SERVICE ORIENTED ARCHITECTURES

ACIT3855 – FALL 2021

AGENDA

- Quick Review
- Quiz 1
- Microservices – Fowler
- Our Sample Application and First Service
  - Edge Service
  - Connexion
  - JSON, File I/O
  - Testing with PostMan and jMeter
- Lab 2
  - Edge Service

MARTIN FOWLER – SOFTWARE ARCHITECT AT THOUGHTWORKS

In short, the microservice architectural style is an approach to developing a single application as a **suite of small services**, each **running in its own process** and communicating with lightweight mechanisms, often an HTTP resource API. These services are **built around business capabilities** and **independently deployable** by fully automated deployment machinery. There is a **bare minimum of centralized management** of these services, which may be written in different programming languages and use different data storage technologies.

-- James Lewis and Martin Fowler (2014)

MARTIN FOWLER – SOFTWARE ARCHITECT AT THOUGHTWORKS

Microservices provide benefits…

- Strong Module Boundaries: Microservices reinforce modular structure, which is particularly important for larger teams.
- Independent Deployment: Simple services are easier to deploy, and since they are autonomous, are less likely to cause system failures when they go wrong.
- Technology Diversity:With microservices you can mix multiple languages, development frameworks and data-storage technologies.

MARTIN FOWLER – SOFTWARE ARCHITECT AT THOUGHTWORKS

…but come with costs

- Distribution: Distributed systems are harder to program, since remote calls are slow and are always at risk of failure.
- Eventual Consistency: Maintaining strong consistency is extremely difficult for a distributed system, which means everyone has to manage eventual consistency.
- Operational Complexity:You need a mature operations team to manage lots of services, which are being redeployed regularly.

OPENAPI AND CONNEXION

- Let’s review the sample OpenAPI Specification and Connexion Application from the reading

QUIZ 1

- Quiz is on the Learning Hub
- I will provide you with the password in the chat window on the Virtual Classroom
- You have ~15 minutes to complete it

COURSE SCHEDULE – TUESDAY SET

Week	Topics	Notes
1	- Services Based Architecture Overview - RESTful API Review	Lab 1
2	- Microservices Overview - Edge Service	Lab 2, Quiz 1
3	- Database Per Service - Storage Service (SQLite)	Lab 3, Quiz 2
4	- Logging, Debugging and Configuration - Storage Service (MySQL)	Lab 4, Quiz 3
5	- RESTful API Specification (OpenAPI) - Processing Service	Lab 5, Quiz 4, Assignment 1 Due
6	- Synchronous vs Asynchronous Communication - Message Broker Setup	Lab 6A, Quiz 5
7	- Messaging and Event Sourcing	Lab 6B, No Quiz
8	- Midterm Review - Containerization of Services (Docker/Docker Compose)	Lab 7, Quiz 6, Assignment 2 Due (Midterm Review)
9	- Dashboard UI and CORS	Lab 8, Quiz 7
10	- Issues and Technical Debt	Lab 9, No Quiz
11	- Deployment – Configuration and Logging	Lab 10, Quiz 8
12	- Deployment – Reverse Proxy and Load Balancing - Deployment – Scaling (RESTful APIs)	Lab 11, Quiz 9
13	- Final Exam Review - Assignment 3 In Class	Assignment 3 Due
14	- Final Exam	

