Engineer

End-to-end-test

🏃 en test, der går gennem hele systemet, som en bruger gør

Hvad det er

En end-to-end-test kontrollerer et helt, kørende system, sådan som brugeren møder det. Den starter applikationen med dens rigtige frontend, backend, database og konfiguration, gennemfører et forløb, for eksempel at oprette en konto eller betale for en ordre, og kontrollerer til sidst det, brugeren ville se på skærmen. I daglig tale forkortes den ofte til E2E-test, og selve praksissen kaldes end-to-end-testning.

I *Software Engineering at Google* sorterer Adam Bender tests langs to akser. Størrelse handler om ressourcer: små tests kører i én proces, mellemstore på én maskine, og store tests kører, hvor de vil. Omfang handler om, hvor meget kode en test sætter i gang, og tests med stort omfang går under flere navne, blandt andet funktionelle tests, end-to-end-tests og systemtests. Joseph Graves tilføjer i bogens kapitel om større tests, at tests med det største omfang typisk har flere rigtige afhængigheder og færre test doubles.

Det, de rigtige afhængigheder giver, er det, Graves kalder fidelity: hvor godt en test afspejler den virkelige adfærd i det system, der bliver testet. En enhedstest af en checkout-formular beviser, at formularens kode er rigtig i en sandkasse; en end-to-end-test af den samme formular beviser, at den udrullede side, ruten bag den, prisreglerne og databasen er enige med hinanden. Prisen betales i isolation: jo mere et miljø ligner produktion, desto sværere er det at forsegle det hermetisk mod alt andet, og Graves bemærker, at de to egenskaber ofte er i direkte konflikt.

For en webapplikation er brugerens grænseflade browseren, så det er dér, en end-to-end-test står. Graves siger det ligeud: for slutbrugerne er »det offentlige API selve applikationen«. Playwright Test, som eksemplet nedenfor bruger,

beskriver sig selv som et end-to-end-testframework til moderne webapps. Det samler testkøreren, assertions, isolation og parallel kørsel og styrer Chromium, Firefox og WebKit, enten headless eller med synligt vindue.

Brug begrebet i en beslutning

Brug »Hvad er en end-to-end-test?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Ét team, der tager en idé fra skitse til noget, jeres kunder kan logge ind på, i stedet for at kaste den mellem tre leverandører.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har ført en GIS-kortapp til 10x omsætningsvækst — 10x omsætningsvækst, drevet af kortproduktet

Har ført en GIS-kortapp til 10x omsætningsvækst Kontakt os →

Hvorfor det betyder noget

Mindre tests efterlader huller, som kun en test af det samlede system kan lukke, og Graves opregner de almindeligste. En test double kan glide væk fra det, den erstatter, og intet fortæller testen, at den rigtige implementering har ændret sig. En enhedstest dækker koden inde i én kørbar fil, mens udrulningskonfigurationen, opstartsscripts og indstillingerne for produktionsinstanserne ligger uden for den. Problemer, der kun viser sig under belastning, er det tredje hul, og det fjerde er adfærd, ingen havde forudset, og det er netop sådan, de fleste problemer når frem til brugerne. Bender skriver, at Google for det meste forbeholder store tests til end-to-end-tests af hele systemet, som handler mere om at validere konfiguration end om enkelte stykker kode.

De kontrollerer også brugerfladen i det system, brugeren møder. En formular, der sender til en rute, som backenden lige har omdøbt, eller en release, hvis konfiguration peger på den forkerte tjeneste, består hver eneste enhedstest på begge sider og fejler, første gang et rigtigt menneske prøver den.

Prisen er hastighed og pålidelighed. Graves bemærker, at Googles store tests har en standard-timeout på 15 minutter eller en time, og at de kan være ikke-hermetiske, fordi de deler ressourcer med andre tests og med trafik, og dermed ikke-deterministiske. Bender nævner de sædvanlige kilder til den ikke-determinisme: urets tid, trådplanlægning, netværkslatens og, længere nede, hardwareafbrydelser og browserens renderingsmotor. Derfor er bogens grove rettesnor en pyramide: omkring 80 % snævre enhedstests, 15 % integrationstests og 5 % end-to-end-tests, der validerer hele systemet. Større

tests fungerer ifølge Bender som fornuftstjek, efterhånden som produktet udvikler sig, og bør ikke være holdets vigtigste måde at fange fejl på, og hold hos Google kører dem ofte kun under build- og releaseprocessen, så de ikke sinker den daglige udvikling.

