

POLITECHNIKA WROCŁAWSKA

WYDZIAŁ ELEKTRONIKI

KIERUNEK: Informatyka (INF)
SPECJALNOŚĆ: Inżynieria systemów informatycznych (INS)

Projekt z rozproszonych i obiektowych systemów baz danych

Rozproszony system bazodanowy przeznaczony do
obsługi kina

AUTOR:
Radosław Taborski - 209347
Piotr Konieczny - 209174

PROWADZĄCY PROJEKT:
dr inż. Robert Wójcik

OCENA PROJEKTU:

Spis treści

Spis rysunków	4
Spis tabel	5
Spis listingów	6
1 Wstęp	7
1.1 Cele projektu	7
1.2 Założenia projektowe	7
1.3 Zakres projektu	8
2 Replikacja w systemie baz danych MySQL	9
2.1 Pojęcie replikacji i podstawowe informacje	9
2.2 Replikacja master-slave	9
2.3 Testowanie mechanizmów replikacji	10
3 Model konceptualny i fizyczny baz danych	12
3.1 Model konceptualny	12
3.2 Model fizyczny	13
4 Implementacja baz danych w środowisku MySQL	15
4.1 Konfiguracja kontenerów z środowiskiem MySQL 5.7 i phpMyAdmin	15
4.2 Konfiguracja mechanizmów replikacji master-slave	18
4.3 Realizacja bazy danych	19
4.3.1 Tabele	19
4.3.2 Widoki	20
4.3.3 Procedury	22
5 Projekt i implementacja aplikacji klienckiej oraz REST API	24
5.1 Funkcje aplikacji	24
5.1.1 Diagram przypadków użycia	24
5.1.2 Scenariusz wybranych przypadków użycia	25
5.2 Realizacja wybranych funkcjonalności aplikacji	26
5.2.1 Interfejs aplikacji	28
5.3 Realizacja REST API	33

SPIS TREŚCI	3
6 Wdrożenie i testowanie aplikacji klienckiej	37
6.1 Wdrożenie	37
6.1.1 Wdrażanie API REST-owego	37
6.1.2 Zmiana domyślnego portu	38
6.2 Testy działania	39
6.2.1 Dodawanie rekordu do rozproszonej bazy danych	39
6.2.2 Usuwanie rekordu z rozproszonej bazy danych	41
7 Podsumowanie	42
Literatura	43

Spis rysunków

1.1	Struktura systemu	8
3.1	Model konceptualny węzła rozproszonej bazy danych wykonany w programie Microsoft Visio	12
3.2	Model fizyczny węzła rozproszonej bazy danych wykonany w programie Microsoft Visio	13
4.1	Konfiguracja serwera master w phpmyadmin	19
4.2	Konfiguracja serwera slave w phpmyadmin	19
4.3	Generowanie tabeli seansów	20
4.4	Generowanie tabeli biletów - przykład użycia klucza głównego, kluczy obcych i ograniczenia "not null"	20
4.5	Generowanie tabeli sal - przykład użycia CHECK	20
4.6	Wygenerowane widoki	21
4.7	Generowanie widoku seansów	21
4.8	Generowanie widoku Miejsc	21
4.9	Przykładowa procedura	22
4.10	Przykładowa procedura	23
5.1	Diagram przypadków użycia	24
5.2	Rejestracja użytkownika	28
5.3	Rezerwacja biletów	29
5.4	Widok biletów zarezerwowanych przez użytkownika	29
5.5	Widok filmów dostępnych w kinie	29
5.6	Dodawanie nowego filmu	30
5.7	Edycja filmu	30
5.8	Widok seansów dostępnych w kinie	31
5.9	Dodawanie seansu	31
5.10	Edycja seansu	32
5.11	Widok sal dostępnych w kinie	32
5.12	Podanie błędnego loginu i hasła	33
5.13	Błąd komunikacji z serwerem	33
6.1	Dodawanie nowego filmu poprzez aplikację	40
6.2	Tabela Filmy na węźle <i>master</i> o porcie 8082	40
6.3	Tabela Filmy na węźle <i>slave</i> 1 o porcie 8083	41
6.4	Nowy film wyświetlany w aplikacji	41

Spis tabel

5.1	Scenariusz rejestracji	25
5.2	Scenariusz logowania	25
5.3	Scenariusz zarządzania kinem	26
5.4	Przykładowe zastosowanie REST API	33

Spis listingów

4.1	Instalacja programu docker w systemie ubuntu 16.04	15
4.2	Pobranie obrazów z repozytorium	15
4.3	Instalacja narzędzia Docker Compose	15
4.4	Utworzony plik docker-compose.yml	16
4.5	Konfiguracja serwera master- modyfikacja pliku my.cnf	18
4.6	Konfiguracja opóźnienia dla węzła slave	19
5.1	Implementacja funkcji GET po stronie aplikacji	26
5.2	Implementacja funkcji POST po stronie aplikacji	27
5.3	Skrypt PHP API aplikacji	33
5.4	Skrypt PHP obsługujący zapytania GET	35
6.1	Instalacja Node.js w systemie operacyjnym Ubuntu 16.04	37
6.2	Instalacja Angular-CLI w systemie operacyjnym Ubuntu 16.04	37
6.3	Uruchomienie aplikacji na localhost:4200 16.04	37
6.4	Zawartość pliku server.js	38
6.5	Zawartość pliku api.js	39
6.6	Uruchomienie aplikacji klienckiej na własnym porcie	39

Rozdział 1

Wstęp

1.1 Cele projektu

Celem projektu jest stworzenie systemu wspomagającego obsługę kina w oparciu o rozproszoną i obiektową bazę danych. System będzie umożliwiać zarządzanie kinem – z wykorzystaniem relacyjnych baz danych replikujących między sobą dane. W pojedynczym węźle bazy danych zawarte będą tabele opisujące między innymi – seanse filmowe, przydział ich do poszczególnych sal kinowych. Aplikacja będzie umożliwiać ponadto tworzenie nowych wpisów w zależności od rodzaju użytkownika obsługującego program. Pracownik kina będzie wprowadzać nowe seanse do bazy; podczas bezpośredniej sprzedaży biletów będzie również wykreślał miejsca na sali już zajęte – miejsca zawarte na biletach, poszukiwanie rezerwacji wykonanej na konkretną osobę (po imieniu lub nazwisku, czy też numerze rezerwacji). Użytkownik(Klient) będzie mógł rezerwować konkretne miejsce na określony seans.

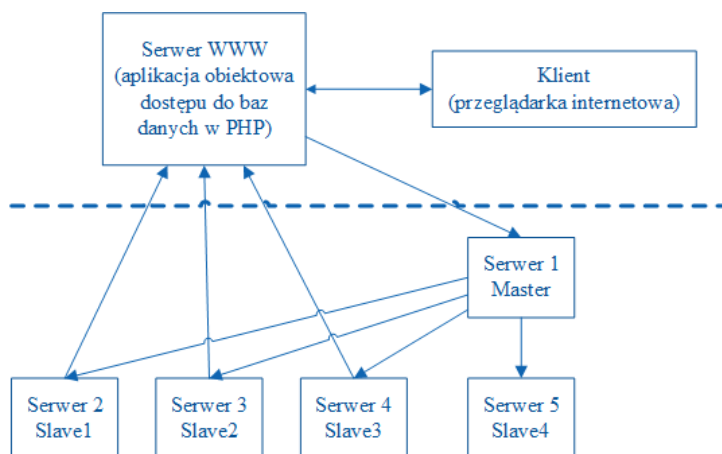
1.2 Założenia projektowe

Projekt został wykonany przy użyciu MySQL 5.7. Rozproszoność systemu oparta została o kontenery programu docker, na których skonfigurowane zostały węzły zarówno *slave* jak i *master*. W trakcie realizacji projektu zostały wykorzystane mechanizmy replikacji master-slave oraz master-slave z opóźnieniem. Do wykonania projektu bazy danych wykorzystane zostało narzędzie Microsoft Visio. Zarządzanie bazą danych odbywało się z poziomu narzędzia zwanego phpMyAdmin. Aplikacja kliencka została wykonana w technologii webowej, z wykorzystaniem platformy programistycznej Angular2 oraz języka programowania TypeScript. Komunikacja między bazą danych a aplikacją kliencką zapewnia API REST-owe napisane w języku PHP.

System składa się z:

- jednego węzła master;
- trzech zwyczajnych węzłów slave;
- jednego węzła slave z opóźnieniem;
- aplikacji klienckiej;
- API REST-owego.

API REST-owe dodaje dane wysłane z aplikacji klienckiej do węzła master, natomiast odczyt danych dokonuje się losowo z jednego z trzech zwyczajnych węzłów slave. Węzeł opóźniony służy jako kopia zapasowa danych, głównie na wypadek wystąpienia poleceń typu DROP w węźle master. Struktura systemu została przedstawiona na rysunku 1.1.



Rysunek 1.1 Struktura systemu

1.3 Zakres projektu

Zakres projektu dotyczy zaprojektowania i implementacji rozproszonego systemu bazodanowego dla kina. Projekt składa się z kilku etapów.

- Określenie wymagań funkcjonalnych aplikacji bazodanowej
- Testowanie mechanizmów replikacji oraz rozpraszania danych
- Opracowanie modelu koncepcyjnego i fizycznego bazy danych
- Implementacja bazy danych, procedur i widoków
- Projektowanie i implementacja aplikacji klienckiej
- Wdrożenie i testowanie aplikacji klienckiej

Rozdział 2

Replikacja w systemie baz danych MySQL

2.1 Pojęcie replikacji i podstawowe informacje

Replikacja bazy danych polega na powielaniu bazy danych między różnymi serwerami baz danych, co ma miejsce np. przy pracy w klastrze. Replikacja pozwala na:

- **skalowalność:** dzięki temu możliwe jest rozłożenie obciążenia między wieloma serwerami; operacje zapisu i aktualizacji rekordów mogą odbywać się na jednym serwerze, a pobieranie i przeszukiwanie danych na innych, a znacznie obciążające serwer operacje - na jeszcze innych. Na jednej z kopii mogą pracować analitycy, deweloperzy itp.
- **bezpieczeństwo:** dzięki replikacji tworzymy kopie istniejącej bazy produkcyjnej, które co prawda nie uchronią nas przed operacjami typu *DROP*, ale zapewnią ciągły dostęp do bazy danych w przypadku awarii sprzętu głównego serwera.
- **analizę:** skomplikowane operacje analityczne, różnego rodzaju przeliczenia i analizy statystyczne mogą być wykonywane na osobnym serwerze bez obciążania głównej bazy.
- **separację:** możemy udostępnić kopię bazy produkcyjnej dla deweloperów i testerów, aby swoje prace wykonywali na kopii bazy danych.

