Team Delta Phase II Source

Team Members:

Kalim Dausuel

SUMMER 2024

University of Maryland Global Campus

JUL 23, 2024

Table of Contents

1. Introduction

The ScreenTime Widget is an Android application designed to help users monitor and manage their device usage effectively. This Phase II Source Code document outlines the final implementation of the widget, including its core functionality, testing strategies, and deployment considerations.

Key features of the ScreenTime Widget include:

- Daily and session-based screen time tracking
- Home screen widget for easy visibility of screen time data
- Notifications when set time limits are reached
- Background operation to ensure accurate tracking

2. Project Setup

The project setup has been finalized and verified. Here's an overview of the development environment and tools used:

2.1 Development Environment

- Android Studio Koala (version 2024.1.1)
- Kotlin 2.0 as the primary programming language
- Minimum SDK: Android 12.0 (API level 31)
- M2 Macbook Pro

2.2 Version Control

- Git for version control
- GitHub repository link: https://github.com/kdausuel/screentime-capstone

2.3 Project Structure

The application is organized into the following packages, each responsible for specific functionalities:

- com.teamdelta.screentime.action
    - Handles user interactions and actions within the widget
    - Contains the LaunchConfigActionCallback for opening the configuration

activity
- com.teamdelta.screentime.data
    - Manages data persistence and retrieval
    - Houses the DataManager singleton for handling SharedPreferences
- com.teamdelta.screentime.notify
    - Responsible for the notification system
    - Includes NotificationActivity and NotificationLauncher for displaying alerts
- com.teamdelta.screentime.receiver
    - Contains broadcast receivers for system events and widget lifecycle
    - Includes ScreenStateManager, SessionResetReceiver, and ScreenTimeWidgetReceiver
- com.teamdelta.screentime.timer
    - Manages the core timer functionality
    - Contains Timer class, DailyTimer, SessionTimer, and TimerManager objects
- com.teamdelta.screentime.ui
    - Handles all user interface components
    - Includes ConfigActivity, ConfigUI, WidgetUI, ScreenTimeGlanceWidget, and TimeDisplayUtility
- com.teamdelta.screentime.worker
    - Manages background tasks and scheduling
    - Contains DailyResetScheduler and DailyResetWorker for daily timer resets

This structure separates concerns, enhances maintainability, and allows for easier testing and future expansion of the application.

2.4 Key Dependencies

- AndroidX libraries for modern Android development
- Glance for widget creation
- WorkManager for background task scheduling
- JUnit, Mockito, and Robolectric for testing

3. Core Functionality

The core functionality of the ScreenTime Widget has been fully implemented. Here's an overview of the main components:

3.1 Timer System

- `DailyTimer` and `SessionTimer` objects in the TimerObjects file extend the abstract `Timer` class which defines timer properties and behaviors.
- `TimerManager` coordinates timer updates and widget refresh operations

Example from `Timer.kt`:

```
package com.teamdelta.screentime.timer

import android.provider.ContactsContract.Data
import com.teamdelta.screentime.data.DataManager

/**
* Abstract base class for timers in the ScreenTime application.
*
* This class provides common functionality for different types of timers,
* including persistence of timer state using DataManager.
*
* @property timerId The unique identifier for this timer type.
*/
abstract class Timer(private val timerId : String) {

   /**
    * Resets the timer to its initial limit value.
    */
   fun reset() {
       DataManager.setTimerCurrentValue(timerId, limit ?:-1)
   }

   /**
    * The current value of the timer in seconds.
    */
   val currentValue : Int?
       get() = DataManager.getTimerCurrentValue(timerId)

   /**
    * The limit value of the timer in seconds.
    */
   val limit: Int?
       get() = DataManager.getTimerLimit(timerId)

   /**
    * Whether the timer is currently running or paused.
    */
   var isRunning: Boolean
       get() = DataManager.isTimerRunning(timerId)!!
```

```
        set(value) = DataManager.setTimerRunning(timerId, value)!!

    /**
     * Sets the limit for this timer.
     *
     * @param value The new limit value in seconds.
     */
    fun setLimit(value: Int) = DataManager.setTimerLimit(timerId, value)

    /**
     * Updates the current value of the timer.
     *
     * @param value The new current value in seconds.
     */
    fun updateCurrentValue(value: Int) =
DataManager.setTimerCurrentValue(timerId, value)

    /**
     * Checks if the timer has reached its limit (i.e., current value is
zero).
     *
     * @return True if the limit is reached, false otherwise.
     */
    fun isLimitReached(): Boolean = currentValue == 0
}
```

