Nama : Akmal Muhamad Firdaus

NIM : 1301204188

Jurnal Praktikum Modul 12 dan 13

1. Model

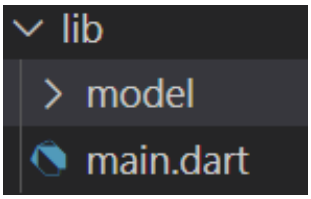
1.1. Pengenalan Model

Model mengacu pada bagian dari arsitektur aplikasi yang bertanggung jawab untuk mengelola data dan logika bisnis aplikasi. Model dalam Flutter biasanya digunakan untuk mengatur, memperoleh, dan memanipulasi data, serta menyediakan metode untuk berkomunikasi dengan lapisan lain dalam aplikasi, seperti tampilan (view) dan pengontrol (controller).

Model sendiri biasanya bagian yang bersentuhan langsung dengan database dan mengkonversinya menjadi class dart yang dapat diakses lebih mudah. Umumnya model akan dibuat dari response JSON.

1.2. Membuat Model Class

Untuk membuat model biasanya dimasukan kedalam direktori khusus, biasanya bernama model dalam direktori lib.



Setelah itu buat file dart baru pada direktori model. Sebagai contoh response json berikut akan dibuatkan model pada flutter.

```
{
  “user_id” : 1,
  “id” : 13,
  “title” : “Bedroom Pop”
}
```

Maka bentuk dari model flutternya adalah sebagai berikut.

```dart
class Album {
  final int userId;
  final int id;
  final String title;
  const Album({
    required this.userId,
    required this.id,
    required this.title,
  });
  factory Album.fromJson(Map<String, dynamic> json) {
    return Album(
      userId: json['user_id'],
      id: json['id'],
      title: json['title'],
    );
  }
}
```

2. Navigation

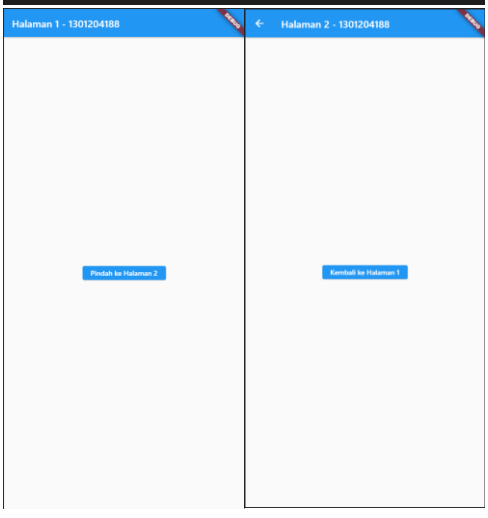
2.1. Navigation Pindah Halaman

Navigation Pindah Halaman adalah proses berpindah dari satu halaman ke halaman lain dalam aplikasi. Ini melibatkan perpindahan antara tampilan (widget) yang berbeda untuk menyajikan konten atau fitur yang berbeda kepada pengguna.

Flutter menyediakan berbagai cara untuk melakukan navigasi antar halaman, termasuk penggunaan widget Navigator, MaterialPageRoute, PageRouteBuilder, dan banyak lagi. Berikut adalah contoh menggunakan Navigator:

```dart
import 'package:flutter/material.dart';
import 'Halaman2.dart';

Run | Debug | Profile
void main() {
  runApp(MaterialApp(home: Halaman1()));
}

class Halaman1 extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Halaman 1 - 1301204188'),
      ), // AppBar
      body: Center(
        child: ElevatedButton(
          child: Text('Pindah ke Halaman 2'),
          onPressed: () {
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => Halaman2()),
            );
          },
        ), // ElevatedButton
      ), // Center
    ); // Scaffold
  }
}
```

```dart
import 'package:flutter/material.dart';

class Halaman2 extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Halaman 2 - 1301204188'),
      ), // AppBar
      body: Center(
        child: ElevatedButton(
          child: Text('Kembali ke Halaman 1'),
          onPressed: () {
            Navigator.pop(context);
          },
        ), // ElevatedButton
      ), // Center
    ); // Scaffold
  }
}
```

