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 menghandle 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.

```dart
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.

```dart
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.

```dart
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: () => const CircularProgressIndicator(),
    error: (err, stack) => Text('Error: $err'),
    data: (config) {
      return Text(config.host);
    },
  );
}
```

GUIDED 11 – Consume API

Poin yang dipelajari dalam Guided 1 ini, yaitu :

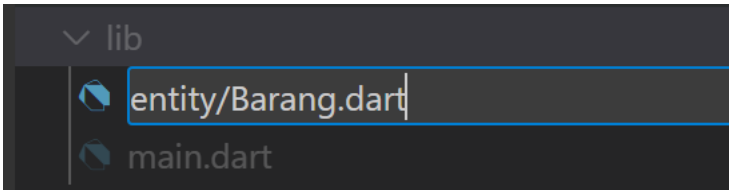
1. Memahami cara menggunaan API.
2. Memahami fungsi class model dan request.
3. Memahami penggunaan Future Provider.

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

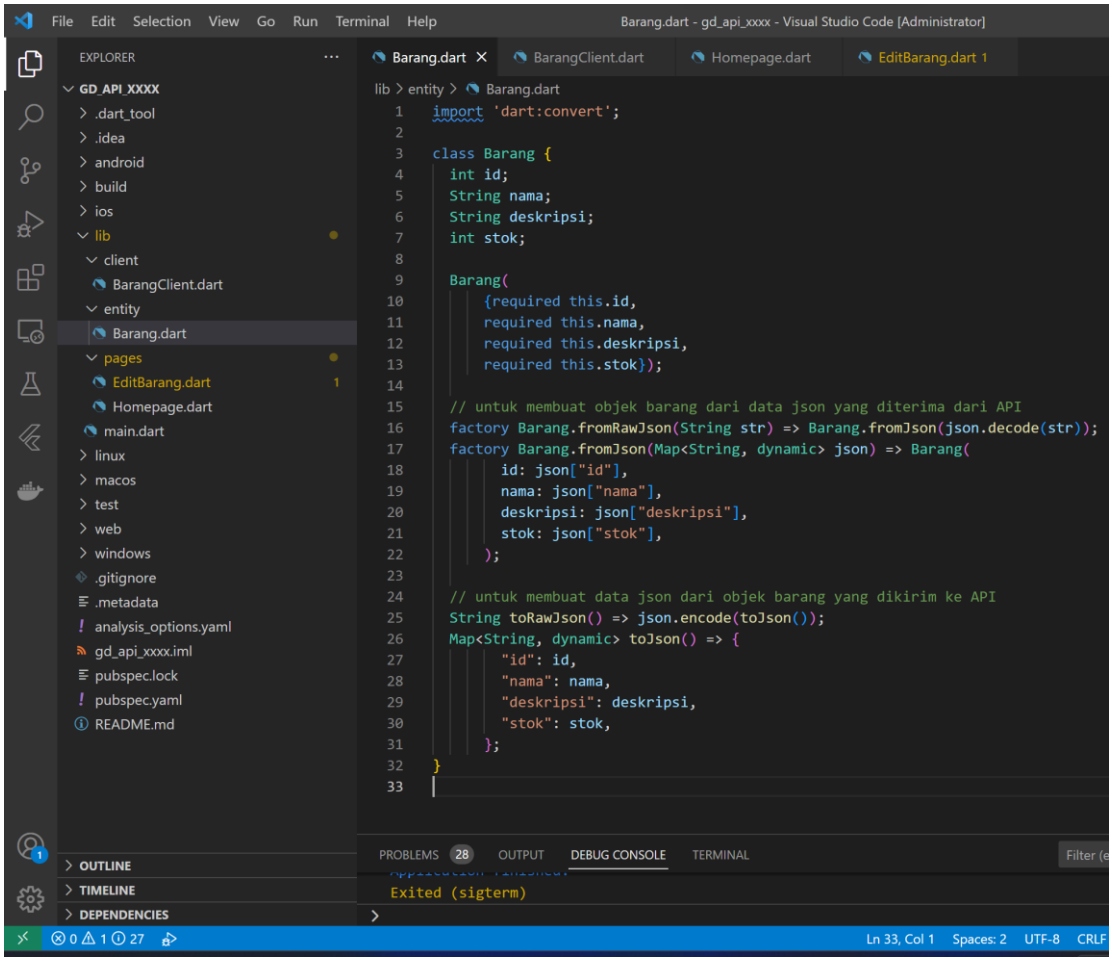
1. Buat project baru dengan menjalankan command flutter create gd_api_xxxx.
2. Buka project menggunakan Visual Studio Code.
3. Di modul ini kita menggunakan dua library yaitu http dan flutter_riverpod. Tambahkan dependency dengan membuka terminal dan jalankan command flutter pub add http flutter_riverpod.

```
ACER@LAPTOP-HU9UNJ8S 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
```

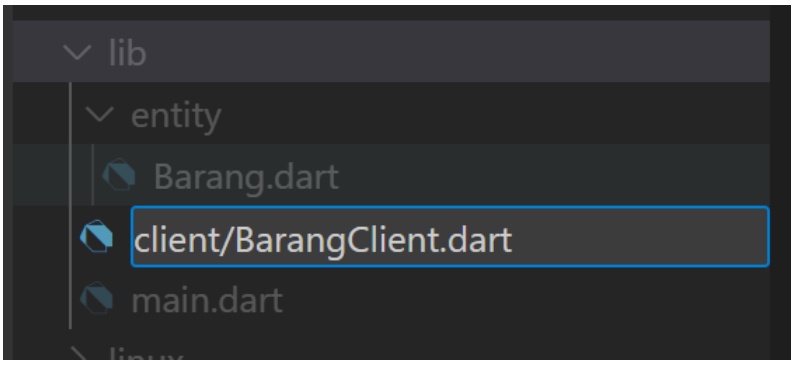
4. Buat sebuah file dart dengan melakukan klik kanan pada folder lib, pilih new file dan masukkan “entity/Barang.dart”.



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



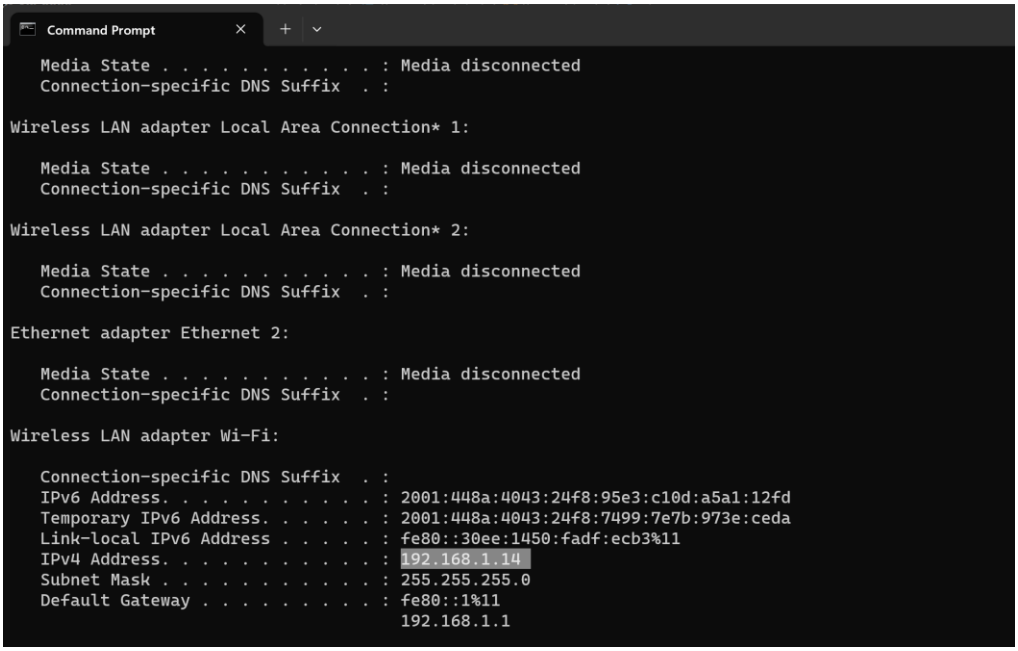
6. Buat sebuah file dart dengan klik kanan pada folder lib, pilih new file dan masukkan “requests/BarangRequests.dart”.



