# First Sprint

Streaming event data compliance checking in Python

| | |
|---|---|
| Jingjing Huo | 376323 |
| Sabya Shaikh | 384606 |
| Zheqi Lyu | 378653 |

1 SetUp

2 Source Code description

2.1 Client

2.1.1 Menu for User

2.1.2 Logging for Client

2.1.3 Client Exception

2.2 Server

2.2.1 Build automata

2.2.1.1 Logic for Building Automata

2.2.1.2 Server-side Thread handling

2.2.2 Database Functions

3 Testing Project

4 Phase Review

# 1 SetUp

## 1.1 Server

To run server application, we need to run the server.py file.
Locate the project directory - StreamingEventCompliance
Run the command: *python server.py*

## 1.2 Client

To run client application , we need to run the client.py file with either one argument or two arguments
Enter the project directory - StreamingEventCompliance
Run the command:
*python client/client.py <username>*

or

*python client/client.py <username> <absolute_filepath>*

- Argument 1: **username -** This is an arbitrary username that user can provide to the client. The user must use the same name for any other runs in future to display the deviations pdf corresponding to the file the user had shared in first run for compliance checking. If only one argument is passed it is considered as username.
- Argument 2: **Absolute filepath -** It is the absolute path of the file that the user wants to check compliance for. It must have .xes extension. For running the file you could use “Example.xes” available in client directory inside StreamingEventCompliance project.

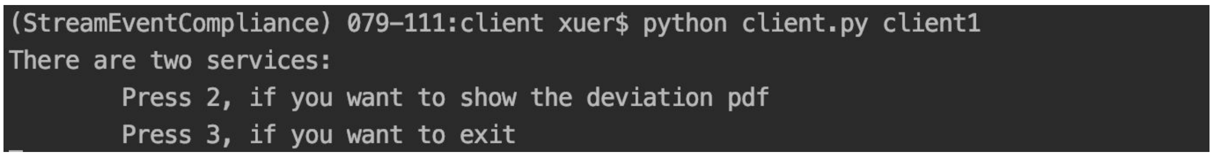
# 2 Source Code Description

## 2.1 Client

### 2.1.1 Menu for the User

Based on the arguments passed during execution of client.py the menu option will be displayed.
If one argument is passed, we consider it to be username and assume the compliance checking is already done and hence user has option only to see the deviation pdf or exiting

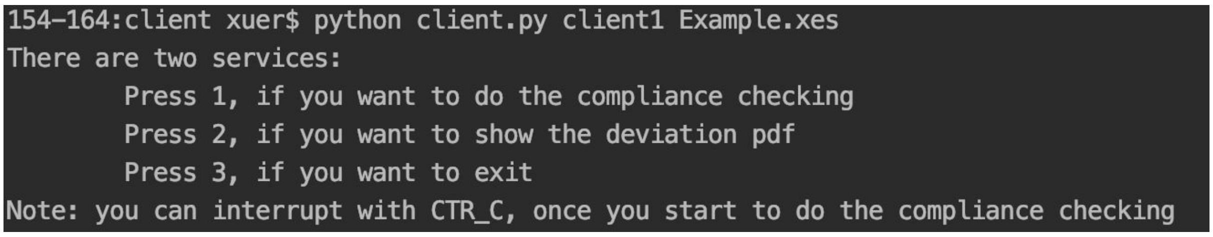
the client session.



```
(StreamEventCompliance) 079-111:client xuer$ python client.py client1
There are two services:
        Press 2, if you want to show the deviation pdf
        Press 3, if you want to exit
```

Selecting 2 will allow user to receive a pdf file containing deviations. This file is fetched from server database based on the username.
Selecting 3 will allow user to exit the loop and end the client session.

If two arguments are passed the client offers three options to the user as shown in screenshot below.



```
154-164:client xuer$ python client.py client1 Example.xes
There are two services:
        Press 1, if you want to do the compliance checking
        Press 2, if you want to show the deviation pdf
        Press 3, if you want to exit
Note: you can interrupt with CTR_C, once you start to do the compliance checking
```

Selecting 1 will allow the user to run compliance checker service from the server for the file provided by the user while initiating the client
Selecting 2 will allow user to receive a pdf file with deviation if the user had already done the compliance check else it will display error message
Selecting 3 will allow user to exit the loop and end the client session.

### 2.1.2 Logging for Client

The client offers  two functions for logging.

- log_info()  -  It is used to log *info* messages
- log_error() -  It is used to log *error* messages

We we will have three types of format stored in log file.

