# Aplikacja do zarządzania hotelem i rezerwacji noclegów

## 1. O aplikacji

Opisywana aplikacja jest prostym narzędziem do zarządzania siecią hoteli oraz jako klient do rezerwowania noclegów w wybranym terminie. Aplikacja z założenia ma służyc pojedynczemu podmiotowi w branży hotelarskiej.

Po wypakowaniu plików do pustego katalogu kontenery należy zbudować komendą: ***docker-compose up***

Po zbudowaniu aplikacja będzie działać ne serwerze lokalnym, na portach:

- Backend - ***http://localhost:8080/***
- Frontend - ***http://localhost:4200/***

## 2. Użyte narzędzia

Od strony technicznej aplikacja podzielona jest na warstwę użytkową - frontend, oraz na backend i bazę danych. Do stworzenia poszczególnych warstw użyto następujących narzędzi:

- Frontend:
  - Klient z użyciem biblioteki Angular 16.0.2
- Backend:
  - Aplikacja Spring używająca zależności:
    - Spring Data JPA
    - Spring Web
    - Spring AOP
    - i inne ułatwiające tworzenie aplikacji...
- Baza danych:
  - Baza PostgreSQL
- Konteneryzacja
  - Docker
  - Docker-Compose

# 3. Działanie

**Kontener:**

Plik 'docker-compose.yml' jest skonfigurowany, aby po wykonaniu komendy zbudować 2 kontenery - jeden na serwer i drugi na backend. Budowa może potrwać kilka minut. Po zbudowaniu obrazy kontenerów powinny mieć rozmiar około 2.5GB. Kontenery są wystawione na porty wymienione w punkcie 1.

**Baza danych:**

Server jest skonfigurowany do połączenia z bazą PostgreSQL. Do projektu jest dołączona prosta baza w kontenerze. Źródło danych można ustawić w pliku *resources/application.yaml.* Projekt poprzednio używał nieistniejącego już serwisu chmurowego ElephantSQL i nazwy zmiennych bazy danych zostały ustawione na takie same, aby nie trzeba było re-kompilować pliku .jar. Obecna baza nie posiada wolumenu i zostaje utracona w momencie wyłączenia kontenera.

**Część serwerowa/backend:**

Serwer jest wykonany w technologii Spring. Aplikacja jest skonfigurowana do połączenia się z serwerem PostgreSQL, który zawiera bazę danych z tabelami "Users", "Rooms" i "Reservations". Dla tych tabelek są zostały utworzone klasy reprezentujące dane oraz repozytoria, które implementują istniejące repozytoria Spring Data JPA oraz repozytorium do paginacji (w obecnej chwili aplikacja nie ma zaplementowanej mechaniki paginacji). Wykorzystując repozytoria, zostały zaimplementowane kontrolery wykonujące zapytania CRUD skonfigurowane pod odpowiednie endpointy.
W konfiguracji jest ustawione, aby zezwolić na CORS dla adresu z frontendem. Dodatkowo serwer zawiera logger Spring AOP, który wypisuje na konsole nazwę metody, oraz komunikaty w momencie jej uruchomienia i zakończenia.
W kwietniu 2025 istniał problem z kompilacją, związany z biblioteką Lombok, która jest używana w kodzie.

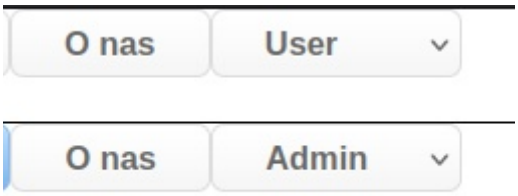
**Część użytkowa/frontend:**

Frontend do użytkowania powstał w technologii Angular. Serwis HTTP łączy się pod znany adres serwera na którym jest backend i pobiera stamtąd odpowiednie zapytania. Do przełączania pomiędzy funkcjami aplikacji służy router i dla każdej ścieżki serwis HTTP wykonuje przeważnie jedno zapytanie (wyjątkiem jest lista rezerwacji poszczególnych użytkowników, gdzie dla każdego użytkownika jest osobne zapytanie o jego rezerwacje). W przyszłości planowane jest dodatkowo logowanie do aplikacji. W obecnej chwili istnieje jedynie flaga do przełączania kontekstu między menedżerem, a klientem hotelu.

