Universitas	: Universitas Logistik dan Bisnis Internasional (ULBI)
Program Studi	: D4 Teknik Informatika
Semester	: 5 (Lima)
Mata Kuliah	: Pemrograman IV (Pemrograman Mobile)
Topik	: Praktikum 6 Membuat Input dan Validasi Widget

INPUT WIDGET

Salah satu bentuk interaksi dengan pengguna adalah dengan menerima input. Ada beberapa input widget yang bisa digunakan supaya pengguna bisa berinteraksi dengan aplikasi. Perhatikan bahwa input pengguna ini berkaitan dengan state yang dapat sering berubah. Karena itu umumnya input widget akan ditempatkan di dalam **StatefulWidget**.

Buat dahulu dahulu project bernama pertemuan6. Kemudian buat file Bernama **bottom_navbar.dart** dan copy paste kode berikut

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

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

  @override
  State<DynamicBottomNavbar> createState() => _DynamicBottomNavbarState();
}

class _DynamicBottomNavbarState extends State<DynamicBottomNavbar> {
  int _currentPageIndex = 0;

  final List<Widget> _pages = <Widget>[
    const MyInput(),
    const MyInput(),
  ];

  void onTabTapped(int index) {
    setState(() {
      _currentPageIndex = index;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: _pages[_currentPageIndex],
      bottomNavigationBar: BottomNavigationBar(
        currentIndex: _currentPageIndex,
        onTap: onTabTapped,
        items: const [
          BottomNavigationBarItem(
            icon: Icon(Icons.task_alt_outlined),
            label: 'Latihan',
```

```
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.input_outlined),
            label: 'Form Validation',
          ),
        ],
        backgroundColor: Colors.blueAccent,
        selectedItemColor: Colors.yellow,
        unselectedItemColor: Colors.white,
      ),
    );
  }
}
```

Pada **main.dart** ubah class MyApp menjadi seperti berikut

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        useMaterial3: false,
      ),
      home: DynamicBottomNavbar(),
    );
  }
}
```

Kemudian setelah itu lanjut ke bawah pada file main.dart.

A. TextField

TextField merupakan sebuah widget yang digunakan untuk menerima input berupa teks yang berasal dari keyboard. Terdapat beberapa cara yang bisa kalian gunakan untuk mendapatkan nilai dari TextField. Yang pertama dilakukan adalah membuat StatefulWidget terlebih dahulu

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

  @override
  State<MyInput> createState() => _MyInputState();
}

class _MyInputState extends State<MyInput> {
  @override
  Widget build(BuildContext context) {
    return const Placeholder();
  }
}
```

Cara yang bisa kita gunakan salah satunya adalah dengan TextEditingController. Dengan controller, kita cukup membuat variabel TextEditingController lalu menambahkannya ke widget **TextField**.

```
TextEditingController _controller = TextEditingController();

TextField(
  controller: _controller,
),
```

Ketika menggunakan controller, pastikan untuk menghapus controller ketika halaman atau widget sudah tidak digunakan. Ini bertujuan supaya tidak menimbulkan kebocoran memori (*memory leak*).

```
@override
void dispose() {
  _controller.dispose();
  super.dispose();
}
```

Dan berikut adalah contoh penggunaan TextEditingController secara lengkap

```
class _MyInputState extends State<MyInput> {
  TextEditingController _controller = TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Input Widget'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          children: [
            TextField(
              controller: _controller,
              decoration: const InputDecoration(
                hintText: 'Write your name here...',
                labelText: 'Your Name',
              ),
            ),
            const SizedBox(height: 20),
            ElevatedButton(
              child: const Text('Submit'),
              onPressed: () {
                showDialog(
                    context: context,
```

```
                    builder: (context) {
                      return AlertDialog(
                        content: Text('Hello, ${_controller.text}'),
                      );
                    });
              },
            )
          ],
        ),
      ),
    );
  }

  @override
  void dispose() {
    _controller.dispose();
    super.dispose();
  }
}
```

Berikut ini adalah contoh penerapan widget TextField:

HASIL DISINI

B. Form

Form adalah widget yang digunakan untuk membuat dan mengelola formulir (form) dalam aplikasi Flutter. Form memungkinkan kita untuk mengelola input pengguna, melakukan validasi, dan mengelola pengiriman data formulir. Form adalah bagian penting dalam memproses masukan dari pengguna dan memastikan bahwa masukan tersebut sesuai dengan syarat yang diinginkan sebelum mengirimkannya atau menyimpannya. Form juga memungkinkan kita untuk menangani berbagai tindakan seperti pengiriman data formulir, validasi, reset, dll.

```
//property
final GlobalKey<FormState> _formKey = GlobalKey<FormState>();

//build
Form(
  key: _formKey,
  child: InputWidget,
);
```

Pada kode yang di atas _formKey adalah sebuah objek GlobalKey<FormState> yang digunakan untuk mengidentifikasi dan mengelola form di dalam aplikasi kita. **GlobalKey**

adalah cara untuk memberikan referensi global ke suatu widget sehingga kita dapat mengakses dan mengontrolnya di dalam aplikasi kita. Pada kasus ini _formKey digunakan untuk mengakses dan mengelola state dari Form widget.

Untuk praktek penggunaan **Form** nanti kita coba pada sesi Praktikum.

C. Switch

Switch merupakan inputan yang mengembalikan nilai boolean true atau false. Untuk membuat switch, yang pertama kali dilakukan adalah membuat variabel **lightOn** diinisialisasi dengan nilai **false**.

