Compose Input: A Demonstration Of Text Input And Validation With Android Compose

Abstract:

This document provides a demonstration of implementing text input and validation within Android's Jetpack Compose framework. By leveraging Compose's declarative UI model, we can create a straightforward yet dynamic user interface that includes an interactive text input field. The demonstration focuses on the principles of validation within Compose, showcasing a scenario where user input is validated in real-time, and error messages are displayed based on validation criteria such as minimum character length or format restrictions. This approach highlights how Jetpack Compose enables efficient state management and responsive design, essential for creating intuitive and robust Android applications. Through this demonstration, developers can better understand how to build user-friendly forms and other text input components with enhanced validation logic in Compose.

Introduction:

In modern Android applications, user input validation is essential for providing a smooth and secure user experience. Traditionally, implementing input validation involves managing UI state, handling error messages, and responding to user interactions. With the introduction of Jetpack Compose, Android's declarative UI toolkit, creating and managing input fields with validation has become more streamlined and efficiency

SYSTEM REQUIREMENTS:

Hardware Requirements:

Processor: Intel Core i5 (or equivalent) or higher

RAM: 8 GB or higher for smooth performance, especially when running Android emulators

Storage: At least 500 MB of free disk space (for project files, Android SDK, and dependencies)

Graphics: Integrated graphics are acceptable, but dedicated graphics (e.g., NVIDIA, AMD)

will improve emulator performance

Display: Minimum resolution of 1366x768; 1920x1080 or higher recommended for ease of

Development

Software Requirements:

Operating System: Windows 10 or higher, macOS 10.13 (High Sierra) or higher, or a recent version of Linux

Java Development Kit (JDK): JDK 8 or above

Android Studio: Version 4.0 or higher (latest stable release recommended)

Android SDK: Required SDK versions for Android development, including Android SDK tools, platform tools, and Android API levels 21-30

Programming Language: Kotlin (with necessary plugins in Android Studio)

Database: Room Database (integrated within the Android projec

TOOLS:

Android Studio: Integrated Development Environment (IDE) for Android development.

