Projekt systemu rezerwacji miejsc w kinie

Sprawozdanie techniczne – Backend

Jakub Janiczuk

Bartosz Bahonko

30 stycznia 2026

Spis treści

1 Cel i zakres Projektu

Jako zadanie projektowe wybraliśmy zaprojektowanie oraz implementacje aplikacji bazodanowej z aplikacją webową do rezerwacji miejsc w kinie. Mieliśmy również zaimplementować elementy związane z bazą danych takie jak:

- Trigger
- Tranzakcja
- funckja składowana
- Prodecura składowana
- Tabela Wirtualna

2 Zastosowane technologie

Obszar	**Technologia**
Język programowania	Python 3.12
Framework backendowy	FastAPI
ORM	SQLAlchemy
Walidacja danych	Pydantic
Serwer ASGI	Uvicorn
Baza danych	SQLite (środowisko projektowe)
Autoryzacja haseł	passlib (pbkdf2_sha256)
Zarządzanie zależnościami	pip + requirements.txt
Kontrola wersji	Git
Repozytorium	GitHub
Fronted	HTML, CSS, JavaScript (vanilla)

3 Architektura systemu

Backend został zaprojektowany w architekturze:

- REST API,
- monolitycznej aplikacji backendowej,
- rozdzielenia warstw: API, logika biznesowa, modele, schemy.

Frontend systemu został zaprojektowany jako:

- statyczne strony HTML, wspierane przez CSS i JavaScript,
- warstwa prezentacji komunikująca się z backendem poprzez REST API,
- dynamiczne aktualizacje treści (np. repertuar, rezerwacje) za pomocą JavaScript,
- dynamiczne aktualizacje treści (np. repertuar, rezerwacje) za pomocą JavaScript,
- modularna struktura plików (HTML, CSS, JS) ułatwiająca rozwój i utrzymanie.

3.1 Struktura projektu

```
backend/
          app/
                api/
                      filmy.py
                      repertuar.py
                      seanse.py
                      sale.py
                      rezerwacje.py
                      platnosci.py
                      raporty.py
                      uzytkownicy.py
                      cennik.py
                models.py
                schemas.py
                db.py
                main.py
          requirements.txt
          kino.db
frontend/
          css/
                logowanie.css
                strona_glowna.css
          js/
                api.js
                auth.js
                filmy.js
                filmy_admin.js
                raport_sprzedazy.js
                repertuar.js
                rezerwacja.js
                rezerwacje_uzytkownika.js
```

```
        sala.js
        seans.js
        strona_glowna.js
    admin_seans.html
    filmy.html
    filmy_admin.html
    index.html
    logowanie.html
    raport_sprzedazy.html
    rejestracja.html
    repertuar.html
    rezerwacja.html
    rezerwacje_uzytkownika.html
    sala.html
    strona_glowna.html
```

4 Zaimplementowane przypadki użycia (Use Case)

4.1 UC-DB1: Pobranie repertuaru i seansów

Cel: Uzyskanie listy filmów oraz seansów zaplanowanych w określonym dniu lub dla wybranego filmu.

Aktorzy: Klient, System rezerwacji

Endpoint:

- `GET /repertuar`

Funkcjonalność:

- pobranie wszystkich seansów,
- filtrowanie po dacie,
- filtrowanie po filmie,
- zwrot danych filmu, godziny i numeru sali.

Powiązane tabele: Film, Seanse, Sale

4.2 UC-DB2: Mapa dostępności miejsc dla seansu

Cel: Wyświetlenie statusu wszystkich miejsc (wolne, zarezerwowane, opłacone) dla danego seansu.

Aktorzy: Klient, System rezerwacji

Endpoint:

- `GET /seanse/{id}/miejsca`

Funkcjonalność:

- pobranie układu miejsc sali,
- wyliczenie statusu miejsc (Wolne, Zarezerwowane, Opłacone),
- priorytety statusów miejsc.

