

PEMROGRAMAN MOBILE (PRAKTIKUM) II
MODUL 6 SQLITE DATABASE



Disusun Oleh:

Rafid Nagara Darmakusuma (081911633034)

Dosen Pengampu:

Purbandini S.Si., M.Kom. .

PROGRAM STUDI SISTEM INFORMASI
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS AIRLANGGA

2021/2022

Tugas

Berikut ini adalah tugas praktikum modul 6 adalah

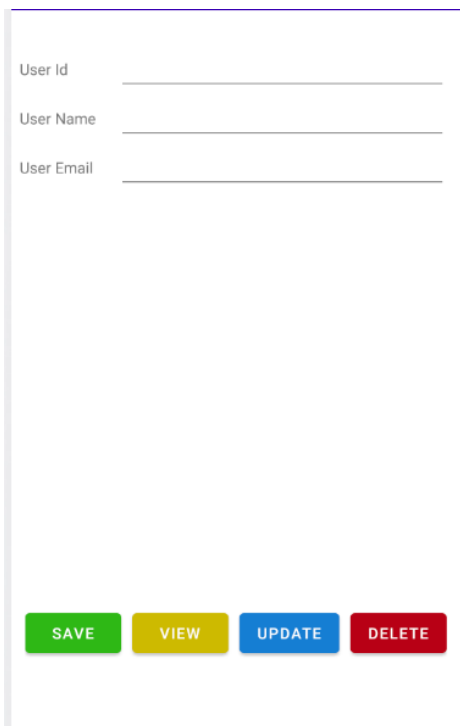
1. implementasikan source code pada poin D (Praktikum),
2. screenshot hasilnya,
3. beri penjelasan dari setiap potongan source code tersebut!

File yang dikumpulkan adalah laporan pdf dan file apk yang dijadikan satu dalam file berbentuk .rar atau .zip! Format pengumpulan: Prak PM 6_NIM_Nama.rar/.zip

Pengumpulan paling lambat hari Sabtu/ 13 November 2021 pukul 23.59 Wib

Hasil Implementasi Praktikum

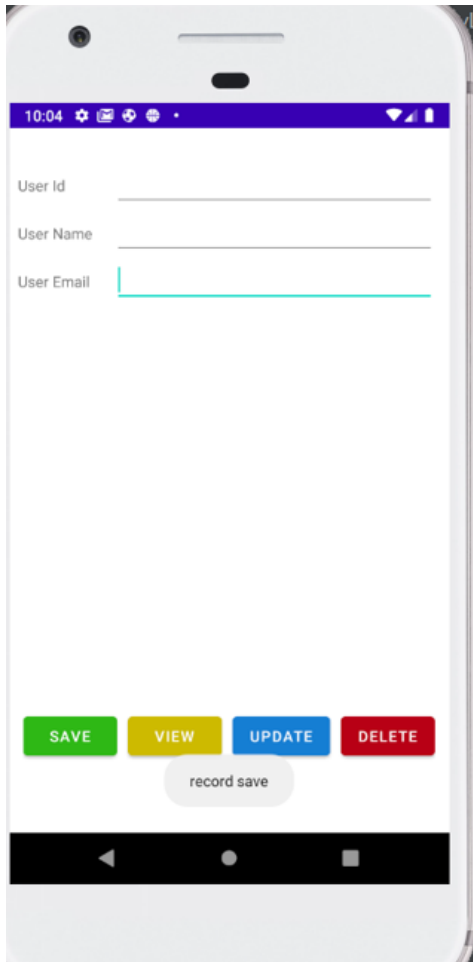
Pada praktikum ini, dibuat sebuah project baru bernama modul6 . Aplikasi ini memiliki tampilan layout utama yaitu Main Activity yang ada pada file activity_main.xml yang akan tampil dan menjalankan fungsi fitur pada file logika pada MainActivity.kt. Tampilan dari layout Main Activity seperti berikut ini:



The screenshot displays a mobile application interface for user management. It features three input fields at the top: 'User Id', 'User Name', and 'User Email', each followed by a horizontal line for text entry. Below these fields, there is a large empty rectangular area. At the bottom of the screen, there are four colored buttons: a green 'SAVE' button, a yellow 'VIEW' button, a blue 'UPDATE' button, and a red 'DELETE' button.

dimana pada tampilan awal akan menampilkan form untuk melakukan pendataan nama dan email dari pegawai. pengguna dapat melakukan pengisian form dan menekan tombol save untuk menyimpan data yang telah diinputkan pada form dan akan muncul notif teks kecil

menggunakan fungsi toast yang memberitahukan bahwa proses penyimpanan berhasil, seperti pada gambar berikut ini



Source code:

```
fun saveRecord(view: View){
    val id = u_id.text.toString()
    val name = u_name.text.toString()
    val email = u_email.text.toString()
    val databaseHandler: DatabaseHandler= DatabaseHandler(context: this)
    if(id.trim()!=" " && name.trim()!=" " && email.trim()!=" "){
        val status = databaseHandler.addEmployee(EmpModelClass(Integer.parseInt(id),name, email))
        if(status > -1){
            Toast.makeText(applicationContext, text: "record save",Toast.LENGTH_LONG).show()
            u_id.text.clear()
            u_name.text.clear()
            u_email.text.clear()
        }
    }else{
        Toast.makeText(applicationContext, text: "id or name or email cannot be blank",Toast.LENGTH_LONG).show()
    }
}
```

kode diatas merupakan fungsi yang akan melakukan proses penyimpanan data yang akan diproses oleh file DatabaseHandler.kt. Namun sebelum melakukan proses inisialisasi dari setiap elemen

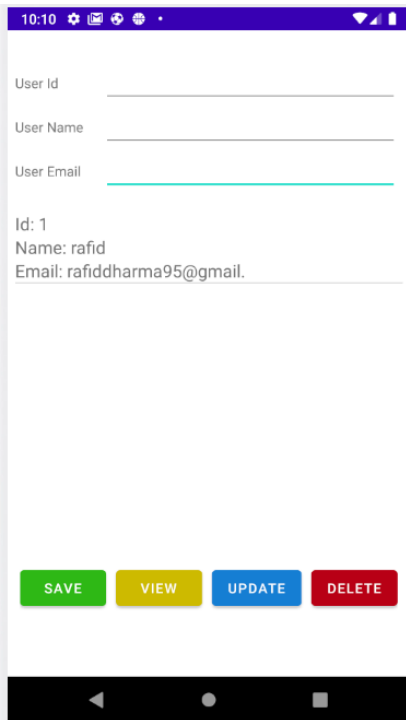
dan fungsi dari proses input, edit, dan delete database perlu dilakukan import beberapa extension dari library android seperti berikut ini :

```
import android.app.AlertDialog
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.*
import kotlinx.android.synthetic.main.activity_main.*
import android.content.DialogInterface
```

dan melakukan konfigurasi untuk dapat mengakses library `kotlinx.android.synthetic.mai.activity_main.*` pada file `build.gradle` pada gradle script seperti berikut ini:

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'kotlin-android-extensions'
```

selain itu proses pada Main Activity dapat menampilkan data yang telah diinputkan atau diubah ke dalam database sql Lite dengan menekan tombol view, maka akan muncul list dari view data yang ada pada database. seperti berikut ini :

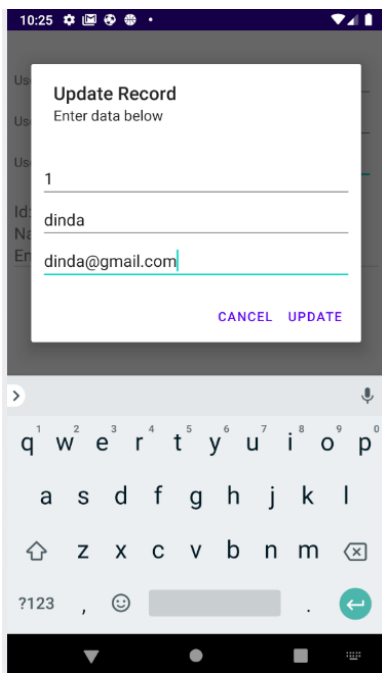


Source code:

```
fun viewRecord(view: View){
    //creating the instance of DatabaseHandler class
    val databaseHandler: DatabaseHandler= DatabaseHandler( context: this)
    //calling the viewEmployee method of DatabaseHandler class to read the records
    val emp: List<EmpModelClass> = databaseHandler.viewEmployee()
    val empArrayId = Array<String>(emp.size){"0"}
    val empArrayName = Array<String>(emp.size){"null"}
    val empArrayEmail = Array<String>(emp.size){"null"}
    var index = 0
    for(e in emp){
        empArrayId[index] = e.userId.toString()
        empArrayName[index] = e.userName
        empArrayEmail[index] = e.userEmail
        index++
    }
    //creating custom ArrayAdapter
    val myListAdapter = MyListAdapter( context: this,empArrayId,empArrayName,empArrayEmail)
    listView.adapter = myListAdapter
}
```

kode diatas merupakan fungsi View Record dimana saat tombol view di tekan proses fungsi ini akan berjalan untuk menampilkan data kedalam view Main Activity yang telah tersimpan dalam database dengan memanggil class DatabaseHandler pada file DatabaseHandler.kt

Selanjutnya pada Main Activity juga terdapat fitur untuk melakukan perubahan terhadap data yang ada dalam data base. Dimana pengguna dapat melakukan perubahan nama dan email dari pegawai sesuai Id dari data yang ada. Perubahan data pegawai dapat dilakukan seperti berikut in :



Sebelum:

User Id

User Name

User Email

Id: 1

Name: rafid

Email: rafiddharma95@gmail.

SAVE

VIEW

UPDATE

DELETE

Sesudah :

User Id

User Name

User Email

Id: 1

Name: dinda

Email: dinda@gmail.com

SAVE

VIEW

UPDATE

DELETE

dengan source code template tampilan layout update data seperti berikut:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:padding="10dp"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <EditText
        android:id="@+id/updateId"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="enter id" />

    <EditText
        android:id="@+id/updateName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="enter name" />

    <EditText
        android:id="@+id/updateEmail"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="enter email" />

</LinearLayout>
```

dan untuk fitur terakhir dari pengolahan data pegawai yaitu dilakukan delete dari data pegawai sesuai ID yang ada pada database .Berikut tampilan dari proses delete data dengan menekan tombol delete

Source code:

```
fun deleteRecord(view: View){
    //creating AlertDialog for taking user id
    val dialogBuilder = AlertDialog.Builder( context: this)
    val inflater = this.layoutInflater
    val dialogView = inflater.inflate(R.layout.delete_dialog, root: null)
    dialogBuilder.setView(dialogView)

    val dltId = dialogView.findViewById(R.id.deleteId) as EditText
    dialogBuilder.setTitle("Delete Record")
    dialogBuilder.setMessage("Enter id below")
    dialogBuilder.setPositiveButton( text: "Delete", DialogInterface.OnClickListener { _, _ ->

        val deleteId = dltId.text.toString()
        //creating the instance of DatabaseHandler class
        val databaseHandler: DatabaseHandler= DatabaseHandler( context: this)
        if(deleteId.trim()!=""){
            //calling the deleteEmployee method of DatabaseHandler class to delete record
            val status = databaseHandler.deleteEmployee(EmpModelClass(Integer.parseInt(deleteId), userName: "", userEmail: ""))
            if(status > -1){
                Toast.makeText(applicationContext, text: "record deleted",Toast.LENGTH_LONG).show()
            }
        }else{
            Toast.makeText(applicationContext, text: "id or name or email cannot be blank",Toast.LENGTH_LONG).show()
        }
    })
    dialogBuilder.setNegativeButton( text: "Cancel", DialogInterface.OnClickListener { _, _ ->
        //pass
    })
    val b = dialogBuilder.create()
    b.show()
}
```

source code diatas merupakan fungsi dari proses delete dengan melakukan proses pengecekan ID pada database. Jika id yang diinputkan bernilai tidak bernilai negatif dan tidak kosong maka akan dilakukan proses delete data pegawai dengan memanggil class dari databaseHandler dan menampilkan toast atau notifikasi “record deleted” jika user tidak menginputkan sesuai Id yang tertera maka akan muncul notifikasi “id or name or email cannot be blank”. dan berikut template tampilan layout dari proses delete pada file delete_dialog.xml :


```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:padding="10dp"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <EditText
        android:id="@+id/deleteId"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="enter id" />

</LinearLayout>

