Pavlo Liulia
Nr albumu: 132796
I EF-AA/DU, grupa L04

Aplikacje internetowe

System do rezerwacji biletów lotniczych

Repozytorium backendowe - https://github.com/desfero/booking

Repozytorium frontendowe – https://github.com/desfero/booking-fr

Link do aplikacji: https://desfero.github.io/booking-fr/search

GraphQL: https://air-booking.herokuapp.com/graphql

1. Użyte technologie

Backend

GraphQL – Jeżyk zapytań stworzony przez Facebook jako zamiana standardowemu REST.

Express – Framework do tworzenia elastycznych aplikacji internetowych.

Node.js - środowisko uruchomieniowe zaprojektowane do tworzenia wysoce skalowalnych aplikacji internetowych, szczególnie serwerów www napisanych w języku JavaScript.

MongoDB - otwarty, nierelacyjny system zarządzania bazą danych napisany w języku C++.

Mongoose – ODB (Object-Document Mapper) dla bazy danych MongoDB

Heroku - platforma chmurowa stworzona w modelu Paas (*Platform as a Service*) obsługująca kilka języków programowania.

Github Pages – hostowanie frontendu

mLab MongoDB – serwer bazy danych

TravisCI – system CI

Frontend

Angular2, RxJS, Bootstrap 4

2. Struktura bazy danych

Aplikacja jest oparta o nierelacyjną bazę danych MongoDB i składa się z następujących kolekcji:

- Schedule – przechowywuje informacje o rozkładach lotów
- Booking – przechowywuje informacje o rezerwacjach użytkowników
- Users – przechowywuje informacje o użytkownikach

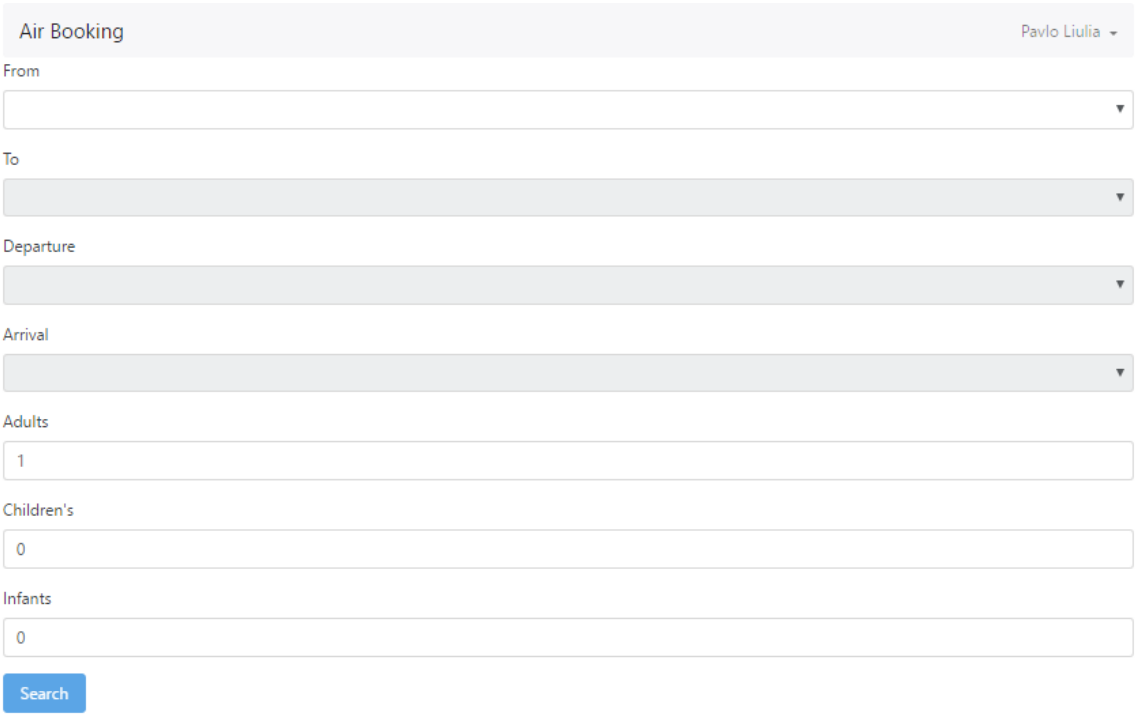
3. Uruchomienie

Aplikacja jest podzielona na dwa osobne projekty: booking i booking-fr.

