


TAMING

THE

TROWSER

Introduction

Trowser, the testing browser, is a beast of a software, might not be for the faint-hearted, and not easy to get a full grip on. Therefore I have written this guide on how to understand what it can do, and suggest how it could help you with your software testing. I think the audience is testing practitioners who enjoy rich, exploratory testing, with the purpose of finding important quality-related information about web services.

We will start with an introduction, then go through a hands-on example that gives you the basics of how to use Trowser. Third part gives suggestions on how to make it even more powerful in your context, and finally an epilogue.

Trowser started with the idea that my own browser could enable more powerful testing. APIs give power, fast access to tools as well, so I built it all in. I had never seen LLMs do exploratory testing before, and it was fascinating to see that their different perspective complemented mine, and was useful. They can't think for real, but they can take instructions, interact with a web site, change what they do depending on the results, and report the findings better than most human testers can. They can also document the heuristics and oracles they used, they can write down what they learned for next session, and I have yet to see a test session where I didn't learn anything new (not always important stuff, though).

However, they have trouble understanding what is important, they can't separate right from wrong, so we still need a human tester in charge, guiding the test effort, and being responsible for the test results.

That's where Trowser stands out, it is not only a tool for LLM testing and automated regression tests, it also enables the human tester to do faster testing, and makes it easier to judge the results from the machine.

Trowser also has downsides, it is a one-man project that didn't want to buy a proper certificate for signing, it only runs on Windows 64-bit, and it only works on browser software.

Security is a real concern, so read SECURITY.md before you launch the .exe; be careful with what you do where, and act responsibly in general.

If you are new to testing, it will be hard to use, but also massive learning opportunities. There's a lot of power in it, but you have to be able to decide when to use that power and how to interpret the results. You will not have Trowser as your only browser for testing, and sometimes you will use other tools in order to really evaluate how your software is doing.

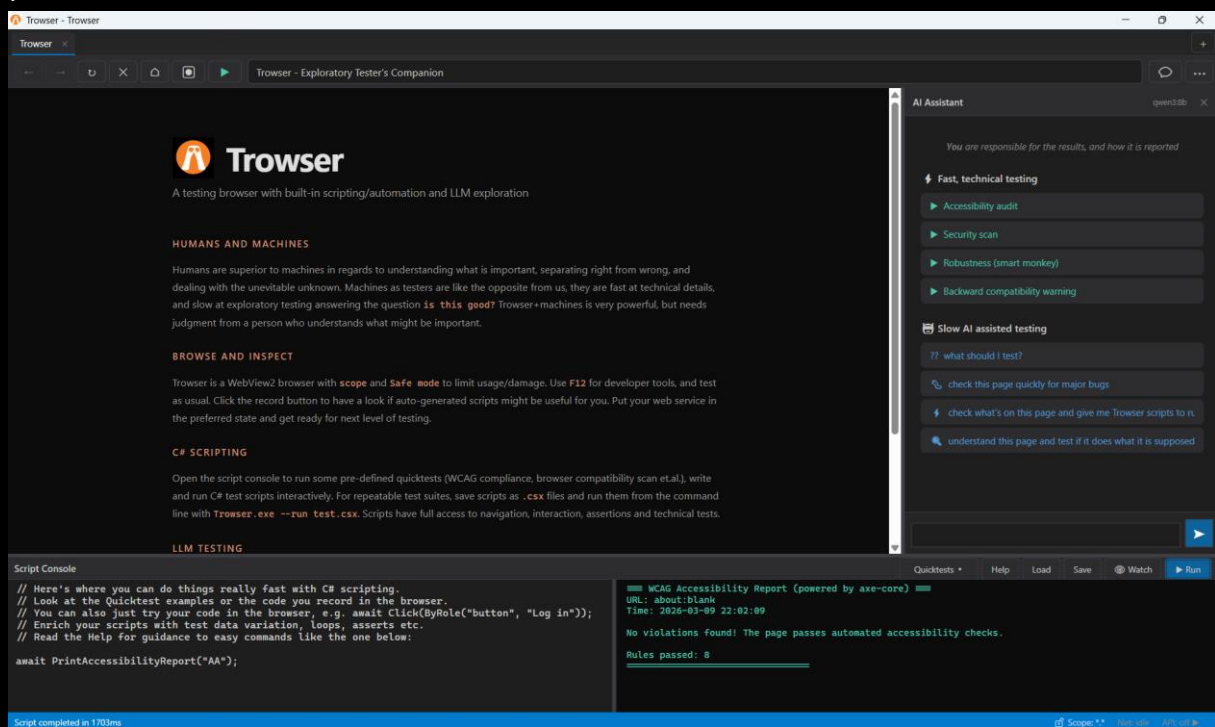
Hopefully this guide can help you sort things out, and if you will feel just a little bit as powerful as I did during the creation process, that is good enough.

