P O L I T E C H N I K A Ś W I Ę T O K R Z Y S K A

Wydział Elektrotechniki, Automatyki i Informatyki

Maksymilian Sowula

Paweł Marek

Jakub Szczur

Daniel Cieślak

Numer grupy dziekańskiej: 1ID21A

Wypożyczalnia płyt DVD

Inteligentne Usługi Informacyjne – Projekt

1. Skład zespołu oraz podział prac

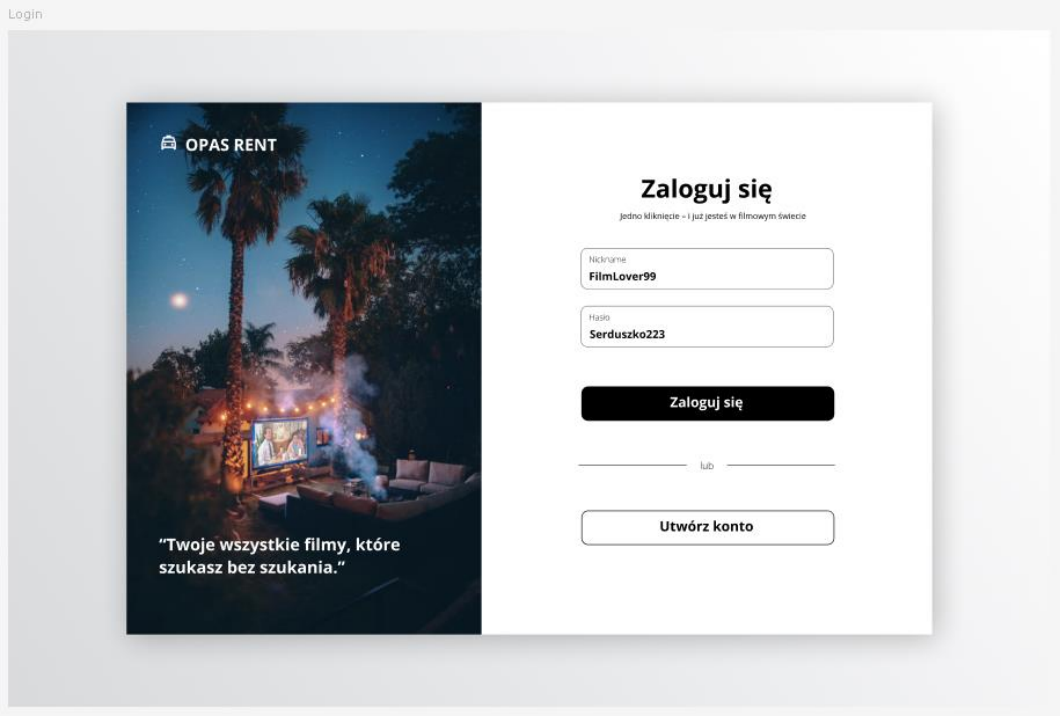
Tabela 1.1. Skład zespołu oraz procentowy podział prac

Członek zespołu	Wykonywana praca	Udział procentowy
Maksymilian Sowula	Backend, Testy	25%
Paweł Marek	Frontend, Dokumentacja techniczna	25%
Jakub Szczur	Frontend, UX/UI	25%
Daniel Cieślak	Backend, Sprawozdanie	25%

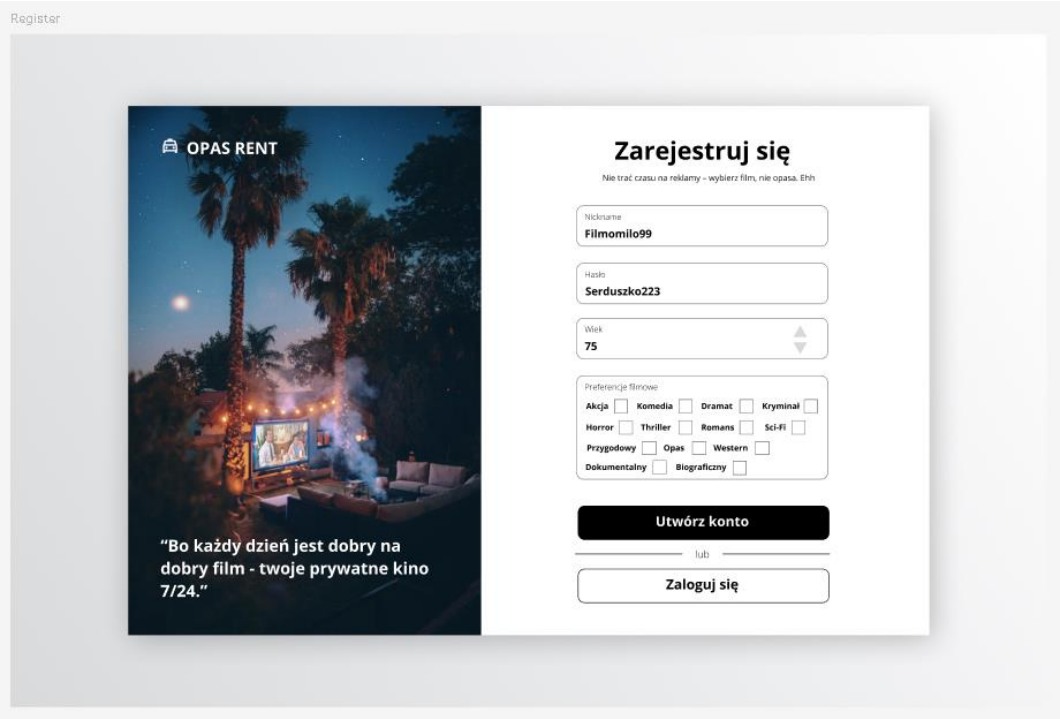
2. Zrzuty ekranu aplikacji

2.1. Makiety aplikacji

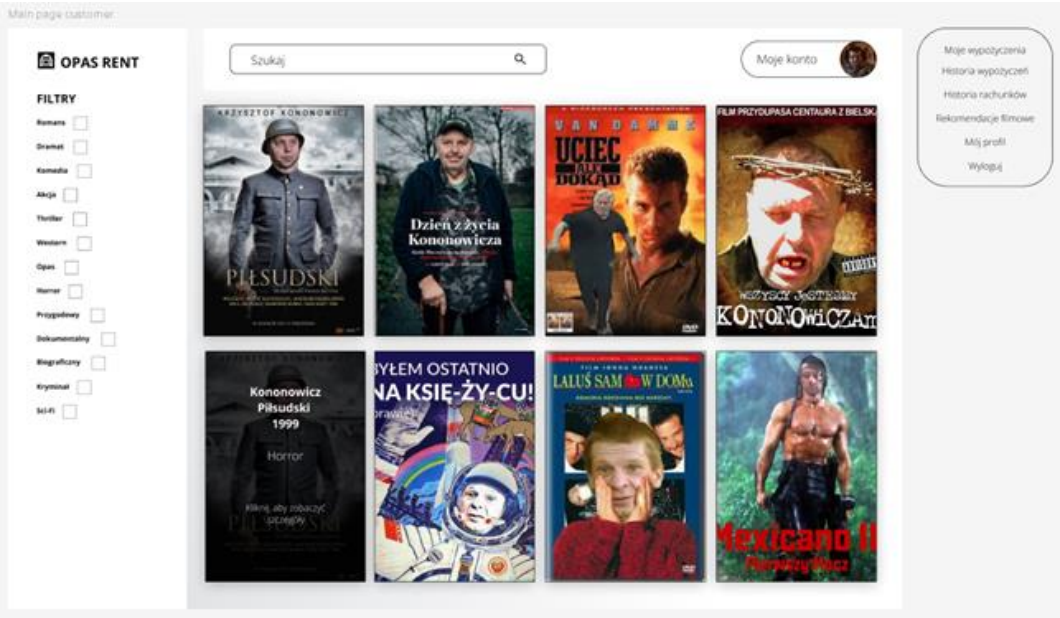
Makiety ekranów aplikacji zostały wykonane za pomocą narzędzia Figma i zostały przedstawione na zdjęciach poniżej.



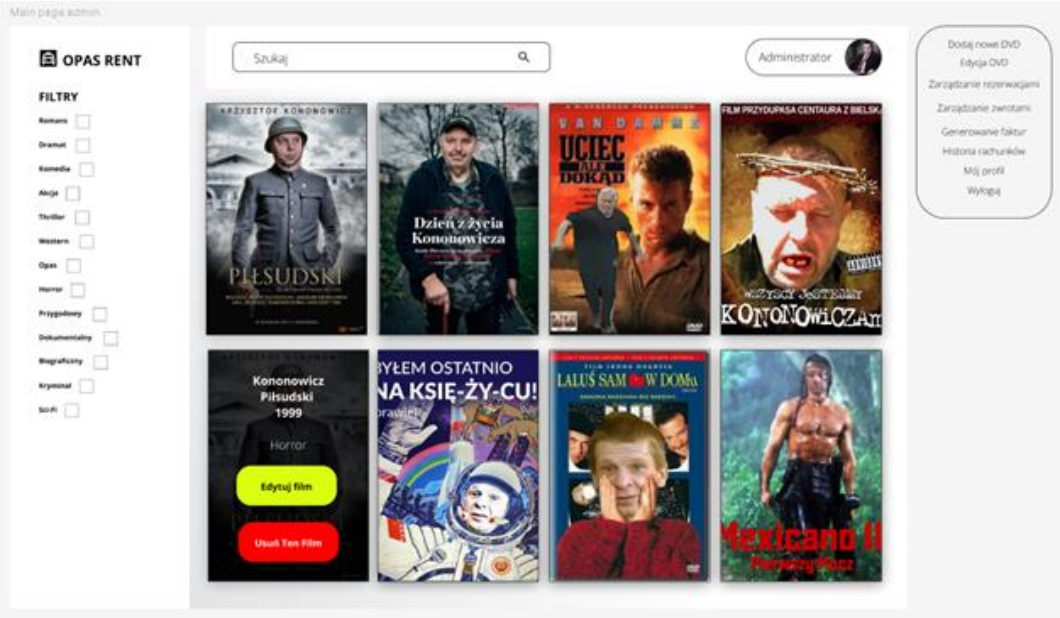
Rysunek 2.1. Makieta ekranu logowania



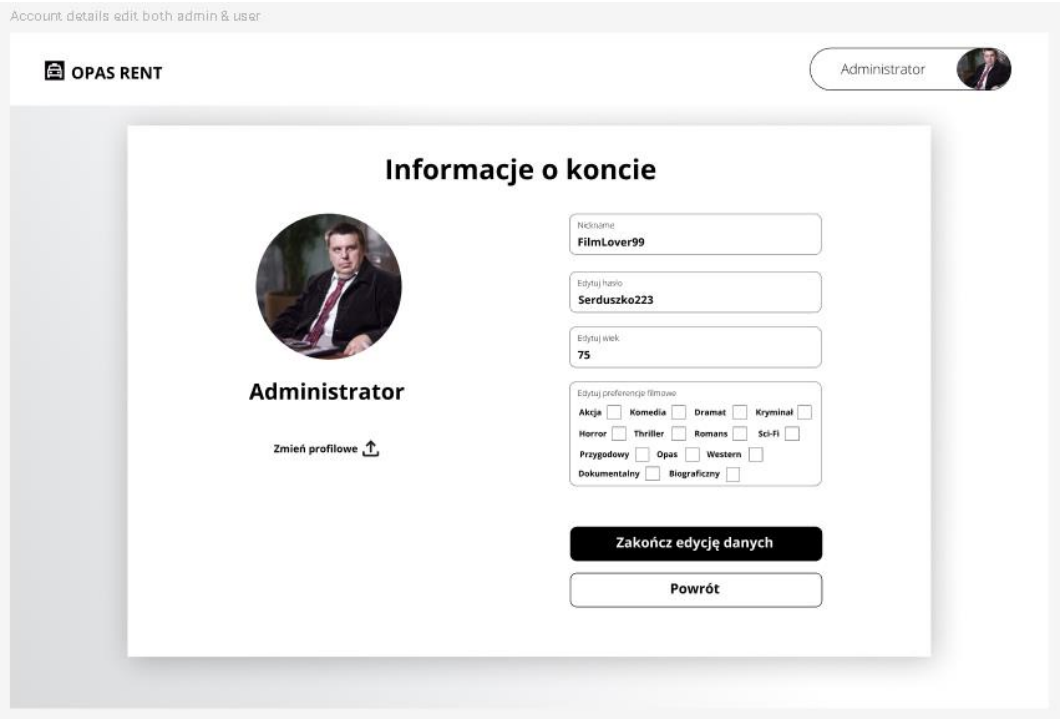
Rysunek 2.2. Makieta ekranu rejestracji



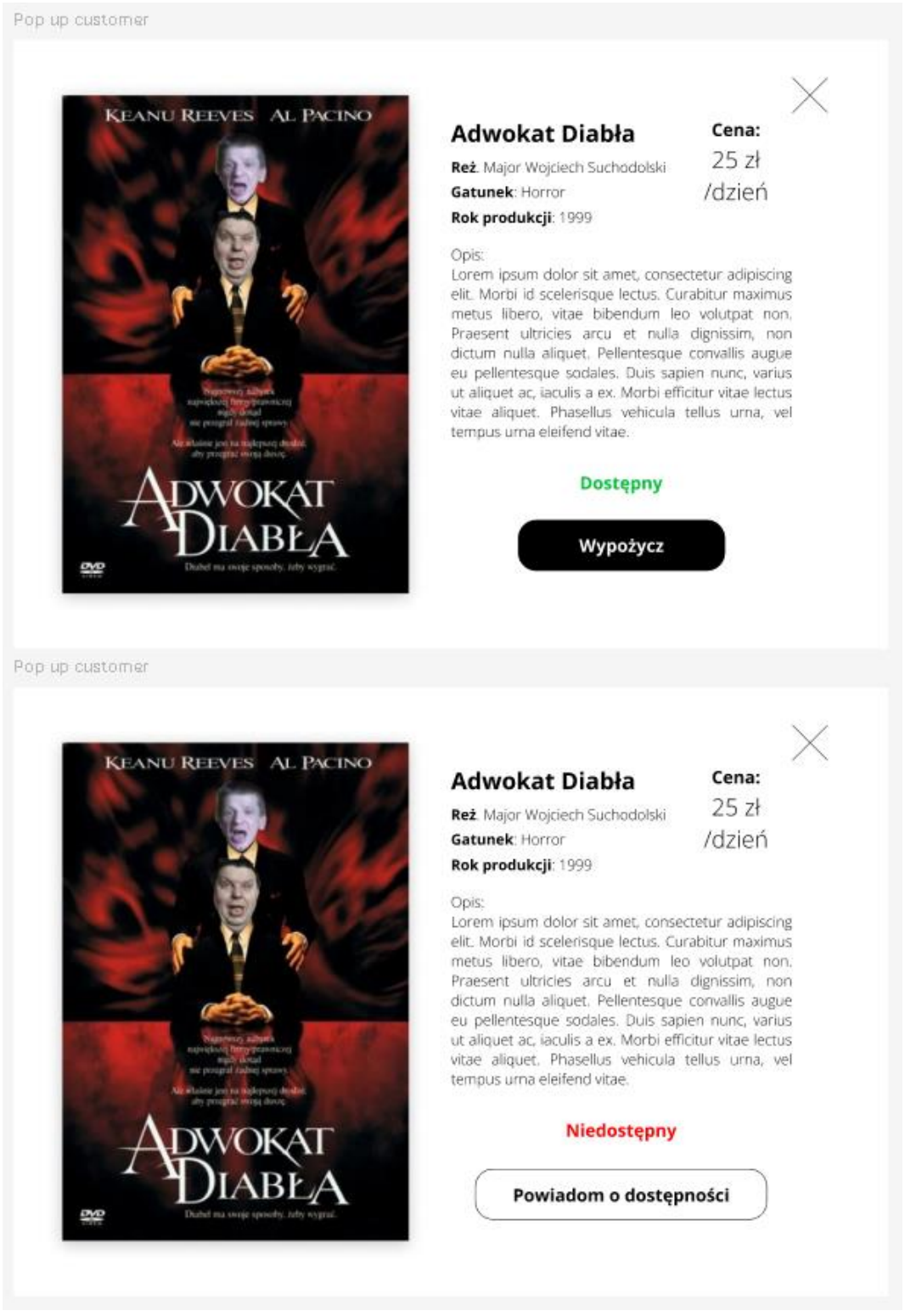
Rysunek 2.3. Makieta panelu po stronie klienta



