Team Delta Phase I Source

Team Members:

Kalim Dausuel

SUMMER 2024

University of Maryland Global Campus

JUL 09, 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 report outlines the development process, core functionality, documentation practices, and testing procedures implemented in the creation of this widget.

2. Project Setup

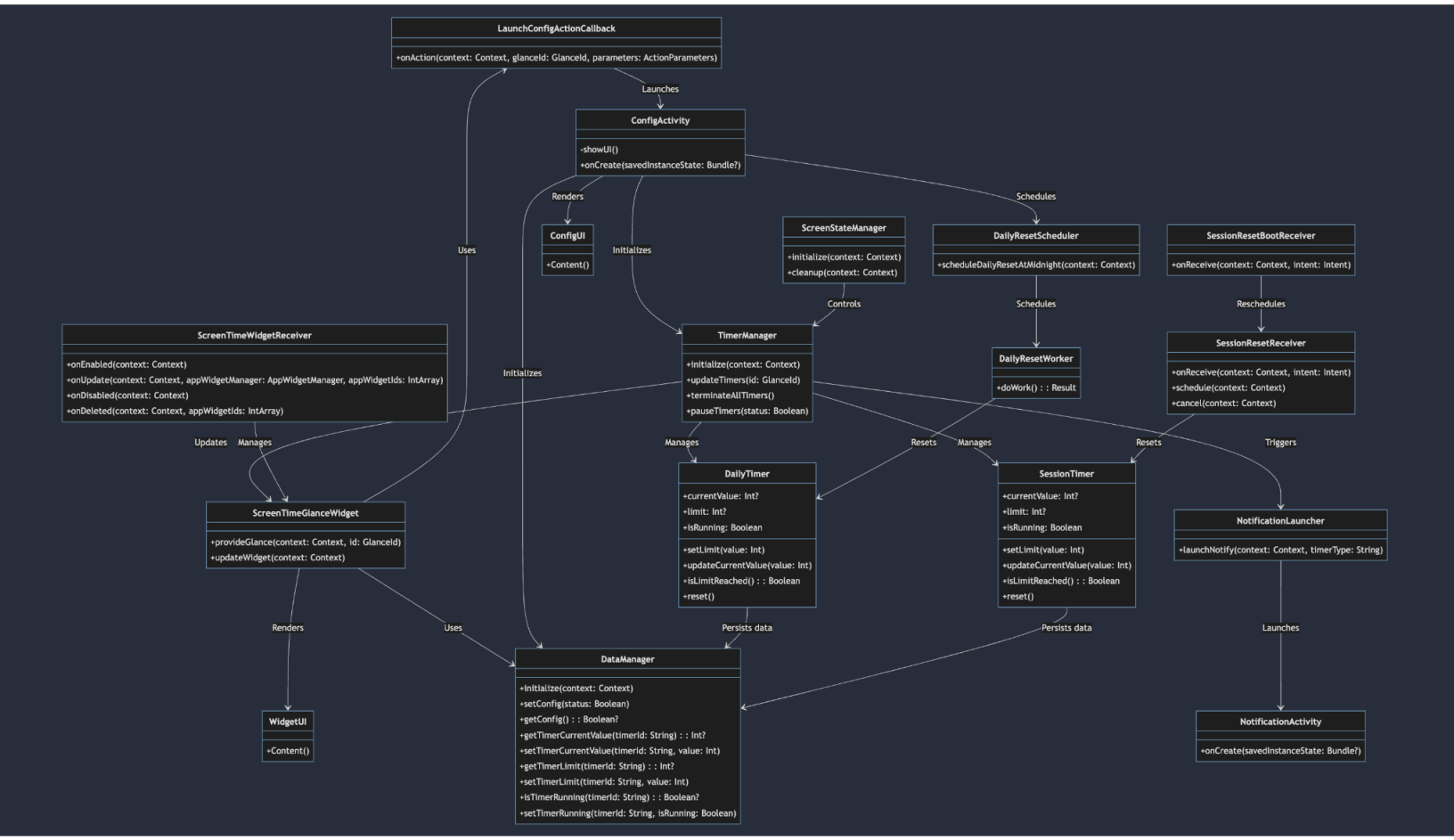
1. Development Environment:
    - Android Studio: The project uses Android Studio Koala (version 2024.1.1) as the primary Integrated Development Environment.
    - Kotlin: Kotlin 2.0 was chosen as the primary programming language due to its modern features as well as being the recommended language by Google for Android development.
2. Version Control:
    - Git: Git is utilized for version control.
    - GitHub: The repository is hosted on GitHub at https://github.com/kdausuel/screentime-capstone
3. Application Directory Structure:
    - The app is organized into these folders:
        - `com.teamdelta.screentime.action`: For user touch interactions
        - `com.teamdelta.screentime.data`: For data management
        - `com.teamdelta.screentime.notify`: For notification related functionality
        - `com.teamdelta.screentime.receiver`: For broadcast receivers (interactions between Android OS and application)
        - `com.teamdelta.screentime.timer`: For timer logic
        - `com.teamdelta.screentime.ui`: For user interface components
        - `com.teamdelta.screentime.worker`: For background workers