Walkthrough of Trowser

You can follow along with Trowser in this chapter, hands-on experience usually gives better learning.

Download <https://www.thetesteye.com/code/Trowser-1.0.zip> and unzip on your machine. You will need a Windows 64-bit machine with WebView2 installed (a part of Edge). If you want to avoid security warning you can follow the steps in SMARTScreen.md.

Double-click Trowser.exe and notice a splash screen that is displayed and disappears. It reads “understand what is important, then test it”. It is my old essence of testing (the new is “understand relations”) and it is a reminder to constantly understand what matters now. Which testing has the best chance of helping me right now?



Trowser is a stripped-down browser with few standard features. It can show web pages (WebView2), it has address bar, tabs, refresh, stop, back and forward, search, and that’s about it if we skip everything testing related.



The home button is quite pointless, but Claude Opus 4.6 added it by itself, and I like the welcome page, so I let it stay.

Click on the record button and save a .csx file (Roslyn C# Script File). We save at once, and continuously when recording. This means it is kept on a crash, and you can also look at the file if you want to see what is recorded along the way.

Type <https://www.thetesteye.com/exercises/RTD/buggy/> in the address bar and press Enter. This is an exercise site with a lot of bugs. Click “Get Rnadam String”.

Add your own favorite string to the collection. Notice that you get a warning if you really want to add a string. This is a usability bug that stops fast usage, a bug the LLM won’t notice unless you have very good instructions or requirements. A tester, or a frequent user, will notice this quickly though.

Click the Stop button; we have a lot to go through. Press Ctrl+0 (or Script Console in top right menu), click the Load button and open the newly recorded .csx file.

```
Script Console
await Navigate("https://www.thetesteye.com/exercises/RTD/buggy/index.html");
await WaitForElement("body", 15000);

await Click(ByRole("button", "🔍 Get Rnadam String"));
await Select(ByLabel("Category"), "Custom"); // "Custom"
await Type(ByLabel("Test String"), "favorite string");
await Type(ByLabel("Notes (optional)"), "it is very good");
await Click(ByRole("button", "➕ Add String"));
```