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.

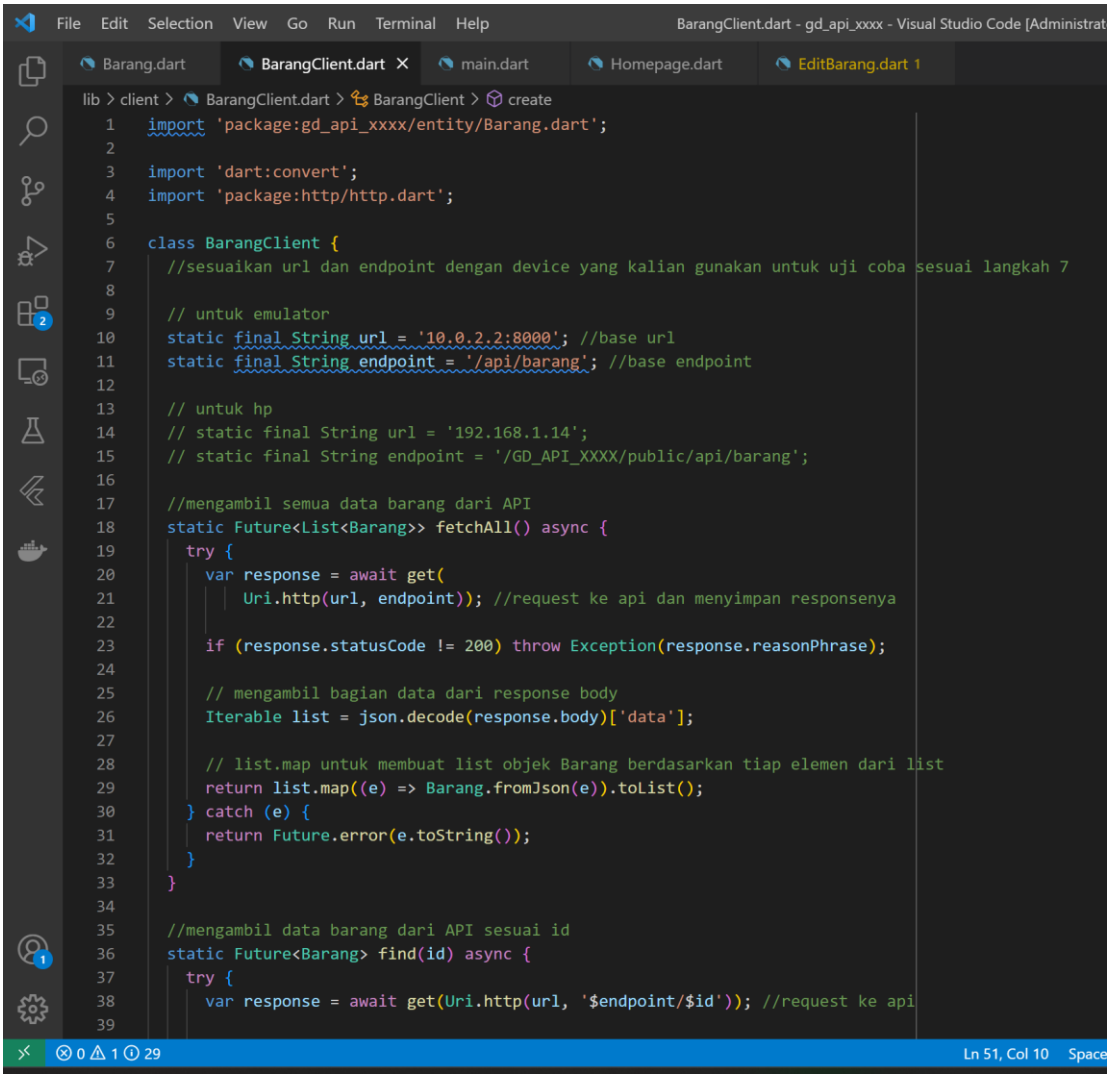


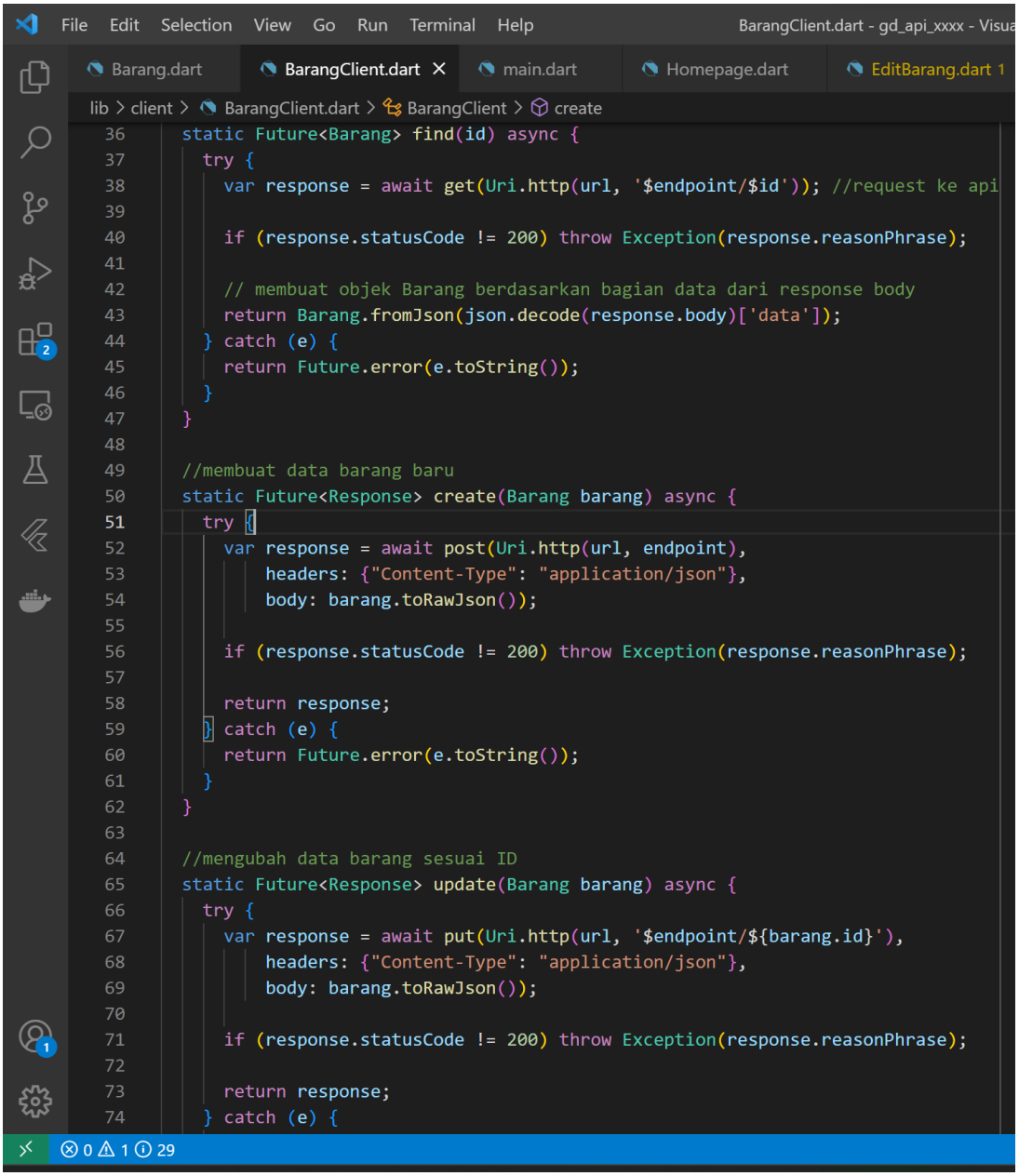
Untuk merujuk ke API, gunakan nama folder API yang ada di htdocs dan ditambahkan '/public'. Sehingga contoh url untuk mengakses API menjadi :

