



PROGRAMOWANIE URZĄDZEŃ MOBILNYCH 1

WYKŁAD 10

- Jetpack Compose
- Material Design
- Elementy Struktury Interfejsu Użytkownika
- Scaffold
- Obsługa kolekcji

Scaffold w **Material Design** to **struktura organizująca** elementy interfejsu, zapewniająca spójność i **standardowy wygląd aplikacji**.

Slot na górny pasek aplikacji (np. **TopAppBar**)

```
@Composable
fun ScaffoldExample() {
    Scaffold(
        topBar = { },
        floatingActionButton = { },
        bottomBar = { },
    ) { innerPadding ->
        Box (Modifier
            .padding(innerPadding)){ }
    }
}
```



Scaffold w **Material Design** to **struktura organizująca** elementy interfejsu, zapewniająca spójność i **standardowy wygląd aplikacji**.

Slot na przycisk akcji
(FAB),

Slot na górny pasek
aplikacji (np. **AppBar**)

```
@Composable
fun ScaffoldExample() {
    Scaffold(
        topBar = { },
        floatingActionButton = { },
        bottomBar = { },
    ) { innerPadding ->
        Box (Modifier
            .padding(innerPadding)) { }
    }
}
```



Scaffold w **Material Design** to **struktura organizująca** elementy interfejsu, zapewniająca spójność i **standardowy wygląd aplikacji**.

```
@Composable
fun ScaffoldExample() {
    Scaffold(
        topBar = { },
        floatingActionButton = { },
        bottomBar = { },
    ) { innerPadding ->
        Box (Modifier
            .padding(innerPadding)) { }
    }
}
```

Slot na przycisk akcji
(FAB),

Slot na dolny pasek
aplikacji (np.
BottomAppBar),

Slot na górny pasek
aplikacji (np. **TopAppBar**)



Scaffold w **Material Design** to **struktura organizująca** elementy interfejsu, zapewniająca spójność i **standardowy wygląd aplikacji**.

```
@Composable
fun ScaffoldExample() {
    Scaffold(
        topBar = { },
        floatingActionButton = { },
        bottomBar = { },
    ) { innerPadding ->
        Box (Modifier
            .padding(innerPadding)) { }
    }
}
```

Slot na przycisk akcji
(**FAB**),

Slot na dolny pasek
aplikacji (np.
BottomAppBar),

Główna zawartość

Slot na górny pasek
aplikacji (np. **TopAppBar**)



Scaffold w **Material Design** to **struktura organizująca** elementy interfejsu, zapewniająca spójność i **standardowy wygląd aplikacji**.

```
@Composable
fun ScaffoldExample() {
    Scaffold(
        topBar = { },
        floatingActionButton = { },
        bottomBar = { },
    ) { innerPadding ->
        Box (Modifier
            .padding(innerPadding)) { }
    }
}
```

Slot na górny pasek aplikacji (np. **TopAppBar**)

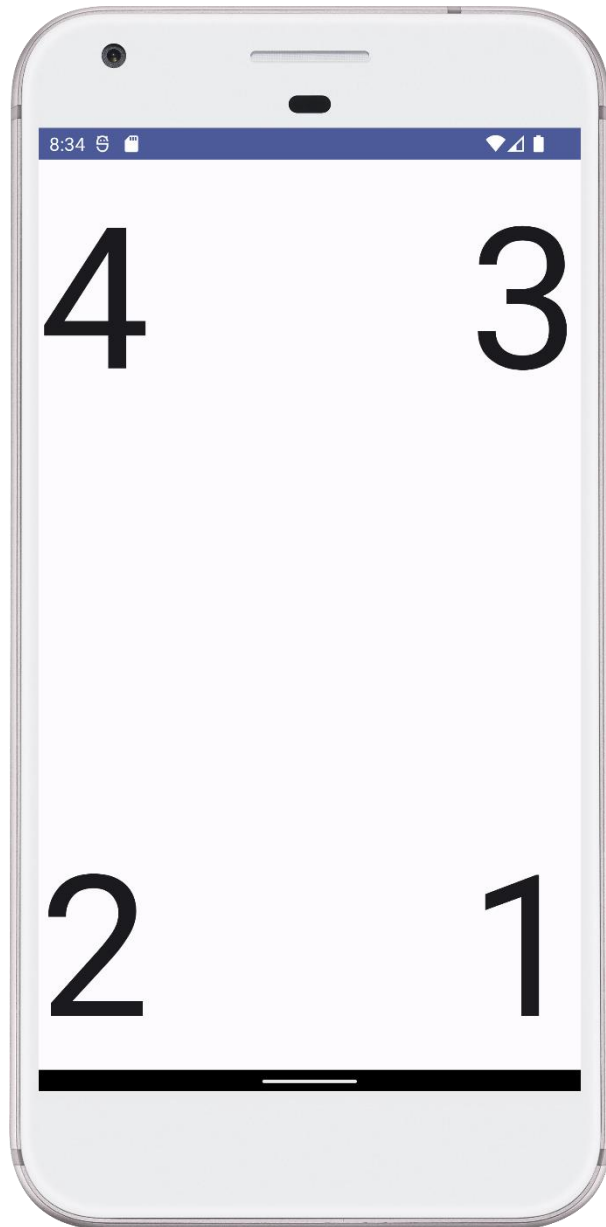
Slot na przycisk akcji (FAB),

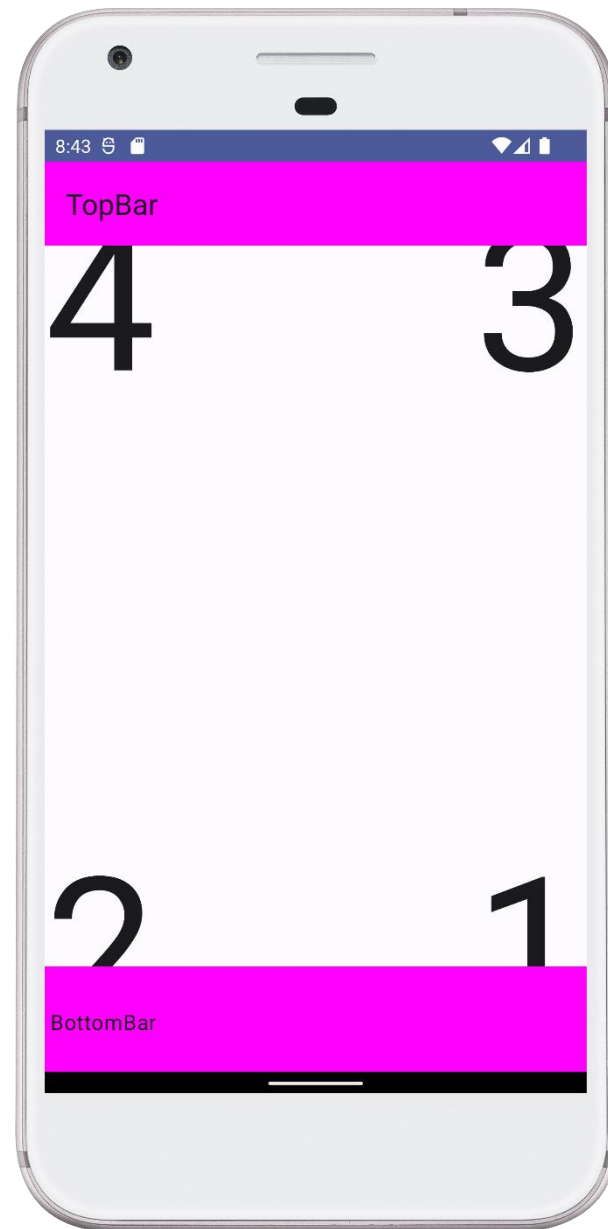
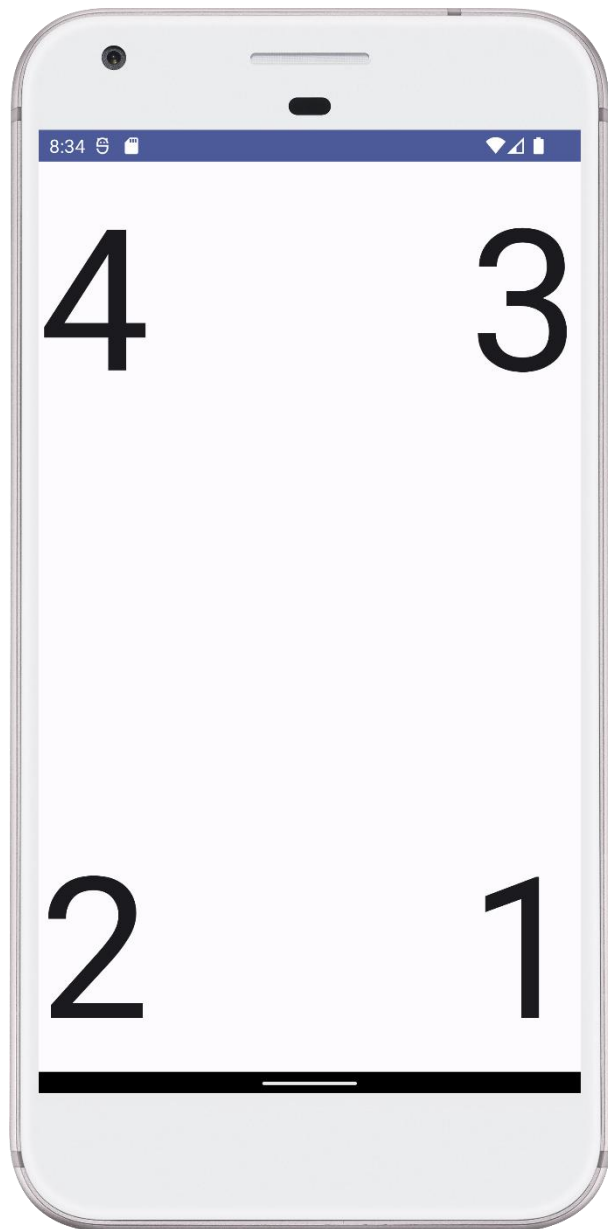
Slot na dolny pasek aplikacji (np. **BottomAppBar**),

