Bazy danych II – miniprojekt

Autorzy:

- **Stanisław Strojniak**
- **Wiktor Kostka**

Link do repozytorium na GitHubie:

https://github.com/stawkey/room-reservation

Nasz projekt jest implementacją uniwersalnego systemu rezerwacji sal. Można w nim obsługiwać sale, zarządzać ich rezerwacjami, a także przypisywać im konkretne tagi świadczące o ich udogodnieniach. Projekt posiada implementację zarówno w backendzie, jak i we frontendzie.

Użyte technologie:

Backend:

- Baza danych: PostgreSQL
- ORM: Hibernate + Java – mapowanie klas Javy na tabele bazy danych bez konieczności pisania ręcznych zapytań SQL
- Spring Boot – konfiguracja serwera HTTP, tworzenie REST API z użyciem mappingu, @Repository i @Service, integracja z bazą danych
- Swagger – testowanie i dokumentowanie API
- OpenAPI Generator – generowanie szkieletu backendu na podstawie konfiguracji .yaml

Frontend:

- React 18 + JavaScript – stworzenie UI i interakcja z backendem

- Material-UI – zestaw gotowych komponentów do użycia w UI
- Axios - komunikacja z API
- Vite – budowanie i uruchamianie aplikacji frontendowej

Struktura backendu:

Aplikacja została podzielona na warstwy:

Controller – obsługa żądań HTTP,

Service – logika biznesowa,

Repository – dostęp do bazy danych,

Entity/DTO/Mapper – model danych i mapowanie.

Proces powstawania:

Na początku napisaliśmy dokumentację OpenAPI w pliku .yaml, co umożliwiło wygenerowanie interfejsów z endpointami oraz plików Data Transfer Object (DTO) za pomocą OpenAPI Generator skonfigurowanego pod Spring Boot 3. Następnie stworzyliśmy klasy encji w Hibernate, repozytoria, warstwę serwisową oraz kontrolery. Na końcu przygotowaliśmy prosty frontend, który umożliwia prezentacje możliwości backendu.

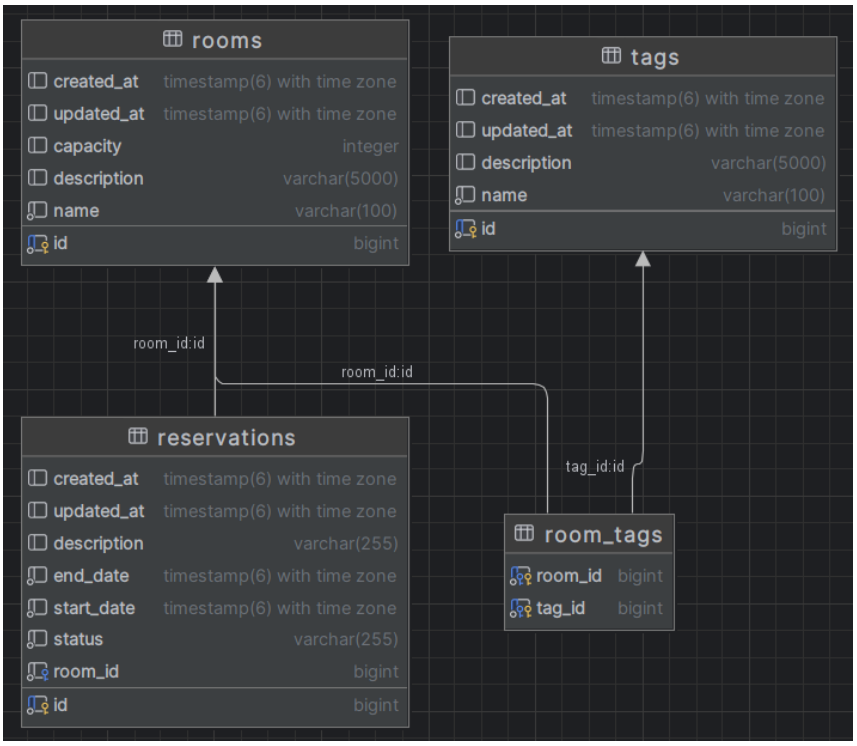
Model danych i projekt bazy danych

Opis encji i relacji:

Room – sala do rezerwacji, posiada wiele rezerwacji i wiele tagów (wiele-do-wielu).

Reservation – rezerwacja sali, każda rezerwacja ma jedną salę.

Tag – cecha sali (np. projektor), powiązany z wieloma salami (wiele-do-wielu).



Operacje transakcyjne

Przykład operacji transakcyjnej:

Tworzenie rezerwacji (adnotacja @Transactional): sprawdzenie konfliktu terminów i zapis rezerwacji w jednej transakcji.

```
@Transactional
public Result<Reservation> createReservation(Long roomId, Reservation
reservation) {
    // sprawdzenie konfliktów i zapis w jednej transakcji
}
```

Prezentacja możliwości technologii

Spring Boot – wiele rzeczy jest skonfigurowanych na start, gotowe mechanizmy REST, transakcje, bezpieczeństwo.

