Reliable Java end-to-end testing for modern web apps

Last updated: 3 October 2021
Contents
About Playwright ................................................................................................. 4
Why Playwright? ................................................................................................. 4
  Multiple language support ................................................................................. 4
  Support for all browser families ...................................................................... 4
  Fast and reliable execution .............................................................................. 5
  Powerful automation capabilities .................................................................. 5
Recommended installs for Playwright Java .......................................................... 6
  Install Node.js and the Node Package Manager (npm) .................................... 6
    After the Node installation ............................................................................ 6
  Install Java ....................................................................................................... 6
  Install Maven .................................................................................................. 6
  Install an Integrated Development Environment (IDE) ................................ 6
Playwright with Apache Maven ............................................................................ 7
Core concepts of Playwright ................................................................................ 7
  Browser ........................................................................................................... 7
  Browser contexts ............................................................................................. 7
  Pages and frames ............................................................................................ 7
  Selectors .......................................................................................................... 7
  Locators ........................................................................................................... 8
Playwright Inspector ............................................................................................ 9
  Open the Playwright Inspector ...................................................................... 9
Command Line Tools ............................................................................................ 10
  Record scripts automatically ........................................................................ 10
    Preserve and restore authenticated state .................................................... 10
  Install browsers .............................................................................................. 10
Open a page with browsers and emulation options .............................................. 11
  Open Chromium: .......................................................................................... 11
  Open Web Kit: ................................................................................................ 11
  Open emulate an Apple iPhone 11. .............................................................. 11
  Open emulate colour scheme and viewport (screen) size .............................. 11
  Open emulate geolocation, language and timezone ..................................... 11
Inspect Selectors ................................................................................................ 11
Take Screenshot .................................................................................................. 12
Generate PDF ...................................................................................................... 12
Auto Waiting ................................................................................................................. 13
Inputs ................................................................................................................................. 14
  Text input (“fill”) .............................................................................................................. 14
  Checkboxes and radio buttons (“check” and “uncheck”) ................................................... 14
  Select options (“selectOption”) ...................................................................................... 14
  Mouse click (“click” and “dblclick”) .............................................................................. 14
  Type characters (“type”) ............................................................................................... 14
  Keys and shortcuts (“press”) .......................................................................................... 15
Assertions............................................................................................................................. 15
Reuse authentication states .................................................................................................. 16
  Save the authentication state .......................................................................................... 16
  Restore the authentication state ..................................................................................... 16
  Save the authentication state after a recording .............................................................. 16
  Restore the authentication state for a new recording ..................................................... 16
Emulation ............................................................................................................................. 16
Error Handling ..................................................................................................................... 17
Browser Flags and Configuration Settings (Options) ............................................................. 17
  Chromium Flags ............................................................................................................ 17
  Firefox Configuration Settings ...................................................................................... 17

Disclaimer

Some content of this document has been sourced from the official Playwright web site:
https://playwright.dev/.

The Playwright team really did an excellent job in documenting the Playwright features and
providing useful examples.
About Playwright

Playwright is a free open source library for browser automation developed and maintained by Microsoft. Like Selenium, Playwright offers multiple language bindings, including Java.

The JavaScript/TypeScript version of Playwright has its own test runner, called Playwright Test. This document describes the Java library use of Playwright, which can be used with any Java (unit) test framework, such as JUnit or TestNG.

Playwright is led by the same team that originally built Puppeteer at Google. It therefore builds on the strength of Puppeteer, particularly on browser context flexibility and network manipulation capabilities, including handling and modifying of requests (stubbing and mocking).

Unlike Cypress, Playwright does not run inside a browser and does therefore not suffer from the same browser security limitations as Cypress, such as cross-site scripting restrictions (for example when using external authentication and iframes).

Why Playwright?

Playwright enables fast, reliable and capable testing and automation across all modern browsers.

