PEMROGRAMAN MOBILE
PENGANTAR BAHASA PEMROGRAMAN DART - BAGIAN 10

OLEH:

Nama : Agung Rizky S

NIM : 2241720187

Kelas : TI – 3C

PROGRAM STUDI D-IV TEKNIK INFORMATIKA

JURUSAN TEKNOLOGI INFORMASI

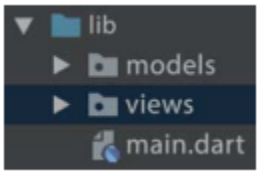
POLITEKNIK NEGERI MALANG

2024

Praktikum 1: Dasar State dengan Model-View

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,
```

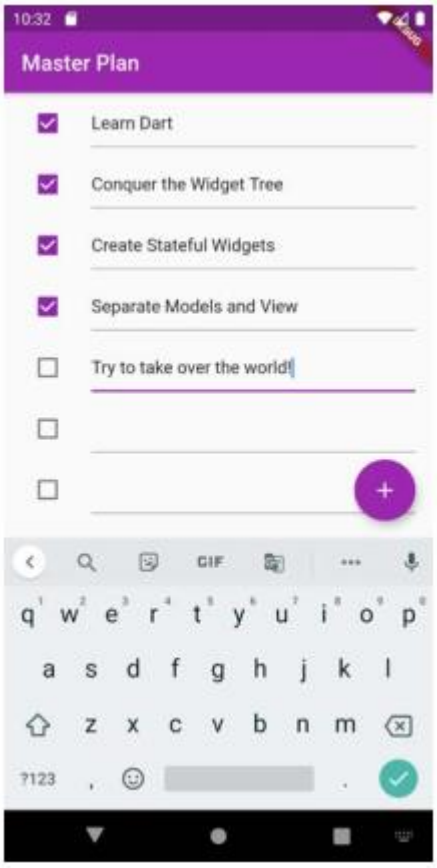
Langkah 13: Terakhir, tambah method dispose()

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.



Hasil :

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?
    - Tujuan dari langkah ini adalah untuk memudahkan proses pengimporan model-model yang dibutuhkan (Task dan Plan) ke dalam file lain dalam proyek.
    - Mengapa dilakukan demikian? Karena dalam pengelolaan proyek aplikasi skala besar. Dengan adanya file data_layer.dart, maka hanya perlu mengimpor satu file ketika ingin menggunakan model Task atau Plan, sehingga kode menjadi lebih rapi dan mudah dikelola.
3. Mengapa perlu variabel plan di langkah 6 pada praktikum tersebut? Mengapa dibuat konstanta ?
    - Mengapa diperlukan variabel plan? Variabel plan digunakan untuk menyimpan data terkait rencana (plan) yang sedang ditampilkan dan dimanipulasi dalam aplikasi.
    - Mengapa dibuat sebagai konstanta? Karena pada saat pertama kali aplikasi dijalankan, akan menginginkan objek Plan dengan nilai default yang tidak berubah. Namun, setelah data diubah (misalnya, ketika pengguna menambah atau mengubah tugas), objek Plan baru akan diciptakan, tetapi awalnya Plan ini bersifat tetap karena hanya digunakan untuk inisialisasi.
4. Lakukan capture hasil dari Langkah 9 berupa GIF, kemudian jelaskan apa yang telah Anda buat!
    - Checkbox: Untuk menandai apakah tugas sudah selesai atau belum (task.complete).
    - TextFormField: Untuk mengedit deskripsi tugas (task.description).
    - Ketika pengguna menandai checkbox atau mengubah teks pada TextFormField, metode setState dipanggil untuk memperbarui status tugas dalam daftar dan memperbarui tampilan UI dengan data terbaru.
5. Apa kegunaan method pada Langkah 11 dan 13 dalam *lifecyle state* ?
    - Kegunaan dalam lifecycle: initState() adalah method yang dipanggil sekali ketika State widget pertama kali diinisialisasi. Method ini digunakan untuk menyiapkan data awal atau menginisialisasi objek-objek yang dibutuhkan.
    - ScrollController digunakan untuk mengontrol perilaku scroll, dan listener ditambahkan untuk menghapus fokus dari TextFormField saat daftar di-scroll, sehingga mencegah masalah saat keyboard terbuka di iOS.
    - Kegunaan dalam lifecycle: dispose() dipanggil ketika State dihapus dari widget tree. Method ini digunakan untuk membersihkan resource yang tidak diperlukan lagi, seperti ScrollController.
    - Dengan memanggil dispose(), Anda memastikan bahwa memori tidak bocor dengan menghapus controller yang sudah tidak dipakai, menjaga efisiensi aplikasi saat widget tidak lagi digunakan.
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

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.

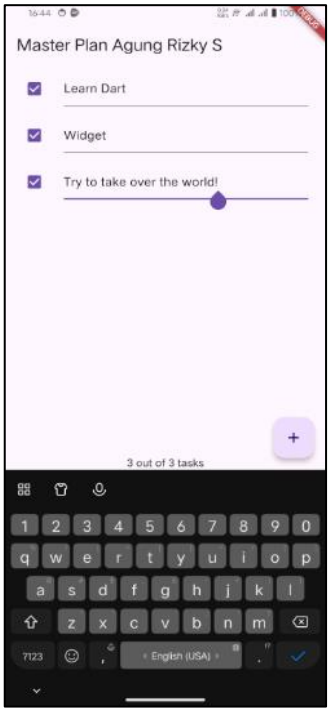
Langkah 9: Tambah widget SafeArea

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.

Hasil :



Tugas Praktikum 2: InheritedWidget

