[Browser and web automation (robocorp.com)](https://robocorp.com/docs/development-guide/browser)

**Browser and web automation**

You are already using one of the most powerful automation tools: **your browser**! As more and more tools and companies become web-based, learning how to automate the browser is one of the most useful skills for any Software Robot Developer.

**Is browser automation what you need?**

Before delving into browser automation, consider the task you are automating. Does the web application you are working with allow API access? APIs do not change as often as the graphical user interfaces. Automating using APIs means your automation scripts don't break as easily. Check the [HTTP](https://robocorp.com/docs/development-guide/http) section for information on working with HTTP APIs.

GET https://api.spacexdata.com/v4/launches

GET https://api.spacexdata.com/v4/rockets/{{rocket\_id}}

Sometimes there is no API provided, or the API is missing some functionality to complete the process. In these cases, you need to interact with the application's user interface; filling and submitting forms, pressing buttons, clicking on elements, scraping content, and many other interactions.

**How do you "automate" a browser?**

At a high level, browser automation is made possible by a "driver", a piece of software that can manage and control the browser just like a human would do, but executing instructions written in code. Historically, obtaining and setting up drivers for browser automation has required many steps. The RPA Framework set of libraries for Python and Robot Framework automatically takes care of this setup for you so that you can get started immediately.

**Which browser should you use in your automation?**

You can choose the browser you prefer. Our suggestion is to use [Google Chrome](https://www.google.com/chrome/) unless the application you are trying to automate works only with a specific browser. Different browsers might behave in different ways. A robot could work with Chrome, but not Safari or Internet Explorer.

**Which automation library should you use?**

Robocorp provides two main options for browser automation.

The [RPA.Browser.Selenium](https://robocorp.com/docs/libraries/rpa-framework/rpa-browser-selenium) library uses [Selenium](https://www.selenium.dev/) under the hood, currently the most established tool for browser automation. Technically, it is based on the [SeleniumLibrary](https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html" \t "_blank) to which it adds many convenient features to make your life easier as a developer.

The [RPA.Browser.Playwright](https://robocorp.com/docs/libraries/rpa-framework/rpa-browser-playwright) library ([Robot Framework Browser](https://robotframework-browser.org/)), based on the newer [Playwright](https://playwright.dev/) open-source project backed by Microsoft, provides an exciting alternative approach that promises to modernize the whole browser automation scene. You can learn more about how to work with this library and its pros and cons in [our dedicated Playwright section](https://robocorp.com/docs/development-guide/browser/playwright). Give a try!

**Opening the browser**

The first step is to open a browser. When using the [RPA.Browser.Selenium](https://robocorp.com/docs/libraries/rpa-framework/rpa-browser-selenium) library, the easiest way is to use the [Open Available Browser](https://robocorp.com/docs/libraries/rpa-framework/rpa-browser-selenium/keywords#open-available-browser) keyword, which will set up everything for you automatically. Internally, it will detect which browsers are installed on your machine and start the first browser it finds (it prefers Chrome!).

When using the [RPA.Browser.Playwright](https://robocorp.com/docs/libraries/rpa-framework/rpa-browser-playwright) library, use the [New Page](https://robocorp.com/docs/libraries/rpa-framework/rpa-browser-playwright/keywords#new-page) or the [Open Browser](https://robocorp.com/docs/libraries/rpa-framework/rpa-browser-playwright/keywords#open-browser) keywords.

**Other ways to use the browser**

When you open a browser using the Open Available Browser keyword, you get a blank state: the browser is not logged into any specific account and will use default settings. This setup is sufficient for many use cases, but there are cases in which you might want to use alternative approaches:

* [Attaching to a running browser](https://robocorp.com/docs/development-guide/browser/how-to-attach-to-running-chrome-browser) (Chrome only). You might have a browser profile where you have already performed a complicated login step or have plugins or configurations you want to use in your automation.
* You can open the current user's default browser using the [Open User Browser](https://robocorp.com/docs/libraries/rpa-framework/rpa-browser-selenium/keywords#open-user-browser) keyword.
* You can get full control of all configuration options by using the [Open Browser](https://robocorp.com/docs/libraries/rpa-framework/rpa-browser-selenium/keywords#open-browser) keyword. Then you are responsible for setting up the driver for your desired browser. Check the keyword documentation for more info.

