

ZADANIE REKRUTACYJNE – JAVA DEVELOPER

Biznes case:

Twoim zadaniem jest opracowanie prostego systemu do zarządzania bazą danych lotów i pasażerów.

Opis zadania:

Napisz program w języku Java (8+) , który będzie umożliwiał operacje CRUD na danych lotów i pasażerów. Program powinien umożliwiać dodawanie, odczytywanie, aktualizowanie i usuwanie informacji o lotach oraz pasażerach oraz pozwalać na dodanie/usunięcie pasażera do konkretnego lotu.

Klasa `Flight` powinna zawierać podstawowe informacje o locie, takie jak numer lotu, trasę, datę i godzinę wylotu, ilość dostępnych miejsc.

Klasa `Passenger` powinna zawierać podstawowe informacje o pasażerze, takie jak imię, nazwisko, numer telefonu.

Operacje:

1. Dodawanie danych:

- a. Dodaj nowy lot do bazy danych.
- b. Dodaj nowego pasażera do bazy danych.

2. Aktualizacje:

- a. Zaktualizuj informacje o locie (np. zmień trasę lub datę).
- b. Zaktualizuj informacje o pasażerze (np. zmień numer telefonu).

3. Usuwanie:

- a. Usuń lot z bazy danych.
- b. Usuń pasażera z bazy danych.

4. Odczyt danych:

- a. Wyświetl listę wszystkich pasażerów.
- b. Dodaj możliwość wyszukiwania lotów na podstawie różnych kryteriów, takich jak trasa, data wylotu, dostępności miejsc, itp.

Zwróć uwagę na:

- W podpunkcie 4b: dostępność miejsc powinna się zmniejszać w momencie dodawania pasażera do danego lotu lub zwiększać w momencie jego usunięcia.
- Obsługę błędów - np. walidację danych wejściowych.
- Testowanie - sprawdź, czy operacje działają poprawnie.
- Dokumentację oraz instrukcję uruchomienia
- Możesz użyć dowolnej bazy danych

Spis treści

1.Opis3

2.Technologia3

3.Instrukcja uruchomienia3

 3.1.Baza danych.....3

 3.2.Testowanie endpointów w Postman.....3

 3.3.Testowanie aplikacji8

1. Opis

W ramach zadania rekrutacyjnego została wykonana prosta aplikacja składająca się z części frontendowej oraz backendowej, umożliwiająca zarządzanie danymi lotów i pasażerów.

2. Technologia

Wykorzystana technologia to:

- MySQL
- Spring Boot
- Java 17
- React.js

3. Instrukcja uruchomienia

Kod źródłowy aplikacji znajduje się na [github.com](https://github.com/kwiatkowskaaleksandra/recruitment_task.git): https://github.com/kwiatkowskaaleksandra/recruitment_task.git. Po sklonowaniu projektu i otwarciu go w IDE, w folderze ``src/main/frontend`` znajduje kod odpowiedzialny za frontend aplikacji. Natomiast w ``src/main/java`` znajduje się kod backendu. Testy jednostkowe napisane w JUnit 5 oraz Mockito znajdują się w folderze ``src/test/java``.

3.1. Baza danych

Konfiguracja bazy danych znajduje się w pliku `src/main/resources/application.properties`, gdzie należy ustawić własną nazwę użytkownika i hasło. Przy pierwszym uruchomieniu aplikacji, baza danych zostanie zainicjowana i zostaną utworzone odpowiednie tabele. W folderze `resources` znajduje się także plik `baza.sql` z przykładowymi danymi, który można zaimportować do bazy danych.

3.2. Testowanie endpointów w Postman

Wszystkie endpointy zostały przetestowane w Postman. W poniższym linku zostały przedstawione przykłady dla każdego z nich.

Endpointy: <https://documenter.getpostman.com/view/21395963/2sA3BobCYp>

Ponadto, wszystkie żądania kontrolera zostały opisane w postaci komentarzy JavaDoc.

