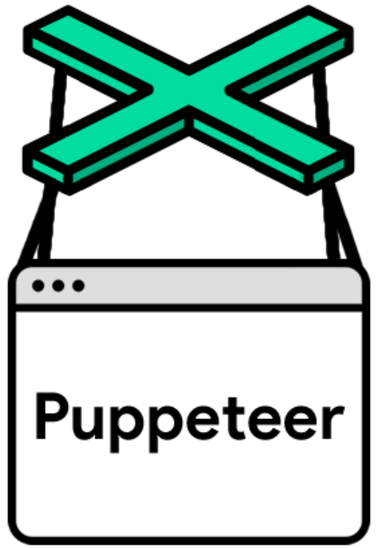
Puppeteer beyond testing

Mieszko Wawrzyniak

What is Puppeteer?

Puppeteer is a Node library which provides a high-level API to control Chrome or Chromium over the DevTools Protocol. Puppeteer runs headless by default, but can be configured to run full (non-headless) Chrome or Chromium.

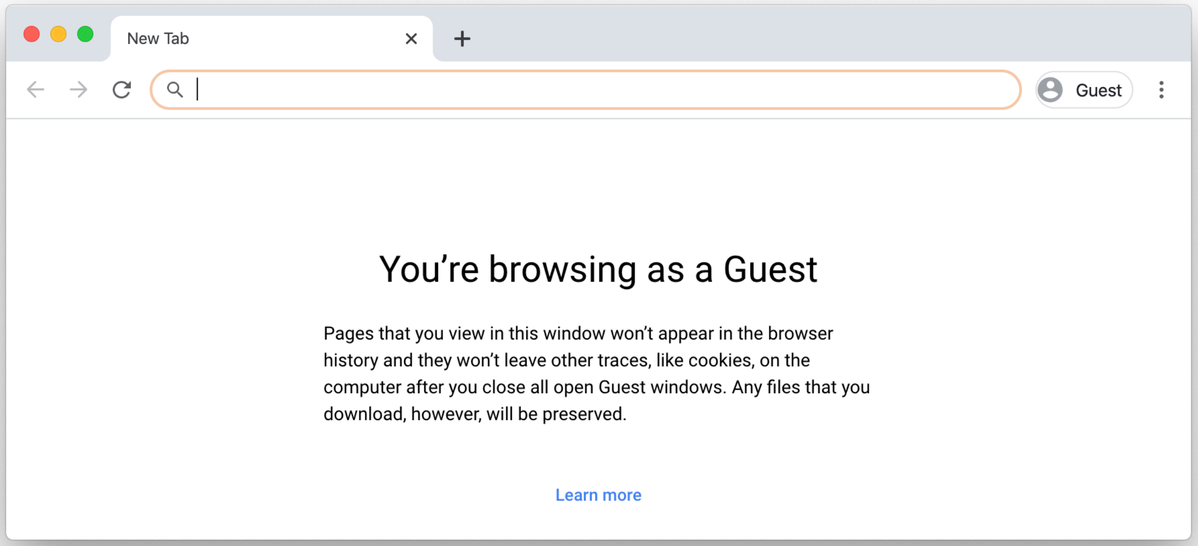


Project repo https://github.com/puppeteer/puppeteer

Playground https://try-puppeteer.appspot.com

Puppeteer is just a Chromium browser API

It can do the same staff as a normal user would but automatically



Hello World

This code will open new Chromium window, open new tab and navigate to https://example.com

```
1    const browser = await puppeteer.launch();
2    const page = await browser.newPage();
3
4    await page.goto("https://example.com", { waitUntil: "networkidle2" });
```

We can automate manual testing

Imagine that you have a manual tester who before each release have to repeat the same work in order to ensure that your web page is still loading and working correctly

Puppeteer can handle all that work on CI

✓ ✓ ✓

E2E testing

Puppeteer can login into your web application and perform the same actions as a customer would

