Web Applications 2

Lab 5: reactive programming, event driven architecture

Learning Objectives

By successfully completing this lab, students will develop the following skills:

- Learn how to implement a reactive server
- Use asynchronous messaging using kafka

Description

The transport company offers the opportunity to buy ordinal tickets and season ones at an advantageous price. One of those, for example, called “weekend pass”, destined to its under 27 customers, allows travel without limits on Saturdays and Sundays to reach the major national destinations. In order to integrate the payment and catalog feature to the existing architecture, two microservices must be added to the existing architecture. The first one is ticketCatalogueService which is in charge of managing the sales process, providing the list of ordinal tickets and season tickets. The service is accessed via a REST API, which exposes various end-points, some accessible to everybody, some restricted to authenticated users and others accessible only to users having administrative privileges. This service cooperates with TravelerService, in order to retrieve additional information of the user, which are used to check compatibility constraints with sales available. In case the acquisition process can be performed, it sends back an order with status PENDING. From this point two things happens: the pending order is saved in the database, and send payment informations to the PaymentService. The latter is in charge of sending payment informations to the bank through a kafka topic, replying back the result in a different topic. The following endpoints must be implemented by TicketCatalogue:

- GET /tickets → returns a JSON representation of all available tickets. Those tickets are represented as a JSON object consisting of price, ticketId, type (ordinal or type of pass). A ticketAcquired is represented as a JSON object consisting of the fields “sub” (the unique ticketID), “iat” (issuedAt, a timestamp), “validfrom” (useful for subscriptions), “exp” (expiry timestamp), “zid” (zoneID, the set of transport zones it gives access to), type (ordinal, weekend pass, etc), “jws” (the encoding of the previous information as a signed JWT) [Note that this JWT will be used for providing physical access to the train area and will be signed by a key that has nothing to do with the key used by the LoginService]
- POST /shop/{ticket-id} →it accepts a json containing the number of tickets, ticketId, and payment information (credit card number, expiration date, cvv, card holder). Only authenticated users can perform this request. In case those tickets have age restrictions, it asks the TravellerService the user profile in order to check if the operation is permitted. In case it is, it saves the order in the database with the PENDING status, then it transmits the billing information and the total cost of the

order to the payment service through a kafka topic, and it returns the orderId. When the Kafka listener receives the outcome of the transaction, the status of order is updated according to what the payment service has stated and, if the operation was successful, the purchased products are added to the list of acquired tickets in the TravellerService.
The client to check the order result, must do polling to check the outcome.

- GET /orders →Get list of all user orders
- GET /orders/{order-id} →Get a specific order. This endpoint can be used by the client to check the order status after a purchase.
- POST /admin/tickets → Admin users can add to catalog new available tickets to purchase.
- GET /admin/orders → This endpoint retrieves a list of all orders made by all users
- GET /admin/orders/<user-id>/ →Get orders of a specific user

The paymentService, when receives a payment request in a kafka topic, sends back in another topic the outcome. It saves in a database the transactionID
Following endpoints must be implemented:

- Get /admin/transactions → get transactions of all users
- Get /transactions → get transactions of the current user

Those microservices must be reactive.

Hints

https://dev.to/magnuspedro/setting-up-spring-kafka-with-kotlin-53ea

Submission rules

Download a copy of the zipped repository and upload it in the “Elaborati” section of “Portale della didattica”. Label your file “Lab5-Group<N>.zip” (one copy, only - not one per each team member).
Work is due by Friday May 27, 23.59 for odd numbered teams, and by Friday June 3, 23.59 for even numbered ones.