

POLITECHNIKA WROCŁAWSKA  
WYDZIAŁ ELEKTRONIKI

---

KIERUNEK: Informatyka  
SPECJALNOŚĆ: Inżynieria systemów informatycznych

**Rozproszony system bazodanowy  
przeznaczony do rezerwacji biletów do  
teatru**

Projekt z rozproszonych i obiektowych systemów  
baz danych

AUTORZY:  
Kamil Babicki 200824, Samir Senhadri 200003

PROWADZĄCY ZAJĘCIA:  
Dr inż. Robert Wójcik, W4/I-6

OCENA PRACY:



# Spis treści

<b>1</b>	<b>Wstęp</b>	<b>5</b>
1.1	Cele projektu . . . . .	5
1.2	Założenia projektowe . . . . .	5
<b>2</b>	<b>Analiza wymagań</b>	<b>7</b>
2.1	Opis działania systemu . . . . .	7
2.2	Wymagania funkcjonalne . . . . .	7
2.3	Wymagania нефunkcjonalne . . . . .	8
2.3.1	Wykorzystane technologie i narzędzia . . . . .	8
2.3.2	Wymagania dotyczące rozmiaru bazy danych . . . . .	8
2.3.3	Wymagania dotyczące bezpieczeństwa systemu . . . . .	8
2.4	Przyjęte założenia projektowe . . . . .	9
<b>3</b>	<b>Projekt systemu</b>	<b>11</b>
3.1	Projekt bazy danych . . . . .	11
3.1.1	Model logiczny . . . . .	11
3.1.2	Model fizyczny i ograniczenia integralności bazy danych . . . . .	12
3.2	Projekt aplikacji użytkownika . . . . .	13
<b>4</b>	<b>Implementacja systemu</b>	<b>15</b>
4.1	Realizacja bazy danych . . . . .	15
4.2	Realizacja mechanizmu replikacji . . . . .	15
4.3	Realizacja elementów aplikacji . . . . .	17
4.3.1	Realizacja części klienckiej . . . . .	17
<b>5</b>	<b>Testowanie systemu</b>	<b>21</b>
5.1	Testowanie poprawności działania systemu . . . . .	21
5.2	Testowanie mechanizmów bezpieczeństwa bazy danych . . . . .	22
5.3	Wnioski z testów . . . . .	22
<b>6</b>	<b>Podsumowanie pracy</b>	<b>23</b>
	<b>Bibliografia</b>	<b>25</b>
	<b>Spis rysunków</b>	<b>27</b>



# Rozdział 1

## Wstęp

Prezentowany projekt został opracowany w trakcie realizowania laboratorium z przedmiotu *Rozproszone i obiektowe systemy baz danych*. W trakcie prac projektowych stworzona została rozproszona baza danych miejsc w teatrze. W bazie tej przechowywane są informacje na temat rezerwacji, klientów, spektakli, cen biletów i innych danych potrzebnych do działania teatru.

### 1.1 Cele projektu

Celem projektu jest zaprojektowanie oraz implementacja systemu usprawniającego pracę teatru. Interfejsem użytkownika będzie aplikacja umożliwiającej dostęp do rozproszonej bazy danych systemu rezerwacji biletów do teatru. System będzie posiadał trzy serwery, na których zainstalowane będą bazy danych systemu rezerwacji biletów do teatru. Dostęp do danych będzie odbywał się poprzez przeglądarkę internetową, za pomocą specjalnie zaimplementowanej aplikacji umieszczonej na serwerze WWW. Stworzona aplikacja umożliwi sprawdzanie zajętości sal na wybrane spektakle oraz rezerwację biletów.

### 1.2 Założenia projektowe

W projekcie będzie zrealizowany 3-warstwowy model komunikacji klient/serwer w postaci tzw. „cienkiego klienta”. Dostęp do aplikacji realizującej funkcje biznesowe będzie realizowany poprzez podanie adresu serwera WWW. W projekcie zostaną wykorzystywane jednorodne bazy danych. Dostęp do systemu baz danych zostanie zrealizowany w oparciu o funkcje aplikacji. Komunikacja w systemie baz danych będzie przebiegała zgodnie z modelem master slave. Zapis będzie możliwy tylko na serwer z rolą mastera, natomiast odczyt będzie możliwy z wszystkich trzech serwerów bazodanowych. W celu zrównoważenia obciążenia wykorzystany zostanie Load Balancing. Dostęp do baz danych będzie możliwy za pośrednictwem odpowiednich serwerów zarządzających bazami danych MySQL, a proces replikacji między nimi zostanie zrealizowany z wykorzystaniem programu MySQL Workbench. Ze stworzonego systemu baz danych będzie korzystać aplikacja webowa, zaimplementowana w języku Ruby. Do specyfikacji funkcji systemu wykorzystany zostanie język modelowania UML.



# Rozdział 2

## Analiza wymagań

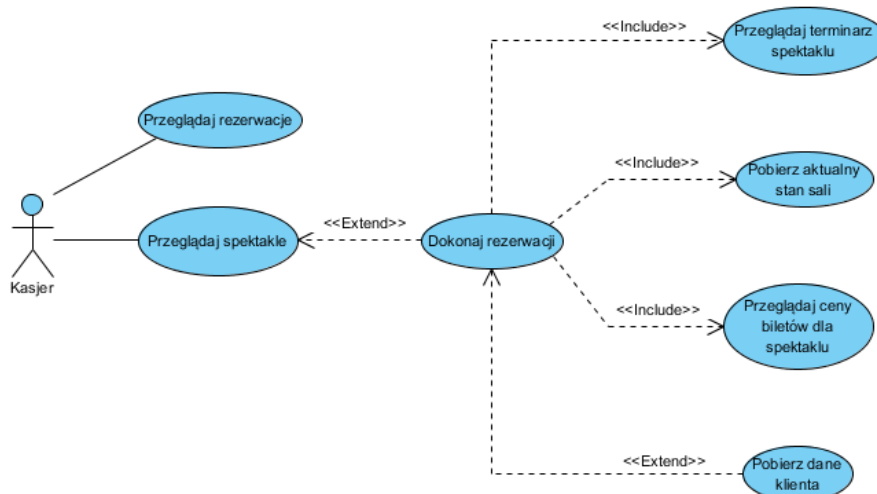
Tworzona aplikacja powinna realizować poniższe wymagania funkcjonalne dotyczące logiki biznesowej oraz wymagania niefunkcjonalne dotyczące zagadnień bezpieczeństwa czy sposobu działania.

