LAPORAN JOBSHEET 10
PEMOGRAMAN MOBILE

Nama : Randa Heru Kusuma (22)

NIM : 2341760009

Kelas : 3D

Prodi : Sistem Informasi Bisnis

LINK GITHUB: https://github.com/randaheru/randaheru-praktikum10_flutter

JURUSAN TEKNOLOGI INFORMASI
TAHUN AJARAN
2024/2025

PRAKTIKUM 1

Buatlah sebuah project flutter baru dengan nama **master_plan**

Buat file bernama task.dart dan buat class Task. Class ini memiliki atribut description dengan tipe data String dan complete dengan tipe data Boolean, serta ada konstruktor. Kelas ini akan menyimpan data tugas untuk aplikasi kita

```dart
class Task {
  final String description;
  final bool complete;

  const Task({
    this.complete = false,
    this.description = '',
  });
}
```

Kita juga perlu sebuah List untuk menyimpan daftar rencana dalam aplikasi to-do ini. Buat file plan.dart di dalam folder **models**

```dart
import './task.dart';

class Plan {
  final String name;
  final List<Task> tasks;

  const Plan({this.name = '', this.tasks = const []});
}
```

Di folder **models**. Kodenya hanya berisi export data_layer.dart

```dart
export 'plan.dart';
export 'task.dart';
```

Ubah isi kode main.dart

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

Run | Debug | Profile
void main() => runApp(const MasterPlanApp());

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        primarySwatch: Colors.purple,
      ), // ThemeData
      home: const PlanScreen(),
    ); // MaterialApp
  }
}
```

Pada folder views, buatlah sebuah file plan_screen.dart dan gunakan templat StatefulWidget untuk membuat class PlanScreen. Isi kodenya adalah sebagai berikut. Gantilah teks **'Randa'** dengan nama panggilan Anda pada title AppBar

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

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

  @override
  State createState() => _PlanScreenState();
}

class _PlanScreenState extends State<PlanScreen> {
  Plan plan = const Plan();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Master Plan Randa')),
      body: _buildList(),
      floatingActionButton: _buildAddTaskButton(),
    ); // Scaffold
  }

  Widget _buildList() {
    return ListView.builder(
      itemCount: plan.tasks.length,
      itemBuilder: (context, index) => _buildTaskTile(plan.tasks[index], index),
    );
  }
```

Anda akan melihat beberapa error di langkah 6, karena method yang belum dibuat. Ayo kita buat mulai dari yang paling mudah yaitu tombol **Tambah Rencana**. Tambah kode berikut di bawah method build di dalam class _PlanScreenState.

```dart
  Widget _buildAddTaskButton() {
    return FloatingActionButton(
      child: const Icon(Icons.add),
      onPressed: () {
        setState(() {
          plan = Plan(
            name: plan.name,
            tasks: List<Task>.from(plan.tasks)..add(const Task()),
          ); // Plan
        });
      },
    ); // FloatingActionButton
  }
}
```

buat widget berupa List yang dapat dilakukan scroll, yaitu ListView.builder. Buat widget ListView

```dart
Widget _buildList() {
  return ListView.builder(
    itemCount: plan.tasks.length,
    itemBuilder: (context, index) => _buildTaskTile(plan.tasks[index], index),
  );
}
```

PRAKTIKUM 1

ListTile untuk menampilkan setiap nilai dari plan.tasks. Kita buat dinamis untuk setiap index data, sehingga membuat view menjadi lebih mudah.

```dart
Widget _buildTaskTile(Task task, int index) {
  return ListTile(
    leading: Checkbox(
      value: task.complete,
      onChanged: (selected) {
        setState(() {
          plan = Plan(
            name: plan.name,
            tasks: List<Task>.from(plan.tasks)
              ..[index] = Task(
                description: task.description,
                complete: selected ?? false,
              ), // Task
          ); // Plan
        });
      },
    ), // Checkbox
    title: TextFormField(
      initialValue: task.description,
      onChanged: (text) {
        setState(() {
          plan = Plan(
            name: plan.name,
            tasks: List<Task>.from(plan.tasks)
              ..[index] = Task(description: text, complete: task.complete),
          ); // Plan
```

Run atau tekan **F5** untuk melihat hasil aplikasi yang Anda telah buat. Capture hasilnya untuk soal praktikum nomor 4.

```dart
late ScrollController scrollController;
Plan plan = const Plan();
```

Tambahkan method initState() setelah deklarasi variabel scrollController

```
  scrollController = ScrollController()
    ..addListener(() {
      FocusScope.of(context).requestFocus(FocusNode());
    });
}
```

