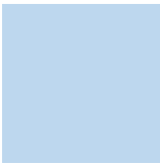


MODUL

PEMROGRAMAN BERBASIS PLATFORM



PERTEMUAN – 12

Modul API 2

PROGRAM STUDI INFORMATIKA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ATMA JAYA YOGYAKARTA





TUJUAN

Setelah menyelesaikan modul ini, praktikan diharapkan mampu :

1. Memahami cara consume API pada flutter.
2. Memahami penggunaan package http pada flutter.
3. Memahami penggunaan future provider dari riverpod.

DASAR TEORI

A. Consume API

Consume API adalah sebuah istilah yang digunakan ketika kita ingin menggunakan suatu api dalam aplikasi. Pada modul kali ini kita akan mempelajari cara consume REST API, yang sudah kita buat di modul sebelumnya.

B. HTTP Package

http adalah sebuah package dari flutter yang dapat kita gunakan untuk membuat request http ke suatu URL. Http memiliki beberapa fungsi seperti `http.get(...)`, `http.post(...)` dan lainnya untuk tiap method. Nilai balikan dari fungsi ini berupa Future. Http juga memungkinkan kita untuk menyetel request body, header, params, dan auth dari setiap request. Syntax dasar untuk membuat request adalah seperti berikut:

1. synchronous:

```
//sesuaikan http atau https
get(Uri.http("dummy.restapiexample.com",
"/api/v1/employees"))
.then((res) => print(res.body))
.catchError((err) => print(err));
```



2. asynchronous

```
try{
var res =await get(Uri.http("dummy.restapiexample.com",
"/api/v1/employees"));
print(res.body);
}
catch(e){
print(e.toString());
}
```

Kode di atas mengirim sebuah get request ke alamat "http://dummy.restapiexample.com/api/v1/employees". Karena berupa Future, untuk handle response yang diterima kita gunakan callback dengan .then() atau .catch() (ketika error). Response yang diterima bisa diakses pada variabel res, disini kita mencoba melihat isi body dari response.

C. Entity

Class yang merepresetasikan suatu entitas/data. Class ini akan kita gunakan untuk menampung data yang diterima dari API sehingga lebih mudah digunakan pada aplikasi.

D. API Client

Class ini akan menyimpan fungsi-fungsi wrapper yang melakukan request ke api dan memproses datanya menjadi objek entity sebelum digunakan pada aplikasi.

E. Riverpod

Riverpod adalah sebuah context provider pada flutter. Riverpod menyediakan beragam jenis provider. Beberapa fungsi yang akan kita gunakan pada modul kali ini adalah:

1. ProviderScope:

Menyimpan state dari provider. Aplikasi perlu dibungkus dengan ProviderScope agar bisa menggunakan Riverpod.



```
void main() {  
  runApp(  
    // Enabled Riverpod for the entire application  
    ProviderScope(  
      child: MyApp(),  
    ),  
  );  
}
```

2. ConsumerWidget/ConsumerStatefulWidget:

Superclass pengganti StatelessWidget/StatefulWidget untuk menggunakan Riverpod. Dengan widget ini kita bisa mendapat akses ke WidgetRef yang berfungsi untuk mengambil value dari sebuah provider.

```
class Example extends ConsumerWidget {  
  const Example({Key? key}): super(key: key);  
  
  @override  
  Widget build(BuildContext context, WidgetRef ref) {  
    final value = ref.watch(helloWorldProvider);  
    return Text(value); // Hello world  
  }  
}
```

3. FutureProvider:

Provider untuk fungsi asynchronous. FutureProvider menyediakan 3 slot (loading,error,data) yang bisa dimasukkan sebuah widget. Widget yang ditampilkan nanti tergantung dari kondisi provider saat ini.

```
final configProvider = FutureProvider<Configuration>((ref) async  
{  
  final content = json.decode(  
    await rootBundle.loadString('assets/configurations.json'),  
  ) as Map<String, Object?>;  
  
  return Configuration.fromJson(content);  
});  
  
Widget build(BuildContext context, WidgetRef ref) {  
  AsyncValue<Configuration> config = ref.watch(configProvider);  
  
  return config.when(  
    loading: () => CircularProgressIndicator(),  
    error: (error, stackTrace) => Text('Error: $error'),  
    data: (config) => Text('Config: $config'),  
  );  
}
```



```
109     loading: () => const CircularProgressIndicator(),  
110     error: (err, stack) => Text('Error: $err'),  
111     data: (config) {  
112         return Text(config.host);  
113     },  
114 );  
115 }  
116
```



