Haketilo - Feature #66

Write tests

07/26/2021 04:13 PM - koszko

Status:	Closed	**Start date:**	11/27/2021
Priority:	Normal	**Due date:**	
Assignee:	koszko	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			

Description

It seems problematic to test software that is meant to run as a browser extension - and it indeed is, especially when it comes to testing stuff that uses WebExtension APIs, not to mention inter-context messaging. However, there are some ways of testing code in-browser and we can always come up with our own ones. It is also not impossible to mock an environment with sites to inject scripts to (by meddling with /etc/hosts or employing an HTTP proxy).

Creativity wanted

Tests system quirks (listed here to save time of those who ever try to hack around it):

- It seems OK to navigate a Selenium driver to a page in a Pytest fixture but **only** when the fixture is function-scoped.
- Browser takes a lot of time to start, so it only makes sense to reuse a browser instance in subsequent tests. Here we achieve this by Pytest fixture scoping.
- Pytest fixtures that are not function-scoped **have to** be defined in conftest.py to work properly.
- Surprisingly, it is possible install a WebExtension when running Firefox in safe mode, but the extension has to be installed as **temporary**.
- There seems to be no straightforward way to learn extension's internal UUID (needed to navigate to extension's bundled files/pages) using Selenium. For unit tests the solution is to make the test extension open its own page when started (or redirect a certain HTTP request to its own page).
- Even when Firefox is started from Selenium with an empty profile, it loads the extensions that are installed globally on the system. Those can interfere with the tests and should be disabled, for example by a properly-crafted extensions.json file. When running an a different system, it might be necessary to adjust the extensions.json. Because of that it is preferred to just test in Firefox' safe mode when possible.
- Selenium driver's execute_script method runs the script in some special, one-time context. If we want to run the script in page's context, we can inject it by adding a <script> tag to the page.
- If executed script returns a Promise, Selenium's execute_script method will wait for the promise to resolve and will return its result.
- When an error happens in a script injected via the <script> tag, it is a pain to debug it... Nevertheless, some useful info might then be obtained from the geckodriver.log file Selenium creates. console.log-based debugging is also possible if we disable headless mode in Firefox (which in our case is enabled by defining an environment variable in Makefile's test rule) and make the test code sleep in the relevant part of the failing test.
- Want to start a Selenium-driven IceCat instance together with a Python command-line where the driver can be manipulated? Run make test-environment.
- Want to only run a single test? Instead of make test run pytest -vv -k <test-function-name>. Optionally add MOZ_HEADLESS=<whatever> before the command to stop IceCat window from appearing. Rn running all tests on RockPro64 takes over 19s and running a single one takes over 10s.

The above were noted when testing with IceCat60. Might or might not be true for newer browsers...

Subtasks:

Feature # 97: Make tests system parametrizable through configure script	**Closed**

History

#1 - 07/26/2021 04:14 PM - koszko

- Description updated

#2 - 07/28/2021 07:27 AM - jahoti

Mocking sites is definitely critical, albeit probably better done with a hijacking proxy of some sort (my words, not a known technical term) than with the extremely limited options of /etc/hosts.

As for conducting the actual tests themselves, shouldn't it be enough to use a browser driver for all the UI interactions and then do the rest by adding

JS to "test builds" of the extension, which can communicate with the main test suite using AJAX? Perhaps- quite likely- I'm completely misunderstanding the issue.

#3 - 07/28/2021 09:24 AM - koszko

jahoti wrote:

> Mocking sites is definitely critical, albeit probably better done with a hijacking proxy of some sort (my words, not a known technical term) than with the extremely limited options of /etc/hosts.

If it can allow us to do tests without having to run a VM/chroot/container/UML/Linux namespace, then proxy is preferable.

> As for conducting the actual tests themselves, shouldn't it be enough to use a browser driver for all the UI interactions

I haven't used browser drivers before, but they indeed seem to be the way to go.

> and then do the rest by adding JS to "test builds" of the extension, which can communicate with the main test suite using AJAX?

Yep, that's what we should do.

> Perhaps- quite likely- I'm completely misunderstanding the issue.

You are not :)

#4 - 08/05/2021 11:47 AM - jahoti

- Assignee set to jahoti

This is now off to a (very slow) start.

It's currently in a separate folder to Hachette; should that continue, or would it be best to add the test suite to the extension repository.

#5 - 08/05/2021 12:30 PM - koszko

jahoti wrote:

> This is now off to a (very slow) start.
>
> It's currently in a separate folder to Hachette; should that continue, or would it be best to add the test suite to the extension repository.

I see no problem in having it in the repo. After all, it is somehow related to the code (i.e. changes in code force changes in tests).