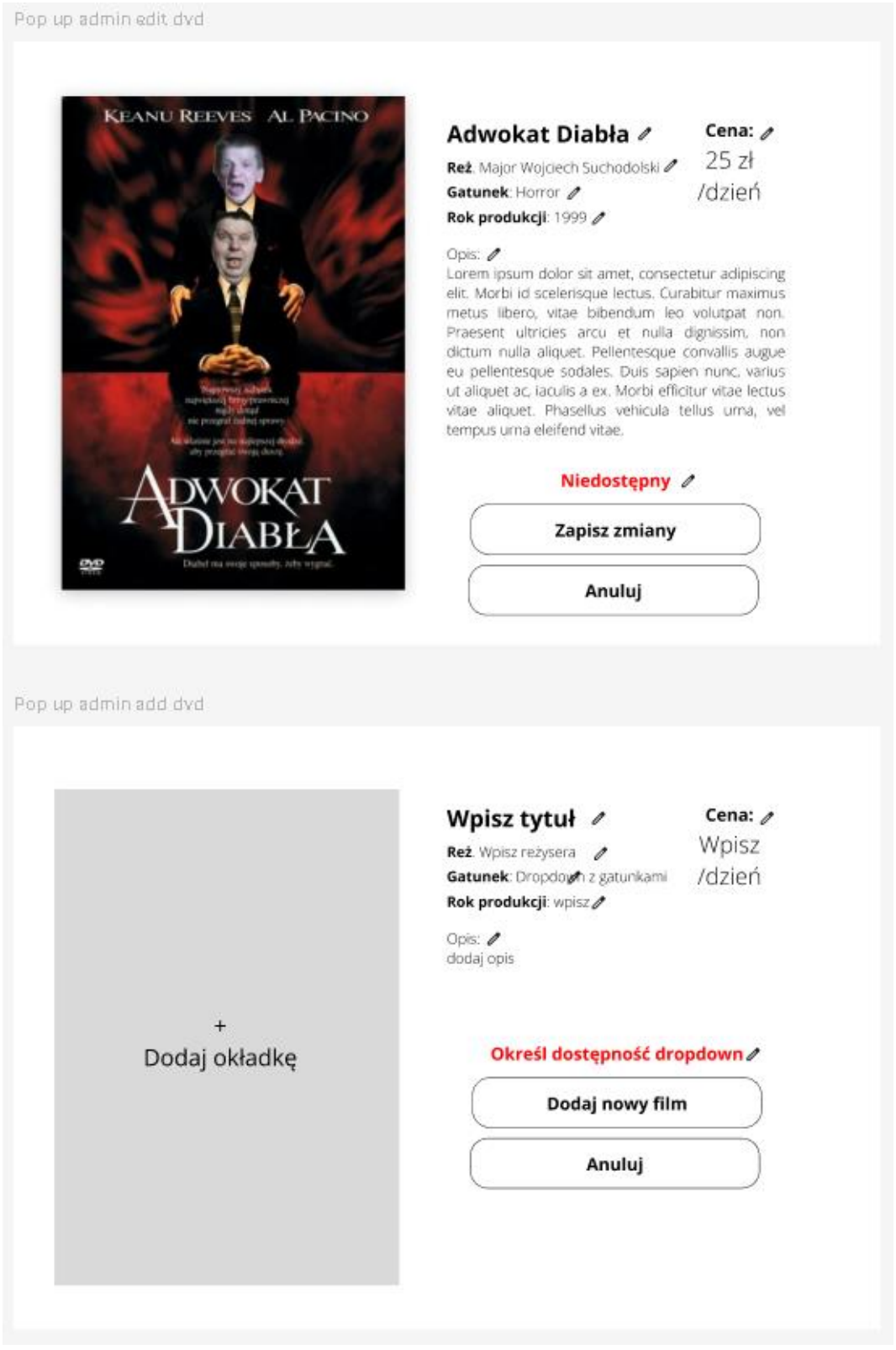
Rysunek 2.4. Makieta panelu po stronie administratora



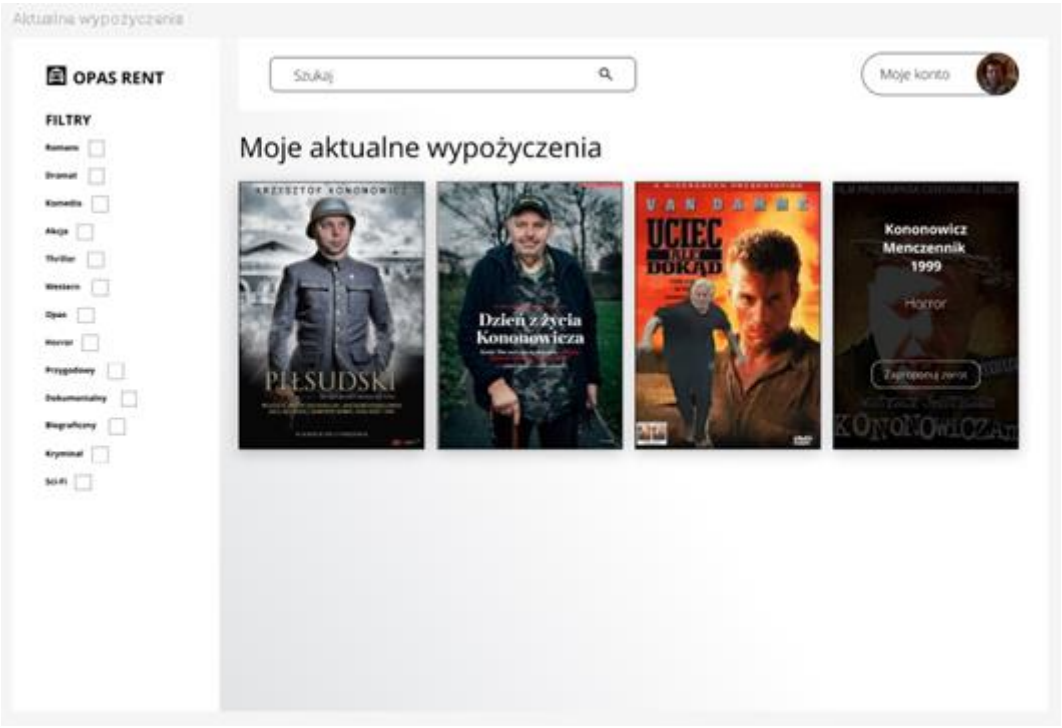
Rysunek 2.5. Makieta ekranu edycji informacji o koncie (dla klienta oraz administratora)



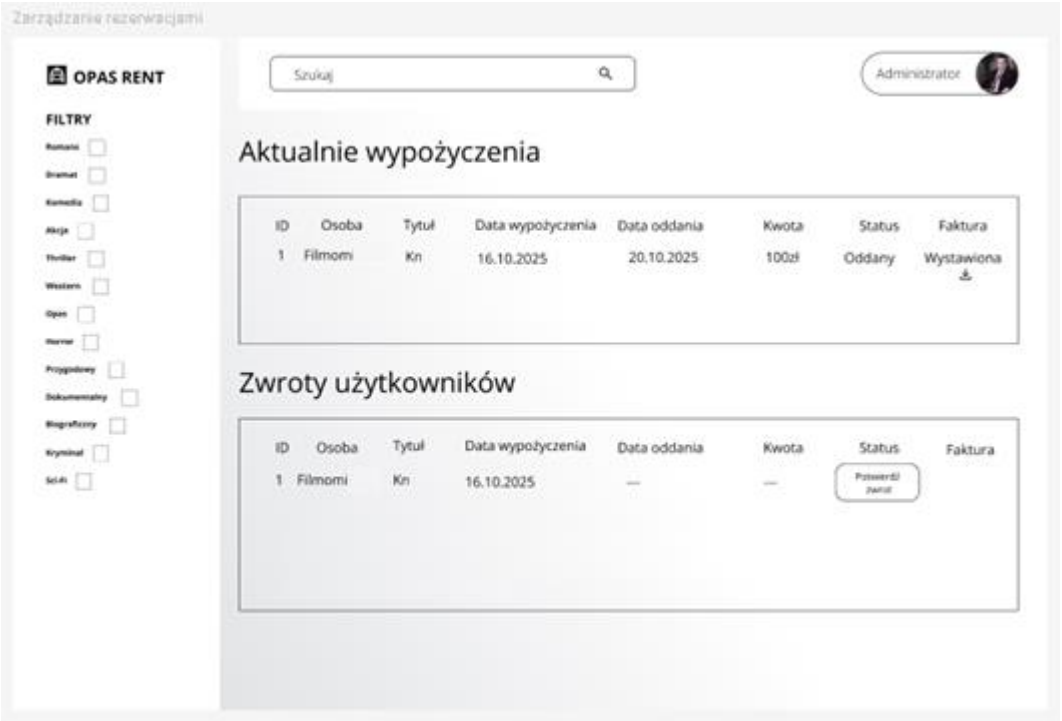
Rysunek 2.6. Makieta ekranów z informacjami o wybranym filmie po stronie klienta



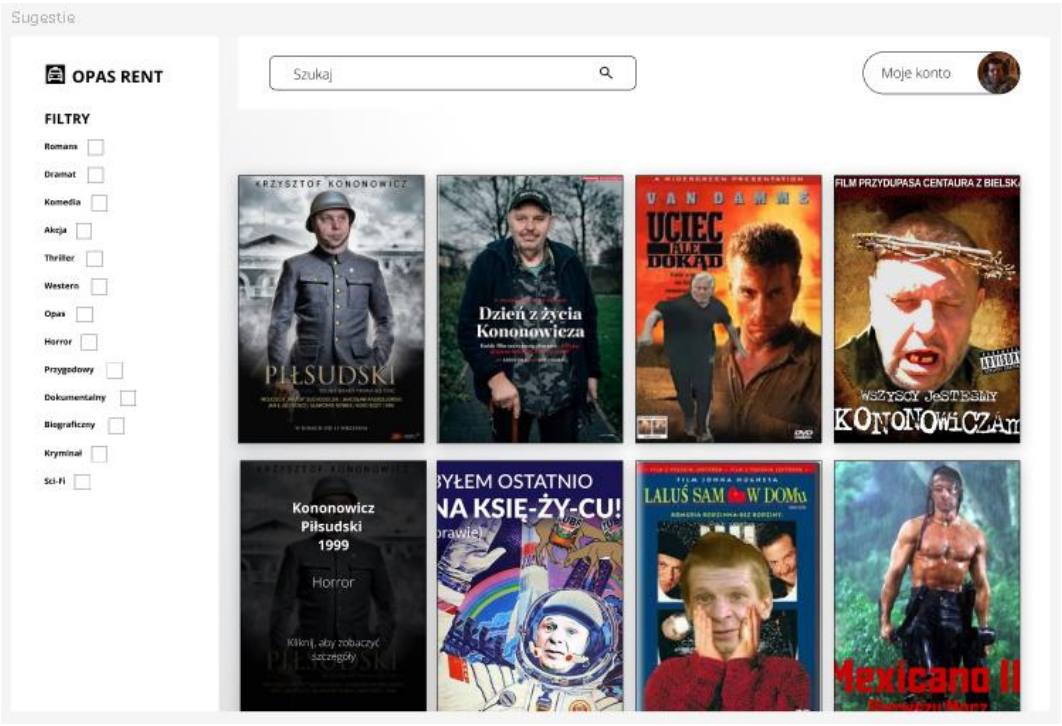
Rysunek 2.7. Makieta ekranów z informacjami o wybranym filmie po stronie administratora



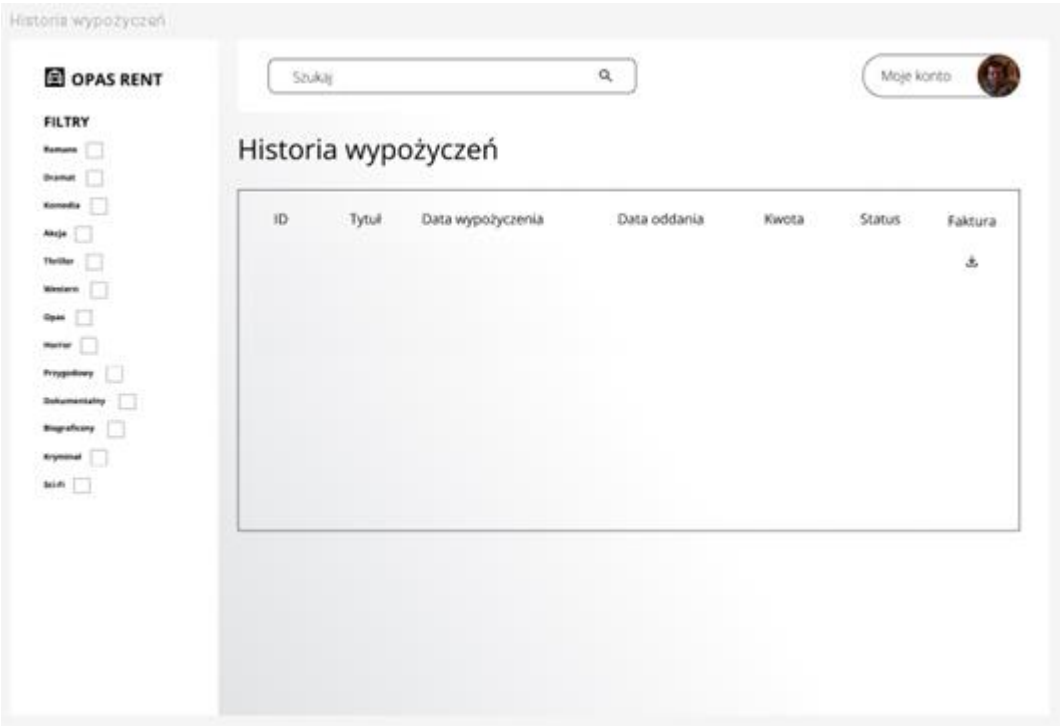
Rysunek 2.8. Makieta ekranu z aktualnymi wypożyczeniami (widoczne u klienta)



Rysunek 2.9. Makieta ekranu z aktualnymi wypożyczeniami klientów (widoczne u administratora)



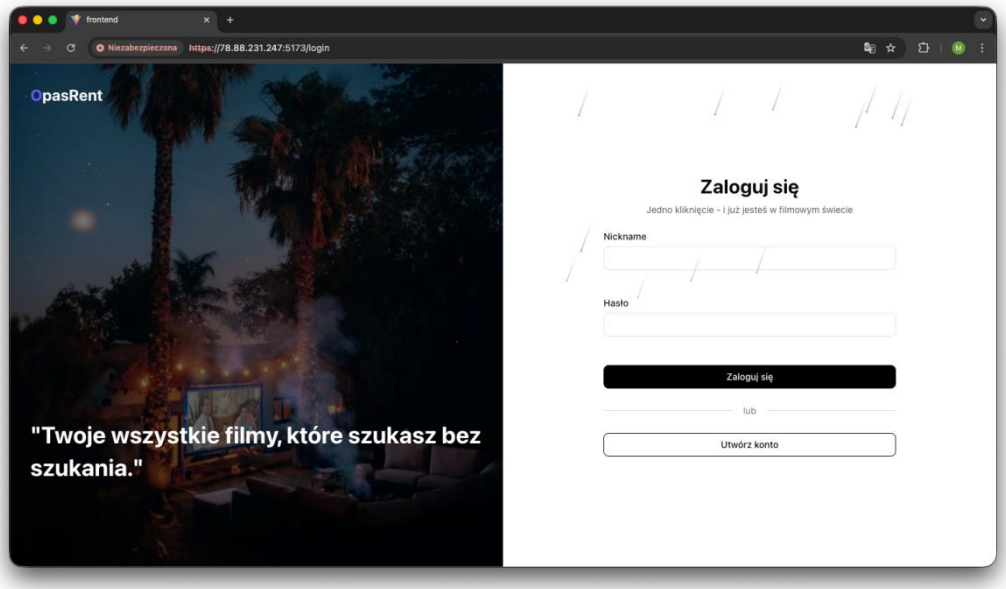
Rysunek 2.10. Makieta ekranu z sugestiami (widoczne u klienta)



