LAPORAN RESMI

MODUL IV

FRAGMENT DAN ROOM DATABASE

PEMROGRAMAN BERGERAK

NAMA	**: SEPTIYA YUTANTRI**
N.R.P	**: 200441100023**
DOSEn	**: ACHMAD DAFID, S.T., M.T**
ASISTEN	**: MUHAMMAD YAFIE ANWARY RAHMAN**
TGL PRAKTIKUM	**: 07 APRIL 2023**

Disetujui : 29 Mei 2023
Asisten

M. YAFIE ANWARY RAHMAN
190441100052

LABORATORIUM BISNIS INTELIJEN SISTEM
PRODI SISTEM INFORMASI
FAKULTAS TEKNIK
UNIVERSITAS TRUNOJOYO MADURA

BAB I
PENDAHULUAN

1.1 Latar Belakang

Di era sekarang perkembangan teknologi sudah meningkat, manusia sudah tidak bisa dipisahkan dengan yang namanya teknologi. Untuk saat ini teknologi menjadi kebutuhan banyak orang. Dunia IT berkembang begitu pesat karena ditunjang dengan adanya perkembangan teknologi yang semakin canggih dan modern, pemanfaatan mobile learning yang dirasa perlu untuk menunjang proses belajar mengajar mahasiswa di pembelajaran Pemrograman Objek dan Perangkat Bergerak. Mobile Learning berbasis android dipilih mengingat hampir seluruh siswa di kelas memiliki handphone. Diharapkan dengan adanya mobile learning mampu meningkatkan semangat belajar karena didukung oleh teknologi yang bagus.

Pemebelajaran praktikum ini mempelajari tentang Fragment dan Room Database dimana kedua komponen tersebut merupakan dasar untuk membuat aplikasi yang mengintegrasikan beberapa komponen dan menjadi sebuah aplikasi yang fungsional dan proses migrasi database jadi lebih mudah. Fragment merupakan bagian UI aplikasi yang dapat digunakan kembali. Sebuah fragmen menentukan dan mengelola tata letaknya sendiri, memiliki siklus proses sendiri, serta dapat menangani peristiwa inputnya sendiri. Fragmen tidak dapat berjalan sendiri. Fragmen harus dihosting oleh aktivitas atau fragmen lain. Hierarki tampilan fragmen menjadi bagian dari, atau dilampirkan ke, hierarki tampilan host. Membagi UI menjadi beberapa fragmen akan memudahkan dalam mengubah tampilan aktivitas saat runtime. Sedangkan untuk Room Database merupakan library bagian dari Android Jetpack yang dapat meningkatkan produktivitas dalam pengembangan aplikasi Android. Room adalah library yang dapat membuat akses database lebih mudah, namun tetap menggunakan kekuatan penuh dari SQLite. Penjelasan singkat tersebut harus dipahami dengan baik, sesuai arahan saat mengikuti praktikum.

Prasyarat untuk dapat mengikuti praktikum ini dengan baik adalah memiliki pengetahuan dalam Bahasa pemrograman berorientasi objek penuh seperti java, C++, dan juga Kotlin. Untuk pengembangan, disarankan menggunakan sumber

referensi selain modul praktikum ini, sehingga kitab bisa belajar lebih mendalam tentang materi yang terdapat di modul dari sumber lain.

1.2 Tujuan

- Mahasiswa dapat membuat aplikasi dengan menggunakan fragment.
- Mahasiswa dapat membuat Aplikasi dengan database.

BAB II
DASAR TEORI

2.1 Fragment

➢ Fragment

Fragment mewakili perilaku atau bagian dari antarmuka pengguna dalam FragmentActivity. Kita bisa mengombinasikan beberapa fragmen dalam satu aktivitas untuk membangun UI multipanel dan menggunakan kembali sebuah fragmen dalam beberapa aktivitas. Kita bisa menganggap fragmen sebagai bagian modular dari aktivitas, yang memiliki daur hidup sendiri, menerima kejadian masukan sendiri, dan yang bisa kita tambahkan atau hapus saat aktivitas berjalan (semacam "subaktivitas" yang bisa digunakan kembali dalam aktivitas berbeda).

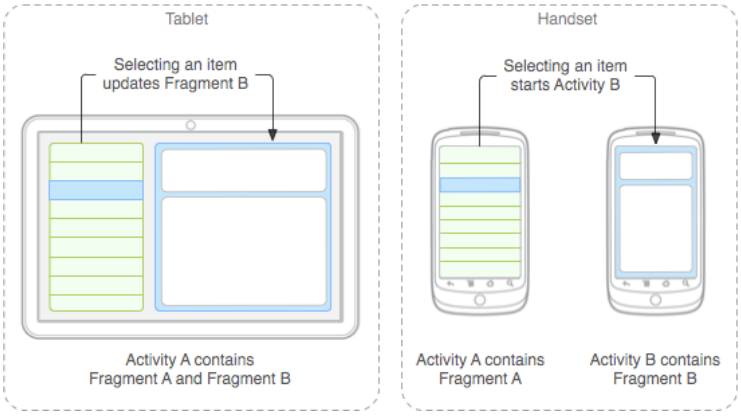
Fragmen harus selalu tersemat dalam aktivitas dan daur hidup fragmen secara langsung dipengaruhi oleh daur hidup aktivitas host-nya. Misalnya, saat aktivitas dihentikan sementara, semua fragmen di dalamnya juga dihentikan sementara, dan bila aktivitas dimusnahkan, semua fragmen juga demikian. Akan tetapi, saat aktivitas berjalan (dalam status daur hidup dilanjutkan), Kita bisa memanipulasi setiap fragmen secara terpisah, seperti menambah atau membuangnya. Saat melakukan transaksi fragmen, Kita juga bisa menambahkannya ke back-stack yang dikelola oleh aktivitas—setiap entri backstack merupakan catatan transaksi fragmen yang terjadi. Dengan back-stack pengguna dapat membalikkan transaksi fragmen (mengarah mundur), dengan menekan tombol Kembali.