“open available browser” or “open chrome browser”  🡪 RPA.Browser library

“open browser” 🡪 RPA.Browser.Selenium library

**Headless browser?**

While developing your web-based automation, it is useful to see the browser GUI at work. Once you have verified the process works, you might want to tell the browser to run in "headless" mode, which means no window or interaction will be shown to the user. You can always reverse this change to debug the process later. The headless mode is used when running the robot in a cloud container.

**Locators: a fundamental concept**

Humans can easily see and interact with elements using a mouse, keyboard, touch controls, or other input devices. Your software robot will need you to point it to the elements you want to interact with using **locators**. Using locators, you can tell the robot-browser which form inputs to fill, buttons to press, elements to scroll into view, etc. We have instructions covering [using locators in web applications](https://robocorp.com/docs/development-guide/browser/how-to-find-user-interface-elements-using-locators-in-web-applications).

**Logging into web applications**

Very often, logging into a web application is the first step you need to do.

For applications that require a simple user name and password, you will simply instruct the browser to interact with a login form using locators and the relevant keywords.

In more complex cases, you might need to [setup two-step authentication](https://robocorp.com/docs/development-guide/browser/how-to-handle-google-2-step-verification), or attach to a browser where you have manually logged into the application first.

**Downloading files via the browser**

If a file is publicly available on the internet and you know the URL, the easiest way to download it in the context of your software robot is probably to use the RPA Framework [HTTP library](https://robocorp.com/docs/development-guide/http/downloading-files-from-an-http-server).

Suppose the file is only available after you log into the application or resulting from an export operation that happens after a button is clicked, for example. In that case, you can download it using the browser. In this case, you probably want to be able to [change the browser default download directory](https://robocorp.com/docs/development-guide/browser/how-to-change-browser-download-directory).

**Learn more about browser and web automation**

Now you should have a clearer idea of what browser automation means. However, there is much more to learn! One way is to follow our [Beginners' course](https://robocorp.com/docs/courses/beginners-course), which will guide you through solving a fun use case with browser automation. Completing the course will grant you the Robocorp Level I certificate! Also, check out the robot examples in this section and on our [Portal](https://robocorp.com/portal/).

**All the articles and examples in this section**

[Web scraper robot](https://robocorp.com/docs/development-guide/browser/web-scraper-robot)

[A web scraper software robot.](https://robocorp.com/docs/development-guide/browser/web-scraper-robot)

[RPA form challenge robot](https://robocorp.com/docs/development-guide/browser/rpa-form-challenge)

[Creating a software robot to solve the form challenge at rpachallenge.com](https://robocorp.com/docs/development-guide/browser/rpa-form-challenge)

[Web store order robot](https://robocorp.com/docs/development-guide/browser/web-store-order-robot)

[A robot for processing Excel-driven web store orders using Robot Framework and RPA Framework.](https://robocorp.com/docs/development-guide/browser/web-store-order-robot)

[How to use an already running (Chrome) browser for your web automation robots](https://robocorp.com/docs/development-guide/browser/how-to-attach-to-running-chrome-browser)

[Learn how to take over control of your already running browser with your rpaframework automation scripts using the `Attach Chrome Browser` keyword.](https://robocorp.com/docs/development-guide/browser/how-to-attach-to-running-chrome-browser)

[How to change the browser default download directory](https://robocorp.com/docs/development-guide/browser/how-to-change-browser-download-directory)

[Changing the browser default download location with the RPA.Browser.Playwright and RPA.Browser.Selenium libraries.](https://robocorp.com/docs/development-guide/browser/how-to-change-browser-download-directory)

[How to find user interface elements using locators in web applications](https://robocorp.com/docs/development-guide/browser/how-to-find-user-interface-elements-using-locators-in-web-applications)

[The art of helping your robot to find forms to fill in and buttons to click on in web applications.](https://robocorp.com/docs/development-guide/browser/how-to-find-user-interface-elements-using-locators-in-web-applications)

[How to handle Google's 2-step verification](https://robocorp.com/docs/development-guide/browser/how-to-handle-google-2-step-verification)

[Use the Authenticator to handle Google's 2-step verification in your robot code](https://robocorp.com/docs/development-guide/browser/how-to-handle-google-2-step-verification)

[How to handle website notices if they appear](https://robocorp.com/docs/development-guide/browser/how-to-handle-website-notices-if-they-appear)

