UnCodemy

# Playwright API Testing Guide PDF

Playwright can test APIs without a browser using the request fixture, which sends GET, POST, PUT and DELETE calls and checks status codes, headers and JSON bodies. It also lets you create test data through the API, then verify the result in the UI, which makes end-to-end tests faster and more reliable.

## Key takeaways

- The request fixture sends HTTP calls without opening a browser.
- Assert status code, response body and important headers.
- Use API calls to prepare data and clean up after UI tests.
- Keep base URL and tokens in configuration or environment variables.
- Test error cases such as 400, 401, 404 and validation failures too.

## What is API testing with Playwright?

Playwright API testing uses the built-in APIRequestContext to send HTTP requests from test code and assert on responses, either on its own or together with browser-based UI checks.

## Why test APIs with Playwright?

If your team already uses Playwright for the interface, adding API tests keeps one runner, one report and one language. API tests run quickly because they skip rendering, and they find backend bugs earlier than UI tests. They also help set up data: creating a user through the API takes milliseconds, while clicking through a signup form takes seconds.

## How do I send a request?

```
import { test, expect } from '@playwright/test';

test('create and read a todo', async ({ request }) => {
  const created = await request.post('/api/todos', { data: { title: 'Buy milk' } });
  expect(created.status()).toBe(201);
  const body = await created.json();
  expect(body.title).toBe('Buy milk');

  const fetched = await request.get(`/api/todos/${body.id}`);
  expect(fetched.ok()).toBeTruthy();
});
```

Set `baseURL` in the configuration so paths stay short. The `request` fixture is created per test, which keeps tests isolated.

## What should I assert?

| Check | Example |
| --- | --- |
| Status code | 200, 201, 204, 400, 401, 404 |
| Body fields | Required keys, data types, values |
| Headers | Content-Type, caching headers |
| Timing | Response within an agreed limit |
| Error format | Message and code for invalid input |
| Check negative cases as carefully as success paths, since clients depend on predictable errors. | |

## How do I handle authentication?

Many APIs need a token. Log in once in a setup step, save the token in an environment variable or a storage state file, and pass it with `extraHTTPHeaders` in the configuration. For browser tests you can log in through the API, then reuse the cookies in the page, which avoids repeating the login screen in every test. Never commit real secrets; read them from environment variables.

## How do I combine API and UI steps?

Create data with the API, open the page and verify what the user sees, then delete the data with the API. For example, create an order via `request.post`, visit the order page and assert the status badge. This pattern keeps UI tests focused on one behaviour and removes long fragile setup flows.

## How do I clean up?

Use fixtures with teardown or `test.afterEach` to delete created records. Prefer unique names, such as a timestamp suffix, so parallel tests do not collide. If the environment resets nightly, still clean up to keep runs independent.

## What about schema and contract checks?

Validate that responses contain required fields and types. You can write a small helper that checks keys and types, or use a JSON schema library in the test code. Contract checks catch accidental changes, such as a renamed field, before the front end breaks.

## How does it compare with Postman?

Postman is friendly for exploration and sharing collections. Code-based tests in Playwright live in the same repository as the app tests, run in CI with the same command and can reuse helpers and fixtures. Many teams use Postman to explore and Playwright to automate.

## What mistakes should I avoid?

Testing against production data, sharing mutable records between tests, ignoring negative cases, hard-coding tokens, asserting entire large bodies instead of key fields and skipping cleanup. Keep tests small and independent.

## How do I test pagination, filters and sorting?

Send requests with query parameters and assert that the number of items, their order and the returned page metadata match the request. Test boundaries such as the first page, the last page and a page beyond the end. Check that invalid parameters return a clear client error and not a server error. These tests are cheap through the API and catch bugs that are tedious to find by clicking in the interface.

## How do I report and share API test results?

Use the same HTML or JUnit reporter as your UI tests so one dashboard shows everything. Attach the request and response summary for failed calls using test annotations or custom attachments, with secrets removed. When the backend team can open a report and see the exact payload, fixing a defect takes less discussion and less time.

## How do I test rate limits and timeouts?

Send a small burst of calls and confirm the service returns the documented limit response, then confirm normal calls succeed again after the window. Only run such checks in an environment where the owners agree, and keep the volume low so shared systems are not disturbed. Also set a request timeout in the test so a hung service fails clearly instead of stalling the pipeline.

## Frequently asked questions

### Can Playwright test APIs without a browser?

Yes. The request fixture sends HTTP calls directly, so API tests run without opening a browser and complete much faster than UI tests.

### How do I send a POST request in Playwright?

Use request.post with a URL and a data object, then assert on response.status() and the parsed JSON body.

### How can APIs speed up UI tests?

Create and delete test data through the API, so the UI test only checks one behaviour instead of clicking through long setup screens.

### Where should I store tokens?

Use environment variables or a generated storage state file, and never commit real secrets to the repository.

### Should I replace Postman with Playwright?

Not necessarily. Postman is good for exploration, while Playwright suits automated tests that run in the same pipeline as your UI tests.

## Sources and further reading

- Playwright documentation
- Playwright API reference
- Playwright API testing

## About this PDF

These PDFs are study material for learning and revision. They are not a certificate, a degree or a promise of a job, and technology changes, so check official documentation for the latest details. Spot an error? Write to info@uncodemy.com and the trainer team will review it.

More free resources and courses: uncodemy.com | info@uncodemy.com | +91 98183 66550