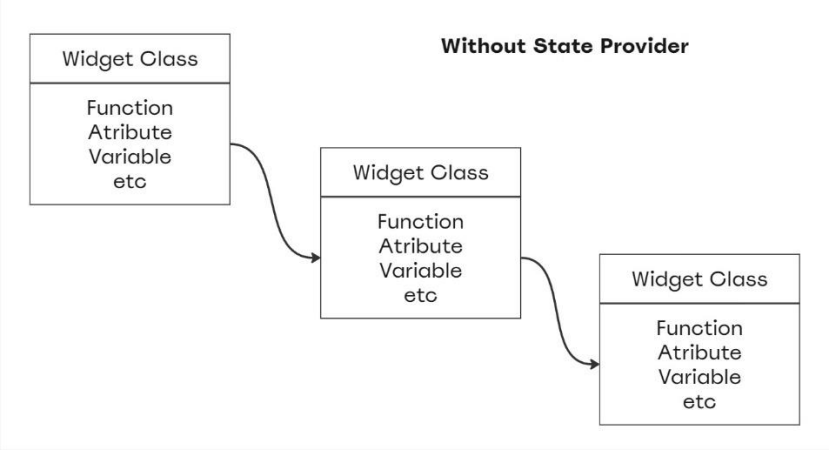
PRAKTIKUM MCS BAB 4

State Management Provider

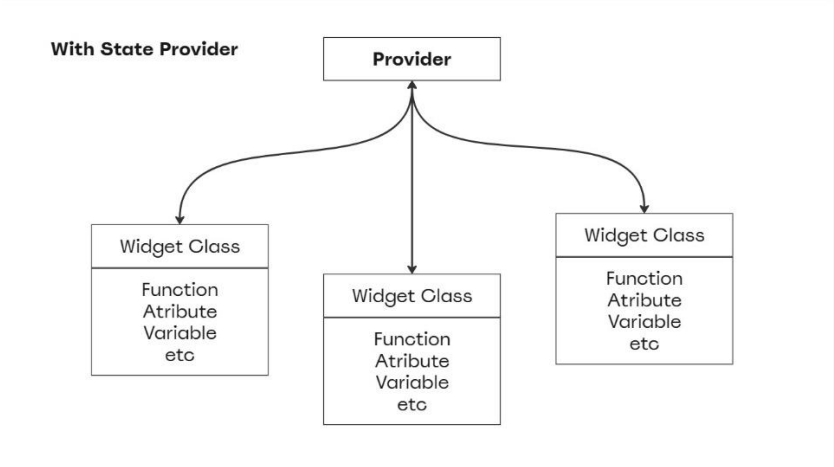
PENDAHULUAN

Pada praktikum MCS bab 4 akan belajar State Management Provider. Dalam flutter segala bentuk kondisi, aksi, perpindahan/perubahan data ataupun segala sesuatu yang sekiranya akan mempengaruhi tampilan di sebut State. Sehingga State Management adalah cara dari aplikasi untuk mengatur, mengelola dan menangani state secara efisien terutama saat data atau state tersebut melibatkan beberapa widget.

Flutter menyediakan banyak package untuk State Management, diantaranya adalah setstate, provider, bloc, getx ataupun riverpod. Provider merupakan salah satu package State Management yang menawarkan efisiensi, kesederhanaan dan fleksibilitas dalam hal mengelola state yang melibatkan beberapa widget tanpa harus melewati widget secara manual. Berikut perbedaannya



Tanpa provider proses flow atau alur state panjang karena bussines logic tergabung dengan UI. Penggabungan tersebut akan membuat state harus diturunkan secara manual menggunakan constructor atau widget parameter dari widget induk ke widget anaknya (metode ini disebut prop drilling). Semakin kompleks dan dalam susunan hirarki widget pada suatu aplikasi maka semakin banyak level yang harus di lewati state tersebut meskipun diantaranya tidak membutuhkan state tersebut. Hal akan membuat code menjadi rumit dan sulit untuk di maintenance. Flow di dalamnya pun menjadi tidak efisien

