KAFKA PRODUCER & READER APP

Kafka Producer app is responsible of reading the click events from the Json file provided. Firstly a safe KafkaProducer is created with acknowledge all, idempotence true and other configurations set. The reason of creating a safe KafkaProducer was to not lose any event since they were not coming from a data storage but from users.

Producer reads file line-by-line using Stream interface and each line is treated as a live data coming from an application. Once Producer reads a line the current timestamp is captured to bind that value to the "Product View" event, which will then imply the date and time the click occurred by the user.

Requirement of publishing one event in a second is achieved by using Thread.sleep functionality and this wait call is executed after the timestamp is captured to prevent losing the real time of the click event.

In order to append the timestamp to the "Product View" event, the line read is parsed by JsonParser to a JsonObject and "click_timestamp" key-value pair is appended to it. Once event data is prepared then producer sends it to the Kafka.

Kafka Consumer app is responsible of reading data in Kafka and write it to a PostgreSQL database. Firstly, a KafkaConsumer is created which implements Runnable interface. Consumer reads each data in Kafka and extracts the Json type data's fields, then these fields are inserted to the columns of the custom table created inside the target PostgreSQL database. All the attributes of the Json data are stored inside the table. This custom table is called "product_view_history".

product_view_history
event
[PK] message_id
user_id
product_id
source
click_timestamp

SPARK ETL PROCESS APP

ETL process is responsible of reading data from source PostgreSQL data-db, transform data as needed and save transformed data into target PostgreSQL database called bestseller-db. Spark is chosen to implement the ETL process since it is high in speed and easy to use.

In order to find the bestseller products, Orders, Order_Items, Products tables are loaded as DataFrames to Spark.

For finding the bestseller ten products of each category, strategy below is followed,

1) Find each purchase of users by inner joining orders and order_items tables and selecting distinct user_id and product_id pair

2) Append category_id column to the results gathered on former step by inner joining products table and count user_id for each product_id and category_id pair

3) Append the former result with the row numbering for each unique category_id

4) Select product_id, category_id, sale_amount where row number does not exceed ten which is equivalent to selecting top ten bestseller product of each category.

Relevant SQL of the strategy,

```
SELECT product_id, category_id
FROM (
  SELECT ROW_NUMBER() OVER (PARTITION BY category_id ORDER BY sale_amount DESC) AS rank, sale_amount_of_product.*
  FROM (
    SELECT products.product_id, category_id, COUNT(user_id) AS sale_amount
    FROM (
    SELECT DISTINCT user_id, OI.product_id
    FROM orders AS O
    INNER JOIN order_items AS OI ON O.order_id = OI.order_id
    ) AS all_purchases
    INNER JOIN products ON all_purchases.product_id = products.product_id
    GROUP BY products.product_id, category_id
    ORDER BY sale_amount DESC
  ) AS sale_amount_of_product
) AS sale_ranking
WHERE sale_ranking.rank <= 10
```

Table retrieved from that strategy is implemented as Spark-SQL and it is stored in the target database in a table called bestseller_product.

For finding the overall bestseller top ten products, bestseller_product table is used. Ordering this table by sale_amount column descending and selecting first ten rows yields overall bestseller products.

Bestseller_product table is used by the recommender REST API to recommend user products that share common category_id with products he/she viewed before.

bestseller_product
product_id
category_id
sale_amount

Recommender REST API with Spring Boot

Recommender REST API is written with Java's Spring Boot framework. REST API is responsible of querying database filled by Spark ETL process or Kafka Consumer app and show results to the user.

Spring Boot project is separated into several modules such as Models, Services, ApiController and Repository to increase the testability and reusability.

Model module has entity classes that represent tables in PostgreSQL via JPA dependency and response classes that are used to send HTTP responses in proper format.

ApiController module is responsible of listening and responding to HTTP requests.

Service module is responsible of being intermediary of ApiController and Repository to execute application level operations.

Repository module is responsible of defining database operations on entities.

REST API endpoints,

***GET /api/products-viewed?user-id=x**

This endpoint receives a user-id and queries PostgreSQL to find last ten products viewed by user. For finding the history of product views, a custom query has been implemented which selects ten products ordered by click_timestamp from the product_view_history table filled by Kafka Consumer app and creates a ProductViewResponse model object which contains user-id, products viewed and type in it. Created ProductViewResponse object is returned as response.

Relevant SQL,

```
SELECT product_id
FROM public.product_view_history
WHERE user_id = ?
ORDER BY click_timestamp DESC
LIMIT 10
```

***DELETE /api/delete-product-view-history?user-id=x&product-id=y**

This endpoint receives user-id and product-id then executes delete operation on product_view_history table with given criteria. If the deletion is successful then Boolean true is returned, if deletion fails then Boolean false is returned to the user.

***GET /api/get-product-recommendation?user-id=x**

This endpoint receives a user-id and queries PostgreSQL to find bestseller products for the products user has viewed. Firstly, a query that returns bestseller products in viewed products' categories by user is executed, this query simply selects products from bestseller_product table with condition of product to have category in a subquery, which is join of product_view_history and products table filtered by user-id and limited by three. If the number of returned products are less than five, no product recommendation is given to the user.

Relevant SQL query to recommend user products of viewed product categories,

```
SELECT product_id
FROM bestseller_product
WHERE category_id IN (
    SELECT category_id
    FROM product_view_history AS PVH
    INNER JOIN products AS P ON PVH.product_id = P.product_id
    WHERE user_id = ? LIMIT 3
)
```

If this query returns no results at all, then user is recommended with general bestseller products without any category constraint. For recommending user general bestseller products a second query is executed which is simply selecting top ten rows of bestseller_product table ordered by sale_amount descending.

Relevant SQL query to recommend user top most sold ten products,

```
SELECT product_id
FROM bestseller_product
ORDER BY sale_amount DESC
LIMIT 10
```