Full Stack Development Case Study - Java Developer 2

Introduction

Aim of the project is to implement a system for aviation industry. This system should be able to calculate all the possible routes from point A to point B to provide users better experience while booking their flights.

The requirements are the followings:

- A database for storing data should be initialized. Any database (PostgreSQL, MySQL, MSSQL or H2) is allowed to use.
- A Spring Boot Java REST API should be implemented.
- Hibernate framework should be used as an ORM framework to map Java classes and database tables.
- Swagger support should be provided, and Swagger UI should be reachable.

Database Requirements

In context of the project, the following entities should be maintained in database:

Locations

Represents the locations which will be displayed in “from” and “to” dropdown boxes while searching flights. Locations entities should contain the following fields:

- **Name**
- **Country**
- **City**
- **Location code**
    - For airports, location codes are the 3 characters long IATA codes. Example: **SAW** for Sabiha Gökçen Airport, **IST** for istanbul airport. (For searching IATA code of a particular airport: https://www.iata.org/en/publications/directories/code-search/)
    - For other locations, any coding pattern can be used (example: CCIST for İstanbul City Center)

Transportations

Represents the transportations from one location to another. Transportation entities should have the following fields:

- **Origin Location**
- **Destination Location**
- **Transportation Type:** FLIGHT, BUS, SUBWAY, UBER
- **Operating Days**: An array of integers representing on which days of the week the corresponding transportation is operated. The value of the array being [1, 3, 5, 6] means that the corresponding transportation is only active on Monday, Wednesday, Friday and Saturday.

Full Stack Development Case Study - Java Developer 2

API Requirements

A Spring Boot Java REST API should be implemented and combined with Hibernate for database integration. Through rest controllers, API needs to provide endpoints for:

- CRUD operations for locations
- CRUD operations for transportations
- Returning all the valid **route**s from one location to another on selected date.

Definition of Route

- A route can be defined as a sequence of available and connected transportations. Each route can have 3 connected transportations:
  - **Before flight transfer**: Optional. Transportation type must be other than FLIGHT
  - **Flight**: Mandatory. Transportation type must be "FLIGHT".
  - **After flight transfer**: Optional. Transportation type must be other than FLIGHT.
- To consider two transportations "connected", the arrival location of the prior transportation should match the departure location of the subsequent transportation.

  Examples:
  - A bus ride from **Taksim Square** to **İstanbul Airport** and a flight from **İstanbul Airport** to **London Heathrow Airport** are connected transportations. ✓
  - Funicular from **Kabataş Pier** to **Taksim Square** and a flight from **İstanbul Airport** to **London Heathrow Airport** are not connected transportations. ✗
- To consider a transportation "available", the selected date should be among the operating days of the transportation.

  Example:

  A flight from İstanbul Airport to London Heathrow Airport has operating days **[1, 3, 7]**

  If the paremeter "date" is selected as:
  - "12 March 2025", then the flight mentioned above is available. ✓
  - "11 March 2025" instead, then the transportation mentioned above is not available. Since 11 March is Tuesday which is not among the operating days of the flight. ✗

Limitations

A sequence of connected transportations cannot be considered as a valid route if:

- There are more than 3 transportations from origin to destination.
- There is no flight among them
- There are more than one flights among them
- There are more than one before flight transfers among them
- There are more than one after flight transfers among them

Full Stack Development Case Study - Java Developer 2

- The corresponding route has **ANY** transportation which is not available. Please note that this criteria is not only applied to flights, but also before flight and after flight transfers.

Valid route examples (considering all the transportations operate every day of the week):

- UBER ➞ FLIGHT ➞ BUS ✓
- FLIGHT ➞ BUS ✓
- UBER ➞ FLIGHT ✓
- FLIGHT ✓

Invalid route examples:

- UBER ➞ BUS ➞ FLIGHT ✗ (multiple before flight transfers)
- UBER ➞ BUS ✗ (no flight)
- UBER ➞ FLIGHT ➞ FLIGHT ✗ (multiple flights)
- FLIGHT ➞ FLIGHT ✗ (multiple flights)
- FLIGHT ➞ SUBWAY➞ UBER ✗ (multiple after flight transfers)

An example valid route “Taksim square -> Wembley Stadium” would look like this:

- A **bus ride** from **Taksim Square** to **İstanbul Airport**
- A **flight** from **İstanbul Airport** to **London Heathrow Airport**
- A **Uber ride** from **London Heathrow Airport** to **Wembley Stadium**

