Dialog and List View

Asst. Prof. Monlica Wattana, Ph.D
Department of Computer Science,
Khon Kaen University

SC 362 007 Mobile Device Programming

CP 410 804 Programming for Mobile Application

Outline

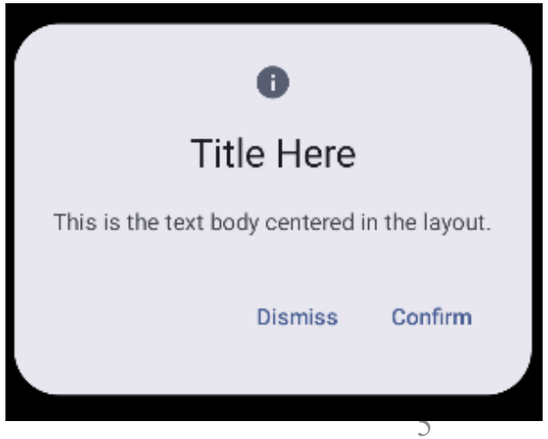
- Dialog
- Lists and Grids View

Dialogs

- A dialog is a small window that prompts the user to make a decision or enter additional information.
- A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed.
- A dialog is always created and displayed as a part of an Activity.

Dialogs

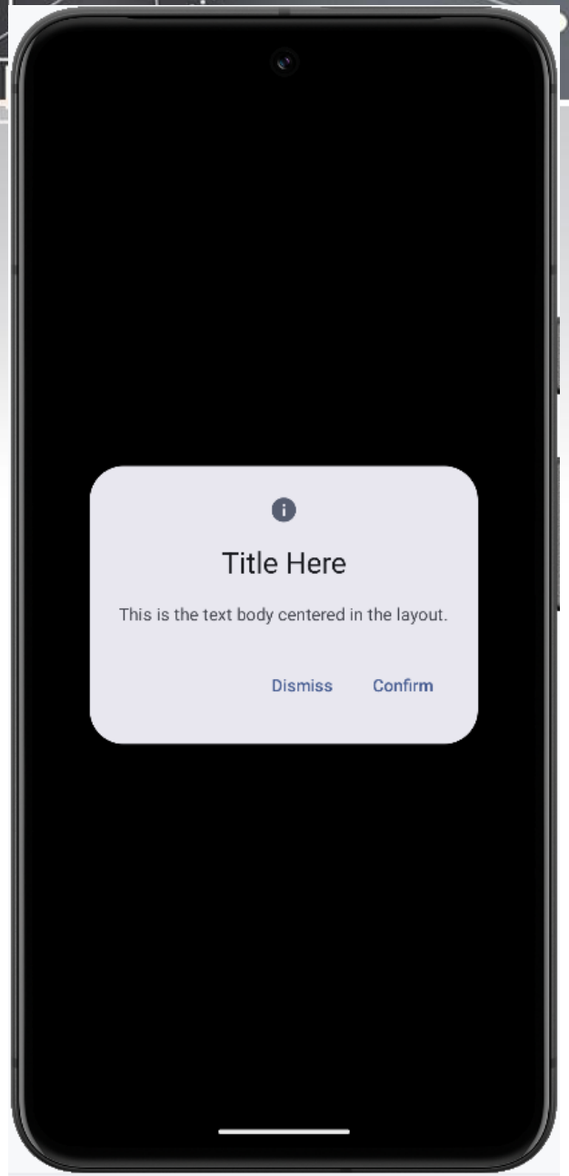
- AlertDialog has specific parameters for handling particular elements of the dialog.
  - title: The text that appears along the top of the dialog.
  - text: The text that appears centered within the dialog.
  - icon: The graphic that appears at the top of the dialog.
  - onDismissRequest: The function called when the user dismisses the dialog, such as by tapping outside of it.
  - dismissButton: A composable that serves as the dismiss button.
  - confirmButton: A composable that serves as the confirm button.



Alert Dialog

Simple Alert Dialog

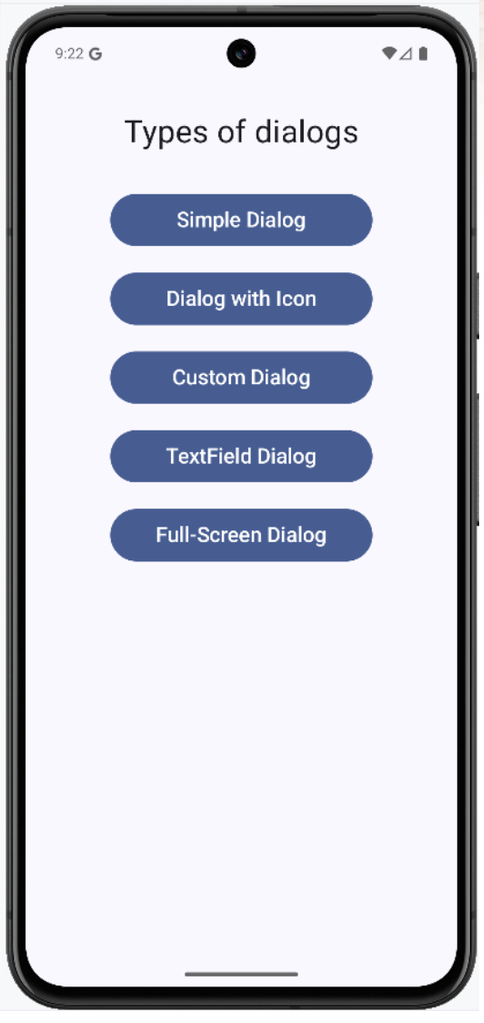
```
AlertDialog(
    onDismissRequest = { /* Close Dialog */ },
    icon = {
        Icon(Icons.Default.Info, contentDescription = null) // icon    },
    title = {
        Text(text = "Title Here") // title  },
    text = {
        Text(text = "This is the text body centered in the layout.") // text  },
    confirmButton = {
        TextButton(onClick = { /* ... */ }) {
            Text("Confirm") // confirmButton        }      },
    dismissButton = {
        TextButton(onClick = { /* ... */ }) {
            Text("Dismiss") // dismissButton      }     }
)
```



Alert Dialog

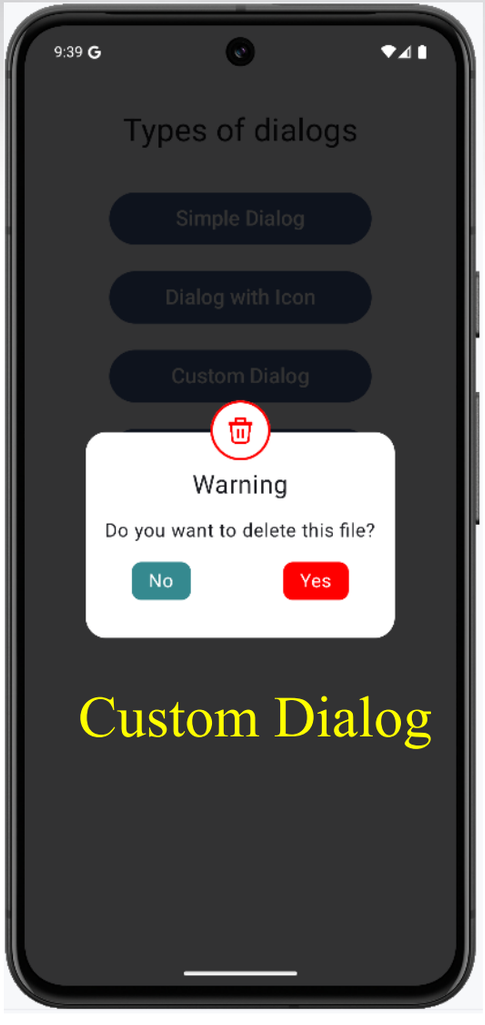
- Simple Dialog
- Dialog with Icon
- Custom Dialog
- TextField Dialog
- Full-Screen Dialog

Alert Dialog



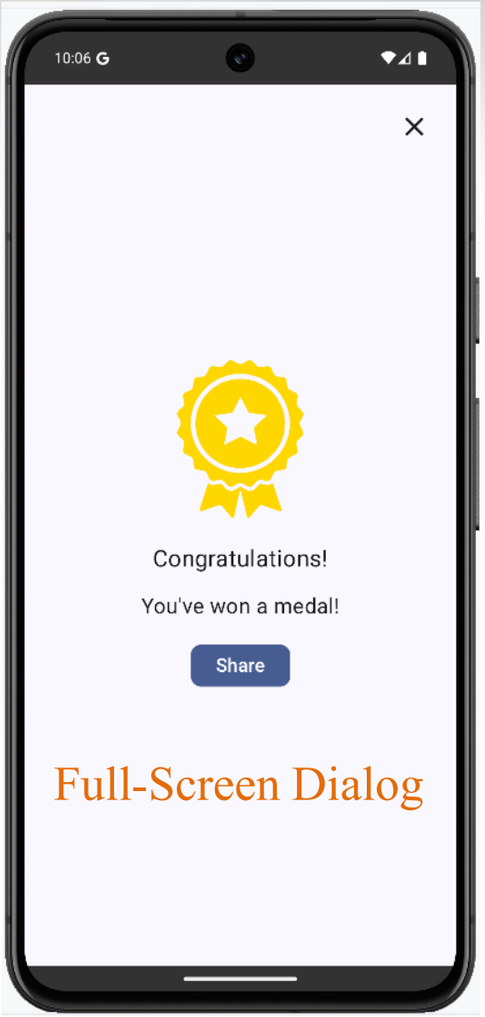
Simple Dialog

Dialog with Icon



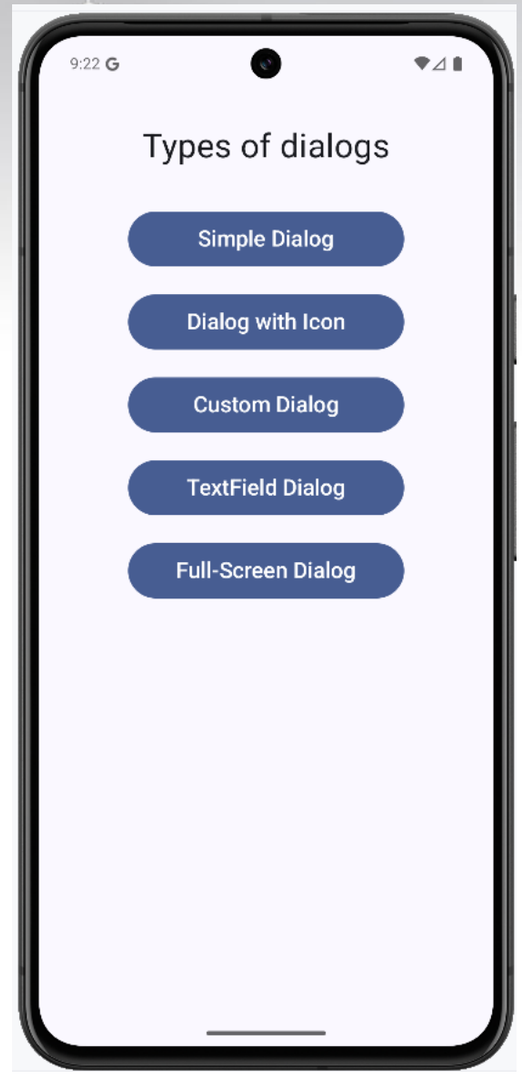
Custom Dialog

Alert Dialog

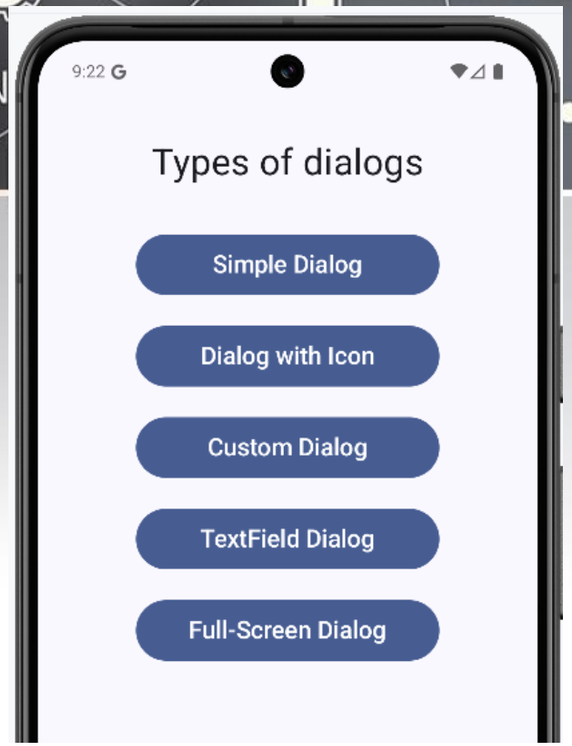


Alert Dialog Example

```
@Composable
fun MyScreen(modifier : Modifier){

    val contextForToast = LocalContext.current.applicationContext
    var showSimpleDialog by rememberSaveable() { mutableStateOf( value = false) }
    var showDialogIcon  by rememberSaveable { mutableStateOf( value = false) }
    var showDialogCustom by rememberSaveable { mutableStateOf( value = false) }
    var showDialogTextField by rememberSaveable { mutableStateOf( value = false) }
    var showDialogFull by rememberSaveable {mutableStateOf( value = false)}
    val buttonWidth = 250.dp

    Column(modifier = modifier
        .fillMaxSize()
        .padding( all = 16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.spacedBy( space = 25.dp)
    ){
```



