ICT Upskilling Program

Applied Software Testing and Quality Assurance Techniques

Prepared by:
Mai Taha

Supervisor:
Asraf Al-Smadi
February 2026

Table Of Contents

0. Project Overview

0.1 Supervisor

Supervisor: **Ashraf Al-Smadi**

0.2 Project Title

End-to-End Software Testing for Web, Mobile, and API Applications Using Manual, Automation, and Performance Testing.

0.3 Goals and Objectives

The main goal of this project is to apply software testing concepts on real web, mobile, and backend applications using manual testing, UI automation, API testing, and performance testing.

The objectives of this project are

- Understand system workflows and business domains
- Design and execute manual test cases
- Automate critical user journeys using Selenium and TestNG
- Perform API functional testing using Postman
- Execute API performance testing using k6
- Analyze test results and identify potential quality risks
- Document findings and provide improvement recommendations

0.4 Project Description

This project intends to implement software testing concepts on multiple systems in order to practice and demonstrate different testing methodologies and tools.

The SauceDemo web application was used as an example e-commerce frontend to perform manual testing and UI automation testing using Selenium and TestNG. The purpose of this phase was to test the user interface feature, business logic, and end-to-end user flows.

DummyJSON was used as a different example mock backend system to perform API functional testing using Postman and API performance testing using k6. The DummyJSON APIs were selected to practice independent REST API validation, automation, and performance testing unrelated to the web application.

Testing was performed on each system separately based on their functionality, and this project does not demonstrate an integrated application. Instead, this project demonstrates a full QA cycle by applying manual, automated, API, and performance testing methodologies on different applications.

0.5 Project Requirements (Hardware & Software)

Hardware Requirements

- Personal computer
- Minimum 8GB RAM recommended
- Stable internet connection

Software Requirements

Operating System

Linux

Web Browsers

Mozilla Firefox

Development Environment

Eclipse IDE

Programming & Runtime

Java JDK

Node.js with npm

Web UI Automation

Selenium WebDriver

TestNG

API & Performance Testing

Postman

Newman

k6

Version Control

Git

GitHub

0.6 Applications Under Test

The following applications were used as systems under test:

- SauceDemo Web Application (UI testing and automation)
- DummyJSON REST APIs (API functional and performance testing)

0.7 Expected Outcomes

By the end of this project, the following outcomes are expected:

- A complete manual test plan and test cases for the web application
- Automated UI test scripts
- API functional testing with automated assertions
- API performance testing with smoke and load profiles
- Execution reports and performance metrics
- A structured QA report documenting all testing activities

1. SauceDemo Web Application Testing

1.0 Executive Summary

This project demonstrates the practical application of software quality assurance and testing techniques across multiple independent systems. The project covers manual testing, UI automation, API testing, and API performance testing using industry-standard tools and methodologies.

Manual testing and UI automation were applied to the SauceDemo web application to validate core e-commerce functionality, identify defects, and ensure stability of critical user flows. Automated UI tests were implemented using Selenium WebDriver and TestNG to support regression testing.

API functional testing was conducted on the DummyJSON platform using Postman and Newman to validate REST API behavior, assertions, and error handling. Additionally, API performance testing was performed using k6 to evaluate system responsiveness, throughput, and stability under smoke and load conditions.

The project emphasizes risk-based testing, measurable quality metrics, and performance analysis. Overall, it provides a comprehensive QA workflow aligned with professional software testing practices.

1.1 Introduction

The SauceDemo web application was selected as the primary system under test for UI validation. It simulates a real e-commerce platform that allows users to log in, browse products, manage a shopping cart, and complete the checkout process.

This section focuses on validating core user workflows through manual testing and UI automation to ensure functional correctness and usability

1.2 Objectives

The objectives of SauceDemo testing are

- Validate login and authentication behavior
- Verify product listing and product details pages
- Test shopping cart operations such as adding and removing items
- Ensure checkout flow completes successfully
- Identify functional and usability defects
- Automate critical user journeys using Selenium and TestNG

1.3 Domain Understanding

SauceDemo represents a simplified e-commerce domain. Users authenticate using provided credentials, browse available products, add selected items to the cart, and proceed through the checkout process to complete an order.

The main business flow includes:

- User login
- Product browsing
- Cart management
- Checkout process
- Order confirmation

Understanding this flow is essential to designing effective test scenarios and validating system behavior.

1.5 Manual Testing

Manual testing was conducted to validate the core functionality of the SauceDemo web application. Test scenarios were designed based on business workflows, covering both positive and negative user paths.

1.5.1 Test Plan

The manual test plan focused on verifying key e-commerce features, including login, product browsing, cart management, and checkout flow. Testing was performed in a demo environment using provided test accounts and sample products.

Functional test cases were executed to validate system behavior, while negative scenarios were used to ensure proper error handling.

Scenario

The system under test is a **web-based retail application** that allows users to browse products, manage a shopping cart, and complete the checkout process.

The current testing scope focuses on the following web features:

- User login and authentication
- Product browsing and product details pages
- Shopping cart functionality:
  - Add items to cart
  - Remove items from cart
  - Update product quantities
- Checkout flow:
  - Cart review
  - Checkout process
  - Order confirmation display

Key requirement

- Only logged-in users can proceed to checkout
- Users must be able to add, update, and remove products from the cart
- Cart totals must update correctly when quantities change
- Checkout flow must complete successfully without errors
- The confirmation page must display a valid order confirmation message

Scope & Objectives

scope

This test plan focuses on testing the **SauceDemo web application** to validate core e-commerce functionality, including user login, product browsing, shopping cart operations, and the checkout flow up to order confirmation. Testing is limited to the web user interface only.

Test Objectives

The objectives of this test plan are to:

- Validate core web-based e-commerce functionality
- Ensure correct behavior for authenticated and unauthenticated users
- Identify functional and usability defects
- Verify checkout completion and confirmation display
- Ensure overall stability of the web application

In-scope / Out-of-scope

in Scope

Web UI functional testing

Manual testing of:

Login

Product listing

Cart operations

Checkout flow

Basic UI automation for critical user flows

Defect reporting and documentation

Out of Scope

API testing

Backend integration testing

Performance and load testing at API level

Security and penetration testing

Real payment processing

Mobile application testing

Assumptions & Dependencies

Assumptions

The SauceDemo web application is accessible during the testing period.

