MODUL 9
Room Database

CAPAIAN PEMBELAJARAN

1. Mahasiswa dapat membuat Aplikasi dengan database.

KEBUTUHAN ALAT/BAHAN/SOFTWARE

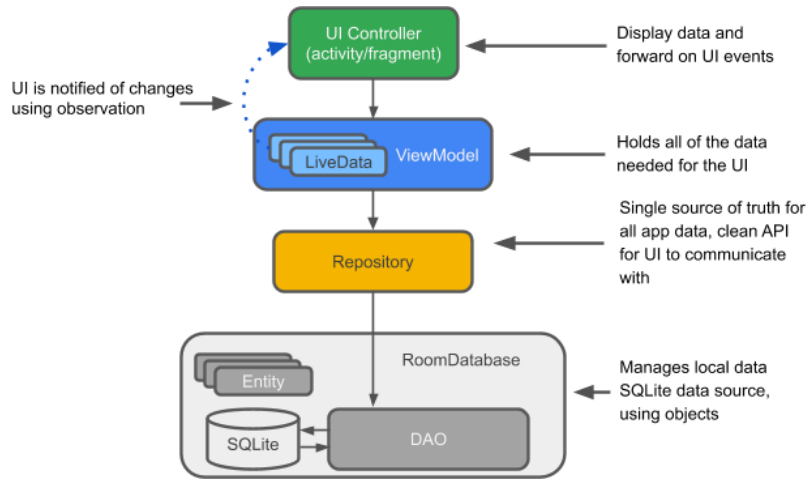
1. Android Studio 3.4.
2. Handphone Android versi 7.0 (Nougat)
3. Kabel data USB.
4. Driver ADB.

DASAR TEORI

https://codelabs.developers.google.com/codelabs/android-room-with-a-view-kotlin/#0

Komponen arsitektur membantu kita menyusun aplikasi dengan cara yang kuat, dapat diuji, dan dapat dipelihara dengan kode yang lebih sederhana. Saat ini kita akan berfokus pada subset komponen, yaitu LiveData, ViewModel, dan Room.

Berikut diagram yang menunjukkan bentuk dasar arsitektur:

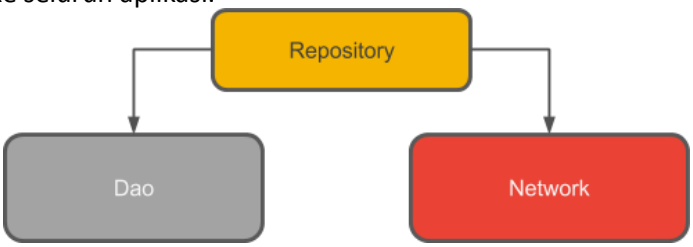


Entity: Kelas beranotasi yang menjelaskan tabel database saat bekerja dengan Room.
SQLite database:: Di penyimpanan perangkat. Room persistence library membuat dan mengelola database ini untuk kita.
DAO: Data access object. Pemetaan query SQL ke fungsi. Ketika kita menggunakan DAO, kita memanggil metode, dan Room mengurus sisanya.

Apa itu DAO? Di DAO (objek akses data), kita menentukan kueri SQL dan mengaitkannya dengan panggilan metode. Kompiler memeriksa SQL dan menghasilkan kueri dari anotasi kenyamanan untuk kueri umum, seperti @Insert. Room menggunakan DAO untuk membuat API bersih untuk kode kita.
DAO harus berupa antarmuka atau kelas abstrak. Secara default, semua kueri harus dijalankan pada thread terpisah. Room memiliki dukungan coroutines, memungkinkan pertanyaan kita dijelaskan dengan pengubah penangguhan dan kemudian dipanggil dari coroutine atau dari fungsi suspensi lain.

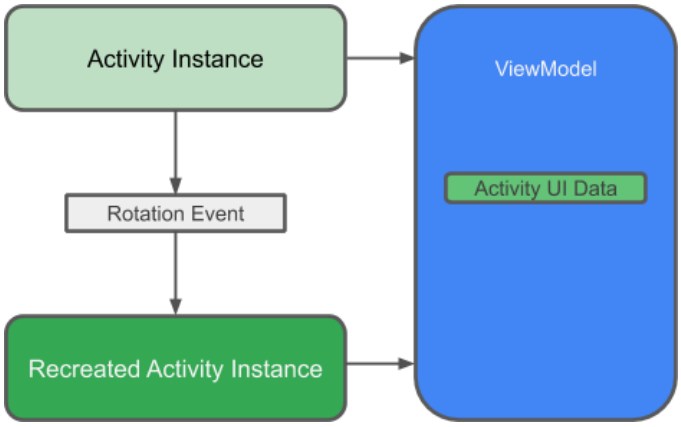
Room database : Menyederhanakan pekerjaan basis data dan berfungsi sebagai titik akses ke basis data SQLite yang mendasarinya (menyembunyikan SQLiteOpenHelper). Database Room menggunakan DAO untuk mengeluarkan pertanyaan ke database SQLite.
Apa itu database Room? Room adalah lapisan basis data di atas basis data SQLite. Room menangani tugas-tugas biasa yang kita gunakan untuk menangani dengan SQLiteOpenHelper. Room menggunakan DAO untuk mengeluarkan pertanyaan ke basis datanya. Secara default, untuk menghindari kinerja UI yang buruk, Room tidak memungkinkan kita untuk mengeluarkan pertanyaan pada thread utama. Ketika kueri Room mengembalikan LiveData, kueri secara otomatis dijalankan secara tidak sinkron pada background thread. Room menyediakan pemeriksaan waktu kompilasi terhadap pernyataan SQLite.

Repository: Kelas yang kita buat yang terutama digunakan untuk mengelola beberapa sumber data.
Apa itu Repository? Kelas repository mengabstraksi akses ke banyak sumber data. Repository bukan bagian dari library Komponen Arsitektur, tetapi merupakan praktik terbaik yang disarankan untuk pemisahan kode dan arsitektur. Kelas Repository menyediakan API bersih untuk akses data ke seluruh aplikasi.



Mengapa menggunakan Repositori? Repositori mengelola kueri dan memungkinkan kita untuk menggunakan beberapa backend. Dalam contoh paling umum, Repositori mengimplementasikan logika untuk memutuskan apakah akan mengambil data dari jaringan atau menggunakan hasil yang di-cache dalam database lokal.

ViewModel: Bertindak sebagai pusat komunikasi antara Repositori (data) dan UI. UI tidak perlu lagi khawatir tentang asal-usul data. Contoh ViewModel adalah Activity/Fragment.
Apa itu ViewModel? Peran ViewModel adalah untuk menyediakan data ke UI dan survive dari perubahan konfigurasi. ViewModel bertindak sebagai pusat komunikasi antara Repositori dan UI. Kita juga dapat menggunakan ViewModel untuk berbagi data antar fragmen. ViewModel adalah bagian dari lifecycle library.