Tambahkan controller dan keyboard behavior pada ListView di method _buildList

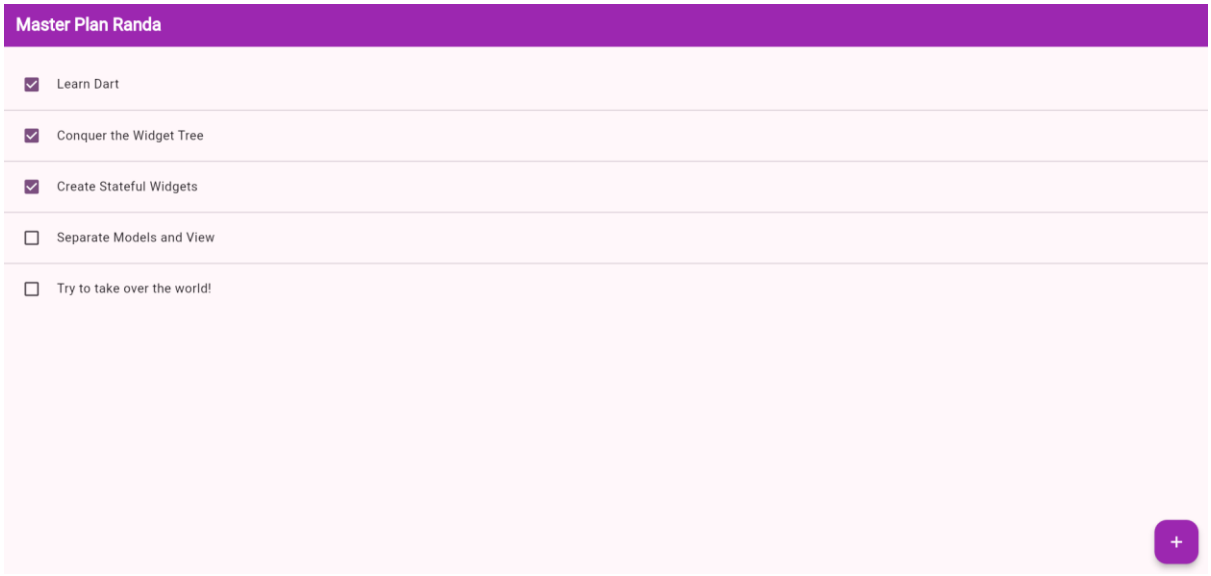
```
Widget _buildList() {
  return ListView.builder(
    controller: scrollController,
    keyboardDismissBehavior:
        Theme.of(context).platform == TargetPlatform.iOS
            ? ScrollViewKeyboardDismissBehavior.onDrag
            : ScrollViewKeyboardDismissBehavior.manual,
    itemCount: plan.tasks.length,
    itemBuilder: (context, index) =>
        _buildTaskTile(plan.tasks[index], index),
  );
}
```

Terakhir, tambahkan method dispose() berguna ketika widget sudah tidak digunakan lagi.

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

Langkah 14: Hasil

Lakukan Hot restart (bukan hot reload) pada aplikasi Flutter Anda. Anda akan melihat tampilan akhir seperti gambar berikut. Jika masih terdapat error, silakan diperbaiki hingga bisa running.



TUGAS PRAKTIKUM 1: Dasar State dengan Model-View

1. Selesaikan langkah-langkah praktikum tersebut, lalu dokumentasikan berupa GIF hasil akhir praktikum beserta penjelasannya di file README.md! Jika Anda menemukan ada yang error atau tidak berjalan dengan baik, silakan diperbaiki.

```
# Laporan Praktikum 1

**Pemrograman Mobile - Membangun Navigasi di Flutter**

---

## Identitas

* **Nama**       : Randa Heru Kusuma
* **NIM**        : 2341760009
* **Praktikum**  : 10 - Membangun Navigasi di Flutter

---

## Hasil

Berikut adalah tampilan hasil praktikum 2:

![Hasil Praktikum 10 - Tampilan 1](images/hasilPraktikum2.png)
![Hasil Praktikum 10 - Tampilan 2](images/hasilPraktikum2.2.png)

---

## Deskripsi Praktikum

Pada praktikum ini dipelajari bagaimana cara **membangun navigasi antar halaman di Flutter** menggunakan
`Navigator` dan `MaterialPageRoute`, serta penerapan pengiriman data antar halaman.
```

2. Jelaskan maksud dari langkah 4 pada praktikum tersebut! Mengapa dilakukan demikian?

Langkah tersebut dilakukan untuk merapikan proses impor model di dalam aplikasi Flutter. Dengan membuat satu file bernama **data_layer.dart** yang berisi perintah export 'plan.dart'; dan export 'task.dart';, kita tidak perlu lagi mengimpor kedua file

model tersebut satu per satu di setiap file yang membutuhkan. Cukup impor **data_layer.dart** saja, maka kedua model sudah otomatis ikut terimpor. Cara ini membuat struktur kode lebih sederhana, mudah dikelola, dan lebih rapi terutama ketika jumlah model semakin banyak seiring berkembangnya aplikasi. Selain itu, langkah ini juga mengurangi pengulangan kode karena tidak perlu menuliskan banyak baris import di setiap file. Ini adalah praktik yang umum dilakukan untuk meningkatkan efisiensi dan kerapian pada proyek Flutter.

3. Mengapa perlu variabel plan di langkah 6 pada praktikum tersebut? Mengapa dibuat konstanta ?

   Variabel **plan** diperlukan karena data rencana (plan) inilah yang akan menyimpan daftar task yang ada pada aplikasi. Semua perubahan data task (misalnya menambah task, mengubah teks task, atau mencentang task sebagai selesai) akan disimpan di dalam variabel plan ini, lalu ditampilkan ke UI. Oleh karena itu variabel plan harus dibuat di dalam State supaya datanya bisa berubah dan ketika berubah, UI bisa ikut ter-update. Sementara itu, plan dibuat sebagai **const** pada awal pembuatan hanya untuk memberikan nilai awal default yang kosong, karena pada saat pertama aplikasi berjalan belum ada data task sama sekali. Penggunaan const di sini karena object Plan awal tersebut bersifat tetap sebagai nilai awal sebelum terjadi perubahan, dan const membuat object awal tersebut lebih efisien serta immutable sampai ada perubahan yang dibuat melalui setState.

