TECHNISCHE UNIVERSITÄT DARMSTADT

Prof. Dr. Max Mühlhäuser

TELEKOOPERATION
Fachbereich Informatik
Hochschulstr. 10
64289 Darmstadt

TK1: Distributed Systems – Programming & Algorithms

3rd Programming Exercise
Submission Date: 13.01.2021

By handing in a solution you confirm that you are the exclusive author(s) of all the materials.

P3. Data Analytics via Apache Spark

You are provided with a dataset by Fraport. The dataset contains data about arriving and departing flights at the Frankfurt Airport from August and September 2018. The flights are stored as lines of JSONs and each file contains only arrival or only departure flights for one day. More information about the JSONs' structure:

https://developer.fraport.de/store/documentation

In this exercise you will use the **Apache Spark** API to extract information from the given dataset according to the listed tasks.

Spark is a powerful analytics engine allowing fast large-scale data processing in a cluster. You have already been introduced into its strengths and its advantages compared to Hadoop. In this exercise you will get to know some of its core functions by applying it on local data, using the given dataset.

Use the template (Gradle project) from the Moodle course for the implementation. **Do not modify the Gradle build file. Do not modify nor delete any method signatures, classes, packages or interfaces since they are required for testing (there will be private tests for the correction of the assignment).** You can add classes and methods in the existing packages and classes.

It is mandatory to use the **Spark** API for the processing of the tasks. It is not sufficient to simply return the resulting values without the usage of Spark.

Tasks

Implement the following tasks in the class **AirportInfoImpl**:

1. **Statistics of the number of departing flights for each destination**
   - ***Dataframe*** *with destination airport and its count in each row*
   - *sorted in descending order*
   - *exclude flights with empty* ***arrivalAirport***
2. **Number of flights to Berlin for each gate**
   - ***Dataframe*** *with gate and its count in each row*

    - *sorted in descending order*
    - *exclude flights with empty **gate***
    - *exclude gates with 0 flights to Berlin*
3. **Number of flights for each aircraft on date X**
    - ***JavaPairRDD** with key **modelName** and its count as its value*
    - *exclude aircrafts with 0 flights on the given date*
    - *use **originDate** (format: **"YYYY-MM-DD")** to filter by date*
4. **On which day did Ryanair have a strike (contained in the dataset)?**
    - ***String** of the date (format: **"YYYY-MM-DD"**)*
    - *two strikes took place in the period, returning one of them is sufficient*
5. **Flights of airline X with the flightstatus Y:**
    - ***Dataset** of **Flight** objects*
    - *only contain flights of the given airline*
    - *only contain flights with one of the given **flightStatus***
    - *include all flights in the given **Dataset** matching the airline and one of the status*
6. **Average number of flights between the timestamps X and Y (both included)**
    - *timestamps format: **"hh:mm:ss"***
    - *only consider **scheduledTime** in the **Flight** objects*
    - *exclude flights with empty **scheduledTime***

Additionally you need to implement a small parser with **GSON** in the class **FlightParserImpl**, allowing deserialization from JSON objects to **Flight** objects, which you will need to test your solution of tasks 5 and 6.

You can find detailed information about the tasks and the required results in the documentation in the template.

Hints

- Use the public tests to verify the correctness of your parser and the structure of your resulting datasets.
- It is possible to implement tasks 5 and 6 without a working parser. The **FlightParser** and the tasks are rated and tested independently.
- An example implementation of some basic functions of the Spark API is available in the **AirportInfoImpl** class.
- **FlightParser**: Use a map functions (**mapPartitions**) on the dataset, in which you use GSON to deserialize each line into objects of **FlightObj**. **GsonBuilder** to register the type adapter **FlightAdapter** for **FlightObj** and to create a Gson object.

The project already includes all dependencies you need for working on this exercise. You do not need to install Spark or Hadoop. However, installing Spark allows you to use spark-shell (you also need scala).

If you get the Exception “Failed to locate the winutils binary in the hadoop binary path; java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.”:

- If you are working on a Windows machine, you might need to download winutils.exe (Moodle course) and create a new directory "winutils", containing a "bin" directory, e.g. "C:\winutils\bin".
- After that, add a system environment variable "HADOOP_HOME" with value "C:\winutils" (not to the bin).
- (Alternatively, add 'System.setProperty("hadoop.home.dir", "path_to_winutils_directory");' as the first line of your code and change the value to the path of your winutils directory.)

If you want to install Spark (optional, not required for this exercise):

- https://www.knowledgehut.com/blog/big-data/how-to-install-apache-spark-on-windows (Windows)
- https://www.tutorialspoint.com/apache_spark/apache_spark_installation.htm (Linux)

Some helpful sources to get started:

- https://spark.apache.org/examples.html
- https://spark.apache.org/docs/latest/sql-getting-started.html
- https://spark.apache.org/docs/latest/sql-data-sources-json.html
- https://spark.apache.org/docs/latest/api/java/index.html

Methods you might need:

• ***Dataset***:	*select, where, filter, map, reduce, groupBy, sort*
• ***RelationalGroupedDataset***:	*count*
• ***JavaRDD***:	*map, mapToPair, filter, reduce, sort*
• ***JavaPairRDD***:	*map, reduceByKey, filter, reduce, sort, aggregate*
• ***functions***:	*explode, col*
• ***Encoders***:	*javaSerialization, STRING, LONG*