# 4. Instrukcja obsługi

Rodzaj użytkownika:

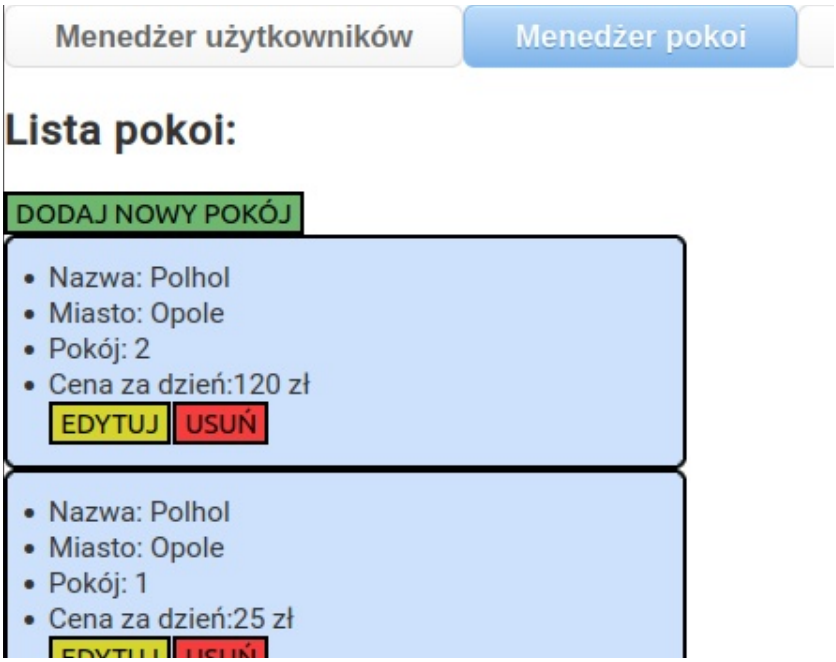
- W obecnej chwili aplikacja nie posiada jeszcze opcji logowania. Aby zmienić konstekst pomiędzy menedżerem hotelu, a klientem, należy wybrać opcję na pasku nawigacji. Po wybraniu opcji "User" można korzystać z aplikacji jako przykładowy klient, a po wybraniu "Admin", jako menedżer.




Rysunek 4.1 - Opcja przełączania kontekstu między klientem, a mendedżerem hotelu.

Menedżer ma do dyspozycji następujące funkcje:

- Zarządzanie pokojami - istniejące pokoje można edytować lub usunąć. Można też dodać do listy nowy pokój za pomocą przycisku.

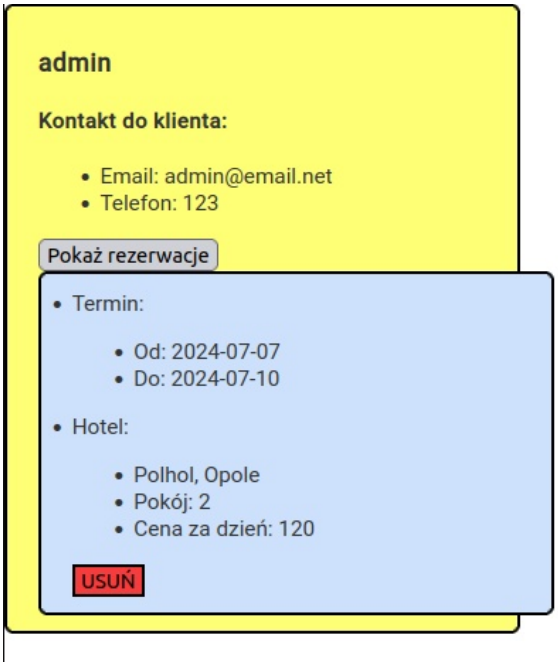



Rysunek 4.2 - Menu zarządzania pokojami. Kliknięcie "DODAJ NOWY POKÓJ" otworzy dialog dodania pokoju.


