University of Tunis Manar

FSEG TUNIS

End of Studies Report

submitted in partial fulfillment of the requirements for the degree of

BACHELOR IN BUSINESS INTELLIGENCE (BI)

Department of Information Technology

Next-Generation Testing: Automated, Accurate, and Agile.

Host company : TCE Enterprise

By:

Arij OUIRGHI

ACADEMIC ADVISOR:

Dr. Wassim Ayadi

PROFESSIONAL SUPERVISOR:

Mr.Hamza Guizani

Academic year

2022/2023

Approval

The project titled " Next-Generation Testing: Automated, Accurate, and Agile. " submitted by:

Arij OUIRGHI

Of May 2023 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of BACHELOR IN BUSINESS INTELLIGENCE (BI) .

Examining Committee:

Academic Supervisor:

Name	Institution	Signature

Professional Supervisor:

Name	Institution	Signature

Abstract

The aim of this project was to enhance software testing capabilities and bring them to the next level. The project was carried out over a period of three months for a German platform called WorkerHero, which connects workers with job opportunities using innovative technology. With over 45,000 users already registered, WorkerHero is known as the LinkedIn for the blue-collar sector. Our project involved extensive testing of the platform's features, including the integration of modern technologies to streamline the job search process. Using a combination of manual and automated testing methods, we were able to thoroughly test the platform's functionality and identify potential issues, which we then addressed promptly. By leveraging our testing expertise, we were able to ensure that WorkerHero's users receive a seamless and hassle-free experience when searching for job opportunities. Our project's outcomes will contribute to the advancement of software testing and serve as a benchmark for future testing projects.

Keywords: — Software testing, Innovation, Technology, Manual testing, Automated testing, Functionality, Issues, Testing expertise, Reporting and Monitoring.

Dedication

To those who have experienced the burning desire to achieve a goal, no matter how difficult the path may be, know that success is within reach. Over the past three years, my journey has been filled with challenges and setbacks, but I never lost sight of my vision. With unwavering passion and determination, I worked tirelessly day and night, driven by the belief that my hard work and dedication would pay off in the end.

To my parents,
who have always been my unwavering support system and have never let me falter. Their words of encouragement have given me the strength to keep pushing forward, even when the odds seemed insurmountable.

To my professors,
who have guided me, challenged me, and nurtured my curiosity and thirst for knowledge, I am deeply grateful for the invaluable lessons and skills that they have imparted upon me.

To my friends,
who have been with me through thick and thin, who have listened to me vent my frustrations and shared in my joys, I thank you for being my constant cheerleaders, my confidants, and my inspiration.

And to my supervisors,
who have seen my potential and have guided me along the way, providing invaluable advice and mentorship, I am forever grateful for the opportunity to learn from you and to grow under your guidance.

This is just the beginning of a long and fulfilling journey.

Acknowledgment

I would like to express my sincere gratitude to my supervisors : **Dr. Wassim Ayadi** and **Mr.Hamza Guizani** for their guidance, patience, and support throughout this project. Their insights and expertise have been invaluable.

I would like to extend a special thank you to **Mr.Khalil Mejdi**, for his unwavering support and encouragement throughout my journey. His valuable insights, feedback, and guidance have been instrumental in shaping my professional growth and development.

Finally, I would like to thank all the participants who took part in this study, their contribution has been critical to the success of this project.

Thank you all for your support and encouragement. This project would not have been possible without your help.

Table of Contents

List of Figures

List of Tables

General Introduction

Quality assurance in the IT market involves ensuring customer satisfaction and compliance with quality standards.

1. Providing confidence to the client
2. Ensuring the validity of each development stage.

Software quality assurance is not an easy task, as clients and developers often have differing views on how to achieve quality performance. To address this, planned and systematic activities are necessary to provide adequate assurance tailored to requirements and expectations.

Our project goal is to simplify the tasks of the QA service by automating and reporting test management processes.

This report is divided into five chapters, each addressing a different aspect of the project.

- In Chapter 1, we provide a general overview of the organization, project, and methodology used, which is the SCRUM agile method.
- Chapter 2 covers the preliminary study, where we discuss the types and levels of tests, and present a comparative study of existing test automation tools.
- In Chapter 3, we focus on requirements gathering, including identifying actors, defining functional and non-functional requirements, creating a product backlog, and release planning.
- Chapter 4 is dedicated to the project design phase, including preliminary design, technology selection, and development environment setup.
- In Chapter 5, we discuss sprint planning, design, and implementation, including testing and continuous integration/continuous deployment (CI/CD), as well as reporting via dashboarding.
- The report concludes with an overall evaluation of the solution deployment and potential perspectives related to the project.

Overall, this report provides a comprehensive view of the project, its development process, and the outcomes achieved.

Chapter 1

General Presentation Of The Project

Introduction

In this first chapter, we will introduce TCE, a dynamic and innovative firm that plays a crucial role in organizational change in Tunisia. We will also discuss the project's overall framework, problem, and goal.

I. Company presentation

This section will mainly cover the host company presentation, including its primary activities, and goals.

1. Who TCE is

TCE is a dynamic and innovative firm based in Tunisia that serves as a catalyst for organizational change. It was created by one person and has since grown to a team of 6 experts. TCE specializes in consulting, training workshops, entrepreneurship, international cooperation, and other fields of expertise. The firm has collaborated with various stakeholders and partners, including the Ministry of Industry & SMEs, US Embassy in Tunisia, Indian Embassy in Tunisia, Korean Embassy in Tunisia, and others.

2. What TCE does

TCE provides consulting services and training workshops in various fields of expertise, including entrepreneurship, international cooperation, and business planning and modelling. TCE has successfully delivered more than 35 consultancy missions and projects in less than 4 years. The firm is also involved in various projects aimed at contributing to the development of the ecosystem in Tunisia, such as the Tunisian Diaspora of Startups and the SWITCHMED program.

3. How TCE does it

TCE collaborates with various stakeholders and partners, including ministries, embassies, coworking spaces, and private initiatives, to provide technical assistance, networking events, and entrepreneurship programs to businesses and startups.

Figure 1: TCE Entreprise logo

The firm identifies, structures, and reaches out to enablers of the digital community with the aim to boost job creation within technology-based startups. TCE also provides technical assistance to businesses to improve their business plans, discover new creative methods of fundraising, and eventually expand and onboard new employees.

II. Project overview

The aim of this section is to accomplish three objectives: firstly, to provide a clear definition of the problem that the internship aims to solve; secondly, to propose a deliverable solution; and thirdly, to outline the approach that will be taken to achieve the desired objectives.

1. General context

The internship is the senior year project at Fseg Tunis in order to obtain a bachelor degree in Business Intelligence (BI) for the academic year 2022-2023.
The internship took place at the company “TCE Entreprise” from the February to May.

2. Analysis of the current state

Before the introduction of the project, the quality assurance department in the company was mainly relying on manual testing for the WorkerHero application. The process was time-consuming and prone to human error. Moreover, as the application continued to grow and evolve, the manual testing process became increasingly complex and challenging to manage.

3. Problem definition

The main problem faced by the quality assurance department was the inefficiency of the manual testing process. As the application continued to expand, it became difficult to ensure that every component and feature was tested thoroughly. Additionally, the manual testing process was error-prone, which resulted in bugs and issues being overlooked or missed. This, in turn, led to an increase in the number of customer complaints and a decrease in overall satisfaction.

4. Proposed solution

The proposed solution was to implement full-stack testing, which includes both manual and automated testing. Manual testing would be conducted using TestRail and Jira, while automated testing would be carried out using Cypress. This approach would allow the testing team to identify bugs and issues more efficiently and ensure that every feature and component of the application is tested thoroughly. Additionally, the use of TestRail and Jira would enable the team to track their progress and identify areas that require further attention. Finally, the monitoring and reporting via Github and Grafana would allow for faster and more accurate reporting of any issues or bugs that arise.

5. Project outline

The project will consist of six phases:

- Planning and Preparation: In this phase, the testing team will conduct a detailed analysis of the application's features and components, identify the areas that require testing, and create comprehensive testing plans and scripts. The team will also assess the necessary resources, including tools, equipment, and personnel.
- Implementation - Manual Testing: In this phase, the testing team will execute the manual testing plans and scripts to identify any issues or bugs. They will also document their findings and make any necessary changes to the testing plan.
- Implementation - Automated Testing: In this phase, the team will automate the testing process using Cypress, which will involve creating test cases and running them automatically. The team will also integrate the automation testing with TestRail and Jira to manage and track the test cases.

- Reporting: The team will utilize Github in this step to produce and publish reports on the testing progress and issues discovered throughout the testing process. The reports will consist of in-depth analysis and insights, such as graphs, charts, and visualizations, to provide stakeholders with a comprehensive understanding of the testing process.
- Monitoring: In this stage, the team will use Grafana to observe and track the application's performance during testing. This will require creating dashboards to track metrics such as response times, error rates, and other crucial performance indicators to guarantee that the application meets the required performance standards.
- Analysis and Issue Resolution: The team will examine the testing outcomes in this final stage and detect any remaining issues or bugs. They will prioritize these issues based on their severity and impact and address them accordingly. Additionally, the team will document their discoveries and report the results to the stakeholders.

Figure 2 : Project outline

6.Work Methodology

As a member of TCE Entreprise's QA team, we adhere to an Agile methodology to manage our projects and provide exceptional software products. Our QA Manager acts as a Scrum Master and offers support to help us achieve our objectives.

We conduct daily stand-up meetings to discuss our progress, challenges, and priorities, ensuring that we are aligned and working towards the same goals. We also hold sprint planning, review, and retrospective meetings to manage, assess, and enhance our work.

To facilitate our Agile methodology, we employ various tools and technologies to work more effectively and efficiently. For instance, we utilize JIRA to manage our project backlog, track issues and defects, and assign tasks to team members. Similarly, we use Confluence to document our processes, procedures, and test cases, and share our knowledge and insights with other team members.

Furthermore, we prioritize continuous improvement as part of our Agile methodology and strive to deliver high-quality software products to our clients. We conduct frequent testing, both manual and automated, to identify and resolve defects early in the development cycle. Additionally, we use test automation frameworks such as Cypress to ensure that our testing is consistent, dependable, and scalable.

In conclusion, our Agile methodology enables us to deliver high-quality software products on time and within budget, while also promoting a culture of collaboration, innovation, and continuous improvement.

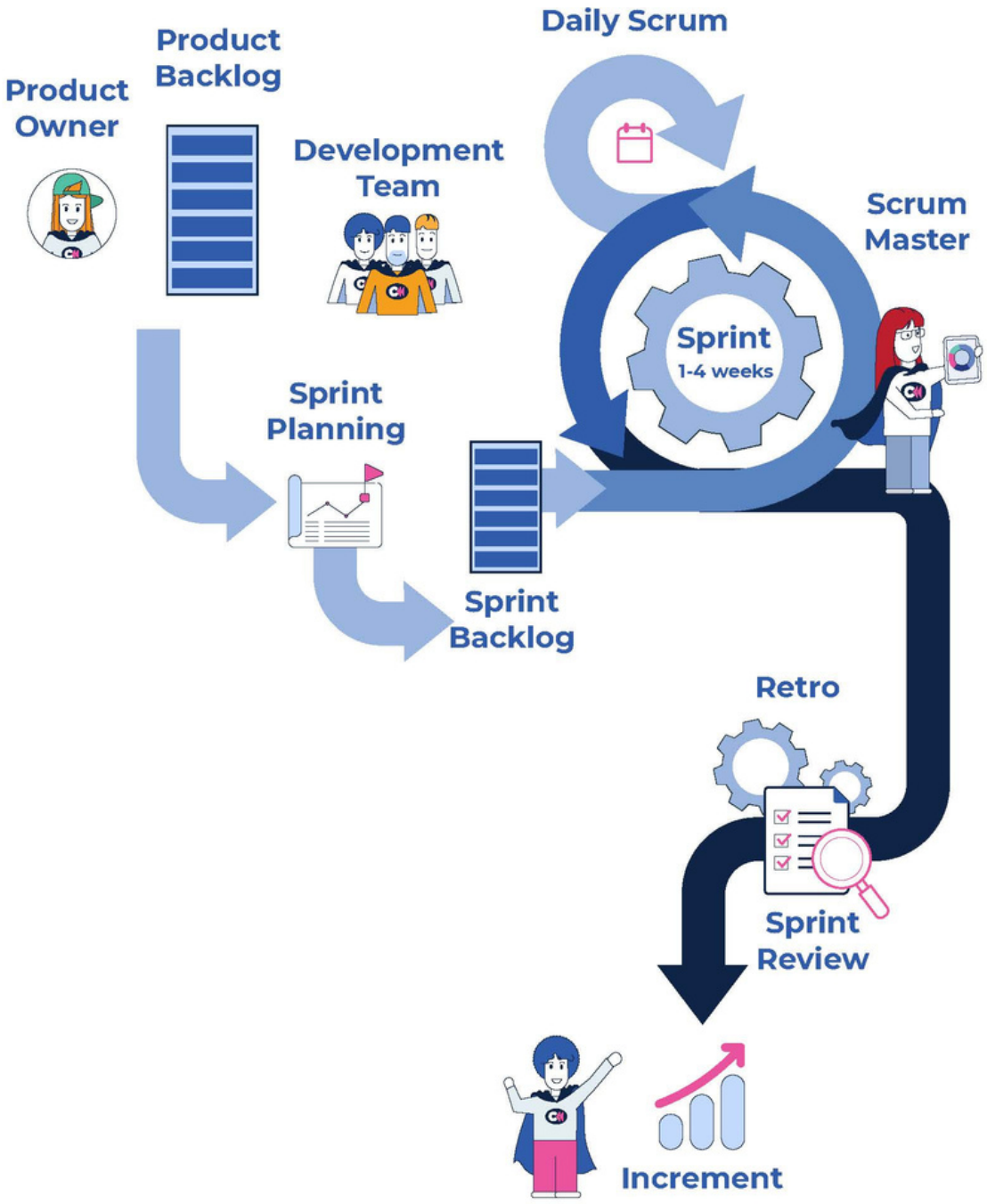


Figure 3 : Scrum Process

Conclusion

This chapter has given a summary of the project, which includes details about the company, its context, and the issue that requires resolution. The next chapter will delve into the preliminary study and the current status of the project.

Chapter 2

Preliminary study

I.Preliminary Study

The purpose of this chapter is twofold: firstly, to examine the preliminary study that was conducted to identify the problem encountered by the quality assurance department; and secondly, to explore the concept of software quality, testing, and software testing in greater detail. Furthermore, we will examine the various levels and types of software testing, and differentiate between a test plan and a test strategy. Finally, the chapter will conclude with a discussion of the test process and acceptance testing.

1.Software Quality

The concept of software quality pertains to the degree to which a software product meets its specified requirements and is suitable for use. This encompasses a variety of factors, such as functionality, performance, reliability, maintainability, usability, and security.
Guaranteeing software quality is critical to providing a product that satisfies customer expectations and preventing the detrimental effects of software defects, which can include higher costs, customer dissatisfaction, and harm to the brand's image.

2.Testing

The act of testing involves evaluating a software product or system to ensure that it complies with the specified requirements and operates as intended. Testing can be conducted at various points in the software development lifecycle, including unit testing, integration testing, system testing, and acceptance testing.

Testing is a vital part of software quality assurance because it aids in the early detection of flaws and problems during the development process, when they are less expensive and simpler to rectify.

3.Software Testing

Software testing is a specialized type of testing that concentrates on assessing software products or systems. It employs a variety of techniques and approaches to verify that the software satisfies the specified requirements and operates as intended.

A. Level of Software Testing

Software testing can be categorized into different levels depending on the scope and goals of the testing. These levels include:

- Unit Testing: This level of testing involves examining individual components or modules of the software to ensure that each unit operates as intended.
- Integration Testing: This level of testing involves verifying the integration between different modules of the software to ensure that they work together correctly.
- System Testing: This level of testing entails evaluating the entire system as a whole to ensure that it satisfies the specified requirements.
- Acceptance Testing: This level of testing is performed by the end-users or the client to ensure that the software fulfills their expectations and requirements.

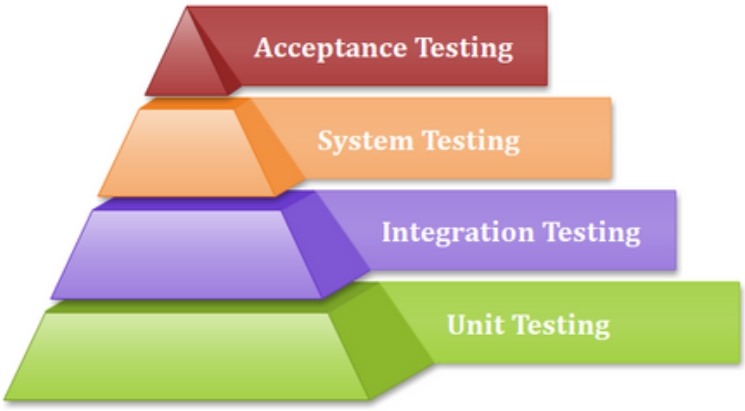


Figure 4 : Hierarchy Of Software Testing Levels

B. Types of Software Testing

Software testing can also be categorized into various types based on its objectives and techniques. These types include:

- Functional Testing: This type of testing evaluates the software's functionality to ensure that it operates as intended.
- Non-functional Testing: This type of testing examines the non-functional aspects of the software, such as performance, security, usability, compatibility, and reliability.
- Regression Testing: This type of testing verifies that changes or modifications made to the software do not have any adverse effects on the existing functionalities.
- Smoke Testing: This type of testing ensures that the critical functionalities of the software are working correctly before proceeding with further testing.
- Exploratory Testing: This type of testing involves exploring the software without any pre-planned test cases to identify defects.
- Ad-hoc Testing: This type of testing entails identifying defects without any predefined test cases or plan.