[Click on an element if it appears. Ignore if not!](https://robocorp.com/docs/development-guide/browser/how-to-handle-website-notices-if-they-appear)

[How to switch between browser windows](https://robocorp.com/docs/development-guide/browser/how-to-switch-between-browser-windows)

[How to span your robot over multiple browser windows.](https://robocorp.com/docs/development-guide/browser/how-to-switch-between-browser-windows)

[How to work with iframes](https://robocorp.com/docs/development-guide/browser/how-to-work-with-iframes)

[Working with forms inside and outside of an iframe.](https://robocorp.com/docs/development-guide/browser/how-to-work-with-iframes)

[Targeting dropdown elements in web applications](https://robocorp.com/docs/development-guide/browser/working-with-custom-dropdowns-in-web-applications)

[How to automate dropdown elements in webpages.](https://robocorp.com/docs/development-guide/browser/working-with-custom-dropdowns-in-web-applications)

[A new option for web automation: Using the Robot Framework Browser library, based on Playwright](https://robocorp.com/docs/development-guide/browser/playwright)

[Getting started with Playwright-based browser automation using the Robot Framework Browser library in Robocorp.](https://robocorp.com/docs/development-guide/browser/playwright)

[How to debug Playwright-based (Robot Framework Browser) robots in Visual Studio Code](https://robocorp.com/docs/development-guide/browser/how-to-debug-playwright-robots-in-visual-studio-code)

[How to develop and debug Playwright-based robots in Visual Studio Code using Robocorp Code extension.](https://robocorp.com/docs/development-guide/browser/how-to-debug-playwright-robots-in-visual-studio-code)

**Are you stuck? Get help in the forums or on our Slack!**

If you have questions or need help with your automation project, [register a free account](https://id.robocorp.com/signup) and get help in our [Community forums](https://forum.robocorp.com/) or Slack!

Learn more about the libraries mentioned on this page:

* [RPA.Browser.Selenium](https://robocorp.com/docs/libraries/rpa-framework/rpa-browser-selenium)
* [RPA.HTTP](https://robocorp.com/docs/libraries/rpa-framework/rpa-http)

**Selector**

selector Selector of the text field. See the [Finding elements](https://robocorp.com/docs/libraries/rpa-framework/rpa-browser-playwright#Finding%20elements) section for details about the selectors.

## Finding elements

All keywords in the library that need to interact with an element on a web page take an argument typically named selector that specifies how to find the element. Keywords can find elements with strict mode. If strict mode is true and locator finds multiple elements from the page, keyword will fail. If keyword finds one element, keyword does not fail because of strict mode. If strict mode is false, keyword does not fail if selector points many elements. Strict mode is enabled by default, but can be changed in library [importing](https://robocorp.com/docs/libraries/rpa-framework/rpa-browser-playwright#Importing) or [Set Strict Mode](https://robocorp.com/docs/libraries/rpa-framework/rpa-browser-playwright/keywords#set-strict-mode) keyword. Keyword documentation states if keyword uses strict mode. If keyword does not state that is used strict mode, then strict mode is not applied for the keyword. For more details, see Playwright [strict documentation](https://playwright.dev/docs/api/class-page#page-query-selector).

|  |  |  |
| --- | --- | --- |
| **Strategy** | **Match based on** | **Example** |
| Css | CSS selector. | css=.class > \#login\_btn |
| xpath | XPath expression. | xpath=//input[@id="login\_btn"] |
| Text | Browser text engine. | text=Login |
| Id | Element ID Attribute. | id=login\_btn |

사례 #1

1. Browser에서 해당 위치에서 오른쪽 버튼 🡪 “검사” 기능

텍스트이(가) 표시된 사진

자동 생성된 설명

1. **class**=”**best-matches**” 내 <**a** href=..>인 element를 확인
2. Console 🡪 O 에서 document.querySelector(‘.best-matches a’) 를 입력하여 검색 확인
3. Selenium에서 적당한 library(click element, click button 등)를 선택하여 테스트

사례 #2

아래 title의 text인 “Products”를 읽으려 할 때

텍스트이(가) 표시된 사진

자동 생성된 설명

1. (F12 Elements 탭 open )
2. Web 화면의 대상 (여기서는 “PRODUCTS”)에서 오른쪽 버튼 🡪 검사 선택 🡪 F12 Elements 탭 해당부분 검색됨.
3. 검색 부분에서 오른쪽 버튼 🡪 Copy 🡪 Copy Selector 선택하여 locator 획득

#header\_container > div.header\_secondary\_container > span

1. F12 Console 탭에서 document.querySelector('#header\_container > div.header\_secondary\_container > span').textContent 🡪 “Products” 가져올 수 있음

**Locator syntax**

<https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html>

SeleniumLibrary supports finding elements based on different strategies such as the element id, XPath expressions, or CSS selectors. The strategy can either be explicitly specified with a prefix or the strategy can be implicit.

