Astana IT University,

Department of Computer Science and Engineering,

Advanced Quality Assurance

BrowserStack and Sauce Labs for Cross-Browser Testing

Abdikenov Nurzhas Nursabituly

CSE-2405M

Adamova A.D

Astana, 2025

I. Introduction

In today's fast-paced digital ecosystem, users access web applications across a diverse range of browsers, devices, and operating systems. Ensuring consistent user experience across this spectrum is both essential and technically challenging. **Cross-browser testing** has emerged as a vital quality assurance practice to validate that web applications function correctly and look consistent regardless of the environment.

This project focuses on two leading cloud-based testing platforms — **BrowserStack** and **Sauce Labs** — which enable automated cross-browser testing at scale. These services provide access to hundreds of real browsers and devices, eliminating the need for complex local setups and enabling developers and QA engineers to test their applications in real-world conditions.

The selection of the topic **"BrowserStack and Sauce Labs for Cross-Browser Testing"** is primarily driven by prior hands-on experience with both tools during earlier course assignments. These experiences allowed for an initial exploration of their capabilities, highlighting their strengths in automated testing, real-device access, and integration with modern CI/CD workflows. The practicality and versatility of BrowserStack and Sauce Labs made a lasting impression, prompting a deeper investigation into their comparative advantages and limitations. Moreover, as both platforms are widely used in industry settings, this topic offers not only academic relevance but also direct practical value for future professional applications in quality assurance and software testing.

The objective of this project is to evaluate and compare the practical implementation of BrowserStack and Sauce Labs for cross-browser testing using **Selenium WebDriver with Python**. It demonstrates how each platform can be used to run automated tests across multiple browser-device combinations, streamlining the testing process for modern web applications.

The target application chosen for testing is https://www.saucedemo.com, a deliberately designed e-commerce demo site that supports functional automation testing. This platform allows testing of key workflows like login, add-to-cart, and navigation, which serve as the foundation for our comparative analysis.

The introduction concludes by outlining the structure of this report:

- The **main part** details the theoretical foundation of cross-browser testing, tools used, testing process, test cases, and execution on each platform;
- The **results section** presents performance comparisons, screenshots, logs, and highlights from test sessions;
- Finally, the **conclusion** reflects on the practical value of the tools and insights gained during the project.

II. Main Part

A. Theoretical Basis

- *Cross-Browser Testing and Its Importance*

Cross-browser testing is the process of verifying that a web application behaves consistently across different browsers, operating systems, and device configurations. It ensures functional correctness, visual consistency, and usability regardless of the platform used by the end user. Variations in browser rendering engines, JavaScript interpretation, and CSS support can cause layout shifts, broken interactions, or inconsistent performance. For quality assurance teams, this form of testing is essential to minimize user experience issues and avoid platform-specific bugs [1].

As web applications are increasingly expected to be responsive and accessible across both desktop and mobile environments, the need for scalable and efficient cross-browser testing solutions has grown. Manual testing on physical devices is often impractical due to the high cost and time involved. Hence, cloud-based platforms such as **BrowserStack** and **Sauce Labs** have become integral to modern testing strategies [1].

- *Selenium WebDriver*

Selenium WebDriver is one of the most widely used tools for browser automation. It allows testers to write scripts in multiple programming languages including Python, Java, and C#, and interact directly with the browser through a driver interface. WebDriver automates user interactions like clicking buttons, filling forms, and navigating between pages, thereby supporting robust end-to-end testing.

Its architecture supports integration with test frameworks such as unittest or pytest and CI/CD pipelines, making it suitable for both local development and enterprise-grade automated testing scenarios [1].

- *BrowserStack and Sauce Labs*

BrowserStack and Sauce Labs are commercial cloud platforms that provide access to real browsers and devices over the internet. They allow users to run automated Selenium tests

without the need for maintaining local device farms. Both platforms support automated testing, manual live testing, debugging tools, network throttling, and visual validation.

