An Automated Testing Tool for Web Application

Course Project 5

Leader:

Student 1 Student 2 Student 3 Student 4 Student 5

***Abstract* – Nowadays, web-based applications develop rapidly. The web application complexity and testing difficulty are increasing significantly. In fact, developers will easily make mistakes on software development process [1]. Also, the software will lead to failure due to misunderstandings between different stakeholders such as developers, testers, and product owners. So, software testing plays a vital role to find defects and ensure different stakeholders have clear understanding of the software in software development process. In this project, we used Teamwork as scrum software to develop an automated testing tool using Selenium WebDriver for testing web applications. We also implemented Behavior-Driven Development (BDD) framework – Behave in our testing tool to provide descriptive test cases. With our tool, user can provide the test cases in descriptive language (English) as specifications. Thus, testers can focus on writing test suite towards the specifications and developers can analyze their code and produce a testing report for the test efficiently. It will not only reduce the communication cost of software development, but also help to ensure the correctness of the software towards business value. We created a demo web application and API for our automated testing tool evaluation.**

***Index Terms* – Automated testing, Web application testing, Web application testing tool, Behavior-Driven Development**

I. INTRODUCTION

A typical web application consists of frontend (Web pages) and backend (API & database). Users can interact with web applications over the Internet using a browser [4]. For example, user input to the web application using browser and submit the input data into database through API request. The content of the web application is usually dynamic based on the user input. Web application runs on web servers are more dynamic and accessible compared to traditional applications which typically run on specific devices. Therefore, there has been a sharp rise in software systems implementing web-based applications. However, web applications are more complex than traditional applications. Traditional software testing tools are not adequate for web applications since traditional testing tools do not have the ability to tackle the features of web applications. For instance, a web application can work on different browsers. This requires a testing tool that supports multiple browsers which the traditional testing tool cannot. In addition, the web application involves communication with server which requires a testing tool that has the ability to simulate and test the interaction between browser and server. Testing web applications is significant because it can reveal software defects as early as possible and reduce the cost of technical debt in the later stage of software development cycle [5].

Apart from web application testing tool, the test needs to be automated. Automated testing can reduce human effort and time in the testing process compared to manual testing. Humans are prone to error, there is a risk of human error in the testing process. It will lead to inevitable errors in the software. With automated testing, it can execute test cases accurately and repeatedly which can improve the quality of testing process and ensure developers deliver quality software. In addition, testers can scale up and reuse automated testing tools easily. After creating an automated test case, tester may reuse the test case to execute the test on different browsers. Also, testers can create multiple data samples for the same test case to scale up their test by using automated testing tool.

Besides, in the typical project, lack of communication is the major reason that led to project failure [13], especially the large-scale project. Large-scale software projects require interaction between many stakeholders [14]. Therefore, a testing tool that connects different stakeholders will substantially increase the project success rate.

Most developers are familiar with Test-Driven Development (TDD) and conduct testing with this approach. TDD is used to test a certain piece of functionality. It is written in programming languages which it not easy for all stakeholders to understand except developers. There will be misunderstandings between different stakeholders. If we use TDD approach in the testing process, stakeholders may not be able to validate and provide feedback on software. To avoid these problems, the Behavior-Driven Development (BDD) approach came into picture. BDD typically involves developer, tester and product owner [6]. They will come up with concrete scenarios of the user story. These scenarios are written in descriptive language like English which is more readable and easier to understand. Executable tests for the scenarios can be written. BDD provides end to end testing instead of unit testing like TDD.

In this paper, based on the scrum framework, we develop a testing tool to test web applications automatically with descriptive test cases. It enhances not only the understanding of software between different stakeholders, but also improves the quality of software to mitigate the chance of software failure. Therefore, an automated testing tool with BDD approach for web application is developed by us based on selenium WebDriver and Behave feature. The test cases (scenarios) are written in English as specification for our testing tool to execute

which provide usability to users. Our testing tool can support test execution on different browsers such as Google Chrome, Mozilla Firefox, and Microsoft Edge which can provide compatibility to users. In the test execution, our tool will perform user input simulation, input validation, network request and response validation. Our tool also screenshots each step of test cases for testing report generation. After test execution, an HTML based testing report will be returned. In the testing report, screenshot, and testing result (pass/fail) of each step within scenarios can be visualized.

To evaluate our testing tool, we developed a demo form submission web application and API. Users can input different fields in the form and submit the data to the database through API. Our testing tool can simulate a form submission (user story) on different browsers and perform input validation, network validation and report generation for this user story.

The paper is organized as follows: Section II gives a review of existing testing techniques. Section III describes the technical background of the automated testing tool. Section IV presents the details of the automated testing tool using proposed techniques. Section V presents the software process for the testing tool development. Section VI presents the evaluation of the testing tool. Section VII presents the conclusion of the work in this paper.

II. Related Work

As mentioned in the Introduction, traditional testing method has its limitations, like time consuming, programming language barrier for stakeholders, etc, we need to have a tool for automated testing for web applications. The following are the facts finding about the existing solutions in the industry for solving the problem.

A. Advantages of Selenium WebDriver

We chose Selenium WebDriver which is an open-source automation testing tool for the software testing tool in our project. According to a survey conducted by a data mining company, Enlyft, it holds the largest market share of 25% [2]. This is attributed to several key factors making Selenium outstanding from other automation testing tools available in the market.

One of the advantages is Selenium extensive features which caters to the different requirements of software testers. It offers comprehensive functionality that enables testers to automate testing web-based applications. It also provides large support for various programming languages and allows testers to write automation scripts in their preferred language. It is also compatible for multiple browsers including Google Chrome, Mozilla Firefox, Safari and Microsoft Edge.

Secondly, Selenium allows testers to choose different programming languages, frameworks and third-party libraries based on their requirements. This flexibility enables testers to adapt their automation approach to suit the unique needs of their projects. It can integrate with other testing tools and frameworks, such as Cypress and Puppeteer which can streamline the test execution and reporting.

Lastly, Selenium has a strong community compared to other tools like Apache JMeter, Playwright, Cypress, and Puppeteer. It has an extensive and active user community that contributes to its development, support, and knowledge sharing. This community significantly benefits testers by providing abundant resources including documentation, tutorials, forums and plugins. Testers can leverage this knowledge and expertise to efficiently and effectively enhance their automation practice and stay up-to-date with the latest advancements in Selenium.

B. Descriptive Language

During our studies on various automation testing tools, we found that all of them require some level of coding for creating test cases. Although tools like Apache JMeter provide a GUI-based interface that enables testers to create test cases without writing code, we found that there are some limitations, such as flexibility. Though the GUI-based interface enhances testing efficiency and deals with varying coding expertise, it cannot handle complex logic, data-driven testing, and custom data generation for randomized test values.

To address this limitation and enhance test case development, we found that it is necessary to develop a descriptive language for testing tools that promotes readability, maintainability, collaboration, and reusability. In this project, we decided to implement a descriptive language of Gherkin, which is a popular language in Behavior-Driven Development (BDD) frameworks. It is a human-readable format for defining the behavior of software based on different scenarios. We used Behave for BDD framework. It acts as a bridge between the descriptive Gherkin language and implementation code. Testers can write steps in Gherkin language and map to corresponding code. Then, this code enables Selenium which interacts with web elements, perform actions and make assertion.

We introduced a layer of abstraction which enhances the expressiveness of the test cases. Testers can write test scenarios in natural language format and enhance the collaboration with stakeholders and easily understood. It also enhanced the maintainability of the test suite by using natural language because changes in requirements and business logic can easily update. It reduces the cost of changing the code.

C. Using Allure for Reporting

During the review of the existing options, we found that most of the similar applications included a reporting function. Therefore, we decided to include Allure to enhance reporting capabilities, test execution reports and detailed test case information with Selenium. With Allure, the testing tool can create comprehensive and visual reports for testers. We have HTML format of report with screenshots of each test case which enable testers to include examine error stack or other artifacts that provide additional context and evidence of test execution. It is useful for capturing visual evidence of test failures or unexpected behavior for troubleshooting and debugging.

Additionally, it can also generate reports with metrics such as the total number of test cases executed, the number of successful tests and specific details about failed test cases. These reports provide testers with insights into the overall quality and

reliability of the test cases and make further enhancement to the web application.

D. Network Request and Response Validation

The automation testing tool should support the automated execution of test cases designed for validating network requests and responses. Testers can be able to define and configure test cases on various network requests and corresponding responses. Request headers, request payload and request methods, such as GET, POST, PUT, etc, can be defined when creating new test cases. Testers are able to customize these parameters to simulate different types of requests and verify the correct construction. Testers can specify the desired response codes, response headers and response payload. This function of defining the expected network response parameters can enable tester to ensure the application has correct responses.

III. PRELIMINARIES

The automated testing tool aims to reduce or hopefully eliminate the communication cost between different stakeholders. By using the tools, it will change the approach of the software development process. It is required to find the desire architecture that achieve the goal.

Considering our tool, it should effectively reduce the communication cost. Also, the approach should perform ability to reduce ambiguity in understanding the requirements. In most cases, it is the root cause of expensive debugging costs (i.e., technical debt) in the last stage. So, we must have this ability to ensure consistency towards the goals. Last but not least, we hope the software development process is motivated by testing to ensure the quality of the code.

As per design challenges discussed, we hope the software development process is motivated by testing. For system architecture decision, we can narrow down the scope to Test Driven Development (TDD), Behavior Driven Development (BDD) and Acceptance Test Driven Development (ATDD). These are the popular approaches in software processes which depend more on testing. Briefly describing, TDD is an agile software development strategy that addresses both design and testing, which has emerged as a novel software development approach that involves writing automated unit tests in an iterative Test-First manner [11]. On top of that, it derives ATDD and BDD. ATDD is a method of building a software created based on acceptance test created by customer, developer, and tester and BDD is a Test-First, Agile Testing practice and collaborative process that creates a shared understanding of requirements between the business and the Agile Team [12]. Therefore, it is required to undertake a judicious assessment in following perspective: Effectiveness of reduce communication cost, Ability of reduce ambiguity in understanding the requirements.

Firstly, for TDD, the primary focus is on writing tests before implementing the corresponding code units. This approach ensures that the code units are built with a clear understanding of the intended functionality. In the other world, if developer has some misunderstanding in communication, the unit test also will include the same misunderstanding. It is a technique that builds the correct software, but not building the software that user wants. Consequently, while TDD is effective in building correct software, it may not be the most suitable solution for addressing our specific needs [13].

Secondly, ATDD is promoting user acceptance test. The core concept behind ATDD is to involve users in the software development process, allowing their input to guide the development in the right direction. This approach is primarily focused on testing and ensuring that the developed software meets the users' expectations and requirements [13].