1. logging any normal event

   *Timesstamp-MessageType-Username-FunctionName-Message*
2. logging the event specific data

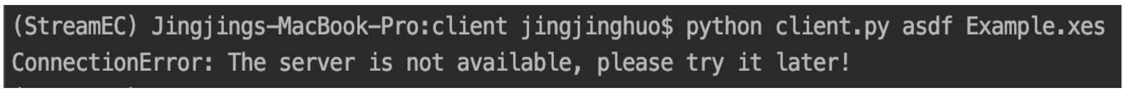
   *Timesstamp- MessageType-Username-FunctionName-ThreadId-CaseId-Activity-Message*
3. built-in log format for http requests

We can change the default values for logging level, time format and filename by manipulating the client/config.py file for variables LOG_LEVEL and LOG_FORMAT, CLIENT_LOG_PATH respectively.

### 2.1.3 Client Exception

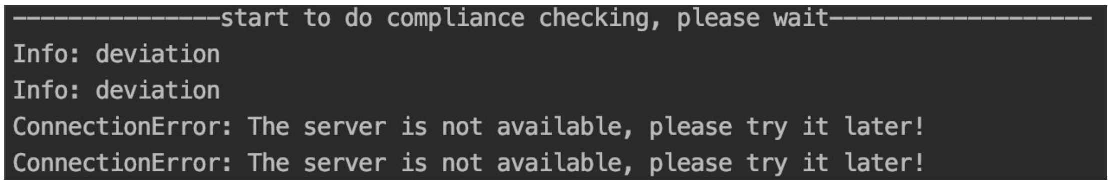
The client offers two exceptions:

1. ConnectionException -
   This exception will occur when user is not able to access server at the beginning:



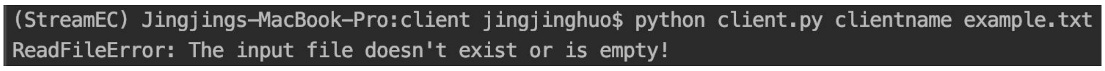
```
(StreamEC) Jingjings-MacBook-Pro:client jingjinghuo$ python client.py asdf Example.xes
ConnectionError: The server is not available, please try it later!
```

   or during the compliance checking the server break down:



```
---------------start to do compliance checking, please wait--------------------
Info: deviation
Info: deviation
ConnectionError: The server is not available, please try it later!
ConnectionError: The server is not available, please try it later!
```

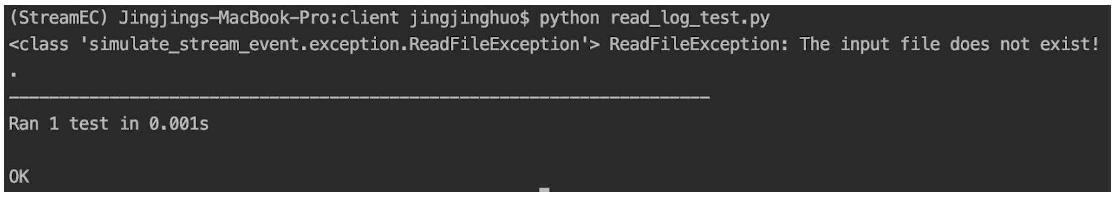
2. ReadFileException - This exception will occur when client is not able to locate file, read the file, import the file into tracelog or eventlog.



```
(StreamEC) Jingjings-MacBook-Pro:client jingjinghuo$ python client.py clientname example.txt
ReadFileError: The input file doesn't exist or is empty!
```

   Testing: Run the command (under the folder “client”):

   *python read_log_test.py*



```
(StreamEC) Jingjings-MacBook-Pro:client jingjinghuo$ python read_log_test.py
<class 'simulate_stream_event.exception.ReadFileException'> ReadFileException: The input file does not exist!
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

## 2.2 Server

### 2.2.1 Build Automata

#### 2.2.1.1. Logic for Building Automata   (Training automata)

**Data Structures used:**

- **CaseMemorizer.dictionary_cases** is used to store the cases that are being processed currently.
  It’s of the dictionary type containing several sub_dicts. The last 4 events and all the events that are not processed in a case will be stored in the list of corresponding sub_dict. That means, in every sub_dict the first 4 events are the latest events that have been processed and the 5th event is always the current event which has to be processed.
  Initially the CaseMemorizer is assigned ‘ * ’ when there there are no events processed for a case_id. If a new event “a” has occurred with that case_id then

```
CaseMemorizer.dictionary_cases= {
        {
                case_id1 : [ * , * , * , * , ‘a’]
        }
}
```

During processing the event ‘a’, other new events from this case_id1 will add into the tail. When the process of the event ‘a’ is finished, an old event will be removed from the head. Then the event ‘b’ that should be next processed will still in the position 5.