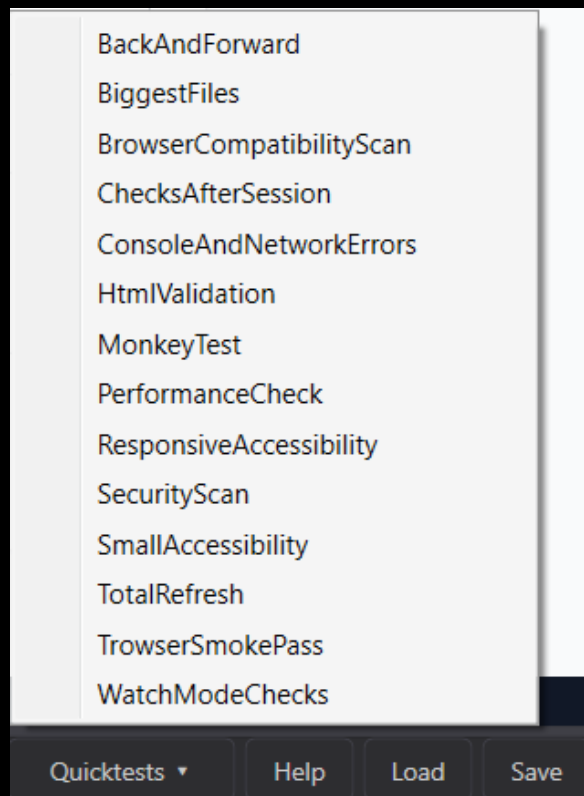
This is a typical script file in Trowser. It is C# scripting with Roslyn, you might not be used with the syntax (await is synchronous, async is asynchronous), but it is not difficult. Maybe you won’t write any scripts yourself (LLMs are very good at it), but it is good to know the basics. ByRole and ByLabel are key concepts in Trowser APIs, the LLMs find them handy and they are easy to understand.

Run the script, and see the actions be performed and console output being written to the right part of the Script Console.

Make any change and Save (just to get you to feel that you are the one in charge here).

Click on Help and scroll through what is available script wise, just to get a basic feeling.

Quicktests



Click on the Quicktests menu and select and run the item that feels most compelling.

These are scripts that both show examples of scripting, but also help you do some testing really fast. All of them can be changed by you, but must be saved as a new file.

BackAndForward – Go 10 times back and then 10 times forward to see if you end up at the same place. Could find problems with navigation that very well might happen to your users.

BiggestFiles – It is not uncommon that web services use too large files. Run this now and notice a 955,9 KB .bmp logo. That is a bug, no need to use a big .bmp for that small logo.

BrowserCompatibilityScan – If your web service has a lot of customers with big variety, many of them will use older browsers. And even if you have a lot of physical devices and a Browserstack account, it is quite nifty to be able to make a quick check on the thing you are testing right now. And perhaps more importantly, to be able to compare with last version of your software (Angular is often built with “2 major versions back”, so when a new iOS or Android version comes, some of your users could get into trouble).

This test is custom built in Trowser by checking all the loaded JavaScript functions. A warning doesn't mean it will fail, because perhaps that function never is used. Also beware that the checks go with the vanilla version for Android, most users have upgraded their browsers.

ChecksAfterSession – A combination of some of the quicktests, to be run at the end to see we didn't miss anything important during the test session.

ConsoleAndNetworkErrors – Looks at the whole history of errors in console and network traffic. For fresh start, use `ClearConsoleLog();` and `ClearNetworkLog();`

HtmlValidation – Inspects HTML with 3rd party html-validate. Very picky, few sites have no comments, and it can work perfectly with a lot of errors.

MonkeyTest – Random actions with 3rd party gremlins.js. Haven't seen it catch a bug, but fun to watch. Trowser has its own Smart Monkey, that uses the knowledge of what can be interacted with on the page. Smart Monkey is only allowed when Scope is set (hard to know what could happen otherwise).

PerformanceCheck – Evaluation of web performance with 3rd party web-vitals. Similar to what Developer Tools offer, but handy to have scriptable.

ResponsiveAccessibility – Changes screen size and checks WCAG compliance with 3rd party axe-core. Best practices are not checked, but WCAG compliance is a hygiene factor for sites that want to reach a lot of people.

SecurityScan – A small scan consisting of JavaScript CVEs, HTTP headers, CORS, and endpoint discovery (CVEs powered by 3rd party Retire.js, the others custom built based on OWASP). This is not a penetration test, but a lightweight way to see common problems. Note that many sites don't follow best practices for Response Headers, but it is best to do that on purpose.

SmallAccessibility – WCAG compliance test without different viewports. Might be removed in a future version, to make better room for other nice quicktests.

TotalRefresh – Refreshes the page 100 times. Should not be a problem, but one never knows...

