Jetpack Compose UI

Using Material v3 Widgets

Common Widgets

- Common principles in @Composable Widget APIs
- Library Dependencies
  - Material v2, v3 (online documentation, quick guides)
  - Material v3 Expressive (announced in May 2025, more “builtin animation”)
- Action Widgets: *Buttons, SegmentedButtons, FABs, IconButton*
- TextInput: *TextField*
- Container: *Row, Column, Box, Dialog, Card, Divider, Grid*
- Selection: *Checkbox, Radio, Slider, Switch, Chip, Date/Time Picker, Menu*
- Communication: *Badge, Progress, SnackBar, Text, Image*
- Navigation: *AppBar, NavigationBar, NavigationDrawer, TabBar*
- Working with modifiers

API Design Principles

Principle #1: Return Unit

```
@Composable
public fun Text(
    text: String,
    modifier: Modifier = Modifier,
    color: Color = Color.Unspecified,
    fontSize: TextUnit = TextUnit.Unspecified,
    fontStyle: FontStyle? = null,
    fontWeight: FontWeight? = null,
    fontFamily: FontFamily? = null,
    letterSpacing: TextUnit = TextUnit.Unspecified,
    textDecoration: TextDecoration? = null,
    textAlign: TextAlign? = null,
    lineHeight: TextUnit = TextUnit.Unspecified,
    overflow: TextOverflow = TextOverflow.Clip,
    softWrap: Boolean = true,
    maxLines: Int = Int.MAX_VALUE,
    minLines: Int = 1,
    onTextLayout: ((TextLayoutResult) -> Unit)? = null,
    style: TextStyle = LocalTextStyle.current
): Unit
```

```
@Composable
public fun Slider(
    state: SliderState,
    modifier: Modifier = Modifier,
    colors: SliderColors = SliderDefaults.colors(),
    thumb: @Composable ((SliderState) -> Unit) = { sliderThumb -> }
    track: @Composable ((SliderState) -> Unit) = { sliderState -> }
): Unit
```

```
@Composable
public fun Button(
    onClick: () -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    shape: Shape = ButtonDefaults.shape,
    colors: ButtonColors = ButtonDefaults.buttonColors(),
    elevation: ButtonElevation? = ButtonDefaults.buttonElevation(),
    border: BorderStroke? = null,
    contentPadding: PaddingValues = ButtonDefaults.ContentPadding,
    interactionSource: MutableInteractionSource? = null,
    content: @Composable (RowScope.() -> Unit)
): Unit
```

```
@Composable
public fun Card(
    modifier: Modifier = Modifier,
    shape: Shape = CardDefaults.shape,
    colors: CardColors = CardDefaults.cardColors(),
    elevation: CardElevation = CardDefaults.cardElevation(),
    border: BorderStroke? = null,
    content: @Composable (ColumnScope.() -> Unit)
): Unit
```

Principle #2: Modifier parameter

```
@Composable
public fun Text(
    text: String,
    modifier: Modifier = Modifier,
    color: Color = Color.Unspecified,
    fontSize: TextUnit = TextUnit.Unspecified,
    fontStyle: FontStyle? = null,
    fontWeight: FontWeight? = null,
    fontFamily: FontFamily? = null,
    letterSpacing: TextUnit = TextUnit.Unspecified,
    textDecoration: TextDecoration? = null,
    textAlign: TextAlign? = null,
    lineHeight: TextUnit = TextUnit.Unspecified,
    overflow: TextOverflow = TextOverflow.Clip,
    softWrap: Boolean = true,
    maxLines: Int = Int.MAX_VALUE,
    minLines: Int = 1,
    onTextLayout: ((TextLayoutResult) -> Unit)? = null,
    style: TextStyle = LocalTextStyle.current
): Unit
```

```
public fun Slider(
    state: SliderState,
    modifier: Modifier = Modifier,
    colors: SliderColors = SliderDefaults.colors(),
    thumb: @Composable ((SliderState) -> Unit) = { sliderThumb -> }
    track: @Composable ((SliderState) -> Unit) = { sliderState -> }
): Unit
```

```
@Composable
public fun Button(
    onClick: () -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    shape: Shape = ButtonDefaults.shape,
    colors: ButtonColors = ButtonDefaults.buttonColors(),
    elevation: ButtonElevation? = ButtonDefaults.buttonElevation(),
    border: BorderStroke? = null,
    contentPadding: PaddingValues = ButtonDefaults.ContentPadding,
    interactionSource: MutableInteractionSource? = null,
    content: @Composable (RowScope.() -> Unit)
): Unit
```

```
@Composable
public fun Card(
    modifier: Modifier = Modifier,
    shape: Shape = CardDefaults.shape,
    colors: CardColors = CardDefaults.cardColors(),
    elevation: CardElevation = CardDefaults.cardElevation(),
    border: BorderStroke? = null,
    content: @Composable (ColumnScope.() -> Unit)
): Unit
```

