Unit 4 — Navigation & App Architecture

Mobile Application Development — Lecture 4

Masoud Hamad

State University of Zanzibar (SUZA)

Semester II, 2025/2026

Outline

Activities

- An **Activity** = a screen/task the user can interact with
- A Compose app typically uses a **single Activity** with many Composable screens
- Lifecycle: `onCreate` → `onStart` → `onResume` → `onPause` → `onStop` → `onDestroy`

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent { MyApp() }
    }
}
```

Setup

```
// build.gradle.kts
implementation("androidx.navigation:navigation-compose:2.7.0")
```

```
@Composable
fun MyApp() {
    val navController = rememberNavController()

    NavHost(navController, startDestination = "home") {
        composable("home") { HomeScreen(navController) }
        composable("details/{id}") { backStackEntry ->
            val id = backStackEntry.arguments?.getString("id")
            DetailScreen(id)
        }
    }
}
```

Navigating

```
// Forward navigation
Button(onClick = { navController.navigate("details/42") }) {
    Text("View details")
}

// Back navigation
navController.popBackStack()

// Pop to a specific route
navController.navigate("home") {
    popUpTo("home") { inclusive = true }
}
```

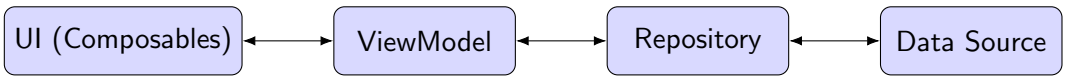
Typed Arguments

```
composable(
    route = "profile/{userId}",
    arguments = listOf(navArgument("userId") {
        type = NavType.IntType
    })
) { entry ->
    val userId = entry.arguments?.getInt("userId")
    ProfileScreen(userId)
}
```

Scaffold

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun MainScreen() {
    Scaffold(
        topBar = { TopAppBar(title = { Text("Home") }) },
        floatingActionButton = {
            FloatingActionButton(onClick = { }) {
                Icon(Icons.Default.Add, null)
            }
        }
    ) { innerPadding ->
        Column(Modifier.padding(innerPadding)) {
            // content
        }
    }
}
```

MVVM + Unidirectional Data Flow



- **UI** observes state from ViewModel
- **ViewModel** holds state, survives config changes
- **Repository** hides data-source details
- **Data Source:** Room DB, Retrofit, DataStore

ViewModel

```
class CounterViewModel : ViewModel() {
    private val _count = MutableStateFlow(0)
    val count: StateFlow<Int> = _count.asStateFlow()

    fun increment() { _count.value++ }
}
```

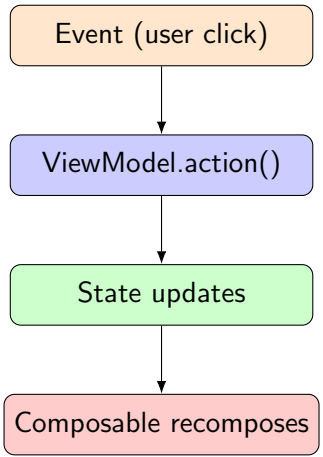
```
@Composable
fun CounterScreen(viewModel: CounterViewModel = viewModel()) {
    val count by viewModel.count.collectAsState()
    Button(onClick = viewModel::increment) {
        Text("Count: $count")
    }
}
```

StateFlow vs MutableState

- `MutableState` — local UI state, scoped to a Composable
- `StateFlow` — architectural state from ViewModel to many collectors
- `StateFlow` is **testable** and **lifecycle-aware**

Rule of thumb: ephemeral UI state → `MutableState`. Business state → `StateFlow` in a ViewModel.

Unidirectional Data Flow (UDF)



Benefits: predictable, testable, single source of truth.

Multi-Screen App

```
enum class Screen { Home, Detail, Settings }

@Composable
fun App() {
    val nav = rememberNavController()
    NavHost(nav, startDestination = Screen.Home.name) {
        composable(Screen.Home.name) { HomeScreen(
            onItemClick = { id -> nav.navigate("${Screen.Detail.name}/$id") },
            onSettings = { nav.navigate(Screen.Settings.name) }
        )}
        composable("${Screen.Detail.name}/{id}") { entry ->
            DetailScreen(entry.arguments?.getString("id") ?: "")
        }
        composable(Screen.Settings.name) { SettingsScreen() }
    }
}
```

Summary

- Single-activity + `NavHost` is the modern Android pattern
- `Scaffold` provides structure (app bar, FAB, drawer, snackbar)
- MVVM + UDF separate UI from business logic
- ViewModel survives config changes and exposes state via `StateFlow`

Next: Lists, LazyColumn and scrollable content.

Questions?