Rysunek 4.3 - Okienko dodwania pokoju. W przypadku wybrania opcji "EDYTUJ", okienko zostanie wstępnie wypłenione istniejącymi danymi.

- Zarządzanie użytkownikami - mendżer widzi listę swoich klientów wraz z ich danymi kontaktowymi. Dla każdego klienta może rozwinąć listę rezerwacji i usunąć wybraną rezerwację.


Rysunek 4.4 - Menu zarządzania klientami.

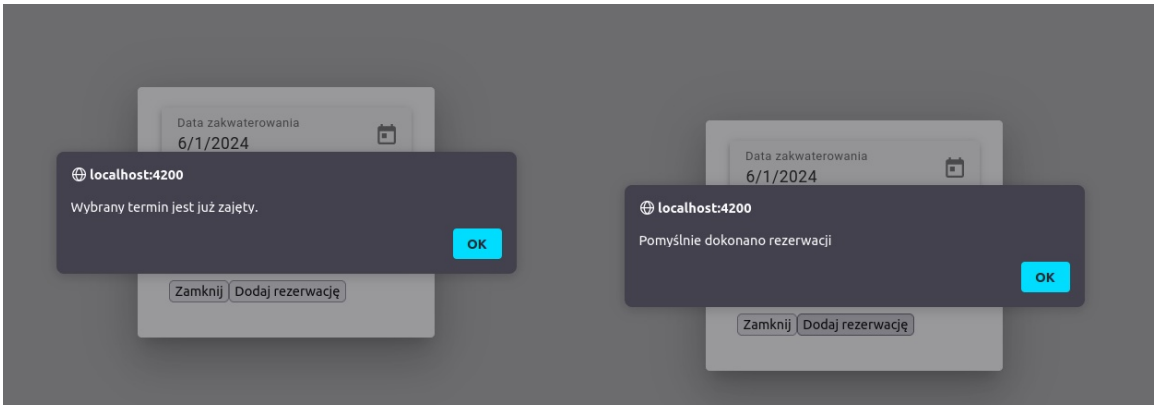



Rysunek 4.5 - Po kliknięciu "Pokaż rezerwacje", menedżer widzi rezerwacje danego użytkownika i może je usunąć.

Funkcje dostępne dla klienta:

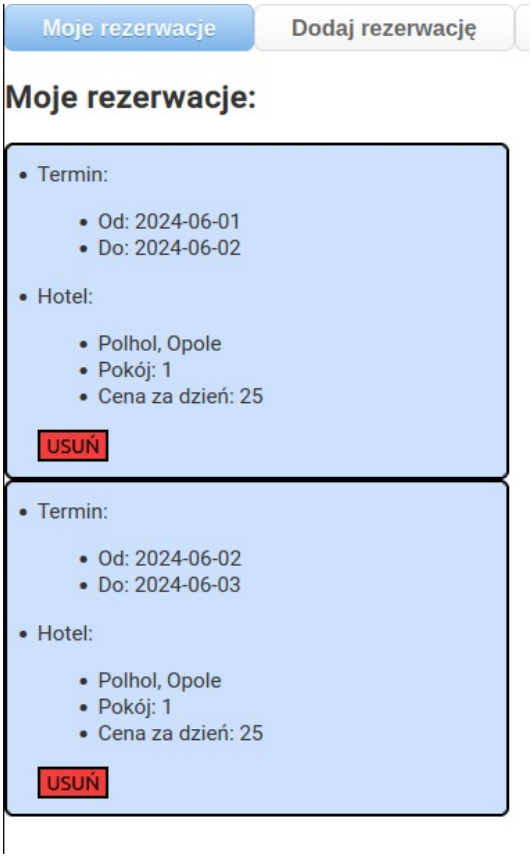
- Dodanie nowej rezerwacji - klient wybiera pokój z listy dostępnych pokoi. Następnie wpisuje daty zakwaterowania i wymeldunku i zatwierdza. W przypadku, gdy dany pokój jest już zajęty w wybranym terminie, klient zostanie o tym poinformowany i rezerwacja się nie powiedzie.