TrowserSmokePass – This is a simple smoke pass of Trowser itself. It is actually one of few automated tests in place, most bugs have been found by human and LLM usage, and focus has been on development speed (so far).

WatchModeChecks – A combination of some of the other quicktests, as an example of things you might want to check on each navigation. Retire.js is the one that has helped me most here.



Click the Watch mode and see that current script will be run on each navigation. Very useful for quick scans you want to be executed on many pages; a passive scanning that does the job for you almost for free. Here is the place to add specific tests for things that are important to always be on the page, e.g. the name of a logged in user, company name, specific footer link etc.

Testing is never better than the communication of the results

Now comes the difficult part, what do we do with the results?

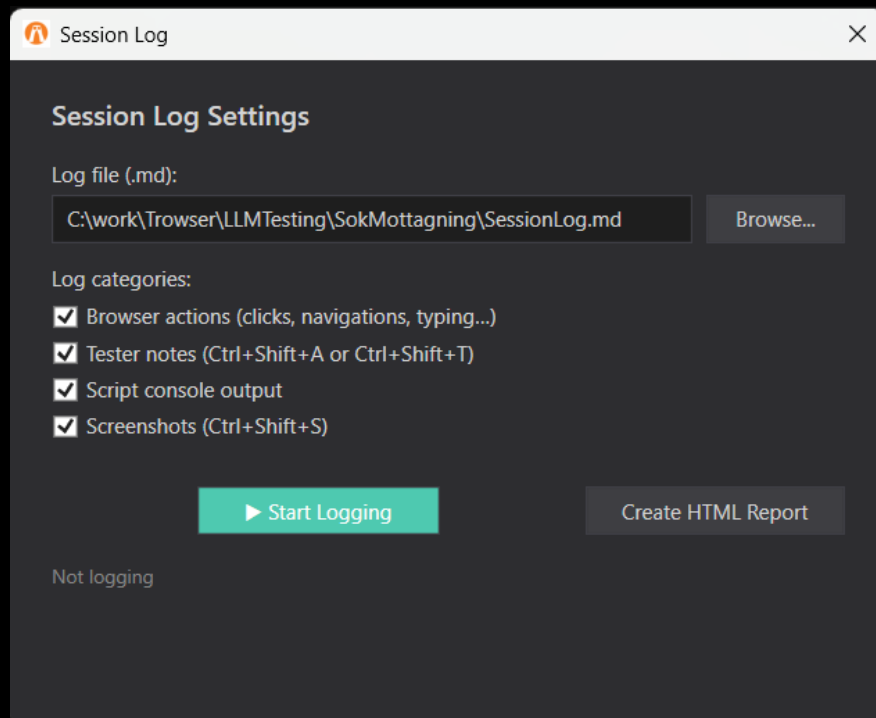
In the specific case with the buggy Random Test Data we have no stakeholder. There is no one asking for information about the state of the product, so the testing we do is purely for our learning.

But at work, you will have important stakeholders: developers, product owners, testers, and maybe sales and legal as well. They have explicit needs that they might have told you about (otherwise ask them!), but they also have implicit information needs they don't know about. As an example they might say that performance isn't of concern at the moment, but if the product is barely usable because of that, they would be interested. That's a beauty of quicktests, a short check of an area that might discover a risk you didn't know about.

You also need to evaluate the results, not everything is important, and communicating it would not be productive (a list of all the findings from html-validate could easily give besserwisser-vibes that won't help you in your future team work.)

Some things are worth reporting directly, and some things might just be noted down for your own sake, perhaps never used, or perhaps re-used in another situation where you know more about what matters. You need to understand the software, the stakeholders, the users, your company etc. in order to figure out what to do with your findings. When you understand the relations between people and software, your communication will be even better.

To get better material for your reporting, you can activate Session Log in the menu.

