Assignment 1

Nicola Rinaldi – Francesco Torgano

Assumptions

We assume that the architecture will need to take care of 2 types of data: registry data and log data.
We assume that the source and destination of the data might be multiple to handle both having multiple sources and multiple destinations (e.g.: for having data backups in different databases).
We assume that the extract class will be kept always alive (so that there’s no need to handle reading/storing the timestamp outside of the class)

Definitions

Source: The source of the data, in the example given during the lecture was OneStream.
Location: place where the data is/will be stored inside of the source/destination.

Solution Description

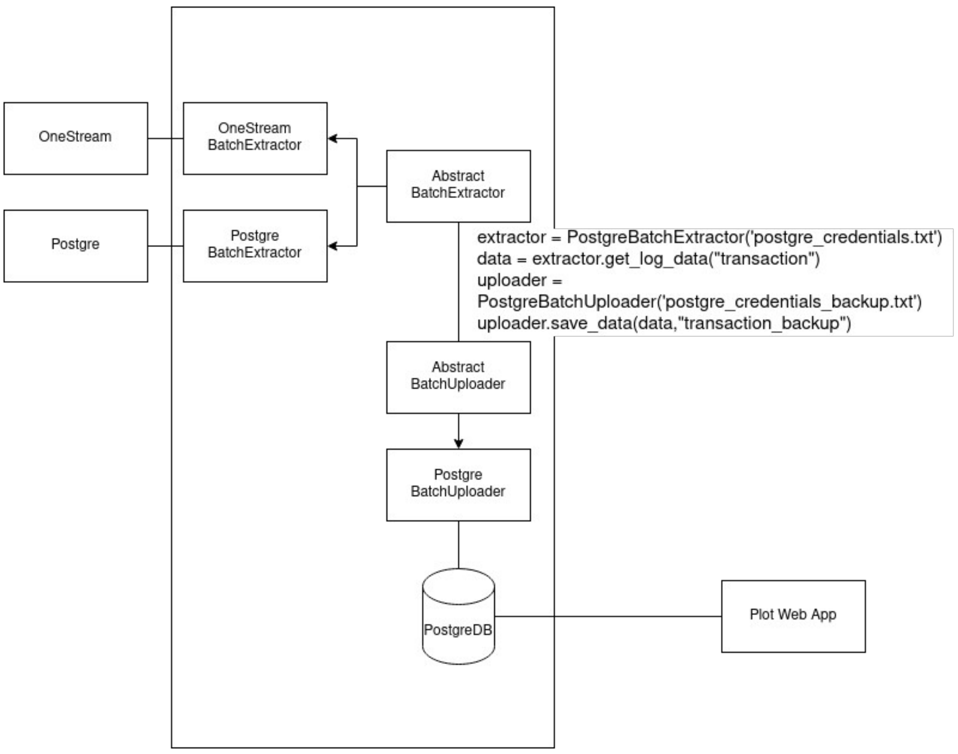
We developed an architecture with 2 abstract classes: BatchExtractor and BatchUploader.
This structure applies the ETL (Extract, Transform and Load) idea to the problem at hand, explicitly handling the Extract and Load part with BatchExtractor and BatchUploader respectively.

BatchExtractor

BatchExtractor has an abstract method called get_log_data that takes the location of the data (that will vary depending on the place where it’s stored) and that will return the fresh data (data newer than the last retrieval). BatchExtractor has another abstract method called get_registry_data that does the same but for registry data.
We implemented a concrete version of this class: PostgreBatchExtractor that is initialized with a file taking the database credentials to initialize the source and retrieves data from database tables.

BatchUploader

BatchUploader has an abstract method called save_data that takes as input the data we want to save and the location we want to save it to; this method will take care of saving the input data on the proper location.
We implemented a concrete version of this class: PostgreBatchUploader that is initialized the same way as PostgreBatchExtractor but will be used to store data in a different database.



```
extractor = PostgreBatchExtractor('postgre_credentials.txt')
data = extractor.get_log_data("transaction")
uploader =
PostgreBatchUploader('postgre_credentials_backup.txt')
uploader.save_data(data,"transaction_backup")
```

Software Architecture Pillars connections

1. The requirement was having a place to retrieve data from to develop a web application to show statistics and plots from the data itself.
We implement a system of 2 classes that allows to retrieve data from a source and store it to another location that we have control over, so that we are decoupled from the source which we might not have direct control over.
2. The 2 classes are abstract to guide the developers in their work, abstract classes are the right choice because we might need to change the source/destination of the data.
4-5. The concrete classes implemented following the 2 abstract classes can be used independently from each other and mixed together to allow different combinations of source/destination so that no code is “wasted”.
The concrete classes might take a file as input, containing the information to interact with the data source/destination so that the same concrete class can work with different data sources/destination of the same type too.

Possible improvements

Since we abstracted to the destination of the data too, the interface to interact with it might change depending on which we use. This could add a direct dependency to the eventual web app that should be developed.
To solve this problem we could wrap the destination of data with an adapter pattern to expose a simpler and more stable interface to interact with and, at the same time, allowing us to limit the allowed operations.