COURSE SCHEDULE – THURSDAY SETS

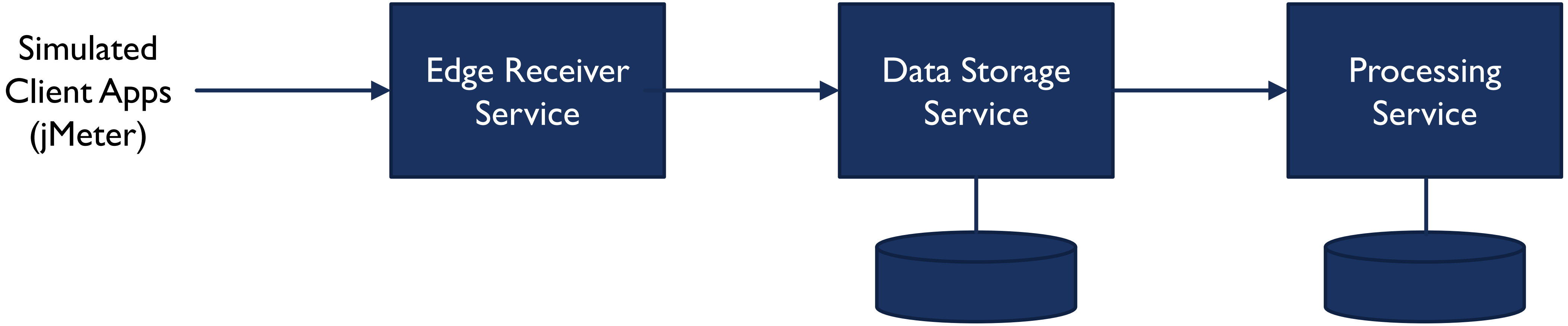
Week	Topics	Notes
1	• Services Based Architecture Overview • RESTful API Review	Lab 1
2	• Microservices Overview • Edge Service	Lab 2, Quiz 1
3	• Database Per Service • Storage Service (SQLite)	Lab 3, Quiz 2
4	**No Class - Holiday**	
5	• Logging, Debugging and Configuration • Storage Service (MySQL)	Lab 4, Quiz 3, Assignment 1 Due
6	• RESTful API Specification (OpenAPI) • Processing Service	Lab 5, Quiz 4
7	• Synchronous vs Asynchronous Communication • Message Broker Setup • Messaging and Event Sourcing	Lab 6, Quiz 5
8	• Midterm Review • Containerization of Services (Docker/Docker Compose)	Lab 7, Quiz 6, Assignment 2 Due (Midterm Review)
9	• Dashboard UI and CORS	Lab 8, Quiz 7
10	• **No Class – Holiday.** Issues and Technical Debt (Take Home)	Lab 9, No Quiz
11	• Deployment – Configuration and Logging	Lab 10, Quiz 8
12	• Deployment – Reverse Proxy and Load Balancing • Deployment – Scaling (RESTful APIs)	Lab 11, Quiz 9
13	• Final Exam Review • Assignment 3 In Class	Assignment 3 Due
14	• Final Exam	

OUR SAMPLE APPLICATION

Our sample application will have three initial services:

- Receiver Service (Lab 1 and 2)
- Storage Service (Lab 3)
- Processing Service (Lab 5)

We will also be adding logging and external configuration to our services starting in Lab 4