```
CaseMemorizer.dictionary_cases = {
        {
                case_id1 : [ * , * , * , ‘a’, ‘b’]
        }
}
```

- **windowsMemory:** It contains the list of activities for one case_id that we need for processing the current event, i.e. event ‘e’ in the following case.

  windowsMemory = [ ‘a’, ‘b’ , ‘c’ , ‘d’ , ‘e’]

  Here actually the windowsMemory is the first 5 events in cooresponding sub_dict in CaseMemorizer

- **autos** dictionary is created based on automata class with window size(int), nodes (dictionary) and connections(list). This is the automata that is created after training.

```
autos= {
        1 : { window size: 1,nodes: {node: ‘’, degree: ‘’}, connection_list : “list”} ,
        2 : { window size: 2,nodes: {node: ‘’, degree: ‘’}, connection_list : “list”} ,
        3 : { window size: 3,nodes: {node: ‘’, degree: ‘’}, connection_list : “list”} ,
        4 : { window size: 4,nodes: {node: ‘’, degree: ‘’}, connection_list : “list”} ,
}
```

**Program Flow:**

1. A new event is read and appended to CaseMemorizer based on the case_id.
2. Create a new thread for this event
3. Using the caseMemorizer, windowsMemory is calculated which has the latest 4 activities and the current activity which is being processed(in-total 5 activities).
4. This windowsMemory is further used to calculate source_node and sink_node for different window sizes.

   For example:

   windowsMemory = [ ‘a’, ‘b’ , ‘c’ , ‘d’ , ‘e’]

| **window size** | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| **source_node** | d | c,d | b,c,d | a,b,c,d |
| **sink_node** | e | d,e | c,d,e | b,c,d,e |

*Note: To calculate  the source_node and sink_node we need only 4 latest activities for a particular case_id hence, we only store the lastest 4 activities plus current processed activity in the windowsMemory.*

5. The calculated source_node and sink_node is inserted into **autos** dictionary by incrementing the count in case the the connection and nodes already exist -update_node(),update_automata(). (*Functions described in Database section-2.2.1.2).*
6. After running the execution of all the threads the probability is calculated using set_probability() *(Function described in Database section-2.2.1.2)*
7. Finally all the autos data is inserted into Database using insert_node_and_connection(autos) (*Function described in Database section-2.2.1.2)*

### 2.2.1.2 Server-side Thread handling :

Data structures used

- **CaseMemorizer.lock_list:** It's a dict and also stored in CaseMemorizer, containing the case_id and a lock of type Class threading.Lock(built-in). Every case will have a lock, and it  will be created when a event from a new case comes. When one thread of an event starts, it will first check if the lock for the case of this event is locked. Only when this thread get the corresponding lock, the windowMemory can be calculated.
- The structure is as below:

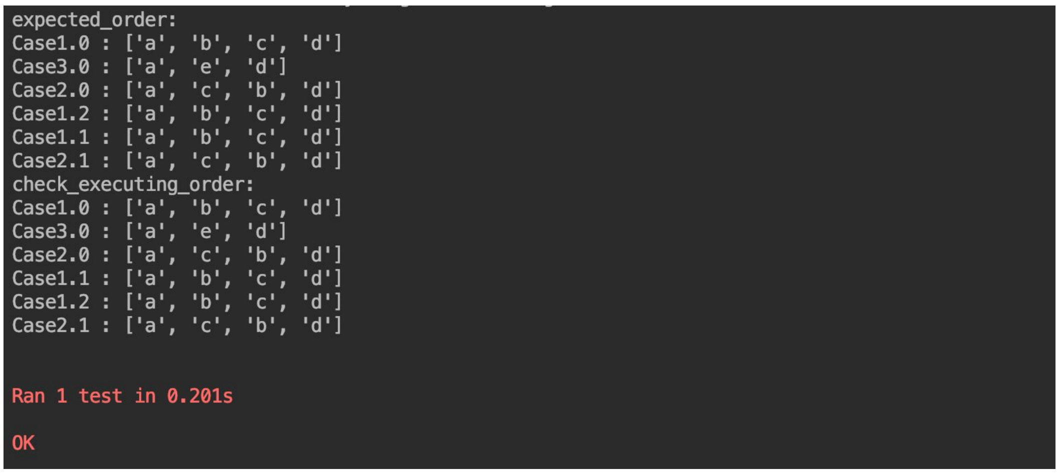
```
lock_list = {
        case_id1 : _thread_lock,
        case_id2 : _thread_lock,
        case_id3:  _thread_lock,
}
```

- **ThreadMemorizer:** It is used to store thread related data.

**Program Flow:**

1. A thread is created for every event that is read from the file
2. For each event, the lock  for the case_id is checked. Based on this, the access to caseMemorizer for that case_id is prohibited. If it is not locked then, windowsMemory is created from caseMemorizer and processing is done for generating connections of varying window sizes and appended to automata (autos dictionary) .
3. The lock is released from the caseMemoriser and now other threads can use the caseMemorizer to create windowsMemory and process it further to create automata.
4. After this all the threads are joined.

**Testing:** *building_automata_test.py*


