ACIT 3855 – Lab 2 – Edge Service and Testing

Instructor	Mike Mulder (mmulder10@bcit.ca) – Sets A and C Tim Guicherd (tguicherd@bcit.ca) – Set B
Total Marks	10
Due Dates	Demo and Submission by End of the Lesson 3 Class: • Monday, Jan. 22nd for Set C • Thursday, Jan. 25th for Sets A and B

Purpose

- Build an Edge Service (Receiver Service) that receives your two event types and stores them in a JSON file in a specific format
- Testing of your Edge Service with PostMan and Apache jMeter

Part 1 – Stubbing Out the Service with Connexion

Reference: https://connexion.readthedocs.io/en/latest/

In your IDE (i.e., PyCharm, Visual Studio Code), create a new Python project. Call it **Receiver**. This will make it easier to know which service this represents later than calling it something like Lab2.

Install the following packages:

- connexion[flask]
- swagger-ui-bundle (needed for your app to generate a UI endpoint)

Copy the openapi.yml file containing your OpenAPI specification from Lab 1 into your project.

Create a basic connexion app called app.py:

```
import connexion
from connexion import NoContent

# Your functions here

app = connexion.FlaskApp(__name__, specification_dir='')
app.add_api("openapi.yml")

if __name__ == "__main__":
    app.run(port=8080)
```

For each event in your OpenAPI specification, create a function as follows:

- Named based on the operationId
- By default, the parameter for the requestBody data of an endpoint must be named as **body**
- Prints the parameter to the console (i.e., the raw Python dictionary)
- Returns a 201 response code (example: return NoContext, 201)

Run the connexion application. Go the following URL: http://localhost:8080/ui

You should see your API specification and are able to test it out.

Use PostMan to verify both of your service API endpoints.

Part 2 – Logging Your Event Data (In a Fixed Size Queue)

Store your recent events in a JSON file along with counts of how many of each you have received in total:

- Remove your print statements
- Replace them with a simple storage mechanism that writes the following data to a file called events.json:
    - The total number of events of your first type received (i.e., a count)
    - The total number of events of your second type received (i.e., a count)
    - The last 5 events of your first type received from newest to oldest (i.e., an array of objects)
    - The last 5 events of your second type received from newest to oldest (i.e., an array of objects)
    - The JSON for each event must contain the following:
        - Current timestamp (called received_timestamp). It should show the current date and time (to the fraction of a second). Use datetime to get the current date/time and strftime to format it to a string.
        - String showing the data from the event (called msg_data). This string should be generated as a formatted string (using concatenation, string formatting expressions or f-string) and contain the values of at least *two* properties your event message.
        - Note that new events are added to the beginning of the array
        - Once 5 events are in the list for the given type, the oldest event is removed from the end and the new event is added to the beginning. The length of the array should not exceed 5.
        - See the JSON example below.
    - There should only be one events.json file, but it will store the counts and recent events of each of your two types.

  Note: You could create a new function that takes in the event type and event message and updates the file with that new event (i.e., updates the counts, adds the event data to the appropriate list based on the event type). This function would then be called by the two functions that receive the request body data.
- Use a constant to define the maximum number of events to store (i.e., MAX_ EVENTS)
- Use a constant to define the filename where the events are stored (i.e., EVENT_FILE)

DO NOT store the data from the file in a global variable. Instead, read it from file each time you receive an event, update it and then overwrite the file with the updated JSON.

The contents of the JSON file will look something like this (with the keys and event data matching that of your system):


```
{
    "num_bp": 1,
    "recent_bp": [
        {
            "msg_data": "Patient d290f1ee-6c54-4b01-90e6-d701748f0851 with a BP of
120/80.",
            "received_timestamp": "2024-01-15 07:35:22.075212"
        }
    ],
    "num_hr": 10,
    "recent_hr": [
        {
            "msg_data": "Patient d290f1ee-6c54-4b01-90e6-d701748f0851 with a heart
rate of 85.",
            "received_timestamp": "2024-01-15 07:36:05.026523"
        },
        {
            "msg_data": "Patient d290f1ee-6c54-4b01-90e6-d701748f0851 with a heart
rate of 86.",
            "received_timestamp": "2024-01-15 07:36:06.482410"
        },
        {
            "msg_data": "Patient d290f1ee-6c54-4b01-90e6-d701748f0851 with a heart
rate of 81.",
            "received_timestamp": "2024-01-15 07:36:08.045196"
        },
        {
            "msg_data": "Patient d290f1ee-6c54-4b01-90e6-d701748f0851 with a heart
rate of 89.",
            "received_timestamp": "2024-01-15 07:36:09.579387"
        },
        {
            "msg_data": "Patient d290f1ee-6c54-4b01-90e6-d701748f0851 with a heart
rate of 78.",
            "received_timestamp": "2024-01-15 07:36:10.584571"
        }
    ]
}
```


Use PostMan (or equivalent) to verify both of your service API endpoints and the contents of the JSON file.

