**Test Automation Plan For ManafOnlineStore Web Application ( Version 1.0 )**

This document is a high-level document that specifies the test automation procedure of the ManafOnlineStore project. The documents describe the need for performing automation testing on the existing web application, which is ongoing. The document is not a static one and so may have some changes to suit the new requirements and appropriate process deemed necessary. The document has the overview of the project, the test automation objectives that need to be achieved, and the Application details, which include the tech stacks for the development of the web application. In this document, the scope of testing, tools, roles and responsibilities of members of the team, etc, are captured.

**Project Overview**

The e-commerce application allows users to browse products, add items to the shopping cart, and complete the purchase. The application serves as a comprehensive online marketplace, offering a diverse range of products spanning various categories, including electronics, fashion, home goods, and more. Users can create accounts, providing a personalised experience with features such as order history, saved items, and exclusive promotions. Robust authentication mechanisms ensure the security and privacy of user information. The application provides an intuitive and user-friendly interface for customers to easily navigate through products. Advanced search functionality allows users to find specific items based on criteria such as category, brand, price range, and customer ratings. Each product listing includes detailed information, high-quality images, and customer reviews to aid in informed purchasing decisions. Customers can contribute by leaving reviews and ratings, fostering a sense of community and trust. The architecture is designed for scalability to accommodate potential growth in user traffic and product offerings. Continuous monitoring and performance optimisation ensure a responsive and reliable platform. Security protocols, including data encryption and secure sockets layer (SSL), are implemented to protect user data and financial transactions. The application complies with relevant data protection regulations and industry standards.

**Assumptions**

The test automation engineers are skilled in Selenium webdriver with java and have the ability to design a framework. Assumptions for Test Automation Plan - ManafOnlineStore E-commerce Application:

* **Stable Application Environment:** The existing ManafOnlineStore e-commerce web application maintains a stable environment throughout the automation process. Changes in the application environment, such as frequent updates or modifications, may impact the reliability of automated scripts.
* **Consistent Test Data:** Consistent and representative test data is available for conducting automated tests. Automation scripts rely on specific test data to validate functionality. Inconsistencies or variations in test data may lead to inaccurate test results.
* **Application Accessibility:** The ManafOnlineStore application will be accessible during the scheduled automation testing periods. Any unforeseen downtime or unavailability of the application may affect the execution of automated scripts and lead to false negatives.
* **Stakeholder Cooperation:** Stakeholders, including developers and product owners, will provide timely cooperation in addressing automation-related issues.

Collaborative efforts are essential for efficient automation. Delays in communication or resolution may hinder the progress of the automation plan.

* **Automation Tool Compatibility:** The selected automation tools (Selenium, Junit) are compatible with the ManafOnlineStore application's technologies and frameworks.

Incompatibility issues may arise if the chosen automation tools do not align with the technologies used in the application, leading to script failures.

* **No Frequent UI Changes:** The user interface (UI) of the ManafOnlineStore application undergoes minimal changes during the automation process.

Frequent UI changes may require continuous script maintenance, potentially increasing the overhead of test automation.

* **Test Environment Stability:** The test environment closely mirrors the production environment, ensuring that automation results accurately reflect the application's behaviour in a live setting. Discrepancies between the test and production environments may result in false positives or negatives during automated testing.
* **Network Stability:** A stable and reliable network connection is available during the execution of automated test scripts. Network issues may lead to timeouts, disruptions, or inconsistencies in test execution, impacting the reliability of automation results.
* **Security and Authorization**: The necessary access rights and authorizations for automation testing are granted to the testing team. Lack of proper permissions may restrict access to certain functionalities, hindering the execution of comprehensive automated tests.
* **Maintenance of Test Scripts**: Regular maintenance efforts will be allocated to update and enhance automated test scripts as needed. Application changes or updates may require adjustments to existing scripts to ensure they remain effective and aligned with the current application state.

