LAPORAN PRAKTIKUM PEMROGRAMAN MOBILE

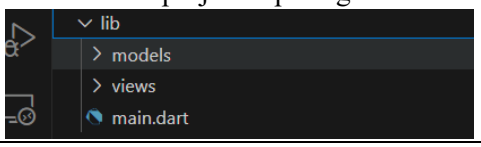
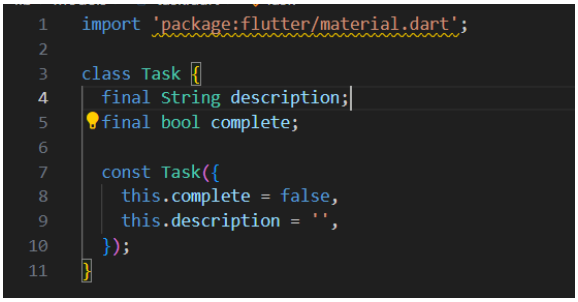
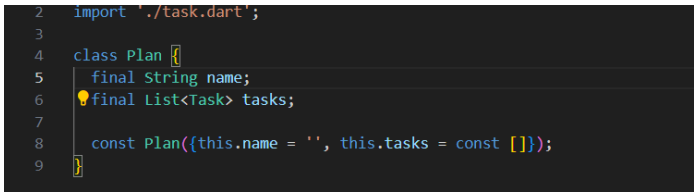
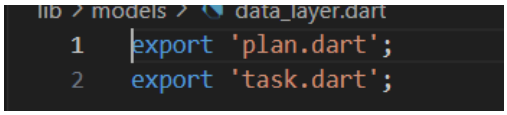
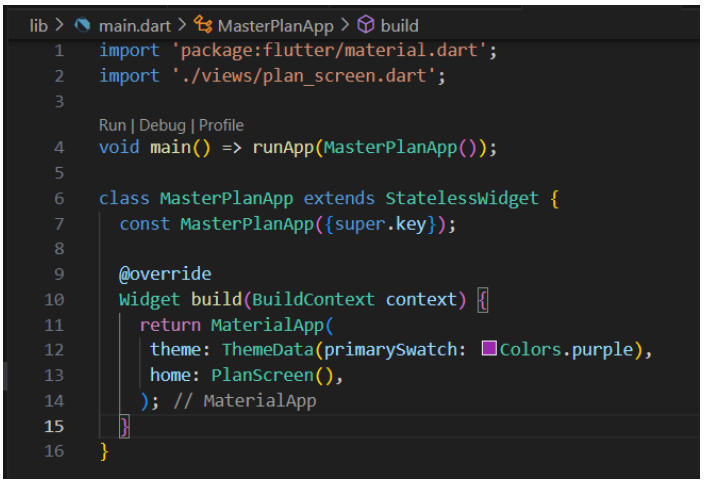
Pertemuan 10 – Dasar State Management

SIB 3C

Ismi Atika
2341760036

Jurusan Teknologi Informasi
Program Studi Sistem Informasi Bisnis
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 terbaik adalah memulai dari data layer agar struktur aplikasi lebih jelas. Di dalam folder model, buat file task.dart, lalu buat class Task dengan atribut description bertipe String dan complete bertipe bool 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.	Buat file data_layer.dart di folder models untuk mengekspor beberapa model sekaligus, sehingga proses impor menjadi lebih ringkas. 
5.	Ubah isi kode main.dart sebagai berikut. 