4.Bugs or Anomalies

In software testing, errors, flaws, or faults can be discovered in the software, commonly known as bugs. Bugs cause the software to behave in ways that were not intended. It is critical to detect and report any bugs found during testing to the development team. This enables them to fix the bugs before the software is released to end-users.

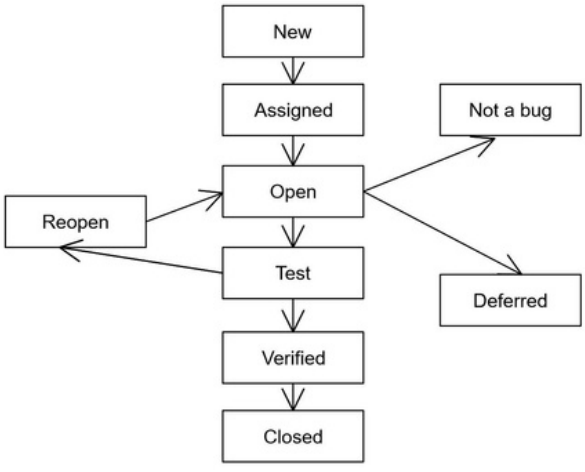


Figure 5 : Bug Life Cycle

The presented table outlines various types of anomalies and their corresponding definitions, providing a useful framework for identifying and categorizing issues that may arise during software testing and development.

Anomaly Level	Definition
Critical	An issue that completely blocks the intended functionality of the application or causes a system crash.
Major	An issue that affects the functionality of the application or significantly impacts the user experience.
Minor	An issue that has a minimal impact on the functionality of the application or user experience, but still needs to be addressed.
Cosmetic	An issue that does not impact the functionality or user experience of the application, but could be visually unappealing or confusing to the user.

Table 1: Anomaly Levels and Definitions

5.Difference between Test Plan and Test Strategy

Testing is an essential process that involves the assessment of a software product or system to verify its compliance with specified requirements and intended functionality. This process can be conducted at various stages throughout the software development lifecycle, including unit testing, integration testing, system testing, and acceptance testing.

A. Test Plan

A crucial document in software testing is the test plan, which provides a comprehensive outline of the objectives, scope, approach, and resources required for testing a software application. This document includes vital information about the various types of testing that will be performed, the timeline for each testing phase, the testing environment, and the roles and responsibilities of the testing team members.

B. Test Strategy

A test strategy is a high-level document that defines the overall testing approach for a software application, including testing objectives, scope, techniques, and types. Additionally, it outlines the risks associated with the software and the measures taken to mitigate them. The primary distinction between a test plan and a test strategy is that a test plan is a comprehensive document that provides specific information about the testing process, whereas a test strategy is a broad document that provides an overview of the testing approach. While the test plan is focused on the particulars of testing, the test strategy concentrates on the general testing goals and approach.

6.Test Process

To ensure the quality and dependability of the software being tested, the test process involves several stages that must be followed.

Figure 6 :Test Process Stages

A.Test Requirements

The initial stage of the testing process involves defining the software's requirements that need to be tested. During this stage, the testing team reviews the software requirements document to identify the testable features and functionalities. This is done to ensure that the testing is in line with the software's intended purpose and meets the end-users' needs.

B.Test Planning

Test planning is a crucial stage in the testing process, which involves several tasks such as determining the scope of the testing effort, identifying the test environment, creating a test plan document, and identifying the resources needed for testing. Additionally, test planning includes defining the test objectives, strategies, and cases that will be employed during the testing process. To ensure the success of the testing process, it is essential to create a comprehensive test plan that outlines a detailed timeline for testing and includes contingency plans to address any issues that may arise.

C. Analysis and Design of Test Cases:

During the test design stage, the testing team analyzes the software requirements and creates test cases that cover all possible scenarios. The test cases should be comprehensive and cover all possible inputs, outputs, and conditions. Additionally, the team identifies the appropriate test environment and any necessary tools required to execute the tests.

D. Execution of Test Cases:

The test execution stage involves running the test cases that were designed in the previous stage and recording the results. The testing team should ensure that all the test cases are executed systematically, and the results are recorded accurately. During this stage, any issues or bugs that are identified are documented and reported to the development team for fixing.

E. Retesting and Regression Testing:

Retesting refers to the process of rerunning the test cases that failed during a previous testing cycle to ensure that they have been fixed properly. The objective of retesting is to confirm that the fixes have resolved the identified issues or bugs effectively.

Regression testing, on the other hand, refers to the process of rerunning the test cases that passed during previous testing cycles to ensure that they still pass after new changes or updates have been made to the software. The primary objective of regression testing is to ensure that any new changes or updates to the software do not have unintended impacts on the previously tested and functional areas of the software.

F. Release Testing:

Release testing ensures that software is ready for production deployment after passing all necessary quality checks. The objective is to verify stability, security, performance, and completeness of the release package, configuration settings, and release documentation. Acceptance testing is then conducted in the production environment to confirm end-user requirements are met and that the software works well with other systems and software.

7. Software Recipe

A software recipe is a high-level plan for the software development process, including the necessary steps, activities, and tasks, as well as the required resources, tools, and technologies. It serves as a guide throughout the project to ensure the software is developed to meet quality standards, time frames, and resource requirements.

- Context
  - Overview of the project, its purpose, and its scope
  - Description of the stakeholders involved
  - Any constraints or limitations that may affect the project

- Objectives
  - High-level goals and objectives of the project
  - Specific deliverables or outcomes that must be achieved

- Activities
  - High-level goals and objectives of the project
  - Breakdown of the development process into distinct activities or phases
  - Listing of the tasks required for each activity or phase
  - Identification of any dependencies or prerequisites for each task

- Results
  - Key performance indicators or metrics used to measure project progress and success
  - Expected results or outcomes for each phase or activity
  - Evaluation criteria for measuring the quality of the final product

- Conclusion
  - Summary of the key points covered in the software recipe
  - Statement on the overall feasibility and practicality of the project
  - Recommendations for further action or improvements as needed.

II. Test Automation Tools

1. Overview of Existing Methods

A.Introduction to Test Automation Tools

Test Automation Tools refer to software programs or applications that enable automated testing of software applications or systems. These tools can perform various types of tests such as functional, performance, and regression testing.

B. Benefits of Test Automation Tools

Another benefit of using Test Automation Tools is that they can help increase test coverage, as they are capable of executing a large number of test cases in a shorter amount of time than manual testing. This can help ensure that all areas of the software application or system are thoroughly tested. Additionally, Test Automation Tools can help improve collaboration between different teams involved in the software development process, as they provide a common platform for testing and reporting issues. Finally, Test Automation Tools can help reduce the overall cost of testing, as they can be used to identify issues earlier in the development process, when they are typically less expensive to fix.

C. Common Test Automation Tools:

- Selenium WebDriver:

Selenium WebDriver is a widely used open-source software tool that enables the automation of web browsers. It is compatible with multiple programming languages, including Java, C#, Python, and Ruby, making it accessible to a wide range of developers and testers.

Figure 7 : Selinium Logo

- Appium:

Appium is an open-source mobile automation tool that supports multiple programming languages for Android and iOS platforms.

Figure 8 : Appium Logo

- TestComplete:

TestComplete is a paid automation tool that tests desktop, web, and mobile apps and supports programming languages such as Python, VBScript, and JavaScript.

Figure 9 : TestComplete Logo

- Cypress:

Cypress is an open-source tool used for end-to-end testing of web applications. It is built on top of Node.js and supports JavaScript.

Figure 10 : Cypress Logo

2. Selection Criteria for Test Automation Tools

When selecting a test automation tool, there are several factors to consider. The following are some of the most important selection criteria:

- Usability and Ease of Use: The tool should be easy to use and require minimal training for testers to get started.
- Integration with Development Tools: The tool should integrate easily with the development environment and testing frameworks.
- Flexibility and Customization: The tool should be flexible enough to handle various types of testing and allow customization to meet specific needs.
- Cost and Licensing: The tool should be cost-effective and have a licensing model that suits the organization's needs.
- Test Maintenance and Support: The tool should have good support and maintenance options to ensure that issues can be resolved quickly and efficiently.

3. Comparison between Cypress and Selenium WebDriver

A.Overview of Cypress and Selenium WebDriver

Cypress is a fast and user-friendly test automation tool based on JavaScript, while Selenium WebDriver is an open-source tool supporting multiple programming languages.

B.Comparison

Selection Criteria	**Cypress**	**Selenium WebDriver**
Usability and Ease of Use	High	Medium
Integration with Dev Tools	High	High
Flexibility and Customiz.	High	Medium
Cost and Licensing	Free	Free/Open Source
Test Maintenance and Supp.	High	High

Table 2: .Comparison Table on Selection Criteria

C. Test Automation Tool Choice

According to the comparison table, Cypress seems to be the superior option for test automation tools due to its excellent performance in all selection criteria, except for cost and licensing, which is still reasonable compared to other commercial tools. It provides a more convenient and adaptable testing experience, which can save both time and effort in the long term. Overall, it is a better choice for those seeking a more user-friendly and flexible testing solution.

Conclusion

In this chapter, we learned about the significance of conducting a preliminary study in software engineering before beginning a software development project. We also discussed the various stages of the testing process and the crucial role of selecting appropriate test automation tools. The next chapter will focus on the "Analysis and Specification of Requirements."

Chapter 3

Analysis and Specification of Requirements

I. Specification of Requirements

The first step of the upcoming chapter is to identify the actors involved in the project and specify both functional and non-functional requirements. After this, we will proceed to model the requirements using a use case diagram.

1. Identification of Actors

An actor is a person, hardware, or software that interacts with the system to perform one or more functions related to the use cases.

The actors involved in our system are:

- QA manager:

The QA manager actor in our team identifies obstacles encountered by the team and tries to resolve them while ensuring that the root causes of the problems are eliminated. They monitor progress in relation to the plan and implement control actions to achieve objectives.

- Functional QA:

The main actor in our project is the Functional Tester. They have the necessary administrative rights to use the services offered by our solution.

- Automation QA:

The Automation QA actor can write automation scripts, develop test cases, execute automated test cases, and review bug reports

2.Functional Requirements

Our solution must meet the functional requirements that we will present by actor:

- QA manager:

Monitor progress against what was defined in the plan and implement control actions to achieve objectives.

- Functional Tester:

All necessary information about documentation, design, and requirements must be available to enable testers to run the system and judge correct behavior.

Plan a test plan by choosing the date and time of execution. However, results can be extracted at any given time and then continue testing.

Monitor the progress of the execution to view and analyze the execution progress of the test case step by step.

- Automation Tester:

All test hardware platforms must be properly installed and configured to function correctly.

All standard software tools, including testing tools, must be successfully installed to function properly.

Automate test cases.

Execute tests in parallel on multiple environments.

Run tests on multiple web browsers.

Generate execution reports after running a test case.

- The system must automatically generate detailed reports indicating the steps of the scenario executed. In case of an error, a screenshot will be displayed to the user to indicate the location of the error.

- Analyze a test report containing all necessary information such as the status of each passed or failed step. This will help the user to narrow down the source and know the cause by using the logs written in the report.

3.Non-Functional Requirements

Performance: This project must provide speed and velocity reports to help improve performance.

Accessibility: Ensuring accessibility for all users, including those with disabilities, is crucial alongside providing speed and velocity reports to improve performance.

Maintainability: Compliance with coding standards, attribute and variable names, method names, and the arrangement of comments to have readable and well-structured code.

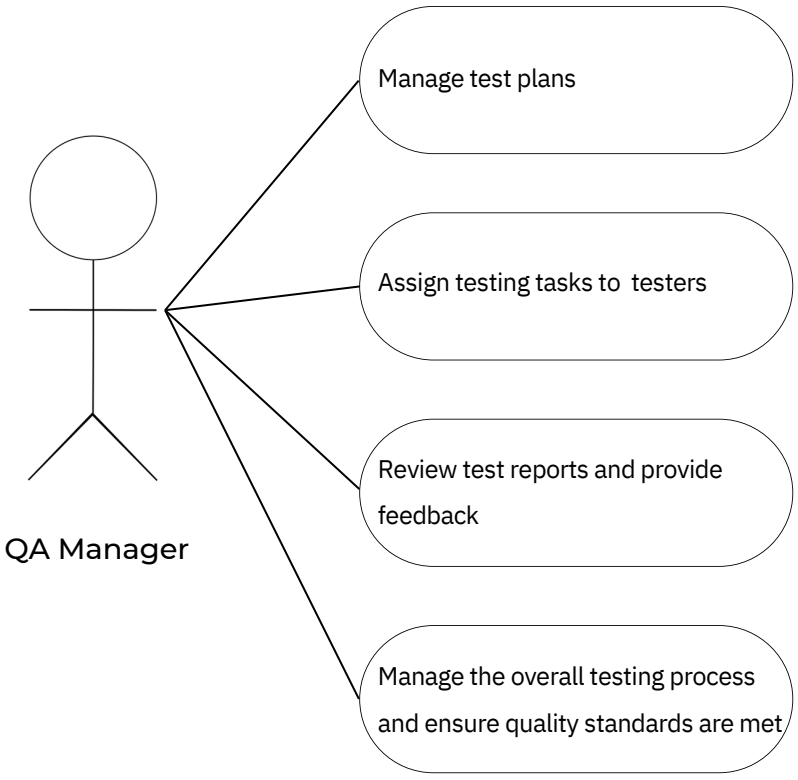
Modularity: The solution must be designed in independent modules to ensure low coupling and application scalability and simplicity of maintenance. For example, we developed the various sprints separately.

II. Modeling of Requirements

In software engineering, a use case diagram is a visual representation of the behavior of a system as perceived by its users, depicting their actions and expected outcomes. In the upcoming section, we will provide an overview of the functions that our application offers by using a use case diagram to present a global view. Subsequently, we will focus on refining the most critical use cases.

1.Global Use Case Diagram

The diagram below shows the different actors and their interactions with the system:



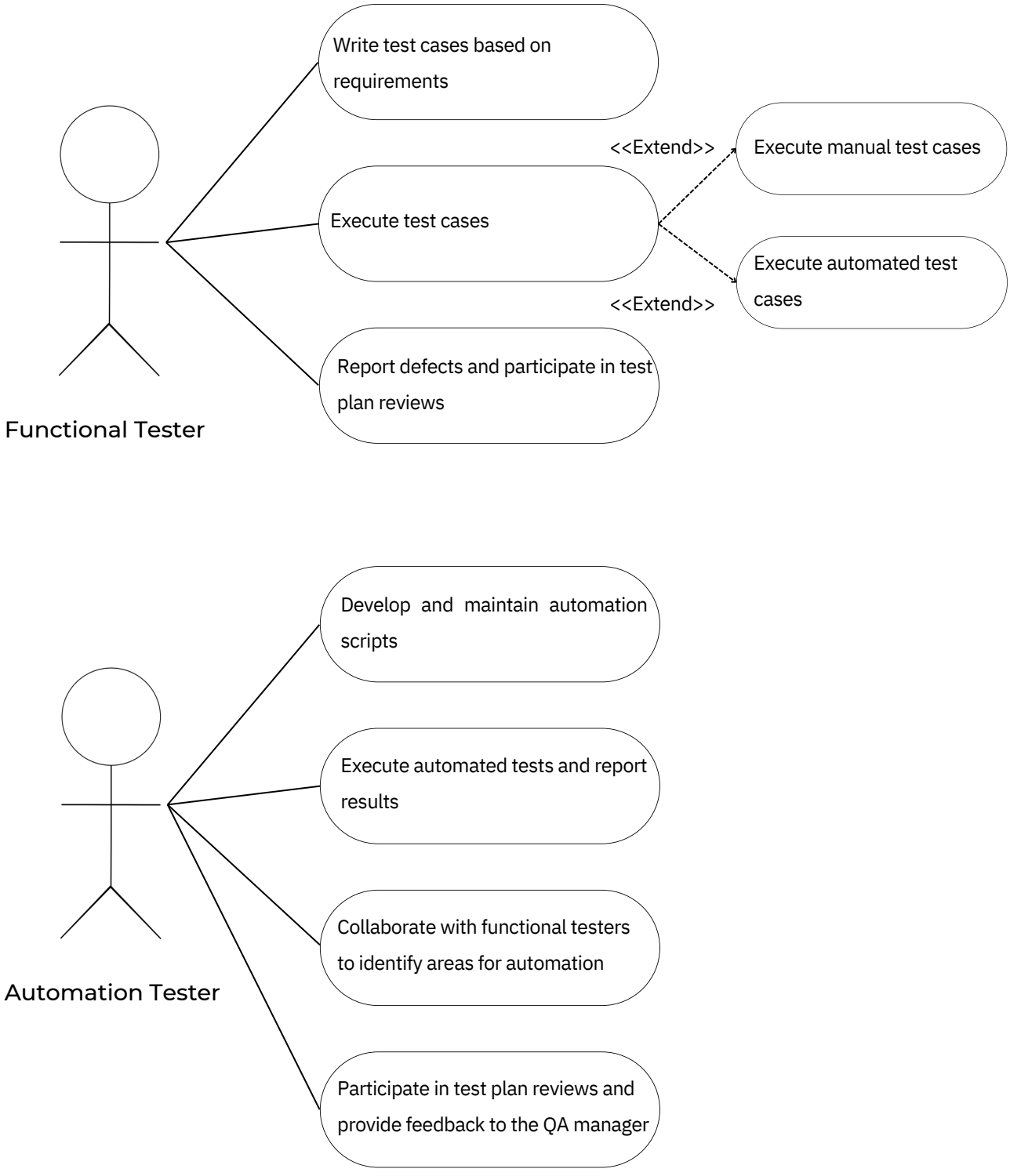


Figure 11 : Global Use Case Diagram

2.Textual Description of Use Cases for "Functional Tester" Actor

The upcoming section will primarily emphasize the creation of written descriptions for each use case, formulation of test cases, implementation of said test cases, evaluation of test results, and the development of automation scripts for both the "Functional Tester" and "QA Automation" roles.