GUIDED 11 – Consume API

Poin yang dipelajari dalam Guided 1 ini, yaitu :

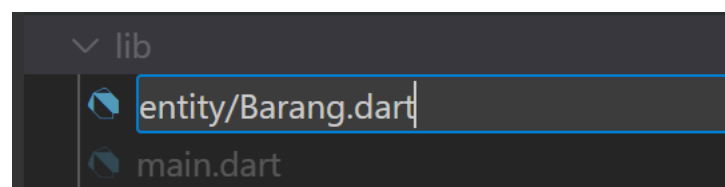
1. Memahami cara menggunakan API.
2. Memahami fungsi class model dan request.
3. Memahami penggunaan Future Provider.

Pada guided 11 ini, kita akan mencoba memahami cara menyambungkan api dengan flutter dengan menggunakan package http dan riverpod sebagai context provider. Silahkan ikuti langkah – langkah berikut ini :

1. Buat project baru dengan menjalankan command flutter create gd_api_xxxx.
2. Buka project menggunakan Visual Studio Code.
3. Di modul ini kita menggunakan dua library yaitu http dan flutter_riverpod. Tambahkan dependency dengan membuka terminal dan jalankan command flutter pub add http flutter_riverpod.

```
ACER@LAPTOP-HU9UN78S MINGW64 /d/sdos/gd_api_xxxx
$ flutter pub add http flutter_riverpod
Resolving dependencies...
  async 2.10.0 (2.11.0 available)
  characters 1.2.1 (1.3.0 available)
  collection 1.17.0 (1.18.0 available)
  flutter_lints 2.0.3 (3.0.1 available)
+ flutter_riverpod 2.3.7 (2.4.6 available)
+ http 0.13.6 (1.1.0 available)
+ http_parser 4.0.2
```

4. Buat sebuah file dart dengan melakukan klik kanan pada folder lib, pilih new file dan masukkan "entity/Barang.dart".

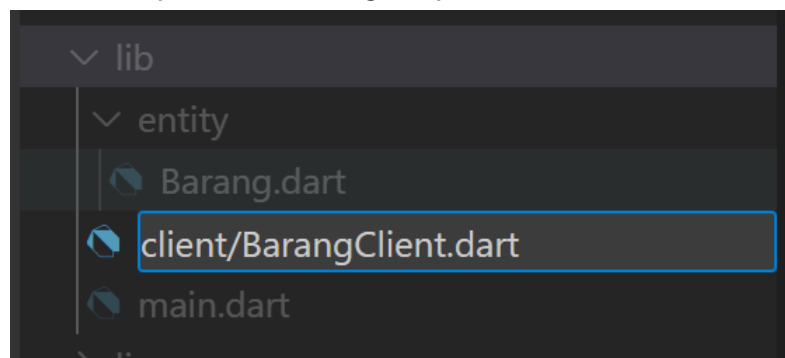




5. Definisikan class dengan nama Barang. Atribut pada class ini disesuaikan dengan kolom-kolom yang ada pada tabel.

```
lib > entity > Barang.dart
1  import 'dart:convert';
2
3  class Barang {
4    int id;
5    String nama;
6    String deskripsi;
7    int stok;
8
9    Barang(
10     {required this.id,
11      required this.nama,
12      required this.deskripsi,
13      required this.stok});
14
15    // untuk membuat objek barang dari data json yang diterima dari API
16    factory Barang.fromRawJson(String str) => Barang.fromJson(json.decode(str));
17    factory Barang.fromJson(Map<String, dynamic> json) => Barang(
18      id: json["id"],
19      nama: json["nama"],
20      deskripsi: json["deskripsi"],
21      stok: json["stok"],
22    );
23
24    // untuk membuat data json dari objek barang yang dikirim ke API
25    String toRawJson() => json.encode(toJson());
26    Map<String, dynamic> toJson() => {
27      "id": id,
28      "nama": nama,
29      "deskripsi": deskripsi,
30      "stok": stok,
31    };
32  }
33
```

6. Buat sebuah file dart dengan klik kanan pada folder lib, pilih new file dan masukkan "requests/BarangRequests.dart".



7. Karena API kita berada di local, ada perbedaan cara untuk mengaksesnya berdasarkan device yang kita gunakan.

- **Emulator:** gunakan **10.0.2.2:8000**.