Lastly, BDD emphasizes collaboration between different stakeholders, including developers, testers, and users, to achieve a shared understanding of what the user wants from the software. BDD shifts the focus towards user behavior and aims to align the development efforts with the desired user outcomes. By fostering collaboration and shared understanding, BDD aims to develop software that accurately meets user needs while considering their behavior and expectations.

Therefore, both ATDD and BDD can help to reduce the ambiguity in understanding the requirements. However, compared with ATDD, BDD uses other techniques such as conversations, collaboration, and automation to ensure developers are delivering what the user wants based on a common language where all stakeholders can understand. So BDD has a greater advantage in reducing communication costs than ATDD.

To conclude, both TDD, ATDD and BDD are good approaches to ensure the quality of the delivery. However, TDD may lead to misunderstanding of requirements and ATDD is not good enough to reduce the communication cost. Therefore, we decided to implement BDD in our tools in order to let our users apply this behavior to achieve our goals.

After deciding to apply BDD, we need to think of our technology set surrounding BDD. There are three types of technologies we need to find the most suitable, which are Automated Browser, Descriptive Test-cases Framework and Reporting Tools respectively. To compare which is the best options, we will find out the quality attributes (QAs) according to our goals to measure if the choices are fit to our goals.

A. Automated Browser

Goals

To simulate user behaviors in web applications, an automated browser is required in our tool. There are many options that can be chosen for automatic testing. To find the best options, we can compare the candidate with the following Quality Attributes: compatibility, developer-friendly, usability.

Design Challenges

The foundation of our system evolves around three key aspects: compatibility, developer-friendly and usability. To ensure compatibility, our system requires an automatic browser that is highly compatible. In other words, we want it to support a diverse range of different popular browsers. For example, Google Chrome, Mozilla Firefox, Internet Explorer, etc. Since it is the testing tool for developers, it will be more significant than other systems because the user is the developer from different countries or companies.

Additionally, the system should be developer-friendliness. As modern web applications incorporate multiple programming languages, it is vital that our system accommodates and supports these languages effectively. This ensures that developers can work with ease, utilizing their preferred language in the development process without unnecessary complications.

Lastly, the system should provide robust features and a comprehensive API for developers to develop and customize the tool effectively. It helps to ensure productivity for team members, which can speed up the project progress.

Smart Choice Family and Technology Cards

Selenium [7]

Technology/Automated Browser/Simulating Library

Description: Selenium is a automated browser that for automating web application for testing purposes. On top of that, boring web-based administration tasks can also be automated as well.

Selenium

Consequences:

★ ★ ★ ★ ★ Compatibility

Selenium supports Chrome, Edge, Firefox, Safari and IE, which is the most popular browser in the world. By using Selenium, it can use to test all popular browser.

★ ★ ★ ★ Developer-friendly

Selenium provides many Application Programming Interface(API) for different programming languages. It supports Java, Python, C#, Ruby, JavaScript and Kotlin, which is the most popular programming languages for developing web application. By using Selenium, we can easily implement it in the web application framework such as Django, Cucumber, Nest Js etc.

★ ★ ★ ★ Usability

Selenium provides a web driver for user to perform automated action. It provides many methods for developer to access the sources, for example, Xpath and CSS selector. It undoubtedly increase the usability for user to use.

Cypress

Technology/Automated Browser/Simulating Library

Description: Cypress is a open-source testing tool that provides testing for the modern web acpplication

cypress

Consequences:

★ ★ ★ Compatibility

Cypress only support for Chrome-family browser, WebKit and Firefox.

★ Developer-friendly

Cypress is supporting JavaScript only. Although the documentations is clear, it only can write in JavaScript.

★ ★ ★ ★ ★ Usability

Cypress provides a web driver for user to perform automated action. It provides many methods for developer to access the sources. On top of that, cypress also provide methods for component testing. It supports the major web application framework and provides some API for testing them. For example, React, Angular and Vue.

Puppteer

Technology/Automated Browser/Simulating Library

Description: Puppeteer is a Node.js library which provides a high-level API to control Chrome over the DevTools Protocol. It can runs in headless mode.

Puppeteer

Consequences:

★ Compatibility

Puppeteer only support Chrome & Chromium browser.

★ Developer-friendly

Puppeteer only support JavaScript for development.

★ ★ ★ ★ ★ Usability

Puppeteer provides high-level API for developer to use. On top of that, Puppeteer is highly integrated with Chrome and the DevTools, which mean that it provide a good experiences for testing and debugging in Chrome.

TABLE I
COMPARISON ON AUTOMATED BROWSER.

Design Decision	Driver Points	Comments	Final Choice
Selenium	5+4+4 =13	Selenium is fit the solution in both compatibility, developer-friendly and Usability	Y
Cypress	3+1+5 =9	This is less appropriate for this solution due to less programming language supports	
Puppeteer	1+1+5 = 7	This is less appropriate for this solution due to less browser and programming language supports	

Selenium supports major browsers and various programming languages. These are the advantages that Cypress and Puppeteer did not have. On top of that, Selenium has huge community. From the statistics of npm trends, we can see that Selenium has over 1,500,000 weekly downloads.[10] A huge community helps to resolve difficulties and handle technical debts during development. Also, the API provided by Selenium is enough to simulate the user's behavior. Therefore, we choose Selenium as the core automated browser in our development.

B. Descriptive Test-cases Framework

Goals

Based on the findings from related work, the utilization of descriptive language in testing tools is important. In order to identify a suitable tool for our specific solution, we must evaluate its compatibility with key quality attributes such as readability, usability, debuggability, and operating system (OS) support. These quality attributes play a pivotal role in determining the effectiveness and suitability of the tool for our intended purpose.

Design Challenges

Since it is used for descriptive language, readability takes an important role. Given that these tools will be utilized by various stakeholders, including developers, testers, and clients,

it is imperative to prioritize readability to minimize communication costs and facilitate effective collaboration.

Additionally, the ability to implement a wide range of features to meet the diverse requirements of different types of websites is essential for usability. A highly usable tool should offer flexibility and functionality to cater to a variety of scenarios and use cases. Therefore, usability is important.

Moreover, debuggability plays crucial role in the developing of our tools. As a testing tool, it must possess robust debuggability capabilities to assist users in pinpointing and resolving errors effectively. This is the reason debuggability is a challenge to ensure seamless and efficient bug identification and resolution.

Last but not least, OS support is pivotal factor that is important because different users will use diverse range of operating systems in the real-world user. This compatibility ensures that the tools can be accessed and utilized by a wide range of users, regardless of their preferred operating system.

Smart Choice Family and Technology Cards

Behave

Technology/development tools

Description: Behave is the most popular Python BDD frameworks. It is a semi-official library of implementing Cucumber by using cucumber's components.

Consequences:

★ ★ ★ ★ ★ Readability

Behave make use of Gherkin language, which is a plain-text language with a simple structure, is easily to understand by non-programmers. Thus the readability is high

★ ★ ★ ★ Usability

It can integrate with the major framework such as Django and Flask, and also have a good documentation, which is easily to use and implement. Besides, it is a standalone program, which can be easily used by non-programmer people. Therefore, it can integrate with framework and can as a standalone program if we want, which is high usability.

★ ★ ★ ★ Debuggability

Behave will check if the program is do the task as expected. If something go wrong, it will told you in understandable way.

★ ★ ★ ★ ★ OS support

Behave supports Windows, MacOS and Linux, which is the major OS in the real-world company.

SpecFlow

Technology/development tools

Description: Behave is the most popular Python BDD frameworks

Consequences:

★ ★ ★ ★ ★ Readability

SpecFlow make use of Gherkin language, which is a plain-text language with a simple structure, is easily to understand by non-programmers. Thus the readability is high

★ ★ Usability

It can integrate with .NET applications only, so it needs to install .NET and Microsoft Visual Studio, which required many times and memories to run the program.

★ ★ ★ ★ ★ Debuggability

SpecFlow can integrate with Visual Studio perfectly so it supports debugging the testes. Therefore, we can insert the breakpoint to our test cases. Whenever we execute the generated tests in debug mode, the execution will stop at the specific breakpoint we defined, and we can see the variable that time and thus debug efficiently

★ OS support

SpecFlow only support windows, which is used by some of the company.

Pytest BDD

Technology/development tools

Description: Behave is the most popular Python BDD frameworks

Consequences:

★ ★ ★ ★ ★ Readability

Pytest Bdd implements a subset of the Gherkin language, which is a plain-text language with a simple structure, is easily to understand by non-programmers. Thus the readability is high

★ Usability

The step definition naming requires a defined prexi or suffix, which is more complicated. Also the additional code is required for declaring scenarios, implementing scenario outlines and sharing steps. Which is generally more difficult to use

★ ★ ★ Debuggability

Pytest BDD have the trace about where the steps will go to when trigger error. But it seems no other debugging tools and techniques to use.

★ ★ ★ ★ ★ OS support

Pytest BDD supports Windows, MacOS and Linux, which is the major OS in the real-world company.

TABLE II
COMPARSION ON DESCRIPTIVE TEST-CASES FRAMEWORK.

Design Decision	Driver Points	Comments	Final Choice
Behave	5+4+4+5= 18	Behave is fit to the solution especially the great debuggability to define where the bug placed.	Y
SpecFlow	5+2+5+1=13	This is less appropriate for this solution due to lack of OS supports	
Pytest BDD	5+1+3+5 = 14	This is less appropriate for this solution due to less usability	

As per above analysis conducted, it is evident that Behave, SpecFlow and Pytest BDD are sufficient readable enough since they are all implementing a subset of the Gherkin language, which is a plain-text language with a simple structure, is easily understand by non-programmers. However, from the perspective of documentations and integration, Behave undoubtedly has a distinguished advantage over SpecFlow and Pytest BDD.

Besides, both Behave and SpecFlow offer superior debuggability, which is important in our project for effectively pinpointing the occurrence of problems. However, SpecFlow only supports Windows platform, whereas Behave and Pytest BDD extend their support to major operating systems such as Windows, macOS, and Linux.

To conclude, Behave excels in all evaluated aspects when compared to the other candidates. Consequently, we decided to use Behave for Descriptive test-cases framework, given its exceptional capabilities.

C. Reporting Tools

Goals

Reporting plays a vital role for effective visualization of the results. To ensure the success of our solution, the tools must exhibit the following quality attributes: Compatibility, Automatability and Visualizable.

Design Challenges

In the context of reporting tools, we need to find tools that are compatible for our testing tools mentioned above. It is imperative to identify a solution that is compatible with our aforementioned testing tools.

