BrowserWatcher

Boni García

Version 1.2.0, 27-06-2022

Table of Contents

Chapter 1. Motivation

When an automated end-to-end test (e.g., with Selenium WebDriver) fails, it can be challenging to discover the underlying error cause. A common strategy is to gather the browser console logs to discard problems on the client-side. The problem is the browser log gathering feature is not standard in the W3C WebDriver specification. The driver for Chrome (i.e., chromedriver) provides a custom implementation for log gathering using LoggingPreferences. Nevertheless, this feature is not available in other drivers such as geckodriver (i.e., the driver for Firefox). A standard solution is proposed in the new W3C WebDriver BiDi, but this approach is not yet widely adopted in all browsers.

What is BrowserWatcher?

BrowserWatcher is a browser extension designed to monitor web browsers such as Chrome, Firefox, or Edge. Its features are cross-browser console log gathering/displaying, tab recording, Content Security Policy (CSP) disabling, and JavaScript/CSS injection.

Chapter 2. Setup

BrowserWatcher can be used in two different ways. First, it can be installed as an extension in any browser implementing the browser extension API (such as Chrome, Firefox, Edge, etc.). You can use the following files to install BrowserWatcher in your browser:

- browserwatcher-1.2.0.crx: Chrome Extension (CRX), for Chromium-based browsers, such as Chrome or Edge. To install it, you can drag and drop this file into the extensions page (*chrome://extensions/*).
- browserwatcher-1.2.0.xpi: XPInstall (XPI), for Firefox. To install it, visit the *about:addons* page and click on *Install Add-on From File...*.

Second, you can use BrowserWatcher automatically through WebDriverManager. See the section about WebDriverManager for a basic introduction to this use.

Chapter 3. Features

Once BrowserWatcher is installed in a browser, you can see its logo in the browser navigation bar, as follows:

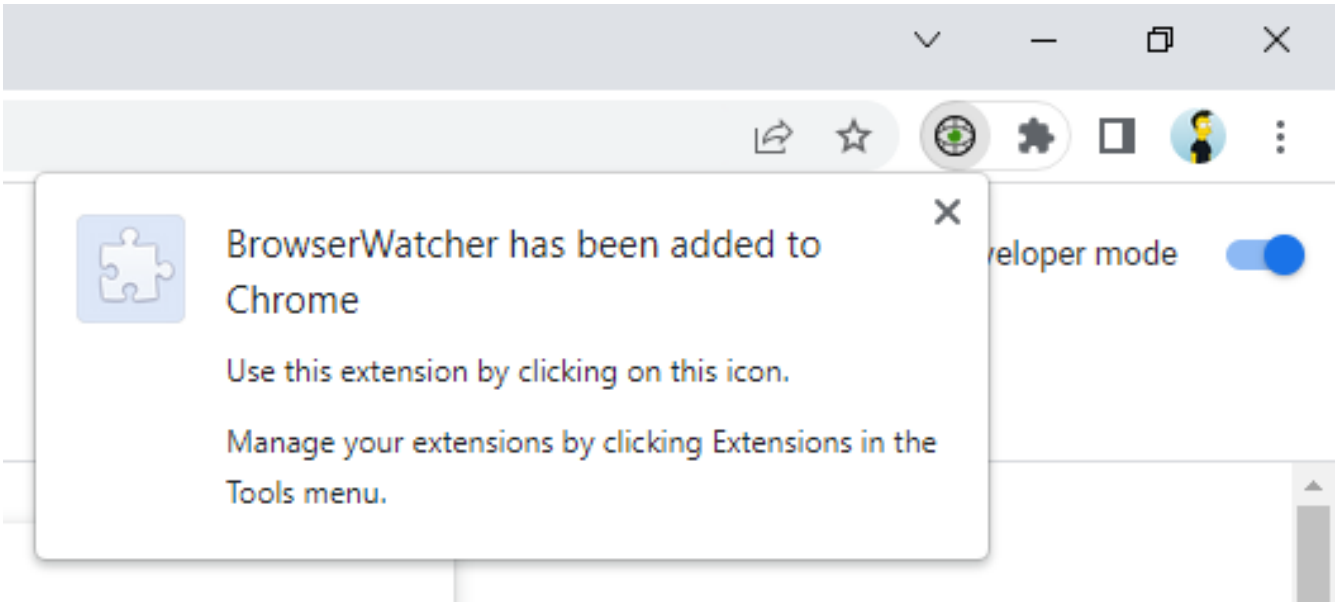

Figure 1. BrowserWatcher installed in Chrome

3.1. Console Log Gathering

BrowserWatcher allows gathering the browser console following a cross-browser approach. Internally, it uses a *monkey patching* technique to override the JavaScript object `console`. In particular, the following methods are overridden: `log`, `warn`, `error`, `info`, `dir`, `time`, `timeEnd`, `table`, and `count`. In addition, it implements several JavaScript listeners for the following events: `error` (JavaScript error trace), `unhandledrejection` (when a JavaScript *Promise* with no rejection handler is rejected), `securitypolicyviolation` (when the content security policy is violated), and `xhr-error` (error response from `XMLHttpRequest`). This way, the gathered logs are accessible in the property `_bwLogs` in the `console` object. The following picture shows an example of the gathered logs of a the sample page:

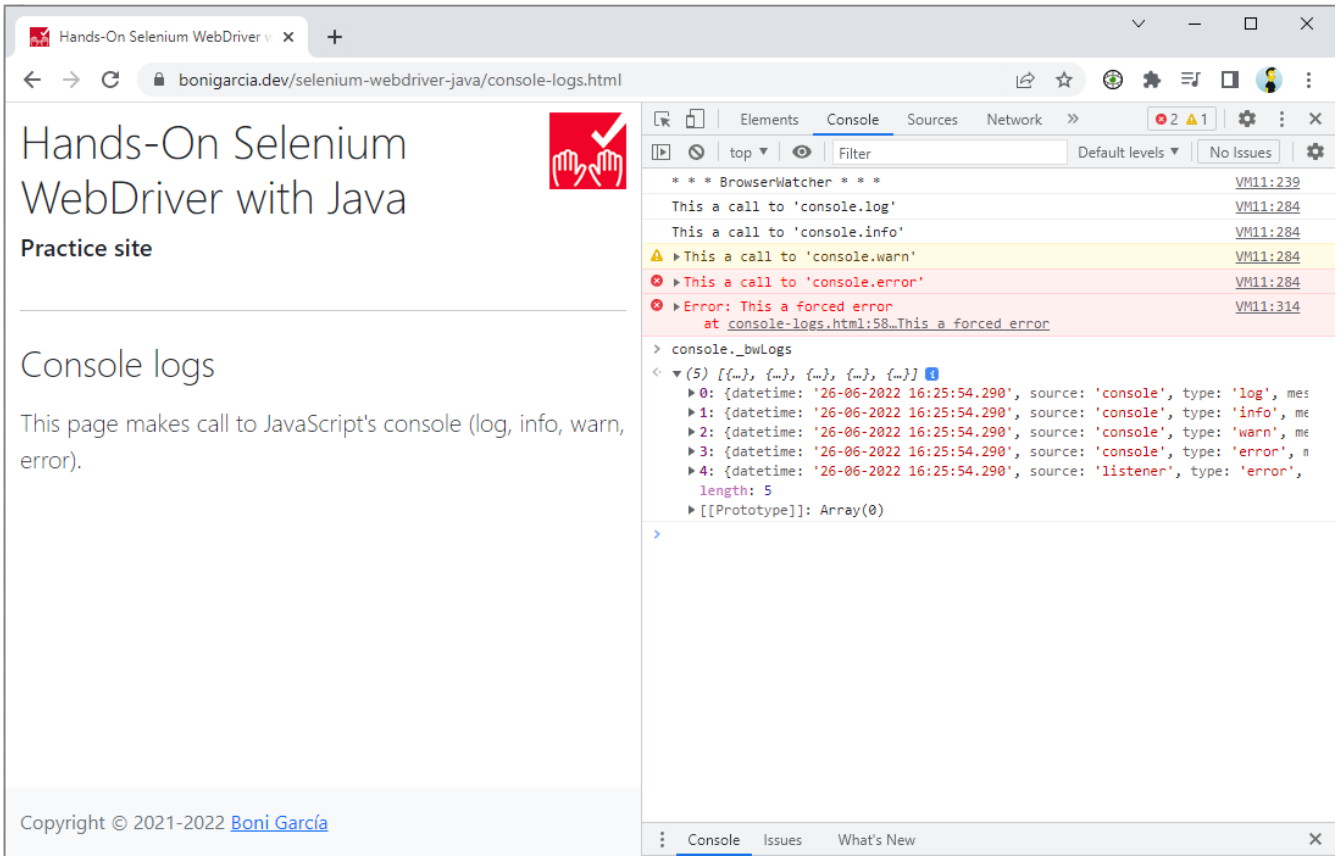

Figure 2. Console log gathering example

BrowserWatcher has a configuration page, displayed when clicked in its icon. This page has three toggle buttons to set up different features. The first one allows enabling (and disabling) the log gathering feature.

