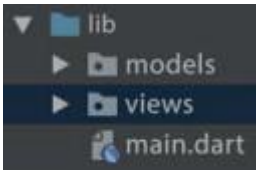
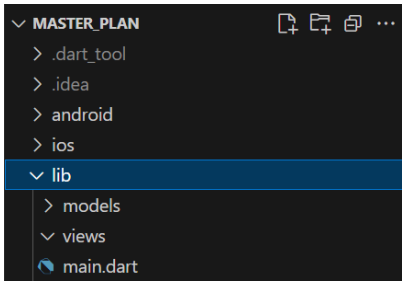
3. Praktikum 1: Dasar State dengan Model-View

Selesaikan langkah-langkah praktikum berikut ini menggunakan editor Visual Studio Code (VS Code) atau Android Studio atau code editor lain kesukaan Anda.

Langkah 1: Buat Project Baru

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.

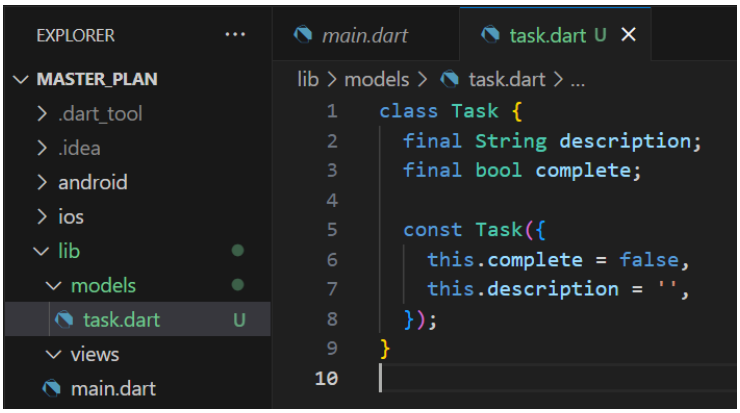




Langkah 2: Membuat model **task.dart**

Praktik terbaik untuk memulai adalah pada lapisan data (*data layer*). Ini akan memberi Anda gambaran yang jelas tentang aplikasi Anda, tanpa masuk ke detail antarmuka pengguna Anda. Di folder model, 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. Tambahkan kode berikut:

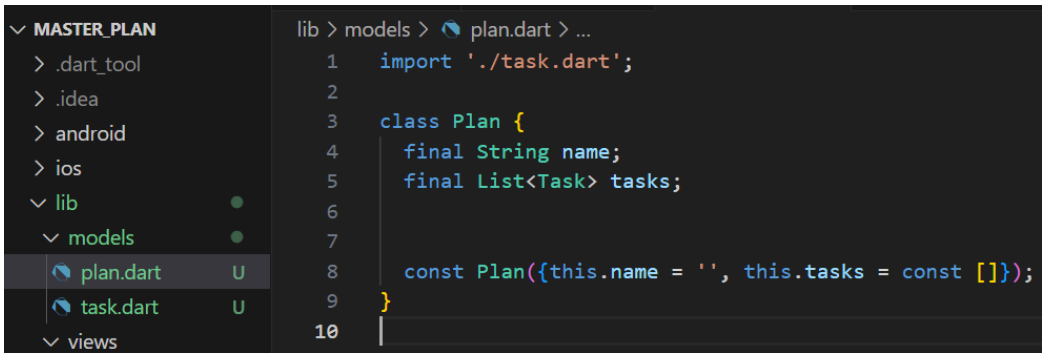
```
class Task {
  final String description;
  final bool complete;

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



Langkah 3: Buat file **plan.dart**

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.

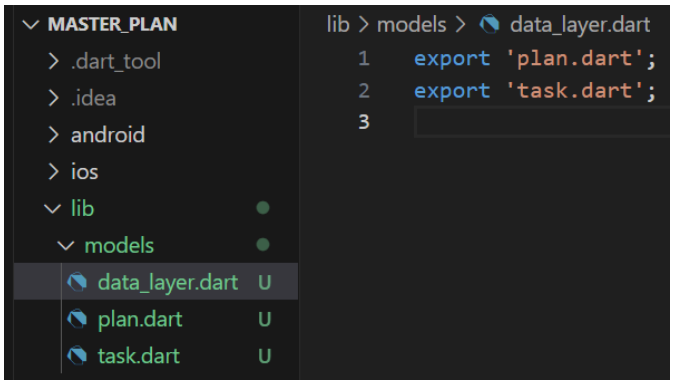
```
import './task.dart';

class Plan {
  final String name;
  final List<Task> tasks;
  const Plan({this.name = '', this.tasks = const []});
}
```



Langkah 4: Buat file **data_layer.dart**

Kita dapat membungkus beberapa data layer ke dalam sebuah file yang nanti akan mengekspor kedua model tersebut. Dengan begitu, proses impor akan lebih ringkas seiring berkembangnya aplikasi. Buat file bernama data_layer.dart di folder **models**. Kodenya hanya berisi export seperti berikut.

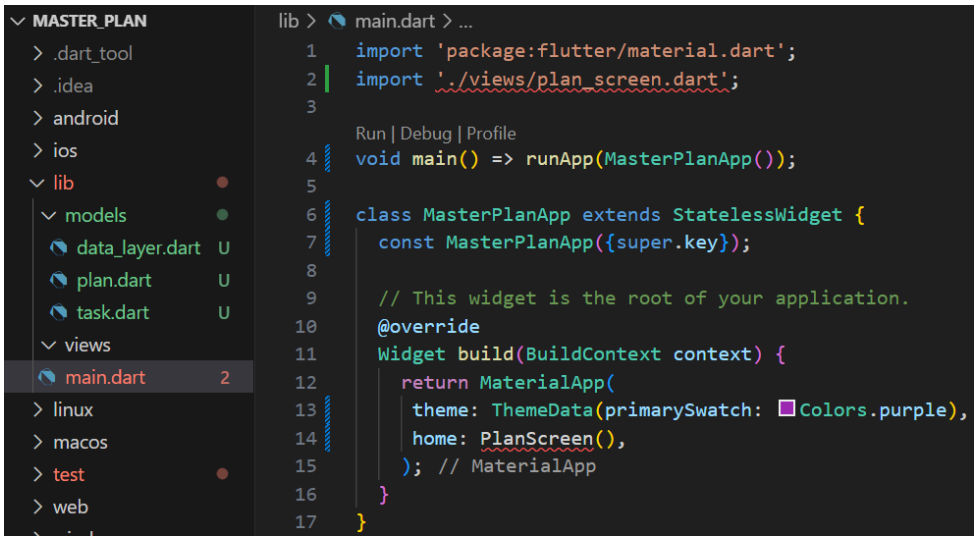
```
export 'plan.dart';
export 'task.dart';
```



Langkah 5: Pindah ke file **main.dart**

Ubah isi kode main.dart sebagai berikut.

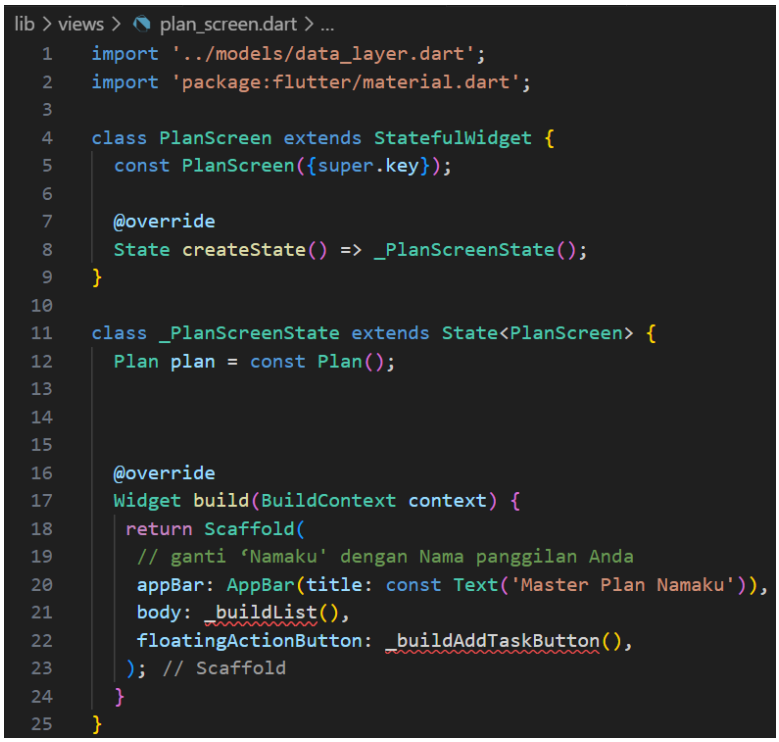
```
import 'package:flutter/material.dart';
import './views/plan_screen.dart';

void main() => runApp(MasterPlanApp());

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

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



Langkah 6: buat **plan_screen.dart**

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 .

