Playwright

Axpo Group

Playwright Dokumentation Testing

Filip Mitrovic

2025

# Einrichtung

Aus diesem video eingerichtet: [Playwright Beginner Tutorial 2 | How to install](https://www.youtube.com/watch?v=IB2P1FBXjcQ)

(In vs code extensions downloadet: Github copilot

MCP Server

Hier Playwright MCP downloadet: [Discover and install MCP Servers in VS Code](https://code.visualstudio.com/mcp))

Bei test1.spec.js ist der erste test geschrieben und über Command mit diesem prompt ausgeführt:

npx playwright test tests/test1.spec.js

## Testresultate löschen

falls test results schon voll sind mit dem test dann muss ich das ausführen:

Remove-Item -Recurse -Force .\test-results

Dann werden diese test resultate gelöscht und man kann den test restarten

Die Tests werden auf [Playwright Test Report](http://localhost:9323/) berichtet und es wird stehen wo es gefailed ist

# Wie sind die Tests in Playwright aufgebaut?

import { test, expect } from '@playwright/test';

test('has title', async ({ page }) => {

await page.goto('https://playwright.dev/');

// Expect a title "to contain" a substring.

await expect(page).toHaveTitle(/Playwright/);

});

test('get started link', async ({ page }) => {

await page.goto('https://playwright.dev/');

// Click the get started link.

await page.getByRole('link', { name: 'Get started' }).click();

// Expects page to have a heading with the name of Installation.

await expect(page.getByRole('heading', { name: 'Installation' })).toBeVisible();

});

## Erklärung:

**1. Import der Testfunktionen**

import { test, expect } from '@playwright/test';

**Bedeutung:**

* test: Hauptfunktion, um einen Testfall zu definieren.
* expect: Assertion-Bibliothek von Playwright, um Erwartungen auszudrücken (z. B. expect(...).toBeVisible()).

**2. Testdefinition**

test('has title', async ({ page }) => {

...

});

**Aufbau:**

* test(...): Definiert einen einzelnen Testfall.
* Erster Parameter: Ein beschreibender Name des Tests ('has title').
* Zweiter Parameter: Eine **asynchrone Funktion**, die ein Playwright-Objekt (in diesem Fall page) bekommt.

**3. Testinhalt Schritt für Schritt**

**Test 1: 'has title'**

await page.goto('https://playwright.dev/');

* **Navigiere zur Seite**: Öffnet die Webseite im Browser.

await expect(page).toHaveTitle(/Playwright/);

* **Erwartung**: Der Seitentitel soll den Text „Playwright“ enthalten (Regulärer Ausdruck /Playwright/).

**Test 2: 'get started link'**

await page.goto('https://playwright.dev/');

* Wieder: **Seite aufrufen**.

await page.getByRole('link', { name: 'Get started' }).click();

* **Aktion ausführen**: Finde ein Element mit der Rolle link und dem Text Get started, und klicke darauf.

await expect(page.getByRole('heading', { name: 'Installation' })).toBeVisible();

* **Erwartung**: Nach dem Klick erscheint eine Überschrift mit dem Namen Installation, die sichtbar sein muss.

**4. Ablaufreihenfolge**

Wenn du npx playwright test ausführst, passiert Folgendes:

1. Alle Tests in der Datei werden **der Reihe nach** ausgeführt.
2. Für **jeden test(...) Block** wird:
   * Ein **neuer Browser-Kontext** gestartet (isoliert von anderen Tests).
   * Eine **neue Seite (page)** erzeugt.
   * Die **Testfunktion** ausgeführt.
   * Nach dem Test wird der Kontext geschlossen.

Dadurch sind Tests unabhängig voneinander und beeinflussen sich nicht gegenseitig.

**5. Teststruktur im Überblick**

Ein typischer Playwright-Test sieht so aus:

//Testdefinition

test('Beschreibung', async ({ page }) => {

// Setup: z. B. zur Seite navigieren

await page.goto('URL');

// Aktion: z. B. klicken, tippen, scrollen

await page.click('Selektor');

// Assertion: z. B. Inhalt prüfen

await expect(page.locator('Selektor')).toHaveText('...');

});

**6. Zusätzliche Funktionen**

Playwright erlaubt auch:

* beforeEach, afterEach: Setup/Teardown für jeden Test.
* describe(...): Gruppierung von Tests.
* Fixtures: Gemeinsame Konfiguration z. B. für Authentifizierung.
* parallele Testausführung: Standardmäßig parallelisiert.
* Konfiguration in playwright.config.ts.

# Wie generiere ich tests mit codegen in Playwright?

Im Terminal diesen Command ausgeben:

Beispiel

npx playwright codegen --target=javascript https://playwright.dev/

Ich habe mich mal durchgeklickt in der Website und dann werden Tests automatisch generiert und ich kann diese dann jeweils speichern oder kopieren:

Ein Bild, das Text, Screenshot, Software, Display enthält.

KI-generierte Inhalte können fehlerhaft sein.