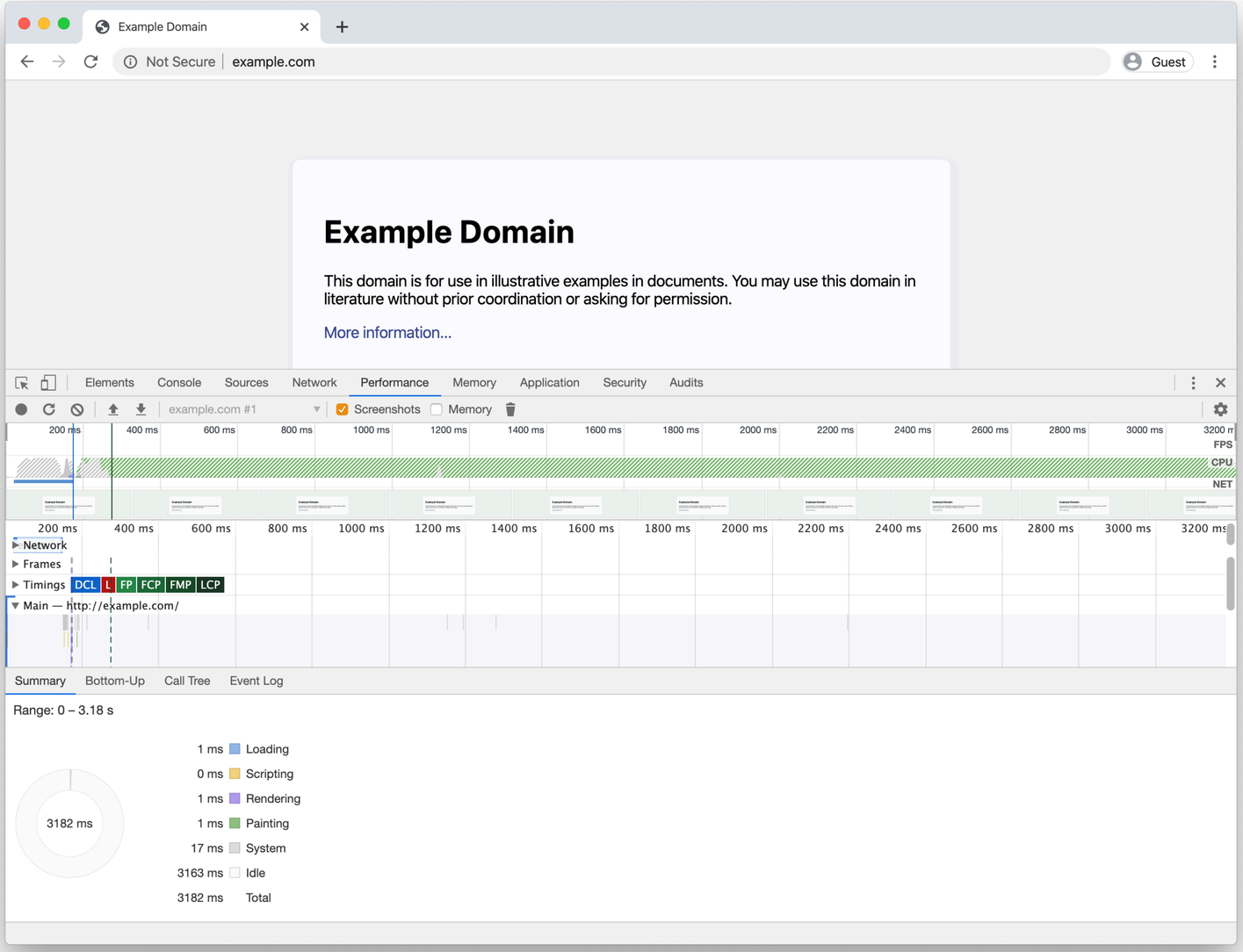
```
test(
  "Header of the page",
  async () => {
    const h1Handle = await page.$(".learn_header");
    const html = await page.evaluate(h1Handle => h1Handle.innerHTML, h1Handle);

    expect(html).toBe("What will you learn");
  },
  timeout
);
```

page.$ is equivalent to document.querySelector

https://medium.com/touch4it/end-to-end-testing-with-puppeteer-and-jest-ec8198145321

