LAPORAN PRAKTIKUM

JOBSHEET IX: DASAR STATE MANAGEMENT

MATA KULIAH PEMROGRAMAN MOBILE

Dosen Pengampu : Ade Ismail, S.Kom., M.TI

Disusun oleh :

Nama : Vita Eka Saraswati

NIM : 2341760082

No. Absen : 29

JURUSAN TEKNOLOGI INFORMASI

PROGRAM STUDI D-IV SISTEM INFORMASI BISNIS

POLITEKNIK NEGERI MALANG

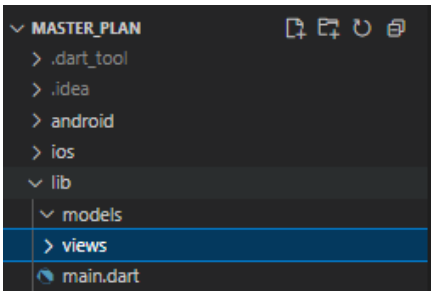
2025

JOBSHEET MINGGU IX

DASAR STATE MANAGEMENT

PRAKTIKUM 1 – DASAR STATE DENGAN MODEL-VIEW

1. Buatlah sebuah project flutter baru dengan nama **master_plan** di folder **src week-10** repository GitHub Anda atau sesuai style laporan praktikum yang telah disepakati. Lalu buatlah susunan folder dalam project seperti gambar berikut ini.



2. **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:

```
lib > models > task.dart > Task
1  class Task {
2    final String description;
3    final bool complete;
4
5    const Task({
6      this.complete = false,
7      this.description = '',
8    });
9  }
```

3. Buat file `plan.dart` di dalam folder **models** dan isi kode seperti berikut.

```
lib > models > plan.dart > Plan
1  import './task.dart';
2
3  class Plan {
4    final String name;
5    final List<Task> tasks;
6
7    const Plan({this.name = '', this.tasks = const []});
8  }
```

4. Buat file bernama `data_layer.dart` di folder **models**. Kodenya hanya berisi `export` seperti berikut.

```
lib > models > data_layer.dart
1   export 'plan.dart';
2   export 'task.dart';
```

5. Ubah isi kode main.dart sebagai berikut.

```
lib > main.dart > MasterPlanApp
1   import 'package:flutter/material.dart';
2   import './views/plan_screen.dart';
3
    Run | Debug | Profile
4   void main() => runApp(MasterPlanApp());
5
6   class MasterPlanApp extends StatelessWidget {
7     const MasterPlanApp({super.key});
8
9     @override
10    Widget build(BuildContext context) {
11      return MaterialApp(
12        theme: ThemeData(primarySwatch: Colors.purple),
13        home: PlanScreen(),
14      ); // MaterialApp
15    }
16  }
```

6. **Buat file 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.

```
lib > views > plan_screen.dart > _PlanScreenState
1   import '../models/data_layer.dart';
2   import 'package:flutter/material.dart';
3
4   class PlanScreen extends StatefulWidget {
5     const PlanScreen({super.key});
6
7     @override
8     State createState() => _PlanScreenState();
9
10  }
11
12  class _PlanScreenState extends State<PlanScreen> {
13    Plan plan = const Plan();
14
15    @override
16    Widget build(BuildContext context) {
17      return Scaffold(
18        // ganti 'Namaku' dengan Nama panggilan Anda
19        appBar: AppBar(title: const Text('Master Plan Namaku')),
20        body: _buildList(),
21        floatingActionButton: _buildAddTaskButton(),
22      ); // Scaffold
23    }
24  }
```

7. Buat method _buildAddTaskButton(). Tambah kode berikut di bawah method build di dalam class _PlanScreenState.

```
lib > views > plan_screen.dart > _PlanScreenState > _buildAddTaskButton
1   import '../models/data_layer.dart';
2   import 'package:flutter/material.dart';
3
4   class PlanScreen extends StatefulWidget {
5     const PlanScreen({super.key});
6
7     @override
8     State createState() => _PlanScreenState();
9   }
10
```

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Master Plan Vita')),
      body: _buildList(),
      floatingActionButton: _buildAddTaskButton(),
    ); // Scaffold
  }

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

8. Buat widget ListView seperti kode berikut ini.

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

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

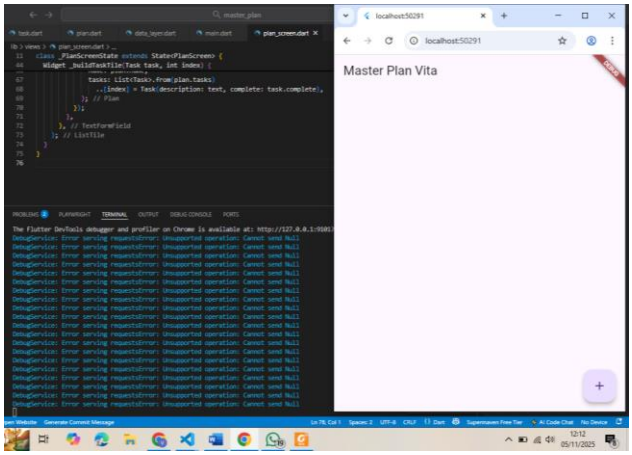
9. **Buat widget _buildTaskTile**

   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.

```dart

  Widget _buildTaskTile(Task task, int index) {
    return ListTile(
      leading: Checkbox(
          value: task.complete,
          onChanged: (selected) {
            setState(() {
              plan = Plan(
                name: plan.name,
                tasks: List<Task>.from(plan.tasks)
                  ..[index] = Task(
                    description: task.description,
                    complete: selected ?? false,
                  ), // Task
              ); // Plan
            });
          }), // Checkbox
      title: TextFormField(
        initialValue: task.description,
        onChanged: (text) {
          setState(() {
            plan = Plan(
              name: plan.name,
              tasks: List<Task>.from(plan.tasks)
                ..[index] = Task(
                  description: text,
                  complete: task.complete,
                ), // Task
            ); // Plan
          });
```

```
        },
      ), // TextFormField
    ); // ListTile
  }
```

