MODUL PRAKTIKUM PEMROGRAMAN UNTUK PERANGKAT BERGERAK 1

Indra Azimi, S.T., M.T.

Reza Budiawan, S.T., M.T., OCA

Cahyana S.T., M.Kom.

D3 Rekayasa Perangkat Lunak Aplikasi
Telkom University

Daftar Isi

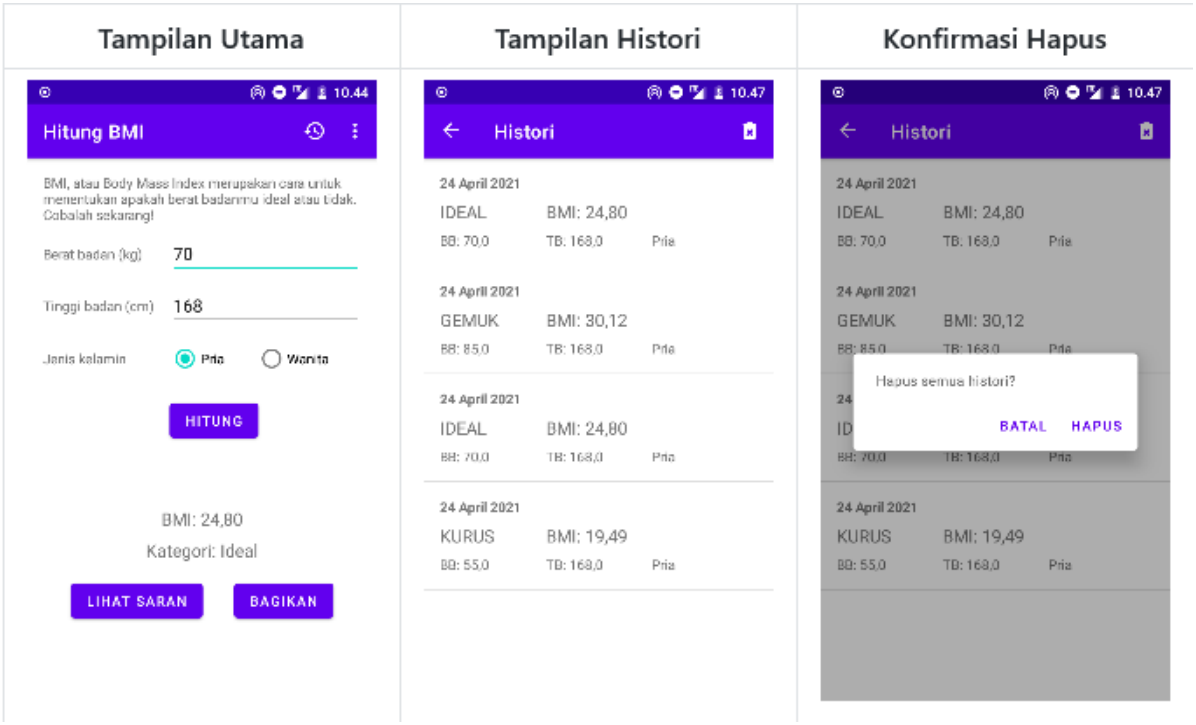
Modul App Architecture & RecyclerView

1. Overview

Pada modul kali ini, kita akan melanjutkan pembuatan aplikasi dari modul 5 (App Navigation), yaitu aplikasi untuk menghitung BMI. Aplikasi terakhir yang dibuat waktu itu merupakan aplikasi yang menggunakan beberapa layar untuk ditampilkan dengan pattern single activity – multiple fragment. Pengembangan aplikasi yang akan dikerjakan pada modul ini diantaranya:

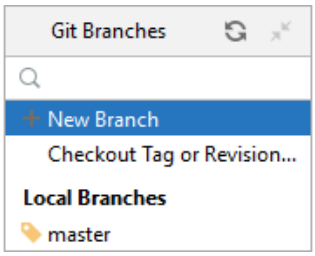
- mengadopsi desain arsitektur MVVM (Model-View-ViewModel),
- menambah fungsionalitas untuk menyimpan data histori BMI,
- menampilkan histori tersebut ke dalam list dengan RecyclerView, dan
- membuat fitur hapus histori dengan dialog konfirmasi penghapusan.

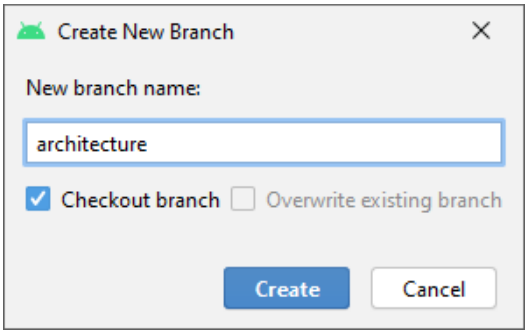
Berikut ini merupakan tampilan akhir dari aplikasi yang akan kita buat. Kode lengkapnya dapat dilihat di repository Github https://github.com/indraazimi/hitung-bmi. Nantinya, kita menggunakan library Android Jetpack seperti ViewModel, LiveData dan Room. Teori dan praktik dari komponen ini dapat dilihat materinya di Udacity & Google Codelab (link terdapat di LMS).



2. Getting Started

Karena modul 5 sudah ada Git-nya, di modul ini kita tidak membuat repository baru, namun cukup membuat branch baru saja. Buka project Hitung BMI dari modul 5 di Android Studio. Klik menu VCS > Git > Branches… lalu pilih New Branch. Masukkan “architecture” sebagai nama branch yang baru, lalu klik Create. Kerjakan task-task pada modul di branch baru ini.





Task pada modul ini juga akan membuat beberapa class baru. Pada pembuatannya, perhatikan penempatan package class tersebut, dan pastikan tidak menghapus kode package yang telah terbentuk pada class yang sudah ada. Karena kesalahan package dapat menyebabkan kode aplikasi menjadi error dan aplikasi menjadi tidak bisa dijalankan.

Dilarang keras untuk copy – paste kode dari modul/sumber lain!

Ngoding pelan-pelan akan membuat kamu lebih jago di masa depan.
Lakukan commit setiap selesai 1 sub-task. Selamat ngoding!

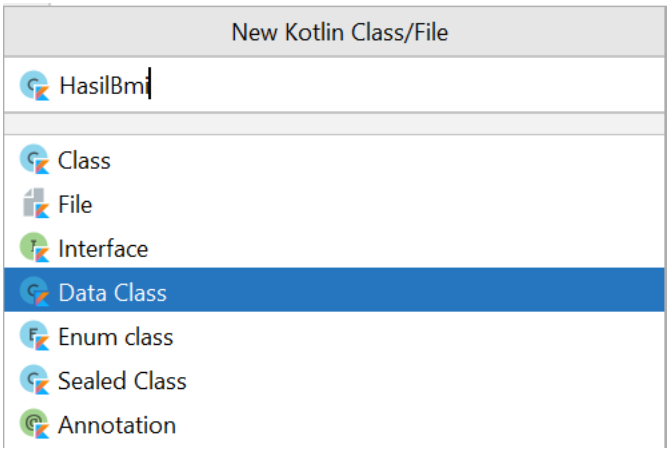
3. Task

3.1. Membuat view model untuk fitur hitung

Pada task ini, kita akan membuat komponen ViewModel dan LiveData untuk fitur hitung. Hal ini memastikan informasi hasil perhitungan tidak hilang ketika ada perubahan state. Langkah pertama yang perlu dilakukan yaitu menambahkan dependency. Pada app/build.gradle (module), tambahkan kode berikut di bagian dependencies, lalu lakukan Gradle sync.

```
dependencies {
    ...
    implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.3.1'
    implementation 'androidx.lifecycle:lifecycle-livedata-ktx:2.3.1'
}
```

Berikutnya, untuk menampung nilai BMI dan kategorinya dalam sebuah objek, diperlukan sebuah data kelas HasilBmi. Buatlah sebuah class baru dengan klik kanan pada package “data”, New → Kotlin Class/File. Berikan nama “HasilBmi”.



