POLITEKNIK NEGERI JEMBER

Flutter Navigation, Layouts, Forms, and Input Elements

Tujuan Pembelajaran

Setelah menyelesaikan tutorial ini, mahasiswa diharapkan untuk dapat:

- Memahami navigasi dan *routing* dasar pada Flutter.
- Memahami elemen *input* dan *form* pada Flutter.
- Memahami alur pembuatan *form* dan data pada Flutter.
- Memahami dan menerapkan *clean architecture* sederhana

Navigasi Halaman pada Flutter

Pada saat belajar pengembangan *web*, kalian pasti sudah belajar bahwa dalam sebuah *website* kita dapat berpindah-pindah halaman sesuai dengan *URL* yang diakses. Hal yang sama juga berlaku pada pengembangan aplikasi, dimana kita juga dapat melakukan perpindahan dari satu 'halaman' ke 'halaman' yang lainnya. Bedanya, pada sebuah aplikasi, yang kita gunakan untuk berpindah bukanlah dengan mengakses *URL* yang berbeda.

Untuk mengimplementasikan navigasi pada Flutter, sebenarnya sudah disediakan sistem yang cukup lengkap untuk melakukan navigasi antar halaman. Salah satu cara yang dapat kita gunakan untuk berpindah-pindah halaman adalah dengan menggunakan *widget* Navigator. *Widget* Navigator menampilkan halaman-halaman yang ada kepada layar seakan sebagai sebuah tumpukan (*stack*). Untuk menavigasi sebuah halaman baru, kita dapat mengakses Navigator melalui BuildContext dan memanggil fungsi yang ada, seperti misalnya push(), pop(), serta pushReplacement().

Note: Di dalam Flutter, layar dan halaman seringkali disebut dengan terminologi *route*.

Berikut adalah penjelasan mengenai beberapa penggunaan Navigator yang paling sering dijumpai dalam pengembangan aplikasi.

Push (push())

```
...
    if (item.name == "Tambah Mood") {
        Navigator.push(context,
            MaterialPageRoute(builder: (context) => const
MoodEntryFormPage()));
    }
...
```

Method push() menambahkan suatu *route* ke dalam *stack route* yang dikelola oleh Navigator. Method ini menyebabkan *route* yang ditambahkan berada pada paling atas stack, sehingga *route* yang baru saja ditambahkan tersebut akan muncul dan ditampilkan kepada pengguna.

Pop (pop())

```
...
    onPressed: () {
        Navigator.pop(context);
    },
...
```

Method pop() menghapus *route* yang sedang ditampilkan kepada pengguna (atau dalam kata lain, *route* yang berada pada paling atas *stack*) dari *stack route* yang dikelola oleh Navigator. Method ini menyebabkan aplikasi untuk berpindah dari *route* yang sedang ditampilkan kepada pengguna ke *route* yang berada di bawahnya pada *stack* yang dikelola Navigator.

Push Replacement (pushReplacement())

```
...
    onTap: () {
        Navigator.pushReplacement(
```

```
        context,
        MaterialPageRoute(
            builder: (context) => MyHomePage(),
        ));
    },
...
```

Method pushReplacement() menghapus *route* yang sedang ditampilkan kepada pengguna dan menggantinya dengan suatu *route*. Method ini menyebabkan aplikasi untuk berpindah dari *route* yang sedang ditampilkan kepada pengguna ke suatu *route* yang diberikan. Pada *stack route* yang dikelola Navigator, *route* lama pada atas *stack* akan digantikan secara langsung oleh *route* baru yang diberikan tanpa mengubah kondisi elemen *stack* yang berada di bawahnya.

Walaupun push() dan pushReplacement() sekilas terlihat mirip, namun perbedaan kedua *method* tersebut terletak pada apa yang dilakukan kepada *route* yang berada pada atas *stack*. push() akan menambahkan *route* baru diatas *route* yang sudah ada pada atas *stack*, sedangkan pushReplacement() menggantikan *route* yang sudah ada pada atas *stack* dengan *route* baru tersebut. Penting juga untuk memperhatikan kemungkinan urutan dan isi dari *stack*, karena jika kondisi *stack* kosong serta kita menekan tombol **Back** pada gawai, maka sistem akan keluar dari aplikasi tersebut.