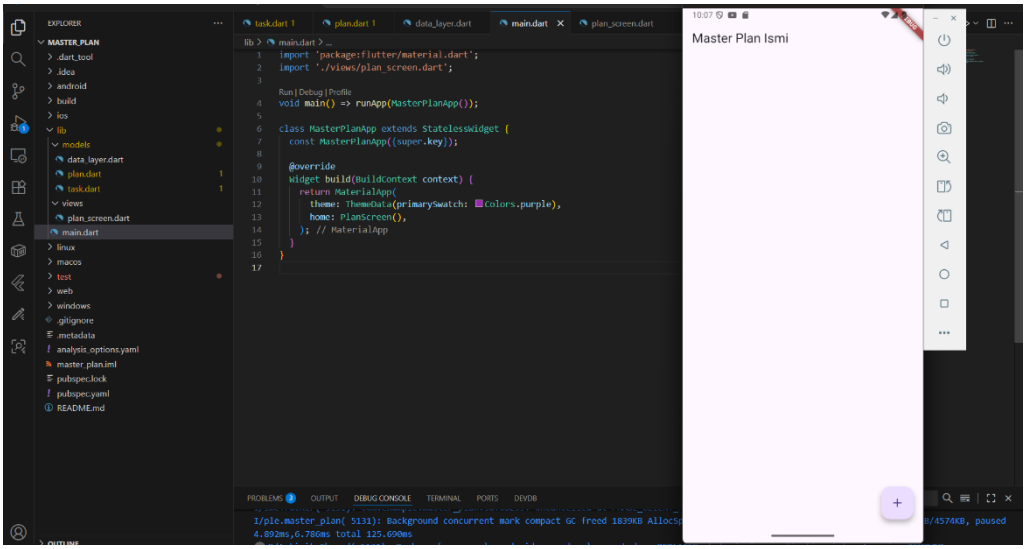
6.	Pada folder views, buatlah sebuah file plan_screen.dart dan gunakan templat StatefulWidget untuk membuat class PlanScreen. Isi kodenya adalah sebagai berikut. Gantilah teks ‘Namaku' dengan nama panggilan Anda pada title AppBar.
	lib > views > plan_screen.dart > _PlanScreenState > build
	`import '../models/data_layer.dart';`
	`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 Ismi')),`
	`  body: _buildList(),`
	`  floatingActionButton: _buildAddTaskButton(),`
	`  ); // Scaffold`
	`  }`
	`}`

7.	Akan muncul beberapa error di langkah 6 karena ada method yang belum dibuat. Mulailah dengan menambahkan kode tombol Tambah Rencana di bawah method build dalam class _PlanScreenState.
	`// Tambahkan method ini di bawah build()`
	`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`
	`  }`
	`}`

8.	Kita akan buat widget berupa List yang dapat dilakukan scroll, yaitu ListView.builder. Buat widget ListView seperti kode berikut ini.
	`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. Run atau tekan F5 untuk melihat hasil aplikasi yang Anda telah buat. Capture hasilnya untuk soal praktikum nomor 4.
10.	Anda bisa menambah dan menandai tugas, serta scroll saat daftar makin banyak. Namun di iOS, keyboard bisa menutupi input terbawah. Untuk mengatasinya, tambahkan ScrollController di file plan_screen.dart, tepat setelah variabel plan pada class State.

```dart
// Tampilan tiap tugas
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
        });
      },
      decoration: const InputDecoration(
        hintText: 'Masukkan tugas...',
      ), // InputDecoration
    ), // TextFormField
  ); // ListTile
}
}
```



```
class _PlanScreenState extends State<PlanScreen> {
  Plan plan = const Plan();
  late ScrollController scrollController;

```

11. Tambahkan method initState() setelah deklarasi variabel scrollController seperti kode berikut.

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

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

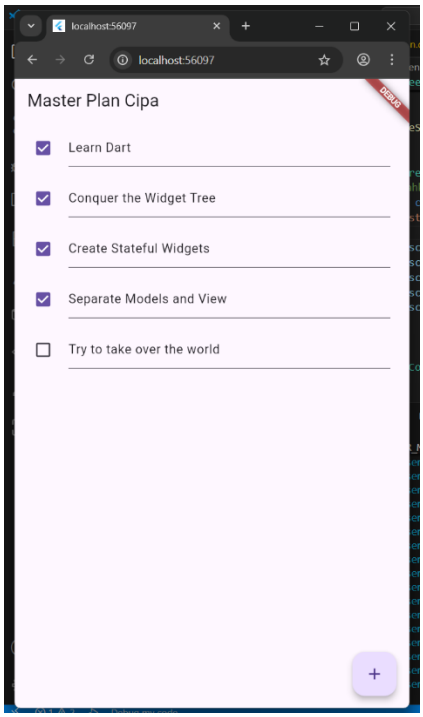
12. Tambahkan controller dan keyboard behavior pada ListView di method _buildList seperti kode berikut ini.

```
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),
  );
}
```

