Microservices Architecture and Programming

Lab Practical and date – Practical 2, 7th August 2020

Name and Roll Number- Het Shah, 17BIT103

Practical Objective- Ballerina – Hands-on with language concepts

Steps Involved-

We explored the Ballerina Language and coded a RESTFUL API in python and implementing GET, PUT, POST and DELETE

Background

Ballerina is an open-source language developed by WSO2. It was designed specifically to replace configuration-based solutions to solve the initial problem of language difficulties. It is a compiled language, focusing largely on **microservice development** and integration, and most marketing material makes an effort to point out that Ballerina is “cloud-native”. Ballerina use HTTP/1.1 by default

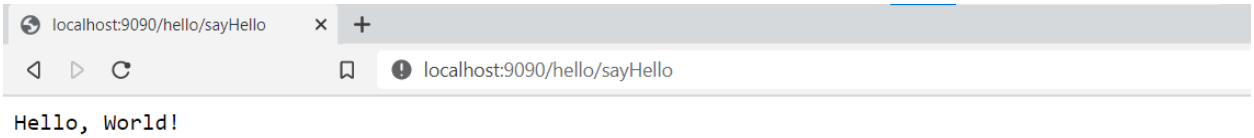
Code:

```
import ballerina/http;

service hello on new http:Listener(9090) {

    resource function sayHello(http:Caller caller,
        http:Request req) returns error? {

        check caller->respond("Hello, World!");
    }
}
```

Ballerina file is saved as .bal extension and to run the file we use the command ballerina run file_name.bal

```
E:\Desktop\sem 7\map>ballerina run hello_service.bal
Compiling source
        hello_service.bal
Running executables

[ballerina/http] started HTTP/WS listener 0.0.0.0:9090
```

The program compiles and then the server is started at 0.0.0.0:9090



The output of the program as seen at the web browser at the address http://localhost:9090/hello/sayHello

Running Ballerina Using Docker

Prerequisites:

1. Docker Desktop (Windows)
2. Docker compose (Linux)
3. Docker (Linux)
4. Ballerina (support for Mac, Windows and Linux distros)

Setup:

1. If using windows, any container can work i.e. Windows/ Linux, make sure that in Docker settings the setting “Expose daemon on tcp://localhost:2375 without TLS” is checked. This ensures the http is connectable.
2. Open any coding environment, preferable *VS code,* and type in the following code:

```
import ballerina/http;
import ballerina/log;
import ballerina/docker;   //importing docker for connection
@docker:Expose {}
listener http:Listener helloWorldEP = new(9090);
@docker:Config {
name: "helloworld",
tag: "v1.0"
}
@http:ServiceConfig {
basePath: "/helloWorld"
}
service helloWorld on helloWorldEP {
resource function sayHello(http:Caller outboundEP, http:Request request) {
http:Response response = new;
response.setTextPayload("Hello World from Docker ! \n");
var responseResult = outboundEP->respond(response);
if (responseResult is error) {
error err = responseResult;
log:printError("Error sending response", err);
}}}
```

Lines 6-9 provide us the config for docker container setup, mainly including name and version of the docker container.

Line 11 provides the base URL which will be connected

Line 13 onwards, we can define functions which act as URL extensions, for example, sayHello can be called in browser via the URL *"/helloWorld/sayHello".*

Line 16 has an inbuild response function which sets the message to be sent when the URL for sayHello is called via curl or browser.

3. Save the file with an extension *".bal"*; e.g. Helloworld.bal
4. While docker is running, open terminal and type "ballerina build <filename>.bal"
5. To run the image on port 9090, type in terminal "docker run -d -p 9090:9090 <name>:<tag>", where name and tag are same as in docker config on line 7-8.
6. After completion, to check if image is created, type "docker images"
7. If its running, go to the following page in browser if on Windows/Linux: http://localhost:9090/helloWorld/sayHello

   If on Linux, you can use curl too, type in "curl http://localhost:9090/helloWorld/sayHello"