BrowserStack offers seamless integration with CI tools such as Jenkins and GitHub Actions and supports automated screenshots and performance testing. Sauce Labs provides extended analytics, test result dashboards, and compatibility with Appium for mobile testing. These platforms improve test coverage, scalability, and speed while ensuring high test reliability [1, 2].

- *Software Quality Standards*

Cross-browser testing aligns with international quality assurance standards such as ISO/IEC 25010, which defines quality attributes like functionality, usability, compatibility, and performance efficiency. Adhering to these standards ensures that the software meets stakeholder expectations in terms of quality and user satisfaction [1]. In addition, frameworks such as ISTQB define best practices for test design and execution, advocating risk-based and exploratory testing as part of the test strategy. Applying these practices ensures that testing is not only thorough but also aligned with business priorities [3].

- *Development and Testing Environments*

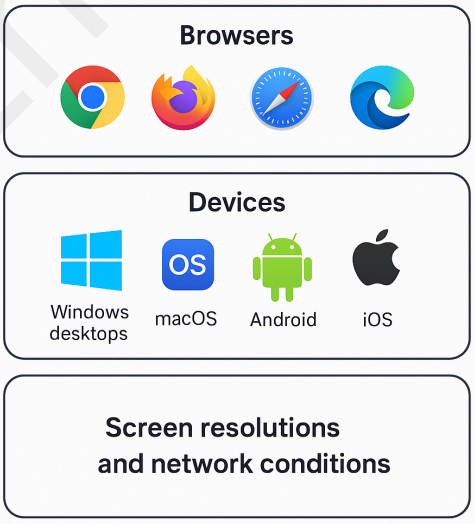

Figure 1: Browsers, Devices, Screen Resolutions

Using BrowserStack and Sauce Labs, testers can dynamically configure these environments, run tests in parallel, and collect detailed logs and video recordings for analysis. This dramatically reduces time-to-feedback in agile development cycles and supports test automation at scale [1, 2].

B. Methodology

- *Overall Testing Approach*

The testing process in this project followed a **task-driven exploratory testing** approach integrated within an **Agile development mindset**. Rather than relying strictly on pre-written manual test cases, the testing efforts were iterative and focused on quickly evaluating functionality across various environments using automated scripts. This mirrors common QA practices in modern software teams, where rapid feedback and flexibility are essential for continuous integration and delivery cycles [1].

Test automation was prioritized for repeatable and stable workflows—such as login, product search, and cart interactions—while exploratory manual testing was used for visually inspecting rendering issues and browser-specific anomalies. This hybrid approach enabled both efficient verification of core functionality and discovery of edge cases that may not be immediately apparent in scripted tests.

- *Toolchain and Automation Flow*

Selenium WebDriver (Python) was used to implement all test scenarios. The test scripts were written using PyTest and executed across multiple environments using both **BrowserStack** and **Sauce Labs**. These services allow for remote browser testing on real devices and virtual machines through cloud infrastructure.

The automation pipeline followed a consistent pattern:

1. Initialize the WebDriver session using the respective platform's remote grid URL.
2. Load the target application (https://www.saucedemo.com).
3. Execute predefined actions (login, add to cart, verify cart badge).
4. Capture results, console logs, and optionally screenshots or video (enabled by default in both services).

5. End session and record status.

The test environments were selected to include a combination of:

Table A: The Test Environments

Category	Configurations
Browsers	Chrome, Firefox, Safari, Edge
Operating Systems	Windows 10, macOS Ventura, Android 13, iOS 17
Resolutions	Desktop (1920×1080), Tablet (768×1024), Mobile (414×896)

Parallel execution was configured to optimize run time and simulate multi-user interactions under different environments. This was particularly valuable for measuring performance and responsiveness across platforms [1, 2].