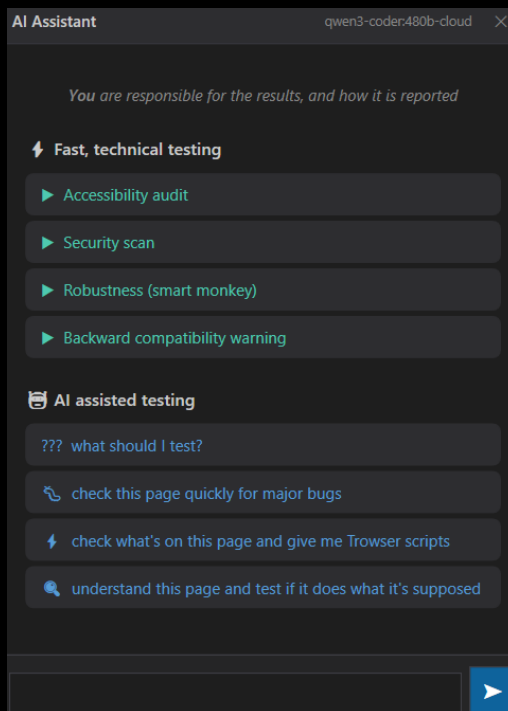


This will record what happens so you can look at it afterwards. It writes to file and to the API, which means an LLM can see what you are doing and do their own tests on the same theme in a second Trowser instance.

AI Assistant

Press Ctrl+L or the button next to top right menu, to display the chat window with AI Assistant.

This is the part of Trowser I am least satisfied with, it felt like a killer feature, but it often feels slow and doesn't have the same power as when a local agent uses the REST API. It might be better later on, and I think the Chat mode can be handy and provide additional value.



At the top we have the important reminder *you are responsible for the result, and how it is reported*.

Next follows links to some of the quicktests. This is because these are faster and more reliable than if you would ask an LLM to do the same (you can do that as well, as a second opinion). I want to promote the tester being in charge, using technical tools when appropriate, and LLMs as a different perspective for a better whole.

Next comes quick access to prompts for the LLM, but before we can run them we must configure the settings (in top right menu).

AI / LLM

Interaction Mode

- Chat — testing advisor, observes and guides (default)
- Full — interactive exploratory testing (more tokens)
- Economical — single snapshot + test script (fewer tokens)

Chat: AI observes the page and advises what to test. Full: AI interacts with the browser. Economical: AI writes a test script.

Provider

OpenAI-compatible (OpenAI, Ollama) ▾

API Endpoint

http://localhost:11434/v1

OpenAI: <https://api.openai.com/v1> | Ollama: <http://localhost:11434/v1> | Copilot: auto-detected

API Key

Leave empty for local models or GitHub Copilot (auto-obtained via gh CLI)

Model

qwen3-coder:480b-cloud

Examples: gpt-4o, claude-sonnet-4-20250514, llama3, deepseek-r1

Temperature **Max Tokens (optional)**

0,5

There are three interaction modes to choose from:

- **Chat** – LLM is instructed to give the best help regarding testing that it can based on your prompts. If requested, it can retrieve information with `browser_snapshot`, and use testing heuristics from the exploratory testing community.
- **Full** – this is the expensive mode where a lot of data is transferred, the LLM can interact with the web site, check the result and report its findings.
- **Economical** – fewer interactions, and instructed to rather run scripts for the testing than to check the status of the page over and over.

Be aware that all LLMs send the full chat history for each prompt, so start from scratch whenever the context isn't necessary anymore.

Three provider types are supported: OpenAI Compatible, Anthropic, and GitHub Copilot, and this supports most of what you probably have available. Ollama is Open AI compatible, but for local models to be good enough, you need a very powerful computer.

API endpoint, API key, and Model must be exactly right, this part is good-old software. As for temperature I have been running quite a lot with 0.7, without any real data suggesting it is the best.

Max tokens is getting more and more important, it can cost some money quite fast, so this is entirely up to you.

