International Journal of Enhanced Research in Science, Technology & Engineering
ISSN: 2319-7463, Vol. 13 Issue 5, May-2024, Impact Factor: 8.375

Automated UI Testing on Frontend of a Web Application

Shivam Dhonde[1], Atharva Joshi[2], Tanmay Jadhav[3], Shital Hote[4], Mr. Manish Jansari[5]

[1,2,3,4]Final Year, Department of Computer Engineering Pune Institute of Computer Technology, Pune
[5]Assist. Professor, Department of Computer Engineering Pune Institute of Computer Technology, Pune

ABSTRACT

"This paper delves into the critical role of automated testing within software engineering practices, focusing on its profound impact on software testing methodologies. It illuminates the crucial function of automated testing in gauging the efficacy and capabilities of software programs, ensuring their quality and reliability. Furthermore, the paper navigates through the complexities of measurement guidance, addressing discrepancies and emphasizing the indispensability of automation in testing processes. It offers comprehensive insights into the evolving landscape of automation solutions, particularly highlighting techniques such as XPath generation from HTML structures for facilitating thorough UI testing. The primary objective of this study is to provide a nuanced understanding of the practical application of automated testing in software development, elucidating its far-reaching effects on software performance and development agility. In essence, the paper sheds light on contemporary practices that harness automation to bolster software development endeavors and foster innovative strides in software testing and engineering, ultimately bolstering the dependability of UI testing."

***Index Terms*—Behavior-driven development (BDD), Modeldriven approach, Automated Testing, Behave, XPath, GUI Testing**

INTRODUCTION

Software design is the design and application of sound engineering principles to create suitable software that is both reliable and works on real systems. Creating a good software product requires a good development process. Software development is a human activity that involves many tasks. These activities; Analysis, design, implementation and testing all lead to the creation of the final product. Because these activities occur constantly during development, it takes time to create a working version of the system. Software testing is one of the most important activities in software development used to define and verify software systems. Testing helps software developers ensure that the software they create performs its intended function and determines whether any problems are fixable. As the software development life cycle is a complex process, there is an urgent need to deliver new products within the stipulates time. In the software industry, automation plays an important role in test development. Various automation tools are available to streamline the testing process. Automation in UI testing is a vital component of modern software development, aiming to streamline the testing process and enhance efficiency. By leveraging automation tools and frameworks, organizations can achieve faster test execution, improved test coverage, and more reliable results. This paper explores the role of automation in UI testing, highlighting its importance in accelerating the software development lifecycle and ensuring the delivery of highquality products. Additionally, it discusses the challenges faced in traditional manual testing approaches such as Human error, Resource Intensive, Limited Coverage, Time Consuming, etc and how automation addresses these issues. Through real-world examples and case studies, the paper demonstrates the impact of automation on reducing time-to-market and enhancing overall software quality.

AUTOMATION IN GUI TESTING

Automation plays an important role in GUI testing. It helps to reduce time taken for GUI testing and increase the efficiency. Overall, using automation in GUI testing would drastically impact the process of GUI testing. Here are few points regarding Automation in GUI testing as follows:

A. Role of Automation in UI Testing

Automated UI testing is a critical component of software development, ensuring the robustness and reliability of web applications. It allows for the rapid execution of test cases compared to manual testing. Automation enables testers to cover a broader range of test scenarios, including edge cases and boundary conditions.

B. Dynamic Element Identification

One of the challenges in traditional UI testing lies in the identification of dynamic elements within web interfaces. Automated UI testing addresses this issue by locating elements using Xpath. This dynamic element identification ensures that the test scripts remain robust even in the face of evolving web applications.

C. Self-learning Test Automation

Self-learning in automated UI testing refers to the ability of testing frameworks or tools to adapt and improve their testing capabilities based on past test execution results and experiences. Through continuous feedback loops, the testing system learns from test results and refines its understanding of the application under test. This adaptive capability significantly reduces maintenance efforts and ensures the sustainability of UI test automation in the long run.

D. Test Case Generation and Optimization

Automation generates test cases based on the extracted HTML structure and XPath of interactive elements. It maximizes test coverage while minimizing redundancy and resource consumption. This automated test case generation minimizes the manual effort required for test script development and maintenance.

E. Intelligent Test Execution

Intelligent Test Execution prioritizes test cases based on factors such as criticality, risk, and dependencies. Highpriority tests are executed first to provide faster feedback on critical functionalities or areas of the application. It Analyzes code changes and their impact on the application to intelligently select a subset of regression tests that are most relevant to the changes.