**Default locator strategy**

By default, locators are considered to use the keyword specific default locator strategy. All keywords support finding elements based on id and name attributes, but some keywords support additional attributes or other values that make sense in their context. For example, [*Click Link*](https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Click%20Link) supports the href attribute and the link text and addition to the normal id and name.

Examples:

|  |  |  |
| --- | --- | --- |
| [*Click Element*](https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Click%20Element) | example | # Match based on **id** or **name**. |
| [*Click Link*](https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Click%20Link) | example | # Match also based on **link text** and **href**. |
| [*Click Button*](https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Click%20Button) | example | # Match based on **id**, **name** or **value**. |

If a locator accidentally starts with a prefix recognized as [*explicit locator strategy*](https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Explicit%20locator%20strategy) or [*implicit XPath strategy*](https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Implicit%20XPath%20strategy), it is possible to use the explicit default prefix to enable the default strategy.

Examples:

|  |  |  |
| --- | --- | --- |
| [*Click Element*](https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Click%20Element) | name:foo | # Find element with **name** foo. |
| [*Click Element*](https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Click%20Element) | default:name:foo | # Use default strategy with **value** name:foo. |
| [*Click Element*](https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Click%20Element) | //foo | # Find element using **XPath** //foo. |
| [*Click Element*](https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Click%20Element) | default: //foo | # Use default strategy with **value** //foo. |

(아래와 같은 추가 내용 더 있음)

**Explicit locator strategy**

**Implicit XPath strategy**

**Chaining locators**

**Using WebElements**

**Custom locators**

<https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html> 확실히 파악할 필요 있음.

**Xpath**

<https://heodolf.tistory.com/7>

처음 python으로 크롤러를 만들때 requests와 bs4를 사용하였는데, bs4에는 XPATH에 대한 개념이 존재하지않았다. 그래서 자연스럽게 CSS Selector에 익숙해져 있었는데, Selenium과 Scrapy를 접하면서 XPATH를 사용하는 예제들이 많이 보였고 자연스럽게 XPATH에 대해서 알아보고 사용하게 되었다. 이 포스트는 사용하면서 느낀 두 선택자에 대한 정리이다.

  우선 CSS Selector는 태그의 패턴을 이용해 탐색하는 선택자이며, 웹 개발자에게 가장 익숙한 선택자일 것이다. stylesheets나 javascript에서 document.querySelector() 또는 jquery로 많이 접해봤을 것이다. 그만큼 우리에게 익숙하고 널리 사용되고 있는 선택자이다.

  필자도 javascript로 웹 개발을 배우면서 많이 사용했었는데 그 때마다 가장 아쉽게 느껴졌던 부분이 '원하는 범위의 값을 가진 태그'를 선택하는 선택자가 없다는 것이었다. 예를 들어, '**value가 50이상**인 input태그를 선택'한다고 했을 때 방법이 없다. 그냥 a태그 전체를 탐색해서 50이상인 태그만 스타일을 적용해주는 스크립트를 작성해야만 했다. 또, 선택한 태그들의 값을 바로 뽑아서 사용하고 싶어도 map으로 한번 가공해줘야했다.

  그런데, 그것이 가능해졌다. 바로 XPATH 때문이다. XPATH는 XML 경로 언어라고 하며 Node의 경로를 query로 탐색하는 선택자이다. 위에서 CSS Selector가 하지못했던 일을 XPATH는 할 수 있다. "**//input[value>=50]**". 매우 간단한 방법으로 처리한다. 또한, 선택된 태그의 값만 가지고 오려면 "**//input[value>=50]/@value**" 이렇게 사용하면 된다. 이 외에도 **4번째에 있는 자식 태그**를 탐색하기위해 "div:nth-child(4)"와 같이 복잡한 문법이 아니라, "**//div/\*[4]**" 간결한 문법을 제공한다.