Principle #3: Lambda Parameter(s)

```
@Composable
public fun Text(
    text: String,
    modifier: Modifier = Modifier,
    color: Color = Color.Unspecified,
    fontSize: TextUnit = TextUnit.Unspecified,
    fontStyle: FontStyle? = null,
    fontWeight: FontWeight? = null,
    fontFamily: FontFamily? = null,
    letterSpacing: TextUnit = TextUnit.Unspecified,
    textDecoration: TextDecoration? = null,
    textAlign: TextAlign? = null,
    lineHeight: TextUnit = TextUnit.Unspecified,
    overflow: TextOverflow = TextOverflow.Clip,
    softWrap: Boolean = true,
    maxLines: Int = Int.MAX_VALUE,
    minLines: Int = 1,
    onTextLayout: ((TextLayoutResult) -> Unit)? = null,
    style: TextStyle = LocalTextStyle.current
): Unit
```

```
public fun Slider(
    state: SliderState,
    modifier: Modifier = Modifier,
    colors: SliderColors = SliderDefaults.colors(),
    thumb: @Composable ((SliderState) -> Unit) = { sliderThumb -> }
    track: @Composable ((SliderState) -> Unit) = { sliderState -> }
): Unit
```

```
@Composable
public fun Button(
    onClick: () -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    shape: Shape = ButtonDefaults.shape,
    colors: ButtonColors = ButtonDefaults.buttonColors(),
    elevation: ButtonElevation? = ButtonDefaults.buttonElevation(),
    border: BorderStroke? = null,
    contentPadding: PaddingValues = ButtonDefaults.ContentPadding,
    interactionSource: MutableInteractionSource? = null,
    content: @Composable (RowScope.() -> Unit)
): Unit
```

```
@Composable
public fun Card(
    modifier: Modifier = Modifier,
    shape: Shape = CardDefaults.shape,
    colors: CardColors = CardDefaults.cardColors(),
    elevation: CardElevation = CardDefaults.cardElevation(),
    border: BorderStroke? = null,
    content: @Composable (ColumnScope.() -> Unit)
): Unit
```

Principle #4: Content Trailing Lambdas

```
@Composable
@ComposableInferredTarget
public inline fun Column(
    modifier: Modifier = Modifier,
    verticalArrangement: Arrangement.Vertical = Arrangement.Top,
    horizontalAlignment: Alignment.Horizontal = Alignment.Start,
    content: @Composable (ColumnScope.() -> Unit)
): Unit
```

```
@Composable
public fun Card(
    modifier: Modifier = Modifier,
    shape: Shape = CardDefaults.shape,
    colors: CardColors = CardDefaults.cardColors(),
    elevation: CardElevation = CardDefaults.cardElevation(),
    border: BorderStroke? = null,
    content: @Composable (ColumnScope.() -> Unit)
): Unit
```

```
public fun LazyColumn(
    modifier: Modifier = Modifier,
    state: LazyListState = rememberLazyListState(),
    contentPadding: PaddingValues = PaddingValues(0.dp),
    reverseLayout: Boolean = false,
    verticalArrangement: Arrangement.Vertical = Arrangement.Center,
    horizontalAlignment: Alignment.Horizontal = Alignment.Start,
    flingBehavior: FlingBehavior = ScrollableDefaults.___,
    userScrollEnabled: Boolean = true,
    overscrollEffect: OverscrollEffect? = rememberOverscrollEffect(),
    content: LazyListScope.() -> Unit
): Unit
```

```
@Composable
public fun Button(
    onClick: () -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    shape: Shape = ButtonDefaults.shape,
    colors: ButtonColors = ButtonDefaults.buttonColors(),
    elevation: ButtonElevation? = ButtonDefaults.buttonElevation(),
    border: BorderStroke? = null,
    contentPadding: PaddingValues = ButtonDefaults.ContentPadding,
    interactionSource: MutableInteractionSource? = null,
    content: @Composable (RowScope.() -> Unit)
): Unit
```

Principle #5: Only a Few Non-Default Parameters

```
@Composable
public fun Text(
    text: String,
    modifier: Modifier = Modifier,
    color: Color = Color.Unspecified,
    fontSize: TextUnit = TextUnit.Unspecified,
    fontStyle: FontStyle? = null,
    fontWeight: FontWeight? = null,
    fontFamily: FontFamily? = null,
    letterSpacing: TextUnit = TextUnit.Unspecified,
    textDecoration: TextDecoration? = null,
    textAlign: TextAlign? = null,
    lineHeight: TextUnit = TextUnit.Unspecified,
    overflow: TextOverflow = TextOverflow.Clip,
    softWrap: Boolean = true,
    maxLines: Int = Int.MAX_VALUE,
    minLines: Int = 1,
    onTextLayout: ((TextLayoutResult) -> Unit)? = null,
    style: TextStyle = LocalTextStyle.current
): Unit
```

```
public fun Slider(
    state: SliderState,
    modifier: Modifier = Modifier,
    colors: SliderColors = SliderDefaults.colors(),
    thumb: @Composable ((SliderState) -> Unit) = { sliderThumb -> }
    track: @Composable ((SliderState) -> Unit) = { sliderState -> }
): Unit
```

```
@Composable
public fun Button(
    onClick: () -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    shape: Shape = ButtonDefaults.shape,
    colors: ButtonColors = ButtonDefaults.buttonColors(),
    elevation: ButtonElevation? = ButtonDefaults.buttonElevation(),
    border: BorderStroke? = null,
    contentPadding: PaddingValues = ButtonDefaults.ContentPadding,
    interactionSource: MutableInteractionSource? = null,
    content: @Composable (RowScope.() -> Unit)
): Unit
```

```
@Composable
public fun Card(
    modifier: Modifier = Modifier,
    shape: Shape = CardDefaults.shape,
    colors: CardColors = CardDefaults.cardColors(),
    elevation: CardElevation = CardDefaults.cardElevation(),
    border: BorderStroke? = null,
    content: @Composable (ColumnScope.() -> Unit)
): Unit
```

