LAPORAN PRAKTIKUM
PERTEMUAN 10

DASAR STATE MANAGEMENT

SIB 3C

Oleh:

Syifa Revalina Kamila

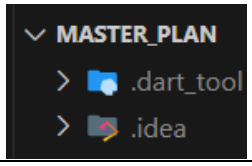
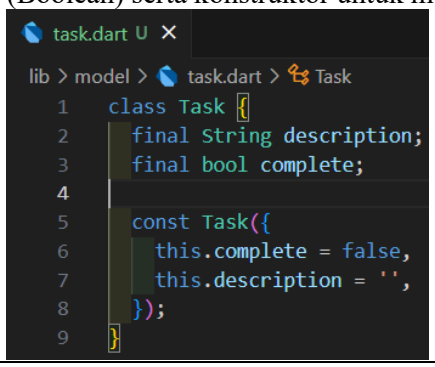
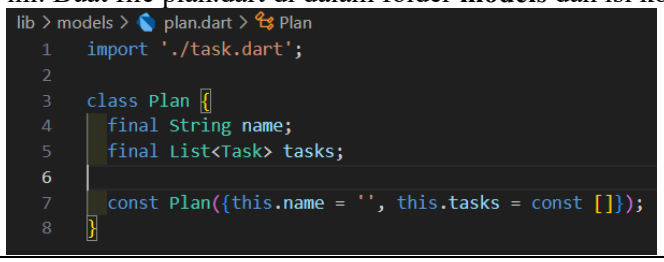
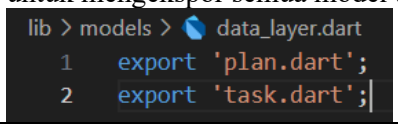
2341760041

PROGRAM STUDI SISTEM INFORMASI BISNIS

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG 2025/2026

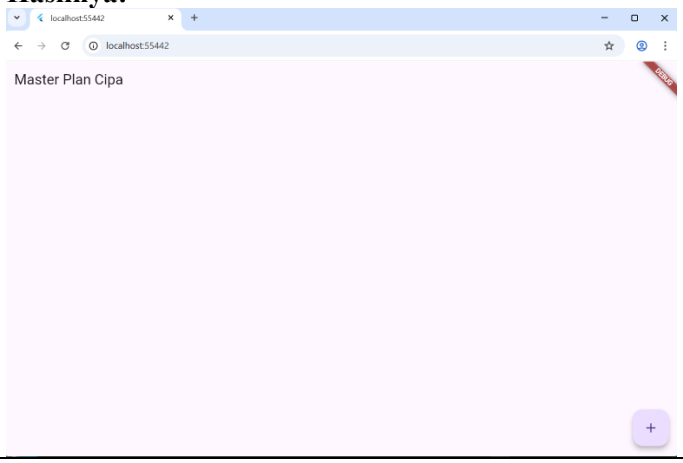
Praktikum 1: Dasar State dengan Model-View

No	Langkah - Langkah
1	Buatlah sebuah project flutter baru dengan nama master_plan di folder src week-10 repository GitHub Anda atau sesuai style laporan praktikum yang telah disepakati. Lalu buatlah susunan folder dalam project seperti gambar berikut ini. 
2	Langkah awal adalah membuat model di task.dart. Buat file task.dart di folder model, lalu definisikan class Task dengan atribut description (String) dan complete (Boolean) serta konstruktor untuk menyimpan data tugas aplikasi. 
3	Kita juga perlu sebuah List untuk menyimpan daftar rencana dalam aplikasi to-do ini. Buat file plan.dart di dalam folder **models** dan isi kode seperti berikut. 
4	Kita bisa menggabungkan beberapa model dalam satu file agar proses impor lebih efisien. Buat file **data_layer.dart** di folder **models**, yang berisi perintah **export** untuk mengekspor semua model tersebut. 
5	Ubah isi kode main.dart sebagai berikut. 

