Team Delta Test Plan

Team Members:

Kalim Dausuel

SUMMER 2024

University of Maryland Global Campus

JUL 14, 2024

Table of Contents

1. Introduction

The ScreenTime widget project aims to develop an Android widget application that helps users monitor and control their screen time effectively. This test plan outlines the comprehensive testing strategy for ensuring the quality, functionality, and user experience of the ScreenTime widget.

The widget includes features such as daily and session-based screen time tracking, a home screen widget for easy visibility, notifications when set time limits are reached, and background operation to ensure accurate tracking. This test plan will cover various aspects of testing to ensure the widget meets its intended functionality and performance goals.

2. GitHub Repository Information

The project's GitHub repository (https://github.com/kdausuel/screentime-capstone) serves as the central hub for code storage and version control.

3. Testing Objectives

The primary objectives of this testing effort are:

1. Verify the accuracy of screen time tracking for both daily and session timers.
2. Ensure proper functionality of the widget UI and its updates.
3. Validate the notification system for time limit alerts.
4. Confirm the persistence of user settings and timer data.
5. Test the widget's performance under various device states and conditions.
6. Verify the proper functioning of the configuration activity.
7. Ensure compatibility with devices running Android 12 (API level 31).
8. Validate the background operations and scheduling of timer resets.

4. Test Strategy

The ScreenTime widget project employs a comprehensive testing strategy that efficiently covers various aspects of the application within the constraints of solo development. For each build of the widget we use JUnit, Mockito, and Robolectric for

automated testing. A single, comprehensive test file (ScreenTimeWidgetUnitTests.kt) contains the automated tests, leveraging the previously mentioned libraries to cover unit tests, integration tests, and aspects of functional and UI testing without requiring a physical device or emulator. The strategy consists of the following types of testing:

1. **Unit Testing:** JUnit, Mockito, and Robolectric are used to test individual components and functions in isolation. This includes testing the DataManager, Timer classes, and utility functions. Robolectric allows these tests to run without a physical Android device or emulator, simulating the Android environment.
2. **Integration Testing:** The test suite includes tests that verify interactions between different components, such as the TimerManager's interaction with the DailyTimer and SessionTimer. Robolectric enables testing of Android-specific components and their interactions without a real device.
3. **Functional Testing:** Automated tests in the ScreenTimeWidgetUnitTests.kt file verify that all features of the widget work as expected, including timer functionality and widget updates. Robolectric facilitates testing of Android framework interactions, such as SharedPreferences and Broadcast Receivers.
4. **User Interface Testing:** While most UI testing is done manually on a physical device, Robolectric allows for some automated testing of UI components and their basic behaviors in a simulated environment.
5. **Performance Testing:** Basic performance checks are included in the automated test suite, with Robolectric providing a simulated environment for testing background operations and resource usage.
6. **Usability Testing:** This is primarily conducted through manual testing on a physical device, as it involves subjective assessment of the user experience.
7. **Regression Testing:** The entire automated test suite, powered by JUnit, Mockito, and Robolectric, runs on every push to the main branch using GitHub Actions. This ensures that after any changes or bug fixes, all existing functionality is re-tested to prevent the introduction of new issues.

5. Test Scope

In Scope:

- Daily and session timer functionality
- Widget UI and updates
- Configuration activity
- Notification system
- Data persistence
- Background operations (timer decrement, resets)
- Screen state management (screen on/off handling)
- Widget addition and removal
- Realistic battery consumption testing

Out of Scope:

- Testing on Android versions below API level 31
- Extensive battery consumption testing
- Localization testing
- Testing with third-party launcher apps

6. Test Cases

To ensure a thorough testing process while maintaining manageability for a single developer, a carefully selected subset of test cases has been chosen. These cases are designed to cover the core functionality, critical user interactions, and potential edge cases of the widget. While not exhaustive, this set of test cases provides comprehensive coverage of the application's key features and most important user scenarios.

The selected test cases focus on:

1. Essential timer functionalities (initialization, countdown, limits)
2. Widget UI updates and interactions
3. Critical system interactions (screen on/off, device restarts)
4. Configuration changes and their effects
5. Edge cases and potential error scenarios

By concentrating on these areas, the test suite achieves a balance between thoroughness and efficiency, allowing for effective quality assurance within the constraints of a one-person development team.

