SCIMS Framework

Index

Purpose

This framework is use to test the SCMIS microservice. This framework handle the HTTP, TCP, WebSocket etc. communication.

Classes

HttpHandler

This class is used to handle the Http related request. User can provide the base URL and port to this class and handle send GET , POST and DELETE request. In future you can able to handle more HTTP methods using this class base on the use cases and request.

Methods

1. **send_get_request(self, base_url: str, paths: str, port: int, params: Dict, header: dict = None, timeout: float = 5.0) -> Response**: This method is used to send the GET request to HTTP connection. User need to provide the required parameters.

- **base_url**: User need to provide the base URL to this method for sending the HTTP request. Like *localhost*, *127.0.0.1* etc.
- **paths**: String, path in url. like: "/number". "/number/double/"
- **port**: Int, Port number for communication
- **params**: Dictionary, parameters need to be pass
- **header**: Dictionary, Headers which need to add inside request.
- **timeout**: Timeout in second's for request, Its an optional argument. Default value is 5.0 seconds.

This This method handle two exception

1. requests.exceptions.Timeout -> For request time out
2. Exception -> Generic Exception

This method return `Response` Object. User can call various method of this object to get the meaning full information from the response. **Example**:

```python
from Framework.HttpHandler import HttpHandler

http_handler = HttpHandler()
params = {
    "number": 2.2
}
response = http_handler.send_get_request(base_url="127.0.0.1", paths="/number",
port=8080, params=params)

print(response.status_code) # This will print the status code of the response like
200, 400, 404, 500, 501 etc.
print(response.text) # This will print the text from the response.
print(response.json()) # This will print the json-encoded content of a response,
if any
```

2. **send_post_request(self, base_url: str, paths: str, port: int, data: Dict, header=None, timeout: float = 5.0) -> Response:** This method is used to send the POST request of the HTTP.

- **base_url**: User need to provide the base URL to this method for sending the HTTP request. Like *localhost*, *127.0.0.1* etc.
- **paths**: String, path in url. like: "/number". "/number/double/"
- **port**: Int, Port number for communication
- **data**: Dictionary format for the data which need to be send with the POST request.
- **header**: Dictionary, Headers which need to add inside request.
- **timeout**: Timeout in second's for request, Its an optional argument. Default value is 5.0 seconds.

This This method handle two exception

1. requests.exceptions.Timeout -> For request time out
2. Exception -> Generic Exception

This method return `Response` Object. User can call various method of this object to get the meaning full information from the response. **Example**

```python
from Framework.HttpHandler import HttpHandler

http_handler = HttpHandler()
data = {
    "userName": "zebra.com",
    "password": "zebra123"
}
response = http_handler.send_post_request(base_url="127.0.0.1", paths="/number",
port=8080, data=data)

print(response.status_code) # This will print the status code of the response like
```

```
200, 400, 404, 500, 501 etc.
print(response.text) # This will print the text from the response.
print(response.json()) # This will print the json-encoded content of a response,
if any
```

3. **send_delete_request(self, base_url: str, paths: str, port: int, headers=None, timeout: float = 5.0) -> Response:** This method send the Delete request to provided base url and port number.

- **base_url**: User need to provide the base URL to this method for sending the HTTP request. Like *localhost*, *127.0.0.1* etc.
- **paths**: String, path in url. like: "/number". "/number/double/".
- **port**: Int, Port number for communication.
- **header**: Dictionary, Headers which need to add inside request.
- **timeout**: Timeout in second's for request, Its an optional argument. Default value is 5.0 seconds.

