#11 | Dasar State Management

| **Summary** | Mampu mengelola state di Flutter |
| --- | --- |
| **URL** | 11-basic-state |
| **Category** | Mobile Flutter |
| **Environment** | web |
| **Status** | Published |
| **Feedback Link** | https://github.com/jti-polinema/flutter-codelab/issues |
| **Author** | Habibie Ed Dien |
| **Author LDAP** | hbb-polinema |
| **Analytics Account** | G-H7507PFEJB |

[Dasar Manajemen State di Flutter](#_ok7k5uux6)

[Tujuan Praktikum](#_21yzqg98x7h6)

[Sumber Daya yang Dibutuhkan](#_dusjnkkj26ob)

[Pengetahuan yang Anda harus Miliki](#_319h1e1sjzdz)

[Manajemen State](#_t0tqoph2ohlh)

[Konsep Model-View](#_86gx1e3vopk3)

[Praktikum 1: Dasar State dengan Model-View](#_vmpz7ap0erhd)

[Langkah 1: Buat Project Baru](#_spr93e7x3yo2)

[Langkah 2: Membuat model task.dart](#_g8csgggwssw)

[Langkah 3: Buat file plan.dart](#_2q85gi9dhaiq)

[Langkah 4: Buat file data\_layer.dart](#_p0t5r1yzb5qp)

[Langkah 5: Pindah ke file main.dart](#_xmhut8m5gdv1)

[Langkah 6: buat plan\_screen.dart](#_4b6h89e2cyxj)

[Langkah 7: buat method \_buildAddTaskButton()](#_5krokjvrwldm)

[Langkah 8: buat widget \_buildList()](#_e3g15rw1g6tl)

[Langkah 9: buat widget \_buildTaskTile](#_j2bkm5ipifyc)

[Langkah 10: Tambah Scroll Controller](#_okn37ljf0voe)

[Langkah 11: Tambah Scroll Listener](#_flqbsie0sby)

[Langkah 12: Tambah controller dan keyboard behavior](#_ccxeudkxbh36)

[Langkah 13: Terakhir, tambah method dispose()](#_dg2knoo8i4rr)

[Langkah 14: Hasil](#_l7ym0d8h6vlx)

[Tugas Praktikum 1: Dasar State dengan Model-View](#_dz35nqtk0wjb)

[Praktikum 2: Mengelola Data Layer dengan InheritedWidget dan InheritedNotifier](#_1e543m1dm1c4)

[Langkah 1: Buat file plan\_provider.dart](#_u3l46a7g73v0)

[Langkah 2: Edit main.dart](#_gfa7d3gyzo4o)

[Langkah 3: Tambah method pada model plan.dart](#_wc7bdoy0rvka)

[Langkah 4: Pindah ke PlanScreen](#_dxl4engxhz7o)

[Langkah 5: Edit method \_buildAddTaskButton](#_dswkw4kv0st7)

[Langkah 6: Edit method \_buildTaskTile](#_y44t6huj0yys)

[Langkah 7: Edit \_buildList](#_ufiogbhfaq4m)

[Langkah 8: Tetap di class PlanScreen](#_dxub23gyz21t)

[Langkah 9: Tambah widget SafeArea](#_b28b1kisggzv)

[Tugas Praktikum 2: InheritedWidget](#_9k4ljuvnqeo3)

[Praktikum 3: Membuat State di Multiple Screens](#_riuqbijaf0zt)

[Langkah 1: Edit PlanProvider](#_uv7foq4euher)

[Langkah 2: Edit main.dart](#_di3ldp29lo3b)

[Langkah 3: Edit plan\_screen.dart](#_ma6gko34okso)

[Langkah 4: Error](#_oxnid84td09d)

[Langkah 5: Tambah getter Plan](#_auxtuu9crnqv)

[Langkah 6: Method initState()](#_8b58y7cyb598)

[Langkah 7: Widget build](#_5e2z95f7zwdd)

[Langkah 8: Edit \_buildTaskTile](#_beimcgaw4uuc)

[Langkah 9: Buat screen baru](#_ux10u3wh6amn)

[Langkah 10: Pindah ke class \_PlanCreatorScreenState](#_wz37fkekarcl)

[Langkah 11: Pindah ke method build](#_dognjm4cb57g)

