

Examining Cross Browser Inconsistencies with Web Based Event Loggers

Rahul Kochar

Dr. David Maxwell (Supervisor)

Dr. Claudia Hauff (Supervisor)

Technische Universiteit Delft

Delft, Netherlands

ABSTRACT

Cross Browser Inconsistencies (XBI) were created when different browser vendors implemented their products without deciding upon common protocols for interoperability. It is hard to pinpoint these inconsistencies with precision because of a lack of a good tool. Here we show how to use a web-based event logger (LogUI) to find XBI at a granular level. LogUI attaches itself to the DOM of a webpage and makes a log file based on how the browser responds to user interactions. A test suite is made by simulating user interactions (Selenium WebDriver) to test different browsers and interaction events (individually and in sequences) to generate log files which are then analyzed to spot differences in the actions performed and entries logged. It is found that XBI are few and hard to find, the two XBI found are that browsers load differently and sometimes change the order of actions being performed.

KEYWORDS

Web based event logging, User simulation and testing, Selenium WebDriver, LogUI

1 INTRODUCTION

Different browsers use different engines which means pages can be rendered differently and browsers can respond to user interactions differently. Dowden and Dowden [1] has multiple examples of differences in CSS code across different browsers and [2] found that specific HTML5[3] and CSS3[4] features are at fault leading to differences in HTML/CSS grammar across browsers. Unfortunately, their work is limited to PhantomJS¹, Mozilla Servo² and Mozilla Firefox leaving a hole for other popular browsers like Chrome, Safari and Edge.

In the famous first browser war³ in 1995 where different browser vendors had their ideas and implemented their standards leading to widespread incompatibility issues. The nascent internet as a whole was built and developed to outdo fierce and ruthless competitors. This is chaotic for consumers and annoying for all stakeholders to say the least.

¹<https://phantomjs.org/>

²<https://servo.org/>

³https://en.wikipedia.org/wiki/Browser_wars

26 years later, in 2021, the narrative is very different on the surface because organizations like W3C[5] and Internet Engineering Task Force (IETF)[6] which attempt to bring all stakeholders on the same page to agree on standards and protocols which benefit all involved. IETF has undeniably seen success, however, as they write on their front page[?], "The Internet, a loosely-organized international collaboration of autonomous, interconnected networks, supports communication through voluntary adherence to open protocols and procedures defined by Internet Standards." with the keywords being "loosely-organized" and "voluntary adherence". Many stakeholders do not see eye to eye *yet* and differences continue to exist. Watanabe1 et al. [7] did a literature survey on XBI and it is evident that differences in browser standards was a large problem in the 1990s and it, unambiguously exists today.

2 BACKGROUND

It is from here that our suspicions about the performance of event driver logging by various browser engines arise. Table 1 shows different engines used by different browsers and are expected to result in observable and substantial differences. Apple attempted to solve this problem by requiring all browser vendors to use AppleWebKit⁴ thus enforcing a standard and forcefully making the engine they developed, mainstay. Fortunately, Microsoft did not follow this approach and the open source communities behind Linux were among the first to criticize Apple because this limits creativity and fair competition by outright banning all competitors. To accommodate MacOS users, browser vendors made their browsers such that they can be used with different engines example Firefox uses AppleWebKit[8] on MacOS but Gecko/SpiderMonkey on others.

This leads to the formulation of research questions and requirements (of test suite).

- **RQ1: How do web browsers handle different interaction events?**

Are different interaction events logged?

- **RQ2: How do different browsers report a sequence of events, and what differences exist, if any?**

How accurately do the log files represent the sequence of actions performed by WebDriver? This is broad and includes the sequence of events, time delays between events and number of occurrences of events.

This is a list of requirements/features that will help answer the research questions.

⁴<https://en.wikipedia.org/wiki/WebKit>

Browser	Rendering Engine	JavaScript Engine
Firefox	Gecko, Quantum	SpiderMonkey
Chrome	Blink, Webkit	V8
Edge	EdgeHTML	V8 (Chromium)
Safari	WebKit	Nitro, JavaScriptCore
Opera	Blink (Chromium), Presto	Chrome V8
Internet Explorer	Trident	Chakra, JScript

Table 1: Modern Browser Engines.

- **F1: All tests should be completely automated**
No human interaction should be required after starting.
- **F2: Tests should be deterministic (repeatable)**
Tests runs in similar conditions should yield similar results.
- **F3: Test multiple browsers**
At least 2, ideally more - Chrome, Firefox, Safari, Opera and Edge

2.1 Finding Browser Inconsistencies

Lots of literature has been written on the differences and adherence of agreements by browser vendors in the last decades. ([9] shows the logical construction of a compliance test to evaluate the extent of compliance of agreements, standards and protocols among browser vendors. Their paper is about proving the logical strength of an approach to quantify compliance so that web standards can be formalized. Features are not exhaustively tested on different browsers. Nyrhinen and Mikkonen [10], [11]) are two other papers that explore similar ideas but do not attempt to test a sufficient number of features such that compliance by different browser vendors can be quantified. Their work is theoretical with some experimental proof. An easier approach is to simply try out all possible interactions and count which ones are consistent, which ones are not. Checking all of them can be hard and therefore a list of most frequently used interactions could be used to estimate consistency of a feature and individual browser compliance by dividing the number of consistent/compliant features with inconsistent/non-compliant features. Such a compliance table as described already exists and is maintained by Mozilla. A paper that uses this data to quantify browser compliance could not be found. More importantly, this table checks compliance of features individually and not in sequences whereas users perform a sequence of actions when interacting with a web page. Mozilla has trusted browser vendors to find these inconsistencies which may pop up in a sequence of actions themselves.

Automated tools have been built to find inconsistencies in browser behaviour, called Cross-Browser Issues (XBI) using concepts like Machine Learning [12], Search-Based techniques [13] and Rule-Based ([14], [15]). These approaches have issues for example machine learning has a cost attached to accuracy, training time. These approaches require large amounts of data in the form of training set which is hard to procure. A test suite that tries out different interactions on different browsers one at a time (similar to unit testing) is easier to make, use and scale. It was earlier mentioned that [2] has identified specific CSS and HTML elements that are troublesome,

[16] solves this problem, very neatly by making an automated tool to fix these inconsistencies in HTML and CSS grammar.