Buatlah isi dari data class HasilBmi sebagai berikut:

```
data class HasilBmi(
    val bmi: Float,
    val kategori: KategoriBmi
)
```

Selanjutnya, buat class HitungViewModel pada package ui. Klik kanan pada package ui, New → Kotlin File/Class, berikan informasi nama class “HitungViewModel”. Pada class ini, pindahkan proses perhitungan bmi dan kategorinya dari class HitungFragment. Sehingga, kode yang dihasilkan menjadi seperti di bawah ini.

```
class HitungViewModel : ViewModel() {

    // Hasil BMI bisa null jika pengguna belum menghitung BMI
    private val hasilBmi = MutableLiveData<HasilBmi?>()

    fun hitungBmi(berat: String, tinggi: String, isMale: Boolean) {
        val tinggiCm = tinggi.toFloat() / 100
        val bmi = berat.toFloat() / (tinggiCm * tinggiCm)
        val kategori = if (isMale) {
            when {
                bmi < 20.5 -> KategoriBmi.KURUS
                bmi >= 27.0 -> KategoriBmi.GEMUK
                else -> KategoriBmi.IDEAL
            }
        }
```

```
        else {
            when {
                bmi < 18.5 -> KategoriBmi.KURUS
                bmi >= 25.0 -> KategoriBmi.GEMUK
                else -> KategoriBmi.IDEAL
            }
        }
        hasilBmi.value = HasilBmi(bmi, kategori)
    }

    fun getHasilBmi() : LiveData<HasilBmi?> = hasilBmi
}
```

Pada kode di atas, class HitungViewModel meng-inherit ViewModel. Di class ini, kita melakukan perhitungan BMI dan juga menentukan kategorinya. Pemisahan proses perhitungan dan tampilan ini dimaksudkan untuk memastikan bahwa view memang hanya bertanggung jawab untuk menampilkan data, sedangkan variabel yang ditampilkan dan proses perhitungannya didapat dari komponen ViewModel untuk mempertahankan state dari variabel tersebut.

Perpindahan proses perhitungan dari HitungFragment ke HitungViewModel pastinya berdampak sehingga diperlukan perubahan pada class HitungFragment.kt. Perhatikan lalu sesuaikan kode yang dihapus dan ditambahkan pada class ini. Kode berwarna abu-abu menunjukkan tidak ada perubahan pada kode tersebut.

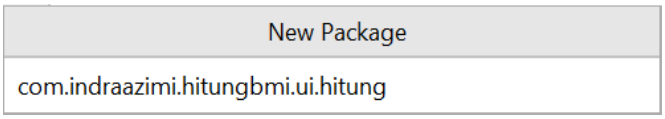
```
class HitungFragment : Fragment() {

    private val viewModel: HitungViewModel by viewModels()
    private lateinit var binding: FragmentHitungBinding
    private lateinit var kategoriBmi: KategoriBmi

    override fun onCreateView(...): View? {
        ...
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        viewModel.getHasilBmi().observe(viewLifecycleOwner, {
            if (it == null) return@observe
            binding.bmiTextView.text = getString(R.string.bmi_x, it.bmi)
            binding.kategoriTextView.text = getString(R.string.kategori_x,
                    getKategori(it.kategori))
            binding.buttonGroup.visibility = View.VISIBLE
        })
    }
```

```
    private fun hitungBmi() {
        ...

        val tinggi = binding.tinggiEditText.text.toString()
        if (TextUtils.isEmpty(tinggi)) {
            ...
        }
        val tinggiCm = tinggi.toFloat() / 100

        val selectedId = binding.radioGroup.checkedRadioButtonId
        ...
        val isMale = selectedId == R.id.priaRadioButton

        val bmi = berat.toFloat() / (tinggiCm * tinggiCm)
        val kategori = getKategori(bmi, isMale)

        binding.bmiTextView.text = getString(R.string.bmi_x, bmi)
        binding.kategoriTextView.text = getString(R.string.kategori_x, katgori)
        binding.buttonGroup.visibility = View.VISIBLE

        viewModel.hitungBmi(berat, tinggi, isMale)
    }

    private fun getKategori(bmi: Float, isMale: Boolean): String {
    private fun getKategori(kategori: KategoriBmi): String {
        kategoriBmi = if (isMale) {
            when {
                bmi < 20.5 -> KategoriBmi.KURUS
                bmi >= 27.0 -> KategoriBmi.GEMUK
                else -> KategoriBmi.IDEAL
            }
        } else {
            when {
                bmi < 18.5 -> KategoriBmi.KURUS
                bmi >= 25.0 -> KategoriBmi.GEMUK
                else -> KategoriBmi.IDEAL
            }
        }

        val stringRes = when (kategoriBmi) {
        val stringRes = when (kategori) {
            KategoriBmi.KURUS -> R.string.kurus
            KategoriBmi.IDEAL -> R.string.ideal
            KategoriBmi.GEMUK -> R.string.gemuk
        }
        return getString(stringRes)
    }

    private fun shareData() {
        ...
    }
}
```

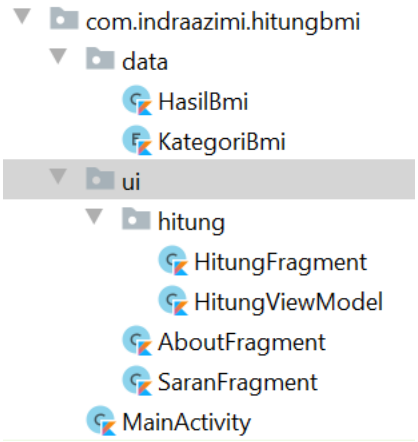
Jika telah selesai melakukan perubahan, jalankan aplikasi dan pastikan aplikasi dapat menghitung BMI dan menentukan kategori seperti sebelumnya. Lakukan perubahan state dengan merotasi layar. Pastikan hasil BMI tidak hilang. Apa yang terjadi ketika tombol lihat saran ditekan? Aplikasi akan crash. Cek error code yang muncul di bagian Logcat untuk mengetahui permasalahannya. Bug ini akan kita tangani di task selanjutnya. Untuk saat ini, lakukan saja commit sesuai judul task 3.1.

3.2. Memperbaiki navigasi dengan view model

Pada task ini, kita akan memperbaiki error dari task sebelumnya. Akan tetapi, sebelum itu, ubah dulu peletakan class HitungFragment.kt dan HitungViewModel.kt dari package /ui ke package /ui/hitung. Hal ini dimaksudkan agar file menjadi lebih rapi. Pertama, buat package baru bernama hitung. Klik kanan pada package ui, New → Package.



Berikutnya, drag-n-drop kedua file sebelumnya ke package ini. Sehingga, konfigurasi peletakan file pada bagian explorer sebagai berikut:



Pastikan file lain yang terpengaruh, menyesuaikan kodenya. Contohnya pada file navigation.xml di /res/navigation, pastikan name pada <fragment> mengacu pada package HitungFragment yang baru.

```
<?xml version="1.0" encoding="utf-8"?>
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    ...>
```

```
    <fragment
        android:id="@+id/hitungFragment"
        android:name="com.indraazimi.hitungbmi.ui.HitungFragment"
        android:name="com.indraazimi.hitungbmi.ui.hitung.HitungFragment"
        android:label="@string/app_name"
        tools:layout="@layout/fragment_hitung">
        ...
    </fragment>
    ...
</navigation>
```

Setelah melakukan perpindahan posisi file, kita akan memperbaiki error sebelumnya. Error pada task sebelumnya terjadi karena nilai kategoriBmi pada HitungFragment tidak ada nilai awalnya. Pada saat navigasi dilakukan, nilai kategoriBmi bukan bagian dari view model yang di-observe nilainya. Sehingga, ketika ada perubahan, nilai kategori ini masih dianggap belum berubah.

