

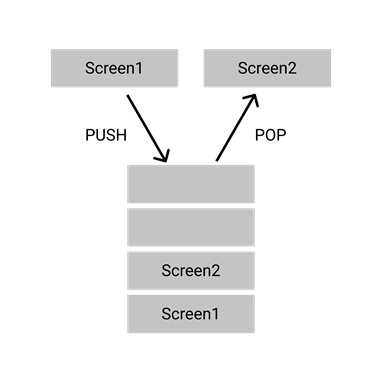
**TUJUAN PRAKTIKUM**

* Memperkenalkan apa itu navigation.
* Melakukan passing data saat pindah halaman.

**ALOKASI WAKTU :** 1 x Pertemuan = 120 Menit

**DASAR TEORI**

**Navigation** memiliki fungsi untuk menentukan bagaimana alur atau mekanisme peralihan dari suatu halaman ke halaman lainnya. Misalnya dari splash screen ke halaman login, dari halaman login ke home page, dst. Navigasi pada Flutter yakni bahwa ketika berpindah halaman/activity akan menjadi tumpukan (stack). Jadi ketika berpindah dari satu halaman ke halaman lain (push), maka halaman pertama akan ditumpuk oleh halaman kedua. Kemudian apabila kembali dari halaman kedua ke pertama, maka halaman kedua akan dihapus (pop).



**Gambar 4.1** Konsep Stack pada Navigasi

1. **Navigator.push()**

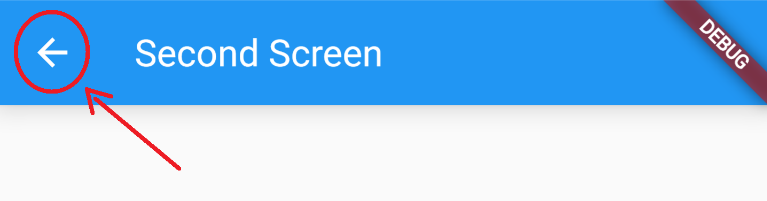
Untuk melakukan perpindahan halaman dapat menggunakan sebuah metode Navigator.push(), berikut contoh kodenya:

| **Contoh Program 4.1** Contoh Navigator Push |
| --- |
| ElevatedButton(  child: Text('Pindah Screen'),  onPressed: () {  Navigator.*push*(context, MaterialPageRoute(builder: (context) {  return SecondScreen();  }));  }, ), |

Pada kode di atas Navigator.push() memiliki dua parameter. Pertama ialah context dan yang kedua Route. Parameter context ini merupakan variabel BuildContext yang ada pada method build. Parameter route berguna untuk menentukan tujuan ke mana akan berpindah halaman. Route tersebut diisi dengan MaterialPageRoute yang di dalamnya terdapat builder yang nantinya akan diisi dengan tujuan halamannya. Untuk melakukan perpindahan halaman tersebut sebelumnya perlu membuat event onPressed pada sebuah tombol yang ada di halaman pertama.

**Gambar 4.2** Navigasi Push

Pada versi Flutter terbaru terdapat sedikit perbedaan dimana ketika melakukan perpindahan halaman menggunakan Navigator.push() akan secara otomatis menampilkan tombol kembali di kiri atas pada halaman kedua aplikasi.



**Gambar 4.3** Tombol Kembali

Jika tidak ingin menampilkan tombol tersebut, maka dapat dihilangkan dengan cara cukup menambahkan kode berikut didalam Scaffold appbar yang digunakan.

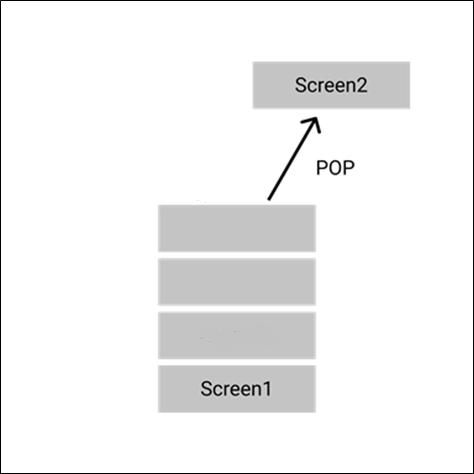
| **Contoh Program 4.2** Menghilangkan Tombol Kembali |
| --- |
| Scaffold(  appBar: AppBar(  title: Text('Second Screen'),  **automaticallyImplyLeading: false,**  ), ) |