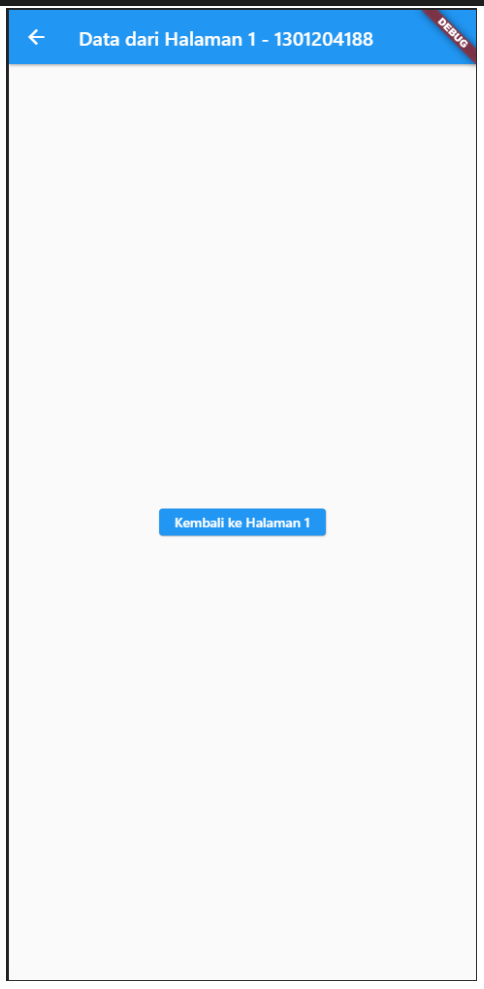


2.2. Navigation Mengirim Data

Untuk mengirim data antara halaman dalam aplikasi Flutter, kita dapat menggunakan parameter pada metode navigasi. Berikut adalah cara umum untuk mengirim data antara halaman:

```dart
Navigator.push(
  context,
  MaterialPageRoute(
      builder: (context) =>
          const Halaman2(title: "Data dari Halaman 1")),
);
```

```dart
import 'package:flutter/material.dart';

class Halaman2 extends StatelessWidget {
  const Halaman2({Key? key, required this.title}) : super(key: key);
  final String title;
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('${title} - 1301204188'),
      ), // AppBar
      body: Center(
        child: ElevatedButton(
          child: Text('Kembali ke Halaman 1'),
          onPressed: () {
            Navigator.pop(context);
          },
        ), // ElevatedButton
      ), // Center
    ); // Scaffold
  }
}
```



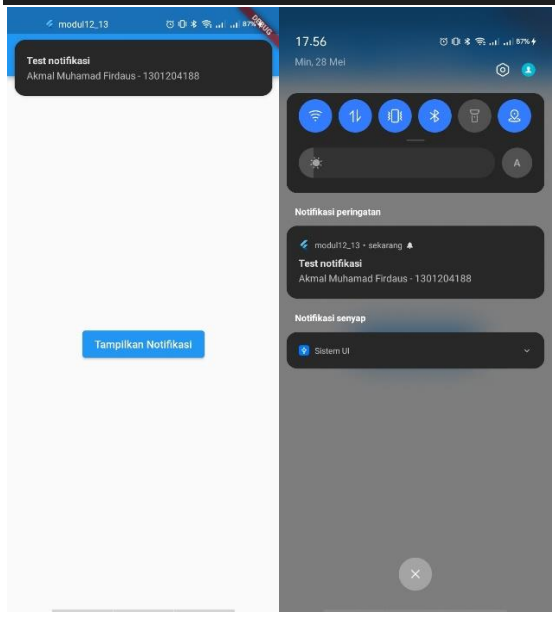
3. Notification

Notification pada Flutter adalah pesan yang ditampilkan kepada pengguna sebagai pemberitahuan atau informasi penting di luar jendela utama aplikasi. Notifikasi dapat muncul di bilah notifikasi atau panel notifikasi perangkat, memberikan pengguna informasi penting atau mengingatkan mereka tentang suatu kejadian atau tugas tertentu.