3 LEVERAGING MODERN WEB TECHNOLOGY

3.1 The Web Page

Modern web page consists of two main things, browser elements (tabs, search bars, plugins, etc) and a DOM (Document Object Model)[17]. DOM is a tree-like structure that contains elements that make the web page interesting. Browser vendors initially did not agree upon protocols to work with DOM leading to incompatibilities and inconsistencies.

3.2 LogUI

An easy way to check if browsers treat elements of DOM equitably is to gather and evaluate user interactions. Unfortunately, gathering user interaction data is not a trivial task, fraught with difficulties because of dynamic behaviour(s) in web pages and other inherent complexities. To compound the problem, there is a lack of a good one size fits all tool because individual researchers make their own apparatus specific to their work. LogUI[18] is a simple to use yet powerful user interaction logging software made possible by leveraging modern web technology. A listener is attached to elements of the DOM and is triggered by an action performed by a user. The listener logs an entry into a log file when it is triggered thus giving an inside view of how elements of DOM are treated.

There are six types of tests for web applications (Functional Testing, Usability Testing, Interface Testing, Compatibility Testing, Performance Testing and Security Testing)[19], [20] has lists of papers related to the six types of tests. The most suitable test for this task is compatibility testing (of browser engines and GUI). Alégroth et al. [21] talks about a Visual Graphical Testing (VGT) test suite with a Continuous Integration and Continuous Deployment (CI/CD) in an industrial setting. They write that "test scripts operate on the same level of abstraction as the human user, they are intuitive and therefore easy to develop" which suggests that to test the listeners attached on DOM elements, a user should be simulated interacting with the said DOM elements. For example, clicking on URL and hovering on images. Their work is quite advanced and specific to their use case but it shows the general direction to work towards. To exploit browser inconsistencies further, [22] lists different test items that should be tested like image, forms, cookies, hyperlinks, databases and so on.

User interaction data is gathered by simply simulating users on a web page (section 4) and inconsistencies are detected by comparing log files of different browsers. To do this, listeners need to be evaluated. This approach is automated, easy to scale and maintain.

3.3 Testing Web Applications

Marchetta and Tonella [23] test a web application that uses AJAX[24] with many different techniques such as White Box, Black Box and State-Based Testing. LogUI is also asynchronous which means the sequence in which two events occur may not be preserved in the log files. This can change the outcome of the test and/or conclusions hence formed. Debroy et al. [25] examines testing a web application in an industrial setting with Selenium Webdriver⁵ and NUnit[26] using Test Driver Development⁶. They explain that Web Automation and Web Tests are different things, Web Tests code does not know anything about user simulation or browsers while Web Automation code does not have any knowledge of the frameworks and tools used to create the web application. It simply simulates the user as instructed.

4 FRAMEWORKS AND TOOLS

4.1 Browser Support

To select a tool to simulate a user, an important criteria is the number of browsers supported by that tool. Table 2 has various candidates on the first column and supported browsers along the row.

Puppeteer[27] is very popular among developers for browser automation but it primarily supports Chrome and Chromium browsers. Guzmán Castillo et al. [28] uses Puppeteer to perform similar tasks to those that we would like to perform and this gives confidence that Puppeteer is a capable candidate but for limited browsers. It was later learnt that NightwatchJS [29] solves the problems of Puppeteer.

CasperJS⁷ is another testing framework that works on PhantomJS (webkit) and SlimerJS (Gecko) headless browsers only. This is not ideal as we would like to test on as many browsers as possible. Other potential candidates include Puppeteer⁸, Selenium Grid⁹, Selenium RC¹⁰ and Selenium Webdriver[30].

Selenium IDE¹¹ performs record and replays. A user's actions are recorded and then replayed at will to perform a test. The process of recording is not desirable because manual user interactions are not always consistent, simulating mouse events with code gives greater confidence that the mouse event has been performed the same way every time. "Selenium Grid allows us to run tests in parallel on multiple machines, and to manage different browser versions and browser configurations centrally"¹². These features

⁵<https://www.selenium.dev/documentation/en/webdriver/>

⁶<https://developer.ibm.com/devpractices/software-development/articles/5-steps-of-test-driven-development/>

⁷<https://casperjs.org/>

⁸<https://developers.google.com/web/tools/puppeteer/>

⁹<https://www.selenium.dev/documentation/en/grid/>

¹⁰https://www.selenium.dev/documentation/en/legacy_docs/selenium_rc/

¹¹<https://www.selenium.dev/selenium-ide/>

¹²<https://www.selenium.dev/documentation/en/grid/>

are not required.

Webdriver meets the number of browsers supported requirement and it needs to be determined if it makes sense to choose for example Puppeteer with limited browsers if Puppeteer has more functionality than Webdriver offers. We will start with examining the usability, Webdriver works on all platforms, Linux, MacOS and Windows across Ruby, Java, Perl, Python, C# and JavaScript [31]. There is no major disadvantage of anyone language over the other [31] so Python is used. It is important to clarify that Webdriver was the same as Selenium RC. Selenium 1.0 + Selenium Webdriver = Selenium 2.0 and is today commonly called Webdriver. We will follow the same convention as it widely accepted and used. García et al. [32] examine the Selenium ecosystem and benchmark Selenium Webdriver on several parameters among other things and find that it is a capable framework (for our requirements).

4.2 Functionality

In terms of functionalities, we require mouse events[33] and keyboard¹³ events because these are the two interfaces used by a user to interact with a GUI (browser).

LogUI allows attaching listeners of all kinds that are supported by the browser. Selenium Webdriver does not support as many mouse events¹⁴ as those in Mozilla docs but they do support the important ones like click, double click, right-click and hover. A nuance is that scrolling on Selenium is inconvenient[34]. For keyboard, all keys and combinations are supported¹⁵. Since Webdriver[35] has almost all the required functionality on many different browsers, it does not make sense to compare with other candidates because it has already been shown that others do not support a sufficient number of browsers.

5 A TESTING SUITE FOR LOGUI

A test suite is a series of tests performed automated in an orderly fashion by a testing framework (Pytest). Each test has two parts, setup (perform test actions which is simulating a user) and then verifying these actions (analyzing log files). This addresses Feature F1 and F2 because the tests are automated and they are also guaranteed to be repeatable - assuming that external infrastructure (LogUI) does not change.

Every test in the current test suite is parameterized with the parameters being different browsers ensuring that the same test is run for all browsers (Feature F3). Pytest²³ allows creating fixtures like "beforeEach" and "before" methods to setup this infrastructure. The "beforeEach" ensures that every individual parameter of a test will get its own flight. Since LogUI is run locally on one port, tests can not be parallelized, at most one flight can be active at a time so that other flights don't eavesdrop and log data that they should not be.

