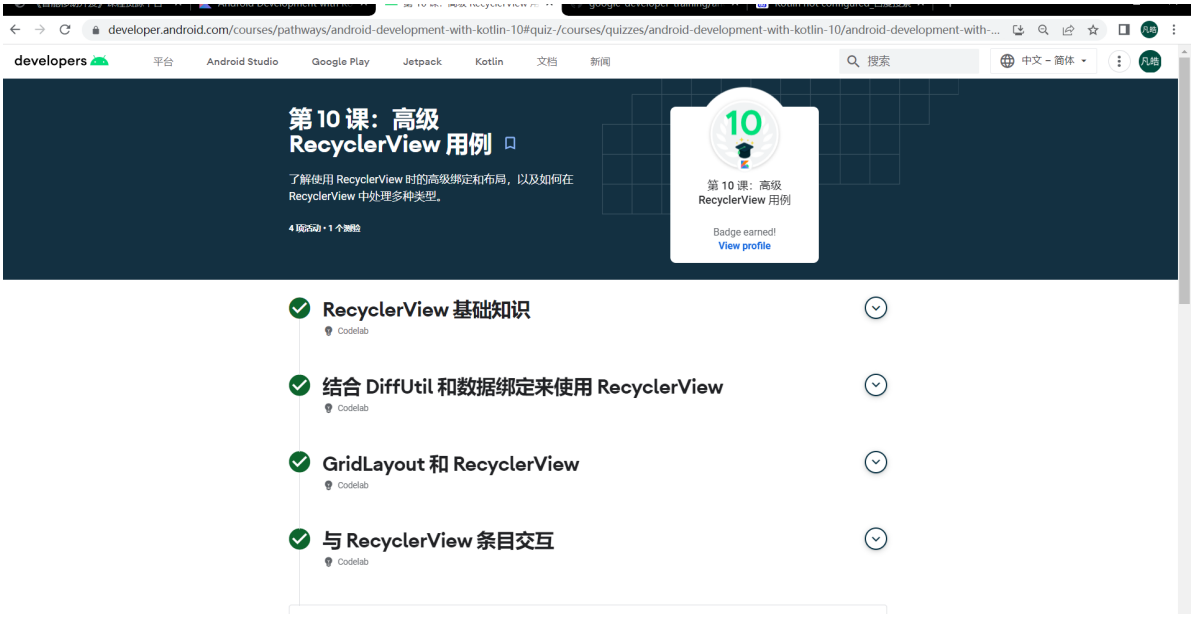
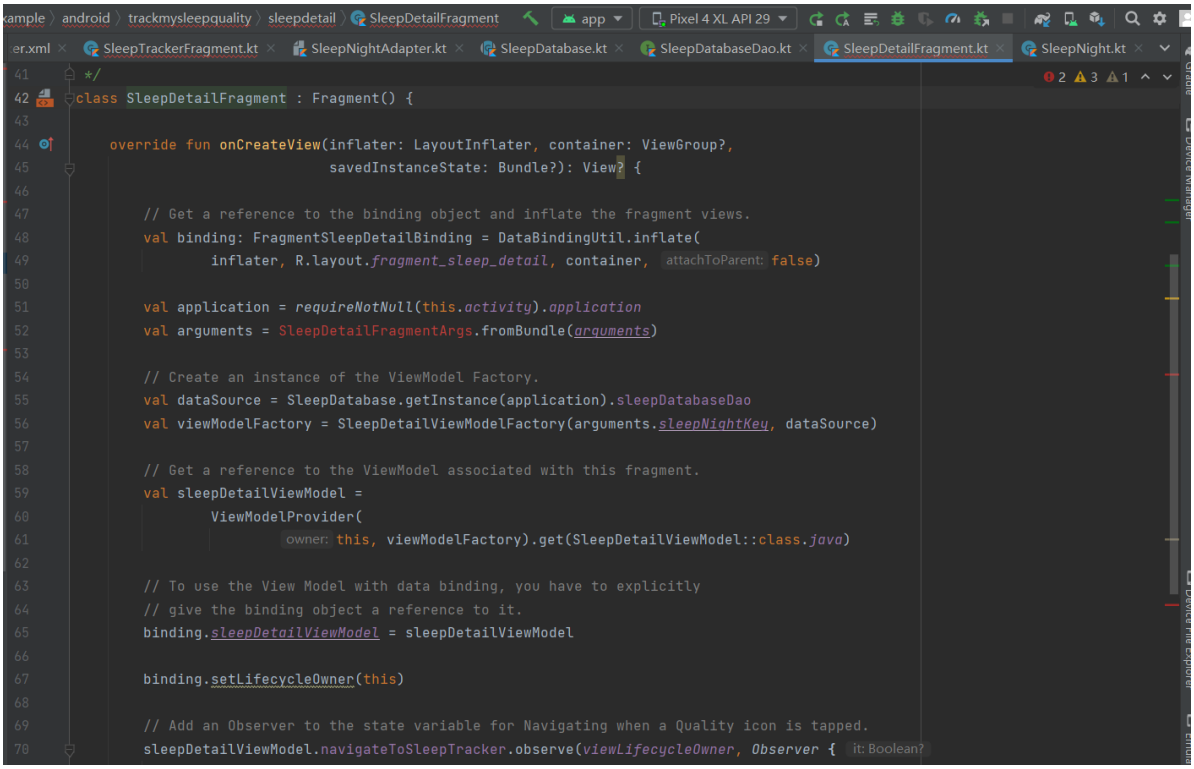
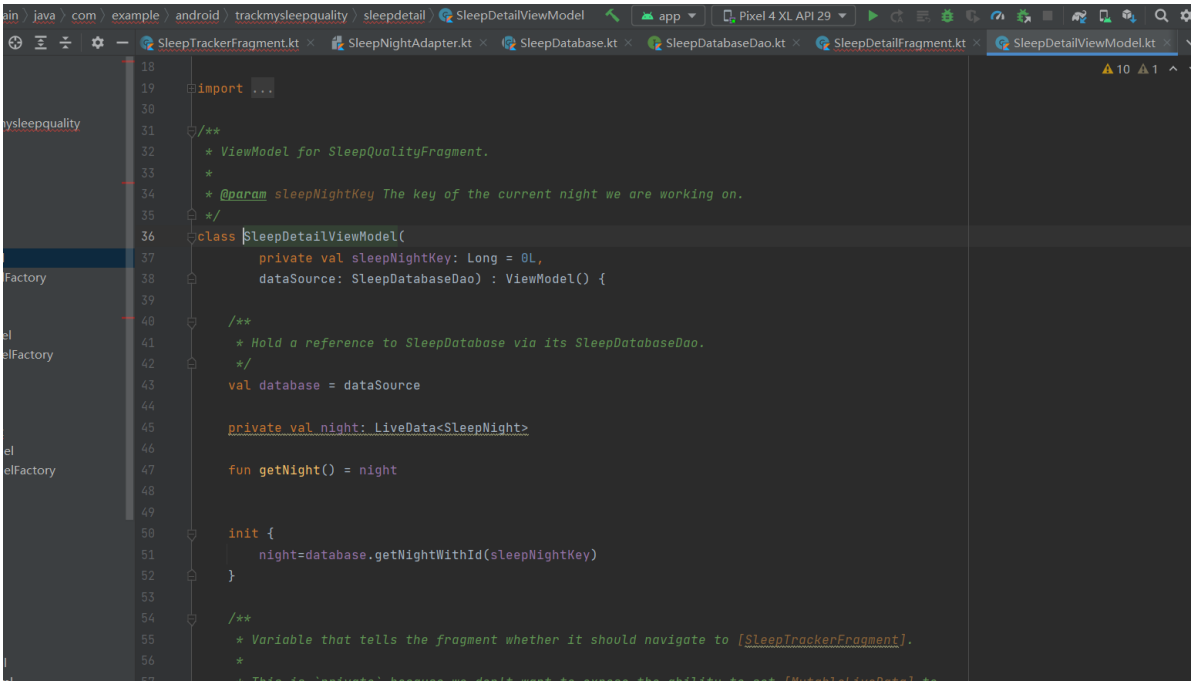
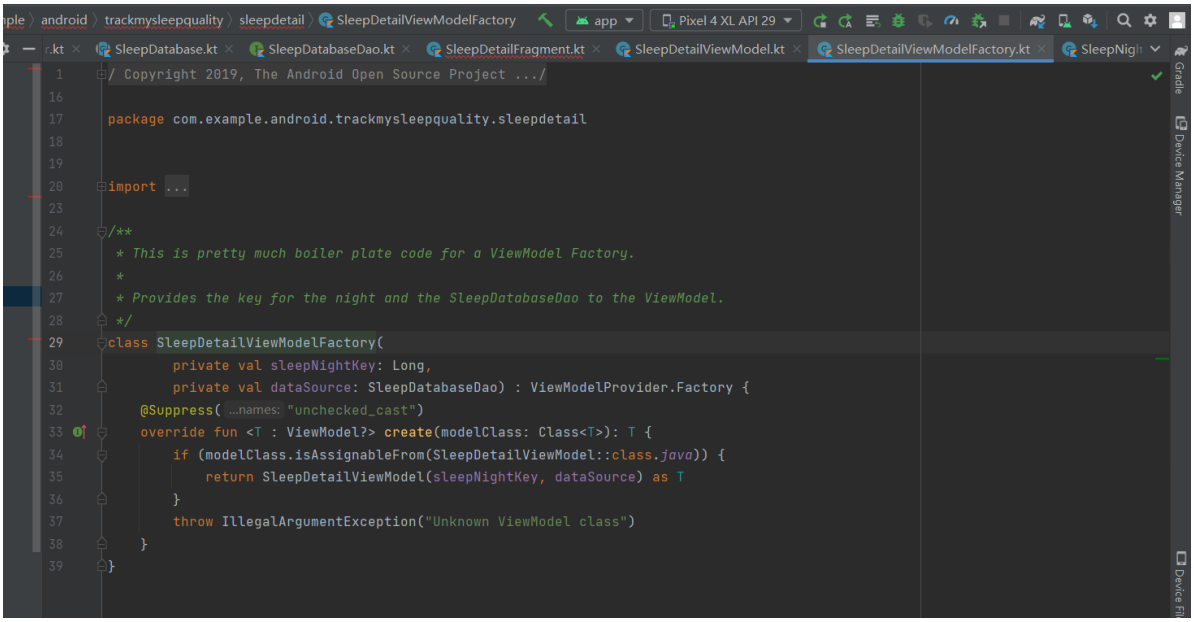
注：标红的地方不会影响程序，只是我没有清除缓存重新开始