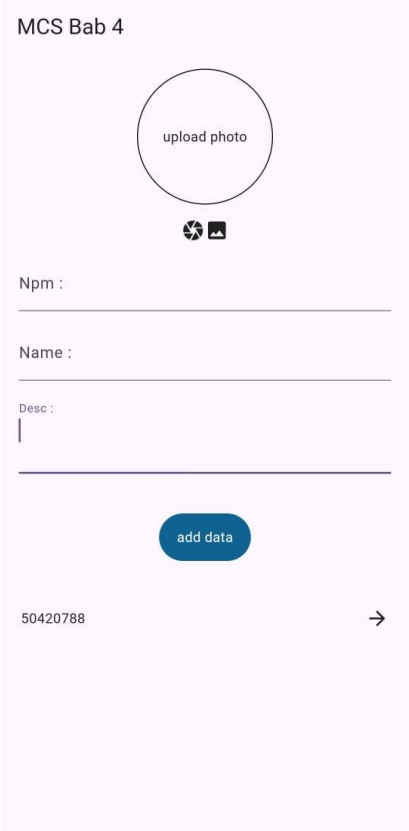


Sedangkan saat menggunakan provider, bussines logic akan dibuat terpisah dari UI. State akan dikelola secara terpusat sehingga dapat diakses oleh widget manapun tanpa harus melewati widget induk terlebih dahulu. Provider memungkinkan widget untuk mendengar (listen) state yang ada tanpa harus meneruskan data melalui constructor atau parameter.

Aplikasi yang akan dibuat pada bab 4 ini adalah aplikasi input data sederhana yang akan menggunakan provider untuk mengelola statenya. Meskipun dapat menginput data, aplikasi ini tidak berfokus pada bagaimana data disimpan seperti pada bab sebelumnya. Sehingga data hanya disimpan pada sebuah List. Fokus dari aplikasi ini adalah bagaimana cara menerapkan dan menggunakan provider untuk mengelola state dengan memisahkan bussines logic dengan UI.

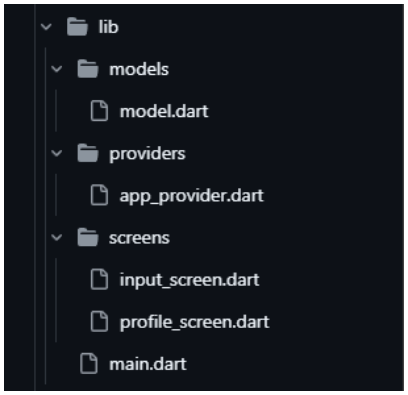
PRAKTIKUM BAB 4

Tampilan aplikasi yang akan dibangun



Penjelasan cara kerja aplikasi akan diterangkan oleh Penanggung Jawab (PJ)

Buatlah *project flutter* baru pada android studio dan pilihlah tempat untuk menyimpan *project* tersebut. Setelah *project* selesai terbentuk, buatlah beberapa file **.dart** di dalam folder ***lib*** dan tambahkan beberapa *package* berikut ke dalam file **pubspec.yaml**




```
dependencies:
  flutter:
    sdk: flutter


  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.8
  provider: ^6.1.2
  image_picker: ^1.1.2
```

Buat model sebagai kerangka dari List yang akan menghandle data yang akan di input dengan menambahkan barisan code pada **file model.dart**

```
import 'dart:io' as io;

class Model{
  io.File? image;
  int? npm;
  String? name;
  String? desc;

  Model({
    required this.image,
    required this.npm,
    required this.name,
    required this.desc,
  });
}
```

Class ini digunakan untuk merepresentasikan data yang nanti akan digunakan atau diinputkan oleh user. Class model ini terdiri dari properti yang berisi nama dari variabel beserta tipe datanya seperti variabel image yang berupa io file (input/output file), npm yang berupa int (angka), serta name dan desc yang merupakan string. Properti ini seperti melakukan inisialisasi variabel untuk data yang akan digunakan nanti sehingga dapat lebih terstruktur dan mudah untuk dimanupulasi

Selain properti, terdapat *constructor* yang menginisialisasi properti ketika ada objek baru dibuat. Ketika sistem akan membuat object baru (dalam contoh kasus ini user menginputkan data baru) *constructor* ini akan dipanggil dan memastikan semua data terisi karena menggunakan kata kunci *required*.