4. Dependency Management:

- Gradle: I am using Gradle to manage project dependencies and build configurations.
- Key dependencies include:
  - AndroidX libraries for modern Android development
  - Glance for widget creation
  - WorkManager for background task scheduling

3. Project Structure

UML Diagram



Components

1. `ScreenTimeGlanceWidget`: The main widget class that provides the Glance widget content and handles updates.
2. `WidgetUI`: Responsible for rendering the widget's user interface.
3. `ConfigActivity`: Handles the widget configuration, allowing users to set timer limits.
4. `DataManager`: A singleton object managing data persistence across the application.
5. `TimerManager`: Coordinates timer updates and widget refresh operations.
6. `DailyTimer` and `SessionTimer`: Represent the two types of timers in our application.
7. `ScreenStateManager`: Monitors the device's screen state to pause/resume timers accordingly.
8. `NotificationLauncher`: Handles the display of notifications when time limits are reached.
9. `DailyResetScheduler` and `DailyResetWorker`: Manage the daily reset of timers at midnight.
10. `ScreenTimeWidgetReceiver`: Manages the lifecycle of the ScreenTimeGlanceWidget, handling events such as widget creation, updates, and deletion.
11. `SessionResetBootReceiver`: Responsible for rescheduling the session reset after device reboot.
12. `SessionResetReceiver`: Handles the resetting of the session timer at specified intervals or conditions.
13. `ConfigUI`: Represents the user interface components of the configuration screen.
14. `ScreenStateManager`: Monitors and manages the screen state of the device, controlling the TimerManager accordingly.
15. `DailyResetScheduler`: Schedules the daily reset of timers at midnight.
16. `DailyResetWorker`: Performs the actual reset of the daily timer when scheduled.
17. `LaunchConfigActionCallback`: Acts as a bridge between the ScreenTimeGlanceWidget and the ConfigActivity, handling the action of launching the configuration screen.

4. Core Functionality

The ScreenTime Widget implements the required features efficiently and with a modular design to satisfy the core functionality requirements:

1. Implementation of Required Features:
    - a. Daily and Session Timers:
        i. DailyTimer and SessionTimer classes implement time tracking for daily total and continuous session use.
        ii. Timers accurately count down and reset as per project specifications.
    - b. Widget Display:
        i. ScreenTimeGlanceWidget provides a home screen widget showing current timer values.
        ii. WidgetUI renders the widget interface, updating in real-time.
    - c. Notifications:
        i. NotificationActivity.kt handles alerting users when time limits are reached via an AlertDialog.
    - d. Configuration:
        i. ConfigActivity allows users to set and modify timer limits.

2. Efficient Algorithms:
    - a. Timer Management:
        i. TimerManager uses Kotlin coroutines for efficient, non-blocking timer updates.
    - b. Data Persistence:
        i. DataManager employs SharedPreferences for fast, lightweight data storage and retrieval.
    - c. Screen State Tracking:
        i. ScreenStateManager utilizes system broadcasts for efficient screen state monitoring.
    - d. Background Processing:
        i. WorkManager (via DailyResetWorker) is used for efficient, battery-friendly background tasks.
3. Modular Component Organization:
    - a. Separation of Concerns:
        i. Timer logic (DailyTimer, SessionTimer) is separated from UI (WidgetUI, ConfigUI).

        ii. Data management (DataManager) is isolated from business logic (TimerManager).
    - b. Reusable Components:
        i. NotificationLauncher can be used for different notification types.
        ii. ScreenTimeGlanceWidget serves as a reusable widget template.