Using Common Widgets

Snippet Code Template

```
@Composable
fun YourFirstCompose (modifier: Modifier = Modifier) {
    Column (modifier = modifier.background(Color.Red)) {
        // Insert more Widgets here
    }
}
```

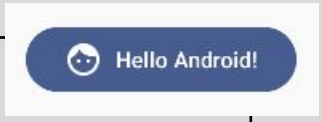
- *The brown modifier is the parameter name of YourFirstCompose*
- *The blue modifier belongs to Column, and we are passing brown to blue by name*

```
@Composable
fun YourFirstCompose (decorator: Modifier = Modifier) {
    Column (modifier = decorator.background(Color.Red)) {
        // Insert more Widgets here
    }
}
```

Dependencies

```
// Bill of Materials
implementation(platform(“androidx.compose:compose-bom:2025.12.00”))

// Material3
implementation(“androidx.compose.material3:material3”)
implementation(“androidx.compose.material3:material-icons-extended”)

// Android Studio Preview support
implementation(“androidx.compose.ui:ui-tooling-preview”)
debugImplementation(“androidx.compose.ui:ui-tooling”)

// Integration with activities
// Optional - Integration with activities
implementation("androidx.activity:activity-compose:1.11.0")
// Optional - Integration with ViewModels
implementation("androidx.lifecycle:lifecycle-viewmodel-compose:2.8.5")
```

Buttons




```
@Composable
fun SampleCode(modifer: Modifier = Modifier) {
    Button(onClick = { println (“....”)}) {
        Icon(imageVector = Icons.Default.Face, contentDescription = "")
        Text(text = "Hello Android!",
             modifier = modifier.padding(start = 8.dp))
    }
    IconButton(onClick = { println (“....”)}) {
        Icon(imageVector = Icons.Default.Face, contentDescription = "")
    }
}
```

Layout & Measurements

Layout @Composable

Standard layout components

- `Row { }`: arranges children horizontally along the X-axis
- `Column { }`: arranges children vertically along the Y-axis
- `Box { }`: stack children along the Z-axis (out of the screen towards the viewer)

Units of measurement

- .sp (scaled point, in place of “font points”) for text size
- .dp (device independent pixel, in place of “screen pixels”) for box size/width/height

Compose Row/Column ⇔ CSS Flexbox

If you happen to know CSS FlexBox…

	Row	Column	CSS Flexbox
View Equivalent	`<LinearLayout orientation="horizontal">`	`<LinearLayout orientation="vertical">`	flex-direction: row flex-direction: column
Main Axis Placement	horizontalArrangement (along the X-axis)	verticalArrangement (along the Y-axis)	justify
Cross Axis Placement	verticalAlignment (along the Y-axis)	horizontalAlignment (along the X-axis)	align

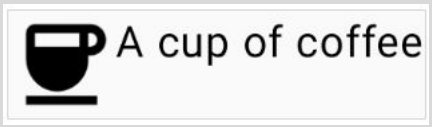
Common Import for Compose

```
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout._______        // containers
import androidx.compose.material3._____                  // widgets
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview
```

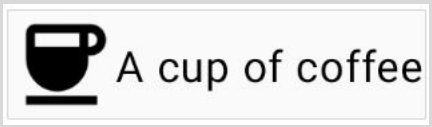
- Don't have to memorize these names. Use Android Studio **auto completion** to assist you in selecting the right one.
- Whenever multiple names show up, select those from the `androidx` or `androidx.compose` group.

Row & Vertical Alignment

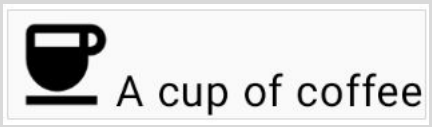
```
Row /* (verticalAlignment = Alignment.Top) */ {
   Image(painterResource(id = R.drawable.coffee),
       contentDescription = null, modifier = Modifier.size(40.dp)
   Text("A cup of coffee")
}
```




```
Row(verticalAlignment = Alignment.CenterVertically) {
   Image(painterResource(id = R.drawable.coffee),
       contentDescription = null, modifier = Modifier.size(40.dp)
   Text("A cup of coffee")
}
```



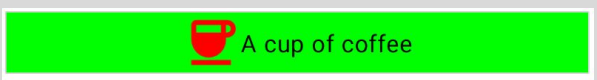

```
Row(verticalAlignment = Alignment.Bottom) {
   Image(painterResource(id = R.drawable.coffee),
       contentDescription = null, modifier = Modifier.size(40.dp)
   Text("A cup of coffee")
}
```



Row & Horizontal Arrangement

```
Row (horizontalArrangement = Arrangement.Start) {
   Image()
   Text()
}
```


```
Row (horizontalArrangement = Arrangement.Center) {
   Image()
   Text()
}
```



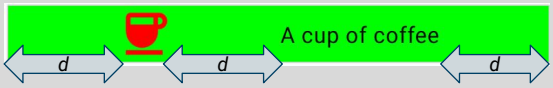

```
Row (horizontalArrangement = Arrangement.End) {
   Image()
   Text()
}
```

Row & Horizontal Arrangement

```
Row (horizontalArrangement = Arrangement.SpaceBetween) {
   Image()
   Text()
}
```