Bila kita menambahkan fragmen sebagai bagian dari layout aktivitas, fragmen tersebut berada di ViewGroup di dalam hierarki tampilan aktivitas dan fragmen menentukan layout tampilannya sendiri. Kita bisa menyisipkan fragmen ke dalam layout aktivitas dengan mendeklarasikan fragmen dalam file layout aktivitas, sebagai<fragment>, atau dari kode aplikasi kita elemen menambahkannya ke ViewGroup yang ada.

➢ Filosofi Desain

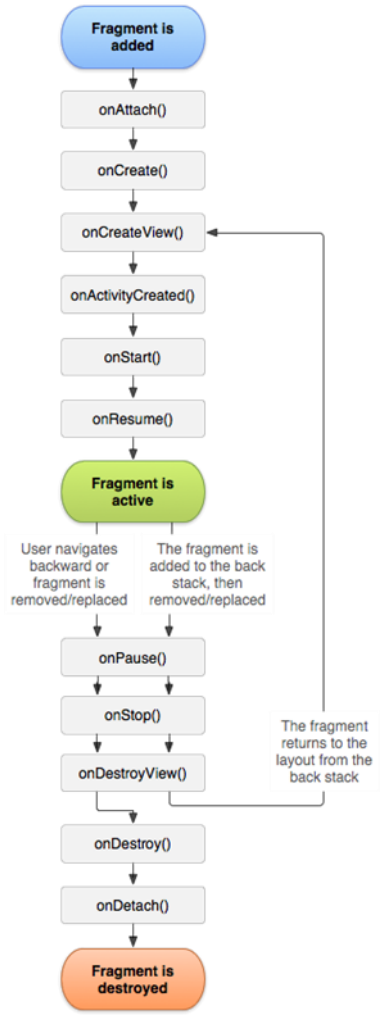
Android memperkenalkan fragmen di Android 3.0 (API level 11), terutama untuk mendukung desain UI yang lebih dinamis dan fleksibel pada layar besar, seperti tablet. Karena layar tablet jauh lebih besar daripada layar handset, maka lebih banyak ruang untuk mengombinasikan dan bertukar komponen UI. Fragmen

memungkinkan desain seperti itu tanpa perlu mengelola perubahan kompleks pada hierarki tampilan. Dengan membagi layout aktivitas menjadi beberapa fragmen, kita bisa mengubah penampilan aktivitas saat waktu proses dan mempertahankan perubahan itu di back-stack yang dikelola oleh aktivitas. Mode-mode tersebut kini tersedia secara luas melalui library dukungan fragmen. Misalnya, aplikasi berita bisa menggunakan satu fragmen untuk menampilkan daftar artikel di sebelah kiri dan fragmen lainnya untuk menampilkan artikel di sebelah kanan—kedua fragmen ini muncul di satu aktivitas, berdampingan, dan masing-masing fragmen memiliki serangkaian metode callback daur hidup dan menangani kejadian masukan penggunanya sendiri. Sehingga, sebagai ganti menggunakan satu aktivitas untuk memilih artikel dan aktivitas lainnya untuk membaca artikel, pengguna bisa memilih artikel dan membaca semuanya dalam aktivitas yang sama, sebagaimana diilustrasikan dalam layout tablet pada gambar 1.



Gambar 1. Dua modul UI yang ditentukan oleh fragmen bisa digabungkan ke dalam satu aktivitas untuk d esain tablet, namun dipisahkan untuk desain handset.

Misalnya—untuk melanjutkan contoh aplikasi berita —aplikasi bisa menyematkan dua fragmen dalam Aktivitas A, saat berjalan pada perangkat berukuran tablet. Akan tetapi, pada layar berukuran handset, ruang untuk kedua fragmen tidak sehingga Aktivitas A hanya menyertakan fragmen untuk daftar artikel, dan saat pengguna memilih artikel, Aktivitas B akan dimulai, termasuk fragmen kedua untuk membaca artikel. Sehingga, aplikasi mendukung tablet dan handset dengan menggunakan kembali fragmen dalam kombinasi berbeda, seperti diilustrasikan dalam gambar 1.



Gambar 2. Daur hidup fragmen (saat aktivitasnya berjalan).

