# Playwright, a better Puppeteer?

I love Puppeteer, it's almost the perfect tool to test sites and do screenshots of the projects we're working on. It was originally a Chrome only tool, meaning it would only work with headless Chromium; it then graduated to working with headless Firefox as well... but Safari/WebKit is missing.

This is the most basic example of using Puppeteer. It will navigate to a site and create a screenshot of the above-the-fold content.

We can use Puppeteer to navigate the pages and simulate interactions

```
const puppeteer = require('puppeteer')
const screenshot = 'github-chromium.png';

(async () => {
  const browser = await puppeteer.launch({
    product: 'chrome',
    headless: true
  })
  const page = await browser.newPage()
  await page.goto('https://publishing-project.rivendellweb.net')
  await page.screenshot({ path: screenshot })
  browser.close()
  console.log('See screenshot: ' + screenshot)
})()
```

## Note

I also experienced a weird issue when testing multi browser support in Puppeteer. Whatever browser engine was installed first would work but the other engine would not install at all so Puppeteer scripts that use multiple engines failed consistently for me

I filed an issue in the Puppeteer repository and will update this post when/if I hear anything.

That's where Playwright comes in. Playwright comes bundled with Chromium, Firefox Nightly and WebKit (the open source version of Safari) out of the box so testing in multiple browsers and doing visual testing with different device emulations becomes a breeze.

The basic scripts gets a little more repetitive as we're doing the same thing in three different browsers, but it looks like this:

For each browser we do the following:

1. Launch the browser
2. Create a new page
3. Navigate to the URL we want to test
4. Create a screenshot of the page
5. Close the browser

```
const {
  chromium,
  firefox,
  webkit,
} = require('playwright');

(async () => {
  const browser = await chromium.launch();
  const page = await browser.newPage();
  await page.goto('http://whatsmyuseragent.org/');
  await page.screenshot({ path: 'http://whatsmyuseragent.org/'`example-chr
  const browser2 = await firefox.launch();
  const page2 = await browser2.newPage();
  await page2.goto('http://whatsmyuseragent.org/');
  await page2.screenshot({ path: `example-firefox.png` });
  aw'http://whatsmyuseragent.org/'`example-firefox.png`it.launch();
  const page3 = await browser3.newPage();
  await page3.goto('http://whatsmyuseragent.org/');
```

```
  await page3.screenshot({ path: `example-webkit.png` });
  await browser3.close();
})();
'http://whatsmyuseragent.org/'`example-webkit.png` });
  await browser3.close();
})();
```

If that's all we could do it would be a fun experiment. But there's a lot more we can do.

One of my favorite examples is to use Chromium's device emulation to simulate a set of mobile devices so we can see an approximation of what our content will look like.

It is very important to point out that **using Playwright, Puppeteer or any automated browseing tool does not replace testing on real devices**. This feature on Playwright and Puppeteer is equivalent to the device emulation on DevTools.

```
(async () => {
  const { chromium, devices } = require('playwright');
  const browser = await chromium.launch();

  const pixel2 = devices['Pixel 2'];
  const context = await browser.newContext({
    ...pix'Pixel 2';

  const page = await context.newPage();
  await page.goto('http://whatsmyuseragent.org/');
  await page.screenshot({ path: 'http://whatsmyuseragent.org/'`example-pix
  await browser.close();
})();
```

But what's even more interesting is that we can use Playwright to navigate bot single and multi page applications.

This example uses Firefox Nightly but it will work the same with Chromium and WebKit.

This example does the following:

1. Requires the browser that we want to use (in this case Firefox Nightly)
2. Launches the browser
    1. Sets `headless` to false, so you can see a browser window executing the commands
    2. Uses `slowMo` with a 2000 milliseconds value to slow down the actions so we can see what happens
3. Goes to the specified page
4. Clicks on the first element matching the parameter, in this case the link inside the header of an article
5. Captures a screenshot of the page we visited
6. Closes the browser

```
(async () => {
  const { firefox } = require('playwright');

  const browser = await firefox.launch({
    headless: false,
    slowMo: 2000,
  });

  const page = await browser.newPage();
  await page.goto('https://publishing-project.rivendellweb.net');
  await p'https://publishing-project.rivendellweb.net'e.screenshot({ path
  await browser.close();
})();
```

We can even have read Chromium's accessibility tree snapshot and combine it with the elements we've covered before to get a picture of what Chromium is doing from an accessibility standpoint.

For example, to get a dump of the accessibility snapshot we can use the following code. Note that I've chosen not to run the headless browser because I want to see what the browser is doing.

```
(async () => {
  const { chromium } = require('playwright');
```

```
  const browser = await chromium.launch({
    headless: false,
  });

  const page = await browser.newPage();
  await page.goto('https://publishing-project.rivendellweb.net/');
  const snapshot = await page.accessibility.snapshot();
  console.log(snapshot);
  await browser.close;
})();
```

We've just scratched the surface of what Playwright can do. The specifics will depend on what you're trying to do.