Engineer

Ende-til-ende-test

🏃 en test som går gjennom hele systemet slik en bruker gjør

Hva det er

En ende-til-ende-test kontrollerer et helt kjørende system slik brukeren møter det. Den starter applikasjonen med ekte frontend, backend, database og konfigurasjon, går gjennom en hel reise, for eksempel registrering eller betaling av en bestilling, og sjekker til slutt det brukeren ville sett. Mange bruker den engelske skrivemåten end-to-end-test, ofte forkortet til E2E-test, og selve praksisen kalles ende-til-ende-testing.

I boka *Software Engineering at Google* sorterer Adam Bender tester langs to akser. Størrelse handler om ressurser: små tester kjører i én prosess, mellomstore på én maskin og store hvor som helst. Omfang handler om hvor mye kode en test berører, og tester med stort omfang går under flere navn, blant dem funksjonelle tester, ende-til-ende-tester og systemtester. Joseph Graves legger til i bokas kapittel om større tester at testene med størst omfang vanligvis involverer flere ekte avhengigheter og færre testdobler.

Det de ekte avhengighetene gir, er virkelighetstrohet, som Graves definerer som hvor godt en test gjenspeiler den virkelige oppførselen til systemet som testes. En enhetstest av et kasseskjema beviser at skjemaets kode er riktig i et avgrenset miljø; en ende-til-ende-test av det samme skjemaet beviser at den utrullede siden, ruten bak den, prisreglene og databasen stemmer overens med hverandre. Prisen betales i isolasjon: jo mer et miljø ligner produksjon, desto vanskeligere er det å forsegle det mot alt annet, og Graves påpeker at virkelighetstrohet og hermetisk isolasjon ofte står i direkte konflikt.

For en webapplikasjon er brukerens grensesnitt nettleseren, og det er der en ende-til-ende-test står. Graves sier det rett ut: for sluttbrukerne er «det offentlige API-et selve applikasjonen». Playwright Test, som eksempelet nedenfor bruker, beskriver seg selv som et rammeverk for ende-til-ende-testing

av moderne webapper. Det samler testkjører, resultatkontroller, isolasjon og parallell kjøring, og styrer Chromium, Firefox og WebKit, med eller uten synlig vindu.

Bruk begrepet i en beslutning

Bruk «Hva er en ende-til-ende-test?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Ett team som tar en idé fra skisse til noe kundene dine kan logge inn på, i stedet for at den sendes videre mellom tre leverandører.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Ledet en GIS-kartapp til 10 ganger mer omsetning — 10x inntektsvekst, ledet fra kartproduktet

Ledet en GIS-kartapp til 10 ganger mer omsetning Ta kontakt →

Hvorfor det er viktig

Mindre tester etterlater hull som bare en test av det sammensatte systemet kan tette, og Graves lister opp de vanligste. En testdobbel kan gli bort fra det den erstatter, uten at testen får vite at den ekte implementasjonen er endret. En enhetstest dekker koden inne i én binærfil, mens utrullingskonfigurasjonen, oppstartsskriptene og innstillingene til produksjonsinstansene ligger utenfor. Problemer som bare viser seg under last, er det tredje hullet, og det fjerde er oppførsel ingen forutså, som er nettopp slik de fleste problemer når brukerne. Bender skriver at Google stort sett reserverer store tester for ende-til-ende-tester av hele systemet, som validerer konfigurasjon mer enn enkeltdeler av koden.

De kontrollerer også grensesnittet i systemet brukeren møter. Et skjema som sender data til en rute backend nettopp har gitt nytt navn, eller en utgivelse der konfigurasjonen peker på feil tjeneste, består alle enhetstester på begge sider og feiler første gang et ekte menneske prøver det.

Prisen er fart og pålitelighet. Graves skriver at Googles store tester har et standard tidsavbrudd på 15 minutter eller en time, og at de kan være ikke-hermetiske, altså dele ressurser med andre tester og trafikk, og dermed ikke-deterministiske. Bender nevner de vanlige kildene til slik ikke-determinisme: klokketid, trådplanlegging, forsinkelser i nettverket og, lenger ned, maskinvareavbrudd og nettlesernes renderingsmotorer. Derfor er bokas tommelfingerregel en pyramide: omtrent 80 % enhetstester med smalt omfang, 15 % integrasjonstester og 5 % ende-til-ende-tester som validerer hele systemet. Større tester, skriver Bender, fungerer som fornuftssjekker etter hvert som

produktet utvikles, og bør ikke være teamets viktigste måte å fange feil på; team hos Google kjører dem ofte bare under bygg og utgivelse, slik at de ikke bremser den daglige utviklingen.