**Testing Objectives**

* Accelerate the testing process by implementing automated tests.
* Achieve a high level of code coverage to ensure that a significant portion of the codebase is exercised by automated tests.
* Provide faster feedback to developers through automated systems and end-to-end tests.
* Ensure that all implemented features and functionalities are thoroughly tested and meet the specified requirements.
* Ensure that recent code changes or updates do not adversely impact existing functionalities.
* Support ongoing development and maintenance by quickly identifying and rectifying defects or issues arising from code changes.
* Validate the integrity and accuracy of data during any database migrations or updates.
* Ensure that the application remains compatible with the latest browser and device updates.
* Continuously monitor and ensure optimal performance as the application evolves.
* Identify opportunities to enhance usability and the overall user experience.
* Validate the successful integration of automated tests into the existing CI/CD pipeline.
* Confirm that documentation, including test cases and test data, is up-to-date.
* Assist in code reviews by providing insights into the impact of code changes on existing functionalities.
* Identify and address the root causes of recurring issues or defects.
* Ensure that automated test scripts and associated documentation accurately reflect the current state of the application.
* Design automated tests to be easily adaptable to changes in the application's requirements, user interface, or underlying architecture.
* Validate the application's ability to gracefully handle errors and recover without compromising the user experience.
* Assess the application's ability to scale by simulating increased user loads and transaction volumes.

**Application Details**

* Web-based e-commerce application built with Java and Spring Boot.
* Frontend developed using HTML, CSS, Angular, and JavaScript.
* Supported browsers include Chrome, Firefox, and Safari.

**Current Testing Process**

* Manual testing is currently performed for critical functionalities.
* Developers handle unit testing using JUnit for individual components.

**Automation Plan**

**Scope of Automation**

This comprehensive scope outlines the key areas for test automation in the existing e-commerce project. It encompasses both functional and non-functional aspects, ensuring a thorough and balanced approach to automated testing.

* **Functional Testing**: Functional testing will mainly focus on automating the various important features that are stated in the user stories or the requirements. Automate the features with high business value and are considered as features that define the definition of done. Automate test scenarios for critical user flows, including
* User registration and authentication.
* Product search and browsing.
* Adding items to the shopping cart.
* Checkout and order processing.
* Payment system.
* **Regression Testing:** Automate regression tests to verify the stability of existing functionalities with each new build. Include tests for features that have been implemented or modified during ongoing development in order to ensure that bug fixes and any minute changes to the system haven’t introduced any defect or error in the existing features and the previous behaviour of the application, the automated test scripts shall be run and used to check for these issues. There shall be regression testing for the following situations:
* After Code Changes: Whenever there is a new code change, whether it's a bug fix or a new feature, regression testing should be conducted. This helps ensure that the recent modifications did not introduce unintended side effects or break existing functionality.
* After System Integrations: If there is any system integration to the e-commerce web application that the application interacts with, regression testing should be performed after integrating new changes to ensure that the overall system works seamlessly.
* Before Releases: Before a new version of the software is released to users, a comprehensive regression test suite should be executed. This ensures the software is stable and new features or fixes haven't introduced critical issues.
* Periodically: Even without specific changes, There shall be regression testing to help catch any latent defects that might have been introduced in the codebase over time.
* Automated Continuous Integration: A continuous integration (CI) pipeline is in place, and automated regression tests are integrated into the CI process. This ensures that tests are run automatically whenever there is a new code commit, providing quick feedback to developers.
* After Environment Changes: If there are changes to the development, testing, or production environments, regression testing will be performed to ensure that the web application functions correctly in the updated environment.
* After Configuration Changes
* After Critical Bug Fixes: If a critical bug is identified and fixed, regression testing is crucial to confirm that the bug fix didn't introduce new issues or regressions.
* **Test Coverage:** It's important to note that achieving 100% test coverage does not guarantee a completely bug-free application. Achieve a minimum code coverage of 80% for critical business logic and components. It's possible to have high coverage but still miss certain edge cases or scenarios. Test coverage will be used as part of a holistic testing strategy, complementing other testing practices, including functional testing, integration testing, and user acceptance testing.Achieve a targeted Test coverage percentage by automating tests for key components and business logic. Prioritise test coverage for critical and frequently accessed areas of the application.
* **Cross-Browser and Mobile Testing:** Automate tests to ensure consistent functionality across different browsers (Chrome, Firefox, Safari) and devices (desktop, tablet, mobile).The responsiveness and usability of the application on various screen sizes will be achieved through the automation tests.
* **Security Testing:** Automate security tests to identify vulnerabilities in authentication, authorisation, and data protection. Implement automated tests for secure communication and encryption protocols. Wrong credentials shall be used, and unauthorised access shall be performed to observe how the system reacts to such conditions.
* **Usability and Accessibility**: Automate tests to evaluate the user experience, ensuring a user-friendly interface and adherence to design guidelines. Verify accessibility features to support users with diverse abilities.
* **Error Handling and Recovery:** Automate tests to validate the application's response to errors and its ability to recover gracefully. Include scenarios simulating unexpected errors during user interactions.
* **Build Stability:** Integrate automated tests into the CI/CD pipeline to verify the stability of each build. Detect and report build failures promptly to ensure a reliable development process.
* **Performance Testing**: Automate performance tests to evaluate the application's responsiveness and resource utilisation under various load conditions.