[Langkah 12: Buat widget \_buildListCreator](#_hkjs1mhd2lno)

[Langkah 13: Buat void addPlan()](#_9pt62kmscvzi)

[Langkah 14: Buat widget \_buildMasterPlans()](#_v171501uo3db)

[Tugas Praktikum 3: State di Multiple Screens](#_q5xm2f93gy3w)

[Tugas Kelompok PBL](#_3gkrz7x5i2n)

[Selamat!](#_s22qmcj4zu61)

[Apa selanjutnya?](#_9mqifffit2ew)

[Referensi](#_an95hoqxrlhg)

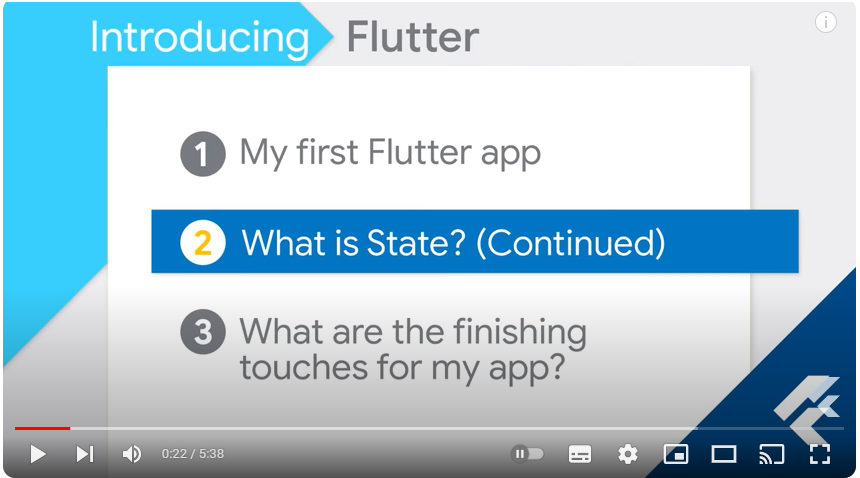
# Dasar Manajemen State di Flutter

**Terakhir diperbarui:** 07 November 2023

**Penulis:** Habibie Ed Dien

Pada codelab ini, Anda akan mempelajari tentang dasar manajemen state di Flutter beserta contoh penggunaannya. Cara kerja, manfaat, dan cara mengelola state di multi screen.

Video berikut menjelaskan tentang apa itu state dan bagaimana manfaatnya. Silakan simak dan pahami!



## Tujuan Praktikum

Setelah menyelesaikan codelab ini Anda akan mampu untuk:

* menjelaskan konsep arsitektur model-view
* mengelola data layer dengan InheritedWidget dan InheritedNotifier
* membuat app state di multiple screens

| **Catatan:** Materi ini diadaptasi dari Buku “Flutter Cookbook, 100+ step-by-step recipes for building cross-platform, professional grade apps with Flutter 3.10.x and Dart 3.x - Second Edition” oleh Simone Alessandria (2023). |
| --- |

## Sumber Daya yang Dibutuhkan

Berikut merupakan sumber daya yang diperlukan untuk menyelesaikan praktikum ini:

* PC atau Laptop dengan spesifikasi minimum RAM 4GB Processor Core i3
* Koneksi internet minimal kecepatan 1 Mbps
* Chrome Browser
* Perangkat mobile (Android/iOS)

## Pengetahuan yang Anda harus Miliki

* Algoritma dan Pemrograman Dasar
* Pemrograman Berorientasi Objek
* Bahasa Pemrograman Dart

# Manajemen State

Duration: 10:00

Seiring berkembangnya aplikasi, pengelolaan aliran data melalui aplikasi tersebut menjadi lebih kompleks dan penting. Komunitas Flutter telah merancang beberapa solusi untuk menangani pengelolaan state. **Semua solusi ini memiliki satu aspek yang sama: pemisahan UI dan logika bisnis**.

Sebelum mendalami bagaimana manajemen state itu bekerja, pada codelab ini Anda akan praktik menggunakan state ke tingkat yang lebih tinggi.

Dalam codelab ini, Anda akan membuat aplikasi pencatatan tugas. Dalam aplikasi ini, pengguna

akan dapat membuat daftar tugas yang berisi banyak tugas. Pengguna akan dapat menambah dan melengkapi tugasnya.

