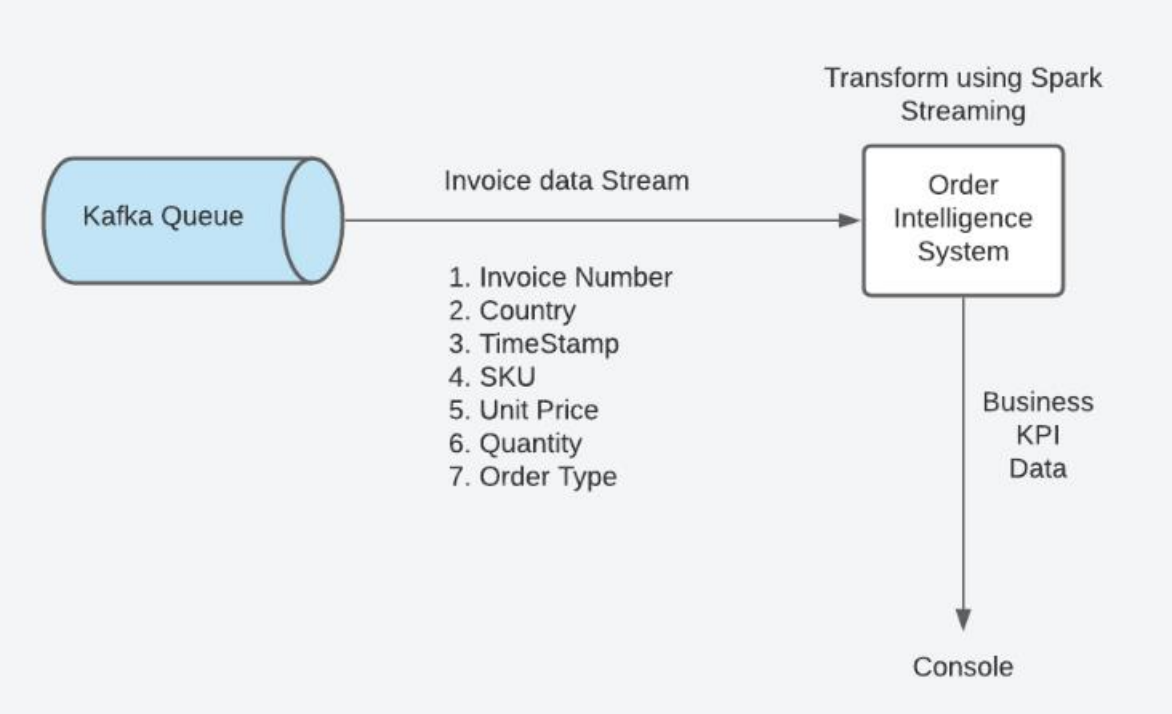
Code Logic - Retail Data Analysis

In this document, the code and the overall steps taken to solve the project is described.

Code Flow

- In the Retail Data Analysis Project, we will be sourcing the published data from the Kafka stream.
- First a Spark session is created and data is read from Kafka by proving the server IP address, port number and topic name.
- Create Schema of the data columns sourced from Kafka. This is input into the Order Intelligent System.
- Perform transformation on the data using UDF in spark streaming application.
- Write the output into the console\file sink (console-output file).
- Calculate KPIs on a tumbling window of one minute on orders across the globe. Watermarking is used to manage late arriving data.
    - The first three KPIs on a per-country basis
    - And the other one is Time based.
- File Sink - Write the KPI output as JSON files to the path provided
- AwaitTermination() – Spark runs continuously until external termination is provided.



CODE AND LOGIC

```
# Import Dependencies
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark.sql.functions import from_json
from pyspark.sql.window import Window

#Create spark Session
spark = SparkSession  \
        .builder  \
        .appName("StructuredSocketRead")  \
        .getOrCreate()
spark.sparkContext.setLogLevel('ERROR')
```

Here Spark session is created using the builder function

```
# Read Input
raw_order = spark  \
        .readStream  \
        .format("kafka")  \
        .option("kafka.bootstrap.servers","18.211.252.152:9092")  \
        .option("subscribe","real-time-project")  \
        .option("startingOffsets", "latest")  \
        .load()
```

