Software Design with Python

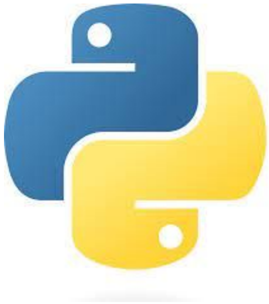
Assignment 2

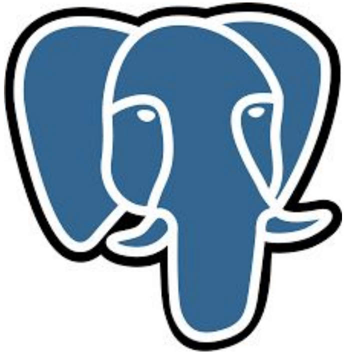
Ozioma Okonicha
Abdulrahman Takriti
Abdulmueez Emiola
Tishkin Pavel

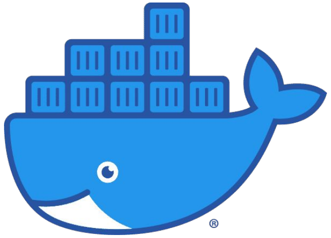
Motivation

Software design patterns come in handy when dealing with object oriented programing.They allow managing the relationship between the object. Furthermore, they help in limiting the access from a client-end in a way, and simplifying the interfacing between multi class in an another way.