10.0.2.2 adalah alamat yang digunakan untuk mengakses localhost pada emulator.

- **Hp:** buka command prompt dan jalankan ipconfig, gunakan alamat pada IP IPv4.

```
Command Prompt
Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . :

Wireless LAN adapter Local Area Connection* 1:

Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . :

Wireless LAN adapter Local Area Connection* 2:

Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . :

Ethernet adapter Ethernet 2:

Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . :

Wireless LAN adapter Wi-Fi:

Connection-specific DNS Suffix . :
IPv6 Address. . . . . : 2001:448a:4043:24f8:95e3:c10d:a5a1:12fd
Temporary IPv6 Address. . . . . : 2001:448a:4043:24f8:7499:7e7b:973e:ceda
Link-local IPv6 Address . . . . . : fe80::30ee:1450:fadf:ecb3%11
IPv4 Address. . . . . : 192.168.1.14
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : fe80::1%11
                          192.168.1.1
```

Untuk merujuk ke API, gunakan nama folder API yang ada di htdocs dan ditambahkan '/public'. Sehingga contoh url untuk mengakses API menjadi :

http://192.168.1.14/GD_API_XXXX/public/api/barang.

Note: pastikan laptop dan hp kalian terkoneksi dalam satu jaringan dan project Laravel berada dalam folder htdocs.

8. Buat class dengan nama BarangClient. Pada class ini kita akan membuat fungsi wrapper untuk proses CRUD ke API.



```
lib > client > BarangClient.dart > BarangClient > create
1  import 'package:gd_api_xxxx/entity/Barang.dart';
2
3  import 'dart:convert';
4  import 'package:http/http.dart';
5
6  class BarangClient {
7      //sesuaikan url dan endpoint dengan device yang kalian gunakan untuk uji coba sesuai langkah 7
8
9      // untuk emulator
10     static final String url = '10.0.2.2:8000'; //base url
11     static final String endpoint = '/api/barang'; //base endpoint
12
13     // untuk hp
14     // static final String url = '192.168.1.14';
15     // static final String endpoint = '/GD_API_XXXX/public/api/barang';
16
17     //mengambil semua data barang dari API
18     static Future<List<Barang>> fetchAll() async {
19         try {
20             var response = await get(
21                 Uri.http(url, endpoint)); //request ke api dan menyimpan responsnya
22
23             if (response.statusCode != 200) throw Exception(response.reasonPhrase);
24
25             // mengambil bagian data dari response body
26             Iterable list = json.decode(response.body)['data'];
27
28             // list.map untuk membuat list objek Barang berdasarkan tiap elemen dari list
29             return list.map((e) => Barang.fromJson(e)).toList();
30         } catch (e) {
31             return Future.error(e.toString());
32         }
33     }
34
35     //mengambil data barang dari API sesuai id
36     static Future<Barang> find(id) async {
37         try {
38             var response = await get(Uri.http(url, '$endpoint/$id')); //request ke api
39         }
```

167
168



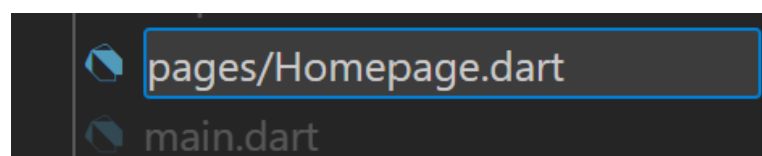
```
lib > client > BarangClient.dart > create
36 static Future<Barang> find(id) async {
37   try {
38     var response = await get(Uri.http(url, '$endpoint/$id')); //request ke api
39
40     if (response.statusCode != 200) throw Exception(response.reasonPhrase);
41
42     // membuat objek Barang berdasarkan bagian data dari response body
43     return Barang.fromJson(json.decode(response.body)['data']);
44   } catch (e) {
45     return Future.error(e.toString());
46   }
47 }
48
49 //membuat data barang baru
50 static Future<Response> create(Barang barang) async {
51   try {
52     var response = await post(Uri.http(url, endpoint),
53       headers: {"Content-Type": "application/json"},
54       body: barang.toRawJson());
55
56     if (response.statusCode != 200) throw Exception(response.reasonPhrase);
57
58     return response;
59   } catch (e) {
60     return Future.error(e.toString());
61   }
62 }
63
64 //mengubah data barang sesuai ID
65 static Future<Response> update(Barang barang) async {
66   try {
67     var response = await put(Uri.http(url, '$endpoint/${barang.id}'),
68       headers: {"Content-Type": "application/json"},
69       body: barang.toRawJson());
70
71     if (response.statusCode != 200) throw Exception(response.reasonPhrase);
72
73     return response;
74   } catch (e) {
```