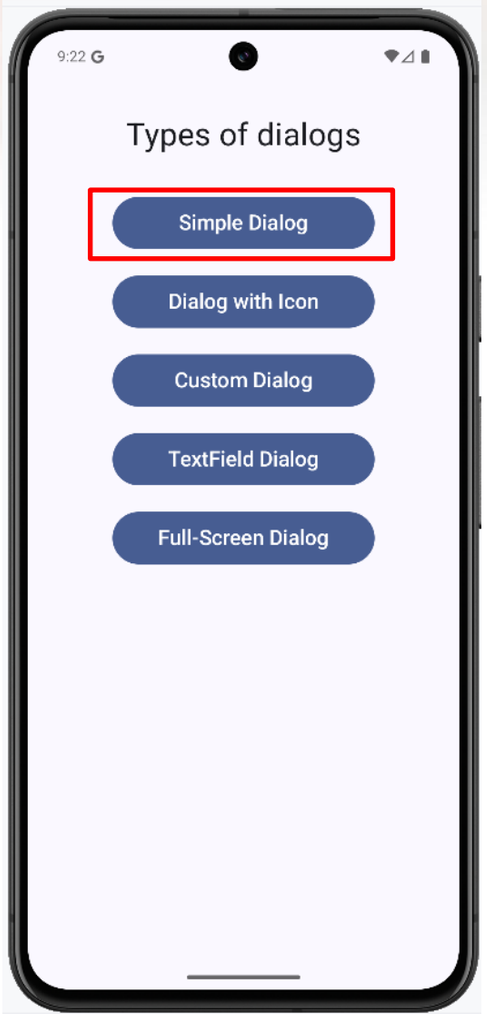
Alert Dialog Example

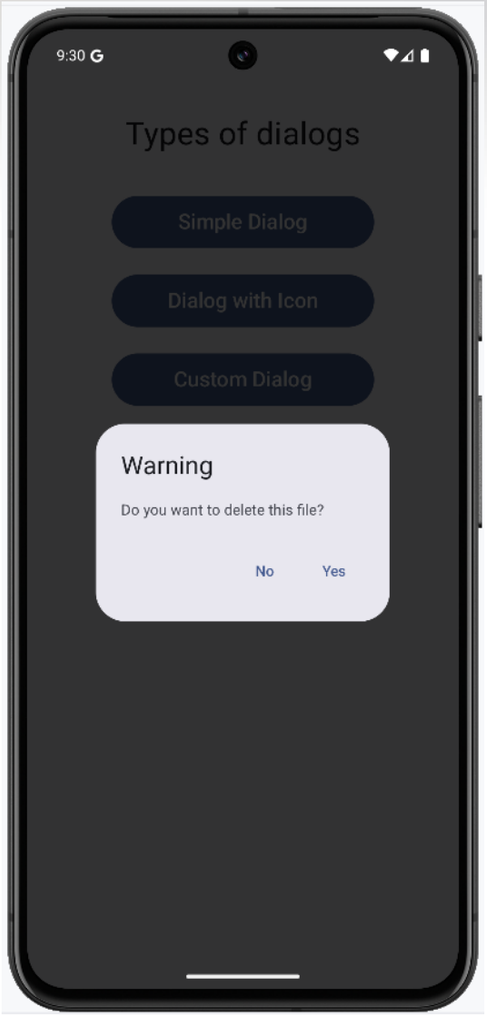



```
Text(modifier = Modifier
    .padding( all = 16.dp),
    text = "Types of dialogs",
    fontSize = 30.sp)
Button(modifier = Modifier.width(buttonWidth),
    onClick = { showSimpleDialog = true
    }) {
    Text(modifier= Modifier.padding(vertical = 5.dp),
        text = "Simple Dialog", fontSize = 20.sp)
}
Button(modifier = Modifier.width(buttonWidth),
    onClick = {  showDialogIcon = true
    }) {
    Text(modifier= Modifier.padding(vertical = 5.dp),
        text = "Dialog with Icon", fontSize = 20.sp)
}
Button(modifier = Modifier.width(buttonWidth),
    onClick = { showDialogCustom = true
    }) {
    Text(modifier= Modifier.padding(vertical = 5.dp),
        text = "Custom Dialog", fontSize = 20.sp)
}
```

```
    Button(modifier = Modifier.width(buttonWidth),
        onClick = { showDialogTextField= true
        }) {
        Text(modifier= Modifier.padding(vertical = 5.dp),
            text = "TextField Dialog", fontSize = 20.sp)
    }
    Button(modifier = Modifier.width(buttonWidth),
        onClick = { showDialogFull = true
        }) {
        Text(modifier= Modifier.padding(vertical = 5.dp),
            text = "Full-Screen Dialog", fontSize = 20.sp)
    }
}
```

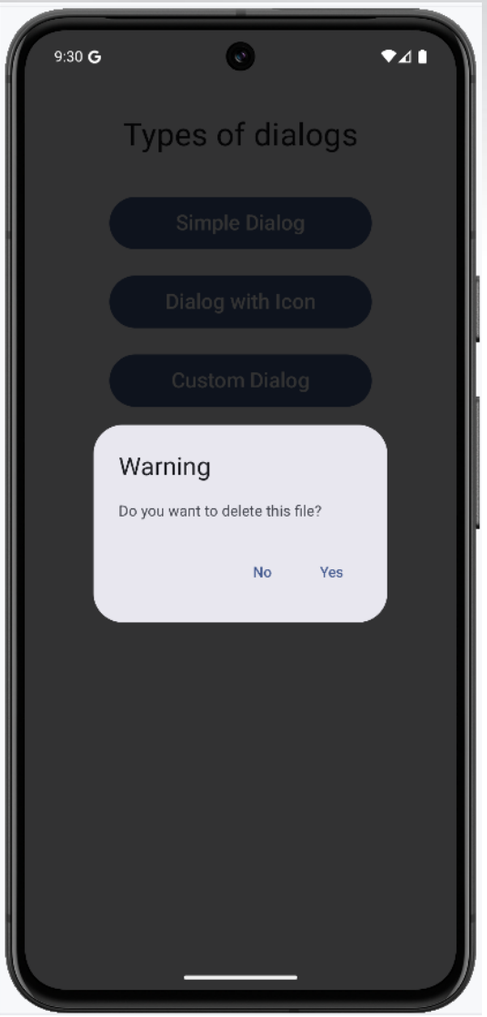
Alert Dialog Example: Simple Dialog



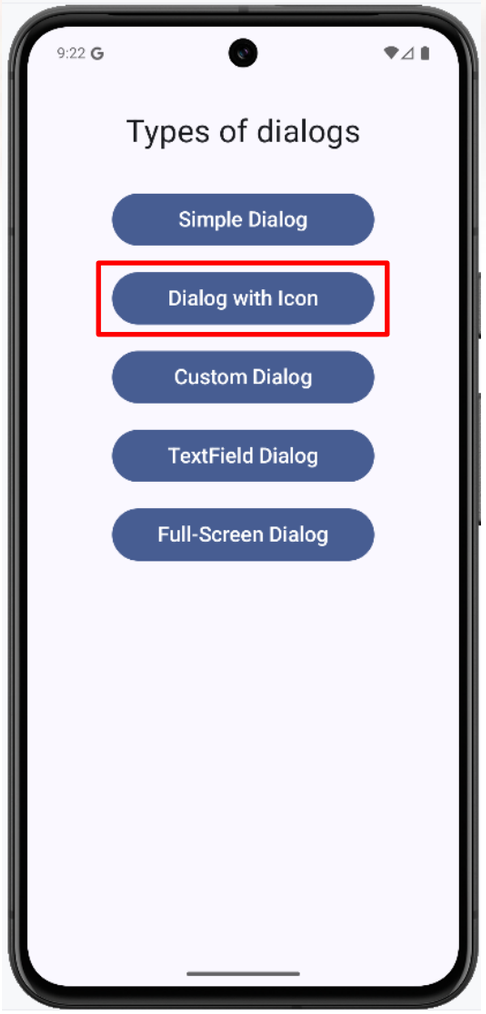


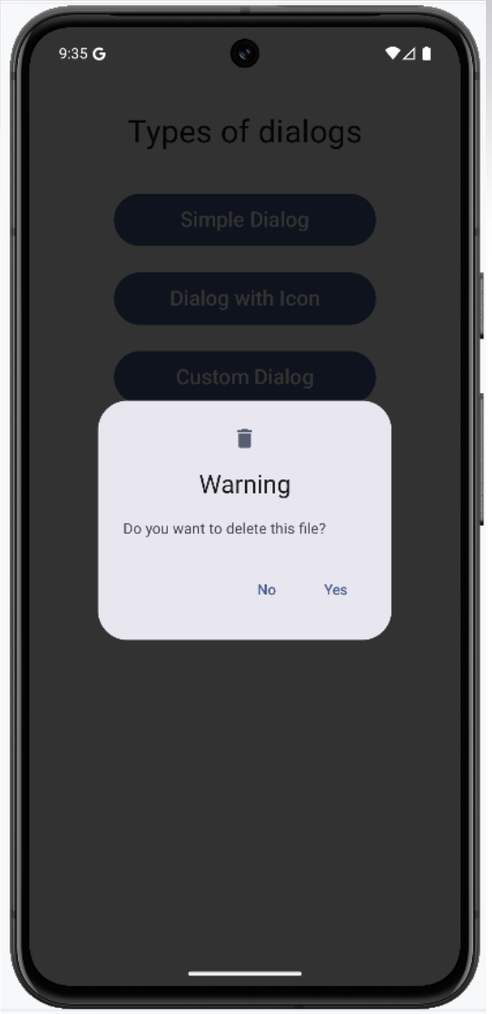
Alert Dialog Example: Simple Dialog

```
////Simple Dialog
if (showSimpleDialog) {
    AlertDialog(
        onDismissRequest = { showSimpleDialog = false },
        confirmButton = {
            TextButton(
                onClick = {
                    showSimpleDialog = false
                    Toast.makeText(contextForToast, text = "Yes",
                        duration = Toast.LENGTH_SHORT).show()
                }
            ) { Text(text = "Yes") }
        },
        dismissButton = {
            TextButton(
                onClick = {
                    showSimpleDialog = false
                    Toast.makeText(contextForToast, text = "No",
                        duration = Toast.LENGTH_SHORT).show()
                }
            ) { Text(text = "No") }
        },
        title = { Text(text = "Warning") },
        text = { Text(text = "Do you want to delete this file?") }
    )
}
```



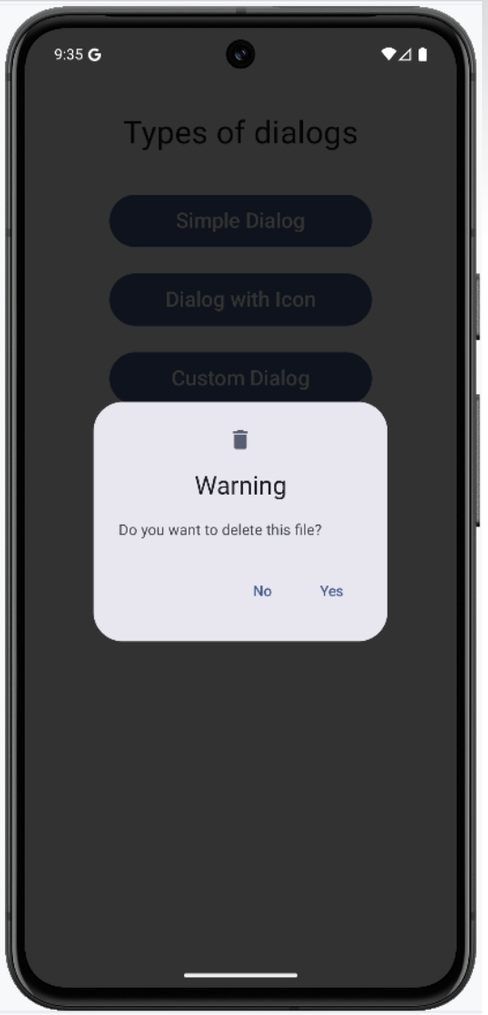
Alert Dialog Example: Dialog with Icon



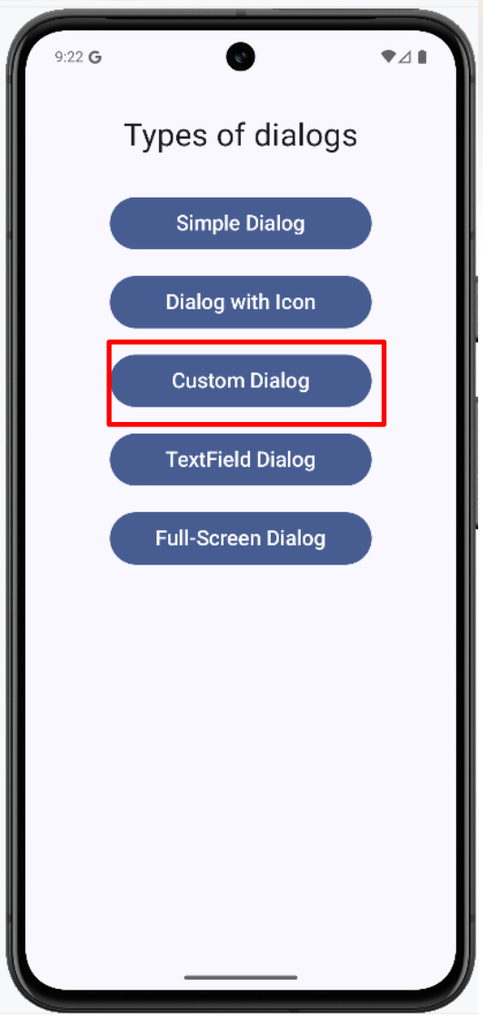


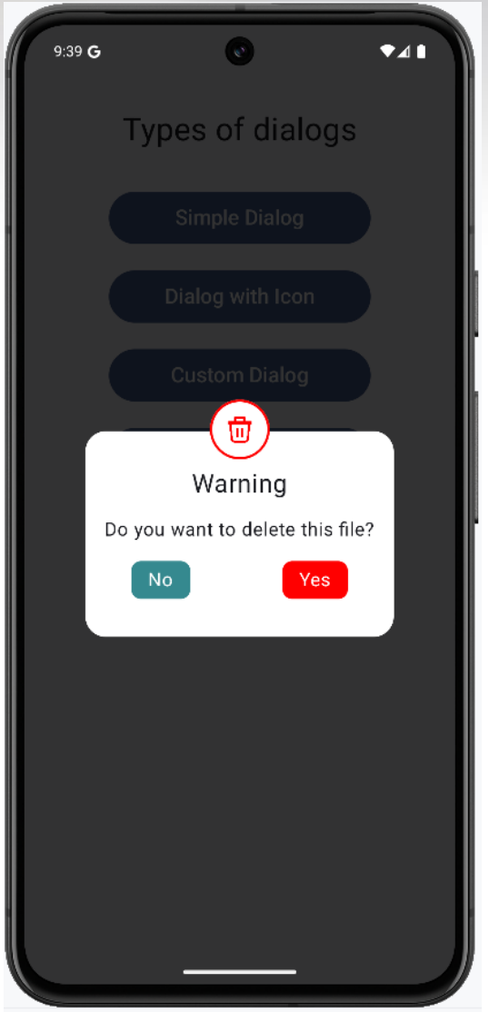
Alert Dialog Example: Dialog with Icon

```
////Dialog Icon
if (showDialogIcon){
    AlertDialog(
        onDismissRequest = { showDialogIcon = false },
        confirmButton = {
            TextButton(
                onClick = {
                    showDialogIcon = false
                    Toast.makeText(contextForToast,
                        text = "Yes", duration = Toast.LENGTH_SHORT).show()
                }
            ) { Text(text = "Yes") }
        },
        dismissButton = {
            TextButton(
                onClick = {
                    showDialogIcon = false
                    Toast.makeText(contextForToast,
                        text = "No", duration = Toast.LENGTH_SHORT).show()
                }
            ) {  Text(text = "No")  }
        },
        title = { Text(text = "Warning") },
        text = { Text(text = "Do you want to delete this file?") },
        icon = { Icon(imageVector = Icons.Default.Delete,contentDescription = null) }
    )
}
```