Several enhancements are planned for Phase 2 of the project to ensure the widget passes the test cases, including:

- Improved configuration UI for setting/resetting individual timer limits
- Code optimization to reduce duplication between timer objects
- Consistent data management using SharedPreferences and DataManager
- Performance enhancements to move operations to background thread
- Separating UI elements from ConfigAcitivity as was done with WidgetUI
- Fixing UI for properly displaying negative time (user is using phone past the timer reaching zero)
- Reconfiguration UI properly pulling existing values for daily and session timer

5. Documentation

I have worked towards adhering to Kotlin style guidelines. Each class has KDoc comments to explain its purpose and functionality. As I move into Phase 2, I will extend this to all functions and complex code blocks via in-line comments.

- GitHub Repository:
    - The project's GitHub repository (https://github.com/kdausuel/screentime-capstone) serves as the central hub for code storage and version control.
    - The repository includes all source code, documentation, and project-related files.
- README File:
    - The project includes a detailed README.md file in the root directory, providing:
        - Project overview and purpose
        - Setup instructions
        - Usage guide
        - List of key features

        - Troubleshooting tips
- Git Commit Messages:
    - I have maintained clear and descriptive Git commit messages, providing a history of project changes and decision-making.
- Dependency Documentation:
    - I have documented all project dependencies in the build.gradle files, including version numbers for future reference.

6. Unit Testing Strategy

To ensure the reliability and correctness of ScreenTime, a comprehensive unit testing strategy has been implemented. This strategy focuses on covering all major functionalities of the application and utilizes appropriate testing frameworks for Android development.

Testing Frameworks

The following testing frameworks have been employed in this project:

1. JUnit 4: The primary unit testing framework for Java and Kotlin on the JVM.
2. Mockito: Used for creating and configuring mock objects in unit tests.
3. AndroidX Test: Provides additional testing utilities for Android-specific components.

Test Coverage

The unit tests aim to cover all major functionalities of the application. Here's an overview of the test coverage for each main component:

1. Timer Logic (DailyTimer and SessionTimer):
    - Initialization of timers with various time limits
    - Countdown functionality
    - Reset behavior
    - Limit reached detection
2. Data Management (DataManager):
    - Saving and retrieving timer values
    - Persisting and loading configuration settings
    - Data integrity across app restarts
3. Widget UI (ScreenTimeGlanceWidget and WidgetUI):
    - Correct display of timer values
    - UI updates in response to timer changes
4. Notification System:

    - Triggering notifications at appropriate times
    - Correct notification content
    - Handling of multiple simultaneous notifications
5. Screen State Management:
    - Proper pausing and resuming of timers based on screen state
    - Accurate tracking of active screen time
6. Background Processing:
    - Continued timer operation when app is in the background
    - Proper scheduling and execution of daily resets