Mengapa menggunakan ViewModel? ViewModel menyimpan data UI aplikasi kita dengan cara yang sadar siklus yang selamat dari perubahan konfigurasi. Memisahkan data UI aplikasi kita dari kelas Activity dan Fragmen, memungkinkan kita mengikuti prinsip tanggung jawab tunggal dengan lebih baik: Activity dan fragmen bertanggung jawab untuk menggambar data ke layar, sementara ViewModel dapat menangani memegang dan memproses semua data yang diperlukan untuk UI . Di ViewModel, gunakan LiveData untuk data yang dapat diubah yang akan digunakan atau ditampilkan oleh UI. Menggunakan LiveData memiliki beberapa manfaat:

- Kita dapat menempatkan pengamat pada data (bukan polling untuk perubahan) dan hanya memperbarui UI ketika data benar-benar berubah.
- Repositori dan UI sepenuhnya dipisahkan oleh ViewModel.
- Tidak ada panggilan database dari ViewModel (ini semua ditangani di Repositori), membuat kode lebih dapat diuji.

viewModelScope. Di Kotlin, semua coroutine dijalankan di dalam CoroutineScope. Lingkup mengontrol masa pakai coroutine melalui pekerjaannya. Ketika kita membatalkan pekerjaan lingkup, itu membatalkan semua coroutine dimulai dalam lingkup itu. Pustaka siklus hidup AndroidX-viewmodel-ktx menambahkan viewModelScope sebagai fungsi ekstensi dari kelas ViewModel, memungkinkan kita untuk bekerja dengan scope.

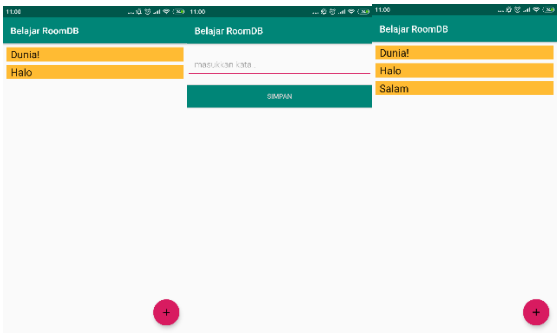
LiveData: Kelas pemegang data yang dapat diamati. Selalu memegang / menyimpan versi data terbaru, dan memberi tahu pengamatnya ketika data telah berubah. LiveData sadar akan siklus hidup. Komponen UI hanya mengamati data yang relevan dan jangan berhenti atau melanjutkan pengamatan. LiveData secara otomatis mengelola semua ini karena menyadari perubahan status siklus hidup yang relevan saat mengamati.
Saat data berubah, kita biasanya ingin mengambil tindakan, seperti menampilkan data yang diperbarui di UI. Ini berarti kita harus mengamati data sehingga ketika itu berubah, kita dapat bereaksi. Tergantung pada bagaimana data disimpan, ini bisa rumit. Mengamati perubahan pada data di berbagai komponen aplikasi kita dapat membuat jalur ketergantungan yang eksplisit dan kaku antar komponen. Ini membuat pengujian dan debugging menjadi sulit, antara lain. LiveData, lifecycle library class untuk observasi data, memecahkan masalah ini. Gunakan nilai kembalian tipe LiveData dalam deskripsi metode kita, dan Room menghasilkan semua kode yang diperlukan untuk memperbarui LiveData ketika database diperbarui.

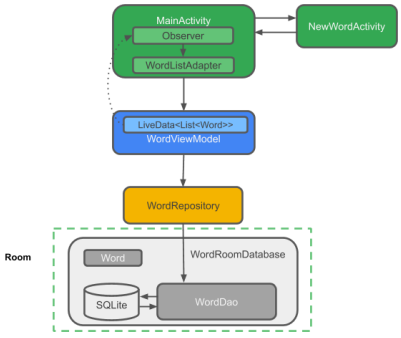
Beberapa penjelasan akan disampaikan di langkah praktik.

PRAKTIK

1. Kita akan membuat aplikasi sebagai berikut.

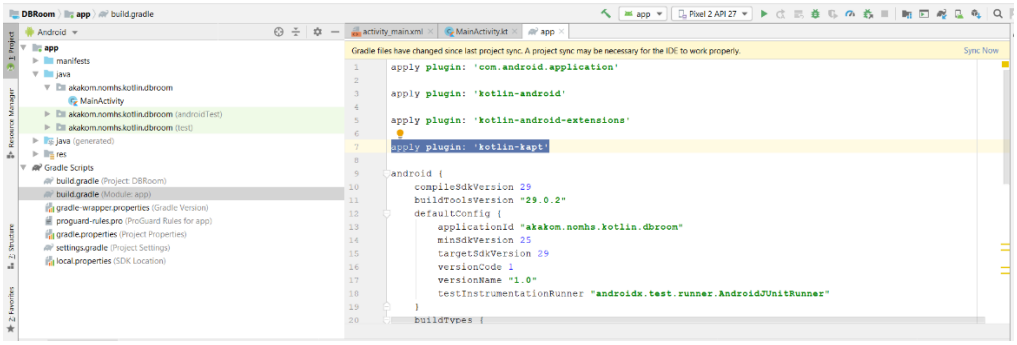
a. Bekerja dengan database untuk mendapatkan dan menyimpan data, dan pra-populasikan database dengan beberapa kata.
b. Menampilkan semua kata dalam RecyclerView di MainActivity.
c. Membuka aktivitas kedua saat pengguna menekan tombol +. Saat pengguna memasukkan kata, tambahkan kata ke database dan daftar.





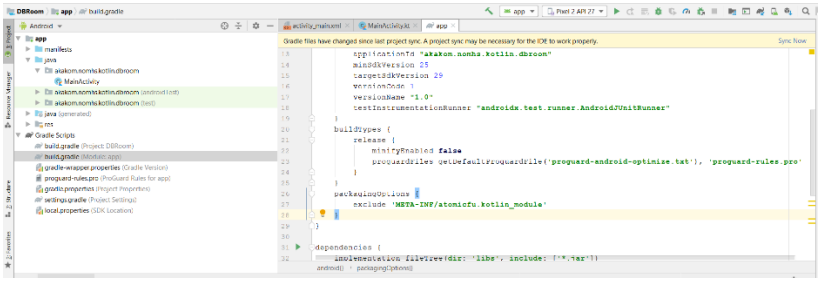
2. Buat sebuah project baru. Buat juga sebuah Empty Actitivy
3. Perbaharui Gradle, dengan langkah berikut.
   a. Di build.gradle (**Module: app**), tambahkan berikut

```
apply plugin: 'kotlin-kapt'
```



   b. Masih di build.gradle(Module:app), tambahkan blok packagingOptions di dalam blok android untuk mengecualikan *atomic functions module* dari package dan mencegah warnings.

