Software Test Plan:

CSUN Dashboard

Prepared by: Joe, Aya, Dylan, Tigarn, Nora, Mubeen

1.0 INTRODUCTION

The design of the app is focussed on simplicity, accessibility, and functionality. Users may create and name key tasks, as well as provide descriptions and set completion deadlines. Subtasks with particular details help to further break down bigger goals into manageable components. A user-friendly calendar highlights task timelines visually, enabling effective time management. The app's navigation is simple, with sections dedicated to primary chores, a calendar. Users may effortlessly navigate between different parts, maintaining a consistent user experience. A potential point system adds a gamified element, encouraging rivalry and collaboration, assuming future internet access.Our app makes it easier to manage bigger life goals and their related subtasks by giving users the tools they need to efficiently define, track, and achieve goals. The features that we have implemented into our app is a main goal page where all the goals for a specific date are listed, there's also a page for sub-tasks, and finally a scheduling page which displays future tasks.

2.0 OBJECTIVES AND TASKS

2.1 Objectives

1. **Setting Test Objectives and Scope**:Test Plan articulates the objectives of testing, including what parts of the software will be tested and to what level of detail. It defines the scope of testing to ensure that all relevant functionalities and features are covered.
2. **Defining Tasks and Responsibilities**: The Test Plan outlines specific tasks to be performed during testing and assigns responsibilities to team members. It clearly defines who is responsible for what aspect of testing, ensuring accountability and clarity within the team.
3. **Documenting Service Level Agreements (SLAs)**: Some cases, the Test Plan may include SLAs that briefly describe the level of service expected from the testing team, such as response times for addressing reported issues or resolving defects. These SLAs serve as a commitment to quality and performance standards

2.2 Task

1. **Functionality Testing Tasks**:
    a. Add a new task.
    b. Edit an existing task.
    c. Delete a task.
    d. Mark a task as completed.
    e. Categorize tasks into different categories.
    f. Set due dates for tasks.
    g. Set reminders for tasks.
    h. Sort tasks by different criteria (e.g., date, priority).
    i. Undo/redo actions in the task list.
2. **Integration Testing Tasks**:
    a. Test synchronization of tasks across all Android devices (e.g., mobile app).
    b. Verify integration with third-party services (e.g., calendar sync).
    c. Test handling of notifications for reminders and updates.
    d. Ensure proper integration with other internal modules or components of the app.
3. **Post-Testing Tasks**:
    a. Document test results, including any issues or bugs found during testing.
    b. Prioritize and assign severity levels to reported issues.
    c. Verify fixes for reported issues through regression testing.
    d. Conduct final acceptance testing before release.

3.0 SCOPE

General

The Software Requirements specification document for “Goal” application, developed by the CSUN group, known as “Chat JTDAN”, delineates the creation of a software solution designed to facilitate efficient task organization and management. The primary aim of this application is to empower users by streamlining their workflow, augmenting productivity, and ensure effective task tracking and completion. To make sure the features mentioned work we will be testing for following functionalities.

1. **Functionality Testing**: This involves testing each feature or function of our app to verify that it performs as expected. This would include tasks such as adding, editing, deleting goals, and categorizing tasks. It also involves checking for any bugs or glitches in the functionality.
2. **Integration Testing**: This involves testing how different components or modules of the app work together. For our app, it includes testing how tasks sync on android phones, how notifications are handled, and how the app integrates with other tools or services (e.g. calendar page, individual page).
3. **User Interface (UI) Testing**: This focuses on the visual aspects of the app, ensuring that the interface is intuitive, visually appealing to users. Testers would check things like layout, color schemes

4.0 TESTING STRATEGY

In our approach to testing we're going to use three different methods. Unit testing, Integration testing and finally End to End testing.

1. Unit testing will just involve individual components and/or functions to be tested. This is entirely done with Java Virtual Machine. This will account for about 70% of the testing done.
2. Integration testing will test how components work with one another within the application. This will have to be run in Emulation. This will account for about 20% of the testing done.
3. End to End testing is more comprehensive and tests behaviors that the user will be doing. Such as pushing buttons, creating goals etc. They will be performed both faster and slower than an actual user could actually do them. So if there is a chance the application could be broken, this will test that. This will account for 10% of the testing done.

Unit Testing
Definition:

In order to Unit test locally, they run entirely with a Java virtual machine locally. It is used to test raw java logic. Every function within the applicator class will be tested to make sure they work.

A j-unit dependency will be needed for this in the Android Studio Gradle file.

Participants:

Mubeen will be in charge of writing the test class and executing it.

Methodology:

To run this test, Mubeen will have to create a test class in order to test each individual component of the android app.