A.Textual Description of "Writing Test Cases" Use Case

Title	**Writing Test Cases**
Summary	The Functional Tester will generate test cases utilizing exploratory testing techniques to encompass all possible requirement scenarios.
Actor	Functional Tester
Precondition	The tester launches a test management tool.
Postcondition	Test cases have been successfully written.
Nominal Scenario	1. The functional tester understands the client's requirements. 2. They prepare the test management tool. 3. They start writing test cases.
Alternative Scenario	1. The functional tester does not understand the client's requirements. 2. Requirements have changed, so test cases are incorrect.

Table 3:Textual Description of "Writing Test Cases" Use Case

B. Textual Description of "Executing Test Cases" Use Case

Title	**Execution of a Test Case**
Summary	Launching the execution of a test case by setting the execution environment, browser, and scenario
Actor	Functional Tester
Precondition	The tester has to execute the test case
Postcondition	The execution of the test case has been successfully completed
Nominal Scenario	1. The functional tester selects the test case to execute 2. The tester selects the web browser for the test case execution 3. The tester chooses the execution environment 4. The system launches the execution of the test case
Alternative Scenario	N/A (Not Applicable)

Table 4: Textual Description of "Executing Test Cases" Use Case

B. Textual Description of "Analyzing a Test Report" Use Case

Title	**Analyzing a Test Report**
Summary	The tester analyzes the test report to check the status of the steps and correct any errors if necessary.

Actor	Functional Tester
Precondition	The tester has the test report to analyze.
Postcondition	The test case analysis has been successfully performed.
Nominal Scenario	1. The functional tester selects the test report to analyze. 2. The tester checks the status of the steps. 3. The tester selects the steps with errors. 4. The tester corrects the errors. 5. The tester re-executes the test.
Alternative Scenario	1. The functional tester analyzes the test report. 2. The tester moves on to the next test case execution.

Table 5: Textual Description of "Analyzing Test Report" Use Case

3.Textual Description of Use Cases for "QA Automation" Actor

A. Textual Description of "Writing Automation Scripts" Use Case

Title	**Writing Automation Scripts**
Summary	The automation QA will prepare automation scripts .
Actor	Automation QA

Precondition	Automation QA launches the scripts.
Postcondition	The script writing has been successfully completed.
Nominal Scenario	1. Automation QA understands the test cases from the functional tester. 2. They prepare the environment. 3. They start writing automation scripts.
Alternative Scenario	1. The functional tester analyzes the test report. 2. The tester moves on to the next test case execution.

Table 6 : Textual Description of "Writing Automation Scripts" Use Case

B. Textual Description of "Developing automation scripts" Use Case

Title	**Developing automation scripts**
Summary	QA Automation develops automation scripts.
Actor	QA Automation
Precondition	QA Automation starts development.
Postcondition	Development has been completed successfully.
Normal Flow	QA Automation prepares the development environment.

Table 7 : Textual Description of "Developing automation scripts" Use Case

C. Textual Description of "Executing Automated Test Case" Use Case

Title	**Executing Automated Test Case**
Summary	Launch the execution of a test case by setting the execution environment, browser, and scenario.
Actor	QA Automation
Precondition	QA Automation loads the page to be executed.
Postcondition	The test case execution was successful.
Nominal Scenario	1. QA Automation chooses the feature to be executed. 2. The system launches the script execution.
Alternative Scenario	1. The feature is incomplete for execution. 2. The feature is erroneous for execution.

Table 8: Textual Description of "Executing Automated Test Case" Use Case

III. Product Backlog

The product backlog serves as a crucial instrument for Agile teams to arrange a prioritized catalogue of customer requirements and desires. The backlog is established before sprint initiation and each user story includes the subsequent particulars:

- An exclusive identifier that increases for each new story.
- A name, which describes the projected functionality from the customer's standpoint.

- A comprehensive narrative of the story.
- The significance of the story, denoted by an integer assigned by the product owner that indicates its priority. The story's importance can be modified throughout the project's execution. The greater the importance value, the higher the priority for fulfilling the story.

IV. Sprint Planning

	Sprint	Sprint Objective	Period
Release 1	**1**	• **Test case development for Intro screens/ Registration functionalities** • **Backend and frontend testing** • **Bug detection and reporting** • **Test Results Reporting and Email Notification** • **Test automation scenario** • **Re-testing after bug fixing** • **Report generation for test runs**	12/02 - 21/02
	2	• **Test case development for Authentification functionality** • **Backend and frontend testing** • **Bug detection and reporting** • **Test Results Reporting and Email Notification** • **Test automation scenario** • **Re-testing after bug fixing** • **Report generation for test runs.**	22/02 - 03/03
	3	• **Test case development for Authentification functionality** • **Backend and frontend testing** • **Bug detection and reporting** • **Test Results Reporting and Email Notification** • **Test automation scenario**	04/03 - 13/03

Release 1	**3**	• **Re-testing after bug fixing** • **Report generation for test runs.**	14/03 - 23/03
	4	• **Test case development for Profile settings functionalities** • **Backend and frontend testing** • **Bug detection and reporting** • **Test Results Reporting and Email Notification** • **Test automation scenario** • **Re-testing after bug fixing** • **Report generation for test runs.**	24/03 - 02/04
	5	• **Test case development for Applicant Dashboard functionalites** • **Backend and frontend testing** • **Bug detection and reporting** • **Test Results Reporting and Email Notification** • **Test automation scenario** • **Re-testing after bug fixing** • **Report generation for test runs.** • **Workflow Implementation for CI/CD**	03/04 - 12/04
	6	• **Test case development for Applicant Profile functionalites** • **Backend and frontend testing** • **Bug detection and reporting** • **Test Results Reporting and Email Notification** • **Test automation scenario** • **Re-testing after bug fixing** • **Report generation for test runs.** • **Test execution statistics dashboard creation**	13/04 - 22/04

Release 1	**7**	• **Test case development for Employer Dashboard/Recruiting functionalities** • **Backend and frontend testing** • **Bug detection and reporting** • **Test Results Reporting and Email Notification** • **Test automation scenario** • **Re-testing after bug fixing** • **Report generation for test runs**	23/04 - 02/05

	8	• **Authentification Performance Testing:**
		  ◦ **Key Performance Indicators (KPIs) Identification**
		  ◦ **Test Scenario Definition**
		  ◦ **Performance Testing Execution**
		  ◦ **Performance Metrics Analysis**
		• **Home Page Accessibility Testing:**
		  ◦ **Accessibility Standards Identification**
		  ◦ **Accessibility Testing Execution**
		  ◦ **Accessibility Violations Analysis**
		  ◦ **Test Results Documentation**
			03/05 - 12/05

Table 9: Sprint Planning

Conclusion

In this chapter, we have established the boundaries of our application. Initially, we determined the prerequisites, followed by the formation of the product backlog and sprint planning for the project. Lastly, we constructed a comprehensive use case diagram to portray the context view of the project. The subsequent chapter will focus on the project's design phase.

Chapter 4

Technical Design of the Project

Introduction

In this chapter, we showcase the project design using a range of UML diagrams, including sprint backlog, sequence, activity, and state diagrams. We also provide technical details on CI/CD with monitoring and visualization, as well as performance and accessibility testing. Additionally, we present the planned physical and software architecture. Lastly, we discuss the technologies we will utilize.

I. Sprint backlog

Table 10 presents the sprint backlog containing the tasks to be developed:

ID	User Story	Task
ID-US1	**Launch functional testing for all the test cases.**	**Understand WorkerHero requirements.**
		Training on usable environments.
		Preparation of Backend testing environment (Postman).
		Test APIs.
		Execute manual tests in environments.
		Create a Jira bug ticket for the developer in case of bugs.
		Report test runs with TestRail

ID-US2	**Launch automation testing for all the test cases.**	**Install Cypress on local machine**
		Configure Cypress settings and create project structure.
		Define the test scenario for each user story.
		Create test cases based on the test scenario.
		Execute the tests using Cypress.
		Generate HTML and JSON reports of test results.
		Create Jira tickets for any identified bugs
ID-US3	**Continuous Integration/Continuous Deployment (CI/CD) with monitoring and visualization.**	**Configure Github Actions to run automated tests on code changes.**
		Generate test reports in JSON format during the Github Actions build.
		Publish the test results on GitHub Pages for easy access and visibility.
		Use an API to extract the data from the test report JSON file and send it to Grafana.
		Create a Grafana dashboard to display the test results in a user-friendly and informative way.
		Define objectives, scope, and metrics for the performance testing.
		Create a performance test plan and document the details of the testing process.

ID-US4	**Performance and Accessibility Testing process.**	**Install and configure the necessary, including JMeter and the Google Lighthouse Chrome extension.**
		Create a login scenario in JMeter that simulates 100 virtual users logging in simultaneously with a ramp-up period of 10 seconds.
		Execute the login scenario using JMeter to measure the performance of the login process.
		Analyze the results of the performance testing and identify any bottlenecks or performance issues.
		Create a performance testing report that includes the details of the testing process, results, and recommendations for improvements.
		Use the Google Lighthouse Chrome extension to perform accessibility testing on the home page web application.
		Analyze the results of the accessibility testing and identify any accessibility issues.
		Create an accessibility testing report that includes the details of the testing process, results, and recommendations for improvements.

Table 10: Sprint backlog

II. Needs Analysis

This chapter focuses on showcasing project design through various UML diagrams, starting with the sprint backlog. Additionally, we will elaborate on sequence, activity, and state diagrams and their relation to the technical, physical, and software architecture to be employed. Lastly, we provide an overview of the technologies that will be implemented.

1.Sequence diagram "Launching a functional test case"

Figure 12 illustrates the sequence diagram that portrays the testing interactions among functional environments, highlighting the key stages involved.

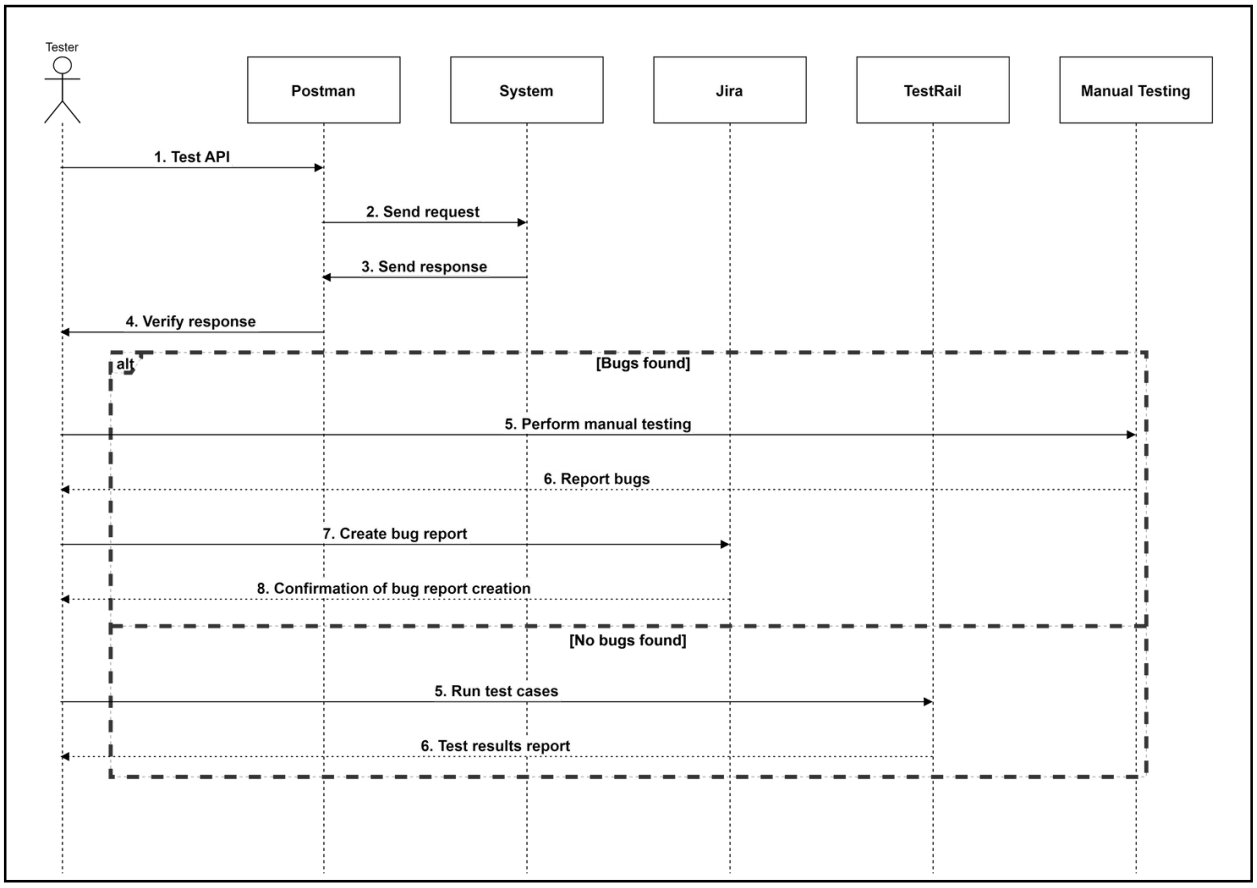


Figure 12:Sequence diagram "Launching a functional test case"

- **Steps :**
- Tester performs a functional test of the API using Postman.
- Postman sends the request to the System.
- The System sends the response back to Postman.
- Postman verifies the response and sends the results to the Tester.
- If bugs are found, the Tester performs manual testing to diagnose the issue.
- The Tester reports any bugs discovered during manual testing.
- The Tester creates a bug report in Jira.
- Jira sends a confirmation of bug report creation to the Tester.
- If no bugs are found, the Tester runs the test cases on TestRail.
- TestRail sends the test results report back to the Tester.

2.Sequence diagram "Automation of non-regression tests"

Figure 13's sequence diagram depicts the automation of non-regression tests through Cypress, with the added feature of detecting regressions during the test run.

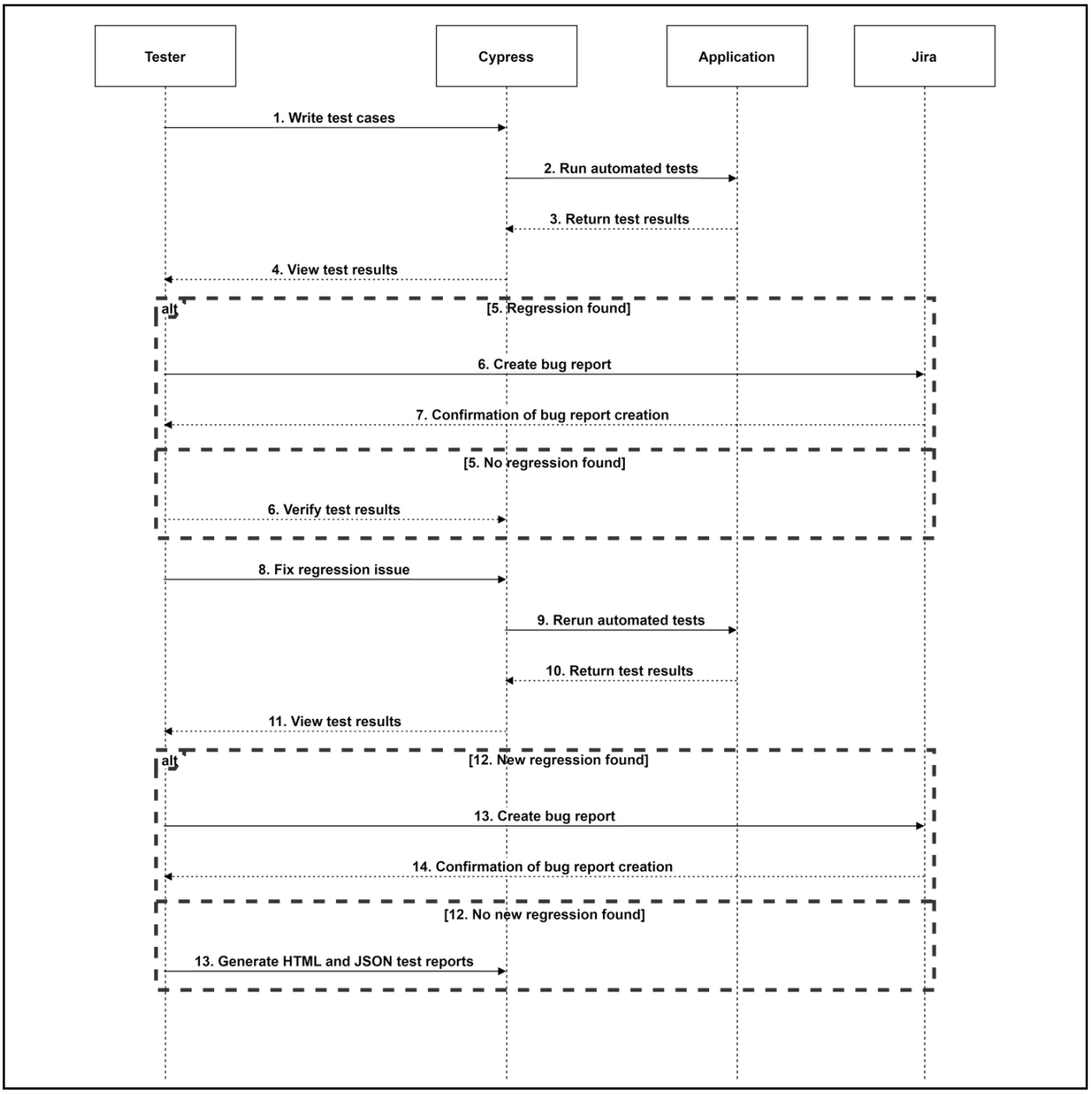


Figure 13:Sequence diagram "Automation of non-regression tests"

- **Steps :**
- The Tester runs the automated non-regression tests using Cypress.
- Cypress sends API requests to the System to perform the tests.
- Application returns the test results to Cypress.

