Effective Testing with API Simulation and (Micro)Service Virtualisation

Module Five: Stateful Simulations

Purpose of this Lab

- Understanding and orchestration of Hoverflies state-store
- Building matchers which depend on state
- Building responses which mutate state

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 api-simulation-training/5-stateful-simulation
```

Exercise 1: Creating a Matcher Which Requires State

During this exercise, we will learn about Hoverflies state store, and how we can use it with matchers in order to simulate state.

Steps

1. Make sure Hoverfly is running:

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

2. Make sure the shopping API is running:

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

3. 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
```

4. Now, we will be simulating a shopping basket API. Using Hoverfly and curl, create a simulation of the following API requests (No need for loose matching):

```
$ curl localhost:8081/api/v1/shopping-basket
[]
$ curl -H 'Content-Type: application/json' -X PUT -d '{"item":"bacon"}'
localhost:8081/api/v1/shopping-basket
```

We have one request which gets the content of the shopping basket, and one request which adds bacon to the shopping basket.

5. Put Hoverfly into simulate mode, and try the following requests in order:

1. Get the contents of the shopping basket.
2. Add bacon to the shopping basket.
3. Get the contents of the shopping basket.

Can you see something which is unrealistic about the responses you are getting? Feel free to discuss with others.

6. Now, let's look at Hoverflies state-store. First, let's run help and then look at all the available commands:

```
$ hoverctl state-store --help
...
```

And then, let's look at what is currently stored in the state store (It should be empty):

```
$ hoverctl state-store get-all
The state for Hoverfly is empty
```

7. Let's see how we can modify the state store. First we can set a state key and value:

```
$ hoverctl state-store set basket bacon
Successfully set state key and value:
"basket"="bacon"
$ hoverctl state-store get-all
State of Hoverfly:
"basket"="bacon"
```

And second of all we can delete the contents of the state store:

```
$ hoverctl state-store delete-all
State has been deleted
$ hoverctl state-store get-all
The state for Hoverfly is empty
```

8. Now let's take a look at how we can match on state. This can be done by adding a "requiresState" field with a key and value:

```
"request": {
    "requiresState": {
        "basket" : "bacon"
    }
},
"response": {
...
}
```

Now we can see how it is done, let's try editing the simulation so there are two pairs for retrieving items from a shopping basket. One can return an empty basket and will not require any state, and one can return a basket containing bacon and requires the state "bacon=true"

Tip: We can just copy and paste the existing shopping basket pair, add the state requirement to one of them, and modify its body to return bacon.

9. Now we can import the simulation and validate that the stateful matching is working by deleting the state-store and making a request:

```
$ hoverctl state-store delete-all
```

```
State has been deleted
$ curl http://localhost:8081/api/v1/shopping-basket --proxy localhost:8500
[]
```

And then setting the state and making the same request:

```
$ hoverctl state-store set basket bacon
Successfully set state key and value:
"basket"="bacon"
$ curl http://localhost:8081/api/v1/shopping-basket --proxy localhost:8500
```

Exercise 2: Modifying State on a Match

In the previous exercise, we modified the state-store directly and demonstrated how we can match on different states. In this example, we will keep that approach, but learn how we can use a simulation to modify the state-store for us. This is a more realistic approach, as just like with the real API, state will be modified by certain requests.

Steps

1. Make sure Hoverfly is running:

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

2. Make sure the shopping API is running:

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

3. 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'
```

4. 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/stateful-exercise-one-simulation.json
Successfully imported simulation from
answers/stateful-exercise-one-simulation.json
```

5. Now, let's take a look at how we can transition state when we receive a match:

```
"request": {
    // fields
},
"response": {
    "status": 200,
    "body": "eggs and large bacon",
    "transitionsState" : {
        "payment-flow" : "complete",
    }
}
```

The above example would have exactly the same effect as using the state-store to set "payment-flow=complete" using the state store.

6. Now try and modify your simulation, so when bacon is added to the basket, state is changed so retrieving the basket would return one containing bacon.

7. Once finished and imported, let's try the simulation out to validate that it works as expected:

```
$ ./stop-shopping-service.sh
service successfully shut down
$ hoverctl state-store delete-all
State has been deleted
$ curl localhost:8081/api/v1/shopping-basket --proxy localhost:8500
[]
$ curl -H 'Content-Type: application/json' -X PUT -d '{"item":"bacon"}'
localhost:8081/api/v1/shopping-basket --proxy localhost:8500
$ curl localhost:8081/api/v1/shopping-basket --proxy localhost:8500
[]
$ localhost:8081/api/v1/shopping-basket
[{"item":"bacon"}]
$ hoverctl state-store get-all
State of Hoverfly:
"basket"="bacon"
```

Exercise 3: Deleting State on a Match

In the previous exercise, we created state transitions on matching. In this one, we will learn how to delete state altogether on a match.

Steps

1. Make sure Hoverfly is running:

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

2. Make sure the shopping API is running:

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

3. 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
```

4. 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/stateful-exercise-two-simulation.json
Successfully imported simulation from
answers/stateful-exercise-two-simulation.json
```

5. Now lets try out a new API request. This one can be used to delete a shopping basket:

```
$ curl -X DELETE localhost:8081/api/v1/shopping-basket
```

Notice how when making this request, the state of the shopping basket should become empty.

Add this new delete request to your simulation.

6. Now, let's take a look at how we can delete state altogether we receive a match:

```
"request": {
    // fields
},
```

```
"response": {
    "status": 200,
    "body": "eggs and large bacon",
    "removesState" : [
        "payment-flow"
    ]
}
```

7. Now, using the above strategy, modify the simulation so when we hit the delete endpoint, the shopping basket becomes empty.

8. Once finished and imported, let's try the simulation out to validate that it works as expected:

```
$ ./stop-shopping-service.sh
service successfully shut down
$ hoverctl state-store delete-all
State has been deleted
$ curl localhost:8081/api/v1/shopping-basket --proxy localhost:8500
[]
$ curl -H 'Content-Type: application/json' -X PUT -d '{"item":"bacon"}'
localhost:8081/api/v1/shopping-basket --proxy localhost:8500
$ curl localhost:8081/api/v1/shopping-basket --proxy localhost:8500
[]
$ localhost:8081/api/v1/shopping-basket
[{"item":"bacon"}]
$ curl -X DELETE localhost:8081/api/v1/shopping-basket
$ curl localhost:8081/api/v1/shopping-basket --proxy localhost:8500
[]
```