DVT Interview Document

Take-Home Assignment: PNR Information Aggregator

Objective:
Build a small backend service in **Java using Vert.x** that aggregates booking-related information for a PNR. The goal is to evaluate your ability to implement reactive patterns, manage dependent requests, and handle errors gracefully.

Scenario:

You are tasked with creating a service that provides comprehensive **booking information** for a given Passenger Name Record (PNR) - which is a random 6-digit identifier. You are required to **create a REST endpoint** that retrieve that booking while leveraging **reactive programming for internal logic**.

You are expected to use a persistence (preferably MongoDB or Couchbase) to fetch data, a sample booking with two passengers is provided for you. However, the data for that booking needs to be aggregated from different sources. You are expected to map this data accordingly.

1 - Trip Information: Contains passenger details and flight information.

```
{

  "bookingReference": "GHTW42",
  "cabinClass": "ECONOMY",
  "passengers": [
    {
      "firstName": "James",
      "middleName": "Morgan",
      "lastName": "McGill",
      "passengerNumber": 1,
      "customerId": null,
      "seat": "32D"
    },
    {
      "firstName": "Charles",
      "lastName": "McGill",
      "passengerNumber": 2,
      "customerId": "1216",
      "seat": "31D"
    }
  ],

  "flights": [

    {
      "flightNumber": "EK231",
      "departureAirport": "DXB",
      "departureTimeStamp": "2025-11-11T02:25:00+00:00",
      "arrivalAirport": "IAD",
      "arrivalTImeStamp": "2025-11-11T08:10:00+00:00"
    }
  ]
}
```

2 - Baggage: Baggage allowance for the given booking.

```
{
  "baggageAllowances":
          [
            {
              "passengerNumber": 2,
              "allowanceUnit": "kg",
              "checkedAllowanceValue": 25,
              "carryOnAllowanceValue": 7
            },
            {

              "passengerNumber": 1,
              "allowanceUnit": "kg",
              "checkedAllowanceValue": 25,
              "carryOnAllowanceValue": 7
            }
          ]
}
```

3- ETicket Information: This is fetched per-passenger, Requires both **PNR and passenger number** from the booking information to retrieve the eticket. **One of the passengers does not have an eticket**, **you are expected to handle passengers without tickets accordingly**.

```
{
  "passengerNumber": 2,
  "ticketUrl": "emirates.com?ticket=someTicketRef"
}
```

Requirements:

1 - REST API Endpoint

Implement the following endpoint:

- GET /booking/{pnr}
  - Create this using Spring Boot's @RestController
  - Returns a JSON payload aggregating the following:
    - Trip information (from source 1)
    - Baggage information (from source 2)
    - Ticket information (from source 3, dependent on both PNR and passenger number)

2 - Reactive Programming

- Data transformation and aggregation logic for each source should be **non-blocking and independent** , wherever possible.

3 - Key Considerations

- Use **Spring Boot** for:
  - REST controllers (`@RestController`)
  - Configuration / Dependency injection
  - Exposing endpoints
  - Using either Spring Data MongoDB **or** reactive repository
- Use **Vert.x** for:
  - Async CRUD execution
  - Event-driven messaging (e.g., when a PNR is fetched, publish an event)
  - Non-blocking DB calls (optional if you use Vert.x Mongo client directly)
- Use **MongoDB** (preferred) or **CouchDB** for persistence.

Your final response should be in the following format:

```json
{
  "pnr": "GHTW42",
  "cabinClass": "ECONOMY",
  "passengers": [
    {
      "passengerNumber": 1,
      "fullName": "James Morgan McGill",
      "seat": "32D"
    },
    {
      "passengerNumber": 2,
      "customerId": "1216",
      "fullName": "Charles Mcgill",
      "seat": "31D",
      "ticketUrl": "emirates.com?ticket=someTicketRef"
    }
  ],
  "flights": [
    {
      "flightNumber": "EK231",
      "departureAirport": "DXB",
      "departureTimeStamp": "2025-11-11T02:25:00+00:00",
      "arrivalAirport": "IAD",
      "arrivalTImeStamp": "2025-11-11T08:10:00+00:00"
    }
  ]
}
```

3 - Bonus (Optional)

- Hide the data in the path param of your endpoint by using a passengers customer ID.
- Add a **Vert.x EventBus consumer** that listens for `"pnr.fetched"` events and logs or sends notifications.
- Add **circuit-breaking or retry** logic for failed database writes.
- Add **WebSocket** endpoint using Vert.x to broadcast real-time “pnr fetched” events.
- Add **containerization** (Dockerfile + docker-compose with MongoDB).

Evaluation Criteria

1. Correctness: Aggregated PNR information is returned correctly.
2. Reactive Implementation: Correct use of Vert.x patterns for dependent calls.
3. Error Handling: Graceful handling of any potential points of failure.
4. Code Quality: Clear, maintainable code.