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

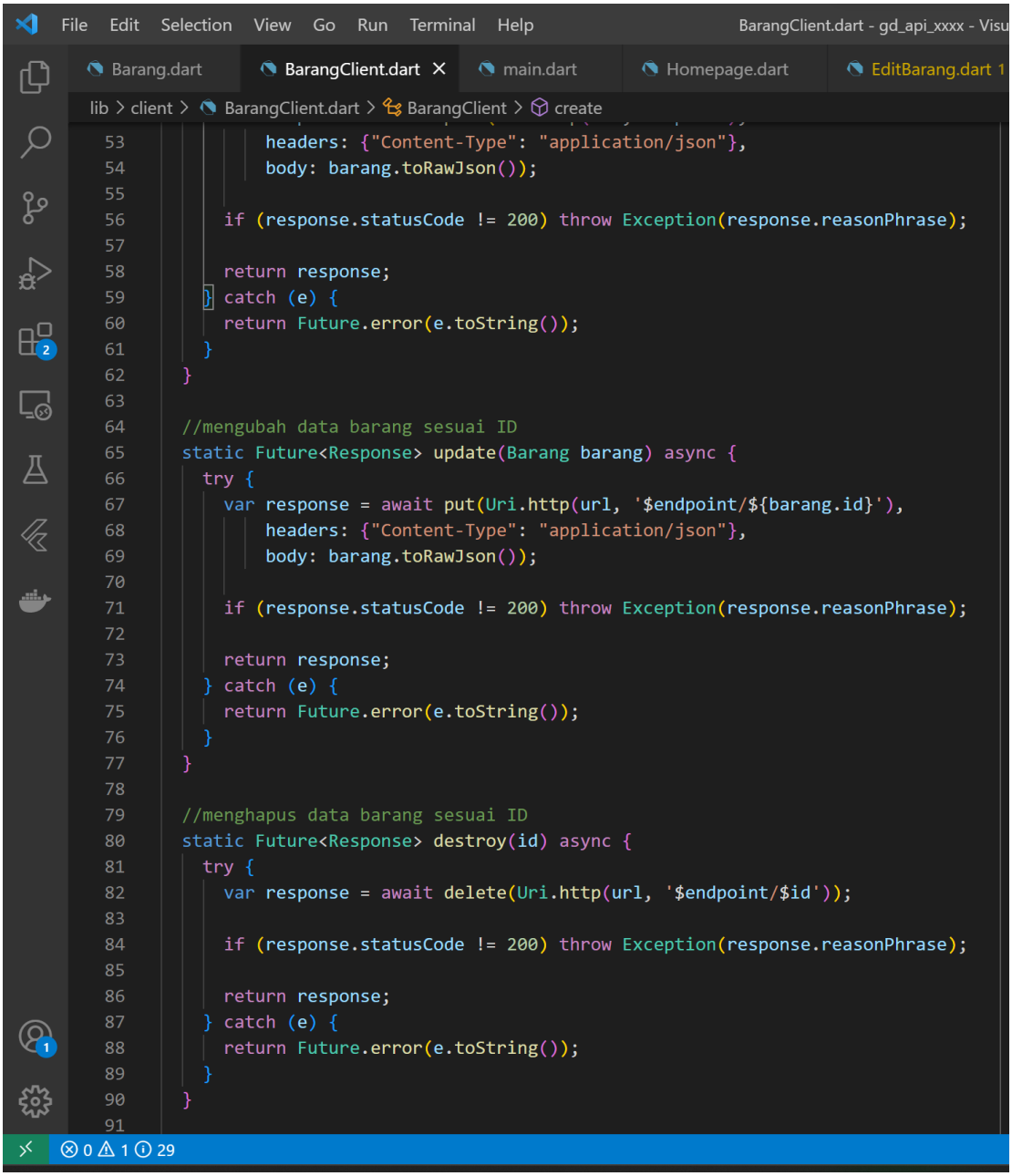
Note: pastikan laptop dan hp kalian terkoneksi dalam satu jaringan dan project Laravel berada dalam folder htdocs.

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

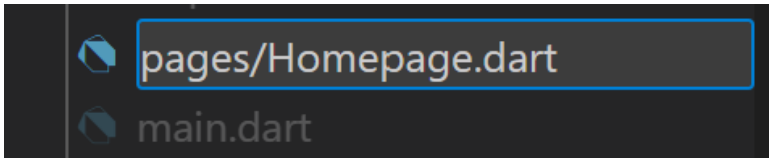

File Edit Selection View Go Run Terminal Help
BarangClient.dart - gd_api_xxxx - Visual Studio Code [Administrat
Barang.dart
BarangClient.dart
main.dart
Homepage.dart
EditBarang.dart 1
lib > client > BarangClient.dart > BarangClient > create
1 import 'package:gd_api_xxxx/entity/Barang.dart';
2
3 import '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 responsenya
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
0 1 29
Ln 51, Col 10
Space