Kotlin

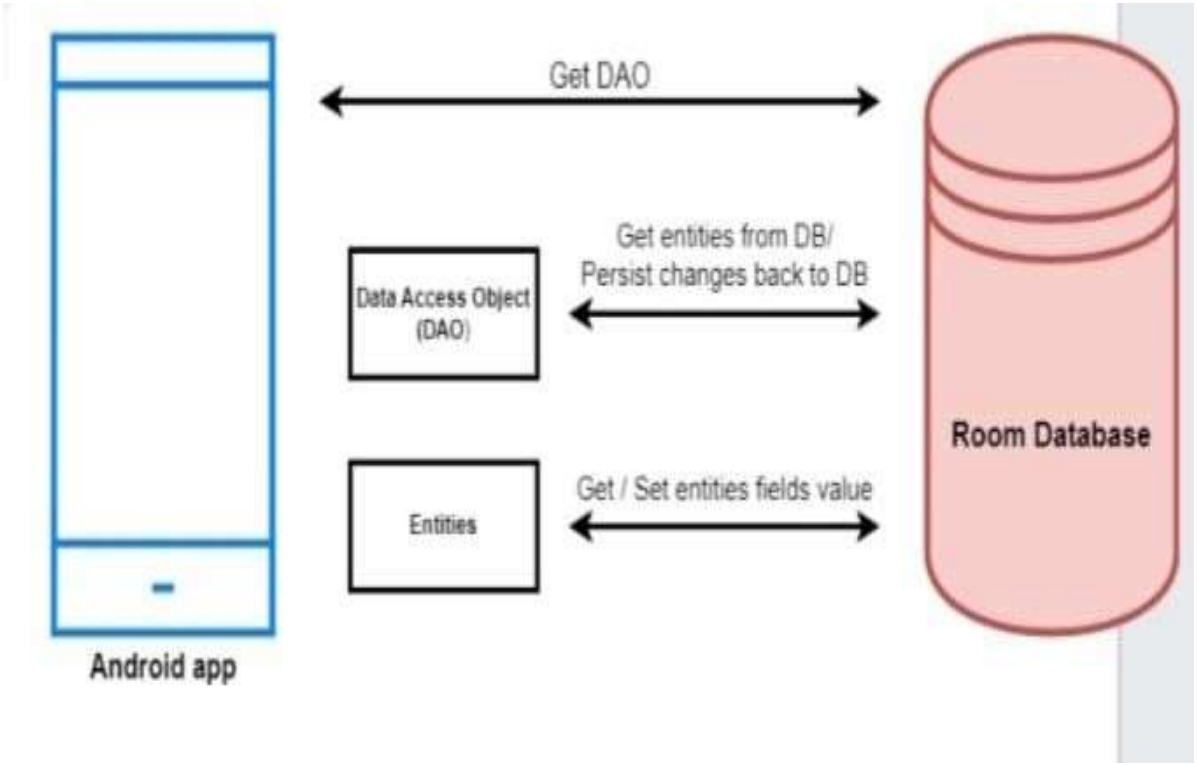
Room Database

Visions:

The vision for this project is to empower Android developers with an intuitive and efficient approach to handling user input and validation using Jetpack Compose. By leveraging Compose's declarative, state-driven architecture, this project aims to simplify the development of responsive and user-friendly interfaces that handle validation seamlessly, ultimately enhancing user experience and application quality.

.

Architecture:



CODE IMPLEMENTATION (SAMPLE CODE):

Admin activity

```
package com.example.surveyapplication


import android.os.Bundle
import android.util.Log
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
```

```
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.LazyRow
import androidx.compose.foundation.lazy.items
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.example.surveyapplication.ui.theme.SurveyApplicationTheme

class AdminActivity : ComponentActivity() {
    private lateinit var databaseHelper: SurveyDatabaseHelper
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        databaseHelper = SurveyDatabaseHelper(this)
        setContent {
```

```
            val data = databaseHelper.getAllSurveys();
            Log.d("swathi", data.toString())
            val survey = databaseHelper.getAllSurveys()
            ListListScopeSample(survey)
        }
    }
}
@Composable
fun ListListScopeSample(survey: List<Survey>) {

    Image(
        painterResource(id = R.drawable.background), contentDescription = "",
        alpha =0.1F,
        contentScale = ContentScale.FillHeight,
        modifier = Modifier.padding(top = 40.dp)
    )

    Text(
        text = "Survey Details",
        modifier = Modifier.padding(top = 24.dp, start = 106.dp, bottom = 24.dp),
        fontSize = 30.sp,
        color = Color(0xFF25b897)
    )
```

```
Spacer(modifier = Modifier.height(30.dp))
LazyRow(
    modifier = Modifier
        .fillMaxSize()
        .padding(top = 80.dp),

    horizontalArrangement = Arrangement.SpaceBetween
) {
    item {

        LazyColumn {
            items(survey) { survey ->
                Column(
                    modifier = Modifier.padding(
                        top = 16.dp,
                        start = 48.dp,
                        bottom = 20.dp
                    )
                ) {
                    Text("Name: ${survey.name}")
                    Text("Age: ${survey.age}")
                    Text("Mobile_Number: ${survey.mobileNumber}")
                    Text("Gender: ${survey.gender}")
```

```
                    Text("Diabetics: ${survey.diabetics}")
                }
            }
        }
    }
}
```

Login activity

```
Intent(
                        context,
                        AdminActivity::class.java
                    )
                )
            }
            else {
                error =  "Invalid username or password"
            }

        } else {
            error = "Please fill all fields"
        }
    },
    colors = ButtonDefaults.buttonColors(backgroundColor = Color(0xFF84adb8)),
```

```
            modifier = Modifier.padding(top = 16.dp)
        ) {
            Text(text = "Login")
        }
        Row {
            TextButton(onClick = {context.startActivity(
                Intent(
                    context,
                    RegisterActivity::class.java
                )
            )}
            )
            { Text(color = Color(0xFF25b897),text = "Register") }
            TextButton(onClick = {
            })


            {
                Spacer(modifier = Modifier.width(60.dp))
                Text(color = Color(0xFF25b897),text = "Forget password?")
            }
        }
    }
}
```

```
private fun startMainPage(context: Context) {
    val intent = Intent(context, MainActivity::class.java)
    ContextCompat.startActivity(context, intent, null)
}
```

