LAPORAN PRAKTIKUM
PEMROGRAMAN BERBASIS MOBILE
PERTEMUAN KE-10

Disusun Oleh :

NAMA	**: Raden Isnawan Argi Aryasatya**
NIM	**: 195410257**
JURUSAN	**: Informatika**
JENJANG	**: S1**
KELAS	**: 5**

Laboratorium Terpadu

Sekolah Tinggi Manajemen Informatika Komputer

AKAKOM

YOGYAKARTA

2021

PERTEMUAN KE-10
(COROUTINE DAN ROOM)

TUJUAN
Mahasiswa dapat membuat aplikasi yang menerapkan coroutines dan Room.

DASAR TEORI

1. Selamat datang

Coroutines
Di Kotlin, coroutine adalah cara untuk menangani task yang sudah berjalan lama secara elegan dan efisien. Kotlin coroutine memungkinkan kita mengonversi kode berbasis panggilan balik ke kode sekuensial. Kode yang ditulis secara berurutan biasanya lebih mudah dibaca dan bahkan dapat menggunakan fitur bahasa seperti pengecualian. Pada akhirnya, coroutine dan callback melakukan hal yang sama: keduanya menunggu hingga hasilnya tersedia dari task yang sudah berjalan lama dan melanjutkan eksekusi.

Coroutines adalah asynchronous
Coroutine berjalan secara independen dari langkah-langkah eksekusi utama program. Eksekusi ini bisa paralel atau pada prosesor terpisah. Bisa juga bahwa sementara sisa aplikasi sedang menunggu input, kita memasukkan sebuah bit pemrosesan. Salah satu aspek penting dari async adalah kita tidak dapat mengharapkan bahwa hasilnya tersedia, sampai kita secara eksplisit menunggu untuk itu.

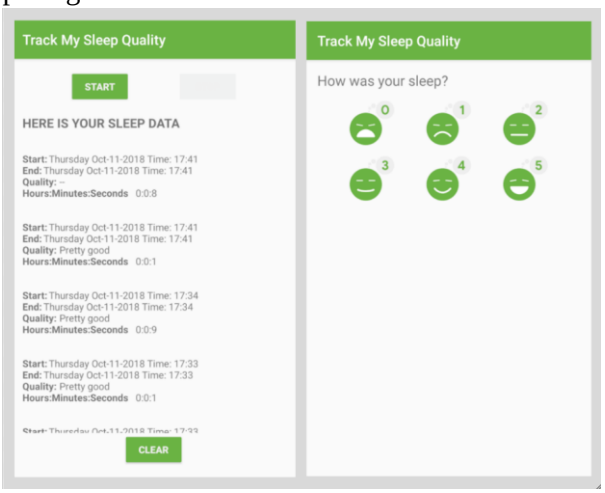
Coroutines are non-blocking.
Non-blocking berarti coroutine tidak memblokir thread utama atau UI. Jadi dengan coroutine, pengguna selalu memiliki pengalaman semulus mungkin, karena interaksi UI selalu mendapat prioritas.

Coroutines menggunakan fungsi suspend untuk membuat urutan kode asinkron.
Penangguhan kata kunci adalah cara Kotlin menandai suatu fungsi, atau jenis fungsi, yang tersedia untuk coroutine. Ketika coroutine memanggil fungsi yang ditandai dengan menangguhkan, alih-alih memblokir sampai fungsi kembali seperti panggilan fungsi normal, coroutine menunda eksekusi hingga hasilnya siap. Kemudian coroutine dilanjutkan di tempat yang ditinggalkannya, dengan hasilnya. Sementara coroutine ditangguhkan dan menunggu hasilnya, ia membuka blokir thread-nya. Dengan begitu, fungsi atau coroutine lain dapat berjalan. Kata kunci yang ditangguhkan tidak menentukan thread bahwa kode berjalan. Fungsi suspend dapat berjalan pada thread latar belakang, atau pada thread utama.

PRAKTIK

2. Ikhtisar Aplikasi

Dalam codelab ini, Anda membuat view model, coroutine, dan bagian tampilan data dari aplikasi TrackMySleepQuality. Aplikasi ini memiliki dua layar, yang diwakili oleh fragmen, seperti yang ditunjukkan pada gambar di bawah ini.



Layar pertama, ditampilkan di sebelah kiri, memiliki tombol untuk memulai dan menghentikan pelacakan. Layar menampilkan semua data tidur pengguna. Tombol Clear secara permanen menghapus semua data yang telah dikumpulkan aplikasi untuk pengguna.

Layar kedua, ditampilkan di sebelah kanan, adalah untuk memilih peringkat kualitas tidur. Dalam aplikasi, peringkat diwakili secara numerik. Untuk tujuan pengembangan, aplikasi menampilkan ikon wajah dan angka yang setara.

Alur pengguna adalah sebagai berikut:

- Pengguna membuka aplikasi dan disajikan dengan layar pelacakan tidur.
- Pengguna mengetuk tombol Start. Ini mencatat waktu mulai dan menampilkannya. Tombol Start dinonaktifkan, dan tombol Stop diaktifkan.
- Pengguna mengetuk tombol Stop. Ini mencatat waktu berakhir dan membuka layar kualitas tidur.
- Pengguna memilih ikon kualitas tidur. Layar ditutup, dan layar pelacakan menampilkan waktu berakhirnya tidur dan kualitas tidur. Tombol Stop dinonaktifkan dan tombol Start diaktifkan. Aplikasi siap untuk satu malam lagi.
- Tombol Clear diaktifkan setiap kali ada data di database. Saat pengguna mengetuk tombol Clear, semua datanya dihapus tanpa bantuan — tidak ada "Anda yakin?" pesan.

---

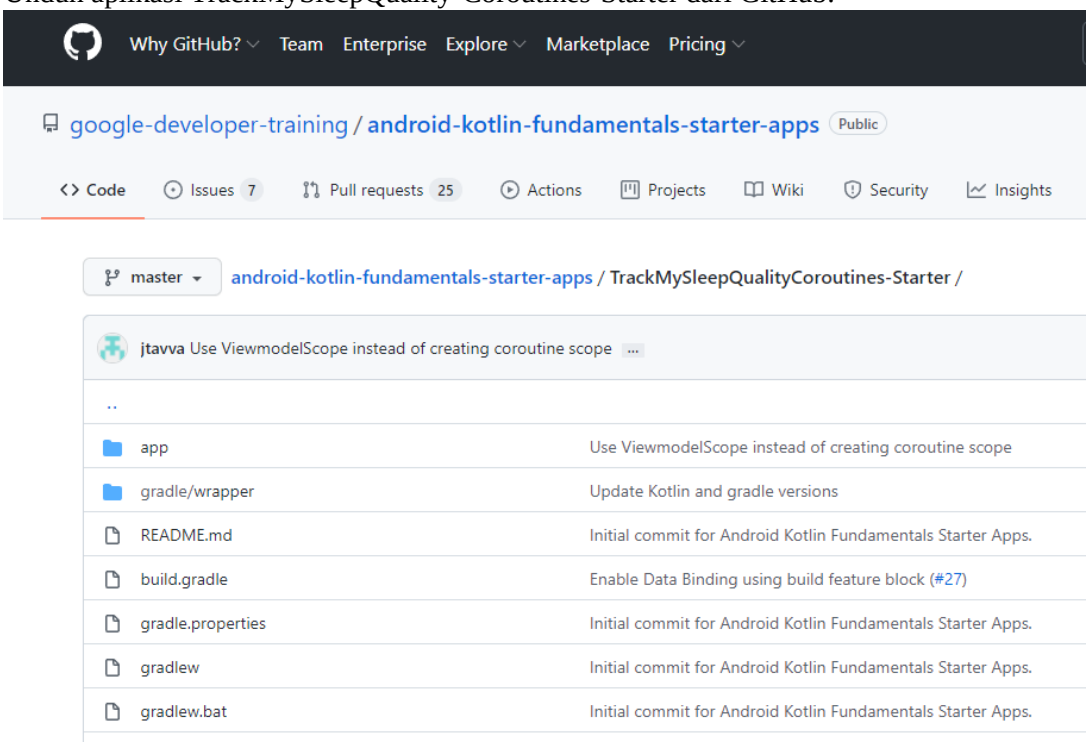
3. Tugas: Memeriksa kode starter

Dalam tugas ini, Anda menggunakan TextView untuk menampilkan data pelacakan tidur yang diformat. (Ini bukan antarmuka pengguna akhir. Anda akan meningkatkan UI di codelab lain.) Anda dapat melanjutkan dengan aplikasi TrackMySleepQuality yang Anda buat di codelab sebelumnya atau mengunduh aplikasi starter untuk codelab ini.

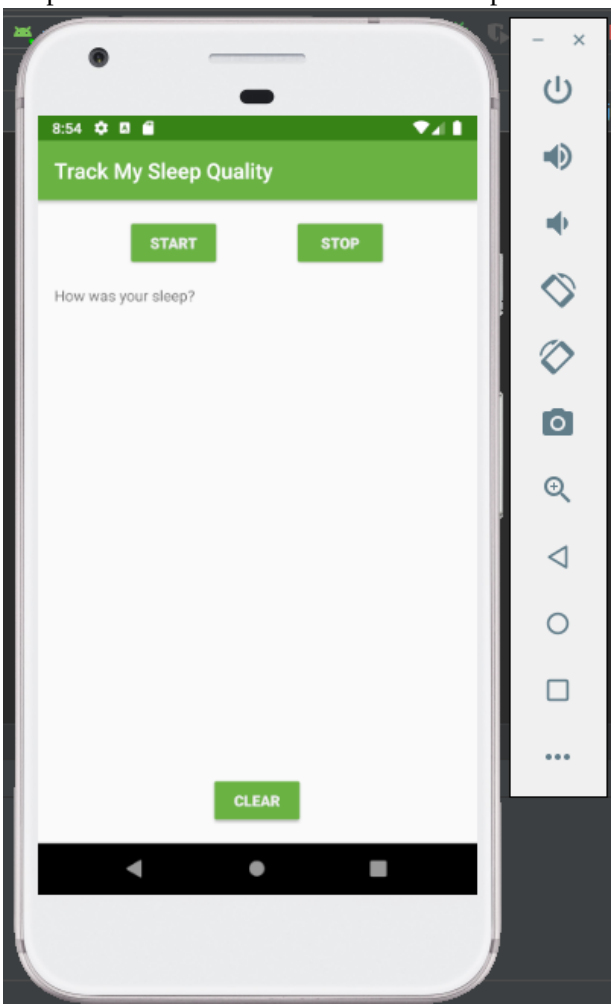
Aplikasi tentang database Room, anda dapat mempelajari di link:
https://developer.android.com/codelabs/kotlin-android-training-room-database#0

