Unit 5 — Connect to the Internet

Mobile Application Development — Lecture 5

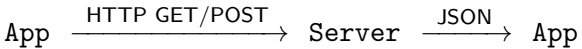
Masoud Hamad

State University of Zanzibar (SUZA)

Semester II, 2025/2026

Outline

REST APIs in Mobile Apps



- **Retrofit** — type-safe HTTP client for Android/Kotlin
- **kotlinx.serialization** — JSON $\leftrightarrow$ Kotlin data class
- **OkHttp** — Retrofit's underlying HTTP engine
- **Coil** — image loading for Compose

Permissions

```
<!-- AndroidManifest.xml -->
<uses-permission android:name="android.permission.INTERNET" />
```

Dependencies:

```
implementation("com.squareup.retrofit2:retrofit:2.9.0")
implementation("com.squareup.retrofit2:converter-kotlinx-serialization:2.9.0")
implementation("org.jetbrains.kotlinx:kotlinx-serialization-json:1.6.0")
implementation("io.coil-kt:coil-compose:2.5.0")
```

Data Model

```
@Serializable
data class Book(
    val id: String,
    val title: String,
    val author: String,
    val coverUrl: String
)
```

API Service Interface

```
interface BookApi {
    @GET("books")
    suspend fun getBooks(): List<Book>

    @GET("books/{id}")
    suspend fun getBook(@Path("id") id: String): Book

    @POST("books")
    suspend fun createBook(@Body book: Book): Book
}
```

Building Retrofit

```
object ApiClient {
    private const val BASE_URL = "https://api.example.com/"
    private val json = Json { ignoreUnknownKeys = true }

    val api: BookApi = Retrofit.Builder()
        .baseUrl(BASE_URL)
        .addConverterFactory(
            json.asConverterFactory("application/json".toMediaType())
        )
        .build()
        .create(BookApi::class.java)
}
```

Sealed UI State

Represent network state explicitly:

```
sealed interface BooksUiState {
    object Loading : BooksUiState
    data class Success(val books: List<Book>) : BooksUiState
    data class Error(val message: String) : BooksUiState
}
```

ViewModel

```
class BooksViewModel(private val repo: BookRepository) : ViewModel() {
    var state by mutableStateOf<BooksUiState>(BooksUiState.Loading)
        private set

    init { load() }

    fun load() = viewModelScope.launch {
        state = BooksUiState.Loading
        state = try {
            BooksUiState.Success(repo.fetchBooks())
        } catch (e: IOException) {
            BooksUiState.Error("Network error: ${e.message}")
        } catch (e: HttpException) {
            BooksUiState.Error("Server error ${e.code()}")
        }
    }
}
```

Compose UI

```
@Composable
fun BooksScreen(viewModel: BooksViewModel) {
    when (val s = viewModel.state) {
        BooksUiState.Loading -> CircularProgressIndicator()
        is BooksUiState.Error -> Column {
            Text(s.message)
            Button(onClick = viewModel::load) { Text("Retry") }
        }
        is BooksUiState.Success -> LazyColumn {
            items(s.books) { book -> BookCard(book) }
        }
    }
}
```

Coil — AsyncImage

```
AsyncImage(
    model = imageUrl,
    contentDescription = "Cover",
    placeholder = painterResource(R.drawable.loading),
    error = painterResource(R.drawable.error_img),
    contentScale = ContentScale.Crop,
    modifier = Modifier.size(72.dp)
)
```

HTTP Caching with OkHttp

```
val cacheSize = 10L * 1024 * 1024    // 10 MB
val client = OkHttpClient.Builder()
    .cache(Cache(context.cacheDir, cacheSize))
    .build()

Retrofit.Builder()
    .baseUrl(BASE_URL)
    .client(client)
    .build()
```

Error Handling Best Practices

- Always assume the network **can fail**
- Wrap calls in try/catch; expose errors in UI state
- Retry policies: immediate, exponential backoff, user-initiated
- Show meaningful UI states: **Loading / Success / Error / Empty**
- Never block the main thread — use `suspend` + coroutines

Summary

- Retrofit converts interface methods to HTTP calls
- `suspend` functions + coroutines = clean async code
- Model UI state explicitly as a sealed interface
- Coil handles image loading in Compose

Next: Persisting data locally with Room & DataStore.

Questions?