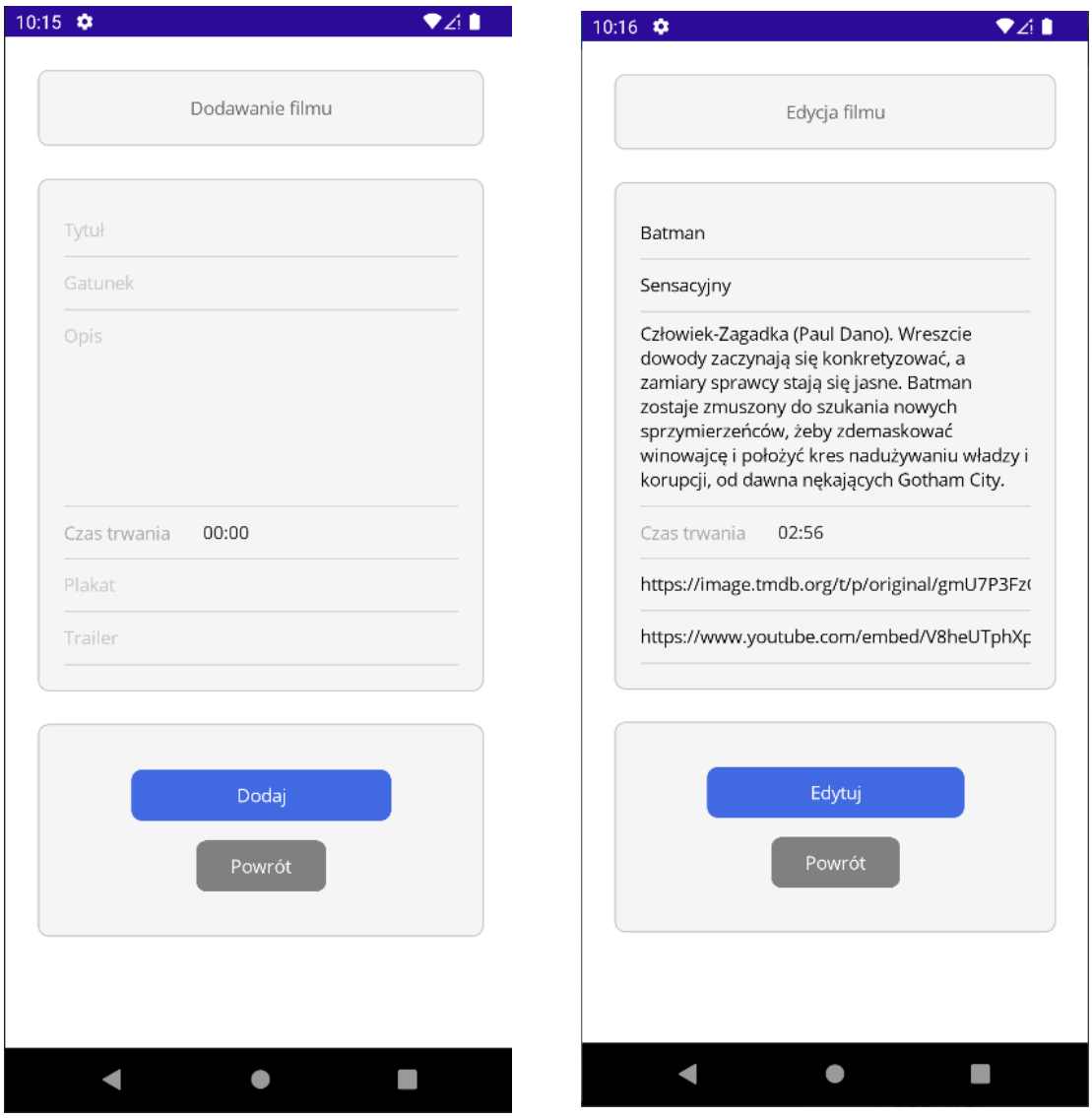
Rys. 25 - Widok menu użytkownika z uprawnieniami administratora [opracowanie własne]

Opisane powyżej interfejsy dotyczą klienta z rolą zwykłego użytkownika. W przypadku zalogowania do aplikacji klienta jako administrator, do jego dyspozycji pozostają dodatkowe funkcjonalności. Na rysunku 25 przedstawiono widok bocznego menu dla klienta z uprawnieniami administratora. Oprócz widoków dostępnych dla zwykłego użytkownika, dodatkowo wyświetlane są moduły zarządzania filmami oraz repertuarem.