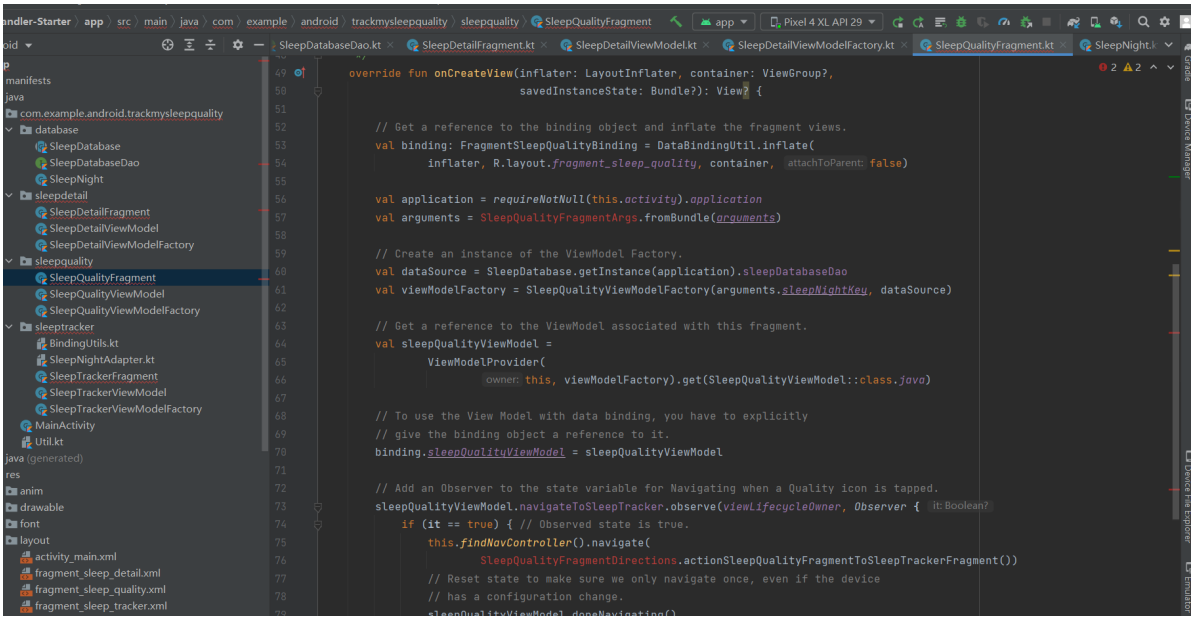




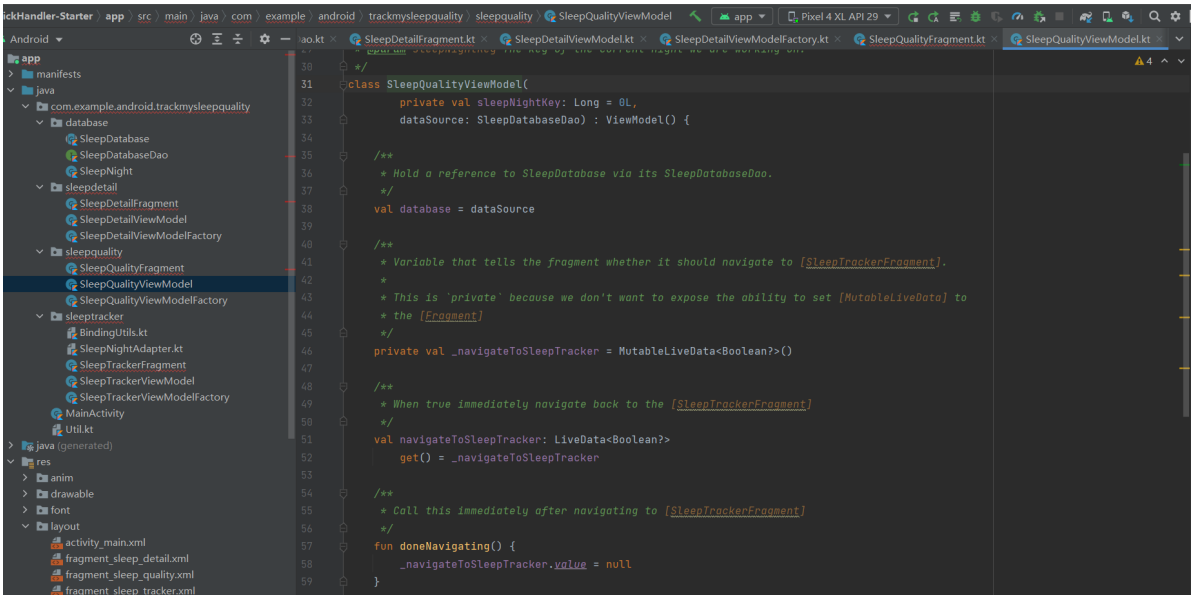
```kotlin
import ...

/**
 * ViewModel for SleepQualityFragment.
 *
 * @param sleepNightKey The key of the current night we are working on.
 */
class SleepDetailViewModel(
        private val sleepNightKey: Long = 0L,
        dataSource: SleepDatabaseDao) : ViewModel() {

    /**
     * Hold a reference to SleepDatabase via its SleepDatabaseDao.
     */
    val database = dataSource

    private val night: LiveData<SleepNight>

    fun getNight() = night


    init {
        night=database.getNightWithId(sleepNightKey)
    }

    /**
     * Variable that tells the fragment whether it should navigate to [SleepTrackerFragment].
     *
```



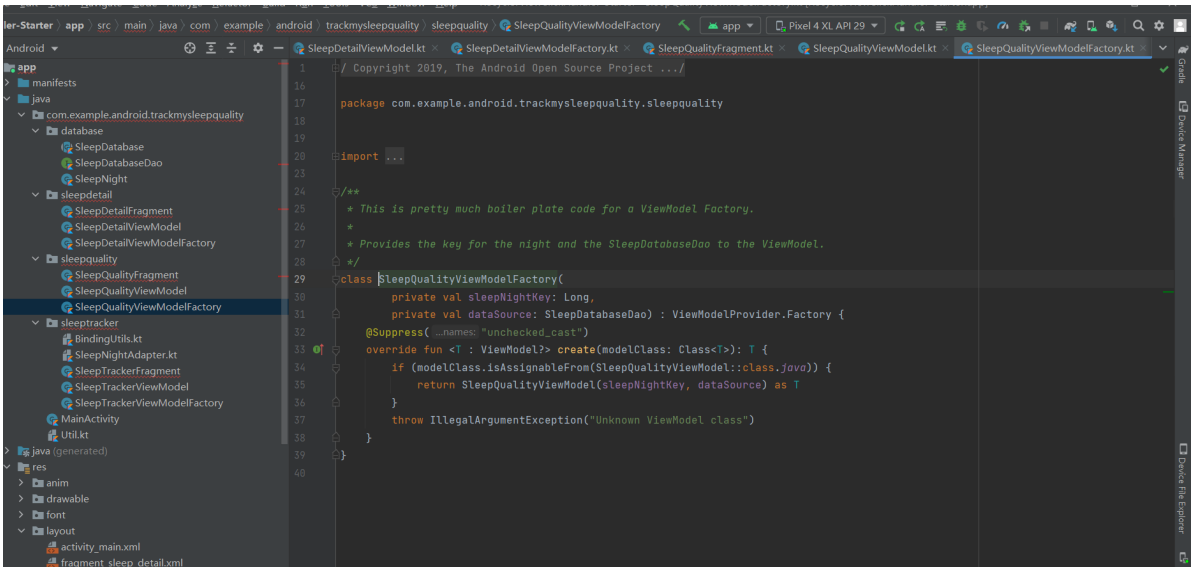
```kotlin
/ Copyright 2019, The Android Open Source Project .../

package com.example.android.trackmysleepquality.sleepdetail


import ...

/**
 * This is pretty much boiler plate code for a ViewModel Factory.
 *
 * Provides the key for the night and the SleepDatabaseDao to the ViewModel.
 */
class SleepDetailViewModelFactory(
        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(SleepDetailViewModel::class.java)) {
            return SleepDetailViewModel(sleepNightKey, dataSource) as T
        }
        throw IllegalArgumentException("Unknown ViewModel class")
    }
}
```



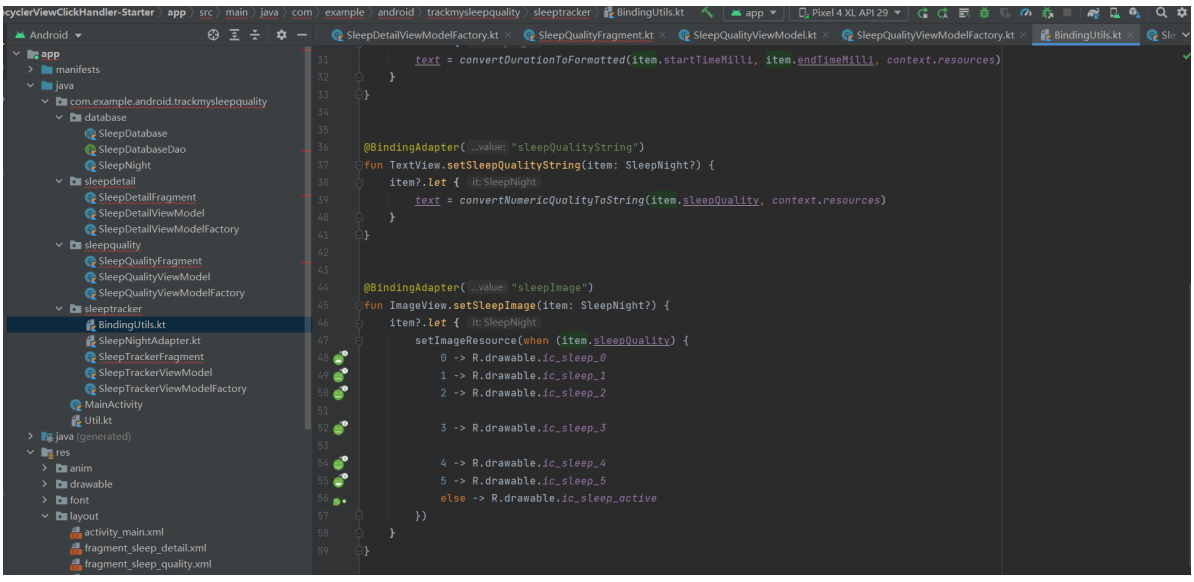
```kotlin
    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                              savedInstanceState: Bundle?): View? {

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

        val application = requireNotNull(this.activity).application
        val arguments = SleepQualityFragmentArgs.fromBundle(arguments)

        // Create an instance of the ViewModel Factory.
        val dataSource = SleepDatabase.getInstance(application).sleepDatabaseDao
        val viewModelFactory = SleepQualityViewModelFactory(arguments.sleepNightKey, dataSource)

        // Get a reference to the ViewModel associated with this fragment.
        val sleepQualityViewModel =
                ViewModelProvider(
                        owner: this, viewModelFactory).get(SleepQualityViewModel::class.java)

        // To use the View Model with data binding, you have to explicitly
        // give the binding object a reference to it.
        binding.sleepQualityViewModel = sleepQualityViewModel

        // Add an Observer to the state variable for Navigating when a Quality icon is tapped.
        sleepQualityViewModel.navigateToSleepTracker.observe(viewLifecycleOwner, Observer {  it: Boolean?
            if (it == true) { // Observed state is true.
                this.findNavController().navigate(
                        SleepQualityFragmentDirections.actionSleepQualityFragmentToSleepTrackerFragment())
                // Reset state to make sure we only navigate once, even if the device
                // has a configuration change.
```