Furthermore, we seek a reporting tool that can generate reports automatically, particularly for automated testing purposes. Testing reports are the important part for automated testing if it can be created automatically as an integral part of Continuous Integration (CI) pipelines, it should be good to provide real-time feedback to the development team. Also, in order to reduce the communication cost, the ability of virtualization is a must. Virtualization can ease communication between non-developers and developers.

By prioritizing compatibility, automatic report generation, and advanced visualization capabilities, we aim to select a reporting tool that optimizes the effectiveness and efficiency of our testing tool.

Smart Choice Family and Technology Cards

Allure

Technology/Reporting tools

Description: Allure is a automation test reporting tool that designed to create fancy and clear testing reports in minutes.

Consequences:

★ ★ ★ ★ ★ Compatibility

Allure Report supports for almost all major testing library in different languages such as pytest, cucumber.js and behave, which is having high compatibility. Whatever we choose for the testing tools, the allure report can be integrated with our choice.

★ ★ ★ ★ Automatability

Allure reports can be created as an integral part of Continuous Integration (CI) pipelines thus it can make sure the most recent reports are accessible during each test run.

Also the reporting process is made simpler by integration with CI platforms, which also provide the development team with real-time feedback.[X]

★ ★ ★ ★ Visualizability

Generated reports by Allure will be visually appealing with graphical charts. Graphs will help the viewer understand the result in an easily comprehensible way by using Bar Charts or Pie Charts. So even non-technical people involved in the software development process such as Stakeholders and Business Analysts will understand it with ease. But it is not providing some customization for the output reports.

ExtentReport

Technology/Reporting tools

Description: ExtentReports is the library that create a beautiful, interactive and detailed crafted report and real-time analytics.

Consequences:

★ ★ Compatibility

Allure Report supports for Java, .NET and Klov only, which is not supporting many of major testing library. It can be used in SpecFlow

★ ★ ★ ★ Automatability

ExtentReport offers seamless integration with popular build tools and CI servers like Maven,Jenkins and TeamCity, which is enough for taking a part in CI for generating reports.

★ ★ ★ ★ ★ Visualizability

Generated reports by ExtentReports is support many format, such as HTML, Email or even pushing data to MongoDB for Klov. Also, it allows customization of the report by custom CSS or JavaScript via the configuration XML or directly from code. Therefore, the report is customizable and can do what we want.

ReportPortal

Technology/Reporting tools

Description: ReportPortal is an AI-powered test automation dashboard that aggregate and analyze test reports to ascertain release health.

Consequences:

★ ★ ★ ★ ★ Compatibility

ReportPortal supports almost all major testing library in different languages.

★ ★ ★ ★ Automatability

Report Portal can be integrated with bug tracking systems such as Jira and Rally. Also, it can be integrated with Gitlab CI and Jenkins. Last but not least, it provides the investigation of the failure reasons immediately after the test is completed. These real-time analytics are also automatic.

★ ★ ★ Visualizability

ReportPortal configure simple and understandable reports, which is easily understand by users.

TABLE III
COMPARSION ON REPORTING TOOLS.

Design Decision	Driver Points	Comments	Final Choice
Allure Report	5+4+4 = 13	Allure report fulfil all the needs with good documentations	Y
ExtentReport	2+4+5=11	ExtentReport offering a good customization of report but cannot offering good compatibility, which is the important part in our tools	
ReportPortal	5+4+3=12	Report Portal can fulfil some of our needs but	

Allure Report, ExtentReport and ReportPortal demonstrate good automatability that easily to integrated with CI/CD tools such as Jira. They both provide good, visualized reports, but ExtentReport provides more customization on top of that. However, it is important to note that ExtentReport supports a narrower range of programming languages compared to the other candidates.

In summary, both Allure Report and ReportPortal excel in various aspects. However, Allure Report offers more extensive features and superior visualizability, making it the preferred choice for our reporting tools. Therefore, we have decided to utilize Allure Report as our primary reporting solution.

IV. SOLUTION

The following parts will go through our solution architecture including the system decomposition and the adopted design pattern. To illustrate how the solution architecture manifests the earlies set of design decisions, when it applies constraints on implementation, dictates the solution structure and inhibits quality attributes.

A. Implementation of Architectural Decision

Our system should fulfill both the functional and non-functional requirements. For functional requirements, such as our system should automatically test the if a new feature on a web application falls in the scope of user expectation according to the user specified test case and some more detailed functional requirements. While non-functional requirements include compatibility, for example, to be executable on general browsers etc. We present our solution to not only meeting the functional and non-functional requirements, but also the software architecture quality.

Our architectural decision is mainly driven by the non-functional requirements. The rationale will be illustrated in the following.

Addressing Compatibility by Selenium Library

By addressing one of the non-functional requirements – compatibility, we incorporated Selenium into our architectural decision. Our system needs to be compatible with multiple browsers to support different test scenarios. Selenium offers a wide range of browser drivers. The set-up process is very straightforward, just simply calling the corresponding driver getter function, and we could have control of different drivers. The browser driver options include Google Chrome, Microsoft Edge, Mozilla Firefox, Internet Explorer, and Safari. By supporting automation testing on these browsers, we achieve a high coverage of most user browser preferences, ensuring compatibility with a wide range of user choices.

Addressing Testability and Usability by Behave Framework

To meet the non-functional requirement of testability and usability, we have selected Behave framework. Behave framework supports behavior-driven design, allowing the extension of testability into our system. This framework allows us to define clear and understandable behavior scenarios that can be easily mapped to automated test cases. With Behave, we can write tests in a natural language format, making it easier for users and testers to understand and validate the system's behavior. Enhancing the testability of our system by providing a structured approach in writing test cases.

B. System Decomposition

a. Class Diagram

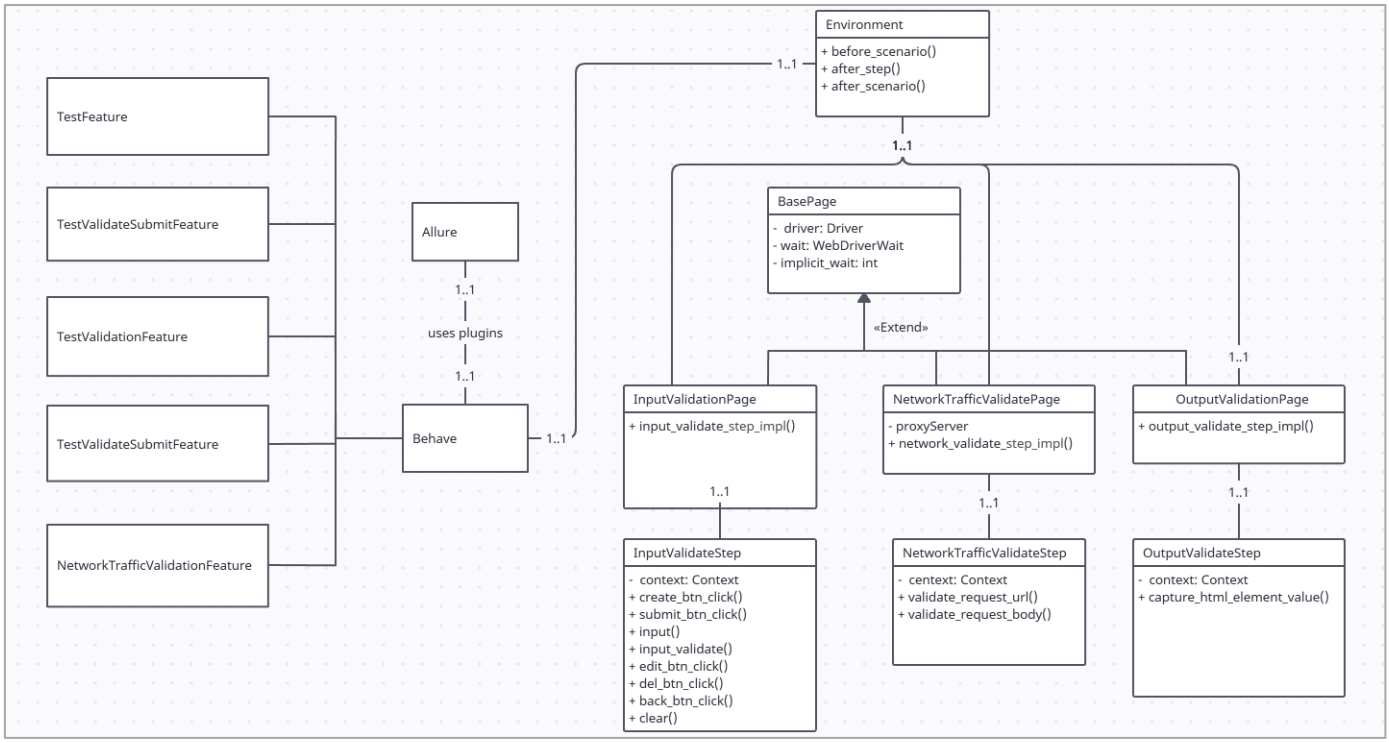

Figure 1: Class Diagram

Our system decomposition is demonstrated in the Figure 1 class diagram. There are a few main characteristics: Implementation of Behavior-Driven Design, Modularity, Implementation of descriptive test case framework and Automatic report generation plugin.

To initiate the implementation of Behavior-Driven Design (BDD), our system relies on the behave framework as its foundation. BDD requires the simulation of user requirements into automated tests that reflect the specified behavior. As shown in the class diagram, we have implemented these behavior test cases in the form of Feature files which contain various types of tests, such as input validation, output validation, and network validation. These tests aim to simulate the desired system behavior as in user requirements.

To illustrate the second characteristic – modularity, we decompose the system into multiple modules. We highlight the presence of three Page classes that are inherited from class Base Page. As a result, they shared the common properties while remaining individual responsibility for its own test page set up. Moreover, each Page set up class associates with a corresponding Step class that contains the component function to perform automatic testing. The system classes are loosely coupling but highly cohesive. That would increase the system extensibility and pave the way for better system maintenance.