- *Configuration and CI Integration*

While continuous integration (CI) is a common practice in modern software testing, this project did not include practical implementation of CI tools such as GitHub Actions. However, it is worth noting that both **BrowserStack** and **Sauce Labs** support integration with CI/CD pipelines through secure API access, environment configuration files, and automated test result dashboards. These capabilities make both platforms suitable for teams adopting scalable DevOps workflows and aiming for continuous testing across diverse environments [2], [5].

C. Project Execution

- *Task Definition*

The main task was to automate core user interactions on a web application and execute those tests across real browser-device environments using **BrowserStack** and **Sauce Labs**. The chosen test site, https://www.saucedemo.com, offers stable workflows and is widely used for UI automation testing [5].

The test scenarios implemented included:

Table B: Test Cases Executed on SauceDemo

Test Case ID	Description	Expected Result
TC-01	Login with valid credentials	Redirects to the inventory page `/inventory.html`
TC-02	Add a product to the cart	"Add to Cart" button changes to "Remove"; item added
TC-03	Verify cart icon updates	Cart badge displays with value `1`
TC-04	View product detail page	Product name on detail page matches clicked item
TC-05	Logout from the session	Redirects to login screen `/` and session is cleared

These were selected to reflect typical e-commerce user behaviors and were implemented using **Selenium WebDriver in Python** [4].

- *Test Planning*

Configurations were taken from **Table A**

Test runs were configured to execute in parallel on both BrowserStack and Sauce Labs, minimizing total execution time and highlighting UI or behavioral inconsistencies across platforms [1], [2].

- *Test Case Implementation*

All test scripts were developed in **Python** using **Selenium WebDriver** with modular structure for readability and reuse [4]. Each script initialized a remote WebDriver session using platform-specific capabilities and executed one functional test case.

Code 1: Login Test on Chrome, Windows 10 (Local WebDriver)

```
try:
    wait = WebDriverWait(driver, 10)
    driver.get("https://www.saucedemo.com")

    # TC-01: Login
    wait.until(EC.presence_of_element_located((By.ID,
"user-name"))).send_keys("standard_user")
    driver.find_element(By.ID,
"password").send_keys("secret_sauce")
    driver.find_element(By.ID, "login-button").click()
    wait.until(EC.presence_of_element_located((By.ID,
"inventory_container")))
    print("TC-01 Login successful")
assert "inventory.html" in driver.current_url
driver.quit()
```

