**Final Assignment**

**Q1. What is the test automation framework? What is selenium?  how does it work? and why do you need it?**

Answer:-

**Test Automation Framework:-**

A test automation framework is a set of components that facilitate executing tests and comprehensive reporting of test results. The major components that implement a test automation framework successfully are equipment, testing tools, scripts, procedures, and most importantly, test automation engineers.

* **Components of Test Automation Frameworks**

### Test Data Management

Harnessing data and extracting useful information is the biggest hassle during software testing automation. The availability of data to carry out tests is usually a major problem. To ensure the success of automation efforts, it’s necessary to have a strategic [test data management](https://www.testim.io/blog/test-data-is-critical-how-to-best-generate-manage-and-use-it/) approach. Thus, a software company should equip their framework with resources like libraries or plugins for scavenging through test data and finding what can be used for positive testing. Your framework should also have a simulation tool to make the data more digestible and lucid. If the data is simplified, test data management becomes a lot easier.

### Testing Libraries

The core of an application’s testing process comprises of managing and running the test cases. It’s ideal to get your test cases well-defined and organized so you can perform testing efficiently and effectively. A testing library is where you create and store the test cases. Testing libraries include unit testing, integration, and end-to-end testing, and behavior-driven development. Let’s see what each of them means.

### Unit Testing

Unit testing libraries are a must-have for shaping up a vital part of any test automation framework. This is something which is done not only by the testers but the developers as well. Testers use them to define test methods through specified formal annotations. Unit testing is also used for running simplified and straightforward tests. Unit testing libraries support most of the programming languages. For instance, if you’re a Java developer you probably use something like JUnit or TestNG. On the other hand, C# developers are likely to use NUnit or xUnit.NET. When it comes to [JavaScript unit test frameworks](https://www.testim.io/blog/best-unit-testing-framework-for-javascript/), you have many options at your disposal, including like [QUnit](https://www.testim.io/blog/qunit-tutorial-start-unit-testing-and-measuring-code-coverage/), Mocha, Jest, Ava, [Jasmine](https://www.testim.io/blog/jasmine-testing-get-started-quickly-and-easily/), to name just a few. If you’re a developer, it’s a [good practice](https://www.testim.io/blog/unit-testing-best-practices/) to unit test your code as soon as you develop each module. This reduces the defect count during the later phases of testing.

### Integration Testing

So unit testing is where you test each module or functionality of an application. In unit testing, you must ensure each unit of the application is completely isolated. That means that, during unit testing, units can’t talk to one another. Also, they can’t interact with any dependency that lives outside the code, such as the database or the filesystem. When it comes to JavaScript apps, external dependencies are typically HTTP services, or [APIs](https://www.testim.io/blog/unit-test-rest-api/). However, in the real world, units do interact with each other and with external dependencies. That’s why unit tests aren’t enough. Testing units in isolation is valuable and necessary, but you also need to test the integrations—both between units and between them and external dependencies—if you want to ensure your application works as intended. That’s where integration testing comes in handy. Bear in mind that, by and large, testing frameworks for integration testing are the same you’d use for unit testing—for example, JUnit for Java and NUnit for .NET. The difference lies in the way you use these frameworks. In other words: the tools are the same; [the difference between unit tests and integration tests](https://www.testim.io/blog/unit-test-vs-integration-test/) lie in the way tests are carried out.

### Behavior-Driven Development

As important as they are to your testing library, unit and end-to-end testing have a problem. They rely a lot on the implementation of the functionality which is tested. So if you change the code, you’ll need to change the test case.

How do we address this issue? Behavior-driven development (BDD) is key. Don’t get confused by the name. BDD is not related to development. It’s a collection of best practices. When those practices are applied to automation testing, BDD enables you to write great test cases.

It’s written in an English-like language that’s understandable for the team. You can convert scenarios and features of expected behavior into code. BDD enables the alignment of code with the intent and scope of automated tests.

**Selenium**

Selenium is an open-source tool that automates web browsers. It provides a single interface that lets you write test scripts in programming languages like Ruby, Java, NodeJS, PHP, Perl, Python, and C#, among others. A browser-driver then executes these scripts on a browser-instance on your device.

Here’s how those components work:

### Selenium WebDriver

Also known as Selenium 2.0, WebDriver executes test scripts through browser-specific drivers. It consists of:

* **API**

Application Programming Interface. Ports test scripts you write in Ruby, Java, Python, or C# to Selenese (Selenium’s own scripting language), through bindings.

