Effective Testing with API Simulation and (Micro)Service Virtualisation

Module Three: Matching

Purpose of this Lab

- How to simulate multiple API requests using matching
- How to debug misses by looking at the closest miss
- How to use loose matching to simulate many requests with a single matcher
- How scoring can be used to give multiple matches an order of precedence

Questions

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

Pre-requisites

Make sure you have cloned the api-simulation-training Git repository, and that you are in the correct directory!

```
$ cd api-simulation-training/3-matching
```

Exercise 1: Simulating Multiple Requests with Hoverfly

During the first exercise, we will use Hoverfly to capture multiple requests and responses, each of which will become their own matcher->response within a simulation.

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, make the following request to the flights service via Hoverfly (note that your curled flight results may look different to the results shown below):

```
$ curl localhost:8081/api/v1/flights?plusDays=1 --proxy localhost:8500 | jq
[
  {
    "origin": "Berlin",
    "destination": "Dubai",
    "cost": "3103.56",
    "when": "20:53"
  },
  {
    "origin": "Amsterdam",
    "destination": "Boston",
    "cost": "2999.69",
    "when": "19:45"
  }
]
```

Tip: We are piping our output into jq in order to format the JSON in a way which is human readable

6. Now, repeat the same request three times, only each time increment the value of the query parameter named ‘plusDays’:

```
$ curl localhost:8081/api/v1/flights?plusDays=1 --proxy localhost:8500 | jq
…
$ curl localhost:8081/api/v1/flights?plusDays=2 --proxy localhost:8500 | jq
…
$ curl localhost:8081/api/v1/flights?plusDays=3 --proxy localhost:8500 | jq
…
```

7. Now, let’s take a look at the simulation that we have produced, by exporting it and then opening it in a text editor:

```
$ hoverctl export matching-exercise-one-simulation.json
Successfully exported simulation to matching-exercise-one-simulation.json
$ atom matching-exercise-one-simulation.json
```

Take a look at the simulation file, and see if you recognise your recorded data. Each request you captured should respond to an element in the pairs array. In other words, we have produced a separate matcher->response pair for each request.

8. Now we can use our simulation to simulate the flights API. First, stop the flights service to make sure we are unable to communicate with it:

```
$ ./stop-flights-service.sh
service successfully shut down
$ curl localhost:8081/api/v1/flights?plusDays=1
curl: (7) Failed to connect to localhost port 8081: Connection refused
```

9. Now, put Hoverfly into simulate mode:

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

10. Replay the requests we made earlier. Even though the flight service is not running, Hoverfly will impersonate it, replaying the requests and responses that we recorded. It will pick the response to use based on matching (i.e. the results returned when the plusDays parameter is set to 1 will be the exact data captured when the request was made with the same parameter value)

```
$ curl localhost:8081/api/v1/flights?plusDays=1 --proxy localhost:8500 | jq
…
$ curl localhost:8081/api/v1/flights?plusDays=2 --proxy localhost:8500 | jq
…
$ curl localhost:8081/api/v1/flights?plusDays=3 --proxy localhost:8500 | jq
…
```

Advanced One

Sometimes, the service we are developing might have an endpoint that has not yet been implemented. This is the case for our flights service, which will eventually be able to provide recommended destinations for a user based on their location. The endpoint will work as follows:

```
Example Request:

$ curl http://localhost:8081/api/v1/recommendations?location=acity

Example Response:

HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8

["London", "Paris"]
```

We have a service under test which will needs to call this endpoint, so in the meantime we can simulate it.

Manually edit your simulation directly, adding pairs which simulate the following requests:

1. A request which returns a list of cities
2. A request which returns no cities
3. A request which returns a 404 status code

Once this is done, you can validate your work by switching into simulate mode and curling your endpoints.

Once this exercise is finished, **do not move onto the next exercise.** Wait for the remainder of the presentation first, after which you will be prompted to move on.

Exercise 2: Simplifying our Simulation with Loose Matching

Now, we are going make use of looser matching in order to simplify our simulation. We will first look at some of the problems with our current simulation, and then try and think of a better solution that will make it more adaptable.