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

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

```

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.
   Jawab: Hasil akhir praktikum menampilkan aplikasi Master Plan Ismi dengan daftar tugas yang dapat ditandai selesai. Beberapa tugas sudah dicentang, dan satu masih aktif. Tampilan menggunakan tema ungu dengan tombol (+) di kanan bawah untuk menambah tugas. Aplikasi berjalan lancar tanpa error dan berhasil menerapkan konsep stateful widget serta model-view structure di Flutter.
   https://github.com/tugaskuliahh1/master_plan/blob/main/image/Hasil_1-3.mp4
2. Jelaskan maksud dari langkah 4 pada praktikum tersebut! Mengapa dilakukan demikian?
   Jawab: Langkah 4 bertujuan untuk membuat file data_layer.dart sebagai tempat menggabungkan semua file model agar proses import di file lain lebih sederhana. Dengan cara ini, cukup satu kali import untuk mengakses semua model tanpa perlu menulis banyak baris. Selain mempermudah pengelolaan kode, langkah ini juga membuat struktur project lebih rapi dan terorganisir.
3. Mengapa perlu variabel plan di langkah 6 pada praktikum tersebut? Mengapa dibuat konstanta ?
   Jawab: Variabel plan digunakan sebagai sumber data utama di PlanScreen untuk menampilkan dan mengelola tugas. Objeknya dibuat konstanta (const Plan()) agar memiliki nilai awal tetap, mencegah error saat pertama dijalankan, dan membuat proses rebuild lebih efisien.
4. Lakukan capture hasil dari Langkah 9 berupa GIF, kemudian jelaskan apa yang telah Anda buat!
   Jawab: Hasil langkah 9 menampilkan aplikasi Master Plan Ismi dengan tema ungu, judul di bagian atas, dan tombol (+) di kanan bawah untuk menambah tugas. Tampilan ini menunjukkan bahwa aplikasi sudah berhasil menampilkan halaman utama PlanScreen sesuai rancangan praktikum.
   https://github.com/tugaskuliahh1/master_plan/blob/main/image/Hasil_1-1.png
5. Apa kegunaan method pada Langkah 11 dan 13 dalam *lifecyle state* ?
   Jawab: Method pada langkah 11 (initState()) digunakan untuk inisialisasi awal ketika widget pertama kali dibuat, seperti menyiapkan controller atau listener. Sedangkan method pada langkah 13 (dispose()) berfungsi untuk membersihkan resource yang sudah tidak digunakan, agar tidak terjadi memory leak saat widget dihapus.
6. Kumpulkan laporan praktikum Anda berupa link commit atau repository GitHub ke dosen yang telah disepakati !
   Jawab: https://github.com/tugaskuliahh1/master_plan.git

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.
	`@override`
	`Widget build(BuildContext context) {`
	`  return MaterialApp(`
	`  theme: ThemeData(primarySwatch: Colors.purple),`
	`  // Bagian home diubah dari PlanScreen() menjadi PlanProvider(...)`
	`  home: PlanProvider(`
	`  notifier: ValueNotifier<Plan>(const Plan()),`
	`  child: const PlanScreen(),`
	`  ),`
	`  ); // MaterialApp`
	`  }`
	`}`

3.	Tambahkan dua *method* di dalam model class Plan seperti kode berikut.
	`// Tambahan baru: menghitung jumlah tugas yang sudah selesai`
	`int get completedCount => tasks`
	`  .where((task) => task.complete)`
	`  .length;`
	
	`// Tambahan baru: menampilkan pesan jumlah tugas selesai`
	`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.
	`class _PlanScreenState extends State<PlanScreen> {`
	`  // variabel plan dihapus karena akan diganti dengan data dari PlanProvider`
	`  late ScrollController scrollController;`
	
	`  @override`
	`  void initState() {`
	`  super.initState();`
	`  scrollController = ScrollController()`
	`  ..addListener(() {`
	`  FocusScope.of(context).requestFocus(FocusNode());`
	`  });`
	`  }`
	
	`  @override`
	`  void dispose() {`
	`  scrollController.dispose();`
	`  super.dispose();`
	`  }`
	
	`  @override`
	`  Widget build(BuildContext context) {`
	`  return Scaffold(`
	`  appBar: AppBar(title: const Text('Master Plan Ismi')),`
	`  body: _buildList(),`
	`  floatingActionButton: _buildAddTaskButton(),`
	`  ); // Scaffold`
	`  }`

