

[End-to-end Testing](#)
[How-tos & Guides](#)

7 min read

Creating automated tests for Safari and WebKit

Learn how to create automated tests that can run against Safari browsers using open-source tools like Selenium and Playwright.



Alvin Charity

Published September 16, 2022

Table of contents

Difficulties in testing against Safari

Testing Safari with Selenium and SafariDriver

Testing Safari using Playwright

Testing Safari using Cypress

Continuous Integration support for testing Safari

Running Safari tests within your own infrastructure

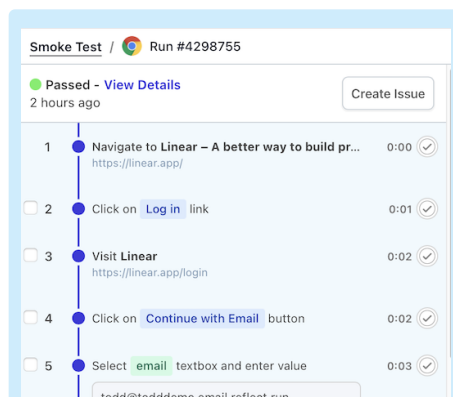
Safari is the default browser for all Apple devices, and is the second most popular web browser worldwide. Despite its popularity, building test automation to run on Safari or its underlying WebKit engine remains difficult, especially compared to the relative ease of other popular browsers like Chrome, Firefox, and Edge.

In this article, you will learn about some challenges faced by developers who want to automate testing in Safari. Additionally, you'll read about what currently works to automate testing in Safari, and what is still lacking. Finally, you will explore some options for continuous integration and virtualization of the MacOS platform to help automate your Safari testing.

Difficulties in testing against Safari

One of the major challenges you may face as a developer looking to test in Safari is that the browser and engine are only available on Apple devices. Although there are workarounds for using non-Apple hardware, including projects for running MacOS software inside Docker container, all of these workarounds are in violation of Apple's End User License Agreement (EULA) and Terms of Service (TOS). Apple prohibits use of any MacOS software on any device that is not "Apple branded." This means that any tool running Safari (for testing purposes or otherwise) on non-Apple hardware is violating this license.

Safari itself is only made to run on MacOS hardware, so there additionally is no official Safari browser on Linux. Because of this, any Safari container you may see in the Docker registry will be a WebKit-based browser for Linux, and not the official MacOS Safari. While these WebKit browsers may have some degree of feature parity with Safari, it is crucial to test accurately against the official Safari app on MacOS hardware.



Smoke Test / Run #4298755

Passed - View Details
2 hours ago

Create Issue

- 1 Navigate to Linear – A better way to build pr...
https://linear.app/ 0:00 ✓
- 2 Click on Log in link 0:01 ✓
- 3 Visit Linear
https://linear.app/login 0:02 ✓
- 4 Click on Continue with Email button 0:02 ✓
- 5 Select email textbox and enter value 0:03 ✓

Tired of flaky end-to-end tests?

Create fast and reliable tests for anything that runs in a browser.

[Learn more](#)

requirements.

Testing Safari with Selenium and SafariDriver

Selenium is a popular library for browser-based testing and automation. Selenium allows testing against Safari via **SafariDriver**, which implements the cross-browser WebDriver API spec that Selenium uses under the hood. WebDriver / Selenium support for Safari has been **available since 2016**, and is the only approach for cross-browser testing that uses a standards-based API.

The example below will use Python, so make sure you have a recent version of Python3 and `pip` installed before you proceed.

To get started with Selenium and Safari driver use `pip` to install selenium and webdriver-manager:

```
pip install selenium webdriver-manager
```

Once these libraries have finished downloading, you can create a new folder for your script. In the folder you just created, create a file called `main.py`. Paste the following code into `main.py` and save the file:

```
from selenium.webdriver.common.by import By
from selenium import webdriver
import time

SafariDriver = webdriver.Safari()
SafariDriver.get("https://webkit.org/status")
search_box = SafariDriver.find_element(By.ID, "search")
search_box.send_keys("CSS")
value = search_box.get_attribute("value")
search_box.submit()
time.sleep(1)
print(len(value))
SafariDriver.quit()
```

To run this script in your terminal, enter the following command

```
python3 main.py
```

The example above will open the page located at `https://webkit.org/status`, search for the text "CSS", and print the total number of results found for the query.

Testing Safari using Playwright

maintains a forked version of each browser which contains the necessary hooks that enable Playwright to drive the browser UI.