Główna zawartość

zapewnia **automatyczne** dostosowanie **paddingu** wewnątrz Scaffold

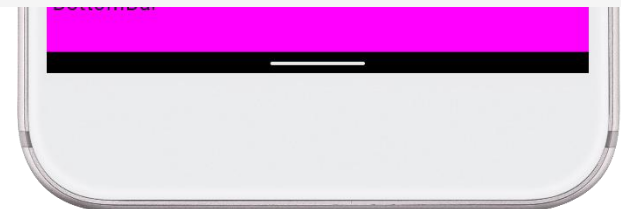
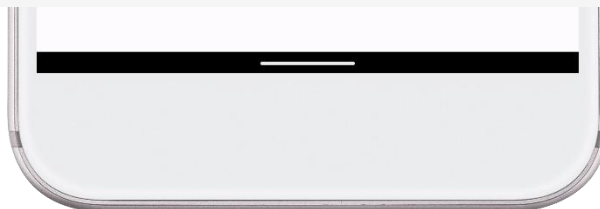


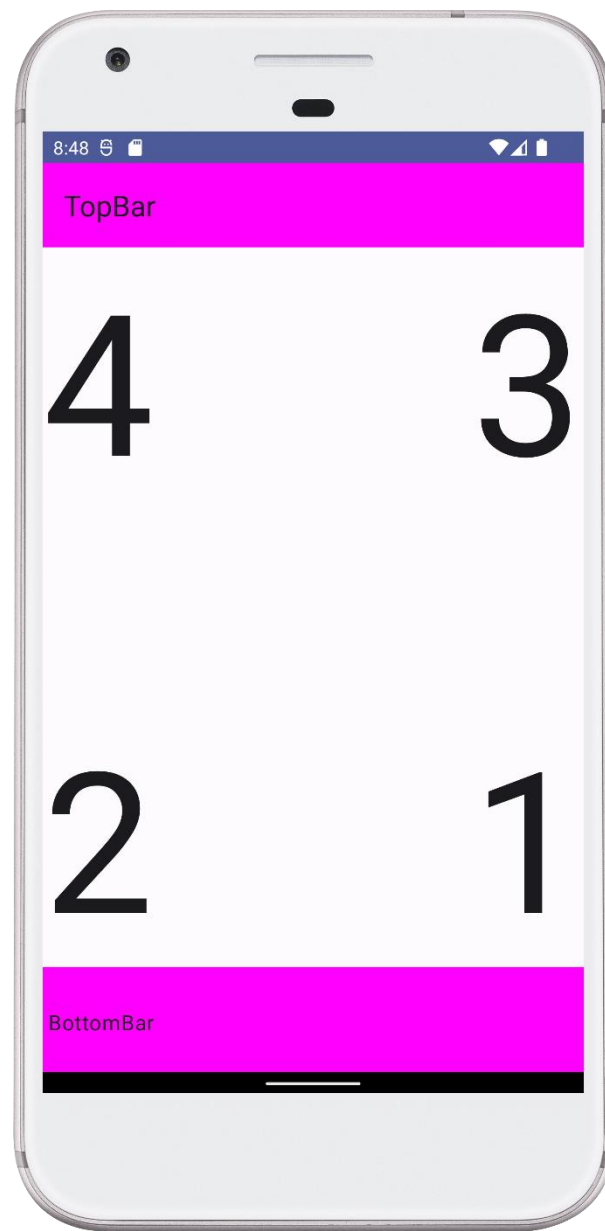
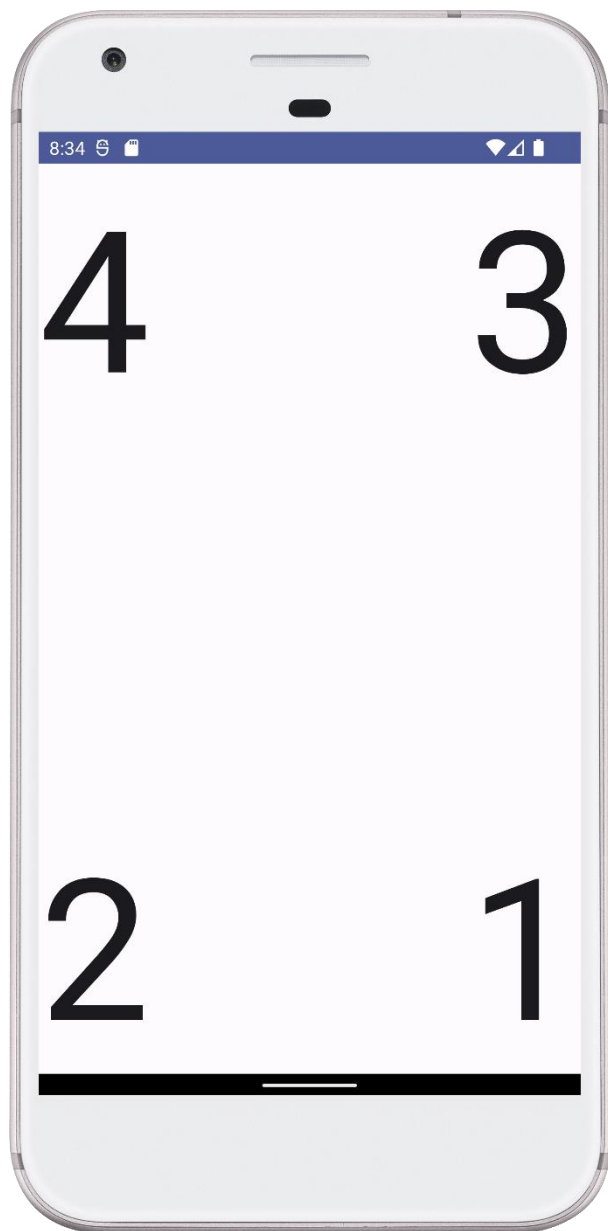






```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun PaddingValuesIssue(){
    Scaffold (
        topBar = { TopAppBar(title = { Text("TopBar") }, colors = TopAppBarDefaults.smallTopAppBarColors(con
        ) },
        content = { paddingValues ->
            Box(modifier = Modifier
                .fillMaxSize()
                .padding(paddingValues) // wykorzystujemy automatycznie wygenerowaną wartość marginesów
            ) {
                Text(text = "1", fontSize = 150.sp, modifier = Modifier.align(Alignment.BottomEnd))
                Text(text = "2", fontSize = 150.sp, modifier = Modifier.align(Alignment.BottomStart))
                Text(text = "3", fontSize = 150.sp, modifier = Modifier.align(Alignment.TopEnd))
                Text(text = "4", fontSize = 150.sp, modifier = Modifier.align(Alignment.TopStart))
            }
        },
        bottomBar = { BottomAppBar(containerColor = Color.Magenta) { Text(text = "BottomBar") }}
    )
}
```






