Travel App

I. Descrierea funcționalităților aplicației

Travel App este o aplicație care se ocupă cu gestionarea unui itinerariu de vacanță. Aceasta ia în considerare detalii precum zbor, destinație, client, perioada sejurului și cazare. Aplicația poate fi potrivită pentru companiile de travel, aceasta punând la dispoziție următoarele funcționalități:

1.Clientul își poate alege destinația către care dorește să călătorească din lista de destinații pusă la dispoziție.

- Pentru a vedea lista de destinații se apelează endpoint-ul:
  - **GET:http://localhost:8080/destination**
- Pentru cazul în care administratorul aplicației dorește să obțină informații despre un client se apelează:
  - **GET:http://localhost:8080/client/{id}**

2. După parcurgerea listei de destinații și alegerea uneia, clientul își poate rezerva o vacanță:

- **POST:http://localhost:8080/holiday**

***Obs. Endpoint-ul primeste un request body de tip HolidayRequest ce are ca atribute numai startDate și endDate. Pe lângă request body, request-ul are nevoie si de 2 request params ce reprezintă id-ul clientului și id-ul destinației. În cazul în care id-ul destinației nu este valid se aruncă excepția cu mesajul "Destination with id – does not exist!". Dacă id-ul clientului nu este înregistrat atunci se face un redirect către endpoint-ul de înregistrare client cu mesajul "Before choosing a destination please register.":**

- **POST: http://localhost:8080/client/**

3. După ce călătoria a fost rezervată, utilizatorul poate continua și își poate selecta atât cazarea dorită cât și zborurile potrivite pentru el:

- Pentru vizualizarea tuturor cazărilor aflate în acea locație:
  - **GET:http://localhost:8080/accommodation/destination/{destinationName}**

***Obs. În momentul afișării cazărilor vor fi selectate doar acele cazări cu o capacitate mai mare decât 0. În momentul în care un client își alege o cazare, capacitatea acesteia scade cu o unitate, iar în momentul în care clientul renunță aceasta crește.**

- Pentru vizualizarea tuturor zborurilor care au ca destinație sau punct de plecare numele dat ca path variable:
  - **GET: http://localhost:8080/flight/destination/{destinationName}**
- Pentru a rezerva cazarea acelei vacanțe:
  - **PUT: http://localhost:8080/holiday/{id}/accommodation**

- Pentru a rezerva un zbor:
    - **PUT: http://localhost:8080/holiday/{id}/flight**