IMPORTANT: If there are multiple transfer options for the same route, all should be returned as if they are separate routes. Example:

- Taksim Square **(UBER)** ➞ Istanbul Airport ➞ London Heathrow Airport **(BUS)** ➞ Webley Stadium
- Taksim Square **(UBER)** ➞ Istanbul Airport ➞ London Heathrow Airport **(UBER)** ➞ Webley Stadium
- Taksim Square **(SUBWAY)** ➞ Istanbul Airport ➞ London Heathrow Airport **(BUS)** ➞ Webley Stadium
- Taksim Square **(SUBWAY)** ➞ Istanbul Airport ➞ London Heathrow Airport **(UBER)** ➞ Webley Stadium
- Taksim Square **(BUS)** ➞ Istanbul Sabiha Gökçen Airport ➞ London Heathrow Airport **(BUS)** ➞ Webley Stadium
- Taksim Square **(BUS)** ➞ Istanbul Sabiha Gökçen Airport ➞ London Heathrow Airport **(UBER)** ➞ Webley Stadium

Full Stack Development Case Study - Java Developer 2

Non-Functional Requirements

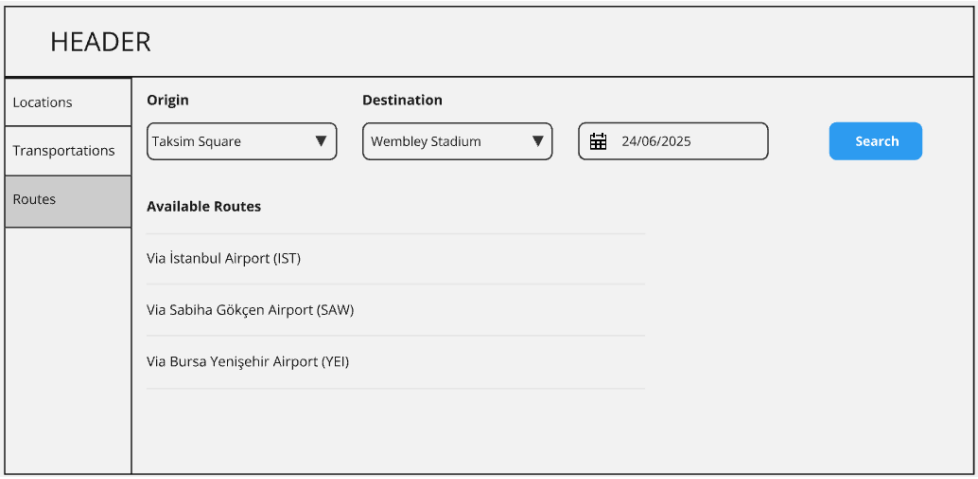
- For endpoints which may receive high load, a cache support should be added. Any caching tool is allowed to use (Redis, Couchbase etc.).
- Spring Boot or JUnit tests should be added for service classes.
- The project should be Dockerized. Both docker-compose and Dockerfile must be provided.
- Login and authentication flow (Implementation of registration flow can be skipped. Users can be inserted to database manually instead).

  There should be 2 different user types:
  - **Admins** who can call all the endpoints.
  - **Agencies** who can only call the route listing endpoint. If agencies try to call location end transportation endpoints, they should receive HTTP status code 403.

  If no authentication is provided in the request (HTTP Authorization header), then HTTP status code 401 should be returned.

Front-end Requirements

- A responsive SPA with header and side bar menu should be implemented in React.
- There should be 3 links in the side bar
  - “Locations” for handling CRUD opertaions of locations. This should only be visible for admins.
  - “Transportations” for handling CRUD opertaions of transportations. This should only be visible for admins.
  - “Routes” for listing all valid routes between two different locations. This should be visible for all types of users.
- In “Routes” page, origin and destination locations and trip date are selected on dropdown boxes. Then the available routes should be displayed in a listing view upon search (The following design is just a mock for suggestion, there is no restriction).



- After clicking an available route, route details should be displayed in a side panel (alternatives will also be accepted).

"Nice-to-Have"s

- Back-end validations on api.
- Instead of using "auto DDL" feature of Hibernate, a database migration tool like Liquibase can be added for managing database changes.
- Displaying the route details on a map by drawing some lines or arrows like the following. Design is up to you