

Dokumentation

Android App: ScreenTime

– ÜBERBLICK –

In dieser Dokumentation beschreibe ich kurz, wie meine App ScreenTime konzipiert ist und welche Funktionen in dem Projekt realisiert wurden. Mein Anspruch war es eine App zu entwickeln, die dem User einen Überblick über die am Smartphone-Bildschirm verbrachte Zeit bietet. Außerdem sollte die Möglichkeit bestehen, für jede App ein Zeitlimit festzulegen. ScreenTime wurde für SDK Versionen 26+ entwickelt und setzt alle gerade genannten Funktionalitäten um. Einzig das Feature, dass das Überschreiten des Limits einer App auch entdeckt wird, wenn die ScreenTime App selbst geschlossen ist, würde in einer zukünftigen Version der App noch implementiert werden müssen.

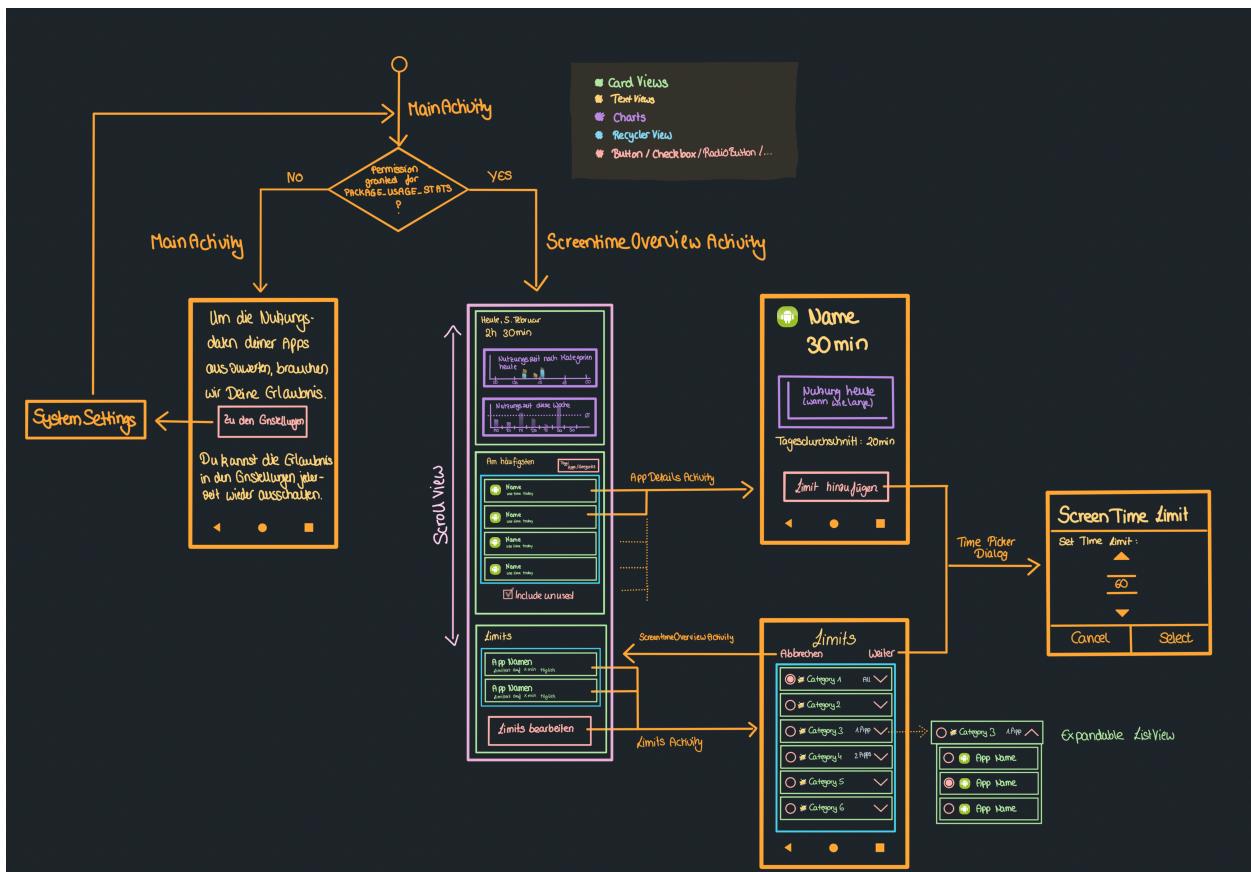


Abbildung 1: Konzeptentwurf der ScreenTime App

I ANFORDERUNGEN

Über die Schnittstelle zum System sollen Daten zur Nutzung des Smartphones gesammelt und visualisiert werden. Zudem soll es möglich sein, Limits für die Nutzung bestimmter Apps festzulegen. Die App soll folgende Funktionen bieten:

1. Übersicht über die Bildschirmzeit geben
 - Wie viel Zeit wurde heute insgesamt am Bildschirm verbracht?
 - Wie viel Zeit wurde in welcher App oder Kategorie verbracht?
 - Welche Apps oder Kategorien waren am längsten im Vordergrund?
2. Limit für die Bildschirmzeit einzelner Apps festlegen

II KONZEPT

II.1 Vorarbeit

Um einschätzen zu können, ob die Idee für ein erstes Android-App Projekt geeignet war, habe ich mir zunächst überlegt, welche Daten ich für die Umsetzung brauchen würde:

- Nutzungszeiten aller Apps
 - UsageStatsManager
 - Permission vom User einholen
- Metadaten aller Apps, z.B. App-Name, Icon, Kategorie
 - PackageManager
 - Permission im Manifest

All dieses Daten sind auch für Android-Anfänger relativ einfach abzurufen und das Projekt sollte somit für mich umsetzbar sein.

II.2 Erster Entwurf

Abbildung 1 zeigt den ersten Entwurf der ScreenTime App. Dieser Entwurf ist eine Mischung aus Wireframes und Ablaufdiagramm, was sowohl für das Layout-Design als auch für die Programmierung sehr hilfreich war. Wurde die Permission für den Usage Access noch nicht erteilt, wird man zuerst in die Systemeinstellungen weitergeleitet. Wurde die Permission erteilt, öffnet die App direkt die ScreenTimeOverviewActivity. Hier werden die Bildschirmzeit-Daten und App-Limits auf drei CardViews übersichtlich zusammengefasst und sie ist Ausgangspunkt für alle weiteren Aktionen. Von dieser Activity aus kann man zum einen in die App-DetailsActivity wechseln, in der Details zu einer bestimmten App eingesehen und ein Limit für diese App festgelegt werden können. Zum anderen kann man in die LimitsActivity wechseln. Hier sind die Apps nach Kategorien geordnet und für jede App kann ein Limit festgelegt werden.

III REALISIERUNG

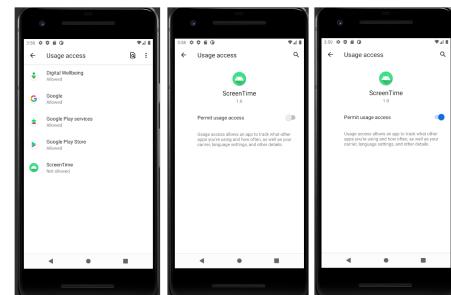


Abbildung 2: Erteilen der PACKAGE_USAGE_STATS in den Systemeinstellungen

(1) *Permissions* Voraussetzung für die Funktionalität der App ist die Usage Access Permission. Diese muss vom User manuell erteilt werden. Die Permission kann jederzeit unter Set-

tings → Apps & Notifications → Advanced → Special app access → Usage Access erteilt und auch wieder entzogen werden.

(2) *RecyclerView* Die Bildschirmzeiten der Apps und Kategorien sollten möglichst übersichtlich in jeweils einer nach Dauer der Nutzung sortierten Liste dargestellt werden. Da im Voraus nicht feststeht, wie viele Einträge eine solche Liste hat oder in welcher Reihenfolge sie stehen, und da die Liste auch immer wieder aktualisiert werden soll, eignet sich an dieser Stelle ein RecyclerView[3], welcher dynamische Listen ermöglicht. Die Elemente, die die Daten für den RecyclerView liefern, wurden mithilfe des in Abbildung 3 gezeigten Item Interfaces realisiert. Die Klassen AppItem und CategoryItem implementieren dieses Interface und enthalten daher alle relevanten Informationen wie den Namen des Items, die Nutzungszeit und das festgelegte Limit.

```
interface Item
{
    val itemName: String
    val packageName: String
    val icon: Drawable?
    var useTime: Long
    var readableUseTime: String
    var wasUsed: Boolean

    val categoryId: Int

    val category : AppCategory

    var limit: Int

    operator fun compareTo(other: Item): Int
    {
        return compareValues(useTime, other.useTime)
    }

    fun updateUseTime(time: Long) : Long
}
```

