

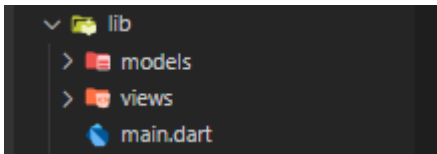
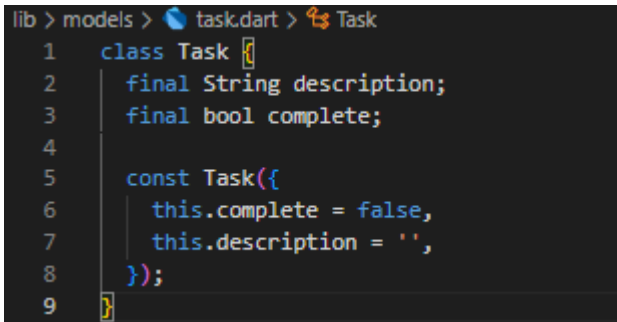


Mata Kuliah : Pemrograman Mobile
Program Studi : D4 – Sistem Informasi Bisnis
Semester : 5

Kelas : SIB - 3D
NIM : 2341760178
Nama : Saria Fauzani
Jobsheet Ke- : 10
Link Github : https://github.com/Sariafauzani/master_plan

Laporan Jobsheet

Praktikum 1: Dasar State dengan Model-View

Langkah	Jawaban/Deskripsi
1	Langkah 1: Buat Project Baru
	<p>Buatlah sebuah project flutter baru dengan nama master_plan. Lalu buatlah susunan folder dalam project seperti gambar berikut ini.</p> 
2	Langkah 2: Membuat model task.dart
	<p>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:</p> 



3	Langkah 3: Buat file plan.dart
	<p>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.</p> <pre>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 }</pre>
4	Langkah 4: Buat file data_layer.dart
	<p>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.</p> <pre>lib > models > data_layer.dart 1 export 'plan.dart'; 2 export 'task.dart';</pre>
5	Langkah 5: Pindah ke file main.dart
	<p>Ubah isi kode main.dart sebagai berikut.</p> <pre>lib > main.dart > MasterPlanApp 1 import 'package:flutter/material.dart'; 2 import './views/plan_screen.dart'; 3 4 Run Debug Profile 5 void main() => runApp(MasterPlanApp()); 6 7 class MasterPlanApp extends StatelessWidget { 8 const MasterPlanApp({super.key}); 9 10 @override 11 Widget build(BuildContext context) { 12 return MaterialApp(13 theme: ThemeData(primarySwatch: Colors.purple), 14 home: PlanScreen(), 15); // MaterialApp 16 }</pre>
6	Langkah 6: buat plan_screen.dart



