

# Design and P.o.C. of web services and client

Streaming event data compliance checking in Python

Jingjing Huo 376323

Sabya Shaikh 384606

Zheqi Lyu 378653

## 1 Service/Back-end

### 1.1 API

#### 1.1.1 Route Interface

#### 1.1.2 services

#### 1.1.3 objects

#### 1.1.4 utils

### 1.2 Exception Handling

### 1.3 Logging

### 1.4 CORS

## 2 Client/Front-end

### 2.1 Functionality

#### 2.1.1 Check compliance

#### 2.1.2 Request Deviations PDF file

### 2.2 Exception Handling

### 2.3 Logging

## 3 History-reporting feature

## 4 Unit Tests

### 4.1 Unit Tests for Client

#### 4.1.1 Unit Tests for Check compliance

#### 4.1.2 Unit Test for Request Deviations Pdf file

### 4.2 Unit Tests for Server

#### 4.2.1 Unit Tests for buildAutomata

#### 4.2.2 Unit Test for complianceChecker

## 5 Phase Review

## Appendix 1 -- Sequence Diagram

## Appendix 2 -- Result Snippet

# 1 Service/Back-end

Currently we have listed three services offered by the server:

**complianceChecker:** Checking compliance of the event.

**deviationsPDF:** Giving a PDF file showing the deviations details.

**buildAutomata:** Initializing automata by training local event log.

## 1.1 API

### 1.1.1 Route Interface

#### ***call\_compliance\_check()***

**Description:** This function provides the interface to check the compliance of the event.

**URL:** [http://127.0.0.1:8000/compliance-checker?uuid=client\\_uuid](http://127.0.0.1:8000/compliance-checker?uuid=client_uuid)

**Method:** POST

**Path Parameter:** client\_uuid: It is client identification name that has requested the service. It is provided by client.

**Body Data Type:** application/json

**Body Content:**

```
event= {  
    case_id: "string"  
    activity: "string"  
}
```

**Response:**

**Body Data type:** status code, application/json

**Body Content:** e.g. 200, deviation information in json

#### ***call\_deviation\_pdf()***

**Description:** This function will render the deviation pdf with particular client\_id in the browser, when the client has already done the "compliance checking".

**Url:** [http://127.0.0.1:8000/show-deviation-pdf?uuid = client\\_uuid](http://127.0.0.1:8000/show-deviation-pdf?uuid = client_uuid)

**Method:** GET

**Path parameter:** client\_uuid: It is client identification name that has requested the service. It is provided by client.

**Body Data Type:** Null

**Body Content:** Null

**Response:**

**Body Data type:** status code, application/pdf

**Body Content:** e.g. 200, pdf

### 1.1.2 services

This module describes the services on the server side.

[service.complianceChecker.py](#)

**Function:** complianceChecker.compliance\_checker(client\_uuid, event)

**Description:** This function does the compliance checking of the particular event received from client by comparing the automata information in the database. Returns alerts, when some deviations occur.

**param** client\_uuid: user name

**param** event: the event that we want to check the compliance

**return** text: the deviation information or success information.

[services.deviationPDF.py](#)

**Function:** deviationPDF.build\_deviation\_pdf(client\_uuid)

**Description:** Creates a deviation PDF for the given client\_uuid, based on the deviations history stored in AlertLog entity in database. This pdf is stored in local as "<client\_uuid>\_deviations.pdf".

**param** client\_uuid: user name

**return** bool: whether the file is created successfully

**Function:** deviationPDF.show\_deviation\_pdf(client\_uuid)

**Description:** Returns the file "<client\_uuid>\_deviations.pdf" if present in the local.

Else if no file present with that name then, this function calls the build\_deviation\_pdf(client\_uuid) to create a pdf

**param** client\_uuid: user name

**return** File: PDF File

[services.buildAutomata.py](#)

**Function:** buildAutomata.build\_automata()

**Description:** Reads the training event log from utils.config.TRAINING\_EVENT\_LOG and build automata. It generates the probability between SourceNode and SinkNode with different prefix size and stores corresponding information into the database.

**return** bool: indicate if the automata is created successfully.

**Function:** buildAutomata.test\_automata\_status()

**Description:** This function will check whether the automata is built. If the automata isn't yet built, then this function calls the build\_automata() function to build the automata. And the status will be stored in "Compliance.config". It raises the exceptions when the automata can not be built.