Rysunek 4.6 - Użytkownik może wybrać pokój z listy utworzonej przez menedżera.




Rysunek 4.7 - W zależność od tego czy wybrany termin jest wolny, użytkownik może dostać różną odpowiedź od serwera.

- Zarządzanie rezerwacjami - klient widzi listę dokonaych przez siebie rezerwacji wraz ze szczegółami. Każdą z rezerwacji można usunać.




Rysunek 4.8 - Menu klienta do zarządzania swoimi rezerwacjami.

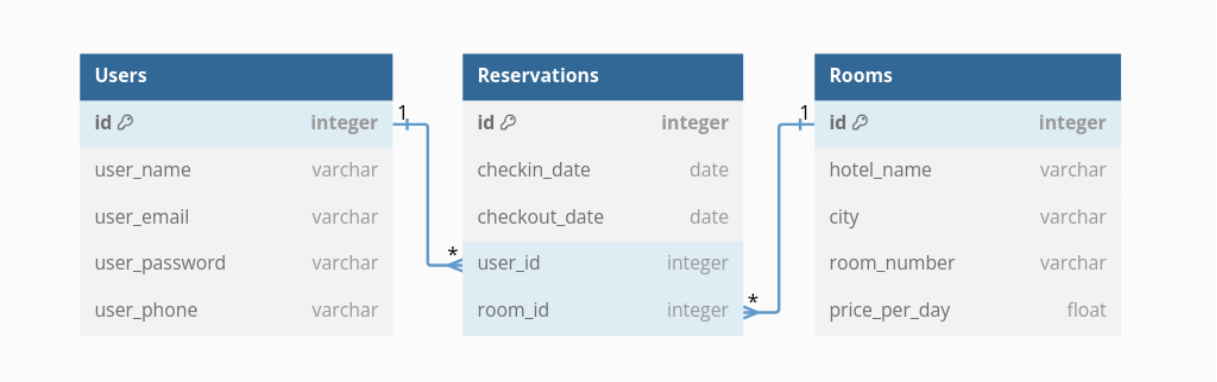
# 5. Testy jednostkowe

Aplikacja posiada zestaw prostych testów jednostkowych dla strony serwerowej. Testy są sporządzone dla każdego kontrolera i sprawdzają poprawność zapytań sieciowych, oraz poprawne budowanie klasy kontrolera przez aplikację. Dodatkowo na klasy testowe jest nałożony aspekt @Transactional, aby nie zapisywać rekordów do bazy danych.