Kemudian buka **file app_provider.dart** dan tambahkan kode berikut

```
import 'package:flutter/material.dart';
import 'package:image_picker/image_picker.dart';
import 'dart:io' as io;
import 'package:mcs_bab_4/models/model.dart';
import 'package:mcs_bab_4/screens/profile_screen.dart';

class AppProvider extends ChangeNotifier{
  List<Model> data = [];
  io.File? image;
  ImagePicker imagePicker = ImagePicker();
  TextEditingController inputNpmController = TextEditingController();
  TextEditingController inputNameController = TextEditingController();
  TextEditingController inputDescController = TextEditingController();
  String npmLabel = 'Npm :';
  String nameLabel = 'Name :';
  String descLabel = 'Desc :';
  int? index;
  io.File? imageProfile;
  int? npm;
```

```
  String? name;
  String? desc;

  @override
  void dispose(){
    super.dispose();
    inputNpmController.dispose();
    inputNameController.dispose();
    inputDescController.dispose();
  }

  imageFromCamera() async{
    XFile? pickedFile = await imagePicker.pickImage(source:
ImageSource.camera);
    io.File image = io.File(pickedFile!.path);
    this.image = image;
    notifyListeners();
  }

  imageFromGallery() async{
    XFile? pickedFile = await imagePicker.pickImage(source:
ImageSource.gallery);
    io.File image = io.File(pickedFile!.path);
    this.image = image;
    notifyListeners();
  }

  addData() async{
    data.add(
        Model(
          image: image,
          npm: int.parse(inputNpmController.text),
          name: inputNameController.text,
          desc: inputDescController.text,
        ));
    image = null;
    inputNpmController.text = "";
    inputNameController.text = "";
    inputDescController.text = "";
    notifyListeners();
  }

  deleteData({
    required BuildContext context,
  }) async{
    await data.removeAt(index!);
    Navigator.pop(context);
    notifyListeners();
  }

  goToProfileScreen({
    required BuildContext context,
    required int index,
    required Model model,
  }) async{
    this.index = index;
    imageProfile = await model.image;
    npm = await model.npm;
    name = await model.name;
    desc = await model.desc;
    Navigator.push(
```

```
      context,
      MaterialPageRoute(builder: (context) => const ProfileScreen(),),
    );
    notifyListeners();
  }
}
```

Secara garis besar, file ini berisi class dengan nama AppProvider yang berfungsi untuk mengelola *bussines logic* termasuk *state* menggunakan *ChangeNotifier* yang nantinya akan memberi notifikasi widget lain kalau terjadi perubahan state yang nantinya akan mempengaruhi tampilan UI. Class ini juga akan menangani controller dan fungsi dari masing masing widget utama pada aplikasi ini. Blok awal kode merupakan inisialisasi properti seperti inisialisasi List sebagai wadah penyimpanan data inputan user, nama variabel sampai controller widget.

Blok selanjutnya yaitu method dispose() yang akan menjalankan fungsi pembersihan (dispose) controller dari textfield saat sudah tidak digunakan untuk mencegah kebocoran memori.

Penjelasan tiap fungsi/method yang ada:

- Method imageFromCamera() yang bersifat asynchronous. Method ini menangani pengambilan gambar dari kamera. Setelah gambar diambil, gambar tersebut akan disimpan kedalam properti (variabel) image dan memanggil notifyListeners() untuk memberitau widget (disebut: consumer) yang menggunakan method ini bahwa telah terjadi perubahan state.
- Method imageFromGallery() memiliki fungsi dan cara kerja yang sama persis seperti method imageFromCamera(), pembedanya adalah method ini memungkinkan pengguna untuk memilih gambar dari aplikasi galeri di perangkat.
- Method addData() bersifat asynchronous. Method ini berfungsi untuk menangani proses tambah data ke dalam list sesuai dengan struktur list yang sudah di inisialisasi pada model dengan mengambil inputan dari user. Setelah list dibuat, input field akan di reset lagi menjadi kosong.
- Method deleteData() bersifat asynchronous. Method ini berfungsi untuk menangani proes hapus data list berdasarkan index Setelah data dihapus, halaman atau screen tersebut akan ditutup menggunakan Navigator.pop(context);
- Method goToProfileScreen() bersifat asynchronous. Method ini berfungsi untuk menangani perpindahan halaman ke halaman detail profile. Method ini mengambil data berdasarkan index-nya dan menyinpan ke variabel sementara untuk dibawa ke ProfileScreen menggunakan Navigator.push()

