**Engineer**

# Playwright

Werkzeug, das einen Browser in Tests steuert

## Was es ist

Playwright ist ein quelloffenes Framework von Microsoft zur Automatisierung von Webbrowsern, mit einer Programmierschnittstelle für Chromium, Firefox und WebKit. Playwright Test, auf npm als `@playwright/test` veröffentlicht, ist der darauf aufgebaute Testrunner. Ihn meinen die meisten Menschen, wenn sie den Namen verwenden. Dasselbe Projekt liefert die reine Bibliothek für Skripte, die keine Tests sind, ein Kommandozeilenwerkzeug, einen an KI-Agenten gerichteten MCP-Server und eine Erweiterung für Visual Studio Code. Alles steht unter der Apache License 2.0.

Neben JavaScript und TypeScript ist dieselbe Browsersteuerung auch in Python verfügbar, wo das Projekt sein pytest-Plugin empfiehlt, in Java mit beliebigen Testframeworks wie JUnit und in .NET mit Basisklassen für MSTest, NUnit und xUnit.

`npx playwright install` lädt die eigenen Builds der drei Engines herunter. Da jede Veröffentlichung auf neuere Versionen dieser Engines wechselt, muss der Befehl nach einem Upgrade möglicherweise erneut ausgeführt werden. Playwright kann auch Google Chrome und Microsoft Edge steuern, wenn sie bereits auf dem Rechner installiert sind. Tests laufen standardmäßig headless, also ohne sichtbares Fenster. Das Beispiel der Installationsanleitung führt Tests über konfigurierte Browser-Projekte parallel in allen drei Engines aus. Ohne diese Projekte ist Chromium der Standardbrowser des Runners. Jeder Test erhält einen neuen Browserkontext, der einem neuen Browserprofil entspricht. Cookies und gespeicherte Browserdaten eines Tests gelangen dadurch nicht in den nächsten.

Ein Test liest sich wie die Schritte, die ein Mensch ausführen würde:

```
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();
});
```

Zwei Mechanismen leisten den größten Teil der Arbeit. Vor einer Aktion führt Playwright Prüfungen der Ausführbarkeit durch und wartet, bis alle bestanden sind. Für einen Klick muss der Locator genau ein Element liefern. Dieses muss sichtbar und stabil sein, was bedeutet, dass sein Begrenzungsrechteck in mindestens zwei aufeinanderfolgenden Animationsframes gleich bleibt. Es muss den Klick empfangen können, statt unter einem anderen Element zu liegen, und aktiviert sein. Bestehen die Prüfungen nicht innerhalb des Zeitlimits, endet die Aktion mit einem `TimeoutError`. Assertions in der Form `await expect(locator)` verhalten sich ebenso: Sie fragen das Element immer wieder neu ab, bis die Bedingung gilt oder das Zeitlimit der Assertion von standardmäßig fünf Sekunden abläuft.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist Playwright?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Ein Team führt eine Idee von der Skizze bis zu einem Produkt, bei dem Ihre Kunden sich anmelden können, statt sie zwischen drei Dienstleistern weiterzureichen.

**Unsere Leistungen entdecken** Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird.
GIS-Kartenprodukt zu 10-fachem Umsatz geführt — 10x Umsatzwachstum durch die Leitung des Kartenprodukts

**GIS-Kartenprodukt zu 10-fachem Umsatz geführt** Kontakt aufnehmen →

## Warum es wichtig ist

Playwright hilft bei der Entscheidung, welches Verhalten in einem echten Browser geprüft werden muss und wie diese Prüfungen so verlässlich werden, dass man einem roten Build glaubt. Manche Fehler entstehen erst, wenn die Seite zusammengesetzt ist: ein Dialog, der die Schaltfläche verdeckt, die er freigeben sollte; ein Layout, das ein Feld bei einer bestimmten Breite versteckt; ein Skript, das in einer Engine funktioniert und in einer anderen einen Fehler wirft. Tests isolierter Codeteile können sie nicht sehen.