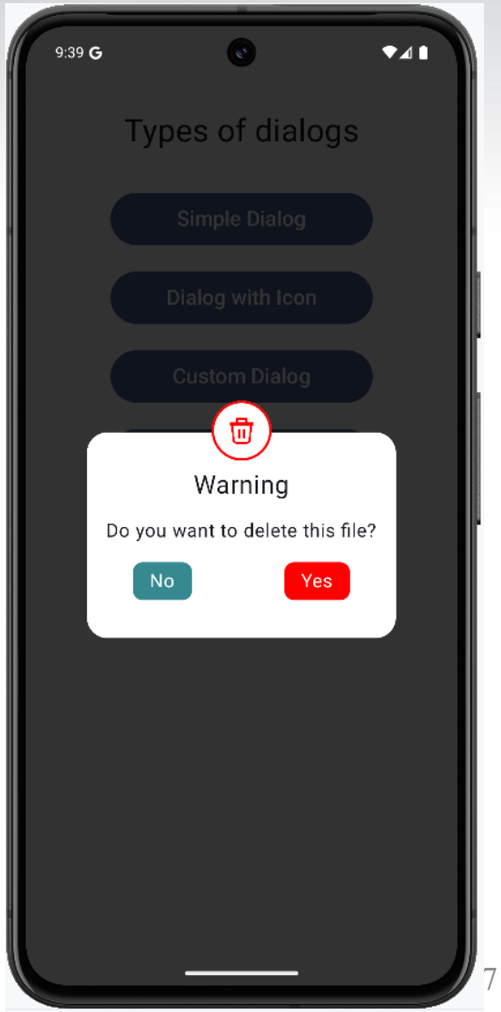
Alert Dialog Example: Custom Dialog





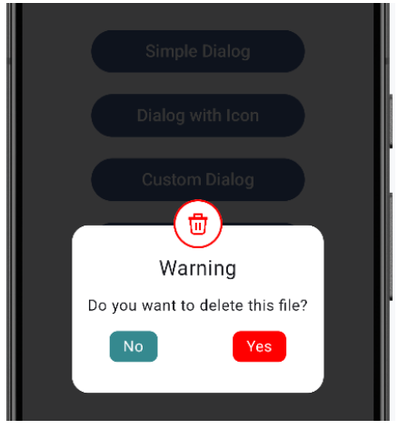
Alert Dialog Example: Custom Dialog

```
////Custom Dialog
if (showDialogCustom){
    val negativeButtonColor: Color = Color( color = 0xFF35898F)
    val positiveButtonColor: Color = Color( color = 0xFFFF0000)
    val spaceBetweenElements: Dp = 18.dp
    Dialog(onDismissRequest = {
        showDialogCustom = false
    }
    ) {
        Surface(
            modifier = Modifier.fillMaxWidth( fraction = 0.92f),
            color = Color.Transparent // dialog background
        ) {
            Box(
                modifier = Modifier
                    .fillMaxWidth()
                ) { // text and buttons
                Column(
                    modifier = Modifier
                        .padding(top = 30.dp) // this is the empty space at the top
                        .fillMaxWidth()
                        .background(
                            color = Color.White,
                            shape = RoundedCornerShape(percent = 10)
                        ),
                    horizontalAlignment = Alignment.CenterHorizontally
                ) {
```



Alert Dialog Example: Custom Dialog

```
Spacer(modifier = Modifier.height(height = 36.dp))
Text(
    text = "Warning",
    fontSize = 24.sp
)
Spacer(modifier = Modifier.height(height = spaceBetweenElements))

Text(
    modifier = Modifier.padding(horizontal = 16.dp),
    text = "Do you want to delete this file?",
    fontSize = 18.sp
)
Spacer(modifier = Modifier.height(height = spaceBetweenElements))
// buttons
```

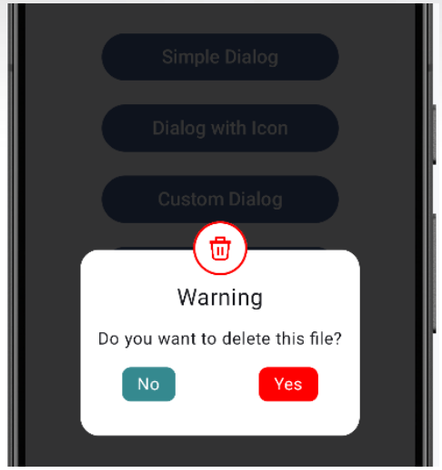


Call the DialogButton function to create a custom button.

```
// buttons
Row(
    modifier = Modifier.fillMaxWidth(),
    horizontalArrangement = Arrangement.SpaceAround
) {
    DialogButton(buttonColor = negativeButtonColor, buttonText = "No") {
        Toast
            .makeText(contextForToast, text = "No Click",
                duration = Toast.LENGTH_SHORT)
            .show()
        showDialogCustom = false
    }
    DialogButton(buttonColor = positiveButtonColor, buttonText = "Yes") {
        Toast.makeText(contextForToast, text = "Yes Click",
            duration = Toast.LENGTH_SHORT).show()
        showDialogCustom = false
    }
}
Spacer(modifier = Modifier.height(height = spaceBetweenElements * 2))
}
```

Alert Dialog Example: Custom Dialog

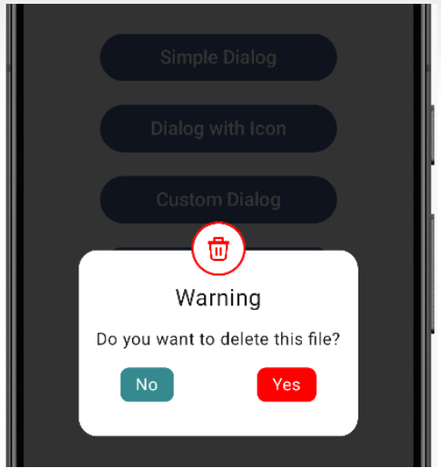
```
                // delete icon
                Icon(
                    painter = painterResource(id = R.drawable.trash),
                    contentDescription = "Delete Icon",
                    tint = positiveButtonColor,
                    modifier = Modifier
                        .background(color = Color.White, shape = CircleShape)
                        .border(width = 2.dp, shape = CircleShape, color = positiveButtonColor)
                        .padding(all = 16.dp)
                        .align(alignment = Alignment.TopCenter)
                        .size( size = 30.dp)
                )
            }
        }
    }
}
```



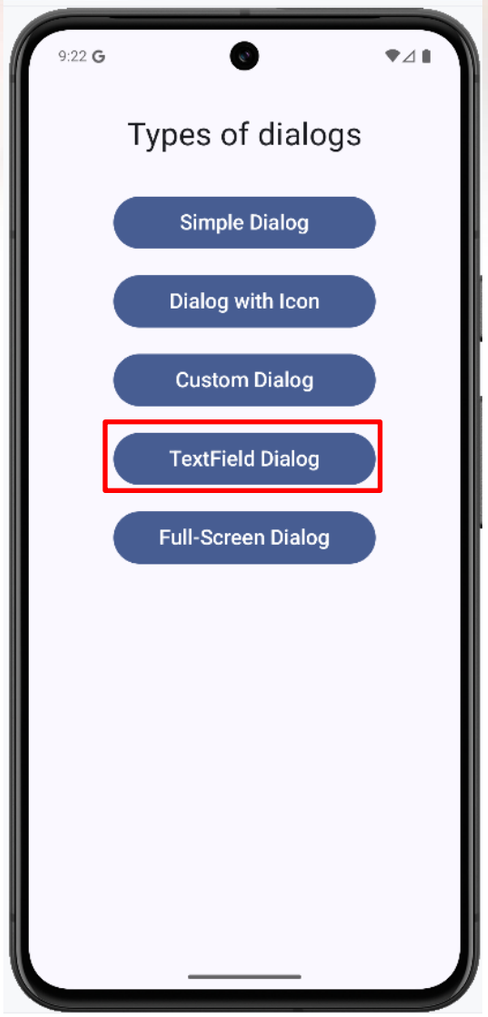
Alert Dialog Example: Custom Dialog

fun DialogButton

```
@Composable
fun DialogButton(
    cornerRadiusPercent: Int = 26,
    buttonColor: Color,
    buttonText: String,
    onDismiss: () -> Unit
) {
    Box(
        modifier = Modifier.background(
                color = buttonColor,
                shape = RoundedCornerShape(percent = cornerRadiusPercent)
            )
            .clickable { onDismiss()}
            .padding(horizontal = 16.dp, vertical = 6.dp)
    ) {
        Text(
            text = buttonText,
            color = Color.White,
            fontSize = 18.sp
        )
    }
}
```

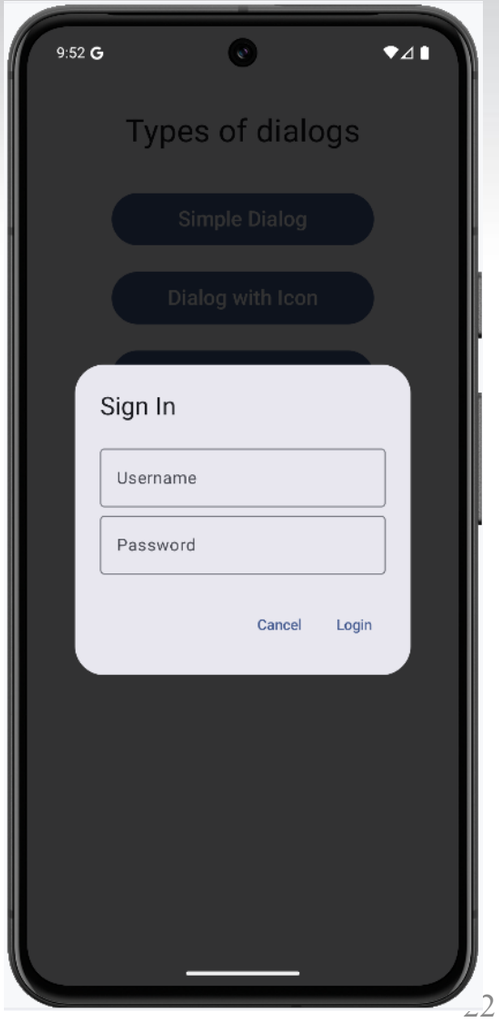


Alert Dialog Example: TextField Dialog



Alert Dialog Example: TextField Dialog

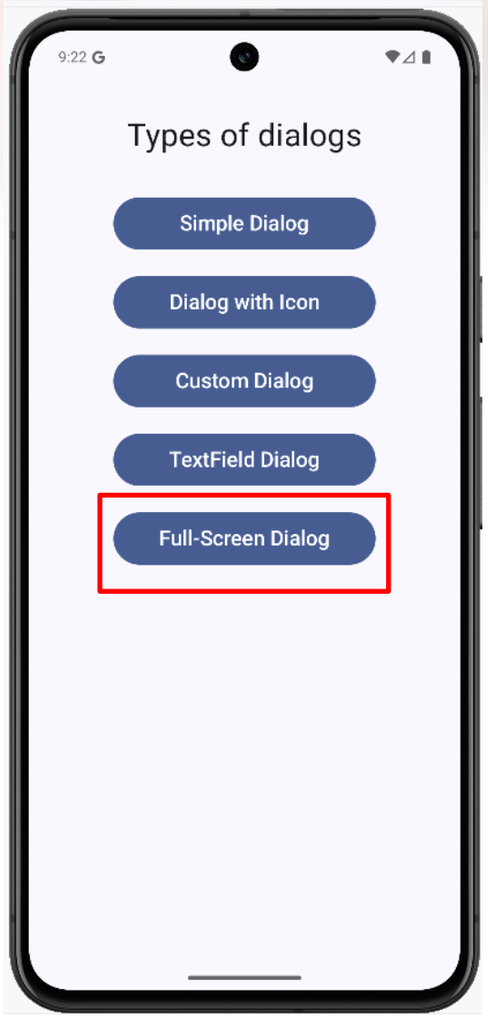
```
//////TextField Dialog
if (showDialogTextField){
    var textFieldUsername by remember { mutableStateOf( value = "") }
    var textFieldPassword by remember { mutableStateOf( value = "") }
    AlertDialog(
        onDismissRequest = { showDialogTextField = false },
        title = { Text(text = "Sign In") },
        text = {
            Column {

                OutlinedTextField(
                    value = textFieldUsername,
                    onValueChange = { textFieldUsername = it },
                    label = { Text( text = "Username") }
                )
                OutlinedTextField(
                    value = textFieldPassword ,
                    onValueChange = { textFieldPassword  = it },
                    label = { Text( text = "Password") },
                    visualTransformation = PasswordVisualTransformation()
                )

            }
        },
```

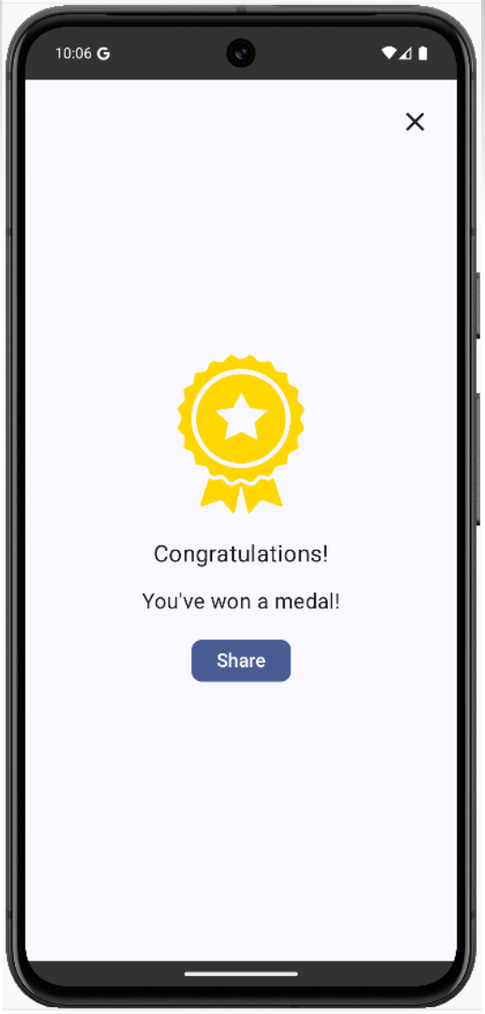


Alert Dialog Example: TextField Dialog

```
        confirmButton = {
            TextButton(
                onClick = {
                    showDialogTextField = false
                    Toast.makeText(contextForToast,
                        text = "Username: $textFieldUsername and Password: $textFieldPassword ",
                        duration = Toast.LENGTH_SHORT).show()
                    // Clear value
                    textFieldUsername=""
                    textFieldPassword = ""
                }
            ) { Text( text = "Login") }
        },
        dismissButton = {
            TextButton(onClick = { showDialogTextField = false
                textFieldUsername=""
                textFieldPassword = ""
            }
            ){  Text( text = "Cancel") }
        }
    )
}
```