AppBar to komponent z biblioteki **Material Design** (zarówno w wersji 2, jak i 3), który reprezentuje **górny pasek aplikacji**.

Adnotacja ta jest wymagana, ponieważ **AppBar** z biblioteki Material 3 jest jeszcze w **fazie eksperymentalnej**.

```
> 1 @OptIn(ExperimentalMaterial3Api::class)
   2 @Composable
   3 fun MyAppBar() {
   4     AppBar(
   5         title = { Text(text = "My App") },
   6         colors = AppBarDefaults.appBarColors(
   7             containerColor = Color.Cyan
   8         ),
   9         navigationIcon = {
  10             IconButton(onClick = { }) {
  11                 Icon(
  12                     imageVector = Icons.Default.Menu
  13                 )
  14             }
  15         },
  16         actions = {
  17             IconButton(onClick = { }) {
  18                 Icon(
  19                     imageVector = Icons.Default.Favorite
  20                 )
  21             }
  22         }
  23     )
  24 }
```




TopAppBar to komponent z biblioteki **Material Design** (zarówno w wersji 2, jak i 3), który reprezentuje **górny pasek aplikacji**.

Adnotacja ta jest wymagana, ponieważ **TopAppBar** z biblioteki Material 3 jest jeszcze w **fazie eksperymentalnej**.

Slot na tytuł paska aplikacji.

```
> 1 @OptIn(ExperimentalMaterial3Api::class)
   2 @Composable
   3 fun MyTopAppBar() {
   4     TopAppBar(
   5         title = { Text(text = "My App") },
   6         colors = TopAppBarDefaults.topAppBarColors(
   7             containerColor = Color.Cyan
   8         ),
   9         navigationIcon = {
  10             IconButton(onClick = { }) {
  11                 Icon(
  12                     imageVector = Icons.Default.Menu
  13                 )
  14             }
  15         },
  16         actions = {
  17             IconButton(onClick = { }) {
  18                 Icon(
  19                     imageVector = Icons.Default.Favorite
  20                 )
  21             }
  22         }
  23     )
  24 }
```




TopAppBar to komponent z biblioteki **Material Design** (zarówno w wersji 2, jak i 3), który reprezentuje **górny pasek aplikacji**.

Adnotacja ta jest wymagana, ponieważ **TopAppBar** z biblioteki Material 3 jest jeszcze w **fazie eksperymentalnej**.

Slot na tytuł paska aplikacji.

Parametr pozwalający dostosować **kolory paska aplikacji**.

```
> 1 @OptIn(ExperimentalMaterial3Api::class)
2   @Composable
3   fun MyTopAppBar() {
4       TopAppBar(
5           title = { Text(text = "My App") },
6           colors = TopAppBarDefaults.topAppBarColors(
7               containerColor = Color.Cyan
8           ),
9           navigationIcon = {
10              IconButton(onClick = { }) {
11                  Icon(
12                      imageVector = Icons.Default.Menu
13                  )
14              }
15          },
16          actions = {
17              IconButton(onClick = { }) {
18                  Icon(
19                      imageVector = Icons.Default.Favorite
20                  )
21              }
22          }
23      )
24  }
```



TopAppBar to komponent z biblioteki **Material Design** (zarówno w wersji 2, jak i 3), który reprezentuje **górny pasek aplikacji**.


Adnotacja ta jest wymagana, ponieważ **TopAppBar** z biblioteki Material 3 jest jeszcze w **fazie eksperymentalnej**.

Slot na tytuł paska aplikacji.

Parametr pozwalający dostosować **kolory paska aplikacji**.

Slot na **ikonę nawigacyjną**, która zwykle znajduje się po **lewej stronie** paska.

```
1  @OptIn(ExperimentalMaterial3Api::class)
2  @Composable
3  fun MyTopAppBar() {
4      TopAppBar(
5          title = { Text(text = "My App") },
6          colors = TopAppBarDefaults.topAppBarColors(
7              containerColor = Color.Cyan
8          ),
9          navigationIcon = {
10             IconButton(onClick = { }) {
11                 Icon(
12                     imageVector = Icons.Default.Menu
13                 )
14             }
15         },
16         actions = {
17             IconButton(onClick = { }) {
18                 Icon(
19                     imageVector = Icons.Default.Favorite
20                 )
21             }
22         }
23     )
24 }
```



TopAppBar to komponent z biblioteki **Material Design** (zarówno w wersji 2, jak i 3), który reprezentuje **górny pasek aplikacji**.

Adnotacja ta jest wymagana, ponieważ **TopAppBar** z biblioteki Material 3 jest jeszcze w **fazie eksperymentalnej**.


Slot na tytuł paska aplikacji.

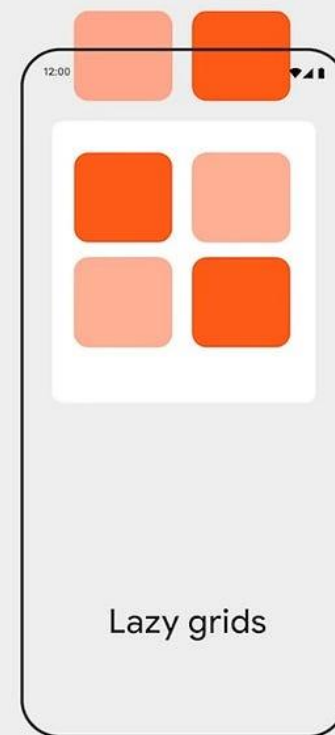
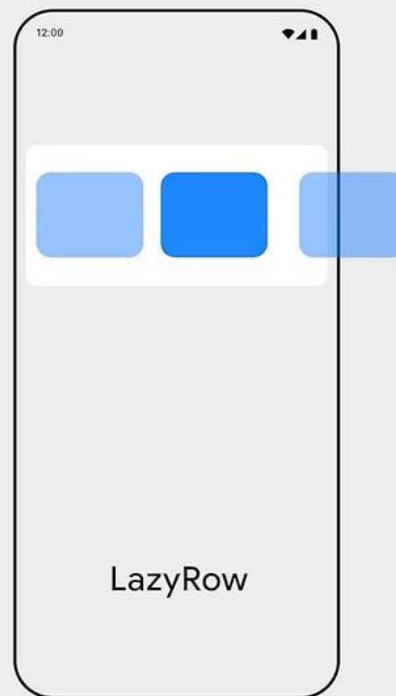
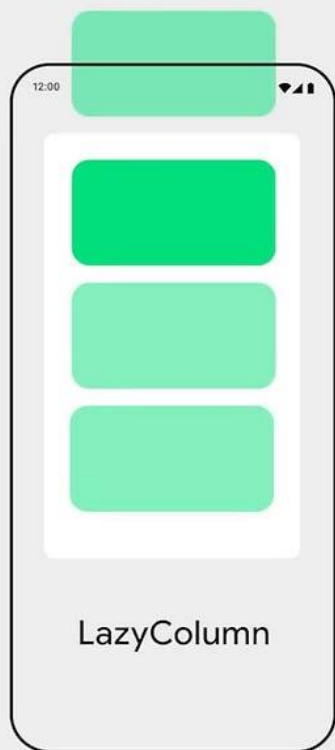
Parametr pozwalający dostosować **kolory paska aplikacji**.