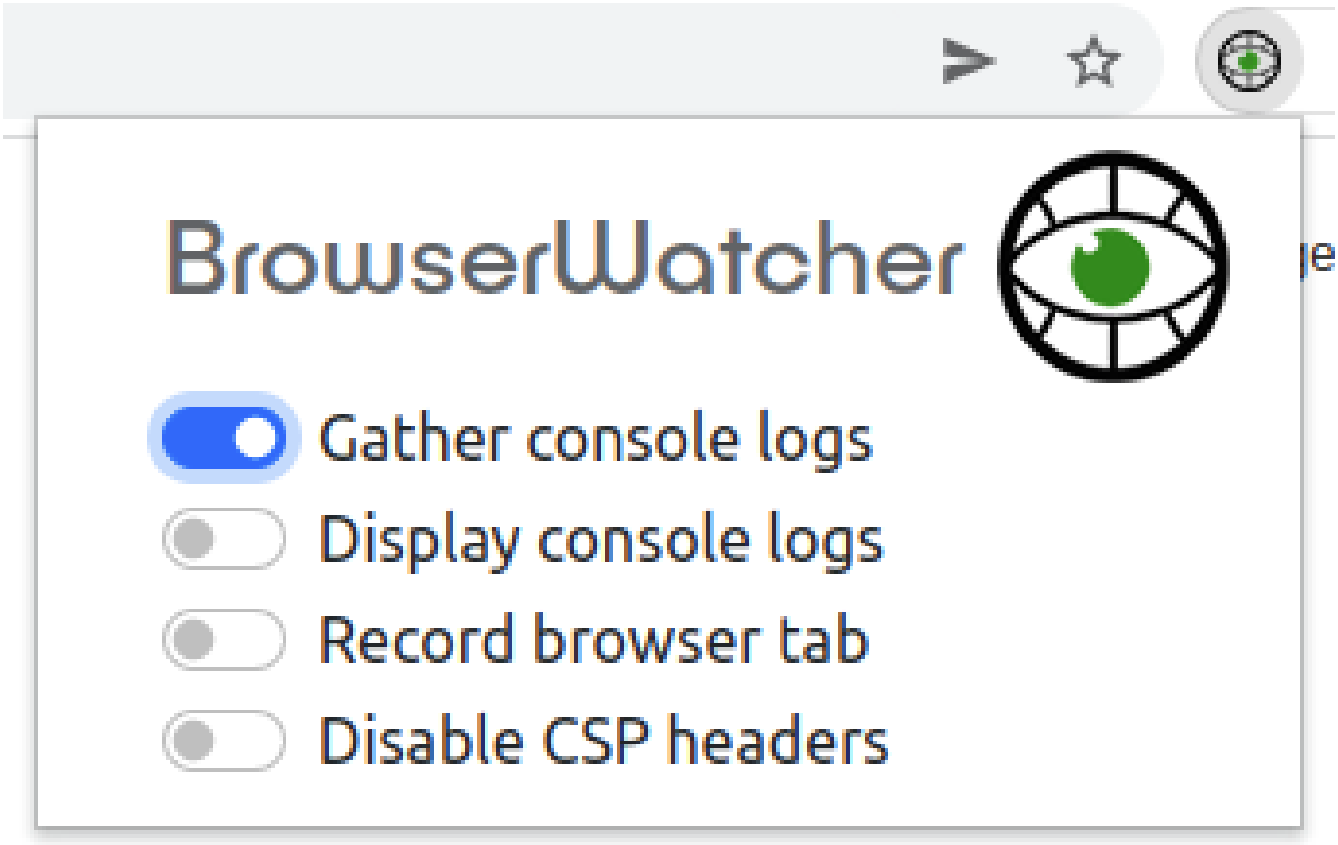

Figure 3. BrowserWatcher popup configuration page (console log gathering)

3.2. Console Log Displaying

In addition to log gathering, BrowserWatcher allows displaying the console logs (and listened events) as dialog notifications on the page. This feature can be enabled using the following button:

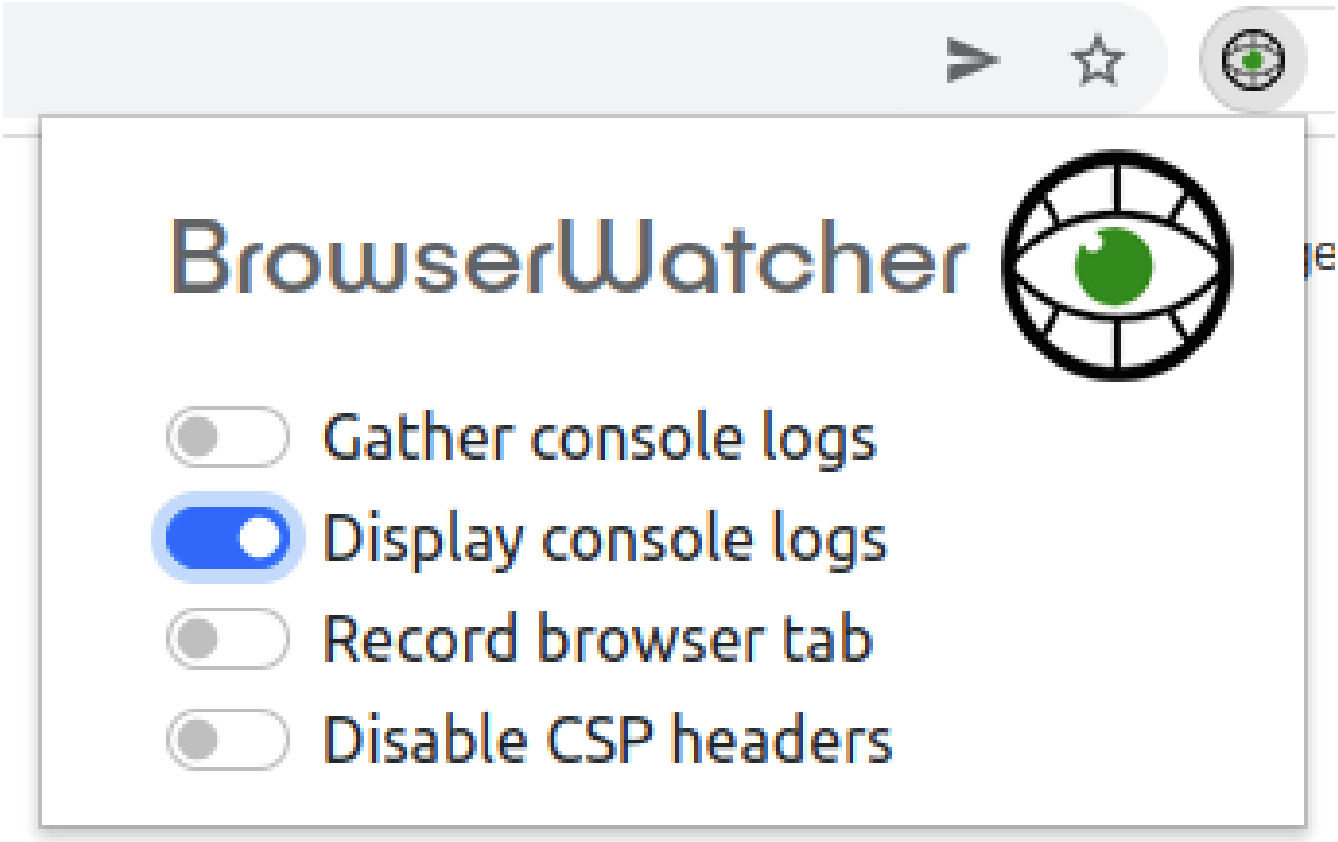

Figure 4. BrowserWatcher popup configuration page (console log displaying)