Test user accounts and demo data are available and can be used safely.

Testing is performed in a demo environment using sample products and data.

The application behavior remains stable during the test execution period.

No real customer or payment data is used during testing.

Dependencies

Successful login depends on the availability of authentication services.

Checkout completion depends on integrated backend services provided by the demo application.

The checkout flow relies on a third-party payment gateway operating in sandbox mode.

Stable internet connectivity is required to execute all web-based test scenarios.

Test Approach (levels + types)

Test Levels

System Testing:
End-to-end testing of the web application to validate complete user flows from login to checkout confirmation.

Test Types

Functional Testing: To verify core e-commerce features.

Manual Testing: for detailed validation of user interactions and UI behavior.

Automated UI Testing: For critical user journeys such as login, add to cart, and checkout.

Regression Testing: To ensure existing functionality is not impacted by new changes.

Exploratory Testing: To identify unexpected behavior and usability issues.

Environment & Tools

Web Application: SauceDemo (https://www.saucedemo.com)

Browser: fireFox

Operating System: Linux

Test data: Demo users and sample products provided by the application

Defect Management Process

Defect Severity

Critical: Blocks core functionality such as login or checkout completion.

High: Major functionality works incorrectly with no workaround.

Medium: Functionality works with limitations or incorrect behavior.

Low: Minor UI or cosmetic issues with no functional impact.

Defect Priority

High: Must be fixed before release.

Medium: Fix recommended before release if time allows.

Low: The fix can be deferred to a later release.

Defect Workflow

Defects identified during test execution.

Defect logged with complete details and evidence.

Defect reviewed and prioritized.

Fix applied by the development team.

Defects retested by QA.

Defects are closed or reopened if not resolved.

Required Defect Fields

Defect ID

Summary

Description

Steps to Reproduce

Expected Result

Actual Result

Severity

Priority

Status

Attachments (screenshots)

Metrics & Reporting

Metrics

- Test case pass/fail rate
- Test coverage by feature
- Number of defects by severity
- Defect trends during execution
- Retest and closure rate

Reporting

- Daily test execution status updates
- Defect status updates during testing
- Final test summary report at the end of the test cycle

Schedule & Resources

Schedule

- Test planning and preparation: 2 days
- Manual test case design: 2 days
- Test execution and defect reporting: 2 days
- Retesting and closure: 1 day

Resources

- QA Engineer: Responsible for test planning, execution, and reporting.
- Tools: Web browser, test management documents, defect tracking sheets.

Effort & Buffer

Total estimated effort: 7 working days

Buffer: 1–2 days for rework and unexpected issues

Risks & Mitigation

Risk	Mitigation	Contingency
Test environment unavailable	Schedule testing during off-peak hours	Resume testing when system is available
Limited testing time	Focus on critical user flows	Defer low-priority tests
Unstable application behavior	Document issues clearly	Retest after stabilization

Risk-Based Prioritization

Testing activities were prioritized based on business impact and user risk. Core user flows that directly affect the ability to place an order were considered high-risk and were tested first and more extensively.

High-risk areas included:

User login and authentication

Shopping cart operations

Checkout process and order confirmation

Medium-risk areas included:

Product sorting and filtering

Navigation between pages

Low-risk areas included:

UI layout and cosmetic elements

This risk-based approach ensured that critical functionality was validated early and thoroughly.

Requirements Traceability

To ensure full test coverage, test cases were mapped to functional requirements of the SauceDemo application. This traceability ensures that all critical requirements are validated through manual and automated testing.

The following table illustrates sample traceability between requirements and test cases:

Requirement ID	Requirement Description	Test Case IDs
RQ-01	User authentication and login	TC00, TC01
RQ-02	Product browsing and sorting	TC02
RQ-03	Add and remove items from cart	TC03, TC05, TC06
RQ-04	Checkout process	TC07, TC08

This traceability helps ensure that all defined requirements are tested and supports impact analysis when changes occur.

Quality Metrics, KPIs, and Trade-offs

To measure testing effectiveness and overall quality, the following key performance indicators (KPIs) were defined:

- Test case execution rate (percentage of executed test cases)
- Test pass/fail ratio
- Number of defects by severity level
- Defect leakage (defects found after test execution)
- Retest and defect closure rate

Due to time constraints, testing efforts were focused on high-risk and high-impact areas such as login, cart operations, and checkout flow. Lower-risk UI cosmetic issues were deprioritized to ensure critical functionality was thoroughly validated.

Risk Mitigation Strategy

Potential risks were mitigated through the following actions:

- Prioritizing high-risk test scenarios early in the test cycle
- Combining manual testing with UI automation to reduce regression risk
- Documenting known limitations and defects clearly for future fixes

This approach ensured efficient use of time and resources while maintaining acceptable product quality.

Entry/Exit Criteria & Sign-off

Entry Criteria

- The application is deployed and accessible.
- Test cases are reviewed and approved.
- The test environment is ready.
- Test data is available.

Exit Criteria

- All planned test cases are executed.
- Critical and high-severity defects are documented.
- Test results and evidence are completed.
- The test summary report is finalized.

Sign-off

- QA approval confirms that testing activities are complete, and results are documented.
- The release decision is based on test results and defect status.

1.5.2 Test Scenarios

Happy Path Scenarios

The following scenarios represent the expected system behavior when valid data and actions are provided:

1. The user logs in successfully using valid credentials.
2. The user views the product listing page without errors.
3. The user opens a product details page.
4. The user adds a product to the shopping cart.
5. The user updates the quantity of a product in the cart, and the total is updated correctly.
6. The user removes a product from the cart.
7. Logged-in users proceed to checkout with items in the cart.
8. The user completes the checkout process successfully.
9. The order confirmation page is displayed after checkout completion.

