Pytest Plugin Reference

Introduction

Playwright provides a Pytest plugin to write end-to-end tests. To get started with it, refer to the getting started guide.

Usage

To run your tests, use Pytest CLI.

```
pytest --browser webkit --headed
```

If you want to add the CLI arguments automatically without specifying them, you can use the pytest.ini file:

```
# content of pytest.ini
[pytest]
# Run firefox with UI
addopts = --headed --browser firefox
```

CLI arguments

Note that CLI arguments are only applied to the default `browser`, `context` and `page` fixtures. If you create a browser, a context or a page with the API call like browser.new_context(), the CLI arguments are not applied.

- `--headed`: Run tests in headed mode (default: headless).
- `--browser`: Run tests in a different browser `chromium`, `firefox`, or `webkit`. It can be specified multiple times (default: `chromium`).
- `--browser-channel` Browser channel to be used.
- `--slowmo` Slows down Playwright operations by the specified amount of milliseconds. Useful so that you can see what is going on (default: 0).

- `--device` Device to be emulated.
- `--output` Directory for artifacts produced by tests (default: `test-results`).
- `--tracing` Whether to record a trace for each test. `on`, `off`, or `retain-on-failure` (default: `off`).
- `--video` Whether to record video for each test. `on`, `off`, or `retain-on-failure` (default: `off`).
- `--screenshot` Whether to automatically capture a screenshot after each test. `on`, `off`, or `only-on-failure` (default: `off`).
- `--full-page-screenshot` Whether to take a full page screenshot on failure. By default, only the viewport is captured. Requires `--screenshot` to be enabled (default: `off`).

Fixtures

This plugin configures Playwright-specific fixtures for pytest. To use these fixtures, use the fixture name as an argument to the test function.

```
def test_my_app_is_working(fixture_name):
    pass
    # Test using fixture_name
    # ...
```

Function scope: These fixtures are created when requested in a test function and destroyed when the test ends.

- `context`: New browser context for a test.
- `page`: New browser page for a test.

Session scope: These fixtures are created when requested in a test function and destroyed when all tests end.

- `playwright`: Playwright instance.
- `browser_type`: BrowserType instance of the current browser.
- `browser`: Browser instance launched by Playwright.
- `browser_name`: Browser name as string.
- `browser_channel`: Browser channel as string.
- `is_chromium`, `is_webkit`, `is_firefox`: Booleans for the respective browser types.

Customizing fixture options: For `browser` and `context` fixtures, use the following fixtures to define custom launch options.

- `browser_type_launch_args`: Override launch arguments for browser_type.launch(). It should return a Dict.
- `browser_context_args`: Override the options for browser.new_context(). It should return a Dict.

Its also possible to override the context options (browser.new_context()) for a single test by using the `browser_context_args` marker:

```
import pytest

@pytest.mark.browser_context_args(timezone_id="Europe/Berlin", locale="en-GB")
def test_browser_context_args(page):
    assert page.evaluate("window.navigator.userAgent") == "Europe/Berlin"
    assert page.evaluate("window.navigator.languages") == ["de-DE"]
```

Parallelism: Running Multiple Tests at Once

If your tests are running on a machine with a lot of CPUs, you can speed up the overall execution time of your test suite by using `pytest-xdist` to run multiple tests at once:

```
# install dependency
pip install pytest-xdist
# use the --numprocesses flag
pytest --numprocesses auto
```

Depending on the hardware and nature of your tests, you can set `numprocesses` to be anywhere from `2` to the number of CPUs on the machine. If set too high, you may notice unexpected behavior.

See Running Tests for general information on `pytest` options.

Examples

Configure Mypy typings for auto-completion

test_my_application.py

```
from playwright.sync_api import Page

def test_visit_admin_dashboard(page: Page):
    page.goto("/admin")
    # ...
```

Configure slow mo

Run tests with slow mo with the `--slowmo` argument.

```
pytest --slowmo 100
```

Slows down Playwright operations by 100 milliseconds.

Skip test by browser

test_my_application.py

```
import pytest

@pytest.mark.skip_browser("firefox")
def test_visit_example(page):
    page.goto("https://example.com")
    # ...
```

Run on a specific browser

conftest.py

```
import pytest

@pytest.mark.only_browser("chromium")
def test_visit_example(page):
```

```
    page.goto("https://example.com")
    # ...
```

Run with a custom browser channel like Google Chrome or Microsoft Edge

```
pytest --browser-channel chrome
```

test_my_application.py

```
def test_example(page):
    page.goto("https://example.com")
```

Configure base-url

Start Pytest with the `base-url` argument. The `pytest-base-url` plugin is used for that which allows you to set the base url from the config, CLI arg or as a fixture.

```
pytest --base-url http://localhost:8080
```

test_my_application.py

```
def test_visit_example(page):
    page.goto("/admin")
    # -> Will result in http://localhost:8080/admin
```

Ignore HTTPS errors

conftest.py

```
import pytest

@pytest.fixture(scope="session")
def browser_context_args(browser_context_args):
    return {
        **browser_context_args,
```

```
        "ignore_https_errors": True
    }
```

Use custom viewport size

conftest.py

```
import pytest

@pytest.fixture(scope="session")
def browser_context_args(browser_context_args):
    return {
        **browser_context_args,
        "viewport": {
            "width": 1920,
            "height": 1080,
        }
    }
```

Device emulation

conftest.py

```
import pytest

@pytest.fixture(scope="session")
def browser_context_args(browser_context_args, playwright):
    iphone_11 = playwright.devices['iPhone 11 Pro']
    return {
        **browser_context_args,
        **iphone_11,
    }
```

Or via the CLI `--device="iPhone 11 Pro"`

Persistent context

conftest.py

```
import pytest
from playwright.sync_api import BrowserType
from typing import Dict

@pytest.fixture(scope="session")
def context(
    browser_type: BrowserType,
    browser_type_launch_args: Dict,
    browser_context_args: Dict
):
    context = browser_type.launch_persistent_context("./foobar", **{
        **browser_type_launch_args,
        **browser_context_args,
        "locale": "de-DE",
    })
    yield context
    context.close()
```

When using that all pages inside your test are created from the persistent context.

Using with `unittest.TestCase`

See the following example for using it with `unittest.TestCase`. This has a limitation, that only a single browser can be specified and no matrix of multiple browsers gets generated when specifying multiple.

```
import pytest
import unittest

from playwright.sync_api import Page


class MyTest(unittest.TestCase):
    @pytest.fixture(autouse=True)
    def setup(self, page: Page):
        self.page = page

    def test_foobar(self):
        self.page.goto("https://microsoft.com")
        self.page.locator("#foobar").click()
        assert self.page.evaluate("1 + 1") == 2
```

Debugging

Use with pdb

Use the `breakpoint()` statement in your test code to pause execution and get a pdb REPL.

```
def test_bing_is_working(page):
    page.goto("https://bing.com")
    breakpoint()
    # ...
```

Deploy to CI

See the guides for CI providers to deploy your tests to CI/CD.