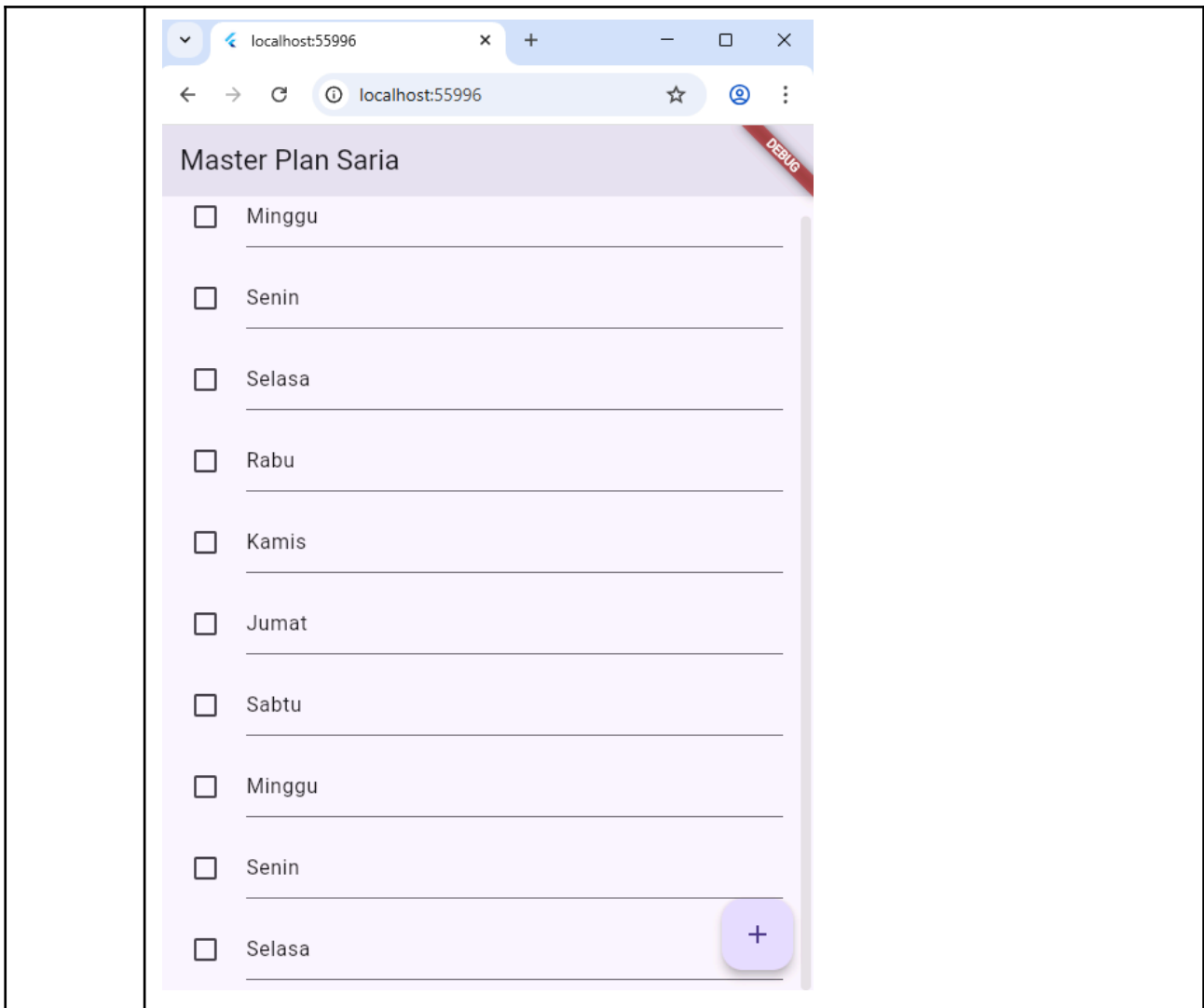
	<p>Pada folder views, buatlah sebuah file plan_screen.dart dan gunakan templat StatefulWidget untuk membuat class PlanScreen. Isi kodenya adalah sebagai berikut. Gantilah teks 'Namaku' dengan nama panggilan Anda pada title AppBar.</p> <pre>lib > views > plan_screen.dart > _PlanScreenState 1 import '../models/data_layer.dart'; 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 Plan plan = const Plan(); 13 14 @override 15 Widget build(BuildContext context) { 16 return Scaffold(17 // ganti 'Namaku' dengan Nama panggilan Anda 18 appBar: AppBar(title: const Text('Master Plan Saria')), 19 body: _buildList(), 20 floatingActionButton: _buildAddTaskButton(), 21); // Scaffold 22 } 23 }</pre>
7	Langkah 7: buat method _buildAddTaskButton()
	<p>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.</p> <pre>24 Widget _buildAddTaskButton() { 25 return FloatingActionButton(26 child: const Icon(Icons.add), 27 onPressed: () { 28 setState(() { 29 plan = Plan(30 name: plan.name, 31 tasks: List<Task>.from(plan.tasks) 32 ..add(const Task()), 33); // Plan 34 }); 35 }, 36); // FloatingActionButton 37 }</pre>
8	Langkah 8: buat widget _buildList()



	<p>Kita akan buat widget berupa List yang dapat dilakukan scroll, yaitu ListView.builder. Buat widget ListView seperti kode berikut ini.</p> <pre>39 Widget _buildList() { 40 return ListView.builder(41 itemCount: plan.tasks.length, 42 itemBuilder: (context, index) => 43 _buildTaskTile(plan.tasks[index], index), 44); 45 }</pre>
9	Langkah 9: buat widget _buildTaskTile
	<p>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.</p> <pre>47 Widget _buildTaskTile(Task task, int index) { 48 return ListTile(49 leading: Checkbox(50 value: task.complete, 51 onChanged: (selected) { 52 setState(() { 53 plan = Plan(54 name: plan.name, 55 tasks: List<Task>.from(plan.tasks) 56 ..[index] = Task(57 description: task.description, 58 complete: selected ?? false, 59), // Task 60); // Plan 61 }); 62 }), // Checkbox 63 title: TextFormField(64 initialValue: task.description, 65 onChanged: (text) { 66 setState(() { 67 plan = Plan(68 name: plan.name, 69 tasks: List<Task>.from(plan.tasks) 70 ..[index] = Task(71 description: text, 72 complete: task.complete, 73), // Task 74); // Plan 75 }); 76 }), // TextFormField 77), // ListTile 78); 79 } 80 }</pre>
10	Langkah 10: Tambah Scroll Controller



