Workshop: Architecture & Navigation

Class 5: MVVM & Multi-Screen Apps Time: 75 - 90 Minutes

Introduction

So far, we have built everything in `MainActivity`. If we kept doing that, our file would be 5,000 lines long. Today we graduate to **Architecture**. We will separate our code into:

1. **UI (View):** Draws the screen.
2. **Logic (ViewModel):** Holds the data.
3. **Navigation:** Moves between screens.

We will convert our "Task App" into a 2-screen app: **The List** -> **The Details**.

Part 0: Theory - The "Chef & Waiter" (MVVM)

MVVM (Model - View - ViewModel) is the standard industry architecture.

- **The View (The Waiter):** The UI (Composables). It is "dumb". It doesn't know how to cook; it just takes the order (clicks) and brings the food (data) to the user.
- **The ViewModel (The Chef):** The Logic. It stays in the kitchen. It holds the data, fetches from databases, and calculates things. It never touches the UI directly.

Why? If you rotate your phone, the Activity (View) is destroyed and recreated. If your data is in the View, it disappears. The **ViewModel** survives rotation. It is safe storage.

Step 1: Add Navigation Dependencies

Compose Navigation isn't always installed by default.

1. Open `build.gradle.kts (Module :app)`.
2. In the `dependencies { }` block, add this line:

   implementation("androidx.navigation:navigation-compose:2.7.7") // Version may vary

   (Note: If the IDE suggests a newer version, use it).

3. Click **Sync Now** at the top right.

OBS: Depending on the Android version you need to update both the gradle files, like this:

1. Update Project-Level Gradle:
   Open build.gradle (Project: TaskManager). Replace the contents with this:

```
buildscript {
    ext {
        compose_version = '1.2.1'  // Stable version for this setup
        kotlin_version = '1.7.10'
    }
}

plugins {
    id 'com.android.application' version '7.2.2' apply false
    id 'com.android.library' version '7.2.2' apply false
    id 'org.jetbrains.kotlin.android' version '1.7.10' apply false
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

2. Update Module-Level Gradle:
   Open build.gradle (Module: app). Replace the contents with this:

```
plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
}

android {
    compileSdk 33

    defaultConfig {
        applicationId "com.example.taskmanager" // Check your package name!
        minSdk 21
        targetSdk 33
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
```

```
        vectorDrawables {
            useSupportLibrary true
        }
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
    buildFeatures {
        compose true
    }
    composeOptions {
        kotlinCompilerExtensionVersion '1.3.0'
    }
    packagingOptions {
        resources {
            excludes += '/META-INF/{AL2.0,LGPL2.1}'
        }
    }
}

dependencies {
    // Navigation (Added for this class)
    implementation("androidx.navigation:navigation-compose:2.5.3")

    // Core Android & Compose
    implementation 'androidx.core:core-ktx:1.9.0'
    implementation "androidx.compose.ui:ui:$compose_version"
    implementation "androidx.compose.material:material:$compose_version"
    implementation "androidx.compose.ui:ui-tooling-preview:$compose_version"
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.5.1'
    implementation 'androidx.activity:activity-compose:1.6.1'
```

```
    // Testing
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
    androidTestImplementation "androidx.compose.ui:ui-test-junit4:$compose_version"
    debugImplementation "androidx.compose.ui:ui-tooling:$compose_version"
    debugImplementation "androidx.compose.ui:ui-test-manifest:$compose_version"
}
```

3. Sync: Click "Sync Now" at the top right and wait for the build to finish.

Step 2: Create the ViewModel (The Chef)

We need a place to hold our Task List so it isn't hardcoded inside the UI.

1. Right-click your package folder > **New** > **Kotlin Class/File**.
2. Name: `TaskViewModel`.
3. Paste this code:

```
import androidx.lifecycle.ViewModel

// Inheriting from ViewModel gives us "Superpowers" (Surviving rotation)

class TaskViewModel : ViewModel() {

    // 1. THE DATA (Our Menu)
    private val _tasks = listOf(
        TaskItem(1, "Complete Lab 4", "Today, 5:00 PM", "Finish the checkbox logic."),
        TaskItem(2, "Buy Groceries", "Tomorrow", "Milk, Eggs, Bread."),
        TaskItem(3, "Call Mom", "Sunday", "Ask about the recipe."),
        TaskItem(4, "Fix Android Bug", "ASAP", "Crash on line 42.")
    )

    // Expose the list to the UI
    val tasks: List<TaskItem> = _tasks

    // 2. THE LOGIC (Cooking)
    // A function to find a specific task by its ID
    fun getTask(id: Int): TaskItem? {
        return _tasks.find { it.id == id }
    }
}
```

Note: You need to update your `TaskItem` data class in `MainActivity` to include a new field: `val details: String`.

Update `TaskItem` in MainActivity:

```
data class TaskItem(
    val id: Int,
    val label: String,
    val dueTime: String,
    val details: String // New field!
)
```

Step 3: Create the Detail Screen

We need a destination to navigate *to*.

1. In `MainActivity.kt`, create a new Composable called `DetailScreen`.
2. It needs to accept a `taskId` (so it knows which task to show) and the `viewModel` (to find the data).