Identify and address performance bottlenecks to optimise application speed.

- Automate end-to-end scenarios, including user registration, product search, and order processing.

- Focus on critical paths and frequently used functionalities in the application.

**Testing Environment**

The Test environment is a setup that closely replicates the characteristics of a production environment and is used for testing software applications. It provides a controlled and isolated space where testing activities can be performed to ensure the reliability, functionality, performance, and other aspects of the software. The primary purpose of a test environment is to validate that the application behaves as expected in different scenarios before it is deployed to the production environment. It consists of hardware and software infrastructures, which makes the testing process efficient and effective.

* Development, testing, and staging environments mirroring production configurations.
* Operating systems: Windows and macOS.
* Browsers: Chrome, Firefox, Safari.

**Automation Tools and Frameworks**

Using Selenium WebDriver with Java and TestNG (or JUnit) for the automation of tests in a Java Spring Boot backend application offers several advantages. Here are some reasons why this combination is a suitable choice, along with tips for achieving excellence in test automation:

* Cross-Browser and Cross-Platform Testing: Selenium WebDriver supports multiple browsers (Chrome, Firefox, Safari, etc.) and platforms, ensuring that the e-commerce application functions consistently across various user environments.
* Robust Web Testing: Selenium is a powerful tool for web automation, providing capabilities for navigating through web pages, interacting with elements, and validating the application's behaviour.
* Community Support and Documentation: Selenium has a large and active community, making it easier to find solutions to common challenges. Extensive documentation and resources are available for learning and troubleshooting.
* Seamless Integration: Since the application is developed in Java, using Java for test automation ensures seamless integration with the existing codebase. It allows sharing of common libraries, functions, and data structures between development and testing.
* Object-Oriented Programming (OOP): Java's object-oriented nature facilitates the creation of modular and maintainable test scripts. Test automation code can be organized using OOP principles, enhancing readability and scalability.
* Test Management and Parallel Execution (TestNG): TestNG provides powerful test management features, allowing the creation of test suites, parameterisation, and parallel test execution. This is crucial for optimising test execution time.
* Annotations and Configuration: TestNG offers annotations that make it easy to define test methods, setup, and teardown methods. Configuration options allow fine-grained control over test execution behaviour.
* Data-Driven Testing: Both TestNG and JUnit support data-driven testing, allowing the use of external data sources to drive tests with different input values. This is beneficial for testing various scenarios without duplicating code.

- Selenium WebDriver with Java for web automation.

- TestNG for test management and parallel execution.

- JUnit for unit test automation handled by developers.

**Test Data**

Test data for the automated tests will be generated dynamically during test execution to enhance flexibility and avoid hardcoding. This approach allows for the creation of varied and realistic scenarios.

* Data-Driven Testing with TestNG Data Providers

The TestNG framework will be leveraged for data-driven testing by utilizing data providers. Data providers in TestNG enable the seamless integration of external data sources, such as Excel spreadsheets, into test cases.

* Excel as a Data Source

The test data required for different test scenarios will be stored in Excel spreadsheets. This allows for easy management, updates, and collaboration on test data. The data in Excel will be organized to align with the test cases and test scenarios, ensuring clarity and maintainability.

* TestNG Data Provider Configuration

TestNG data providers will be configured to retrieve data from the Excel sheets dynamically. The data-driven approach facilitates the execution of the same test case with multiple sets of input data, enabling comprehensive coverage of various scenarios.

***Benefits of Dynamic Test Data***

Flexibility: Dynamic test data generation ensures adaptability to changing requirements and scenarios, reducing the need for manual intervention in updating hardcoded values.

Reusability: The use of data providers allows for the reuse of test cases with different datasets, promoting efficient test script maintenance.

Scalability: As the test suite grows, the dynamic generation of test data supports scalability, making it easier to manage an expanding set of test cases.

