Engineer

End-to-end-test

ett test som går igenom hela systemet som en användare gör

Vad det är

Ett end-to-end-test kontrollerar ett helt körande system så som användaren möter det. Det startar applikationen med riktig frontend, backend, databas och konfiguration, går igenom en hel resa, till exempel en registrering eller betalningen av en beställning, och kontrollerar till sist det som användaren skulle se. I vardagligt tal förkortas det ofta till E2E-test, och arbetssättet kallas end-to-end-testning.

I boken *Software Engineering at Google* sorterar Adam Bender tester längs två axlar. Storlek handlar om resurser: små tester körs i en enda process, medelstora på en enda maskin och stora var som helst. Omfattning handlar om hur mycket kod ett test berör, och tester med stor omfattning går under flera namn, bland dem funktionstester, end-to-end-tester och systemtester. Joseph Graves tillägger i bokens kapitel om större tester att testerna med störst omfattning vanligtvis involverar flera riktiga beroenden och färre testdubblar.

Det som de riktiga beroendena ger är verklighetstrohet, som Graves definierar som hur väl ett test återspeglar det verkliga beteendet hos systemet som testas. Ett enhetstest av ett kassaformulär bevisar att formulärets kod är rätt i en avgränsad miljö; ett end-to-end-test av samma formulär bevisar att den driftsatta sidan, routen bakom den, prisreglerna och databasen stämmer överens med varandra. Priset betalas i isolering: ju mer en miljö liknar produktion, desto svårare är det att försegla den mot allt annat, och Graves påpekar att verklighetstrohet och hermetiska tester ofta står i direkt konflikt.

För en webbapplikation är användarens gränssnitt webbläsaren, och det är där ett end-to-end-test står. Graves säger det rakt ut: för slutanvändare är ”det publika API:et själva applikationen”. Playwright Test, som exemplet nedan

använder, beskriver sig själv som ett ramverk för end-to-end-testning av moderna webbappar. Det samlar testkörare, assertions, isolering och parallell körning, och styr Chromium, Firefox och WebKit, med eller utan synligt fönster.

Använd begreppet i ett beslut

Använd ”Vad är ett end-to-end-test?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Ett team som tar en idé från en skiss till något dina kunder kan logga in i, i stället för att lämna den vidare mellan tre leverantörer.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Ledde GIS-kartappen till 10 gånger högre intäkter — 10x intäktstillväxt, med kartprodukten i spetsen

Ledde GIS-kartappen till 10 gånger högre intäkter Hör av er →

Varför det spelar roll

Mindre tester lämnar luckor som bara ett test av det sammansatta systemet kan täppa till, och Graves räknar upp de vanligaste. En testdubbel kan glida ifrån det den ersätter, utan att testet får veta att den riktiga implementationen har ändrats. Ett enhetstest täcker koden inuti en enda binärfil, medan driftsättningens konfiguration, startskripten och inställningarna för produktionsinstanserna ligger utanför. Problem som bara visar sig under last är den tredje luckan. Den fjärde är beteende som ingen förutsåg, och det är så de flesta problem når användarna. Bender skriver att Google mest reserverar stora tester för end-to-end-tester av hela systemet, som validerar konfiguration mer än enskilda kodbitar.

De kontrollerar också gränssnittet i det system som användaren möter. Ett formulär som skickar data till en route som backend just har döpt om, eller en release vars konfiguration pekar på fel tjänst, klarar alla enhetstester på båda sidor och fallerar första gången en riktig människa provar.

Priset är hastighet och tillförlitlighet. Graves skriver att Googles stora tester har en standardtimeout på 15 minuter eller en timme, och att de kan vara icke-hermetiska, alltså dela resurser med andra tester och trafik, och därmed icke-deterministiska. Bender nämner de vanliga källorna till sådan icke-determinism: klocktid, trådschemaläggning, latens i nätverket och, längre ned, hårdvaruavbrott och webbläsarnas renderingsmotorer. Därför är bokens tumregel en pyramid: ungefär 80 % enhetstester med smal omfattning, 15 % integrationstester och 5 % end-to-end-tester som validerar hela systemet. Större tester, skriver Bender,

fungerar som rimlighetskontroller allteftersom produkten utvecklas och bör inte vara teamets främsta sätt att hitta buggar; team på Google kör dem ofta bara under bygge och release, så att de inte bromsar det dagliga utvecklingsarbetet.