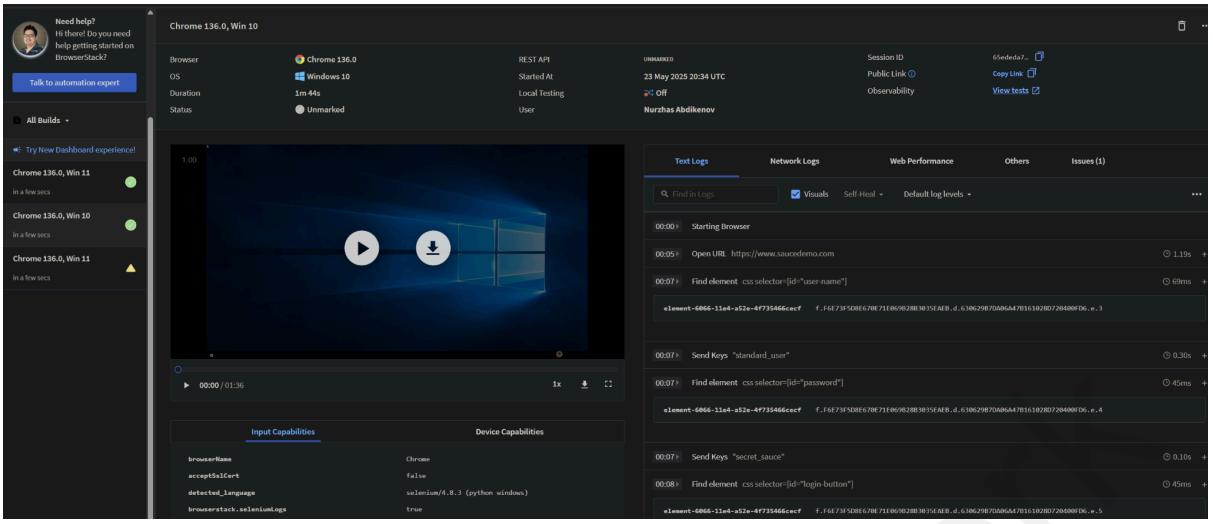

Figure 2: Login Test on BrowserStack Dashboard

Screenshots of other tests as well as code snippets are provided in the **Appendix.**

Code 2: Remote WebDriver Setup for Sauce Labs

```
from selenium import webdriver
sauce_options = {
    "browserName": "firefox",
    "platformName": "MacOS",
    "browserVersion": "latest",
    "sauce:options": {
        "name": "Login Test",
        "build": "Build 1"
    }
}

driver = webdriver.Remote(

command_executor="https://USERNAME:ACCESS_KEY@sauceurl.com/wd/hub"
,

options=webdriver.ChromeOptions().set_capability("sauce:options",
sauce_options)
)
```

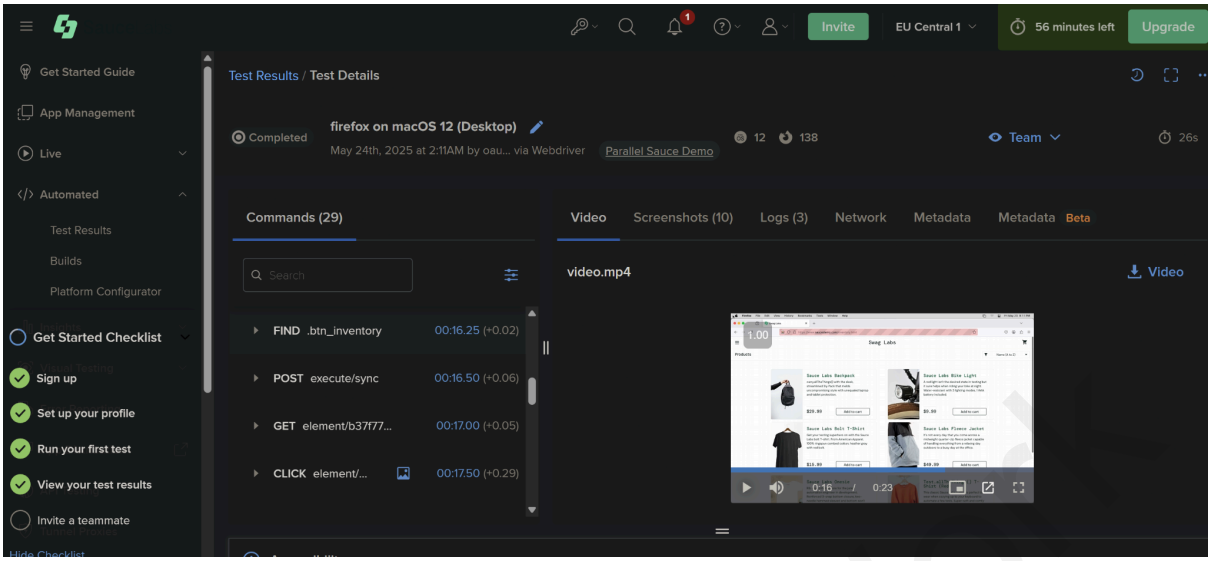

Figure 3: Sauce Labs Dashboard

BrowserStack uses a nearly identical setup but with platform-specific endpoint and capability syntax [2], [5].

- *Execution and Artifacts*

Each test case was executed on the selected browser and platform configurations using BrowserStack and Sauce Labs. The platforms automatically recorded session-level artifacts, including real-time video playback, screenshots at each step, console logs, and detailed session metadata [2], [5]. These artifacts were used to verify the correctness of each test case, identify failures, and compare results across environments.

