TP05: Add interactive components to mobile App

1. Before we begin

In this **PRACTICE**, we learn how to add Buttons and TextFields to our app.

2. What we'll learn

- ⊡ How to add **Buttons** with different shapes to our Android app.
- ⊡ How to add **TextFields** with their different keyboard types to our Android app.

3. Buttons

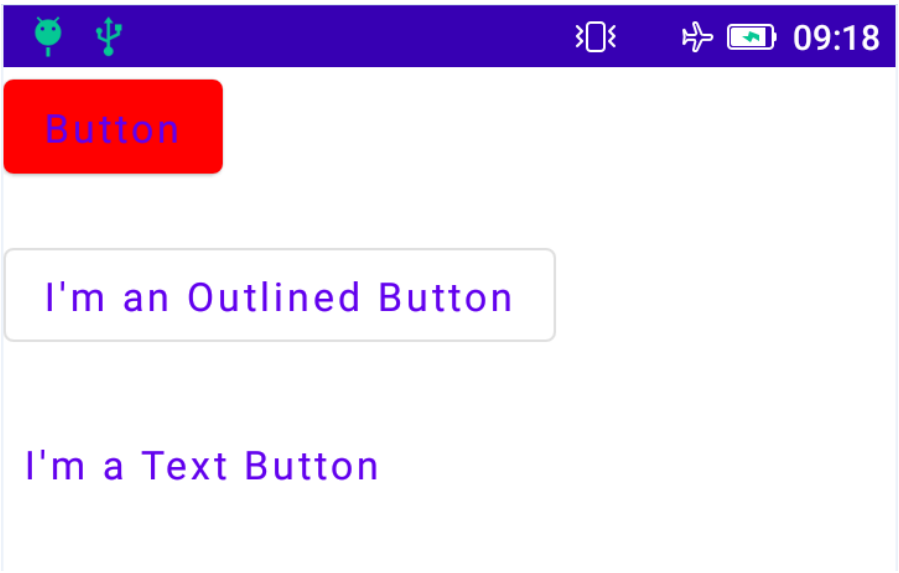
A Button is a UI component in Android which is used to navigate between different screens. With the help of a button, the user can interact with the app and perform multiple actions inside it[1].

A Button has a onClick-Function. We can add a Text-Composable or any other Composables as child elements of the Button[2].

3.1 Example of Simple, Outlined and Text Buttons:

```
@Composable
fun ButtonExample() {
  Column{ this: ColumnScope
    Button(onClick = { /* Do something! */ }, colors =
    ButtonDefaults.textButtonColors( backgroundColor = Color.Red )) { this: RowScope
        Text( text: "Button")
    }
    Spacer(Modifier.size(20.dp))
    OutlinedButton(onClick = { /* Do something! */ }) { this: RowScope
        Text( text: "I'm an Outlined Button")

    }
    Spacer(Modifier.size(20.dp))
    TextButton(onClick = { /* Do something! */ }) { this: RowScope
        Text( text: "I'm a Text Button")
    }
}}
```

[1] https://www.geeksforgeeks.org/button-in-android-using-jetpack-compose/
[2] https://foso.github.io/Jetpack-Compose-Playground/material/button/



3.2 Button Style (Different Corner Sizes)

Here is an example with three decorated buttons:

Clicking on these buttons is illustrated below:

