

POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

PROJEKT Z BAZ DANYCH

Baza danych kina

Termin zajęć: czwartek, 11:15–13:00

AUTOR/AUTORZY:

Jan Bronicki

Marcin Radke

Maciej Marczyszyn

PROWADZĄCY ZAJĘCIA:

dr inż. Konrad Kluwak

Wrocław, 2021 r.

Spis treści:

1. Wstęp	4
1.1. Cel projektu.....	4
1.2. Zakres projektu.....	4
2. Analiza wymagań	4
2.1. Opis działania i schemat logiczny systemu.....	4
2.2. Wymagania funkcjonalne.....	4
2.3. Wymagania нефункционалне.....	4
2.3.1. Wykorzystywane technologie i narzędzia.....	4
2.3.2. Wymagania dotyczące rozmiaru bazy danych.....	4
2.3.3. Wymagania dotyczące bezpieczeństwa systemu.....	4
2.4. Przyjęte założenia projektowe.....	4
3. Projekt systemu	4
3.1. Projekt bazy danych.....	4
3.1.1. Analiza rzeczywistości i uproszczony model konceptualny.....	4
3.1.2. Model logiczny i normalizacja.....	4
3.1.3. Model fizyczny i ograniczenia integralności danych.....	4
3.1.4. Inne elementy schematu – mechanizmy przetwarzania danych.....	4
3.1.5. Projekt mechanizmów bezpieczeństwa na poziomie bazy danych.....	4
3.2. Projekt aplikacji użytkownika.....	4
3.2.1. Architektura aplikacji i diagramy projektowe.....	4
3.2.2. Interfejs graficzny i struktura menu.....	4
3.2.3. Projekt wybranych funkcji systemu.....	4
3.2.4. Metoda podłączania do bazy danych – integracja z bazą danych.....	4

3.2.5. Projekt zabezpieczeń na poziomie aplikacji.....	4
4. Implementacja systemu baz danych	4
4.1. Tworzenie tabel i definiowanie ograniczeń.....	4
4.2. Implementacja mechanizmów przetwarzania danych.....	5
4.3. Implementacja uprawnień i innych zabezpieczeń.....	5
4.4. Testowanie bazy danych na przykładowych danych.....	5
5. Implementacja i testy aplikacji	5
5.1. Instalacja i konfigurowanie systemu.....	5
5.2. Instrukcja użytkowania aplikacji.....	5
5.3. Testowanie opracowanych funkcji systemu.....	5
5.4. Omówienie wybranych rozwiązań programistycznych.....	5
5.4.1. Implementacja interfejsu dostępu do bazy danych.....	5
5.4.2. Implementacja wybranych funkcjonalności systemu.....	5
5.4.3. Implementacja mechanizmów bezpieczeństwa.....	5
6. Podsumowanie i wnioski	5
Literatura	5
Spis rysunków	5
Spis tabel	5