Performance budget enforcement



window.performance.timing extraction

```
1   const browser = await puppeteer.launch();
2
3   const page = await browser.newPage();
4   await page.goto("https://github.com");
5
6   const timing = await page.evaluate(() ⇒
7     JSON.stringify(window.performance.timing)
8   );
9
10  console.log(timing);
```


```
{
  "navigationStart": 1584197312355,
  "unloadEventStart": 0,
  "unloadEventEnd": 0,
  "redirectStart": 0,
  "redirectEnd": 0,
  "fetchStart": 1584197312355,
  "domainLookupStart": 1584197312375,
  "domainLookupEnd": 1584197312377,
  "connectStart": 1584197312377,
  "connectEnd": 1584197312582,
  "secureConnectionStart": 1584197312404,
  "requestStart": 1584197312582,
  "responseStart": 1584197312629,
  "responseEnd": 1584197312681,
  "domLoading": 1584197312637,
  "domInteractive": 1584197312775,
  "domContentLoadedEventStart": 1584197312775,
  "domContentLoadedEventEnd": 1584197312775,
  "domComplete": 1584197313375,
  "loadEventStart": 1584197313375,
  "loadEventEnd": 1584197313389
}
```

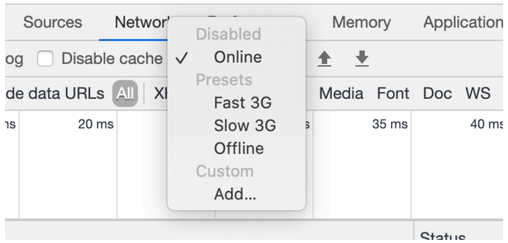
window.performance.timing extraction

We can use extracted timing to calculate performance metrics

```
const metrics = {
  dnsLookup: timing.domainLookupEnd - timing.domainLookupStart,
  tcpConnect: timing.connectEnd - timing.connectStart,
  request: timing.responseStart - timing.requestStart,
  response: timing.responseEnd - timing.responseStart,
  domLoaded: timing.domComplete - timing.domLoading,
  domInteractive: timing.domInteractive - timing.navigationStart,
  pageLoad: timing.loadEventEnd - timing.loadEventStart,
  fullTime: timing.loadEventEnd - timing.navigationStart
}
```

https://automationrhapsody.com/performance-testing-in-the-browser/

We can also emulate different network conditions

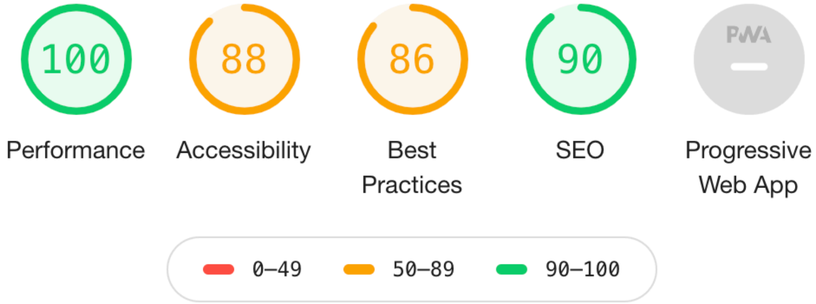



```
const client = await page.target().createCDPSession();
// await client.send('Network.enable'); // Already enabled by pptr.
await client.send("Network.emulateNetworkConditions", {
  offline: false,
  // values of 0 remove any active throttling. crbug.com/456324#c9
  latency: 800,
  downloadThroughput: Math.floor((1.6 * 1024 * 1024) / 8), // 1.6Mbps
  uploadThroughput: Math.floor((750 * 1024) / 8) // 750Kbps
});
```

https://github.com/puppeteer/examples/blob/master/lighthouse/throttling.js

Lighthouse

Lighthouse is an open-source, automated tool for improving the quality of web pages. You can run it against any web page, public or requiring authentication. It has audits for performance, accessibility, progressive web apps, SEO and more.




```
const {report} = await lighthouse(url, {
  port: remoteDebugPort,
  output: 'html',
  logLevel: 'info',
  disableNetworkThrottling: true,
  //disableCpuThrottling: true,
  //disableDeviceEmulation: true,
});
```