Slot na **ikonę nawigacyjną**, która zwykle znajduje się po **lewej stronie** paska.

Slot na **przyciski akcji**, które zwykle znajdują się po **prawej stronie** paska.

```
1  @OptIn(ExperimentalMaterial3Api::class)
2  @Composable
3  fun MyTopAppBar() {
4      TopAppBar(
5          title = { Text(text = "My App") },
6          colors = TopAppBarDefaults.topAppBarColors(
7              containerColor = Color.Cyan
8          ),
9          navigationIcon = {
10             IconButton(onClick = { }) {
11                 Icon(
12                     imageVector = Icons.Default.Menu
13                 )
14             }
15         },
16         actions = {
17             IconButton(onClick = { }) {
18                 Icon(
19                     imageVector = Icons.Default.Favorite
20                 )
21             }
22         }
23     )
24 }
```





LazyColumn to komponent służący do wyświetlania przewijalnych list. Działa na zasadzie **leniwego ładowania**, renderując **tylko** te elementy, które są **aktualnie widoczne** na ekranie.

Optymalizuje renderowanie, tworząc **tylko** te **elementy**, które są **widoczne** na ekranie.

```
@Composable
fun WordList() {
    val words = listOf("Kotlin", "Compose", "Android",
        "Studio", "Jetpack", "LazyColumn")
    LazyColumn(modifier = Modifier.fillMaxSize(),
        contentPadding = PaddingValues(16.dp)
    ) {
        items(words) { word ->
            Text(text = word, fontSize = 32.sp,
                modifier = Modifier.fillMaxWidth()
                    .padding(vertical = 8.dp))
            Spacer(modifier = Modifier.height(8.dp))
        }
    }
}
```

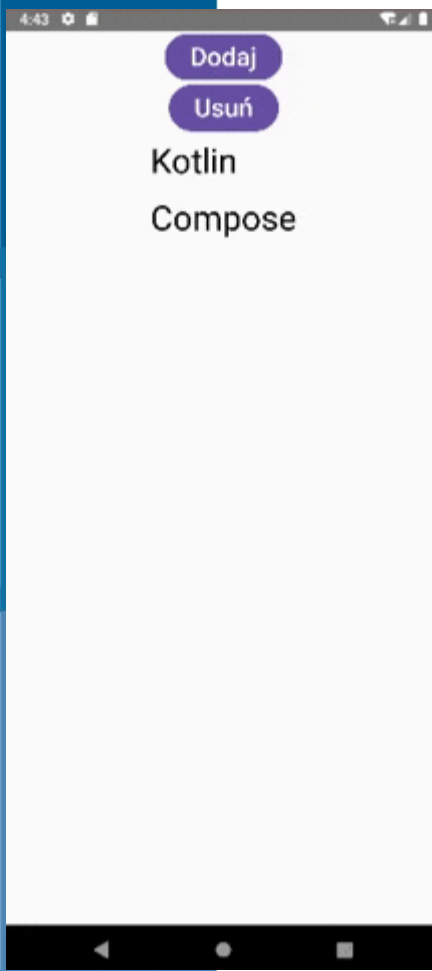
LazyColumn to komponent służący do wyświetlania przewijalnych list. Działa na zasadzie **leniwego ładowania**, renderując **tylko** te elementy, które są **aktualnie widoczne** na ekranie.

Optymalizuje renderowanie, tworząc **tylko te elementy**, które są **widoczne** na ekranie.

Renderuje element przekazanej listy

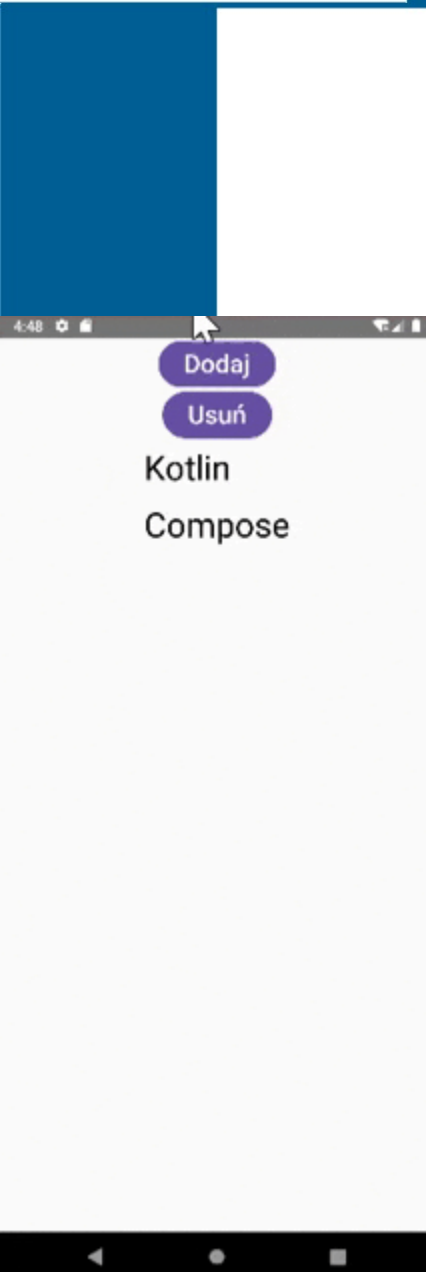
```
@Composable
fun WordList() {
    val words = listOf("Kotlin", "Compose", "Android",
        "Studio", "Jetpack", "LazyColumn")
    LazyColumn(modifier = Modifier.fillMaxSize(),
        contentPadding = PaddingValues(16.dp)
    ) {
        items(words) { word ->
            Text(text = word, fontSize = 32.sp,
                modifier = Modifier.fillMaxWidth()
                    .padding(vertical = 8.dp))
            Spacer(modifier = Modifier.height(8.dp))
        }
    }
}
```

```
val wordsState1 = remember { mutableStateListOf("Kotlin", "Compose") }  
val wordsState2 = mutableListOf("Kotlin", "Compose")  
val wordsState3 = remember {mutableListOf("Kotlin", "Compose")}  
val wordsState4 by remember { mutableStateOf(mutableListOf("Kotlin", "Compose")) }
```