Alert Dialog Example: Full-Screen Dialog

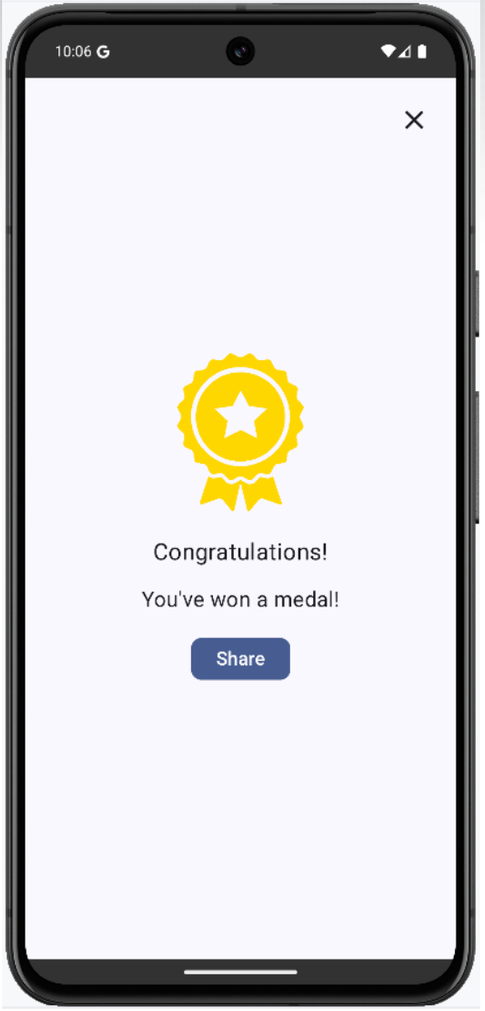




Alert Dialog Example: Full-Screen Dialog

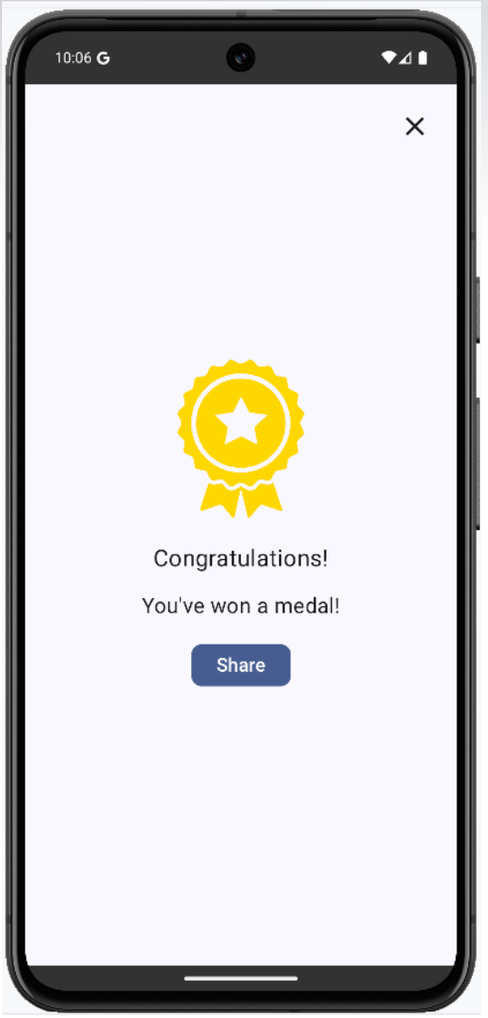
```
///// Full-Screen Dialog
if (showDialogFull) {
    Dialog(
        onDismissRequest = { showDialogFull = false },
        properties = DialogProperties(
            usePlatformDefaultWidth = false // Enable full screen width
        )
    ) {
        Surface(modifier = Modifier.fillMaxSize()) {
            Box(modifier = Modifier.fillMaxSize()) {
                Column(modifier = Modifier.fillMaxSize()
                        .padding( all = 16.dp),
                    verticalArrangement = Arrangement.Center,
                    horizontalAlignment = Alignment.CenterHorizontally
                ) {
                    // Medal Icon
                    Icon(
                        painter = painterResource(id = R.drawable.medal_icon),
                        contentDescription = "Medal icon",
                        modifier = Modifier.size(size = 150.dp),
                        tint = Color( color = 0xFFFFD700) // Gold color
                    )
```

medal_icon.png



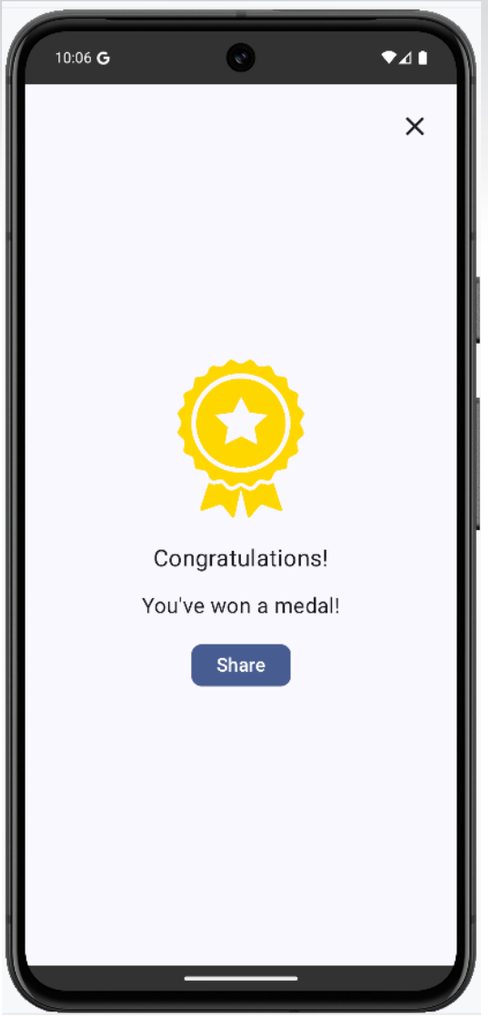
Alert Dialog Example: Full-Screen Dialog

```
        Text(
            text = "Congratulations!",
            fontSize = 22.sp,
            modifier = Modifier.padding(top = 26.dp)
        )
        Text(
            text = "You've won a medal!",
            fontSize = 20.sp,
            modifier = Modifier.padding(top = 20.dp),
        )
        Button(
            onClick = {
                Toast.makeText(contextForToast, text = "Shared!",
                    duration = Toast.LENGTH_SHORT).show()
            },
            modifier = Modifier.padding(top = 20.dp),
            shape = RoundedCornerShape(percent = 25)
        ) {
            Text(text = "Share", fontSize = 18.sp)
        }
    }
```



Alert Dialog Example: Full-Screen Dialog

```
                    //  Close Button (Top-Right)
                    IconButton(
                        onClick = { showDialogFull = false },
                        modifier = Modifier
                            .align(Alignment.TopEnd)
                            .padding( all = 16.dp)
                    ) {
                        Icon(
                            imageVector = Icons.Default.Close,
                            contentDescription = "Close",
                            modifier = Modifier.size( size = 30.dp)
                        )
                    }
                }
            }
        }
    }
}
```

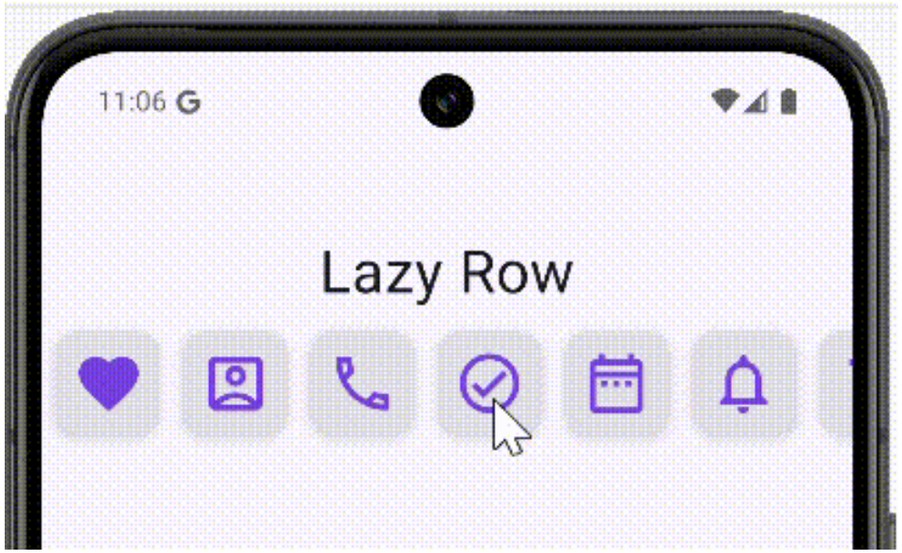


Lists and Grids View

- Lazy Row
- Lazy Column
- Lazy Grid Layouts
- Lazy VerticalStaggeredGrid

Lazy Row

- It is used to display a horizontally scrolling list.
- It only renders the currently visible items on the screen.



Lazy Row: Example

Application Structure:

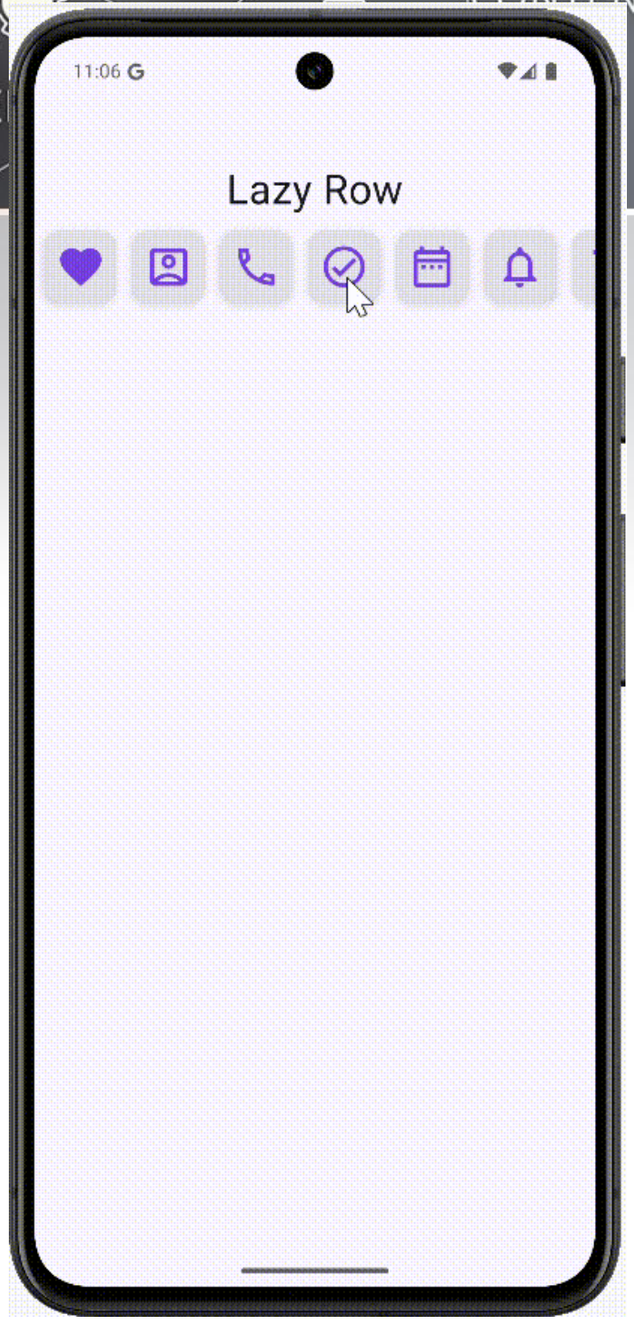
- Data Class: OptionsList()
- private fun prepareOptionsList()
- @Composable fun MyLazyRow()
- @Composable private fun ItemLayoutRow()

Lazy Row: Example

```
data class OptionsList(val icon: ImageVector, val option: String)

// add items to the list
2 Usages
private fun prepareOptionsList(): MutableList<OptionsList> {
    val optionsList = mutableListOf<OptionsList>()
    optionsList.add(OptionsList(icon = Icons.Outlined.Favorite, option = "Favorite"))
    optionsList.add(OptionsList(icon = Icons.Outlined.AccountBox, option = "Account"))
    optionsList.add(OptionsList(icon = Icons.Outlined.Call, option = "Call"))
    optionsList.add(OptionsList(icon = Icons.Outlined.CheckCircle, option = "Check"))
    optionsList.add(OptionsList(icon = Icons.Outlined.DateRange, option = "Date"))
    optionsList.add(OptionsList(icon = Icons.Outlined.Notifications, option = "Notifications"))
    optionsList.add(OptionsList(icon = Icons.Outlined.Delete, option = "Delete"))
    optionsList.add(OptionsList(icon = Icons.Outlined.ThumbUp, option = "ThumbUp"))
    optionsList.add(OptionsList(icon = Icons.Outlined.ShoppingCart, option = "ShoppingCart"))
    optionsList.add(OptionsList(icon = Icons.Outlined.Share, option = "Share"))

    return optionsList
}
```

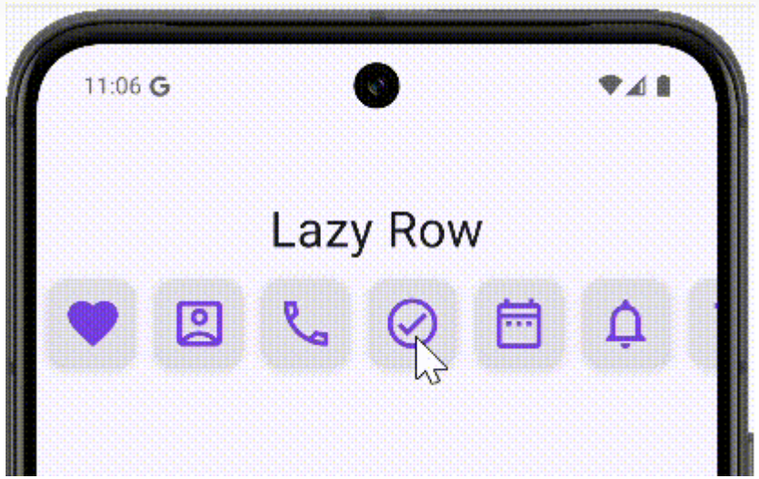
Lazy Row: Example

```
@Composable
fun MyLazyRow(modifier : Modifier) {
    val optionsList = remember {prepareOptionsList()}
    Column {
        Spacer(modifier = modifier.height(height = 20.dp))
        Text(modifier = Modifier.align(Alignment.CenterHorizontally),
            text = "Lazy Row",
            fontSize = 30.sp
        )
        LazyRow(
            horizontalArrangement = Arrangement.spacedBy(space = 12.dp),
            contentPadding = PaddingValues(horizontal = 8.dp, vertical = 12.dp)
        ) {
            items( items = optionsList) { item ->
                ItemLayoutRow(optionsList = item)
            }
        }
    }
}
```



Lazy Row: Example

```
@Composable
private fun ItemLayoutRow(
    optionsList: OptionsList,
    context: Context = LocalContext.current.applicationContext
) {
    Card( shape = RoundedCornerShape(size = 12.dp),
        elevation = CardDefaults.cardElevation(
            defaultElevation = 4.dp
        )
    ) {
        Box(modifier = Modifier.fillMaxWidth()
                .clickable {
                    Toast.makeText(context, text = optionsList.option,
                        duration = Toast.LENGTH_SHORT).show()
                }
                .padding(all = 8.dp),
        ) {
            Icon(modifier = Modifier.size(size = 36.dp),
                imageVector = optionsList.icon,
                contentDescription = null,
                tint = Color( color = 0xFF7850E8)
            )
        }
    }
}
```



Lazy Column

- It is a composable that displays a vertically scrolling list of items.
- Unlike traditional list views, Lazy Column only renders the items that are currently visible on the screen and nearby, optimizing performance and memory usage for large datasets.