Di samping ketiga method Navigator di atas, terdapat juga beberapa method lain yang dapat memudahkan routing dalam pengembangan aplikasi, seperti popUntil(), canPop(), dan maybePop(). Silakan mengeksplorasi method-method tersebut secara mandiri. Untuk mengetahui lebih dalam terkait Navigator, kalian dapat membaca dokumentasi yang ada pada tautan berikut: (https://api.flutter.dev/flutter/widgets/Navigator-class.html)

Input dan Form Pada Flutter

Sama halnya dengan sebuah web, sebuah aplikasi juga dapat berinteraksi dengan pengguna melalui input dan form. Flutter memiliki widget Form yang dapat kita manfaatkan untuk menjadi wadah bagi beberapa *input field widget* yang kita buat. Sama halnya dengan *input field* pada web, Flutter juga memiliki banyak tipe *input field*, salah satunya widget TextField.

Untuk mencoba sampel dari widget Form, jalankan perintah berikut:
flutter create --sample=widgets.Form.1 form_sample

Untuk mengetahui lebih lanjut terkait widget Form, kalian dapat membaca dokumentasi pada tautan berikut: Flutter Form Class

Tutorial: Menambahkan Drawer Menu Untuk Navigasi

Untuk mempermudah navigasi di aplikasi Flutter kita, kita dapat menambahkan *drawer menu*. *Drawer menu* adalah sebuah menu yang muncul dari sisi kiri atau kanan layar. *Drawer menu* biasanya berisi navigasi ke halaman-halaman lain pada aplikasi.

Peringatan!
Pastikan kamu mengikuti tutorial dengan teliti. Perhatikan komentar TODO yang harus anda kerjakan pada kode program.

1. Buka proyek yang sebelumnya telah dibuat dengan menggunakan IDE favoritmu.
2. Buatlah direktori baru bernama widgets di subdirektori lib/. Kemudian, buat berkas dengan nama left_drawer.dart. Tambahkan kode berikut ke dalam berkas tersebut.

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

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

  @override
  Widget build(BuildContext context) {
```

```
    return Drawer(
      child: ListView(
        children: [
          DrawerHeader(
            // TODO: Bagian drawer header
          ),
            // TODO: Bagian routing
        ],
      ),
    );
  }
}
```

3. Berikutnya, tambahkan impor untuk halaman-halaman yang kita ingin masukkan navigasinya ke dalam Drawer Menu. Pada contoh ini, kita akan menambahkan navigasi ke halaman **MyHomePage** dan **MoodEntryFormPage**.

```
import 'package:flutter/material.dart';
import 'package:mental_health_tracker/menu.dart';
// TODO: Impor halaman MoodEntryFormPage jika sudah dibuat
```

Info!

Halaman MoodEntryFormPage akan dibuat pada langkah-langkah setelah ini

4. Setelah berhasil impor, kita akan memasukkan routing untuk halaman-halaman yang kita impor ke bagian TODO: Bagian routing. Hapus komentar tersebut dan isi dengan potongan kode berikut.

```
...
  ListTile(
    leading: const Icon(Icons.home_outlined),
    title: const Text('Halaman Utama'),
    // Bagian redirection ke MyHomePage
    onTap: () {
```

```
      Navigator.pushReplacement(
          context,
          MaterialPageRoute(
            builder: (context) => MyHomePage(),
          ));
    },
  ),
  ListTile(
    leading: const Icon(Icons.mood),
    title: const Text('Tambah Mood'),
    // Bagian redirection ke MoodEntryFormPage
    onTap: () {
      /*
      TODO: Buatlah routing ke MoodEntryFormPage di sini,
      setelah halaman MoodEntryFormPage sudah dibuat.
      */
    },
  ),
...
```

Apabila kalian *copy-paste* secara langsung, pastikan ellipsis (tanda "...") di atas dan di bawah kode tidak ikut ter-*copy*.

5. Selanjutnya, kita akan menghias drawer kita dengan memasukkan drawer header di TODO: Bagian drawer header. Hapus komentar tersebut dan ganti dengan potongan kode berikut.

```
...
    decoration: BoxDecoration(
      color: Theme.of(context).colorScheme.primary,
    ),
    child: const Column(
      children: [
        Text(
          'Mental Health Tracker',
          textAlign: TextAlign.center,
          style: TextStyle(
            fontSize: 24,
```

