SOUTHERN SOFTWARE SOLUTIONS

Software Quality Assurance Plan

for

Class Attendance Software

Release 1.0

Version 0.0 Approved

Prepared by South Software Solution

Insert Contents Here

List of Figures

List of Tables

Revision History

Date	Description	Revised by
3/13/24	Initial Release	South Software Solutions

1 Introduction

1.1 Purpose

The purpose of this document is to provide a comprehensive and detailed guide to the ClassMate software, elucidating its features, functionalities, and the underlying quality assurance methods employed during its development. This document aims to serve as a reference for team members, stakeholders, and other relevant parties involved in the software's conception, design, and ongoing maintenance. By outlining the software's objectives, team structure, testing strategies, and documentation requirements, this document seeks to ensure clarity, transparency, and efficiency in the development and management of the ClassMate software.

1.2 Definitions

- *SQA: Software Quality Assurance*
- *CSV: Comma-Separated Values*
- *PR: Pull Request*
- *Gitea: A web-based platform for collaborative software development, including version control using Git.*
- *QA: Quality Assurance*
- *Agile: An iterative and flexible approach to software development that emphasizes collaboration, adaptability, and customer satisfaction.*
- *User Story: A concise description of a software feature from an end-user perspective, often used in Agile methodologies.*
- *Acceptance Criteria: A set of conditions that must be met to consider a user story or task complete.*
- *Definition of Done: A checklist of criteria that a software increment must meet to be considered complete and ready for release.*

1.3 References

Outsourced documents or supporting references to this document

2 Software Overview

2.1 Description

This class attendance software is a solution with the goal of simplifying the management and tracking of student attendance in educational institutions. It offers faculty members, administrators, and students a user-friendly interface to efficiently handle tasks related to attendance. By automating attendance related tasks such as student enrollment, attendance recording, and reporting, the software's goal is to reduce the workload that is put on faculty and administrators. The key features of this software include large student enrollment, automated email notification sent for password management, generation of random attendance codes, and reporting capabilities for faculty. The user-friendly interface allows for smoother communication between stakeholders and makes attendance monitoring efficient.

2.2 Features and Functionality

The ClassMate software offers a set of features and functionalities to support students, faculty, and administrators in effectively managing and accessing attendance.

- On-boarding: Faculty members can onboard students in large quantities by uploading CSV files containing the student information or uploading student information individually.
- Temporary Password Upon Initiation: Students receive a temporary password through email after being added to the system.
- Student-Centered Attendance: The application allows students to track their own attendance by entering a randomly generated 6-digit code that is provided by the faculty.
- Attendance Evaluation: Faculty can observe attendance reports, including printing class room attendance reports for specific days, and observing individual student attendance records.
- Classroom Management: Faculty has control over classroom management tasks such as adding, editing, or deleting students and attendance records.
- Student Information Access: Upon entering their account, students have the ability to enter a class-specific code for tracking their attendance for the day as well as seeing a summary of their attendance.
- Password Reset: All users have access to an account recovery system, allowing them to reset their passwords through an email verification.
- Admin Management: Administrators have management capabilities, including creating, editing, or deleting faculty accounts, student accounts, and courses.

3 Quality Assurance Method

3.1 Description

This section helps to define and describe the essential methods that benefit our software quality assurance (SQA) process. By outlining the structure and processes to our SQA methods, we provide our internal members with information on how to ensure proper functionality, integrity, and overall quality of our application. This section outlines SQA through our team structure, an individual member's process during our steps of SQA, and the processes that support our reviews and approvals of tasks.

3.2 Quality Assurance Team Structure

How does quality assurance fit into the team? What is the flow of quality assurance in the team? When do specific quality assurance events occur during the sprint?

Throughout our team, we use an agile development framework that helps us define our SQA events seen throughout our sprint cycles. In this section, different sections and subsections of the collective team will be used to outline when specific SQA events take place.

3.2.1 Team Structure

The South Software Solutions Team consists of 15 members that are divided up into the following sub teams:

- 5 members in leadership/configuration management
    - 1 team leader
    - 2 model/controller leaders
    - 2 view leaders
- 3 agile sub teams that consist of 4 group members (two of the leadership/configuration management team members are also part of one of these teams).

It is important to understand the team structure so that further elaboration on flow of quality assurance within the team can be established.

3.2.2 Team Quality Assurance Events

Specific SQA events will be defined by utilizing the previously defined team structure.

A *peer review* occurs after an individual member finishes an iteration of construction and is about to create a PR. This occurs before a PR is even made. A peer review occurs within sub teams by a different team member.

A *pull request* (PR) occurs after a peer review is approved and the individual member submits a PR through Gitea.

A *pull request test* occurs when a PR is submitted and the testing phase begins. A pull request test is done by members on a separate sub team in a rotating fashion. One example of this rotation is team 1 will test team 2's submitted PR's, team 2 will test team 3's PR's, and team 3 will test team 1's PR's.

A *pull request approval* occurs after a pull request test is approved by the tester. A pull request approval is done by any of the 5 leadership/configuration management team members. After a PR is approved, it is merged into the respective repository in Gitea.

This process is iterative and can be cycled for each sprint that takes place. This allows for a rapid development process while still assuring quality within our software.