Negative Scenarios

The following scenarios validate system behavior when invalid actions or conditions occur:

1. The user attempts to access the checkout page without logging in.
2. The user enters invalid login credentials.
3. The user attempts to check out with an empty cart.
4. The user enters an invalid product quantity (e.g., zero or negative).
5. The user refreshes or navigates away during the checkout process.
6. The user removes all items from the cart before checkout.
7. The system displays an appropriate message when a required checkout step is missing.
8. The user attempts to proceed with checkout while the application is unavailable or unresponsive.

1.5.3 Test Cases

A set of detailed manual test cases was created to validate the core functionality of the SauceDemo web application. The test cases cover login functionality, product sorting, cart operations, and navigation between pages.

Both positive and negative scenarios were included to ensure correct system behavior and proper error handling.

The complete list of manual test cases, including test data, expected results, actual results, and execution evidence, is provided in **Appendix A**.

Test Number	Test Case Description	Test Data	Expected Results	Actual Results	Pass/Failed	Execution Evidence
TC00	Verify that user can not log in using invalid password	Username: standard_user Password: invalid password	User should not be able to log in and redirected to the products page	The user did not log in and the products page was not displayed	pass	TC00.…
TC01	Verify that user can log in successfully using valid credentials	Username: standard_user Password: secret_sauce	The user should be logged in successfully and redirected to the products page	The user logged in successfully and products page was displayed	Pass	TC01.…
TC02	Verify that products can be sorted from Z to A	Username: standard_user Password: secret_sauce Sort option: Name (Z to A)	Products should be sorted from Z to A correctly	Products were sorted correctly from Z to A	Pass	TC02.…

1.5.4 Defect Reporting

During manual testing, several defects were identified across different user roles. These defects included functional issues, UI inconsistencies, and performance-related delays.

Each defect was documented with detailed steps to reproduce, expected and actual results, severity, priority, and supporting evidence such as screenshots or screen recordings.

Defect Metrics Summary

During manual testing, a total of **11 defects** were identified and documented across different functional areas of the SauceDemo application. Defects were classified based on severity and priority to assess their impact on system functionality

The distribution of defects is summarized below:

Critical: 2 defects (blocking core functionality such as checkout or cart operations)

High: 4 defects (major functional issues with no acceptable workaround)

Medium: 3 defects (functional issues with limitations or partial impact)

Low: 2 defects (minor UI or cosmetic issues)

Most defects were related to cart behavior, sorting inconsistencies, and UI alignment issues observed under specific user roles. Defect metrics helped prioritize retesting efforts and focus on high-impact areas.

The complete list of reported defects is provided in **Appendix B**.

Root Cause Analysis

An analysis of reported defects indicates that most issues were not isolated to a single feature but were caused by underlying design and implementation limitations.

The main root causes identified include:

Role-based behavior differences: Certain user roles (e.g., problem_user, error_user) exhibited inconsistent behavior due to intentionally injected defects in the application design

UI rendering inconsistencies: Layout and alignment issues were caused by improper handling of responsive UI elements.

Insufficient validation: Some defects were related to missing validation for user inputs during checkout and cart operations.

State management issues: Cart state inconsistencies occurred when items were added or removed repeatedly without proper synchronization.

Identifying these root causes helped guide defect prioritization and informed recommendations for future improvements.

Retesting and Defect Closure

After defect reporting, retesting activities were planned to verify fixes once they become available. Each defect status was tracked to ensure proper resolution before closure

The retesting process includes:

- Re-executing the original test cases associated with reported defects
- Verifying that the expected behavior is achieved after fixes
- Ensuring no regression issues were introduced as a result of changes

Defects were marked as closed once the fix was verified successfully or reopened if the issue persisted. This process ensured accurate defect lifecycle management and improved overall test reliability.

1.6 UI Automation Testing (Selenium + TestNG)

1.6.1 Automation Approach

UI automation testing was implemented to validate critical user workflows and reduce regression risks. Selenium WebDriver with TestNG was selected due to its stability, flexibility, and suitability for Java-based automation frameworks.

Automation focused on high-risk and high-impact scenarios such as login, product sorting, cart operations, checkout, logout, and invalid login handling.

1.6.2 Automated Test Scenarios

The following scenarios were automated as part of UI automation testing:

- Log in with valid credentials
- Product sorting using available sorting options
- Adding all products to the cart
- Removing products from the cart
- Adding products from product details pages
- removeItemFromCart
- Checkout process (entering user information and completing the order)
- Logout functionality
- Login with invalid credentials

These scenarios represent the core user journeys of the application and provide confidence in system stability after changes.

1.6.3 How to Run the Automation Tests

To execute the UI automation tests:

1. Install Java JDK and Eclipse IDE
2. Add Selenium WebDriver and TestNG dependencies
3. Run the TestNG test class from the IDE
4. Review execution results and assertions in the console

Sample execution results and screenshots are provided as supporting evidence.

1.6.4 Automation Structure

The automation framework follows a simple and maintainable structure using TestNG test prioritization. Each test case represents a clear user flow, and assertions are added to validate expected behavior.

Key features of the automation setup include:

- Centralized WebDriver setup and teardown
- Priority-based test execution
- Reusable locators and actions
- Assertions for navigation, content validation, and error handling

1.6.5 Execution Results

The automation of the UI was achieved using Selenium WebDriver with TestNG as the automation tool. There are a total of nine test cases that are to be performed in a sequential manner to achieve an end-to-end test scenario.

The test cases that are automated involve login, sorting of products, addition of products to the cart, removal of products from the cart, checkout, logout, and invalid login.

The assertions were added to test the navigation of the pages, messages displayed on the screen, and the expected behavior of the application.

The test cases were achieved using TestNG, where the test cases were assigned a priority to be performed in a sequential order. Also, the setup and teardown were added to the test cases to close the browser session.

The automation code is available in the GitHub repository of the project.