Untuk itu, modifikasi HitungViewModel.kt agar terdapat method yang dapat memastikan navigasi dapat dimulai atau tidak tergantung dari nilai kategori. Jika nilai kategori sudah ada, maka navigasi dapat dilakukan. Setelah selesai bernavigasi, kembalikan kategori ke nilai awalnya (yaitu menjadi kosong kembali) agar navigasi hanya terjadi sekali, tidak berulang-ulang.

```
class HitungViewModel : ViewModel() {

    // Hasil BMI bisa null jika pengguna belum menghitung BMI
    private val hasilBmi = MutableLiveData<HasilBmi?>()

    // Navigasi akan bernilai null ketika tidak bernavigasi
    private val navigasi = MutableLiveData<KategoriBmi?>()

    fun hitungBmi(berat: String, tinggi: String, isMale: Boolean) {
        ...
    }

    fun mulaiNavigasi() {
        navigasi.value = hasilBmi.value?.kategori
    }

    fun selesaiNavigasi() {
        navigasi.value = null
    }

    fun getHasilBmi() : LiveData<HasilBmi?> = hasilBmi

    fun getNavigasi() : LiveData<KategoriBmi?> = navigasi
}
```

Berikutnya, modifikasi HitungFragment.kt dengan menghapus variabel kategoriBmi dan meng-observe nilai navigasi. Jika navigasi bernilai tidak null, maka lakukan perpindahan layar.

```
class HitungFragment : Fragment() {

    private val viewModel: HitungViewModel by viewModels()
    private lateinit var binding: FragmentHitungBinding
    private lateinit var kategoriBmi: KategoriBmi

    override fun onCreateView(...): View? {
        ...
        binding.button.setOnClickListener { hitungBmi() }
        binding.saranButton.setOnClickListener { view: View ->
            view.findNavController().navigate(HitungFragmentDirections
                .actionHitungFragmentToSaranFragment(kategoriBmi))
        }
        binding.saranButton.setOnClickListener { viewModel.mulaiNavigasi() }
        binding.shareButton.setOnClickListener { shareData() }
        ...
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        viewModel.getNavigasi().observe(viewLifecycleOwner, {
            if (it == null) return@observe
            findNavController().navigate(HitungFragmentDirections
                    .actionHitungFragmentToSaranFragment(it))
            viewModel.selesaiNavigasi()
        })

        viewModel.getHasilBmi().observe(viewLifecycleOwner, {
            ...
        })
    }
    ...
}
```

Setelah selesai melakukan perubahan, jalankan aplikasi dan pastikan tombol lihat saran dapat ditekan dan fitur ini dijalankan dengan baik. Jika sudah sesuai, lakukan commit sesuai judul task.

3.3. Membuat database dengan Room

Berikutnya, kita akan membuat penyimpanan data menggunakan SQLite dengan memanfaatkan library Room. Langkah awal yang perlu dilakukan yaitu memodifikasi project/build.gradle dengan menambahkan nilai versi Room yang akan digunakan.

```
buildscript {
    ext.kotlin_version = "1.4.10"
    ext.nav_version = "2.3.3"
    ext.room_version = "2.2.6"
    repositories {
        google()
        jcenter()
    }
    ...
}
...
```

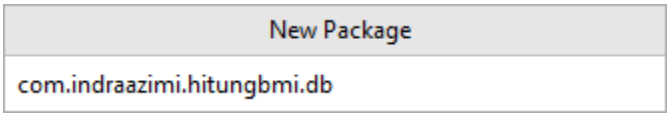
Tambahkan dependency pada app/build.gradle sebagai berikut. Nilai $room_version mengacu pada nilai di project/build.gradle. Selanjutnya lakukan Gradle sync.

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'kotlin-kapt'
    id 'androidx.navigation.safeargs.kotlin'
}
...

dependencies {
    ...
    implementation "androidx.room:room-runtime:$room_version"
    kapt "androidx.room:room-compiler:$room_version"
    ...
}
```

Untuk menambahkan penyimpanan SQLite menggunakan Room, diperlukan 3 komponen:

1. File Dao: Singkatan dari Data Access Object. File ini berupa interface dengan method berupa operasi yang dilakukan untuk melakukan penyimpanan/modifikasi data. Untuk setiap method terdapat annotation (tanda @) yang menyatakan operasi terhadap database. File ini “menjembatani” antara database & bahasa pemrograman.
2. File Entitas/Entity: File ini berupa class untuk mewakili entitas data yang disimpan. Terdapat konfigurasi berupa primary key, nama tabel, dan nama kolom (jika nama kolom berbeda dengan nama variabel). File ini mewakili tabel untuk penyimpanan data.
3. File DB: Merupakan sebuah abstract class yang meng-extends RoomDatabase. Pada class ini, kita mendata entity dan juga Dao yang digunakan pada database. Terdapat juga kode untuk mengambil instance/objek dari class ini.

Pada task ini, kita akan membuat ketiga file tersebut di dalam package “db”. Klik kanan pada package project masing-masing, New → Package. Tuliskan db sebagai package baru.



Berikutnya, buat sebuah class BmiEntity.kt pada package yang telah dibuat sebelumnya.

```
@Entity(tableName = "bmi")
data class BmiEntity(
    @PrimaryKey(autoGenerate = true)
    var id: Long = 0L,
    var tanggal: Long = System.currentTimeMillis(),
    var berat: Float,
    var tinggi: Float,
    var isMale: Boolean
)
```

Tambahkan sebuah interface Dao bernama BmiDao.kt, masih pada package “db”.

```
@Dao
interface BmiDao {

    @Insert
    fun insert(bmi: BmiEntity)

    @Query("SELECT * FROM bmi ORDER BY id DESC LIMIT 1")
    fun getLastBmi(): LiveData<BmiEntity?>
}
```

Terakhir, tambahkan class BmiDb.kt pada package “db”.

```
@Database(entities = [BmiEntity::class], version = 1, exportSchema = false)
abstract class BmiDb : RoomDatabase() {

    abstract val dao: BmiDao

    companion object {

        @Volatile
        private var INSTANCE: BmiDb? = null

        fun getInstance(context: Context): BmiDb {
            synchronized(this) {
                var instance = INSTANCE

                if (instance == null) {
```

```
                    instance = Room.databaseBuilder(
                        context.applicationContext,
                        BmiDb::class.java,
                        "bmi.db"
                    )
                        .fallbackToDestructiveMigration()
                        .build()
                    INSTANCE = instance
                }
                return instance
            }
        }
    }
}
```

Pada task kali ini, kita sudah menambahkan komponen penting untuk melakukan penyimpanan data. Akan tetapi, belum digunakan untuk jalannya fungsi tersebut. Pastikan tidak ada kode yang error dengan cara menjalankan aplikasi. Lakukan commit sesuai judul task.

3.4. Menyimpan data ke database

Pada task ini, kita akan menggunakan beberapa class yang telah dibuat sebelumnya untuk melakukan penyimpanan ke database dan mengambil data BMI terakhir yang tersimpan untuk memastikan data benar-benar telah tersimpan. Langkah awal yang perlu dilakukan yaitu memanggil operasi pada Dao dari ViewModel. Hal tersebut dilakukan karena ViewModel bertugas untuk melakukan operasi penyimpanan dari input yang didapat dari view atau mengambil data untuk ditampilkan pada view. Dalam hal ini, view adalah Fragment.

Karena ViewModel menggunakan komponen Dao, diperlukan pengiriman objek Dao pada konstruktor ViewModel. Sehingga, pada kasus ini, kita perlu memodifikasi konstruktor class HitungViewModel.kt dan juga menambahkan kode pemanggilan insert() dan getLastBmi().