The following picture shows an example of these notifications:

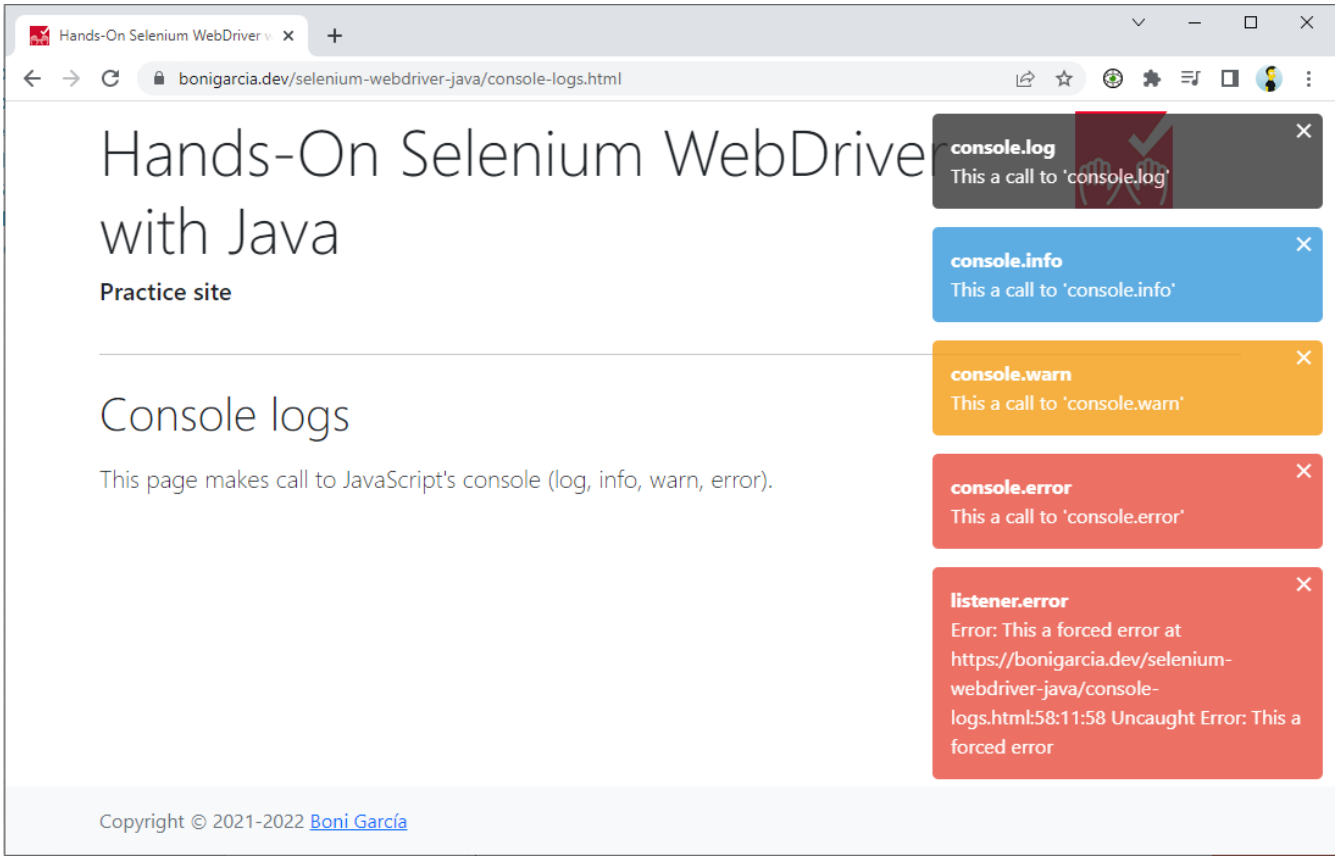

Figure 5. Display logs example

3.3. Tab recording

BrowserWatcher allows recording a browser tab. This feature requires a page loaded in the current tab (in other words, it cannot record empty or configuration pages). Then, it can be started using the following button:

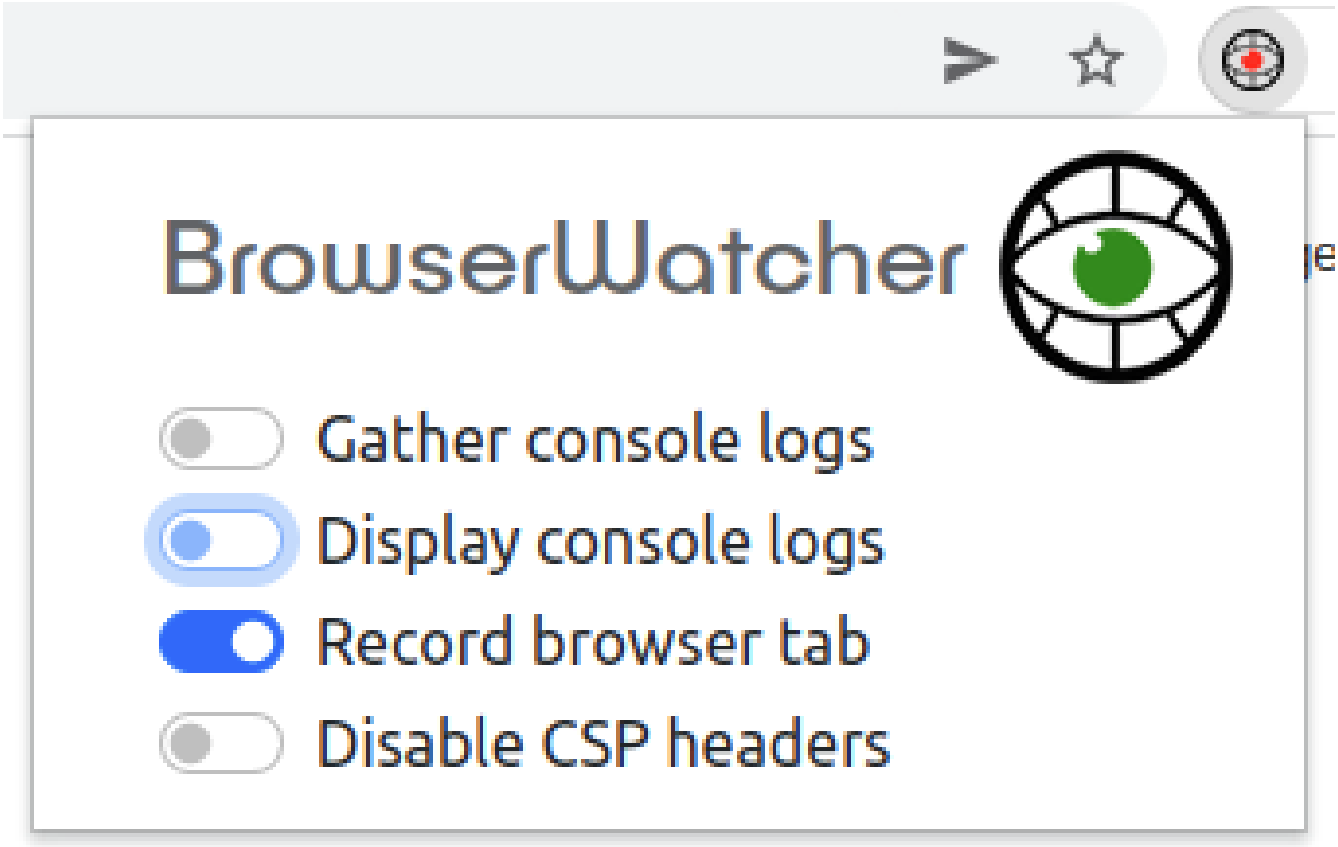

Figure 6. BrowserWatcher popup configuration page (recording)

The recording is stopped using the same button. The recording file will be a WEBM file, available in your *Downloads* folder.

This feature is based on the API tabCapture. Therefore, this feature is not available in Firefox since this API is not implemented by Firefox yet.

There are some alternative ways to start and stop the recordings. First, by using the keyboard shortcut `Alt+R` (for starting the recording) and `Alt+W` (for stopping the recording). Second, by executing the following JavaScript commands:

- `window.postMessage({ type: "startRecording" });` : For starting the recording. The recording file name will be auto generated, composed by the system timestamp plus the prefix `-browser-recording.webm`.
- `window.postMessage({ type: "startRecording", name: "myrecording" });`: For starting the recording and specifyng a custom file recording (`myrecording.webm` in this example).
- `window.postMessage({ type: "startRecording" });` : For stopping the recording.

3.4. Disabling CSP

Content Security Policy (CSP) is the name of an HTTP response header that browsers use to improve the security of web pages. CSP helps to protect from attacks such as cross-site scripting (XSS). Nevertheless, developers might want to disable the CSP headers received from the server for testing purposes. For this reason, BrowserWatcher allows bypassing these CSP headers. This feature (i.e., disabling CSP headers) is enabled by clicking the following button:

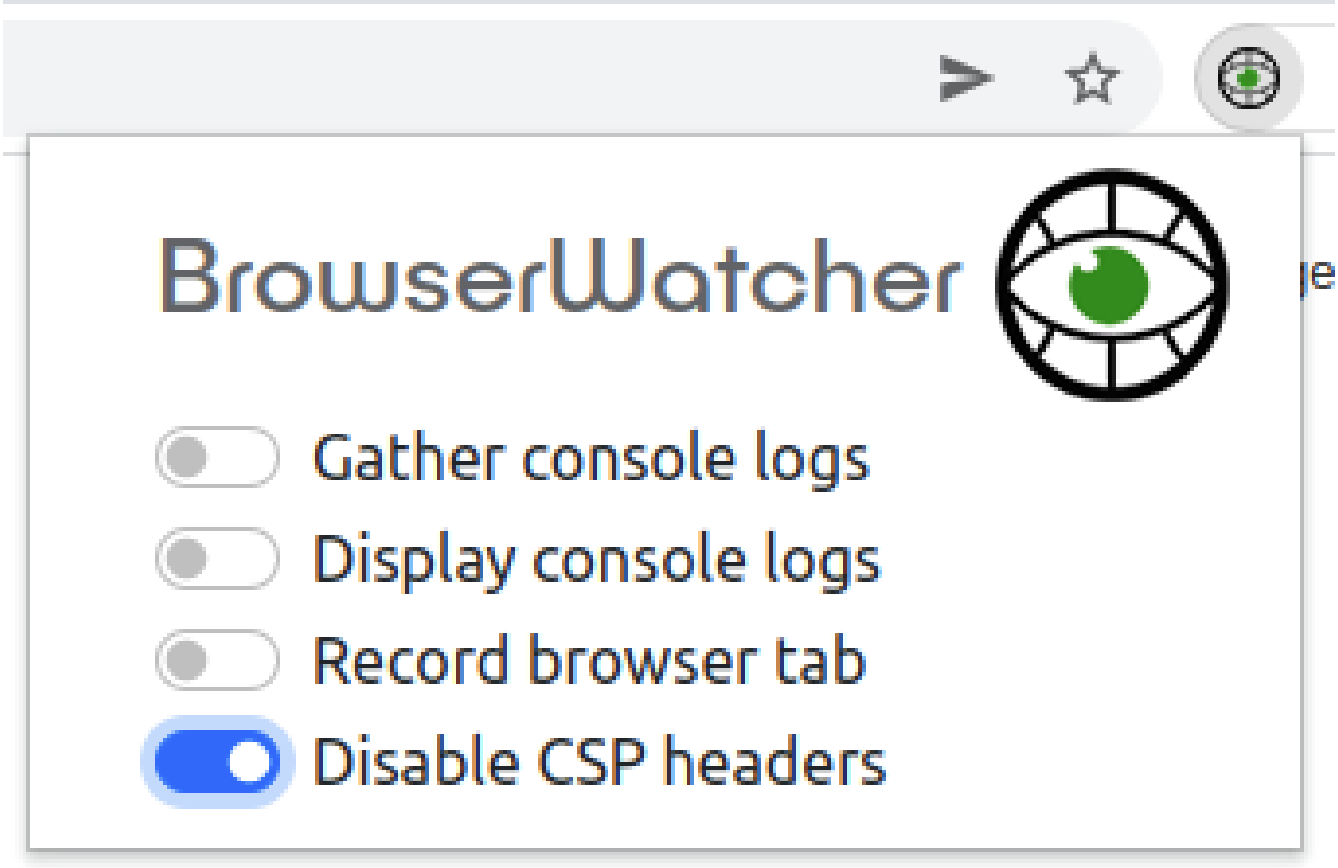

Figure 7. BrowserWatcher popup configuration page (disable CSP headers)

In addition, this feature can be enabled/disabled by executing the following JavaScript commands:

- `window.postMessage({ type: "disableCsp" });` : For disabling CSP.
- `window.postMessage({ type: "enableCsp" });` : For enabling CSP (as it was by default).

3.5. JavaScript and CSS injection

BrowserWatcher has an *options* page, available by clicking on the following menu option:

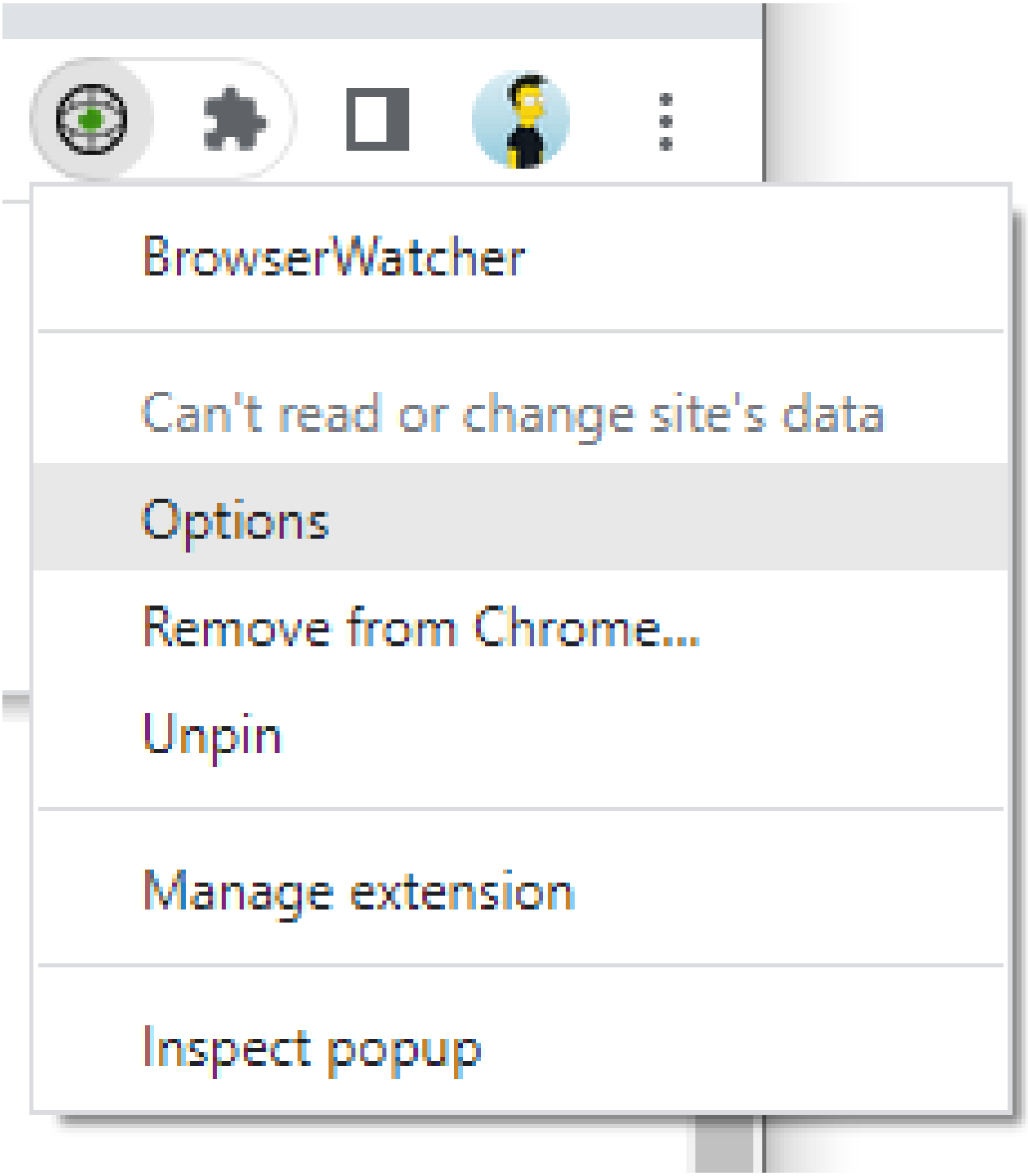

Figure 8. BrowserWatcher menu option

This options page allows injecting JavaScript code, libraries, and CSS stylesheets. The following screenshot shows an example that injects an online JavaScript library to highlight the mouse. The resulting mouse pointer is shown subsequently.