```
expected_order:
Case1.0 : ['a', 'b', 'c', 'd']
Case3.0 : ['a', 'e', 'd']
Case2.0 : ['a', 'c', 'b', 'd']
Case1.2 : ['a', 'b', 'c', 'd']
Case1.1 : ['a', 'b', 'c', 'd']
Case2.1 : ['a', 'c', 'b', 'd']
check_executing_order:
Case1.0 : ['a', 'b', 'c', 'd']
Case3.0 : ['a', 'e', 'd']
Case2.0 : ['a', 'c', 'b', 'd']
Case1.1 : ['a', 'b', 'c', 'd']
Case1.2 : ['a', 'b', 'c', 'd']
Case2.1 : ['a', 'c', 'b', 'd']


Ran 1 test in 0.201s

OK
```

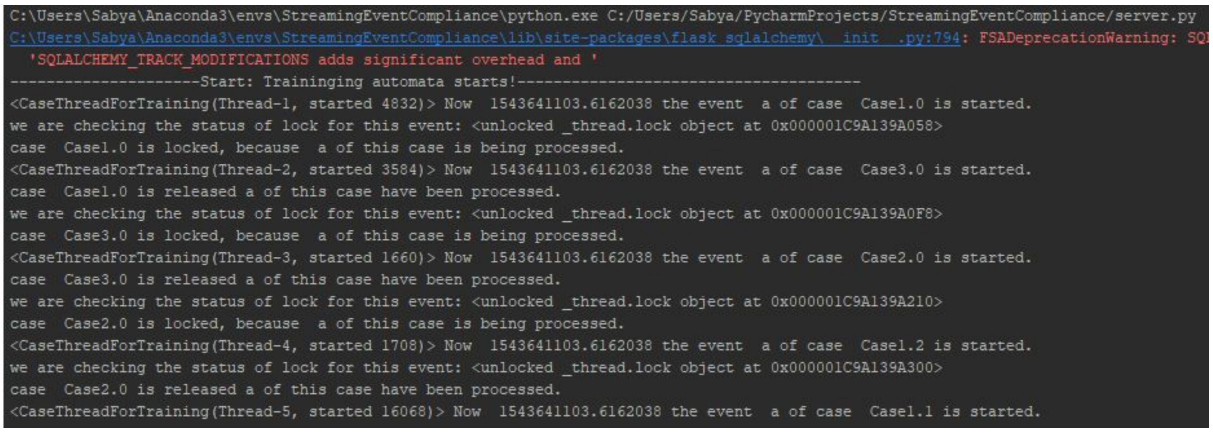
### 2.2.2 Database Functions:

- empty_tables(): clear the tables
- insert_node_and_connection(autos): insert the automata with different window size to the databases
- update_node(self, node, count): insert source_node into the nodes dictionary of autos(automata). In case, the node is already available then increment the degree.
- update_automata(self, connection): insert source_node, sink_node and count into connections list of autos(automata). In case, connection is already available increment the count
- set_probability(autos): calculate the probability based on count of connections from connections list and degree of source_node from node dictionary.
- insert_alert_log(alogs): insert the alert log with different window size to the database
- create_user(uuid): create user with uuid as user name and status as False and store in database
- check_user_status(uuid): check the status of the user in database with uuid as user name
- update_user_status(uuid, status): update the status of user in database with uuid as user
- init_automata_from_database(): read the connections and node table from the database and store into a automata object in memory and return the automatas with different window size in the form of dictionary like {1: auto1, 2: auto2}
- init_alert_log_from_database(uuid): read alert records of the particular user with uuid as user name from the database and store the alert log in memory and return the alertlogs with different window size in the form of dictionary like {1: alog1, 2: alog2}
- delete_alert(): delete alert records of the particular user with uuid as user name

## 3 Testing Project

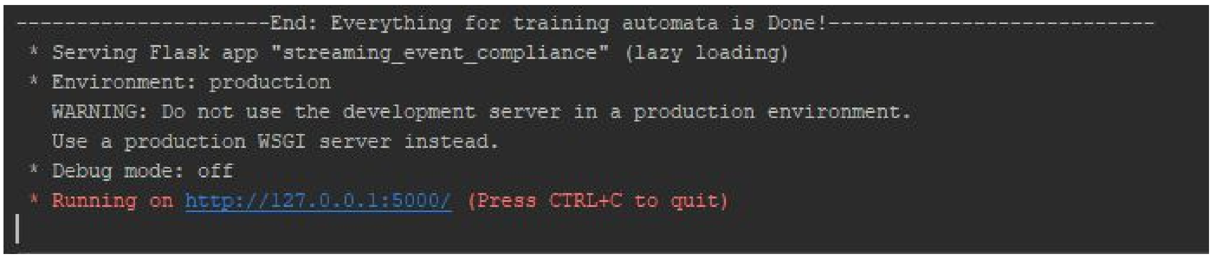
In this section we discuss how to test the output of the entire project.
(Unit test cases for each function has been written in test section of the code)

**Step 1: Run: python server.py** *(explained in section- 1.1)*
You'll see the below data on the console. The training automata logs are printed on console. Details like case_id is locked, the thread is starting etc is shown.