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



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.

```
lib > views > plan_screen.dart > _PlanScreenState
class PlanScreen extends StatefulWidget {
                                    _PlanScreenState   Aa ab .* 2 of 2

  @override
  State createState() => _PlanScreenState();
}

class _PlanScreenState extends State<PlanScreen> {
  Plan plan = const Plan();
  late ScrollController scrollController;
                                    Chat (CTRL + I) / Share (CTRL + L)
```

11. Tambah Scroll Listener

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

```
  late ScrollController scrollController;

  @override
  void initState() {
    super.initState();
    scrollController = ScrollController()
      ..addListener(() {
        FocusScope.of(context).requestFocus(FocusNode());
      });
  }        Chat (CTRL + I) / Share (CTRL + L)
```

12. **Tambah controller dan keyboard behavior**

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

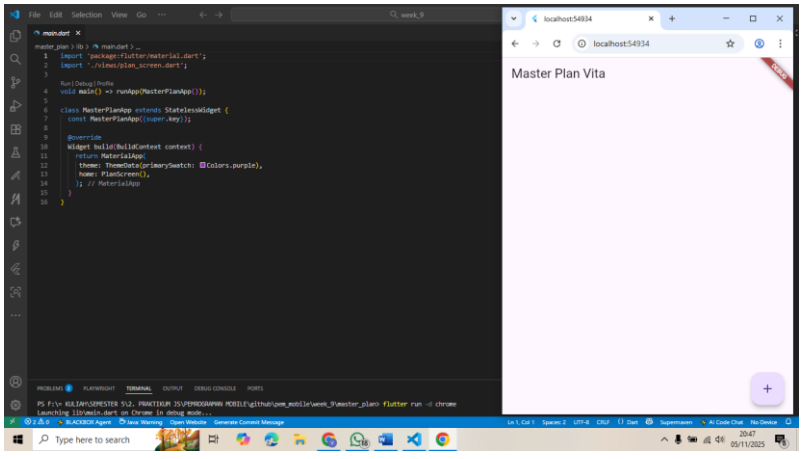
```
}

Widget _buildList() {
  return ListView.builder(
    controller: scrollController, // this allows the keyboard to be dismissed
    keyboardDismissBehavior: Theme.of(context).platform == TargetPlatform.iOS
        ? ScrollViewKeyboardDismissBehavior.onDrag
        : ScrollViewKeyboardDismissBehavior.manual,
    itemCount: plan.tasks.length,
    itemBuilder: (context, index) => _buildTaskTile(plan.tasks[index], index),
  );
}

```

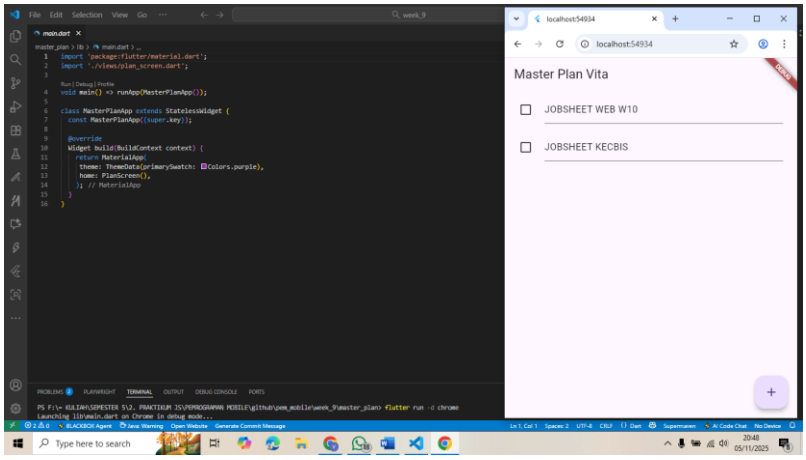
13. **Tambah method dispose()**

    Tambahkan method dispose() berguna ketika widget sudah tidak digunakan lagi.

```
  }

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

```

HASIL OUTPUT PRAKTIKUM 1





TUGAS PRAKTIKUM 1

1. Selesaikan langkah-langkah praktikum tersebut, lalu dokumentasikan berupa GIF hasil akhir praktikum beserta penjelasannya di file README.md! Jika Anda menemukan ada yang error atau tidak berjalan dengan baik, silakan diperbaiki.
2. Jelaskan maksud dari langkah 4 pada praktikum tersebut! Mengapa dilakukan demikian?

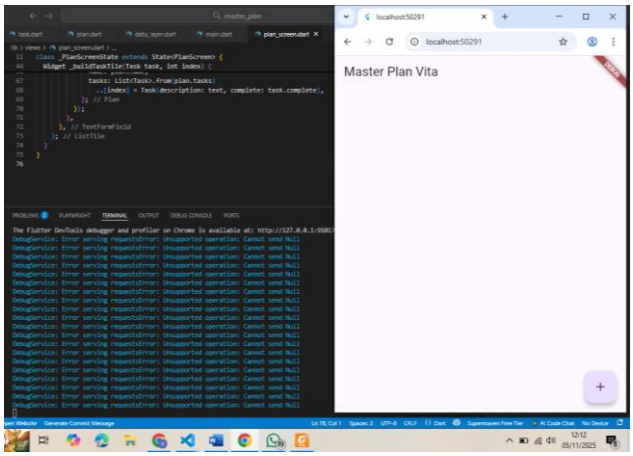
   Jawab : Langkah 4 dilakukan untuk membuat file `data_layer.dart` yang berfungsi sebagai penghubung antar file model di folder `models`. Dengan file ini, tidak perlu mengimpor setiap model satu per satu ke dalam file lain, cukup dengan mengimpor `data_layer.dart` saja.
3. Mengapa perlu variabel plan di langkah 6 pada praktikum tersebut? Mengapa dibuat konstanta ?