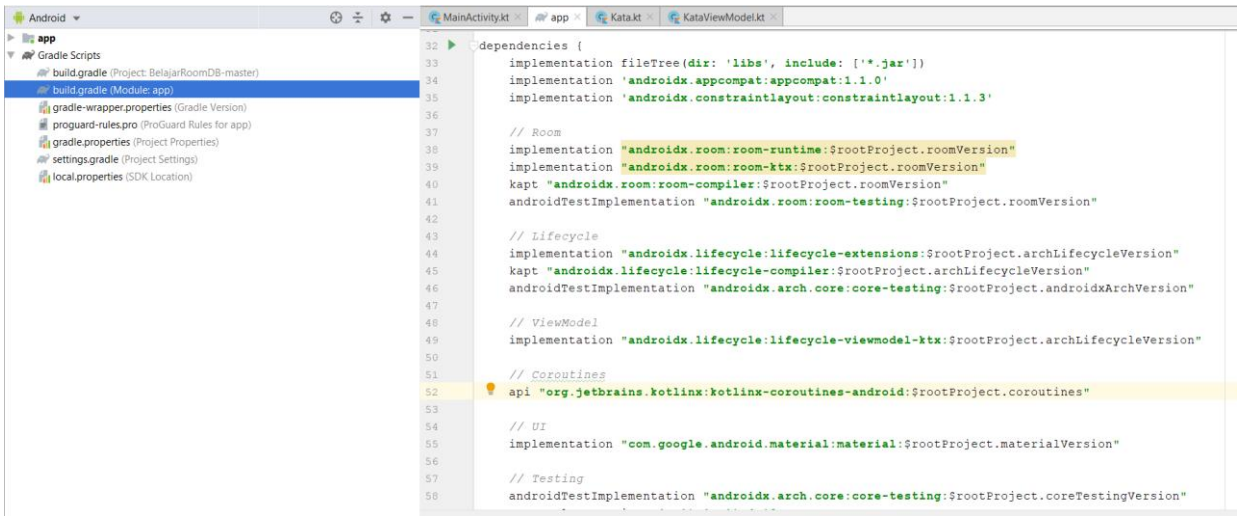
```
packagingOptions {
    exclude 'META-INF/atomicfu.kotlin_module'
}
```



   c. Tambahkan blok kode berikut

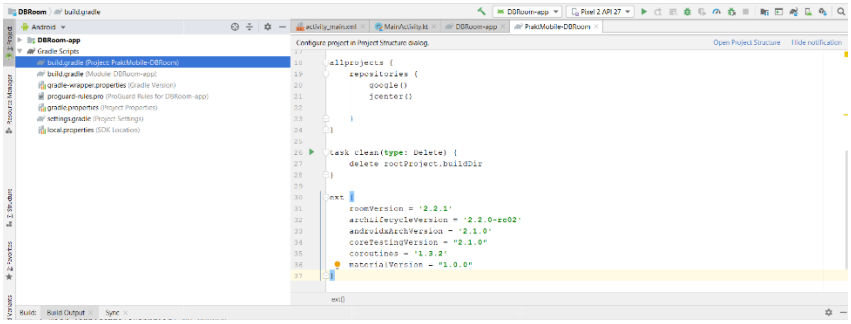
```
// Room
implementation "androidx.room:room-runtime:$rootProject.roomVersion"
implementation "androidx.room:room-ktx:$rootProject.roomVersion"
kapt "androidx.room:room-compiler:$rootProject.roomVersion"
androidTestImplementation "androidx.room:room-testing:$rootProject.roomVersion"

// Lifecycle
implementation "androidx.lifecycle:lifecycle-
```

```
extensions:$rootProject.archLifecycleVersion"
kapt "androidx.lifecycle:lifecycle-compiler:$rootProject.archLifecycleVersion"
androidTestImplementation "androidx.arch.core:core-
testing:$rootProject.androidxArchVersion"

// ViewModel
implementation "androidx.lifecycle:lifecycle-viewmodel-
ktx:$rootProject.archLifecycleVersion"

// Coroutines
api "org.jetbrains.kotlinx:kotlinx-coroutines-android:$rootProject.coroutines"

// UI
implementation
"com.google.android.material:material:$rootProject.materialVersion"

// Testing
androidTestImplementation "androidx.arch.core:core-
testing:$rootProject.coreTestingVersion"
```



4. Di file build.gradle (Project: DBRoom) tambahkan nomor versi ke akhir file, seperti yang diberikan dalam kode di bawah ini.

```
ext {
    roomVersion = '2.2.1'
    archLifecycleVersion = '2.2.0-rc02'
    androidxArchVersion = '2.1.0'
    coreTestingVersion = "2.1.0"
    coroutines = '1.3.2'
    materialVersion = "1.0.0"
}
```

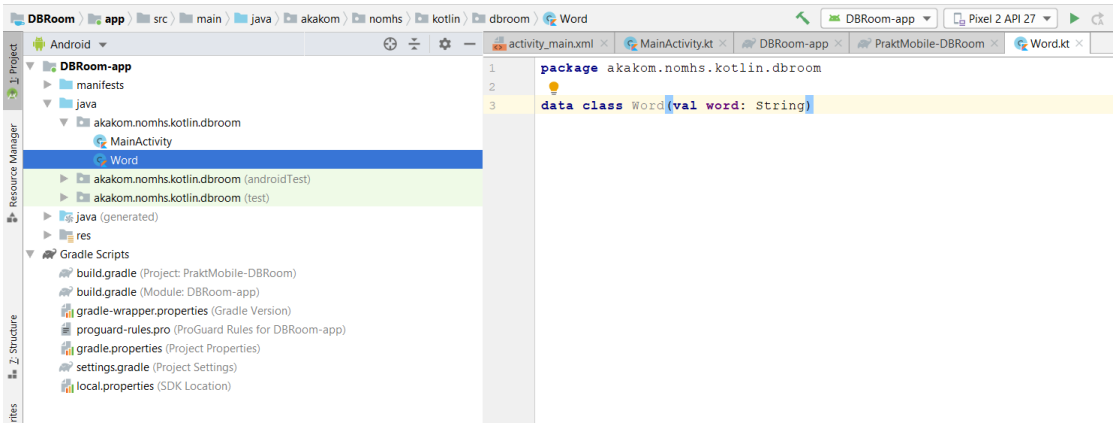


5. Kita akan membuat sebuat Entity sebagai berikut. Room memungkinkan kita membuat tabel melalui Entity.

word_table table
word (Primary Key, String)
"Hello"
"World"

6. Buat file kelas Kotlin baru bernama Word yang berisi kelas data Word. Kelas ini akan menjelaskan Entitas (yang mewakili tabel SQLite) untuk kata-kata yang dimasukkan. Setiap properti publik di kelas mewakili kolom dalam tabel. Room pada akhirnya akan menggunakan properti ini untuk membuat tabel dan instantiate objek dari baris dalam database.

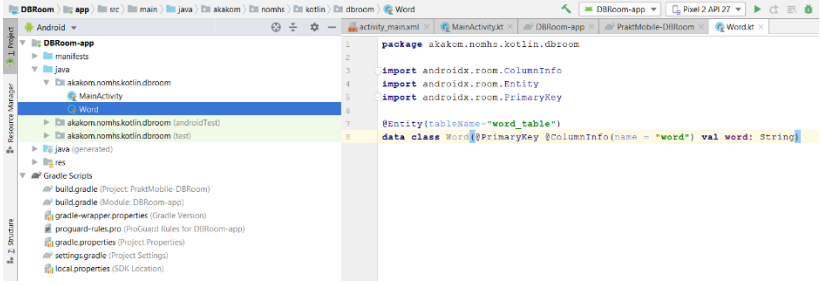
```
data class Word(val word: String)
```



7. Untuk membuat kelas Word bermakna bagi basis data Room, kita perlu memberi anotasi. Anotasi mengidentifikasi bagaimana setiap bagian dari kelas ini berhubungan dengan entri dalam database. Room menggunakan informasi ini untuk menghasilkan kode.

```
@Entity(tableName="word_table")
data class Word(@PrimaryKey @ColumnInfo(name = "word") val word: String)
```

   a. @Entity (tableName = "word_table"). Setiap kelas @Entity mewakili tabel SQLite. Buat anotasi deklarasi kelas Anda untuk menunjukkan bahwa itu adalah entitas. Kita bisa menentukan nama tabel jika kita ingin itu berbeda dari nama kelas. Tabel ini dinamai tabel "word_table".
   b. @PrimaryKey. Setiap entitas membutuhkan kunci utama. Agar semuanya sederhana, setiap kata bertindak sebagai kunci utama sendiri.
   c. @ColumnInfo (name = "word"). Tentukan nama kolom dalam tabel jika kita ingin itu berbeda dari nama variabel anggota. Kolom ini dinamai "word".
   d. Setiap properti yang disimpan dalam database harus memiliki visibilitas publik, yang merupakan default Kotlin.