Så ser det ut i praktiken

Ta en webbutik som säljer en tekanna och frågar ett fraktbolags API om beräknad leveranstid. En Playwright-specifikation för en gästs första köp läses som själva resan:

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

test.beforeEach(async ({ page }) => {
    // Fraktbolagets API ägs av andra: svara å dess vägnar här.
    await page.route("https://api.carrier.example/**", (route) =>
        route.fulfill({ status: 200, body: '{"days": 2}' }),
    );
});

test("en gäst kan köpa en tekanna", async ({ page }) => {
    await page.goto("/shop/teapot");
    await page.getByRole("button", { name: "Lägg i varukorgen" }).click();
    await page.getByRole("link", { name: "Varukorg" }).click();
    await expect(page.getByText("Leverans om 2 dagar")).toBeVisible();
    await page.getByLabel("E-post").fill("buyer@example.com");
    await page.getByRole("button", { name: "Slutför beställningen" }).click();
    const thanks = page.getByRole("heading", { name: "Tack för ditt köp" });
    await expect(thanks).toBeVisible();
    await page.getByRole("link", { name: "Visa din beställning" }).click();
    await expect(page.getByText("Tekanna")).toBeVisible();
});
```

Varje locator namnger det som användaren ser: en knapp som heter ”Lägg i varukorgen”, ett fält märkt ”E-post”, en rubrik som säger ”Tack för ditt köp”. Playwright rekommenderar rollbaserade locatorer i första hand, och locatorerna är strikta: matchar de mer än ett element kastar åtgärden ett fel i stället för att gissa. Före varje klick och varje ifyllning väntar Playwright tills elementet är redo för åtgärden, och en assertion som `toBeVisible()` försöker igen tills villkoret är uppfyllt, så specifikationen klarar sig utan fasta väntetider. Anropet `page.route` besvarar webbläsarens förfrågningar till fraktbolaget med ett fast svar, så att testet aldrig beror på en server som teamet inte själv driver; mönstret måste matcha hela URL:en, och `**` matchar över snedstreck. Routningen når bara de förfrågningar som sidan själv skickar: frågar butikens server fraktbolaget i stället,

ser webbläsaren aldrig den förfrågan, och då måste en testdubbel ta över på serverns sida. Varje test får dessutom en ny webbläsarkontext, som Playwright liknar vid en helt ny webbläsarprofil, så kakorna och varukorgen från ett test inte kan läcka in i nästa.

Konfigurationen startar applikationen och talar om för Playwright var den finns:

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

Blocket `webServer` kör `npm run start` och väntar tills URL:en svarar innan det första testet startar; på en bärbar dator återanvänder det en server som redan körs, och i CI startar det alltid sin egen. Med `baseURL` kan specifikationen skriva `/shop/teapot` i stället för en fullständig adress. `npx playwright test` kör sviten utan fönster, `--headed` visar webbläsarfönstret och `npx playwright show-report` öppnar HTML-rapporten. Som standard körs ett test som fallerar inte om; här tillåter CI två omförsök, ett test som fallerar och sedan går igenom rapporteras som instabilt (flaky) och inte som godkänt, och ett spår (trace) spelas in vid första omförsöket för att visa vad som skilde sig.

I större skala är webbläsarlagret den minsta delen av arbetet. Testsviten för Go, SQL och webbläsaren, från 2025, har 981 testfunktioner i Go fördelade på 310 filer, 543 testspecifikationer i Vitest och Playwright, däribland 48 end-to-end-filer och 30 webbläsarspecifikationer, och 494 beteendetestfiler i SQL. Målet var att pröva varje lager där dess beteende finns, inte allt utifrån via en webbläsare, så en lagrad funktion testas direkt i databasen och inte genom tre applikationslager ovanför. Sidan sticker inte under stol med kostnaden: sviten tar tid, varje ändring får betala för den, och vid den storleken kräver den ett eget underhåll.

Var det går fel

Sviten är en glasstrut. Bender kallar det en glasstrut när en svit har många end-to-end-tester och få integrations- eller enhetstester; sådana sviter blir ofta långsamma, opålitliga och svåra att arbeta med, och mönstret dyker gärna upp i projekt som har forcerats från prototyp till produktion. Graves tillägger att när koden är svår att enhetstesta blir end-to-end-tester de enda som någon kan skriva, och då har projektet producerat legacy-kod inom några dagar. Varje ny tjänst mångdubblar dessutom vägarna genom systemet, så en end-to-end-svit kan aldrig täcka dem alla. Skjut ned varje fall till det minsta test som kan fånga det, spara webbläsaren till de resor som bara hela systemet kan visa, och bygg mot pyramiden från projektets första dagar.