1. Wstęp

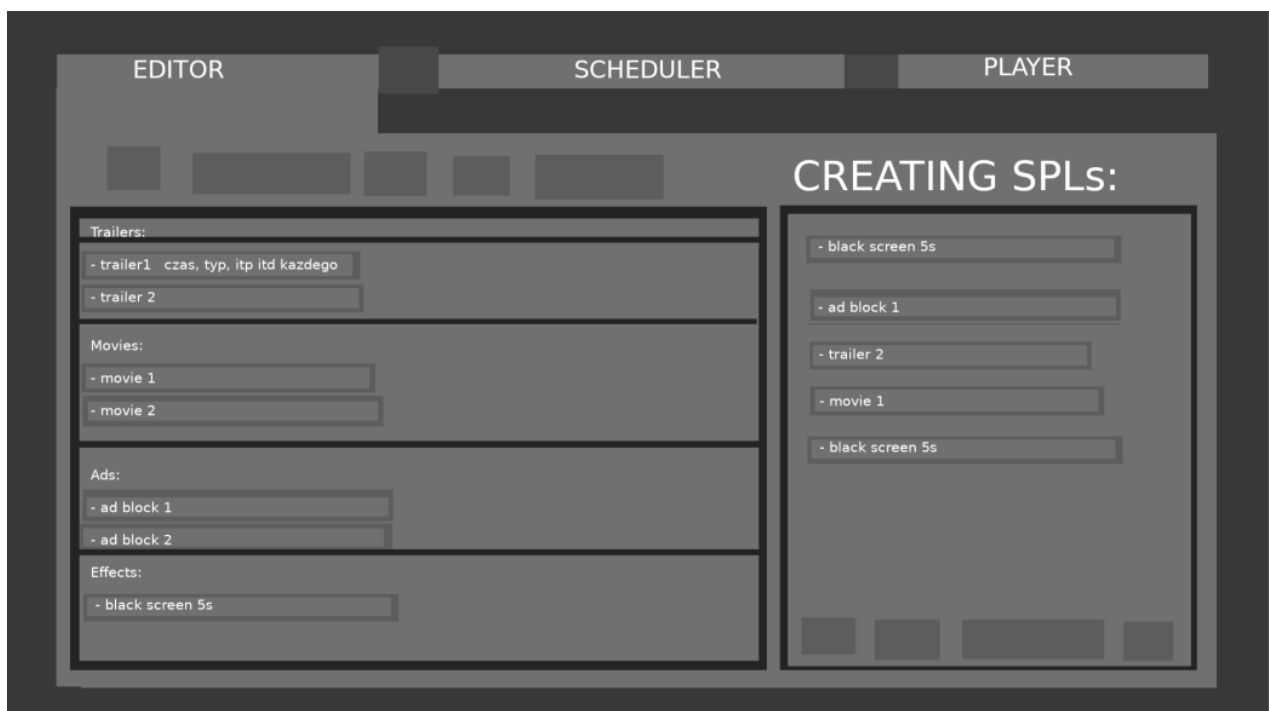
.1.1. Cel projektu

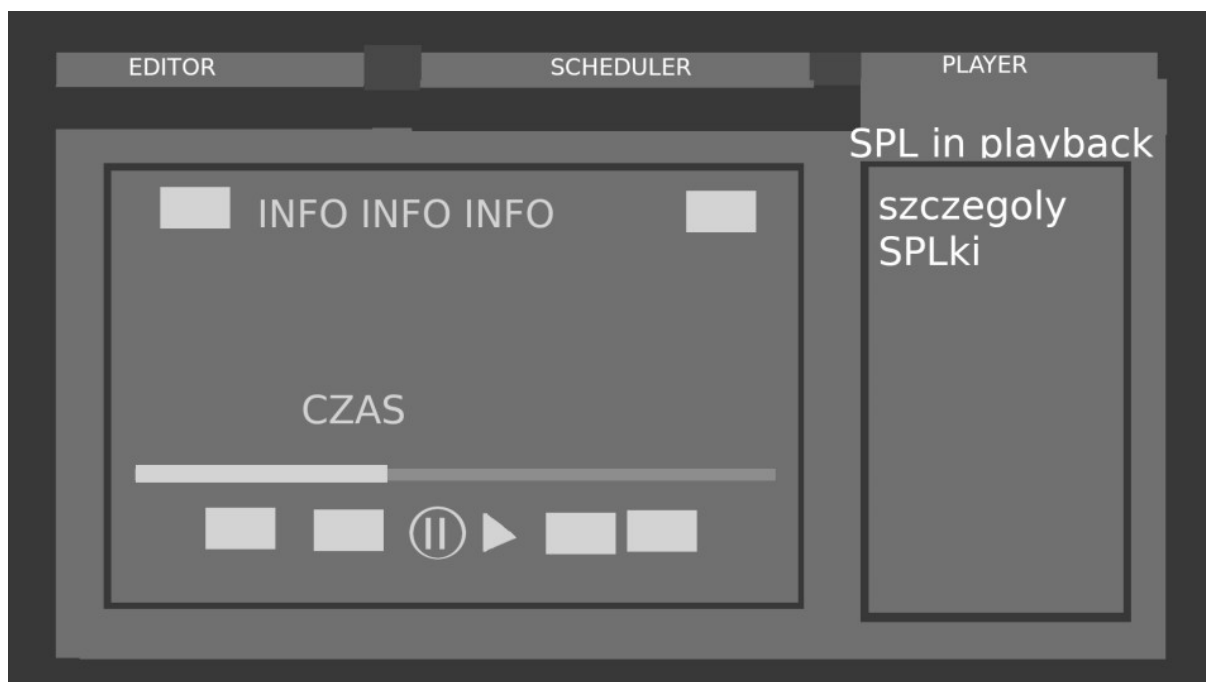
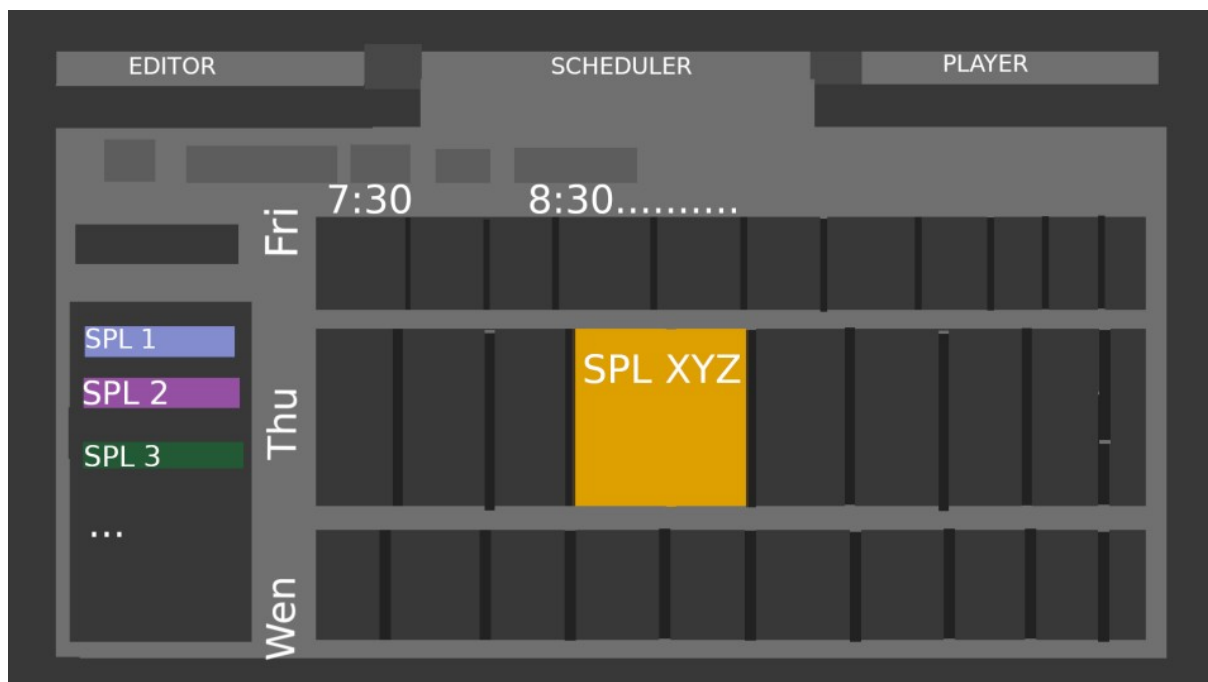
.1.2. Zakres projektu

2. Analiza wymagań

Baza danych dla operatorów kin. Operator ma możliwość tworzenia oraz planowania reperturu jaki wyświetlany jest w poszczególnych salach oraz zarządzania hardwarem, konfiguracją oraz serwisem kina w dowolnej porze dnia.

