Fazal Sandhi

Quick Start to become QA Engineer

Metropolia University of Applied Sciences

Master of Engineering

Information Technology

Master’s Thesis

18 June 2023

Abstract

Author:	Fazal Sandhi
Title:	Quick Start to become a QA Engineer
Number of Pages:	71 pages
Date:	16 May 2023
Degree:	Master of Engineering
Degree Programme:	Information Technology
Professional Major:	Networking and Services
Supervisors:	Ville Jääskeläinen

As someone exploring a career in Quality Assurance (QA), encountering unfamiliar responsibilities was part of the journey. Extensive research was conducted, involving online video consumption and various research methods. The main goal was to create a detailed guide for people who want to become skilled in Quality Assurance.This thesis is a valuable resource for those entering the QA field. It provides essential insights to help someone beginning his journey. It's important to note that this thesis isn't a step-by-step guide; instead, it's meant to be a comprehensive reference book. In its pages, one finds explanations of important ideas and industry standards outlined by ISTQB. Additionally, the thesis covers basic programming languages needed for testing, emphasizes the importance of automation tools, and explains other essential tools commonly used by QA professionals.This investigation revealed the absence of a coherent framework and comprehensive insights within available resources, leading to the creation of this thesis.

Keywords: Quality Assurance, ISTBQ fundamentals, Testing tools, Technology used in QA

Contents

List of Abbreviations

API	Application Programming Interfaces
ATDD	Acceptance Test Driven Development
BDD	Behavioral-Driven Development
CD	Continuous Developmnet
CI	Continuous Integration
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	HyperText Markup Language
ISTBQ	International Software Testing Qualifications Board
OL	Ordered Lists
QA	Quality Assurance
SDLC	Software Development Lifecycle
TDD	Test Driven Development
UL	Unordered Lists

1 Introduction

Quality Assurance (QA), a crucial aspect of ensuring products meet specified quality standards, involves a variety of tasks to identify and address issues. This encompasses products ranging from software to gadgets, making QA a multifaceted discipline. However, QA can be challenging due to the complexity of issue identification and resolution, requiring extensive knowledge and the utilization of diverse tools.

The primary challenge addressed in this thesis revolves around how individuals can be effectively guided to learn about QA. The objective is to furnish essential information that contributes to success in the QA field. This is paramount because proficient QA leads to enhanced product quality. Equipping QA professionals with the necessary skills and knowledge empowers them to identify and rectify issues, resulting in improved products and increased customer satisfaction.

Within the landscape of Quality Assurance, this thesis serves as a guiding beacon. It offers valuable insights into essential skills required for success in QA. The learning journey involves understanding the guidance provided by the International Software Testing Qualifications Board (ISTQB), grasping programming languages, acknowledging the significance of automation tools, and utilizing various other tools integral to daily QA operations.

Starting with an exploration of ISTQB standards and certifications, the foundational elements of HTML, CSS, and JavaScript are elucidated—vital knowledge for QA professionals. The discourse progresses to cover tools like Robot Framework and Jira, which facilitate project and test case management. The mention of Xrays in Jira is made to underscore efficient test case management. Finally, the thesis explores the utility of GitHub in managing automation code and test cases.

It's crucial to note that while this thesis does not provide an exhaustive tutorial, it establishes a robust foundation for launching a successful career in QA.

2 ISTBQ standards

In this exploration of software testing based on the Certified Tester Foundation Level Syllabus v4.0 by the International Software Testing Qualifications Board (ISTQB), we journey through six chapters, covering key concepts, test planning, design techniques, management strategies, defect life cycle, and the role of test tools. The aim is to provide a practical understanding of software testing methodologies for improved software quality and development efficiency.

2.1 Fundamentals of Testing

What is Testing?

Purpose of Testing: Assess software quality, reduce the risk of software failure, discover defects, and evaluate software artifacts.

Misconceptions: Testing involves more than just executing tests; it includes verification and validation aligned with the software development lifecycle.

Static vs. Dynamic Testing: Testing can be static (reviews and static analysis) or dynamic (execution of software).

Test Objectives

Typical Objectives: Evaluate work products, trigger failures, ensure coverage, reduce risk, verify requirements, comply with legal standards, provide information to stakeholders, build confidence, and validate completeness.

Testing and Debugging

Distinction: Testing and debugging are separate activities; testing can trigger failures or directly find defects.

Debugging Process: Reproduction of failure, diagnosis, fixing the cause, confirmation testing, and regression testing.

Why is Testing Necessary?

Quality Control: Testing as a form of quality control contributes to achieving project goals within scope, time, quality, and budget constraints.

Testing's Contributions to Success: Detect defects, evaluate quality, represent users, meet contractual/legal requirements, and contribute to project decisions.

Testing and Quality Assurance (QA)

Difference: Testing is quality control; QA is a process-oriented, preventive approach focusing on processes, applicable to both development and testing.

Errors, Defects, Failures, and Root Causes

Cause and Effect: Errors lead to defects, which may result in failures; root causes are fundamental reasons for problems.

Testing Principles

Seven Principles: Testing shows the presence of defects, exhaustive testing is impossible, early testing saves time and money, defects cluster together, tests wear out, testing is context-dependent, and the absence-of-defects fallacy.

Test Activities, Testware, and Test Roles

Test Process: Defined by various activities like planning, monitoring, analysis, design, implementation, and completion.

Context Dependency: Test activities tailored based on the system, project, and organizational context.

Testware: Output work products include planning documents, test cases, logs, reports, and completion reports.

Essential Skills and Good Practices in Testing

Generic Skills: Testing knowledge, thoroughness, communication, analytical thinking, technical knowledge, and domain knowledge.

Whole Team Approach: Encourages collaboration and knowledge sharing within the team for effective testing.

Independence of Testing: Achieves a certain degree of independence to enhance defect discovery.

2.2 Testing Throughout the Software Development Lifecycle

Testing in the Context of a Software Development Lifecycle (SDLC)

SDLC models provide a high-level representation of the software development process. Examples of SDLC models: **sequential** (e.g., waterfall), **iterative** (e.g., spiral), and **incremental** (e.g., Unified Process). Activities in software development can align with detailed methods and Agile practices. The choice of SDLC impacts scope, timing, documentation, test techniques, automation, and roles of testers.

Impact of SDLC on Testing

Sequential models involve testers in initial phases but limit dynamic testing early on. Iterative models allow testing at all levels in each iteration, demanding fast feedback and extensive regression testing. Agile favours lightweight documentation, extensive test automation, and experience-based testing techniques.

Software Development Lifecycle and Good Testing Practices

Good testing practices, regardless of SDLC, involve corresponding test activities for every development activity. Different test levels have specific objectives, enabling comprehensive testing without redundancy. Test analysis and design begin during the corresponding development phase for early testing.

Testing as a Driver for Software Development

Test Driven Development (TDD), Acceptance Test Driven Development (ATDD), and Behavioral-Driven Development (BDD) are development approaches emphasizing early testing and a shift-left approach. TDD involves coding through test cases; ATDD derives tests from acceptance criteria; BDD expresses behaviour in natural language.

DevOps and Testing

DevOps integrates development and operations to achieve common goals.

Benefits include fast feedback, stable test environments, increased focus on non-functional characteristics, and reduced regression risk. Challenges include defining the DevOps pipeline, introducing/maintaining Continuous Integration (CI) and Continuous Developmnet (CD) tools, and establishing test automation.

Shift-Left Approach

Early testing is known as a shift-left approach, emphasizing testing earlier in the SDLC. Practices for achieving shift-left include reviewing specifications, writing test cases before coding, using CI/CD, completing static analysis, and performing non-functional testing early.

Retrospectives and Process Improvement

Retrospectives are post-project meetings to discuss success, areas for improvement, and incorporation of improvements. Benefits for testing include increased effectiveness/efficiency, improved quality of testware, team bonding/learning, and better cooperation between development and testing.

Test Levels and Test Types

Test Levels

Five test levels: Component testing, Component integration testing, System testing, System integration testing, and Acceptance testing.

Each level has specific attributes like test objectives, test basis, defects, and responsibilities.

Test Types

Four test types: Functional testing, Non-functional testing, Black-box testing, and White-box testing.

Functional testing checks functional completeness, correctness, and appropriateness; Non-functional testing evaluates other attributes like performance, usability, and reliability.

One should also describe here black-box and white-box testing (shorly).

Confirmation Testing and Regression Testing

Confirmation testing ensures successful defect fixes; Regression testing confirms no adverse consequences due to changes. Impact analysis optimizes regression testing extent; automated regression tests are recommended, especially in CI/CD.

Maintenance Testing

Maintenance includes corrective, adaptive, and performance improvement changes. Maintenance testing depends on risk, system size, and change size.
Triggers include modifications, upgrades, migrations, and retirements; Impact analysis guides changes, and testing includes evaluating change success and checking for regressions.

2.3 Static Testing Summary

Static testing is a method of evaluating software work products without executing the software. This process involves the manual examination of code, specifications, system architectures, or other work products. It aims to improve quality, detect defects, and assess characteristics such as readability, completeness, correctness, testability, and consistency. Static testing can be applied for both verification and validation.

Static Testing Basics:

Examinable Work Products: Virtually any work product can be examined using static testing, including requirement specifications, source code, test plans, and more.

Value of Static Testing:

Detects defects early in the software development lifecycle.

Identifies defects not easily found by dynamic testing (e.g., unreachable code, non-executable work product issues). Provides the ability to evaluate and build confidence in work products. Allows for the measurement of quality characteristics that are not dependent on executing code.

Feedback and Review Process:

- **Benefits of Early and Frequent Stakeholder Feedback:**
  Early communication of potential quality problems. Prevention of misunderstandings about requirements. Improvement of development team understanding and focus on valuable features.
- **Review Process Activities:**
  Planning: Defining the scope, purpose, participants, and criteria for the review.
  Review Initiation: Ensuring preparedness and access to the work product. Individual Review: Reviewers assess the quality of the work product and identify anomalies using various techniques.
  Communication and Analysis: Discussion and analysis of identified anomalies, decisions on status, ownership, and required actions.
  Fixing and Reporting: Creation of defect reports, corrective actions, and reporting of review results.
- **Roles and Responsibilities in Reviews:**
  Manager, Author, Moderator, Scribe, Reviewer, Review Leader play various roles. More detailed roles are possible, as described in the ISO/IEC 20246 standard.
- **Review Types:**
  Various review types exist, ranging from informal reviews to formal inspections. Selection depends on factors like SDLC, development process maturity, work product criticality, and legal/regulatory requirements. Common types include Informal Review, Walkthrough, Technical Review, and Inspection.
- **Success Factors for Reviews:**
  Defining clear objectives and measurable exit criteria. Choosing the appropriate review type for given objectives and context. Conducting

reviews on small chunks to maintain concentration. Providing feedback to stakeholders and authors for improvement. Ensuring adequate time for participants and support from management. Integrating reviews into the organization's culture for learning and process improvement. Providing adequate training for all participants.

In summary, static testing plays a crucial role in defect detection, quality improvement, and building confidence in software work products throughout the software development lifecycle. It complements dynamic testing and involves various stakeholders in structured review processes. Frequent feedback and clear communication are key success factors in static testing.

2.4 Test Analysis and Design

This section provides an overview of Test Analysis and Design, focusing on different test techniques. Here's a summary of the key points:

2.4.1 Test Techniques Overview

Test techniques support test analysis and design. Classified as black-box, white-box, and experience-based. Black-box techniques are independent of internal structure; white-box techniques depend on it. Experience-based techniques rely on tester knowledge and complement other techniques.

2.4.2 Black-Box Test Techniques

- **Equivalence Partitioning**
  Divides data into equivalence partitions based on expected behaviour. Each partition should be tested at least once. Coverage measured as the percentage of partitions covered.
- **Boundary Value Analysis**

Focuses on exercising partition boundaries. 2-value and 3-value BVA differ in coverage items per boundary. Coverage measured as the percentage of boundary values covered.

- **Decision Table Testing**
  Used for testing system requirements with different conditions and outcomes. Coverage items are columns with feasible combinations of conditions. Coverage measured as the percentage of exercised columns.
- **State Transition Testing**
  Models system behaviour through states and transitions. Coverage criteria include all states, valid transitions, and all transitions. All states coverage is less stringent than valid transitions coverage.

2.4.3 White-Box Test Techniques

- **Statement Testing and Coverage**
  Focuses on executable statements. Coverage measured as the percentage of statements exercised. 100% coverage ensures all statements executed but may not detect all defects.
- **Branch Testing and Coverage**
  Focuses on branches in control flow graph. Coverage measured as the percentage of branches exercised. 100% branch coverage includes 100% statement coverage.
- **Value of White-box Testing**
  Strengths include considering entire software implementation. Weaknesses include potential omission of defects if requirements are unclear.

2.4.4 Experience-Based Test Techniques

- **Error Guessing**
  Anticipates errors based on tester knowledge and application history. Fault attacks involve creating a list of possible errors and designing tests.
- **Exploratory Testing**

Simultaneously designs, executes, and evaluates tests. Effective when specifications are inadequate or under time pressure.

- **Checklist-Based Testing**
  Designs, implements, and executes tests based on checklists. Checklists should be regularly updated based on defect analysis.

2.4.5 Collaboration-Based Test Approaches

- **Collaborative User Story Writing**
  Collaborative creation of user stories using techniques like brainstorming. Emphasizes the "3 C’s": Card, Conversation, Confirmation.
- **Acceptance Criteria**
  Conditions for user story acceptance by stakeholders. Used to define scope, reach consensus, describe scenarios, and guide testing.
- **Acceptance Test-Driven Development (ATDD)**
  Test-first approach where test cases are created before implementing a user story. Test cases based on acceptance criteria and examples. Positive, negative, and non-functional testing included.

2.5 Managing the Test Activities

2.5.1 Test Planning

Purpose and Content of a Test Plan
A test plan outlines objectives, resources, and processes for a test project. It serves as a means of communication and adherence to test policies and strategies. The typical content includes context, assumptions, stakeholders, communication, risk register, test approach, budget, and schedule.

Tester's Contribution to Iteration and Release Planning

In iterative SDLCs, release planning and iteration planning involve testers in defining user stories, risk analyses, estimating test effort, and planning testing for both release and iteration.

Entry Criteria and Exit Criteria

Entry criteria define preconditions for activities, while exit criteria define completion conditions. These criteria differ based on test objectives. They include resource availability, testware availability, and initial quality level of the test object.

Estimation Techniques

Test effort estimation involves techniques like estimation based on ratios, extrapolation, wideband Delphi, and three-point estimation. Estimations are based on assumptions and are subject to errors.

Test Case Prioritization

Test cases are prioritized based on factors like risk, coverage, and requirements. Prioritization helps in defining the order of test execution, considering dependencies and resource availability.

Test Pyramid

The test pyramid model illustrates different test granularities, supporting test automation and effort allocation. It categorizes tests into layers, emphasizing varying granularity levels and execution times.

Testing Quadrants

Testing quadrants categorize test levels and types in Agile development. Quadrants distinguish business-facing or technology-facing tests supporting the team or critiquing the product.

2.5.2 Risk Management

Risk management addresses uncertainties to increase the likelihood of achieving objectives. It involves risk analysis and risk control, with a test approach called risk-based testing.

Risk Definition and Risk Attributes

Risk is characterized by likelihood and impact. Risk likelihood is the probability of occurrence, and risk impact is the consequences of the occurrence.

Project Risks and Product Risks

Project risks relate to project management, while product risks concern product quality characteristics. Both can impact project objectives.

Product Risk Analysis

Product risk analysis identifies and assesses risks to focus testing efforts, influencing the scope, test levels, techniques, effort estimation, and risk reduction activities.

Product Risk Control

Product risk control involves mitigation and monitoring. Actions like selecting testers with suitable skills, conducting reviews, applying appropriate test techniques, and dynamic testing help mitigate risks.

2.5.3 Test Monitoring, Test Control, and Test Completion

Test monitoring gathers information to assess progress against exit criteria. Test control uses this information for effective testing. Test completion collects data at project milestones.

Metrics used in Testing

Metrics, like project progress, test progress, product quality, defect, risk, coverage, and cost metrics, help monitor and control testing.

Purpose, Content, and Audience for Test Reports

Test reports communicate test information during and after testing, supporting control and completion. Test progress reports and test completion reports cater to different audiences and frequencies.

Communicating the Status of Testing

Various means, such as verbal communication, dashboards, electronic channels, online documentation, and formal reports, are used to communicate test status based on stakeholders and context.

2.5.4 Configuration Management

Configuration management identifies, controls, and tracks test work products. It ensures traceability, version control, and proper support for testing in DevOps pipelines.

2.5.5 Defect Management

Defect management is crucial for handling reported anomalies. The process involves defect identification, analysis, classification, response, and closure. Defect reports provide information for issue resolution, tracking, and process improvement.

2.6 Test Tools

2.6.1 Tool Support for Testing:

Test tools play a crucial role in supporting various testing activities. These tools can fall into different categories, such as management tools, static testing tools, test design and implementation tools, test execution and coverage tools, non-functional testing tools, DevOps tools, collaboration tools, and others. Their purpose is to enhance efficiency and effectiveness in managing the software development life cycle, requirements, tests, defects, and configuration.

2.6.2 Benefits and Risks of Test Automation:

Potential Benefits:

Time Savings: Automation reduces repetitive manual tasks, such as regression tests and data entry.

Error Prevention: Automation ensures consistency and repeatability, preventing simple human errors.

Objective Assessment: Automation provides objective measures like coverage that can be complex for humans to derive.

Access to Information: Easier access to testing information for better test management and reporting.

Reduced Execution Time: Faster execution enables earlier defect detection, quicker feedback, and a faster time to market.

Enhanced Test Design: Automation frees up time for testers to design more profound and effective tests.

Potential Risks:

Unrealistic Expectations: Misjudging the benefits and ease of use of a tool.

Inaccurate Estimations: Underestimating time, costs, and effort required for tool introduction, script maintenance, and process changes.

Inappropriate Tool Use: Using a tool when manual testing is more suitable for the context.

Overreliance on Tools: Ignoring the need for human critical thinking by relying too much on automation.

Dependency on Tool Vendors: Risks associated with tool vendors, such as business closure, tool retirement, or poor support.

Open-Source Software Risks: Risks with open-source tools, like abandonment or frequent updates.

Compatibility Issues: Tools may not be compatible with the development platform.

Regulatory Compliance: Choosing a tool that does not comply with regulatory requirements or safety standards.