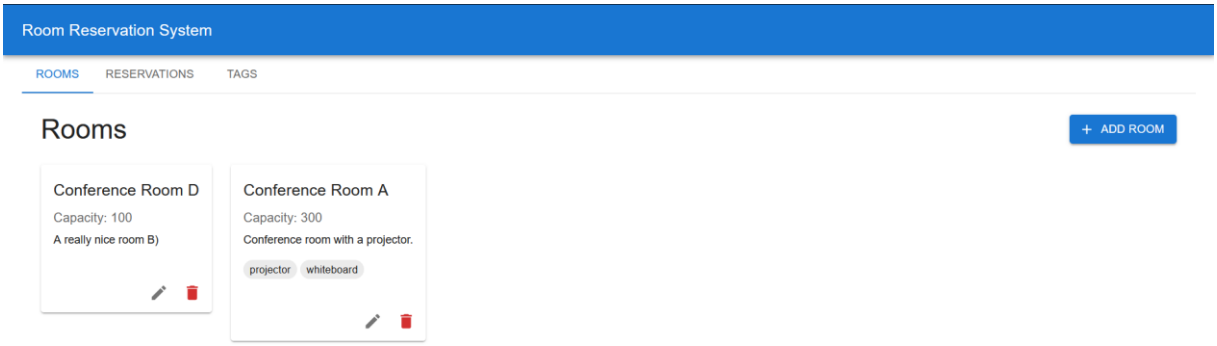
Hibernate/JPA – mapowanie obiektowo-relacyjne, łatwe zapytania.

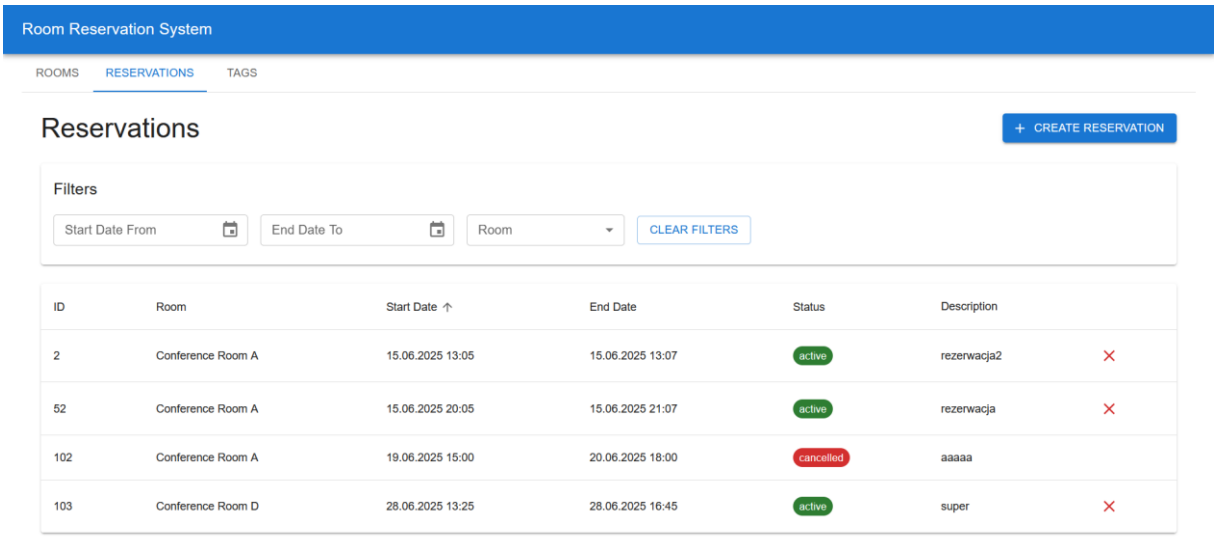
PostgreSQL – wydajna, skalowalna baza relacyjna.

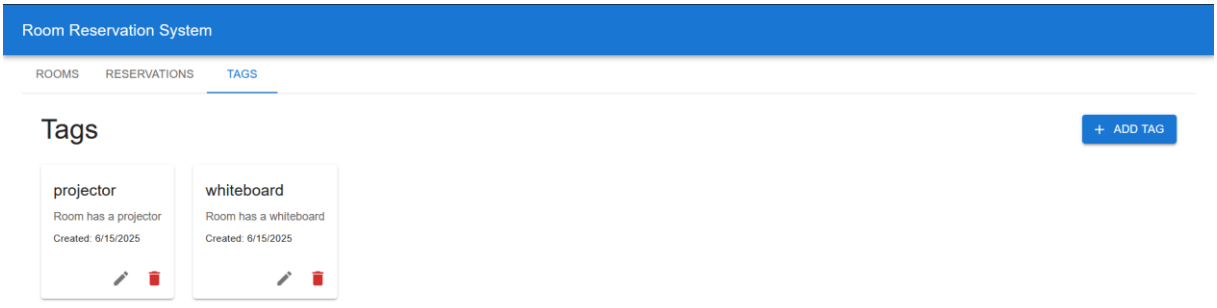
React + MaterialUI– nowoczesny frontend, dynamiczny interfejs.

