

Мокінг (підміна) відповідей на мережеві запити

Повний практичний гайд з перехоплення та підміни HTTP-відповідей у Playwright для стабільного й швидкого автоматизованого тестування.

Що таке мокінг?

Мокінг (*mocking*) – це техніка заміни справжньої відповіді сервера на штучну, заздалегідь задану в тесті. Браузер і фронтенд-код «думають», що отримали реальні дані від бекенду, але насправді їм повертаються дані, які ви підготували самостійно. Це дозволяє повністю контролювати те, що бачить клієнтський код, і перевіряти його реакцію на будь-які сценарії – від успішних відповідей до помилок і порожніх результатів.

Мокінг – фундаментальний інструмент для будь-якого розробника або тестувальника, який працює з мережевими запитами. Він дає змогу ізолювати фронтенд від бекенду, зробити тести передбачуваними та значно прискорити їх виконання. У контексті Playwright мокінг реалізується через перехоплення мережевих маршрутів і повернення підготовлених відповідей замість реальних.



Аналогія: замість реального банківського API ви показуєте програмі «підроблений» лист від банку з потрібними числами. Додаток реагує так само, як на справжні дані, але реальний банк не потрібен.

Навіщо мокати відповіді?

Мокінг мережевих відповідей – це не просто зручність, а необхідність для побудови надійного та ефективного процесу тестування. Коли тести залежать від реального сервера, бази даних або зовнішніх API, будь-який збій на стороні бекенду може призвести до хибних падінь тестів, які не мають жодного стосунку до якості фронтенд-коду. Мокінг усуває цю залежність повністю.

Крім стабільності, мокінг відкриває можливість тестувати сценарії, які важко або неможливо відтворити в реальному середовищі: відповіді з помилками 500, порожні списки, дані з граничними значеннями, тайм-аути. Без мокінгу підготовка таких умов вимагала б складних маніпуляцій із базою даних або спеціальних тестових середовищ.



Незалежність

Не залежати від бекенду, бази даних та зовнішніх сервісів. Тести працюють навіть коли сервер недоступний.



Рідкісні сценарії

Перевіряти помилки, порожні списки, великі дані без підготовки даних у продакшені чи staging-середовищі.



Швидкість

Відповідь приходить миттєво, без мережі. Тести виконуються в рази швидше за інтеграційні.



Стабільність

Поведінка не зміниться через збої на сервері. Результати тестів детерміновані та повторювані.



- 💡 Мокінг особливо корисний у CI/CD-пайплайнах, де швидкість і стабільність тестів критично важливі для частих деплоїв.

Механізм: route + fulfill

В Playwright весь мокінг будується на двох ключових елементах: методі `page.route()` для перехоплення запитів та методі `route.fulfill()` для повернення підготовленої відповіді. Ця пара утворює основу для будь-якої підміни мережевих даних – від простої заміни тіла відповіді до складних сценаріїв із модифікацією оригінальних даних.

route.fulfill()

Цей метод каже Playwright: «не відправляй цей запит на сервер, а віддай браузеру ось цю відповідь». Ви задаєте статус-код, заголовки та тіло відповіді – і браузер обробляє їх так, ніби дані прийшли з реального сервера.

Важливе правило

Кожен перехоплений запит **має бути або продовжений** через `route.continue()`, або **задоволений** через `route.fulfill()`. Якщо нічого не викликати, запит зависне і тест завершиться по таймауту – це одна з найпоширеніших помилок.