Main activity

```
class MainActivity : ComponentActivity() {
    private lateinit var databaseHelper: SurveyDatabaseHelper
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        databaseHelper = SurveyDatabaseHelper(this)
        setContent {
            FormScreen(this, databaseHelper)
        }
    }
}

@Composable
fun FormScreen(context: Context, databaseHelper: SurveyDatabaseHelper) {

    Image(
        painterResource(id = R.drawable.background), contentDescription = "",
        alpha =0.1F,
        contentScale = ContentScale.FillHeight,
        modifier = Modifier.padding(top = 40.dp)
```

```
    )

// Define state for form fields
var name by remember { mutableStateOf("") }
var age by remember { mutableStateOf("") }
var mobileNumber by remember { mutableStateOf("") }
var genderOptions = listOf("Male", "Female", "Other")
var selectedGender by remember { mutableStateOf("") }
var error by remember { mutableStateOf("") }
var diabeticsOptions = listOf("Diabetic", "Not Diabetic")
var selectedDiabetics by remember { mutableStateOf("") }

Column(
    modifier = Modifier.padding(24.dp),
    horizontalAlignment = Alignment.Start,
    verticalArrangement = Arrangement.SpaceEvenly
) {

    Text(
        fontSize = 36.sp,
        textAlign = TextAlign.Center,
        text = "Survey on Diabetics",
        color = Color(0xFF25b897)
    )

    Spacer(modifier = Modifier.height(24.dp))
```

```
Text(text = "Name :", fontSize = 20.sp)
TextField(
    value = name,
    onValueChange = { name = it },
)

Spacer(modifier = Modifier.height(14.dp))

Text(text = "Age :", fontSize = 20.sp)
TextField(
    value = age,
    onValueChange = { age = it },
)

Spacer(modifier = Modifier.height(14.dp))

Text(text = "Mobile Number :", fontSize = 20.sp)
TextField(
    value = mobileNumber,
    onValueChange = { mobileNumber = it },
)

Spacer(modifier = Modifier.height(14.dp))

Text(text = "Gender :", fontSize = 20.sp)
RadioGroup(
    options = genderOptions,
```

```
        selectedOption = selectedGender,
        onSelectedChange = { selectedGender = it }
    )

    Spacer(modifier = Modifier.height(14.dp))

    Text(text = "Diabetics :", fontSize = 20.sp)
    RadioGroup(
        options = diabeticsOptions,
        selectedOption = selectedDiabetics,
        onSelectedChange = { selectedDiabetics = it }
    )

    Text(
        text = error,
        textAlign = TextAlign.Center,
        modifier = Modifier.padding(bottom = 16.dp)
    )
    // Display Submit button
    Button(
        onClick = { if (name.isNotEmpty() && age.isNotEmpty() &&
mobileNumber.isNotEmpty() && genderOptions.isNotEmpty() &&
diabeticsOptions.isNotEmpty()) {
            val survey = Survey(
                id = null,
                name = name,
                age = age,
                mobileNumber = mobileNumber,
```

```
                gender = selectedGender,
                diabetics = selectedDiabetics
            )
            databaseHelper.insertSurvey(survey)
            error = "Survey Completed"

        } else {
            error = "Please fill all fields"
        }
        },
        colors = ButtonDefaults.buttonColors(backgroundColor = Color(0xFF84adb8)),
        modifier = Modifier.padding(start = 70.dp).size(height = 60.dp, width = 200.dp)
    ) {
        Text(text = "Submit")
    }
  }
}
@Composable
fun RadioGroup(
  options: List<String>,
  selectedOption: String?,
  onSelectedChange: (String) -> Unit
) {
  Column {
    options.forEach { option ->
      Row(
        Modifier
```

```
                .fillMaxWidth()
                .padding(horizontal = 5.dp)
        ) {
            RadioButton(
                selected = option == selectedOption,
                onClick = { onSelectedChange(option) }
            )
            Text(
                text = option,
                style = MaterialTheme.typography.body1.merge(),
                modifier = Modifier.padding(top = 10.dp),
                fontSize = 17.sp
            )
        }
    }
  }
}
```