3 Essential Web Technologies for Quality Assurance

Quality Assurance professionals play a crucial role in ensuring the reliability and functionality of web applications. A fundamental understanding of web technologies can significantly enhance their ability to conduct effective testing. This section covers three essential web technologies that every QA professional should be familiar with.

3.1 HTML

HTML (Hypertext Markup Language) serves as the backbone for structuring content on the web. QA professionals, engaged in testing web applications, can significantly benefit from a solid understanding of HTML. This comprehensive guide aims to equip QA professionals with fundamental HTML knowledge essential for effective testing.

3.1.1 Document Structure

HTML documents adhere to a standardized structure that includes:

```html
<!DOCTYPE html>
<html>
<head>
    <title>Page Title</title>
</head>
<body>

<!-- Content goes here -->

</body>
</html>
```

Listing 1. Standard HTML document structure.

In Listing 1, the standard HTML document structure is presented, adhering to a predefined format. The document begins with the declaration **<!DOCTYPE html>**, which specifies the HTML version being used. Following this, the **<html>**

element serves as the root container encompassing the entire HTML document. Within the **<html>** element, the **<head>** section is designated for meta-information about the document, such as the title, which is encapsulated within the **<title>** element. Finally, the **<body>** element contains the primary content of the page. This structure provides a foundational framework for organizing and presenting content within HTML documents.

3.1.2 Basic Elements

The basic elements of an HTML document include headings, paragraphs, lists, links, images, and more. These elements are essential for structuring content and providing a cohesive layout on a webpage.

Headings

HTML provides six levels of headings, ranging from **<h1>** to **<h6>**. Headings are used to define the hierarchical structure of a document, with **<h1>** representing the most important heading and **<h6>** the least important. For example, **<h1>** is typically used for the main title of a page, while **<h2>** may be used for section headings, and so on. By using headings, one can organize content and make it easier for users to navigate and understand the structure of the webpage.

```html
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<!-- ... -->
<h6>Heading 6</h6>
```

Listing 2. HTML headings.

Paragraph

Paragraphs in HTML are defined using the **<p>** element. They are used to group together blocks of text and provide structure to the content. Paragraphs are commonly used for body text, article content, and other textual information on a webpage.

```html
<p>This is a paragraph.</p>
```

Listing 3. Paragraphs in HTML

Links

Links, also known as hyperlinks, are created using the <a> element in HTML. They allow users to navigate between different web pages or sections within the same page. Links are an essential part of web navigation and are used to connect related content together.

```html
<a href="https://www.example.com">Visit Example</a>
```

Listing 4. Hyperlinks

Lists

HTML supports two types of lists: ordered lists (<ol>) and unordered lists (<ul>). Ordered lists are numbered, while unordered lists are bulleted. Lists are used to organize items in a structured format, making it easier for users to scan and understand the information presented.

Unordered List

```html
<ul>
    <li>Item 1</li>
    <li>Item 2</li>
</ul>
```

Listing 5. Unordered List

Ordered List

```html
<ol>
    <li>First</li>
    <li>Second</li>
</ol>
```

Listing 6. Ordered List

3.1.3 Forms

Forms are created using the <form> element in HTML. They allow users to input data and submit it to a server for processing. Forms are commonly used for user registration, login, contact forms, and more.

```html
<form action="/submit" method="post">
    <label for="username">Username:</label>
    <input type="text" id="username" name="username" required>

    <label for="password">Password:</label>
    <input type="password" id="password" name="password" required>

    <input type="submit" value="Submit">
</form>
```

Listing 7. Forms in HTML

3.1.4 Attributes

ID and Class

Attributes provide additional information about HTML elements and are used to modify their behavior or appearance. Two commonly used attributes are id and class. The id attribute uniquely identifies an element on a page, while the class attribute specifies one or more class names to apply styling or scripting purposes

```html
<div id="container" class="important">
    <!-- Content -->
</div>
```

Listing 8. ID and Class

Source and Alt (Image)

Images are inserted into an HTML document using the <img> element. They are used to enhance the visual appeal of a webpage and convey information through graphics and illustrations. Images can help break up large blocks of text and make the content more engaging for users.

```
<img src="image.jpg" alt="Description">
```

Listing 9. Images

3.1.5 QA-Specific Considerations

Inspecting Elements

QA professionals benefit from using browser developer tools (F12 or right-click and "Inspect") to scrutinize and comprehend the structure of web elements.

Automated Testing

In automated testing, understanding locators (XPath, CSS selectors) is crucial for identifying HTML elements, especially when using frameworks like Selenium.

A proficient grasp of HTML empowers QA professionals in testing web applications comprehensively. While this guide covers fundamental HTML elements, continuous learning and hands-on experience remain integral for maximizing testing effectiveness.

3.2 CSS

CSS, or Cascading Style Sheets, is a stylesheet language used to define the presentation of HTML (Hypertext Markup Language) elements on web pages. It enables one to control the layout, formatting, and visual appearance of web content, making it an essential tool for creating aesthetically pleasing and user-friendly websites.

Key Concepts

Selectors in CSS are patterns used to target specific HTML elements on a web page for styling purposes. These patterns can be based on various criteria such as element names, IDs, classes, attributes, or their hierarchical relationships within the HTML document.

```css
/* Selects all <p> elements and sets their text color to blue */
p {
    color: blue;
}
```

Listing 10. Selectors

By using the selector p, all **<p>** elements in the HTML document are targeted. The corresponding CSS declaration **{ color: blue; }** sets the text color of these **<p>** elements to blue.

Listing 10 illustrates how CSS selectors can be employed to apply styling rules to specific HTML elements, enhancing the visual presentation of web content.

CSS properties specify the visual style of selected elements, while values determine how those properties should be applied. Properties include attributes like **`color`, `font-size`, `margin`, `padding`, `background-color`, `border`**, etc.

```css
/* Sets the font size of all paragraphs to 16 pixels */
p {
    font-size: 16px;
}
```

Listing 11. Properties and Values in CSS

When applied to the selector **p**, the CSS declaration **{ font-size: 16px; }** sets the font size of all paragraphs (**<p>** elements) to **16 pixels**.

Listing 11 showcases how CSS properties and values work together to specify the appearance of HTML elements, allowing for consistent and visually appealing web design.

Selectors and declarations are fundamental components of CSS rules. Selectors determine which HTML elements will be styled, while declarations specify the styles to be applied to those elements.

```
/* Selects all elements with class "highlight" and sets their background
color to yellow */
.highlight {
    background-color: yellow;
}
```

Listing 12. Selectors and declarations

For example, in the CSS rule **.highlight { background-color: yellow; }**, the selector **.highlight** targets all elements with the class "highlight". The declaration **background-color: yellow;** specifies that the background color of these elements should be yellow.

Element Selector targets all instances of a specific HTML element. For Example in Listing 13, the selector h1 targets all <h1> elements in the HTML document. The declaration color: red; specifies that the text color of these elements should be red.

```
/* Selects all <h1> elements and sets their font color to red */
h1 {
    color: red;
}
```

Listing 13. Element Selectors

Class Selector targets elements with a specific class attribute. For instance, in the Listing 14 CSS rule, the selector **.button** targets all elements with the class "button". The declaration **background-color: green;** specifies that the background color of these elements should be green.

```
/* Selects all elements with class "button" and sets their background
color to green */
.button {
    background-color: green;
}
```

Listing 14. Class Selectors

ID Selector targets a single element with a specific ID attribute. Consider the Listing 15 CSS rule, the selector **#header** targets the element with the ID "header". The declaration **font-size: 24px;** specifies that the font size of this element should be **24 pixels**.

```
/* Selects the element with ID "header" and sets its font size to 24
pixels */
#header {
    font-size: 24px;
}
```

Listing 15. ID Selectors

Attribute Selector selector targets elements with a specific attribute or attribute value. For example, in the Listing 16 CSS rule, the selector **[title]** targets all elements that have a title attribute. The declaration **font-weight: bold;** specifies that the font weight of these elements should be bold.

```
/* Selects all elements with the title attribute and sets their font
weight to bold */
[title] {
    font-weight: bold;
}
```

Listing 16. Attribute Selector

CSS supports various units for specifying measurements, such as pixels (`**px**`), percentages (`**%**`), em, rem, etc. Understanding these units is crucial for creating responsive and scalable designs.

The CSS box model is a fundamental concept that describes the layout of elements on a web page. It comprises four main components: content, padding, border, and margin. These components collectively influence how elements are sized and spaced within a layout.

```
/* Adds padding of 20 pixels to all sides of the element */
.box {
    padding: 20px;
}
```

Listing 17. CSS Box model

In the Listing 17, the .box class represents an HTML element. The padding: 20px; declaration specifies that a padding of 20 pixels should be applied to all sides of the element, creating space between the content and the element's border. This padding affects the overall size and appearance of the element within the layout.

CSS frameworks, such as Bootstrap and Foundation, offer a collection of pre-designed CSS styles and components that facilitate web development. These frameworks provide a standardized set of styles for elements like buttons, forms, navigation bars, and more, allowing developers to create consistent and visually appealing web interfaces efficiently.

```
<!-- Example using Bootstrap classes to create a button -->
<button class="btn btn-primary">Click me</button>
```

Listing 18. Bootstrap class for Buttons

In Listing 18, Bootstrap classes are utilized to style a button element. The btn class applies basic button styling, while the btn-primary class specifies the button's primary color scheme. By leveraging CSS frameworks like Bootstrap, developers can expedite the development process and ensure consistency across different parts of their web applications. Familiarity with these frameworks can be advantageous for QA professionals involved in testing web applications, as it allows them to understand the standardized styling and behavior expected from various UI components.

By mastering CSS fundamentals and understanding its syntax, selectors, properties, and units, QA professionals can effectively test and ensure the visual consistency and responsiveness of web applications.

3.3 JavaScript

JavaScript is a high-level programming language commonly used for creating interactive and dynamic functionality on web pages. As a QA professional, understanding JavaScript fundamentals is essential for testing web applications effectively.

Key Concepts

JavaScript variables serve as containers for storing data values, providing a means to manipulate and manage data within a script. These variables can accommodate various data types, including numbers, strings, booleans, arrays, objects, and functions, enabling developers to work with diverse data structures and values effectively.

An example illustrating JavaScript variables and their data types is presented in Listing 19 below. This example showcases variable declarations and assignments, demonstrating how different types of data can be stored and accessed within a script.

```
// Variable declaration and assignment
let greeting = 'Hello, world!';
let age = 25;
let isUserLoggedIn = true;
let numbersArray = [1, 2, 3, 4, 5];
let person = { name: 'John', age: 30 };
```

Listing 19. JavaScript variables and data types.

In this listing, several variables are declared and initialized with values of different data types. For instance, the variable greeting stores a string value ('Hello, world!'), age holds a numerical value (25), isUserLoggedIn represents a boolean value (true), numbersArray contains an array of numbers ([1, 2, 3, 4, 5]), and person stores an object with properties describing a person (name: 'John', age: 30). This example demonstrates the versatility of JavaScript variables in accommodating various data types and structures.

Functions in JavaScript serve as reusable blocks of code designed to execute specific tasks. They offer a structured approach to organizing code logic and promoting reusability within scripts. Functions can accept input parameters and optionally return values, providing a flexible mechanism for performing various operations.

```
// Function declaration
function greet(name) {
    return 'Hello, ' + name + '!';
}

// Function call
let message = greet('Alice'); // Returns 'Hello, Alice!'
```

Listing 21. Functions

In this Listing 21, the **greet** function is declared with a single parameter **name**, which represents the name of the person to be greeted. Inside the function, a greeting message is constructed using the provided name, and the resulting message is returned to the caller. The function is then invoked with the argument 'Alice', resulting in the generation of the message 'Hello, Alice!'.

JavaScript supports conditional statements such as if, else if, and else, allowing developers to execute different code blocks based on specified conditions. These statements enable developers to implement decision-making logic within their scripts, facilitating dynamic behavior and response to varying circumstances.

```
let hour = new Date().getHours();
let greeting;

if (hour < 12) {
    greeting = 'Good morning!';
} else if (hour < 18) {
    greeting = 'Good afternoon!';
} else {
    greeting = 'Good evening!';
}
```

Listing 22. Conditional statements

In this Listing 22, the current hour of the day is retrieved using the **getHours** method of the **Date** object. Based on the value of the **hour** variable, a suitable **greeting** message is assigned to the greeting variable using conditional statements. Depending on the time of day, the script dynamically selects and assigns an appropriate greeting message ('Good morning!', 'Good afternoon!', or 'Good evening!').

JavaScript provides various types of loops, including for, while, and do-while, for iterating over arrays, objects, or executing code repeatedly.

```
// Loop through an array
let numbers = [1, 2, 3, 4, 5];
for (let i = 0; i < numbers.length; i++) {
    console.log(numbers[i]);
}
```

Listing 23. Loops

In this Listing 23, a for loop iterates over the elements of the 'numbers' array. It starts with an index variable 'i' initialized to 0 and continues as long as 'i' is less than the length of the array. The loop body logs each element of the array to the console.

The Document Object Model (DOM) represents the structure of HTML documents, and JavaScript allows one to manipulate it dynamically. One can access and modify HTML elements, attributes, and styles using JavaScript.

```
// Change text content of an element
document.getElementById('myElement').textContent = 'New text';

// Add a CSS class to an element
document.querySelector('.box').classList.add('highlight');
```

Listing 24. DOM manipulation

The Listing 24 demonstrate essential DOM manipulation techniques. In the first snippet, the getElementById method targets an HTML element with the ID 'myElement', subsequently updating its text content to 'New text' via the textContent property. This action dynamically modifies the displayed text within the specified element. In the second snippet, the querySelector method selects the first HTML element with the class 'box', followed by the classList.add method, which adds the 'highlight' CSS class to the selected element. This addition of a CSS class enables the application of specific styles to the element, facilitating visual enhancements or modifications. Overall, these snippets exemplify JavaScript's capability to interact with and manipulate the DOM, empowering developers to create dynamic and engaging web experiences.

JavaScript includes error handling mechanisms like try, catch, and finally blocks to handle exceptions and prevent runtime errors from crashing the application.

JavaScript enables event-driven programming, allowing one to respond to user interactions like clicks, mouse movements, keyboard inputs, etc., by attaching event handlers to HTML elements.

JavaScript supports asynchronous programming with features like callbacks, promises, and async/await, enabling non-blocking execution of code and better handling of tasks such as fetching data from servers.

By mastering JavaScript basics, including variables, functions, conditionals, loops, DOM manipulation, and error handling, QA professionals can effectively test web applications for functionality, interactivity, and responsiveness.

4 Tools available for QA (Quality Assurance) professionals

There are numerous tools available for QA (Quality Assurance) professionals, catering to various aspects of the testing process. Here are some important QA tools along with examples.

1. Jira (Test Management Tool)
2. Jmeter (Performance Testing Tool)
3. Postman (API Testing Tool)
4. Robot Framework Based on Selenium

4.1 Jira

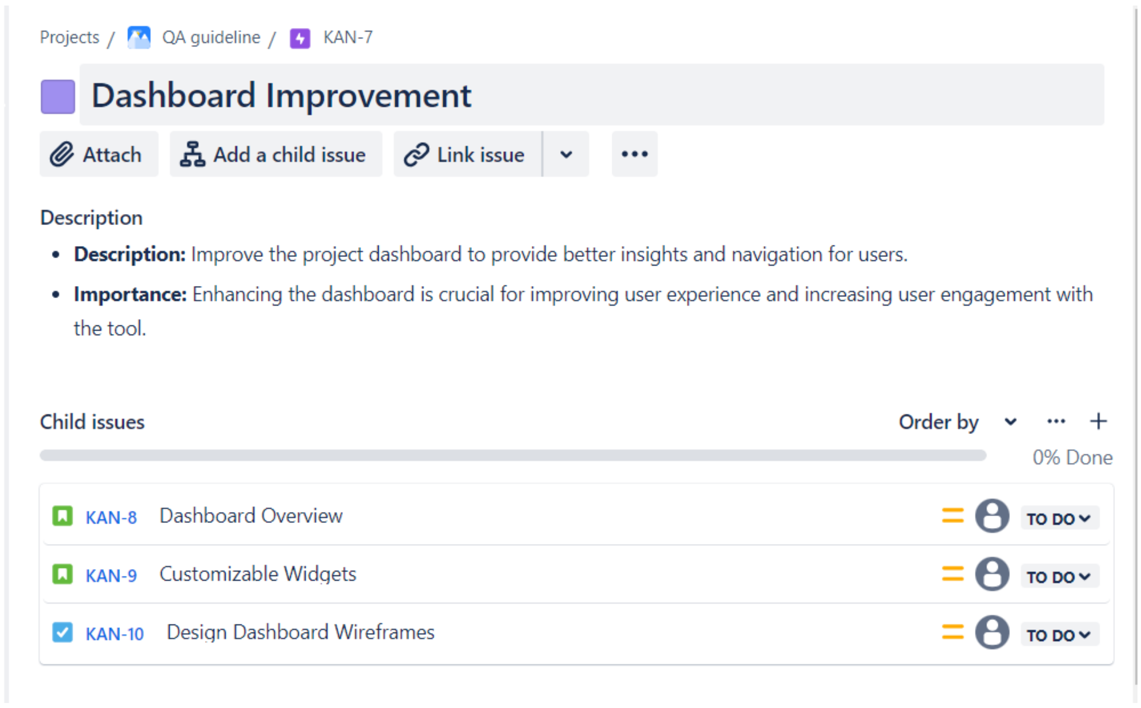
Jira is a helpful tool for managing projects. It helps teams work together better by organizing tasks and keeping track of what needs to be done. In this section, we'll learn about some important features of Jira and why they're useful for getting things done efficiently.

4.1.1 Epics and Stories

Imagine building a house. An **epic** is like saying, "Let's build a house." It's a big idea or goal. **Stories** are like saying, "Let's build a kitchen," "Let's paint the walls," or "Let's install a door." They are smaller tasks that help achieve the big goal.

Epics help us see the big picture of what we're trying to do. Stories help us break down the big tasks into smaller, manageable pieces. They make it easier to understand and work on.

Let’s Create Epics and stories with example.
To create an epic or story in Jira, one usually go to one’s project, click "Create," and choose either "Epic" or "Story" from the options.
Then one give it a name and describe what it's about. For an epic, one might describe the big goal. For a story, one might explain the specific task.