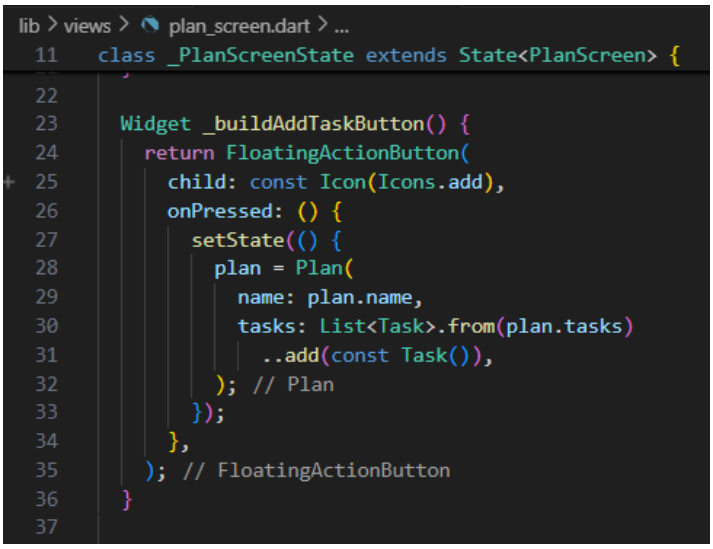
```
import '../models/data_layer.dart'; import
'package:flutter/material.dart';

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 Namaku')),
    body: _buildList(),
    floatingActionButton: _buildAddTaskButton(),
   );
  }
}
```



Langkah 7: buat method **_buildAddTaskButton()**

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 .

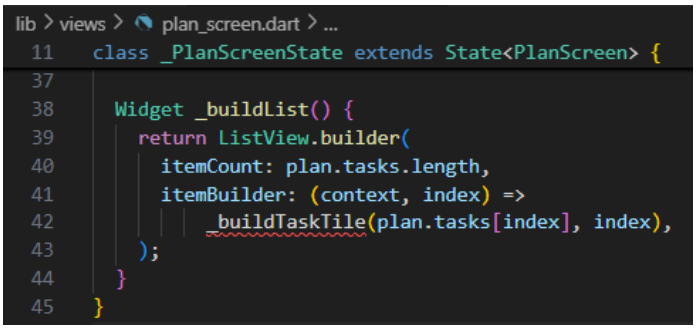
```
Widget _buildAddTaskButton() {
  return FloatingActionButton(
  child: const Icon(Icons.add),
  onPressed: () {
    setState(() {
     plan = Plan(
      name: plan.name,
      tasks: List<Task>.from(plan.tasks)
      ..add(const Task()),
     );
    });
   },
  );
}
```



Langkah 8: buat widget **_buildList()**

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),
  );
}
```



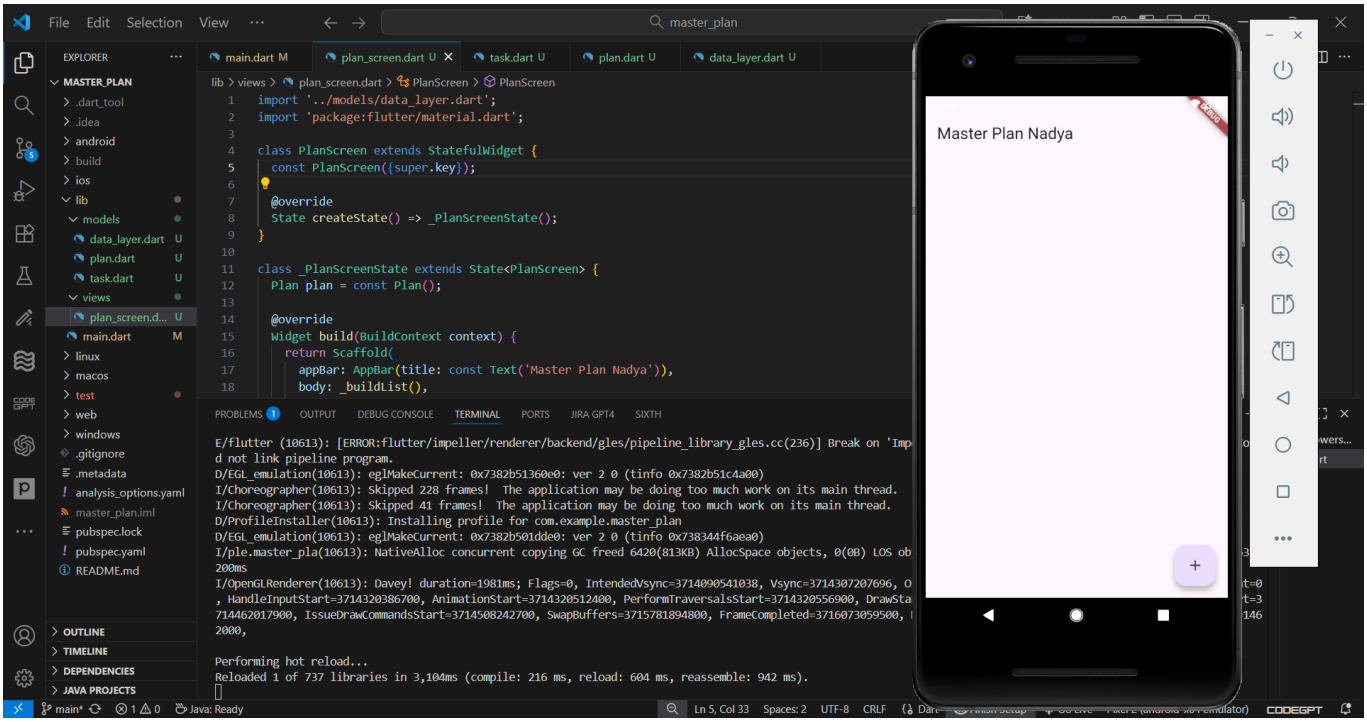
Langkah 9: buat widget **_buildTaskTile**

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.

```
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,
              ),
          );
        });
      }),
      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,
              ),
          );
```

```
        });
      },
    ),
  );
}
```

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



Langkah 10: Tambah Scroll Controller

Anda dapat menambah tugas sebanyak-banyaknya, menandainya jika sudah beres, dan melakukan scroll jika sudah semakin banyak isinya. Namun, ada salah satu fitur tertentu di iOS perlu kita tambahkan. Ketika keyboard tampil, Anda akan kesulitan untuk mengisi yang paling bawah. Untuk mengatasi itu, Anda dapat menggunakan ScrollController untuk menghapus focus dari semua TextField selama event scroll dilakukan. Pada file plan_screen.dart , tambahkan variabel scroll controller di class State tepat setelah variabel plan .

```
late ScrollController scrollController;
```

lib > views > plan_screen.dart > ...

```
10
11  class _PlanScreenState extends State<PlanScreen> {
12    Plan plan = const Plan();
13    late ScrollController scrollController;
14
```

Langkah 11: Tambah Scroll Listener

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

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

lib > views > plan_screen.dart > ...