- Cypress displays the test results to the Tester.
- If a regression is found, go to step 6; otherwise, go to step 6a.
- Tester creates a bug report in Jira.
- Jira confirms the creation of the bug report to the Tester. 6a. Tester verifies the test results.
- Tester fixes the regression issue.
- Cypress reruns the automated tests.
- Application returns the new test results to Cypress.
- Cypress displays the new test results to the Tester.
- If a new regression is found, go to step 13; otherwise, go to step 13a.
- Tester creates a bug report in Jira.
- Jira confirms the creation of the bug report to the Tester. 13a. Tester generates HTML and JSON test reports.

3.Activity Diagram

In Figure 14, we present the process of launching a test execution in an activity diagram.

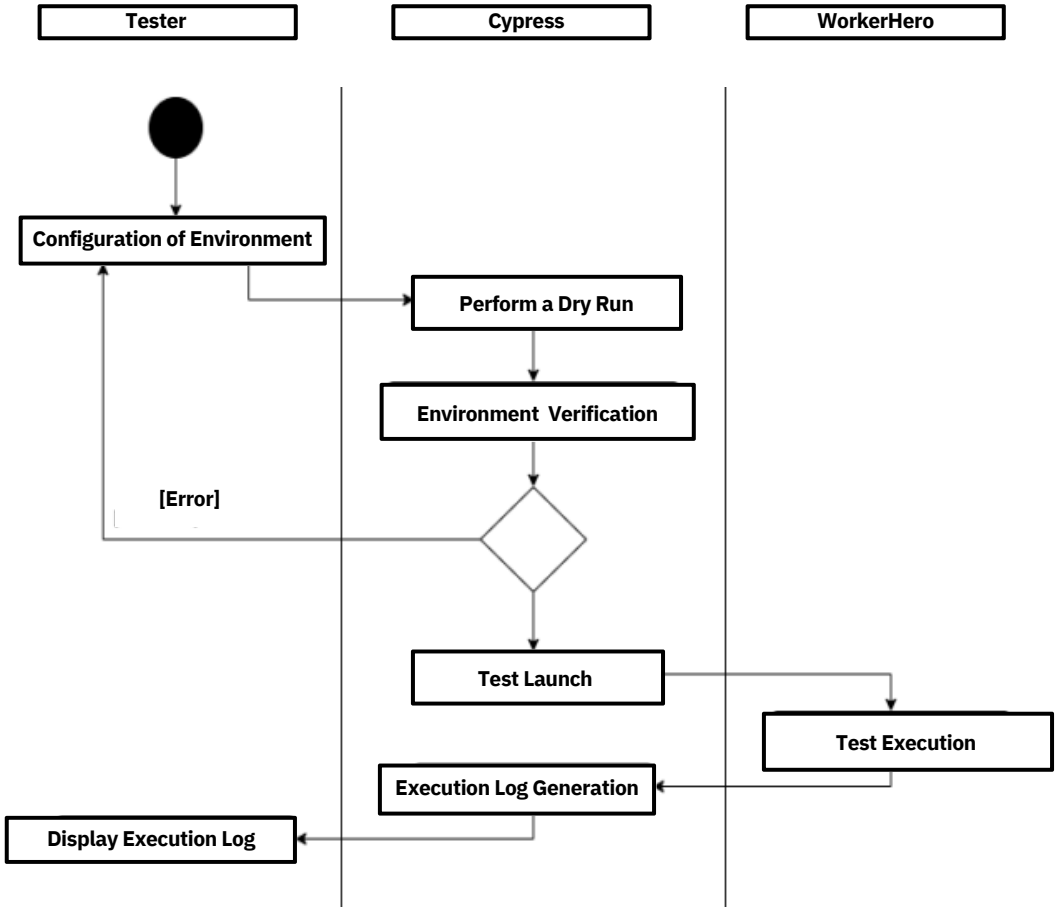


Figure 14 : Activity Diagram

4.State transition diagram for a test case

During the analysis phase of this sprint, Figure 15 portrays a state transition diagram that elucidates the various stages of a test case, showcasing the state transitions.

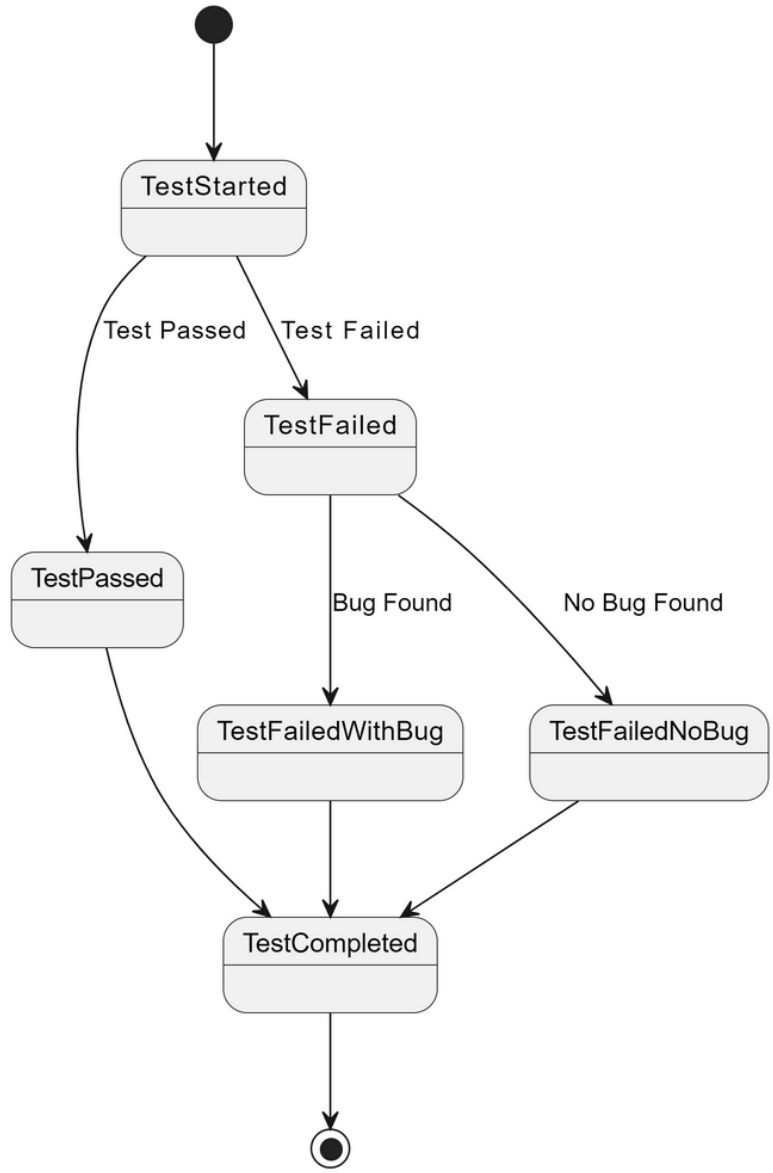


Figure 15: State transition diagram for a test case

5.CI/CD with Monitoring and Visualization Diagram

The depicted figure demonstrates the CI/CD process for the WorkerHero application, integrating continuous monitoring and visualization via GitHub and Grafana.

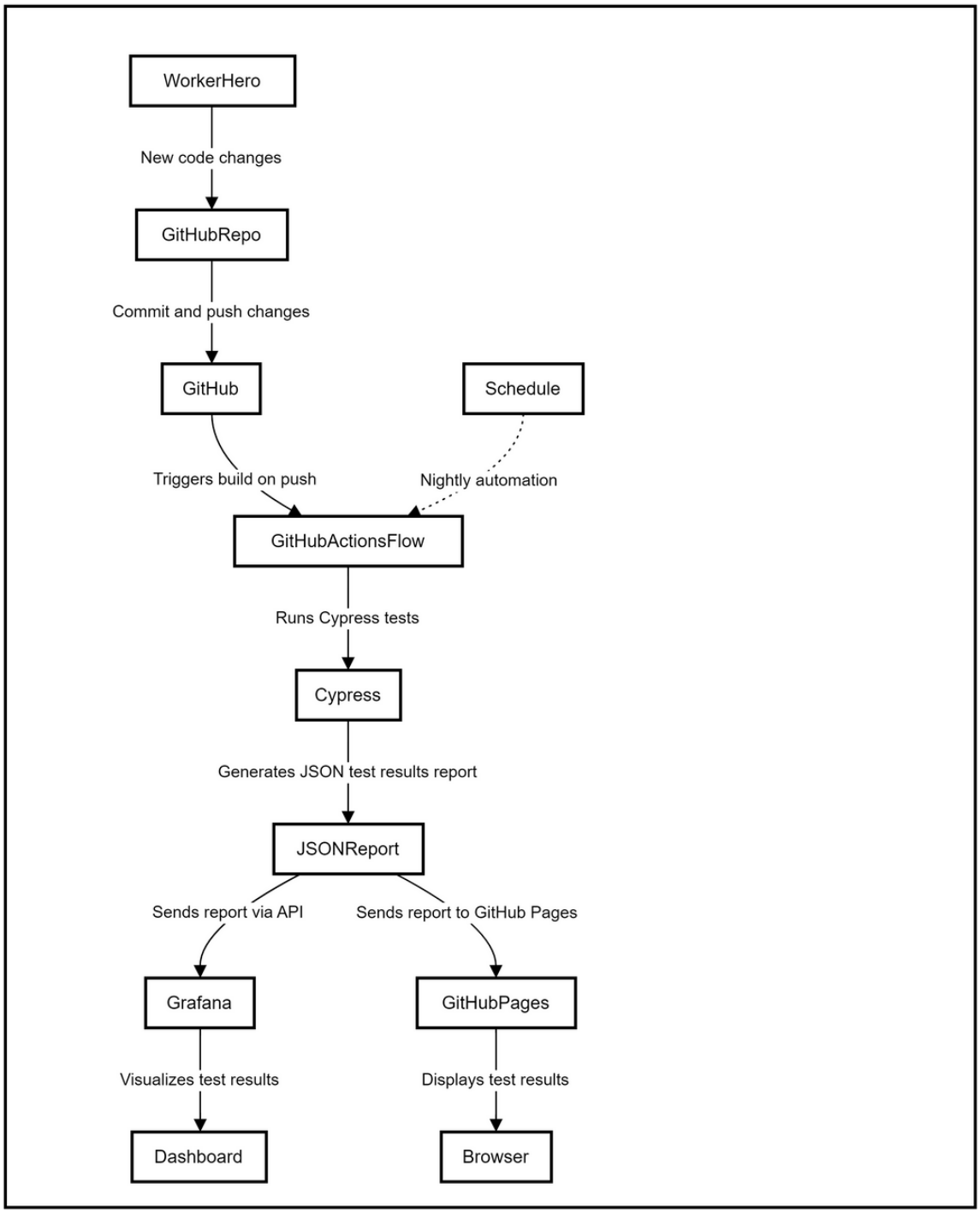


Figure 16: CI/CD with Monitoring and Visualization Diagram

6.Performance and Accessibility Testing Process Diagram

This diagram showcases the fundamental steps entailed in the process of performance and accessibility testing.

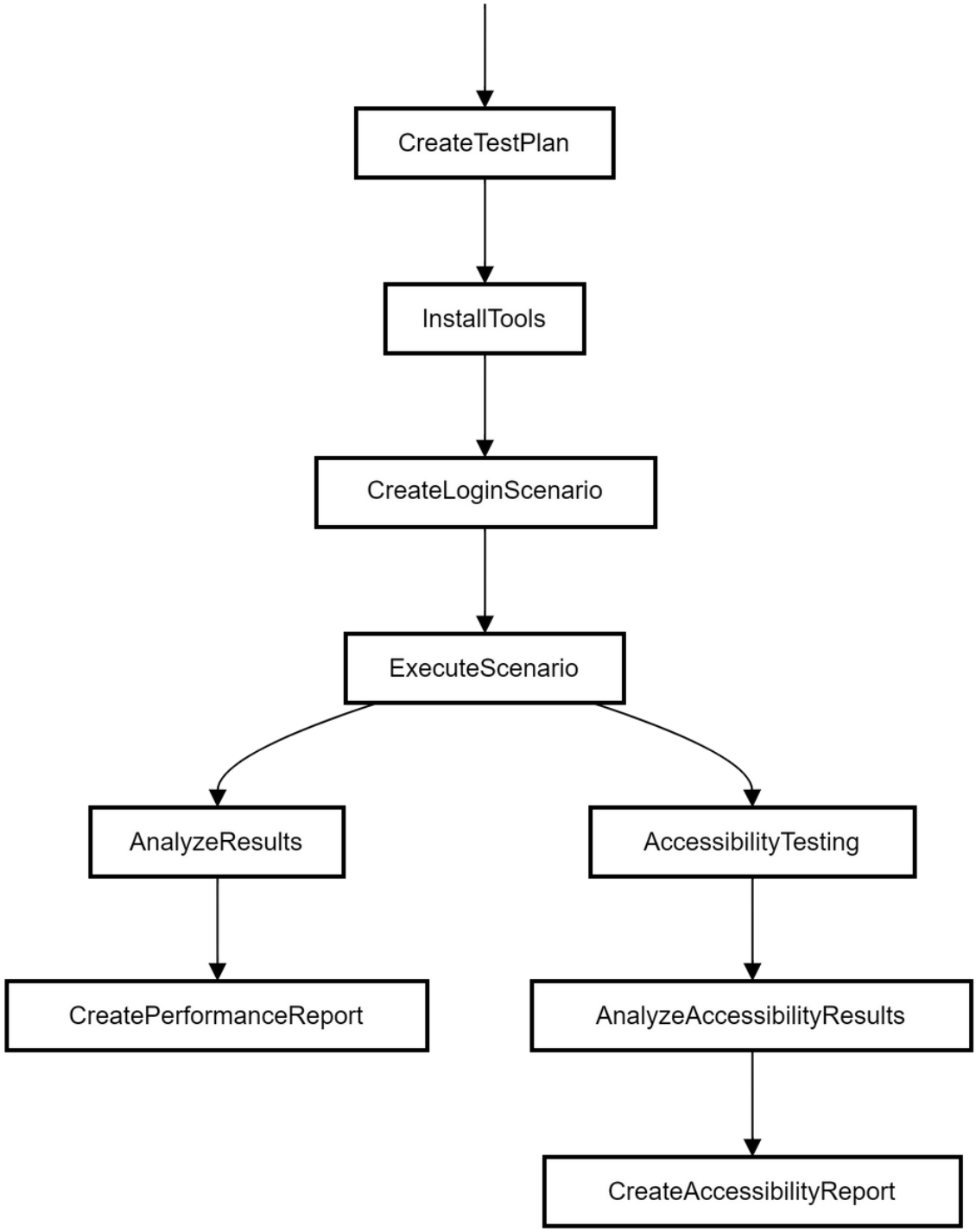


Figure 17: Performance and Accessibility Testing Process Diagram

III. Technical architecture

During the technical design phase of our project, we create a comprehensive physical and software architecture model to obtain a bird's eye view of our system.

Figure 16 showcases the technical architecture, exhibiting the overall system structure that comprises the testing stages and environments employed.

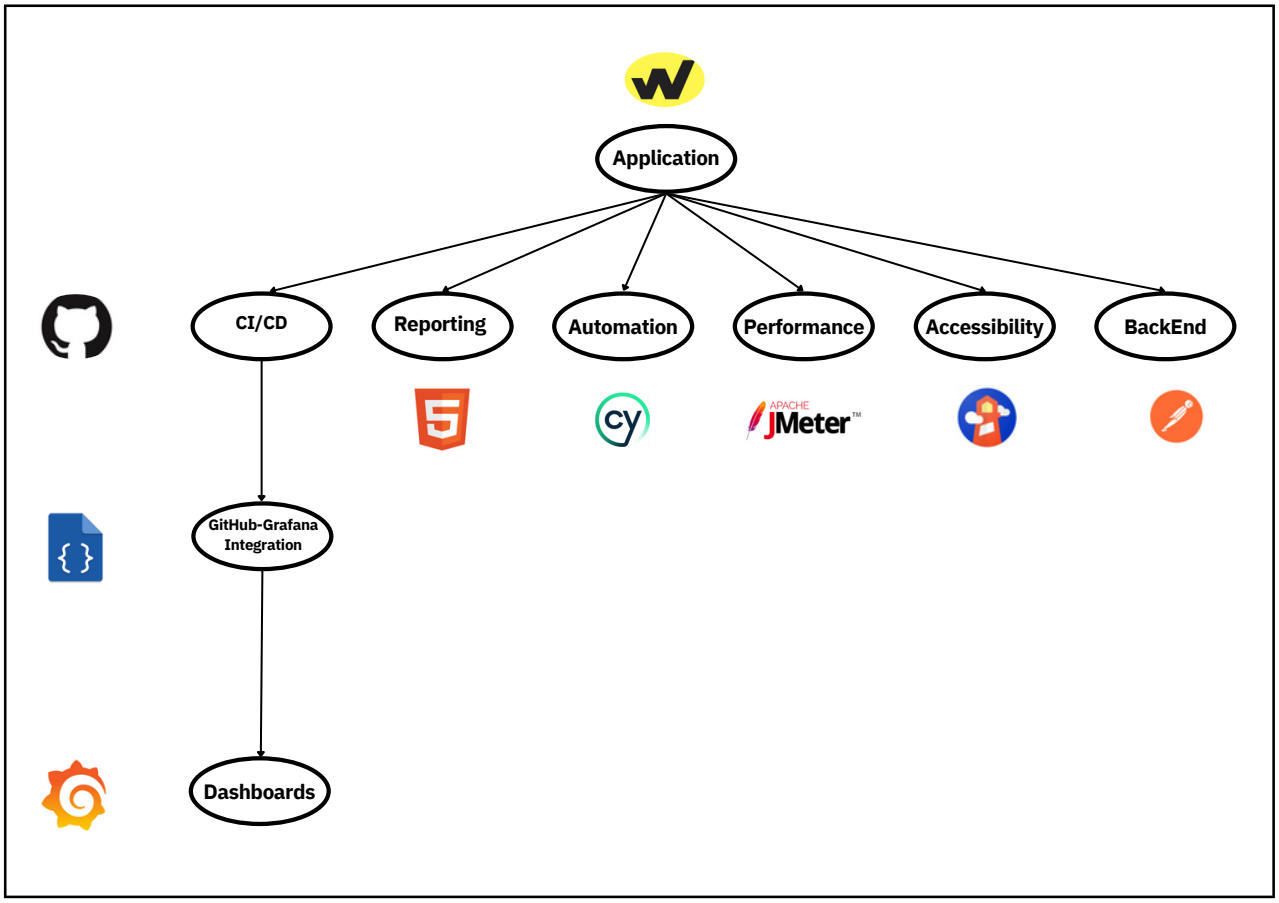


Figure 18: Project architecture

1.Physical architecture

This chapter commences with the presentation of the physical architecture of our project, depicted in Figure 17.