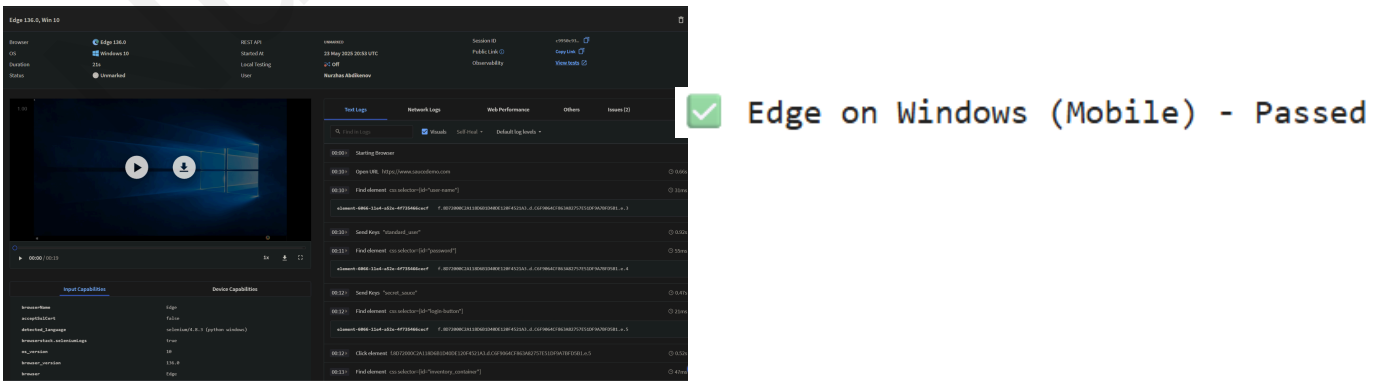

Figure 4: Session Log Output Captured in BrowserStack Dashboard

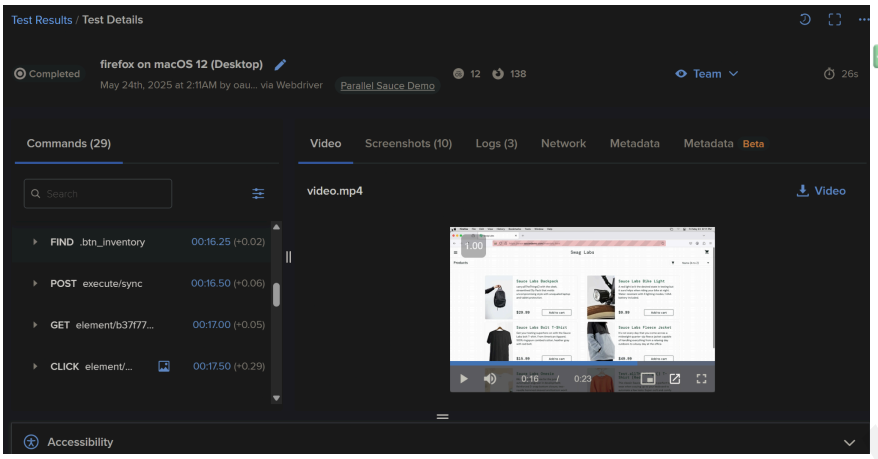

Figure 5: Test Execution Report on Sauce Labs

The final artifacts included:

- Console logs
- Screenshot comparisons
- Test duration metrics
- Success/failure status per environment

III. Results

The test execution covered five core functional test cases across multiple browsers and operating systems:

- **TC-01**: Login with valid credentials
- **TC-02**: Add a product to the cart
- **TC-03**: Verify the cart badge
- **TC-04**: View product detail page
- **TC-05**: Logout from the session

Each of these test cases was implemented using Selenium WebDriver [4] and executed on both **BrowserStack** and **Sauce Labs** using configurations defined in Table A. Both platforms provided session logs, screenshots, and video recordings via their dashboards for result verification [2], [5].