F. Enhancing Test Coverage

Traditional UI testing often struggles to achieve comprehensive test coverage due to resource limitations. Enhancing test coverage in automated testing involves strategies such as requirement-based testing to ensure comprehensive alignment with specifications, boundary value analysis to assess the application's behavior at critical limits, and exploratory testing sessions to uncover unforeseen scenarios and usability issues beyond scripted tests

IMPLEMENTATION

A. Input website URL

The initial step of the proposed methodology involves identifying and specifying the URL(s) of the website(s) or web application(s) that require UI testing. This could include the mainly landing page.

B. Extract HTML structure of web pages

After the website URLs are determined, we extracted the HTML structure of the web pages programmatically. This process typically involves web scraping using specialized tools or libraries. Web scraping involves fetching the HTML content of web pages using HTTP requests and parsing the retrieved HTML documents to extract relevant information such as tags, attributes, and text content. Then we analyzed, the extracted HTML structure to identify the hierarchy of elements, including divs, spans, inputs, buttons, forms, and other UI components. This information serves as the foundation for generating XPath expressions and defining test scenarios.

C. Generate XPath for UI elements

The objective of this step is to effectively identify and interact with UI elements on the web application under test using XPath expressions in Selenium with Python Behave framework. To identify the inetractive useful elements from the website we have used the selenium different methods such as by its class name, id, text, aria-label, etc. There are two strategies of the xpath i.e. Absolute xpath and relative xpath, so we used absolute full xpath which is unique.

Sample XPath of "userName":
/HTML[1]/BODY[1]/DIV[1]/DIV[2]/DIV[1]/DIV[2]/DIV[1]/
MD-CONTENT[1]/FORM[1]/DIV[1]/DIV[1]/MD-INPUT-
CONTAINER[1]/INPUT[1]

By genearting this xpaths for each element respectively, we tested it according to the need and checked whether is it possible go with this approach or is there is correction reuired or not if yes then of course go with that. Incorporating XPath expressions within Selenium with Python Behave framework facilitates precise and efficient UI element identification, contributing to the overall effectiveness of automated testing efforts.

D. Create a feature file for test scenarios

After extracting the HTML structure of web pages and generating XPath expressions for UI elements, feature file is created automatically to determine test scenarios to define the test scenarios and behaviors to be validated through UI testing. A feature file is typically written in a human-readable format using a behavior-driven development (BDD) framework such as Gherkin syntax. Gherkin allows for clear and concise representation of test scenarios in a structured format. Each feature file consists of one or more scenarios, each representing a specific test case or user story Steps within a scenario are defined using keywords such as Given, When, and Then, representing the initial state, user actions, and expected outcomes, respectively.

For example:

Feature: Login to eHub Application

Scenario: User logs in with valid credentials
Given I launch Chrome Browser
When I am on the login page
When I enter the username "Test@user" and password "Test@pass"
And I click the login button
Then I should be logged in successfully

E. Execute UI tests based on feature files

After creating feature file we go for the step defintion implementation part. Step definitions in the context of Behavior-Driven Development (BDD) frameworks such as Behave provide the implementation logic for the scenarios outlined in the feature files. Each step definition corresponds to a specific step written in the feature files using Gherkin syntax, such as Given, When, and Then. Generated feature file is then mataches with its step definition to execute the steps taken by the webdriver to execute the implementation. During test execution, Behave matches these steps to their respective step definitions based on textual similarity, ensuring that the appropriate automation logic is executed for each step. For execution follows steps are required:

Setup: Ensuring the Selenium WebDriver set up and initialized in your testing environment. This includes importing necessary libraries and configuring the WebDriver. **Running Tests**: Executing the Behave test suite using the command-line interface or an integrated development environment (IDE) that supports Behave. cmd for behave

F. Adapting XPaths for dynamic changes

In automated testing, robust and resilient XPath expressions are essential for accurately identifying UI elements on web pages. However, web applications are prone to changes in their structure, such as updates to HTML attributes, modifications in element positioning, or alterations in the DOM hierarchy. To ensure the longevity and reliability of automated tests, it is crucial to implement strategies for adapting XPath expressions to accommodate dynamic changes in the application. Through this way we can adapt the xpaths accordingly the element presents on website. XPath expressions can be made more resilient to changes in the application's UI structure, thereby enhancing the maintainability, flexibility, and scalability of automated testing efforts. This can be acheived through our xpath approach.