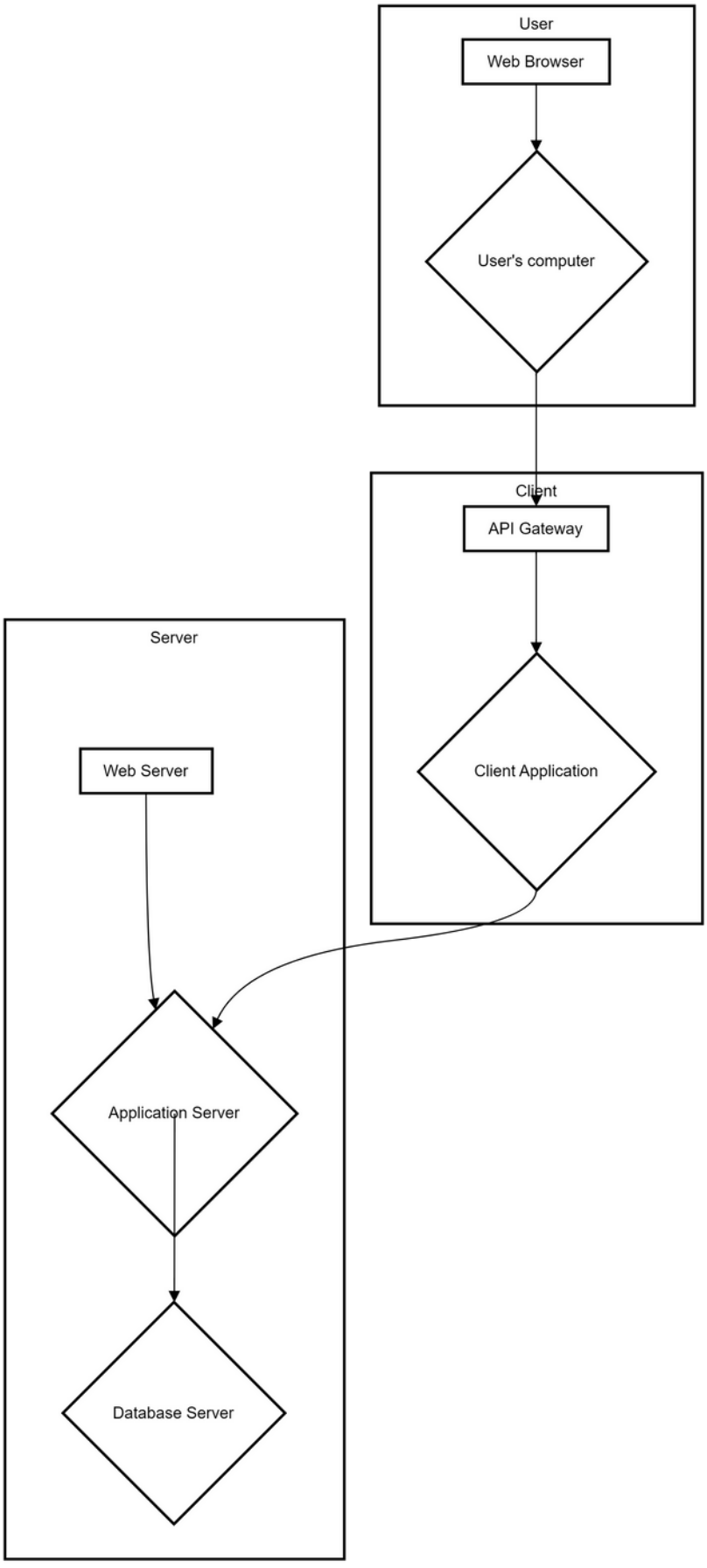


Figure 19:Physical architecture

2.Automation Software architecture

The following figure showcases the software architecture of our automation framework that has been developed using the Cypress testing tool.

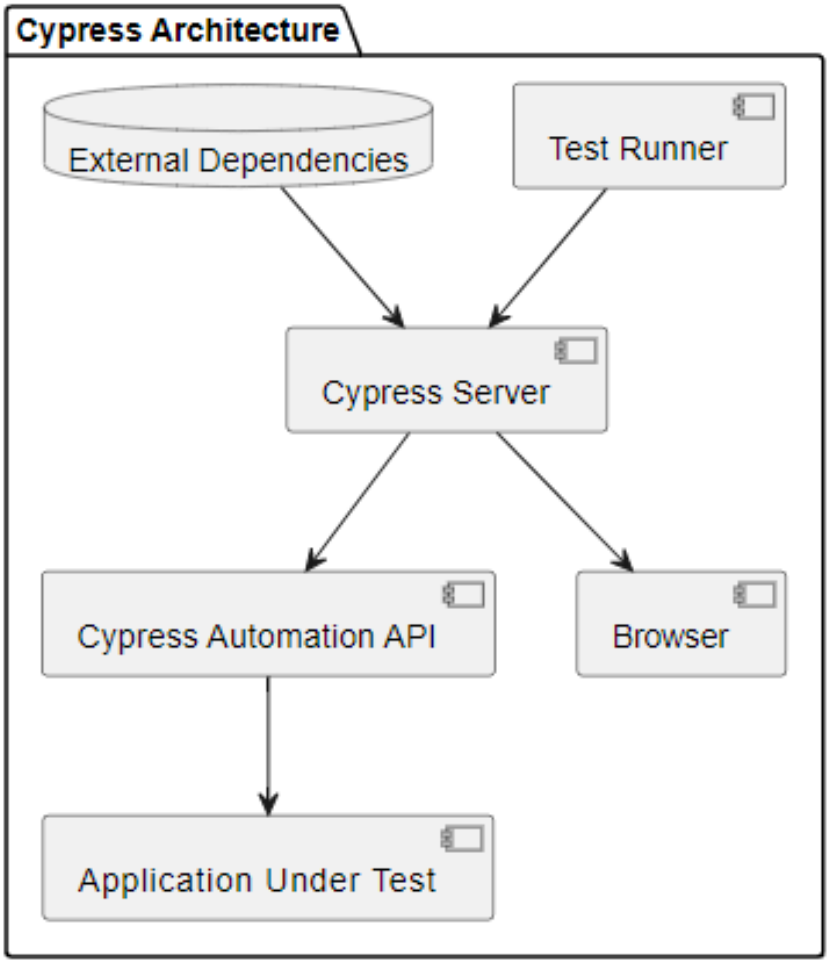


Figure 20:Software architecture

III. Technical environment

This section provides comprehensive information on the technical environment leveraged for the development of the application, represented in a table format that highlights the various technologies utilized throughout the project development stage.

Technology	Description
Postman	An API development and testing tool that enables the creation, administration, and execution of API requests.
Cypress	A JavaScript-based end-to-end testing framework utilized for testing web applications.
GitHub	A web-based Git version control repository hosting service.
Grafana	An open-source platform utilized for data visualization and monitoring.
Google Lighthouse	An open-source tool that audits web page quality, generates reports, and provides suggestions for enhancing page speed and other performance metrics.
JMeter	An open-source performance testing tool used for web applications.

Table 11:Technical environment

Conclusion

In this chapter, we covered the technical aspects of the project, including the overall architecture, both physical and software, as well as the technical environment required for our automation framework. In the upcoming chapter, we will shift our focus to the implementation phase of our solution.

Chapter 5

Implementation of Release Process

Introduction

In this chapter, we will discuss the implementation of a release in our software development project, which includes setting up the development environment, configuring the project, and implementing automated testing and reporting using different tools.

I. Setting up the Development Environment

This section will detail the tools and processes utilized to prepare our development and testing environment to ensure a successful release.

1. Cypress Installation and Configuration

These are the steps to set up the development environment for Cypress:

1.Install Node.js: Cypress is built on Node.js, so you'll need to install it first.
2.Install Cypress: After installing Node.js, you can install Cypress by running the following command in your terminal or command prompt: **npm install cypress --save-dev**
This will install the latest version of Cypress and save it as a development dependency in your project's `package.json` file.
3.Create a Cypress directory: In your project's root directory, create a new directory named cypress. This is where Cypress will store all of its configuration files, test scripts, and other related files.
4. Initialize Cypress: Run the following command in your terminal or command prompt to initialize Cypress: **npx cypress open**
This will create the necessary configuration files and open the Cypress Test Runner.

5.Configure Cypress in the "package.json" file:

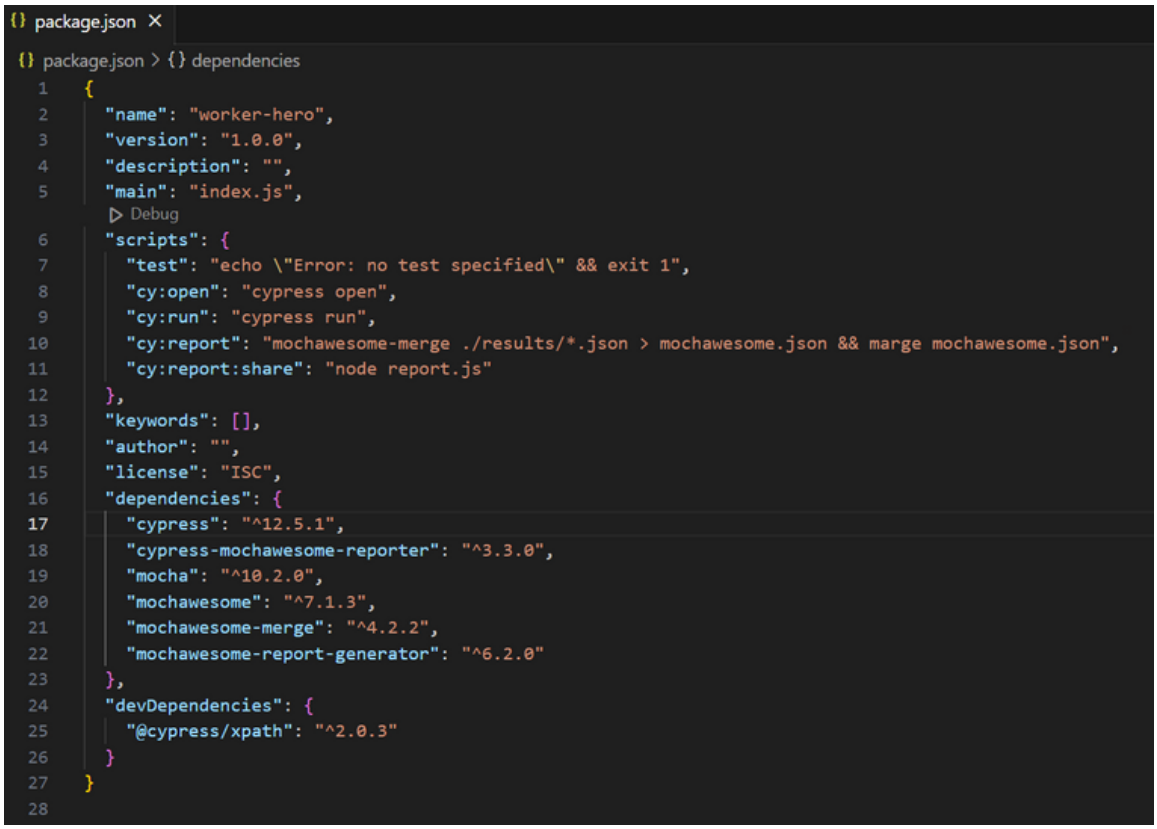


Figure 21 : Dependencies injected in the package.json file

These are the non-default configurations made:

- **"cy:open": "cypress open"**: This script opens the Cypress Test Runner in interactive mode. By default, Cypress Test Runner can be opened by running **cypress open** command in the terminal.
- **"cy:run": "cypress run"**: This script runs Cypress tests in headless mode. By default, running tests in headless mode can be done by running **cypress run** command in the terminal.
- **"cy:report": "mochawesome-merge ./results/*.json > mochawesome.json && marge mochawesome.json"**: This script generates a merged HTML report of test results. The **mochawesome-merge** command merges all the JSON files generated by Cypress tests, and then **marge** command generates an HTML report based on the merged JSON file.
- **"cy:report:share": "node report.js"**: This script runs a Node.js script to upload the generated HTML report to an online platform so it can be shared with others.

- **"cypress-mochawesome-reporter": "^3.3.0"**, **"mochawesome": "^7.1.3"**, **"mochawesome-merge": "^4.2.2"**, **"mochawesome-report-generator": "^6.2.0"**: These are non-default dependencies that you have installed to generate an HTML report of test results using the **mochawesome** reporter.

6.Write test scripts: Now that your development environment is set up and configured, you can start writing test scripts .

2. Advanced Setup with Page Object Model (POM) Pattern

- Dividing the `cypress` e2e folder into `pages` and `tests` is a widely adopted practice in test automation. The `pages` directory usually contains the Page Object Model (POM) files that represent the pages of the application being tested. These files encapsulate the page elements and actions that are available on each page, and can be reused across multiple tests. On the other hand, the `tests` folder contains the actual test scripts that leverage the Page Objects defined in the `pages` directory to interact with the application and verify expected behavior. By separating the two concerns, it becomes easier to maintain and update the test scripts, as any changes to the application's UI can be made in the `pages` files, while the `tests` files can remain focused on the actual testing logic.

- Adopting this approach also promotes a more modular approach to testing, as tests can be built around smaller, reusable components rather than one large, monolithic script. Moreover, it can help to reduce code duplication and make the overall codebase more organized and easier to navigate.

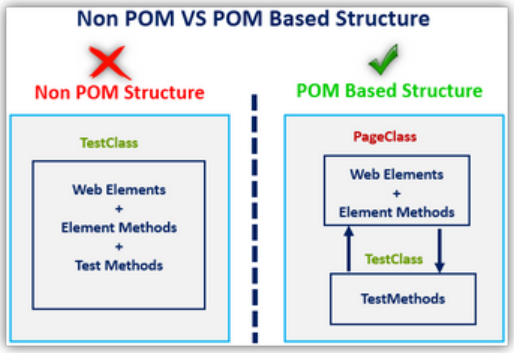


Figure 22 : Comparison of Project Object Model (POM) based structure and Non-POM based structure.

