Uniwersytet Wrocławski

Rafał Lewandków

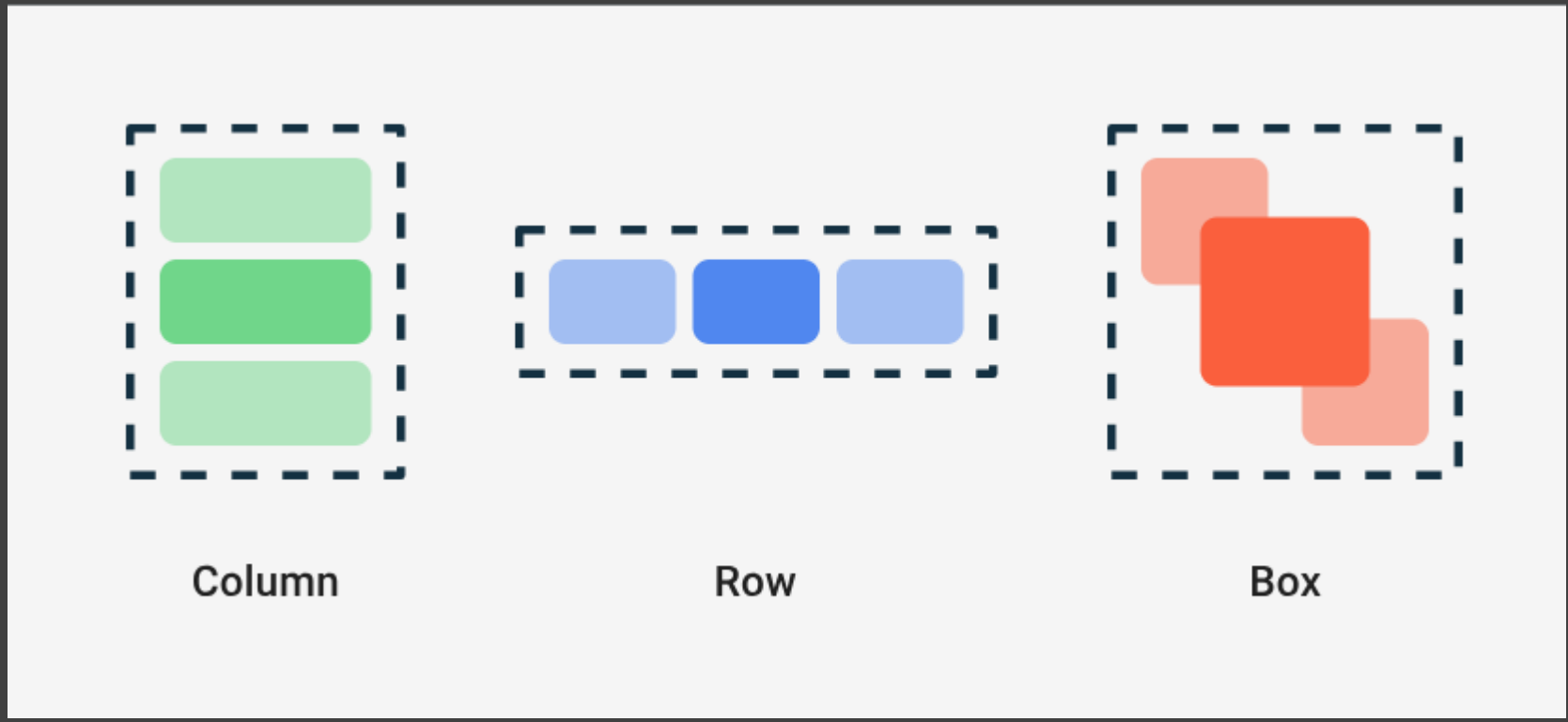
# PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

