Effective Testing with API Simulation and (Micro)Service Virtualisation

Module Four: Dynamic Responses

Purpose of this Lab

- Understanding of Hoverfly's 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
```

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.

Now, make a simulation of the following request:

```
Example Request:

$ curl 'localhost:8081/api/v1/flights?from=London&to=Paris?plusDays=2'
```

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

We are assuming you know how to make a simulation based on your work in the previous module. If you are unsure, then consult the tutorial in that module, or ask for help.

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: Just omit the query from the matcher altogether

Again, we are assuming you know how to make a looser matcher based on your work in the previous module. If you are unsure, then consult the tutorial in that module, or ask for help.

Validate that your simulation works by making some requests to it:

```
$ curl 'localhost:8081/api/v1/flights?plusDays=1&from=London&to=Paris'
--proxy localhost:8500 | jq
[
  {
    "origin": "Berlin",
    "destination": "Dubai",
    "cost": "3103.56",
    "when": "2006-01-02T15:04:05Z07:00"
  }
]
curl ‘localhost:8081/api/v1/flights?plusDays=1&from=Milan&to=Tokyo’ --proxy
localhost:8500 | jq
[
  {
    "origin": "Berlin",
    "destination": "Dubai",
    "cost": "3103.56",
    "when": "2006-01-02T15:04:05Z07:00"
  }
]

curl ‘localhost:8081/api/v1/flights?plusDays=1&from=Paris&to=Lisbon’
--proxy localhost:8500 | jq
[
  {
    "origin": "Berlin",
    "destination": "Dubai",
    "cost": "3103.56",
    "when": "2006-01-02T15:04:05Z07:00"
  }
]
```

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

7. 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. Edit the simulation to allow for 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 templating variables 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

8. 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:

```
$ curl 'localhost:8081/api/v1/flights?plusDays=1&from=London&to=Paris'
--proxy localhost:8500 | jq
[
  {
    "origin": "London",
    "destination": "Paris",
    "cost": "3103.56",
    "when": "2006-01-02T15:04:05Z07:00"
  }
]
curl ‘localhost:8081/api/v1/flights?plusDays=1&from=Milan&to=Tokyo’ --proxy
localhost:8500 | jq
[
  {
```

```
    "origin": "Milan",
    "destination": "Tokyo",
    "cost": "3103.56",
    "when": "2006-01-02T15:04:05Z07:00"
  }
]

curl ‘localhost:8081/api/v1/flights?plusDays=1&from=Paris&to=Lisbon’
--proxy localhost:8500 | jq
[
  {
    "origin": "Paris",
    "destination": "Lisbon",
    "cost": "3103.56",
    "when": "2006-01-02T15:04:05Z07:00"
  }
]
```

Once this exercise is finished, **export your simulation for use in the next exercise.**

Exercise 2: Using Helper Methods for Response Generation

During this exercise, we will learn how we can use templating helper methods with Hoverfly in order to dynamically generate data.

Steps

1. If we try out our simulation from the last exercise, we will notice that if the value of plusDays changes, the date remains the same. This does not make sense, because the real API would return a date on the relevant day:

```
$ curl 'localhost:8081/api/v1/flights?plusDays=1&from=London&to=Paris'
--proxy localhost:8500 | jq
[
  {
    "origin": "London",
    "destination": "Paris",
    "cost": "3103.56",
```

```
    "when": "2006-01-02T15:04:05Z07:00"
  }
]
$ curl 'localhost:8081/api/v1/flights?plusDays=5&from=London&to=Paris'
--proxy localhost:8500 | jq
[
  {
    "origin": "London",
    "destination": "Paris",
    "cost": "3103.56",
    "when": "2006-01-02T15:04:05Z07:00"
  }
]
```

2. Modify your simulation so the date will always be today's date.

There are now less step by step instructions, as we should now understand how to do all of of the boilerplate already. If not, consult the slides, the previous exercises, or ask!

Here are the available helper method which may be of use:

Description	Example	Result
Request scheme	{{ iso8601DateTime }}	2006-01-02T15:04:05Z07:00
Query parameter value	{{ iso8601DateTimePlusDays availablevariable }}	2006-03-02T15:04:05Z07:00

3. Now, modify your simulation so the output date is always the current day incremented by the value of days entered.

4. Validate your simulation by making some requests to it, and seeing if the date changes.

Advanced Exercise

In the previous module, we used scoring to create a generalised matcher for an endpoint, and more specific, higher-score matchers which can be used to override it. If we wanted to though, we could achieve the same result with templating.

We should now have a simulation of the following request:

```
Example Request:

$ curl http://localhost:8081/api/v1/flights?plusDays=0&from=London&to=Milan
```

The request should:

1. Alter the airports in the response based on the request.
2. Alter the departure time based on the day requested

Now, try modifying your template so if plusDays is missing, then the response is empty. **We do not need to create a new matcher.**

Tip: Documentation for conditional templating can be found here https://github.com/aymerick/raymond