Slik ser det ut i praksis

Ta en nettbutikk som selger en tekanne og spør et fraktselskaps API om forventet leveringstid. En Playwright-spesifikasjon for en gjests første kjøp leses som selve reisen:

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

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

test("en gjest kan kjøpe en tekanne", async ({ page }) => {
    await page.goto("/shop/teapot");
    await page.getByRole("button", { name: "Legg i handlekurven" }).click();
    await page.getByRole("link", { name: "Handlekurv" }).click();
    await expect(page.getByText("Levering om 2 dager")).toBeVisible();
    await page.getByLabel("E-post").fill("buyer@example.com");
    await page.getByRole("button", { name: "Fullfør bestillingen" }).click();
    const thanks = page.getByRole("heading", { name: "Takk for bestillingen" });
    await expect(thanks).toBeVisible();
    await page.getByRole("link", { name: "Se bestillingen din" }).click();
    await expect(page.getByText("Tekanne")).toBeVisible();
});
```

Hver lokator navngir det brukeren ser: en knapp som heter «Legg i handlekurven», et felt merket «E-post», en overskrift som sier «Takk for bestillingen». Playwright anbefaler rollelokatorer først, og lokatorene er strenge: treffer de mer enn ett element, kaster handlingen en feil i stedet for å gjette. Før hvert klikk og hver utfylling venter Playwright til elementet er klart for handlingen, og en kontroll som `toBeVisible()` prøver på nytt til betingelsen holder, så spesifikasjonen klarer seg uten faste ventepauser. Kallet `page.route` svarer på nettleserens forespørsler til fraktselskapet med et fast svar, slik at testen aldri avhenger av en server teamet ikke drifter selv; mønsteret må treffe hele URL-en, og `**` treffer på tvers av skråstreker. Ruting når bare forespørslene siden selv sender: spør butikkens server fraktselskapet i stedet, ser

nettleseren aldri den forespørselen, og da må en testdobbel ta over på serverens side. Hver test får dessuten en fersk nettleserkontekst, som Playwright sammenligner med en splitter ny nettleserprofil, så informasjonskapslene og handlekurven fra én test ikke kan lekke over i den neste.

Konfigurasjonen starter applikasjonen og forteller Playwright hvor den finnes:

```
// 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` kjører `npm run start` og venter til URL-en svarer, før den første testen starter; på en bærbar maskin gjenbruker den en server som allerede kjører, og i CI starter den alltid sin egen. Med `baseURL` kan spesifikasjonen skrive `/shop/teapot` i stedet for en full adresse. `npx playwright test` kjører suiten uten vindu, `--headed` viser nettleservinduet, og `npx playwright show-report` åpner HTML-rapporten. Playwright prøver som standard ingen feilende test på nytt; her tillater CI to nye forsøk, en test som feiler og så består, rapporteres som ustabil (flaky) og ikke som bestått, og et spor tas opp ved første nye forsøk for å vise hva som var annerledes.

I større skala er nettleserlaget den minste delen av arbeidet. Testsuiten for Go, SQL og nettleseren, fra 2025, har 981 testfunksjoner i Go fordelt på 310 filer, 543 spesifikasjoner i Vitest og Playwright, blant dem 48 ende-til-ende-filer og 30 nettleserspesifikasjoner, og 494 atferdstestfiler i SQL. Oppgaven var å kontrollere oppførselen til hvert lag der den faktisk hører hjemme, og ikke alt utenfra gjennom en nettleser, så en lagret funksjon testes rett i databasen

og ikke via tre applikasjonslag oppå den. Siden legger ikke skjul på kostnaden: suiten er treg, hver endring betaler for den, og i denne størrelsen trenger den eget vedlikehold.

Hvor det går galt

Suiten er en iskremkjeks. Bender kaller det en iskremkjeks når en suite har mange ende-til-ende-tester og få integrasjons- eller enhetstester; slike suiter blir gjerne trege, upålitelige og tunge å jobbe med, og mønsteret dukker ofte opp i prosjekter som har hastet fra prototype til produksjon. Graves legger til at når koden er vanskelig å enhetsteste, er ende-til-ende-tester de eneste noen får skrevet, og da har prosjektet produsert legacy-kode i løpet av få dager. Hver ny tjeneste mangedobler dessuten veiene gjennom systemet, så en ende-til-ende-suite kan aldri dekke dem alle. Skyv hvert tilfelle ned til den minste testen som kan fange det, spar nettleseren til reisene bare hele systemet kan vise, og bygg mot pyramiden fra prosjektets første dager.