**Open Browser** vs. **Open Available Browser**

https://rpaframework.org/libraries/browser\_selenium/

**Open Browser**

* Opens a new browser instance to the optional url.
* The browser argument specifies which browser to use. The supported browsers are listed in the table below. The browser names are case-insensitive and some browsers have multiple supported names.

**Open Available Browser**

* Attempts to open a browser on the user's device from a set of supported browsers. Automatically downloads a corresponding webdriver if none is already installed.
* Optionally can be given a url as the first argument, to open the browser directly to the given page.

**Hint**

* Selenium 외 Playwright에도 유용한 keyword가 많이 있음
* Keyword 가 여러 다른 library에 속한 경우 도 있음

예: Type Text 등

* New Page는 browser 움직임 없이도 동작함
* Auto closing browser

By default, browser instances created during task execution are closed at the end of the task. This can be prevented with the auto\_close argument when [importing](https://robocorp.com/docs/libraries/rpa-framework/rpa-browser-selenium#Importing) the library.

Value needs to be set to False or anything considered false (see Boolean arguments).

Set the auto\_close option to keep the browser open. This is currently an undocumented feature.

Here’s an example in Robot Framework - the same option is available in Python:

\*\*\* Settings \*\*\*

Library RPA.Browser.Selenium auto\_close=${FALSE}

Tried with rpa framework 12.5.1 and worked as excepted in VS.

참조: <https://devahea.github.io/2019/04/13/Headless-Browser%EB%9E%80/>

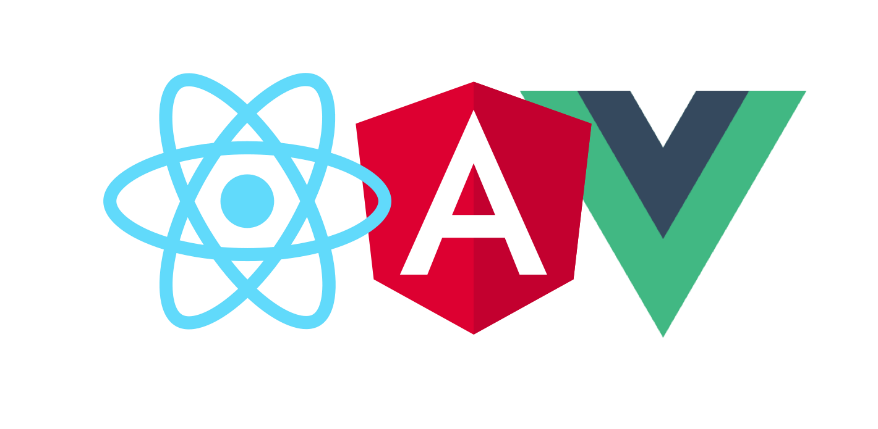
**Headless Browser란?**

A headless browser is a web browser without a graphical user interface.

말 그대로 GUI환경이 아닌 브라우저를 뜻 합니다.

CLI(Command Line Interface)에서 동작하는걸 뜻하며, 프로그래밍하기에 용이 해집니다.

2000년대 까지만 하더라도 Javascript는 Web Page에서 동적으로 보이기 위해 사용 되어졌지만 Web의 성장함으로써 Frontend진영에서 Javascript Framework가 생기고, 이를 활용하여 개발한 SPA(Single Page Application)이 활발하게 되었습니다.



**react-angular-vue**

SPA이전엔 HTTP Client(Jsoup, OKHttp)를 활용하여 Web Scraping(또는 Crawling)가능 해졌지만 Javascript를 실행해야지만 Scraping이 가능 해졌습니다.

Headless Browser의 등장으로 복잡한 인증 및 Javascript 실행으로 인해 실제 사람이 접속 하듯이 Scraping 할 수 있게 되었습니다.