- *Performance and Execution Time*

BrowserStack demonstrated superior performance during test execution. On average, BrowserStack completed each test case approximately **5 seconds faster** than Sauce Labs for equivalent OS and browser combinations. One significant exception occurred on **macOS Safari**, where the test took over **3 minutes and 30 seconds**, likely due to remote VM provisioning delays [2], [4].

- *Parallel Execution and Session Limits*

A key difference in platform behavior was observed during parallel test runs:

- **BrowserStack** supported **5 concurrent sessions** on its free tier, allowing full parallel execution of all test cases [2], [9].
- **Sauce Labs** limited the trial user to **1 parallel session** and **60 minutes of total test time**, significantly increasing execution duration [5], [9].

- *Cost and Trial Access*

BrowserStack's trial plan enabled unrestricted test execution with no session limits, which proved highly advantageous for batch testing and multi-browser workflows [2]. In contrast, Sauce Labs imposed stricter controls on concurrency and access time, limiting its efficiency in this project [5], [9].

Table C: Summary Table

Platform	**Avg. Test Time Per Case**	**Safari (macOS) Time**	**Parallel Support**	**Trial Restrictions**
BrowserStack	21sec,~5 seconds faster	3 min 30 sec	5 parallel tests	Free with no hard limits
Sauce Labs	26sec.~5 seconds slower	N/A (not completed)	1 parallel test	60 minutes, 1 parallel cap

IV. Conclusion

This project evaluated the application of cross-browser testing using two widely adopted platforms: **BrowserStack** and **Sauce Labs**. Through the automation of five core test cases on the SauceDemo application—login, add-to-cart, cart verification, product detail access, and logout—the study demonstrated the technical feasibility and practical differences between the two tools in a controlled environment.

The hands-on experimentation confirmed that both platforms support Selenium WebDriver-based test automation across a wide range of browser and operating system combinations [4]. However, key differences emerged in performance, accessibility, and user experience. **BrowserStack consistently delivered faster execution**, averaging five seconds less per test compared to Sauce Labs, with the exception of macOS Safari, which showed a significant delay [2]. **BrowserStack's interface and user instructions were also more intuitive**, allowing quicker onboarding and test setup.

Sauce Labs, while technically robust, imposed trial limitations that included a **60-minute cap** and **one-parallel-test restriction**, limiting its suitability for batch testing or broader exploratory runs [5], [9]. Nonetheless, it remains a valuable tool with strong test reporting, session tracking, and analytics capabilities.

Overall, **both BrowserStack and Sauce Labs are excellent tools for automation testing**, offering enterprise-grade infrastructure, real-device support, and CI integration features [2], [5]. From a practical standpoint, **BrowserStack's more user-friendly interface and trial flexibility** made it better suited for efficient and scalable test execution in this context.

This project enhanced practical skills in test planning, Selenium scripting, remote WebDriver setup, and multi-environment testing. It also emphasized the importance of tool evaluation based on execution speed, usability, and licensing—a critical consideration in real-world QA workflows [1], [2], [5], [9].

V. References

[1] J. L. Mitchell and R. Black, *Advanced Software Testing, Vol. 3: Guide to the ISTQB Advanced Technical Test Analyst Certification*, Rocky Nook, 2015.

[2] C. Y. Laporte and A. April, *Software Quality Assurance*, Wiley, 2017.

[3] SeleniumHQ, "Selenium WebDriver," Selenium Documentation.

[4] BrowserStack, "Automate Documentation," BrowserStack Documentation.

[5] Sauce Labs, "Getting Started with Selenium in Python," Sauce Labs Documentation.

[6] OWASP Foundation, "OWASP Top 10 Security Risks," OWASP Project.

[7] W3C, "WebDriver," World Wide Web Consortium (W3C), 2023.