Replikację można podzielić na:

- replikację typu *master-slave* - wtedy na bazie produkcyjnej (*master*) wykonywane są operacje modyfikacji danych, natomiast na pozostałe przenoszona jest kopia bazy danych z serwera głównego
- replikację typu *master-master*, inaczej duplikacja, gdzie zmiany czy modyfikacje danych mogą być wykonane na dowolnym komputerze i dochodzi do obustronnej synchronizacji baz danych; dzięki takiemu rozwiązaniu zmiany przeprowadzone na jednej z baz danych zostaną również prowadzone na pozostałych.

2.2 Replikacja master-slave

Replikacja danych w *MySQL* opiera się o bardzo prostą zasadę: serwer główny (*master*) prowadzi swego rodzaju dziennik, w którym zapisuje każdą czynność, którą wykonał. Wykorzystuje do tego logi binarne zawierające instrukcje, które wykonał *master*. Serwer zapasowy

(*slave*) odczytuje te dane i kolejno wykonuje zapytania, uzupełniając bazę kolejnymi rekordami. Efektem tej pracy są dwie identyczne bazy danych.

Po skonfigurowaniu mechanizmu replikacji na serwerze master pojawia się dodatkowy wątek, który odpowiada za wysyłanie bin-logów do serwerów *slave*. Z kolei serwer zapasowy ma dwa wątki:

- **I/O Thread [wątek wejścia-wyjścia]** - odpowiada za odbieranie dziennika od serwera głównego i zapisuje go w plikach tymczasowych (relay-log),
- **SQL Thread [wątek SQL]** - zajmuje się parsowaniem tych plików i wykonywaniem zapytań do bazy.

System bazodanowy MySQL umożliwia trzy różne metody replikacji, co przekłada się na format danych zapisywanych do bin-logów. Za wybór metody replikacji odpowiada zmienna `binlog_format`, która może przyjąć wartość: ROW, STATEMENT, MIXED.

Metody replikacji:

- **SBR (statement-based replication)** - w tym trybie, serwer do pliku zapisuje zapytania jakie wykonał.
- **RBR (row-based replication)** - do bin-logów zapisywane są wyniki działań zapytań na serwerze master. Zapisywana jest informacja jaki rekord został w jaki sposób zmieniony.
- **MFL (mixed-format logging)** - jest to połączenie dwóch powyższych typów replikacji.

Technika replikacji **SBR** jest bardzo szybka i wydajna, ponieważ w jej przypadku serwer główny zapisuje do pliku zapytanie jakie wykonał, następnie serwer zapasowy je odczytuje i wykonuje. Niestety do pliku logów zapisywane są tylko zapytania SQL, co przysporzy nam problemów, gdy nasze zapytania będą bardziej złożone.

Problem ten rozwiązała metoda **RBR**, która do bin-logów zapisuje wyłącznie zmiany jakie zaszły po wykonaniu polecenia - logowane są informacje na temat sposobu modyfikacji konkretnych rekordów. Niestety metoda ta jest znacznie wolniejsza od poprzedniej oraz zwiększa ilość wysyłanych danych pomiędzy replikującymi się serwerami.

Z pomocą przyszła nam metoda **MFL**, w której w większości przypadków, logowane są zapytania SQL tak jak w przypadku **SBR**, natomiast dla zapytań, których wynik nie jest przewidywalny, włączana jest replikacja **RBR**.

2.3 Testowanie mechanizmów replikacji

W projekcie użyta zostanie konfiguracja Master-Slave. Poniżej wypunktowane zostały wnioski z przeprowadzonych testów replikacji bazodanowej MySQL.

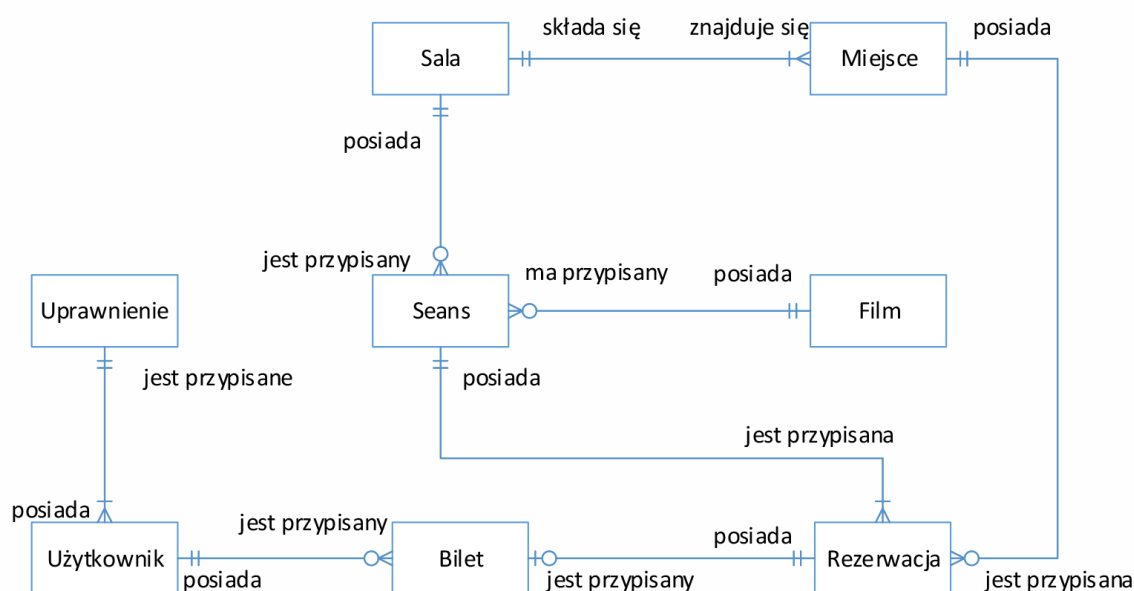
- Po skonfigurowaniu replikacji wymagane jest utworzenie bazy danych *slave*, która posiada tą samą strukturę co *master*
- Dane zawarte w bazie *master* nie zostaną automatycznie skopiowane do bazy *slave* po skonfigurowaniu replikacji. Należy ręcznie zsynchronizować dane w tabelach.

- W przypadku wyłączenia bazy danych *slave* i modyfikacji bazy *master* baza *slave* zostanie zsynchronizowana po ponownym podłączeniu do sieci.
- *Slave* – odczyt; *master* – zapis, modyfikacja, usuwanie. Gdy *slave* jest wyłączony zapytania GET są wysyłane do innego *slave*, a w ostateczności *mastera*. Gdy jest wyłączony *master* można jedynie odczytywać dane z serwera. Natomiast na ten czas jakakolwiek modyfikacja danych jest niemożliwa.
- Od wersji *MySQL* 5.7 możliwa jest replikacja Master-Slave z opóźnieniem. Domyślnie *master* natychmiastowo wysyła bin-log do węzłów typu *slave*, jednak możliwe jest celowe wprowadzenie opóźnienia, np. w celu ochrony bazy danych przed poleceniem DROP, który wykonany na *masterze*, usunie również bazę/ tabelę na standardowych węzłach *slave*. Odpowiednio duże opóźnienie daje możliwość na reakcję ze strony admina, tak aby w razie konieczności ocalić opóźniony węzeł.

Rozdział 3

Model konceptualny i fizyczny baz danych

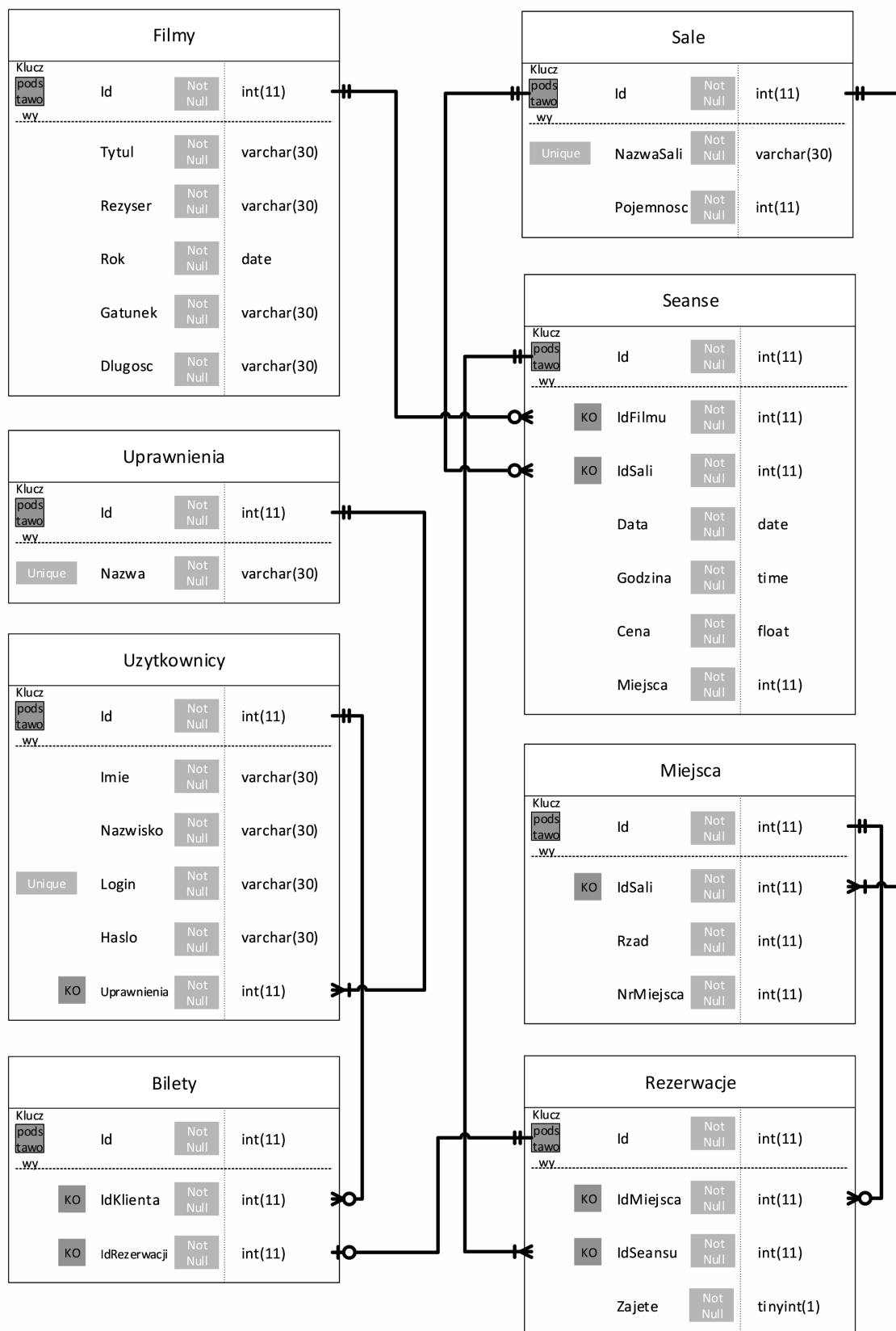
3.1 Model konceptualny



Rysunek 3.1 Model konceptualny węzła rozproszonej bazy danych wykonany w programie Microsoft Visio