- Przykład dodawania nowego lotu:

```
@PostMapping("/addNewFlight")
ResponseBody<Flight> addNewFlight(@RequestBody FlightRequest flightRequest) {
    return ResponseEntity.ok(flightService.addNewFlight(flightRequest));
}
```

The screenshot displays a REST client interface for a POST request to `http://localhost:8080/api/flights/addNewFlight`. The request body is a JSON object with the following fields: `flightNumber` ("L0126"), `availableSeats` (30), `departureDate` ("2025-03-20T09:30:00"), `flightDuration` ("10:20:00"), `departureCity` ("Kielce"), `arrivalCity` ("Opole"), and `intermediateAirports` (an array containing one object with `airportName` "Okecie Warszawa").

The response body is a JSON object with the following fields: `idFlight` (2), `flightNumber` ("L0126"), `availableSeats` (30), `departureDate` ("2025-03-20T09:30:00"), `flightDuration` ("10:20:00"), `flightRoute` (an object with `idFlightRoute` (2), `departureCity` ("Kielce"), `arrivalCity` ("Opole"), and `intermediateAirports` (an array containing one object with `idIntermediateAirport` (2) and `airportName` "Okecie Warszawa")), and `passengers` (null).

```
1 {
2   "flightNumber": "L0126",
3   "availableSeats": 30,
4   "departureDate": "2025-03-20T09:30:00",
5   "flightDuration": "10:20:00",
6   "departureCity": "Kielce",
7   "arrivalCity": "Opole",
8   "intermediateAirports": [
9     { "airportName": "Okecie Warszawa" }
10  ]
11 }
```

```
1 {
2   "idFlight": 2,
3   "flightNumber": "L0126",
4   "availableSeats": 30,
5   "departureDate": "2025-03-20T09:30:00",
6   "flightDuration": "10:20:00",
7   "flightRoute": {
8     "idFlightRoute": 2,
9     "departureCity": "Kielce",
10    "arrivalCity": "Opole",
11    "intermediateAirports": [
12      {
13        "idIntermediateAirport": 2,
14        "airportName": "Okecie Warszawa"
15      }
16    ]
17  },
18   "passengers": null
19 }
```

Wszystkie wprowadzane dane są odpowiednio walidowane. Każde z pól musi zostać uzupełnione.

Dodatkowo:

- numer lotu musi zaczynać się od „LO”, a jego długość musi wynosić 5 znaków,
- data wylotu nie może być podana z przeszłości

Przyjmowane dane to:

- numer lotu
- ilość dostępnych miejsc
- data i czas wylotu
- długość trwania lotu
- miejsce startu lotu
- miejsce docelowe lotu
- opcjonalnie przesiadki w trakcie lotu

- Pobranie wszystkich lotów

```
@GetMapping("/getAll")
ResponseEntity<Page<Flight>> getAll(@RequestParam int page, @RequestParam int size)
{
    Pageable pageable = PageRequest.of(page, size);
    return ResponseEntity.ok(flightService.getAllFlights(pageable));
}
```

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:8080/api/flights/getAll?page=0&size=5
- Params:** Query Params table with columns Key and Value.
- Body:** JSON response in Pretty view.