Appendix C – UI Automation Code

```
35
36      @Test(priority = 1)
37
38      public void Login() {
39
40
41          WebElement theUserNameInputField = driver.findElement(By.id("user-name"));
42          theUserNameInputField.sendKeys(TheEmail);
43
44          WebElement thePasswordInputfield = driver.findElement(By.id("password"));
45          thePasswordInputfield.sendKeys(ThePassword);
46
47          WebElement theLoginButton = driver.findElement(By.id("login-button"));
48          theLoginButton.click();
49
50          Assert.assertTrue(driver.getCurrentUrl().contains("inventory"));
51
52      }
53
```

Sample automation code and execution evidence are provided in the project repository.

2. DummyJSON Backend Testing

2.1 Introduction

DummyJSON was utilized as a mock backend service to conduct API functional and performance tests.

DummyJSON provides RESTful APIs, which mimic real backend services for authentication, product management, users, and cart operations.

In this section, backend functionality will be validated using API testing tools, and performance tests will be conducted on the system.

2.2 Objectives

The objectives of DummyJSON backend testing are

- Validate API endpoints behavior using functional API testing
- Verify correct handling of requests and responses
- Test both positive and negative API scenarios
- Automate API validation using assertions
- Evaluate API performance under different load profiles
- Identify potential performance bottlenecks and limitations

2.3 Domain Understanding

DummyJSON represents a simulated e-commerce backend system. It exposes REST APIs that allow clients to authenticate users, retrieve product data, manage carts, and access user information.

The main backend domains tested include:

- Authentication (login)
- Products (list, search, categories, add, update, delete)
- Users
- Carts

Understanding these domains helped in designing comprehensive API test cases and performance scenarios.

2.4 Scope of Testing

In Scope

- API functional testing using Postman
- Validation of request and response structure
- Positive and negative API scenarios
- API automation using Newman
- API performance testing using k6

Out of Scope

- Real database validation
- Security and penetration testing
- Production environment testing
- Data persistence verification (DummyJSON uses mock data)

2.5 API Functional Testing (Postman)

2.5.1 API Test Plan

For API functional testing, Postman was used for the DummyJSON REST API. The test plan for API functional testing was based on the following aspects:

- Request handling for the API.
- Response status code for the API.
- Response body for the API.
- Error handling for the API.
- Negative test cases for the API.
- Positive test cases for the API.

Testing was performed in a controlled environment using environmental variables. The requests sent to the API had automated assertions.

2.5.2 API Test Cases

API test cases were designed to cover the main backend functionalities, including authentication, product operations, user retrieval, and cart management.

The test cases included:

- Authentication using valid and invalid credentials
- Retrieving product lists and product details
- Searching and filtering products
- Adding, updating, and deleting products
- Retrieving users and cart data
- Handling invalid inputs and non-existing resources

Both positive and negative test cases were executed to ensure correct API behavior and proper error handling

2.5.3 Environment Variables & Token Handling

Environment variables were utilized to enhance the reusability and maintainability of the tests. The base URL was stored as an environment variable to avoid hardcoded values in the requests.

The authentication tokens received after calling the login endpoint were captured and stored as environment variables. These tokens were then utilized in subsequent requests that required authentication.

The above strategy allowed dynamic execution of the test scenarios involving the API.

2.5.4 Automated Assertions

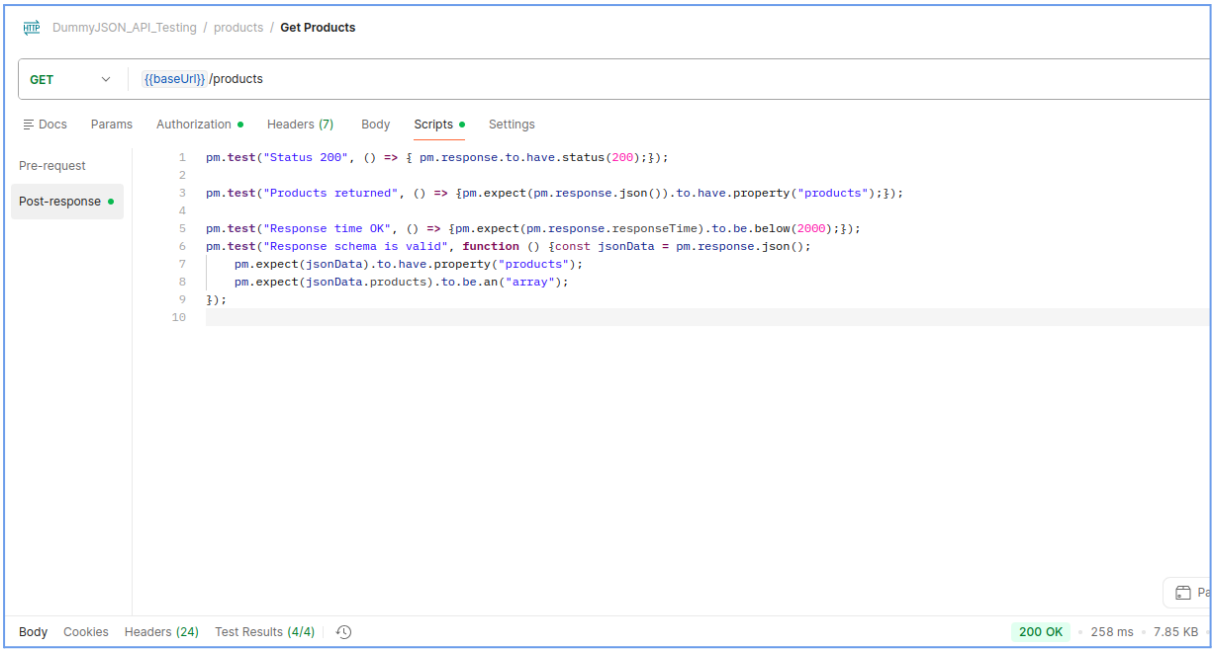
Automated assertions were added to each API request to validate:

- HTTP status codes
- Response body content
- Presence of required fields
- Response time thresholds

These assertions ensured immediate validation of API behavior and helped identify issues quickly during execution.