W bazie danych dominują relacje typu jeden do wielu. Na każdej sali może odbywać się wiele seansów, natomiast każdy seans posiada tylko jeden film i jedną salę. Każde miejsce również ma przypisaną jedną konkretną salę. Na jedno miejsce może przypadać wiele rezerwacji, w zależności od seansu. Również każdy użytkownik może mieć wiele kupionych biletów lub nie mieć ich wcale. Oraz istnieje wielu użytkowników o tych samych uprawnieniach. Jedyną relacją, która nie jest jeden do wielu, to powiązanie między tabelą *Bilety* i *Rezerwacje*. Na każdą rezerwację może być maksymalnie jeden bilet, który odpowiada jednej rezerwacji, rozumianej tutaj jako miejsce na konkretny seans, które może być zajęte lub też nie.

3.2 Model fizyczny



Rysunek 3.2 Model fizyczny węzła rozproszonej bazy danych wykonany w programie Microsoft Visio

Tabele:

- *Filmy* - przechowuje najważniejsze informacje o filmach, takie jak ich tytuł, nazwiska reżyserów, daty premiery, gatunek oraz czas trwania, żadna z tych wartości nie może pozostać pusta;
- *Sale* - jedynymi niezbędnymi w projekcie parametrami charakteryzującymi sale są jej unikalna nazwa i pojemność ukazująca liczbę dostępnych miejsc siedzących;
- *Seanse* - każdy seans ma przypisany film oraz salę. Dodatkowo przechowuje takie informacje jak data i godzina wyświetlenia seansu, cenę oraz pozostałą liczbę wolnych miejsc;
- *Miejsca* - w tej tabeli przechowywane są wszystkie miejsca siedzące dostępne w kinie. Każde miejsce znajduje się w swojej określonej sali. Każde miejsce dodatkowo ma też numer i rząd w którym się znajduje na sali;
- *Rezerwacje* - jest to spis wszystkich miejsc na wszystkie dostępne seanse. Dodatkowo przechowywana jest wartość zero-jedynkowa odpowiadająca za stan czy miejsce jest już zajęte;
- *Uprawnienia* - w projekcie przewidziane są uprawnienia dwojakiego rodzaju: uprawnienia administratora i użytkownika. Informacja ta ma kluczowe znaczenie podczas logowania do systemu i wyświetlania w nim dostępnych funkcjonalności;
- *Użytkownicy* - każdy użytkownik jest zobligowany podczas procesu rejestracji do podania takich informacji o sobie jak imię i nazwisko, oraz podania hasła i unikalnego loginu przez który będzie się logował i to właśnie te dane są przechowywane w tej tabeli. Dodatkowo również do każdego użytkownika dodawane są jego uprawnienia: administratora lub zwykłego użytkownika;
- *Bilety* - każdy bilet jest przypisany do konkretnego użytkownika i do konkretnego rekordu z tabeli *Rezerwacje*.

Rozdział 4

Implementacja baz danych w środowisku MySQL

By zaimplementować kilka rozproszonych węzłów bazy danych posłużono się programem docker, który określany jest jako narzędzie pozwalające umieścić program oraz jego zależności w lekkim, przenośnym, wirtualnym kontenerze, który można uruchomić na prawie każdym serwerze z systemem.

4.1 Konfiguracja kontenerów z środowiskiem MySQL 5.7 i phpMyAdmin

W celu instalacji programu *Docker* w systemie Ubuntu 16.04 posłużono się poniższymi komendami.

```
1 curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
2 sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
3 sudo apt-get update
4 apt-cache policy docker-ce
5 sudo apt-get install -y docker-ce
```

Listing 4.1 Instalacja programu docker w systemie ubuntu 16.04

Następnie zostały pobrane obrazy *MySQL 5.7* oraz *phpmyadmin*

```
1 docker pull mysql
2 docker pull phpmyadmin/phpmyadmin
```

Listing 4.2 Pobranie obrazów z repozytorium

Kolejnym krokiem było zapewnienie by można było uruchomić kilka identycznych kontenerów jednocześnie. W tym celu skorzystano z narzędzia Docker Compose. By je zainstalować użyto poniższe komendy.

```
1 sudo curl -L https://github.com/docker/compose/releases/download/1.18.0/docker-compose -'uname -s' -'uname -m' -o /usr/local/bin/docker-compose
2 sudo chmod +x /usr/local/bin/docker-compose
```

```
3 docker-compose --version
```

Listing 4.3 Instalacja narzędzia Docker Compose

W poniższym listingu zamieszczona została zawartość pliku `docker-compose.yml`. Narzędzie dzięki temu plikowi generuje pięć kontenerów z MySQL 5.7 oraz pięć instancji narzędzia `phpmyadmin`. Czyli w sumie dziesięć serwisów. Każdy został przekierowany na osobny port. Przez co z łatwością można operować tymi serwisami z poziomu przeglądarki internetowej poprzez `localhost`.

```
1 version: "2"
2 services:
3     db_master:
4         image: mysql
5         ports:
6             - "3307:3306"
7         environment:
8             MYSQL_USER: user
9             MYSQL_PASSWORD: test
10            MYSQL_ROOT_PASSWORD: master
11        volumes:
12            - master:/var/lib/mysql
13        networks:
14            - default
15    db_slave1:
16        image: mysql
17        ports:
18            - "3308:3306"
19        environment:
20            MYSQL_USER: user
21            MYSQL_PASSWORD: test
22            MYSQL_ROOT_PASSWORD: slave1
23        volumes:
24            - slave1:/var/lib/mysql
25        networks:
26            - default
27    db_slave2:
28        image: mysql
29        ports:
30            - "3309:3306"
31        environment:
32            MYSQL_USER: user
33            MYSQL_PASSWORD: test
34            MYSQL_ROOT_PASSWORD: slave2
35        volumes:
36            - slave2:/var/lib/mysql
37        networks:
38            - default
39    db_slave3:
40        image: mysql
41        ports:
42            - "3310:3306"
43        environment:
44            MYSQL_USER: user
45            MYSQL_PASSWORD: test
```



```
46         MYSQL_ROOT_PASSWORD: slave3
47     volumes:
48         - slave3:/var/lib/mysql
49     networks:
50         - default
51 db_slave4:
52     image: mysql
53     ports:
54         - "3311:3306"
55     environment:
56         MYSQL_USER: user
57         MYSQL_PASSWORD: test
58         MYSQL_ROOT_PASSWORD: slave4
59     volumes:
60         - slave4:/var/lib/mysql
61     networks:
62         - default
63 phpmyadminMaster:
64     image: phpmyadmin/phpmyadmin
65     links:
66         - db_master:db
67     ports:
68         - "8082:80"
69     environment:
70         MYSQL_USER: user
71         MYSQL_PASSWORD: test
72         MYSQL_ROOT_PASSWORD: master
73 phpmyadminSlave1:
74     image: phpmyadmin/phpmyadmin
75     links:
76         - db_slave1:db
77     ports:
78         - "8083:80"
79     environment:
80         MYSQL_USER: user
81         MYSQL_PASSWORD: test
82         MYSQL_ROOT_PASSWORD: slave1
83 phpmyadminSlave2:
84     image: phpmyadmin/phpmyadmin
85     links:
86         - db_slave2:db
87     ports:
88         - "8084:80"
89     environment:
90         MYSQL_USER: user
91         MYSQL_PASSWORD: test
92         MYSQL_ROOT_PASSWORD: slave2
93 phpmyadminSlave3:
94     image: phpmyadmin/phpmyadmin
95     links:
96         - db_slave3:db
97     ports:
98         - "8085:80"
```

```
99     environment:
100         MYSQL_USER: user
101         MYSQL_PASSWORD: test
102         MYSQL_ROOT_PASSWORD: slave3
103     phpmyadminSlave4:
104         image: phpmyadmin/phpmyadmin
105         links:
106             - db_slave4:db
107         ports:
108             - "8086:80"
109         environment:
110             MYSQL_USER: user
111             MYSQL_PASSWORD: test
112             MYSQL_ROOT_PASSWORD: slave4
113     volumes:
114         master:
115         slave1:
116         slave2:
117         slave3:
118         slave4:
```

Listing 4.4 Utworzony plik docker-compose.yml

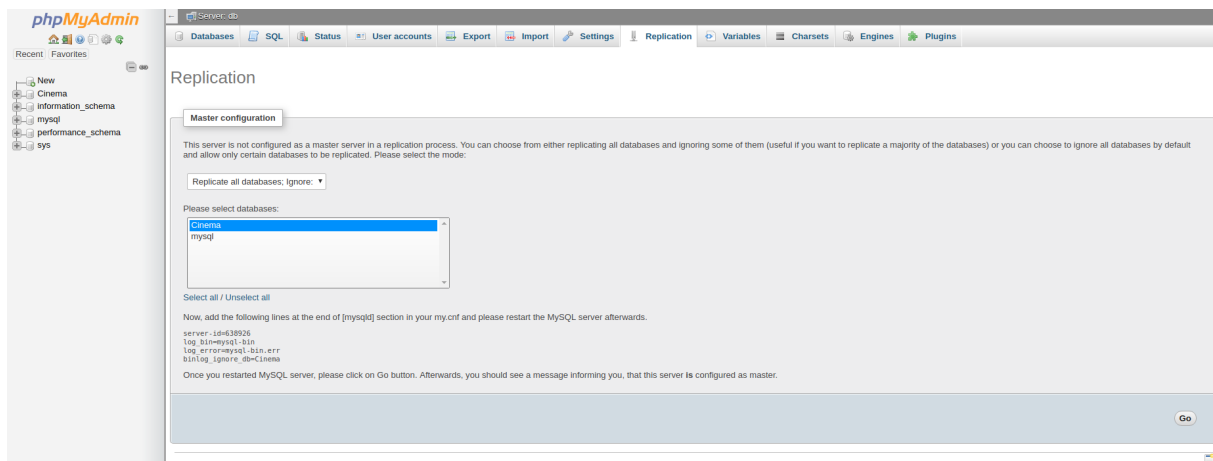
Generowanie serwisów z pliku odbywa się poprzez wydanie komendy *docker-compose up* znajdując się w katalogu z plikiem docker-compose.yml.

