ANDROID APPLICATION DEVELOPMENT

A demonstration of text input and validation with android compose

Team ID-NM2024TMID05772

Submitted by

Kalaiyarasan S(Team Leader)	-	6B91679C74AC430802F419527E898025
Rahul R(Team Member)	-	B99F1A099AB5BA791834A01FC236EE40
Chandrasekar A(Team Member)	-	29AD5F41A9D9B2AB8AA40400FDA7FB2B
Aadhithya R(Team Member)	-	DC1491FF9457D74B8454367D7AD93AE6

SEMESTER -V
B.E COMPUTER SCIENCE AND ENGINEERING
ACADEMIC YEAR- 2024-2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
ANNA UNIVERSITY REGIONAL CAMPUS COIMBATORE
COIMBATORE - 641046
NOVEMBER 2024

List Of Contents

A demonstration of text input and validation with android compose

1. Abstraction

In a text input and validation setup using Jetpack Compose, abstraction is essential to keep the code modular and reusable. We can abstract the functionality by creating a dedicated composable that encapsulates the input field, validation logic, and error handling. This composable would accept parameters such as validation rules and error messages, allowing the same component to be reused for different input requirements. This abstraction makes the design cleaner, allowing developers to focus on the specific validation logic and user interface elements separately.

2. Hardware and Software Use

Software: Jetpack Compose library, which simplifies UI construction and state management.

Hardware: Android device, handling user input with a touchscreen and displaying validation feedback

3.Description:

This project demonstrates the implementation of a user input field with validation using Android's Jetpack Compose. The goal is to create a modular, reusable, and interactive text input component that performs real-time validation and provides immediate feedback to users. Jetpack Compose, a modern UI toolkit from Android, enables developers to build native Android apps with a declarative approach, making the UI code more concise and intuitive.

Project Objectives:

- **Demonstrate Abstraction**: Separate UI and validation logic to enhance code readability and reusability.
- **Showcase Real-Time Feedback**: Use Compose's state management to deliver responsive feedback based on user input.
- **Highlight Hardware and Software Integration**: Leverage the touchscreen for data input and Compose libraries for managing UI states, enabling an efficient and interactive user interface.

Functionality

1. **Custom Validation Rules**
   Allow developers to pass a lambda or a list of validation rules to the composable. This would support complex and dynamic validation scenarios, such as ensuring a password meets strength requirements or verifying an email format.
2. **Support for Multiple Input Types**
   Expand the composable to handle different types of inputs (e.g., text, numeric, email, password). This can be achieved by accepting parameters such as keyboardType and visualTransformation.
3. **Error Styling and Animation**
   Add customizable error message styling and animations to improve the user experience. For example, a shaking animation or a color transition when validation fails.
4. **Success Indicators**
   Provide optional feedback for successful validation, such as a checkmark icon or a subtle green highlight.
5. **Accessibility Improvements**
   Integrate accessibility features like screen reader support and descriptive labels for input fields to make the application inclusive.
6. **State Preservation**
   Ensure that the composable can handle state preservation across configuration changes, such as screen rotation, by using rememberSaveable.
7. **Testing and Debugging Tools**
   Implement unit tests and UI tests for the input composable to validate its functionality under different scenarios.

4.Source code:

```
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}")
                    }
                }
            }
        }
    }
}
package com.example.surveyapplication
import android.content.Context
import android.content.Intent
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.runtime.*

import androidx.compose.ui.Alignment

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.text.font.FontFamily

import androidx.compose.ui.text.font.FontWeight

import androidx.compose.ui.text.input.PasswordVisualTransformation

import androidx.compose.ui.tooling.preview.Preview

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import androidx.core.content.ContextCompat

import com.example.surveyapplication.ui.theme.SurveyApplicationTheme


class LoginActivity : ComponentActivity() {

    private lateinit var databaseHelper: UserDatabaseHelper

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)

        databaseHelper = UserDatabaseHelper(this)

        setContent {

                LoginScreen(this, databaseHelper)

        }

    }

}

@Composable
```

```
fun LoginScreen(context: Context, databaseHelper: UserDatabaseHelper) {
    var username by remember { mutableStateOf("") }
    var password 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_login), contentDescription = "")
        Text(
            fontSize = 36.sp,
            fontWeight = FontWeight.ExtraBold,
            fontFamily = FontFamily.Cursive,
            color = Color(0xFF25b897),
            text = "Login"
        )
        Spacer(modifier = Modifier.height(10.dp))
        TextField(
            value = username,
            onValueChange = { username = it },
            label = { Text("Username") },
            modifier = Modifier
                .padding(10.dp)
                .width(280.dp)
        )
```