Powiązane tabele: Miejsce, Seans, Rezerwacja, Rezerwacja_Miejsca

4.3 UC-DB3: Utworzenie rezerwacji

Cel: Zablokowanie wybranych miejsc i utworzenie rezerwacji ze statusem przetwarzana.

Aktorzy: Klient, System rezerwacji

Endpoint:

- `POST /rezerwacje`

Funkcjonalność:

- walidacja dostępności miejsc,
- utworzenie rezerwacji,
- zapis miejsc w tabeli pośredniej,
- obsługa statusu *Oczekująca*.

Powiązane tabele: Rezerwacja, Miejsce, Rezerwacja_Miejsca

4.4 UC-DB4: Potwierdzenie rezerwacji (płatność)

Cel: Zmiana statusu rezerwacji i powiązanych miejsc na zapłacone.

Aktorzy: Klient, System płatności

Endpointy:

- `POST /platnosci/start`
- `POST /platnosci/confirm`

Funkcjonalność:

- symulacja systemu płatności,
- wyliczenie kwoty na podstawie cennika,
- zmiana statusu rezerwacji na *Potwierdzona*,
- zmiana statusu miejsc na *Opłacone*.

Powiązane tabele: Rezerwacja, Rezerwacja_Miejsca, Miejsce

4.5 UC-DB5: Anulowanie rezerwacji

Cel: Anulowanie rezerwacji i zwolnienie miejsc.

Aktorzy: Klient, System rezerwacji

Endpoint:

- `POST /rezerwacje/{id}/anuluj`

Funkcjonalność:

- zmiana statusu rezerwacji,
- zwolnienie miejsc w `Rezerwacja_Miejsca`.

Powiązane tabele: Rezerwacja, Rezerwacja_Miejsca, Miejsce

4.6 UC-DB6: Zarządzanie repertuarem (operator)

Cel: Dodawanie, edycja i usuwanie filmów oraz seansów.

Aktorzy: Operator systemu

Endpointy:

- `POST /filmy`
- `PUT /filmy/{id}`
- `DELETE /filmy/{id}`
- `POST /seanse`
- `DELETE /seanse/{id}`

Funkcjonalność:

- dodanie, edycja i usuwanie filmu,
- dodanie i usuwanie seansu,
- sprawdzenie poprawności danych (np. brak konfliktów sali).

Powiązane tabele: Film, Seans, Sala

4.7 UC-DB8: Sprzątanie rezerwacji wygasłych

Cel: Automatyczne anulowanie rezerwacji, których czas ważności minął.

Aktorzy: Proces automatyczny (cron)

Funkcjonalność:

- obsługa pola `data_wygasniecia`,
- przygotowanie logiki pod zadanie cykliczne (cron).

Powiązane tabele: Rezerwacja

4.8 UC-DB9: Rejestracja użytkownika

Cel: Utworzenie konta użytkownika (klienta lub operatora).

Aktorzy: Klient, Administrator

Endpoint:

- `POST /uzytkownicy`

Funkcjonalność:

- walidacja unikalności e-mail,
- bezpieczne hashowanie hasła,
- rozróżnienie typu użytkownika (klient / operator).

Powiązane tabele: Uzytkownik subsectionUC-EXT1: Wyszukiwanie filmów po fragmencie tytułu

4.9 UC-DB10: Wyszukiwanie filmów po fragmencie tytułu

Cel: Umożliwienie szybkiego wyszukiwania filmów na podstawie części nazwy.

Aktorzy: Klient, Operator

Opis:

- użytkownik podaje fragment tytułu filmu,
- system zwraca listę pasujących filmów,
- wyszukiwanie jest niewrażliwe na wielkość liter.

Endpoint:

- `GET /filmy/szukaj?q=fragment`

Powiązane tabele: Film

4.10 UC-DB11: Filtrowanie repertuaru

Cel: Rozszerzenie pobierania repertuaru o zaawansowane filtrowanie.

Aktorzy: Klient

Opis:

- filtrowanie repertuaru po dacie,
- filtrowanie repertuaru po identyfikatorze filmu,
- możliwość łączenia filtrów.