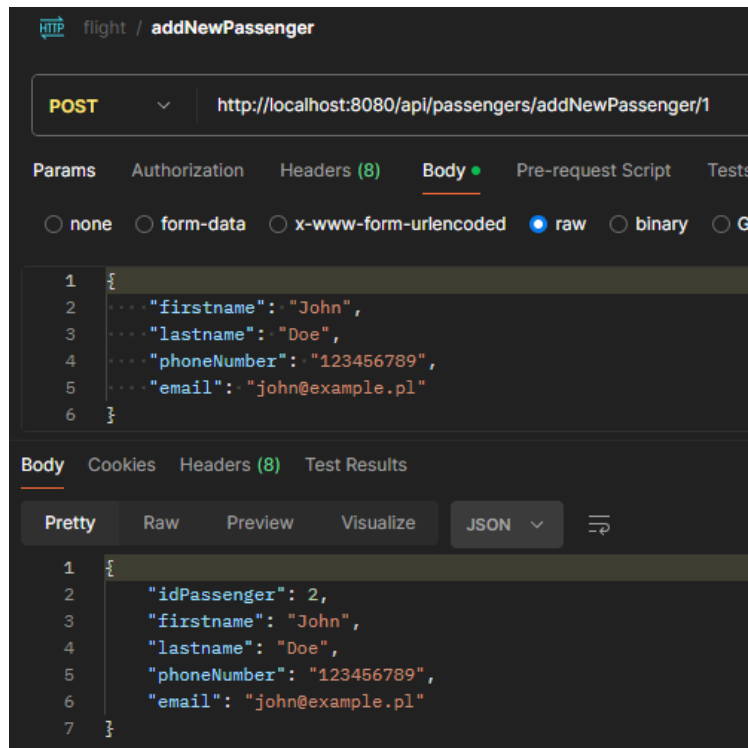
Key	Value
page	0
size	5

```
{
  "content": [
    {
      "idFlight": 1,
      "flightNumber": "L0126",
      "availableSeats": 29,
      "departureDate": "2025-03-20T09:30:00",
      "flightDuration": "10:20:00",
      "flightRoute": { ... },
      "passengers": [ ... ]
    },
    {
      "idFlight": 2,
      "flightNumber": "L0126",
      "availableSeats": 130,
      "departureDate": "2025-03-20T08:30:00",
      "flightDuration": "09:45:00",
      "flightRoute": { ... }
    }
  ]
}
```

Jako parametry przyjmuje page i size, które służą do paginacji.

- Dodawanie pasażera do listy pasażerów

```
@PostMapping("/addNewPassenger/{idFlight}")
public ResponseEntity<Passenger> addNewPassenger(@RequestBody Passenger passenger,
@PathVariable Long idFlight) {
    return ResponseEntity.ok(passengerService.addNewPassenger(passenger, idFlight));
}
```