```
Widget _buildAddTaskButton() {
  return FloatingActionButton(
    child: const Icon(Icons.add),
    // Bagian ini akan error sementara karena variabel plan sudah dihapus
    onPressed: () {
      setState(() {
        // nanti bagian ini diganti untuk ambil data dari PlanProvider
      });
    },
  ); // FloatingActionButton
}

Widget _buildList() {
  return ListView.builder(
    controller: scrollController,
    keyboardDismissBehavior: Theme.of(context).platform == TargetPlatform.iOS
        ? ScrollViewKeyboardDismissBehavior.onDrag
        : ScrollViewKeyboardDismissBehavior.manual,
    // ini juga akan error sementara karena plan belum ada
    itemCount: 0,
    itemBuilder: (context, index) {
      // nanti di Langkah 5 diganti agar ambil dari PlanProvider
      return const ListTile();
    },
  ); // ListView.builder
}
}
```

5. Tambahkan BuildContext sebagai parameter dan gunakan PlanProvider sebagai sumber datanya. Edit bagian kode seperti berikut.

```
// Langkah 5: method diperbarui agar menggunakan PlanProvider
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
}
```

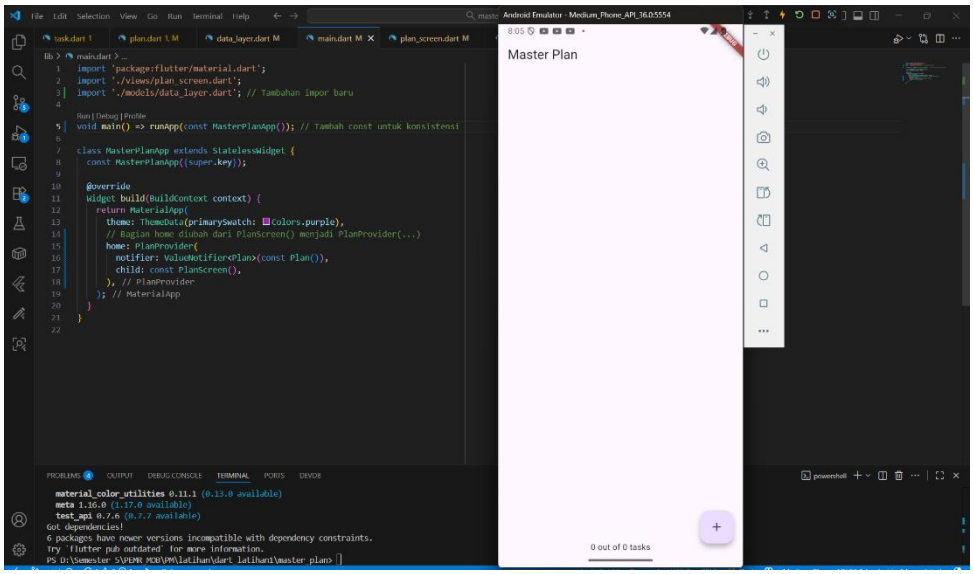
6. Tambahkan parameter BuildContext, gunakan PlanProvider sebagai sumber data.
Ganti TextField menjadi TextFormField untuk membuat inisial data provider menjadi lebih mudah.