```
15    @override
16    void initState() {
17      super.initState();
18      scrollController = ScrollController()
19        ..addListener(() {
20          FocusScope.of(context).requestFocus(FocusNode());
21        });
22    }
```

Langkah 12: Tambah controller dan keyboard behavior

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

```
return ListView.builder(
  controller: scrollController,
 keyboardDismissBehavior: Theme.of(context).platform ==
 TargetPlatform.iOS
        ? ScrollViewKeyboardDismissBehavior.onDrag
        : ScrollViewKeyboardDismissBehavior.manual,
```

```
lib > views > plan_screen.dart > ...
11  class _PlanScreenState extends State<PlanScreen> {
47
48    Widget _buildList() {
49      return ListView.builder(
50        controller: scrollController,
51        keyboardDismissBehavior: Theme.of(context).platform ==
52                TargetPlatform.iOS
53            ? ScrollViewKeyboardDismissBehavior.onDrag
54            : ScrollViewKeyboardDismissBehavior.manual,
55        itemCount: plan.tasks.length,
56        itemBuilder: (context, index) =>
57            _buildTaskTile(plan.tasks[index], index),
58      );
59    }
```

Langkah 13: Terakhir, tambah method dispose()

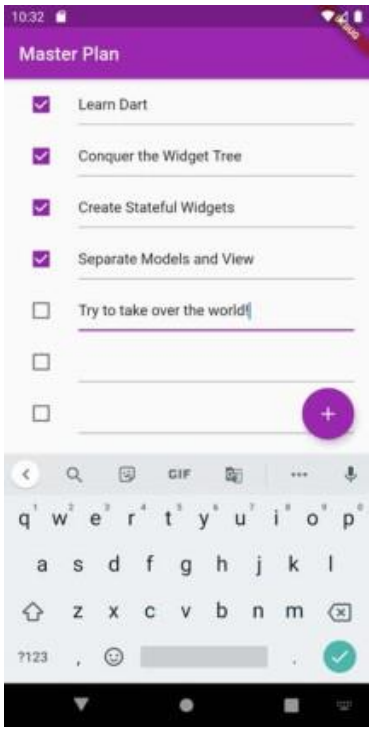
Terakhir, tambahkan method `dispose()` berguna ketika widget sudah tidak digunakan lagi.

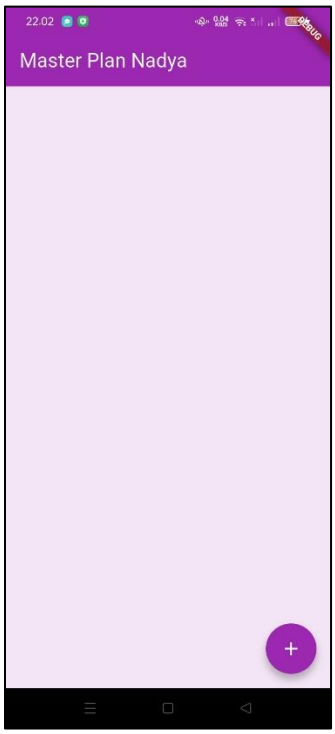
```
@override
void dispose() {
  scrollController.dispose();
  super.dispose();
}
```

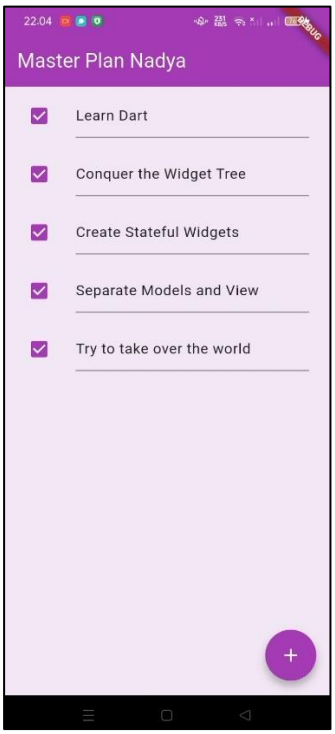
```
lib > views > plan_screen.dart > ...
11  class _PlanScreenState extends State<PlanScreen> {
95
96    @override
97    void dispose() {
98      scrollController.dispose();
99      super.dispose();
100   }
101 }
102
```

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.







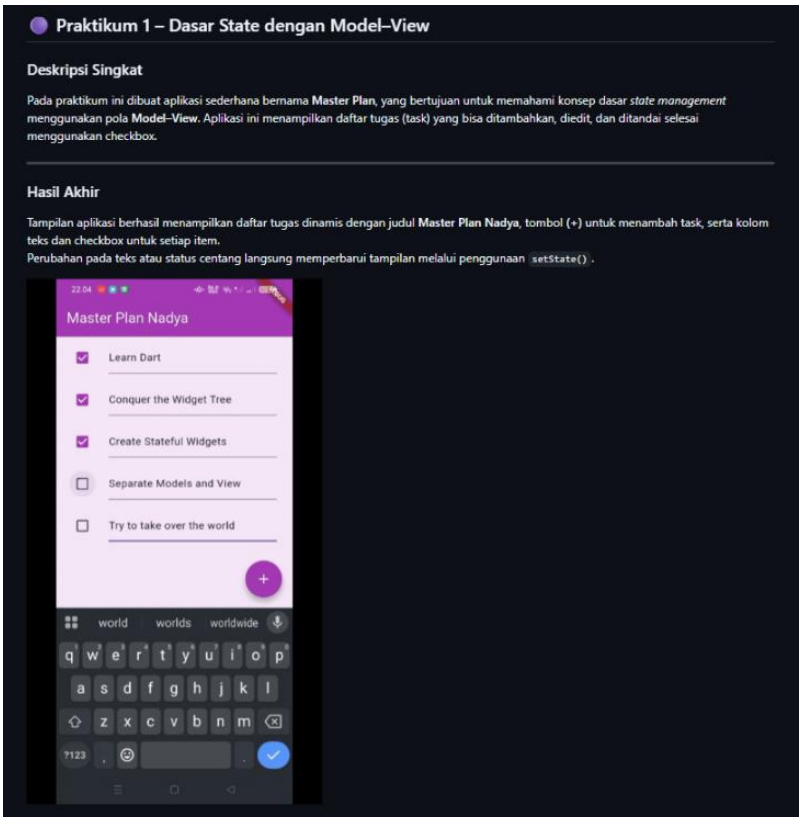
> **Catatan:** Kedua fitur hot reload dan hot restart memiliki performa lebih cepat dibanding melakukan build ulang secara keseluruhan aplikasi. Secara umum:
>
> - Gunakan *hot reload* untuk melihat perubahan pada tampilan UI, jadi perubahan paling banyak terjadi di metode *build*. State pada aplikasi tetap dipertahankan dan Anda akan melihat perubahannya hampir secara instan.
> - Gunakan *hot restart* untuk melihat perubahan pada state aplikasi, seperti memperbarui **variabel global**, **static fields**, atau metode `main()` .

4. 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?
3. Mengapa perlu variabel plan di langkah 6 pada praktikum tersebut? Mengapa dibuat konstanta ?
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* ?
6. Kumpulkan laporan praktikum Anda berupa link commit atau repository GitHub ke dosen yang telah disepakati !

Jawaban:

1. Tampilan README.md dengan dokumentasi berupa GIF



2. Pada langkah 4 dibuat file data_layer.dart yang berisi perintah export untuk menghubungkan file plan.dart dan task.dart. Tujuannya supaya saat file lain membutuhkan model data, kita cukup mengimpor satu file saja tanpa harus memanggil keduanya secara terpisah. Cara ini membuat struktur project jadi lebih rapi, mudah dikelola, dan efisien ketika aplikasi semakin besar.
3. Variabel plan pada langkah 6 berfungsi untuk menyimpan data utama aplikasi, yaitu daftar rencana dan tugas yang dibuat oleh pengguna. Variabel ini dibuat dengan konstanta (const) karena datanya bersifat tetap dan hanya bisa diubah dengan membuat objek baru melalui setState(). Hal ini dilakukan agar state aplikasi lebih aman, tidak mudah berubah sembarangan, dan mengikuti konsep immutable data pada Flutter.
4. Pada langkah 9, kita membuat tampilan daftar tugas menggunakan ListTile yang berisi checkbox dan kolom input teks. Checkbox digunakan untuk menandai apakah tugas sudah selesai, sedangkan kolom teks untuk mengedit atau menulis deskripsi tugas. Setiap kali pengguna mencentang atau mengetik sesuatu, method setState() dipanggil supaya tampilan daftar tugas ikut diperbarui secara otomatis.
5. Method initState() digunakan untuk menyiapkan komponen yang dibutuhkan saat pertama kali halaman ditampilkan, seperti inisialisasi ScrollController agar keyboard tertutup saat layar digulir. Sedangkan dispose() digunakan untuk membersihkan controller tersebut ketika halaman ditutup supaya tidak terjadi kebocoran memori. Keduanya merupakan bagian dari lifecycle widget yang memastikan aplikasi berjalan lebih efisien dan stabil.

5. Praktikum 2: Mengelola Data Layer dengan InheritedWidget dan InheritedNotifier

Bagaimana seharusnya Anda mengakses data pada aplikasi?

Beberapa pilihan yang bisa dilakukan adalah meletakkan data dalam satu kelas yang sama sehingga menjadi bagian dari life cycle aplikasi Anda. Kemudian muncul pertanyaan, bagaimana meletakkan model dalam pohon widget ? sedangkan model bukanlah widget, sehingga tidak akan tampil pada screen.

Solusi yang memungkinkan adalah menggunakan `InheritedWidget` . Sejauh ini kita hanya menggunakan dua jenis widget, yaitu `StatelessWidget` dan `StatefulWidget` . Kedua widget tersebut digunakan untuk layouting UI di screen. Di mana satu bersifat statis dan dinamis. Sedangkan `InheritedWidget` itu berbeda, ia dapat meneruskan data ke sub-widget turunannya (biasanya ketika Anda menerapkan *decomposition widget*). Jika dilihat dari perspektif user, itu tidak akan terlihat prosesnya (*invisible*). `InheritedWidget` dapat digunakan sebagai pintu untuk komunikasi antara **view** dan **data** layers.

Pada codelab ini, kita akan memperbarui kode dari aplikasi Master Plan dengan memisahkan data todo list ke luar class view-nya. Setelah Anda menyelesaikan praktikum 1, Anda dapat melanjutkan praktikum 2 ini. Selesaikan langkah-langkah praktikum berikut ini menggunakan editor Visual Studio Code (VS Code) atau Android Studio atau code editor lain kesukaan Anda.

Langkah 1: Buat file **plan_provider.dart**

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

