



Smart
Internz

android 

ANDROID STUDIO – EXPERIENCE BASED PROJECT LEARNING

Compose Input: A Demonstration of Text Input and Validation with Android Compose

Submitted by

SANKAR RAMAN.A	-	711022104044
SELVASURIYA.S	-	711022104046
VETRIVEL.M	-	711022104060
VINOTH.S	-	711022104063

BACHELOR OF COMPUTER SCIENCE AND ENGINEERING

IN

FIFTH SEMESTER

COMPUTER SCIENCE AND ENGINEERING

INFO INSTITUTE OF ENGINEERING, COIMBATORE – 641107

NOVEMBER/DECEMBER - 2024

BONAFIDE CERTIFICATE

Certified that this project “**Compose Input: A Demonstration of Text Input and Validation with Android Compose**” is the Bonafide work of SANKARRAMAN A (711022104044), SELVASURIYA S (711022104046), VETRIVEL M(711022104060), VINOTH S (711022104063) who carried out the project work under any supervision.

SIGNATURE

STAFF COORDINATOR

Mrs. A. SARANYA M.E.,

ASSISTANT PROFESSOR

DEPT. COMPUTER SCIENCE AND ENGINEERING
INFO INSTITUTE OF ENGINEERING,
KOVILPALAYAM COIMBATORE - 641107

SIGNATURE

HEAD OF THE DEPARTMENT

Dr. G. SELVAVINAYAGAM Ph.D.,

HEAD OF THE DEPARTMENT

DEPT. COMPUTER SCIENCE AND
ENGINEERING
INFO INSTITUTE OF ENGINEERING,
KOVILPALAYAM COIMBATORE - 641107

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We sincerely thank to Tamil Nadu Skill Development Corporation (TNSDC), Naan Mudhalvan" Platform and ANDROID STUDIO – EXPERIENCE BASED PROJECT LEARNING (EBPL) for encouragement towards our project work for providing necessary skill training.

We sincerely thank our Principal Dr. N. KOTTISWARAN, M.E., Ph.D., and Head of the Department Dr. G. SELVAVINAYAGAM, M.E., Ph.D., and also Staff Coordinator Mrs. A. SARANYA M.E for her encouragement towards our project works.

We also thank our project guide and our parents for the complete and whole hearted support, motivation guidance and help in making our project activities.

Abstract:

"Compose Input: A Demonstration of Text Input and Validation with Android Compose" presents

a practical guide to implementing text input fields and validation in Android apps using Jetpack Compose. It covers creating `TextField` components, handling state, and performing both synchronous and asynchronous input validation.

The demonstration focuses on providing real-time feedback, managing errors, and ensuring a seamless user experience with accessible and responsive forms. This article is aimed at Android developers seeking to build efficient and user-friendly input validation systems with Compose.

Introduction:

User input is a core aspect of mobile app development, and ensuring it is properly validated is crucial for a smooth user experience. This demonstration explores how to manage text input and validation in Android apps using Jetpack Compose. By leveraging Compose's declarative UI components like `TextField`, developers can easily handle input state, perform real-time validation, and provide clear error feedback. This guide highlights best practices for building responsive, accessible input forms while ensuring data integrity and improving user experience.

Data Flow Diagram:

The Data Flow Diagram (DFD) for this project represents the flow of data between the system's components. Below is a simple explanation of the flow:

DFD Breakdown:

1.Receive User Input:

- The app captures the text entered by the user.

2.Validate Input:

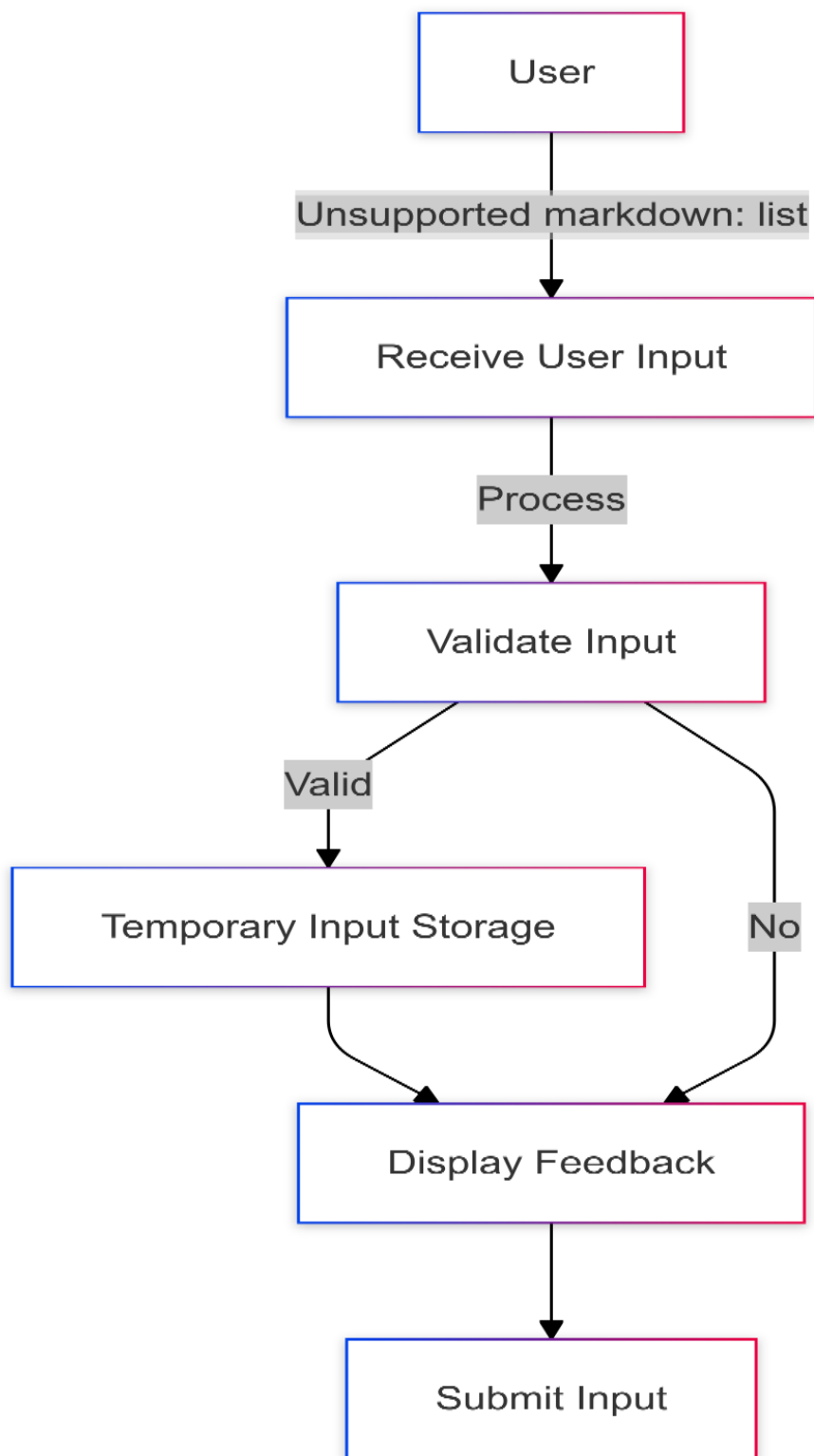
- The input is checked against predefined validation rules (e.g., non-empty, format checks).

3.Display Feedback:

- If the input fails validation, feedback like "Invalid input, please try again" is shown.
- If valid, feedback such as "Input is valid" is provided.

4.Submit Input:

- If validation passes, the input is submitted for further processing (e.g., stored in a database or sent to another system).



Use Case Diagram:

The Use Case Diagram depicts the interactions between users and the system. This helps visualize how users will interact with the TEXT INPUT VALIDATION app.

Actors:

- **User:** The person using the Android app to input and validate text.
- **System (Android App):** The backend and UI components managing the input and validation.

Main Use Cases:

1.Enter Text:

- The user enters text into a text field.

2.Validate Input:

- The system checks if the entered text meets certain validation criteria (e.g., not empty, matches a pattern).