```
bool lightOn = false;
```

Kemudian buat widget **Switch** di bawah widget **ElevatedButton**

```
Switch(
              value: lightOn,
              onChanged: (bool value) {
                setState(() {
                  lightOn = value;
                });

                ScaffoldMessenger.of(context).showSnackBar(
                  SnackBar(
                    content: Text(lightOn ? 'Light On' : 'Light Off'),
                    duration: Duration(seconds: 1),
                  ),
                );
              },
            ),
```

Pada contoh tersebut value dari Switch berupa boolean di mana ketika boolean tersebut false maka Switch akan berada pada posisi nonaktif. Switch umumnya digunakan sebagai konfigurasi on/off pada halaman setting.

Bisa dilihat widget Snackbar di atas digunakan untuk memberikan umpan balik singkat kepada pengguna setelah mereka mengganti status Switch. Penggunaan Snackbar ini merupakan cara yang baik untuk memberikan pesan sementara atau pemberitahuan singkat kepada pengguna tanpa mengganggu tampilan utama aplikasi.

Berikut ini adalah contoh penerapan widget Switch:

HASIL DISINI

D. Radio

Radio merupakan inputan yang digunakan untuk memilih salah satu dari beberapa pilihan dalam suatu kelompok. Yang pertama kali dilakukan adalah membuat variabel **language** dengan tipe data **String?**. Tipe data **String?** disebut sebagai “nullable string” atau "string yang dapat berisi nilai null." Artinya, variabel ini dapat memiliki nilai string (non-null) atau nilai null.

```
String? language;
```

Kemudian buat widget **Column** dan **ListTile** di bawah widget **Switch**

```
Column(
              mainAxisSize: MainAxisSize.min,
              children: [
                RadioListTile<String>(
                  title: const Text('Dart'),
                  value: 'Dart',
                  groupValue: language,
                  onChanged: (String? value) {
                    setState(() {
                      language = value;
                      showSnackbar();
                    });
                  },
                ),
                RadioListTile<String>(
                  title: const Text('Kotlin'),
                  value: 'Kotlin',
                  groupValue: language,
                  onChanged: (String? value) {
                    setState(() {
                      language = value;
                      showSnackbar();
                    });
                  },
                ),
                RadioListTile<String>(
                  title: const Text('Swift'),
                  value: 'Swift',
                  groupValue: language,
                  onChanged: (String? value) {
                    setState(() {
                      language = value;
                      showSnackbar();
                    });
                  },
                ),
```

```
            ],
          ),
```

Pada contoh tersebut terdapat variable language yang digunakan pada **groupValue** tiap Radio. Language inilah yang menyimpan nilai Radio yang dipilih. Nilainya akan berubah ketika fungsi onChanged terpanggil. Perhatikan pada bagian pemanggilan method atau function showSnackbar(); mengalami error.

Coba buat sebuah method atau function showSnackbar untuk menampilkan hasil yang dipilih dari widget Radio ! Sehingga hasilnya seperti berikut : DISINI

E. Checkbox

Checkbox merupakan inputan benar atau salah. Checkbox akan berisi centang jika nilainya adalah benar dan kosong jika salah, yang pertama kali dilakukan adalah membuat variabel **agree** diinisialisasi dengan nilai **false**.

```
bool agree = false;
```

Kemudian buat widget **ListTile** di bawah widget **Column** dari widget Radio yang sudah dibuat sebelumnya

```
CheckboxListTile(
                  title: const Text('Agree / Disagree'),
                  value: agree,
                  onChanged: (bool? value) {
                    setState(() {
                      agree = value!;
                    });
                  },
                  controlAffinity: ListTileControlAffinity.leading,
                ),
```

Coba buat Snackbar untuk menampilkan hasil yang dipilih dari widget Checkbox! Sehingga hasilnya seperti berikut : DISINI

Ada beberapa tautan yang dapat kalian baca untuk memahami tentang widget-widget input yang ada pada Flutter, antara lain:

- Input and selections widgets
- TextField Class

- Switch Class
- Radio Class
- Checkbox Class

PERCOBAAN PRAKTIKUM

1. Membuat Form dan Validasi

Saat ini kita akan coba membuat form untuk melakukan validasi terhadap form dan melakukan submit data. Yang pertama dilakukan adalah buat sebuah file baru di dalam folder lib dengan nama **input_validation.dart,** kemudian buat sebuah StatefulWidget dengan nama **MyFormValidation** dengan struktur **Scaffold** diikuti dengan Widget **Form** seperti berikut

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

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

  @override
  State<MyFormValidation> createState() => _MyFormValidationState();
}

class _MyFormValidationState extends State<MyFormValidation> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
          title: const Text('Form Validation'),
        ),
        body: Form(
          child: TextFormField(),
        ));
  }
}
```

Sekarang buka file **bottom_navbar.dart** kemudian lihat kode berikut

```
11  class _DynamicBottomNavbarState extends State<DynamicBottomNavbar> {
12    int _currentPageIndex = 0;
13
14    final List<Widget> _pages = <Widget>[
15      const MyInput(),
16      const MyInput(),
17    ]; // <Widget>[]
18
```

Ubah menjadi

```
12  class _DynamicBottomNavbarState extends State<DynamicBottomNavbar> {
13    int _currentPageIndex = 0;
14
15    final List<Widget> _pages = <Widget>[
16      const MyInput(),
17      const MyFormValidation(),
18    ]; // <Widget>[]
```

Jangan lupa untuk melakukan inisialisasi property GlobalKey pada **input_validation.dart** disimpan di atas **@override Widget build** karena kita akan menggunakan Form

```
final GlobalKey<FormState> _formKey = GlobalKey<FormState>();
```