```
class HitungViewModel() : ViewModel() {
class HitungViewModel(private val db: BmiDao) : ViewModel() {

    ...
    private val navigasi = MutableLiveData<KategoriBmi?>()

    // Variabel ini sudah berupa LiveData (tidak mutable),
    // sehingga tidak perlu dijadikan private
    val data = db.getLastBmi()
```

```
    fun hitungBmi(berat: String, tinggi: String, isMale: Boolean) {
        ...
        hasilBmi.value = HasilBmi(bmi, kategori)

        viewModelScope.launch {
            withContext(Dispatchers.IO) {
                val dataBmi = BmiEntity(
                    berat = berat.toFloat(),
                    tinggi = tinggi.toFloat(),
                    isMale = isMale
                )
                db.insert(dataBmi)
            }
        }
    }
    ...
}
```

Perhatikan bahwa pemanggilan method insert berada pada block coroutine. Hal ini dilakukan agar prosesnya berjalan secara asinkron terhadap main thread (proses yang memperlihatkan tampilan pada pengguna). Couroutine memerlukan scope saat berjalan, dan untuk komponen ViewModel, scope ini sudah disediakan, yaitu viewModelScope.

Karena HitungViewModel.kt memerlukan sebuah parameter pada konstruktornya, kita tidak bisa lagi menggunakan kode “by viewModels()” untuk membuat objek ViewModel. Namun kita perlu sebuah komponen ViewModelFactory untuk membuat objek ViewModel tersebut. Untuk itu, buat class baru pada package hitung bernama HitungViewModelFactory.kt dengan kode ini:

```
class HitungViewModelFactory(
    private val db: BmiDao
) : ViewModelProvider.Factory {
    @Suppress("unchecked_cast")
    override fun <T : ViewModel?> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(HitungViewModel::class.java)) {
            return HitungViewModel(db) as T
        }
        throw IllegalArgumentException("Unknown ViewModel class")
    }
}
```

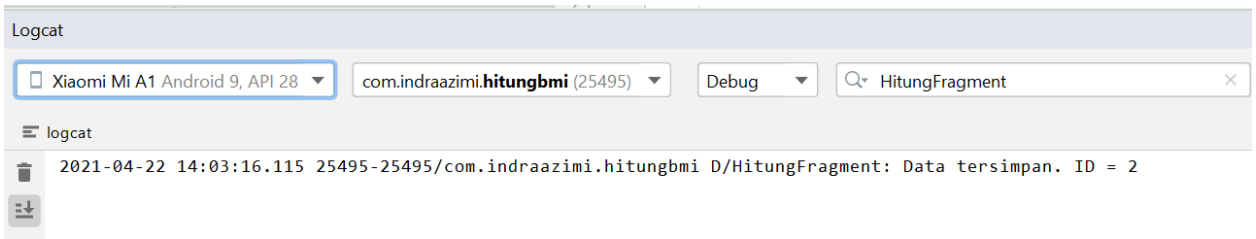
Berikutnya, lakukan modifikasi pada file HitungFragment.kt agar menggunakan ViewModelFactory. Selain itu, kita juga akan menambahkan kode untuk meng-observe variabel “data” dari ViewModel yang mengambil nilai data BMI yang terakhir disimpan.

Kode di bawah ini memperlihatkan modifikasi dari HitungFragment.kt.

```
class HitungFragment : Fragment() {

    private val viewModel: HitungViewModel by viewModels()
    private val viewModel: HitungViewModel by lazy {
        val db = BmiDb.getInstance(requireContext())
        val factory = HitungViewModelFactory(db.dao)
        ViewModelProvider(this, factory).get(HitungViewModel::class.java)
    }

    private lateinit var binding: FragmentHitungBinding

    override fun onCreateView(...): View? {
        ...
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        ...
        viewModel.getHasilBmi().observe(viewLifecycleOwner, {
            ...
        })

        viewModel.data.observe(viewLifecycleOwner, {
            if (it == null) return@observe
            Log.d("HitungFragment", "Data tersimpan. ID = ${it.id}")
        })
    }
    ...
}
```

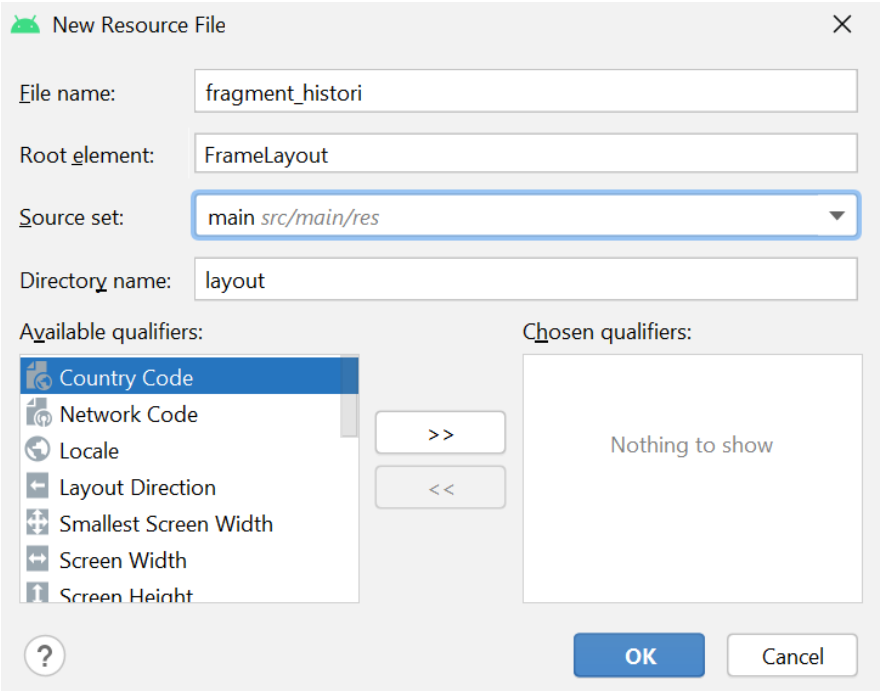
Perhatikan bahwa pembentukan objek ViewModel menggunakan lazy initialization. Inisialisasi ini akan membuat objek viewModel baru “terisi” ketika pemanggilan terhadap objek ini dilakukan pertama kali. Dengan kata lain, variabel ini tidak akan diinisialisasi jika kita tidak menggunakannya.

Jika telah selesai mengkodekan, pastikan tidak ada kode yang error dan jalankan aplikasi. Hitung BMI, lalu perhatikan, layar Logcat akan menampilkan ID dari data yang disimpan terakhir kali. Jika sudah memastikan hal ini berjalan dengan baik, lakukan commit sesuai judul task.



3.5. Menambahkan fitur histori

Untuk menampilkan data histori yang disimpan, kita memerlukan menu untuk di-klik dan tampilan yang memperlihatkan data yang sudah disimpan sebelumnya. Sehingga, diperlukan penambahan tampilan awal berupa fragment. Langkah awal, buat sebuah layout baru. Klik kanan pada /res/layout, New → Layout Resource File. Beri nama fragment_histori untuk layout tersebut.



Kode dari tampilan tersebut (fragment_histori.xml) dan strings.xml-nya adalah sebagai berikut:

```xml
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/emptyView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="@string/belum_ada_data" />
</FrameLayout>
```

```xml
<resources>
    ...
    <string name="belum_ada_data">Belum ada data</string>
</resources>
```

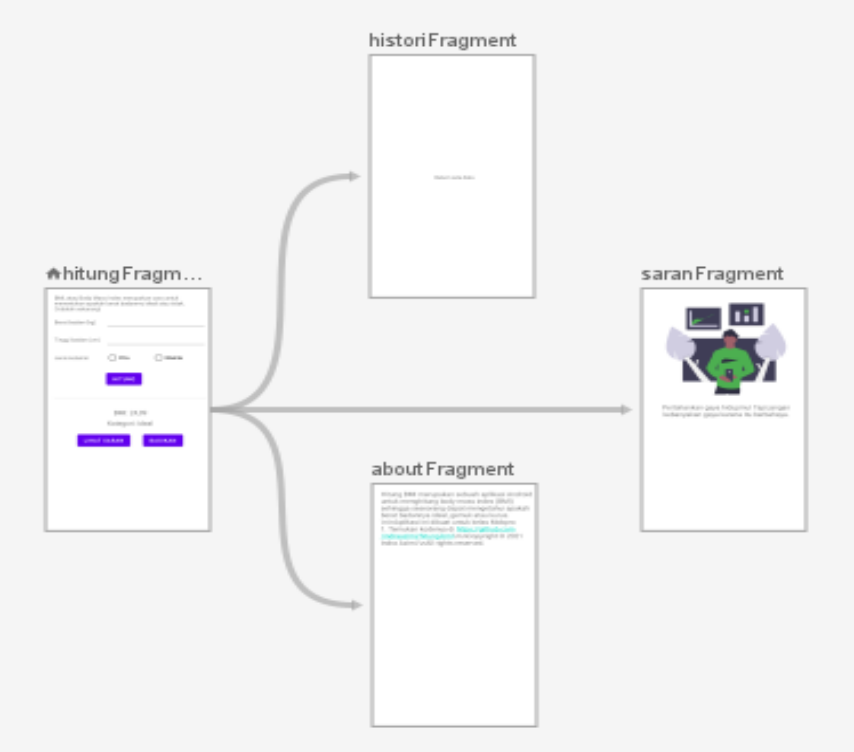
Berikutnya, buat fragment untuk menampilkan layout tersebut. Fragment ini kita bedakan package-nya dari class yang sudah kita buat sebelumnya. Hal ini dilakukan agar file dari project lebih rapi dan terstruktur. Klik kanan package ui, pilih New → Package, berikan nama histori dari package ui.

New Package
com.indraazimi.hitungbmi.ui.histori

Pada package ini, tambahkan sebuah class bernama HistoriFragment.kt.