For implementing these buttons we should create 03 functions, one by button, and a main function called: **`ButtonStyle()`** function this last calls the three buttons functions that are **`ButtonCount()`**implimenting the 1st button, **`ButtonDisable()`**implimenting the 2nd button, and **`ButtonNoRipple()`**implimenting the 3rd button:

```
@Composable
fun ButtonStyle() {
    Column(
        modifier = Modifier.fillMaxSize(),
        // Gap between children = 32.dp
        verticalArrangement = Arrangement.spacedBy(32.dp, alignment = Alignment.CenterVertically),
        horizontalAlignment = Alignment.CenterHorizontally
    ) { this: ColumnScope

        val gradientColors = listOf(Color( color: 0xFF7b4397), Color( color: 0xFFdc2430))
        val roundCornerShape = RoundedCornerShape(topEnd = 30.dp, bottomStart = 30.dp)

        ButtonCount(
            gradientColors = gradientColors,
            roundedCornerShape = roundCornerShape
        )

        ButtonDisable(
            gradientColors = gradientColors,
            roundedCornerShape = roundCornerShape
        )

        ButtonNoRipple(
            gradientColors = gradientColors,
            roundedCornerShape = roundCornerShape
        )
    }
}
```

```
@Composable
fun ButtonCount(
    gradientColors: List<Color>,
    roundedCornerShape: RoundedCornerShape
) {
    var clickCount by remember {  mutableStateOf( value: 0)    }

    Box(
        modifier = Modifier
            .background(
                brush = Brush.horizontalGradient(colors = gradientColors),
                shape = roundedCornerShape
            )
            .clip(roundedCornerShape)
            .clickable {
                clickCount++
            }
            .padding(PaddingValues(horizontal = 60.dp, vertical = 16.dp)),
        contentAlignment = Alignment.Center
    ) { this: BoxScope
        Text(
            text = "Click $clickCount",
            fontSize = 26.sp,
            color = Color.White,
            fontWeight = FontWeight.Bold
        )
    }
}
```

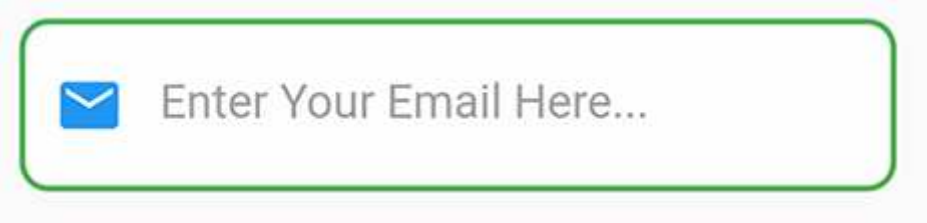
```
@Composable
fun ButtonDisable(
    gradientColors: List<Color>,
    roundedCornerShape: RoundedCornerShape,
    disabledColors: List<Color> = listOf( Color.Gray.copy(alpha = 0.2f),Color.Gray.copy(alpha = 0.2f)
    )) {
    var enabled by remember { mutableStateOf( value: true)  }

    Box(
        modifier = Modifier
            .background(
                brush = Brush.horizontalGradient(colors = if (enabled) gradientColors else disabledColors),
                shape = roundedCornerShape
            )
            .clip(roundedCornerShape)
            .clickable(enabled = enabled) {
                enabled = false
            }
            .padding(PaddingValues(horizontal = 40.dp, vertical = 16.dp)),
        contentAlignment = Alignment.Center
    ) { this: BoxScope
        Text(
            text = if (enabled) "Disable Me!" else "I'm Disabled!",
            fontSize = 26.sp,
            color = if (enabled) Color.White else Color.Black.copy(alpha = 0.4f),
            fontWeight = FontWeight.Bold
        )
    }
}
```

To run this app with three buttons we should call the function **`ButtonStyle()`** in the MainActivity Class (inside the **`setContent()`** function).

```
@Composable
fun ButtonNoRipple(
    gradientColors: List<Color>,
    roundedCornerShape: RoundedCornerShape,
    context: Context = LocalContext.current.applicationContext) {
    // To disable ripple effect
    val interactionSource = MutableInteractionSource()
    Box(
        modifier = Modifier
            .background(
                brush = Brush.horizontalGradient(colors = gradientColors),
                shape = roundedCornerShape
            )
            .clip(roundedCornerShape)
            .clickable(indication = null, interactionSource = interactionSource) {
                Toast
                    .makeText(context, text: "No Ripple", Toast.LENGTH_SHORT)
                    .show()
            }
            .padding(PaddingValues(horizontal = 46.dp, vertical = 16.dp)),
        contentAlignment = Alignment.Center
    ) { this: BoxScope
        Text( text = "No Ripple",
            fontSize = 26.sp,
            color = Color.White,
            fontWeight = FontWeight.Bold
        )
    }
}
```

4. TextFields

A TextField is a UI element that lets users type in text as an input. This input can then be stored and used for various desired functions. In General, there is no hint for a TextField. However, we can customize TextField to display hints to the user[3].