G. Collecting and presenting test results

After test execution, results including pass/fail statuses, error messages are collected and logged Test results are aggregated and presented in a comprehensive format using reporting tools

Providing insights for continuous improvement The collected test results serve as valuable feedback for identifying patterns, trends, and recurring issues in the software under test. Insights derived from test results are used to drive continuous improvement efforts, including refining test cases, enhancing test coverage, and optimizing testing strategies.

RESULTS AND DISCUSSION

Automating test cases significantly enhances time efficiency compared to manual testing, as it intelligently generates and executes test cases. This approach not only saves time but also ensures thorough testing coverage and consistency in results. Taking Sarvatra'seHub website as an example, which is a comprehensive platform offering a wide range of

services, including but not limited to ecommerce, booking systems, and content management. With its extensive database and intricate user management system, ensuring the reliability and functionality of the website is paramount. The frontend of Sarvatra'seHub website is built using a combination of AngularJS, HTML, CSS, and Bootstrap, showcasing a modern and user-friendly interface. However, with the complexity of the website's features and the continuous updates and changes in web technologies, manually testing every aspect of the website becomes impractical and time-consuming.

Lets go through the implementation steps for a Frontend Website.

Step 1: Consider website:

https://uatehub.sarvatra.in:6443/ehubuat1/#!
/login

Step 2: Extract HTML structure of web page: We are extracting the HTML structure of a web page and identify the elements on the basis of id, name, area-label, class, type, etc.

```
for index, element in enum(interactive_elements):
  # Get the element name based on text content, id, class, or type
      element_name = (
          element.get_attribute("id") or element.get_attribute("name") or element.text.strip() or element.get_attribute("aria-label") or element.get_attribute("class") or element.get_attribute("type") or f"Element{index}"
      )
```

Step 3: Generate XPath for finding interactive elements: we are generating the xpaths of this interactive elements. We can use this XPath to locate the element while performing automation testing

```
{
    "userName": "/HTML[1]/BODY[1]/DIV[1]/
    DIV[2]/DIV[1]/DIV[2]/DIV[1]/MD-CONTENT[1]
    /FORM[1]/DIV[1]/DIV[1]/INPUTCONT[1]/INP[1]",
    "password": "/HTML[1]/BODY[1]/DIV[1]/DIV[2]/
    DIV[1]/DIV[2]/DIV[1]/MD-CONTENT[1]/FORM[1]/
    DIV[1]/DIV[2]/MD-INPUT-CONTAINER[1]/INPUT[1]"
}
```

Step 4: Create a feature file:

Now, the feature file has been created. Below is an example of how the feature file appears.

```
{
  Feature: Login to eHub Application
                                    Scenario: User logs in with valid credentials
      Given I launch Chrome Browser
      When I am on the login page
      When I enter the user "Agent_22" and pass "Test@12"
      And I click the login button
      Then I should be logged in successfully
}
```

Step 5: All steps that are written in the step definition are executed/implemented:

Now this feature file is executed and the GUI testing is executed automatically. Below are the screenshots of the execution process.