Langkah 1: Unduh dan jalankan aplikasi pemula

1. Unduh aplikasi TrackMySleepQuality-Coroutines-Starter dari GitHub.

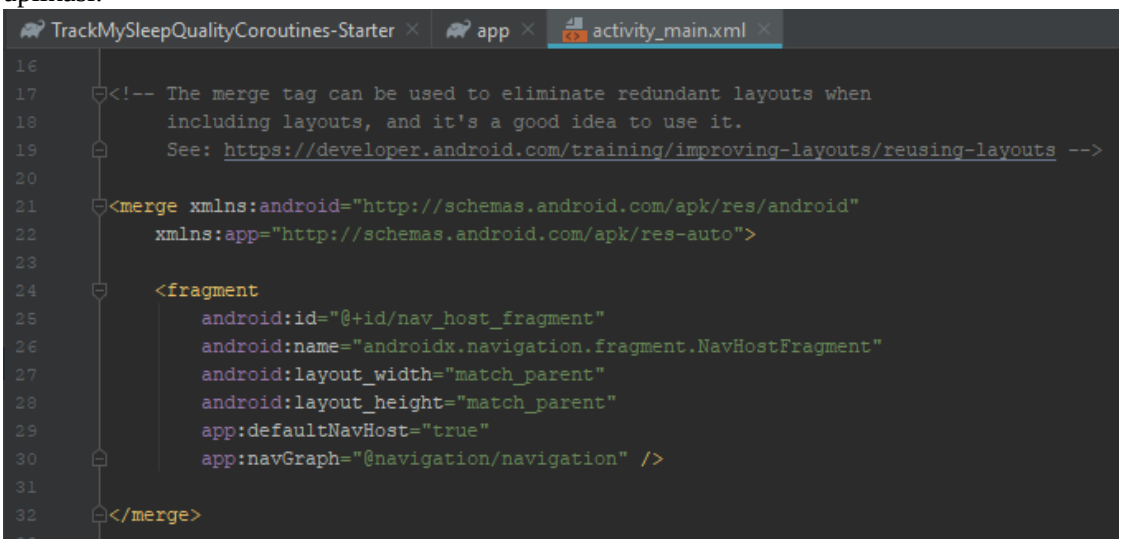


2. Bangun dan jalankan aplikasi. Aplikasi menampilkan UI untuk fragmen SleepTrackerFragment, tetapi tidak ada data. Tombol tidak merespons ketukan.

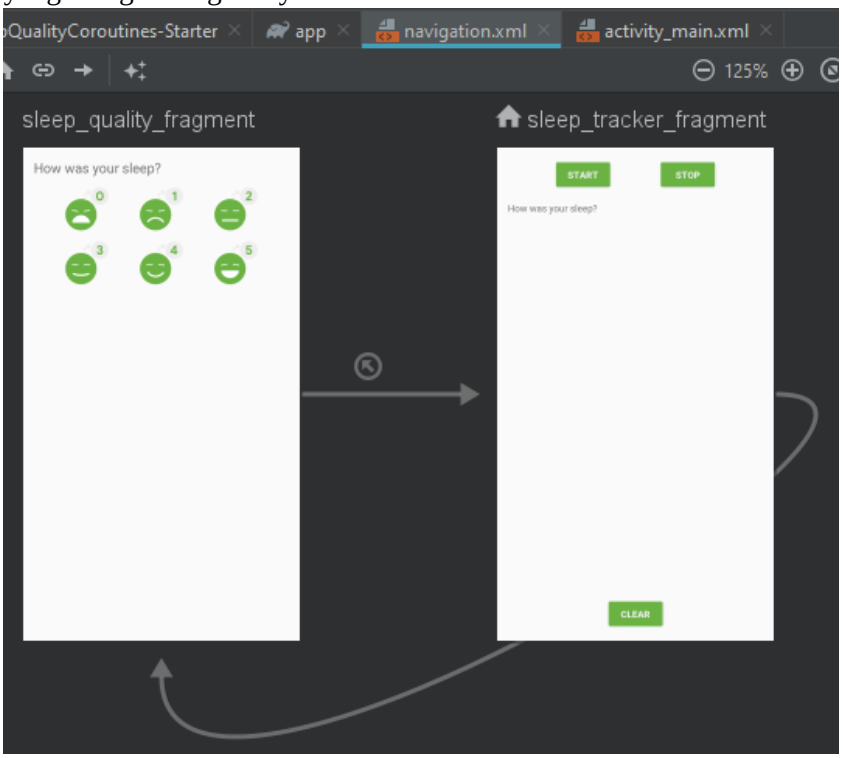


Langkah 2: Periksa kode

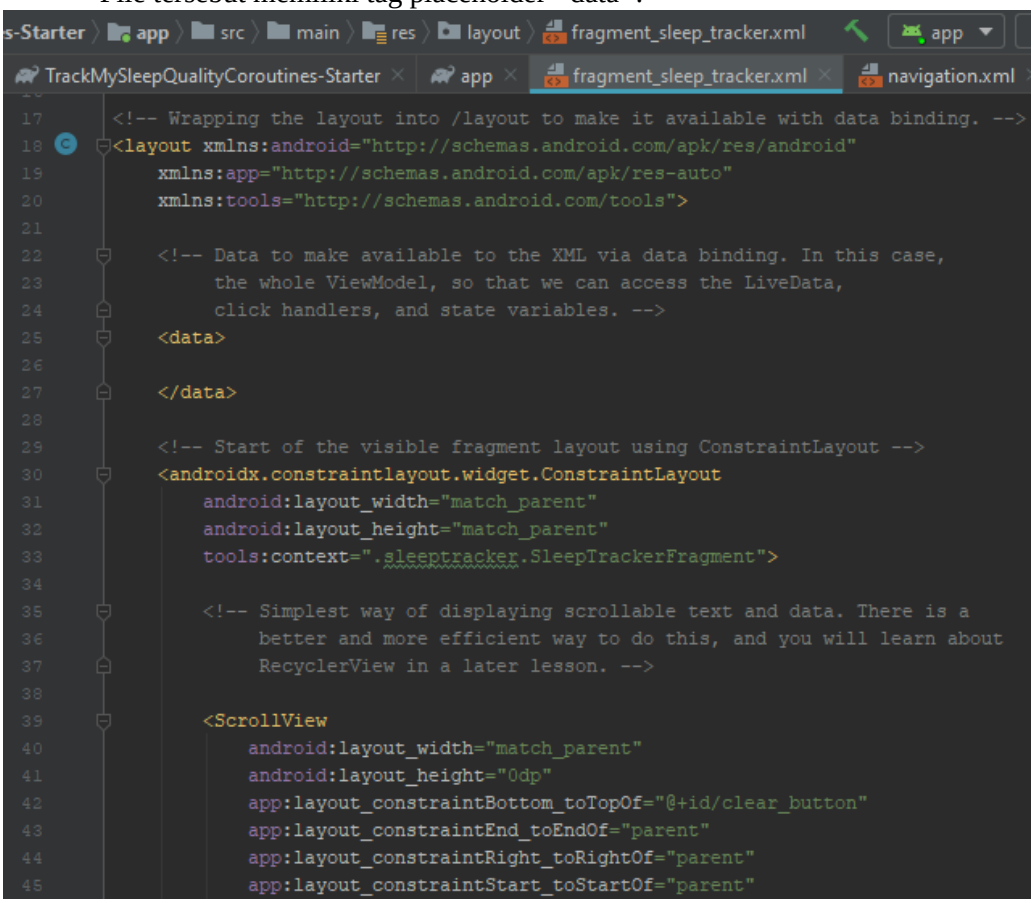
1. Buka res/layout/activity_main.xml. Tata letak ini berisi fragmen nav_host_fragment. Juga, perhatikan tag <merge>. Tag gabungan dapat digunakan untuk menghilangkan tata letak yang berlebihan saat menyertakan tata letak, dan sebaiknya gunakan. Contoh tata letak redundan adalah ConstraintLayout> LinearLayout> TextView, di mana sistem mungkin dapat menghilangkan LinearLayout. Pengoptimalan semacam ini dapat menyederhanakan hierarki tampilan dan meningkatkan kinerja aplikasi.



2. Di folder navigation, buka navigation.xml. Anda dapat melihat dua fragmen dan tindakan navigasi yang menghubungkannya.



3. Di folder layout, buka fragment_sleep_tracker.xml dan klik pada tampilan Kode untuk melihat tata letak XML-nya. Perhatikan hal-hal berikut:
    - Data tata letak dibungkus dalam elemen <layout> untuk mengaktifkan pengikatan data.
    - ConstraintLayout dan tampilan lainnya disusun di dalam elemen <layout>.
    - File tersebut memiliki tag placeholder <data>.



```
            app:layout_constraintTop_toBottomOf="@+id/stop_button">

            <!-- In the TextView, we can access the nightsString LiveData,
                 which keeps it displayed and updated in the TextView
                 whenever it changes. -->

            <TextView
                android:id="@+id/textview"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginStart="16dp"
                android:layout_marginTop="16dp"
                android:layout_marginEnd="16dp"
                android:text="How was your sleep?" />
        </ScrollView>

        <!-- With data binding and LiveData, we can track the buttons' visibility states
             from the ViewModel. The click handler is in the ViewModel as well, and
             you can set it for the Views using this lambda pattern. -->

        <Button
            android:id="@+id/start_button"
            style="@style/SleepButtons"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="16dp"
            android:text="Start"
            app:layout_constraintBaseline_toBaselineOf="@id/stop_button"
            app:layout_constraintEnd_toStartOf="@+id/stop_button"
            app:layout_constraintHorizontal_chainStyle="spread"
            app:layout_constraintStart_toStartOf="parent" />

        <Button
            android:id="@+id/stop_button"
            style="@style/SleepButtons"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="16dp"
            android:layout_marginEnd="16dp"
            android:text="Stop"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toEndOf="@+id/start_button"
            app:layout_constraintTop_toTopOf="parent" />

        <Button
            android:id="@+id/clear_button"
            style="@style/SleepButtons"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="16dp"
            android:layout_marginEnd="16dp"
            android:layout_marginBottom="16dp"
            android:text="Clear"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent" />

    </androidx.constraintlayout.widget.ConstraintLayout>
</layout>
```

Aplikasi pemula juga menyediakan dimensi, warna, dan gaya untuk UI. Aplikasi ini berisi database Room, DAO, dan entitas SleepNight. Jika Anda tidak menyelesaikan codelab "Buat Database Ruangan" sebelumnya, pastikan Anda mempelajari aspek-aspek kode ini sendiri.