Endpoint:

- `GET /repertuar?data=YYYY-MM-DD`
- `GET /repertuar?id_filmu=X`

Powiązane tabele: Film, Seans, Sala

4.11 UC-DB12: Pobranie wszystkich seansów

Cel: Umożliwienie operatorowi pobrania kompletnej listy seansów.
Aktorzy: Operator
Opis:

- system zwraca wszystkie seanse niezależnie od daty,
- funkcjonalność przydatna w panelach administracyjnych.

Endpoint:

- `GET /seanse`

Powiązane tabele: Seans, Film, Sala

4.12 UC-DB13: Automatyczne generowanie miejsc w sali

Cel: Uproszczenie konfiguracji sal kinowych.
Aktorzy: Operator
Opis:

- operator definiuje salę,
- system automatycznie generuje miejsca na podstawie parametrów,
- eliminacja ręcznego wprowadzania miejsc.

Endpoint:

- `POST /sale/{id}/generuj_miejsca`

Powiązane tabele: Sala, Miejsce

4.13 UC-DB14: Pobranie rezerwacji użytkownika

Cel: Udostępnienie użytkownikowi listy jego rezerwacji.

Aktorzy: Klient

Opis:

- system zwraca wszystkie rezerwacje danego użytkownika,
- prezentowany jest status rezerwacji oraz przypisane miejsca.

Endpoint:

- `GET /uzytkownicy/{id}/rezerwacje`

Powiązane tabele: Uzytkownik, Rezerwacja, Rezerwacja_Miejsca

4.14 UC-DB15: Symulacja systemu płatności

Cel: Odseparowanie logiki płatności od logiki rezerwacji.

Aktorzy: Klient, System płatności (symulowany)

Opis:

- rozpoczęcie płatności dla rezerwacji,
- wyliczenie kwoty na podstawie cennika,
- potwierdzenie płatności,
- aktualizacja statusów rezerwacji i miejsc.

Endpointy:

- `POST /platnosci/start`
- `POST /platnosci/confirm`

Powiązane tabele: Rezerwacja, Rezerwacja_Miejsca

4.15 UC-DB16: Centralny cennik biletów

Cel: Centralizacja logiki cen biletów.

Aktorzy: System

Opis:

- ceny biletów przechowywane w jednym module konfiguracyjnym,
- obsługa różnych typów biletów (np. normalny, ulgowy),
- możliwość łatwej zmiany cen bez modyfikacji logiki API.

Endpoint:

- `GET /cennik`

5 Bezpieczeństwo

- hasła przechowywane jako hash,
- walidacja danych wejściowych,
- przygotowana struktura pod role użytkowników.

6 Przykładowe elementy związane z bazą danych

- Widok – Repertuar Kina
- Procedura – RejestracjaUzytkownika
- Trigger – Automatyczne sprzątanie wygasłych rezerwacji oraz seansów
- Tranzakcja – tworzenie rezerwacji
- Funkcja składowana – obliczanie ceny

7 Implementacja aplikacji

Aplikacja została zaimplementowana w języku Python 3.12 z wykorzystaniem frameworku FastAPI dla warstwy backendowej oraz SQLite jako bazy danych. Komunikacja między frontendem a backendem odbywa się poprzez REST API, a dane są przesyłane w formacie JSON.

7.1 Połączenie z bazą danych

Backend korzysta z ORM `SQLAlchemy`, który umożliwia bezpośrednią obsługę bazy SQLite w sposób obiektowy:

- **Engine** – odpowiada za połączenie z plikiem bazy danych (`kino.db`),
- **Session** – jednostka pracy, która pozwala na wykonywanie zapytań, transakcji i modyfikacji danych,
- **Modele** – klasy odpowiadające tabelom w bazie danych, np. `Film`, `Seans`, `Sala`, `Miejsce`, `Rezerwacja`, `Rezerwacja_Miejsca`, `Uzytkownik`.