1. **Navigator.pop()**

Setelah dapat berpindah ke halaman lain dan ingin kembali ke halaman sebelumnya, maka dapat menggunakan Navigator.pop(). Penulisan Navigator.pop seperti berikut:

| **Contoh Program 4.3** Contoh Navigator Pop |
| --- |
| OutlinedButton(  child: Text('Kembali'),  onPressed: () {  Navigator.*pop*(context);  }, ), |

Pada Navigator.pop hanya cukup menambahkan parameter context yang merupakan variabel dari method build. Sama seperti sebelumnya untuk kembali dari halaman kedua perlu menambahkan event onPressed pada tombol yang ada di halaman kedua.



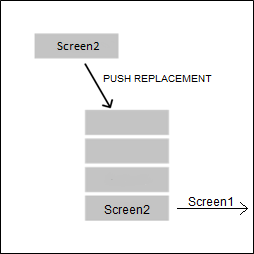
**Gambar 4.4** Navigasi Pop

1. **Navigator.pushReplacement()**

Selain menambahkan stack untuk berpindah halaman, terdapat cara lain yaitu dengan mengganti/replace halaman yang sedang aktif menggunakan method Navigator.pushReplacement(). Berikut contoh penulisan kodenya:

| **Contoh Program 4.4** Contoh Navigator Push Replacement |
| --- |
| ElevatedButton(  child: Text('Pindah Screen tanpa Stack'),  onPressed: () {  Navigator. *pushReplacement*(  context,  MaterialPageRoute(builder: (context) {  return SecondScreen();  }  ));  }, ), |

Pada kode Navigator.pushReplacement() sama seperti Navigator.push() yang memiliki dua parameter. Fungsi kedua parameter tersebut juga sama dengan metode sebelumnya. Dan untuk berpindahan halaman tersebut juga perlu membuat event onPressed pada tombol yang ada di halaman pertama. Perbedaan yang mencolok pada metode Navigator.push() adalah setelah berhasil pindah halaman maka pada halaman kedua tidak akan bisa kembali ke halaman pertama, karena halaman pertama sudah hilang atau tergantikan oleh halaman kedua.



**Gambar 4.5** Navigasi Push Replacement

1. **Passing data**

Seringkali beberapa halaman pada aplikasi perlu saling berinteraksi dengan berbagi dan saling mengirimkan data. Pada Flutter dapat memanfaatkan constructor dari sebuah class untuk mengirimkan data antar halaman. Penerapan ini sama seperti konsep dari OOP yang telah dipelajari semester lalu. Berikut contoh kode yang dilakukan untuk mengirimkan pesan dari First Screen menuju Second Screen:

Pada First Screen disiapkan terlebih dahulu variabel yang akan dikirim kepada Secong Screen yaitu variabel pesanKirim yang berisi string “Hello from First Screen!”.

| **Contoh Program 4.5** Menyiapkan Variabel Pesan |
| --- |
| class FirstScreen extends StatelessWidget {  final String pesanKirim = 'Hello from First Screen!';  .  .  . } |

Selanjutnya pada kelas FirstScreen dapat memasukkan variabel pesanKirim tersebut saat akan melakukan push atau berpindah halaman ke SecondScreen di dalam fungsi onPressed pada tombol yang diteka, maka pesan dikirimkan sebagai parameter dari constructor kelas SecondScreen. Kodenya seperti berikut:

| **Contoh Program 4.6** NavigasiMengirim Data |
| --- |
| onPressed: () {  Navigator.*push*(context, MaterialPageRoute(builder: (context) {  return **SecondScreen(pesan: pesanKirim);**  })); }, |

Agar SecondScreen bisa menerima data tersebut, maka perlu mengubah default constructor-nya dan menambahkan variabel untuk menampung datanya. Berikut kodenya:

| **Contoh Program 4.7** Menerima Data |
| --- |
| class SecondScreen extends StatelessWidget {  **final String pesan;**   SecondScreen**({Key? key, required this.pesan})** : super(key: key); } |