```
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!;
  }
}
```

lib > provider > plan_provider.dart

```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!;
  }
}
```

Langkah 2: Edit **main.dart**

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(),
   ),
);
```

lib > main.dart

```dart
import 'package:flutter/material.dart';
import './views/plan_screen.dart';
import './provider/plan_provider.dart';
import './models/data_layer.dart';

void main() => runApp(MasterPlanApp());

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

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

Langkah 3: Tambah method pada model **plan.dart**

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

```
int get completedCount => tasks
  .where((task) => task.complete)
  .length;

String get completenessMessage =>
  '$completedCount out of ${tasks.length} tasks';
```

lib > models > plan.dart

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

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

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

  int get completedCount => tasks
      .where((task) => task.complete)
      .length;

  String get completenessMessage =>
      '$completedCount out of ${tasks.length} tasks';
}
```

Langkah 4: Pindah ke PlanScreen

Edit PlanScreen agar menggunakan data dari PlanProvider . Hapus deklarasi variabel plan (ini akan membuat error). Kita akan perbaiki pada langkah 5 berikut ini.
Menghapus bagian yang saya blok.

lib > views > plan_screen.dart > _PlanScreenState

```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();
  late ScrollController scrollController;

```

Langkah 5: Edit method **_buildAddTaskButton**

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

```
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()),
      );
    },
  );
}
```

lib > views > plan_screen.dart > _PlanScreenState

```
41    Widget _buildAddTaskButton(BuildContext context) {
42      ValueNotifier<Plan> planNotifier = PlanProvider.of(context);
43      return FloatingActionButton(
44        backgroundColor: Colors.purple,
45        shape: const CircleBorder(),
46        child: const Icon(Icons.add, color: Colors.white),
47        onPressed: () {
48          Plan currentPlan = planNotifier.value;
49          planNotifier.value = Plan(
50            name: currentPlan.name,
51            tasks: List<Task>.from(currentPlan.tasks)..add(const Task()),
52          );
53        },
54      );
55    }
```

Langkah 6: Edit method **_buildTaskTile**

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

```
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,
            ),
        );
      }),
    title: TextFormField( initialValue:
      task.description, 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,
```

```
          ),
        );
      },
    ),
  );
}
```

lib > views > plan_screen.dart

```
12  class _PlanScreenState extends State<PlanScreen> {
25    Widget build(BuildContext context) {
26      return Scaffold(
```

```
66    Widget _buildTaskTile(Task task, int index, BuildContext context) {
67      ValueNotifier<Plan> planNotifier = PlanProvider.of(context);
68      return ListTile(
69        leading: Checkbox(
70          activeColor: Colors.purple,
71          value: task.complete,
72          onChanged: (selected) {
73            Plan currentPlan = planNotifier.value;
74            planNotifier.value = Plan(
75              name: currentPlan.name,
76              tasks: List<Task>.from(currentPlan.tasks)
77                ..[index] = Task(
78                  description: task.description,
79                  complete: selected ?? false,
80                ),
81            );
82          },
83        ),
84        title: TextFormField(
85          initialValue: task.description,
86          onChanged: (text) {
87            Plan currentPlan = planNotifier.value;
88            planNotifier.value = Plan(
89              name: currentPlan.name,
90              tasks: List<Task>.from(currentPlan.tasks)
91                ..[index] = Task(
92                  description: text,
93                  complete: task.complete,
94                ),
95            );
96          },
97        ),
98      );
99    }
```

Langkah 7: Edit **_buildList**

Sesuaikan parameter pada bagian _buildTaskTile seperti kode berikut.

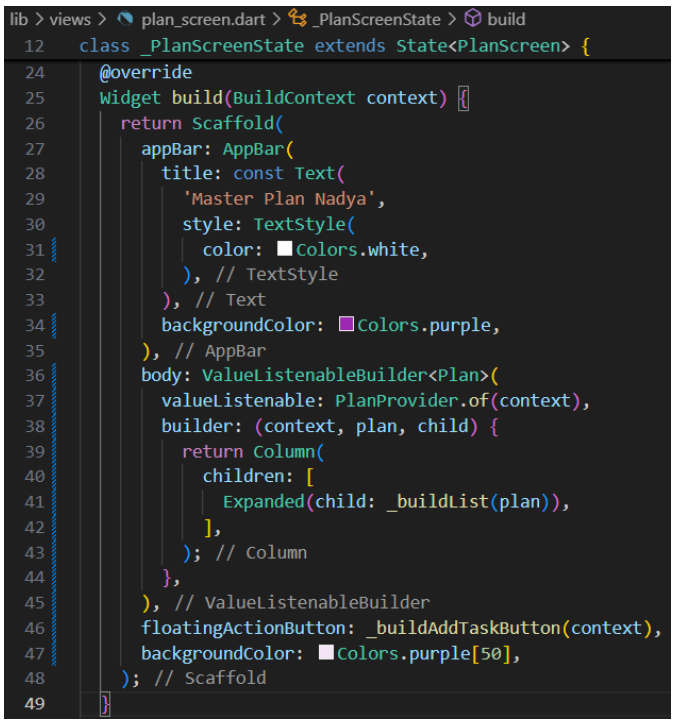
```
Widget _buildList(Plan plan) {
  return ListView.builder(
  controller: scrollController,
  itemCount: plan.tasks.length,
  itemBuilder: (context, index) =>
      _buildTaskTile(plan.tasks[index], index, context),
  );
}
```

lib > views > plan_screen.dart > _PlanScreenState > _buildTaskTile

```
12  class _PlanScreenState extends State<PlanScreen> {
58    Widget _buildList(Plan plan) {
59      return ListView.builder(
60        controller: scrollController,
61        itemCount: plan.tasks.length,
62        itemBuilder: (context, index) =>
63            _buildTaskTile(plan.tasks[index], index, context),
64      );
65    }
```

Langkah 8: Tetap di **class PlanScreen**

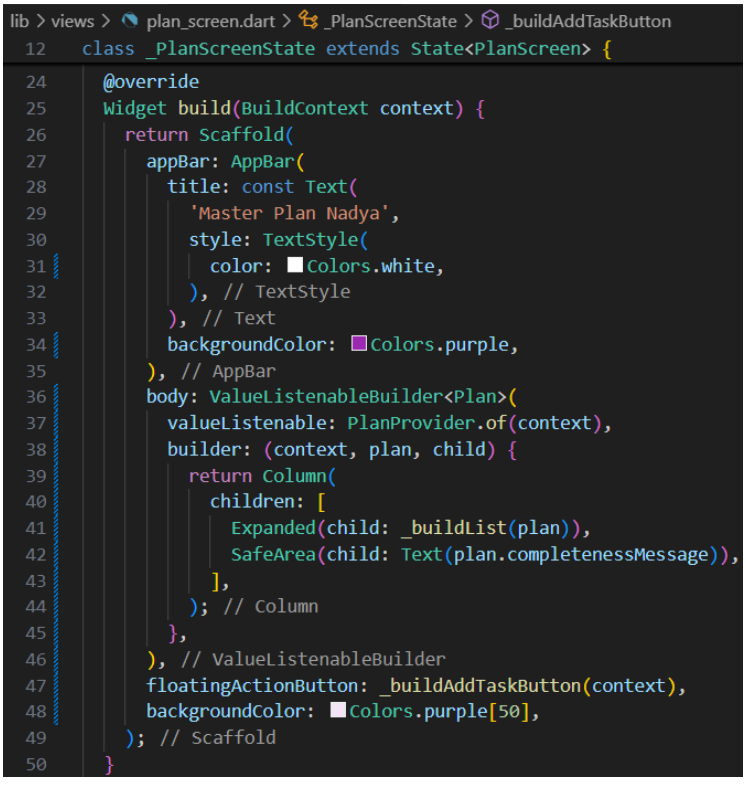
Edit method build sehingga bisa tampil progress pada bagian bawah (footer). Caranya, bungkus (wrap) _buildList dengan widget Expanded dan masukkan ke dalam widget Column seperti kode pada Langkah 9.