169
170



```
lib > client > BarangClient.dart > BarangClient > create
53         headers: {"Content-Type": "application/json"},
54         body: barang.toRawJson();
55
56         if (response.statusCode != 200) throw Exception(response.reasonPhrase);
57
58         return response;
59     } catch (e) {
60         return Future.error(e.toString());
61     }
62 }
63
64 //mengubah data barang sesuai ID
65 static Future<Response> update(Barang barang) async {
66     try {
67         var response = await put(Uri.http(url, '$endpoint/${barang.id}'),
68             headers: {"Content-Type": "application/json"},
69             body: barang.toRawJson());
70
71         if (response.statusCode != 200) throw Exception(response.reasonPhrase);
72
73         return response;
74     } catch (e) {
75         return Future.error(e.toString());
76     }
77 }
78
79 //menghapus data barang sesuai ID
80 static Future<Response> destroy(id) async {
81     try {
82         var response = await delete(Uri.http(url, '$endpoint/$id'));
83
84         if (response.statusCode != 200) throw Exception(response.reasonPhrase);
85
86         return response;
87     } catch (e) {
88         return Future.error(e.toString());
89     }
90 }
91 }
```

9. Buat sebuah file dart dengan melakukan klik kanan pada folder lib, pilih new file dan masukkan "pages/Homepage.dart".



10. Pada modul ini kita akan menggunakan Riverpod untuk membaca status pengambilan data dari API. Sebagai persiapan menggunakan



Riverpod, aplikasi perlu dibungkus dengan ProviderScope. Buka halaman main.dart dan lakukan perubahan berikut.

```
lib > main.dart > ...
1 import 'package:flutter/material.dart';
2 import 'package:flutter_riverpod/flutter_riverpod.dart';
3 import 'package:gd_api_xxxx/pages/Homepage.dart';
4
5 void main() {
6   //aplikasi dibungkus ProviderScope agar bisa menggunakan riverpod
7   runApp(ProviderScope(child: const MyApp()));
8 }
9
10 class MyApp extends StatelessWidget {
11   const MyApp({super.key});
12
13   // This widget is the root of your application.
14   @override
15   Widget build(BuildContext context) {
16     return MaterialApp(
17       title: 'Flutter Demo',
18       theme: ThemeData(
19         primarySwatch: Colors.blue,
20       ), // ThemeData
21       home: HomePage(),
22     ), // MaterialApp
23   }
24 }
25
```

11. Buka Homepage.dart, disini kita akan membuat halaman stateless yang menampilkan seluruh data dalam bentuk list.



```
lib > pages > Homepage.dart > Homepage > build
1  import 'package:flutter/material.dart';
2  import 'package:flutter_riverpod/flutter_riverpod.dart';
3
4  import 'package:gd_api_xxxx/client/BarangClient.dart';
5  import 'package:gd_api_xxxx/entity/Barang.dart';
6  import 'package:gd_api_xxxx/pages/EditBarang.dart';
7
8  class Homepage extends ConsumerWidget {
9    Homepage({super.key});
10
11    //provider untuk mengambil list data barang dari API
12    final listBarangProvider = FutureProvider<List<Barang>>((ref) async {
13      return await BarangClient.fetchAll();
14    });
15
16    // aksi ketika floating button ditekan
17    void onAdd(context, ref) {
18      Navigator.push(
19        context,
20        MaterialPageRoute(
21          builder: (context) =>
22            const EditBarang()).then((value) => ref.refresh( // MaterialPageRoute
23              listBarangProvider)); //refresh list data barang ketika kembali ke halaman ini
24    }
25
26    // aksi ketika tombol delete ditekan
27    void onDelete(id, context, ref) async {
28      try {
29        await BarangClient.destroy(id); //menghapus data barang berdasarkan id
30        ref.refresh(listBarangProvider); //refresh list data barang
31        showSnackBar(context, "Delete Success", Colors.green);
32      } catch (e) {
33        showSnackBar(context, e.toString(), Colors.red);
34      }
35    }
36
37    // widget untuk item dalam list
38    ListTile scrollViewItem(Barang b, context, ref) => ListTile(
39      title: Text(b.nama),
```