[3] https://www.geeksforgeeks.org/textfield-with-hint-text-in-android-using-jetpack-compose/

TextField is a user interface control that is used to allow the user to enter the text. This widget is used to get the data from the user as numbers or text[4].

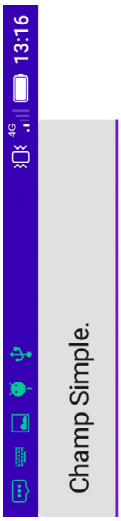
1. Simple TextField

```
@Composable
fun SimpleTextField() {
    var text by remember { mutableStateOf(TextFieldValue( text: "")) }
    Column() { this: ColumnScope

    TextField(
        value = text,
        onValueChange = { it: TextFieldValue
            text = it
        }
    )
}}
```

In this example, we created a variable text, it's **mutableState TextFieldValue**. **mutableState** - It return an observable value for Compose. If value changed UI get changed automatically. **TextFieldValue** - A class holding information about the editing state.

In TextField() function we use two arguments, **value** & **onValueChange**. **value** - We need to set the TextFieldValue. We created a variable (text) for this. And we assigned text to this argument.

onValueChange - It will return new value (TextFieldValue) when user enter the text. We assign the **newText** to text**,** then only user entered text will set into TextField. It creates a simple text box. But it doesn't have any labels. UI is also not so good:

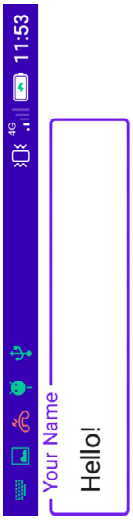


[4] https://www.jetpackcompose.net/textfield-in-jetpack-compose

2. OutLined TextField with Label and PlaceHolder

```
@Composable
fun LabelAndPlaceHolder() {
    var text by remember { mutableStateOf(TextFieldValue( text: "")) }
    Column{ this: ColumnScope
    OutlinedTextField(
        value = text,
        onValueChange = { it: TextFieldValue
            text = it
        },
        label = { Text(text = "Your Name") },
        placeholder = { Text(text = "Your Placeholder/Hint") },
    )
}}
```

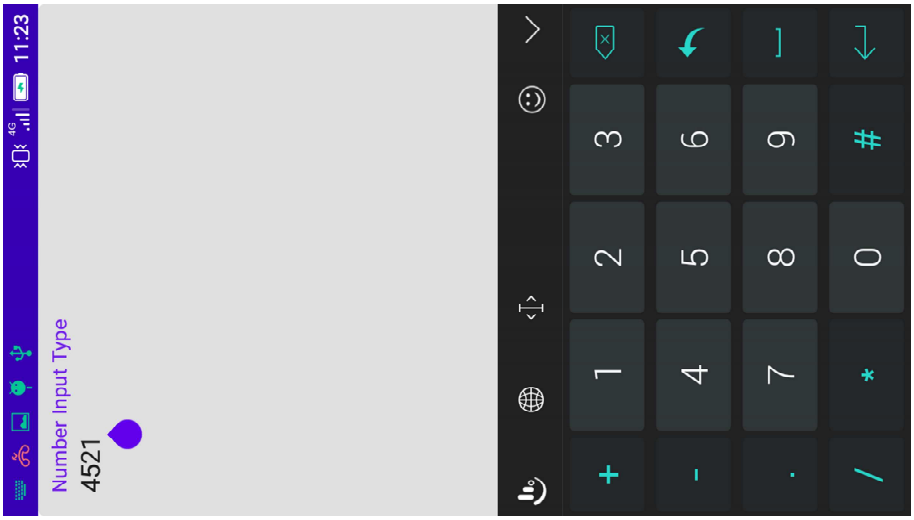
label - If the textfield has focus then label will be floated to the top of the TextField. placeholder - It displays descriptive text within the box when TextField is empty:



3. Keyboard Options

To get the number input from TextField , use keyboardOptions. We use number keyboard type. So it accepts only number input from the user.