Example Unit Test

Here is an example of a test developed to ensure the DataManager properly stores/retrieves the initial timer limits set by the user, can pull the current time left on a timer, and report the status of a timer (running or paused) :

```
@Test
    fun testTimerLimitSettingAndRetrieval() {
        // Test Daily Timer
        DataManager.setTimerLimit("daily", 21600) // 6 hours in seconds
        assertEquals(21600, DataManager.getTimerLimit("daily"))

        // Test Session Timer
        DataManager.setTimerLimit("session", 7200) // 2 hours in seconds
        assertEquals(7200, DataManager.getTimerLimit("session"))

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

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

Continuous Integration

To ensure consistent quality and catch potential issues early, I am working on integrating our unit tests into a continuous integration (CI) pipeline using GitHub Actions. This pipeline will automatically run all unit tests on every push to the repository and pull request, providing immediate feedback on the impact of code changes.

7. Test Cases

These test cases cover a wide range of scenarios and edge cases to ensure the widget works. They address core timer functionality, user interface elements, system interactions, data persistence, and other real-world usage scenarios.

Test#	Situation	Inputs	Expected Output(s)
TC001	Daily Timer Initialization	Set daily limit to 8 hours	Daily timer displays 8:00 remaining
TC002	Session Timer Initialization	Set session limit to 2 hours	Session timer displays 2:00 remaining
TC003	Daily Timer Countdown	Active screen time for 30 minutes	Daily timer displays 7:30 remaining
TC004	Session Timer Countdown	Active screen time for 15 minutes	Session timer displays 1:45 remaining
TC005	Daily Timer Limit Reached	Active screen time reaches daily limit	Notification displayed; Daily timer shows 0:00
TC006	Session Timer Limit Reached	Active screen time reaches session limit	Notification displayed; Session timer shows 0:00
TC007	Screen Off Detection	Turn off device screen	Both timers pause countdown
TC008	Screen On Detection	Turn on device screen after pause	Both timers resume countdown
TC009	Session Reset	Screen off for 15 minutes	Session timer resets to original limit
TC010	Daily Reset at Midnight	System time passes midnight	Daily timer resets to original limit; Session timer

Test#	Situation	Inputs	Expected Output(s)
			unaffected
TC011	Widget Update	Any timer value change	Widget display updates with new timer values
TC012	Data Persistence	Reset phone	Timer values and settings are retained
TC013	Configuration Changes - Daily	Modify daily limit to 6 hours	Daily timer updates to show 6:00 remaining
TC014	Configuration Changes - Session	Modify session limit to 1 hour	Session timer updates to show 1:00 remaining
TC015	Invalid Configuration Input	Enter negative value for timer limit	Error message displayed; limit not changed
TC016	Multiple Notification Trigger	Reach both daily and session limits simultaneously	Both notifications displayed without conflict
TC017	Widget Addition	Add widget to home screen	Widget displays correctly with current timer values
TC018	Widget Removal	Remove last widget instance from home screen	Timer data is deleted; no crashes occur
TC020	Rapid Screen On/Off	Quickly turn screen on and off multiple times	Timers accurately track active time without errors
TC021	Long-term Usage	Simulate usage over several days	Daily resets occur correctly; no cumulative errors
TC022	Timezone Change	Change device timezone	Daily reset timing adjusts correctly
TC023	Daylight Saving Time Change	Simulate DST change	Daily reset timing remains correct
TC024	Notification Interaction	Dismiss time limit notification	Application continues normal operation

Test#	Situation	Inputs	Expected Output(s)
TC025	Background Operation	Use other apps for extended period	Screen Time Widget continues tracking accurately
TC026	Session Timer > Daily Timer	Set session limit to 10 hours, daily limit to 8 hours	Error message; Session limit not allowed to exceed daily limit
TC027	Daily Limit Decrease	Change daily limit from 8 hours to 4 hours when 6 hours remain	Timer updates to 4:00 remaining; excess time is not carried over
TC028	Session Limit Decrease	Change session limit from 2 hours to 1 hour when 1:30 remains	Timer updates to 1:00 remaining; excess time is not carried over
TC029	Daily Limit Increase	Change daily limit from 4 hours to 6 hours when 2 hours remain	Timer updates to 4:00 remaining (2 hours remaining + 2 hours added)
TC030	Session Limit Increase	Change session limit from 1 hour to 2 hours when 0:30 remains	Timer updates to 1:30 remaining (0:30 remaining + 1 hour added)
TC032	Device Restart	Restart device	Application resumes with correct timer values upon device restart
TC033	Date Change Without Midnight	Change device date forward	Daily timer continues countdown; no premature reset
TC034	Date Change Backwards	Change device date backwards	Daily timer continues countdown; no unexpected reset
TC038	Rapid Limit Changes	Change limits multiple times rapidly	Last set valid limit should be applied without errors
TC039	Timer Sync	Check both timers after 1 hour of usage	Both timers should have decreased by exactly 1 hour

Test#	Situation	Inputs	Expected Output(s)
TC040	Session Continuity	Use device for 2 hours without 15-min break	Session timer should not reset during continuous use
TC041	Multiple Widget Instances	Add multiple widgets to home screen	All widgets show the same, correct timer values
TC044	Battery Optimization	Enable battery optimization for the app	Timers continue to function correctly in the background
TC045	Large Time Limits	Set daily limit to 23 hours, session to 22 hours	Application handles large values correctly
TC046	Small Time Limits	Set daily limit to 1 minute, session to 30 seconds	Application handles small values correctly
TC047	Airplane Mode	Toggle airplane mode on/off	Timers continue to function correctly
TC048	Data Clear	Clear app data from system settings	Application resets to initial state with default values
TC050	Complete App Uninstall	Uninstall app and clear data	Upon reinstall, app starts with factory settings

8. Version Control Strategy

Current Workflow

As of now, all changes have been committed directly to the main branch with simple, explanatory commit messages. While this approach has worked for initial development, I plan on adopting a more structured version control strategy that can greatly benefit the project as it matures. The strategy I am going to employ is detailed in this section.

Git Workflow

The project utilizes a simplified Git workflow centered around two types of branches:

1. Main Branch:
    - Named 'main'
    - Represents the official release history
    - Always maintained in a production-ready state
2. Feature Branches:
    - Created for new features or significant changes
    - Named descriptively to reflect the feature or fix being implemented
    - Merged directly into 'main' upon completion

Development Process

The development process will follow these steps:

1. Initiation of New Work:
    - A new branch is created from 'main' for each feature or bug fix
2. Feature Development:
    - Regular commits are made with messages
3. Feature Completion:
    - Testing is conducted on the feature branch
4. Feature Integration:
    - Upon satisfactory completion, the feature branch is merged into main:
5. Branch Cleanup:
    - Feature branches are deleted after successful merging

Version Tagging

Release versions will be marked using Git tags on the main branch.

Commit Message Format

In accordance with best practices, all commit messages will follow this format:

```
<type>(<scope>): <subject>

<body>

<footer>
```

Where:

- `<type>` is one of: feat, fix, docs, style, refactor, test, chore
- `<scope>` represents the module affected
- `<subject>` is a short description of the change
- `<body>` provides more detailed explanatory text, if necessary
- `<footer>` is used for referencing issue trackers

Versioning

The project will use Semantic Versioning for clear release numbering:

- MAJOR version for incompatible API changes
- MINOR version for backwards-compatible functionality additions
- PATCH version for backwards-compatible bug fixes

9. Project Installation Guide

This section provides a comprehensive guide for locally installing the project.

Repository Submission

The complete source code for this project is available in the GitHub repository:

https://github.com/kdausuel/screentime-capstone

This repository contains all the necessary files, including source code, configuration files, and documentation.

Documentation Overview

This document serves as the comprehensive documentation file required for the project submission. It includes the following key components:

1. Project Overview
2. Installation Instructions
3. Usage Guidelines
4. Technical Details

1. Project Overview

The Screen Time Widget is an Android application designed to help users monitor and manage their device usage. Key features 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

This project demonstrates proficiency in Android development, including widget creation, background processing, and data persistence.

2. Installation Instructions

To install and run the Screen Time Widget:

1. Clone the GitHub repository:

   ```
   git clone https://github.com/kdausuel/screentime-capstone.git
   ```

2. Open the project in Android Studio Koala (2024.1.1) or later.
3. Ensure you have the Android SDK for the minimum supported API level (as specified in the `build.gradle` file) installed.
4. Build the project in Android Studio. (https://developer.android.com/studio/run)
5. Run the application on an emulator or physical Android device running Android 12.0 or higher.

3. Usage Guidelines

After installation:

1. Add the ScreenTime Widget to your Android home screen.
2. Accept the permission requests.
3. Set your desired daily and session time limits (time unit is in seconds for now).
4. The widget will now display your remaining screen time for both daily and session limits.
5. You will receive a notification when you reach your set time limits.
6. Tap the gear icon on the widget to open the configuration screen.

Note: The full configuration UI is planned for Phase 2 of the project.

4. Technical Details

- Language: Kotlin 2.0
- Minimum SDK Version: Android 12.0 (API level 31)
- Key Dependencies:
    - AndroidX libraries
    - Glance for widget creation
    - WorkManager for background tasks