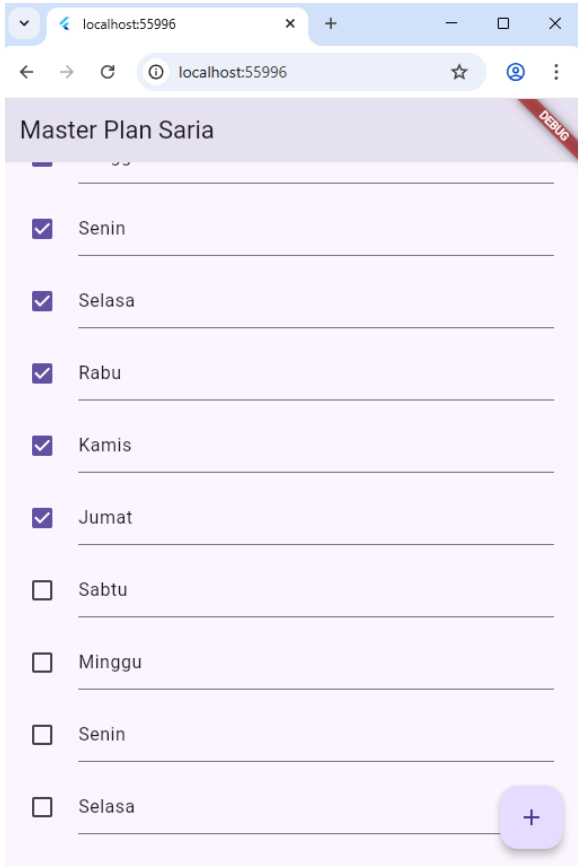
	<p>Anda dapat menambah tugas sebanyak-banyaknya, menandainya jika sudah beres, dan melakukan scroll jika sudah semakin banyak isinya. Namun, ada salah satu fitur tertentu di iOS perlu kita tambahkan. Ketika keyboard tampil, Anda akan kesulitan untuk mengisi yang paling bawah. Untuk mengatasi itu, Anda 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.</p> <pre>class _PlanScreenState extends State<PlanScreen> { Plan plan = const Plan(); late ScrollController scrollController;</pre>
11	Langkah 11: Tambah Scroll Listener
	<p>Tambahkan method initState() setelah deklarasi variabel scrollController seperti kode berikut.</p> <pre>15 @override 16 void initState() { 17 super.initState(); 18 scrollController = ScrollController() 19 ..addListener(() { 20 FocusScope.of(context).requestFocus(FocusNode()); 21 }); 22 }</pre>
12	Langkah 12: Tambah controller dan keyboard behavior
	<p>Tambahkan controller dan keyboard behavior pada ListView di method _buildList seperti kode berikut ini.</p> <pre>55 return ListView.builder(56 controller: scrollController, 57 keyboardDismissBehavior: Theme.of(context).platform == TargetPlatform.iOS 58 ? ScrollViewKeyboardDismissBehavior.onDrag 59 : ScrollViewKeyboardDismissBehavior.manual, 60 itemCount: plan.tasks.length,</pre>
13	Langkah 13: Terakhir, tambah method dispose()
	<p>Terakhir, tambahkan method dispose() berguna ketika widget sudah tidak digunakan lagi.</p> <pre>24 @override 25 void dispose() { 26 scrollController.dispose(); 27 super.dispose(); 28 }</pre>
14	Hasil



Tugas Praktikum 1: Dasar State dengan Model-View

Langkah	Jawaban/Deskripsi
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: Pada langkah 4 diminta membuat file data_layer.dart di folder models yang berfungsi sebagai penghubung (wrapper) yang mengeksport ulang file plan.dart dan task.dart. Mengapa dilakukan karena agar proses impor di bagian lain aplikasi menjadi lebih ringkas, rapi, dan mudah dikelola.



3	<p>Mengapa perlu variabel plan di langkah 6 pada praktikum tersebut? Mengapa dibuat konstanta ?</p> <p>Jawab: Karena variabel tersebut digunakan untuk menyimpan data rencana utama (plan) yang akan ditampilkan dan dikelola di layar PlanScreen. Sedangkan mengapa dibuat konstanta, karena saat ini objek masih bersifat tetap (belum ada data dinamis). const membantu efisiensi memori dan performa.</p>
4	<p>Lakukan capture hasil dari Langkah 9 berupa GIF, kemudian jelaskan apa yang telah Anda buat!</p> <p>Jawab:</p>  <p>Pada langkah ini, membuat tampilan interaktif untuk aplikasi Master Plan. Pengguna dapat menambahkan tugas baru (task) ke dalam daftar rencana (plan). Tombol “+” di pojok kanan bawah berfungsi untuk menambah task baru. Tampilan daftar akan otomatis diperbarui setiap kali ada task baru yang dimasukkan.</p>
5	<p>Apa kegunaan method pada Langkah 11 dan 13 dalam lifecycle state ?</p> <p>Jawab: Method yang digunakan pada langkah 11 yaitu untuk menyiapkan dan mengaktifkan logika awal (misalnya scroll listener). Sedangkan method pada langkah</p>



	13 digunakan untuk membersihkan dan menonaktifkan logika ketika widget tidak lagi digunakan.
6	Kumpulkan laporan praktikum Anda berupa link commit atau repository GitHub ke dosen yang telah disepakati !