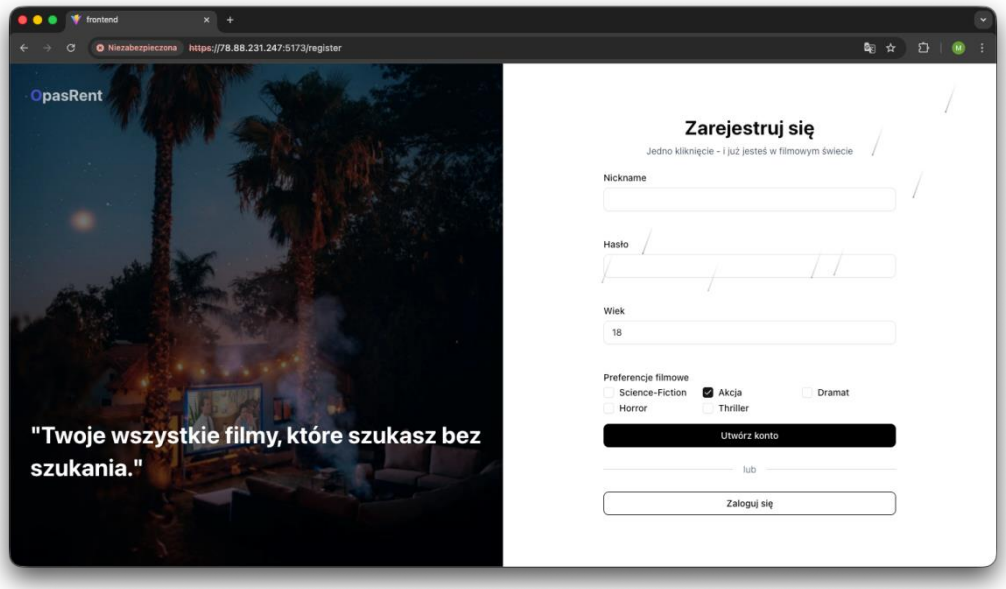
Rysunek 2.11. Makieta ekranu z historią wypożyczeń (widoczne u klienta)

2.2. Widok zaimplementowanych makiet

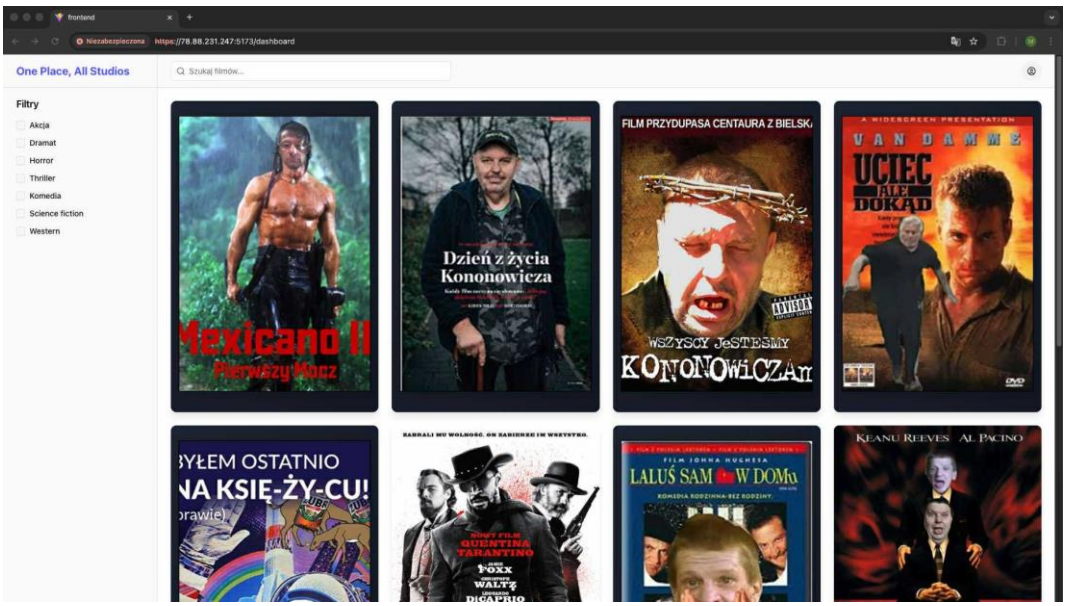
Ekrany gotowej aplikacji zostały wykonane za pomocą frameworka React i zostały przedstawione na zdjęciach poniżej.



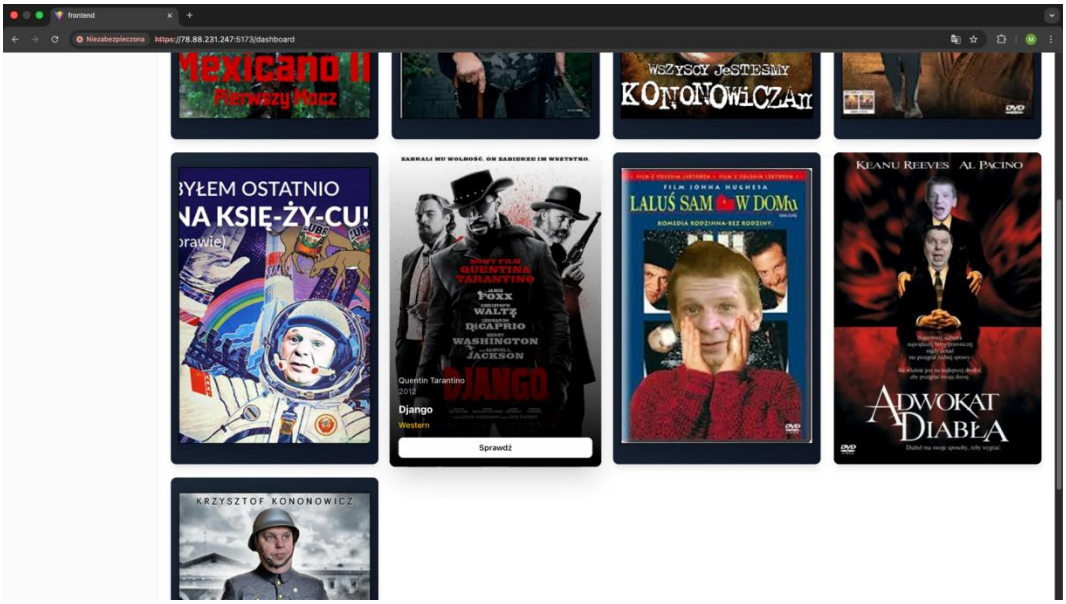
Rysunek 2.12. Ekran logowania użytkownika w aplikacji



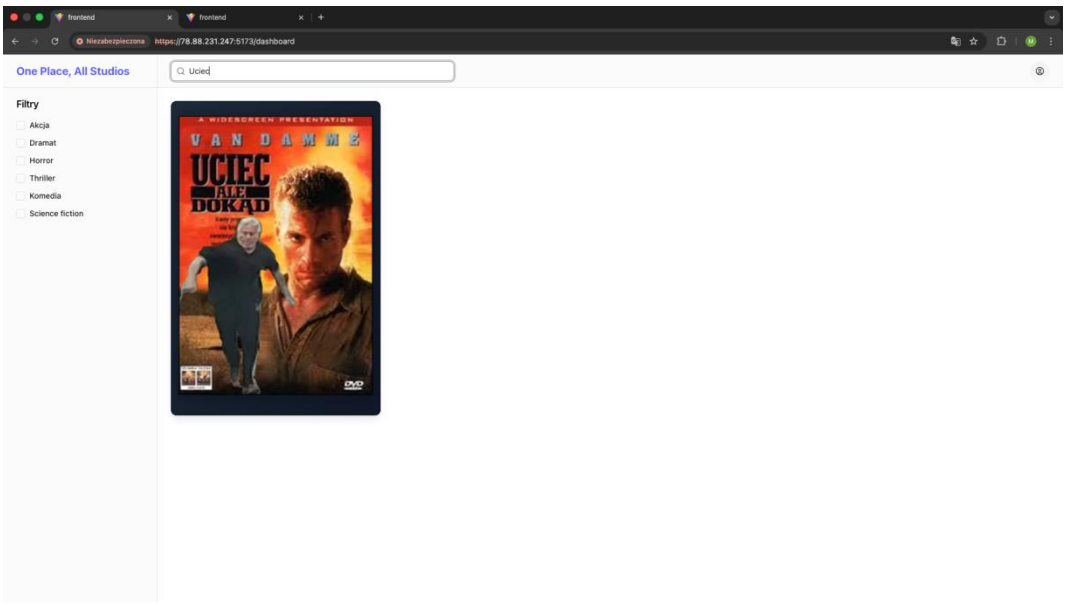
Rysunek 2.13. Ekran rejestracji użytkownika w aplikacji



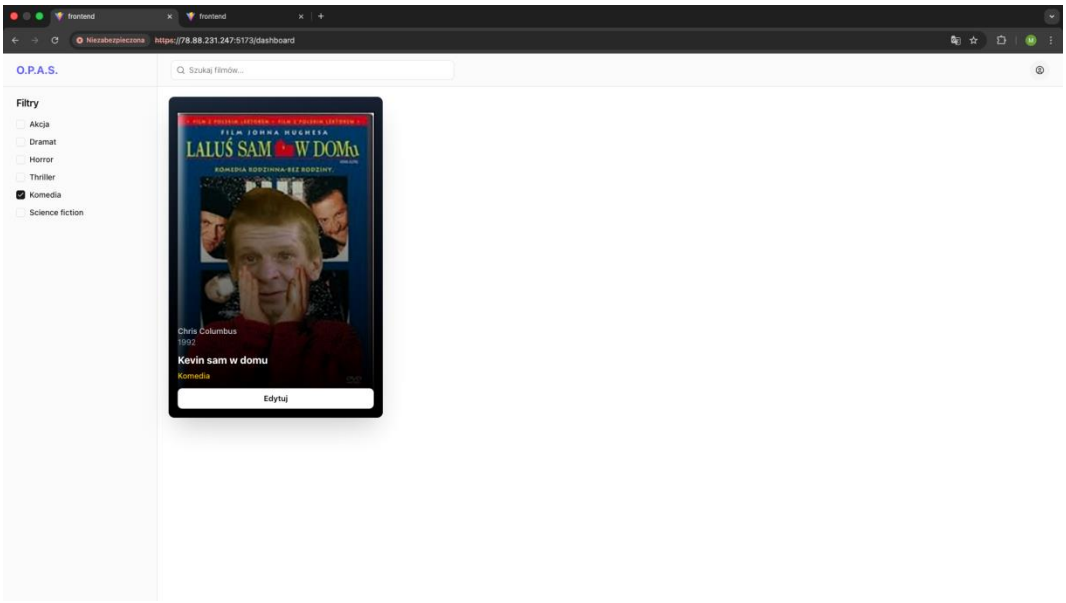
Rysunek 2.14. Ekran panelu głównego aplikacji (dla klienta oraz administratora)



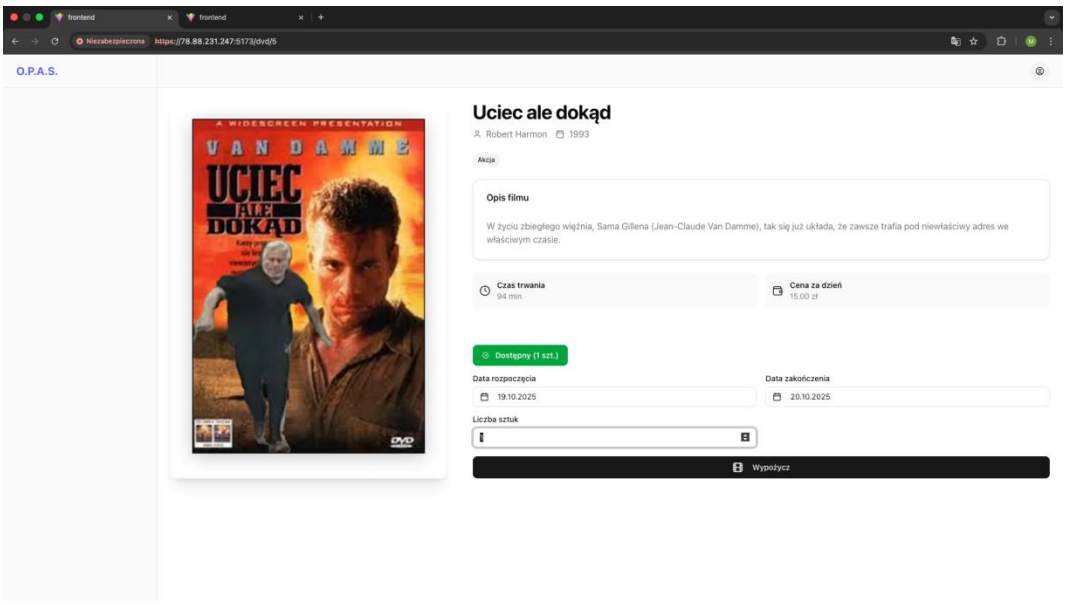
Rysunek 2.15. Ekran panelu głównego po najechaniu na myszką na film (dla klienta, administrator ma opcję „Edytuj” zamiast „Sprawdź”)



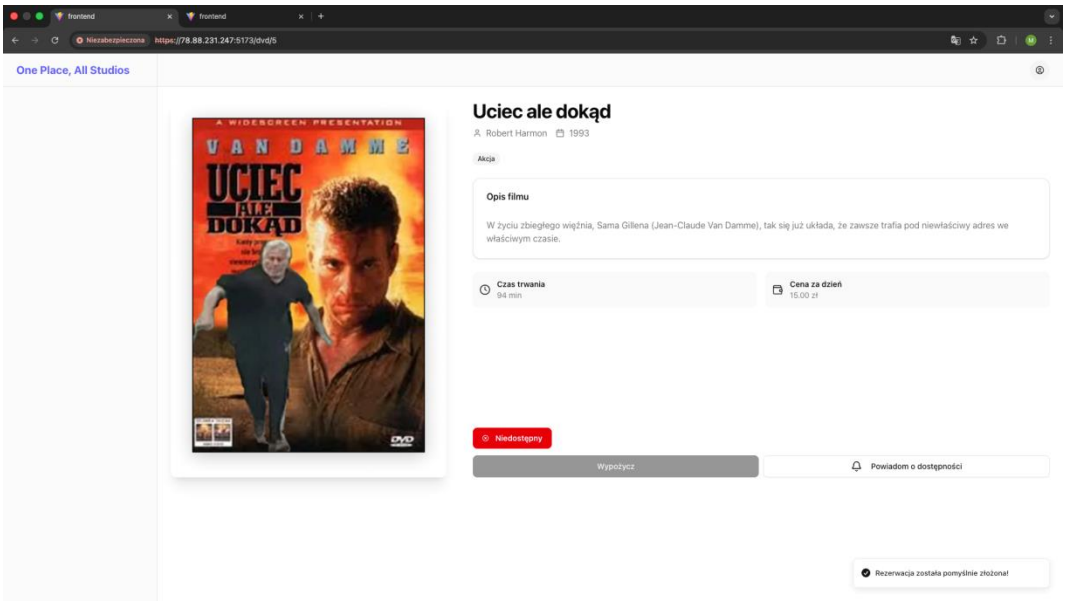
Rysunek 2.16. Ekran przedstawiający mechanizm wyszukiwania filmów (dla klienta oraz administratora)



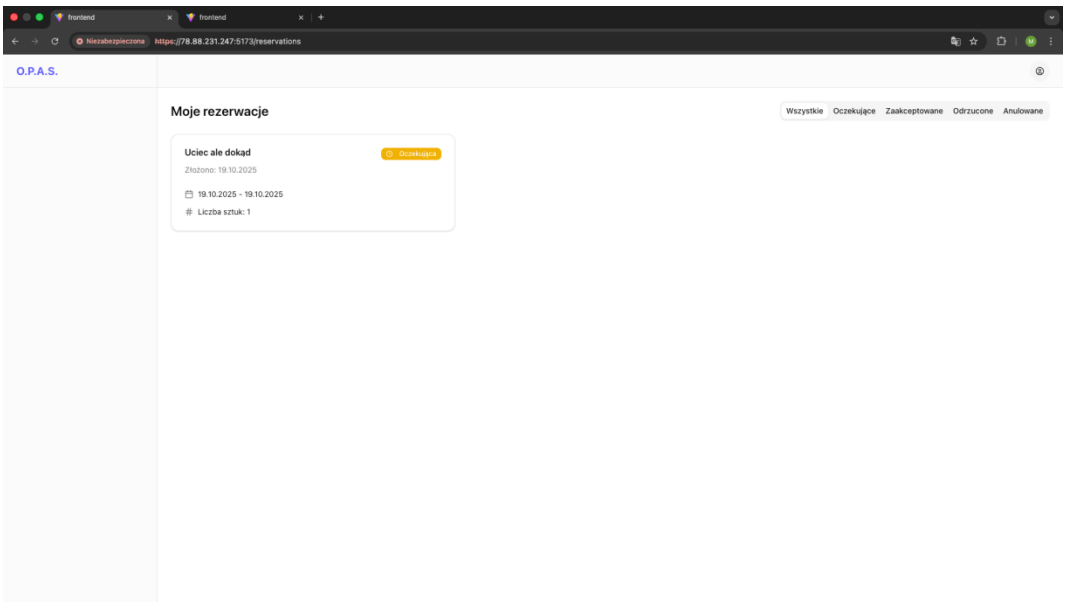
Rysunek 2.17. Ekran przedstawiający mechanizm filtrowania filmów po gatunku (dla klienta oraz administratora)