Berikut adalah contoh menggunakan notification pada flutter:

```dart
import 'package:flutter/material.dart';
import 'package:flutter_local_notifications/flutter_local_notifications.dart';

final FlutterLocalNotificationsPlugin flutterLocalNotificationsPlugin =
    FlutterLocalNotificationsPlugin();
Run | Debug | Profile
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await flutterLocalNotificationsPlugin.initialize(
    InitializationSettings(
      android: const AndroidInitializationSettings('@mipmap/ic_launcher'),
    ), // InitializationSettings
  );
  runApp(MaterialApp(home: Halaman1()));
}
```

```dart
Future<void> showNotification() async {
  var s = 'your_channel_description';
  var androidPlatformChannelSpecifics = AndroidNotificationDetails(
    'ID_Channel_Test_Notifikasi',
    'Nama_Channel_Test_Notifikasi',
    importance: Importance.max,
    priority: Priority.high,
  );

  var platformChannelSpecifics =
      NotificationDetails(android: androidPlatformChannelSpecifics);

  await flutterLocalNotificationsPlugin.show(
    0,
    'Test notifikasi',
    'Akmal Muhamad Firdaus - 1301204188',
    platformChannelSpecifics,
  );
}

class Halaman1 extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Halaman 1 - 1301204188'),
      ), // AppBar
      body: Center(
        child: ElevatedButton(
          child: Text('Tampilkan Notifikasi'),
          onPressed: () {
            showNotification();
          },
        ), // ElevatedButton
      ), // Center
    ); // Scaffold
  }
}
```

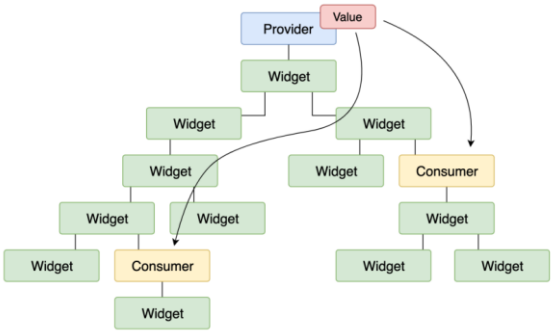


4. State Management

4.1. Pengenalan State Management

State Management adalah cara mengelola dan memperbarui state (data) dalam aplikasi Flutter. State management diperlukan ketika aplikasi memiliki state yang kompleks, seperti data yang perlu diperbarui, disimpan, dan dibagikan di antara berbagai komponen atau widget dalam aplikasi.

4.2. State Management Provider



State Management Provider adalah library Flutter yang digunakan untuk manajemen state secara efisien dan sederhana. Library ini memungkinkan kita untuk membagikan dan mengakses state aplikasi secara global di seluruh widget-widget dalam hierarki widget Flutter tanpa harus melewati banyak tingkatan widget.

Library ini menyediakan provider dan consumer untuk menghubungkan state dengan widget yang membutuhkannya. Berikut adalah langkah-langkah dasar penggunaan Provider dalam manajemen state:

1. Tambahkan dependensi provider ke file pubspec.yaml pada proyek Flutter

```
dependencies:
  flutter:
    sdk: flutter
  flutter_local_notifications: ^14.1.0
  provider: ^6.0.5
```

2. Jalankan perintah *flutter pub get* untuk mengunduh dan memperbarui dependensi
3. Buat sebuah class yang akan menjadi state atau model. Misalnya, jika Anda ingin mengelola state counter, Anda dapat membuat class Counter seperti ini:

```
class Counter {
  int value = 0;

  void increment() {
    value++;
  }
}
```

4. Buat class *ChangeNotifier* yang meng-extend class Counter dan akan mengubah state dan memberi tahu widget-widget yang bergantung pada perubahan state:

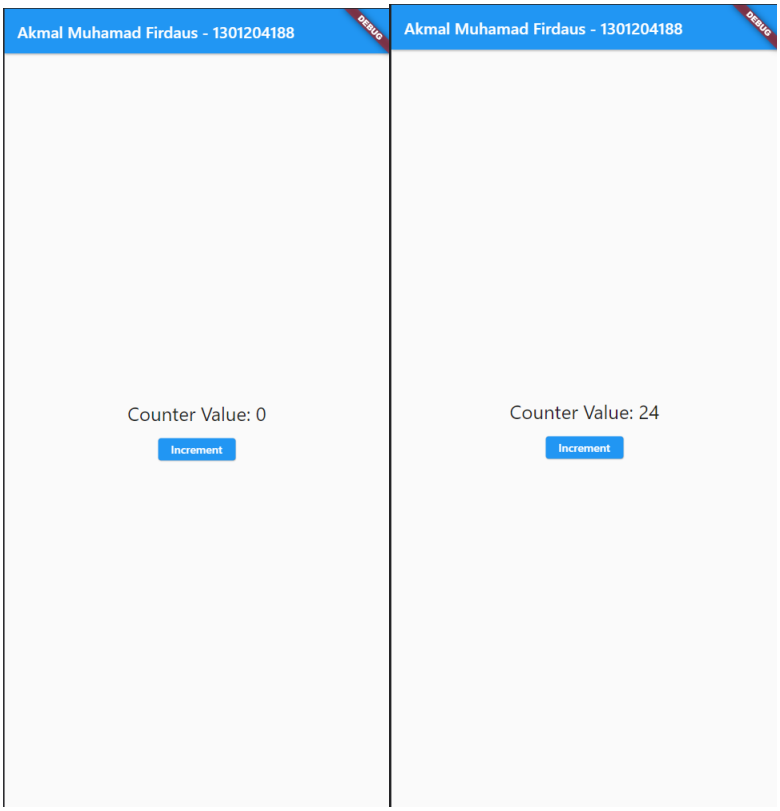
```
class CounterNotifier extends ChangeNotifier {
  Counter counter = Counter();

  void incrementCounter() {
    counter.increment();
    notifyListeners();
  }
}
```

5. Di dalam widget utama aplikasi, wrap widget tersebut dengan *ChangeNotifierProvider*:

```
void main() {
  runApp(
    ChangeNotifierProvider(
      create: (context) => CounterNotifier(),
      child: MyApp(),
    ), // ChangeNotifierProvider
  );
}
```

6. Sekarang, dalam widget mana pun di dalam aplikasi, kita dapat menggunakan Consumer untuk mengakses dan memperbarui state:

```
class CounterWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final counterNotifier = Provider.of<CounterNotifier>(context);

    return Scaffold(
      appBar: AppBar(
        title: Text('Akmal Muhamad Firdaus - 1301204188'),
      ), // AppBar
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(
              'Counter Value: ${counterNotifier.counter.value}',
              style: TextStyle(fontSize: 24),
            ), // Text
            SizedBox(height: 16),
            ElevatedButton(
              onPressed: () {
                counterNotifier.incrementCounter();
              },
              child: Text('Increment'),
            ), // ElevatedButton
          ],
        ), // Column
      ), // Center
    ); // Scaffold
  }
}
```


Akmal Muhamad Firdaus - 1301204188
DEBUG
Counter Value: 0
Increment
Akmal Muhamad Firdaus - 1301204188
DEBUG
Counter Value: 24
Increment

5. Networking

Networking adalah proses berkomunikasi antara aplikasi Flutter dengan server atau layanan luar menggunakan protokol HTTP. Dalam pengembangan aplikasi Flutter, terdapat beberapa paket atau library yang dapat digunakan untuk melakukan operasi jaringan, seperti pengambilan data dari API, mengirim permintaan ke server, atau mengunggah file.

Untuk dapat menggunakan library http pada proyek flutter, lakukan langkah-langkah berikut ini:

1. Tambahkan dependensi http ke file pubspec.yaml pada proyek Flutter:

```yaml
dependencies:
  flutter:
    sdk: flutter
  flutter_local_notifications: ^14.1.0
  provider: ^6.0.5
  http: ^1.0.0
```

2. Jalankan perintah *flutter pub get* untuk mengunduh dan memperbarui dependensi.
3. Import paket http ke dalam file Dart tempat Anda akan menggunakan operasi jaringan:

5.1. Fetch & Delete Data

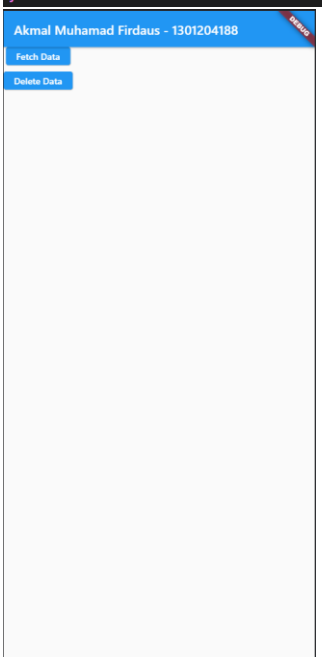
Fetch adalah Pengambilan data (fetching) dan Delete adalah penghapusan data (deleting). Fetch & Delete Data dapat merujuk pada proses mengambil (fetch) data dari server menggunakan permintaan HTTP (misalnya GET request) dan menghapus (delete) data dari server menggunakan permintaan HTTP (misalnya DELETE request).

