Unit 3 — Display Lists & Material Design

Mobile Application Development — Lecture 3

Masoud Hamad

State University of Zanzibar (SUZA)

Semester II, 2025/2026

Outline

Scrollable Column

For a **small, fixed** number of items:

```
Column(
    modifier = Modifier.verticalScroll(rememberScrollState())
) {
    repeat(20) { Text("Item #$it") }
}
```

Warning

Do **NOT** use `verticalScroll` for large lists — all items render at once, wasting memory and CPU.

LazyColumn — The Right Way

```
LazyColumn {
    items(students) { student ->
        StudentRow(student)
    }
}
```

- Only renders **visible** items (like RecyclerView)
- Efficient for thousands of items
- `LazyRow` for horizontal scrolling

Item DSL

```
LazyColumn {
    item {
        Text("Header", style = MaterialTheme.typography.titleLarge)
    }

    items(count = 10) { index ->
        Text("Row $index")
    }

    items(students, key = { it.id }) { student ->
        StudentCard(student)
    }

    item { Text("Footer") }
}
```

Providing a `key` improves performance and preserves state across list changes.

Student Card

```
data class Student(val id: Int, val name: String, val course: String)

@Composable
fun StudentCard(student: Student) {
    Card(
        modifier = Modifier.fillMaxWidth().padding(8.dp),
        elevation = CardDefaults.cardElevation(defaultElevation = 4.dp)
    ) {
        Column(Modifier.padding(16.dp)) {
            Text(student.name, fontWeight = FontWeight.Bold)
            Text(student.course, color = Color.Gray)
        }
    }
}
```

Student List

```
@Composable
fun StudentList(students: List<Student>) {
    LazyColumn(contentPadding = PaddingValues(8.dp)) {
        items(students, key = { it.id }) { s ->
            StudentCard(s)
        }
    }
}
```

LazyVerticalGrid

```
LazyVerticalGrid(
    columns = GridCells.Fixed(2),
    contentPadding = PaddingValues(8.dp)
) {
    items(products) { product -> ProductCard(product) }
}
```

`GridCells.Adaptive(minSize = 120.dp)` adapts columns to screen size.

Clickable Items

```
@Composable
fun StudentCard(student: Student, onClick: (Student) -> Unit) {
    Card(
        modifier = Modifier.fillMaxWidth().padding(8.dp)
            .clickable { onClick(student) }
    ) { /* ... */ }
}
```

Hoist the click handler to the parent (e.g., to navigate).

Section Headers & Dividers

```
LazyColumn {
    val grouped = students.groupBy { it.course }
    grouped.forEach { (course, list) ->
        stickyHeader {
            Text(course,
                modifier = Modifier.fillMaxWidth()
                    .background(Color.LightGray).padding(8.dp))
        }
        items(list) { StudentCard(it) }
    }
}
```

Item Placement Animations

```
@OptIn(ExperimentalFoundationApi::class)
LazyColumn {
    items(items, key = { it.id }) { item ->
        Row(Modifier.animateItemPlacement()) { /* ... */ }
    }
}
```

Pull-to-Refresh

```
PullToRefreshBox(
    isRefreshing = refreshing,
    onRefresh = { viewModel.refresh() }
) {
    LazyColumn { /* ... */ }
}
```

Performance Tips

- Always provide a **stable key** in `items(list, key = ...)`
- Avoid heavy work inside composable bodies — hoist to ViewModel
- Use `derivedStateOf` for computed state
- Prefer `LazyColumn` over `Column(verticalScroll)` for > 20 items
- Profile with Layout Inspector / Compose Metrics if frames drop

Summary

- `LazyColumn / LazyRow / LazyVerticalGrid` are the go-to for lists
- `items(list, key = { it.id })` for efficient, stable rendering
- Hoist click handlers; keep list items stateless

Next: Persisting data with Room & DataStore.

Questions?