**return** bool: the status of the automata

### [1.1.3 objects](#)

This module describes the objects, which are referred in this project.

[objects.automata.automata.py](#)

**Class** Node: contains attribute: unique node name and the outgoing degree of the node

**Description:** this object represents the source node or sink node in window size with different prefix size.

**Class:** Automata

**Description:** this object has manytomany relationship between source node and sink node, with extra informations, i.e. connection and probability.

[objects.automata.alertLog.py](#)

**Class** User: contains attribute: unique user name and status of compliance checking

**Description:** This entity will contain all the usernames that have previously requested for conformance checking and whether the automata was build successfully will be stored in status field.

**Class** AlertLog: contains attribute: user\_id, window\_size, source\_node, sink\_node, alert\_count, alert\_type

**Description:** Each alert will be a instance of this object.

[objects.exceptions.exception.py](#)

This module describes the exception objects.

**Class:** EventException, AutomataException, ConnectionDBException, FileException  
(Detail see chapter Exception Handling)

#### 1.1.4 utils

This module will provide some convenient utilities, for instance, config.py contains some default path and parameters.

## 1.2 Exception Handling

1. **EventError:** Received an incomplete event or data type of received event is incorrect.Eg: case\_id is missing or activity is missing, activity is integer
2. **AutomataError:** Request to show Automata is received but the automata is not built due to internal issues. (isn't error)
3. **ConnectDBError:** Connection to Database refused, when inserting or accessing the database.
4. **FileError:** 1. "case:id" and "concept:case:id" are not found in EventLog/ file format error when reading Event log file for training; 2. Deviations Pdf creation failed when writing

## 1.3 Logging

We plan to log all activities and also all events processed and at the server in the below three log files.

1.**training.log** -While training we shall log into *training.log* file stored in local

2.**test.log** -While testing we shall log into *test.log* file stored in local

The Training logs and Test logs shall have the below format:

*Timestamp-LoggingLevel-Process-Message*

**Timestamp** - It is the time when the particular activity is being logged in the log file

**LoggingLevel**- Text logging level for the message ('DEBUG', 'INFO', 'WARNING', 'ERROR', 'CRITICAL')

**Process**- It is function/process during which the event was logged

**Message**- User written message

For example :The message can contain the *Client\_uuid,thread\_Id, case\_id, activity*,  
Client\_uuid- Identifier for the client that has requested  
thread\_id - The id of the thread that was used to process the event  
case\_id and activity is from the events received via post from client

## 1.4 CORS

We plan to use “Route specific CORS via decorator”  
For example: for our service we shall add `@cross_origin()`

Installation: ***pip install -U flask-cors***

Import statement: ***from flask\_cors import CORS***

```
@app.route('/compliance-checker', methods=['POST'])
```

```
@cross_origin()
```

```
def call_compliance_checker():
```

```
@app.route('/show-deviation-pdf', methods=['GET'])
```

```
@cross_origin()
```

```
def call_show_deviation_pdf():
```

## 2 Client/Front-end

### 2.1 Functionality

Client can request for two services from the server

1. Check compliance of an event
2. Request for PDF with deviations

#### 2.1.1 Check compliance

1. The trace log file is read from the local path, converted to event log and sorted based on timestamp. This is executed in function - ***read\_log(path)***;
2. This event log is further simulated as streaming event using function ***simulate\_stream\_event(event\_log)***.

In this function each event in event\_log is converted to json and a function `invoke_event_thread(event)` is called to invoke thread for each event;

3. The ***invoke\_event\_thread(event)*** is used to start a thread and throw this event to server using the below service.

**r = requests.post("http://127.0.0.1:5000/compliance-checker?uuid=client\_uuid", json=event);** If the server is not available, the client will catch the exception "ConnectionException", interrupt throwing thread to the client and display the information in the console.

4. The client will parse the response from the server and display the response on the console in case of deviation. For example: "Deviation occurred for case id = <case\_id> and activity =<activity>";

### 2.1.2 Request Deviations PDF file

- The client requests the "show-deviation-pdf" service using the below script.  
**pdf=requests.get("http://127.0.0.1:5000/show-deviation-pdf?uuid=client\_uuid")**
- This pdf will be shown in the browser. The user can download it from the browser.