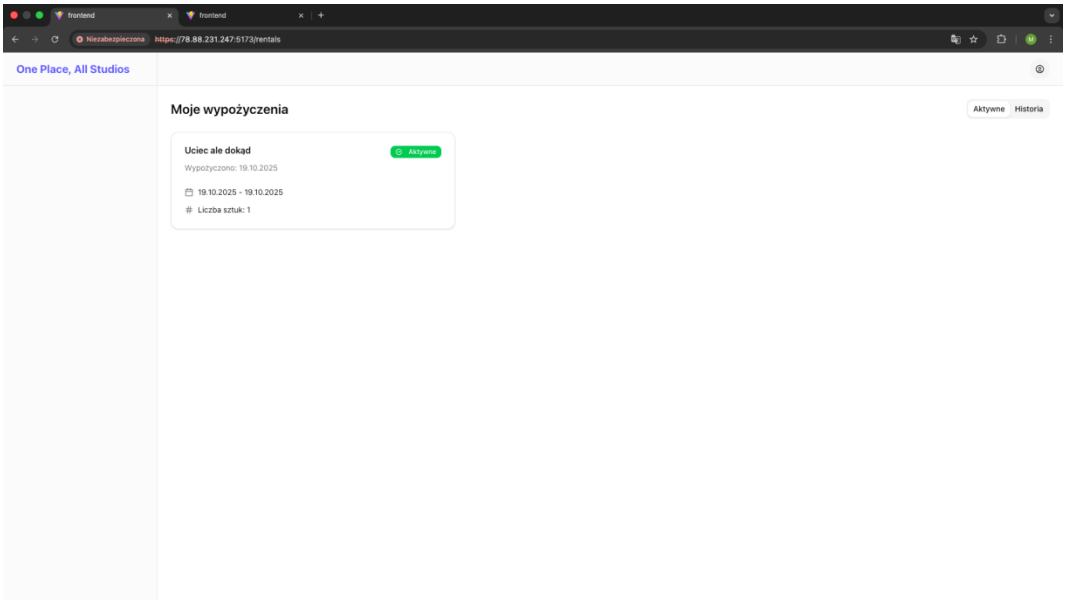
Rysunek 2.18. Ekran z możliwością wypożyczenia wybranego filmu jeśli ten jest dostępny (widoczne u klienta)



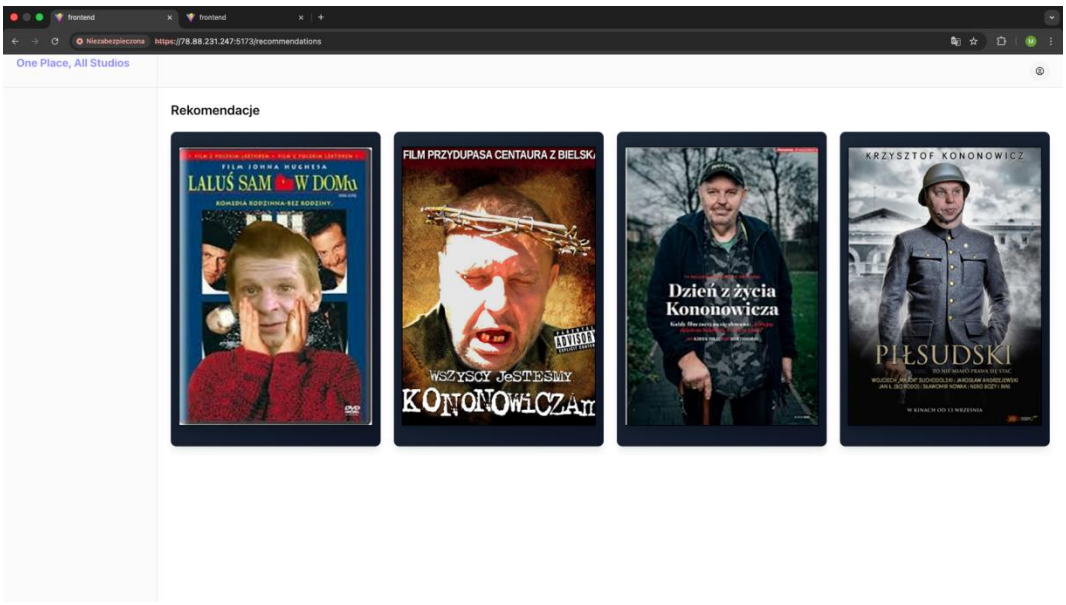
Rysunek 2.19. Ekran z możliwością wypożyczenia wybranego filmu jeśli ten nie jest dostępny (widoczne u klienta)



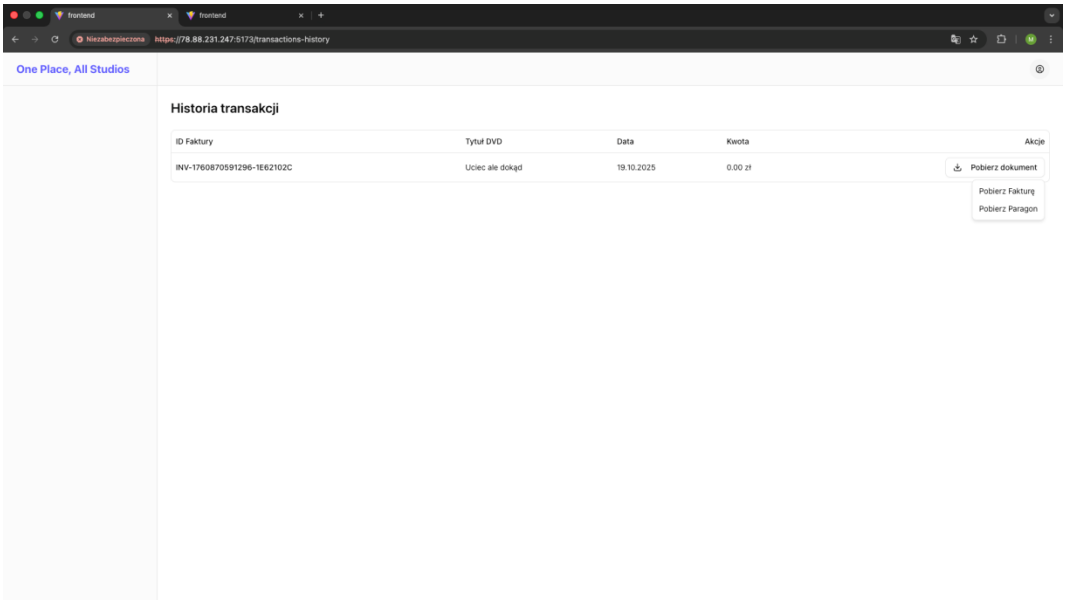
Rysunek 2.20. Ekran z listą rezerwacji (widoczne u klienta)



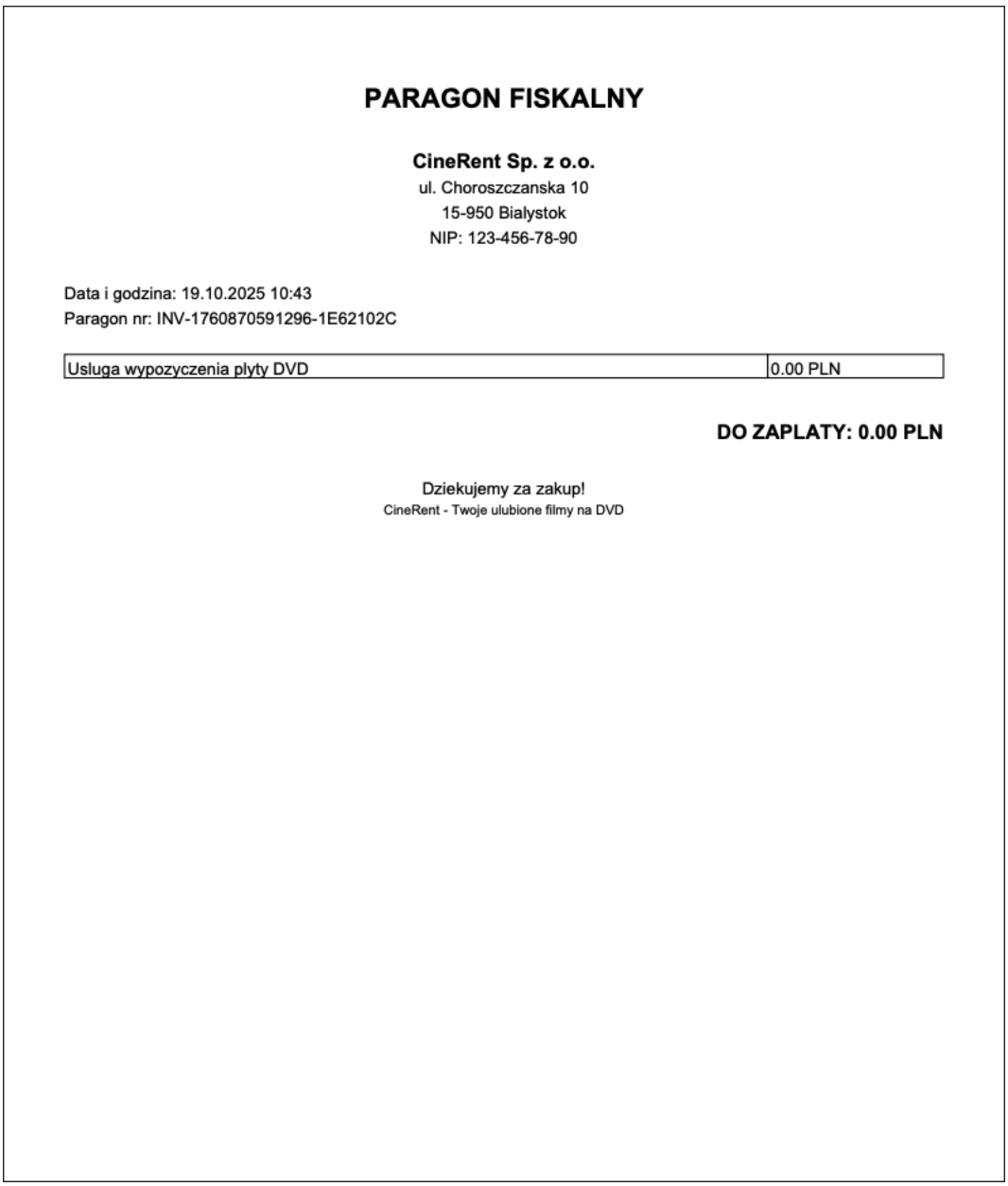
Rysunek 2.21. Ekran z listą wypożyczeń (widoczne u klienta)



Rysunek 2.22. Ekran z rekomendacjami (widocze u klienta)



Rysunek 2.23. Ekran z historią transakcji (widoczne u klienta)



PARAGON FISKALNY

CineRent Sp. z o.o.
ul. Choroszczanska 10
15-950 Bialystok
NIP: 123-456-78-90

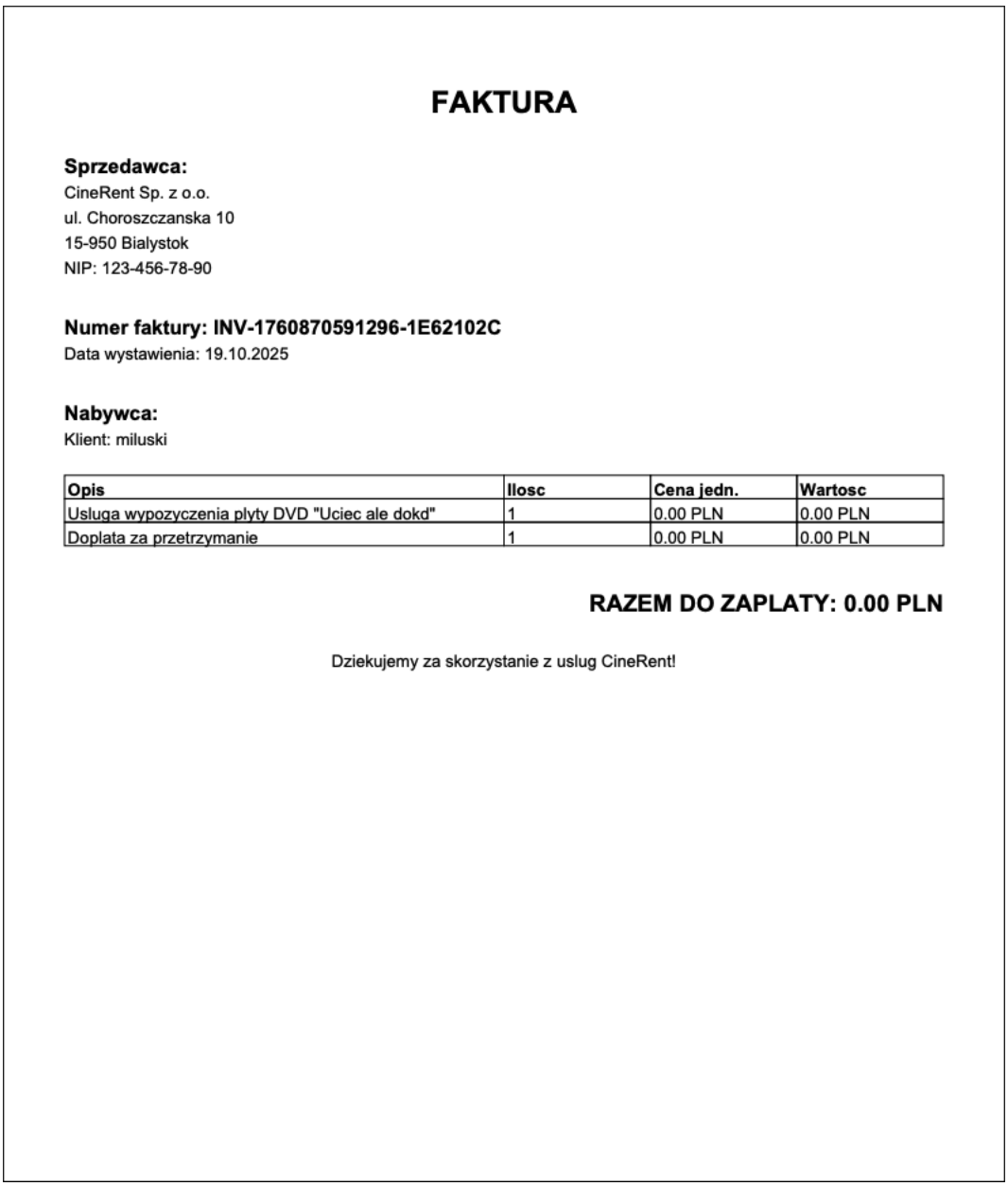
Data i godzina: 19.10.2025 10:43
Paragon nr: INV-1760870591296-1E62102C

Usluga wypozyczenia plyty DVD	0.00 PLN

DO ZAPLATY: 0.00 PLN

Dziekujemy za zakup!
CineRent - Twoje ulubione filmy na DVD

Rysunek 2.24. Widok wygenerowanego paragonu za dokonaną transakcję



FAKTURA

Sprzedawca:
CineRent Sp. z o.o.
ul. Choroszczanska 10
15-950 Bialystok
NIP: 123-456-78-90

Numer faktury: INV-1760870591296-1E62102C
Data wystawienia: 19.10.2025

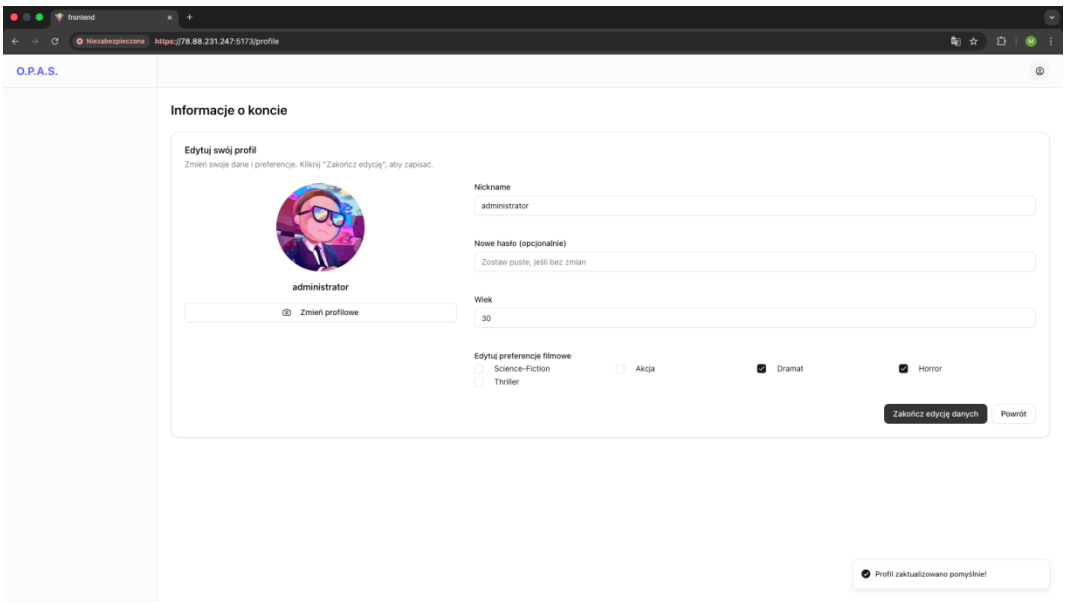
Nabywca:
Klient: miluski

Opis	Ilosc	Cena jedn.	Wartosc
Usluga wypozyczenia plyty DVD "Uciec ale dokd"	1	0.00 PLN	0.00 PLN
Doplata za przetrzymanie	1	0.00 PLN	0.00 PLN