* **Library**

Houses the API and language-specific bindings. Although plenty of third-party bindings exist to support different programming languages, the core client-side bindings supported by the main project are: Selenium Java (as selenium jar files), Selenium Ruby, Selenium dot net (or Selenium C#, available as .dll files), Selenium Python, and Selenium JavaScript (Node).

* **Driver**

Executable module that opens up a browser instance and runs the test script. Browser-specific—for instance, Google develops and maintains Chrome driver for Selenium to support automation on Chromium/Chrome.

* **Framework**

Support libraries for integration with natural or programming language test frameworks, like Selenium with Cucumber or Selenium with Test NG.

* **How it Works:**

The Web Driver protocol has a local end (‘client’) which sends the commands (test script) to a browser-specific driver. The driver executes these commands on its browser-instance. So, if the test script calls for execution on Chrome and Firefox, the Chrome Driver will execute the test on Chrome; the Gecko Driver will do the same on Firefox.

**Note:** Test scripts execute only when the Web Driver’s client and browser/driver are connected. They don’t have to be on the same device. To enable test execution on multiple remote drivers, you need Remote Web Driver and the Grid.

* **Selenium Grid**

The Grid can minimize test runtime—by executing multiple test scripts on any number of remote devices at once. This is called parallel testing. Selenium Grid is a smart server that routes test commands to browser instances on remote devices. The two main components needed for this (other than the test script from client-side/tester) are:

* **The ‘Hub’ (server):**

Accepts access requests from WebDriver client. Routes JSON test commands to remote drivers on registered ‘nodes’.

* **‘Node’ (remote device):**

Contains a native OS, browsers, and remote WebDriver.

**How it works:** WebDriver-client executes the test on a faraway device through remote WebDriver. Remote WebDriver is like your regular WebDriver, except its two components are the Client (your test script) and Server (a Java servlet that actually executes the test on the remote device).

In your test script, you define ‘desired capabilities’ (device, platform, browser, etc.) of the node where the test will execute. The Hub receives this script, runs through the registered nodes to find one that matches the [desired capabilities](https://www.browserstack.com/automate/capabilities), and assigns the test to it for execution.

**Note:**Setting up the Grid is pretty easy, but scaling, configuring, and maintaining its integrity can take up a lot of resources. Make sure to adopt it after careful consideration.

**Q2. The most common tools that are used for configuration management are packer and ansible. You need to concisely compare both of them.**

|  |  |
| --- | --- |
| **Ansible** | **Packer** |
| * **Introduction**   Radically simple configuration-management, application deployment, task-execution, and multi-node orchestration engine. Ansible is an IT automation tool. It can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates. Ansible’s goals are foremost those of simplicity and maximum ease of use.  Ansible belongs to "Server Configuration and Automation" category of the tech stack.  **Features :**   * Ansible's natural automation language allows sysadmins, developers, and IT managers to complete automation projects in hours, not weeks. * Ansible uses SSH by default instead of requiring agents everywhere. Avoid extra open ports, improve security, eliminate "managing the management", and reclaim CPU cycles. * Ansible automates app deployment, configuration management, workflow orchestration, and even cloud provisioning all from one system.   "Agentless" is the top reason why over 251 developers like Ansible  Ansible open source tools. It seems that Ansible with 38.2K GitHub stars and 16K forks on GitHub.  DigitalOcean, 9GAG, and Rainist are some of the popular companies that use Ansible.  Ansible has a broader approval, being mentioned in 960 company stacks & 587 developers stacks | * Introduction   Create identical machine images for multiple platforms from a single source configuration. Packer automates the creation of any type of machine image. It embraces modern configuration management by encouraging you to use automated scripts to install and configure the software within your Packer-made images.  Packer can be primarily classified under "Infrastructure Build Tools".  **Features**   * Super fast infrastructure deployment. Packer images allow you to launch completely provisioned and configured machines in seconds, rather than several minutes or hours. * Multi-provider portability. Because Packer creates identical images for multiple platforms, you can run production in AWS, staging/QA in a private cloud like OpenStack, and development in desktop virtualization solutions such as VMware or VirtualBox. * Improved stability. Packer installs and configures all the software for a machine at the time the image is built. If there are bugs in these scripts, they'll be caught early, rather than several minutes after a machine is launched.   while over 24 developers mention "Cross platform builds" as the leading cause for choosing Packer.  Packer open source tool.  Packer with 9.1K GitHub stars and 2.47K GitHub forks.  Packer is used by Instacart, Oscar Health, and Razorpay.  Packer is listed in 115 company stacks and 21 developer stacks. |