---

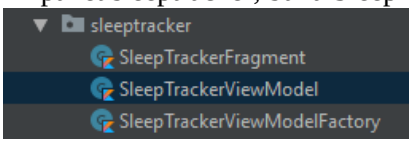
4. Tugas: Menambahkan ViewModel

Sekarang setelah Anda memiliki database dan UI, Anda perlu mengumpulkan data, menambahkan data ke database, dan menampilkan data. Semua pekerjaan ini dilakukan dalam model tampilan. Model tampilan pelacak tidur Anda akan menangani klik tombol, berinteraksi dengan database melalui DAO, dan memberikan

data ke UI melalui LiveData. Semua operasi database harus dijalankan dari thread UI utama, dan Anda akan melakukannya menggunakan coroutine.

Langkah 1: Tambahkan SleepTrackerViewModel

1. Di paket sleeptracker, buka SleepTrackerViewModel.kt.

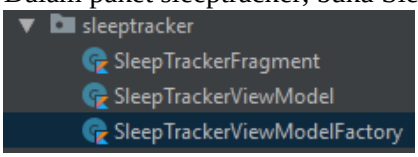


2. Periksa kelas SleepTrackerViewModel, yang disediakan untuk Anda di aplikasi pemula dan juga ditampilkan di bawah. Perhatikan bahwa kelas memperluas AndroidViewModel. Kelas ini sama dengan ViewModel, tetapi mengambil konteks aplikasi sebagai parameter konstruktor dan membuatnya tersedia sebagai properti. Anda akan membutuhkannya nanti.

```
class SleepTrackerViewModel(
        val database: SleepDatabaseDao,
        application: Application) : AndroidViewModel(application) {
}
```

Langkah 2: Tambahkan SleepTrackerViewModelFactory

1. Dalam paket sleeptracker, buka SleepTrackerViewModelFactory.kt.



2. Periksa kode yang disediakan untuk Anda untuk pabrik, yang ditunjukkan di bawah ini:

```
class SleepTrackerViewModelFactory(
        private val dataSource: SleepDatabaseDao,
        private val application: Application) : ViewModelProvider.Factory {
    @Suppress( ...names: "unchecked_cast")
    override fun <T : ViewModel?> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(SleepTrackerViewModel::class.java)) {
            return SleepTrackerViewModel(dataSource, application) as T
        }
        throw IllegalArgumentException("Unknown ViewModel class")
    }
}
```

Perhatikan hal-hal berikut:

- SleepTrackerViewModelFactory yang disediakan mengambil argumen yang sama dengan ViewModel dan memperluas ViewModelProvider.Factory.
- Di dalam pabrik, kode menimpa create (), yang menggunakan tipe kelas apa pun sebagai argumen dan mengembalikan ViewModel.
- Di badan create (), kode memeriksa apakah ada kelas SleepTrackerViewModel yang tersedia, dan jika ada, mengembalikan instance-nya. Jika tidak, kode akan memunculkan pengecualian.

Langkah 3: Perbarui SleepTrackerFragment

1. Di SleepTrackerFragment.kt, dapatkan referensi ke konteks aplikasi. Letakkan referensi di onCreateView (), di bawah binding. Anda memerlukan referensi ke aplikasi tempat fragmen ini dilampirkan, untuk diteruskan ke penyedia pabrik model tampilan.

   Fungsi requirementNotNull Kotlin menampilkan IllegalArgumentException jika nilainya null.

```
val application = requireNotNull(this.activity).application
```

2. Anda memerlukan referensi ke sumber data Anda melalui referensi ke DAO. Di onCreateView (), sebelum return, tentukan dataSource. Untuk mendapatkan referensi ke DAO database, gunakan SleepDatabase.getInstance (application) .sleepDatabaseDao.

```
val dataSource = SleepDatabase.getInstance(application).sleepDatabaseDao
```

3. Di onCreateView (), sebelum return, buat instance viewModelFactory. Anda harus meneruskannya ke dataSource dan aplikasi.

```
val viewModelFactory = SleepTrackerViewModelFactory(dataSource, application)
```

4. Sekarang setelah Anda memiliki pabrik, dapatkan referensi ke SleepTrackerViewModel. Parameter SleepTrackerViewModel::class.java merujuk ke kelas Java runtime dari objek ini.

```
val sleepTrackerViewModel =
        ViewModelProvider(
                owner: this, viewModelFactory).get(SleepTrackerViewModel::class.java)
```

5. Kode Anda yang sudah selesai akan terlihat seperti ini:

```
val application = requireNotNull(this.activity).application

val dataSource = SleepDatabase.getInstance(application).sleepDatabaseDao

val viewModelFactory = SleepTrackerViewModelFactory(dataSource, application)

val sleepTrackerViewModel =
        ViewModelProvider(
                owner: this, viewModelFactory).get(SleepTrackerViewModel::class.java)
```

Inilah metode onCreateView () sejauh ini:

```
    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                              savedInstanceState: Bundle?): View? {

        // Get a reference to the binding object and inflate the fragment views.
        val binding: FragmentSleepTrackerBinding = DataBindingUtil.inflate(
                inflater, R.layout.fragment_sleep_tracker, container, attachToParent: false)

        val application = requireNotNull(this.activity).application

        val dataSource = SleepDatabase.getInstance(application).sleepDatabaseDao

        val viewModelFactory = SleepTrackerViewModelFactory(dataSource, application)

        val sleepTrackerViewModel =
                ViewModelProvider(
                        owner: this, viewModelFactory).get(SleepTrackerViewModel::class.java)

        return binding.root
    }
}
```

Langkah 4: Tambahkan data binding untuk model tampilan

Dengan ViewModel dasar di tempat, Anda harus menyelesaikan penyiapan data binding di

SleepTrackerFragment untuk menghubungkan ViewModel dengan UI.

Di file tata letak fragment_sleep_tracker.xml:

1. Di dalam bagian <data>, buat <variable> yang mereferensikan kelas SleepTrackerViewModel.

```
<data>
    <variable
        name="sleepTrackerViewModel"
        type="com.example.android.trackmysleepquality.sleeptracker.SleepTrackerViewModel" />
</data>
```

Di SleepTrackerFragment.kt:

1. Setel aktivitas saat ini sebagai pemilik siklus proses binding. Tambahkan kode ini di dalam metode onCreateView (), sebelum pernyataan return:

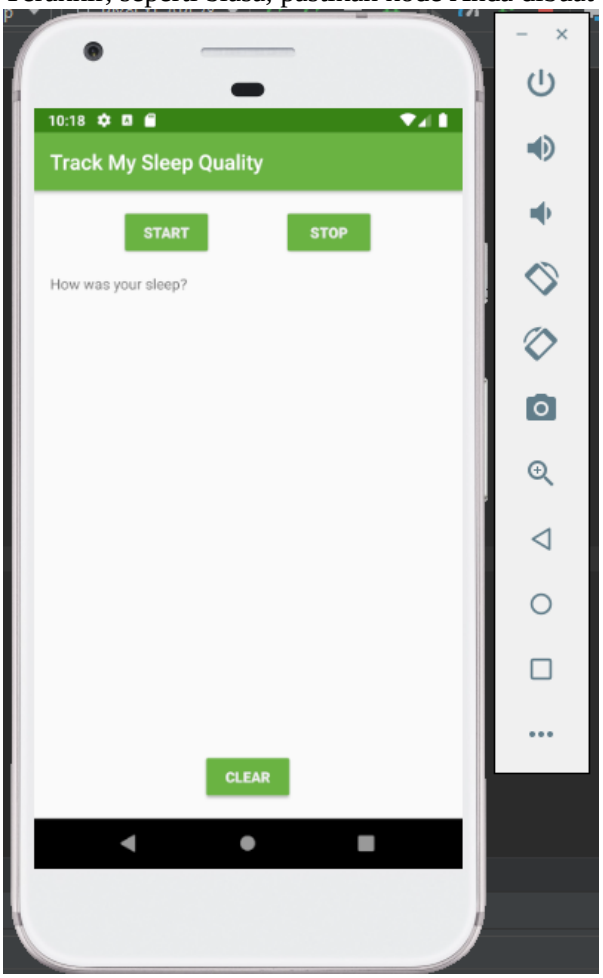
```
binding.setLifecycleOwner(this)
```

2. Tetapkan variabel pengikat sleepTrackerViewModel ke sleepTrackerViewModel. Letakkan kode ini di dalam onCreateView (), di bawah kode yang membuat SleepTrackerViewModel:

```
binding.sleepTrackerViewModel = sleepTrackerViewModel
```

3. Anda mungkin melihat kesalahan, karena Anda harus membuat ulang objek binding. Bersihkan dan bangun kembali proyek untuk menghilangkan kesalahan.

4. Terakhir, seperti biasa, pastikan kode Anda dibuat dan dijalankan tanpa kesalahan.



---

5. Tugas: Mengumpulkan dan menampilkan data

Anda ingin pengguna dapat berinteraksi dengan data tidur dengan cara berikut:

- Saat pengguna mengetuk tombol Start, aplikasi membuat tidur malam baru dan menyimpan tidur malam di database.
- Saat pengguna mengetuk tombol Stop, aplikasi memperbarui malam dengan waktu berakhir.
- Saat pengguna mengetuk tombol Clear, aplikasi menghapus data dalam database.

