1. Retrofit (GET, POST, @Query, @Body) – Chi tiết + Ví dụ thực tế

Retrofit là thư viện HTTP mạnh nhất trong Android, dùng để gọi API RESTful. Dựa trên OkHttp.

1.1 Tạo Retrofit Instance

```
val retrofit = Retrofit.Builder()
    .baseUrl("https://jsonplaceholder.typicode.com/")
    .addConverterFactory(GsonConverterFactory.create())
    .build()

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

1.2 @GET – gọi API lấy dữ liệu

```
interface ApiService {

    @GET("posts")
    suspend fun getPosts(): List<Post>
}
```

Model Post:

```
data class Post(
    val userId: Int,
    val id: Int,
    val title: String,
    val body: String
)
```

Gọi API trong ViewModel:

```
val posts = api.getPosts()
```

1.3 @Query – gửi tham số URL

GET với query parameter:

```
https://api.com/users?page=2
```

```
@GET("users")
suspend fun getUsers(@Query("page") page: Int): UserResponse
```

Gọi:

```
api.getUsers(2)
```

1.4 POST + @Body – gửi dữ liệu JSON

```
@POST("posts")
suspend fun createPost(@Body post: Post): Post
```

Gọi:

```
val response = api.createPost(Post(1, 0, "New title", "Hello world"))
```

1.5 @Path – thay đổi endpoint động

```
@GET("posts/{id}")
suspend fun getPost(@Path("id") id: Int): Post
```

1.6 @Header – gắn token

```
@GET("me")
suspend fun getProfile(@Header("Authorization") token: String): User
```

Hoặc set global bằng Interceptor.

---

2. OkHttp Interceptor – Hệ thống "điều khiển request/response"

Retrofit chạy dựa trên **OkHttp**. Interceptor giúp:

- Thêm token vào mọi request
- Ghi log API
- Retry khi lỗi
- Edit request/response

2.1 Logging Interceptor (dùng để debug)

```
val logging = HttpLoggingInterceptor().apply {
    level = HttpLoggingInterceptor.Level.BODY
}

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

2.2 Add Header Interceptor (gắn token tự động)

```
val authInterceptor = Interceptor { chain ->
    val newRequest = chain.request().newBuilder()
        .addHeader("Authorization", "Bearer TOKEN_HERE")
        .build()

    chain.proceed(newRequest)
}

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

2.3 Interceptor để check network

```
val networkInterceptor = Interceptor { chain ->
    val request = chain.request()

    if (!NetworkUtils.isNetworkAvailable()) {
        throw IOException("No Internet")
    }
```

```
    chain.proceed(request)
}
```

3. JSON parsing: Gson & Moshi

Retrofit cần converter để parse JSON → object.

3.1 GsonConverterFactory (Dễ dùng nhất)

```
implementation("com.squareup.retrofit2:converter-gson:2.9.0")
```

Model JSON tự tạo **data class** → Retrofit parse tự động.

JSON → data class

```
{
  "id": 1,
  "name": "Peter"
}
```

```
data class User(val id: Int, val name: String)
```

3.2 Moshi (modern, type-safe hơn)

```
implementation("com.squareup.retrofit2:converter-moshi:2.9.0")
```

Model phải có annotation:

```
@JsonClass(generateAdapter = true)
data class User(
    @Json(name = "id") val id: Int,
    @Json(name = "name") val name: String
)
```

Nói trong phỏng vấn:

> "Gson dễ dùng, phổ biến; Moshi hiện đại hơn, an toàn, hỗ trợ Kotlin tốt, xử lý null mạnh hơn."

---

4. Error handling (HTTP code, try-catch, Response)

Gọi API thất bại là chuyện bình thường. Bạn phải xử lý:

- lỗi mạng
- lỗi server
- lỗi JSON
- lỗi HTTP (401, 404…)
- timeout

4.1 Phân biệt HTTP Code

Code	Loại	Ý nghĩa
200	Success	Thành công
201	Created	Tạo mới thành công
400	Bad Request	Request sai
401	Unauthorized	Thiếu token
403	Forbidden	Không đủ quyền
404	Not found	Không tìm thấy
500	Server error	Server lỗi

4.2 Cách xử lý bằng Response

Retrofit cho phép trả về:

```
suspend fun login(): Response<LoginResponse>
```

→ để check lỗi.

Ví dụ xử lý:

```
val res = api.login()

if (res.isSuccessful) {
    val data = res.body()
} else {
```

```
    val error = res.errorBody()?.string()
}
```

4.3 Try-catch để bắt lỗi mạng

```
try {
    val res = api.getPosts()

} catch (e: HttpException) {
    // HTTP error
} catch (e: IOException) {
    // network error
} catch (e: Exception) {
    // unknown error
}
```

4.4 Sử dụng sealed class cho UI Error State

```
sealed class ApiResult {
    object Loading : ApiResult()
    data class Success<T>(val data: T) : ApiResult()
    data class Error(val msg: String) : ApiResult()
}
```

---

5. Ví dụ hoàn chỉnh: Gọi API + xử lý JSON + error handling

Step 1: Interface API

```
interface ApiService {

    @GET("posts")
    suspend fun getPosts(): Response<List<Post>>
}
```

Step 2: Retrofit Builder

```
val retrofit = Retrofit.Builder()
    .baseUrl("https://jsonplaceholder.typicode.com/")
    .client(
        OkHttpClient.Builder()

.addInterceptor(HttpLoggingInterceptor().setLevel(HttpLoggingInterceptor.Level.BOD
Y))
            .build()
    )
    .addConverterFactory(GsonConverterFactory.create())
    .build()

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

Step 3: ViewModel xử lý API + error

```
class MainVM : ViewModel() {

    val uiState = MutableLiveData<ApiResult>()

    fun loadPosts() {
        viewModelScope.launch {

            uiState.value = ApiResult.Loading

            try {
                val res = api.getPosts()
                if (res.isSuccessful) {
                    uiState.value = ApiResult.Success(res.body()!!)
                } else {
                    uiState.value = ApiResult.Error("HTTP ${res.code()}")
                }

            } catch (e: Exception) {
                uiState.value = ApiResult.Error(e.message ?: "Unknown error")
            }
        }
    }
}
```