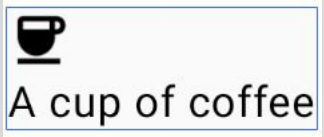

```
Row (horizontalArrangement = Arrangement.SpaceAround) {
   Image()
   Text()
}
```


```
Row (horizontalArrangement = Arrangement.SpaceEvenly) {
   Image()
   Text()
}
```

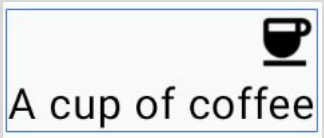


Column & Horizontal Alignment

```
Column /* (horizontalAlignment = Alignment.Start) */ {
   Image(painterResource(id = R.drawable.coffee),
       contentDescription = null)
   Text("A cup of coffee")
}
```




```
Column(horizontalAlignment = Alignment.End) {
   Image(painterResource(id = R.drawable.coffee),
       contentDescription = null)
   Text("A cup of coffee")
}
```




```
Column(horizontalAlignment = Alignment.CenterHorizontally) {
   Image(painterResource(id = R.drawable.coffee),
       contentDescription = null)
   Text("A cup of coffee")
}
```

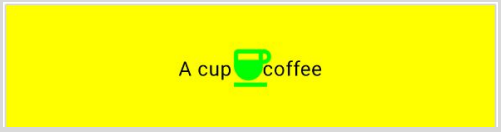
Box & Content Alignment

```
Box (contentAlignment = Alignment.TopStart) {
   Image()
   Text()
}
```


```
Box (contentAlignment = Alignment.BottomEnd) {
   Image()
   Text()
}
```


```
Box (contentAlignment = Alignment.Center) {
   Image()  // Image behind the text
   Text()
}
```


```
Box (contentAlignment = Alignment.Center) {
   Text()  // Text behind the image
   Image()
}
```



Android Studio Relevant Features

- Edit mode (Code Only/Split/Design Only)
- Hover for online docs
- Layout Inspector (with Emulator)

Handling User Input

TextFields

Value-based

- User input is captured by `onValueChange`
- Input & output formatting are handled by `VisualTransformation`
- Secure text field (like password input) is not supported

```
var email by remember { mutableStateOf(“”) }
TextField(value = email,
          onValueChange = { email = it })
```

State-based (new in Material3 v 1.4.0)

- No `onValueChange`, TextField internally updates the input variables
  - *The state variable survives recomposition and configuration changes*
- Input formatting is handled by `InputTransformation`
- Output formatting is handled by `OutputTransformation`
- `SecureTextField` is a @Composable with configurable text obfuscation mode

```
val email = rememberTextFieldState(“”)
TextField(state = email)
```

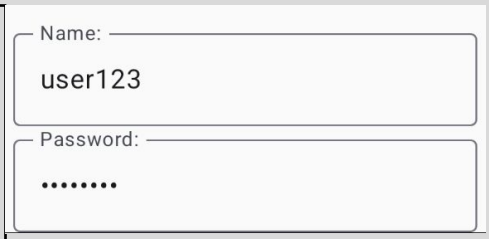
State-based vs. Value-based

```
public fun TextField(
    state: TextFieldState,
    labelPosition: TextFieldLabelPosition =.Attached,
    inputTransformation: InputTransformation? = null,
    outputTransformation: OutputTransformation? = null,
    lineLimits: TextFieldLineLimits = TextFieldLineLimits.Default,
    onTextLayout: (Density.(() -> TextLayoutResult?) -> Unit)? = null,
    scrollState: ScrollState = rememberScrollState(),
    contentPadding: PaddingValues = ...,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    readOnly: Boolean = false,
    textStyle: TextStyle = LocalTextStyle.current,
    label: @Composable (TextFieldLabelScope.() -> Unit)? = null,
    placeholder: @Composable (() -> Unit)? = null,
    leadingIcon: @Composable (() -> Unit)? = null,
    trailingIcon: @Composable (() -> Unit)? = null,
    prefix: @Composable (() -> Unit)? = null,
    suffix: @Composable (() -> Unit)? = null,
    supportingText: @Composable (() -> Unit)? = null,
    isError: Boolean = false,
    keyboardOptions: KeyboardOptions = KeyboardOptions.Default,
    onKeyboardAction: KeyboardActionHandler? = null,
    shape: Shape = TextFieldDefaults.shape,
    colors: TextFieldColors = TextFieldDefaults.colors(),
    interactionSource: MutableInteractionSource? = null
): Unit
```

