# Selenium

## History and Motivation

# History and Motivation - Selenium

- Created by Jason Huggins
- Motivation: to create and efficient automated testing over time consuming work of manual testing
- JavaScriptRunner developed in 2004 became open source program Selenium Core
- Four Components
  - Selenium IDE
  - Selenium Grid
  - Selenium RC
  - WebDriver

# History and Motivation – Selenium IDE

- Created by Shinya Kasanti
- Motivation: increase the speed of writing test cases
- Available through Firefox since 2006
- Automate browsers with recording and playback feature
- Built-in help and test results reporting
- No programming language experience required

# History and Motivation – Selenium Grid

- Created by Patrick Lightbody
- Motivation: speed up the time to execute test
- Runs by hub-nodes model
- Runs in conjunction with Selenium RC
- Perform multiple tests in parallel on different browers

# History and Motivation – Selenium RC

- Created by Paul Hammant
- Motivation: create a tool that runs against the "Same Origin Policy"
- Same Origin Policy
  - Security policy that ensures content of original browser cannot be accessed by a script from another browser.
  - Only codes loaded within the browser can operated with the site's original domain
  - Restrict JavaScript code to access elements from other domains

# History and Motivation – Selenium RC continues

- How it overcame the same origin policy
  - Acts as a client configured HTTP proxy and mask AUT under a false URL
  - Can be used with web application that is not from the same domain as Javascript code
- Selenium RC  Selenium Core  Browsers
- Supports many languages
- Has mature API
- Perform looping and conditional operations
- Support Data-Driven Testing

# History and Motivation – WebDriver

- Created by Simon Stewart in 2006
- Motivation: powerful web applications; more restriction on programs
- Does not depend on JavaScript
- Supports same and more languages than Selenium RC
- Faster test execution
- Communicates directly to the browers
- Only needs IDE (with Selenium commands) and a browser to operate

# History and Motivation – Selenium 2

- Created by the Selenium Team in 2008
- Motivation: provide the best possible framework
- Faster and improved version by merging Selenium RC and WebDriver
- Eliminates Selenium RC's limitations
- Eliminates WebDriver's limitations
- User friendly

# History and Motivation – Selenium 2 continues

WebDriver

- Directly communicates with browser at OS level
- Support HtmlUnit, helps speeds up test execution
- Supports mobile based applications
- Simple API

Selenium RC

- Interacts with broader range of browsers and new browsers
- automatically generate an HTML file of test results by built in command
- Supports testing frameworks like TestNG and Junit
- Matured API

# USING SELENIUM IN AGILE PROCESSES

# AGILE DEVELOPMENT

- Agile methodologies solves challenges that are present in the Waterfall model
- Waterfall model
  - Fail to satisfy customers
  - Schedule overruns
  - Final product often does not meet current needs
- Agile solution
  - Deliver software frequently and regularly
  - Allow requirements to evolve throughout software development

# QA IN MODERN SOFTWARE DEVELOPMENT

- Introduction of Agile transformed QA
- New challenges
  - Changing requirements
  - Large system sizes
  - Frequent delivery
  - Compatibility with different browsers and devices
- Difficulty for testers from manual testing & exploratory backgrounds to keep with the pace of delivery

# QA IN MODERN SOFTWARE DEVELOPMENT (CONTINUED)

- To address new challenges
  - Automation tools
- Selenium
  - Automates testing on client side
  - Dramatic reduction of manual labor
  - Seamless integration with popular development platforms
  - Support for wide range of languages
  - Supports **continuous integration**

# CONTINUOUS INTEGRATION (CI)

- Practice used in Agile development
- Solves challenges of late software integration
- Automated testing is core to  continuous integration
- Selenium's role
  - Automate client side testing
  - Reduces hundreds of manual labor work
- Benefits
  - Prevent new code from breaking existing functionality
  - Quicker feedback
  - Faster delivery of high-quality code

