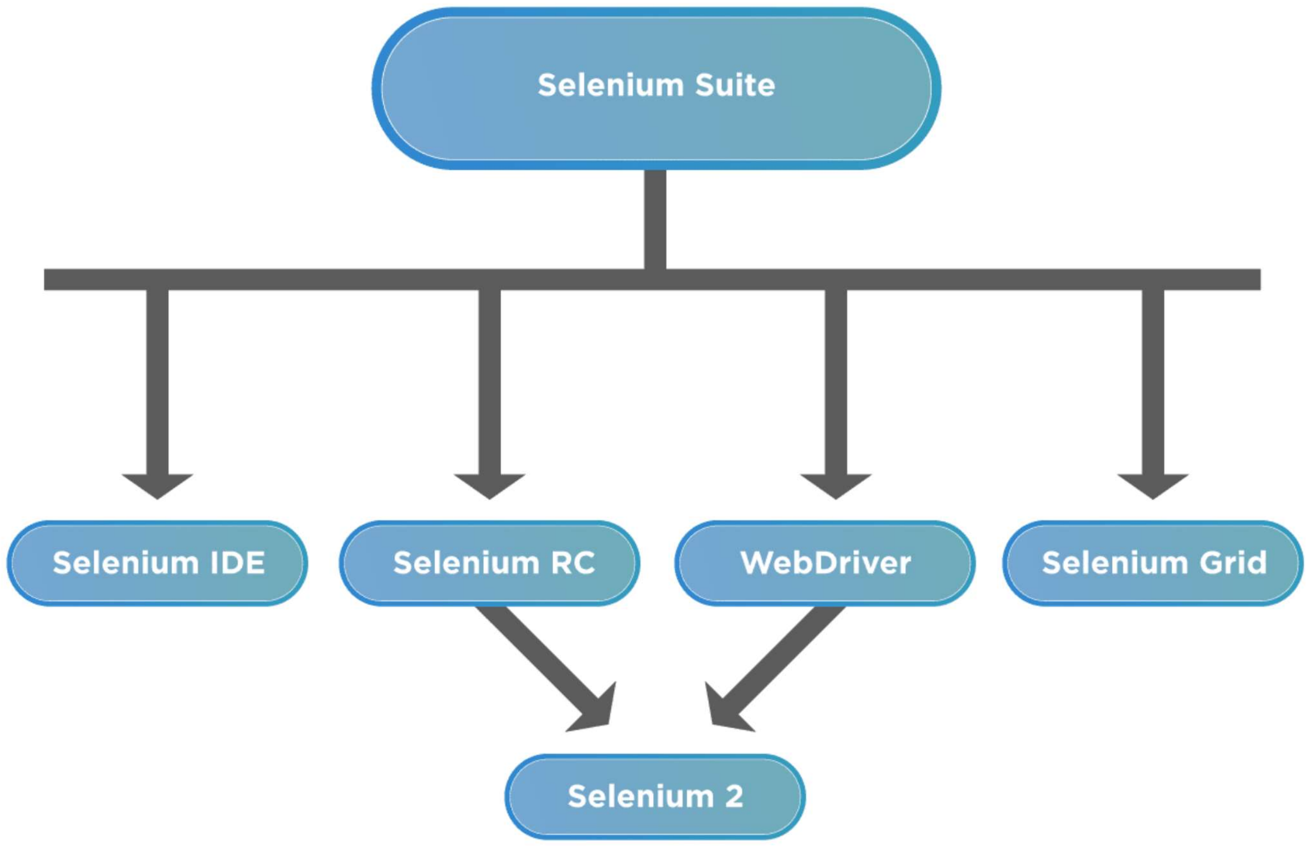
Selenium Webdriver is one of the critical members of this family and is well-known for its diversity and stability for web automation. Selenium *Webdriver* has kind of become a de-facto for ***UI automation*** and more than 80% of the companies are using it. Let's understand the details of this tool by covering the following sections:

What is Selenium WebDriver?

Selenium WebDriver is a set of open-source ***APIs,*** which provided the capabilities to interact with any of the modern web-browsers and then, in-turn to automate the user actions with that browser. It is an essential component of the ***Selenium*** family. As we know, Selenium is not an independent tool; rather, it is a collection of tools that make the Selenium suite, which was created when two projects ***Selenium RC and WebDriver were merged.***



WebDriver was integrated with *Selenium RC* to overcome a few of the limitations of *Selenium RC* and has now become the de-facto for Web automation. You must be aware that after Selenium 2, there are new versions have been released. By now in Jan 2021 they have reached ***Selenium 4.***

Why use Selenium WebDriver for Web Automation?