Kita harus mendesain masing-masing fragmen sebagai komponen aktivitas modular dan bisa digunakan kembali. Yakni, karena setiap fragmen mendefinisikan layoutnya dan perilakunya dengan callback daur hidupnya sendiri, kita bisa memasukkan satu fragmen dalam banyak aktivitas, sehingga kita harus mendesainnya untuk digunakan kembali dan mencegah memanipulasi satu fragmen dari fragmen lain secara langsung. Ini terutama penting karena dengan fragmen modular kita bisa mengubah kombinasi fragmen untuk ukuran layar yang berbeda. Saat mendesain aplikasi untuk mendukung tablet maupun handset, kita bisa menggunakan kembali fragmen dalam konfigurasi layout yang berbeda untuk mengoptimalkan pengalaman pengguna berdasarkan ruang layar yang tersedia. Misalnya, pada handset, fragmen mungkin perlu dipisahkan untuk menyediakan UI panel tunggal bila lebih dari satu yang tidak cocok dalam aktivitas yang sama. cukup Membuat Fragmen.

- **Menambahkan antarmuka pengguna**

Fragmen biasanya digunakan sebagai bagian dari antarmuka pengguna aktivitas dan menyumbangkan layoutnya sendiri ke aktivitas. Untuk menyediakan layout fragmen, kita harus mengimplementasikan metode callback onCreateView(), yang dipanggil sistem Android bila tiba saatnya fragmen menggambar layoutnya. Implementasi kita atas metode ini harus mengembalikan View yang menjadi root layout fragmen. Untuk mengembalikan layout dari onCreateView(), Kita bisa memekarkannya dari resource layout yang ditentukan di XML. Untuk membantu melakukannya, onCreateView() menyediakan objek LayoutInflater.

- **Menambahkan fragmen ke aktivitas**

Biasanya, fragmen berkontribusi pada sebagian UI ke aktivitas host, yang disematkan sebagai bagian dari hierarki tampilan keseluruhan aktivitas. Ada dua cara untuk menambahkan fragmen ke layout aktivitas.

1. Deklarasikan fragmen dalam file layout aktivitas
2. Membuat transaksi fragmen dalam aktivitas

- **Mengelola Fragmen**

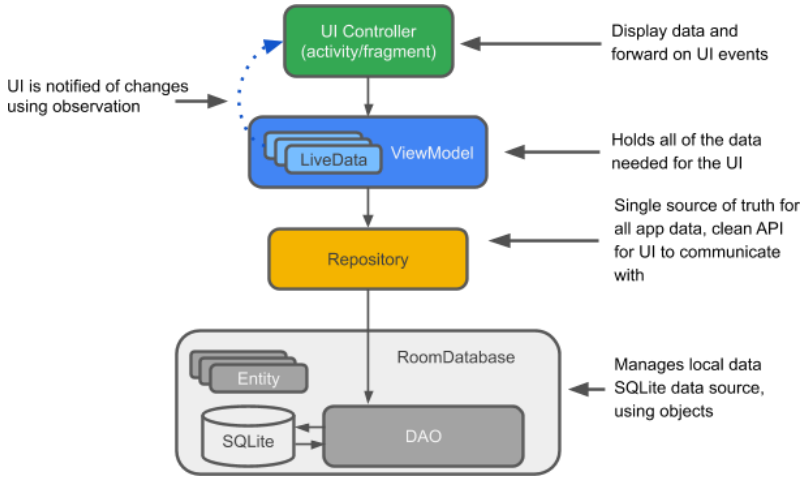
Untuk mengelola fragmen dalam aktivitas, kita perlu menggunakan FragmentManager. Untuk mendapatkannya, panggil getSupportFragmentManager() dari aktivitas kita.

Beberapa hal yang dapat Kita lakukan dengan FragmentManager antara lain:

1) Dapatkan fragmen yang ada di aktivitas dengan findFragmentById() (untuk fragmen yang menyediakan UI dalam layout aktivitas) atau findFragmentByTag() (untuk fragmen yang menyediakan atau tidak menyediakan UI).
2) Tarik fragmen dari back-stack, dengan popBackStack() (menyimulasikan perintah *Kembali* oleh pengguna).
3) Daftarkan listener untuk perubahan pada back-stack, dengan addOnBackStackChangedListener().

2.2 Menu dan Dialog

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.

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.

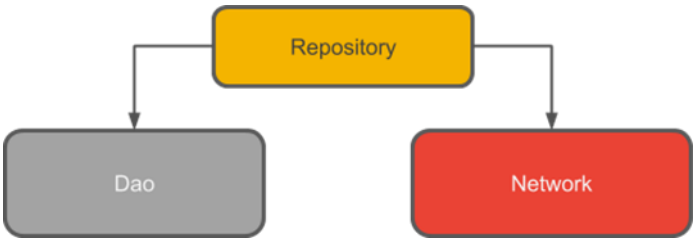
DAO harus berupa antarmuka atau kelas abstrak. Secara default, semua kueri harus dijalankan pada thread terpisah. Room memiliki dukungan coroutines, memungkinkan pertanyaan kita dijelaskan dengan pengubah penangguhan dan kemudian dipanggil dari coroutine atau dari fungsi suspensi lain.

- **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.