# CONTINUOUS INTEGRATION (CONTINUED)

- Continuous integration process
  - Each commit triggers workflow of test suites
    - Unit testing
    - Integration testing
    - Client-side testing (Selenium)
  - Commits that fail any test is blocked from merging onto the main branch
  - Commits must pass all tests before merging

# CONTINUOUS INTEGRATION (CONTINUED)

- Selenium integrates with other development platforms to achieve continuous integration
- Some popular integrations
  - Git
    - Version control system
  - Jenkins
    - Automation server
    - Automates tasks related to building and deploying a project
  - Maven
    - Build management tool
    - Manage dependencies, tests, project structures
  - TestNG
    - Selenium itself lacks test reporting features
    - TestNG generates test results in report format

# OTHER USES OF SELENIUM IN REDUCING MANUAL LABOR

- Automation-aided exploratory testing to uncover defects
- Bug reproduction scripts speed up defect resolution
- Testing across different browsers and devices

# DRAWBACKS OF SELENIUM

- Learning curve to learn the tool
  - Is an investment before output and quality is increased
- Selenium WebDriver require programmer knowledge
  - Does not allow for codeless testing (Selenium IDE does)
- Selenium lacks reporting features
  - TestNG used to complement

# CONCLUSION

- Selenium fits well in Agile development strategies
  - Dramatically reduces manual labor
  - Increase quality and output
  - Speed up tasks

# Selenium Code With JavaScript

*A Demonstration with JavaScript and the Web Browsing Automata*

# Selenium

Selenium, at its heart, automates web browsing.

It can test multiple web browsers on one machine or many.

The automated framework follows a number of design goals that encourages many different types of programming and fits into different infrastructures.

Selenium supports popular back-end languages in C#, Java, Python, and Ruby.

# Selenium with JavaScript

Selenium also supports JavaScript, and the following coding demonstration will utilize that.

The coding demonstration will rely on a OSX environment.

Knowledge of Document Object Model (DOM) and general syntax for targeting HTML elements is encouraged.

The developer should have NPM (or Yarn) installed, then they can begin with the following command

## Selenium’s Setup

Selenium also supports JavaScript, and the following coding demonstration will utilize that.

The coding demonstration will rely on a OSX environment.

Knowledge of Document Object Model (DOM) and general syntax for targeting HTML elements is encouraged.

