Stop Fighting Your Tests 🛑

With Playwright MCP

Axon Active · January 2026

Today's Journey 🗺️

🤔 The Problem

- Slow feedback loop
- Requires significant resources to create and maintain automation test

✨ The Solution

- MCP: AI-to-Browser bridge

🎬 Live Demo

What is Playwright, Playwright MCP? 🎭

Playwright

End-to-end testing framework

- Write tests in JS/TS/Java/...
- Supports: Chrome, Firefox, Safari, ...
- Auto-wait, screenshots, mock/track/modify network traffic

> 👨‍💻 **You** write code → Playwright runs it

Playwright MCP

AI-to-Browser bridge

- Uses Playwright under the hood
- Enables AI to see, click, type, navigate
- Debug-style communication

> ✨ **AI** decides → Playwright MCP executes

MCP: Dynamic Discovery · Stateful · Standardized Schema · 2-way Communication

Different Applications: Playwright / MCP 🎯

Use Playwright for:

- Regression testing
- Complex assertion criteria
- Complex mock APIs
- Stable applications
- Fixed test steps

> 📋 Deterministic

Use Playwright MCP for:

- Exploring features
- Reproducing bugs
- Rapidly changing UIs
- Debugging
- Prototyping tests

> 🧠 Adaptive

💡 **Best practice**: Use MCP to *generate*, Playwright to *run* regression suites

Setup

System Requirements

- **Node.js 18+** required
- MCP-compatible client (Claude Code, VS Code, Cursor, etc.)

Claude Code CLI

```
claude mcp add playwright \
  npx @playwright/mcp@latest
```

VS Code / Cursor

Add to your MCP settings:

```
{
  "mcpServers": {
    "playwright": {
      "command": "npx",
      "args": ["@playwright/mcp@latest"]
    }
  }
}
```

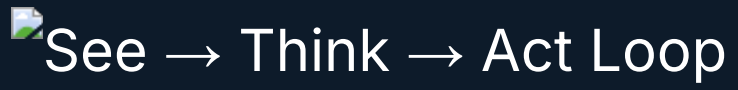
Verify installation: `npx @playwright/mcp@latest --help`

📖 Full docs: github.com/microsoft/playwright-mcp

How Does AI work with Playwright MCP? 🔧

Let's dive into the **technical concepts**

Set up a simple loop for ✨ to DEBUG your test



See → Think → Act → Repeat 🔄

> /investigate "Todo List" test report: "should mark a task as complete" failed

1 🎭 **reads browser** → ✨

```
checkbox "Watch Netflix" [unchecked]
checkbox "Go shopping" [unchecked]
text "Watch Netflix" (no strikethrough)
```

A11y Tree: semantic, compact

2 **Test report tells ✨ what failed**

> "Checkbox clicked but strikethrough never appeared"

3 ✨ **generates** → 🎭 **executes**

```
await page.getByRole('checkbox').first().click();
```

4 ✨ sees no change → **repeat loop** until root cause found

Playwright MCP Testing Use Cases 🤖

1. Failed Test Investigation

2. Exploratory Testing → Test Plan

3. User Story → Test Plan → Automated Tests

4. Bug Retest

5. Bug Logging / Document Issues

Each use case = **AI prompt** + **Playwright MCP browser automation**

Use Case 1: Failed Test Investigation 🐛

Prompt:

Act as Debugging Specialist. Investigate failing test [FILE + ERROR].

- Live reproduce the failure
- Analyze element state (visibility, ARIA, overlays)
- Check console errors & network (4xx/5xx)
- Provide: RCA, proposed fix, live verification of fix

Now, From a QA Perspective... 🎯

How can **QA Engineers** leverage AI + Playwright MCP in daily work?

Use Case 2: Exploratory Testing & Test Plan Generation 🔍

Prompt: Act as QA Engineer. Explore [URL/Feature]. Navigate main flows, inspect UI/UX, test edge cases, check accessibility.

Output: Generate test plan with:

- Summary of application under test
- 5-10 test cases (happy path + edge cases)
- Bugs/observations found
- Playwright snippets for automation

Use Case 3: User Story → Test Plan → Automated Tests 📝

Prompt: Act as QA Lead. Convert user stories [PASTE STORIES] to test suite.

Phases:

- **Phase 1**: Extract acceptance criteria, generate test plan (happy path + 2 edge cases per story)
- **Phase 2**: Verify live via Playwright MCP, confirm selectors exist
- **Phase 3**: Generate TypeScript test file with POM patterns, getByRole locators

Use Case 4: Bug Retest ❌→✅

Prompt: Act as QA Engineer. Retest bug [BUG ID + REPRO STEPS].

Steps:

- Execute repro steps via Playwright
- Inspect page state, network calls, DOM
- Verdict: 'BUG FIXED' or 'BUG PERSISTS' with evidence
- Bonus: Generate regression test script

Use Case 5: Bug Logging / Document Issues 📋

Prompt: Act as QA Engineer. Document bug [DESCRIBE ISSUE].