Now that we know what Selenium *WebDriver* is and what it does let's take a look at why it is the optimum choice to use for web automation.

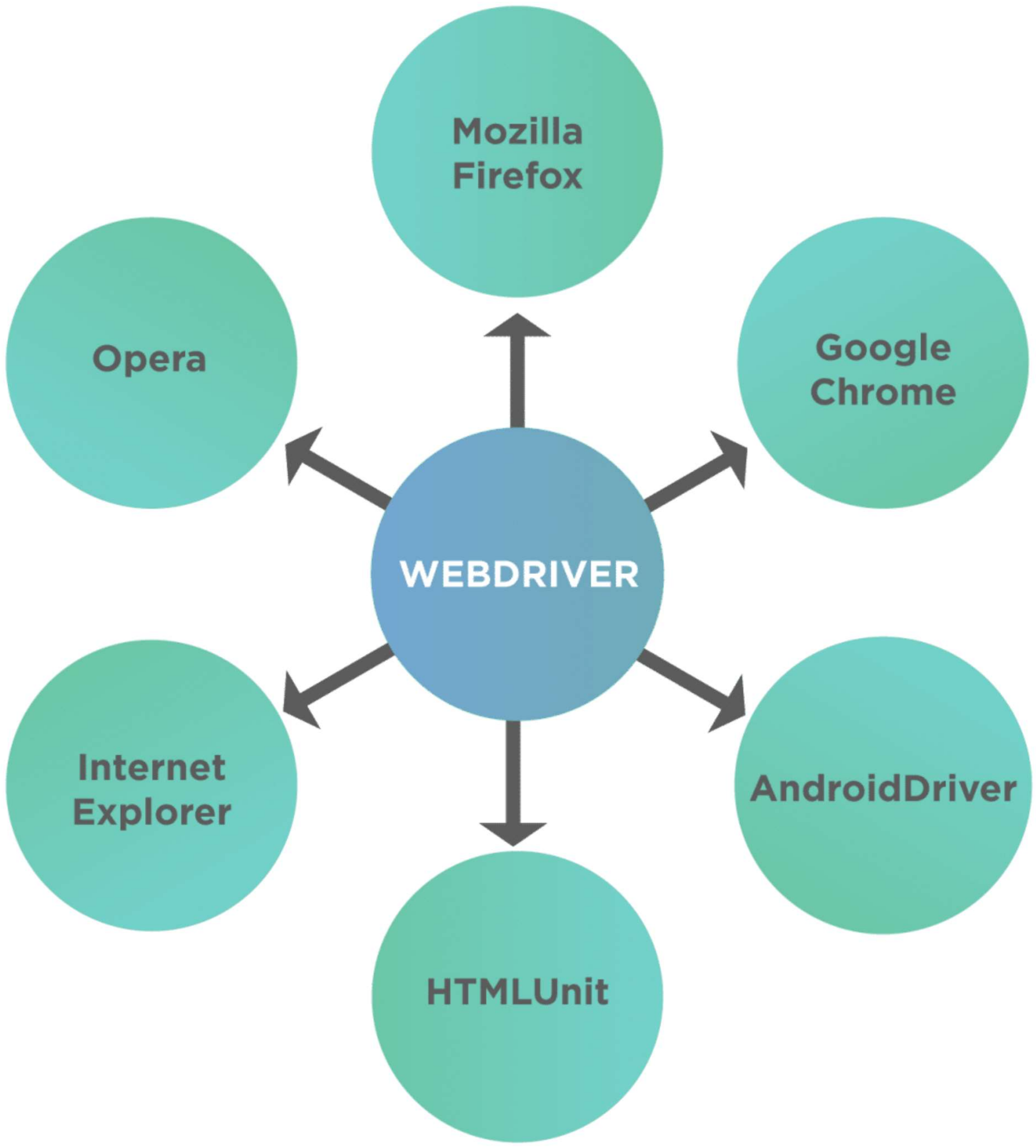
Feature*	***Description***
Dynamic Web Page Automation	Selenium *WebDriver* can automate dynamic websites where the content of pages changes by user actions.
Works Close to Browser	Browser vendors ship their *WebDriver* implementation. Hence they are tightly coupled to the browser giving a better testing experience.
Technology Agnostic	Selenium *WebDriver* allows you to automate the test cases for all the web applications, irrespective of the technology used for the development of the application under test.
Mimics Real User	Selenium *WebDriver* allows QA to mimic user actions on the websites. Selenium WebDriver can mimic typical user actions like form filling, clicking, double-clicking, key-press, as well as advanced user actions like drag and drop, click and hold, etc.
Supports Cross Browser Testing	Selenium *WebDriver* has the most significant advantage when doing cross-browser testing - where a QA can test for the same website, using the same piece of code on different browsers. It enables the verification and validation of test cases on multiple sets of browsers at the same time.
Supports parallel Execution	If there are more scripts to be executed on multiple browsers, then doing them one by one is time-consuming. So Selenium *WebDriver*allows parallel execution, using frameworks like TestNG, so that execution of test cases is faster. This allows large scale execution of test cases in a short time.
View Execution Results	Selenium *WebDriver* allows a QA to view the live execution of the automated test run on the computer system as well as on any other CI/CD pipeline machine by supporting functionalities like the screenshot, video-recording of test cases, etc.
Supports modern development techniques	Selenium *WebDriver* integrates very well with modern Software Development principles like Behaviour Driven Development via integration with the Cucumber library.