# Wie öffne ich den trace Viewer?

Command:

npx playwright show-trace

Dann kannn man die entsprechende File aussuchen die man im Trace viewer sehen möchte

# Wo stelle ich ein das playwright screenshots oder aufnahmen macht bei fehlern?

Playwright.config.js:

use: {

    // 👇 Bei Fehler Screenshot machen

    screenshot: 'only-on-failure',

    // 👇 Tracing beim ersten Fehler aktivieren

    trace: 'retain-on-failure',

    // 👇 Optional: Videoaufnahme bei Fehler (kann Speicher fressen)

    video: 'retain-on-failure',

    // baseURL kannst du aktivieren, wenn du oft dieselbe Domain nutzt

    // baseURL: 'https://example.com',

  },

Fehlernachweise finde ich hier:

playwright-report/ ← interaktiver HTML-Report

test-results/ ← Screenshots, Videos, Traces

**Test ausführen**

npx playwright test

Dann siehst du im Report z. B.:

* 🔴 fehlgeschlagene Tests
* 📸 Screenshot
* 🎥 Video (wenn aktiviert)
* 🧭 Trace-Viewer (mit DOM, Netzwerk, Events)

npx playwright show-report

oder einfach im ordner öffnen

# Kann man in Playwright definieren wann tests automatisch ausgeführt werden?

Sollte funktionieren mit **Windows task scheduler**

Ich müsste es so einrichten das es automatisch diesen command laufen lässt:

cd C:\Users\filip\OneDrive\Desktop\Playwright\_Automation

npx playwright test

# Wo wird definiert was getestet wird?

Im playwright.config.js definiere ich was getestet werden soll also welches Verzeichnis in diesem Fall ist es der Tests Verzeichnis im Projekt:

export default defineConfig({

  testDir: './tests',

wenn ich nur einen Test ausführen möchte, dann muss ich genau Verzeichnis beim Command geben Z.B.:

npx playwright test tests/record1\_demo.spec.js

Report anzeigen:

npx playwright show-report

so führe ich beide tests gleichzeitig aus:

npx playwright test

# Testen planen

*​*Testumgebung

**​Hardware:** Die Testgeräte umfassen PCs mit den folgenden Spezifikationen:

* ​ Prozessor: Mindestens Intel i5
* ​ Arbeitsspeicher: Mindestens 8 GB RAM
* ​ Speicherplatz: Mindestens 256 GB SSD
* ​ Bildschirmauflösung: Mindestens 1920x1080
* ​ Software: Die Tests werden unter Verwendung von Visual Studio durchgeführt.
* ​ Visual Studio Version: 2019 oder neuer
* ​ .NET-Framework: Version 4.8 oder höher

*​*Testmittel

**​Testdaten:**

​Die Daten haben wir in den Testfällen schon definiert. Wir werden diese für das Testen verwenden.

​

**​ Testwerkzeuge und Software:**

​ Playwright und Ky2help

**​ Personelle Ressourcen:**

**​ Tester: Filip Mitrovic**  
**Entwickler:** XY

## Anforderungsnalyse Vorlage

|  |  |  |
| --- | --- | --- |
| **Anf.-Nr.** | **Typ** | **Anforderung** |
|  |  |  |

## Anforderungsanalyse Beispiel:

|  |  |  |
| --- | --- | --- |
| **Anf.-Nr.** | **Typ** | **Anforderung** |
| 1 | Funktional | Das Programm soll vom Benutzer eine Prüfungs-Note als Gleitkommazahl abfragen. |
| 2 | Funktional | Das Programm soll die Note auf Validität prüfen (>= 1, <=6, halbe/ganze Noten). |
| 3 | Qualität | Eingabefehler(1) sollen abgefangen werden. |
| 4 | Qualität | Das Programm soll portabel sein.(2) |
| 5 | Randbedingung | Das Programm soll bis zum vorletzten Unterrichts-Block des Moduls fertiggestellt sein (inkl. Tests). |
| 6 | Randbedingung | Das Programm soll in C# geschrieben sein. |

# Testfallspezifikation Beispiel

**Testfallnummer:** 1.1

**Anforderungsnummer:** 1

**Voraussetzungen:**

Das Programm wurde gestartet

**Eingabe:**

1. j
2. 5.5

**Ausgabe:**

1. Möchten Sie eine weitere Note eingeben? [j/n]
2. Geben Sie eine Note ein:
3. Ihr Notenschnitt ist 5.5

# Testprotokoll bei manuelles testen

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test-Nr.** | **Testfall-Nr.** | **Datum** | **Tester** | **Resultat** | **Bemerkung** | **Unterschrift** |
| 1.1.1 | 1.1 | 07.08.2025 | Filip Mitrovic | OK | - |  |
| 1.2.1 | 1.2 | 07.08.2025 | Filip Mitrovic | OK | - |  |
| 1.3.1 | 1.3 | 07.08.2025 | Filip Mitrovic | NOK | Notenschnitt wird nicht korrekt berechnet |  |
| 1.4.1 | 1.4 | 07.08.2025 | Filip Mitrovic | OK |  |  |
| 1.5.1 | 1.5 | 07.08.2025 | Filip Mitrovic | NOK | Laufzeitfehler bei Eingabe «n» |  |

# Testabschlussbericht Fazit

Grobe Beispiel:

Die Tests sind weitgehend erfolgreich verlaufen.

Es bestehen lediglich 2 Fehler, welche jedoch störend wirken und behoben werden müssen.

Das Produkt muss überarbeitet werden, bevor es an den Kunden übergeben werden kann.

# Zukunfsziele

Ziel ist mit ai copilot automatisiert testen lassen und das er die tests automatisch schreibt und report generiert usw. Wie bei diesem Video:

[Lassen Sie KI Ihre Site erkunden und Tests mit Playwright MCP schreiben!](https://www.youtube.com/watch?v=IixdI2bTR1g)

## Ai einbauen test

Github copilot heruntergeladen und gebe prompts in vs code beim Copilot

Wenn ich die test ausführungen live ansehen möchte muss ich headed Modus anmachen entweder ich sage es dem Copilot oder ändere es manuell

test.use({ headless: false }); // Sichtbarer Browser!

Funktioniert mittlerweile nicht so gut, mit codegen viel besser mittlerweile

Eventuell .md dateien erstellen und der KI erklären was er genau zu tun hat usw.

Wie hier:

[Manual Testing with Playwright MCP – No Code, Just Prompts!](https://www.youtube.com/watch?v=2vnttb-YZrA&t=209s)