Below are the detailed test cases for the ScreenTime Widget:

TC-001: Daily Timer Initialization

Name: TC-001: Daily Timer Initialization

Requirement: FR-1: The widget must initialize the daily timer correctly with the user-set limit.

Preconditions: The ScreenTime Widget is installed and added to the home screen.

Steps:

1. Open the widget configuration screen.
2. Set the daily timer limit to 8 hours.
3. Set the session timer limit to 2 hours.
4. Save the configuration and return to the home screen.
5. Observe the widget display.

Expected results:

1. The widget should display "Daily: 08:00" (8 hours) and "Session: 02:00" (2 hours).
2. The daily timer should begin counting down when the screen is on.

TC-002: Session Timer Initialization

Name: TC-002: Session Timer Initialization

Requirement: FR-2: The widget must initialize the session timer correctly with the user-set limit.

Preconditions: The ScreenTime Widget is installed and added to the home screen.

Steps:

1. Open the widget configuration screen.
2. Set the daily timer limit to 8 hours.

3. Set the session timer limit to 2 hours.
4. Save the configuration and return to the home screen.
5. Observe the widget display.

Expected results:

1. The widget should display "Daily: 08:00" (8 hours) and "Session: 02:00" (2 hours).
2. The session timer should begin counting down when the screen is on.

TC-003: Daily Timer Countdown

Name: TC-003: Daily Timer Countdown

Requirement: FR-3: The daily timer must accurately count down screen time.

Preconditions: TC-001 has been executed successfully.

Steps:

1. Note the initial time displayed for the daily timer.
2. Keep the screen on for exactly 30 minutes.
3. Check the daily timer display on the widget.

Expected results:

1. The daily timer should display 7 hours and 30 minutes remaining (±5 seconds).

TC-004: Session Timer Countdown

Name: TC-004: Session Timer Countdown

Requirement: FR-4: The session timer must accurately count down continuous screen time.

Preconditions: TC-002 has been executed successfully.

Steps:

1. Note the initial time displayed for the session timer.
2. Keep the screen on for exactly 30 minutes.
3. Check the session timer display on the widget.

Expected results:

1. The session timer should display 1 hour and 30 minutes remaining (±5 seconds).

TC-005: Daily Timer Limit Reached

Name: TC-005: Daily Timer Limit Reached

Requirement: FR-5: The widget must notify the user when the daily timer limit is reached.

Preconditions: TC-001 has been executed successfully.

Steps:

1. Keep the device screen on until the daily timer reaches zero.
2. Observe the screen of the device.
3. If a notification appears, tap on it.
4. Dismiss the notification.
5. Check the widget display.

Expected results:

1. A notification should appear when the daily timer reaches zero.
2. The notification should state that the daily time limit has been reached.
3. Tapping the notification should close the notification.
4. After dismissal, the widget should display "Daily: 00:00".
5. As long as the screen is on, the widget should begin displaying negative values for the daily timer until the midnight reset.

TC-006: Session Timer Limit Reached

Name: TC-006: Session Timer Limit Reached

Requirement: FR-6: The widget must notify the user when the session timer limit is reached.

Preconditions: TC-002 has been executed successfully.

Steps:

1. Keep the device screen on continuously until the session timer reaches zero.
2. Observe the screen of the device.
3. If a notification appears, tap on it.
4. Dismiss the notification.
5. Check the widget display.

Expected results:

1. A notification should appear when the session timer reaches zero.
2. The notification should state that the session time limit has been reached.
3. Tapping the notification should close the notification.
4. After dismissal, the widget should display "Session: 00:00".
5. As long as the screen is on, the widget should begin displaying negative values for the session timer until the screen has been off for at least 15 minutes.

TC-007: Screen Off Detection

Name: TC-007: Screen Off Detection

Requirement: FR-7: Both timers must pause when the screen is turned off.

Preconditions: TC-003 and TC-004 have been executed successfully.

Steps:

1. Note the current times displayed for both daily and session timers.
2. Turn off the device screen.
3. Wait for 5 minutes.
4. Turn the screen back on.
5. Check the times displayed for both timers.

