LAPORAN PRAKTIKUM MODUL 3

A. Analisis Error Handling & Logging

a. Penanganan Error

Pada library **HTTP**, error ditangani secara manual dengan blok try-catch, sehingga pengembang perlu menulis logika pengecualian sendiri setiap kali terjadi kesalahan.

```
 try {
    final response = await http.get(Uri.parse(baseUrl));

    if (response.statusCode == 200) {
      final List<dynamic> data = jsonDecode(response.body);
      print('[HTTP] Success in ${stopwatch.elapsedMilliseconds} ms');
      return data;
    } else {
      throw Exception('HTTP Error: ${response.statusCode}');
    }
   } catch (e) {
    print('[HTTP] Error: $e');
    rethrow;
   } finally {
    stopwatch.stop();
   }
 }
}
```

pada **Dio**, error ditangani secara otomatis dengan mekanisme Interceptor, yang memudahkan pengelolaan error global seperti timeout, koneksi gagal, dan status kode HTTP yang tidak valid.

```
try {
    final response = await dio.get(baseUrl);
    final data = response.data is String
        ? jsonDecode(response.data)
        : response.data;

    print('[DIO] Success in ${stopwatch.elapsedMilliseconds} ms');
    return data;
   } on DioException catch (e) {
    print('[DIO] Dio Error: ${e.message}');
    throw Exception('Dio Error: ${e.message}');
   } catch (e) {
    print('[DIO] Unknown Error: $e');
    throw Exception('Unknown Error: $e');
   } finally {
    stopwatch.stop();
   }
```

```
  }
}
```

b. Response time

No	1	2	3	4	5
HTTP	21	25	15	19	19
DIO	44	67	59	27	61

c. Analisis

Dio memiliki sistem logging dan penanganan error yang lebih terstruktur, terutama saat debugging proses request-response. HTTP lebih ringan dan sederhana, namun membutuhkan penanganan manual untuk log dan error.

B. Eksperimen Async Handling (Chained Request)

alur asynchronous diimplementasikan untuk memanggil data parfum berdasarkan kategori, di mana data utama diambil terlebih dahulu, kemudian disaring (filtered) sesuai kategori.

```
Future<void> loadPerfumesByCategory(String category) async {
  try {
    // ambil data utama
    final data = await dioService.fetchPerfumes();
    perfumes.assignAll(List<Map<String, dynamic>>.from(data));

    // chained request - filter berdasarkan kategori
    selectedCategory.value = category;
    perfumes.assignAll(
      perfumes.where((item) =>
          item["category"].toString().toLowerCase() ==
          category.toLowerCase()).toList(),
    );
  } catch (e) {
    Get.snackbar("Error", "Load error: $e");
  }
}
```

Analisis

- Async-Await mempermudah urutan eksekusi kode yang bergantung satu sama lain.
- Callback chaining juga bisa digunakan, tetapi async-await jauh lebih terbaca dan mudah di-debug.
- Struktur linear ini penting agar aplikasi tetap responsif dan tidak menggantung saat proses API berjalan.
- Struktur callback chaining terlihat lebih rumit dan berpotensi menimbulkan bug karena nesting function yang sulit dilacak. Sebaliknya, async–await lebih linear dan mudah dibaca.

C. Diskusi & Refleksi

Setelah dibandingkan, hasil eksperimen menunjukkan bahwa:

- Dio lebih cocok digunakan untuk proyek yang besar dan kompleks karena fitur logging, interceptor, serta manajemen error yang lebih rapi.
- HTTP cocok untuk proyek kecil yang membutuhkan performa cepat tanpa konfigurasi berat.
- Kombinasi GetX dengan dua library ini membuat manajemen data menjadi efisien dan mudah diatur.
- Pada sisi tampilan, data dapat dimuat berdasarkan kategori “All”, “Perfume”, dan “Ingredients”, serta ditambahkan secara dinamis.

Pendekatan async–await dengan library **Dio** direkomendasikan sebagai solusi terbaik karena:

- Meningkatkan keterbacaan dan pemeliharaan kode.
- Mengurangi potensi bug akibat nested callback.
- Memberikan performa tinggi dan fleksibilitas dalam mengelola request API yang kompleks.

Dengan demikian, kombinasi **Dio + Async–Await** layak dijadikan standar untuk pengembangan aplikasi berbasis API modern yang membutuhkan stabilitas dan efisiensi jangka panjang.