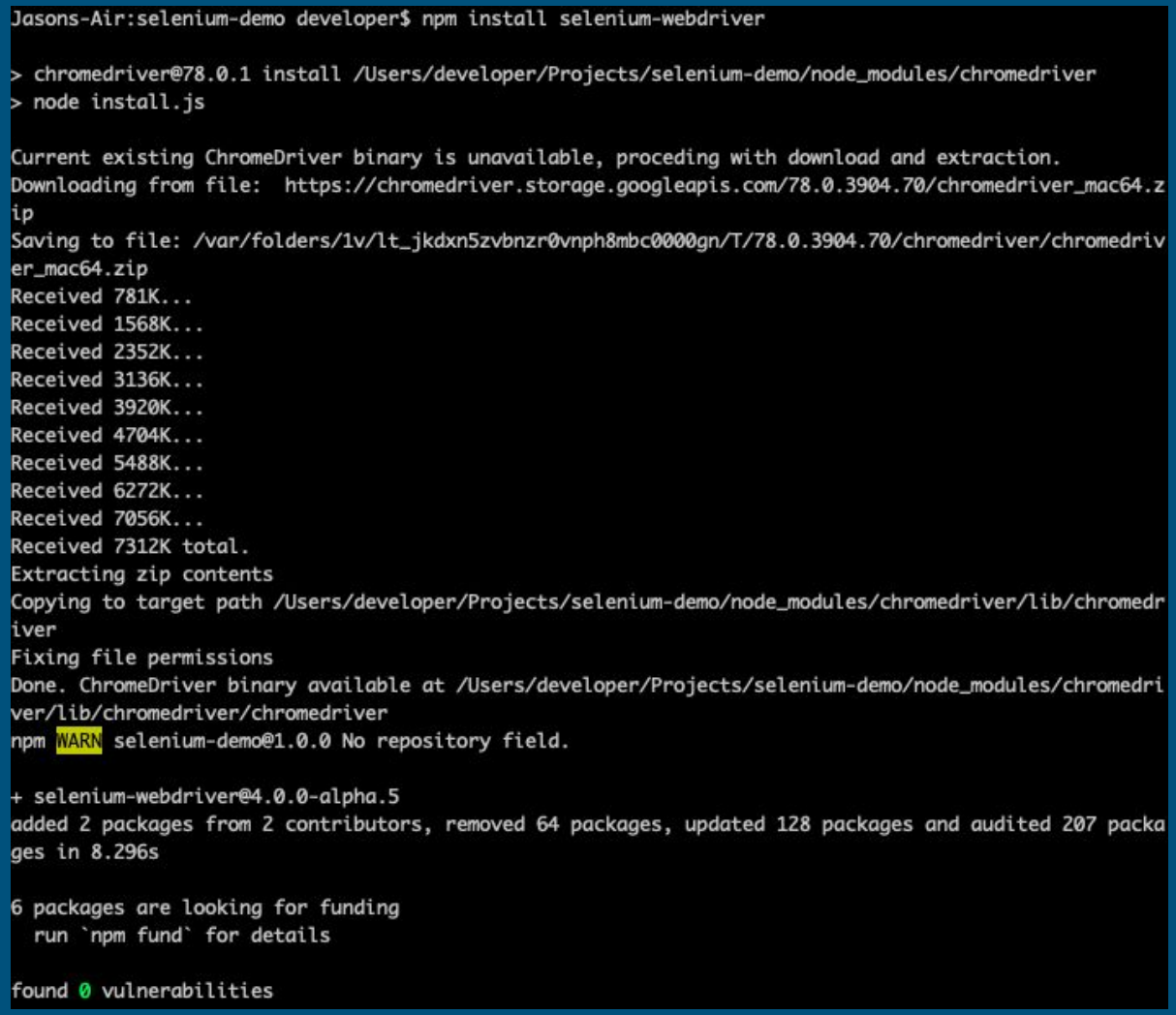
The developer should have NPM (or Yarn) installed.

# Selenium's Setup

Installation begins with the terminal command:

```
$ npm install selenium-webdriver
```

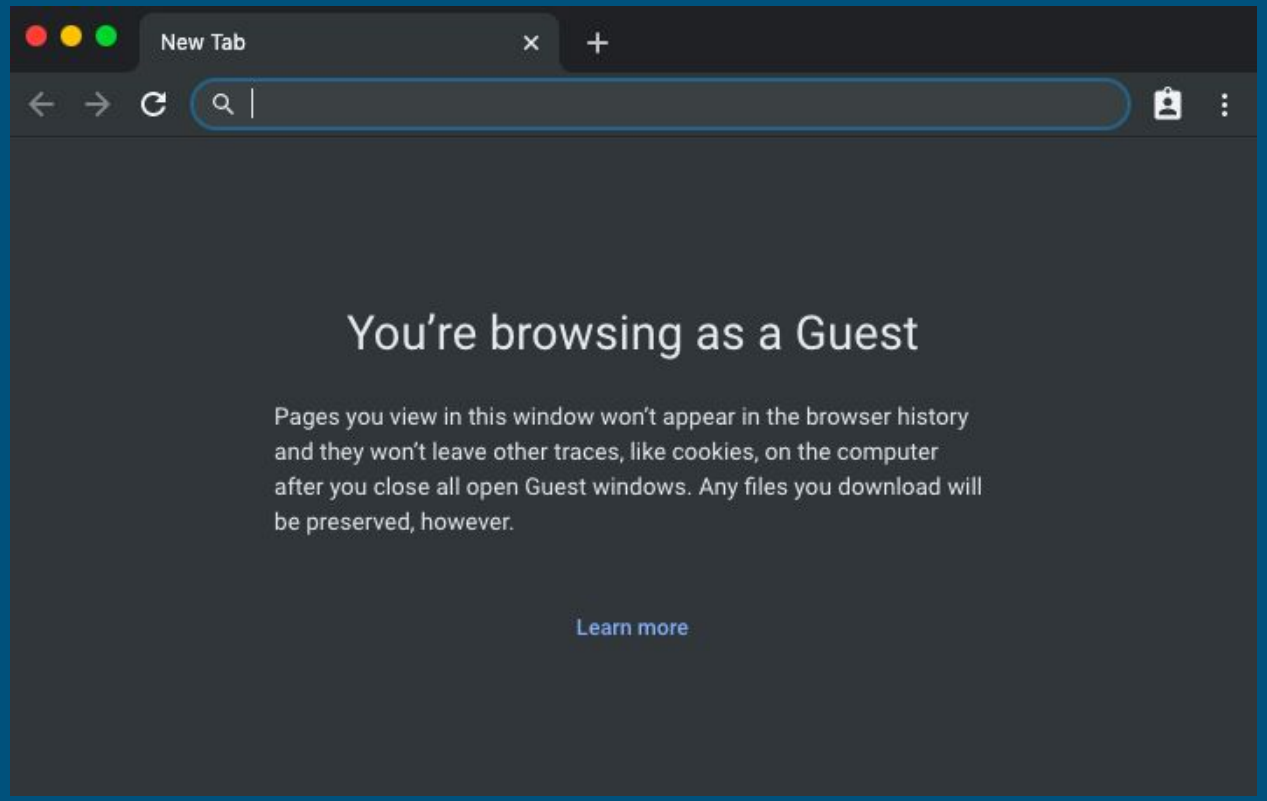
Within the project directory.