Do poprawnego działania aplikacji potrzebne jest środowisko node.js oraz baza danych MongoDB.

1. Zainstalować zależności backedn-u za pomocą polecenia ‘npm install’ z katalogu części odpowiedzialnej za backend (repozytorium booking)
2. Zainstalować zależności frontend-u za pomocą polecenia ‘npm install’ z katalogu części odpowiedzialnej za frontned (repozytorium booking-fr)
3. Uruchomić bazę danych poleceniem mongod
4. Uruchomić backend za pomocą polecenia ‘npm start’ z katalogu części odpowiedzialnej za backend (repozytorium booking)
5. Uruchomić frontend za pomocą polecenia ‘npm start’ z katalogu części odpowiedzialnej za frontend (repozytorium booking-fr)

4. Opis funkcjonalność do wyszukiwania lotów po stronie frontend



Rys.1. Strona do wyszukiwania lotów

Po stronie frontend-u za wyszukiwanie połączeń odpowiada komponent o nazwie *SearchComponent*.

Pobiera on dostępne rozkłady lotów z backendu za pomocą zapytania GraphQL i przypisuje do zmiennej *schedules*

```
const Schedules = gql`
  query Schedules {
    schedules {
      _id,
      departure,
      arrival,
      from,
      to,
      price,
      seats
    }
  }
`;
```

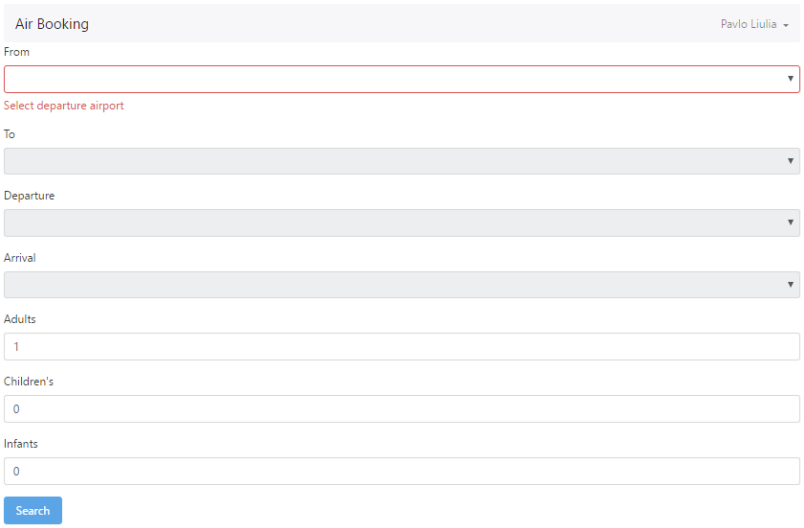
Rys.2. Zapytanie GraphQL

```
this.apollo.watchQuery({
  query: Schedules
}).subscribe(({data}) => {
  this.schedules = data.schedules;
});
```

Rys.3. Pobieranie rozkładów używaj zapytania GraphQL

Po zakończeniu pobierania mamy możliwość wyboru połączenia, którym jesteśmy zainteresowani.

Wszystkie pola są obowiązkowe do wpisania. Ponieważ nie możemy wybrać pola *To* bez wybrania *From* jest ustawiona walidacja, która blokuje możliwość wybrania wartości. Podobnie jest z polami *Departure* i *Arrival*.



Rys. 4. Walidacja

```
<div class="form-group" [class.has-danger]="from.errors && (from.dirty || from.touched)">
  <label for="from">From</label>
  <select id="from"
          class="form-control"
          required
          [(ngModel)]="selectedFrom"
          name="from"
          #from="ngModel">
    <option *ngFor="let schedule of schedules" value="{{schedule.from}}">{{schedule.from}}</option>
  </select>
  <div *ngIf="from.errors && (from.dirty || from.touched)"
       class="form-control-feedback">
    <span [hidden]="!from.errors.required">Select departure airport</span>
  </div>
</div>
```