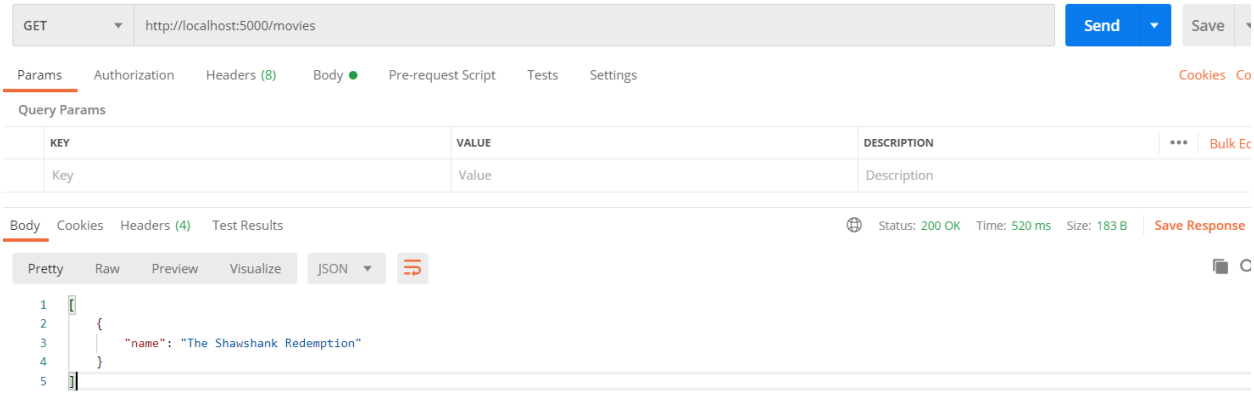
Known Issues:

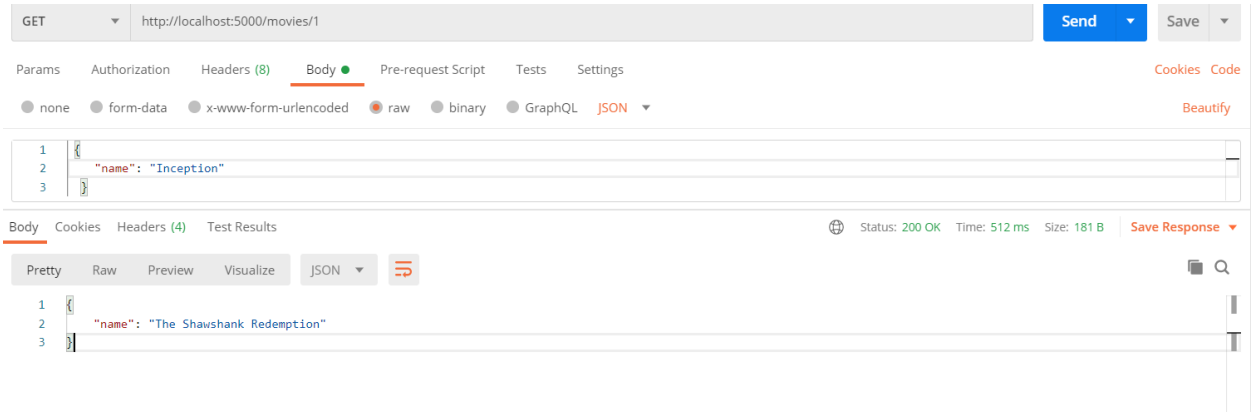
- While installing docker desktop, make sure virtualization is enabled in BIOS.
- While installing ballerina, give it access to network when it is asked, else connection to docker will fail.

RESTFUL API using Python

In this program we used python flask to make all the API calls. We are storing the data in the form of JSON and the data consists of names of movies. We use PostMan to make the API calls to the server and then get the result as well.