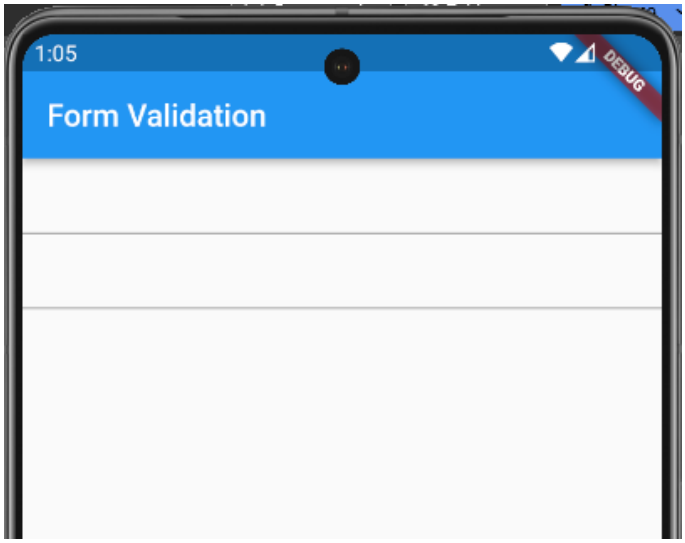
Lalu pada Form kita juga harus menambah parameter key dengan nilai yang sudah diinisialisasikan di atas yaitu **_formKey**

```
body: Form(
  key: _formKey,
  child: TextFormField(),
)); // Form // Scaffold
```

Selanjutnya adalah membungkus **TextFormField** kedalam widget **Column** karena pada kasus ini kita akan membuat lebih dari satu Form, dengan cara **klik widget TexFormField → refactor → Wrap with column.** Setelah itu tambah 1 widget TextFormField sehingga menjadi 2 buah widget TextFormField

```
body: Form(
  key: _formKey,
  child: Column(
    children: [
      TextFormField(),
      TextFormField(),
    ],
  ), // Column
)); // Form // Scaffold
```

Dari kode di atas kita berhasil membuat sebuah **Form**, namun butuh beberapa penyesuaian untuk tampilannya



Wrap widget **Form** ke dalam Padding dengan cara **refactor →Wrap with padding.** Sehingga seperti di bawah

```
body: Padding(
  padding: const EdgeInsets.all(16.0),
  child: Form(
    key: _formKey,
    child: Column(
      children: [
        TextFormField(),
        TextFormField(),
      ],
    ), // Column
  ), // Form
)); // Padding // Scaffold
```

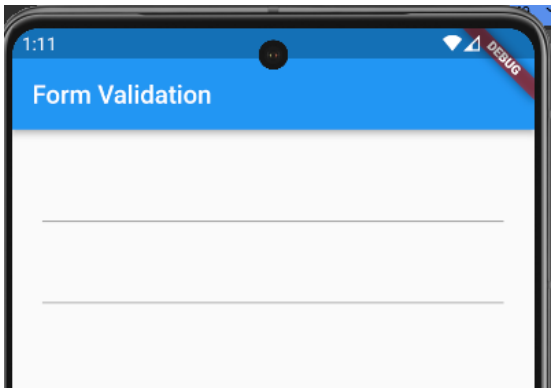
Dan juga untuk menghindari overflow kita juga akan tambahkan widget **SingleChildScrollView** agar halaman dapat di*scroll.* Dengan cara wrap widget **Padding** ke dalam SingeChildScrollView sehingga seperti di bawah

```
body: SingleChildScrollView(
  child: Padding(
    padding: const EdgeInsets.all(16.0),
    child: Form(
      key: _formKey,
      child: Column(
        children: [
          TextFormField(),
          TextFormField(),
        ],
      ), // Column
    ), // Form
  ), // Padding
)); // SingleChildScrollView // Scaffold
```

Dan juga untuk setiap widget TextFormField lakukan Wrap with padding sehingga tampak seperti berikut

```
child: Form(
  key: _formKey,
  child: Column(
    children: [
      Padding(
        padding: const EdgeInsets.all(8.0),
        child: TextFormField(),
      ), // Padding
      Padding(
        padding: const EdgeInsets.all(8.0),
        child: TextFormField(),
      ), // Padding
    ],
  ), // Column
), // Form
```

Sekarang Form kita sudah memiliki padding



Langkah selanjutnya adalah membuat insialisasi controller

```
TextEditingController _controllerEmail = TextEditingController();
TextEditingController _controllerNama = TextEditingController();
```

Jangan lupa membuat method dispose

```
@override
  void dispose() {
    _controllerEmail.dispose();
    _controllerNama.dispose();
    super.dispose();
  }
```

Sehingga seperti berikut

```
class _MyFormValidationState extends State<MyFormValidation> {
  final GlobalKey<FormState> _formKey = GlobalKey<FormState>();
  TextEditingController _controllerEmail = TextEditingController();
  TextEditingController _controllerNama = TextEditingController();

  @override
  void dispose() {
    _controllerEmail.dispose();
    _controllerNama.dispose();
    super.dispose();
  }
```

Dan jangan lupa controller yang dibuat dipanggil pada kedua form dengan menggunakan parameter controller

```
children: [
  Padding(
    padding: const EdgeInsets.all(8.0),
    child: TextFormField(
      controller: _controllerEmail,
    ), // TextFormField
  ), // Padding
  Padding(
    padding: const EdgeInsets.all(8.0),
    child: TextFormField(
      controller: _controllerNama,
    ), // TextFormField
  ), // Padding
],
```

Jika sudah, kita coba buat decoration pada form pertama (Email) di bawah **controller**

```
decoration: const InputDecoration(
                        hintText: 'Write your email here...',
                        labelText: 'Email',
                        border: OutlineInputBorder(
                          borderRadius: BorderRadius.all(
                            Radius.circular(10),
                          ),
                        ),
                        fillColor: Color.fromARGB(255, 222, 254, 255),
                        filled: true,
                      ),
```