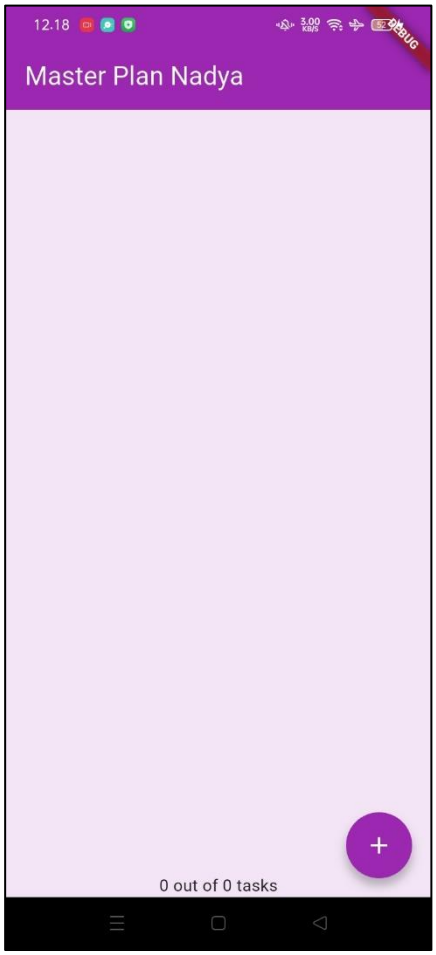
Langkah 9: Tambah widget **SafeArea**

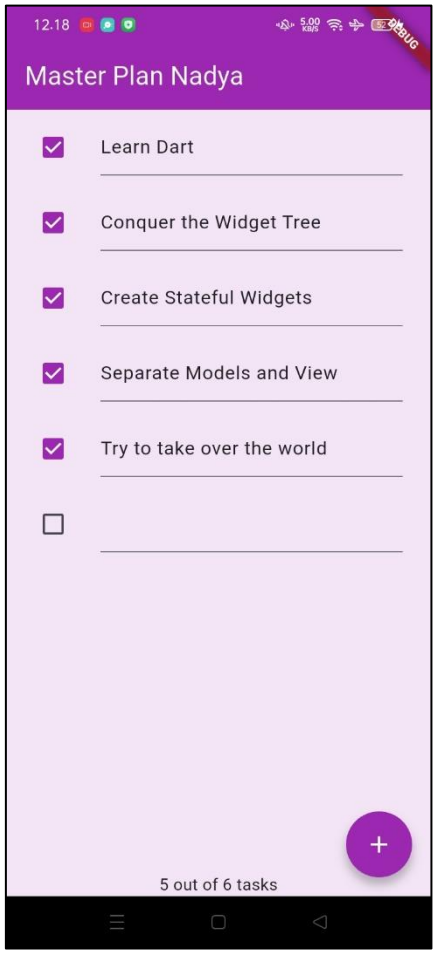
Terakhir, tambahkan widget SafeArea dengan berisi completenessMessage pada akhir widget Column . Perhatikan kode berikut ini.

```
@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: [
          Expanded(child: _buildList(plan)),
          SafeArea(child: Text(plan.completenessMessage))
        ],
      );
    },
    ),
    floatingActionButton: _buildAddTaskButton(context),
  );
}
```



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.





6. Tugas Praktikum 2: InheritedWidget

a. 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.

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

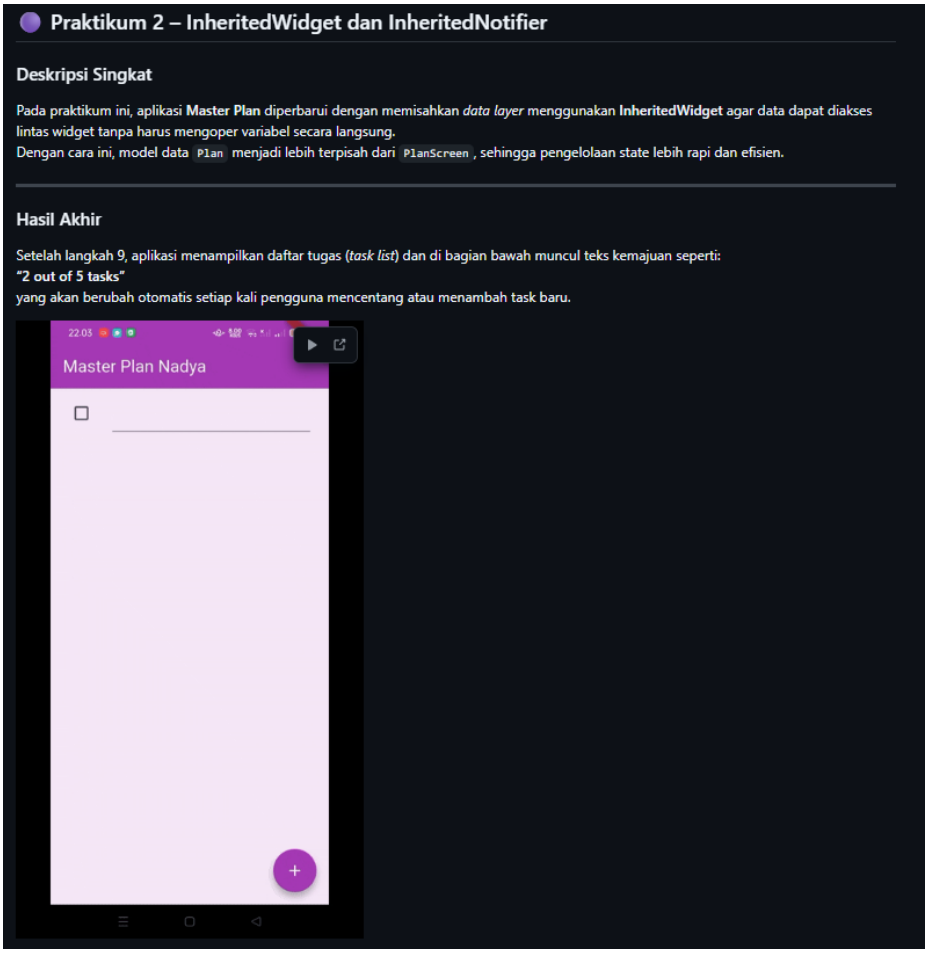
c. Jelaskan maksud dari method di langkah 3 pada praktikum tersebut! Mengapa dilakukan demikian?

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

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

Jawaban:

1. Tampilan README.md dengan dokumentasi berupa GIF



2. InheritedWidget itu adalah widget khusus di Flutter yang dipakai untuk membagikan data ke widget lain tanpa harus dikirim lewat parameter satu per satu. Di praktikum ini, digunakan InheritedNotifier karena selain bisa mewariskan data seperti InheritedWidget, dia juga bisa langsung memperbarui tampilan saat ada perubahan data lewat ValueNotifier, jadi kita nggak perlu panggil setState() di banyak tempat.

3. Method yang ditambahkan di langkah 3 berfungsi untuk menghitung jumlah task yang sudah selesai dan menampilkan pesan kemajuan tugas secara otomatis. Jadi setiap kali ada perubahan pada daftar tugas, aplikasi bisa langsung menunjukkan seberapa banyak tugas yang sudah dikerjakan tanpa kita harus hitung manual lagi.