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 simulate mode:

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

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. Import your simulation from the previous exercise. If you did not finish the previous exercise, import the one provided for you in the answers directory:

```
$ hoverctl import answers/matching-exercise-one-simulation.json
Successfully imported simulation from
answers/matching-exercise-one-simulation.json
```

6. Using this imported data we should be able to simulate these three requests:

```
$ curl localhost:8081/api/v1/flights?plusDays=1 --proxy localhost:8500 | jq
…
$ curl localhost:8081/api/v1/flights?plusDays=2 --proxy localhost:8500 | jq
…
$ curl localhost:8081/api/v1/flights?plusDays=3 --proxy localhost:8500 | jq
…
```

We will be unable to simulate any other similar requests with different values for 'plusDays'. Try making the following request, and take a look at the response body to see what happens (**note that we are not piping the curl output to jq in this example!)**:

```
$ curl localhost:8081/api/v1/flights?plusDays=6 --proxy localhost:8500
Hoverfly Error!

There was an error when matching

Got error: Could not find a match for request, create or record a valid
matcher first!

The following request was made, but was not matched by Hoverfly:

{
    "Path": "/api/v1/flights",
    "Method": "GET",
    "Destination": "localhost:8081",
    "Scheme": "http",
    "Query": {
        "plusDays": [
            "7"
        ]
    },
    "Body": "",
    "Headers": {
        "Accept": [
            "*/*"
        ],
        "Proxy-Connection": [
            "Keep-Alive"
        ],
        "User-Agent": [
            "curl/7.54.0"
        ]
    }
}

Whilst Hoverfly has the following state:

{}

The matcher which came closest was:

{
    "path": {
        "exactMatch": "/api/v1/flights"
    },
    "method": {
        "exactMatch": "GET"
    },
    "destination": {
        "exactMatch": "localhost:8081"
    },
    "scheme": {
        "exactMatch": "http"
    },
    "query": {
        "exactMatch": "plusDays=4"
    },
    "body": {}
}

But it did not match on the following fields:

[query]
```

```
Which if hit would have given the following response:

{
    "status": 200,
    "body":
"[{\"origin\":\"Milan\",\"destination\":\"Tokyo\",\"cost\":\"641.57\",\"whe
n\":\"14:37\"}]",
    "encodedBody": false,
    "headers": {
        "Content-Type": [
            "application/json;charset=UTF-8"
        ],
        "Date": [
            "Wed, 27 Sep 2017 12:33:00 GMT"
        ],
        "Hoverfly": [
            "Was-Here"
        ]
    },
    "templated": false
}
```

Do you understand the message you are receiving? It is the closest match message, and is especially useful for debugging. **Discuss or ask if you are struggling**.

7. In some situations we can't anticipate all the possible requests that might be made to an API simulation. This is why we need to think about looser matching, enabling us to support any possible request. Open the simulation in your text editor, and see if you can simplify it so it contains a single matcher which supports a 'plusDays' value of 0..n

```
$ atom . answers/matching-exercise-one-simulation.json
```

Tip 1: You can just delete all the matchers apart from the first one.

Tip 2: Although there are many ways to do this, omitting a field from the matcher would be simplest. This can be done by literally deleting the field altogether from the simulation JSON. Which field do you think we need to delete?

8. Once you think you have fixed the problem, import the simulation into Hoverfly and try different combinations of queries to see if they work:

```
$ hoverctl import answers/matching-exercise-one-simulation.json
Successfully imported simulation from
answers/matching-exercise-one-simulation.json
$ curl localhost:8081/api/v1/flights?plusDays=32 --proxy localhost:8500 |
jq
…
$ curl localhost:8081/api/v1/flights?plusDays=1000 --proxy localhost:8500 |
jq
…
$ curl localhost:8081/api/v1/flights?plusDays=21 --proxy localhost:8500 |
jq
…
```

If all is well, you should always get a list of flights back.

9. If we use field omission to ignore the query altogether, our matcher doesn't require 'plusDays' to be present. Can you think of an alternative way of matching which will require 'plusDays' to be present, but with any possible value?

Tip 1: Take a look at regex matching or glob matching. The documentation for this can be found on http://docs.hoverfly.io in the matching section.