Projects / QA guideline / KAN-7

Dashboard Improvement

Description

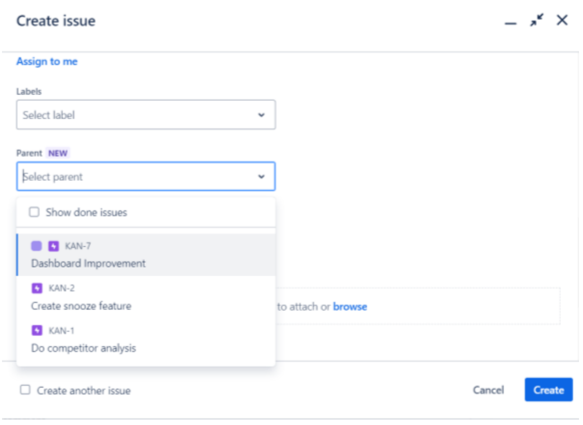
- **Description:** Improve the project dashboard to provide better insights and navigation for users.
- **Importance:** Enhancing the dashboard is crucial for improving user experience and increasing user engagement with the tool.

Child issues — Order by — 0% Done

KAN-8	Dashboard Overview	TO DO
KAN-9	Customizable Widgets	TO DO
KAN-10	Design Dashboard Wireframes	TO DO

Figure 1. Epic with the description and goal.

When creating a Story, one need to select the parent Epic from the dropdown menu under which one want to create the Story.


Create issue

Assign to me

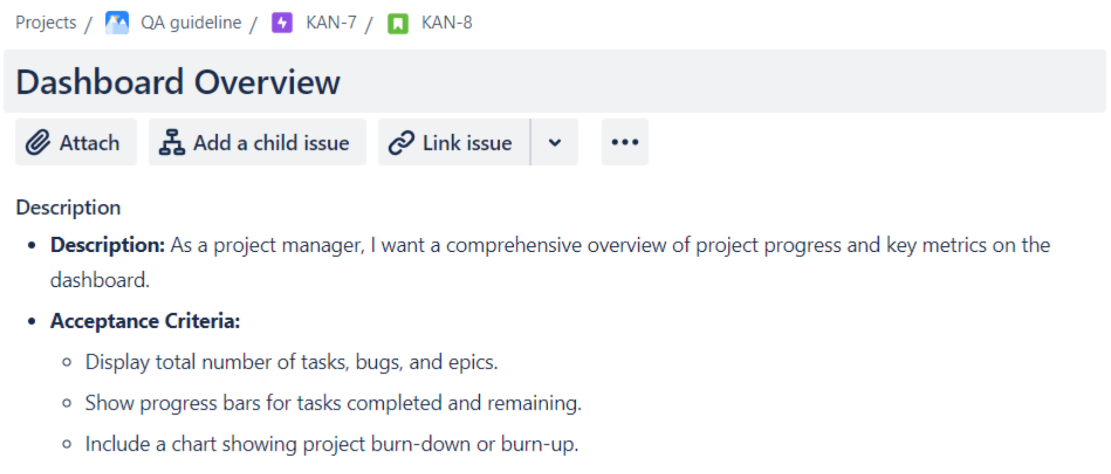
Labels: Select label

Parent NEW: Select parent

- Show done issues
- KAN-7 Dashboard Improvement
- KAN-2 Create snooze feature
- KAN-1 Do competitor analysis

Figure 2. Selecting a parent in a Stroy

When creating a Story, it's crucial to include a clear description and well-defined acceptance criteria. These details serve as guidelines for both developers and QA/testers, ensuring a common understanding of the task's requirements and expected outcomes.



Projects / QA guideline / KAN-7 / KAN-8

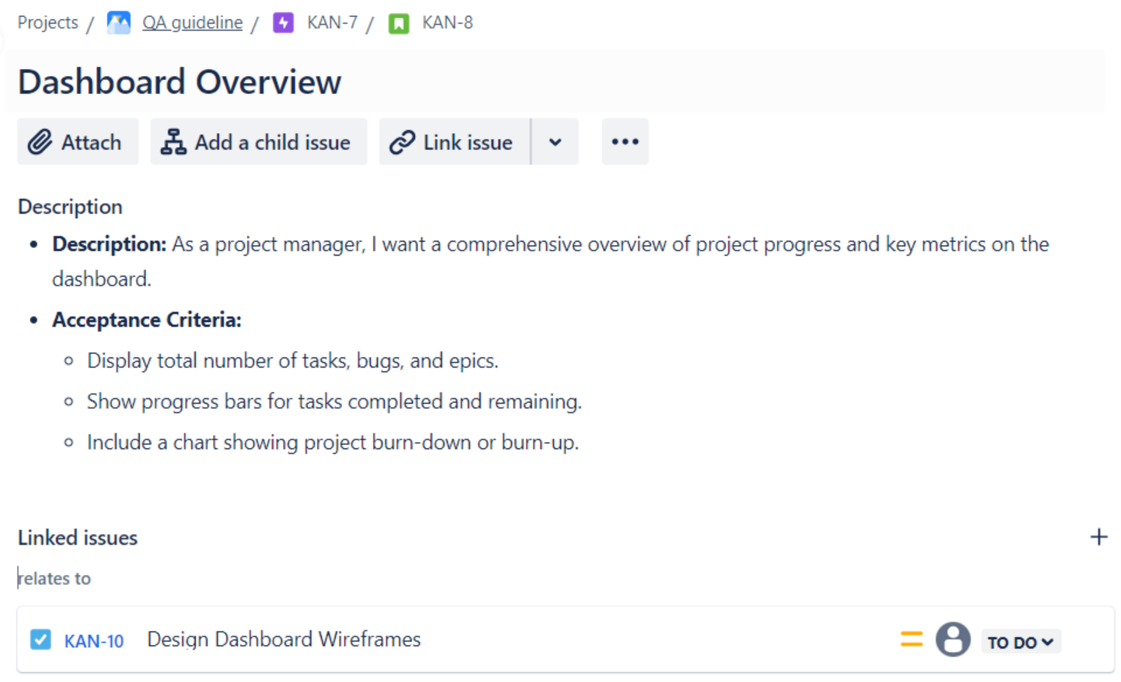
Dashboard Overview

Description

- **Description:** As a project manager, I want a comprehensive overview of project progress and key metrics on the dashboard.
- **Acceptance Criteria:**
  - Display total number of tasks, bugs, and epics.
  - Show progress bars for tasks completed and remaining.
  - Include a chart showing project burn-down or burn-up.

Figure 3. Example of story with description and acceptance criteria.

When one create a Story, make sure to split it into smaller tasks and subtasks. This helps keep things organized and makes it easier to understand what needs to be done, which is important for smooth development and testing



Projects / QA guideline / KAN-7 / KAN-8

Dashboard Overview

Description

- **Description:** As a project manager, I want a comprehensive overview of project progress and key metrics on the dashboard.
- **Acceptance Criteria:**
  - Display total number of tasks, bugs, and epics.
  - Show progress bars for tasks completed and remaining.
  - Include a chart showing project burn-down or burn-up.

Linked issues

relates to

KAN-10 Design Dashboard Wireframes TO DO

Figure 4. Task Linked to Story

We can give tasks, stories, or big projects to the person who will do the work.

4.1.2 Report a Bug

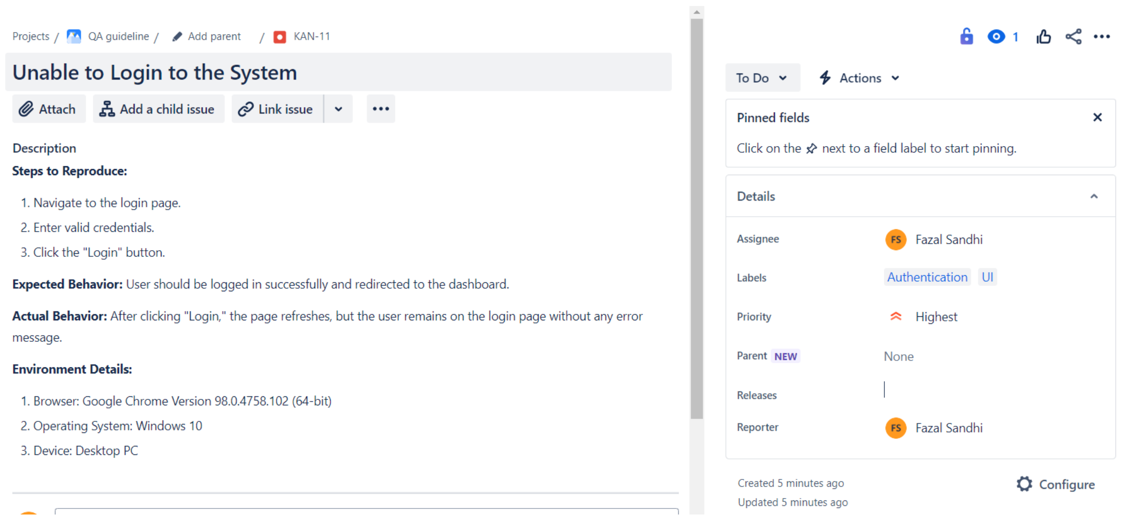
When one encounter something unexpected, broken, or when a program doesn't perform as intended, it's referred to as a bug.

To report a bug in Jira, first, click the "Create" button at the top-right corner of the screen. Then, choose "Bug" from the list of issue types. Write a brief title summarizing the problem and describe the bug in detail.

In the description field, provide a detailed explanation of the bug. Include information such as:

- Steps to reproduce the bug.
- Expected behavior.
- Actual behavior (what one observed).
- Environment details (e.g., browser version, operating system).
- Any relevant screenshots or attachments.

After describing the bug, it's important to set its priority and severity. This helps show how urgent it is to fix the bug and how much it affects the system or users. Then, assign the bug to the right team or person responsible for fixing it. Also, add labels like "UI" or "performance" to categorize the bug for better organization and understanding.


Projects / QA guideline / Add parent / KAN-11
Attach
Add a child issue
Link issue
To Do
Actions
Pinned fields
Click on the next to a field label to start pinning.
Configure

Unable to Login to the System

Description
Steps to Reproduce:

1. Navigate to the login page.
2. Enter valid credentials.
3. Click the "Login" button.

Expected Behavior: User should be logged in successfully and redirected to the dashboard.

Actual Behavior: After clicking "Login," the page refreshes, but the user remains on the login page without any error message.

Environment Details:

1. Browser: Google Chrome Version 98.0.4758.102 (64-bit)
2. Operating System: Windows 10
3. Device: Desktop PC

Details

Assignee	FS Fazal Sandhi
Labels	Authentication UI
Priority	Highest
Parent NEW	None
Releases	
Reporter	FS Fazal Sandhi

Created 5 minutes ago
Updated 5 minutes ago

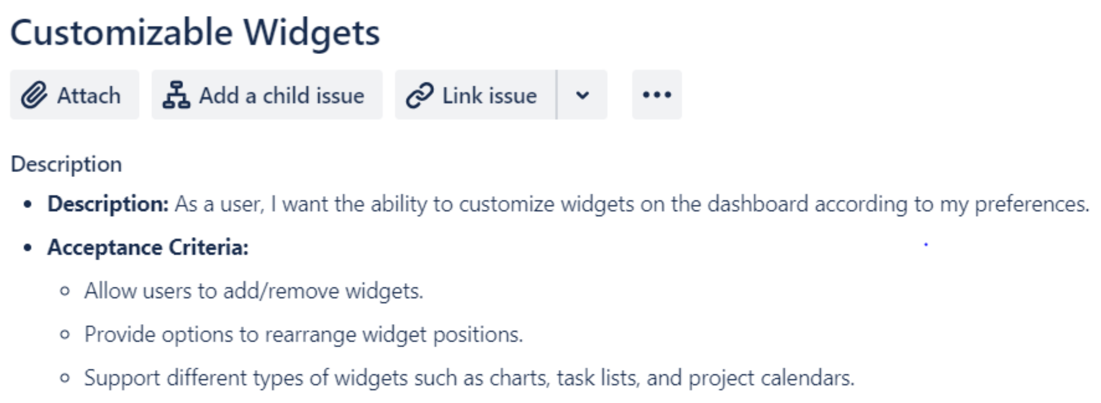
Figure 5. Bug example

4.1.3 Backlog refinement

Backlog refinement is when we look at the list of things we want to do in a software project, decide which ones are most important, and figure out how to do them. As a QA person, one get to join in these discussions to make sure the upcoming features are easy to test and to spot any problems early on. In Jira, which is a tool we use to keep track of our work, we can create and organize these tasks, talk to each other about them, and see how things are progressing. So, basically, it's a way for the team to plan and organize their work, and for QA to make sure testing is on track.

4.1.4 Acceptance criteria

In Jira, acceptance criteria are like a checklist that tells us when a task or story is done right. They're simple statements that explain what needs to happen for the work to be finished. For example, if we're building a login page, the acceptance criteria might say that users should be able to enter their username and password and successfully log in. As a QA person, having these criteria helps us know what to test and make sure everything works as it should. In Jira, we can write down these criteria right where we're working on the task, so everyone knows what needs to be done. It keeps us all on the same page and helps us build things the right way.



Customizable Widgets

Description

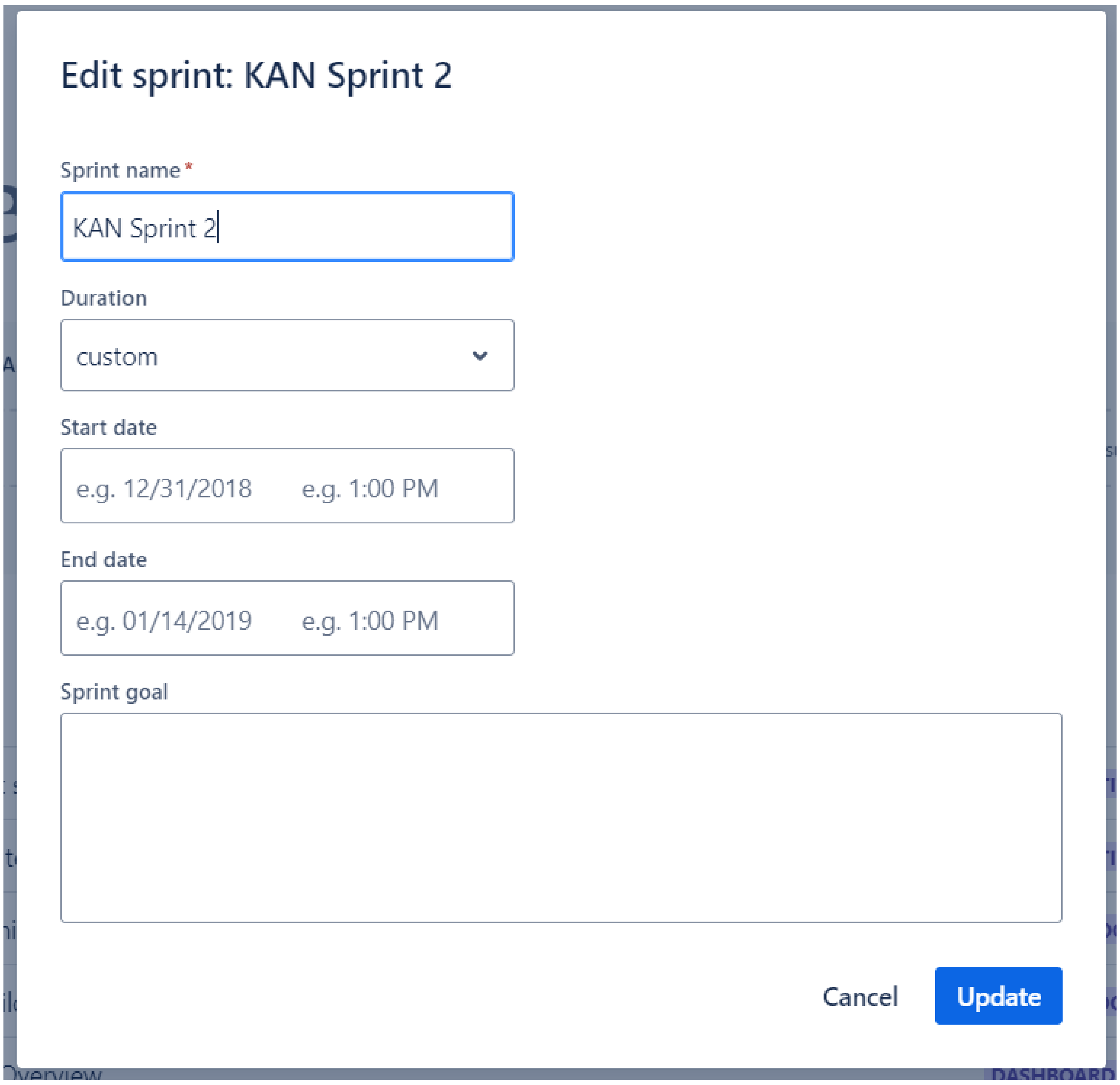
- **Description:** As a user, I want the ability to customize widgets on the dashboard according to my preferences.
- **Acceptance Criteria:**
  - Allow users to add/remove widgets.
  - Provide options to rearrange widget positions.
  - Support different types of widgets such as charts, task lists, and project calendars.

Figure 6. Story with acceptance criteria

4.1.5 Sprint

In Agile, a sprint is like a short, focused work period, usually lasting a couple of weeks. During a sprint, a team tackles a specific set of tasks or stories they've planned out beforehand. It's a way to stay on track and deliver useful stuff to the project bit by bit.

To create a sprint, one need to first enable the feature in the project settings. One can do this by going to "Project Settings" and then selecting "Features." Once one have enabled sprints, one will see the option to create a new sprint.
Now, navigate to the "Boards" section from the left navigation menu and click on the "Create Sprint" button located on the left side of the columns. After clicking the button, a new section will appear on top of the backlog. Here, one can edit the sprint name, add some goals, and set the time period for the sprint. Once one have done that, one can drag and drop the tickets one want to complete during that sprint. After selecting the tickets, one can start the sprint by clicking on the "Start" button.



