The Android Prep Lab (Kotlin)

Part - 6 of 10

Android Networking

What is the role of networking in Android apps?

Networking in Android apps is used to communicate with remote servers, APIs, and cloud services. It enables apps to:

- Fetch data from external servers (e.g., weather, news, social media).
- Send data to backend servers (e.g., posting content, submitting forms).
- Enable real-time communication (e.g., chat apps, notifications).

Effective networking is essential for the functionality and user experience of modern Android apps, and handling it correctly ensures apps are scalable, efficient, and secure.

What are the different methods of networking in Android?

- **HttpURLConnection**: The native Android API for making HTTP requests. It's low-level and requires manual handling of request/response parsing, which can be tedious.
- **OkHttp**: A popular, modern HTTP client that simplifies network requests with features like connection pooling, automatic retries, and interceptors.
- **Retrofit**: A high-level REST API client built on top of OkHttp. It abstracts much of the HTTP request handling and provides integration with JSON parsing libraries like Gson or Moshi.
- **Volley**: A library by Google designed for managing network requests. It simplifies the process but is not as flexible or feature-rich as Retrofit.
- **Ktor**: A Kotlin-based HTTP client for building asynchronous network requests, offering a coroutine-based approach.

What is the difference between HttpURLConnection and OkHttp?

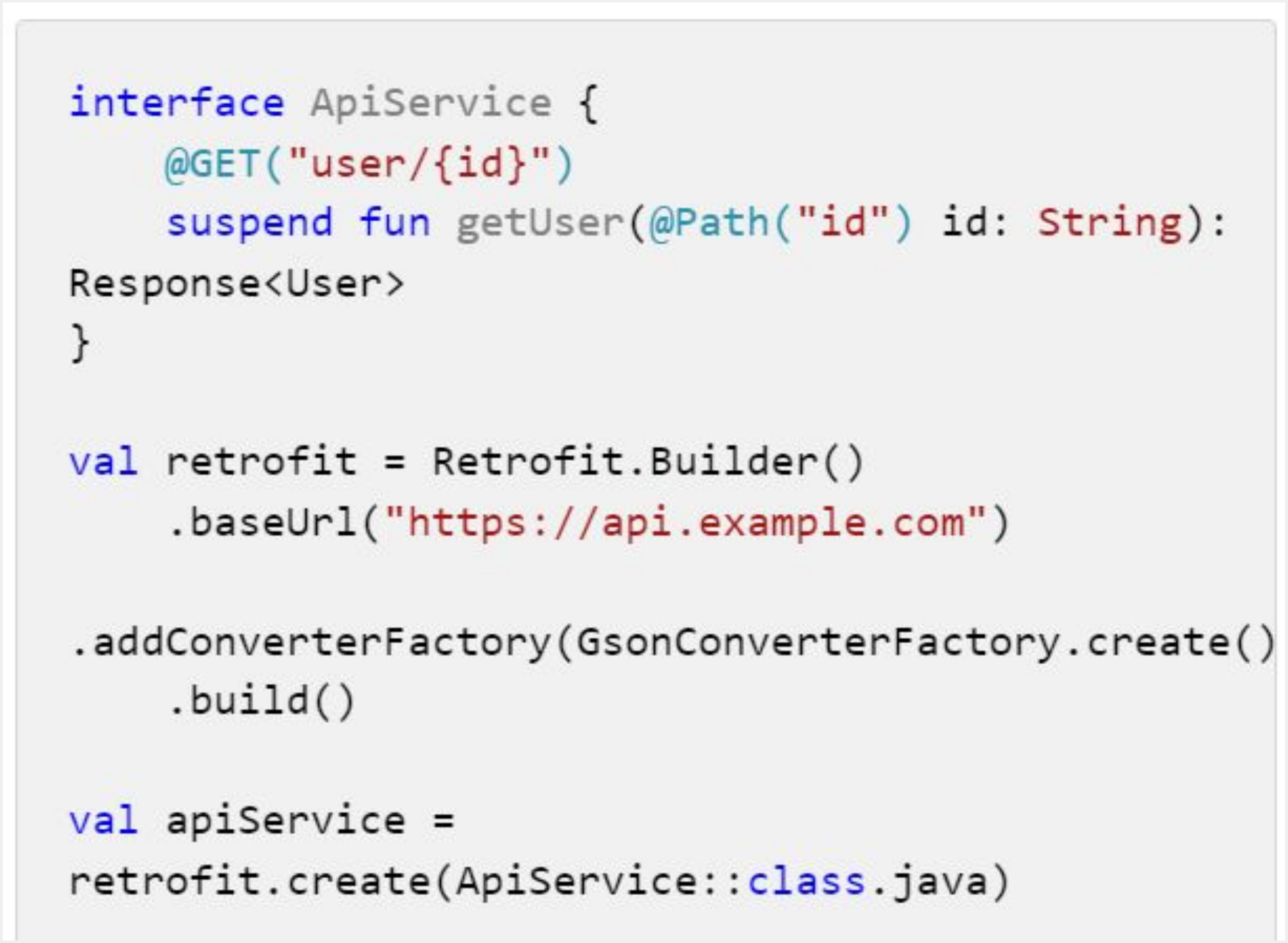
- **HttpURLConnection** is a low-level, native Android class for making network requests. It requires manual handling of connection setup, input/output streams, and error management.
    a. **Pros**: Built into Android, no external libraries required.
    b. **Cons**: Less flexible, requires more boilerplate code for error handling and setup.
- **OkHttp** is a more modern and feature-rich HTTP client, providing easy-to-use APIs for making HTTP requests, handling responses, and dealing with issues like caching, retries, and connection pooling.
    a. **Pros**: More efficient, handles connections automatically, better for managing complex requests (e.g., multipart uploads).
    b. **Cons**: Requires an external dependency (though commonly used in Android).

