

Design of a Website Testing Suite with automated reporting and issue tracking

CS907 Dissertation Project

Dissertation Report

Hongjin Chen

Supervisor: Dr. Jonathan Foss

Department of Computer Science

Abstract

In the contemporary software development landscape, quality assurance and testing stand out as the linchpins of successful project delivery. With technological advancements surging at a rapid pace and escalating consumer demands for seamless digital experiences, optimizing software functionality and performance is paramount. However, many developers, especially novices and those engaged in smaller-scale projects, find software testing daunting. They often grapple with its intricacies or lack the requisite skills and resources for effective testing.

This project emerged as a solution to bridge this gap. A user-centric Web Application UI testing platform was introduced, aiming to furnish users with a streamlined yet potent testing tool. This initiative simplifies UI testing, negating the need for deep dives into intricate technical nuances. Drawing inspiration from the **Gherkin language**, the platform merges technical jargon with everyday vernacular, making test script creation both intuitive and descriptive. Such an amalgamation not only democratizes the testing landscape but also amplifies test quality and accuracy, ensuring optimal end-user experiences.

For students in software engineering courses and nascent enterprises, this platform holds immense promise. It doubles as a practical learning tool, fostering deeper comprehension of software testing's significance and methodologies. Concurrently, it offers businesses a cost-effective, high-performance solution, carving out a competitive edge in a saturated market.

This research adopted an iterative design approach, commencing with an exhaustive analysis of existing automated testing tools. Leveraging insights from this analysis, the project meticulously designed and birthed a website, subjected to rigorous user and performance tests. Adhering to user-centered design principles, multiple optimization cycles were steered by feedback from prospective users. On a technical front, the solution embodies the classical **MVC architecture**: the frontend is orchestrated with **React.js** and **Redux** for seamless component development and unified state management, supplemented by **Material-UI** for aesthetic UI consistency, and **Axios** manages data exchanges between the frontend and backend. The backend is architected with **Python** complemented by the **Flask** framework, with **Selenium** automating the testing procedures and **MySQL** ensuring robust data storage, all overseen via **phpMyAdmin**.

The developed platform is anchored around four pivotal features:

1. **Sturdy automated test cases**, powered by **Selenium**, streamline test design and execution.
2. **Real-time HTML test report generation** fosters collaboration and swift issue redressal.
3. A **holistic task management system**, beyond mere task tracking, links directly with pertinent test outcomes.

-
4. Expansive user and team management facets assure fluid registration, team assembly, and inter-team synergies.

In summary, this project aims to provide small development teams and independent developers with a more convenient and efficient testing platform, significantly reducing their learning costs in UI functional testing. The platform offers students a practical and highly applicable learning experience. The project will continue to pay attention to user needs and constantly optimize and upgrade itself.

Keywords: UI testing platform , User-centric Web Application , Automated testing tools ,Software development ,Selenium, UI testing, React.js, Flask

Acknowledgements

I would like to thank my supervisor Dr Jonathan Foss for his professional advice and guidance throughout the project. In addition, I would like to thank all the students and friends who participated in the testing and provided design input in the early stages. Finally, I would like to thank my family for all the support they have given me this year.

Contents

Abstract	ii
Acknowledgements	iv
List of Figures	viii
List of Tables	ix
List of Algorithms	ix
1 Introduction	1
2 Background Literature Review	3
2.1 Evolution from Manual to Automated Testing	3
2.2 Testing frameworks	3
2.3 Testing web-based applications	4
2.4 Gherkin Language and its Influence on GWT language	6
3 Results	8
3.1 Project Overview	8
3.2 Technology stacks	9
3.3 Creation and Execution of Automated Test Cases	11
3.3.1 Design of GWT Language	11
3.3.2 Implementation of GWT Language	13
3.3.3 Interpreter	17
3.4 Test report	18
3.4.1 Test Report Generation	19
3.4.2 Storing Test Reports	19
3.4.3 Displaying Test Reports	20
3.5 Implementation of Other Basic Website Features	22
3.5.1 User	22
4 Project Management	32
4.1 Project Management Approach	32
4.2 Code Management	32
4.3 Operational Overview and Deployment	32
4.3.1 Server Specifications	32
4.4 Project Access	32
4.5 Limitations on deployment	33

5	Test	34
5.1	User Test	34
5.1.1	Design	34
5.1.2	Feedback	34
5.2	Stress Test	36
5.2.1	Testing Environment and Procedure	36
5.2.2	Analysis of Stress Test Results	36
5.2.3	Potential Causes	38
6	Appraisal and reflection	39
6.1	Change in project conceptualisation	39
6.2	Interpreting feedback and reflecting	39
6.3	Technical challenge	39
7	Ethics	41
7.1	Data Protection and Privacy	41
7.2	Potential Misuse of Testing Tools	41
8	Conclusions	42
8.1	Future Development	42
	Appendices	47
A	List of user behaviour implementations in Selenium	47
B	Locator List	48
C	Action mapping list	48
D	API document	48
D.1	User Management	48
D.2	Team Management	48
D.3	Task Management	48
D.4	Test Case Management	49
D.5	Report Management	50
E	Building a Docker Image	50
E.1	Running the Docker Container	50
E.2	Accessing the Application	50
F	User Feedback Questionnaire	50
F.1	System Usability Scale (SUS)	51
F.2	Open-Ended Questions	51

G	Invitation to Participate in Testing Evaluation	51
G.1	Testing Details	52
G.2	Module Highlights	52
G.3	Feedback Requested	52
H	Questionnaire results table	52
I	Stress test result table	52

List of Figures

2.1	Screenshots of test cases for this project	5
2.2	Gherkin language example	6
3.1	Web development structure	10
3.2	Database structure	12
3.3	Test case page	15
3.4	Given function	18
3.5	When and Then function	18
3.6	Test Run Function	19
3.7	Test report page	20
3.8	Select and send reports to team members	21
3.9	Test report emails in the target's mailbox	21
3.10	Login and register page	23
3.11	User center page	23
3.12	The interface to create a team	24
3.13	Group space component	24
3.14	Group page	25
3.15	Task management page	25
3.16	Add new task list and add new tasks	26
3.17	Task management	26
3.18	Test case management	27
3.19	A test case may correspond to multiple TestEvent objects	28
3.20	Team Management page	28
3.21	Add new members	29
3.22	Transfer Management Rights	29
3.23	Delete team	29
3.24	Tutorial example	30
3.25	User guideline	30
3.26	User guideline dialog	31
5.1	Configuration of Stress Test	36
5.2	Average Response Time vs Concurrent Users	36
5.3	Maximum Response Time vs Concurrent Users	37
5.4	Standard Deviation of Response Time vs Concurrent Users	37
5.5	System Throughput vs Concurrent Users	37

List of Tables

2.1	Testing frameworks table	3
3.1	Function List	8
3.2	Database model table	11
3.3	Given-When-Then Meaning	12

3.4	GWT language table	14
A.1	List of user behaviour implementations in Selenium	47
B.1	Locator List	48
C.1	Action mapping list	48
D.1	User Management APIs	49
D.2	Team Management APIs	49
D.3	Task Management APIs	49
D.4	Test Case Management APIs	50
D.5	Report Management APIs	50
I.1	Stress test result table	53

1 Introduction

Software testing, a pivotal element in successful software development projects, often poses significant challenges, especially for newcomers or those tackling smaller projects. Conducting thorough manual user interface (UI) functionality testing leads to a significant amount of work, much of which is repetitive. Meanwhile, using most of the testing tools available on the market requires users to have programming knowledge, as these tools largely rely on a foundational understanding of coding and an in-depth grasp of web development.

In the face of these challenges, there arises a critical hypothesis: Designing an intuitive testing language module and a drag-and-drop programming interface would be more user-friendly and make writing tests easier. This hypothesis has been the guiding principle of this project. If proven true, it implies a paradigm shift in how testing platforms should be designed, catering more to those without a technical background.

After delving into the aforementioned challenges, this project is dedicated to crafting a user-centric web application UI testing platform. The primary pursuit is to streamline the testing automation process, potentially economizing software development expenses. The platform encourages all users, irrespective of their technical prowess, to design test cases based on user activities. In line with this, the project introduced the Given-When-Then (GWT) language: an intuitive testing language module. With the graphical interface, users can effortlessly build test cases by dragging and dropping predefined GWT modules, sidestepping traditional coding hassles. This pioneering idea embodies the project's core ethos. Additionally, to enhance teamwork, the system incorporates a Kanban system, promoting effective issue management during testing and enhancing team synergy.

A pivotal feature of this platform is its design ethos, greatly influenced by the GWT language. Although unique, it borrows inspiration from the acclaimed Gherkin language of Behavior-Driven Development (BDD). Gherkin shines for its innate use of natural language constructs that aptly depict software behavior. Modeling the testing language after Gherkin's philosophy, the platform aims for a simplified test creation and execution approach. A deeper dive into the Gherkin language and its relevance will be expounded in the following section.

This strategic decision in language design not only seeks to lower entry barriers but also aims to bolster the site's appeal and user engagement. The intricacies of this design and its potential ramifications will be discussed further in the ensuing design chapter.

The platform primarily caters to university students enrolled in courses like software engineering and web development, and to smaller enterprises desiring to craft their management websites. For software engineering scholars, this platform stands as an invaluable educational tool, enriching their grasp of software testing's significance and methodologies. Simultaneously, for smaller enterprises inexperienced in

UI testing, this platform, with its outstanding user-centric design, presents immense value.

2 Background Literature Review

This section delves deep into the landscape of automated testing, examining prevalent tools, methodologies, and languages. The focus lies on their strengths, weaknesses, and potential impact on UI testing.

2.1 Evolution from Manual to Automated Testing

The evolution of automated testing has significantly addressed the inherent challenges of manual testing. While manual testing requires executing test cases without automated tools or scripts[12], which can be laborious and inefficient, automated testing empowers testers to craft repeatable and reusable scenarios [11]. This distinction becomes particularly pronounced in functional UI testing, the central theme of this research.

Functional testing falls under the umbrella of black-box testing. Here, testers are guided by program specifications to design and execute tests, aiming to ensure that the software aligns with its specified functional requirements and fulfills the user's expectations[3].

2.2 Testing frameworks

Several testing frameworks and tools, such as **TestComplete**[19], **Selenium**[4], and **Watir**[5], have emerged as catalysts making testing more streamlined than traditional manual methods. However, utilizing these tools mandates testers to possess a robust technical acumen, spanning knowledge of programming languages, testing frameworks, and tool mastery.

Name	Types of applications supported	Supported browsers	Supported languages	Operating system	Permission to use
TestComplete	Desktop applications, web applications and mobile applications	IE, Firefox, Google chrome	VBScript, JScript, Python and DelphiScript	Microsoft windows	Payed use
Selenium	Webapplication	Chrome, Firefox, Safari and Edge	Java, C#, Python and JavaScript	Cross platform	Open source
Watir	Webapplication	Chrome, Firefox, Safari and Edge	Ruby, Python and c#	Cross platform	Open source

Table 2.1: Testing frameworks table

Central to this project, **Selenium** stands out as a multifaceted platform offering an expansive set of functionalities. Its in-depth features can address myriad user interactions. However, this doesn't negate the significance of introducing the **GWT (Given-When-Then)** language into the testing landscape. The primary motivation behind designing the **GWT** language was to create a more intuitive representation of user interactions. While the **GWT** language finds its roots in existing testing languages, it's tailored to encapsulate a broader spectrum of user behaviors and seeks continuous enhancement.

Selenium, with its comprehensive feature set, is undeniably powerful. But there's a trade-off: when the objective is to cultivate a language that prioritizes

speed, simplicity, and user-friendliness, some of the intricate functionalities may be sidelined. In such scenarios, the emphasis is on crafting a syntax that resonates with users through its readability and intuitive structure, even if it means letting go of certain complexities.

2.3 Testing web-based applications

While contemporary tools such as **Selenium** have indisputably facilitated monumental progress in the domain of testing efficiency, they are not exempt from inherent challenges. Web-based applications, notably **Uilicious**[20] and **Metersphere**[14], which capitalize on the robust capabilities of **Selenium** for the purpose of automated UI testing, invariably manifest certain limitations.

Using **Uilicious** as an example, instead of relying strictly on specific locator mechanisms, it uses an intelligent approach to match elements based on descriptive attributes. At first glance, this method seems to align well with how humans browse. However, a challenge arises when multiple elements match the given description. In such cases, the system automatically selects the first match. This can be seen in commands like `I.fill("Search Bar", "Running Shoes");`, which can lead to inaccuracies when the descriptor, whether it's an **ID**, **Class Name**, or **Name**, points to multiple similar elements in an HTML document. To address this, I'm proposing a more detailed location system where users specify both the type of locator and its related descriptor. I will explain the details of this new approach in the following section.