## WYKŁAD 6

### Jetpack Compose

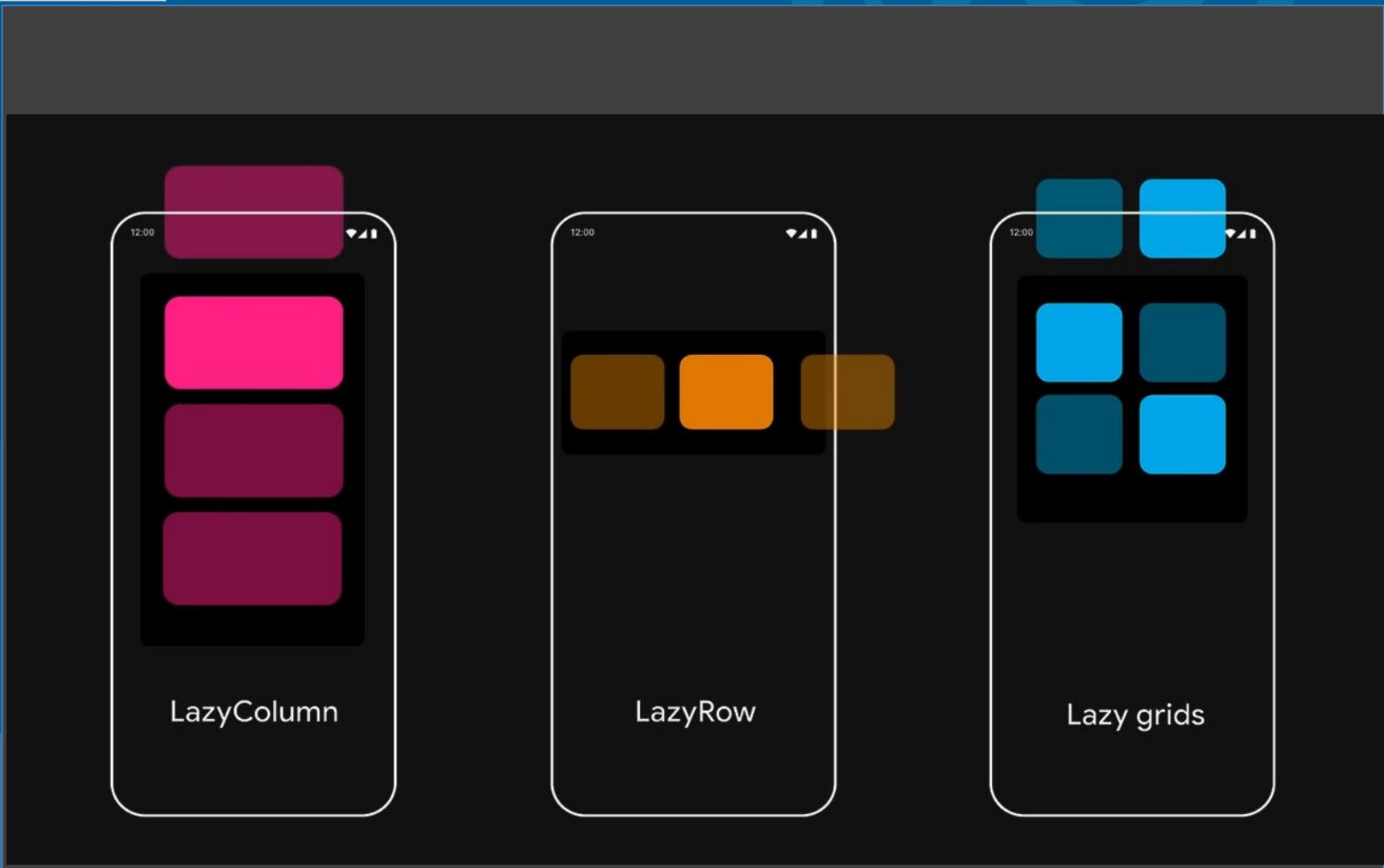
- LazyColumn, LazyRow
- Listy
- Grid
- Pager
- State