3.Show Validation Result:

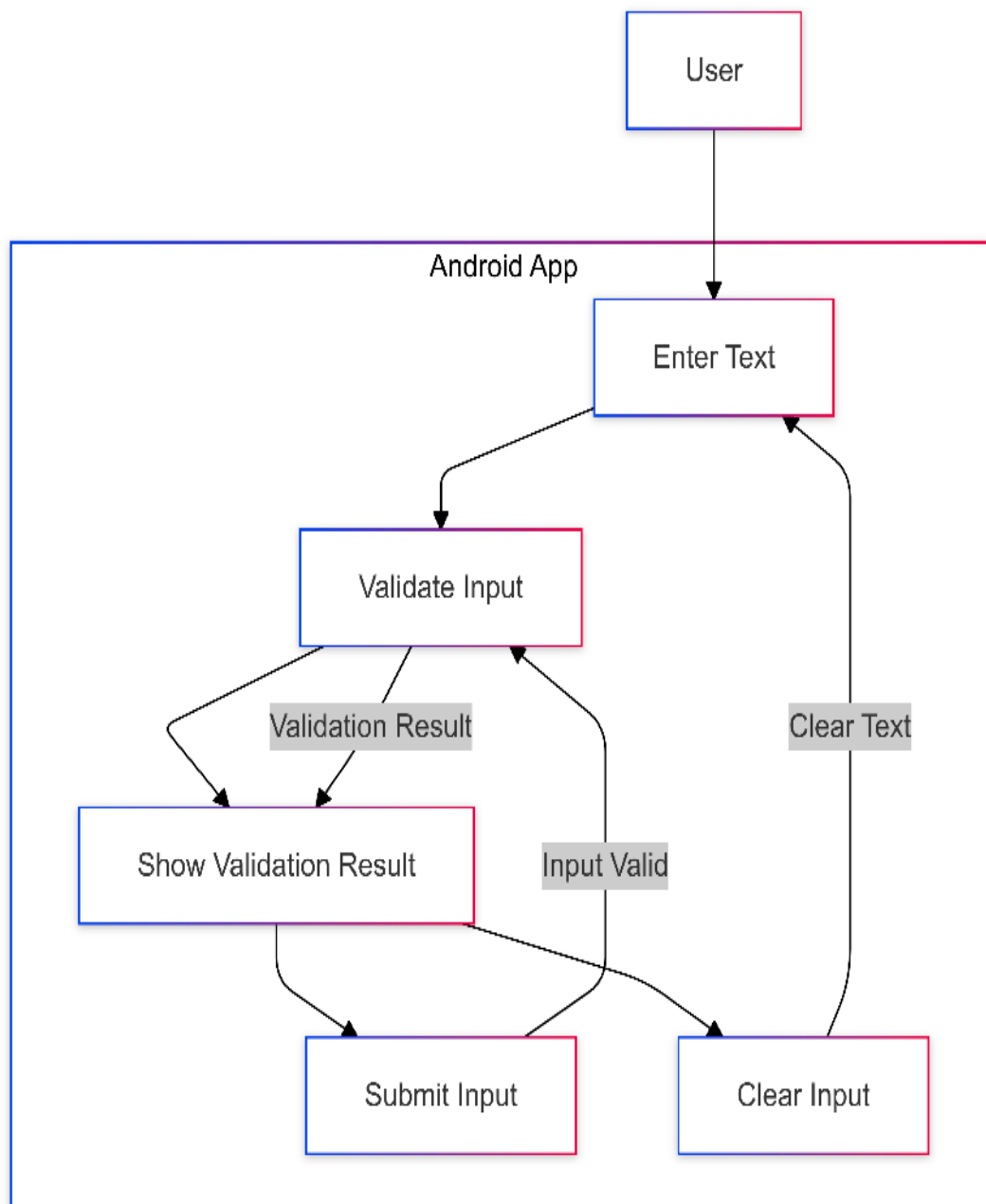
- The system provides feedback, such as an error message or confirmation that the input is valid.

4.Clear Input:

- The user clears the text input field.

5.Submit Input:

- The user submits the valid input for further processing or storage.