Furthermore, our system has implemented a descriptive test case framework in addition to the underlying BDD framework mentioned earlier. By incorporating the descriptive test case framework into our system, the user will be able to input a file containing the regression test cases that are documented in human language and that could be interpreted by our system to perform automated testing. Figure 2 is a demonstration on one of the user descriptive test cases, which will be compiled by our system to automatically replay the step afterward and return the result of the test by next system component – automatic report generation.

```
Feature: Create output cases using Selenium

 @test_Chrome
 Scenario Outline: Create a record with Chrome
     Given : I Click on Create button
     And : I Enter a "<Name>" in Name field
     And : I Enter a "<Email>" in Email field
     And : I Enter a "<Message>" in Message field
     And : I Click on Submit button
     And : I Click on Back button
     And : Check the new input "<Name>", "<Email>", "<Message>"
  Examples:
     |Name|Email|Message|
     |1|1@g.com|1|
```

Figure 2: Descriptive Test Case

b. Design Principles and Code Qualities

Couling and Cohesion

The system design adheres to data coupling and functional cohesion Class associations are based on data structures, where the Step classes are referenced as variables within the Page module. This pattern highlights the data coupling and functional cohesion. Consequently, enabled less interdependency, less coordination, and less information flow between the classes.

Single-responsibility Principle

This principle emphasizes each class should take up its logical responsibility only. Because of this principle, we separated the steps function into various classes instead of a big class full of all step functions. This principle emphasizes each class should take up its logical responsibility only. Because of this principle, we separated the steps function into various classes instead of a class full of all step functions. For example, our system design divides the step class by responsibility. The `InputValidateStep` class will only be responsible for carrying out the functions related to validating the input.

Open-closed Principle

The Open-closed principle requires our code pattern to meet two conditions: Open for extension and Close for modification.

Our code pattern adheres to this principle by incorporating a base class – Base page and a base interface – Step Registry. The base page is the base set up of the test scenarios, which are then inherited by the sub set-up classes. While the Step Registry is the built-in base interface form the behave package. It is very easy to extend another function module in the future, such as Media Validation. In such case, the developers only need to extend the media validation page set up module from the base page set up module, and implement the media validate step with an annotation with Step Registry. Aligning with the OCP rule that is to extend without modifying the existing code or refining the implementation without changing the functional purpose.

c. Software Qualities

Reusability

Our system design considers one of the software qualities, that is reusability. The reuse strategy encourages less duplication and generalization of the base module. In our system design, we adopted inheritance to create a general interface or base class. Thus, the other modules could just be implemented or inherited from the base class to acquire those common methods or common characteristics. The module reusability aspect of our system design has yielded notable advantages in terms of simplified maintenance and decreased duplication.

Stability

A stable architectural design minimizes the impact of changes on other modules. An unstable design would require modifying N-1 modules when changing one module, as the design decision depends on N modules. In our design, we addressed the stability quality by adopting the design pattern, loose coupling and highly cohesive. For example, the Step modules and Page modules in our system design are not tight coupling, instead, they are associated with each other only, that is they are highly cohesive. This design approach ensures that making changes to one Step module in the future will not impact the other Step modules or the overall Pages setup. Instead, it will solely affect its associated Page module. Therefore, our system design remains stable and resilient to changes.

Composability

The design is highly composable. It is easy to compose a higher-level component from supportive components. To perform a full feature test, our system design requires the following components: Feature file, Page module and corresponding Step module. The Page module and Step module are Plug-and-play components. To build a new function of our system, the developers just need to build a new Page module and a new Step module. Our design demonstrates a high level of composability, making it effortless to construct higher-level components using supportive components.

Simplicity

In our system design, we embrace the principle of simplicity and prioritize a clean and concise approach. We have chosen the widely used and popular framework called "behave" to achieve this. It is a clean and concise framework. All that is required is a Feature file, and the behave framework will automatically guide each scenario to the appropriate page setup. The page set up associated with its corresponding step module containing the automation step function. A clean and simple design is better than many customized solutions in terms of maintenance efforts.

C. Example Walkthrough

Sequence Diagram

Figure 3 is a sequence diagram illustrating the procedures of one of the system's use cases – automatic output validation. The use case scenario will be as in the next section.

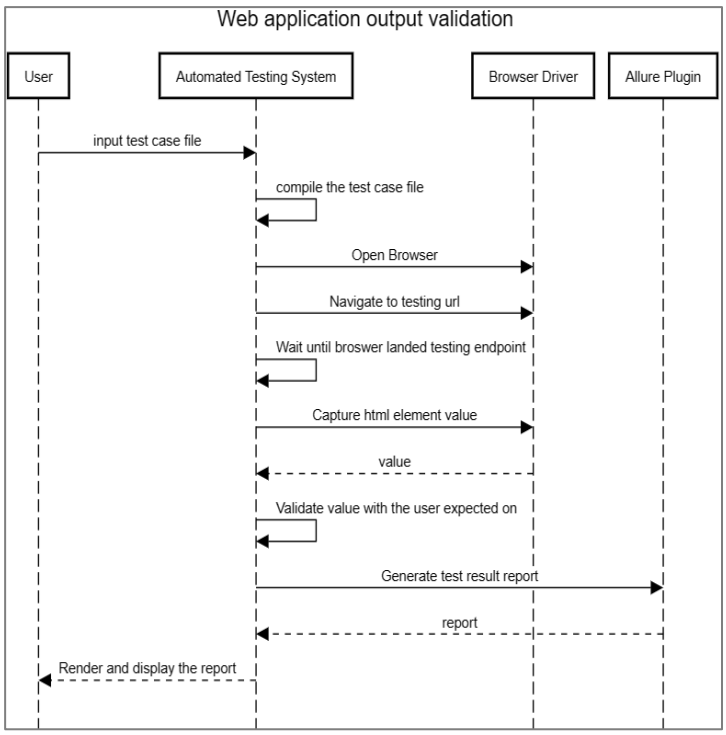

Figure 3: Sequence Diagram of Use Case - Automatic Output Validation

Use Case Scenario

A quality assurance engineer needs to test the functionality of a web application's new features. One of them is a new registration form. He would like to have automatic testing on validating the new form output. Hoping the automatic testing function can ease his workload.

Step breakdown

Step 1	He will input a file documenting the test case procedures to our system.
Step 2	Our system will compile and understand the test case procedures.
Step 3	System automatically opens browser and navigates to the testing endpoint.
Step 4	Input the registration form with user specified values after browser driver has landed on testing endpoint.
Step 5	Capture the registration form output value.
Step 6	Compare the captured output value with user expected value.
Step 7	Generate test result report by allure plugin.
Step 8	Render and display the report result to user.

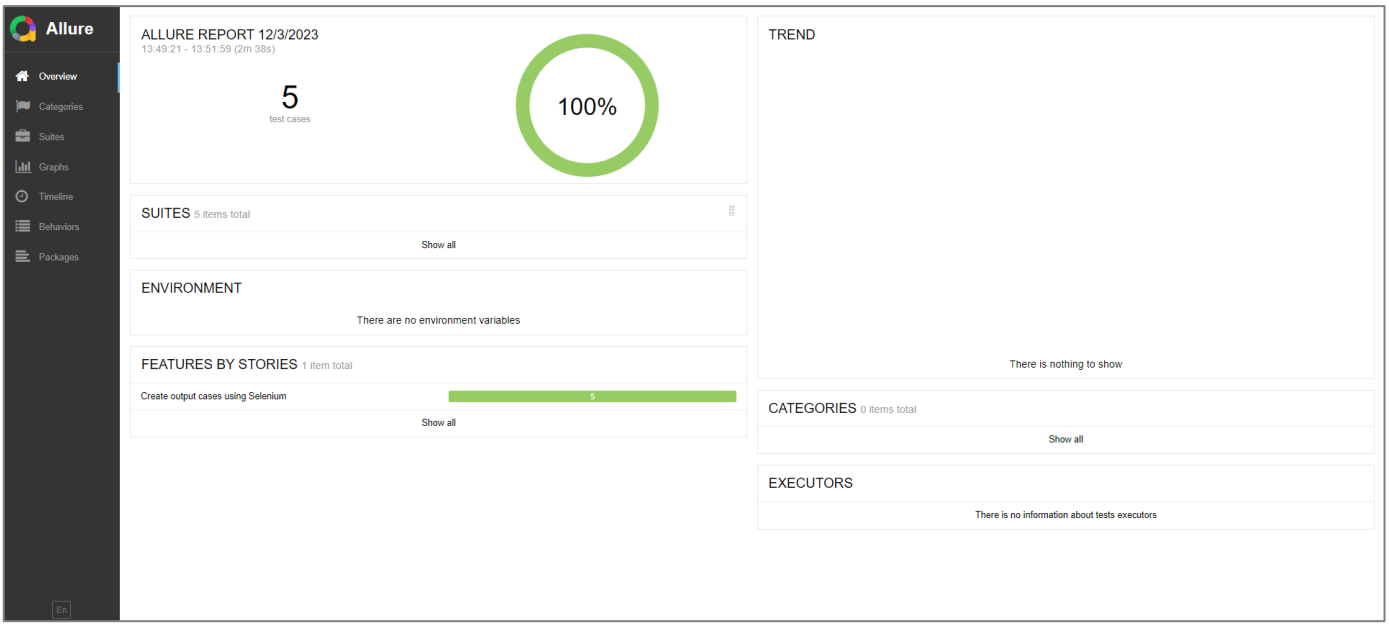

Figure 4: Report Generated by Our System.

Figure 5: Test Case Metrics Generated by Our System.

Use Case Output

As mentioned in Section III, there will be a features file generated that will be used in the testing process. This features file serves as input for the testing tools and is created based on user-provided input in the form of a templated CSV file.

When the testing tools are executed, our program automatically generates or updates the test file to generate the corresponding features file. This program has been specifically developed to address the lack of support for generating feature files within the Behave library. Figure 6 illustrates the actual input provided by the user, and Figure 7 shows the resulting output features file generated by our program.

Subsequently, the test cases are executed based on the user-provided input. If there are no test cases available, the testing process will skip the corresponding test case and notify the developer or tester to develop the test case according to the user requirements. By automating the generation of the features file and facilitating the execution of test cases, our solution simplifies the testing process and enhances collaboration between developers and testers.

After the steps from testing, there will be a report generated to indicate the test case result. Figure 4 and Figure 5 show the actual output report generated by our system. The report includes the following elements.