189
190



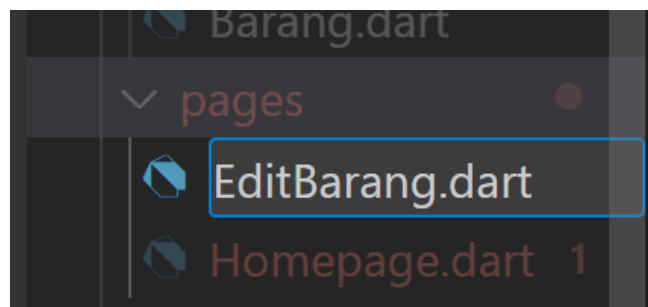
```
lib > pages > Homepage.dart > build
37 // widget untuk item dalam list
38 ListTile scrollViewItem(Barang b, context, ref) => ListTile(
39   title: Text(b.nama),
40   subtitle: Text(b.deskripsi),
41   onTap: () => Navigator.push(context,
42     MaterialPageRoute(builder: (context) => EditBarang(id: b.id)))
43     .then((value) => ref.refresh(listBarangProvider)),
44   trailing: IconButton(
45     onPressed: () => onDelete(b.id, context, ref),
46     icon: const Icon(Icons.delete)); // IconButton // ListTile
47
48 @override
49 Widget build(BuildContext context, WidgetRef ref) {
50   var listener = ref.watch(listBarangProvider); //
51
52   return Scaffold(
53     appBar: AppBar(
54       title: const Text("GD API XXXX"),
55     ), // AppBar
56     floatingActionButton: FloatingActionButton(
57       child: const Icon(Icons.add),
58       onPressed: () => onAdd(context, ref),
59     ), // FloatingActionButton
60     body: listener.when(
61       data: (barang) => SingleChildScrollView(
62         child: Column(
63           children: barang
64             .map((barang) => scrollViewItem(barang, context, ref))
65             .toList()), // Column
66       ), //muncul ketika data berhasil diambil // SingleChildScrollView
67       error: (err, s) =>
68         Center(child: Text(err.toString())), //muncul ketika error
69       loading: () => const Center(
70         child: CircularProgressIndicator(),
71       ), //muncul ketika loading // Center
72     ),
73   ); // Scaffold
74 }
75 }
```

191
192



```
lib > pages > Homepage.dart > build
52   return Scaffold(
53     appBar: AppBar(
54       title: const Text("GD API XXXX"),
55     ), // AppBar
56     floatingActionButton: FloatingActionButton(
57       child: const Icon(Icons.add),
58       onPressed: () => onAdd(context, ref),
59     ), // FloatingActionButton
60     body: listener.when(
61       data: (barang) => SingleChildScrollView(
62         child: Column(
63           children: barang
64             .map((barang) => scrollViewItem(barang, context, ref))
65             .toList(), // Column
66         ), //muncul ketika data berhasil diambil // SingleChildScrollView
67       error: (err, s) =>
68         Center(child: Text(err.toString())), //muncul ketika error
69       loading: () => const Center(
70         child: CircularProgressIndicator(),
71       ), //muncul ketika loading // Center
72     ),
73   ); // Scaffold
74 }
75 }
76
77 // untuk menampilkan snackbar
78 void showSnackBar(BuildContext context, String msg, Color bg) {
79   final scaffold = ScaffoldMessenger.of(context);
80   scaffold.showSnackBar(
81     SnackBar(
82       content: Text(msg),
83       backgroundColor: bg,
84       action: SnackBarAction(
85         label: 'hide', onPressed: scaffold.hideCurrentSnackBar), // SnackBarAct
86     ), // SnackBar
87   );
88 }
89
```