Selanjutnya tambahkan baris code pada **file input_screen.dart**

```
import 'package:flutter/material.dart';
import 'package:mcs_bab_4/models/model.dart';
import 'package:mcs_bab_4/providers/app_provider.dart';
import 'package:provider/provider.dart';

class InputScreen extends StatelessWidget {
  const InputScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Consumer<AppProvider>(
      builder: (context, appProvider, child) {
        return Scaffold(
          appBar: AppBar(title: const Text("MCS Bab 4"),),
          body: ListView(
            children: [
              const SizedBox(height: 14,),

              Center(
                child: Container(
                  width: MediaQuery.of(context).size.width/3,
                  height: MediaQuery.of(context).size.width/3,
                  decoration: BoxDecoration(
                    borderRadius: BorderRadius.circular(100),
                    border: Border.all(color: Colors.black, width: 1,),
                    // color: Colors.red,
                  ),
                  child: ClipRRect(
                    borderRadius: BorderRadius.circular(100),
                    child: appProvider.image != null?
                    Image.file(appProvider.image!, fit: BoxFit.fill,) :
                    const Center(child: Text("upload photo")),
                  )
                ),
              ),

              const SizedBox(height: 14,),

              Row(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  GestureDetector(
                    child: const Icon(Icons.camera),
                    onTap: () async{await appProvider.imageFromCamera();},
                  ),
                  GestureDetector(
                    child: const Icon(Icons.image),
                    onTap: () async{await
appProvider.imageFromGallery();},
                  ),
                ],
              ),

              const SizedBox(height: 14,),

              //input npm
              Container(
                margin: const EdgeInsets.symmetric(horizontal: 18),
                child: TextFormField(
```

```
            controller: appProvider.inputNpmController,
            decoration: InputDecoration(
              label: Text(appProvider.npmLabel),
            ),
          ),
        ),

        const SizedBox(height: 14,),

        //input name
        Container(
          margin: const EdgeInsets.symmetric(horizontal: 18),
          child: TextFormField(
            controller: appProvider.inputNameController,
            decoration: InputDecoration(
              label: Text(appProvider.nameLabel),
            ),
          ),
        ),

        const SizedBox(height: 14,),

        //input desc
        Container(
          margin: const EdgeInsets.symmetric(horizontal: 18),
          child: TextFormField(
            controller: appProvider.inputDescController,
            decoration: InputDecoration(
              label: Text(appProvider.descLabel),
            ),
            maxLines: 2,
          ),
        ),

        const SizedBox(height: 40,),

        Center(
          child: GestureDetector(
            child: Container(
              padding: const EdgeInsets.symmetric(
                vertical: 14, horizontal: 18,
              ),
              decoration: BoxDecoration(
                color: const Color(0xff0F6292),
                borderRadius: BorderRadius.circular(50)
              ),
              child: const Text(
                "add data", style: TextStyle(color: Colors.white),
              ),
            ),
            onTap: (){appProvider.addData();},
          ),
        ),

        const SizedBox(height: 40,),

        ListView.builder(
          itemCount: appProvider.data.length,
          shrinkWrap: true,
          physics: const NeverScrollableScrollPhysics(),
          itemBuilder: (context, index) {
```

```
                    Model model = appProvider.data[index];
                    return Container(
                      margin: const EdgeInsets.symmetric(vertical: 6
,horizontal: 20),
                      child: Row(
                        mainAxisAlignment: MainAxisAlignment.spaceBetween,
                        children: [
                          Text("${model.npm}",),
                          GestureDetector(
                            child: const Icon(Icons.arrow_forward),
                            onTap: () => appProvider.goToProfileScreen(
                              context: context,
                              index: index,
                              model: model,
                            ),
                          )
                        ],
                      ),
                    );
                  },
                )
              ],
            ),
          );
        },
      );
    }
}
```

File ini berisi class dengan nama InputScreen yang merupakan widget consumer dari AppProvider (dari file app_provider.dart). hal ini ditandai dengan penggunaan keyword `Consumer<AppProvider>` pada saat awal pembuatan widget. Tidak ada penulisan bussines logic pada file ini, fungsi dari file ini sepenuhnya menangani tampilan form input data untuk user. Bussines logic akan di tangani oleh AppProvider sehingga setiap widget yang memerlukan bussines logic dapat memanggil fungsi pada AppProvider. Sebagai consumer dari AppProvider, widget pada file ini dapat bereaksi terhadap perubahan state tanpa perlu state management manual didalam masing masing widget child didalamnya.

