Data Pipeline challenge

Context

You have the ***customer_courier_chat_messages*** event with data about individual messages exchanged between customers and couriers via the in-app chat. An example of the event data is below:

```
[
	{
		"senderAppType": "Customer iOS",
		"customerId": 17071099,
		"fromId": 17071099,
		"toId": 16293039,
		"chatStartedByMessage": false,
		"orderId": 59528555,
		"orderStage": "PICKING_UP",
		"courierId": 16293039,
		"messageSentTime": "2019-08-19T08:01:47Z"
	},
	{
		...
	}
]
```

You also have access to the ***orders*** event where you have an orderId and cityCode field.

```
[
	{
		"orderId": 59528555,
		"cityCode": "BCN"
	},
	{
		...
	}
]
```

Task

Your task is to build a data pipeline on top of a Data Lake folder structure (for the sake of simplicity, on your local file system) to aggregate individual messages into conversations. Take into consideration that a conversation is unique per order. The required fields for the output Data Lake table ***customer_courier_conversations*** are the following:

- **order_id**
- **city_code**
- **first_courier_message**: Timestamp of the first courier message
- **first_customer_message**: Timestamp of the first customer message
- **num_messages_courier**: Number of messages sent by courier
- **num_messages_customer**: Number of messages sent by customer
- **first_message_by**: The first message sender (courier or customer)
- **conversation_started_at**: Timestamp of the first message in the conversation
- **first_responsetime_delay_seconds**: Time (in secs) elapsed until the first message was responded
- **last_message_time**: Timestamp of the last message sent
- **last_message_order_stage**: The stage of the order when the last message was sent

Requirements

PASS CRITERIA (ALL POINTS ARE MANDATORY)	☐ The application is written in **Python**, **Java** or **Scala** ☐ The **software project** includes the **build file (Poetry, Gradle, Maven, or sbt)** ☐ The project includes the **source code** ☐ The project includes **unit tests** ☐ The project includes **e2e tests** ☐ The project includes **input and output data** located in the corresponding Data Lake folders ☐ The project includes a **README.md** file with step-by-step instructions to **build**, **run** and **test** the application, with all the exact **commands** that we need to execute and any **explanatory notes** ☐ There is a **Data Lake catalog folder structure** in place to move the data across the pipeline. Note that we are referring only to

	local directories, we are NOT asking to create a Hive metastore or to mock a Cloud data catalog. ☐ The **README.md** file includes an explanation on what Data Quality checks can be implemented and how ☐ The **README.md** file includes an explanation on how to **orchestrate the pipeline** with any orchestration tool, specifying the **tasks**, **dependencies**, **scheduling**, **automations (e.g. in case of failures)**, and **way of execution (e.g. how to run it, and what would be the expected behavior if we re-run it)** ☐ The project is compressed in a ZIP file and submitted through the Greenhouse link within **7-10 calendar days**. If due to particular circumstances you need an extension of time, please align with your recruiter.
BONUS POINTS (OPTIONAL)	☐ The **README.md** file includes an explanation on how to handle **late arriving events** ☐ There is a task with the implementation of the proposed **Data Quality checks** ☐ There is an **Airflow DAG** in place with the **orchestration** of the different tasks that need to be run ☐ The project includes **Docker** files to containerize the application with all the necessary software and dependencies. If you have problems due to your OS while installing Docker, then you can add the Docker files even if you are not able to test them (please let us know including the corresponding heads up in the README).
HINTS	• The column names should be in **Snake Case** format • The final dataset should have **order_id granularity** • Make sure to use proper **partitioning** and proper **file formats** and **schemas** • Your solution must be **ready for production and scalable**, i.e. include all the optimizations that you consider necessary to **process a large amount of events**. Keep in mind that we want to see **how you work in a quasi-real production scenario**.

	• You are free to choose any **frameworks** that you consider to be the best to solve this type of problem. We recommend that you **justify your decision**. Also keep in mind that the reviewer of your assignment may or may not be familiar with the tooling of your choice, so please add all the **explanatory notes** that you consider necessary to understand the solution without previous context and guide the reviewer through it. • Follow **Software Engineering best practices** (comments, clean code) and **style guides** (naming conventions, ...) • Make sure that you include in the **README** instructions on how to setup the necessary tools, e.g. for a Scala project it could be: install **sbt** and **AdoptOpenJDK 11**. • In general, keep in mind that we will not be able to grade your assignment if we cannot build, run and test the application by executing exactly the commands that you provide in your step-by-step instructions in the **README**. • We recommend that you document in the **README** what are your **assumptions on the problem** and what are the **areas of improvement on the solution** that you are delivering