```
C:\Users\Sabya\Anaconda3\envs\StreamingEventCompliance\python.exe C:/Users/Sabya/PycharmProjects/StreamingEventCompliance/server.py
C:\Users\Sabya\Anaconda3\envs\StreamingEventCompliance\lib\site-packages\flask_sqlalchemy\__init__.py:794: FSADeprecationWarning: SQ
  'SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and '
--------------------Start: Traininging automata starts!--------------------------------------
<CaseThreadForTraining(Thread-1, started 4832)> Now  1543641103.6162038 the event  a of case  Case1.0 is started.
we are checking the status of lock for this event: <unlocked _thread.lock object at 0x000001C9A139A058>
case  Case1.0 is locked, because  a of this case is being processed.
<CaseThreadForTraining(Thread-2, started 3584)> Now  1543641103.6162038 the event  a of case  Case3.0 is started.
case  Case1.0 is released a of this case have been processed.
we are checking the status of lock for this event: <unlocked _thread.lock object at 0x000001C9A139A0F8>
case  Case3.0 is locked, because  a of this case is being processed.
<CaseThreadForTraining(Thread-3, started 1660)> Now  1543641103.6162038 the event  a of case  Case2.0 is started.
case  Case3.0 is released a of this case have been processed.
we are checking the status of lock for this event: <unlocked _thread.lock object at 0x000001C9A139A210>
case  Case2.0 is locked, because  a of this case is being processed.
<CaseThreadForTraining(Thread-4, started 1708)> Now  1543641103.6162038 the event  a of case  Case1.2 is started.
we are checking the status of lock for this event: <unlocked _thread.lock object at 0x000001C9A139A300>
case  Case2.0 is released a of this case have been processed.
<CaseThreadForTraining(Thread-5, started 16068)> Now  1543641103.6162038 the event  a of case  Case1.1 is started.
```

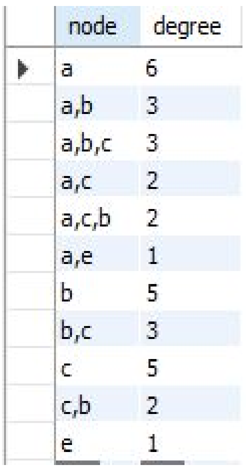
Once the training is completed you can see the below data on the screen



```
----------------------End: Everything for training automata is Done!----------------------------
 * Serving Flask app "streaming_event_compliance" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

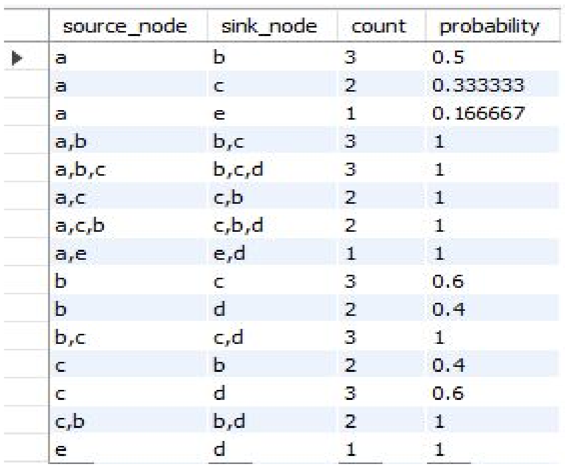
You can also check the data in database under connection and node tables.