Integration Testing
Definition:

This will test android specific functionality. This will test activities, fragments, services etc. These tests will be run in emulation because they need the android framework to work. j-unit will be used for these tests as well as Espresso and Mockkito. Dependencies for those will need to be included in the Gradle.

Participants:

Dylan will be in charge of making the test class.

Methodology:

Scripts will be written to test integration between classes and functions. In addition, it will

also test the database. Dylan will have to run these tests while the app is running in emulation on Android Studio

End to End testing:
Definition:

To simulate what a user might do with the application. Creating goals, editing, deleting, etc. It will also be testing the RecyclerView of the app to make sure it is refreshing.

Participants:

Joe will be in charge of this.

Methodology:

Joe will intentionally create goals, subgoals, editing, deleting. In testing this he will also create goals with various large numbers of characters in an attempt to "break the app." He will also make sure that the RecyclerView also updates after each operation.

5.0 ENVIRONMENT REQUIREMENTS (TOOLS)

The testing will occur inside of the Android Studio application. In order to test we will need to add dependencies to the build gradle:

1) J-Unit
2) Mockkito
3) Espresso

The unit testing can be done locally in android studio, but the Integration testing and the End-to-End testing both need to be run while the application is running (either on hardware or in emulation).

6.0 TEST SCHEDULE

Test milestones for the app will be all of the individual functions present on each page. As the navigation bar is consistent across all of the pages, the tests for that feature of the app will be to ensure that the buttons actually take the user to their corresponding pages - the main page, calendar, individual task, and archive. For the main page, specifically, the tasks must be able to be added, deleted, or completed. The home page will also be tested for the editing of tasks on that particular page – the information should carry over to the individual task page as well. Selecting a task should also take the user to the individual task page, from which the user should be able to perform the aforementioned actions on tasks and additionally edit their contents. In a similar vein to the main page, the archive page should allow the user to access tasks as well as permanently delete them or restore them to be visible on the main page again. Lastly, the calendar page will also be tested for the same functionality of taking the user to the individual task page if a task is selected from a date and also allow tasks to be deleted or edited but not added. The other testing milestones for the calendar are the selection of dates and traversal from month to month on that page. The estimated time required for every one of these tasks is less than a minute and the only tools, facilities, and staff required are a device that can run the app, the app itself, and one user - all of which must be present throughout the whole testing process, which should take approximately 10 minutes to half an hour. Item transmittal events for this application only concern tasks, which are carried over between the main page, archive page, calendar page, and individual task page. Task information for any given task should remain consistent across all pages.

7.0 CONTROL PROCEDURES

Problem Reporting

If an incident is encountered during the testing process, a message must be written describing the problem in detail with a mention of which page or pages the problem occurs on. The message is to be sent to Tigran, our group’s main tester.

Change Requests

Change requests are to be done similarly to problem reports for this app. Tigran will be the primary point of contact for any modifications, but the whole team must agree on the change, likely based on how reasonably implementable the change appears to be and how consistent it is with the original or current concept for the app. In the event that there is a change request, it should be documented in a detailed message with specifications about which pages the page is to be implemented on. That message should be emailed to Tigran as well.

8.0 RESOURCES/ROLES & RESPONSIBILITIES

Aya (Test Manager): Responsible for managing the overall testing process. Coordinates with different teams for test planning, execution, and reporting. Ensures that testing activities are aligned with project objectives and timelines.

Tigran (Test Analyst/Designer): Works closely with the Test Manager to design test cases and scenarios. Analyzes requirements and translates them into test cases for various testing phases. Collaborates with developers and testers to ensure comprehensive test coverage.

Dylan (Test Execution Lead): Leads the execution of test cases and scenarios. Coordinates with testers to ensure timely execution and reporting of test results. Identifies and reports defects, tracks their resolution, and retests fixed issues.

Mubeen (Developer/Tester): Participates in both development and testing activities. Develops automated test scripts and tools to support testing efforts. Performs manual testing, analyzes test results, and collaborates with the testing team.

9.0 RISKS/ASSUMPTIONS

Assumption: Availability of Test Environment

Contingency Plan: Have backup environments ready in case of technical issues or downtime in the primary test environment. Ensure that system administrators are prepared to troubleshoot and resolve any environment-related issues promptly.

Assumption: Timely Delivery of Test Items

Contingency Plan: Maintain close communication with the development team to track progress and anticipate any delays in delivering test items. Prioritize critical test cases and start testing with available items while awaiting the complete set. Consider parallel testing or resource reallocation to speed up testing if necessary.

Assumption: Stable Application Build for Testing

Contingency Plan: Implement version control and rollback procedures to revert to a stable build in case of critical issues or regressions. Maintain a log of changes and updates to track potential causes of instability. Collaborate closely with the development team to address and resolve build-related issues promptly