Compared to other formats, the structure of the **GWT** language stands out for its clarity. When dealing with complex test cases, its design, influenced by the **Gherkin** language, is noticeably more organized and straightforward. Here's a typical example from **Uilicious**. When compared with **GWT**, it provides a cleaner and clearer representation, as shown in Figure 2.1.

```
// Visit the e-commerce site
I.goTo("https://ecommerce-example.com");

// Log in
I.click("Login");
I.wait(2); // wait 2 seconds to make sure the login
    ↪ dialogue is open
I.fill("Email Field", "user@example.com"); // Wait 2
    ↪ seconds to make sure the login dialogue has opened.
I.fill("Password Field", "securepassword"); I.click("
    ↪ Submit"); //Wait 2 seconds to make sure the login
    ↪ dialogue has opened.
I.click("Submit"); // Check if the login is successful.
```

```
// Check if the login was successful
I.see("Welcome back, user!"); I.click("Submit"); // Check
    ↪ if login was successful.

// Search for products
I.see("https://ecommerce-example.com/goods"); // Search
    ↪ for products.
I.see("");
I.fill("Search Bar", "Running Shoes"); // Search for
    ↪ products.
I.click("Search Button"); // Select the product and watch
    ↪ the price.

// Select the product and watch the price
I.click("Product: Nike Running Shoes"); I.see("$100"); I.
    ↪ see("Running Shoes")
I.see("$100"); // Select the product and watch the price.
```



Figure 2.1: Screenshots of test cases for this project

Compared to **Metersphere**, which has a robust set of features ideal for managing large projects over time, its web interface can be somewhat complex. This complexity means users need to spend more time learning how to navigate and understand its features. Additionally, **Metersphere** requires users to be very precise when creating test cases, detailing each step, from actions to expected results. This also means users need to be familiar with specific technical terms. On the other hand, my project focuses on UI automation testing and has a simpler and more

user-friendly interface. This allows users to start testing quickly, while **Metersphere** often requires a longer adjustment period. While **Metersphere** is proven and reliable due to its extensive real-world use, I believe my project will become equally reliable as it continues to develop.

In summation, while **Metersphere** provides a holistic testing platform, my project offers enhanced efficiency and user experience in certain specific domains.

2.4 Gherkin Language and its Influence on GWT language

The **Gherkin** language has a significant role in the field of software development, especially in **BDD** [17]. It is known for its clear and straightforward language, which makes it easier to describe software behavior without getting lost in technical details. This makes **Gherkin** accessible to a wide audience, including both experts and beginners in the field.

What sets **Gherkin** apart is its plain-text representation of software features and scenarios, systematically organized in the '**Given-When-Then**' format. When contrasted with the descriptive testing language of platforms like **Uilicious**, which embraces a more linear approach, **Gherkin** emerges as a multi-layered testing language. This hierarchical structure, anchored in the '**Given-When-Then**' steps, segments user actions into three specific phases. Such a segmentation not only clarifies user intent and the subsequent reactions to each action but also promotes the assembly of complex test cases.

To provide a clearer picture, below is a demonstration of the **Gherkin** language exemplifying a software feature and its related scenario:

```
Scenario: Pop element from a stack
  Given a non-empty Stack
  When the stack has N elements
  And element E is on top of the stack
  Then a pop operation should return E
  And the new size of the stack should be N-1
```

Figure 2.2: Gherkin language example

Instead, if natural language is used, the corresponding test cases might be as follows:

"Imagine a stack that is not empty. This stack has a known quantity, N , representing the total number of items within it. At the top of this collection is an item labeled 'E'. When a specific action, termed '*pop*', is invoked, the expected result should be the retrieval of this 'E' item. After this action, a recount of the items within the stack should show a decrease by one, resulting in $N - 1$."

From the above example, it's clear that compared to natural language descriptions, the **Gherkin language**, as an acceptance testing technique, offers a more precise and accurate source of customer requirements. Considering that 85% of defects in developed software stem from vague, incomplete, or misunderstood requirements[17], it's vital to use an appropriate testing language to describe test cases. With this in mind, this project aims to design a tool that can describe requirements in a clearer, more comprehensive, and accurate manner.

3 Results

3.1 Project Overview

With the rapid advancement of technology, building a comprehensive and efficient automated testing platform has become an essential means for enterprises to enhance productivity and ensure software quality. This section aims to provide a detailed overview of the platform's core features and their pivotal roles in the modern testing process. As illustrated in Table 3.1, the platform offers a comprehensive set of features.

Function Category		Function name
User	User Registration and Login	Register new account
		Login with existing account
	User Information Management	View and edit personal information
		Change password
Team	Team Creation and Management	Create new team
		View and manage team information
Task	Task Creation and Management	Create tasks within a team
		View team task list
		Delete tasks and task lists
Test Case	Test Case Creation and Management	Create and save test cases
		Test Script generation
		Test Script execution
	Test Report Creation	HTML report generation
		HTML report sending
	Test Report Management	Retrieve test reports
		View test reports

Table 3.1: Function List

1. Automated Test Cases

The platform is centered around automated test cases. By integrating the Selenium tool, a stable and efficient automated testing environment is provided for users. This feature allows testers to create and execute precise test scripts without manual intervention, greatly enhancing work efficiency and accuracy. Opting for Selenium as the test execution tool not only ensures the stability of the testing process but also gains endorsement from many leading automated testing platforms in the industry. The drag-and-drop test case creation interface significantly simplifies user interaction. Compared to traditional coding methods, this interface reduces operational complexity, making the testing process more streamlined and intuitive. Additionally, the modular "block" combination strategy is designed to reduce errors caused by manual input. In essence, thanks to this intuitive drag-and-drop operation, users can quickly construct and adjust test scenarios, markedly boosting efficiency.

2. Test Report

This feature emphasizes the clear presentation of testing results and the importance of team collaboration. The platform supports the rapid generation of test reports in HTML format and facilitates effortless sharing with other team members. Through this function, teams can stay updated in real-time on the testing progress, ensuring that potential issues are identified and resolved promptly to guarantee project quality.

3. Task Management

The task management module is dedicated to enhancing team collaboration efficiency. Its functions serve not just as a conventional "to do list" but also directly link to relevant test reports, assisting teams in pinpointing issues and taking swift actions. This module contributes to improved communication and collaboration among team members, ensuring the continuity and stability of the project.

4. User and Team Management

While the functionalities of user and team management might seem basic in comparison, they play an indispensable role in maintaining a smooth workflow. The user module offers an intuitive mechanism for user registration, login, and information management. Simultaneously, the team module supports the creation, management, and collaboration of team members, further enhancing inter-team communication and cooperation.

In summary, the platform integrates a range of advanced features designed to meet the growing demands of modern software testing teams. Through careful design and implementation, I am confident that the platform will provide users with an automated testing experience that is efficient, reliable and compliant with industry standards.

I have prepared a demo video. Through this video, you can get an intuitive understanding of the website's functions, dynamic styles, and interactive features. Below is the link to the video: [Click to view the demo video.](#)

3.2 Technology stacks

This project uses the traditional MVC (Model-View-Controller) architecture, with a variety of technologies and frameworks used for front-end and back-end development.

- **Frontend**

In the client-side development portion of the project, *React.js* [16] was chosen as the main framework due to its component-based development approach, which is not only considered scalable but also easy to maintain. User login states are centrally managed using *Redux*, ensuring not only data consistency but also providing a recognized unified state management solution, simplifying

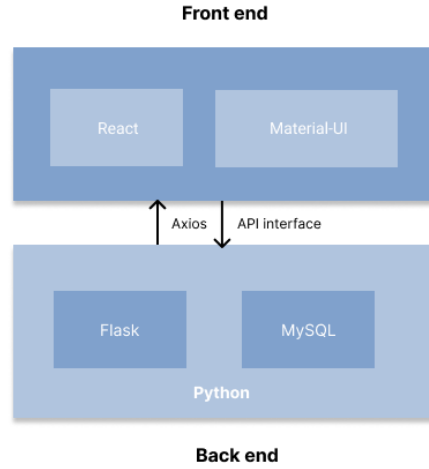


Figure 3.1: Web development structure

the tracking of user login statuses and permissions. In terms of interface design, *Material-UI* [13] was selected mainly because it offers a consistent and streamlined user experience. *Axios* [2] was utilized for the communication and data transfer between the front-end and back-end.

- **Backend**

For the backend development of the project, *Python* [9] was chosen for its robust standard library and simplified syntax. The *Flask* framework [15], a lightweight and adaptable *Python* web framework, was utilized to craft stable and efficient web applications. *Selenium*, a well-known testing framework, was used for automation testing to ensure the software's quality and performance.

In addition to this, the detailed API documentation is already in the appendix (see Section D), which is divided into five tables.

- **Database**

MySQL [6] serves as the primary database system for data management. Whether operated locally or during deployment, the data and structural adjustments are managed using the *phpMyAdmin* [8] tool.

To better understand our database structure and the role of each data model, the table below (Table 3.2) provides a brief description of the functions and key attributes of each table.

Where necessary, the **CASCADE** deletion option is used on fields, such as those involving `user_id`. This ensures that when a user is deleted, all associated records are also appropriately removed, preserving the integrity of the database.

Model Name	Description/Role	Key Attributes
User	Represents a user in the system.	User ID, Username, Email, etc.
User Contribution	Tracks user contributions over time.	Contribution ID, User ID, etc.
Team	Represents a team entity.	Team ID, Name, Manager, etc.
User Team Relationship	Describes relationship between user & team.	Team Member ID, User ID, Team ID.
Test Event	Specific testing event.	Test Event ID, Name, Created By, etc.
Test Case	Individual test case data structure.	Test Case ID, Type, Subtype, etc.
Test Case Element	Specific element of a test case.	Element ID, Type, Subtype, etc.
Test Report	Report of a test event.	Report ID, Test Event ID, etc.
Task	Specific task details.	Task ID, Title, Status, etc.
Task List	Describes a list of tasks.	Task List ID, Name, Created At, etc.

Table 3.2: Database model table

The following diagrams, as shown in Figure 3.2, reveal the linkages between the tables, illustrating in detail how they are connected and interact with each other. These connections clarify how data is shared and referenced across tables.

Given that this project encompasses team collaboration features, particularly in the issue management segment, there is a palpable risk of database contention. For instance, concurrent requests might endeavor to write to the database simultaneously, potentially leading to contention or even deadlocks.

To safeguard the atomicity of database operations, a strategy has been adopted in routes involving data insertion into the database: the use of the `with db.session.begin()` context manager. This ensures that all database operations, be it deletions or additions, are executed within a single transaction. Only when every operation has been successfully executed will `db.session.commit()` be invoked to cement the transaction. However, should any anomalies arise during the operations, the `except` block is poised to capture such exceptions and invoke `db.session.rollback()` to reverse the transaction. This approach ensures the database remains unaffected by partial modifications, thereby preserving the integrity and consistency of transactions.

3.3 Creation and Execution of Automated Test Cases

3.3.1 Design of GWT Language The GWT language is designed to intuitively map to user behaviors. It is structured around three primary constructs: "Given," "When," and "Then." Each of these constructs plays a specific role in defining test scenarios. Table 3.3.1 below provides a brief overview of these constructs, along with example usages.