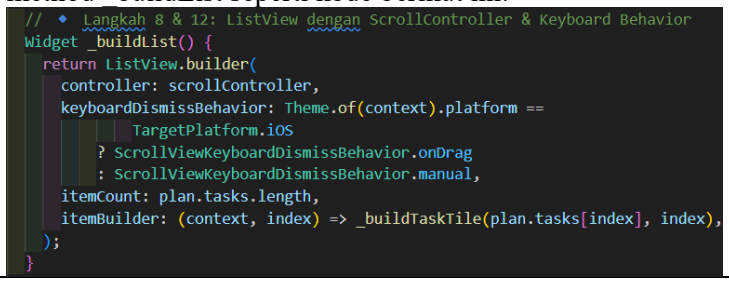
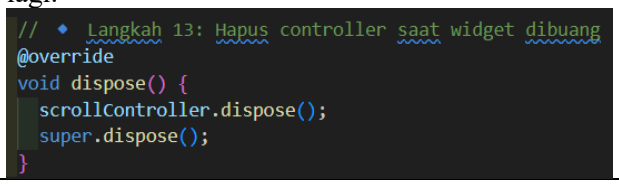
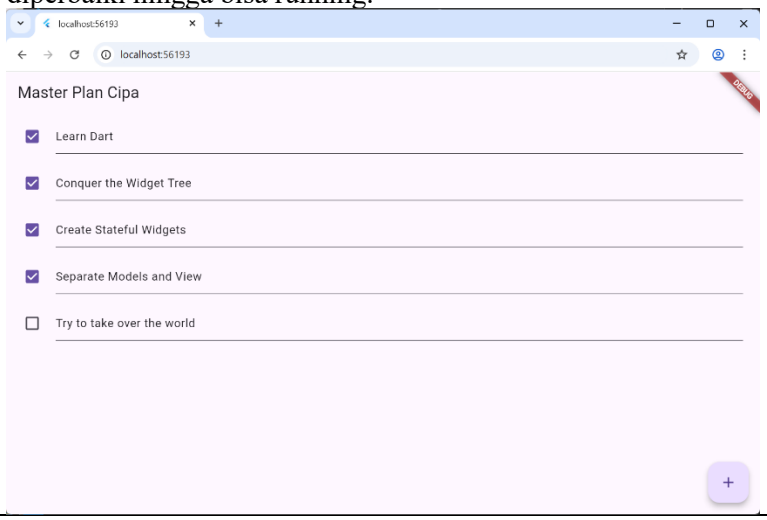
6	Pada folder views, buatlah sebuah file plan_screen.dart dan gunakan templat StatefulWidget untuk membuat class PlanScreen. Isi kodenya adalah sebagai berikut. Gantilah teks **'Namaku'** dengan nama panggilan Anda pada title AppBar. lib > views > plan_screen.dart > _PlanScreenState > build `import '../models/data_layer.dart';` `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(` `  // ganti 'Namaku' dengan Nama panggilan Anda` `  appBar: AppBar(title: const Text('Master Plan Cipa')),` `  body: _buildList(),` `  floatingActionButton: _buildAddTaskButton(),` `  ); // Scaffold` `  }` `}`
7	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. `// 🔹 Langkah 7: Membuat tombol tambah rencana/tugas` `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()), // menambahkan task` `  ); // Plan` `  });` `  },` `  ); // FloatingActionButton` `}`
8	Kita akan buat widget berupa List yang dapat dilakukan scroll, yaitu ListView.builder. Buat widget ListView seperti kode berikut ini. `// 🔹 Langkah 8: Membuat widget list untuk menampilkan semua task` `Widget _buildList() {` `  return ListView.builder(` `  itemCount: plan.tasks.length,` `  itemBuilder: (context, index) =>` `  _buildTaskTile(plan.tasks[index], index),` `  );` `}`
9	Dari langkah 8, kita butuh ListTile untuk menampilkan setiap nilai dari plan.tasks. Kita buat dinamis untuk setiap index data, sehingga membuat view menjadi lebih mudah. Tambahkan kode berikut ini.

```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.task
s)                ..[index] = Task(
                  description: task.description,
                  complete: selected ?? false,
                ),
            );
          });
        },
      ),
      title: TextFormField(
        initialValue: task.description,
        decoration: const InputDecoration(
          hintText: 'Tulis deskripsi tugas...',
        ),
        onChanged: (text) {
          setState(() {
            plan = Plan(
              name: plan.name,
              tasks: List<Task>.from(plan.task
s)                ..[index] = Task(
                  description: text,
                  complete: task.complete,
                ),
            );
          });
        },
      ),
    );
  }
}
```