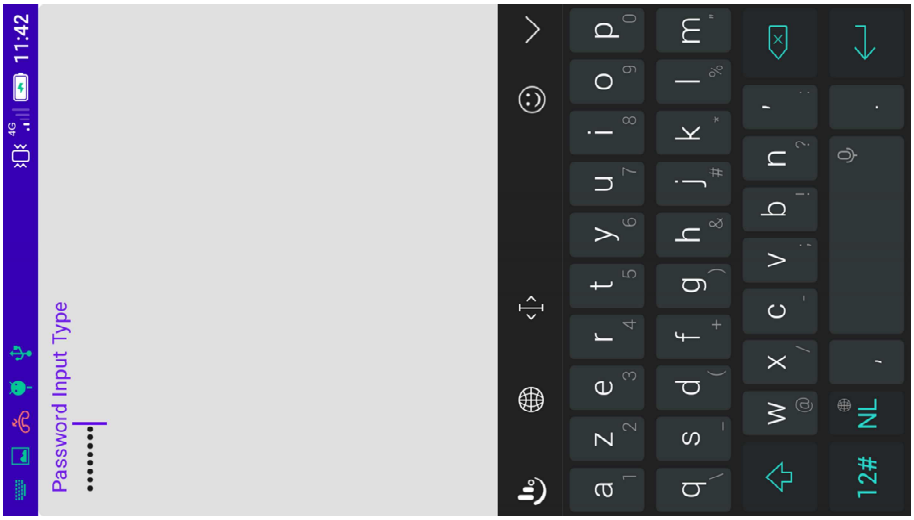
```
@Composable
fun TextFieldNumbers() {
    var text by remember { mutableStateOf(TextFieldValue( text: "")) }
    TextField(
        value = text,
        label = { Text(text = "Number Input Type") },
        keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),
        onValueChange = { it ->
            text = it
        }
    )
}
```



Following Keyboard Types available in Compose:

- KeyboardType.Text
- KeyboardType.Ascii
- KeyboardType.Number
- KeyboardType.Phone
- KeyboardType.Uri
- KeyboardType.Email
- KeyboardType.Password
- KeyboardType.NumberPassword

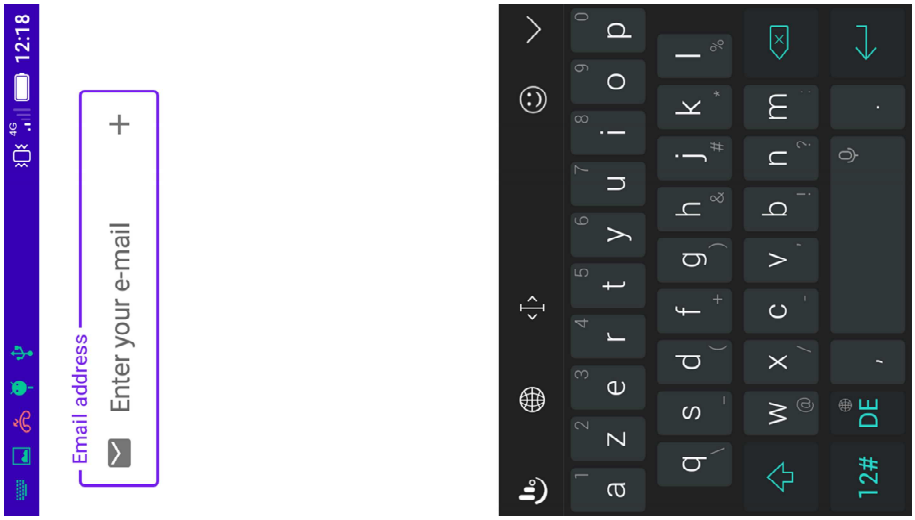
```
@Composable
fun TextFieldPasswords() {
    var text by remember { mutableStateOf(TextFieldValue( text: "")) }
    TextField(
        value = text,
        label = { Text(text = "Password Input Type") },
        visualTransformation=PasswordVisualTransformation(),
        keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Password),
        onValueChange = { it ->
            text = it
        }
    )
}
```