**node table:**



| node | degree |
|---|---|
| a | 6 |
| a,b | 3 |
| a,b,c | 3 |
| a,c | 2 |
| a,c,b | 2 |
| a,e | 1 |
| b | 5 |
| b,c | 3 |
| c | 5 |
| c,b | 2 |
| e | 1 |

**connection table:**

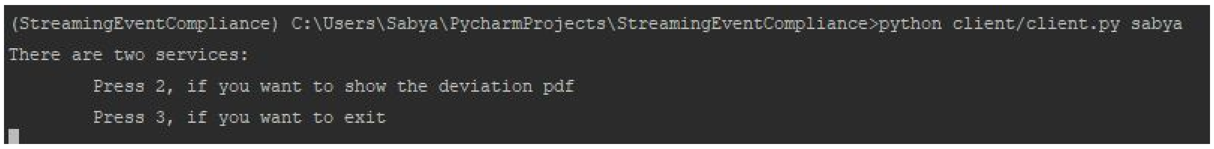


| | source_node | sink_node | count | probability |
|---|---|---|---|---|
| ▶ | a | b | 3 | 0.5 |
| | a | c | 2 | 0.333333 |
| | a | e | 1 | 0.166667 |
| | a,b | b,c | 3 | 1 |
| | a,b,c | b,c,d | 3 | 1 |
| | a,c | c,b | 2 | 1 |
| | a,c,b | c,b,d | 2 | 1 |
| | a,e | e,d | 1 | 1 |
| | b | c | 3 | 0.6 |
| | b | d | 2 | 0.4 |
| | b,c | c,d | 3 | 1 |
| | c | b | 2 | 0.4 |
| | c | d | 3 | 0.6 |
| | c,b | b,d | 2 | 1 |
| | e | d | 1 | 1 |

This means now the server is idle and available for the client.

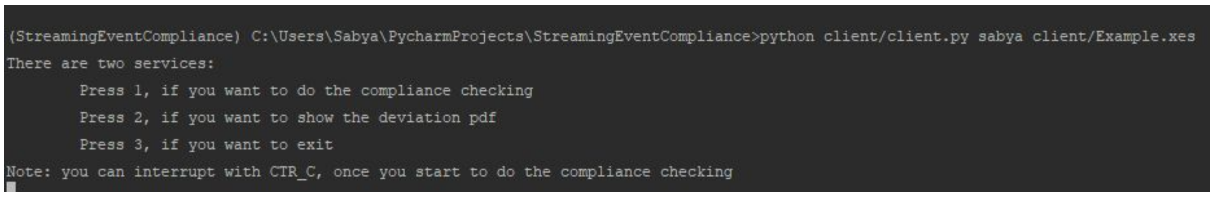
**Step 2: Run: python client/client.py <username>  client/Example.xes** *(explained in section- 1.1)*

When only one argument is passed



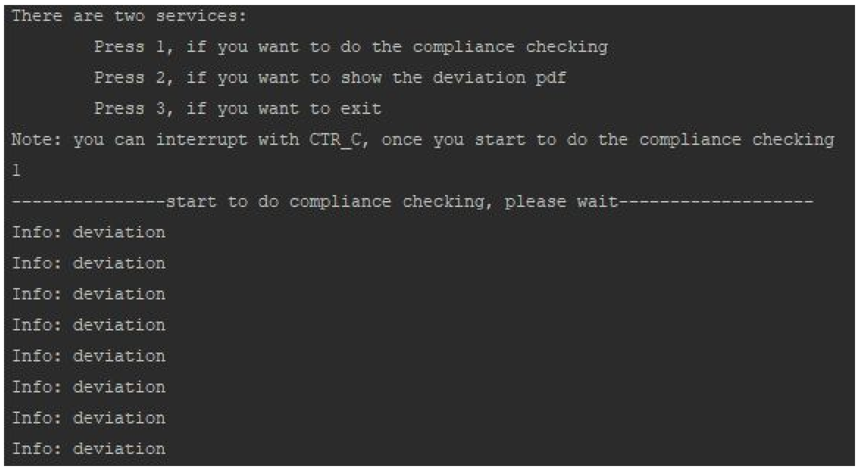
```
(StreamingEventCompliance) C:\Users\Sabya\PycharmProjects\StreamingEventCompliance>python client/client.py sabya
There are two services:
        Press 2, if you want to show the deviation pdf
        Press 3, if you want to exit
```

