Web-based UI Automation Testing using Test Cafe

*Rohit Raj
[*]Department of Computer Science and Engineering, R V College of Engineering, Bengaluru, India.

----------**********----------

Abstract:

In recent years, software testing is an unavoidable aspect of the software Development Life cycle, and its important part in both the pre-and post-development processes necessitates the use of improved and efficient procedures. Two distinguishing aspects of a successful and effective software testing project are the use of proper testing methods and the suitable test automation tools/framework. When it comes to testing software and ensuring its quality, one testing method is rarely sufficient; Instead, a mixture of several relevant testing methodologies is sometimes necessary. The significant advancement in the testing process is Test Automation, which is the use of specific software to carry out the testing process and compare actual results to projected outcomes. This paper presents a comprehensive study of the various test automation framework and also compares the different automation tools. This paper provides insights into the test cafe automation tool which minimizes the effort of the tester and improves the efficiency and quality of the products by automating the whole testing process.

Keywords — **SDLC, Test cafe, testing, Selectors.**

----------**********----------

I. INTRODUCTION

Software programs have become an important part of our daily lives since they can affect millions of people in numerous areas of life, necessitating the development of secure and dependable software. Unfortunately, humans are prone to making mistakes, therefore the fundamental truths of humans' central role in software development make errors an unavoidable part of the process. Errors in software (bugs) can have a detrimental impact on a live operation and even result in death. It is critical to address such faults early in the development process because they become more costly as time goes on.

The process of assessing a software program to identify flaws or defects is known as software testing. Testing takes up around half of a software project's development time and effort, according to the Software Development Life Cycle (SDLC) method. Software is not complete until it has passed testing in the SDLC. The goal of testing is not to show that the system is error-free, but to give assurance that it is running properly before it is installed. An error-free system indicates that the software system was either not tested or tested poorly.

Manual and automated software testing are both possible. Manual testing does not necessitate the use of any testing technology; nevertheless, it does necessitate a lot of labour and more people. Manual testing can be automated in the same way that a programmer may develop programs to automate any manual operation using a coding language. The use of automation tools and/or frameworks in the execution of tests are referred to as automated testing. It necessitates an understanding of automation technologies as well as, in certain cases, programming abilities.

In comparison to manual testing, which is not only costly and time-consuming but also prone to error, automated testing improves testing accuracy and saves tester effort and time.

The significant advancement in the testing process is Test Automation, which is the use of specific software to carry out the testing process and compare actual results to projected outcomes. Test automation saves time by eliminating the need for manual testing, which can be time-consuming.

Test automation has supplanted manual testing by reducing the necessity for it and exposing the number of faults and flaws that manual testing cannot detect.

II. AUTOMATION TOOLS

An automation tool is a piece of software that enables the actual software in consideration to be tested; in other words, an automation tool assists and facilitates software testing. The rapid and widespread development in technology has an impact on how businesses design, validate, distribute, and run the software. Before deciding on a tool, a detailed assessment of the options available should be carried out.

A. Overview of Various Automation Tools

Before deciding on a tool, a detailed assessment of the options available should be carried out.

Test Cafe

It is a free and open source Node.js end-to-end automation tool for testing web applications. Test Cafe doesn't require any web driver or any other software to run the test cases. Test Cafe has their own plugins to generate end-to-end coverage reports. Test Cafe has features to wait for the web page to load and XHR's before the test cases start. It has also smart features to assets some time to load the elements of the web page before appears. It is an instant development tool, in which you can change the code, immediately after restarting the test and you can see the results instantly. It has also Test Cafe Studio features, which is a codeless way to record and maintain the test cases.

There are some limitations of the Test Cafe. For identification of different elements from the DOM, Test Cafe uses only CSS Selectors. It doesn't allow to parallel run the multiple test cases on the same browser. It doesn't support third party framework.

Night Watch JS

It's a Node.js-based automated testing framework for web apps and websites that uses the W3C WebDriver API. It is an end-to-end testing solution that aims to make writing automated tests and implementing Continuous Integration easier.

Night watch interacts with a WebDriver server via a restful HTTP API (such as Chrome Driver or Selenium Server).It executes commands and assertions on DOM elements using the powerful Selenium WebDriver API.