The code below shows an example of Playwright using Python. Playwright recommends that you use the **pytest package** when installing Playwright with pip.

To do this, enter the following command in your terminal:

```
pip install pytest-playwright
```

The command above will install an executable called `playwright`. You will need this executable to install browsers and engines to be used with your Playwright scripts. Besides Safari, Playwright includes support for several other browsers including Google Chrome, Firefox, and Edge. The full list of supported browser is available [here](#). To install these browsers, enter the following command in your terminal

```
playwright install
```

After your Playwright installation is complete, create a new folder and copy the code below into a new file called `playwright.py`:

```
from playwright.sync_api import sync_playwright

def webkit(playwright):
    browser = playwright.webkit.launch()
    page = browser.new_page()
    page.goto("https://webkit.org/status")
    page.locator('#search').fill('css')
    value = page.locator('#feature-count').all_text_contents()
    print(f"total results = {' '.join(value)}")
    page.close()

with sync_playwright() as playwright:
    webkit(playwright)
```

To run this script on your machine, enter the following command in your terminal

```
python3 playwright.py
```

As with the previous example, this script will print the total number of search results to your terminal console.

Testing Safari using Cypress

WebKit browsers, plans have been underway since 2020 to add WebKit support to Cypress and experimental support for Safari testing has been released as part of Cypress's 10.8 release.

In order to enable testing against Safari Webkit, you'll need to set the following flag in your Cypress configuration:

```
experimentalWebKitSupport: true
```

To run tests against Safari, you'll need to pass the `--browser webkit` flag when running your Cypress test via the command line:

```
cypress run --browser webkit
```

Interestingly, the Cypress team didn't implement WebKit support directly. Instead, they've used Playwright's own fork of WebKit; specifically the `playwright-webkit` plugin.

More information on the current status of Safari support within Cypress can be found in [this GitHub issue](#).

Continuous Integration support for testing Safari

[GitHub](#) and [CircleCI](#) are two popular Continuous Integration (CI) vendors that support running tests on MacOS within their infrastructure, and thus support running tests on Safari.

GitHub Actions

When creating a GitHub Action, you must specify what "runner" will be used to execute the action. GitHub maintains [several runners](#) that run on top of MacOS. To use MacOS as a part of your Github CI, add `runs-on: macos-latest` to your yaml file. For example:

```
jobs:
  build:
    runs-on: macos-12
```

More information can be found on the [GitHub Blog](#).

CircleCI

In addition to GitHub Actions, CircleCI also allows you to run tests against Safari on the MacOS platform. On the surface, GitHub Actions and CircleCI may appear similar, however CI is CircleCI's main product and so they have a wealth of additional options available for you. You can read more about using CircleCI for Safari tests at the [CircleCI docs](#).

There are a few great options available that provide hosting for virtualized MacOS servers that allow you to run Safari tests in infrastructure that you control, but without requiring you maintain your own fleet of Apple hardware that's on-premise or co-located.

AWS EC2 MacOS instances

Amazon offers [MacOS EC2 instances](#). This pay-as-you-go service allows you to spin up a MacOS instance on demand for whatever purposes you need. AWS offers both Intel and ARM-based instances and every instance that is created has built-in access to all of the AWS related tools you need as part of your workflow. There are a number of additional options available for using AWS to test Safari, head to the [Amazon EC2 Mac Instances page](#) to find out more.

Anka by Veertu

[Veertu](#) offers MacOS virtualization in a container-like environment that allows you to run tests against MacOS and Safari. [Anka](#) is their virtualization layer built directly on top of the MacOS's hypervisor that supports stopping, starting, and provisioning Mac virtual machines as needed. Like AWS's Mac EC2 instances, Anka supports both Intel or ARM-based machines.

MacStadium

[MacStadium](#) describes itself as a "Mac focused cloud" and offers tools and services that fit this description. In addition to offering MacOS virtualization for testing and CI, MacStadium provides remote desktops and bare metal access to a MacOS machine for fine-grained, low level access to Apple's hardware. True to the "cloud" title, MacStadium also has services covering storage, networking, and monitoring.

Try [Reflect](#): A modern cross-browser testing platform

[Reflect](#) is a no-code testing platform that lets you build and run tests across all popular browsers, including Safari, without any installation required. Creating a test in Reflect is easy: the tool records your actions as you use your site and automatically translates those actions into a repeatable test that you can run across all modern browsers instantly.

Starting creating Safari tests in minutes. [Try Reflect for free](#).

Get started with Reflect today