SOEN390 - Testing Plan for Sprint #3

1. Testing approach

During the development of the system, we will be writing test in all levels:
  i.   Unit tests
  ii. Integration tests
  iii. System tests

i.   Unit testing

Unit testing refers to the smallest testable part of a software. The objective of unit testing is to verify that each unit of the software performs as designed.

ii.   Integration testing

The goal of integration testing is to test multiple components of a system together, focusing on the interactions between them instead of testing the system.

iii.   System Testing
System testing is a level of software testing that evaluates the complete and integrated software system. The purpose of system testing is to verify that the entire system functions according to the specified requirements.

To execute these tests, we must first create a form of automation. In our case, a Continuous Integration (CI) pipeline will be used to trigger automatically upon new commits or merges to the codebase. This will help fetch the latest code, builds the software, run automated tests at various levels (unit, integration, system), and generate reports on test results and code coverage. Examples of CI tools are Gitlab CI/CD, Azure DevOps, Github Actions, Jenkins, and Travis CI.

We will adopt a comprehensive testing approach that includes both automated unit testing and manual acceptance testing. Unit tests will focus on validating individual components and functionalities of the software, while acceptance tests will ensure that the overall system meets the specified requirements and user expectations. To determine which tool suits our project the best, we will be trying automated unit testing and manual acceptance testing on Travis CI, Jenkins and Github Actions in sprint #3.

2. Testing tool

<u>Best C# testing frameworks research (conducted in Sprint#1):</u>

Adapted from JUnit, NUnit is an open-source unit testing framework for the .NET framework, and it is widely used in C# development. This framework would allow us to write and execute unit tests in a structured and organized manner. It also has fast testing execution speeds. While NUnit does have documentation and resources available, it may not be as beginner friendly as other other testing frameworks for team members that haven't used it in past projects.

xUnit.Net is an open-source testing framework that is based on the .NET framework. It is popular for its intuitive terminology and structure for writing test, making it easy for developers to understand and use. While xUnit.net has documentation and resources available, it may not be as extensive or well-established as NUnit.

MSTest (Microsoft Unit testing framework for .NET) is a unit testing framework developed by Microsoft for testing .NET applications. It is tightly integrated with Visual Studio and offers simplified test creation, execution and debugging. Since MSTest is included with Visual Studio, there's no need to install additional packages or frameworks to start writing and executing tests. However, since it is primarily designed for Windows, it has limited cross-platform support. Furthermore, MSTest may be less extensible and more challenging to integrate with other tools or extend its functionality.

<u>Choice of testing frameworks</u>

The testing for the backend specifically for sprint #2 was done using xUnit.net for multiple reasons:

    i.   **Integration with Visual Studio:**
                xUnit.Net seamlessly integrates with Visual Studio, providing a familiar and efficient testing environment for team members. This integration streamlines the testing workflow, allowing us to

xUnit.net

write, run, and debug tests within the same IDE we use for development.

ii. **Rich Feature Set:**
xUnit.Net offers a rich feature set for unit testing, including support for parameterized tests, test categorization, and extensibility through custom assertions and test runners. These features would allow us to write expressive and comprehensive tests that a multitude of scenarios.

iii. **Clear and Readable Syntax:**
The xUnit.Net framework provides a clear and readable syntax for writing tests, making it easier for developers to understand the test cases, identify failures, and maintain tests over time. This clarity enhances the maintainability and comprehensibility of the test suite, facilitating collaboration among our team.

xUnit.Net will continue to be used to ensure the quality and reliability of backend code during future sprints.

As for the front-end, unit testing has been applied to all pages.

3. Testing Metrics and Coverage

We aim to achieve 80% code coverage for both front-end and back-end the end of Sprint #3. Test coverage will be measured using Visual Studio's built-in code coverage analysis tools. In addition to test coverage, we will track the following metrics to assess the overall quality of our software:

i. **Code Duplication:**
We will monitor the percentage of duplicated code within our database using tools such as Visual Studio's Code Clone Analysis. By minimizing code duplication, we aim to improve maintainability and reduce the risk of introducing bugs.

ii. **Cyclomatic Complexity:**
We will measure the cyclomatic complexity of our methods and classes using a tool called ReSharper. High cyclomatic complexity can indicate

code that is difficult to understand and maintain, increasing the likelihood of defects.

  iii. **Coupling:**
We will assess the level of coupling between modules and components in our system using a static code analysis tool like ReSharper. High coupling can lead to increased dependencies and fragility, making our codebase more prone to defects and harder to refactor.

In addition to these metrics, we will also monitor the following aspects of our testing process:

  iv. **Defect Density:**
We will track the number of defects identified during testing per thousand lines of code (KLOC) to measure the effectiveness of our testing efforts and identify areas of the codebase that require additional testing coverage.

  v. **Test Execution Time:**
We will monitor the time taken to execute our automated test suite using xUnit.Net. By identifying any performance inefficiencies in our testing process, we can optimize our testing strategy for the following sprints to ensure timely feedback on code changes.

4. Acceptance tests

For sprint #3, acceptance tests will be based on the acceptance criteria outlined in the user stories that were updated during sprint #2. These criteria define the specific behaviors and functionalities that must be present in the software to meet user expectations. Once these acceptance tests are passed, we will be able to confidently assert that the software meets the defined requirements and is ready for release.

To ensure consistency and clarity in our acceptance test approach, we will be using the following user acceptance testing template offered by Jira, a platform significantly used by our team to track our progress throughout the development cycle.

1. Summary: A brief overview of the test scenario or objective.

2. Description: Detailed information about the test scenario, including relevant context or background.

3. Test steps: Sequential steps to be executed to verify the functionality or behavior under test.

4. Test Case ID: A unique identifier for tracking and referencing the test case.

5. Status: Indicates whether the test case is currently in progress, completed, or failed.

6. Expected Results: Expected outcomes or behaviors that indicate the successful completion of the test.

7. Actual Results: The actual outcomes observed during test execution, which will be compared against the expected results to determine the test outcome.

In addition to these components, we will be adding a component of our own:

8. Pass/Fail Criteria: Criteria for determining whether the test has passed or failed based on the comparison of actual and expected results.

By following this template, we ensure comprehensive coverage of all relevant aspects of the testing process, facilitating clear communication and documentation of test scenarios and outcomes.

Conclusion

This testing plan outlines our approach, tools, metrics, and acceptance testing strategy for sprint #3. By following this plan, we aim to ensure the quality, reliability, and compliance of our software with the specified requirements and user expectations.