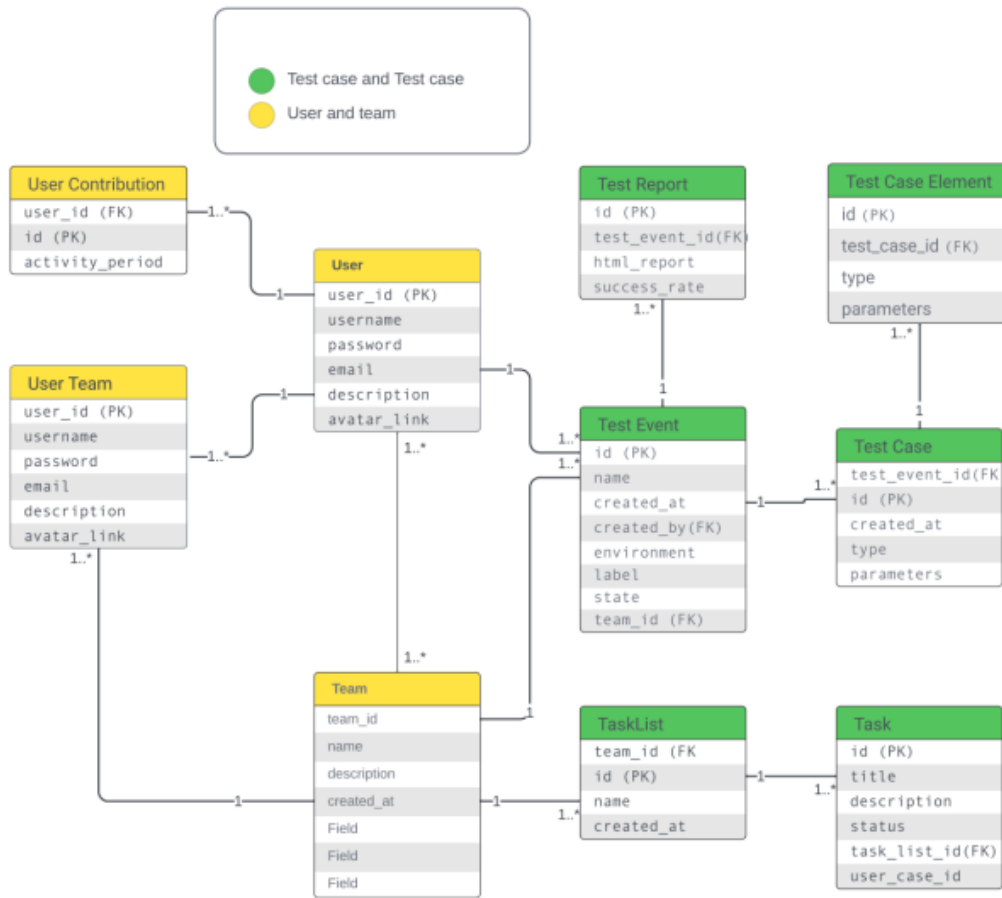


Figure 3.2: Database structure

Drag and Drop Constructs	Example
Given	given a particular URL
When	when this happens
Then	assert this condition

Table 3.3: Given-When-Then Meaning

- **Locator Selection in GWT Language Design**

During the design phase of the **GWT language**, pinpointed element identification was prioritized. While an intelligent matching of elements based on provided descriptions or attributes might be more in line with human browsing habits (as seen with **Uilicious**), this method can run into an issue: the system tends to choose the first matching element when multiple elements match the given description or attribute, introducing some level of unpredictability.

To circumvent such ambiguities and ensure consistent test behaviors, my project mandates users to specify the locator type they wish to employ, such as ID, Class Name, or Name. These are easily recognizable and widely used locators, ensuring accuracy and minimizing chances of misoperations.

Although Selenium offers a variety of locators with 8 distinct built-in element identification strategies in WebDriver [18], for the sake of a simplified user interface and to lower the learning curve, I initially incorporated only the ID, Class Name, or Name locator types. A table of all the locator explanations can be seen in the appendix (see Section B).

- **Design of User Behaviors**

Diving into the specific user behavior patterns currently embodied in the **GWT language**, the following Table 3.4 breaks down the GWT behaviors, detailing their definitions and accompanying use cases.

The structure above organizes the **GWT language**'s modules (Given, When, Then) and respective user behaviors in a logical and coherent manner. It is hoped this would guide users efficiently through the process of test case design and execution.

Their specific implementations are discussed below.

3.3.2 Implementation of GWT Language

- **User Behaviors and Their Implementation in Selenium**

Each user behaviour is supported by a specific implementation. The specific details of these behaviours and the corresponding Selenium methods are listed in the appendix (see Section A).

Despite the original design intentions, certain user behaviors were not feasible for implementation due to technical limitations. A case in point is the "User upload file" behavior. On various websites, the upload mechanism involves a **drag-and-drop** action, where a user drags a selected file to a specific area for uploading. While Selenium faces challenges in mimicking the action of invoking the file dialog and selecting a file, the mechanism often hiding behind such **drag-and-drop** functionalities is a straightforward `<input type="file">` element. By directly supplying the file path to this element and then pressing the upload button, the desired file upload effect can be replicated. As a result, this process can be split into two separate user actions. Given these intricacies, the decision was made to exclude this user behavior from the current design. A deeper exploration of its precise implementation will be reserved for upcoming tutorials.

- **The Drag & Drop User Interface**

Upon the successful design and realization of user behaviors, attention was

	Name	Description
Given	Users open the page	The user launches and opens a specific web page or application page, ready to start their session or interaction.
When	User input data	The user provides or fills in data in an input field, such as text boxes, dropdowns, etc.
	User click the button	The user clicks on a button on the page, typically to submit a form, open a new page, or trigger some function.
	User double clicks	The user double-clicks on an element on the page. This is usually used to open an item or initiate a specific part of an application.
	User right clicks	The user right-clicks on an element on the page. Typically, this opens a context menu offering options related to the selected item.
	User moves to element	The user moves the mouse cursor over an element on the page without clicking. This is often used to trigger tooltips or other hover effects.
	User refreshes the page	The user refreshes the current page, typically to load the latest content or fix some error on the page.
	User waits	The user pauses on the current page for a specified duration, possibly waiting for an animation, load, or other processes to complete.
Then	The user is now on this page	Verify if the user has successfully been redirected or navigated to the expected page.
	Check element exists	Confirm the existence of a specific element on the page to ensure completeness of load or functionality.
	Check element visible	Verify if a specific element on the page is visible to the user, possibly pertaining to the element's display status or position.
	Check text exists	Confirm the presence of expected text on the page, usually checking for messages, titles, or other key information.
	Check element selected	Verify if a specific page element (like a checkbox or radio button) is selected or activated.

Table 3.4: GWT language table

then redirected towards enhancing the front-end interface. This strategic shift aimed to preemptively address potential hurdles, notably the prevalent challenges tied to manual test language input, such as input mistakes and nuanced complexities.

The solution materialized in the form of a redesigned interface. Within this novel design, user actions found representation in modular "blocks." These

blocks were precision-crafted, each symbolizing distinct actions or conditions, ranging from launching a webpage, clicking a button, to text input. By introducing an effortless **drag-and-drop** mechanism to these blocks and associating them with input parameters, users found themselves equipped to emulate real-world actions in their test cases. This meticulous design strategy had automated testing at its core.

Furthermore, the implementation of the **drag-and-drop** feature dramatically transformed the landscape of test case creation. Not only was the error rate linked to manual inputs drastically reduced, but the precision of tests also saw a marked improvement. This intuitive design's ripple effects were apparent in the spike in user productivity. I also enriched the interface with a "palette" of language constructs. This "palette" refers to a collection or set of predefined language elements or building blocks available for users to utilize. This initiative ensured even those with minimal coding prowess could seamlessly design and execute tests.

The established foundation paved the way for the next step: executing the **drag-and-drop** interface, see Figure 3.3. Two main components emerged prominently: the LeftSideBar and the DroppableArea. Leveraging the powerful capabilities of react-dnd, an intuitive interface was crafted, emphasizing an optimal user experience.



Figure 3.3: Test case page

The LeftSideBar serves as a showcase for all **GWT language** components. Each of these components is fashioned as a draggable item. To enhance user clarity, I integrated conspicuous icons and descriptions, ensuring immediate recognition

of each component's functionality.

In contrast, the DroppableArea on the right was conceptualized as the primary workspace. More than just a receptacle for components, it also facilitated component reorganization, allowing for test scenarios to be adjusted as per user preferences. The user-centric design ensured that every action undertaken resonated with the user's objectives.

Additionally, I positioned a trash bin icon in the corner of the DroppableArea. Should users decide to part with a component, it can be dragged over this icon. A confirmation mechanism has been implemented to guard against unintended deletions.

Upon finalizing all operations, users can click the "run and save" button. The backend service captures the structured component data within the DroppableArea, processes it, generates the corresponding test cases, and commits them to the designated storage solution.

To encapsulate, my endeavors were not restricted to just devising a user-friendly **drag-and-drop** interface. Instead, I committed to delivering a holistic user experience, balancing both simplicity and precision in test scenario construction.

- **Language Export format**

After the user activates the "Run and Save" button, the system meticulously converts the user's input into structured test cases in JSON format.

The structure of these test cases is shown below: the outer layer encapsulates the "Given" element, while the nested "test_case_elements" capture the "When" and "Then" sections. The When and Then elements can only be added to the structure if the outermost Given element is present.

In addition to editing the test case components, the user can also determine the test environment, such as specifying the browser in which the test will be executed. These settings are called environment variables and are seamlessly integrated into the overall JSON data. An example of the structure of this JSON data is illustrated in Listing 1.

```
1 {
2   "id": 1692975587692,
3   "type": "Given",
4   "subtype": "Users open the page",
5   "params": [
6     {
7       "value": "https://ecommerce-example.com",
8       "type": "URL"
9     }
10  ],
11  "isNew": false,
12  "isChild": false,
13  "index": 0,
```

```

15     "selectorValue": "URL",
16     "children": [
17         {
18             "type": "When",
19             "subType": "User click the button",
20             "params": [
21                 {
22                     "value": "Login",
23                     "type": "Class Name"
24                 }
25             ],
26             "isNew": false,
27             "isChild": true,
28             "parentId": 1692975587692,
29             "index": 0
30         }
31     ]
32 }

```

Listing 1: JSON data structure

There are a number of benefits to choosing **JSON** as the format for test cases. Firstly, the structured nature of **JSON** is easy to handle and parse, providing a direct route to transforming these cases into automated test scripts. This structured format is also flexible, paving the way for easily adding, removing, or changing test cases. Not to mention, its inherent readability ensures that users can inspect and grasp the essence of test cases without any unnecessary complexity. Finally, the adaptability of **JSON**-formatted test cases extends to their storage and sharing capabilities, thus enhancing team collaboration.

3.3.3 Interpreter When the server gets the **JSON** data, it creates a "TestCase" object, which works like a translator. This project uses Python's **Selenium** package and the **Unittest** framework [10] to define a set of actions. There's also an action map that connects different actions with specific functions. When a test case runs, the system uses the action map to decide which functions to run.

In the backend, the system first reads the data. Then it checks each action's type and details. For example, if the action says, "User clicks the button," it runs the 'click_button' function. If it says, "User inputs data," then it uses the 'enter_text' function.

More function names corresponding to user behaviour can be found in the appendix(see Section C).

The translator also deals with any errors or unexpected situations. If there's a mistake in the action details or if something goes wrong, the system notes the error and lets the user know. This way, users can see if their test cases worked, and if not, they can find out why.

```

# Handling the "Given" step
if test_case['type'] == 'Given' and test_case['subtype'] == 'Users open the page':
    parameters = {param['type']: param['value']
                  for param in test_case['parameters']}
    open_website(self.driver, parameters.get('URL'))

```

Figure 3.4: Given function

```

# Parse and run test_case_elements.
for test_case_element in test_case['test_case_elements']:
    action_subtype = test_case_element['subtype']
    parameters = {(param['type'] if param['type'] else 'empty'): param['value']
                  for param in test_case_element['parameters']}
    action_function = action_mapping.get(action_subtype)
    if action_function:
        if action_subtype == "User input data":
            param_dict = test_case_element['parameters'][0]
            locator_type, locator_value = construct_locator(
                parameters)
            text_value = param_dict.get('textValue')
            action_function(self, self.driver,
                           locator_type, locator_value, text_value)

        elif action_subtype == "The user is now on this page" or "Check text exists" or "User waits":
            expected_url = parameters.get('empty')
            action_function(self, self.driver, expected_url)

        else:
            if any(key in parameters for key in ['ID', 'Name', 'Class Name']):
                locator_type, locator_value = construct_locator(
                    parameters)

                # Remove the used keys from parameters
                for key in ['ID', 'Name', 'Class Name']:
                    if key in parameters:
                        del parameters[key]

                action_function(
                    self, self.driver, locator_type, locator_value, **parameters)
            else:
                action_function(self, self.driver, **parameters)

```

Figure 3.5: When and Then function

The whole process starts when the translator gets the JSON data. This data tells the system which web browser to use and what test cases to run. Depending on the test setting, a function goes through the list of test cases. For each one, it creates a matching test method and adds it to the "TestCases" class.

This class, which comes from `unittest.TestCase`, gets the test environment ready before each test case and cleans up after. Next, all the test cases from the "TestCases" class are gathered into a test suite. An "HTMLTestRunner" object is made and runs the test suite. As part of Python's `unittest`, `HTMLTestRunner` [7] can make an HTML report of the test results.

Lastly, the system makes an HTML report and saves it in a specific place. After all the test cases have run, it creates a "TestResult" object that has the results for all the tests in the current setting, as shown in Figure 3.6.

3.4 Test report

The platform in question is adept at autonomously generating and executing test codes predicated upon user-defined test cases. It subsequently engenders HTML test

```

def run_tests(environment, test_case_list, report_file):
    TestCases.environment = environment
    TestCases.test_case_list = test_case_list

    for i, test_case in enumerate(test_case_list):
        test_method = create_test_method(test_case)
        setattr(TestCases, f'test_case{i}', test_method)

    test_suite = unittest.TestSuite()
    test_loader = unittest.TestLoader()
    test_suite.addTest(test_loader.loadTestsFromTestCase(TestCases))

    runner = HtmlTestRunner.HTMLTestRunner(
        output=report_file, combine_reports=True)
    test_result = runner.run(test_suite)

    return test_result

```

Figure 3.6: Test Run Function

reports, ensuring they are systematically stored for subsequent perusal and analysis by respective team members.