When two arguments are passed



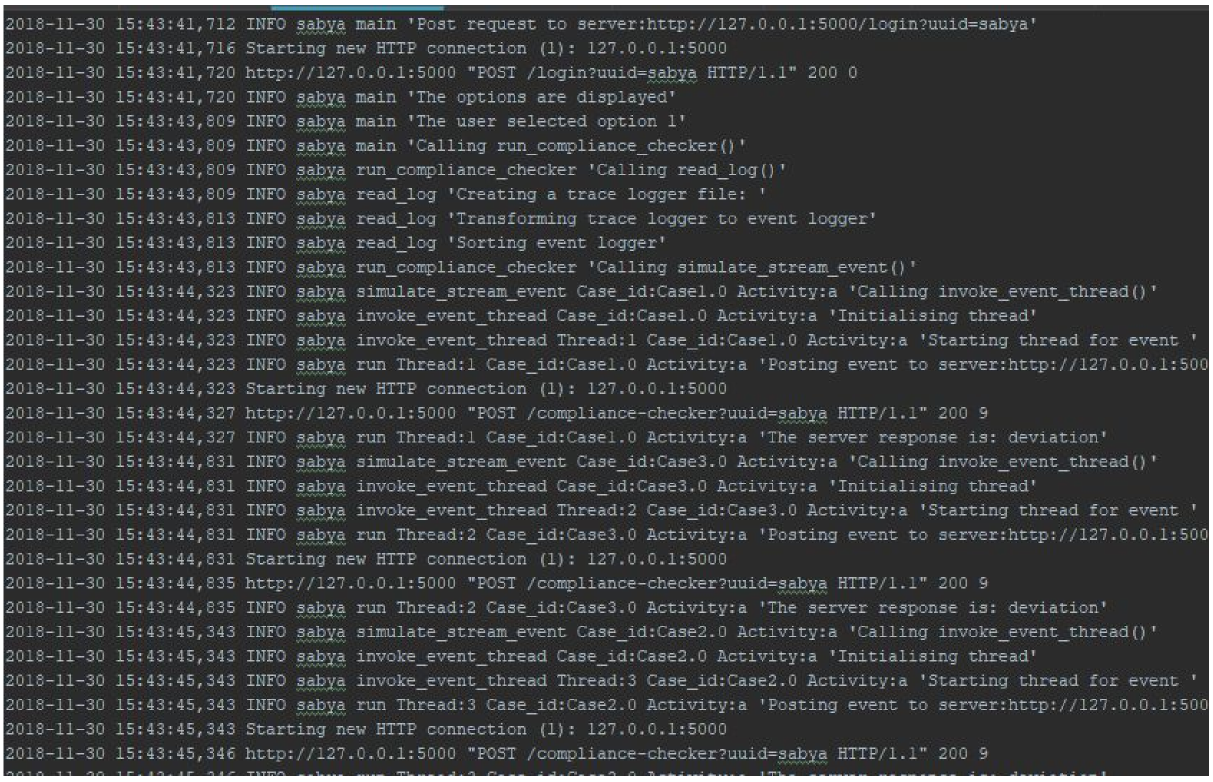
```
(StreamingEventCompliance) C:\Users\Sabya\PycharmProjects\StreamingEventCompliance>python client/client.py sabya client/Example.xes
There are two services:
        Press 1, if you want to do the compliance checking
        Press 2, if you want to show the deviation pdf
        Press 3, if you want to exit
Note: you can interrupt with CTR_C, once you start to do the compliance checking
```

Step 3: **Select 1** to do compliance checking



```
There are two services:
        Press 1, if you want to do the compliance checking
        Press 2, if you want to show the deviation pdf
        Press 3, if you want to exit
Note: you can interrupt with CTR_C, once you start to do the compliance checking
1
---------------start to do compliance checking, please wait-------------------
Info: deviation
Info: deviation
Info: deviation
Info: deviation
Info: deviation
Info: deviation
Info: deviation
Info: deviation
```