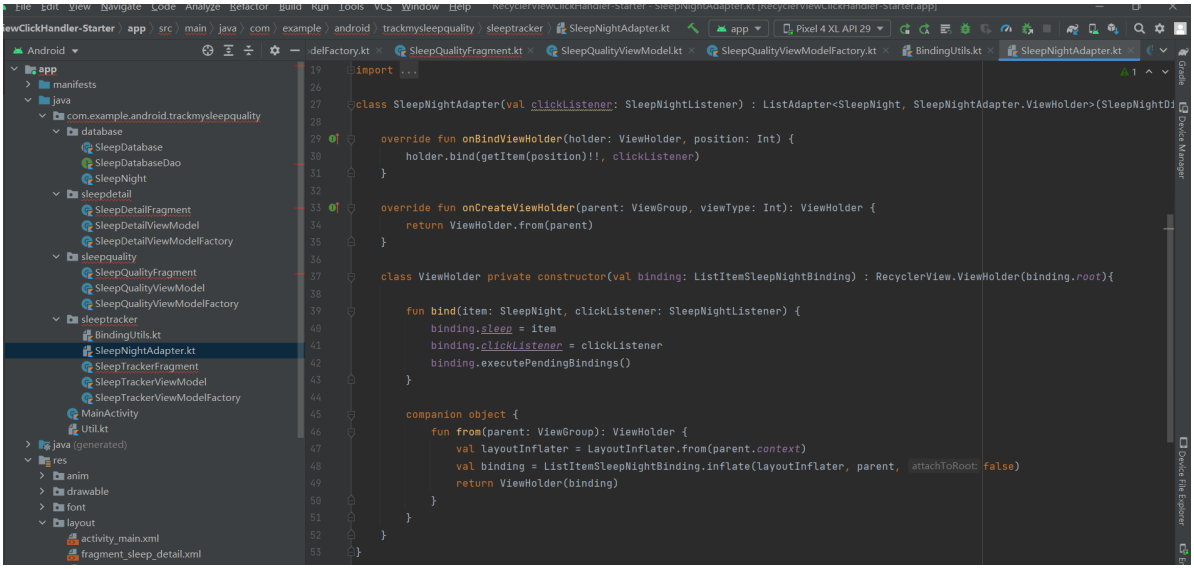
```
/**
 */
class SleepQualityViewModel(
        private val sleepNightKey: Long = 0L,
        dataSource: SleepDatabaseDao) : ViewModel() {

    /**
     * Hold a reference to SleepDatabase via its SleepDatabaseDao.
     */
    val database = dataSource

    /**
     * Variable that tells the fragment whether it should navigate to [SleepTrackerFragment].
     *
     * This is `private` because we don't want to expose the ability to set [MutableLiveData] to
     * the [Fragment]
     */
    private val _navigateToSleepTracker = MutableLiveData<Boolean?>()

    /**
     * When true immediately navigate back to the [SleepTrackerFragment]
     */
    val navigateToSleepTracker: LiveData<Boolean?>
        get() = _navigateToSleepTracker

    /**
     * Call this immediately after navigating to [SleepTrackerFragment]
     */
    fun doneNavigating() {
        _navigateToSleepTracker.value = null
    }
```



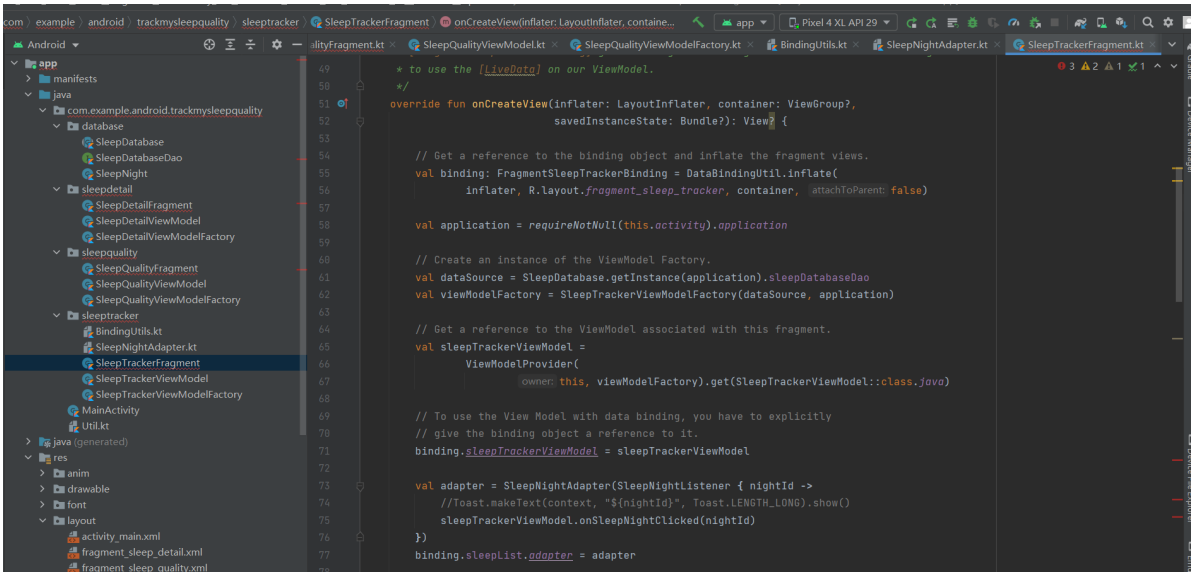
```
/ Copyright 2019, The Android Open Source Project .../

package com.example.android.trackmysleepquality.sleepquality


import ...

/**
 * This is pretty much boiler plate code for a ViewModel Factory.
 *
 * Provides the key for the night and the SleepDatabaseDao to the ViewModel.
 */
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")
    }
}
```