Edit sprint: KAN Sprint 2

Sprint name*

KAN Sprint 2

Duration

custom

Start date

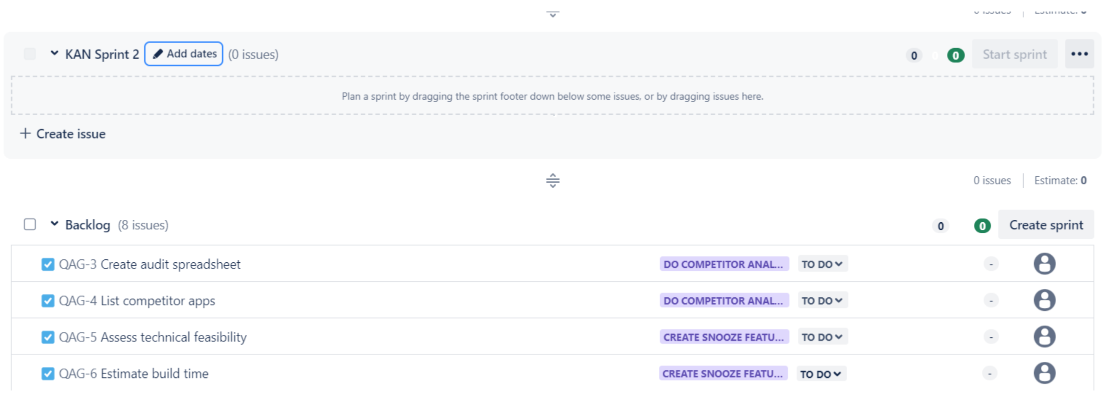
e.g. 12/31/2018 e.g. 1:00 PM

End date

e.g. 01/14/2019 e.g. 1:00 PM

Sprint goal

Figure 7. Screen to add sprint goals and timeline



KAN Sprint 2 Add dates (0 issues)

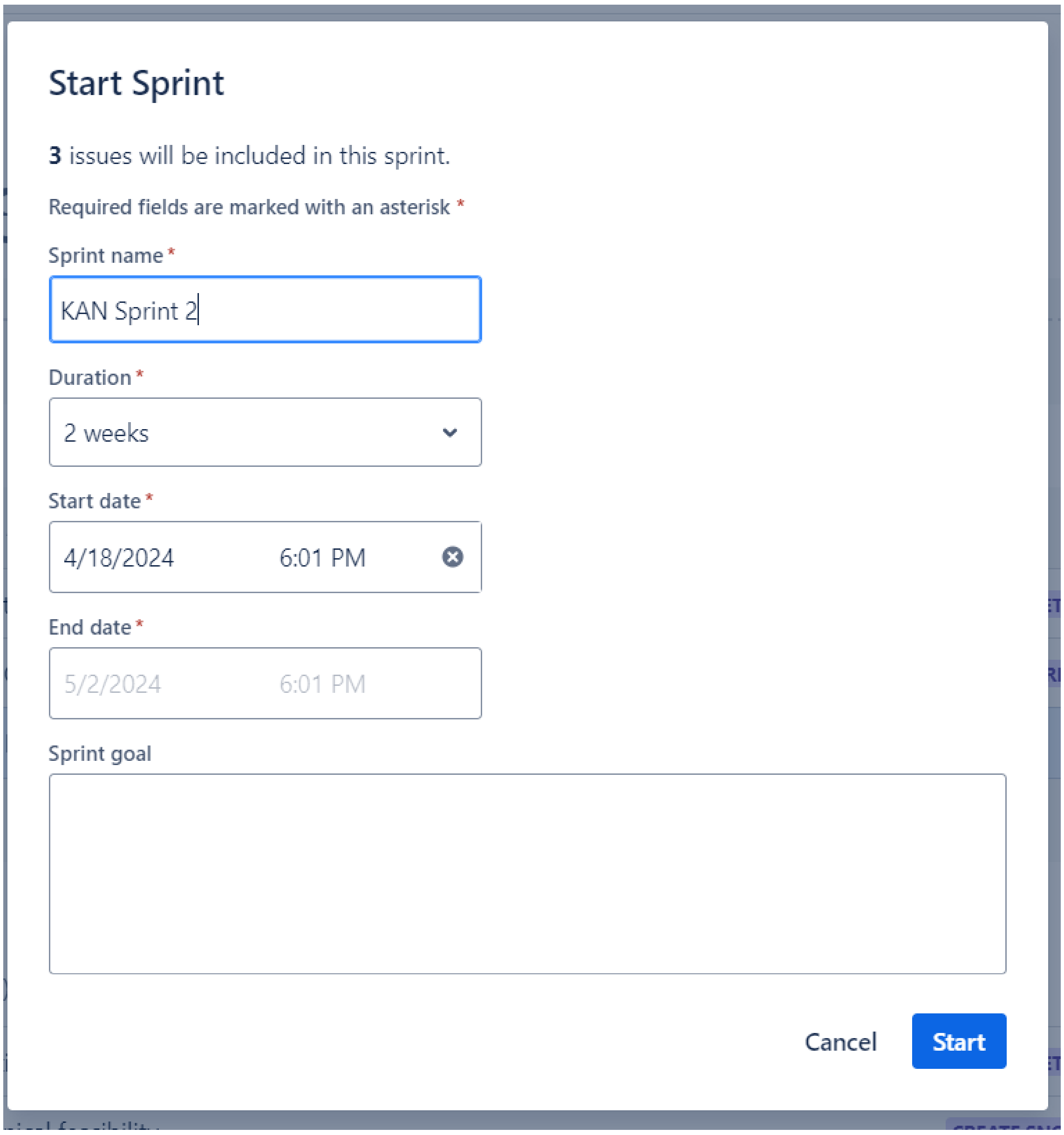
Plan a sprint by dragging the sprint footer down below some issues, or by dragging issues here.

0 issues | Estimate: 0

Backlog (8 issues)

QAG-3 Create audit spreadsheet	DO COMPETITOR ANAL...	TO DO	-
QAG-4 List competitor apps	DO COMPETITOR ANAL...	TO DO	-
QAG-5 Assess technical feasibility	CREATE SNOOZE FEATU...	TO DO	-
QAG-6 Estimate build time	CREATE SNOOZE FEATU...	TO DO	-

Figure 8. Sprint area for selecting tickets


Start Sprint

3 issues will be included in this sprint.

Required fields are marked with an asterisk *

Sprint name *
KAN Sprint 2

Duration *
2 weeks

Start date *
4/18/2024 6:01 PM

End date *
5/2/2024 6:01 PM

Sprint goal

Figure 9. Start sprint

4.1.6 Xray installation and configuration

Xray is a popular test management app for Jira that enhances testing capabilities within Jira.

To install Xray Test Management for Jira, start by clicking on "Apps" beside the "Create" button in one's Jira instance. Then, click on "Explore more" and search for the keyword "Xray." From the search results, click on "Xray Test Management

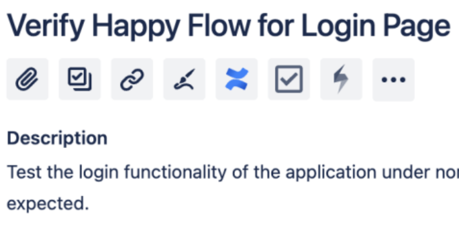
Tool for Jira," and then click on "Try it for free" on the popup. Next, click on "Start free trial." This may take some time. Once the app is installed, one will see a notification in the bottom-left corner to get started with configuration. Review all the configurations and update settings if needed. Now, go to "Apps," click on "Manage apps," then click on "Xray," and finally click on "Get started." Here, click on the "Configure project" button. A popup will appear; select the project where one want to add Xray and click on "Configure." After selecting the project, wait for some time. After that, one will need to manually add the issue type for Xray from the project's issue type settings.

4.1.7 Create Xray

To create an Xray test case in Jira, first, click on the "Create" button, then select "Xray." Next, add the summary and details about the test case the one creating. Once one have added the basic details, one need to include the steps to perform the specific task.

For example, if one is creating an Xray test case for the login page's happy flow (where all information is correct and no errors are expected), follow these steps:

1. Click on the "Add Step" button. A new area will appear with three sections: Action, Data, and Expected Result.
    - In the Action section, write "Go to Login page" or "Visit the login page" and provide the link to the page in the Data section.
    - In the Expected Result section, write "Login page should be visible." One can also add a screenshot of the login page.
2. Click on "New Step" to add the next action.
    - In the Action section, write "Add username and password provided in the Data files and click on the login button."
    - In the Data section, provide the username and password.
    - In the Expected Result section, write "Success message should appear, and the user should be redirected to the dashboard." One can also add a screenshot of the dashboard.


Verify Happy Flow for Login Page

Description

Test the login functionality of the application under normal conditions, where all information provided is correct, and no errors are expected.

Figure 10. Xray Title and detail

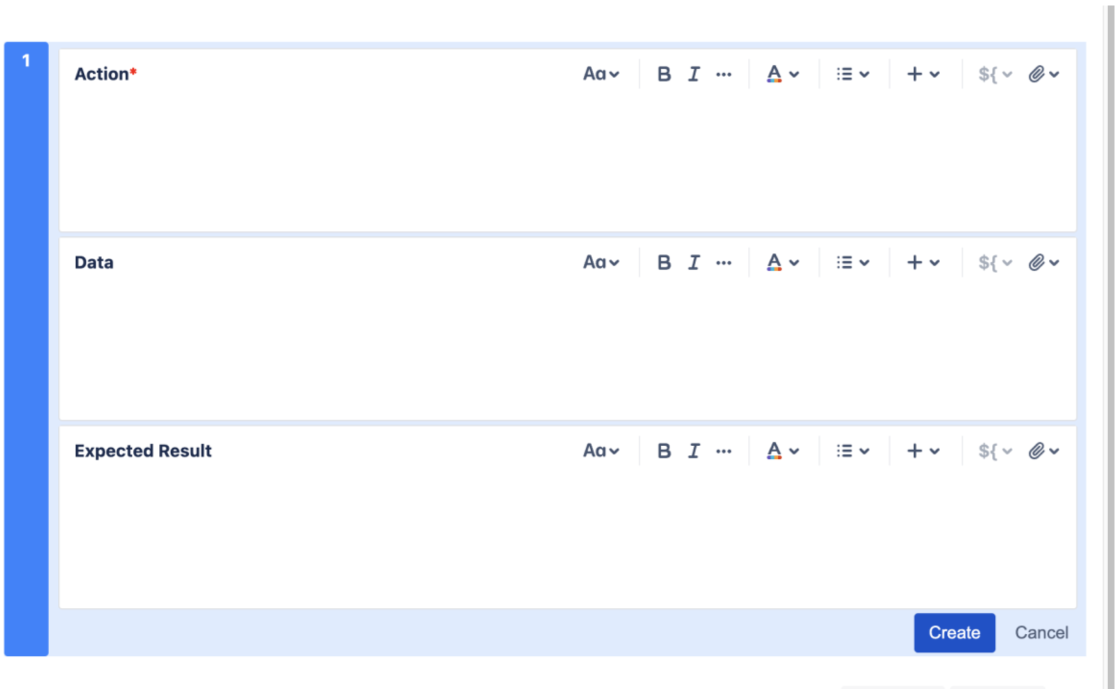
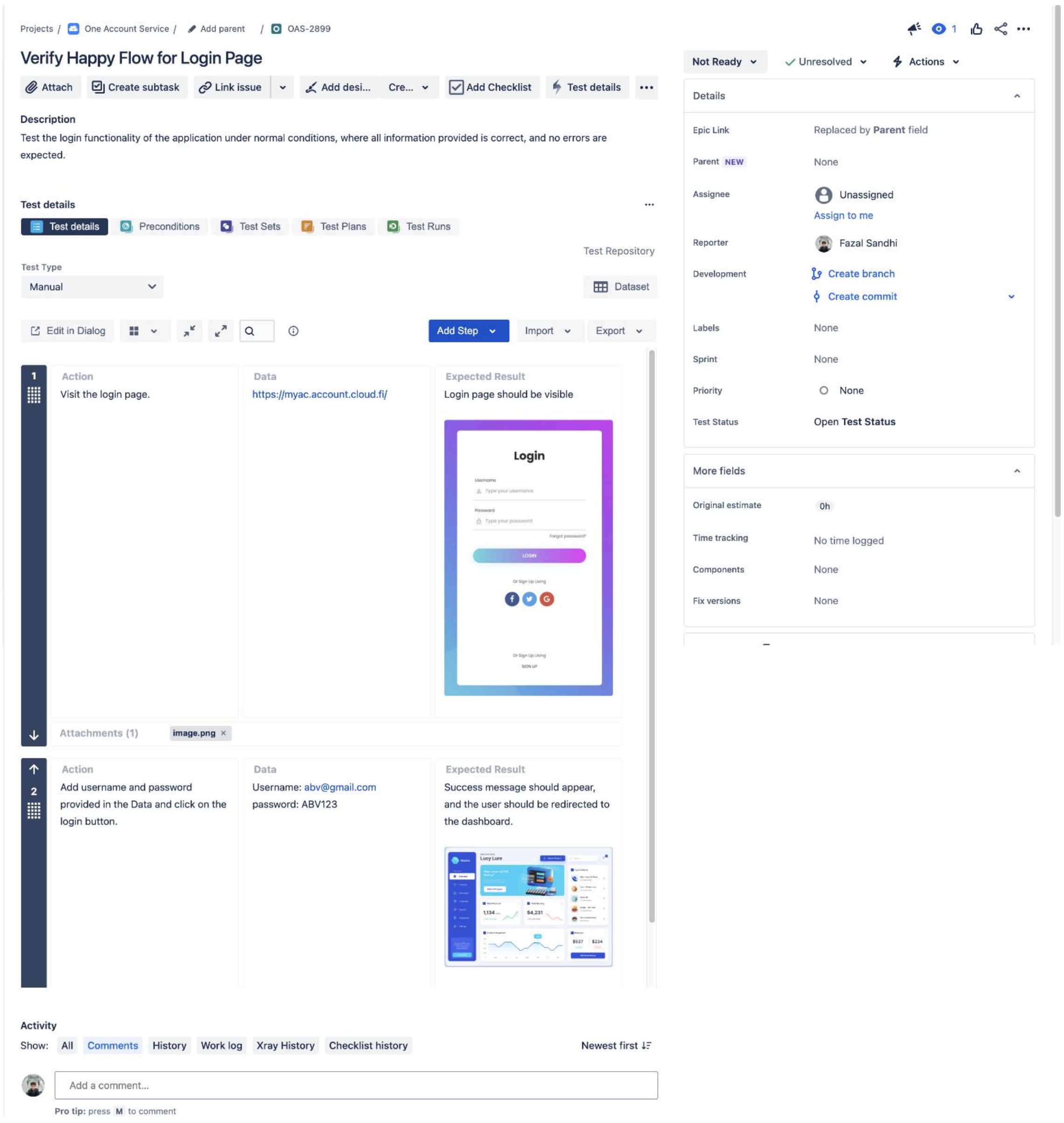


Figure 11. Area to add the steps



Projects / One Account Service / Add parent / OAS-2899

Verify Happy Flow for Login Page

Attach | Create subtask | Link issue | Add desi... | Cre... | Add Checklist | Test details

Description

Test the login functionality of the application under normal conditions, where all information provided is correct, and no errors are expected.

Test details

Test details | Preconditions | Test Sets | Test Plans | Test Runs

Test Repository

Test Type: Manual

Dataset

Edit in Dialog | Add Step | Import | Export

	Action	Data	Expected Result
1	Visit the login page.	https://myac.account.cloud.fi/	Login page should be visible
2	Add username and password provided in the Data and click on the login button.	Username: abv@gmail.com password: ABV123	Success message should appear, and the user should be redirected to the dashboard.

Attachments (1) image.png

Activity

Show: All | Comments | History | Work log | Xray History | Checklist history — Newest first

Add a comment...

Pro tip: press M to comment

Not Ready | Unresolved | Actions

Details

Epic Link	Replaced by Parent field
Parent NEW	None
Assignee	Unassigned, Assign to me
Reporter	Fazal Sandhi
Development	Create branch, Create commit
Labels	None
Sprint	None
Priority	None
Test Status	Open Test Status

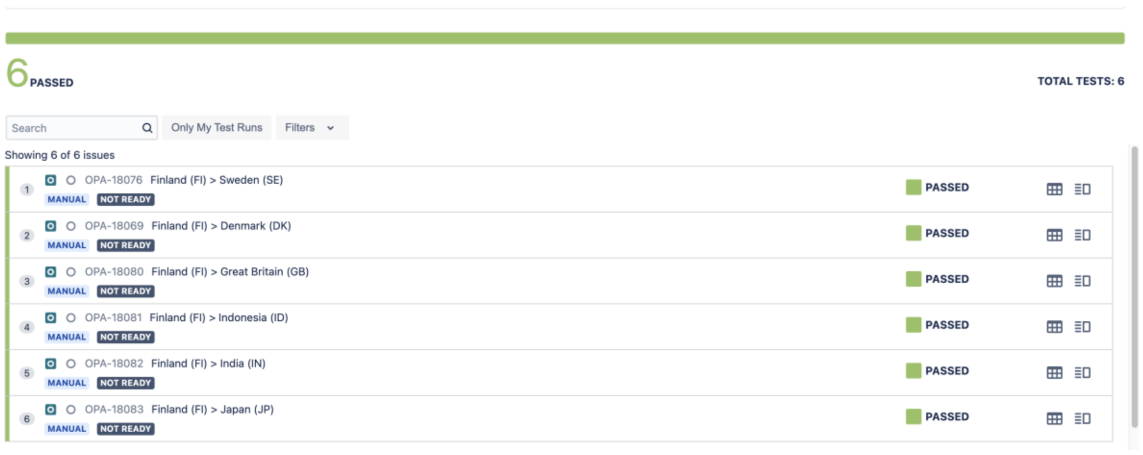
More fields

Original estimate	0h
Time tracking	No time logged
Components	None
Fix versions	None

Figure 12. Complete xray

4.1.8 Test Execution

In Jira, test execution means running tests to check if software works correctly. Here's how one do it: first, one go to the part of Jira where tests are managed. Then, one pick the tests one want to run. When one is ready, one start the test. One follow the steps in each test, like clicking buttons or entering information. As one do this, one note down what happens—whether it works as expected or if there are any problems. If something doesn't work right, one let the team know so they can fix it. Once one finish all the tests, one mark them as done. That's how one perform test execution in Jira—it's all about checking that the software does what it's supposed to do.