4.2 Konfiguracja mechanizmów replikacji master-slave

By móc skorzystać z replikacji master-slave każdy węzeł rozproszonej bazy danych musi utworzoną bazę. Dodatkowo trzeba dla każdego węzła zmodyfikować plik konfiguracyjny *my.cnf* serwisu z MySQL. W pliku tym należy nadać każdemu węzłowi unikalny parametr *server-id*. Dodatkowo w węźle master w zakładce *[mysqld]* należy dodać linie z listingu 4.5, a następnie w phpmyadmin wcisnąć przycisk "GO"

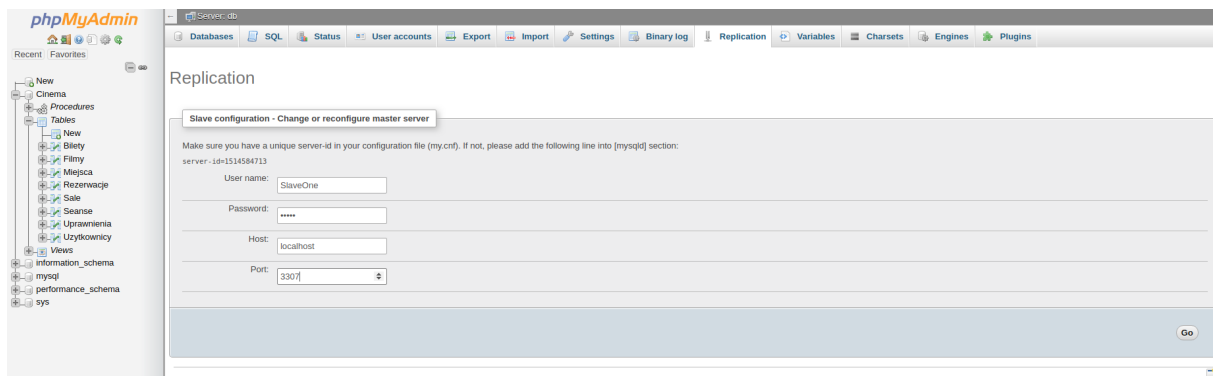
```
1 server-id=638926
2 log_bin=mysql-bin
3 log_error=mysql-bin.err
4 binlog_ignore_db=Cinema
```

Listing 4.5 Konfiguracja serwera master- modyfikacja pliku my.cnf



Rysunek 4.1 Konfiguracja serwera master w phpmyadmin

By ustawić serwer jako slave, należy podać login i hasło do konta w węźle master, podać adres hosta oraz port, a następnie wcisnąć przycisk "GO".



Rysunek 4.2 Konfiguracja serwera slave w phpmyadmin

By dodać opóźnienie do węzła slave można z poziomu panelu phpmyadmin w zakładce SQL wydać komendę:

```
1 CHANGE MASTER TO MASTER_DELAY = 120;
```

Listing 4.6 Konfiguracja opóźnienia dla węzła slave

Dla takich danych opóźnienie wyniesie 120 sekund.

4.3 Realizacja bazy danych

4.3.1 Tabele

Tabele generowano za pomocą skryptów MySQL.

- zdefiniowano klucze główne;
- zdefiniowano klucze obce;
- dodano ograniczenia jeżeli jakiś element nie może być pusty;

- tam gdzie to konieczne (login użytkownika, nazwa sali i nazwa uprawnień) dodano ograniczenia, aby dane te były unikalne w swoich tabelach;
- sprawdzanie czy podana liczba należy do przedziału wykorzystując słowo kluczowe *CHECK* (pojemność sali musi się znajdować w przedziale od 20 do 450 miejsc);
- zabezpieczenie przed nadpisywaniem już wcześniej utworzonych tabel *IF NOT EXISTS*.

```
CREATE TABLE IF NOT EXISTS Seanse
(
  Id int AUTO_INCREMENT primary key,
  IdFilmu int not null,
  IdSali int not null,
  DataSeansu date not null,
  Godzina time not null,
  Cena float not null,
  WolneMiejsca int not null,
  foreign key (IdFilmu) references Filmy(Id),
  foreign key (IdSali) references Sale(Id)
);
```

Rysunek 4.3 Generowanie tabeli seansów

```
CREATE TABLE IF NOT EXISTS Bilety
(
  Id int AUTO_INCREMENT primary key,
  IdKlienta int not null,
  IdRezerwacji int not null,
  foreign key (IdKlienta) references Uzytkownicy(Id),
  foreign key (IdRezerwacji) references Rezerwacje(Id)
);
```

Rysunek 4.4 Generowanie tabeli biletów - przykład użycia klucza głównego, kluczy obcych i ograniczenia "not null"

```
CREATE TABLE IF NOT EXISTS Sale
(
  Id int AUTO_INCREMENT primary key,
  NazwaSali varchar(30) unique not null,
  Pojemnosc int not null
  CHECK (pojemnosc BETWEEN 20 AND 450)
);
```

Rysunek 4.5 Generowanie tabeli sal - przykład użycia CHECK

4.3.2 Widoki

Utworzono trzy widoki:

Miejsca		_Seanse_		_Bilety_	
Id	int(11)	Id	int(11)	Id	int(11)
Tytul	varchar(30)	IdFilmu	int(11)	IdKlienta	int(11)
Data	date	Tytul	varchar(30)	Tytul	varchar(30)
Godzina	time	Rezyser	varchar(30)	Data	date
Cena	float	Rok	date	Godzina	time
NazwaSali	varchar(30)	Gatunek	varchar(30)	Cena	float
Rzad	int(11)	Dlugosc	varchar(30)	NazwaSali	varchar(30)
NrMiejsca	int(11)	IdSali	int(11)	Rzad	int(11)
Zajete	tinyint(1)	NazwaSali	varchar(30)	NrMiejsca	int(11)
		Data	date		
		Godzina	time		
		Cena	float		
		Miejsca	int(11)		

Rysunek 4.6 Wygenerowane widoki

- **_Seanse_** - rozszerza tabelę *Seanse* o dodatkowe informacje o filmie, którego dotyczy seans i sali, na której seans zostanie wyświetlony;
- **_Miejsca_** - rozszerza tabelę *Rezerwacje* o informacje takie jak nazwa sali, rząd i numer miejsca.
- **_Bilety_** - rozszerza tabelę *Bilety* o informacje takie jak tytuł filmu, data, cena, nazwa sali, rząd oraz numer miejsca

```

CREATE OR REPLACE VIEW _Seanse_ AS
SELECT Seanse.Id, Filmy.Id AS IdFilmu, Filmy.Tytul,
Filmy.Rezyser, Filmy.Rok, Filmy.Gatunek, Filmy.Dlugosc,
Sale.Id AS IdSali, Sale.NazwaSali, Seanse.DataSeansu,
Seanse.Godzina, Seanse.Cena, Seanse.WolneMiejsca
FROM Seanse
LEFT JOIN Filmy ON Filmy.Id=IdFilmu
LEFT JOIN Sale ON Sale.Id=IdSali;

```

Rysunek 4.7 Generowanie widoku seansów

```









































































CREATE OR REPLACE VIEW _Miejsca_ AS
SELECT Rezerwacje.Id, Rezerwacje.IdSeansu,
Sale.NazwaSali, Miejsca.Rzad, Miejsca.NrMiejsca,
Rezerwacje.Zajete FROM Rezerwacje
LEFT JOIN Miejsca ON Miejsca.Id=Rezerwacje.IdMiejsca
LEFT JOIN Sale ON Sale.Id=IdSali;

```

Rysunek 4.8 Generowanie widoku Miejsc

4.3.3 Procedury

Rysunek 4.9 przedstawia wszystkie procedury jakie zostały zaimplementowane. Natomiast na rysunku 4.10 pokazana została jedna przykładowa procedura, która dodając seans dodaje również pulę miejsc równą pojemności sali, w której odbędzie się seans do tabeli *Rezerwacje*.

Nazwa	Działanie				Typ
BiletyUsera	 Edytuj	 Wykonaj	 Eksport	 Usuń	PROCEDURE
DodajAdmina	 Edytuj	 Wykonaj	 Eksport	 Usuń	PROCEDURE
DodajFilm	 Edytuj	 Wykonaj	 Eksport	 Usuń	PROCEDURE
DodajSale	 Edytuj	 Wykonaj	 Eksport	 Usuń	PROCEDURE
DodajSeans	 Edytuj	 Wykonaj	 Eksport	 Usuń	PROCEDURE
DodajUzytkownika	 Edytuj	 Wykonaj	 Eksport	 Usuń	PROCEDURE
EdytujFilm	 Edytuj	 Wykonaj	 Eksport	 Usuń	PROCEDURE
EdytujSale	 Edytuj	 Wykonaj	 Eksport	 Usuń	PROCEDURE
EdytujSeans	 Edytuj	 Wykonaj	 Eksport	 Usuń	PROCEDURE
KupBilet	 Edytuj	 Wykonaj	 Eksport	 Usuń	PROCEDURE
SprawdzUzytkownika	 Edytuj	 Wykonaj	 Eksport	 Usuń	PROCEDURE
UsunFilm	 Edytuj	 Wykonaj	 Eksport	 Usuń	PROCEDURE
UsunRezerwacje	 Edytuj	 Wykonaj	 Eksport	 Usuń	PROCEDURE
UsunSale	 Edytuj	 Wykonaj	 Eksport	 Usuń	PROCEDURE
UsunSeans	 Edytuj	 Wykonaj	 Eksport	 Usuń	PROCEDURE
WyswietlBilet	 Edytuj	 Wykonaj	 Eksport	 Usuń	PROCEDURE
WyswietlMiejsca	 Edytuj	 Wykonaj	 Eksport	 Usuń	PROCEDURE
WyswietlSeanse	 Edytuj	 Wykonaj	 Eksport	 Usuń	PROCEDURE