## 2.2 Exception Handling

1. Server is not available: **ConnectionError**: error code 61
2. No response of the request thread(Internet issues), connection **timed out** 110
3. The Path for EventLog is not available: **OSError**
4. "case:id" and "concept:case:id" are not found in EventLog/ file format error: **ReadFileError**,
5. The user interrupt the service, while the service is still running: **KeyboardInterrupt** (If the user know his uuid, he can request the pdf, that means the corresponding alerts in db still exist)
6. **ErrorFromServer**: 1. the server has checked that in the event there is no case\_id, server cannot process that event. 2: the automata is not built yet.

## 2.3 Logging

We plan to log all the activities and also all the events that are being processed by the client in the *client.log* file in the local.

The *client.log* file shall have format as below:

### *Timestamp-LogLevel-Process-Message*

**Timestamp** - It is the time when the particular activity is being logged in the log file

**LogLevel**- Text logging level for the message ('DEBUG', 'INFO', 'WARNING', 'ERROR', 'CRITICAL')

**Process**- It is function/process during which the event was logged

**Message**- User written message

For example: The message can contain the *Client\_uuid*, *thread\_id*, *case\_id*, *activity* for which the process is being logged

*Client\_uuid*- Identifier for the client that has requested

*thread\_id* - The id of the thread that was used to process the event

*case\_id* and *activity* is from the events received via post from client

### 3 History-reporting feature

The history of all the deviated events will be stored in AlertLog entity in database. The Automata entity along with the AlertLog will be used to create the pdf file containing “location” of deviations, e.g. which activities/transitions/states are most problematic. This feature can be requested using “show-deviation-pdf” service.

Alertlog structure is

#### **AlertLog**

Client_Id	Window_Size	Source_Node	Sink_Node	Alert_Cause	Alert_Count
Integer - Primary Key	Integer - Primary Key	String -Primary Key	String- Primary Key	Char -‘M’ or ‘T’	Integer

The field descriptions of **AlertLog** entity are as below:

**Client\_Id:** The client that requested the service for a particular event

**Window\_Size:** Type: Integer. Same as Automata

**Source\_Node:** Type: String. Same as Automata

**Sink\_Node:** Type: String.

Case 1: If the event can occur after the current Source\_Node but probability of this event to occur is lower than threshold then the Sink\_Node is that particular event with reference to Sink Entity;

Case 2: If the event should never occur after the current Source\_Node then then the Sink\_Node is that particular event with no reference to existing Automata.

**Alert\_Cause:** Type: Char: ‘M’ or ‘T’.

- ‘T’(Threshold): When the alert is due to lower probability of occurrence of Sink\_Node than threshold.
- ‘M’(Missing Sink\_Node from Automata): When there is no path between Source\_Node and Sink\_Node in Automata.

**Alert\_Count:** Type: Integer. The number of times the alert was triggered for a particular path from Souce\_Node to Sink\_Node.

## 4 Unit Tests

### 4.1 Unit Tests for Client

#### 4.1.1 Unit Tests for Check compliance: C001, C002, C003

C001: Reading&SortingEventLogTestCase		
Preconditions	Files are in computer	
Test Data	example_1.xes	example_2.txt/ example_3.excel
Steps	run read_log() function with this file	
Predict results	A sorted event log by timestamp (class: 'pm4py.log.log.EventLog')	Give information "The file form is wrong, please give the file in .xes form."

C002: Mutli-ThreadingThrowingEventTestCase		
Preconditions	The server is runing	
Test Data	Generating the event Objects (100) randomly in correct form	Generating the event Objects(10000) randomly in correct form
Steps	<ol style="list-style-type: none"><li>1. Create a client Object</li><li>2. Initialize T(ThreadMemorizer) and threads_index</li><li>3. Using requests.get(SERVER_URL + "/posts/count") to get number_original_posts</li><li>4. For each event invoke_event_thread(event, T, client_uuid)</li><li>5. After all the requests are sent, using requests.get(SERVER_URL + "/posts/count") to get number_current_posts</li><li>6. Compare the number_current_posts and the number_original_posts + the size of T</li></ol>	
Predict results	True	True