EDGE SERVICE

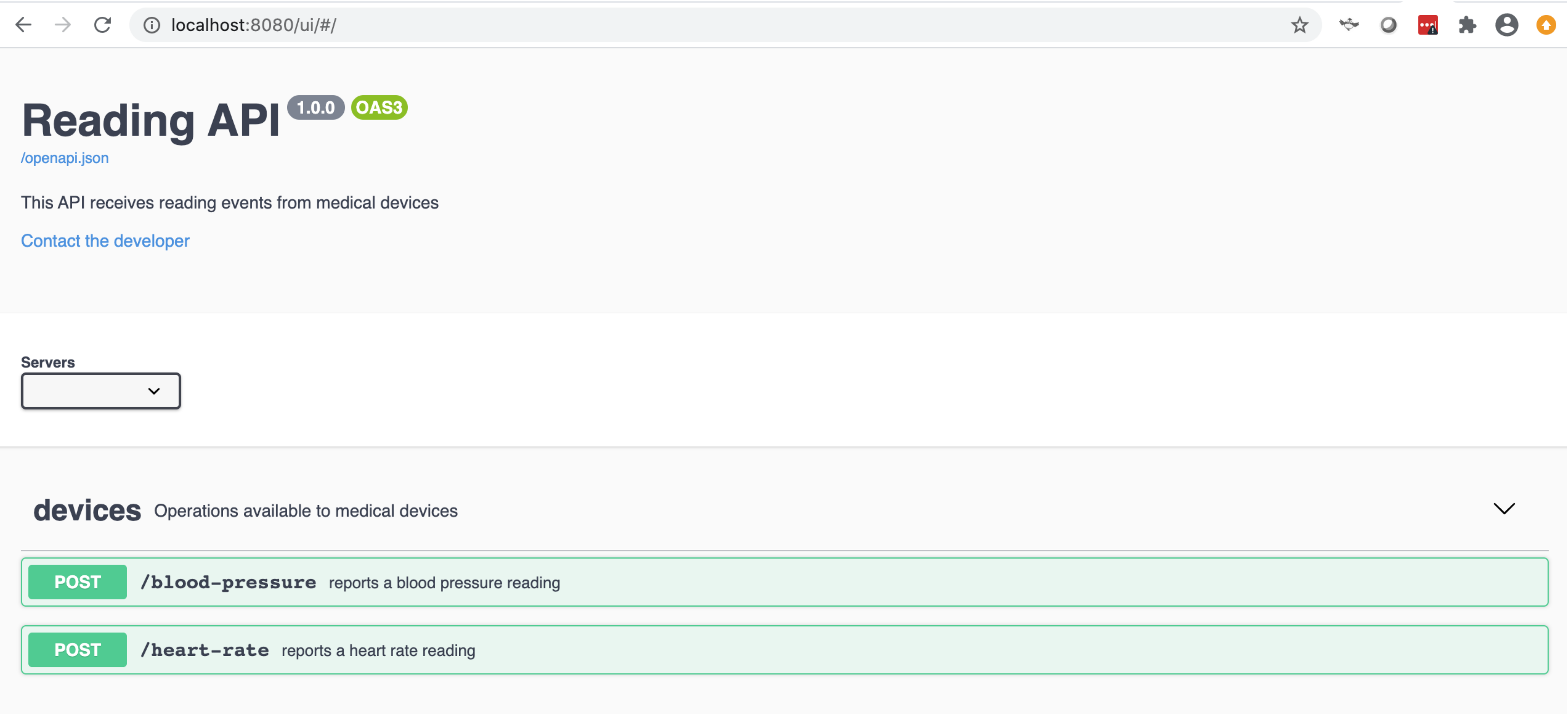
- An **Edge Service** is one which is exposed to the public internet.
- Typically it receives requests from external applications, and routes them to the correct internal service within out application.
- An Edge Service could be implemented using a web application server, such as Nginx or Apache, acting as a reverse proxy or an API gateway
  - Reverse Proxy – We will set one of these up in ACIT 4850. A web server setup as a single endpoint for multiple web applications.
  - API Gateway – Specialized service that authenticates and routes incoming requests, and can limit incoming traffic. Usually provided by an open-source or commercial product.

We are going to build our own simple Edge Service (the Receiver Service) that receives our two Events.

CONNEXION

- Connexion – A Python Framework
  - Built on top of Flask
  - Automatically handles HTTP requests defined in an OpenAPI specification (2.0 or 3.0)
  - https://connexion.readthedocs.io/en/latest/
- Your openapi.yml file defines your endpoints.
  - You add the openapi.yml file to your Connection application
  - You create a function for each endpoint with a name matching the operationId of the endpoint
  - Connexion automatically routes incoming requests to the correct function based on the operation id and passes in the request message as a parameter
  - JSON requests are automatically converted to Python objects (i.e., lists and/or dictionaries)

CONNEXION – UI DOCUMENTATION



localhost:8080/ui/#/

Reading API 1.0.0 OAS3

/openapi.json

This API receives reading events from medical devices

Contact the developer

Servers

devices Operations available to medical devices

POST `/blood-pressure` reports a blood pressure reading

POST `/heart-rate` reports a heart rate reading

REVIEW – PYTHON NAMING

- Remember our naming conventions in Python?
  - Functions and Variables?

lower_snake_case

<u>Examples:</u>
first_name, x, systolic_bp
get_response, report_bp_reading

  - Constant Values?