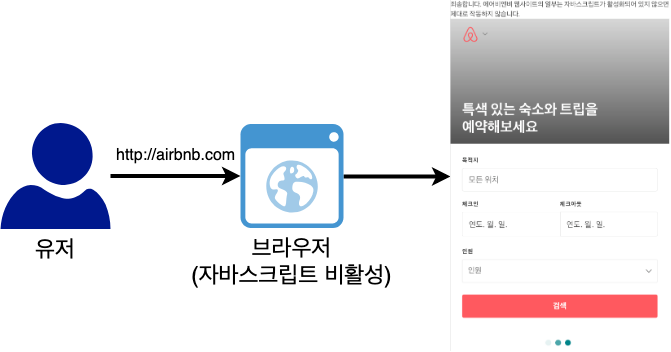
**Http Client와 Headless Browser**

HTTP Client는 URL입력과 동시에 Html Content를 바로 불러와 속도가 빠릅니다. 그에 비해 Headless Browser는 백그라운드상에서 실제 Browser가 돌기에 상대적으로 속도가 느립니다. 그래도 SPA접근과 복잡한 인증을 쉽게 해결 가능 한데요.  
Javascript 구동 여부는 Airbnb 페이지에서 확인 해보겠습니다.

CLI환경에서 에서 둘의 차이점을 확인 할 수 있습니다.

# curl HttpClient로 에어비앤비 Scraping

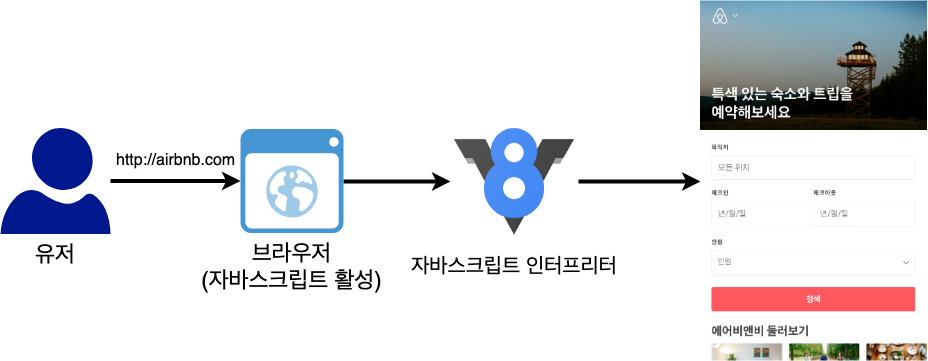
curl https://www.airbnb.co.kr > curl-Airbnb.html



**javascript-NoJavascript**

# Chrome 헤드리스 모드로 에어비앤비 Scraping

chrome --headless --dump-dom --virtual-time-budget=3000 https://www.airbnb.co.kr > headless-Airbnb.html



**javascript-Javascript**

ls -lh

-rw-r--r-- 1 a1004024 staff 241K Apr 13 21:50 httpclient-Airbnb.html

-rw-r--r-- 1 a1004024 staff 327K Apr 13 21:50 headless-Airbnb.html

두개 파일의 용량을 보면 HTTP Client 결과는 241K이며 Headless Browser의 결과는 327K 였습니다.

두개의 파일을 비교 해도 가져올수 있는 데이터를 확인 할 수있었습니다.

| **Client** | **속도** | **인증처리** | **Jascript실행여부** | **종류** |
| --- | --- | --- | --- | --- |
| Http Client | 빠름 | 어려움 | X | curl, wget, 그외 라이브러리들 |
| Headless Browser | 느림 | 쉬움 | O | Puppeteer, Selenium, PhantomJS 등 |

Http Client를 사용가능한 곳은 Http Client가 더 나은 선택 일 수 있습니다.

**Headless Browser를 통해 인증과 Scraping(네이버 카페)**

개인적으로 좋아하는 Node.js 기반 Headless Browser인 [Puppeteer](https://pptr.dev/)를 활용한 예제를 구성 하였습니다.

실제 네이버 카페를 개설 한 후 비공개 카페의 글을 Scraping 하는 예제 입니다.

이 에제는 총 5단계의 거쳐 스크래핑을 할 예정 입니다.