[8] T. A. Khan, "Comparative Analysis of Cloud-Based Cross-Browser Testing Tools," *International Journal of Software Testing Technology*, vol. 8, no. 2, pp. 56–63, 2022.

[9] S. Patel and M. Shah, "Automation Testing: Selenium WebDriver Integration with BrowserStack and Jenkins," in *Proc. IEEE Int. Conf. Smart Tech*, 2021, pp. 121–126.

VI. Appendix

A. Code Snippets

Github link of whole codebase

Code 3: Setting Credentials for BrowserStack

```
from selenium import webdriver

USERNAME = 'your-username'
ACCESS_KEY = 'your-key'

desired_cap = {
    'os': 'Windows',
    'os_version': '10',
    'browserName': 'Chrome',
    'browser_version': 'latest',
    'name': 'Full Test Flow - SauceDemo',
    'build': 'Jupyter Demo Build',
    'browserstack.debug': True
}

driver = webdriver.Remote(

command_executor=f'https://{USERNAME}:{ACCESS_KEY}@hub.browserstac
k.com/wd/hub',
    desired_capabilities=desired_cap
)
```

Code 4: Config for Parallel Testing

```
test_configs = [
    {"browser": "Chrome", "os": "Windows", "os_version": "10",
"device": "Desktop", "resolution": "1920x1080"},
    {"browser": "Firefox", "os": "macOS", "os_version": "Ventura",
"device": "Desktop", "resolution": "1920x1080"},
    {"browser": "Safari", "os": "macOS", "os_version": "Ventura",
"device": "Tablet", "resolution": "768x1024"},
    {"browser": "Edge", "os": "Windows", "os_version": "10",
"device": "Mobile", "resolution": "414x896"},
    {"browser": "Chrome", "os": "Android", "os_version": "13",
"device": "Mobile", "resolution": "414x896"},
```

```
]

# Define the test function
def run_test(config):
    desired_cap = {
        'browserName': config["browser"],
        'os': config["os"],
        'osVersion': config["os_version"],
        'name': f'{config["browser"]} on {config["os"]}
({config["device"]})',
        'build': 'Parallel Test Demo',
        'browserstack.debug': True,
        'resolution': config["resolution"]
    }
```