https://developers.google.com/web/tools/lighthouse

https://github.com/puppeteer/examples/blob/master/lighthouse/throttling.js

Generating PDFs

It's often hard to find a libry for dynamic PDF creation. In that case Puppeteer comes to the rescue. It will convert our HTML documents into PDFs.

```
1   const puppeteer = require("puppeteer");
2
3   (async () => {
4     const browser = await puppeteer.launch();
5     const page = await browser.newPage();
6     await page.goto("https://news.ycombinator.com", {
7       waitUntil: "networkidle2"
8     });
9     await page.pdf({ path: "hn.pdf", format: "A4" });
10
11    await browser.close();
12  })();
```

https://github.com/puppeteer/puppeteer

Generating screenshots

Puppeteer is also able to generate screenshots

```
const puppeteer = require("puppeteer");

(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto("https://example.com");
  await page.screenshot({ path: "example.png" });

  await browser.close();
})();
```

https://github.com/puppeteer/puppeteer

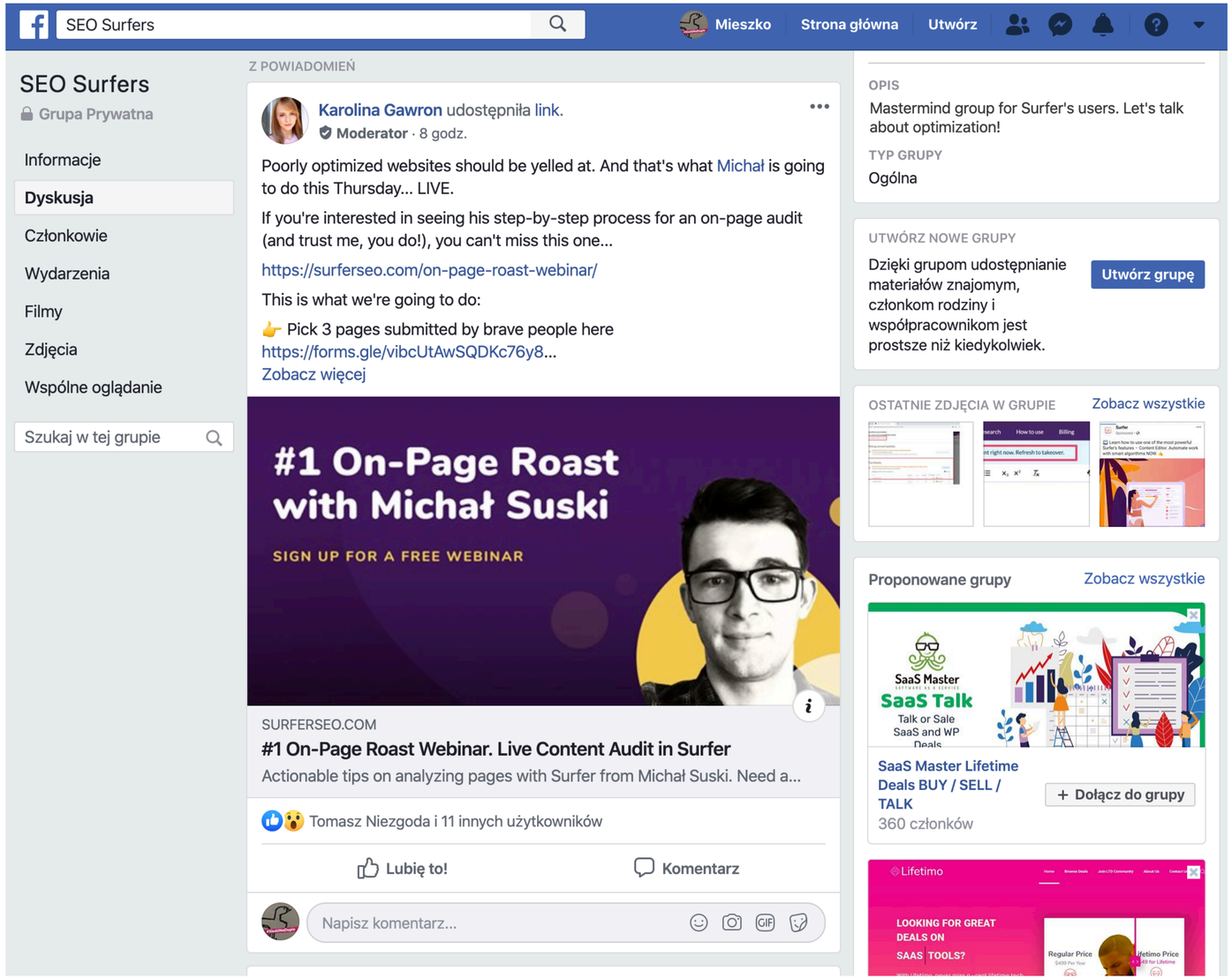
Gotenberg

A Docker-powered stateless API for converting HTML, Markdown and Office documents to PDF.

It is using Chrome under the hood

https://github.com/thecodingmachine/gotenberg

Web Scraping



Puppeteer can be used for scraping web applications which aren’t Server Side Rendered and don’t have easily accessible API

Thanks to Puppeteer it is possible to extract content from for example Facebook Groups using DOM Selectors or Regex queries

Web Scraping

Web Scraping is wild 🔥 There is no do it all code snippet 😩

```ts
const getAttendanceWithRegex = (): string => {
  const elements = document.querySelectorAll('*');
  const strings = Array.prototype.map.call(elements, (el: HTMLElement) => el.textContent) as string[];

  const fullRegex = /^([\d.]+[kKmMbBtT]{0,}) (([gG]oing)|([wW]ent)) · ([\d.]+[kKmMbBtT]{0,}) [iI]nterested$/;
  const fullResult = strings.find(s => fullRegex.test(s));

  if (fullResult) {
    return fullResult;
  }

  const interestedRegex = /^([\d.]+[kKmMbBtT]{0,}) [iI]nterested$/;
  const interestedResult = strings.find(s => interestedRegex.test(s));

  if (interestedResult) {
    return `0 ${interestedResult}`;