To write a test, create a parameterized fixture with Pytest[36] and

¹³<https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent>

¹⁴https://www.selenium.dev/documentation/en/support_packages/mouse_and_keyboard_actions_in_detail

¹⁵<https://www.selenium.dev/documentation/en/webdriver/keyboard/>

²³<https://pytest.org/>

	Chrome	Firefox	Edge	Opera	Safari
Selenium IDE ¹⁶	✓	✓	✓	✓	✓
Puppeteer ¹⁷	✓	✓* ¹⁸	✗	✗	✗
Selenium Grid ¹⁹	✓	✓	✓	✓	✓
Selenium Web-driver ²⁰	✓	✓	✓	✓	✓
CasperJS ²¹	✗	✓**	✗	✗	✓**
NightwatchJS ²²	✓	✓	✓	✓	✓

Table 2: Browser support of use simulation tools

(*) - Experimental support

(**) - Headless only

pass the browsers to be evaluated in a list as a parameter. Open a browser and an HTML page using WebDriver and simulate a user. Once WebDriver has finished, a log file can be downloaded from LogUI.

Log files log actions performed when a listener attached to an event is triggered. The listeners are specified in the configuration object (section 6.2 instructions[37]) which instructs LogUI to attach listeners to various events. These events are tested to see if they are triggered (they should show up in the log file). Log files are then parsed and verified to answer research questions. The next paragraph explains how LogUI is used and integrated into the test suite.

LogUI is arranged in applications, flights and sessions[38]. An application houses multiple flights and a flight houses multiple sessions. In the test suite (collection of tests), each flight will have one session to keep things simple. Every time the test suite is run, LogUI makes an application for that run. Each test in every configuration needs to have its own flight so that log files of different tests are kept apart. Flights must have a unique name, whichever application they may be in because of how LogUI is built.

6 METHODOLOGY

This section will help the reader reproduce the work described in this report. Required frameworks, tools and their versions are in section 6.1, section 6.3 is about the test HTML page used and configuration object of LogUI. The following sections describe the individual tests conducted, an example search session and lastly, an evaluation of the test results.

6.1 Browsers and Other Version Details

LogUI client (version 0.5.3d²⁴) and server (version 0.5.3²⁵) are used. Firefox²⁶ (version 0.29.0) and AppleWebKit version (537.36) will be tested. Since all browsers on MacOS use AppleWebKit, all browsers on MacOS are tested indirectly, their counterparts in Windows and Linux however are not tested because of operating system, programming language and version inconsistencies. From here Safari will be used to refer to AppleWebKit (and other browsers

on MacOS) and Firefox will refer to Gecko (on Windows and Linux).

Other important version details are Python (version 3.8.5), Pytest (version 5.4.1)²⁷ and Selenium Webdriver (version 3.141.0)[30].

6.2 Configuration Object and Listeners

LogUI attaches listeners to DOM elements which are triggered when a user interacts in a specific way with that element. The interaction that triggers a listener and some other things are specified in the configuration object which looks like figure 1.

The configuration object[39] is a JSON[40] object divided into three parts that are highlighted in the picture The first "logUIConfiguration" setups and connect to LogUI with endpoints and authorization tokens. It has a subsection "browserEvents" where listeners like pageFocus, trackCursor and URLChanges can be given true and false values. These listeners show up as "browserEvent" in log files. The second section "applicationSpecificData" relates to LogUI application and is meant user details (userID). The last and most interesting section "trackingConfiguration" allows setting listeners on elements of DOM. The figure 1 has two listeners "querybox-focus" and "querybox-losefocus" which are triggered on the events "focus" and "blur" having names "QUERYBOX_FOCUS" and "QUERYBOX_BLUR". These events will show up in the log file under "interactionEvent". The line corresponding to "focus" will look as shown in figure 2. The first line shows interactionEvent and the second line has listener type "focus" and it's name "QUERYBOX_FOCUS". The names allow distinguishing two listeners of the same type (focus). Mozilla has a list of eventsWeb [41] that can be attached as listeners.

6.3 Test Page and Listeners

Sample Search²⁸ is a test HTML page (figure: 3) that has a query box at the top that returns the predefined query results. Query results are placed in an id called "left-rail", query result summary in "query-text" has id "result stats" and the image on the right "right-rail" with id "ENTITY_CARD". Click listeners are attached to the query bar, left rail and result stats, double click and right-click result stats and left rail respectively. Hover is attached to entity card and left rail. The query box has id "querybox" and has keyup, submit, focus and blur listeners. The web page has a focus (is cursor on that

²⁴<https://github.com/logui-framework/client/releases/tag/0.5.3d>

²⁵<https://github.com/logui-framework/server/releases/tag/v0.5.3-update>

²⁶<https://github.com/mozilla/geckodriver>

²⁷<https://docs.pytest.org/en/6.2.x/>

²⁸<https://github.com/logui-framework/example-apps/tree/main/sample-search>

```

let configurationObject = {
  logUIConfiguration: {
    endpoint: 'ws://localhost:8000/ws/endpoint/',
    authorisationToken: 'token',
    verbose: true,
    browserEvents: {
      blockEventBubbling: true,
      eventsWhileScrolling: true,
      URLChanges: true,
      contextMenu: true,
      pageFocus: true,
      trackCursor: true,
      cursorUpdateFrequency: 500,
      cursorLeavingPage: true,
    },
  },
  applicationSpecificData: {
    userID: 'userID',
  },
  trackingConfiguration: {
    'querybox-focus': {
      selector: '#input-box',
      event: 'focus',
      name: 'QUERYBOX_FOCUS',
    },
    'querybox-losefocus': {
      selector: '#input-box',
      event: 'blur',
      name: 'QUERYBOX_BLUR',
    },
  },
},

```

Figure 1: LogUI configuration object

web page), contextMenu, cursorUpdateFrequency and trackCursor listeners. A page resize listener is available by default.

6.4 Tests

6.4.1 Details. This automated test suite uses Selenium WebDriver whose behaviour is not completely understood. It may interfere and for this reason, all tests have been checked manually to verify the results using the same setup as automated tests. This is discussed further in section 6.5, for now, it will be assumed that Selenium WebDriver has one glitch - its cursor.