.2.1. Opis działania i schemat logiczny systemu





.2.2. Wymagania funkcjonalne

- Możliwość injusteowania contentu i jego transkrypcji
- Możliwość tworzenia własnych SPL (Show Playlist)
- Możliwość tworzenia repertuaru
- Możliwość zarządzania hardwarem i jego konfiguracją
- Możliwość zarządzania serwisem

- Usuwanie konkretnych pozycji z bazy danych
- Tworzenie raportów (np. który film był najbardziej oglądany)

.2.3. Wymagania niefunkcjonalne

- Baza : relacyjna
- Interfejs graficzny

.2.3.1. Wykorzystywane technologie i narzędzia

- Django (backend)
- SQLite

.

.2.3.2. Wymagania dotyczące bezpieczeństwa systemu

- Zabezpieczenie Kdm przed ich przedwczesnym użyciem
- Zabezpieczenie dostępu do bazy danych przed osobami postronnymi

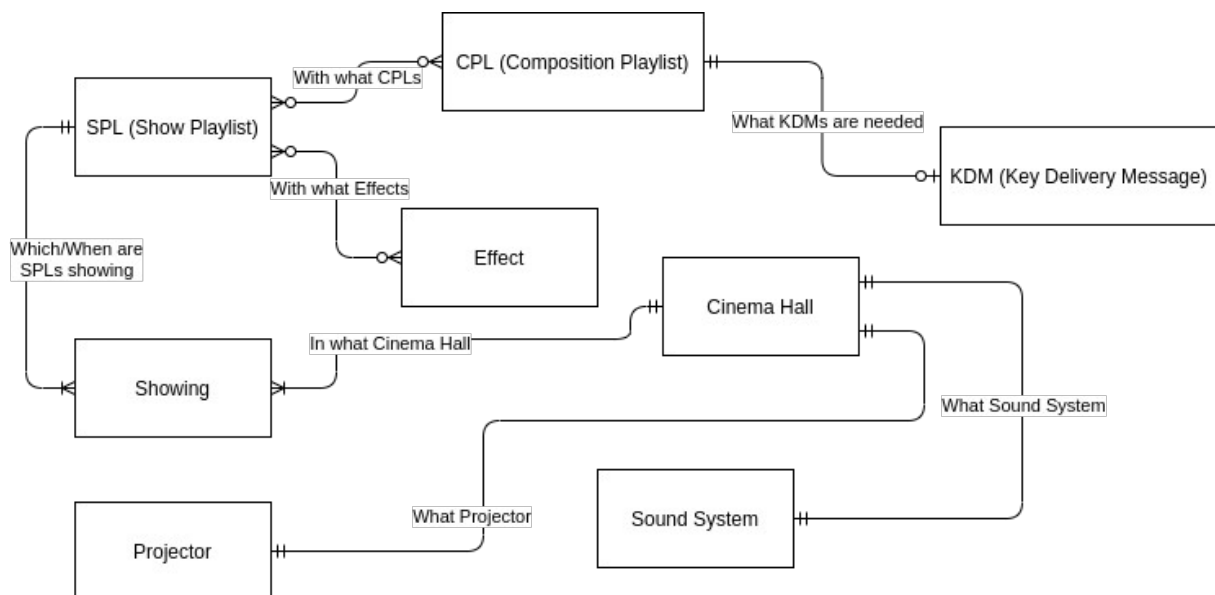
.