12. Terakhir buat file baru pada folder pages dengan nama EditBarang.dart





13. Halaman ini akan kita gunakan untuk tambah dan edit data sekaligus.
Ubah isinya menjadi berikut:

```
lib > pages > EditBarang.dart > _EditBarangState > build
1  import 'package:flutter/material.dart';
2  import 'package:flutter/src/widgets/framework.dart';
3  import 'package:gd_api_xxxx/client/BarangClient.dart';
4  import 'package:gd_api_xxxx/entity/Barang.dart';
5  import 'package:gd_api_xxxx/pages/Homepage.dart';
6
7  class EditBarang extends StatefulWidget {
8    const EditBarang({super.key, this.id});
9    final int? id;
10
11    @override
12    State<EditBarang> createState() => _EditBarangState();
13  }
14
15  class _EditBarangState extends State<EditBarang> {
16    final _formKey = GlobalKey<FormState>();
17    final nameController = TextEditingController();
18    final descController = TextEditingController();
19    final stockController = TextEditingController();
20    bool isLoading = false;
21
22    void loadData() async {
23      setState(() {
24        isLoading = true;
25      });
26      try {
27        // mencari data barang dari API dan menampilkan valuenya
28        Barang res = await BarangClient.find(widget.id);
29        setState(() {
30          isLoading = false;
31          nameController.value = TextEditingController(text: res.nama);
32          descController.value = TextEditingController(text: res.deskripsi);
33          stockController.value = TextEditingController(text: res.stok.toString());
34        });
35      } catch (err) {
36        showSnackBar(context, err.toString(), Colors.red);
37        Navigator.pop(context);
38      }
39    }
40  }
```




```
lib > pages > EditBarang.dart > _EditBarangState > build

40
41 @override
42 void initState() {
43   super.initState();
44   // jika diberikan id, muat data sebelumnya
45   if (widget.id != null) {
46     loadData();
47   }
48 }
49
50 @override
51 Widget build(BuildContext context) {
52   // aksi ketika form disubmit
53   void onSubmit() async {
54     if (!_formKey.currentState!.validate()) return;
55
56     // objek barang berdasarkan input
57     Barang input = Barang(
58       id: widget.id ?? 0,
59       nama: nameController.text,
60       deskripsi: descController.text,
61       stok: int.parse(stockController.text));
62
63     try {
64       if (widget.id == null) {
65         await BarangClient.create(input);
66       } else {
67         await BarangClient.update(input);
68       }
69
70       showSnackBar(context, 'Success', Colors.green);
71       Navigator.pop(context);
72     } catch (err) {
73       showSnackBar(context, err.toString(), Colors.red);
74       Navigator.pop(context);
75     }
76   }
77
78   return Scaffold(
```

205

206

207



```
lib > pages > EditBarang.dart > _EditBarangState > build
78   return Scaffold(
79     appBar: AppBar(
80       title: Text(widget.id == null ? "Tambah Barang" : "Edit Barang"),
81     ), // AppBar
82     body: Container(
83       padding: const EdgeInsets.all(10),
84       child: isLoading
85         ? const Center(
86           child: CircularProgressIndicator(),
87         ) // Center
88       : Form(
89         key: _formKey,
90         child: Column(
91           children: [
92             Container(
93               padding: const EdgeInsets.symmetric(
94                 horizontal: 25, vertical: 10), // EdgeInsets.symmetric
95               child: TextFormField(
96                 decoration: const InputDecoration(
97                   border: UnderlineInputBorder(),
98                   labelText: 'Masukkan nama',
99                 ), // InputDecoration
100                controller: nameController,
101                validator: (value) {
102                  if (value == null || value.isEmpty) {
103                    return 'Field Required';
104                  }
105                  return null;
106                },
107              ), // TextFormField
108            ), // Container
109            Container(
110              padding: const EdgeInsets.symmetric(
111                horizontal: 25, vertical: 10), // EdgeInsets.symmetric
112              child: TextFormField(
113                decoration: const InputDecoration(
114                  border: UnderlineInputBorder(),
115                  labelText: 'Masukkan deskripsi',
116                ), // InputDecoration
```