Schema Validation

Basic response schema validation was applied to selected API endpoints to ensure structural correctness of responses. Schema checks verified the presence of required fields and correct data types, such as validating that product lists are returned as arrays. This validation adds an extra layer of confidence in API reliability beyond status code verification.



Postman schema validation test for GET /products endpoint

Advanced API Insights and Error Handling

Advanced API testing scenarios were executed to validate system behavior under invalid and edge-case conditions. These scenarios included sending requests with invalid credentials, accessing nonexistent resources, and performing operations with incorrect input data.

The API demonstrated consistent handling of error cases by returning appropriate HTTP status codes and informative response messages. Negative test cases helped verify system robustness and ensured that unexpected inputs did not cause system instability.

Insights gained from these scenarios indicate that while the DummyJSON APIs handle most invalid inputs gracefully, the mock nature of the backend may result in non-standard behavior for certain operations such as update and delete requests. These findings were documented and considered during performance testing analysis.

API Defect Analysis and Validation Gaps

API functional testing revealed a critical validation gap in the Add Product endpoint. The API accepts invalid values for the price field, including negative numbers, zero, null values, and non-numeric strings, and still returns a successful creation response.

This behavior indicates missing server-side input validation and can lead to inconsistent or corrupted business data. The issue was identified through data-driven testing using a CSV file, which demonstrates the effectiveness of parameterized testing in uncovering backend validation defects.

The detailed defect report for this issue is provided in **Appendix B**.

2.5.5 Newman Execution Report

The Postman collection was run using Newman via the command line interface. Newman was used to running the API and creating an HTML report for the execution.

The Newman report contains information such as the requests made, pass or fail results, response time, and assertion results. The evidence of execution and the HTML report are included in the project.

2.6 API Performance Testing (k6)

2.6.1 Performance Test Objectives

The objective of API performance testing was to evaluate the responsiveness, stability, and scalability of DummyJSON APIs under different load conditions. Performance testing aimed to ensure that selected API endpoints can handle concurrent users while maintaining acceptable response times.

2.6.2 Target Endpoints

The following API endpoints were selected for performance testing, as they represent critical backend operations:

- DELETE /products/1
- GET /products
- GET /products/{id}

Additional endpoints were included to provide broader backend coverage during execution.

2.6.3 Smoke and Load Profiles

Two performance testing profiles were defined using k6:

- **Smoke Testing:** Executed with a small number of virtual users (3 users) for a short duration to verify basic system availability and responsiveness.
- **Load Testing:** Executed with 25 concurrent virtual users to simulate sustained user traffic and evaluate system behavior under load conditions.

These profiles helped identify performance characteristics at different usage levels.

Ramp-up Strategy

A gradual ramp-up strategy was applied during load testing to simulate realistic user behavior. Instead of introducing all virtual users at once, users were increased progressively to allow the system to stabilize under increasing load.
This approach helped identify performance degradation points and reduced the risk of sudden spikes that do not reflect real-world usage patterns.

Think Time Strategy

Think time was incorporated into performance testing scripts to simulate realistic user behavior. A short delay was added between consecutive API requests to represent the time a user typically spends reviewing data or navigating between actions.

Introducing think time helps avoid unrealistically aggressive request patterns and provides more accurate performance insights under normal usage conditions.

2.6.4 KPIs and Thresholds

Key performance indicators (KPIs) were defined to assess API performance:

- Average response time
- 95th percentile response time (p95)
- Request throughput (requests per second)
- Error rate

Thresholds were configured to ensure that the 95th percentile response time remained below 2 seconds and that the error rate stayed within acceptable limits.

KPI Interpretation and Bottleneck Identification

Performance metrics collected during k6 execution were analyzed to identify system behavior under load. The average and 95th percentile response times remained within acceptable thresholds, indicating that the APIs can handle concurrent requests efficiently under normal conditions.

Throughput measurements demonstrated stable request processing rates during both smoke and load tests. An increased error rate observed during full collection execution was primarily related to the inclusion of negative test scenarios and authentication-related dependencies.

Potential bottlenecks identified include repeated authentication requests under load and backend processing limitations of the mock DummyJSON service. These bottlenecks are not representative of production systems but provide valuable insights into interpreting performance results when using simulated backends.

2.6.5 Execution Results

The k6 performance tests were executed successfully for both smoke and load profiles. Results indicated that API response times remained within the defined thresholds, with the 95th percentile response time staying well below the 2-second limit.

Performance metrics such as response time distribution, throughput, and error rates were captured from the k6 execution output and documented as execution evidence.

2.6.6 Performance Analysis and Recommendations

Performance analysis revealed that DummyJSON APIs demonstrated strong response time performance under concurrent load. Observed error rates were mainly attributed to negative test scenarios, authentication dependencies, and limitations inherent to the mock backend environment.

Potential performance bottlenecks include repeated authentication requests and backend processing overhead during concurrent access. Recommended improvements include caching authentication tokens, isolating negative scenarios from performance test runs, optimizing backend request handling, and introducing rate limiting and connection pooling to improve scalability.

3. CI/CD Integration (GitHub)

3.1 Introduction

Continuous Integration and Continuous Deployment (CI/CD) practices were considered as part of this project to demonstrate how automated testing can be integrated into a development workflow. GitHub was used as a version control platform to manage source code and testing artifacts.

This section highlights the importance of CI/CD in improving software quality and enabling continuous testing.

3.2 Pipeline Overview

A basic CI/CD workflow was designed using GitHub to support automated testing activities. The pipeline concept focuses on triggering automated tests whenever code changes are pushed to the repository.

The pipeline includes the following stages:

- Source code management using Git
- Automated UI test execution
- Automated API testing
- Reporting of test execution results

This approach ensures early detection of defects and maintains application stability.

3.3 Automation Execution

Automated test scripts for UI and API testing were stored and managed within a GitHub repository. Test execution can be triggered manually or automatically through CI/CD workflows.

Although full deployment automation was not implemented, the integration demonstrates how automated testing can be incorporated into a continuous integration environment to support regression testing and quality assurance activities.