1. Selesaikan langkah-langkah praktikum tersebut, lalu dokumentasikan berupa GIF hasil akhir praktikum beserta penjelasannya di file `README.md`! Jika Anda menemukan ada yang error atau tidak berjalan dengan baik, silakan diperbaiki sesuai dengan tujuan aplikasi tersebut dibuat.
2. Jelaskan mana yang dimaksud `InheritedWidget` pada langkah 1 tersebut! Mengapa yang digunakan `InheritedNotifier`?
    - PlanProvider adalah kelas yang dibuat dengan mewarisi dari InheritedNotifier<ValueNotifier<Plan>>. InheritedWidget digunakan untuk menyediakan data ke subtree widget tanpa harus melakukan pemanggilan manual data ke setiap widget turunannya. InheritedWidget memudahkan berbagi data antar widget dalam pohon widget Flutter.
    - InheritedNotifier adalah turunan dari InheritedWidget yang dilengkapi dengan kemampuan untuk memerhatikan perubahan nilai dari suatu Notifier (dalam hal ini, ValueNotifier).
    - Mengapa menggunakan InheritedNotifier?
      Karena inheritedNotifier hanya akan memperbarui widget turunannya saat ValueNotifier berubah, sehingga mengurangi jumlah rebuild yang tidak perlu pada widget yang tidak terkait dengan data yang diperbarui.
3. Jelaskan maksud dari method di langkah 3 pada praktikum tersebut! Mengapa dilakukan demikian?
    - completedCount: Menghitung jumlah tugas (task) yang sudah selesai dengan memfilter task mana yang statusnya complete.
      tasks.where((task) => task.complete).length;
      Ini dilakukan untuk memudahkan menghitung task yang sudah selesai tanpa harus melakukan perhitungan manual di luar model.

- completenessMessage: Mengembalikan pesan teks yang menunjukkan berapa banyak tugas yang sudah selesai dibandingkan dengan total jumlah tugas.
  '$completedCount out of ${tasks.length} tasks';
  Pesan ini menunjukkan progres pengguna dalam menyelesaikan task, yang akan digunakan pada UI untuk memberikan umpan balik kepada pengguna tentang progress mereka.

- Mengapa ini dilakukan?
  Karena untuk memisahkan logika dari tampilan (UI) dan menyimpannya dalam model.

4. Lakukan capture hasil dari Langkah 9 berupa GIF, kemudian jelaskan apa yang telah Anda buat!
   - SafeArea digunakan untuk memastikan bahwa tampilan UI tidak tumpang tindih dengan area yang penting pada layar (seperti notch, status bar, atau area di sekitar navigasi pada perangkat dengan layar penuh).
   - Di dalam widget Column, menambahkan dua bagian:
     - Expanded: Berisi ListView yang menampilkan daftar tugas (tasks).
     - SafeArea: Berisi teks progress (completenessMessage) yang menunjukkan berapa banyak task yang sudah selesai dibandingkan dengan total task.
5. Kumpulkan laporan praktikum Anda berupa link commit atau repository GitHub ke spreadsheet yang telah disediakan!

Praktikum 3: Membuat State di Multiple Screens

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` 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` 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.

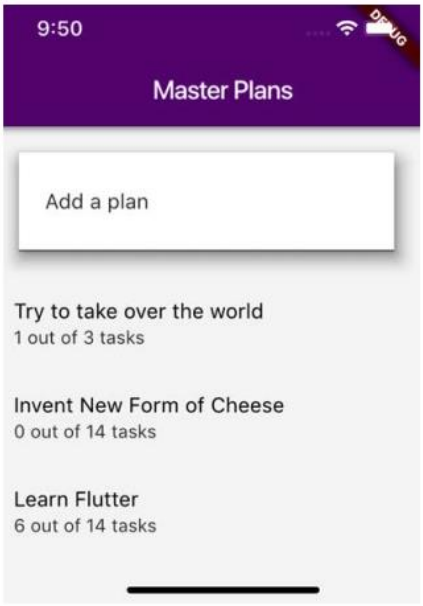
```
void addPlan() {
  final text = textController.text;
    if (text.isEmpty) {
      return;
    }
    final plan = Plan(name: text, tasks: []);
    ValueNotifier<List<Plan>> planNotifier =
PlanProvider.of(context);
    planNotifier.value = List<Plan>.from(planNotifier.value)..
add(plan);
    textController.clear();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

Langkah 14: Buat widget _buildMasterPlans()

Tambahkan widget seperti kode berikut.