It has limited Framework/Library Support when writing different test cases. It requires explicit waits. Extracting data from the web tables is quite difficult.

Selenium

It is an open-source testing framework for automating browsers for web application testing. It is capable of running multiple tests across multiple browsers.

It makes testing across multiple browsers easier. It is powerful enough to run multiple tests across multiple browsers. It facilitates testing across multiple browsers. To run tests, Selenium Web driver does not require an additional server.

Furthermore, the Web driver launches and controls a browser. JUnit XML reporting is built in, allowing you to integrate your tests into your Jenkins build process. By using Selenium Grid, it is possible to easily scale the application under test.

Selenium performance can be improved, as well as the time needed to run all of the tests can be reduced. Tests are not always very stable due to the heavy use of AJAX and async programming in modern web development technologies. Furthermore, programmed Test Scripts make Test Maintenance more difficult.

III. METHODOLOGY

The project deals with the design and development of a component that automates the testing process. Implement features that allow the users to efficiently manage the End to End Test Coverage and lastly, generate the test report. Most of the code written in framework is reusable across different use cases. However, to validate the different elements on the website like drop down menu, login function, or the framework is smart enough to compile the single code base to native elements of the platform. Since applications are developed on different platforms, a centric solution is required.

The objective of Automation:

- Build a generic framework that can be used in other test case scenarios.
- Build a concise, interactive format of report that can be analyzed by the user.
- End-to-End Test coverage.
- UI can be tested against Vertica Database.
- UI can be tested against all compatible Browsers.

A. Architecture of an Applications:

It has follows different modules:

- Test Cafe (Validation Against different elements presents on the DOM).
- Validation of populated data displayed on the UI against data present in the Vertica Database.
- Generate the test reports on the CL/CLI.

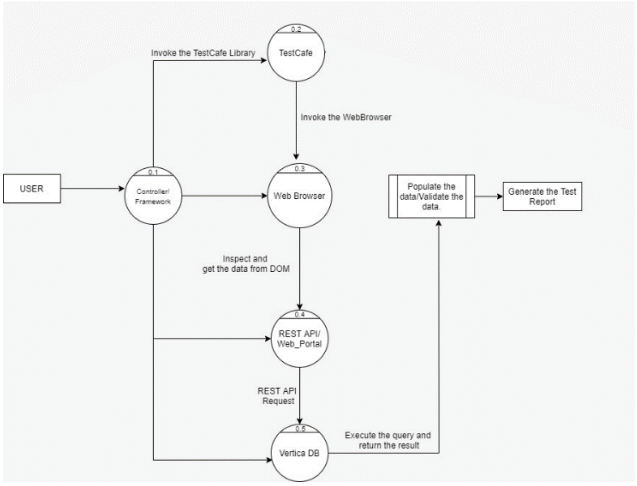

Fig. 1 Overall architecture of an application

B. Architecture of a Test Cafe:

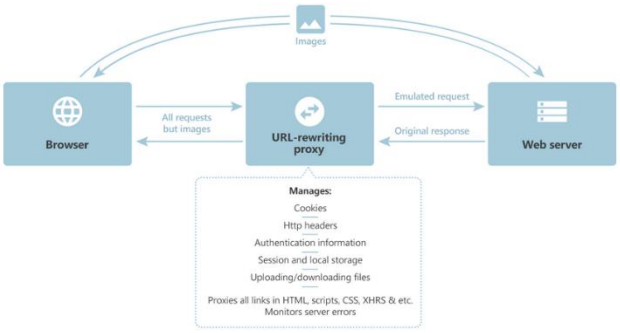

Fig. 2 Architecture of a Test Cafe

Test Cafe runs on the test cafe server, and it will open test cafe browsers in which web page will be shown. Test Cafe has URL-rewriting proxy features that allow the browser to automatically redirect the web page to the browser. The web browser sends the request to the URL-rewriting proxy. Then URL proxy sends an emulated requests to the web server and the web server sends back the original response and it should visible in the web browser.
The URL-rewriting proxy also manages the cookies, HTTP Headers, Authentication Information, Session, and Local Storage and also provides all links in HTML scripts, XHR, CSS.

C. Validation against UI:

The website has different components/dashboards or elements present on the DOM such as Radio Buttons, Drop-Down menus, Title of the Header, and many different components to validate the website after deploying. Usually, the tester has to do it manually, which leads to a decrease the efficiency and increase the manual efforts and also compromise in delivering quality products.