Register activity

```
class RegisterActivity : ComponentActivity() {
   private lateinit var databaseHelper: UserDatabaseHelper
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      databaseHelper = UserDatabaseHelper(this)
      setContent {
```

```
                RegistrationScreen(this,databaseHelper)

        }
    }
}

@Composable
fun RegistrationScreen(context: Context, databaseHelper: UserDatabaseHelper) {

    var username by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var email by remember { mutableStateOf("") }
    var error by remember { mutableStateOf("") }

    Column(
        modifier = Modifier.fillMaxSize().background(Color.White),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {

        Image(painterResource(id = R.drawable.survey_signup), contentDescription = "")

        Text(
            fontSize = 36.sp,
            fontWeight = FontWeight.ExtraBold,
            fontFamily = FontFamily.Cursive,
```

```
    color = Color(0xFF25b897),
    text = "Register"
)

Spacer(modifier = Modifier.height(10.dp))
TextField(
    value = username,
    onValueChange = { username = it },
    label = { Text("Username") },
    modifier = Modifier
        .padding(10.dp)
        .width(280.dp)

)

TextField(
    value = email,
    onValueChange = { email = it },
    label = { Text("Email") },
    modifier = Modifier
        .padding(10.dp)
        .width(280.dp)
)

TextField(
    value = password,
    onValueChange = { password = it },
```

```
    label = { Text("Password") },
    visualTransformation = PasswordVisualTransformation(),
    modifier = Modifier
        .padding(10.dp)
        .width(280.dp)
)



if (error.isNotEmpty()) {
    Text(
        text = error,
        color = MaterialTheme.colors.error,
        modifier = Modifier.padding(vertical = 16.dp)
    )
}

Button(
    onClick = {
        if (username.isNotEmpty() && password.isNotEmpty() && email.isNotEmpty()) {
            val user = User(
                id = null,
                firstName = username,
                lastName = null,
                email = email,
                password = password
            )
            databaseHelper.insertUser(user)
```

```
            error = "User registered successfully"
            // Start LoginActivity using the current context
            context.startActivity(
                Intent(
                    context,
                    LoginActivity::class.java
                )
            )

        } else {
            error = "Please fill all fields"
        }
    },
    colors = ButtonDefaults.buttonColors(backgroundColor = Color(0xFF84adb8)),
    modifier = Modifier.padding(top = 16.dp),

) {
    Text(text = "Register")
}
Spacer(modifier = Modifier.width(10.dp))
Spacer(modifier = Modifier.height(10.dp))

Row() {
    Text(
        modifier = Modifier.padding(top = 14.dp), text = "Have an account?"
    )
    TextButton(onClick = {
```

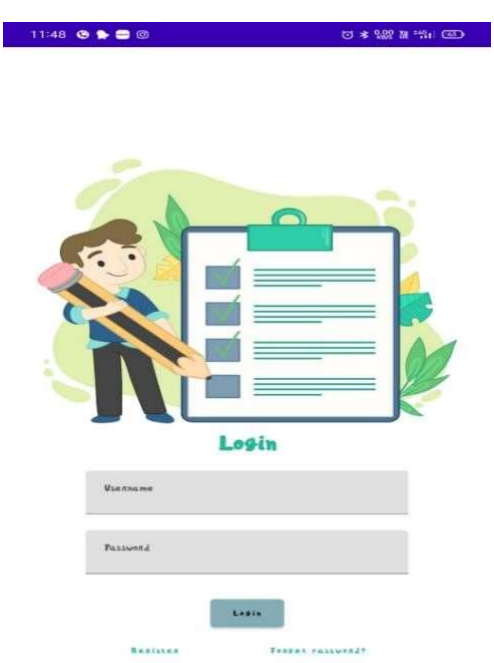
```
                context.startActivity(
                    Intent(
                        context,
                        LoginActivity::class.java
                    )
                )
            })

            {
                Spacer(modifier = Modifier.width(10.dp))
                Text( color = Color(0xFF25b897),text = "Log in")
            }
        }
    }
}
private fun startLoginActivity(context: Context) {
    val intent = Intent(context, LoginActivity::class.java)
    ContextCompat.startActivity(context, intent, null)
}
```