Langkah 1: Tandai fungsi DAO sebagai fungsi penangguhan

Di SleepDatabaseDao.kt, ubah metode praktis untuk menangguhkan fungsi.

1. Buka database/SleepDatabaseDao.kt, tambahkan kata kunci suspend ke semua metode kecuali getAllNights () karena Room sudah menggunakan utas latar belakang untuk @Query spesifik yang mengembalikan LiveData. Kelas SleepDatabaseDao yang lengkap akan terlihat seperti ini.

```
@Dao
interface SleepDatabaseDao {

    @Insert
    suspend fun insert(night: SleepNight)

    @Update
    suspend fun update(night: SleepNight)

    @Query( value: "SELECT * from daily_sleep_quality_table WHERE nightId = :key")
    suspend fun get(key: Long): SleepNight?

    @Query( value: "DELETE FROM daily_sleep_quality_table")
    suspend fun clear()

    @Query( value: "SELECT * FROM daily_sleep_quality_table ORDER BY nightId DESC LIMIT 1")
    suspend fun getTonight(): SleepNight?

    @Query( value: "SELECT * FROM daily_sleep_quality_table ORDER BY nightId DESC")
    fun getAllNights(): LiveData<List<SleepNight>>
}
```

Langkah 2: Siapkan coroutine untuk operasi database

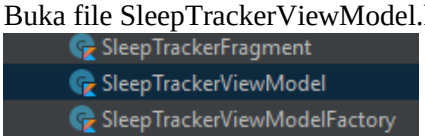
Saat tombol Start di aplikasi Sleep Tracker diketuk, Anda ingin memanggil fungsi di SleepTrackerViewModel untuk membuat instance baru SleepNight dan menyimpan instance di database. Mengetuk salah satu tombol memicu operasi database, seperti membuat atau memperbarui SleepNight. Karena operasi database bisa memakan waktu cukup lama, Anda menggunakan coroutine untuk mengimplementasikan handler klik untuk tombol aplikasi.

1. Buka file build.gradle level aplikasi. Di bawah bagian dependencies, Anda memerlukan dependensi berikut, yang telah ditambahkan untuk Anda.

```
implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:2.2.0"

// Kotlin Extensions and Coroutines support for Room
implementation "androidx.room:room-ktx:$room_version"
```

2. Buka file SleepTrackerViewModel.kt.



3. Tentukan variabel yang dipanggil tonight untuk menahan malam saat ini. Buat variabel MutableLiveData, karena Anda harus bisa mengamati data dan mengubahnya.

```
private var tonight = MutableLiveData<SleepNight?>()
```

4. Untuk menginisialisasi variabel tonight secepat mungkin, buat blok init di bawah definisi tonight dan panggil initializeTonight (). Anda mendefinisikan initializeTonight () di langkah berikutnya.

```
init {
    initializeTonight()
}
```

5. Di bawah blok init, implementasikan initializeTonight (). Gunakan viewModelScope.launch untuk memulai coroutine di ViewModelScope. Di dalam kurung kurawal, dapatkan nilai untuk malam ini dari database dengan memanggil getTonightFromDatabase (), dan tetapkan nilainya ke tonight.value. Anda mendefinisikan getTonightFromDatabase () di langkah berikutnya.

   Perhatikan penggunaan kurung kurawal untuk peluncuran. Mereka mendefinisikan ekspresi lambda, yang merupakan fungsi tanpa nama. Dalam contoh ini, Anda meneruskan lambda ke peluncur coroutine peluncuran. Pembuat ini membuat coroutine dan menetapkan eksekusi lambda tersebut ke petugas operator yang sesuai.

```
private fun initializeTonight() {
    viewModelScope.launch { this: CoroutineScope
        tonight.value = getTonightFromDatabase()
    }
}
```

6. Implementasikan getTonightFromDatabase (). Tentukan sebagai fungsi penangguhan private yang mengembalikan SleepNight nullable, jika tidak ada SleepNight yang saat ini dimulai. Ini membuat Anda mengalami kesalahan, karena fungsinya harus mengembalikan sesuatu.

```
private suspend fun getTonightFromDatabase(): SleepNight? {
```

7. Di dalam tubuh fungsi getTonightFromDatabase (), dapatkan malam ini (malam terbaru) dari database. Jika waktu mulai dan waktu berakhir tidak sama, artinya malam sudah selesai, kembalikan null. Jika tidak, kembalikan malam itu.

```
var night = database.getTonight()
if (night?.endTimeMilli != night?.startTimeMilli) {
    night = null
}
return night
```

   Fungsi penangguhan getTonightFromDatabase () Anda yang telah selesai akan terlihat seperti ini. Seharusnya tidak ada lagi kesalahan.

```
private suspend fun getTonightFromDatabase(): SleepNight? {
    var night = database.getTonight()
    if (night?.endTimeMilli != night?.startTimeMilli) {
        night = null
    }
    return night
}
```

Langkah 3: Tambahkan handler klik untuk tombol Start

Sekarang Anda dapat mengimplementasikan onStartTracking (), pengendali klik untuk tombol Start. Anda perlu membuat SleepNight baru, memasukkannya ke dalam database, dan menetapkannya untuk malam ini. Struktur onStartTracking () akan mirip dengan initializeTonight ().

1. Di SleepTrackerViewModel.kt, mulailah dengan definisi fungsi untuk onStartTracking (). Anda bisa

meletakkan handler klik di bawah getTonightFromDatabase ().

```
fun onStartTracking () {
```

2. Di dalam onStartTracking (), luncurkan coroutine di viewModelScope, karena Anda memerlukan hasil ini untuk melanjutkan dan memperbarui UI.

```
viewModelScope.launch { this: CoroutineScope
```

3. Di dalam peluncuran coroutine, buat SleepNight baru, yang merekam waktu saat ini sebagai waktu mulai.

```
val newNight = SleepNight()
```

4. Masih di dalam peluncuran coroutine, panggil insert () untuk memasukkan newNight ke dalam database. Anda akan melihat kesalahan, karena Anda belum mendefinisikan fungsi insert () suspend ini. Perhatikan bahwa ini bukan insert () yang sama dengan metode dengan nama yang sama di SleepDatabaseDAO.kt

```
insert (newNight)
```

5. Juga di dalam peluncuran coroutine, perbarui tonight.

```
tonight.value = getTonightFromDatabase()
```

6. Di bawah onStartTracking (), definisikan insert () sebagai fungsi penangguhan pribadi yang menggunakan SleepNight sebagai argumennya.

```
private suspend fun insert(night: SleepNight) {
```

7. Dalam metode insert (), gunakan DAO untuk memasukkan malam ke dalam database.

```
database.insert(night)
```

   Perhatikan bahwa coroutine dengan Room menggunakan Dispatchers.IO, jadi ini tidak akan terjadi pada utas utama.

8. Di file tata letak fragment_sleep_tracker.xml, tambahkan handler klik untuk onStartTracking () ke start_button menggunakan keajaiban pengikatan data yang Anda siapkan sebelumnya. Notasi fungsi @ {() -> membuat fungsi lambda yang tidak membutuhkan argumen dan memanggil penangan klik dalam sleepTrackerViewModel.

```
android:onClick = "@{() -> sleepTrackerViewModel.onStartTracking ()}"
```

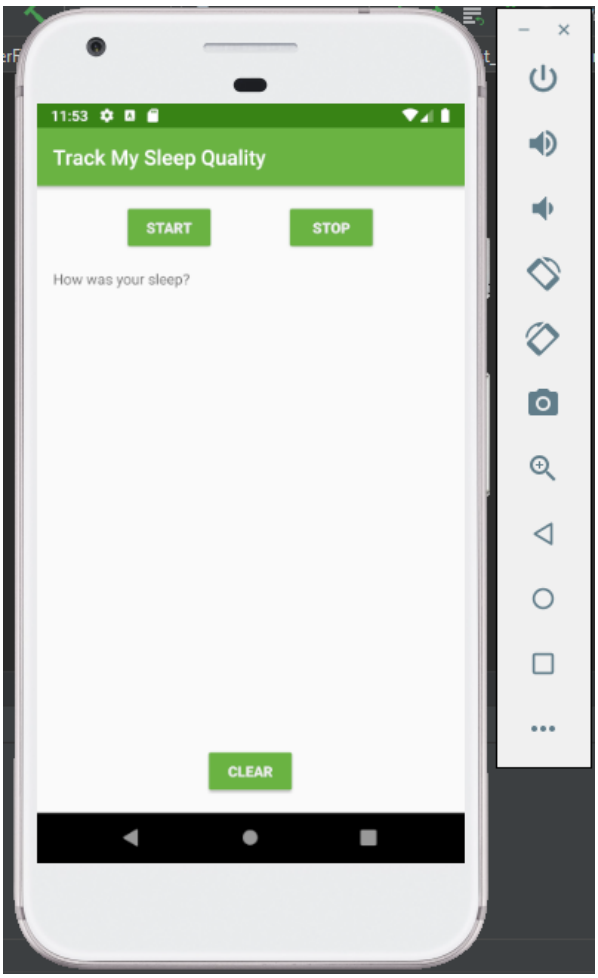
9. Buat dan jalankan aplikasi Anda. Ketuk tombol Start. Tindakan ini membuat data, tetapi Anda belum dapat melihat apa pun. Anda memperbaikinya selanjutnya.

(screenshot aplikasi di halaman selanjutnya)

(screenshot aplikasi di halaman selanjutnya)

(screenshot aplikasi di halaman selanjutnya)

(screenshot aplikasi di halaman selanjutnya)



Langkah 4: Tampilkan data

Di SleepTrackerViewModel.kt, variabel malam merujuk LiveData karena getAllNights () di DAO menampilkan LiveData. Ini adalah fitur Room yang setiap kali data dalam database berubah, malam LiveData diperbarui untuk menampilkan data terbaru. Anda tidak perlu mengatur LiveData secara eksplisit atau memperbaruinya. Room mengupdate data agar sesuai dengan database. Namun, jika Anda menampilkan malam dalam tampilan teks, itu akan menunjukkan referensi objek. Untuk melihat konten objek, ubah data menjadi string berformat. Gunakan peta Transformasi yang dijalankan setiap malam saat menerima data baru dari database.