Jika dijabarkan pada pengiriman data dari FirstScreen yaitu ‘pesan: pesanKirim’ bermakna bahwa kode ‘pesan: …’ menandakan variabel yang dibutuhkan dari SecondScreen, dan kode ‘… pesanKirim’ menandakan variabel String yang dikirimkan dari FirstScreen. Terakhir data pesan tersebut dapat langsung ditampilkan pada halaman kedua seperti berikut:

| **Contoh Program 4.8** Menampilkan Data Yang Diterima |
| --- |
| class SecondScreen extends StatelessWidget {  final String pesan;   SecondScreen({Key? key, required this.pesan}) : super(key: key);   @override  Widget build(BuildContext context) {  return Scaffold(  appBar: AppBar(  title: Text('Second Screen'),  ),  body: Center(  child: **Text(pesan),**  ),  );  } } |

Selain itu, perlu diketahui bahwa data yang dikirimkan bisa banyak atau lebih dari satu data dan tentunya bisa berbagai tipe data seperti String, Integer, Double, bahkan Object. Kemudian dalam melakukan pengiriman data antar halaman terdapat dua tipe parameter yang digunakan yaitu sebagai parameter required dan parameter optional, hal ini perlu diperhatikan saat mengembangkan aplikasi. Kedua tipe ini memungkinkan digunakan bersamaan dalam satu proses pengiriman data antar halaman.

* 1. **Parameter required**

Parameter required dilakukan seperti contoh kode sebelumnya yang dimana data dari halaman kedua (SecondScreen) wajib dikirimkan oleh halaman pertama (FirstScreen) apabila tidak mengirimkan data tersebut maka akan terjadi error aplikasi. Untuk implementasinya dapat dengan cukup menambahkan kode ‘required’ pada parameter constructor di halaman tujuan (SecondScreen), berikut contohnya.

| **Contoh Program 4.9** Parameter Required |
| --- |
| class SecondScreen extends StatelessWidget {  final String pesan;  final String pesanDua;   SecondScreen({  Key? key,  **required** this.pesan,  **required** this.pesanDua  }) : super(key: key); } |

* 1. **Parameter optional**

Berbeda dengan parameter required, parameter optional ini tidak mengharuskan halaman asal untuk mengirimkan data yang dibutuhkan dari halaman tujuan sehingga data tersebut bersifat opsional. Untuk implementasinya cukup dengan menghapus kode ‘required’ pada parameter constructor di halaman tujuan. Karena data yang dikirim tidak wajib maka data tersebut memungkinkan bernilai kosong (tidak bernilai), dalam pemrograman Flutter data yang tidak bernilai tersebut akan menjadi error, sehingga untuk mengatasi hal tersebut perlu membuat default value dari parameter yang dibutuhkan oleh halaman tujuan (SecondScreen). Berikut contoh penggunaannya.

| **Contoh Program 4.10** Parameter Optional |
| --- |
| class SecondScreen extends StatelessWidget {  final String pesan;  final String pesanDua;   SecondScreen({  Key? key,  this.pesan = **"Selamat Datang",**  this.pesanDua = **"Ini bagian nilai default"**  }) : super(key: key); } |

1. **Launch Url**

Dalam melakukan perpindahan halaman, terkadang aplikasi juga perlu melakukan perpindahan ke halaman yang berada diluar aplikasi. Contoh nya di dalam aplikasi ingin melakukan perpindahan ke profil Instagram tertentu, atau ingin melakukan Dial ke nomor tertentu, dan masih banyak lagi tujuan lainnya. Perpindahan tersebut dapat dilakukan dengan memberikan link Url tertentu, seperti Instagram maka link yang perlu dijalankan yaitu https://www.instagram.com, dan lainnya.