4. Lakukan capture hasil dari Langkah 9 berupa GIF, kemudian jelaskan apa yang telah Anda buat!

   Membuat sebuah tampilan daftar tugas (to-do list) dinamis dimana setiap baris task mempunyai dua komponen utama yaitu Checkbox dan TextFormField. Checkbox berfungsi untuk menandai apakah task tersebut sudah selesai (complete) atau belum, sedangkan TextFormField berfungsi untuk mengubah atau mengedit teks deskripsi task secara langsung. Ketika user mencentang Checkbox atau mengetik teks baru pada TextFormField, akan terjadi perubahan state melalui setState(), sehingga object plan akan diperbarui dan tampilan UI ikut berubah secara real-time. Dengan kata lain, langkah ini membuat sistem pencatatan task interaktif, dimana user bisa menambahkan task, mengedit teks task, dan menandai task selesai secara langsung dalam satu tampilan. GIF yang ditampilkan nanti menunjukkan perubahan UI tersebut ketika user mengetik atau menekan checkbox sehingga terlihat bagaimana state management bekerja dalam aplikasi ini.

5. Apa kegunaan method pada Langkah 11 dan 13 dalam *lifecyle state* ?

   Secara keseluruhan, langkah-langkah yang dilakukan dalam praktikum ini bertujuan untuk membangun fitur daftar tugas yang dinamis dengan pengelolaan state yang baik. Kita menggunakan `Plan` untuk menyimpan data task, lalu mengatur perubahan

data melalui `setState()` agar tampilan selalu ter-update setiap ada perubahan. Method `initState()` digunakan untuk menyiapkan controller dan listener saat widget pertama kali dibuat, sedangkan `dispose()` digunakan untuk membersihkan controller agar tidak terjadi kebocoran memori ketika widget sudah tidak digunakan lagi. Dengan pendekatan seperti ini, aplikasi menjadi lebih rapi, interaktif, dan efisien dalam mengelola resource.

6. Kumpulkan laporan praktikum Anda berupa link commit atau repository GitHub ke dosen yang telah disepakati !

   https://github.com/randaheru/randaheru-praktikum10_flutter

PRAKTIKUM 2

Buat folder baru provider di dalam folder lib, lalu buat file baru dengan nama plan_provider.dart berisi kode seperti berikut.

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

class PlanProvider extends InheritedNotifier<ValueNotifier<Plan>> {
  const PlanProvider({super.key, required Widget child, required
   ValueNotifier<Plan> notifier})
  : super(child: child, notifier: notifier);

  static ValueNotifier<Plan> of(BuildContext context) {
   return context.
    dependOnInheritedWidgetOfExactType<PlanProvider>()!.notifier!;
  }
}
```

Gantilah pada bagian atribut home dengan PlanProvider seperti berikut. Jangan lupa sesuaikan bagian impor jika dibutuhkan.

```
      home: PlanProvider(
        notifier: ValueNotifier<Plan>(const Plan()),
        child: const PlanScreen(),
      ), // PlanProvider
    ); // MaterialApp
  }
}
```

Tambahkan dua *method* di dalam model class Plan seperti kode berikut.

```
// Menghitung jumlah task yang sudah complete
int get completedCount => tasks.where((task) => task.complete).length;

// Memberikan pesan ringkasan progress
String get completenessMessage =>
    '$completedCount out of ${tasks.length} tasks';
}
```

Edit PlanScreen agar menggunakan data dari PlanProvider. Hapus deklarasi variabel plan (ini akan membuat error).

```
@override
Widget build(BuildContext context) {
  // Ambil ValueNotifier<Plan> dari PlanProvider
  planNotifier = PlanProvider.of(context);
```

Tambahkan BuildContext sebagai parameter dan gunakan PlanProvider sebagai sumber datanya

```dart
Widget _buildAddTaskButton(BuildContext context) {
  ValueNotifier<Plan> planNotifier = PlanProvider.of(context);
  return FloatingActionButton(
    backgroundColor: Colors.purple,
    child: const Icon(Icons.add, color: Colors.white),
    onPressed: () {
      Plan currentPlan = planNotifier.value;
      planNotifier.value = Plan(
        name: currentPlan.name,
        tasks: List<Task>.from(currentPlan.tasks)..add(const Task()),
      ); // Plan

      Future.delayed(const Duration(milliseconds: 100), () {
        scrollController.animateTo(
          scrollController.position.maxScrollExtent,
          duration: const Duration(milliseconds: 200),
          curve: Curves.easeOut,
        );
      }); // Future.delayed
    },
  ); // FloatingActionButton
}
```

Tambahkan parameter BuildContext, gunakan PlanProvider sebagai sumber data. Ganti TextField menjadi TextFormField untuk membuat inisial data provider menjadi lebih mudah.

