Automated testing framework with browserstack integration

Denislav Lefterov¹, Svetoslav Enkov²
Plovdiv University “Paisii Hilendarski”, Bulgaria
Plovdiv University “Paisii Hilendarski”, Bulgaria
denislav.lefterov@uni-plovdiv.bg, enkov@uni-plovdiv.bg

Abstract: Nowadays ensuring high-quality software product requires a lot of testing efforts. Automated tests in the 21st century are a must. Whether it’s code peer-review, unit, integration, system or exploratory user testing – it all has to be done with given insurance and expertise! Therefore, we look to automate our testing where it is conceivable. Most applications today have the equivalent of Web and Mobile versions for the same functionality, but different platforms of action in order to facilitate the users. Therefore, parallel testing of both subtypes is required to provide smooth maintenance and rapidity. Running as many builds as possible, including tests to ensure quality is essential when comes up to reliable software product and agile development. With the implementation of Continuous Integration and Continuous Delivery tools, we guarantee a better way to deploy automated tests across multiple instances and execute them against those environments. BrowserStack is one common cloud solution to these requirements. It is a very powerful tool, which can be attached to our development process. This article explores detailed approach on the automation of cross-browser, device and compatibility testing in BrowserStack platform, using a custom extended automation framework to provide direct configuration output to this environment and give ease in the future Web and Mobile development.

Keywords: AUTOMATION, FRAMEWORK, TESTING, QUALITY, ASSURANCE, BROWSERSTACK

1. Introduction

There are many cloud-based applications available today that help with software development, regardless of the area we want to use or improve. It is especially important to keep in mind that when it comes to software quality assurance it is almost mandatory to take most of the automated tests on cloud infrastructure, such as: Amazon Web Service (AWS), BrowserStack, SourceLabs and etc. All of these platforms have similar functionalities and are easy to use.

The problem area of the current scientific work is focused on that in order to be able to take advantage of all the available features and functionalities, we need to have a well-defined and extended automated testing framework that programmatically interacts and communicates with the above-mentioned cloud-based systems, because they all provide own rules of operation and configuration, resulting in obstacles and delays in the transformation from on-premises development to cloud-based. In this study, we will track the extension of an automated software framework geared toward connecting to a cloud-based cross-browser testing platform and utilizing all the benefits of this approach.

2. BrowserStack introduction

BrowserStack is one of the world’s most trusted platforms for cross-browser testing. It allows interactively testing and debugging websites across thousands of browsers and real mobile device browsers to ensure that applications are working flawlessly. In real-time, it is difficult to test an application in different browsers using multiple operating systems, with varying versions.

It will cost a lot to buy many computer machines to install different operating systems. It is even more difficult to buy all the Android and iOS mobile devices that are available in the market today. Due to this reason, it is not practical to buy and set up all the devices. Thus, we go with the BrowserStack to avoid the complexity of switching between the operating systems, browsers, and different versions.

When it comes to mobile application testing, we can avoid buying all the mobile devices that are available today. Browser Stack is very flexible and scalable. We can test anywhere and anytime with the help of Browser Stack.

We can use Browser Stack as a remote lab and can use this as Real Desktop Browsers. It gives us free Javascript unit tests and supports 750 configurations. There is no set up required for using BrowserStack. We can use it directly on any independent machine by using the Browser Stack URL and its login credentials. [1]

Features of BrowserStack

• Cross-browser testing with different browsers on different operating systems;
• Native app testing on mobile;
• Hybrid application testing;
• Automate web and mobile applications.

3. Automated testing framework

The benefit of creating an automation framework is to use it as a template for every new project, offering us the leverage to avoid all of the known problems. When automated tests are created, the first thing we do is to interact with the browser. This can include navigating to a page, clicking a button, or filling in a login form and many different actions. After that, we need to verify and report the actual versus the expected results. While we have many different tests at our disposal, how and when we use them is dependent on the scenario. In some cases, we will execute several tests in a specified order. In others, only execute specific tests. In order to achieve all of this, testers usually need to implement different frameworks or libraries along with the Selenium Web driver. As we said most projects have common user actions that need to be accomplished in an automated flow. These interactions are developed and implemented in the framework itself and the testers can use them right away without wasting time to write or re-write them again from the start. [2]

With the advantages of the framework, there is no need to waste time developing this functionality since it is already done generally. Along with all positives, this development is integrated and extended from Selenium and Appium. This gives us the flexibility to easily choose which tests we want to execute: whole test suite / scenario, or only smoke, sanity tests and etc. also other perspectives or a specific set of tests. It supports data-driven behavior testing and flexible configuration setup.