8. Kita akan membuat file DAO untuk : Memesan semua kata sesuai abjad, Memasukkan sebuah kata, Menghapus semua kata. Buat file kelas Kotlin baru bernama WordDao. Salin

dan tempel kode berikut ke WordDao dan perbaiki impor yang diperlukan untuk membuatnya dikompilasi.

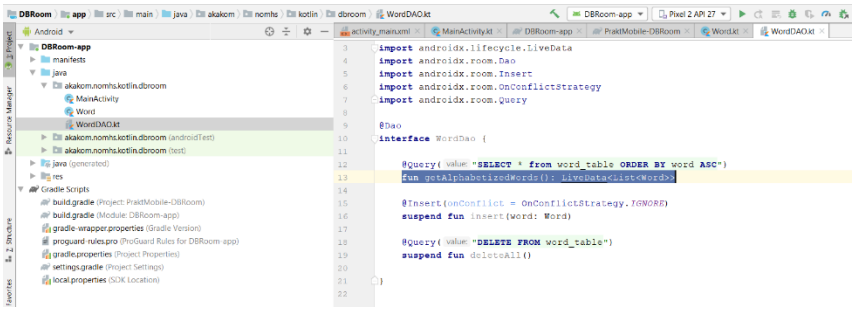
```
@Dao
interface WordDao {

    @Query("SELECT * from word_table ORDER BY word ASC")
    fun getAlphabetizedWords(): LiveData<List<Word>>

    @Insert(onConflict = OnConflictStrategy.IGNORE)
    suspend fun insert(word: Word)

    @Query("DELETE FROM word_table")
    suspend fun deleteAll()
}
```

a. WordDao adalah interface; DAO harus berupa interface atau kelas abstrak.
b. Anotasi @Dao mengidentifikasinya sebagai kelas DAO untuk Room.
c. suspend fun insert(word: Word): Menyatakan fungsi tunda untuk memasukkan satu kata.
d. Anotasi @Insert adalah anotasi metode DAO khusus di mana kita tidak harus menyediakan SQL apa pun!
e. onConflict = OnConflictStrategy.IGNORE: Strategi konflik yang dipilih mengabaikan kata baru jika itu persis sama dengan yang sudah ada dalam list.
f. suspend fun deleteAll (): Menyatakan fungsi tunda untuk menghapus semua kata.
g. Tidak ada penjelasan kenyamanan untuk menghapus banyak entitas, jadi ini dijelaskan dengan @Query umum.
h. @Query ("DELETE FROM word_table"): @Query mengharuskan untuk memberikan kueri SQL sebagai parameter string ke anotasi, memungkinkan kueri baca kompleks dan operasi lainnya.
i. fun getAlphabetizedWords (): List <Word>: Metode untuk mendapatkan semua kata dan mengembalikannya ke List dari Word.
j. @Query ("SELECT * from word_table ORDER BY word ASC"): Permintaan yang mengembalikan daftar kata yang diurutkan dalam urutan menaik.

9. Saat data berubah, kita biasanya ingin mengambil tindakan, seperti menampilkan data yang diperbarui di UI. Ini berarti kita harus mengamati data sehingga ketika itu berubah, kita dapat bereaksi. Tergantung pada bagaimana data disimpan, ini bisa rumit. Mengamati perubahan pada data di berbagai komponen aplikasi akan dapat membuat jalur ketergantungan yang eksplisit dan kaku antar komponen. Ini antara lain yang membuat pengujian dan debugging menjadi sulit. LiveData, lifecycle library class untuk observasi data, memecahkan masalah ini. Gunakan nilai kembalian tipe LiveData dalam deskripsi metode Anda, dan Room menghasilkan semua kode yang diperlukan untuk memperbarui LiveData ketika database diperbarui. Di WordDao, ubah deklarasi metode getAlphabetizedWords () sehingga List <Word> yang dikembalikan dibungkus dengan LiveData.

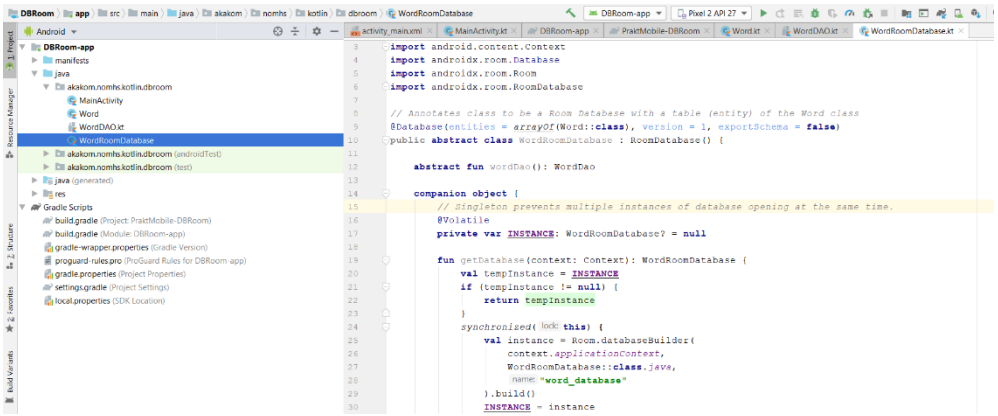
```
@Query("SELECT * from word_table ORDER BY word ASC")
fun getAlphabetizedWords(): LiveData<List<Word>>
```

Kemudian, kita melacak perubahan data melalui Observer di MainActivity.