```
class HistoriFragment : Fragment() {

    private lateinit var binding: FragmentHistoriBinding

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                              savedInstanceState: Bundle?): View? {
        binding = FragmentHistoriBinding.inflate(layoutInflater,
                  container, false)
        return binding.root
    }
}
```

Pada navigations.xml, tambahkan fragment yang telah dibuat sebelumnya. Tambahkan pula aksi dari hitungFragment ke historiFragment sehingga navigation graph kita menjadi seperti ini:

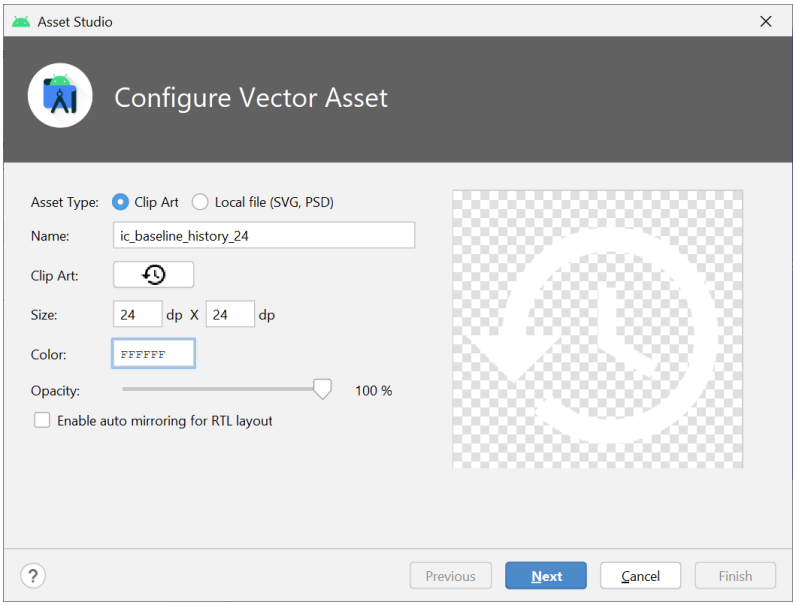


Berikut kode navigation.xml dan strings.xml untuk digunakan pada navigasi di atas.

```
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    ...>

    <fragment
        android:id="@+id/hitungFragment"
        ...
        <action
            android:id="@+id/action_hitungFragment_to_aboutFragment"
            app:destination="@id/aboutFragment" />
        <action
            android:id="@+id/action_hitungFragment_to_historiFragment"
            app:destination="@id/historiFragment" />
    </fragment>
    ...
    <fragment
        android:id="@+id/historiFragment"
        android:name="com.indraazimi.hitungbmi.ui.histori.HistoriFragment"
        android:label="@string/histori"
        tools:layout="@layout/fragment_histori" />
</navigation>
```

```
<resources>
    ...
    <string name="histori">Histori</string>
</resources>
```

Selanjutnya, tambahkan menu pada option menu yang sudah ada. Menu ini memerlukan sebuah icon. Jadi, kita akan menambahkan icon ke project Android terlebih dahulu. Klik kanan pada bagian /res/drawable, New → Image Vector Asset. Pilih clip art berupa history, dan atur warna menjadi putih.



Gunakan icon ini pada menu. Pada options_menu.xml di /res/menu/, tambahkan sebuah menu dengan pengaturan icon & showAsAction bersifat ifRoom.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item
        android:id="@+id/menu_histori"
        android:icon="@drawable/ic_baseline_history_24"
        android:title="@string/histori"
        app:showAsAction="ifRoom" />
    ...
</menu>
```

Tambahkan aksi klik pada menu ini dengan mengubah method “onOptionsItemSelected”.

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    if (item.itemId == R.id.menu_about) {
        findNavController().navigate
                (R.id.action_hitungFragment_to_aboutFragment)
        return true
    }

    when(item.itemId) {
        R.id.menu_histori -> {
            findNavController().navigate
                  (R.id.action_hitungFragment_to_historiFragment)
            return true
        }
        R.id.menu_about -> {
            findNavController().navigate
                  (R.id.action_hitungFragment_to_aboutFragment)
            return true
        }
    }
    return super.onOptionsItemSelected(item)
}
```

Jalankan aplikasi, dan pastikan terdapat menu yang muncul. Jika diklik, menu ini akan mengarahkan kita ke tampilan HistoriFragment. Hal yang ditampilkan adalah sebuah TextView dengan tulisan “Belum ada data”. Kita akan mengambil data dari database di task selanjutnya. Jika aplikasi sudah berjalan dengan baik sesuai yang seharusnya, lakukan commit sesuai judul task.

3.6. Mengambil data dari database

Pada task ini, kita akan mengambil data histori perhitungan BMI dari database. Karena kita perlu seluruh data, kita harus mengubah method getLastBmi() dengan menghapus klausa LIMIT 1. Adapun ORDER BY id DESC tetap dipertahankan karena lazimnya histori diurutkan dari data yang paling baru ke yang paling lama. Pada BmiDao.kt, ubah method getLastBmi:

```
@Dao
interface BmiDao {

    @Insert
    fun insert(bmi: BmiEntity)

    @Query("SELECT * FROM bmi ORDER BY id DESC LIMIT 1")
    fun getLastBmi(): LiveData<BmiEntity?>
    @Query("SELECT * FROM bmi ORDER BY id DESC")
    fun getLastBmi(): LiveData<List<BmiEntity>>
}
```

Pengambilan data biasanya dilakukan pada ViewModel, dan ditampilkan pada Fragment. Kita sudah memiliki HistoriFragment, tapi belum ada ViewModel untuk ini. Jadi pada package histori, klik kanan dan pilih New → Kotlin Class/File. Beri nama HistoriViewModel, lalu sesuaikan isinya:

```
class HistoriViewModel(db: BmiDao) : ViewModel() {
    val data = db.getLastBmi()
}
```

Seperti sebelumnya, karena mengakses data memerlukan objek Dao dan objek ini dikirimkan melalui parameter konstruktor dari ViewModel, maka kita memerlukan sebuah ViewModelFactory. Klik kanan pada package histori, pilih New → Kotlin Class/File. Beri nama HistoriViewModelFactory.

```
class HistoriViewModelFactory(
    private val db: BmiDao
) : ViewModelProvider.Factory {
    @Suppress("unchecked_cast")
    override fun <T : ViewModel?> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(HistoriViewModel::class.java)) {
            return HistoriViewModel(db) as T
        }
        throw IllegalArgumentException("Unknown ViewModel class")
    }
}
```

Pada HistoriFragment.kt, buat objek ViewModel menggunakan lazy initialization, dan observe variabel "data" yang menyimpan nilai database. Gunakan Log untuk memastikan data telah berhasil diambil.

