Engineer

Наскрізний тест

🏃 тест усієї системи тим шляхом, яким іде користувач

Що це таке

Наскрізний тест перевіряє всю запущену систему так, як її бачить користувач. Він запускає застосунок зі справжнім фронтендом, бекендом, базою даних і конфігурацією, проходить увесь шлях — скажімо, реєстрацію чи оплату замовлення — і наприкінці перевіряє те, що побачила б людина. Англійською це end-to-end test, коротко E2E-тест, а саму практику називають наскрізним тестуванням.

У книжці *Software Engineering at Google* Adam Bender розкладає тести за двома осями. Розмір стосується ресурсів: малі тести виконуються в одному процесі, середні можуть використовувати кілька процесів на одній машині, а великі — охоплювати кілька машин. Обсяг показує, скільки коду тест зачіпає, і тести великого обсягу мають кілька назв, серед них функціональні, наскрізні та системні тести. Joseph Graves у розділі книжки про більші тести додає, що тести найбільшого обсягу зазвичай залучають кілька справжніх залежностей і менше тестових дублерів.

Ці справжні залежності дають достовірність: Graves визначає її як те, наскільки точно тест відображає справжню поведінку системи, яку він перевіряє. Модульний тест форми оформлення замовлення доводить, що код форми правильний в ізольованому середовищі; наскрізний тест тієї самої форми доводить, що розгорнута сторінка, маршрут за нею, правила ціноутворення й база даних узгоджуються між собою. Платити доводиться герметичністю: що більше середовище схоже на продакшн, то важче відгородити його від усього іншого, і, за словами Graves, ці дві властивості часто прямо суперечать одна одній.

Для вебзастосунку інтерфейсом користувача є браузер, тож саме там і стоїть наскрізний тест. Graves каже про це прямо: для кінцевих користувачів «публічний API — це і є сам застосунок». Playwright Test, на якому побудовано приклад нижче, описує себе як фреймворк для наскрізного тестування сучасних вебзастосунків. У ньому зібрано запускач тестів, перевірки, ізоляцію та паралельний запуск, і він керує Chromium, Firefox і WebKit — як із вікном браузера, так і без нього.

Застосуйте поняття до рішення

Застосуйте «Що таке наскрізний тест?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Одна команда веде ідею від ескізу до продукту, у який ваші клієнти вже можуть увійти, а не перекидає роботу між трьома підрядниками.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Збільшила дохід GIS-застосунку в 10 разів — 10x зростання виторгу завдяки картографічному продукту

Збільшила дохід GIS-застосунку в 10 разів Зв'язатися з нами →

Чому це важливо

Менші тести лишають прогалини, які може закрити лише тест зібраної системи, і Graves перелічує найпоширеніші. Тестовий дублер може розійтися з тим, що він замінює, і тест так і не дізнається, що справжня реалізація змінилася. Модульний тест покриває код усередині одного бінарника, а конфігурація розгортання, скрипти запуску й налаштування робочих екземплярів лишаються поза ним. Третя прогалина — проблеми, що з’являються лише під навантаженням, а четверта — поведінка, якої ніхто не передбачив; саме так більшість проблем і доходить до користувачів. Bender пише, що в Google великі тести здебільшого залишають для наскрізних тестів усієї системи, які перевіряють радше конфігурацію, ніж окремі фрагменти коду.

До того ж вони перевіряють зібраний інтерфейс, із яким взаємодіє користувач. Форма, що надсилає дані на маршрут, який бекенд щойно перейменував, або реліз, чия конфігурація вказує не на той сервіс, проходять усі модульні тести з обох боків — і ламаються, щойно ними спробує скористатися жива людина.

Ціна — швидкість і надійність. Graves зазначає, що великі тести в Google мають тайм-аут за замовчуванням 15 хвилин або годину і що вони можуть бути негерметичними, тобто ділити ресурси з іншими тестами й трафіком, а

отже, бути недетермінованими. Bender називає звичні джерела цієї недетермінованості: системний час, планування потоків, затримку мережі, а нижче — апаратні переривання та рушії рендерингу браузерів. Тому книжка й пропонує як приблизне правило піраміду: близько 80% модульних тестів вузького обсягу, 15% інтеграційних і 5% наскрізних, що перевіряють усю систему. Більші тести, пише Bender, слугують перевірками на здоровий глузд у міру розвитку продукту і не мають бути основним способом, яким команда ловить помилки; команди в Google часто запускають їх лише під час збирання й випуску, щоб вони не гальмували щоденну розробку.