6 PASSED

TOTAL TESTS: 6

Search | Only My Test Runs | Filters

Showing 6 of 6 issues

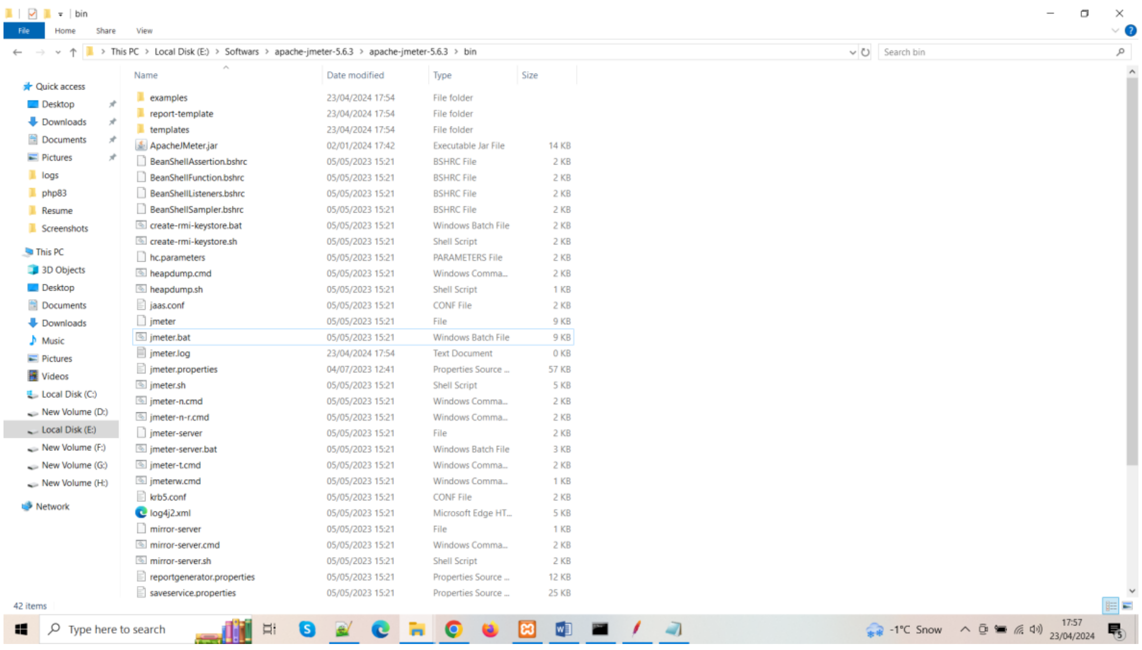
1	OPA-18076	Finland (FI) > Sweden (SE)	MANUAL NOT READY	PASSED
2	OPA-18069	Finland (FI) > Denmark (DK)	MANUAL NOT READY	PASSED
3	OPA-18080	Finland (FI) > Great Britain (GB)	MANUAL NOT READY	PASSED
4	OPA-18081	Finland (FI) > Indonesia (ID)	MANUAL NOT READY	PASSED
5	OPA-18082	Finland (FI) > India (IN)	MANUAL NOT READY	PASSED
6	OPA-18083	Finland (FI) > Japan (JP)	MANUAL NOT READY	PASSED

Figure 13. Test Execution Result

4.2 JMeter

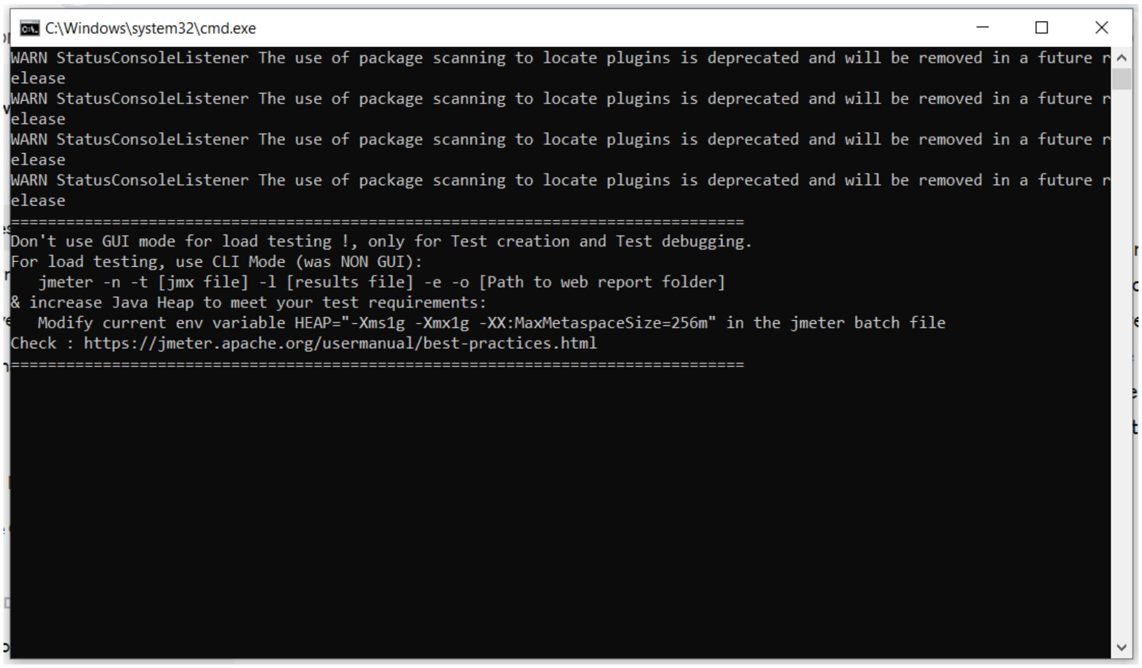
This section explains using JMeter for performance testing. First, learn how to install JMeter on the machine. Then, find out how to create a test plan to decide what and how to test. This includes setting up things like simulating users and checking results. Once the plan is ready, then see how to run the test and check the results.

Getting started with Apache JMeter is straightforward. First, head over to the official Apache JMeter website and download the latest version that matches one's operating system. Once downloaded, simply extract the files to a folder of one's preference. To launch JMeter, navigate to the "bin" directory within the extracted folder and run the appropriate script—either "jmeter.bat" for Windows or "jmeter.sh" for Unix/Linux systems. These steps ensure that one have JMeter up and running on one's machine, ready for performance testing tasks.



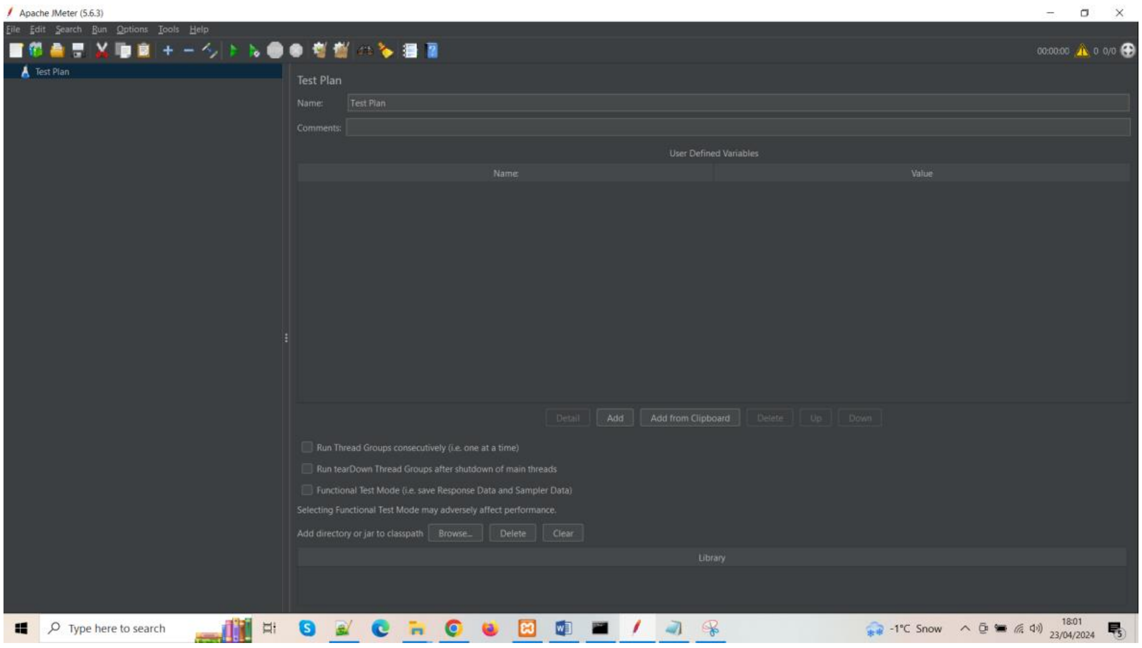
Name	Date modified	Type	Size
examples	23/04/2024 17:54	File folder	
report-template	23/04/2024 17:54	File folder	
templates	23/04/2024 17:54	File folder	
ApacheJMeter.jar	02/01/2024 17:42	Executable Jar File	14 KB
BeanShellAssertion.bshrc	05/05/2023 15:21	BSHRC File	2 KB
BeanShellFunction.bshrc	05/05/2023 15:21	BSHRC File	2 KB
BeanShellListeners.bshrc	05/05/2023 15:21	BSHRC File	2 KB
BeanShellSampler.bshrc	05/05/2023 15:21	BSHRC File	2 KB
create-rmi-keystore.bat	05/05/2023 15:21	Windows Batch File	2 KB
create-rmi-keystore.sh	05/05/2023 15:21	Shell Script	2 KB
hc.parameters	05/05/2023 15:21	PARAMETERS File	2 KB
heapdump.cmd	05/05/2023 15:21	Windows Comma...	2 KB
heapdump.sh	05/05/2023 15:21	Shell Script	1 KB
jaas.conf	05/05/2023 15:21	CONF File	2 KB
jmeter	05/05/2023 15:21	File	9 KB
jmeter.bat	05/05/2023 15:21	Windows Batch File	9 KB
jmeter.log	23/04/2024 17:54	Text Document	0 KB
jmeter.properties	04/07/2023 12:41	Properties Source ...	57 KB
jmeter.sh	05/05/2023 15:21	Shell Script	5 KB
jmeter-n.cmd	05/05/2023 15:21	Windows Comma...	2 KB
jmeter-n-r.cmd	05/05/2023 15:21	Windows Comma...	2 KB
jmeter-server	05/05/2023 15:21	File	2 KB
jmeter-server.bat	05/05/2023 15:21	Windows Batch File	3 KB
jmeter-t.cmd	05/05/2023 15:21	Windows Comma...	2 KB
jmeterw.cmd	05/05/2023 15:21	Windows Comma...	1 KB
krb5.conf	05/05/2023 15:21	CONF File	2 KB
log4j2.xml	05/05/2023 15:21	Microsoft Edge HT...	5 KB
mirror-server	05/05/2023 15:21	File	1 KB
mirror-server.cmd	05/05/2023 15:21	Windows Comma...	2 KB
mirror-server.sh	05/05/2023 15:21	Shell Script	2 KB
reportgenerator.properties	05/05/2023 15:21	Properties Source ...	12 KB
saveservice.properties	05/05/2023 15:21	Properties Source ...	25 KB

Figure 14. the "bin" folder of the installation.



```
C:\Windows\system32\cmd.exe
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future r
elease
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future r
elease
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future r
elease
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future r
elease
================================================================================
Don't use GUI mode for load testing !, only for Test creation and Test debugging.
For load testing, use CLI Mode (was NON GUI):
   jmeter -n -t [jmx file] -l [results file] -e -o [Path to web report folder]
& increase Java Heap to meet your test requirements:
   Modify current env variable HEAP="-Xms1g -Xmx1g -XX:MaxMetaspaceSize=256m" in the jmeter batch file
Check : https://jmeter.apache.org/usermanual/best-practices.html
================================================================================
```

Figure 15. jmeter.bat output



Test Plan

Name: Test Plan

Comments:

Run Thread Groups consecutively (i.e. one at a time)

Run tearDown Thread Groups after shutdown of main threads

Functional Test Mode (i.e. save Response Data and Sampler Data)

Selecting Functional Test Mode may adversely affect performance.

Add directory or jar to classpath

Figure 16. JMeter UI

Creating a test plan in Apache JMeter is easy. First, when one open JMeter, one will see its interface. It's where one make one’s performance tests. Now, in a test plan, one have threads and thread groups. Threads are like pretend users, and thread groups are where one set up how these pretend users will act. For example, one might want to pretend 100 users are visiting one’s website at the same time. One do this by setting up a thread group. To do that, one right-click on "Test Plan" on the left side and choose "Add" -> "Threads (Users)" -> "Thread Group". This lets one set how many pretend users there are, how fast they show up, and how many times they repeat their actions. Once Ons’ve set up one’s thread group, one can add more things like samplers and listeners to complete one’s test plan and see how well one’s website or app handles the pretend traffic.

Configuring a thread group in Apache JMeter is the next step after creating one’s test plan. To start configuring, right-click on the thread group one just made. Then, select "Add" and "Sampler". This allows one to add different types of requests to one’s test plan, like HTTP requests or others, depending on what one want to test. For instance, if one is testing a website, one might add an HTTP request sampler to simulate users accessing web pages. This step is crucial for setting up the actions one’s pretend users will take during the test.

Adding listeners in Apache JMeter helps one see the results of one’s performance test in a clear way. To start, one right-click on the thread group one created or any sampler one added earlier. Then, one choose "Add" and "Listener". Think of listeners like a window where one can peek into what's happening during one’s test. They show one things like response times, errors, and other important details. For example, if one want to see how fast one’s website is responding to requests, one might add a "View Results Tree" listener. This listener displays all the responses from one’s server, letting one check if everything is working as expected. Adding listeners is crucial for understanding how one’s application performs under different conditions and helps one spot any issues that need fixing.

Running one’s test in Apache JMeter is as simple as clicking a button. To start, one just need to click on the "Play" button in the toolbar. This button looks like a triangle pointing to the right. When one click it, JMeter will begin executing all the requests one have set up in one’s test plan. It's like pressing "go" and letting JMeter do its thing. This step is crucial because it's when one’s test actually runs and one gather data about how one’s application performs under the load one have simulated. It's a bit like starting a race—one is eager to see how well one's application handles the traffic one have thrown at it. Running the test lets one collect valuable information that one can use to optimize one's application and make it more robust.

To perform performance testing on a public URL, such as the OpenAI homepage, using Apache JMeter, first, ensure one have JMeter installed and open a new test plan. Within the test plan, create a thread group to simulate users accessing the webpage concurrently. Then, add an HTTP Request sampler and specify the URL of the webpage one want to test, like "https://www.openai.com/". After configuring the thread group and sampler, add a listener, such as the View Results Tree, to observe the test results. Finally, run the test by clicking the "Play" button, and JMeter will execute the requests to the specified URL, providing one with performance metrics such as response times and throughput. Analyze the results to assess how the webpage performs under the simulated load. Remember to start with a small number of threads and gradually increase to avoid overwhelming the server, and always adhere to the website's terms of service and performance testing guidelines.

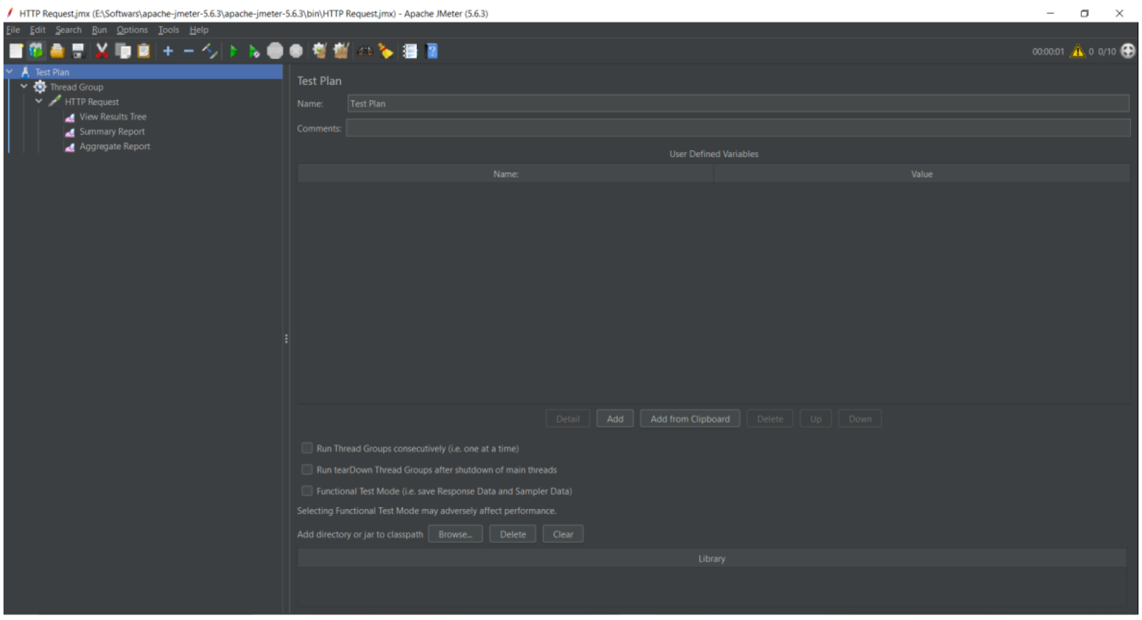
Analyzing the results of one's performance test in Apache JMeter is crucial for understanding how well one's website or application performs. Let's break down the key listeners:

1. **View Results Tree**: This listener lets one see individual requests made during the test. One can check details like request and response data, headers, and response times. It's like looking at each action taken by one's application one by one, helping one spot any specific issues or slowdowns.

2. **Summary Report**: With this listener, one get a summary of the overall test results. It shows metrics such as average response time, requests per second (throughput), error rate, and median response time. It's handy for getting a quick overview of how one's application performed during the test, highlighting any trends or areas needing attention.

3. **Aggregate Report**: This listener combines results from multiple samples into one table. It provides statistics like average, median, and minimum and maximum response times. This is helpful for comparing different parts of one's test plan or configurations, pinpointing where improvements are needed.

These listeners, along with others like Graph Results and Assertion Results, help one dig deeper into one's test data. By running tests multiple times under various conditions, one can ensure the accuracy of one's findings. Analyzing the results lets one identify bottlenecks, validate optimizations, and make informed decisions to enhance one's application's performance and scalability.