3. Projekt systemu

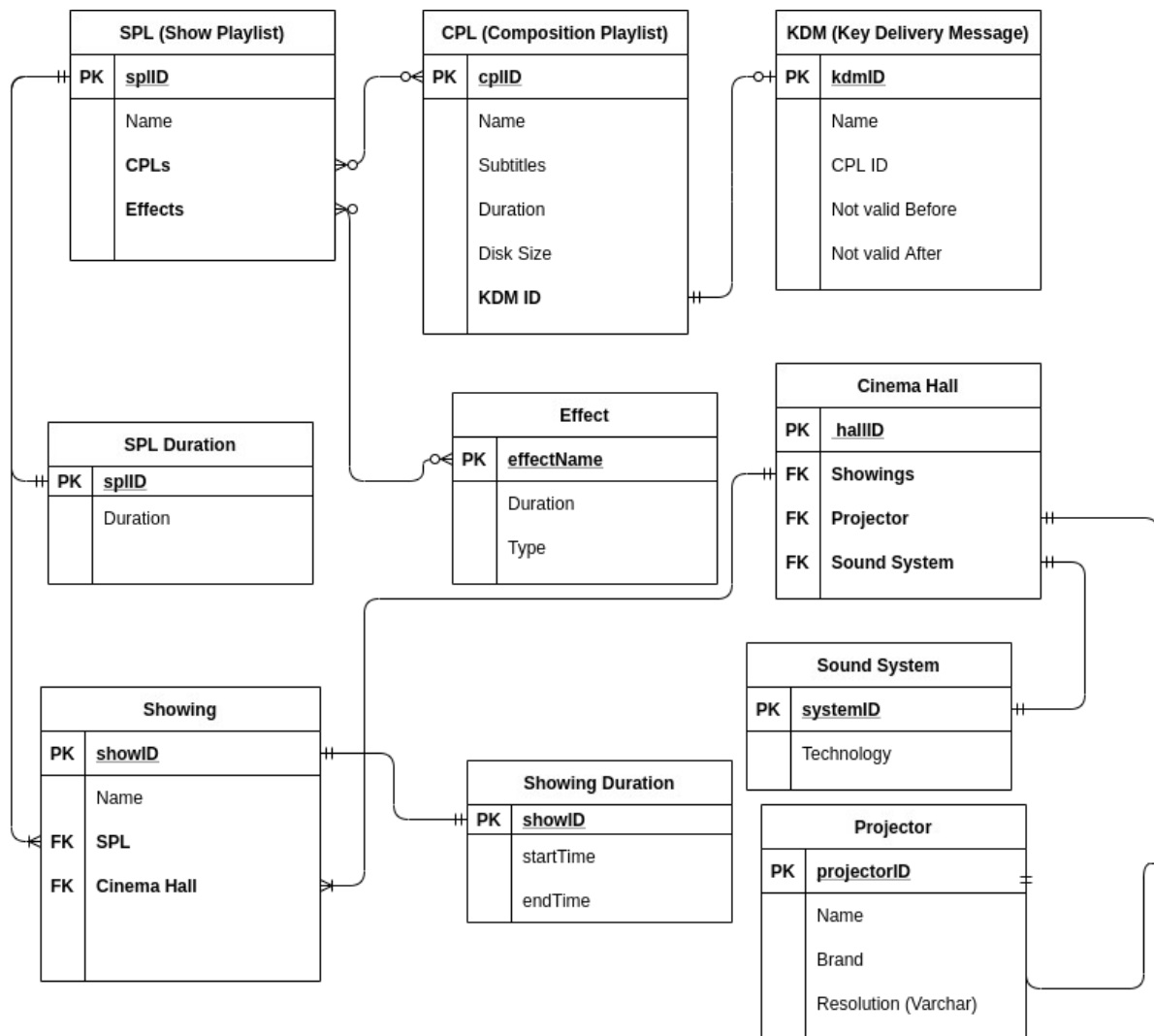
Projekt i struktury bazy danych, mechanizmów zapewniania poprawności przechowywanych informacji, oraz kontroli dostępu do danych.

.3.1. Projekt bazy danych

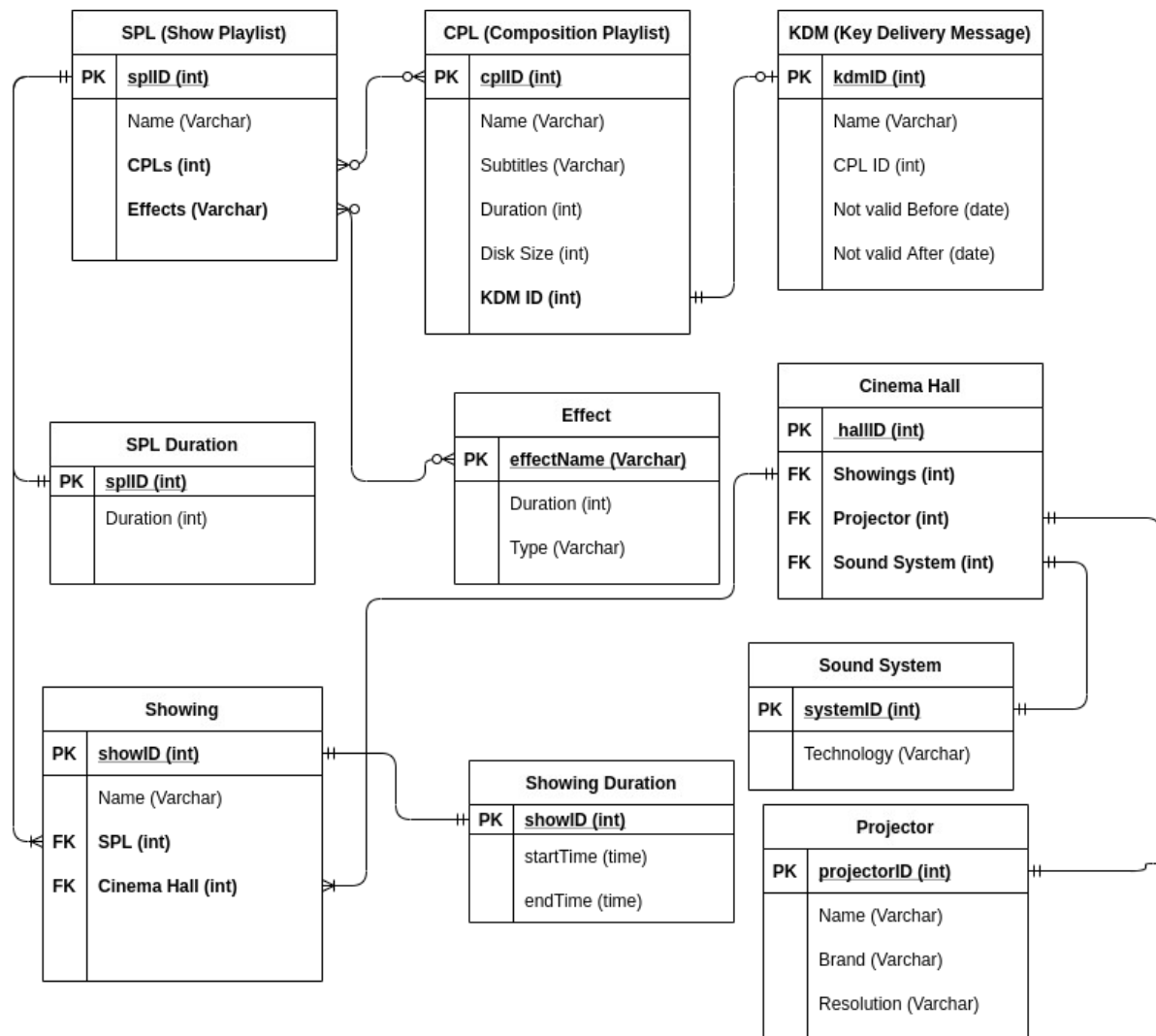
.3.1.1. Analiza rzeczywistości i uproszczony model konceptualny



