1. Pada arsitektur MVP, untuk membuat 1 layar (activity), dibutuhkan beberapa class, yaitu:
2. Model: merupakan bentuk dari data-data yang dibutuhkan, pada umumnya data yang akan ditampung/diolah oleh model adalah data yang di dapatkan dari repository.
3. View: merupakan class logic untuk memberikan interaksi-interaksi pada halaman interface.
4. Presenter: kelas ini merupakan kelas untuk mengatur dan meneruskan interaksi dari view ke model, serta mengelola response dari model ke view.
5. Cara komunikasi dari View ke Presenter melibatkan pemanggilan fungsi dari view ke presenter melalui tampilan aplikasi, kemudian presenter melakukan logika bisnis dan berinteraksi dengan model untuk mengirimkan aksi yang diberikan oleh view.
6. Cara komunikasi dari Presenter ke View yaitu presenter memberikan informasi yang di dapatkan dari Model yang di proses di presenter kemudian di kirimkan ke view untuk di tampilkan pada aplikasi
7. Berikut adalah source code untuk membuat database dan model untuk membuat aplikasi android keuangan sederhana.

**Deskripsi:**

Saya akan membuat database menggunakan Room persistence dengan menggunakan Bahasa Kotlin. Terdapat 3 class yang di butuhkan untuk membuat Database pada Android.

**Class Entity/Model**

|  |
| --- |
| @Entity(tableName = "uang\_masuk\_table") @Parcelize data class UangMasuk(  @PrimaryKey(autoGenerate = true)  @ColumnInfo(name = "uang\_masuk\_id")  var uangMasukId: Int = 0,   @ColumnInfo(name = "terima\_dari")  var terimaDari: String,   @ColumnInfo(name = "keterangan")  var keterangan: String,   @ColumnInfo(name = "jumlah")  var jumlah: Double, ) : Parcelable |

**Class Dao**

|  |
| --- |
| @Dao interface DatabaseDao {  @Insert(onConflict = OnConflictStrategy.*REPLACE*)  fun insertData(uangMasuk: UangMasuk) } |

**Class Database**

|  |
| --- |
| @Database(entities = [UangMasuk::class], version = 1) abstract class AppDatabase : RoomDatabase() {  abstract fun databaseDao(): DatabaseDao   companion object {  private const val DB\_NAME = "keuangan\_sederhana.db"   fun getDatabase(context: Context): AppDatabase {  return Room.databaseBuilder(context, AppDatabase::class.*java*, DB\_NAME)  .build()  }  } } |

1. Untuk membuat perubahan pada aplikasi yang terlah rilis yaitu
2. Pertama melakukan penyesuaian terhadap table-table yang akan di ubah, berikut perubahan yang di butuhkan pada class sebelumnya.

Class Entity/Model

|  |
| --- |
| @Entity(tableName = "uang\_masuk\_table") @Parcelize data class UangMasuk(  @PrimaryKey(autoGenerate = true)  @ColumnInfo(name = "uang\_masuk\_id")  var uangMasukId: Int = 0,   @ColumnInfo(name = "terima\_dari")  var terimaDari: String,   @ColumnInfo(name = "keterangan")  var keterangan: String,   @ColumnInfo(name = "jumlah")  var jumlah: Double,   @ColumnInfo(name = "tanggal")  var tanggal: String,   @ColumnInfo(name = "nomor")  var nomor: String, ) : Parcelable |

1. Kedua yaitu agar data local user tidak hilang, project yang di buat perlu mengubah nomor versi dari database ke yang lebih tinggi dan menambahkan instruksi migrasi.

Class Database

|  |
| --- |
| @Database(entities = [UangMasuk::class], version = 2) abstract class AppDatabase : RoomDatabase() {  abstract fun databaseDao(): DatabaseDao   companion object {  private const val DB\_NAME = "keuangan\_sederhana.db"   val MIGRATION\_1\_2: Migration = object : Migration(1, 2) {  override fun migrate(database: SupportSQLiteDatabase) {  database.execSQL("ALTER TABLE uang\_masuk\_table ADD COLUMN tanggal TEXT NOT NULL DEFAULT ''")  database.execSQL("ALTER TABLE uang\_masuk\_table ADD COLUMN nomor TEXT")  }  }   fun getDatabase(context: Context): AppDatabase {  return Room.databaseBuilder(context, AppDatabase::class.*java*, DB\_NAME)  .addMigrations(MIGRATION\_1\_2)  .build()  }  } } |