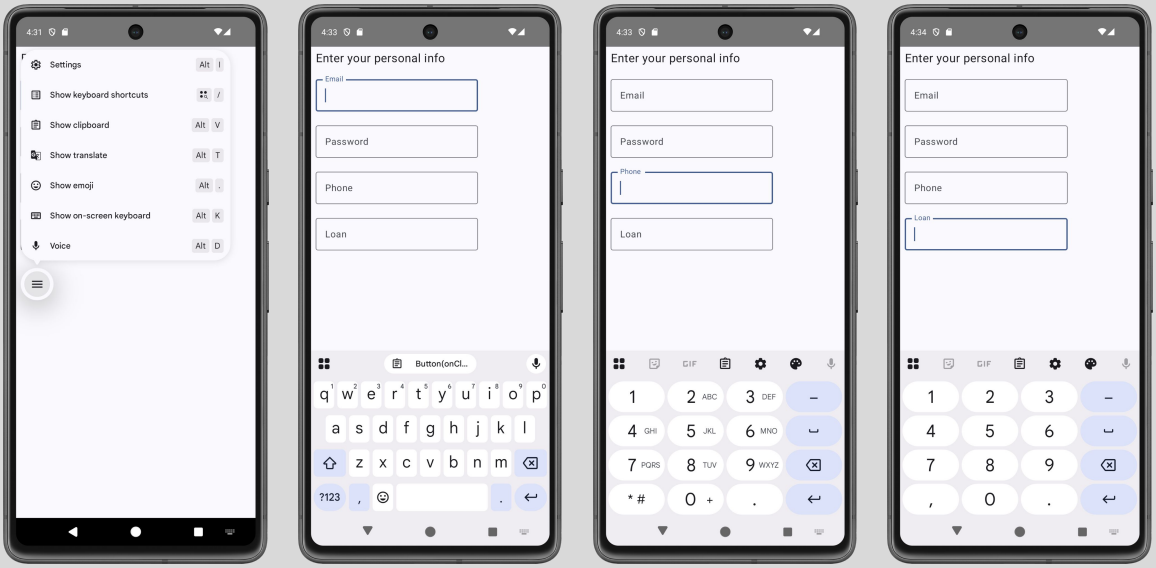
```
public fun TextField(
    value: String,
    onValueChange: (String) -> Unit,
    visualTransformation: VisualTransformation = .None,
    singleLine: Boolean = false,
    maxLines: Int = if (singleLine) 1 else Int.MAX_…,
    minLines: Int = 1,


    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    readOnly: Boolean = false,
    textStyle: TextStyle = LocalTextStyle.current,
    label: @Composable (() -> Unit)? = null,
    placeholder: @Composable (() -> Unit)? = null,
    leadingIcon: @Composable (() -> Unit)? = null,
    trailingIcon: @Composable (() -> Unit)? = null,
    prefix: @Composable (() -> Unit)? = null,
    suffix: @Composable (() -> Unit)? = null,
    supportingText: @Composable (() -> Unit)? = null,
    isError: Boolean = false,
    keyboardOptions: KeyboardOptions = KeyboardOptions.Default,
    keyboardActions: KeyboardActions = KeyboardActions.Default,
    shape: Shape = TextFieldDefaults.shape,
    colors: TextFieldColors = TextFieldDefaults.colors()
    interactionSource: MutableInteractionSource? = null,
): Unit
```

TextField

```
@Composable
fun SampleCode() {
    var loginName by remember { mutableStateOf(“”) }
    var password by rememberTextFieldState(“”)
    OutlinedTextField(value = loginName,
        onValueChange = { loginName = it },
        label = { Text("Name:") })
    OutlinedSecureTextField(state = password,
         label = { Text("Password:") })
}
```



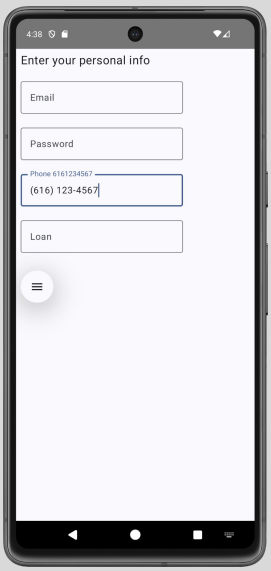
Emulator On-Screen Keyboard



Textfield Output Formatting (State-based)

```
val phone = rememberTextFieldState("")
TextField(state = phone,
    keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Phone),
    outputTransformation = OutputTransformation {
        if (this.length > 0) this.insert(0, "(")
        if (length > 4) insert(4, ") ")   // "this." is optional
        if (length > 9) insert(9, "-")
    })
```

- `OutputTransformation` defines an extension function on the `TextFieldBuffer` class
- "`this`" in the above code refers to `TextFieldBuffer`

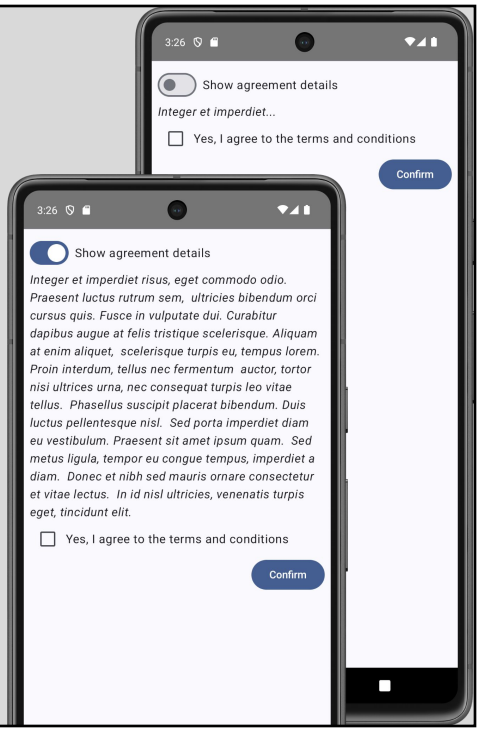


Simple Selection Input

Checkbox, Switch, Radio Button, Slider

Checkbox (single)