Beberapa topik pembahasan di codelab ini terdiri dari:

* Pemisahan Model-View
* Mengelola data layer dengan InheritedWidget dan InheritedNotifier
* Membuat *state* aplikasi dapat diakses pada *multi screens*

## Konsep Model-View

Model dan View merupakan konsep yang sangat penting dalam arsitektur aplikasi flutter. **Model** adalah kelas yang menangani dengan data untuk suatu aplikasi, sedangkan View adalah kelas yang menampilkan data tersebut di layar.

Di Flutter, View dibuat dari widget. Selanjutnya, model akan menjadi kelas dasar Dart itu yang tidak mewarisi apa pun dari Framework Flutter. Masing-masing kelas ini bertanggung jawab

untuk satu dan hanya satu pekerjaan. **Model** berkaitan dengan penanganan data untuk aplikasi Anda. **View** fokus untuk menampilkan UI di layar HP. Ketika Anda konsisten dengan konsep arsitektur model dan view, kode Anda akan menjadi lebih sederhana dan mudah dibaca.

Dalam codelab ini, kita akan membangun aplikasi **master\_plan** dengan membuat kelas model dan view.

# Praktikum 1: Dasar State dengan Model-View

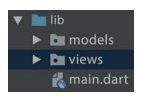
Duration: 60:00

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

| **Perhatian:** Diasumsikan Anda telah berhasil melakukan setup environment Flutter SDK, VS Code, Flutter Plugin, dan Android SDK pada pertemuan pertama. |
| --- |

## Langkah 1: Buat Project Baru

Buatlah sebuah project flutter baru dengan nama **master\_plan** di folder **src week-11** repository GitHub Anda. 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; |
| --- |

## 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());  });  } |
| --- |

## 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, |
| --- |

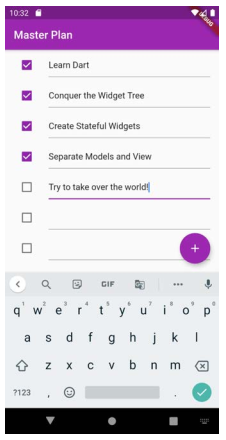
## Langkah 13: Terakhir, tambah method dispose()

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

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

## Langkah 14: Hasil

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



| **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(). Kondisi app state akan reset (kembali seperti awal). |
| --- |

## 

# Tugas Praktikum 1: Dasar State dengan Model-View

Duration: 90:00

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 spreadsheet yang telah disediakan!

# Praktikum 2: Mengelola Data Layer dengan InheritedWidget dan InheritedNotifier

Duration: 60:00

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.

| **Perhatian:** Diasumsikan Anda telah berhasil menyelesaikan Praktikum 1 Model-View. |
| --- |

## 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!;  }  } |
| --- |

## 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(),  ),  ); |
| --- |

## 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'; |
| --- |

## 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.

## 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()),  );  },  );  } |
| --- |

## 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,  ),  );  },  ),  );  } |
| --- |

## 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),  );  } |
| --- |

## 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.

# Tugas Praktikum 2: InheritedWidget

Duration: 90:00

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

# Praktikum 3: Membuat State di Multiple Screens

Duration: 60:00

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.

| **Perhatian:** Diasumsikan Anda telah berhasil menyelesaikan praktikum 2. |
| --- |

## 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!;  }  } |
| --- |

## 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(),  ),  );  } |
| --- |

## 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}); |
| --- |

## 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.

## 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<Plan> 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.  completenessMessage)),  ],);},),  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<Plan> 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(), |
| --- |

## 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.

| final textController = TextEditingController();  @override  void dispose() {  textController.dispose();  super.dispose();  } |
| --- |

## Langkah 11: Pindah ke method build

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

| @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())  ]),  );  } |
| --- |

## 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.

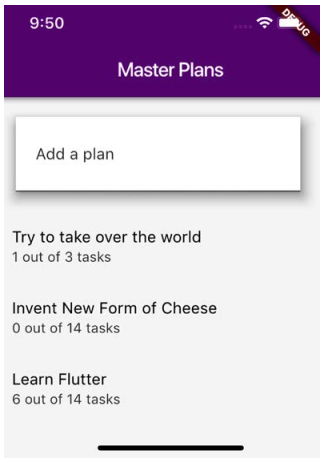
| void addPlan() {  final text = textController.text;  if (text.isEmpty) {  return;  }  final plan = Plan(name: text, tasks: []);  ValueNotifier<List<Plan>> planNotifier =  PlanProvider.of(context);  planNotifier.value = List<Plan>.from(planNotifier.value)..  add(plan);  textController.clear();  FocusScope.of(context).requestFocus(FocusNode());  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,)));  });  });  } |
| --- |