Browsertests werden instabil, wenn Test und Seite um den richtigen Zeitpunkt konkurrieren. Die Seite lädt Daten, animiert und rendert neu. Ein Test, der klickt, sobald seine Zeile ausgeführt wird, klickt deshalb manchmal zu früh. Die übliche Abhilfe war eine feste Pause: auf einem langsamen Rechner zu kurz, auf einem schnellen verschwendete Zeit. Automatisches Warten und wiederholte Assertions ersetzen die Pause durch die Bedingung, von der der Test tatsächlich abhängt. Der Leitfaden für Assertions sagt es klar: Webseiten zeigen Informationen asynchron an, und Assertions, die nur einmal prüfen, führen zu instabilen Tests.

Die zweite Entscheidung betrifft die Suche nach Elementen. Der Leitfaden für Locators empfiehlt zuerst Attribute, die Nutzer wahrnehmen, und vor allem Rollen-Locatoren. `getByRole` mit einem zugänglichen Namen kommt dem am nächsten, „wie Nutzer und assistive Technologien die Seite wahrnehmen“. Das hat einen nützlichen Nebeneffekt: Ein Test, der die Schaltfläche mit dem Namen „Sign in“ anklickt, schlägt fehl, wenn sie ihren Namen verliert. Derselbe Fehler würde einem Screenreader-Nutzer begegnen. Der Leitfaden sagt aber auch, dass Rollen-Locatoren „keine Barrierefreiheits-Audits und Konformitätstests ersetzen“. Test-IDs sind die Ausweichmöglichkeit. Der Leitfaden nennt sie die robusteste Option und zugleich „nicht an Nutzer gerichtet“. Eine Suite, die alles über IDs findet, beweist deshalb weniger über das Erlebnis eines Besuchers. CSS- und XPath-Selektoren werden nicht empfohlen, weil sich das DOM häufig ändert.

Der Preis ist Zeit. Einen Browser zu starten und eine Seite zu laden macht Browsertests zur langsamsten Testart eines Projekts. Scheitert ein solcher Test auf einem Build-Server, hilft der Trace Viewer bei der Fehlersuche. Das dokumentierte Konfigurationsbeispiel zeichnet beim ersten Wiederholungsversuch eines fehlgeschlagenen Tests einen Trace auf. Dabei sind zwei Wiederholungen in CI und keine lokal eingestellt. Im Runner selbst sind standardmäßig weder Tracing noch Wiederholungen aktiviert. Mit dem Trace lässt sich jede Aktion mit den Seitenzuständen davor und danach, den Netzwerkanfragen und der Konsole durchgehen.

## Wie es in der Praxis aussieht

Eine Plattform hier hält ihre Geschäftslogik in der Datenbank. Die Aufgabe bestand darin, eine gestufte Testsuite über vier Ebenen aufzubauen. Ihr erklärtes Ziel war, das Verhalten jeder Ebene dort zu prüfen, wo es stattfindet, „statt alles von außen durch einen Browser zu prüfen“. Die Go-API enthält 981 Testfunktionen. Die Datenbank enthält 494 Dateien mit Verhaltenstests. Eine gespeicherte Funktion wird somit in der Datenbank geprüft, statt durch drei

darüberliegende Anwendungsebenen. Das Frontend enthält 543 Vitest- und Playwright-Specs, darunter 48 End-to-End-Dateien und 30 Browser-Specs. Die Seite nennt eine gemeinsame Zahl für beide Werkzeuge. Die gesamte Suite läuft als 10 Jobs in einem 612 Zeilen langen Workflow. Die Seite benennt den Preis offen: Die Suite ist langsam, belastet jede Änderung und braucht bei dieser Größe eigene Wartung.

Diese Website verwendet für ihre eigenen Browserprüfungen eine andere Bibliothek, Puppeteer. Ihre Credits-Seite zeigt dieselbe Abwägung. Die Prüfungen decken ab, was ein Validator nicht erfassen kann, etwa gemessenen Kontrast, Fokusreihenfolge, Zielgröße, über den Viewport hinausragende Inhalte und Seitenumbrüche beim Drucken. Die verbindliche Qualitätsprüfung führt vor jedem Push alle außer der Viewport-Überlaufprüfung aus. Diese wartet wie der pixelgenaue visuelle Regressionstest darauf, dass jemand sie bei Bedarf startet.

## Wo es schiefgeht

Die meisten Probleme mit Playwright entstehen durch Tests, die weniger prüfen, als ihre Namen behaupten.