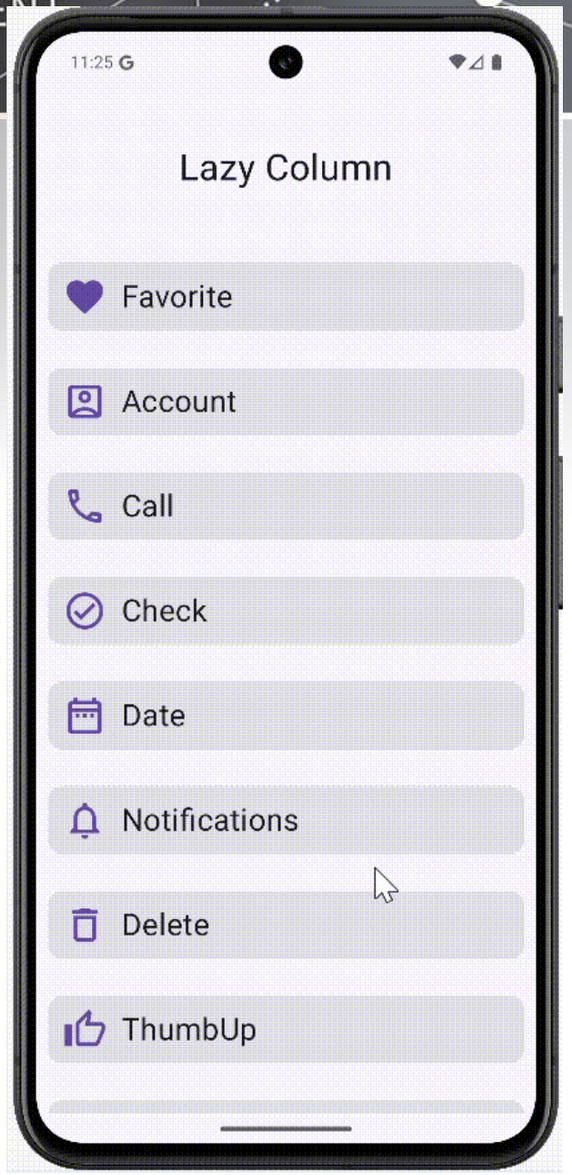
Lazy Column: Example

Application Structure:

- Data Class: OptionsList()
- private fun prepareOptionsList()
- @Composable fun MyLazyColumn()
- @Composable private fun ItemLayoutColumn()

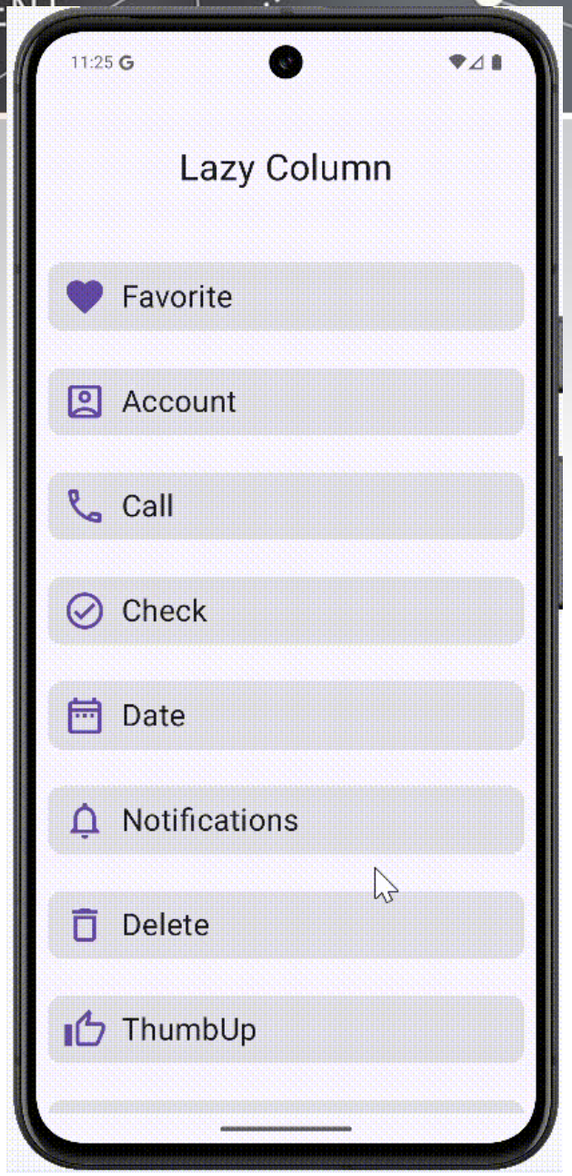
Lazy Column : Example

```
data class OptionsList(val icon: ImageVector, val option: String)

// add items to the list
private fun prepareOptionsList(): MutableList<OptionsList> {
    val optionsList = mutableListOf<OptionsList>()
    optionsList.add(OptionsList(icon = Icons.Outlined.Favorite, option = "Favorite"))
    optionsList.add(OptionsList(icon = Icons.Outlined.AccountBox, option = "Account"))
    optionsList.add(OptionsList(icon = Icons.Outlined.Call, option = "Call"))
    optionsList.add(OptionsList(icon = Icons.Outlined.CheckCircle, option = "Check"))
    optionsList.add(OptionsList(icon = Icons.Outlined.DateRange, option = "Date"))
    optionsList.add(OptionsList(icon = Icons.Outlined.Notifications, option = "Notifications"))
    optionsList.add(OptionsList(icon = Icons.Outlined.Delete, option = "Delete"))
    optionsList.add(OptionsList(icon = Icons.Outlined.ThumbUp, option = "ThumbUp"))
    optionsList.add(OptionsList(icon = Icons.Outlined.ShoppingCart, option = "ShoppingCart"))
    optionsList.add(OptionsList(icon = Icons.Outlined.Share, option = "Share"))

    return optionsList
}
```

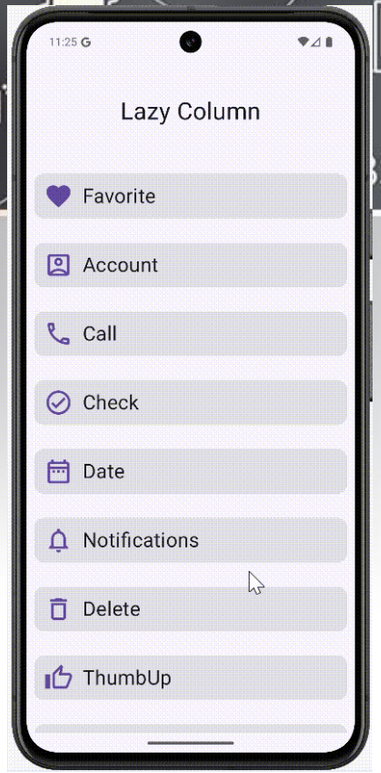


Lazy Column : Example

```
@Composable
fun MyLazyColumn(modifier : Modifier) {
    val optionsList = remember { prepareOptionsList() }
    Column {
        Spacer(modifier = modifier.height(height = 20.dp))
        Text(modifier = Modifier.align(Alignment.CenterHorizontally),
            text = "Lazy Column",
            fontSize = 30.sp
        )
    LazyColumn(
        modifier = modifier.fillMaxSize(),
        verticalArrangement = Arrangement.spacedBy(space = 30.dp), // gap between items
        contentPadding = PaddingValues(all = 10.dp) // padding for LazyColumn layout
    ) {
            items( items = optionsList) { item ->
                ItemLayoutColumn(optionsList = item)
            }
        }
    }
}
```



Lazy Column : Example




```
@Composable
private fun ItemLayoutColumn(
    optionsList: OptionsList,
    context: Context = LocalContext.current.applicationContext
) {
    Card(
        shape = RoundedCornerShape(size = 12.dp)
    ) {
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .clickable {
                    Toast
                        .makeText(context, text = optionsList.option,
                            duration = Toast.LENGTH_SHORT).show()
                }
                .padding(vertical = 10.dp, horizontal = 12.dp),
            verticalAlignment = Alignment.CenterVertically
        ) {
```

```
            Icon(
                modifier = Modifier.size(size = 36.dp),
                imageVector = optionsList.icon,
                contentDescription = null,
                tint = Color( color = 0xFF6650a4)
            )
            Spacer(modifier = Modifier.width(width = 12.dp))
            Text(
                text = optionsList.option,
                fontSize = 25.sp
            )
        }
    }
}
```

Lazy grids

- The LazyVerticalGrid and LazyHorizontalGrid composables provide support for displaying items in a grid.
- A Lazy vertical grid will display its items in a vertically scrollable container, spanned across multiple columns, while the Lazy horizontal grids will have the same behaviour on the horizontal axis.

Lazy grids

- The number of columns depends on the column's parameter. There are two options – **Fixed** and **Adaptive**.

1. Fixed Columns:

GridCells.Fixed() class creates a grid with fixed number of columns.

```
class Fixed(private val count: Int) : GridCells {}
```

- The count parameter determines the number of columns.
- For example, if count = 5, there will be five columns. Each column will have a width of 1/5th of the parent's width.
- For example, if the parent's width is 100 pixels, each column will have a width of 20 pixels. The columns will be evenly distributed.

Lazy grids

2. Adaptive Columns:

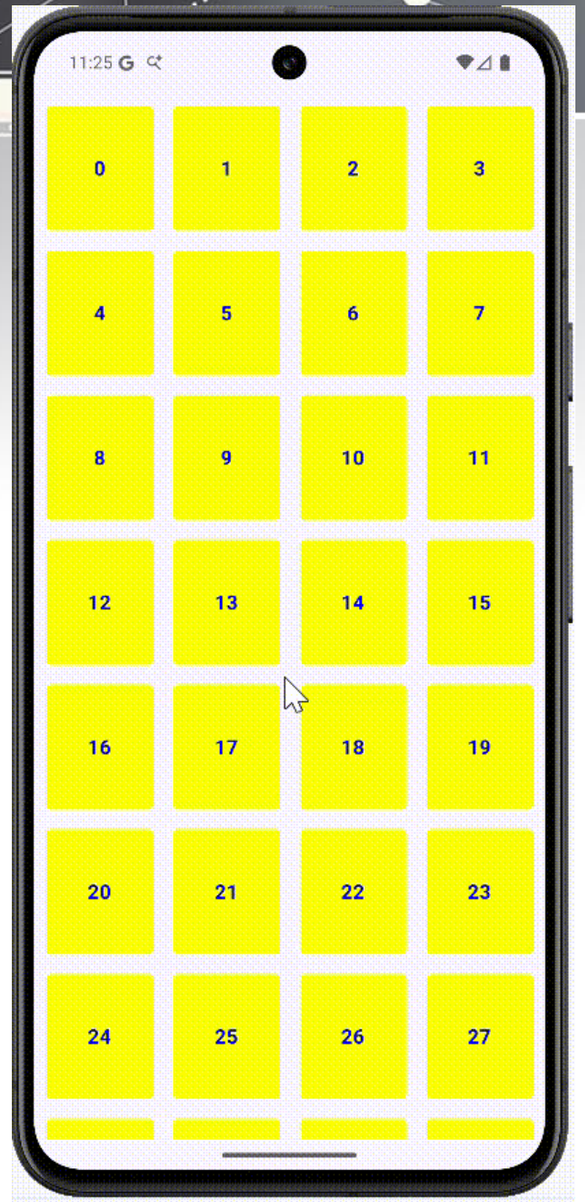
GridCells.Adaptive() class creates a grid with adaptive size.

```
class Adaptive(private val minSize: Dp) : GridCells {}
```

- It creates as many columns as possible with the condition that each column has at least minSize width. All the columns have equal width.
- For example, if minSize = 20.dp, there will be as many columns as possible, and every column has a width of at least 20.dp. If the screen is 88.dp wide, there will be four columns with 22.dp each.

Lazy grids: Lazy Vertical Grid

```
@Composable
fun MyLazyVerticalGrid() {
    LazyVerticalGrid(
        modifier = Modifier.fillMaxSize(),
        columns = GridCells.Fixed(count = 4), // fix size
        verticalArrangement = Arrangement.spacedBy(space = 16.dp), // top and bottom margin between each item
        horizontalArrangement = Arrangement.spacedBy(space = 16.dp), // left and right margin between each item
        contentPadding = PaddingValues(all = 10.dp) // margin for the whole layout
    ) { this: LazyGridScope
        items(count = 100) { this: LazyGridItemScope index ->
            Box(
                modifier = Modifier
                    .fillMaxWidth()
                    .height(height = 100.dp)
                    .background(color = Color.Yellow, shape = RoundedCornerShape(size = 4.dp)),
                contentAlignment = Alignment.Center
            ) { this: BoxScope
                Text(
                    text = "$index",
                    color = Color.Blue,
                    fontWeight = FontWeight.Bold
                )
            }
        }
    }
}
```