Broadly speaking Selenium *WebDriver* is one of the most important parts of the Selenium suites, which supports almost all the features needed for the automation of a web application.

Why Selenium WebDriver is popular?

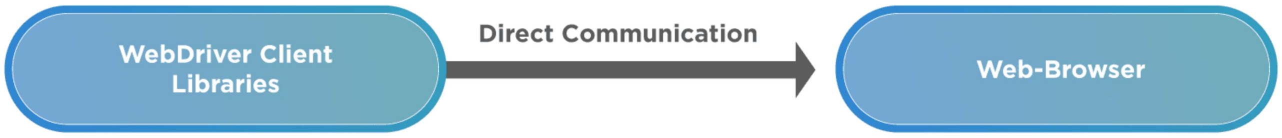
Apart from the above-mentioned capabilities, *WebDriver*, being part of the Selenium family, also encompassed some of the unique characteristics, which adds to its popularity as a web automation tool. A few of those characteristics are:

Multi-Browser Compatibility *- One of the prime reasons for the popularity of Selenium and WebDriver is its cross-browser support using the same piece of code. It gives the ability to run a specific piece of code that mimics a real-world user using a browser's native support to hit direct API calls without the need for any middleware software or device. The below shows a sample list of browsers supported:*



Multi-Language Support *- Not all testers are well versed with a particular language. Since Selenium provides support for many languages so a tester can use any language out of the supported languages and then use WebDriver for automation. This gives the freedom to write code in the language people are comfortable with.*

