Connecting to the Internet

By Muhammad Haris Bin Abid

Introduction to Internet Connectivity in Android

Why internet connectivity is crucial for modern apps

What we'll cover:

- Internet permissions
- Network security configuration
- Retrofit for API calls
- JSON parsing with GSON
- Glide for image loading
- Practical examples

Permissions

- Protect the privacy of an Android user
- Declared with the `<uses-permission>` tag in the `AndroidManifest.xml`

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

<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

- INTERNET: Allows app to open network sockets
- ACCESS_NETWORK_STATE: Allows checking network connectivity status

Permissions granted to your app

- Permissions can be granted during installation or runtime, depending on protection level.
- Each permission has a protection level: normal, signature, or dangerous.
- For permissions granted during runtime, prompt users to explicitly grant or deny access to your app.

Permission protection levels

Protection Level	Granted when?	Must prompt before use?	Examples
Normal	Install time	No	`ACCESS_WIFI_STATE`, `BLUETOOTH`, `VIBRATE`, `INTERNET`
Signature	Install time	No	N/A
Dangerous	Runtime	Yes	`GET_ACCOUNTS`, `CAMERA`, `CALL_PHONE`

Add permissions to the manifest

In AndroidManifest.xml:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sampleapp">
    <uses-permission android:name="android.permission.USE_BIOMETRIC" />
    <application>
        <activity
            android:name=".MainActivity" ... >
            ...
        </activity>
    </application>
</manifest>
```

Request dangerous permissions

- Prompt the user to grant the permission when they try to access functionality that requires a dangerous permission.
- Explain to the user why the permission is needed.
- Fall back gracefully if the user denies the permission (app should still function).

App permissions best practices

- Only use the permissions necessary for your app to work.
- Pay attention to permissions required by libraries.
- Be transparent.
- Make system accesses explicit.

Connect to, and use, network resources

Checking Network Connectivity

```
fun isNetworkAvailable(context: Context): Boolean {
    val connectivityManager = context.getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        val network = connectivityManager.activeNetwork ?: return false
        val capabilities = connectivityManager.getNetworkCapabilities(network) ?: return false
        return capabilities.hasCapability(NetworkCapabilities.NET_CAPABILITY_INTERNET)
    } else {
        val networkInfo = connectivityManager.activeNetworkInfo ?: return false
        return networkInfo.isConnected
    }
}
```

Introduction to Retrofit

Retrofit: Type-safe HTTP client for Android by Square

Benefits:

- Converts HTTP API into Kotlin interface
- Handles URL manipulation, query parameters, and multipart requests
- Supports synchronous and asynchronous requests
- Configurable converters (JSON, XML, Protocol Buffers)
- Integrates well with Kotlin Coroutines

Setting Up Retrofit - Dependencies

[versions]

Your existing versions

retrofit = "2.9.0"

okhttp = "4.11.0"

gson = "2.10.1"

[libraries]

Your existing libraries remain as they are

retrofit = { group = "com.squareup.retrofit2", name = "retrofit", version.ref = "retrofit" }

retrofit-converter-gson = { group = "com.squareup.retrofit2", name = "converter-gson", version.ref = "retrofit" }

okhttp-logging = { group = "com.squareup.okhttp3", name = "logging-interceptor", version.ref = "okhttp" }

gson = { group = "com.google.code.gson", name = "gson", version.ref = "gson" }

Setting Up Retrofit - Dependencies

```
dependencies {
    // Your existing dependencies
    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.appcompat)
    implementation(libs.material)
    implementation(libs.androidx.activity)
    implementation(libs.androidx.constraintlayout)

    // Retrofit and networking dependencies
    implementation(libs.retrofit)
    implementation(libs.retrofit.converter.gson)
    implementation(libs.okhttp.logging)
    implementation(libs.gson)

    // Your existing test dependencies
    testImplementation(libs.junit)
    androidTestImplementation(libs.androidx.junit)
    androidTestImplementation(libs.androidx.espresso.core)
}
```

Creating Retrofit Interface

- Create an interface that defines your API endpoints
- Each method represents an HTTP request
- Annotations specify request type (@GET, @POST, etc.)

kotlin