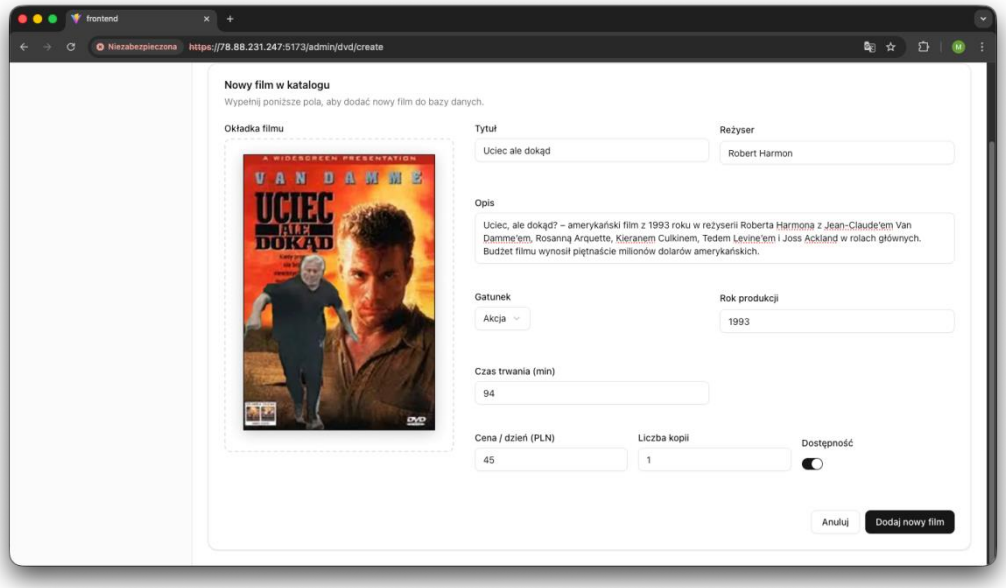
RAZEM DO ZAPLATY: 0.00 PLN

Dziekujemy za skorzystanie z uslug CineRent!

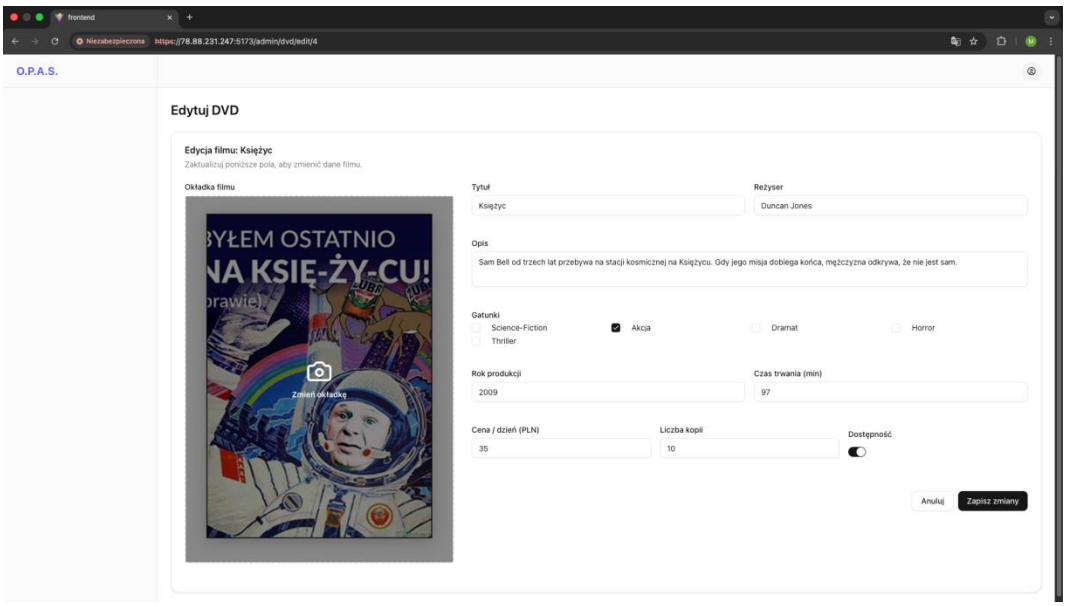
Rysunek 2.25. Widok wygenerowanej faktury za dokonaną transakcję



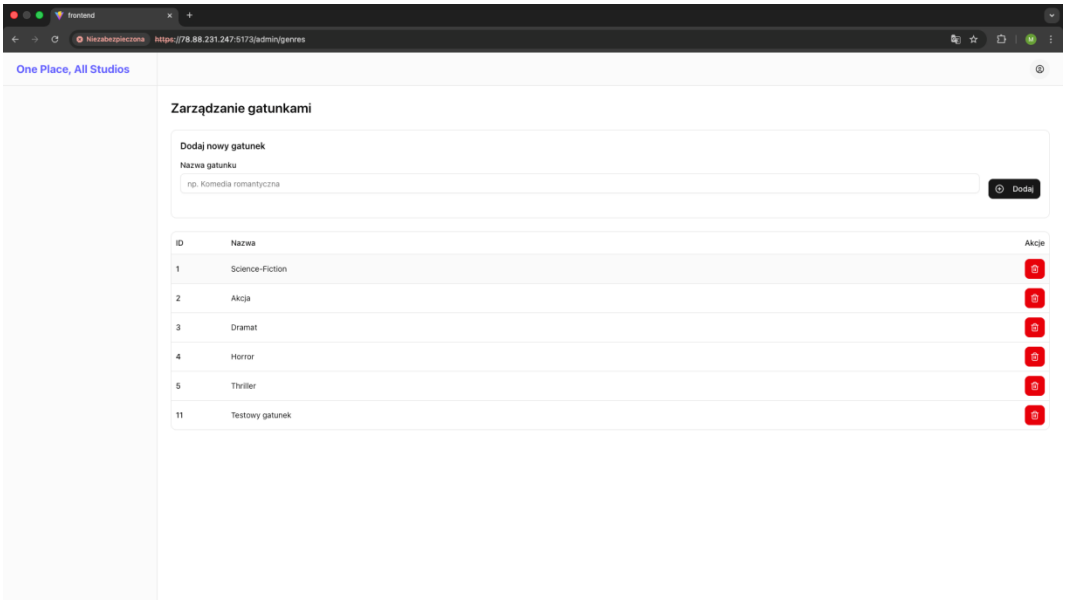
Rysunek 2.26. Ekran przedstawiający informacje o koncie, z możliwością edycji danych (dla klienta oraz administratora)



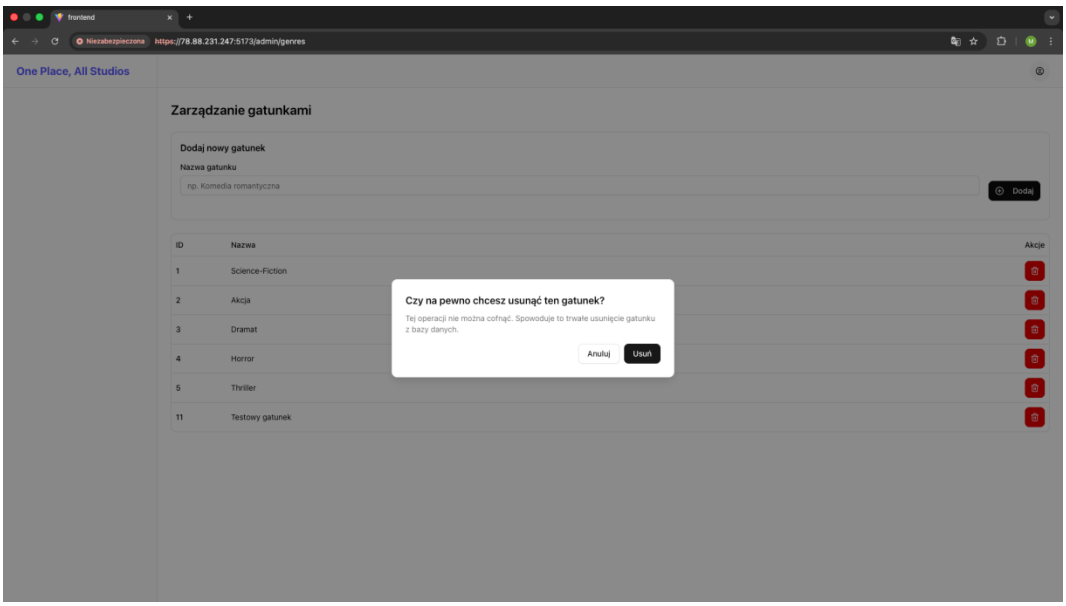
Rysunek 2.27. Ekran z możliwością dodanie nowego filmu (widoczne u administratora)



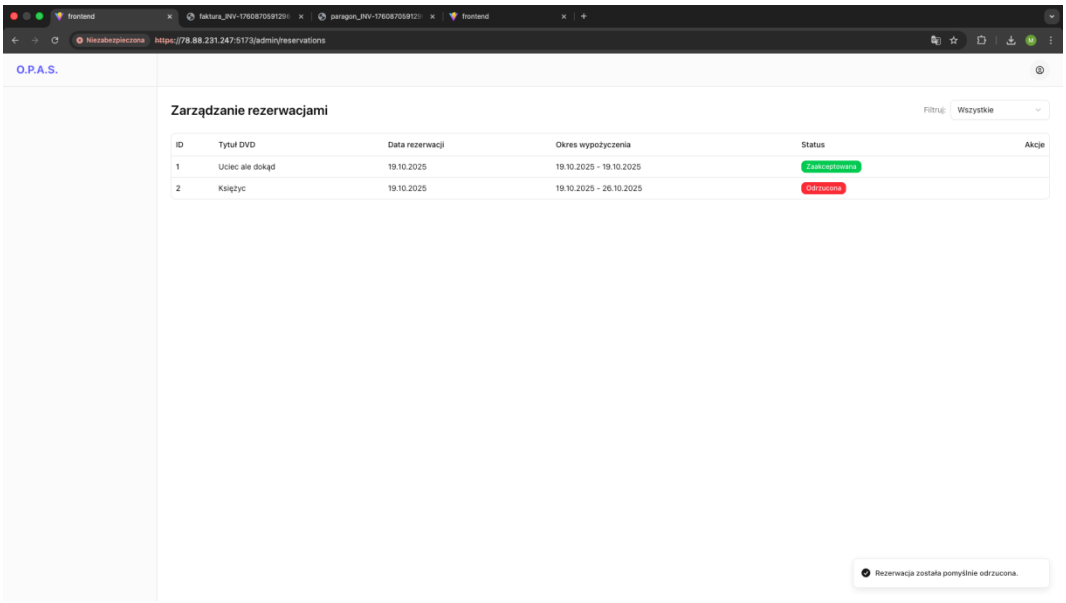
Rysunek 2.28. Ekran z możliwością edycji wybranego filmu (widoczne u administratora)



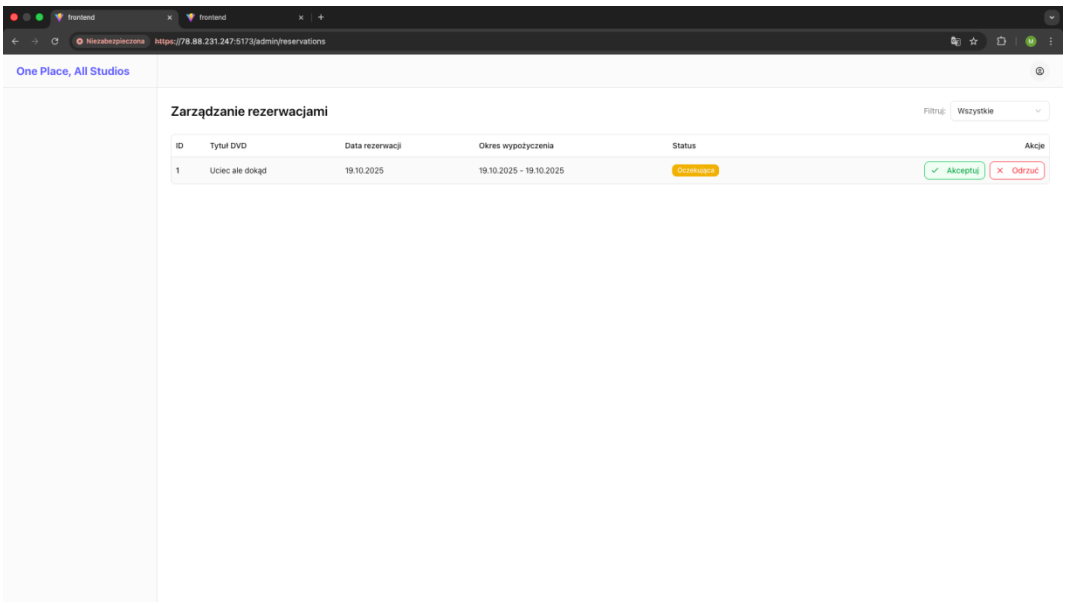
Rysunek 2.29. Ekran zarządzania gatunkami filmów (widoczne u administratora)



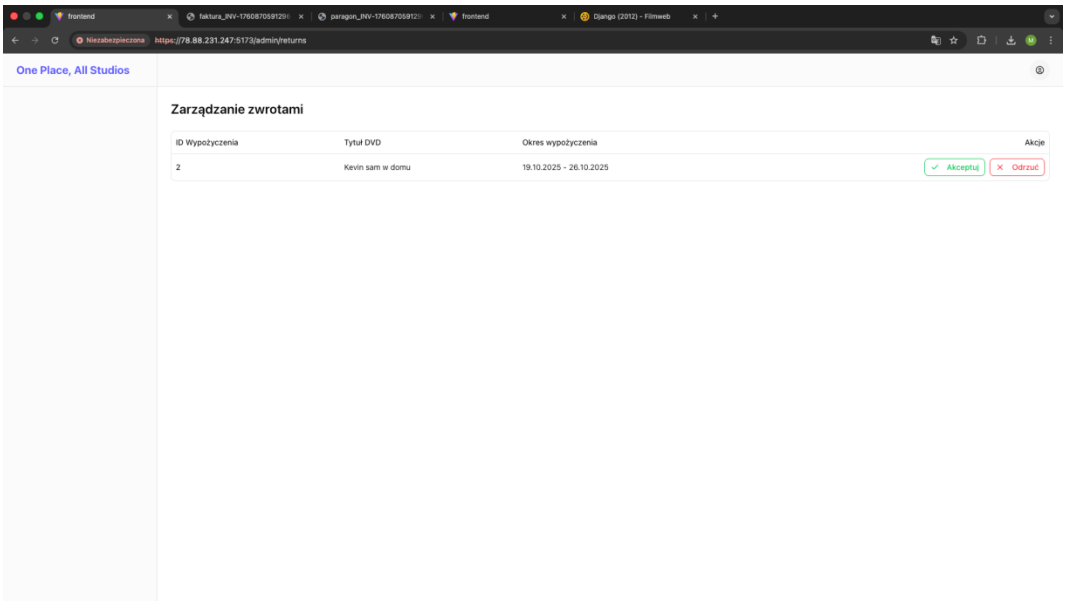
Rysunek 2.30. Ekran przedstawiający okno dialogowe z potwierdzeniem usunięcia gatunku filmowego (widoczne u administratora)



Rysunek 2.31. Ekran zarządzania rezerwacjami (widoczne u administratora)



Rysunek 2.32. Ekran zarządzania rezerwacjami, z możliwością potwierdzenia lub odrzucenia rezerwacji (widoczne u administratora)



Rysunek 2.33. Ekran zarządzania zwrotami, z możliwością potwierdzenia lub odrzucenia zwrotu (widoczne u administratora)

3. Opis użytych technologii oraz frameworków

3.1. Backend

3.1.1. Java

Java to obiektowy język programowania działający na wirtualnej maszynie Java Virtual Machine (JVM). Charakteryzuje się przenośnością kodu, silnym typowaniem i bogatym zestawem bibliotek standardowych. W projekcie wykorzystano Javę w wersji 21, co zapewniło dostęp do nowoczesnych funkcji językowych, takich jak rekordy, pattern matching oraz ulepszona obsługa wielowątkowości.