Part 3 – Strict Validation

Enable validation on the request and response of your API.

Add the following arguments to app.add_api:

- `strict_validation=True`
- `validate_responses=True`

Use PostMan to verify that your API endpoints still work and that invalid request messages are rejected.

Part 4 – Load Testing the Stubbed Service

Install Apache JMeter on your laptop (http://jmeter.apache.org/)

- Download and extract the latest binary. Note that you need Java 8 installed on your laptop.
- Run the .bat (Windows) or .sh (Linux) file from the bin folder

Load test your two API endpoints by creating a Test Plan with the following:

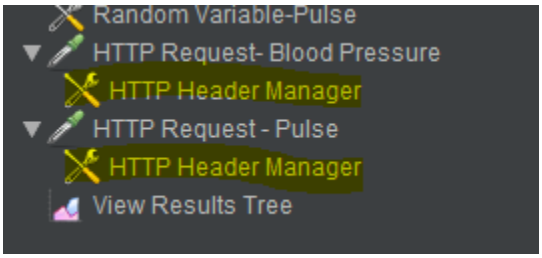
- Thread Group with X users and 10 loop count, where X is your expected peak load of concurrent events
- The Thread Group contains:
    - HTTP Request for your first API endpoint
    - HTTP Request for your second API endpoint
    - Listener -> View Results Tree

You can follow this tutorial as a reference:

- https://www.blazemeter.com/blog/rest-api-testing-how-to-do-it-right/

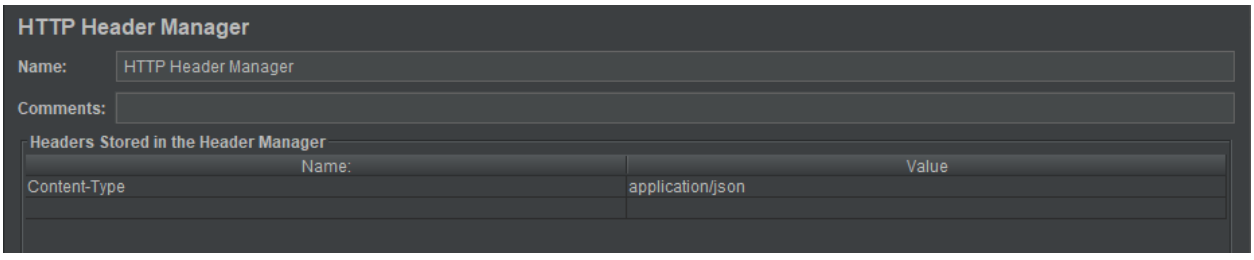
Make sure to set the headers for your HTTP Requests to application/json.

To set the Content-Type of the request body of your HTTP Request to be application/json, you need to add a HTTP Header Manager to your HTTP Request:



To add the HTTP Header Manager, right click on the HTTP Request and select Add -> Config Element -> HTTP Header Manager

You can then edit the HTTP Header Manager for that HTTP Request by adding a Content-Type with a value of application/json



When testing, make sure all your HTTP Requests are successful in the View Results Tree. You may have to adjust your X value (i.e., peak load) depending on the number of concurrent events your service can support.

Demonstration and Submission

Demonstrate the following to your instructor.

app.py file with two functions corresponding to your POST API endpoints that store the counts and the most recent requests for each event type to a JSON file as per the specifications above *You will only get the full 4 marks if all the requirements are met – correct logging, contents of events.log is valid JSON, using constants, etc.*	4 marks
The http://localhost:8080/ui shows the documentation for your two endpoints	2 marks
Valid requests are successfully received by end of your endpoints and invalid are rejected	2 marks
Load testing demo using Apache jMeter	2 marks
Total	**10 marks**

Upload your app.py and openapi.yaml files to the Lab 2 dropbox in the Learning Hub as your submission to receive the marks for your demo.