C003: SimulatingStreamingDataTestCase	
Precon ditions	<ol style="list-style-type: none"><li>1. The server is runing</li><li>2. The client Object is created</li><li>3. 'U123' is not in the table user in the database for TestData1</li><li>4. 'U456' is in the table user in the database for TestData1</li><li>5. In eventLog1 event.keys() does not contain 'concept:name' or 'case:concept:name'</li></ol>



Test Data	uuid = 'U123' & eventLog;	uuid = 'U456' & eventLog;	uuid = 'U123' & eventLog1
Steps	run simulate_stream_event('U123',eventLog)	run simulate_stream_event('U456',eventLog)	run simulate_stream_event('U123',eventLog1)
Predict results	successful invoke_event_thread() for each event in the eventLog	Give information "This username is used, please give another one."	The thread for this event will still be created with the json data {'activity': '?' & 'case_id': '?'}

#### 4.1.2 Unit Test for Request Deviations Pdf file: C004

C004: ShowDeviationPDFTestCase				
Preconditions	1.The server is running 2.Client is created 3.Conformance checking is done and the Alert information for this client has been stored in database	1.The server is running 2.Client is created 3.Conformance checking is done and the Alert information for this client has been stored in database	1.The server is running 2.Client is created 3.Conformance checking for this client has not yet executed	1.The server is running 2.client_uuid is wrong(This happens when the user enters the URL in the browser himself instead of clicking the url provided in the console.)
Test Data	client_uuid.pdf on local machine of the server side	None	None	client_uuid.pdf on local machine of the server side
Steps	call run_show_deviation_pdf() function with client information	1.call run_show_deviation_pdf() function with client information 2.In this case the target pdf cannot be found, but the server should connect to database and create a new pdf file for this client	call run_show_deviation_pdf() function with client information	1.Check whether provided client_uuid is in the database; 2.If yes,call run_show_deviation_pdf() function with this client_uuid; 3.If not, get the message "Please check your Username, we cannot find your information in our system" from the server.

Predict results	1.Print address to Client: “SERVER_URL/show-deviation-pdf?”+client.uuid 2.Click this address, the pdf will show in WebBrowser 3.The user can download it	The message “Please do conformance checking first” will be sent to the client	Url or Message
-----------------	---	--	----------------

## 4.2 Unit Tests for Server

### 4.2.1 Unit Tests for buildAutomata: S001, S002, S003

S001:ReadingEventLogForBuildingAutomataTestCase			
Preconditions	EventLog Files are in computer		
Test Data	EventLog_1.xes in which the content is totally correct	EventLog_2.xes in which some events don't have a 'case:id' key	EventLog_2.txt / example_3.xlsx
Steps	run read_log() function with these files	1.run read_log() function with these files 2.Simple ignore the event that has a wrong form	run read_log() function with these files
Predict results	Return An Event Log contains all events with necessary information (class: 'pm4py.log.log.EventLog')		Give information "The file forma is wrong, please give the file in .xes form."
Priority	Before S002		

S002: MultiThreadsForBuildingAutomataTestCase	
Preconditions	None
Test Data	EventLog1
Steps	1.Generate EventLog2 in which each case is sorted from EventLog1 2.Run buildAutomata.build_automata(EventLog1) 3.Store the processing sequence of events of each case 4.Compare these sequence with EventLog2 to see if all the events are processed in sequential
Predict results	Yes
Priority	After S001

S003: Writing&ReadingAutomataIntoOrFromDatabaseTestCase		
Preconditions	The database has been created	
Test Data	EventLog object contains all events with necessary information('case_id'&'activity') (class: 'pm4py.log.log.EventLog')	One particular event log that we can manually calculate the automaton
Steps	1. Connect to database 2. Run buildAutomata.build_automata() 3. Run buildAutomata.read_automata()	
Predict results	An object Automata	The result reading from database should be consistent with the automaton we manually created for that particular event log (Check the correctness of the probability of each edge).
Priority	After S002	

#### 4.2.2 Unit Test for complianceChecker: S004, S005, S006, S007

S004: Multi-ThreadingSynchronize&lockForCheckingTestCase		
Preconditions	None	
Test Data	EventLog1	
Steps	1.Create one client, providing client_uuid 2.Generate EventLog2 in which each case is sorted from EventLog1 3.Run complianceChecker.compliance_checker(client_uuid, event) for each event from EventLOf1 4.Store the processing sequence of events of each case 5. Compare these sequence with EventLog2 to see if all the events are processed in sequential	
Predict results	Yes	
Priority	After S003	