Multiple language support

This guide covers Playwright Java, but comparable to Selenium, the Playwright API is available in multiple languages:

- JavaScript and TypeScript
- Python
- Java
- .NET

Support for all browser families

- Playwright runs on Chromium, Firefox and WebKit. Playwright has full API coverage for all modern browsers, including Google Chrome and Microsoft Edge (with Chromium), Apple Safari (with WebKit), and Mozilla Firefox. Headless execution is supported for all browsers on all platforms.
- Cross-platform WebKit testing. With Playwright, you can test how your app behaves in Apple Safari with WebKit builds for Windows, Linux and macOS. In other words: You can reproduce Apple Safari on non-Apple operating systems! Test can run locally and in Continuous Integration.
- Test for mobile. Use device emulation to test your responsive web apps in mobile web browsers.
- Headless and headed. Playwright supports headless (without browser UI) and headed (with browser UI) modes for all browsers and all platforms. Headed is great for debugging, and headless is faster and suited for Continuous Integration and cloud executions.
Fast and reliable execution

- **Auto-wait API's.** Playwright interactions auto-wait for elements to be ready. This improves reliability and simplifies test authoring.
- **Timeout-free automation.** Playwright receives browser signals, such as network requests, page navigations, and page load events to eliminate the need for sleep timeouts that cause flakiness. This is a major advantage over Selenium.
- **Fast isolation with browser contexts.** Reuse a single browser instance for multiple isolated execution environments with browser contexts.
- **Resilient element selectors.** Playwright can rely on user-facing strings, like text content and accessibility labels to select elements. These strings are more resilient than selectors that are tightly-coupled to the DOM structure.
- **React and Vue selectors.** Playwright has customised element selectors for the React and Vue web frameworks.
- **All selector engines except for XPath pierce shadow DOM by default.** This is a major breakthrough advantage over Selenium and crucially important for modern web apps.

Powerful automation capabilities

- **Multiple domains, pages and frames.** Playwright is an out-of-process automation driver that is not limited by the scope of in-page JavaScript execution and can automate scenarios with multiple pages.
- **Powerful network control.** Playwright introduces context-wide network interception to stub and mock network requests.
- **Modern web features.** Playwright supports web components through shadow-piercing selectors, geolocation, permissions, web workers, and other modern web API's.
- **Capabilities to cover all scenarios.** Support for file downloads and uploads, out-of-process iframes, native input events, and even dark mode.
Recommended installs for Playwright Java

Install Node.js and the Node Package Manager (npm)
Before you can install and use Playwright, you need to install Node.

Please follow an installation guide for your operating system and version.

You can check that Node has been successfully installed with this command:

```
node -v
```

When you installed Node, you also automatically installed the “npm” Command Line Interface (CLI), which is the package manager for Node. You can check it with:

```
npm -v
```

After the Node installation
Visit https://www.npmjs.com/package/playwright and follow the usage instructions:

```
npm i -D playwright
```

This installs Playwright and browser binaries for Chromium, Firefox and WebKit.

Install Java
Java JDK version 8 (Standard Edition) or higher (Java 17 is recommended) can be installed either from Oracle https://www.oracle.com/java/technologies/javase-downloads.html, or alternatively a version of the Java Open JDK, for example from: https://openjdk.java.net/

Install Maven
The latest stable version of Apache Maven version 3 or higher can be installed from https://maven.apache.org/download.cgi.

Install an Integrated Development Environment (IDE)
An Integrated Development Environment (IDE), such as IntelliJ IDEA Community Edition or Microsoft Visual Studio Code is recommended for development: https://www.jetbrains.com/idea/download/ or https://code.visualstudio.com/download.
Playwright with Apache Maven

Playwright is distributed as a set of Apache Maven modules. The easiest way to use it is to add one dependency to your project’s “pom.xml”:

```xml
<dependency>
  <groupId>com.microsoft.playwright</groupId>
  <artifactId>playwright</artifactId>
  <version>1.15.0</version>
</dependency>
```

Core concepts of Playwright

Playwright cascades down over multiple levels. A browser is the highest level, followed by browser context(s), followed by pages and frames, which contain elements that are accessed by selectors. Selectors can also be repackaged as locators to make the program code easier to read and understand.