### 2.1 Opis działania systemu

System będzie posiadał trzy serwery bazodanowe, na których zainstalowane będą bazy danych systemu rezerwacji biletów do teatru. Dostęp do danych będzie odbywał się poprzez przeglądarkę internetową, za pomocą specjalnie zaimplementowanej aplikacji umieszczonej na serwerze WWW. Stworzona aplikacja umożliwi sprawdzanie zajętości sal na wybrane spektakle oraz rezerwację biletów.

### 2.2 Wymagania funkcjonalne

Wymagania funkcjonalne zostały przedstawione za pomocą diagramu przypadków użycia (Rys. 2.1).



Rysunek 2.1 Diagram przypadków użycia

W naszym systemie przewidziana została tylko jedna rola, jaką jest kasjer. Powinien on mieć możliwość przeglądania dokonanych rejestracji, a także wprowadzenia do systemu

nowych. Przy opcji nowej rezerwacji, pracownik na podstawie informacji przekazanych przez klienta teatru, będzie miał możliwość wyboru sceny, terminu i wolnych miejsc na sali. Dokończenie procesu rezerwacji będzie wymagało wprowadzenia danych klienta. Jeżeli gość istnieje już w bazie danych, jego dane będzie można pobrać na podstawie adresu e-mail.

## 2.3 Wymagania niefunkcjonalne

Tworzona aplikacja ma być aplikacją webową. Umożliwi to użytkownikom dostęp do niej z niemal dowolnego urządzenia elektronicznego. Powinna być również responsywna, aby mimo różnych rozdzielczości ekranów zapewnić zadowalającą czytelność i funkcjonalność. Sam jej wygląd powinien być możliwie jak najprostszy, aby użytkownik nie czuł uczucia dyskomfortu podczas poruszania się po niej oraz by nie musiał poświęcić dużo czasu na nauczanie się jej obsługi.

### 2.3.1 Wykorzystane technologie i narzędzia

Dostęp do baz danych będzie możliwy za pośrednictwem odpowiednich serwerów zarządzających bazami danych MySQL, a proces replikacji między nimi zostanie zrealizowany z wykorzystaniem programu MySQL Workbench. Ze stworzonego systemu baz danych będzie korzystać aplikacja webowa. Logika aplikacji została zaimplementowana przy użyciu języka Ruby wraz z biblioteką Sinatra. Natomiast warstwa prezentacji budowana była w HTML5, CSS3, Bootstrap, JavaScript i AngularJS. Do specyfikacji funkcji systemu wykorzystany zostanie język modelowania UML.

### 2.3.2 Wymagania dotyczące rozmiaru bazy danych

Baza danych stworzona zostanie do obsługi procesu rezerwacji biletów do teatru. Będzie to wymagało m.in. przechowywania danych o klientach, a także aktualnych rezerwacjach. Na stałe w bazie danych wprowadzone zostaną informacje o dostępnych salach. Informacje o aktualnie granych spektaklach, biletów na nie oraz terminarzu, nie będą wymagały dużej ilości miejsca. Najbardziej obciążone tabele bazy będą przechowywały dane klientów i rezerwacji. Zakładając, że w teatrze w danym okresie będzie granych kilka sztuk na różnych salach i w różnych terminach możemy spodziewać się, że dane o tym będą zapisane na co najmniej kilkaset rekordach. Jednakże informacje te będzie można usunąć po odbyciu się spektaklu, a więc baza danych nigdy nie powinna osiągnąć dużych rozmiarów.

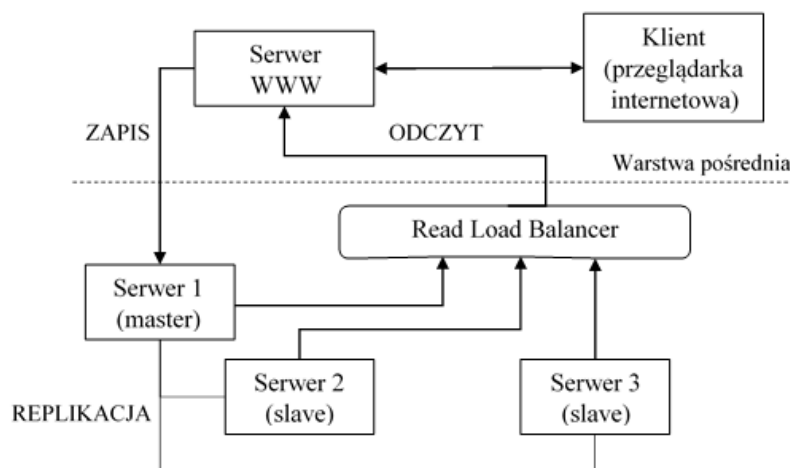
### 2.3.3 Wymagania dotyczące bezpieczeństwa systemu

Najbardziej newralgiczne dane będą dotyczyły klientów teatru. W bazie będą przechowywane takie informacje jak numer telefonu, czy adres e-mail. Jednakże system został przeznaczony tylko dla pracowników teatru, a wcześniej wymienione dane nie są aż tak niebezpieczne by były dodatkowo szyfrowane. Dlatego przyjmujemy, że stworzony system nie wymaga dodatkowych zabezpieczeń. Za bezpieczeństwo wprowadzania danych odpowiadać będą zaimplementowane ograniczenia integralności bazy danych.



## 2.4 Przyjęte założenia projektowe

W projekcie będzie zrealizowany 3-warstwowy model komunikacji klient/serwer w postaci tzw. „cienkiego klienta” (Rys. 2.2). Dostęp do aplikacji realizującej funkcje biznesowe będzie realizowany poprzez podanie adresu serwera WWW. W projekcie zostaną wykorzystywane niejednorodne, relacyjne bazy danych. Dostęp do systemu baz danych zostanie zrealizowany w oparciu o funkcje aplikacji. Komunikacja w systemie baz danych będzie przebiegała zgodnie z modelem master slave. Zapis będzie możliwy tylko na serwer z rolą mastera, natomiast odczyt będzie możliwy z wszystkich trzech serwerów bazodanowych. W celu zrównoważenia obciążenia wykorzystany zostanie Load Balancing.



