Gliwice, 01.09.2017

**Bazy Danych**

**Kino**

Autorzy:

Sebastian Oprzędek

Bartek Gruba

Łukasz Kozień

Magdalena Kaleta

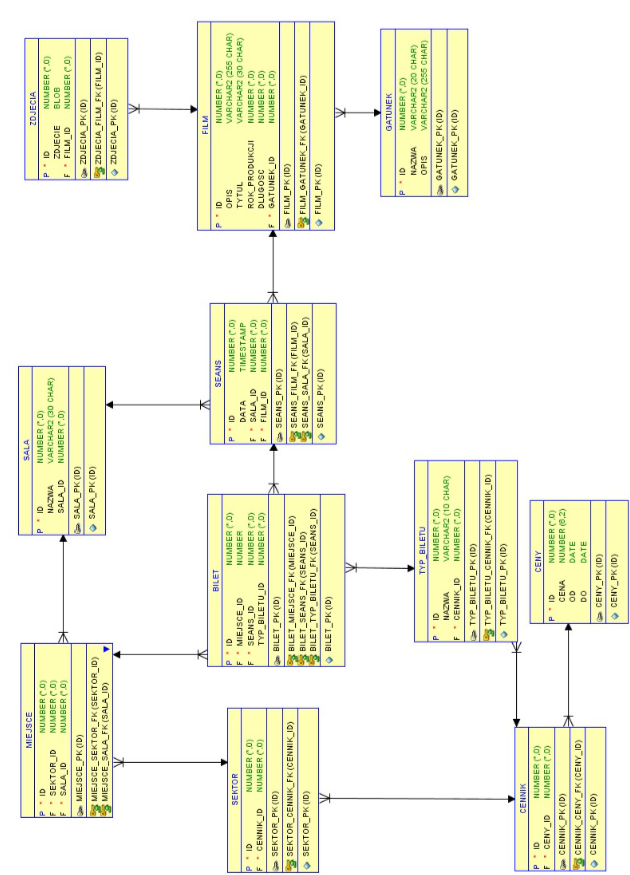
Edyta Haręza

**1. Temat**

Tematem naszego projektu było stworzenie systemu do administrowania kina, zarazem umożliwiającego rezerwację biletów na seans. Chcieliśmy, żeby obsługa była możliwe jak najprostsza i intuicyjna, tak aby każdy bez problemu mógł z niej skorzystać.

**2. Analiza, projektowanie**

Pierwszym i zarazem bardzo ważnym krokiem było stworzenie diagramu bazy danych. Potrzebowaliśmy uwzględnić bardzo dużo opcji, jakie powinien umożliwiać system obsługi kina. Musieliśmy zastanowić się bardzo dokładnie nad tym co i jak chcemy zrobić. Po dokładnym uzgodnieniu z prowadzącym i kilku poprawkach, udało nam się stworzyć możliwie jak najlepszy schemat. Na schemacie tym, oparliśmy później cały nasz projekt.



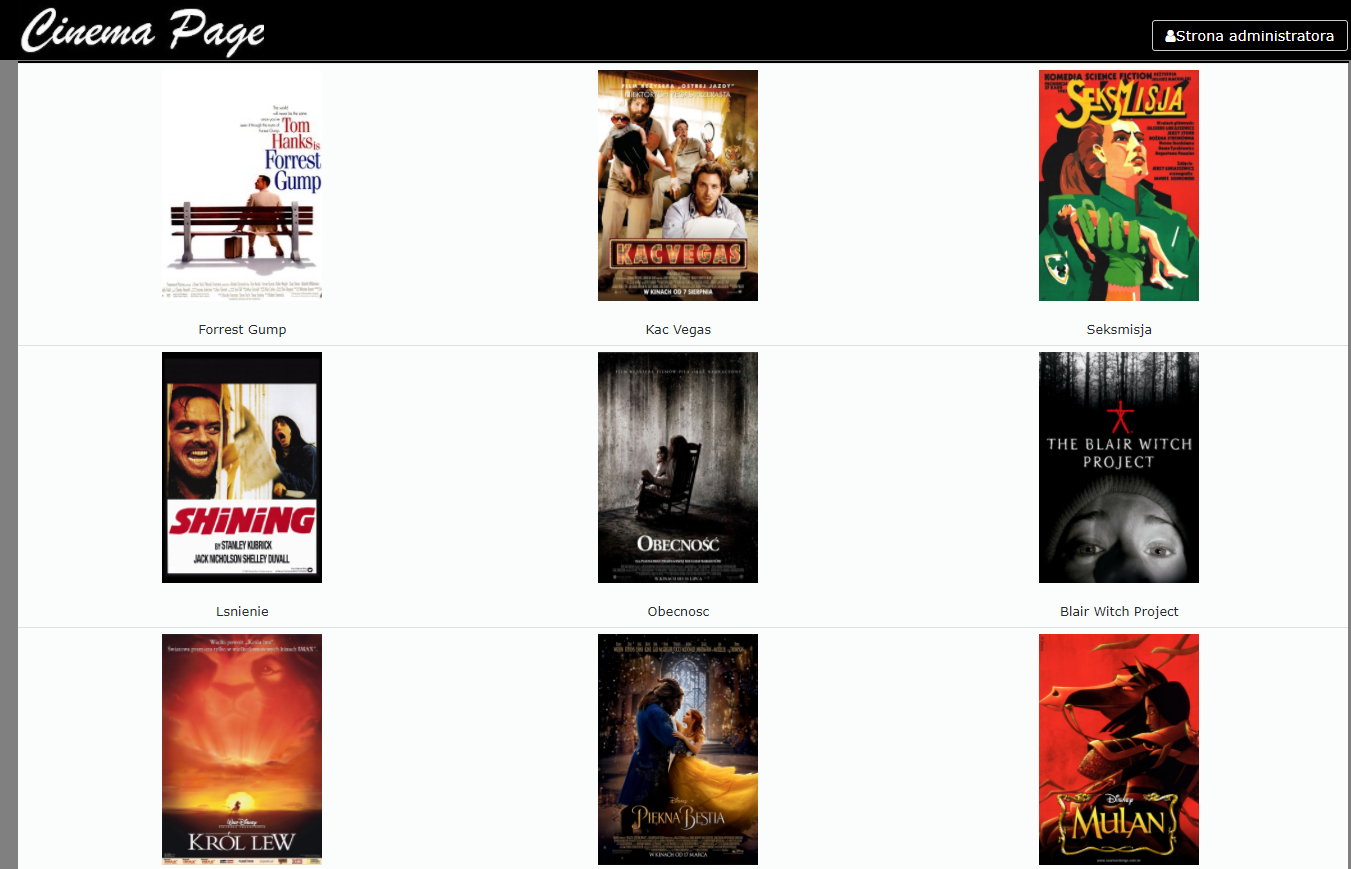
Jak widać na powyższym diagramie, będziemy mieli dostęp do takich danych jak:

* sale wraz z ich ilością miejsc i sektorów
* seanse odbywające się w tych salach
* możliwość zarządzania biletami i regulacji ich cen w zależności od typu biletu i sektora.
* możliwość dodawania na bieżąco filmów do
* wyświetlanie zdjęć filmu

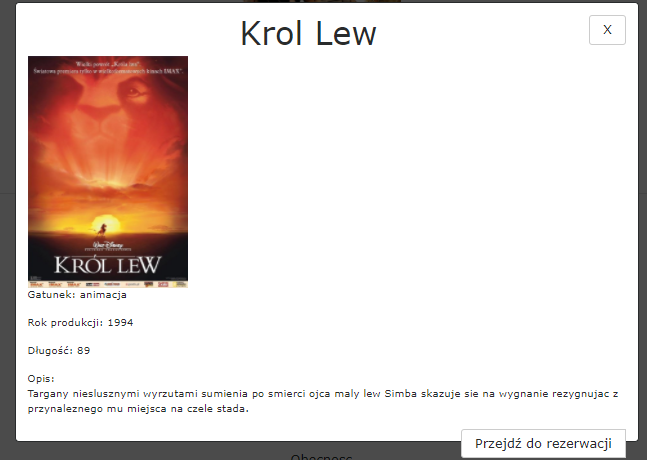
**3. Specyfikacja zewnętrzna**

**Klient:**

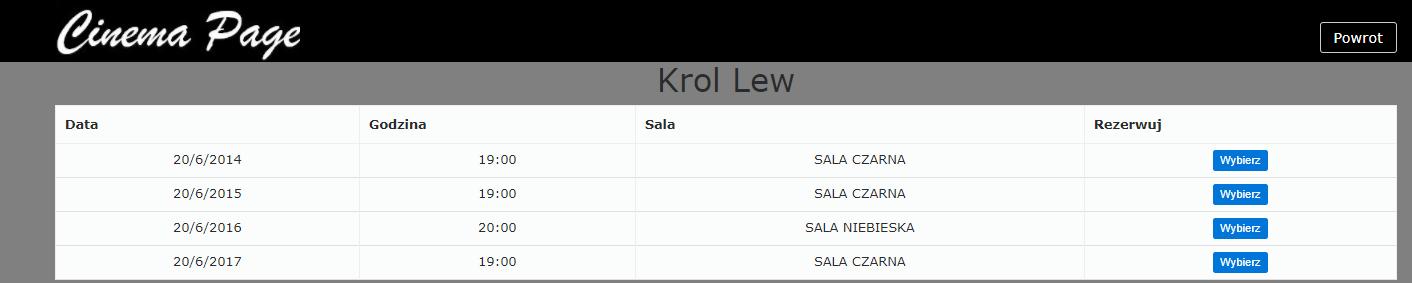
Po wejściu na stronę naszego kina naszym oczom ukażę się poniższy widok:



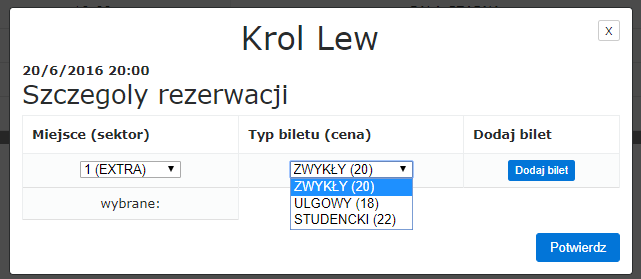
Jest to część widoczna od strony klienta. Po wyborze interesującego nas filmu pojawi się okienko z informacją o wybranym filmie oraz możliwością rezerwacji.



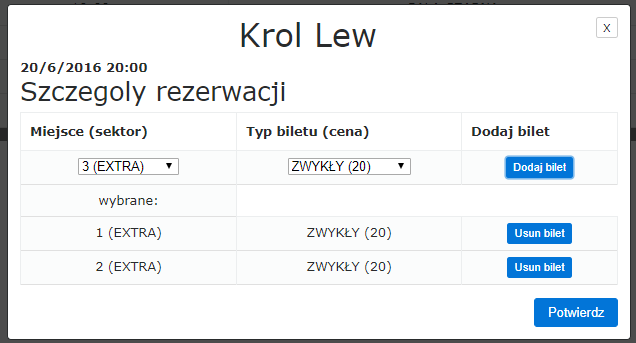
Po przejściu do rezerwacji mamy możliwość wyboru seansu:



Następnie ustalamy szczegóły rezerwacji. Wybieramy interesujący nas sektor oraz typ biletu.



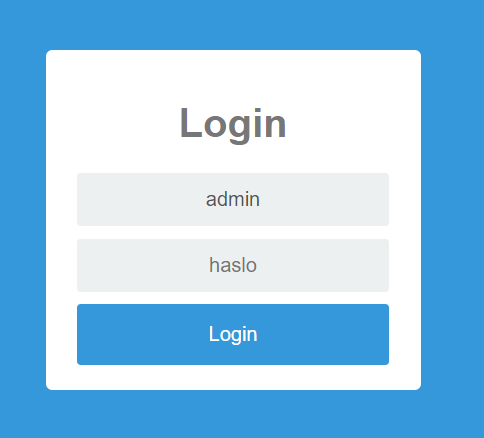
W razie pomyłki mamy również możliwość usunięcia biletu. Aby zakończyć rezerwacje klikamy przycisk **„Potwierdz”**



**3.1. Obsługa strony przed administratora**

Logowanie do panelu admina:

\*hasło - admin



Po zalogowaniu naszym oczom ukaże się pasek z zakładkami do zarządzania kinem oraz przycisk wylogowania się z panelu.

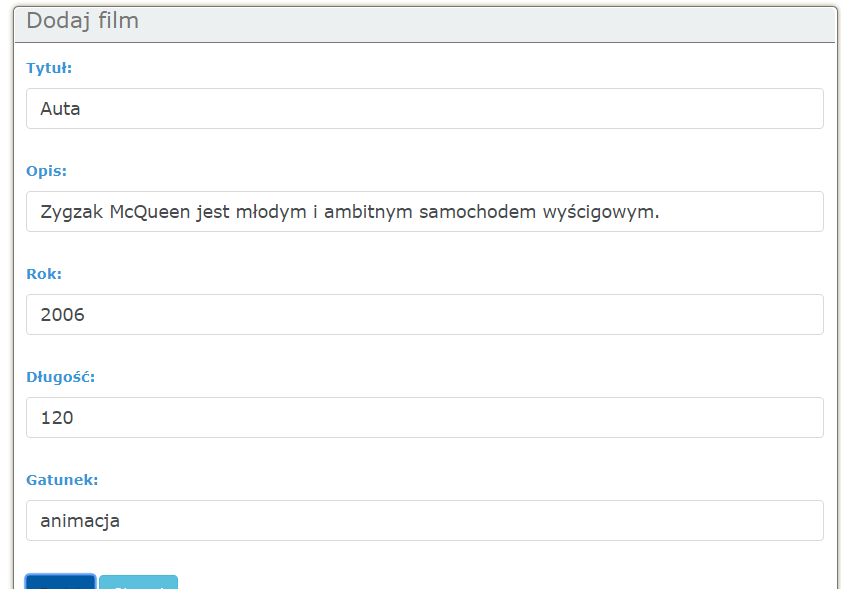


**3.2. Filmy**

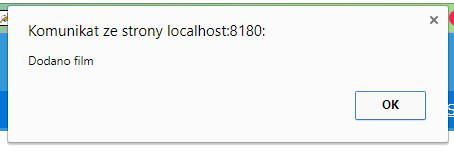
Zakładka „Filmy” służy do zarządzania samymi filmami. Mamy tutaj możliwość dodania oraz edycji filmu.