S005: AlertTestCase	
Preconditions	None
Test Data	EventLog1 with some suspicious events(Set S) we have known
Steps	1.Create one client, providing client_uuid 2.Generate EventLog2 in which each case is sorted from EventLog1 3.Run complianceChecker.compliance_checker(client_uuid, event) for each event from EventLOf1 4.Print all alerts that are created 5.Compare with set S to calculate the ratio
Predict results	Similarity should reach more than 95%
Priority	After S004

S006: GeneratPDFTestCase(This belongs to this part, because this function will automatically called at the end of the compliance checking of one eventlog)	
Preconditions	None
Test Data	None
Steps	1. Choose on client who have done the compliance checking 2.Reading all alerts form database 3.Draw the deviationAutomata manually 4.Generate deviationPDF by calling build_deviation_pdf(client) 5.Compare these two deviationAutomaton
Predict results	Exactly the same
Priority	After S005

S007: TimeOfConformanceCheckerTestCase	
Preconditions	None
Test Data	An EventLog Object
Steps	1.Create a client 2.Initial a timer 3.Cal complianceChecker,compliance_checker(client_uuid,event) for each Event fomr EventLog 4.Get the duration of processing
Predict results	The results should be more than 300 events per second
Priority	After S005

## 5 Phase Rewiew:

### **Zheqi Lyu:**

About team: This time I tried to assign some tasks to a particular person. But most of the time, we needed to discuss with each other. Because it was still about structure and there were many uncertainty. So we sat together and discussed our own understanding and cleared our doubts.

About this phase: During this phase, we constructed our server and client. And we tried to simulate every steps in implementation as exhaustively as possible. This helps us to have a clear skeleton of our project.

About me: Because of the lack of python skill, sometimes it may take long to solve it, when some problems came. So as usual, I'm trying to improve my python skill and do the project more efficiently. In the next phase, we will do as scrum programming. We will set up some small tasks, that need to be implemented in the first sprint, and assign to the particular person.

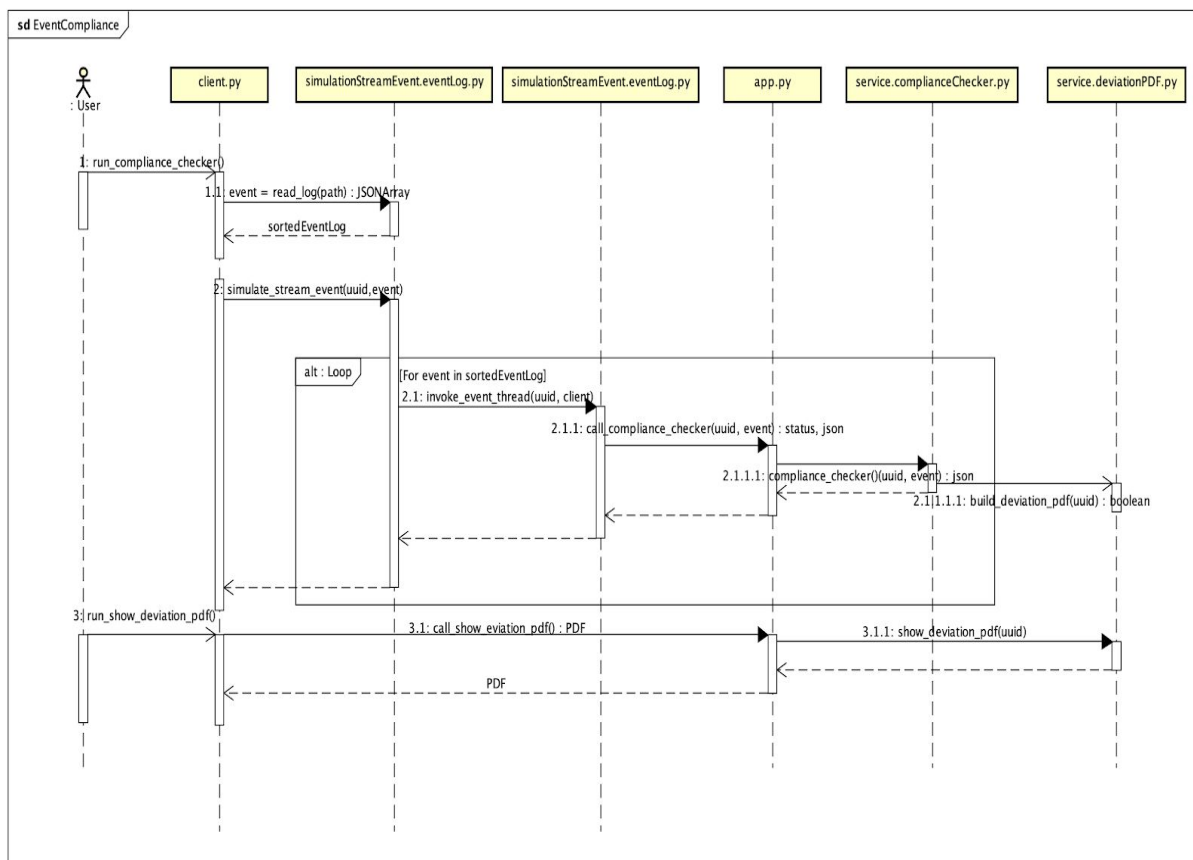
**Jingjing Huo:** Although the most of the tasks we still discuss together, and that makes the life easier, but because of the limitation of my python skill (or more general speaking, my programming capability), I feel that this phase is hard for me. I haven't much experience on multithreading or exceptions handling, and also haven't written Test Cases without explicit code in hand. So you can see that in addition to what I can learn in this limited time, I have fully utilized my imagination to write these things, so I really can't guarantee the quality. But this is a good start, by thinking these implement details I feel that I still have some problems in understanding our requirements or the skeleton of our project, I will solve them in the next stage. I hope everything gets better.