Rysunek 2.2 Schemat systemu



# Rozdział 3

## Projekt systemu

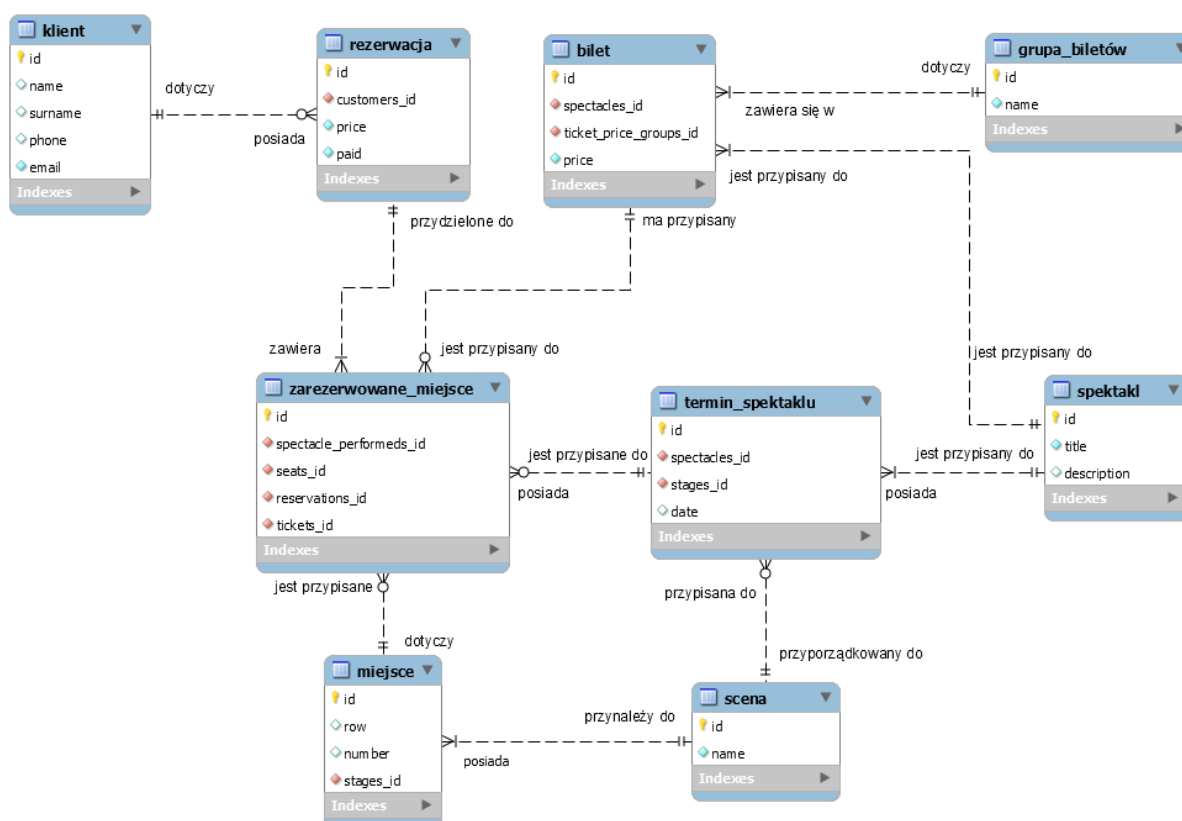
W tym rozdziale przedstawiony został sposób w jaki zaprojektowana będzie aplikacja i jej baza danych.

### 3.1 Projekt bazy danych

Na etapie projektowania okazało się, że do właściwej pracy systemu będzie potrzebna baza danych składająca się z dziewięciu tabel.

#### 3.1.1 Model logiczny

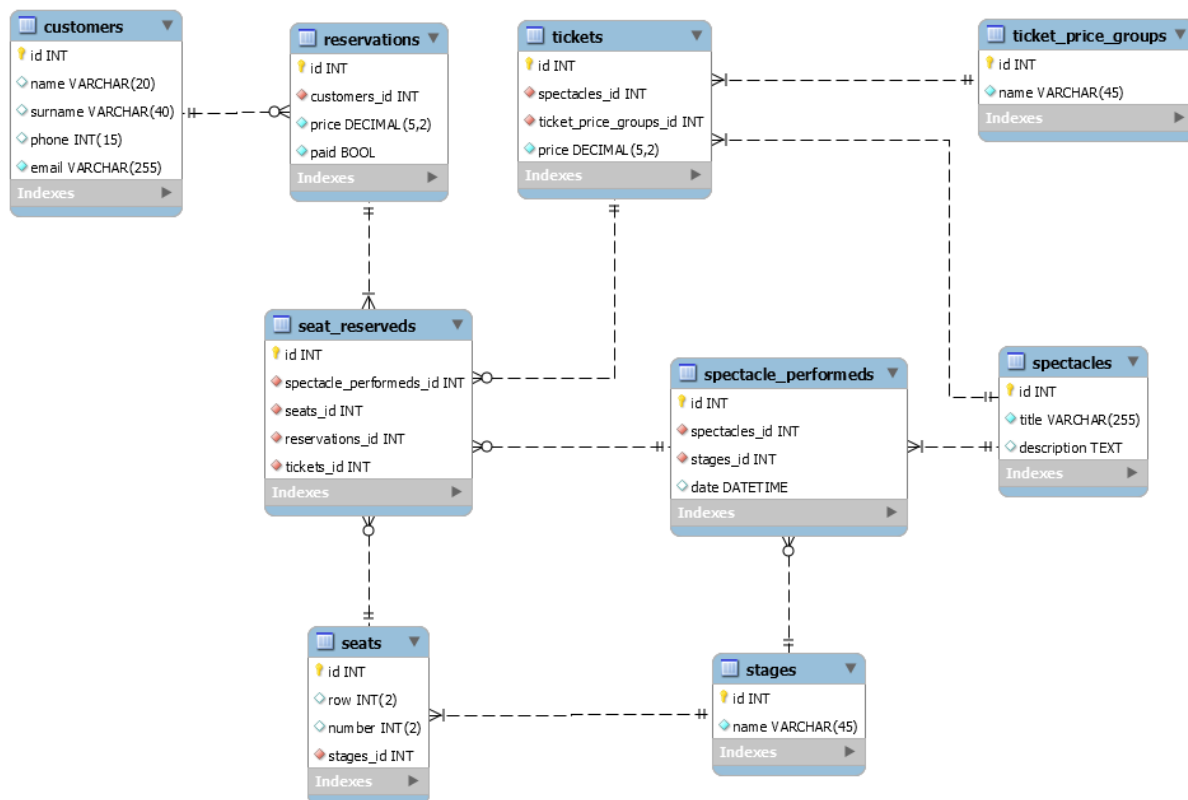
Model logiczny bazy danych został przedstawiony na rysunku 3.1.



Rysunek 3.1 Schemat logiczny bazy danych

### 3.1.2 Model fizyczny i ograniczenia integralności bazy danych

Fizyczny model bazy danych (Rys. 3.2) został stworzony dla bazy MySQL. W modelu tym obok nazw kolumn w tabelach przedstawione zostały oczekiwane typy danych.