```
            fontWeight: FontWeight.bold,
            color: Colors.white,
          ),
        ),
        Padding(padding: EdgeInsets.all(8)),
        Text(
          "Ayo jaga kesehatan mentalmu setiap hari disini!",
          // TODO: Tambahkan gaya teks dengan center alignment, font
ukuran 15, warna putih, dan weight biasa
        ),
      ],
    ),
...
```

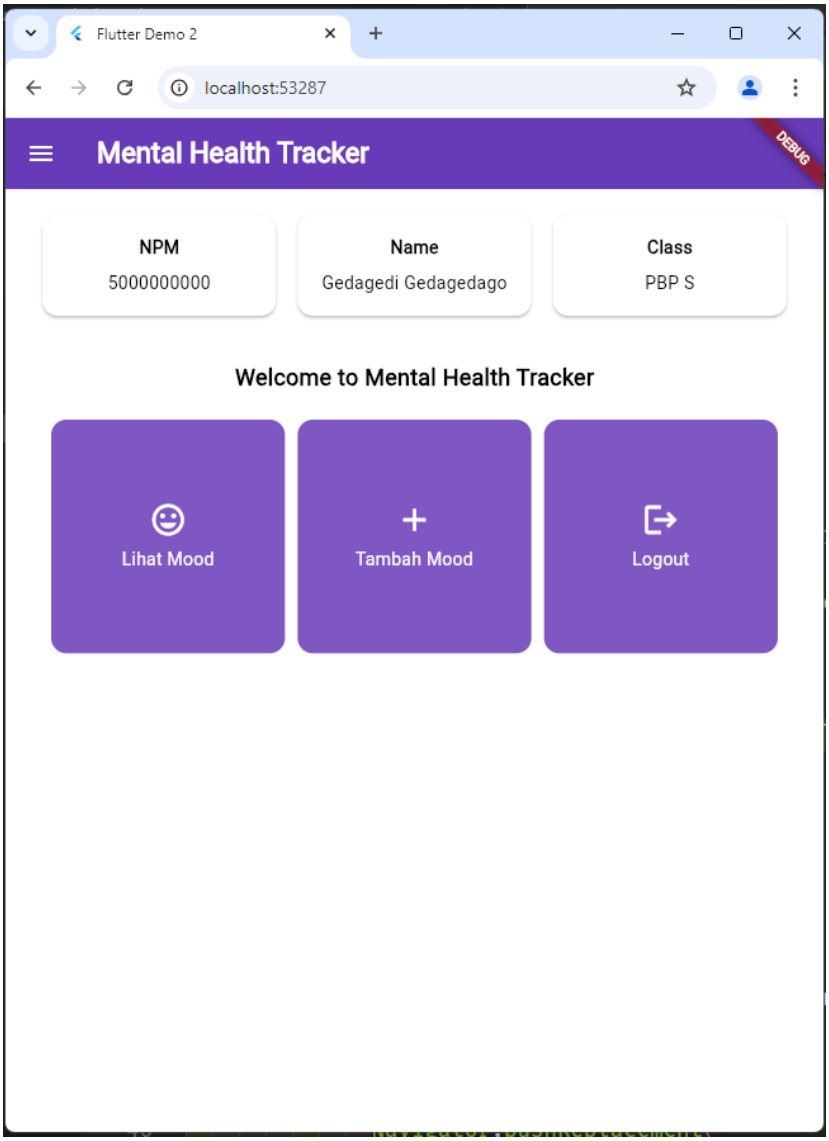
6. Selamat, kamu telah berhasil membuat *drawer menu*! Sekarang, masukkan drawer tersebut ke halaman yang ingin kamu tambahkan drawer. Untuk poin ini, kami akan berikan contoh untuk memasukkan ke halaman menu.dart.

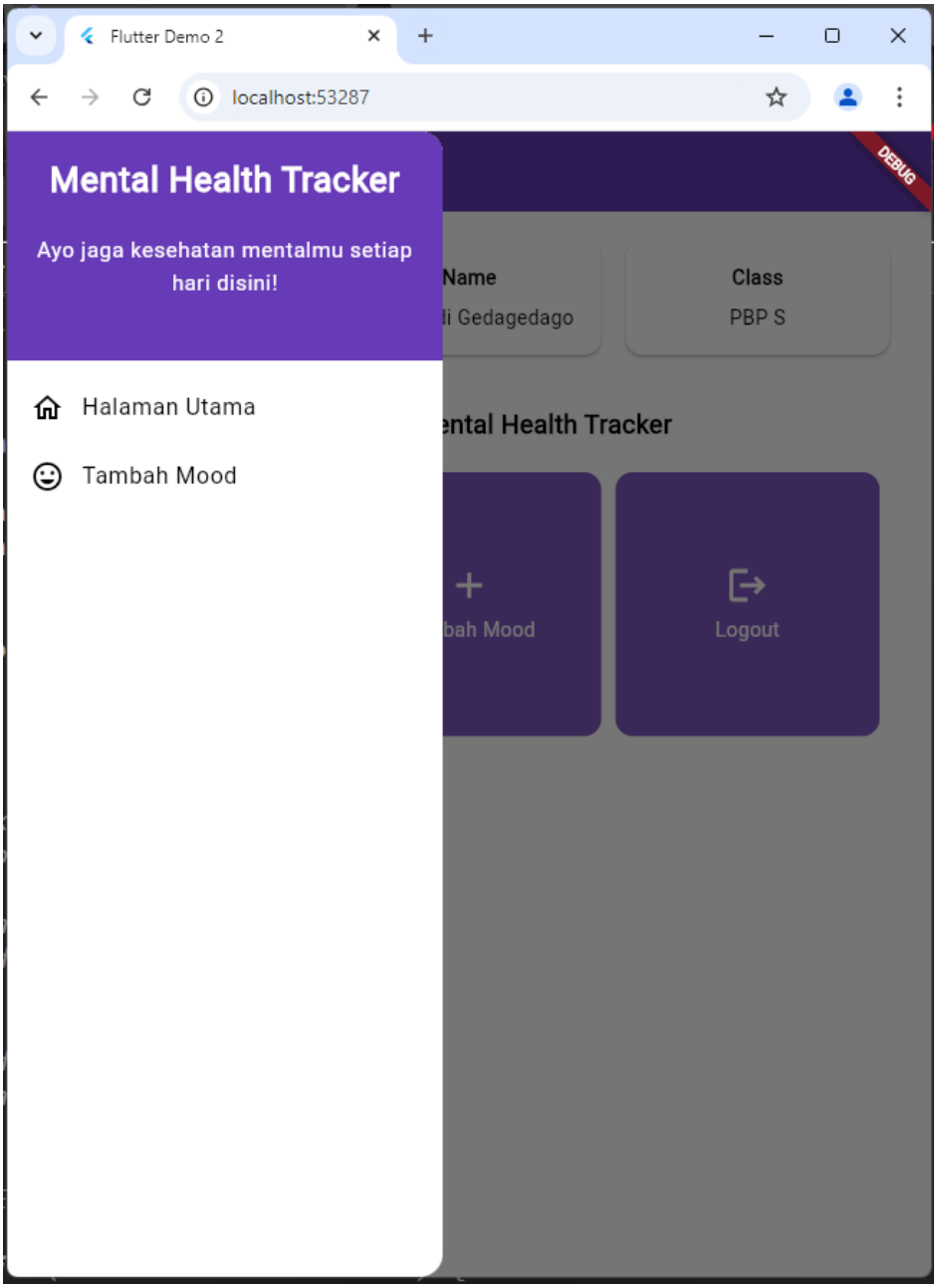
```
...
// Impor drawer widget
import 'package:mental_health_tracker/widgets/left_drawer.dart';
...
class MyHomePage extends StatelessWidget {
  ...