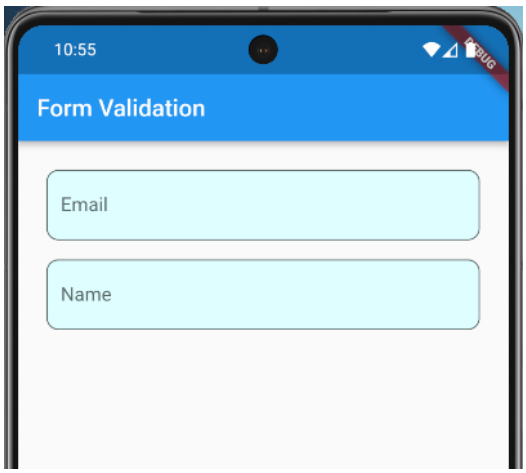
Decoration pada form kedua (Nama) masukkan di bawah **controller**

```
decoration: const InputDecoration(
                        hintText: 'Write your name here...',
                        labelText: 'Name',
                        border: OutlineInputBorder(
                          borderRadius: BorderRadius.all(
                            Radius.circular(10),
                          ),
                        ),
                        fillColor: Color.fromARGB(255, 222, 254, 255),
                        filled: true,
                      ),
```

Untuk kode lengkapnya seperti berikut

```
children: [
  Padding(
    padding: const EdgeInsets.all(8.0),
    child: TextFormField(
      controller: _controllerEmail,
      decoration: const InputDecoration(
        hintText: 'Write your email here...',
        labelText: 'Email',
        border: OutlineInputBorder(
          borderRadius: BorderRadius.all(
            Radius.circular(10),
          ), // BorderRadius.all
        ), // OutlineInputBorder
        fillColor: Color.fromARGB(255, 255, 249, 222),
        filled: true,
      ), // InputDecoration
    ), // TextFormField
  ), // Padding
  Padding(
    padding: const EdgeInsets.all(8.0),
    child: TextFormField(
      controller: _controllerNama,
      decoration: const InputDecoration(
        hintText: 'Write your name here...',
        labelText: 'Name',
        border: OutlineInputBorder(
          borderRadius: BorderRadius.all(
            Radius.circular(10),
          ), // BorderRadius.all
        ), // OutlineInputBorder
        fillColor: Color.fromARGB(255, 255, 249, 222),
        filled: true,
      ), // InputDecoration
    ), // TextFormField
  ), // Padding
],
```

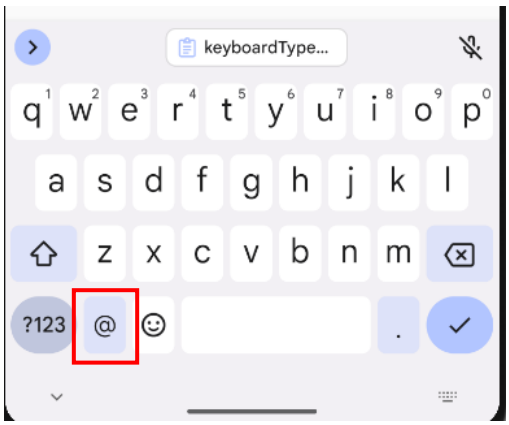
hintText teks yang ditampilkan di dalam input field saat tidak ada teks yang dimasukkan. **labelText** adalah teks yang berfungsi sebagai label di atas input field. Label ini memberikan deskripsi atau identifikasi dari input field tersebut. **border** adalah properti yang mengatur jenis bingkai (border) yang mengelilingi input field. Dalam kasus ini, kita menggunakan OutlineInputBorder yang memberikan garis bingkai luar dengan sudut bulat. Kita juga mengatur radius sudutnya dengan borderRadius. **fillColor** ini mengatur warna latar belakang dari input field saat dalam keadaan aktif (diisi). Dalam kode tersebut, warna latar belakangnya adalah warna kuning muda dengan menggunakan kode warna ARGB. **filled** adalah boolean yang menentukan apakah input field akan diisi dengan warna latar belakang (seperti yang ditentukan dalam fillColor) atau tidak saat dalam keadaan aktif. Jika diatur sebagai true, maka input field akan diisi dengan warna latar belakang yang telah ditentukan. Sehingga ketika running akan seperti berikut



Kemudian pada TextFormField email tambahkan parameter keyboardType di bawah controller

```
keyboardType: TextInputType.emailAddress,
```

Sehingga ketika kalian mengklik field email maka keyboard akan memunculkan tanda @



Selanjutnya adalah bagaimana membuat validasi pada kedua form tersebut yang disimpan di bawah method **dispose**, yang pertama kita akan melakukan validasi form email wajib diisi. Kita dapat membuat method baru dengan nama **_validateEmail**

```
String? _validateEmail(String? value) {
    if (value!.isEmpty) {
      return 'Email wajib diisi';
    }

    return null;
  }
```

Kode _validateEmail di atas adalah sebuah fungsi validasi dalam Flutter yang memeriksa apakah sebuah input email tidak kosong. Jika input email kosong, maka fungsi ini mengembalikan pesan kesalahan 'Email wajib diisi', sehingga pengguna tahu bahwa kolom email harus diisi sebelum melanjutkan. Jika input email sudah diisi, maka fungsi ini mengembalikan null, menandakan validasi berhasil dan tidak ada pesan kesalahan yang ditampilkan.