Rysunek 3.2 Schemat fizyczny bazy danych

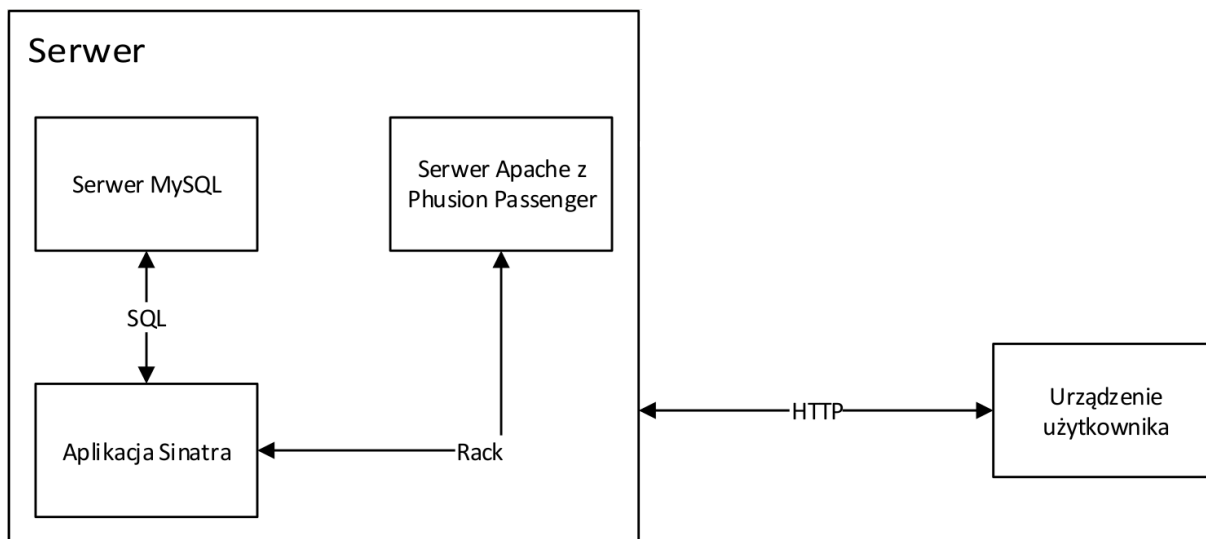
Dodatkowo na odpowiednie tabele nałożone zostały następujące ograniczenia integralności:

- customers - na wszystkie kolumny not-null (NN), dodatkowo unique (U) na kolumnę email,
- reservations - na wszystkie kolumny NN,
- tickets - na wszystkie kolumny NN, dodatkowo U na parę kolumn spectacles\_id i ticket\_price\_groups\_id,
- ticket\_price\_groups - na wszystkie kolumny NN, na kolumnę name U,
- seat\_reserveds - na wszystkie kolumny NN, na parę spectacle\_performeds\_id i seats\_id U,
- spectacle\_performeds - na wszystkie kolumny NN, na parę stages\_id i date U,
- spectacles - na wszystkie kolumny NN,
- seats - na wszystkie kolumny NN, na grupę row, number i stages\_id U,
- stages - na wszystkie kolumny NN, na kolumnę name U.

## 3.2 Projekt aplikacji użytkownika

Aplikacja użytkownika składa się z dwóch części:

- Backend - program napisany w języku Ruby z wykorzystaniem biblioteki Sinatra. Komunikuje się on z bazą danych oraz odpowiada na żądania frontendu. Działa na serwerze
- Frontend - program napisany przy użyciu platformy programistycznej AngularJS w języku JavaScript. Działa na urządzeniu użytkownika.



Rysunek 3.3 Schemat działania aplikacji użytkownika



# Rozdział 4

## Implementacja systemu

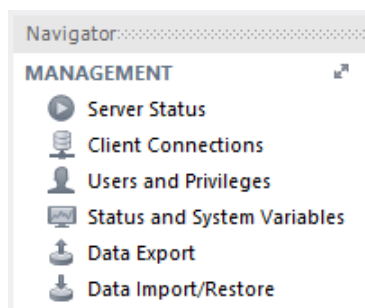
W tym rozdziale opisany został sposób w jaki stworzona została struktura bazy danych oraz jak wyglądała implementacja logiki biznesowej aplikacji.

### 4.1 Realizacja bazy danych

Baza danych została zaimplementowana w podejściu Database First. Została więc zaprojektowana, a następnie wdrożona za pomocą programu MySQL Workbench. Projektowanie odbywało się całkowicie w trybie graficznym. Następnie korzystając z narzędzi programu wygenerowany został skrypt sql tworzący bazę danych, który został wczytany do serwerów typu slave.

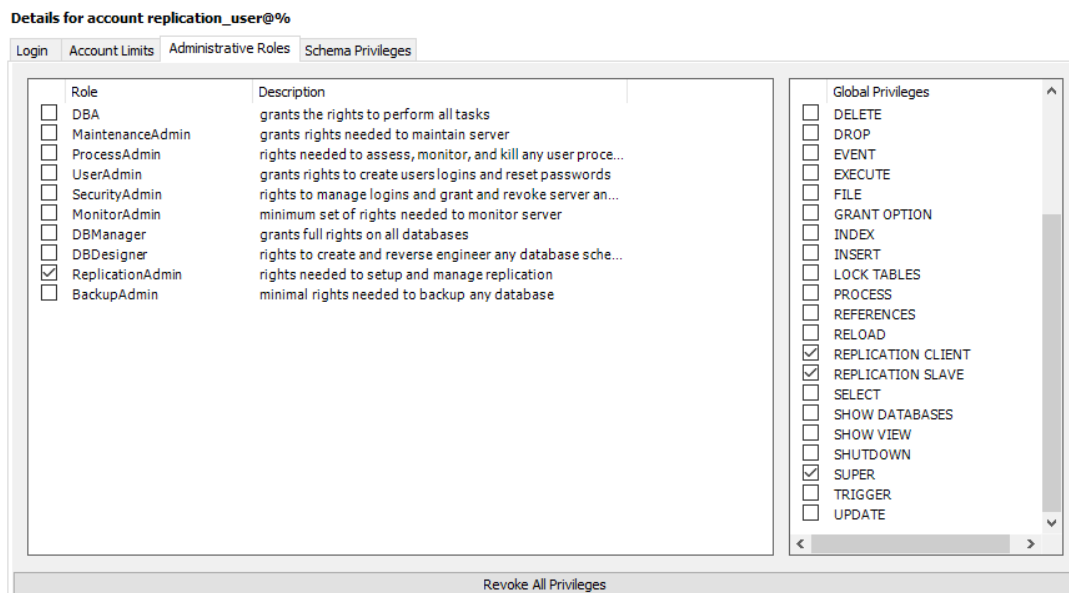
### 4.2 Realizacja mechanizmu replikacji

Replikacja baz danych została przeprowadzona z wykorzystaniem programu MySQL Workbench. Przed przystąpieniem do jakichkolwiek czynności należy upewnić czy wszystkie bazy posiadają dokładnie taką samą strukturę. W naszym systemie struktura została zaprojektowana, a następnie wprowadzona na serwerze typu master. Po sprawdzeniu poprawności jej działania, korzystając z wcześniej wymienionego programu, struktura bazy została wyeksportowana w postaci skryptu sql. Następnie skrypt ten został wczytany na serwery typu slave. Opcja eksportu i importu bazy danych znajduje się w lewym menu programu w grupie "Management" (Rys. 4.1).



