Effective Testing with API Simulation and (Micro)Service Virtualisation

Module Four: Dynamic Responses

Purpose of this Lab

- Understanding of Hoverflies templating mechanism
- Create a simulation with dynamic responses

Questions

Feel free to ask questions and ask for help at any point during the workshop. Also, please collaborate with others.

Pre-requisites

Make sure we are in the correct directory!

```
$ cd ../4-dynamic-responses
```

Exercise 1: Populating a Response Based on Data in the Request

During this exercise, we will learn about and implement Hoverflies templating mechanism in order to dynamically build responses based on the request.

Steps

1. Make sure Hoverfly is running:

```
$ hoverctl start
target Hoverfly is already running
```

2. Make sure the flights API is running:

```
$ ./run-flights-service.sh
service started
```

3. Make sure Hoverfly is in capture mode:

```
$ hoverctl mode capture
Hoverfly has been set to capture mode
```

4. Make sure all the existing data is deleted from Hoverfly:

```
$ hoverctl delete
Are you sure you want to delete the current simulation? [y/n]: y
Simulation data has been deleted from Hoverfly

Hoverfly has been set to simulate mode with a matching strategy of
'strongest'
```

5. Now we want to produce another simulation of our flights request, only this time will add some extra query parameters to our request. ‘to’ is used to set the destination airport, and ‘from’ is used to set the origin airport.

Using record and replay, create a simulation of the following request:

```
$ curl 'localhost:8081/api/v1/flights?from=London&to=Paris'
```

Tip: Make sure you surround your URL with single quotes, otherwise the ampersand will be evaluated by the console.

6. Now, just like in the previous module, edit the simulation you have created to make the matcher looser. In this case we want it to support any value for ‘to’ and any value for ‘from’.

Tip: You can use ‘globMatcher’, ‘regexMatcher’ matchers or field omission to achieve this.

Put Hoverfly into simulate mode, and then validate that you have successfully produced a simulation by make some requests to it, each time with different query parameter values:

```
$ curl ‘localhost:8081/api/v1/flights?plusDays=1&from=London&to=Paris’
--proxy localhost:8500 | jq
…
curl ‘localhost:8081/api/v1/flights?plusDays=1&from=Milan&to=Tokyo’ --proxy
localhost:8500 | jq
```

```
...
curl 'localhost:8081/api/v1/flights?plusDays=1&from=Paris&to=Lisbon'
--proxy localhost:8500 | jq
```

Notice how the response remains static whilst the query parameters remain change. Would this behaviour make sense if it were the real API?

8. We can make the content of the response change based on these query parameters by using templating. What we would like is the 'to' and 'from' fields in the response JSON to always contain the same 'to' and 'from' that were provided in the query. Export the simulation and edit it to do this.

1. First of all, make sure 'data.pairs.response[0].templated' is set to true, otherwise templating will not be active.
2. Now replace the 'to' and 'from' in the response body with template variables. The following are available:

Field	Example	Request	Result
Request scheme	{{ Request.Scheme }}	http://www.foo.com	http
Query parameter value	{{ Request.QueryParam.myParam }}	http://www.foo.com?myParam=bar	bar
Query parameter value (list)	{{ Request.QueryParam.NameOfParameter.[1] }}	http://www.foo.com?myParam=bar1&myParam=bar2	bar2
Path parameter value	{{ Request.Path.[1] }}	http://www.foo.com/zero/one/two	one

9. Validate that your simulation is now dynamic. This can be achieved by importing the simulation into Hoverfly and then making some requests to it, each with different query parameter values.