Kemudian jangan lupa tambahkan validasi format email, disini kita coba menggunakan string Regex sehingga method _validate email menjadi seperti berikut

```
String? _validateEmail(String? value) {
    const String expression = "[a-zA-Z0-9+._%-+]{1,256}"
        "\\@"
        "[a-zA-Z0-9][a-zA-Z0-9\\-]{0,64}"
        "("
        "\\."
        "[a-zA-Z0-9][a-zA-Z0-9\\-]{0,25}"
        ")+";
    final RegExp regExp = RegExp(expression);

    if (value!.isEmpty) {
      return 'Email wajib diisi';
    }

    if (!regExp.hasMatch(value)) {
      return "Tolong inputkan email yang valid!";
    }
    return null;
  }
```

Langkah selanjutnya adalah membuat validasi pada form nama dengan ketentuan minimal karakter yang diisi sepanjang 3 huruf. Kita buat kembali method baru dengan nama **_validateNama**

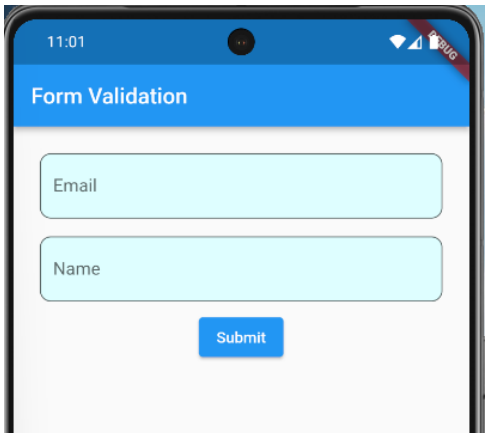
```
String? _validateNama(String? value) {
    if (value!.length < 3) {
      return 'Masukkan setidaknya 3 karakter';
    }
    return null;
  }
```

Selanjutnya adalah memanggil method yang sudah dibuat pada Form, yaitu dengan cara membuat parameter **validator** di dalam widget **TextFormField** sebagai berikut

```
children: [
  Padding(
    padding: const EdgeInsets.all(8.0),
    child: TextFormField(
      controller: _controllerEmail,
      keyboardType: TextInputType.emailAddress,
      validator: _validateEmail,
      decoration: const InputDecoration( // InputDecoration ···
    ), // TextFormField
  ), // Padding
  Padding(
    padding: const EdgeInsets.all(8.0),
    child: TextFormField(
      controller: _controllerNama,
      validator: _validateNama,
      decoration: const InputDecoration( // InputDecoration ···
    ), // TextFormField
  ), // Padding
],
```

Untuk melakukan validasi setidaknya kita butuh sebuah tombol untuk melakukan validasi kedua form di atas. Jadi langkah selanjutnya adalah membuat sebuah widget Button di bawah dari Padding TextFormField sebagai berikut

```
ElevatedButton(
                    child: const Text("Submit"),
                    onPressed: () {},
                  ),
```



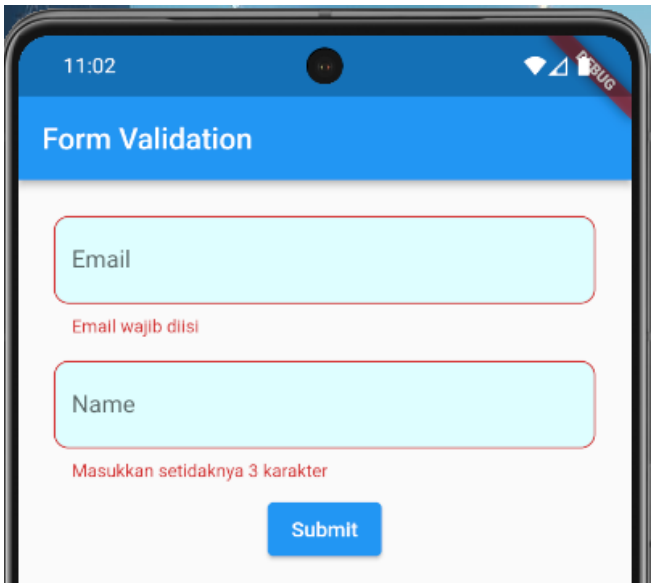
Pada parameter onPressed buatlah kondisi seperti berikut

```
onPressed: () {
                      if (_formKey.currentState!.validate()) {

                      }else{

                      }
                    },
```

Dengan kode di atas sebenarnya kita sudah dapat melakukan validasi ketika tombol Submit ditekan.

