🟢 PHẦN 1 — CÂU HỎI DỄ (bắt buộc phải biết)

---

1. Retrofit là gì?

Đáp án: Thư viện HTTP giúp gọi API REST bằng annotation (@GET, @POST…). Dựa trên OkHttp.

---

2. @GET dùng để làm gì?

Đáp án: Gửi request GET để lấy dữ liệu từ server.

---

3. @POST dùng để làm gì?

Đáp án: Gửi dữ liệu lên server (thường dùng kèm @Body).

---

4. @Query khác @Path thế nào?

Đáp án:

- `@Query`: tham số dạng `?page=2`
- `@Path`: thay thế trực tiếp trong URL (`posts/{id}`)

---

5. @Body dùng khi nào?

Đáp án: Khi cần gửi JSON object trong POST/PUT.

---

6. OkHttp là gì?

Đáp án: Networking engine ở dưới Retrofit, xử lý request/response HTTP.

---

7. Interceptor trong OkHttp để làm gì?

Đáp án: Can thiệp vào request/response: thêm header, log API, check token, retry…

---

8. Gson dùng để làm gì?

Đáp án: Convert JSON ↔ object tự động.

---

9. Moshi khác Gson như thế nào?

Đáp án: Moshi modern hơn, hỗ trợ Kotlin tốt, parse an toàn hơn.

---

10. HTTP 200/400/500 nghĩa là gì?

Đáp án:

- **200**: success
- **400**: request sai
- **401**: không có token
- **404**: không tìm thấy
- **500**: server lỗi

🟠 PHẦN 2 — CÂU HỎI TRUNG BÌNH (Fresher → Junior)

---

11. Response là gì?

Đáp án: Wrapper chứa: status code, body, errorBody → giúp xử lý lỗi HTTP.

---

12. Khi nào nên dùng Response thay vì trả trực tiếp T?

Đáp án: Khi cần xử lý success/error rõ ràng.

---

13. Sự khác nhau giữa execute() và enqueue() trong Retrofit?

Đáp án:

- execute(): chạy **sync**, block thread
- enqueue(): chạy **async**, KHÔNG block → dùng trong Android

---

14. Interceptor khác NetworkInterceptor như thế nào?

Đáp án:

- Interceptor: chạy **trước** network call
- NetworkInterceptor: chạy **sau**, có access header thực sự từ server

---

15. LoggingInterceptor dùng để làm gì?

Đáp án: In log request, response, header, body → debug API.

---

16. Làm sao gắn token Authorization vào mọi request?

Đáp án: Tạo OkHttp Interceptor:

```
.addInterceptor { chain -> ... }
```

---

17. Nếu server trả JSON field thiếu thì Gson xử lý thế nào?

Đáp án: Gson bỏ qua field không có và set default value (0, null,...).

---

18. Serialization và deserialization là gì?

Đáp án:

- Serialization: object → JSON
- Deserialization: JSON → object

---

19. Timeout trong OkHttp là gì?

Đáp án: Thời gian tối đa khi chờ kết nối/đọc/ghi:

```
connectTimeout
readTimeout
writeTimeout
```

---

20. Error handling trong Retrofit thường xử lý ở đâu?

Đáp án: Trong ViewModel, dùng try-catch + Response.isSuccessful.

---

21. Thế nào là network error và HTTP error?

Đáp án:

- HTTP error: server trả mã lỗi (400, 404, 500)
- Network error: mất mạng, timeout → IOException

---

22. Cách tốt nhất để wrap API result?

Đáp án: Dùng **sealed class**:

```
Success, Loading, Error
```

---

23. Retrofit dùng thread nào?

Đáp án: Tự động chạy trên thread background khi dùng suspend.

---

24. Khi nào cần ConverterFactory?

Đáp án: Để Retrofit parse JSON (Gson, Moshi…).

---

25. Retrofit có tự động retry request không?

Đáp án: Không, phải tự thêm retry Interceptor.

🟥 PHẦN 3 — CÂU HỎI KHÓ (Junior – Mid Android)

---

26. Bạn giải thích quá trình từ lúc Retrofit gửi request tới khi nhận response?

Đáp án (tóm tắt chuẩn):

1. Retrofit build request
2. OkHttp tạo Call object
3. Interceptor chạy (add header, log…)
4. Network call → server
5. Server trả response
6. ConverterFactory parse JSON
7. Trả về object tới ViewModel

---

27. Làm sao làm API chaining với Retrofit (call A xong mới call B)?

Đáp án: Dùng coroutine:

```
val a = api.getUser()
val b = api.getPosts(a.id)
```

---

28. Làm API song song như thế nào?

Đáp án: Dùng async:

```
val user = async { api.getUser() }
val posts = async { api.getPosts() }
```

---

29. Khác nhau giữa suspend GET và normal GET trong Retrofit?

Đáp án:

- suspend GET chạy async trong coroutine
- auto background thread
- không cần enqueue()

---

30. Tại sao ResponseBody chỉ đọc 1 lần?

Đáp án: Vì stream input chỉ đọc được 1 lần → phải clone hoặc convert sớm.

---

31. Tại sao Moshi an toàn hơn Gson?

Đáp án:

- strict type checking
- tránh silent error
- hỗ trợ non-null tốt hơn
- Kotlin adapter mạnh hơn

---

32. Tại sao convert JSON bằng reflection (Gson) chậm hơn code-gen (Moshi)?

Đáp án: Reflection nặng → Moshi dùng code-generated adapter tối ưu.

---

33. Server trả JSON field unknown → Retrofit xử lý sao?

Đáp án: Gson bỏ qua field unknown. Moshi strict hơn và có thể ném exception.

---

34. Làm sao debug body request khi POST?

Đáp án: Bật LoggingInterceptor.level = BODY.

---

35. Khi API trả errorBody dạng JSON, làm sao parse nó?

Đáp án:

```
val error = Gson().fromJson(res.errorBody()?.string(), ErrorResponse::class.java)
```

---

36. Làm sao handle Refresh token trong Interceptor?

Đáp án (trả lời chuẩn interview):

- Interceptor check 401

- Call API refresh token
- Retry request với token mới
- Save token trong SharedPreferences/DataStore

---

37. Có thể inject Retrofit vào ViewModel không?

Đáp án: Không trực tiếp. Nên inject vào Repository → gọi trong ViewModel.

---

38. Khi server trả về danh sách lớn, làm sao tối ưu decode JSON?

Đáp án:

- Dùng Moshi code-gen
- Streaming JSON parser
- Pagination

---

39. Tại sao Retrofit + Coroutines an toàn hơn Retrofit + Callback?

Đáp án: Không bị memory leak vì coroutine gắn với scope lifecycle.

---

40. Bạn giải thích circuit breaker / retry logic trong networking?

Đáp án: Cơ chế tránh spam request khi server lỗi, retry có delay exponential.