Як це виглядає на практиці

Візьмімо інтернет-магазин, що продає чайник і запитує в API перевізника, скільки триватиме доставка. Специфікація Playwright для першої покупки гостя читається як сам цей шлях:

```
// e2e/checkout.spec.ts
import { test, expect } from "@playwright/test";

test.beforeEach(async ({ page }) => {
    // API перевізника належить іншим: відповідаємо за нього тут.
    await page.route("https://api.carrier.example/**", (route) =>
        route.fulfill({ status: 200, body: '{"days": 2}' }),
    );
});

test("гість може купити чайник", async ({ page }) => {
    await page.goto("/shop/teapot");
    await page.getByRole("button", { name: "Додати в кошик" }).click();
    await page.getByRole("link", { name: "Кошик" }).click();
    await expect(page.getByText("Доставка за 2 дні")).toBeVisible();
    await page.getByLabel("Електронна пошта").fill("buyer@example.com");
    await page.getByRole("button", { name: "Оформити замовлення" }).click();
    const thanks = page.getByRole("heading", { name: "Дякуємо за покупку" });
    await expect(thanks).toBeVisible();
    await page.getByRole("link", { name: "Переглянути замовлення" }).click();
    await expect(page.getByText("Чайник")).toBeVisible();
});
```

Кожен локатор називає те, що бачить користувач: кнопку «Додати в кошик», поле з підписом «Електронна пошта», заголовок «Дякуємо за покупку». Playwright радить насамперед локатори за роллю, а його локатори суворі: коли опису відповідає кілька елементів, дія кидає помилку, замість того щоб вгадувати. Перед кожним кліком чи заповненням поля Playwright чекає,

доки елемент стане придатним до дії, а перевірка на кшталт `toBeVisible()` повторюється, доки умова не справдиться, тож специфікація обходиться без пауз. Виклик `page.route` відповідає на запити браузера до перевізника фіксованим тілом, і тест ніколи не залежить від сервера, яким команда не керує; шаблон має охоплювати всю URL-адресу, а `**` проходить і крізь скісні риски. Маршрутизація перехоплює лише ті запити, які надсилає сама сторінка: якщо перевізника питає сервер магазину, браузер цього запиту не бачить, і тоді тестовий дублер має стояти на боці сервера. Кожен тест до того ж отримує свіжий контекст браузера, який Playwright порівнює з абсолютно новим профілем браузера, тож кукі й кошик одного тесту не просочуються в наступний.

Конфігурація запускає застосунок і повідомляє Playwright, де його шукати:

```
// playwright.config.ts
import { defineConfig } from "@playwright/test";

export default defineConfig({
    testDir: "e2e",
    retries: process.env.CI ? 2 : 0,
    use: {
        baseURL: "http://localhost:3000",
        trace: "on-first-retry",
    },
    webServer: {
        command: "npm run start",
        url: "http://localhost:3000",
        reuseExistingServer: !process.env.CI,
    },
});
```

Блок `webServer` виконує `npm run start` і чекає, доки URL почне відповідати, а вже тоді запускає перший тест; на ноутбуці він бере сервер, який уже працює, а в CI завжди запускає власний. `baseURL` дає змогу писати в специфікації `/shop/teapot` замість повної адреси. Команда `npx playwright test` запускає набір без вікна браузера, прапорець `--headed` показує вікно, а `npx playwright show-report` відкриває HTML-звіт. За замовчуванням Playwright не повторює тести, що впали; тут у CI дозволено два повтори, тест, який спершу впав, а потім пройшов, позначається як нестабільний (flaky), а не як пройдений, а під час першого повтору записується трасування, яке показує, що пішло інакше.

У більшому масштабі браузерний рівень — найменша частина роботи. Набір тестів для Go, SQL і браузера — робота 2025 року — налічує 981 тестову функцію на Go у 310 файлах, 543 специфікації Vitest і Playwright, зокрема 48 файлів наскрізних тестів і 30 браузерних специфікацій, а також 494 файли поведінкових тестів на SQL. Завдання полягало в тому, щоб перевіряти поведінку кожного рівня там, де вона справді живе, а не все ззовні через браузер, тож збережені функції перевіряють просто в базі даних, без трьох рівнів застосунку зверху. Сторінка не приховує ціни: набір працює повільно, за нього платить кожна зміна, і за такого обсягу він сам потребує супроводу.

Де виникають проблеми

Набір тестів — ріжок морозива. Так Bender називає набір, у якому багато наскрізних тестів і мало інтеграційних чи модульних; такі набори зазвичай повільні, ненадійні й незручні в роботі, а сама картина часто з'являється в проєктах, які поспіхом перевели з прототипу в продакшн. Graves додає: коли код важко покрити модульними тестами, наскрізні тести — єдині, які взагалі вдається написати, і за лічені дні проєкт уже має легасі-код. Кожен новий сервіс до того ж множить шляхи крізь систему, тож наскрізний набір ніколи не покриє їх усіх. Опускайте кожен випадок до найменшого тесту, який здатен його впіймати, лишайте браузеру ті сценарії, які може показати лише вся система, і будуйте піраміду з перших днів проєкту.