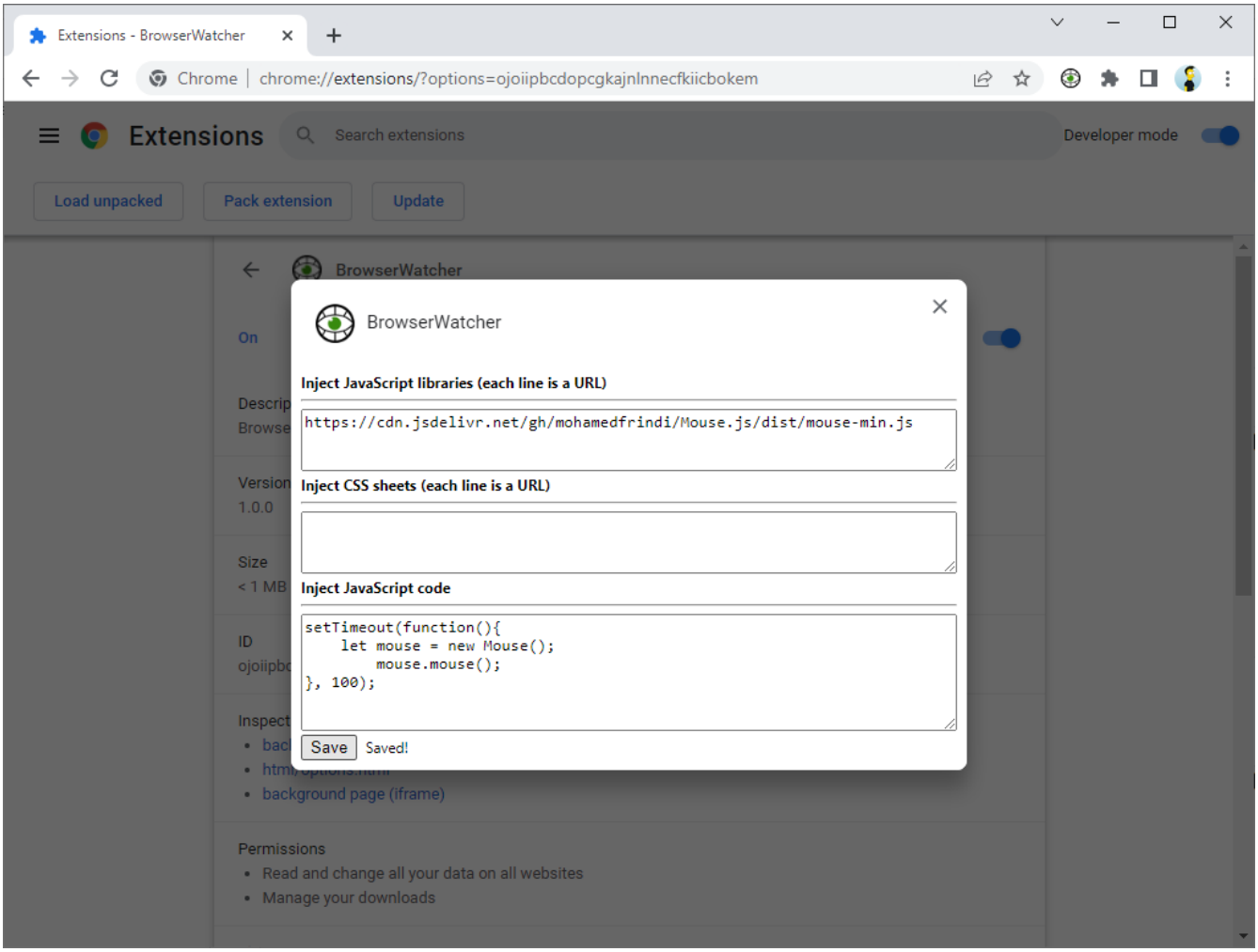

Figure 9. BrowserWatcher options page (example for highlighting mouse)