File Edit Selection View Go Run Terminal Help
BarangClient.dart - gd_api_xxxx - Visua
Barang.dart
BarangClient.dart
main.dart
Homepage.dart
EditBarang.dart 1
lib > client > BarangClient.dart > BarangClient > create
static Future<Barang> find(id) async {
try {
var response = await get(Uri.http(url, '$endpoint/$id')); //request ke api
if (response.statusCode != 200) throw Exception(response.reasonPhrase);
// membuat objek Barang berdasarkan bagian data dari response body
return Barang.fromJson(json.decode(response.body)['data']);
} catch (e) {
return Future.error(e.toString());
}
}
//membuat data barang baru
static Future<Response> create(Barang barang) async {
try {
var response = await post(Uri.http(url, endpoint),
headers: {"Content-Type": "application/json"},
body: barang.toRawJson());
if (response.statusCode != 200) throw Exception(response.reasonPhrase);
return response;
} catch (e) {
return Future.error(e.toString());
}
}
//mengubah data barang sesuai ID
static Future<Response> update(Barang barang) async {
try {
var response = await put(Uri.http(url, '$endpoint/${barang.id}'),
headers: {"Content-Type": "application/json"},
body: barang.toRawJson());
if (response.statusCode != 200) throw Exception(response.reasonPhrase);
return response;
} catch (e) {
⊗ 0 ⚠ 1 ⓘ 29