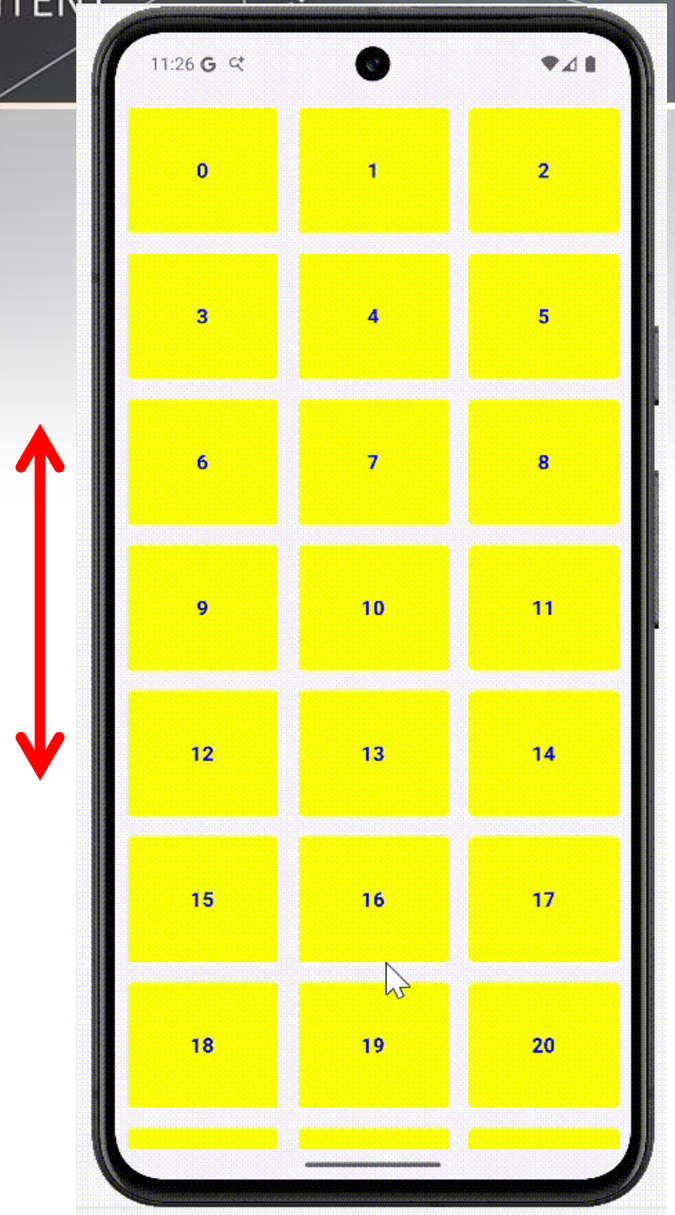


Fix size

Lazy grids: Lazy Vertical Grid

```
@Composable
fun MyLazyVerticalGrid() {
    LazyVerticalGrid(
        modifier = Modifier.fillMaxSize(),
        columns = GridCells.Adaptive(minSize = 100.dp), // adaptive size
        verticalArrangement = Arrangement.spacedBy(space = 16.dp), // top and bottom margin between each item
        horizontalArrangement = Arrangement.spacedBy(space = 16.dp), // left and right margin between each item
        contentPadding = PaddingValues(all = 10.dp) // margin for the whole layout
    ) { this: LazyGridScope
        items(count = 100) { this: LazyGridItemScope index ->
            Box(
                modifier = Modifier
                    .fillMaxWidth()
                    .height(height = 100.dp)
                    .background(color = Color.Yellow, shape = RoundedCornerShape(size = 4.dp)),
                contentAlignment = Alignment.Center
            ) { this: BoxScope
                Text(
                    text = "$index",
                    color = Color.Blue,
                    fontWeight = FontWeight.Bold
                )
            }
        }
    }
}
```

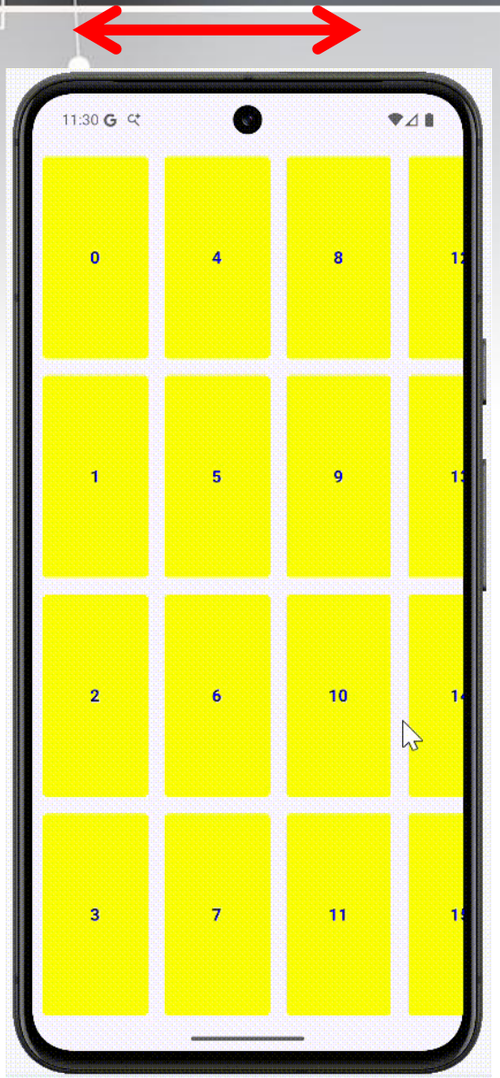
Install successfully finished in 5 s 348 ms



Adaptive size

Lazy grids: Lazy Horizontal Grid

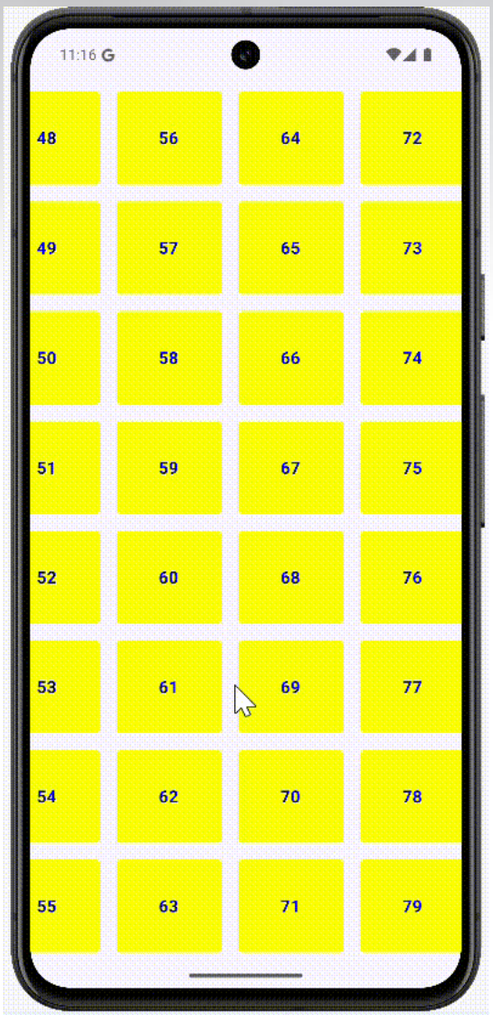
```
@Composable
fun MyLazyHorizontalGrid() {
    LazyHorizontalGrid(
        modifier = Modifier.fillMaxSize(),
        rows = GridCells.Fixed(count = 4),
        verticalArrangement = Arrangement.spacedBy(space = 16.dp), // top and bottom margin for each item
        horizontalArrangement = Arrangement.spacedBy(space = 16.dp), // left and right margin for each item
        contentPadding = PaddingValues(all = 10.dp) // margin for the whole layout
    ) { this: LazyGridScope
        items(count = 100) { this: LazyGridItemScope index ->
            Box(
                modifier = Modifier
                    .fillMaxHeight()
                    .width(width = 100.dp)
                    .background(color = Color.Yellow, shape = RoundedCornerShape(size = 4.dp)),
                contentAlignment = Alignment.Center
            ) { this: BoxScope
                Text(
                    text = "$index",
                    color = Color.Blue,
                    fontWeight = FontWeight.Bold
                )
            }
        }
    }
}
```



Fix size

Lazy grids: Lazy Horizontal Grid

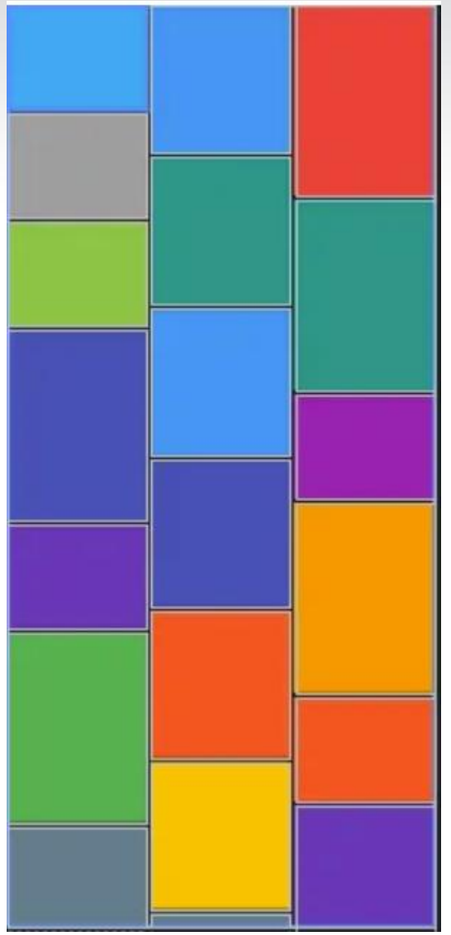
```
@Composable
fun MyLazyHorizontalGrid() {
    LazyHorizontalGrid(
        modifier = Modifier.fillMaxSize(),
        rows = GridCells.Adaptive(minSize = 80.dp),
        verticalArrangement = Arrangement.spacedBy(space = 16.dp), // top and bottom margin for each item
        horizontalArrangement = Arrangement.spacedBy(space = 16.dp), // left and right margin for each item
        contentPadding = PaddingValues(all = 10.dp) // margin for the whole layout
    ) { this: LazyGridScope
        items(count = 100) { this: LazyGridItemScope index ->
            Box(
                modifier = Modifier
                    .fillMaxHeight()
                    .width(width = 100.dp)
                    .background(color = Color.Yellow, shape = RoundedCornerShape(size = 4.dp)),
                contentAlignment = Alignment.Center
            ) { this: BoxScope
                Text(
                    text = "$index",
                    color = Color.Blue,
                    fontWeight = FontWeight.Bold
                )
            }
        }
    }
}
```



Adaptive size

Lazy grids: LazyVerticalStaggeredGrid

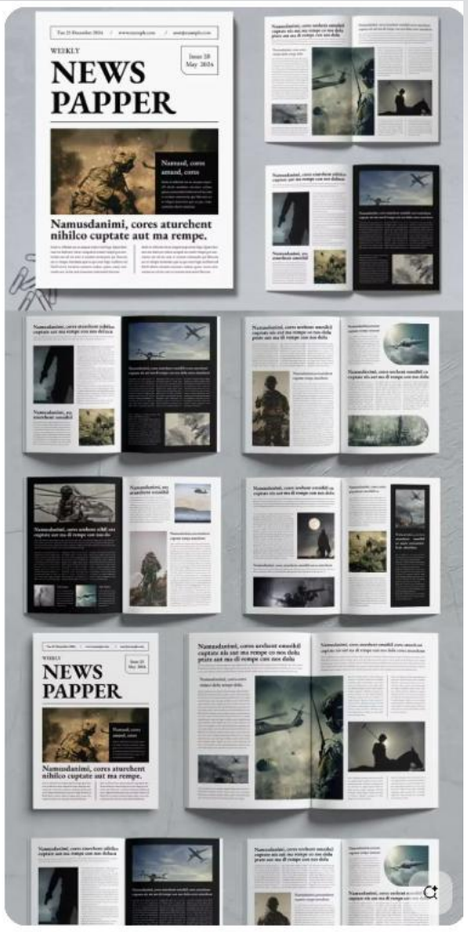
- A composable that displays items in a vertical grid where items can have different heights
- Unlike a standard grid, rows do not need to be aligned. Items are placed in the next available space (Masonry layout).
- Best known as the "Pinterest" or "Google Keep" style layout.



Lazy grids: LazyVerticalStaggeredGrid

Common Use Cases:

- Photo Galleries: Displaying images with different aspect ratios (Portrait/Landscape mixed).
- Note-Taking Apps: Like Google Keep, where note length varies based on text content.
- News Feeds: Mixing short headlines with long articles or cards.
- Product Catalogs: E-commerce apps displaying products where image sizes might vary.

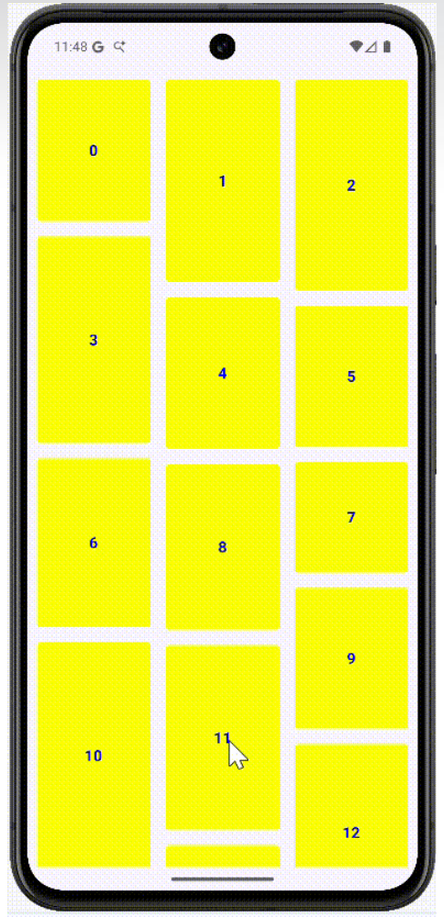


Lazy grids: LazyVerticalStaggeredGrid

***import androidx.compose.foundation.lazy.staggeredgrid.items

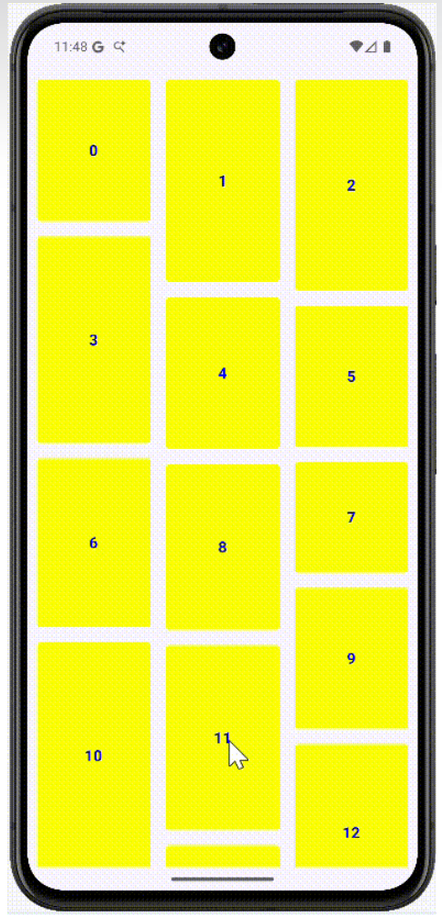
```
data class StaggeredItem(
    val index: Int,
    val height: Dp
)
```

```
@Composable
fun MyLazyVerticalStaggeredGrid(modifier: Modifier = Modifier) {
    // Generate 100 items with random heights (100-250dp).
    val itemsList = remember {
        List( size = 100) { index ->
            StaggeredItem(
                index = index,
                height = Random.nextInt( from = 100,  until = 250).dp
            )
        }
    }
```