After the session is created, this code is used to connect to the Kafka bootstrap server : 18.211.252.152 and port : 9092. The data is published in the topic real-time-project" and the data is loaded. The starting point of the query is considered as 'Latest' of each Topic.

```
# Define Schema
JSON_Schema = StructType() \
        .add("invoice_no", LongType()) \
                .add("country",StringType()) \
        .add("timestamp", TimestampType()) \
        .add("type", StringType()) \
        .add("total_items",IntegerType())\
        .add("is_order",IntegerType()) \
        .add("is_return",IntegerType()) \
```

```
.add("items", ArrayType(StructType([
StructField("SKU", StringType()),
StructField("title", StringType()),
StructField("unit_price", FloatType()),
StructField("quantity", IntegerType())
])))
```

The schema of the data sourced from Kafka is described .

```
order_stream = raw_order.select(from_json(col("value").cast("string"),
JSON_Schema).alias("data")).select("data.*")
```

From the input JSON file received from Kafka the data is extracted.

Data Transformation

The data extracted from the source is then transformed by calling User Defined Function

#is_a_order function

helps identify the type of order, if type=”Order’ return 1 else 0.

#is_a_return

Helps identify if type of order is return , if type=’Return’ print 1 else 0

#total_items_count

Is calculated by repetitive summing of total count with the quantity of items

#total_cost

Total price is obtained by adding the price of products (unit_price * quantity).

After the UDF is run using the utility function.The columns transformed or generated as per the requirement is added to create the new/extended stream.

#adding the new columns created to the stream

```
order_extended_stream = order_stream \
    .withColumn("total_items", add_total_item_count(order_stream.items)) \
    .withColumn("total_cost", add_total_cost(order_stream.items,order_stream.type)) \
    .withColumn("is_order", is_order(order_stream.type)) \
    .withColumn("is_return", is_return(order_stream.type))
```

Console output is Generated.

#Selecting the columns displayed in the console and appending the data to the batch

```
order_query_console = order_extended_stream \
      .select("invoice_no", "country",
"timestamp","type","total_items","total_cost","is_order","is_return") \
      .writeStream \
      .outputMode("append") \
      .format("console") \
      .option("truncate", "false") \
      .trigger(processingTime="1 minute") \
      .start()
```

KPI Calculation

Here for calculating the KPI's on the real time streaming data, tumbling window of 1 minute (Window function is used). Watermark is also used to manage late arriving data.

1. Time based KPIs - Calculation

Total volume of sale = sum of total_cost
OPM = count of invoice_no
Average transaction size = Average of total_cost
Average Return = Average of is_Return

```
# Time based KPIs - Calculation
agg_time = order_extended_stream \
   .withWatermark("timestamp","1 minutes") \ #allows lateness by 1 minute
   .groupby(window("timestamp","1 minutes", "1 minute")) \
   .agg(sum("total_cost").alias("total_volume_of_sales"),
      count("invoice_no").alias("OPM"),
      avg("total_cost").alias("average_transaction_size"),
      avg("is_Return").alias("rate_of_return")) \

.select("window.start","window.end",”OPM”,"total_volume_of_sales","average_transacti
on_size","rate_of_return")
```

2. Time and country based KPIs - Calculation

Total volume of sale = sum of total_cost
OPM = count of invoice_no
Average Return = Average of is_Return

```
# Time and Country based KPIs - Calculation
agg_time_country = order_extended_stream \
   .withWatermark("timestamp", "1 minutes") \ #allows lateness by 1 minute
   .groupBy(window("timestamp", "1 minutes","1 minutes"), "country") \
   .agg(sum("total_cost").alias("total_volume_of_sales"),
```

```
        count("invoice_no").alias("OPM"),
        avg("is_Return").alias("rate_of_return")) \

.select("window.start","window.end","country","OPM","total_volume_of_sales","rate_of_return")
```

File Sink

1.

```
# Write time based KPI values to the File sink
ByTime = agg_time.writeStream \
    .format("json") \
    .outputMode("append") \
    .option("truncate", "false") \
    .option("path", "timeKPI/") \
    .option("checkpointLocation", "timeKPI/cp/") \
    .trigger(processingTime="1 minutes") \
    .start()
```

2.

```
# Write time and country based KPI values
ByTime_country = agg_time_country.writeStream \
    .format("json") \
    .outputMode("append") \
    .option("truncate", "false") \
    .option("path", "time_countryKPI/") \
    .option("checkpointLocation", "time_countryKPI/cp/") \
    .trigger(processingTime="1 minutes") \
    .start()
```

As specified in the problem statement the output of the KPI calculated is written in to the file sink using the above codes and the path specifies the location where the Files are located.