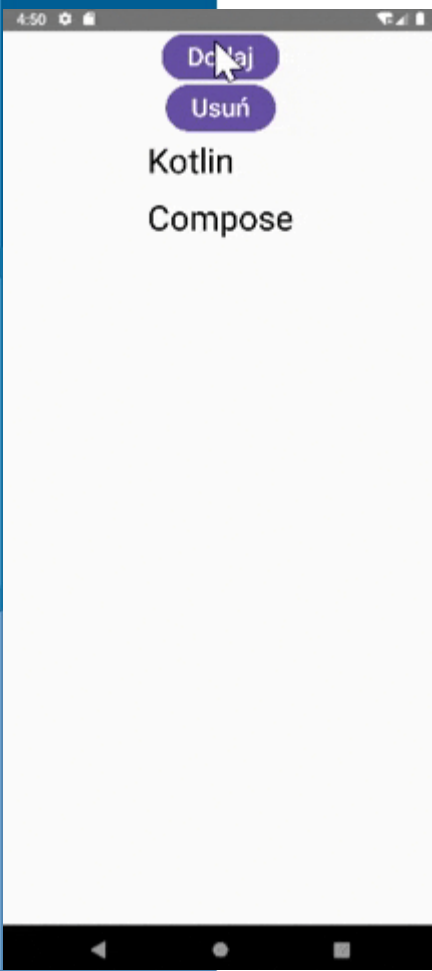
```
@Composable
fun WordList() {
    val wordsState = remember { mutableStateListOf("Kotlin", "Compose") }

    Column(
        Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Button(onClick = { wordsState.add("Jetpack") }) {
            Text(text: "Dodaj", fontSize = 24.sp)
        }
        Button(onClick = { wordsState.remove(element: "Jetpack") }) {
            Text(text: "Usuń", fontSize = 24.sp)
        }
        LazyColumn {
            items(wordsState) { word ->
                Text(word, fontSize = 32.sp, modifier = Modifier.padding(8.dp))
            }
        }
    }
}
```



```
@Composable
fun WordList() {
    val wordsState = mutableListOf("Kotlin", "Compose")

    Column(
        Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Button(onClick = { wordsState.add("Jetpack") }) {
            Text(text: "Dodaj", fontSize = 24.sp)
        }
        Button(onClick = { wordsState.remove(element: "Jetpack") }) {
            Text(text: "Usuń", fontSize = 24.sp)
        }
        LazyColumn {
            items(wordsState) { word ->
                Text(word, fontSize = 32.sp, modifier = Modifier.padding(8.dp))
            }
        }
    }
}
```

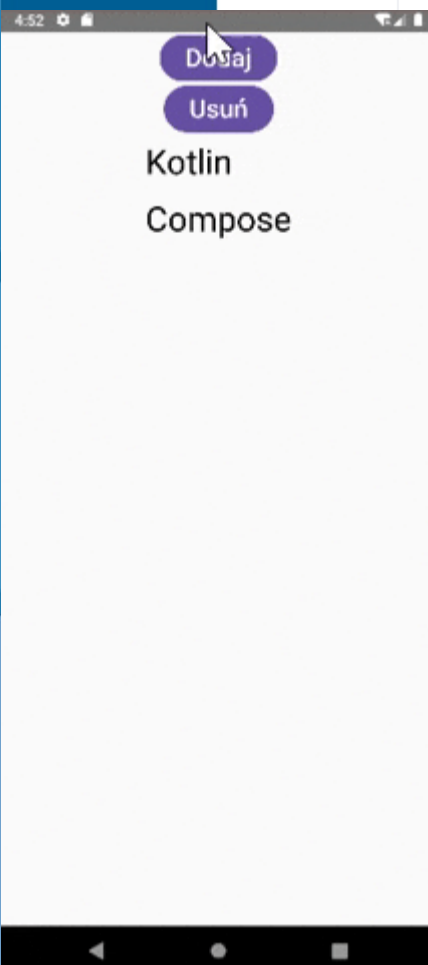


```
@Composable
fun WordList() {
    val wordsState = remember {mutableListOf("Kotlin", "Compose")}

    Column(
        Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Button(onClick = { wordsState.add("Jetpack") }) {
            Text(text: "Dodaj", fontSize = 24.sp)
        }
        Button(onClick = { wordsState.remove(element: "Jetpack") }) {
            Text(text: "Usuń", fontSize = 24.sp)
        }
        LazyColumn {
            items(wordsState) { word ->
                Text(word, fontSize = 32.sp, modifier = Modifier.padding(8.dp))
            }
        }
    }
}
```


@Composable

```
fun WordList() {  
    val wordsState by remember { mutableStateOf(mutableListOf("Kotlin", "Compose")) }  
  
    Column(  
        Modifier.fillMaxSize(),  
        horizontalAlignment = Alignment.CenterHorizontally  
    ) {  
        Button(onClick = { wordsState.add("Jetpack") }) {  
            Text(text: "Dodaj", fontSize = 24.sp)  
        }  
        Button(onClick = { wordsState.remove(element: "Jetpack") }) {  
            Text(text: "Usuń", fontSize = 24.sp)  
        }  
        LazyColumn {  
            items(wordsState) { word ->  
                Text(word, fontSize = 32.sp, modifier = Modifier.padding(8.dp))  
            }  
        }  
    }  
}
```



Typ	mutableStateListOf	mutableListOf	mutableStateOf <MutableList>
Obserwowalność	Tak	Nie	Tak
Modyfikowalność	Tak	Tak	Tak
Aktualizacja UI	Automatyczna	Brak	Tak – tylko przy przypisaniu nowej listy

Typ	mutableStateListOf	mutableListOf	mutableStateOf <MutableList>
Obserwowalność	Tak	Nie	Tak
Modyfikowalność	Tak	Tak	Tak
Aktualizacja UI	Automatyczna	Brak	Tak – tylko przy przypisaniu nowej listy

```
val wordsState1 = remember { mutableStateListOf("Kotlin", "Compose") }
```

- **mutableStateListOf z remember:** Lista jest zarówno **modyfikowalna**, jak i **obserwowalna**. Każda zmiana w liście **automatycznie** powoduje ponowne renderowanie UI.

Typ	mutableStateListOf	mutableListOf	mutableStateOf <MutableList>
Obserwowalność	Tak	Nie	Tak
Modyfikowalność	Tak	Tak	Tak
Aktualizacja UI	Automatyczna	Brak	Tak – tylko przy przypisaniu nowej listy

```
val wordsState1 = remember { mutableStateListOf("Kotlin", "Compose") }
```

- **mutableStateListOf z remember:** Lista jest zarówno **modyfikowalna**, jak i **obserwowalna**. Każda zmiana w liście **automatycznie** powoduje ponowne renderowanie UI.

```
val wordsState2 = mutableListOf("Kotlin", "Compose")
```

- **mutableListOf:** Standardowa **modyfikowalna** lista, która **nie jest obserwowalna**. Zmiany w liście **nie powodują aktualizacji UI**.

Typ	mutableStateListOf	mutableListOf	mutableStateOf <MutableList>
Obserwowalność	Tak	Nie	Tak
Modyfikowalność	Tak	Tak	Tak
Aktualizacja UI	Automatyczna	Brak	Tak – tylko przy przypisaniu nowej listy

```
val wordsState1 = remember { mutableStateListOf("Kotlin", "Compose") }
```

- **mutableStateListOf z remember:** Lista jest zarówno **modyfikowalna**, jak i **obserwowalna**. Każda zmiana w liście **automatycznie** powoduje ponowne renderowanie UI.

```
val wordsState2 = mutableListOf("Kotlin", "Compose")
```

- **mutableListOf:** Standardowa **modyfikowalna** lista, która **nie jest obserwowalna**. Zmiany w liście **nie powodują aktualizacji UI**.

```
val wordsState3 = remember { mutableListOf("Kotlin", "Compose") }
```

- **mutableListOf z remember:** Lista jest **modyfikowalna**, ale **nie jest obserwowalna**. Stan listy jest zachowywany między kompozycjami.