Hardware Requirements:

1. **Processor:** Quad-core (Intel i5+ or equivalent; i7/i9 recommended).
2. **RAM:** 8 GB minimum (16 GB+ recommended).
3. **Storage:** 20 GB+ free space (SSD recommended).
4. **Graphics:** Integrated or dedicated GPU for hardware acceleration.

Software Requirements:

1. **OS:** Windows 10+, macOS 10.14+, Linux (modern distros).
2. **Android Studio:** Version 4.2+ (Arctic Fox or later recommended).
3. **JDK:** Version 8+ (11 recommended)
4. **Libraries:** Jetpack Compose, Kotlin 1.5+, Gradle 7.0+ .
5. **Emulator:** Supports API level 21+ .
6. **Android SDK:** SDK Tools, Build Tools, Platform Tools.
7. **Version Control:** Git.

Dependencies:

- **Jetpack Compose:**
implementation "androidx.compose.ui:ui:\$compose_version"
implementation "androidx.compose.material:material:\$compose_version"
implementation "androidx.compose.ui:ui-tooling"
preview:\$compose_version"
- **Kotlin Compose:** implementation "org.jetbrains.kotlin:kotlin-stdlib:\$kotlin_version"

Program Code:

1.User:

```
package com.example.surveyapplication

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

@Entity(tableName = "user_table")
data class User(
    @PrimaryKey(autoGenerate = true) val id: Int?,
    @ColumnInfo(name = "first_name") val firstName: String?,
    @ColumnInfo(name = "last_name") val lastName: String?,
    @ColumnInfo(name = "email") val email: String?,
    @ColumnInfo(name = "password") val password: String?,
)
```

2.UserDao:

```
package com.example.surveyapplication

import androidx.room.*

@Dao
interface UserDao {

    @Query(value = "SELECT * FROM user_table WHERE email = :email")
    suspend fun getUserByEmail(email: String): User?

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertUser(user: User)
```

```

    @Update
    suspend fun updateUser(user: User)

    @Delete
    suspend fun deleteUser(user: User)
}

class User {

}

```

3.User Database:

```

package com.example.surveyapplication

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

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

    abstract fun userDao(): UserDao

    companion object {

        @Volatile
        private var instance: UserDatabase? = null

        fun getDatabase(context: Context): UserDatabase {
            return instance ?: synchronized(this) {
                val newInstance = Room.databaseBuilder(
                    context.applicationContext,
                    UserDatabase::class.java,
                    "user_database"
                ).build()
            }
        }
    }
}

```

```

        instance = newInstance
        newInstance
    }
}
}
}

```

4.User Database:

```

package com.example.surveyapplication

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

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

    abstract fun userDao(): UserDao

    companion object {

        @Volatile
        private var instance: UserDatabase? = null

        fun getDatabase(context: Context): UserDatabase {
            return instance ?: synchronized(this) {
                val newInstance = Room.databaseBuilder(
                    context.applicationContext,
                    UserDatabase::class.java,
                    "user_database"
                ).build()
                instance = newInstance
                newInstance
            }
        }
    }
}

```

Output:

1. UI Elements:

- Title: “Compose Text Input and Validation.”
- TextField: Input field labeled "Enter text."
- Submit Button: Labeled "Submit."
- Error Message: "Input must be at least 5 characters long" if input is invalid.
- Success Message: "Form submitted successfully!" appears after valid input.

2. Validation:

- Valid Input: At least 5 characters.
- Invalid Input: Shows error message and red border if input is shorter than 5 characters.

3. User Actions:

- Clicking "Submit" or pressing "Done" on the keyboard triggers validation.

4. Feedback:


- Error: Red border and error message for invalid input.
- Success: Toast saying "Input is valid!" and success message for valid input.