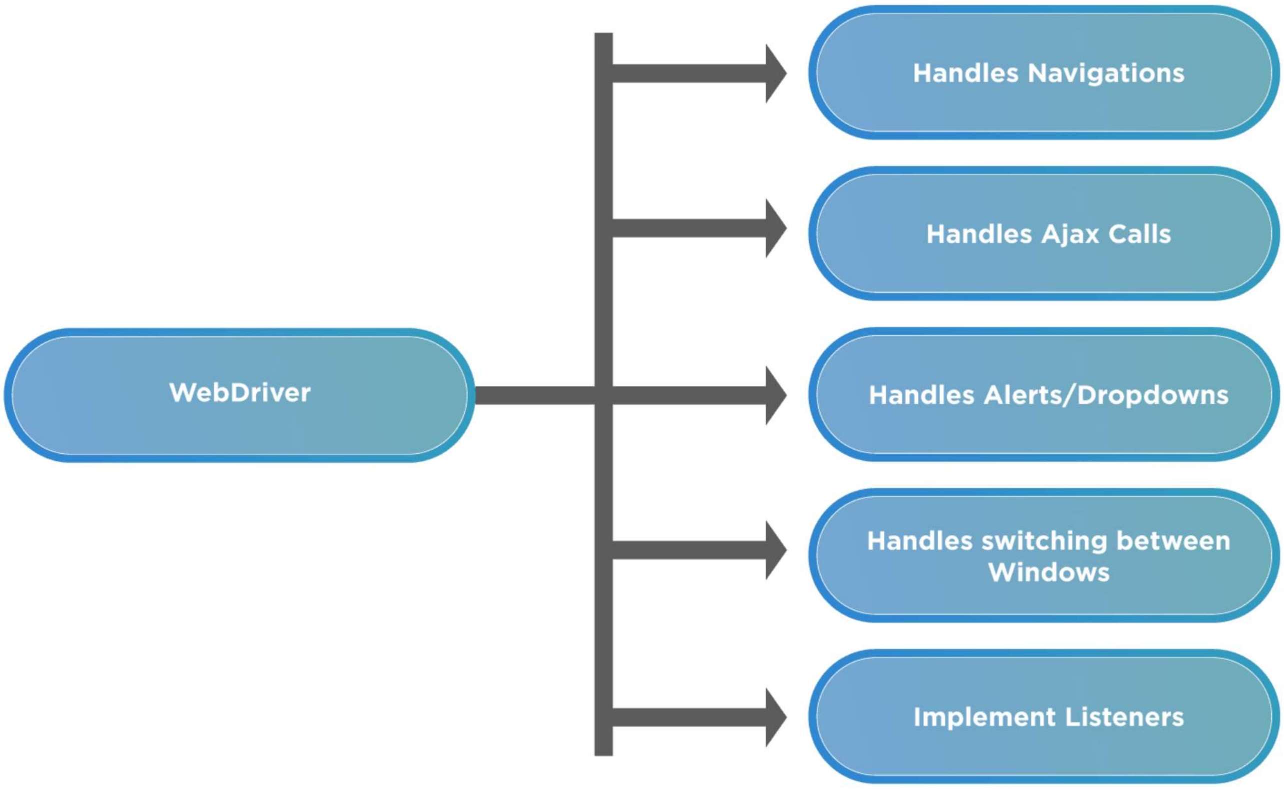
Faster Execution *- Unlike Selenium RC, WebDriver doesn't depend on a middleware server to communicate with the browser. WebDriver directs communications with browsers using a defined protocol (JSON Wire), which enables it to communicate faster than most Selenium tools. Also since JSON Wire itself uses JSON, which is very lightweight, so the amount of data transfer per call is very minimum. The below figure shows clearly how the WebDriver communicates with the Browser:*



Locating Web Elements *- In order to perform actions like Click, Type, Drag, and Drop we first need to identify on which web element (like button, checkbox, drop-down, textarea) we need to perform an action. To facilitate this, WebDriver has provided methods to identify web elements using various HTML attributes - like id, name, class, CSS, tag name, XPath, linktext etc.*
Handling dynamic web elements *- There are times when there are web elements on-page, which change with every reload of the page. Since the HTML attributes change, it becomes a challenge to identify these elements. Selenium provides multiple methods to handle these situations -*

- ***Absolute Xpath*** *- this contains the complete XML path of the element in question.*
- ***Contains( )*** *- using these functional elements can be searched with partial or full text and can be used to handle dynamic elements.*
- ***Starts-With( )*** *- this function is based on finding elements using starting text of the attribute under question.*

Handling Waiting for Elements *- Not all the pages have the same structure. Some are lightweight, while some have a considerable amount of data handling or AJAX calls. Many times the web elements take some time to load. To account for this WebDriver has provided multiple waiting mechanisms that can be used to pause the script execution for a required amount of time based on certain conditions and then continue once the condition is full-filled. The following figure shows a sample list that shows the capabilities of WebDriver which helps in handling the dynamic behavior of web pages.*



What are the drawbacks of Selenium WebDriver?