.3.1.2. Model logiczny i normalizacja



3.1.3. Model fizyczny i ograniczenia integralności danych



.3.1.4. Inne elementy schematu - mechanizmy przetwarzania danych

Mechanizmy przetwarzania danych, które będą zaimplementowane w bazie danych:

- Dodanie kolejnego CPL będzie wiązało się z możliwością dodanie nowego elementu w encji KDM. Próba wyświetlenia zawartości CPL zwróci nam jej ID, nazwę, napisy, ID Kdm do której jest przypisana, czas trwania oraz rozmiar. Próba wyświetlenia KDM zwróci nam jej ID, nazwę, ID cpl do której jest przypisana oraz okres aktywności
- Dodanie CPL lub Effectu do encji SPL automatycznie wydłuży nam czas trwania SPL znajdujący się w tabeli SPL duration. Próba wyświetlenia SPL wyświetli jej ID, ID CPL znajdujących się w niej oraz nazwy efektów. Próba wyświetlenia Efektów wyświetli ich nazwę, czas trwania oraz typ

- Stworzenie nowego Show będzie wiązało się z dodaniem do niego przynajmniej jednej sali oraz określenia godziny rozpoczęcia wyświetlania się Show. Dodatkowo w tabeli Show Times będzie obliczany, na podstawie zadanego czasu rozpoczęcia oraz długości trwania SPL, czas zakończenia się wyświetlania show.
- Dodanie nowej sali do tabeli będzie wiązało się z dodaniem do niej systemu dźwięku oraz projektora, oraz show. Próba wyświetlenia Sali Kinowej wyświetli jej Numer, ID projektora sound systemu oraz Show której jest w niej wyświetlane.
- Dodanie Projektora wiąże się z dodaniem jego marki, ID, rozdzielczości i nazwy.
- Dodanie Sound Systemu wiąże się z dodaniem jego ID i nazwy

.3.1.5. Projekt mechanizmów bezpieczeństwa na poziomie bazy danych

Zostanie dodany system zarządzania użytkownikami. Będą wyodrębnione 2 typy użytkowników : Admin oraz Operator, każdy typ użytkownika będzie się musiał zalogować aby korzystać w dostępnych uprawnieniach. Operator będzie miał dostęp do dodawania sprzętu i tworzenia nowych Shows i wyznaczania ich czasu odtwarzania na poszczególnych salach. Administrator będzie miał dodatkowo możliwość oddawania CPL oraz ich KDM i efektów.

.3.2. Projekt aplikacji użytkownika

.3.2.1. Architektura aplikacji i diagramy projektowe

.3.2.2. Interfejs graficzny i struktura menu

.3.2.3. Projekt wybranych funkcji systemu

.3.2.4. Metoda podłączania do bazy danych - integracja z bazą danych

.3.2.5. Projekt zabezpieczeń na poziomie aplikacji

4. Implementacja systemu baz danych

Implementacja i testy bazy danych w wybranym systemie zarządzania bazą danych.

.4.1. Tworzenie tabel i definiowanie ograniczeń

Cała baza danych została zaimplementowana w django. Dla każdej encji stworzono jej niezbędne pola i ich ograniczenia (np. w postaci maksymalnej liczby znaków) oraz metodę wyświetlania ich za pomocą nazw obiektów. Stworzono niezbędne relacje one to one oraz dla wybranych encji (Show, SPL i CinemaHall) relacje many to many. Dodatkowo wykorzystano bazę użytkowników automatycznie stworzoną przez django.

```

class KDM(models.Model):
    name = models.CharField(max_length=200)
    not_valid_before = models.DateTimeField(auto_now_add=False, auto_now=False, blank=True, default=now)
    not_valid_after = models.DateTimeField(auto_now_add=False, auto_now=False, blank=True, default=now)
    KDMKey = models.IntegerField(null=False)

    def __str__(self):
        return self.name

class CPL(models.Model):
    name = models.CharField(max_length=200)
    subtitles = models.CharField(max_length=200)
    duration = models.IntegerField(null=False)
    disk_size = models.IntegerField(null=False)
    KDM = models.ForeignKey(KDM, blank=True, null=True, on_delete=models.SET_NULL)

    def __str__(self):
        return self.name

class Effect(models.Model):
    name = models.CharField(max_length=200)
    type = models.CharField(max_length=200)
    duration = models.IntegerField(null=False)

    def __str__(self):
        return self.name

class SPL(models.Model):
    name = models.CharField(max_length=200)
    Effects = models.ManyToManyField(Effect, blank=True)
    CPLs = models.ManyToManyField(CPL, blank=True)

    def __str__(self):
        return self.name