   Jawab:

   - Variabel `plan` digunakan untuk menyimpan data utama berupa daftar tugas (tasks) yang akan ditampilkan dan dimodifikasi di layar. Variabel ini berperan sebagai *state* dari aplikasi yang nilainya dapat berubah ketika pengguna menambah, menghapus, atau menandai tugas sebagai selesai.
   - Pada saat inisialisasi, `plan` dibuat sebagai konstanta (`const Plan()`) karena nilainya masih bersifat tetap atau belum memiliki data yang dinamis. Konstanta menandakan bahwa objek awal tersebut belum mengalami perubahan sebelum dimodifikasi melalui fungsi `setState()`.

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

   Jawab:



   - Pada langkah 9, tampilan aplikasi menampilkan daftar tugas dengan judul “Master Plan Vita”. Di bagian bawah terdapat tombol *Floating Action Button* berbentuk ikon “+” untuk menambah tugas baru.
   - Jika tombol tersebut ditekan, akan muncul baris tugas baru yang terdiri dari *TextFormField* untuk menuliskan deskripsi tugas dan *Checkbox* untuk menandai apakah tugas tersebut sudah selesai atau belum.

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

   Jawab:

   - **Langkah 11 – Method `initState()`**

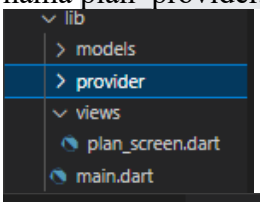
     Method ini digunakan untuk melakukan inisialisasi pertama kali ketika widget `PlanScreen` dibuat. Pada bagian ini, *ScrollController* diinisialisasi dan ditambahkan *listener* untuk menghapus fokus dari *TextField* saat pengguna melakukan *scroll*. Tujuannya agar keyboard otomatis tertutup ketika pengguna menggulir layar, terutama pada perangkat iOS.
   - **Langkah 13 – Method `dispose()`**

     Method ini digunakan untuk membersihkan resource ketika widget sudah tidak digunakan lagi. Dalam hal ini, *scrollController* di-*dispose* agar tidak terjadi kebocoran memori (*memory leak*).

6. Kumpulkan laporan praktikum Anda berupa link commit atau repository GitHub ke dosen yang telah disepakati !
   Jawab: https://github.com/vitasaraswati/pem_mobile/tree/main/week_9/master_plan

PRAKTIKUM II - MENGELOLA DATA LAYER DENGAN INHERITEDWIDGET DAN INHERITEDNOTIFIER

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




```
plan_provider.dart
master_plan > lib > provider > plan_provider.dart > PlanProvider
1  import 'package:flutter/material.dart';
2  import '../models/data_layer.dart';
3
4  class PlanProvider extends InheritedNotifier<ValueNotifier<Plan>> {
5    const PlanProvider({super.key, required Widget child, required
6     ValueNotifier<Plan> notifier})
7    : super(child: child, notifier: notifier);
8
9    static ValueNotifier<Plan> of(BuildContext context) {
10    return context.
11     dependOnInheritedWidgetOfExactType<PlanProvider>()!.notifier!;
12   }
13 }
```

2. Edit file main.dart. Gantilah pada bagian atribut home dengan PlanProvider seperti berikut. Jangan lupa sesuaikan bagian impor jika dibutuhkan

```
11   @override
12   Widget build(BuildContext context) {
13     return MaterialApp(
14       theme: ThemeData(primarySwatch: Colors.purple),
15       home: PlanProvider(
16         notifier: ValueNotifier<Plan>(const Plan()),
17         child: const PlanScreen(),
18       ), // PlanProvider
19     ); // MaterialApp
20   }
21 }
```

3. **Tambah method pada model** plan.dart
   Tambahkan dua *method* di dalam model class Plan seperti kode berikut

```
master_plan > lib > models > plan.dart > ...
1  import './task.dart';
2
3  class Plan {
4    final String name;
5    final List<Task> tasks;
6
7    const Plan({this.name = '', this.tasks = const []});
8
9    // Getter untuk menghitung jumlah task yang sudah selesai
10   int get completedCount => tasks
11       .where((task) => task.complete)
12       .length;
13
14   // Getter untuk menampilkan pesan progress
15   String get completenessMessage =>
16       '$completedCount out of ${tasks.length} tasks';
17 }
```

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.

```
master_plan > lib > views > plan_screen.dart > _PlanScreenState
1  import '../models/data_layer.dart';
2  import 'package:flutter/material.dart';
3
4  class PlanScreen extends StatefulWidget {
5    const PlanScreen({super.key});
6
7    @override
8    State createState() => _PlanScreenState();
9  }
10
11 class _PlanScreenState extends State<PlanScreen> {
12   late ScrollController scrollController;        Chat (CTRL + I) / Share (CTRL
13
14   @override
15   void initState() {
16     super.initState();
17     scrollController = ScrollController()
18       ..addListener(() {
19         FocusScope.of(context).requestFocus(FocusNode());
20       });
```

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

```
38 Widget _buildAddTaskButton(BuildContext context) {
39   ValueNotifier<Plan> planNotifier = PlanProvider.of(context);
40   return FloatingActionButton(
41     child: const Icon(Icons.add),
42     onPressed: () {
43       Plan currentPlan = planNotifier.value;
44       planNotifier.value = Plan(
45         name: currentPlan.name,
46         tasks: List<Task>.from(currentPlan.tasks)..add(const Task()),
47       ); // Plan
48     },
49   ); // FloatingActionButton
50 }
```