208

209



```
lib > pages > EditBarang.dart > _EditBarangState > build
117 controller: descController,
118 validator: (value) {
119   if (value == null || value.isEmpty) {
120     return 'Field Required';
121   }
122
123   return null;
124 },
125 ), // TextFormField
126 ], // Container
127 Container(
128   padding: const EdgeInsets.symmetric(
129     horizontal: 25, vertical: 10), // EdgeInsets.symmetric
130   child: TextFormField(
131     keyboardType: TextInputType.number,
132     decoration: const InputDecoration(
133       border: UnderlineInputBorder(),
134       labelText: 'Masukkan stok',
135     ), // InputDecoration
136     controller: stockController,
137   ), // TextFormField
138 ), // Container
139 Container(
140   width: double.infinity,
141   padding: const EdgeInsets.symmetric(
142     horizontal: 25, vertical: 10), // EdgeInsets.symmetric
143   child: ElevatedButton(
144     onPressed: onSubmit,
145     child: Text(
146       widget.id == null ? 'Tambah' : 'Edit',
147     ), // Text
148   ), // ElevatedButton
149 ), // Container
150 ],
151 ), // Column // Form
152 ), // Container
153 ); // Scaffold
154 }
155 }
```

14. Saat pengujian jangan lupa jalankan XAMPP dan project Laravel yang sudah dibuat di GD 1 API.

```
ACER@LAPTOP-HU9UNJ8S MINGW64 /c/xampp/htdocs/apiPbp (api)
$ php artisan serve
```

```
INFO Server running on [http://127.0.0.1:8000].
```

15. Hasil akhir GD dapat dilihat pada video demo yang ada di situs kuliah.



PEMBAHASAN GUIDED 11

Pada guided 11 di atas, anda telah mencoba untuk membuat sebuah aplikasi Flutter yang terhubung dengan API. Pada aplikasi ini ada dua halaman yang dibuat :

1. Homepage :

Halaman ini menampilkan sebuah list dengan data yang diambil dari API. Karena disini kita menggunakan provider dari riverpod, halaman ini mengextends ConsumerWidget.

```
class Homepage extends ConsumerWidget {
```

Untuk mengambil data, kita buat sebuah FutureProvider yang memanggil fungsi fetchAll dari client.

```
//provider untuk mengambil list data barang dari API
final listBarangProvider = FutureProvider<List<Barang>>((ref) async {
  return await BarangClient.fetchAll();
});
```

Untuk membaca provider, buat sebuah listener yang mengawasi status pengambilan data dari provider. Untuk itu kita menggunakan WidgetRef yang disediakan oleh ConsumerWidget.

```
@override
Widget build(BuildContext context, WidgetRef ref) {
  var listener = ref.watch(listBarangProvider); //
```

Kemudian untuk menampilkan data kita masukkan masing-masing widget untuk slot error, loading dan data.

```
body: listener.when(
  data: (barang) => SingleChildScrollView(
    child: Column(
      children: barang
        .map((barang) => scrollViewItem(barang, context, ref))
        .toList(), // Column
    ), //muncul ketika data berhasil diambil // SingleChildScrollView
  error: (err, s) =>
    Center(child: Text(err.toString())), //muncul ketika error
  loading: () => const Center(
    child: CircularProgressIndicator(),
  ), //muncul ketika loading // Center
),
```



Pada tiap item dalam list, ada sebuah button untuk menghapus data. Ketika button ditekan akan memanggil fungsi destroy dan me-*refresh* data.

```
// aksi ketika tombol delete ditekan
void onDelete(id, context, ref) async {
  try {
    await BarangClient.destroy(id); //menghapus data barang berdasarkan id
    ref.refresh(listBarangProvider); //refresh list data barang
    showSnackBar(context, "Delete Success", Colors.green);
  } catch (e) {
    showSnackBar(context, e.toString(), Colors.red);
  }
}

// widget untuk item dalam list
ListTile scrollViewItem(Barang b, context, ref) => ListTile(
  title: Text(b.nama),
  subtitle: Text(b.deskripsi),
  onTap: () => Navigator.push(context,
    MaterialPageRoute(builder: (context) => EditBarang(id: b.id)))
    .then((value) => ref.refresh(listBarangProvider)),
  trailing: IconButton(
    onPressed: () => onDelete(b.id, context, ref),
    icon: const Icon(Icons.delete)); // IconButton // ListTile
```