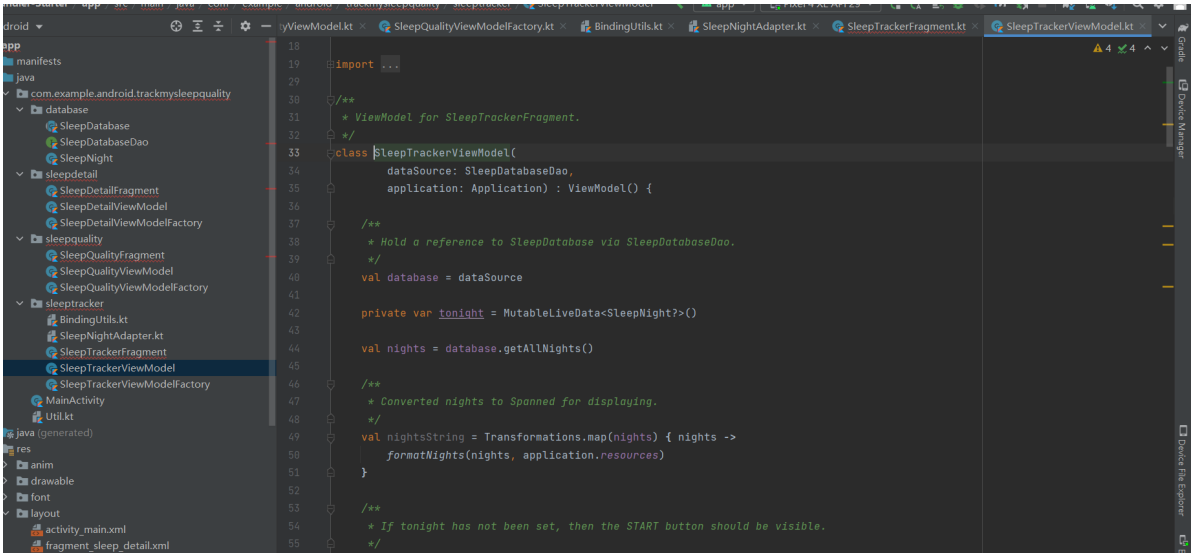
```
            text = convertDurationToFormatted(item.startTimeMilli, item.endTimeMilli, context.resources)
        }
    }


    @BindingAdapter( ...value: "sleepQualityString")
    fun TextView.setSleepQualityString(item: SleepNight?) {
        item?.let { it: SleepNight
            text = convertNumericQualityToString(item.sleepQuality, context.resources)
        }
    }


    @BindingAdapter( ...value: "sleepImage")
    fun ImageView.setSleepImage(item: SleepNight?) {
        item?.let { it: SleepNight
            setImageResource(when (item.sleepQuality) {
                0 -> R.drawable.ic_sleep_0
                1 -> R.drawable.ic_sleep_1
                2 -> R.drawable.ic_sleep_2

                3 -> R.drawable.ic_sleep_3

                4 -> R.drawable.ic_sleep_4
                5 -> R.drawable.ic_sleep_5
                else -> R.drawable.ic_sleep_active
            })
        }
    }
```



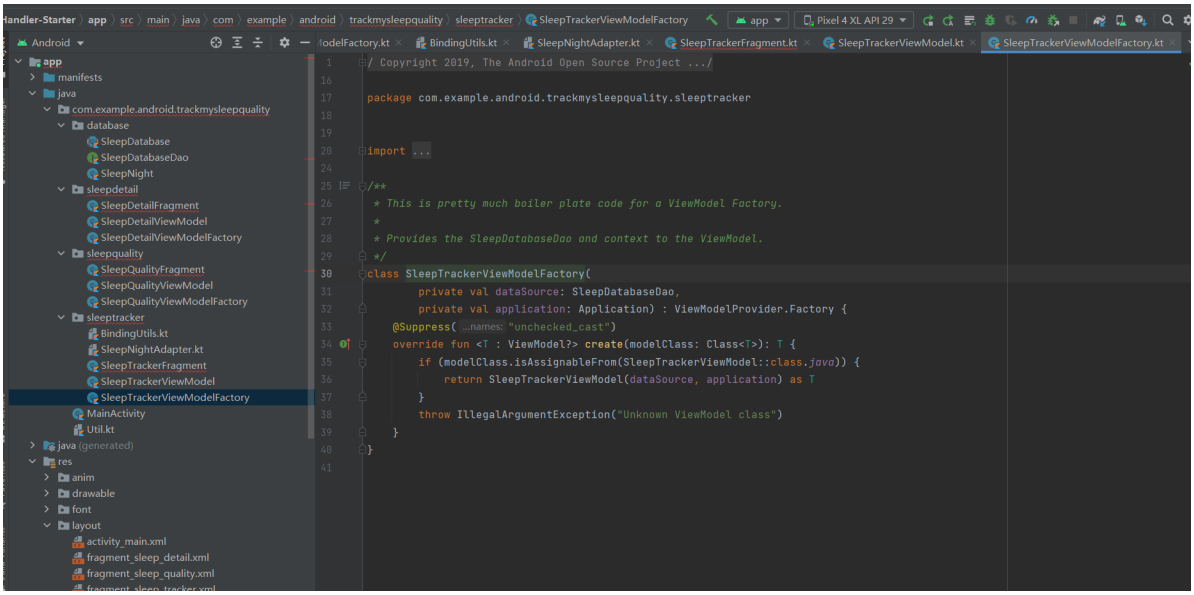
```
import ...

class SleepNightAdapter(val clickListener: SleepNightListener) : ListAdapter<SleepNight, SleepNightAdapter.ViewHolder>(SleepNightDi

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        holder.bind(getItem(position)!!, clickListener)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        return ViewHolder.from(parent)
    }

    class ViewHolder private constructor(val binding: ListItemSleepNightBinding) : RecyclerView.ViewHolder(binding.root){

        fun bind(item: SleepNight, clickListener: SleepNightListener) {
            binding.sleep = item
            binding.clickListener = clickListener
            binding.executePendingBindings()
        }

        companion object {
            fun from(parent: ViewGroup): ViewHolder {
                val layoutInflater = LayoutInflater.from(parent.context)
                val binding = ListItemSleepNightBinding.inflate(layoutInflater, parent,  attachToRoot: false)
                return ViewHolder(binding)
            }
        }
    }
}
```