This This method handle two exception

1. requests.exceptions.Timeout -> For request time out
2. Exception -> Generic Exception

This method return `Response` Object. User can call various method of this object to get the meaning full information from the response. **Example**

```python
from Framework.HttpHandler import HttpHandler

http_handler = HttpHandler()

response = http_handler.send_post_request(base_url="127.0.0.1", paths="/number",
port=8080)

print(response.status_code) # This will print the status code of the response like
200, 400, 404, 500, 501 etc.
print(response.text) # This will print the text from the response.
print(response.json()) # This will print the json-encoded content of a response,
if any
```

TcpHandler:

This class handle the TCP related request. TCP communication is happened on socket communication. So we have use the python Socket libraries to handle the TCP communication.

Methods

1. **__connect_to_tcp_port(self, base_url: str, port: int, timeout=5.0) -> socket:** This method is a private method and used to connect with the TCP server. Its take base url and port to connect with the TCP

socket. **If connection already establish then it will not got for connection and return object.**

- **base_url**: User need to provide the base URL to this method for sending the HTTP request. Like *localhost*, *127.0.0.1* etc.
- **port**: Int, Port number for communication.

This method handle two exception

- socket.error -> If any error related to socket comes
- Exception -> Generic Exception

This method `socket` object and by using this user can call handle various functions.

2. **send_value_to_tcp(self, base_url: str, port: int, value: bytes, timeout=5.0) -> str:**

This method is send the data to the given TCP port.

- **base_url**: User need to provide the base URL to this method for sending the HTTP request. Like *localhost*, *127.0.0.1* etc.
- **port**: Int, Port number for communication.
- **value**: Value which need to be send, It should be in bytes format.

```
# please pass like:
receive = tcp_handler.send_value_to_tcp(base_url="127.0.0.1", port=9000,
value=b'0', timeout=5.0)
                    # or
receive = tcp_handler.send_value_to_tcp(base_url="127.0.0.1", port=9000,
value="0".encode(),timeout=5.0)
```

- **timeout**: timeout for request, Its an Optional argument as default value is `5.0` seconds.

This method handle two exception

- socket.error -> If any error related to socket comes
- Exception -> Generic Exception

This method return `Data` which is received as response in String format. user can validate the response and perform the operation on it.

3. **split_receive_value(self, value) -> List:**

This method is used to split the String type value into list of char. for eg: `python a = "%4.0&" output: ["%", "4", ".", "0", "&"]`

- value: String value This function will return the `List of Char`. User can validate the value from the list or perform action base on requirement.

4. **close_socket_connection(self):** This method close the socket connection if open.

Example:

```
from Framework.TcpHandler import TcpHandler
from Framework.LogHandler import *

# create an object of TcpHandler
tcp_handler = TcpHandler()
# Send an value to the TCP port and capture value in one variable
receive = tcp_handler.send_value_to_tcp(base_url="127.0.0.1", port=9000,
value=b'15542', timeout=5.0)
# split that value to identify that response should start with % and end with &
receive = tcp_handler.split_receive_value(receive)
assert receive[0] == "%", "% is not in Response"
assert receive[-1] == "&", "& is not in Response"
# check if it will return correct value or not.
assert float("".join(receive[1:-1])) == pow(15542, 2), "Response not contain the
power of 15542"
print_info_log(f"Response: {float(''.join(receive[1:-1]))}")
```

WebSocketHandler:

This class handle the web socket communication. We have used python `websocket` library to handle the communication. Methods of the class as below:

1. __connect_to_ws(self, base_url: str, port: int, timeout=5.0) -> websocket.WebSocket:

This method is private and used to connect with the WebSocket. **If connection already establish then it will directly return the object.**

- **base_url**: User need to provide the base URL to this method for sending the HTTP request. Like *localhost*, *127.0.0.1* etc.
- **port**: Int, Port number for communication.
- **timeout**: Timeout for request

This method return `WebSocket Object` on which user can communicate.

2. **send_value_to_ws(self, base_url: str, port: int, value, timeout=5.0):**

This method is used to send and receive the data from WebSocket.