**Sabya Shaikh:** This phase helped us to get started with implementation. It was easier to build a skeleton of most of the functions that can be implemented in further stages. This helped to organise the Code/Project and to have an overview of how and what must be implemented in future.

My concern is with the sharing of the tasks. We sat together and discussed every details that went into the code and into the documentation. While one person coded in one machine the other two helped find the different ways to implement that functionality by googling the same on their machines. We feel this was effective to get each team member's insight on the code as well as on documentation. Each team member is free to correct each others work as we work parallely on shared doc. We feel meeting together and dedicating specific hours in a

week to work on Lab tasks is the best way to work as a team which makes everyone involved in every task. So it gets difficult to properly assign the tasks to a single person. Do let us know if you have any concerns with our working techniques.

## Appendix 1 -- Sequence Diagram



## Appendix 2 -- Result Snippet

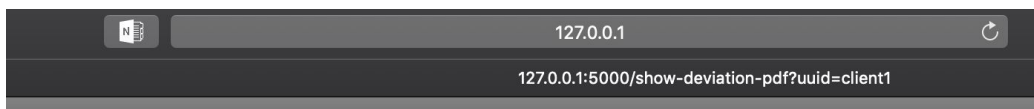
**client -- simulate\_stream\_event:**

```
def simulate_stream_event(client_uuid, event_log):
    for event in event_log:
        dic = {}
        for item in event.keys():
            if item == 'concept:name':
                dic['activity'] = event.get(item)
            elif item == 'case:concept:name':
                dic['case_id'] = event.get(item)
        invoke_event_thread(json.dumps(dic), T, client_uuid)
    end_message = {'case_id': 'NONE', 'activity': 'END'}
    invoke_event_thread(json.dumps(end_message), T, client_uuid)
```

**Default response from server, when client throws the event to the server:**

```
Info deviation
Info deviation
Info deviation
Info deviation
Info deviation
Info deviation
PDF is on http://127.0.0.1:5000/show-deviation-pdf?uuid=client1
Info deviation
Info deviation
Info deviation
```

**With the URL in the console, the client can attach the default deviation pdf in the browser:**




PDF

**Server accepts the compliance-check request from the client and displays the event in the console:**

```
case_id: Case1.1
activity: d
case_id: Case2.1
activity: d
case_id: NONE
activity: END
127.0.0.1 - - [20/Nov/2018 18:29:03] "POST /compliance-checker?uuid=client1 HTTP/1.1" 200 -
```

**Database Structure in MySQL, which are created by server:**

▼  compliancechecker

▼  Tables

▶  AlertLog

▶  Automata

▶  Node

▶  User