```
class HistoriFragment : Fragment() {

    private val viewModel: HistoriViewModel by lazy {
        val db = BmiDb.getInstance(requireContext())
        val factory = HistoriViewModelFactory(db.dao)
        ViewModelProvider(this, factory).get(HistoriViewModel::class.java)
    }

    private lateinit var binding: FragmentHistoriBinding

    override fun onCreateView(...): View? {
        ...
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        viewModel.data.observe(viewLifecycleOwner, {
            Log.d("HistoriFragment", "Jumlah data: ${it.size}")
        })
    }
}
```

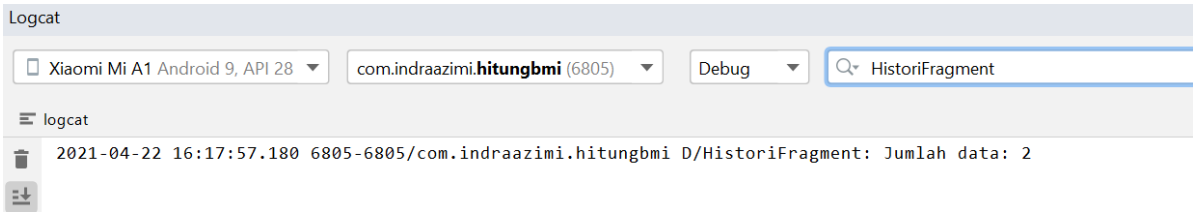
Sampai di sini, kita telah mengambil data dari database di HistoriFragment. Hal ini menjadikan kode pengambilan data pada HitungViewModel.kt menjadi tidak lagi relevan dan dapat dihapus.

```
class HitungViewModel(private val db: BmiDao) : ViewModel() {
    ...
    // Variabel ini sudah berupa LiveData (tidak mutable),
    // sehingga tidak perlu dijadikan private
    val data = db.getLastBmi()
    ...
}
```

Setelah dihapus, HitungFragment.kt akan menjadi error. Hapus blok kode yang error tersebut:

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    ...
    viewModel.data.observe(viewLifecycleOwner, {
        if (it == null) return@observe
        Log.d("HitungFragment", "Data tersimpan. ID = ${it.id}")
    })
}
```

Pastikan tidak ada kode yang error, dan jalankan aplikasi. Saat menjalankan tampilan HistoriFragment, pada Logcat akan ditampilkan jumlah data tersimpan.



Jika aplikasi sudah berjalan sesuai yang diharapkan, lakukan commit sesuai judul task.

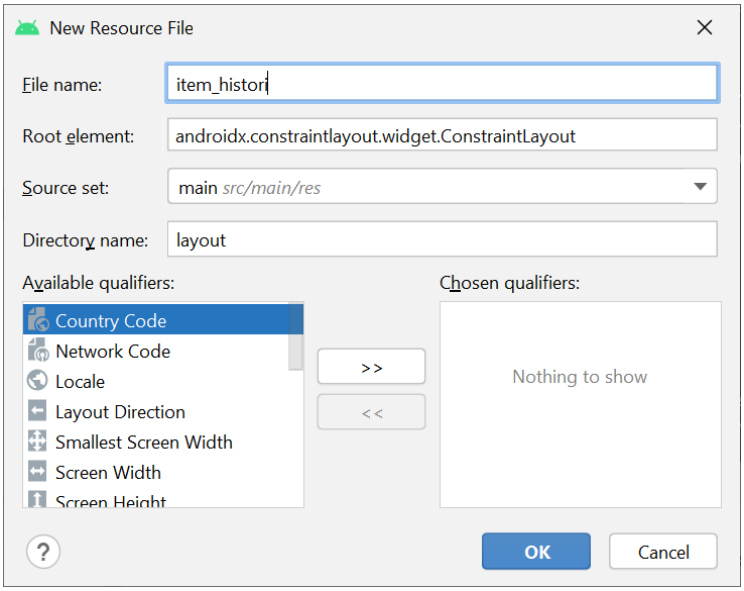
3.7. Menampilkan data dalam bentuk list

Sejauh ini, kita baru melakukan pengambilan data dari database dan belum menampilkan data tersebut ke pengguna. Jadi, di task ini kita akan menampilkan data tersebut ke dalam bentuk list menggunakan RecyclerView. Ini adalah lanjutan materi RecyclerView di modul 04. Saat itu data yang ditampilkan bersifat statis (ditulis di kodingan), sedangkan sekarang datanya lebih dinamis karena diambil dari database. Silahkan baca kembali modul 04 jika diperlukan.

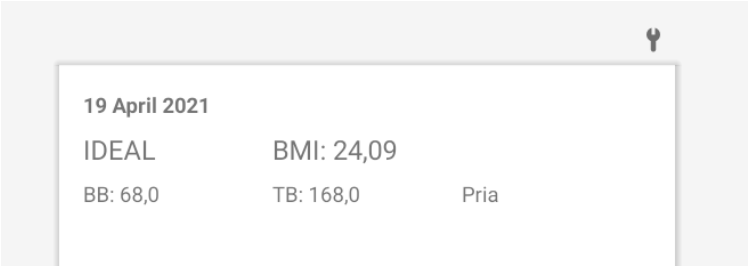
Sebagai langkah awal, tambahkan RecyclerView pada fragment_histori.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout 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"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
        tools:listitem="@layout/item_histori" />

    <TextView
        android:id="@+id/emptyView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="@string/belum_ada_data" />
</FrameLayout>
```

Selanjutnya, tambahkan 1 layout sebagai konfigurasi tampilan untuk menampilkan 1 item. Klik kanan pada /res/layout, pilih New → Layout Resource File. Beri nama layout ini item_histori.



Atur tampilan pada layout ini sebagai berikut:



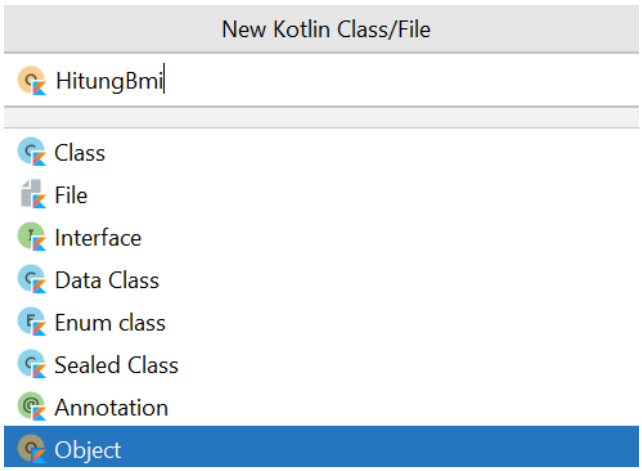

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <TextView
        android:id="@+id/tanggalTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="16dp"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:text="19 April 2021" />
```

```
    <TextView
        android:id="@+id/kategoriTextView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="8dp"
        android:textSize="18sp"
        app:layout_constraintEnd_toEndOf="@+id/beratTextView"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/tanggalTextView"
        tools:text="IDEAL" />

    <TextView
        android:id="@+id/bmiTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        app:layout_constraintStart_toEndOf="@+id/kategoriTextView"
        app:layout_constraintTop_toTopOf="@+id/kategoriTextView"
        tools:text="BMI: 24,09" />

    <TextView
        android:id="@+id/beratTextView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="8dp"
        android:layout_marginBottom="16dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/tinggiTextView"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintHorizontal_chainStyle="spread"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/kategoriTextView"
        tools:text="BB: 68,0" />

    <TextView
        android:id="@+id/tinggiTextView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintEnd_toStartOf="@+id/genderTextView"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toEndOf="@+id/beratTextView"
        app:layout_constraintTop_toTopOf="@+id/beratTextView"
        tools:text="TB: 168,0" />
```

```
    <TextView
        android:id="@+id/genderTextView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginEnd="16dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toEndOf="@+id/tinggiTextView"
        app:layout_constraintTop_toTopOf="@+id/beratTextView"
        tools:text="Pria" />

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

Langkah berikutnya yang harus dilakukan adalah membuat adapter untuk RecyclerView yang sudah dibuat sebelumnya. Karena adapter ini juga membutuhkan perhitungan BMI dan kategorinya, maka untuk menghindari redundansi kode, kita akan memindahkan proses perhitungan BMI dan kategori di HitungViewModel ke sebuah objek tersendiri agar kode tersebut dapat dipakai juga di adapter kita.

Untuk melakukan hal ini, bentuk sebuah objek bernama HitungBmi.kt. Klik kanan pada package "data", pilih New → Kotlin Class/File. Berikan nama HitungBmi sebagai nama file, lalu pilih "Object" sebagai kategorinya seperti terlihat pada gambar berikut.



Di dalam objek tersebut, buat method hitung, isi dengan kode perhitungan BMI & kategorinya. Kode diambil dari HitungViewModel.kt dengan sedikit penyesuaian.