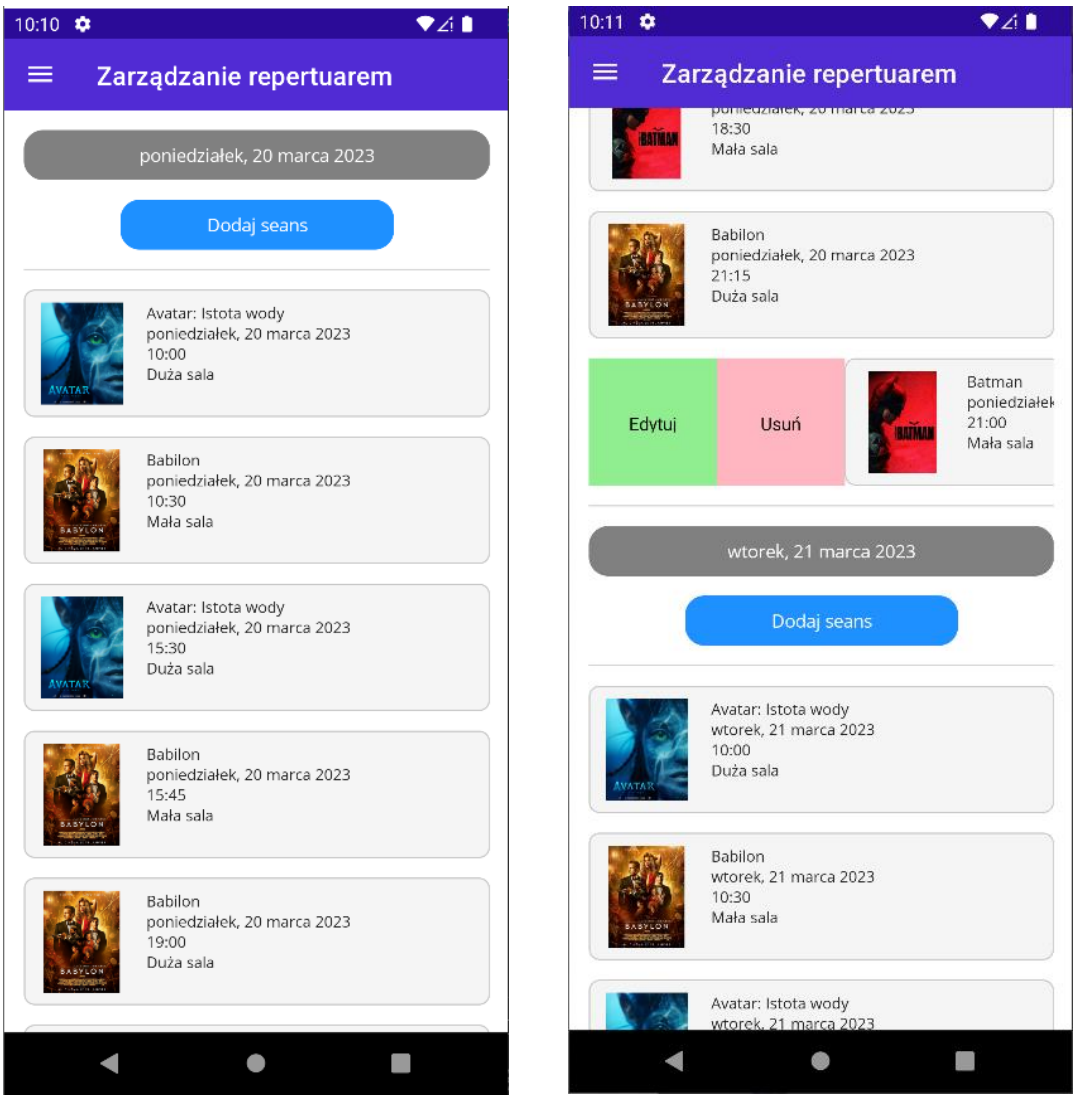
Rys. 26 - Ekran zarządzania filmami
[opracowanie własne]

Na rysunku 26 przedstawiono widok zarządzania filmami. Na ekranie w formie listy wyświetlane są filmy dodane do bazy danych. Na liście zaimplementowano mechanizm *SwipeView*, który po przesunięciu w bok danej pozycji z listy wyświetla opcje edycji oraz usunięcia filmu. W nagłówku widoku administrator ma możliwość kliknięcia przycisku *"Dodaj nowy film"*, który przeniesie go do widoku dodawania nowego filmu, przedstawionego na rysunku 27.



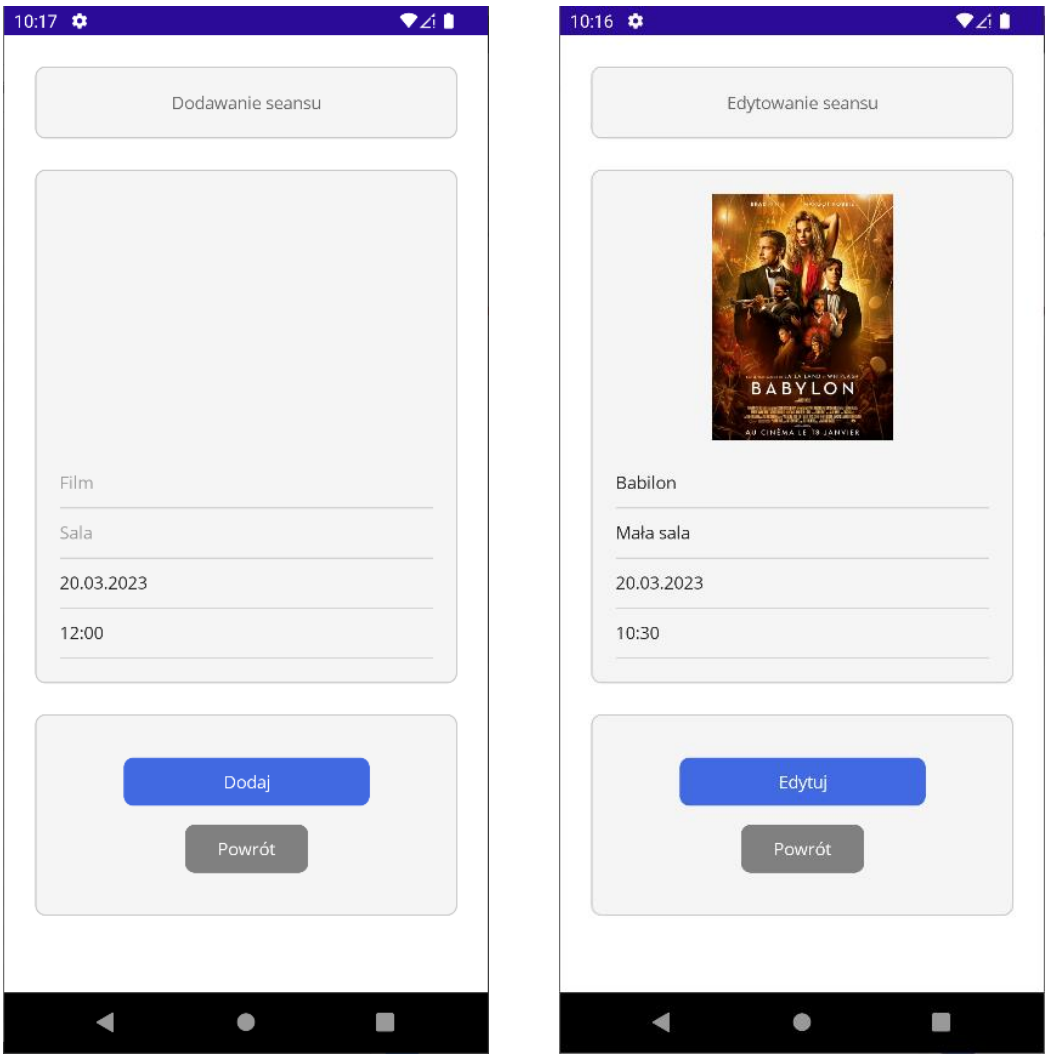
Rys. 27 - Widoki dodawania oraz edycji filmu
[opracowanie własne]

Widoki dodawania oraz edycji filmu, przedstawione na rysunku 27, posiadają odpowiadające sobie pola, takie jak tytuł, gatunek, opis, czas trwania oraz adresy URL plakatu i zwiastuna filmu. Po kliknięciu w przycisk edycji filmu, pola są automatycznie wypełniane dotychczasowymi danymi o filmie.