Sample Output:

Admin Module:

Login Page:

2:16




Login

Login

[Register](#) [Forget password?](#)

Register Page:

2:16



Register

Register

[Have an account? Log in](#)

Admin Page:

14:24

Survey Details

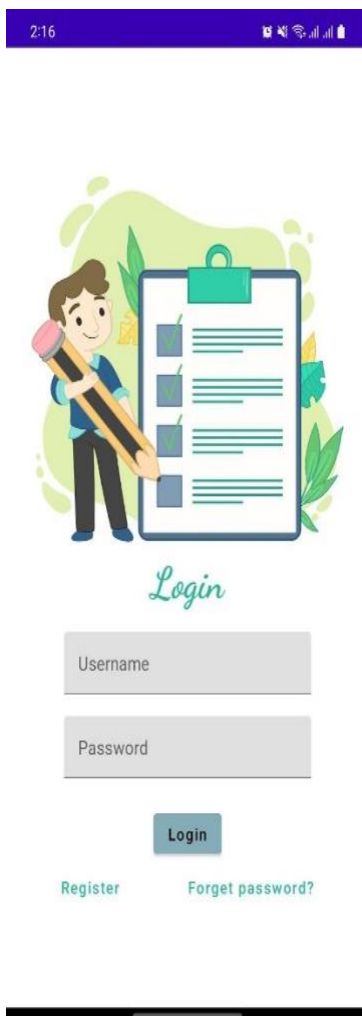
Name: Krishna
Age: 37
Mobile_Number: 6897616678
Gender: Male
Diabetics: Diabetic

Name: Pavani
Age: 23
Mobile_Number: 7167816818
Gender: Female
Diabetics: Not Diabetic


Name: Pavani
Age: 23
Mobile_Number: 7167816818
Gender: Female
Diabetics: Not Diabetic

User Module:

Login Page:



2:16



Login

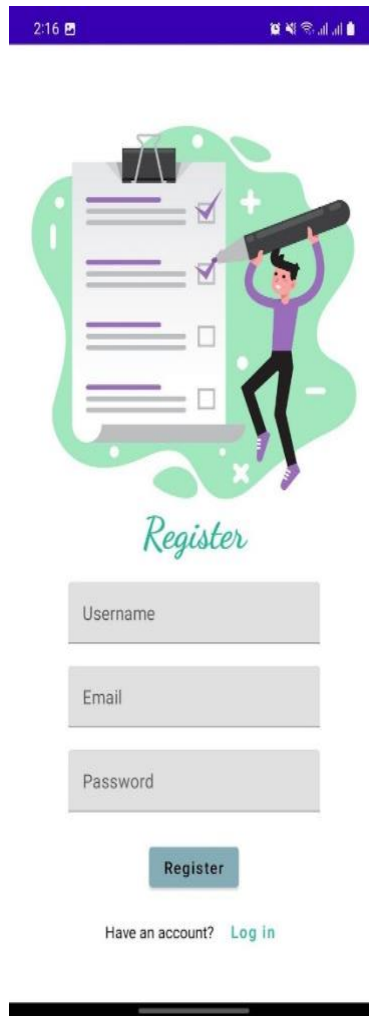
Username

Password


Login

[Register](#) [Forget password?](#)

Register Page:



2:16



Register

Username

Email

Password

Register

Have an account? [Log in](#)

Main Page:



2:16

Survey on Diabetics

Name :

Age :

Mobile Number :

Gender :

☐ Male

☐ Female

☐ Other

Diabetics :

☐ Diabetic

☐ Not Diabetic

Submit

Conclusion:

The output of this Compose UI demonstrates how text input validation can be visually represented and how the user is provided with immediate feedback upon submitting the form. The form validates the input length, shows errors when the input is invalid, and displays success messages when the input passes validation.

Future Enhancement:

1. Advanced Validation:

- Support for regex (email, phone number, password) and multiple input fields.

2. Real-time Feedback:

- Inline validation and real-time error messages.

3. Custom Error Handling:

- Tooltips, custom error messages, and icons.

4. Form Reset:

- Add "Reset" button and auto-clear fields after submission.

5. Password Handling:

- Password visibility toggle and strength meter.

6. Keyboard and Focus Management:

- Dynamic keyboard types and automatic focus management.

7. Localization & Accessibility:

- Multi-language support and improved accessibility features.

8. Network Validation:

- Server-side validation and loading states for form submission.

9. UI/UX Enhancements:

- Multi-step forms and custom input styles.