Element	Value
Timestamp	11/2/2023 22:40:52 - 22:45:08 (4m 16s)
Suites	Showing the details of each test cases, including the test result status, test case title, test input, test body, and duration.
Environment	The adopted environment variables.
Features by Stories	The details of each test case in the new feature.
Test Case Metrics	Figure 4 shows the test case metrics. Including the statistic indicating the percentage of succeeded test case.

```
Feature: Create output cases using Selenium
        @test_Chrome
        Scenario Outline: Create a record with Chrome
                Given : I Click on Create button
                And : I Enter a "<Name>" in Name field
                And : I Enter a "<Email>" in Email field
                And : I Enter a "<Message>" in Message field
                And : I Click on Submit button
        Examples:
                |Name|Email|Message|
                |"Fergusssss"|"test@example.com22"|"test message"|

        @test_Chrome
        Scenario Outline: Delete a record with Chrome
                Given : I Click on Delete button with the "1"-th row
        Examples:
                |Name|Email|Message|
                |"Fergusssss"|"test@example.com22"|"test message"|

        @test_Firefox
        Scenario Outline: Create a record with Firefox
                Given : I Click on Create button
                And : I Enter a "<Name>" in Name field
                And : I Enter a "<Email>" in Email field
                And : I Enter a "<Message>" in Message field
                And : I Click on Submit button
        Examples:
                |Name|Email|Message|
                |"Fergusssss"|"test@example.com22"|"test message"|

        @test_Edge
        Scenario Outline: Create a record with Edge
                Given : I Click on Create button
                And : I Enter a "<Name>" in Name field
                And : I Enter a "<Email>" in Email field
                And : I Enter a "<Message>" in Message field
                And : I Click on Submit button
        Examples:
                |Name|Email|Message|
                |"Fergusssss"|"test@example.com22"|"test message"|

        @test_Chrome
        Scenario Outline: Edit a record with Chrome
                Given : I Click on Edit button with the "1"-th row
                And : I Clear Name field
                And : I Clear Email field
                And : I Clear Message field
                And : I Enter a "<Name>" in Name field
                And : I Enter a "<Email>" in Email field
                And : I Enter a "<Message>" in Message field
                And : I Click on Submit button
                And : I Click on Back button
                And : Check the "1"-th row "<Name>", "<Email>", "<Message>"
        Examples:
                |Name|Email|Message|
                |"Fergusssss"|"test@example.com22"|"test message"|
```

Figure 7: The Generated Feature File.

	Features	Scenario	Examples	Pre-step	Another Step	Name	browser
2	TestCSV2	Create a record with Chrome	Name is "Fergusssss" AND Email is "test@example.com22" AND Message is "test message"	I Click on Create button	I Enter a "<Name>" in Name field And I Enter a "<Email>" in Email field And I Enter a "<Message>" in Message field And I Click on Submit button	Create output cases using Selenium	Chrome
3	Test2	Create a record with Firefox	Name is "Ferguszz" AND Email is "test@example.com22" AND Message is "test message"	I Click on Create button	I Enter a "<Name>" in Name field And I Enter a "<Email>" in Email field And I Enter a "<Message>" in Message field And I Click on Submit button	Create test cases using Selenium	Chrome
4	TestCSV2	Delete a record with Chrome	Name is "Fergusssss" AND Email is "test@example.com22" AND Message is "test message"	I Click on Delete button with the "1"-th row		Create output cases using Selenium	Chrome
5	TestCSV2	Create a record with Firefox	Name is "Fergusssss" AND Email is "test@example.com22" AND Message is "test message"	I Click on Create button	I Enter a "<Name>" in Name field And I Enter a "<Email>" in Email field And I Enter a "<Message>" in Message field And I Click on Submit button	Create output cases using Selenium	Firefox
6	TestCSV2	Create a record with Edge	Name is "Fergusssss" AND Email is "test@example.com22" AND Message is "test message"	I Click on Create button	I Enter a "<Name>" in Name field And I Enter a "<Email>" in Email field And I Enter a "<Message>" in Message field And I Click on Submit button	Create output cases using Selenium	Edge
7	TestCSV2	Edit a record with Chrome	Name is "Fergusssss" AND Email is "test@example.com22" AND Message is "test message"	I Click on Edit button with the "1"-th row	I Clear Name field And I Clear Email field And I Clear Message field And I Enter a "<Name>" in Name field And I Enter a "<Email>" in Email field And I Enter a "<Message>" in Message field And I Click on Submit button And I Click on Back button And Check the "1"-th row "<Name>","<Email>","<Message>"	Create output cases using Selenium	Chrome

Figure 6: Example Segment of CSV File.

V. SOFTWARE PROCESS

For this project, we have adopted Scrum as our software development model. Scrum is an agile project management framework known for its flexible and iterative approach to software development and project delivery. Scrum emphasizes the development process on delivering valued features early and consistently, enables teams to effectively respond to changing requirements and deliver incremental value to customers.

In our project, the product backlog implemented using a web application called Teamwork [8], served as a repository for a prioritized collection of features, enhancements, and bug fixes that encompassed the project's requirements and tasks. The owner was responsible for managing and prioritizing the items within the product backlog.

According to the project scope, the project was structured into a series of 3 Sprints, which offered a consistent rhythm for development, testing, and review activities. Additionally, we established a weekly scrum meeting to ensure effective coordination among team members. During these meetings, team members shared updates on their progress, discussed any challenges encountered, and collaboratively planned for the upcoming tasks.

During Sprint 1, which spanned from Weeks 2 to 5. There are several crucial activities that were carried out to advance the project. The priority was to develop user stories, enabling a clear understanding of the project's functional requirements from the user's perspective. Additionally, the software architecture was sketched out to address the non-functional requirements, ensuring that the software solution would meet the desired compatibility, testability and usability. Sprint planning occurred during this period, where the development team collaborated to identify and prioritize the tasks to be completed during the Sprint. This planning process ensured that the team had a clear roadmap and set realistic goals for the upcoming weeks. Furthermore, the software architecture was continuously refined and optimized based on insights gained during the development process. This iterative approach allowed for the incorporation of feedback and the adaptation of the architecture to meet evolving project needs. The tasks of Sprint 1 was listed on Table IV and the Gantt Chart on Figure 8 showed the progress of the tasks in Sprint 1.

TABLE IV
DETERMINED TASKS IN SPRINT 1.

1	Develop user stories and user requirements.
2	Determine software architecture to address non-functional requirements, code, test, review.
3	Determine workday for each story and prioritize product backlog.
4	Determine the method to identify, track, or resolve technical debts.
5	Setup the demo application.
5.1	Create a demo web application for automated testing.
5.2	Create an API for the demo web application.
5.3	Configure the database setting for web application.
5.4	Configurate connection between web application and API.
5.5	Create Automation Testing Tool Base.

Task 1 focused on developing user stories and gathering user requirements to ensure an understanding of the desired functionality and user expectations. Task 2 involved determining the software architecture taking into consideration non-functional requirements. This included conducting reviews to ensure the quality of the software solution. Task 3 aimed to determine the workday for each user story and prioritize the product backlog, ensuring efficient task management and delivery. Task 4 involved identification, tracking, and resolving any technical debts that arose during the development process, ensuring the software's long-term stability and maintainability. Task 5 focused on setting up a demo application to showcase the functionality of the software. This included several sub-tasks, such as creating a demo web application for automated testing with utilizing Node.js and React (Sub-task 5.1), developing an API server to support the web application by leveraging JDK (Java Development Kit) and Apache Maven (Sub-task 5.2), configuring the database settings (Sub-task 5.3), and establishing the necessary connection between the web application and the API (Sub-task 5.4). To streamline this process, Docker was employed as a platform, allowing for deployment and management of the database, while a MySQL image was utilized as the database instance. The demo web application will be setup a base structure for the project automation testing tools. It involved setting up a Selenium project skeleton using Python programming language (Sub-task 5.5). In task 5, these technologies and tools contributed to the successful setup and integration of the demo application, ensuring its efficient functionality and reliable performance for the demo application.

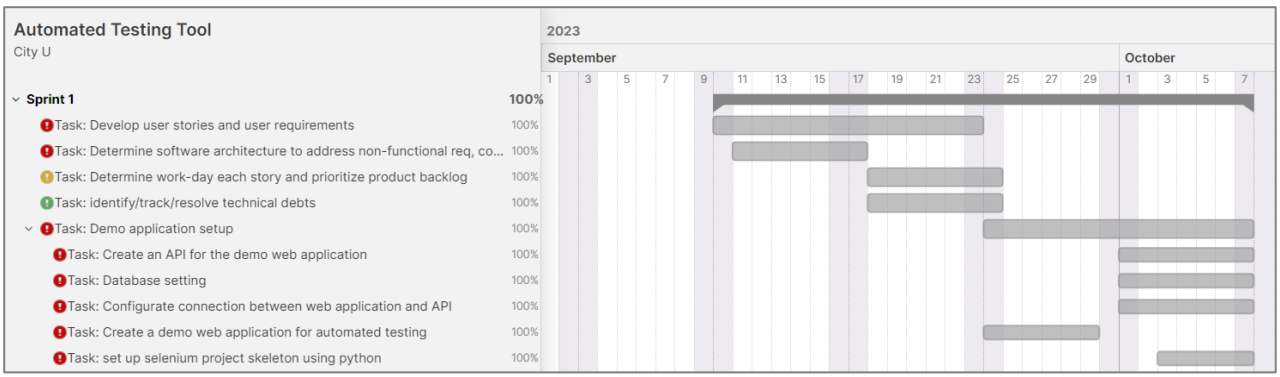

Figure 8: Gantt Chart of Sprint 1.

In brief of Sprint 1, several key tasks were completed. User stories and requirements were developed, and the software architecture was determined. A demo web application and API were created, and the database was configured using Docker and MySQL. An automation testing tool base was established with Selenium in Python. Sprint 1 achieved the initial steps necessary for the project's development, setting the stage for further enhancements and feature implementation in the upcoming Sprints.

TABLE V
USER STORIES.

1	User can test the web application automatically.
2	User can use descriptive language to describe the steps of test.
3	User can use different browsers for automated test.
4	User can verify all relevant aspects of the web application's output, including data and interactions.
5	User can assert error message exist after input field error.
6	User can view report of automated test result.
7	User can specify test case examples in a CSV file for feature file.

The tasks of Sprint 2 were listed on Table VI and the Gantt Chart on Figure 9 showed the progress of the tasks in Sprint 2.

TABLE VI
DETERMINED TASKS IN SPRINT 2.

1	Develop feature to add a record into database through web application.
2	Add a submit message to demo front page.
3	Automate of network request and response validation.
4	Use Behave BDD to implement and map the code.
5	Configure and initialize of web browsers for automation testing.
6	Conduct code quality review.
7	Add the feature of auto-generate report.
8	Add the feature of record the test steps execution result with screenshot.
9	Add input validation to demo web application.
10	Add output validation to demo web application.
11	Add coding for read data from a CSV file and updating feature files.

In the Sprint 2, which spanned from Weeks 6 to 9. The focus will be on implementing additional features and functionalities based on the backlog. The development process involved coding and reviewing the implemented functionality of this project.

User stories are an essential component of Agile and Scrum methodologies for software development which serve to capture and communicate the requirements and needs of the end-users or stakeholders. In our project, we have created and documented multiple user stories, which are listed in Table V for clarity of purpose.