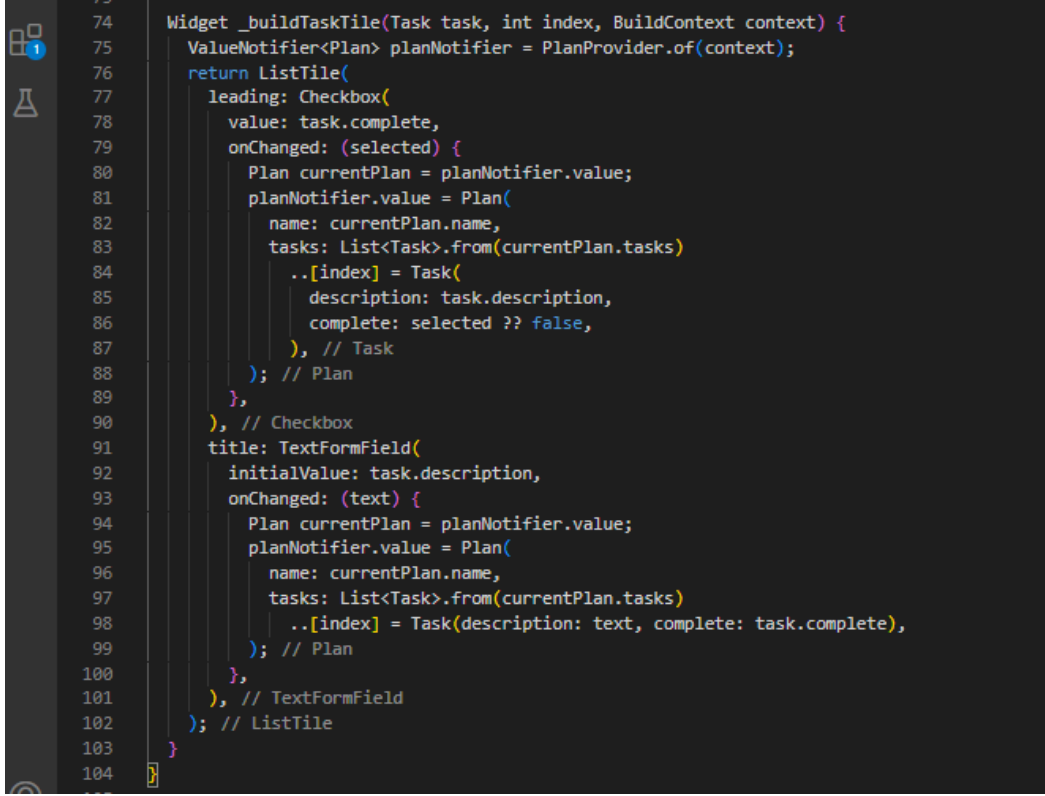
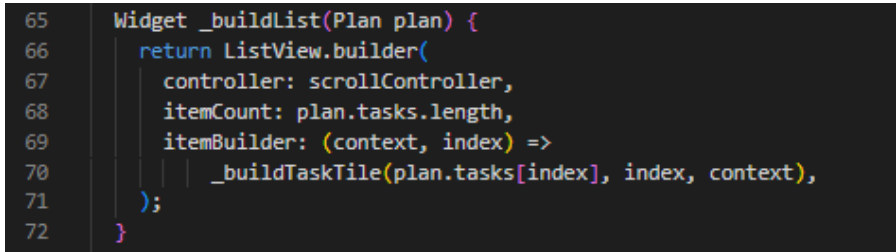
Praktikum 2: Mengelola Data Layer dengan InheritedWidget dan InheritedNotifier

Langkah	Jawaban/Deskripsi
1	Langkah 1: Buat file plan_provider.dart
	<p>Buat folder baru provider di dalam folder lib, lalu buat file baru dengan nama plan_provider.dart berisi kode seperti berikut.</p>
2	Langkah 2: Edit main.dart
	<p>Gantilah pada bagian atribut home dengan PlanProvider seperti berikut. Jangan lupa sesuaikan bagian impor jika dibutuhkan.</p>
3	Langkah 3: Tambah method pada model plan.dart
	Tambahkan dua method di dalam model class Plan seperti kode berikut.