# Jetpack Compose



Uniwersytet Wrocławski

# Jetpack Compose

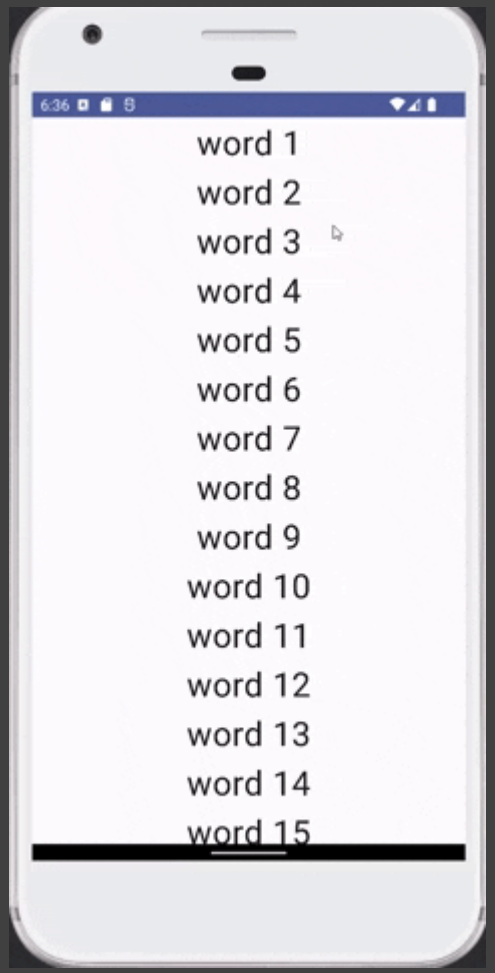


Uniwersytet Wrocławski

# LazyColumn

```
private fun generateWordList(): MutableList<String> {
    return (1..50).map { "word $it" }.toMutableList()
}
```

```
@Composable
fun ListOfWords(words: MutableList<String>){
    LazyColumn{
        items(words.size){
            var word by remember {
                mutableStateOf(words[it])
            }
            Text(
                text = word,
                fontSize = 32.sp,
                textAlign = TextAlign.Center,
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(2.dp)
                    .clickable {
                        word += "Clicked!!!"
                        words[it] = word
                    }
            )
        }
    }
}
```



Uniwersytet Wrocławski

# LazyColumn

Podczas pracy z listami w Jetpack Compose, możesz natknąć się zarówno na

- **mutableStateOf(listOf<T>())**
- **mutableStateListOf<T>()**.

Na pierwszy rzut oka wyglądają podobnie, ale mają różne zastosowania.

```
val mutableList = remember { mutableListOf("A", "B", "C") }
val state = remember { mutableStateOf(mutableList) }

// This won't trigger recomposition
state.value.add("D")
// The value hasn't changed
state.value == mutableList // true
```

Uniwersytet Wrocławski

# LazyColumn

Podczas pracy z listami w Jetpack Compose, możesz natknąć się zarówno na

- **mutableStateOf(listOf<T>())**
- **mutableStateListOf<T>()**.

Na pierwszy rzut oka wyglądają podobnie, ale mają różne zastosowania.

```
val state = remember { mutableStateListOf("A", "B", "C") }

// This will trigger recomposition
state.add("D")
```

# LazyColumn

```
LazyColumn {
    // Add a single item
    item {
        Text(text = "First item")
    }

    // Add 5 items
    items(5) { index ->
        Text(text = "Item: $index")
    }

    // Add another single item
    item {
        Text(text = "Last item")
    }
}
```

Uniwersytet Wrocławski

# LazyColumn

```
LazyColumn {
    // Add a single item
    item {
        Text(text = "First item")
    }

    // Add 5 items
    items(5) { index ->
        Text(text = "Item: $index")
    }

    // Add another single item
    item {
        Text(text = "Last item")
    }
}
```

```
/**
 * import androidx.compose.foundation.lazy.items
 */
LazyColumn {
    items(messages) { message ->
        MessageRow(message)
    }
}
```