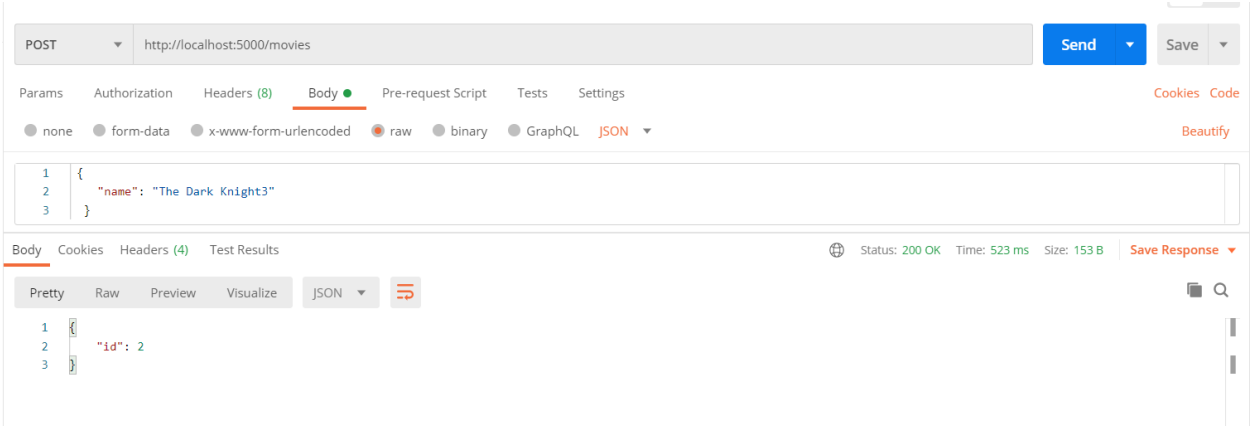
1) **GET** command by Default, shows all the entries