Let us delve into the intricate design and implementation methodologies concerning the generation, storage, and exposition of these test reports.

3.4.1 Test Report Generation As each different test case is executed, a corresponding test report is generated. For this purpose, various libraries can be utilised, in particular the `HtmlTestRunner` library in the *Python* domain. Such libraries are responsible for coordinating the generation process, encapsulating important details including test case terminology, status indicators (*pass/fail*), duration, and any exceptions or anomalies that may occur.

To illustrate, in a given test class, once the code starts executing, an `HTML runner` is traditionally instantiated that directs the output to the specified report file:

```

runner = HtmlTestRunner.HTMLTestRunner(output=report_file
    ↪ , combine_reports=True)

```

At the end of the test, the `HTML runner` generates the report and stores it in a predetermined directory with an `HTML` layout.

3.4.2 Storing Test Reports After generation, the report can be stored in a `filesystem` and a `database`. Given that a test case might correspond to multiple testing environments and considering that `HtmlTestRunner` produces a report detailing only one browser result, a test case might link to one or multiple test reports. At this juncture, the addresses of test reports are stored in a `JSON` format, capable of accommodating multiple addresses. Moreover, the success rate within the `TestEvent` represents the average success rate across all environments.

3.4.3 Displaying Test Reports When presenting the TestReport, a single test case may correspond to multiple TestEvent objects, with each TestEvent linking to a report page.

Within the report page, it exhibits the intricate details of the test report, such as its name, creation time, creator, associated test case ID, testing environment, labels, status, and success rate. Notably, the test environment details are displayed as *Chip* labels, presenting each environment (like "chrome" or "edge") as a distinct tag. Additionally, if there are HTML-formatted sub-reports, they are embedded within the page, each displayed within its separate frame. For navigational convenience, a button is provided, allowing users to swiftly transition to the associated test case page.

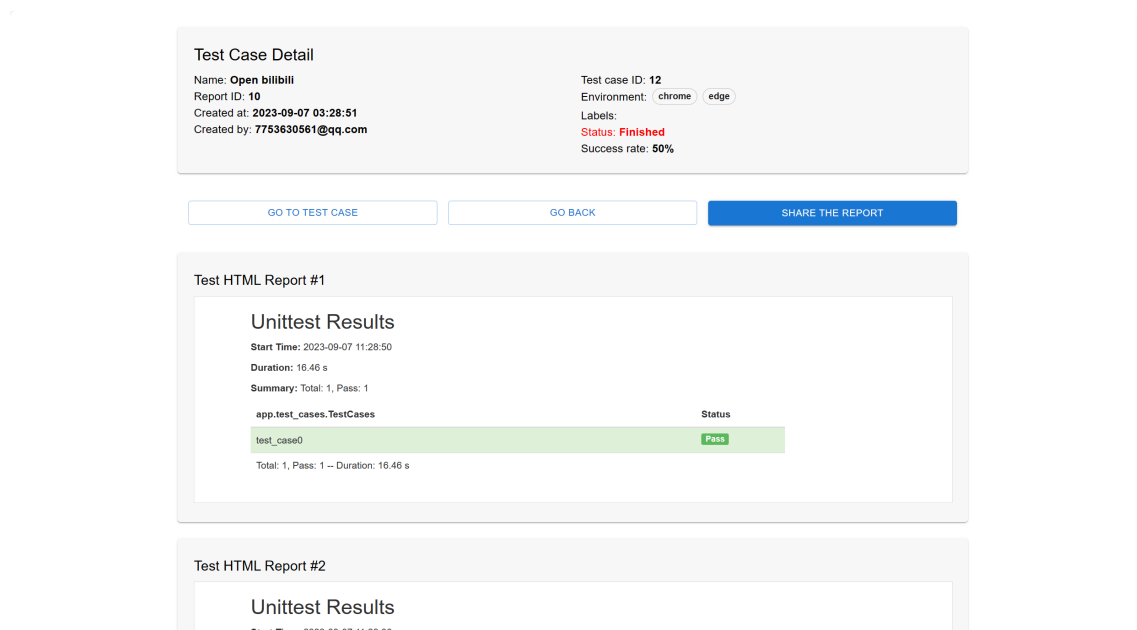


Figure 3.7: Test report page

Currently, users have two avenues to share these test reports. They can either directly share the page link, such as <http://localhost:3000/testReport/142>, or utilize the "Share Report" button on the page, as shown in Figure 3.8. Upon clicking this button, users can select a member from the team list. The system then dispatches the auto-generated test report straight to the chosen recipient's mailbox, facilitating seamless information sharing amongst the team, as illustrated in Figure 3.9.

To implement the email sending feature, I followed the steps outlined below:

- **Data Extraction and Parsing:**

Specific test reports and associated test events are initially pulled from the database. To effectively display them in an email, the saved JSON data is de-

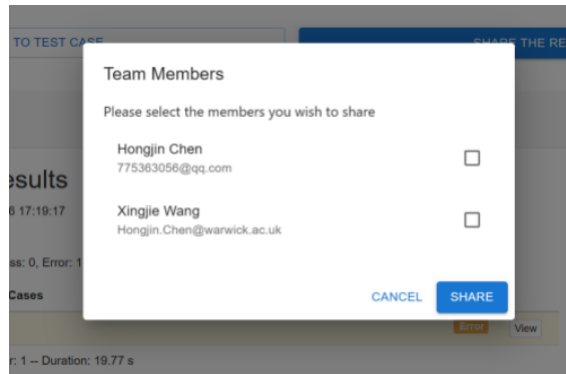


Figure 3.8: Select and send reports to team members



Figure 3.9: Test report emails in the target's mailbox

coded. Subsequently, relevant HTML reports are extracted from their respective paths.

- **Data Rendering and Formatting:**
Using the Jinja2 templating engine, the extracted data is integrated into a preset HTML template. This approach ensures that test reports sent to recipients are well-organized and highlight vital information.
- **Email Dispatch:**
After preparing the formatted HTML report, it employs the SMTP protocol with QQ's SMTP server to send the email. Here, this application connects to the SMTP server outside and then forwards the email to the specified email ad-

resses.

- **Error Handling:**

To provide users with instant feedback, this system incorporates error-handling throughout the sending process. This system promptly identifies and reports various potential issues, from JSON decoding hitches to mail dispatch challenges.

Currently, I use a set personal QQ email, 775363056@qq.com, as the source for dispatching emails. This ensures that test emails are sent reliably, sidestepping the intricate setup of external servers. However, this approach is temporary. In the upcoming phases, this system is aiming to allow users to employ their own email addresses for dispatching, thereby increasing the system's adaptability and personal touch.

3.5 Implementation of Other Basic Website Features

In this section, I've listed all the features below. Most of these functionalities were implemented without resorting to specialized libraries or encountering significant challenges. Therefore, I'll provide a concise overview of their implementation and a guide on how to use them.

3.5.1 User

- **User Registration and Login**

This section revolves around facilitating users to establish a new account and subsequently log into their profiles. Central to this are the following primary features and attributes:

User registration encompasses two pivotal components. Firstly, a form completion step where users can input fundamental details including their name, email, and password. Secondly, there is a stringent data verification process to ensure the legitimacy of all user inputs, such as email format scrutiny and password strength evaluation.

For user login, the system primarily focuses on identity verification where users gain access by inputting their email and password, which the system cross-checks for authenticity. An additional feature is the password recovery option, enabling users who forget their passwords to reset them via their registered email.

From a security standpoint, passwords saved in the database undergo encryption to ensure they remain impenetrable by unauthorized personnel. Furthermore, session management plays a crucial role post-login. The system diligently oversees user sessions to confirm they remain logged in throughout their activity until they opt to log out. This component is quintessential to any website or application, safeguarding user data and offering a tailored user experience.

Login

Email *

775363056@qq.com

Password *

LOGIN

REGISTER

FORGOT PASSWORD

BACK

Register

username

Email

Password

Confirm Password

SUBMIT

BACK

Figure 3.10: Login and register page

• User Information Management

This section encompasses a variety of functionalities aimed at enhancing user autonomy over their profiles:

At the forefront of these is the ability to modify one’s password, a critical measure to bolster account security. Alongside this, users are granted the flexibility to update their personal information, ensuring that their details remain current and accurate. Changing the registered email is another significant feature, facilitating users who might switch their primary email addresses.

Furthermore, a logout function is integrated, empowering users to end their sessions whenever desired, enhancing privacy and security. Notably, a unique aspect of this section is the inclusion of the ”contribution graph.” Mirroring the concept from platforms like GitHub, users can visually track their activities, giving them insights into their engagement and contributions over time. The third-party library `react-calendar-heatmap` is used to display a graph of user contributions.

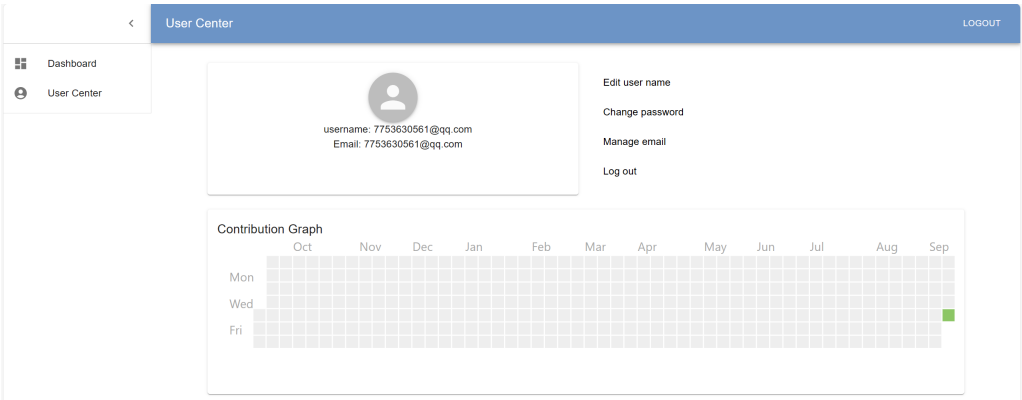


Figure 3.11: User center page

• Team

In the "Team" section, the focus is primarily on "Team Creation and Management."

Under "Team Creation," users are provided with the tools to form a new team from scratch. During this process, they're prompted to input essential details such as the team's name and description. Additionally, an integral part of the creation process is selecting which members to include in the team, enabling the initiator to handpick participants.

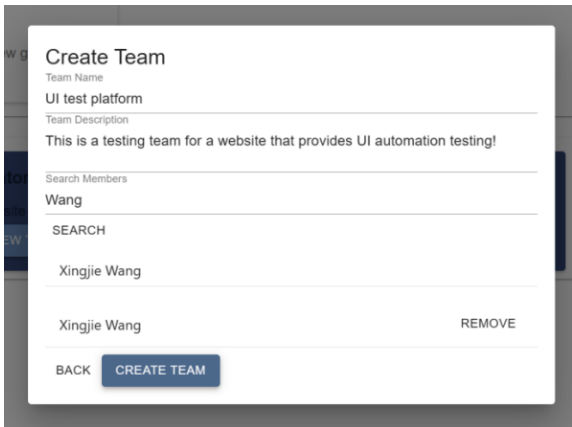


Figure 3.12: The interface to create a team

Once the team is set up, the "View and Manage Team Information" functionality comes into play. It serves multiple purposes, from allowing members to see the list of teams they've joined, to viewing a roster of current team members. Additionally, for those with the requisite permissions, managing the members' list—adding new members or removing existing ones—is a straightforward process.

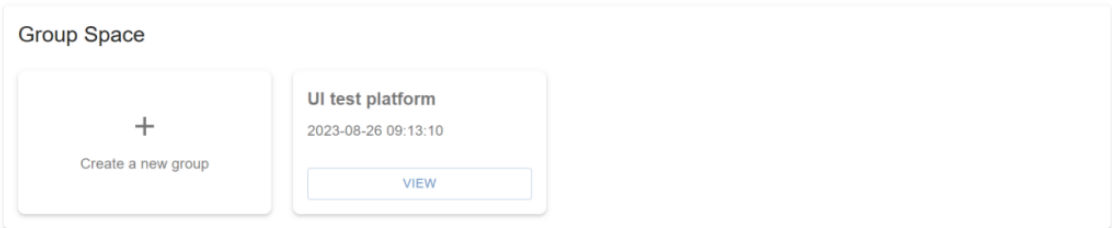


Figure 3.13: Group space component

- **Task**

In team-based task management, the "Task" section stands as its core. This area comprises a suite of tools specially crafted for seamless collaboration and streamlined operations.