1. Buka file Util.kt dan hapus komentar kode untuk definisi formatNights () dan pernyataan impor terkait. Untuk menghapus komentar kode di Android Studio, pilih semua kode yang ditandai dengan // dan tekan Cmd + / atau Control + /.

```
//fun formatNights(nights: List<SleepNight>, resources: Resources): Spanned {
//    val sb = StringBuilder()
//    sb.apply {
//        append(resources.getString(R.string.title))
//        nights.forEach {
//            append("<br>")
//            append(resources.getString(R.string.start_time))
//            append("\t${convertLongToDateString(it.startTimeMilli)}<br>")
//            if (it.endTimeMilli != it.startTimeMilli) {
//                append(resources.getString(R.string.end_time))
//                append("\t${convertLongToDateString(it.endTimeMilli)}<br>")
//                append(resources.getString(R.string.quality))
//                append("\t${convertNumericQualityToString(it.sleepQuality, resources)}<br>")
//                append(resources.getString(R.string.hours_slept))
//                // Hours
//                append("\t ${it.endTimeMilli.minus(it.startTimeMilli) / 1000 / 60 / 60}:")
```

```
//                // Minutes
//                append("${it.endTimeMilli.minus(it.startTimeMilli) / 1000 / 60}:")
//                // Seconds
//                append("${it.endTimeMilli.minus(it.startTimeMilli) / 1000}<br><br>")
//            }
//        }
//    }
//    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) {
//        return Html.fromHtml(sb.toString(), Html.FROM_HTML_MODE_LEGACY)
//    } else {
//        return HtmlCompat.fromHtml(sb.toString(), HtmlCompat.FROM_HTML_MODE_LEGACY)
//    }
//}
```

2. Perhatikan formatNights () mengembalikan tipe Spanned, yang merupakan string berformat HTML. Ini sangat nyaman karena TextView Android memiliki kemampuan untuk merender HTML dasar.

```
fun formatNights(nights: List<SleepNight>, resources: Resources): Spanned {
```

3. Buka res > nilai> strings.xml. Perhatikan penggunaan CDATA untuk memformat sumber daya string untuk menampilkan data tidur.

```
<!-- Output to TextView styled with a little HTML -->
<string name="title"><![CDATA[<h3>HERE IS YOUR SLEEP DATA</h3>]]></string>
<string name="start_time"><![CDATA[<b>Start:</b>]]></string>
<string name="end_time"><![CDATA[<b>End:</b>]]></string>
<string name="quality"><![CDATA[<b>Quality:</b>]]></string>
<string name="hours_slept"><![CDATA[<b>Hours:Minutes:Seconds</b>]]></string>
```

4. Buka SleepTrackerViewModel.kt. Di kelas SleepTrackerViewModel, tentukan variabel yang disebut nights. Dapatkan semua nights dari database dan tetapkan ke variabel nights.

```
 private val nights = database.getAllNights()
```

5. Tepat di bawah definisi nights, tambahkan kode untuk mengubah malam menjadi sebuah nightsString. Gunakan fungsi formatNights () dari Util.kt.

   Lewatkan nights ke fungsi map () dari kelas Transformasi. Untuk mendapatkan akses ke sumber daya string Anda, tentukan fungsi pemetaan sebagai memanggil formatNights (). Sediakan nights dan objek Resources.

```
 val nightsString = Transformations.map(nights) { nights ->
     formatNights(nights, application.resources)
```

6. Buka file tata letak fragment_sleep_tracker.xml. Di TextView, di properti android: text, Anda sekarang bisa mengganti string sumber daya dengan referensi ke nightsString.

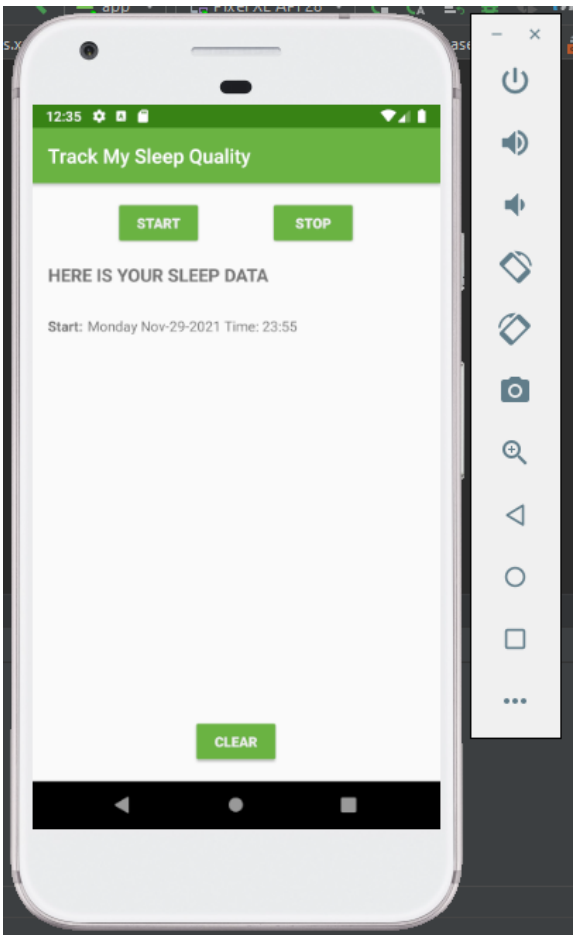
```
 android:text="@{sleepTrackerViewModel.nightsString}" />
```

7. Buat ulang kode Anda dan jalankan aplikasi. Semua data tidur Anda dengan waktu mulai akan ditampilkan sekarang.

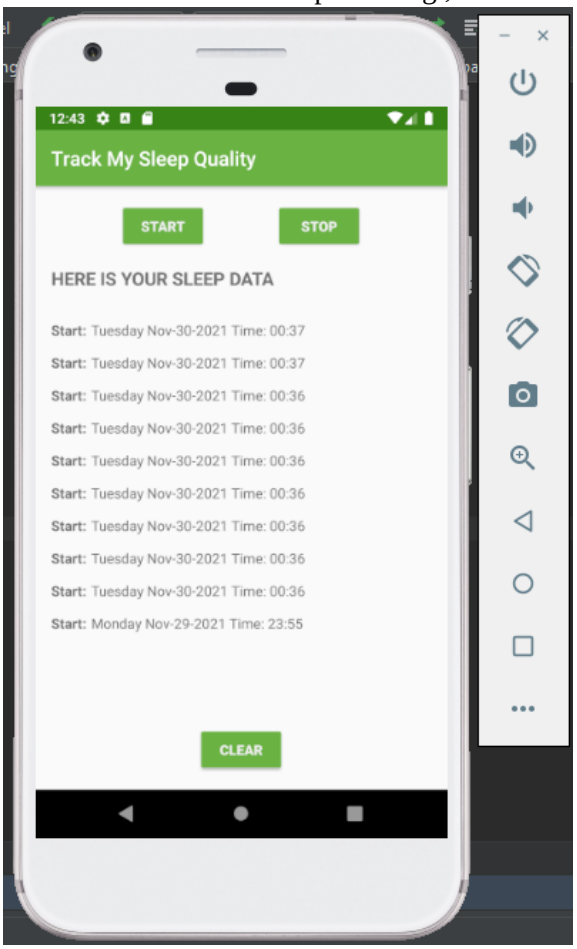
(screenshot aplikasi di halaman selanjutnya)

(screenshot aplikasi di halaman selanjutnya)

(screenshot aplikasi di halaman selanjutnya)



8. Ketuk tombol Start beberapa kali lagi, dan Anda akan melihat lebih banyak data.



Langkah 5: Tambahkan handler klik untuk tombol Stop

Menggunakan pola yang sama seperti di langkah sebelumnya, implementasikan penangan klik untuk tombol Stop di SleepTrackerViewModel.

1. Tambahkan onStopTracking () ke ViewModel. Luncurkan coroutine di viewModelScope. Jika waktu berakhir belum disetel, setel endTimeMilli ke waktu sistem saat ini dan panggil update () dengan data malam. Di Kotlin, sintaks return @label menentukan fungsi tempat pernyataan ini dikembalikan, di antara beberapa fungsi bertingkat.

```
fun onStopTracking() {
    viewModelScope.launch { this: CoroutineScope
        val oldNight = tonight.value ?: return@launch
        oldNight.endTimeMilli = System.currentTimeMillis()
        update(oldNight)
    }
```

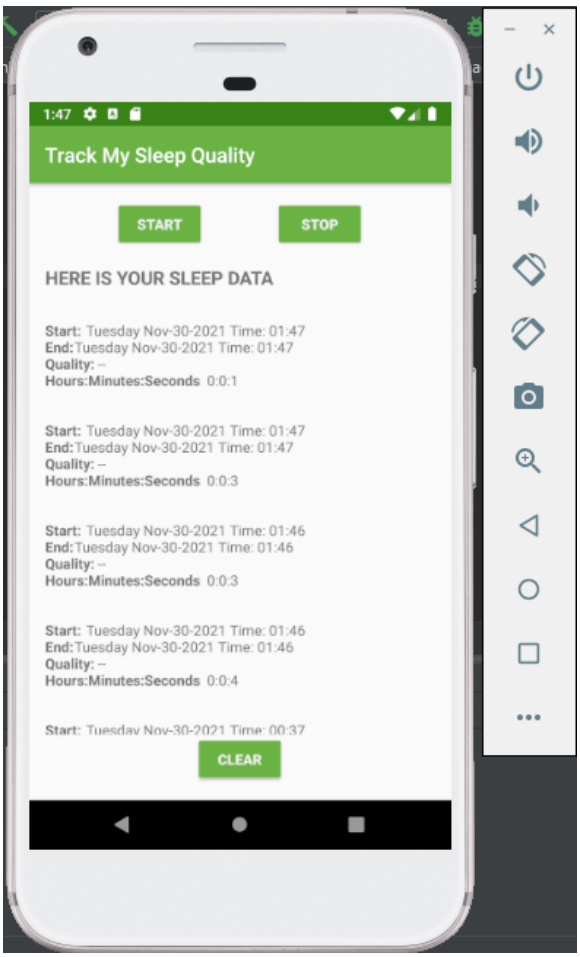
2. Implementasikan update () menggunakan pola yang sama seperti yang Anda gunakan untuk mengimplementasikan insert ().

```
private suspend fun update(night: SleepNight) {
    database.update(night)
}
```

3. Untuk menghubungkan handler klik ke UI, buka file tata letak fragment_sleep_tracker.xml dan tambahkan handler klik ke stop_button.

```
android:onClick="@{() -> sleepTrackerViewModel.onStopTracking()}"
```

4. Buat dan jalankan aplikasi Anda.
5. Ketuk Start, lalu ketuk Stop. Anda melihat waktu mulai, waktu berakhir, kualitas tidur tanpa nilai, dan waktu tidur.



