

[Skip to main content](#)



[PlaywrightDocsAPI](#)

[Node.js](#)

- [Node.js](#)
- [Python](#)
- [Java](#)
- [.NET](#)

[Community](#)

Search⌕

- [Getting Started](#)
  - [Installation](#)
  - [Writing tests](#)
  - [Generating tests](#)
  - [Running and debugging tests](#)
  - [Trace viewer](#)
  - [Setting up CI](#)
- [Getting started - VS Code](#)
- [Release notes](#)
- [Canary releases](#)
- [Playwright Test](#)
  - [Test configuration](#)
  - [Test use options](#)
  - [Annotations](#)
  - [Command line](#)

- [Emulation](#)
  - [Fixtures](#)
  - [Global setup and teardown](#)
  - [Parallelism](#)
  - [Parameterize tests](#)
  - [Projects](#)
  - [Reporters](#)
  - [Retries](#)
  - [Sharding](#)
  - [Timeouts](#)
  - [TypeScript](#)
  - [UI Mode](#)
  - [Web server](#)
- [Guides](#)
  - [Library](#)
  - [Accessibility testing](#)
  - [Actions](#)
  - [Assertions](#)
  - [API testing](#)
  - [Authentication](#)
  - [Auto-waiting](#)
  - [Best Practices](#)
  - [Browsers](#)
  - [Chrome extensions](#)
  - [Clock](#)
  - [Components \(experimental\)](#)
  - [Debugging Tests](#)
  - [Dialogs](#)
  - [Downloads](#)
  - [Evaluating JavaScript](#)
  - [Events](#)
  - [Extensibility](#)
  - [Frames](#)
  - [Handles](#)
  - [Isolation](#)
  - [Locators](#)
  - [Mock APIs](#)
  - [Mock browser APIs](#)
  - [Navigations](#)
  - [Network](#)
  - [Other locators](#)
  - [Page object models](#)
  - [Pages](#)
  - [Screenshots](#)
  - [Visual comparisons](#)
  - [Test generator](#)
  - [Trace viewer](#)
  - [Videos](#)
  - [WebView2](#)
- [Migration](#)

- [Integrations](#)
- [Supported languages](#)
- 
- Guides
- API testing

On this page

# API testing

## Introduction

Playwright can be used to get access to the [REST](#) API of your application.

Sometimes you may want to send requests to the server directly from Node.js without loading a page and running js code in it. A few examples where it may come in handy:

- Test your server API.
- Prepare server side state before visiting the web application in a test.
- Validate server side post-conditions after running some actions in the browser.

All of that could be achieved via [APIRequestContext](#) methods.

## Writing API Test

[APIRequestContext](#) can send all kinds of HTTP(S) requests over network.

The following example demonstrates how to use Playwright to test issues creation via [GitHub API](#). The test suite will do the following:

- Create a new repository before running tests.
- Create a few issues and validate server state.
- Delete the repository after running tests.

## Configuration

GitHub API requires authorization, so we'll configure the token once for all tests. While at it, we'll also set the `baseURL` to simplify the tests. You can either put them in the configuration file, or in the test file with `test.use()`.

playwright.config.ts

```
import { defineConfig } from '@playwright/test';
export default defineConfig({
  use: {
    // All requests we send go to this API endpoint.
    baseURL: 'https://api.github.com',
    extraHTTPHeaders: {
      // We set this header per GitHub guidelines.
```

```

    'Accept': 'application/vnd.github.v3+json',
    // Add authorization token to all requests.
    // Assuming personal access token available in the environment.
    'Authorization': `token ${process.env.API_TOKEN}`,
  },
}
});

```

## Proxy configuration

If your tests need to run behind a proxy, you can specify this in the config and the `request` fixture will pick it up automatically:

`playwright.config.ts`

```

import { defineConfig } from '@playwright/test';
export default defineConfig({
  use: {
    proxy: {
      server: 'http://my-proxy:8080',
      username: 'user',
      password: 'secret'
    },
  },
});

```

## Writing tests

Playwright Test comes with the built-in `request` fixture that respects configuration options like `baseUrl` or `extraHTTPHeaders` we specified and is ready to send some requests.

Now we can add a few tests that will create new issues in the repository.

```

const REPO = 'test-repo-1';
const USER = 'github-username';

test('should create a bug report', async ({ request }) => {
  const newIssue = await request.post(`/repos/${USER}/${REPO}/issues`, {
    data: {
      title: '[Bug] report 1',
      body: 'Bug description',
    }
  });
  expect(newIssue.ok()).toBeTruthy();

  const issues = await request.get(`/repos/${USER}/${REPO}/issues`);
  expect(issues.ok()).toBeTruthy();
  expect(await issues.json()).toContainEqual(expect.objectContaining({
    title: '[Bug] report 1',
    body: 'Bug description'
  })));
});

test('should create a feature request', async ({ request }) => {
  const newIssue = await request.post(`/repos/${USER}/${REPO}/issues`, {
    data: {
      title: '[Feature] request 1',
      body: 'Feature description',
    }
  });
  expect(newIssue.ok()).toBeTruthy();
  const issues = await request.get(`/repos/${USER}/${REPO}/issues`);
  expect(issues.ok()).toBeTruthy();
  expect(await issues.json()).toContainEqual(expect.objectContaining({
    title: '[Feature] request 1',
    body: 'Feature description'
  })));
});

