**Engineer**

# Playwright

verktyget som styr en webbläsare i tester

## Vad det är

Playwright är ett ramverk med öppen källkod från Microsoft för att automatisera webbläsare, med ett och samma programmeringsgränssnitt för Chromium, Firefox och WebKit. Playwright Test, publicerat på npm som `@playwright/test`, är testköraren som är byggd ovanpå det, och det är den som de flesta menar när de använder namnet. Samma projekt levererar också det rena biblioteket för skript som inte är tester, ett kommandoradsverktyg och en MCP-server riktad till AI-agenter, och ett tillägg för Visual Studio Code, allt under Apache License 2.0.

Utöver JavaScript och TypeScript går samma styrning av webbläsaren att nå från Python, där projektet rekommenderar sitt plugin för pytest, från Java, med valfritt testramverk som JUnit, och från .NET, med basklasser för MSTest, NUnit och xUnit.

Kommandot `npx playwright install` laddar ned projektets egna byggen av de tre motorerna, och eftersom varje release flyttar fram de versionerna kan kommandot behöva köras igen efter en uppgradering. Playwright kan också styra Google Chrome och Microsoft Edge där de redan finns på maskinen. Som standard körs testerna headless, utan synligt fönster. Exemplet i installationsguiden kör tester parallellt över alla tre motorerna genom konfigurerade webbläsarprojekt; utan dessa projekt är Chromium testkörarens standardwebbläsare. Varje test får en ny webbläsarkontext, motsvarigheten till en ny webbläsarprofil, så att kakor och lagring från ett test aldrig följer med till nästa.

Ett test läses som de steg en person skulle ta:

```
import { test, expect } from "@playwright/test";

test("signing in opens the inbox", async ({ page }) => {
    await page.goto("https://example.com/sign-in");
    await page.getByLabel("Email").fill("reader@example.com");
    await page.getByRole("button", { name: "Sign in" }).click();
    await expect(page.getByRole("heading", { name: "Inbox" })).toBeVisible();
});
```

Två mekanismer bär det mesta av tyngden. Före en handling kör Playwright kontroller av att elementet går att agera på, i dokumentationen kallade actionability checks, och väntar tills alla har gått igenom. För ett klick måste locatorn peka ut exakt ett element, och det elementet måste vara synligt, stabilt, vilket betyder samma omgivande ruta under minst två animationsbildrutor i följd, kunna ta emot klicket i stället för att ligga under ett annat element, och aktiverat. Om kontrollerna inte går igenom inom tidsgränsen slutar handlingen i ett `TimeoutError`. Påståenden, assertions, som skrivs som `await expect(locator)` beter sig på samma sätt: de hämtar elementet om och om igen tills villkoret är uppfyllt eller tidsgränsen för påståendet, fem sekunder som standard, löper ut.

Använd begreppet i ett beslut

Använd ”Vad är Playwright?” 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

Beslutet som Playwright tjänar är vilket beteende som måste kontrolleras i en riktig webbläsare, och hur de kontrollerna blir tillräckligt pålitliga för att ett rött bygge ska bli trott. Vissa fel uppstår först när sidan är ihopsatt: en dialog som täcker knappen den skulle visa, en layout som döljer ett fält vid en viss bredd, ett skript som fungerar i en motor och kastar ett fel i en annan. Tester som kör kodbitar isolerat ser dem aldrig.

Webbläsartester blir opålitliga, flaky, när testet och sidan tävlar med varandra. Sidan hämtar data, animerar och renderar om, och ett test som klickar i samma ögonblick som raden körs klickar ibland för tidigt. Det vanliga botemedlet var en

fast paus, för kort på en långsam maskin och bortkastad tid på en snabb. Automatisk väntan och påståenden som försöker igen ersätter pausen med det villkor som testet faktiskt beror på. Guiden om assertions säger det rakt ut: webbsidor visar information asynkront, och påståenden som bara kontrollerar en gång leder till flaky tester.

Det andra valet är hur ett test hittar saker. Guiden om locators rekommenderar i första hand attribut som användaren möter, och rollocators framför allt, eftersom `getByRole` med ett tillgängligt namn är det närmaste ett test kommer hur användare och hjälpmedel uppfattar sidan. Det har en bieffekt som är värd att ha: ett test som klickar på knappen med namnet ”Sign in” går sönder om knappen förlorar sitt namn, vilket är samma defekt som den som använder skärmläsare skulle möta. Guiden säger också att rollocators varken ersätter granskningar av tillgänglighet eller tester av överensstämmelse. Test-id är reservlösningen. Guiden kallar dem det mest motståndskraftiga alternativet och samtidigt något som användaren aldrig ser, så en svit som hittar allt via id bevisar mindre om vad en besökare upplever. Selektorer i CSS och XPath avråder guiden från, eftersom DOM ändras ofta.