Test Cafe provides a smart assertion query that automatically validates the different components, by writing the assertion query. It will inspect the element from the DOM and fetch the Selectors of the particular components. The assertion query will start running and match the selectors, then it validates the data present in the selectors and

generates the test case results. But some of the use cases, it's very difficult to assert the query and validate, because it is not easy to organize the test cases.

For authentication and authorization, the website has SSO in which every time you need to login into the website to validates the test case. Test cafe has a Test hook feature that allows running the test hooks before starting the test case and also after completed the test case.

```
fixture(`WEB Portal Test`, { noReloadBetweenTests: false })
    .page(page.url)
    .beforeEach(login)
    export async function login(page)
{
    await t
    .typeText(page.usernames, 'username')
    .typeText(page.passwords, 'password')
    .click(page.submitbtn)
    .maximizeWindow()

};
```

Fig. 3 Code Snippet of generic login function

For dropdown menus, the website has different dropdown options, Test cafe provides assertion to check the dropdown options only once at a time, in that case, it will increase the lines of code, and for every dropdown option, a tester has to write the assertion query. We developed a generic function that allows the tester to run only one assertion query by passing the array of dropdown selectors, it will automatically check for all the dropdown options. It can be reusable in other use cases as well. It increases efficiency as well as saves time.

```
export async function dropDown(locatorArray, label)
{
    for(let i=0;i<locatorArray.length;i++)
    {
        var a = locatorArray[i];
        await t.click(Selector(label).withText(a))
        await t.wait(1000);
        await t.click(Selector(label).withText(a))
        await t.expect(a).ok('DropDown Failed',{timeout:12000})

    }
};
```

Fig. 4 Code Snippet of generic dropdown menu function

The website has different web tables, we need to validate the headers of rows and columns of the table. Test Cafe provides assertion to verify the inner text of the rows and columns of the table, but again it will check only once at a time, so for all the rows we need to write the assertion query, it will increase the lines of code and also decrease the efficiency. Developed a function that automatically checks all the rows and columns by passing a selector of the table and array of the headers. It can be reusable in other use cases as well.

The website has different filters and after selecting the filters they should visible on the dashboard and then compare the selected filters against the dashboard filters displayed. We organize our test cases in that approach, first, it should select the filters and selected filters will be stored in the array and then sort the array. Dashboard filters are also stored into an array and sort array. Then compare the array, if both the arrays are equal, then its pass otherwise it fails.

When test cases got failed it will automatically take a screenshot and store it into the local drive that's easy to find out the error.

D. Validation against Vertica Database:

After Validating the front-end different dashboards/components elements, need to validate the backend data that's showing on the front end, it should be matched with the data that are present in the Vertica Database. As a manual tester usually, need to check both the front end as well as the back end data and compare the values. It will increase the time and also decrease the efficiency. Here, we are automating the whole process using Test Cafe. First, Test Cafe provides an assertion query to find the selector and fetch the values from the selector and store it into the map. REST-API web services make a connection between the Vertica Database and will enable the

query to populate the data from the Vertica database on the front end and also store the Vertica database data into the map and then compare both the maps if it matches all the data correctly, it will pass the test case otherwise it fails.

E. Code Reusability

Most of the code written in framework is reusable across different use cases. However, to validate the different elements on the website like drop down menu, login function, or the framework is smart enough to compile the single code base to native elements of the platform.

IV. RESULTS AND ANALYSIS

The ultimate goal of the project is to provide the testers to generate end to end test coverage reports without doing manually, by automating the whole testing process. The experiments show how the framework is beneficial to the testers. This papers also provide an overview of the different automation tools.

	Night Watch	**Selenium**	**Test Cafe**
Upgradable without Downtime	Supported	Supported	Supported
Easy Setup	Supported	Unsupported	Supported
Cloud Integration	Supported	Supported	Supported
Test Reporting	Supported	Supported	Supported
Cross Browsing Support	Supported	Supported	Supported
Built in waits	Supported	Unsupported	Supported
Parallel/Grid Execution	Supported	Supported	Supported
Ease of Integration with existing infrastructure	Supported (with exceptions)	Supported (with exceptions)	Supported
Support for CI/CD	Supported	Supported	Supported
Third Party Framework Support	Supported	Supported	Unsupported
Python/ Pytest Scripts	Supported (with exceptions)	Supported	Supported (with exceptions)
Easy to Debug	Supported	Supported	Supported
Unit Testable	Supported	Supported	Supported
Code Maintenance	Supported	Supported	Supported

Fig. 5 Comparison of different automation tools.

