Test Plan for Trello Automation Project

Project Name: Trello Automation

Prepared By: Necip Arikan

Date: 06.11.2024
Version: 1.1

1. INTRODUCTION

This test plan outlines the strategy, objectives, resources, and scope of testing for the Trello Automation Project. The objective of this project is to automate the testing of the Trello web application using SpecFlow and Selenium for both UI and API tests. This document serves as a guideline for the testing process and helps ensure that the application functions as intended.

2. SCOPE

The scope of testing includes:

- Functional Testing: All major functionalities of the Trello application, including user login/logout, board creation and management, card creation, updating, and deletion (CRUD operations), and API interactions.

- Integration Testing: Verification of the integration between the UI and API components.

- Regression Testing: Retesting existing functionalities after updates to ensure no regressions occur.

- Negative Testing: Validation of the application's behavior with invalid inputs.

3. QUALITY OBJECTIVES

3.1 Primary Objectives

- Ensure all functionalities of the Trello application operate as intended.

- Automate key test scenarios to improve testing efficiency and coverage.

3.2 Secondary Objectives

- Identify and address any defects prior to production deployment.

- Foster collaboration among team members through effective communication and documentation of test processes.

4. TEST APPROACH

The testing approach will be a combination of manual and automated testing strategies, focusing on both UI and API layers.

4.1 Test Automation

Automated tests will be implemented using SpecFlow for BDD and Selenium for UI interactions, and tests will be conducted using NUnit as the testing framework.

5. ROLES AND RESPONSIBILITIES

- Test Architect: Responsible for the overall test strategy and design.

- QA Engineers: Execute test cases, report defects, and ensure quality standards are met.

- Developers: Collaborate with QA to fix defects and ensure application stability.

6. ENTRY AND EXIT CRITERIA

6.1 Entry Criteria

- Test environment is set up and accessible.

- Application builds are stable and deployed in the testing environment.

- Test cases are written and reviewed.

6.2 Exit Criteria

- All critical test cases have been executed with results documented.

- Defects identified have been addressed or documented for future sprints.

- All stakeholders have signed off on test completion.

7. SUSPENSION CRITERIA AND RESUMPTION REQUIREMENTS

7.1 Suspension Criteria

- Major defects discovered that hinder further testing.

- Unavailability of necessary resources or environments.

7.2 Resumption Criteria

- Resolution of critical defects.

- Restoration of testing environment and resources.

8. TEST STRATEGY

8.1 QA Role in Test Process

QA is responsible for designing test cases, executing tests, and ensuring that the application meets quality standards throughout the development lifecycle.

8.2 Bug Life Cycle

- Identification: Bugs are reported by QA or users.

- Assignment: Bugs are assigned to developers for resolution.

- Fixing: Developers address bugs and submit fixes.

- Verification: QA verifies the fixes and retests as necessary.

8.3 Testing Types

- Functional Testing

- Regression Testing

- Integration Testing

- Load/Performance Testing (if applicable)

8.4 Bug Severity and Priority Definition

Severity List:

- Critical: System crash or data loss.

- Major: Significant functionality failure or usability issues.

- Minor: Minor functionality issues or cosmetic defects.

Priority List:

- High: Must be fixed before release.

- Medium: Will be fixed in the next release.

- Low: Future consideration.

9. RESOURCE AND ENVIRONMENT NEEDS

9.1 Testing Tools

- SpecFlow for BDD

- Selenium WebDriver for UI Automation

- NUnit for test execution and reporting

- Serilog for logging

9.2 Configuration Management

- Use of `appsettings.json` for configuration management, ensuring sensitive information is secure.

9.3 Test Environment

- A dedicated environment mirroring production setup for accurate testing results.

10. TEST SCHEDULE

- Test Planning: [Start Date] to [End Date]

- Test Execution: [Start Date] to [End Date]

- Test Reporting: [Date]

11. TEST CASES

11.1 UI Test Cases

1. Create Board

Positive Cases:

- Create a board with a valid name "Test".

- Verify that the board "Test" appears in the boards list after creation.

Negative Cases:

- Attempt to create a board without entering a name and verify the error message.

- Attempt to create a board with only spaces as the name and verify the error message.

Edge Cases:

- Create a board with an extremely long name (e.g., 1000 characters).

- Create a board with a name that includes emojis or other Unicode characters.

2. Card Management on Created Board

Positive Cases:

- Create a card named "Card 1" on the board "Test".

- Update the card "Card 1" to "Updated Card 1" and verify the change.

- Delete the card "Updated Card 1" and verify it is no longer in the list.

Negative Cases:

- Attempt to create a card without providing a name and verify the error message.

- Attempt to update a card with invalid data (e.g., an empty string) and verify the error message.

- Attempt to delete a card that does not exist and verify the error message.

Edge Cases:

- Create multiple cards in quick succession and validate that all are created properly.

- Attempt to update a card while the board is deleted and verify error handling.

11.2 API Test Cases

1. Create Board via API

Positive Cases:

- Successfully create a board named "Test Board from API" via API.

- Verify that the newly created board can be retrieved correctly using its ID.

Negative Cases:

- Attempt to create a board with an invalid API key/token and verify the error message.

- Attempt to create a board with an invalid payload and verify the error response.

Edge Cases:

- Create a board via API while exceeding the rate limit and verify the error handling.

- Attempt to create a board when the API service is down and ensure proper error messaging.

2. Card Management via API

Positive Cases:

- Create a card named "API Card 1" on the board "Test Board from API" and verify the response.

- Retrieve the list of cards for the board and verify "API Card 1" is present.

Negative Cases:

- Attempt to create a card without providing a name and verify the error message.

- Attempt to retrieve a card using an invalid ID and verify the error message.

Edge Cases:

- Create a card with a very long name (e.g., 256 characters) and check for response handling.

- Attempt to update a card while the board it belongs to is deleted and verify error handling.