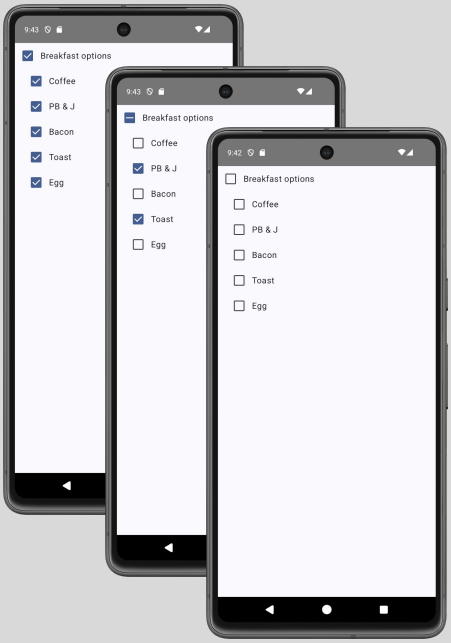
```
var agree by remember { mutableStateOf(false) }
var showDetails by remember { mutableStateOf(false) }
Switch(checked = showDetails,
    onCheckedChange = { showDetails = it }
)
if(showDetails) Text("<longer text>")
else Text("<shorter text>")
Checkbox(checked = agree,
    onCheckedChange = { agree = it }
)
Text("Yes, I agree to the terms and conditions")
```



- Use a switch to trigger **immediate** effect/action
- Use a checkbox to **prepare** for an action, but the action itself is triggered by a (separate) button

Checkbox (Group)

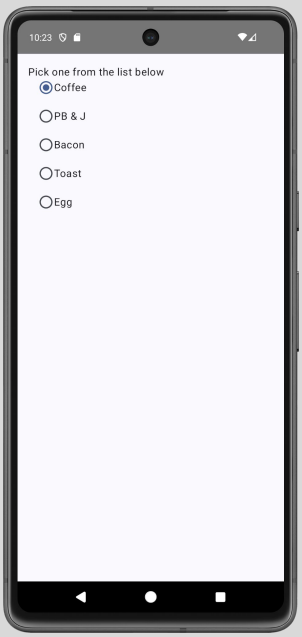
```
val breakfastOptions = listOf("Coffee",  "PB & J", "Bacon", "Toast", "Egg")
val optionChecked = remember { mutableStateListOf(false, false, false, false, false) }
val allBreakfastState = when {
    optionChecked.all { it } -> ToggleableState.On
    optionChecked.none { it } -> ToggleableState.Off
    else -> ToggleableState.Indeterminate
}
Column() {
    Row(verticalAlignment = Alignment.CenterVertically) {
        TriStateCheckbox(state = allBreakfastState,
            onClick = {
                val newState = allBreakfastState != ToggleableState.On
                optionChecked.replaceAll { newState }
            }
        )
        Text("Breakfast options")
    }
    optionChecked.forEachIndexed { pos, selection ->
        Row {
            Checkbox(checked = selection,
                onCheckedChange = { optionChecked[pos] = it})
            Text(breakfastOptions[pos])
        }
    }
}
```



Radio Button

```
val options = listOf("Coffee", "PB & J", "Bacon", "Toast", "Egg")
val (currentOption, setCurrentOption) = remember { mutableStateOf(options[0])}
val (currentIndex, setCurrentIndex) = remember { mutableStateOf(0)}

options.forEachIndexed { pos, text ->
    Row(modifier = Modifier.selectable(
        selected = (text == currentOption),
        onClick = {
            setCurrentOption(text)
            setCurrentIndex(pos)
        })) {
            RadioButton(selected = text == currentOption, onClick = null)
            Text(text)
        }
   }
```

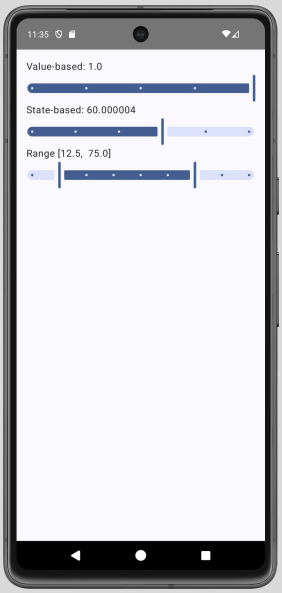
Use of `.selectable()` modifier on each Row()



Slider/Range Slider (state-based)

```
val sliderVal = remember { mutableStateOf(0f) }
val sliderPos = rememberSliderState(value = 0f, steps = 4,
                                    valueRange = 0f..100f)
val sliderRange = rememberRangeSliderState(valueRange = 0f..100f,
    activeRangeStart = 12.5f, activeRangeEnd = 75f, steps = 7)

Text("Value-based: ${sliderVal}")
Slider(value = sliderVal, onValueChange = {sliderVal = it})
Text("State-based: ${sliderPos.value}")
Slider(state = sliderPos)
Text("Range [${sliderRange.activeRangeStart},${sliderRange.activeRangeEnd}]" )
  ●   RangeSlider(state = sliderRange)
```