```
// Langkah 6: method baru untuk menampilkan setiap task
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(
        hintText: 'Tulis deskripsi tugas...',
      ), // InputDecoration
      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,
            ), // Task
        ); // Plan
      },
    ), // TextFormField
  ); // ListTile
}
```

7.	Sesuaikan parameter pada bagian _buildTaskTile seperti kode berikut.
	`// Langkah 7: versi baru _buildList menerima Plan langsung`
	`Widget _buildList(Plan plan) {`
	`  return ListView.builder(`
	`  controller: scrollController,`
	`  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.
	`// Langkah 8: build diubah untuk menampilkan progress di bawah daftar`
	`@override`
	`Widget build(BuildContext context) {`
	`  ValueNotifier<Plan> planNotifier = PlanProvider.of(context);`
	`  return Scaffold(`
	`  appBar: AppBar(title: const Text('Master Plan Ismi')),`
	`  body: ValueListenableBuilder<Plan>(`
	`  valueListenable: planNotifier,`
	`  builder: (context, plan, _) {`
	`  return Column(`
	`  children: [`
	`  // daftar tugas`
	`  Expanded(child: _buildList(plan)),`
	`  // footer progress`
	`  Container(`
	`  padding: const EdgeInsets.all(16),`
	`  color: Colors.purple.shade50,`
	`  child: Text(`
	`  plan.completenessMessage,`
	`  style: const TextStyle(fontWeight: FontWeight.bold),`
	`  ), // Text`
	`  ), // Container`
	`  ],`
	`  ); // Column`
	`  },`
	`  ), // ValueListenableBuilder`
	`  floatingActionButton: _buildAddTaskButton(context),`
	`  ); // Scaffold`
	`  }`

9.	Terakhir, tambahkan widget SafeArea dengan berisi completenessMessage pada akhir widget Column. Perhatikan kode berikut ini.
	`@override`
	`Widget build(BuildContext context) {`
	`  return Scaffold(`
	`  appBar: AppBar(title: const Text('Master Plan')),`
	`  body: ValueListenableBuilder<Plan>(`
	`  valueListenable: PlanProvider.of(context),`
	`  builder: (context, plan, child) {`
	`  return Column(`
	`  children: [`
	`  // daftar tugas`
	`  Expanded(child: _buildList(plan)),`
	`  // footer progress`
	`  SafeArea(child: Text(plan.completenessMessage),),`
	`  ],`
	`  ); // Column`
	`  },`
	`  ), // ValueListenableBuilder`
	`  floatingActionButton: _buildAddTaskButton(context),`
	`  ); // Scaffold`
	`  }`
	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. Ini merupakan hal terpenting dalam mengelola state di aplikasi Anda.



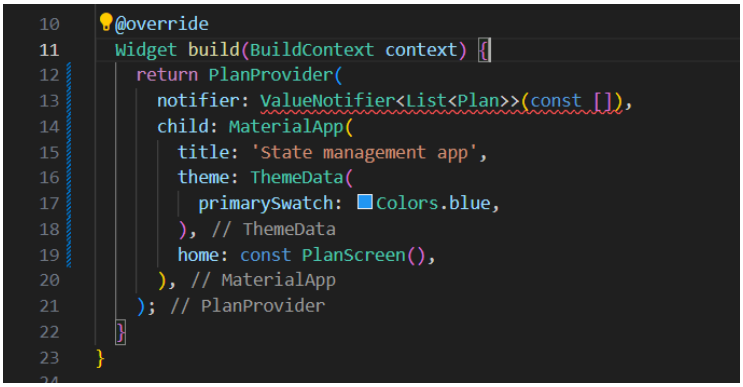
Tugas Praktikum 2: InheritedWidget

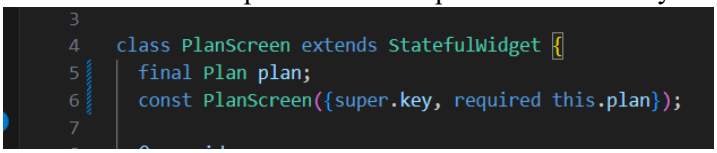