Pada flutter, untuk melakukan aktivitas tersebut perlu menambahkan dependency/library ‘url\_launcher’. Selanjutnya dengan mudah dapat membuat fungsi \_launchURL berikut yang akan dipanggil pada aksi tombol tertentu bagian ‘onPress’, dan untuk memanggil fungsi tersebut perlu mengirimkan sebuah paramater String berupa url yang ingin dituju.

| **Contoh Program 4.11** Launch Url |
| --- |
| Future<void> \_launchURL(String url) async {  if (await canLaunch(url)) {  await launch(url);  } else {  throw 'Could not launch $url';  }  } |

1. **Callback**

Ketika sedang membuat project aplikasi yang cukup besar sering terjadi pengulangan kode yang digunakan, contohnya ketika membuat sebuah Widget tertentu seperti tombol Login dan tombol Register dimana kode tombol tersebut dibuat berulang yang membedakannya hanya teks button dan fungsi onPress pada tombol tersebut yaitu tombol Login berfungsi melakukan validasi inputan, sedangkan Register pindah halaman form registrasi. Bayangkan apabila aplikasi besar seperti Gojek atau Tokopedia tentunya banyak sekali tombol yang mirip, hal tersebut membuat kode yang dibangun akan semakin panjang karena banyak menulis kode yang berulang dan semakin tidak rapi.

Ketika membuat sebuah widget sebagai template tentunya ada suatu hal yang akan berbeda ketika widget tersebut dipanggil, sebagai contoh seperti sebelumnya fungsi Login dengan fungsi Register pasti berbeda, dengan menggunakan Callback fungsi tersebut dapat dikirim secara dinamis dan tentunya tidak terbatas oleh parameter yang digunakan serta return value yang diinginkan. Agar tidak semakin bingung, mari buat contoh kode tersebut.

Sebagai contoh disini akan membuat sebuah fitur login yang hanya terdapat form password serta tombol login dan tombol register. Sebelumnya siapkan terlebih dahulu tampilan aplikasi tersebut.

| **Contoh Program 4.12** Tampilan Contoh Aplikasi Callback |
| --- |
| import 'package:flutter/material.dart'; import 'package:flutter/widgets.dart'; class ContohCallback extends StatefulWidget {  const ContohCallback({Key? key}) : super(key: key);  @override  \_ContohCallbackState createState() => \_ContohCallbackState(); } class \_ContohCallbackState extends State<ContohCallback> {  String \_password = "";  @override  Widget build(BuildContext context) {  return Scaffold(  appBar: AppBar(  title: Text('Callback'),  ),  body: Padding(  padding: const EdgeInsets.all(18.0),  child: Column(  children: [  TextFormField(  decoration: InputDecoration(labelText: "Password"),  onChanged: (value) {  setState(() {  \_password = value;  });  },  ),  SizedBox(height: 24),  ElevatedButton(  onPressed: () {},  child: Text("Login"),  style: ElevatedButton.*styleFrom*(  minimumSize: Size(160, 40),  ),  ),  SizedBox(height: 12),  ElevatedButton(  onPressed: () {},  child: Text("Register"),  style: ElevatedButton.*styleFrom*(  minimumSize: Size(160, 40),  ),  ),  ],  ),  ),  );  } } |

Kemudian kedua tombol tersebut dapat dipisahkan menjadi satu template tombol yang umum. Caranya dengan membuat file baru yang menampung style tombol tersebut sebagai contoh buat file common\_button.dart, dan isi file tersebut adalah style widget button yang digunakan. Copy style button yang ada di file utama ke dalam file baru.

| **Contoh Program 4.13** File common\_button.dart |
| --- |
| import 'package:flutter/material.dart'; class CommonButton extends StatelessWidget {  const CommonButton({Key? key}) : super(key: key);  @override  Widget build(BuildContext context) {  return ElevatedButton(  onPressed: () {},  child: Text("Login"),  style: ElevatedButton.*styleFrom*(  minimumSize: Size(160, 40),  ),  );  } } |

Selanjutnya karena di tombol tersebut membutuhkan data dinamis teks dan fungsi onPressed maka kelas CommonButton membutuhkan passing data saat kelasnya dipanggil. Fungsi yang diterima oleh CommonButton inilah yang disebut sebagai Callback. Cara mengirim fungsinya yaitu seperti berikut.

| **Contoh Program 4.14** File common\_button.dart |
| --- |
| class CommonButton extends StatelessWidget {  **final String label;  *// ini deklarasi fungsi callback* final Function(String) callback;**    const CommonButton({  Key? key,  **required this.label,  required this.callback,**  }) : super(key: key);  @override  Widget build(BuildContext context) {  return ElevatedButton(  **onPressed: () {  callback("ini data callback dari $label");  },  child: Text(label),**  style: ElevatedButton.*styleFrom*(  minimumSize: Size(160, 40),  ),  );  } } |