```

```

class Projector(models.Model):
    name = models.CharField(max_length=200)
    brand = models.CharField(max_length=200)
    resolution = models.CharField(max_length=200)

    def __str__(self):
        return self.name

class SoundSystem(models.Model):
    name = models.CharField(max_length=200)
    technology = models.CharField(max_length=200)

    def __str__(self):
        return self.name

```

```

class Show(models.Model):
    name = models.CharField(max_length=200)
    SPLs = models.ManyToManyField(SPL)
    cinema_hall = models.ForeignKey("CinemaHall", blank=True, null=True, on_delete=models.SET_NULL)
    start_date = models.DateTimeField(auto_now_add=False, auto_now=False, default=now)

    def __str__(self):
        return self.name

class CinemaHall(models.Model):
    name = models.CharField(max_length=200)
    showings = models.ManyToManyField(Show, blank=True)
    projector = models.ForeignKey(Projector, blank=True, null=True, on_delete=models.SET_NULL)
    sound_system = models.ForeignKey(SoundSystem, blank=True, null=True, on_delete=models.SET_NULL)

    def __str__(self):
        return self.name

```

.4.2. Implementacja mechanizmów przetwarzania danych

Mechanizmy przetwarzania danych implementowane są głównie w pliku views.py który odpowiada za podłączanie bazy danych do aplikacji. Dla każdej encji wykorzystuje się zapytania query odpowiadające za obsługę wszystkich obiektów danej encji, służy to do wypisywania wszystkich parametrów danego obiektu. Dodatkowo dla Show i SPL stworzono zapytanie query automatycznie obliczające czas trwania SPL oraz date zakończenia Show. Dla każdej encji stworzono opcje tworzenia i usuwania wybranych obiektów szukanych po ID oraz dla encji SPL, Show oraz CinemaHall stworzono opcje bardziej szczegółowego podglądu. Bardziej złożone encje mają również możliwość modyfikacji zawartości.

```

@login_required(login_url='login')
def shows(request):
    shows_list = Show.objects.all()
    end_date_list = {}
    for show in shows_list:
        duration = 0
        for spl in show.SPLs.all():
            for cpl in spl.CPLs.all():
                duration = duration + cpl.duration
            for effect in spl.Effects.all():
                duration = duration + effect.duration
        end_date_list[show.id] = show.start_date + timedelta(seconds=duration)
    content = {'shows_list': shows_list, 'end_date_time': end_date_list}
    return render(request, 'CinemaBase/shows.html', content)

```

```
@login_required(login_url='login')
def soundsystems(request):
    soundsystem_list = SoundSystem.objects.all()

    content = {'soundsystem_list': soundsystem_list}
    return render(request, 'CinemaBase/soundsystems.html', content)
```

.4.3. Implementacja uprawnień i innych zabezpieczeń

Zabezpieczenia ze strony bazy danych obejmują zabezpieczenie przed pojawieniem się błędu podczas usuwania encji która znajduje się jako parametr innej. Zabezpieczenie to jest wykonane za pomocą zmiennej `on_delete` danego parametru i ustawiona jest ona na wartość `model.SET_NULL` co oznacza że w miejscu gdzie dany obiekt był zgnieźdzony pojawia się `null`. Uprawnienia dla użytkowników zostały skonfigurowane po stronie aplikacji poprzez stworzenie blokady wejścia na konkretne strony odpowiadające za tworzenie, modyfikacje oraz usuwanie obiektów konkretnych encji, do tego celu stworzono 2 grupy użytkowników: `operator` (może przelądać wszystko ale modyfikować tylko `Show`, `CinemaHalls`, `Projectors` i `SoundSystems`) oraz `admin` (nieograniczony dostęp) dostęp do konkretnych możliwości ustawiono stosując odpowiednie dekoratory. Dodatkowo django wprowadza automatyczną opcję szyfrowania haseł.

```

def unauthenticated_user(view_func):
    def wrapper_func(request, *args, **kwargs):
        if request.user.is_authenticated:
            return redirect('home')
        else:
            return view_func(request, *args, **kwargs)
    return wrapper_func

def allowed_users(allowed_roles=[]):
    def decorator(view_func):
        def wrapper_func(request, *args, **kwargs):
            group = None
            if request.user.groups.exists():
                group = request.user.groups.all()[0].name

            if group in allowed_roles:
                return view_func(request, *args, **kwargs)
            else:
                return HttpResponse('You are not allowed to view this page')

        return wrapper_func

    return decorator

def admin_only(view_func):
    def wrapper_function(request, *args, **kwargs):
        group = None
        if request.user.groups.exists():
            group = request.user.groups.all()[0].name

        if group == 'admins':
            return view_func(request, *args, **kwargs)
        else:
            return redirect('home')

    return wrapper_function

```

.4.4. Testowanie bazy danych na przykładowych danych

Dla każdej encji danych dodano po kilka przykładów. Następnie tam gdzie tobyło możliwe zostały one zmodyfikowane a część usunięto. Do sprawdzenia zabezpieczeń stworzono 2 użytkowników i sprawdzono ich możliwości dostępu

5. Implementacja i testy aplikacji

Skrócone sprawozdanie z etapu implementacja i testowania aplikacji.

.5.1. Instalacja i konfigurowanie systemu

Aby skorzystać z systemu bazydanych należy uruchomic plik manage.py z parametrem runserver. Komenda ta uruchami lokalny server obsługujący baze danych.