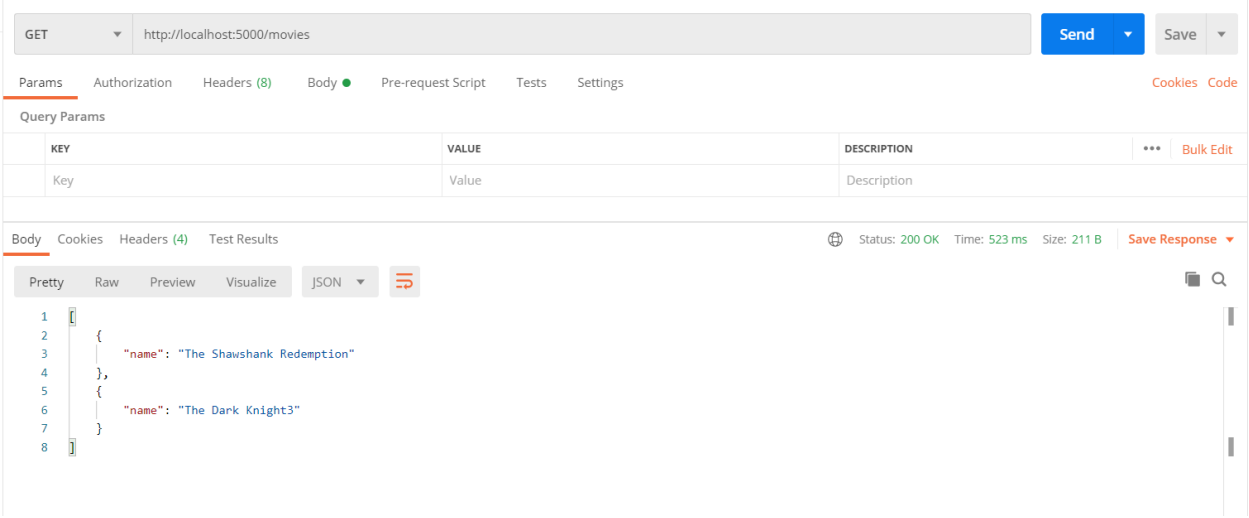




GET method to show the data at index i

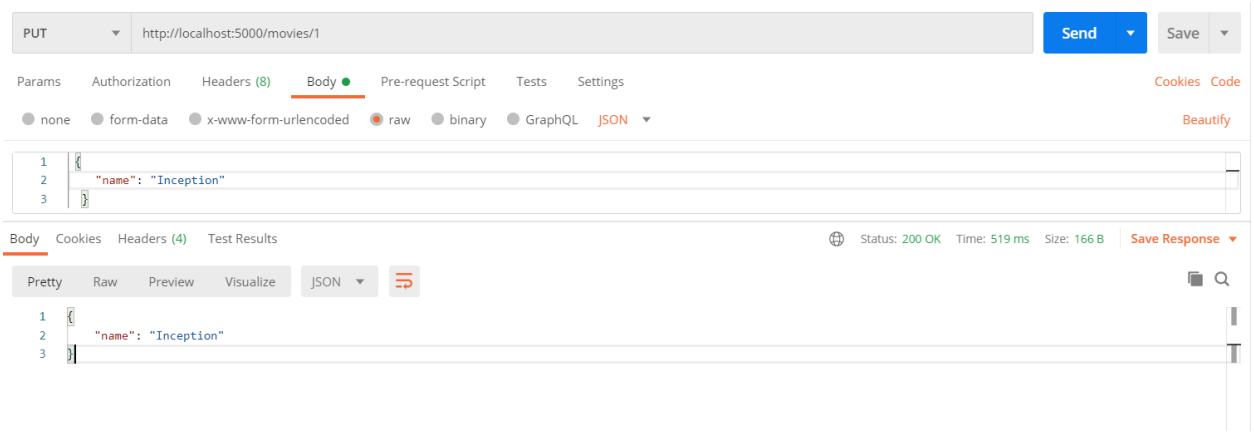
2) **POST** command- to Insert the values

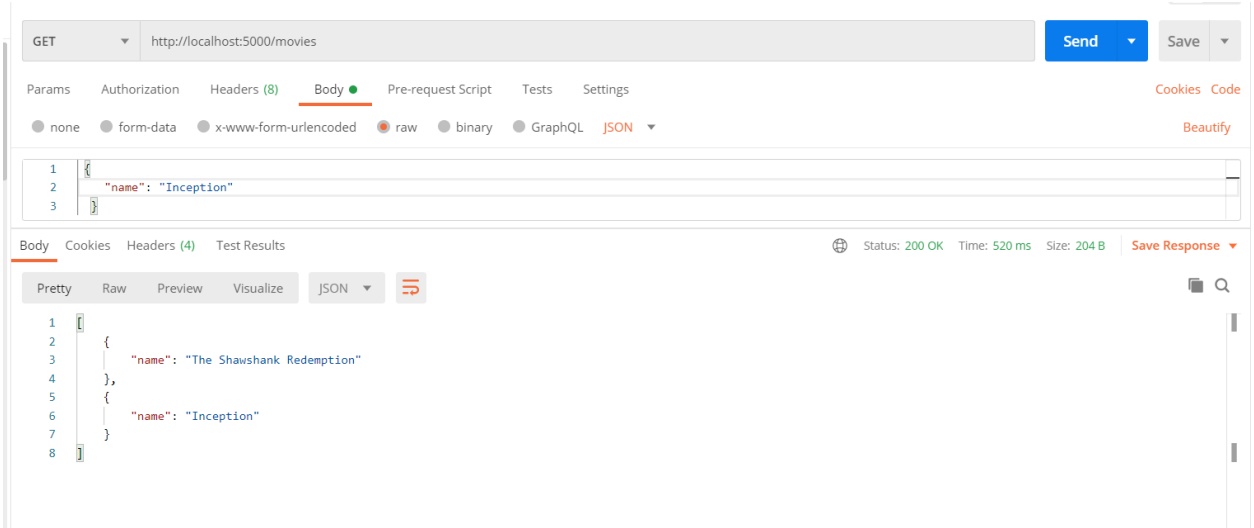




GET command verifying that the values have been inserted

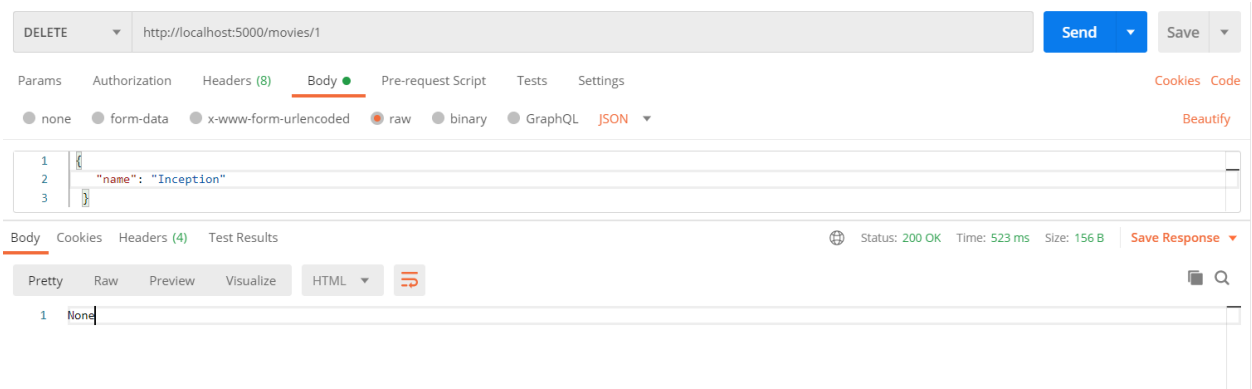
3) **PUT** command- to update the value of data at index i



