AutoScout24 GmbH,
Bothestraße 11-15, 81675,
München. Germany

Coding Challenge Listing Report

Junior Fullstack Software Engineer (m/w/d)

Applicant: Subash Ale Magar (subashale@gmail.com)

Start: 19th January 2021
End: 26th January 2021

OVERVIEW

This task was prepared to give you a "taste" of how our domain looks like and how you develop a solution for the proposed case. The code will be used as a base for discussion on future interview steps. Two CSV files will be provided as a data sources for this exercise. You will be asked to provide four different reports based on this data. Under "Requirements", "Objectives" and later under "Acceptance Criteria" you will find further information about what to do. We kindly ask you to invest a maximum of four to five hours into this task. If you have any questions, please contact us.

Tasks: Data analysis, application development

1. Average Listing Selling Price per Seller Type
2. Percentile distribution of available cars by Make
3. The average price of the 30% most contacted listings
4. The Top 5 most contacted listings per Month
5. Definition of the CSV files

Platform: Web application

The application is based on Restful web services, assembled with angular to consume data and render interface. The web services are provided with a Spring boot framework where Apache spark is used to analyze the result of each task. Application is made of Angular framework.

Some Approach, Implementation, Requirement, Dev tools:

Architecture

1. MVC architecture (Client, Server) , RestFul web services and Client App

Server Side technologies

1. Java 1.8+
2. Maven 4.0.0
3. Spring boot 2.4.2
4. Apache spark core/sql 2.12

Front-end technologies:

1. NodeJs 14.11,
2. NPM 6.14.8
3. Angular 11.0.5
4. Bootstrap 4.5.2

Development tools

1. Windows 10
2. VSCode 1.52.1
3. IntelliJ IDEA 2020.3
4. Postman Canary
5. Tested browser: Chrome, Firefox, Opera

Application state:

1. Development version

URLs:

1. REST APIs (api-domain): http://127.0.0.1:8080/apis
2. Front end application (app-domain): http://127.0.0.1:4200

Github:

- https://github.com/subashale/springboot-spark-ng.git

How to Run

The application is neither hosted nor packaged as an executable program. As mentioned, the application maturity is still in the development phase. Therefore, we must require the mentioned tools and technologies to run the application. Once the repository is cloned from git link and platform is ready, we follow these steps:

1. Open fullstack (link) directory the application in a suitable IDE then using **`IDE inbuilt feature`** or using **`maven spring-boot: run`**. The application will be served in default address api-domain/api.
2. Once the application runs successfully open fullstack-ng directory link then to run use **`ng serve`** command in the terminal/command after that visit app-domain.

Approach of analysis

Since the major tasks are related to analysis of data, in this challenge we have used Apache spark library which supports an analytics engine for large-scale data processing. All the tasks are solved via using Spark SQL engine library which is SQL based wrapper for Spark.

Major focus: Impression

In the coding challenge we focus to show the ability of having working experience with different concepts, technologies, approaches, platforms, quick fixes and to show problem solving skills.

Incomplete task / Future Improvement

Though the problem solving task was sufficiently finished there are few major requirements, practices are yet to cover.

1. Command line interface
2. Packing, Dockerization
3. Test cases for both front and back end
4. Security and processing latency of analyzing and interaction
5. Satisfying design
6. Fully Fledged coding documentation

Demo: screenshot, Logic and latency:

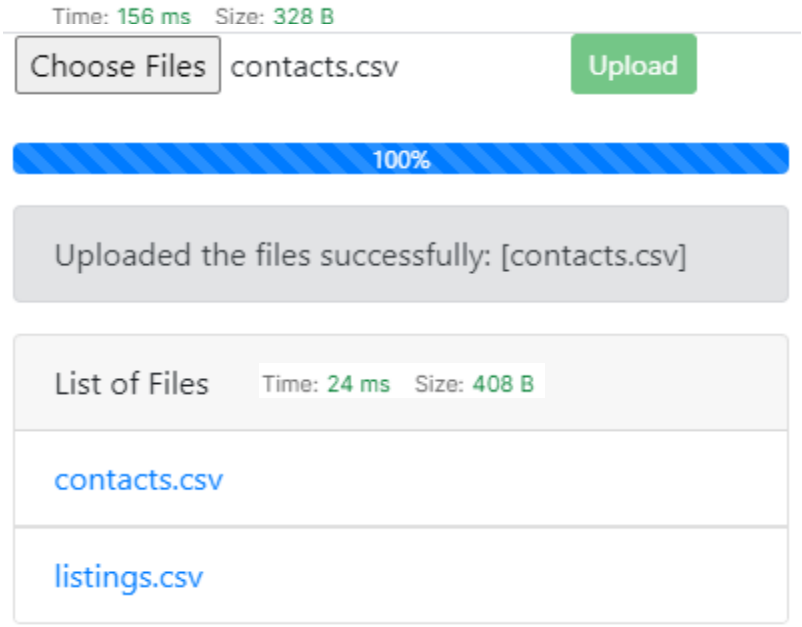
Working screenshots are also attached in the email.

A. File upload interface

This interface uploads the files into the fixed directory. List of the file of that directory's name and the URI is displayed.

File upload API: api-domain/api/upload

File List Files: app-domain/api/files



Task solutions with Spark SQL

After reading data via SparkSession we create tempView with defined data types of each files.

```
SELECT CAST(listing_id as int), CAST(from_unixtime(contact_date/1000) as TIMESTAMP) contact_date FROM contacts
```

```
SELECT CAST(id as int) id, CAST(make as String) make, CAST(price as int) price, CAST(mileage as int) mileage, CAST(seller_type as String) seller_type FROM listings
```

1. Average Listing Selling Price per Seller Type

We solve the problem using AVG and GROUP BY inbuilt methods. Adding type and alias gave the final result.

```
select seller_type,
CAST(ROUND(AVG(price)) as int) as `Average in Euro` from listings group by seller_type
```

Time: 9.67 s Size: 428 B

1. Average Seller Price

Seller type	Average in Euro
other	€ 25.318,-
private	€ 26.080,-
dealer	€ 25.037,-

2. Percentual distribution of available cars by Make

   As above we use GROUP BY and percentile the result using subquery which finds the total price of listings.

```
SELECT make as Make
CAST(ROUND(sum(price)/(select
sum(price) from listings)*100) as
int) as Distribution        from
listings group by make
```

Time: 4.79 s Size: 560 B

2. (%) Distribution by Make

Make	Distribution
BWM	7%
Audi	16%
Mercedes-Benz	15%
Renault	14%
VW	10%
Toyota	17%
Mazda	13%
Fiat	8%

3. Average price of the 30% most contacted listings

   The task was little confusing for me to understand therefore two optional approach are listed. Subquries feature are used to slove the task First we take ofTotal then use that to get only 30% based on contacts data. Finally we formatted the output as stated in the task.

3. Average price of 30% the most contacted listings

Average price
€ 25.381,-

Time: 14.92 s Size: 285 B

```
ofTotal = SELECT DISTINCT CAST(30*(SELECT
count(*) from contacts)/100 as int) as tempTotal FROM contacts

SELECT cast(round(avg(listings.price)) as int) as `Average price`
from listings
INNER JOIN
      (SELECT DISTINCT contacts.listing_id, c from contacts INNER JOIN
      (select listing_id, count(listing_id) as c
      from contacts
      GROUP by listing_id ORDER by c DESC) as tmp
      on contacts.listing_id = tmp.listing_id
      ORDER by tmp.c DESC LIMIT ofTotal) as t
on t.listing_id = listings.id
```

Or we can get 30% result of from listings scenario by append belwo condition

```
LIMIT (SELECT CAST(round(DISTINCT(count(id))*30/100) as int) as
tempLimit from listings)) as top
```

4. The Top 5 most contacted listings per Month

   To make the process easy we again create Temporary View which gives flexibility to get month and year required in the task.

```
SELECT CAST(listing_id as int) id CAST(date_format(CAST(contact_date as
date), 'LL.y') as String) as date FROM contacts
```