Sådan ser det ud i praksis

Tag en webshop, der sælger en tekande og spørger et fragtfirmas API om leveringstiden. En Playwright-spec for en gæsts første køb læser næsten som selve forløbet:

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

test.beforeEach(async ({ page }) => {
    // Fragtfirmaets API ejes af andre: svar på dets vegne her.
    await page.route("https://api.carrier.example/**", (route) =>
        route.fulfill({ status: 200, body: '{"days": 2}' }),
    );
});

test("en gæst kan købe en tekande", async ({ page }) => {
    await page.goto("/shop/teapot");
    await page.getByRole("button", { name: "Læg i kurven" }).click();
    await page.getByRole("link", { name: "Kurv" }).click();
    await expect(page.getByText("Levering om 2 dage")).toBeVisible();
    await page.getByLabel("E-mail").fill("buyer@example.com");
    await page.getByRole("button", { name: "Afgiv ordre" }).click();
    const thanks = page.getByRole("heading", { name: "Tak for din ordre" });
    await expect(thanks).toBeVisible();
    await page.getByRole("link", { name: "Se din ordre" }).click();
    await expect(page.getByText("Tekande")).toBeVisible();
});
```

Hver locator navngiver det, brugeren ser: en knap, der hedder Læg i kurven, et felt med etiketten E-mail og en overskrift, der siger Tak for din ordre. Playwright anbefaler at finde elementer efter rolle først, og dets locators er strikse: matcher mere end ét element, kaster handlingen en fejl i stedet for at gætte. Før hvert klik og hver udfyldning venter Playwright på, at elementet kan betjenes, og en assertion som `toBeVisible()` prøver igen, indtil betingelsen holder, så spec'en indeholder ingen faste ventepauser. Kaldet til `page.route` besvarer browserens forespørgsler til fragtfirmaet med et fast svar, så testen aldrig afhænger af en server, holdet ikke selv driver; mønstret skal matche hele URL'en, og `**` matcher på tværs af skråstreger. Routing når kun de forespørgsler, siden selv

sender: spørger butikkens server i stedet fragtfirmaet, ser browseren aldrig den forespørgsel, og så må en test double tage over på serverens side. Hver test får desuden en frisk browserkontekst, som Playwright sammenligner med en helt ny browserprofil, så den ene tests cookies og indkøbskurv ikke kan sive over i den næste.

Konfigurationen starter applikationen og fortæller Playwright, hvor den bor:

```
// 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,
    },
});
```

Blokken `webServer` kører `npm run start` og venter, til URL'en svarer, før den første test går i gang; på en bærbar genbruger den en server, der allerede kører, og på CI starter den altid sin egen. Med `baseURL` kan spec'en skrive `/shop/teapot` i stedet for en fuld adresse. `npx playwright test` kører suiten headless, `--headed` viser browservinduet, og `npx playwright show-report` åbner HTML-rapporten. Som standard prøver Playwright ikke en fejlet test igen; her tillader CI to forsøg mere, en test, der fejler og derefter består, bliver rapporteret som flaky og ikke som bestået, og ved første genforsøg optages et trace, der viser, hvad der var anderledes.

I større skala er browserlaget den mindste del af arbejdet. Den testsuite, der er bygget til Go, SQL og browseren, fra 2025, har 981 testfunktioner i Go fordelt på 310 filer, 543 Vitest- og Playwright-specs, heraf 48 end-to-end-filer og 30 browser-specs, og 494 adfærdstestfiler i SQL. Opgaven var at kontrollere hvert lags adfærd dér, hvor adfærden hører hjemme, frem for at kontrollere alt udefra gennem en browser, så en stored function bliver testet i selve databasen i stedet

for gennem de tre applikationslag, der ligger over den. Siden lægger ikke skjul på prisen: suiten er langsom, den lægger en ekstra byrde på hver ændring, og i den størrelse kræver den selv vedligeholdelse.

Hvor det går galt

Suiten er en isvaffel. Sådan kalder Bender en suite med mange end-to-end-tests og få integrations- eller enhedstests. Den slags suiter er typisk langsomme, upålidelige og svære at arbejde med, og mønstret opstår ofte i projekter, der er hastet fra prototype til produktion. Graves tilføjer, at når koden er svær at enhedsteste, er end-to-end-tests de eneste, nogen kan skrive, og så har projektet skabt legacy-kode på få dage. Hver ny tjeneste ganger desuden antallet af veje gennem systemet, så en end-to-end-suite aldrig kan dække dem alle. Skub hvert tilfælde ned til den mindste test, der kan fange det, gem browseren til de forløb, kun hele systemet kan vise, og byg hen imod pyramiden fra projektets første dage.

Selektorerne følger markuppen. En locator som `button.buttonIcon.episode-actions-later`, Playwrights eget eksempel på noget, man bør undgå, går i stykker, næste gang nogen omdøber en klasse, selvom brugeren ikke ser nogen forskel. Graves advarer om, at UI-tests har ry for at være upålidelige og dyre, blandt andet fordi ændringer i udseendet får dem til at fejle, mens adfærden er den samme. Find elementer efter rolle, etiket eller tekst, sådan som en bruger finder dem, eller efter et test-id, holdet selv ejer. Har backenden et offentligt API, så del forløbet ved grænsen mellem UI og API, og kør resten af end-to-end-testen gennem API'et.

Testen sover. En fast pause er for kort på en langsom dag og for lang alle andre dage. Graves bemærker, at tests, der bygger på sleeps og timeouts, alle begynder at fejle, når maskinerne, der kører dem, bliver overbelastede, og genkørslerne lægger endnu mere last på. Playwrights best practices viser en mere snedig udgave af den samme fejl: `expect(await page.getByText("welcome").isVisible()).toBe(true)` kontrollerer én gang og går videre, mens `await expect(page.getByText("welcome")).toBeVisible()` venter på betingelsen. Brug web-first assertions, og når en test venter på backenden, så spørg efter tilstandsændringen med jævne mellemrum og med en timeout.

Testene deler data. Én test opretter en konto, som den næste forventer, eller to parallelle workers køber den sidste tekande på lager. Suiten består i én rækkefølge og fejler i en anden, eller kun når den kører parallelt. Playwrights råd

er at holde hver test fuldstændig isoleret, med sin egen lagring, session, data og cookies, og at styre dataene i enhver database, testene bruger. Opret det, hver test har brug for, med unikke navne, og byg aldrig på det, en tidligere test efterlod.

En andens server er med i testen. En rigtig betalingsudbyder, et fragtfirma eller en tjeneste til kort og ruter gør suiten afhængig af et system, holdet ikke kan styre, og trafik dertil kan koste penge. Graves skriver, at det ikke anbefales at lade automatiske tests bruge et rigtigt tredjeparts-API, og at sådan en afhængighed er et vigtigt sted at dele testene; Playwrights eget råd er kun at teste det, man selv har kontrol over. Svar på tredjepartens vegne med `page.route`, når browseren sender kaldet, som spec'en ovenfor gør, eller med en test double, som serveren er sat op til at kalde, når serveren selv sender det, og lad testen slutte ved den afhængighed.

Genforsøg skjuler ustabiliteten. Genforsøg på CI gør en rød build grøn, og omkostningen vokser. Bender regner det igennem: ved 10.000 tests om dagen giver en risiko på 0,1 % for en falsk fejl i hver test 10 ustabile fejl om dagen, der skal undersøges, og når ustabiliteten nærmer sig 1 %, begynder testene at miste deres værdi. At køre testen igen bytter maskintid for udviklertid og udskyder kun den egentlige årsag. Dokumentationen til pytest siger det samme fra en anden vinkel: tests på et højere niveau er mere tilbøjelige til at være flaky, fordi de afhænger af mere tilstand. Behandl enhver test, Playwright melder som flaky, som en fejl, åbn tracet fra genforsøget, og ret testen, eller slet den.

Læs mere

- Testing Overview — Adam Benders kapitel i *Software Engineering at Google*: testers størrelse og omfang, pyramiden, isvaflen og prisen for ustabile tests.
- Larger Testing — Joseph Graves om fidelity, de huller, enhedstests efterlader, grænserne ved UI og tredjeparter, og om at drive ustabiliteten ud.
- Installation — Playwrights egen introduktion: installation af Playwright Test, kørsel af eksempeltesten, headed-tilstand og HTML-rapporten.
- Best Practices — test af synlig adfærd, isolerede tests, tredjepartsservere holdt udenfor, locators og web-first assertions.
- Retries — hvordan Playwright prøver fejlede tests igen og mærker hver test som passed, flaky eller failed.
- Flaky tests — pytests forklaring af, hvorfor tests bliver ustabile, fra ukontrolleret systemtilstand og testrækkefølge til trådsikkerhed, og hvad man kan gøre ved det.

Ordliste

Vælg ud fra jeres faktiske vilkår

Beskriv arbejdsgangen, de eksisterende grænseflader og de brugere, der har brug for ændringen. Vi kan definere en testbar leverance og en overdragelse, der gør teamet i stand til at vedligeholde løsningen.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er en end-to-end-test?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

welcome@engineer.company +45 50 10 61 84 København, Danmark

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/end-to-end-test/