Technology stack







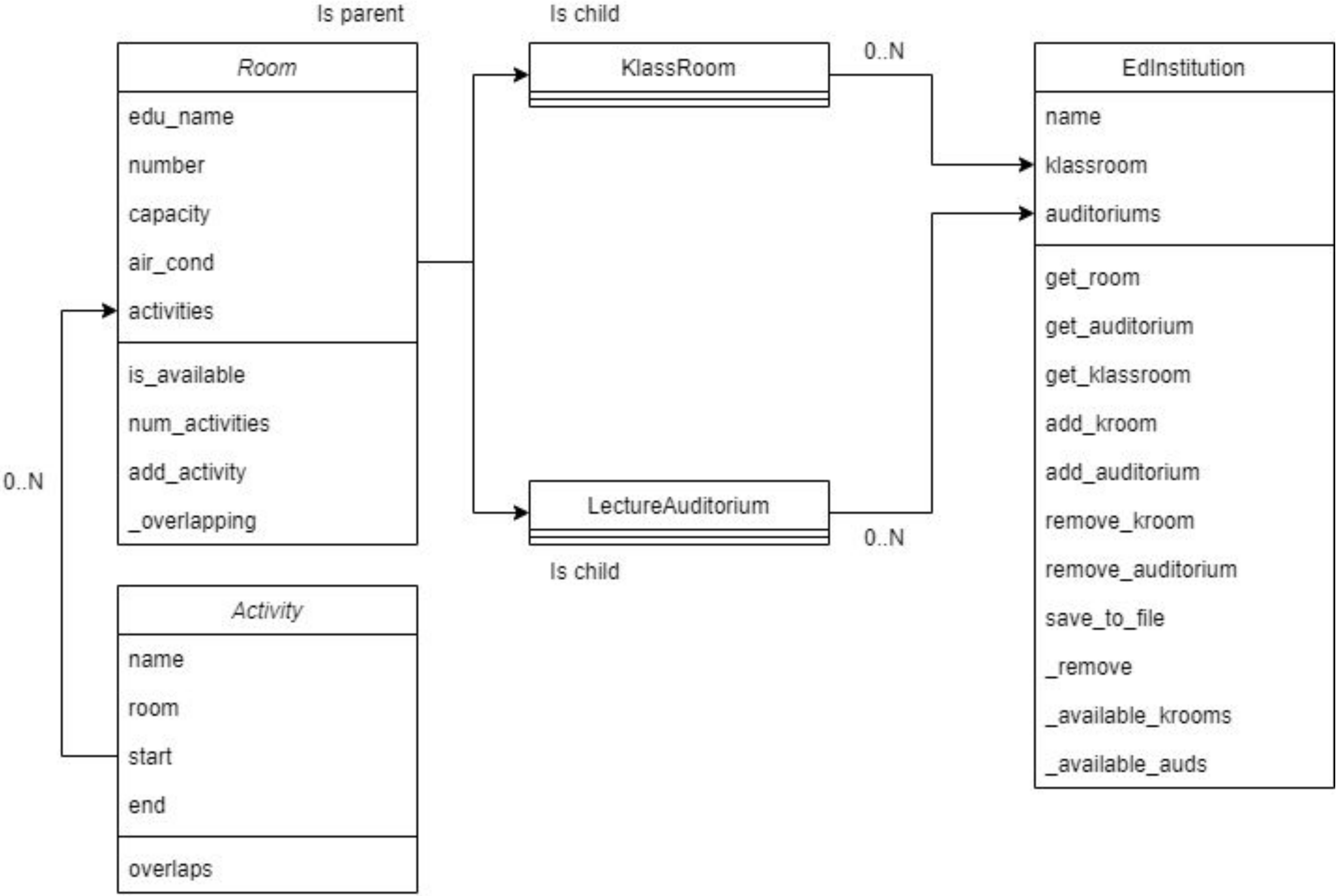
Task 1

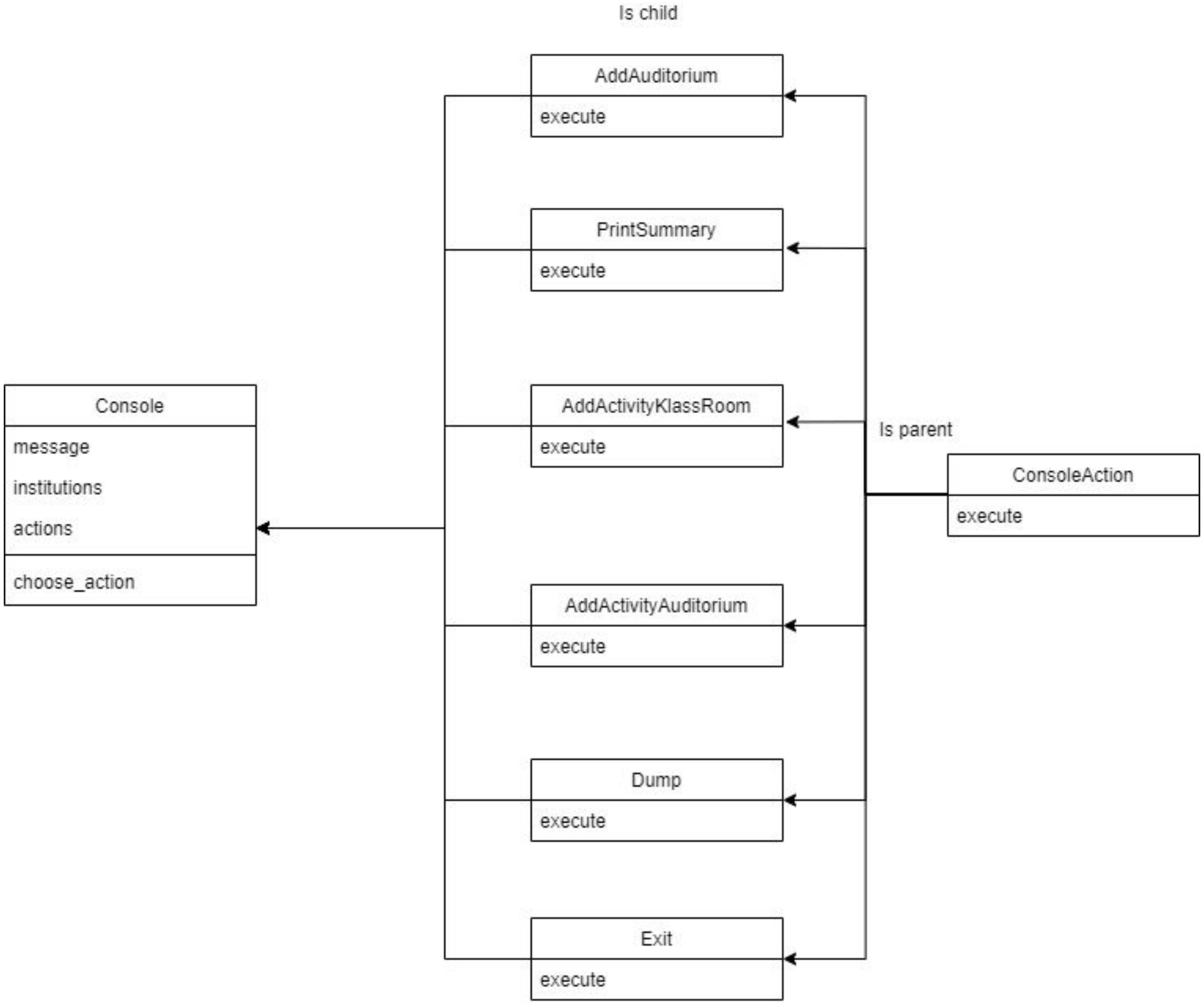
CLI University system