```

```

    }
  });
  expect(newIssue.ok()).toBeTruthy();

  const issues = await request.get(`/repos/${USER}/${REPO}/issues`);
  expect(issues.ok()).toBeTruthy();
  expect(await issues.json()).toContainEqual(expect.objectContaining({
    title: '[Feature] request 1',
    body: 'Feature description'
  })));
});
});

```

## Setup and teardown

These tests assume that repository exists. You probably want to create a new one before running tests and delete it afterwards. Use `beforeAll` and `afterAll` hooks for that.

```

test.beforeAll(async ({ request }) => {
  // Create a new repository
  const response = await request.post('/user/repos', {
    data: {
      name: REPO
    }
  });
  expect(response.ok()).toBeTruthy();
});

test.afterAll(async ({ request }) => {
  // Delete the repository
  const response = await request.delete(`/repos/${USER}/${REPO}`);
  expect(response.ok()).toBeTruthy();
});

```

## Using request context

Behind the scenes, [request fixture](#) will actually call [apiRequest.newContext\(\)](#). You can always do that manually if you'd like more control. Below is a standalone script that does the same as `beforeAll` and `afterAll` from above.

```

import { request } from '@playwright/test';
const REPO = 'test-repo-1';
const USER = 'github-username';

(async () => {
  // Create a context that will issue http requests.
  const context = await request.newContext({
    baseURL: 'https://api.github.com',
  });

  // Create a repository.
  await context.post('/user/repos', {
    headers: {
      'Accept': 'application/vnd.github.v3+json',
      // Add GitHub personal access token.
      'Authorization': `token ${process.env.API_TOKEN}`,
    },
    data: {

```

```

    name: REPO
  }
});

// Delete a repository.
await context.delete(`/repos/${USER}/${REPO}`, {
  headers: {
    'Accept': 'application/vnd.github.v3+json',
    // Add GitHub personal access token.
    'Authorization': `token ${process.env.API_TOKEN}`,
  }
});
}) ();

```

## Sending API requests from UI tests

While running tests inside browsers you may want to make calls to the HTTP API of your application. It may be helpful if you need to prepare server state before running a test or to check some postconditions on the server after performing some actions in the browser. All of that could be achieved via [APIRequestContext](#) methods.

### Establishing preconditions

The following test creates a new issue via API and then navigates to the list of all issues in the project to check that it appears at the top of the list.

```

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

const REPO = 'test-repo-1';
const USER = 'github-username';

// Request context is reused by all tests in the file.
let apiContext;

test.beforeAll(async ({ playwright }) => {
  apiContext = await playwright.request.newContext({
    // All requests we send go to this API endpoint.
    baseURL: 'https://api.github.com',
    extraHTTPHeaders: {
      // We set this header per GitHub guidelines.
      'Accept': 'application/vnd.github.v3+json',
      // Add authorization token to all requests.
      // Assuming personal access token available in the environment.
      'Authorization': `token ${process.env.API_TOKEN}`,
    },
  });
});

test.afterAll(async ({ }) => {
  // Dispose all responses.
  await apiContext.dispose();
});

test('last created issue should be first in the list', async ({ page }) => {
  const newIssue = await apiContext.post(`/repos/${USER}/${REPO}/issues`, {
    data: {

```

```

        title: '[Feature] request 1',
      }
    });
    expect(newIssue.ok()).toBeTruthy();

    await page.goto(`https://github.com/${USER}/${REPO}/issues`);
    const firstIssue = page.locator(`a[data-hovercard-
type='issue']`).first();
    await expect(firstIssue).toHaveText('[Feature] request 1');
  });
}

```

## Validating postconditions

The following test creates a new issue via user interface in the browser and then uses checks if it was created via API:

```

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

const REPO = 'test-repo-1';
const USER = 'github-username';

// Request context is reused by all tests in the file.
let apiContext;

test.beforeAll(async ({ playwright }) => {
  apiContext = await playwright.request.newContext({
    // All requests we send go to this API endpoint.
    baseURL: 'https://api.github.com',
    extraHTTPHeaders: {
      // We set this header per GitHub guidelines.
      'Accept': 'application/vnd.github.v3+json',
      // Add authorization token to all requests.
      // Assuming personal access token available in the environment.
      'Authorization': `token ${process.env.API_TOKEN}`,
    },
  });
});

test.afterAll(async ({ }) => {
  // Dispose all responses.
  await apiContext.dispose();
});

test('last created issue should be on the server', async ({ page }) => {
  await page.goto(`https://github.com/${USER}/${REPO}/issues`);
  await page.getByText('New Issue').click();
  await page.getByRole('textbox', { name: 'Title' }).fill('Bug report 1');
  await page.getByRole('textbox', { name: 'Comment body' }).fill('Bug
description');
  await page.getByText('Submit new issue').click();
  const issueId = page.url().substr(page.url().lastIndexOf('/'));

  const newIssue = await apiContext.get(
    `https://api.github.com/repos/${USER}/${REPO}/issues/${issueId}`
  );
  expect(newIssue.ok()).toBeTruthy();
  expect(newIssue.json()).toEqual(expect.objectContaining({
    title: 'Bug report 1'
  }));
});

```

## Reusing authentication state

Web apps use cookie-based or token-based authentication, where authenticated state is stored as [cookies](#). Playwright provides [apiRequestContext.storageState\(\)](#) method that can be used to retrieve storage state from an authenticated context and then create new contexts with that state.

