A Demonstration of Text Input and Validation with Android Compose

Abstract:

This demonstration showcases a simple login form implementation using Android Jetpack Compose, focusing on text input validation for email and password fields. The project leverages Compose declarative UI framework to create a responsive and efficient user interface.

Introduction:

Android Jetpack Compose provides a modern and efficient way to build user interfaces. In this demonstration, we'll explore how to create text input fields with validation using Compose.

Project Description:

The app is a sample project that demonstrates how to use the Android Compose UI toolkit to build a survey app. The app allows the user to answer a series of questions. It showcases some of the key features of the Compose UI toolkit, data management, and user interactions.

System requirement:

Hardware Requirements:

- Device: Android smartphone or tablet.
- Processor: Quad-core or higher.
- RAM: 2 GB or higher.
- Storage: 16 GB or higher.
- Display: 720p or higher resolution.

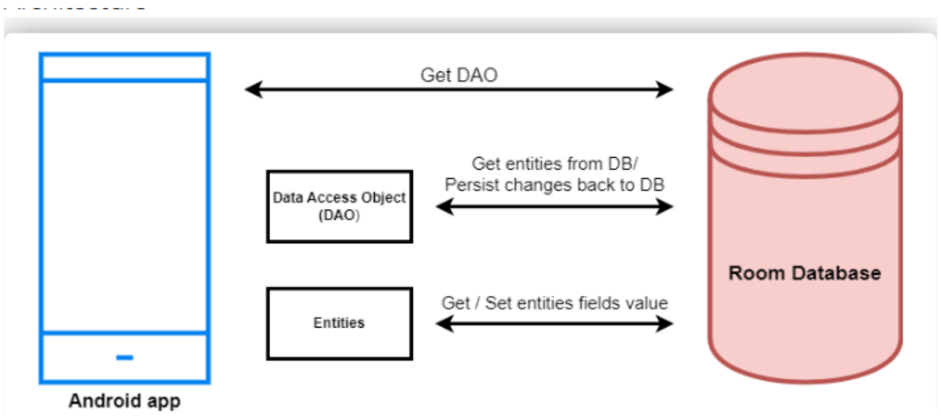
Software Requirements:

- Operating System: Android 11 (API level 30) or higher
- Android Jetpack Compose: 1.2.1

Tools used:

- Android Studio (Arctic Fox 2020.3.1)
- Android Jetpack Compose (1.2.1)
- Kotlin (1.7.10)

Architecture:


Get DAO
Data Access Object (DAO)
Get entities from DB/ Persist changes back to DB
Entities
Get / Set entities fields value
Room Database
Android app

Tasks:

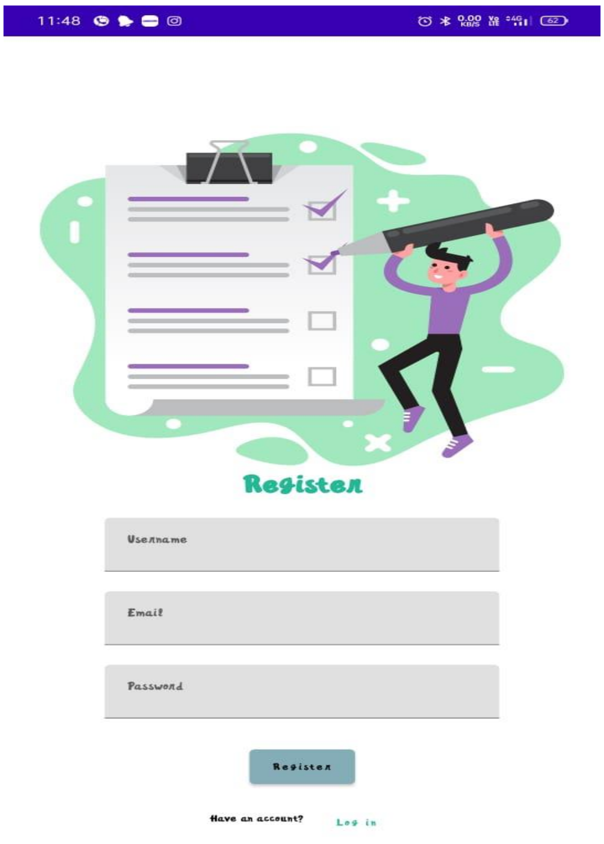
- 1.Required initial steps
- 2.Creating a new project.
- 3.Adding required dependencies.
- 4.Creating the database classes.
- 5.Building application UI and connecting to database.
- 6.Using AndroidManifest.xml
- 7.Running the application.