1. Selesaikan langkah-langkah praktikum tersebut, lalu dokumentasikan berupa GIF hasil akhir praktikum beserta penjelasannya di file README.md! Jika Anda menemukan ada yang error atau tidak berjalan dengan baik, silakan diperbaiki sesuai dengan tujuan aplikasi tersebut dibuat.
   Jawab: https://github.com/tugaskuliahh1/master_plan/blob/main/image/Hasil_2-1.mp4
2. Jelaskan mana yang dimaksud InheritedWidget pada langkah 1 tersebut! Mengapa yang digunakan InheritedNotifier?
   Jawab: InheritedWidget adalah widget yang membagikan data ke widget lain tanpa perlu lewat constructor. Pada praktikum ini digunakan InheritedNotifier karena selain membagikan data, widget ini juga dapat memberi tahu widget lain secara otomatis saat data berubah, sehingga tampilan langsung diperbarui tanpa perlu memanggil setState().
3. Jelaskan maksud dari method di langkah 3 pada praktikum tersebut! Mengapa dilakukan demikian?
   Jawab: Method completedCount menghitung jumlah tugas yang selesai, sedangkan completenessMessage menampilkan progres seperti “3 out of 5 tasks”. Keduanya dibuat agar aplikasi otomatis menampilkan kemajuan tugas tanpa perhitungan manual.
4. Lakukan capture hasil dari Langkah 9 berupa GIF, kemudian jelaskan apa yang telah Anda buat!
   Jawab: Hasil dari langkah 9 menampilkan aplikasi Master Plan yang sudah berhasil menampilkan tampilan utama dengan judul di bagian atas, tombol tambah (+) di kanan bawah, dan teks progres di bagian bawah bertuliskan “0 out of 0 tasks”. Tampilan ini menunjukkan bahwa struktur data Plan dan penggunaan InheritedNotifier telah berjalan dengan benar, di mana aplikasi siap menampilkan serta memperbarui daftar tugas secara dinamis saat data ditambahkan.
   https://github.com/tugaskuliahh1/master_plan/blob/main/image/Hasil_2.png
5. Kumpulkan laporan praktikum Anda berupa link commit atau repository GitHub ke dosen yang telah disepakati !
   Jawab: https://github.com/tugaskuliahh1/master_plan.git

Praktikum 3 - Membuat State di Multiple Screens

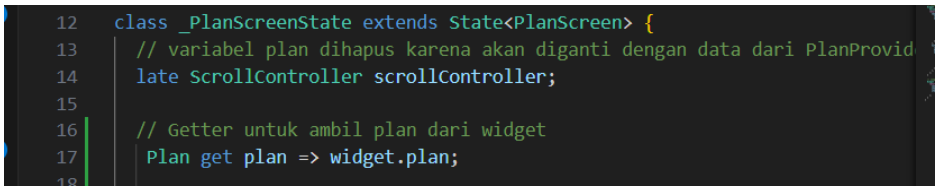
No.	Langkah - Langkah

1.	Perhatikan kode berikut, edit class PlanProvider sehingga dapat menangani List Plan.
	 
	lib > provider > plan_provider.dart > PlanProvider
	`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!;`
	`  }`
	`}`

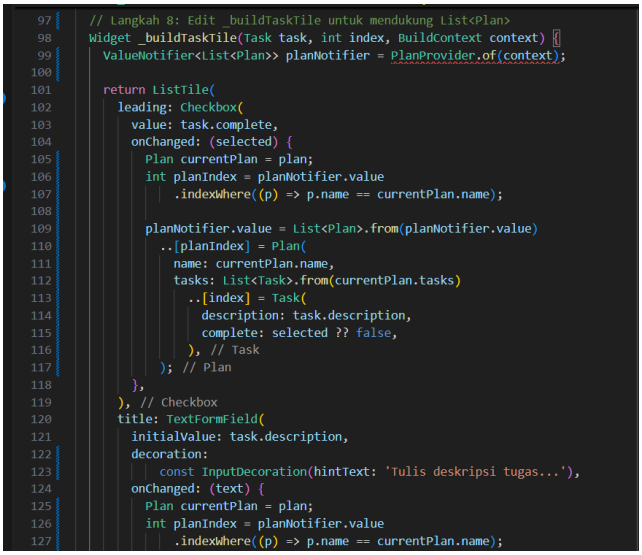
2.	Langkah sebelumnya dapat menyebabkan error pada main.dart dan plan_screen.dart. Pada method build, gantilah menjadi kode seperti ini.
	 
	`@override`
	`Widget build(BuildContext context) {`
	`  return PlanProvider(`
	`  notifier: ValueNotifier<List<Plan>>(const []),`