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.

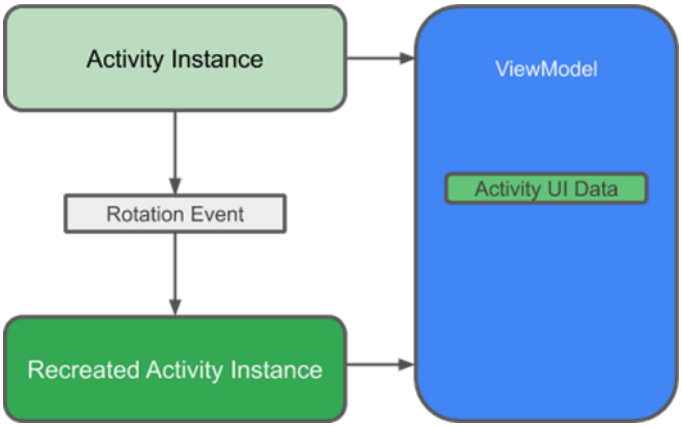
- **Repository** : Kelas yang kita buat yang terutama digunakan untuk mengelola beberapa sumber data.

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.



- **ViewModel** : Bertindak sebagai pusat komunikasi antara Repositori (data) dan UI. UI tidak perlu lagi khawatir tentang asal-usul data. Contoh ViewModel adalah Activity/Fragment.

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.



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:

1) Kita dapat menempatkan pengamat pada data (bukan polling untuk perubahan) dan hanya memperbarui UI ketika data benar-benar berubah.
2) Repositori dan UI sepenuhnya dipisahkan oleh ViewModel.
3) 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.

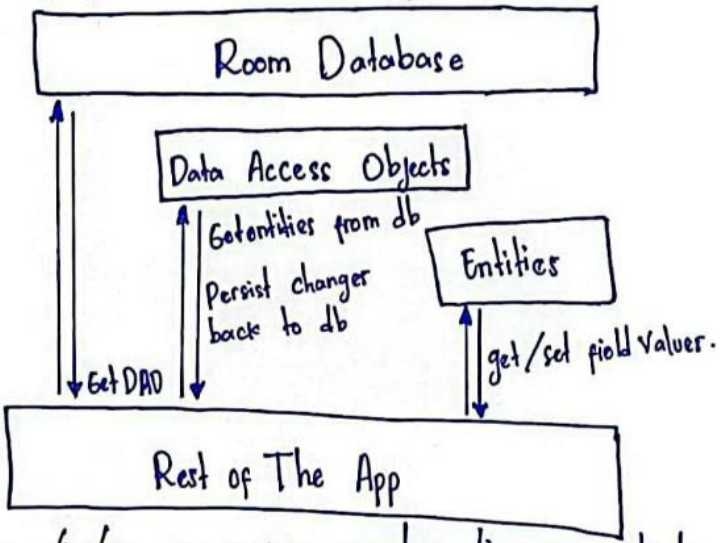
BAB III
· TUGAS PENDAHULUAN

3.1 Soal

1. Gambarlah diagram Arsitektur room pada Android!
2. Jelaskan keterkaitan room database pada diagram arsitektur yang kalian gambar

3.2 Jawaban

1. Gambar Diagram Arsitektur library Room



2. Gambar diatas merupakan gambar diagram arsitektur room database. Room ini dibagi menjadi 3 buah komponen utama yaitu:
   - Class database : Menyimpan database yang berfungsi sebagai titik akses utama bagi koneksi kedata persiten aplikasi
   - Entitidata : Menampilkan tabel pada database
   - Objek akses data (DAO) : Menyediakan metode yang dapat membuat CRUD aplikasi

   penjelasan : Class database menyediakan aplikasi dengan instance DAO yang terkait dengan database. Aplikasi dapat menggunakan DAO untuk mengambil data dari database sebagai instance objek entity data yang ditentukan untuk memperbarui baris dari tabel yang sesuai atau membuat baris baru untuk pengujian.

BAB IV

IMPLEMENTASI

4.1 SourCode

a. AndroidManifest

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
android:name=".activity.EditActivity"></activity>
        <activity android:name=".activity.MainActivity">
            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

b. EditActivity

```
package com.lazday.kotlinroommvvm.activity

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import com.lazday.kotlinroommvvm.R
import com.lazday.kotlinroommvvm.room.Constant
import com.lazday.kotlinroommvvm.room.Note
import com.lazday.kotlinroommvvm.room.NoteDB
import kotlinx.android.synthetic.main.activity_edit.*
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch

class EditActivity : AppCompatActivity() {

    private val db by lazy { NoteDB(this) }
    private var noteId = 0

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