Hasilnya:



10	Anda bisa menambahkan banyak tugas, menandai yang sudah selesai, dan menggulir saat daftar tugas makin panjang. Namun, di iOS ada kendala ketika keyboard muncul, yaitu sulit mengakses kolom input di bagian bawah. Solusinya, gunakan **ScrollController** untuk menghapus fokus dari semua **TextField** saat pengguna menggulir. Tambahkan variabel **scroll controller** di class **State** pada file **plan_screen.dart**, tepat setelah variabel **plan**. `// 🔹 Langkah 10: Tambahkan ScrollController` `late ScrollController scrollController;`
11	Tambahkan method initState() setelah deklarasi variabel scrollController seperti kode berikut.

	
12	Tambahkan controller dan keyboard behavior pada ListView di method _buildList seperti kode berikut ini. 
13	Terakhir, tambahkan method dispose() berguna ketika widget sudah tidak digunakan lagi. 
14	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.

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

   Jawab : membuat respresentasi data tugas (Task) yang akan digunakan dalam aplikasi dengan memisahkan model ke folder models, struktur proyek menjadi lebih terorganisir dan mudah

dikembangkan. Mengapa dilakukan demikian? Karena dalam arsitektur flutter yang baik, data (model) dipisahkan dari view agar aplikasi mudah dikembangkan, data dapat digunakan di berbagai bagain apliaksi.

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

   Jawab: Variable plan berfungsi sebagai wadah utama untuk menyimpan seluruh daftar task yang dimiliki pengguna. Mengapa dibuat konstanta? Karena konstanta digunakan plan dan task tidak bisa diubah langsung, saat menambah/mengubah data kita membuat salinan baru (copy) dari plan bukan mengedit secara langsung.

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

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

   Jawab: langkah 11 pada bagian initsatate() kegunaannya adalah method yang dijalankan satu kali saat pertama kali widget StatefulWidget dibuat, dan langkah 13 kegunaanya pada dispose() adalah method yang dijalankan ketika widget dihapus dari tree (sudah tidak digunakan lagi).

Praktikum 2 : Mengelola Data Layer dengan InheritedWidget dan InheritedNotifier

No	Langkah - Langkah
1	Buat folder baru provider di dalam folder lib, lalu buat file baru dengan nama plan_provider.dart berisi kode seperti berikut. lib > provider > plan_provider.dart > PlanProvider `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!;` `  }` `}`
2	Gantilah pada bagian atribut home dengan PlanProvider seperti berikut. Jangan lupa sesuaikan bagian impor jika dibutuhkan. `return MaterialApp(` `  theme: ThemeData(primarySwatch: Colors.purple),` `  home: PlanProvider(` `  notifier: ValueNotifier<Plan>(const Plan()),` `  child: const PlanScreen(),` `  ) // PlanProvider` `  ); // MaterialApp` `  }` `}`
3	Tambahkan dua *method* di dalam model class Plan seperti kode berikut. `// • Menghitung jumlah task yang sudah selesai` `int get completedCount => tasks.where((task) => task.complete).length;` `// • Menampilkan pesan ringkasan progres` `String get completenessMessage =>` `  '$completedCount out of ${tasks.length} tasks';` `}`
4	Edit PlanScreen agar menggunakan data dari PlanProvider. Hapus deklarasi variabel plan (ini akan membuat error). Kita akan perbaiki pada langkah 5 berikut ini.
5	Tambahkan BuildContext sebagai parameter dan gunakan PlanProvider sebagai sumber datanya. Edit bagian kode seperti berikut.