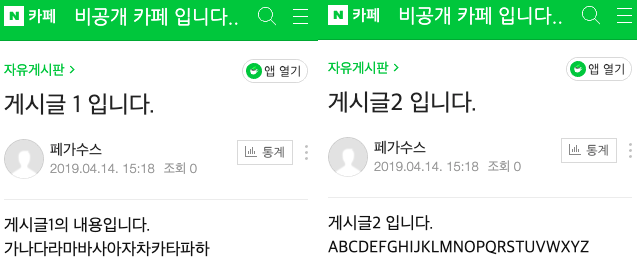
1. Headless Browser 실행
2. 네이버 로그인
3. 네이버 카페 게시판 접근
4. 게시판의 게시글들의 URL 추출
5. 추출된 게시글의 내용 추출

샘플에 사용된 네이버 카페 게시판



**스크린샷 2019-04-14 오후 4 42 23**

Scraping 해올 게시글



**noname**

**1. Headless Browser 실행**

npm에서 puppeteer 패키지를 다운 받습니다.

npm i puppeteer

다운 받은 패키지를 Node.js 애플리케이션 개발에 불러옵니다.

// src/naver-cafe.js 파일에 작성

const puppeteer = require('puppeteer');

(async () => {

const browser = await puppeteer.launch({headless: false})

const page = await browser.newPage()

// 2, 3, 4, 5 내용 채울 예정

await browser.close()

})();

**2. 네이버 로그인**

jQuery를 사용할때 쓰던 CSS Selector를 통해 Element에 접근을 하여 원하는 작업을 진행 합니다.

// 네이버 로그인 페이지 접근

await page.goto('https://nid.naver.com/nidlogin.login?svctype=262144&url=http://m.naver.com/aside/')

// 네이버 로그인

await page.type('#id', '네이버 아이디', {delay: 100})

await page.type('#pw', '네이버 비밀번호', {delay: 100})

await page.click('[type="submit"]')

await page.waitForNavigation()

**3. 네이버 카페 게시판 접근**

Scraping을 하고 싶은 네이버 게시판의 URL로 이동 합니다.

await page.goto('https://m.cafe.naver.com/ArticleList.nhn?search.clubid=29734529&search.menuid=1&search.boardtype=L')

**4. 게시판의 게시글들의 URL 추출**

Chrome devtools를 이용하여 Scraping 대상을 분석 하고 코드로 작성 하여야 합니다.  
Puppeteer에서 제공하는 Page클래스의 evaluate함수를 이용하면 Chrome devtools console에서 작업하는거와 동일하게 작업이 가능 해집니다.

// 자유게시판 글 Scraping

const articles = await page.evaluate(args => {

const articles = []

document.querySelectorAll('#articleListArea > ul li')

.forEach((v, i) => {

const article = {}

article.title = v.querySelector('.tit').innerText

article.nick = v.querySelector('.nick').innerText

article.href = v.querySelector('a.\_articleListItem').href

articles.push(article)

})

return articles

})

// 자유게시판 글들 확인

console.log(articles)

위의 작업을 실행한 결과 입니다.

[ { title: '게시글2 입니다.',

nick: '페가수스',

href:

'https://m.cafe.naver.com/ArticleRead.nhn?clubid=29734529&articleid=3&page=1&boardtype=L&menuid=1' },

{ title: '게시글 1 입니다.',

nick: '페가수스',

href:

'https://m.cafe.naver.com/ArticleRead.nhn?clubid=29734529&articleid=2&page=1&boardtype=L&menuid=1' } ]

**5. 추출된 게시글의 내용 추출**

추출된 각 게시글의 URL에 방문하여 게시글의 내용을 Scraping 합니다.

const articlePage = await browser.newPage()

for (let article of articles) {

await articlePage.goto(article.href)

const content = await articlePage.$eval('#postContent', element => element.innerText)

article.content = content;

}

// 자유게시판 글들을 내용까지 채움

console.log(articles)

브라우저의 새탭을 열어 아까 가져온 게시글 URL에 차례대로 방문하여 내용을 Scraping 해옵니다.

[ { title: '게시글2 입니다.',

nick: '페가수스',

href:

'https://m.cafe.naver.com/ArticleRead.nhn?clubid=29734529&articleid=3&page=1&boardtype=L&menuid=1',

content: '게시글2 입니다.\n\nABCDEFGHIJKLMNOPQRSTUVWXYZ' },

{ title: '게시글 1 입니다.',

nick: '페가수스',

href:

'https://m.cafe.naver.com/ArticleRead.nhn?clubid=29734529&articleid=2&page=1&boardtype=L&menuid=1',

content: '게시글1의 내용입니다. \n\n가나다라마바사아자차카타파하' } ]

**마무리**

Headless Browser란 Http Client에 비해 Javascript를 구동하여 추가 데이터를 가져 온다고 이해 하시면 좋습니다.