Berikut adalah contoh Fetch dan Delete data

```
Performing hot restart...                                        179ms
Restarted application in 179ms.
Behrasil request: 1, quidem molestiae enim, 1
Berhasil delete
```

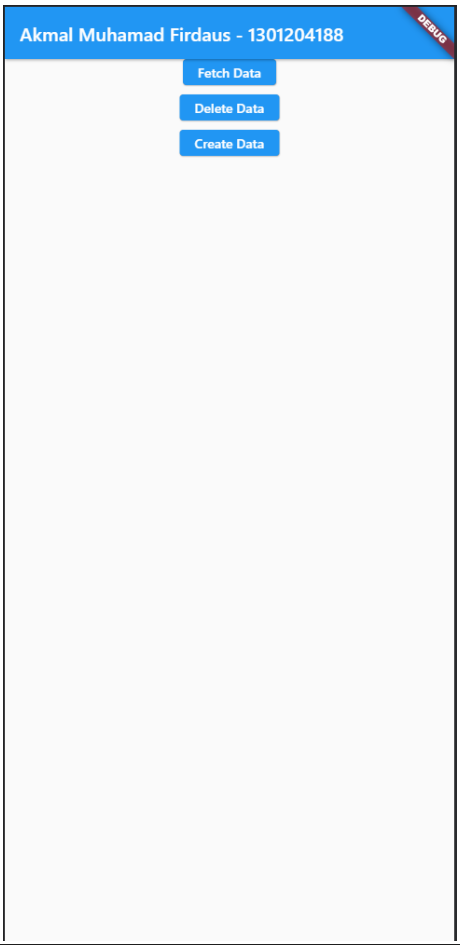
```dart
Future<Album> fetchAlbum() async {
  final response = await http
      .get(Uri.parse('https://jsonplaceholder.typicode.com/albums/1'));
  if (response.statusCode == 200) {
    Album data = Album.fromJson(jsonDecode(response.body));
    print('Behrasil request: ${data.id}, ${data.title}, ${data.userId}');
    return Album.fromJson(jsonDecode(response.body));
  } else {
    throw Exception('Failed to load album');
  }
}

Future<http.Response> deleteAlbum(String id) async {
  final http.Response response = await http.delete(
    Uri.parse('https://jsonplaceholder.typicode.com/albums/$id'),
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
  );

  if (response.statusCode == 200) {
    print('Berhasil delete');
    return response;
  } else {
    throw Exception('Failed to load album');
  }
}
```

```dart
@override
Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        title: Text('Akmal Muhamad Firdaus - 1301204188'),
      ), // AppBar
      body: Column(
        children: [
          ElevatedButton(
            onPressed: fetchAlbum,
            child: Text('Fetch Data'),
          ), // ElevatedButton
          SizedBox(height: 10),
          ElevatedButton(
            onPressed: () {
              deleteAlbum("3");
            },
            child: const Text('Delete Data'),
          ) // ElevatedButton
        ],
      ), // Column
    ), // Scaffold
  ); // MaterialApp
}
```



5.2. Send & Update data

Untuk melakukan send data dapat menggunakan post sedangkan untuk menupdate data dapat menggunakan method update put. Berikut adalah contohnya:




```
Future<http.Response> createAlbum(String title) {
  return http.post(
    Uri.parse('https://jsonplaceholder.typicode.com/albums'),
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
    body: jsonEncode(<String, String>{
      'title': title,
    }),
  );
}
```

```
Performing hot restart...                                    104ms
Restarted application in 104ms.
Data from create album {
  "title": "Akmal Muhamad Firdaus",
  "id": 101
}
```

5.3. Parse JSON

Parse json sudah dilakukan sebelumnya pada poin 5.1 saat proses pengambilan data, lalu memparsing json tersebut ke model Album.

```dart
import 'dart:convert';
Future<Album> fetchAlbum() async {
  final response = await http
      .get(Uri.parse('https://jsonplaceholder.typicode.com/albums/1'));
  if (response.statusCode == 200) {
    Album data = Album.fromJson(jsonDecode(response.body));
    print('Behrasil request: ${data.id}, ${data.title}, ${data.userId}');
    return Album.fromJson(jsonDecode(response.body));
  } else {
    throw Exception('Failed to load album');
  }
}
```