Селектори прив'язані до розмітки. Локатор на кшталт `button.buttonIcon.episode-actions-later` — власний приклад Playwright того, чого слід уникати, — ламається, щойно хтось перейменує клас, хоча для користувача все виглядає так само. Graves попереджає, що UI-тести мають репутацію ненадійних і дорогих, зокрема тому, що їх ламають зміни зовнішнього вигляду, тоді як поведінка лишається тією самою. Шукайте елементи за роллю, підписом або текстом — так, як їх знаходить користувач, — або за тестовим ідентифікатором, який належить команді. Якщо бекенд має публічний API, розріжте сценарій на межі між UI та API і проведіть решту наскрізного тесту через API.

Тест спить. Фіксована пауза закоротка в повільний день і задовга в усі інші. Graves зазначає, що тести, які покладаються на паузи й тайм-аути, починають падати всі разом, коли машини, на яких вони працюють, перевантажені, а повторні запуски лише додають навантаження. Найкращі практики Playwright показують тонший варіант тієї самої помилки: `expect(await page.getByText("welcome").isVisible()).toBe(true)` перевіряє один раз і йде далі, а

`await expect(page.getByText("welcome")).toBeVisible()` чекає на умову. Використовуйте web-first перевірки, а коли тест чекає на бекенд, опитуйте зміну стану з тайм-аутом.

Тести мають спільні дані. Один тест створює обліковий запис, на який розраховує наступний, або два паралельні воркери купують останній чайник на складі. Набір проходить в одному порядку й падає в іншому або лише тоді, коли тести йдуть паралельно. Playwright радить повністю ізолювати кожен тест — із власним сховищем, сесією, даними й кукі — і контролювати дані в будь-якій базі даних, з якою працюють тести. Створюйте все, що потрібно тесту, з унікальними назвами й ніколи не покладайтеся на те, що лишив попередній тест.

У тесті чужий сервер. Справжній платіжний провайдер, перевізник чи картографічний сервіс робить набір залежним від системи, якою команда не керує, а трафік до нього може коштувати грошей. Graves пише, що автоматичним тестам не рекомендують звертатися до справжнього стороннього API і що така залежність — важливий шов, по якому варто розділяти тести; сам Playwright радить тестувати лише те, що ви контролюєте. Відповідайте за третю сторону через `page.route`, коли запит надсилає браузер, як це робить специфікація вище, або через тестовий дублер, на який у конфігурації вказує сервер, коли запит надсилає він сам, і нехай саме ця залежність стане місцем, де тест закінчується.

Повтори приховують нестабільність. Повтор у CI робить червону збірку зеленою, а витрати тим часом накопичуються. Bender пророблює арифметику: якщо щодня запускати 10 000 тестів і кожен має 0,1% імовірності хибного падіння, команда отримує 10 нестабільних падінь на день, які треба розслідувати, а коли нестабільність наближається до 1%, тести починають втрачати цінність. Повторний запуск міняє машинний час на час інженерів і лише відкладає пошук справжньої причини. Документація pytest каже те саме з іншого боку: тести вищого рівня частіше бувають нестабільними, бо залежать від більшого обсягу стану. Вважайте кожен тест, який Playwright позначив як нестабільний, помилкою: відкривайте трасування з повтору й виправляйте тест або видаляйте його.

Що почитати далі

- Testing Overview — розділ Adam Bender у *Software Engineering at Google*: розмір і обсяг тестів, піраміда, ріжок морозива й ціна нестабільних тестів.
- Larger Testing — Joseph Graves про достовірність, прогалини, які лишають модульні тести, межі UI та сторонніх сервісів і боротьбу з нестабільністю.
- Installation — вступ від самого Playwright: встановлення Playwright Test, запуск прикладу, режим із вікном браузера та HTML-звіт.
- Best Practices — тестування видимої користувачеві поведінки, ізоляція тестів, сторонні сервери поза тестами, локатори й web-first перевірки.
- Retries — як Playwright повторює тести, що впали, і позначає кожен як passed, flaky або failed.
- Flaky tests — пояснення pytest, чому тести бувають нестабільними, від неконтрольованого стану системи й порядку тестів до потокобезпеки, і що з цим робити.

Словник

Обирайте за фактичними умовами

Опишіть робочий процес, наявні інтерфейси та користувачів, яким потрібна зміна. Ми можемо визначити перевірюваний результат і спосіб передачі, який дасть вашій команді змогу підтримувати його.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке наскрізний тест?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/end-to-end-test/