Langkah 6: Tambahkan handler klik untuk tombol Clear

1. Demikian pula, implementasikan onClear () dan clear ().

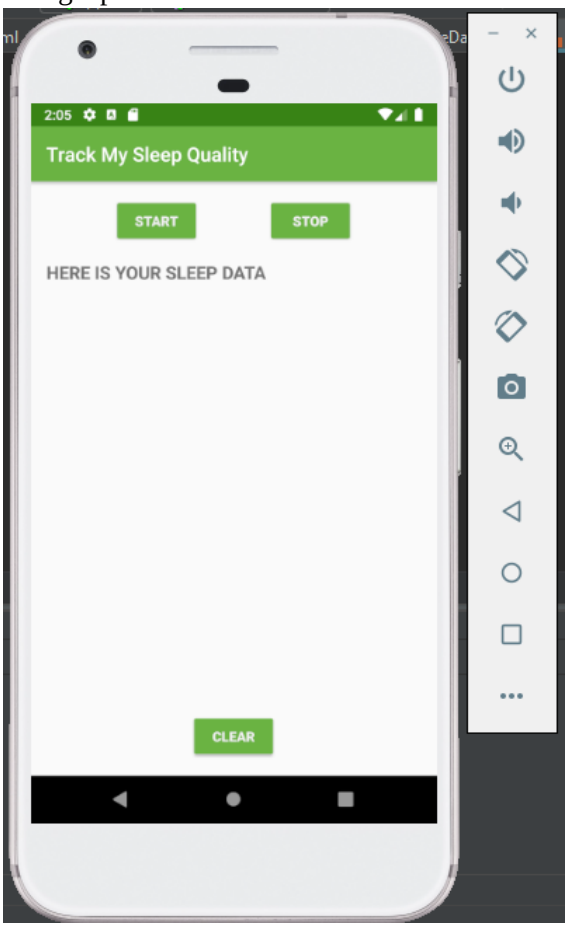
```
fun onClear() {
    viewModelScope.launch { this: CoroutineScope
        clear()
        tonight.value = null
    }
}
suspend fun clear() {
    database.clear()
}
```

2. Untuk menghubungkan handler klik ke UI, buka fragment_sleep_tracker.xml dan tambahkan handler klik ke clear_button.

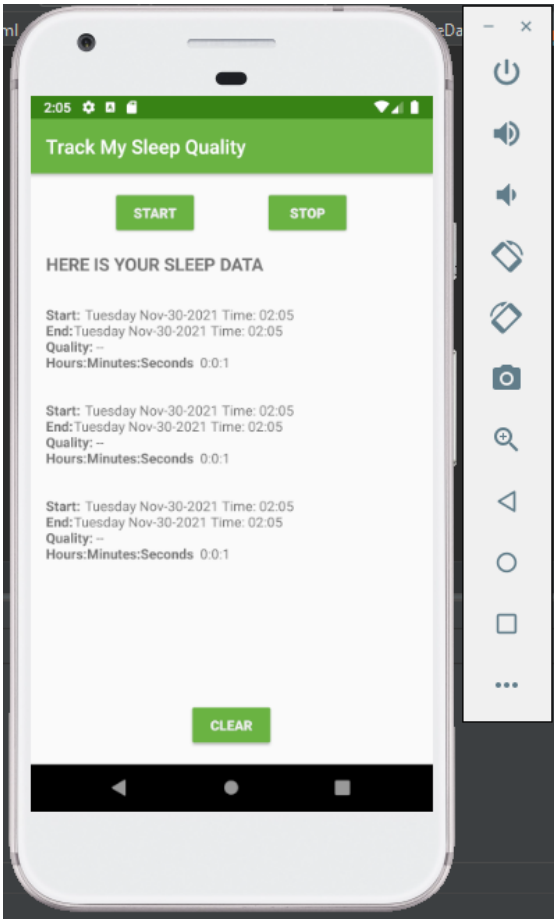
```
android:onClick="@{() -> sleepTrackerViewModel.onClear()}"
```

3. Buat dan jalankan aplikasi Anda.
4. Ketuk Clear untuk menghapus semua data. Kemudian ketuk Start dan Stop untuk membuat data baru.

Menghapus data



Menambah data baru



TUGAS

Buat aplikasi baru dengan mengembangkan project diatas

Di tugas ini, saya akan menambahkan beberapa fitur yang berhubungan dengan livedata untuk mengendalikan

status button. Selain itu, saya akan menambahkan fitur untuk meng-update sleep-quality record in the database, fitur menggunakan LiveData untuk meng-track button states (status button), dan menampilkan snackbar sebagai response dari event button.

Step 1: Tambahkan navigasi untuk sleep-quality tracking

Grafik navigasi sudah menyertakan jalur dari SleepTrackerFragment ke SleepQualityFragment dan jalur untuk kembali lagi. Namun, kode untuk handler click yang mengimplementasikan navigasi dari satu fragmen ke fragmen berikutnya belum dituliskan. Sekarang, kita menambahkan kode di ViewModel.

Navigasi akan bekerja seperti tampilan di tab Design navigation.xml berikut:

Langkah-langkah:

1. Buka SleepTrackerViewModel. Kita harus menambahkan navigasi sehingga ketika user menekan tombol Stop, aplikasi akan menavigasi ke SleepQualityFragment untuk mengumpulkan quality rating.

2. Di SleepTrackerViewModel, buat LiveData yang berubah saat kita ingin menavigasi ke SleepQualityFragment. Gunakan enkapsulasi untuk hanya versi LiveData yang diperoleh ke ViewModel.

```
private val _navigateToSleepQuality = MutableLiveData<SleepNight>()

val navigateToSleepQuality: LiveData<SleepNight>
    get() = _navigateToSleepQuality
```

3. Tambahkan fungsi doneNavigating() yang menyetel ulang variabel yang memicu navigasi.

```
val navigateToSleepQuality: LiveData<SleepNight>
    get() = _navigateToSleepQuality

fun doneNavigating() {
    _navigateToSleepQuality.value = null
}
```

4. Di onStopTracking(), trigger navigasi ke dalam SleepQualityFragment. Buat variabel _navigateToSleepQuality di akhir fungsi di dalam blok launch{}. Perhatikan bahwa variabel ini disetel

ke malam hari. Saat variabel ini memiliki nilai, aplikasi menavigasi ke SleepQualityFragment melewati malam.

```
_navigateToSleepQuality.value = oldNight
```

5. SleepTrackerFragment perlu mengamati _navigateToSleepQuality agar aplikasi tahu kapan harus menavigasi. Di SleepTrackerFragment, di onCreateView(), tambahkan observer untuk navigasiToSleepQuality().

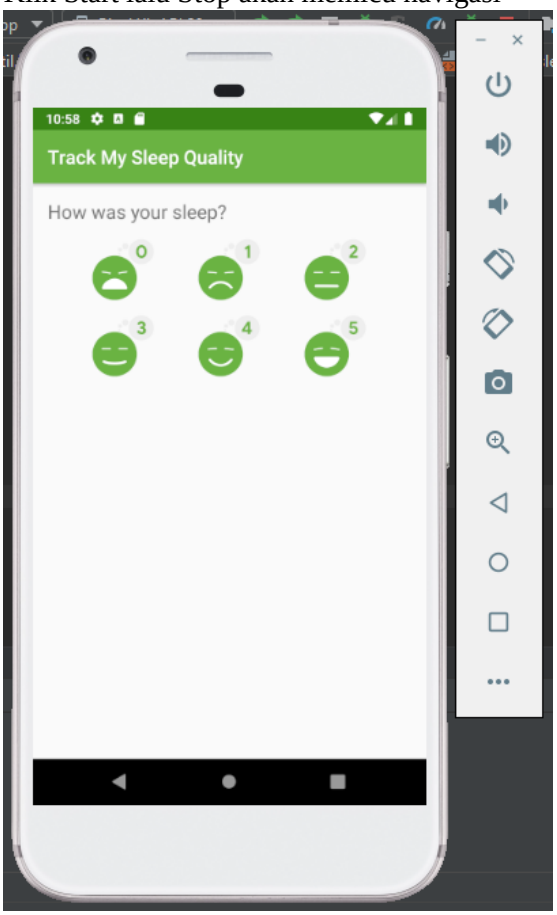
```
sleepTrackerViewModel.navigateToSleepQuality.observe( owner: this, Observer {
```

6. Di dalam observer, navigasikan dan teruskan ID malam saat ini, lalu panggil doneNavigating().

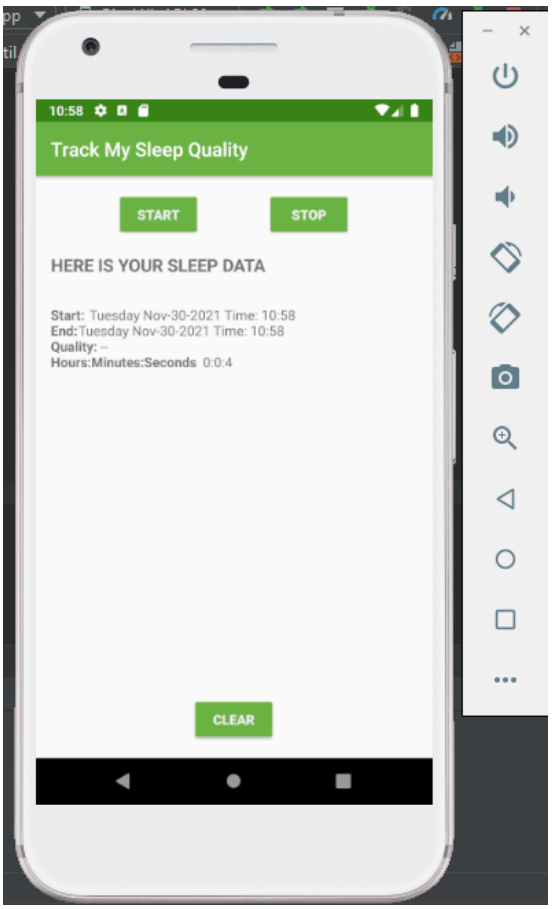
```
    night ->
    night?.let { it: SleepNight
        this.findNavController().navigate(
                SleepTrackerFragmentDirections
                        .actionSleepTrackerFragmentToSleepQualityFragment(nigh
        sleepTrackerViewModel.doneNavigating()
    }
})
```

7. Build dan jalankan aplikas. Klik start lalu klik Stop. Hal tersebut akan menavigasikan aplikasi ke layar SleepQualityFragment. Untuk kembali, gunakan tombol kembali yang ada di emulator.