UPPER_CASE

<u>Examples:</u>
PI
NUM_READINGS

These are typically defined at the top of our Python script or module.

REVIEW – FILE I/O AND OS.PATH.ISFILE

Reading from a file in Python

file_handle = open(filename, “r”)

file_contents = file_handle.read()

file_handle.close()

Writing to a file in Python

file_handle = open(filename, “w”)

file_handle.write(data_to_write)

file_handle.close()

os.path.isfile

- You will get an exception if you try to read from a file that doesn’t exist
- import os.path
- os.path.isfile(filename) returns True if the file exists, False otherwise

REVIEW – JSON MODULE

- Python has a built-in json module
  - https://docs.python.org/3/library/json.html
- Serialization – Convert Python data to a JSON string
  - json.dumps
  - json_str = json.dumps(python_data)
- Deserialization – Convert a JSON string to Python data
  - json.loads
  - python_data = json.loads(json_str)

You will use this in your Lab today to “log” requests to a file

TESTING – POSTMAN AND APACHE JMETER

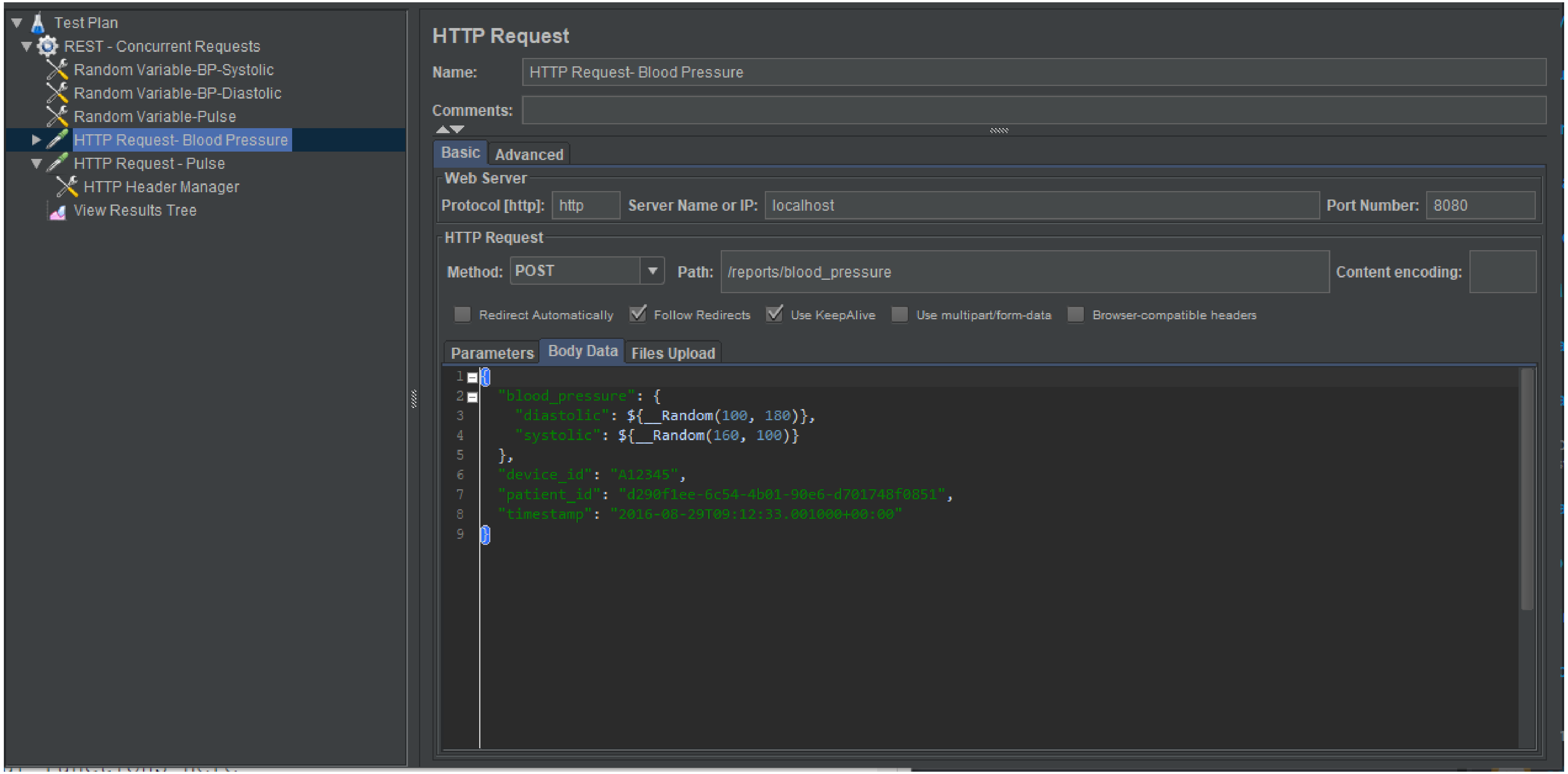
PostMan

- Should be familiar from ACIT 2515 and other classes
- Used to test RESTful API endpoints

Apache jMeter

- Java based tool that can be used to test the functionality and performance of web applications
- It allows us to create test scenarios as a series of HTTP requests
- We can have it apply through scenarios concurrently to simulate many users (i.e., high load on the system)

JMETER – EXAMPLE HTTP REQUEST



- Test Plan
  - REST - Concurrent Requests
    - Random Variable-BP-Systolic
    - Random Variable-BP-Diastolic
    - Random Variable-Pulse
    - HTTP Request- Blood Pressure
    - HTTP Request - Pulse
      - HTTP Header Manager
    - View Results Tree