Abbildung 3: Das Item-Interface

Die Klasse ItemContainer hält eine Liste von Items und führt darauf Operationen durch. Ergänzt wird dieses System durch den RecyclerAdapterItem und seine jeweiligen Kindklassen für AppItems und CategoryItems. Der Adap-

ter befüllt die jeweiligen RecyclerViews mit den entsprechenden Daten.

(3) *Usage Stats* Die aktuellen Nutzungsstatistiken werden, wie in Abbildung 4 zu sehen, mit Hilfe des UsageStatsManagers abgerufen.

```
private fun getDailyUsageStats(): List<UsageStats>
{
    val calendar = Calendar.getInstance()
    calendar.add(Calendar.DAY_OF_MONTH, -1)

    val usageStatsManager: UsageStatsManager = getSystemService(USAGE_STATS_SERVICE) as UsageStatsManager

    return usageStatsManager.queryUsageStats(UsageStatsManager.INTERVAL_DAILY,
        calendar.timeInMillis,
        System.currentTimeMillis())
}
```

Abbildung 4: Nutzungsstatistiken des aktuellen Tages mit Hilfe des UsageStatsManagers beziehen

Mit dem PackageManager werden anschließend die Metadaten der einzelnen Apps abgerufen, so dass die entsprechenden AppItems mit Namen, Icons und Kategorien befüllt werden können. Um Daten zentral zu speichern und für alle Activities der App verfügbar zu machen, wurde die von Application erbende Klasse ScreenTimeApp als Singleton implementiert. Die ScreenTimeApp hält zwei ItemContainer, einen für die Apps und einen für die Kategorien. Die Liste mit allen Apps wird einmal beim Start der App aus den aktuellen UsageStats erstellt und zentral in der ScreenTimeApp Klasse gespeichert. Anschließend wird aus dieser Liste die Liste der Kategorien abgeleitet und gespeichert. Die beiden Container werden im Laufe der Nutzung immer wieder mit den aktuellen Nutzungszeiten und Limits geupdated und stellen so jederzeit aktuelle Informationen zu allen Apps und Kategorien zur Verfügung. Ein erstes Befüllen des RecyclerViews mit den Items sah noch recht rudimentär aus, doch mit ein wenig Formatierung und Hintergrundberechnungen in den jeweiligen Item und ItemContainer Klassen ließ sich dies gut beheben. Abbildung 5 zeigt den ersten Versuch neben dem ersten lesbaren Zustand der App-Liste.

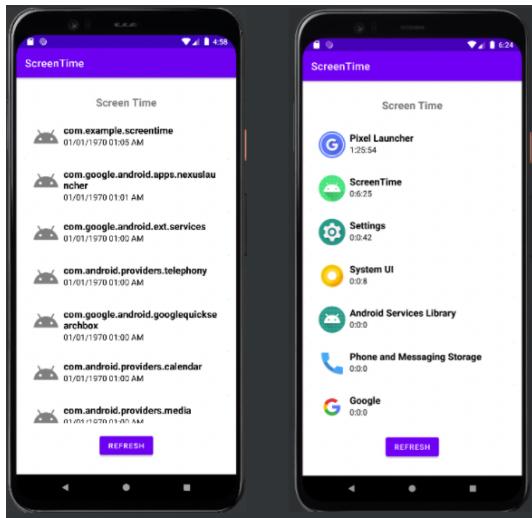


Abbildung 5: Die erste unformatierte Ausgabe der Nutzungsstatistiken neben der formatierten

An diesem Punkt stand das Grundgerüst der App. Das aktuelle Datum und die gesamte Bildschirmzeit des Tages wurden nun in die Übersicht im Abschnitt *Today* eingefügt, um direkt auf einen Blick die wichtigsten Informationen zu haben. Im Abschnitt *Most Used* sollte nun auch noch möglichst einfach zwischen den beiden RecyclerViews von Apps und Kategorien hin und her gewechselt werden können. Für diese Funktionalität wurde ein ViewFlipper [5] verwendet.

