

Перехоплення мережевих запитів у Playwright

Повний практичний посібник для розробників і тестувальників, які хочуть опанувати моніторинг, очікування, модифікацію та блокування HTTP-запитів у своїх автотестах.

Вступ: Що таке мережеві запити і навіщо їх перехоплювати?

Мережевий запит (HTTP request) — це повідомлення, яке браузер відправляє на сервер, коли ви відкриваєте сторінку, натискаєте кнопку, завантажуєте зображення тощо. Кожна взаємодія користувача з вебдодатком у переважній більшості випадків супроводжується одним або кількома мережевими запитами — від отримання HTML-сторінки до завантаження шрифтів, стилів, скриптів і даних з API.

Відповідь (response) — це те, що сервер повертає у відповідь на запит. Вона містить статус-код (наприклад, 200 — успіх, 404 — не знайдено), заголовки та тіло відповіді з даними. Розуміння цього обміну є фундаментальним для будь-якого тестувальника або розробника, який працює з автоматизацією браузера.

У цьому посібнику ми крок за кроком розберемо всі можливості Playwright для роботи з мережевими запитами — від простого спостереження до повного контролю над трафіком між браузером і сервером. Ви навчитеся логувати запити, чекати на конкретні відповіді, модифікувати заголовки та блокувати небажані ресурси.

Аналогія: браузер як офіціант у ресторані

Уявіть, що браузер – це **офіціант у ресторані**. Користувач робить замовлення (клікає, заповнює форму) → офіціант несе запит на кухню (сервер) → кухня готує страву і повертає її. Кожна така «пробіжка» між столиком і кухнею – це мережевий запит і відповідь. Ця аналогія допоможе вам інтуїтивно зрозуміти, що саме відбувається під капотом браузера.

Перехоплення (interception) – це можливість «підслухати» або змінити цей обмін, не зачіпаючи реальний сервер. Це як якщо б ви могли зупинити офіціанта на шляху до кухні, подивитися на замовлення, змінити його або навіть повернути офіціанта назад із готовою стравою, не чекаючи кухні. Playwright надає саме такий рівень контролю.

	<h3>Спостерігати</h3> <p>Логувати та аналізувати запити й відповіді в реальному часі</p>		<h3>Чекати</h3> <p>Синхронізуватися з конкретним запитом або відповіддю для надійних тестів</p>
	<h3>Змінювати</h3> <p>Модифікувати заголовки, URL, метод або тіло запиту перед відправкою</p>		<h3>Блокувати</h3> <p>Відключати зовнішні ресурси, рекламу чи аналітику для прискорення тестів</p>

Усі ці можливості роблять Playwright надзвичайно потужним інструментом не лише для тестування UI, а й для повного контролю мережевого рівня вебдодатку. Далі ми розглянемо кожну з цих технік детально, з прикладами коду та практичними порадами.

Основи: Event Listener vs методи очікування

Event Listener (слухач подій) – це функція, яка викликається щоразу, коли відбувається певна подія. При підписці на `request` і `response` код виконується для *кожного* запиту й відповіді, що проходять через сторінку. Це пасивний підхід – ви просто «слухаєте» все, що відбувається.

На відміну від цього, **методи очікування** (`waitForRequest`, `waitForResponse`) – це активний підхід. Вони повертають `Promise` і дочекуються саме потрібної події. Це критично важливо для тестів, де вам потрібно переконатися, що конкретний запит був відправлений або конкретна відповідь отримана.

Два підходи в Playwright

Підхід	Що робить	Коли використовувати
Event Listener (<code>page.on</code>)	Пасивне спостереження за всіма подіями	Моніторинг, логування, аналіз мережевої активності
Методи очікування (<code>waitFor...</code>)	Активне очікування конкретного запиту/відповіді	Синхронізація з UI, асерти, перевірка факту відправки

- ☐ **Ключова різниця:** Event Listener не чекає – він реагує на всі події. Методи очікування повертають `Promise` і дочекуються саме потрібної події. Плутати ці підходи – одна з найпоширеніших помилок новачків.

1. Моніторинг: спостереження за всіма запитами

Навіщо це потрібно?

Моніторинг мережевих запитів дозволяє бачити, які запити йдуть зі сторінки, які URL викликаються та які статуси повертаються. Це незамінний інструмент для налагодження та розуміння поведінки додатку під час виконання тестів.

Як працює: `page.on('request', ...)` та `page.on('response', ...)` додають обробники, які викликаються при кожному запиті й відповіді. Вони працюють як «камери спостереження» — фіксують усе, що проходить повз.

Коли використовувати: коли потрібно просто спостерігати за мережовою активністю, без умовного очікування або змін. Ідеально для дебагу та аудиту.