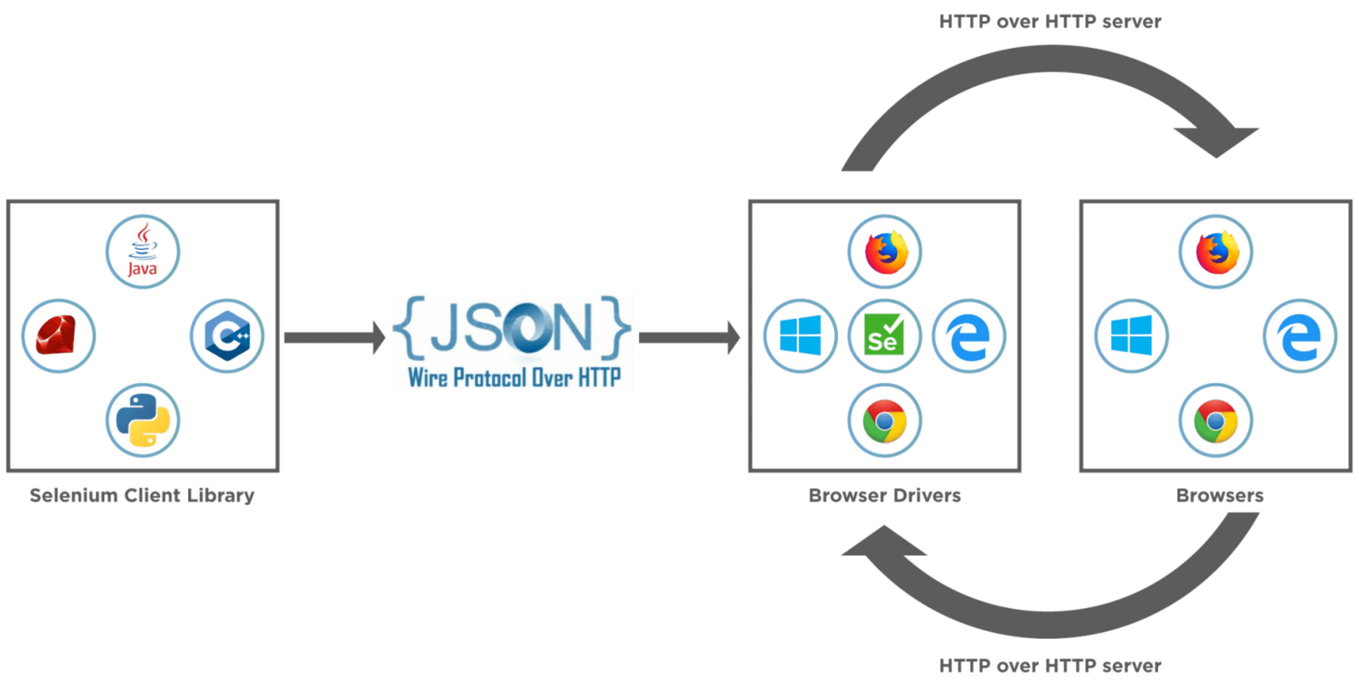
Although Selenium works a long way out in solving the *UI *and functional automation of web applications, it is not without its drawbacks. Let's look at some of the shortcomings/drawbacks:

- ***Requires Programming Knowledge and Expertise*** *- Since WebDriver allows you to automate the user actions using code written in a certain programming language, anyone who wants to use it should have a basic understanding of how coding in the language works. People who do not have an understanding of coding in a programming language will find it hard to use WebDriver.*
- ***No Support for Desktop Applications.*** *- Selenium ecosystem, including WebDriver, was built for the automation of web-applications. As such if you are looking to automate windows based applications, you will not be able to do so.*
- ***No Customer Support*** *- Selenium ecosystem, including WebDriver is completely open-source, which means it is driven by individuals and not by any company. Because of this, there is not a dedicated support team to look into your issues. If a person is stuck somewhere, there are many communities, forums which a person can rely on, but that's about it.*
- ***No Built In Object Repository*** *- Paid tools like UFT/QTP provide a centralized location to store objects/elements, called the Object Repository. This ability is not provided by default in Selenium WebDriver. This can be overcome using approaches like the Page Object Model, but it requires considerable coding skills and expertise.*
- ***Lack of built-in reporting*** *- Selenium WebDriver can help you run your automation tests but to provide a reporting capability, you would need to integrate it with a testing framework like Junit, TestNG, PyTest, Allure, etc.*
- ***Managing Browser-Selenium Dependencies*** *- Since Selenium has to rely on compatibility between the browser drivers and the actual browser itself, at many times due to incompatibility or bugs in either the browser driver or browser, functionality breaks, and users have to rely on community support to get it fixed.*

Understanding of Selenium WebDriver Architecture

Being a part of the overall component system, we deduce that the ***Selenium WebDriver*** is not a standalone testing tool. It comprises various components that are required to run tests. These are the architectural components of Selenium.

