

Lab: Mock APIs

Web APIs are usually implemented as HTTP endpoints. Playwright provides APIs to mock and modify network traffic, both HTTP and HTTPS. Any requests that a page does, including XHRs and fetch requests, can be tracked, modified and mocked. With Playwright you can also mock using HAR files that contain multiple network requests made by the page.

Mock API requests

The following code will intercept all the calls to `*/**/api/v1/fruits` and will return a custom response instead. No requests to the API will be made. The test goes to the URL that uses the mocked route and asserts that mock data is present on the page.

Create/Edit a file with following code: `test_example.py`

```
import pytest
from playwright.sync_api import Page, Route, expect

def test_mock_the_fruit_api(page: Page):
    def handle(route: Route):
        json = [{"name": "Strawberry", "id": 21}]
        # fulfill the route with the mock data
        route.fulfill(json=json)

    # Intercept the route to the fruit API
    page.route("*/**/api/v1/fruits", handle)

    # Go to the page
    page.goto("https://demo.playwright.dev/api-mocking")

    # Assert that the Strawberry fruit is visible
    expect(page.get_by_text("Strawberry")).to_be_visible()
```

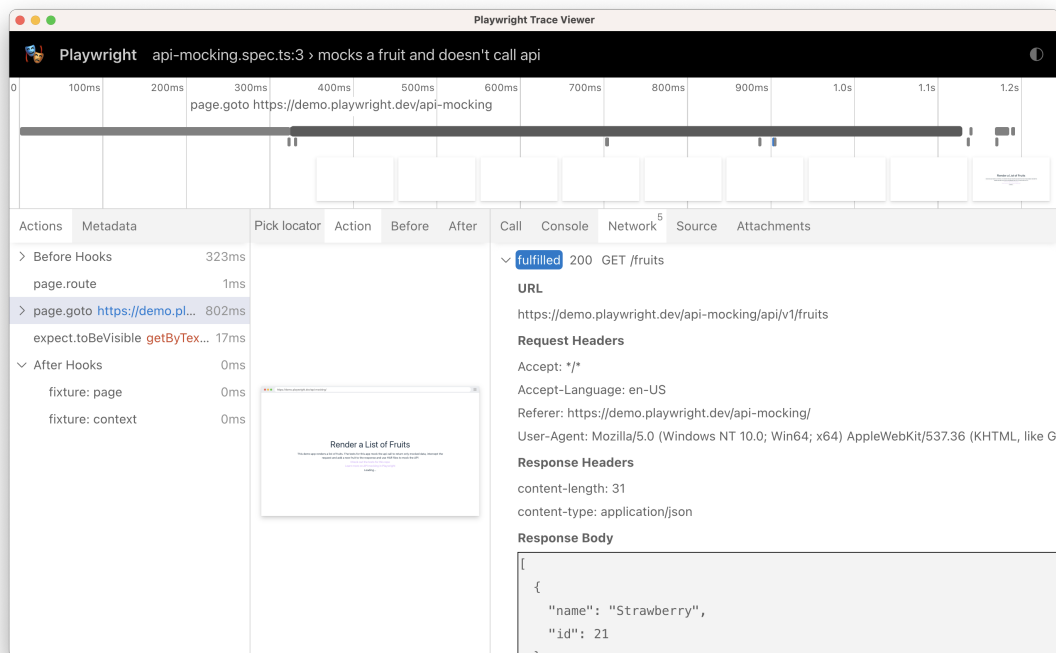
Traces can be recorded by running your tests with the `--tracing` flag.

Note: Delete the `test-results` folder before running below commands.

```
pytest --tracing on
```

```
playwright show-trace UPDATE_PATH\trace.zip
```

You can see from the trace of the example test that the API was never called, it was however fulfilled with the mock data.



Modify API responses

Sometimes, it is essential to make an API request, but the response needs to be patched to allow for reproducible testing. In that case, instead of mocking the request, one can perform the request and fulfill it with the modified response.

In the example below we intercept the call to the fruit API and add a new fruit called 'playwright', to the data. We then go to the url and assert that this data is there:

Create/Edit a file with following code: `test_example.py`

```
import pytest
from playwright.sync_api import Page, Route, expect

def test_gets_the_json_from_api_and_adds_a_new_fruit(page: Page):
    def handle(route: Route):
        response = route.fetch()
        json = response.json()
        json.append({ "name": "Loquat", "id": 100})
        # Fulfill using the original response, while patching the response body
        # with the given JSON object.
        route.fulfill(response=response, json=json)

    page.route("https://demo.playwright.dev/api-mocking/api/v1/fruits", handle)

    # Go to the page
    page.goto("https://demo.playwright.dev/api-mocking")
```

```
# Assert that the new fruit is visible
expect(page.get_by_text("Loquat", exact=True)).to_be_visible()
```

Traces can be recorded by running your tests with the `--tracing` flag.

Note: Delete the `test-results` folder before running below commands.

```
pytest --tracing on
```

```
playwright show-trace UPDATE_PATH\trace.zip
```

In the trace of our test we can see that the API was called and the response was modified.

The screenshot displays the Playwright Trace Viewer interface. At the top, the title bar reads "Playwright Trace Viewer". Below it, the test name is "api-mocking.spec.ts:23 > gets the json from api and adds a new fruit". The main area shows a timeline of events from 0 to 1.8 seconds. A key event is "page.goto https://demo.playwright.dev/api-mocking" at approximately 600ms. Below the timeline, the "Actions" tab is selected, showing a list of actions with their durations: "Before Hooks" (643ms), "page.route" (2ms), "page.goto https://demo.pl..." (643ms), "expect.toBeVisible getByT..." (365ms), "apiResponse.json" (0ms), "route.fulfill" (1ms), and "After Hooks" (0ms). A small preview of the application is shown next to the "page.goto" action. The "Network" tab is also visible, showing a list of network requests. The selected request is a "200 GET /fruits" request, which is marked as "fulfilled". The "Request Headers" and "Response Headers" are displayed for this request.

Actions	Metadata
> Before Hooks	643ms
page.route	2ms
> page.goto https://demo.pl...	643ms
expect.toBeVisible getByT...	365ms
apiResponse.json	0ms
route.fulfill	1ms
> After Hooks	0ms
fixture: page	0ms
fixture: context	0ms

Network

Call	Console	Network	Source	Attachments
> 301 GET /api-mocking text/html				
> 200 GET / text/html				
> 200 GET /index-6d10c910.css text/css				
> 200 GET /index-a4aa9aeb.js application/javascript				
> [api] 200 GET /fruits application/octet-stream				
✓ fulfilled 200 GET /fruits				

URL
https://demo.playwright.dev/api-mocking/api/v1/fruits

Request Headers
Accept: */*
Accept-Language: en-US
Referer: https://demo.playwright.dev/api-mocking/
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like...)

Response Headers
x-fastly-request-id: 199ff00e90d60794b2a21a49b24a20c29d4daac9
date: Fri, 23 Jun 2023 13:36:21 GMT

By inspecting the response we can see that our new fruit was added to the list.