Selektorene er bundet til markupen. En lokator som `button.buttonIcon.episode-actions-later`, Playwrights eget eksempel på hva man bør unngå, brekker neste gang noen gir en klasse nytt navn, selv om brukeren ikke ser noen forskjell. Graves advarer om at UI-tester har ry for å være upålitelige og dyre, delvis fordi endringer i utseendet knekker dem mens oppførselen er den samme. Finn elementer etter rolle, etikett eller tekst, slik en bruker finner dem, eller etter en test-ID teamet eier. Der backend har et offentlig API, deler du reisen ved grensen mellom UI og API og kjører resten av ende-til-ende-testen gjennom API-et.

Testen sover. En fast ventetid er for kort en treg dag og for lang alle andre dager. Graves påpeker at tester som lener seg på pauser og tidsavbrudd, begynner å feile samtidig når maskinene som kjører dem, er overbelastet, og de nye kjøringene legger på enda mer last. Playwrights beste praksis viser en mer subtil variant av den samme feilen: `expect(await page.getByText("welcome").isVisible()).toBe(true)` sjekker én gang og går videre, mens `await expect(page.getByText("welcome")).toBeVisible()` venter på betingelsen. Bruk web-first-kontroller, og når en test venter på backend, spør etter tilstandsendringen i en løkke med et tidsavbrudd.

Testene deler data. Én test oppretter en konto som den neste regner med, eller to parallelle arbeidere kjøper den siste tekannen på lager. Suiten består i én rekkefølge og feiler i en annen, eller bare når den kjører parallelt. Playwright anbefaler å holde hver test helt isolert, med egen lagring, økt, data og

informasjonskapsler, og å kontrollere dataene i enhver database testene bruker. Opprett det hver test trenger, med unike navn, og stol aldri på det en tidligere test har etterlatt seg.

En annens server er med i testen. En ekte betalingsleverandør, et fraktselskap eller en karttjeneste gjør suiten avhengig av et system teamet ikke kontrollerer, og trafikken dit kan koste penger. Graves skriver at det ikke anbefales at automatiserte tester bruker et ekte tredjeparts-API, og at en slik avhengighet er en viktig søm å dele testene ved; Playwrights eget råd er å teste bare det du kontrollerer. Svar på tredjepartens vegne med `page.route` når nettleseren gjør kallet, slik spesifikasjonen over gjør, eller med en testdobbel serveren er satt opp til å kalle når serveren gjør kallet selv, og la den avhengigheten være der testen slutter.

Nye forsøk skjuler ustabiliteten. Å kjøre på nytt i CI gjør et rødt bygg grønt, og kostnaden hoper seg opp. Bender regner det ut: med 10 000 tester om dagen og 0,1 % sjanse for en falsk feil i hver får teamet 10 ustabile feil om dagen å undersøke, og når ustabiliteten nærmer seg 1 %, begynner testene å miste verdi. Nye kjøringer bytter maskintid mot ingeniørtid og utsetter bare rotårsaken. Dokumentasjonen til pytest sier det samme fra en annen kant: tester på høyere nivå er oftere ustabile fordi de avhenger av mer tilstand. Behandle hver test Playwright melder som ustabil, som en feil, åpne sporet fra det nye forsøket, og rett testen eller slett den.

Les mer

- Testing Overview — Adam Benders kapittel i *Software Engineering at Google*: teststørrelse og omfang, pyramiden, iskremkjeksen og hva ustabile tester koster.
- Larger Testing — Joseph Graves om virkelighetstrohet, hullene enhetstester etterlater, grensene mot UI og tredjeparter, og hvordan man får bukt med ustabilitet.
- Installation — Playwrights egen introduksjon: installere Playwright Test, kjøre eksempeltesten, kjøring med synlig vindu og HTML-rapporten.
- Best Practices — teste oppførsel brukeren ser, isolere tester, holde tredjepartsservere utenfor, lokatorer og web-first-kontroller.
- Retries — hvordan Playwright prøver feilende tester på nytt og merker hver av dem som passed, flaky eller failed.
- Flaky tests — pytests forklaring på hvorfor tester blir ustabile, fra ukontrollert systemtilstand og testrekkefølge til trådsikkerhet, og hva man gjør med det.

Ordliste

Velg ut fra deres faktiske rammer

Beskriv arbeidsflyten, eksisterende grensesnitt og brukerne som trenger endringen. Vi kan definere en testbar leveranse og en overlevering som gjør teamet i stand til å vedlikeholde løsningen.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er en ende-til-ende-test?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/end-to-end-test/