Workshop: The Pokedex (Networking & APIs)

Class 6: Retrofit, Coroutines & Coil Time: 90 Minutes

Introduction

Today your app stops being an island. We will connect to the internet to fetch real data using the **PokeAPI**.

The Tech Stack:

1. **Retrofit:** The library that turns our API into Kotlin code.
2. **Gson:** The translator. It turns JSON text (from the web) into Kotlin Objects.
3. **Coil:** The image loader. It fetches images from URLs and caches them.
4. **Coroutines:** The threading. It lets us download data in the background without freezing the app.

Phase 0: Dependencies & Permissions

We need to buy the tools before we build the house.

1. Add Internet Permission Open `manifests/AndroidManifest.xml`. Add this line **above** the `<application>` tag:
<uses-permission android:name="android.permission.INTERNET" />
If you skip this, your app will crash instantly.

2. Add Libraries Open `build.gradle (Module: app)`. Add these to `dependencies`:

```
// Networking (Retrofit + Gson)
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'

// Image Loading (Coil)
implementation("io.coil-kt:coil-compose:2.4.0")
```

*Click **Sync Now**.*

Phase 1: The Theory (JSON & HTTP)

Concept: JSON The internet speaks JSON. It looks like this:

```
{
  "results": [
    { "name": "bulbasaur", "url": "https://pokeapi.co/.../1/" },
    { "name": "ivysaur", "url": "https://pokeapi.co/.../2/" }
  ]
}
```

Concept: HTTP GET We send a **GET** request to a URL (Endpoint). The server replies with the JSON above.

Phase 2: The Data Model

We need Kotlin classes that match the JSON shape exactly.

1. Create a new file `PokemonModels.kt`.
2. Add these data classes:

```
// 1. The Wrapper: Represents the whole JSON response
data class PokemonResponse(
    val results: List<Pokemon>
)

// 2. The Item: Represents one Pokemon in the list

data class Pokemon(
    val name: String,
    val url: String
) {
    // Helper to extract ID from URL (e.g., ".../pokemon/1/")
    val id: String
        get() = url.split("/").dropLast(1).last()
    // Helper to construct the Image URL
    val imageUrl: String
        get() =
"https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/$id.png"
}
```

Phase 3: The API Interface (Retrofit)

Concept: The Interface We don't write the networking code manually. We define an **Interface**, and Retrofit generates the code for us.

Concept: Suspend Functions Network calls take time (100ms - 3s). If we run this on the **Main Thread** (UI Thread), the app freezes. The `suspend` keyword tells Kotlin: *"Pause this function here, do the work in the background, and resume when done."*

1. Create a file `PokeApi.kt`.

```
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory
import retrofit2.http.GET

// 1. Define the endpoints
interface PokeApiService {
    @GET("pokemon?limit=50") // Fetch first 50
    suspend fun getPokemonList(): PokemonResponse
}

// 2. Create the Singleton Instance
object RetrofitInstance {
    private val retrofit = Retrofit.Builder()
        .baseUrl("https://pokeapi.co/api/v2/")
        .addConverterFactory(GsonConverterFactory.create()) // Use Gson to parse
        .build()
    val api: PokeApiService = retrofit.create(PokeApiService::class.java)
}
```

Phase 4: The ViewModel (State Management)

Concept: UI States When loading data, the UI can be in 3 states:

1. **Loading:** Show a spinner.
2. **Success:** Show the data.
3. **Error:** Show a "Retry" button.

We use a **Sealed Interface** to represent this strictly.

1. Create `PokemonViewModel.kt`.

```
import androidx.lifecycle.ViewModel
```

```
import androidx.lifecycle.viewModelScope
import kotlinx.coroutines.launch
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.getValue
import androidx.compose.runtime.setValue

// The 3 possible states of our screen
sealed interface UiState {
    object Loading : UiState
    data class Success(val data: List<Pokemon>) : UiState
    data class Error(val message: String) : UiState
}

class PokemonViewModel : ViewModel() {
    // The State we expose to the View
    var pokemonState: UiState by mutableStateOf(UiState.Loading)
        private set

    // A list to track favorites (local memory only)
    var favoriteIds: List<String> by mutableStateOf(emptyList())
        private set

    init {
        fetchPokemon()
    }

    fun fetchPokemon() {
        // Start a Coroutine (Background thread)
        viewModelScope.launch {
            try {
                // Set state to Loading
                pokemonState = UiState.Loading

                // Make the network call
                val response = RetrofitInstance.api.getPokemonList()

                // If successful, update state
                pokemonState = UiState.Success(response.results)

            } catch (e: Exception) {
                // If failed (No internet?), update error
                pokemonState = UiState.Error("Failed to load: ${e.message}")
            }
        }
```

```
    }

    fun toggleFavorite(id: String) {
        if (favoriteIds.contains(id)) {
            favoriteIds = favoriteIds - id
        } else {
            favoriteIds = favoriteIds + id
        }
    }
}
```

Phase 5: The UI (Coil & Grid)

Concept: AsyncImage `Image` is for local resources. `AsyncImage` (from Coil) is for internet URLs. It handles downloading and caching automatically.

Concept: LazyVerticalGrid Instead of a list, let's make a grid of cards.