Penggunaan provider pertamakali dapat dilihat pada fitur upload gambar. Fungsi yang menangai proses upload gambar sudah di definisikan pada class AppProvider sehingga pada widget button (GestureDetector) hanya perlu memanggil fungsi tersebut pada provider dan saat sistem mendeteksi adanya perubahan data (file terupload), sistem akan mengirimkan pemberitahuan telah terjadi perubahan state melalui notifyListener() pada fungsi tersebut. Begitu juga penerapan pada widget lain seperti textformfield, button gesture detector “add data” dan juga gesture detector untuk pindah ke profile screen.

Tambahkan kode pada **file profile_screen.dart**

```
import 'package:flutter/material.dart';
import 'package:mcs_bab_4/providers/app_provider.dart';
import 'package:provider/provider.dart';

class ProfileScreen extends StatelessWidget {
  const ProfileScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Consumer<AppProvider>(
      builder: (context, appProvider, child) {
        return Scaffold(
          appBar: AppBar(
            title: const Text("Mhs Profile"),
            actions: [
              IconButton(
                onPressed: () => appProvider.deleteData(context: context),
                icon: const Icon(Icons.delete),
              ),
              const SizedBox(width: 8,)
            ],
          ),
          body: SingleChildScrollView(
            child: Center(
              child: Column(
                children: [
                  Image.file(appProvider.imageProfile!),

                  const SizedBox(height: 20,),

                  Text("${appProvider.npm!}"),

                  const SizedBox(height: 20,),

                  Text("${appProvider.name}"),

                  const SizedBox(height: 20,),

                  Text("${appProvider.desc}"),
                ],
              ),
            ),
          )
        );
      },
    );
  }
}
```

File profile_screen.dart ini memiliki fungsi utama yang sama persis dengan input_screen yaitu hanya menangani tampilan dari program. Pada file ini akan menampilkan detail dari data yang dibawa saat berpindah halaman dari input_screen ke profile_screen. Sebagai consumer dari AppProvider, dapat dilihat tidak ada penulisan proses logika pada file ini. Logika untuk membawa data list berdasarkan index dari halaman satu ke halaman lain sudah di tangani

sepenuhnya oleh AppProvider. File ini hanya perlu memanggil property yang berisi data yang sudah di bawa.

Ubahlah kode pada **file main.dart**

```
import 'package:flutter/material.dart';
import 'package:mcs_bab_4/providers/app_provider.dart';
import 'package:mcs_bab_4/screens/input_screen.dart';
import 'package:provider/provider.dart';

void main(){
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        ChangeNotifierProvider(
          create: (context) => AppProvider(),
        ),
      ],
      child: MaterialApp(
        title: 'State Management Provider',
        debugShowCheckedModeBanner: false,
        theme: ThemeData(
          colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
          useMaterial3: true,
        ),
        home: const InputScreen(),
      ),
    );
  }
}
```

Seperti pada project flutter lainnya. File main.dart merupakan file utama yang akan dijalankan pertama kali ketika program dieksekusi. Jika pada project sebelumnya method widget build yang menangani rendering widget mengembalikan Widget MaterialApp, pada project ini yang dikembalikan adalah widget MultiProvider yang memungkinkan penggunaan beberapa provider. Menggunakan parameter Providers untuk mendefinisikan provider yang akan digunakan dalam bentuk list. Dalam kasus ini didefinisikan ChangeNotifierProvider yang merupakan provider yang akan menangani ChangeNotifier yang sudah digunakan pada saat membuat method. ChangeNotifierProvider menyediakan instance dari class AppProvider yang akan digunakan pada aplikasi sebagai state management.

LAPORAN PENDAHULUAN (LP)

1. Sebutkan macam-macam state management
2. Apa peran dari state management dalam pembangunan aplikasi
3. Jelaskan apa kelebihan state management Provider dengan state management lainnya
4. Jelaskan apa kekurangan state management Provider dengan state management lainnya

LAPORAN AKHIR (LA)

1. Berikan kesimpulan pada bab 4