	``dart Widget _buildAddTaskButton(BuildContext context) { ValueNotifier<Plan> planNotifier = PlanProvider.of(context); return FloatingActionButton(child: const Icon(Icons.add), onPressed: () { Plan currentPlan = planNotifier.value; planNotifier.value = Plan(name: currentPlan.name, tasks: List<Task>.from(currentPlan.tasks)..add(const Task()),); // Plan },); // FloatingActionButton } ```
6	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 }, ```
7	Sesuaikan parameter pada bagian _buildTaskTile seperti kode berikut. ```dart 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),); } ```
8	Edit method build sehingga bisa tampil progress pada bagian bawah (footer). Caranya, bungkus (wrap) _buildList dengan widget Expanded dan masukkan ke dalam widget Column seperti kode pada Langkah 9.
9	Terakhir, tambahkan widget SafeArea dengan berisi completenessMessage pada akhir widget Column. Perhatikan kode berikut ini. ```dart @override Widget build(BuildContext context) { return Scaffold(appBar: AppBar(title: const Text('Master Plan Cipa')), body: ValueListenableBuilder<Plan>(valueListenable: PlanProvider.of(context), builder: (context, plan, child) { return Column(children: [Expanded(child: _buildList(plan)), SafeArea(child: Padding(padding: const EdgeInsets.all(8.0), child: Text(plan.completenessMessage, style: const TextStyle(fontWeight: FontWeight.bold, color: Colors.purple,), // TextStyle), // Text), // Padding), // SafeArea],); // Column },), // ValueListenableBuilder floatingActionButton: _buildAddTaskButton(context),); // Scaffold } ```

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?

   Jawab: Pada langkah 1, InheritedWidget berfungsi sebagai mekanisme untuk *menurunkan data ke widget turunan* tanpa harus meneruskan data melalui konstruktor secara manual. Dalam konteks praktikum ini, digunakan InheritedNotifier karena selain mewariskan data seperti InheritedWidget, ia juga dapat mendengarkan perubahan data (notifikasi) dari objek seperti ValueNotifier. Dengan demikian, ketika data dalam ValueNotifier berubah, semua widget yang bergantung padanya akan otomatis diperbarui tanpa perlu memanggil setState() secara manual, sehingga lebih efisien untuk manajemen state yang reaktif.