# Selenium API

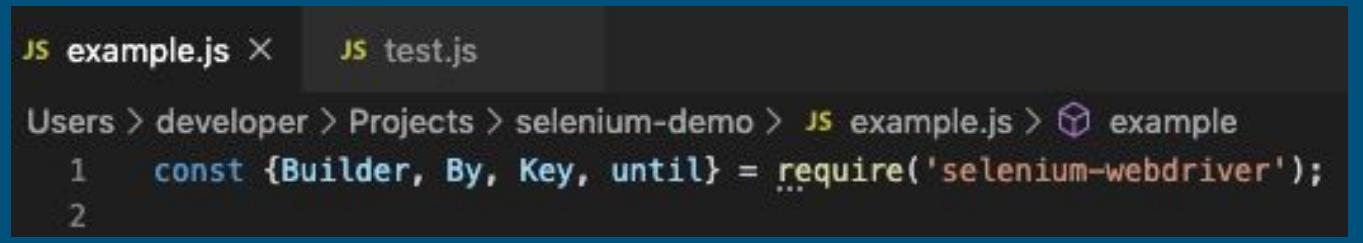
`driver.get(website_url)` is a fundamental feature of Selenium.

This automates visiting given website url in the browser Selenium calls.



## Selenium API

The driver is initialized by importing from `selenium web-driver`.




`By`, `Key`, and `until` are additional helper tools.

`Builder` is what will build the virtual driver.

```
before(async function() {
    driver = await new Builder().forBrowser('chrome').build();
});
```

## Selenium API

With the helper tools, we can target specific elements in the DOM.

```
await driver.get('http://www.google.com/');
await driver.findElement(By.name('q'))
```

Then we can send input as well.

```
await driver.findElement(By.name('q')).sendKeys('webdriver', Key.RETURN);
```

Since it automates web browsing, there is code for waiting for output as well.

```
await driver.wait(until.titleIs('webdriver - Google Search'), 1000);
```

## Selenium API with mocha

Selenium used in conjunction with a testing framework is very powerful.