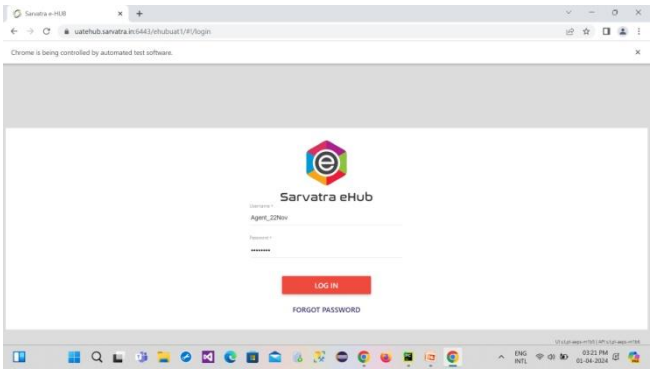

Fig. 1.Login page Test case implementing

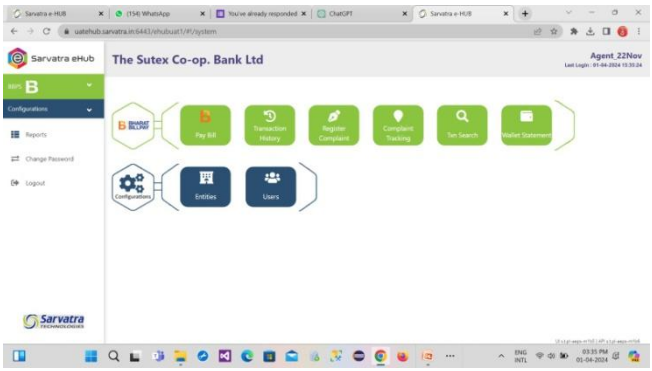

Fig. 2.Login page Test case implementing

Step 6: Collecting and presenting results

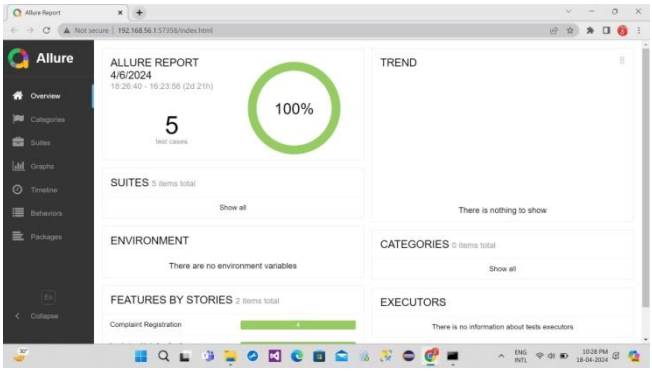

Fig. 3. Allure Report (Fig A)

Once the execution is done, we get the detailed reports about the testcases on Allure Software. We also get the time taken to execute each test case in this Allure report. Below are the screenshots of the allure reports (fig 3, fig 4). This way the overall implementation of all step definitions of feature file is executed and we get the final results.

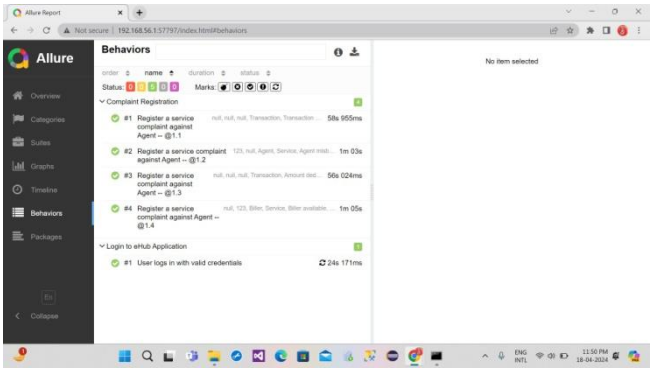

Fig. 4.Allure reports (Fig B)

This majorly uses the concept of XPaths for locating the interactive elements and then executing the feature file. We also get the details about the results which includes the number of test cases Passed/Failed, etc.

CONCLUSIONS

In conclusion, the implementation of Automation in UI testing represents a significant advancement in software quality assurance, offering a more efficient, reliable, and scalable approach to validating the user interface of modern web applications. By leveraging techniques such as web scraping, XPath generation, and automated test case generation, we

have demonstrated the ability to systematically extract HTML structures, identify UI elements, and define test scenarios in a human-readable format Furthermore, our methodology encompasses adaptive strategies for handling dynamic changes in web environments, ensuring the robustness and stability of UI tests across different states of the application