HTTP Request.jmx (E:\Softwars\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin\HTTP Request.jmx) - Apache JMeter (5.6.3)

- Test Plan
  - Thread Group
    - HTTP Request
      - View Results Tree
      - Summary Report
      - Aggregate Report

Test Plan

Name: Test Plan

Comments:

User Defined Variables

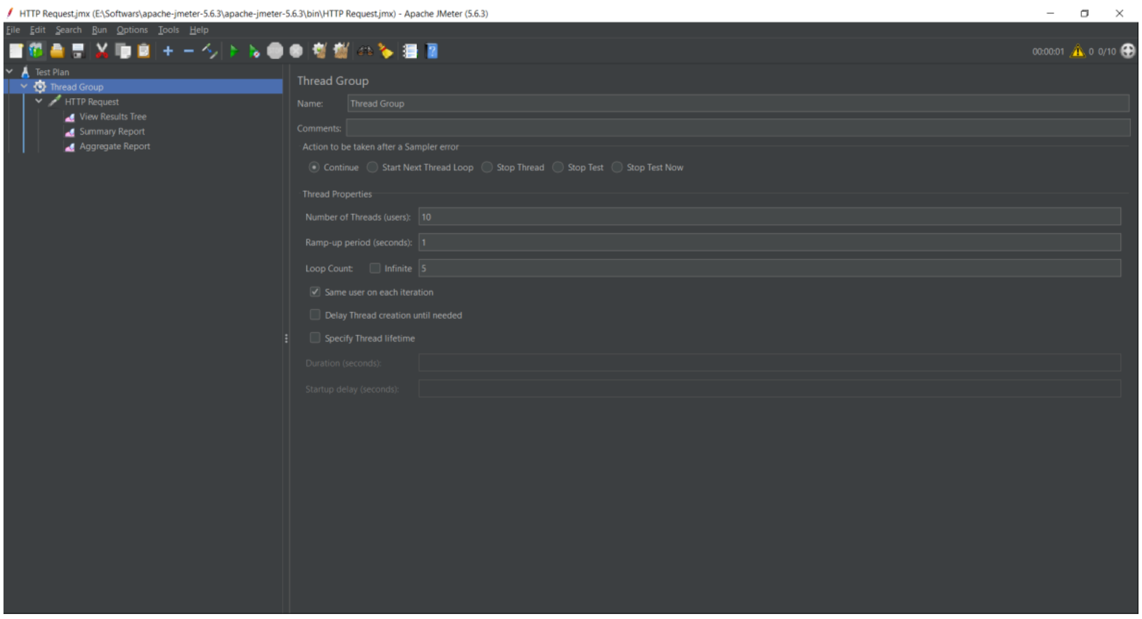
Name:	Value

- [ ] Run Thread Groups consecutively (i.e. one at a time)
- [ ] Run tearDown Thread Groups after shutdown of main threads
- [ ] Functional Test Mode (i.e. save Response Data and Sampler Data)

Selecting Functional Test Mode may adversely affect performance.

Add directory or jar to classpath

Figure 17. Test plan


HTTP Request.jmx (E:\Softwars\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin\HTTP Request.jmx) - Apache JMeter (5.6.3)
File Edit Search Run Options Tools Help
00:00:01 0 0/10
Test Plan
Thread Group
HTTP Request
View Results Tree
Summary Report
Aggregate Report

Thread Group

Name: Thread Group

Comments:

Action to be taken after a Sampler error

[✓ Continue] Start Next Thread Loop Stop Thread Stop Test Stop Test Now

Thread Properties

Number of Threads (users): 10

Ramp-up period (seconds): 1

Loop Count: ☐ Infinite 5

☑ Same user on each iteration

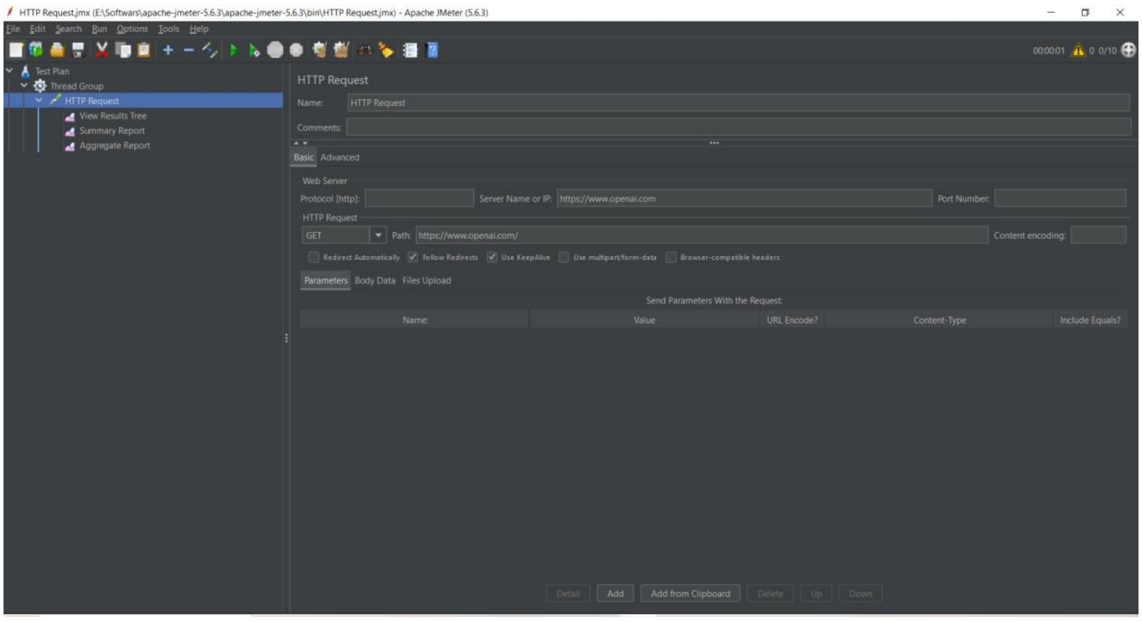
☐ Delay Thread creation until needed

☐ Specify Thread lifetime

Duration (seconds):

Startup delay (seconds):

Figure 18. Thread group


HTTP Request.jmx (E:\Softwars\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin\HTTP Request.jmx) - Apache JMeter (5.6.3)
File Edit Search Run Options Tools Help
00:00:01 0 0/10
Test Plan
Thread Group
HTTP Request
View Results Tree
Summary Report
Aggregate Report
Basic Advanced
Parameters Body Data Files Upload
Detail Add Add from Clipboard Delete Up Down

HTTP Request

Name: HTTP Request

Comments:

Web Server

Protocol [http]: Server Name or IP: https://www.openai.com Port Number:

HTTP Request

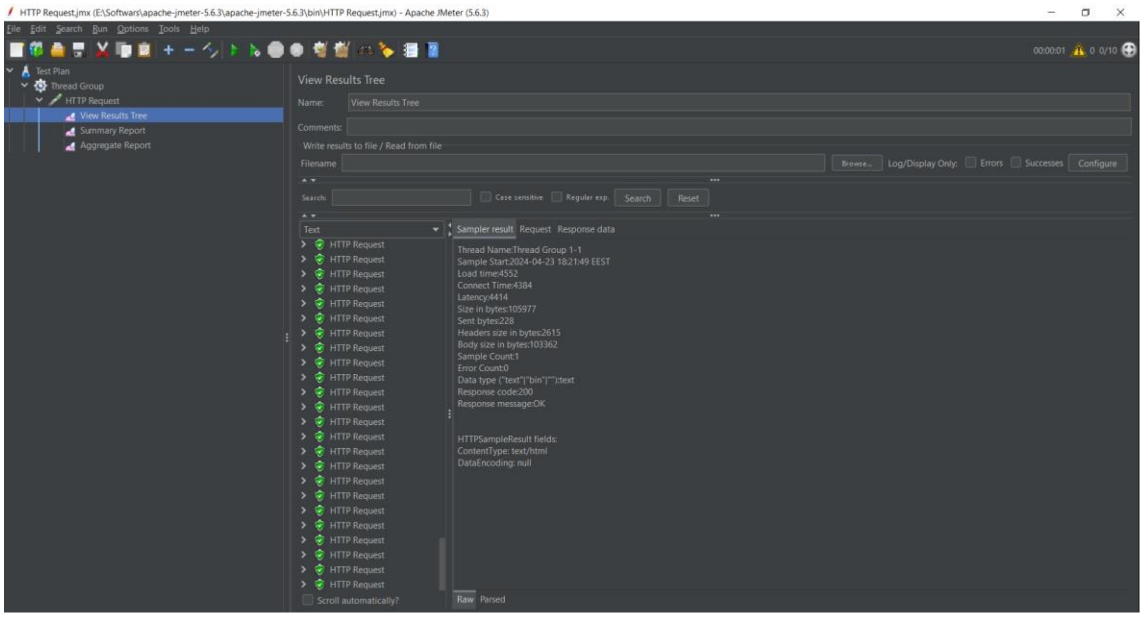
GET Path: https://www.openai.com/ Content encoding:

☐ Redirect Automatically ☑ Follow Redirects ☑ Use KeepAlive ☐ Use multipart/form-data ☐ Browser-compatible headers

Send Parameters With the Request:

Name:	Value	URL Encode?	Content-Type	Include Equals?

Figure 19. Sampler

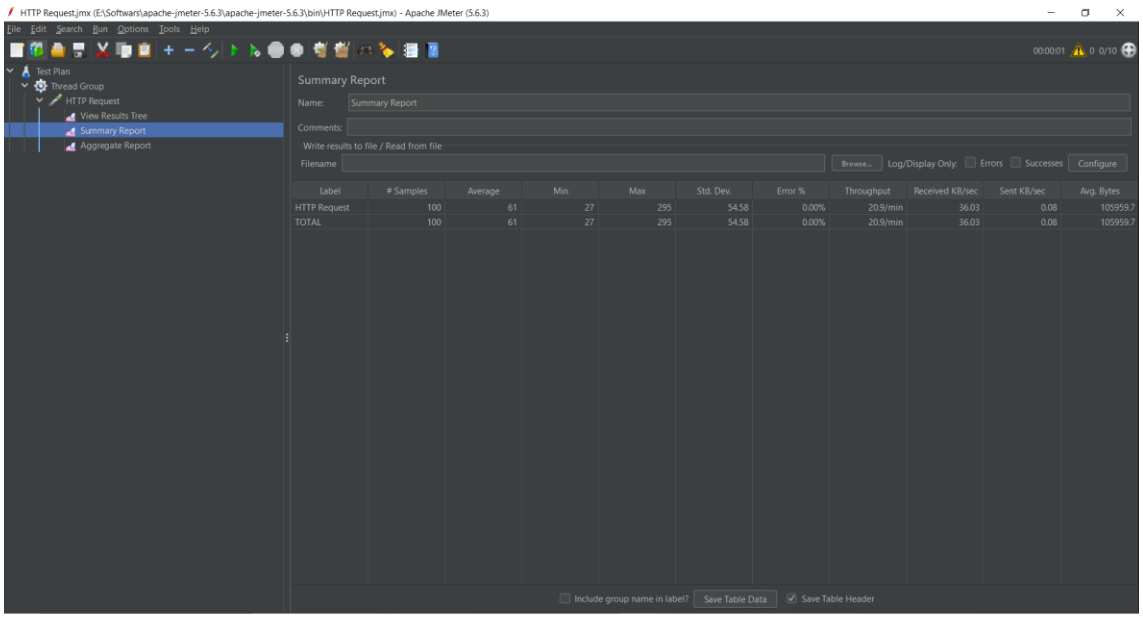

HTTP Request.jmx (E:\Softwars\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin\HTTP Request.jmx) - Apache JMeter (5.6.3)
File Edit Search Run Options Tools Help
Test Plan
Thread Group
HTTP Request
View Results Tree
Summary Report
Aggregate Report
Sampler result
Request
Response data
Raw
Parsed
Scroll automatically?

View Results Tree

Name: View Results Tree

```
Thread Name:Thread Group 1-1
Sample Start:2024-04-23 18:21:49 EEST
Load time:4552
Connect Time:4384
Latency:4414
Size in bytes:105977
Sent bytes:228
Headers size in bytes:2615
Body size in bytes:103362
Sample Count:1
Error Count:0
Data type ("text"|"bin"|""):text
Response code:200
Response message:OK


HTTPSampleResult fields:
ContentType: text/html
DataEncoding: null
```

Figure 20. View result tree listener

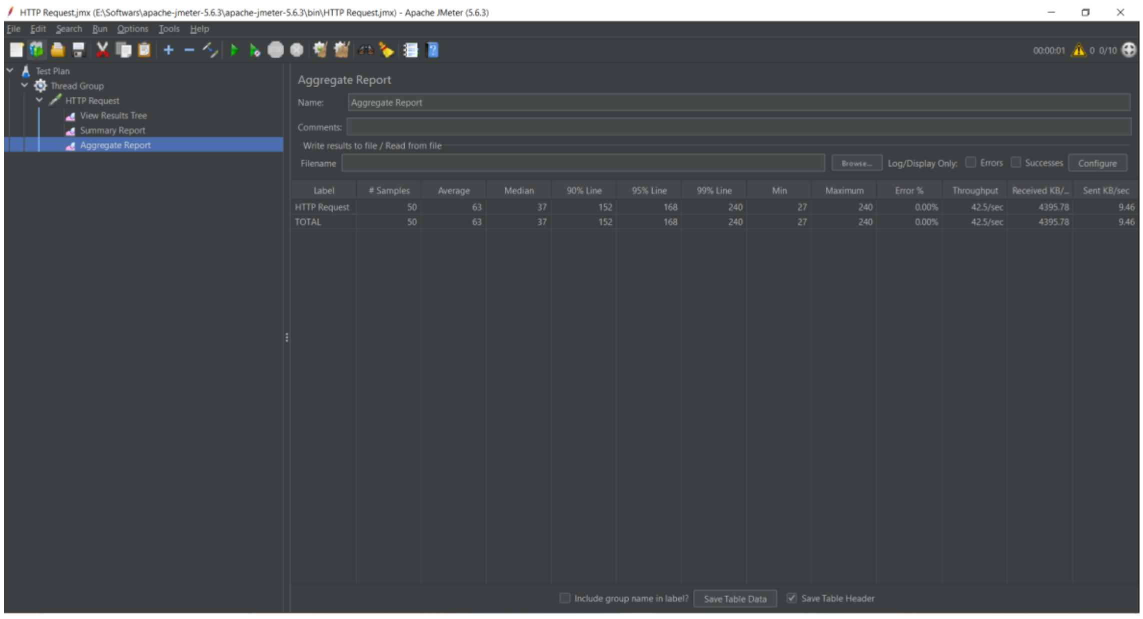

HTTP Request.jmx (E:\Softwars\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin\HTTP Request.jmx) - Apache JMeter (5.6.3)
File Edit Search Run Options Tools Help
Test Plan
Thread Group
HTTP Request
View Results Tree
Summary Report
Aggregate Report
Include group name in label?
Save Table Data
Save Table Header

Summary Report

Name: Summary Report

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	100	61	27	295	54.58	0.00%	20.9/min	36.03	0.08	105959.7
TOTAL	100	61	27	295	54.58	0.00%	20.9/min	36.03	0.08	105959.7

Figure 21. Summary report listener


HTTP Request.jmx (E:\Softwars\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin\HTTP Request.jmx) - Apache JMeter (5.6.3)
File Edit Search Run Options Tools Help
00:00:01 0 0/10
Test Plan
Thread Group
HTTP Request
View Results Tree
Summary Report
Aggregate Report
Browse...
Configure
Save Table Data

Aggregate Report

Name: Aggregate Report

Comments:

Write results to file / Read from file

Filename

Log/Display Only: ☐ Errors ☐ Successes

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/...	Sent KB/sec
HTTP Request	50	63	37	152	168	240	27	240	0.00%	42.5/sec	4395.78	9.46
TOTAL	50	63	37	152	168	240	27	240	0.00%	42.5/sec	4395.78	9.46

☐ Include group name in label? ✓ Save Table Header

Figure 22. Aggregate report listener

Parameterization, assertions, and timers are essential aspects of performance testing in Apache JMeter, allowing one to make one’s tests more flexible, validate responses, and introduce realistic delays between requests. To parameterize one’s test, one can define variables like ${username} and ${password} to simulate different user inputs. This is done by navigating to "Add" -> "Config Element" -> "User Defined Variables" and specifying the variables one want to use. Assertions play a crucial role in validating the responses received from the server. By right-clicking on the sampler, one can add assertions to check if the response meets certain criteria, ensuring the correctness of one’s application's behavior under load. Additionally, timers help in introducing delays between requests, mimicking real-world scenarios where users don't send requests instantaneously. One can add a timer by right-clicking on the sampler and choosing "Add" -> "Timer". These features allow one to create more realistic and accurate performance tests, helping one identify and address any issues in one’s application's performance.

4.3 Postman

Postman is a tool to check if computer programs (APIs) work right. It helps Quality Assurance (QA) teams by sending messages to a computer and watching what comes back. It used to be a small part of a web browser, but now it's its own program one can use on a computer or online.

QA engineers use Postman to make sure computer programs (APIs) work OK. They do different tests like asking for information (GET), sending information (POST), changing information (PUT), or removing it (DELETE). This helps make sure the programs behave right in different situations. They also make special tests called "collections." These collections organize lots of tests neatly. They can run these tests automatically to watch how well the program works all the time. If they want to see how the program deals with lots of people using it at once, they can do that too with Postman. They can also make different situations, like pretending the program is in a "development" or "production" place, to check if it works everywhere. Postman also helps make notes about the program so others can understand it easily. Plus, they can work together with others on the tests and share everything easily. This makes it easier to talk about what they find and work as a team.

People like using Postman for QA because it's easy to use. It has simple menus and buttons, so anyone can use it, even if they're not great with computers. It lets one do lots of different things, like testing programs, making tests run alone, watching how programs work, and working with others on tests. One can use it on a computer, online, or even by typing commands, so it works with any computer. Lots of people use it and help each other out, so if one get stuck, one can usually find someone who knows how to fix it. And if one're using other tools to check programs, Postman can work with them too, so one don't have to switch back and forth between different programs all the time.
In Postman, a collection is like a folder where one keep all one's related API requests together. Collections have special features like Authorization, Pre-request Scripts, Tests, Variables, and Run.

Authorization in a collection means one can set up how one want to log in or authenticate once, and it'll apply to all the requests in that collection. So, one don't have to set it up for each request separately.