3.2 Widget UI

- `ScreenTimeGlanceWidget` provides the widget content
- `WidgetUI` handles the rendering of the widget interface

Example from `WidgetUI.kt`:

```
package com.teamdelta.screentime.ui.widget

import androidx.compose.runtime.Composable
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.glance.GlanceModifier
import androidx.glance.GlanceTheme
import androidx.glance.ImageProvider
import androidx.glance.action.clickable
import androidx.glance.appwidget.action.actionRunCallback
```

```
import androidx.glance.background
import androidx.glance.currentState
import androidx.glance.layout.Alignment
import androidx.glance.layout.Column
import androidx.glance.layout.Row
import androidx.glance.layout.fillMaxSize
import androidx.glance.layout.fillMaxWidth
import androidx.glance.layout.padding
import androidx.glance.text.Text
import androidx.glance.text.TextStyle
import com.teamdelta.screentime.R
import com.teamdelta.screentime.action.LaunchConfigActionCallback
import com.teamdelta.screentime.ui.TimeDisplayUtility.formatTime
import com.teamdelta.screentime.ui.widget.ScreenTimeGlanceWidget.dailyTime
import
com.teamdelta.screentime.ui.widget.ScreenTimeGlanceWidget.sessionTime

/**
* Object containing UI components and functions for the ScreenTime widget.
*/
object WidgetUI {

   /**
    * Function that displays the main content for the ScreenTime widget.
    * It includes two timer displays for the daily and session timer and a
settings icon.
    */
   @Composable
   fun Content() {
       val dailyTime = currentState(key = dailyTime) ?: 0
       val sessionTime = currentState(key = sessionTime) ?: 0
       GlanceTheme {
           Column(
               modifier = GlanceModifier
                   .fillMaxSize()
                   .background(GlanceTheme.colors.background)
           ) {
               Row(
                   modifier = GlanceModifier
                       .fillMaxWidth()
                       .padding(8.dp)
               ) {
                   TimerDisplay(
                       // Daily Timer display
                       label = "Daily",
                       time = formatTime(dailyTime),
```

```
                        modifier = GlanceModifier.defaultWeight()
                    )
                    TimerDisplay(
                        // Session Timer display
                        label = "Session",
                        time = formatTime(sessionTime),
                        modifier = GlanceModifier.defaultWeight()
                    )
                }

                // Settings icon/button functionality
                Row(
                    modifier = GlanceModifier.fillMaxWidth(),
                    horizontalAlignment = Alignment.End
                ) {
                    androidx.glance.Image(
                        provider = ImageProvider(R.drawable.icon_settings),
                        contentDescription = "Settings",
                        modifier = GlanceModifier
                            .padding(8.dp)

.clickable(actionRunCallback(LaunchConfigActionCallback::class.java))
                    )
                }
            }
        }
    }

    /**
     * A function that displays a timer with a label and the time.
     *
     * @param label The label text to be displayed above the time.
     * @param time The time text to be displayed below the label.
     * @param modifier A GlanceModifier to be applied to the
     *  Column container.Defaults to [GlanceModifier].
     */
    @Composable
    private fun TimerDisplay(
        label: String,
        time: String,
        modifier: GlanceModifier = GlanceModifier
    ) {
        Column(
            modifier = modifier.padding(4.dp),
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
```

```
            Text(
                text = label,
                style = TextStyle(
                    color = GlanceTheme.colors.onBackground,
                    fontSize = 14.sp
                )
            )
            Text(
                text = time,
                style = TextStyle(
                    color = GlanceTheme.colors.primary,
                    fontSize = 18.sp
                )
            )
        }
    }
}
```

3.3 Configuration

- `ConfigActivity` allows users to set and modify timer limits
- `ConfigUI` provides the user interface for configuration