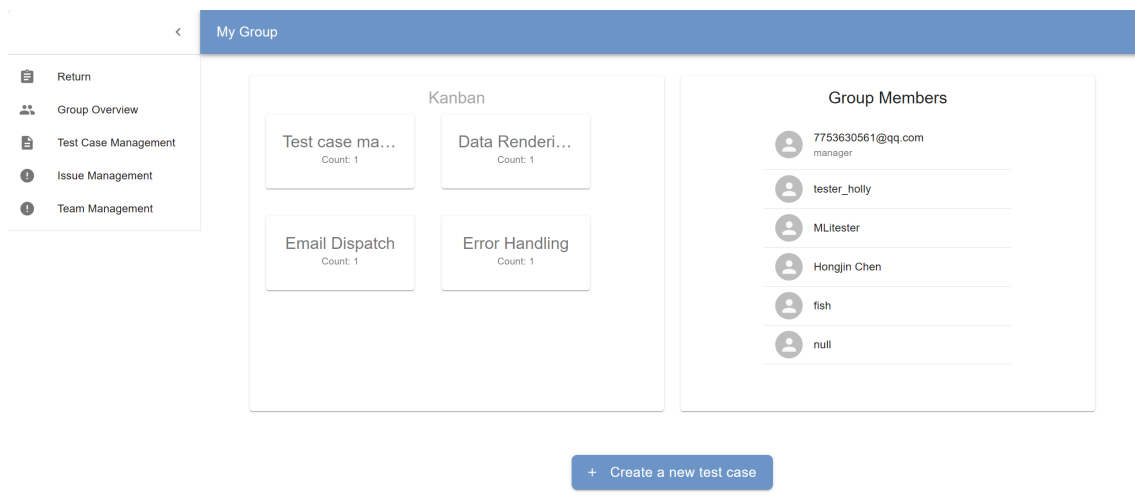


Figure 3.14: Group page

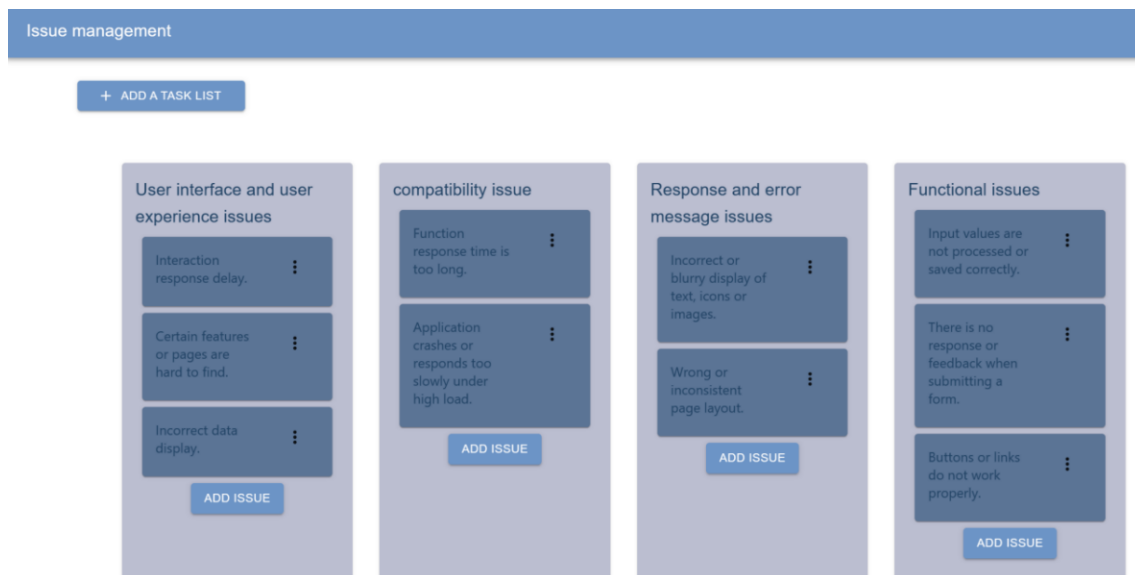


Figure 3.15: Task management page

The foundational feature is "Create Task Lists within a Team." Through this utility, teams can architect organized lists tailored for various projects or objectives. Once the list is established, team members can conveniently "Add Tasks", ensuring every step towards a goal is meticulously outlined. To synchronize efforts among team members, the "View Team Task List" function is

pivotal. It displays a panoramic view of all tasks, providing clarity on collective goals and individual contributions.

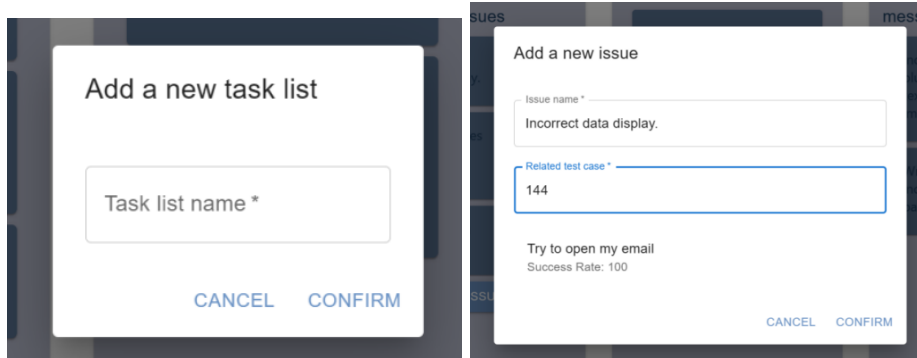


Figure 3.16: Add new task list and add new tasks

What sets the "Task" section apart is its democratic ethos towards task management. Every member is empowered with the rights to both append and remove tasks. This inclusive approach is geared towards nurturing mutual accountability and synergy. However, it also brings the challenge of concurrent editing to the fore. To counter instances where multiple members might overlap edits, stringent protocols are in place to maintain data consistency and reliability. In this pursuit of data sanctity and to mitigate concurrency dilemmas, project initiates a fresh transaction prior to any database action. This ensures that if all steps run smoothly, the transaction gets committed; else, in case of discrepancies, an immediate rollback ensures operational cohesiveness.

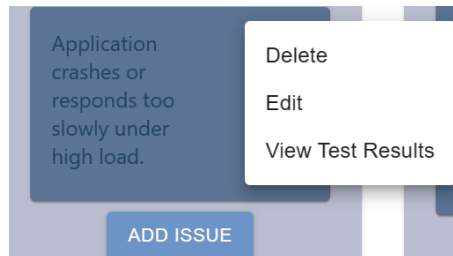


Figure 3.17: Task management

Another distinct feature of the "Task" section is its egalitarian approach to task management. Every team member possesses the authority to both add and remove tasks. While this strategy aims to foster shared responsibility and collaboration among team members, it introduces challenges with concurrent editing. For scenarios where multiple members might edit a task simultaneously, rigorous measures have been put in place to ensure data consistency and integrity.

To further enhance user experience, the platform integrates **React DnD**, which facilitates **drag-and-drop** functionality for tasks via **useDrag** and **useDrop**. Additionally, the system generates a unique ID for every new task or task list using the **uuidv4** function, ensuring the uniqueness and integrity of each task within the system.

- **Test Case Management**

On this page, users can search for specific scripts by ID, name and creator using multiple text input boxes. At the same time, a drop-down menu allows users to filter scripts according to their status (e.g. **"Finished"** or **"In Progress"**).

Search ID

Search Name

Search Creator

Status Filter▼

+ Create a new test case

ID	Name	Time	Label	State	Creator	Operation	
195	Login page	Sat, 26 Aug 2023 09:19:17 GMT	The first test!	Finished	Hongjin Chen	VIEW REPORT	VIEW SCRIPT
196	Try to open my email	Sat, 26 Aug 2023 13:49:41 GMT		Finished	Hongjin Chen	VIEW REPORT	VIEW SCRIPT
211	Try to open my email box	Sun, 27 Aug 2023 15:54:47 GMT		Finished	Hongjin Chen	VIEW REPORT	VIEW SCRIPT
212	Open my website	Sun, 27 Aug 2023 15:56:46 GMT		Finished	Hongjin Chen	VIEW REPORT	VIEW SCRIPT
213	Try to click the button	Sun, 27 Aug 2023 15:59:22 GMT		Finished	Hongjin Chen	VIEW REPORT	VIEW SCRIPT
214	Right click example	Sun, 27 Aug 2023 16:22:34 GMT		Finished	Hongjin Chen	VIEW REPORT	VIEW SCRIPT
218	Try to find a new problem	Mon, 28 Aug 2023 03:24:57 GMT		Finished	Hongjin Chen	VIEW REPORT	VIEW SCRIPT
219	This is a problem to get parameters	Mon, 28 Aug 2023 03:28:24 GMT		Finished	Hongjin Chen	VIEW REPORT	VIEW SCRIPT
220	This is another try.	Mon, 28 Aug 2023 03:29:50 GMT		Finished	Hongjin Chen	VIEW REPORT	VIEW SCRIPT
221	Share to other teammate	Mon, 28 Aug 2023 03:32:48 GMT		Finished	Hongjin Chen	VIEW REPORT	VIEW SCRIPT

Rows per page: 10 1-10 of 11 < >

Figure 3.18: Test case management

All scripts that meet the filtering criteria are displayed in a clearly structured table, where each script shows its attributes such as ID, name, creation time, label, status and creator. In addition, each script is accompanied by action buttons such as **"View Report"** and **"View Script"**.

Clicking on **"View Report"** brings up a dialogue box showing a detailed report related to the selected script. In order to improve user experience, the page

also provides a paging function, which makes it easy for users to browse a large number of scripts.

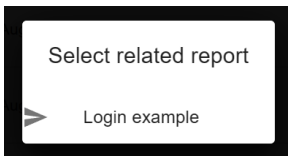


Figure 3.19: A test case may correspond to multiple TestEvent objects

● **Team Management**

This is an interface dedicated to team management, with the following key features:

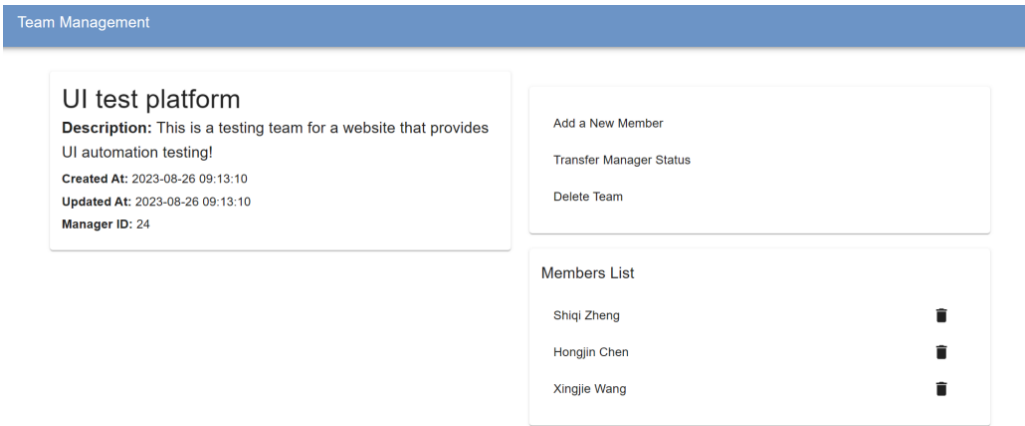


Figure 3.20: Team Management page

1. **Team Information Display:** The page clearly showcases the team name, description, creation date, latest update, and details about the manager.
2. **Add Team Members:** Users can find and add new members by entering an ID or name. Within the selected member list, members can be removed at any time, ensuring the accuracy of the final list, see Figure 3.21.
3. **Remove Team Members:** For streamlined management, a trash can icon is placed next to each member. Upon clicking, a confirmation dialog pops up to prevent accidental deletions.
4. **Transfer Management Rights:** Users can select and replace the team manager from a dropdown list that comprises all team members, see Figure 3.22.
5. **Delete Team:** Users can delete teams that are inactive or no longer needed. A confirmation box appears before deletion to avoid mistakes, see Figure 3.23.