3. Jelaskan maksud dari method di langkah 3 pada praktikum tersebut! Mengapa dilakukan demikian?

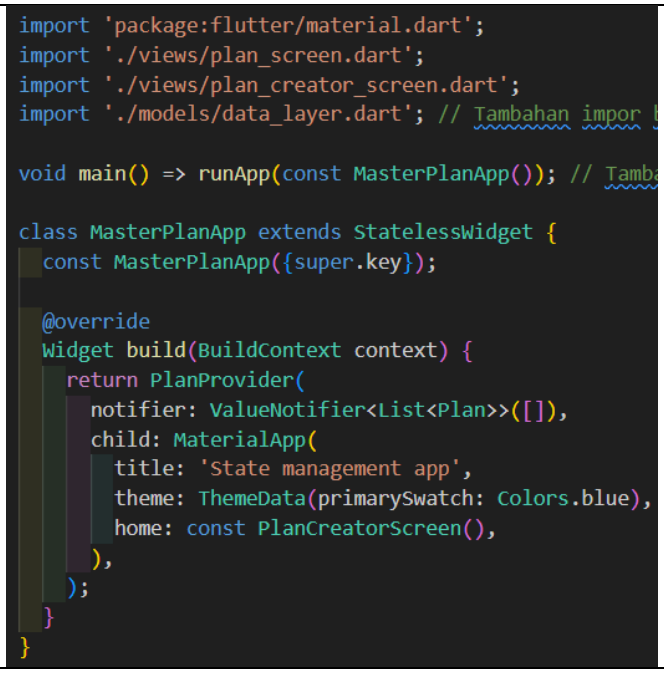
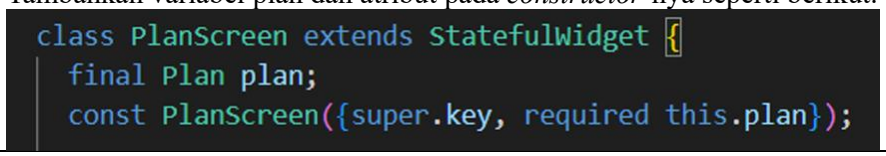
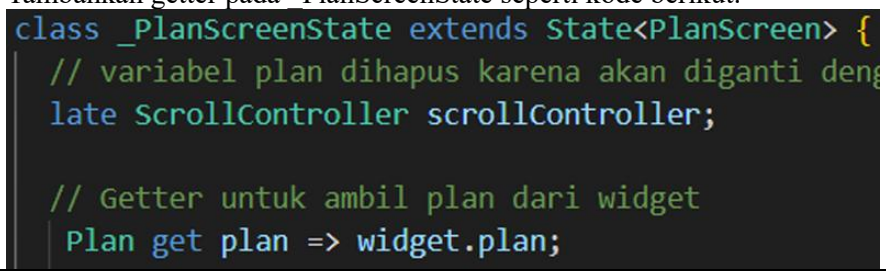
   Jawab: Method pada langkah 3 digunakan untuk mengambil instance PlanProvider dari context widget melalui PlanProvider.of(context). Tujuannya adalah agar widget lain dapat mengakses dan memanipulasi data Plan yang disimpan dalam ValueNotifier tanpa harus meneruskan variabel secara eksplisit antar widget. Hal ini dilakukan agar arsitektur aplikasi menjadi lebih rapi dan data Plan dapat dikelola secara global dalam satu sumber kebenaran (*single source of truth*).

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

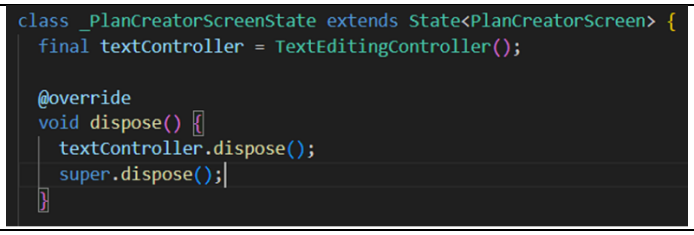
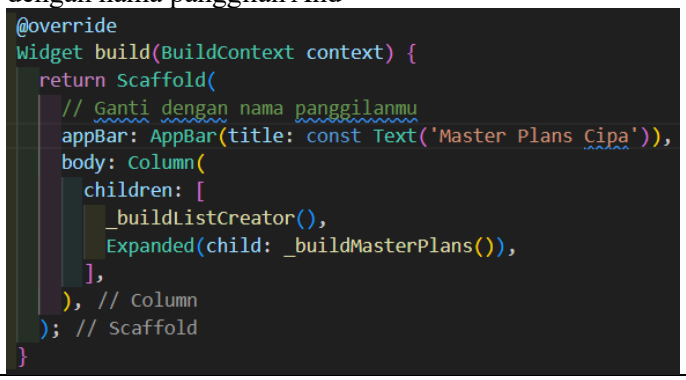
5. Kumpulkan laporan praktikum Anda berupa link commit atau repository GitHub ke dosen yang telah disepakati !