```
        setContentView(R.layout.activity_edit)
        setupView()
        setupLstener()
    }

    private fun setupView(){

supportActionBar!!.setDisplayHomeAsUpEnabled(true)
        when (intentType()) {
            Constant.TYPE_CREATE -> {
                supportActionBar!!.title = "BUAT BARU"
                button_save.visibility = View.VISIBLE
                button_update.visibility = View.GONE
            }
            Constant.TYPE_READ -> {
                supportActionBar!!.title = "BACA"
                button_save.visibility = View.GONE
                button_update.visibility = View.GONE
                getNote()
            }
            Constant.TYPE_UPDATE -> {
                supportActionBar!!.title = "EDIT"
                button_save.visibility = View.GONE
                button_update.visibility = View.VISIBLE
                getNote()
            }
        }
    }

    private fun setupLstener(){
        button_save.setOnClickListener {
            CoroutineScope(Dispatchers.IO).launch {
                db.noteDao().addNote(
                    Note(
                        0,
                        edit_title.text.toString(),
                        edit_note.text.toString()
                    )
                )
                finish()
            }
        }
        button_update.setOnClickListener {
            CoroutineScope(Dispatchers.IO).launch {
                db.noteDao().updateNote(
                    Note(
                        noteId,
                        edit_title.text.toString(),
                        edit_note.text.toString()
                    )
                )
                finish()
            }
        }
    }

    private fun getNote(){
        noteId = intent.getIntExtra("note_id", 0)
        CoroutineScope(Dispatchers.IO).launch {
```

```
            val notes =
db.noteDao().getNote(noteId).get(0)
            edit_title.setText( notes.title )
            edit_note.setText( notes.note )
        }
    }

    override fun onSupportNavigateUp(): Boolean {
        finish()
        return super.onSupportNavigateUp()
    }

    private fun intentType(): Int {
        return intent.getIntExtra("intent_type", 0)
    }
}
```

c. MainActivity

```
package com.lazday.kotlinroommvvm.activity

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import androidx.appcompat.app.AlertDialog
import androidx.recyclerview.widget.LinearLayoutManager
import com.lazday.kotlinroommvvm.R
import com.lazday.kotlinroommvvm.room.Constant
import com.lazday.kotlinroommvvm.room.Note
import com.lazday.kotlinroommvvm.room.NoteDB
import kotlinx.android.synthetic.main.activity_main.*
import kotlinx.coroutines.*

class MainActivity : AppCompatActivity() {

    private val db by lazy { NoteDB(this) }
    lateinit var noteAdapter: NoteAdapter

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        setupView()
        setupListener()
        setupRecyclerView()
    }

    override fun onResume() {
        super.onResume()
        loadData()
    }

    private fun loadData(){
        CoroutineScope(Dispatchers.IO).launch {
            noteAdapter.setData(db.noteDao().getNotes())
            withContext(Dispatchers.Main) {
                noteAdapter.notifyDataSetChanged()
            }
        }
```

```
    }

    private fun setupView (){
        supportActionBar!!.apply {
            title = "Catatan"
        }
    }

    private fun setupListener(){
        button_create.setOnClickListener {
            intentEdit(Constant.TYPE_CREATE, 0)
        }
    }

    private fun setupRecyclerView () {

        noteAdapter = NoteAdapter(
            arrayListOf(),
            object : NoteAdapter.OnAdapterListener {
                override fun onClick(note: Note) {
                    intentEdit(Constant.TYPE_READ,
note.id)
                }

                override fun onUpdate(note: Note) {
                    intentEdit(Constant.TYPE_UPDATE,
note.id)
                }

                override fun onDelete(note: Note) {
                    deleteAlert(note)
                }

            })

        list_note.apply {
            layoutManager =
LinearLayoutManager(applicationContext)
            adapter = noteAdapter
        }

    }

    private fun intentEdit(intent_type: Int, note_id:
Int) {
        startActivity(
            Intent(this, EditActivity::class.java)
                .putExtra("intent_type", intent_type)
                .putExtra("note_id", note_id)
        )

    }

    private fun deleteAlert(note: Note){
        val dialog = AlertDialog.Builder(this)
        dialog.apply {
            setTitle("Konfirmasi Hapus")
            setMessage("Yakin hapus ${note.title}?")
            setNegativeButton("Batal") { dialogInterface,
```

```
i ->
                    dialogInterface.dismiss()
                }
                setPositiveButton("Hapus") { dialogInterface,
i ->
                    CoroutineScope(Dispatchers.IO).launch {
                        db.noteDao().deleteNote(note)
                        dialogInterface.dismiss()
                        loadData()
                    }
                }
            }

            dialog.show()
        }
    }
```

d. NoteAdapter

```
package com.lazday.kotlinroommvvm.activity

import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView
import com.lazday.kotlinroommvvm.R
import com.lazday.kotlinroommvvm.room.Note
import kotlinx.android.synthetic.main.adapter_main.view.*

class NoteAdapter (var notes: ArrayList<Note>, var
listener: OnAdapterListener) :
    RecyclerView.Adapter<NoteAdapter.NoteViewHolder>(){

    override fun onCreateViewHolder(parent: ViewGroup,
viewType: Int): NoteViewHolder {
        return NoteViewHolder(
            LayoutInflater.from(parent.context)
                .inflate(
                    R.layout.adapter_main,
                    parent,
                    false
                )
        )
    }

    override fun getItemCount() = notes.size

    override fun onBindViewHolder(holder: NoteViewHolder,
position: Int) {
        val note = notes[position]
        holder.view.text_title.text = note.title
        holder.view.text_title.setOnClickListener {
            listener.onClick(note)
        }
        holder.view.icon_edit.setOnClickListener {
            listener.onUpdate(note)
        }
        holder.view.icon_delete.setOnClickListener {
```