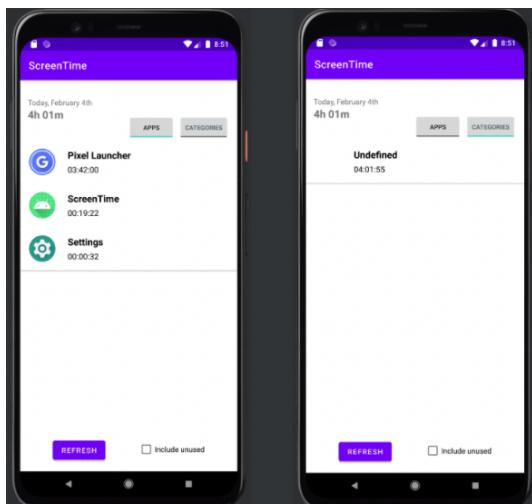


Abbildung 6: Aktuelles Datum und ViewFlipper

(4) Visualisierung der Daten Zur visuellen Aufbereitung der Bildschirmzeiten wurde die Bibliothek *MPAndroidChart*[6] verwendet. Leider ist die Dokumentation recht veraltet und fehlerhaft, letztendlich hat es aber doch funktioniert, wie Abbildung 7 zeigt.

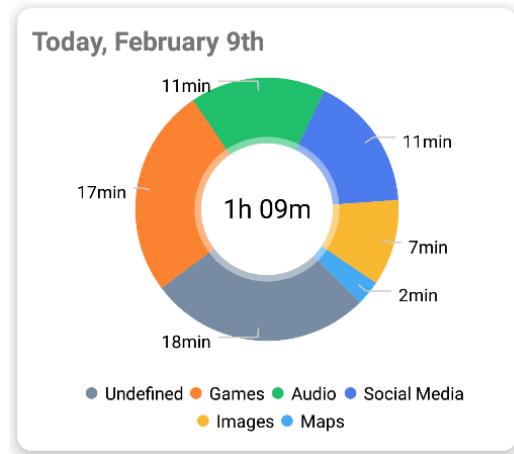


Abbildung 7: Pie-Chart der aktuellen Bildschirmzeit

In einer späteren Version der App sollen zusätzlich die Daten der vergangenen Woche und des vergangenen Monats ausgewertet werden, um mit Diagrammen anschaulich Durchschnitte und Trends darstellen zu können.

(5) Limits Limits können in der `LimitActivity` bearbeitet werden. Hier werden in einem ExpandableListView [2] alle Apps unter ihren jeweiligen Kategorien angezeigt. Mit einem Klick (Tap) auf eine App in dieser Liste öffnet sich der in Abbildung 8 zu sehende Dialog[1]. Hier kann ein Limit für die App festgelegt, geändert oder gelöscht werden. Dieses Limit wird dann in der zentralen App Liste gespeichert, so dass alle Activities darauf zugreifen können. Die gesetzten Limits werden auch auf der Übersichtsseite im Abschnitt *Limits* angezeigt.

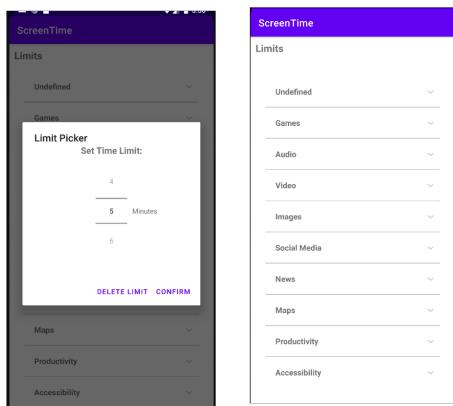


Abbildung 8: Dialog, um das App Limit festzulegen und ExpandableListView der Kategorien und ihrer Apps