REFERENCES

[1]. F. Macchi, P. Rosin, J. M. Mervi and L. Turchet, "Image-based Approaches for Automating GUI Testing of Interactive Webbased Applications," 2021 28th Conference of Open Innovations Association (FRUCT), Moscow, Russia, 2021, pp. 278-285, doi: 10.23919/FRUCT50888.2021.9347592.
[2]. F. Ricca, M. Leotta, and A. Stocco, “Three open problems in the context of e2e web testing and a vision: Neonate,” Advances in Computers, 01 2018.
[3]. K. P. Moran, C. Bernal-Cardenas, M. Curcio, R. Bonett, and D. Poshy- ´ vanyk, “Machine learning-based prototyping of graphical user interfaces for mobile apps,” IEEE Transactions on Software Engineering, 2018.
[4]. M. Leotta, A. Stocco, F. Ricca, and P. Tonella, “Automated migration of DOM-based to visual web tests,” in Proceedings of 30th Symposium on Applied Computing, ser. SAC 2015. ACM, 2015, pp. 775–782
[5]. P. Tonella, F. Ricca, and A. Marchetto, Recent advances in web testing, Adv. Comput. 93 (2014), 1–51. http://doi.org/10.1016/B9780-12-800162-2.00001-4.
[6]. M. Leotta, Z. Oliveira, A. Memon, Approaches and tools for automated end-to-end web testing, Adv. Comput. 101 (2016), 193–237. http://doi.org/10.1016/bs.adcom.2015.11.007.
[7]. E. Alégroth, Z. Gao, R. Oliveira, A. Memon, Conceptualization and evaluation of componentbased testing unified with visual GUI testing: an empirical study, 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), 2015, pp.
[8]. 1–10. http://doi.org/10.1109/ICST.2015.7102584.
[9]. Xie, X., Xu, B., Nie, c., Shi, L., and Xu, L. 2005. Configuration Strategies for Evolutionary Testing. In Proceedings of the 29th Annual international Computer Software and Applications Conference Volume 02 (July 26 - 28, 2005). COMPSAC. IEEE Computer Society, Washington, DC.
[10]. Benoit Baudry, Automatic Test Case Optimization: A Bacteriologic Algorithm, IEEE SOFTWARE Published by the IEEE Computer
[11]. Society, 2005.
[12]. A. Bertolino "Software Testing Forever: Old and New processes and techniques for Validating Today’s Applications", Keynote at 9th International Conference Product-Focused Software process Improvement (PROFES 2008), Monte Porzio Catone, June 2008, LNCS 5089 , 2008.
[13]. V. Mohan, D. Jeya Mala "IntelligenTester -Test Sequence Optimization framework using Multi-Agents", Journal of Computers, June 2008.
[14]. Memon, A M., Soffa, M. L. and Pollack, M. E., Coverage criteria for gui testing. ESECIFSE9: Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering, New York, NY, USA, 2001, ACM Press, pages 256-267
[15]. L.C. Briland.,“On the many ways Software Engineering can benefit from knowledge engineering”, Proc. 14th SEKE, Italy, pp3-6, 2002.
[16]. Benoit Baudry, Automatic Test Case Optimization: A Bacteriologic Algorithm, IEEE SOFTWARE Published by the IEEE Computer
[17]. Society, 2005.
[18]. P. McMinn, “Search-based test data generation: A survey”, Journal on Software Testing, Verification and Reliability, 14(2):105–156, June 2004.
[19]. Partridge, D.” The relationships of AI to software engineering”, Software Engineering and AI (Artificial Intelligence), IEE Colloquium on (Digest No.087), Apr 1992.
[20]. Bennett, S.; Linkens, D.A.; Tanyi, E.B.; Scott, A . “Application of AI and model building techniques to software engineering”, Software Engineering and AI (Artificial Intelligence), IEE Colloquium on (Digest No.087), Apr 1992.
[21]. Partridge, D.” The relationships of AI to software engineering”, Software Engineering and AI (Artificial Intelligence), IEE Colloquium on (Digest No.087), Apr 1992.
[22]. Last, M., Kandel, A., and Bunke, H. 2004 Artificial Intelligence Methods in Software Testing (Series in Machine Perception &Artifical Intelligence " Vol. 56). World Scientific Publishing Co., Inc.
[23]. Partridge D. “To add AI, or not to add AI?, In Proceedings of Expert Systems, the 8th Annual BCS SGES Technical conference, pages 313, 1988.
[24]. E. Borjesson and R. Feldt, "Automated System Testing Using Visual GUI Testing Tools: A Comparative Study in Industry," 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, Montreal, QC, Canada, 2012, pp. 350-359, doi: 10.1109/ICST.2012.115.

[25]. Alegroth E, Feldt R, Olsson H (2013a) Transitioning Manual System Test Suites to Automated Testing: an Industrial Case Study. In: IEEE Sixth international conference on software testing, verification and validation, ICST 2013. IEEE, pp 56–65
[26]. Alegroth E, Nass M, Olsson H (2013b) JAutomate: a tool for systemand acceptance-test automation. In: IEEE Sixth international conference on software testing, verification and validation (ICST), 2013. IEEE, pp 439–446
[27]. Berner S, Weber R, Keller R (2005) Observations and lessons learned from automated testing. In: Proceedings of the 27th international conference on software engineering. ACM, pp 571–579
[28]. Cadar C, Godefroid P, Khurshid S, Pasareanu CS, Sen K, Tillmann N, Visser W (2011) Symbolic execution for software testing in practice: preliminary assessment. In: 33rd international conference on software engineering, ICSE 2011. IEEE, pp 1066–1071
[29]. Grechanik M, Xie Q, Fu C (2009a) Creating GUI testing tools using accessibility technologies. In: International conference on software testing, verification and validation workshops, 2009, ICSTW'09. IEEE, pp 243–250
[30]. Grechanik M, Xie Q, Fu C (2009b) Experimental assessment of manual versus tool-based maintenance of GUI-directed test scripts. In: IEEE international conference on software maintenance, ICSM 2009. IEEE, pp 9–18