B. Figures

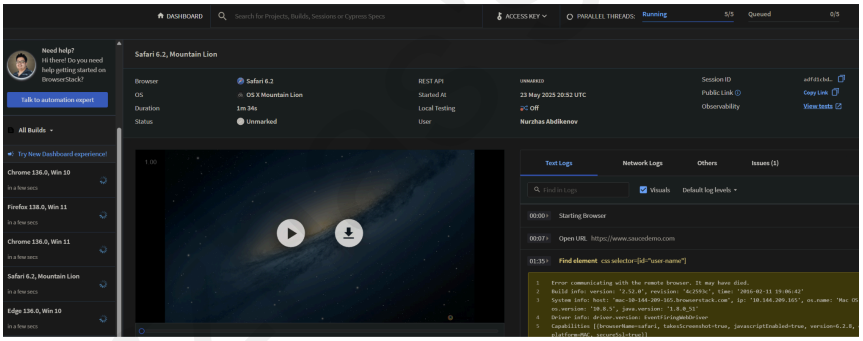

Figure 6: Parallel Test Cases

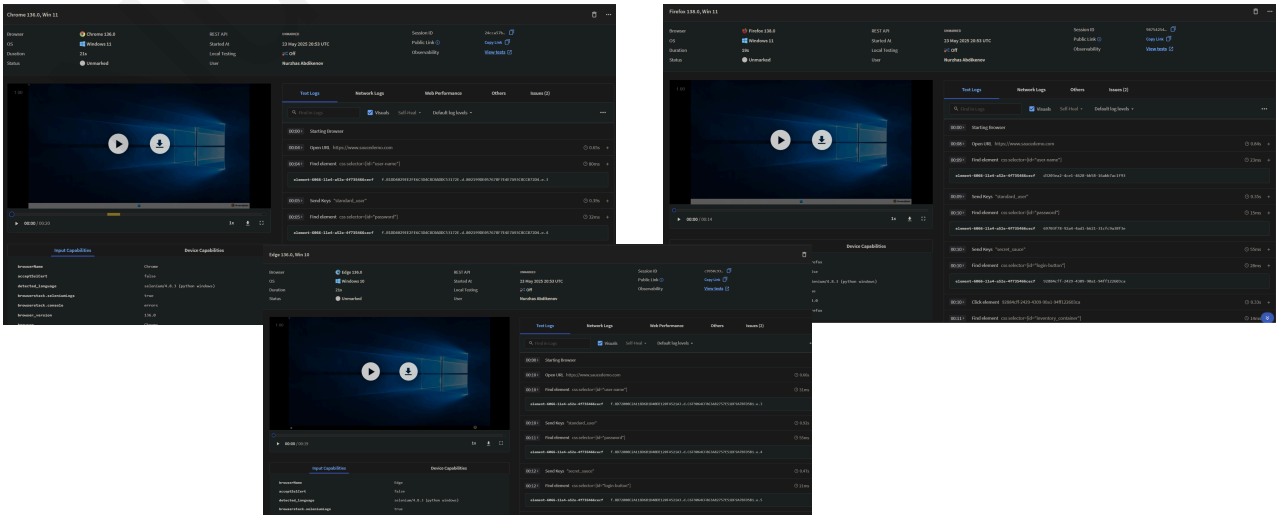

Figure 7: BrowserStack Dashboard of Tests on Chrmoe, Edge and Firefox

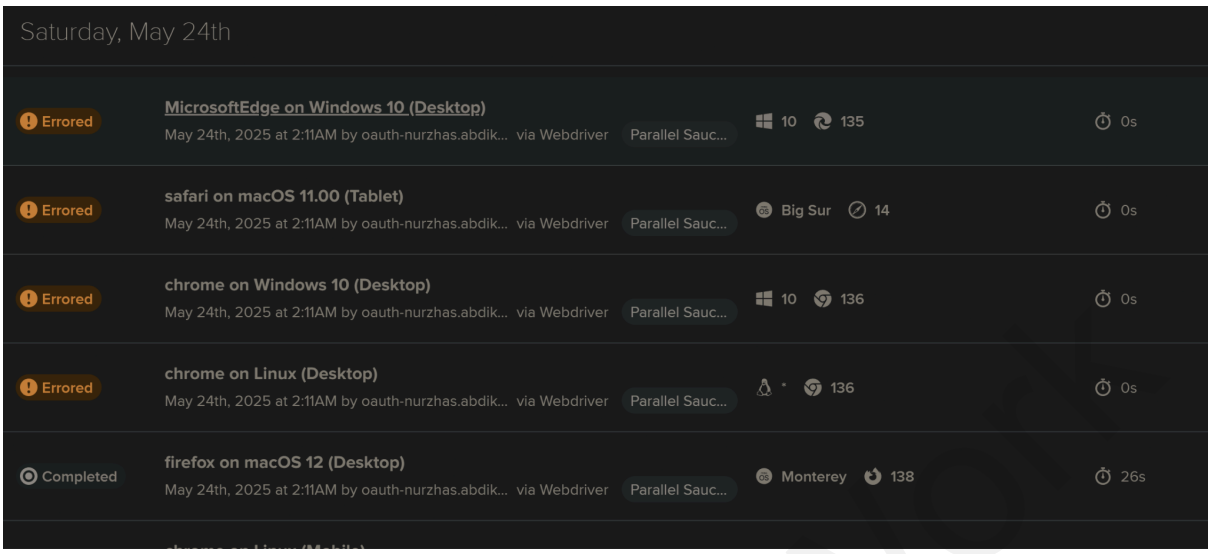

Figure 8: SauceLabs 5 Parallel Execution Fail