Rysunek 4.1 Grupa opcji Management programu MySQL Workbench

Następnie dla celów replikacji w bazie danych typu master stworzony został użytkownik replication\_user o uprawnieniach przedstawionych na rysunku 4.2.

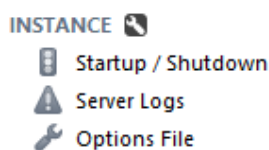


Rysunek 4.2 Uprawnienia użytkownika replication\_user

Następnie wybierając opcję "Options File" (Rys. 4.3) przechodzi się do ustawień serwera bazy danych. Na serwerze mastera należało aktywować i ustawić następujące opcje:

- server\_id na 1,
- binlog-do-db na nazwę bazy danych theatre,
- log-bin na mysql55-bin.

Pierwsza opcja dotyczy nadania serwerowi numeru id, dzięki czemu możliwa będzie komunikacja z innymi serwerami. W drugiej określa się bazę danych jaka będzie podlegać replikacji. W ostatniej ustawia się nazwę dla pliku w którym będą przechowywane logi z działania serwera.



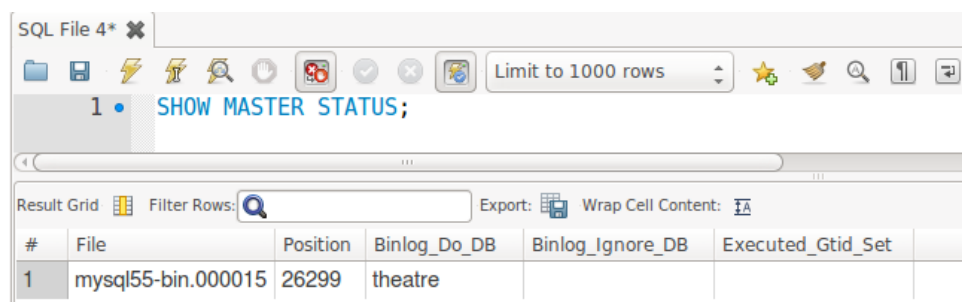
Rysunek 4.3 Grupa opcji Instance programu MySQL Workbench

Po zatwierdzeniu wprowadzonych danych warto zrestartować serwer, a następnie zwerifikować status mastera za pomocą odpowiedniej komendy, przedstawionej wraz z wynikiem na rysunku 4.4.

Następnie należy wykonać konfigurację serwerów typu slave. Ponownie wybieramy opcję "Options File" (Rys. 4.3), gdzie należy ustawić następujące opcje:

- server\_id na liczbę większą od 1,
- replicate-do-db na nazwę bazy danych theatre.

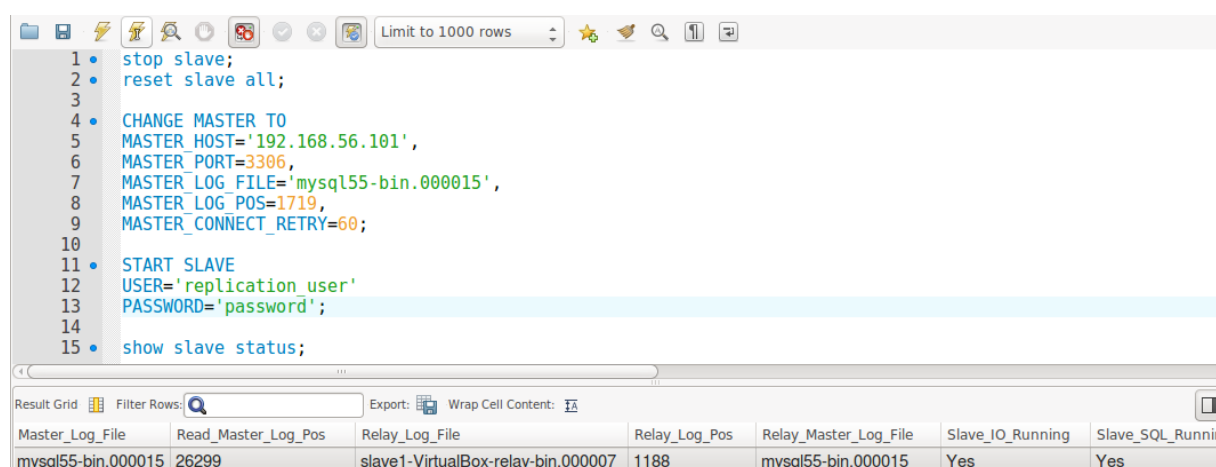




#	File	Position	Binlog_Do_DB	Binlog_Ignore_DB	Executed_Gtid_Set
1	mysql55-bin.000015	26299	theatre		

Rysunek 4.4 Wyświetlenie statusu serwera typu master

Tak jak przy masterze ustawiane jest id oraz opcja replicate-do-db wskazująca bazę jaką ma być replikowana. Po zatwierdzeniu wprowadzonych danych należy zrestartować serwer. Serwer typu slave nie jest jeszcze poprawnie ustawiony, należy wykonać jeszcze kilka komend, które wraz z wynikiem przedstawione zostały na rysunku 4.5.



```

1 • stop slave;
2 • reset slave all;
3
4 • CHANGE MASTER TO
5 MASTER_HOST='192.168.56.101',
6 MASTER_PORT=3306,
7 MASTER_LOG_FILE='mysql55-bin.000015',
8 MASTER_LOG_POS=1719,
9 MASTER_CONNECT_RETRY=60;
10
11 • START SLAVE
12 USER='replication_user'
13 PASSWORD='password';
14
15 • show slave status;

```

Master_Log_File	Read_Master_Log_Pos	Relay_Log_File	Relay_Log_Pos	Relay_Master_Log_File	Slave_IO_Running	Slave_SQL_Running
mysql55-bin.000015	26299	slave1-VirtualBox-relay-bin.000007	1188	mysql55-bin.000015	Yes	Yes

Rysunek 4.5 Ustawienie oraz sprawdzenie statusu serwera typu slave

Na początku wykonywane są komendy odpowiedzialne za ewentualne zatrzymanie działania slave'a i zresetowanie wszelkich ustawionych na nim opcji. W następnej komendzie opcja MASTER\_LOG\_FILE i MASTER\_LOG\_POS to nazwa pliku i pozycja jaką można odczytać po wyświetleniu statusu mastera (Rys. 4.4). Dalej serwer jest uruchamiany z wykorzystaniem użytkownika replication\_user. Na samym końcu umieszczona została komenda wyświetlająca aktualny status serwera typu slave. O prawidłowości wykonanej replikacji świadczą kolumny Slave\_IO\_Running i Slave\_SQL\_Running, które powinny być ustawione na wartość YES.