4. Di langkah 9, saya menambahkan widget SafeArea yang menampilkan teks kemajuan seperti "2 out of 5 tasks" di bagian bawah aplikasi. Bagian ini menunjukkan progres tugas yang berubah otomatis setiap kali saya menambah atau mencentang task baru. Jadi tampilannya sekarang lebih interaktif dan bisa langsung memperlihatkan progres pekerjaan tanpa perlu di-refresh.

7. Praktikum 3: Membuat State di Multiple Screens

Satu kalimat populer atau viral yang beredar dalam komunitas Flutter adalah "**Lift State Up**". Mantra ini merujuk ke sebuah ide di mana objek State seharusnya berada lebih tinggi dari pada widget yang membutuhkannya di dalam sebuah widget tree. InheritedWidget yang telah kita buat sebelumnya bekerja dengan sempurna pada satu screen, tapi apa yang akan terjadi jika kita tambah screen kedua ?

Pada codelab ini, Anda akan menambah screen lain pada aplikasi Master Plan sehingga bisa membuat kelompok daftar plan lebih dari satu. Selesaikan langkah-langkah praktikum berikut ini menggunakan editor Visual Studio Code (VS Code) atau Android Studio atau code editor lain kesukaan Anda.

Langkah 1: Edit **PlanProvider**

Perhatikan kode berikut, edit class PlanProvider sehingga dapat menangani List Plan.

```
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!;
  }
}
```

lib > provider > plan_provider.dart > ...

```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!;
  }
}
```

Langkah 2: Edit **main.dart**

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,
      ),
```

```
      home: const PlanScreen(),
    ),
  );
}
```

lib > main.dart > ...

```dart
import 'package:flutter/material.dart';
import './views/plan_screen.dart';
import './provider/plan_provider.dart';
import './models/data_layer.dart';

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

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

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

Langkah 3: Edit plan_screen.dart

Tambahkan variabel `plan` dan atribut pada *constructor*-nya seperti berikut.

```
final Plan plan;
const PlanScreen({super.key, required this.plan});
```

lib > views > plan_screen.dart > _PlanScreenState > initState

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

class PlanScreen extends StatefulWidget {
  final Plan plan;
  const PlanScreen({super.key, required this.plan});

```

Langkah 4: Error

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.

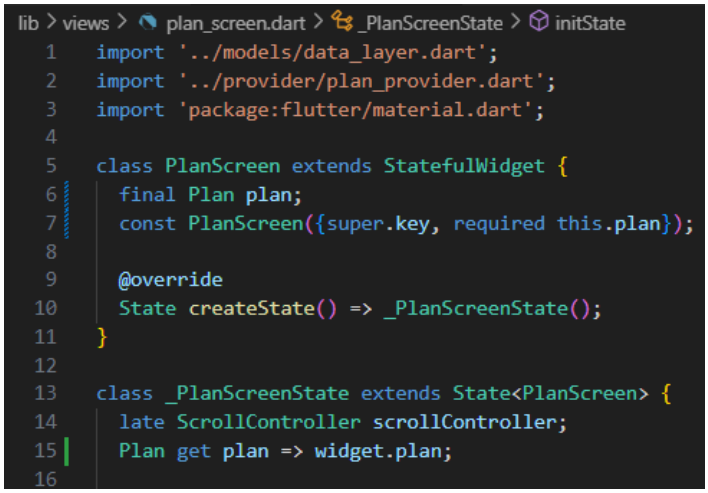
lib > views > plan_screen.dart > _PlanScreenState > initState

```dart
class _PlanScreenState extends State<PlanScreen> {
  Widget build(BuildContext context) {
      ), // AppBar
      body: ValueListenableBuilder<Plan>(
        valueListenable: PlanProvider.of(context),
        builder: (context, plan, child) {
          return Column(
            children: [
              Expanded(child: _buildList(plan)),
              SafeArea(child: Text(plan.completenessMessage)),
            ],
          ); // Column
        },
      ), // ValueListenableBuilder
      floatingActionButton: _buildAddTaskButton(context),
      backgroundColor: Colors.purple[50],
    ); // Scaffold
  }

  Widget _buildAddTaskButton(BuildContext context) {
    ValueNotifier<Plan> planNotifier = PlanProvider.of(context);
    return FloatingActionButton(
```

Langkah 5: Tambah **getter Plan**

Tambahkan getter pada _PlanScreenState seperti kode berikut.

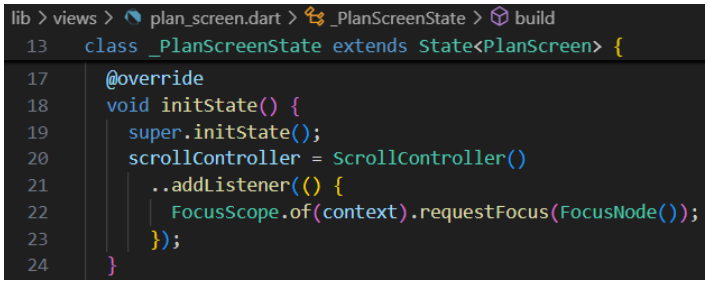
```
class _PlanScreenState extends State<PlanScreen> {
late ScrollController scrollController;
  Plan get plan => widget.plan;
```



Langkah 6: Method **initState()**

Pada bagian ini kode tetap seperti berikut.

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



Langkah 7: Widget **build**

Pastikan Anda telah merubah ke List dan mengubah nilai pada currentPlan seperti kode berikut ini.

```
@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) {
        Plan currentPlan = plans.firstWhere((p) => p.name == plan. name);
        return Column(
          children: [
            Expanded(child: _buildList(currentPlan)),
            SafeArea(child:Text(currentPlan.completenessM
            essage)),
          ],);},),
    floatingActionButton: _buildAddTaskButton(context,)
 ,);
}

Widget _buildAddTaskButton(BuildContext context) {
  ValueNotifier<List<Plan>> planNotifier = PlanProvider. of(context);
```

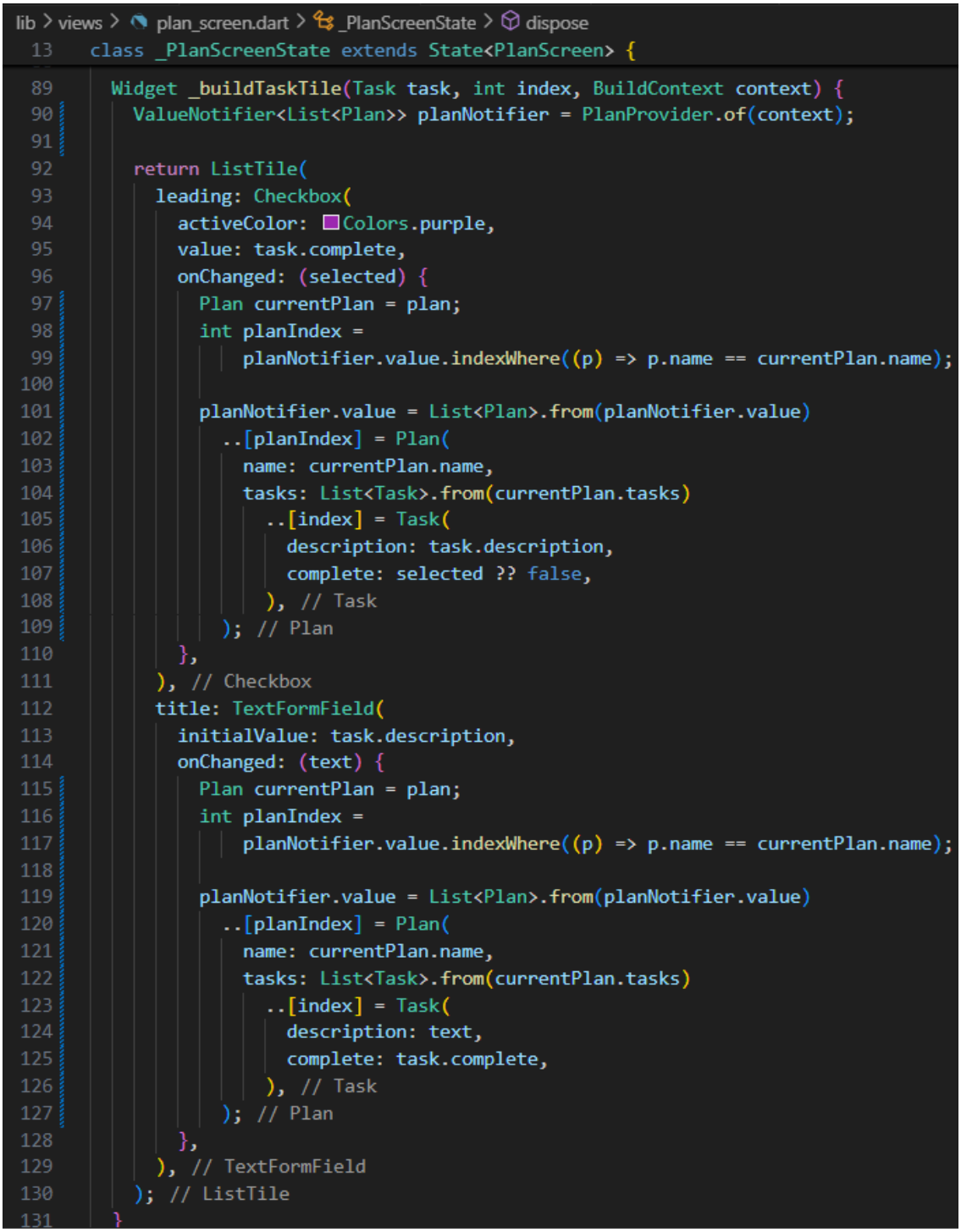
```
return FloatingActionButton( child:
  const Icon(Icons.add), onPressed:
  () {
    Plan currentPlan = plan;
    int planIndex =
        planNotifier.value.indexWhere((p) => p.name == currentPlan.name); List<Task>
    updatedTasks = List<Task>.from(currentPlan.tasks)
      ..add(const Task());
    planNotifier.value = List<Plan>.from(planNotifier.value)
      ..[planIndex] = Plan(  name:
        currentPlan.name, tasks:
        updatedTasks,
      );
    plan = Plan(
      name: currentPlan.name,
      tasks: updatedTasks,
    );},);
}
```

Langkah 8: Edit **_buildTaskTile**

Pastikan ubah ke `List` dan variabel `planNotifier` seperti kode berikut ini.