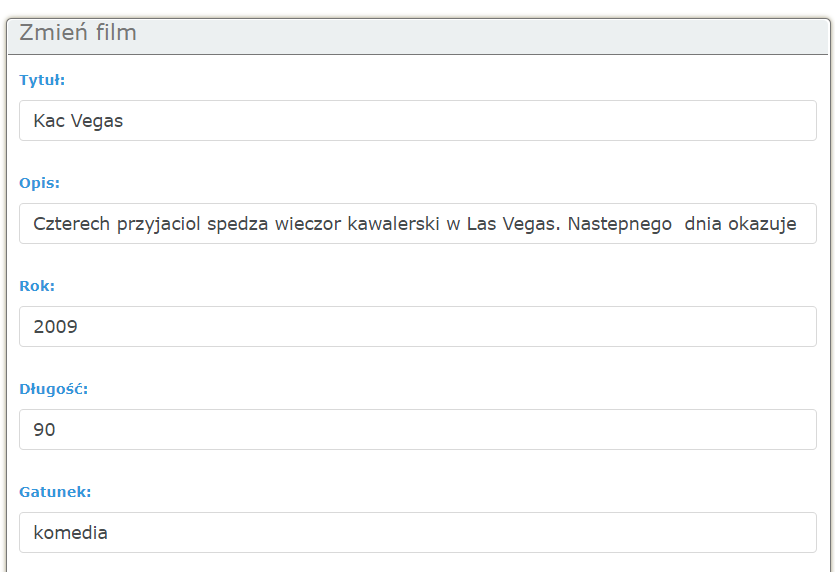
Dodanie filmu odbywa się za pomocą przycisku „**+**” :



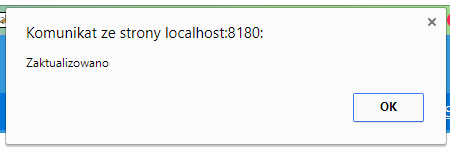
Po wciśnięciu przycisku dodaj na stronie pojawi się komunikat:



**Edycja:**



Po wciśnięciu przycisku pojawi się poniższy komunikat.



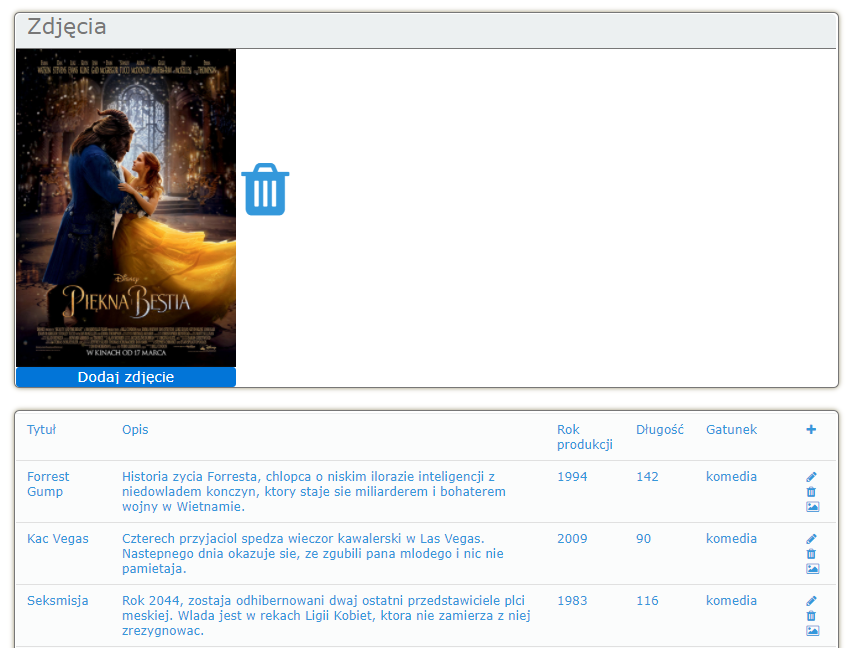
**Usuwanie:**

Odbywa się automatycznie po kliknięciu na drugi przycisk od góry. Pierwszy służy do edycji. Ostatni zaś do dodania zdjęć.



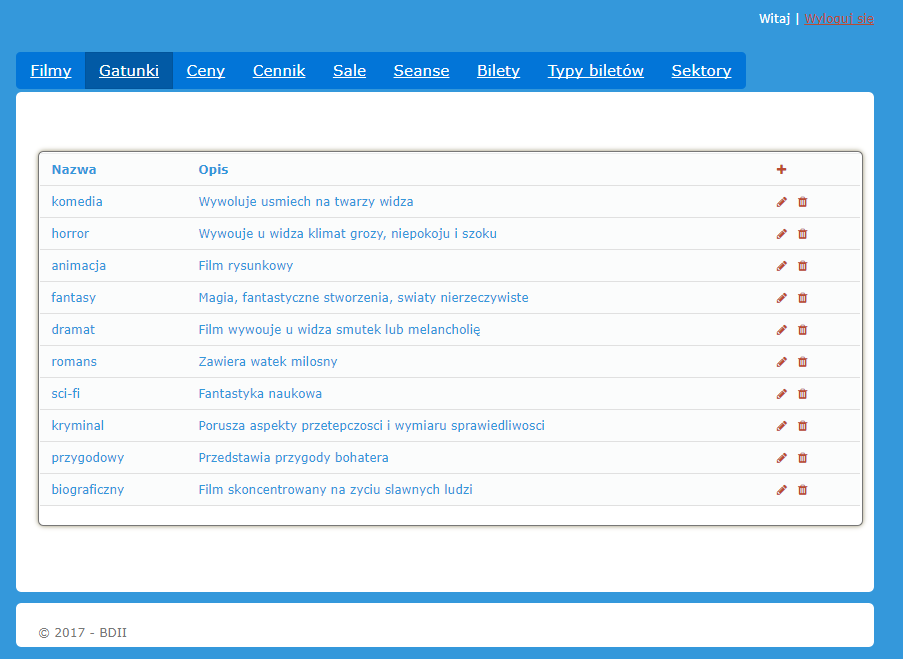
**Dodawanie zdjęć:**

Zdjęcia dodajemy za pomocą przycisku  .