```dart
Widget _buildTaskTile(Task task, int index, BuildContext context) {
  ValueNotifier<Plan> planNotifier = PlanProvider.of(context);
  return ListTile(
    leading: Checkbox(
      value: task.complete,
      onChanged: (selected) {
        Plan currentPlan = planNotifier.value;
        planNotifier.value = Plan(
          name: currentPlan.name,
          tasks: List<Task>.from(currentPlan.tasks)
            ..[index] = Task(
              description: task.description,
              complete: selected ?? false,
            ), // Task
        ); // Plan
      },
    ), // Checkbox
    title: TextFormField(
      initialValue: task.description,
      decoration: const InputDecoration(border: InputBorder.none),
      onChanged: (text) {
        Plan currentPlan = planNotifier.value;
        planNotifier.value = Plan(
          name: currentPlan.name,
          tasks: List<Task>.from(currentPlan.tasks)
            ..[index] = Task(description: text, complete: task.complete),
        ); // Plan
      },
    ), // TextFormField
  ); // ListTile
}
```

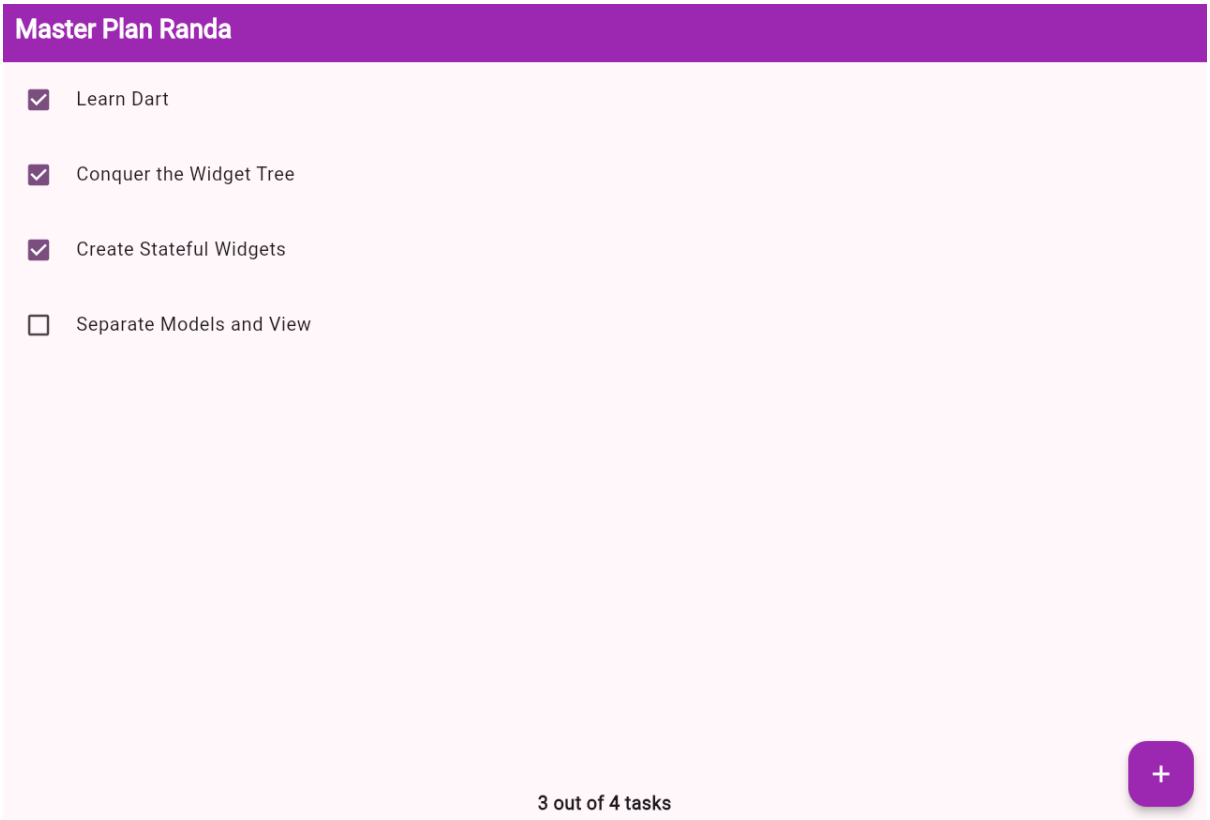
Sesuaikan parameter pada bagian _buildTaskTile

```
Widget _buildList(Plan plan) {
  return ListView.builder(
    controller: scrollController,
    keyboardDismissBehavior:
        Theme.of(context).platform == TargetPlatform.iOS
            ? ScrollViewKeyboardDismissBehavior.onDrag
            : ScrollViewKeyboardDismissBehavior.manual,
    itemCount: plan.tasks.length,
    itemBuilder: (context, index) =>
        _buildTaskTile(plan.tasks[index], index, context),
  );
}
```

Edit method build sehingga bisa tampil progress pada bagian bawah (footer).
Caranya, bungkus (wrap) _buildList dengan widget Expanded dan masukkan ke dalam widget dan Terakhir, tambahkan widget SafeArea dengan
berisi completenessMessage pada akhir widget Column

