Running Tracker

To build the Running Tracker as a robust and high-quality Android app, I had to design it with separation of concerns in mind by making use of different components, all declared in the manifest. Furthermore, it was important to handle the expected interruptions from the user when they might switch between apps in a suitable manner. Moreover, since mobile devices are resource-constrained, the operating system might destroy any running apps due to a lack of memory. Hence it was important to store all memory in a secure database, as opposed to in the individual app components. In conclusion, the Model-View-ViewModel (MVVM) design pattern was used for the architecture of the app, as it was deemed the most suitable – it allowed me to reuse components without having to refactor the code base. The Room persistence library was used for the database as it provides convenience annotations that minimise repetitive and error-prone boiler-plate code.

Diagram

Description automatically generated

Figure 1 (Android Room with a View - Java, n.d.)

The architecture depicted in Figure 1 was used as a baseline to design the Running Tracker as it allowed for the usage of different modules, compartmentalised in a way that each one only depended on the one below it. It implements the MVVM pattern where the Room Database (MyRoomDatabase.java) is the Model and the Activities (MainActivity.java, StatisticsActivity.java and ViewRun.java) are the View.

Graphical user interface, text, application, chat or text message

Description automatically generated

Figure 2 (Save data in a local database using Room, 2020)

In alignment with Figure 2 which displays the components in Room, the entity Run.java was created for the database-table ‘runs’, containing all the values of the runs, such as e.g., speed, distance and rating. The entity implements Serializable to ensure that instances of the entity (entries in the database) could be sent between activities through intents. It also implements getters and setters for the values of the Run-objects to be able to display them to the user in the Views. Furthermore, a data access object (DAO) (RunDao.java) was created by the database which maps SQL queries to functions that get entities and entity-values from the database and updates the database according to changes made by the user.

The Room Database (MyRoomDatabase.java) serves as an access point to the underlying SQLite database, creating a database if an instance doesn’t already exist. If it does, it will simply use a DAO-query to clear it (delete entries). The Repository (MyRepository.java) class was added as a way of interacting with the Room Database (MyRoomDatabase.java) on behalf of the ViewModel (RunViewModel.java). It implements methods that use the DAO to query the database, returning LiveData objects to the ViewModel. Consequently, it provides an abstraction so the User Interface (UI), i.e., the activities, are not required to know where the data comes from. Another advantage of using a ViewModel is that ViewModel-objects survive recreations of Activities due to interruptions.

The repository uses an ExecutorService to execute DAO-queries on a thread at some time in the future. For time-sensitive queries, it makes use of asynchronous computation by creating Futures which execute DAO-queries where the executor waits until the computation has finished before the result is retrieved using ‘.get()’. Finally, the repository also implements a DAO-query returning a *cursor* pointing to the entire ‘runs’-table to be used in the query() function of the ContentProvider (RunProvider.java), allowing other applications to access the data of the Running Tracker with a *read-only* permission.

Diagram

Description automatically generated

Figure 3 (Activity, 2020)

The main activity (MainActivity.java) is the activity that is launched when the app starts. In the onCreate() callback method, when the activity moves from launched (blue state in Figure 3) to running state (green state in Figure 3), it fills the recycler view using an adapter (RunAdapter.java) that gets data (Run-entries) from the database using the ViewModel (RunViewModel.java). It also checks if the activity has been interrupted in the onCreate() and if yes, it restores all the values displayed to the user according to the Bundle savedInstanceState. In onSaveInstanceState(), all these important values are added to the bundle just before the activity is *temporarily* destroyed on interruption.

Diagram

Description automatically generated

Figure 4 (Bound services overview, 2020)

When the user clicks the startButton, the service (RunningService.java) that tracks the user’s movement is started. Firstly, main *binds* to the service to set up the communication where the updated GPS location of the user is sent back to the activity from the service through broadcasting events. Secondly, the activity *starts* the service. Therefore, onStartCommand() is one of the first callback methods called when a bound service is started (see Figure 4 to see callback method visualisation). In onStartCommand(), the service starts a notification which is a way of letting the user know when a service is running in the background if the activity that started it is no longer visible to them. The Location Manager is also created then, using the LocationListener (MyLocationListener.java) which gets updates on the GPS location of the user every 5 seconds or if a minimum of 5m have been “travelled”. When the service is first created, onCreate() is called where the inner class Tracker.java is created which “runs” the service on a new Thread. As the service is running, it gets updated locations from the Location Listener and creates a FutureTask that sends the location back to the activity by broadcasting it in an ICallback event. The callback is executed on another thread as a FutureTask to be able to sleep until it is done, ensuring that a new broadcast isn’t begun (beginBroadcast()) before the previous one has finished (finishBroadcast()).

When the user clicks on an item in the RecyclerView in the MainActivity, the activity starts the ViewRun activity for result and sends the selected run to the new activity in an intent (startActivityForResult(intent, 1)). In ViewRun, the user can view all the values of the Run and annotate them, e.g., type of exercise (running, jogging or walking based on speed), rating of run, distance and time. As ViewRun is finished, the annotations are sent back to the MainActivity as the *result* and using ViewModel, the changes are pushed to the database. The user can also delete a run in the ViewRun activity which sends back the id of the run to the MainActivity where it is deleted using the ViewModel.

A run is completed when the user clicks the finishButton and it is first stopped (stopService()) and then unbound (see Figure 4 for last callback methods unbindService() and onDestroy()). Then, updateScreenValues() is called to display the time, distance and speed of the completed run to the user on the UI (the View in MVVM). The average speed of it is compared against the average speed of the fastest run so far and if it is higher, an alert is shown to the user to let them know of this. At the bottom of the screen in the MainActivity, the user can also see the total distance they’ve “travelled” so far today in km.

To see an overview of running statistics, the user might click on the statisticsButton and see a line graph visualising the number of km travelled per run with distance on the y-axis and run on the x-axis, starting from the first run and stretching to the latest one. If the user is currently running they might view statistics or leave the application temporarily without interrupting the run. When the user re-enters the application, it should be restored accordingly but if they enter through the notification after closing the app, it will not be restored properly which is a weakness of the app functionality. A future improvement on the app would be to manage the lifecycle of it in that case.

Sources

Android. n.d. Android Room With A View - Java. [online] Available at: <https://developer.android.com/codelabs/android-room-with-a-view#1> [Accessed 4 January 2021].

Android. 2020. Activity. [online] Available at: <<https://developer.android.com/reference/android/app/Activity>> [Accessed 4 January 2021].

Android. 2020. Bound services overview. [online] Available at: <<https://developer.android.com/guide/components/bound-services>>[Accessed 4 January 2021].

Android. 2020. Save Data In A Local Database Using Room. [online] Available at: <https://developer.android.com/training/data-storage/room> [Accessed 4 January 2021].