3.4 Benefits of CI/CD Integration

Integrating automated testing within a CI/CD pipeline provides several benefits, including:

- Faster feedback on code changes
- Early detection of defects
- Reduced manual testing effort
- Improved test coverage through automation
- Increased confidence in application stability

4. Conclusion

This project provided a comprehensive hands-on application of software testing principles across multiple independent systems. Manual testing and UI automation were applied on the SauceDemo web application to validate frontend functionality, business workflows, and user experience. Functional defects were identified, analyzed, and documented using a structured defect management process.

Backend testing was conducted separately using DummyJSON APIs, where API functional testing was performed with Postman and Newman, followed by API performance testing using k6. Smoke and load testing profiles were executed to evaluate system responsiveness, stability, and scalability under concurrent usage.

The project successfully demonstrated the application of different testing techniques, including manual testing, UI automation, API testing, and performance testing. It also highlighted the importance of risk-based testing, metrics-driven quality assessment, and performance analysis. Overall, the project strengthened practical QA skills and provided a solid understanding of end-to-end software testing practices.

5. References

- SauceDemo. https://www.saucedemo.com
- DummyJSON API Documentation. https://dummyjson.com
- GitHub Repository – Final QA Project https://github.com/maiAtaha/FinalQAProject

6. Appendix

6.1 Appendix A – Manual Test Cases

pendix contains the detailed manual test cases executed for the SauceDemo web application. Test cases were designed to cover both positive and negative scenarios and include test data, expected results, actual results, execution status, and supporting evidence.

Test Number	Test Cae Descreption	Test Data	Expected Results	Actual Results	Pass/Failed	notes
TC00	Verify that user can not log in using invalid password	Username: standard_user Password: invalid password	User should not be aple to log in and redirected to the products page	User did'nt log in and the products page was not displayed	Pass	TC00.png
TC01	Verify that user can log in successfully using valid credentials	Username: standard_user Password: secret_sauce	User should be logged in successfully and redirected to the products page	User logged in successfully and products page was displayed	Pass	TC01.webm
TC02	Verify that products can be sorted from Z to A	Username: standard_user Password: secret_sauce Sort option: Name (Z to A)	Products should be sorted from Z to A correctly	Products were sorted correctly from Z to A	Pass	TC02.webm
TC03	Verify that user can add a product to the cart	Username: standard_user Password: secret_sauce Product: Sauce Labs Backpack (Sauce Labs Backpack)	Product should be added to the cart successfully	Product was added to the cart successfully	Pass	TC03.webm
TC04	Verify that cart icon navigates to cart page	Username: standard_user Password: secret_sauce Click on cart icon	User should be redirected to the cart page	User was redirected to the cart page successfully	Pass	TC04.webm
TC05	Verify that user can remove a product from the cart	Username: standard_user Password: secret_sauce Product: Sauce Labs Backpack	Product should be removed from the cart	Product was removed successfully from the cart	Pass	TC05.webm

Test Number	Test Cae Description	Test Data	Expected Results	Actual Results	Pass/Failed	notes
TC06	Verify that user can proceed to checkout	Username: standard_user Password: secret_sauce with (Sauce Labs Backpack) items in cart	User should be redirected to checkout information page	User was redirected to checkout page successfully	Pass	TC06.webm
TC07	Verify that checkout information can be submitted successfully	Username: standard_user Password: secret_sauce first name :mai , last name :taha, postal code :11188	User should proceed to order overview page	User was redirected to order overview page	Pass	TC07.webm
TC08	Verify that order can be completed successfully	Username: standard_user Password: secret_sauce Click Finish button	Order confirmation message should be displayed	Order confirmation message displayed successfully	Pass	TC08.webm
TC09	Verify that user can add a product to the cart	Username: error_user Password: secret_sauce Product: Sauce Labs Fleece Jacket	Product should be added to the cart successfully	Product was not added to the cart	Failed	TC09.webm
TC10	Verify that checkout information can be submitted successfully	Username: error_user Password: secret_sauce Product: Sauce Labs Backpack first name : mai last name : taha	User should be able to write in last name feild	user can not write in the last name feild	Failed	TC10.webm
TC11	Verify that user cannot submit checkout information with the last name missing	Username: error_user Password: secret_sauce Product: Sauce Labs Backpack first name : mai last name : taha postal code : 11188	User should not be able to submit checkout information with the last name missing	user was able to submit checkout information with the last name missing	Failed	TC11.webm
TC12	Verify finish button functionality during checkout	Username: error_user Password: secret_sauce Checkout process	User should be able to click Finish and complete checkout	Finish button was not clickable	Failed	TC12.webm
TC13	Verify cancel button functionality during checkout	Username: error_user Password: secret_sauce Click Cancel button on checkout page	User should be redirected back to the cart page	User was redirected back to the cart page successfully	Pass	TC13.webm
TC14	Verify sorting functionality from Z to A	Username: error_user Password: secret_sauce Sort option: Name (Z to A)	Products should be sorted correctly	Sorting did not work as expected	Failed	TC14.webm
TC15	Verify back to products button functionality	Username: error_user Password: secret_sauce Click Back to Products button	User should be redirected to products page	User was redirected successfully to products page	Pass	TC15.webm