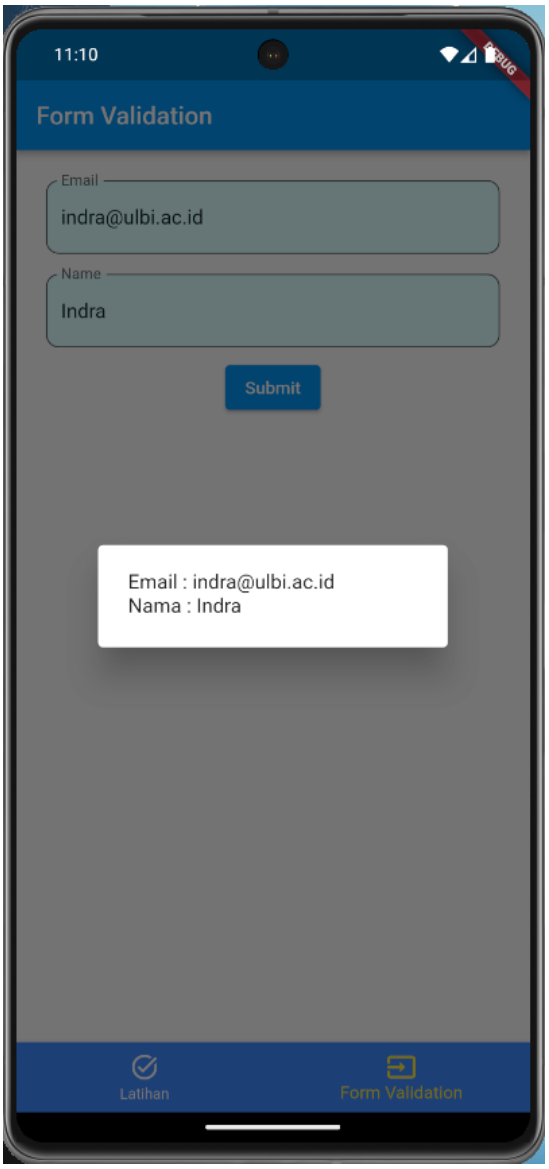


Namun disini kita coba buat Snackbar pada kondisi if dan else seperti berikut

```
onPressed: () {
                      if (_formKey.currentState!.validate()) {
                        ScaffoldMessenger.of(context).showSnackBar(
                          const SnackBar(
                            content: Text('Processing Data'),
                            duration: Duration(seconds: 2),
                          ),
                        );
                      } else {
                        ScaffoldMessenger.of(context).showSnackBar(
                          const SnackBar(
                            content: Text('Please complete the form'),
                          ),
                        );
                      }
                    },
```

Coba jalankan aplikasi. Sampai sini kita berhasil membuat validasi menggunakan widget Form.

Tapi sepertinya kita belum menampilkan data yang diinput pada AlertDialog. Sekarang coba kalian buat sebuah Dialog untuk menampilkan data yang diinput seperti di bawah (clue : menggunakan AlertDialog dan menggunakan Column)



2. **Membuat Insert yang disimpan pada State Widget menggunakan Map**

Masih pada project yang sama. Yang perlu dilakukan adalah membuat file dart baru pada folder lib dengan nama **input_form.dart,** kemudian **copy** semua code yang ada pada **input_validation.dart** dan **paste** pada **input_form.dart**

Ubah juga nama class yang awalnya adalah MyFormValidation menjadi **MyInputForm**

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

  @override
  State<MyInputForm> createState() => _MyInputFormState();
}

class _MyInputFormState extends State<MyInputForm> {
```

Ubah judul pada AppBar menjadi Form Input

```
title: const Text('Form Input'),
```

Kemudian ke file **bottom_navbar.dart** tambahkan class MyInputForm

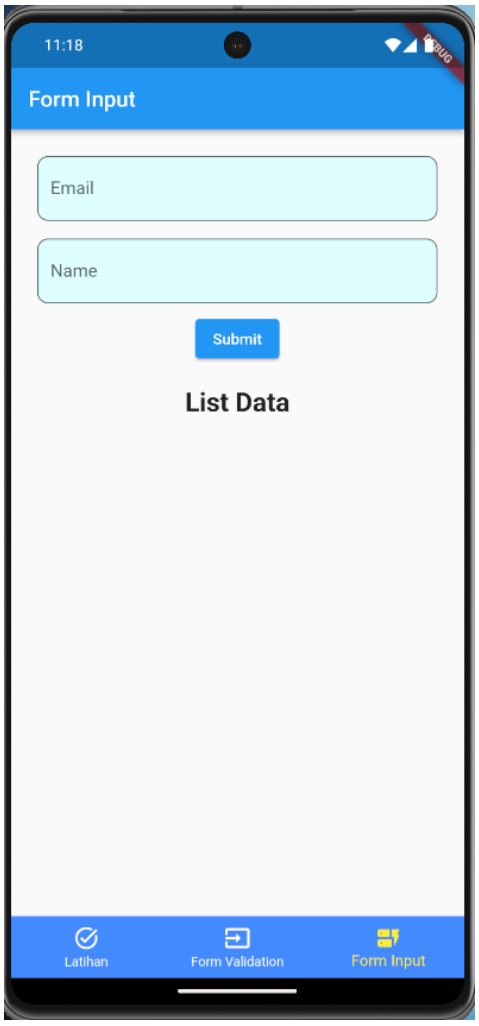
```
13  class _DynamicBottomNavbarState extends State<DynamicBottomNavbar> {
14    int _currentPageIndex = 0;
15
16    final List<Widget> _pages = <Widget>[
17      const MyInput(),
18      const MyFormValidation(),
19      const MyInputForm(),
20    ]; // <Widget>[]
```

Kemudian tambah widget BottomNavigationBarItem

```
29    Widget build(BuildContext context) {
30      return Scaffold(
31        body: _pages[_currentPageIndex],
32        bottomNavigationBar: BottomNavigationBar(
33          currentIndex: _currentPageIndex,
34          onTap: onTabTapped,
35          items: const [
36            BottomNavigationBarItem(
37              icon: Icon(Icons.task_alt_outlined),
38              label: 'Latihan',
39            ), // BottomNavigationBarItem
40            BottomNavigationBarItem(
41              icon: Icon(Icons.input_outlined),
42              label: 'Form Validation',
43            ), // BottomNavigationBarItem
44            BottomNavigationBarItem(
45              icon: Icon(Icons.dynamic_form_rounded),
46              label: 'Form Input',
47            ), // BottomNavigationBarItem
48          ],
49          backgroundColor: Colors.blueAccent,
50          selectedItemColor: Colors.yellow,
51          unselectedItemColor: Colors.white,
52        ), // BottomNavigationBar
53      ); // Scaffold
54    }
55  }
```

Kembali ke **input_form.dart** kemudian tambah beberapa widget berikut yang nanti digunakan untuk menampilkan data di bawah Widget ElevatedButton

```
const SizedBox(height: 20),
                  const Center(
                    child: Text(
                      'List Data',
                      style:
                          TextStyle(fontSize: 24, fontWeight:
FontWeight.bold),
                    ),
                  ),
```



Selanjutnya adalah membuat deklarasi Map _myDataList untuk menampung data

```
final List<Map<String, dynamic>> _myDataList = [];
```

Nanti data yang diinput akan tersimpan sementara pada _myDataList disimpan pada state _MyInputFormState. Selanjutnya adalah kita akan membuat sebuah ListView builder untuk menampilkan data yang disimpan, buat di bawah widget **Center** yang dibuat sebelumnya.