Rys. 28 - Ekran zarządzania repertuarem
[opracowanie własne]

Podobnie jak w przypadku ekranu zarządzania filmami, widok zarządzania repertuarem przygotowano w formie listy z implementacją mechanizmu *SwipeView*. Seanse w widoku są pogrupowane według daty. Nagłówek każdej grupy posiada przycisk *"Dodaj seans"*, który po kliknięciu przenosi administratora do ekranu dodawania seansu, przedstawionego na rysunku 29.



Rys. 29 - Widoki dodawania oraz edycji seansu
[opracowanie własne]

Widoki dodawania oraz edycji seansu, posiadają takie same pola. Po otwarciu widoku edycji, pola są automatycznie wypełniane danymi z bazy danych. Podczas dodawania lub edycji seansu, administrator może wybierać filmy i sale tylko i wyłącznie spośród tych dodanych do bazy danych. Po wybraniu filmu, automatycznie prezentowany jest jego plakat.

5. Wybrane elementy implementacji systemu

Podczas opracowywania systemu samoobsługowego kina dużą wagę przyłożono do automatyzacji i usprawnienia często powtarzających się zadań w celu zapewnienia przejrzystości kodu oraz ułatwienia rozbudowywania aplikacji.

5.1. Mapowanie modeli

Do mapowania encji bazy danych na modele DTO (ang. Data Transfer Object[27]) użyto narzędzia AutoMapper, które pozwala znacząco ograniczyć ilość kodu potrzebnego do obsłużenia odwzorowania obiektów. AutoMapper w swoim działaniu automatycznie mapuje publiczne właściwości klas pomiędzy dwoma obiektami, jeżeli nazwy oraz typy właściwości dla obu obiektów są zgodne i istnieją. W innym wypadku, jeśli zachodzi taka potrzeba, w konfiguracji AutoMappera można wskazać, w jaki sposób narzędzie powinno wykonać mapowanie dla danej właściwości.

W implementacji systemu przygotowano klasę *HallSeat*, która jest odwzorowaniem tabeli w bazie danych przechowującej informacje o miejscach na sali, a której właściwości odpowiadają poszczególnym kolumnom tabeli. Właściwości wirtualne z kolei stanowią o relacjach z innymi encjami bazy danych. Zastosowanie słowa kluczowego *virtual* pozwala na nadpisanie metod dostępowych (*get, set*) przez Entity Framework i użycie mechanizmu *"lazy loading"* podczas odwołania do właściwości.

```
public class HallSeat
{
    public int Id { get; set; }
    public int Row { get; set; }
    public int Column { get; set; }
    public int HallId { get; set; }
    public int SeatTypeId { get; set; }


    public virtual SeatType SeatType { get; set; }
    public virtual Hall Hall { get; set; }
    public virtual IEnumerable<Reservation> Reservations { get; set; }
    public virtual IEnumerable<Screening> Screenings { get; set; }

}
```

Listing 1 - Klasa HallSeat reprezentująca encję w bazie danych

Podczas rezerwacji, z serwera pobierane są informacje o miejscach na danej sali. W odpowiedzi do klienta zwracana jest lista obiektów klasy *SeatDto*, które zostały zmapowane bezpośrednio z encji *HallSeat*.

[27] obiekty, których zadaniem jest tylko i wyłącznie przetrzymywanie danych wysyłanych pomiędzy systemami

```
public class SeatDto
{
    public int Id { get; set; }
    public int Row { get; set; }
    public int Column { get; set; }
    public string SeatTypeName { get; set; }
    public bool IsReserved { get; set; }
    public double Price { get; set; }
}
```

Listing 2 - Klasa SeatDto, do której jest mapowany obiekt encji typu HallSeat

Dla tak zaimplementowanych klas przygotowano konfigurację AutoMappera, w której wskazano jak mapować obiekt klasy *HallSeat* na klasę *SeatDto*. Właściwości *Id, Row* oraz *Column* w klasie *SeatDto* zostaną automatycznie zmapowane z klasy *HallSeat*. Aby zmapować właściwości *SeatTypeName* oraz *Price* należy odwołać się do powiązanych encji w metodzie *MapFrom*. *IsReserved* przechowuje informację o tym, czy dane miejsce jest zarezerwowane. Mapowanie w tym wypadku polega na sprawdzeniu czy istnieją jakiekolwiek powiązane z miejscem rezerwacje.

```
CreateMap<HallSeat, SeatDto>()
   .ForMember(m => m.SeatTypeName,
              c => c.MapFrom(s => s.SeatType.Name))
   .ForMember(m => m.Price,
              c => c.MapFrom(s => s.SeatType.SeatPrice.WeekPrice))
   .ForMember(m => m.IsReserved,
              c => c.MapFrom(s => s.Reservations != null && s.Reservations.Any()));
```

Listing 3 - Przykład konfiguracji mapowania obiektów narzędzie AutoMapper

Użycie AutoMappera polega na przekazaniu do metody generycznej *Map* na obiekcie klasy *AutoMapper* obiektów, które chcemy mapować, ze wskazaniem na jaki typ klasy należy wykonać mapowanie.

```
//pobranie miejsc oraz powiązanych rezerwacji na dany seans
var hallSeats = ...
//mapowanie z użyciem AutoMappera
var seatsDtos = _mapper.Map<List<SeatDto>>(hallSeats);
```

Listing 4 - Użycie metody Map narzędzia AutoMapper

5.2. Middleware obsługi błędów (Error handling middleware)

Aby zapewnić stabilność oraz niezawodność działania systemu należy zadbać o poprawną obsługę błędów, które mogą wystąpić podczas jego działania. Błąd w trakcie pracy systemu sygnalizowany jest poprzez zgłoszenie wyjątku przez kod, który napotkał błąd. Mechanizm wyjątków pozwala na obsługę zgłoszonych błędów przez system.