Приклад коду

```
// Підписка на події 'request' та 'response'  
page.on('request', request =>  
    console.log('>>', request.method(), request.url())  
);  
page.on('response', response =>  
    console.log('<<', response.status(), response.url())  
);  
await page.goto('https://example.com');
```

1

>> – вихідний запит

Браузер відправляє запит на сервер.
`request.method()` повертає HTTP-метод (GET, POST тощо).

2

<< – вхідна відповідь

Сервер повертає відповідь. `response.status()` повертає HTTP-код (200, 404 тощо).

2. Очікування конкретного запиту: waitForRequest

Навіщо: щоб переконатися, що після певної дії на сторінці (наприклад, кліку) браузер дійсно відправив потрібний запит. Це дозволяє робити надійні асерти та синхронізуватися з мережевим рівнем додатку.

- ⚠ **Важливо про порядок викликів:** метод `waitForRequest` **не чекає await** відразу — він починає «слухати» запити. `Await` ми ставимо на `promise` **після** дії, яка має ініціювати запит. Інакше можна пропустити запит або отримати таймаут.

Базовий приклад

```
// Крок 1: Почати слухати запити (БЕЗ await!)
const requestPromise = page.waitForRequest(
  'https://example.com/resource'
);

// Крок 2: Виконати дію, що має викликати запит
await page.getText('trigger request').click();

// Крок 3: Дочекатися саме цього запиту
const request = await requestPromise;
```



Цей патерн «почати слухати → виконати дію → дочекатися» є фундаментальним у Playwright і запобігає стану гонки (race condition).

Альтернатива з предикатом

Коли потрібно фільтрувати за URL і методом одночасно, використовуйте функцію-предикат:

```
const requestPromise = page.waitForRequest(
  request =>
    request.url() === 'https://example.com' &&
    request.method() === 'GET',
);
await page.getText('trigger request').click();
const request = await requestPromise;
```

3. Очікування відповіді: waitForResponse

Навіщо: щоб синхронізуватися з відповіддю сервера. Часто UI оновлюється саме після отримання відповіді, тож дочекатися response надійніше, ніж дочекатися зміни UI-елемента. Це особливо важливо для додатків з асинхронним завантаженням даних через API.

Метод `waitForResponse` працює за тим самим патерном, що й `waitForRequest`: спочатку починаємо слухати, потім виконуємо дію, і нарешті дочекуємося потрібної відповіді.

Приклад коду

```
// Чекаємо відповідь на запит до API
const responsePromise = page.waitForResponse(
  '**/api/fetch_data'
);
await page.getText('Update').click();
const response = await responsePromise;
```

Вибір формату залежить від складності вашого фільтра. Для простих випадків достатньо рядка або glob-патерну. Для складних – використовуйте функцію-предикат, яка дає максимальну гнучкість і дозволяє перевіряти URL, метод, заголовки та інші параметри одночасно.

Що можна передати в метод?

- ◆ **Рядок** – точний URL або його частина
- ◆ **Glob-патерн** – шаблон з підстановками (**/api/**)
- ◆ **RegExp** – регулярний вираз для складних випадків
- ◆ **Функція-предикат** – для фільтрації за кількома критеріями

Glob-патерни для URL

Glob-патерни – це шаблони зі спеціальними символами для зіставлення рядків. У Playwright вони застосовуються до URL так само, як до шляхів файлів. Вони значно спрощують фільтрацію запитів, дозволяючи охоплювати цілі групи URL одним виразом замість точних збігів.

Таблиця символів

Символ	Значення
*	Будь-яка послідовність символів (крім роздільника шляху)
?	Будь-який один символ
[]	Діапазон символів (наприклад, [0-9])
{}	Вибір варіантів (наприклад, {txt,doc})

Практичні приклади для URL

**/api/users

Будь-який URL, що закінчується на /api/users. Подвійна зірочка ** означає будь-який шлях на початку.

**/*.png,.jpg}

Зображення з розширеннями .png або .jpg. Фігурні дужки дозволяють вказати кілька варіантів.

/api/

Будь-який URL, що містить /api/ в шляху. Зручно для перехоплення всіх API-запитів одночасно.

Glob-патерни – це найзручніший спосіб фільтрації URL у Playwright. Вони читабельніші за регулярні вирази і покривають більшість типових сценаріїв. Використовуйте їх як стандартний підхід, а RegExp – лише для складних випадків.

4. Модифікація запитів: page.route

Навіщо: щоб змінити заголовки, метод, URL або взагалі блокувати певні запити (наприклад, рекламу, аналітику, зовнішні скрипти). Метод `page.route` дає повний контроль над тим, що браузер відправляє на сервер.