1. Untuk mendapatkan nomor secara otomatis, perlu penyesuaian pada class Dao dan pembuatan fungsi untuk mendapatkan nomor secara otomatis
2. Penyesuaian class Dao

|  |
| --- |
| @Dao interface DatabaseDao {  // ...  @Query("SELECT COUNT(uang\_masuk\_id) FROM uang\_masuk\_table WHERE tanggal = :todayDate")  fun getTotalOrderToday(todayDate: String): LiveData<Int>  } |

1. Pembuatan Fungsi

|  |
| --- |
| fun generateOrderNumber(): String {  val orderCount = database.getTotalOrderToday().value + 1  val formattedDate = SimpleDateFormat("yyMMdd", Locale.getDefault()).format(Date())  return "UM/$formattedDate/$orderCount" } |

1. Untuk melakukan release ke playstore kembali setelah beberapa perubahan sesuai dengan soal no 7, berikut adalah tahapannya
2. Pertama membuat Tabel Rekening sebagai Entity dan Model

|  |
| --- |
| @Entity(tableName = "rekening\_table") @Parcelize data class Rekening(  @PrimaryKey(autoGenerate = true)  @ColumnInfo(name = "rekening\_id")  var rekeningId: Long = 0,   @ColumnInfo(name = "nama\_bank")  var namaBank: String,   @ColumnInfo(name = "nomor\_rekening")  var nomorRekening: String,   @ColumnInfo(name = "atas\_nama")  var atasNama: String ): Parcelable |

1. Menambahkan metode insert rekening pada dao

|  |
| --- |
| @Dao interface DatabaseDao {  // ...  @Insert(onConflict = OnConflictStrategy.REPLACE)  fun insertRekening(rekening: Rekening)  } |

1. Kemudian menyesuaikan nomor versi dan melakukan migrasi

|  |
| --- |
| @Database(entities = [UangMasuk::class, Rekening::class], version = 3) abstract class AppDatabase : RoomDatabase() {  abstract fun databaseDao(): DatabaseDao   companion object {  private const val DB\_NAME = "keuangan\_sederhana.db"   val MIGRATION\_2\_3: Migration = object : Migration(2, 3) {  override fun migrate(database: SupportSQLiteDatabase) {  database.execSQL(  "CREATE TABLE IF NOT EXISTS rekening\_table (" +  "rekening\_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, " +  "nama\_bank TEXT NOT NULL, " +  "nomor\_rekening TEXT NOT NULL, " +  "atas\_nama TEXT NOT NULL)"  )   database.execSQL("ALTER TABLE uang\_masuk\_table ADD COLUMN rekening\_id INTEGER DEFAULT 0")  }  }   fun getDatabase(context: Context): AppDatabase {  return Room.databaseBuilder(context, AppDatabase::class.*java*, DB\_NAME)  .addMigrations(MIGRATION\_2\_3)  .build()  }  } } |

1. Berikut adalah fungsi untuk menghitung total harga sebelum di Pajak

|  |
| --- |
| data class TaxResult(val pajakRp: Double, val netSales: Double)  fun calculateTaxAndPriceBeforeTax(input: Double): TaxResult {  val netSales = input - (input \* 0.1)  val pajakRp = input - netSales   return TaxResult(pajakRp, netSales) } |

1. Berikut adalah fungsi untuk menghitung total harga setelah diskon dengan inputan diskon persen bertingkat

|  |
| --- |
| data class DiscountResult(val totalDiskon: Int, val totalHargaSetelahDiskon: Int)  fun calculateTotalDiscountAndTotalPriceAfterDiscount(totalBeforeDiscount: Int, discounts: IntArray): DiscountResult {  var totalBeforeDiscountTemp = totalBeforeDiscount  var totalDiscount = 0   for (discount in discounts) {  val discountAmount = (totalBeforeDiscountTemp \* discount) / 100  totalDiscount += discountAmount  totalBeforeDiscountTemp -= discountAmount  }   return DiscountResult(totalDiscount, totalBeforeDiscountTemp) } |

1. Berikut adalah fungsi untuk menghitung share revenue ojek online

|  |
| --- |
| data class PriceResult(val netUntukResto: Double, val shareUntukOjol: Double)  enum class FoodDeliveryService(val markupPercentage: Double) {  *GOFOOD*(0.10),  *GRABFOOD*(0.20),  *SHOPPEFOOD*(0.25) }  fun calculatePriceWithMarkup(originalPrice: Double, service: FoodDeliveryService): PriceResult {  val hargaMarkup = originalPrice + (originalPrice \* service.markupPercentage)  val shareUntukOjol = hargaMarkup \* 0.20  val netUntukResto = hargaMarkup - shareUntukOjol   return PriceResult(netUntukResto, shareUntukOjol) } |