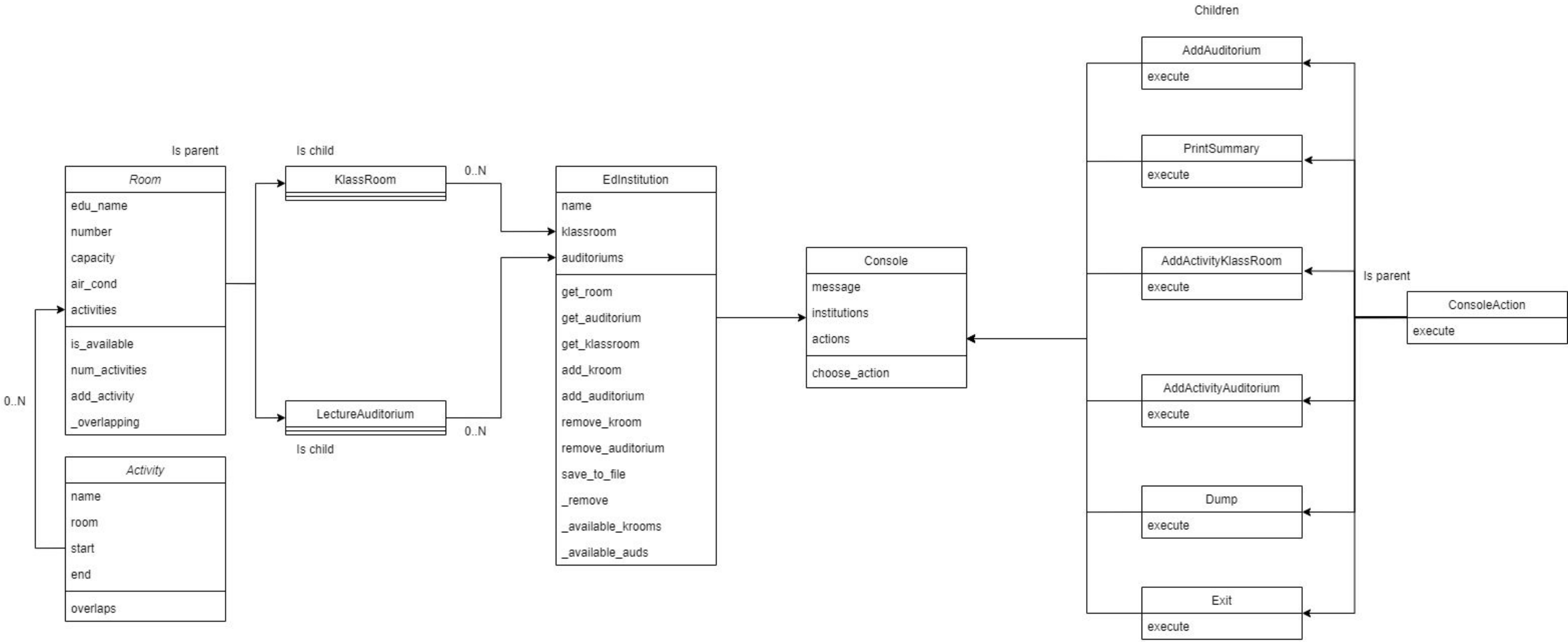
Design Patterns

We were able to incorporate the following design patterns in the implementation of the first task:

1. Command
2. Ducktyping


Is parent
Is child
Room
edu_name
number
capacity
air_cond
activities
is_available
num_activities
add_activity
_overlapping
KlassRoom
0..N
LectureAuditorium
0..N
Is child
EdInstitution
name
klassroom
auditoriums
get_room
get_auditorium
get_klassroom
add_kroom
add_auditorium
remove_kroom
remove_auditorium
save_to_file
_remove
_available_krooms
_available_auds
0..N
Activity
name
room
start
end
overlaps


Is child
AddAuditorium
execute
PrintSummary
execute
AddActivityKlassRoom
execute
Is parent
ConsoleAction
execute
Console
message
institutions
actions
choose_action
AddActivityAuditorium
execute
Dump
execute
Exit
execute


Is parent
Is child
Room
edu_name
number
capacity
air_cond
activities
is_available
num_activities
add_activity
_overlapping
0..N
Activity
name
room
start
end
overlaps
KlassRoom
0..N
LectureAuditorium
0..N
Is child
EdInstitution
name
klassroom
auditoriums
get_room
get_auditorium
get_klassroom
add_kroom
add_auditorium
remove_kroom
remove_auditorium
save_to_file
_remove
_available_krooms
_available_auds
Console
message
institutions
actions
choose_action
Children
AddAuditorium
execute
PrintSummary
execute
AddActivityKlassRoom
execute
Is parent
ConsoleAction
execute
AddActivityAuditorium
execute
Dump
execute
Exit
execute