	`  child: MaterialApp(`
	`  title: 'State management app',`
	`  theme: ThemeData(`
	`  primarySwatch: Colors.blue,`
	`  ), // ThemeData`
	`  home: const PlanScreen(),`
	`  ), // MaterialApp`
	`  ); // PlanProvider`
	`  }`
	`}`

3.	Tambahkan variabel plan dan atribut pada *constructor*-nya seperti berikut.
	 
	`class PlanScreen extends StatefulWidget {`
	`  final Plan plan;`
	`  const PlanScreen({super.key, required this.plan});`

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.
	 
	`class _PlanScreenState extends State<PlanScreen> {`
	`  // variabel plan dihapus karena akan diganti dengan data dari PlanProvid`
	`  late ScrollController scrollController;`
	`  // Getter untuk ambil plan dari widget`
	`  Plan get plan => widget.plan;`

6.	Pada bagian ini kode tetap seperti berikut.
	 
	`@override`
	`void initState() {`
	`  super.initState();`
	`  scrollController = ScrollController()`
	`  ..addListener(() {`
	`  FocusScope.of(context).requestFocus(FocusNode());`
	`  });`
	`}`

7.	Pastikan Anda telah merubah ke List dan mengubah nilai pada currentPlan seperti kode berikut ini.  
8.	Pastikan ubah ke List dan variabel planNotifier 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
}
```

```dart
// Tombol untuk menambah task baru
Widget _buildAddTaskButton(BuildContext context) {
  ValueNotifier<List<Plan>> planNotifier = PlanProvider.of(context);

  return FloatingActionButton(
    child: const Icon(Icons.add),
    onPressed: () {
      Plan currentPlan = plan;

      // cari posisi plan aktif di list
      int planIndex =
          planNotifier.value.indexWhere((p) => p.name == currentPlan.name);

      // buat daftar task baru dengan tambahan task kosong
      List<Task> updatedTasks =
          List<Task>.from(currentPlan.tasks)..add(const Task());

      // update plan di posisi yang benar
      planNotifier.value = List<Plan>.from(planNotifier.value)
        ..[planIndex] = Plan(
          name: currentPlan.name,
          tasks: updatedTasks,
        ); // Plan

      // update juga plan lokal
      setState(() {
        currentPlan = Plan(
          name: currentPlan.name,
          tasks: updatedTasks,
        ); // Plan
      });
    },
  ); // FloatingActionButton
}
```

```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);
```

```
          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: text,
                  complete: task.complete,
                ), // Task
            ); // Plan
        },
      ), // TextFormField
    ); // ListTile
  }
```