Jika dijabarkan pada fungsi callback yaitu menggunakan tipe data Function sebagai fungsi yang memiliki return value, selain tipe Function terdapat juga tipe Void yang tidak memiliki return value, selanjutanya di dalam tipe Function terdapat parameter yang di dalam kurung yaitu String, ini merupakan nilai parameter yang akan dikembalikan kepada kelas yang memanggil CommonButton tersebut, selayaknya fungsi biasa parameter tersebut boleh lebih dari satu ataupun kosong, namun tipe data parameter harus sesuai. Terakhir callback merupakan nama variabel dari fungsi tersebut. Selanjutnya tinggal dimasukkan dalam fungsi onPressed pada tombol.

Lalu cara untuk menggunakan tombol tersebut yaitu seperti berikut. Ubah isi widget Column menjadi seperti ini.

| **Contoh Program 4.15** Menggunakan Callback |
| --- |
| . . .  Column(  children: [  TextFormField(  decoration: InputDecoration(labelText: "Password"),  onChanged: (value) {  setState(() {  \_password = value;  });  },  ),  SizedBox(height: 24),  **CommonButton(  label: "Login",  callback: (value) {  var text = \_password == "12345678"  ? "$value: Login Berhasil"  : "$value: Login Gagal";  ScaffoldMessenger.*of*(context).showSnackBar(SnackBar(  content: Text(text),  ));  },  ),**  SizedBox(height: 12),  **CommonButton(  label: "Register",  callback: (value) {  var text = "$value: Password Tersimpan";  ScaffoldMessenger.*of*(context).showSnackBar(SnackBar(  content: Text(text),  ));  },  ),**  ], ),  . . . |

Aplikasi sudah dapat dijalankan dan amatilah cara penggunaan callback. Pada cara sebelumnya tombol dibuat menggunakan ElevatedButton dimana terdapat pengaturan style secara kasar tertulis berulang antara kedua tombol tersebut, namun sekarang dengan menggunakan CommonButton tidak perlu menuliskan style yang sama berulang, disini hanya perlu menuliskan kode yang benar-benar dibutuhkan secara berbeda dari tiap tombol. Dengan cara ini aplikasi besar yang dibangun akan lebih efektif dan rapi.

**LANGKAH PRAKTIKUM**

Pada bagian langkah praktikum ini akan melanjutkan project flutter yang sebelumnya yaitu pada Modul III membuat halaman login yang interaktif. Disini akan membuat sebuah fungsi login berpindah ke halaman utama dengan mengirimkan data username yang dipakai. Pada halaman utama berisi sebuah input form untuk mengisi data pribadi user, setelah diisi maka dapat berpindah halaman dengan mengirimkan data tersebut ke dalam halaman profil yang lebih rapi. Selanjutnya pada halaman profil terdapat tombol kembali dan tombol logout.

1. Buka project flutter sebelumnya pada Modul III.
2. Pertama ubahlah terlebih dahulu kode yang ada di LoginPage pada proses tombol login dengan menghapus syarat username yang digunakan, karena disini hanya akan menggunakan password sebagai autentikasi login agar username dapat secara dinamis dikirim ke halaman utama. Dan berikan navigasi ketika proses login sukses dengan mengirimkan data username ke halaman utama. Agar user tidak dapat kembali ke halaman login dengan mudah maka navigasi menggunakan metode pushReplacement.

| **Langkah Praktikum 4.1** Menghapus Autentikasi Username di LoginPage |
| --- |
| Widget \_loginButton(BuildContext context) {  return Container(  .  .  .  onPressed: () {  print("$isLoginSuccess");   String text = "";  if (**password == "flutter123"**) {  setState(() {  text = "Login Success";  isLoginSuccess = true;  });  **Navigator.pushReplacement(context,  MaterialPageRoute(builder: (context) {  return HomePage(username: username);  }));**  } else {  setState(() {  text = "Login Failed";  isLoginSuccess = false;  });  }  .   .   .  ); } |