```kotlin
interface ApiService {
    @GET("users")
    fun getUsers(): Call<List<User>>

    @GET("users/{id}")
    fun getUserById(@Path("id") userId: Int): Call<User>

    @POST("users")
    fun createUser(@Body user: User): Call<User>

    @GET("posts")
    fun getPosts(@Query("userId") userId: Int? = null): Call<List<Post>>
}
```

Setting Up Retrofit Instance

```
object RetrofitClient {
    private const val BASE_URL = "https://api.example.com/"

    private val loggingInterceptor = HttpLoggingInterceptor().apply {
        level = if (BuildConfig.DEBUG)
            HttpLoggingInterceptor.Level.BODY
        else
            HttpLoggingInterceptor.Level.NONE
    }

    private val client = OkHttpClient.Builder()
        .addInterceptor(loggingInterceptor)
        .connectTimeout(15, TimeUnit.SECONDS)
        .readTimeout(15, TimeUnit.SECONDS)
        .build()

    private val retrofit = Retrofit.Builder()
        .baseUrl(BASE_URL)
        .addConverterFactory(GsonConverterFactory.create())
        .client(client)
        .build()

    val apiService: ApiService = retrofit.create(ApiService::class.java)
}
```

Introduction to GSON

- GSON: Google's library for converting Java/Kotlin Objects to/from JSON
- Automatically handles serialization (object → JSON) and deserialization (JSON → object)
- Works seamlessly with Retrofit via GsonConverterFactory
- Handles complex nested objects and collections

JSON Parsing with GSON

- Define Kotlin data classes that match your JSON structure
- Use annotations to customize field names

kotlin

```kotlin
// JSON: {"id": 1, "name": "John", "email": "john@example.com", "profile_pic": "http://..."}
data class User(
    val id: Int,
    val name: String,
    val email: String,
    @SerializedName("profile_pic") val profilePic: String
)

// JSON: {"id": 1, "userId": 1, "title": "Post Title", "body": "Post content..."}
data class Post(
    val id: Int,
    val userId: Int,
    val title: String,
    val body: String
)
```

Making Synchronous API Calls

```
// WARNING: This code must NOT run on the main thread
fun getUserSync(userId: Int): User? {
    val call = RetrofitClient.apiService.getUserById(userId)
    val response = call.execute() // Blocks until the response is ready

    if (response.isSuccessful) {
        return response.body()
    } else {
        Log.e("API", "Error: ${response.code()}")
        return null
    }
}

// Must be called from a background thread
Thread {
    val user = getUserSync(1)
    // Use the result
    runOnUiThread {
        // Update UI with user
    }
}.start()
```

Making Asynchronous API Calls

```
// This can be called from any thread
fun getUserAsync(userId: Int, callback: (User?) -> Unit) {
    val call = RetrofitClient.apiService.getUserById(userId)

    call.enqueue(object : Callback<User> {
        override fun onResponse(call: Call<User>, response: Response<User>) {
            if (response.isSuccessful) {
                callback(response.body())
            } else {
                Log.e("API", "Error: ${response.code()}")
                callback(null)
            }
        }

        override fun onFailure(call: Call<User>, t: Throwable) {
            Log.e("API", "Failed: ${t.message}")
            callback(null)
        }
    })
}

// Usage (can be called from main thread)
getUserAsync(1) { user ->
    if (user != null) {
        // Update UI with user
    } else {
        // Show error
    }
}
```

Glide

- Third-party image-loading library in Android
- Focused on performance for smoother scrolling
- Supports images, video stills, and animated GIFs

Adding dependency in gradle

```
[versions]
# Other version declarations
glide = "4.16.0"  # Using the latest stable version

[libraries]
# Other library declarations
glide-core = { group = "com.github.bumptech.glide", name = "glide", version.ref = "glide" }
glide-compiler = { group = "com.github.bumptech.glide", name = "compiler", version.ref = "glide" }

[plugins]
# Plugin declarations if any
```

In build.gradle:

```
dependencies {
// Other dependencies
implementation(libs.glide.core)
annotationProcessor(libs.glide.compiler)
}
```

Load an Image

```
Glide.with(context)
    .load(url)
    .into(imageView);
```

Thank you