```
ListView.builder(
                    shrinkWrap: true,
                    itemCount: _myDataList.length,
                    itemBuilder: (context, index) {
                      final data = _myDataList[index];
                      return Padding(
                        padding: const EdgeInsets.all(8.0),
                        child: Row(
                          children: [
                            CircleAvatar(
                              backgroundColor: Colors.grey,
                              child: Text(
                                'ULBI',
                                style: TextStyle(
                                  fontWeight: FontWeight.bold,
                                ),
                              ),
                            ),
                            SizedBox(width: 10),
                            Expanded(
                              child: Column(
                                crossAxisAlignment: CrossAxisAlignment.start,
                                children: [
                                  Text(data['name'] ?? ''),
                                  Text(data['email'] ?? ''),
                                ],
                              ),
                            ),
                          ],
                        ),
                      );
                    },
                  ),
```

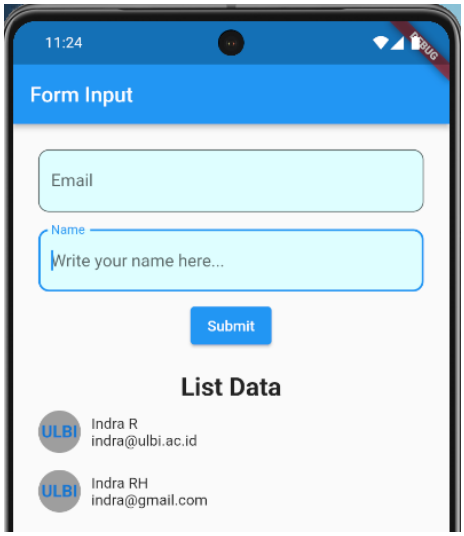
Kemudian tambah sebuah method **_addData** untuk menambah data ke dalam Map _myDataList

```
void _addData() {
    final data = {
      'name': _controllerNama.text,
      'email': _controllerEmail.text,
    };
    setState(() {
      _myDataList.add(data);
      _controllerNama.clear();
      _controllerEmail.clear();
    });
  }
```

Dan ubah kode yang ada pada **ElevatedButton** pada parameter **onPressed** menjadi sepeti di bawah

```
ElevatedButton(
                    child: const Text("Submit"),
                    onPressed: () {
                      if (_formKey.currentState!.validate()) {
                        _addData();
                      }
                    },
                  ),
```

Running aplikasi dan coba submit data



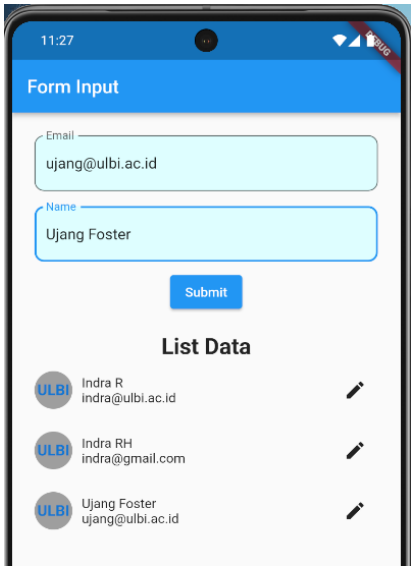
Kemudian sekarang kita coba melakukan update data dengan cara buat sebuah method dengan nama**_editData**

```
void _editData(Map<String, dynamic> data) {
    setState(() {
      _controllerEmail.text = data['email'];
      _controllerNama.text = data['name'];
    });
  }
```

Kemudian tambah IconButton edit pada ListView builder di bawah Widget **Expanded**

```
IconButton(
                              onPressed: () {
                                setState(() {
                                  _editData(data);
                                });
                              },
                              icon: const Icon(Icons.edit),
                            ),
```

Jalankan aplikasi, ketika kita klik button edit kita sudah mendapatkan value dari data yang dipilih namun ketika klik button submit data malah bertambah bukannya terupdate



Sebelum itu kalian lihat jika terdapat banyak data pada ListView, halaman yang discroll adalah semua halaman dikarenakan kita menggunakan **SingleChildScrollView**

LIHAT DISINI

Kita akan membuat agar ListView saja yang dapat kita scroll. Caranya adalah pada body widget **SingleChildScrollView klik kanan → Refactor → Remove this widget**

```
72    @override
73    Widget build(BuildContext context) {
74      return Scaffold(
75          appBar: AppBar(
76            title: const Text('Form Input'),
77          ), // AppBar
78          body: SingleChildScrollView(
79            child: Padding(
80              padding: const EdgeInsets.all(16.0),
```

Jika sudah pada **widget ListView.builder klik kanan→ Refactor → Wrap with widget → Isi dengan Expanded.** Sekarang seharusnya yang dapat kita scroll adalah ListView saja

LIHAT DISINI

Selanjutnya untuk dapat melakukan update selanjutnya adalah kita inisialisasi variabel editedData menggunakan Map.