1. Open `MainActivity.kt`.
2. Add the imports (you might need to Alt+Enter).
3. Create the `PokedexScreen`.

```
@Composable
fun PokedexScreen(viewModel: PokemonViewModel = viewModel()) {
    val state = viewModel.pokemonState
    val favorites = viewModel.favoriteIds

    Column(modifier = Modifier.fillMaxSize().padding(16.dp)) {
        Text("Pokedex", fontSize = 30.sp, fontWeight = FontWeight.Bold)
        Spacer(modifier = Modifier.height(16.dp))

        // DECISION TREE: What do we show?

        when (state) {
            is UiState.Loading -> {
                CircularProgressIndicator(modifier = Modifier.align(Alignment.CenterHorizontally))
            }
            is UiState.Error -> {
                Text("Error: ${state.message}", color = Color.Red)
                Button(onClick = { viewModel.fetchPokemon() }) {
                    Text("Retry")
                }
```

```
            }
            is UiState.Success -> {
                PokemonGrid(
                    pokemonList = state.data,
                    favorites = favorites,
                    onFavoriteClick = { id -> viewModel.toggleFavorite(id) }
                )
            }
        }
    }
}
```

4. Create the `PokemonGrid` and `PokemonCard`.

```
@Composable
fun PokemonGrid(
    pokemonList: List<Pokemon>,
    favorites: List<String>,
    onFavoriteClick: (String) -> Unit
) {
    LazyVerticalGrid(
        columns = GridCells.Fixed(2), // 2 items per row
        verticalArrangement = Arrangement.spacedBy(8.dp),
        horizontalArrangement = Arrangement.spacedBy(8.dp)
    ) {
        items(pokemonList.size) { index ->
            val pokemon = pokemonList[index]
            val isFav = favorites.contains(pokemon.id)

            PokemonCard(pokemon, isFav, onFavoriteClick)
        }
    }
}

@Composable
fun PokemonCard(
    pokemon: Pokemon,
    isFavorite: Boolean,
    onFavoriteClick: (String) -> Unit
) {
    Card(
            elevation = 4.dp,
            backgroundColor = Color.White
    ) {
```

```
        Column(
            horizontalAlignment = Alignment.CenterHorizontally,
            modifier = Modifier.padding(12.dp).fillMaxWidth()
        ) {
            // COIL IMAGE LOADER
            AsyncImage(
                model = pokemon.imageUrl,
                contentDescription = pokemon.name,
                modifier = Modifier.size(100.dp)
            )

            Text(
                text = pokemon.name.replaceFirstChar { it.uppercase() },
                fontWeight = FontWeight.Bold,
                fontSize = 18.sp
            )

            // Favorite Button
            IconButton(onClick = { onFavoriteClick(pokemon.id) }) {
                Icon(
                    imageVector = if (isFavorite) Icons.Default.Favorite else
Icons.Default.FavoriteBorder,
                    contentDescription = "Favorite",
                    tint = if (isFavorite) Color.Red else Color.Gray
                )
            }
        }
    }
}
```

Phase 6: Run It

1. Update `onCreate` to call `PokedexScreen()`.
2. Run the app.
3. **Note:** If you see images, CONGRATULATIONS! You just built a network-connected app.
4. If it crashes, check the Internet Permission in `AndroidManifest.xml`.

🛑 Homework

The "Shiny" Mode

Goal: Modify the data model and UI logic.

- **Task:** Add a `Switch` at the top of the screen labeled "Show Shiny".
- **Logic:** When enabled, update the `Pokemon` model logic to return the URL for the shiny sprite (usually just add `/shiny/` to the URL path in the `imageUrl` getter).
- **Result:** All images should reload with the different colored shiny versions.

Delivery zip with code and screenshot in:
https://www.dropbox.com/request/Fu4aNPNGcxA489d07gqX

OBS: If you are having trouble on old versions of Android, please use this gradle configurations:
Project:

```
buildscript {
    ext {
        compose_version = '1.4.3'
    }
}// Top-level build file where you can add configuration options common to all
sub-projects/modules.
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 '1.8.10' apply false
}

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

Module:

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

android {
    compileSdk 33

    defaultConfig {
        applicationId "com.example.mypokeapplication"
        minSdk 32
        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 compose_version
    }
    packagingOptions {
        resources {
            excludes += '/META-INF/{AL2.0,LGPL2.1}'
        }
    }
}

dependencies {
    // Networking (Retrofit + Gson)
    implementation 'com.squareup.retrofit2:retrofit:2.9.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
    implementation "androidx.lifecycle:lifecycle-viewmodel-compose:$compose_version"

    // Image Loading (Coil)
    implementation("io.coil-kt:coil-compose:2.4.0")
```

```
    implementation 'androidx.core:core-ktx:1.7.0'
    implementation "androidx.compose.ui:ui:$compose_version"
    implementation "androidx.compose.material:material:$compose_version"
    implementation "androidx.compose.ui:ui-tooling-preview:$compose_version"
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'
    implementation 'androidx.activity:activity-compose:1.3.1'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.3.0'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.7.0'
    androidTestImplementation "androidx.compose.ui:ui-test-junit4:$compose_version"
    debugImplementation "androidx.compose.ui:ui-tooling:$compose_version"
    debugImplementation "androidx.compose.ui:ui-test-manifest:$compose_version"
}
```