```
            // Footer: menampilkan progress
            SafeArea(
              child: Padding(
                padding: const EdgeInsets.all(8.0),
                child: Text(
                  plan.completenessMessage,
                  style: const TextStyle(
                    fontWeight: FontWeight.bold,
                    fontSize: 16,
                  ), // TextStyle
                ), // Text
              ), // Padding
            ), // SafeArea
```

Akhirnya, **run** atau tekan **F5** jika aplikasi belum running. Tidak akan terlihat perubahan pada UI, namun dengan melakukan langkah-langkah di atas, Anda telah menerapkan cara memisahkan dengan baik antara **view** dan **model**.



TUGAS PRAKTIKUM 2: InheritedWidget

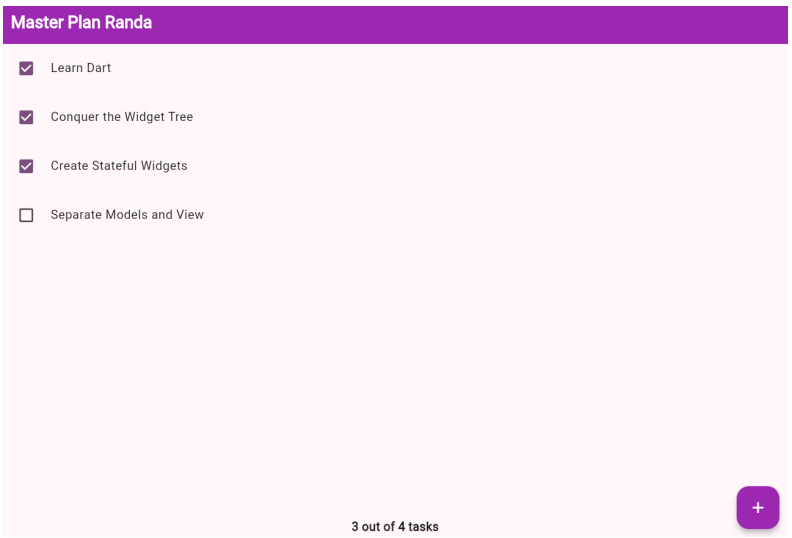
1. Selesaikan langkah-langkah praktikum tersebut, lalu dokumentasikan berupa GIF hasil akhir praktikum beserta penjelasannya di file README.md! Jika Anda menemukan ada yang error atau tidak berjalan dengan baik, silakan diperbaiki sesuai dengan tujuan aplikasi tersebut dibuat.

2. Jelaskan mana yang dimaksud InheritedWidget pada langkah 1 tersebut! Mengapa yang digunakan InheritedNotifier?

```
class PlanProvider extends InheritedNotifier<ValueNotifier<Plan>> {
  const PlanProvider({
    super.key,
    required super.child,
    required super.notifier,
  });
```

Artinya, setiap kali data Plan berubah, widget yang tergantung pada PlanProvider otomatis akan rebuild. Jika kita hanya menggunakan InheritedWidget biasa, perubahan data tidak otomatis memberi tahu widget anak; kita harus memanggil setState manual di seluruh tree. Dengan InheritedNotifier, perubahan Plan langsung ter-propagate ke semua widget yang dependOnInheritedWidgetOfExactType<PlanProvider>()

3. Jelaskan maksud dari method di langkah 3 pada praktikum tersebut! Mengapa dilakukan demikian?
   Dengan menambahkan method completedCount dan completenessMessage pada class Plan, aplikasi dapat menampilkan progress tugas secara otomatis di UI tanpa perlu menulis kode tambahan di PlanScreen. Pendekatan ini memisahkan logika perhitungan progress dari tampilan, sehingga struktur kode menjadi lebih bersih, rapi, dan mudah dipelihara. Setiap kali terjadi perubahan pada status task, method completenessMessage akan langsung menghasilkan pesan progress yang terbaru, sehingga tampilan UI selalu mencerminkan kondisi data terkini secara real-time.
4. Lakukan capture hasil dari Langkah 9 berupa GIF, kemudian jelaskan apa yang telah Anda buat!



   Dengan struktur ini, aplikasi dapat menampilkan daftar tugas secara dinamis, menerima input user, dan **menampilkan progress secara real-time** tanpa perlu menulis kode manual untuk update UI setiap kali data berubah.
5. Kumpulkan laporan praktikum Anda berupa link commit atau repository GitHub ke dosen yang telah disepakati !
   https://github.com/randaheru/randaheru-praktikum10_flutter

PRAKTIKUM 3

Edit class PlanProvider sehingga dapat menangani List Plan.

```
// Method untuk mengakses List<Plan> dari context
static ValueNotifier<List<Plan>> of(BuildContext context) {
  final provider =
      context.dependOnInheritedWidgetOfExactType<PlanProvider>();
  assert(provider != null, 'Tidak ada PlanProvider di context');
  return provider!.notifier!;
}
```

Langkah sebelumnya dapat menyebabkan error
pada main.dart dan plan_screen.dart. Pada method build, gantilah menjadi

```
@override
Widget build(BuildContext context) {
  return PlanProvider(
    notifier: ValueNotifier<List<Plan>>([]), // List kosong awal
    child: MaterialApp(
      title: 'State management app',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ), // ThemeData
      home: const PlanScreen(),
    ), // MaterialApp
  ); // PlanProvider
}
}
```

Itu akan terjadi error setiap kali memanggil PlanProvider.of(context). Itu terjadi karena screen saat ini hanya menerima tugas-tugas untuk satu kelompok Plan, tapi sekarang PlanProvider menjadi list dari objek plan