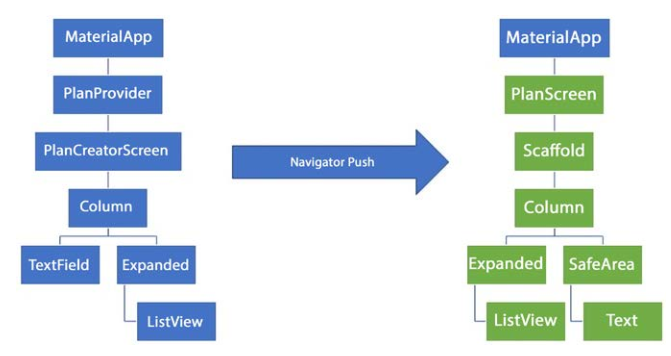
Terakhir, **run** atau tekan **F5** untuk melihat hasilnya jika memang belum running. Bisa juga lakukan **hot restart** jika aplikasi sudah running. Maka hasilnya akan seperti gambar berikut ini.



# Tugas Praktikum 3: State di Multiple Screens

Duration: 90:00

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!



1. Lakukan capture hasil dari Langkah 14 berupa GIF, kemudian jelaskan apa yang telah Anda buat!
2. Kumpulkan laporan praktikum Anda berupa link commit atau repository GitHub ke spreadsheet yang telah disediakan!

# Tugas Kelompok PBL

Duration: 90:00

1. Tambahkan *class model* yang dibutuhkan sesuai aplikasi OCR kelompok Anda! Bagilah tugas dengan anggota kelompok Anda untuk merancang proses state yang akan dilakukan pada aplikasi kelompok Anda. Lalu terapkan state management pada project kelompok yang telah Anda pelajari melalui praktikum ini.
2. Anda boleh menggunakan plugin/library pihak ketiga seperti riverpod, BLoC, provider atau lainnya untuk mengelola state pada project kelompok PBL.
3. Jika telah berhasil sesuai kebutuhan, maka push hasil kontribusi masing-masing anggota kelompok ke repository project PBL. Beri keterangan commit dengan jelas fitur apa saja yang ditambahkan, tiap screen/fitur untuk setiap commit itu lebih baik.

# Selamat!

Duration: 0:00

Selamat Anda telah menyelesaikan Codelab ini. Anda telah mempelajari terkait state management dan contoh penggunaannya.

Pada codelab berikutnya, Anda akan mempelajari tentang Pemrograman Asynchronous di Flutter.

Jangan sungkan jika Anda menemukan kesalahan pada codelab ini untuk merevisi atau sekedar melaporkan issue melalui tautan di pojok kiri bawah ([Report a mistake](https://github.com/jti-polinema/flutter-codelab/issues)).

## Apa selanjutnya?

Silakan cek beberapa sumber belajar lainnya…

* [Konsep State Management Flutter : Definisi dan Kegunaannya](https://www.youtube.com/watch?v=YZkRIHXhdGo)
* [InheritedWidget class documentation API official](https://api.flutter.dev/flutter/widgets/InheritedWidget-class.html)
* <https://riverpod.dev/>
* <https://pub.dev/packages/provider>

## Referensi

* [Buku “Flutter Cookbook, 100+ step-by-step recipes for building cross-platform, professional grade apps with Flutter 3.10.x and Dart 3.x - Second Edition” oleh Simone Alessandria (2023)](https://drive.google.com/file/d/1W9uSLC9wlf9fFudmCWFHTx9dan5SXII0/view?usp=sharing)
* [Flutter for Beginners Second Edition](https://drive.google.com/file/d/1hJbctx-_O3B6kUeG70bJD5xfbnsWwBXY/view?usp=sharing)
* <https://www.youtube.com/flutterdev>
* <https://github.com/PacktPublishing/Flutter-for-Beginners-Second-Edition>