```
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,
                ),);
        }),
      title: TextFormField( initialValue:
        task.description, onChanged:
        (text) {
          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: text,
                  complete: task.complete,
                ),
            );
},),);}
```



Langkah 9: Buat screen baru

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.

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

lib > main.dart > ...

```dart
import 'package:flutter/material.dart';
import './views/plan_creator_screen.dart';
import './provider/plan_provider.dart';
import './models/data_layer.dart';

void main() => runApp(const MasterPlanApp());

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

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

lib > views > plan_creator_screen.dart > ...

```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 Nadya')),
      body: const Center(
        child: Text('Halaman untuk membuat dan melihat Master Plan'),
      ), // Center
    ); // Scaffold
  }
}
```

Langkah 10: Pindah ke class _PlanCreatorScreenState

Kita perlu tambahkan variabel TextEditingController sehingga bisa membuat TextField sederhana untuk menambah Plan baru. Jangan lupa tambahkan dispose ketika widget unmounted seperti kode berikut.

```dart
final textController = TextEditingController();

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

lib > views > plan_creator_screen.dart > ...

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

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

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

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Master Plans Nadya')),
      body: const Center(
        child: Text('Halaman untuk membuat dan melihat Master Plan'),
      ), // Center
    ); // Scaffold
  }
}
```

Langkah 11: Pindah ke method build

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

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

lib > views > plan_creator_screen.dart > _PlanCreatorScreenState > build

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

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

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

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

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

  Widget _buildListCreator() => Container();
  Widget _buildMasterPlans() => Container();

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

Langkah 12: Buat widget **_buildListCreator**

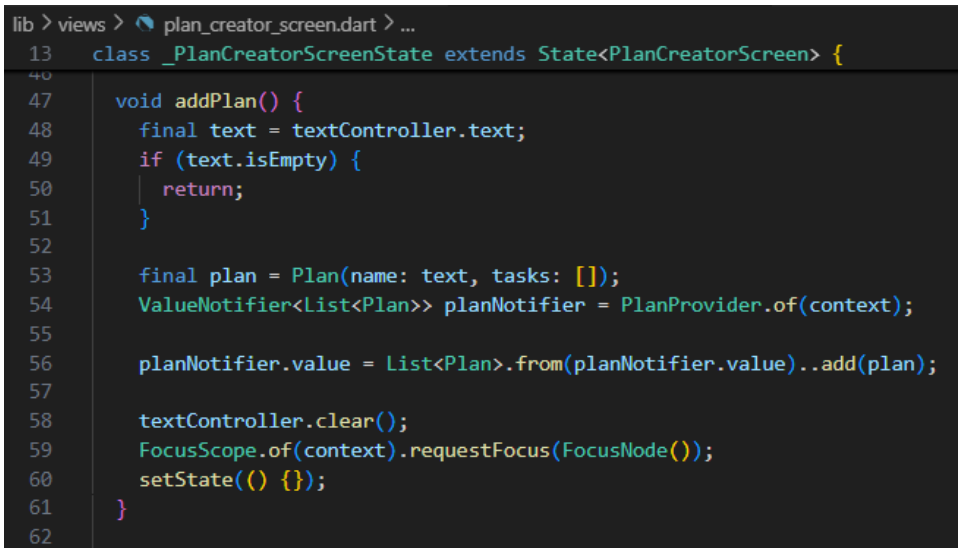
Buatlah widget berikut setelah widget build.

```
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)),
        onEditingComplete: addPlan),
    ));
}
```

Langkah 13: Buat **void addPlan()**

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

```
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(() {});
}
```



Langkah 14: Buat **widget _buildMasterPlans()**

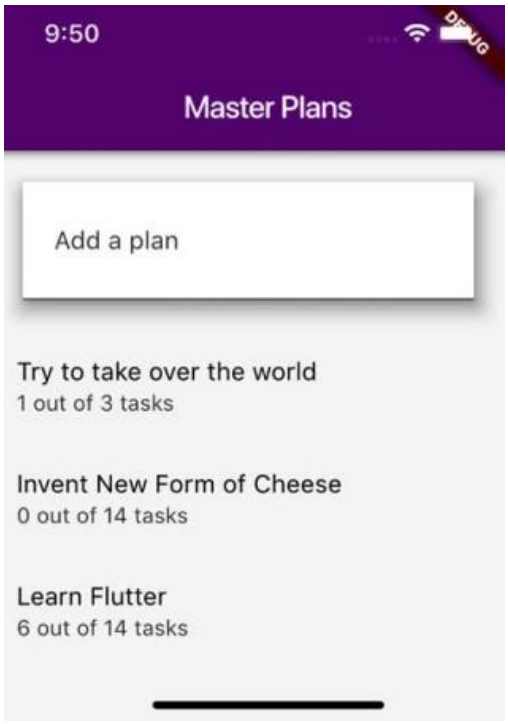
Tambahkan widget seperti kode berikut.