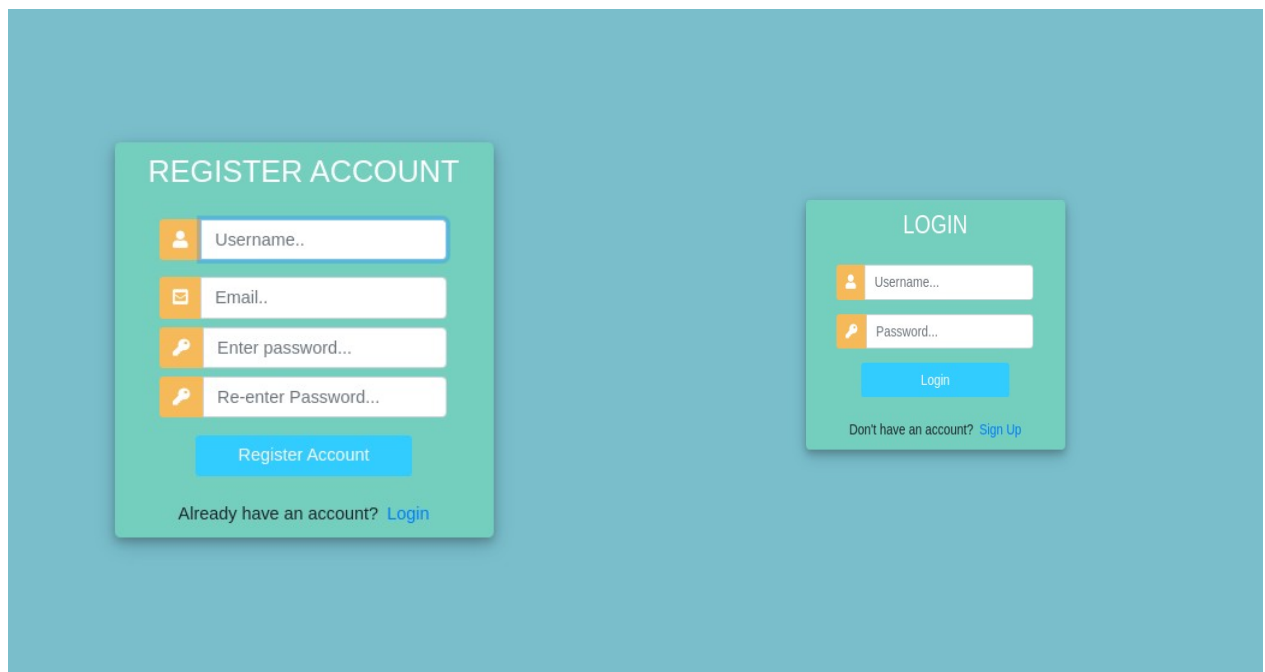
.5.2. Instrukcja użytkowania aplikacji

Aplikacja jest intuicyjna w użytkowaniu. Aby móc z niej korzystać należy zalogować się na konto, które ma dostęp na poziomie operatora lub administratora. Na samej górze strony znajduje się pasek nawigacji który przenosi uzytkownika do stron wybranych encji gdzie w zależności od uprawnień może je edytować, dodawać oraz usuwać.

Cinema DataBase	Dashboard	Shows	Cinema Halls	Sound Systems	SPLs	CPLs	KDMs	Effects	Projectors	Hello, admin1 Logout
Create Show										
	ID	Name	Start Date	End Date	Update	Remove				
View	6	Show 32	May 24, 2021, 6:56 p.m.	May 24, 2021, 9:33 p.m.	Update	Remove				
View	7	Show 2	May 26, 2021, 7:11 p.m.	April 26, 2025, 6:31 a.m.	Update	Remove				
View	8	Show 3	May 26, 2021, 7:11 p.m.	May 28, 2021, 5:24 a.m.	Update	Remove				
View	9	Show inf.	May 26, 2021, 7:12 p.m.	April 27, 2025, 9:21 p.m.	Update	Remove				

.5.3. Testowanie opracowanych funkcji systemu

Aplikacje testowano tworząc za pomocą panelu do rejestracji kilka konto o różnych poziomach dostępu oraz testowano wszystkie funkcję aplikacji.



.5.4. Omówienie wybranych rozwiązań programistycznych

.5.4.1. Implementacja interfejsu dostępu do bazy danych

Sam dostęp do bazy danych implementowany w pliku views.py gdzie tworzone są zapytania query, a następnie wysyłane są w postaci dictionary content na wybraną stronę html gdzie są w odpowiedni sposób przetwarzane.

```
@login_required(login_url='login')
@allowed_users(allowed_roles=["admins", "operators"])
def shows(request):
    shows_list = Show.objects.all()
    end_date_list = {}
    for show in shows_list:
        duration = 0
        for spl in show.SPLs.all():
            for cpl in spl.CPLs.all():
                duration = duration + cpl.duration
            for effect in spl.Effects.all():
                duration = duration + effect.duration
        end_date_list[show.id] = show.start_date + timedelta(seconds=duration)
    content = {'shows_list': shows_list, 'end_date_time': end_date_list}
    return render(request, 'CinemaBase/shows.html', content)
```


