29th Dec. 2019

Alertless design

# PURPOSE

The purpose of this document is to provide the developers of Alertless app a clear understanding of the requirements, underlying architecture, involved components, system dependencies, development roadmap and testing scenarios for the application to be developed.

# OVERVIEW

Most of the existing mobile apps today generate a lot of notifications whether the app is related to social media, messaging, video streaming, e-commerce, news media, etc. Unlike previous days today a single app can generate a variety of notification like for offers, order updates, app upgrades, season sales, etc.

## Problem Statement

Even though these notifications are to provide an information or an update to the users regarding the corresponding app, sometimes the users are not able to go through the hundreds of notifications piling up in the notification bar and only want the urgent notifications to show in the notification bar. One way is to go through individual app’s notification settings and disable the notifications, but one disadvantage of this approach is that in case user want to get the notifications again, the user will have to again go to the app settings and enable it. Doing it for one app seems easy but when you have a large number of apps in your mobile the task of disabling/enabling becomes quite hectic. Also, users might want a particular group of notifications to be disabled for a specific period of time or occasion and enabled the remaining time. This is where the Alertless app comes into picture which provides a solution to both the problems.

## Proposed Solution

The Alertless app will provide users a way to centrally control (enable/disable) notifications of all the apps installed in his/her system on a profile by profile basis. A profile defines an event or an occasion for which the user wants to enable/disable a certain group of notifications. Below are the features that will be supported by the Alertless app:

1. There will be a configurable schedule associated with every profile which will determine how long a profile will be active.
2. The schedule can be periodic as well as non-periodic.
3. Users will be able to create multiple profiles like vacation, office, travel, home, etc.
4. If multiple profiles will be active the overall effect on notifications will be additive.

# DesigN

## Database Design

In this section, we will discuss the choice of database, entities and their relations.

#### Choice of Database

Given problem involves a well-defined set of system entities like profile, schedule, silent apps, etc. which are related to each other. This motivates towards using Relational Database for our application. Given the current scope of the problem a local database would suffice the requirements. Nowadays, android libraries like “Room” provide a great abstraction for communicating to an SQLite database removing all the boilerplate code, so we would be going forward with that.

#### Entities

A screenshot of a cell phone

Description automatically generated

## Application Design

#### Entities (Referred from Database design)

#### ProfileDetails

#### AppDetails

#### ProfileApp

#### DailyTimeRange

#### Schedule

#### WeekSchedule (extends Schedule)

*Set Bits :* [*https://stackoverflow.com/questions/4674006/set-specific-bit-in-byte*](https://stackoverflow.com/questions/4674006/set-specific-bit-in-byte)

#### ProfileSchedule

#### Model Classes

1. ScheduleType (Enum)
2. Profile Class

* ProfileDetails
* List<AppDetails>
* List<Schedule>

#### NotificationListenerService

NotificationListenerService (NLS)provided by the Android JDK helps us detect and manage other apps notifications. Whenever notifications are posted or removed NLS callback methods are called. Our system will override the NLS callback methods to take custom actions depending on which profiles are active.

#### Interaction among the services (Data flow)

The various Services/Activities involved in the system are below:

1. ProfileEditor
2. SchedulePicker
3. AppSelector
4. NotificationManager

The ProfileEditor creates a Profile Object from below details:

* ProfileDetails (from User)
* List<AppDetails> (from AppSelector Service)
* List<Schedule> (from SchedulePicker Service)

#### RoadMap

##### Development of Database Entities/relations classes

##### Development of Model Classes

##### Development of DAO Layer

##### Development of MainActivity which lists profiles

##### Development of ProfileEditActivity

##### Development of SchedulerActivity

##### Development of AppSelectorActivity

##### Development of AlertNotfiyService

##### Testing the App

##### For week schedule check date range ms starts with 12:00 AM

##### **See how to see logs in android phone for the Alertless app**

##### **Check whether base dao methods for find can be made protected**

## Challenges

#### Storing model and entity classes separately

#### Separation of concerns so that each activity manages its own DAO operations.

#### Having a constant TimeRangeModel Object whose value is modified and hard to debug

#### Referencing two tables with a single foreign key (<https://stackoverflow.com/questions/7844460/foreign-key-to-multiple-tables> )

## Code to Address

#### Fix resource leak in AppFetchTask

#### Null Checks at different layers <https://www.baeldung.com/java-avoid-null-check>

#### On pressing back button state is lost, see Activity lifecycle to maintain states.

#### Add loading GIF when loading all apps in AppSelectorActivity

#### For date range, check if date shows in date picker if the start date is gone.

#### Unsaved profile name delete fails

#### Animation when moving enabled/disabled items together in recycler list

#### Check diff between top and bottom back button, which one is better ?

#### Popup dialog for delete profile.

#### Do not allow special character in profile names and limit length to 20 char

#### After profile save, return to main activity

#### Allow profile deletion from main activity

#### Show app icons in app selector activity

#### Improve code for update cases like multirange schedule so that, for update we don’t have to delete and then do insert (write own diffutils)

#### Check if we need to recover notification when profile is turned off

#### Tracker to see, how many notifications have been silent on a day.

## Good to have

* Support for time zones while storing date

References:

1. <https://medium.com/androiddevelopers/room-time-2b4cf9672b98>
2. <https://stackoverflow.com/questions/7363112/best-way-to-work-with-dates-in-android-sqlite>
3. <https://stackoverflow.com/questions/2400955/how-to-store-java-date-to-mysql-datetime>

* Toggle mode for Silent Enable/Disable
* Use LiveData with Room so that UI refreshes as database changes
* Make a generic Repository class, All repositories extend that.
* Provide option for duration in week schedule selector
* Customize user preference for default radio selection

## References

### Dependency Injection:

<https://proandroiddev.com/how-to-dagger-2-with-android-part-1-18b5b941453f>

### Transaction across multiple Daos

<https://developer.android.com/reference/android/arch/persistence/room/RoomDatabase.html#runintransaction>

### Nested Daos in room

<https://stackoverflow.com/questions/46767224/android-room-daos-with-nested-daos>

### Foreign Key constraints

<https://www.sqlitetutorial.net/sqlite-foreign-key/>

### Alert Dialog:

<https://stackoverflow.com/questions/2115758/how-do-i-display-an-alert-dialog-on-android>

<https://www.jdoodle.com/execute-sql-online/>

create table multi\_range\_schedule(id int, date\_schedule\_id int, date\_range\_id);

insert into multi\_range\_schedule values(1, 22, 4);

insert into multi\_range\_schedule values(1, 22, 5);

insert into multi\_range\_schedule values(1, 22, 6);

insert into multi\_range\_schedule values(1, 33, 4);

insert into multi\_range\_schedule values(1, 44, 5);

insert into multi\_range\_schedule values(1, 44, 6);

insert into multi\_range\_schedule values(1, 55, 5);

insert into multi\_range\_schedule values(1, 55, 6);

select date\_schedule\_id, count (date\_range\_id)

from multi\_range\_schedule

where date\_schedule\_id in

(select date\_schedule\_id

from multi\_range\_schedule where date\_range\_id in (5,6)

group by date\_schedule\_id

having count(distinct date\_range\_id) = 2) group by date\_schedule\_id having count (distinct date\_range\_id) = 2;