Task 2

Gaming Users System

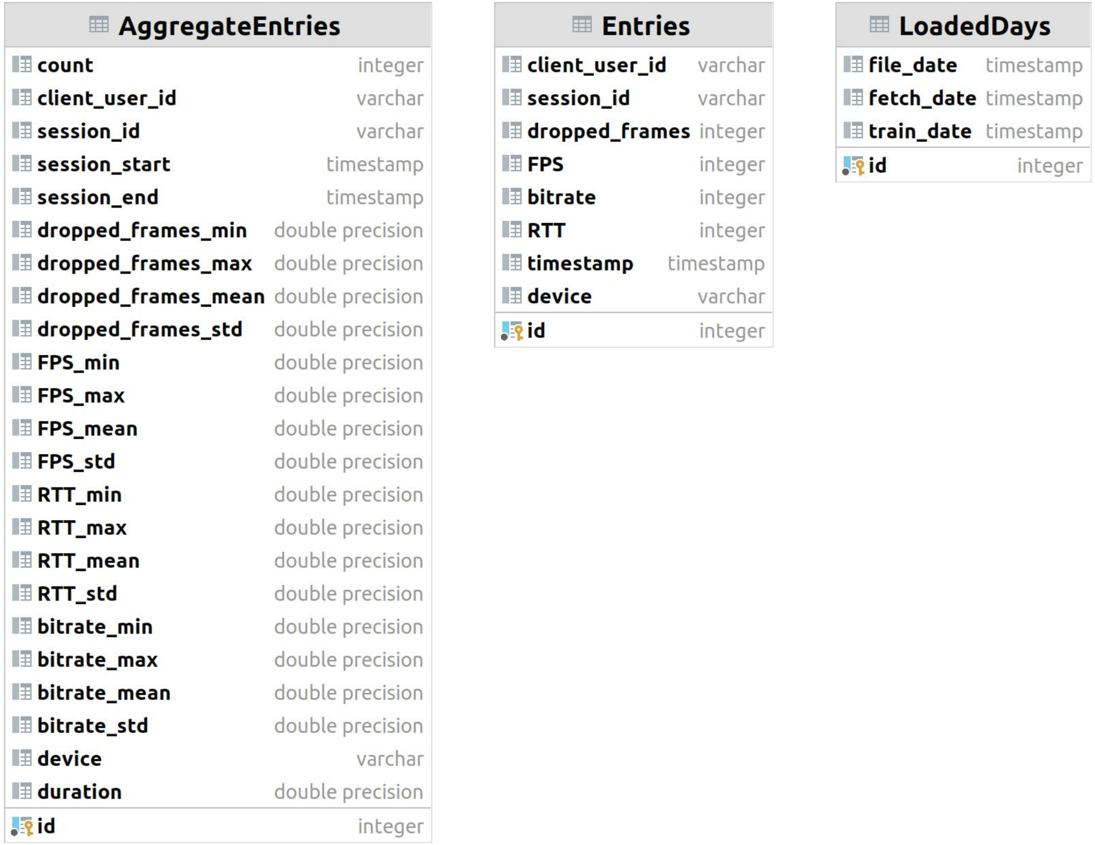
Design Patterns

We were able to incorporate the following design patterns in the implementation of the second task:

1. Singleton
2. Iterator

Database

We made use of postgreSQL database to store the data gotten from the .csv files in the Google drive. With the help of **SQLAlchemy** and **pandas**, we could manipulate the information as we want.