  @override
  Widget build(BuildContext context) {
    // Scaffold menyediakan struktur dasar halaman dengan appBar dan
body.
    return Scaffold(
      appBar: AppBar(
        ...
        // Mengganti warna icon drawer menjadi putih
        iconTheme: const IconThemeData(color: Colors.white),
      ),
      // Masukkan drawer sebagai parameter nilai drawer dari widget
```

```
Scaffold
      drawer: const LeftDrawer(),
      ...
    );
  }
}
...
```

7. Selamat, drawer dan navigasi sudah dibuat secara sempurna. Silakan jalankan program untuk melihat hasilnya. Jangan lupa kerjakan TODO yang masih ada **sebelum mengumpulkan tutorial** (tutorial yang dikumpulkan sudah **tidak memiliki** TODO). **Jangan lupa** juga tambahkan drawer ke halaman MoodEntryFormPage jika halaman tersebut sudah dibuat.


Flutter Demo 2
localhost:53287
Mental Health Tracker
DEBUG
NPM
5000000000
Name
Gedagedi Gedagedago
Class
PBP S
Welcome to Mental Health Tracker
Lihat Mood
Tambah Mood
Logout



Tutorial: Menambahkan Form dan Elemen Input

Sekarang, kita akan membuat sebuah form sederhana untuk memasukkan data barang pada aplikasi sehingga nantinya kamu dapat menambahkan data baru untuk ditampilkan.

1. Buat berkas baru pada direktori lib dengan nama moodentry_form.dart. Tambahkan kode berikut ke dalam berkas moodentry_form.dart.