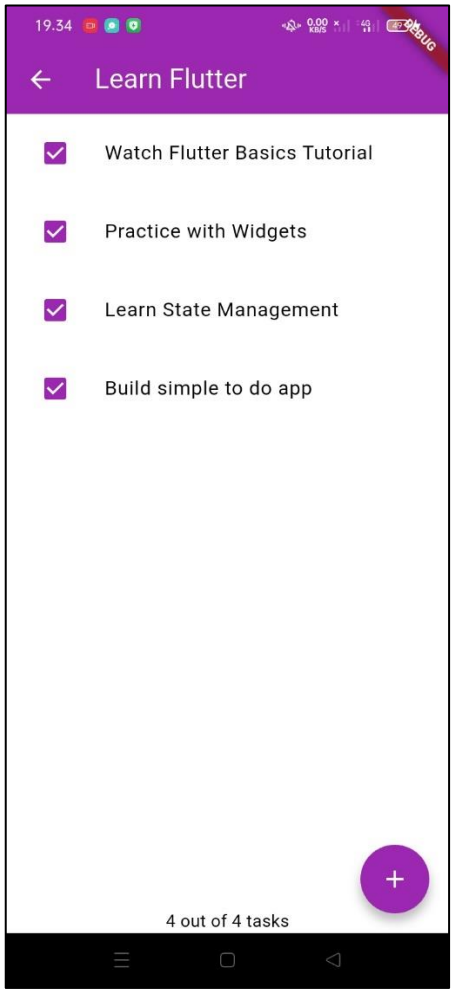
```
Widget _buildMasterPlans() {
 ValueNotifier<List<Plan>> planNotifier = PlanProvider.of(context); List<Plan>
  plans = planNotifier.value;
  if (plans.isEmpty) {
   return Column(
    mainAxisAlignment: MainAxisAlignment.center, children:
    <Widget>[
     const Icon(Icons.note, size: 100, color: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

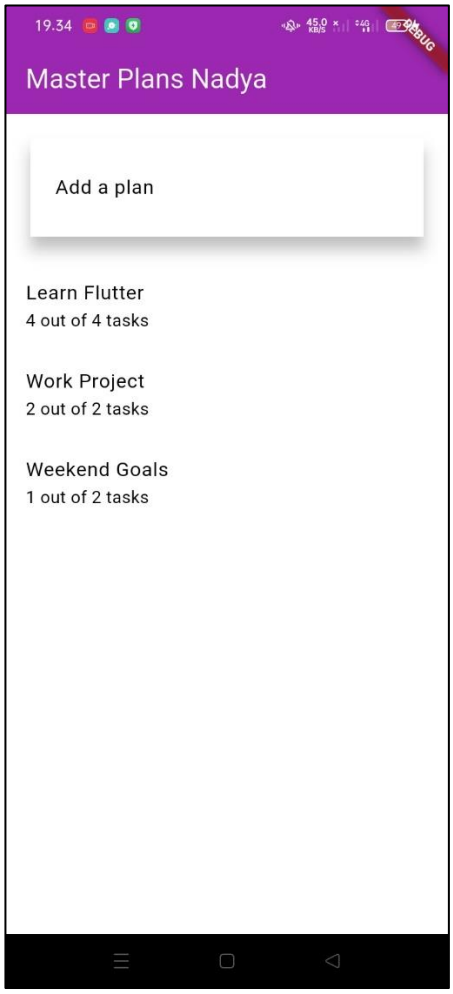
lib > views > plan_creator_screen.dart > ...

```dart
class _PlanCreatorScreenState extends State<PlanCreatorScreen> {
  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(
          title: Text(plan.name),
          subtitle: Text(plan.completenessMessage),
          onTap: () {
            Navigator.of(context).push(
              MaterialPageRoute(
                builder: (_) => PlanScreen(plan: plan),
              ), // MaterialPageRoute
            );
          },
        ); // ListTile
      },
    ); // ListView.builder
  }
```

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.

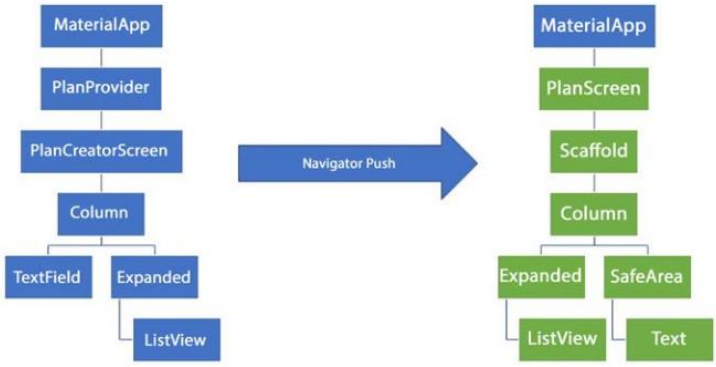






8. Tugas Praktikum 3: State di Multiple Screens

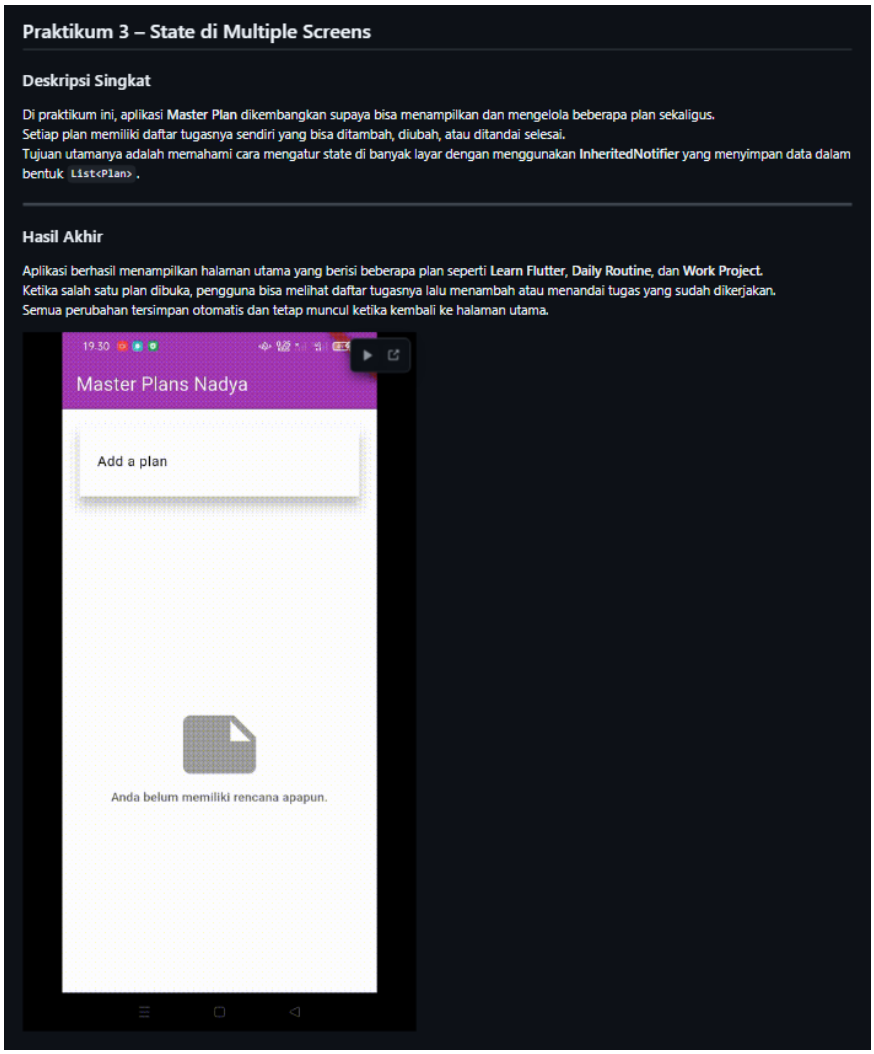
1. Selesaikan langkah-langkah praktikum tersebut, lalu dokumentasikan berupa GIF hasil akhir praktikum beserta penjelasannya di file README.md! Jika Anda menemukan ada yang error atau tidak berjalan dengan baik, silakan diperbaiki sesuai dengan tujuan aplikasi tersebut dibuat.
2. Berdasarkan Praktikum 3 yang telah Anda lakukan, jelaskan maksud dari gambar diagram berikut ini!



3. Lakukan capture hasil dari Langkah 14 berupa GIF, kemudian jelaskan apa yang telah Anda buat!
4. Kumpulkan laporan praktikum Anda berupa link commit atau repository GitHub ke dosen yang telah disepakati !

   Jawaban:

1. Tampilan README.md dengan dokumentasi berupa GIF.



2. Diagram di praktikum ini menjelaskan bagaimana alur data berpindah antar layar dan bagaimana state dikelola secara global. PlanProvider berperan sebagai pusat data utama yang menyimpan semua plan dalam aplikasi. Provider ini ditempatkan di level atas widget tree agar bisa diakses dari mana saja, baik dari halaman utama maupun halaman detail plan. Setiap kali pengguna menambah, mengedit, atau mencentang tugas di salah satu plan, perubahan tersebut langsung diperbarui di PlanProvider. Karena semua halaman menggunakan data dari provider yang sama, maka perubahan yang terjadi pada satu halaman otomatis akan terlihat di halaman lain tanpa harus membuat ulang data. Dengan konsep ini, aplikasi bisa mengelola banyak halaman (multiple screens) tanpa kehilangan state, sehingga data tetap sinkron dan konsisten meskipun pengguna berpindah antar layar.
3. Pada langkah 14, saya membuat tampilan utama yang berfungsi untuk membuat dan menampilkan daftar semua plan. Setiap plan menampilkan nama dan progres jumlah tugasnya. Saat plan diklik, aplikasi akan membuka halaman detail berisi daftar task yang bisa ditandai selesai atau ditambah sesuai kebutuhan. GIF hasil akhir menunjukkan bagaimana saya menambah plan baru, membuka halaman detail, dan memperbarui status tugas dengan lancar menggunakan konsep state management di beberapa layar.