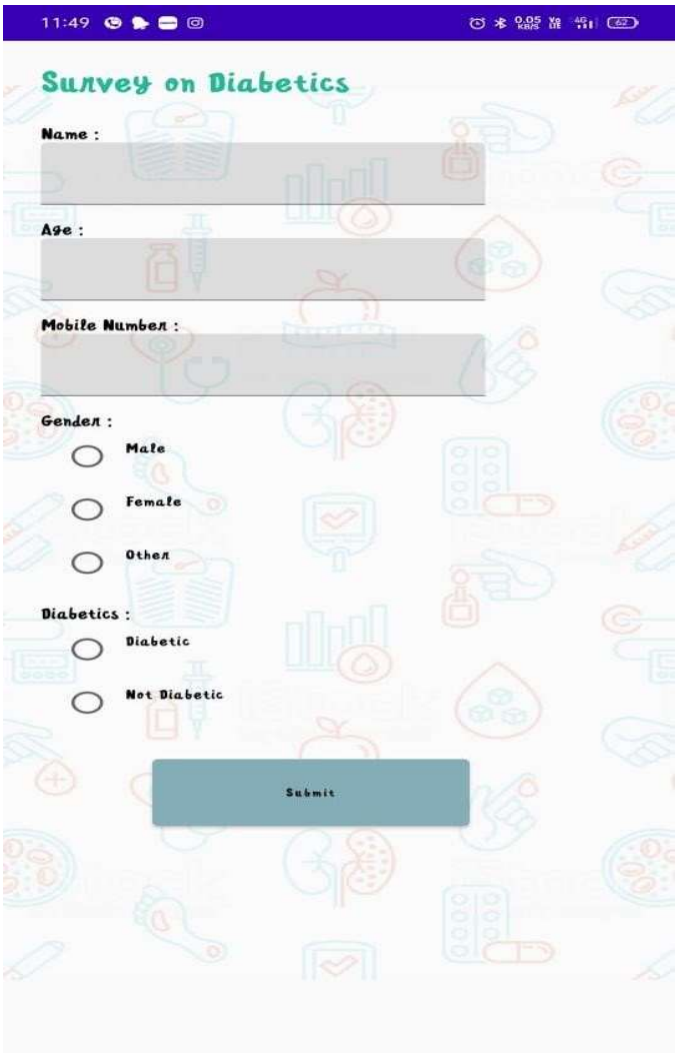
Survey.kt

```
package com.example.surveyapplication

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

@Entity(tableName = "survey_table")
```

```
data class Survey(
    @PrimaryKey(autoGenerate = true) val id: Int?,
    @ColumnInfo(name = "name") val name: String?,
    @ColumnInfo(name = "age") val age: String?,
    @ColumnInfo(name = "mobile_number") val mobileNumber: String?,
    @ColumnInfo(name = "gender") val gender: String?,
    @ColumnInfo(name = "diabetics") val diabetics: String?,
```

Output:





Challenges:

Managing real-time input validation and error feedback increases code complexity.

Syncing UI state with data in traditional Android requires extensive boilerplate code.

Handling multiple fields and validation rules can make code harder to maintain.

Solution:

Jetpack Compose's declarative, state-driven approach simplifies UI updates and validation.

State management in Compose enables real-time feedback with minimal code.

Conditional rendering allows error messages to display dynamically as user types.

Reduced boilerplate improves readability, customization, and maintainability.

Conclusion:

This demonstration showcases how Jetpack Compose simplifies handling text input and validation in Android applications. By leveraging Compose's declarative approach and efficient state management, developers can build responsive, user-friendly input fields with real-time validation. This approach reduces code complexity, enhances readability, and improves the user experience by providing immediate feedback and clear guidance for correcting input errors.

Future enhancements

- **Advanced Validation Rules**: Add more complex validation (e.g., regex patterns for email, password strength indicators).
- **Multiple Field Validation**: Implement forms with multiple fields and cross-field validation (e.g., matching passwords).
- **Error Styling**: Customize error messages with animations, icons, or different styles for better user experience.

Localization and Accessibility: Support multiple languages and screen readers to make inputs accessible to all users