Steps:

- Reproduce & record steps
- Collect: console logs, network errors, screenshot
- Analyze root cause (hidden/disabled/covered element)
- Generate bug report: title, severity, environment, steps, actual vs expected, technical evidence

Beyond Prompts: Commands 🎯

Structured workflows for **repeatable quality**

What Are Commands? 🤔

Traditional Prompts 💬

Ad-hoc instructions to AI

- "Generate test cases for this feature"
- "Help me debug this test"
- Flexible but **inconsistent**
- Requires expertise to craft
- Results vary between runs

> 🎲 **One-off** interactions

Commands ⚙️

Structured, repeatable workflows

- Pre-defined multi-step process
- Built-in knowledge base
- Consistent output format
- Quality checks included
- Version controlled

> 🔄 **Systematic** process

💡 **Think:** Prompts = Manual testing · Commands = Automated testing

Why Commands Over Prompts? 🎯

Benefits

✅ **Consistency** - Same process
✅ **Knowledge** - Team expertise
✅ **Onboarding** - Instant practices
✅ **Quality** - Built-in validation
✅ **Scalability** - Repeat

Example Commands

`/qa-test-plan-generation`

Story → Test plan

`/qa-implement-automation-tests`

Plan → TypeScript tests

`/qa-run-test-with-mcp`

Execute & debug

Example: Test Plan Generation Command 🔄

Phase 1: Prerequisites

Step 1: Read `sofia-test/CLAUDE.md`
Step 2: Read `CLAUDE.md` (Azure DevOps section)
Step 3: Read `CLAUDE_KNOWLEDGE_BASE.md`
Step 4: Read `CLAUDE_AUTOMATION_BEST_PRACTICE.md`

Phase 2: Data Collection

- `/read-workitems <story-id>`
- `az repos pr show` (parallel)
- Glob search page objects
- Glob search existing tests
- Glob search frontend components

Phase 3: Analysis & Test Generation

- Analyze data (reference Knowledge Base)
- Generate test cases (reference Best Practices)

Phase 4: Document Creation

- Create test plan markdown file

Phase 5: Follow-up Actions

- Get user approval
- Create Azure DevOps test cases (optional)
- Implement automation tests (optional)

Example: MCP Testing Command

Phase 0: Prerequisites - Read `sofia-test/CLAUDE.md` + `MCP_GUIDE.md`

Phase 1: Setup - `pwsh Mcp-Setup.ps1 "$ARGUMENTS"` - Parse input, Fetch work item, Authenticate

↓ Mode Selection ↓

Bug Retest `bug:12345`

1. Fetch Bug
2. Parse Steps
3. Execute with MCP
4. Report (Fixed/Still bugs)

Test Plan `plan.md`

1. Parse Plan
2. Extract TCs
3. Execute Each TC
4. Report (Pass/Fail Rate)

Exploratory `explore:123`

1. Fetch Story
2. Parse ACs
3. Create Plan & Explore
4. Report (Coverage/Bugs)

Example: Implement Automation Tests Workflow

📖 **Step 0: Read Files (Parallel)** - CLAUDE.md, KNOWLEDGE_BASE.md, BEST_PRACTICE.md

🔧 **Step 1: Parse Input** - `pwsh Implement-AutomationTests.ps1` → Auto-detect Work Item/Test Plan → Output JSON

❓ **Step 2: Ask Clarifications** - Review unclear items → Ask user → ⏸️ WAIT FOR RESPONSE

🔍 **Step 3: Research & Plan (Parallel)**

- Explore: Find patterns, similar tests, page objects
- Present plan & approach → ⏸️ WAIT FOR APPROVAL

⚙️ **Step 4: Implement (Parallel Agents)**

- Agent 1: Page Objects (methods, localization)
- Agent 2: Test Files (step() helper, assertions)

✅ **Step 5: Run & Verify**

yarn format → test:e2e → Fix → Re-run

📊 **Step 6: Present Results**

Files, Tests (Pass/Fail), Methods, Next steps

What This Means for You 👥

💻

Developers

Earlier feedback loop

🧪

QA Engineers

Spend less time on testing process

Limitations - Be Honest ⚠️

🤖 Does NOT Handle Well:

❌ Complex visual assertions

❌ Non-deterministic content

- Real-time data feeds
- Time-sensitive tests

❌ Heavy authentication flows

- Multi-factor auth, CAPTCHA, biometrics

Still Needs Human Review:

⚠️ Business logic validation

⚠️ Edge case prioritization

- ✨ finds many issues, you decide importance

⚠️ Security-sensitive tests

- Don't expose credentials to ✨

Rule of thumb:

- 80% ✨ work
- 20% Human judgment

Questions? 🙋

Thank You!

Let's make testing fun again