What is Retrofit, and how does it simplify network calls in Android?

Retrofit is a type-safe HTTP client for Android and Java, built on top of **OkHttp**. It simplifies network requests by:

- **Abstracting HTTP Requests**: Retrofit automatically converts HTTP requests and responses into Java or Kotlin objects using annotations.
- **Integration with JSON parsers**: It works with libraries like **Gson**, **Moshi**, or **Jackson** to handle JSON serialization/deserialization.
- **Easy Integration with Coroutines**: Retrofit can be easily integrated with Kotlin Coroutines to make asynchronous requests clean and simple.

Example of Retrofit setup:



```
interface ApiService {
    @GET("user/{id}")
    suspend fun getUser(@Path("id") id: String):
Response<User>
}

val retrofit = Retrofit.Builder()
    .baseUrl("https://api.example.com")

.addConverterFactory(GsonConverterFactory.create()
    .build()

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

5

What is the difference between synchronous and asynchronous network calls?

- **Synchronous Network Call**: A synchronous request blocks the current thread until the request is completed and a response is received. This can cause the app to freeze or hang if the request takes too long.
    a. **Example**: HttpURLConnection's default behavior is synchronous.
- **Asynchronous Network Call**: An asynchronous request doesn't block the main thread. The request is made in the background, and a callback is invoked when the response is received.
    a. **Example**: Using Retrofit with Kotlin coroutines makes the request asynchronous and non-blocking.

6

How do you handle network failures in Android?

Handling network failures involves:

- **Retry Mechanism**: Libraries like **OkHttp** and **Retrofit** provide built-in retry mechanisms to automatically retry failed requests.
- **Error Handling**: Always handle IOException or TimeoutException to manage issues like no internet connection or request timeout.
- **Connectivity Check**: Before making network requests, use the **ConnectivityManager** to check if the device is connected to the internet.
- **Error UI Feedback**: Show appropriate error messages to the user if the request fails.

Example of error handling with Retrofit:

```
try {
    val response = apiService.getUser("123")
    if (response.isSuccessful) {
        // Handle success
    } else {
        // Handle error response (e.g., 404,
500)
    }
} catch (e: IOException) {
    // Handle network failure (e.g., no
internet connection)
}
```

What is an interceptor in OkHttp, and how is it used?

An **Interceptor** in **OkHttp** is a powerful tool that allows you to intercept and modify requests and responses before they are sent or after they are received.

- **Request Interceptor**: Allows modifying the request before it's sent to the server (e.g., adding headers).
- **Response Interceptor**: Allows modifying the response before it reaches the caller (e.g., logging responses or handling specific status codes).

Example of an interceptor to add an Authorization header:

```kotlin
val interceptor = Interceptor { chain ->
    val newRequest = chain.request().newBuilder()
        .addHeader("Authorization", "Bearer $accessToken")
        .build()
    chain.proceed(newRequest)
}

val client = OkHttpClient.Builder()
    .addInterceptor(interceptor)
    .build()
```

8

How do you secure network calls in Android?

- **Use HTTPS**: Always ensure that API calls are made over HTTPS (encrypted communication), not HTTP.
- **TLS**: Enable TLS (Transport Layer Security) to secure data transfer.
- **SSL Pinning**: Implement SSL pinning to prevent man-in-the-middle (MITM) attacks by ensuring the app only communicates with specific server certificates.
- **OAuth 2.0**: Use **OAuth 2.0** or token-based authentication for securing API endpoints.
- **Network Security Configuration**: Android 9 (Pie) introduced a **Network Security Configuration** feature to configure security settings in the app manifest.

What is Gson, and how does it work with Retrofit for JSON parsing?

- **Gson** is a popular JSON library in Java/Kotlin used to serialize and deserialize objects to and from JSON. In Retrofit, Gson is used as a converter to automatically convert HTTP responses into Java/Kotlin objects and vice versa.
- Example of using Gson with Retrofit:

```
val retrofit = Retrofit.Builder()
    .baseUrl("https://api.example.com/")
    .addConverterFactory(GsonConverterFactory.create())  // Gson integration
    .build()

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

What is the role of the @GET, @POST, and other annotations in Retrofit?

Retrofit uses **annotations** to define the HTTP method, URL, and parameters for network requests:

- @GET: Used to define an HTTP GET request.
- @POST: Used to define an HTTP POST request.
- @PUT, @DELETE, @PATCH: Used for other HTTP methods.
- @Path: Replaces URL parameters.
- @Query: Adds query parameters.
- @Body: Defines the body of the request (for POST, PUT, etc.).

Example:

```
interface ApiService {
    @GET("users/{userId}")
    suspend fun getUser(@Path("userId") userId: String): Response<User>

    @POST("users")
    suspend fun createUser(@Body user: User): Response<User>
}
```

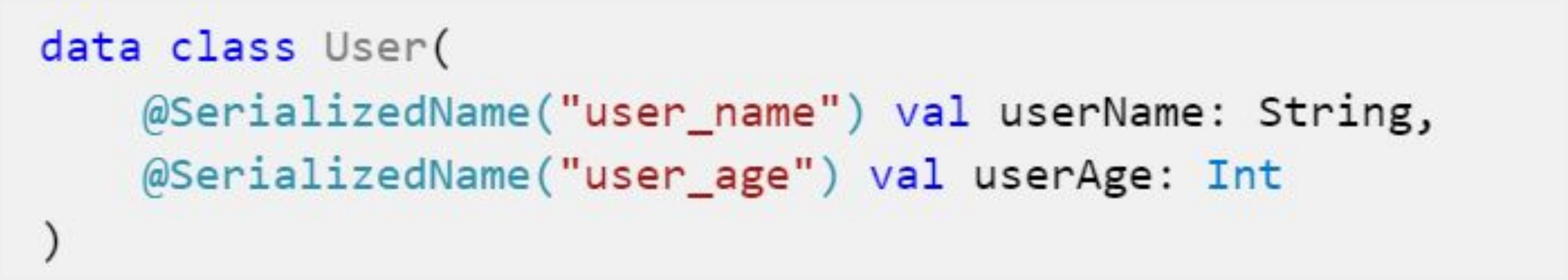
What are the benefits of using Retrofit over raw HttpURLConnection?

- **Simplified API**: Retrofit abstracts away the low-level HTTP handling and makes it easier to deal with network calls.
- **Automatic JSON Parsing**: Retrofit automatically converts JSON responses into Kotlin/Java objects using converters like **Gson**, **Moshi**, or **Jackson**.
- **Coroutines Support**: Retrofit supports Kotlin coroutines for asynchronous calls, making code cleaner and easier to read.
- **Error Handling**: Retrofit provides built-in error handling with custom error responses.
- **Interceptors**: Retrofit supports OkHttp interceptors for logging, adding headers, and modifying requests.

12

What is the role of the @SerializedName annotation in Gson?

- The @SerializedName annotation in **Gson** is used to map the JSON field names to different property names in the Java/Kotlin object. This is helpful when the JSON field names don't match the variable names in the object.



```
data class User(
    @SerializedName("user_name") val userName: String,
    @SerializedName("user_age") val userAge: Int
)
```

What is the importance of the ConnectivityManager in Android?

The **ConnectivityManager** is used to check the device's network connectivity status:

- It provides methods to check whether the device is connected to the internet.
- It can be used to check if the device is connected to Wi-Fi or mobile data.
- Helps in handling network-dependent operations by informing users when no connection is available.

https://www.linkedin.com/in/pavankreddy92/

Follow for More