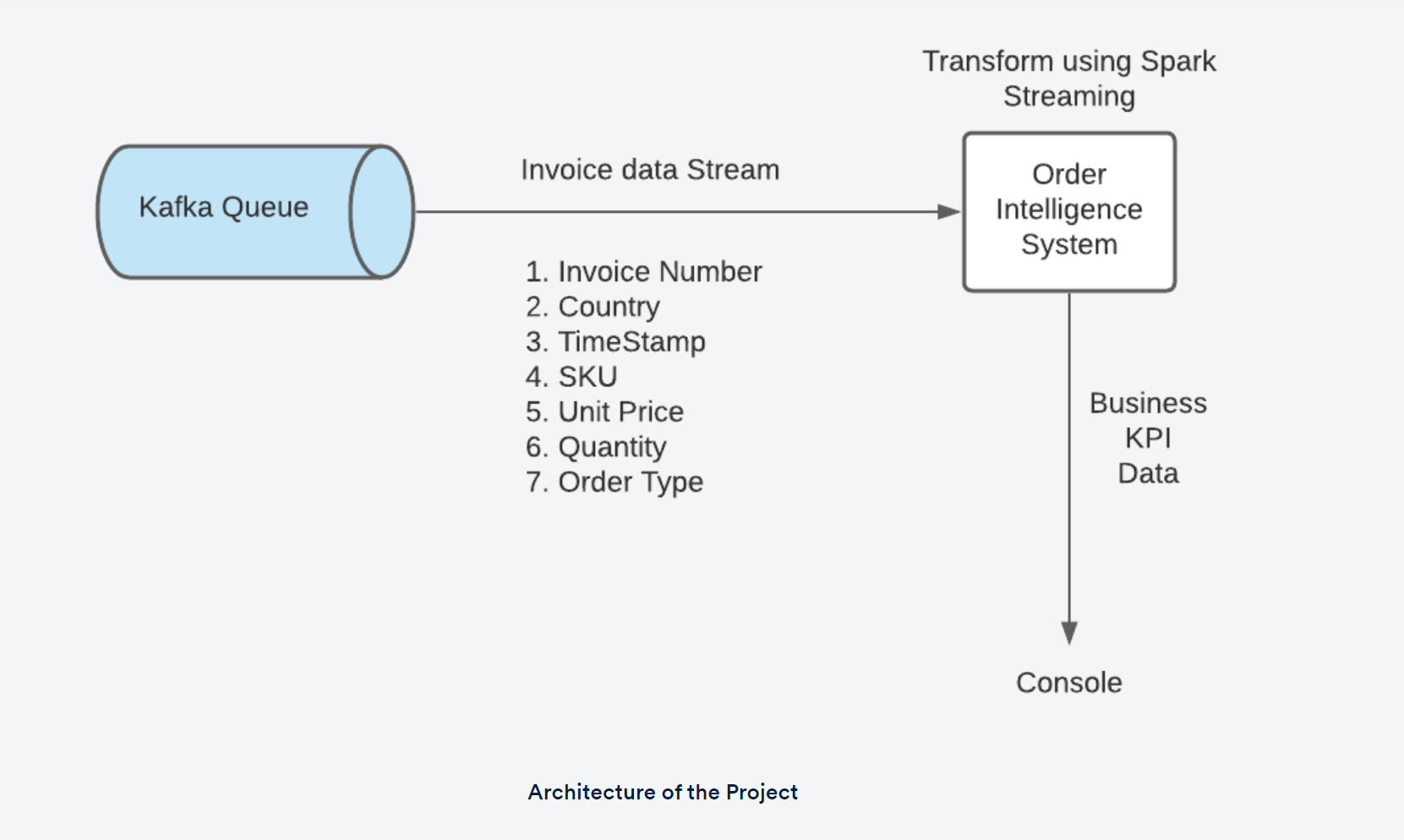
Solution approach



The tasks that you will be performing in this project are as follows -

* **Reading input data from Kafka**
  + Code to take raw Stream data from Kafka server
  + Details of the Kafka broker are as follows:
    - **Bootstrap Server** - 18.211.252.152
    - **Port** - 9092
    - **Topic**- real-time-project
* **Calculating additional columns and writing the summarised input table to the console**
  + The following attributes from the raw Stream data have to be taken into account for the project:
    - **invoice\_no**: Identifier of the invoice
    - **country**: Country where the order is placed
    - **timestamp**: Time at which the order is placed
  + In addition to these attributes, the following **UDFs**have to be calculated and added to the table:
    - **total\_cost**: Total cost of an order arrived at by summing up the cost of all products in that invoice (The return cost is treated as a loss. Thus, for return orders, this value will be negative.)
    - **total\_items**: Total number of items present in an order
    - **is\_order**: This flag denotes whether an order is a new order or not. If this invoice is for a return order, the value should be 0.
    - **is\_return**: This flag denotes whether an order is a return order or not. If this invoice is for a new sales order, the value should be 0.
  + The input table must be generated for each **one-minute window**.
  + Code to define the schema of a single order
  + Code to define the aforementioned UDFs and any utility functions are written to calculate them
  + Code to write the final summarised input values to the console. This summarised input table has to be stored in a Console-output file. This can be done by simply appending ‘**> file-name**’ to the Spark-Submit code as follows:
* spark2-submit\_command > file\_name

An example table written to the console is also needed. It should look like below.

**Note**: The below is just a reference format that can be followed, however, you are free to pick your own approach to solve the problem.

Graphical user interface, table

Description automatically generated

* **Calculating time-based KPIs**:
  + Code to calculate time-based KPIs **tumbling window of one minute** on orders **across the globe**. These KPIs were discussed in the previous segment.
  + KPIs have to be calculated for a 10-minute interval for evaluation; so, ten 1-minute window batches have to be taken.
  + Time-based KPIs can be structured like below. (These tables do not need to be outputted and are just for reference as to how your KPI tables must be structured when all the files are combined.)

**Note**: The below is just a reference format that can be followed, however, you are free to pick your own approach to solve the problem.

Table

Description automatically generated

* **Calculating time- and country-based KPIs**:
  + Code to calculate time- and country-based KPIs **tumbling window of one minute on orders on a per-country basis**. These KPIs were discussed in the previous segment.
  + KPIs have to be calculated for a 10-minute interval for evaluation; so, ten 1-minute window batches have to be taken.
  + Time- and country-based KPIs can be structured like below. (These tables do not need to be outputted and are just for reference as to how your KPI tables must be structured.)

**Note**: The below is just a reference format that can be followed, however, you are free to pick your own approach to solve the problem.

Graphical user interface, text, table

Description automatically generated

* **Writing all the KPIs to JSON files**:
  + Code to write the KPIs calculated above into JSON files for each one-minute window.
  + These have to be written for a 10-minute interval.

All the resultant JSON files have to be downloaded and then archived into separate ZIP files for time-based and time- and country-based KPIs, respectively, along with the Console-output file.