Transakcje są wykorzystywane przy operacjach wymagających atomowości, np. tworzenie rezerwacji, gdzie zarówno zapis rezerwacji, jak i przypisanie miejsc musi zakończyć się sukcesem lub zostać anulowane w całości.

7.2 Obsługa operacji CRUD i logiki biznesowej

- **Odczyt danych** – wykonywany poprzez zapytania SELECT w SQLAlchemy, zwracające dane w formie obiektów Python, które następnie są konwertowane do JSON i wysyłane do frontend-u przez REST API,
- **Tworzenie i modyfikacja danych** – operacje INSERT i UPDATE realizowane są w blokach transakcyjnych (`with session.begin():`) w celu zapewnienia spójności danych,
- **Procedury składowane / funkcje pomocnicze** – logika biznesowa, np. walidacja dostępności miejsc, obliczanie ceny biletów, zmiana statusów rezerwacji i miejsc,
- **Trigger** – np. automatyczne sprzątanie wygasłych rezerwacji.

7.3 Frontend i dynamiczna prezentacja danych

Frontend został zrealizowany przy użyciu statycznych stron HTML z obsługą CSS i JavaScript:

- Komunikacja z backendem odbywa się przez REST API przy użyciu `fetch()`, przesyłając dane w formacie JSON,
- Dane dynamicznie aktualizują zawartość strony, np. lista repertuaru, status miejsc w salach czy lista rezerwacji użytkownika,
- Filtrowanie i wyszukiwanie danych odbywa się poprzez dynamiczne generowanie URL z parametrami GET (np. fragment tytułu filmu i data seansu) i wywołanie endpointu backendu,
- Przenoszenie informacji między stronami (np. wybrany seans) odbywa się przy użyciu `localStorage` w przeglądarce,
- Frontend renderuje dane w formie dynamicznie tworzonych elementów HTML (`div`, `button`), a interakcje użytkownika (np. kliknięcie "Rezerwuję") inicjują odpowiednie wywołania API lub zapis danych lokalnie.

7.4 Przepływ danych w aplikacji

1. Użytkownik wprowadza dane na stronie (np. fragment tytułu filmu, datę seansu),
2. Frontend wysyła żądanie HTTP GET lub POST do backendu FastAPI,
3. Backend pobiera lub modyfikuje dane w SQLite poprzez SQLAlchemy, stosując transakcje i procedury w razie potrzeby,

4. Dane są zwracane w formacie JSON do frontend-u,

5. Frontend dynamicznie wyświetla dane lub zapisuje wybrane informacje w `localStorage` do użycia na kolejnych stronach.

8 Podsumowanie i wnioski

W ramach projektu stworzono system rezerwacji kinowej, w którym zastosowano nowoczesne podejście do budowy aplikacji webowej. Backend został zaimplementowany w Pythonie z wykorzystaniem FastAPI i SQLite, a komunikacja z frontendem odbywa się poprzez REST API w formacie JSON. W projekcie zastosowano ORM SQLAlchemy, co umożliwia bezpieczne i efektywne operacje na bazie danych, z wykorzystaniem transakcji, procedur składowanych oraz triggerów do automatycznych działań.

Frontend został zrealizowany przy użyciu statycznych stron HTML, CSS i JavaScript, z dynamiczną aktualizacją treści oraz interaktywnymi elementami, takimi jak wyszukiwanie repertuaru czy status miejsc w salach. Dzięki zastosowaniu modularnej struktury plików kod jest przejrzysty, łatwy w utrzymaniu i rozwijaniu.

Zastosowanie wyżej opisanych rozwiązań pozwoliło na:

- zapewnienie spójności danych w bazie poprzez transakcje i procedury,
- dynamiczne i responsywne interfejsy użytkownika,
- modularną i skalowalną architekturę backendu i frontend,
- łatwą integrację nowych funkcjonalności w przyszłości.

9 Repozytorium

Repozytorium Github z kodem można znaleźć pod linkiem: `https://github.com/kubajaniczuk-bit/Bazy_danych_projekt`