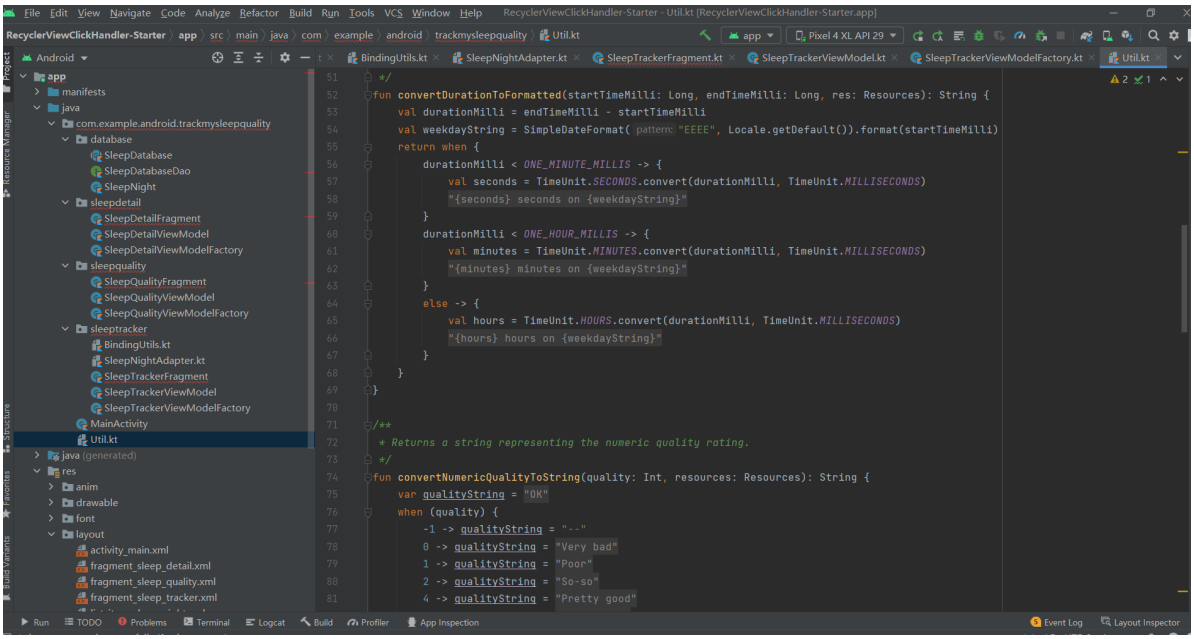
```
     * to use the [LiveData] on our ViewModel.
     */
    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                              savedInstanceState: Bundle?): View? {

        // 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

        // Create an instance of the ViewModel Factory.
        val dataSource = SleepDatabase.getInstance(application).sleepDatabaseDao
        val viewModelFactory = SleepTrackerViewModelFactory(dataSource, application)

        // Get a reference to the ViewModel associated with this fragment.
        val sleepTrackerViewModel =
                ViewModelProvider(
                        owner: this, viewModelFactory).get(SleepTrackerViewModel::class.java)

        // To use the View Model with data binding, you have to explicitly
        // give the binding object a reference to it.
        binding.sleepTrackerViewModel = sleepTrackerViewModel

        val adapter = SleepNightAdapter(SleepNightListener { nightId ->
            //Toast.makeText(context, "${nightId}", Toast.LENGTH_LONG).show()
            sleepTrackerViewModel.onSleepNightClicked(nightId)
        })
        binding.sleepList.adapter = adapter
```



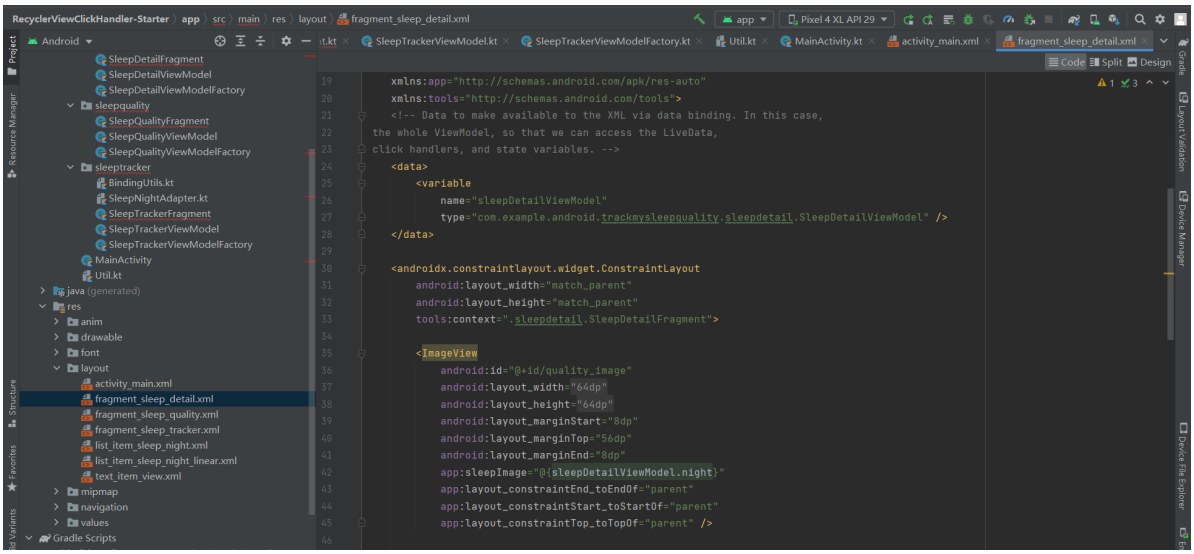
```
import ...

/**
 * ViewModel for SleepTrackerFragment.
 */
class SleepTrackerViewModel(
        dataSource: SleepDatabaseDao,
        application: Application) : ViewModel() {

    /**
     * Hold a reference to SleepDatabase via SleepDatabaseDao.
     */
    val database = dataSource

    private var tonight = MutableLiveData<SleepNight?>()

    val nights = database.getAllNights()

    /**
     * Converted nights to Spanned for displaying.
     */
    val nightsString = Transformations.map(nights) { nights ->
        formatNights(nights, application.resources)
    }

    /**
     * If tonight has not been set, then the START button should be visible.
     */
```



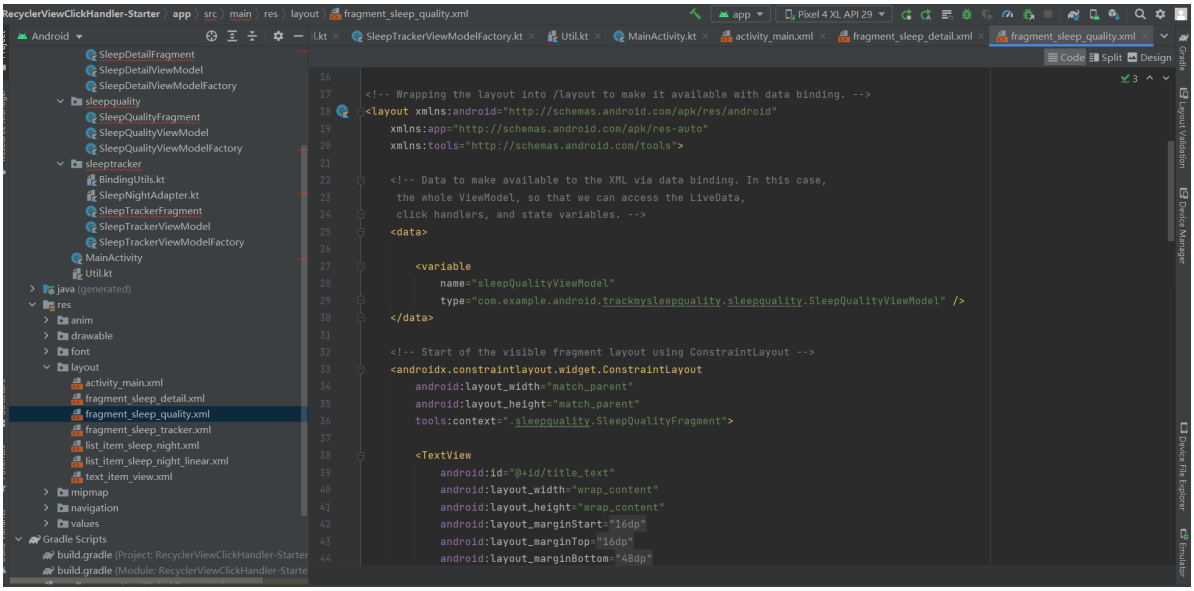
```
/ Copyright 2019, The Android Open Source Project .../

package com.example.android.trackmysleepquality.sleeptracker


import ...

/**
 * This is pretty much boiler plate code for a ViewModel Factory.
 *
 * Provides the SleepDatabaseDao and context to the ViewModel.
 */
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")
    }
}
```