In a browser opened in WebDriver, the cursor is not constant in the sense that it does not always exist. In fact, the cursor does not exist until it is asked to exist and disappears immediately after. When a click event (left, right or double) is performed by WebDriver, the respective click listener is triggered but a listener that tracks the cursor is not. This listener is triggered during hover events only suggesting that the WebDriver cursor exists only for the duration of the hover. Another quirk of WebDriver is that certain actions which are impossible on a normal browser are possible. For instance, right-clicking locks many browser interactions like hover and left clicks. The menu bar that is opened needs to be closed to resume

interacting with the web page. WebDriver allows interacting with the web page, even after right-clicking.

It was noticed that a real cursor (of the machine) sometimes interfered with a test involving hover, to prevent this all browsers are used in headless mode. Testing starts with browser and interaction events in standalone mode - each of them is tested once and then three times (with a time delay if applicable). Lastly, a few tests with a combination of events are conducted to capture parts of common sequences of events performed by users - for example, hover on a URL before clicking. Lastly, a very complex test case has been made which combines all browser and interaction events and is discussed section 6.5. Tests that are not discussed in the following paragraphs are passing tests and can be understood using the description column in table 3.

6.4.2 Page Focus and Viewport. In F1 Safari logs browser events very quickly (before the web page has finished loading) and logs an out of focus browser. This is seen at the start only, opening a new tab to switch focus and revert is logged correctly. For V1 Safari logs viewport resizes but not the original dimensions (it started with). Firefox logs original dimensions but failed to log new dimensions. This is probably because this event is triggered once when LogUI configuration object is loaded and is not triggered again.

6.4.3 Query Box Interaction. Some Safari tests (Q1, Q2, Q3 and C1), log files could not be downloaded²⁹ and were repeated manually. All the tests passed and have hence been marked with a ✓*. Blur listener is triggered (becomes active but does not log anything) when a focus listener is triggered and it traditionally tracks the cursor. Given WebDriver's disappearing cursor, after some time (about 2-3 seconds), blur listener realizes that the query box is no longer in focus and logs a blur event later than it should be logged.

6.4.4 Clicks - Left, Right and Double Left. Three left clicks were logged by Safari but the test fails because query box did not log a blur event. The listener being tested responded correctly but not in the expected sequence and therefore is a failing test.

Right-click browser events are flaky on Safari, single interaction events are logged consistently but triple right clicks with a time interval of 5 seconds in between is logged for once in total (instead of thrice). A theory is that right-click listeners are disabled after being triggered once, this phenomenon was noticed in viewport resizes where browser events are not triggered consistently.

A double click is defined as two left clicks performed very quickly one after the other, WebDriver performs these clicks with the same timestamp according to log files. The expected logs are two left clicks (with the same timestamp) followed by a double click event. Safari logs one left-click when performing a single double click

²⁹<https://github.com/logui-framework/server/issues/3>

```

{"eventType": "interactionEvent",
 "eventDetails": {"type": "focus", "name": "QUERYBOX_FOCUS"},
 "sessionID": "ed92f713-6cb0-4d8a-8afa-e2e9190fc052",
 "timestamps": {"eventTimestamp": "2021-06-26T09:51:51.214Z",
 "sinceSessionStartMillis": 25, "sinceLogUILoadMillis": 25},
 "applicationSpecificData": {"userID": "test_rahul1"},
 "metadata": [],
 "applicationID": "0d57445b-a24a-4d3d-b6f1-b7c85b2f14a4",
 "flightID": "f90c2375-9fd0-4436-b099-9c47116cbcdd"},

```

Figure 2: Interaction event - query box_focus

Test	Safari	Firefox	Description
F1: test_page_focus	✗	✓	Page is in focus
V1: test_viewport_resize	✗	✗	Resize viewport
Q1: test_querybox_focus	✓	✓	Focus on query box
Q2: test_querybox_keyup	✓*	✓	Send keyboard input to query box
Q3: test_querybox_submit	✓*	✓	Submit query
Q4: test_querybox_blur	✓	✓	Blur query box on exit
T1: test_timestamp	✓	✓	Verify timestamp delay of two events
C1: test_click_once	✓*	✓	Click event is logged
C2: test_click_thrice	✓	✓	Click events are logged
RC1: test_right_click_once_browser	✗	✓	Right click once (browser event)
RC2: test_right_click_thrice_browser	✗	✓	Right click thrice (browser event)
RC3: test_right_click_once	✓	✓	Right click once
RC4: test_right_click_thrice	✗	✓	Right click thrice
D1: test_doubleclick_once	✗	✓	Double click
D2: test_doubleclick_thrice	✗	✗	Double click thrice
H1: test_hover_once	✓*	✓	Hover over element once
H2: test_hover_thrice	✗	✓	Hover over element thrice
H3: test_hover_different	✓	✓	Hover on different elements
D1: test_drag_drop	✗	✗	Drag and drop element
CT1: test_hover_hover_click_hover	✗	✗	Hover twice, click and hover
CT2: test_very_complex	✗	✗	Refer section 6.5

Table 3: Test Results.

(*) - failing tests when automated but pass in manual test.

but strangely logs none when performing three double clicks one after the other with a time delay in between them. Firefox counts one double click when three are performed with a time delay of 5 seconds. When a hover event is used instead of a time delay, logs show the expected behaviour of hovers followed by double clicks.

6.4.5 Hover. Each hover action performed by WebDriver should result in two entries in log files - mouseenter and mouseexit with a time difference of about 38 milliseconds. For H1 and H2, Safari did not log either hover events, manual testing of individual hover events on Safari however show satisfactory results. Drag and drop events do not trigger any listeners with WebDriver, in headless and GUI mode but they work when tested manually. A plausible explanation could not be found. It is possible that WebDriver flawed

because the listeners are triggered when testing manually. H3 hovers over two different elements, first a query result in left rail and then the image in right rail. Mouseenter is logged for both hovers but a mouseexit is not logged for right rail because the listener is not triggered.

6.4.6 Complex Tests. Timestamp tests work fine, sometimes a browser fails to log something correctly leading to timestamp tests failing but timestamps are not the reason for failing tests. Extensive testing of timestamps has confirmed that they are very accurate and have been observed to be off by a fraction of a second.

In CT1 two hover events are followed by a click with a 3 second gap between all of them but a click event is logged between mouseenter

and mouseexit of the second hover on Safari and Firefox, despite the time delay. It is strange behaviour. CT2 is a longer sequence of events discussed in section 6.5.