```
body: ValueListenableBuilder<Plan>(
  valueListenable: planNotifier,
  builder: (context, plan, child) {
    return Column(
```

Tambahkan getter pada _PlanScreenState

```
class _PlanScreenState extends State<PlanScreen> {
  late ScrollController scrollController;
```

Pada bagian ini kode tetap

```
@override
void initState() {
  super.initState();
  scrollController = ScrollController()
    ..addListener(() {
      FocusScope.of(context).requestFocus(FocusNode());
    });
}
```

Merubah ke List dan mengubah nilai pada currentPlan

```dart
@override
Widget build(BuildContext context) {
  ValueNotifier<List<Plan>> plansNotifier = PlanProvider.of(context);

  return Scaffold(
    appBar: AppBar(title: Text(_plan.name)),
    body: ValueListenableBuilder<List<Plan>>(
      valueListenable: plansNotifier,
      builder: (context, plans, child) {
        // Ambil plan yang sesuai
        Plan currentPlan =
            plans.firstWhere((p) => p.name == _plan.name, orElse: () => _plan);

        return Column(
          children: [
            Expanded(child: _buildList(currentPlan)),
            SafeArea(
              child: Padding(
                padding: const EdgeInsets.all(8.0),
                child: Text(
                  currentPlan.completenessMessage,
                  style: const TextStyle(
                      fontWeight: FontWeight.bold, fontSize: 16), // TextStyle
```

Pastikan ubah ke List dan variabel planNotifier

```dart
Widget _buildTaskTile(Task task, int index, BuildContext context) {
  ValueNotifier<List<Plan>> planNotifier = PlanProvider.of(context);

  return ListTile(
    leading: Checkbox(
      value: task.complete,
      onChanged: (selected) {
        Plan currentPlan = _plan;
        int planIndex =
            planNotifier.value.indexWhere((p) => p.name == currentPlan.name);

        List<Task> updatedTasks = List<Task>.from(currentPlan.tasks)
          ..[index] = Task(description: task.description, complete: selected ?? false);

        planNotifier.value = List<Plan>.from(planNotifier.value)
          ..[planIndex] = Plan(name: currentPlan.name, tasks: updatedTasks);

        setState(() {
          _plan = Plan(name: currentPlan.name, tasks: updatedTasks);
        });
      },
    ), // Checkbox
    title: TextFormField(
      initialValue: task.description,
      decoration: const InputDecoration(border: InputBorder.none),
      onChanged: (text) {
```

Pada folder view, buatlah file baru dengan nama plan_creator_screen.dart dan deklarasikan dengan StatefulWidget bernama PlanCreatorScreen. Gantilah di main.dart pada atribut home menjadi seperti berikut.

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

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

  @override
  State<PlanCreatorScreen> createState() => _PlanCreatorScreenState();
}

class _PlanCreatorScreenState extends State<PlanCreatorScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Master Plans Namaku'), // ganti 'Namaku' sesuai keinginan
      ), // AppBar
      body: const Center(
        child: Text('Halaman untuk membuat plan baru akan tampil di sini'),
      ), // Center
    ); // Scaffold
  }
}
```

```dart
home: const PlanCreatorScreen(),
```

Kita perlu tambahkan variabel TextEditingController sehingga bisa membuat TextField sederhana untuk menambah Plan baru

```dart
class _PlanCreatorScreenState extends State<PlanCreatorScreen> {
  // Controller untuk TextField input plan baru
  final textController = TextEditingController();

  @override
  void dispose() {
    textController.dispose(); // penting agar tidak memory leak
    super.dispose();
  }
```

Letakkan method Widget build berikut di atas void dispose. Gantilah “**randa**”

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: const Text('Master Plans Randa')),
    body: Column(
      children: [
        _buildListCreator(),
        Expanded(child: _buildMasterPlans()),
      ],
    ), // Column
  ); // Scaffold
}
```

Buatlah widget berikut setelah widget build.

```dart
// Widget TextField untuk menambah plan baru
Widget _buildListCreator() {
  return Padding(
    padding: const EdgeInsets.all(20.0),
    child: Material(
      color: Theme.of(context).cardColor,
      elevation: 10,
      child: TextField(
        controller: textController,
        decoration: const InputDecoration(
          labelText: 'Add a plan',
          contentPadding: EdgeInsets.all(20),
        ), // InputDecoration
        onEditingComplete: addPlan,
      ), // TextField
    ), // Material
  ); // Padding
}
```

Tambahkan method berikut untuk menerima inputan dari user berupa text plan.