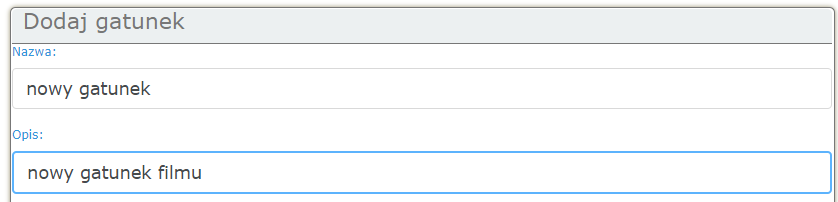
**3.3. Gatunki**

Zakładka „Gatunki” służy do zarządzania gatunkami filmów.



**Dodawanie:**

Za pomocą przycisku **„+”.**



**Usuwanie:**

Jest możliwe po wybraniu przycisku . Usuwanie odbywa się automatycznie po naciśnięciu.

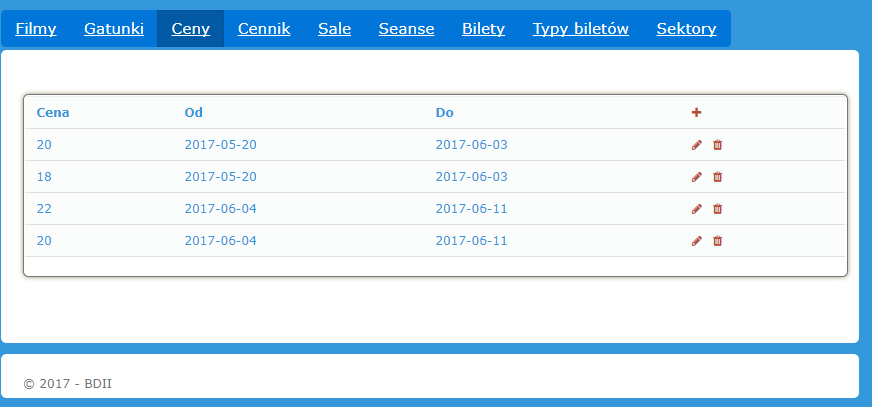
**Edycja:**

Obywa się za pomocą przycisku .

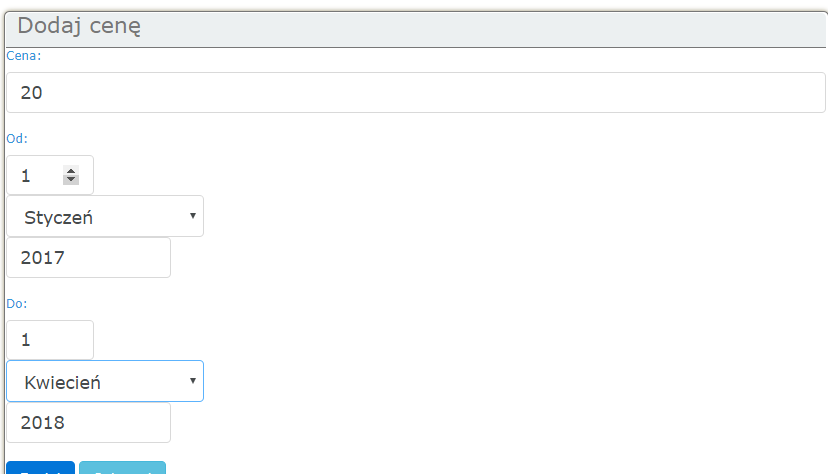


**3.3. Ceny**

Zakładka „Ceny” pozwala nam zarządzać naszymi cenami.



Za pomocą przycisku **„+”.**

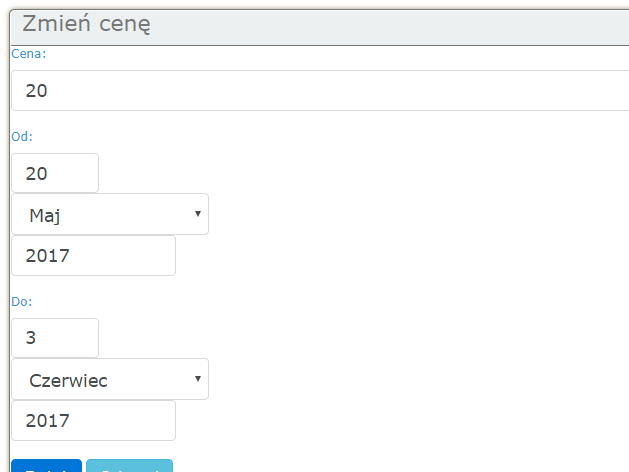


**Usuwanie:**

Jest możliwe po wybraniu przycisku . Usuwanie odbywa się automatycznie po naciśnięciu.

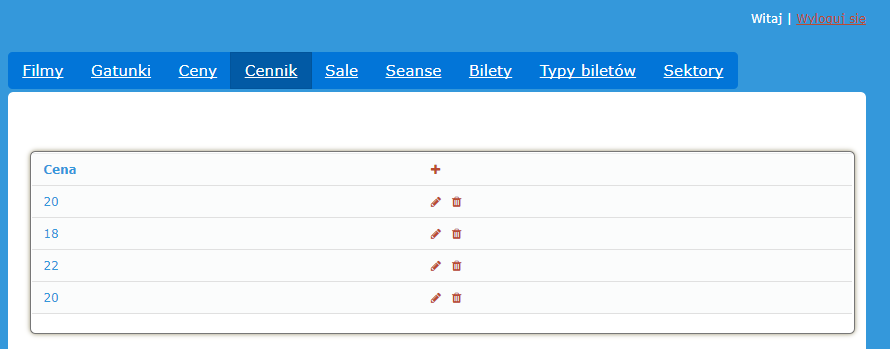
**Edycja:**

Obywa się za pomocą przycisku .



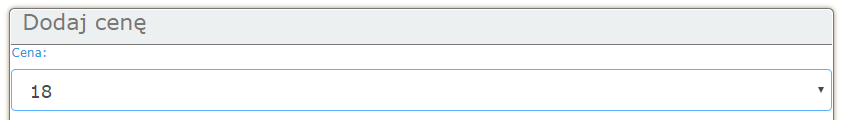
**3.4 Cennik**

Zakładka „Cennik” służy do zarządzania cennikiem naszego kina.



**Dodawanie:**

Za pomocą przycisku **„+”.**

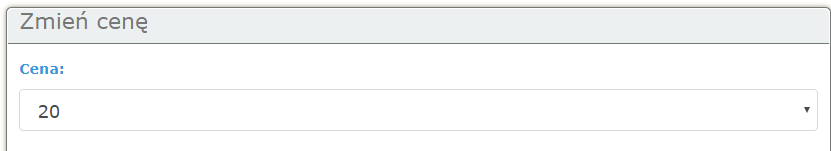


**Usuwanie:**

Jest możliwe po wybraniu przycisku . Usuwanie odbywa się automatycznie po naciśnięciu.