Di halaman ini juga terdapat floating button yang membawa ke halaman EditPage dengan Navigator.push.

```
// aksi ketika floating button ditekan
void onAdd(context, ref) {
  Navigator.push(
    context,
    MaterialPageRoute(
      builder: (context) =>
        const EditBarang()).then((value) => ref.refresh( // MaterialPageRoute
listBarangProvider)); //refresh list data barang ketika kembali ke halaman ini
}
```

Untuk parameter EditPage dikosongkan yang menandakan bahwa kita ingin melakukan penambahan data. Pada Navigator.push kita berikan callback, sehingga ketika kita kembali ke halaman ini data akan otomatis ter-*refresh*.

2. EditPage:

Di halaman ini karena tidak menggunakan provider tidak perlu menggunakan ConsumerStatefulWidget. Halaman ini kita gunakan untuk menambah data atau mengupdate data. Di halaman ini



terdapat parameter id yang ketika kosong artinya kita ingin menambah data dan jika terisi artinya mengupdate data.

```
class EditBarang extends StatefulWidget {  
  const EditBarang({super.key, this.id});  
  final int? id;  
  
  @override  
  State<EditBarang> createState() => _EditBarangState();  
}
```

Jika id terisi, kita muat terlebih dahulu data lamanya dengan memanggil fungsi loadData di initState. Dalam fungsi ini kita mengambil data barang berdasarkan idnya di API dan menampilkan datanya ke field input.

```
void loadData() async {  
  setState(() {  
    isLoading = true;  
  });  
  try {  
    // mencari data barang dari API dan menampilkan valuenya  
    Barang res = await BarangClient.find(widget.id);  
    setState(() {  
      isLoading = false;  
      nameController.value = TextEditingValue(text: res.nama);  
      descController.value = TextEditingValue(text: res.deskripsi);  
      stockController.value = TextEditingValue(text: res.stok.toString());  
    });  
  } catch (err) {  
    showSnackBar(context, err.toString(), Colors.red);  
    Navigator.pop(context);  
  }  
}  
  
@override  
void initState() {  
  super.initState();  
  // jika diberikan id, muat data sebelumnya  
  if (widget.id != null) {  
    loadData();  
  }  
}
```

Dalam fungsi submit kita membuat data Barang berdasarkan input dan menjalankan aksi create/update dengan mengirim data Barang tersebut.



```
// aksi ketika form disubmit
void onSubmit() async {
  if (!_formKey.currentState!.validate()) return;

  // objek barang berdasarkan input
  Barang input = Barang(
    id: widget.id ?? 0,
    nama: nameController.text,
    deskripsi: descController.text,
    stok: int.parse(stockController.text));

  try {
    if (widget.id == null) {
      await BarangClient.create(input);
    } else {
      await BarangClient.update(input);
    }

    showSnackBar(context, 'Success', Colors.green);
    Navigator.pop(context);
  } catch (err) {
    showSnackBar(context, err.toString(), Colors.red);
    Navigator.pop(context);
  }
}
```

264

265

266 ATURAN Pengerjaan Guided :

- 267
- Guided dikerjakan selama waktu perkuliahan berlangsung.
 - 268 - Penamaan projek guided harus sesuai dengan yang sudah
 - 269 dicontohkan.
 - 270 - Guided dikumpulkan melalui github dengan penamaan setiap file
 - 271 pada github adalah : **NAMAGUIDED_XXXX** (contoh :
 - 272 **Guided1_Intent_9999**)
 - 273 ○ **NAMAGUIDED → SESUAI DENGAN CONTOH DALAM MODUL INI**
 - 274 ○ **XXXX → 4 DIGIT TERAKHIR NPM**
 - 275 - Setelah diupload melalui github, jangan lupa untuk
 - 276 mengumpulkan keseluruhan link file github melalui situs kuliah.
 - 277 - Cara upload ke github, silahkan melihat pada modul **"UPLOAD**
 - 278 **GITHUB"**.