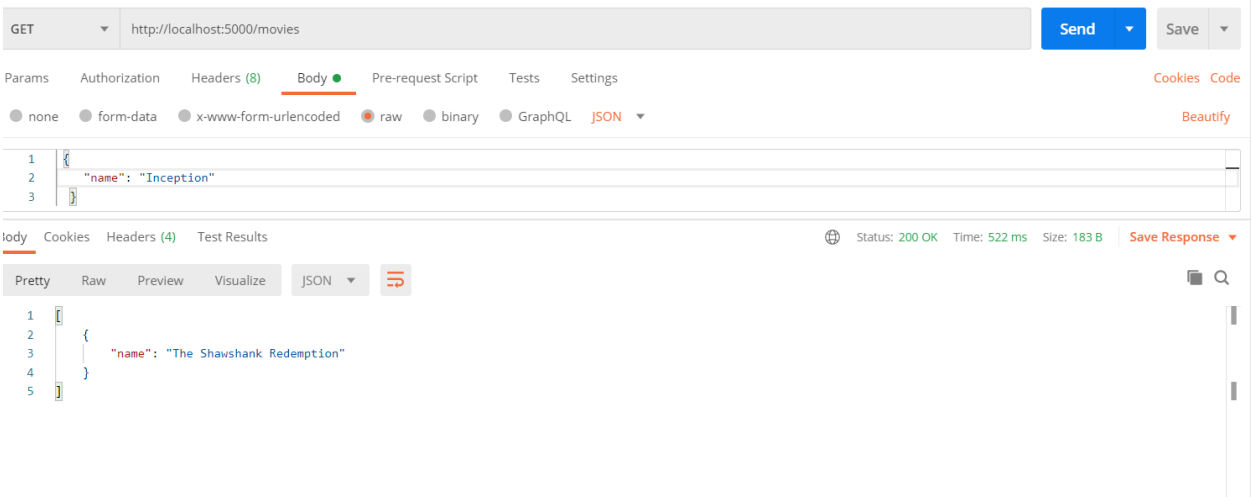


GET command verifying that data at index 2 has been updated

4) **DELETE**- to delete data at index i



It gives the response of None showing that the delete operation has been successful



GET method verifying that the data at index 1 has been deleted

Conclusion

In this practical we learned about the Ballerina language which is used for micro-services and learned how to deploy it on docker and windows as well. Later we implemented RESTFUL API using python flask and learned the behavior of GET, POST, PUT and DELETE

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Aug 20 23:06:48 2020

@author: HETSHAH
"""

from flask import Flask, jsonify, request

app = Flask(__name__)

movies = [
    {
        "name": "The Shawshank Redemption"

    }
]


@app.route('/movies')
def hello():
    return jsonify(movies)


@app.route('/movies/<int:index>', methods=['GET'])
def get1(index):
    movie = request.get_json()
    return jsonify(movies[index - 1]), 200


@app.route('/movies', methods=['POST'])
def add_movie():
    movie = request.get_json()
    movies.append(movie)
    return {'id': len(movies)}, 200


@app.route('/movies/<int:index>', methods=['PUT'])
def update_movie(index):
    movie = request.get_json()
    movies[index] = movie
    return jsonify(movies[index]), 200


@app.route('/movies/<int:index>', methods=['DELETE'])
def delete_movie(index):
    movies.pop(index)
    return 'None', 200


app.run()
```