## 4.3 Realizacja elementów aplikacji

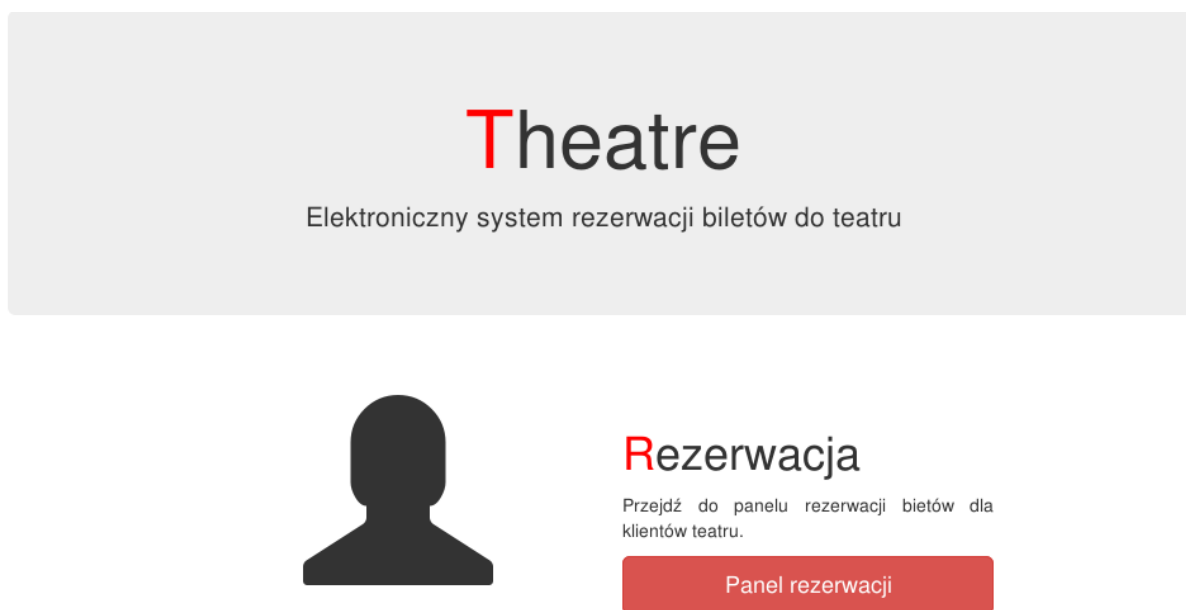
### 4.3.1 Realizacja części klienckiej

Część kliencka została zrealizowana jako aplikacja webowa z wykorzystaniem przede wszystkim frameworków AngularJS i Bootstrap. Pierwszy z nich odpowiedzialny był za połączenie z serwerem, przygotowywanie danych do wyświetlenia, operacji na nich, a także walidacji. Drugi został wykorzystany w celu lepszej organizacji treści na stronie i do utrzymania responsywności strony.

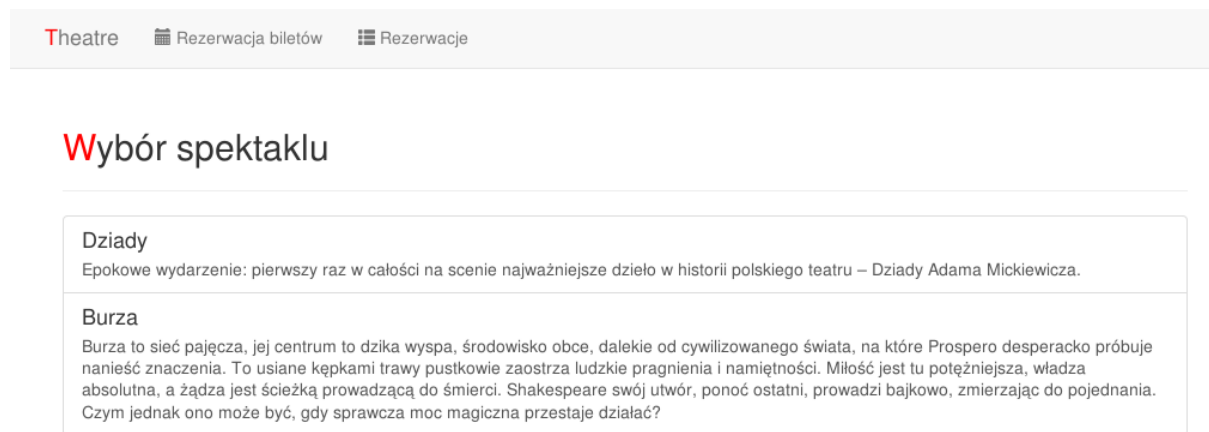
Przy wejściu do aplikacji pokazuje się strona startowa (Rys. 4.6). Z niej użytkownik systemu może przejść do panelu rezerwacji, wciskając widoczny przycisk. Po wykonaniu tej czynności wyświetla się strona z listą spektakli (Rys. 4.7). Po wyborze jednego z nich przechodzi się do następnego widoku (Rys. 4.9).

W tym momencie pracownik ma możliwość zarezerwowania biletów dla klienta. Na początku powinien wybrać scenę i datę kiedy ma odbyć się spektakl. Po wykonaniu tego na stronie pojawi się model przedstawiający rozłożenie miejsc na sali. Siedzeniom przypisane zostały odpowiednie numery i różne kolory. Kolorem zielonym oznaczone zostały dostępne miejsca. Kolorem siwym wyróżniono miejsca już przez kogoś zarezerwowane. Natomiast kolorem czerwonym oznaczane są miejsca wybrane dla aktualnie obsługiwanego klienta. Poniżej sceny widoczna jest sekcja "Wybrane miejsca", gdzie ponownie wypisane zostały wybrane miejsca, ale dodatkowo do każdego z nich należy przypisać bilet. W prezentowanym przypadku mamy doczynienie z biletami normalnym i ulgowym. Jednakże nazwy i oczywiście ceny dla biletów można definiować własne. Po wyborze biletów należy już tylko uzupełnić dane o kliencie. Wszystkie te dane są odpowiednio walidowane, a w przypadku niepoprawnie wprowadzonych danych, użytkownikowi wyświetlany jest krótki komunikat tuż przy błędnym polu. Jeżeli klient dokonywał wcześniej jakieś inne rezerwacje, jego dane powinny znajdować się w systemie. W celu ich pobrania, pod polem "Email" umieszczony został link, który pobiera informację o kliencie na podstawie wprowadzonego adresu e-mail. Po uzupełnieniu wszystkich wymaganych informacji należy kliknąć przycisk "Zarezerwuj". Po krótkiej chwili pod formularzem powinien pojawić się komunikat czy rezerwacja się powiodła.