Uniwersytet Wrocławski

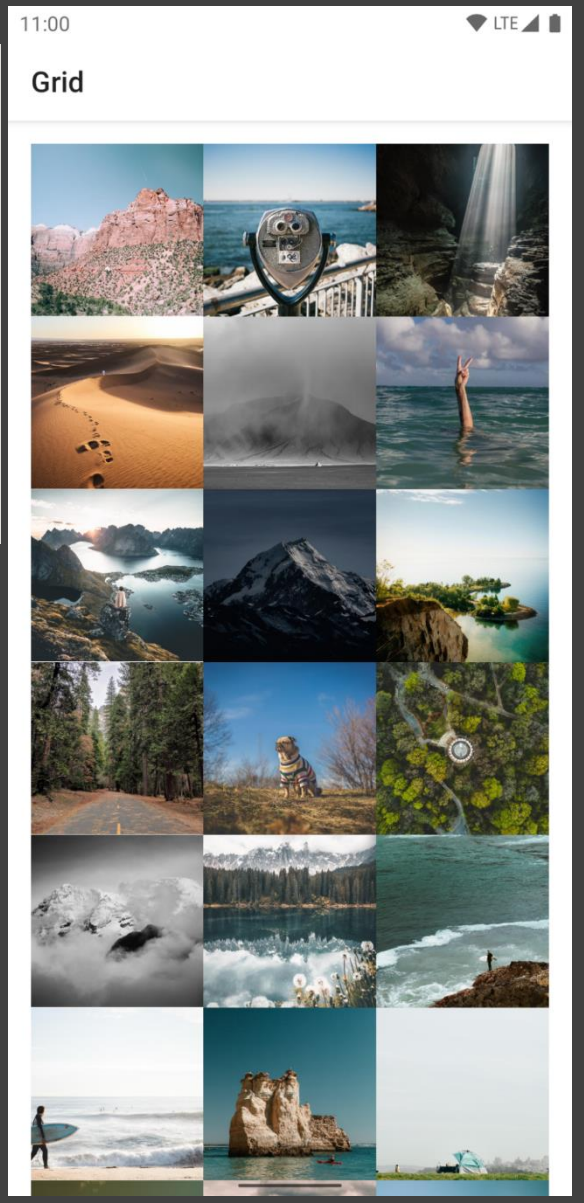
# LazyVerticalGrid

```
import androidx.compose.foundation.border
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.wrapContentSize
import androidx.compose.foundation.lazy.grid.LazyVerticalGrid
import androidx.compose.foundation.lazy.grid.items
import androidx.compose.foundation.lazy.grid.itemsIndexed
import androidx.compose.material.Text

val itemsList = (0..5).toList()
val itemsIndexedList = listOf("A", "B", "C")

val itemModifier = Modifier.border(1.dp, Color.Blue).height(80.dp).wrapContentSize()

LazyVerticalGrid(
    columns = GridCells.Fixed(3)
) {
    items(itemsList) {
        Text("Item is $it", itemModifier)
    }
    item {
        Text("Single item", itemModifier)
    }
    itemsIndexed(itemsIndexedList) { index, item ->
        Text("Item at index $index is $item", itemModifier)
    }
}
```

Uniwersytet Wrocławski

# Grid

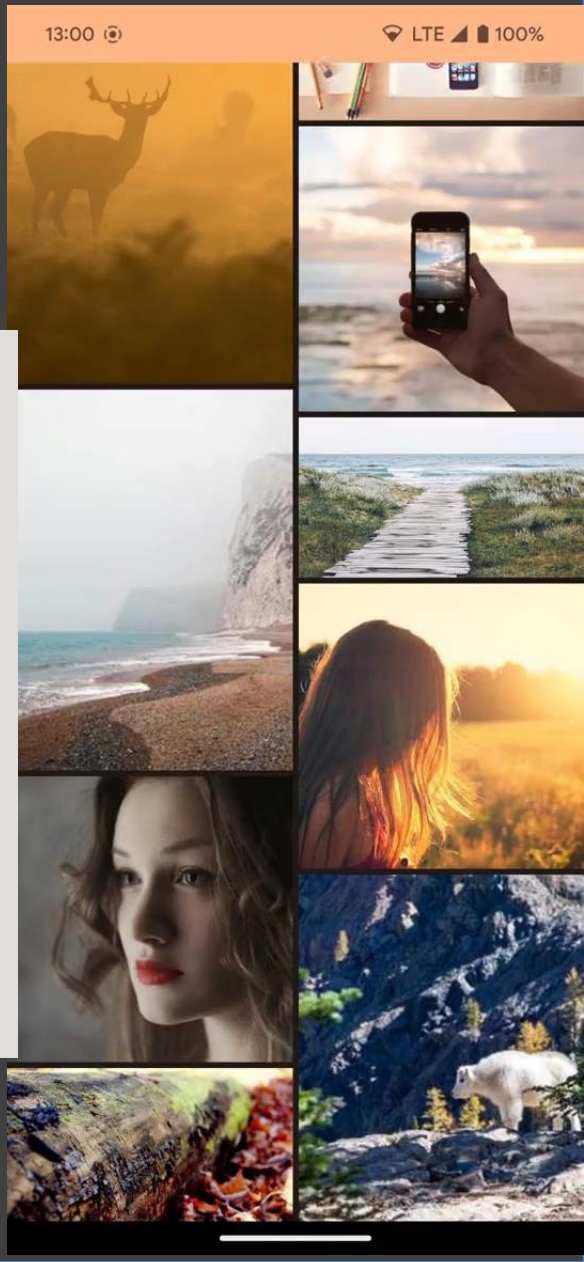
```
LazyVerticalGrid(
    columns = GridCells.Adaptive(minSize = 128.dp)
) {
    items(photos) { photo ->
        PhotoItem(photo)
    }
}
```