Currently the data is static which is sent from the server as the compliance checking is yet to be done. The threading and logging features are implemented.
Logs can be checked in **client/client_log.log**



```
2018-11-30 15:43:41,712 INFO sabya main 'Post request to server:http://127.0.0.1:5000/login?uuid=sabya'
2018-11-30 15:43:41,716 Starting new HTTP connection (1): 127.0.0.1:5000
2018-11-30 15:43:41,720 http://127.0.0.1:5000 "POST /login?uuid=sabya HTTP/1.1" 200 0
2018-11-30 15:43:41,720 INFO sabya main 'The options are displayed'
2018-11-30 15:43:43,809 INFO sabya main 'The user selected option 1'
2018-11-30 15:43:43,809 INFO sabya main 'Calling run_compliance_checker()'
2018-11-30 15:43:43,809 INFO sabya run_compliance_checker 'Calling read_log()'
2018-11-30 15:43:43,809 INFO sabya read_log 'Creating a trace logger file: '
2018-11-30 15:43:43,813 INFO sabya read_log 'Transforming trace logger to event logger'
2018-11-30 15:43:43,813 INFO sabya read_log 'Sorting event logger'
2018-11-30 15:43:43,813 INFO sabya run_compliance_checker 'Calling simulate_stream_event()'
2018-11-30 15:43:44,323 INFO sabya simulate_stream_event Case_id:Case1.0 Activity:a 'Calling invoke_event_thread()'
2018-11-30 15:43:44,323 INFO sabya invoke_event_thread Case_id:Case1.0 Activity:a 'Initialising thread'
2018-11-30 15:43:44,323 INFO sabya invoke_event_thread Thread:1 Case_id:Case1.0 Activity:a 'Starting thread for event '
2018-11-30 15:43:44,323 INFO sabya run Thread:1 Case_id:Case1.0 Activity:a 'Posting event to server:http://127.0.0.1:500
2018-11-30 15:43:44,323 Starting new HTTP connection (1): 127.0.0.1:5000
2018-11-30 15:43:44,327 http://127.0.0.1:5000 "POST /compliance-checker?uuid=sabya HTTP/1.1" 200 9
2018-11-30 15:43:44,327 INFO sabya run Thread:1 Case_id:Case1.0 Activity:a 'The server response is: deviation'
2018-11-30 15:43:44,831 INFO sabya simulate_stream_event Case_id:Case3.0 Activity:a 'Calling invoke_event_thread()'
2018-11-30 15:43:44,831 INFO sabya invoke_event_thread Case_id:Case3.0 Activity:a 'Initialising thread'
2018-11-30 15:43:44,831 INFO sabya invoke_event_thread Thread:2 Case_id:Case3.0 Activity:a 'Starting thread for event '
2018-11-30 15:43:44,831 INFO sabya run Thread:2 Case_id:Case3.0 Activity:a 'Posting event to server:http://127.0.0.1:500
2018-11-30 15:43:44,831 Starting new HTTP connection (1): 127.0.0.1:5000
2018-11-30 15:43:44,835 http://127.0.0.1:5000 "POST /compliance-checker?uuid=sabya HTTP/1.1" 200 9
2018-11-30 15:43:44,835 INFO sabya run Thread:2 Case_id:Case3.0 Activity:a 'The server response is: deviation'
2018-11-30 15:43:45,343 INFO sabya simulate_stream_event Case_id:Case2.0 Activity:a 'Calling invoke_event_thread()'
2018-11-30 15:43:45,343 INFO sabya invoke_event_thread Case_id:Case2.0 Activity:a 'Initialising thread'
2018-11-30 15:43:45,343 INFO sabya invoke_event_thread Thread:3 Case_id:Case2.0 Activity:a 'Starting thread for event '
2018-11-30 15:43:45,343 INFO sabya run Thread:3 Case_id:Case2.0 Activity:a 'Posting event to server:http://127.0.0.1:500
2018-11-30 15:43:45,343 Starting new HTTP connection (1): 127.0.0.1:5000
2018-11-30 15:43:45,346 http://127.0.0.1:5000 "POST /compliance-checker?uuid=sabya HTTP/1.1" 200 9
```

## 4 Phase Review

**Zheqi Lyu:**
In this phase we focus on manipulating database, building automata and completing the client, exception and logging at the phase. And my main issue took account for database and client. The main difficulties that I met were database updating with flask and how to set a global value(to do with jingjing). Previously I used different db.session to create the database and update database and only a part of the events could be added to the database, because I didn't know how to call a variable, which initiates after we create a app=Flask(__main__). So I created different db.session and tried to push them to the app after the server is running. But it didn’t work. For a deeper reason I didn’t quite understand the structure of python module and package, especially for the file “__init__.py”. Since we use thread and multiprocessor, now I have a clear idea of the difference between thread and processor. Summarizing the above, I was confused at the initial stage, but I learned a lot from this phase.

**Sabya Shaikh:** We have made a great progress in the implementation of the code and i feel we are right on track. The step by step building of code is really helping to code easily. Distributing the tasks in this case was necessary. Although we were helped by each other to gain insights on the coding standards and technique.There was overlapping of the tasks with each others. For eg, Building automata had a little of everyone's contribution as we could not completely decouple it from other tasks. We often met to discuss how to proceed further in case of any doubts or hurdles. Overall, it was a good sprint.

**Jingjing Huo:** This phase most time I was working on multi-threads handling, and some exceptions on the client side. Sometimes I spent a lot of time in solving a very tiny problem, I figure out that I need a more accurate positioning of the problem and a more efficient search for solving it. Maybe next time I should learn to make full use of the debugging tools and think about the problem more before I search it. Or maybe I need to look at the docstrings of python first instead of going online when I meet a problem. In terms of cooperation among team members, I think everything works fine and we help each other and progress together.