```
                listener.onDelete(note)
            }
        }

        class NoteViewHolder(val view: View) :
    RecyclerView.ViewHolder(view)

        fun setData(newList: List<Note>) {
            notes.clear()
            notes.addAll(newList)
        }

        interface OnAdapterListener {
            fun onClick(note: Note)
            fun onUpdate(note: Note)
            fun onDelete(note: Note)
        }
    }
```

e. Constant

```
package com.lazday.kotlinroommvvm.room

class Constant {
    companion object {
        const val TYPE_READ     = 0
        const val TYPE_CREATE   = 1
        const val TYPE_UPDATE   = 2
    }
}
```

f. Note

```
package com.lazday.kotlinroommvvm.room

import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity
data class Note(
    @PrimaryKey(autoGenerate = true)
    val id : Int = 0,
    val title: String,
    val note: String
)
```

g. NoteDao

```
package com.lazday.kotlinroommvvm.room

import androidx.room.*

@Dao
interface NoteDao {
    @Insert
    suspend fun addNote(note: Note)
```

```
    @Query("SELECT * FROM note ORDER BY id DESC")
    suspend fun getNotes() : List<Note>

    @Query("SELECT * FROM note WHERE id=:note_id")
    suspend fun getNote(note_id: Int) : List<Note>

    @Update
    suspend fun updateNote(note: Note)

    @Delete
    suspend fun deleteNote(note: Note)
}
```

h. NoteDB

```
package com.lazday.kotlinroommvvm.room

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase

@Database(
    entities = [Note::class],
    version = 1
)
abstract class NoteDB : RoomDatabase(){

    abstract fun noteDao() : NoteDao

    companion object {

        @Volatile private var instance : NoteDB? = null
        private val LOCK = Any()

        operator fun invoke(context: Context) = instance
?: synchronized(LOCK){
            instance ?: buildDatabase(context).also {
                instance = it
            }
        }

        private fun buildDatabase(context: Context) =
Room.databaseBuilder(
            context.applicationContext,
            NoteDB::class.java,
            "note12345.db"
        ).build()

    }
}
```

i. ExampleInstrumentTest

```
package com.lazday.kotlinroommvvm

import androidx.test.platform.app.InstrumentationRegistry
import androidx.test.ext.junit.runners.AndroidJUnit4
```

```
import org.junit.Test
import org.junit.runner.RunWith

import org.junit.Assert.*

/**
 * Instrumented test, which will execute on an Android
device.
 *
 * See [testing
documentation](http://d.android.com/tools/testing).
 */
@RunWith(AndroidJUnit4::class)
class ExampleInstrumentedTest {
    @Test
    fun useAppContext() {
        // Context of the app under test.
        val appContext =
InstrumentationRegistry.getInstrumentation().targetContex
t
        assertEquals("com.lazday.kotlinroommvvm",
appContext.packageName)
    }
}
```

j. ExampleUnirTest

```
package com.lazday.kotlinroommvvm

import org.junit.Test

import org.junit.Assert.*

/**
 * Example local unit test, which will execute on the
development machine (host).
 *
 * See [testing
documentation](http://d.android.com/tools/testing).
 */
class ExampleUnitTest {
    @Test
    fun addition_isCorrect() {
        assertEquals(4, 2 + 2)
    }
}
```

k. Activity_edit

```
<?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="match_parent"
```

```
    tools:context=".activity.EditActivity"
    android:padding="20dp"
    >

    <EditText
        android:id="@+id/edit_title"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="Judul"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        />
    <EditText
        android:id="@+id/edit_note"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="Tulis Catatan"
        android:minLines="3"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"

app:layout_constraintTop_toBottomOf="@+id/edit_title"
        android:layout_marginTop="10dp"
        android:gravity="top"
        />
    <Button
        android:id="@+id/button_save"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="SAVE"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"

app:layout_constraintTop_toBottomOf="@+id/edit_note"
        android:layout_marginTop="20dp"
        />
    <Button
        android:id="@+id/button_update"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="UPDATE"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"

app:layout_constraintTop_toBottomOf="@+id/button_save"
        android:layout_marginTop="20dp"
        />

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

l. Activity_main

```
<?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="match_parent"
    tools:context=".activity.MainActivity">

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

app:layout_constraintBottom_toTopOf="@+id/button_create"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:listitem="@layout/adapter_main"
        />

    <Button
        android:id="@+id/button_create"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Tulis Catatan"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        android:layout_margin="10dp"
        />

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

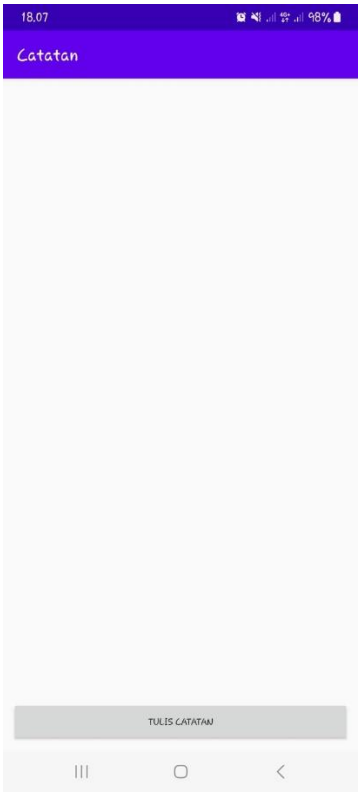
m. Adapter_main

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android
"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:padding="10dp">

    <TextView
        android:id="@+id/text_title"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        tools:text="Nanti kita cerita hari ini"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"

app:layout_constraintEnd_toStartOf="@+id/icon_edit"
        />
    <ImageView
        android:id="@+id/icon_edit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_edit"
```