```
@Composable
fun DetailScreen(taskId: Int, viewModel: TaskViewModel, onBack: () -> Unit) {
    // ASK THE VIEWMODEL FOR DATA
    val task = viewModel.getTask(taskId)
    if (task == null) {
        Text("Task not found")
        return
    }

    // UI CODE
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
    ) {
        Button(onClick = onBack) {
            Text("Back")
        }
        Spacer(modifier = Modifier.height(16.dp))
        Text(text = task.label, fontSize = 30.sp, fontWeight = FontWeight.Bold)
        Text(text = "Due: ${task.dueTime}", color = Color.Gray)
```

```
        Spacer(modifier = Modifier.height(24.dp))
        // The Details
        Text(text = "Description:", fontWeight = FontWeight.Bold)
        Text(text = task.details, fontSize = 18.sp)
    }

}
```

Step 4: The Navigator (NavHost)

Concept: Routes Navigation in Compose works like a website URLs.

- Home page: `"task_list"`
- Detail page: `"detail_screen/1"` or `"detail_screen/2"`

Concept: NavController This is the object that actually performs the action `navigate()`.

1. Create a new Composable `AppNavigation` in `MainActivity.kt`.
2. This will act as the "Traffic Controller" for the whole app.

```
@Composable
fun AppNavigation() {
    // 1. Create the Controller
    val navController = rememberNavController()
    // 2. Create the ViewModel (One instance shared across screens)
    // We use the library function 'viewModel()' to get it.
    val viewModel: TaskViewModel = viewModel()
    // 3. The NavHost (The Stage)
    NavHost(navController = navController, startDestination = "task_list") {
        // ROUTE 1: The List
        composable("task_list") {
            TaskScreen(
                viewModel = viewModel,
                onTaskClick = { taskId ->
                    // Navigate to details, passing the ID in the URL
                    navController.navigate("detail_screen/$taskId")
                }
            )
        }

        // ROUTE 2: The Details
        // We define a placeholder "{taskId}"
```

```
        composable(
            route = "detail_screen/{taskId}",
            arguments = listOf(navArgument("taskId") { type = NavType.IntType })
        ) { backStackEntry ->
            // Extract the ID from the URL
            val taskId = backStackEntry.arguments?.getInt("taskId") ?: 0

            DetailScreen(
                taskId = taskId,
                viewModel = viewModel,
                onBack = { navController.popBackStack() } // Go back
            )
        }
    }
}
```

Note: You might need to add imports manually. `viewModel()` comes from `androidx.lifecycle.viewmodel.compose`.

Step 5: Refactoring the List Screen

We need to update `TaskScreen` to stop creating fake data and start using the `ViewModel`. It also needs to handle clicks.

Update your `TaskScreen` signature and body:

```
@Composable
fun TaskScreen(
    viewModel: TaskViewModel, // Pass ViewModel in
    onTaskClick: (Int) -> Unit // Callback when item is clicked
) {
    // Ask ViewModel for the list
    val tasks = viewModel.tasks
    LazyColumn(
        modifier = Modifier.fillMaxSize().background(Color(0xFFF5F5F5)),
        contentPadding = PaddingValues(16.dp),
        verticalArrangement = Arrangement.spacedBy(8.dp)
    ) {
        items(tasks) { task ->
            // Pass the click event down to the Row
            TaskRow(task = task, onClick = { onTaskClick(task.id) })
        }
```

```
    }
}
```

Also update `TaskRow` to be clickable:

```
@Composable
fun TaskRow(task: TaskItem, onClick: () -> Unit) { // Add onClick param
    // ... State setup ...
    Surface(
        onClick = onClick, // Enables the Ripple effect and click
        modifier = Modifier.fillMaxWidth(),
        // ... rest of modifiers
    ) {
        // ... rest of Row code
    }
}
```

Step 6: The Final Connection

Update `onCreate` to start the Navigation system instead of just the screen.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
        AppNavigation()
    }
}
```

Phase 7: Run & Verify

1. Run the app.
2. You see the list.
3. Click "Buy Groceries".
4. **Magic:** You navigate to a new screen showing "Milk, Eggs, Bread".
5. Click "Back". You return to the list.

🧠 Why did we do this?

- **Separation:** The `DetailScreen` doesn't know about the List. The `List` doesn't know about the Details. They are independent.

- **Single Source of Truth:** The data lives in `TaskViewModel`. If we delete a task in the View Model, both screens update automatically.

Exercise: The "About" Screen (Beginner)

Goal: Practice creating a new Screen and adding a simple Navigation Route.

Scenario: The app needs a screen that explains who built it and what version it is.

Tasks:

1. **Create UI:** Create a new Composable function `AboutScreen(onBack: () -> Unit)`.
    - It should contain a `Column`, a Title "About TaskManager", and your name as the "Developer".
    - Include a Button that calls `onBack`.
2. **Add Route:** In `AppNavigation`, add a new `composable("about_screen")` block that calls your new screen.
3. **Add Entry Point:** In `TaskScreen`, add a Button at the top (or in a TopAppBar if you know how) that says "About".
4. **Connect:** When the "About" button is clicked, call `navController.navigate("about_screen")`.

Hint:

```
// Inside AppNavigation
composable("about_screen") {
    AboutScreen(onBack = { navController.popBackStack() })
}
```

Delivery zip with code and screenshot in:
https://www.dropbox.com/request/zswaQSHclOE3X3zBXclV