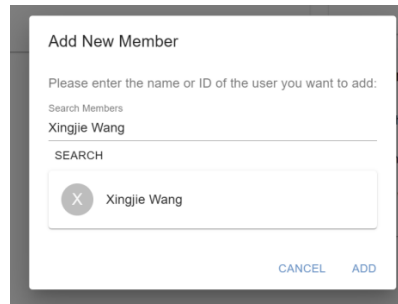


Figure 3.21: Add new members

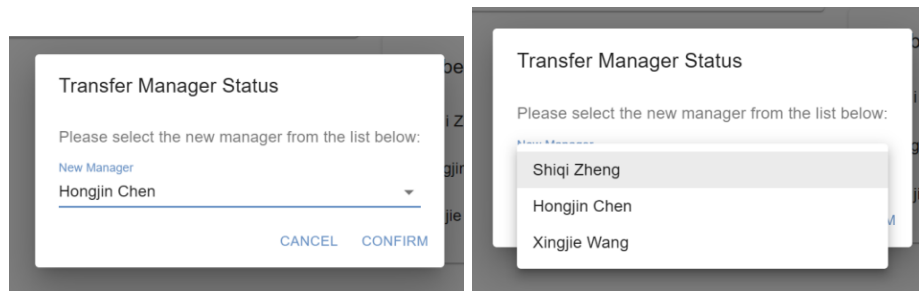


Figure 3.22: Transfer Management Rights

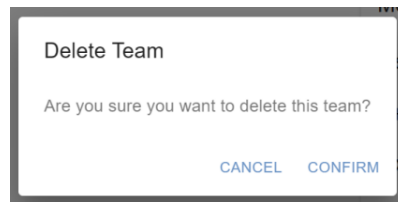


Figure 3.23: Delete team

Only users who are managers of the teams can see the buttons "Add Team Member", "Remove Team Member" and "Delete Team"; normal members do not have this access.

Any addition or removal of members leads to real-time updates of the team information on the page. All in all, this interface offers a one-stop solution for team management, aiming to simplify and optimize the team management process.

- **User Guidelines**

In the user guide pages, it provides a detailed introduction to "Basics HTML instructions", "Locator Selection Strategy", "User Behavior Function" and so on, see Figure 3.24. These are essential contents to help users better understand and utilize the website.

Locator Selection

In web automation testing, correctly locating page elements is crucial. This not only ensures that your scripts can interact with the correct elements but also ensures the stability and reliability of the test. Here are some suggestions and steps to help you efficiently locate elements using Selenium.

Use Browser's Developer Tools:

Almost all modern browsers come with powerful developer tools that allow you to view and interact with the HTML structure of a page. This is the first step in locating page elements.

Understanding Locators:

This project is based on Selenium, which provides several locators for you to choose from. This project supports class name, id, and name. If a button has id="submit-button", then it is the locator element we need to use.

Locator	Description
class name	Locate elements whose class attribute matches the search value (compound class names are not allowed)
id	Locate elements whose id attribute matches the search value
name	Locate elements whose name attribute matches the search value

Example

Below is an example to help you understand what locators are and where you can find them:

```
<html>  
<body>
```

Figure 3.24: Tutorial example

In addition to this, as shown in Figure 3.26, on the pages of the site, question mark icons are added in appropriate places, which, when the user hovers over them, will indicate the function of the current page and how to use it correctly.

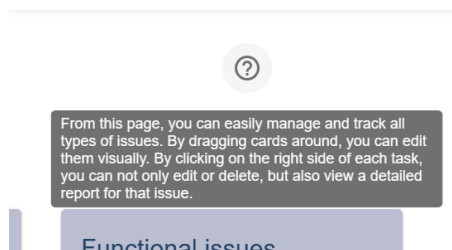


Figure 3.25: User guideline

After receiving feedback from users about the site's user guide, I added instructional tip boxes to the core pages to help users understand the functionality of each module and to guide them on how to proceed with functional testing.

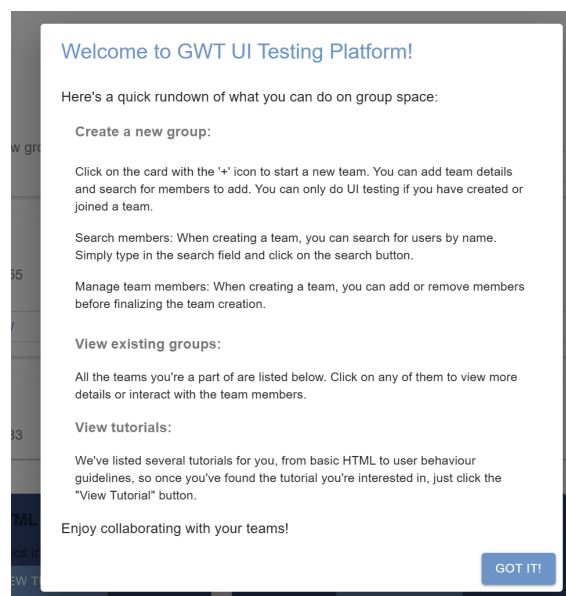


Figure 3.26: User guideline dialog

4 Project Management

4.1 Project Management Approach

As the project's sole developer, it was essential to adopt a project management methodology that was both structured and adaptive. To this end, a detailed schedule was meticulously crafted, serving as a foundational blueprint. This schedule outlined all crucial stages and tasks, coupled with their respective deadlines. By tracking progress in real time and routinely conducting self-assessments, adherence to the established timeline was assured.

4.2 Code Management

Leveraging **Git** and **GitHub** ensured proficient version control and efficient code repository management. A multifaceted branch strategy was religiously adhered to; even as a solo developer, this approach proved invaluable. All primary developments and modifications were undertaken on a separate branch. Concurrently, additional branches, such as “**testing**” and “**production**,” were earmarked for code verification and production deployment. Though this method introduced an added layer of complexity, its merits were undeniable. It provided enhanced organization to the codebase, and in doing so, preemptively addressed potential quality issues.

4.3 Operational Overview and Deployment

In the realm of project management, beyond the facets of planning, organizing, and controlling, the tangible aspects of deployment and operational management stand paramount. Presented below is a comprehensive overview of the operational and deployment specifics of the project, offering a lucid perspective on the hands-on management and execution aspects.

4.3.1 Server Specifications

- **Server Type:** Tencent Cloud Lightweight Cloud Server
- **IP Address:** 139.155.144.136
- **Deployment Environment:** CentOS 7
- **Security Configurations:** A firewall has been configured and enabled, complemented by the activation of HTTPS for enhanced security.

4.4 Project Access

- **Project Homepage:**
 - <https://perksummit.club/> (Due 2023-10-05)

– <http://139.155.144.136/> (Due 2023-10-05)

- GitHub Repository: <https://github.com/hongjinchen/dissertation-UI-test-platform>

To facilitate more convenient project deployment, a detailed description of three deployment strategies is provided in the **GitHub** repository. A tutorial on using dockerfile for a simple front-end project is in the appendix (see Section E), for more details check out the markdown file uploaded along with the report, or check out the **Github codebase**. First, a method based on **Dockerfile** is introduced, allowing users to easily build and initiate the project environment. Next, for scenarios that desire greater flexibility and control, a traditional deployment method not dependent on **Dockerfile** is provided. Lastly, the third approach illustrates how to deploy the backend project on a cloud server while using offline frontend deployment, ensuring a streamlined and efficient online deployment. These suggested methods aim to accelerate the project’s launch. Additionally, a link is provided for readers to directly view the project online.

4.5 Limitations on deployment

One reason for deploying the project on a cloud server is the nature of **Flask**, the backend framework chosen for this project. By default, its route handling functions operate in a single-threaded or single-process environment. In a development setting, such as local configurations, **Flask**’s built-in development server is designed as single-threaded, meaning it can only handle one request at a time.

However, in a production setting, **uWSGI** has been selected as the deployment server, which is inherently designed to handle multiple requests concurrently. However, a notable limitation persisted: both on the *Linux* cloud server and my local *Windows 11* system, browsers available for **Selenium** testing couldn’t provide comprehensive coverage. This challenge remains a pivotal area targeted for future enhancements.

Besides, when using **Tencent Cloud servers** for online link testing, there might be noticeable slowdowns in speed. This is primarily because, when the server tries to access cross-border resources, the speed can be affected due to factors like network routing, firewalls, and other related issues. For instance, in one of my recent test cases, just the time taken to access a website was as long as 96 seconds. The detailed test report can be viewed at this link (<https://perksummit.club/testReport/10>).

5 Test

5.1 User Test

5.1.1 Design In the recent user testing, the primary objective was delineated: the intention was to pinpoint issues within the interface and to gain a comprehensive understanding of user reactions towards specific features. The foundational hypothesis posed was two-fold: gauging whether the system was user-centric and intuitive, and assessing its tangible value to end users.

To serve this purpose, ten mock users were meticulously selected: five with a software development background and the other five devoid of such an experience, ensuring a spectrum of feedback from varied skill levels and backgrounds.

The drafted test plan encompassed a specific task and scenario, namely, assessing the login functionality of one website C for its operational viability. This task was meticulously framed to ensure precision while also granting users some autonomy. Moreover, a suite of open and closed-ended questions was integrated to guarantee a harmonious blend of both **quantitative and qualitative data**. For a detailed view and understanding of the questionnaire’s design and content, the questionnaire can be accessed directly via this link: <https://www.wjx.cn/vm/ha54IB1.aspx#>. Specific questions are likewise appended to the appendix (see Section F).

To eliminate any potential order effects, the Latin square methodology was employed, ensuring a randomized test sequence for systems A, B (Uilicious), and C (Metersphere). System A, the cornerstone of this research, was tested by every participant. Meanwhile, a straightforward randomization process determined if participants would subsequently test System B or System C. It was ensured that throughout the testing phase, both Systems B and C were evaluated by approximately an equal number of users. This approach ensured a balanced and randomized exposure, minimizing biases.

Throughout the test phase, meticulous notes were made detailing user challenges and their interaction dynamics with each system. To maintain data authenticity, feedback was gathered immediately post-testing of each system. Users grappling with issues or ambiguities were encouraged to encapsulate those in their concluding feedback.

5.1.2 Feedback During the user testing phase, feedback on a series of open-ended questions was collected by this project. The details can be viewed through this link: <https://kdocs.cn/l/ciz04G12HJH6>. These suggestions were compiled from online surveys and one-on-one interviews on social applications.

From the detailed survey results (see Section H) and open feedback, it was observed that users primarily focused on two main areas. Firstly, there are issues related to UI/UX, which include mobile compatibility, user interface design, consistency of features, browser-specific problems, and clarity of tutorials. Many users

reported suboptimal performance of the system on mobile platforms, such as delayed feedback after submission and unclear password guidance. Additionally, the system lacks a "back" button, and occasional blank screens were seen as areas needing improvement.

Meanwhile, when compared with Systems B and C, most users found this project's functionality to be more streamlined. Although System C boasts advanced features like online database creation and UI testing, it's not very user-friendly for non-tech users. In contrast, the drag-and-drop programming interface of this system and the simple command line sentences of System B seemed more approachable. While Systems B and C may look more professional with a range of features, some additional processes, such as linking to WeChat or initiating trial services, were deemed inconvenient. Particularly, the complex interface of System C was seen as a hurdle in user experience.

It's worth noting that although the testing method of this project was considered straightforward, users still faced some technical challenges. Such feedback was not only for our platform but also resonated with Systems B and C. While most users without a technical background appreciated the simplicity of our system, they also expressed a desire for stronger technical support. Common questions like "What is a locator?" and "What is an element?" indicate that offering users a foundational HTML tutorial and basic knowledge is crucial.

Further user interviews revealed that, compared to Systems B and C, the majority felt that this system was more streamlined in terms of functionality. User satisfaction surveys also showed a very positive response towards the drag-and-drop programming interface of this project. Most users felt confident using it, believing that they could get the hang of it quickly. Feedback on the GWT language was minimal, but its design, which is closer to natural language, was widely regarded as easy to understand, making it more accessible for beginners. However, limited feedback might be due to users' unfamiliarity with testing scenarios, restricting extensive exploration of user operations. In the future, feedback from individuals with extensive experience in related fields will be beneficial.

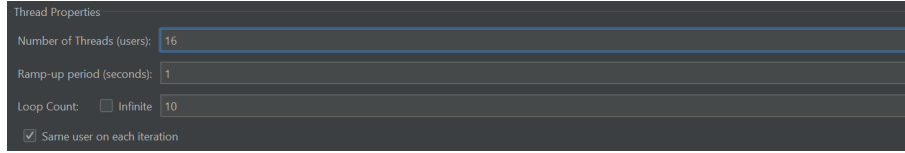
However, streamlined features don't always translate to the best user experience. A user who specializes in automated testing noted that while this system excels in generating basic test cases, these cases have limited reusability in various real-world scenarios. They perceived the system more as an educational tool with limited capability in handling complex situations. Thus, a balance still needs to be struck between feature richness and user-friendliness. It was suggested that tailoring functionalities according to specific user needs during registration might be more appropriate.