Browser

A browser refers to an instance of Chromium, Firefox, or WebKit. Playwright scripts generally start with launching a browser instance and end with closing the browser. Browser instances can be launched in headless (without a GUI) or headed mode.

Browser contexts

A browser context is an isolated incognito-alike session within a browser instance. Browser contexts are fast and cheap to create. Each test scenario should run in its own new browser context, so that the browser state is isolated between the tests.

Browser contexts can also be used to emulate multi-page scenarios involving mobile devices, permissions, locale and color scheme.

Pages and frames

A browser context can have multiple pages. A page refers to a single tab or a popup window within a browser context. A page should be used to navigate to URLs and interact with the page content.

A page can have one or more frame objects attached to it. Each page has a main frame and page-level interactions (like click) are assumed to operate in the main frame.

A page can have additional frames attached with the iframe HTML tag. These frames can be accessed for interactions inside the frame.

Selectors

Playwright can search for elements using CSS selectors, XPath selectors, HTML attributes like “id”, “data-test-id”, and even text content.
You can explicitly specify the selector engine you are using, or let Playwright detect it.

All selector engines except for XPath pierce shadow DOM by default. First they search for the elements in the light DOM in the iteration order, and then they search recursively inside open shadow roots in the iteration order.

In particular, in the CSS engine, any descendant combinator or child combinator pierces an arbitrary number of open shadow roots, including the implicit descendant combinator at the start of the selector. It does not search inside closed shadow roots or iframes.

If you’d like to opt-out of this behaviour, you can use the “:light” CSS extension or “text:light” selector engine. They do not pierce shadow roots. You don’t typically need to do this, though.

Some selector examples (there are more available, including special selectors for the React and Vue web frameworks):

<table>
<thead>
<tr>
<th>Selector</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find node by text substring</td>
<td>page.click(&quot;text=Hello w&quot;);</td>
</tr>
<tr>
<td>Playwright supports a shorthand for selecting elements using these attributes:</td>
<td>page.click(&quot;data-test-id=foo&quot;);</td>
</tr>
<tr>
<td>id, data-testid, data-test-id, data-test</td>
<td>page.click(&quot;div&quot;);</td>
</tr>
<tr>
<td>CSS and XPath selector engines are automatically detected (&quot;xpath=&quot; does not have to be prefixed, if the XPath starts with &quot;/&quot;)</td>
<td>page.click(&quot;div&quot;);</td>
</tr>
<tr>
<td>Select by attribute, with CSS selector</td>
<td>page.click(&quot;div&quot;);</td>
</tr>
<tr>
<td>Selecting based on layout, with CSS selector</td>
<td>page.click(&quot;input:right-of(:text(&quot;Username&quot;))&quot;);</td>
</tr>
<tr>
<td>Pick n-th match. Note that unlike CSS’s nth-match, the provided index is 0-based.</td>
<td>page.click(&quot;input:right-of(:text(&quot;Username&quot;))&quot;);</td>
</tr>
<tr>
<td>Only search light DOM, outside WebComponent shadow DOM</td>
<td>page.click(&quot;css:light=div&quot;);</td>
</tr>
</tbody>
</table>

**Locators**

Locators represent a view to the element(s) on the page. They capture the logic sufficient to retrieve elements at any given moment. Locators can be created with the “Page.locator(selector)” method, for example:

```javascript
Locator  locator = page.locator("text=Submit");
locator.click();
```

The difference between a “Locator” and an “ElementHandle” is that the latter points to a particular element, while “Locator” captures the logic of how to retrieve that element.

Locators are particularly useful when using the Page Object Model design pattern.