Typ	mutableStateListOf	mutableListOf	mutableStateOf <MutableList>
Obserwowalność	Tak	Nie	Tak
Modyfikowalność	Tak	Tak	Tak
Aktualizacja UI	Automatyczna	Brak	Tak – tylko przy przypisaniu nowej listy

```
val wordsState1 = remember { mutableStateListOf("Kotlin", "Compose") }
```

- **mutableStateListOf z remember:** Lista jest zarówno **modyfikowalna**, jak i **obserwowalna**. Każda zmiana w liście **automatycznie** powoduje ponowne renderowanie UI.

```
val wordsState2 = mutableListOf("Kotlin", "Compose")
```

- **mutableListOf:** Standardowa **modyfikowalna** lista, która **nie jest obserwowalna**. Zmiany w liście **nie powodują aktualizacji UI**.

```
val wordsState3 = remember { mutableListOf("Kotlin", "Compose") }
```

- **mutableListOf z remember:** Lista jest **modyfikowalna**, ale **nie jest obserwowalna**. Stan listy jest zachowywany między kompozycjami.

```
val wordsState4 by remember { mutableStateOf(mutableListOf("Kotlin", "Compose")) }
```

- **mutableStateOf z mutableListOf i remember:** Lista jest **przechowywana** w MutableState, ale zmiany w liście **nie są automatycznie obserwowane**. Aby zaktualizować UI, trzeba **przypisać nową listę**.



```
@Composable
fun NewWordList() {
    val words = listOf("Kotlin", "Compose", "Android", "Studio", "Jetpack", "Material", "Design")
    LazyColumn(
        modifier = Modifier.fillMaxSize(),
        contentPadding = PaddingValues(16.dp)
    ) {
        items(words) {...}
    }
}
```


5:23 [status icons]

★ Kotlin
Framework: Kotlin



★ Compose
Framework: Compose



★ Android
Framework: Android



★ Studio
Framework: Studio



★ Jetpack
Framework: Jetpack



★ Material
Framework: Material



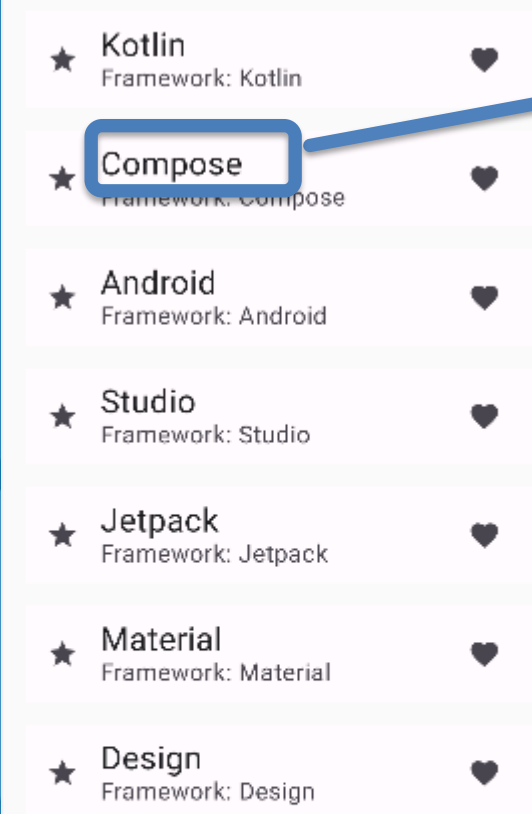
★ Design
Framework: Design



```
items(words) { word ->
```

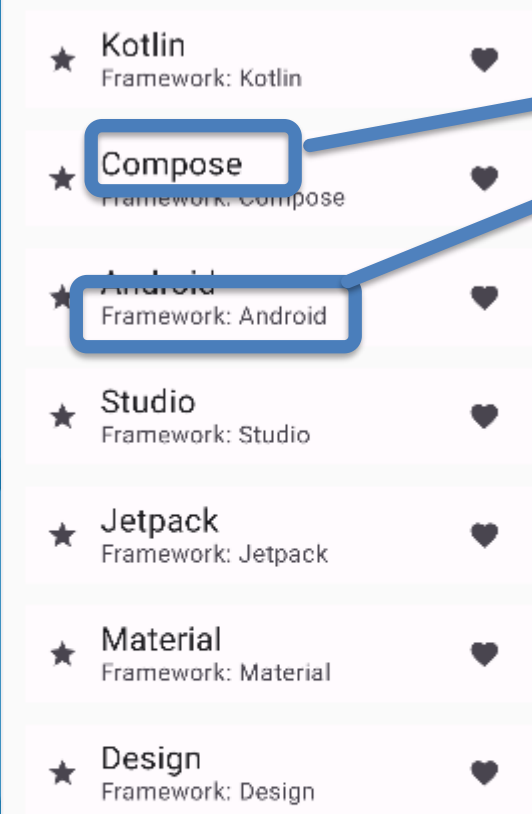
```
    ListItem(  
        headlineContent = { Text(text = word, fontSize = 24.sp) },  
        supportingContent = { Text(text = "Framework: $word", fontSize = 18.sp) },  
        leadingContent = {  
            Icon(  
                imageVector = Icons.Default.Star,  
                contentDescription = "Ikona gwiazdy"  
            )  
        },  
        trailingContent = {  
            IconButton(onClick = { }) {  
                Icon(  
                    imageVector = Icons.Default.Favorite,  
                    contentDescription = "Ulubione"  
                )  
            }  
        },  
        modifier = Modifier.padding(vertical = 8.dp)  
    )  
}
```

5:23 [status icons]



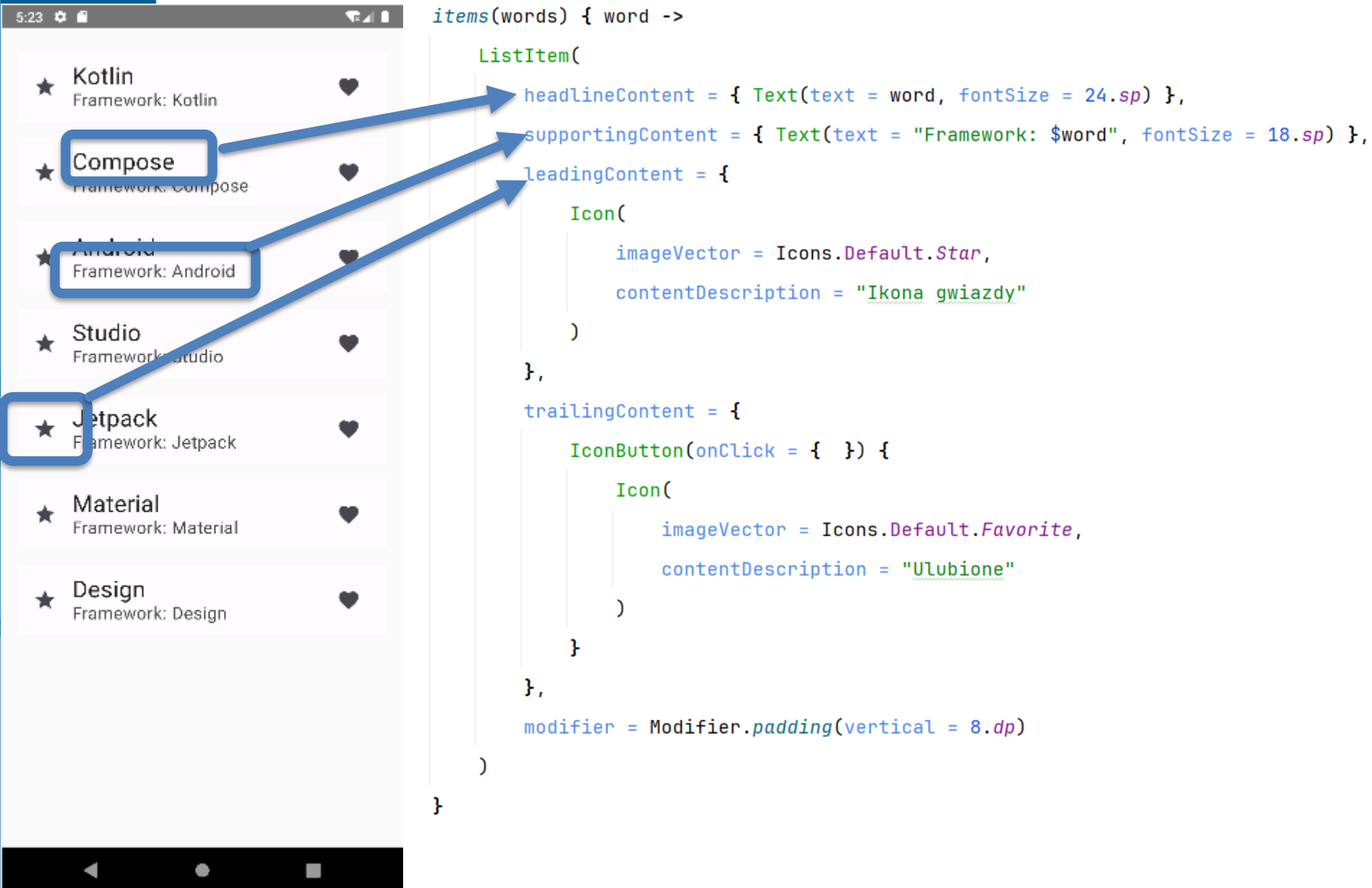
```
items(words) { word ->
    ListItem(
        headlineContent = { Text(text = word, fontSize = 24.sp) },
        supportingContent = { Text(text = "Framework: $word", fontSize = 18.sp) },
        leadingContent = {
            Icon(
                imageVector = Icons.Default.Star,
                contentDescription = "Ikona gwiazdy"
            )
        },
        trailingContent = {
            IconButton(onClick = { }) {
                Icon(
                    imageVector = Icons.Default.Favorite,
                    contentDescription = "Ulubione"
                )
            }
        },
        modifier = Modifier.padding(vertical = 8.dp)
    )
}
```

