Software Testing Document for Scholarli

College of Engineering and Computer Science

Florida Atlantic University

CEN4010 – Principles of Software Engineering

Dr. Safak Kayikci

Prepared By: Juan Rodriguez, Kevin Tran, Thau Tran-Nguyen

Table of Contents

1. Test Plan Identifier

The test plan identifier: SCHT-TPD-1.0.0-20240727

SCHT: This abbreviation represents the project name “Scholarli Tracker”

TPD: This abbreviation represents the Test Plan Document

The numbers for the version are as follow:

Major Version (1.x.x): This represents a major revision in the project.

Minor Version (x.1.x): This represents a minor revision in the project.

Patch Version (x.x.1): This represents a patch in the project, usually are bug fixes.

Indicated from the version, this is a master plan for Scholarli. Following the version number, the date of the document.

1.2 Revisions History

Name	Date	Version
Thau Tran-Nguyen	7/27/2024	1.0
Juan Rodriguez	7/27/2024	1.0
Kevin Tran	7/27/2024	1.0

2. Introduction

2.1 Purpose

The purpose of this Software Test Document (STD) is to provide and outline the architecture and system design for Scholarli development, which is a semester-based student assignment tracker. This document serves as a guide for the development team, users and investors involved in the project.

2.2 Scope of the System to be Tested

The scope of the testing encompasses various aspects of Scholarli. This ranges from the user interface, backend process, database interactions, security and performance benchmarks.

2.3 Overview

This document provides an overview of Scholarli's system architecture, data design, component design and human interface design. This can be used for the development and investors to understand the technical design behind Scholarli. Users can use this document in the future to help understand and communicate changes with the development team

2.4 References from SDD

https://github.com/KatherineSantore/AssignmentTracker_Client

https://www.researchgate.net/figure/High-level-software-architecture-of-DSMS_fig5_332656904

3. Test Item

This section will outline the specific features and components of Scholarli that will be tested.

3.1 User interface

- Menus
    - Ensure that all menus and buttons are functional and accessible across multiple screens.
- Forms
    - Testing data validation and valid submissions from users in all input forms (login, registration, application events)
- Design
    - Verify that the application has a consistent look and layout across many devices

3.2 User registration

- Registration
    - Test that the process of registration is fully functional
- Login
    - Ensure that users can log in when CORRECT credentials are input and incorrect information is rejected
- Password
    - Ensure that if the user is unable to recall the password the password recovery process is functional and secure.

3.3 User account

- Profile update
    - Ensure that users can update their profile information (email and password)
- Deletion
    - Test that the account deletion process is functional and that all the user's data is fully removed

3.4 Semester Management

- Create
    - Ensure that the process to create new semesters is functional and is handling data correctly
- Editing
    - Verify that existing semesters can be updated and handles new information correctly
- Deletion
    - Test deletion process to ensure data is properly removed

3.5 course management

- Course manipulation (creation, editing, deletion)
    - Verify that all the operations can be completed and are functional

3.6 assignment management

- Assignment events
    - Ensure that the lifecycle of the assignment (notification and scheduling) is working correctly

3.7 Notifications

- Notification systems
    - Test that the notification system is operating correctly and is sending out the correct data

3.8 Calendar

- Calendar event creation
    - Ensure that the assignments are correctly added to the user's calendar
- Reminders
    - Verify that the user can edit and receive the reminders for upcoming assignments

3.9 Performance and security

- Load testing
    - Verify that the system can handle various loads to ensure a stable application
- Security
    - Test for vulnerabilities in user authentication, data storage, and notification data.

3.10 cross platform compatibility

- Ensure platform is functional across all platforms (IOS, android, web)

4. Features to be Tested

User registration:

Users need to be able to complete registration for the app. This includes an email address and a password creation.

User login:

Users need to be able to login to the app.

User account:

Users need to be able to manage their account. This includes an email or password change. This also includes account deletion.

Semester Management:

Users will be able to manage their semesters. This includes the creation, editing and deletion of their semesters.

Course Management:

Users will be able to manage their courses. This includes the creation, editing and deletion of their courses.

Assignment Management:

Users will be able to manage their assignments. This includes the creation, editing and deletion of their assignments.

Notifications:

Users need to receive notifications about their assignments.

Calendar:

Users need to be able to interact with their calendar.

5. Features not to be tested; ****

Availability: Most of the user data is stored locally, so the availability might not need testing before the initial release.

6. Approach

For this project, we can use a combination of testing approaches. Some items are common, so automatic testing can be used.

Login and registration can use automated testing tools like Espresso for Android, XCTest for iOS and Selenium for a web interface. Automated testing can be used for regression testing to ensure that the new changes do not break existing functionality. As well as performance testing for the app.

Sequential testing can be used for more user case tests. Human QA testers can verify the UI/UX components for functionality and accessibility. QA testers can test the function of the buttons, swipe motions, creation and deletion of various tasks

Interface testing: The interface testing could be tested using automated and sequential testing. Scripts and automated testing tools can be used for interface testing. QA testers can also test the interface sequentially. They can test the functions one-by-one until each function passes the pass/fail test.

Performance testing: QA testers can test the performance of the app under different loads. The most the important performance testing needed is the performance testing of the database in the app.

7. Item pass/fail Criteria

7.1 Functional Requirements

Correctness

- Pass: The output matches the expected results as defined in the test cases. This includes accurate data handling, correct functioning of components, and proper error handling.
- Fail: The output does not match the expected results and there are errors in data handling, components that malfunction, or improper error handling.