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

```dart
Widget _buildTaskTile(BuildContext context, Plan plan, Task task, int index) {
  ValueNotifier<Plan> planNotifier = PlanProvider.of(context);

  return ListTile(
    leading: Checkbox(
      value: task.complete,
      onChanged: (selected) {
        final updatedTasks = List<Task>.from(plan.tasks);
        updatedTasks[index] = Task(
          description: task.description,
          complete: selected ?? false,
        ); // Task

        planNotifier.value = Plan(name: plan.name, tasks: updatedTasks);
      },
    ), // Checkbox
    title: TextFormField(
      initialValue: task.description,
      onChanged: (text) {
        final updatedTasks = List<Task>.from(plan.tasks);
        updatedTasks[index] = Task(
          description: text,
          complete: task.complete,
        ); // Task

        planNotifier.value = Plan(name: plan.name, tasks: updatedTasks);
      },
    ), // TextFormField
  ); // ListTile
}
}
```

7. **Edit** _buildList
   Sesuaikan parameter pada bagian _buildTaskTile seperti kode berikut

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

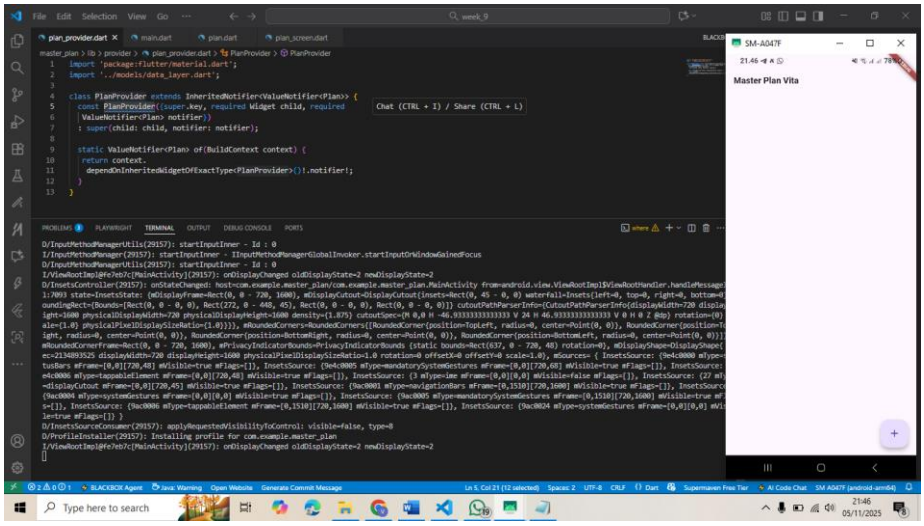
8. **Tetap di** class PlanScreen
   Edit method build sehingga bisa tampil progress pada bagian bawah (footer). Caranya, bungkus (wrap) _buildList dengan widget Expanded dan masukkan ke dalam widget Column seperti kode pada Langkah 9

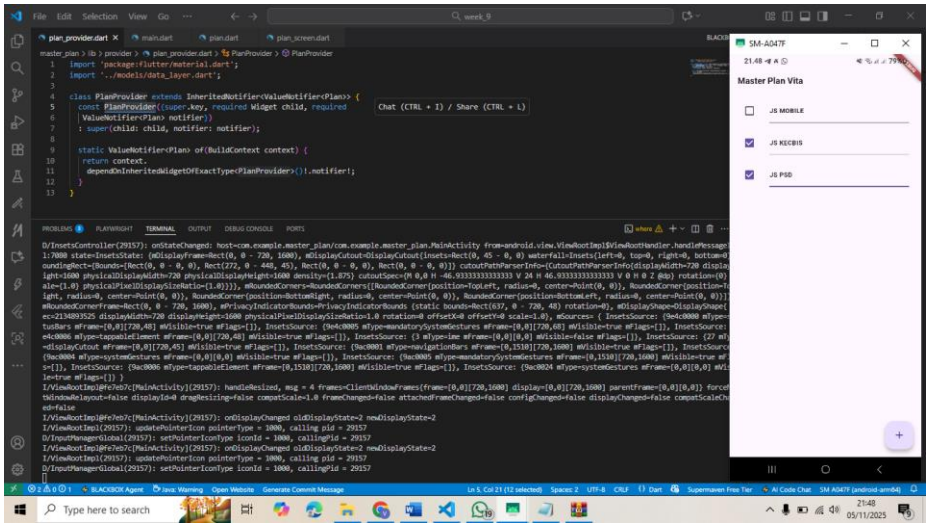
```dart
@override
Widget build(BuildContext context) {
  // Ambil data dari PlanProvider
  ValueNotifier<Plan> planNotifier = PlanProvider.of(context);
  Plan plan = planNotifier.value;

  return Scaffold(
    appBar: AppBar(title: const Text('Master Plan Vita')),
    body: Column(
      children: [
        // Langkah 8: bungkus _buildList() dengan Expanded
        Expanded(child: _buildList(context, plan)),

        // Langkah 9: Tambahkan footer progress di bawah
        Container(
          padding: const EdgeInsets.all(16.0),
          alignment: Alignment.center,
          child: Text(
            plan.completenessMessage,
            style: const TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
          ), // Text
        ), // Container
      ],
    ), // Column
    floatingActionButton: _buildAddTaskButton(context),
  ); // Scaffold
}
```