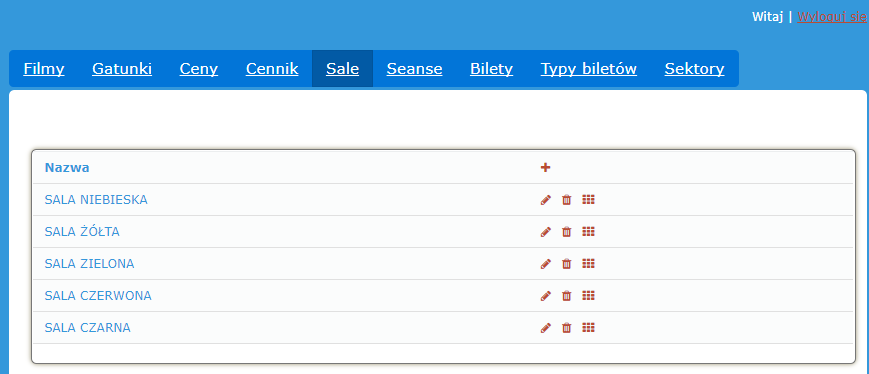
**Edycja:**

Obywa się za pomocą przycisku .



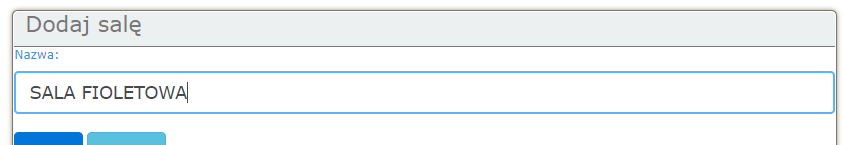
**3.5 Sale**

Zakładka „Sale” służy do zarządzania salami w naszym kinie.



**Dodawanie:**

Za pomocą przycisku **„+”.**

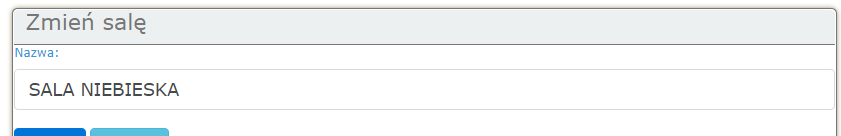


**Usuwanie:**

Jest możliwe po wybraniu przycisku . Usuwanie odbywa się automatycznie po naciśnięciu.

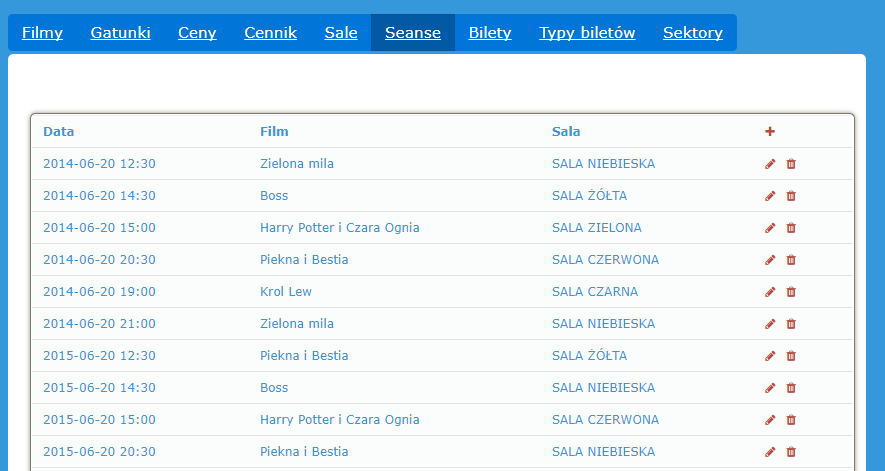
**Edycja:**

Obywa się za pomocą przycisku .



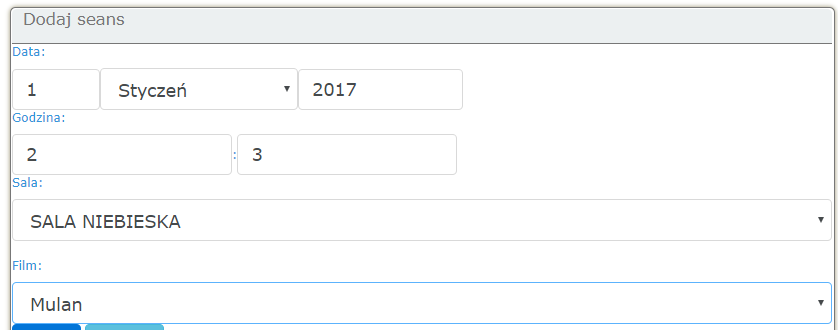
**3.6 Seanse**

Zakładka „Seanse” służy do zarządzania seansami.



**Dodawanie:**

Za pomocą przycisku **„+”.**

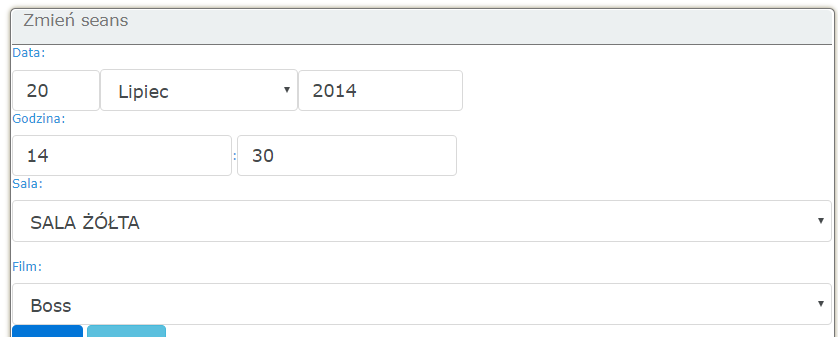


**Usuwanie:**

Jest możliwe po wybraniu przycisku . Usuwanie odbywa się automatycznie po naciśnięciu.

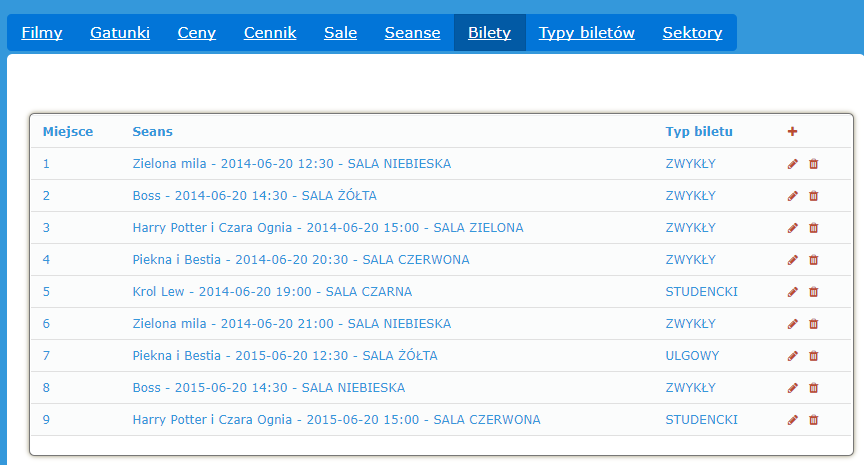
**Edycja:**

Obywa się za pomocą przycisku .