- **base_url**: User need to provide the base URL to this method for sending the HTTP request. Like *localhost*, *127.0.0.1* etc.
- **port**: Int, Port number for communication.
- timeout: Timeout for request, an Optional argument and default value is `5.0` seconds
- Response from the WebSocket
- close_ws_connection(self): """ This method is used to close the WS connection :return: None """

3. **close_ws_connection(self):** This method is used to close the WS connection if open.

Example

```
from Framework.WebSocketHandler import WebSocketHandler
from Framework.LogHandler import *

#create an object of WebSocket
ws = WebSocketHandler()
# send the value and receive the response into one variable
response = ws.send_value_to_ws(base_url="127.0.0.1", port=8090, value=64,
timeout=5.0)
# validate the response
assert float(response) == calculate_cube_root(64), "Invalidate response, Wrong
cube root value send by WebSocket"
```

Used

All these methods are design in such a way that user can reuse the method to handle the multiple scenarios. User can call these methods by passing different arguments and test multiple use cases. Lets take one example:

*User want to send the GET request on 2 different HTTP urls. like one on `http://127.0.0.1:8080` and second `http://192.168.1.3:8080`. And send the same data for both.

Code

```
from Framework.HttpHandler import HttpHandler

http_handler = HttpHandler()

response1 = http_handler.send_get_request(base_url="127.0.0.1", paths="/number",
port=8080, params={"number": 64})
response2 = http_handler.send_get_request(base_url="192.168.1.3", paths="/float",
port=8080, params={"number": 64})
```

User can also perform load on the port like:

```
from Framework.HttpHandler import HttpHandler

http_handler = HttpHandler()
for i in range(0, 500000):
    response1 = http_handler.send_get_request(base_url="127.0.0.1",
paths="/number", port=8080, params={"number": 64})
```

4. **def calculate_cube_root(self, value):** Calculate the cube root and return the value

- value: Value which cube root. This method return the cube root of the value

Performance Testing

Load on HTTP Request

Using this framework user can perform the load on HTTP port. User need to call the below method of the class:

- **def perform_load(self, base_url: str, port: int, paths: str, num_req: int, method: str = "GET", **kwargs):** This method is used to perform the load on HTTP port. User need to pass the num_req which is a number of request
- *base_url*: Base url of the HTTP
- *port*: HTTP port on which user need to send the request
- *paths*: path of the HTTP request
- *num_req*: Number of request which need to be send
- *method*: HTTP methods like GET POST DELETE
- *kwargs*: Other argument pleas check doc of HTTP methods.

This method return the **True** if all response are proper otherwise **False**.

Load on TCP request

Using this framework user can perform the load on TCP port. User need to call the below method of the class:

- **def perform_load_on_tcp(self, base_url: str, port: int, value: bytes, num_req: int, timeout=5.0):** This method is used to perform the load on the WebSocket. It will send number of continuous request on the provided url. User need to pass the How much requests need to be send.
- *base_url*: User need to provide the base URL to this method for sending the HTTP request. Like localhost, 127.0.0.1 etc.
- *port*: Int, Port number for communication
- *value*: Value which need to be send.
- *num_req*: Number of request which need to be send
- *timeout*: Time out for one request.

This methods Return **true** if all the request are fine Otherwise return **false**.

Load on WebSocket

Using this framework user can perform the load on WebSocket port. User need to call the below method of the class:

- **def perform_load_on_ws(self, base_url: str, port: int, value: bytes, num_req: int, timeout=5.0):**

This method is used to perform the load on the WebSocket. It will send number of continuous request on the provided url. User need to pass the How much requests need to be send.

- *base_url*: User need to provide the base URL to this method for sending the HTTP request. Like localhost, 127.0.0.1 etc.
- *port*: Int, Port number for communication
- *value*: Value which need to be send.

- *num_req*: Number of request which need to be send
- *timeout*: Time out for one request. This method return **True** if all the request are fine otherwise **False**

Future scope

In future we are planing to add the methods which is used to capture feature flag of the microservice. Base on the more request we can add more methods inside the framework.