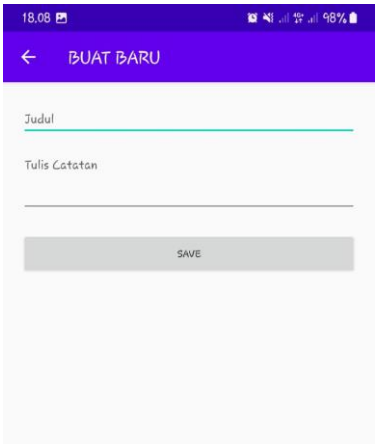
```
            android:padding="10dp"
            app:layout_constraintTop_toTopOf="parent"

    app:layout_constraintEnd_toStartOf="@+id/icon_delete"
            />
        <ImageView
            android:id="@+id/icon_delete"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/ic_delete"
            android:padding="10dp"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            />

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

4.2 Hasil Run

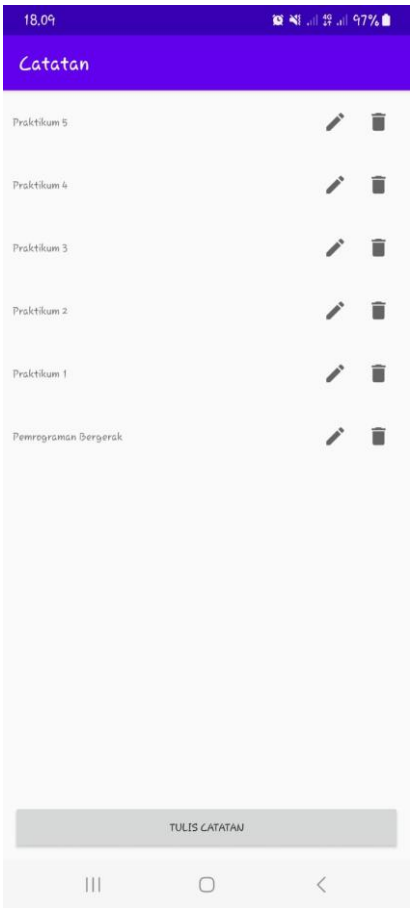
a. Pada gambar pertama hasil run berikut menunjukkan tampilan awal atau fragment 1 pada sistem yang telah dibuat



b. Gambar kedua hasil run menunjukkan jika setelah kita mengklik tulis catatan, maka akan muncul untuk menulis catatan yang kita inginkan.

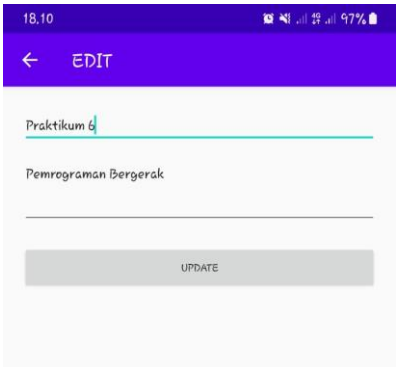


c. Pada gambar hasil run ketiga berikut, menunjukkan bahwa setelah kita memasukkan beberapa catatan akan muncul seperti gambar tersebut yang juga terdapat gambar pensil dan sampah, dimana pensil sendiri digunakan untuk mengubah catatan tersebut sesuai yang kita inginkan.

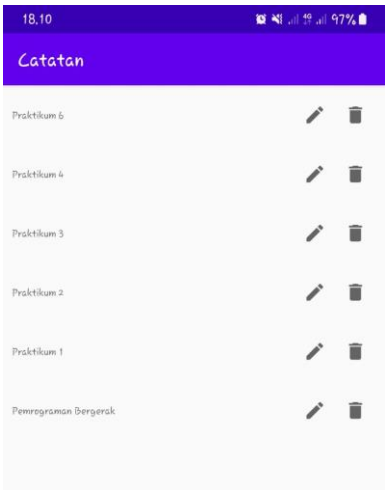


d. Gambar hasil run ke empat ini menunjukkan bahwa jika setelah mengklik tanda pensil seperti yang ada pada keterangan gambar ketiga,

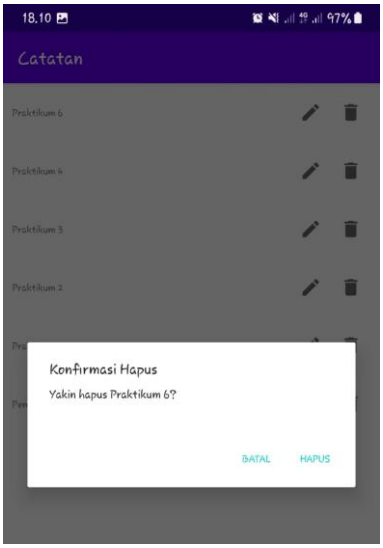
maka akan muncul halaman berikut, dimana kita dapat mengubah catatan sesuai yang kita inginkan, jika sudah selesai kemudian klik update dibagian bawahnya.



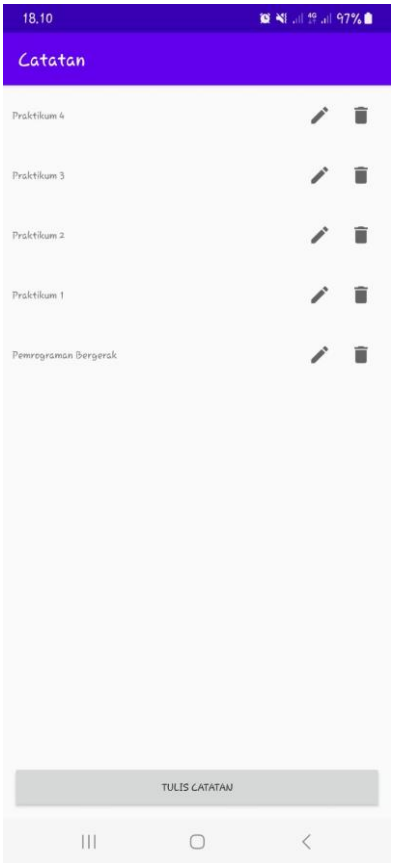
e. Pada gambar ke lima menunjukkan bahwa setelah catatan diupdate atau diedit maka akan tampil seperti semula, tapi yang awalnya bertuliskan Praktikum 5 menjadi Praktikum 6 seperti gambar dibawah ini.



f. Pada gambar ke enam dibawah ini menunjukkan bahwa setelah kita mengklik gambar sampah yang ada disebelah kanan gambar pensil, maka akan menampilkan sebuah peringatan yang bertuliskan “konfirmasi hapus” dimana terdapat pilihan “hapus” dan “batal” jika kita ingin menghapus catatan tersebut, atau batal menghapusnya.



g. Terakhir gambar hasil run dibawah ini menunjukkan bahwa pada tulisan “Praktikum 5” yang di update menjadi “Praktikum 6” di hapus, catatan yang tersedia atau terdapat pada gambar hanya sampai pada “Praktikum 4”



BAB V
PENUTUP

5.1 Analisa