Node.js기반 Headless Browser인 Puppeteer를 사용한 네이버 카페 Scrping하였고, 이를 활용하면 네이버 로그인 인증도 쉽게 할 수 있고, 게시글을 사람이 직접 가져오는 방법처럼 예제를 구성 하였습니다.

소개한 예제는 Github에 공유 하겠습니다.  
다음 순서엔 다른 Headless Browser와의 비교를 소개 하겠습니다.  
<https://github.com/kji6252/study-puppeteer>  
[https://pptr.dev](https://pptr.dev/)

# <https://www.selenium.dev/documentation/webdriver/>

# WebDriver

WebDriver drives a browser natively, learn more about it.

WebDriver drives a browser natively, as a user would, either locally or on a remote machine using the Selenium server, marks a leap forward in terms of browser automation.

Selenium WebDriver refers to both the language bindings and the implementations of the individual browser controlling code. This is commonly referred to as just WebDriver.

Selenium WebDriver is a [W3C Recommendation](https://www.w3.org/TR/webdriver1/)

* WebDriver is designed as a simple and more concise programming interface.
* WebDriver is a compact object-oriented API.
* It drives the browser effectively.

## Selenium

[Selenium](https://www.guru99.com/introduction-to-selenium.html) is a free (open source) automated testing suite for web applications across different browsers and platforms. Primarily it is used for automating web applications for testing purposes, but is certainly not limited to just that. Selenium has the support of all of the major browser vendors who have taken (or are taking) steps to make Selenium a native part of their browser. It is also the core technology in countless other browser automation tools, APIs and frameworks.

Selenium is not just a single tool but a set of different software tools each with a different approach to support the test automation of an organization. From a broader perspective previously it had four components as follows:

* Selenium Integrated Development Environment (IDE)
* Selenium Remote Control (RC)
* WebDriver
* Selenium Grid

An year ago, Selenium RC and WebDriver are merged into a single framework to form **Selenium 2.x**. Perhaps, **Selenium 1** refers to **Selenium RC**. The current released version is **Selenium 3.x**.

## WebDriver

Selenium-RC worked the same way for each supported browser. It injected **javascript** functions into the browser when the browser was loaded and then used its javascript to drive the AUT within the browser. [Selenium WebDriver](https://docs.seleniumhq.org/docs/03_webdriver.jsp) fits in the same role as Selenium-RC did and has incorporated the original 1.x bindings and included the WebDriver API. It refers to both the language bindings and the implementations of the individual browser controlling code. This is commonly referred to as just **WebDriver**. In short, [WebDriver](https://developer.mozilla.org/en-US/docs/Web/WebDriver) is the remote control interface that enables introspection and control of user agents. WebDriver provides a platform and language-neutral wire protocol as a way for out-of-process programs to remotely instruct the behavior of web browsers.

## [Highlights of WebDriver](https://www.seleniumhq.org/projects/webdriver/)

* WebDriver is designed in a simpler and more concise programming interface along with addressing some limitations in the Selenium-RC API.
* WebDriver is a compact Object Oriented API when compared to Selenium1.0
* It drives the browser much more effectively and overcomes the limitations of Selenium 1.x which affected our functional test coverage, like the file upload or download, pop-ups and dialogs barrier
* WebDriver overcomes the limitation of Selenium RC's [Single Host origin policy](https://en.wikipedia.org/wiki/Same-origin_policy).

## Current Implementation

WebDriver is the name of the key interface against which tests should be written in Java/C#/Ruby/Python/NodeJS, the implementing classes which you can use are listed as below:

* [ChromeDriver](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/chrome/ChromeDriver.html)
* [EventFiringWebDriver](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/support/events/EventFiringWebDriver.html)
* [FirefoxDriver](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/firefox/FirefoxDriver.html)
* [HtmlUnitDriver](http://seleniumhq.github.io/htmlunit-driver/org/openqa/selenium/htmlunit/HtmlUnitDriver.html)
* [InternetExplorerDriver](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/ie/InternetExplorerDriver.html)
* [PhantomJSDriver](https://github.com/detro/ghostdriver/blob/master/binding/java/src/main/java/org/openqa/selenium/phantomjs/PhantomJSDriver.java)
* [RemoteWebDriver](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/remote/RemoteWebDriver.html)
* [SafariDriver](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/safari/SafariDriver.html)