```dart
  // Method untuk menambahkan plan baru ke PlanProvider
 void addPlan() {
  final text = textController.text;
    if (text.isEmpty) {
      return;
    }
    final plan = Plan(name: text, tasks: []);
    ValueNotifier<List<Plan>> planNotifier =
PlanProvider.of(context);
    planNotifier.value = List<Plan>.from(planNotifier.value)..
add(plan);
    textController.clear();
    FocusScope.of(context).requestFocus(FocusNode());
    setState(() {});
}
```

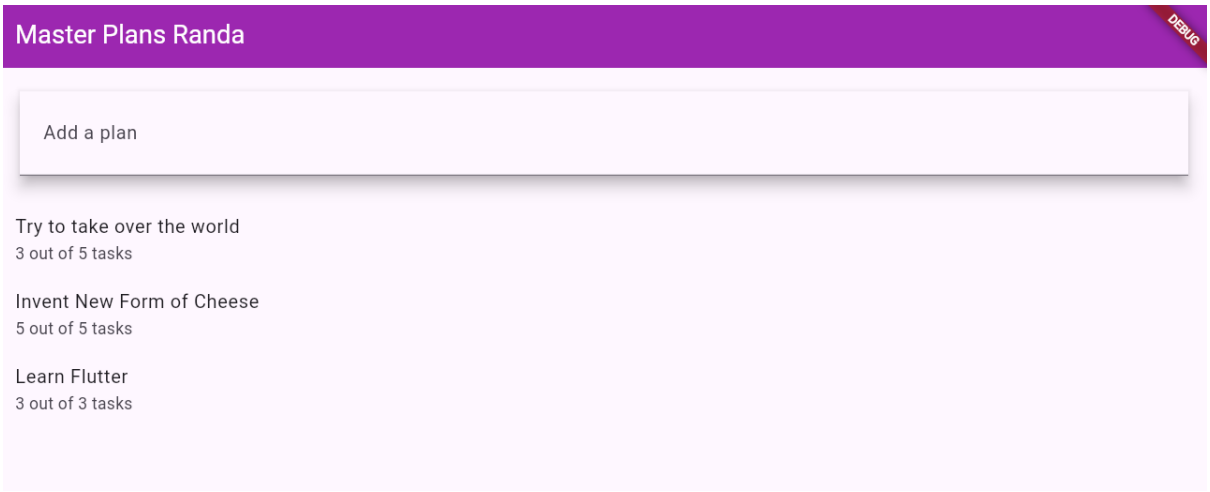
Tambahkan widget

```dart
// Widget menampilkan daftar plan
Widget _buildMasterPlans() {
  ValueNotifier<List<Plan>> planNotifier = PlanProvider.of(context);
  List<Plan> plans = planNotifier.value;

  if (plans.isEmpty) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
        const Icon(Icons.note, size: 100, color: Colors.grey),
        Text(
          'Anda belum memiliki rencana apapun.',
          style: Theme.of(context).textTheme.headlineSmall,
        ), // Text
      ], // <Widget>[]
    ); // Column
  }