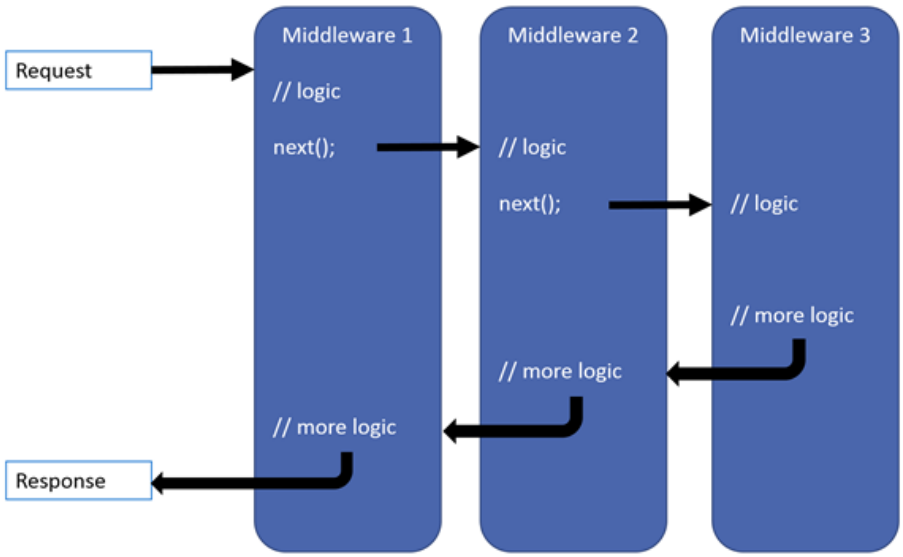
Obsługa wyjątków polega na umieszczeniu kodu, który potencjalnie może zgłosić wyjątek w bloku *try-catch*. Jednak dołączanie instrukcji *try-catch* do każdego miejsca w kodzie, w którym może wystąpić wyjątek byłoby rozwiązaniem mocno zawiłym. Aby temu zapobiec, a jednocześnie obsługiwać wyjątki zgłoszone z dowolnego miejsca w kodzie, zaimplementowano klasę *ErrorHandlingMiddleware*, która implementuje interfejs *IMiddleware*. Klasa została zarejestrowana w systemie w sposób, który pozwala na procesowanie przychodzących zapytań przez zaimplementowany middleware. Procesowanie zapytań polega na wywoływaniu kolejnych middleware API w bloku *try*. Jeżeli w dowolnym momencie działania aplikacji oraz w dowolnym miejscu w kodzie zostanie zgłoszony wyjątek, zostanie on przechwycony w bloku *catch* przez utworzony middleware. W listingu nr 5 przedstawiono przykładową implementację middleware, która może zostać rozszerzona o rozróżnienie obsługi konkretnych typów wyjątków.

```
public class ErrorHandlingMiddleware : IMiddleware
{
    public async Task InvokeAsync(HttpContext context, RequestDelegate next)
    {
        try
        {
            await next.Invoke(context);
        }
        catch (Exception ex)
        {
            context.Response.StatusCode = StatusCodes.Status500InternalServerError;
            await context.Response.WriteAsync("Something went wrong...");
        }
    }
}
```

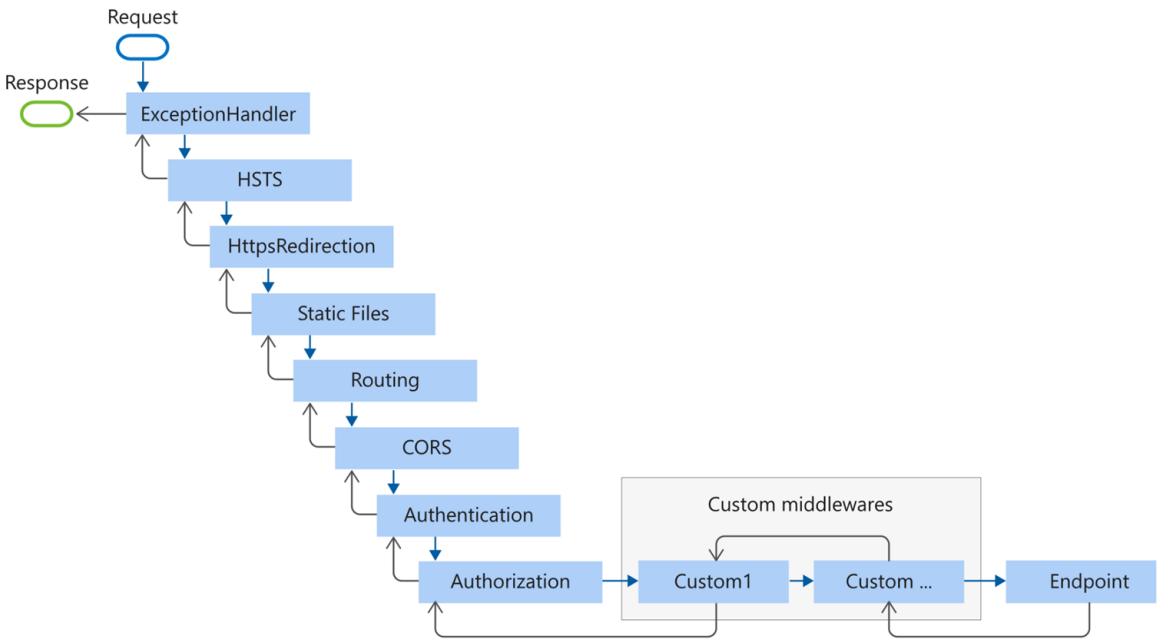
Listing 5 - Implementacja middleware obsługi błędów

Koncept *Middleware* w API sprowadza się do sekwencyjnego wywoływania kolejnych delegatów jeden po drugim, które mogą wywoływać akcje przed oraz po wywołaniu kolejnego delegata. Z tego powodu middleware obsługujące błędy powinny być umieszczone na początku kolejki, aby móc obsłużyć błędy, które wystąpiły w późniejszych etapach wywoływania innych

middleware. Schemat ideowy oraz hierarchię middleware w WebAPI przedstawiono na rysunkach 30 i 31.