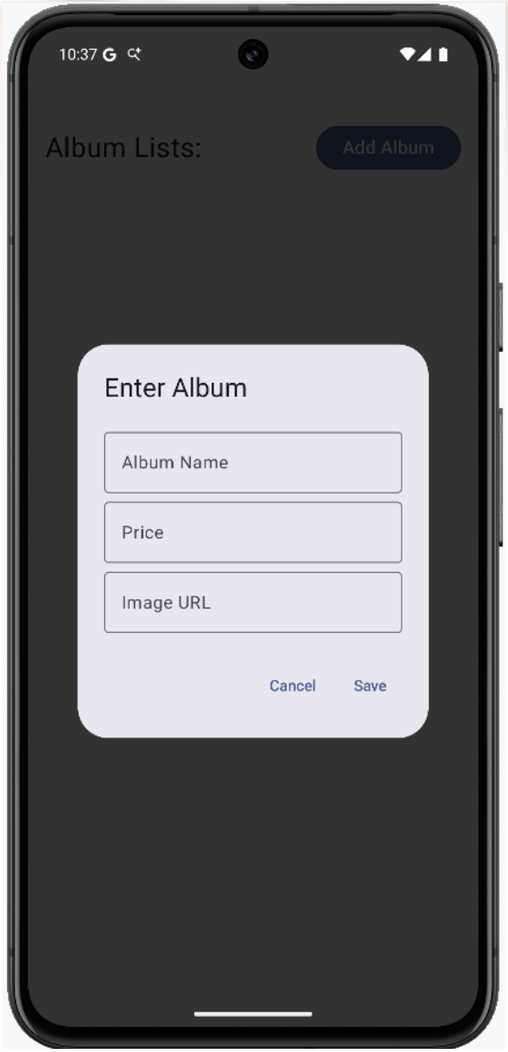


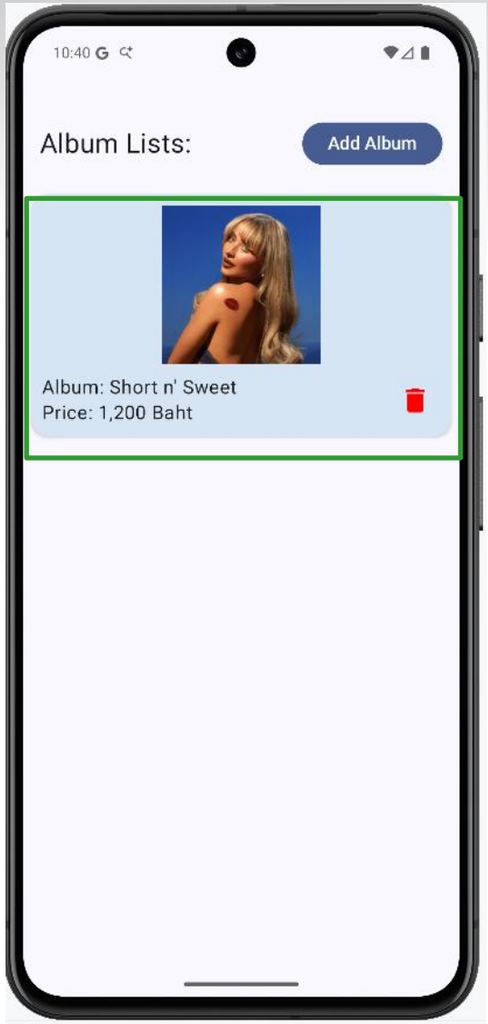
Lazy grids: LazyVerticalStaggeredGrid

```
    LazyVerticalStaggeredGrid(
        columns = StaggeredGridCells.Adaptive(minSize = 100.dp),
        // or StaggeredGridCells.Fixed(2) for exact column count.

        modifier = modifier.fillMaxSize(),
        verticalItemSpacing = 16.dp,
        horizontalArrangement = Arrangement.spacedBy( space = 16.dp),
        contentPadding = PaddingValues(all = 10.dp)
    ) {
        items(itemsList) { item ->
            Box(modifier = Modifier.fillMaxWidth()
                    .height(item.height) //Apply random height to stagger items.
                    .background(
                        color = Color.Yellow,
                        shape = RoundedCornerShape(size = 4.dp)
                    ),
                contentAlignment = Alignment.Center
            ) {
                Text(text = "${item.index}",
                    color = Color.Blue,
                    fontWeight = FontWeight.Bold
                )
            }
        }
    }
}
```

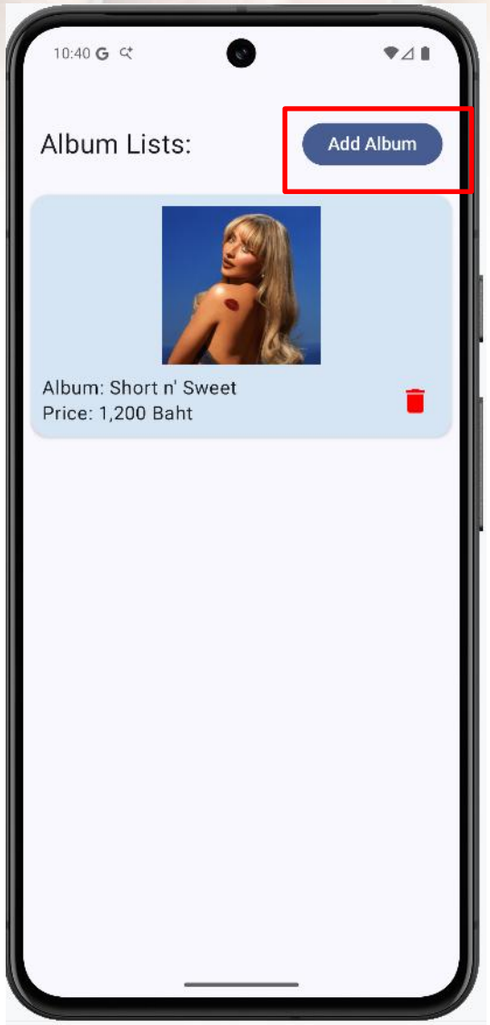


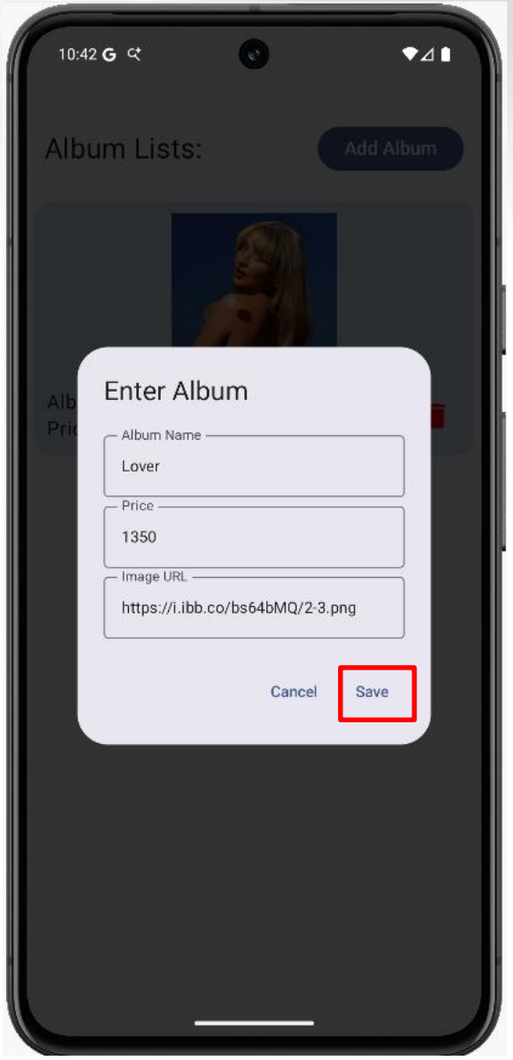
Lazy Column and Dialog : Example

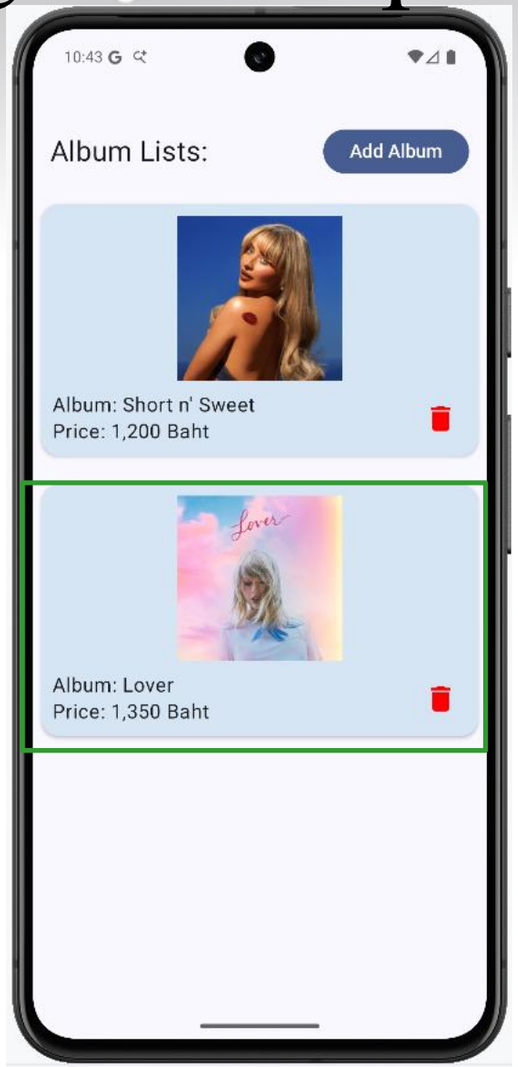




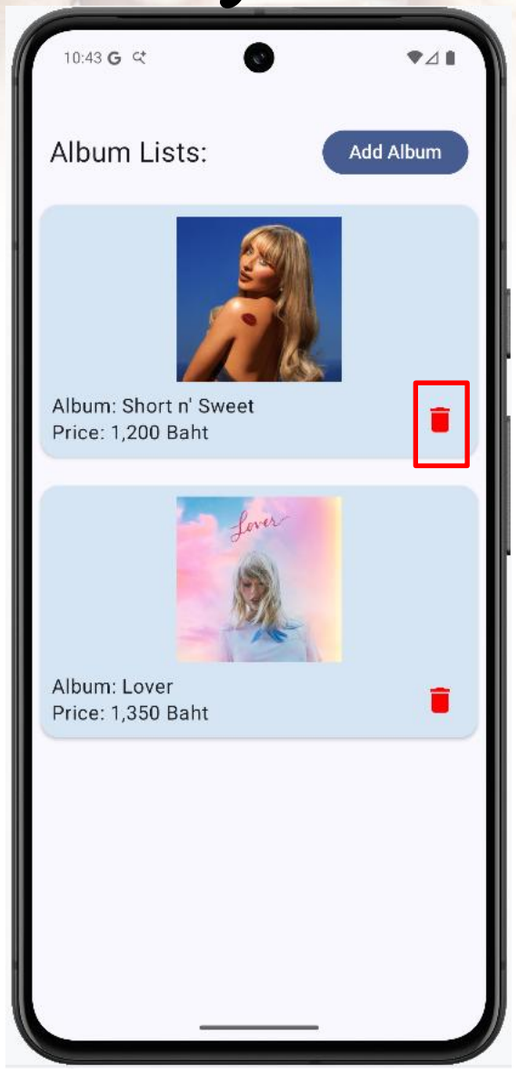
Lazy Column and Dialog : Example

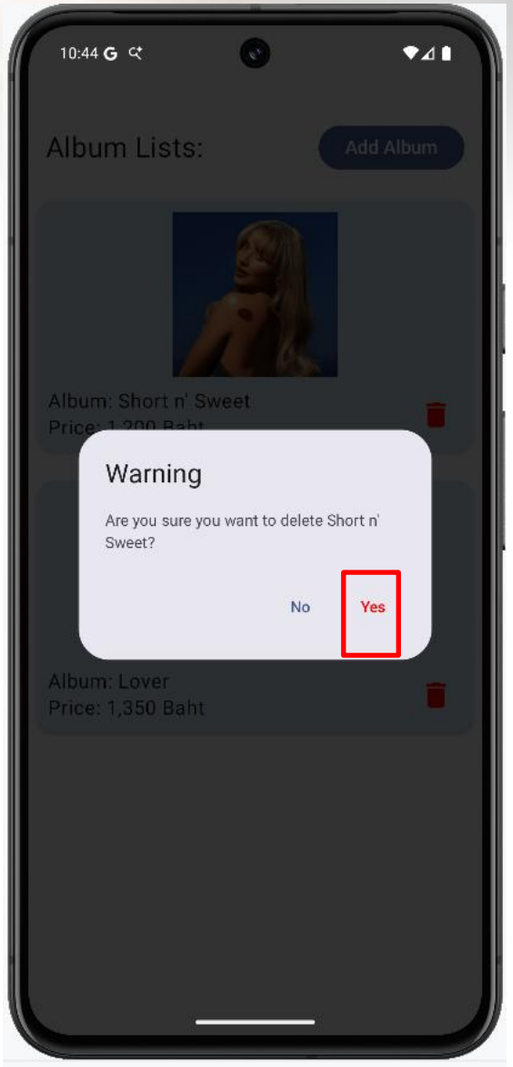


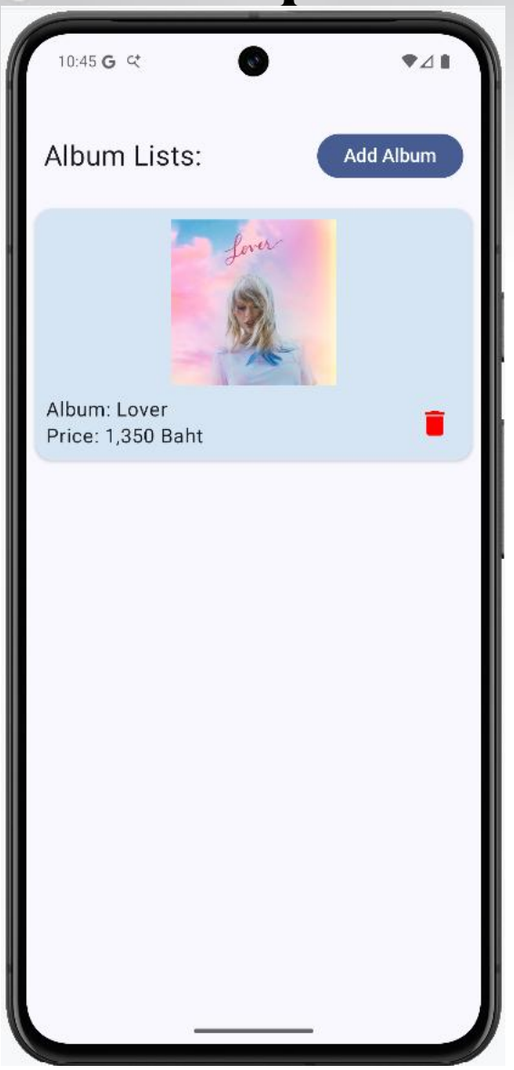




Lazy Column and Dialog : Example







Internet connecting and image show

- For use internet : App>> manifests >> Androidmanifest.xml

```
<uses-permission android:name="android.permission.INTERNET"/>
```

- Use Coil library to show image

```
implementation("io.coil-kt:coil-compose:2.6.0")
```

Lazy Column and Dialog : Example

- For use internet : App>> manifests >> Androidmanifest.xml

<uses-permission android:name="android.permission.INTERNET"/>

MainActivity.kt | AndroidManifest.xml | build.gradle.kts (:app)