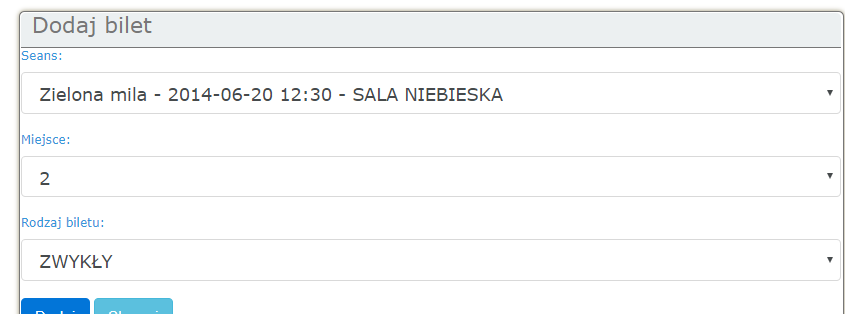
**3.7 Bilety**

Zakładka „Bilety” służy do zarządzania biletami.



**Dodawanie:**

Za pomocą przycisku **„+”.**

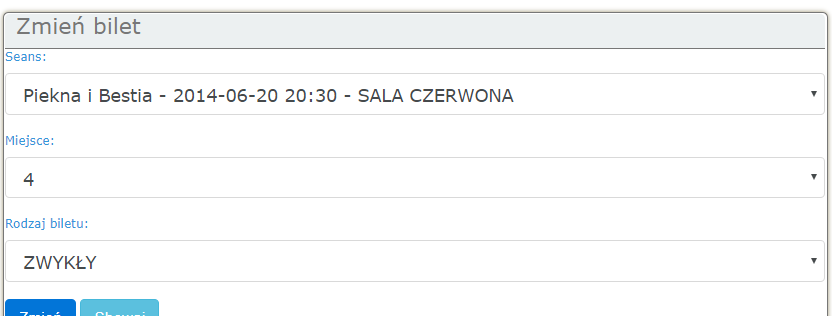


**Usuwanie:**

Jest możliwe po wybraniu przycisku . Usuwanie odbywa się automatycznie po naciśnięciu.

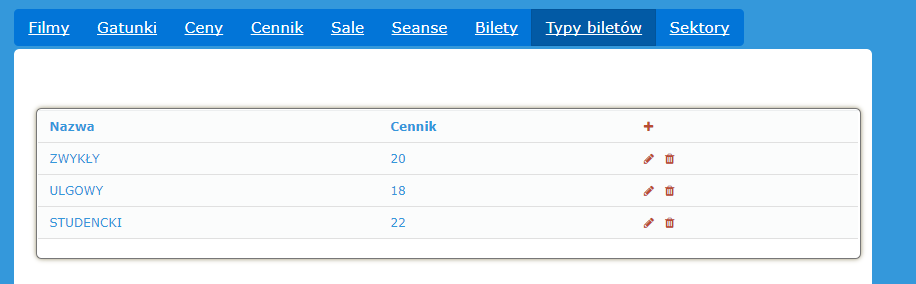
**Edycja:**

Obywa się za pomocą przycisku .



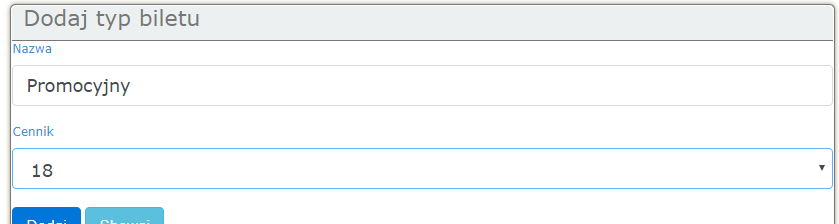
**3.8 Typy biletów**

Zakładka „Typy biletów” służy do zarządzania typami biletów.



**Dodawanie:**

Za pomocą przycisku **„+”.**

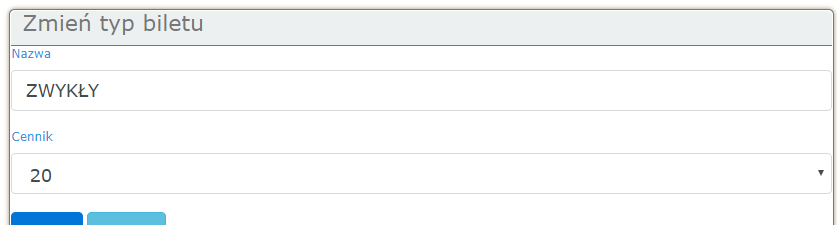


**Usuwanie:**

Jest możliwe po wybraniu przycisku . Usuwanie odbywa się automatycznie po naciśnięciu.

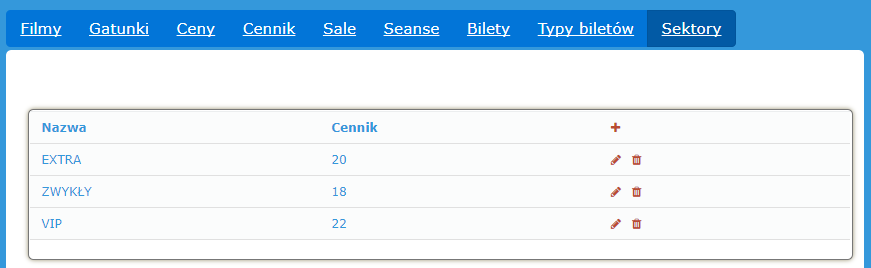
**Edycja:**

Obywa się za pomocą przycisku .



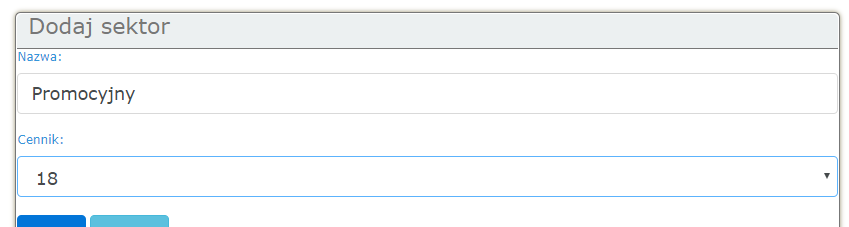
**3.9 Sektory**

Zakładka „Sektory” służy do zarządzania sektorami.



**Dodawanie:**

Za pomocą przycisku **„+”.**



**Usuwanie:**

Jest możliwe po wybraniu przycisku . Usuwanie odbywa się automatycznie po naciśnięciu.

**Edycja:**

Obywa się za pomocą przycisku .