Locators can make scripts much easier to read and understand.
Playwright Inspector

There are many tools (such as the built-in browser developer tools) and ways to debug Playwright scripts, but the Playwright Inspector is the default recommendation for script debugging.

The Playwright Inspector shows a toolbar to step through the Playwright script for debugging:

Open the Playwright Inspector

The Playwright Inspector can easily be used in the Command-Line Interface (that is also used to record scripts).

The Playwright Inspector can also be called from within a script:

page.pause();
Command Line Tools

The Playwright Command-Line Interface (CLI) offers many great features. These are the most important ones:

Record scripts automatically
Playwright can record scripts in Java. The CLI can be used to record user interactions and generate Java code. This example records on the Google web site:

```mvn exec:java -e -Dexec.mainClass=com.microsoft.playwright.CLI -Dexec.args="codegen google.com "```

Preserve and restore authenticated state
Run “codegen” with “--save-storage” to save cookies and localStorage at the end after recording. This is useful to separately record an authentication step and reuse it later. In this example, “auth.json” will contain the stored state:

```mvn exec:java -e -Dexec.mainClass=com.microsoft.playwright.CLI -Dexec.args="codegen --save-storage=auth.json"
```

Run with “--load-storage” to consume a previously stored state. This way, all cookies and localStorage will be restored, bringing most web apps to the authenticated state.

Example of opening with restored state:

```mvn exec:java -e -Dexec.mainClass=com.microsoft.playwright.CLI -Dexec.args="open --load-storage=auth.json my.web.app"
```

Example of recording with restored state.

```mvn exec:java -e -Dexec.mainClass=com.microsoft.playwright.CLI -Dexec.args="codegen --load-storage=auth.json my.web.app"
```

Install browsers
Install all supported browsers:

```mvn exec:java -e -Dexec.mainClass=com.microsoft.playwright.CLI -Dexec.args="install"```
Open a page with browsers and emulation options

**Open Chromium:**

```mvn
eexec:java -e -Dexec.mainClass=com.microsoft.playwright.CLI -Dexec.args="open google.com"
```

**Open Web Kit:**

```mvn
eexec:java -e -Dexec.mainClass=com.microsoft.playwright.CLI -Dexec.args="wk google.com"
```

**Open emulate an Apple iPhone 11.**

```mvn
eexec:java -e -Dexec.mainClass=com.microsoft.playwright.CLI -Dexec.args='open --device="iPhone 11" google.com'
```

**Open emulate colour scheme and viewport (screen) size**

```mvn
eexec:java -e -Dexec.mainClass=com.microsoft.playwright.CLI -Dexec.args="open --viewport-size=800,600 --color-scheme=dark twitter.com"
```

**Open emulate geolocation, language and timezone**

```mvn
eexec:java -e -Dexec.mainClass=com.microsoft.playwright.CLI -Dexec.args='open --timezone="Europe/Rome" --geolocation="41.890221,12.492348" --lang="it-IT" maps.google.com'
```

**Inspect Selectors**

During “open” or “codegen”, you can use the following API inside the developer tools console of any browser.

Example: In Chromium, you open the Developer Tools with the “<F12>” key, and then click on “Console” to type:
Query Playwright selector, using the actual Playwright query engine:

```java
playwright.$(selector)#
```

Same as “playwright.$“, but returns all matching elements:

```java
playwright.$$$(selector)#
```

Reveal element in the elements panel (if the browser development tools of the respective browser supports it):

```java
playwright.inspect(selector)#
```

Generates selector for the given element:

```java
playwright.selector(element)#
```

**Take Screenshot**

Wait 3 seconds before capturing a screenshot after page loads (“load” event fires):

```
mvn exec:java -e -Dexec.mainClass=com.microsoft.playwright.CLI -Dexec.args='screenshot --device="iPhone 11" --color-scheme=dark --wait-for-timeout=3000 twitter.com twitter-iphone.png'
```