1. Ketika kode tersebut dibuat akan terjadi error karena disini belum membuat file sebagai halaman utama, buatlah file bernama home\_page.dart kemudian buat kelas HomePage di dalam file tersebut serta jangan lupa untuk menambahkan variabel username karena pada halaman login mengirimkan data berupa username, sehingga perlu diterima oleh halaman utama.

| **Langkah Praktikum 4.2** Membuat Kelas HomePage |
| --- |
| import 'package:flutter/material.dart'; import 'package:flutter/widgets.dart';  class HomePage extends StatefulWidget {  final String username;   const HomePage({Key? key, required this.username}) : super(key: key);   @override  State<HomePage> createState() => \_HomePageState(); }  class \_HomePageState extends State<HomePage> {  @override  Widget build(BuildContext context) {  return Scaffold(  appBar: AppBar(  title: Text('Halaman Utama'),  ),  body: Container(),  );  } } |

1. Setelah berhasil, selanjutnya buatlah tampilan form pada halaman utama dengan memberikan beberapa identitas dan tombol login untuk submit hasil form yang telah diinput. Jangan lupa untuk menampilkan username yang dikirim oleh LoginPage. Pada modul ini tidak akan fokus pada penjelasan tampilan aplikasi. Agar tidak terlalu panjang maka akan dibuat satu per satu tampilannya. Pertama buatlah text yang menampilkan username

| **Langkah Praktikum 4.3** Membuat Tampilan Kelas HomePage |
| --- |
| class \_HomePageState extends State<HomePage> {  @override  Widget build(BuildContext context) {  return Scaffold(  appBar: AppBar(  title: Text('Halaman Utama'),  ),  body: Container(  padding: EdgeInsets.symmetric(vertical: 12, horizontal: 24),  child: Center(  child: Column(  children: [  **Text(  "Hai ${widget.username}\nSelamat Datang",  textAlign: TextAlign.center,  style: TextStyle(fontSize: 24),  ),**   ],  ),  ),  ),  );  } } |

1. Selanjutnya membuat tampilan form. Disini salah satu style form akan disimpan dalam function agar kode tidak terlalu redundant (berulang). Buatlah function \_formInput() yang memiliki berbagai parameter dinamis.

| **Langkah Praktikum 4.4** Membuat Style FormInput |
| --- |
| class \_HomePageState extends State<HomePage> {  . . .  Widget \_formInput(  {required String hint,  required String label,  required Function(String value) setStateInput,  int maxLines = 1}) {  return TextFormField(  enabled: true,  maxLines: maxLines,  decoration: InputDecoration(  hintText: hint,  labelText: label,  contentPadding: const EdgeInsets.all(12.0),  border: OutlineInputBorder(  borderRadius: BorderRadius.all(Radius.circular(8.0)),  borderSide: BorderSide(color: Colors.blue),  ),  enabledBorder: OutlineInputBorder(  borderRadius: BorderRadius.all(Radius.circular(8.0)),  borderSide: BorderSide(color: Colors.blue),  ),  ),  onChanged: setStateInput,  );  }  . . .  } |

1. Form yang telah dibuat hanya sebagai template style yang akan digunakan, selanjutnya buatlah 5 form berdasarkan template form tersebut, dan masukkan ke dalam function baru bernama \_buildForm() agar kode terlihat lebih rapi. Sebelum itu jangan lupa menambahkan variabel untuk menampung nilai dari isi form tersebut.

| **Langkah Praktikum 4.5** Membuat Form |
| --- |
| class \_HomePageState extends State<HomePage> {  String \_namaLengkap = "";  String \_email = "";  String \_nomorHandphone = "";  String \_alamatRumah = "";  . . .  Widget \_buildForm() {  return Column(  children: [  SizedBox(height: 12),  \_formInput(  hint: "Masukkan Nama Lengkap",  label: "Nama \*",  setStateInput: (value) {  setState(() {  \_namaLengkap = value;  });  },  ),  SizedBox(height: 12),  \_formInput(  hint: "Masukkan Alamat Email",  label: "Email \*",  setStateInput: (value) {  setState(() {  \_email = value;  });  },  ),  SizedBox(height: 12),  \_formInput(  hint: "Masukkan Nomor HP Aktif",  label: "Nomor HP \*",  setStateInput: (value) {  setState(() {  \_nomorHandphone = value;  });  },  ),   SizedBox(height: 12),  \_formInput(  hint: "Masukkan Alamat Rumah",  label: "Alamat",  setStateInput: (value) {  setState(() {  \_alamatRumah = value;  });  },  maxLines: 3,  ),  ],  );  }  . . .  } |