If you are ready, go back to the AI Assistant and see how it works for you on your software. There is a Show as HTML to make the result better-looking, but you will get better results with the REST API.

Status Bar



The status bar shows to the left which page is loading, when it is ready, and time taken for scripts. To the right we have five interesting buttons:

- **Scope** – Ability to limit which pages Trowser is allowed to navigate to, supports regex. One of the first features built, but rarely used since the LLMs tend to stay on target.
- **Network interception** – Ability to set breakpoints and change the request or response, and also replay same request in parallel (I wanted at least something small for load testing). Also includes very simple mocking, e.g. to quickly test error states.
- **Session Log** – Actions, messages, screenshots are saved to file.
- **REST API** – Turns on the REST API, the real enabler of new sorts of testing (rest of Trowser is not much new, just a lot more convenient).
- **Messages** – Click here to see messages you have received during the session, e.g. from an LLM coach, or an LLM pair tester.

Browser

Navigation Scope (comma-separated patterns, *.* = no restrictions)

Examples: https://*.myapp.com | https://staging.app.com, https://api.app.com | *.*

REST API Port

 Use different ports to run multiple Trowser instances

Route traffic through proxy

Proxy URL

e.g. http://localhost:8080 for OWASP ZAP, Burp Suite, Fiddler. Requires restart to take effect.

Security

API Key (required for all API requests)

Pass via header: Authorization: Bearer <key>. Auto-generated each session; use --api-key to set a fixed key.

Safe mode — block JS execution, C# scripting, network mocking, cookie manipulation

Enable when browsing untrusted sites to prevent the API from executing code or intercepting requests.

By default the REST API and MCP requires a session unique API key to function. It can be turned off for troubleshooting purposes (navigate browser to localhost:6923/api/state), but don't do that too much. Since the API is very powerful, allows navigating, clicking and most things you can do, it should not be open. It is on your local machine, but one never knows what can happen.

You can have two Trowser instances on different ports to enable ping-pong tests.

You can route the traffic to a proxy, e.g. your real security testing tools.

You can turn on a safe mode, if JavaScript execution and other heavy stuff isn't wanted.

I'll just mention that the menu has a Help and About, because we need to move on to the exciting chapter on how to make the LLMs really work in your context.

Context for LLMs

I am fascinated that the latest models (Claude Opus 4.6 and GPT 5.3-Codex) actually can do good software testing. I have used them both for software that is new to me, and software that I know very well.

The way you instruct the LLM have quite an impact on the result, and we will go through the different possibilities available.

You need a local agent (I have used Cursor and Copilot), you need quite some LLM credits, and you need the REST API active in Trowser, unless you use the MCP with equivalent features. You need to give access to relevant files in the **trowserkit** folder, which has what you need to get started.

The Prompt

The prompt gives the baseline for what to do, and small word changes can make a difference. Here is a typical prompt I use that you can try:

```
We will now do some really good testing with Trowser REST API, full documentation available in this folder, start with .github/copilot-instructions.md
```

```
I have navigated my Trowser instance at localhost:6923 to https://www.thetesteye.com/exercises/RTD/good/index.html where I want you to do deep exploratory testing to find bugs. This is a tool for software testers to generate random test data, and testers are very picky so we don't want any bugs.
```

```
Use the Trowser REST API (api-reference-small.md for overview, api-reference-full.md if you need details) to get information about the page and interact. Use testheuristics.md (in current folder) for further test inspiration than the requirements.
```

```
API key is trowser_XXX  
Look for situations where the Trowser API or its documentation can be enhanced to support more powerful testing.
```

```
If there is a file called ServiceSpecifics.md, use that information as guidance as well. If you learn important things (technical shortcuts or other good things to know) append that information to the file (create it if it doesn't exist, because testing is about learning.)
```

```
Finalize by creating automated regression tests with the CLI interface, which is documented in scripting-reference.md.
```

```
Create a test report for this functionality in a nice-looking .html file in this folder.
```

```
Explain the heuristics and oracles that helped you, if applicable.
```

```
Make a separate report for the Trowser API (and documentation) bugs and enhancements. Include an honest quote about your experience with the Trowser API.
```

```
Put our chat and your output (including thinking) in a separate file.
```