Selektorerna följer markupen. En locator som `button.buttonIcon.episode-actions-later`, Playwrights eget exempel på vad man bör undvika, går sönder nästa gång någon byter namn på en klass, fast användaren inte ser någon skillnad. Graves varnar för att UI-tester har rykte om sig att vara opålitliga och dyra, delvis för att ändringar i utseendet får dem att fallera medan beteendet är detsamma. Hitta element efter roll, etikett eller text, så som en användare hittar dem, eller efter ett test-id som teamet äger. Där backend har ett publikt API delar du resan vid gränsen mellan UI och API och kör resten av end-to-end-testet genom API:et.

Testet sover. En fast väntetid är för kort en långsam dag och för lång alla andra dagar. Graves påpekar att tester som förlitar sig på pauser och timeouter alla börjar fallera när maskinerna som kör dem är överbelastade, och omkörningarna lägger på ytterligare last. Playwrights bästa praxis visar en subtilare variant av samma misstag: `expect(await page.getByText("welcome").isVisible()).toBe(true)` kontrollerar en gång och går vidare, medan `await expect(page.getByText("welcome")).toBeVisible()` väntar på villkoret. Använd web-first-assertions, och när ett test väntar på backend, fråga efter tillståndsändringen gång på gång, med en timeout.

Testerna delar data. Ett test skapar ett konto som nästa test räknar med, eller två parallella arbetare köper den sista tekannan i lager. Sviten går igenom i en ordning och fallerar i en annan, eller bara när den körs parallellt. Playwright rekommenderar att varje test hålls helt isolerat, med egen lagring, session, data och egna kakor, och att man kontrollerar datan i alla databaser som testerna använder. Skapa det som varje test behöver, med unika namn, och lita aldrig på det som ett tidigare test lämnat efter sig.

Någon annans server är med i testet. En riktig betalleverantör, ett fraktbolag eller en karttjänst gör sviten beroende av ett system som teamet inte kontrollerar, och trafiken dit kan kosta pengar. Graves skriver att det inte rekommenderas att automatiska tester använder ett riktigt tredjeparts-API, och att ett sådant beroende är en viktig söm där testerna kan delas; Playwrights eget råd är att bara testa det man kontrollerar. Svara å tredjepartens vägnar med `page.route` när webbläsaren gör anropet, så som specifikationen ovan gör, eller med en testdubbel som servern är konfigurerad att anropa när servern gör anropet själv, och låt det beroendet vara där testet slutar.

Omförsök döljer instabiliteten. Att köra om i CI gör ett rött bygge grönt, och kostnaden växer. Bender räknar på det: med 10 000 tester om dagen och 0,1 % risk för ett falskt fel i vart och ett får teamet 10 instabila fel om dagen att utreda, och när instabiliteten närmar sig 1 % börjar testerna förlora sitt värde. Omkörningar byter maskintid mot ingenjörstid och skjuter bara upp grundorsaken. Dokumentationen för pytest säger samma sak från ett annat håll: tester på högre nivå är oftare instabila eftersom de beror på mer tillstånd. Behandla varje test som Playwright rapporterar som instabilt som en bugg, öppna spåret från omförsöket, och rätta testet eller ta bort det.

Läs mer

- Testing Overview — Adam Benders kapitel i *Software Engineering at Google*: teststorlek och omfattning, pyramiden, glasstruten och vad instabila tester kostar.
- Larger Testing — Joseph Graves om verklighetstrohet, luckorna som enhetstester lämnar, gränserna mot UI och tredje part, och hur man får bukt med instabilitet.
- Installation — Playwrights egen introduktion: installera Playwright Test, köra exempeltestet, köra med synligt fönster och HTML-rapporten.
- Best Practices — testa beteende som användaren ser, isolera tester, hålla tredjepartsservrar utanför, locatorer och web-first-assertions.
- Retries — hur Playwright kör om fallerande tester och märker vart och ett som passed, flaky eller failed.
- Flaky tests — pytests förklaring till varför tester blir instabila, från okontrollerat systemtillstånd och testordning till trådsäkerhet, och vad man gör åt det.

Ordlista

Välj utifrån era faktiska förutsättningar

Beskriv arbetsflödet, befintliga gränssnitt och användarna som behöver förändringen. Vi kan definiera en testbar leverans och en överlämning som gör att teamet kan underhålla lösningen.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är ett end-to-end-test?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/end-to-end-test/