II. Manual Testing

1.Functional Testing

A.TestRail

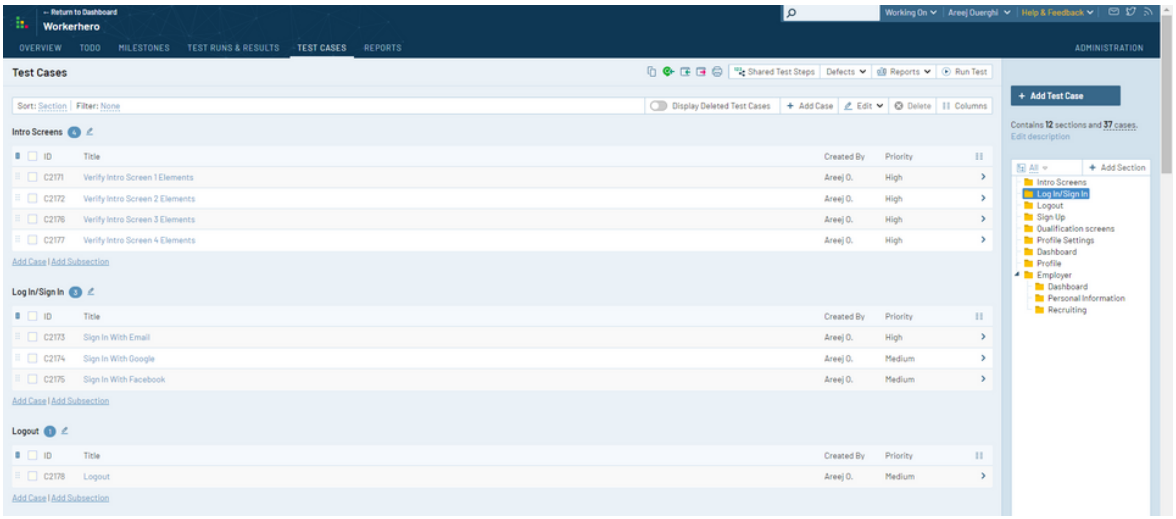


Figure 23 : TestRail Test Cases Interface

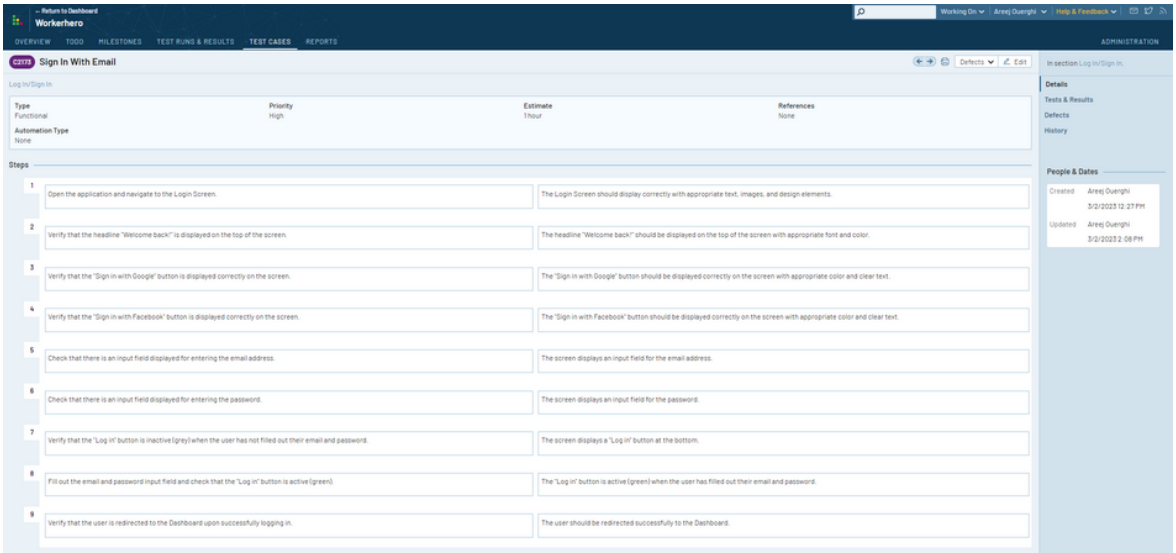


Figure 24 :TestRail Documentation Interface

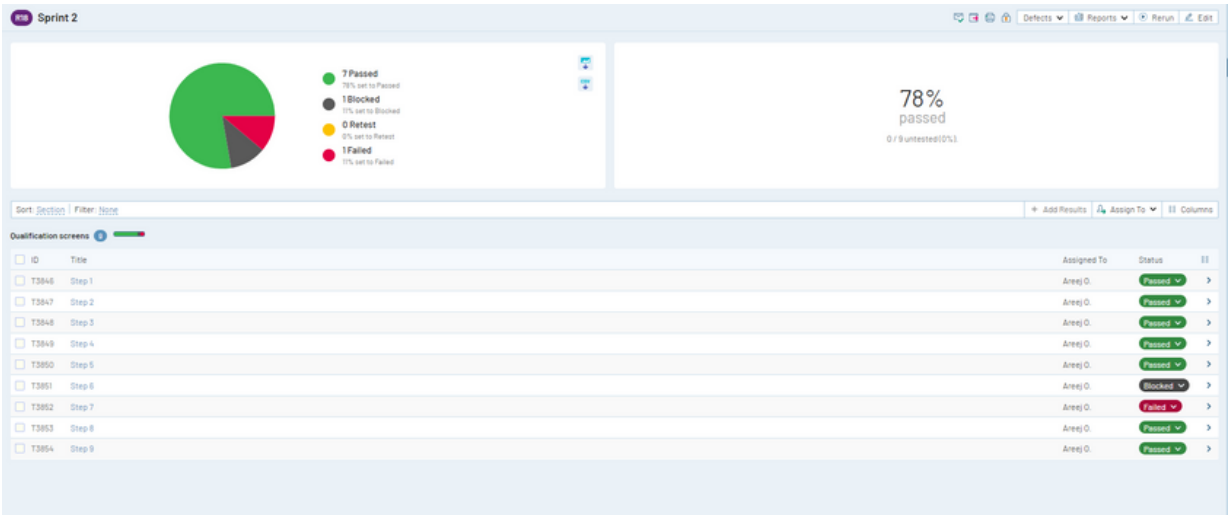


Figure 25:TestRail Run Results Interface

B.Postman

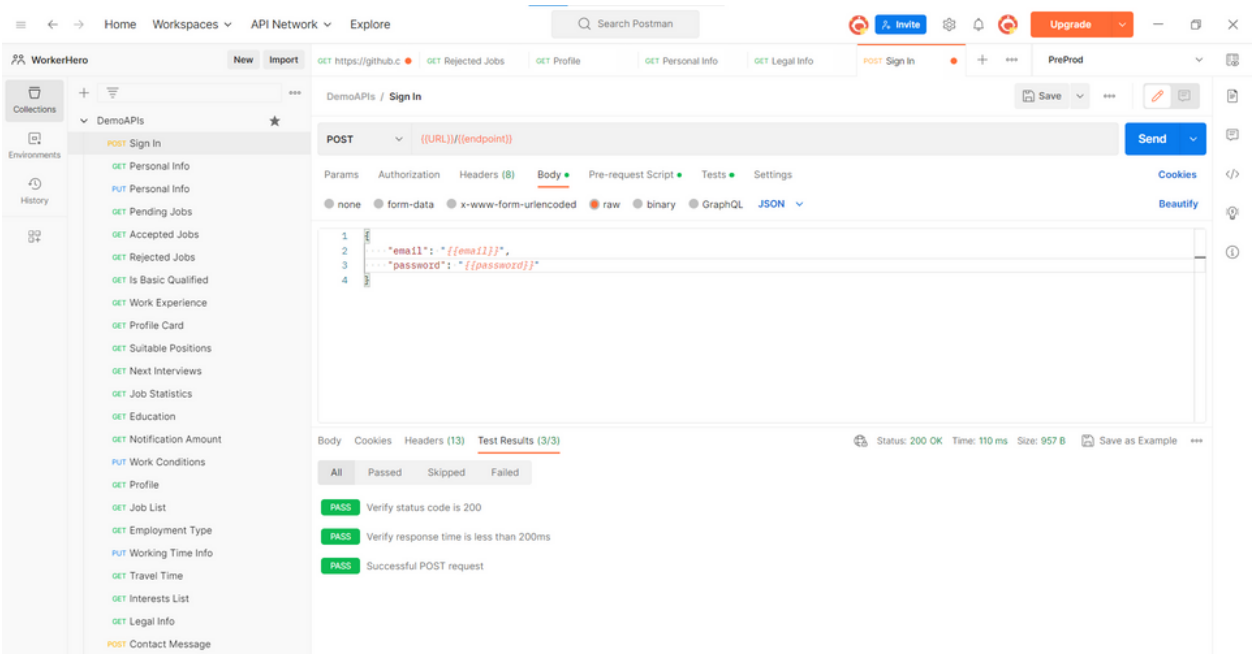


Figure 26: Postman Interface

C.Jira Software

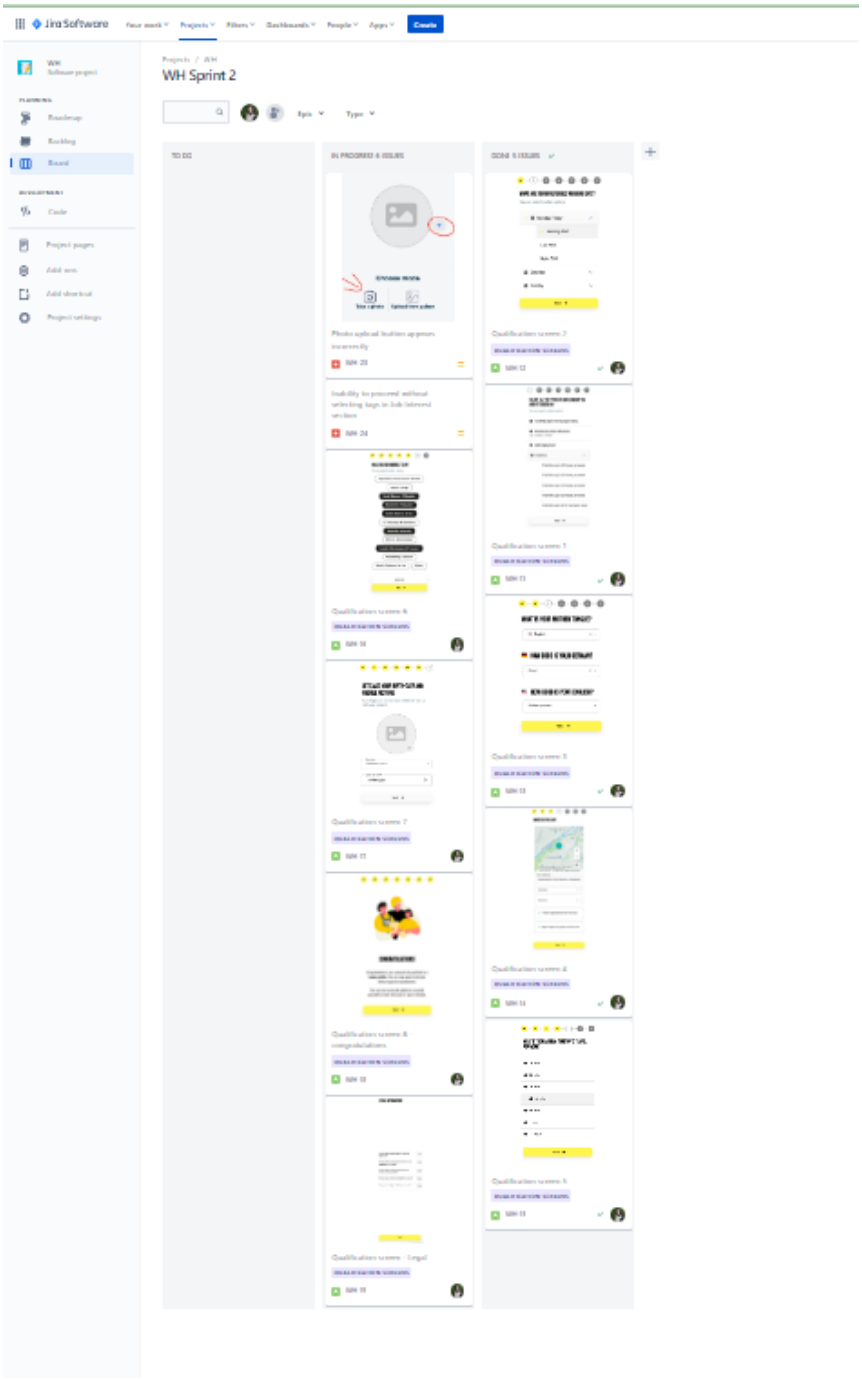


Figure 27: Jira Software Board

2.Non-Functional Testing

A.Google Lighthouse

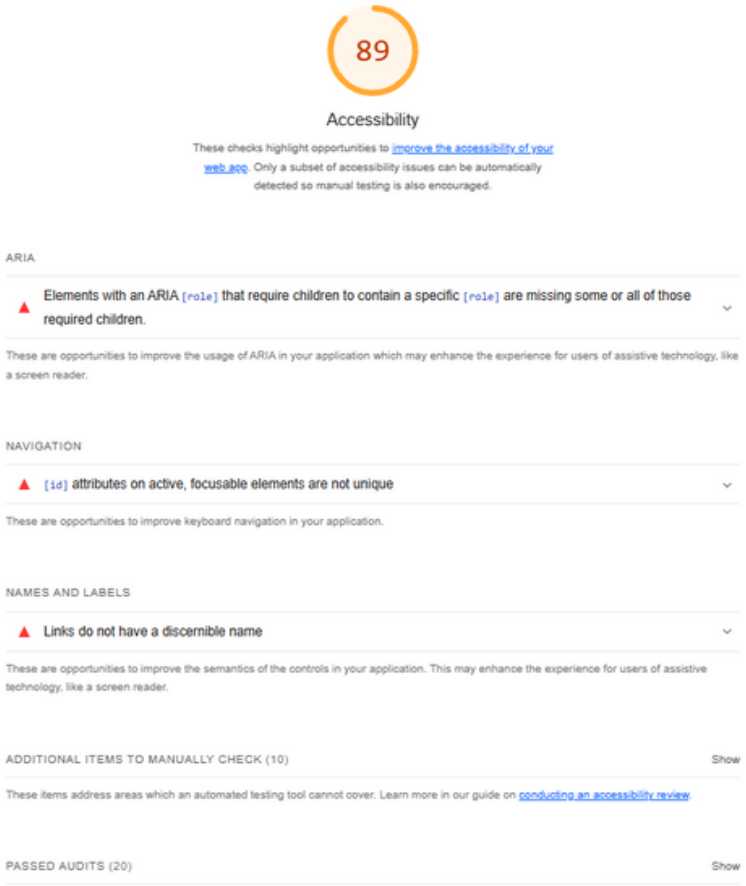
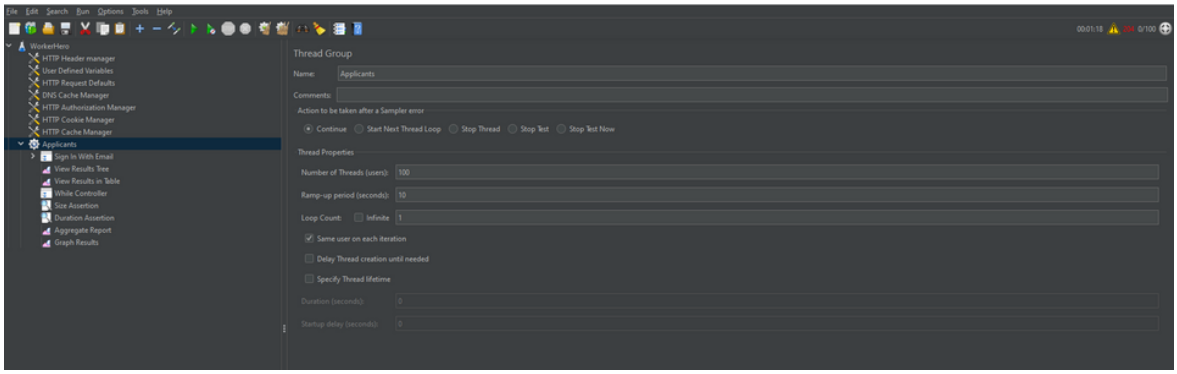


Figure 29: Home Page Accessibility Testing Report

B.JMeter



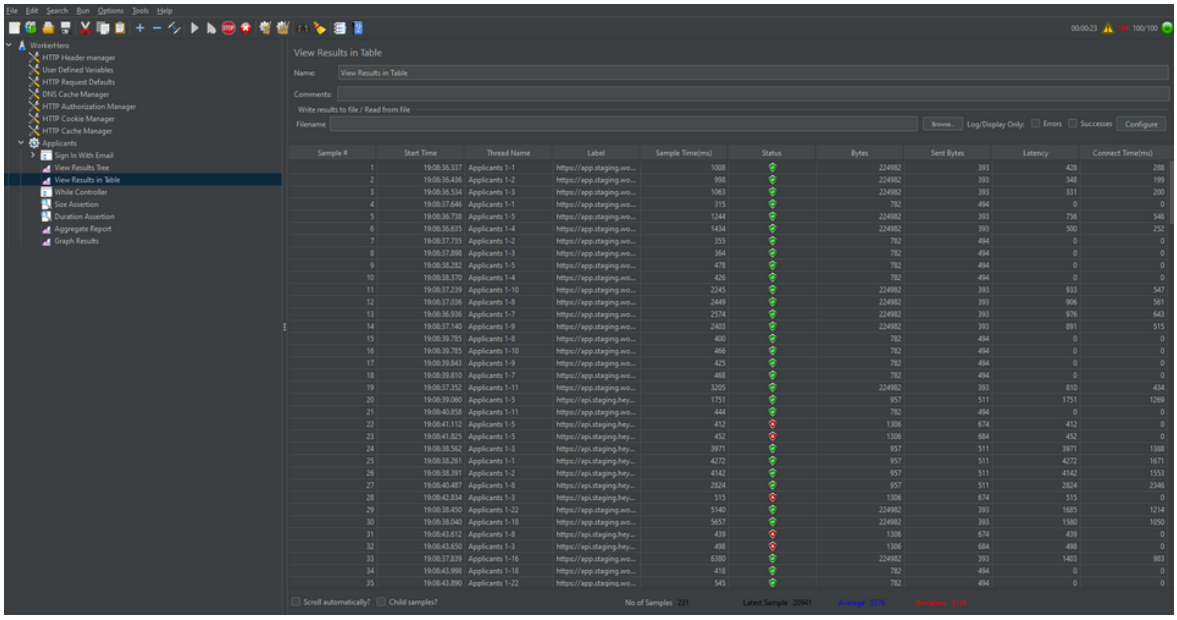


Figure 30 : JMeter Interface

III. Test Automation

1.Project Object Model (POM) Pages

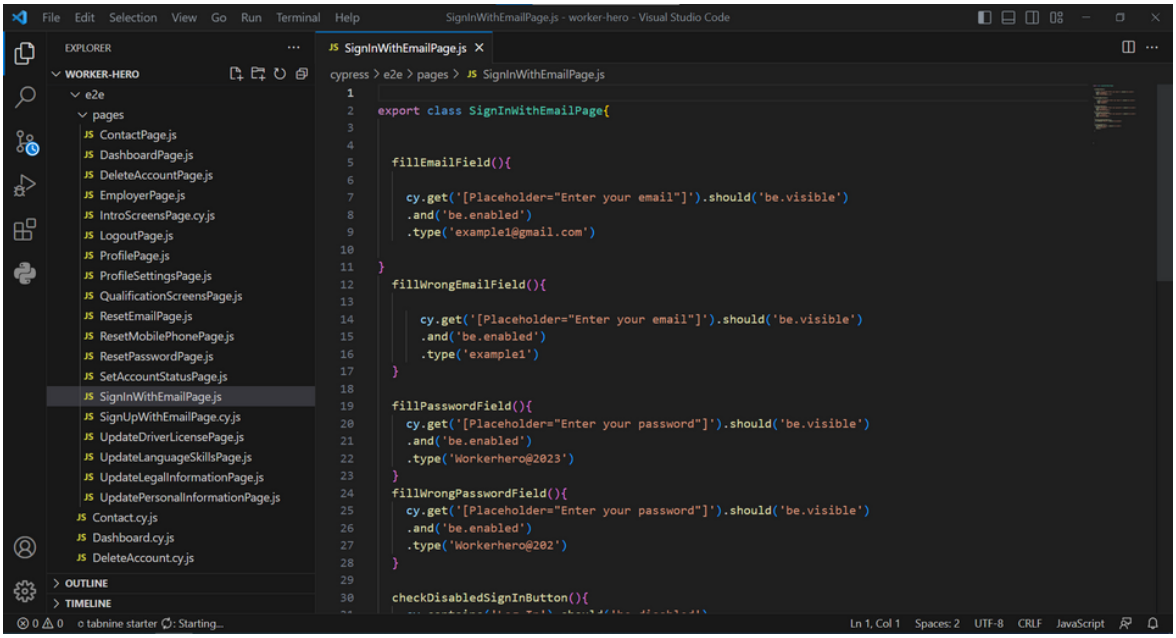


Figure 31 : POM Page Interface

- **Selectors :**
- Selectors in web development are used to identify and locate specific HTML elements on a web page. These elements can include things like buttons, text fields, images, and more. Selectors are typically used to perform operations on these elements, such as clicking a button or entering text into a text field.