HTTP Request

Name: HTTP Request- Blood Pressure

Comments:

Basic | Advanced

Web Server

Protocol [http]: http   Server Name or IP: localhost   Port Number: 8080

HTTP Request

Method: POST   Path: /reports/blood_pressure   Content encoding:

□ Redirect Automatically ☑ Follow Redirects ☑ Use KeepAlive □ Use multipart/form-data □ Browser-compatible headers

Parameters | Body Data | Files Upload

```
{
  "blood_pressure": {
    "diastolic": ${__Random(100, 180)},
    "systolic": ${__Random(160, 100)}
  },
  "device_id": "A12345",
  "patient_id": "d290f1ee-6c54-4b01-90e6-d701748f0851",
  "timestamp": "2016-08-29T09:12:33.001000+00:00"
}
```

TODAY'S TOOLS

RESTful API Specification: SwaggerHub and OpenAPI

- Define a RESTful API in a yaml format (**Done in Lab 1**)

RESTful API Implementation: Python Connexion

- Built on top of Flask but allows integration with an OpenAPI specification

RESTful API Testing: PostMan and Apache jMeter

- Postman – same as ACIT 2515
- Apache jMeter – for load testing

You will be using these in your Lab today.

We will go through an example together in a moment.

DEMO – JMETER AND EDGE RECEIVER SERVICE

- Lets look at a sample stubbed out Edge Service using Connection
  - How to see the RESTful API documentation
  - File I/O
  - JSON
- Then we'll test it using PostMan and jMeter

TODAY'S LAB

You will be creating your Receiver Service (i.e., Edge Service) today in Lab 1 based on your OpenAPI specification from last week.

- It will receive each of your two events
- It will write those events as json data to a file

You will be testing it out with PostMan and Apache jMeter

Next week you will be creating a Storage Service and integrating it with your Receiver Service