6.5 Search Session Simulation

A small sequence of events that are representative of normal human actions on a web page is simulated and tested manually on different browsers. The sequence is opening a browser, querying a string in the query box, clicking submit, hovering while reading on a query result (left rail) and an image (entity card) returned by the query and finally clicking on a query result (left rail). To test more actions, a double click is performed on result stat after clicking submit and a right-click on left rail at the end of the sequence.

Figure 4 shows the expected sequence of events on the first row, and the events observed in log files on respective browsers below. Each letter represents an event which is shown in table 4. Figures 5, 6 shows a comparison between testing on Safari, Firefox with WebDriver against manually on the respective browsers. Grey boxes represent statusEvents, yellow boxes browserEvents and green boxes interactionEvents. Inconsistencies in events are shown by outlining the box in red. Arrows can also be in red and double-sided to indicate the two events should be swapped. Some boxes have multiple letters, x2 shows the event happening twice and a plus sign is used to show that two events have occurred in the same box. The listener cursorUpdate is repetitive and is expected at every mouse event but WebDriver simulates clicks without a mouse and therefore this inconsistency is not shown. Empty boxes are for missing events.

In figure 4 going left to right on the top row, the first inconsistency is at the beginning, Safari switches browser and status events around meaning the second entry in the log file is "Started". Gecko skips browser event entirely. Manual testing shows that event D (keyup of querybox) M occurs twice, always and the same trend is seen on AppleWebKit but not on Gecko which shows D once. It is not known why a hover event (G) is logged before double click (F) but it is seen on both engines. The mismatch with cursor updates (event M) is probably because WebDriver cursor is active in between the hover events (G and H) resulting in log files showing cursor movement for event F. This behaviour of events switching around is seen again with the second hover on the right rail with events J (right rail mouseexit) and K (right-click left rail) being swapped, cursorUpdate is added to K even though it should not be. The timestamps show nothing out of the ordinary, looking at the log files (timestamps), it is logical to conclude that a double click event (F) happened in between the hover events (G and H).

Figure 5 compares the same sequence of events as above on Safari against a manual test on Safari. Manual test shows a statusEvent followed by browserEvent but the automated test switches this around. A delay of 5 seconds was given between double click (F) and hover (G) but the log files have reversed this. This was also observed in the automated test and is quite interesting. To attempt to understand this, a longer (10 to 15 second) time was used before the second hover event (J and K) and this time the events were

logged in the correct order. The mouseexit event was followed by a right-click and did not get swapped. This proves that waiting for a long time forces logging of hover events correctly - however, waiting for 10 seconds in the automated test did not change the results. The events were still swapped suggesting that the source of this problem is more complicated and perhaps a bug is in both LogUI and WebDriver.

Figure 6 compares the same sequence of events as above on Gecko against a manual test of Firefox (Gecko). In the automated test, Gecko did not log a browser event at the start but the manual test logged both status and browser events in the right order. The same time for hover was used here as in the previous manual test and the same observations were made i.e. waiting for a longer period forced a log entry of hover events in the correct order. This confirms the conclusions made in the above paragraph. It is easy to see how often a cursorUpdate event (+ M) occurs in manual tests but is absent in automated tests. This was done to highlight the non-existent cursor problem of Selenium WebDriver.

6.6 Evaluation

WebDriver simulates a certain sequence of event(s) and the same events (as per listeners attached) are expected in the log files. There should be a 1 : 1 correlation such that the sequence of events performed by WebDriver can be simply read off the log files. Inconsistencies exist when logs log extra events, log fewer events or when they log events different from those performed. These sequences can be easily checked by iterating over log files while making assertions to guarantee that events have occurred in the correct order and the correct number of times. It is not required to check timestamps because log files are already ordered by time. However, WebDriver automates user interactions meaning it can perform a sequence in half a second which would ideally take a human 5 seconds. In such tests, timestamps are vital.

Different types of listeners were tested on different browsers and figure 3 shows clearly which listeners work well and which don't on various browsers. Safari has lots of file not found errors which are considered passing tests because those listeners worked when testing manually.

Safari has 10/21 passing tests of which 4 could not be tested (file not found error) while Firefox has 16/21 passing tests. Tests involving timestamps worked well on all browsers tested, mouse events like hovering, drag drops and clicks are patchy - most likely because of how the mouse in WebDriver switches between existing and not existing. Keyboard events like sending keys have responded well.

7 FUTURE WORK

This is a list of things that can be done next and also includes some suggestions for improvements.

- (1) **Browsers:**
Test Chrome, Opera and Edge with their own engines (other than AppleWebKit).
- (2) **Selenium IDE:**

- **Realistic:**
Selenium IDE can perform record and replay attacks. This was previously dismissed because it involved manual work and we wanted to build an automated test suite. Given the problem with the non-existence of a mouse and the possibility to perform events that are not possible in a normal browser, Selenium IDE makes tests more realistic.
 - **Amount of Testable Listeners**
Selenium WebDriver[42] can perform few actions but there are many more listeners[41] which can be attached to elements in DOM. Selenium IDE can test most of these listeners which are not possible for WebDriver at the moment.
- (3) **Scalability:**
- Pytest is used to make fixtures[43] which set up the test infrastructure and help incorporate LogUI into the test suite. A massive drawback is that for tests to be run on multiple browsers, the browsers are provided as parameters to parameterized tests. Pytest starts the test suite by collecting all tests which means it opens all of these browsers, even if one test is to be run, Pytest collects all of them.
 - It was noticed that each browser takes roughly 0.7GB with GUI enabled and 0.45GB in headless mode. This is very expensive for RAM. A possible solution is to not use parameterized tests and run each test with a for loop on the list of browsers. This will make collecting and analyzing tests harder, code difficult to maintain and is bad practice in general. Selecting another framework like unittest[44] in Python or maybe something in another language which does first collect all tests is the best idea. It can also be solved by using lazy evaluation so that the browsers that are collected are not constructed until used by the test, Python does not support lazy evaluation.
- (4) **Configuration Object:**
The configuration object is hard coded into driver.js file. It would be very convenient to be able to read the object from a JSON file so that tests can personalize the configuration object. Unfortunately, my knowledge of Nginx and web technology in general was insufficient to figure out how to disable a CORS[45] error caused by reading a local JSON file.
- (5) **LogUI API:**
LogUI infrastructure is setup by simulating a user with Selenium. It would be convenient to create applications, flights, enable and disable flights, download log files and return authorization tokens without simulating users on browsers.