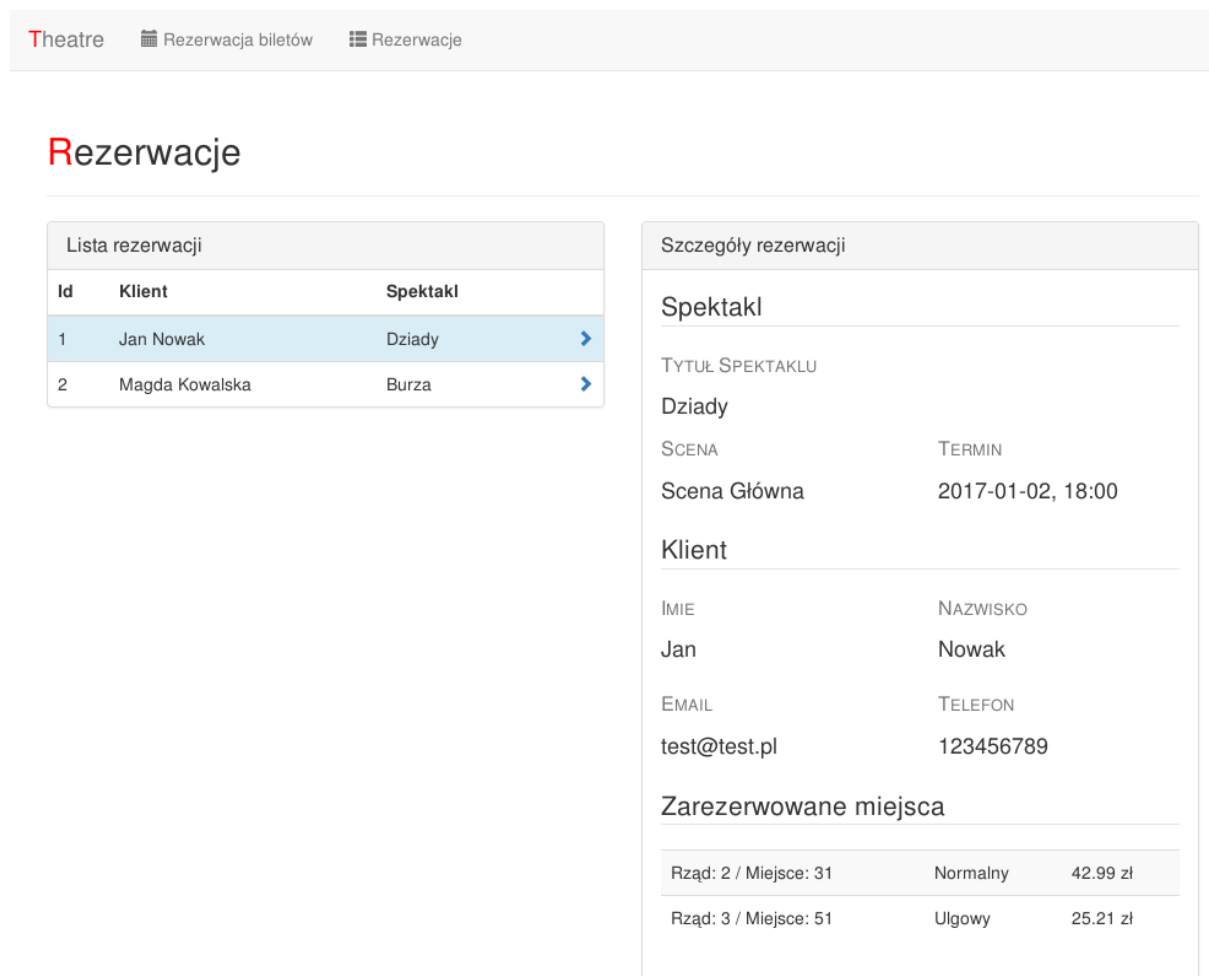
Ostatni widok jaki został zaimplementowany to widok z listą aktualnie dokonanych rezerwacji (Rys. 4.8). W lewej kolumnie znajduje się wspomniana lista, a w prawej szczegóły o wybranej rezerwacji.



Rysunek 4.6 Strona główna aplikacji



Rysunek 4.7 Widok z listą spektakli



Rysunek 4.8 Widok z listą dokonanych rezerwacji

Theatre

Rezerwacja biletów

Rezerwacje

## Wybór miejsc

**Spektakl:** Burza [zmień](#)

**Scena:** Scena 8 piętro

**Data:** 2017-04-23, 17:00

Scena 8 piętro

SCENA																	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58
61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98

### Wybrane miejsca

Rząd: 4 / Miejsce: 69	Normalny - 38.0 zł
Rząd: 4 / Miejsce: 70	Ulgowy - 24.87 zł

**RAZEM: 62,87 zł**

### Osoba rezerwująca

<b>Email</b> <input type="text" value="jan.nowak@test.pl"/> <a href="#">Istniejący klient - pobierz resztę danych</a>	<b>Telefon</b> <input type="text" value="111111111"/>
<b>Imię</b> <input type="text" value="Jan"/>	<b>Nazwisko</b> <input type="text" value="Nowak"/>

Zarezerwuj

Rysunek 4.9 Widok przedstawiający przykładową rezerwację

# Rozdział 5

## Testowanie systemu

Testowanie zostało opisane w dwóch podrozdziałach dotyczących odpowiednio testowania działania systemu i testowania mechanizmów bezpieczeństwa bazy danych.

### 5.1 Testowanie poprawności działania systemu

W celu sprawdzenia poprawności działania systemu przeprowadzono proces rezerwacji biletów dla fikcyjnego klienta teatru. Proces ten zakończył się prawidłowo. Oznacza to, że widoczne na rysunku 5.1 dane klienta powinny zostać zapisane zarówno w bazie trzymanej na serwerze typu master, jak i w bazach dwóch pozostałych serwerów typu slave. W celu sprawdzenia tego wykonano odpowiednią komendę na wszystkich serwerach i na wszystkich otrzymano taki sam wynik zaprezentowany na rysunku 5.2.