```
import 'package:flutter/material.dart';
// TODO: Impor drawer yang sudah dibuat sebelumnya

class MoodEntryFormPage extends StatefulWidget {
  const MoodEntryFormPage({super.key});

  @override
  State<MoodEntryFormPage> createState() => _MoodEntryFormPageState();
}

class _MoodEntryFormPageState extends State<MoodEntryFormPage> {
  @override
  Widget build(BuildContext context) {
    return Placeholder();
  }
}
```

2. Ubah widget Placeholder dengan potongan kode berikut.

```
Scaffold(
  appBar: AppBar(
    title: const Center(
      child: Text(
        'Form Tambah Mood Kamu Hari ini',
      ),
    ),
    backgroundColor: Theme.of(context).colorScheme.primary,
    foregroundColor: Colors.white,
  ),
  // TODO: Tambahkan drawer yang sudah dibuat di sini
  body: Form(
    child: SingleChildScrollView(),
  ),
);
```

Penjelasan Kode:

i. Widget Form berfungsi sebagai wadah bagi beberapa *input field widget* yang nanti akan kita buat.

ii. Widget SingleChildScrollView berfungsi untuk membuat *child widget* di dalamnya menjadi *scrollable*.

3. Buat variabel baru bernama _formKey dengan nilai GlobalKey<FormState>(); lalu tambahkan _formKey tersebut ke dalam atribut key milik widget Form. Atribut key akan berfungsi sebagai handler dari form state, validasi form, dan penyimpanan form.

```
...
class _MoodEntryFormPageState extends State<MoodEntryFormPage> {
  final _formKey = GlobalKey<FormState>();
  ...
...
```

```
...
body: Form(
  key: _formKey,
  child: SingleChildScrollView(),
),
...
```

4. Selanjutnya, kita akan mulai mengisi widget Form dengan *field*. Buatlah beberapa variabel untuk menyimpan input dari masing-masing *field* yang akan dibuat.

```
...
class _MoodEntryFormPageState extends State<MoodEntryFormPage> {
       final _formKey = GlobalKey<FormState>();
       String _mood = "";
       String _feelings = "";
       int _moodIntensity = 0;
...
```

5. Buatlah *widget* Column sebagai *child* dari SingleChildScrollView.

```
...
body: Form(
       key: _formKey,
```

```
      child: SingleChildScrollView(
            child: Column()
      ),
...
```

6. Buatlah widget TextFormField yang di-wrap oleh Padding sebagai salah satu *children* dari widget Column. Setelah itu, tambahkan atribut crossAxisAlignment untuk mengatur alignment *children* dari Column.

```
...
child: Column(
  crossAxisAlignment: CrossAxisAlignment.start,
  children: [
    Padding(
      padding: const EdgeInsets.all(8.0),
      child: TextFormField(
        decoration: InputDecoration(
          hintText: "Mood",
          labelText: "Mood",
          border: OutlineInputBorder(
            borderRadius: BorderRadius.circular(5.0),
          ),
        ),
        onChanged: (String? value) {
          setState(() {
            _mood = value!;
          });
        },
        validator: (String? value) {
          if (value == null || value.isEmpty) {
            return "Mood tidak boleh kosong!";
          }
          return null;
        },
      ),
    ),
```