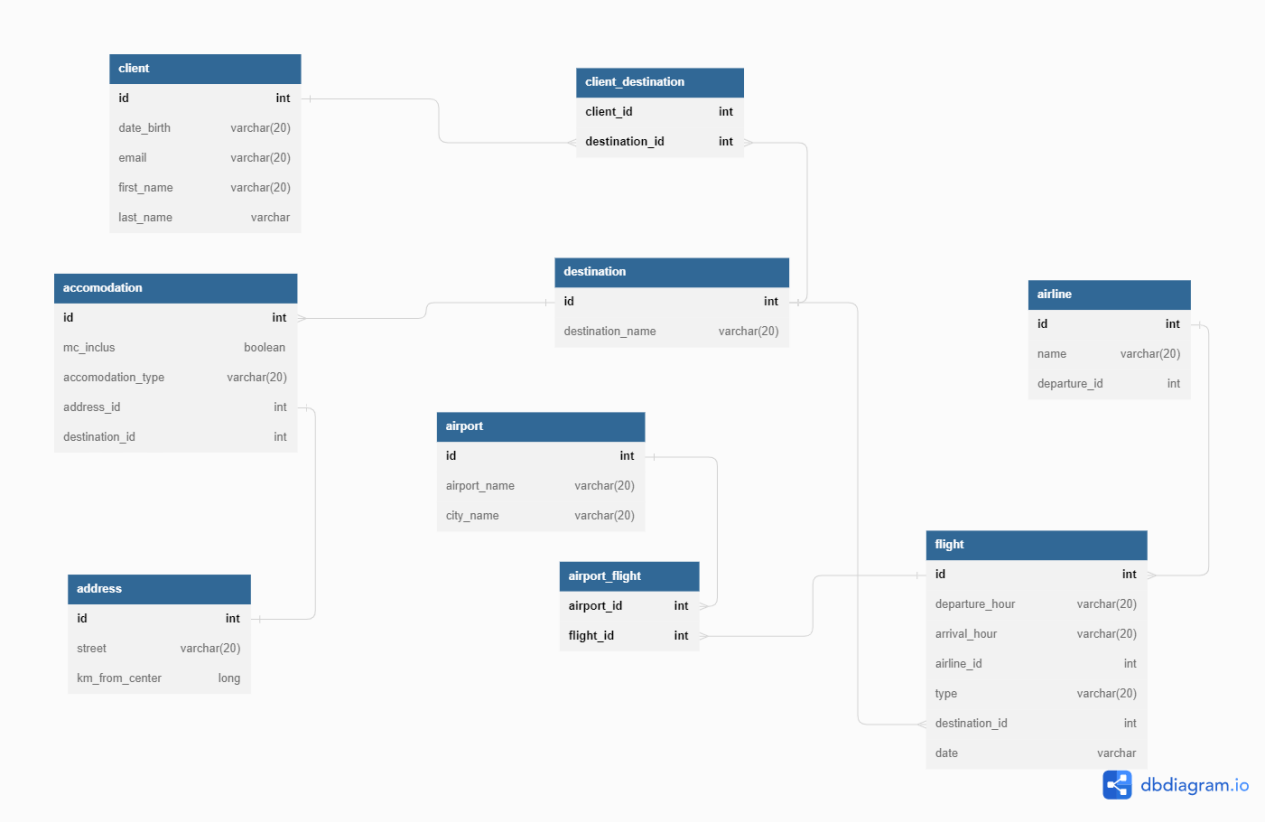
4. În cazul în care clientul se răzgândește în privința zborurilor sau cazărilor, acesta le poate șterge din vacanța rezervată.

- Ștergere zbor din lista de zboruri rezervate în funcție de id-ul din request param (pentru cazul în care zborul nu a fost rezervat se afișează un mesaj):
    - **DELETE:http://localhost:8080/holiday/{id}/flight**
- Ștergerea cazării din vacanța rezervată:
    - **DELETE: http://localhost:8080/holiday/{id}/accommodation**

5. Dacă utilizatorul s-a răzgândit și își dorește anularea vacanței, poate face acest lucru accesând:

- **PUT: http://localhost:8080/holiday/cancellation/{id}**

II. Diagrama relațională



III. Descrierea entităților și serviciilor

Entități

În vederea realizării aplicației s-au realizat 7 entități după cum urmează:

1. Accommodation
    1.1. Id
    1.2.AccommodationType
    1.3.Name
    1.4.PricePerNight
    1.5.CheckInHour
    1.6.CheckOutHour
    1.7.Capacity
    1.8.Destination
2. Address
    2.1.Accommodation
    2.2.KmFromCenter
    2.3.StreetAndNumber
3. Airport
    3.1.Id
    3.2.AirportName
    3.3.Flights
4. Client
    4.1.Id
    4.2.FirstName
    4.3.LastName
    4.4.BirthDate
    4.5.Holidays
5. Destination
    5.1.Id
    5.2.DestinationName
    5.3.Accommodations
    5.4.Holidays
6. Flight
    6.1.Id
    6.2.Airline
    6.3.Destination
    6.4.DepartureHour
    6.5.ArrivalHour
    6.6.Date
    6.7.Price
7. Holiday (reprezinta relatia de many to may între destinație și client)
    7.1.Id

7.2.Client
7.3.Destination
7.4.FirstDay
7.5.EndDay
7.6.Accommodation
7.7.Flights
7.8.IsCanceled

Împreună cu:

1. HolidayRequest
    1.1. Reprezintă DTO-ul corespunzător lui Holiday în vederea rezervării unei vacanțe; conține numai atributele startDate și endDate.
2. AirlineType
    2.1.Reprezintă compania aeriană al unui zbor .(Wizzair, JetBlue etc.)
3. AccommodationType
    3.1.Reprezintă tipul de cazare.(Hotel,motel,cabană,camping etc.)