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

```dart
@override
Widget build(BuildContext context) {
  // Ambil data dari PlanProvider
  ValueNotifier<Plan> planNotifier = PlanProvider.of(context);
  Plan plan = planNotifier.value;

  return Scaffold(
    appBar: AppBar(title: const Text('Master Plan Vita')),
    body: Column(
      children: [
        // Langkah 8: bungkus _buildList() dengan Expanded
        Expanded(child: _buildList(context, plan)),

        // Langkah 9: Tambahkan footer progress di bawah
        Container(
          padding: const EdgeInsets.all(16.0),
          alignment: Alignment.center,
          child: Text(
            plan.completenessMessage,
            style: const TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
          ), // Text
        ), // Container
      ],
    ), // Column
    floatingActionButton: _buildAddTaskButton(context),
  ); // Scaffold
}
```

HASIL OUTPUT PRAKTIKUM 2





TUGAS PRAKTIKUM 2

1. Selesaikan langkah-langkah praktikum tersebut, lalu dokumentasikan berupa GIF hasil akhir praktikum beserta penjelasannya di file README.md! Jika Anda menemukan ada yang error atau tidak berjalan dengan baik, silakan diperbaiki sesuai dengan tujuan aplikasi tersebut dibuat.
2. Jelaskan mana yang dimaksud InheritedWidget pada langkah 1 tersebut! Mengapa yang digunakan InheritedNotifier?

   Jawab:

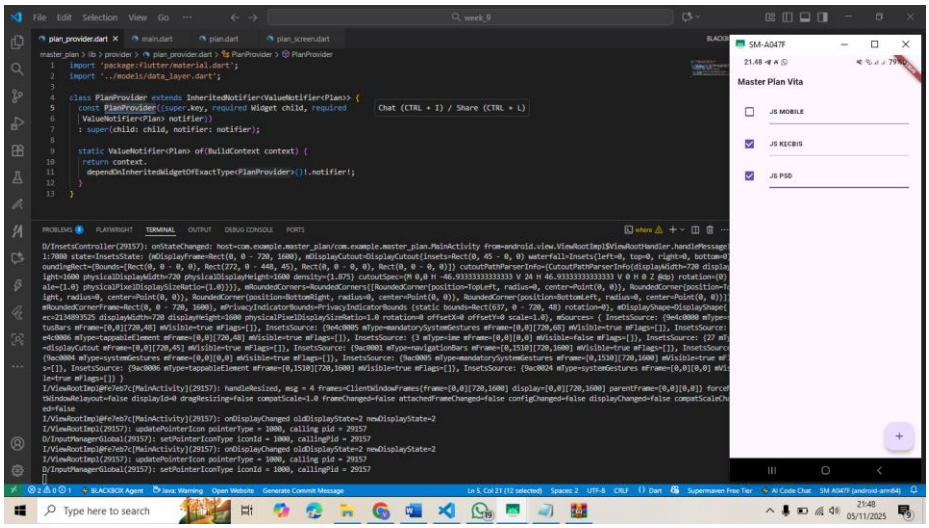
   - `InheritedWidget` adalah widget berfungsi untuk **meneruskan data ke seluruh widget turunannya** dalam *widget tree* tanpa perlu mengirim data satu per satu melalui parameter konstruktor
   - Praktikum ini digunakan **`InheritedNotifier`** merupakan turunan dari `InheritedWidget`. `InheritedNotifier` dapat *memberi tahu (notify)* widget-widget yang tergantung jika terjadi perubahan data (misalnya ketika ada task baru atau status task berubah).
3. Jelaskan maksud dari method di langkah 3 pada praktikum tersebut! Mengapa dilakukan demikian?

   Jawab:

   - `completedCount` digunakan untuk menghitung jumlah tugas yang telah selesai (checkbox dicentang).
   - `completenessMessage` digunakan untuk membuat teks yang menampilkan progres secara dinamis.

     Keduanya dibuat agar tampilan progress di aplikasi bisa **menunjukkan jumlah tugas yang sudah selesai dibanding total tugas yang ada**, misalnya: “2 out of 3 tasks”
4. Lakukan capture hasil dari Langkah 9 berupa GIF, kemudian jelaskan apa yang telah Anda buat!

   Jawab:



Tidak terlihat perubahan pada UI, namun dengan melakukan langkah-langkah di atas, telah menerapkan cara memisahkan dengan baik antara **view** dan **model**. Setiap perubahan pada checkbox atau teks otomatis memperbarui progres tanpa perlu *hot reload*

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

   Jawab: https://github.com/vitasaraswati/pem_mobile/tree/main/week_9/master_plan

PRAKTIKUM III – MEMBUAT STATE DI MULTIPLE SCREEN

1. **Edit** PlanProvider.

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

```
master_plan > lib > provider > plan_provider.dart > ...
4   class PlanProvider extends InheritedNotifier<ValueNotifier<List<Plan>>> {
5     const PlanProvider({
6       super.key,
7       required Widget child,
8       required ValueNotifier<List<Plan>> notifier,
9     }) : super(child: child, notifier: notifier);
10
11    static ValueNotifier<List<Plan>> of(BuildContext context) {
12      return context
13          .dependOnInheritedWidgetOfExactType<PlanProvider>()!
14          .notifier!;
15    }
16  }
```

2. **Edit** main.dart

   Langkah sebelumnya dapat menyebabkan error pada main.dart dan plan_screen.dart.