All three of the UI tools on the shortlist were good, and each had its own set of advantages and disadvantages. Test Cafe does not require any external plugins to run tests on different browsers because it has its own plugins and also it has a smart assertion query mechanism to validate different elements on the dashboard, which saves testers time and effort of installing plugins/drivers. It could be easily integrated with Existing infrastructure. It would cover more than 75% of the requirements of UI testing for products and hence would be a good candidate to start with.

After successfully completed all the test cases, it will generate the coverage reports by using test cafe plugins, and also after each test case, it will reflect the status of test cases on the CL/CLI and if there are any errors it will show on the Command Line Interface.

Fig. 6 Snapshot of test report on the CLI.

V. CONCLUSION

A well-defined test automation framework will aid the testing team in increasing the reusability of test components, creating readily maintainable scripts, and obtaining high-quality test automation scripts. An automation tool is a piece of software that allows the actual program in question to be tested; in other words, an automation tool aids and facilitates software testing. The Test Cafe automation tool has

evolved into an essential component of successful software testing. This paper provides an overview of the test cafe automation tool which reduces the effort of the tester and improves the quality of the products.

REFERENCES

[1] M. Böhme, "Assurances in Software Testing: A Roadmap," 2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), 2019, pp. 5-8, doi: 10.1109/ICSE-NIER.2019.00010.

[2] Alaqail, Hesham & Ahmed, Shakeel. (2018). Overview of Software Testing Standard ISO/IEC/IEEE 29119. 18.

[3] Monika Thakur, Sanjay. Review on Structural Software Testing Coverage Approaches, International Journal of Advance Research, Ideas and Innovations in Technology, 2017.

[4] M. A. Jamil, M. Arif, N. S. A. Abubakar and A. Ahmad, "Software Testing Techniques: A Literature Review," 2016 6th International Conference on Information and Communication Technology for The Muslim World (ICT4M), 2016, pp. 177-182, doi: 10.1109/ICT4M.2016.045.

[5] Umar, Mubarak Albarka & Chen, Zhanfang. (2019). A Study of Automated Software Testing: Automation Tools and Frameworks. 8. 217-225. 10.5281/zenodo.3924795.

[6] Milad Hanna, Amal Elsayed Aboutabl and Mostafa-Sami M Mostafa. Automated Software Testing Frameworks: A Review. International Journal of Computer Applications 179(46):22-28, June 2018.

[7] Shruti Malve, Pradeep Sharma, "Investigation of Manual and Automation Testing using Assorted Approaches," International Journal of Scientific Research in Computer Science and Engineering, Vol.5, Issue.2, pp.81-87, 2017.

[8] K. Valliammai, Dr. P. Sujatha, "Analysis of Efficiency of Automated Software Testing Methods: Direction of Research", International Journal of Science and Research (IJSR), Volume 5 Issue 12, December 2016, 34 – 38.

[9] 9. Shah, Hezbullah & Soomro, Tariq, Node.js Challenges in Implementation. Global Journal of Computer Science and Technology. (2017), 17. 72-83.

[10] 10. Biswajeet Sethi, Samaresh Mishra, Prasant Kumar Patnaik, 2016, A Study of NoSQL Database, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) Volume 03, Issue 04 (April 2016).

[11] 11. Silva, Yasin & Almeida, Isadora & Queiroz, Michell. (2017). SQL: From Traditional Databases to Big Data. 413-418. 10.1145/2839509.2844560.

[12] 12. Neumann, Andy, Nuno Laranjeiro, and Jorge Bernardino. "An analysis of public REST web service APIs." IEEE Transactions on Services Computing (2018).

[13] 13. Zhao, J. T., S. Y. Jing, and L. Z. Jiang. "Management of API gateway based on micro-service architecture." Journal of Physics: Conference Series. Vol. 1087. No. 3. IOP Publishing, 2018.