169
170

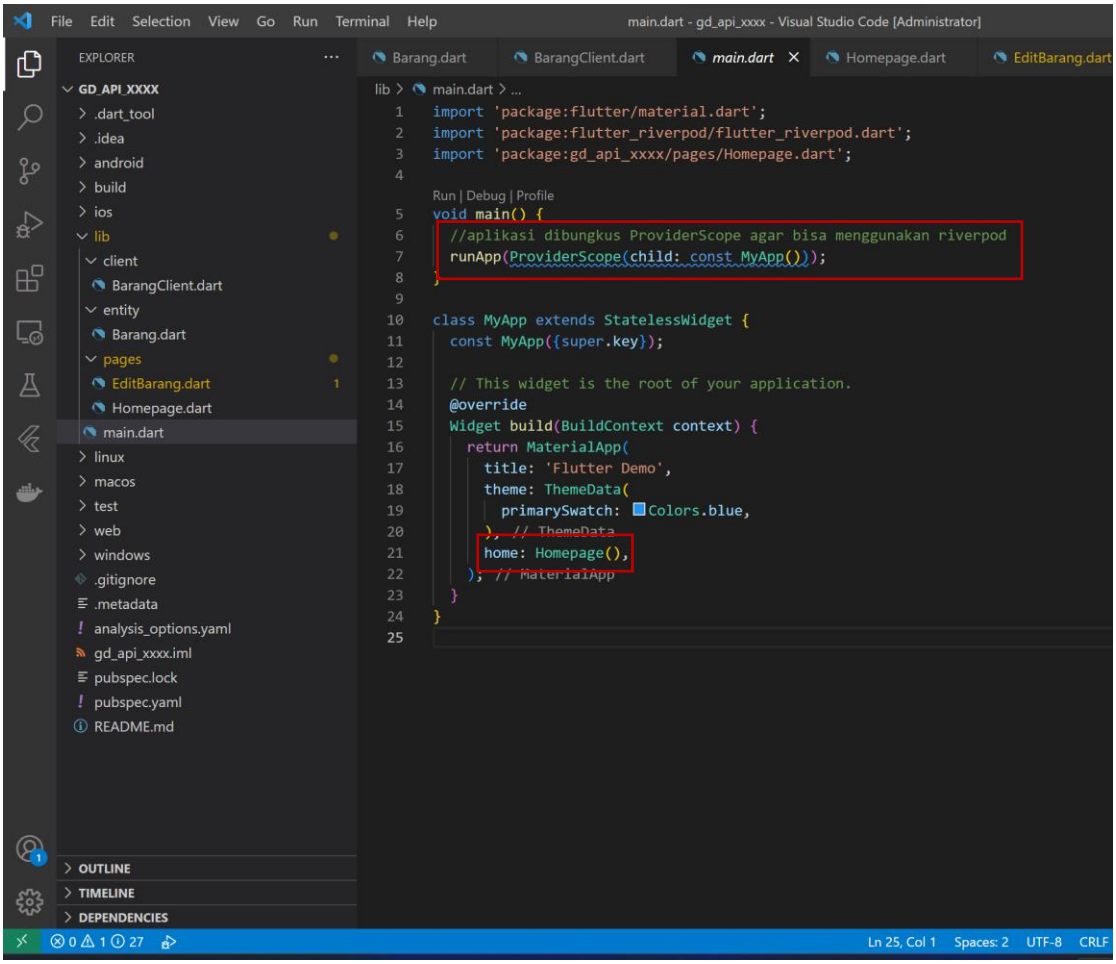


9. Buat sebuah file dart dengan melakukan klik kanan pada folder lib, pilih new file dan masukkan “pages/Homepage.dart”.

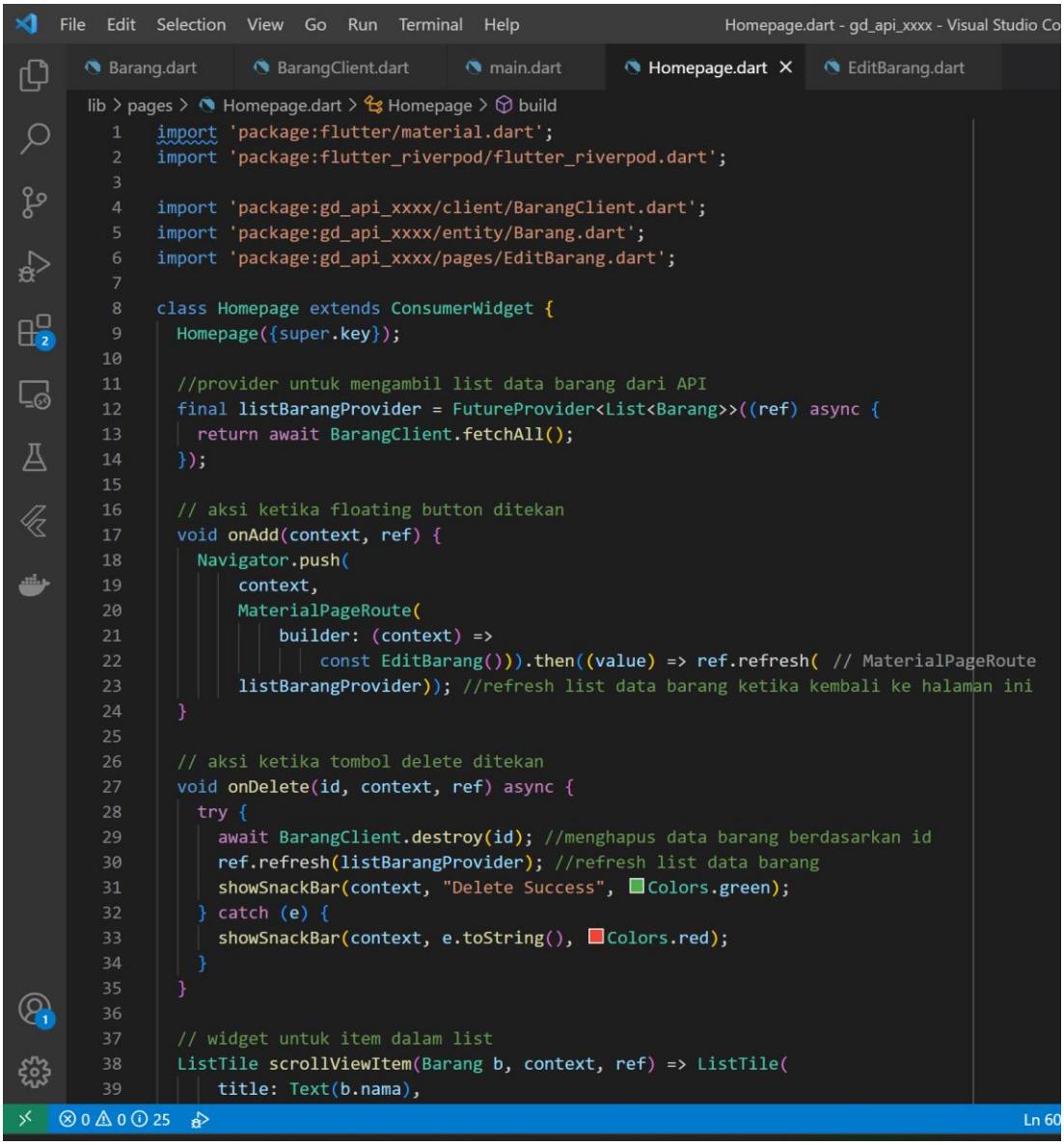


10. Pada modul ini kita akan menggunakan Riverpod untuk membaca status pengambilan data dari API. Sebagi persiapan menggunakan

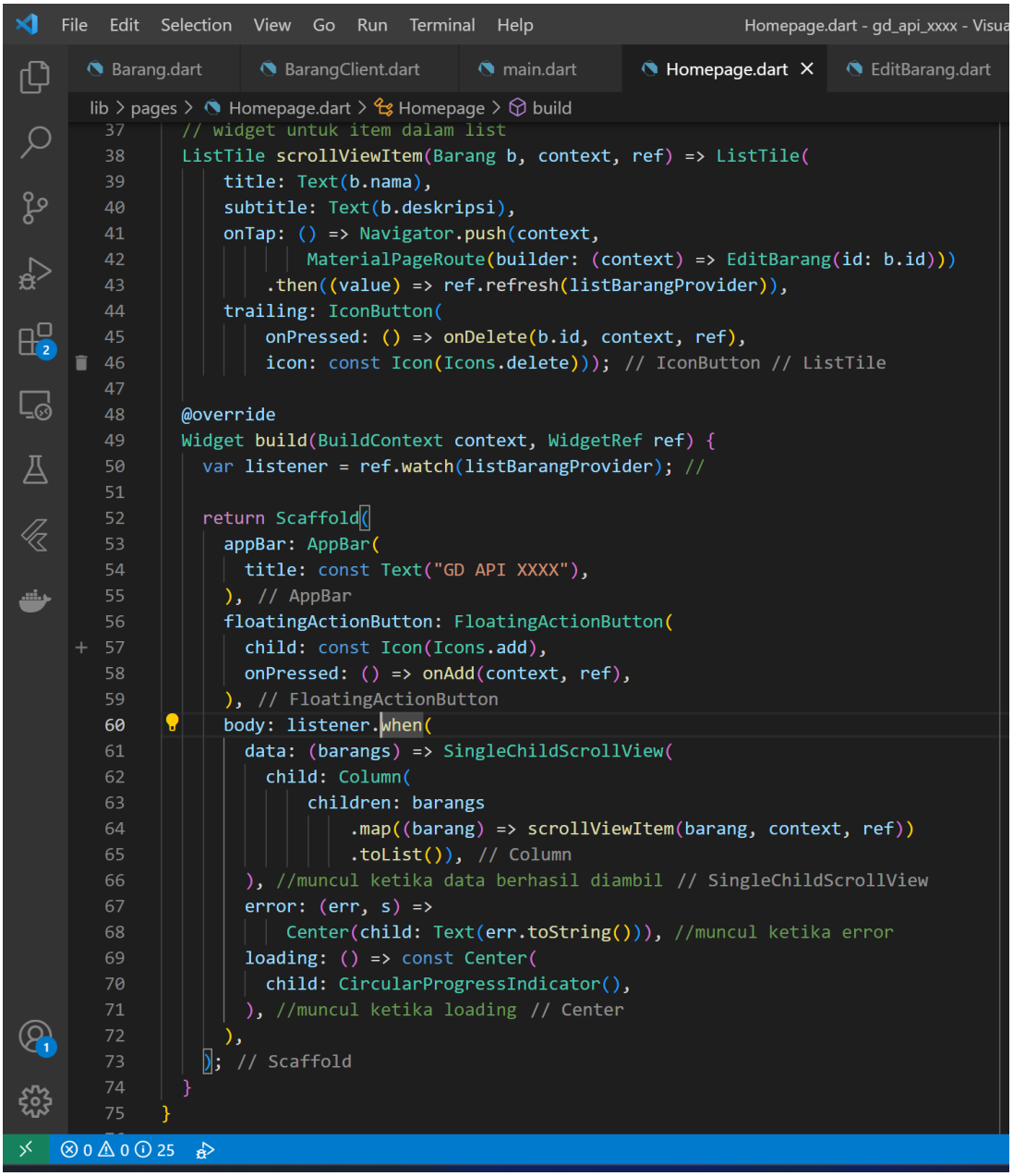
Riverpod, aplikasi perlu dibungkus dengan ProviderScope. Buka halaman main.dart dan lakukan perubahan berikut.