5:23 [status icons]



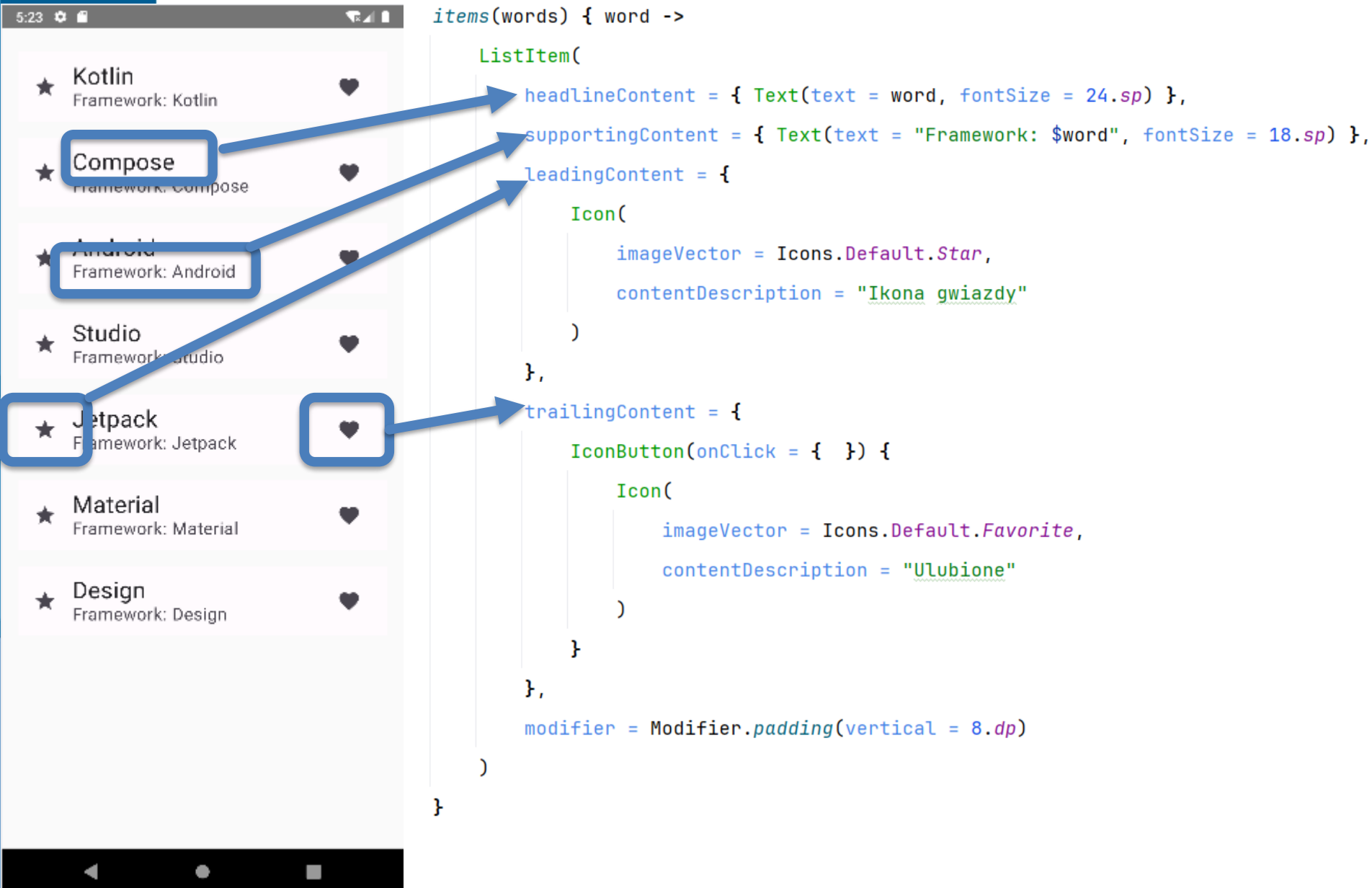
```
items(words) { word ->
    ListItem(
        headlineContent = { Text(text = word, fontSize = 24.sp) },
        supportingContent = { Text(text = "Framework: $word", fontSize = 18.sp) },
        leadingContent = {
            Icon(
                imageVector = Icons.Default.Star,
                contentDescription = "Ikona gwiazdy"
            )
        },
        trailingContent = {
            IconButton(onClick = { }) {
                Icon(
                    imageVector = Icons.Default.Favorite,
                    contentDescription = "Ulubione"
                )
            }
        },
        modifier = Modifier.padding(vertical = 8.dp)
    )
}
```

5:23



```
items(words) { word ->
    ListItem(
        headlineContent = { Text(text = word, fontSize = 24.sp) },
        supportingContent = { Text(text = "Framework: $word", fontSize = 18.sp) },
        leadingContent = {
            Icon(
                imageVector = Icons.Default.Star,
                contentDescription = "Ikona gwiazdy"
            )
        },
        trailingContent = {
            IconButton(onClick = { }) {
                Icon(
                    imageVector = Icons.Default.Favorite,
                    contentDescription = "Ulubione"
                )
            }
        },
        modifier = Modifier.padding(vertical = 8.dp)
    )
}
```

5:23



The screenshot shows an Android application interface on the left and its Kotlin code on the right. The interface displays a list of frameworks: Kotlin, Compose, Android, Studio, Jetpack, Material, and Design. Each item has a star icon and a heart icon. Blue boxes highlight specific UI elements, and blue arrows point from these boxes to the corresponding code properties in the `ListItem` function.