So first let's take a look at this image below

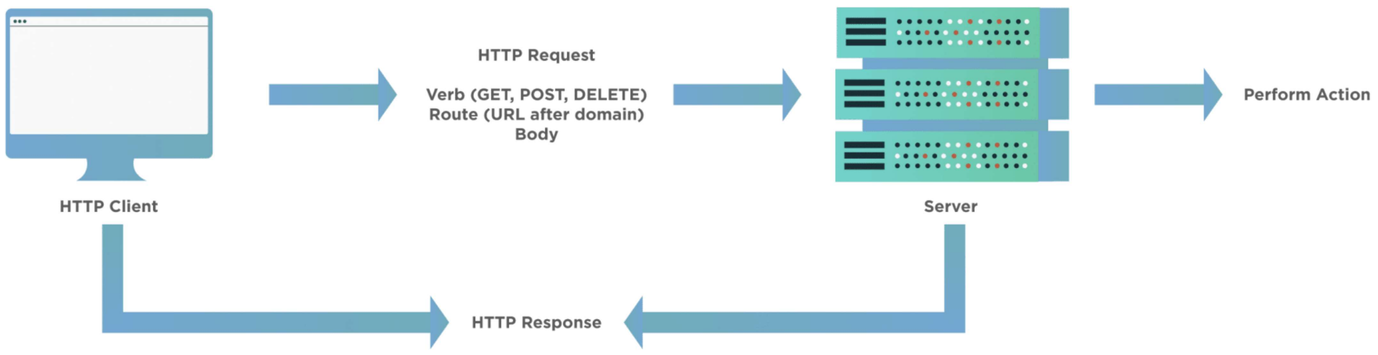


This image tells us about the ***core selenium webdriver architecture and the major selenium components*** which comprise *WebDriver.*

Selenium WebDriver Client Libraries / Language Bindings *– Software Testers want to select languages that they are comfortable with. Since WebDriver Architecture supports different languages, so there are bindings available for a range of languages like* ***Java,*** *C#,* ***Python,*** *Ruby, PHP, etc. Anyone who has a basic knowledge of working with any programming language can get specific language bindings and can start off. This is how Selenium Achitecture provides flexibility to testers to do automation in their comfort zone.*
JSON WIRE PROTOCOL *- As per the Selenium Architecture above, the JSON Wire Protocol facilitates all the communication that is happening in Selenium between the browser and the code. This is the heart of Selenium. JSON Wire Protocol provides a medium for data transfer using a RESTful (Representational State Transfer) API which provides a transport mechanism and defines a RESTful web service using JSON over HTTP.*
Browser Drivers *– Since there are various browsers that are supported by Selenium, each browser has its own implementation of the W3C standard that Selenium provides. As such browser-specific binaries are available that are specific to the browser and hides the implementation logic from the end-user. JSONWire protocol establishes a connection between the browser binaries and the client libraries.*
Browsers *– Selenium will be only able to run tests on the browsers if they are locally installed, either on the local machine or on the server machines. So browser installation is necessary.*

How Selenium WebDriver works?

In the section above, we saw the architecture of Selenium. Now let's see how behind the scenes all the communication happens? Take a look at the image below – this shows a view of how the actual workflow looks like.



When a user writes a ***WebDriver code in Selenium*** and executes it, the following actions happen in the background –

An HTTP request generates, and it goes to the respective browser driver (Chrome, IE, Firefox). There is an individual request for each Selenium command.
The browser driver receives the request through an HTTP server.
The HTTP server decides which actions/instructions need to execute on the browser.
The browser executes the instructions/steps as decided above.
The HTTP server then receives the execution status and then sends back the status to an automation script, which then shows the result (as passed or an exception or error).

How to use Selenium WebDriver for Web Automation?

Selenium WebDriver provides a very seamless, user-friendly, and code friendly approach to automation using various browsers. Since it supports most of the major browser vendors, it's just a matter of using the respective browser driver and browser and setting up Selenium to use the same.