Język Java został użyty do implementacji warstwy serwerowej aplikacji, obejmującej logikę biznesową, kontrolery REST, serwisy, repozytoria oraz integrację z bazą danych. Dzięki zastosowaniu Javy uzyskano wysoką niezawodność, możliwość testowania jednostkowego oraz łatwą integrację z frameworkiem Spring Boot, który znacząco ułatwia konfigurację i zarządzanie komponentami aplikacji.

Java została również wybrana ze względu na szerokie wsparcie społeczności, stabilność działania w środowiskach produkcyjnych oraz zgodność z narzędziami do budowy (Maven) i testowania aplikacji (JUnit, Mockito).

3.1.2. Spring Boot

Spring Boot to framework oparty na ekosystemie Spring, który umożliwia szybkie tworzenie aplikacji webowych i mikroserwisów. Jego główną zaletą jest mechanizm auto-konfiguracji, dzięki któremu znacznie ograniczono ilość ręcznych ustawień wymaganych do uruchomienia projektu.

W projekcie Spring Boot (wersja 3.5.6) stanowi podstawę warstwy backendowej. Aplikacja została oparta na architekturze REST API, która zapewnia komunikację z frontendem React poprzez żądania HTTP. Framework odpowiada m.in. za:

- integrację z bazą danych PostgreSQL za pośrednictwem Spring Data JPA,
- obsługę bezpieczeństwa z wykorzystaniem JWT (JSON Web Token) oraz ciasteczek HttpOnly,
- konfigurację HTTPS z certyfikatami SSL,
- generowanie dokumentacji API przy użyciu Swagger/OpenAPI,
- logowanie zdarzeń aplikacji i zarządzanie wyjątkami,
- automatyczne testowanie z użyciem bazy H2.

Spring Boot został uruchomiony w kontenerze Docker na porcie 443, z pełną obsługą HTTPS. Dzięki jego modularności możliwe było wprowadzenie jasnego podziału aplikacji na warstwy (kontrolery, serwisy, repozytoria), co zwiększyło czytelność i utrzymywalność kodu.

3.2. Frontend

3.2.1. TypeScript

TypeScript to nadzbiór języka JavaScript opracowany przez Microsoft, który dodaje do niego statyczne typowanie oraz mechanizmy znane z języków obiektowych. Pozwala to na wcześniejsze wykrywanie błędów, lepsze autouzupełnianie kodu i większą stabilność projektu.

W projekcie TypeScript został zastosowany w warstwie frontendowej do implementacji logiki interfejsu użytkownika oraz komunikacji z backendem. Użycie typów dla komponentów i danych przesyłanych przez REST API zredukowało ryzyko błędów wynikających z niezgodności typów oraz zwiększyło czytelność kodu.

TypeScript współpracuje w projekcie z Vite, nowoczesnym narzędziem do bundlowania i uruchamiania aplikacji React z funkcją Hot Module Replacement (HMR). Dzięki temu środowisko deweloperskie jest wydajne i pozwala na natychmiastowe podglądy wprowadzanych zmian w kodzie.

3.2.2. React

React to biblioteka frontendowa opracowana przez Meta (Facebook), służąca do budowy interfejsów użytkownika w oparciu o komponenty i wirtualny model DOM. W projekcie OpasRent użyto wersji React 18, z wykorzystaniem nowoczesnych funkcji, takich jak hooks (useState, useEffect, useContext) oraz context API do zarządzania stanem aplikacji.

React odpowiada za dynamiczne renderowanie widoków, obsługę formularzy logowania i rejestracji, prezentację katalogu płyt DVD oraz interakcje użytkownika z systemem wypożyczeń. Komunikacja z backendem Spring Boot odbywa się poprzez zapytania HTTP, obsługiwane za pomocą biblioteki axios.

Wersja deweloperska Reacta uruchamiana jest na porcie 5173 (serwer Vite), natomiast wersja produkcyjna jest budowana i serwowana przez Nginx. Zastosowanie Reacta pozwoliło uzyskać nowoczesny, responsywny interfejs, który reaguje płynnie na działania użytkownika i jest łatwy w dalszym rozwoju.

3.3. Baza Danych

3.3.1. PostgreSQL

React to biblioteka frontendowa opracowana przez Meta (Facebook), służąca do budowy interfejsów użytkownika w oparciu o komponenty i wirtualny model DOM. W projekcie użyto wersji React 18, z wykorzystaniem nowoczesnych funkcji, takich jak hooks (useState, useEffect, useContext) oraz context API do zarządzania stanem aplikacji.

React odpowiada za dynamiczne renderowanie widoków, obsługę formularzy logowania i rejestracji, prezentację katalogu płyt DVD oraz interakcje użytkownika z systemem wypożyczeń. Komunikacja z backendem Spring Boot odbywa się poprzez zapytania HTTP, obsługiwane za pomocą biblioteki axios.

Wersja deweloperska Reacta uruchamiana jest na porcie 5173 (serwer Vite), natomiast wersja produkcyjna jest budowana i serwowana przez Nginx. Zastosowanie Reacta pozwoliło uzyskać nowoczesny, responsywny interfejs, który reaguje płynnie na działania użytkownika i jest łatwy w dalszym rozwoju.

3.4. Koncept UX/UI

3.4.1. Figma

Figma to webowe narzędzie do projektowania interfejsów użytkownika (UI) oraz tworzenia prototypów interakcji (UX). Umożliwia projektowanie w czasie rzeczywistym z poziomu przeglądarki, co wspiera współpracę zespołową i szybką iterację projektu graficznego.

W projekcie Figma została wykorzystana do zaprojektowania koncepcji interfejsu użytkownika aplikacji webowej. Opracowane makiety obejmowały ekrany logowania, rejestracji, przeglądania katalogu płyt, historii wypożyczeń oraz szczegółów transakcji, z podziałem na administratora i użytkownika.

Projekt w Figmie określał również paletę kolorów, typografię, rozmieszczenie elementów na stronie oraz zasady nawigacji. Dzięki temu frontend React mógł zostać zbudowany w sposób spójny wizualnie, zgodny z zasadami użyteczności (UX) i nowoczesnym stylem aplikacji webowych.

Wykorzystanie Figmy przyczyniło się do poprawy czytelności interfejsu, skrócenia czasu implementacji widoków oraz ułatwiło komunikację pomiędzy projektantem a zespołem deweloperskim.

4. Architektura systemu

4.1. Opis architektury systemu

Architektura systemu OpasRent została zaprojektowana w oparciu o podejście wielowarstwowe oraz wzorzec Model-View-Controller (MVC), który zapewnia czytelne rozdzielenie logiki biznesowej, warstwy prezentacji i dostępu do danych.

System składa się z trzech głównych komponentów:

- Aplikacji klienta (frontend) – zrealizowanej w technologii React,
- Aplikacji serwera (backend) – zbudowanej w oparciu o Spring Boot i wzorzec MVC,
- Bazy danych – opartej o PostgreSQL.

Całość rozwiązania została przygotowana jako aplikacja konteneryzowana przy użyciu Docker Compose, co umożliwia łatwe uruchamianie i skalowanie środowiska w różnych konfiguracjach (deweloperskiej, testowej i produkcyjnej).

4.2. Ogólna koncepcja architektury

Architektura systemu zakłada rozdzielenie aplikacji na niezależne, lecz współpracujące ze sobą moduły komunikujące się za pomocą interfejsu REST API.

Frontend (React) odpowiada za interakcję z użytkownikiem, wizualizację danych oraz obsługę zdarzeń, natomiast backend (Spring Boot) przetwarza logikę biznesową, zarządza bezpieczeństwem, sesjami oraz komunikacją z bazą danych PostgreSQL.

W ramach backendu zaimplementowano architekturę opartą o wzorzec Model–View–Controller (MVC):

- Model reprezentuje warstwę danych i odwzorowuje strukturę tabel w bazie danych.
- View stanowi warstwę pośrednią odpowiadającą za przygotowanie danych do prezentacji w formacie JSON.
- Controller odpowiada za obsługę żądań HTTP, delegowanie zadań do warstwy serwisu oraz zwracanie odpowiednich odpowiedzi.

4.3. Aplikacja klienta (frontend)

Warstwa frontendowa została zrealizowana przy użyciu frameworka React 18 oraz narzędzia Vite, które zapewnia szybkie budowanie i natychmiastowy podgląd zmian (hot reload).

Interfejs użytkownika umożliwia pełną obsługę cyklu życia wypożyczeń: przeglądanie katalogu płyt DVD, rejestrację i logowanie użytkowników, dokonywanie rezerwacji, wypożyczeń, zwrotów oraz pobieranie faktur w formacie PDF.

Dodatkowo interfejs został zabezpieczony poprzez mechanizm uwierzytelniania oparty JWT (JSON Web Token), który umożliwia bezpieczne zarządzanie sesją użytkownika bez konieczności przechowywania danych logowania po stronie serwera. Po poprawnym zalogowaniu użytkownika backend generuje token JWT zawierający zaszyfrowane informacje o jego tożsamości oraz czasie ważności sesji. Token ten jest następnie przekazywany do przeglądarki w formie HttpOnly cookie, co uniemożliwia jego odczyt lub modyfikację przez skrypty JavaScript działające w kontekście strony. Takie rozwiązanie znacząco zwiększa poziom bezpieczeństwa aplikacji, gdyż minimalizuje ryzyko ataków typu XSS (Cross-Site Scripting) oraz Session Hijacking. Dodatkowo ciasteczka te są oznaczone flagą Secure, dzięki czemu mogą być przesyłane wyłącznie przez szyfrowane połączenie HTTPS. Mechanizm ten zapewnia nie tylko poufność przesyłanych danych, ale również integralność i autentyczność żądań wysyłanych do serwera, co jest kluczowe w kontekście ochrony danych osobowych i transakcyjnych w systemach zarządzania wypożyczeniami.

Frontend komunikuje się z backendem wyłącznie przez zaszyfrowane połączenie HTTPS (port 4443), korzystając z automatycznie generowanego certyfikatu SSL.

4.4. Aplikacja serwera (backend)

Aplikacja serwerowa została napisana w języku Java 21 z wykorzystaniem Spring Boot 3.5.6. Jej architektura odzwierciedla wzorzec MVC, w którym:

- kontrolery (@RestController) udostępniają zasoby REST API,
- serwisy (@Service) realizują logikę biznesową,
- repozytoria (@Repository) odpowiadają za komunikację z bazą danych przy użyciu Spring Data JPA.

Backend zapewnia obsługę wszystkich procesów związanych z zarządzaniem wypożyczeniami DVD, w tym:

- rejestrację i uwierzytelnianie użytkowników,
- zarządzanie katalogiem płyt DVD,
- przetwarzanie transakcji wypożyczeń i zwrotów,
- generowanie faktur i potwierdzeń w formacie PDF (biblioteka iText5).

Zastosowano kompleksowy mechanizm bezpieczeństwa oparty o:

- JWT Authentication z 512-bajtowym kluczem i bezpiecznymi ciasteczkami,
- SSL/TLS do szyfrowania połączeń,
- uruchamianie aplikacji w kontenerze jako nieuprzywilejowany użytkownik.

Dla celów testowych backend wykorzystuje wbudowaną bazę H2, co pozwala na izolowane testowanie logiki aplikacji bez potrzeby uruchamiania PostgreSQL.

4.5. Warstwa danych

Warstwa danych systemu oparta jest o relacyjną bazę danych PostgreSQL, działającą w osobnym kontenerze Docker.

Schemat bazy został zaprojektowany w sposób umożliwiający odwzorowanie rzeczywistych relacji pomiędzy encjami, takimi jak:

- Użytkownicy (Users),
- Płyty DVD (DVDs),
- Wypożyczenia (Rentals),
- Rezerwacje (Reservations),
- Faktury (Invoices).

Połączenie z bazą danych realizowane jest poprzez sterownik JDBC oraz warstwę Spring Data JPA, co zapewnia wysoką wydajność i elastyczność przy pracy z danymi.

Dzięki zastosowaniu connection poolingu możliwe jest efektywne zarządzanie wieloma jednoczesnymi połączeniami, co ma szczególne znaczenie w środowiskach produkcyjnych.

4.6. Konteneryzacja

System został w pełni skonteneryzowany z użyciem Docker Compose, co pozwala na uruchomienie wszystkich komponentów (frontend, backend, baza danych) w jednym poleceniu.

Poszczególne usługi komunikują się w ramach prywatnej sieci Docker:

- Backend (Spring Boot) – port 4443 (HTTPS),
- Frontend (React + Vite) – port 5173 (HTTP),
- PostgreSQL – port 5432.

Dzięki zastosowaniu kontenerów, system jest w pełni przenośny i może być wdrażany w środowiskach lokalnych, testowych i produkcyjnych bez konieczności zmian w konfiguracji. Całość jest zautomatyzowana poprzez skrypt init.sh, który generuje wszystkie niezbędne certyfikaty, hasła i pliki konfiguracyjne.

4.7. Dokumentacja i loggowanie

Aplikacja serwera udostępnia interaktywną dokumentację API opartą o Swagger / OpenAPI, dostępną pod adresem głównym backendu. Pozwala ona na przeglądanie i testowanie wszystkich dostępnych endpointów oraz sprawdzanie struktur żądań i odpowiedzi.

Dodatkowo system zapisuje szczegółowe logi działania w katalogu logs/, umożliwiając analizę wydajności i błędów. W środowisku testowym logi są oddzielane od produkcyjnych, co ułatwia utrzymanie i diagnostykę systemu.

5. Funkcjonalności systemu

System OpasRent został zaprojektowany tak, aby kompleksowo obsługiwać procesy związane z wypożyczaniem płyt DVD – od rejestracji użytkownika, poprzez przegląd katalogu, aż po generowanie rachunków i zarządzanie zwrotami.

Aplikacja rozróżnia dwa poziomy dostępu: użytkownika standardowego (klienta) oraz administratora systemu. Każdy z nich ma przypisane odpowiednie zestawy funkcjonalności.

5.1. Funkcjonalności użytkownika

5.1.1. Rejestracja konta

Każdy użytkownik systemu może utworzyć własne konto poprzez formularz rejestracyjny. Podczas rejestracji wymagane jest podanie unikalnej nazwy użytkownika (nickname), hasła, wieku oraz listy preferowanych gatunków filmowych. System weryfikuje poprawność danych – sprawdza m.in. minimalny wiek wymagany do korzystania z wypożyczalni, a także unikalność podanej nazwy użytkownika w bazie danych. Hasło przed zapisaniem jest szyfrowane z użyciem algorytmu argon2, co zapewnia bezpieczeństwo danych użytkownika.

Proces rejestracji odbywa się poprzez endpoint POST */api/v1/auth/register*.

5.1.2. Logowanie do systemu

Użytkownik loguje się do systemu, podając swoją nazwę i hasło. Po pomyślnym uwierzytelnieniu system generuje parę tokenów JWT: token dostępowy (ważny przez 15 minut) oraz token odświeżania (ważny przez 7 dni). Tokeny te są przechowywane w bezpiecznych ciasteczkach typu HttpOnly, co uniemożliwia ich odczytanie po stronie przeglądarki. Podczas logowania backend rozpoznaje przypisaną rolę użytkownika (np. USER lub ADMIN), co determinuje zakres jego uprawnień.

Żądanie logowania obsługiwane jest przez endpoint POST */api/v1/auth/login*.

5.1.3. Odświeżanie tokenu / Wylogowanie

System umożliwia automatyczne odświeżenie tokenu dostępowego po jego wygaśnięciu oraz bezpieczne wylogowanie z systemu. Podczas wylogowania token odświeżania zostaje unieważniony, a ciasteczko usunięte z przeglądarki, co zapobiega ponownemu wykorzystaniu sesji.

Za te operacje odpowiadają endpointy POST */api/v1/auth/refresh* i POST */api/v1/auth/logout*.

5.1.4. Przegląd kolekcji płyt DVD

Zalogowany użytkownik ma dostęp do pełnej kolekcji płyt DVD znajdujących się w ofercie wypożyczalni. Dane prezentowane są w formie listy zawierającej podstawowe informacje o filmach, takie jak tytuł, gatunek, opis czy dostępność. System umożliwia wyszukiwanie i filtrowanie pozycji według tytułu, opisu lub gatunku filmowego, co ułatwia znalezienie interesującej pozycji.

Funkcjonalność realizowana jest poprzez endpoint GET */api/v1/dvd*.

