Project Exercise – Ecommerce Order Processing

Read entire document at least once before starting out on any design and development.

Let's say you are part of a team that is planning to migrate a monolithic ecommerce platform from a legacy database engine with an outdated schema to a more modern database engine with an extensible schema and implement enhancements. At the same time, your team is thinking to follow a microservice architecture where you can back the appropriate database domains with Spring Boot based microservices (RESTful or otherwise).

A modern ecommerce platform can have many sets of microservices: customer, order, product, inventory, payment, shipping/delivery etc.
For this exercise, you are going to focus on the ***order microservice/s*** with following tasks:

1. Understand the following legacy order schema:

```
order_id
order_status
order_customer_id
order_item_name
order_item_qty
order_subtotal
order_tax
order_shipping_charges
order_total
order_payment_method
order_payment_date
order_payment_confirmation_number
order_billing_addressline1
order_billing_addressline2
order_billing_city
order_billing_state
order_billing_zip
order_shipping_addressline1
order_shipping_addressline2
order_shipping_city
order_shipping_state
order_shipping_zip
```

2. Design a new database schema that:
    a. follows the concepts of Database Normalization. *You remember that from your DBMS class. Sorry to take you back to those haunting memories. 😬*

b. provides an ability to have more than 1 item per order. *What order schema in 2020 limits to 1 item per order? ☺*
c. provides an ability to pay the order by accepting more than one payment transactions. For example, pay for an order amount using two credit cards (split the total). Make some assumptions about the billing address/es.
d. adds other metadata to the order: *createdDate*, *modifiedDate*. *What else?*
e. supports more shipping/delivery methods: In-store pickup, curbside delivery, ship to home, 3rd party delivery etc.

3. Develop one or more microservices to back this new database schema. Your new service/s has to support two types of clients:
   a. RESTful clients that work with single order object, e.g.
      i. get order by id
      ii. create an order
      iii. cancel an order

      These clients are customer-facing: web apps, mobile apps, bots etc. So, your service must support these clients via RESTful APIs.
   b. Clients that work with bulk order objects:
      i. create orders in bulk/batch,
      ii. update order status in bulk/batch.

      These clients are the internal services: POS Terminals, fulfilment systems, warehouse picking systems etc. You have an option to choose RESTful or async communication to accept data from these clients. *Get ready with your reasoning. ☺*

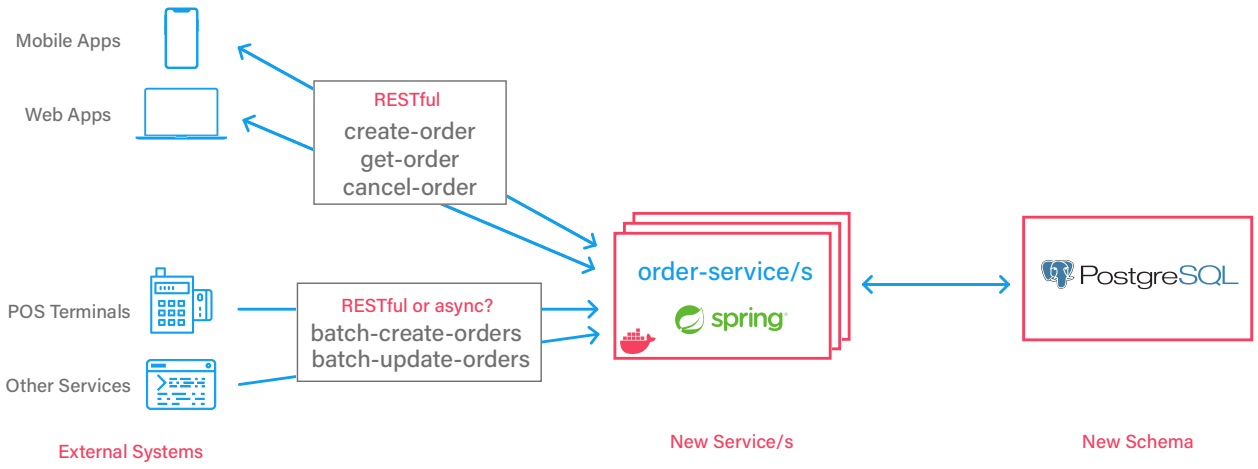

Figure: Diagram for the folks who like to see things on a whiteboard.

4. While developing your solution, think about following:
    a. Can a single RESTful API service handle all these requirements? Or, do you need more than one microservice to support these two types of clients? Are these different kinds of services: RESTful, batch, message queue based?
    b. How would you deal with network connectivity issues or data quality errors? Between client and your service? Between your service and the database?
    c. How would you handle *order_id* generation: autoincremented database sequencer? *Remember, you are dealing with distributed systems where other services can generate these identifiers on their own.*
    d. How would you improve performance of the bulk order processing? While you are at it, prioritize customer-facing apps' performance over the internal client services' performance. *Don't want to lose a paying customer because of a slow API.*
    e. How would you make your life easier for debugging any issues with your services? Logging? Custom Metrics? *Once deployed to production, you would be asked: "What happened to that order by customer xyz?".*
    f. What if this migration is going to take months? How do you keep the current business running while the new development is going on? Do you want to keep both databases online during this time?

Requirements:

1. Use Java8 (or higher version of Java), or Kotlin for the programming language.
2. Spring Boot, Spring Data JPA, and other Spring modules for the microservices.
3. Design new schema keeping PostgreSQL in the mind.
4. For the RESTful API, use Swagger to document your endpoints. Use appropriate HTTP verbs and status codes. Assume no AuthN or AuthZ is required for the API.
5. Implement unit and/or integration tests.
6. Containerize your services using Docker. Use docker-compose if required.
7. If you are thinking about *async communication* patterns, use Apache Kafka.
8. See, if you can follow few of the 12 Factor App Principles (https://12factor.net).

As you have noticed by now, this is not a typical take-home exercise. It's pretty open-ended with no single right or wrong solution. Many requirements are unknown or ambiguous. IRL, while working in a team, you would be asking questions to your team, architects, and business stakeholders for clarity. For this exercise, make assumptions to the best of your understanding. Also, you may not be able to complete all the requirements, and that's okay. We are hiring for many levels: Associate to Senior level Engineers. So, give your best shot with the time you have. Give your imagination free rein. All the best and we are excited to discuss your implementation in the interview.