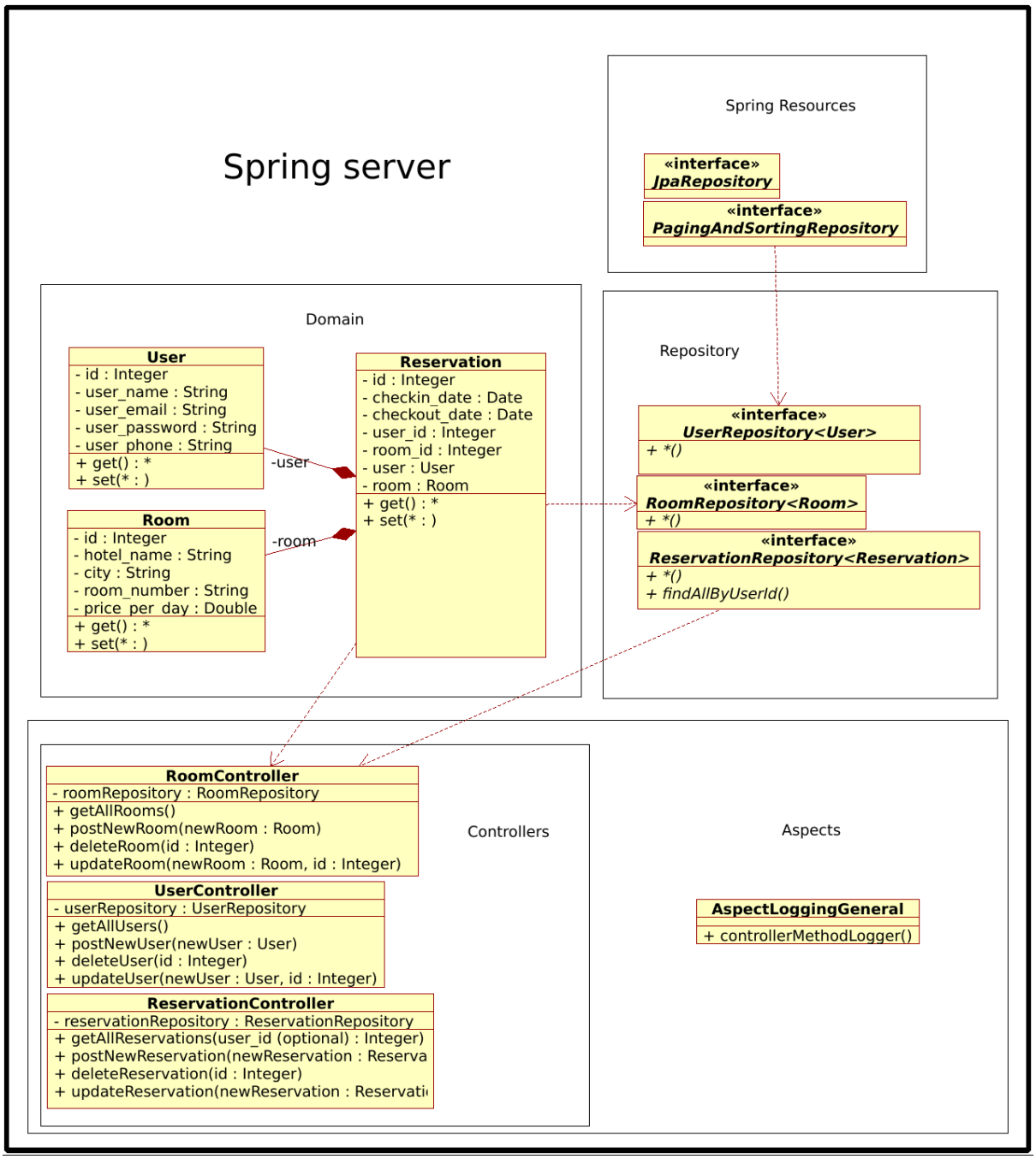
```
no usages
@SpringBootTest
@AutoConfigureMockMvc
@Transactional
public class ReservationControllerTests {

    1 usage
    @Autowired
    private ReservationController reservationController;
    4 usages
    @Autowired
    private MockMvc mockMvc;

    no usages
    @Test
    void reservationControllerExists() throws Exception{
        assertThat(reservationController).isNotNull();
    }

    no usages
    @Test
    void reservationGetEndpointsWork() throws Exception{
        this.mockMvc.perform(get( urlTemplate: "/reservations")).andExpect(status().isOk());
        this.mockMvc.perform(get( urlTemplate: "/reservations?user_id=1")).andExpect(status().isOk());
    }
```

Rysunek 5.1 - Zestaw testów dla kontrolera rezerwacji.

# 6. Schemat bazy danych



# 7. Schemat serwera



# 8. Endpointy

UserController:

- getAllUsers: **/users**
- postNewUser: **/user**
- deleteUser: **/user/{id}**
- updateUser: **/user/{id}**

RoomController:

- getAllRooms: **/rooms**
- postNewRoom: **/room**
- deleteRoom: **/room/{id}**
- updateRoom: **/room/{id}**

ReservationController:

- getAllReservations: **/reservations?user_id=**
- postNewReservation: **/reservation**
- deleteReservation: **/reservation/{id}**
- updateReservation: **/reservation/{id}**