Uniwersytet Wrocławski

# LazyVerticalStaggeredGrid

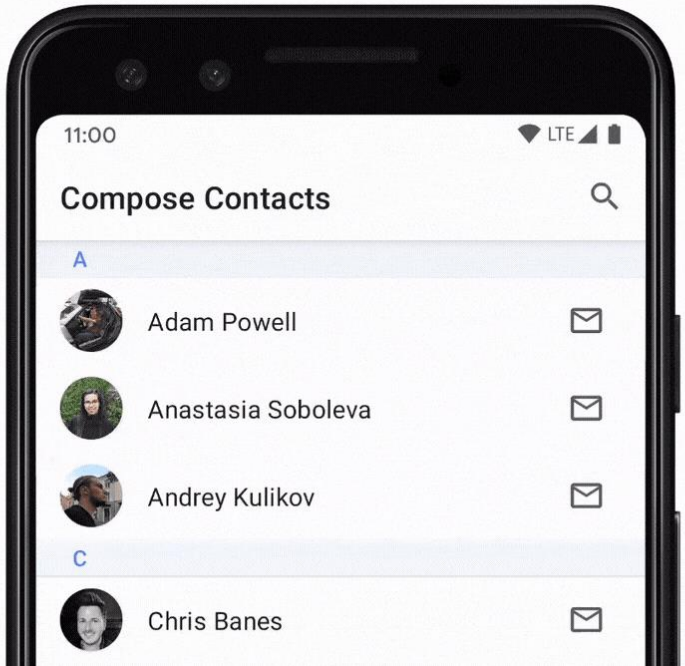
```
LazyVerticalStaggeredGrid(
    columns = StaggeredGridCells.Adaptive(200.dp),
    verticalItemSpacing = 4.dp,
    horizontalArrangement = Arrangement.spacedBy(4.dp),
    content = {
        items(randomSizedPhotos) { photo ->
            AsyncImage(
                model = photo,
                contentScale = ContentScale.Crop,
                contentDescription = null,
                modifier = Modifier.fillMaxWidth().wrapContentHeight()
            )
        }
    },
    modifier = Modifier.fillMaxSize()
)
```



Uniwersytet Wrocławski

# StickyHeader

```
@OptIn(ExperimentalFoundationApi::class)
@Composable
fun ListWithHeader(items: List<Item>) {
    LazyColumn {
        stickyHeader {
            Header()
        }

        items(items) { item ->
            ItemRow(item)
        }
    }
}
```



Uniwersytet Wrocławski

# ListItem

Headline
Supporting text
100+

Headline
Supporting text

A

Headline
Supporting text

# ListItem

```
@Composable
fun ListItem(
    headlineContent: @Composable () -> Unit,
    modifier: Modifier = Modifier,
    overlineContent: @Composable (() -> Unit)? = null,
    supportingContent: @Composable (() -> Unit)? = null,
    leadingContent: @Composable (() -> Unit)? = null,
    trailingContent: @Composable (() -> Unit)? = null,
    colors: ListItemColors = ListItemDefaults.colors(),
    tonalElevation: Dp = ListItemDefaults.Elevation,
    shadowElevation: Dp = ListItemDefaults.Elevation,
)
```