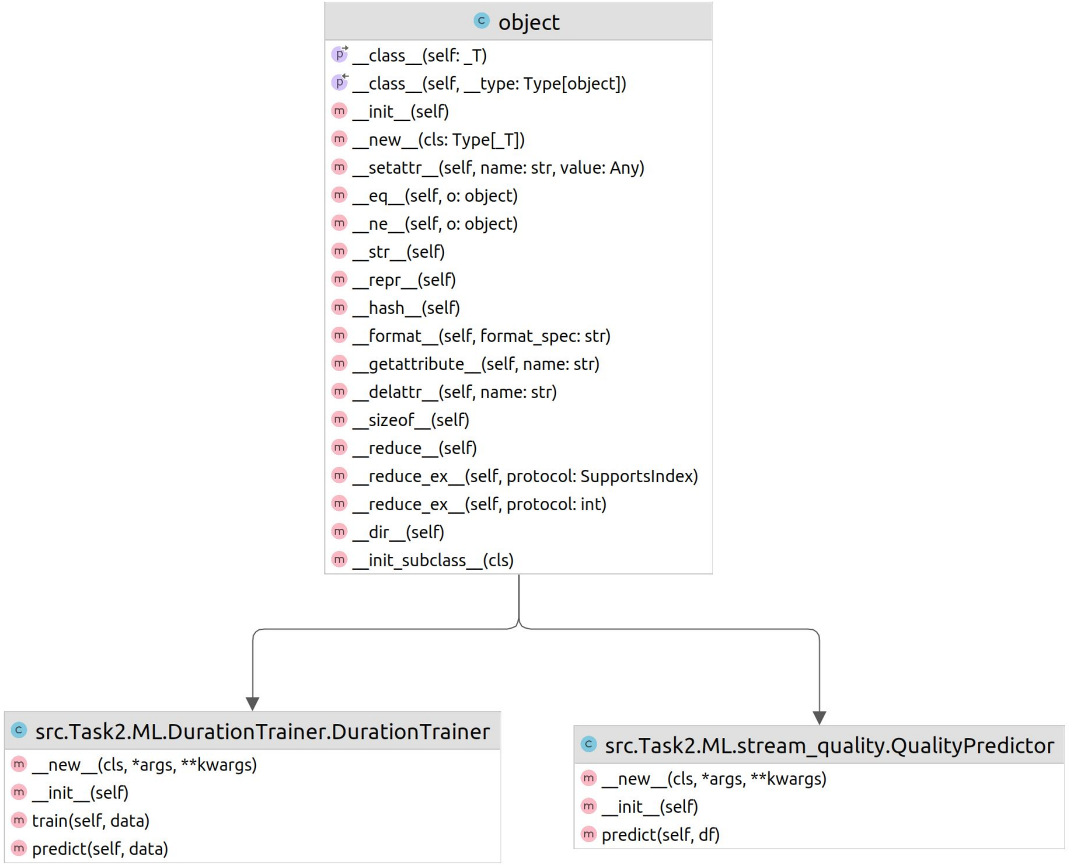


AggregateEntries	
count	integer
client_user_id	varchar
session_id	varchar
session_start	timestamp
session_end	timestamp
dropped_frames_min	double precision
dropped_frames_max	double precision
dropped_frames_mean	double precision
dropped_frames_std	double precision
FPS_min	double precision
FPS_max	double precision
FPS_mean	double precision
FPS_std	double precision
RTT_min	double precision
RTT_max	double precision
RTT_mean	double precision
RTT_std	double precision
bitrate_min	double precision
bitrate_max	double precision
bitrate_mean	double precision
bitrate_std	double precision
device	varchar
duration	double precision
id	integer

Entries	
client_user_id	varchar
session_id	varchar
dropped_frames	integer
FPS	integer
bitrate	integer
RTT	integer
timestamp	timestamp
device	varchar
id	integer

LoadedDays	
file_date	timestamp
fetch_date	timestamp
train_date	timestamp
id	integer

ML models



object

- __class__(self: _T)
- __class__(self, __type: Type[object])
- __init__(self)
- __new__(cls: Type[_T])
- __setattr__(self, name: str, value: Any)
- __eq__(self, o: object)
- __ne__(self, o: object)
- __str__(self)
- __repr__(self)
- __hash__(self)
- __format__(self, format_spec: str)
- __getattribute__(self, name: str)
- __delattr__(self, name: str)
- __sizeof__(self)
- __reduce__(self)
- __reduce_ex__(self, protocol: SupportsIndex)
- __reduce_ex__(self, protocol: int)
- __dir__(self)
- __init_subclass__(cls)

src.Task2.ML.DurationTrainer.DurationTrainer

- __new__(cls, *args, **kwargs)
- __init__(self)
- train(self, data)
- predict(self, data)

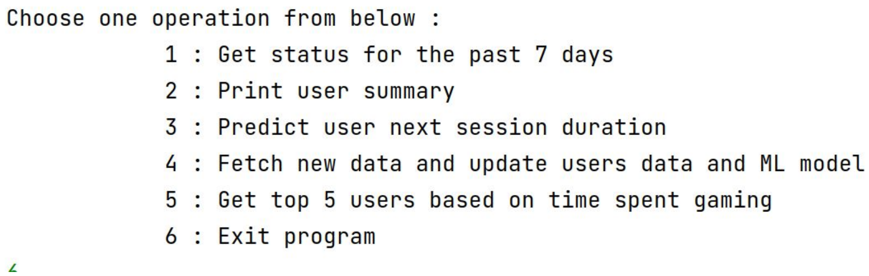
src.Task2.ML.stream_quality.QualityPredictor

- __new__(cls, *args, **kwargs)
- __init__(self)
- predict(self, df)