10. Kelas database Room Anda harus abstrak dan extends RoomDatabase. Biasanya, Anda hanya perlu satu instance dari database Room untuk seluruh aplikasi. Buat file kelas Kotlin bernama **WordRoomDatabase** dan tambahkan kode ini ke dalamnya:

```
// Annotates class to be a Room Database with a table (entity) of the Word class
@Database(entities = arrayOf(Word::class), version = 1, exportSchema = false)
public abstract class WordRoomDatabase : RoomDatabase() {

    abstract fun wordDao(): WordDao

    companion object {// Singleton prevents multiple instances of
                      // database opening at the same time.
        @Volatile
        private var INSTANCE: WordRoomDatabase? = null

        fun getDatabase(context: Context): WordRoomDatabase {
            val tempInstance = INSTANCE
            if (tempInstance != null) {
                return tempInstance
            }
            synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    WordRoomDatabase::class.java,
                    "word_database"
                ).build()
                INSTANCE = instance
                return instance
            }
        }
    }
}
```



a. Kelas database untuk Room harus abstrak dan extends RoomDatabase
b. Kita memberikan keterangan kelas menjadi database Room dengan @ Database dan menggunakan parameter penjelasan untuk menyatakan entitas yang termasuk dalam database dan mengatur nomor versi. Setiap entitas terkait dengan tabel yang akan dibuat dalam database. Dalam aplikasi nyata, kita harus mempertimbangkan

pengaturan direktori untuk Room untuk digunakan untuk mengekspor skema sehingga kita dapat memeriksa skema saat ini ke sistem kontrol versi kita.

c. Kita membuat database menyediakan DAO dengan membuat method abstrak "getter" untuk setiap @Dao.

d. Kita telah mendefinisikan sebuah singleton, WordRoomDatabase, untuk mencegah agar beberapa instance database tidak dibuka secara bersamaan.

e. getDatabase mengembalikan singleton. Ini akan membuat database saat pertama kali diakses, menggunakan pembuat basis data Room untuk membuat objek RoomDatabase dalam konteks aplikasi dari kelas WordRoomDatabase dan menamakannya "word_database".

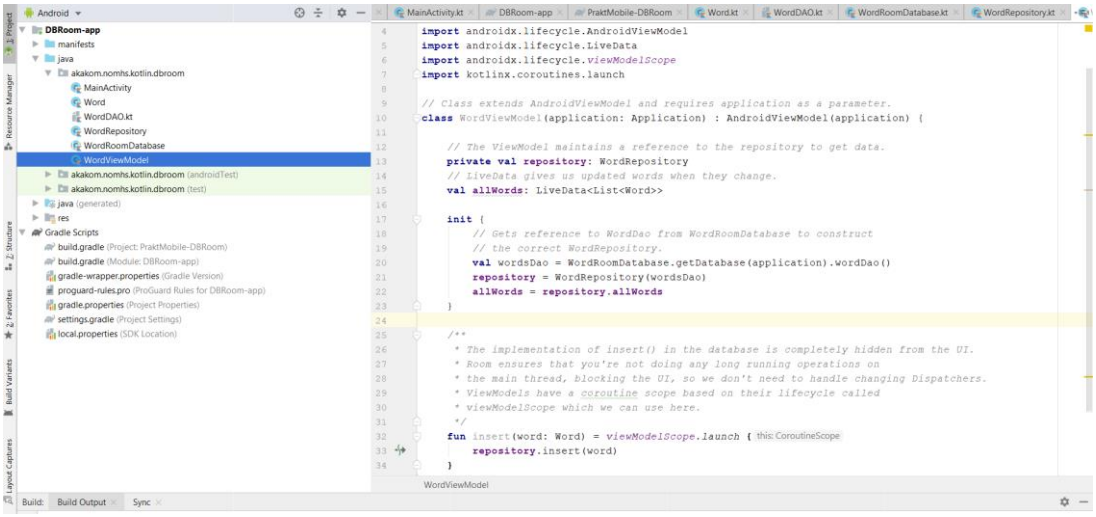
11. Selanjutnya, buat file kelas Kotlin bernama WordRepository dan tuliskan kode berikut ke dalamnya:

```
class WordRepository(private val wordDao: WordDao) {
  val allWords: LiveData<List<Word>>=wordDao.getAlphabetizedWords()
  suspend fun insert(word: Word) {
        wordDao.insert(word)
  }
}
```

DAO dilewatkan ke konstruktor repositori sebagai lawan dari keseluruhan database. Hal ini karena ini hanya membutuhkan akses ke DAO, karena DAO berisi semua method read/write untuk database. Tidak perlu mengekspos seluruh database ke repositori.
Daftar kata-kata adalah milik umum, yang diinisialisasi dengan mendapatkan daftar kata-kata LiveData dari Room; kita bisa melakukan ini karena cara kita mendefinisikan method getAlphabetizedWords untuk mengembalikan LiveData di langkah "LiveData class". Room mengeksekusi semua pertanyaan pada thread terpisah. Kemudian diamati LiveData akan memberi tahu pengamat di thread utama ketika data telah berubah. Pengubah suspend memberi tahu kompiler bahwa ini perlu dipanggil dari coroutine atau fungsi penangguhan lain.

12. Buat sebuah file Kotlin class untuk WordViewModel dan tuliskan kode berikut:

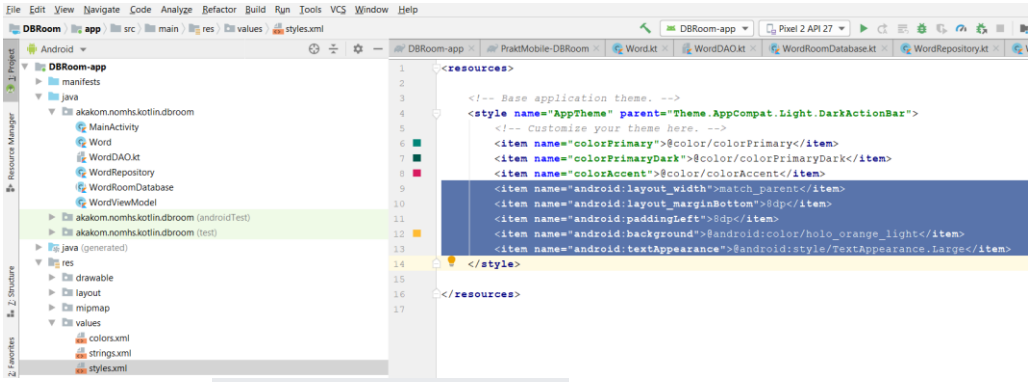
```
// Class extends AndroidViewModel and requires application as a parameter.
class WordViewModel(application: Application) : AndroidViewModel(application) {
    private val repository: WordRepository
    val allWords: LiveData<List<Word>>

    init {
        val wordsDao = WordRoomDatabase.getDatabase(application,
                                viewModelScope).wordDao()
        repository = WordRepository(wordsDao)
        allWords = repository.allWords
    }

    fun insert(word: Word) = viewModelScope.launch {
        repository.insert(word)
    }
}
```



a. Membuat kelas yang disebut WordViewModel yang mendapatkan Application sebagai parameter dan extends AndroidViewModel.
b. Menambahkan variabel private member untuk menyimpan referensi ke repositori.
c. Menambahkan variabel anggota LiveData public ke cache daftar kata.
d. Membuat blok init yang mendapatkan referensi ke WordDao dari WordRoomDatabase.
e. Di blok init, dibangun WordRepository berdasarkan WordRoomDatabase.
f. Di blok init, menginisialisasi LiveData allWords menggunakan repositori.
g. Membuat metode insert() yang memanggil metode insert () Repositori. Dengan cara ini, implementasi insert () dienkapsulasi dari UI. Kita tidak ingin memasukkan untuk memblokir thread utama, jadi kami meluncurkan coroutine baru dan memanggil sisipan repositori, yang merupakan fungsi suspend. Seperti yang disebutkan, ViewModels memiliki ruang lingkup coroutine berdasarkan siklus hidup mereka yang disebut viewModelScope, yang kita gunakan di sini.