```

selain Main Activity juga terdapat layout untuk setiap proses yang dilakukan dalam perubahan dan penambahan data. Dimana saat user telah melakukan penambahan data dan akan muncul pada Main Activity maka list dari view yang ada pada elemen listView pada activity_main.xml. Seperti pada source code berikut ini:

```

    android:layout_marginTop="20sp">
    <ListView
        android:id="@+id/listView"
        android:layout_width="wrap_content"
        android:layout_height="320sp"/>

```

yang memiliki tampilan dasar seperti ini:

Item 1
Sub Item 1
Item 2
Sub Item 2
Item 3
Sub Item 3
Item 4
Sub Item 4
Item 5
Sub Item 5

dimana nilai item akan diatur pada file MyListAdapter.kt dengan memanggil nilai atribut dari database yang telah dibuat pada DatabaseHandler.kt dengan source code berikut:

```
import android.app.Activity
import android.view.View
import android.view.ViewGroup
import android.widget.ArrayAdapter
import android.widget.TextView

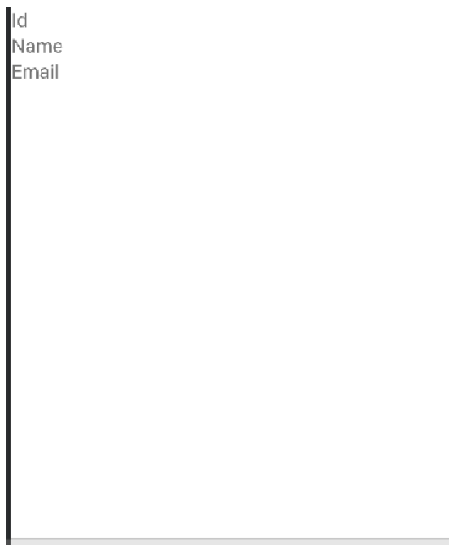
class MyListAdapter(private val context: Activity, private val id: Array<String>, private val name: Array<String>, private val email: Array<String>)
: ArrayAdapter<String>(context, R.layout.custom_list, name) {

    override fun getView(position: Int, view: View?, parent: ViewGroup): View {
        val inflater = context.layoutInflater
        val rowView = inflater.inflate(R.layout.custom_list, root = null, attachToRoot = true)

        val idText = rowView.findViewById(R.id.textViewId) as TextView
        val nameText = rowView.findViewById(R.id.textViewName) as TextView
        val emailText = rowView.findViewById(R.id.textViewEmail) as TextView

        idText.text = "Id: ${id[position]}"
        nameText.text = "Name: ${name[position]}"
        emailText.text = "Email: ${email[position]}"
        return rowView
    }
}
```

dan memiliki tampilan template layout untuk data pada file custom_list.xml seperti berikut ini:



dengan source code:

```
<?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"
    android:id="@+id/linearLayout">

    <TextView
        android:id="@+id/textViewId"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Id"
        android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium"/>

    <TextView
        android:id="@+id/textViewName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Name"
        android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium"/>
```