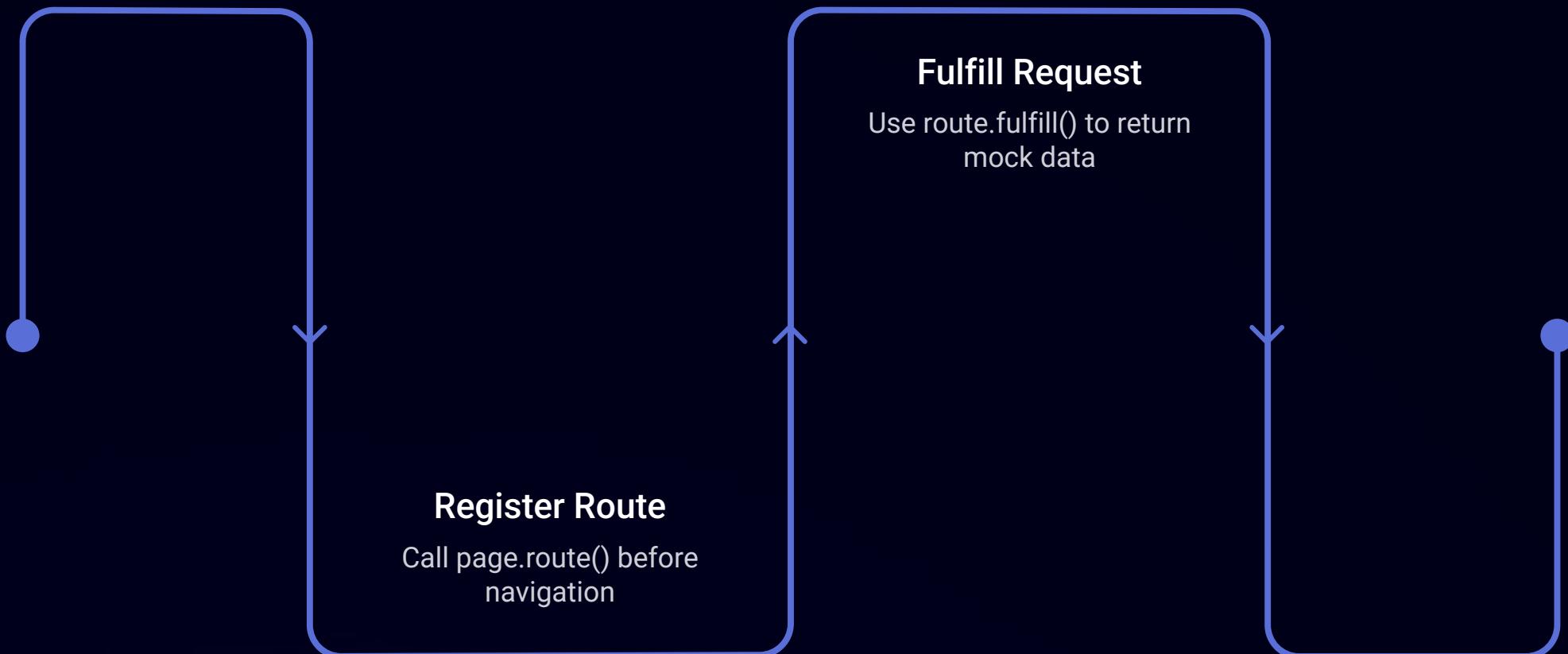
Порядок дій

1 Зареєструвати маршрут

Викликати `page.route()` **до** `page.goto()` або до дії, що ініціює запит.

2 Обробити запит

В обробнику викликати `route.fulfill()` з потрібними даними відповіді.



Цей механізм дозволяє повністю контролювати мережевий рівень тестованого додатку, не змінюючи жодного рядка продакшн-коду.

Простий мок: заміна відповіді на фіксовані дані

Найпростіший і найпоширеніший сценарій мокінгу – повна заміна відповіді на фіксовані дані без звернення до реального сервера. У цьому випадку ви реєструєте маршрут для певного URL-патерну і повертаєте заздалегідь підготовлену відповідь зі статусом та тілом.

У прикладі нижче ми перехоплюємо будь-який запит, що закінчується на `/api/login`, і повертаємо відповідь зі статусом 200 та текстом 'accept':

```
await page.route('**/api/login', route =>
  route.fulfill({
    status: 200,
    body: 'accept',
  })
);
await page.goto('https://example.com');
```



status: 200

HTTP-код «успішно». Інші типові коди: **201** (створено), **404** (не знайдено), **500** (помилка сервера). Вибір коду визначає, як фронтенд обробить відповідь.



body

Тіло відповіді. Може бути рядком (`string`), `Buffer` або об'єктом для `JSON`. Це те, що фронтенд-код прочитає як дані від сервера.