	<pre>7 const Plan({this.name = '', this.tasks = const []}); 8 int get completedCount => tasks.where((task) => task.complete).length; 9 10 String get completenessMessage => 11 '\$completedCount out of \${tasks.length} tasks'; 12 }</pre>
4	Langkah 4: Pindah ke PlanScreen
	<p>Edit PlanScreen agar menggunakan data dari PlanProvider. Hapus deklarasi variabel plan (ini akan membuat error). Kita akan perbaiki pada langkah 5 berikut ini.</p> <pre>11 class _PlanScreenState extends State<PlanScreen> { 12 // 13 late ScrollController scrollController;</pre>
5	Langkah 5: Edit method _buildAddTaskButton
	<p>Tambahkan BuildContext sebagai parameter dan gunakan PlanProvider sebagai sumber datanya. Edit bagian kode seperti berikut.</p> <pre>51 Widget _buildAddTaskButton(BuildContext context) { 52 ValueNotifier<Plan> planNotifier = PlanProvider.of(context); 53 return FloatingActionButton(54 child: const Icon(Icons.add), 55 onPressed: () { 56 Plan currentPlan = planNotifier.value; 57 planNotifier.value = Plan(58 name: currentPlan.name, 59 tasks: List<Task>.from(currentPlan.tasks)..add(const Task()), 60); // Plan 61 }, 62); // FloatingActionButton 63 }</pre>
6	Langkah 6: Edit method _buildTaskTile
	<p>Tambahkan parameter BuildContext, gunakan PlanProvider sebagai sumber data. Ganti TextField menjadi TextFormField untuk membuat inisial data provider menjadi lebih mudah.</p>



	 <pre>74 Widget _buildTaskTile(Task task, int index, BuildContext context) { 75 ValueNotifier<Plan> planNotifier = PlanProvider.of(context); 76 return ListTile(77 leading: Checkbox(78 value: task.complete, 79 onChanged: (selected) { 80 Plan currentPlan = planNotifier.value; 81 planNotifier.value = Plan(82 name: currentPlan.name, 83 tasks: List<Task>.from(currentPlan.tasks) 84 ..[index] = Task(85 description: task.description, 86 complete: selected ?? false, 87), // Task 88); // Plan 89 }, 90), // Checkbox 91 title: TextFormField(92 initialValue: task.description, 93 onChanged: (text) { 94 Plan currentPlan = planNotifier.value; 95 planNotifier.value = Plan(96 name: currentPlan.name, 97 tasks: List<Task>.from(currentPlan.tasks) 98 ..[index] = Task(description: text, complete: task.complete), 99); // Plan 100 }, 101), // TextFormField 102); // ListTile 103 } 104</pre>
7	Langkah 7: Edit _buildList
	<p>Sesuaikan parameter pada bagian _buildTaskTile seperti kode berikut.</p>  <pre>65 Widget _buildList(Plan plan) { 66 return ListView.builder(67 controller: scrollController, 68 itemCount: plan.tasks.length, 69 itemBuilder: (context, index) => 70 _buildTaskTile(plan.tasks[index], index, context), 71); 72 }</pre>
8	Langkah 8: Tetap di class PlanScreen
	<p>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.</p>
9	Langkah 9: Tambah widget SafeArea
	<p>Terakhir, tambahkan widget SafeArea dengan berisi completenessMessage pada akhir widget Column. Perhatikan kode berikut ini.</p>

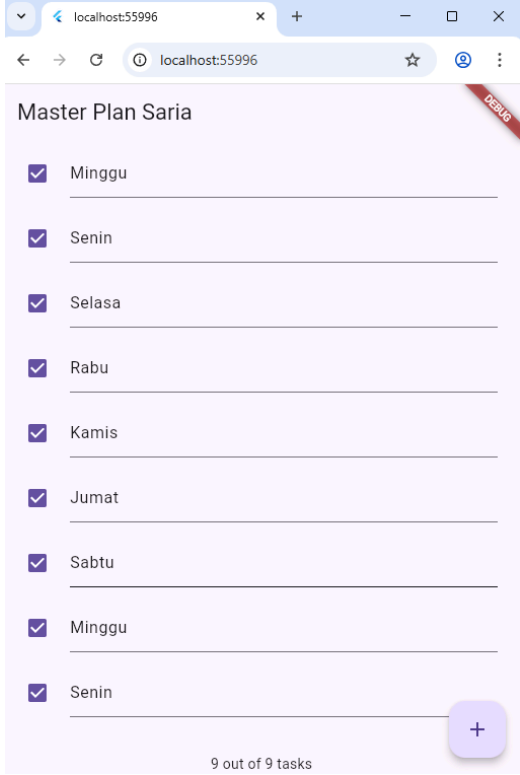


```
31 @override
32 Widget build(BuildContext context) {
33   return Scaffold(
34     // ganti 'Namaku' dengan Nama panggilan Anda
35     appBar: AppBar(title: const Text('Master Plan Saria')),
36     body: ValueListenableBuilder<Plan>(
37       valueListenable: PlanProvider.of(context),
38       builder: (context, plan, child) {
39         return Column(
40           children: [
41             Expanded(child: _buildList(plan)),
42             SafeArea(child: Text(plan.completenessMessage)),
43           ],
44         ); // Column
45       },
46     ), // ValueListenableBuilder
47     floatingActionButton: _buildAddTaskButton(context),
48   ); // Scaffold
49 }
```