In conclusion, the feedback collected offers valuable guidance for this project. Measures have already been taken to address identified UI/UX problems, but tutorial content still needs further enhancement. Moreover, the high praise for the drag-and-drop programming interface and the recognized ease of use of the GWT

language are distinct advantages. Once the aforementioned issues are addressed, users can expect a more superior experience.

5.2 Stress Test

5.2.1 Testing Environment and Procedure The stress tests were conducted using **Apache JMeter** [1], deployed on a local device (Configured as shown in the Figure 5.1). Due to device constraints in terms of performance and resources, the maximum concurrent user simulation was limited to 16 (Because the number of threads in the device is 16). To provide a comprehensive assessment of the system’s capabilities, the number of concurrent users was gradually ramped up, starting from a single user and incrementally increased to the maximum of 16. At each stage, average response times were meticulously documented. This systematic approach aids in gauging how the system’s performance metrics evolve with an uptick in user count. The ensuing segments of this report delve deeper into the results, offering insights into the system’s behavior across different levels of concurrency.



Thread Properties	
Number of Threads (users):	16
Ramp-up period (seconds):	1
Loop Count:	<input type="checkbox"/> Infinite 10
<input checked="" type="checkbox"/> Same user on each iteration	

Figure 5.1: Configuration of Stress Test

5.2.2 Analysis of Stress Test Results The stress test results, as detailed in the appendix (see Section I), brought several key observations to the forefront:

1. **Average Response Time:** A steady incline is observed in the average response time as concurrent users multiply. Interestingly, this rise remains within moderate bounds, indicating that the system holds its ground and retains relative stability even as the pressure mounts.

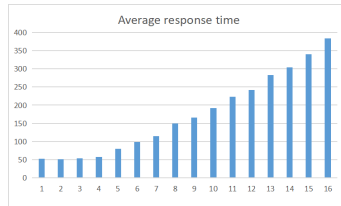


Figure 5.2: Average Response Time vs Concurrent Users

2. **Maximum Response Time:** With the surge in concurrency, a parallel upward trajectory is evident in the maximum response times.

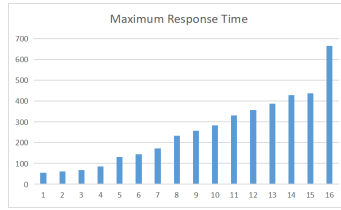


Figure 5.3: Maximum Response Time vs Concurrent Users

3. **Standard Deviation (Std. Dev.):** Serving as a barometer for response time consistency, the standard deviation unsurprisingly amplifies in scenarios of escalated concurrency. This underscores an evident challenge to maintain uniformity in response times under intense loads.

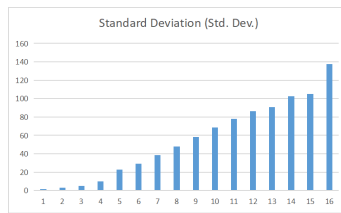


Figure 5.4: Standard Deviation of Response Time vs Concurrent Users

4. **Throughput:** This metric, which captures the count of requests processed per second, registers a slight elevation as the user count swells. Yet, this augmentation is not stark, subtly hinting at potential system bottlenecks.

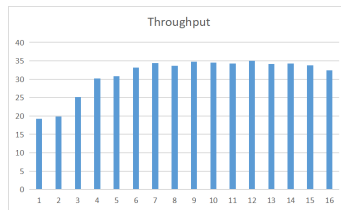


Figure 5.5: System Throughput vs Concurrent Users

Drawing from the analysis, several inferences can be formed:

- Even in the face of mounting concurrent demands, the system showcases resilience and an appreciable capability in managing requests.
- The uniformity of response times, however, shows signs of strain as concurrency levels soar.
- A closer look at the throughput figures implies potential performance bottlenecks. The rate of request processing doesn't exhibit proportional growth alongside concurrent users.

5.2.3 Potential Causes A confluence of factors might be at play, influencing the observed throughput ceiling:

- Python's innate execution speed, which tends to be on the slower side, could be a contributory element.
- The employment of **ORM frameworks (SQLAlchemy)**, for interfacing with databases might be adding an extra layer of performance overhead.
- External factors like the server's hardware makeup and the prevailing network conditions might be other pivotal determinants influencing performance metrics.

6 Appraisal and reflection

6.1 Change in project conceptualisation

At the outset of the project, a principal challenge faced was the methodological design for creating test cases and the implementation of the user interface. Initially, a command-line approach, reminiscent of **Uilicious**, was utilized. While functional, it was perceived as lacking in user-friendliness and innovation, particularly for the non-technical demographic. To cater to this audience, a shift was made to the **Gherkin-style** language module and a **drag-and-drop** mechanism was incorporated using the **react-dnd** library. This transition facilitated users in creating test cases through a more intuitive **drag-and-drop** mechanism, marking a pivotal development for the project. The integration of **react-dnd** and the design of this new interface posed both challenges and rewards.

6.2 Interpreting feedback and reflecting

However, subsequent feedback highlighted that significant emphasis was placed on the clarity and comprehensiveness of tutorials by users, irrespective of their technical background. It was observed that the current guide posed challenges for users not well-versed in technology. Despite the project's objective being to cater to such an audience, further enhancements are evidently required to better meet their needs. The future direction is anticipated to include refined tutorials and increased intelligent technical support. Feedback also underscored the potential benefit of foundational **HTML** tutorials. Additionally, the "Ask **Uilicious AI**" feature from **Uilicious**, which provides users with immediate technical assistance, was recognized as a commendable benchmark.

Throughout the project's lifecycle, the importance of user feedback was underscored. Inputs from mentors and peers illuminated areas for improvement, necessitating iterative refinements. Feedback from end users emphasized the importance of early engagement and a design approach rooted in genuine user needs, ensuring a more aligned product offering. A comprehensive and inclusive design approach was identified as pivotal for broader software adoption.

6.3 Technical challenge

From a technical perspective, the project predominantly relies on several mature libraries, such as **Selenium**, **Flask**, **React**, and the **unittest** framework. Benefiting from their substantial user base and robust community support, many potential issues have been circumvented. However, the backend choice of **Python** coupled with the **Flask** framework did not deliver optimal concurrent performance. **Flask**, inherently synchronous, isn't tailored for handling massive concurrent requests. Although there's potential to bolster concurrency in the later stages by pairing **Flask**

with a **WSGI** server, allowing for multi-process or multi-threaded instances, inherent limitations still persist. If the project aims for a full-scale launch anticipating extensive user access, considering a shift in the backend framework would be prudent. Nonetheless, for rapid prototyping and proof-of-concept validation, **Python** and **Flask** are undoubtedly excellent choices.

In reflection, technical skills were enhanced, and a deeper appreciation for user-centric design and project management was cultivated. Emphasis on user feedback and real-world testing is seen as crucial for future projects.

In conclusion, this project not only broadened technical horizons but also provided invaluable experience in managing a project from inception to completion. The insights from this endeavor are anticipated to significantly shape future software development and project management approaches.

7 Ethics

7.1 Data Protection and Privacy

In line with the University of Warwick's high ethical standards, this educational programme does not collect or store personal information about its users. Whilst some assessments may include input from fellow students or friends, all participation is entirely voluntary and no personal details are involved. Such measures ensure that the project is consistent with the University's ethical standards, emphasising voluntary participation without the use of deception, undue influence or targeting of sensitive groups.

7.2 Potential Misuse of Testing Tools

This project aims to improve and simplify the software testing process. However, it is clearly recognised that there are some potential risks of misuse of the tools, such as attempting to conduct '*Denial of Service Attacks*' (*DoS*). Such malicious uses not only deviate from the original design intent of the tool, but may also breach legal and ethical norms. Based on the results of the stress tests, current response times and data transfer speeds do not reach levels that could threaten server stability, thus reducing the risk of inappropriate use. Nonetheless, I strongly urge all users to maintain high ethical standards at all times when using this tool and to ensure that its use is legal.

8 Conclusions

This project has successfully developed a user-centric Web application interface testing platform, aiming to simplify the automation process of UI testing. Drawing inspiration from the philosophy behind the Gherkin language, we have created a more intuitive **GWT** testing syntax, enabling users with a modest technical background to effortlessly design and execute test workflows.

Compared to leading platforms in the industry like **Uilicious** and **Metersphere**, although there are some disparities in key functionalities such as automated test cases, generation and sharing of test reports, task management, and team collaboration, the user-friendliness of our platform is undeniably a distinctive advantage. Especially our innovative drag-and-drop method for creating test cases offers users a convenient way to realize functional testing. The report generation and sharing feature also facilitate team communication and collaboration.

On the technical front, we have adhered to the classic **MVC** architecture. The front-end utilizes **React.js** and **Redux**, ensuring smooth component-based development and unified state management. **Material-UI** ensures a consistent and elegant user interface, while **Axios** manages data exchange between the front-end and back-end. The back-end is constructed with Python in tandem with the Flask framework and incorporates **Selenium** for automating the testing process. Data storage relies on the stable **MySQL** system, with **phpMyAdmin** aiding in data management.

For our target audience, namely students in university software engineering courses and startup enterprises, the design of this platform undoubtedly caters to their UI testing needs. After a series of stress tests on both users and the server side, we've gathered feedback and further refined the project to ensure its performance and stability.

During the development process, we've employed **Git** and **GitHub** for version control, and the project is hosted on a **lightweight cloud server** provided by **Tencent Cloud**, with reinforced security configurations. Furthermore, we place a high emphasis on user data privacy and security, implementing stringent measures to ensure data safety and prevent potential misuse of the testing tools.

In conclusion, not only does this project offer a feature-rich and user-friendly Web application UI testing platform, but it also brings fresh perspectives and insights to the realm of software testing. We eagerly anticipate further refining and expanding this platform in the near future to meet the practical needs of a broader user and enterprise base.

8.1 Future Development

Based on reflections and feedback from users, several key areas have been identified for long-term development. The primary focus of this project is to refine existing features, elevate the user experience, and make the tool more intuitive and user-friendly. Here's a breakdown of the plans:

1. Support for Multiple Testing Environments (Browsers):

In the domain of web development, it's acknowledged that different browsers can significantly influence a site's functionality and appearance. While tests on **CentOS** systems for **Google Chrome** and **Mozilla Firefox** have been successful, extending these tests to **Microsoft Edge** and **Safari** on **CentOS** has proven to be challenging. To address these challenges, the following strategies have been outlined:

- **Remote WebDriver Strategy:** To facilitate tests on Safari, it is planned to deploy Safari's **WebDriver** on a **macOS** device. With this configuration, test commands can be relayed remotely from a **Linux** server, with Safari on **macOS** executing these tests. This method offers the advantage of genuine browser environment testing, ensuring greater precision in test outcomes.
- **Leveraging Browser Core Engines:** Although testing on actual browsers remains the benchmark, insights can be obtained by testing on their core engines. For example, the use of **WebKitGTK** and other **WebKit** variants on **Linux** is being explored to emulate a Safari testing environment.

2. Enhancing GWT Language Design:

Currently, the **GWT** language in the project encompasses 13 modules, which, while covering basic user operations, will require future expansion based on user feedback.

3. Back-End Project Performance Enhancement:

Though the system's performance meets current standards, there's room for optimization. In the database realm, the use of **SQLAlchemy** as an **ORM** has been found to add complexity to queries. To boost performance, a shift towards raw **SQL** for data management is under consideration. Furthermore, the back-end routing code should be fully reviewed and revamped to minimize database redundancy and duplicate operations. Also, to cater to increased concurrent user demands, an upgrade to the server infrastructure is being planned.

4. User Tutorial Refinement and Real-time Support:

To further enhance user experience, more intuitive guides and tutorials will be developed, with a special emphasis on basic technical concepts. Online help features, like a chatbot to automatically answer frequently asked questions, are being considered, inspired by the "Ask Ullicious AI" feature in the **Ullicious** project. This feature is projected to be especially beneficial for users without a technical background.

Moreover, to simplify the user operation process, an innovative element locating strategy will be developed. By allowing users to simply copy-paste their

desired elements into a specified input box, the system will be designed to autonomously extract appropriate locators and their values. This approach promises to amplify user efficiency and simplify tasks.