**Eine feste Pause ersetzt eine Bedingung.** Ein Test wartet zwei Sekunden und liest dann die Seite oder liest einen Wert einmalig und vergleicht ihn mit einer allgemeinen Assertion. Auf einem ausgelasteten Build-Server sind zwei Sekunden manchmal zu kurz. Prüfen Sie mit einer auf Webseiten ausgerichteten Assertion wie `toHaveText`, was ein Besucher sehen würde, und lassen Sie sie erneut prüfen.

**Elemente werden anhand ihres Markups gefunden.** Ein Selektor durch drei verschachtelte `div`-Elemente oder eine Gestaltungsklasse scheitert beim nächsten Redesign, obwohl die Seite weiterhin funktioniert. Finden Sie das Element über Rolle und zugänglichen Namen oder über Beschriftung oder Text. Ergänzen Sie nur dort eine Test-ID, wo nichts für den Nutzer Wahrnehmbares es identifiziert.

**Ein erfolgreicher Rollen-Locator gilt als Barrierefreiheitsprüfung.** `getByRole` beweist, dass ein Element die vom Test verlangte Rolle und den verlangten Namen hat. Über Kontrast, Fokusreihenfolge oder den Rest der Seite sagt es nichts. Führen Sie dafür im Test eine Regel-Engine wie axe aus, wie es Playwrights eigener Leitfaden über `@axe-core/playwright` tut. Der Scan erfasst die Seite „in ihrem aktuellen Zustand zum Zeitpunkt des Aufrufs“. Öffnen Sie also zuerst das Menü oder den Dialog.

**Ein Klick wird erzwungen.** Die Option `force` schaltet die nicht wesentlichen Prüfungen der Ausführbarkeit ab. Bei einem Klick prüft Playwright dann nicht mehr, ob das Ziel das Ereignis tatsächlich empfängt. Ein Test, der das braucht, umgeht oft etwas, das auch ein Besucher vorfinden würde, etwa ein Banner über der Schaltfläche. Behandeln Sie den ursprünglichen Fehler als Befund. Schließen Sie entweder das Banner so, wie ein Besucher es tun würde, oder beheben Sie die Überlappung.

**Ein instabiler Test wird so oft wiederholt, bis er besteht.** Bei aktivierten Wiederholungen wird ein Test, der zuerst scheitert und bei einer Wiederholung besteht, als instabil statt als fehlgeschlagen markiert. Ein über Wochen instabiler Test konkurriert entweder mit der Seite um den richtigen Zeitpunkt oder findet einen echten sporadischen Fehler. Der bei seiner ersten Wiederholung aufgezeichnete Trace ist der Ausgangspunkt für die Suche.

**Alles wird durch den Browser geprüft.** Eine Regel zu Preisen oder Berechtigungen durch Klicks über drei Ansichten zu testen ist langsam und scheitert aus Gründen, die nichts mit der Regel zu tun haben. Prüfen Sie die Regel dort, wo sie liegt, und behalten Sie Browsertests für das, was nur ein Browser zeigen kann: den Ablauf für den Nutzer, das Layout und die Unterschiede zwischen Engines.

## Weiterführende Lektüre

- Installation — Playwright Test installieren, Beispieltests ausführen, der HTML-Bericht und UI Mode.
- Locators — die empfohlene Reihenfolge zum Finden von Elementen, von Rolle und Beschriftung bis zu Test-IDs.
- Automatisches Warten — die Prüfungen der Ausführbarkeit hinter jeder Aktion und was `force` abschaltet.
- Assertions — wiederholende und nicht wiederholende Assertions, das Standardzeitlimit, Soft Assertions und Polling.
- Trace Viewer — bei einer Wiederholung einen Trace aufzeichnen und einen fehlgeschlagenen Test Aktion für Aktion durchgehen.
- Browser — die Engines installieren, die Browserdistributionen Chrome und Edge ausführen und die zwei Headless-Modi.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

**Preise ansehen** Kontakt aufnehmen →

# Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist Playwright?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

**Unsere Leistungen entdecken**

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

**Mit Sorgfalt und Liebe gestaltet von D.**

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

**Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.**

**Gedruckt von https://engineer.company/de/glossary/playwright/**