Please for now only focus on things that are not going to change quickly. Btw, currently considered changes are:

1. Giving the same semantics to *** in both domain and path parts of a pattern
2. Removing page_info_server

#6 - 08/06/2021 02:42 AM - jahoti

> Please for now only focus on things that are not going to change quickly.

I'll make sure to once it gets to that stage; currently the entire extent of the test suite is a stripped down MITM proxy :).

#7 - 09/05/2021 04:29 AM - jahoti

- % Done changed from 0 to 10

The basic infrastructure to support creating a "virtual network" in now in the jahoti branch, and can be used on its own (with some extra web pages and bypassing an error Chromium throws).

As soon as I finally manage to get Selenium working, this can actually be put to use in the way it was intended.

#8 - 09/09/2021 01:52 PM - koszko

Have you considered using UML (no, not that diagraming language, I mean User Mode Linux) to run tests inside? I'm suggesting this just in case

#9 - 09/11/2021 04:44 AM - jahoti

> Have you considered using UML (no, not that diagraming language, I mean User Mode Linux) to run tests inside? I'm suggesting this just in case

I didn't actually- thanks for suggesting it! Once the basic test system is running and reporting results, that will be the next job before finally working on a thorough set of tests once (hopefully) the core features and interface are stable.

#10 - 11/24/2021 09:38 AM - koszko

- Status changed from New to In Progress

- Assignee changed from jahoti to koszko

jahoti wrote:

> Have you considered using UML (no, not that diagraming language, I mean User Mode Linux) to run tests inside? I'm suggesting this just in case

> I didn't actually- thanks for suggesting it! Once the basic test system is running and reporting results, that will be the next job before finally working on a thorough set of tests once (hopefully) the core features and interface are stable.

I've been thinking more about this recently. We want some way to isolate the test environment from other netowrking on the machine to avoid interference. We could use either a full virtual machine, UML or some container (e.g. a Docker one). I earlier suggested UML because full virtual machine seems like on overkill and I personally like UML more than Docker (trauma after having to resist Docker Hub's nonfree js back when I was a student). Whether we choose Docker or UML, we still need operating system's entire userspace to be installed in it. While I don't consider it unacceptably unfeasible, it is a bit of an inconvenience.

We can instead use Linux network namespaces. This way we can still use host's programs. I know, it adds the requirement that host OS needs to be using Linux (which would also be the case if we used UML), yet I think this is acceptable (it is the build system that has to be portable). We don't want to require root access to run tests, so we need to use user namespaces. Those used to be disabled by default on Debian kernels which is yet another inconvenience that I think we should just swallow. However, here I use jxself's linux-libre .deb package and it seems to have user namespaces enabled by default :)

So, to spawn a shell in its own network namespace with uid=0 and gid=0, we do:

```
unshare -Urn
```

Now we have not only network isolation but also the ability to mess with the firewall.
Please look into unshare's manpage to find more details. Or don't. I will be working on this crazy namespace stuff now :)

#11 - 11/27/2021 01:51 PM - koszko

- % Done changed from 10 to 50

unshare doesn't seem to work when in chroot and I currently run tests agains a browser I have installed inside a chroot. I will try to modify the makefile rule to test if unshare can be used and only use it if it can. Otherwise, port 1337 must be free the our internal test proxy to listen on.

I added to the Python test code from jahoti branch. The koszko branch now has a make test target added which uses Selenium WebDriver to spawn a headless IceCat instance which then accesses an example page over HTTPS through our internet-mocking proxy server. There's also a make test-environment target that spawns a headed IceCat instance as well as a Python console where WebDriver attached to this IceCat instance can be accessed under the driver variable.

In both make test and make test-environment the IceCat instance is made to accessed a mocked internet through a proxy which serves content as defined in test/world_wide_library.py.

Current single test case only checks if a mocked page gets loaded properly. More tests that will check Haketilo's code will be added. This will probably involve doing #79.

In my case, IceCat was loading some globally-installed extensions that would interfere with the test, so I included a minimal profile that disables them. This is obviously just a workaround that will not work everywhere...
Nevertheless, I suggest we instead rely on Firefox' safe mode in unit tests and only use the minimal profile when testing Haketilo as a whole. Currently make test and make test-environment use safe mode exclusively.

For all this I've been using Parabola's IceCat 60 (the older browser we use for testing, the less likely we are to later learn that we're unknowingly relying on some feature that is unavailable somewhere else). Chromium-based tests would also be cool to have, but I am not considering them a priority

EDIT
You currently need the following to run tests:

- a supported browser (currently Parabola's IceCat60, would be cool if you could add support for other ones)
- python3
- pytest
- Python binding for Selenium (python-selenium in Parabola)
- geckodriver (geckodriver in Parabola)

#12 - 11/30/2021 04:07 AM - jahoti

It's working nicely! I've had a look at the unshare manpage as well, and while it deserves a more thorough read you've definitely found the right tool.