13. Tambahkan style untuk daftar item dalam values/styles.xml:

```
<!-- Base application theme. -->
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
</style>
```

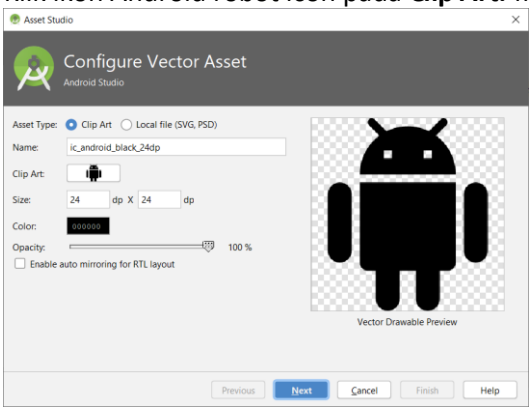


14. Tambahkan layout layout/recyclerview_item.xml :

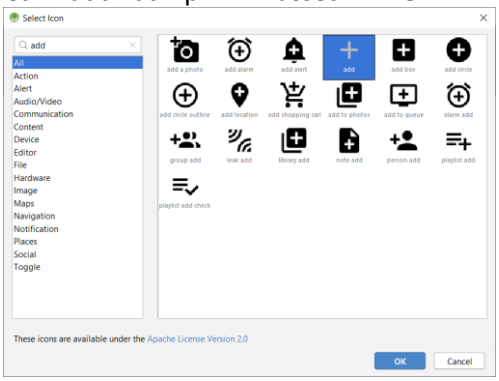
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <TextView
        android:id="@+id/textView"
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@android:color/holo_orange_light" />
</LinearLayout>
```

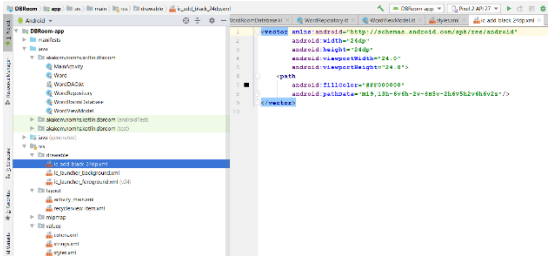
15. Tampilan FAB harus sesuai dengan tindakan yang tersedia, jadi kami ingin mengganti ikon dengan simbol '+'. Kita perlu menambahkan Vector Aset baru.
    a. Pilih **File > New > Vector Asset**.
    b. Klik ikon Android robot icon pada **Clip Art:** field.



    c. Cari "add" dan pilih '+' asset. Klik **OK**.



    d. Konfirmasi path ikon sebagai main > drawable dan klik **Finish** untuk menambahkan asset.



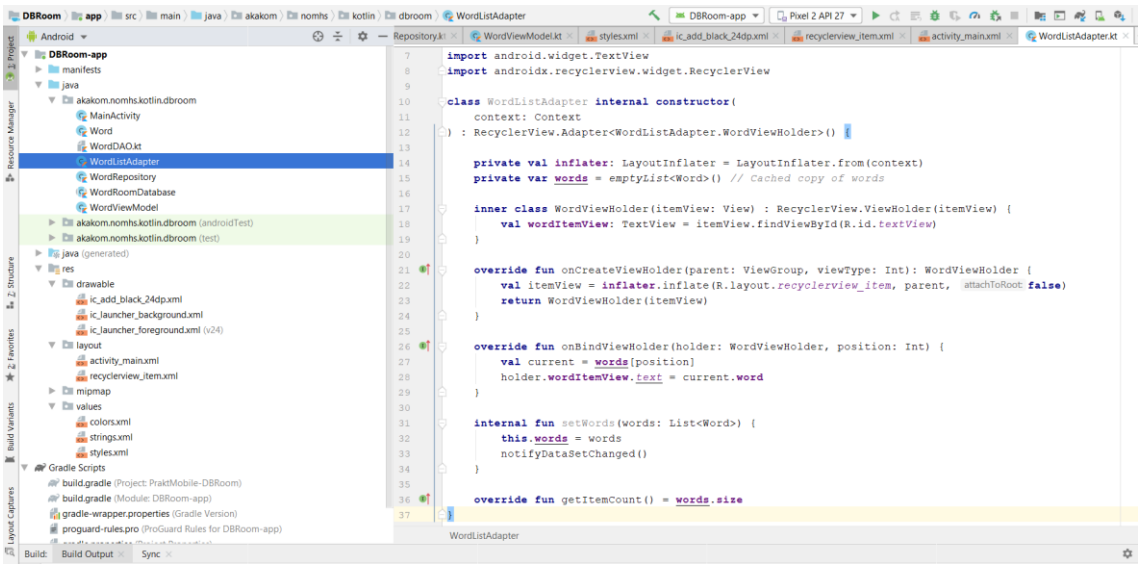
16. Di layout/activity_main.xml, update FAB untuk menambahkan drawable baru.

```
<com.google.android.material.floatingactionbutton.FloatingActionButton
    android:id="@+id/fab"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:src="@drawable/ic_add_black_24dp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="16dp" />
```

17. Kita akan menampilkan data dalam RecyclerView, yang sedikit lebih bagus daripada hanya membuang data dalam TextView. Codelab ini berasumsi bahwa kita tahu cara kerja RecyclerView, RecyclerView.Layout, RecyclerView.ViewHolder, dan RecyclerView.Adapter. Perhatikan bahwa variabel kata-kata dalam adaptor menyimpan data.

18. Buat file kelas Kotlin untuk WordListAdapter yang memperluas RecyclerView.Adapter. Berikut kodenya.

```
class WordListAdapter internal constructor(context: Context) :
RecyclerView.Adapter<WordListAdapter.WordViewHolder>() {

    private val inflater: LayoutInflater =
                            LayoutInflater.from(context)
    private var words = emptyList<Word>() // Cached copy of words

    inner class WordViewHolder(itemView: View) :
                RecyclerView.ViewHolder(itemView) {
        val wordItemView: TextView =
                itemView.findViewById(R.id.textView)
    }

    override fun onCreateViewHolder(parent: ViewGroup,
                viewType: Int): WordViewHolder {
        val itemView = inflater.inflate(R.layout.recyclerview_item,
                parent, false)
        return WordViewHolder(itemView)
    }

    override fun onBindViewHolder(holder: WordViewHolder,
                position: Int) {
        val current = words[position]
        holder.wordItemView.text = current.word
    }

    internal fun setWords(words: List<Word>) {
        this.words = words
        notifyDataSetChanged()
    }

    override fun getItemCount() = words.size
}
```

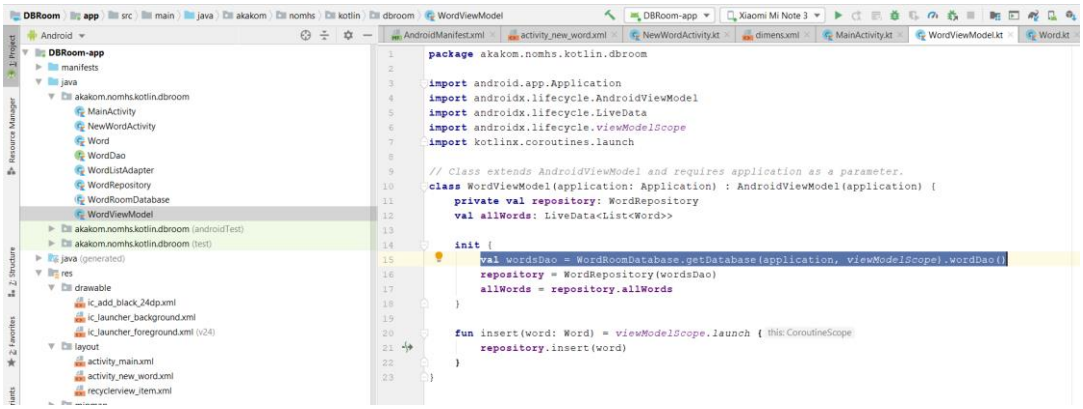


19. Tambahkan RecyclerView dalam method onCreate() dari MainActivity. Dalam method onCreate() sesudah setContentView:

```
val recyclerView = findViewById<RecyclerView>(R.id.recyclerview)
val adapter = WordListAdapter(this)
recyclerView.adapter = adapter
recyclerView.layoutManager = LinearLayoutManager(this)
```

20. Jalankan aplikasi untuk memastikan semuanya berfungsi. Tidak ada item, karena kita belum menghubungkan data, jadi aplikasi harus menampilkan latar belakang abu-abu tanpa item daftar.
21. Untuk meluncurkan coroutine kita membutuhkan CoroutineScope. Perbarui metode getDatabase dari kelas WordRoomDatabase, untuk juga mendapatkan lingkup coroutine sebagai parameter. Perbarui penginisialisasi pengambilan basis data di blok init dari WordViewModel untuk juga melewati ruang lingkup.

```
val wordsDao = WordRoomDatabase.getDatabase(application, viewModelScope).wordDao()
```



22. Di WordRoomDatabase, kita membuat implementasi kustom dari RoomDatabase. Callback(), yang juga mendapatkan CoroutineScope sebagai parameter konstruktor. Kemudian, mengganti metode onOpen untuk mengisi basis data. Berikut adalah kode untuk membuat panggilan balik di dalam kelas WordRoomDatabase:

```
private class WordDatabaseCallback(
    private val scope: CoroutineScope
) : RoomDatabase.Callback() {

    override fun onOpen(db: SupportSQLiteDatabase) {
        super.onOpen(db)
        INSTANCE?.let { database ->
            scope.launch {
                populateDatabase(database.wordDao())
            }
        }
    }

    suspend fun populateDatabase(wordDao: WordDao) {
        // Delete all content here.
        wordDao.deleteAll()

        // Add sample words.
        var word = Word("Hello")
        wordDao.insert(word)
        word = Word("World!")
        wordDao.insert(word)

        // TODO: Add your own words!
    }
}
```

23. Terakhir, tambahkan callback ke urutan pembuatan basis data tepat sebelum memanggil .build () di Room.databaseBuilder ().

```
.addCallback(WordDatabaseCallback(scope))
```

Kode lengkapnya menjadi sebagai berikut.

```
@Database(entities = arrayOf(Word::class), version = 1,
exportSchema = false)
public abstract class WordRoomDatabase : RoomDatabase() {

    abstract fun wordDao(): WordDao

    private class WordDatabaseCallback(
        private val scope: CoroutineScope
    ) : RoomDatabase.Callback() {

        override fun onOpen(db: SupportSQLiteDatabase) {
            super.onOpen(db)
            INSTANCE?.let { database ->
                scope.launch {
                    populateDatabase(database.wordDao())
                }
            }
        }

        suspend fun populateDatabase(wordDao: WordDao) {
            // Delete all content here.
            wordDao.deleteAll()

            // Add sample words.
            var word = Word("Hello")
            wordDao.insert(word)
            word = Word("World!")
            wordDao.insert(word)

            // TODO: Add your own words!
        }
    }

    companion object {
        @Volatile
        private var INSTANCE: WordRoomDatabase? = null

        fun getDatabase(
            context: Context,
            scope: CoroutineScope
        ): WordRoomDatabase {
            // if the INSTANCE is not null, then return it,
            // if it is, then create the database
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    WordRoomDatabase::class.java,
                    "word_database"
                )
                    .addCallback(WordDatabaseCallback(scope))
                    .build()
                INSTANCE = instance
                // return instance
                instance
            }
        }
    }
}
```

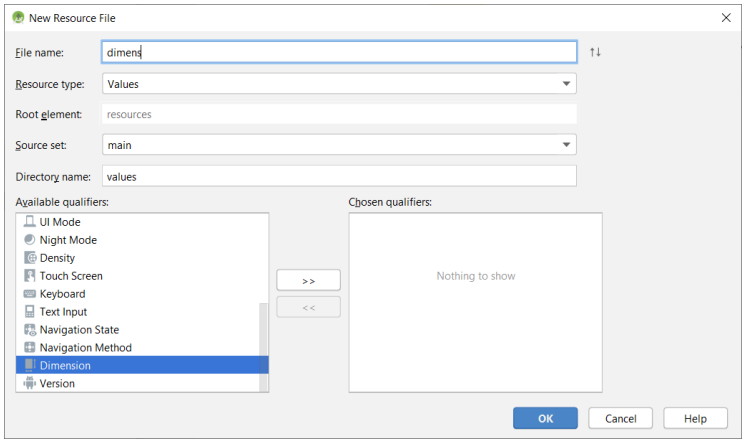
24. Tambahkan berikut pada string resources values/strings.xml

```
<string name="hint_word">Kata...</string>
<string name="button_save">Simpan</string>
<string name="empty_not_saved">Kata tidak dapat disimpan karena
kosong.</string>
```

25. Tambahkan pada color resource value/colors.xml

```
<color name="buttonLabel">#d3d3d3</color>
```

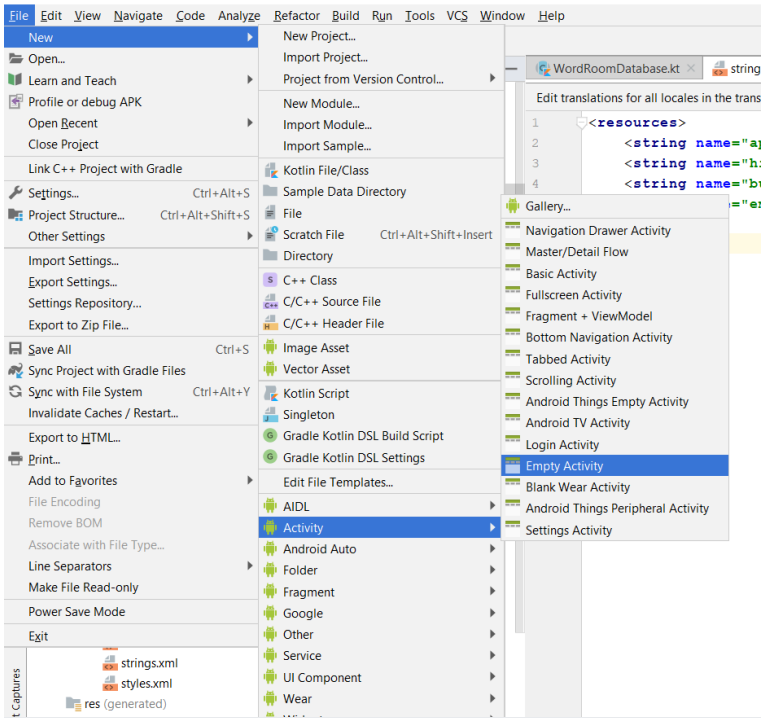
26. Buat sebuah dimension resource file:
    a. Klik app module dalam **Project** window.
    b. Pilih **File > New > Android Resource File**
    c. Dari Available Qualifiers, pilih **Dimension**
    d. Beri nama file: dimens



27. Tambahkan pada dimension resources dalam values/dimens.xml:

```
<dimen name="small_padding">6dp</dimen>
<dimen name="big_padding">16dp</dimen>
```

28. Buat sebuah empty Android Activity baru dengan template Empty Activity
    a. Pilih File > New > Activity > Empty Activity
    b. Tuliskan NewWordActivity untuk nama Activity.



29. Verifikasi bahwa activity baru telah ditambahkan ke Android Manifest!