   Then we store distinct dates which then loops throught the join query to extract the result. Finally we format Selling Price and Mileage using helper functions.

```
tempDate = SELECT DISTINCT date FROM contactss ORDER BY date

SELECT tmp.Ranking, listings.id as `Listing Id`, make as Make, price as
`Selling Price`, mileage `Mileage`, tmp.cnt `Total Amount of contacts`

FROM listings

INNER JOIN(

        SELECT id, count(id) as cnt,

        row_number() over (order by count(id) DESC) as Ranking

        from contactss

        WHERE date = tempDate

        GROUP by contactss.id

        ORDER by cnt DESC

        limit 5) as tmp

ON listings.id = tmp.id

ORDER BY tmp.Ranking
```

Time: 53.04 s Size: 4.23 KB

4. The Top 5 most contacted listings per Month

01.2020

Ranking	Listing Id	Make	Selling Price	Mileage	Total Amount of contacts
1	1061	Renault	€ 5.641,-	7.000 KM	21
2	1132	Mercedes-Benz	€ 34.490,-	7.000 KM	18
3	1077	Mercedes-Benz	€ 8.007,-	4.000 KM	17
4	1122	Audi	€ 40.481,-	2.000 KM	17
5	1250	Renault	€ 8.446,-	5.000 KM	17

02.2020

Ranking	Listing Id	Make	Selling Price	Mileage	Total Amount of contacts
1	1271	Mercedes-Benz	€ 47.165,-	6.500 KM	37
2	1138	Toyota	€ 13.986,-	8.000 KM	33
3	1235	Mercedes-Benz	€ 5.847,-	5.500 KM	32
4	1006	Renault	€ 47.446,-	7.500 KM	32
5	1250	Renault	€ 8.446,-	5.000 KM	31

03.2020

Ranking	Listing Id	Make	Selling Price	Mileage	Total Amount of contacts
1	1061	Renault	€ 5.641,-	7.000 KM	31
2	1181	Renault	€ 8.933,-	3.500 KM	30
3	1235	Mercedes-Benz	€ 5.847,-	5.500 KM	29
4	1271	Mercedes-Benz	€ 47.165,-	6.500 KM	29
5	1258	Mazda	€ 44.776,-	1.000 KM	29

04.2020

Ranking	Listing Id	Make	Selling Price	Mileage	Total Amount of contacts
1	1181	Renault	€ 8.933,-	3.500 KM	37
2	1118	Audi	€ 38.382,-	2.000 KM	33
3	1006	Renault	€ 47.446,-	7.500 KM	29
4	1123	VW	€ 39.077,-	7.000 KM	28
5	1262	Renault	€ 43.778,-	8.000 KM	28

05.2020

Ranking	Listing Id	Make	Selling Price	Mileage	Total Amount of contacts
1	1204	Toyota	€ 36.895,-	3.500 KM	35
2	1098	Toyota	€ 11.345,-	3.500 KM	32
3	1298	Mazda	€ 15.989,-	6.500 KM	30
4	1018	Renault	€ 33.165,-	3.000 KM	29
5	1275	Mazda	€ 15.705,-	7.000 KM	27

06.2020

Ranking	Listing Id	Make	Selling Price	Mileage	Total Amount of contacts
1	1258	Mazda	€ 44.776,-	1.000 KM	18
2	1006	Renault	€ 47.446,-	7.500 KM	15
3	1271	Mercedes-Benz	€ 47.165,-	6.500 KM	14
4	1037	Fiat	€ 14.940,-	7.000 KM	14
5	1012	Audi	€ 10.286,-	3.000 KM	14

5. Definition of the CSV files

   Defined type are used here to extract the result. We used Dataset.schema() method to extract casted types of each file. Afterwards purified the types based on requirements using custom methods (helper class) for Field, Type and Required topics.

   Time: 279 ms Size: 688 B

5. Defination of CSV files

Contacts.csv

field	type	required
listing_id	alphanumeric	yes
contact_date	alphanumeric	yes

listings.csv

field	type	required
id	numeric	yes
make	alphanumeric	yes
price	numeric	yes
mileage	numeric	yes
seller_type	alphanumeric	yes