Storage state is interchangeable between [BrowserContext](#) and [APIRequestContext](#). You can use it to log in via API calls and then create a new context with cookies already there. The following code snippet retrieves state from an authenticated [APIRequestContext](#) and creates a new [BrowserContext](#) with that state.

```
const requestContext = await request.newContext({
  httpCredentials: {
    username: 'user',
    password: 'passwd'
  }
});
await requestContext.get('https://api.example.com/login');
// Save storage state into the file.
await requestContext.storageState({ path: 'state.json' });

// Create a new context with the saved storage state.
const context = await browser.newContext({ storageState: 'state.json' });
```

## Context request vs global request

There are two types of [APIRequestContext](#):

- associated with a [BrowserContext](#)
- isolated instance, created via [apiRequest.newContext\(\)](#)

The main difference is that [APIRequestContext](#) accessible via [browserContext.request](#) and [page.request](#) will populate request's Cookie header from the browser context and will automatically update browser cookies if [APIResponse](#) has Set-Cookie header:

```
test('context request will share cookie storage with its browser context',
  async ({
    page,
    context,
  }) => {
    await context.route('https://www.github.com/', async route => {
      // Send an API request that shares cookie storage with the browser
      context.
      const response = await context.request.fetch(route.request());
      const responseHeaders = response.headers();

      // The response will have 'Set-Cookie' header.
      const responseCookies = new Map(responseHeaders['set-cookie']
        .split('\n')
        .map(c => c.split(';')[0].split('=')));
      // The response will have 3 cookies in 'Set-Cookie' header.
      expect(responseCookies.size).toBe(3);
      const contextCookies = await context.cookies();
```



```

    // The browser context will already contain all the cookies from the
    API response.
    expect(new Map(contextCookies.map(({ name, value }) =>
      [name, value]))
    ).toEqual(responseCookies);

    await route.fulfill({
      response,
      headers: { ...responseHeaders, foo: 'bar' },
    });
  });
  await page.goto('https://www.github.com/');
});

```

If you don't want [APIRequestContext](#) to use and update cookies from the browser context, you can manually create a new instance of [APIRequestContext](#) which will have its own isolated cookies:

```

test('global context request has isolated cookie storage', async ({
  page,
  context,
  browser,
  playwright
}) => {
  // Create a new instance of APIRequestContext with isolated cookie
  storage.
  const request = await playwright.request.newContext();
  await context.route('https://www.github.com/', async route => {
    const response = await request.fetch(route.request());
    const responseHeaders = response.headers();

    const responseCookies = new Map(responseHeaders['set-cookie']
      .split('\n')
      .map(c => c.split('; ', 2)[0].split('=')));
    // The response will have 3 cookies in 'Set-Cookie' header.
    expect(responseCookies.size).toBe(3);
    const contextCookies = await context.cookies();
    // The browser context will not have any cookies from the isolated API
    request.
    expect(contextCookies.length).toBe(0);

    // Manually export cookie storage.
    const storageState = await request.storageState();
    // Create a new context and initialize it with the cookies from the
    global request.
    const browserContext2 = await browser.newContext({ storageState });
    const contextCookies2 = await browserContext2.cookies();
    // The new browser context will already contain all the cookies from
    the API response.
    expect(
      new Map(contextCookies2.map(({ name, value }) => [name, value]))
    ).toEqual(responseCookies);

    await route.fulfill({
      response,
      headers: { ...responseHeaders, foo: 'bar' },
    });
  });
  await page.goto('https://www.github.com/');

```

```
await request.dispose();  
});
```

[Previous](#)  
[Assertions](#)

[Next](#)  
[Authentication](#)

- [Introduction](#)
- [Writing API Test](#)
  - [Configuration](#)
  - [Writing tests](#)
  - [Setup and teardown](#)
- [Using request context](#)
- [Sending API requests from UI tests](#)
  - [Establishing preconditions](#)
  - [Validating postconditions](#)
- [Reusing authentication state](#)
- [Context request vs global request](#)

## Learn

- [Getting started](#)
- [Playwright Training](#)
- [Learn Videos](#)
- [Feature Videos](#)

## Community

- [Stack Overflow](#)
- [Discord](#)
- [Twitter](#)
- [LinkedIn](#)

## More

- [GitHub](#)
- [YouTube](#)
- [Blog](#)
- [Ambassadors](#)

Copyright © 2024 Microsoft