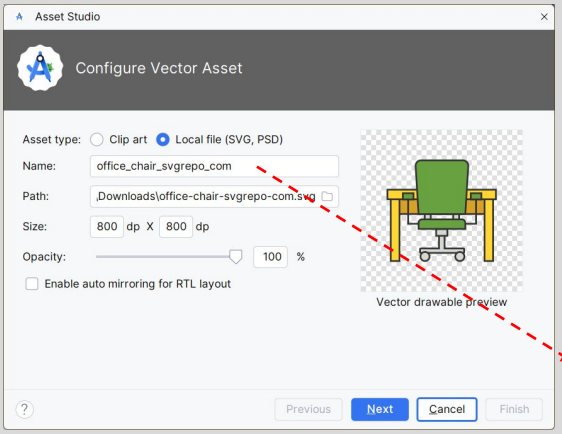
- steps: number of intermediate values in-between min/max

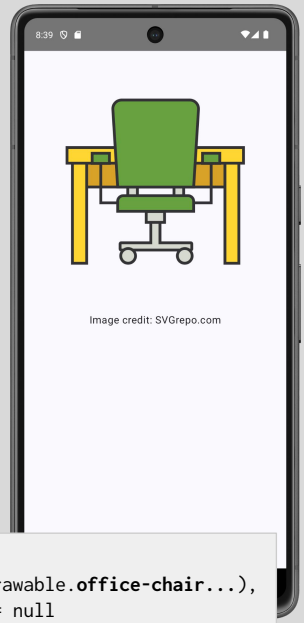


Images

Image Asset (SVG)

1. From App ⇒ res ⇒ drawable, select New ⇒ Vector Asset
2. Select Local File, assign a name for the vector asset
3. You should find a new file under `res/drawable/`
4. Use painterResource() to access the image






```
Image(
   painterResource(R.drawable.office-chair...),
   contentDescription = null
)
```

Bitmap Images

1. If needed, rename the image file to use only lowercase letters and underscore (keep the extension .jpg, .png) in the name
2. Drag the file into `res/drawable/`
3. Use `painterResource()` to access the image

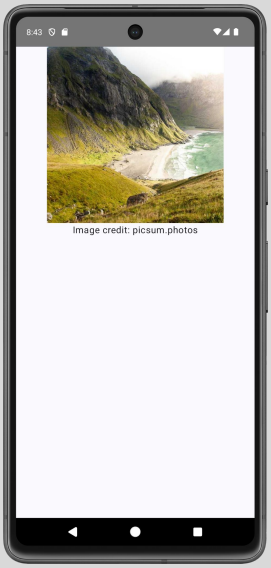
```
// Use the R.drawable.____ name WITHOUT the .jpg, .png extension
Image(
   painterResource(R.drawable.your_image_file_name),
   contentDescription = null
)
```

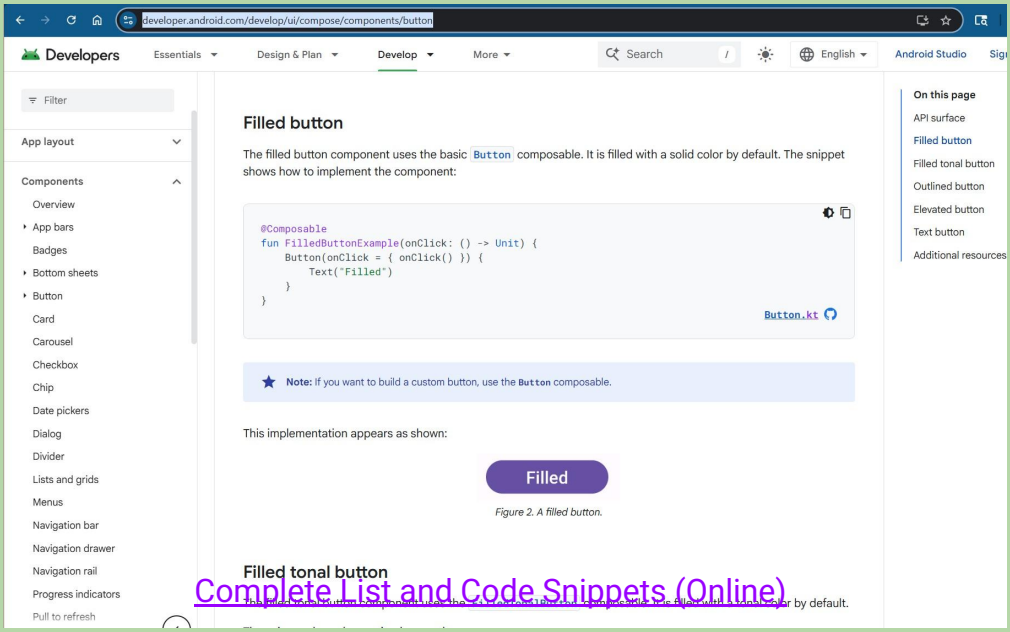
Image on the Internet

```
// Module build.gradle.kts
implementation ("io.coil-kt.coil3:coil-compose:3.3.0")
implementation ("io.coil-kt.coil3:coil-network-okhttp:3.3.0")
```

```
// AndroidManifest.xml
<manifest xmlns:android="......">
   <uses-permission android:name="android.permission.INTERNET" />
   <application ... />
</manifest>
```

```
AsyncImage(
   model = “https://picsum.photos/400”,
   contentDescription = null
)
```





Complete List and Code Snippets (Online)