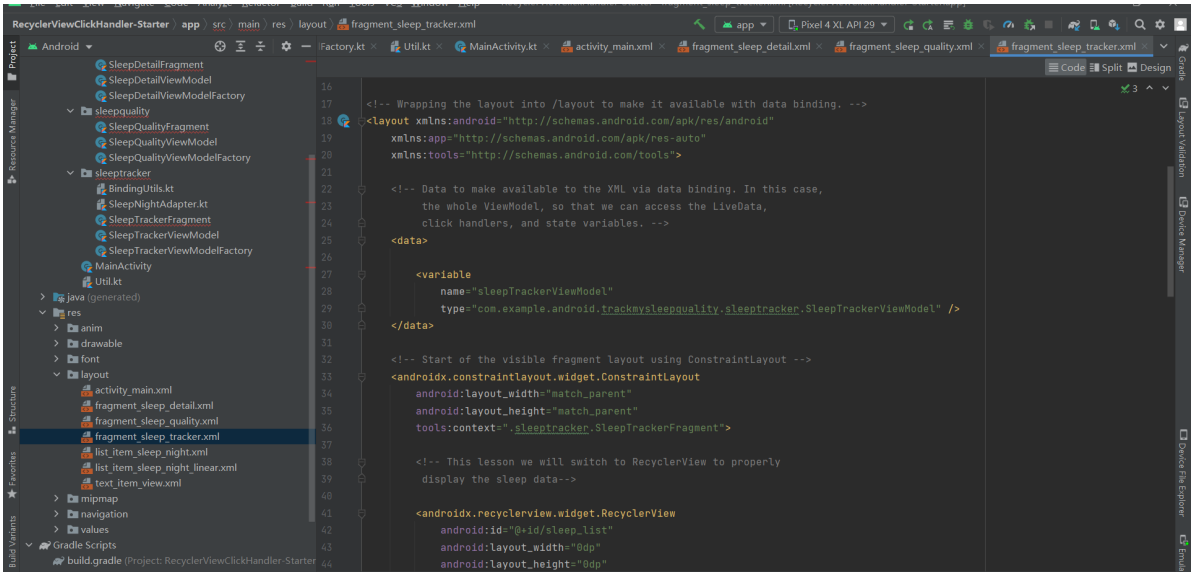
```
*/
fun convertDurationToFormatted(startTimeMilli: Long, endTimeMilli: Long, res: Resources): String {
    val durationMilli = endTimeMilli - startTimeMilli
    val weekdayString = SimpleDateFormat( pattern: "EEEE", Locale.getDefault()).format(startTimeMilli)
    return when {
        durationMilli < ONE_MINUTE_MILLIS -> {
            val seconds = TimeUnit.SECONDS.convert(durationMilli, TimeUnit.MILLISECONDS)
            "{seconds} seconds on {weekdayString}"
        }
        durationMilli < ONE_HOUR_MILLIS -> {
            val minutes = TimeUnit.MINUTES.convert(durationMilli, TimeUnit.MILLISECONDS)
            "{minutes} minutes on {weekdayString}"
        }
        else -> {
            val hours = TimeUnit.HOURS.convert(durationMilli, TimeUnit.MILLISECONDS)
            "{hours} hours on {weekdayString}"
        }
    }
}

/**
 * Returns a string representing the numeric quality rating.
 */
fun convertNumericQualityToString(quality: Int, resources: Resources): String {
    var qualityString = "OK"
    when (quality) {
        -1 -> qualityString = "--"
        0 -> qualityString = "Very bad"
        1 -> qualityString = "Poor"
        2 -> qualityString = "So-so"
        4 -> qualityString = "Pretty good"
```



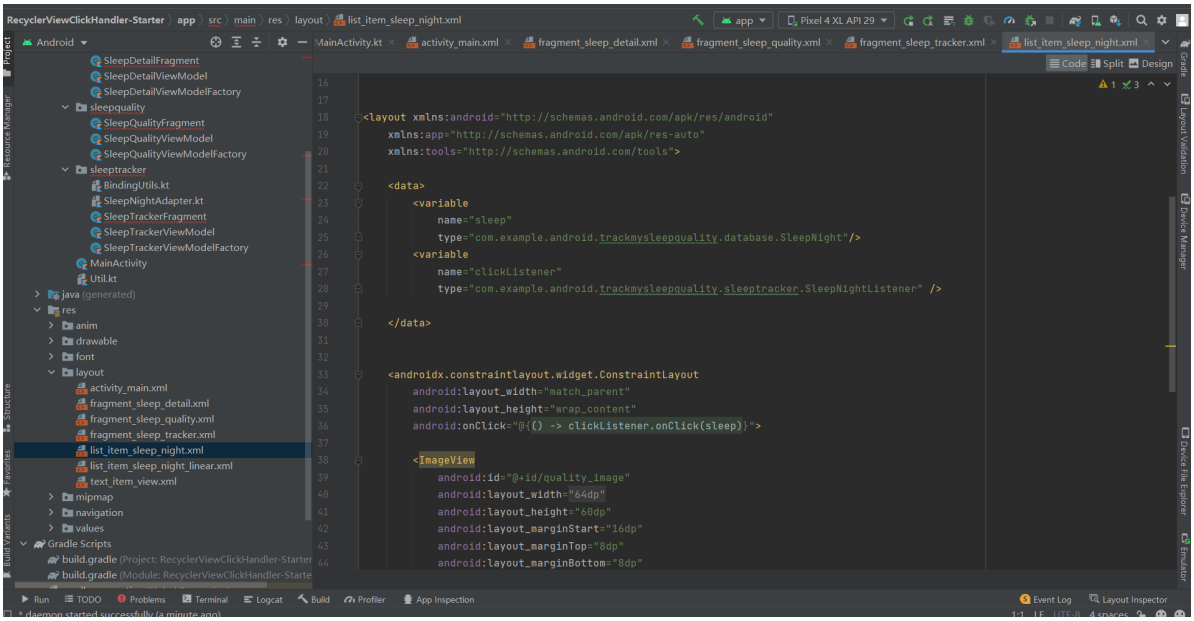
```
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">
    <!-- Data to make available to the XML via data binding. In this case,
the whole ViewModel, so that we can access the LiveData,
click handlers, and state variables. -->
    <data>
        <variable
            name="sleepDetailViewModel"
            type="com.example.android.trackmysleepquality.sleepdetail.SleepDetailViewModel" />
    </data>

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".sleepdetail.SleepDetailFragment">

        <ImageView
            android:id="@+id/quality_image"
            android:layout_width="64dp"
            android:layout_height="64dp"
            android:layout_marginStart="8dp"
            android:layout_marginTop="56dp"
            android:layout_marginEnd="8dp"
            app:sleepImage="@{sleepDetailViewModel.night}"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />
```