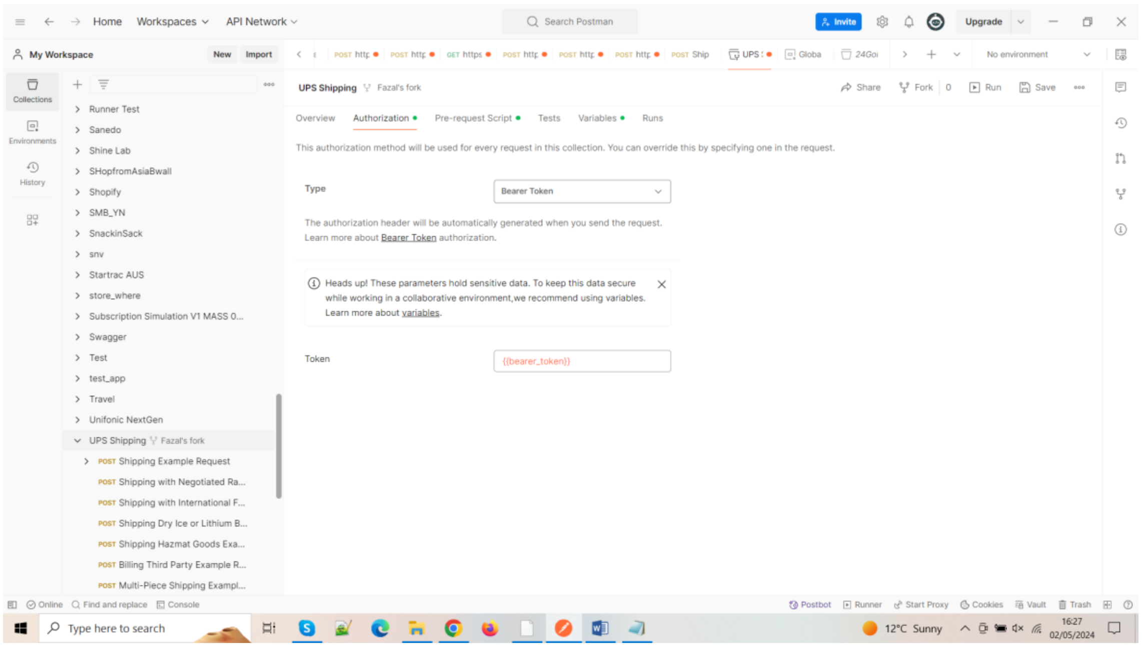
Pre-request Scripts are like little tasks one can do before sending a request. For example, one can change some data or set things up before making the request.

Tests in a collection are checks one can set up to make sure the API responses are correct. One can say things like, "Check if the status code is 200" or "Make sure the response has certain data."

Variables let one store information that one can reuse in different requests. It's like having placeholders for data that one can use wherever one need them.

The **Run** feature lets one run all the requests in one's collection one after another. This is handy because one can test everything in one's collection at once, saving time.

Collections are a simple way to organize and test one's APIs in Postman, making it easier to manage and run one's tests.



This authorization method will be used for every request in this collection. You can override this by specifying one in the request.

Type Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment,we recommend using variables. Learn more about variables.

Token {{bearer_token}}

Figure 23. Collection example for UPS Shipping

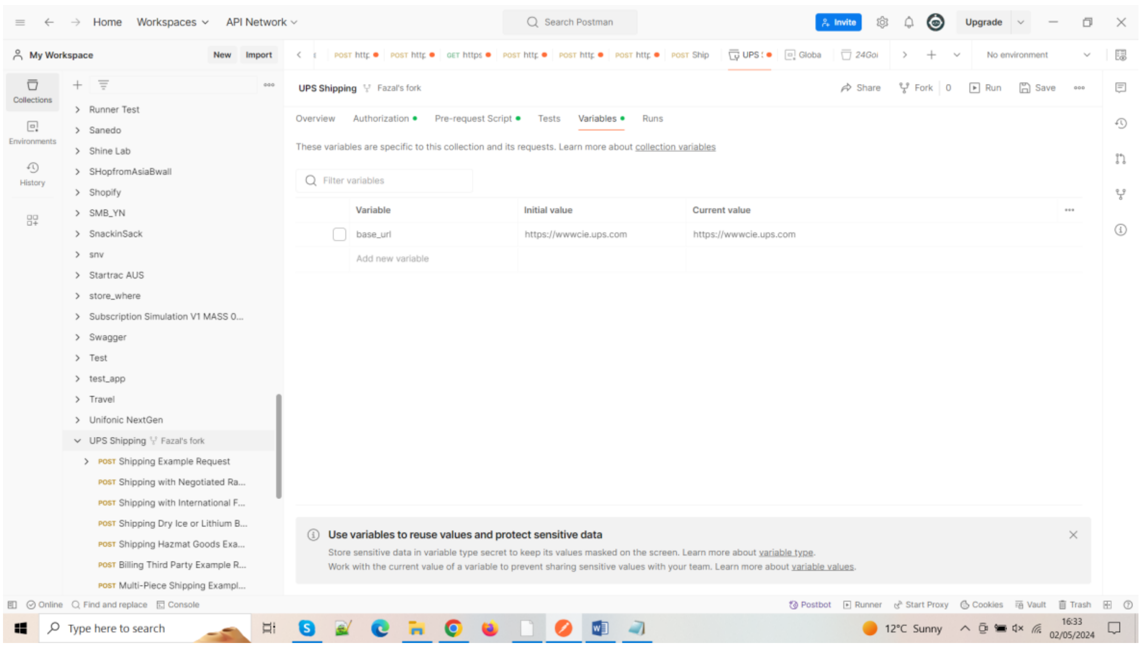
Variables in Postman are like placeholders for values that one use in one's requests. For example, one might have a variable called **{{base_url}}** for the main URL of one's API, or **{{token}}** for an authentication token.

These variables can be set up in two main places: in a collection or in an environment.

Collection Variables: These are specific to a collection. One can set them up once, and they'll be used in all the requests within that collection.

Environment Variables: These are more flexible. One can set them up in an environment, and they'll apply to all requests in that environment. This means one can have different values for different environments like development, testing, or production.

By using variables, one can avoid repeating the same values over and over again in one's requests. It makes one's requests cleaner and easier to manage.


UPS Shipping · Fazal's fork

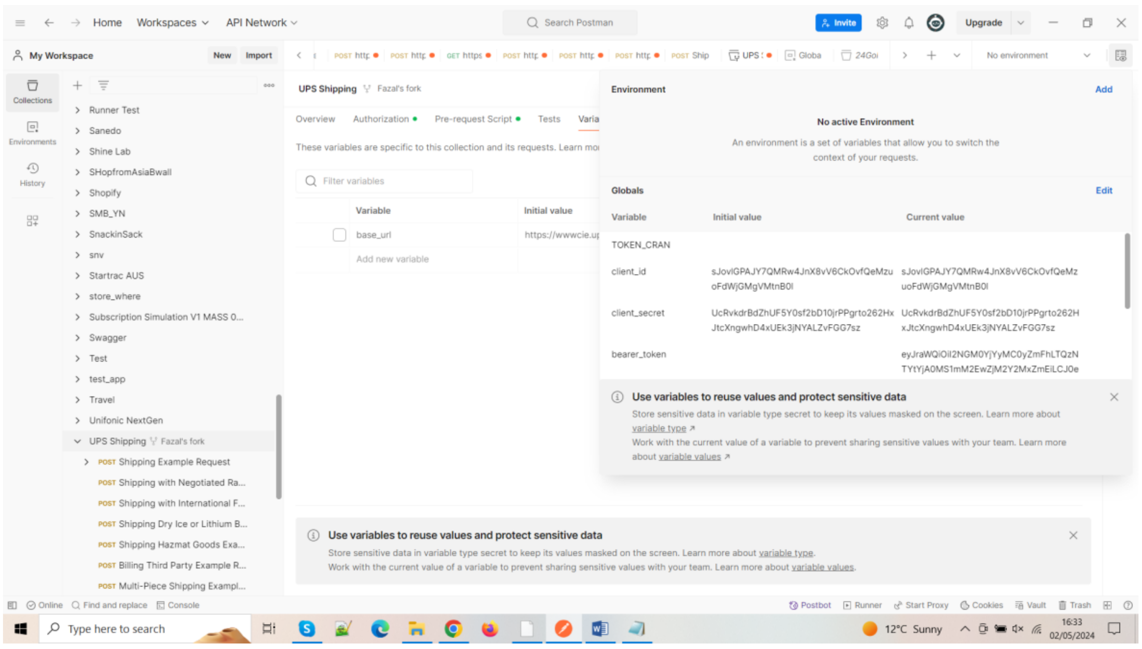
Overview · Authorization · Pre-request Script · Tests · Variables · Runs

These variables are specific to this collection and its requests. Learn more about collection variables

Variable	Initial value	Current value
base_url	https://wwwcie.ups.com	https://wwwcie.ups.com
Add new variable		

Use variables to reuse values and protect sensitive data
Store sensitive data in variable type secret to keep its values masked on the screen. Learn more about variable type.
Work with the current value of a variable to prevent sharing sensitive values with your team. Learn more about variable values.

Figure 24. Collection variable



These variables are specific to this collection and its requests.

Variable	Initial value
base_url	https://wwwcie.u

Environment

No active Environment

An environment is a set of variables that allow you to switch the context of your requests.

Globals

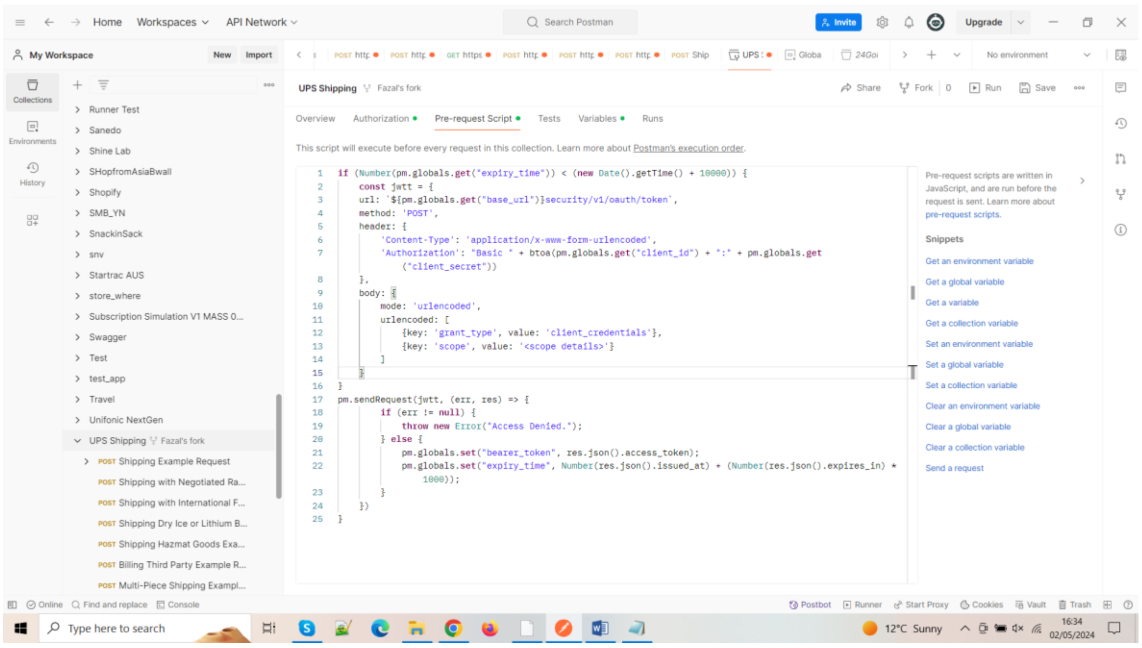
Variable	Initial value	Current value
TOKEN_CRAN		
client_id	sJovlGPAJY7QMRw4JnX8vV6CkOvfQeMzuoFdWjGMgVMtnB0l	sJovlGPAJY7QMRw4JnX8vV6CkOvfQeMzuoFdWjGMgVMtnB0l
client_secret	UcRvkdrBdZhUF5Y0sf2bD10jrPPgrto262HxJtcXngwhD4xUEk3jNYALZvFGG7sz	UcRvkdrBdZhUF5Y0sf2bD10jrPPgrto262HxJtcXngwhD4xUEk3jNYALZvFGG7sz
bearer_token		eyJraWQiOiI2NGM0YjYyMC0yZmFhLTQzNTYtYjA0MS1mM2EwZjM2Y2MxZmEiLCJ0e

Use variables to reuse values and protect sensitive data

Store sensitive data in variable type secret to keep its values masked on the screen. Learn more about variable type

Work with the current value of a variable to prevent sharing sensitive values with your team. Learn more about variable values

Figure 25. Environment variable



This script will execute before every request in this collection. Learn more about Postman's execution order.

```javascript
if (Number(pm.globals.get("expiry_time")) < (new Date().getTime() + 10000)) {
    const jwtt = {
    url: `${pm.globals.get("base_url")}security/v1/oauth/token`,
    method: 'POST',
    header: {
        'Content-Type': 'application/x-www-form-urlencoded',
        'Authorization': "Basic " + btoa(pm.globals.get("client_id") + ":" + pm.globals.get
            ("client_secret"))
    },
    body: {
        mode: 'urlencoded',
        urlencoded: [
            {key: 'grant_type', value: 'client_credentials'},
            {key: 'scope', value: '<scope details>'}
        ]
    }
}
pm.sendRequest(jwtt, (err, res) => {
        if (err != null) {
            throw new Error("Access Denied.");
        } else {
            pm.globals.set("bearer_token", res.json().access_token);
            pm.globals.set("expiry_time", Number(res.json().issued_at) + (Number(res.json().expires_in) *
                1000));
        }
    })
}
```

Pre-request scripts are written in JavaScript, and are run before the request is sent. Learn more about pre-request scripts.

Figure 26. Pre-request Script

This pre-request script from figure 26 checks if a token is about to expire or has expired. If it is, it automatically renews the token by sending a request to a token endpoint with the client credentials. After receiving a response, if successful, it updates the stored bearer token and expiry time. This ensures that there's always a valid token available for making requests, preventing interruptions due to expired tokens.

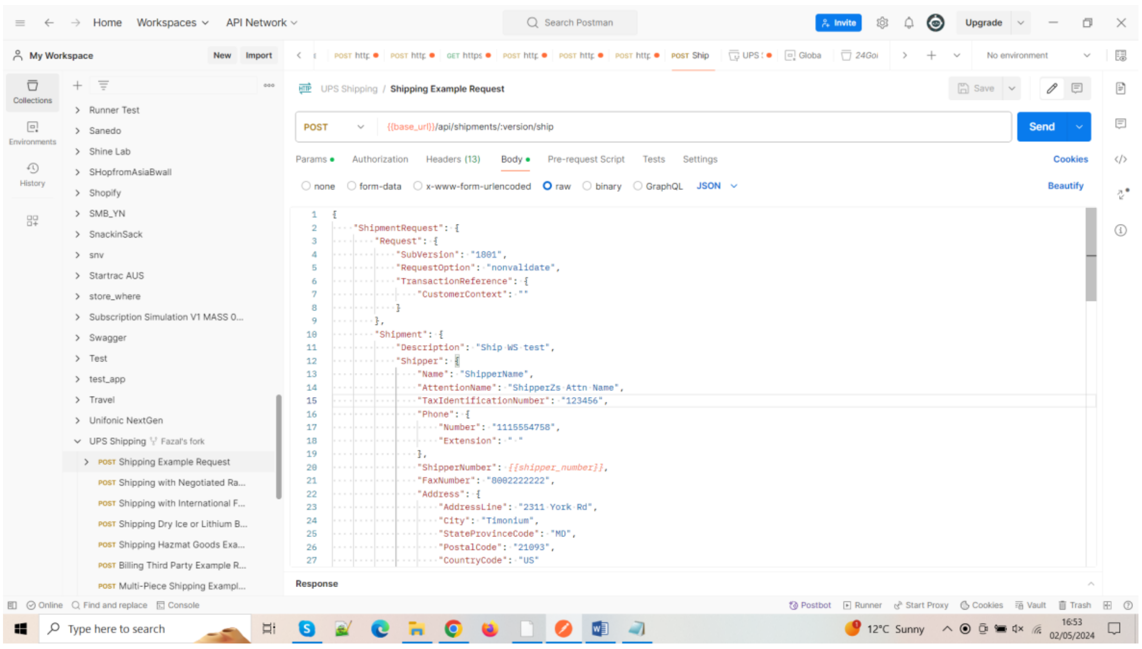
To add a new request to a collection in Postman, one have two options. One can either click the "New" button at the top right corner of the Postman sidebar, or one can right-click on the collection and select "Add request" from the menu. Once one've selected to add a request, one can choose the type of request one want to make, like POST, GET, DELETE, or PATCH.

In the example of creating a shipment, we use the endpoint {{base_url}}/api/shipments/:version/ship. Here, we've used a variable called base_url, which could be defined in either the collection's variables or in an environment.

When creating a request, one have several features to configure:

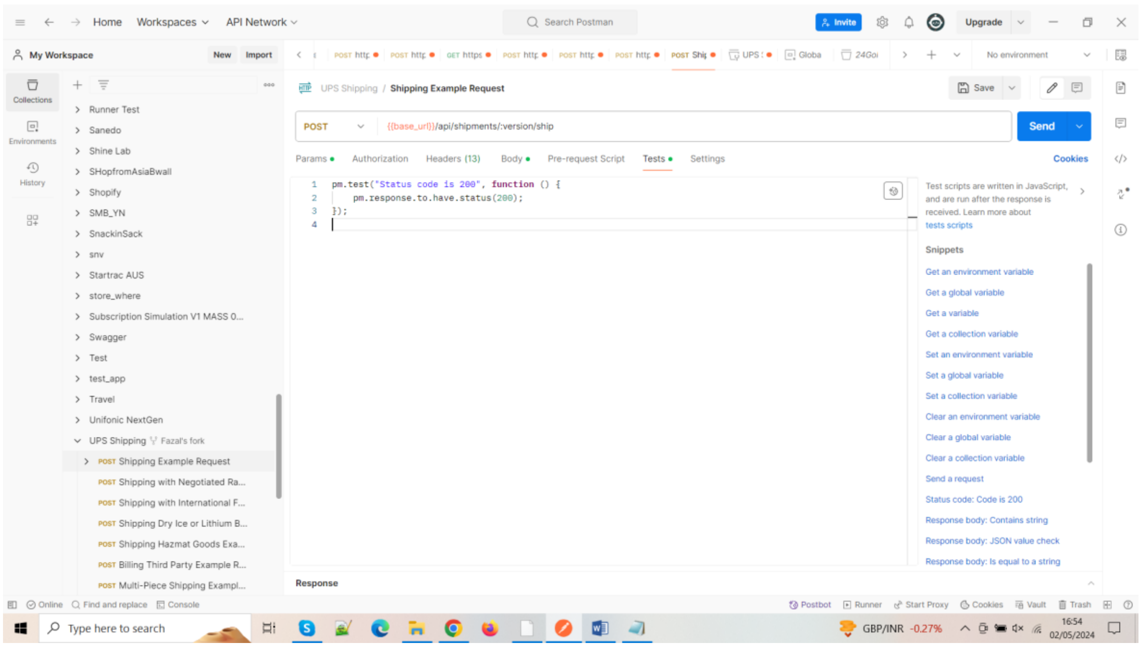
- **Params**: If one need to add parameters in the URL, one can specify them here, and they'll be added to the URL.
- **Authorization**: By default, the request will use the authorization settings from the collection. But if one need to change it for this specific request, one can do that here.
- **Headers**: One can pass headers here, like specifying the content type as JSON.
- **Body**: This is where one specify the body of the request. It could be in different formats like JSON, form-data, or raw text.
- **Pre-request**: Here, one can add any additional tasks one want to perform before sending the request.
- **Tests**: In the tests section, one can define what kind of data one expect in the API response. For example, one might expect JSON or an array.