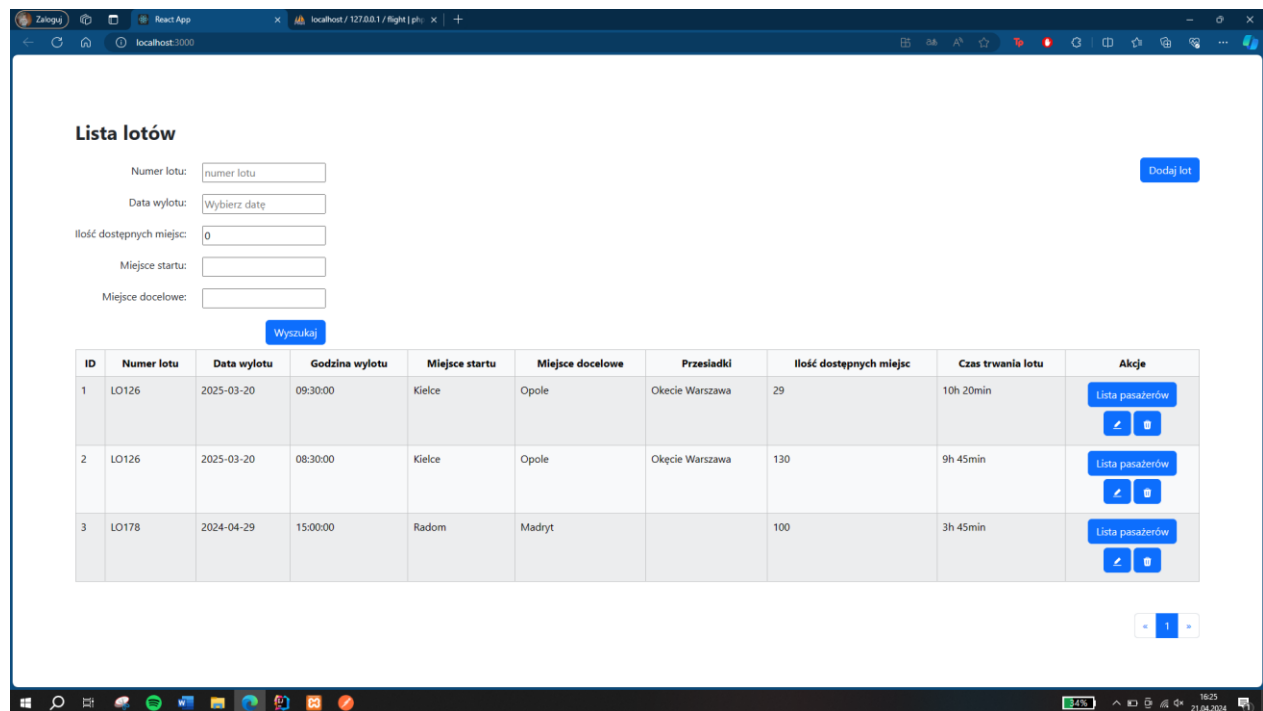
Przyjmuje jako parametr id lotu. Wprowadzone dane są walidowane w serwisie. Każde z pól musi zostać uzupełnione, ponadto numer telefonu musi mieć długość 9 znaków, a adres email musi zawierać znak @. Jeśli któreś z danych jest źle wprowadzone, zostaje rzucony odpowiedni wyjątek z wiadomością. Jeśli pasażer zostanie poprawnie dodany, ilość wolnych miejsc w locie jest zmniejszone o 1.

3.3. Testowanie aplikacji

Aby uruchomić aplikację frontendową, pobrać Node.js w wersji 21.0.0, a następnie należy przejść do folderu `src/main/frontend` za pomocą konsoli i uruchomić następujące polecenia:

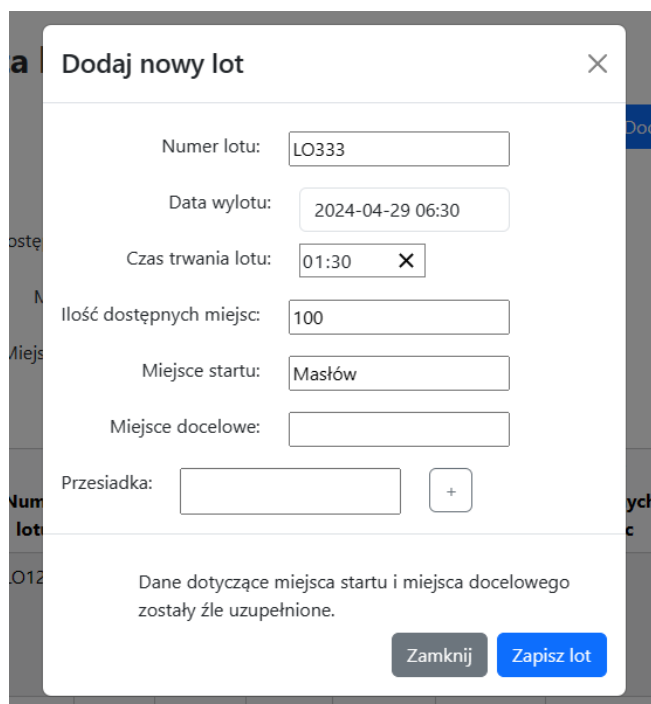
- `npm install`
- `npm start`

Aplikacja uruchomi się w przeglądarce.



Wyświetla się lista wszystkich lotów z użyciem paginacji. Po kliknięciu w 'Dodaj lot', otworzy się okno, gdzie należy podać dane.