Expected results:

1. Both timers should show the same time as before the screen was turned off.

TC-008: Screen On Detection

Name: TC-008: Screen On Detection

Requirement: FR-8: Both timers must resume when the screen is turned on.

Preconditions: TC-007 has been executed successfully.

Steps:

1. Turn the screen on and keep it on for 5 minutes.
2. Check the times displayed for both timers.

Expected results:

1. Both timers should count down by 5 minutes (±5 seconds).

TC-009: Session Reset

Name: TC-009: Session Reset

Requirement: FR-9: The session timer must reset after 15 minutes of inactivity.

Preconditions: TC-004 has been partially executed with some time elapsed on the session timer.

Steps:

1. Turn off the screen for 15 minutes.
2. Turn the screen back on.
3. Check the session timer display.

Expected results:

1. The session timer should reset to its original full limit.

TC-010: Daily Reset at Midnight

Name: TC-010: Daily Reset at Midnight

Requirement: FR-10: The daily timer must reset at midnight.

Preconditions: TC-003 has been partially executed with some time elapsed on the daily timer.

Steps:

1. Set the device time to 23:59.
2. Wait for the device time to change to 00:00.
3. Check the daily timer display.

Expected results:

1. The daily timer should reset to its original full limit at 00:00.

TC-011: Widget Update

Name: TC-011: Widget Update

Requirement: FR-11: The widget display must update in real-time.

Preconditions: TC-003 and TC-004 have been partially executed with some time elapsed on both timers.

Steps:

1. Observe the widget display for 1 minute.

Expected results:

1. Both timer displays should update every second accurately.

TC-013: Configuration Changes - Daily

Name: TC-013: Configuration Changes - Daily

Requirement: FR-12: Users must be able to change the daily limit, which should be immediately reflected.

Preconditions: TC-001 has been executed successfully.

Steps:

1. Click the settings icon on the widget to open the configuration screen.
2. Change the daily limit to 4 hours.
3. Save the changes and return to the home screen.
4. Observe the widget display.

Expected results:

1. The widget should immediately display the new daily limit of 4 hours.
2. The session timer should not be affected by the change and continue counting down.

TC-014: Configuration Changes - Session

Name: TC-014: Configuration Changes - Session

Requirement: FR-13: Users must be able to change the session limit, which should be immediately reflected.

Preconditions: TC-002 has been executed successfully.

Steps:

1. Open the widget configuration screen.
2. Change the session limit to 1 hour.
3. Save the changes and return to the home screen.
4. Observe the widget display.

Expected results:

1. The widget should immediately display the new session limit of 1 hour.
2. The daily timer should not be affected by the change and continue counting down.

TC-015: Invalid Configuration Input

Name: TC-015: Invalid Configuration Input

Requirement: FR-14: The widget must handle invalid configuration inputs gracefully.

Preconditions: The widget configuration screen is open.

Steps:

1. Try to enter a negative value for the daily limit.
2. Try to enter a non-numeric value for the session limit.
3. Try to set the session limit higher than the daily limit.
4. Attempt to save these changes.

Expected results:

1. The widget should not accept invalid inputs.
2. An error message should be displayed if the user tries to set the session limit higher than the daily limit.
3. The values should remain unchanged until a valid limit is set for the timers.

TC-018: Widget Removal

Name: TC-018: Widget Removal

Requirement: FR-16: The widget must be removable without affecting the app installation.

Preconditions: There is only one ScreenTime widget on the home screen.

Steps:

1. Remove the ScreenTime widget from the home screen.
2. Try to add the widget again.

Expected results:

1. The widget is removed from the home screen.
2. Re-adding the widget requires the user to set up the timer limits again.

TC-021: Long-term Usage

Name: TC-021: Long-term Usage

Requirement: FR-17: The widget must function correctly over extended periods.

Preconditions: The ScreenTime widget is set up with daily and session limits.

Steps:

1. Use the device normally for 3 days.
2. Check the widget at various times each day.

Expected results:

1. The widget tracks time accurately.
2. Daily and session resets occur correctly.
3. No crashes or unexpected behaviors occur over the 3-day period.

TC-032: Device Restart

Name: TC-032: Device Restart

Requirement: FR-20: The widget must retain settings and current timer values after a device restart.