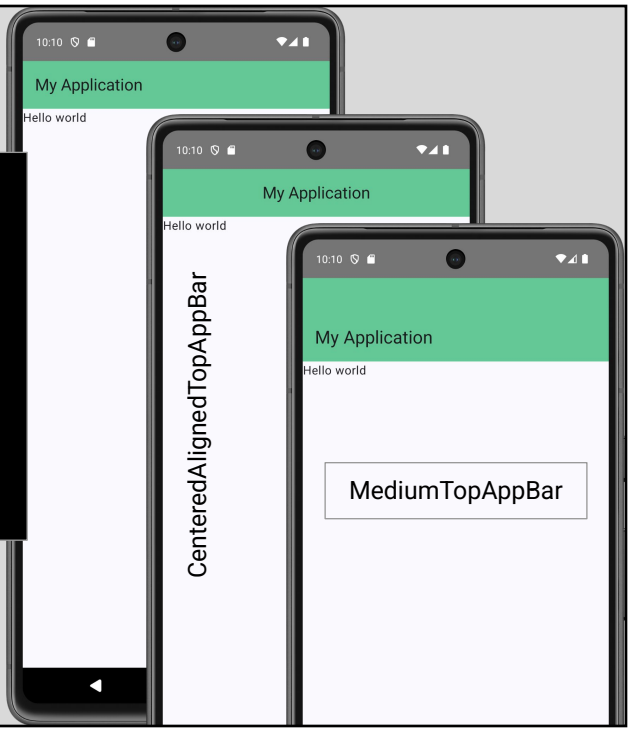
Application UI Structure

App Structure: Scaffold

- TopAppBar
- BottomBar
- Floating Action Button

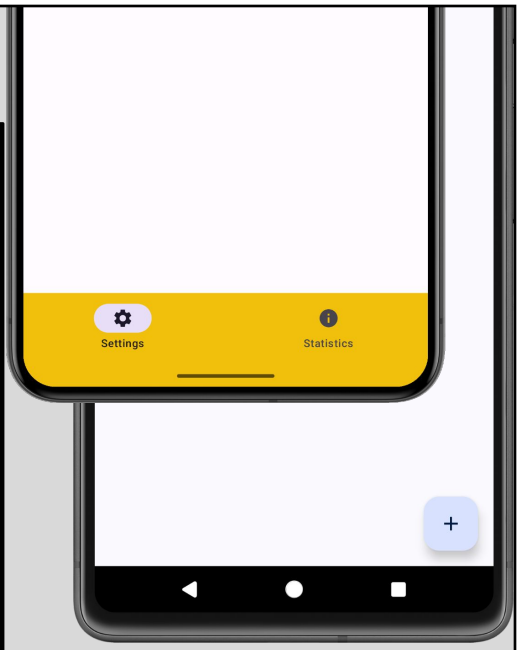
TopAppBar

```
Scaffold(
    modifier = Modifier.fillMaxSize(),
    topBar = {
        TopAppBar(
            title = {
                Text("Compose Components")
            },
            colors = TopAppBarDefaults.topAppBarColors(
                containerColor = Color(100, 200, 150))
    }) { innerPadding ->
      // The main content of UI here
      // The main content of UI here
      // The main content of UI here
    })
```

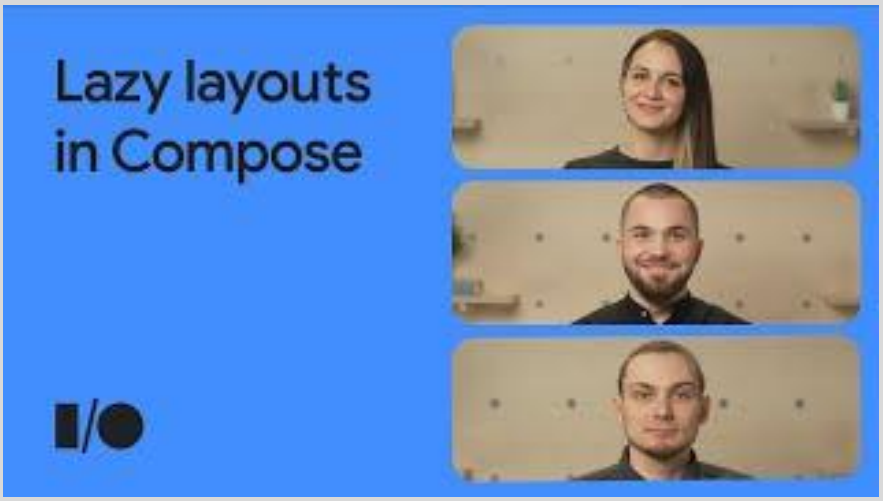


Bottom Navigation Bar

```
Scaffold(
    bottomBar = {
        NavigationBar {
            NavigationBarItem(selected = false,
               onClick = { /* code */},
               label = { Text (“___”) }
               icon = { Icon(imageVector = Icons.Default.____)})
            NavigationBarItem(selected = false,
               onClick = { /* code */},
               icon = { Icon(imageVector = Icons.Default.____)})
        }
    },
    floatingActionButton = {
        FloatingActionButton(onClick = {}) {
            Icon(imageVector = Icons.Filled.Add, ____)
        }
    }
) { innerPadding ->
      // The rest of UI here
}
```



List and Grids



Warning: this YouTube video was published in 2023, some of the APIs may have been updated

Summary

- Common Widgets
- Standard Layout Containers
  - Row: arrange widgets along the X-axis
  - Column: arrange widgets along the Y-axis
  - Box: stack widgets along the Z-axis
- Widget Placement
  - Arrangement: placement along main axis
  - Alignment: placement along cross-axis