Nilai dari setiap atribut dari data yang diinputkan pada database terlebih dahulu di inisialisasi dan disimpan pada adapter class yang bernama class EmpModelClass pada file EmpModelClass.kt . Berikut source code dari inisialisasi pada class EmpModelClass :

```
class EmpModelClass (var userId:Int, val userName:String, val userEmail:String)
```

setelah itu baru dilakukan proses penyimpanan hasil inputan menggunakan variable yang telah diinputkan kedalam database yang telah dibuat pada file DatabaseHandler.kt dengan source code berikut:

```
import android.annotation.SuppressLint
import android.content.Context
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper
import android.content.ContentValues
import android.database.Cursor
import android.database.sqlite.SQLiteException

//creating the database logic, extending the SQLiteOpenHelper base class
class DatabaseHandler(context: Context):
    SQLiteOpenHelper(context,DATABASE_NAME,null,DATABASE_VERSION) {
    companion object {
        private val DATABASE_VERSION = 1
        private val DATABASE_NAME = "EmployeeDatabase"
        private val TABLE_CONTACTS = "EmployeeTable"
        private val KEY_ID = "id"
        private val KEY_NAME = "name"
        private val KEY_EMAIL = "email"
    }
    override fun onCreate(db: SQLiteDatabase?) {
```

// TODO("not implemented") //To change body of created functions use File | Settings | File Templates.

//creating table with fields

```
val CREATE_CONTACTS_TABLE = ("CREATE TABLE " + TABLE_CONTACTS +  
"  
    + KEY_ID + " INTEGER PRIMARY KEY," + KEY_NAME + " TEXT,"  
    + KEY_EMAIL + " TEXT" + ")")  
db?.execSQL(CREATE_CONTACTS_TABLE)  
}
```

override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {

// TODO("not implemented") //To change body of created functions use File | Settings | File Templates.

```
db!!.execSQL("DROP TABLE IF EXISTS " + TABLE_CONTACTS)  
onCreate(db)  
}
```

//method to insert data

```
fun addEmployee(emp: EmpModelClass):Long{  
    val db = this.writableDatabase  
    val contentValues = ContentValues()  
    contentValues.put(KEY_ID, emp.userId)  
    contentValues.put(KEY_NAME, emp.userName) // EmpModelClass Name  
    contentValues.put(KEY_EMAIL,emp.userEmail ) // EmpModelClass Phone  
    // Inserting Row  
    val success = db.insert(TABLE_CONTACTS, null, contentValues)  
    //2nd argument is String containing nullColumnHack  
    db.close() // Closing database connection  
    return success  
}
```

```

}

//method to read data

@SuppressLint("Range")

fun viewEmployee():List<EmpModelClass>{

    val empList:ArrayList<EmpModelClass> = ArrayList<EmpModelClass>()

    val selectQuery = "SELECT * FROM $TABLE_CONTACTS"

    val db = this.readableDatabase

    var cursor: Cursor? = null

    try{

        cursor = db.rawQuery(selectQuery, null)

    }catch (e: SQLiteException) {

        db.execSQL(selectQuery)

        return ArrayList()

    }

    var userId: Int

    var userName: String

    var userEmail: String

    if (cursor.moveToFirst()) {

        do {

            userId = cursor.getInt(cursor.getColumnIndex("id"))

            userName = cursor.getString(cursor.getColumnIndex("name"))

            userEmail = cursor.getString(cursor.getColumnIndex("email"))

            val emp= EmpModelClass(userId = userId, userName = userName, userEmail =
userEmail)

            empList.add(emp)

        } while (cursor.moveToNext())

    }

    return empList

}

//method to update data

```

```

fun updateEmployee(emp: EmpModelClass):Int{
    val db = this.writableDatabase
    val contentValues = ContentValues()
    contentValues.put(KEY_ID, emp.userId)
    contentValues.put(KEY_NAME, emp.userName) // EmpModelClass Name
    contentValues.put(KEY_EMAIL,emp.userEmail ) // EmpModelClass Email

    // Updating Row
    val success = db.update(TABLE_CONTACTS, contentValues,"id="+emp.userId,null)
    //2nd argument is String containing nullColumnHack
    db.close() // Closing database connection
    return success
}

//method to delete data
fun deleteEmployee(emp: EmpModelClass):Int{
    val db = this.writableDatabase
    val contentValues = ContentValues()
    contentValues.put(KEY_ID, emp.userId) // EmpModelClass UserId
    // Deleting Row
    val success = db.delete(TABLE_CONTACTS,"id="+emp.userId,null)
    //2nd argument is String containing nullColumnHack
    db.close() // Closing database connection
    return success
}
}

```