8 CONCLUSION

This paper has discussed the need for checking browser engine compatibility and evaluating it with a novel idea - compatibility tests in combination with web-based logging (LogUI) which exploits modern web technology to attach listeners to elements of DOM allowing LogUI to log data discretely and precisely without making any compromises of any kind. An extensive evaluation of user automation tools is done (section 4) and Selenium WebDriver was chosen for its vast selection of browsers. Unfortunately, many

of these browsers were not able to work resulting in a smaller than hoped for study but we did manage to cover TODO.

Different types of listeners were evaluated, sometimes in complex sequences to see how they react. The listeners are attached to elements of DOM and can therefore be considered part of that element. This gives an inside view of how the browser engine treats that particular element and action making it ideal for examining cross-browser inconsistencies. Both Research Questions 1 and 2 are addressed in table 3 which has an overview of listeners tested and their outcomes. Mouse (except hover) and keyboard events do not show any problems and drag and drop events could not be evaluated because they did not respond to Selenium.

One variable in this test suite is Selenium WebDriver, to address this the last test CT2 (section 6.5) performs a sequence of events that bring out multiple inconsistencies, some of which have been discussed individually earlier. Most importantly, it presents the findings with an easy to see and understand diagram with a realistic test. From these experiments, it can be concluded that while there are some browser inconsistencies such as loading of pages (events A and B in section 6.5), for all other use cases and purposes, they behave in the same way.

This is shown with a sequence diagram comparing expected behaviour of browsers to automated tests (figure 4) and manual tests against automated tests (figures 5 and 6). When listeners are tested individually, Safari has shown many failing tests which is remarkable. This is cleared up when the same tests are repeated manually suggesting that the problem is with Selenium WebDriver or with the AppleWebKit drivers used. Key Cross Browser Inconsistencies (XBI) identified are that browsers load differently and reorder some interactions such as hovers.

I would like to conclude this report by appreciating my supervisor, Dr. David Maxwell for his energy, leadership and wonderfully detailed advice all the time, including at weekends and other odd hours, Dr. Claudia Hauff for her thought provoking questions and guidance, my colleagues who worked along with me, Paul van Wijk, Sam van Berkel and Marc Visser. Lastly, my family and friends for always standing with me and believing in me.

Sample
Search

Search

[Click Me](#)

Results for **D**, appended after DOM load

Delft University of Technology
<https://www.tudelft.nl>
 Delft University of Technology also known as TU Delft, is the oldest and largest Dutch public technological university.

Delft - Wikipedia
<http://en.wikipedia.org/wiki/Delft>
 Delft is a popular tourist destination in the Netherlands, famous for its historical connections with the reigning House of Orange-Nassau, for its blue pottery, for...

Delftware - Wikipedia
<http://en.wikipedia.org/wiki/Delftware>
 Delftware or Delft pottery, also known as Delft Blue (Dutch: Delfts blauw), is a general term now used for Dutch tin-glazed earthenware, a form of faience.

Visit Delft - These are the best things to do - Holland.com
<https://www.holland.com/>
 Delft is famous for its ceramic Delft Blue pottery. It is known as the birth place of the famous painter Johannes Vermeer, known from "the girl with the Pearl". And it is known as a charming canal-ringed town with historical monuments and medieval architecture.

Homepage | Gemeente Delft
<http://www.delft.nl>
 Municipality Delft · Municipal services. Moving from abroad · Reporting a change of address · Official matters. Immigration procedure · Housing, Housing in the ...



Delft
 City in The Netherlands

Delft, a canal-ringed city in the western Netherlands, is known as the manufacturing base for Delftware, hand-painted blue-and-white pottery. In its old town, the medieval Oude Kerk is the burial site of native son and Dutch Master painter Johannes Vermeer. Once the seat of the royal House of Orange, the 15th-century Nieuwe Kerk houses the family's tombs and overlooks Delft's lively market square.

Area 24.06 sq km
City Hall Delft City Hall
Province South Holland

Figure 3: Sample search web page

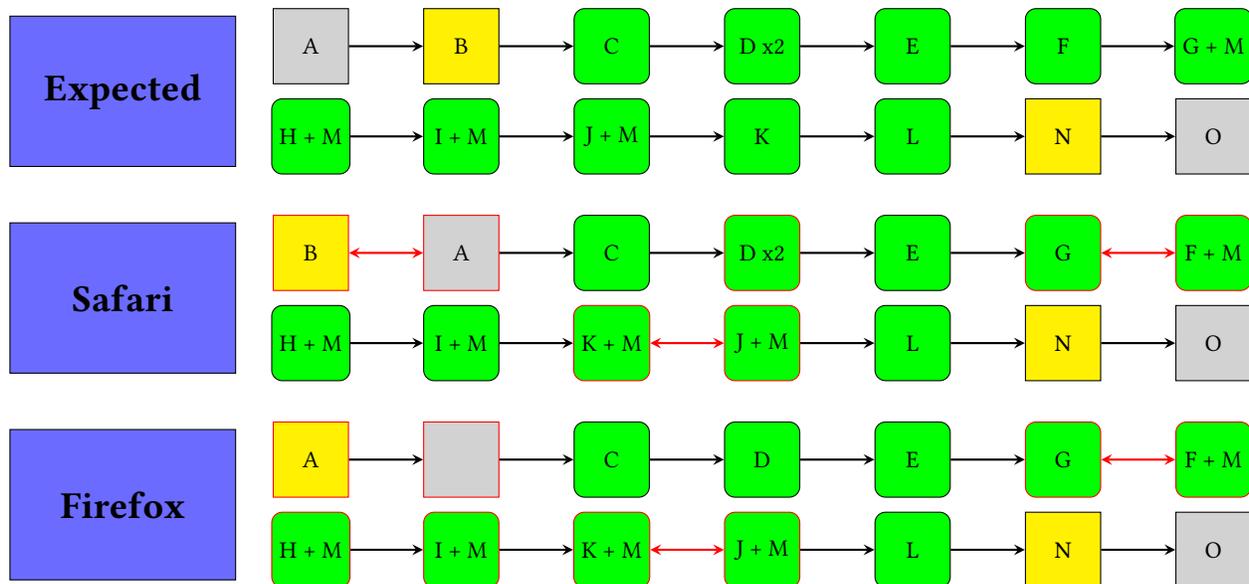


Figure 4: Case Study: Expected vs Chrome log files vs Firefox log files

REFERENCES

[1] Martine Dowden and Michael Dowden. *Compatibility and Defaults*, pages 127–143. Apress, Berkeley, CA, 2020. ISBN 978-1-4842-5750-0. doi: 10.1007/978-1-4842-5750-0_5. URL https://doi.org/10.1007/978-1-4842-5750-0_5. last accessed on 26 June 2021.