Test Number	Test Cae Descreption	Test Data	Expected Results	Actual Results	Pass/Failed	notes
TC16	Verify product images display correctly	Username: problem_user Password: secret_sauce	Each product should display its correct image	All products displayed the same image	Failed	TC16.png
TC17	Verify sorting by price from low to high	Username: problem_user Password: secret_sauce Sort option: Price (Low to High)	Products should be sorted by price from low to high	Sorting by price did not work	Failed	TC17.webm
TC18	Verify that user can add product to cart	Username: problem_user Password: secret_sauce Product: Sauce Labs Bike Light	Product should be added to cart successfully	Product added successfully	Pass	TC18.webm
TC19	Verify last name field behavior on checkout information page	Username: problem_user Password: secret_sauce First Name: Mai Last Name: taha	Last name should be entered correctly without affecting first name	First name value was overwritten while typing last name	Failed	TC19.webm
TC20	Verify add to cart button alignment	Username: visual_user Password: secret_sauce Product: Sauce Labs Bike Light	Add to cart button should be properly aligned	Add to cart button is misaligned	Failed	TC20.png
TC21	Verify checkout button alignment	Username: visual_user Password: secret_sauce Product: Sauce Labs Bike Light	Checkout button should appear in correct position	Checkout button appears in incorrect position	Failed	TC21.png
TC22	Verify menu icon alignment	Username: visual_user Password: secret_sauce	Menu icon should be properly aligned	Menu icon is tilted/misaligned	Failed	TC22.png
TC23	Verify application performance for performance_glitch_user	Username: performance_glitch_user Password: secret_sauce	Actions should respond within acceptable time	Login, sort, and navigation actions took approximately 4 seconds	Failed	TC23.webm

6.2 Appendix B - Defect Report

Manual Testing Bugs

This appendix contains the complete list of defects identified during manual testing of the SauceDemo web application. Each defect was documented with clear reproduction steps, expected and actual results, severity, priority, and supporting evidence.

A total of **12 defects** were identified during testing. These defects include functional issues, UI inconsistencies, and performance-related delays observed under specific user roles.

Test Case Number	Name	Reporter	Sammary	Execution Evidence	Browser	Expected Result	Actual Result	Severity	Priority	Steps to Reproduce
TC09	Verify add to cart functionality	Mai Taha	Add to Cart functionality does not work	TC09.webm	FireFox	Selected product should be added to the cart successfully	Product is not added to the cart	High	High	Login With: Username: error_user Password: secret_sauce From the products page, locate Sauce Labs Fleece Jacket. Click on Add to Cart. Click on the cart icon.
TC10	Verify that checkout information can be submitted successfully	Mai Taha	user can not write in the last name feild	TC10.webm	FireFox	User should be able to write in last name feild	user can not write in the last name feild	Medium	Medium	Log in using: Username: error_user Password: secret_sauce Add Sauce Labs Backpack to the cart. Navigate to the cart page. Click Checkout. Enter a value in First Name field. Try typing in the Last Name field
TC11	Verify that user cannot submit checkout information with the last name missing	Mai Taha	user was able to submit checkout information with the last name missing	TC11.webm	FireFox	User should not be able to submit checkout information with the last name missing	user was able to submit checkout information with the last name missing	Medium	Medium	Log in using: Username: error_user Password: secret_sauce Add Sauce Labs Backpack to the cart. Navigate to Checkout. Enter First Name and Postal Code. Leave Last Name field empty. Click Continue.
TC12	Verify finish button functionality during checkout	Mai Taha	Finish button is not clickable on the checkout confirmation page	TC12.webm	FireFox	Finish button should complete the checkout process	Finish button does not respond when clicked	Critical	High	Log in using: Username: error_user Password: secret_sauce Add any product to the cart. Complete checkout information. Navigate to the checkout overview page. Click on Finish button.
TC14	Verify sorting functionality from Z to A	Mai Taha	Product sorting from Z to A does not work	TC14.webm	FireFox	Products should be sorted from Z to A	Products order does not change	Medium	Medium	Log in using: Username: error_user Password: secret_sauce On the products page, open the sort dropdown. Select Name (Z to A).
TC16	Verify product images display correctly	Mai Taha	All products display the same image on the products page	TC16.png	FireFox	Each product should display its correct image	All products show identical images	Medium	Medium	Log in using: Username: problem_user Password: secret_sauce Navigate to the products page. Observe the images of all products.
TC17	Verify sorting by price (Low to High)	Mai Taha	Sorting products by price from low to high does not work	TC17.webm	FireFox	Products should be sorted by ascending price	Products remain unsorted	Medium	Medium	Log in to the application Navigate to the products page Select "Price (Low to High)" from the sorting options
TC19	Verify checkout form input behavior	Mai Taha	Input values are written in the wrong checkout form fields	TC19.webm	FireFox	Each input field should keep its own value	Typing in one field overwrites another field	High	High	Log in using: Username: problem_user Password: secret_sauce Add any product to the cart. Navigate to Checkout. Enter a value in First Name field. Enter a value in Last Name field. Observe the First Name field
TC20	Verify add to cart button alignment	Mai Taha	Add to Cart button is misaligned on the product card	TC20.png	FireFox	Add to Cart button should be properly aligned	Button appears in incorrect position	Low	Low	Log in using: Username: visual_user Password: secret_sauce Navigate to the products page. Observe the Add to Cart button on product cards.
TC21	Verify checkout button position	Mai Taha	Checkout button appears in the wrong position after adding items	TC21.png	FireFox	Checkout button should appear in correct position	Checkout button is misaligned	Low	Medium	Log in using: Username: visual_user Password: secret_sauce Add a product to the cart. Navigate to the cart page. Observe the Checkout button position.
TC22	Verify menu icon alignment	Mai Taha	Menu icon is visually misaligned	TC22.png	FireFox	Menu icon should be properly aligned	Menu icon appears tilted	Low	Low	Log in using: Username: visual_user Password: secret_sauce Observe the menu icon in the header.
TC23	Verify application performance	Mai Taha	Application response time is slow during common actions	TC23.webm	FireFox	Actions should respond within acceptable time limits	Actions take around 4 seconds to respond	High	Medium	Log in using: Username: performance_glitch_user Password: secret_sauce Perform common actions: Login Sorting products Navigation between pages

API Testing Bug

Bug ID	Related Test Case	Bug Title	Summary	Endpoint	Steps to Reproduce	Expected Result	Actual Result	Severity	Priority
API-BUG-01	Add Product (csv file)	Add Product API allows invalid price values	The Add Product API accepts invalid values for the price field, including negative numbers, zero, null values, and non-numeric strings, and still creates the product	POST /products/add	Open Postman Send a POST request to /products/add Use any of the following invalid values for the price field: Negative number Zero Null String value Execute the request Observe the response status code and body	The API should reject invalid input Response status code should be 400 (Bad Request) or 422 (Validation Error) Product should not be created	The API returns 201 Created Product is created successfully with invalid price values	High Reason: This bug allows invalid business data to be stored, which can cause pricing, reporting, and financial calculation issues.	High Reason: Input validation is critical for backend data integrity and should be fixed before release.