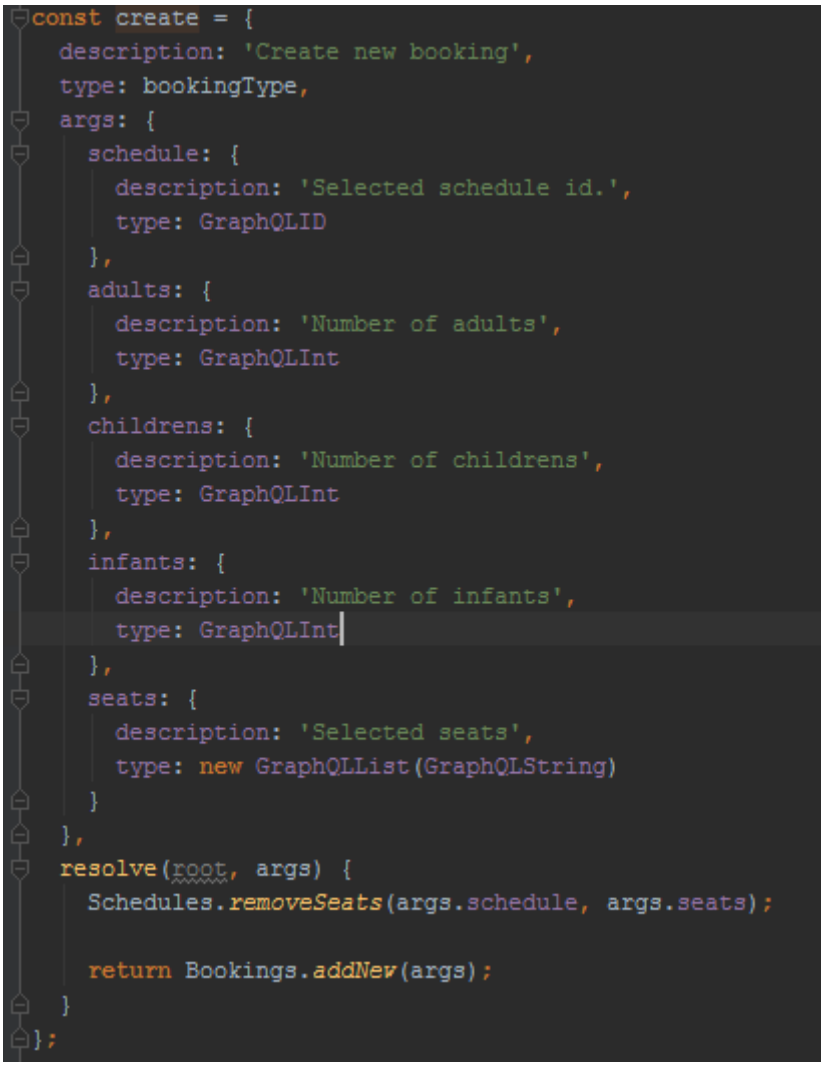
Po kliknięciu na przycisk Search wybrany przez nas kierunek zostaje zapisany w serwisie o nazwie StateService, który przechowywuje informacje o rezerwacji.

5. Opis funkcjonalności do zatwierdzenia lotu po stronie backendu

Dane o locie z frontendu są przesyłane za pomocą GraphQL i w pierwszej kolejności za odbiór tych danych odpowiada pole create, które posiada listę wszystkich parametrów (*args*), typ zwracany (*type*) oraz funkcję, która zostanie wykonana gdy backend otrzyma informacje o nowej rezerwacji.

Funkcja *resolve* wywołuje dwie inne funkcje, które posiadają następująca funkcjonalność.

1. Usuwamy wybrane siedzenia z listy dostępnych siedzeń dla danego lotu
2. Zapisujemy informacje o rezerwacji



Rys. 5. Przechwytywania zapytania GraphQL

removeSeats wyszukuje siedzenia w kolekcji i usuwa.

```
static removeSeats(id, seats) {
  Schedule.findById(id, function (err, schedule) {
    schedule.seats = schedule.seats.filter(s => !seats.includes(s));
    schedule.save();
  });
}
```

Rys.6. Usuwanie siedzeń

Z kolei funkcja *addNew* zapisuje do bazy danych informacje o rezerwacji wykorzystując poniższy model z mongoose:

```
import mongoose from 'mongoose';

const Schema = mongoose.Schema;

const BookingSchema = new Schema({
  schedule: {
    type: Schema.Types.ObjectId,
    ref: 'Schedule',
    required: true
  },
  owner: {
    type: Schema.Types.ObjectId,
    ref: 'User',
    required: false
  },
  adults: {
    type: Number,
    required: true
  },
  childrens: {
    type: Number,
    required: true
  },
  infants: {
    type: Number,
    required: true
  },
  seats: [{type: String, required: true}]

});

export default mongoose.model('Booking', BookingSchema);
```

Rys. 7. Model mongoose odpowiedzialny za rezerwacje