Mocha is such a testing framework. The code on the right show Selenium within mocha's domain specific language.

```
describe('Search Google on Firefox', () => {
    let driver;

    before(async function() {
        driver = await new Builder().forBrowser('firefox').build();
    });

    it('Searches on Google', async function() {
        // Load the page
        await driver.get('https://google.com');

        // Target Search box, then send input
        await driver.findElement(By.name('q'))
                    .sendKeys('test', Key.RETURN);

        // Wait for the results
        await driver.wait(until.titleContains('Google Search'), 2000);

        // Extract title
        let title = await driver.getTitle();

        //Test
        assert.equal(title, 'test - Google Search');
    });
    // close selenium browser
    after(() => driver && driver.quit());
})
```

## Selenium API with mocha

Building suites of tests with automated testing enables very active development.

User interface testing becomes much easier to develop despite difficult to test visually.

```
describe('Search Google on Firefox', () => {
    let driver;

    before(async function() {
        driver = await new Builder().forBrowser('firefox').build();
    });

    it('Searches on Google', async function() {
        // Load the page
        await driver.get('https://google.com');

        // Target Search box, then send input
        await driver.findElement(By.name('q'))
                    .sendKeys('test', Key.RETURN);

        // Wait for the results
        await driver.wait(until.titleContains('Google Search'), 2000);

        // Extract title
        let title = await driver.getTitle();

        //Test
        assert.equal(title, 'test - Google Search');
    });
    // close selenium browser
    after(() => driver && driver.quit());
})
```

## Selenium API with mocha

These tests work well with Agile development and philosophy.

Selenium can be used for other back-ends such as Rails as well.

Continuous deployment tools also package Selenium as well, which is useful for virtual and online platforms like Docker.

It remains a powerful and dominant fixture in today’s web development.

# Alternatives and Competitors to Selenium

# Intro

- There are a variety of alternatives and competitors to Selenium / Selenium WebDriver used in today's software industry. These alternatives boast specific advantages to Selenium which, although is the predominantly used UI automation framework in the industry, is facing competition due to the emergence of these alternatives.
- Popular testing tools that are favored commonly exhibit the following characteristics:
  - easy learning curve
  - greater test stability
  - better reusability
  - fewer number of dependencies.

# Cypress

- One example of a testing tool that is gaining a lot of traction in the software testing industry is Cypress, which is a Javascript-based web automation framework
- Cypress is the ideal testing tool to utilize in an Agile test-driven development environment since UI tests can be created at the same time features are developed, within the browser
- Therefore, developers can test their work almost immediately after development of the feature is complete.

# Cypress (continued)

- Also, unlike Selenium, Cypress does not require that web elements need to be in an interactable state (i.e. visible, element fully loaded) before performing a test on the element
- The bulk of test failures in Selenium tend to be due to elements not being correctly found because Selenium requires elements to be in a ready state, commonly handled using element waits.
- However, if the element somehow fails to load, the test code will never make it to the point of testing the element, which defeats the purpose of the test
- As a deterministic testing tool, Cypress handles this problem by automatically waiting for the web application to reach a ready state before performing any validations, thereby eliminating test flakiness.

# Splinter

- Splinter is another popular web UI test automation tool that shows plenty of potential to become a top testing tool in the software industry
- Splinter is actually an abstraction layer built on top of Selenium, allowing one to reuse Selenium code in a Splinter test framework.
- One of the major advantages of Splinter is the ability to write tests with considerably less code than the same test in Selenium would require.  Splinter is developed with simplicity in mind by bundling up common automated test commands into a convenience method

# Splinter (continued)

- A good example to demonstrate this point is filling forms.  In Selenium, we would need to first find the element then send keys to it, like so:
  - element = browser.find_element.by_name('email')
  - element.send_keys("test_email@gmail.com")
- While in Splinter, the equivalent action can be performed using a one-liner:
  - browser.fill('email', "test_email@gmail.com")

# Splinter (continued)

- Furthermore, Splinter uses the same web drivers as Selenium, allowing software teams to run their tests on common browsers like Chrome and Firefox especially for compatibility testing.
- The only drawback is that Splinter tests are currently only implementable in Python (just like how Cypress tests can only be written in Javascript), which may pose as an inconvenience for those who are less experienced in this programming language.