Praktikum 3 : Membuat State di Multiple Screens

No	Langkah - Langkah
1	Perhatikan kode berikut, edit class PlanProvider sehingga dapat menangani List Plan. lib > provider > plan_provider.dart `import 'package:flutter/material.dart';` `import '../models/data_layer.dart';` `class PlanProvider extends InheritedNotifier<ValueNotifier<List<Plan>>> {` `  const PlanProvider({` `  super.key,` `  required Widget child,` `  required ValueNotifier<List<Plan>> notifier,` `  }) : super(child: child, notifier: notifier);` `  static ValueNotifier<List<Plan>> of(BuildContext context) {` `  return context` `  .dependOnInheritedWidgetOfExactType<PlanProvider>()!` `  .notifier!;` `  }` `}`
2	Langkah sebelumnya dapat menyebabkan error pada main.dart dan plan_screen.dart. Pada method build, gantilah menjadi kode seperti ini.

	
3	Tambahkan variabel plan dan atribut pada *constructor*-nya seperti berikut. 
4	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 tersebut.
5	Tambahkan getter pada _PlanScreenState seperti kode berikut. 
6	Method **initState()** Pada bagian ini kode tetap seperti berikut. 
7	Widget **build** Pastikan Anda telah merubah ke List dan mengubah nilai pada currentPlan seperti kode berikut ini.

	``dart // Langkah 7: Widget build menggunakan List<Plan> dari PlanProvider @override Widget build(BuildContext context) { ValueNotifier<List<Plan>> plansNotifier = PlanProvider.of(context); return Scaffold(appBar: AppBar(title: Text(plan.name)), // gunakan getter plan body: ValueListenableBuilder<List<Plan>>(valueListenable: plansNotifier, builder: (context, plans, child) { // cari plan aktif berdasarkan nama Plan currentPlan = plans.firstWhere((p) => p.name == plan.name, orElse: () => plan,); return Column(children: [Expanded(child: _buildList(currentPlan)), SafeArea(child: Text(currentPlan.completenessMessage)),],); // Column },), // ValueListenableBuilder floatingActionButton: _buildAddTaskButton(context),); // Scaffold } ```
8	Edit **_buildTaskTile** Pastikan ubah ke List dan variabel planNotifier seperti kode berikut ini. ```dart // Langkah 8: Edit _buildTaskTile untuk mendukung List<Plan> 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); planNotifier.value = List<Plan>.from(planNotifier.value) ..[planIndex] = 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(hintText: 'Tulis deskripsi tugas...'), onChanged: (text) { Plan currentPlan = plan; int planIndex = planNotifier.value .indexWhere((p) => p.name == currentPlan.name); ```
9	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 home: const PlanCreatorScreen(),), // MaterialApp ```
10	Kita perlu tambahkan variabel TextEditingController sehingga bisa membuat TextField sederhana untuk menambah Plan baru. Jangan lupa tambahkan dispose ketika widget unmounted seperti kode berikut.

	
11	Letakkan method Widget build berikut di atas void dispose. Gantilah '**Namaku**' dengan nama panggilan And 
12	Buat widget **_buildListCreator** Buatlah widget berikut setelah widget build. 
13	Buat **void addPlan()** Tambahkan method berikut untuk menerima inputan dari user berupa text plan.

```dart
// Langkah 13: Menambah plan baru ke daftar PlanProvider
void addPlan() {
  final text = textController.text;
  if (text.isEmpty) return; // jika kosong, keluar

  final plan = Plan(name: text, tasks: []); // buat plan baru
  ValueNotifier<List<Plan>> planNotifier =
      PlanProvider.of(context) as ValueNotifier<List<Plan>>;

  // tambahkan ke list plan
  planNotifier.value = List<Plan>.from(planNotifier.value)..add(plan);

  // bersihkan input dan tutup keyboard
  textController.clear();
  FocusScope.of(context).requestFocus(FocusNode());

  setState(() {}); // perbarui tampilan
}

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

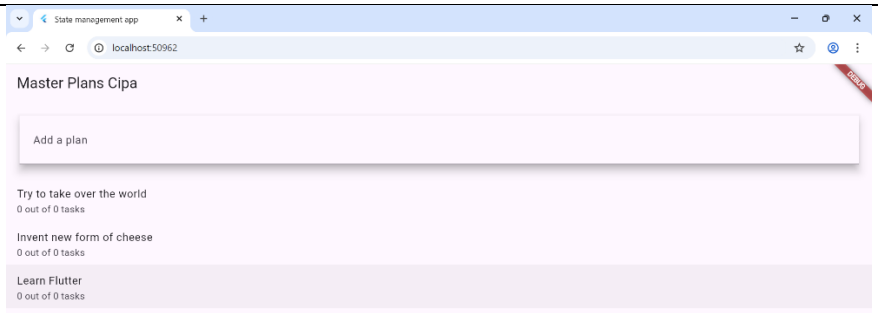
14 Buat **widget _buildMasterPlans()**
Tambahkan widget seperti kode berikut.