Capture a full page screenshot:

```
mvn exec:java -e -Dexec.mainClass=com.microsoft.playwright.CLI -Dexec.args='screenshot --full-page google.com google-full.png'
```

**Generate PDF**

PDF generation only works in headless Chromium.

```
```
Auto Waiting

Playwright automatically performs these actionability checks:

<table>
<thead>
<tr>
<th>Action</th>
<th>Attached</th>
<th>Visible</th>
<th>Stable</th>
<th>Receives Events</th>
<th>Enabled</th>
<th>Editable</th>
</tr>
</thead>
<tbody>
<tr>
<td>check</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>click</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>dblclick</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>tap</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>uncheck</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>hover</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>scrollIntoViewIfNeeded</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>screenshot</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>fill</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>selectText</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>dispatchEvent</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>focus</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>getAttribute</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>innerText</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>innerHTML</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>press</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>setInputFiles</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>selectOption</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>textField</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>type</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

It is possible to wait for a specific selector:

```javascript
page.waitForSelector("text=My Text");
```

Although Auto Waiting will usually do the work, it is also possible to check programmatically:

```javascript
ElementHandle.isChecked()  
ElementHandle.isDisabled()  
ElementHandle.isEditable()  
ElementHandle.isEnabled()  
ElementHandle.isHidden()  
ElementHandle.isVisible()  
Page.ischecked(selector[, options])  
Page.isDisabled(selector[, options])  
Page.isEditable(selector[, options])
```
Page.isEnabled(selector[, options])
Page.isHidden(selector[, options])
Page.isVisible(selector[, options])

Inputs

Text input (“fill”)
This is the easiest way to fill in form fields. It focuses the element and triggers an input event with the entered text. It works for “<input>”, “<textarea>”, “[contenteditable]” and “<label>” associated with an input or textarea. For example:

```
page.fill("text=First Name", "Peter");
```

Checkboxes and radio buttons (“check” and “uncheck”)
This is the easiest way to check and uncheck a checkbox or a radio button. This method can be used with “input[type=checkbox]”, “input[type=radio]”, “[role=checkbox]” or “label” associated with checkbox or radio button. Examples:

```
page.check("#agree");
page.uncheck("#agree");
```

Select options (“selectOption”)
Selects one or multiple options in the “<select>” element. You can specify the option value, label or elementHandle to select. Multiple options can be selected. Examples:

```
page.selectOption("select#colors", "blue");
page.selectOption("select#colors", new String[] {"red", "green", "blue"});
```

Mouse click (“click” and “dblclick”)
```
page.click("button#submit");
page.dblclick("#item");
```

Type characters (“type”)
This method will emit all the necessary keyboard events, with all the keydown, keyup, keypress events in place. You can even specify the optional delay between the key presses to simulate real user behaviour. For example:

```
page.type("#area", "Hello World!");
```
Keys and shortcuts ("press")
This method focuses the selected element and produces a single keystroke. It accepts the logical key names that are emitted in the "keyboardEvent.key" property of the keyboard events like:

Backquote, Minus, Equal, Backslash, Backspace, Tab, Delete, Escape, ArrowDown, End, Enter, Home, Insert, PageDown, PageUp, ArrowRight, ArrowUp, F1 - F12, Digit0 - Digit9, KeyA - KeyZ, etc.

Example:

```javascript
page.press("#name", "Shift+A");
```

Assertions
Playwright provides convenience API's for common tasks, like reading the text content of an element. These API's can be used in your test assertions.