Servicii

Pentru fiecare entitate descrisă mai sus s-a realizat câte o interfață după modelul: NumeEntitateService care moștenește interfața CrudService<T>. Interfața CrudService are rolul de a reține toate metodele de tip CRUD de bază, pentru a evita scrierea lor în fiecare clasă. Interfața NumeEntitateService este implementata de clasa NumeEntitateServiceImpl care moștenește și clasa AbstractService<T>. Aceasta simulează clasele de service, având atributul de JpaRepository<T,Long> și implementând la rândul ei interfața CrudService<T> căreia îi suprascrie metodele. Astfel avem:

1. HolidayService și HolidayServiceImpl
2. AccommodationService și AccommodationServiceImpl
3. AddressService și AddressServiceImpl
4. ArportService și AirportServiceImpl
5. ClientService și ClientServiceImpl
6. DestinationService și DestinationServiceImpl
7. FlightService și FlightServiceImpl
8. HolidayService și HolidayServiceImpl

Repositories

Pentru fiecare entitate înregistrată în baza de date s-a creat o interfață NumeEntitateRepository care extinde JpaRepository<T,Long>. Fiecare interfață este injectată în clasele de NumeEntitateServiceImpl.

1. AccommodationRepository
2. AddressRepository
3. AirportRepository
4. ClientRepository

5. DestinationRepository
6. FlightRepository
7. HolidayRepository

Pachetul exceptions

Conține 3 excepții customizate care se apelează în funcție metodele care se apelează și care, în general, nu găsesc obiectele dorite în baza de date. Gestionarea excepțiilor se face în ControllerAdvice-ul definit și anume GeneralExceptionHandler.

1. HolidayAlreadyCancelledException
2. EntityNotFoundException
3. ClientNotRegisteredException

Pentru exceptiile NoSuchElement,EmptyResultDataAccess sau EntityNotFound se întoarce un HttpStatus Not Found.

Pentru HolidayAlreadyCancelledException se întoarce un HttpStatus de Bad Request.

Pentru ClientNotRegisteredException se întoarce un HttpStatus Ok, însă se afișează mesajul `"Before choosing a destination please register."` Și se face un redirect către metoda POST de înregistrare a unui nou client.

Controllers

1. AccommodationController conține:
    1.1. Două request-uri de GET (getAllByDest și getById) care returnează toate cazările de la o anumită locație, respectiv cazarea găsită după id.
    1.2. Un request POST care adaugă p cazare.
2. ClientController conține:
    2.1. Un request POST care adaugă un client, respectiv GET care returnează un client după id.
3. DestinationController conține:
    3.1. O metodă POST care adaugă o destinație.
    3.2. Două metode GET care returnează toate destinațiile, respectiv destinația găsită în funcție de id-ul dat.
    3.3. O metodă PUT care modifică un obiect de tip Destination.
    3.4. O metodă DELETE care șterge o destinație în funcție de id.
4. FlightController conține:
    4.1. O metoda POST care adaugă un zbor nou.
    4.2. 3 metode GET care returnează: toate zborurile dintr-o anumită perioadă, toate zborurile spre o anumită destinație, zborul căutat în funcție de id.
5. HolidayController conține:
    5.1. O metodă POST care adaugă o vacanță nouă.
    5.2. Două metode DELETE care șterg cazarea,respectiv zborul vacanței.

5.3. Doua metode GET care returnează toate vacanțele unui client, respectiv o vacanță în funcție de id-ul dat.
5.4. 3 metode PUT care anulează o vacanță, îi adaugă un zbor nou sau o cazare nouă.

Popularea bazei de date

Popularea bazei de date se realizează în clasa BootstrapClass care implementează interfața CommandLineRunner și suprascrie metoda run. Clasa are injectate toate repository-urile definite de noi, pentru fiecare entitate, în afară de repository-ul ClientRepository. La runtime se verifică dacă metodele findAll() returnează liste goale sau populate. În cazul în care listele sunt goale se creează și se salvează pe rand diferite obiecte în baza de date.