6.3 Appendix C – UI Automation Test Code

This appendix references the automated UI test scripts developed using Selenium WebDriver and TestNG. The automation suite covers critical user flows, including login, product sorting, cart operations, checkout, logout, and invalid login validation.

Full automation source code is available in the project GitHub repository.

```java
package SauceDemo;

import java.time.Duration;
import java.util.List;
import java.util.Random;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.Select;
import org.testng.Assert;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class myTestCase {

    WebDriver driver;
    String TheWebSite = "https://www.saucedemo.com/";
    Random random = new Random();
    String TheEmail = "standard_user";
    String ThePassword = "secret_sauce";


    @BeforeTest

    public void mySetup() throws InterruptedException {

        driver = new FirefoxDriver();
        driver.get(TheWebSite);
        driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
        driver.manage().window().maximize();

    }

    @Test(priority = 1)

    public void Login() {


        WebElement theUserNameInputField = driver.findElement(By.id("user-name"));
        theUserNameInputField.sendKeys(TheEmail);

        WebElement thePasswordInputfield = driver.findElement(By.id("password"));
        thePasswordInputfield.sendKeys(ThePassword);

        WebElement theLoginButton = driver.findElement(By.id("login-button"));
        theLoginButton.click();

        Assert.assertTrue(driver.getCurrentUrl().contains("inventory"));

    }

```

```
54
55      @Test(priority = 2)
56
57      public void randomSorting() {
58
59          WebElement sortDropdown = driver.findElement(By.className("product_sort_container"));
60          Select sort = new Select(sortDropdown);
61
62          String[] sortOptions = {"az", "za", "lohi", "hilo"};
63          String randomSortOption = sortOptions[random.nextInt(sortOptions.length)];
64          sort.selectByValue(randomSortOption);
65
66          System.out.println("Sorting applied: " + randomSortOption);
67
68          Assert.assertTrue(true);
69          }
70
71      @Test(priority = 3)
72      public void AddAllItems()  {
73
74          List<WebElement> addToCartButtons = driver
75                  .findElements(By.cssSelector(".btn.btn_primary.btn_small.btn_inventory"));
76
77          for(int i =0 ; i <addToCartButtons.size();i++) {
78              addToCartButtons.get(i).click();
79          }
80      }
81
82
83      @Test(priority = 4)
84      public void RemoveItem() {
85
86          List<WebElement> removeFromThecartButtons = driver
87                  .findElements(By.cssSelector(".btn.btn_secondary.btn_small.btn_inventory"));
88
89          for(int i =0 ; i <removeFromThecartButtons.size();i++) {
90              removeFromThecartButtons.get(i).click();
91                  }
92
93
94      }
95
96      @Test(priority = 5)
97      public void addItemFromProductPage() {
98
99          WebElement product1Page = driver.findElement(By.className("inventory_item_name"));
100         product1Page.click();
101
102         WebElement addToCartButton1 = driver.findElement(By.xpath("//button[contains(text(),'Add to cart')]"));
103         addToCartButton1.click();
104
105         WebElement backToProductButton1 = driver.findElement(By.id("back-to-products"));
106         backToProductButton1.click();
107
108     }
```

```
110     @Test(priority = 6)
111     public void removeItemFromCart() {
112
113         WebElement goToCart = driver.findElement(By.className("shopping_cart_link"));
114         goToCart.click();
115
116         WebElement RemoveFromCAart = driver.findElement(By.xpath("//button[contains(text(),'Remove')]"));
117         RemoveFromCAart.click();
118
119     }
120
121     @Test(priority = 7)
122     public void checkoutProcess() {
123
124         WebElement checkoutButton = driver.findElement(By.id("checkout"));
125         checkoutButton.click();
126
127         WebElement firstNameInput = driver.findElement(By.id("first-name"));
128         firstNameInput.sendKeys("Mai");
129
130         WebElement lastNameInput = driver.findElement(By.id("last-name"));
131         lastNameInput.sendKeys("Taha");
132
133         WebElement postalCodeInput = driver.findElement(By.id("postal-code"));
134         postalCodeInput.sendKeys("1703");
135
136         WebElement continueButton = driver.findElement(By.id("continue"));
137         continueButton.click();
138
139         WebElement finishButton = driver.findElement(By.id("finish"));
140         finishButton.click();
141
142         Assert.assertTrue(driver.getPageSource().contains("Thank you for your order"));
143
144         WebElement backHomeButton = driver.findElement(By.id("back-to-products"));
145         backHomeButton.click();
146
147        }
148
149
150     @Test(priority = 8)
151     public void logout() {
152
153         WebElement menuButton = driver.findElement(By.id("react-burger-menu-btn"));
154         menuButton.click();
155
156         WebElement logoutButton = driver.findElement(By.id("logout_sidebar_link"));
157         logoutButton.click();
158
159         Assert.assertTrue(driver.getCurrentUrl().contains("saucedemo"));
160     }
161
162     @Test(priority = 9)
163     public void invalidLogin() {
164
165         WebElement userNameInput = driver.findElement(By.id("user-name"));
166         userNameInput.sendKeys("invalid_user");
167
168         WebElement passwordInput = driver.findElement(By.id("password"));
169         passwordInput.sendKeys("wrong_password");
170
171         WebElement loginButton = driver.findElement(By.id("login-button"));
172         loginButton.click();
173
174         WebElement errorMessage = driver.findElement(By.cssSelector("[data-test='error']"));
175
176         Assert.assertTrue(errorMessage.isDisplayed());
177
178     }
179
180     @AfterTest
181     public void tearDown() {
182         driver.quit();
183     }
184
185
186 }
```

6.4 Appendix D – CI/CD Execution Evidence