- ⚠ **Типова помилка:** зареєструвати `route` **після** `page.goto()`. Частина запитів уже піде до реального сервера, і їх ви не замокаєте. **Завжди** налаштовуйте маршрут **перед** дією, що викликає запит.

Мок на основі оригінальної відповіді

Іноді потрібно не повністю замінювати відповідь сервера, а лише трохи її модифікувати – додати поле, змінити значення або підкорегувати HTML. Для цього в Playwright є метод `route.fetch()`, який виконує оригінальний запит до сервера, повертає реальну відповідь, а далі ви можете її змінити і передати клієнту через `route.fill()`.



Точкові зміни в даних

Перевірити, як UI обробляє невелику зміну в даних – наприклад, інше ім'я користувача або змінену ціну товару.



Додавання полів

Додати або змінити окремі поля без переписування всього тіла відповіді – особливо зручно для великих JSON-структур.



Збереження структури

Зберегти оригінальну структуру відповіді, змінивши лише потрібну частину – мінімальне втручання, максимальний контроль.

Приклад: зміна заголовка HTML

```
await page.route('**/title.html', async route => {
  // 1. Отримати оригінальну відповідь
  const response = await route.fetch();
  // 2. Прочитати тіло і змінити його
  let body = await response.text();
  body = body.replace('<title>', '<title>My prefix:');
  // 3. Повернути модифіковану відповідь
  route.fill({
    response,      // Базові параметри з оригіналу
    body,          // Новий текст тіла
    headers: {
      ...response.headers(),
      'content-type': 'text/html'
    }
  });
});
```

Зверніть увагу на патерн: спочатку `route.fetch()` для отримання реальних даних, потім модифікація, і нарешті `route.fill()` з параметром `response`, який передає оригінальні метадані (статус, заголовки), та новим тілом. Це гарантує, що ваша підміна буде максимально наблизена до реальної відповіді.

Мок JSON-відповіді: додавання або зміна полів

Більшість сучасних API повертають дані у форматі JSON. Playwright надає зручний спосіб працювати з такими відповідями: ви можете розпарсити JSON з оригінальної відповіді, модифікувати об'єкт як звичайний JavaScript-об'єкт і передати його назад через опцію `json` у `route.fetch()`. Playwright автоматично серіалізує об'єкт і встановить правильний заголовок `Content-Type`.

```
await page.route('https://dog.ceo/api/breeds/list/all', async route => {
  const response = await route.fetch();
  const json = await response.json();
  // Додати нове поле до існуючої структури
  json.message['big_red_dog'] = [];
  await route.fulfill({ response, json });
});
```

Опції `route.fulfill()`

`response`

Перевикористання метаданих (статус, заголовки) з оригінальної відповіді. Зручно, щоб не задавати все вручну.

`json`

Тіло як JavaScript-об'єкт. Playwright автоматично перетворить його в JSON-рядок і встановить правильний `Content-Type: application/json`.

`body`

Текстове тіло відповіді – альтернатива `json`. Використовуйте для HTML, plain text або коли потрібен повний контроль над серіалізацією.

`status, headers`

Можна перевизначити окремо. Наприклад, повернути `status: 404` для імітації помилки або додати кастомні заголовки.

Опція `json` – найзручніший спосіб працювати з API-мокінгом, оскільки вона усуває потребу вручну серіалізувати дані та встановлювати заголовки. Це зменшує ймовірність помилок і робить тести чистішими та зрозумілішими.

Повна заміна відповіді

Іноді вам не потрібні оригінальні дані з сервера взагалі – ви хочете повністю замінити тіло відповіді власним об'єктом. Це корисно, коли ви тестуєте конкретний сценарій із заздалегідь відомою структурою даних, або коли реальний API ще не реалізований, а фронтенд вже потрібно тестувати.