Rys. 30 - Schemat ideowy działania middleware
[dokumentacja Microsoft]



Rys. 31 - Hierarchia middleware w ASP.NET Web API
[dokumentacja Microsoft]

5.3. Weryfikacja podpisu powiadomień PayU

Po utworzeniu nowej płatności na platformie PayU, usługa przesyła do API kina powiadomienia o aktualnym statusie płatności. Otrzymywanie powiadomień odbywa się poprzez wysyłanie zapytań POST przez PayU na adres wskazany w parametrze *notifyUrl* zapisanym w pliku json przesłanym do usługodawcy podczas tworzenia nowej płatności.

Aby zapewnić zaufaną komunikację pomiędzy kinem, a PayU należy poddać weryfikacji wartość nagłówka *OpenPayu-Signature* dołączonego do notyfikacji. Weryfikacja zaczyna się od wyizolowaniu z nagłówka wartości *signature*, a następnie połączenia treści otrzymanego powiadomienia z wartością klucza *second_key* wygenerowanego przez PayU na platformie menadżera (rysunek 8). Dla tak wyznaczonego łańcucha znaków należy zastosować funkcję haszującą. Rodzaj algorytmu użytego do szyfrowania jest zawarty w nagłówku zapytania i jest nim najczęściej MD5. Ostatecznie należy porównać wartość wyizolowanej wartości *signature* z otrzymanym podpisem po haszowaniu.

Implementację weryfikacji podpisu dla przychodzących powiadomień zawarto w specjalnie dodanej polityce wymagań, która jest obsługiwana przez middleware autoryzacji. Nowa polityka wymaga dołączenia klasy wymagań (*PayuNotificationSignatureRequirement*) oraz klasy obsługi wymagań (*PayuNotificationSignatureHandler)*. Aby middleware odpowiedzialny za autoryzację mógł korzystać z dodanej polityki, należy zarejestrować serwisy i wymagania, jak pokazano na listingu 6.

```
services.AddAuthorization(options =>
{
    options.AddPolicy("PayuNotificationSignaturePolicy",
        policy => policy.Requirements.Add(new PayuNotificationSignatureRequirement()));
});

services.AddScoped<IAuthorizationHandler, PayuNotificationSignatureHandler>();
```

Listing 6 - Dodanie polityki autoryzacji oraz rejestracja serwisów wymaganych do jej działania

Middleware autoryzacji podczas weryfikacji dostępu do zasobu wywołuje metodę *HandleRequirementAsync* z intefejsu *AuthorizationHandler*, którą implementuje klasa *PayuNotificationSignatureHandler.* Implementację metody *HandleRequirementAsync* przedstawiono w listingu nr 7.

```
protected override Task HandleRequirementAsync(AuthorizationHandlerContext context,
PayuNotificationSignatureRequirement requirement)
{
    _httpContext.Request.Headers.TryGetValue("OpenPayu-Signature", out var header);

    var values = header.ToString()
                       .Split(';', options: StringSplitOptions.TrimEntries);
    var incomingSignature = String.Concat(values.Where(e =>
                                                       e.StartsWith("signature"))
                                    .FirstOrDefault()
                                    .ToString()
                                    .SkipWhile(c => !c.Equals('=')).Skip(1));

    if (!string.IsNullOrEmpty(incomingSignature))
    {
        if (IsValidSignature(incomingSignature).Result)
        {
            context.Succeed(requirement);
        }
    }
    else
    {
        context.Fail();
    }

    return Task.FromResult(0);
}
private async Task<bool> IsValidSignature(string incomingSignature)
{
    _httpContext.Request.EnableBuffering();

    using (var reader = new StreamReader(_httpContext.Request.Body, Encoding.UTF8,
false, 1024, true))
    {
        var body = await reader.ReadToEndAsync();
        _httpContext.Request.Body.Seek(0, SeekOrigin.Begin);

        var toBeSigned = body + SECRET;
        var signature =
Convert.ToHexString(MD5.HashData(Encoding.UTF8.GetBytes(toBeSigned))).ToLower();

        if (incomingSignature == signature)
        {
            return true;
        }
    }

    return false;
}
```

Listing 7 - Implementacja własnej polityki autoryzacji w metodzie HandleRequirementAsync

Aby autoryzacja zadziała tylko i wyłącznie dla endpointu, który jest odpowiedzialny za obsługę przychodzących powiadomień od PayU, nad jego implementacją należy umieścić adnotację informującą o polityce, która ma zostać użyta do autoryzacji. Sytuacja została zaprezentowana na listingu 8.

```
[HttpPost]
[Route("payu/status")]
[Authorize(Policy = "PayuNotificationSignaturePolicy")]
public ActionResult UpdateOrderStatus([FromBody] PayuOrderStatusWrapperDto dto)
{
    _payuService.UpdateOrderStatus(dto);
    return Ok();
}
```

Listing 8 - Endpoint odpowiedzialny za przetwarzanie notyfikacje o statusie płatności, z dodaną adnotacją o użytej polityce autoryzacji

5.4. Skanowanie biletów

Skanowanie wygenerowanych biletów odbywa się poprzez odczytanie kodu QR z wyświetlacza telefonu. Aby zasymulować urządzenie skanujące w kinie, zaimplementowano oraz uruchomiono na laptopie skrypt obsługujący skanowanie biletów. System wykorzystując dostępną kamerę, wykonuje co określony interwał czasu zdjęcia. Z użyciem biblioteki OpenCV przechwycone klatki są analizowane pod kątem obecności kodów QR. W przypadku odnalezienia na zdjęciu kodu QR, jest on odczytywany, a na serwer wysyłane jest zapytanie o skasowanie biletu. Jeśli obecnie grany seans odpowiada sensowi przypisanemu do wejściówki, bilet jest kasowany, a w aplikacji klienckiej użytkownika następuje jego wyszarzenie. Skrypt symulujący kasowanie biletów w kinie został przedstawiony na listingu 9.

```
import cv2
import time
import requests
import json

def handle_ticket(data):
    url = 'https://cienam-api-app.azurewebsites.net/api/ticket'
    headers = {'Content-type': 'application/json','Authorization': 'Bearer
{token}'} #auth token
    payload = {'code': data}
    response = requests.post(url, data=json.dumps(payload), headers=headers)

    if response.status_code == 200:
        print("Bilet został skasowany")
    else:
        print("Nie można skasować biletu")


cap = cv2.VideoCapture(0)
detector = cv2.QRCodeDetector()

while True:
    _, img = cap.read()

    data, bbox, _ = detector.detectAndDecode(img)

    if data:
        print(data)
        try:
            handle_ticket(data)
            time.sleep(3)
        except:
            print("exception")
            pass

    time.sleep(0.1)

    cv2.imshow("scanner", img)
    if cv2.waitKey(1) == ord("q"):
        break

cap.release()
cv2.destroyAllWindows()
```