These features help one customize one’s request according to one’s API's requirements and expectations for testing and integration purposes.


POST {{base_url}}/api/shipments/:version/ship

```
{
    "ShipmentRequest": {
        "Request": {
            "SubVersion": "1801",
            "RequestOption": "nonvalidate",
            "TransactionReference": {
                "CustomerContext": ""
            }
        },
        "Shipment": {
            "Description": "Ship WS test",
            "Shipper": {
                "Name": "ShipperName",
                "AttentionName": "ShipperZs Attn Name",
                "TaxIdentificationNumber": "123456",
                "Phone": {
                    "Number": "1115554758",
                    "Extension": " "
                },
                "ShipperNumber": {{shipper_number}},
                "FaxNumber": "8002222222",
                "Address": {
                    "AddressLine": "2311 York Rd",
                    "City": "Timonium",
                    "StateProvinceCode": "MD",
                    "PostalCode": "21093",
                    "CountryCode": "US"
```

Figure 27. JSON body passed to create shipment


POST {{base_url}}/api/shipments/:version/ship

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

Figure 28. Test to expect status 200

4.4 Robot framework

Robot Framework is a powerful open-source test automation framework used in software quality assurance (QA) to automate testing processes. It's designed to be easy to use, even for those without extensive programming knowledge. The framework follows a keyword-driven approach, where tests are written using simple keywords and natural language, making them easy to read and understand.

In the real world, Robot Framework is used in various industries and applications to automate testing tasks across different types of software projects. For example, in web development, Robot Framework can be used to automate UI testing, where it interacts with web pages, fills out forms, clicks buttons, and verifies expected behavior. In API testing, Robot Framework can send requests to APIs, validate responses, and ensure that APIs are working correctly. It's also used in acceptance testing, where it simulates user interactions to validate that the software meets specified requirements.

For QA professionals, Robot Framework is invaluable because it saves time and improves efficiency in testing processes. By automating repetitive testing tasks, QA teams can focus their efforts on more complex and critical aspects of software testing, such as exploratory testing and edge-case scenarios. Robot Framework also provides detailed test reports and logs, helping QA professionals identify issues quickly and accurately.

To install Robot Framework on one’s machine, one first need to have Python installed. Python comes with a package manager called pip, which we use to install Robot Framework. Once Python is installed, one can use the pip command to install Robot Framework. It goes like this:

```
pip install robotframework
```

When working with Robot Framework, one’s tests are organized into collections of files called test suites. Each test suite can contain multiple test files, and each file can have several test cases. In Robot Framework, we create test files with a

".robot" extension. Typically, one will find all one's test files stored in a folder called "Tests" within one's project directory."

A ".robot" file in Robot Framework follows a structured format. It typically includes:

Settings: This section is used to configure settings for the test suite or individual test cases. Settings can include things like specifying the library to use, configuring timeouts, or defining test setup and teardown procedures.

Variables: In this section, one can define variables that can be used throughout the test suite or specific test cases. These variables can hold values like URLs, usernames, passwords, or any other data that needs to be reused across tests.

Keywords: Keywords are reusable blocks of functionality that perform specific actions or verifications. They are defined in this section and can be called within test cases to perform actions like clicking buttons, verifying text, or making API requests.

Test Cases: This is where one define the actual test cases that one want to execute. Each test case consists of a sequence of steps that use keywords to interact with the system under test and verify its behavior. Test cases can include setup and teardown steps as well as assertions to verify expected outcomes.

In Robot Framework, test setup and teardown are special sections within a test case that define actions to be performed before and after the execution of each test case, respectively.

Test Setup: The test setup section contains steps that are executed before the main body of the test case. These steps are typically used to prepare the test environment or to perform any necessary preconditions for the test case to run successfully. For example, one might use the test setup section to open a browser, log in to a web application, or set up test data.

Test Teardown: The test teardown section contains steps that are executed after the main body of the test case has been executed. These steps are used to clean up the test environment or to perform any necessary postconditions after the test case has run. For example, one might use the test teardown section to close the browser, log out of a web application, or clean up any test data that was created during the test.

locators are like addresses for elements on a web page. Just like one need an address to find a house, one need a locator to find elements like buttons, input fields, or links on a web page. Locators help automation tools, like Robot Framework, to identify and interact with these elements during testing.

There are different types of locators, each with its own way of finding elements.

ID: Each element can have a unique ID, like a house number. ID locators directly target specific elements.

Class Name: Similar elements, like houses in the same neighborhood, can share a class name. Class name locators can target multiple elements with the same class.

Name: Some elements have names, like people. Name locators can find elements based on their names.

XPath: XPath locators use paths to navigate through the HTML structure of a web page, like following a map to find a location.

CSS Selector: CSS selectors are patterns used to select elements based on their attributes or relationships with other elements.

Link Text: For links, the visible text is used as a locator.

Partial Link Text: If one don't know the full text of a link, one can use part of it to locate the element.

Tag Name: Tag name locators target elements based on their HTML tag names, like finding all houses with the same building style.

HTML Element: <input id="username" type="text">

Robot Framework Locator:

```
${id_locator}= id:username
```

HTML Element: <button class="submit-button">Submit</button>

Robot Framework Locator:

```
${class_locator}= class:submit-button
```

HTML Element: <input name="email" type="text">

Robot Framework Locator:

```
${name_locator}= name:email
```

HTML Element: <input type="text" id="username">

Robot Framework Locator:

```
${xpath_locator}= xpath://input[@id='username']
```

HTML Element: <input id="username" type="text">

Robot Framework Locator:

```
${css_locator}= css:input#username
```

HTML Element: <a href="/login">Login</a>

Robot Framework Locator:

```
${link_locator}= link:Login
```

HTML Element: <a href="/login">Login</a>

Robot Framework Locator:

```
${partial_link_locator}= partial link:Log
```

HTML Element: <input type="text">

Robot Framework Locator:

```
 ${tag_locator}= tag:input
```

Following is an example where we automate a registration process on a web page with various types of elements such as text fields, radio buttons, checkboxes, dropdowns, and button

HTML Structture

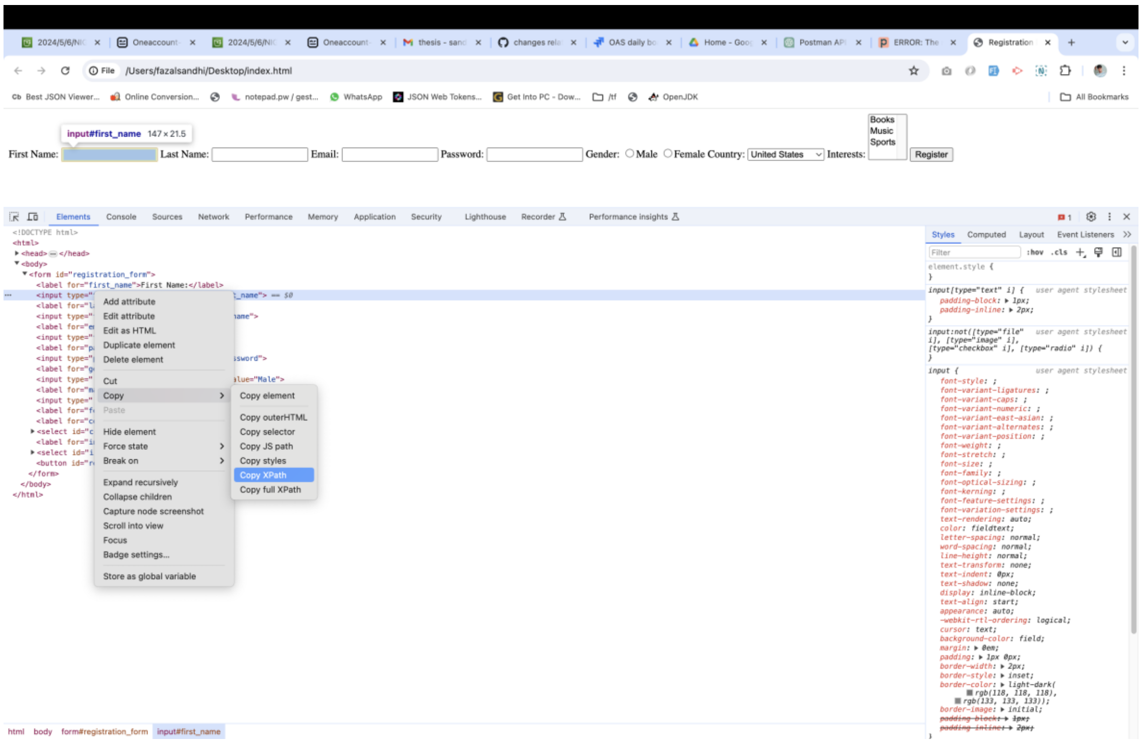
```
<!DOCTYPE html>
<html>
<head>
<title>Registration Page</title>
</head>
<body>
<form id="registration_form">
<label for="first_name">First Name:</label>
<input type="text" id="first_name" name="first_name">
<label for="last_name">Last Name:</label>
<input type="text" id="last_name" name="last_name">
<label for="email">Email:</label>
<input type="text" id="email" name="email">
<label for="password">Password:</label>
<input type="password" id="password" name="password">
<label for="gender">Gender:</label>
```

```
<input type="radio" id="male" name="gender" value="Male"><label
for="male">Male</label>
<input type="radio" id="female" name="gender" value="Female"><label
for="female">Female</label>
<label for="country">Country:</label>
<select id="country" name="country">
<option value="United States">United States</option>
<option value="Canada">Canada</option>
<option value="United Kingdom">United Kingdom</option>
<!-- Other country options -->
</select>
<label for="interests">Interests:</label>
<select id="interests" name="interests" multiple>
<option value="Books">Books</option>
<option value="Music">Music</option>
<option value="Sports">Sports</option>
<!-- Other interest options -->
</select>
<button id="register_button" type="submit">Register</button>
</form>
</body>
</html>
```

Registration.robot

```
*** Settings ***
Library SeleniumLibrary

*** Variables ***
${URL} https://www.example.com/register
${BROWSER} Chrome
${FIRST_NAME} John
${LAST_NAME} Doe
${EMAIL} john.doe@example.com
${PASSWORD} Password123
${GENDER} Male
${COUNTRY} United States
${INTERESTS} Books, Music
${ERROR_MESSAGE} Please fill out this field.

*** Test Cases ***
User Can Register Successfully
[Documentation] Test that a user can register successfully with valid information
Open Browser ${URL} ${BROWSER}
Input Text id:first_name ${FIRST_NAME}
Input Text id:last_name ${LAST_NAME}
Input Text id:email ${EMAIL}
Input Password id:password ${PASSWORD}
Select Radio Button gender male
Select Country ${COUNTRY}
Select Interests ${INTERESTS}
Click Button id:register_button
Page Should Contain Welcome, ${FIRST_NAME}!

User Cannot Register Without Required Fields
[Documentation] Test that user cannot register without filling out required
fields
Open Browser ${URL} ${BROWSER}
Click Button id:register_button
Page Should Contain ${ERROR_MESSAGE}

*** Keywords ***
Input Password
[Arguments] ${locator} ${password}
Input Text ${locator} ${password}
Press Keys ${locator} \\13 # \\13 is the ENTER key


Select Country
[Arguments] ${country}
Select From List by Label id:country ${country}

Select Interests
[Arguments] ${interests}
${interest_list}= Evaluate "${interests}".split(",")
Unselect All from List id:interests
FOR ${interest} IN @{interest_list}
Select From List by Value id:interests ${interest}
END
```

In this example, we're automating the registration process on a website using Robot Framework and SeleniumLibrary. We start by setting up variables for the URL of the registration page, the browser to use, and the user's input such as their name, email, and password. Then, we create two test cases: one to test successful registration and another to test error handling if required fields are not filled. In the successful registration test case, we input valid information, select options from dropdowns and radio buttons, and verify the welcome message after clicking the register button. In the error handling test case, we attempt to register without filling required fields, click the register button, and verify the displayed error message. We also define custom keywords to securely input passwords, select gender, country, and interests, and verify text on the page. This example shows how to automate a registration process with various UI elements using Robot Framework and SeleniumLibrary.



```
<!DOCTYPE html>
<html>
▶<head> ⋯ </head>
▼<body>
  ▼<form id="registration_form">
      <label for="first_name">First Name:</label>
      <input type="                              _name"> == $0
      <label for="l
      <input type="                              name">
      <label for="e
      <input type="
      <label for="p
      <input type="                              ssword">
      <label for="g
      <input type="                              alue="Male">
      <label for="m
      <input type="
      <label for="f
      <label for="c
    ▶<select id="c
      <label for="i
    ▶<select id="i
      <button id="r
    </form>
  </body>
</html>
```

```
element.style {
}
input[type="text" i] {        user agent stylesheet
    padding-block: ▶ 1px;
    padding-inline: ▶ 2px;
}
input:not([type="file"        user agent stylesheet
i], [type="image" i],
[type="checkbox" i], [type="radio" i]) {
}
input {                       user agent stylesheet
    font-style: ;
    font-variant-ligatures: ;
    font-variant-caps: ;
    font-variant-numeric: ;
    font-variant-east-asian: ;
    font-variant-alternates: ;
    font-variant-position: ;
    font-weight: ;
    font-stretch: ;
    font-size: ;
    font-family: ;
    font-optical-sizing: ;
    font-kerning: ;
    font-feature-settings: ;
    font-variation-settings: ;
    text-rendering: auto;
    color: fieldtext;
    letter-spacing: normal;
    word-spacing: normal;
    line-height: normal;
    text-transform: none;
    text-indent: 0px;
    text-shadow: none;
    display: inline-block;
    text-align: start;
    appearance: auto;
    -webkit-rtl-ordering: logical;
    cursor: text;
    background-color: field;
    margin: ▶ 0em;
    padding: ▶ 1px 0px;
    border-width: ▶ 2px;
    border-style: ▶ inset;
    border-color: ▶ light-dark(
          rgb(118, 118, 118),
        rgb(133, 133, 133));
    border-image: ▶ initial;
    ~~padding-block: ▶ 1px;~~
    ~~padding-inline: ▶ 2px;~~
}
```

Figure 29. Developer tool to get the xpath of locaters

Here are some basic built-in keywords in Robot Framework that are important to remember:

1. **Open Browser:** Opens a new browser instance. Example: Open Browser https://www.example.com Chrome
2. **Input Text:** Types text into a text field. Example: Input Text id:username my_username
3. **Click Button:** Clicks on a button element. Example: Click Button id:login_button
4. **Select From List by Value:** Selects an option from a dropdown list by its value attribute. Example: Select From List by Value id:country US
5. **Wait Until Page Contains:** Waits until a specified text appears on the page. Example: Wait Until Page Contains Welcome, User
6. **Run Keyword If:** Executes a keyword conditionally based on a condition. Example: Run Keyword If '${status}' == 'Active' Click Button id:activate_button

7. **Should Be Equal As Strings:** Verifies if two strings are equal. Example: Should Be Equal As Strings ${actual} ${expected}
8. **Log:** Logs a message to the console. Example: Log This is a message for debugging
9. **Sleep:** Pauses test execution for a specified amount of time. Example: Sleep 5 seconds
10. **Run Keywords:** Executes multiple keywords sequentially. Example: Run Keywords Click Button id:login_button AND Input Text id:username my_username

5 Conclusion

In summary, this thesis looked at different parts of software testing, making sure computer programs work correctly. We started by learning the basics of testing and found out different ways to plan and manage tests, along with the tools that help with testing.

Then, we checked out important web technologies like HTML, CSS, and JavaScript. These are key for building websites, and we focused on understanding their structure to test and automate better. Knowing these technologies helps us improve websites and make testing easier.

It's worth noting that this thesis doesn't give a step-by-step guide on these technologies and how they work. But it could be helpful to have one for a deeper understanding. Also, we could have explored more advanced topics in quality assurance, like GitHub actions and other tools that help with continuous integration and continuous deployment (CI/CD).

Lastly, we talked about specific tools that make testing software easier, such as Jira and JMeter. These tools speed up the testing process and help us make better software.

In conclusion, we've learned that good testing is important for making sure software works as it should. By using the right tools and techniques, we can create software that people enjoy using. Let's keep learning and getting better at making software in the future!

References

1. Mozilla Developer Network (MDN). HTML: HyperText Markup Language. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML
2. W3Schools. HTML Tutorial. [Online]. Available: https://www.w3schools.com/html/
3. Mozilla Developer Network (MDN). CSS Documentation. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/CSS
4. W3Schools. CSS Tutorial. [Online]. Available: https://www.w3schools.com/css/
5. CSS-Tricks - Tips, Tricks, and Techniques. [Online]. Available: https://css-tricks.com/
6. Mozilla Developer Network (MDN). JavaScript Documentation. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript
7. W3Schools. JavaScript Tutorial. [Online]. Available: https://www.w3schools.com/js/
8. JavaScript.info - Modern JavaScript Tutorials. [Online]. Available: https://javascript.info/
9. Apache JMeter. [Online]. Available: https://jmeter.apache.org/
10. Jira. [Online]. Available: https://www.atlassian.com/software/jira/service-management
11. Postman. [Online]. Available: https://www.postman.com/
12. JMeter. [Online]. Available: https://jmeter.apache.org/
13. Robot framework. [Online]. Available: https://robotframework.org/