Приклад коду

```
await page.route(  
  'https://dog.ceo/api/breeds/list/all',  
  async route => {  
    const response = await route.fetch();  
    const json = {  
      user: "John Doe",  
      age: 27  
    };  
    await route.fulfill({  
      response,  
      json  
    });  
  }  
);
```

Що відбувається?

01

Перехоплення

Playwright перехоплює запит до вказаного URL.

02

Fetch оригіналу

`route.fetch()` отримує відповідь для метаданих (заголовки, статус).

03

Повна заміна

Тіло відповіді повністю замінюється вашим JSON-об'єктом.

04

Відправка клієнту

Браузер отримує ваші дані як реальну відповідь сервера.

У цьому прикладі параметр `response` використовується лише для передачі базових метаданих (заголовки, статус-код), а тіло повністю замінюється власним об'єктом `json`. Такий підхід дає максимальний контроль над тим, що побачить фронтенд, і не вимагає жодної підготовки даних на стороні сервера.

Зверніть увагу: якщо вам не потрібні навіть метадані оригінальної відповіді, ви можете взагалі не викликати `route.fetch()` і просто задати всі параметри вручну – `status`, `headers` і `json` або `body`.

Поширені помилки

Навіть досвідчені розробники допускають типові помилки при роботі з мокінгом у Playwright. Розуміння цих підвідних каменів допоможе вам уникнути фрустрації під час налагодження тестів і зекономити значну кількість часу. Нижче – чотири найпоширеніші проблеми та способи їх вирішення.

1

Route після goto

Маршрут не спрацює для запитів, які вже пішли до сервера. **Завжди** реєструйте `page.route()` **до** виклику `page.goto()`. Навіть мілісекундна затримка може привести до того, що частина запитів піде повз ваш мок.

2

Забути fulfill або continue

В обробнику `page.route()` **обов'язково** викличте `route.fulfill()` або `route.continue()`. Інакше запит зависне, браузер чекатиме нескінченно, і тест завершиться з помилкою таймауту.

3

Неправильний Content-Type

Для JSON-відповідей краще використовувати опцію `json` у `fulfill()`, тоді заголовок `Content-Type: application/json` встановиться автоматично. Ручне встановлення `body` з JSON-рядком без правильного заголовка може зламати парсинг на клієнті.

4

Конфлікт патернів

Якщо кілька маршрутів можуть підходити до одного URL, спрацює той, який зареєстровано **першим**. Слідкуйте за порядком реєстрації та специфічністю URL-патернів, щоб уникнути несподіваної поведінки.

 **Порада:** увімкніть в Playwright режим `DEBUG=pw:api` для перегляду всіх перехоплених маршрутів і запитів. Це значно спрощує діагностику проблем із мокінгом.

Підсумок: коли що робити

Ми розглянули три основні стратегії мокінгу мережевих відповідей у Playwright. Кожна з них має своє призначення і оптимальний сценарій використання. Ця таблиця допоможе вам швидко обрати правильний підхід для конкретної задачі.

| Ситуація | Рішення | Приклад |
|--|--|---|
| Просто «фейкова» відповідь | <code>route.fulfill({ status, body })</code> або <code>route.fulfill({ status, json })</code> | Мок логіну, фіксовані тестові дані |
| Змінити частину оригінальної відповіді | <code>route.fetch()</code> → модифікувати → <code>route.fulfill()</code> | Додати поле, змінити заголовок HTML |
| Повністю замінити тіло, зберегти заголовки | <code>route.fulfill({ response, json })</code> або { response, body } | Тестування з повністю кастомним payload |



`page.route()`

Перехопити запит за URL-патерном

Обробка

`fulfill`, `fetch + modify`, або `continue`

Результат

Стабільний, швидкий і контролюваний тест

Мокінг – це не обман тестів, а **контроль над середовищем**. Він дозволяє перевіряти фронтенд ізольовано, передбачувано і з максимальною швидкістю. Опанувавши `page.route()` та `route.fulfill()`, ви отримуєте повну владу над мережевим рівнем ваших тестів.