Swagger/OpenAPI – dokumentacja i testowanie API, generowanie kodu klienta.

Screeny frontendu:







Endpointy (dostępne w wygodnym formacie w localhost:8080/ po uruchomieniu backendu)

POST /rooms

Opis: Tworzy nowy pokój z opcjonalnymi tagami
Dane wejściowe:

```
{
  "name": "Room A",
  "capacity": 20,
  "description": "Conference room",
  "tagIds": [1, 2]
}
```

Możliwe do otrzymania statusy http:

- 201 Created – pokój utworzony
- 400 Bad Request – nieprawidłowe dane wejściowe lub błąd serwisu

Dane wyjściowe:
RoomDto zawierający dane nowo utworzonego pokoju

GET /rooms/{room_id}

Opis:
Zwraca informacje o pokoju identyfikowanym przez room_id

Dane wejściowe:
Identyfikator pokoju room_id

Możliwe do otrzymania statusy HTTP:

- 200 OK – pokój znaleziony
- 404 Not Found – pokój nie istnieje

Dane wyjściowe:
RoomDto zawierający dane znalezionego pokoju

GET /rooms?page=X&page_size=Y

Opis:
Zwraca listę pokoi z paginacją

Dane wejściowe:
page: domyślnie 1, minimum 1
page_size: domyślnie 20, liczba między 1 a 100

Możliwe do otrzymania statusy HTTP:
- 200 OK – dane zwrócone poprawnie
- 500 Internal Server Error – błąd przy pobieraniu danych

Dane wyjściowe:
JSON ListRooms200Response

```
{
  "total": 42,
  "totalPages": 3,
  "page": 1,
  "pageSize": 20,
  "items": [ { ...RoomDto... }, ... ]
}
```

DELETE /rooms/{room_id}

Opis:
Usuwa pokój o podanym ID. Jeżeli pokój miał jakieś rezerwacje, zostają one anulowane

Dane wejściowe:
Identyfikator pokoju room_id

Możliwe do otrzymania statusy HTTP:

- 204 No Content – pokój został usunięty
- 404 Not Found – pokój nie istnieje
- 500 Internal Server Error – błąd serwera

Dane wyjściowe:
Brak

PUT /rooms/{room_id}

Opis:
Aktualizuje dane pokoju o podanym ID

Dane wejściowe:
UpdateRoomRequest

```
{
  "name": "Updated Room",
  "capacity": 25,
  "description": "New description",
  "tagIds": [3, 4]
}
```

Możliwe do otrzymania statusy HTTP:

- 200 OK – pokój zaktualizowany
- 404 Not Found – pokój nie istnieje
- 400 Bad Request – błąd walidacji danych

Dane wyjściowe:
RoomDto zaktualizowanego pokoju

GET /rooms/{room_id}/reservations

Opis:

Zwraca paginowaną listę rezerwacji dla danego pokoju, opcjonalnie filtrowaną po dacie.

Dane wejściowe:

- room_id – ID analizowanego pokoju
- startDate, endDate – zakres dat filtrowania (opcjonalnie)
- page, pageSize (opcjonalnie)

Możliwe do otrzymania statusy HTTP:

- 200 OK – dane zwrócone
- 404 Not Found – pokój nie istnieje
- 500 Internal Server Error – błąd systemu

Dane wyjściowe:
JSON Page:

```
{
  "items": [ { ...ReservationDto... } ],
  "page": 1,
  "pageSize": 10,
  "total": 30,
  "totalPages": 3
}
```

POST /rooms/{room_id}/reservations

Opis:
Tworzy nową rezerwację dla podanego pokoju

Dane wejściowe:

- room_id – ID analizowanego pokoju
- startDate, endDate – data początku i końca rezerwacji

Możliwe do otrzymania statusy HTTP:

- 200 Created – rezerwacja utworzona
- 400 Bad Request – zły input
- 404 Not Found – pokój nie istnieje
- 409 Conflict – rezerwacja koliduje z inną, istniejącą rezerwacją
- 500 Internal Server Error – błąd serwera

Dane wyjściowe:
JSON ReservationDto

DELETE /reservations/{id}

Opis:
Anuluje istniejącą rezerwację

Dane wejściowe:

- id – ID analizowanej rezerwacji

Możliwe do otrzymania statusy HTTP:

- 204 No Content – rezerwacja anulowana pomyślnie
- 400 Bad Request – nieprawidłowy input
- 404 Not Found – rezerwacja o podanym ID nie istnieje
- 500 Internal Server Error – błąd serwera

GET /reservations/{id}

Opis:
Pobiera szczegóły rezerwacji na podstawie jej ID

Dane wejściowe:

- id – ID analizowanej rezerwacji

Możliwe do otrzymania statusy HTTP:

- 204 No Content – rezerwacja anulowana pomyślnie
- 400 Bad Request – nieprawidłowy input
- 404 Not Found – rezerwacja o podanym ID nie istnieje
- 500 Internal Server Error – błąd serwera

Dane wyjściowe:
ReservationDto analizowanej rezerwacji

GET /reservations

Opis:
Zwraca listę rezerwacji w sposób stronicowany, z możliwością filtrowania i sortowania

Dane wejściowe:

- start – rozpoczęcie filtrowania od danej daty
- end – zakończenie filtrowania na danej dacie
- room_id – filtrowanie po ID pokojów
- sort – wybór pola sortowania (date, createdAt, …)
- order – kierunek sortowania
- page – numer strony (domyślnie 1)
- page_size – liczba wyników na stronę (domyślnie 20, między 1 a 100)

Możliwe do otrzymania statusy HTTP:

- 200 OK – zwraca listę rezerwacji
- 500 Internal Server Error – błąd serwera

Dane wyjściowe (przykładowe):


```
{
 "page": 1,
 "pageSize": 20,
 "total": 120,
 "totalPages": 6,
 "items": [
  {
   "id": 1234,
```

```
    "roomId": 321,
    "description": "description",
    "start": "2000-01-23T04:56:07.000+00:00",
    "end": "2000-01-23T04:56:07.000+00:00",
    "status": "active",
    "createdAt": "2025-05-24T12:34:56Z",
    "updatedAt": "2025-05-25T08:00:00Z"
   }
 ]
}
```


POST /tags

Opis:
Tworzy nowy tag reprezentujący cechę sali

Dane wejściowe:


```
{
 "name": "projector",
 "description": "Room has a projector available"
}
```


Możliwe do otrzymania statusy HTTP:

- 201 Created – tag został utworzony
- 400 Bad Request – nieprawidłowy input
- 500 Internal Server Error – błąd serwera

Dane wyjściowe (przykładowe):


```
{
 "id": 1,
 "name": "projector",
 "description": "Room has a projector available",
 "createdAt": "2025-05-24T12:34:56Z",
 "updatedAt": "2025-05-24T12:34:56Z"
}
```

GET /tags/{tagId}

Opis:
Zwraca szczegóły konkretnego tagu na podstawie jego ID

Dane wejściowe:
tagId – ID tagu

Możliwe do otrzymania statusy HTTP:

- 200 OK – znaleziono i zwrócono tag
- 404 Not Found – tag nie istnieje
- 500 Internal Server Error – błąd serwera

Dane wyjściowe:
TagDto zawierające informacje danego tagu

GET /tags

Opis:
Zwraca listę wszystkich tagów

Dane wejściowe:
brak

Możliwe do otrzymania statusy HTTP:

- 200 OK – zwrócono listę tagów
- 500 Internal Server Error – błąd serwera

Dane wyjściowe:
Lista tagDto, np.:

```
[
  {
    "id": 1,
    "name": "projector",
    "description": "Room has a projector available",
    "createdAt": "2025-05-24T12:34:56Z",
    "updatedAt": "2025-05-24T12:34:56Z"
  },
  {
    "id": 2,
    "name": "airConditioning",
    "description": "Room has air conditioning",
    "createdAt": "2025-05-24T13:00:00Z",
    "updatedAt": "2025-05-24T13:00:00Z"
  }
]
```

PUT /tags/{tagId}

Opis:
Aktualizuje nazwę i/lub opis istniejącego tagu

Dane wejściowe:

- tagId
- UpdateTagRequest, np.:

```
{
  "name": "projector",
  "description": "Updated description"
}
```

Możliwe do otrzymania statusy HTTP:

- 200 OK – aktualizacja zakończona sukcesem
- 400 Bad Request – nieprawidłowy input
- 404 Not Found – tag nie istnieje
- 500 Internal Server Error – błąd serwera

Dane wyjściowe:
Zaaktualizowany obiekt tagDto

DELETE /tags/{tagId}

Opis:
Usuwa tag o podanym ID

Dane wejściowe:
tagId – ID tagu, który chcemy usunąć

Możliwe do otrzymania statusy HTTP:

- 204 No Content – tag usunięty poprawnie
- 404 Not Found – tag nie istnieje
- 500 Internal Server Error – błąd serwera