Dari hasil praktikum, praktikan menganalisa bahwasanya diantara fragment dan room database merupakan kedua konsep yang memiliki keterkaitan dan penting dalam melakukan pengembangan aplikasi android. Fragment adalah komponen UI yang dapat digunakan kembali dan berguna dalam mengorganisir tampilan UI pada aplikasi yang lebih kompleks. Fragment memudahkan pengembangan aplikasi yang responsif pada berbagai ukuran layar dan memungkinkan penggunaan ulang kode serta membuat kode menjadi lebih modular dan mudah di atur. Dengan menggunakan Fragment dapat memudahkan pengembangan aplikasi yang responsif pada berbagai ukuran layar dan memungkinkan penggunaan ulang kode serta membuat kode menjadi lebih modular dan mudah di atur, sedangkan Room Databse merupakan library android jetpack yang menyediakan lapisan abstraksi pada SQLite dan memudahkan pengembang dalam mengakses database dan melakukan operasi CRUD. Kedua komponen ini memberikan manfaat untuk membantu dan memudahkan proses pengembangan aplikasi yang kompleks dan memiliki database lokal. Guna untuk membuat sebuah aplikasi android yang berkualitas.

5.2 Kesimpulan

Dari pembahasan diatas, praktikan dapat mengambil kesimpulan bahwasanya Berdasarkan informasi yang diberikan, dapat disimpulkan bahwa modul yang mengeksplorasi fragmen dan room database merupakan modul yang terkait dengan pengembangan aplikasi Android. Fragmen adalah bagian dari aplikasi Android yang bertindak sebagai bagian dari layar UI. Fragmen memungkinkan pengembang membagi antarmuka pengguna menjadi beberapa bagian yang dapat digunakan secara mandiri, yang meningkatkan fleksibilitas pengembangan aplikasi. Sementara itu, Room Database merupakan salah satu library yang disediakan oleh Google yang memudahkan dalam mengakses dan menggunakan database di aplikasi Android. Dalam pengembangan aplikasi Android, penggunaan fragmen dan room database sangat penting untuk meningkatkan aplikasi dan performa.

Dengan menjelajahi fragmen, pengembang dapat membagi antarmuka pengguna menjadi bagian yang lebih kecil dan lebih mudah dikelola, sedangkan basis data spasial memungkinkan pengembang mengakses dan memproses data dengan mudah dan efisien. Oleh karena itu, mempelajari tentang fragmen dan room database sangat penting dan diperlukan untuk menjadi pengembang Android yang kompeten.