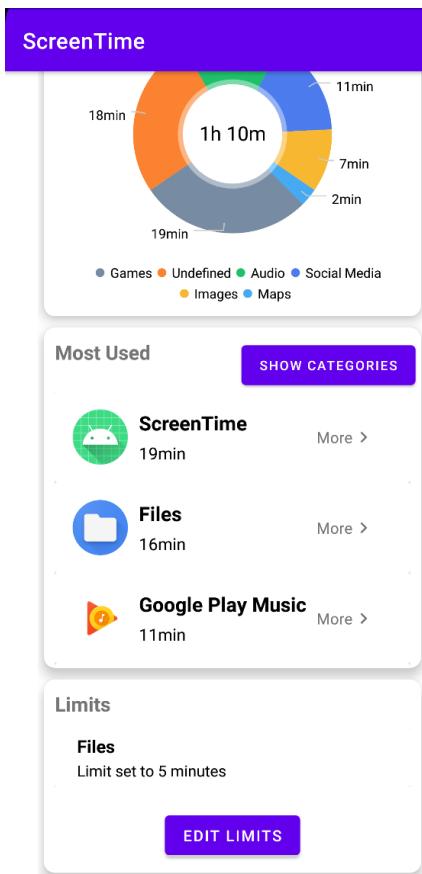


Abbildung 9: Übersicht mit Diagramm, den am häufigsten genutzten Apps und den festgelegten Limits

(6) *App Details* Mit einem Klick (Tap) auf ein App-Item auf der Übersichtsseite wechselt die

App in die *AppDetailsActivity*. Hier werden noch einmal alle Details zu der ausgewählten App angezeigt und man kann auch von hier den oben beschriebenen Dialog öffnen und ein Limit für die jeweilige App festlegen. In einer zukünftigen Version der App sollen hier auch ein Tagesdurchschnitt der Nutzungszeit und ein Diagramm der Nutzung in der vergangenen Woche für die jeweilige App angezeigt werden.

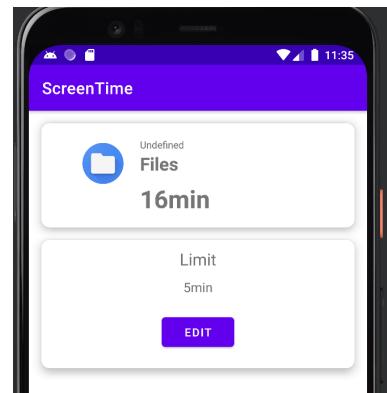


Abbildung 10: Alle Infos zur gewählten App

(7) *Notifications* Wirklich nützlich ist ein festgelegtes Limit nur, wenn man mitbekommt, dass es überschritten wurde.

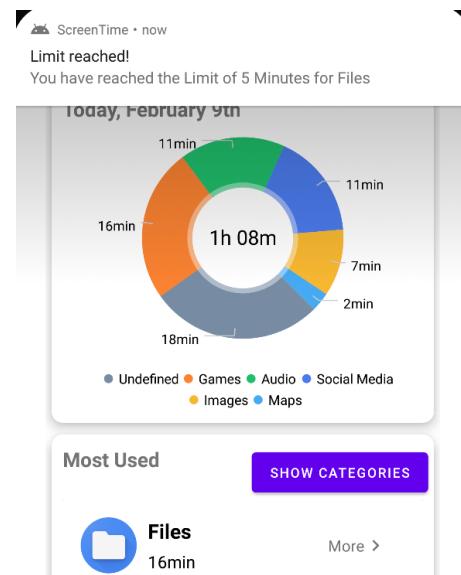


Abbildung 11: Notification, dass das Limit für eine App erreicht wurde

Zu diesem Zweck wäre es ideal, wenn auch bei geschlossener ScreenTime App im Hintergrund geprüft wird, ob eine App ihr Limit gerade überschreitet. Leider ist es mir nicht gelungen, diese Funktionalität im vorgegebenen Zeitrahmen zu implementieren. Dazu müsste ich zum einen die Limits der Apps so speichern, dass sie auch außerhalb der ScreenTime App verfügbar sind, und zum anderen müsste eben dauerhaft die Nutzung aller Apps geprüft werden. Das Speichern der Limits würde ich nach meinen jetzigen Recherchen wahrscheinlich mit der Room Datenbank [4] realisieren. Die Prüfung, ob eine App ihr Limit überschritten hat, funktioniert momentan nur innerhalb der ScreenTime App. Wurde ein Limit überschritten, wird der User, wie in Abbildung 11 zu sehen, mit einer Notification benachrichtigt. Diese Funktionalität müsste dann in einer nächsten Version noch ausgebaut werden.