During Sprint 2, the project team worked on a diverse range of key tasks to advance the project. Task 1 was dedicated to developing a robust feature that would enable seamless addition of records into the database through the web application. Task 2 involved enhancing the front page of the demo application submit message feature interaction. Task 3 focused on automating the comprehensive testing of network request and response validation. Task 4 was centered around leveraging the power of the Behave BDD framework to implement and map the code. Task 5 encompassed the critical configuration and initialization of web browsers, enabling seamless automation testing. Task 6 involved conducting a thorough code quality review, meticulously assessing the codebase to ensure adherence to best practices and maintainability. Task 7 revolved around the implementation of an automated report generation feature, enabling the effortless generation of detailed reports encompassing essential test results and metrics. Task 8 focused on meticulously recording test steps execution results, accompanied by screenshots, providing valuable documentation and visual evidence for debugging and troubleshooting. Task 9 and Task 10 were dedicated to developing robust input and output validation mechanisms for the demo web application, respectively. Lastly, Task 11 involved enhancing the automation feature to read data from a given CSV file and dynamically update the feature files, enhancing data integration capabilities and flexibility.

Figure 9: Gantt Chart of Sprint 2.

In summary of Sprint 2, the project team demonstrated a proficiency in accomplishing the designated tasks, leading to substantial advancements in enhancing the functionality and augmenting the automated testing software. The project timeline adhered to the anticipated schedule, without any occurrences of untimely interruptions.

In Sprint 3, which spanned from Weeks 10 to 13. The project focus will be on completing outstandings tasks remaining in Sprint 2 first. After tasks in Sprint 2 are completed, the maintenance process of the automated testing tool is started which involves testing, debugging and maintenance support. Simultaneously, we started to prepare our presentation, video demonstration and report. The tasks of Sprint 3 are listed on Table VII and the Gantt Chart on Figure 13 showed the progress of the tasks in Sprint 3.

TABLE VII
DETERMINED TASKS IN SPRINT 3.

1	Maintenance support
1.1	Add feature that read from examples from CSV file and update the examples value in our features.
1.2	Develop feature to edit a record.
1.3	Develop feature to delete a record.
1.4	Improve demonstration web page.
2	Testing
2.1	Test different browsers
2.2	Test the input and output flow.
2.3	Test the output validation with different browser.
3	Debugging.
4	Video Demo.
5	Presentation.
6	Report writing.

During Sprint 3, the team focused on the key final tasks to accomplish and wrap up the whole project. Sprint 3 focuses on a range of crucial tasks aimed at further enhancing the project's functionality and overall performance. Task 1 was conducted the maintenance support to optimize demonstration of testing tool and demo web application. There are 4 sub-tasks in Task 1. Task 1.1 was focused on developing the feature to enable the testing tool to read examples from a CSV file and update the corresponding values which is an outstanding task in Sprint 2 [Table VI - Task 11]. Task 1.2 was developing a feature to enable the editing of records, empowering users to modify existing data. Task 1.3 was work on implementing a feature that allows for the deletion of records. Task 1.4 was conducted the improvement of the demo web application to enhance user experience and engagement. After we implemented Task 1 successfully, we got a consummated test case for demonstration which can add, edit, and delete record on the demo web application in a single feature scenario. Simultaneously, we conducted extensive testing and debugging to ensure a robust and error-free application (Task 2 and Task 3). In testing, we use concurrency testing strategy to test our tool. We run our test cases multiple times to see if there are any errors occur. There are 3 sub-tasks in Task 2. Task 2.1 is automation tool testing with a specific focus on different web browsers. Task 2.2 was conducted the testing of input and output flow reliability handling. Task 2.3 was conducted the output validation testing across different browsers to ensure consistency and reliability. However, some problems were found during the testing process. For example, we found that there is a compatibility issue with Internet Explorer. Due to the discontinuation of standalone Internet Explorer support by Selenium as of June 2022 [9], our testing tool is unable to function on Internet Explorer. Also, there is an error on transforming user specified csv file into examples of feature file. Our code is not able to translate into multiple example columns. Furthermore, during our testing process, we identified abnormal behavior when editing a record using Mozilla Firefox. Specifically, upon editing a record and subsequently navigating back to the home page using the back button, we observed that the newly edited record was saved in an abnormal manner. As a result, we have prioritized the resolution of these bugs as Task 3 in our development efforts. We updated our csv transformation code to support generating multiple example columns on feature files. For the abnormal editing behavior on Mozilla Firefox, we identified a specific line of code within our web application that is responsible for triggering the error. This line of code is depicted in Figure 10. The onclick function triggered a save request and an edit request. So, there will be a new record saved after we edited a record and subsequently navigating back to the home page. To address this issue, we implemented a fix by modifying the navigating function and replacing it with a hyperlink. The modified code snippet is illustrated in Figure 11.

```
<div className="button-container">
    <button className="back-button" onClick={()=>navigate("/")}><span>Back</span></button>
    <button className="submit-button" type="submit"><span>Submit</span></button>
</div>
```

Figure 10: Code Segment in Web Application – Original.

```
<div className="button-container">
    <Link to={"/"}><button className="back-button"><span>Back</span></button></Link>
    <button className="submit-button" type="submit"><span>Submit</span></button>
</div>
```

Figure 11: Code Segment in Web Application – Modified.

Lastly, we prepared the video demo and a presentation to showcase the project's results and achievements (Task 4 and 5). In Task 6, we write a report to present our project outcomes and development progress.

The project team utilizes a burndown chart to track progress. It visually represents remaining work versus working time in the Sprints, allowing the project team to monitor progress towards Sprint goals. Overall, the Burndown Chart is shown in Figure 12.

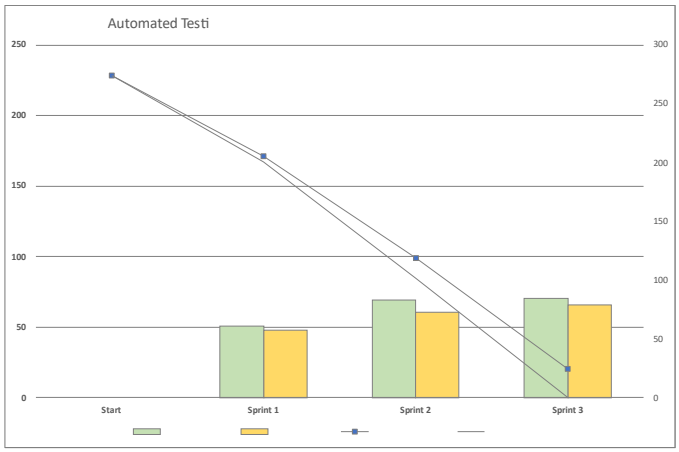

Figure 12: Burndown Chart.

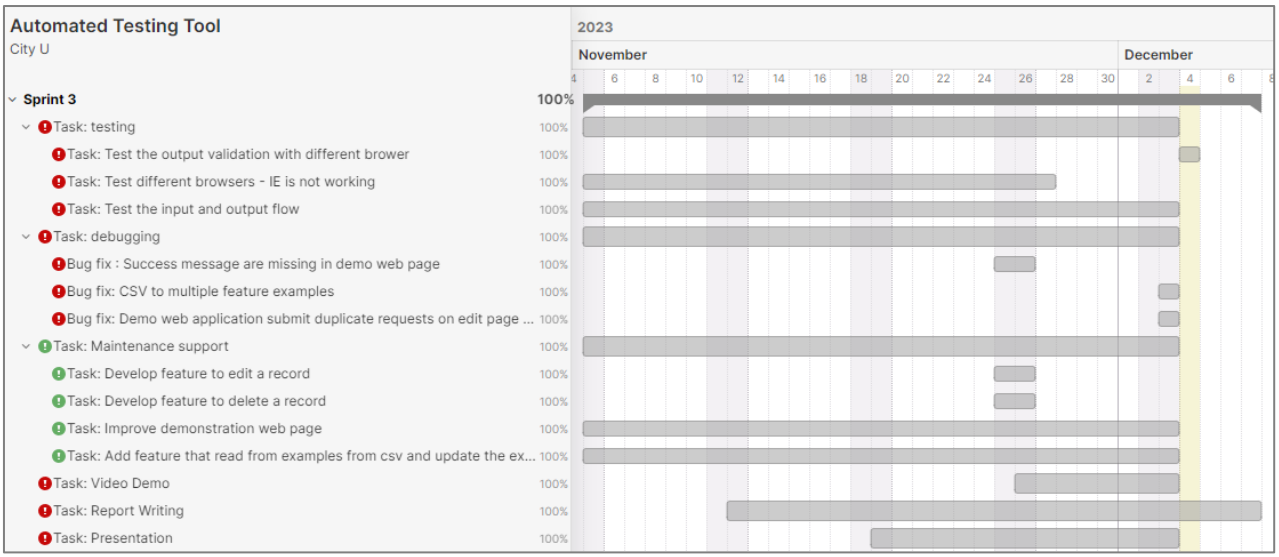

Figure 13: Gantt Chart of Sprint 3.

Over the course of 3 Sprints, the project team has made significant progress in enhancing the functionality, usability, and testing capabilities of the project. Throughout these Sprints, the project team demonstrated proficiency in implementing important features, troubleshooting compatibility issues, and ensuring the reliability and accuracy of the project.

VI. EVALUATION

In the evaluation phase, we conducted an exhaustive assessment of our proposed solution to determine its effectiveness in addressing the problem outlined in the report. We verified and evaluated various aspects of our solution.

Demo Web Application Error

To evaluate our testing tool on dealing with error web application, we simulate an error on our web application. application. Specifically, we removed the submit button on the create page, as depicted in Figure 14.

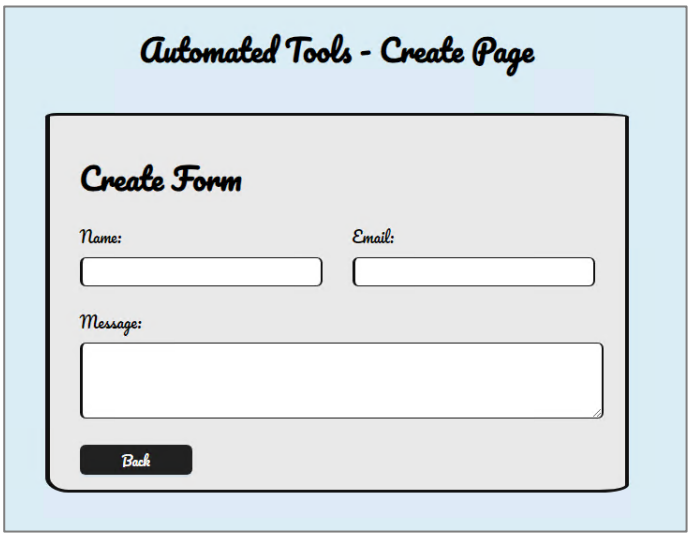

Figure 14: Removal of Submit Button on the Create Page.