   Pada method build, gantilah menjadi kode seperti ini

```
11    @override
12    Widget build(BuildContext context) {
13      return PlanProvider(
14        notifier: ValueNotifier<List<Plan>>(const []),
15        child: MaterialApp(
16          title: 'State management app',
17          theme: ThemeData(primarySwatch: Colors.blue),
18          home: const PlanScreen(),
19        ), // MaterialApp
20      ); // PlanProvider
21    }
22  }
```

3. Edit plan_screen.dart

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

```
5   class PlanScreen extends StatefulWidget {
6     final Plan plan; // Tambahkan variabel plan
7     const PlanScreen({super.key, required this.plan}); // Tambahkan ke constructor
8
9     @override
```

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

5. **Tambah** getter Plan

   Tambahkan getter pada _PlanScreenState seperti kode berikut.

```dart
// Langkah 5: Tambah getter Plan
Plan get plan => widget.plan;

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

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

@override
Widget build(BuildContext context) {
```

6. **Method** initState()

   Pada bagian ini kode tetap seperti berikut.

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

@override
void dispose() {
```

7. **Widget** build

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

```dart
@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,
        ); // cari plan aktif
        return Column(
          children: [
            Expanded(child: _buildList(currentPlan)),
            SafeArea(
              child: Text(
                currentPlan.completenessMessage,
```

```
                    style: const TextStyle(
                      fontWeight: FontWeight.bold,
                      fontSize: 16,
                    ), // TextStyle
                  ), // Text
                ), // SafeArea
              ],
            ); // Column
          },
        ), // ValueListenableBuilder
        floatingActionButton: _buildAddTaskButton(context),
      ); // Scaffold
    }

    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);

          // update plan lokal
          setState(() {
            widget.plan.tasks.add(const Task());
          });
        },
      ); // FloatingActionButton
    }
```

8. **Edit** _buildTaskTile

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

```
  Widget _buildTaskTile(Plan plan, Task task, int index) {
    ValueNotifier<List<Plan>> planNotifier = PlanProvider.of(context);

    return ListTile(
      leading: Checkbox(
        value: task.complete,
        onChanged: (selected) {
          Plan currentPlan = plan;
          int planIndex = planNotifier.value.indexWhere(
            (p) => p.name == currentPlan.name,
          );
          planNotifier.value = List<Plan>.from(planNotifier.value)
            ..[planIndex] = Plan(
              name: currentPlan.name,
              tasks: List<Task>.from(currentPlan.tasks)
                ..[index] = Task(
                  description: task.description,
                  complete: selected ?? false,
                ), // Task
            ); // Plan
        },
      ), // Checkbox
      title: TextFormField(
        initialValue: task.description,
        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),
            ); // Plan
        },
      ), // TextFormField
    ); // ListTile
  }
}
```

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.

```
master_plan > lib > views > plan_creator_screen.dart > ...
1  import 'package:flutter/material.dart';
2
3  class PlanCreatorScreen extends StatefulWidget {
4    const PlanCreatorScreen({super.key});
5
6    @override
7    State<PlanCreatorScreen> createState() => _PlanCreatorScreenState();
8  }
9
10 class _PlanCreatorScreenState extends State<PlanCreatorScreen> {
11   @override
12   Widget build(BuildContext context) {
13     return const Scaffold(
14       body: Center(
15         child: Text('Plan Creator Screen'),
16       ), // Center
17     ); // Scaffold
18   }
19 }
```

10. Pindah ke class _PlanCreatorScreenStat

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

```
master_plan > lib > main.dart > ...
1  import 'package:flutter/material.dart';
2  import './views/plan_creator_screen.dart'; // ubah ke screen baru
3  import './provider/plan_provider.dart';
4  import './models/data_layer.dart';
5
   Run | Debug | Profile
6  void main() => runApp(const MasterPlanApp());
7
8  class MasterPlanApp extends StatelessWidget {
9    const MasterPlanApp({super.key});
10
11   @override
12   Widget build(BuildContext context) {
13     return PlanProvider(
14       notifier: ValueNotifier<List<Plan>>(const []),
15       child: MaterialApp(
16         title: 'State management app',
17         theme: ThemeData(primarySwatch: Colors.blue),
18         // Langkah 9: ubah home jadi PlanCreatorScreen
19         home: const PlanCreatorScreen(),
20       ), // MaterialApp
21     ); // PlanProvider
22   }
23 }
```

11. Pindah ke method build

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

```
16   @override
17   Widget build(BuildContext context) {
18     return Scaffold(
19       appBar: AppBar(title: const Text('Master Plans Vita')),
20       body: Column(
21         children: [
22           _buildListCreator(),
23           Expanded(child: _buildMasterPlans()),
24         ],
25       ), // Column
26     ); // Scaffold
27   }
28
```

12. **Buat widget** _buildListCreator

    Buatlah widget berikut setelah widget build.

```dart
Widget _buildListCreator() {
  return Padding(
    padding: const EdgeInsets.all(20.0),
    child: Material(
      color: Theme.of(context).cardColor,
      elevation: 10,
      child: TextField(
        controller: textController,
        decoration: const InputDecoration(
          labelText: 'Add a plan',
          contentPadding: EdgeInsets.all(20),
        ), // InputDecoration
        onEditingComplete: addPlan,
      ), // TextField
    ), // Material
  ); // Padding
}