  }

  return '0 0';
};
```

In order to scrape a website you need to render it like for PDF generation and evaluate some code using `page.evaluate` which will try to extract desired data

https://medium.com/swlh/an-introduction-to-web-scraping-with-puppeteer-3d35a51fdca0

puppeteer- 

We can use Adblock and other plugins in Puppeteer

```
// puppeteer-extra is a drop-in replacement for puppeteer,
// it augments the installed puppeteer with plugin functionality.
// Any number of plugins can be added through `puppeteer.use()`
const puppeteer = require('puppeteer-extra')

// Add stealth plugin and use defaults (all tricks to hide puppeteer usage)
const StealthPlugin = require('puppeteer-extra-plugin-stealth')
puppeteer.use(StealthPlugin())

// Add adblocker plugin to block all ads and trackers (saves bandwidth)
const AdblockerPlugin = require('puppeteer-extra-plugin-adblocker')
puppeteer.use(AdblockerPlugin({ blockTrackers: true }))
```

https://www.npmjs.com/package/puppeteer-extra

Server Side Rendering

Yeah, it's possible, it helps with SEO
But it may break your website

```
app.get('/', async (req, res, next) ⇒ {
  const {html, ttRenderMs} = await ssr(`${req.protocol}://${req.get('host')}/index.html`);
  // Add Server-Timing! See https://w3c.github.io/server-timing/.
  res.set('Server-Timing', `Prerender;dur=${ttRenderMs};desc="Headless render time (ms)"`);
  return res.status(200).send(html); // Serve prerendered page as response.
});

app.listen(8080, () ⇒ console.log('Server started. Press Ctrl+C to quit'));
```

https://developers.google.com/web/tools/puppeteer/articles/ssr

Server Side Rendering

There are ready solutions basing on headless Chrome

server.js

```
1    const prerender = require('prerender');
2    const server = prerender();
3    server.start();
```

nginx.conf

```
if ($prerender = 1) {

    #setting prerender as a variable forces DNS resolution
    set $prerender "service.prerender.io";
    rewrite .* /$scheme://$host$request_uri? break;
    proxy_pass http://$prerender;
}
if ($prerender = 0) {
    rewrite .* /index.html break;
}
```

https://github.com/prerender/prerender

Gotchas

page.evaluate **is tricky**

page.evaluate(pageFunction[, ...args])