3.3 Individual Quality Assurance Events

It is important to outline what a set of SQA events looks like for an individual from each viewpoint in order to give a reference for team members. The following steps are the steps that are carried out by each team member to complete a proper quality assurance iteration:

For all team members, the following events occur in order:

1. Receive a task on Gitea.
2. Assure the description of the task in Gitea includes a user story, a list of acceptance criteria, and a final "definition of done."
3. Write a list of test scenarios for the task below the user story, the list of acceptance criteria, and the "definition of done" in Gitea.
4. Construct your task (this holds the sub tasks of documenting your code design, implementing and documenting your implementation, self testing, and refactoring). All construction tasks should be located in a single test plan document that can be passed to a peer reviewer.
5. Ask for a peer review. Peer reviews are conceived as soon as a construction interaction of the task is completed. The peer reviewer will then respond to the member who is constructing the task whether or not the task passed. Based on this, you either continue construction or submit a PR.

6. Make a PR. A PR should include all documentation created (construction documentation and peer review documentation).
7. Test a PR. PR testing occurs during the testing phase of the sprint. No more PR's should be created for that sprint unless under certain circumstances.
8. Add tasks to the final sprint document. Your entry into the final sprint document should include a before and after demonstration of your task regarding the application.
9. Create a user guide. This will be constructed after a PR is approved to the production source code and is ready to be used.

Team leaders hold the specific tasks of approving PR's. A PR approval leads to it being merged into the respective branch. A closed PR leads to another construction iteration by the respective member.

3.4 Review and Approval Process

It is important to include the specific steps of all review and approval processes to give a reference for SQA events and tasks. The following descriptions include the specifics of a peer review, a PR test, and a PR approval.

For a peer review, a document should be received that includes the task description, a user story, the acceptance criteria, and the test scenarios for the task. A code review should then take place checking for styling and proper conventions. An adjacent sub-team member will set up their testing environment for the received peer review and test every acceptance criteria for every test scenario. A test report document will be passed to the member who requested the peer review and their following actions will depend on if the peer review passed or not.

For a pull request test, a document should be received that includes the task description, a user story, the acceptance criteria, the test scenario, and a list of test cases for the task. A code review should then take place checking for styling and proper conventions. A team member will set up their testing environment for the received pull request test and test every acceptance criteria for every test scenario (using the test cases if needed). A test report document will be passed to the member who submitted the peer review and the status of the pull request and the actions of the team member who submitted said pull request depend on if the pull request test passed or not.

For a pull request approval, all documentation should be included in the pull request in order for a team leader to test and approve/close the respective PR. A team leader will test all of the acceptance criteria for every test scenario and do a code review checking for styling and proper conventions. A test report document will be passed to the member who requested the peer review and their following actions will depend on if the PR was approved or closed.

4 Testing Strategy

4.1 Types of Testing

The types of testing that will occur during a QA session will be ad-hoc tests. Sub-teams will test out their code to verify that any added features are implemented and working correctly.

4.2 Testing Execution Process

The testing execution process will have the following timeline:

- A peer review will happen immediately after a sub-team member completes their first iteration of coding, and will happen by one of their fellow members.
- A PR review will happen after a peer review gets accepted, and will be verified by all members of the sub-team as working.
- A PR approval will happen when a team lead accepts the PR, after verifying it themselves and then merging the code.

4.3 Test Environment Setup

Since we will be using Gitea for code repositories and issue tracking, team members will be able to pull from the member's forked repository to test it out themselves.

5 Documentation and Reporting

5.1 Required Documentation

This section identifies the documentation required governing the requirements, development, verification, validation, and maintenance of the software that falls within the scope of the software quality assurance plan. The documentation required is as follows:

- *Software Quality Assurance Plan*
- *Configuration Management Plan*
- *Software Requirements Specification*
- *Test Plans*
- *Software User's Guide*
- *Test Reports*
- *Software Requirements Traceability Matrix*
- *Peer Review Data*
- *Training Documentation*

5.2 Reporting Process

Documentation shall be created by relevant stakeholders such as the development team, and change control team. Each document shall be reviewed and approved by the stakeholders to ensure accuracy and completeness. The documentation should be stored in a version control system (e.g. Gitea) apart from source code and other project files. The documentation shall be used by the development team to build the software according to the specifications outlined in the documents. Throughout the development process, stakeholders may provide feedback on the documentation, thus resulting in revisions to improve clarity or accuracy. Revisions shall be made as needed and distributed to the stakeholders. Once the software development is complete, the documentation shall be archived for future reference.

5.3 Issue and Error Reporting

The process for a member to report an issue about the application are as follows:

1. *Identify the issue - A member shall identify the problem and its symptoms. First try to reproduce the issue and document the steps taken, error messages*

received, and expected versus actual outcomes. A member should try to narrow down the possible causes and solutions.

2. *Troubleshoot the issue - Before a member reports an issue, they shall try to troubleshoot to see if they can resolve it on their own. Common methods should include restarting the system, checking network connections, or searching for help. If the member manages to fix the issue, they shall report it and include an explanation of what they did and why.*
3. *Report the issue to team members - A member shall use means of what the team provides such as email or chat to report the issue internally. Relevant information such as the problem description, troubleshooting steps, screenshots, and logs shall be provided.*
4. *Track and update the issue - After a member has reported the issue, they shall track and update until it gets resolved using an issue tracking system (e.g. Gitea). Status of the issue shall be monitored and any changes shall be communicated and documented. This will include questions regarding the issue and any feedback that may be necessary.*
5. *Issue resolution - Once an issue is resolved, documentation of the resolution shall be documented along with how it was resolved, what caused the issue, and the lessons learned from the issue.*
6. *Prevention - The final step shall be to prevent similar issues in the future. Best practices should be implemented based on the issue resolution and analysis of the root cause.*