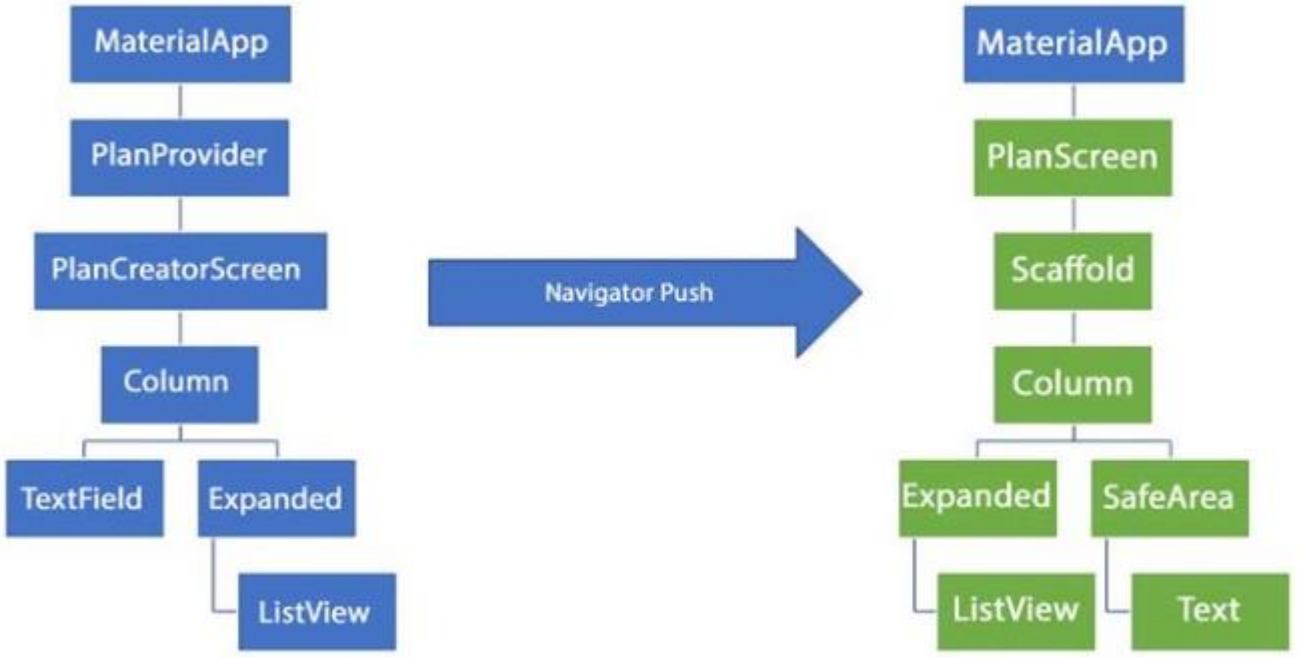
```
Widget _buildMasterPlans() {
  ValueNotifier<List<Plan>> planNotifier = PlanProvider.of(context);
    List<Plan> plans = planNotifier.value;

    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

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!



Jawaban :

- **Diagram Kiri (PlanCreatorScreen)**:
  - Menampilkan hierarki widget pada screen PlanCreatorScreen, yang bertujuan untuk membuat atau menambah rencana baru.
  - MaterialApp adalah root aplikasi.
  - PlanProvider berfungsi untuk menyediakan state list Plan ke semua widget yang membutuhkan.
  - PlanCreatorScreen menampilkan Column yang terdiri dari TextField (untuk input nama rencana) dan ListView yang menampilkan daftar rencana yang telah dibuat.

- **Diagram Kanan (PlanScreen)**:
  - Setelah pengguna menambahkan rencana baru atau memilih salah satu rencana dari daftar, aplikasi melakukan navigasi menggunakan Navigator.push ke PlanScreen.
  - PlanScreen menampilkan detail dari rencana yang dipilih.
  - Column di dalam Scaffold menampung Expanded dengan ListView untuk menampilkan daftar tugas dari rencana tersebut dan SafeArea dengan Text untuk menampilkan pesan kelengkapan rencana.

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 spreadsheet yang telah disediakan!