1. Selanjutnya buat tombol untuk submit form yang diinput. Sama seperti sebelumnya buatlah function \_buildButtonSubmit() untuk menampung widget dari Button.

| **Langkah Praktikum 4.6** Membuat Tombol Submit |
| --- |
| class \_HomePageState extends State<HomePage> {  . . .  Widget \_buildButtonSubmit() {  return Container(  child: ElevatedButton(  onPressed: () {},  child: Text('Submit'),  style: ElevatedButton.styleFrom(  padding: EdgeInsets.symmetric(horizontal: 32, vertical: 16),  textStyle: TextStyle(fontSize: 16)),  ),  );  }  . . .  } |

1. Terakhir gabungkanlah semua komponen function yang telah dibuat ke dalam function utama (Scaffold).

| **Langkah Praktikum 4.7** Menggabungkan Semua Komponen Form dan Button |
| --- |
| class \_HomePageState extends State<HomePage> {  . . .  @override  Widget build(BuildContext context) {  return Scaffold(  appBar: AppBar(  title: Text('Halaman Utama'),  ),  body: Container(  padding: EdgeInsets.symmetric(vertical: 12, horizontal: 24),  child: Center(  child: Column(  children: [  Text(  "Hai ${widget.username}\nSelamat Datang",  textAlign: TextAlign.center,  style: TextStyle(fontSize: 24),  ),  **SizedBox(height: 24),  Text("Lengkapi Biodata"),  SizedBox(height: 12),  \_buildForm(),  SizedBox(height: 24),  \_buildButtonSubmit(),**  ],  ),  ),  ),  );  }  . . .  } |

1. Setelah semua tampilan di HomePage selesai dibangun, selanjutnya membuat fungsi onPress pada tombol submit yang digunakan untuk mengirim input form ke halaman profil, dan jangan lupa mengirimkan data username ke halaman profil. Namun sebelum pindah halaman perlu melakukan validasi input agar data yang diinput sudah benar dengan cara melakukan cek data kosong. Apabila input sudah tepat dapat langsung pindah halaman dengan Navigator.push(). Tambahkan kode berikut di dalam function \_buildButtonSubmit()

| **Langkah Praktikum 4.8** Menambahkan Fungsi Pada Button Submit |
| --- |
| class \_HomePageState extends State<HomePage> {  . . .  Widget \_buildButtonSubmit() {  return Container(  child: ElevatedButton(  **onPressed: () {  if (\_namaLengkap == "" || \_email == "" || \_nomorHandphone == "") {  SnackBar snackBar = SnackBar(  content: Text("\* data tidak boleh kosong"),  );  ScaffoldMessenger.of(context)**  **.showSnackBar(snackBar);  } else {  Navigator.push(context, MaterialPageRoute(builder: (context) {  return ProfilePage(  nama: \_namaLengkap,  username: widget.username,  email: \_email,  nomor: \_nomorHandphone,  alamat: \_alamatRumah,  );  }));  }  },**  child: Text('Submit'),  style: ElevatedButton.styleFrom(  padding: EdgeInsets.symmetric(horizontal: 32, vertical: 16),  textStyle: TextStyle(fontSize: 16)),  ),  );  }  . . .  } |

1. Setelah melakukan kode diatas, tentunya akan terjadi error karena kelas ProfilePage belum dibuat. Buatlah file baru dengan nama profile\_page.dart kemudian isi file tersebut dengan kelas ProfilePage. Dan jangan lupa buat variabel untuk menerima data yang dikirim dari halaman utama serta siapkan constructor-nya.

| **Langkah Praktikum 4.9** Menyiapkan Kelas ProfilePage |
| --- |
| import 'package:flutter/widgets.dart';  class ProfilePage extends StatelessWidget {  **final String nama;  final String username;  final String email;  final String nomor;  final String alamat;   const ProfilePage({  Key? key,  required this.nama,  required this.username,  required this.email,  required this.nomor,  this.alamat = "",  }) : super(key: key);**  @override  Widget build(BuildContext context) {  return Container();  } } |