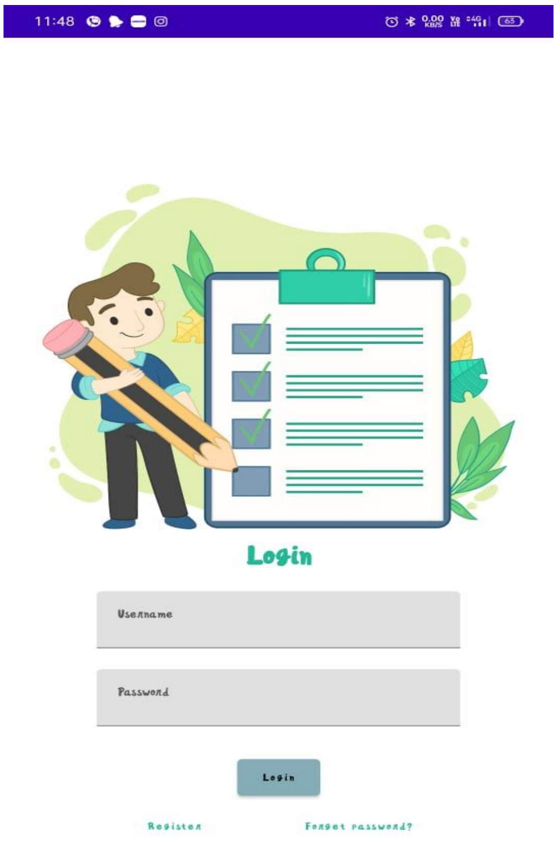
Program:

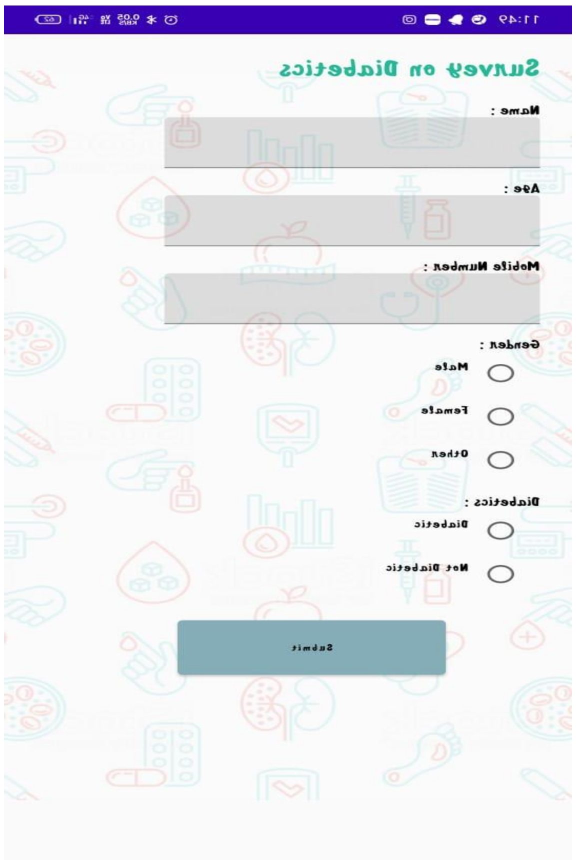
```
package com.example.surveyapplication


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


class UserDatabaseHelper(context: Context) :
    SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) {


    companion object {
        private const val DATABASE_VERSION = 1
        private const val DATABASE_NAME = "UserDatabase.db"

        private const val TABLE_NAME = "user_table"
        private const val COLUMN_ID = "id"
        private const val COLUMN_FIRST_NAME = "first_name"
        private const val COLUMN_LAST_NAME = "last_name"
        private const val COLUMN_EMAIL = "email"
        private const val COLUMN_PASSWORD = "password"
    }

    override fun onCreate(db: SQLiteDatabase?) {
        val createTable = "CREATE TABLE $TABLE_NAME (" +
                "$COLUMN_ID INTEGER PRIMARY KEY AUTOINCREMENT, " +
                "$COLUMN_FIRST_NAME TEXT, " +
                "$COLUMN_LAST_NAME TEXT, " +
                "$COLUMN_EMAIL TEXT, " +
                "$COLUMN_PASSWORD TEXT" +
                ")"

        db?.execSQL(createTable)
    }

    override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {
        db?.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")
        onCreate(db)
    }

    fun insertUser(user: User) {
        val db = writableDatabase
        val values = ContentValues()
```

```
        values.put(COLUMN_FIRST_NAME, user.firstName)
        values.put(COLUMN_LAST_NAME, user.lastName)
        values.put(COLUMN_EMAIL, user.email)
        values.put(COLUMN_PASSWORD, user.password)
        db.insert(TABLE_NAME, null, values)
        db.close()
    }

    @SuppressLint("Range")
    fun getUserByUsername(username: String): User? {
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME WHERE
$COLUMN_FIRST_NAME = ?", arrayOf(username))
        var user: User? = null
        if (cursor.moveToFirst()) {
            user = User(
                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                firstName =
cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
                lastName =
cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
                email = cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
                password =
cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
            )
        }
        cursor.close()
        db.close()
        return user
    }
    @SuppressLint("Range")
    fun getUserById(id: Int): User? {
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME WHERE
$COLUMN_ID = ?", arrayOf(id.toString()))
        var user: User? = null
        if (cursor.moveToFirst()) {
            user = User(
                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                firstName                                                    =
cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
                lastName                                                     =
cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
                email = cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
                password                                                     =
cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
            )
        }
        cursor.close()
        db.close()
        return user
    }

    @SuppressLint("Range")
    fun getAllUsers(): List<User> {
        val users = mutableListOf<User>()
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME", null)
        if (cursor.moveToFirst()) {
            do {
                val user = User(
                    id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                    firstName                                                =
cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
                    lastName                                                            =
cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
                    email = cursor.getString(cursor. getColumnIndex(COLUMN_EMAIL)),
                    password                                                            =
cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
                 )
                 users. Add(user)
              } while (cursor. moveToNext())
           }
           cursor.close()
           db. Close()
           return users
        }

     }
```

Output:




11:48
Login
Username
Password
Login
Register
Forget password?


11:49
Survey on Diabetics
Name :
Age :
Mobile Number :
Gender :
Male
Female
Other
Diabetics :
Diabetic
Not Diabetic
Submit

Conclusion:

This demonstration showcased a simple yet effective text input and validation system using Android Compose. By leveraging Compose intuitive API and built-in UI components, developers can easily create robust and user-friendly input forms.

Future scope:

low-term (Next 3-6 months)

- Implement integration with backend authentication services (e.g., Firebase Auth)
- Add advanced validation techniques (e.g., password strength checking)
- Introduce customizable validation rules and error messages

Mid-term (Next 6-12 months Short)

- Support multiple input fields (e.g., username, phone number)
- Integrate biometric authentication (e.g., fingerprint, face recognition)
- Implement dynamic validation based on user input

Long-term (Next 1-2 years)

- Improve error handling and feedback mechanisms
- Explore machine learning-based validation techniques
- Enhance security features (e.g., encryption, secure storage)