UI Elements (Left):

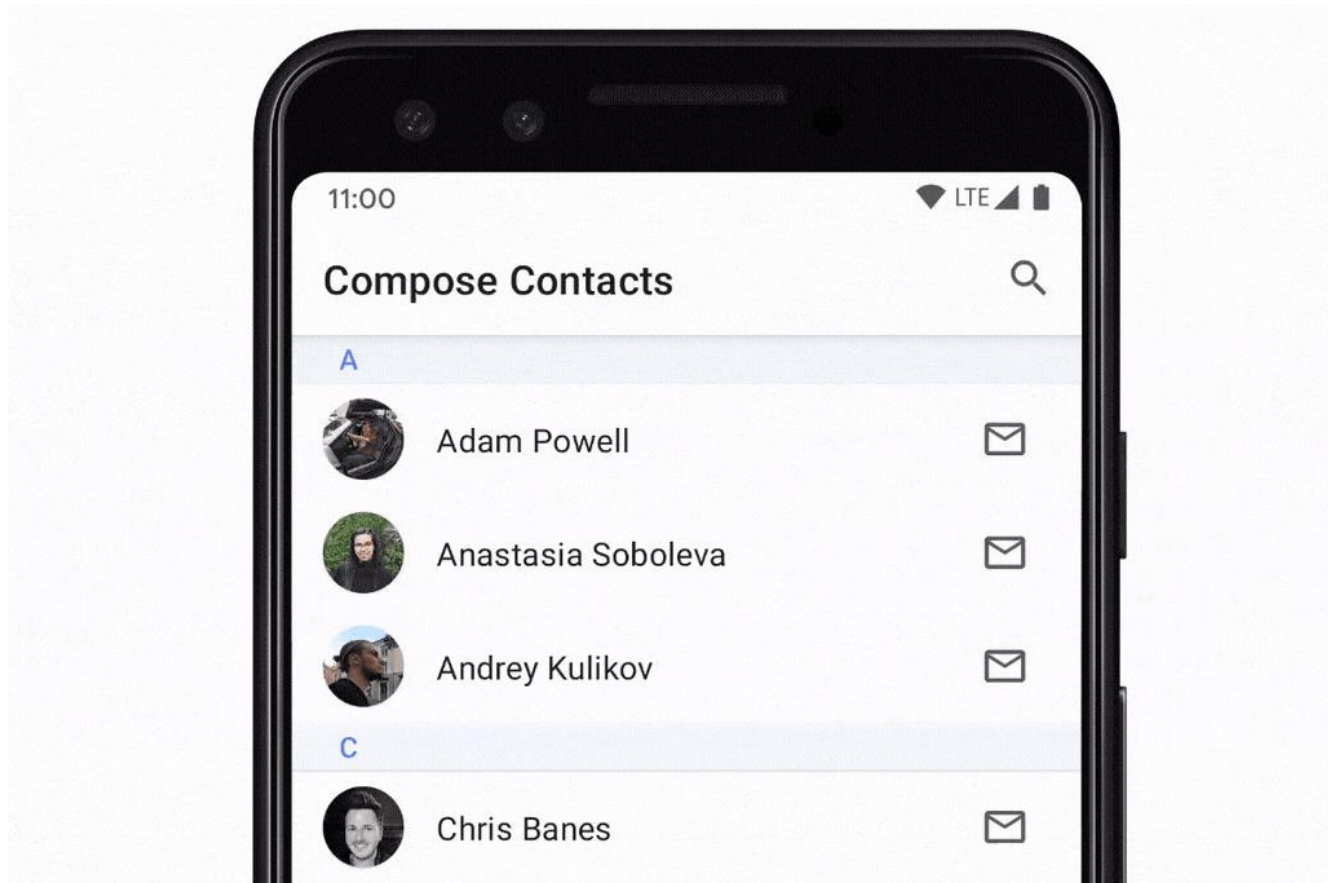
- Star icon (next to Kotlin, Compose, Android, Studio, Jetpack, Material, Design)
- Compose (highlighted with a blue box)
- Android (highlighted with a blue box)
- Studio (highlighted with a blue box)
- Jetpack (highlighted with a blue box)
- Material (highlighted with a blue box)
- Design (highlighted with a blue box)
- Heart icon (next to Kotlin, Compose, Android, Studio, Jetpack, Material, Design)

Kotlin Code (Right):

```
items(words) { word ->
    ListItem(
        headlineContent = { Text(text = word, fontSize = 24.sp) },
        supportingContent = { Text(text = "Framework: $word", fontSize = 18.sp) },
        leadingContent = {
            Icon(
                imageVector = Icons.Default.Star,
                contentDescription = "Ikona gwiazdy"
            )
        },
        trailingContent = {
            IconButton(onClick = { }) {
                Icon(
                    imageVector = Icons.Default.Favorite,
                    contentDescription = "Ulubione"
                )
            }
        },
        modifier = Modifier.padding(vertical = 8.dp)
    )
}
```

Arrows indicate the mapping:

- Star icon (next to Kotlin) → `Icons.Default.Star`
- Compose → `Text(text = word, fontSize = 24.sp)`
- Android → `Text(text = "Framework: $word", fontSize = 18.sp)`
- Studio → `Text(text = "Framework: $word", fontSize = 18.sp)`
- Jetpack → `Text(text = "Framework: $word", fontSize = 18.sp)`
- Material → `Text(text = "Framework: $word", fontSize = 18.sp)`
- Design → `Text(text = "Framework: $word", fontSize = 18.sp)`
- Heart icon (next to Kotlin) → `Icons.Default.Favorite`



```
data class Section(val title: String, val items: List<String>)

@OptIn(ExperimentalFoundationApi::class)
@Composable
fun SectionList() {
    val sections = listOf(
        Section(title: "Języki programowania", listOf("Kotlin", "Java", "Python")),
        Section(title: "Frameworki", listOf("Compose", "Spring", "Django")),
        Section(title: "Narzędzia", listOf("Android Studio", "IntelliJ IDEA", "VS Code"))
    )

    LazyColumn(
        modifier = Modifier.fillMaxSize(),
        contentPadding = PaddingValues(16.dp))
    {
        sections.forEach { section ->
            stickyHeader { SectionHeader(title = section.title) }
            items(section.items) { item -> SectionItem(item = item) }
        }
    }
}
```

5:46

Języki programowania

Kotlin
Java
Python

Frameworki

Compose
Spring
Django

Narzędzia

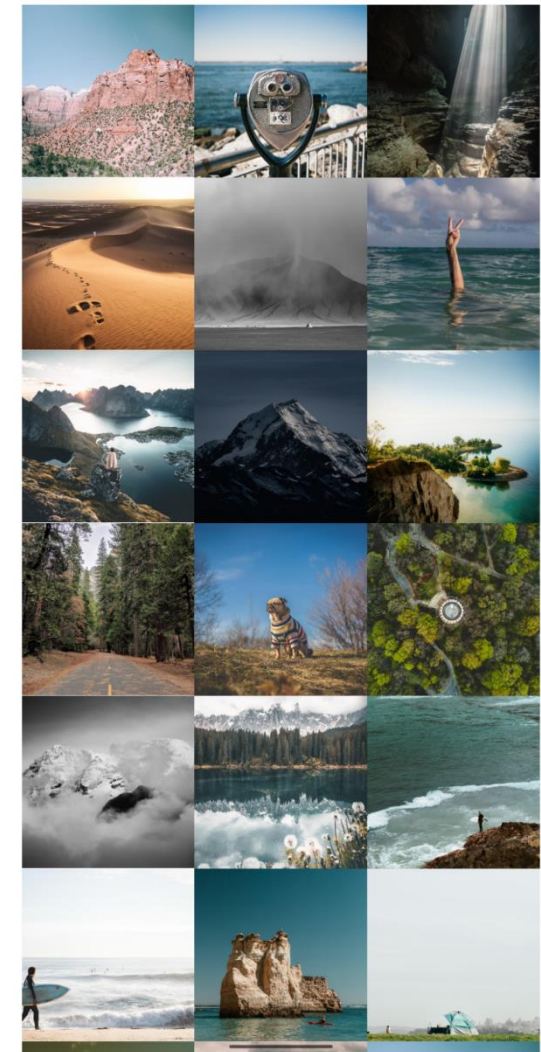
Android Studio
IntelliJ IDEA
VS Code


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

11:00



Grid



LazyVerticalStaggeredGrid

```

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