Preconditions: The ScreenTime Widget is set up and running with some time elapsed on both timers.

Steps:

1. Note the current values of both timers.
2. Restart the device.
3. After restart, check the widget display.

Expected results:

1. The widget appears without needing to be re-added.
2. Timers show the same values as before the restart.
3. Timers continue counting down correctly.

TC-039: Timer Sync Check

Name: TC-039: Timer Sync Check

Requirement: FR-21: Both timers must remain synchronized and accurate.

Preconditions: Both timers are set and running.

Steps:

1. Keep the screen on for exactly 1 hour.
2. Check both timer values.

Expected results:

1. Both timers should decrease by exactly 1 hour (±5 seconds).
2. The timers should remain synchronized with each other.

TC-041: Multiple Widget Instances

Name: TC-041: Multiple Widget Instances

Requirement: FR-22: Multiple instances of the widget must function correctly and stay synchronized.

Preconditions: The ScreenTime Widget app is installed.

Steps:

1. Add two instances of the ScreenTime Widget to the home screen.

2. Observe both widgets for 5 minutes.

Expected results:

1. Both instances show the same values.
2. Both instances update synchronously.
3. Configuration changes are reflected in both instances.

TC-044: Battery Optimization

Name: TC-044: Battery Optimization

Requirement: FR-23: The widget must function correctly even when battery optimization is enabled.

Preconditions: The ScreenTime Widget is set up and running.

Steps:

1. Enable battery optimization for the ScreenTime Widget app.
2. Use the device normally for 2 hours.
3. Check the widget display periodically.

Expected results:

1. The widget continues to track time accurately with no significant errors.

Automated Unit Tests

The following unit tests have been implemented to automatically verify core functionalities of the ScreenTime Widget:

1. Timer Functionality
    - testDailyTimerInitialization()
    - testSessionTimerInitialization()
    - testDailyTimerCountdown()
    - testSessionTimerCountdown()
    - testDailyTimerLimitReached()
    - testSessionTimerLimitReached()
    - testDailyTimerReset()
    - testSessionTimerReset()
    - testTimerInteraction()

2. Data Management
    - testDataManagerInitialization()
    - testDataManagerSetAndGetConfig()
    - testTimerLimitSettingAndRetrieval()
3. Widget UI
    - testWidgetUpdate()
4. Notification System
    - testNotificationLaunch()
5. Screen State Handling
    - testScreenStateManagerInitialization()
    - testScreenStateManagerCleanup()
    - testRapidScreenToggle()
6. Background Tasks
    - testDailyResetWorkerExecution()
7. Edge Cases and Integration
    - testSimultaneousLimitReached()
    - testTimerManagerPauseResume()

Mapping of Manual Test Cases to Unit Tests

Manual Test Case	Related Unit Test(s)
TC-001: Daily Timer Initialization	testDailyTimerInitialization()
TC-002: Session Timer Initialization	testSessionTimerInitialization()
TC-003: Daily Timer Countdown	testDailyTimerCountdown()
TC-004: Session Timer Countdown	testSessionTimerCountdown()
TC-005: Daily Timer Limit Reached	testDailyTimerLimitReached(), testSimultaneousLimitReached()
TC-006: Session Timer Limit Reached	testSessionTimerLimitReached(), testSimultaneousLimitReached()

Manual Test Case	Related Unit Test(s)
TC-007: Screen Off Detection	testScreenStateManagerInitialization(), testRapidScreenToggle()
TC-008: Screen On Detection	testScreenStateManagerInitialization(), testRapidScreenToggle()
TC-009: Session Reset	testSessionTimerReset()
TC-010: Daily Reset at Midnight	testDailyResetWorkerExecution()
TC-011: Widget Update	testWidgetUpdate()
TC-039: Timer Sync Check	testTimerInteraction()

7. Testing Procedures

General Testing Procedure:

1. Set up the test environment with the Android 12 emulator.
2. Install the latest version of the ScreenTime widget.
3. Execute each test case according to its specific steps.
4. Record the actual results and compare them with the expected results.
5. Mark the test case as "Pass" or "Fail" based on the comparison.
6. For failed tests, document the specific behavior using the procedure outlined in the Defect Management section with any associated error messages.