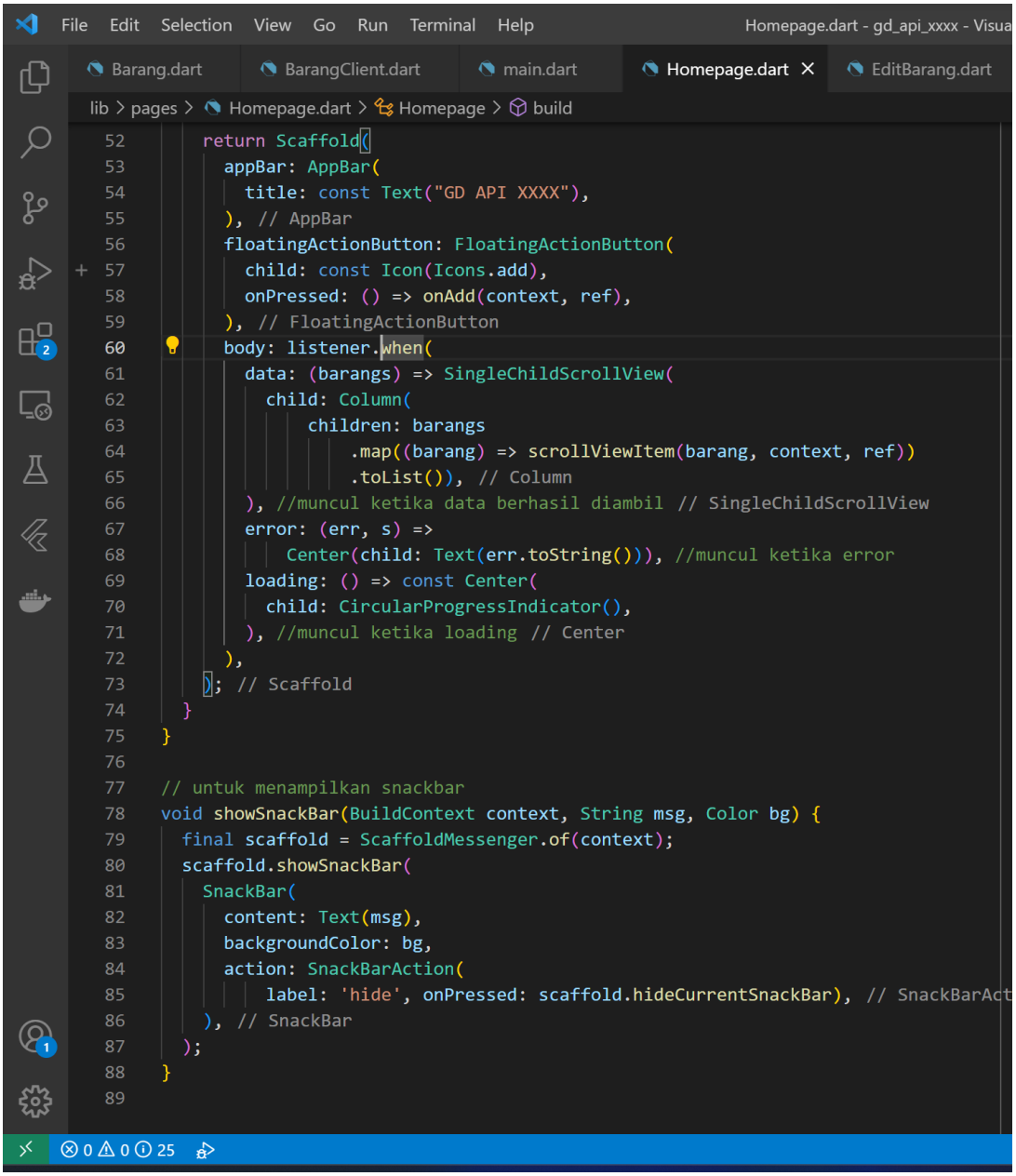
11. Buka Homepage.dart, disini kita akan membuat halaman stateless yang menampilkan seluruh data dalam bentuk list.


File Edit Selection View Go Run Terminal Help
Homepage.dart - gd_api_xxxx - Visual Studio Co
Barang.dart
BarangClient.dart
main.dart
Homepage.dart
EditBarang.dart
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),
0 0 25
Ln 60

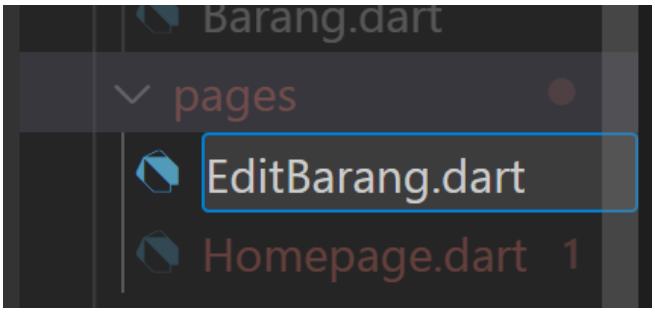
189
190


File Edit Selection View Go Run Terminal Help
Homepage.dart - gd_api_xxxx - Visua
Barang.dart
BarangClient.dart
main.dart
Homepage.dart
EditBarang.dart
lib > pages > Homepage.dart > Homepage > build
// 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
@override
Widget build(BuildContext context, WidgetRef ref) {
var listener = ref.watch(listBarangProvider); //
return Scaffold(
appBar: AppBar(
title: const Text("GD API XXXX"),
), // AppBar
floatingActionButton: FloatingActionButton(
child: const Icon(Icons.add),
onPressed: () => onAdd(context, ref),
), // FloatingActionButton
body: listener.when(
data: (barangs) => SingleChildScrollView(
child: Column(
children: barangs
.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
),
); // Scaffold
}
}
0 0 25