- `pageFunction` <function|string> Function to be evaluated in the page context
- `...args` <...Serializable|JSHandle> Arguments to pass to `pageFunction`
- returns: <Promise<Serializable>> Promise which resolves to the return value of `pageFunction`

It doesn’t have access to your current scope / context
Thats why it accepts extra arguments

Gotchas

page.evaluate **is tricky**

BAD

```
1    const a = 123
2
3    const res = page.evaluate(() ⇒ 2 * a)
4    res ═══ 2 * 123
```

GOOD

```
1    const a = 123
2
3    const res = page.evaluate((localA) ⇒ 2 * localA, a)
4    res ═══ 2 * 123
```

But why is that?

Gotchas

page.evaluate **is tricky**
It's sending function using Function.prototype.toString

```
> foo = (() => { const a = 123; return () => a * 2})()
<· () => a * 2
> foo()
<· 246
> bar = foo.toString()
<· "() => a * 2"
> eval(bar)()
✕ ▶ Uncaught ReferenceError: a is not defined
      at eval (eval at <anonymous> (local-ntp.html:1), <anonymous>:1:9)
      at <anonymous>:1:10
```

Your local variables will not be present in the page context

Gotchas

In JavaScript functions can be overwritten
Which means they WILL be overwritten

```
const elements = document.querySelectorAll('*')
const strings = Array.prototype.map.call(elements, (el) ⇒ el.textContent)
```

This snippet won't work on every page because it's relaying on Array.prototype.map **and** document.querySelectorAll
In order to mitigate this issue iframe **can be used**

```
const foo = () ⇒ 123 * 2

page.evaluate((localFoo) ⇒ {
  const sandbox = document.createElement('iframe')
  document.body.appendChild(sandbox)
  return sandbox.contentWindow.eval(localFoo)
}, foo.toString())
```

Gotchas

In JavaScript functions can be overwritten
Which means they WILL be overwritten

```
> Array.prototype.map = () => console.log(1230)
<- () => console.log(1230)
> [1, 2].map(x => x * 2)
  1230
<- undefined
> sandbox.contentWindow.eval('[1, 2].map(x => x * 2)')
<- ▶ (2) [2, 4]
```

Inside sandbox **page's** document **object can be accessed via** window.parent.document

```
> sandbox.contentWindow.eval('window.parent.document.querySelector("title").innerText')
<- "html - Invoking JavaScript code in an iframe from the parent page - Stack Overflow"
```

Puppeteer for Firefox

⚠️ **BEWARE:** Experimental.

https://github.com/puppeteer/puppeteer/tree/master/experimental/puppeteer-firefox

Playwright

Playwright is a Node library to automate the Chromium, WebKit and Firefox browsers with a single API. It enables cross-browser web automation that is ever-green, capable, reliable and fast.

	Version	Linux	macOS	Win
Chromium	82.0.4057.0	✅	✅	✅
WebKit	13.0.4	✅	✅	✅
Firefox	73.0b13	✅	✅	✅

```
const playwright = require("playwright");

(async () => {
  for (const browserType of ["chromium", "firefox", "webkit"]) {
    const browser = await playwright[browserType].launch();
    const context = await browser.newContext();
    const page = await context.newPage();
    await page.goto("http://whatsmyuseragent.org/");
    await page.screenshot({ path: `example-${browserType}.png` });
    await browser.close();
  }
})();
```

https://github.com/microsoft/playwright

Thanks 🦠

Mieszko 🏄‍♂️ Wawrzyniak

LinkedIn https://www.linkedin.com/in/mieszko-wawrzyniak
Github https://github.com/kaaboaye
Mail mieszkowaw@gmail.com
Surfer SEO https://surferseo.com/

SURFER