Another benefit of creating an automated software framework is the ability to prepare ready-made configured classes and packages for the use of many external sources such as: cloud-based tools, external media, servers or resources. The use of the ready-made techniques in the automated framework saves us time, prevents additional effort and facilitates understanding of the business logic.

In Figure 1 we can see a detailed graphical representation and collaboration of all tools and libraries of the software testing framework architecture and the benefits of using BrowserStack cloud platform.
As we said BrowserStack is a wide open platform, it is distributed under the following architecture: SaaS (Software as a service), which means that this is a cloud based solution in a set of functionalities under certain rules. In order to facilitate the usage, we will go through these requirements and take advantage of all its benefits, we should also strict to those rules and set of desired capabilities, which BrowserStack expects from us.

The architecture is based on ‘Hubs’ and ‘Nodes’. The Hub is the central point that will receive all the requests along with information on which browser, platform (i.e Windows or Linux) and which device the test should be run on. Based on the request received, it will distribute them to the registered nodes. Nodes are where the corresponding tests will run. Each node is a machine (physical/virtual machine) or a real mobile device that is registered with the hub. When we register a node, the hub has the information of the node and it will display the browser and configuration details of the nodes.

The prerequisites required to set up BrowserStack are the Capabilities object and Remote WebDriver. The capabilities object would help to configure the desired properties and platform for the tests, and Remote WebDriver is used to hit the BrowserStack API.

Based on the preferences set in the desired capabilities instance, the Hub will point the tests to a node that matches the preferences.

In the following chapter, we will review how BrowserStack API works.

4. BrowserStack Architecture

Here is a sample code snippet in Java that sets the capability to point the required node to the respective hub:

```java
final String USERNAME = "";
final String AUTOMATE_KEY = "";
final String URL = "https://" + USERNAME + ":" + AUTOMATE_KEY + "@hub-cloud.BrowserStack.com/wd/hub";

try {
    DesiredCapabilities caps = new DesiredCapabilities();
caps.setCapability("browser",browser);
caps.setCapability("browser_version",browser_version);
caps.setCapability("os",os);
caps.setCapability("os_version",os_version);
caps.setCapability("resolution",resolution);
caps.setCapability("project","Project-1");
caps.setCapability("build","1.0");
caps.setCapability("BrowserStack.debug","true");
caps.setCapability("BrowserStack.cls","true");
driver = new RemoteWebDriver(new URL(URL),caps);
} catch (MalformedURLException e) {
    e.getMessage();
}
```

Now let’s have a look at all the methods available in the DesiredCapabilities Class.

1. `getCapability()`
   - This method helps in retrieving the capabilities of the current system on which the tests are being performed.
   ```java
generic java.lang.Object getCapability(java.lang.String capabilityName)
```

2. `setCapability()`
   - The setCapability() method is used to declare the properties of test environments like device name, operating system name, operating system versions, browser, and browser versions.
   ```java
   public void setCapability(java.lang.String capabilityName,java.lang.String value)
   public void setCapability(java.lang.String capabilityName,Platform value)
   public void setCapability(java.lang.String capabilityName,boolean value)
   public void setCapability(java.lang.String capabilityName,java.lang.Object value)
   ```

3. `getBrowserName()`
   - This method helps in retrieving the browser name of the current system.
   ```java
generic java.lang.String getBrowserName()
```

4. `setBrowserName()`
   - This method is used to set the name of the browser on which tests are to be executed.
   ```java
   public void setBrowserName(java.lang.String browserName)
   ```

5. `getVersion()`
   - This method helps in retrieving the version of the browser or the operating system of the current system used for running the tests.
   ```java
generic java.lang.String getVersion()
```

6. `setVersion()`
   - This method helps in defining the version of the browser or the operating system for running the tests.
   ```java
   public void setVersion(java.lang.String version)
   ```

7. `getPlatform()`
   - This method helps in retrieving the details of the operating system.
   ```java
generic Platform getPlatform()
```

8. `setPlatform()`
   - This method helps in defining the desired operating system to be used.
   ```java
   public void setPlatform(Platform platform)
   ```

Methods in DesiredCapabilities for Selenium configuration:

Now we can run the desired test automation suite on BrowserStack. Each test run has a unique session ID associated with it. Based on the session ID, all the details required for test execution will be fetched. Each test execution has bug logs generated, for example: a text log that gives a textual representation of each process running in the background. There is also a visual log that shows screenshots of the test being executed. [3]

Infrastructure model for interaction from our local machine and BrowserStack cloud platform:

1. BrowserStack Local makes a REST call using the user’s access key to BrowserStack.com.
2. BrowserStack.com chooses a repeater to establish a secure connection for Local Testing.