Advanced Two

Sometimes, the service we are developing might have an endpoint that has not yet been implemented. This is the case for our flights service, which will eventually be able to provide an endpoint where we can make a booking. The endpoint will work as follows:

```
Example Request:

$ curl -H 'Content-Type: application/json' -X POST -d
'{"flightID":"KL2345", "cost":"100"}' localhost:8081/api/v1/bookings

Example Response:
```

```
HTTP/1.1 201 CREATED
Content-Type: application/json;charset=UTF-8
```

We have a service under test which will eventually call this endpoint, so in the meantime we can simulate it.

Manually edit your simulation directly, adding a two request-response pairs to simulate the endpoint. **The pair must match on the JSON**, but should also:

1. Not care about whitespacing
2. Not care about the ordering of fields

Once this is done, you can validate your work by switching into simulate mode and curling your endpoint, re-ordering the JSON and adding whitespacing.

Tip 1: The documentation for matchers can be found on http://docs.hoverfly.io in the matching section.

Once this exercise is finished, **do not move onto the next exercise.** Wait for the remainder of the presentation first, after which you will be prompted to move on.

Exercise 3: Making our Simulation Adaptable with Scoring

Although our simulation is simpler, we now have a problem of the same response always being returned regardless of the value of 'plusDays'. What if we wanted `plusDays=0` to always return no flights, and `plusDays=n` to always return the same flights? In this exercise we will use scoring in order to achieve this.

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
```

4. Make sure Hoverfly is in simulate mode:

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

5. Open our simulation from the previous example. If you have not completed the exercise then just open the one in the answers directory:

```
$ atom . answers/matching-exercise-two-simulation.json
```

6. Currently, the existing matcher matches on path, method, destination, scheme and body. This means all matches will be given a score of “5”.

Now we want to simulate other requests to the same endpoint:

1. Return an empty list if ‘plusDays=0’
2. Return a 400 response if ‘plusDays=-1’

Trying modify the simulation so it also simulates these requests. As some requests will match on both the original “weak” matcher, and the new ones, we need to be sure that the new ones take precedence. This can be done by making sure they have a score > 5, which they will do if they match on more fields.

Tip 1: You can just copy and paste the existing matcher and modify it.

Tip 2: For no flights to be returned, we need to modify the response body.

7. Once you think you have come up with a solution, import the simulation into Hoverfly and try different combinations of queries to see if they work:

```
$ hoverctl import answers/matching-exercise-two-simulation.json
Successfully imported simulation from
answers/matching-exercise-two-simulation.json

$ curl localhost:8081/api/v1/flights?plusDays=1 --proxy localhost:8500 | jq
[
  {
    "origin": "Berlin",
    "destination": "Dubai",
    "cost": "3103.56",
    "when": "20:53"
  },
  {
    "origin": "Amsterdam",
    "destination": "Boston",
    "cost": "2999.69",
    "when": "19:45"
  }
]
$ curl localhost:8081/api/v1/flights?plusDays=0 --proxy localhost:8500 | jq
[]
```

9. Make sure you understand how scoring is used to achieve this behaviour, and if you don't feel free to ask any questions.

Advanced Three

Sometimes we can end up with the same score for matchers, but we still want one to take precedence over another. Let's say we are creating a simulation of a yet to be implemented endpoint:

```
Example Request:

$ curl http://localhost:8081/api/v1/recommendations?location=somecity

Example Response:

HTTP/1.1 200 CREATED
Content-Type: application/json;charset=UTF-8


{
    "city" : "London"
}
```

Let’s say we have two matchers, :

```
"request": {
    "path": {
        "exactMatch": "/recommendations"
    },
    "method": {
        "exactMatch": "GET"
    },
    "destination": {
        "exactMatch": "localhost:8081"
    },
    "query": {
        "regexMatch": "location=(.*)"
    },
    "body": {
        "exactMatch": ""
    }
}
```

```
"request": {
    "path": {
        "exactMatch": "/recommendations"
    },
    "method": {
        "exactMatch": "GET"
    },
    "destination": {
        "exactMatch": "localhost:8081"
    },
    "query": {
        "exactMatch": "location=foo"
    },
    "body": {
        "exactMatch": ""
    }
}
```

If we were to make the following request:

```
$ curl http://localhost:8081/api/v1/recommendations?location=foo
```

Then both matchers would match as the score is the same.

Now try and simulate the endpoint:

1. By default we should return “London”
2. If “location=missing” we should return a 404
3. When “location=missing” is matched, then the “location=missing” matcher will need to have the highest score. Right now it has the same score.