Automated Unit Test Procedures

Manual Execution of Unit Tests

1. Environment Setup:
    - Ensure Android Studio is installed with the latest updates.

    - Open the project repository in Android Studio.
    - Verify that all necessary dependencies are installed (JUnit, Mockito, Robolectric).
2. Running Unit Tests:
    - In Android Studio, right-click on the test directory (app/src/test).
    - Select "Run Tests in 'com.teamdelta.screentime'".
        i. Alternatively, use the command line: `./gradlew test`.
3. Interpreting Results:
    - Review the test results in the "Run" window of Android Studio.
    - For each test, check if it passed (green) or failed (red).
    - For failed tests, examine the error message and stack trace to understand the cause of failure.

Continuous Integration (CI) Testing Workflow

1. GitHub Actions Configuration:
    - Our CI pipeline is configured in the `.github/workflows/android_test.yml` file.
    - This workflow is set to run on every push and pull request to the main branch.
2. Automated Test Execution:
    - GitHub Actions automatically runs all unit tests for each push and pull request.
    - The CI environment is set up with all necessary dependencies and Android SDK components.
3. Reviewing CI Results:
    - I check the GitHub Actions tab in the repository for the test results.
    - Each workflow run will show the status of all tests (passed/failed).
    - For failed tests, I review the logs.
    - Pull requests cannot be merged unless all tests pass in the CI environment.
4. Resolve the Issue and Retest (if needed)
5. Push to main branch

8. Testing Schedule

Phase	Date	Duration
Automated Unit Testing	Continuous	N/A
Test Plan Development	JUL 10, 2024	0.5 day
Unit Testing	JUL 10, 2024	0.5 day
Integration Testing	JUL 11, 2024	0.5 day
Functional Testing	JUL 11, 2024 - JUL 12, 2024	1.5 days
UI and Usability Testing	JUL 12, 2024	0.5 day
Performance Testing	JUL 13, 2024	0.5 day
Compatibility Testing	JUL 13, 2024	0.5 day
Bug Fixes and Regression Testing	JUL 14, 2024	0.5 day
Final Test Report Preparation	JUL 14, 2024	0.5 day

9. Testing Resources

Human Resources:

- 1 Test Lead (Kalim Dausuel)
- 1 Android Developer (Kalim Dausuel)

Hardware:

- 1 emulated Android device (running Android 12)
- Personal computer

Software:

- Android Studio Koala (version 2024.1.1)
- JUnit for unit testing
- Mockito for mocking in unit tests
- Robolectric for Android unit testing
- Android Emulator for various device configurations
- Git for version control
- GitHub Actions for continuous integration and automated testing
- Issue tracking system (GitHub Issues)

10. Defect Management

The ScreenTime widget project uses GitHub Issues for defect management. As the sole developer and tester, I manage the entire process of identifying, tracking, and resolving issues.

Issue Identification and Reporting:

- Issues are identified during development, testing, or general usage of the ScreenTime Widget.
- Each issue is immediately logged in GitHub Issues, which automatically assigns a unique number.

Issue Documentation:

- Issues are documented using the following template in GitHub Issues:

```
### Description
[Brief description of the issue]
### Steps to Reproduce (if applicable)
1. [Step 1]
2. [Step 2]
3. [...]
### Expected vs Actual Behavior
- Expected: [What should happen]
- Actual: [What actually happened]
### Notes
[Any additional context, ideas for fixing, or screenshots]
### Priority
[High/Medium/Low]
```

Issue Tracking and Resolution:

- GitHub Issues' built-in Open/Closed states are used to track issue status.
- Issues are prioritized using labels (High, Medium, Low).
- Comments are added to issues to document progress, ideas, or roadblocks.
- Related commits or pull requests are linked to the issue for reference.

Verification and Closure:

- After implementing a fix, the functionality is re-tested thoroughly.
- The results of re-testing are documented as a comment on the issue.
- Issues are closed if the fix is successful, or kept open with notes if problems persist.

Progress Monitoring:

- Open issues are reviewed at the start of each development session.
- GitHub's project board feature is used to visualize the workflow (To Do, In Progress, Done).
- Closed issues are periodically reviewed to identify any patterns or areas needing improvement.

Process Refinement:

- The issue management process is periodically reviewed and adjusted for efficiency based on project needs.