Test Strategy - LSE Transformation

**Version Control**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version No** | **Date** | **Author** | **Change Summary** |
| 0.1 | 08/01/2025 | Smrutipada M | Initial Draft |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**1. Introduction**

* **Purpose**: This document outlines the comprehensive test strategy for the LSE transformation application. It defines the testing approach, resources, schedule, and scope to ensure the application meets the specified requirements and quality standards.
* **Scope**: The scope includes testing all aspects of the UI transformation, including functional and non-functional testing.

**2. Objectives**

* Validate that the transformed UI meets all functional requirements and provides a seamless user experience.
* Ensure the UI is intuitive, user-friendly, and accessible to all users.
* Verify the performance and stability of the UI under various conditions, including high load.
* Ensure compatibility across different browsers, devices, and operating systems.
* Identify and fix defects early in the development cycle to reduce the cost and time of rework.

**3. Test Approach**

* **Types of Testing**:
  + **Functional Testing**: Verify that all UI elements and interactions work as expected. This includes testing buttons, forms, navigation, and other interactive elements.
  + **Usability Testing**: Ensure the UI is intuitive, easy to navigate, and meets user expectations. This involves testing the user interface with real users to gather feedback on usability.
  + **Performance Testing**: Assess the responsiveness, load times, and stability of the UI under different conditions. This includes load testing, stress testing, and endurance testing.
    - **Example**: Use JMeter to simulate N concurrent users accessing the application to measure response times and identify performance bottlenecks.
  + **Compatibility Testing**: Test the UI across various browsers (Chrome, Firefox, Safari, Edge) and devices (desktop, tablet, mobile) to ensure consistent behaviour and appearance.
    - Verify that the application renders correctly on various devices/browsers using **BrowserStack**.

|  |  |  |
| --- | --- | --- |
| **Tier 1 browsers – full support for n-2 versions** | | |
| **Tier 2 browsers - P1 and P2 functionality are maintained and work** | | |
| **Tier 3 browsers – No testing carried out and not officially supported** | | |
|  |  |  |
| **Desktop** | | |
| **Tier 1** | **Tier 2** | **Tier 3** |
| Chrome | Firefox | IE |
| Edge | Opera |  |
| Safari | UC |  |
| **Mobile** | | |
| **Tier 1** | **Tier 2** | **Tier 3** |
| Chrome | Firefox | IE |
| Edge | Opera |  |
| Safari | UC |  |

* + **Regression Testing**: Ensure that new changes do not introduce defects into existing functionality. This involves re-running previously executed tests to verify that the application still works as expected.
    - **Example**: After implementing a new feature, run the full suite of regression tests to ensure no existing functionality is broken.
* **Test Levels**:
  + **Integration Testing**: Test interactions between integrated components to ensure they work together as expected.
  + **System Testing**: Test the complete system to ensure it meets the specified requirements.
  + **User Acceptance Testing (UAT)**: Validate the UI with end-users to ensure it meets their needs and expectations. This is typically the final phase of testing before the application is released.
    - Conduct UAT sessions with key stakeholders to gather feedback on the new UI and make any necessary adjustments.

**4. Test Environment (QA, UAT & PPE)**

* **Hardware Requirements**: Specify the hardware needed for testing, including different devices for compatibility testing.
* **Software Requirements**: List the software, including operating systems, browsers, and any other tools required for testing.
* **Test Data**: Define the data required for testing, including any specific scenarios or edge cases.

**5. Test Tools**

* **Automation Tools**: Use tools like Selenium, Rest-Assured with Cucumber for automated functional testing.

**\*\*** Existing Cypress automation suite to be migrated to Cucumber Java to align it with LSEG unified framework

**\*\*** Axios API automation suite to be migrated to Cucumber Java

* **Performance Testing Tools**: JMeter or other performance tool to be user to assess performance.
* **Bug Tracking Tools**: JIRA/ASANA
* **Test Management Tools**

**6. Test Deliverables**

* **Test Plan**: A detailed plan outlining the testing activities, schedule, and resources.
* **Test Cases**: Documented test cases covering all aspects of the UI transformation.
* **Test Scripts**: Automated test scripts for functional and NFR’s.
* **Test Reports**: Reports on test execution, including pass/fail status, defects, and overall quality assessment.
* **Defect Reports**: Detailed documentation of identified defects, including steps to reproduce, severity, and status.

**7. Roles and Responsibilities**

* **Test Manager**: Oversee the testing process, ensure resources are allocated, and manage the testing schedule.
* **Test Lead**: Lead the testing team, coordinate testing activities, and ensure adherence to the test plan.
* **Test Engineers**: Execute test cases, report defects, and collaborate with developers to resolve issues.
* **Developers**: Fix identified defects and collaborate with testers to ensure quality.
* **Business Analysts**: Provide requirements, clarify doubts, and validate that the UI meets business needs.

**8. Schedule**

* **Milestones**: Define key milestones and deadlines, including the start and end of each testing phase.
* **Test Phases**: Outline the phases of testing (e.g., unit testing, integration testing, system testing, UAT) and their timelines.

**9. Risk Management**

* **Risk Identification**: Identify potential risks that could impact the testing process or the quality of the UI.
  + **Example**: Delays in development, lack of test data, or insufficient test environments.
* **Risk Mitigation**: Define strategies to mitigate identified risks, such as additional testing, contingency plans, or resource allocation.
  + **Example**: Allocate additional resources to address potential delays or create backup test environments.
* **Contingency Plan**: Plan for handling unforeseen issues, including backup resources and alternative testing approaches.
  + **Example**: Have a backup test environment ready in case the primary environment becomes unavailable.

**10. Entry and Exit Criteria**

* **Entry Criteria**: Conditions that must be met to start testing, such as completion of development, availability of test environments, and readiness of test data.
* **Exit Criteria**: Conditions that must be met to conclude testing, such as completion of all test cases, resolution of critical defects, and approval from stakeholders.

**11. Communication Plan**

* **Status Meetings**: Schedule regular meetings to discuss progress, issues, and next steps.
* **Reporting**: Define the format and frequency of test reports, including daily or weekly status updates, defect reports, and final test summary.
  + **Example**: Weekly status reports summarizing test execution, defects, and overall progress.