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools">
4      <uses-permission android:name="android.permission.INTERNET"/>
5
6      <application
7          android:allowBackup="true"
8          android:dataExtractionRules="@xml/data_extraction_rules"
```

Lazy Column and Dialog : Example

- build.gradle.kts(Module: app)

  Add implementation ("io.coil-kt:coil-compose:2.6.0 ") And Sync Now

```
dependencies {
    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.lifecycle.runtime.ktx)
    implementation(libs.androidx.activity.compose)

    implementation("io.coil-kt:coil-compose:2.6.0")

    implementation(platform( dependencyProvider = libs.androidx.compose.bom))
    implementation(libs.androidx.compose.ui)
    implementation(libs.androidx.compose.ui.graphics)
```

Lazy Column and Dialog : Example

Application Structure:

- Data Class: AlbumClass.kt
- @Composable fun MyScreen()

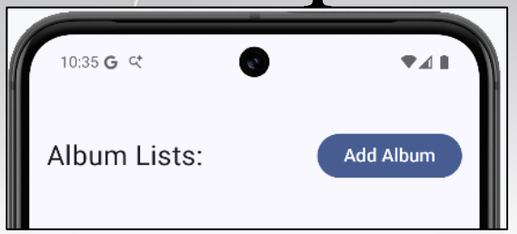
Lazy Column and Dialog : Example

AlbumClass.kt

```
data class AlbumClass(
    var name:String,
    var price: Int,
    var image: String){}
```

image: https://i.ibb.co/bs64bMQ/2-3.png

Lazy Column and Dialog : Example




```
@Composable
fun MyScreen(modifier : Modifier) {
    val contextForToast = LocalContext.current.applicationContext
    // List of data
    val albumItemsList = remember { mutableStateListOf<AlbumClass>() }

    // Add Dialog States
    var addDialog by rememberSaveable() { mutableStateOf( value = false) }
    var textFieldName by rememberSaveable() { mutableStateOf( value = "") }
    var textFieldPrice by rememberSaveable() { mutableStateOf( value = "") }
    var textFieldImage by rememberSaveable() { mutableStateOf( value = "") }

    // Delete Dialog States
    var deleteDialog by rememberSaveable() { mutableStateOf( value = false) }

    // State to track "which item" is being deleted (Very Important)
    var itemToDelete by rememberSaveable() { mutableStateOf<AlbumClass?>( value = null) }
```

```
// Add fillMaxSize to fill the screen
Column(modifier = modifier.fillMaxSize()) {
    Spacer(modifier = Modifier.height( height = 20.dp))
    Row(
        Modifier
            .fillMaxWidth()
            .padding( all = 16.dp),
        verticalAlignment = Alignment.CenterVertically
    ) {
        Text(
            text = "Album Lists:",
            fontSize = 25.sp,
            modifier = Modifier.weight( weight = 1f)
        )
        Button(onClick = { addDialog = true }) {
            Text( text = "Add Album", fontSize = 17.sp)
        }
    }
```

Lazy Column and Dialog : Example

```
// Display List
LazyColumn(
    verticalArrangement = Arrangement.spacedBy( space = 10.dp),
) {
    itemsIndexed(items = albumItemsList) { index, item ->
        Card(
            modifier = Modifier
                .padding(horizontal = 8.dp, vertical = 8.dp)
                .fillMaxWidth(),
            colors = CardDefaults.cardColors(containerColor = Color( color = 0xFFD5E5F3)),
            elevation = CardDefaults.cardElevation(defaultElevation = 2.dp),
            shape = RoundedCornerShape(corner = CornerSize( size = 16.dp)),
            onClick = {
                Toast.makeText(contextForToast, text = "Click on ${item.name}.",
                    duration = Toast.LENGTH_SHORT).show()
            }
        ) {
```


```
            Column(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding( all = 10.dp),
                horizontalAlignment = Alignment.CenterHorizontally
            ) {
                AsyncImage(
                    model = item.image,
                    placeholder = painterResource(id = R.drawable.place_holder),
                    error = painterResource(id = R.drawable.error_image),
                    contentDescription = "album image",
                    modifier = Modifier.size( size = 150.dp)
                )
```

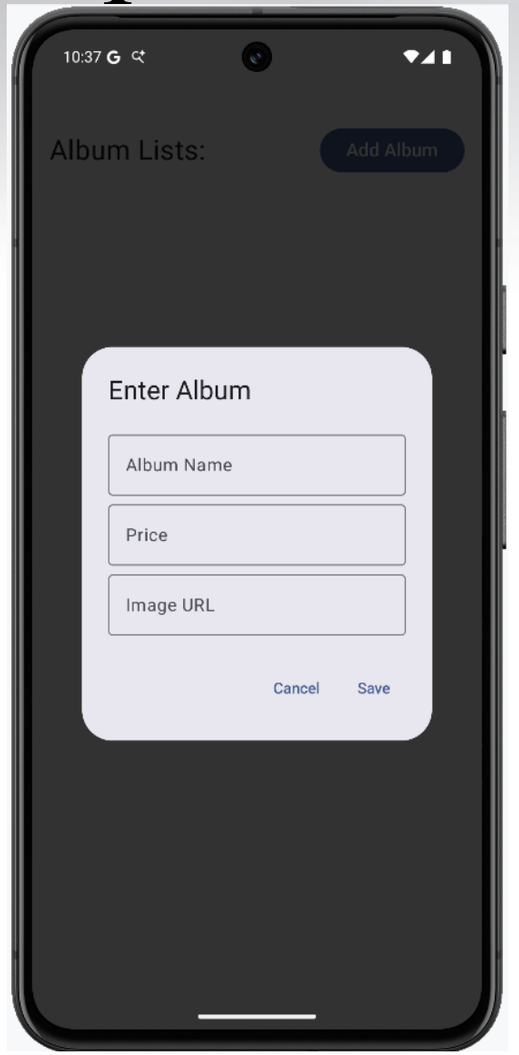
Lazy Column and Dialog : Example

```
Row(
    Modifier
        .fillMaxWidth()
        .padding(top = 10.dp),
    verticalAlignment = Alignment.CenterVertically
) {
    Text(
        text = "Album: ${item.name}\nPrice: ${String.format("%,d", item.price)} Baht",
        fontSize = 18.sp,
        modifier = Modifier.weight( weight = 1f)
    )
    IconButton(
        onClick = {
            itemToDelete = item // Remember which item to delete
            deleteDialog = true // Open Dialog

            Toast.makeText(contextForToast, text = "Click delete ${item.name}",
                duration = Toast.LENGTH_SHORT).show()
        }
    ) {
```


```
                Icon(
                    imageVector = Icons.Default.Delete,
                    contentDescription = "Delete Icon",
                    tint = Color.Red,
                    modifier = Modifier.size( size = 30.dp)
                )
            }
        }
    }
}
```

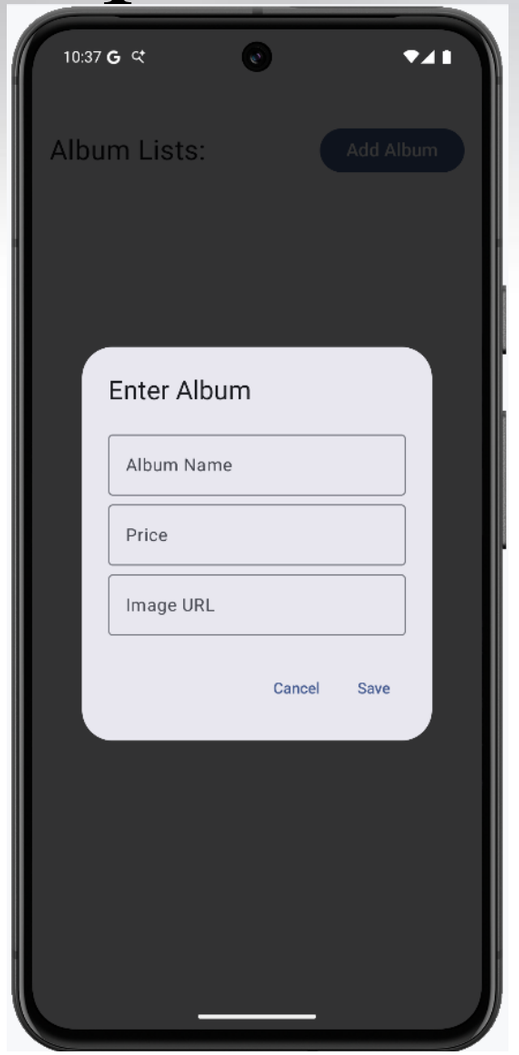
Lazy Column and Dialog : Example

```
// Add Dialog
if (addDialog) {
    AlertDialog(
        onDismissRequest = { addDialog = false },
        title = { Text(text = "Enter Album") },
        text = {
            Column {
                OutlinedTextField(
                    value = textFieldName,
                    onValueChange = { textFieldName = it },
                    label = { Text( text = "Album Name") }
                )
                OutlinedTextField(
                    value = textFieldPrice,
                    onValueChange = { textFieldPrice = it },
                    label = { Text( text = "Price") },
                    keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number)
                )
                OutlinedTextField(
                    value = textFieldImage,
                    onValueChange = { textFieldImage = it },
                    label = { Text( text = "Image URL") }
                )
            }
        },
```



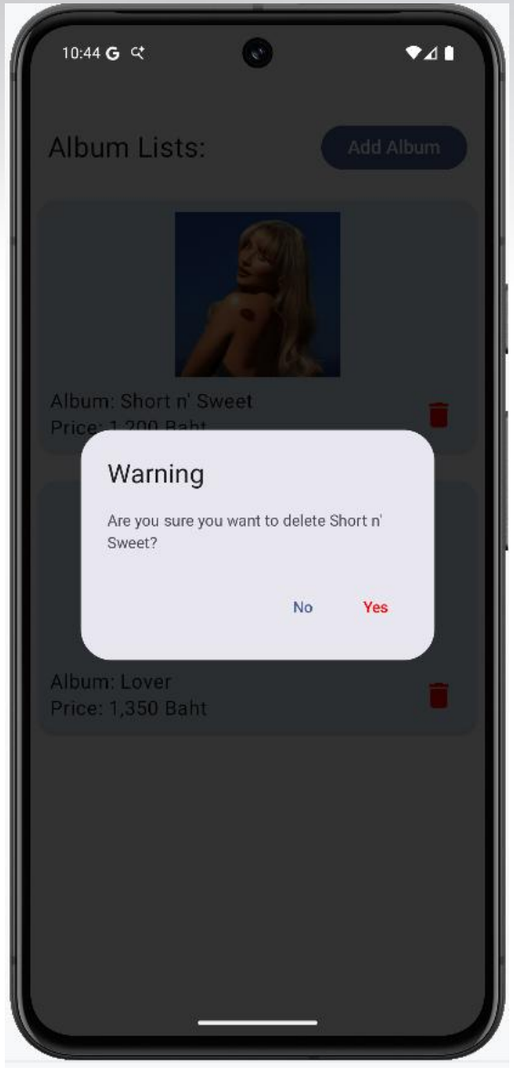
Lazy Column and Dialog : Example

```
            confirmButton = {
                TextButton(
                    onClick = {
                        val priceInt = textFieldPrice.toIntOrNull() ?: 0
                        albumItemsList.add(AlbumClass(textFieldName, priceInt, textFieldImage))
                        // Reset & Close
                        textFieldName = ""
                        textFieldPrice = ""
                        textFieldImage = ""
                        addDialog = false
                    }
                ) { Text( text = "Save") }
            },
            dismissButton = {
                TextButton(
                    onClick = {
                        addDialog = false
                        textFieldName = ""
                        textFieldPrice = ""
                        textFieldImage = ""
                    }
                ) { Text( text = "Cancel") }
            }
        )
    }
```



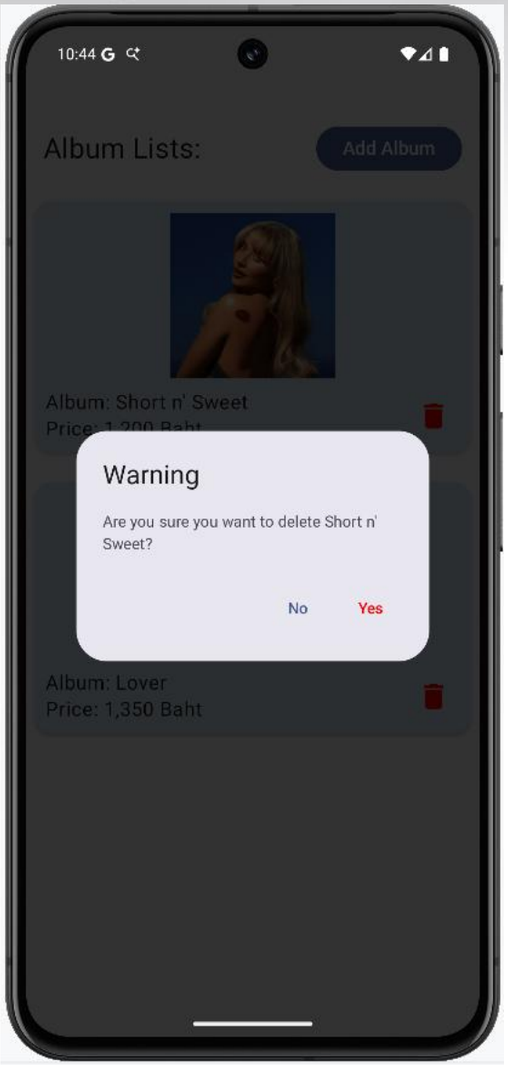
Lazy Column and Dialog : Example

```
// Delete Dialog
if (deleteDialog && itemToDelete != null) {
    AlertDialog(
        onDismissRequest = { deleteDialog = false },
        title = { Text(text = "Warning") },
        text = { Text(text = "Are you sure you want to delete ${itemToDelete!!.name}?") },
        confirmButton = {
            TextButton(
                onClick = {
                    albumItemsList.remove( element = itemToDelete)
                    Toast.makeText(contextForToast,  text = "${itemToDelete!!.name} deleted",
                        duration = Toast.LENGTH_SHORT).show()

                    deleteDialog = false
                    itemToDelete = null // Clear the value
                }
            ) { Text( text = "Yes", color = Color.Red) }
        },
```



Lazy Column and Dialog : Example

```
            dismissButton = {
                TextButton(
                    onClick = {
                        deleteDialog = false
                        itemToDelete = null
                    }
                ) { Text( text = "No") }
            }
        )
    }
}
```



References

- https://semicolonspace.com/jetpack-compose-lazy-grid-layouts/
- https://developer.android.com/jetpack/compose/lists
- https://medium.com/android-news/taking-a-look-at-coil-217e1f02ef5e
- https://semicolonspace.com/jetpack-compose-dialog-material3/
- https://semicolonspace.com/jetpack-compose-custom-dialog-designs/
- https://semicolonspace.com/jetpack-compose-full-screen-dialog/
- https://semicolonspace.com/jetpack-compose-lazyrow/
- https://semicolonspace.com/jetpack-compose-lazycolumn/