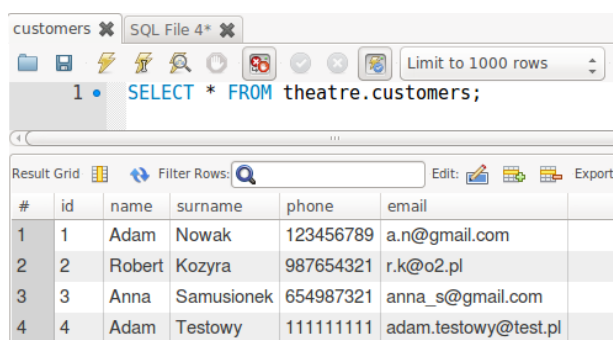
**Osoba rezerwująca**

<b>Email</b> adam.testowy@test.pl <small>Istniejący klient - pobierz resztę danych</small>	<b>Telefon</b> 111111111
<b>Imię</b> Adam	<b>Nazwisko</b> Testowy

Gratulacje! Twoja rezerwacja została zarejestrowana w systemie.

Zarezerwuj

Rysunek 5.1 Potwierdzenie wykonania rezerwacji



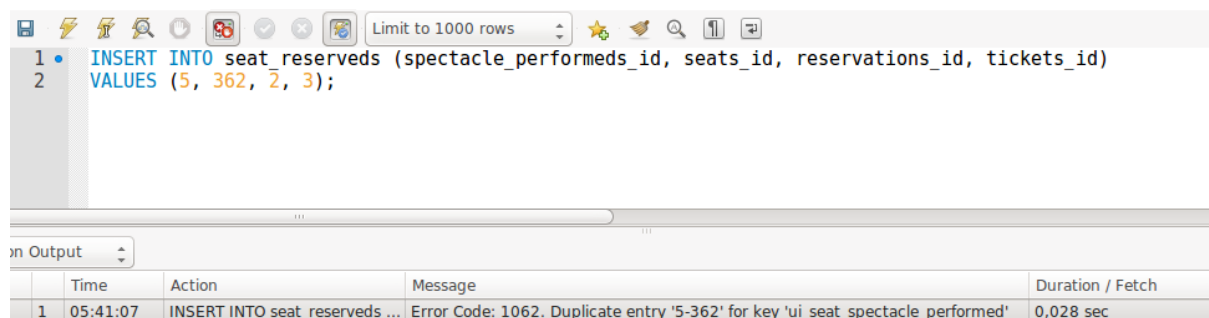
The screenshot shows a SQL client interface with a query window and a result grid. The query is `SELECT * FROM theatre.customers;`. The result grid displays four rows of customer data.

#	id	name	surname	phone	email
1	1	Adam	Nowak	123456789	a.n@gmail.com
2	2	Robert	Kozyra	987654321	r.k@o2.pl
3	3	Anna	Samusionek	654987321	anna_s@gmail.com
4	4	Adam	Testowy	111111111	adam.testowy@test.pl

Rysunek 5.2 Potwierdzenie wykonania rezerwacji

## 5.2 Testowanie mechanizmów bezpieczeństwa bazy danych

Bezpieczeństwo bazy danych może zostać pokazane na podstawie tabeli `seats_reserved`, która odpowiada za przechowywanie informacji o zarejestrowanych miejscach. W tej tabeli zostało ustawione ograniczenie niepozwalające wprowadzić rezerwację na już zajęte miejsce dla danego wystawienia spektaklu. Aktualne wypełnienie bazy danych zostało przedstawione na rysunku ???. Dla sprawdzenia omówionych mechanizmów bezpieczeństwa została wykonana odpowiednia komenda. Jej treść oraz zwrócony wynik zostały zawarte na rysunku 5.3. Jak widać ograniczenie `ui_seat_spectacle_performed` zwraca błąd nie pozwalając na dodanie niepoprawnych danych do bazy.



Rysunek 5.3 Próba zapisania dwóch osób na jedno miejsce zakończona niepowodzeniem

## 5.3 Wnioski z testów

Aplikacja działa poprawnie i zgodnie z przewidywaniami.

# Rozdział 6

## Podsumowanie pracy

W trakcie prac nad realizacją projektu udało się wykorzystać mechanizm replikacji master-slave. Do ich konfiguracji wykorzystane zostały wirtualne maszyny z systemami operacyjnymi Debian oraz Ubuntu. Zrealizowana została również aplikacja kliencka składająca się z części działającej na serwerze zwanej backendem napisanej w języku Ruby łącząca się z bazą danych oraz częścią działającą na urządzeniu użytkownika, która została zaimplementowana w środowisku programistycznym AngularJS. Cały projekt zakończył się sukcesem a działanie programu klienckiego spełnia wszystkie wcześniejsze założenia.





# Bibliografia

- [1] *Hypertext Transfer Protocol – HTTP/1.1* 1999: <https://tools.ietf.org/html/rfc2616> (dostęp 09.12.2015 r.)
- [2] *XMLHttpRequest Living Standard*: <https://xhr.spec.whatwg.org/> (dostęp 11.12.2015 r.)
- [3] Wrembel R., *Systemy rozproszonych baz danych [Online]*: <http://wazniak.mimuw.edu.pl/images/2/26/ZSBD-2st-1.2-w01.tresc-1.1.pdf> (dostęp 23.12.2016 r.)
- [4] *AngularJS API Docs*: <https://docs.angularjs.org/api> (dostęp 23.12.2016 r.)
- [5] *MySQL Documentation*: <https://dev.mysql.com/doc/> (dostęp 11.12.2016 r.)



# Spis rysunków

2.1	Diagram przypadków użycia . . . . .	7
2.2	Schemat systemu . . . . .	9
3.1	Schemat logiczny bazy danych . . . . .	11
3.2	Schemat fizyczny bazy danych . . . . .	12
3.3	Schemat działania aplikacji użytkownika . . . . .	13
4.1	Grupa opcji Management programu MySQL Workbench . . . . .	15
4.2	Uprawnienia użytkownika replication_user . . . . .	16
4.3	Grupa opcji Instance programu MySQL Workbench . . . . .	16
4.4	Wyświetlenie statusu serwera typu master . . . . .	17
4.5	Ustawienie oraz sprawdzenie statusu serwera typu slave . . . . .	17
4.6	Strona główna aplikacji . . . . .	18
4.7	Widok z listą spektakli . . . . .	19
4.8	Widok z listą dokonanych rezerwacji . . . . .	19
4.9	Widok przedstawiający przykładową rezerwację . . . . .	20
5.1	Potwierdzenie wykonania rezerwacji . . . . .	21
5.2	Potwierdzenie wykonania rezerwacji . . . . .	21
5.3	Próba zapisania dwóch osób na jedno miejsce zakończona niepowodzeniem . . . . .	22