For any Selenium test script, there are generally the following 7 steps, which apply to all the test cases and all the applications under test (*AUT*):

. ***Create an instance of WebDriver specific to the Browser:***

Eg: To create an instance of the Firefox driver, we can use the following commands:

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
WebDriver driver = new FirefoxDriver();
```

. ***Navigate to the desired Web page which needs to be automated:***

Eg: To navigate to the ***"https://demoqa.com/text-box",*** *we can use the following command:*

```
driver.get("https://demoqa.com/text-box")
```

. ***Locate an HTML element on the Web page:***

In order to interact with a web page, we need to locate the HTML elements on the web page. We can use any of the element locator strategies mentioned at ***"Selenium Locators"***. Eg: if we want to get the "***Full Name***" text box, we can use the following commands:

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
WebElement usernameElement = driver.findElement(By.id("userName"));
```

. ***Perform an action on an HTML element:***

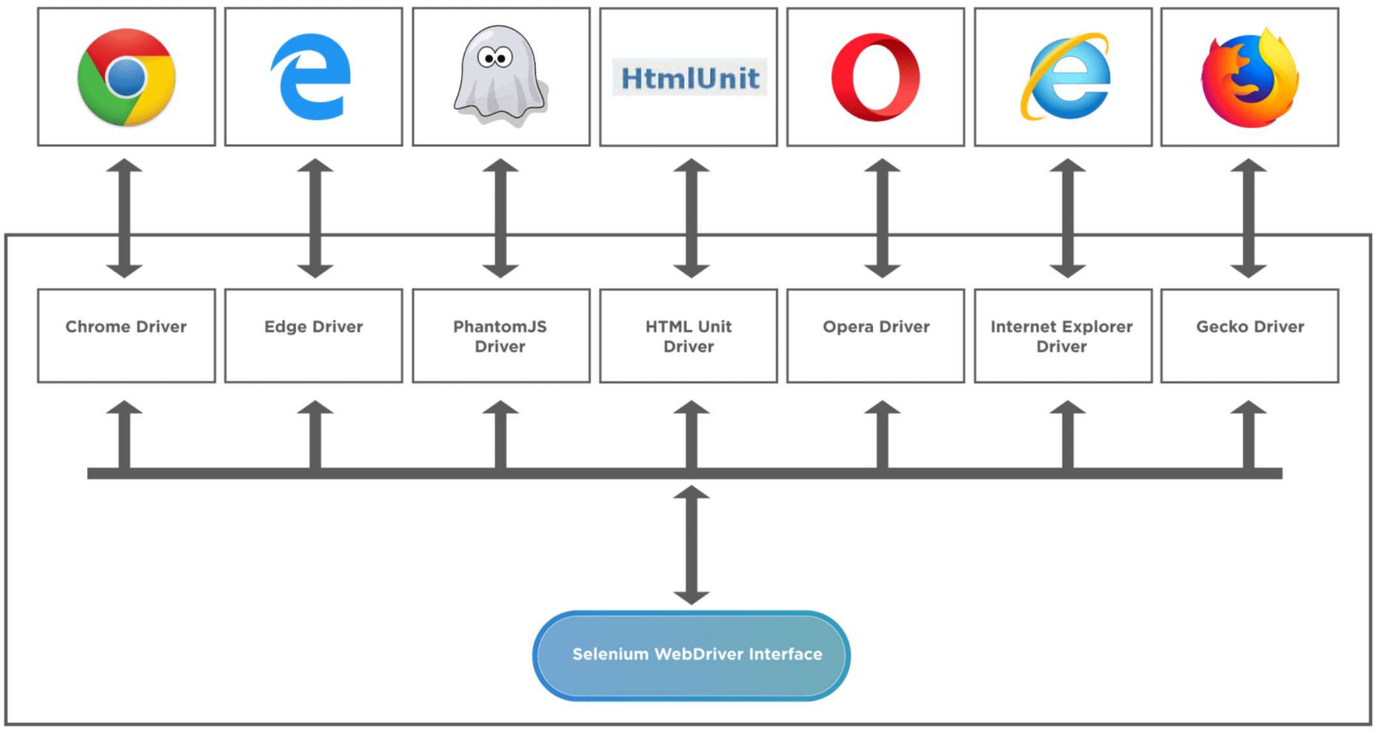
We can perform certain actions on the HTML elements, such as type something using the ***SendKeys*** *method, click on the element if it is a button. Eg: if we want to type the name in the identified text box, we can use the following commands:*

```
usernameElement.sendKeys("Ravinder Singh");
```

. ***Run tests and record test results using a test framework.***

And, we are done with using the *WebDriver* to identify and perform the needed actions on the Web Application. Depending on the browser, on which we need to test our application, we can use the corresponding *WebDriver.*

Here is a list of various browsers and their respective browser drivers:



Recently Microsoft moved their Edge browser on the same platform as Chromium (*which is the parent for Chrome*), and due to this *ChromeDriver* can now also support *Microsoft Edge Chromium.*