Tugas Praktikum 2: InheritedWidget

Langkah	Jawaban/Deskripsi
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 dasar bawaan Flutter yang digunakan untuk mendistribusikan data dari atas pohon widget ke bawah (child widgets) tanpa perlu meneruskan data secara manual lewat konstruktor. Mengapa digunakan, karena aplikasi Master Plan perlu mengubah dan memantau data Plan secara dinamis (tambah tugas, centang, ubah teks), maka InheritedNotifier lebih efisien dan reaktif.
3	Jelaskan maksud dari method di langkah 3 pada praktikum tersebut! Mengapa dilakukan demikian? Jawab: Pada langkah ini ditambahkan dua getter (completedCount dan completenessMessage) untuk menghitung jumlah tugas yang sudah selesai dan menampilkan pesan progres. completedCount menghitung jumlah tugas dengan status complete = true, sedangkan completenessMessage menyusun teks seperti “2 out of 5 tasks”. Hal ini dilakukan agar aplikasi dapat menampilkan kemajuan kerja pengguna secara otomatis setiap kali ada perubahan pada daftar tugas.



4	<p>Lakukan capture hasil dari Langkah 9 berupa GIF, kemudian jelaskan apa yang telah Anda buat!</p> <p>Jawab:</p>  <p>Setiap kali pengguna menambahkan atau mencentang tugas, jumlah tugas yang selesai dihitung otomatis melalui getter <code>completedCount</code> di <code>plan.dart</code>. Pesan progres di bawah (“9 out of 9 tasks”) menampilkan hasil perhitungan tersebut. Ini menunjukkan semua 9 tugas sudah selesai (complete). Tombol + di kanan bawah berfungsi untuk menambah tugas baru ke dalam daftar.</p>
5	<p>Kumpulkan laporan praktikum Anda berupa link commit atau repository GitHub ke dosen yang telah disepakati !</p>

Praktikum 3: Membuat State di Multiple Screens

Langkah	Jawaban/Deskripsi
1	Langkah 1: Edit PlanProvider



	<p>Perhatikan kode berikut, edit class PlanProvider sehingga dapat menangani List Plan.</p> <pre>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 }</pre>
2	Langkah 2: Edit main.dart
	<p>Langkah sebelumnya dapat menyebabkan error pada main.dart dan plan_screen.dart. Pada method build, gantilah menjadi kode seperti ini.</p> <pre>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 }</pre>
3	Langkah 3: Edit plan_screen.dart
	<p>Tambahkan variabel plan dan atribut pada constructor-nya seperti berikut.</p> <pre>5 class PlanScreen extends StatefulWidget { 6 final Plan plan; 7 const PlanScreen({super.key, required this.plan}); 8 }</pre>
4	Langkah 4: Error
	<p>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.</p>
5	Langkah 5: Tambah getter Plan
	<p>Tambahkan getter pada _PlanScreenState seperti kode berikut.</p>



	<pre>class _PlanScreenState extends State<PlanScreen> { late ScrollController scrollController; Plan get plan => widget.plan; }</pre>
6	Langkah 6: Method initState()
	<p>Pada bagian ini kode tetap seperti berikut.</p> <pre>17 @override 18 void initState() { 19 super.initState(); 20 scrollController = ScrollController() 21 ..addListener() { 22 FocusScope.of(context).requestFocus(FocusNode()); 23 } 24 }</pre>
7	Langkah 7: Widget build
	<p>Pastikan Anda telah merubah ke List dan mengubah nilai pada currentPlan seperti kode berikut ini.</p> <pre>34 @override 35 Widget build(BuildContext context) { 36 ValueNotifier<List<Plan>> plansNotifier = PlanProvider.of(context); 37 38 return Scaffold(39 appBar: AppBar(title: Text(plan.name)), 40 body: ValueListenableBuilder<List<Plan>>(41 valueListenable: plansNotifier, 42 builder: (context, plans, child) { 43 Plan currentPlan = plans.firstWhere((p) => p.name == plan.name); 44 return Column(45 children: [46 Expanded(child: _buildList(currentPlan)), 47 SafeArea(child: Text(currentPlan.completenessMessage)), 48], 49); // Column 50 }, 51), // ValueListenableBuilder 52 floatingActionButton: _buildAddTaskButton(context), 53); // Scaffold 54 } 55 56 Widget _buildAddTaskButton(BuildContext context) { 57 ValueNotifier<List<Plan>> planNotifier = PlanProvider.of(context); 58 return FloatingActionButton(59 child: const Icon(Icons.add), 60 onPressed: () { 61 Plan currentPlan = plan; 62 int planIndex = planNotifier.value.indexWhere(63 (p) => p.name == currentPlan.name, 64); 65 List<Task> updatedTasks = List<Task>.from(currentPlan.tasks) 66 ..add(const Task()); 67 planNotifier.value = List<Plan>.from(planNotifier.value) 68 ..[planIndex] = Plan(name: currentPlan.name, tasks: updatedTasks); 69 plan = Plan(name: currentPlan.name, tasks: updatedTasks); 70 }, 71); // FloatingActionButton 72 }</pre>