```
object HitungBmi {

    fun hitung(data: BmiEntity): HasilBmi {
        val tinggiCm = data.tinggi / 100
        val bmi = data.berat / (tinggiCm * tinggiCm)
```

```
        val kategori = if (data.isMale) {
            when {
                bmi < 20.5 -> KategoriBmi.KURUS
                bmi >= 27.0 -> KategoriBmi.GEMUK
                else -> KategoriBmi.IDEAL
            }
        } else {
            when {
                bmi < 18.5 -> KategoriBmi.KURUS
                bmi >= 25.0 -> KategoriBmi.GEMUK
                else -> KategoriBmi.IDEAL
            }
        }
        return HasilBmi(bmi, kategori)
    }
}
```

Kode perhitungan BMI & penentuan kategori tidak lagi diperlukan pada HitungViewModel, sehingga diperlukan modifikasi kode sebagai berikut:

```
class HitungViewModel(private val db: BmiDao) : ViewModel() {
    ...
    fun hitungBmi(berat: String, tinggi: String, isMale: Boolean) {
        val tinggiCm = tinggi.toFloat() / 100
        val bmi = berat.toFloat() / (tinggiCm * tinggiCm)
        val kategori = if (isMale) {
            when {
                bmi < 20.5 -> KategoriBmi.KURUS
                bmi >= 27.0 -> KategoriBmi.GEMUK
                else -> KategoriBmi.IDEAL
            }
        } else {
            when {
                bmi < 18.5 -> KategoriBmi.KURUS
                bmi >= 25.0 -> KategoriBmi.GEMUK
                else -> KategoriBmi.IDEAL
            }
        }
        hasilBmi.value = HasilBmi(bmi, kategori)

        val dataBmi = BmiEntity(
            berat = berat.toFloat(),
            tinggi = tinggi.toFloat(),
            isMale = isMale
        )
        hasilBmi.value = HitungBmi.hitung(dataBmi)
```

```
        viewModelScope.launch {
            withContext(Dispatchers.IO) {
                val dataBmi = BmiEntity(
                    berat = berat.toFloat(),
                    tinggi = tinggi.toFloat(),
                    isMale = isMale
                )
                db.insert(dataBmi)
            }
        }
    }
    ...
}
```

Setelah melakukan modifikasi kode, kita dapat membuat adapter. Pada package histori, klik kanan & pilih New → Kotlin Class/File. Tuliskan HistoriAdapter sebagai nama file.

```
class HistoriAdapter : RecyclerView.Adapter<HistoriAdapter.ViewHolder>() {

    private val data = mutableListOf<BmiEntity>()

    fun updateData(newData: List<BmiEntity>) {
        data.clear()
        data.addAll(newData)
        notifyDataSetChanged()
    }

    override fun onCreateViewHolder(parent: ViewGroup,
                                    viewType: Int): ViewHolder {
        val inflater = LayoutInflater.from(parent.context)
        val binding = ItemHistoriBinding.inflate(inflater, parent, false)
        return ViewHolder(binding)
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        holder.bind(data[position])
    }

    override fun getItemCount(): Int {
        return data.size
    }

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

        private val dateFormatter = SimpleDateFormat("dd MMMM yyyy",
            Locale("id", "ID"))
```

```
        fun bind(item: BmiEntity) = with(binding) {
            tanggalTextView.text = dateFormatter.format(Date(item.tanggal))

            val hasilBmi = HitungBmi.hitung(item)
            kategoriTextView.text = hasilBmi.kategori.toString()
            bmiTextView.text = root.context.getString(R.string.bmi_x,
                hasilBmi.bmi)

            beratTextView.text = root.context.getString(R.string.berat_x,
                item.berat)
            tinggiTextView.text = root.context.getString(R.string.tinggi_x,
                item.tinggi)
            val stringRes = if (item.isMale) R.string.pria else R.string.wanita
            genderTextView.text = root.context.getString(stringRes)
        }
    }
}
```

Class adapter di atas menampilkan string dari strings.xml sebagai berikut:

```
<resources>
    ...
    <string name="berat_x">BB: %1$.1f</string>
    <string name="tinggi_x">TB: %1$.1f</string>
    ...
</resources>
```

Class adapter di atas, hampir sama dengan modul sebelumnya. Perbedaannya, terdapat tambahan method updateData. Method ini dapat dipanggil jika ada perubahan data, sehingga komponen RecyclerView dapat mengubah data yang ia tampilkan.

Atur RecyclerView dan adapter yang telah dibuat sebelumnya agar ditampilkan pada HistoriFragment:

```
class HistoriFragment : Fragment() {
    ...
    private lateinit var binding: FragmentHistoriBinding
    private lateinit var myAdapter: HistoriAdapter

    override fun onCreateView(...): View? {
        binding = FragmentHistoriBinding.inflate(...)
        myAdapter = HistoriAdapter()
        with(binding.recyclerView) {
            addItemDecoration(DividerItemDecoration(context,
                RecyclerView.VERTICAL))
            adapter = myAdapter
            setHasFixedSize(true)
        }
        return binding.root
    }
```

```
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        viewModel.data.observe(viewLifecycleOwner, {
            Log.d("HistoriFragment", "Jumlah data: ${it.size}")
            binding.emptyView.visibility = if (it.isEmpty())
                View.VISIBLE else View.GONE
            myAdapter.updateData(it)
        })
    }
}
```

Setelah melakukan penambahan kode di atas, jalankan aplikasi. Pastikan menu histori pada aplikasi dapat menampilkan data histori dari database dalam bentuk list. Lakukan commit sesuai judul task jika app sudah berjalan sesuai harapan.

3.8. Menghapus data dari database

Pada task kali ini, kita akan menambahkan fungsi hapus semua data histori. Aksi hapus dilakukan dengan cara menyediakan sebuah option menu pada HistoriFragment. Karena penghapusan data bersifat permanen, maka sebagai tindakan preventif, fungsi hapus ini baru akan dijalankan setelah pengguna melakukan konfirmasi penghapusan dari Alert Dialog.

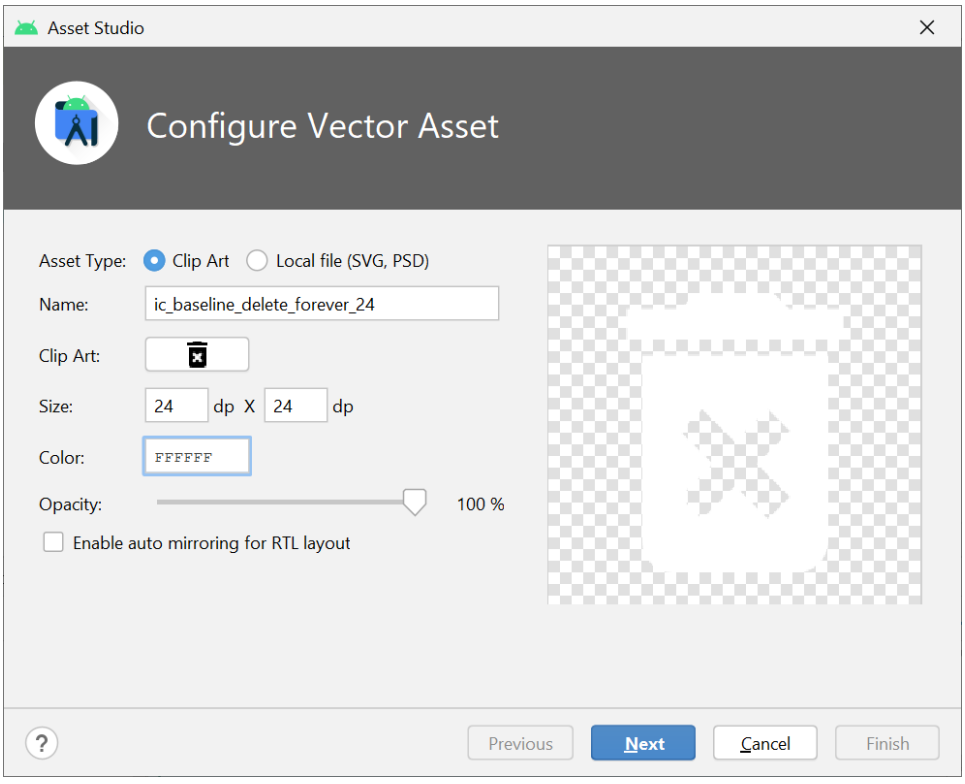
Langkah pertama yang perlu dilakukan yaitu menambahkan method hapus pada BmiDao.kt.