Rysunek 4.9 Przykładowa procedura

Opisy procedur:

- BiletyUsera - wyświetla wszystkie bilety które zostały zakupione przez konkretnego użytkownika;
- DodajAdmina - procedura pozwala na dodanie nowego użytkownika o uprawnieniach administratora;
- DodajFilm - pozwala na dodanie nowego filmu do filmoteki kina;
- DodajSale - pozwala dodać do bazy danych nową salę kinową;
- DodajSeans - umożliwia dodanie nowego seansu do oferty kina;
- DodajUzytkownika - procedura pozwala na dodanie nowego użytkownika o standardowych prawach zwykłego użytkownika;
- EdytujFilm - pozwala na zmniejszenie wszystkich informacji o filmie dostępnych w bazie;
- EdytujSale - pozwala na zmianę nazwy sali;
- EdytujSeans - pozwala modyfikować datę godzinę i cenę seansu;

- KupBilet - tworzy nowy rekord w tabeli Bilety, przypisuje go do konkretnego użytkownika, oraz zmienia ilość wolnych miejsc na seansie, a w tabeli Rezerwacje oznacza miejsce jako zajęte;
- SprawdzUzytkownika - sprawdza czy użytkownik o podanym loginie i hasle istnieje w systemie;
- UsunFilm - usuwa film z filмотeki kina;
- UsunRezerwacje - usuwa wszystkie bilety, anuluje transakcję użytkownika, przywraca miejsce jako niezarezerwowane;
- UsunSale - usuwa sale i miejsca przypisane do sali, anuluje wszystkie seanse, które miały odbyć się na danej sali, i anuluje wszystkie bilety na te seanse;
- UsunSeans - anuluje seans i wszystkie bilety i rezerwacje na niego;
- WyszwietlBilet - wyświetla informacje o bilecie o podanym id;
- WyszwietlMiejsca - pokazuje wszystkie wolne miejsca na konkretny seans, korzystając z widoku _Miejsca_;
- WyszwietlSeanse - wyświetla rozszerzone informacje o seansach z podanym filmem, korzystając z widoku _Seanse_.

```
-- Dodaje seans i rezerwacje zaleznie od ilosci miejsc na sali
delimiter //
DROP PROCEDURE IF EXISTS DodajSeans//
CREATE PROCEDURE DodajSeans(in _idFilmu int, in _idSali int, in _dataSeansu date, in _godzina time, in _cena float )
BEGIN
    SET @id=_idSali;
    SELECT @miejsca:=Pojemnosc FROM Sale WHERE Id=@id;

    INSERT INTO Seanse(IdFilmu,IdSali, DataSeansu, Godzina, Cena, WolneMiejsca) VALUES
    (_idFilmu,@id,_dataSeansu,_godzina,_cena,@miejsca);

    SET @idSeansu = LAST_INSERT_ID();
    SET @i=1;

    WHILE (@i<=@miejsca) DO
        INSERT INTO Rezerwacje(IdMiejsca,IdSeansu, Zajete) VALUES (@i,@idSeansu,'0');
        SET @i=@i+1;
    END WHILE;

END
//
```

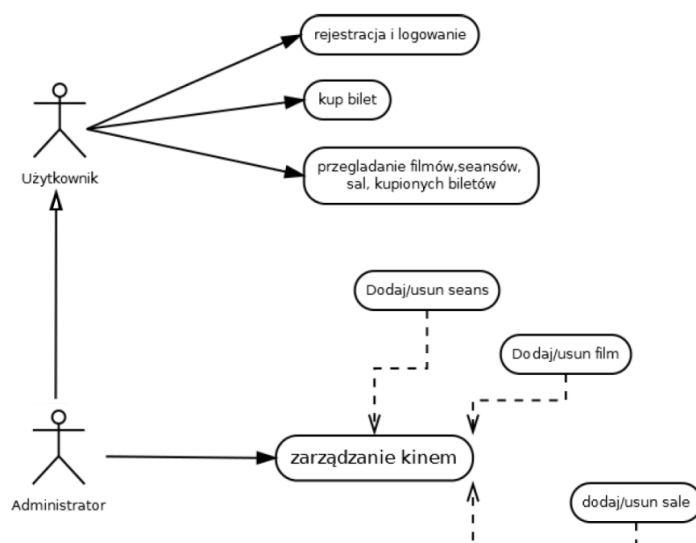
Rysunek 4.10 Przykładowa procedura

Rozdział 5

Projekt i implementacja aplikacji klienckiej oraz REST API

5.1 Funkcje aplikacji

5.1.1 Diagram przypadków użycia



Rysunek 5.1 Diagram przypadków użycia

Użytkownik:

- tworzenie nowego konta (podanie loginu, hasła itp.);
- logowanie;
- przeglądanie filmów, seansów, kupionych biletów;
- kupowanie biletów.

Administrator:

- te same funkcjonalności co użytkownik;

- dodawanie/usuwanie/edytowanie seansów;
- dodawanie/usuwanie/edytowanie filmów;
- dodawanie/usuwanie/edytowanie dostępnych sal.

5.1.2 Scenariusz wybranych przypadków użycia

W poniższych tabelach umieszczone zostały scenariusze wybranych przypadków użycia.

Przypadek użycia	Rejestracja
Opis	Funkcja umożliwiająca założenie konta w systemie
Warunki początkowe	Brak konta
Warunki końcowe	System utworzył konto lub odrzucił żądanie
Przebieg podstawowy	1. Użytkownik podaje swoje dane (login, hasło itp.) 2. System sprawdza czy login nie jest zajęty 3. Jeśli login jest wolny, skok do pkt 6. 4. Wyświetlenie komunikatu o błędach 5. Skok do pkt 1. 6. Zakończenie rejestracji

Tabela 5.1 Scenariusz rejestracji

Przypadek użycia	Logowanie
Opis	Funkcja umożliwiająca użytkownikom uzyskanie dostępu do systemu
Warunki początkowe	Posiadanie własnego konta
Warunki końcowe	System autoryzował, bądź odrzucił użytkownika
Przebieg podstawowy	1. Użytkownik podaje swój login i hasło 2. System wyszukuje użytkownika 3. Jeśli użytkownik zostanie zweryfikowany, skok do pkt 6. 4. Wyświetlenie komunikatu o błędach 5. Skok do pkt 1. 6. Zakończenie logowania

Tabela 5.2 Scenariusz logowania

Przypadek użycia	zarządzanie kinem
Opis	Funkcja umożliwiająca pracownikom zarządzanie kinem
Warunki początkowe	Funkcja dostępna tylko dla zalogowanych pracowników
Warunki końcowe	Dokonanie zmiany w systemie
Przebieg podstawowy	<ol style="list-style-type: none"> 1. Pracownik uruchamia aplikację zarządzania kinem 2. Wywołanie odpowiedniego przypadku w zależności od wybranej akcji do wykonania 3. Jeżeli wybrano "Dodanie filmu", wywołanie przypadku "Dodanie filmu". Skok do pkt 8. 4. Jeżeli wybrano "Dodanie seansu", wywołanie przypadku "Dodanie seansu". Skok do pkt 8. 5. Jeżeli wybrano "Usuwanie filmu", wywołanie przypadku "Usuwanie filmu". Skok do pkt 8. 6. Jeżeli wybrano "Usuwanie seansu", wywołanie przypadku "Usuwanie seansu". Skok do pkt 8. 7. Jeżeli wybrano "Dodawanie sali", wywołanie przypadku "Dodanie sali". Skok do pkt 8. 8. Zakończenie procesu
Przebieg alternatywny	<ol style="list-style-type: none"> 1. Pracownik wchodzi na stronę zarządzania kinem. 2. Wywołanie odpowiedniego przypadku w zależności od wybranej akcji do wykonania 3. Pracownik wylogował się z systemu 4. zakończenie procesu.

Tabela 5.3 Scenariusz zarządzania kinem

5.2 Realizacja wybranych funkcjonalności aplikacji

Aplikacja internetowa powstała z wykorzystaniem platformy programistycznej Angular 2. Jest to framework odpowiedzialny za tworzenia aplikacji po stronie klienta. Do zaprogramowania części funkcjonalnej aplikacji wykorzystuje on TypeScript czyli wolny i otwartoźródłowy język programowania stworzony przez firmę Microsoft jako nadzbiór języka JavaScript. Umożliwia on opcjonalne statyczne typowanie oraz programowanie zorientowane obiektowo oparte na klasach. TypeScript jest nadzbiorem JavaScript, a więc potencjalnie każdy program napisany w języku JavaScript jest poprawnym programem TypeScript. Aplikacje napisane w TypeScript kompilują się bezpośrednio do języka JavaScript. Poniżej znajdują się przykładowe fragmenty kodu napisane w tym języku odpowiedzialne za komunikację z API bazy danych.

```

1 private getData(str: string, id?: string) {
2     var url = this.apiUrl + str + '/' + (id ? id : "");
3     return this._http.get(url)
4         .map(res => res.json());
5 }
6
7 getTickets(id?: string) {
8     return this.getData('bilety', id);
9 }
10
11 getSeat(id?: string) {
12     return this.getData('miejsca', id);

```

```
13 }
14
15 getFilm(id?: string) {
16     return this.getData('filmy', id);
17 }
18
19 getReservation(id?: string) {
20     return this.getData('rezerwacje', id);
21 }
22
23 getRoom(id?: string) {
24     return this.getData('sale', id);
25 }
26
27 getSeance(id?: string) {
28     return this.getData('seanse', id);
29 }
30
31 getTicket(id: string) {
32     return this.getData('bilet', id);
33 }
34
35 getFilmSeance(filmId: string) {
36     return this.getData('seansefilmu', filmId);
37 }
38
39 getFreeSeats(seanceId: string) {
40     return this.getData('wolnemiejsca', seanceId);
41 }
42
43 getUsersTickets(id: string) {
44     return this.getData('biletyusera', id);
45 }
```