> unshare doesn't seem to work when in chroot and I currently run tests agains a browser I have installed inside a chroot. I will try to modify the makefile rule to test if unshare can be used and only use it if it can. Otherwise, port 1337 must be free the our internal test proxy to listen on.

Alternatively, the default of 1337 could be removed; there's a function free_port in selenium.webdriver.common.utils which returns a free port.

> I added to the Python test code from jahoti branch. The koszko branch now has a make test target added which uses Selenium WebDriver to spawn a headless IceCat instance which then accesses an example page over HTTPS through our internet-mocking proxy server. There's also a make test-environment target that spawns a headed IceCat instance as well as a Python console where WebDriver attached to this IceCat instance can be accessed under the driver variable.
>
> In both make test and make test-environment the IceCat instance is made to accessed a mocked internet through a proxy which serves content as defined in test/world_wide_library.py.
>
> Current single test case only checks if a mocked page gets loaded properly. More tests that will check Haketilo's code will be added. This will probably involve doing #79.

I see you've started on that!

> In my case, IceCat was loading some globally-installed extensions that would interfere with the test, so I included a minimal profile that disables them. This is obviously just a workaround that will not work everywhere...
> Nevertheless, I suggest we instead rely on Firefox' safe mode in unit tests and only use the minimal profile when testing Haketilo as a whole. Currently make test and make test-environment use safe mode exclusively.

Definitely- they're hardly unit tests otherwise :).

> For all this I've been using Parabola's IceCat 60 (the older browser we use for testing, the less likely we are to later learn that we're unknowingly relying on some feature that is unavailable somewhere else). Chromium-based tests would also be cool to have, but I am not considering them a priority

To save some debugging for whoever takes on Chromium, some notes:

- Extensions *cannot be used* in headless mode
- While it's probably been fixed, driver.quit() doesn't kill the actual browser with my selenium installation
- The driver needs to be the same version as the browser to work

> EDIT
> You currently need the following to run tests:
>
> - a supported browser (currently Parabola's IceCat60, would be cool if you could add support for other ones)

I'll try #97 and see what other customizations need to be made after that.

- python3
- pytest
- Python binding for Selenium (python-selenium in Parabola)
- geckodriver (geckodriver in Parabola)

#13 - 12/01/2021 02:10 PM - koszko

koszko branch now has a facility to load .js files together with their dependencies to a page in selenium-driven browser.

UPDATE
I improved the facility. It is now possible to use selenium to execute scripts in page's global scope. I had to do workarounds to enable reporting of errors from such scripts.

It now works like this: if a.js depends on b.js which depends on c.js, our new facility can load them into a test page this way:

1. exports_init.js is loaded, raw
2. c.js is loaded, wrapped in an anonymous function, with its exports available through window object
3. b.js is loaded in the same fashion
4. a.js is loaded **without** a wrapping anonymous function but with imports satisfied

Then, subsequent scripts can be loaded and they can access private global variables from a.js

See test/unit/test_patterns.py for an example.

Other quirks I fought:

- Custom-scoped pytest fixtures myst be either in the same file that uses them or in conftest.py. I have no idea why...
- When IceCat60 reports javascript syntax error on line 172 in index.html, it means the error is on line 172 of the script, not 172 of the html page.
- Selenium driver's execute_script method executes scripts in some kind of one-time scope. They have access to window (and hence variables declared by page's scripts using var) but no access to variables declared by const or let nor to variables declared in previous calls to execute_script. That's the reason for all the hassle :)
- "get_log is not implemented by Firefox driver."

Selenium tests seem to be resource-intensive. On my RockPro64 Merely starting IceCat60 takes ~10s and the few test cases we have now already take another 2s. I guess there's little we can do about it right now...

I am now working on querying of URL patterns. Breakages in code pushed to koszko might occur ;)

#14 - 12/03/2021 12:57 AM - jahoti

This seems to be coming together well!

I'm in the process of addressing #97, which (along with a few other build system parameters) should make it to the build-sys branch tomorrow at the very latest.

#15 - 12/13/2021 07:19 PM - koszko

- Description updated

koszko branch now has code to use test extensions in unit tests

Typical scenario:
You have some code that uses WebExtension APIs. You use the webextension Pytest fixture from conftest.py, parameterizing it through a pytest.mark decorator:

```
# You could as well pass an empty dict and use the defaults.
@pytest.mark.ext_data({
    'background_script': 'console.log("this will run in background page");',
    'content_script': 'console.log("this will run at document_start on every page");',
    'test_page': '<h1>this will be the extension-bundled page opened when the test function is called</h1>'
})
def test_something(driver, execute_in_page):
    # do something
    assert something
```

#16 - 02/19/2022 11:22 AM - koszko

- Status changed from In Progress to Closed

- % Done changed from 50 to 100

Quite a lot of test cases are now in master