Kostnaden är tid. Att starta en webbläsare och läsa in en sida gör webbläsartester till den långsammaste sorten ett projekt har. När ett av dem misslyckas på en byggserver hjälper Trace Viewer till att undersöka felet. Det dokumenterade konfigurationsexemplet spelar in en trace vid första omförsöket av ett misslyckat test, med två omförsök på CI och noll lokalt; testkörarens inbyggda standardinställningar ger varken traceinspelning eller omförsök. Tracen stegar igenom varje handling med sidan före och efter den, nätverksanropen och konsolen.

## Så ser det ut i praktiken

En plattform här har sin affärslogik i databasen, och arbetet var att bygga en skiktad testsvit över fyra lager. Den uttalade uppgiften var att verifiera varje lagers beteende där beteendet faktiskt finns, ”i stället för att allt verifierades utifrån genom en webbläsare”. Go-API:t har 981 testfunktioner. Databasen har 494 beteendetestfiler, så att en lagrad funktion testas i databasen i stället för genom tre lager applikation ovanför. Frontenden har 543 testspecifikationer i Vitest och Playwright, däribland 48 end-to-end-filer och 30 webbläsarspecifikationer, och sidan ger ett och samma tal för båda verktygen. Hela sviten körs som 10 jobb i ett arbetsflöde på 612 rader, och sidan är rak om priset: sviten är långsam, den belastar varje ändring, och i den här storleken behöver den eget underhåll.

Den här webbplatsen styr sina egna webbläsarkontroller med ett annat bibliotek, Puppeteer, och kolofonen visar samma avvägning. De kontrollerna täcker det som en validator inte kan göra, som uppmätt kontrast, fokusordning, målstorlek, innehåll som sticker utanför visningsområdet och sidbrytningar vid utskrift. Spärren kör alla utom kontrollen av sådant innehåll före varje push; den kontrollen kör en människa vid behov, precis som den visuella regressionsjämförelsen pixel för pixel.

## Var det går fel

De flesta problem med Playwright kommer från tester som kontrollerar mindre än deras namn påstår.

**En fast paus står i stället för ett villkor.** Ett test väntar två sekunder och läser sedan sidan, eller läser ett värde en gång och jämför det med ett allmänt påstående. På en upptagen byggserver är två sekunder ibland för kort. Gör påståendet om det en besökare skulle se, med ett så kallat web-first-påstående som `toHaveText`, och låt det försöka igen.

**Element hittas via sin märkning.** En selektor genom tre nästlade `div`-element eller en stilklass går sönder vid nästa omdesign medan sidan fortfarande fungerar. Hitta elementet efter roll och tillgängligt namn, eller efter etikett eller text, och lägg till ett test-id bara när en användare annars skulle sakna något att känna igen elementet på.

**En godkänd rollocator tas för en tillgänglighetskontroll.** `getByRole` bevisar att ett element har den roll och det namn som testet bad om. Om kontrast, fokusordning eller resten av sidan säger det ingenting. För det behövs en regelmotor som axe, körd från testet, så som guiden från Playwright själva gör genom `@axe-core/playwright`. Skanningen täcker sidan i det tillstånd den har när den anropas, så öppna menyn eller dialogen först.

**Ett klick tvingas igenom.** Alternativet `force` stänger av de kontroller som inte är nödvändiga, och för ett klick slutar Playwright kontrollera att målet faktiskt tar emot händelsen. Ett test som behöver det går ofta runt något som en besökare också skulle möta, som en banner över knappen. Behandla det ursprungliga felet som ett fynd, och stäng antingen bannern så som en besökare skulle göra eller rätta överlappningen.

**Ett flaky test körs om tills det går igenom.** Med omförsök påslagna markeras ett test som fallerar vid första körningen och går igenom vid ett omförsök som flaky i stället för misslyckat. Ett test som är flaky vecka efter vecka tävlar antingen med sidan eller fångar ett verkligt fel som kommer och går, och den trace som spelades in vid dess första omförsök är stället att börja leta.

**Allt kontrolleras genom webbläsaren.** En regel om priser eller behörigheter som testas genom att klicka sig igenom tre skärmar är långsam och går sönder av skäl som saknar samband med regeln. Testa regeln där den finns, och behåll webbläsartesterna för det som bara en webbläsare kan visa: resan, layouten och skillnaderna mellan motorerna.

## Läs mer

- Installation — att installera Playwright Test, köra exempeltesterna, HTML-rapporten och UI Mode.
- Locators — den rekommenderade ordningen för att hitta element, från roll och etikett till test-id.
- Auto-waiting — kontrollerna bakom varje handling och vad `force` stänger av.
- Assertions — påståenden som försöker igen och sådana som inte gör det, standardtidsgränsen, mjuka påståenden och polling.
- Trace viewer — att spela in en trace vid omförsök och stega igenom ett misslyckat test handling för handling.
- Browsers — att installera motorerna, köra märkesversionerna av Chrome och Edge och de två headless-lägena.

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 Playwright?” 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/playwright/**