```
<activity android:name=".NewWordActivity"></activity>
```

30. Update file activity_new_word.xml dalam folder layout folder dengan code berikut

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

    <EditText
        android:id="@+id/edit_word"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:fontFamily="sans-serif-light"
        android:hint="@string/hint_word"
        android:inputType="textAutoComplete"
        android:padding="@dimen/small_padding"
        android:layout_marginBottom="@dimen/big_padding"
        android:layout_marginTop="@dimen/big_padding"
        android:textSize="18sp" />

    <Button
        android:id="@+id/button_save"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/colorPrimary"
        android:text="@string/button_save"
        android:textColor="@color/buttonLabel" />

</LinearLayout>
```

31. Update kode untuk activity NewWordActivity

```
class NewWordActivity : AppCompatActivity() {

    private lateinit var editWordView: EditText

    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_new_word)
        editWordView = findViewById(R.id.edit_word)

        val button = findViewById<Button>(R.id.button_save)
        button.setOnClickListener {
            val replyIntent = Intent()
            if (TextUtils.isEmpty(editWordView.text)) {
                setResult(Activity.RESULT_CANCELED, replyIntent)
            } else {
                val word = editWordView.text.toString()
                replyIntent.putExtra(EXTRA_REPLY, word)
                setResult(Activity.RESULT_OK, replyIntent)
            }
            finish()
        }
    }

    companion object {
        const val EXTRA_REPLY =
                    "com.example.android.wordlistsql.REPLY"
    }
}
```

32. Kemudian kita akan menyambungkan UI ke basis data dengan menyimpan kata-kata baru yang dimasukkan pengguna dan menampilkan konten saat ini dari basis data kata di RecyclerView. Untuk menampilkan konten saat ini dari database, tambahkan pengamat yang mengamati LiveData di ViewModel. Setiap kali data berubah, panggilan balik onChanged () dipanggil, yang memanggil metode setWord () adaptor untuk memperbarui data cache adaptor dan refresh daftar yang ditampilkan. Di MainActivity, buat variabel anggota untuk ViewModel:

```
private lateinit var wordViewModel: WordViewModel
```

33. Gunakan ViewModelProviders untuk mengaitkan ViewModel dengan Activity. Saat Activity pertama kali dimulai, ViewModelProviders akan membuat ViewModel. Ketika activity dihancurkan, misalnya melalui perubahan konfigurasi, ViewModel tetap ada. Ketika activity dibuat kembali, ViewModelProviders mengembalikan ViewModel yang ada. Di onCreate () di bawah blok kode RecyclerView, dapatkan ViewModel dari ViewModelProvider.

```
wordViewModel = ViewModelProvider(this).
                    get(WordViewModel::class.java)
```

34. Juga di onCreate (), tambahkan observer untuk properti LiveData allWords dari WordViewModel. Metode onChanged () fresh ketika data yang diamati berubah dan aktivity berada di latar depan.

```
wordViewModel.allWords.observe(this, Observer { words ->
    // Update the cached copy of the words in the adapter.
```

```
    words?.let { adapter.setWords(it) }
})
```

35. Kita ingin membuka NewWordActivity saat menekan FAB dan, setelah kembali ke MainActivity, untuk memasukkan kata baru ke dalam basis data atau menampilkan Toast, mulai dengan mendefinisikan kode permintaan:

```
private val newWordActivityRequestCode = 1
```

36. Di MainActivity, tambahkan kode onActivityResult () untuk NewWordActivity. Jika activity kembali dengan RESULT_OK, masukkan kata yang dikembalikan ke database dengan memanggil metode insert () dari WordViewModel.

```
override fun onActivityResult(requestCode: Int, resultCode: Int,
data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)

    if (requestCode == newWordActivityRequestCode && resultCode
                    == Activity.RESULT_OK) {
        data?.getStringExtra(NewWordActivity.EXTRA_REPLY)?.let {
            val word = Word(it)
            wordViewModel.insert(word)
        }
    } else {
        Toast.makeText(
            applicationContext,
            R.string.empty_not_saved,
            Toast.LENGTH_LONG).show()
    }
}
```

37. Di MainActivity, mulai NewWordActivity ketika pengguna menekan FAB. Di MainActivity onCreate, cari FAB dan tambahkan onClickListener dengan kode ini:

```
val fab = findViewById<FloatingActionButton>(R.id.fab)
fab.setOnClickListener {
    val intent = Intent(this@MainActivity,
            NewWordActivity::class.java)
    startActivityForResult(intent, newWordActivityRequestCode)
}
```

38. Jadi kode akhir dari MainActivity adalah sebagai berikut:

```
class MainActivity : AppCompatActivity() {
    private lateinit var wordViewModel: WordViewModel
    private val newWordActivityRequestCode = 1
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val recyclerView =
                    findViewById<RecyclerView>(R.id.recyclerview)
        val adapter = WordListAdapter(this)
        recyclerView.adapter = adapter
        recyclerView.layoutManager = LinearLayoutManager(this)
        wordViewModel = ViewModelProvider(this).
                        get(WordViewModel::class.java)
        wordViewModel.allWords.observe(this, Observer { words ->
         // Update the cached copy of the words in the adapter.
            words?.let { adapter.setWords(it) }
```

```
        })
        val fab = findViewById<FloatingActionButton>(R.id.fab)
        fab.setOnClickListener {
            val intent = Intent(this@MainActivity,
                                NewWordActivity::class.java)
            startActivityForResult(intent,
                                newWordActivityRequestCode)
        }
    }

    override fun onActivityResult(requestCode: Int,
                    resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)

        if (requestCode == newWordActivityRequestCode &&
            resultCode == Activity.RESULT_OK) {
          data?.getStringExtra(NewWordActivity.EXTRA_REPLY)?.let
            {
                val word = Word(it)
                wordViewModel.insert(word)
            }
        } else {
            Toast.makeText(
                applicationContext,
                R.string.empty_not_saved,
                Toast.LENGTH_LONG).show()
        }
    }
}
```

39. Jalankan aplikasi dan amati hasilnya

LATIHAN

1. Modifikasilah aplikasi .

TUGAS

1. Buat aplikasi baru dengan mengembangkan project diatas

REFERENSI

1. https://kotlinlang.org/docs/reference/
2. https://developer.android.com/kotlin
3. https://developer.android.com/courses/kotlin-android-fundamentals/toc
4. https://codelabs.developers.google.com/android-kotlin-fundamentals/

5. https://developer.android.com/kotlin/learn
6. https://developer.android.com/kotlin/resources