**4. Specyfikacja wewnętrzna**

4.1 Baza

Nasza aplikacja opiera się na schemacie dołączonym w analizie.

Tabele w naszej bazie to:  
- BILET  
- MIEJSCE  
- TYP\_BILETU  
- SEKTOR  
- SEANS  
- ZDJECIA  
- FILM  
- GATUNEK  
- CENNIK  
- CENY  
- SALA

Skrypt tworzący bazę dostępny jest pod adresem: <https://github.com/SebastianOprzedek/BDII_Kino/blob/master/skrypty/STW%C3%93RZ_LUB_ZAKTUALIZUJ_BAZE.sql>

4.2 Backend

W naszym projekcie mamy pakiet **pl.bd.kino.etities**, który zawiera klasy mapujące tabele bazy danych oraz pakiet **pl.bd.kino.lists** ułatwiający operacje na nich. Następnie mamy **pakiet.pl.bd.kino.ejb** zawierający metody operujące na bazie danych oraz **pakiet.pl.bd.kino.rest**, który zawiera metody restowe umożliwiające operowanie przez klienta na danych z aplikacji.

4.3 API

Restowe API w naszym projekcie to metoda komunikowania się frontendu z backendem.

## Klasa FilmREST.java

**metoda: POST  
path: /cinema/rest/film**

*Metoda dodająca filmy. Jako zawartość przekazuje obiekt klasy Film. Klasa Film zawiera pola:*

* *id – identyfikator w tabeli*
* *description – krótki opis filmu*
* *title – nazwa filmu*
* *production\_year – rok produkcji filmu*
* *length – długość filmu w minutach*
* *genre – obiekt klasy Genre (przyporządkowanie filmu do gatunku)*
* *photo – obiekt klasy Photo (lista zdjęć powiązana z filmem)*

*Zwraca Stringa informującego o powodzeniu wykonanej operacji.*

**metoda: GET**

**path: /cinema/rest/film/{id}**

*Metoda wyszukująca filmy. Zwraca obiekt klasy Film o podanym id.*

**metoda: GET**

**path: /cinema/rest/film**

*Metoda zwracająca listę filmów. Zwraca obiekt klasy Films.*

**metoda: PUT**

**path: /cinema/rest/film/{id}**

*Metoda aktualizująca film o podanym id. Jako zawartość przekazuje obiekt klasy Film. W zależności od powodzenia lub niepowodzenia wykonania operacji, zwraca odpowiedni informujący o tym String.*

**metoda: DELETE**

**path: /cinema/rest/film/{id}**

*Metoda usuwająca film o podanym id. Nie zwraca żadnej wartości.*

## Klasa GenreREST.java

**metoda: POST  
path: /cinema/rest/genre**

*Metoda dodająca gatunek. Jako zawartość przekazuje obiekt klasy Genre. Klasa Genre zawiera pola:*

* *id – identyfikator w tabeli*
* *name – nazwa gatunku*
* *description – krótki opis gatunku*

*Zwraca Stringa informującego o powodzeniu wykonanej operacji.*

**metoda: GET**

**path: /cinema/rest/genre/{id}**

*Metoda wyszukująca gatunki. Zwraca obiekt klasy Genre o podanym id.*

**metoda: GET**

**path: /cinema/rest/genre**

*Metoda zwracająca listę gatunków. Zwraca obiekt klasy Genres.*

**metoda: PUT**

**path: /cinema/rest/genre/{id}**

*Metoda aktualizująca gatunek o podanym id. Jako zawartość przekazuje obiekt klasy Genre. W zależności od powodzenia lub niepowodzenia wykonania operacji, zwraca odpowiedni informujący o tym String.*

**metoda: DELETE**

**path: /cinema/rest/genre/{id}**

*Metoda usuwająca gatunek o podanym id. Nie zwraca żadnej wartości.*

## Klasa HallREST.java

**metoda: POST  
path: /cinema/rest/hall**

*Metoda dodająca salę. Jako zawartość przekazuje obiekt klasy Hall. Klasa Hall zawiera pola:*

* *id – identyfikator w tabeli*
* *name – nazwa sali*

*Zwraca Stringa informującego o powodzeniu wykonanej operacji.*

**metoda: GET**

**path: /cinema/rest/hall/{id}**

*Metoda wyszukująca sale. Zwraca obiekt klasy Hall o podanym id.*

**metoda: GET**

**path: /cinema/rest/hall**

*Metoda zwracająca listę sal. Zwraca obiekt klasy Halls.*

**metoda: PUT**

**path: /cinema/rest/hall/{id}**

*Metoda aktualizująca salę o podanym id. Jako zawartość przekazuje obiekt klasy Hall. W zależności od powodzenia lub niepowodzenia wykonania operacji, zwraca odpowiedni informujący o tym String.*

**metoda: DELETE**

**path: /cinema/rest/hall/{id}**

*Metoda usuwająca salę o podanym id. Nie zwraca żadnej wartości.*

## Klasa PhotoREST.java

**metoda: POST  
path: /cinema/rest/photo/{id}**

*Metoda dodająca zdjęcie. Jako zawartość przekazuje obiekt klasy String reprezentujący zdjęcie zakodowane w Base64. Klasa Hall zawiera pola:*

* *idc – identyfikator w tabeli*
* *photo – zdjęcie w formacie blob*
* *film – obiekt klasy Film (film, którego dotyczy dodawane zdjęcie)*

*Zwraca pustego Stringa informującego o powodzeniu wykonanej operacji lub ‘’error’’, w przypadku jej niepowodzenia.*

**metoda: GET**

**path: /cinema/rest/photo/{id}**

*Metoda wyszukująca zdjęcie o podanym id. Zwraca String reprezentujący wyszukane zdjęcie zakodowane w Base64.*

**metoda: GET**

**path: /cinema/rest/photo**

*Metoda zwracająca listę zdjęć. Zwraca obiekt klasy Photos.*

**metoda: PUT**

**path: /cinema/rest/photo/{id}**

*Metoda aktualizująca zdjęcie o podanym id. Jako zawartość przekazuje obiekt klasy Photo. W zależności od powodzenia lub niepowodzenia wykonania operacji, zwraca odpowiedni informujący o tym String.*

**metoda: DELETE**

**path: /cinema/rest/photo/{id}**

*Metoda usuwająca zdjęcie o podanym id. Nie zwraca żadnej wartości.*

## Klasa PlacesREST.java

**metoda: POST/{id}  
path: /cinema/rest/place**

*Metoda akutalizująca miejsca w Sali. Jako argument przyjmuje obiekt Places, czyli listę miejsc.*