[2] Joel Martin and David Levine. Property-based testing of browser rendering engines with a consensus oracle. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, volume 02, pages 424–429, 2018. doi: 10.1109/COMPSAC.2018.10270.

[3] Html5. URL <https://developer.mozilla.org/en-US/docs/Glossary/HTML5>. last accessed on 26 June 2021.

[4]

[5] w3c. URL <https://www.w3.org/>. last accessed on 26 June 2021.

[6] Ietf. URL <https://www.ietf.org/>. last accessed on 26 June 2021.

Symbol	Meaning	Symbol	Meaning
A	Started	I	Right rail mouseenter
B	Page has focus	J	Right rail mouseexit
C	Querybox Focus	K	Right click left rail
D	Keyup (querybox change)	L	Left click left rail
E	Click submit and query submit	M	Cursor position updated
F	Double click stats	N	Page does not have focus
G	Left rail mouseenter	O	Stopped
H	Left rail mouseexit		

Table 4: Meaning of letters.

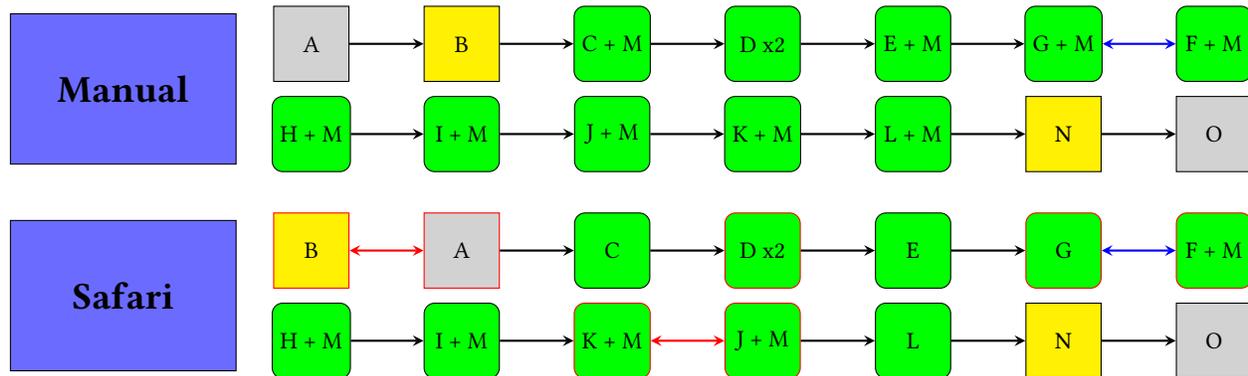


Figure 5: Case Study: Manual Chrome vs Chrome log files

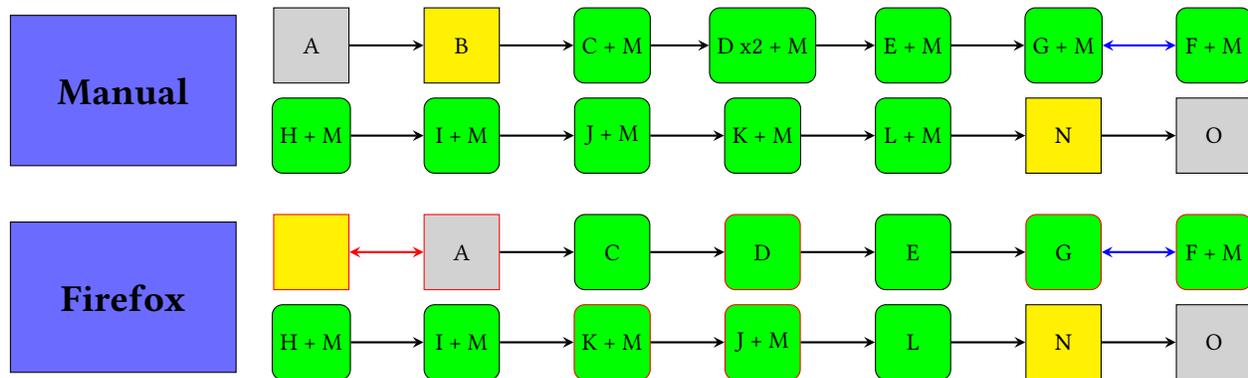


Figure 6: Case Study: Manual Firefox vs Firefox log files

[7] M. Watanabe1, W. F. Christian, and D. Silva. Towards cross-browser incompatibilities detection: asystemic literature review. volume 10. International Journal of Software Engineering & Applications (IJSEA), 2019. URL https://scholar.google.com/scholar?as_sdt=0%2C5&btnG=&hl=en&inst=6173373803492361994&q=Towards%20cross-browser%20incompatibilities%20detection%3A%20asystemic%20literature%20review. last accessed on 26 June 2021.

[8] Firefox user agent string reference. URL <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent/Firefox>. last accessed on 26 June 2021.

[9] Achim D. Brucker and Michael Herzberg. Formalizing (web) standards. In Catherine Dubois and Burkhart Wolff, editors, *Tests and Proofs*, pages 159–166, Cham, 2018. Springer International Publishing. ISBN 978-3-319-92994-1. last accessed on 26 June 2021.

[10] Feetu Nyrhinen and Tommi Mikkonen. Web browser as a uniform application platform: How far are we? In *2009 35th Euromicro Conference on Software Engineering and Advanced Applications*, pages 578–584, 2009. doi: 10.1109/SEAA.2009.37. last accessed on 26 June 2021.

[11] Antero Taivalsaari, Tommi Mikkonen, Dan Ingalls, and Krzysztof Palacz. Web browser as an application platform. In *2008 34th Euromicro Conference Software Engineering and Advanced Applications*, pages 293–302, 2008. doi: 10.1109/SEAA.2008.17. last accessed on 26 June 2021.

[12] Fagner Christian Paes and Willian Massami Watanabe. Layout cross-browser incompatibility detection using machine learning and dom segmentation. SAC '18, page 2159–2166, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450351911. doi: 10.1145/3167132.3167364. URL <https://doi.org/tudelft.idm.oclc.org/10.1145/3167132.3167364>. last accessed on 26 June 2021.