Як працює: ви реєструєте обробник для URL (або патерну). Коли браузер збирається виконати відповідний запит, Playwright зупиняє його і викликає ваш код. Ви повинні або викликати `route.continue()` (продовжити зміненим запитом), або `route.abort()` (скасувати запит).



Реєстрація маршруту

`page.route(pattern, handler)`



Перехоплення

Playwright зупиняє запит



Модифікація

Ваш код змінює запит



Продовження

`route.continue()` або `route.abort()`

Видалення заголовка

```
await page.route('**/*', route => {
  const headers = {
    ...route.request().headers()
  };
  delete headers['X-Secret'];
  route.continue({ headers });
});
```

Зміна методу запиту

```
await page.route('**/*', route =>
  route.continue({ method: 'POST' })
);
```

Тут ми перетворюємо всі запити на POST. На практиці варто фільтрувати конкретніше за URL-патерном.

Цей приклад перехоплює всі запити та видаляє конфіденційний заголовок X-Secret перед відправкою.

5. Блокування запитів: route.abort

Навіщо: щоб не завантажувати певні ресурси – зображення, рекламу, трекери. Це пришвидшує тести і зменшує залежність від зовнішніх сервісів. У великих проектах блокування непотрібних ресурсів може скоротити час виконання тестів на 30-50%.

Приклади коду

```
// Блокуємо всі зображення за розширенням
await page.route('**/*.{png,jpg,jpeg}', route =>
  route.abort()
);

// Блокуємо лише певний тип ресурсів
await page.route('**/*', route => {
  return route.request().resourceType() === 'image'
    ? route.abort()
    : route.continue();
});
```

Типи ресурсів (resourceType)

resourceType – тип ресурсу за класифікацією Chrome DevTools Protocol. Знання цих типів дозволяє точно контролювати, що блокувати:



HTML-документи сторінок



stylesheet

CSS-файли стилів



JavaScript-файли



image

Зображення (png, jpg, svg...)



Веб-шрифти



xhr / fetch

AJAX-запити до API

Поширені помилки

Навіть досвідчені розробники іноді припускаються цих помилок при роботі з перехопленням мережевих запитів у Playwright. Ось три найпоширеніші пастки та способи їх уникнення:

1 Помилковий порядок await

Якщо спочатку зробити `await waitForRequest(...)`, а потім клік — запит вже може пройти до моменту початку очікування. Це спричинить таймаут тесту.

Правильно: спочатку отримати `promise` без `await`, потім виконати дію, потім `await promise`.

2 Забули викликати `route.continue()` або `route.fill()`

Якщо в обробнику `page.route` не викликати жодного з методів завершення (`continue`, `fill`, `abort`), запит зависне назавжди. Тест зупиниться по таймауту, і причину буде складно діагностувати.

3 Неправильний glob-патерн

Переконайтесь, що патерн дійсно збігається з потрібними URL. Наприклад, використовуйте `**/api/**` замість просто `api` для будь-якого шляху, що містить `/api/`. Тестуйте патерни окремо перед використанням у тестах.

✗ Неправильно

```
// Запит може пройти до початку
// очікування — таймаут!
const req = await page.waitForRequest(url);
await page.click('button');
```

✓ Правильно

```
// Спочатку почати слухати,
// потім виконати дію
const reqPromise = page.waitForRequest(url);
await page.click('button');
const req = await reqPromise;
```

Підсумок: коли що використовувати

Ось зведена таблиця, яка допоможе вам швидко обрати правильний метод для кожної задачі при роботі з мережевими запитами в Playwright:

Задача	Метод
Просто подивитися всі запити й відповіді	<code>page.on('request', ...)</code> / <code>page.on('response', ...)</code>
Дочекатися конкретного запиту	<code>page.waitForRequest(...)</code>
Дочекатися відповіді на запит	<code>page.waitForResponse(...)</code>
Змінити запит перед відправкою	<code>page.route(..., route => route.continue({...}))</code>
Повністю блокувати запити	<code>page.route(..., route => route.abort())</code>



Моніторинг

Використовуйте `page.on` для пасивного спостереження та дебагу



Очікування

Використовуйте `waitFor*` для надійної синхронізації тестів



Модифікація

Використовуйте `route.continue()` для зміни запитів



Блокування

Використовуйте `route.abort()` для прискорення тестів

Опанувавши ці п'ять технік, ви отримаєте повний контроль над мережевим рівнем вашого додатку в тестах. Це фундамент для створення надійних, швидких і незалежних від зовнішніх сервісів автотестів у Playwright.