Listing 5.1 Implementacja funkcji GET po stronie aplikacji

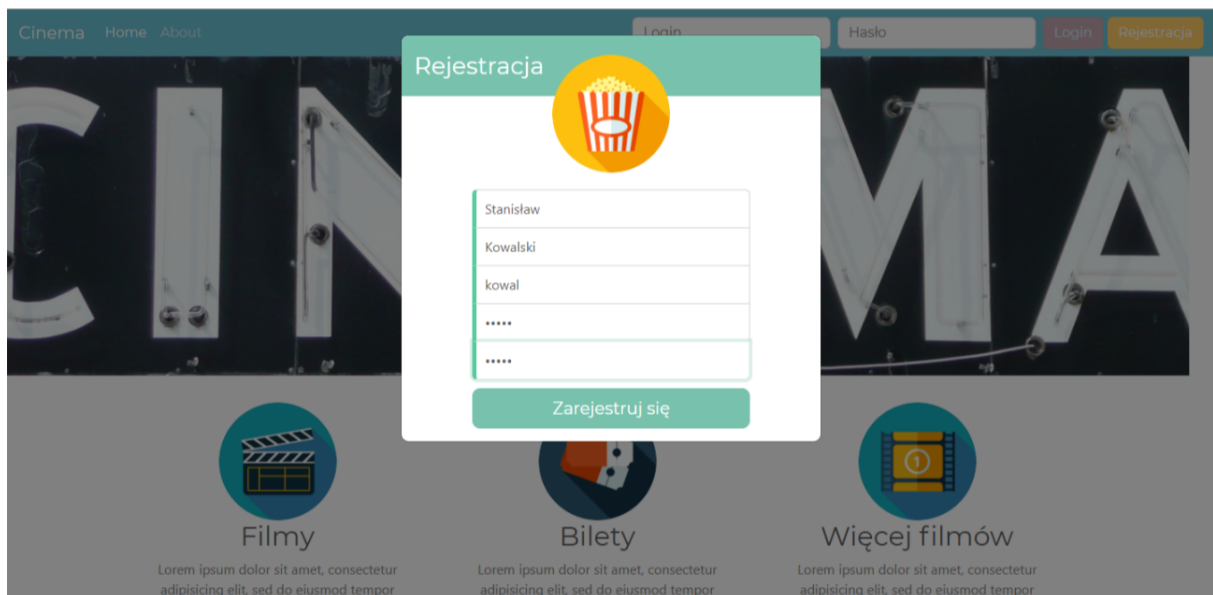
```
1 private post(str: string, data: any) {
2     var url = this.apiUrl + str;
3     return this._http.post(url, JSON.stringify(data), { headers: this.
4         headers })
5         .map((res: Response) => res.json())
6 }
7
8 addFilm(film: Film) {
9     return this.post('film', film);
10 }
11
12 addSeance(seance: Seance) {
13     return this.post('seans', seance);
14 }
15
16 addRoom(room: Room) {
17     return this.post('sala', room);
18 }
```

```
18  
19 addUser(user: User) {  
20     return this.post('uzytkownik', user);  
21 }  
22  
23 addTicket(ticket: Ticket) {  
24     return this.post('bilet', ticket);  
25 }
```

Listing 5.2 Implementacja funkcji POST po stronie aplikacji

5.2.1 Interfejs aplikacji

Aplikacja została napisana tak aby blokować widok niedozwolonych operacji jeśli użytkownik nie ma odpowiednich uprawnień. Poniżej znajdują się zrzuty ekranu widoku interfejsu z konta administracyjnego.



Rysunek 5.2 Rejestracja użytkownika

Cinema Home About Radosław Taborski

Moje bilety
Kup bilet
Filmy
Seanse
Sale

Kup bilet

✓ Zarezerwowano

Film
Skazany

Tytuł	Długość	Data seansu	Godzina	Sala	Wolne miejsca	Cena	Wybierz
Skazany	121 min	2018-10-10	10:18:00	VIP	59	25 zł	⊕
Skazany	121 min	2017-12-13	16:59:00	A2	216	10 zł	⊙
Skazany	121 min	2012-12-20	01:00:12	VIP	64	34 zł	⊙

Miejsce
Rząd: 1 Miejsce: 13

Zarezerwuj miejsce

Rysunek 5.3 Rezerwacja biletów

Cinema Home About Radosław Taborski

Moje bilety
Kup bilet
Filmy
Seanse
Sale

Moje Bilety

Tytuł	Data	Godzina	Sala	Miejsce	Rząd	Cena
Vikings	2017-12-09	18:30:00	A1	12	1	20
Skazany	2018-10-10	10:18:00	A1	56	4	25
Skazany	2018-10-10	10:18:00	A1	8	1	25
Skazany	2018-10-10	10:18:00	A1	1	1	25
Skazany	2018-10-10	10:18:00	A1	2	1	25
Skazany	2018-10-10	10:18:00	A1	4	1	25
Avatar	2017-10-10	18:18:00	A1	2	1	18
Avatar	2017-10-10	18:18:00	A1	1	1	18

Rysunek 5.4 Widok biletów zarezerwowanych przez użytkownika

Cinema Home About Radosław Taborski

Moje bilety
Kup bilet
Filmy
Seanse
Sale

Filmy

Tytuł	Reżyser	Rok	Gatunek	Długość	Edytuj	Usun
Listy do M. 3	Tomasz Konecki	2017-11-10	komedie romantyczna	121 min		
Najlepszy	Łukasz Palkowski	2017-11-17	dramat	112 min		
Vikings	Ciaran Donnelly	2013-10-12	dramat historyczny	45 min		
Avatar	James Cammeron	2009-09-25	fantasy	130 min		
Gwiezdne wojny: Ostatni Jedi	Rian Johnson	2017-12-14	fantasy	112 min		
Skazany	Ric Roman Waugh	2017-10-27	Dramat, Thriller	121 min		

Rysunek 5.5 Widok filmów dostępnych w kinie



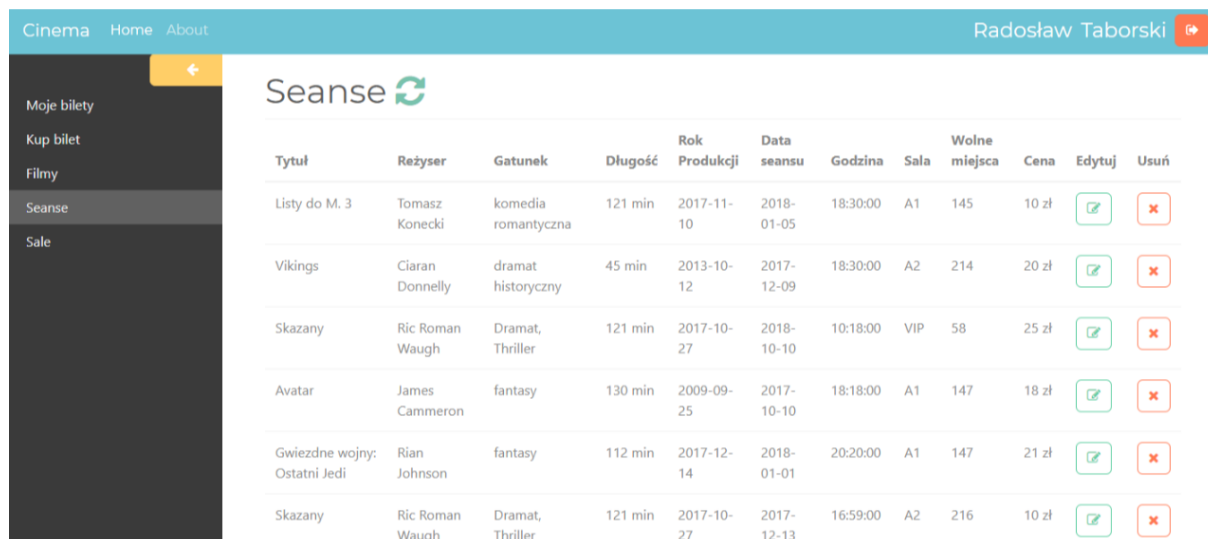
The form is titled "Dodaj film" and contains several input fields for film details. Each field has a green highlight on the left side of the input area. The fields are: "Tytuł" (Title) with the value "Tytuł", "Reżyser" (Director) with the value "Kowalski Jan", "Rok produkcji" (Production year) with the value "21.12.2017", "Gatunek" (Genre) with the value "Akcja", and "Długość (min)" (Duration) with the value "120". At the bottom of the form is a green button labeled "Dodaj film".

Rysunek 5.6 Dodawanie nowego filmu



The form is titled "Edytuj Listy do M. 3" and is overlaid on a background showing a list of films. The form contains input fields for: "Tytuł" (Title) with the value "Listy do M. 3", "Reżyser" (Director) with the value "Tomasz Konecki", "Rok produkcji" (Production year) with the value "10.11.2017", "Gatunek" (Genre) with the value "komedia romantyczna", and "Długość (min)" (Duration) with the value "121". At the bottom of the form are two buttons: "Zapisz zmiany" (Save changes) in green and "Anuluj" (Cancel) in orange.

Rysunek 5.7 Edycja filmu



Tytuł	Reżyser	Gatunek	Długość	Rok Produkcji	Data seansu	Godzina	Sala	Wolne miejsca	Cena	Edytuj	Usuń
Listy do M. 3	Tomasz Konecki	komedie romantyczna	121 min	2017-11-10	2018-01-05	18:30:00	A1	145	10 zł		
Vikings	Ciaran Donnelly	dramat historyczny	45 min	2013-10-12	2017-12-09	18:30:00	A2	214	20 zł		
Skazany	Ric Roman Waugh	Dramat, Thriller	121 min	2017-10-27	2018-10-10	10:18:00	VIP	58	25 zł		
Avatar	James Cammeron	fantasy	130 min	2009-09-25	2017-10-10	18:18:00	A1	147	18 zł		
Gwiezdne wojny: Ostatni Jedi	Rian Johnson	fantasy	112 min	2017-12-14	2018-01-01	20:20:00	A1	147	21 zł		
Skazany	Ric Roman Waugh	Dramat, Thriller	121 min	2017-10-27	2017-12-13	16:59:00	A2	216	10 zł		

Rysunek 5.8 Widok seansów dostępnych w kinie



Dodaj seans

Film
Vikings

Sala
A2

Data
09.12.2017

Godzina
12:21

Cena (zł)
12

Dodaj film

Rysunek 5.9 Dodawanie seansu

The screenshot shows a modal window titled "Edytuj Listy do M. 3" overlaid on a blurred background of the Seanse application. The modal contains three input fields for editing a show listing:

- Data:** A text input field containing "05.01.2018".
- Godzina:** A text input field containing "18:30".
- Cena (zł):** A text input field containing "10".