Listing 9 - skrypt symulujący kasowanie biletów w kinie

6. Testy

Rozwijanie aplikacji wiąże się z ciągłymi zmianami w jej kodzie źródłowym. Testowanie wprowadzanych zmian oraz ich wpływu na istniejący już kod ma krytyczne znaczenie w kontekście utrzymywania aplikacji. Aplikacja kliencka oraz serwerowa w trakcie implementacji były poddawana ciągłym testom manualnym, czyli ręcznemu testowaniu opracowanych części kodu. Aby usprawnić proces testowania, przygotowano automatyczne testy integracyjne, które badają poprawność implementacji głównych funkcji systemu.

6.1. Testy integracyjne

Implementacja testów integracyjnych dotyczyła głównie kontrolerów WebAPI, które w swoim działaniu wymagały integrowania ze sobą różnych typów serwisów oraz mechanizmów, np. opisywany w poprzednim rozdziale middleware. Aby podczas testów nie operować na produkcyjnej bazie danych, w konstruktorze klasy testowej ustawienia dostępu do bazy są nadpisywane na rzecz mechanizmu *InMemoryDatabase*, co pozwala na utworzenie testowej bazy danych w pamięci WebAPI. Na listingu nr 10 przedstawiono kod odpowiedzialny za nadpisanie ustawień.

```
public class UserControllerTests : IClassFixture<WebApplicationFactory<Startup>>
{
    private WebApplicationFactory<Startup> _factory;

    public UserControllerTests(WebApplicationFactory<Startup> factory)
    {
        _factory = factory.WithWebHostBuilder(builder =>
        {
            builder.ConfigureServices(services =>
            {
                var dbContextOptions = services.SingleOrDefault(
                    service =>
                        service.ServiceType == typeof(DbContextOptions<ApiDbContext>));
                services.Remove(dbContextOptions);
                services.AddDbContext<ApiDbContext>(
                    options => options.UseInMemoryDatabase("TestDb"));
            });
        });
    }
    ... //testy kontrolera UserController
}
```

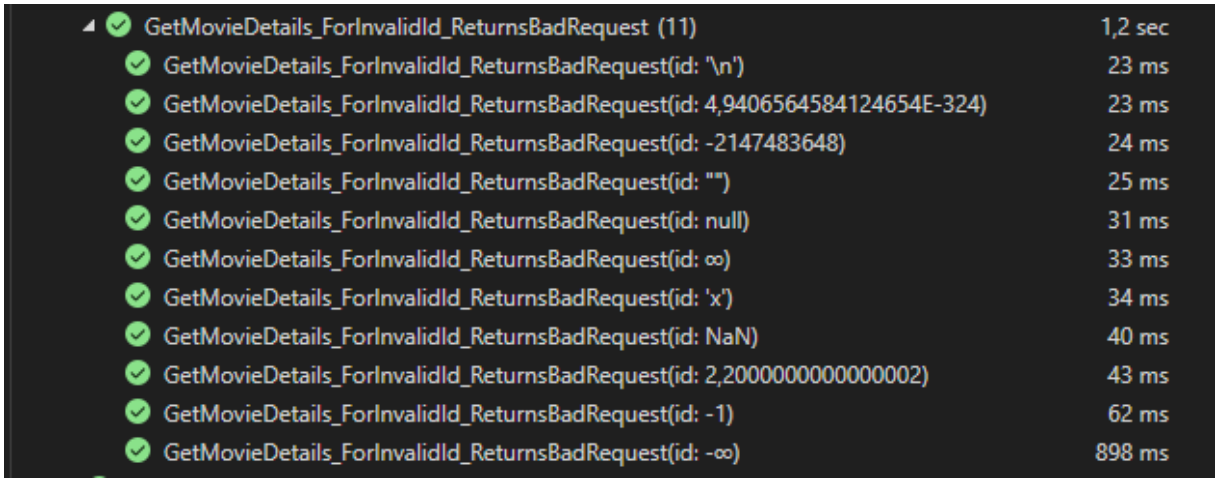
Listing 10 - Nadpisanie ustawień bazy danych dla testów integracyjnych

Test zwracanego statusu HTTP dla poprawnego logowania zaprezentowano na listingu 11.

```
[Fact]
public async Task
LoginUser_WithActiveAccount_WithValidModelAndCredentials_ReturnsOk()
{
    //arrange
    var client = _factory.CreateClient();
    var password = "veryStrongPassword";

    var user = new User()
    {
        Name = "TestName",
        Surname = "TestSurname",
        Email = "test.user@gmail.com",
        IsActivated = true,
        Role = new Role()
        {
            Name = "User"
        }
    };

    var scopeFactory = _factory.Services.GetService<IServiceScopeFactory>();
    using var scope = scopeFactory.CreateScope();
    var dbContext = scope.ServiceProvider.GetService<ApiDbContext>();

    var passwordHasher = scope.ServiceProvider.GetService<IPasswordHasher<User>>();
    user.PasswordHash = passwordHasher.HashPassword(user, password);

    dbContext.Users.Add(user);
    dbContext.SaveChanges();

    var loginUserDto = new LoginUserDto()
    {
        Email = "test.user@gmail.com",
        Password = password
    };

    var json = JsonConvert.SerializeObject(loginUserDto);
    var httpContent = new StringContent(json, UnicodeEncoding.UTF8,
"application/json");

    //act
    var response = await client.PostAsync("/api/account/login", httpContent);

    //assert
    response.StatusCode.Should().Be(System.Net.HttpStatusCode.OK);
}
```