Continuous Maintenance of Test Data

Regular updates and reviews of the test data in Excel will be conducted to ensure alignment with evolving business requirements and application changes. Any necessary adjustments to the data providers or Excel sheets will be documented as part of the test maintenance process.

**Estimation Technique**

Agile Planning Poker is a collaborative and iterative estimation technique commonly used in Agile Scrum methodologies. It leverages the collective knowledge and insights of the team to estimate the effort required for specific tasks, user stories, or features.

Prioritize the backlog of user stories, ensuring that high-priority items are estimated first. This allows the team to focus on critical functionalities early in the process.

Planning Poker Session:

Conduct a Planning Poker session with relevant team members, including developers, testers, and product owners. Each participant receives a deck of planning poker cards with values representing effort or complexity (e.g., Fibonacci sequence: 1, 2, 3, 5, 8, 13).

Discussion and Clarifications: Facilitate discussions among team members to ensure a shared understanding of the user story or task. Participants can ask questions and seek clarifications to enhance their knowledge before estimating.

Individual Estimations:

Each team member privately selects a planning poker card representing their estimation of the effort required for the user story. The estimates are kept confidential initially to avoid bias.

Reveal and Discussion: Simultaneously reveal the selected cards. If there is a consensus, the estimate is recorded. If there are discrepancies, team members discuss their rationales and insights to reach a consensus.

Re-Estimation (If Necessary): If significant disagreements persist, conduct additional rounds of discussion and re-estimation until a consensus is reached. This ensures that the team shares a common understanding of the effort involved.

Recording and Documentation: Record the agreed-upon estimates for each user story. This information is crucial for planning sprints, allocating resources, and tracking progress throughout the test automation project.

For the previously built features, Testers will meet with the developers, product owners and other relevant stakeholders to discuss the features with highest business value which need to be automated. There shall be a consensus estimation of the time frame in accordance with the testers' skill sets.

| **Team Members** | **Responsibilities** |
| --- | --- |
| Product Owner | * Define automation goals and priorities * Prioritize user stories for automation * Provide acceptance criteria and business context |
| Scrum Master | * Facilitate Planning Poker sessions * Ensure Agile principles are followed * Remove impediments, support collaboration |
| Test Automation Lead | * Develop automation strategy and guidelines * Oversee overall automation effort * Coordinate with developers for test environment |
| Test Automation Team | * Develop and maintain automated test scripts * Execute automated tests, analyze results * Collaborate with developers for script integration |
| Developers | * Assist in setting up the test environment * Address automation-related issues and bug fixes * Collaborate with automation team for integration |
| Manual Testers | * Review test scenarios, and provide feedback * Conduct manual testing for scenarios that are not automated * Assist in test data preparation |
| Stakeholders | * Review and provide feedback on automated results * Participate in user acceptance testing (UAT) * Communicate priorities and changes to the team |

**Automation Design**

Adopting a modular structure enables the framework to efficiently handle various functionalities. Modules for product search, order placement, and user account management ensure a focused and organized approach. A layered architecture, with distinct components for test scripts, business logic, and utilities, fosters maintainability and scalability. This separation allows for easier updates and enhancements as the application evolves. The automation framework will make use of the Page Object Model. The implementation of the Page Object Model (POM) involves encapsulating web page elements and actions within dedicated classes. This separation of concerns enhances maintainability by isolating page interactions from test code. POM contributes to code reusability, making it easier to adapt the framework to changes in the application's UI. It also simplifies updates when modifying or adding new features to the web application. This framework architecture leverages the principles of the Page Object Model (POM) for maintainability and reusability, centralizing common functionalities in Abstract Components and managing test execution, reporting, and configurations through TestNG and related components. The integration of Extent Reports, Retry mechanisms, and TestNG Listeners enhances the framework's capabilities for effective test automation.