```
@Dao
interface BmiDao {

    @Insert
    fun insert(bmi: BmiEntity)

    @Query("SELECT * FROM bmi ORDER BY id DESC")
    fun getLastBmi(): LiveData<List<BmiEntity>>

    @Query("DELETE FROM bmi")
    fun clearData()
}
```

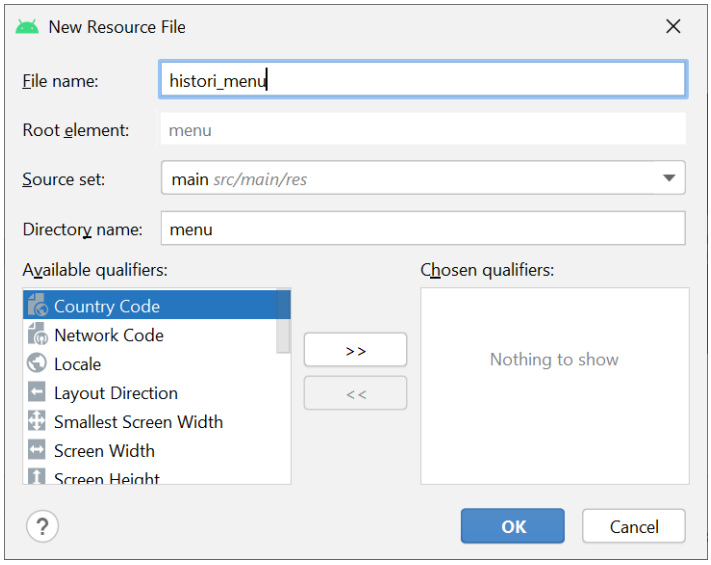
Berikutnya, panggil method ini pada HistoriViewModel.kt. Selain menambahkan pemanggilan, kita juga akan menambahkan modifier private pada parameter konstruktornya.

```
class HistoriViewModel(db: BmiDao) : ViewModel() {
class HistoriViewModel(private val db: BmiDao) : ViewModel() {

    val data = db.getLastBmi()

    fun hapusData() = viewModelScope.launch {
        withContext(Dispatchers.IO) {
            db.clearData()
        }
    }
}
```

Berikutnya, tambahkan options menu untuk HistoriFragment. Menu ini menggunakan sebuah icon yang dapat kita tambahkan. Klik kanan pada /res/drawable, New → Vector Asset, pilih clip art delete_forever dengan konfigurasi warna putih.



Setelah membuat icon, berikutnya membuat menu yang nanti dimunculkan pada fragment. Klik kanan pada /res/menu, New → Menu Resource File. Berikan nama histori_menu.



Kode XML dari menu & strings.xml yang digunakan pada menu adalah sebagai berikut:

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item
        android:id="@+id/menu_hapus"
        android:icon="@drawable/ic_baseline_delete_forever_24"
        android:title="@string/hapus_histori"
        app:showAsAction="ifRoom" />
</menu>
```

```xml
<resources>
    ...
    <string name="hapus_histori">Hapus Histori</string>
    ...
</resources>
```

Tampilkan menu di atas pada HistoriFragment.kt. Caranya sama seperti penjelasan di modul 05 (App Navigation), yaitu dengan meng-override method onCreateOptionsMenu & berikan aksinya dengan override method onOptionsItemSelected. Dikarenakan menu ini dimunculkan pada fragment, pastikan ada kode setHasOptionsMenu(true).

```
class HistoriFragment : Fragment() {
    ...
    override fun onCreateView(...): View? {
        ...
        with(binding.recyclerView) {
            ...
        }
        setHasOptionsMenu(true)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        ...
    }

    override fun onCreateOptionsMenu(menu: Menu, inflater: MenuInflater) {
        super.onCreateOptionsMenu(menu, inflater)
        inflater.inflate(R.menu.histori_menu, menu)
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        if (item.itemId == R.id.menu_hapus) {
            hapusData()
            return true
        }
        return super.onOptionsItemSelected(item)
    }

    private fun hapusData() {
        MaterialAlertDialogBuilder(requireContext())
                .setMessage(R.string.konfirmasi_hapus)
                .setPositiveButton(getString(R.string.hapus)) { _, _ ->
                    viewModel.hapusData()
                }
                .setNegativeButton(getString(R.string.batal)) { dialog, _ ->
                    dialog.cancel()
                }
                .show()
    }
}
```

Kode di atas menggunakan pengambilan string dari strings.xml sebagai berikut.

```
<resources>
    ...
    <string name="konfirmasi_hapus">Hapus semua histori?</string>
    <string name="hapus">Hapus</string>
    <string name="batal">Batal</string>
    ...
</resources>
```

Pada HistoriFragment.kt terdapat method hapusData() yang menampilkan AlertDialogmenggunakan MaterialAlertDialogBuilder. Terdapat 2 button yang digunakan, yaitu hapus (positive button—muncul di bagian kanan) & batal (negative button—muncul di bagian kiri). Jika tombol hapus ditekan, akan dipanggil method hapusData() yang ada di HistoriViewModel. Sedangkan, jika tombol batal ditekan, akan menutup AlertDialogtersebut, dan kembali menampilkan fragment.

Jalankan aplikasi, dan pastikan fungsi sebelumya masih tetap dapat dijalankan. Tambahan fungsi pada task ini yaitu adanya menu hapus histori data. Jika menu ini ditekan, ada mekanisme meminta konfirmasi pengguna melalui AlertDialog. Pastikan dengan memilih tombol yang ada, output-nya sesuai harapan. Jika aplikasi sudah berjalan dengan baik dan lancar, lakukan commit sesuai judul task.

Praktikum telah usai. Selanjutnya pastikan commit yang dilakukan sudah sesuai dengan jumlah task. Berikutnya, lakukan push kode project ke repository Github menggunakan menu VCS > Git > Push… Refresh repository Hitung BMI modul 05 di browser dan pastikan kodenya sudah sesuai dengan kode versi lokal (komputer/laptop masing-masing). Kereeen, kamu bisa!! 👍

4. Summary

Pada modul kali ini kita telah membuat sebuah aplikasi Android yang menggunakan arsitektur MVVM dan memiliki fungsionalitas untuk menyimpan data secara lokal, lalu menampilkannya ke dalam bentuk list. Terdapat beberapa komponen yang kita gunakan seperti ViewModel, LiveData, Room dan juga RecyclerView.

5. Challenge

Setelah mengikuti langkah per langkah, kita telah membuat aplikasi yang memiliki kemampuan menambahkan/menyimpan data dan juga menghapusnya. Sebagai challenge untuk memahami cara kerja materi pada modul, lakukan penambahan fungsi update data pada aplikasi ini. Selain itu, bisa juga ditambahkan mekanisme swipe atau tekan 1 item di RecyclerView yang membuat data dapat dihapus per item.