```
  ],
),
...
```

Penjelasan Kode:

i. onChanged akan dijalankan setiap ada perubahan isi TextFormField.

ii. validator akan melakukan validasi isi TextFormField dan mengembalikan String jika terdapat error.

iii. Terdapat implementasi null-safety pada bagian String? dan value!. Operator ? berfungsi untuk menandakan bahwa variabel tersebut boleh berisi String atau null. Sedangkan operator ! berfungsi untuk menandakan bahwa variabel tersebut dijamin akan tidak akan berisi null.

Untuk mempelajari lebih dalam mengenai null-safety, kalian dapat membaca dokumentasi yang ada pada tautan berikut: Dart Null Safety

7. Buatlah dua TextFormField yang di-*wrap* dengan Padding sebagai *child* selanjutnya dari Column seperti sebelumnya untuk field feelings dan mood intensity.

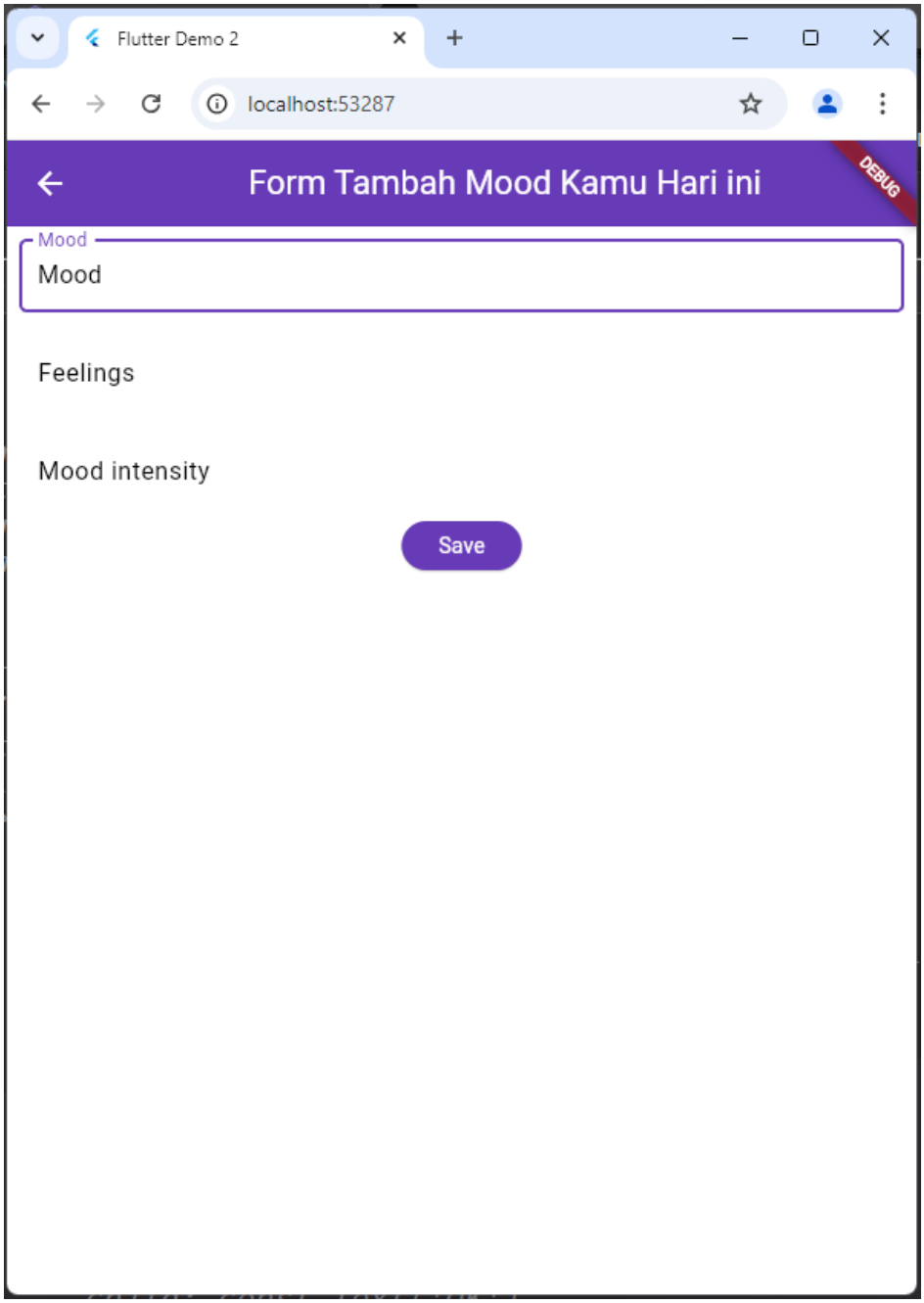
```
...
Padding(
  padding: const EdgeInsets.all(8.0),
  child: TextFormField(
    decoration: InputDecoration(
      hintText: "Feelings",
      labelText: "Feelings",
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(5.0),
      ),
    ),
    onChanged: (String? value) {
      setState(() {
        _feelings = value!;
      });
```

```
    },
    validator: (String? value) {
      if (value == null || value.isEmpty) {
        return "Feelings tidak boleh kosong!";
      }
      return null;
    },
  ),
),
Padding(
  padding: const EdgeInsets.all(8.0),
  child: TextFormField(
    decoration: InputDecoration(
      hintText: "Mood intensity",
      labelText: "Mood intensity",
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(5.0),
      ),
    ),
    onChanged: (String? value) {
      setState(() {
        _moodIntensity = int.tryParse(value!) ?? 0;
      });
    },
    validator: (String? value) {
      if (value == null || value.isEmpty) {
        return "Mood intensity tidak boleh kosong!";
      }
      if (int.tryParse(value) == null) {
        return "Mood intensity harus berupa angka!";
      }
      return null;
    },
  ),
),
...
```

8. Buatlah tombol sebagai *child* selanjutnya dari Column. Bungkus tombol ke dalam widget Padding dan Align. Kali ini kita belum menyimpan data ke dalam *database*, tetapi kita akan memunculkannya pada *pop-up* yang akan muncul setelah tombol ditekan.