* ***Page Object Classes:*** Each page is represented as a Page Object Class, encapsulating elements and methods specific to that page. Page Objects inherit from Abstract Components, which contains common methods and elements shared across multiple pages.
* ***Abstract Components:*** Abstract Components serve as a centralized repository for common methods and elements utilized by Page Object Classes. Ensures reusability and reduces redundancy across different page objects.
* ***BaseTest Class:*** Responsible for the setup and teardown procedures common to all test cases. Configurations such as browser selection, BeforeMethod, AfterMethod, resource cleanup, and JSON data retrieval are handled in this class. Captures screenshots upon encountering errors, enhancing debugging capabilities.
* ***Configuration Management:*** BaseTest manages the configuration settings, including browser selection, test setup, and teardown procedures. JSON file retrieval ensures that test cases have access to necessary data, promoting flexibility in test scenario parameterization.
* ***TestNG Listeners:*** Utilizes TestNG Listeners to define behaviors on various test events, such as onTestStart, onTestSuccess, etc.Integrates Extent Report for comprehensive HTML-based test reporting.
* ***Retry Mechanism:*** Implements a Retry class that integrates the IRetryAnalyzer interface for retrying failed tests. Configurable retry settings allow fine-tuning based on project requirements.
* ***Before and After Methods:*** Utilizes Before and After methods to execute actions before and after each test case. Enhances test case independence by providing a clean slate before and after execution.
* ***Extent Reports:*** Integrates Extent Reports for visually appealing and informative HTML-based test reports. Enhances report readability with detailed information on test execution outcomes.
* ***Test Runners in XML Files:*** Test cases are organized and executed through TestNG XML files, providing flexibility in test suite configurations. Profiles are set in the pom.xml, allowing seamless integration with TestNG XML files for specific test configurations.
* ***TestNG Test Cases:*** Test cases are written using TestNG annotations, inheriting from the BaseTest class. This ensures standardized setup, teardown, and reporting procedures for each test case.
* ***Retry Class:*** The Retry class, implementing the IRetryAnalyzer interface, manages test case retry configurations. Enhances test robustness by allowing configurable retries for intermittent failures.

TestNG Profiles in Pom.xml Profiles are defined in the pom.xml file, offering flexibility and customization for different test scenarios. Test runners in XML files leverage these profiles to execute specific sets of test cases.

**Framework Architecture**



**Automation Execution**

Each web page will have a dedicated Page Object Class, adhering to the Page Object Model (POM) principles. POM promotes modularity and maintainability, encapsulating page-specific elements and actions in individual classes. Automation engineers will work hand in hand with the manual testers to review the test cases that are written and executed. The appropriate tests will be automated to ensure that it works as expected. With TestNG as the automation test management tool, test cases will be written using separate methods. Each test script method is annotated with the @Test of annotation of the TestNG tool. Assertions will be after the desired actions are performed. Test cases make use of POM, and then results are shown in the HTML file in the reports folder.

**Git and GitHub Workflow**

Automation tests will be organized into branches in the Git repository, following a feature branch workflow. Each feature or test suite will have its dedicated branch for development. Pull requests (PRs) will be created for review and merged into the main branch upon successful PR approval.

**Jenkins CI/CD Pipeline**

Jenkins will orchestrate the Continuous Integration/Continuous Deployment (CI/CD) pipeline for the automation framework. Scheduled runs will be configured every morning at 7:30 am to ensure regular and timely test suite execution.

Browser configurations and parameters will be set in Jenkins for flexible and parameterized test execution.

**TestNG Annotation and Grouping**

TestNG annotations will be applied appropriately to each test method for effective test organization. The @Test(groups) attribute will be used for grouping tests, facilitating easy categorization for different test scenarios.

**Regression Testing**

Tests can be grouped for regression testing through TestNG's @Test(groups) attribute.

A dedicated TestNG XML file (test runner) will be configured to run only the specified tests for regression, meeting stakeholder requests.

**Report Generation and Viewing**

Test reports will be generated in HTML format and stored in the designated "report" folder. Jenkins will provide access to these reports for convenient viewing, aiding in result analysis and transparency.

**Branching Strategy**

Feature branches will be created for each new functionality or test suite, ensuring isolation and collaborative development.

Pull requests will be initiated from feature branches to the main branch, undergo review, and merge upon approval.

**Test Data Management**

Test data will be managed in structured formats (e.g., JSON) and version-controlled within the repository. BaseTest class will retrieve required JSON files, ensuring that test cases have access to necessary data for various scenarios. Data may also come from the Database and external sources like Excel files.

**Test Environment Management**

Docker containers will be utilized to create consistent and reproducible test environments. Docker Compose files will define necessary services and dependencies for testing. Configuration files for environment variables, such as URLs and credentials, will be maintained for each environment.

**Parallel Execution**

Leverage TestNG's parallel execution capabilities for faster suite execution, enhancing overall efficiency.

**Dynamic Browser Selection**

Jenkins will be configured to allow dynamic browser selection as a parameter, ensuring cross-browser compatibility testing.