8	Langkah 8: Edit _buildTaskTile
	<p>Pastikan ubah ke List dan variabel planNotifier seperti kode berikut ini.</p> <pre>83 Widget _buildTaskTile(Task task, int index, BuildContext context) { 84 ValueNotifier<List<Plan>> planNotifier = PlanProvider.of(context); 85 86 return ListTile(87 leading: Checkbox(88 value: task.complete, 89 onChanged: (selected) { 90 Plan currentPlan = plan; 91 int planIndex = planNotifier.value.indexWhere(92 (p) => p.name == currentPlan.name, 93); 94 planNotifier.value = List<Plan>.from(planNotifier.value) 95 ..[planIndex] = Plan(96 name: currentPlan.name, 97 tasks: List<Task>.from(currentPlan.tasks) 98 ..[index] = Task(99 description: task.description, 100 complete: selected ?? false, 101), // Task 102); // Plan 103 }, 104), // Checkbox 105 title: TextFormField(106 initialValue: task.description, 107 onChanged: (text) { 108 Plan currentPlan = plan; 109 int planIndex = planNotifier.value.indexWhere(110 (p) => p.name == currentPlan.name, 111); 112 planNotifier.value = List<Plan>.from(planNotifier.value) 113 ..[planIndex] = Plan(114 name: currentPlan.name, 115 tasks: List<Task>.from(currentPlan.tasks) 116 ..[index] = Task(description: text, complete: task.complete), 117); // Plan 118 }, 119), // TextFormField 120); // ListTile 121 }</pre>
9	Langkah 9: Buat screen baru
	<p>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.</p> <pre>18 theme: ThemeData(primarySwatch: Colors.blue), 19 home: const PlanCreatorScreen(), 20), // MaterialApp</pre>



10	Langkah 10: Pindah ke class <code>_PlanCreatorScreenState</code>
	<p>Kita perlu tambahkan variabel <code>TextEditingController</code> sehingga bisa membuat <code>TextField</code> sederhana untuk menambah Plan baru. Jangan lupa tambahkan <code>dispose</code> ketika widget unmounted seperti kode berikut.</p> <pre>13 class _PlanCreatorScreenState extends State<PlanCreatorScreen> { 14 final textController = TextEditingController(); 15 16 @override 17 void dispose() { 18 textController.dispose(); 19 super.dispose(); 20 }</pre>
11	Langkah 11: Pindah ke method build
	<p>Letakkan method Widget build berikut di atas void dispose. Gantilah 'Namaku' dengan nama panggilan Anda.</p> <pre>22 @override 23 Widget build(BuildContext context) { 24 return Scaffold(25 // ganti 'Namaku' dengan nama panggilan Anda 26 appBar: AppBar(title: const Text('Master Plans Saria')), 27 body: Column(28 children: [29 _buildListCreator(), 30 Expanded(child: _buildMasterPlans()), 31], 32), // Column 33); // Scaffold 34 }</pre>
12	Langkah 12: Buat widget <code>_buildListCreator</code>
	<p>Buatlah widget berikut setelah widget build.</p> <pre>36 Widget _buildListCreator() { 37 return Padding(38 padding: const EdgeInsets.all(20.0), 39 child: Material(40 color: Theme.of(context).cardColor, 41 elevation: 10, 42 child: TextField(43 controller: textController, 44 decoration: const InputDecoration(45 labelText: 'Add a plan', 46 contentPadding: EdgeInsets.all(20), 47), // InputDecoration 48 onEditingComplete: addPlan, 49), // TextField 50), // Material 51); // Padding 52 }</pre>