[13] Zhenyue Long, Guoquan Wu, Yifei Zhang, Wei Chen, and Jun Wei. Poster: Repair cross browser layout issues by combining learning and search-based technique. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 470–473, 2021. doi: 10.1109/ICST49551.2021.00062.

[14] Zhen Xu and James Miller. Cross-browser differences detection based on an empirical metric for web page visual similarity. 18(3), April 2018. ISSN 1533-5399.

Examining Cross Browser Inconsistencies with Web Based Event Loggers

- doi: 10.1145/3140544. URL <https://doi-org.tudelft.idm.oclc.org/10.1145/3140544>. last accessed on 26 June 2021.
- [15] Shauvik Roy Choudhary, Mukul R. Prasad, and Alessandro Orso. X-pert: A web application testing tool for cross-browser inconsistency detection. *ISSTA 2014*, pages 417–420, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450326452. doi: 10.1145/2610384.2628057. URL <https://doi-org.tudelft.idm.oclc.org/10.1145/2610384.2628057>. last accessed on 26 June 2021.
- [16] Sonal Mahajan, Abdulmajeed Alameer, Phil McMinn, and William G. J. Halfond. Automated repair of layout cross browser issues using search-based techniques. *ISSTA 2017*, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350761. doi: 10.1145/3092703.3092726. URL <https://doi-org.tudelft.idm.oclc.org/10.1145/3092703.3092726>. last accessed on 26 June 2021.
- [17] Dom. URL https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction. last accessed on 26 June 2021.
- [18] David Maxwell and Claudia Hauff. LogUI: Contemporary Logging Infrastructure for Web-Based Experiments. In *Advances in Information Retrieval (Proc. ECIR)*, pages 525–530, 2021. last accessed on 26 June 2021.
- [19] S Kundu. Web testing: Tools, challenges and methods. volume 9, 2012. URL <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.403.1459&rep=rep1&type=pdf>. last accessed on 26 June 2021.
- [20] B. Al-Ahmed and K. Debei. Survey of testing methods for web applications. volume 9, 2020. URL https://www.researchgate.net/profile/Bilal_Al-Ahmad/publication/348000478_Survey_of_Testing_Methods_for_Web_Applications/links/5fec692a92851c13fed40a1c/Survey-of-Testing-Methods-for-Web-Applications.pdf. last accessed on 26 June 2021.
- [21] Emil Alégroth, Arvid Karlsson, and Alexander Radway. Continuous integration and visual gui testing: Benefits and drawbacks in industrial practice. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, pages 172–181, 2018. doi: 10.1109/ICST.2018.00026. last accessed on 26 June 2021.
- [22] Jiujiu Yu. Exploration on web testing of website. *Journal of Physics: Conference Series*, 1176:022042, mar 2019. doi: 10.1088/1742-6596/1176/2/022042. URL <https://doi.org/10.1088/1742-6596/1176/2/022042>. last accessed on 26 June 2021.
- [23] Ricca, F Marchetta, A and P Tonella. A case study based comparison of web testing techniques applied to ajax web applications. 2008. doi: 10-1007/s10009-008-0086-x. URL <https://link-springer-com.tudelft.idm.oclc.org/content/pdf/10.1007/s10009-008-0086-x.pdf>. last accessed on 26 June 2021.
- [24] Ajax. URL <https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX>. last accessed on 26 June 2021.
- [25] Vidroha Debroy, Lance Brimble, Matthew Yost, and Archana Erry. Automating web application testing from the ground up: Experiences and lessons learned in an industrial setting. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, pages 354–362, 2018. doi: 10.1109/ICST.2018.00042. last accessed on 26 June 2021.
- [26]
- [27] Puppeteer. URL <https://developers.google.com/web/tools/puppeteer/>. last accessed on 26 June 2021.
- [28] Paola Guzmán Castillo, Pau Arce Vila, and Juan Carlos Guerri Cebollada. Automatic qoe evaluation of dash streaming using itu-t standard p.1203 and google puppeteer. In *Proceedings of the 16th ACM International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks, PE-WASUN '19*, page 79–86, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450369084. doi: 10.1145/3345860.3361519. URL <https://doi-org.tudelft.idm.oclc.org/10.1145/3345860.3361519>. last accessed on 26 June 2021.
- [29] Nightwatchjs. URL <https://nightwatchjs.org/>. last accessed on 26 June 2021.
- [30] Webdriver. URL <https://www.selenium.dev/documentation/en/webdriver/>. last accessed on 26 June 2021.
- [31] Downloads, . URL <https://www.selenium.dev/downloads/>. last accessed on 26 June 2021.
- [32] Boni García, Micael Gallego, Francisco Gortázar, and Mario Muñoz-Organero. A survey of the selenium ecosystem. *Electronics*, 9(7), 2020. ISSN 2079-9292. doi: 10.3390/electronics9071067. URL <https://www.mdpi.com/2079-9292/9/7/1067>. last accessed on 26 June 2021.
- [33] Mouseevent. URL <https://developer.mozilla.org/en-US/docs/Web/API/MouseEvent>. last accessed on 26 June 2021.
- [34] .
- [35] Paruchuri Ramya, Vemuri Sindhura, and P. Vidya Sagar. Testing using selenium web driver. In *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pages 1–7, 2017. doi: 10.1109/ICECCT.2017.8117878.
- [36]
- [37] .
- [38] Basic concepts, . URL <https://github.com/logui-framework/server/wiki/Basic-Concepts#applications-flights-and-sessions>. last accessed on 26 June 2021.
- [39] Configuration object. URL <https://github.com/logui-framework/client/wiki/Configuration-Object>. last accessed on 26 June 2021.
- [40] Java script object notation. URL <https://www.json.org/>. last accessed on 26 June 2021.
- [41] Event reference. URL <https://developer.mozilla.org/en-US/docs/Web/Events>. last accessed on 26 June 2021.
- [42] Selenium mouse actions in detail, . URL https://www.selenium.dev/documentation/en/support_packages/mouse_and_keyboard_actions_in_detail/. last accessed on 26 June 2021.
- [43] Pytest fixtures. URL <https://docs.pytest.org/en/latest/how-to/fixtures.html>. last accessed on 26 June 2021.
- [44]
- [45] Cors. URL <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>. last accessed on 26 June 2021.