# ListItem

## Parameters

| name | description |
| --- | --- |
| `headlineContent` | the headline content of the list item |
| `modifier` | [Modifier] to be applied to the list item |
| `overlineContent` | the content displayed above the headline content |
| `supportingContent` | the supporting content of the list item |
| `leadingContent` | the leading content of the list item |
| `trailingContent` | the trailing meta text, icon, switch or checkbox |
| `colors` | [ListItemColors] that will be used to resolve the background and content color for this list item in different states. See [ListItemDefaults.colors] |
| `tonalElevation` | the tonal elevation of this list item |
| `shadowElevation` | the shadow elevation of this list item |

# ListItem

## OneLineListItem

```
@Preview
@Composable
fun OneLineListItem() {
    Column {
        ListItem(
            headlineContent = { Text("One line list item with 24x24 icon") },
            leadingContent = {
                Icon(
                    Icons.Filled.Favorite,
                    contentDescription = "Localized description",
                )
            }
        )
        HorizontalDivider()
    }
}
```

# ListItem

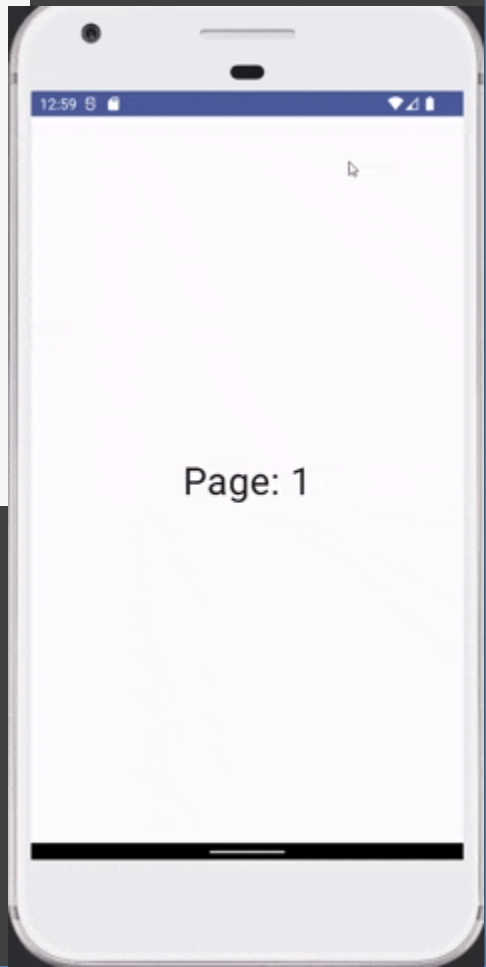
## TwoLineListItem

```
@Preview
@Composable
fun TwoLineListItem() {
    Column {
        ListItem(
            headlineContent = { Text("Two line list item with trailing") },
            supportingContent = { Text("Secondary text") },
            trailingContent = { Text("meta") },
            leadingContent = {
                Icon(
                    Icons.Filled.Favorite,
                    contentDescription = "Localized description",
                )
            }
        )
        HorizontalDivider()
    }
}
```

Uniwersytet Wrocławski

# Pager

```
@OptIn(ExperimentalFoundationApi::class)
@Composable
fun PagerBasics() {
    val pagerState = rememberPagerState()
    HorizontalPager(
        pageCount = 3,
        state = pagerState,
        pageSize = PageSize.Fill
    ) { page ->
        Text(
            text = "Page: ${page + 1}",
            modifier = Modifier
                .fillMaxSize()
                .wrapContentSize(Alignment.Center), // centrowanie wertykalne
            textAlign = TextAlign.Center, // centrowanie horyzontalne
            fontSize = 36.sp
        )
    }
}
```



Uniwersytet Wrocławski

# Choreographer