3.4 Data Management

- `DataManager` handles data persistence using SharedPreferences

3.5 Notification System

- `NotificationActivity` and `NotificationLauncher` manage alert displays

3.6 Background Processing

- `DailyResetScheduler` and `DailyResetWorker` handle daily timer resets
- `ScreenStateManager` monitors device screen state

3.7 Broadcast Receivers

- `SessionResetBootReceiver` reschedules alarms after device reboot
- `SessionResetReceiver` manages session timer resets

- `ScreenTimeWidgetReceiver` handles widget lifecycle events

4. Documentation

The project has been thoroughly documented to enhance readability and maintainability:

4.1 Code Comments

- Each class and major function includes KDoc comments explaining its purpose and functionality

Example from `ConfigActivity.kt`:

```
/**
 * Activity for configuring the ScreenTime widget.
 *
 * This activity allows users to set up and modify timer limits and
other widget settings.
 * It handles the initialization of necessary permissions and
displays the configuration UI.
 */
class ConfigActivity : ComponentActivity() {
    // Implementation details...
}
```

4.2 README File

A comprehensive README.md file in the root directory of the repository provides:

- Project overview and purpose
- Setup instructions
- Usage guide
- List of key features

4.3 Dependency Documentation

All project dependencies are documented in the build.gradle files, including version numbers.

5. Unit Testing

A comprehensive unit testing strategy has been implemented:

5.1 Detailed Test Cases and Results

To keep the length of this document low, I have put the detailed test cases as an appendix at the end. Additionally, an updated test case results table will be submitted along with this document.

5.2 Testing Frameworks

- JUnit 4 for unit testing
- Mockito for creating and configuring mock objects
- Robolectric for Android-specific component testing

5.3 Test Coverage

- Timer functionality
- Data management
- Widget UI updates
- Notification system
- Screen state handling
- Background tasks
- Edge cases and integration scenarios

5.4 Continuous Integration

GitHub Actions is used to run all unit tests on every push and pull request to the main branch.

5.5 Example Unit Test

```
/**
 * Tests the setting and retrieval of timer limits, current values,
and running states in DataManager.
 *
 * This test verifies that:
 * 1. Timer limits can be set and retrieved correctly for both daily
and session timers.
```

```
 * 2. Current timer values can be set and retrieved accurately.
 * 3. The running state of a timer can be set and checked correctly.
 *
 * Test steps:
 * 1. Set and verify the daily timer limit.
 * 2. Set and verify the session timer limit.
 * 3. Set and verify the current value of the daily timer.
 * 4. Set the session timer to running and verify its state.
 */
@Test
fun testTimerLimitSettingAndRetrieval() {
    // Test setting and getting daily timer limit
    DataManager.setTimerLimit("daily", 21600) // 6 hours in seconds
    assertEquals(21600, DataManager.getTimerLimit("daily"))

    // Test setting and getting session timer limit
    DataManager.setTimerLimit("session", 7200) // 2 hours in seconds
    assertEquals(7200, DataManager.getTimerLimit("session"))

    // Test setting and getting current timer value
    DataManager.setTimerCurrentValue("daily", 10800) // 3 hours in
seconds
    assertEquals(10800, DataManager.getTimerCurrentValue("daily"))

    // Test setting and checking timer running state
    DataManager.setTimerRunning("session", true)
    DataManager.isTimerRunning("session")?.let { assertTrue(it) }
}
```

6. Deployment and Maintenance

6.1 Deployment

- The widget will be distributed through the Google Play Store.
- A CI/CD pipeline using GitHub Actions has been set up for automated building and testing.

6.2 Maintenance

- User feedback will be monitored through the Play Store reviews.

7. Conclusion

The ScreenTime widget project has successfully implemented all required features. The use of modern Android development practices, including Kotlin and the Glance library, ensures the widget will be able to be maintained easily.

Appendix A. Test Cases

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()
TC-007: Screen Off Detection	testScreenStateManagerInitialization(), testRapidScreenToggle()
TC-008: Screen On Detection	testScreenStateManagerInitialization(), testRapidScreenToggle()
TC-009: Session Reset	testSessionTimerReset()

Manual Test Case	Related Unit Test(s)
TC-010: Daily Reset at Midnight	testDailyResetWorkerExecution()
TC-011: Widget Update	testWidgetUpdate()
TC-039: Timer Sync Check	testTimerInteraction()