**Feedback Mechanism**

Configure Jenkins to provide instant notifications and alerts for test successes and failures, facilitating quick response and issue resolution.

**Automation Reporting**

In the event of identified defects during the automation run, a defect report is included in the test report. This report outlines each defect's description, priority, severity, and current status in the defect management process. The defect report contributes to effective communication and collaboration for issue resolution. A robust defect-reporting mechanism is integrated into the framework. This component enhances effective communication and collaboration for swift issue resolution. The defect report encompasses the following elements:

* **Error Logging:** Errors encountered during the automation run are systematically logged to the console, providing immediate visibility into any issues that may have arisen.
* **Extent Report Listener:** The Extent Report listener is configured to ensure comprehensive reporting of error details. It captures pertinent information such as the tester's name, timestamp, and other relevant details associated with errors or defects.
* **HTML Reports:** Detailed error information is embedded within the HTML reports generated and stored in the designated report folder. This allows for a granular examination of errors at various stages of the test script, facilitating a thorough understanding of the encountered issues.
* **Collaborative Reporting:** The reporting process is a collaborative effort, with multiple mechanisms contributing to a comprehensive defect report. The HTML reports serve as a detailed repository, providing a centralized location for error details.
* **Jenkins Integration:** Jenkins, as an integral part of the CI/CD pipeline, supplements the defect reporting process. It provides an additional layer of reporting and contributes to the overall visibility of defects identified during automated test execution.

The designed defect reporting mechanism aims to streamline the identification, communication, and resolution of issues within the ManafOnlineShopping E-commerce web application. By leveraging multiple reporting channels, the framework ensures that all stakeholders have access to detailed and pertinent information regarding defects, fostering a collaborative approach to issue resolution. This reporting framework aligns with industry best practices, promoting efficiency and transparency in the defect management process.

**Automation Maintainance**

The maintenance of the automation framework is a critical aspect that involves continuous improvement and optimization. There shall be code reviews by the Test Automation Lead before test scripts are merged into the main branch. The framework is designed to allow the integration data data from the database or Excel files later. Periodic code refactoring and strict adherence to the company’s coding standards. New functionalities added will not cause issues with previous test scripts.

The automation maintenance activities encompass the following:

* **Enhancing Expectation Handling and Error Logging.**

Ongoing refinement of the expectation handling and error logging mechanisms is prioritized to ensure comprehensive and accurate reporting. Regular updates are made to capture diverse scenarios and provide detailed insights into the root causes of issues encountered during automated test execution.

* **Robustness and Scalability**

The framework is continually evolving to enhance robustness and scalability.

Regular evaluations are conducted to identify opportunities for performance optimization, ensuring the framework's efficiency in handling an increasing number of test cases and scenarios.

* **Adding New Functionalities**

The framework is designed to be dynamic and adaptable to evolving requirements. New functionalities are seamlessly integrated to address additional use cases and accommodate the testing of diverse scenarios that may emerge during the development lifecycle.

* **Code Refactoring**

Periodic code refactoring initiatives are undertaken to maintain code cleanliness and adherence to best practices. Code refactoring enhances readability, reduces technical debt, and contributes to the overall maintainability of the automation framework.

**Documentation Updates**

Continuous updates to documentation are made to reflect changes in the framework. Documentation serves as a comprehensive guide for users, ensuring that any modifications, enhancements, or additions to the framework are well-documented for ease of reference.

* **Version Control Management**

Version control is actively maintained to track changes, enhancements, and additions to the automation framework. Git branches are appropriately managed, and commit messages provide clarity on the purpose and impact of each change.

* **Training and Knowledge Sharing**

Training sessions and knowledge-sharing initiatives are conducted to keep the automation team updated on the latest enhancements and functionalities.

Ensuring that team members are well-versed in the evolving framework contributes to its successful utilization and maintenance.

* **Continuous Evaluation and Feedback**

Regular evaluations of the automation framework's performance are conducted.

Feedback from automation users and stakeholders is actively sought, fostering a continuous improvement cycle based on real-world usage scenarios.

The automation maintenance strategy is aligned with industry best practices to guarantee that the framework remains adaptive, efficient, and capable of meeting the evolving needs of the ManafOnlineShopping E-commerce web application. This proactive approach contributes to the sustained effectiveness and longevity of the automation framework.