5. TextField with Icons

```
@Composable
fun TextFieldWithIcons() {
    var text by remember { mutableStateOf(TextFieldValue( text: "")) }
    Column(Modifier.padding(20.dp)){ this: ColumnScope
    OutlinedTextField(
        value = text,
        leadingIcon = { Icon(imageVector = Icons.Default.Email,
            contentDescription = "emailIcon") },
            trailingIcon = { Icon(imageVector = Icons.Default.Add, contentDescription = null) },
        onValueChange = { it: TextFieldValue
            text = it },
        label = { Text(text = "Email address") },
        placeholder = { Text(text = "Enter your e-mail") },)
}}
```

`leadingIcon` will add an icon in the starting area and **`trailingIcon`** will add an icon in the ending area.



In some cases, it's useful to get the value of a textfield every time the text in a text field changes. For example, you might want to build a search screen with autocomplete functionality where you want to update the results as the user types[5].

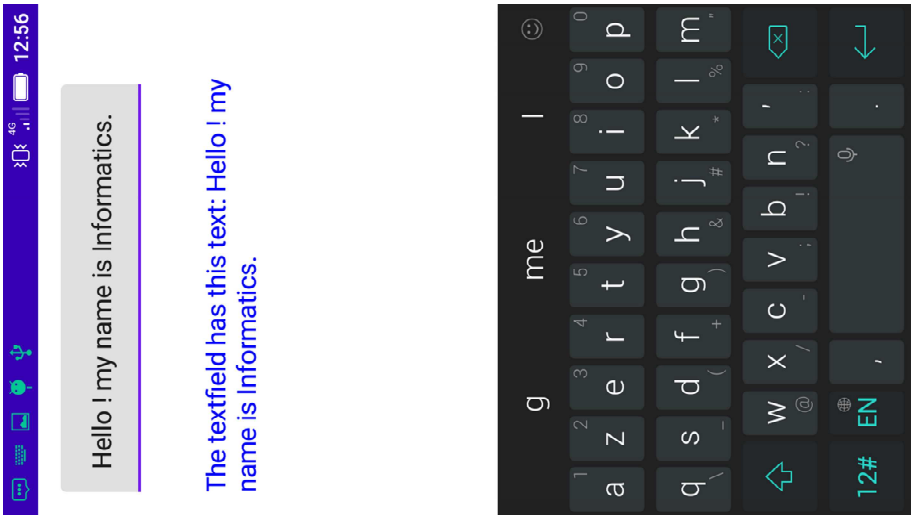
The simplest approach is to supply an onValueChange() callback to a TextField. Whenever the text changes, the callback is invoked.

In this example, every time the TextField changes, the new text value will be saved in a state and set to the TextField and the Text.

Here is an example how we can do it with Compose:

```
@Composable
fun TextFieldDemo() {
    val textState = remember { mutableStateOf(TextFieldValue()) }
    Column(Modifier.padding(16.dp)) { this: ColumnScope
        TextField(
            value = textState.value,
            onValueChange = { textState.value = it }
        )
        Spacer(modifier = Modifier.padding(20.dp))
        Text( text: "The textfield has this text: " + textState.value.text, color = Color.Blue)
    }
}
```

[5] https://foso.github.io/Jetpack-Compose-Playground/cookbook/textfield_changes/



The imports for the **`TextFields`** section are:

```
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Add
import androidx.compose.material.icons.filled.Email
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.text.input.TextFieldValue
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
```