At the bottom of the modal are two buttons: "Zapisz zmiany" (Save changes) in green and "Anuluj" (Cancel) in orange.

The background shows a table with columns: Tytuł, Reżyser, Godzina, Sala, and Wolne miejsca. Visible rows include "Listy do M. 3", "Vikings", "Skazany", and "Avatar".

Rysunek 5.10 Edycja seansu

The screenshot shows the "Sale" page of the Cinema application. The page has a header with "Cinema", "Home", and "About" links, and a user profile "Radosław Taborski" with a logout icon. A left sidebar contains navigation links: "Moje bilety", "Kup bilet", "Filmy", "Seanse", and "Sale" (which is highlighted).

The main content area is titled "Sale" and features a table of available halls:

Id	Nazwa sali	Pojemność	Usuń
1	A1	150	
2	A2	216	
4	VIP	64	

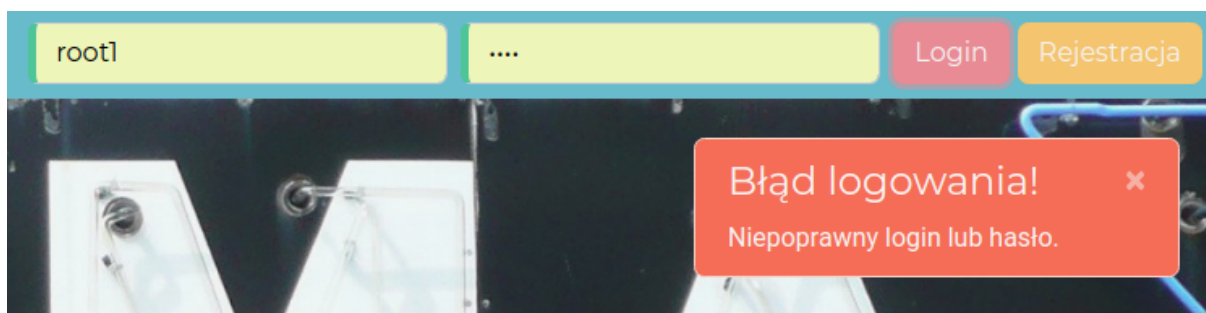
To the right of the table is a "Dodaj salę" (Add hall) form with the following fields:

- Nazwa sali (Hall name)
- Ilość miejsc w rzędzie (Number of seats in the row)
- Ilość rzędów (Number of rows)

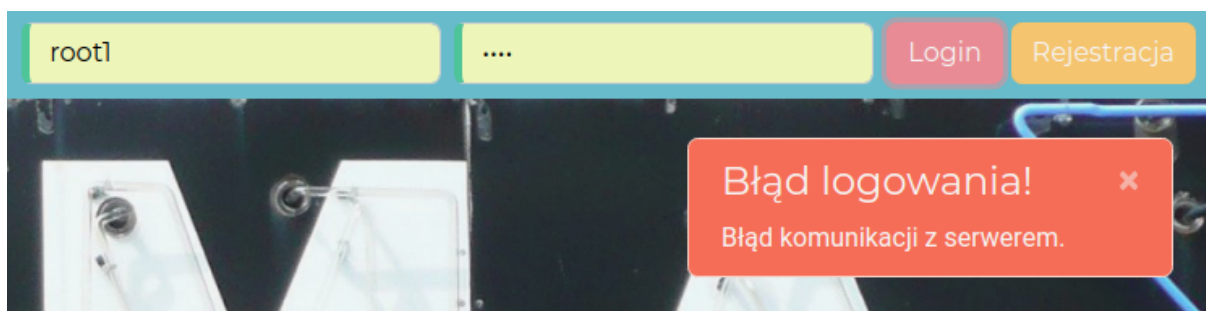
A green "Dodaj salę" button is at the bottom of the form.

Rysunek 5.11 Widok sal dostępnych w kinie

Aplikacja również informuje o błędach z łącznością z serwerem (rysunek 5.13) oraz o podaniu błędnego loginu lub hasła (rysunek 5.12)



Rysunek 5.12 Podanie błędnego loginu i hasła



Rysunek 5.13 Błąd komunikacji z serwerem

5.3 Realizacja REST API

REST (ang. Representational State Transfer) jest wzorcem narzucającym dobre praktyki tworzenia architektury aplikacji rozproszonych. RESTful Webservices (inaczej RESTful web API) jest usługą sieciową zaimplementowaną na bazie protokołu HTTP i głównych zasad wzorca REST. Ważnym założeniem REST jest istnienie zasobów (ang. resources) jako źródeł danych a także żądana akcja. API aplikacji zostało stworzone w postaci skryptu w PHP.

zasób	metoda	zastosowanie
http://90.156.81.49:81/cinemaapi.php/filmy	GET	zostanie zwrócona lista filmów
http://90.156.81.49:81/cinemaapi.php/filmy/1	GET	zostanie zwrócony film o id równym 1
http://90.156.81.49:81/cinemaapi.php/film	PUT	dodanie nowego filmu
http://90.156.81.49:81/cinemaapi.php/film/1	POST	edycja filmu o id równym 1
http://90.156.81.49:81/cinemaapi.php/film/1	DELETE	usunięcie filmu o id równym 1

Tabela 5.4 Przykładowe zastosowanie REST API

Poniżej znajdują się fragmenty skryptu odpowiedzialnego za obsługę zapytań.

```

1 <?php
2 header("Access-Control-Allow-Origin: *");
3 header("Content-Type: application/json");
4 header("Access-Control-Allow-Headers: Content-Type, Authorization, X
  -Requested-With");
5 header("Access-Control-Allow-Methods: GET, POST, PUT, OPTIONS,
  DELETE");
6

```

```
7 require_once("./CinemaApi/get.php");
8 require_once("./CinemaApi/delete.php");
9 require_once("./CinemaApi/put.php");
10 require_once("./CinemaApi/post.php");
11 require_once("./CinemaApi/databases.php");
12
13 // get the HTTP method, path and body of the request
14 $method = $_SERVER['REQUEST_METHOD'];
15 $request = explode('/', trim($_SERVER['PATH_INFO'], '/'));
16 $input = json_decode(file_get_contents('php://input'));
17 $masterAdress = 'localhost:3307';
18
19 // in case of SQL injection
20 function clear($data){
21     return preg_replace('/[^\a-z0-9_+\/i', '', $data);
22 }
23
24 // retrieve the table and key from the path
25 $table = clear(array_shift($request));
26 $key = clear(array_shift($request)+0);
27
28 if($table == ""){
29     header("Content-Type: text/html; charset=utf-8");
30     readfile('./CinemaApi/api.html' );
31     die();
32 }
33
34 // connect to the mysql database
35 if($method == 'GET'){
36     $i = rand(0, count($databases) -1);
37     // die($i . " " . var_dump($databases[$i]));
38     $link = mysqli_connect($databases[$i]->adress, $databases[$i]->
        login, $databases[$i]->password, 'Cinema')
39     or die('Error connecting to MySQL server.');
```

```
40 } else{
41     $link = mysqli_connect($masterAdress, 'root', 'master', 'Cinema')
42     or die('Error connectiong to MySQL server');
```

```
43 }
44 mysqli_set_charset($link, 'utf8');
```

```
45
46 // create SQL based on HTTP method
47 switch ($method) {
48     case 'GET':
49         $sql = getRequest($table, $key); break;
50     case 'DELETE':
51         $sql = deleteRequest($table, $key); break;
52     case 'PUT':
53         $sql = putRequest($table, $input); break;
54     case 'POST':
55         $sql = postRequest($table, $key, $input); break;
56 }
57
58 $result = mysqli_query($link, $sql);
```

```
59
60 // die if SQL statement failed
61 if (!$result) {
62     http_response_code(404);
63     die(mysqli_error());
64 }
65
66 // print results, insert id or affected row count
67 if ($method == 'GET' || $table == 'login') {
68     echo '[';
69     for ($i=0;$i<mysqli_num_rows($result);$i++) {
70         echo ($i>0?',': '').json_encode(mysqli_fetch_object($result));
71     }
72     echo ']';
73 } elseif ($method == 'POST') {
74     echo mysqli_insert_id($link);
75 } else {
76     echo mysqli_affected_rows($link);
77 }
78
79 // close mysql connection
80 mysqli_close($link);
81 ?>
```

Listing 5.3 Skrypt PHP API aplikacji

```
1 <?php
2
3 function getRequest($table, $key){
4
5     $sql = "";
6
7     switch ($table) {
8         case 'bilety': $sql ="select * from _Bilety_".($key?" WHERE id=
9             $key":''); break;
10        case 'filmy': $sql ="select * from Filmy".($key?" WHERE id=$key
11            ':''); break;
12        case 'miejsca': $sql ="select * from Miejsca".($key?" WHERE id=
13            $key":''); break;
14        case 'rezerwacje': $sql ="select * from Rezerwacje".($key?"
15            WHERE id=$key":''); break;
16        case 'sale': $sql ="select * from Sale".($key?" WHERE id=$key
17            ':''); break;
18        case 'seanse': $sql ="select * from _Seanse_".($key?" WHERE id=
19            $key":''); break;
20        case 'bilet': $sql ="CALL WyświetlBilet('$key')"; break;
21        case 'seansefilmu': $sql ="CALL WyświetlSeanse('$key')"; break;
22        case 'wolnemiejsca': $sql ="CALL WyświetlMiejsca('$key')"; break;
23        ;
24        case 'biletyusera': $sql ="CALL BiletyUsera('$key')"; break;
25    }
26
27    if($sql == ""){
28        http_response_code(404);
29    }
30 }
```

```
22     die("no method found");
23 }
24 // echo $sql;
25 return $sql;
26 }
27
28 ?>
```

Listing 5.4 Skrypt PHP obsługujący zapytania GET

Rozdział 6

Wdrożenie i testowanie aplikacji klienckiej

6.1 Wdrożenie

Aplikacja kliencka została stworzona jako aplikacja webowa z wykorzystaniem platformy programistycznej *Angular 2*. By móc skorzystać z tego frameworku należy odpowiednio skonfigurować środowisko.