@override
void dispose() {
```

13. **Buat** void addPlan()

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

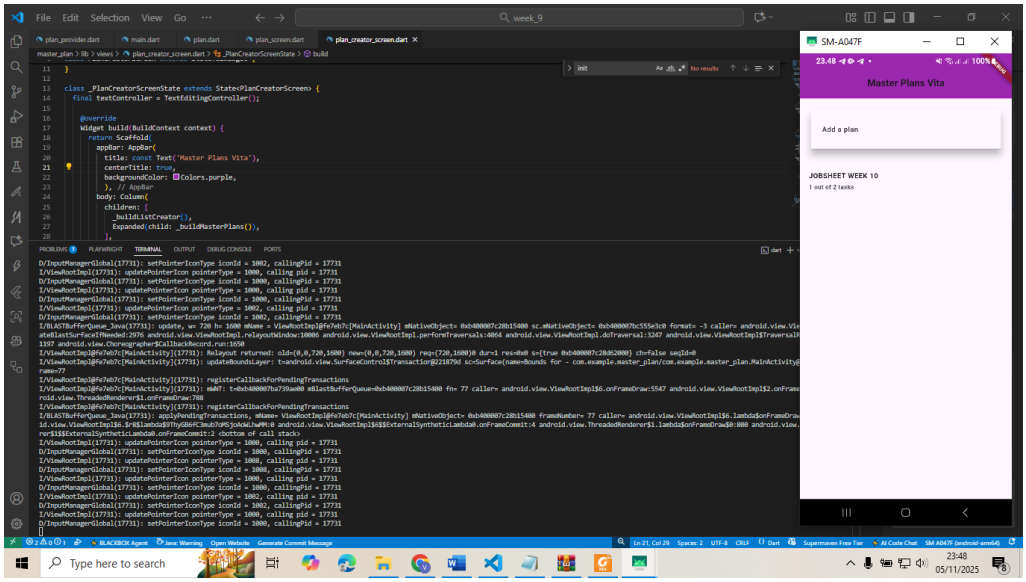
```dart
void addPlan() {
  final text = textController.text;
  if (text.isEmpty) {
    return;
  }

  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(() {});
}

@override
void dispose() {
```

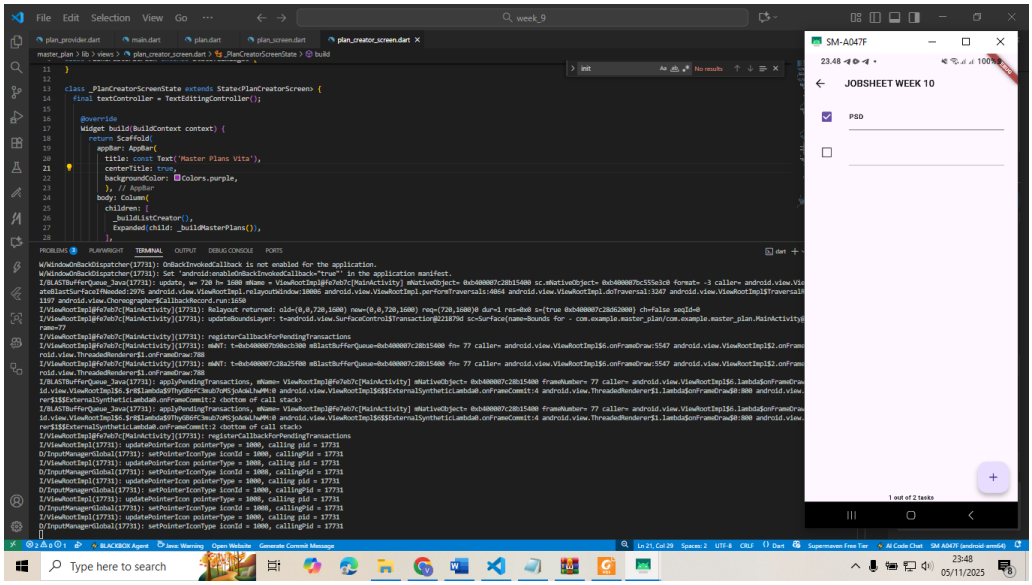
14. **Buat** widget _buildMasterPlans()

    Tambahkan widget seperti kode berikut.

```dart
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)));
        },
      ); // ListTile
    },
  ); // ListView.builder
}