Completeness

- Pass: All components are fully operational and meet the requirements defined in the specification document. No partial functionality is allowed for passing.
- Fail: Any component that is partially functioning or does not meet the specified requirements will not pass the tests.

Usability

- Pass: The application is accessible and easy to use for all users. User Interface components are fully functional, intuitive, and provide a seamless user experience.
- Fail: The application is difficult to use, with non-functional or confusing User Interface components that hinder the user experience.

Interoperability

- Pass: The application interfaces correctly with external systems and APIs that are specified.
- Fail: The application fails to correctly interface with external systems or APIs leading to inconsistencies or errors.

7.2 Non-functional Requirements

Performance

- Pass: The application provides excellent response time, load capacity, and scalability. Performance measures meet the predefined thresholds under normal and high stress conditions.
- Fail: The application fails to meet the performance benchmarks, exhibiting slow response times, poor load handling, or scalability issues.

Security

- Pass: The system ensures secure handling of user data, incorporating secure login processes and data encryption. All security protocols and measures are robust and effective.
- Fail: The system has vulnerabilities in handling user data, insecure login processes, or inadequate data encryption which lead to security breaches of data and potential instability.

Compatibility

- Pass: The application ensures consistency across multiple devices and platforms such as iOS, Android, or web browsers. It performs uniformly on all supported platforms.
- Fail: The application shows inconsistencies or performance issues across different devices and platforms, failing to provide a uniform experience for users.

Reliability

- Pass: The application is stable and has minimal downtime. It consistently performs its intended functions without fail.
- Fail: The application is unstable, frequently crashes when used, or experiences a significant amount of downtime affecting the reliability.

Scalability

- Pass: The application can handle increased load effortlessly, without showing signs of degradation in performance.
- Fail: The application experiences performance issues or fails when handling increased load.

Maintainability

- Pass: The application code is well-documented, modular, and easy to maintain or extend. Updates and bug fixes can be implemented without significant difficulty in doing so.

- Fail: The application code is poorly documented and difficult to maintain or extend, making updates and bug fixes challenging.

Localization

- Pass: The application supports multiple languages and regional settings, providing a consistent user experience across different locations.
- Fail: The application fails to support multiple languages or regional settings, leading to inconsistencies in user experience.

8. Test Deliverables

- Test Plan Document: This document will outline the strategies we will use to test components.
- Test cases: description of individual test cases. Includes expected results and criteria for passing and failing.
- Performance test: Data and comments on the performance tests
- Security test: documentation and comments on security testing results
- Test Execution report: documentation on results and possible defects in components
- Defect report: Documentation and comments on defects that occurred in the tests

9. Environmental Needs

9.1 Hardware

Scholarli needs to be tested on different devices. An Android phone, iOS phone and an operating system with a web browser is needed.

9.2 Software

9.2.1 Operating Systems

Since this app is cross platform, we will need to use a variety of devices for testing. This will include Microsoft Windows, Apple iOS and Android.

9.3 Documents

The following documents are required for software to support testing.

- Scholarli – Software Requirements Specifications
- Scholarli – Software Design Document

10. Roles and Responsibilities

Project Lead: This role would lead the project and create tasks for each developer. They will also receive and interpret the progress reports of the project.

UX/UI Designer: This role would create the user interface for the app. It requires a greater understanding of user-friendly designs and accessibility designs for mobile and web apps.

Front-end Developer: This role would work with the UX/UI designer to add functionality to the UX/UI designer's work.

Back-end Developer: This role would work with the front-end developer to ensure that the back-end functionality, like databases, work accordingly.

QA Tester: This role tests the released project and report the bugs and feedback to their respective developer or designer.

11. Staffing and Trainings Needs

11.1 Staffing Requirements

The team should be a combination of the various roles listed above. Each member should be hired accordingly to fulfill their responsibilities in the project. The required skill set for each role is listed below in the skillsets section.

Skillsets: Technical Skills

Programming Languages: Java, Kotlin, Swift, Objective-C, JavaScript

Mobile Development Framework: Android Studio, Xcode, Flutter, React Native

UI/UX: Sketch, Figma

Database Management: SQL

QA Testing: JUnit, Espresso for Android, XCTest for iOS, Selenium

Project Management: Agile, Jira, Trello, GitHub

12. Schedule

Shown below are the milestones that will be reached by different weeks. This will help guide the development team in the project.

Week 1-2 Milestones

- Define project scope and set goals.
- Review SRS and SDD.
- UI/UX Designer begins wireframing.
- Frontend Developer sets up project structure.
- Backend design a database schema.
- UI/UX presents and revises wireframe based on feedback.

Week 3-6 Milestones:

- Front-end developers integrate with UI components.
- Backend developers create the core functionalities.
- UI/UX prepares app for initial testing.
- Front-end and back-end integrate functionalities.
- QA tester starts writing test cases.
- UI/UX finalize all design elements.
- QA testers being testing and identifying bugs.
- QA testers report bugs to the team for fixing.
- Finalize phase 1 and prepare for phase 2 of development.

Week 7-10 Milestones

- Continued development and testing.
- UI/UX designer reviews and refine user experience.
- QA testers report bugs to the team for fixing.
- Assess progress of the app development.
- Final integration testing.
- Address final feedback and prepare for final testing phase.
- Finalize all features and prepare for deployment.

Week 11-12 Milestones

- Deploy project.
- Review user feedback for bug fixes.