W pierwszej kolejności wymagane jest zainstalowanie środowiska uruchomieniowego Node.js, które zaprojektowane zostało do tworzenia wysoce skalowalnych aplikacji internetowych, szczególnie pod kątem serwisów www napisanych w języku JavaScript. Środowisko to jest dostępne do pobrania ze strony internetowej node.js [6]. Bądź też jak to zostało wykonane w ramach projektu na systemie operacyjnym Ubuntu 16.04 poprzez komendy na listingu 6.1.

```
1 curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
2 sudo apt-get install -y nodejs
```

Listing 6.1 Instalacja Node.js w systemie operacyjnym Ubuntu 16.04

Kolejnym krokiem jest instalacja Angular-CLI poprzez komendy z listingu 6.2

```
1 sudo npm install -g @angular/cli
2 npm install --save express npm install --save express body-
  parserbody-parser
```

Listing 6.2 Instalacja Angular-CLI w systemie operacyjnym Ubuntu 16.04

Tak przygotowane środowisko umożliwia już uruchomienie aplikacji klienckiej. By uruchomić aplikację na *localhost:4200* w przeglądarce internetowej wystarczy będąc w katalogu z aplikacją wydać polecenia z listingu 6.3. W tym przypadku polecenie pierwsze odpowiedzialne jest za przebudowanie projektu i wymagane jest jedynie w przypadku dodania do projektów nowych pakietów.

```
1 ng build
2 ng serve
```

Listing 6.3 Uruchomienie aplikacji na localhost:4200 16.04

6.1.1 Zmiana domyślnego portu

Zmiana portu umożliwia uruchomienie projektu na serwerze, tak by mógł być widoczny bądź w sieci lokalnej bądź globalnie.

W tym celu w głównym katalogu aplikacji umieszczony został plik *server.js* oraz katalog *server*, w którym z kolei umieszczony został plik *api.js*

Na listingu 6.4 przedstawiona została zawartość pliku *server.js*. Numer portu na jaki został przekierowany serwis ustawiony został w 29 linii pliku.

```
1 // Get dependencies
2 const express = require('express');
3 const path = require('path');
4 const http = require('http');
5 const bodyParser = require('body-parser');
6
7 // Get our API routes
8 const api = require('./server/api');
9 const app = express();
10
11 // Parsers for POST data
12 app.use(bodyParser.json());
13 app.use(bodyParser.urlencoded({ extended: false }));
14
15 // Point static path to dist
16 app.use(express.static(path.join(__dirname, 'dist')));
17
18 // Set our api routes
19 app.use('/api', api);
20
21 // Catch all other routes and return the index file
22 app.get('*', (req, res) => {
23   res.sendFile(path.join(__dirname, 'dist/index.html'));
24 });
25
26 /**
27  * Get port from environment and store in Express.
28  */
29 const port = process.env.PORT || '501';
30 app.set('port', port);
31
32 /**
33  * Create HTTP server.
34  */
35 const server = http.createServer(app);
36
37 /**
38  * Listen on provided port, on all network interfaces.
39  */
40 server.listen(port, () => console.log(`API running on localhost:${port}`));
```

Listing 6.4 Zawartość pliku *server.js*

```
1 const express = require('express');
2 const router = express.Router();
3
4 /* GET api listing. */
5 router.get('/', (req, res) => {
```

```
6   res.send('api works');  
7 });  
8  
9 module.exports = router;
```

Listing 6.5 Zawartość pliku api.js

Po utworzeniu tych plików uruchomienie aplikacji klienckiej na nowym porcie odbywa się poprzez komendę z listingu 6.6.

```
1 | sudo node server.js
```

Listing 6.6 Uruchomienie aplikacji klienckiej na własnym porcie

6.1.2 Wdrażanie API REST-owego

Do poprawnego działania aplikacji potrzebne jest również API REST-owe pośredniczące w komunikacji aplikacji klienckiej z rozproszoną bazą danych. Podczas wdrażania projektu na systemie operacyjnym Ubuntu 16.04, API zostało umieszczone na serwerze HTTP- *APACHE2*.

6.2 Testy działania

W tej części zostały zamieszczone obrazy działania wprowadzonej aplikacji. Aplikacja działa poprawnie i zgodnie z przewidywaniami.

Tak jak to już zostało wspomniane w podrozdziale 2.3, by replikacja mogła przebiegać prawidłowo, bazy na węzłach *slave* muszą być spójne z bazą na węźle *master*, a co z tego poniekąd wynika bazy na węzłach *slave* muszą być utworzone i muszą posiadać taką samą strukturę tabel.

6.2.1 Dodawanie rekordu do rozproszonej bazy danych

Na rysunkach od 6.1 do 6.4 zademonstrowane zostało działanie replikacji na przykładzie dodawania filmu do bazy z poziomu aplikacji webowej.

Na rysunku 6.1 wpisane zostały dane przykładowego filmu, a po naciśnięciu przycisku 'Dodaj film' aplikacja poprzez API REST łączy się z węzłem master rozproszonej bazy danych, gdyż wprowadzana będzie modyfikacja, a następnie dodaje dane do tabeli Filmy (rysunek 6.2) dzięki procedurze SQL 'DodajFilm'. Wykonana procedura zostaje replikowana do wszystkich węzłów Slave. Tabela Filmy dla wybranego węzła Slave na porcie 8083 została pokazana na rysunku 6.3. Z węzła Slave również odczytywane są dane poprzez aplikację kliencką, a poprawnie odczytane dane z takiego węzła zostały przedstawione na rysunku 6.4.

Cinema Home About Radosław Taborski

Magiczna zima Muminków	Ira Carpelan	2017-12-22	Familijny	83 min		
Gotowi na wszystko	Michał Rogalski	2018-01-05	Komedia	117 min		
Pitch Perfect 3	Kay Cannon	2017-12-22	Komedia	93 min		
Jumanji: Przygoda w dżungli	Chris McKenna	2017-12-29	Przygodowy	119 min		

Dodaj film

Tytuł
Paddington 2

Reżyser
Paul King

Rok produkcji
29.12.2017

Gatunek
Familijny

Długość (min)
104

Dodaj film

Projekt z rozproszonych baz danych 2017

Rysunek 6.1 Dodawanie nowego filmu poprzez aplikację

localhost:8082/sql.php?server=1&db=Cinema&table=Filmy&pos=0

phpMyAdmin

Recent

Favorites

New

Cinema

Procedures

Tables

New

Bliety

Filmy

Miejsca

Rezerwacje

Sale

Seanse

Uprawnienia

Uzytkownicy

Views

Information_schema

mysql

performance_schema

sys

Server: db - Database: Cinema - Table: Filmy

Browse

Structure

SQL

Search

Insert

Export

Import

Privileges

Operations

Triggers

Showing rows 0 - 10 (11 total, Query took 0.0005 seconds.)

SELECT * FROM `Filmy`

Show all

Number of rows: 25

Filter rows: Search this table

Sort by key: None

+ Options

<

Rysunek 6.2 Tabela Filmy na węźle *master* o porcie 8082

Rozdział 7

Podsumowanie

Z powodzeniem zaprojektowano oraz zaimplementowano system obsługi bazodanowej kina. Wszelkie problemy związane z projektem zostały wyjaśniane i wnikliwie konsultowane z prowadzącym zajęcia projektowe. Mechanizmy wykorzystywane były skrupulatnie prezentowane na zajęciach, po drobnych korektach wprowadzono je do systemu bazodanowego tak, aby dążyć do implementacji pozbawionej błędów formalnych. System po wykonaniu testów -dodawania tabel, wypełniania ich rekordami działał poprawnie. Przetestowano moduły zabezpieczeń wspomniane w całym sprawozdaniu projektowym, również funkcjonowały poprawnie (zabezpieczenia m.in. przed SQLInjection).Projekt całościowo prezentowano na zajęciach projektowych.

W trakcie prac nad realizacją projektu udało się wykorzystać mechanizm replikacji *master-slave* oraz *master-slave* z opóźnieniem. Do ich konfiguracji wykorzystane zostało narzędzie *phpMyAdmin*.

Podczas prac nad projektem próbowano wykorzystać jako węzły rozproszonej bazy danych serwery na fizycznych urządzeniach *Raspberry PI*, jednak ilość dostępnych przez nas urządzeń nie pozwalała w pełni pokazać możliwości replikacji w systemie bazodanowym MySQL. Dodatkowo również na system operacyjny *raspbrian* nie była dostępna najnowsza wersja MySQL, przez co nie było możliwości m.in. wykonania replikacji z opóźnieniem. Również by replikacja działała wymagane było połączenie internetowe we wszystkich urządzeniach, łącznie z urządzeniem, na którym demonstrowane było działanie. W związku z tym w trakcie prac zrezygnowano z fizycznych urządzeń, a do stworzenia węzłów wykorzystano wirtualne kontenery stworzone w programie *docker*, przez co rozproszona baza danych działa w pełni lokalnie. W każdym konterze można było dodać dowolną wersję MySQL oraz *phpMyAdmin*.

Interfejs aplikacji bazodanowej jest przejrzysty i intuicyjny, nie jest skomplikowany, dodawanie kolejnych informacji – rekordów nie stanowi żadnych problemów, co jest ważnym atrybutem ze strony spojrzenia konsumenckiego. Implementacja całego projektu pozwoliła na swobodne korzystanie ze strony aktora pracownik jak i aktora klient. Klient posiada inne uprawnienia(na płaszczyźnie szeroko rozumianej rezerwacji seansu w kinie wraz z możliwością opłacenia). Pracownik ma możliwość przeglądania tych rezerwacji, dokonywać ich modyfikacji (np. nagła zmiana repertuaru). Zgodnie z warunkami zadania zaimplementowano możliwość wyświetlania odpowiednich widoków – lista filmów, seansów i biletów.

Spełniono wszystkie założenia projektowe postawione przez prowadzącego zajęcia jak i sprostanio własnym wymaganiom.

Literatura

- [1] Thomson L., Welling L., *PHP i MySQL. Tworzenie stron WWW*, Helion, Gliwice, 2001.
- [2] Strona internetowa: <http://wazniak.mimuw.edu.pl/index.php> - systemy rozproszone, zaawansowane systemy baz danych, dostęp: 22-11-2017.
- [3] Meloni J. C., *PHP-programowanie*, RM, Warszawa, 2001.
- [4] Knopczyński P., Talarczyk M., *Duplikacja i replikacja MySQL*, dostęp: 22-11-2017
- [5] Strona internetowa: <http://angular.io/> - Angular 2, dostęp: 20-11-2017.
- [6] Strona internetowa: <http://nodejs.org/en/> - node.js, dostęp: 20-11-2017.
- [7] Strona internetowa: <http://www.docker.com/> - docker, dostęp: 25-11-2017.