5.1.5. Rezerwacja płyty DVD

Po zalogowaniu użytkownik może złożyć żądanie rezerwacji wybranego filmu, określając zakres dat oraz liczbę kopii, które chce wypożyczyć. System automatycznie weryfikuje dostępność kopii w wybranym okresie, wykrywa ewentualne konflikty rezerwacji oraz zapisuje zgłoszenie w stanie PENDING (oczekujące). Rezerwacja trafia następnie do panelu administratora, który decyduje o jej akceptacji lub odrzuceniu.

Operacja ta realizowana jest przez endpoint POST */api/v1/reservations/new*.

5.1.6. Przegląd rezerwacji i wypożyczeń

Każdy użytkownik ma dostęp do listy swoich aktywnych rezerwacji oraz bieżących wypożyczeń. Dane mogą być filtrowane według statusu (np. aktywne, oczekujące, zakończone), co pozwala łatwo kontrolować stan wypożyczeń.

Funkcjonalność dostępna jest pod endpointami GET */api/v1/reservations* oraz GET */api/v1/rentals*.

5.1.7. Anulowanie rezerwacji

Jeżeli rezerwacja znajduje się w statusie PENDING, użytkownik ma możliwość jej anulowania. W momencie anulowania system automatycznie zwalnia zarezerwowane kopie, dzięki czemu inne osoby mogą z nich skorzystać.

Operacja odbywa się poprzez endpoint POST */api/v1/reservations/{id}/cancel*.

5.1.8. Zgłoszenie zwrotu wypożyczonej płyty

Po zakończeniu korzystania z wypożyczonego filmu użytkownik może zgłosić chęć jego zwrotu. System zmienia wówczas status wypożyczenia na RETURN_REQUESTED, informując administratora o konieczności przyjęcia zwrotu i ewentualnym naliczeniu opłat.

Zgłoszenie realizowane jest przez endpoint POST */api/v1/rentals/{id}/return-request*.

5.1.9. Historia wypożyczeń

Użytkownik ma możliwość przeglądania historii swoich wcześniejszych wypożyczeń, zawierającej informacje o tytułach, datach wypożyczenia i zwrotu, kosztach. Widok historii umożliwia filtrowanie wyników, np. wyświetlenie tylko zakończonych wypożyczeń.

Dane pobierane są poprzez endpoint GET */api/v1/rentals?filter=HISTORICAL*.

5.1.10. Historia rachunków i dokumentów PDF

System umożliwia użytkownikowi przeglądanie historii wszystkich wygenerowanych rachunków i faktur w formacie PDF. Każdy dokument można pobrać bezpośrednio z interfejsu aplikacji. Dzięki temu użytkownik ma pełny wgląd w historię swoich płatności i transakcji.

Funkcjonalność dostępna jest pod adresem GET */api/v1/transactions*.

5.1.11. Rekomendacje filmowe

OpasRent oferuje inteligentny system rekomendacji, który analizuje historię wypożyczeń użytkownika, jego preferencje gatunkowe, wiek oraz popularność tytułów wśród innych klientów. Na tej podstawie generowana jest lista sugerowanych filmów wraz z uzasadnieniem, dlaczego dana pozycja została polecona.

Rekomendacje pobierane są z endpointu GET */api/v1/user/recommendations*.

5.1.12. Profil użytkownika

Użytkownik ma dostęp do swojego profilu, gdzie może edytować dane takie jak pseudonim, wiek, preferowane gatunki filmowe czy hasło. System wymaga, aby w żądaniu edycji podano co najmniej jedno pole do zmiany, a wszelkie dane są walidowane.

Profil użytkownika dostępny jest pod endpointami GET */api/v1/user* oraz PATCH */api/v1/user/edit*.

5.2. Funkcjonalności administratora

5.2.1. Logowanie do panelu administracyjnego

Administrator systemu korzysta z tego samego panelu logowania co użytkownicy, jednak po uwierzytelnieniu otrzymuje dodatkowe uprawnienia zapisane w tokenie JWT.

5.2.2. Zarządzanie rezerwacjami

Administrator ma dostęp do pełnej listy wszystkich rezerwacji złożonych w systemie. Może je przeglądać, filtrować według statusu oraz podejmować decyzję o ich akceptacji lub odrzuceniu. W przypadku akceptacji system automatycznie tworzy nowe wypożyczenie, zmniejsza liczbę dostępnych kopii filmu oraz powiadamia użytkownika o wyniku operacji.

Operacje te realizowane są przez endpointy GET */api/v1/reservations/all*, POST */api/v1/reservations/{id}/accept* i POST */api/v1/reservations/{id}/decline*.

5.2.3. Zarządzanie zwrotami

Administrator ma możliwość przeglądania wszystkich zgłoszeń zwrotów przesłanych przez użytkowników. Po ich akceptacji system oblicza ewentualne opłaty za spóźnienie, generuje rachunek PDF i zwiększa liczbę dostępnych kopii danego filmu. Zwrot może również zostać odrzucony – na przykład w przypadku błędnego zgłoszenia.

Funkcjonalność ta jest obsługiwana przez endpointy GET */api/v1/rentals/return-requests*, POST */api/v1/rentals/{id}/return-accept* i POST */api/v1/rentals/{id}/return-decline*.

5.2.4. Generowanie dokumentów PDF

Administrator ma możliwość przeglądania wszystkich dokumentów finansowych w systemie oraz ręcznego generowania nowych rachunków. Dokumenty tworzone są przy użyciu biblioteki iText5 zgodnie z obowiązującymi standardami rachunkowymi. System umożliwia generowanie różnych typów dokumentów (np. faktura, rachunek, potwierdzenie zapłaty).

Dostępne endpointy to *GET /api/v1/transactions/all* oraz POST */api/v1/transactions/bill/{id}*.

5.2.5. Zarządzanie kolekcją DVD

Administrator ma pełny dostęp do katalogu płyt DVD – może dodawać nowe tytuły, edytować istniejące oraz usuwać nieaktualne pozycje. System weryfikuje poprawność wprowadzanych danych, m.in. czy podane gatunki istnieją oraz czy wartości liczbowe (np. cena, liczba kopii) są dodatnie.

Funkcjonalność dostępna jest poprzez endpointy POST */api/v1/dvd/create*, PATCH */api/v1/dvd/{id}/edit* oraz DELETE */api/v1/dvd/{id}*.

5.2.6. Zarządzanie gatunkami filmowymi

Administrator może dodawać nowe gatunki filmowe do systemu lub usuwać istniejące, przy czym usunięcie gatunku jest możliwe jedynie wtedy, gdy nie jest on przypisany do żadnej płyty DVD. System weryfikuje unikalność nazw gatunków, aby uniknąć duplikatów.

Operacje te realizowane są przez endpointy POST */api/v1/genres/create* oraz DELETE */api/v1/genres/{id}/delete*.

5.2.7. Historia rachunków systemowych

Administrator ma wgląd w pełną historię wszystkich rachunków i transakcji w systemie, zarówno bieżących, jak i archiwalnych. Dzięki temu może monitorować obroty i analizować dane finansowe wypożyczalni.

Dane dostępne są pod adresem GET */api/v1/transactions/all*.

6. Fragmenty kodu

Tabela 6.1. Backend – kod z pliku DvdController.java

```
package pl.kielce.tu.backend.controller;

import java.util.List;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PatchMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.Parameter;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
import io.swagger.v3.oas.annotations.security.SecurityRequirement;
import io.swagger.v3.oas.annotations.tags.Tag;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import pl.kielce.tu.backend.model.dto.DvdDto;
import pl.kielce.tu.backend.service.dvd.DvdService;

@RestController
@RequiredArgsConstructor
@RequestMapping("/api/v1/dvd")
@Tag(name = "DVD Management", description = "Operations for managing DVDs in
the rental system")
public class DvdController {

    private final DvdService dvdService;

    @GetMapping
    @Operation(summary = "Get all DVDs with optional filtering", description =
"""
            Retrieves a simplified list of all DVDs available in the rental
system with optional filtering capabilities. \
            Returns basic information about each DVD including id, title, gen-
res, and availability status. \
            Supports filtering by search phrase (matches title/description) and
genres (by name or ID). \
            Multiple filters can be combined for more precise results.""", se-
curity = {
            @SecurityRequirement(name = "accessToken") })
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "List of DVDs re-
trieved successfully", content = @Content(schema = @Schema(example = """
                    [
                      {
                          "id": 120,
                          "title": "Incepcja",
                          "genres": ["Sci-Fi", "Action"],
                          "posterUrl":
"http://example.com/posters/inception.jpg",
```

```
                            "rentalPricePerDay": 5.99,
                            "availabilityStatus": "AVAILABLE"
                        },
                        {
                            "id": 124,
                            "title": "Powrót do przyszłości",
                            "genres": ["Sci-Fi"],
                            "posterUrl": "http://example.com/posters/bttf.jpg",
                            "rentalPricePerDay": 4.99,
                            "availabilityStatus": "AVAILABLE"
                        }
                    ]""")),
            @ApiResponse(responseCode = "500", description = "Internal server
error occurred while retrieving DVDs", content = @Content)
    })
    public ResponseEntity<List<DvdDto>> getAllDvds(
            @Parameter(description = "Search phrase to match against DVD title
and description", example = "matrix") @RequestParam(name = "search-phrase", re-
quired = false) String searchPhrase,
            @Parameter(description = "List of genre names to filter DVDs by",
example = "[\"Action\", \"Sci-Fi\"]") @RequestParam(name = "genres-names", re-
quired = false) List<String> genreNames,
            @Parameter(description = "List of genre identifiers to filter DVDs
by", example = "[1, 2]") @RequestParam(name = "genres-ids", required = false)
List<Long> genreIds) {
        return dvdService.handleGetAllDvdsWithOptionalFilters(searchPhrase,
genreNames, genreIds);
    }

    @GetMapping("{id}")
    @Operation(summary = "Get DVD by ID", description = """
            Retrieves detailed information about a specific DVD by its ID. \
            Returns complete DVD data including all metadata, availability sta-
tus, \
            and rental information. The DVD must exist in the system.""", secu-
rity = {
            @SecurityRequirement(name = "accessToken") })
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "DVD retrieved
successfully", content = @Content(schema = @Schema(example = """
                    {
                      "id": 101,
                      "title": "Matrix",
                      "genres": ["Sci-Fi", "Action"],
                      "releaseYear": 1999,
                      "directors": ["Lana Wachowski", "Lilly Wachowski"],
                      "description": "Neo odkrywa prawdę o rzeczywistości i
staje do walki z systemem, który kontroluje ludzi.",
                      "durationMinutes": 136,
                      "available": true,
                      "copiesAvailable": 5,
                      "rentalPricePerDay": 4.00,
                      "posterUrl":
"https://api.example.com/images/dvds/matrix.jpg",
                      "addedAt": "2025-09-01T14:00:00Z"
                    }""")),
            @ApiResponse(responseCode = "404", description = "DVD not found
with the specified ID", content = @Content),
            @ApiResponse(responseCode = "400", description = "Invalid DVD ID
format", content = @Content),
            @ApiResponse(responseCode = "500", description = "Internal server
error occurred while retrieving DVD", content = @Content)
    })
    public ResponseEntity<DvdDto> getEnhancedDvd(@PathVariable String id) {
```

```
        return dvdService.handleGetDvdById(id);
    }

    @PostMapping("/create")
    @Operation(summary = "Create a new DVD (Admin only)", description = """
            Creates a new DVD entry in the rental system with the provided in-
formation. \
            Required fields: title, at least one genre identifier that exists
in the database, \
            release year, at least one director, description, duration in
minutes, \
            number of available copies, and rental price per day. \
            Optional: poster image as base64 encoded string. The genres must
exist in the system.""")
    @SecurityRequirement(name = "accessToken")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "201", description = "DVD successfully
created"),
            @ApiResponse(responseCode = "401", description = "Unauthorized -
Invalid JWT token", content = @Content),
            @ApiResponse(responseCode = "403", description = "Forbidden - Admin
access required", content = @Content),
            @ApiResponse(responseCode = "422", description = "Validation failed
- invalid DVD data (title too short/long, invalid release year, empty directors
list, invalid description length, invalid duration, negative copies, invalid
price, or non-existent genre identifiers)", content = @Content),
            @ApiResponse(responseCode = "500", description = "Internal server
error occurred during DVD creation", content = @Content)
    })
    public ResponseEntity<Void> createDvd(
            @io.swagger.v3.oas.annotations.parameters.RequestBody(description =
"DVD creation data", required = true, content = @Content(schema =
@Schema(example = """
                    {
                      "title": "Matrix",
                      "genresIdentifiers": [1, 2],
                      "releaseYear": 1999,
                      "directors": ["Lana Wachowski", "Lilly Wachowski"],
                      "description": "Neo odkrywa prawdę o rzeczywistości i
staje do walki z systemem.",
                      "durationMinutes": 136,
                      "available": true,
                      "copiesAvailable": 5,
                      "rentalPricePerDay": 4.00,
                      "posterImage": "da-
ta:image/jpeg;base64,/9j/4AAQSkZJRgABAQEASABIAAD..."
                    }""")))) @Valid @RequestBody DvdDto dvdDto) {
        return dvdService.handleCreateDvd(dvdDto);
    }

    @PatchMapping("{id}/edit")
    @Operation(summary = "Update DVD information (Admin only)", description =
"""
            Partially updates an existing DVD's information. The DVD must exist
in the system. \
            All fields are optional but at least one field must be provided for
the update. \
            Field validations are the same as for creation. Poster image can be
provided \
            as base64 encoded string.""")
    @SecurityRequirement(name = "accessToken")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "202", description = "DVD successfully
updated"),
```

```
            @ApiResponse(responseCode = "400", description = "Invalid DVD ID
format", content = @Content),
            @ApiResponse(responseCode = "401", description = "Unauthorized -
Invalid JWT token", content = @Content),
            @ApiResponse(responseCode = "403", description = "Forbidden - Admin
access required", content = @Content),
            @ApiResponse(responseCode = "404", description = "DVD not found
with the specified ID", content = @Content),
            @ApiResponse(responseCode = "422", description = "Validation error
- invalid field value, no fields provided, or non-existent genre identifiers",
content = @Content),
            @ApiResponse(responseCode = "500", description = "Internal server
error occurred during DVD update", content = @Content)
    })
    public ResponseEntity<Void> editDvd(
            @PathVariable String id,
            @io.swagger.v3.oas.annotations.parameters.RequestBody(description =
"Partial DVD update data. At least one field must be provided.", required =
true, content = @Content(schema = @Schema(example = """
                    {
                      "title": "Matrix",
                      "genresIdentifiers": [1, 3],
                      "releaseYear": 2001,
                      "directors": ["Jan Wachowski", "Lilly Wachowski"],
                      "description": "Nao odkrywa prawdę o rzeczywistości i
staje do walki z systemem.",
                      "durationMinutes": 138,
                      "available": true,
                      "copiesAvailable": 5,
                      "rentalPricePerDay": 5.00,
                      "posterImage": "da-
ta:image/jpeg;base64,/9j/4AAQSkZJRgABAQEASABIAAD..."
                    }""")))  @Valid @RequestBody DvdDto dvdDto) {
        return dvdService.handleUpdateDvd(id, dvdDto);
    }

}
```

Kod z tabeli 6.1 przedstawia kontroler REST w aplikacji Spring Boot o nazwie DvdController, który obsługuje operacje związane z zarządzaniem płytami DVD w systemie wypożyczalni. Klasa jest oznaczona adnotacjami @RestController i @RequestMapping("/api/v1/dvd"), dzięki czemu udostępnia zestaw endpointów API dostępnych pod tym adresem. Wstrzykiwana jest do niej usługa DvdService, odpowiedzialna za logikę biznesową. Kontroler udostępnia kilka metod: pobieranie listy wszystkich płyt DVD z możliwością filtrowania po tytule, opisie lub gatunkach (GET /api/v1/dvd), pobieranie szczegółowych informacji o pojedynczej płycie po jej identyfikatorze (GET /api/v1/dvd/{id}), tworzenie nowego wpisu DVD (POST /api/v1/dvd/create) oraz częściową edycję istniejącego wpisu (PATCH /api/v1/dvd/{id}/edit). Operacje tworzenia i edycji są przeznaczone wyłącznie dla administratorów i wymagają autoryzacji. Kod wykorzystuje adnotacje OpenAPI (Swagger) do automatycznej dokumentacji API, opisując każdy endpoint, możliwe odpowiedzi HTTP oraz przykładowe dane wejściowe i wyjściowe. Dodatkowo zastosowano walidację danych wejściowych (@Valid) i obsługę różnych kodów błędów (np. 400, 401, 403, 404, 422, 500). W efekcie DvdController pełni

rolę warstwy prezentacji — przyjmuje żądania HTTP, przekazuje dane do serwisu, a następnie zwraca odpowiedzi w formacie JSON.