References

- [1] *Apache JMeter*. 5.6.2. The Apache Software Foundation. 2023. URL: <https://jmeter.apache.org/>.
- [2] Axios. *Promise based HTTP client for the browser and node.js*. Accessed: 2023-09-01. 2023. URL: <https://www.axios-http.cn/en/>.
- [3] Victor R Basili and Richard W Selby. “Comparing the Effectiveness of Software Testing Strategies”. In: *IEEE Transactions on Software Engineering* 12 (1987), pp. 1278–1296.
- [4] Selenium Contributors. Accessed: 2023-09-01. URL: <https://www.selenium.dev>.
- [5] Watir Contributors. *Watir stands for Web Application Testing in Ruby*. Accessed: 2023-09-01. URL: <https://watir.com>.
- [6] Oracle Corporation. *MySQL*. Accessed: 2023-09-04. 2023. URL: <https://www.mysql.com/>.
- [7] HtmlTestRunner Developers. *HtmlTestRunner: A Test Runner Plugin for unittest framework*. Accessed: 2023-09-04. 2023. URL: <https://pypi.org/project/HtmlTestRunner/>.
- [8] phpMyAdmin Developers. *phpMyAdmin*. Accessed: 2023-09-04. 2023. URL: <https://www.phpmyadmin.net/>.
- [9] Python Software Foundation. *Python*. Accessed: 2023-09-01. 2023. URL: <https://www.python.org/>.
- [10] Python Software Foundation. *unittest — Unit testing framework*. Accessed: 2023-09-04. 2023. URL: <https://docs.python.org/3/library/unittest.html>.
- [11] Heidilyn V Gamido and Marlon V Gamido. “Comparative Review of the Features of Automated Software Testing Tools”. In: *International Journal of Electrical and Computer Engineering (IJECE)* 9.5 (2019), pp. 4473–4478.
- [12] DHEERAJ KAKARAPARTHY. “Overview and Analysis of Automated Testing Tools: Ranorex, Test Complete, Selenium”. In: *International Research Journal of Engineering and Technology* 4.10 (2017), pp. 1575–1579.
- [13] Material-ui. *Material-ui: A popular react UI framework*. Accessed: 2023-09-01. 2023. URL: <https://v4.mui.com/>.
- [14] MeterSphere. *MeterSphere home page*. Accessed: 2023-09-01. Feb. 2023. URL: <https://metersphere.io/index.html>.
- [15] Pallets. *Flask: The Pallets Projects*. Accessed: 2023-09-04. 2023. URL: <https://flask.palletsprojects.com/>.
- [16] React. *React – A JavaScript Library for Building User Interfaces*. Accessed: 2023-09-01. 2023. URL: <https://reactjs.org>.
- [17] Ernani César dos Santos and Patricia Vilain. “Automated Acceptance Tests as Software Requirements: An Experiment to Compare the Applicability of Fit Tables and Gherkin Language”. In: *Agile Processes in Software Engineering and Extreme Programming: 19th International Conference, XP 2018, Porto, Portugal, May 21–25, 2018, Proceedings* 19. Springer. 2018, pp. 104–119.
- [18] selenium. *Locator strategies*. Accessed: 2023-09-02. Apr. 2023. URL: <https://www.selenium.dev/documentation/webdriver/elements/locators/>.
- [19] Smartbear. *Automated UI Testing that Covers You from Device Cloud to Packaged Apps*. Accessed: 2023-09-01. URL: <https://smartbear.com/product/testcomplete/>.

-
- [20] Uilicious. *What is Uilicious?* Accessed: 2023-09-01. 2023. URL: <https://docs.uilicious.com/v3>.

A List of user behaviour implementations in Selenium

Name	Implementation Method
Users open the page	get is a method of driver (i.e., Selenium webdriver). It accepts a URL as a parameter and instructs the browser to navigate to that URL.
User input data	send_keys is a method of WebElement that allows text to be entered into an input field. First, the find_element method must be used to locate the element, and then the send_keys method is called, passing the text to be entered.
User double clicks	ActionChains is a class provided by Selenium for simulating complex user interactions. double_click is a method of ActionChains that simulates a user double-clicking on an element on a web page.
User click the button	click is a method of WebElement that simulates the user clicking an element (such as buttons, links, etc.) on a web page.
User right clicks	context_click is a method of ActionChains that simulates the user right-clicking on an element on a web page.
User moves to element	move_to_element is a method of ActionChains that simulates moving the mouse over a specified element on a web page.
User refreshes the page	refresh is a method of driver that refreshes the web page in the current browser window.
User waits	This is implemented using sleep, which is a function of Python's time module. It pauses the current thread's execution for a specified number of seconds.
The user is now on this page	WebDriverWait and url_to_be are tools provided by Selenium that allow waiting for and checking if the browser's current URL has changed to the expected URL.
Check element exists	For Selenium, the type of element (text, image, button, etc.) does not matter as long as the user has the correct locator. find_element is a method of driver that tries to locate an element on a web page. If the element does not exist, it will throw a NoSuchElementException.
Check element visible	is_displayed is a method of WebElement that returns a boolean value indicating whether an element on a web page is visible to the user.
Check text exists	The entire source code of the current page is obtained using driver.page_source, and it checks whether the text exists in the source code.
Check element selected	is_selected is a method of WebElement that returns a boolean value indicating whether an element on a web page (such as a checkbox or radio button) has been selected.

Table A.1: List of user behaviour implementations in Selenium

Locator	Description
class name	Locates the element whose class attribute matches the search value (compound class names are not allowed).
css selector	Locates the element that matches the CSS selector.
id	Locates the element whose id attribute matches the search value.
name	Locates the element whose name attribute matches the search value.
link text	Locates the anchor element whose link text visible text matches the search value exactly.
partial link text	Locates the anchor element where the link text visible text partially matches the search value. If multiple elements match, only the first one is selected.
tag name	Locates the element whose tag name matches the search value.
xpath	Locates the element that matches the XPath expression.

Table B.1: Locator List

B Locator List

C Action mapping list

User behavior	Function name
Users open the page	open_website
User click the button	click_button
User input data	enter_text
User refreshes the page	refresh_page
User moves to element	move_to_element
User right clicks	right_click
User double clicks	double_click
Check element exists	check_element_exists
Check element visible	check_element_visible
Check element selected	check_element_selected
Check text exists	check_element_text
The user is now on this page	check_url_change
User waits	user_waits

Table C.1: Action mapping list

D API document

D.1 User Management

D.2 Team Management

D.3 Task Management

Name	Api	Methods	Status
Register User	/register	POST	User registration, creating new users and storing them in the database.
Login User	/login	POST	User login, validating user credentials and returning access tokens.
Get User Info	/user/<int:user_id>	GET	Obtain user information such as username and email.
Edit User Info	/infoEdit/<int:user_id>	PUT	Modify username and description
Change User Password	/changePassword/<int:user_id>	PUT	Updating user passwords.
Update User Email	/updateEmail/<int:user_id>	PUT	Update the user's email.
Search Users	/searchUsers	GET	Search the user list to find users based on the given criteria.

Table D.1: User Management APIs

Name	Api	Methods	Status
Create Team	/createTeam	POST	Create a new team.
Get User Teams	/getUserTeams/<int:user_id>	GET	Get a list of all the teams to which the user belongs.
Get Team Members	/getTeamMembers/<int:team_id>	GET	Get a list of team members.
Get Team Test case	/getTeamScript/<int:team_id>	GET	Get the test case information for the team.
Adding team members	/addTeamMember/<int:team_id>	POST	Add team members, allowing team administrators to add members to a team.
Delete team members	/removeTeamMember/<int:team_id>/<int:team_id>	DELETE	Remove team members, allowing team administrators to remove members from a team.
Delete team	/deleteTeam/<int:team_id>	DELETE	Delete a team.

Table D.2: Team Management APIs

Name	Api	Methods	Status
Save Task	/saveTask	POST	Save the task information.
Delete Task	/deleteTask/<int:task_list_id>	POST	Delete the task.
Get Task List	/tasklists/<int:team_id>	GET	Get the list of tasks for the specified team.
Delete Task List	/deleteTaskList/<int:task_list_id>	POST	Deletes the task list for the specified team.

Table D.3: Task Management APIs

D.4 Test Case Management

Name	Api	Methods	Status
Save Test Events	/saveTestEvents	POST	Save test case information.
Execute Test Case	/executeTestCase/int:test_case.id	POST	Execute the specified test case
Get Test Report	/test-report/<int:report_id>	GET	Gets the details of the specified test report.
Get Test Event Names	/testEventName	GET	Gets the names of all test cases.
Get Test Cases	/getTestCases	GET	Get all test cases.
Search Test Cases	/searchTestCase	POST	Search for test cases by ID.

Table D.4: Test Case Management APIs

Name	Api	Methods	Status
Generate Report	/generateReport	POST	Generate a report for the specified test case.
Get Report History	/getReportHistory/<int:user_id>	GET	Gets the report history for the specified user.
Download Report	/downloadReport/<int:report_id>	GET	Download the specified report.

Table D.5: Report Management APIs

D.5 Report Management

E Building a Docker Image

From the root directory of your project, execute the following command to build a Docker image:

```
docker build -t UI_test .
```

This will construct a Docker image named `UI_test` based on your `Dockerfile`.

E.1 Running the Docker Container

To run your application, use the command below:

```
docker run -p 3000:3000 UI_test
```

This initiates a new Docker container instance based on the `UI_test` image. The `-p 3000:3000` flag maps the container's port 3000 to port 3000 on the host machine.

E.2 Accessing the Application

Visit `http://localhost:3000` in your browser, and you should see your application running.

F User Feedback Questionnaire

Dear Participant,

Thank you immensely for dedicating your time to our system evaluation. Your insights are invaluable. This questionnaire bifurcates into two segments: System Usability Scale (SUS) and open-ended queries. Rest assured, your responses will be treated with utmost confidentiality.

F.1 System Usability Scale (SUS)

Kindly respond to the following statements, considering both our system and another testing system. If a statement resonates strongly with you, please rate it as a 5. On the contrary, if it doesn't, a score of 1 would suffice.

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I would imagine that most people would learn to use this system very quickly.
6. I found the system very cumbersome to use.
7. I felt very confident using the system.
8. I needed to learn a lot of things before I could get going with this system.

F.2 Open-Ended Questions

1. Which system do you find more efficient in creating a specific test and why?
2. What difficulties did you encounter while creating the test using both systems?
3. Which system do you believe is easier for generating correct and complex tests?
4. Do you have any suggestions or feedback for my systems?

G Invitation to Participate in Testing Evaluation

I am currently working on a graduation project aimed at devising professional test cases for website login functionalities. To validate the practical effectiveness and usability of my design, I cordially invite you to participate in this evaluation.

G.1 Testing Details

Objective: Through this testing, I aim to understand the accuracy and user-friendliness of the login testing module I designed when applied in real scenarios.

Testing Procedure:

1. Please choose a website you are familiar with or interested in as the test subject.
2. Utilize the test module I provide, written in GWT language, to test the login function of your chosen website.

G.2 Module Highlights

- My testing module comes pre-equipped with a series of user behavior patterns. You need only to follow the instructions of the module and the guidance of the target website to execute and record the test results.

- For comprehensive testing, kindly employ UI testing tools such as Metersphere or Uilicious to conduct the same testing process.

G.3 Feedback Requested

Upon completion, I eagerly anticipate your feedback regarding the usability and results of this module. Your insights will provide invaluable direction for refining my graduation project.

H Questionnaire results table

A number from 1 to 5 indicates the level of agreement with the question.

Question	1	2	3	4	5
I think that I would like to use this system frequently.	40%	30%	30%	0	0
I found the system unnecessarily complex.	30%	30%	0	0	40%
I thought the system was easy to use.	0	0	0	0	100%
I think that I would need the support of a technical person to be able to use this system.	100%	0	0	0	0
I would imagine that most people would learn to use this system very quickly.	0	0	0	0	100%
I found the system very cumbersome to use.	70%	0	30%	0	0
I felt very confident using the system.	0	0	0	30%	70%
I needed to learn a lot of things before I could get going with this system.	70%	30%	0	0	0

I Stress test result table

Test URL: <https://perksummit.club:5000/login>, test function: login

Label	Samples	Average	Min	Max	Std. Dev.	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
16	160	383	0	665	137.79	32.474	17.7	8.18	558
15	150	340	0	438	104.81	33.799	18.42	8.52	558
14	140	304	0	428	102.483	34.305	18.693	8.643	558
13	130	283	0	387	90.635	34.085	18.573	8.587	558
12	120	242	0	358	86.088	35.057	19.103	8.832	558
11	110	223	0	330	78.058	34.268	18.673	8.633	558
10	100	192	0	284	68.527	34.495	18.796	8.691	558
9	90	165	0	258	58.056	34.762	18.942	8.758	558
8	80	150	0	233	47.888	33.599	18.309	8.465	558
7	70	115	0	172	38.570	34.449	18.771	8.679	558
6	60	99	0	145	28.911	33.204	18.093	8.365	558
5	50	80	0	131	22.645	30.864	16.818	7.776	558
4	40	57	0	86	9.725	30.211	16.462	7.611	558
3	30	54	0	68	4.867	25.168	13.714	6.341	558
2	20	51	0	61	2.764	19.861	10.822	5.004	558
1	10	52	0	55	1.673	19.231	10.479	4.845	558

Table I.1: Stress test result table