@override
void dispose() {
```

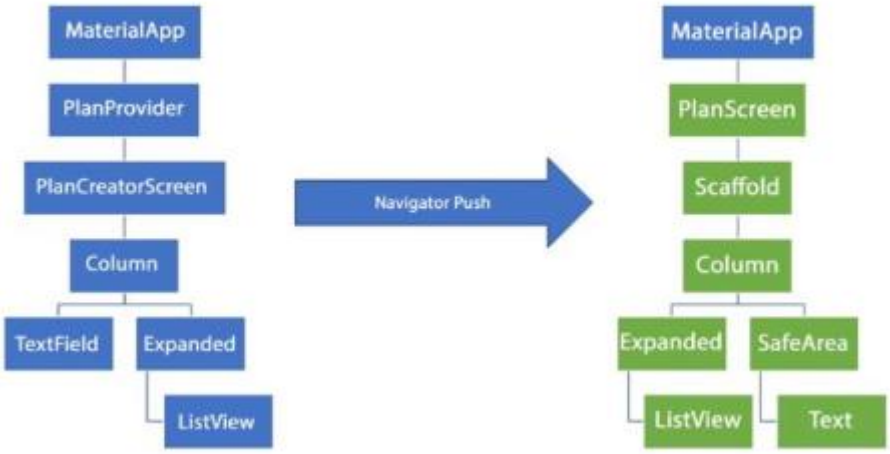
HASIL OUTPUT PRAKTIKUM 3





TUGAS PRAKTIKUM III

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



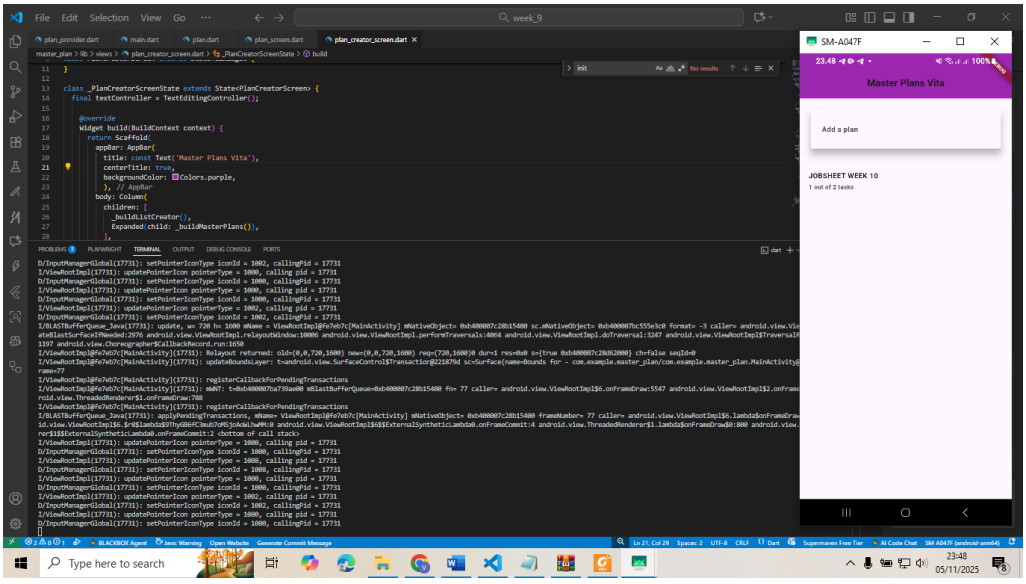
Jawab :

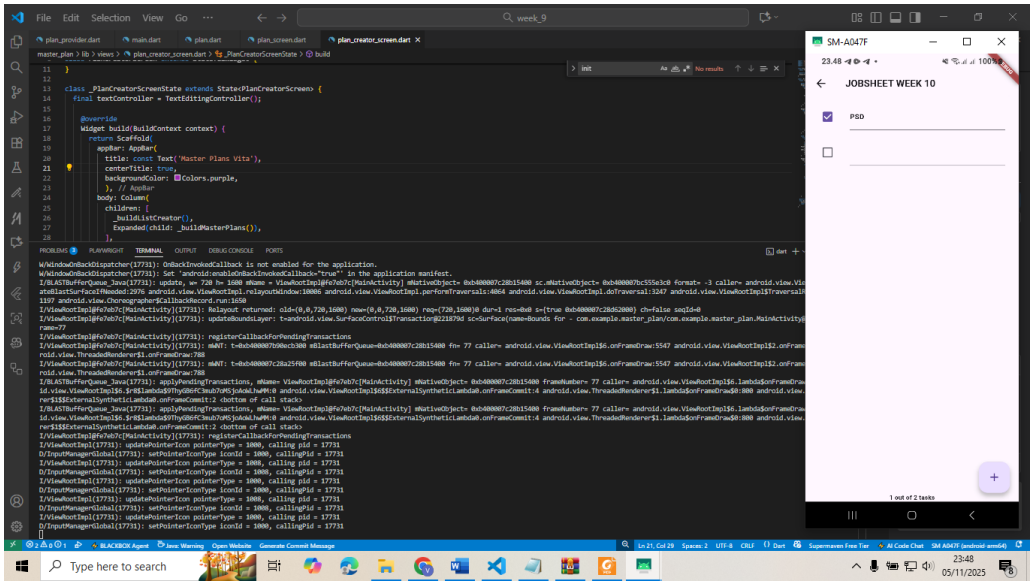
Gambar diagram tersebut menunjukkan **alur navigasi dan perubahan struktur widget (widget tree)** pada aplikasi Flutter dari **halaman daftar rencana (PlanCreatorScreen)** menuju **halaman detail rencana (PlanScreen)**.

- Di sisi kiri, `PlanCreatorScreen` berisi `TextField` untuk menambah rencana dan `ListView` untuk menampilkan daftar rencana. Ketika salah satu rencana ditekan, aplikasi melakukan **navigasi menggunakan `Navigator.push()`** ke halaman baru (`PlanScreen`).
- Di sisi kanan, `PlanScreen` menampilkan **daftar tugas (tasks)** dari rencana yang dipilih dalam `ListView` dan status penyelesaiannya dalam `SafeArea` berisi `Text`.
- Dengan demikian, diagram tersebut menggambarkan **proses perpindahan halaman serta struktur widget utama yang terlibat** dalam alur aplikasi Master Plan.

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

Jawab:





Langkah 14, kita membuat sebuah widget bernama `_buildMasterPlans()` yang berfungsi untuk **menampilkan daftar rencana (plans)** yang sudah dibuat oleh pengguna.

- Widget ini mengambil data dari `PlanProvider` menggunakan `ValueNotifier<List<Plan>>`.
- Jika daftar plan masih kosong, maka akan muncul tampilan **ikon catatan** dan teks *"Anda belum memiliki rencana apapun."*
- Jika sudah ada data, widget akan menampilkan **ListView** berisi daftar `Plan`, dengan:
    - **`title`** = nama rencana (`plan.name`)
    - **`subtitle`** = pesan tingkat penyelesaian (`plan.completenessMessage`)
    - **`onTap`** = saat item diklik, aplikasi akan **berpindah ke halaman detail rencana (`PlanScreen`)** menggunakan `Navigator.push()`.

Hasil yang ditunjukkan di GIF:

- Pengguna mengetik nama rencana di `TextField` lalu menekan *Enter*.
- Rencana baru muncul di daftar.
- Saat salah satu rencana diklik, aplikasi berpindah ke halaman detail (`PlanScreen`).
- Pada halaman tersebut, pengguna bisa menambahkan atau mengedit tugas (task) di dalam rencana.

Langkah 14 ini membuat fitur utama aplikasi yaitu **membuat, menampilkan, dan membuka detail dari daftar rencana pengguna** secara interaktif menggunakan `ValueNotifier` dan `Navigator.push()`.

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

Jawab: https://github.com/vitasaraswati/pem_mobile/tree/main/week_9/master_plan