Klik Start lalu Stop akan memicu navigasi



klik tombol kembali emulator untuk melihat data



Step 2: Record sleep-quality

1. Dalam package sleepquality, buat atau buka SleepQualityViewModel.kt.

2. Buat kelas SleepQualityViewModel yang menggunakan sleepNightKey dan database sebagai argumen. Kita harus memasukkan database dari factory. Kita juga harus memasukkan sleepNightKey dari navigasi.

```
class SleepQualityViewModel(
        private val sleepNightKey: Long = 0L,
        val database: SleepDatabaseDao) : ViewModel() {
```

3. Untuk menavigasi kembali ke SleepTrackerFragment menggunakan pola yang sama seperti di atas, buat _navigateToSleepTracker. Terapkan navigationToSleepTracker dan doneNavigating().

```
    private val _navigateToSleepTracker = MutableLiveData<Boolean?>()

    val navigateToSleepTracker: LiveData<Boolean?>
        get() = _navigateToSleepTracker

    fun doneNavigating() {
        _navigateToSleepTracker.value = null
    }
}
```

4. Buat one click handler, onSetSleepQuality(), untuk semua image kualitas tidur yang akan digunakan.

```
fun onSetSleepQuality(quality: Int) {
    viewModelScope.launch { this: CoroutineScope
        val tonight = database.get(sleepNightKey) ?: return@launch
        tonight.sleepQuality = quality
        database.update(tonight)

        // Setting this state variable to true will alert the observer and trigger navigation.
        _navigateToSleepTracker.value = true
    }
```

5. Dalam paket sleepquality, buka SleepQualityViewModelFactory.kt dan tambahkan kelas SleepQualityViewModelFactory. Kelas ini menggunakan versi kode boilerplate.

```
class SleepQualityViewModelFactory(
        private val sleepNightKey: Long,
        private val dataSource: SleepDatabaseDao) : ViewModelProvider.Factory {
    @Suppress( ...names: "unchecked_cast")
    override fun <T : ViewModel?> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(SleepQualityViewModel::class.java)) {
            return SleepQualityViewModel(sleepNightKey, dataSource) as T
        }
        throw IllegalArgumentException("Unknown ViewModel class")
    }
```

6. Buka SleepQualityFragment.kt.

7. Di onCreateView(), kita perlu mendapatkan argumen yang disertakan dengan navigasi. Argumen ini ada di SleepQualityFragmentArgs. Kita harus mengekstraknya dari bundle.

```
val arguments = SleepQualityFragmentArgs.fromBundle(requireArguments())
```

8. Dapatkan data source dan buat factory untuk menambahkan data source dan SleepNightKey

```
val dataSource = SleepDatabase.getInstance(application).sleepDatabaseDao

val viewModelFactory = SleepQualityViewModelFactory(arguments.sleepNightKey, dataSource)
```

9. Buat reference ViewModel

```
val sleepQualityViewModel =
        ViewModelProvider(
                owner: this, viewModelFactory).get(SleepQualityViewModel::class.java)
```

10. Tambahkan ViewModel ke binding-object. (Jika ada kesalahan dengan binding object, abaikan saja untuk saat ini.)

```
binding.sleepQualityViewModel = sleepQualityViewModel
```

11. Tambahkan observer

```
sleepQualityViewModel.navigateToSleepTracker.observe( owner: this, Observer { it: Boolean?
    if (it == true) { // Observed state is true.
        this.findNavController().navigate(
                SleepQualityFragmentDirections.actionSleepQualityFragmentToSleepTrackerFragment())
        sleepQualityViewModel.doneNavigating()
    }
})
```

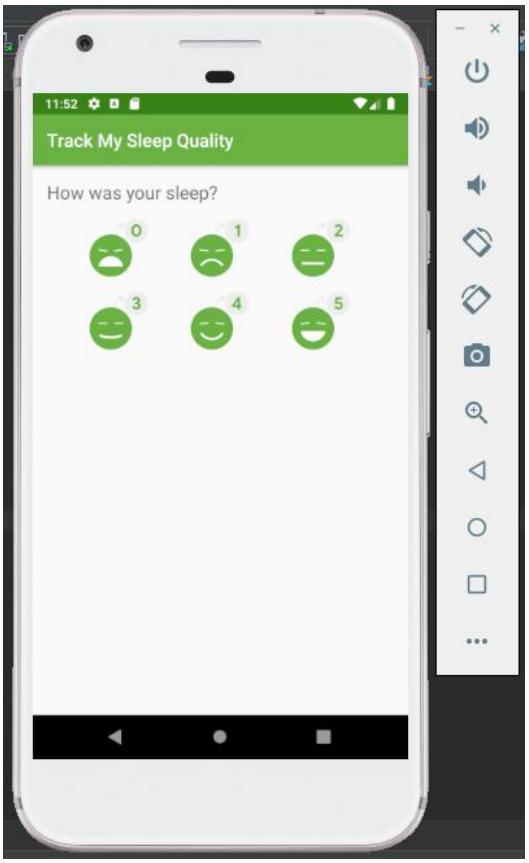
12. Buka file layout fragment_sleep_quality.xml. Di blok <data>, tambahkan variabel untuk SleepQualityViewModel.

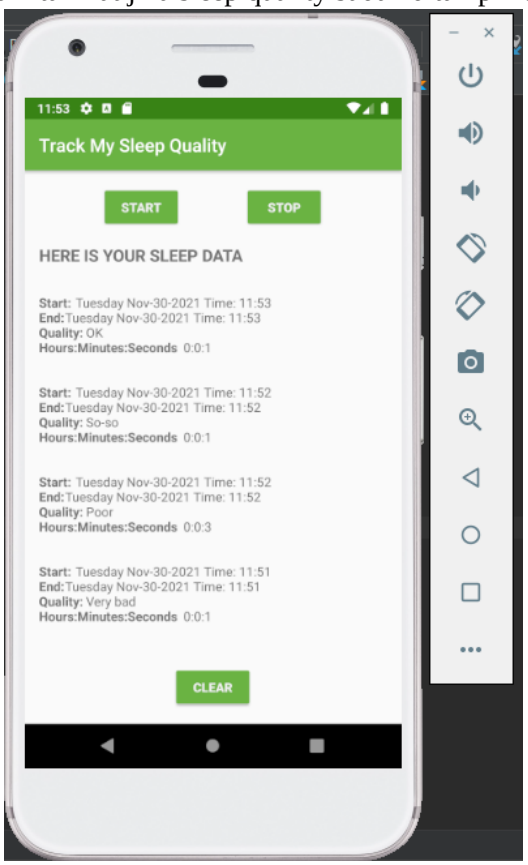
```
<data>
    <variable
        name="sleepQualityViewModel"
        type="com.example.android.trackmysleepquality.sleepquality.SleepQualityViewModel" />
</data>
```

13. Untuk masing-masing dari enam gambar sleep-quality, tambahkan click handler seperti di bawah ini. Cocokkan peringkat kualitas dengan gambar.

```
android:onClick="@{() -> sleepQualityViewModel.onSetSleepQuality(5)}"
```

14. Jalankan aplikasi, pilih Start lalu Stop, lalu pilih kualitas tidur dengan klik image dan kembali ke halaman utama untuk melihat status data. Bisa kita lihat jika sleep quality sudah ditampilkan





Step 3: Mengendalikan tampilan button

1. Buka fragment_sleep_tracker.xml
2. Tambahkan properti android:enabled ke setiap button. Properti android:enabled adalah nilai boolean yang menunjukkan apakah tombol diaktifkan atau tidak.

   Start button:
   ```
   android:enabled="@{sleepTrackerViewModel.startButtonVisible}"
   ```

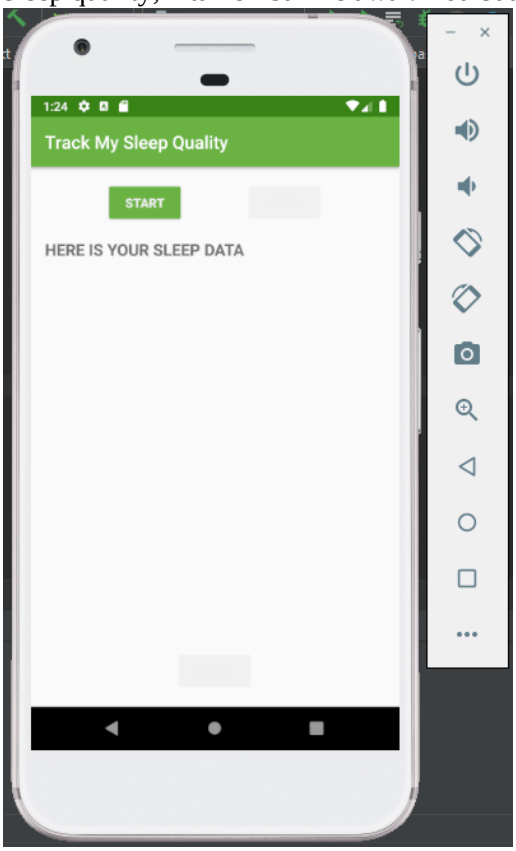
   Stop button:
   ```
   android:enabled="@{sleepTrackerViewModel.stopButtonVisible}"
   ```