```
...
Align(
  alignment: Alignment.bottomCenter,
  child: Padding(
    padding: const EdgeInsets.all(8.0),
    child: ElevatedButton(
      style: ButtonStyle(
        backgroundColor: WidgetStateProperty.all(
            Theme.of(context).colorScheme.primary),
      ),
      onPressed: () {
        if (_formKey.currentState!.validate()) {}
      },
      child: const Text(
        "Save",
        style: TextStyle(color: Colors.white),
      ),
    ),
  ),
),
...
```

9. Selamat! Sekarang *form* sudah dibuat secara sempurna. Silakan jalankan program untuk melihat hasilnya. Seharusnya tampilan form yang sudah dibuat akan seperti gamber berikut.



Tutorial: Memunculkan Data

1. Tambahkan fungsi showDialog() pada bagian onPressed() dari potongan kode yang sebelumnya kamu tambahkan. Munculkan widget AlertDialog pada fungsi tersebut. Kemudian, tambahkan juga fungsi untuk *reset* form. Kode kalian akan terlihat seperti ini.

```
...
child: ElevatedButton(
```

```
style: ButtonStyle(
  backgroundColor: MaterialStateProperty.all(
      Theme.of(context).colorScheme.primary),
),
onPressed: () {
  if (_formKey.currentState!.validate()) {
    showDialog(
      context: context,
      builder: (context) {
        return AlertDialog(
          title: const Text('Mood berhasil tersimpan'),
          content: SingleChildScrollView(
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text('Mood: $_mood'),
                // TODO: Munculkan value-value lainnya
              ],
            ),
          ),
          actions: [
            TextButton(
              child: const Text('OK'),
              onPressed: () {
                Navigator.pop(context);
                _formKey.currentState!.reset();
              },
            ),
          ],
        );
      },
    );
  }
},
child: const Text(
  "Save",
  style: TextStyle(color: Colors.white),
),
```

```
),
...
```

2. Silakan coba jalankan program kalian dan gunakan form yang telah dibuat, kemudian lihat hasilnya. **Jangan lupa untuk terlebih dahulu menambahkan routing pada drawer untuk bisa mengakses form yang telah kalian buat.**

Tutorial: Menambahkan Fitur Navigasi pada Tombol

Sampai sini, kita sudah berhasil membuat suatu *drawer* yang dapat menjalankan fitur navigasi ke halaman lain pada aplikasi, serta suatu halaman *form*. Pada tutorial sebelumnya, kita juga sudah berhasil membuat tiga *button widget* yang dapat melakukan *action* tertentu saat ia ditekan. Sekarang, kita akan menambahkan fitur navigasi pada tombol tersebut sehingga saat ditekan, pengguna akan ditampilkan halaman lain.

Peringatan!

Pastikan kamu mengikuti tutorial dengan teliti. Perhatikan komentar TODO yang harus anda kerjakan pada kode program.

1. Pada *widget* MoodItem pada berkas menu.dart yang sudah dibuat pada tutorial sebelumnya, akan dibuat agar kode yang terletak pada atribut onTap dari InkWell dapat melakukan navigasi ke *route* lain (tambahkan kode tambahan di bawah kode ScaffoldMessenger yang menampilkan *snackbar*).

```
...
// Area responsif terhadap sentuhan
onTap: () {
  // Memunculkan SnackBar ketika diklik
  ScaffoldMessenger.of(context)
    ..hideCurrentSnackBar()
    ..showSnackBar(SnackBar(
        content: Text("Kamu telah menekan tombol ${item.name}!")));
```