- In Cypress, there are several types of selectors available, including CSS Selectors, XPath Selectors, Custom Selectors, DOM Element Selectors, and jQuery Selectors. Each selector type has its own syntax and set of rules for selecting elements.

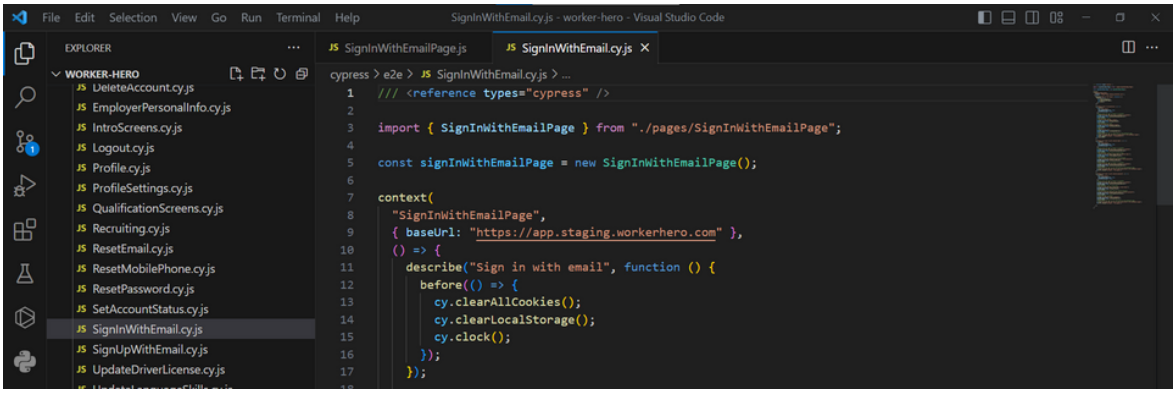
- CSS Selectors are the most commonly used selector type in Cypress. They allow you to select elements based on their class, ID, or attributes. For example, you can use a CSS Selector like `.my-class` to select all elements with a class of 'my-class', or `#my-id` to select an element with an ID of 'my-id'.

- XPath Selectors are another type of selector that can be used in Cypress. They are more powerful than CSS selectors, as they can select elements based on their position, text content, and more.
- Custom Selectors allow you to create your own selectors using JavaScript functions. This can be useful when you need to select elements based on complex logic that cannot be easily expressed using CSS or XPath selectors.
- DOM Element Selectors provide several built-in methods for selecting elements based on their position in the DOM, such as cy.get(':first') or cy.get(':last').
- Finally, jQuery Selectors provide a wide range of options for selecting elements based on their attributes, text content, and more.
- Overall, selectors are a critical component of any Cypress test suite, and can help ensure that your tests are properly targeting the elements you want to interact with.
- **Assertions :**
- Assertions in software testing are used to verify whether a given piece of code or functionality behaves as expected. Cypress provides a rich set of built-in assertions that can be used to make sure that your application behaves correctly.

- Some of the most commonly used Cypress assertions include:

- - **`.should('exist')`** : verifies that an element exists in the DOM.
- - **`.should('be.visible')`** : verifies that an element is visible on the page.
- - **`.should('have.text', 'expected text')`** : verifies that an element has the expected text content.
- - **`.should('have.class', 'expected-class')`** : verifies that an element has the expected class.

2.Test Suite



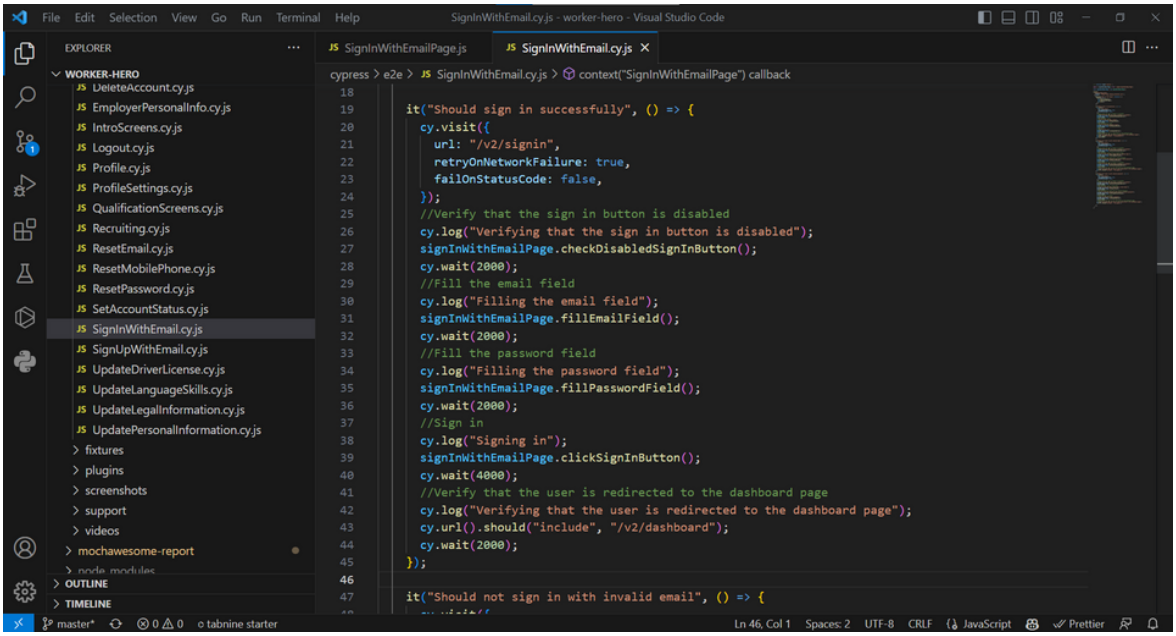


Figure 32 : Test Suite

- This Cypress test suite is testing the sign-in functionality of a web application for various scenarios.
- The "Sign in with email" test case contains three sub-tests, one for successful sign-in and two for unsuccessful sign-in scenarios where the user enters invalid email or password.
- The tests verify whether the sign-in button is enabled/disabled, the email and password fields are filled correctly, the user is redirected to the dashboard page on successful sign-in, and that the user is not redirected to the dashboard page in the case of invalid sign-in attempts.
- The test suite uses page object model design and Cypress assertions to verify the expected behavior of the application.