pliku views.py gdzie porównywany jest czas staru i końca show z obecnym czasem oraz obliczany jest czas pozostały do końca show. Następnie dane te są przetwarzane w pliku html strony głównej.

```
@login_required(login_url='login')
@allowed_users(allowed_roles=["admins", "operators"])
def home(request):
    cinema_list = CinemaHall.objects.all()
    current_show_list = {}
    time_left_list = {}
    now = datetime.datetime.utcnow().replace(tzinfo=utc)
    for hall in cinema_list:
        for show in hall.showings.all():
            duration = 0
            for spl in show.SPLs.all():
                for cpl in spl.CPLs.all():
                    duration = duration + cpl.duration
                for effect in spl.Effects.all():
                    duration = duration + effect.duration
            if now >= show.start_date and now <= show.start_date + timedelta(seconds=duration):
                current_show_list[hall.id] = show.name
                time_left_list[hall.id] = str(show.start_date + timedelta(seconds=duration) - now).split('.')[0]

    content = {'cinema_list': cinema_list, 'current_show_list': current_show_list, 'time_left_list': time_left_list}
    return render(request, 'CinemaBase/dashboard.html', content)
```

```
{% extends 'CinemaBase/main.html' %}
{% block content %}

<div class="col-md-12">
    <h5>Now Playing</h5>
    <hr>
    <div class="card card-body">
        <table class="table table-sm">
            <tr>
                <th>Cinema Hall </th>
                <th>Show Name</th>
                <th>Time Left</th>
            </tr>
            <tr>
                {% for hall in cinema_list %}
                <th>{{ hall.name }}</th>
                <th>{% load get_item %}
                    {{ current_show_list|get_item:hall.id }}</th>
                <th>{% load get_item %}
                    {{ time_left_list|get_item:hall.id }}</th>
                </tr>
            {% endfor %}
        </table>
    </div>
</div>

{% endblock %}
```

Opisana wcześniej możliwość dodawania usuwania i edycji obiektów w encjach została zaimplementowana przy pomocy wykorzystywanych przez django modeli forms. Stworzono

formy dla każdej encji a następnie przy ich pomocy automatycznie był tworzony formularz dodawanie, edycji i usuwania obiektów. Dany form wysyłano na odpowiedni plik html który zmieniał się w zależności od wybranej encji.

```
class ProjectorForm(ModelForm):  
    class Meta:  
        model = Projector  
        fields = '__all__'  
  
class SPLForm(ModelForm):  
    class Meta:  
        model = SPL  
        fields = '__all__'  
  
class CPLForm(ModelForm):  
    class Meta:  
        model = CPL  
        fields = '__all__'  
  
class EffectForm(ModelForm):  
    class Meta:  
        model = Effect  
        fields = '__all__'
```

```
def creatShow(request):  
    form = ShowForm()  
    if request.method == 'POST':  
        form = ShowForm(request.POST)  
        if form.is_valid():  
            form.save()  
            return redirect('/shows')  
  
    content = {'form': form}  
    return render(request, 'CinemaBase/formpage.html', content)
```

.5.4.3. Implementacja mechanizmów bezpieczeństwa

Główny mechanizmem bezpieczeństwa w bazie danych jest konieczność zalogowania się jak użytkownik z odpowiednimi uprawnieniami aby z niej korzystać. Chroni to bazę danych przed ingerencją niepowołanych osób z zewnątrz. Autoryzacja użytkowników stworzona jest za pomocą

dekortatorów funkcji w pythonie. Stworzono 3 dekoratory : określający czy użytkownik jest zalogowany, określający czy użytkownik jest operatorem lub adminem i określający czy użytkownik jest adminem. Następnie dekoratory przypisywane są na odpowiednimi funkcjami w pliku views.py według potrzeb. Innym mechanizmem zabezpieczenia jest zabezpieczenie przed ataki CSRF za pomocą umieszczana na wybranych sstronach gdzie zmieniane są dane z baze danych tokenu CSRF.

```
def unauthenticated_user(view_func):
    def wrapper_func(request, *args, **kwargs):
        if request.user.is_authenticated:
            return redirect('home')
        else:
            return view_func(request, *args, **kwargs)
    return wrapper_func

def allowed_users(allowed_roles=[]):
    def decorator(view_func):
        def wrapper_func(request, *args, **kwargs):
            group = None
            if request.user.groups.exists():
                group = request.user.groups.all()[0].name

            if group in allowed_roles:
                return view_func(request, *args, **kwargs)
            else:
                return redirect('permission')

        return wrapper_func

    return decorator
```

```

@login_required(login_url='login')
@allowed_users(allowed_roles=["admins", "operators"])
def spl(request, pk):
    spl = SPL.objects.get(id=pk)
    cpls = spl.CPLs.all()
    effects = spl.Effects.all()
    content = {'spl':spl, 'cpls':cpls, 'effects':effects}
    return render(request, 'CinemaBase/spl.html', content)

@login_required(login_url='login')
@admin_only
def createspl(request):
    form = SPLForm()
    if request.method == 'POST':
        form = SPLForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('/spl')

    content = {'form':form}
    return render(request, 'CinemaBase/formpage.html', content)

@login_required(login_url='login')
@admin_only
def updatespl(request,pk):
    spl = SPL.objects.get(id=pk)
    form = SPLForm(instance=spl)
    if request.method == 'POST':
        form = SPLForm(request.POST, instance=spl)
        if form.is_valid():
            form.save()
            return redirect('/spl')

    content = {'form': form}
    return render(request, 'CinemaBase/formpage.html', content)

```

```

{% extends 'CinemaBase/main.html' %}
{% load static %}
{% block content %}
    <p>Are you sure you want to delete " {{ item }} "</p>
    <form action="{% url delete_action item.id %}" method="POST">
        {% csrf_token %}
        <a href="{% url back_address %}">Cancel</a></a>
        <input type="submit" name="Confirm">
    </form>
{% endblock content %}

```

6. Podsumowanie i wnioski

Literatura

Spis rysunków

Spis tabel