Coding Challenge

The focus of this code challenge is on the backend. You need to implement the initial steps in a payments processing application. The initial version is basically a CRUD app, but in later stages more complex business logic and validations will be added.

Make sure the code sent runs out of the box. Manage the project with Maven and structure it accordingly. Provide a docker-compose file that can be used to run the application and the database. Keep the tests as self-contained as possible: this means that no external dependencies should be manually installed for the developer to run the tests. Include a README.txt with the project, which should contain step-by-step instructions on how to (1) run the application using docker-compose, (2) configure the client, (3) build the project from Maven, and (4) run the client. Put the project under source control on GitHub and email us the repo link.

Challenge:

We need to implement a backend application that will be used to accept Payments sent by Customers.

1) Design a REST API.

A Payment is a Resource defined with:
• Currency and amount
• Originator (name, ID)
• Beneficiary (name, ID)
• Sender (account type, account number)
• Receiver (account type, account number)
• Status (CREATED, SENT, ACCEPTED, REJECTED)

The API should provide operations to **accept a payment** and **query its status**

2) Create a Spring Boot or Quarkus Java backend that implements this API.

The backend should persist the above data to a database. Decide which DB and framework/library to use to persist and read the data.

3) Create a Spring Boot or Quarkus client Java application that simulates what a real-life Supplier would do, by generating a dozen or so payments and POSTing each payment to the API.

4) Whenever a payment is `POSTed`, the application needs to:

a) persist it locally, with status CREATED

b) Put a message on a Kafka Topic representing a "Transaction created" event with the transaction as a payload.

c) return a status code of 202 to the caller

If there is any error while persisting the payment or when posting to the topic, the application should return a status code of 500 to the caller.
The operation must be idempotent to allow retries from the client, using a client-generated Idempotent Key.

5) Create unit or integration test cases, or both, wherever logically it makes sense in your opinion.

Questions:

1) Our sales team tells us that some of the customers will be sending around 1 million payments per day, mostly during business hours, and that they will perform around 10 million queries per day. Which steps will you take to make sure the application can handle this load?

2) How does your system guarantee that when accepting a payment the payment is saved locally and the message is sent to the topic? What would happen if a container running the application is restarted after saving the payment to the DB and before sending the message to the topic?