```
Map<String, dynamic>? editedData;
```

Variabel di atas yang digunakan untuk menyimpan data yang sedang diedit. Cara di atas digunakan untuk menunjukkan apakah kita sedang dalam mode edit atau mode penambahan data baru. Jika editedData tidak null, maka kita sedang dalam mode edit, dan kita akan mengubah data yang ada dalam daftar dengan data yang diedit.

Kemudian edit method **_addData()** pada bagian **setState** menjadi seperti ini

```
void _addData() {
    final data = {
      'name': _controllerNama.text,
      'email': _controllerEmail.text,
    };
    setState(() {
      if (editedData != null) {
        // Jika editedData ada, maka kita sedang dalam mode edit
        // Sehingga kita perlu memperbarui data yang sedang diedit
        editedData!['name'] = data['name'];
        editedData!['email'] = data['email'];
        // Kosongkan kembali editedData setelah proses edit selesai
        editedData = null;
      } else {
        // Jika editedData kosong, maka kita sedang dalam mode insert
        _myDataList.add(data);
      }
      _controllerNama.clear();
      _controllerEmail.clear();
    });
  }
```

Dan tambah satu baris kode pada method _editData() menjadi

```
void _editData(Map<String, dynamic> data) {
    setState(() {
      _controllerEmail.text = data['email'];
      _controllerNama.text = data['name'];
      editedData = data;
    });
  }
```

Tambah juga sedikit interface pada button, ketika kondisi edit maka teks yang ditampilkan adalah ‘Update’

```
ElevatedButton(
                    child: Text(editedData != null ? "Update" : "Submit"),
                    onPressed: () {
                      if (_formKey.currentState!.validate()) {
                        setState(() {
                          _addData();
                        });
                      }
                    },
                  ),
```

Sekarang jalankan aplikasi, seharusnya sekarang dapat melakukan update data.

Selanjutnya adalah delete, tambahkan sebuah **IconButton** di bawah IconButton Edit untuk menghapus data sebagai berikut

```
IconButton(
                                onPressed: () {
                                  setState(() {
                                    _myDataList.remove(data);
                                  });
                                },
                                icon: const Icon(Icons.delete),
                              ),
```

Coba jalankan aplikasi dan klik button hapus.

Namun bagaimana jika kita ingin membuat sebuah Dialog jika mengharuskan konfirmasi untuk melakukan penghapusan data. Untuk membuatnya kita buat sebuah method dengan nama **_deleteData.**

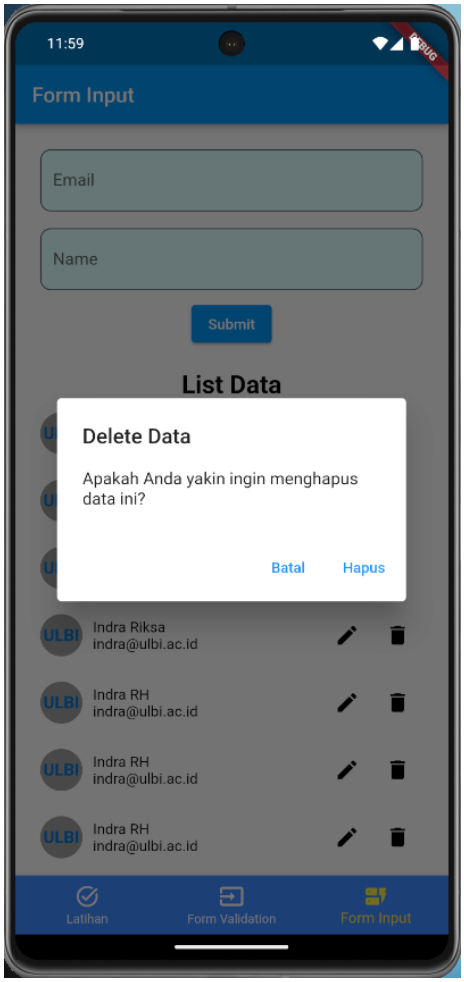
```
void _deleteData(Map<String, dynamic> data) {
    showDialog(
      context: context,
      builder: (context) {
        return AlertDialog(
          title: const Text('Delete Data'),
          content: const Text('Apakah Anda yakin ingin menghapus data ini?'),
          actions: [
            TextButton(
              onPressed: () {
                Navigator.of(context).pop();
              },
              child: const Text('Batal'),
            ),
            TextButton(
              onPressed: () {
                setState(() {
                  _myDataList.remove(data);
                });
                Navigator.of(context).pop();
              },
              child: const Text('Hapus'),
            ),
          ],
        );
      },
    );
  }
```

Kode di atas adalah implementasi fungsi **_deleteData** yang menampilkan dialog konfirmasi penghapusan data saat pengguna mengklik tombol “Hapus”. Dialog ini berisi pesan konfirmasi, dengan pilihan “Batal” untuk membatalkan penghapusan dan “Hapus” untuk menghapus data. Jika pengguna memilih “Hapus” maka data yang diberikan sebagai parameter (`data`) akan dihapus dari daftar data `_myDataList`, dan tampilan akan diperbarui menggunakan `setState`.

Dan jangan lupa pada IconButton hapus bagian setState panggil method _deleteData yang sudah dibuat

```
IconButton(
                                onPressed: () {
                                  setState(() {
                                    _deleteData(data);
                                  });
                                },
                                icon: const Icon(Icons.delete),
                              ),
```

Coba jalankan aplikasi, jika menekan tombol hapus sekarang terdapat konfirmasi



Pengumpulan Hasil :

Jika sudah berhasil, hasil praktikum dimasukkan ke dalam github masing-masing. Buat folder dengan nama **Pertemuan06** kemudian di dalamnya buat 2 folder yaitu **folder Praktikum** dan **Output**.

Isi folder **Praktikum** adalah aplikasi flutter yang sudah dibuat, tetapi **sebelum melakukan push ke github** wajib jalankan perintah "**flutter clean"** pada VSCode. Isi folder Output adalah hasil dari aplikasi flutter yang sudah dibuat wajib format gif.

Setelah melakukan perintah **flutter clean** kalian bisa langsung push ke dalam repository kalian.