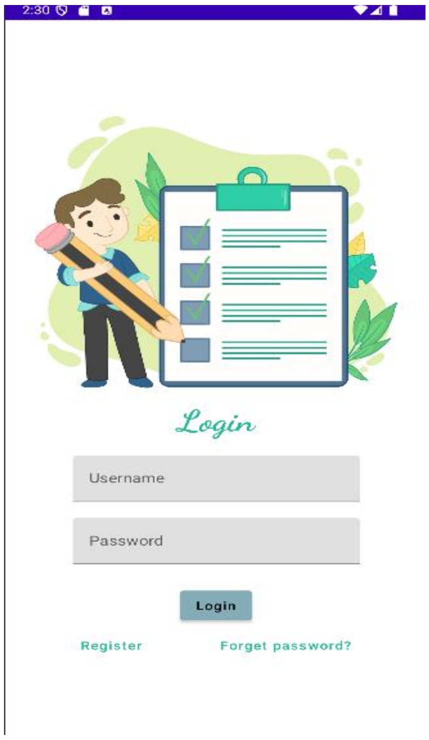
```
TextField(
    value = password,
    onValueChange = { password = it },
    label = { Text("Password") },
    visualTransformation = PasswordVisualTransformation(),
    modifier = Modifier
        .padding(10.dp)
        .width(280.dp)
)
if (error.isNotEmpty()) {
    Text(
        text = error,
        color = MaterialTheme.colors.error,
        modifier = Modifier.padding(vertical = 16.dp)
    )
}
Button(
    onClick = {
        if (username.isNotEmpty() && password.isNotEmpty()) {
            val user = databaseHelper.getUserByUsername(username)
            if (user != null && user.password == password) {
                error = "Successfully log in"
                context.startActivity(
                    Intent(
                        context,
                        MainActivity::class.java
                    )
                )
                //onLoginSuccess()
            }
            if (user != null && user.password == "admin") {
                error = "Successfully log in"
                context.startActivity(
                    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 {
```

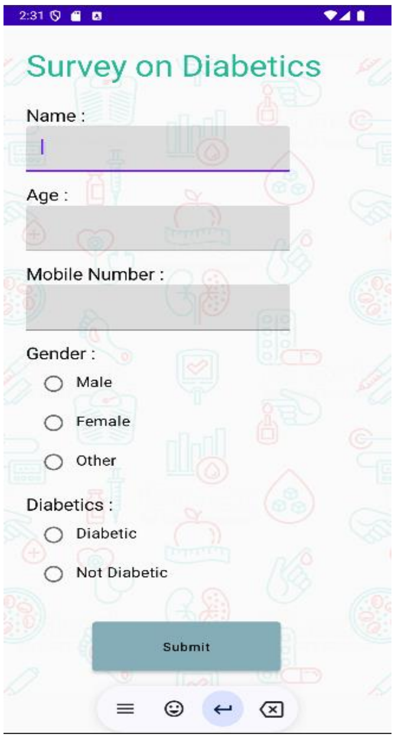
```
            TextButton(onClick = {context.startActivity(
                Intent(
                    context,
                    RegisterActivity::class.java
                )
            )}
            )
            { Text(color = Color(0xFF25b897),text = "Register") }
            TextButton(onClick = {
            })
            {
                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)
}
```

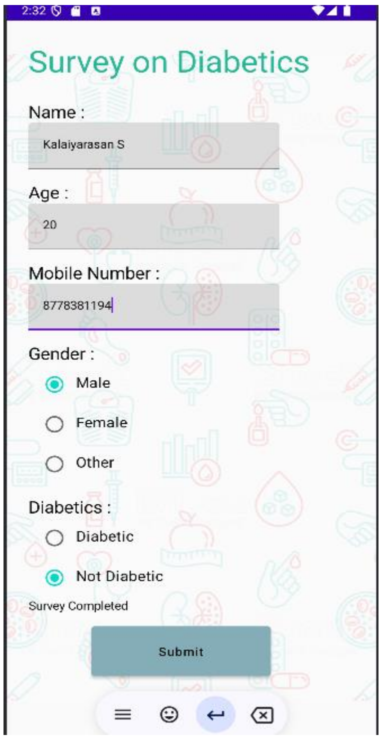
Video Demo Link:

https://drive.google.com/file/d/1Vrg-3IsRfYCmwQVKvCw_yqETjLJYnBAa/view?usp=drive_link

5.Output:







Name : Kalaiyarasan S

Age : 20

Mobile Number : 8778381194

Gender : [✓ Male] Female Other

Diabetics : Diabetic [✓ Not Diabetic]

Survey Completed

6. Conclusion

In conclusion, Jetpack Compose's declarative approach offers a modern, streamlined way to handle text input and validation. With its clean syntax and state management capabilities, developers can build responsive and interactive components that simplify error handling and input validation. This approach not only enhances the development experience by reducing boilerplate code but also creates a better user experience with immediate feedback. Through abstraction and efficient use of hardware and software, Compose transforms how we design and implement input-driven interfaces in Android.