Next, we executed a normal testing scenario using our testing tool. During the testing process, the tool detected the absence of the submit button and terminated the testing execution at line 9, as illustrated in Figure 15.

```
1  Feature: Create test cases using Selenium
2
3      @test_Chrome
4      Scenario Outline: Chrome Test
5          Given : I Click on Create button
6          And : I Enter a "<Name>" in Name field
7          And : I Enter a "<Email>" in Email field
8          And : I Enter a "<Message>" in Message field
9          And : I Click on Submit button
10         And : I Click on Back button
11         And : I Click on Edit button with the "1"-th row
12         And : I Clear Name field
13         And : I Clear Email field
14         And : I Clear Message field
15         And : I Enter a "<Name2>" in Name field
16         And : I Enter a "<Email2>" in Email field
17         And : I Enter a "<Message2>" in Message field
18         And : I Click on Submit button
19         And : I Click on Back button
20         And : Check the "1"-th row "<Name2>", "<Email2>", "<Message2>"
21         And : I Click on Delete button with the "1"-th row
22     Examples:
23         |Name|Email|Message|Name2|Email2|Message2|
24         |Chrome|chrome@example.com|I'm a Chrome|Chrome2|chrome2@example.com|I'm a Chrome2|
```

Figure 15: The Terminated Code in the Test Case Sample.

Lastly, the generated testing report, shown in Figure 16, will show the problem screenshot to users and show the detailed error message for developers which facilitates the understanding between users and developers.

Overall, our evaluation demonstrates that our testing tool effectively identifies and reports errors within web applications.

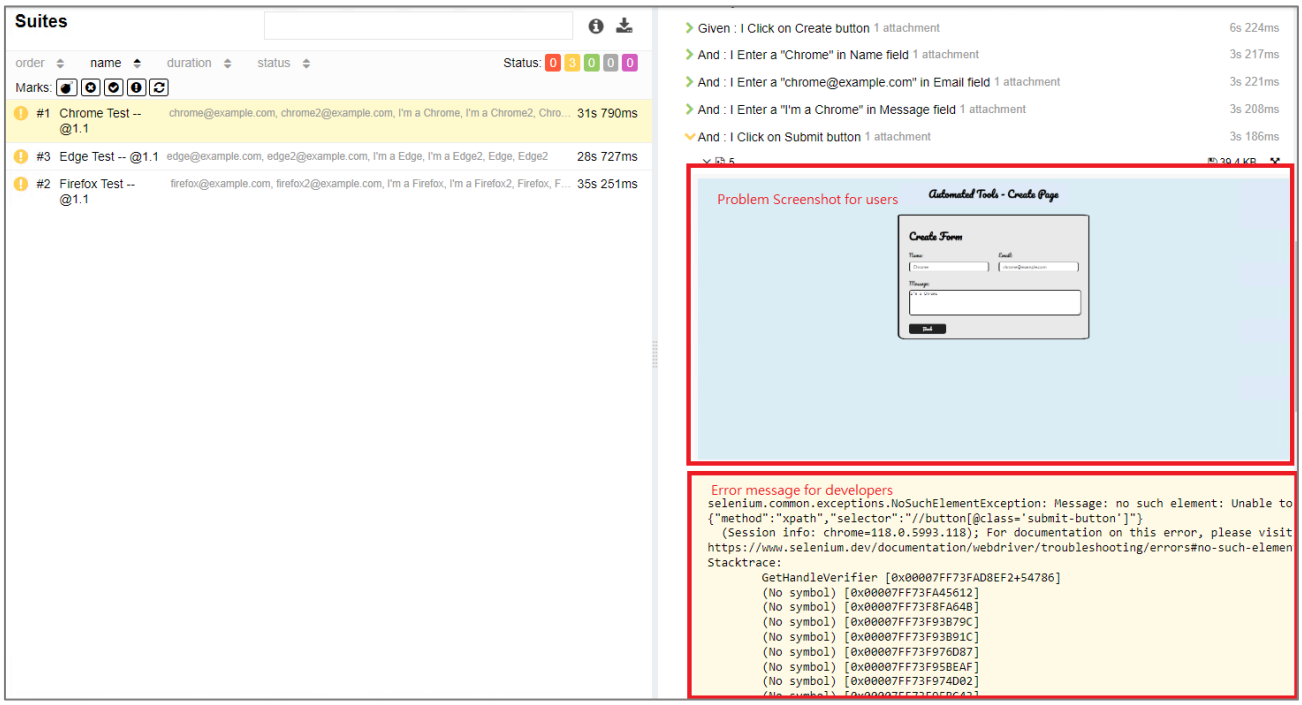

Figure 16: The Generated Report with Terminated Code.

Input Validation

In the evaluation of input validation, a critical aspect to consider is the accurate display of error messages on the website's user interface. The interactive element significantly impacts the overall User Experience. Therefore, it is necessary to validate whether the error message is appropriately shown to users, effectively indicating the erroneous input fields, and preventing the submission of erroneous data to the backend server. In our evaluation, as illustrated in Figure 17, we examined the error message display functionality.

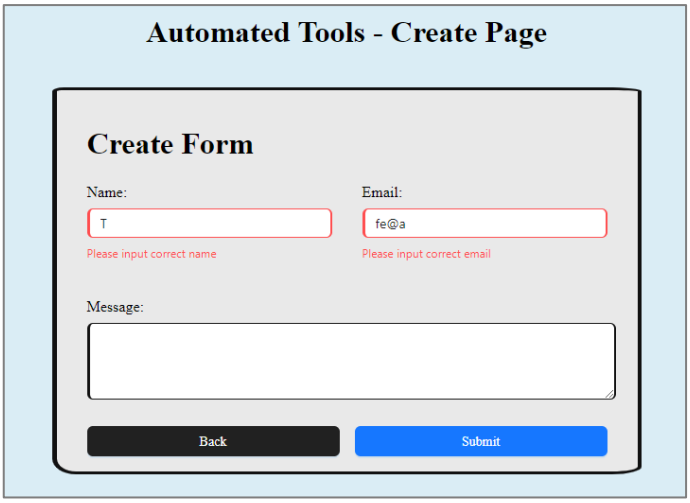

Figure 17: Error Messages in Web Application.

Output Validation

During the evaluation of output validation, it is necessary to ensure the accurate input and expected output from the user interface of the website.

When a new record is added, the program compares the actual output with the expected output specified in the user input. If the new record does not match the given input, the program asserts the discrepancy and reports the error.

Additionally, the output validation process encompasses the examination of various interactions, such as Edit and Delete actions involving different buttons. The test cases are designed to validate and assert the correctness of these interactions. For the Edit action, the test cases validate whether the modified items are accurately updated in the system. Similarly, during the Delete action, the test cases ensure that the intended items are removed from the system as expected, resulting in the desired output.

TABLE VIII
OUTPUT VALIDATION ACCEPTANCE CRITERIA TABLE.

Criteria	Expected Results	Actual Results
1	When a new record is added, the system must compare the actual output with the expected output specified in the user input.	As expected.
2	Test cases should be designed to validate and assert the correctness of these interactions, ensuring that the expected output is achieved.	As expected.
3	During the Edit action, the system must update the edited items correctly and reflect the changes in the output.	As expected.
4	During the Delete action, the system should remove the intended items as expected, resulting in the desired output.	As expected.

Network Traffic Validation

As outlined in the user requirements in Sprint 1, our system will have a function that is to validate the network traffic. This function will be responsible for ensuring the health of the required traffic by verifying the accuracy of the request and response body. This section is the evaluation of this feature. This feature includes the following acceptance criteria:

Given the user has specified the network traffic of his web application should include a pair of request and response, to a specified endpoint, with specified HTTP protocol, request body and response code. If the network traffic does not match the expected conditions, the system will report an error. In vice versa, if the network traffic satisfies the conditions, the system should report it as a successful match.

Evaluation

We have implemented a set of test cases to simulate the user behavior. The test data is as follows:

Test case 1

1. Test Nature: Positive test case
2. Test Data:
    a. API: http://localhost:8000/forms
    b. HTTP protocol: POST
    c. Request Body: Name = Tester, Email = tester@outlook.com message = I'm testing.
    d. Response code = 201
3. Expected Result: System will be able to capture this pair of request and response, then report to user it is a successful match.

Test case 2

1. Test Nature: Negative test case
2. Test Data:
    a. API: http://localhost:8000/test-wrong-link
    b. HTTP protocol: POST
    c. Request Body: Name = Tester, Email = tester@outlook.com message = I'm testing.
    d. Response code = 201
3. Expected Result: System will be able to find out this pair of request and response is not matched (because of the API is wrong), then report to user there is a failed test case.

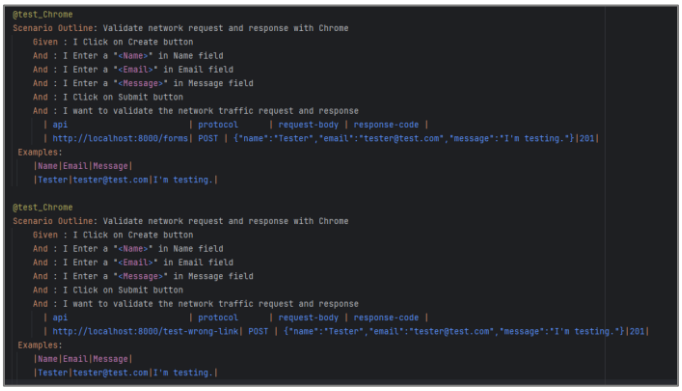

Figure 18: Setup of Test Case 1 and Case 2.

Result

The acceptance criteria are passed when both test cases resulted as expected. The system reported test case 1 is passed and told user there is a problem in user's web application. Below figure shows our application captured a missing network request and reported to the user, indicating there is a problem in the user's web application.