<table>
<thead>
<tr>
<th>Assertion</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text content</td>
<td>String <code>content</code> = page.textContent(&quot;nav:first-child&quot;);</td>
</tr>
<tr>
<td></td>
<td>assertEquals(&quot;home&quot;, content);</td>
</tr>
<tr>
<td>Inner text</td>
<td>String <code>text</code> = page.innerText(&quot;.selected&quot;);</td>
</tr>
<tr>
<td></td>
<td>assertEquals(&quot;value&quot;, text);</td>
</tr>
<tr>
<td>Attribute value</td>
<td>String <code>alt</code> = page.getAttribute(&quot;input&quot;, &quot;alt&quot;);</td>
</tr>
<tr>
<td></td>
<td>assertEquals(&quot;Text&quot;, alt);</td>
</tr>
<tr>
<td>Checkbox state</td>
<td>boolean <code>checked</code> = page.isChecked(&quot;input&quot;);</td>
</tr>
<tr>
<td></td>
<td>assertTrue(checked);</td>
</tr>
<tr>
<td>JS expression</td>
<td>Object <code>content</code> = page.evalOnSelector(&quot;nav:first-child&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;e =&gt; e.textContent&quot;);</td>
</tr>
<tr>
<td></td>
<td>assertEquals(&quot;home&quot;, content);</td>
</tr>
<tr>
<td>Inner HTML</td>
<td>String <code>html</code> = page.innerHTML(&quot;div.result&quot;);</td>
</tr>
<tr>
<td></td>
<td>assertEquals(&quot;&lt;p&gt;Result&lt;/p&gt;&quot;, html);</td>
</tr>
<tr>
<td>Visibility</td>
<td>boolean <code>visible</code> = page.isVisible(&quot;input&quot;);</td>
</tr>
<tr>
<td></td>
<td>assertTrue(visible);</td>
</tr>
<tr>
<td>Enabled state</td>
<td>boolean <code>enabled</code> = page.isEnabled(&quot;input&quot;);</td>
</tr>
<tr>
<td></td>
<td>assertTrue(enabled);</td>
</tr>
</tbody>
</table>
Reuse authentication states

Save the authentication state
The authentication state (cookies and local storage) can be saved to a file and later restored.

This is a very powerful feature that can save a lot of time and effort to reuse authentication states in apps that require (login) authentication.

```java
context.storageState(new BrowserContext.StorageStateOptions().setPath(Paths.get("auth.json")));
```

Restore the authentication state

```java
BrowserContext context = browser.newContext(new Browser.NewContextOptions().setStorageStatePath(Paths.get("auth.json")));
```

Save the authentication state after a recording
Run “codegen” from command line with “--save-storage” to save cookies and localStorage at the end:

```
mvn exec:java -e -Dexec.mainClass=com.microsoft.playwright.CLI -Dexec.args="codegen www.myapp.com --save-storage=auth.json"
```

Restore the authentication state for a new recording
Start a new “codegen” recording session from command line with a previously saved authentication state:

```
mvn exec:java -e -Dexec.mainClass=com.microsoft.playwright.CLI -Dexec.args="open www.myapp.com --load-storage=auth.json "
```

Emulation
Playwright allows overriding various parameters of the device where the browser is running:

- User agent
- Viewport (viewport size, device scale factor, touch support)
- Locale & timezone
- Permissions (such as notifications and geolocation access)
- Geolocation
- Color scheme and media

Most of these parameters are configured during the browser context construction, but some of them (such as viewport size) can be changed for individual pages.
Error Handling
Java example code with timeout:

```java
try {
    // Check if an element is clickable with a timeout of 10 seconds
    page.click("text=My Text", new Page.ClickOptions().setTimeout(10000));
} catch (TimeoutError e) {
    // Error handling goes here
}
```

Browser Flags and Configuration Settings (Options)
In Playwright, browser settings can be set with this command:

```javascript
BrowserType.launch([options])
```

Chromium Flags
Run Chromium with flags:


List of Chromium command-line switches:

http://peter.sh/experiments/chromium-command-line-switches/

List of default flags:


Use this command in chrome-based browsers to see the full list of what is enabled or disabled:

chrome://flags

Firefox Configuration Settings
Use this command in the Firefox browser to see configuration settings:

```bash
about:config
```

A list of all Firefox “about” pages:

```bash
about:about
```