AgentDeskAI / **browser-tools-mcp** Public

Code | Issues 11 | Pull requests 20 | Discussions | Actions | Projects | Security and quality | Insights

main | 4 Branches | 3 Tags | Go to file | Go to file | Code

tedjames Update README.md d298068 · last month

browser-tools-mcp	Finalize version 1.2.0 with graceful shutdo…	last year
browser-tools-server	updated docs and bumped versions	last year
chrome-extension	Finalize version 1.2.0 with graceful shutdo…	last year
docs	removed yaml from docs	last year
.DS_Store	updated readme	last year
.gitignore	chore: update .gitignore	last year
LICENSE	added license	last year
README.md	Update README.md	last month

README | MIT license

THIS PROJECT IS NO LONGER ACTIVE PLEASE USE A DIFFERENT SOLUTION FOR THIS.

BrowserTools MCP

> Make your AI tools 10x more aware and capable of interacting with your browser

This application is a powerful browser monitoring and interaction tool that enables AI-powered applications via Anthropic's Model Context Protocol (MCP) to capture and analyze browser data through a Chrome extension.

Read our docs for the full installation, quickstart and contribution guides.

Roadmap

Check out our project roadmap here: Github Roadmap / Project Board

Updates

v1.2.0 is out! Here's a quick breakdown of the update:

- You can now enable "Allow Auto-Paste into Cursor" within the DevTools panel. Screenshots will be automatically pasted into Cursor (just make sure to focus/click into the Agent input field in Cursor, otherwise it won't work!)
- Integrated a suite of SEO, performance, accessibility, and best practice analysis tools via Lighthouse
- Implemented a NextJS specific prompt used to improve SEO for a NextJS application
- Added Debugger Mode as a tool which executes all debugging tools in a particular sequence, along with a prompt to improve reasoning
- Added Audit Mode as a tool to execute all auditing tools in a particular sequence
- Resolved Windows connectivity issues
- Improved networking between BrowserTools server, extension and MCP server with host/port auto-discovery, auto-reconnect, and gracefu shutdown mechanisms
- Added ability to more easily exit out of the Browser Tools server with Ctrl+C

Quickstart Guide

There are three components to run this MCP tool:

1. Install our chrome extension from here: v1.2.0 BrowserToolsMCP Chrome Extension
2. Install the MCP server from this command within your IDE: `npx @agentdeskai/browser-tools-mcp@latest`
3. Open a new terminal and run this command: `npx @agentdeskai/browser-tools-server@latest`

- Different IDEs have different configs but this command is generally a good starting point; please reference your IDEs docs for the proper config setup

IMPORTANT TIP - there are two servers you need to install. There's...

- browser-tools-server (local nodejs server that's a middleware for gathering logs) and
- browser-tools-mcp (MCP server that you install into your IDE that communicates w/ the extension + browser-tools-server)

`npx @agentdeskai/browser-tools-mcp@latest` is what you put into your IDE `npx @agentdeskai/browser-tools-server@latest` is what you run in a new terminal window

After those three steps, open up your chrome dev tools and then the BrowserToolsMCP panel.

If you're still having issues try these steps:

- Quit / close down your browser. Not just the window but all of Chrome itself.
- Restart the local node server (browser-tools-server)
- Make sure you only have ONE instance of chrome dev tools panel open

After that, it should work but if it doesn't let me know and I can share some more steps to gather logs/info about the issue!

If you have any questions or issues, feel free to open an issue ticket! And if you have any ideas to make this better, feel free to reach out or open an issue ticket with an enhancement tag or reach out to me at @tedx_ai on x

Full Update Notes:

Coding agents like Cursor can run these audits against the current page seamlessly. By leveraging Puppeteer and the Lighthouse npm library, BrowserTools MCP can now:

- Evaluate pages for WCAG compliance
- Identify performance bottlenecks
- Flag on-page SEO issues
- Check adherence to web development best practices
- Review NextJS specific issues with SEO

...all without leaving your IDE 🎉

🔑 Key Additions

Audit Type	Description
Accessibility	WCAG-compliant checks for color contrast, missing alt text, keyboard navigation traps, ARIA attributes, and more.
Performance	Lighthouse-driven analysis of render-blocking resources, excessive DOM size, unoptimized images, and other factors affecting page speed.
SEO	Evaluates on-page SEO factors (like metadata, headings, and link structure) and suggests improvements for better search visibility.
Best Practices	Checks for general best practices in web development.
NextJS Audit	Injects a prompt used to perform a NextJS audit.
Audit Mode	Runs all auditing tools in a sequence.

Audit Type	Description
Debugger Mode	Runs all debugging tools in a sequence.

🛠️ Using Audit Tools

✅ Before You Start

Ensure you have:

- An **active tab** in your browser
- The **BrowserTools extension enabled**

▶️ Running Audits

Headless Browser Automation:
Puppeteer automates a headless Chrome instance to load the page and collect audit data, ensuring accurate results even for SPAs or content loaded via JavaScript.

The headless browser instance remains active for **60 seconds** after the last audit call to efficiently handle consecutive audit requests.

Structured Results:
Each audit returns results in a structured JSON format, including overall scores and detailed issue lists. This makes it easy for MCP-compatibl clients to interpret the findings and present actionable insights.