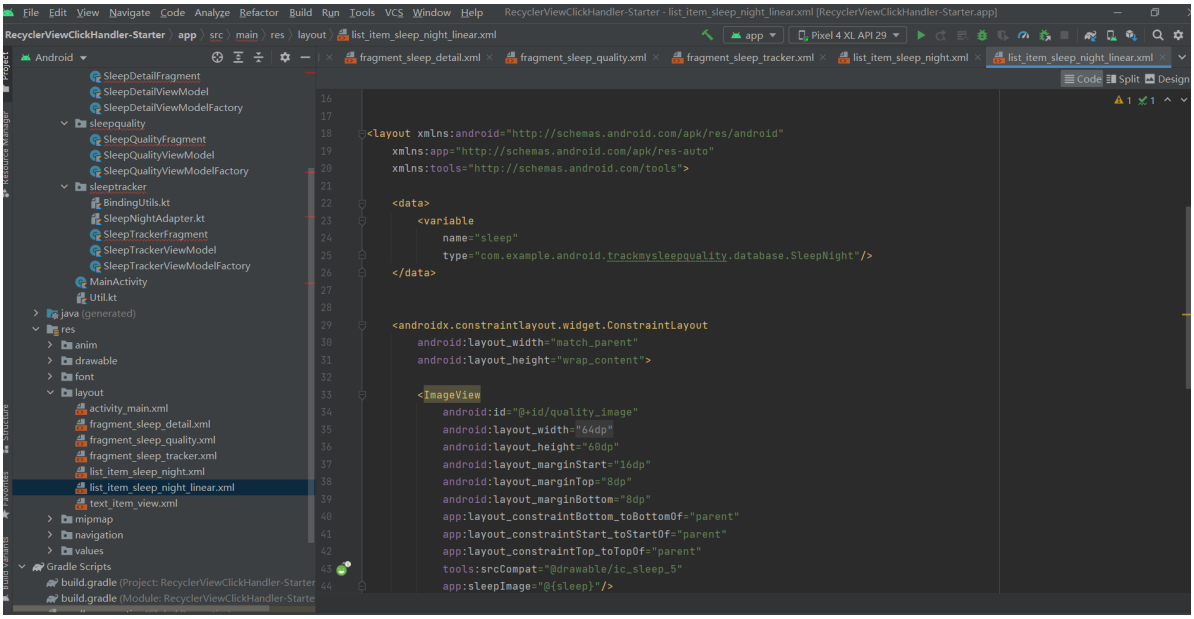
```
<!-- Wrapping the layout into /layout to make it available with data binding. -->
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

    <!-- Data to make available to the XML via data binding. In this case,
     the whole ViewModel, so that we can access the LiveData,
     click handlers, and state variables. -->
    <data>

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

    <!-- Start of the visible fragment layout using ConstraintLayout -->
    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".sleepquality.SleepQualityFragment">

        <TextView
            android:id="@+id/title_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="16dp"
            android:layout_marginTop="16dp"
            android:layout_marginBottom="48dp"
```



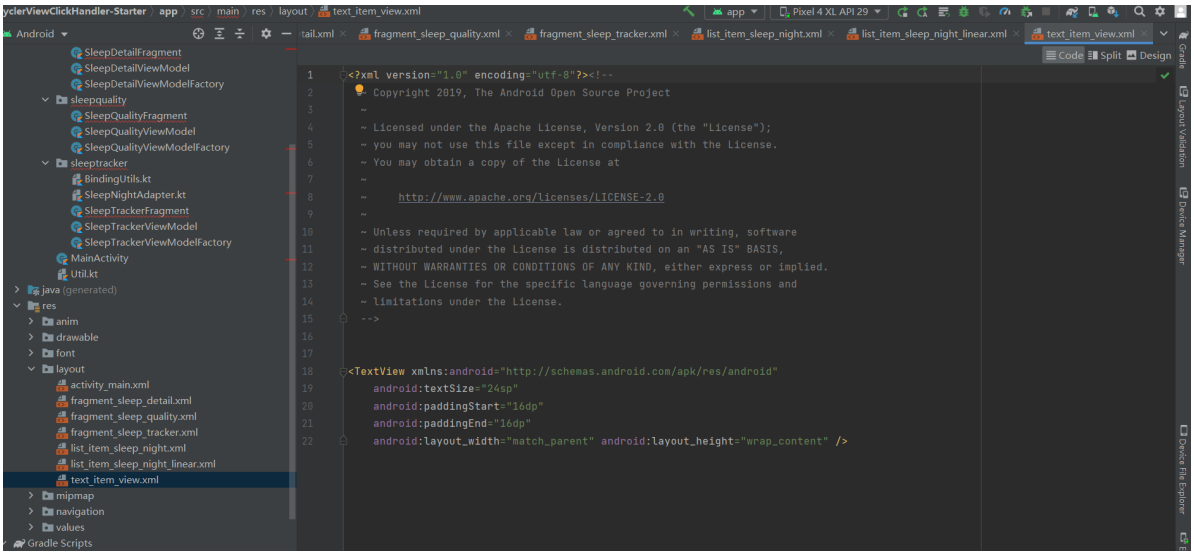
```
<!-- Wrapping the layout into /layout to make it available with data binding. -->
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

    <!-- Data to make available to the XML via data binding. In this case,
         the whole ViewModel, so that we can access the LiveData,
         click handlers, and state variables. -->
    <data>

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

    <!-- Start of the visible fragment layout using ConstraintLayout -->
    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".sleeptracker.SleepTrackerFragment">

        <!-- This lesson we will switch to RecyclerView to properly
         display the sleep data-->

        <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/sleep_list"
            android:layout_width="0dp"
            android:layout_height="0dp"
```



```


<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

    <data>
        <variable
            name="sleep"
            type="com.example.android.trackmysleepquality.database.SleepNight"/>
        <variable
            name="clickListener"
            type="com.example.android.trackmysleepquality.sleeptracker.SleepNightListener" />

    </data>


    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="@{() -> clickListener.onClick(sleep)}">

        <ImageView
            android:id="@+id/quality_image"
            android:layout_width="64dp"
            android:layout_height="60dp"
            android:layout_marginStart="16dp"
            android:layout_marginTop="8dp"
            android:layout_marginBottom="8dp"
```



```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

    <data>
        <variable
            name="sleep"
            type="com.example.android.trackmysleepquality.database.SleepNight"/>
    </data>


    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <ImageView
            android:id="@+id/quality_image"
            android:layout_width="64dp"
            android:layout_height="60dp"
            android:layout_marginStart="16dp"
            android:layout_marginTop="8dp"
            android:layout_marginBottom="8dp"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            tools:srcCompat="@drawable/ic_sleep_5"
            app:sleepImage="@{sleep}"/>
```



```
<?xml version="1.0" encoding="utf-8"?><!--
  ~ Copyright 2019, The Android Open Source Project
  ~
  ~ Licensed under the Apache License, Version 2.0 (the "License");
  ~ you may not use this file except in compliance with the License.
  ~ You may obtain a copy of the License at
  ~
  ~     http://www.apache.org/licenses/LICENSE-2.0
  ~
  ~ Unless required by applicable law or agreed to in writing, software
  ~ distributed under the License is distributed on an "AS IS" BASIS,
  ~ WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  ~ See the License for the specific language governing permissions and
  ~ limitations under the License.
  -->


<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:textSize="24sp"
    android:paddingStart="16dp"
    android:paddingEnd="16dp"
    android:layout_width="match_parent" android:layout_height="wrap_content" />
```