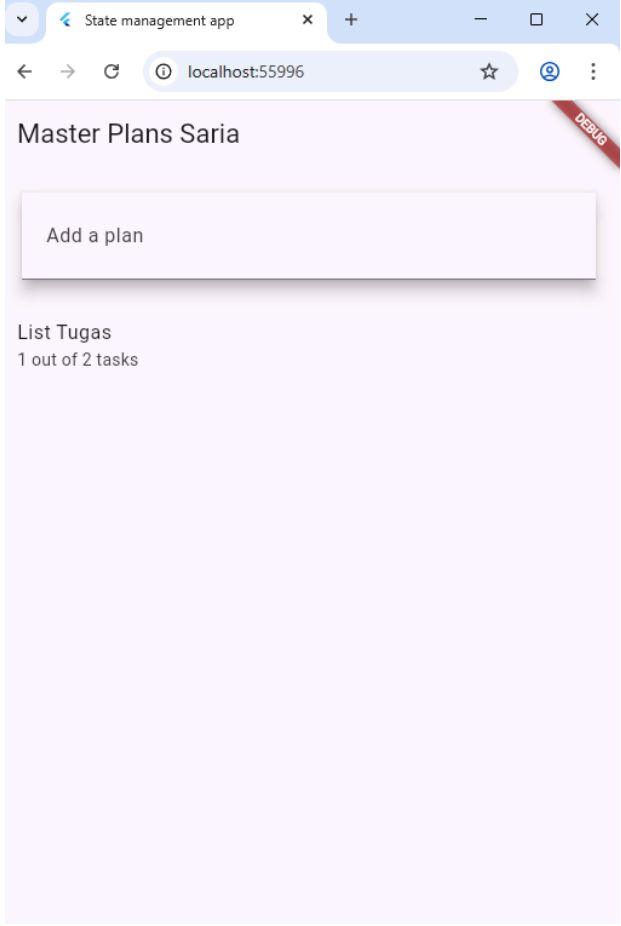
13	Langkah 13: Buat void addPlan()
	<p>Tambahkan method berikut untuk menerima inputan dari user berupa text plan.</p> <pre>55 void addPlan() { 56 final text = textController.text; 57 if (text.isEmpty) { 58 return; 59 } 60 final plan = Plan(name: text, tasks: []); 61 ValueNotifier<List<Plan>> planNotifier = PlanProvider.of(context); 62 planNotifier.value = List<Plan>.from(planNotifier.value)..add(plan); 63 textController.clear(); 64 FocusScope.of(context).requestFocus(FocusNode()); 65 setState(() {}); 66 }</pre>
14	Langkah 14: Buat widget _buildMasterPlans()
	<p>Tambahkan widget seperti kode berikut.</p> <pre>69 Widget _buildMasterPlans() { 70 ValueNotifier<List<Plan>> planNotifier = PlanProvider.of(context); 71 List<Plan> plans = planNotifier.value; 72 73 if (plans.isEmpty) { 74 return Column(75 mainAxisAlignment: MainAxisAlignment.center, 76 children: <Widget>[77 const Icon(Icons.note, size: 100, color: Colors.grey), 78 Text(79 'Anda belum memiliki rencana apapun.', 80 style: Theme.of(context).textTheme.headlineSmall, 81), // Text 82], // <Widget>[] 83); // Column 84 } 85 return ListView.builder(86 itemCount: plans.length, 87 itemBuilder: (context, index) { 88 final plan = plans[index]; 89 return ListTile(90 title: Text(plan.name), 91 subtitle: Text(plan.completenessMessage), 92 onTap: () { 93 Navigator.of(94 context, 95).push(MaterialPageRoute(builder: (_) => PlanScreen(plan: plan))); 96 }, 97); // ListTile 98 }, 99); // ListView.builder 100 }</pre>

Tugas Praktikum 3: State di Multiple Screens



Langkah	Jawaban/Deskripsi
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	<p>Berdasarkan Praktikum 3 yang telah Anda lakukan, jelaskan maksud dari gambar diagram berikut ini!</p> <p>Jawab: Gambar tersebut menunjukkan bagaimana widget tree berubah saat aplikasi berpindah dari halaman pembuat rencana (PlanCreatorScreen) ke halaman detail rencana (PlanScreen) menggunakan Navigator.push(). Setiap halaman memiliki struktur widget-nya sendiri yang disusun dalam MaterialApp utama, namun mereka terhubung melalui PlanProvider dan sistem navigasi Flutter.</p>
3	<p>Lakukan capture hasil dari Langkah 14 berupa GIF, kemudian jelaskan apa yang telah Anda buat!</p> <p>Jawab:</p>



	 <p>Pada Langkah 14, aplikasi berhasil menampilkan daftar plan yang dikelola oleh PlanProvider dan mendukung navigasi antar halaman menggunakan Navigator.push(). Ini menunjukkan implementasi state management sederhana dengan konsep data binding dan navigasi dinamis di Flutter.</p>
4	Kumpulkan laporan praktikum Anda berupa link commit atau repository GitHub ke dosen yang telah disepakati !