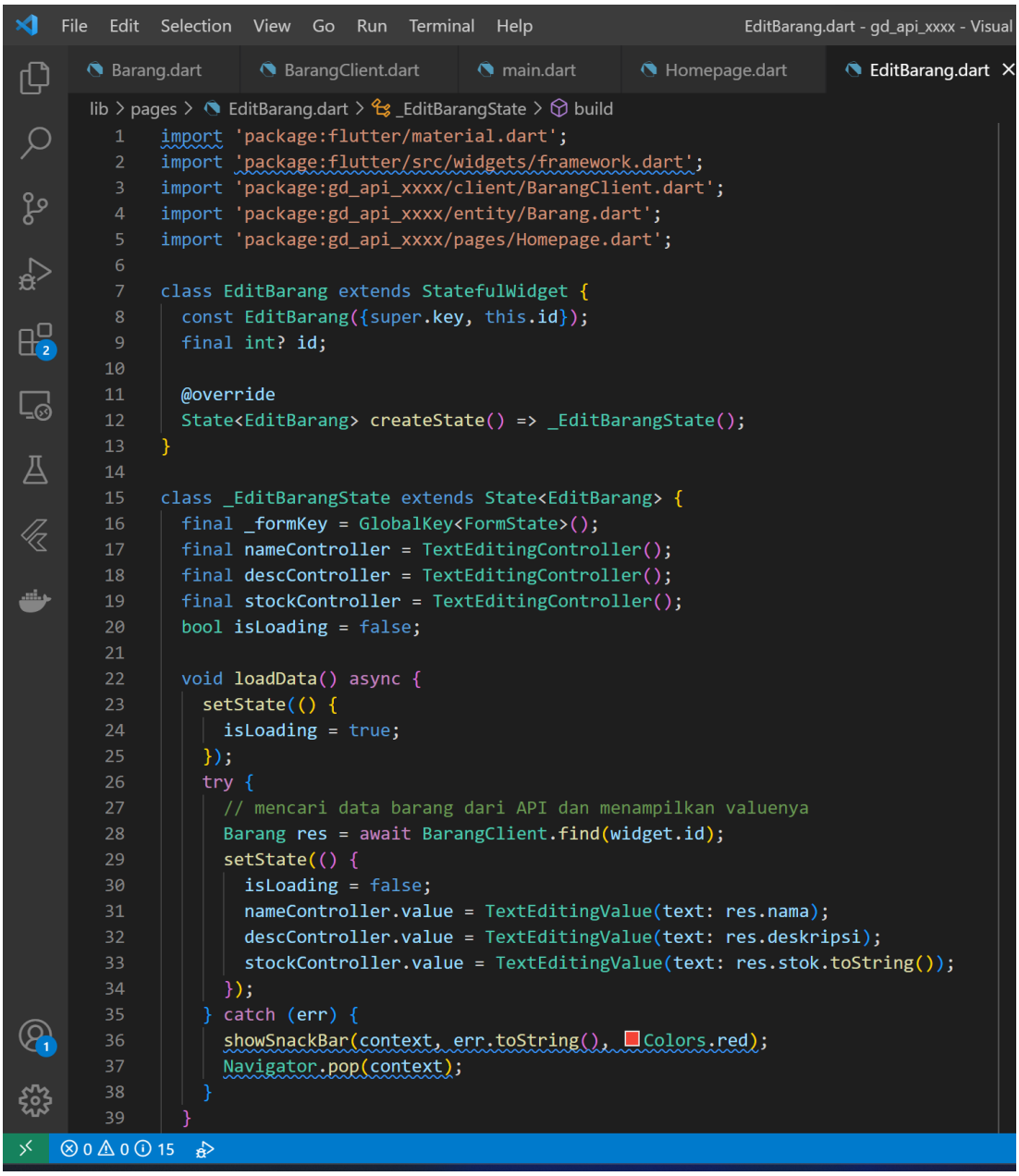
191
192

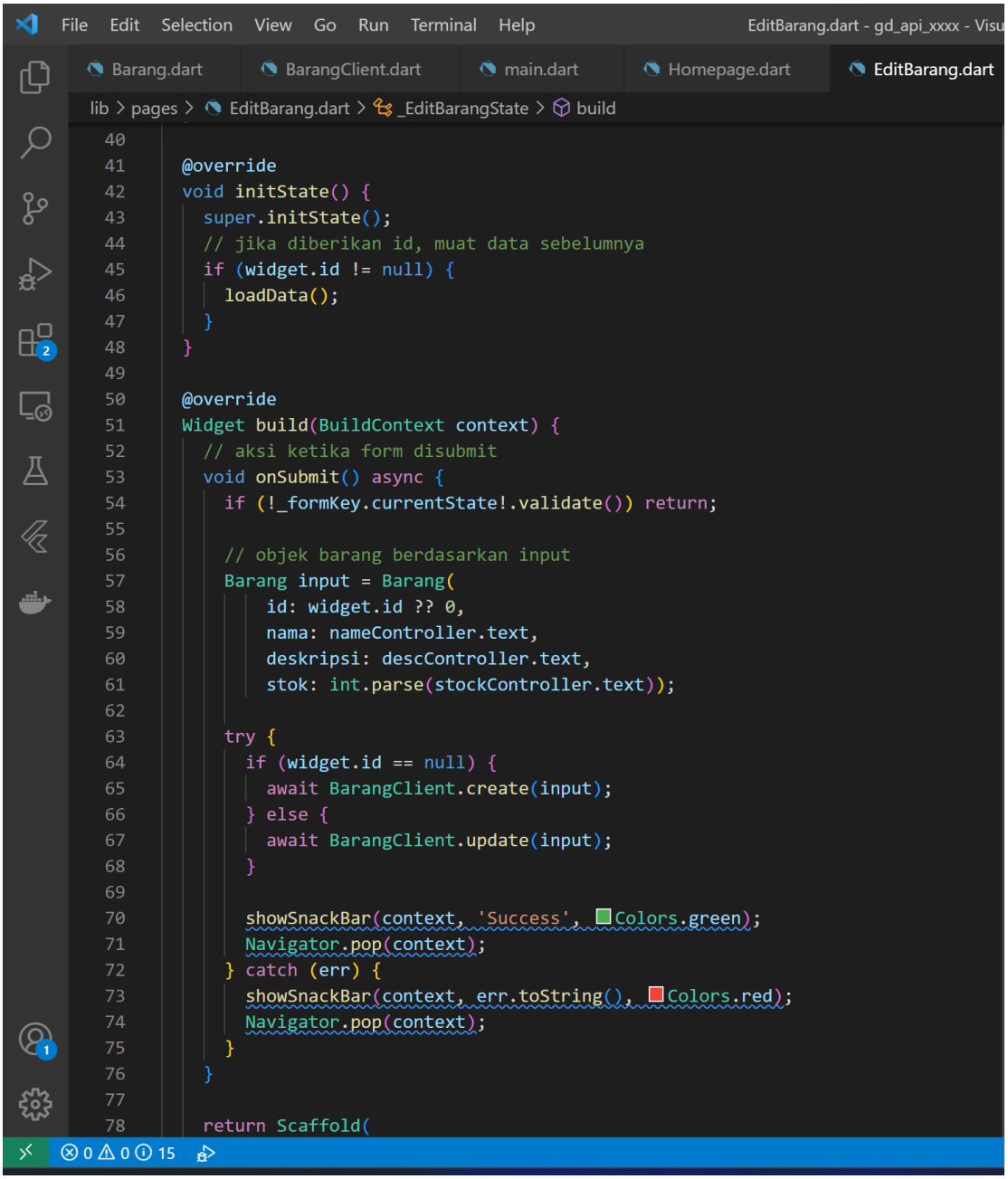


12. Terakhir buat file baru pada folder pages dengan nama EditBarang.dart

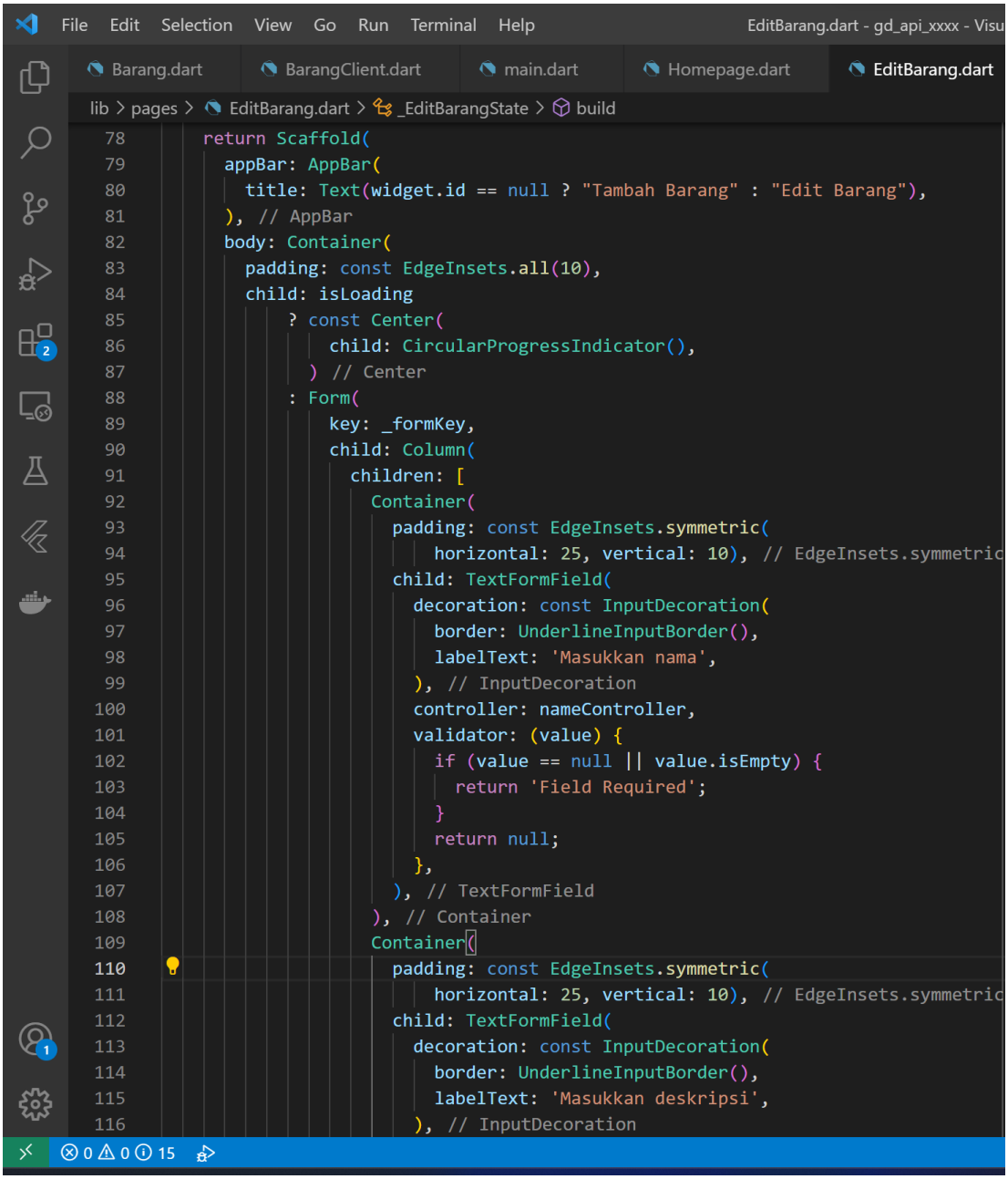


13. Halaman ini akan kita gunakan untuk tambah dan edit data sekaligus. Ubah isinya menjadi berikut:

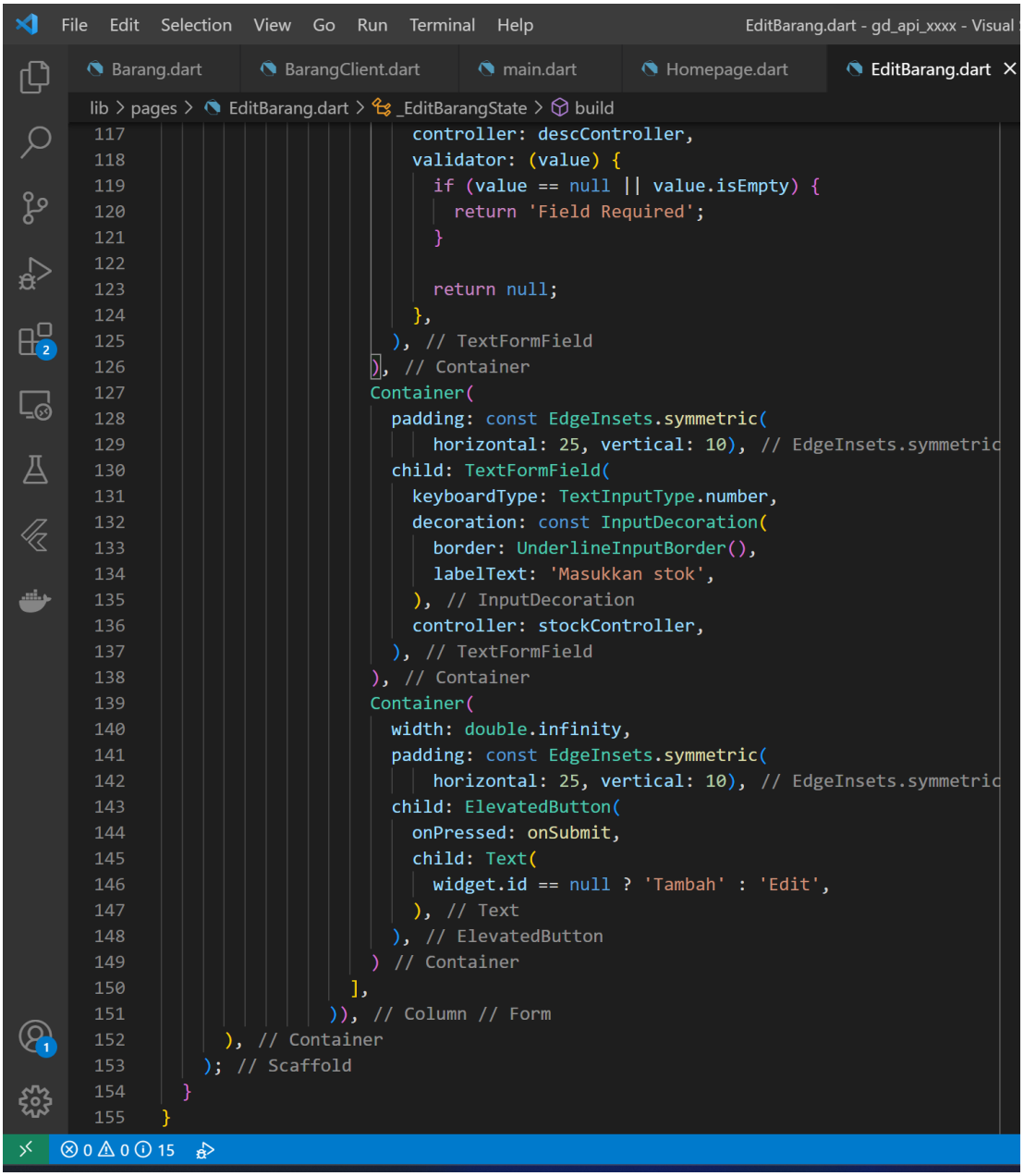



File Edit Selection View Go Run Terminal Help
EditBarang.dart - gd_api_xxxx - Visu
Barang.dart
BarangClient.dart
main.dart
Homepage.dart
EditBarang.dart
lib > pages > EditBarang.dart > _EditBarangState > build
@override
void initState() {
super.initState();
// jika diberikan id, muat data sebelumnya
if (widget.id != null) {
loadData();
}
}
@override
Widget build(BuildContext context) {
// 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);
}
}
return Scaffold(
⊗ 0 ⚠ 0 ⓘ 15

205
206
207


File Edit Selection View Go Run Terminal Help
EditBarang.dart - gd_api_xxxx - Visu
Barang.dart
BarangClient.dart
main.dart
Homepage.dart
EditBarang.dart
lib > pages > EditBarang.dart > _EditBarangState > build
return Scaffold(
appBar: AppBar(
title: Text(widget.id == null ? "Tambah Barang" : "Edit Barang"),
), // AppBar
body: Container(
padding: const EdgeInsets.all(10),
child: isLoading
? const Center(
child: CircularProgressIndicator(),
) // Center
: Form(
key: _formKey,
child: Column(
children: [
Container(
padding: const EdgeInsets.symmetric(
horizontal: 25, vertical: 10), // EdgeInsets.symmetric
child: TextFormField(
decoration: const InputDecoration(
border: UnderlineInputBorder(),
labelText: 'Masukkan nama',
), // InputDecoration
controller: nameController,
validator: (value) {
if (value == null || value.isEmpty) {
return 'Field Required';
}
return null;
},
), // TextFormField
), // Container
Container(
padding: const EdgeInsets.symmetric(
horizontal: 25, vertical: 10), // EdgeInsets.symmetric
child: TextFormField(
decoration: const InputDecoration(
border: UnderlineInputBorder(),
labelText: 'Masukkan deskripsi',
), // InputDecoration
0 0 15

208
209



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 meng*extends* 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: (barangs) => SingleChildScrollView(
    child: Column(
        children: barangs
            .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.

```dart
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.

```dart
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.

```dart
// 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);
  }
}
```

ATURAN PENGERJAAN GUIDED :

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