Listing 11 - Test zwracanego statusu HTTP dla poprawnego logowania

Przypadek testowy z listingu 11 dotyczy użytkownika, który posiada zarejestrowane konto oraz je aktywował, a dane wprowadzone w formularzu logowania są poprawne. W sekcji *arrange* do bazy danych w pamięci API dodawany jest testowy użytkownik, którego konto jest oznaczone jako aktywowane. Następnie utworzony oraz serializowany zostaje model *LoginUserDto* zawierający email oraz hasło dodanego do bazy użytkownika. W sekcji *act* z użyciem klienta Http do API zostaje wysłane żądanie typu POST z modelem w ciele zapytania. Sekcja *assert* z użyciem metod rozszerzających z pakietu *FluentAssertions* weryfikuje, czy zwrócony kod statusu odpowiedzi jest kodem 200 OK. Metoda testowa zawarta w listingu 11 jest oznaczona atrybutem *[Fact]*, czyli dotyczy jednego konkretnego przypadku testowego i nie przyjmuje na wejściu argumentów parametryzujących test. Przeciwnym przypadkiem są metody testowe oznaczone atrybutem *[Theory]*, które mogą przyjmować na wejściu argumenty wpływające na wynik testu. Poniżej przedstawiono test dla metody *GetMovieDetails* z klasy *MovieController*. Metoda testowa jako parametr *id* przyjmuje obiekt klasy *object*, który jest wykorzystywany w ścieżce zapytania do API. Brak powiązania parametru z konkretnym typem prostym pozwala przekazywać jako argumenty zmienne różnych typów, np. double, int, string, które dopiero w metodzie testowej są konwertowane do typu string. Przypadki testowe, czyli parametry przekazywane do testu są umieszczone nad definicją testu w adnotacji *[InlineData()]*. Metoda testowa wykonuje się dla każdego z przypadków testowych z osobna. Wyniki testów przedstawiono na rysunku 32.



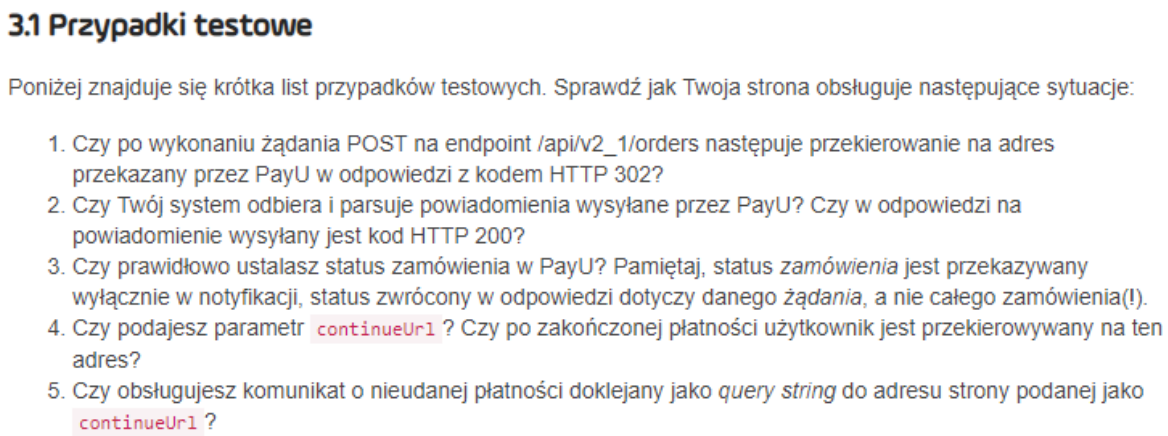
Rys. 32 - Wyniku testów dla listingu 12
[środowisko Visual Studio]

```
[Theory]
[InlineData(int.MinValue)]
[InlineData(null)]
[InlineData('x')]
[InlineData(-1)]
[InlineData('\n')]
[InlineData(double.NegativeInfinity)]
[InlineData(double.PositiveInfinity)]
[InlineData(double.NaN)]
[InlineData(double.Epsilon)]
[InlineData("")]
[InlineData(2.2)]

public async Task GetMovieDetails_ForInvalidId_ReturnsBadRequest(object id)
{
    //arrange
    var client = _factory.CreateClient();
    //act
    var response = await client.GetAsync($"/api/movie/details/{id}");
    //assert
    response.StatusCode.Should().Be(System.Net.HttpStatusCode.BadRequest);
}
```

Listing 12 - Test zwracanego statusu HTTP dla niepoprawnych wartości parametru id

6.2. Przypadki testowe integracji PayU

Dostawca usług płatniczych w dokumentacji do swojego API wskazuje na przypadki testowe, które mogą być krytyczne dla działania system.



3.1 Przypadki testowe

Poniżej znajduje się krótka list przypadków testowych. Sprawdź jak Twoja strona obsługuje następujące sytuacje:

1. Czy po wykonaniu żądania POST na endpoint /api/v2_1/orders następuje przekierowanie na adres przekazany przez PayU w odpowiedzi z kodem HTTP 302?
2. Czy Twój system odbiera i parsuje powiadomienia wysyłane przez PayU? Czy w odpowiedzi na powiadomienie wysyłany jest kod HTTP 200?
3. Czy prawidłowo ustalasz status zamówienia w PayU? Pamiętaj, status *zamówienia* jest przekazywany wyłącznie w notyfikacji, status zwrócony w odpowiedzi dotyczy danego *żądania*, a nie całego zamówienia(!).
4. Czy podajesz parametr `continueUrl` ? Czy po zakończonej płatności użytkownik jest przekierowywany na ten adres?
5. Czy obsługujesz komunikat o nieudanej płatności doklejany jako *query string* do adresu strony podanej jako `continueUrl` ?

Rysuenk 33 - Przypadki testowe proponowane przez PayU [dokumentacja PayU REST API]

Wszystkie przypadki testowe zostały sprawdzone oraz prawidłowo obsłużone, za wyjątkiem przypadku nr 5 z rysunku 33, gdzie celowo status płatności można sprawdzić tylko i wyłącznie z poziomu aplikacji klienta lub linku przesłanego przez PayU na adres mailowy.