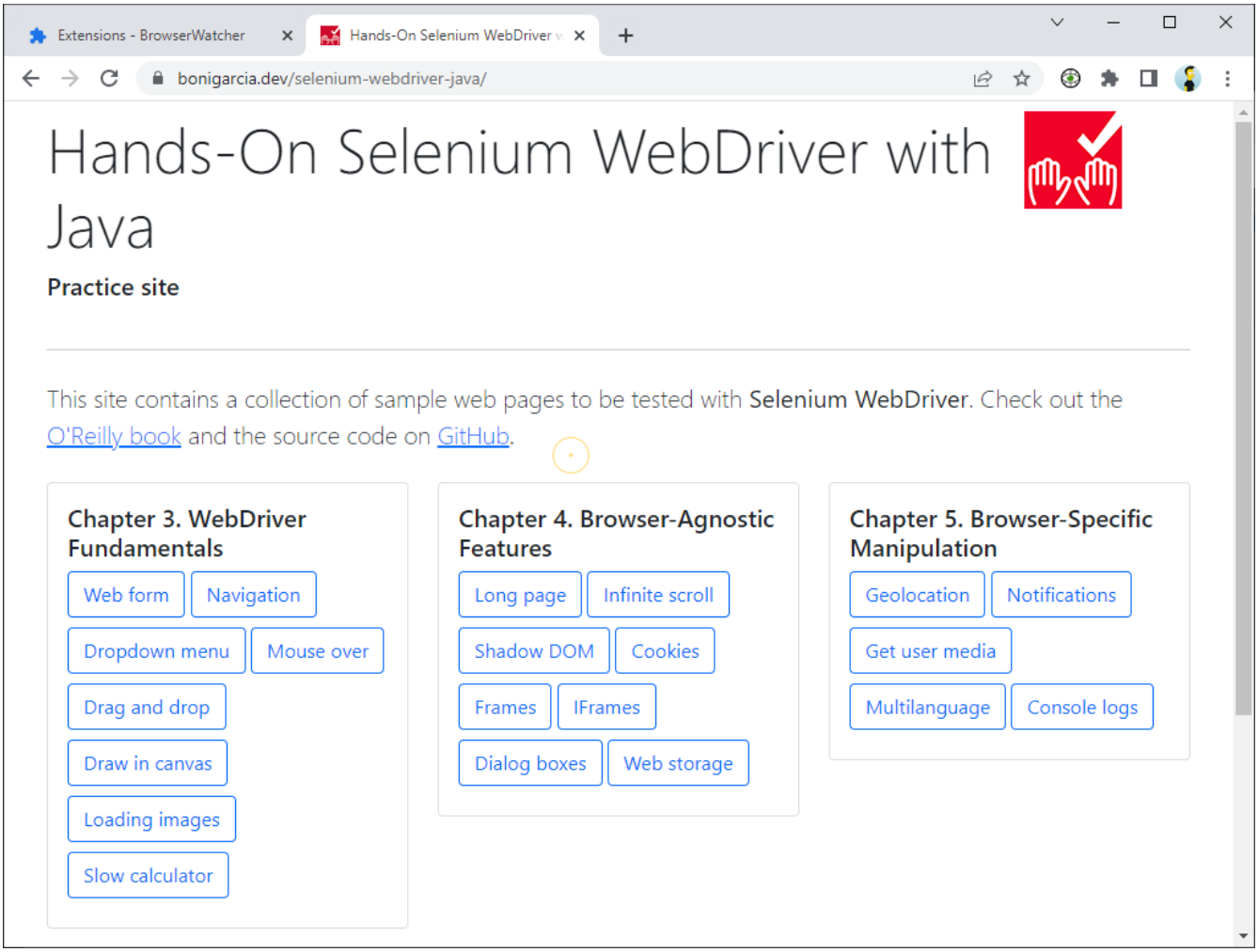

Figure 10. JavaScript and CSS injection page example (highlighting mouse)

Alternatively, this feature can be enabled by executing the following JavaScript commands:

- `window.postMessage({ type: "injectJavaScriptCode", js: "jsCode" });`
- `window.postMessage({ type: "injectJavaScriptLibs", lib: "jsLib" });`
- `window.postMessage({ type: "injectCssSheets", css: "cssSheet" });`

Chapter 4. WebDriverManager

As of version 5.2.0, WebDriverManager provides seamless integration with BrowserWatcher. In other words, WebDriverManager allows injecting BrowserWatcher in browsers controlled with Selenium WebDriver and created through WebDriverManager with the methods `.watch()` and `.create()`. These browsers allow log gathering, displaying, and tab recording through BrowserWatcher.

For example, the following test shows some basic use of log gathering using WebDriverManager and BrowserWatcher:

```
class GatherLogsChromeTest {

    static final Logger log = getLogger(lookup().lookupClass());

    WebDriverManager wdm = WebDriverManager.chromedriver().watch();
    WebDriver driver;

    @BeforeEach
    void setup() {
        driver = wdm.create();
    }

    @AfterEach
    void teardown() {
        driver.quit();
    }

    @Test
    void test() {
        driver.get(
                "https://bonigarcia.dev/selenium-webdriver-java/console-logs.html");

        List<Map<String, Object>> logMessages = wdm.getLogs();

        for (Map<String, Object> map : logMessages) {
            log.debug("[{}] [{}.{}] {}", map.get("datetime"),
                    map.get("source").toString().toUpperCase(),
                    String.format("%1$-7s",
                            map.get("type").toString().toUpperCase()),
                    map.get("message"));
        }

        assertThat(logMessages).hasSize(5);
    }

}
```

Please visit its documentation page for further details.

Chapter 5. Further Documentation

There are other resources related to automated testing you can find helpful. For instance, the following books:

- García, Boni. Hands-On Selenium WebDriver with Java. *O'Reilly Media, Inc.*, 2022.
- García, Boni. Mastering Software Testing with JUnit 5. *Packt Publishing Ltd*, 2017.

Or the following journal papers:

- García, Boni, et al. "Selenium-Jupiter: A JUnit 5 extension for Selenium WebDriver." *Journal of Systems and Software* (2022): 111298.
- García, Boni, et al. "Automated driver management for Selenium WebDriver." *Empirical Software Engineering* 26.5 (2021): 1-51.
- García, Boni, et al. "A survey of the Selenium ecosystem." *Electronics* 9.7 (2020): 1067.
- García, Boni, et al. "Assessment of QoE for video and audio in WebRTC applications using full-reference models." *Electronics* 9.3 (2020): 462.
- García, Boni, et al. "Practical evaluation of VMAF perceptual video quality for WebRTC applications." *Electronics* 8.8 (2019): 854.
- García, Boni, et al. "WebRTC testing: challenges and practical solutions." *IEEE Communications Standards Magazine* 1.2 (2017): 36-42.
- García, Boni, and Juan Carlos Dueñas. "Web browsing automation for applications quality control." *Journal of web engineering* (2015): 474-502.

Chapter 6. About