The repeater exists within the BrowserStack cloud infrastructure.
3. BrowserStack.com supplies BrowserStack Local with the information necessary to establish a connection with the repeater.

4. BrowserStack Local initiates a connection to the repeater on port 443, using our custom SSL-encrypted protocol.

Note: The repeater cannot directly initiate a connection to BrowserStack Local.

5. A secure, bi-directional, and persistent connection is established between the end user machine and the repeater. We use Secure WebSockets as part of our communication framework. If your enterprise firewall blocks the WebSocket protocol, we fall back to a legacy protocol which is also SSL encrypted, but much slower than WebSockets. For best results, we recommend that outgoing WebSocket connections be allowed in your firewall.

Note: The secure connection is only established up to the user’s machine.

5. Extended Framework Architecture

As we said, in the beginning, our case study is focused on expanding the core software test framework, in this chapter we will follow the overall development of the framework targeting Mobile and Web configurations used and envisioned in BrowserStack.

To be able to use and connect our tests to BrowserStack platform and run it as well, we should extend our automation framework to meet the given rules from BrowserStack architecture. We extended our custom automation framework based on Selenium WebDriver and TestNG, written in the programming language Java.

We have two types of tests in our testing framework, Web and Mobile tests. Web tests are concentrated to run on different browsers, versions, resolutions, projects and builds. Mobile tests, on the other hand, are supposed to run on different devices and operating systems (OS), including Android and iOS. For this purpose, we created the following packages: Device Manager to handle device metrics, prerequisites and configurations, Mobile Capabilities to get the given mobile resources and Web Capabilities.

We can manipulate the whole device settings and configurations and also parse it through the testing framework for further explorations.

When objects are locally set up we can proceed to transfer them via BrowserStack cloud base platform to handle the given capabilities, for Mobile and Web testing approach.

For Mobile testing purpose, we created the following file: MobileCapability.java which is the main class to get the options from iOS and Android Capabilities instances, such as OS type and given versions. We also have BrowserStackMobileCapability which will transfer objects directly to BrowserStack Cloud platform.

We can manipulate the whole device settings and configurations and also parse it through the testing framework for further explorations.

When objects are locally set up we can proceed to transfer them via BrowserStack cloud base platform to handle the given capabilities, for Mobile and Web testing approach.

For Mobile testing purpose, we created the following file: MobileCapability.java which is the main class to get the options from iOS and Android Capabilities instances, such as OS type and given versions. We also have BrowserStackMobileCapability which will transfer objects directly to BrowserStack Cloud platform.
super(capabilities);
this.capabilityName = capabilityName;
}

@Override
public DesiredCapabilities get() {
new CapabilityPreferences(this.capability, this.readCapabilities());
return this.capability;
}

Figure 4. Mobile Capability Class Diagram

For Web testing, we created WebCapability.java to inherit different capabilities: depending on browsers, versions, resolutions, projects and builds.

Figure 5. BrowserStack Web Capability Class Diagram

From Web perspective, we are allowed to use all of the BrowserStack functionalities for different browser testing approach and versions, such as: Google Chrome, Mozilla Firefox, Internet Explorer, MS Edge and Safari. In the given class diagram there is a provided solution on every capability to be parsed from the automation testing framework through BrowserStack platform. We also can manage to choose from a list of supported browser versions.

6. Conclusion

In this paper, we have proposed creation of an automated testing framework with BrowserStack integration. Using this approach provide several benefits such as code re-usage, higher portability, easy maintainability, reduced script maintenance, low cost and “ready-go” configuration to use it directly with BrowserStack cloud platform. Automating most of the actions according to the guidelines reduces manual operations. Since maximum coverage is already in-built and achieved at the initial stage, there is very little or no intervention required by individuals to run the automation tests. Parameters can be parsed automatically from the local machine through BrowserStack, everything is set, only the subscription key is needed to activate the whole process between our development and BrowserStack platform. This article explores a detailed approach on the automation of cross-browser, device and compatibility testing in BrowserStack platform, using a custom extended automation framework to provide direct configuration output to this environment and give ease in the future Web and Mobile development.

7. References


For contacts:
UNIVERSITY OF PLOVDIV PAISII HILENDARSKI, 24 TZAR ASEN, 4000 PLOVDIV
Denislav Lefterov, PhD student, Faculty of Mathematics and Informatics
E-mail: denislav.lefterov@uni-plovdiv.bg