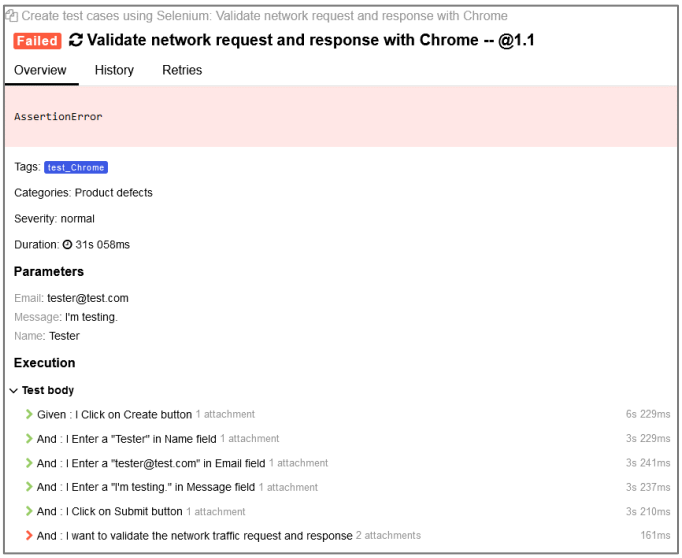

Figure 19: Automatic Report of Error.

CSV to Feature File

In order to evaluate the functionality of our CSV to Feature file conversion feature, we conducted several scenarios to simulate user behavior and input. These scenarios include:

1. Multiple Features in a Single CSV File.
2. Empty Fields in CSV File.
3. Empty Rows in CSV File.

For scenario 1, we tested the capability of our system to handle multiple features within a single CSV file. By providing a CSV file as shown in Figure 20, we successfully generated two features, namely TestCSV2 and Test2. The generated feature files contained the appropriate values, as illustrated in Figure 21 and Figure 22.

	A	B	C	D	E	F	G
1	Features	Scenario	Examples	Pre-step	Another Step	Name	browser
2	TestEmpty	Create a record with	Name is "Fergusssss"	I Click on Create but	I Enter a "<Name>"	Create output cases	Chrome
3							
4	TestEmptyPass	Edit a record with Ch	Name is "Fergusssss"	I Click on Edit buttor	I Clear Name field A	Create output cases	Chrome

Figure 20: The Segment of CSV Sample.

```
Feature: Create output cases using Selenium

        @test_Chrome
        Scenario Outline: Create a record with Chrome
                Given : I Click on Create button
                And : I Enter a "<Name>" in Name field
                And : I Enter a "<Email>" in Email field
                And : I Enter a "<Message>" in Message field
                And : I Click on Submit button
        Examples:
                |Name|Email|Message|
                |"Fergusssss"|"test@example.com22"|"test message"|


        @test_Chrome
        Scenario Outline: Delete a record with Chrome
                Given : I Click on Delete button with the "1"-th row
        Examples:
                |Name|Email|Message|
                |"Fergusssss"|"test@example.com22"|"test message"|


        @test_Firefox
        Scenario Outline: Create a record with Firefox
                Given : I Click on Create button
                And : I Enter a "<Name>" in Name field
                And : I Enter a "<Email>" in Email field
                And : I Enter a "<Message>" in Message field
                And : I Click on Submit button
        Examples:
                |Name|Email|Message|
                |"Fergusssss"|"test@example.com22"|"test message"|
```

Figure 21: The Generated Feature File – TestCSV2.feature.

```
Feature: Create test cases using Selenium

        @test_Chrome
        Scenario Outline: Create a record with Firefox
                Given : I Click on Create button
                And : I Enter a "<Name>" in Name field
                And : I Enter a "<Email>" in Email field
                And : I Enter a "<Message>" in Message field
                And : I Click on Submit button
        Examples:
                |Name|Email|Message|
                |"Ferguszz"|"test@example.com22"|"test message"|
```

Figure 22: The Generated Feature File – Test2.feature.

For scenario 2, we examined the ability to handle empty fields in the CSV file, shown in Figure 23. To evaluate this feature, we prepared a test CSV file with an empty field, such as the absence of "Another Step" in a Delete test. Our system successfully processed this CSV file and generated the corresponding Feature file, accommodating the empty field as expected, shown in Figure 24.

```
Feature: Create output cases using Selenium

        @test_Chrome
        Scenario Outline: Create a record with Chrome
                Given : I Click on Create button
                And : I Enter a "<Name>" in Name field
                And : I Enter a "<Email>" in Email field
                And : I Enter a "<Message>" in Message field
                And : I Click on Submit button
        Examples:
                |Name|Email|Message|
                |"Fergusssss"|"test@example.com22"|"test message"|


        @test_Chrome
        Scenario Outline: Delete a record with Chrome
                Given : I Click on Delete button with the "1"-th row
        Examples:
                |Name|Email|Message|
                |"Fergusssss"|"test@example.com22"|"test message"|


        @test_Firefox
        Scenario Outline: Create a record with Firefox
                Given : I Click on Create button
                And : I Enter a "<Name>" in Name field
                And : I Enter a "<Email>" in Email field
                And : I Enter a "<Message>" in Message field
                And : I Click on Submit button
        Examples:
                |Name|Email|Message|
                |"Fergusssss"|"test@example.com22"|"test message"|
```

Figure 24: The Generated Feature File with Empty Field.

For scenario 3, we also tested the system's capability to handle CSV files with empty rows. By providing a CSV file containing empty rows, illustrated in Figure 25, we verified that our system properly generated the corresponding Features file, accounting for the empty rows, the result shown in Figure 26.

	Features	Scenario	Examples
2	TestCSV2	Create a record with Chrome	Name is "Fergusssss" AND Email is "test@example.com22" AND Message is "test message"
3	Test2	Create a record with Firefox	Name is "Ferguszz" AND Email is "test@example.com22" AND Message is "test message"
4	TestCSV2	Delete a record with Chrome	Name is "Fergusssss" AND Email is "test@example.com22" AND Message is "test message"
5	TestCSV2	Create a record with Firefox	Name is "Fergusssss" AND Email is "test@example.com22" AND Message is "test message"
6	TestCSV2	Create a record with Edge	Name is "Fergusssss" AND Email is "test@example.com22" AND Message is "test message"
7	TestCSV2	Edit a record with Chrome	Name is "Fergusssss" AND Email is "test@example.com22" AND Message is "test message"

Figure 25: The Segment of CSV Sample with Empty Rows.

```
        @test_Chrome
        Scenario Outline: Delete a record with Chrome
                Given : I Click on Delete button with the "1"-th row
        Examples:
                |Name|Email|Message|
                |"Fergusssss"|"test@example.com22"|"test message"|
```

Figure 26: The Generated Feature File with Empty Rows.

TestCSV2	Delete a record with Chrome	Name is "Fergusssss" AND Email is "test@example.com22" AND Message is "test message"	I Click on Delete button with the "1"-th row		Create output cases using Selenium	Chrome

Figure 23: The Segment of CSV Sample with Empty Field.

Through these evaluation scenarios, we demonstrate that our CSV to Feature file conversion feature performs effectively and accurately handles various input scenarios. The resulting feature files align with the expected format and contain the appropriate values.

VII. CONCLUSION

In conclusion, this paper presents our proposal of the automated testing tool for web applications using descriptive test cases. We have developed an automated testing tool that adopts a Behavior-Driven Development (BDD) approach for web applications. The automated tool utilized Selenium WebDriver and Behave features to enable automated testing across various browsers.

As part of our future work, we intend to incorporate Continuous Integration and Continuous Deployment (CI/CD) principles to integrate automated testing into the overall development pipeline. This integration will enable developers and testers to efficiently incorporate automated testing into their workflows, ensuring reliable and efficient testing processes.

The automated testing tool effectively emulates the user input, performs input validation, validates network requests and responses, and captures screenshots at each step of the test cases. These screenshots serve as valuable visual aids for generating comprehensive testing reports. By adopting this automated testing tool, developers and testers can ease the testing process and improve overall efficiency. The tool's comprehensive testing reports provide enhanced visibility and analysis of test results, allowing for effective assessment and validation of the software under test.

Overall, our automated testing tool presents a robust and efficient solution for automating the testing of web applications.

VIII. REFERENCES

[1] A. Uddin & A. Anand, "Importance of Software Testing in the Process of Software Development," *Int. J. for Scientfic Research Development*, Jan. 2019.

[2] Enlyft, “Software Testing Tools Products,” Software Testing Tools, https://enlyft.com/tech/software-testing-tools (accessed Nov. 12, 2023).

[3] L.A. Cisneros, C.I. Reis & M.Maximiano, "An Experimental Evaluation of ITL, TDD and BDD," *Icsea 2018: Thirteenth Int. Conf. Software Eng. Advances*, pp. 20–24, Oct. 2018.

[4] M.R. Girgis, T.M. Mahmoud, B.A. Abdullatif & A.M. Zaki, "An Automated Web Application Testing System," *Int. J. Comput. Applications*, vol. 99, no. 7, pp. 37–44, Aug. 2014.

[5] S. Gojare, R. Joshi & D.Gaigaware, "Analysis and Design of Selenium WebDriver Automation Testing Framework," *2nd Int. Symp. Big Data Cloud Comput. (isbcc’15)*, vol. 50, pp. 341–346, 2015.

[6] P. J. Stevens, “Understanding the Differences Between BDD & TDD,” Cucumber.io, https://cucumber.io/blog/bdd/bdd-vs-tdd/ (accessed Nov. 18, 2023).

[7] S. Raghavendra, Python Testing with Selenium: Learn to Implement Different Testing Techniques Using the Selenium WebDriver, United States: Apress, 2021,

[8] Teamwork Company, “The only all-in-one platform for client work,” Teamwork, https://www.teamwork.com/ (accessed Oct. 1, 2023).

[9] Selenium Development Team, “IE Specific Functionality,” Selenium WebDriver, https://www.selenium.dev/documentation/webdriver/browsers/internet_explorer/ (accessed Dec. 2, 2023).

[10] Software Freedom Conservancy, “Selenium-webdriver,” npm trends, https://npmtrends.com/selenium-webdriver (accessed Nov. 28, 2023).

[11] D. S. Janzen and H. Saiedian, "On the Influence of Test-Driven Development on Software Design," *19th Conference on Software Engineering Education & Training (CSEET'06)*, Turtle Bay, HI, USA, 2006, pp. 141-148, doi: 10.1109/CSEET.2006.25.

[12] A.C. Barus, "The implementation of ATDD and BDD from Testing Perspectives," *1st Int. Conf. Advance Scientific Innovation (ICASI)*, Apr. 2018.

[13] T.B.T. Le, "BDD in Agile Testing: An Experimental Study," *Da Nang Publishing House*, pp. 130–136, 2020.

[14] M. Irshad, R. Britto & K.Petersen, "Adapting Behavior Driven Development (BDD) for Large-scale Software Systems," *J. Syst. Software*, vol. 177, Mar. 2021.

IX. STUDENT BIOGRAPHY