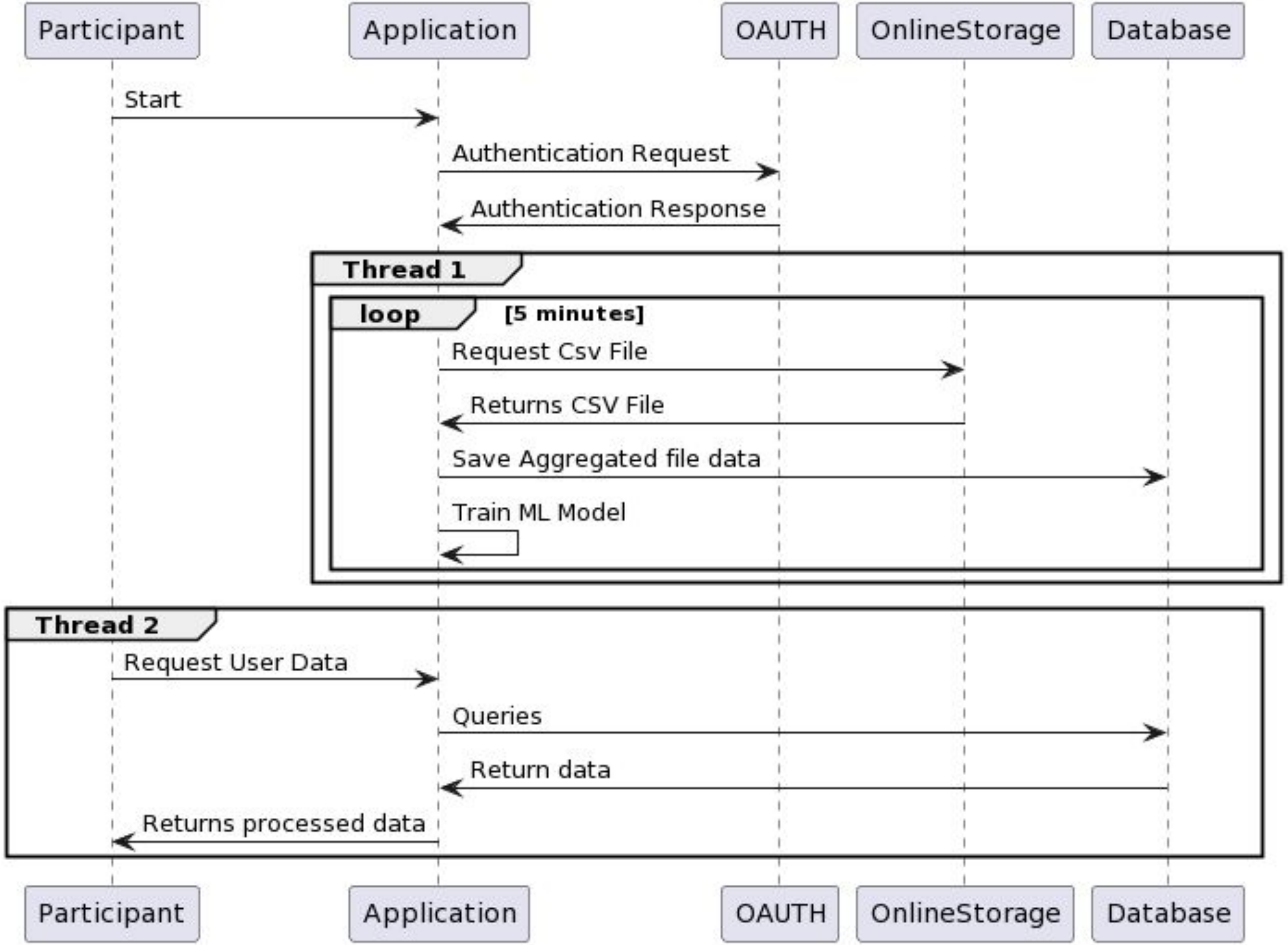
- For the prediction of duration, we make use of the sklearn implementation of the **SGDregressor** algorithm. This allows us to continuously train our model on new data.
- To predict steam quality, we make use of our already pretrained **logistic regression** algorithm from our ML Assignment.

Interactive terminal

The `ui.py` holds everything the user needs to retrieve the statistics.



```
Choose one operation from below :
            1 : Get status for the past 7 days
            2 : Print user summary
            3 : Predict user next session duration
            4 : Fetch new data and update users data and ML model
            5 : Get top 5 users based on time spent gaming
            6 : Exit program
```


Participant
Application
OAUTH
OnlineStorage
Database
Start
Authentication Request
Authentication Response
Thread 1
loop
[5 minutes]
Request Csv File
Returns CSV File
Save Aggregated file data
Train ML Model
Thread 2
Request User Data
Queries
Return data
Returns processed data
Participant
Application
OAUTH
OnlineStorage
Database

Bonus

Docker

Motivation

In order to make it easier for anyone to run the code, we set up our project on docker

Problems faced

1. Both tasks did not capture the user input if we ran them as is. Therefore, we had to use specific commands
2. We made a rookie mistake in task 2. The main algorithm fails if it can not find connection to the database. Initially, we forgot that Postgres must be started before the main application (using depend_on in compose file)

Resources

- Github repo
- Assignment description

THE END

Thank you for your attention