7. Podsumowanie

W ramach pracy udało się zrealizować cel i założenia przedstawione w rozdziale pierwszym, czyli zaimplementować system usprawniający proces zakupu biletów do kina obejmujący moduły serwera, aplikacji klienckiej, skryptu kasowania biletów oraz bazy danych. Poszczególne części systemu zostały zaimplementowane głównie z wykorzystaniem technologii oraz narzędzi oferowanych przez firmę Microsoft. Aplikacja serwerowa została przygotowana w formie WebAPI z wykorzystaniem frameworka ASP.NET. Aplikacja kliencka została zaimplementowana w technologii mobilnej na bazie platformy .NET MAUI z wykorzystaniem wzorca Model-View-ViewModel. Do przechowywania danych niezbędnych do funkcjonowania systemu wykorzystano silnik bazy danych MS SQL Server. Skrypt symulujący kasowanie biletów w kinie przygotowano z użyciem języka Python oraz wysoko zoptymalizowanej biblioteki przetwarzania obrazów OpenCV. System został zintegrowany z platformą testową *Sandbox* dostawcy usług płatniczych PayU. Aplikacja serwerowa oraz baza danych zostały udostępnione na platformie Azure w celu umożliwienia komunikacji z publicznym REST API PayU. Użytkownicy systemu zostali podzieleni na 2 grupy - klientów oraz administratorów. Klienci korzystający z aplikacji mogą zakupić bilet na seans prezentowany w repertuarze, przeglądać historię swoich zamówień oraz prezentować w kinie wygenerowane bilety. Grupa administratorów posiada dodatkowo uprawnienia dodawania, usuwania oraz edycji filmów i repertuaru. Przedstawione funkcjonalności systemu zostały pomyślnie przetestowane z wykorzystaniem testów automatycznych oraz manualnych.

System posiada duży potencjał dalszego rozwoju oraz usprawnień. Architektura bazy danych pozwala na rozbudowę aplikacji o nowe kina i sale kinowe. Do dyspozycji administratora można pozostawić możliwość zarządzania rezerwacjami oraz miejscami na salach kinowych. Dla klientów można zaimplementować interfejs wyszukiwania i filtrowania seansów. Aplikacja mobilna mogłaby zostać spersonalizowana pod kątem motywu UI oraz języka. Istnieje bardzo dużo możliwych ścieżek rozwoju opracowanego systemu oraz aplikacji.

Przygotowana praca pozwoliła bardzo dobrze poznać technologie oferowane przez firmę Microsoft oraz poszerzyć wiedzę z zakresu implementacji WebAPI, aplikacji mobilnych, testowania oprogramowania oraz integracji systemów zewnętrznych.

Bibliografia

1. “ASP.NET overview.” *Microsoft Learn*, 29 September 2022, https://learn.microsoft.com/en-us/aspnet/overview. Accessed 10 January 2023.
2. “C# Version History: Examining the Language Past and Present.” *NDepend*, 6 July 2017, https://blog.ndepend.com/c-versions-look-language-history/. Accessed 10 January 2023.
3. “Entity Framework.” *Microsoft Learn*, 21 July 2022, https://learn.microsoft.com/en-us/aspnet/entity-framework. Accessed 10 January 2023.
4. “Get Started With The OpenAPI Specification.” *Swagger*, https://swagger.io/solutions/getting-started-with-oas/. Accessed 10 January 2023.
5. “The history of C# - C# Guide.” *Microsoft Learn*, 9 December 2022, https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history. Accessed 10 January 2023.
6. “Introducing .NET MAUI - One Codebase, Many Platforms.” *Microsoft Developer Blogs*, 23 May 2022, https://devblogs.microsoft.com/dotnet/introducing-dotnet-maui-one-codebase-many-platforms/. Accessed 12 January 2023.
7. “Jak bracia Lumiére opatentowali kinematograf | Portal historyczny Histmag.org - historia dla każdego!” *Histmag*, 12 February 2011, https://histmag.org/Jak-bracia-Lumire-opatentowali-kinematograf-5172. Accessed 10 January 2023.
8. “JSON Web Token Introduction - jwt.io.” *JWT.io*, https://jwt.io/introduction. Accessed 10 January 2023.
9. “Kinematograf, czyli początki kina. Krótka historia wynalazku i ciekawostki.” *Focus.pl*, 22 September 2020, https://www.focus.pl/artykul/kinematograf-czyli-poczatki-kina-krotka-historia-wynalazku-i-ciekawostki. Accessed 10 January 2023.

10. "Kod QR – Wikipedia, wolna encyklopedia." *Wikipedia*, https://pl.wikipedia.org/wiki/Kod_QR. Accessed 12 January 2023.

11. Miller, Rowan. "Code First to a New Database EF6." *Microsoft Learn*, 14 October 2020, https://learn.microsoft.com/en-us/ef/ef6/modeling/code-first/workflows/new-database. Accessed 10 January 2023.

12. "Model-View-ViewModel (MVVM)." *Microsoft Learn*, 4 November 2022, https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm. Accessed 21 February 2023.

13. "OpenCV - Introduction." 6 January 2023, https://docs.opencv.org/4.7.0/d1/dfb/intro.html. Accessed 11 January 2023.

14. "PayU - Overview." *PayU*, https://developers.payu.com/pl/overview.html. Accessed 11 January 2023.

15. "The Python Tutorial — Python 3.11.1 documentation." *Python Docs*, https://docs.python.org/3/tutorial/index.html. Accessed 10 January 2023.

16. "REST API." *PayU*, https://developers.payu.com/pl/restapi.html#overview. Accessed 21 February 2023.

17. "SQL Server technical documentation." 6 January 2023, https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver15. Accessed 11 January 2023.

18. "The top programming languages | The State of the Octoverse." *GitHub Octoverse*, 2022, https://octoverse.github.com/2022/top-programming-languages. Accessed 10 January 2023.

19. "What is .NET? An open-source developer platform." *Microsoft .NET*, https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet. Accessed 10 January 2023.

20. “What is .NET MAUI? - .NET MAUI.” *Microsoft Learn*, 8 November 2022, https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-7.0. Accessed 10 January 2023.

21. “xUnit.net.” *Home > xUnit.net*, https://xunit.net/. Accessed 10 January 2023.

Spis rysunków

Spis listingów