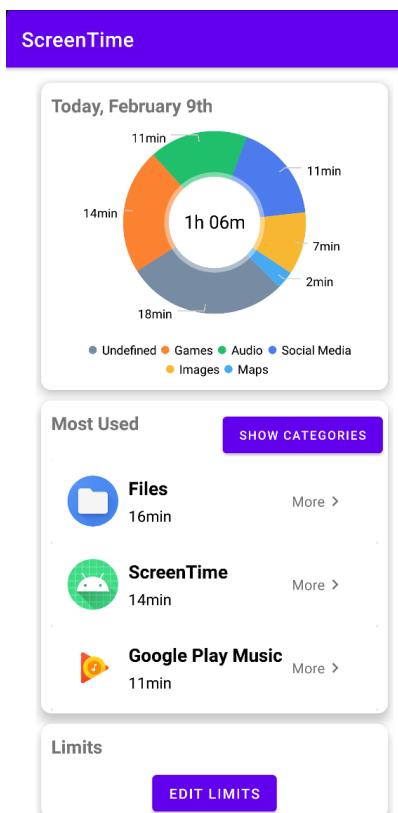


Abbildung 12: Übersichts-Seite der ScreenTime App

(8) *Übersicht* Abbildung 12 zeigt die komplette zentrale Übersichts-Seite der ScreenTime App mit den Bereichen Today, Most Used und Limits

IV AUSWERTUNG

Insgesamt bin ich mit dem Ergebnis meiner Arbeit sehr zufrieden. Ich habe in diesem Projekt, abgesehen von den Übungen zur Vorlesung, zum ersten Mal mit Android und in Android-Studio gearbeitet und auch Kotlin ist für mich neu gewesen. Erst einmal einen Überblick über die ganzen verschiedenen Layout-Elemente und Konzepte zu bekommen, um zu entscheiden, was für meine App die richtige Herangehensweise sein könnte, war zwar eine Herausforderung, hat mir aber auch großen Spaß gemacht. Anfangs hatte ich den Zeitaufwand doch ein wenig unterschätzt, da die ersten Schritte noch recht einfach und teilweise auch aus den Übungen schon bekannt waren. Doch da ich zum einen etwas mit der veralteten Dokumentation von MPAndroidChart zu kämpfen hatte und außerdem mitten im Projekt noch einmal meine ganze Klassenhierarchie auf den Kopf gestellt und die Layouts geändert habe, bin ich am Ende doch ein wenig in Zeitnot geraten. Ich denke aber, dass ich das Implementieren der Warnung außerhalb der ScreenTime App zu Übungszwecken noch nachholen werde, da ich nun auch wissen will, wie das funktioniert. Die Architektur der App ist mir denke ich gut gelungen. Kotlin bietet hier tatsächlich viele interessante Ansätze und Datenstrukturen, die Spaß machen und recht intuitiv sind.

LITERATUR

- [1] GoogleDevelopers. Dialogs,. URL <https://developer.android.com/guide/topics/ui/dialogs>. [Accessed: 7.2.2022].
- [2] GoogleDevelopers. Expandablelistview, . URL <https://developer.android.com/reference/android/widget/ExpandableListView>. [Accessed: 7.2.2022].
- [3] GoogleDevelopers. Create dynamic lists with recyclerview, . URL <https://developer.android.com/guide/topics/ui/layout/recyclerview>. [Accessed: 1.2.2022].
- [4] GoogleDevelopers. Save data in a local database using room, . URL <https://developer.android.com/training/data-storage/room>. [Accessed: 9.2.2022].
- [5] GoogleDevelopers. Viewflipper, . URL <https://developer.android.com/reference/android/widget/ViewFlipper>. [Accessed: 3.2.2022].
- [6] Philipp Jahoda. Mpandroidchart. URL <https://weeklycoding.com/mpandroidchart/>. [Accessed: 6.2.2022].