The MCP server provides tools to run audits on the current page. Here are example queries you can use to trigger them:

Accessibility Audit (`runAccessibilityAudit`)

Ensures the page meets accessibility standards like WCAG.

> **Example Queries:**
>
> - "Are there any accessibility issues on this page?"
> - "Run an accessibility audit."
> - "Check if this page meets WCAG standards."

Performance Audit (`runPerformanceAudit`)

Identifies performance bottlenecks and loading issues.

> **Example Queries:**
>
> - "Why is this page loading so slowly?"
> - "Check the performance of this page."
> - "Run a performance audit."

SEO Audit (`runSEOAudit`)

Evaluates how well the page is optimized for search engines.

> **Example Queries:**
>
> - "How can I improve SEO for this page?"
> - "Run an SEO audit."
> - "Check SEO on this page."

Best Practices Audit (`runBestPracticesAudit`)

Checks for general best practices in web development.

> **Example Queries:**

> - "Run a best practices audit."
> - "Check best practices on this page."
> - "Are there any best practices issues on this page?"

Audit Mode (`runAuditMode`)

Runs all audits in a particular sequence. Will run a NextJS audit if the framework is detected.

> **Example Queries:**
>
> - "Run audit mode."
> - "Enter audit mode."

NextJS Audits (`runNextJSAudit`)

Checks for best practices and SEO improvements for NextJS applications

> **Example Queries:**
>
> - "Run a NextJS audit."
> - "Run a NextJS audit, I'm using app router."
> - "Run a NextJS audit, I'm using page router."

Debugger Mode (`runDebuggerMode`)

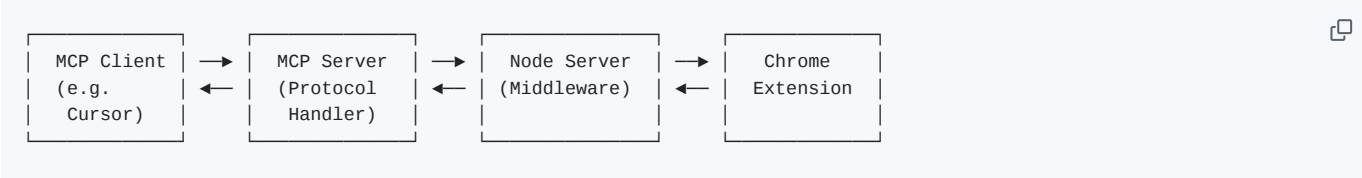
Runs all debugging tools in a particular sequence

> **Example Queries:**
>
> - "Enter debugger mode."

Architecture

There are three core components all used to capture and analyze browser data:

1. **Chrome Extension**: A browser extension that captures screenshots, console logs, network activity and DOM elements.
2. **Node Server**: An intermediary server that facilitates communication between the Chrome extension and any instance of an MCP server.
3. **MCP Server**: A Model Context Protocol server that provides standardized tools for AI clients to interact with the browser.



Model Context Protocol (MCP) is a capability supported by Anthropic AI models that allow you to create custom tools for any compatible client. MCP clients like Claude Desktop, Cursor, Cline or Zed can run an MCP server which "teaches" these clients about a new tool that they can use.

These tools can call out to external APIs but in our case, **all logs are stored locally** on your machine and NEVER sent out to any third-party service or API. BrowserTools MCP runs a local instance of a NodeJS API server which communicates with the BrowserTools Chrome Extension.

All consumers of the BrowserTools MCP Server interface with the same NodeJS API and Chrome extension.

Chrome Extension

- Monitors XHR requests/responses and console logs
- Tracks selected DOM elements
- Sends all logs and current element to the BrowserTools Connector
- Connects to Websocket server to capture/send screenshots
- Allows user to configure token/truncation limits + screenshot folder path

Node Server

- Acts as middleware between the Chrome extension and MCP server
- Receives logs and currently selected element from Chrome extension
- Processes requests from MCP server to capture logs, screenshot or current element
- Sends Websocket command to the Chrome extension for capturing a screenshot
- Intelligently truncates strings and # of duplicate objects in logs to avoid token limits
- Removes cookies and sensitive headers to avoid sending to LLMs in MCP clients

MCP Server

- Implements the Model Context Protocol
- Provides standardized tools for AI clients
- Compatible with various MCP clients (Cursor, Cline, Zed, Claude Desktop, etc.)

Installation

Installation steps can be found in our documentation:

- BrowserTools MCP Docs

Usage

Once installed and configured, the system allows any compatible MCP client to:

- Monitor browser console output
- Capture network traffic
- Take screenshots
- Analyze selected elements
- Wipe logs stored in our MCP server
- Run accessibility, performance, SEO, and best practices audits

Compatibility

- Works with any MCP-compatible client

Releases 3

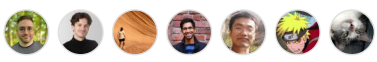
v1.2.0 - SEO, Performance + NextJS Audits; Screenshot Auto-Paste; Bug fixes Latest
on Mar 10, 2025

+ 2 releases

Packages

No packages published

Contributors 7



Languages

JavaScript 65.6% **TypeScript** 31.9% **HTML** 2.5%