3.Test Execution

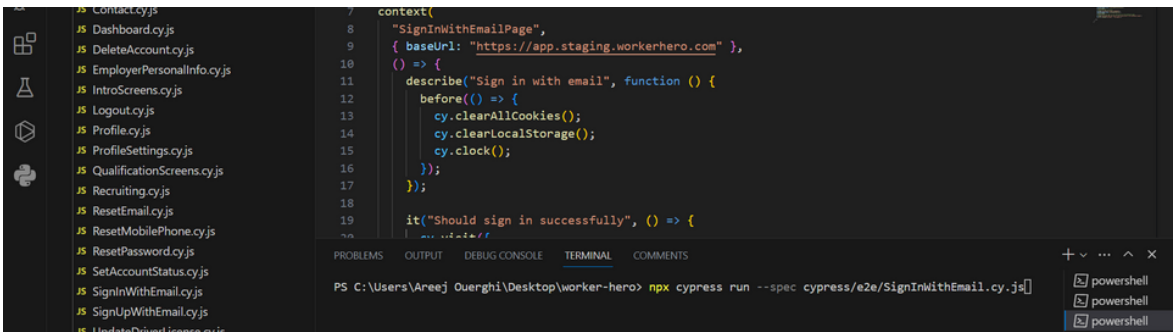


Figure 33 : Test Execution Command

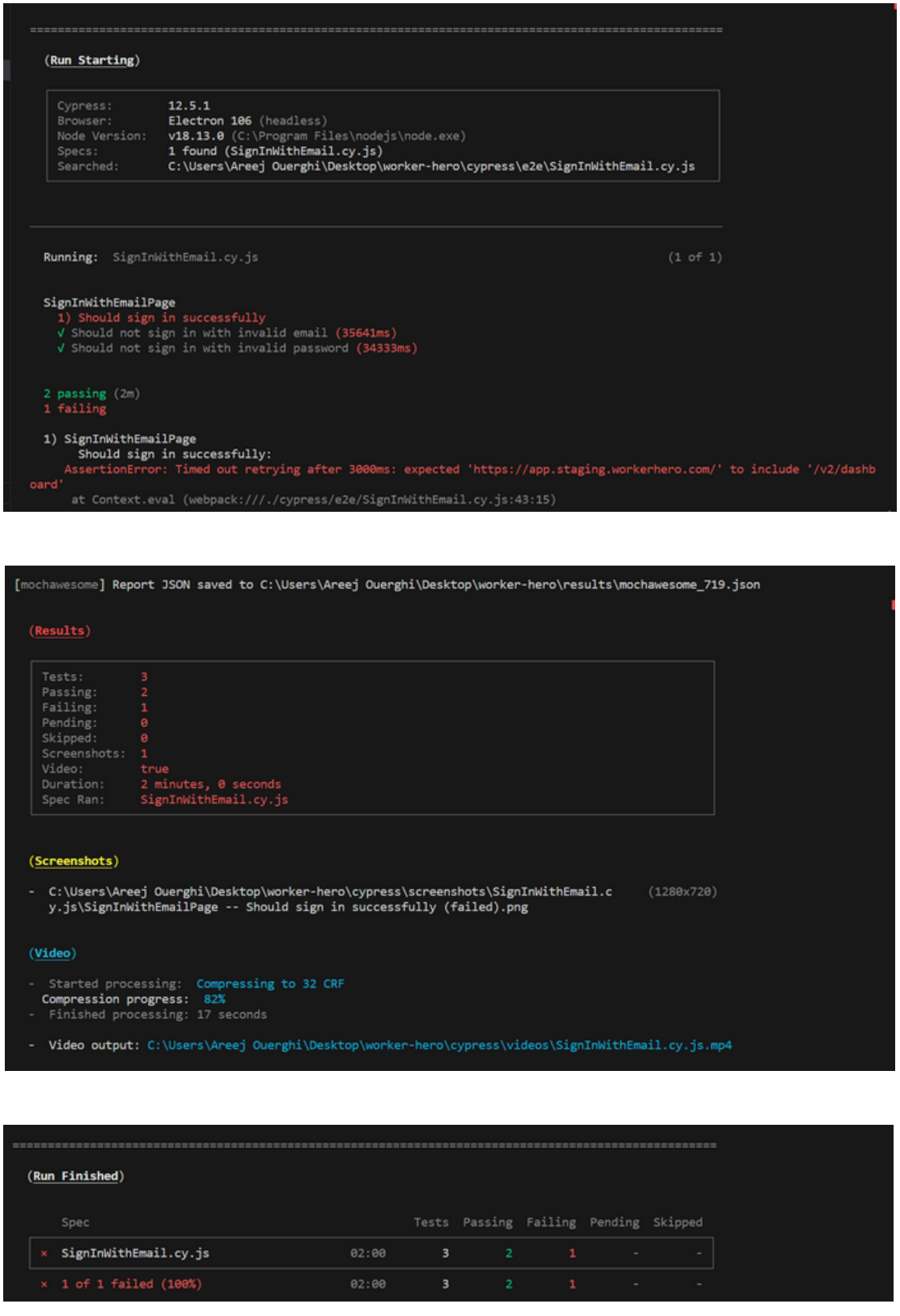


Figure 34 : Test Execution

4.Bugs Tracking

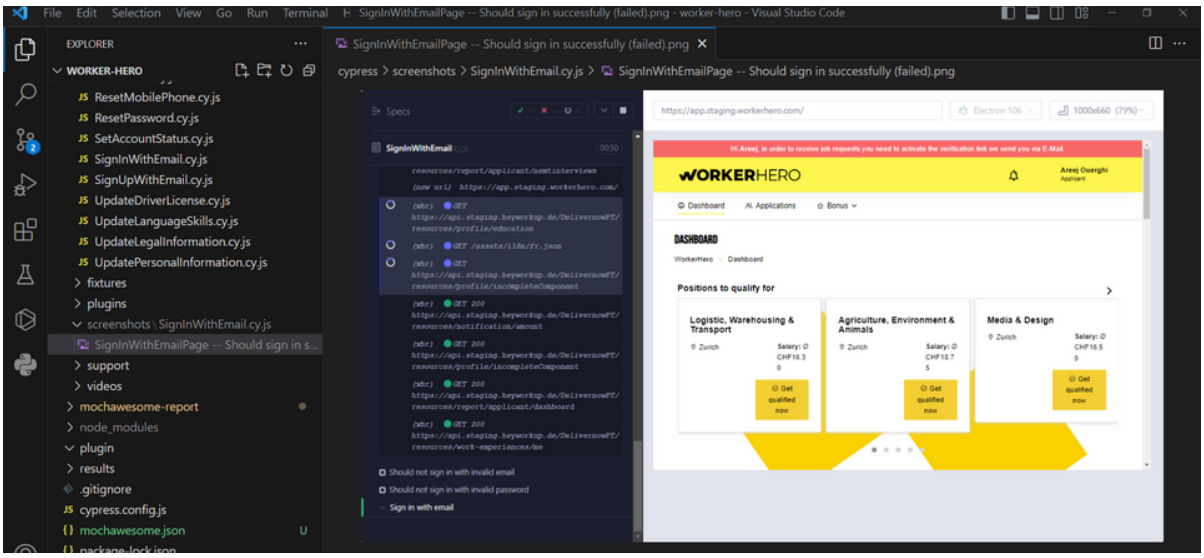


Figure 35 : Bug Screeshot

5.Video Recording

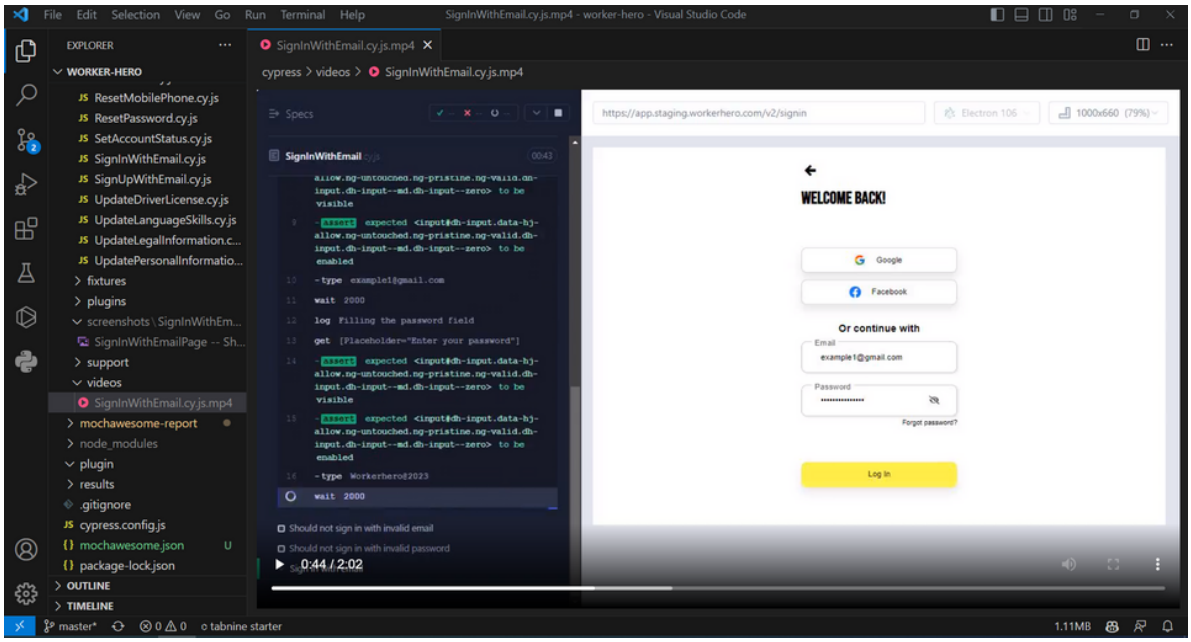


Figure 36 : Video Recording

6.JSON Reporting

Figure 33 : Test Execution

Figure 37 : JSON Report

7.HTML Reporting

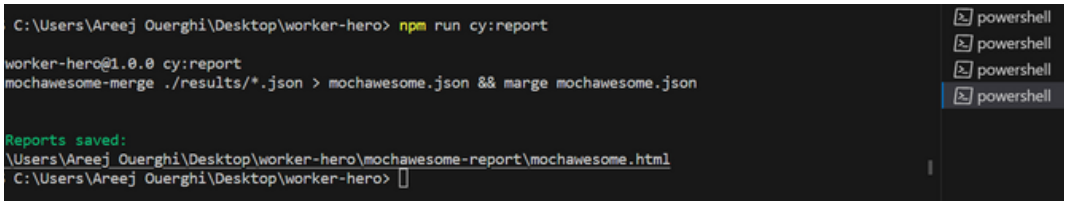


Figure 38 : HTMLReport Generation Command

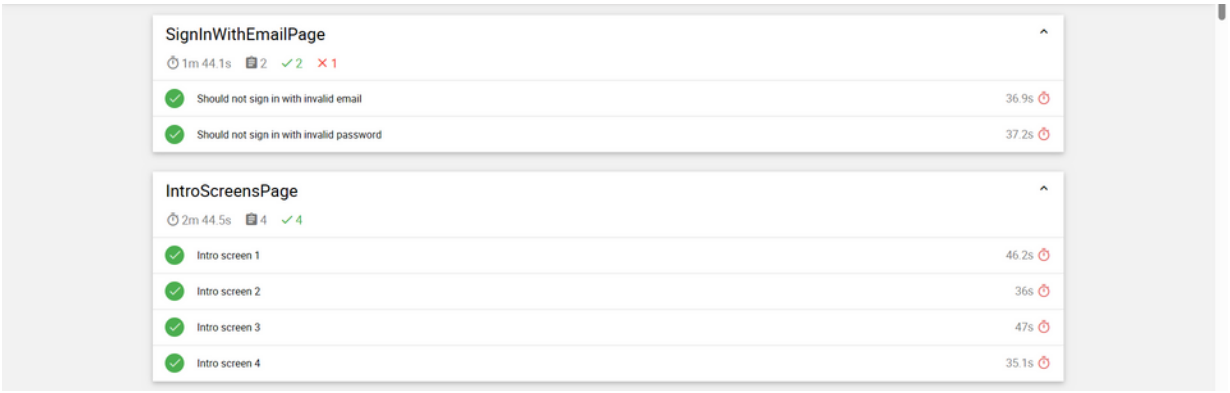


Figure 39 : HTML Report

IV. Continuous Integration and Delivery (CI/CD)

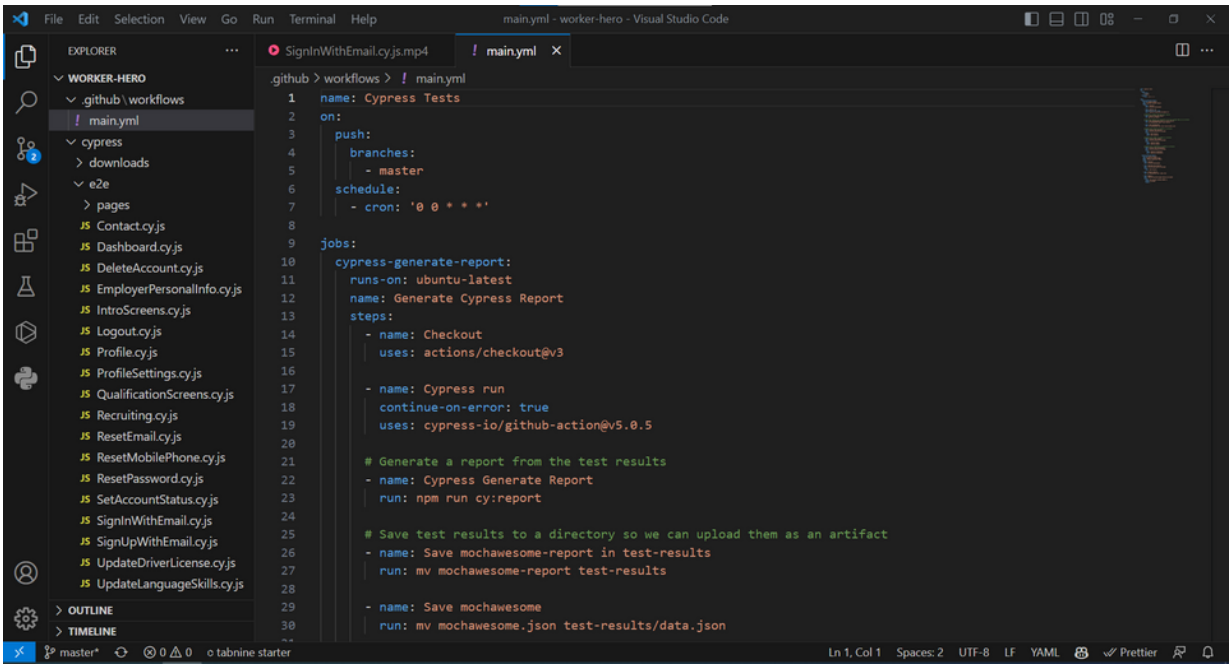


Figure 40 : GitHub Actions workflow in the main.yml file

- This is a GitHub Actions workflow written in YAML format, named "main.yaml" and configured to run on the "master" branch when a push event occurs, as well as once per day on a schedule.
- The workflow has two jobs: "cypress-generate-report" and "cypress-publish-report".

- The first job, "cypress-generate-report", is responsible for running Cypress tests, generating a report, and uploading test results, videos, and screenshots as artifacts. Specifically, this job runs on an Ubuntu machine, and performs the following steps:

- - It checks out the source code using the actions/checkout action.
- - It runs Cypress tests using the cypress-io/github-action action. If there are any test failures, the job continues anyway due to the "continue-on-error" flag.
- - It generates a report from the test results using the "npm run cy:report" command.
- - It saves the report in the "test-results" directory using the "mv" command.
- - It saves the "mochawesome.json" file, which contains the raw test results, to the "test-results/data.json" file path.
- - It uploads the test results, videos, and screenshots as artifacts using the actions/upload-artifact action.
- The second job, "cypress-publish-report", is responsible for deploying the test results report on GitHub Pages. This job depends on the successful completion of the previous job, "cypress-generate-report", to access the test results. Specifically, this job performs the following steps:

- - It checks out the source code using the actions/checkout action.
- - It downloads the test results artifacts using the actions/download-artifact action.
- - It prepares the website by renaming "mochawesome.html" to "index.html".
- - It deploys the website on the "gh-pages" branch using the JamesIves/github-pages-deploy-action action, which uses a personal access token (stored as a secret) to push the website changes to the "gh-pages" branch.

Overall, this GitHub Actions workflow automates the process of running Cypress tests, generating a report, and publishing the report on GitHub Pages.

1.Generate Cypress Report

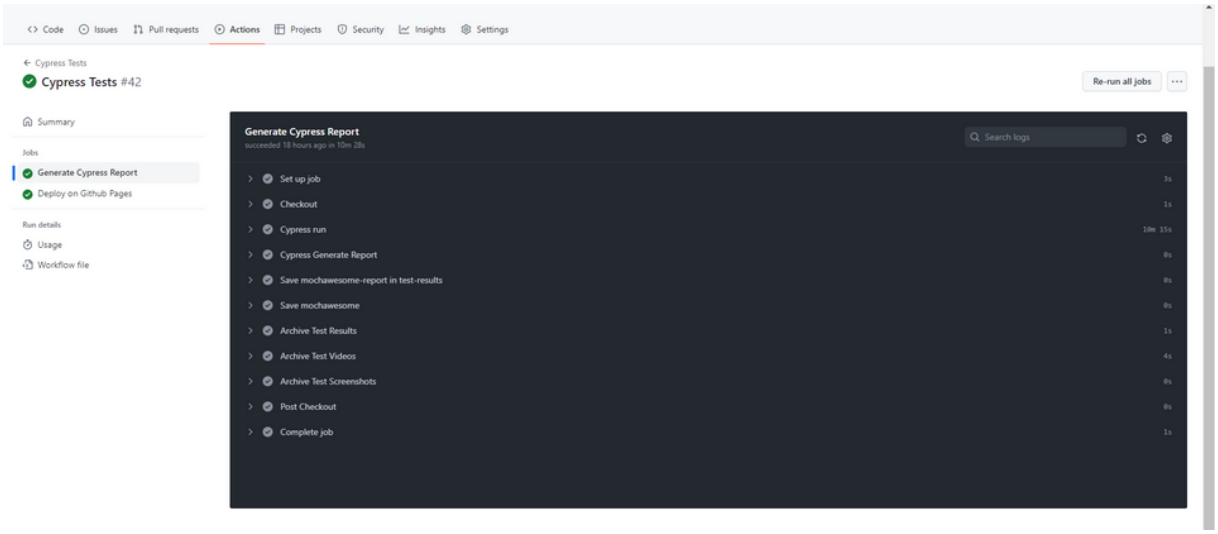


Figure 41 : Generate Cypress Report

2.Deploy on Github Pages

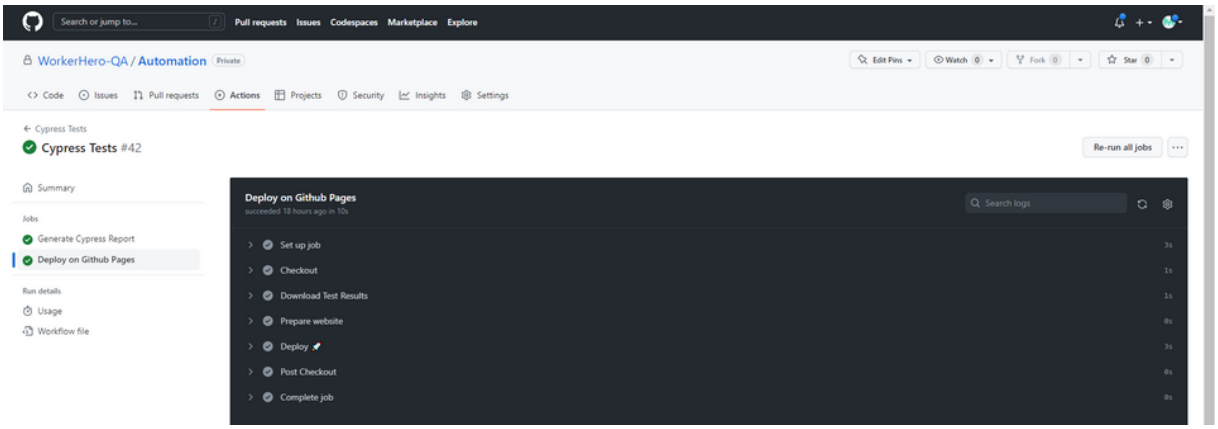


Figure 42 : Deploy on Github Pages

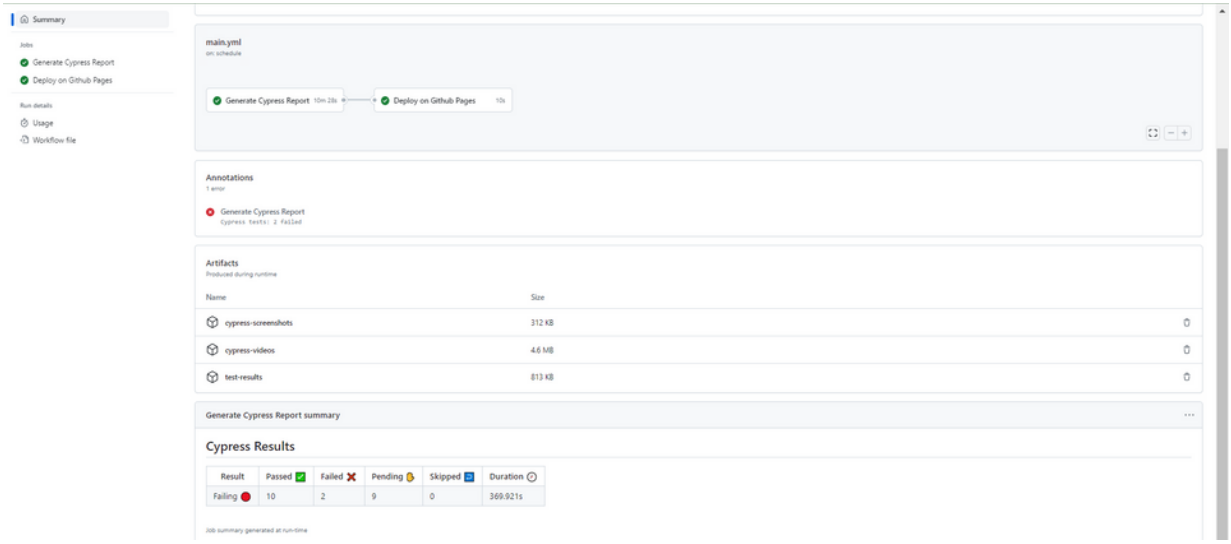


*Figure 43 : G*ithub Summary Page

V. Dashboarding

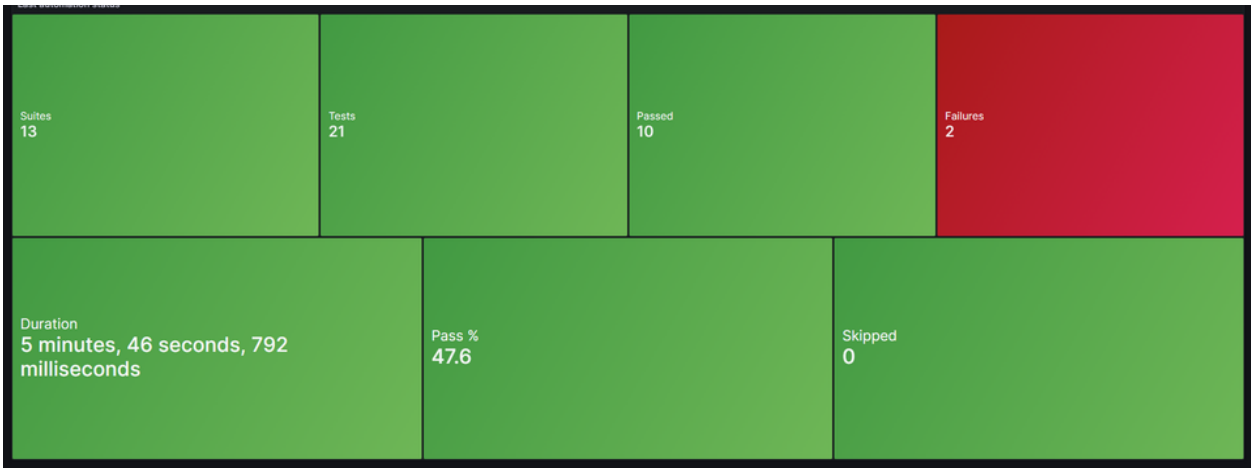


Figure 44 : Test Results Dashboard

VI. Testing and Validation

Software testing is a crucial process that ensures the proper functioning of a system by comparing expected behaviors with actual results. Throughout each sprint, we tested the features of the module and then validated them with the Product Owner. To achieve this, we created a comprehensive table containing all the relevant functional test cases.

Conclusion

Throughout this chapter, we have implemented the release process, which involved conducting various functional tests and automating the different sections as required by our project's needs. In addition to functional testing, we also performed performance and accessibility testing. Moreover, we created dashboards using Grafana to track and report the test results.

General Conclusion

This document presents our end-of-studies project carried out within TCE Entreprise. Our project involved both manual and automated testing to ensure non-regression and guarantee the platform's performance and accessibility. Additionally, we implemented dashboarding to provide an overview of the test results, allowing for efficient monitoring and reporting.

Our solution aims to improve the overall quality of the WorkerHero platform by ensuring the validity of each development stage through a combination of manual and automated non-regression testing. This approach helps identify any anomalies that may arise during the development process and allows for timely corrections, thus reducing the risk of bugs and enhancing the user experience. In addition, the inclusion of performance and accessibility testing in our solution ensures that the platform meets the necessary standards and delivers an optimal experience to all users. Finally, the integration of dashboarding capabilities enables us to provide valuable insights and reporting to the QA team, facilitating planning and execution of test cases. Overall, our solution is designed to enhance the efficiency and effectiveness of the testing process, resulting in a more robust and reliable platform for users.

Looking ahead, we intend to continue our work on the next release by developing additional test scripts and creating more performance scenarios. Additionally, we plan to integrate our automated non-regression testing solution with the JIRA tool to streamline the correction and reporting process.

Bibliography

- [1]:https://www.linkedin.com/company/tcee/ (Consulted on 12/02/2023)
- [2]:https://www.goretro.ai/post/scrum-methodology(Consulted on 03/07/2023) (Consulted on 18/02/2023)
- [3]:https://www.javatpoint.com/unit-testing(Consulted on 25/02/2023)
- [4]:https://www.testingdocs.com/questions/what-is-defect-life-cycle/(Consulted on 03/03/2023)
- [5]:https://invozone.com/blog/software-testing-tips-for-proper-quality-assurance/ (Consulted on 10/03/2023)
- [6]:https://rpi4cluster.com/docker/selenium/(Consulted on 17/03/2023)
- [7]:https://logowik.com/appium-vector-logo-5623.html (Consulted on 24/03/2023)
- [8]:https://www.pngwing.com/en/free-png-kpujt(Consulted on 31/03/2023)
- [9]:https://brandfetch.com/cypress.io(Consulted on 07/04/2023)
- [10]:https://github.com/logos(Consulted on 14/04/2023)

- [11]:https://www.clipartmax.com/middle/m2H7d3Z5Z5b1G6Z5_grafana-prometheus-logo/(Consulted on 21/04/2023)
- [12]:https://icon-icons.com/pt/icone/application-json/92733(Consulted on 28/04/2023)
- [13]:https://en.wikipedia.org/wiki/HTML(Consulted on 05/05/2023)
- [14]:https://brandfetch.com/cypress.io(Consulted on 08/05/2023)
- [15]:https://www.influxdata.com/influxdb-templates/apache-jmeter/(Consulted on 08/05/2023)
- [16]:https://web.dev/lighthouse-whats-new-6.0/(Consulted on 08/05/2023)
- [17]:https://www.svgrepo.com/svg/354202/postman-icon(Consulted on 08/05/2023)
- [18]:https://myskillpoint.com/page-object-model-pom-page-factory-in-selenium-webdriver/ Consulted on 08/05/2023)