  return ListView.builder(
    itemCount: plans.length,
    itemBuilder: (context, index) {
      final plan = plans[index];
      return ListTile(
```

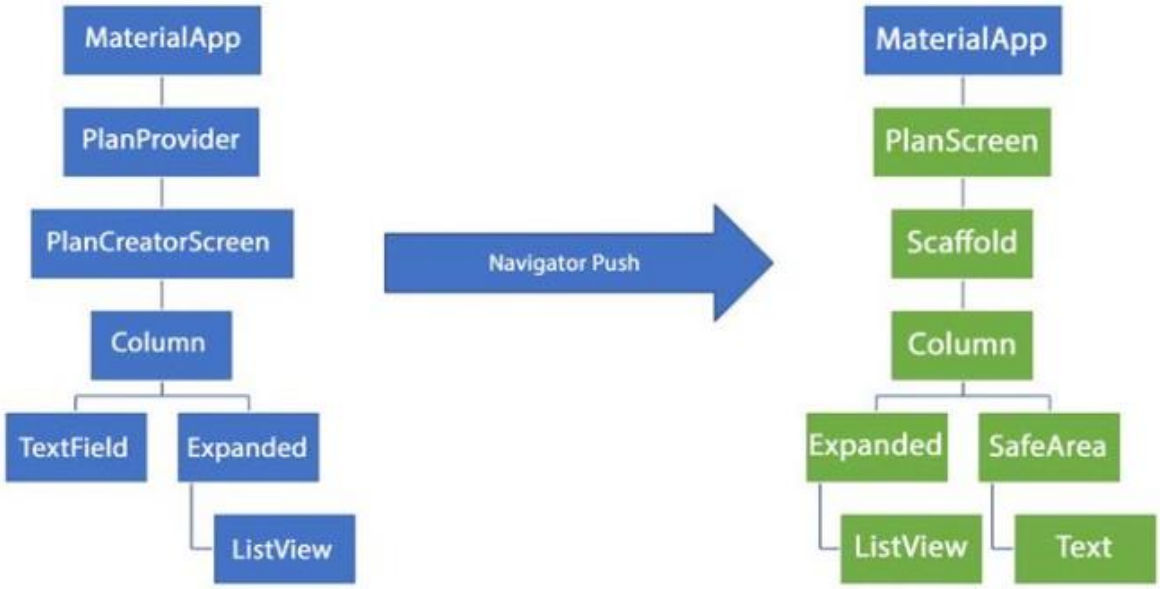
Terakhir, **run** atau tekan **F5** untuk melihat hasilnya jika memang belum running. Bisa juga lakukan **hot restart** jika aplikasi sudah running.



TUGAS PRAKTIKUM 3: State di Multiple Screens

1. Selesaikan langkah-langkah praktikum tersebut, lalu dokumentasikan berupa GIF hasil akhir praktikum beserta penjelasannya di file README.md! Jika Anda menemukan ada yang error atau tidak berjalan dengan baik, silakan diperbaiki sesuai dengan tujuan aplikasi tersebut dibuat.

2. Berdasarkan Praktikum 3 yang telah Anda lakukan, jelaskan maksud dari gambar diagram berikut ini!

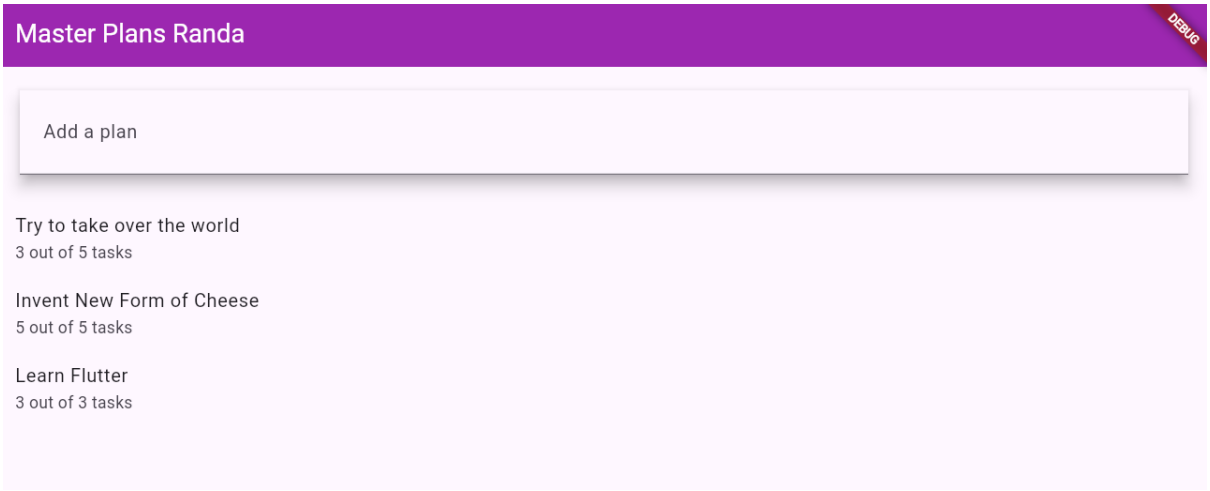


Struktur hirarki widget Flutter dan alur navigasi antara dua layar dalam aplikasi manajemen plan yang dibuat pada Praktikum 3. Berikut penjelasannya:

1. **Bagian kiri (biru) – PlanCreatorScreen**
   - Dimulai dari **MaterialApp**, yang merupakan root aplikasi Flutter.
   - Di bawahnya terdapat **PlanProvider**, yang menyediakan state management untuk menyimpan daftar plan (List<Plan>).
   - Selanjutnya adalah **PlanCreatorScreen**, layar utama untuk membuat plan baru.
   - Di dalam layar ini, terdapat **Column** yang menjadi kontainer vertikal untuk widget-widget anak.
     - **TextField**: digunakan untuk input nama plan baru.
     - **Expanded → ListView**: menampilkan daftar plan yang sudah dibuat. Expanded membuat ListView memenuhi sisa ruang layar.
2. **Bagian kanan (hijau) – PlanScreen**
   - Setelah pengguna menekan salah satu plan di PlanCreatorScreen, aplikasi menggunakan **Navigator Push** untuk berpindah ke **PlanScreen**.
   - Struktur widget di PlanScreen:
     - **Scaffold**: menyediakan layout dasar dengan AppBar, body, dan floatingActionButton.
     - **Column**: mengatur layout vertikal.
       - **Expanded → ListView**: menampilkan daftar task pada plan.
       - **SafeArea → Text**: menampilkan pesan progress atau status kelengkapan plan, tetap berada di area aman layar.
3. **Makna keseluruhan diagram**
   - Diagram menunjukkan **alur navigasi dan hubungan widget** antara layar pembuatan plan dan layar detail plan.

- Bagian kiri fokus pada **input dan daftar plan**, sedangkan bagian kanan fokus pada **detail dan task dari plan yang dipilih**.
- Memperlihatkan penggunaan **state management** (PlanProvider + ValueNotifier) dan **navigasi antar-layar** menggunakan Navigator.push.

3. Lakukan capture hasil dari Langkah 14 berupa GIF, kemudian jelaskan apa yang telah Anda buat!



buildMasterPlans() adalah widget yang digunakan untuk menampilkan daftar plan yang tersimpan di PlanProvider. Jika daftar plan masih kosong, widget ini akan menampilkan ikon dan teks pemberitahuan agar pengguna mengetahui bahwa belum ada plan yang dibuat. Sebaliknya, jika daftar plan tersedia, widget akan menampilkan ListView scrollable, di mana setiap item berupa ListTile yang menampilkan nama plan dan statusnya. Pengguna dapat mengetuk salah satu plan untuk menavigasi ke halaman detail (PlanScreen) dan melihat atau mengedit task pada plan tersebut. Fungsi ini memanfaatkan state management menggunakan ValueNotifier sehingga tampilan UI bersifat dinamis dan responsif terhadap perubahan data.

4. Kumpulkan laporan praktikum Anda berupa link commit atau repository GitHub ke dosen yang telah disepakati
   https://github.com/randaheru/randaheru-praktikum10_flutter