W przypadku błędnie wprowadzonych danych, użytkownik zostanie poinformowany odpowiednim komunikatem.



Dodaj nowy lot

Numer lotu: LO333

Data wylotu: 2024-04-29 06:30

Czas trwania lotu: 01:30 ✕

Ilość dostępnych miejsc: 100

Miejsce startu: Masłów

Miejsce docelowe:

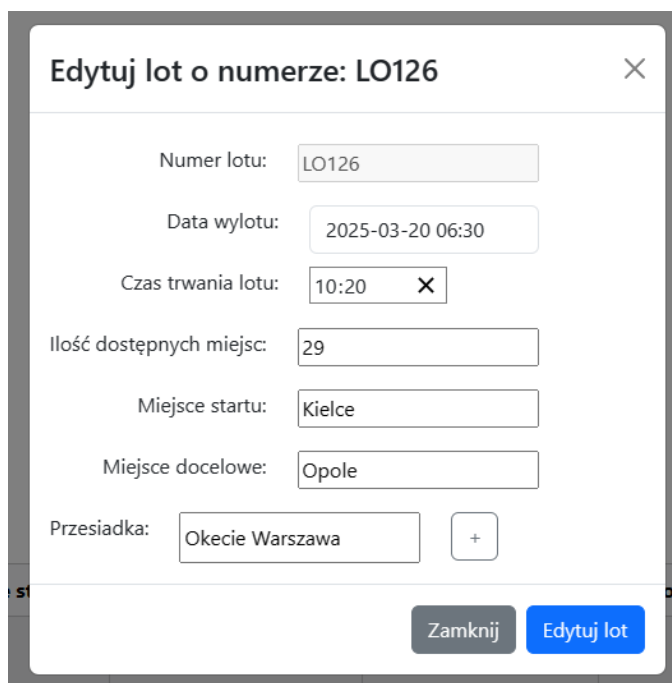
Przesiadka: +

Dane dotyczące miejsca startu i miejsca docelowego zostały źle uzupełnione.

Zamknij Zapisz lot

Po prawidłowym zapisaniu lotu, strona zostanie odświeżona i dodany lot pojawi się na liście.

Każdy lot można dodatkowo edytować, usunąć i wyświetlić listę pasażerów.



Edytuj lot o numerze: LO126

Numer lotu: LO126

Data wylotu: 2025-03-20 06:30

Czas trwania lotu: 10:20 ✕

Ilość dostępnych miejsc: 29

Miejsce startu: Kielce

Miejsce docelowe: Opole

Przesiadka: Okęcie Warszawa +

Zamknij Edytuj lot

Podczas edycji dane są walidowane w taki sam sposób jak podczas dodawania lotu.

Lista pasażerów lotu o numerze: LO126

[Dodaj pasażera](#)

ID	Imię	Nazwisko	Numer telefonu	Email	Akcje
1	John	Doe	123456789	john@example.pl	✎ ✖
2	John22	Doe22	123456789	johnDoe@example.pl	✎ ✖

[<](#) [1](#) [>](#)

Do każdego lotu jest możliwość wyświetlenia listy pasażerów. Aby dodać nowego pasażera należy kliknąć przycisk “Dodaj pasażera”. Zostanie wyświetlone odpowiednie okno, gdzie należy wprowadzić wszystkie dane. W przeciwnym wypadku, użytkownik zostanie poinformowany o błędzie.

Dodaj nowego pasażera

Imię:

Nazwisko:

Email:

Numer telefonu:

Podano błędny numer telefonu.

Zamknij

Zapisz pasażera

Po prawidłowym dodaniu, strona zostanie odświeżona, a na liście pasażerów pojawi się nowo dodany.

Tak jak w przypadku lotów, dla każdego z pasażerów jest możliwość edycji i usunięcia.

Na liście lotów, znajduje się także wyszukiwarka, gdzie po dodaniu filtrów zostanie wyświetlona lista lotów spełniająca wymagania.