**metoda: GET**

**path: /cinema/rest/place/{id}**

*Metoda zwarcająca wszystkie miejsca z Sali o podanym id.*

**metoda: GET**

**path: /cinema/rest/place/free/{id}**

*Metoda zwarcająca wolne miejsca z Sali o podanym id.*

## Klasa PricelistREST.java

**metoda: POST  
path: /cinema/rest/pricelist**

*Metoda dodająca cennik. Jako zawartość przekazuje obiekt klasy Pricelist.*

*Zwraca Stringa informującego o powodzeniu wykonanej operacji.*

**metoda: GET**

**path: /cinema/rest/ pricelist /{id}**

*Metoda wyszukująca cennik. Zwraca obiekt klasy Pricelist o podanym id.*

**metoda: GET**

**path: /cinema/rest/ pricelist**

*Metoda zwracająca listę cennik. Zwraca obiekt klasy Pricelist.*

**metoda: PUT**

**path: /cinema/rest/ pricelist /{id}**

*Metoda aktualizująca cennik o podanym id. Jako zawartość przekazuje obiekt klasy Pricelist. W zależności od powodzenia lub niepowodzenia wykonania operacji, zwraca odpowiedni informujący o tym String.*

**metoda: DELETE**

**path: /cinema/rest/ pricelist /{id}**

*Metoda usuwająca cennik o podanym id. Nie zwraca żadnej wartości.*

## Klasa PriceREST.java

**metoda: POST  
path: /cinema/rest/price**

*Metoda dodająca cenę. Jako zawartość przekazuje obiekt klasy Price.*

*Zwraca Stringa informującego o powodzeniu wykonanej operacji.*

**metoda: GET**

**path: /cinema/rest/ price /{id}**

*Metoda wyszukująca cenę. Zwraca obiekt klasy Price o podanym id.*

**metoda: GET**

**path: /cinema/rest/ price**

*Metoda zwracająca listę cen. Zwraca obiekt klasy Prices.*

**metoda: PUT**

**path: /cinema/rest/ price /{id}**

*Metoda aktualizująca cenę o podanym id. Jako zawartość przekazuje obiekt klasy Price. W zależności od powodzenia lub niepowodzenia wykonania operacji, zwraca odpowiedni informujący o tym String.*

**metoda: DELETE**

**path: /cinema/rest/ price /{id}**

*Metoda usuwająca cenę o podanym id. Nie zwraca żadnej wartości.*

## Klasa ShowREST.java

**metoda: POST  
path: /cinema/rest/show**

*Metoda dodająca seans. Jako zawartość przekazuje obiekt klasy Show. Klasa Show zawiera pola:*

* *id – identyfikator w tabeli*
* *data – godzina seansu*
* *hall – obiekt klasy Hall (sala, na której wyświetlany będzie seans)*
* *film – obiekt klasy Film (film, który wyświetlany będzie w ramach tego seansu)*

*Zwraca Stringa informującego o powodzeniu wykonanej operacji.*

**metoda: GET**

**path: /cinema/rest/ show /{id}**

*Metoda wyszukująca seanse. Zwraca obiekt klasy Show o podanym id.*

**metoda: GET**

**path: /cinema/rest/ show**

*Metoda zwracająca listę seansów. Zwraca obiekt klasy Shows.*

**metoda: PUT**

**path: /cinema/rest/ show /{id}**

*Metoda aktualizująca seans o podanym id. Jako zawartość przekazuje obiekt klasy Show. W zależności od powodzenia lub niepowodzenia wykonania operacji, zwraca odpowiedni informujący o tym String.*

**metoda: DELETE**

**path: /cinema/rest/ show /{id}**

*Metoda usuwająca seans o podanym id. Nie zwraca żadnej wartości.*

## Klasa TicketREST.java

**metoda: POST  
path: /cinema/rest/ticket**

*Metoda dodająca bilet. Jako zawartość przekazuje obiekt klasy Ticket.*

*Zwraca Stringa informującego o powodzeniu wykonanej operacji.*

**metoda: GET**

**path: /cinema/rest/ ticket /{id}**

*Metoda wyszukująca bilety. Zwraca obiekt klasy Ticket o podanym id.*

**metoda: GET**

**path: /cinema/rest/ ticket**

*Metoda zwracająca listę biletów. Zwraca obiekt klasy Tickets.*

**metoda: PUT**

**path: /cinema/rest/ ticket /{id}**

*Metoda aktualizująca bilet o podanym id. Jako zawartość przekazuje obiekt klasy Ticket. W zależności od powodzenia lub niepowodzenia wykonania operacji, zwraca odpowiedni informujący o tym String.*

**metoda: DELETE**

**path: /cinema/rest/ ticket /{id}**

*Metoda usuwająca bilet o podanym id. Nie zwraca żadnej wartości.*

## Klasa TicketTypeREST.java

**metoda: POST  
path: /cinema/rest/ticket\_type**

*Metoda dodająca typ biletu. Jako zawartość przekazuje obiekt klasy TicketType.*

*Zwraca Stringa informującego o powodzeniu wykonanej operacji.*

**metoda: GET**

**path: /cinema/rest/ ticket\_type /{id}**

*Metoda wyszukująca typ biletu. Zwraca obiekt klasy TicketType o podanym id.*

**metoda: GET**

**path: /cinema/rest/ ticket\_type**

*Metoda zwracająca listę typów biletów. Zwraca obiekt klasy TicketTypes.*

**metoda: PUT**

**path: /cinema/rest/ ticket\_type /{id}**

*Metoda aktualizująca typ biletu o podanym id. Jako zawartość przekazuje obiekt klasy TicketType. W zależności od powodzenia lub niepowodzenia wykonania operacji, zwraca odpowiedni informujący o tym String.*

**metoda: DELETE**

**path: /cinema/rest/ ticket\_type /{id}**

*Metoda usuwająca typ biletu o podanym id. Nie zwraca żadnej wartości.*

4.4 Frontend

Każda formatka jest reprezentowana przez parę plików .js i .html.  
Duża część kodu w plikach .js powiela się, jednak uznaliśmy, że zachowamy taki podział, ponieważ zmiana tego wymagała by zbyt dużo pracy.  
Frontend korzysta z API wystawionego przez backend.  
Zdecydowaliśmy się nie korzystać z żadnego frameworku, jednak z perspektywy czasu był to zły pomysł, ponieważ pisanie takiej ilości kodu w czystym JS powoduje, że jest on mało czytelny.

Aby poprawić wygląd aplikacji korzystamy z boostrapa.

5. Wnioski

Po początkowych problemach z implementacją powiązań w bazie danych (głównie OneToMany i ManyToMany), dzięki pomocy Hibernate'a zostały one rozwiązane. W projekcie nie został użyty żaden z popularnych frameworków (Angular, React czy VueJs), gdyż na początku implementacji sądziliśmy, że frontend aplikacji nie będzie skomplikowany. Mimo trudności w pisaniu kodu korzystając z czystego JavaScriptu uważamy, że wykonana praca nie poszła na marne. Popularność frameworków się zmienia, a JavaScript ciągle jest używany, co pomoże nam płynniej przejść przez zmiany na rynku. Projekt pozwolił nam rozwinąć wiedzę na temat tworzenia aplikacji typu enterprise, a początkowo obce aplikacje restowe wydają się być teraz bardzo proste w tworzeniu dzięki poznanym przez nas frameworkom Javy. Wykonany projekt spełnia założenia.