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.

```
      ),| // ThemeData
      home: const PlanCreatorScreen(),
    ), // MaterialApp
  ); // PlanProvider
```

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.

```
class _PlanCreatorScreenState extends State<PlanCreatorScreen> {
  final textController = TextEditingController();

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

11. Letakkan method Widget build berikut di atas void dispose. Gantilah '**Namaku**' dengan nama panggilan Anda.

```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      // 🔸 Ganti "Ismi" dengan nama panggilanmu
      appBar: AppBar(title: const Text('Master Plans Ismi')),
      body: Column(
        children: [
          _buildListCreator(),
          Expanded(child: _buildMasterPlans()),
        ],
      ), // Column
    ); // Scaffold
  }
```

12. Buatlah widget berikut setelah widget build.

```
  // Langkah 12: Widget input 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, // dipanggil ketika tekan Enter
        ),| // TextField
      ), // Material
    ); // Padding
  }
```

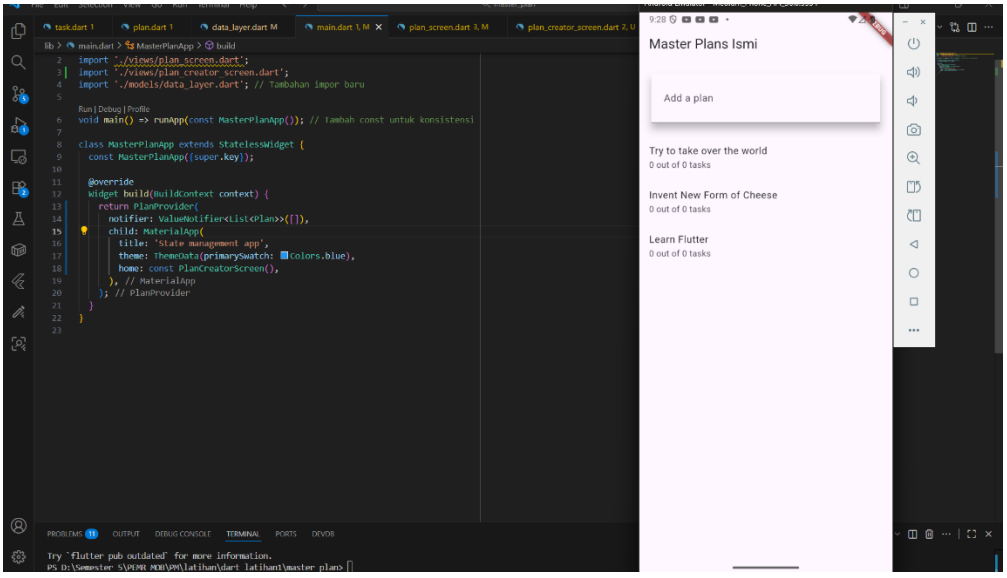
13. Tambahkan method berikut untuk menerima inputan dari user berupa text plan.

```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);

  // 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
}
```

14. Tambahkan widget seperti kode berikut.

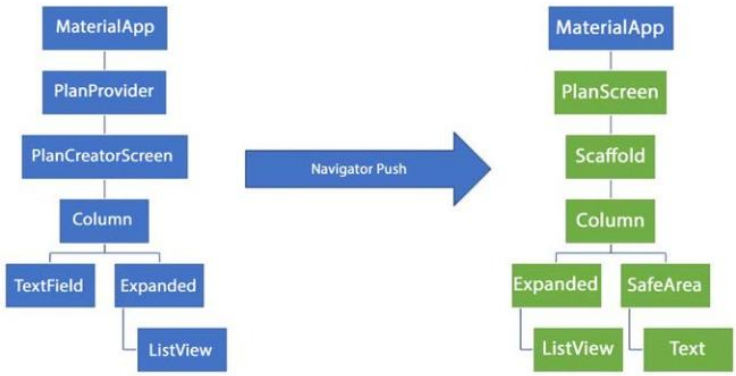
```dart
// Langkah 14: Widget untuk menampilkan daftar plan
Widget _buildMasterPlans() {
  ValueNotifier<List<Plan>> planNotifier = PlanProvider.of(context);

  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),
                ), // MaterialPageRoute
              );
            },
          ); // ListTile
```

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.
   Jawab: https://github.com/tugaskuliahh1/master_plan/blob/main/image/Hasil_3-1.mp4
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/tugaskuliahh1/master_plan/blob/main/image/Hasil_3.png
4. Kumpulkan laporan praktikum Anda berupa link commit atau repository GitHub ke dosen yang telah disepakati !
   Jawab: https://github.com/tugaskuliahh1/master_plan.git