A lot of it is just instructions to get it to work well, but we have:

- Testing mission: “deep exploratory testing to find bugs”
- Super-short context: “This is a tool for software testers to generate random test data, and testers are very picky so we don't want any bugs.”
- “Learning”: ServiceSpecifics.md is a reviewable document that will give a head start for next session of the same software
- Output request: "nice-looking .html file" (works remarkably well)

Maybe you are not interested in the API feedback, but during development this was crucial.

It is for you to find out what to write in your prompt and what to have in other documents. It generally follows directions well, and your follow-up prompt might give the best results. e.g. “dig really deep for the search functionality, using a multitude of test methods”.

trowserkit includes five types of prompts:

- General exploratory testing to find bugs
- Focused session, probably one of many on same software
- Reconnaissance session to give an overview of the product and things to test
- Pair testing, you in one Trowser instance, and LLM working with same pages in another
- Coach, the LLM sees what you do, you can send a note and get testing tips

I really look forward to see examples that other tester come up with.

Testheuristics.md

The file testheuristics.md contains exploratory testing wisdom in a summary of Elisabeth Hendrickson's *Testing Heuristics Cheat Sheet* and Rikard Edgren's *The Little Black Book on Test Design*. These heuristics help the LLM to execute better tests, but you are of course free to change the content, or use a completely different file that is better in your context.

Auxiliary Files

I have kept the original testheuristics.md and rather added other files that contain useful information for the software we are currently testing. Specifications work fine, “verify the Help matches the product” is a great test, and I am looking forward to giving access to all bugs that we know about already, and types of bugs we usually find etc. If you have product-specific quicktests you want the LLM to perform, well add it as another resource in your prompt.

The Agent & Instructions

My experience with LLM development and testing says there are hidden instructions from the agent that changes the results. Cursor is not only my favourite because it was what I started with, it is also because the code output contains less problems than when I have used Copilot CLI. This is extra interesting for testing, since we want diversity it can be good that the testing done takes different paths and give different results. We still have to judge the output anyway,

but if you are familiar with the software that should not take too long (what takes time could be to investigate the new findings, to see what should be done with that information.)

Agents also have extra instructions in `.cursor\rules\trowser.mdc` and `.github/copilot-instructions.md`. Read these and change them so they fit your purposes in general (specific instructions for a testing sessions are easier to do in the prompt). Only `trowser.mdc` explains the MCP usage.

API References

`api-reference-full.md` contains all REST APIs including details. It is so long that the LLMs can't read them all at once, and it has happened several times that I get feedback for new features that we already have.

`api-reference-small.md` is therefore a better starting point, and an LLM with access to file system can then search for details when needed.

I recommend scrolling through them to get a feel of what is available. It is a very good match with the scripting abilities, but not 100%.

These documents (as all!) can of course be edited, where it might be wise to remove stuff you don't want the LLM to bother with.

The Trowser Instance

I sometimes test web services that require login, and it is very convenient to perform the login manually and go to the place where I want the LLM to start. Turn on REST API, include the API key in the prompt and see if the results are useful to you.

Note that not all things done will be visible in the GUI, and I am yet to try the experiment of activating recording and then let the LLM go for it.

Epilogue

Now it is your turn to figure out if this helps you with your testing.

I have built this tool based on my own needs, and I am hoping that it will match at least some other testers. It is quite different from other AI testing tools that often focus on test cases, self-healing, visual comparison, or automated checks we already can do with existing tools.

I want the exploratory tester to still be the champion, but with better tool support.

Rikard Edgren, 18th of March 2026
v1.0.14