1. Kemudian buatlah tampilan halaman profil dengan menampilkan biodata hasil dari inputan halaman utama.

| **Langkah Praktikum 4.10** Menampilkan Biodata |
| --- |
| class ProfilePage extends StatelessWidget {  . . .  **@override  Widget build(BuildContext context) {  return Scaffold(  appBar: AppBar(  title: Text("Halaman Profil"),  ),  body: Container(  padding: EdgeInsets.symmetric(  vertical: 12,  horizontal: 24,  ),  child: Center(  child: Column(  children: [  Text(  "Biodata",  style: TextStyle(  fontSize: 24,  fontWeight: FontWeight.bold,  ),  ),  SizedBox(height: 24),  FlutterLogo(size: 150),  SizedBox(height: 12),  Text(  nama,  style: TextStyle(  fontSize: 20,  fontWeight: FontWeight.bold,  ),  ),  SizedBox(height: 8),  Text(  username,  style: TextStyle(  fontSize: 16,  fontStyle: FontStyle.italic,  ),  ),  SizedBox(height: 24),  Text(  email,  style: TextStyle(fontSize: 16),  ),  SizedBox(height: 16),  Text(  nomor,  style: TextStyle(fontSize: 16),  ),  SizedBox(height: 16),  Text(  alamat,  style: TextStyle(fontSize: 16),  textAlign: TextAlign.center,  ),  ],  ),  ),  ),  );  }** } |

1. Lalu komponen terakhir yaitu membuat tombol kembali dan tombol logout dibawah biodata alamat.

| **Langkah Praktikum 4.11** Membuat Tombol Kembali dan Tombol Logout |
| --- |
| . . .  child: Center(  child: Column(  children: [  . . .  **SizedBox(height: 36),  Row(  mainAxisAlignment: MainAxisAlignment.spaceEvenly,  children: [  ElevatedButton(  onPressed: () {},  child: Text('Kembali'),  style: ElevatedButton.styleFrom(  padding: EdgeInsets.symmetric(  horizontal: 30,  vertical: 18,  ),  ),  ),  OutlinedButton(  onPressed: () {},  child: Text("Logout"),  style: OutlinedButton.styleFrom(  padding: EdgeInsets.symmetric(  horizontal: 30,  vertical: 18,  ),  ),  ),**  ],  )  ],  ), ),  . . . |

1. Langkah terakhir yaitu menambahkan fungsi tombol kembali dan fungsi tombol logout. Seperti yang dijelaskan sebelumnya bahwa untuk kembali ke halaman sebelumnya dapat dilakukan dengan mengeluarkan stack halaman yang ada saat ini menggunakan Navigator.pop. Sedangkan untuk logout menggunakan fungsi tambahan baru yaitu Navigator.pushAndRemoveUntil yang berguna untuk berpindah halaman sekaligus menghapus seluruh stack yang ada dibawahnya.

| **Langkah Praktikum 4.12** Menambahkan Fungsi Tombol Kembali dan Tombol Logout |
| --- |
| . . .  Row(  mainAxisAlignment: MainAxisAlignment.spaceEvenly,  children: [  ElevatedButton(  **onPressed: () {  Navigator.pop(context);  },**  child: Text('Kembali'),  style: ElevatedButton.styleFrom(  padding: EdgeInsets.symmetric(  horizontal: 30,  vertical: 18,  ),  ),  ),  OutlinedButton(  **onPressed: () {  Navigator.pushAndRemoveUntil(  context,  MaterialPageRoute(builder: (context) {  return LoginPage();  }),  (route) => false,  );  },**  child: Text("Logout"),  style: OutlinedButton.styleFrom(  padding: EdgeInsets.symmetric(  horizontal: 30,  vertical: 18,  ),  ),  ),  ], )  . . . |

1. Aplikasi selesai dibangun dan siap dijalankan dengan normal.

**TUGAS PRAKTIKUM**

1. Buatlah sebuah aplikasi bebas yang memiliki minimal 3 halaman dan minimal menerapkan fitur navigator push, navigator push replacement, navigator pop, dan passing data. Aplikasi tidak boleh sama dengan yang ada di Langkah Praktikum.