Tabela 6.2. Frontend – kod z pliku DashboardHeader.tsx

```
import {
  BookCheck,
  BookOpen,
  BookUp,
  CircleUser,
  FilePlus,
  Film,
  History,
  LogOut,
  Menu,
  Search,
  Sparkles,
  Ticket,
} from "lucide-react";
import { Link, useLocation } from "react-router-dom";
import { Button } from "@/components/ui/button";
import {
  DropdownMenu,
  DropdownMenuContent,
  DropdownMenuItem,
  DropdownMenuLabel,
  DropdownMenuSeparator,
  DropdownMenuTrigger,
} from "@/components/ui/dropdown-menu";
import { Input } from "@/components/ui/input";
import { Sheet, SheetContent, SheetTrigger } from "@/components/ui/sheet";
import { useAuth } from "@/contexts/AuthContext";
import { FilterGroup } from "./FilterGroup";

export function DashboardHeader({
  searchPhrase,
  onSearchChange,
  selectedGenres,
  onGenreChange,
}: Partial<{
  searchPhrase: string;
  onSearchChange: (phrase: string) => void;
  selectedGenres: number[];
  onGenreChange: (genreId: number, checked: boolean) => void;
}>) {
  const { logout, isAdmin } = useAuth();
  const pathname = useLocation().pathname;
  const shouldShowFilterGroup = pathname === "/dashboard";

  const userLinks = [
    {
      to: "/reservations",
      label: "Moje rezerwacje",
      icon: <Ticket className="mr-2 h-4 w-4" />,
    },
    {
      to: "/rentals",
      label: "Moje wypożyczenia",
      icon: <Film className="mr-2 h-4 w-4" />,
    },
    {
      to: "/recommendations",
```

```
      label: "Moje rekomendacje",
      icon: <Sparkles className="mr-2 h-4 w-4" />,
    },
    {
      to: "/transactions-history",
      label: "Historia wypożyczeń",
      icon: <History className="mr-2 h-4 w-4" />,
    },
  ];

  const adminLinks = [
    {
      to: "/admin/dvd/create",
      label: "Dodaj nowe DVD",
      icon: <FilePlus className="mr-2 h-4 w-4" />,
    },
    {
      to: "/admin/reservations",
      label: "Zarządzanie rezerwacjami",
      icon: <BookCheck className="mr-2 h-4 w-4" />,
    },
    {
      to: "/admin/returns",
      label: "Zarządzanie zwrotami",
      icon: <BookUp className="mr-2 h-4 w-4" />,
    },
    {
      to: "/admin/genres",
      label: "Zarządzanie gatunkami",
      icon: <BookOpen className="mr-2 h-4 w-4" />,
    },
  ];

  return (
    <header className="flex h-14 items-center gap-4 border-b bg-muted/40 px-4
lg:h-[60px] lg:px-6">
      <Sheet>
        <SheetTrigger asChild>
          <Button variant="outline" size="icon" className="shrink-0 md:hidden">
            <Menu className="h-5 w-5" />
            <span className="sr-only">Toggle navigation menu</span>
          </Button>
        </SheetTrigger>
        <SheetContent side="left" className="flex flex-col">
          <nav className="grid gap-2 text-lg font-medium p-2">
            <div className="mb-8">
              <h1 className="text-2xl font-bold flex items-center animate-pulse
select-none p-2">
                <span className="text-indigo-500">O</span>pasRent
              </h1>
            </div>
            {shouldShowFilterGroup && (
              <FilterGroup
                isMobile
                selectedGenres={selectedGenres}
                onGenreChange={onGenreChange}
              />
            )}
          </nav>
        </SheetContent>
      </Sheet>
      <div className="w-full flex-1">
        {shouldShowFilterGroup && (
          <form onSubmit={(e) => e.preventDefault()}>
```

```
            <div className="relative">
              <Search className="absolute left-2.5 top-2.5 h-4 w-4 text-muted-
foreground" />
              <Input
                type="search"
                placeholder="Szukaj filmów..."
                className="w-full appearance-none bg-background pl-8 shadow-
none md:w-2/3 lg:w-1/3"
                value={searchPhrase ?? ""}
                onChange={(e) => onSearchChange?.(e.target.value)}
              />
            </div>
          </form>
        )}
      </div>
      <DropdownMenu>
        <DropdownMenuTrigger asChild>
          <Button
            variant="secondary"
            size="icon"
            className="rounded-full cursor-pointer"
          >
            <CircleUser className="h-5 w-5" />
            <span className="sr-only">Toggle user menu</span>
          </Button>
        </DropdownMenuTrigger>
        <DropdownMenuContent align="end">
          <DropdownMenuLabel>Moje konto</DropdownMenuLabel>
          <DropdownMenuSeparator />
          {(isAdmin ? adminLinks : userLinks).map((link) => (
            <DropdownMenuItem key={link.to} asChild>
              <Link to={link.to}>
                {link.icon}
                {link.label}
              </Link>
            </DropdownMenuItem>
          ))}
          <DropdownMenuSeparator />
          <DropdownMenuItem asChild>
            <Link to="/profile">
              <CircleUser className="mr-2 h-4 w-4" />
              Mój profil
            </Link>
          </DropdownMenuItem>
          <DropdownMenuSeparator />
          <DropdownMenuItem
            onClick={logout}
            className="text-red-500 focus:text-red-500 focus:bg-red-500/10"
          >
            <LogOut className="mr-2 h-4 w-4" />
            Wyloguj
          </DropdownMenuItem>
        </DropdownMenuContent>
      </DropdownMenu>
    </header>
  );
}
```

```java
package pl.kielce.tu.backend.controller;

import java.util.List;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PatchMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.Parameter;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
import io.swagger.v3.oas.annotations.security.SecurityRequirement;
import io.swagger.v3.oas.annotations.tags.Tag;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import pl.kielce.tu.backend.model.dto.DvdDto;
import pl.kielce.tu.backend.service.dvd.DvdService;

@RestController
@RequiredArgsConstructor
@RequestMapping("/api/v1/dvd")
@Tag(name = "DVD Management", description = "Operations for managing DVDs in
the rental system")
public class DvdController {

    private final DvdService dvdService;

    @GetMapping
    @Operation(summary = "Get all DVDs with optional filtering", description =
"""
            Retrieves a simplified list of all DVDs available in the rental
system with optional filtering capabilities. \
            Returns basic information about each DVD including id, title, gen-
res, and availability status. \
            Supports filtering by search phrase (matches title/description) and
genres (by name or ID). \
            Multiple filters can be combined for more precise results.""", se-
curity = {
            @SecurityRequirement(name = "accessToken") })
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "List of DVDs re-
trieved successfully", content = @Content(schema = @Schema(example = """
                    [
                      {
                          "id": 120,
                          "title": "Incepcja",
                          "genres": ["Sci-Fi", "Action"],
                          "posterUrl":
"http://example.com/posters/inception.jpg",
                          "rentalPricePerDay": 5.99,
                          "availabilityStatus": "AVAILABLE"
                      },
                      {
                          "id": 124,
                          "title": "Powrót do przyszłości",
```

```
                            "genres": ["Sci-Fi"],
                            "posterUrl": "http://example.com/posters/bttf.jpg",
                            "rentalPricePerDay": 4.99,
                            "availabilityStatus": "AVAILABLE"
                        }
                    ]""")),
            @ApiResponse(responseCode = "500", description = "Internal server
error occurred while retrieving DVDs", content = @Content)
    })
    public ResponseEntity<List<DvdDto>> getAllDvds(
            @Parameter(description = "Search phrase to match against DVD title
and description", example = "matrix") @RequestParam(name = "search-phrase", re-
quired = false) String searchPhrase,
            @Parameter(description = "List of genre names to filter DVDs by",
example = "[\"Action\", \"Sci-Fi\"]") @RequestParam(name = "genres-names", re-
quired = false) List<String> genreNames,
            @Parameter(description = "List of genre identifiers to filter DVDs
by", example = "[1, 2]") @RequestParam(name = "genres-ids", required = false)
List<Long> genreIds) {
        return dvdService.handleGetAllDvdsWithOptionalFilters(searchPhrase,
genreNames, genreIds);
    }

    @GetMapping("{id}")
    @Operation(summary = "Get DVD by ID", description = """
            Retrieves detailed information about a specific DVD by its ID. \
            Returns complete DVD data including all metadata, availability sta-
tus, \
            and rental information. The DVD must exist in the system.""", secu-
rity = {
            @SecurityRequirement(name = "accessToken") })
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "DVD retrieved
successfully", content = @Content(schema = @Schema(example = """
                    {
                      "id": 101,
                      "title": "Matrix",
                      "genres": ["Sci-Fi", "Action"],
                      "releaseYear": 1999,
                      "directors": ["Lana Wachowski", "Lilly Wachowski"],
                      "description": "Neo odkrywa prawdę o rzeczywistości i
staje do walki z systemem, który kontroluje ludzi.",
                      "durationMinutes": 136,
                      "available": true,
                      "copiesAvailable": 5,
                      "rentalPricePerDay": 4.00,
                      "posterUrl":
"https://api.example.com/images/dvds/matrix.jpg",
                      "addedAt": "2025-09-01T14:00:00Z"
                    }""")),
            @ApiResponse(responseCode = "404", description = "DVD not found
with the specified ID", content = @Content),
            @ApiResponse(responseCode = "400", description = "Invalid DVD ID
format", content = @Content),
            @ApiResponse(responseCode = "500", description = "Internal server
error occurred while retrieving DVD", content = @Content)
    })
    public ResponseEntity<DvdDto> getEnhancedDvd(@PathVariable String id) {
        return dvdService.handleGetDvdById(id);
    }

    @PostMapping("/create")
    @Operation(summary = "Create a new DVD (Admin only)", description = """
            Creates a new DVD entry in the rental system with the provided in-
```

```
formation. \
            Required fields: title, at least one genre identifier that exists
in the database, \
            release year, at least one director, description, duration in
minutes, \
            number of available copies, and rental price per day. \
            Optional: poster image as base64 encoded string. The genres must
exist in the system.""")
    @SecurityRequirement(name = "accessToken")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "201", description = "DVD successfully
created"),
            @ApiResponse(responseCode = "401", description = "Unauthorized -
Invalid JWT token", content = @Content),
            @ApiResponse(responseCode = "403", description = "Forbidden - Admin
access required", content = @Content),
            @ApiResponse(responseCode = "422", description = "Validation failed
- invalid DVD data (title too short/long, invalid release year, empty directors
list, invalid description length, invalid duration, negative copies, invalid
price, or non-existent genre identifiers)", content = @Content),
            @ApiResponse(responseCode = "500", description = "Internal server
error occurred during DVD creation", content = @Content)
    })
    public ResponseEntity<Void> createDvd(
            @io.swagger.v3.oas.annotations.parameters.RequestBody(description =
"DVD creation data", required = true, content = @Content(schema =
@Schema(example = """
                    {
                      "title": "Matrix",
                      "genresIdentifiers": [1, 2],
                      "releaseYear": 1999,
                      "directors": ["Lana Wachowski", "Lilly Wachowski"],
                      "description": "Neo odkrywa prawdę o rzeczywistości i
staje do walki z systemem.",
                      "durationMinutes": 136,
                      "available": true,
                      "copiesAvailable": 5,
                      "rentalPricePerDay": 4.00,
                      "posterImage": "da-
ta:image/jpeg;base64,/9j/4AAQSkZJRgABAQEASABIAAD..."
                    }""".trim()))) @Valid @RequestBody DvdDto dvdDto) {
        return dvdService.handleCreateDvd(dvdDto);
    }

    @PatchMapping("{id}/edit")
    @Operation(summary = "Update DVD information (Admin only)", description =
"""
            Partially updates an existing DVD's information. The DVD must exist
in the system. \
            All fields are optional but at least one field must be provided for
the update. \
            Field validations are the same as for creation. Poster image can be
provided \
            as base64 encoded string.""")
    @SecurityRequirement(name = "accessToken")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "202", description = "DVD successfully
updated"),
            @ApiResponse(responseCode = "400", description = "Invalid DVD ID
format", content = @Content),
            @ApiResponse(responseCode = "401", description = "Unauthorized -
Invalid JWT token", content = @Content),
            @ApiResponse(responseCode = "403", description = "Forbidden - Admin
access required", content = @Content),
```

```
            @ApiResponse(responseCode = "404", description = "DVD not found
with the specified ID", content = @Content),
            @ApiResponse(responseCode = "422", description = "Validation error
- invalid field value, no fields provided, or non-existent genre identifiers",
content = @Content),
            @ApiResponse(responseCode = "500", description = "Internal server
error occurred during DVD update", content = @Content)
    })
    public ResponseEntity<Void> editDvd(
            @PathVariable String id,
            @io.swagger.v3.oas.annotations.parameters.RequestBody(description =
"Partial DVD update data. At least one field must be provided.", required =
true, content = @Content(schema = @Schema(example = """
                    {
                      "title": "Matrix",
                      "genresIdentifiers": [1, 3],
                      "releaseYear": 2001,
                      "directors": ["Jan Wachowski", "Lilly Wachowski"],
                      "description": "Nao odkrywa prawdę o rzeczywistości i
staje do walki z systemem.",
                      "durationMinutes": 138,
                      "available": true,
                      "copiesAvailable": 5,
                      "rentalPricePerDay": 5.00,
                      "posterImage": "da-
ta:image/jpeg;base64,/9j/4AAQSkZJRgABAQEASABIAAD..."
                    }""")))) @Valid @RequestBody DvdDto dvdDto) {
        return dvdService.handleUpdateDvd(id, dvdDto);
    }

}
```

Kod z tabeli 6.2 przedstawia komponent React DashboardHeader, który odpowiada za nagłówek panelu użytkownika w aplikacji typu dashboard. Komponent obsługuje pasek wyszukiwania filmów, menu użytkownika oraz panel nawigacyjny dla użytkowników i administratorów. Nagłówek dynamicznie dopasowuje zawartość w zależności od ścieżki URL i roli użytkownika. Jeśli użytkownik znajduje się na stronie /dashboard, wyświetlany jest pasek wyszukiwania oraz komponent FilterGroup umożliwiający filtrowanie filmów po gatunkach. Menu użytkownika jest realizowane za pomocą DropdownMenu, które zawiera różne linki w zależności od tego, czy użytkownik jest administratorem, czy zwykłym użytkownikiem. Dodatkowo dostępny jest przycisk wylogowania oraz link do profilu. Komponent wykorzystuje elementy z biblioteki lucide-react do ikon, komponenty UI (Button, Input, Sheet, DropdownMenu) do interfejsu oraz hooki React Router (useLocation) i kontekst uwierzytelniania (useAuth) do obsługi stanu aplikacji i nawigacji. Całość jest responsywna – w widokach mobilnych menu jest dostępne jako wysuwany panel (Sheet).

7. Wnioski

Projekt OpasRent został opracowany w celu zaprezentowania praktycznych aspektów projektowania i implementacji inteligentnych usług informacyjnych.

Zrealizowany system integruje trzy główne warstwy: backendową opartą na Java i Spring Boot w architekturze MVC, frontendową realizowaną w React i TypeScript oraz bazę danych PostgreSQL, zapewniającą trwałe i bezpieczne przechowywanie danych. W ramach projektu wdrożono mechanizmy bezpieczeństwa, w tym szyfrowanie komunikacji za pomocą SSL/TLS, uwierzytelnianie i autoryzację oparte na JWT oraz bezpieczne przechowywanie haseł użytkowników przy użyciu algorytmu argon2

System umożliwia pełną obsługę procesu wypożyczania płyt DVD, w tym rejestrację i logowanie użytkowników, przeglądanie kolekcji i szczegółów filmów, składanie rezerwacji, zgłaszanie zwrotów, przegląd historii wypożyczeń oraz generowanie dokumentów finansowych w formacie PDF. Dodatkowo, funkcjonalność rekomendacji filmowych umożliwia dostarczanie spersonalizowanych sugestii dla użytkowników w oparciu o historię wypożyczeń, preferencje gatunkowe oraz wiek, co stanowi przykład wykorzystania inteligentnych mechanizmów analizy danych. Wszystkie funkcjonalności działają zgodnie z założeniami projektowymi, a system zapewnia stabilną i bezpieczną pracę zarówno dla użytkowników, jak i administratorów.

Projekt umożliwił praktyczne zastosowanie zdobytej wiedzy w zakresie tworzenia aplikacji full-stack, integracji warstw systemu, zarządzania danymi oraz zapewnienia bezpieczeństwa i spersonalizowanej obsługi użytkownika. Realizacja projektu przyczyniła się do pogłębienia umiejętności projektowania systemów informacyjnych oraz implementacji nowoczesnych technologii webowych i bazodanowych.