```dart
Widget _buildMasterPlans() {
  ValueNotifier<List<Plan>> planNotifier =
      PlanProvider.of(context) as ValueNotifier<List<Plan>>;

  return ValueListenableBuilder<List<Plan>>(
    valueListenable: planNotifier,
    builder: (context, plans, _) {
      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(
            title: Text(plan.name),
            subtitle: Text(plan.completenessMessage),
            onTap: () {
              Navigator.of(context).push(
                MaterialPageRoute(builder: (_) => PlanScreen(plan: plan)),
              );
            },
          ); // ListTile
```

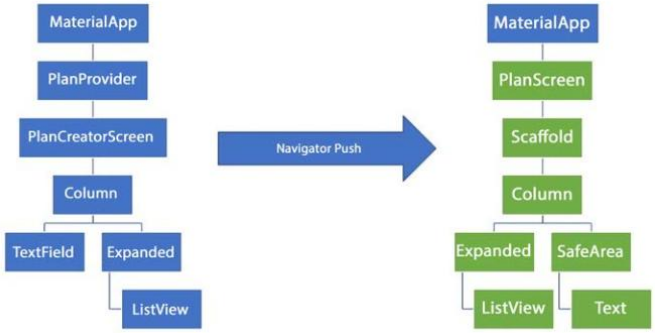
15 Terakhir, **run** atau tekan **F5** untuk melihat hasilnya jika memang belum running. Bisa juga lakukan **hot restart** jika aplikasi sudah running. Maka hasilnya akan seperti gambar berikut ini.



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!



Jawab: Secara keseluruhan, diagram ini menggambarkan bahwa aplikasi menggunakan mekanisme navigasi Navigator.push() untuk berpindah dari halaman pembuat rencana (PlanCreatorScreen) ke halaman detail rencana (PlanScreen). Perpindahan ini tidak hanya mengubah tampilan antar halaman, tetapi juga memperlihatkan bagaimana struktur widget dan data berpindah dari satu konteks ke konteks lainnya. Melalui pendekatan ini, setiap plan yang dipilih akan ditampilkan secara detail dengan daftar tugas di dalamnya, sehingga pengguna dapat melihat isi dan progres dari masing-masing rencana yang telah dibuat.

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

   Jawab: Pada langkah ini, saya berhasil menjalankan aplikasi Master Plan App dengan konsep state management menggunakan InheritedNotifier dan ValueNotifier melalui class PlanProvider. Aplikasi dapat menampilkan daftar rencana seperti “Try to take over the world”, “Invent New Form of Cheese”, dan “Learn Flutter” dengan tampilan utama bertema biru. Hasil ini menunjukkan bahwa sistem penyimpanan dan pembagian data antar widget sudah berjalan dengan baik, serta tampilan aplikasi sudah sesuai dengan rancangan yang diharapkan. https://github.com/sCipA19/Pemograman-Mobile/tree/main/src%20week-10/master_plan