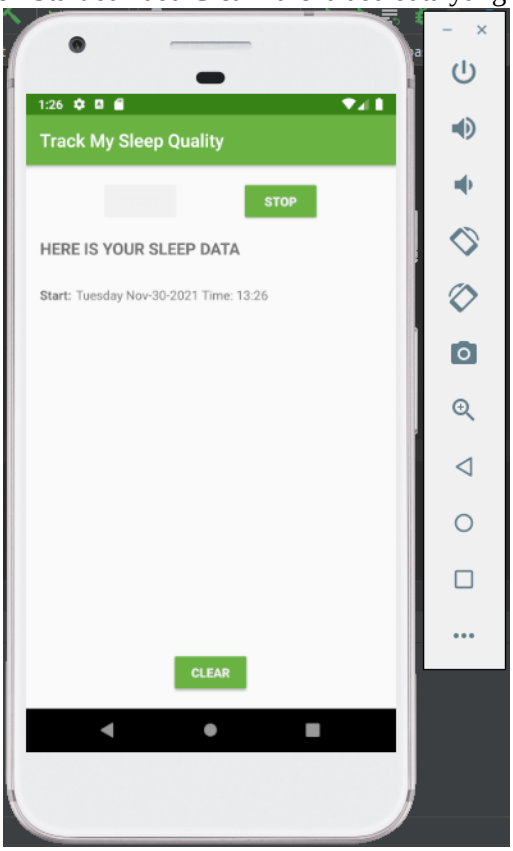
   Clear button:
   ```
   android:enabled="@{sleepTrackerViewModel.clearButtonVisible}"
   ```

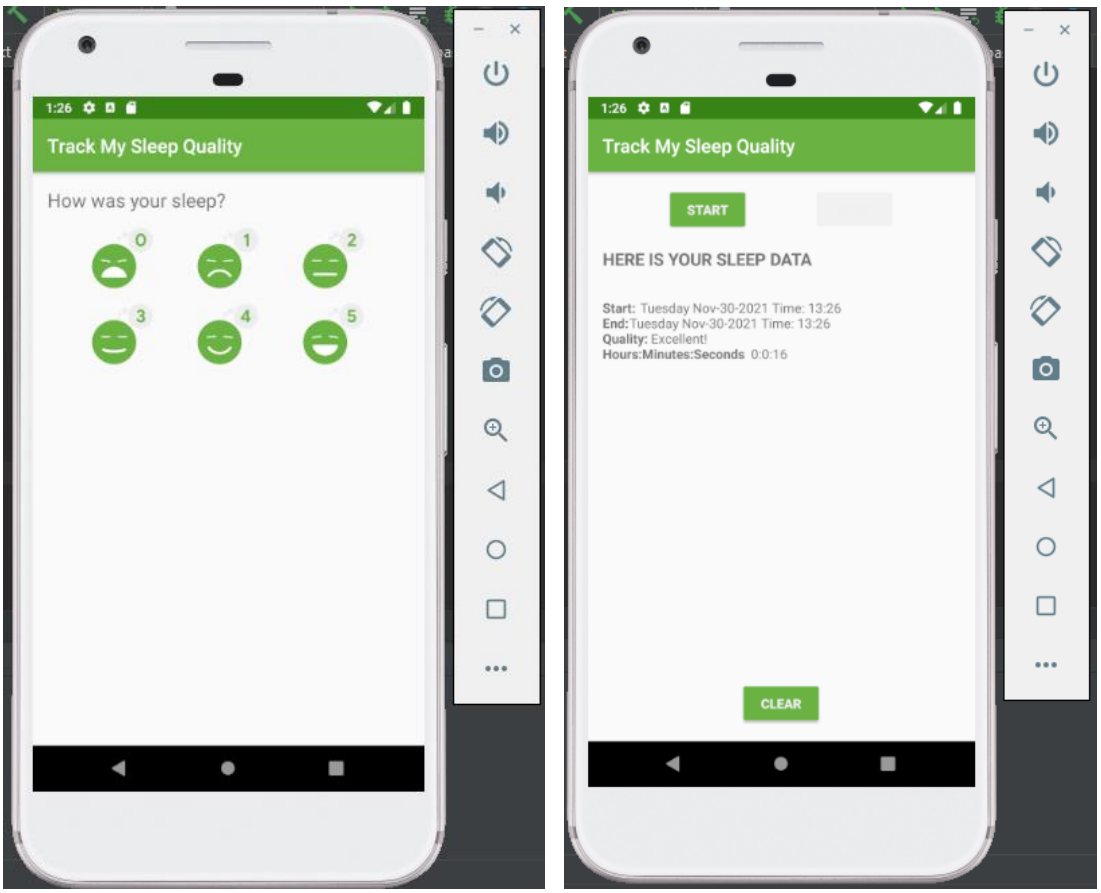
3. Buka SleepTrackerViewModel dan buat tiga variabel yang sesuai. Tetapkan setiap variabel sebuah transformasi yang untuk pengujian.
   - Start button harus enabled ketika tonight bernilai null
   - Stop button harus enabled ketika tonight tidak bernilai null
   - Clear button harus enabled ketika kondisi nights

   ```
   val startButtonVisible = Transformations.map(tonight) { it: SleepNight?
       it == null
   }
   val stopButtonVisible = Transformations.map(tonight) { it: SleepNight?
       it != null
   }
   val clearButtonVisible = Transformations.map(nights) { it: List<SleepNight>!
       it?.isNotEmpty()
   ```

4. Jalankan aplikasi. Di awal hanya ada button start yang muncul. Lalu setelah klik button Start, button Start hilang dan muncul Stop & Clear. Setelah klik Stop, kita memilih sleep quality. Setelah memilih sleep quality, kita kembali ke awal. Ada button Start dan ada Clear karena ada data yang bisa di-clear.







Step 4: Menambahkan snackbar

Setelah pengguna mengosongkan database, aplikasi bisa menunjukkan konfirmasi kepada pengguna menggunakan widget Snackbar. Snackbar memberikan umpan balik singkat tentang operasi melalui pesan di bagian bawah layar. Snackbar menghilang setelah batas waktu yang ditentukan, setelah interaksi pengguna di tempat lain di layar, atau setelah pengguna menggeser snackbar dari layar.

Langkah-langkah:

1. Di SleepTrackerViewModel, buat encapsulated event

```
private var _showSnackbarEvent = MutableLiveData<Boolean>()

val showSnackBarEvent: LiveData<Boolean>
    get() = _showSnackbarEvent
```

2. Lalu implementasikan doneShowingSnackbar()

```
fun doneShowingSnackbar() {
    _showSnackbarEvent.value = false
}
```

3. Di the SleepTrackerFragment, di dalam onCreateView(), tambahkan observer yang

```
sleepTrackerViewModel.showSnackBarEvent.observe( owner: this, Observer { })
```

4. Di dalam block observer, tampilkan snackbar dan reset event

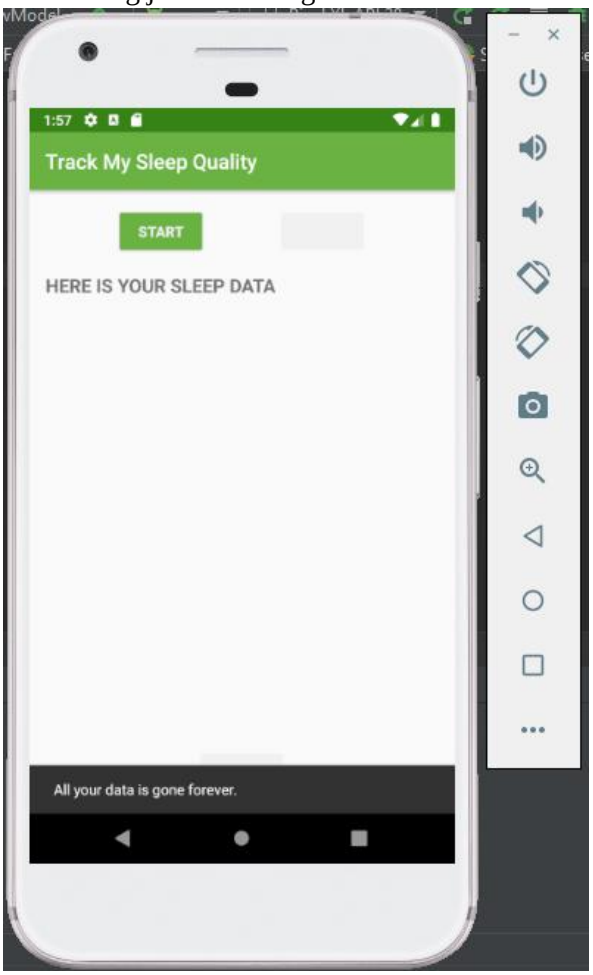
```
if (it == true) { // Observed state is true.
    Snackbar.make(
            requireActivity().findViewById(android.R.id.content),
            getString(R.string.cleared_message),
            Snackbar.LENGTH_SHORT // How long to display the message.
```

```
    ).show()
    sleepTrackerViewModel.doneShowingSnackbar()
}
```

5. Di SleepTrackerViewModel, trigger event di dalam onClear() method. Untuk melakukan ini, buat event value menjadi true di dalam block launch:

```
_showSnackbarEvent.value = true
```

6. Jalankan aplikasi, klik button Clear. Perhatikan ada snackbar di bagian bawah layar. Snackbar tersebut akan hilang jika kita meng-klik button lain atau keluar dari aplikasi.



KESIMPULAN

Di pertemuan ke-10 kali ini, saya berhasil memenuhi tujuan dibuatnya modul ini yaitu dapat membuat aplikasi yang menerapkan couroutines dan room. Aplikasi yang bibuat di pertemuan ini adalah aplikasi untuk mendeteksi atau tracking kualitas tidur pengguna. Alur pengerjaan aplikasi adalah pengguna bisa mengklik button Start saat sebelum tidur, lalu klik button Stop setelah klik selesai tidur sehingga aplikasi bisa memberikan segala info tentang tidur kepada pengguna. Di bagian akhir atau bagian tugas, saya mengembangkan aplikasi sehingga layar bisa menavigasi ke dari satu layar ke layar lainnya. User juga bisa mengklik image supaya aplikasi bisa memberikan info tentang kualitas tidur kepada user. Di pertemuan ini saya belajar banyak hal baru tentang coroutines dan room.

Terima Kasih