```
  // Navigate ke route yang sesuai (tergantung jenis tombol)
  if (item.name == "Tambah Mood") {
    // TODO: Gunakan Navigator.push untuk melakukan navigasi ke
MaterialPageRoute yang mencakup MoodEntryFormPage.
  }
},
...
```

Perhatikan bahwa pada tombol ini, kita menggunakan Navigator.push() sehingga pengguna dapat menekan tombol **Back** untuk kembali ke halaman menu. Selain itu, jika kita menggunakan Navigator.pop(), maka kita dapat membuat kode dalam program untuk kembali ke halaman menu.

2. Coba jalankan program kamu, gunakan tombol yang telah dibuat fungsionalitasnya, dan lihatlah apa yang terjadi. Bandingkan dengan apa yang terjadi jika kita melakukan navigasi melalui *drawer* (tentu saja setelah menyelesaikan seluruh TODO pada *drawer*). `

Tutorial: *Refactoring File*

Setelah membuat halaman moodentry_form.dart, halaman kita sudah semakin banyak. Dengan demikian, mari kita pindahkan halaman-halaman yang sudah dibuat sebelumnya ke dalam satu folder screens untuk mempermudah kita ke depannya.

1. Sebelum mulai, pastikan kamu sudah memiliki **ekstensi Flutter terinstal** di IDE atau *text editor* yang kamu gunakan.
2. Buatlah berkas baru dengan nama mood_card.dart pada direktori widgets.
3. Pindahkan isi widget ItemHomepage dan ItemCard pada menu.dart ke berkas widgets/mood_card.dart.
4. Pastikan untuk mengimpor halaman moodentry_form.dart pada berkas mood_card.dart dan import halaman mood_card.dart pada berkas menu.dart.

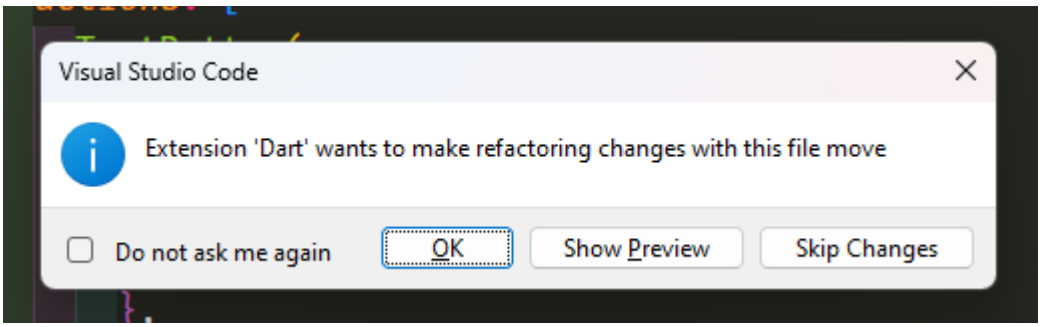
Info!

Anda dapat menghapus baris *import* moodentry_form.dart yang ada pada main.dart sebelumnya karena sudah tidak digunakan lagi

5. Buatlah folder baru bernama screens pada direktori lib.
6. Pindahkan file menu.dart dan moodentry_form.dart ke dalam folder screens.

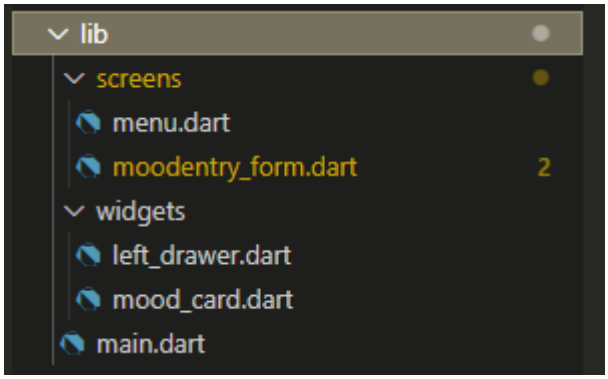
Peringatan!

Pastikan pemindahan file dilakukan **melalui IDE atau *text editor* yang memiliki ekstensi atau *plugin* Flutter**, bukan melalui *file manager* biasa (seperti File Explorer atau Finder). Hal ini dilakukan agar IDE atau *text editor* yang kamu gunakan dapat melakukan *refactoring* secara otomatis.



Jika muncul peringatan seperti gambar diatas, silahkan tekan **OK**

Setelah *refactoring file* dilakukan, seharusnya struktur dari direktori lib adalah seperti berikut.



Kesimpulan

Selamat! Kamu telah menyelesaikan modul ini! Semoga dengan tutorial ini, kalian dapat memahami mengenai *navigation*, *forms*, *input*, dan *layouts* dengan baik.

1. Pelajari dan pahami kembali kode yang sudah kamu tuliskan di atas dengan baik. **Jangan lupa untuk menyelesaikan semua TODO yang ada!**
2. Lakukan add, commit dan push untuk memperbarui repositori GitHub.

```
git add .
git commit -m "<pesan_commit>"
git push -u origin <branch_utama>
```

   - Ubah <pesan_commit> sesuai dengan keinginan. Contoh: git commit -m "modul flutter navigation selesai".
   - Ubah <branch_utama> sesuai dengan nama branch utamamu. Contoh: git push -u origin main atau git push -u origin master.

DAFTAR REFERENSI

1) https://docs.flutter.dev/ui/navigation
2) https://docs.flutter.dev/cookbook/navigation/navigation-basics
3) https://docs.flutter.dev/cookbook/design/drawer