Network

Introduction

Playwright provides APIs to **monitor** and **modify** browser network traffic, both HTTP and HTTPS. Any requests that a page does, including XHRs and fetch requests, can be tracked, modified and handled.

Mock APIs

Check out our API mocking guide to learn more on how to

- mock API requests and never hit the API
- perform the API request and modify the response
- use HAR files to mock network requests.

HTTP Authentication

Perform HTTP Authentication.

Sync **Async**

```
context = browser.new_context(
    http_credentials={"username": "bill", "password": "pa55w0rd"}
)
page = context.new_page()
page.goto("https://example.com")
```

HTTP Proxy

You can configure pages to load over the HTTP(S) proxy or SOCKSv5. Proxy can be either set globally for the entire browser, or for each browser context individually.

You can optionally specify username and password for HTTP(S) proxy, you can also specify hosts to bypass proxy for.

Here is an example of a global proxy:

Sync **Async**

```
browser = chromium.launch(proxy={
  "server": "http://myproxy.com:3128",
  "username": "usr",
  "password": "pwd"
})
```

When specifying proxy for each context individually, **Chromium on Windows** needs a hint that proxy will be set. This is done via passing a non-empty proxy server to the browser itself. Here is an example of a context-specific proxy:

Sync **Async**

```
# Browser proxy option is required for Chromium on Windows.
browser = chromium.launch(proxy={"server": "per-context"})
context = browser.new_context(proxy={"server": "http://myproxy.com:3128"})
```

Network events

You can monitor all the Requests and Responses:

Sync **Async**

```
from playwright.sync_api import sync_playwright, Playwright

def run(playwright: Playwright):
    chromium = playwright.chromium
    browser = chromium.launch()
    page = browser.new_page()
    # Subscribe to "request" and "response" events.
    page.on("request", lambda request: print(">>", request.method, request.url))
```

```
    page.on("response", lambda response: print("<<", response.status,
response.url))
    page.goto("https://example.com")
    browser.close()

with sync_playwright() as playwright:
    run(playwright)
```

Or wait for a network response after the button click with page.expect_response():

Sync **Async**

```
# Use a glob url pattern
with page.expect_response("**/api/fetch_data") as response_info:
    page.get_by_text("Update").click()
response = response_info.value
```

Variations

Wait for Responses with page.expect_response()

Sync **Async**

```
# Use a regular expression
with page.expect_response(re.compile(r"\.jpeg$")) as response_info:
    page.get_by_text("Update").click()
response = response_info.value

# Use a predicate taking a response object
with page.expect_response(lambda response: token in response.url) as
response_info:
    page.get_by_text("Update").click()
response = response_info.value
```

Handle requests

Sync **Async**

```
page.route(
    "**/api/fetch_data",
    lambda route: route.fulfill(status=200, body=test_data))
page.goto("https://example.com")
```

You can mock API endpoints via handling the network requests in your Playwright script.

Variations

Set up route on the entire browser context with browser_context.route() or page with page.route(). It will apply to popup windows and opened links.

Sync Async

```
context.route(
    "**/api/login",
    lambda route: route.fulfill(status=200, body="accept"))
page.goto("https://example.com")
```

Modify requests

Sync Async

```
# Delete header
def handle_route(route):
    headers = route.request.headers
    del headers["x-secret"]
    route.continue_(headers=headers)
page.route("**/*", handle_route)

# Continue requests as POST.
page.route("**/*", lambda route: route.continue_(method="POST"))
```

You can continue requests with modifications. Example above removes an HTTP header from the outgoing requests.

Abort requests

You can abort requests using page.route() and route.abort().

Sync Async

```
page.route("**/*.{png,jpg,jpeg}", lambda route: route.abort())

# Abort based on the request type
page.route("**/*", lambda route: route.abort() if route.request.resource_type ==
"image"  else route.continue_())
```

Modify responses

To modify a response use APIRequestContext to get the original response and then pass the response to route.fulfill(). You can override individual fields on the response via options:

Sync Async

```
def handle_route(route: Route) -> None:
    # Fetch original response.
    response = route.fetch()
    # Add a prefix to the title.
    body = response.text()
    body = body.replace("<title>", "<title>My prefix:")
    route.fulfill(
        # Pass all fields from the response.
        response=response,
        # Override response body.
        body=body,
        # Force content type to be html.
        headers={**response.headers, "content-type": "text/html"},
    )

page.route("**/title.html", handle_route)
```

WebSockets

Playwright supports WebSockets inspection out of the box. Every time a WebSocket is created, the page.on("websocket") event is fired. This event contains the WebSocket instance for further web socket frames inspection:

```
def on_web_socket(ws):
    print(f"WebSocket opened: {ws.url}")
    ws.on("framesent", lambda payload: print(payload))
    ws.on("framereceived", lambda payload: print(payload))
    ws.on("close", lambda payload: print("WebSocket closed"))

page.on("websocket", on_web_socket)
```

Missing Network Events and Service Workers

Playwright's built-in browser_context.route() and page.route() allow your tests to natively route requests and perform mocking and interception.

1. If you're using Playwright's native browser_context.route() and page.route(), and it appears network events are missing, disable Service Workers by setting `browser.new_context.service_workers` to `'block'`.
2. It might be that you are using a mock tool such as Mock Service Worker (MSW). While this tool works out of the box for mocking responses, it adds its own Service Worker that takes over the network requests, hence making them invisible to browser_context.route() and page.route(). If you are interested in both network testing and mocking, consider using built-in browser_context.route() and page.route() for response mocking.
3. If you're interested in not solely using Service Workers for testing and network mocking, but in routing and listening for requests made by Service Workers themselves, please see this experimental feature.