

# Flutter & Dart Complete Guide

Prepared by Eng. Yousef Salah Kassem

DAFL 3  
State Management

## Lecture 2



Chapter  
**02**

## Table Of Content

- Introduction
- What is Family in Riverpod ?
- Auto Dispose
- Project



# Introduction

## Why Riverpod?

- **Modern and robust alternative to Provider**
- Compile-time safety & IDE support
- Works with **dart** (not just Flutter widgets)
- Supports modular architecture

## Key Concepts in Riverpod

- **Provider** – a basic way to expose state
- **Consumer/WidgetRef** – reads provider values
- **StateProvider** – simple mutable state
- **FutureProvider / StreamProvider** – async data
- **StateNotifier / StateNotifierProvider** – advanced state control
- **AsyncValue** – handles loading, error, data

# Family in Riverpod

## What is **family** in Riverpod?

### ✓ Definition:

The **.family** modifier allows you to **create parameterized providers** — i.e., providers that **accept arguments**.

## Use Case:

- Fetch a user by `userId`
- Fetch product details by `productId`
- Access dynamic resources

```
final userProvider = FutureProvider.family<User, String>((ref, userId) async {  
  return await fetchUserById(userId);  
});
```

```
class UserWidget extends ConsumerWidget {  
  final String userId;
```

```
  const UserWidget({required this.userId});
```

```
  @override
```

```
  Widget build(BuildContext context, WidgetRef ref) {  
    final userAsync = ref.watch(userProvider(userId)); // 📌 Pass param here
```

```
    return userAsync.when(  
      data: (user) => Text('Name: ${user.name}'),  
      loading: () => CircularProgressIndicator(),  
      error: (e, _) => Text('Error: $e'),  
    );  
  }  
}
```

# Auto Dispose

## What is **autoDispose** in Riverpod?

### ✓ Definition:

**autoDispose** is a modifier that **automatically disposes the provider's state** when it's no longer being used.

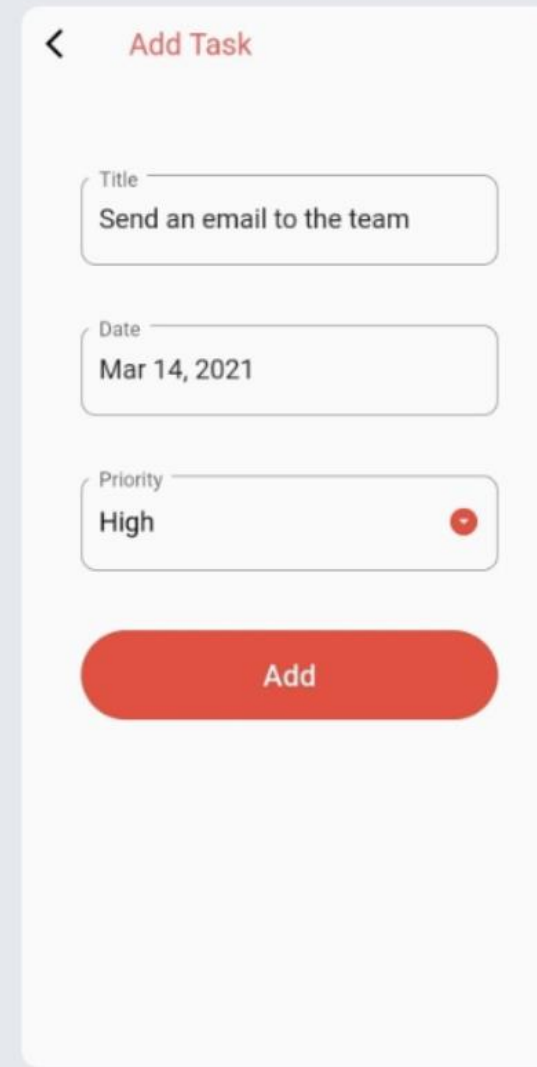
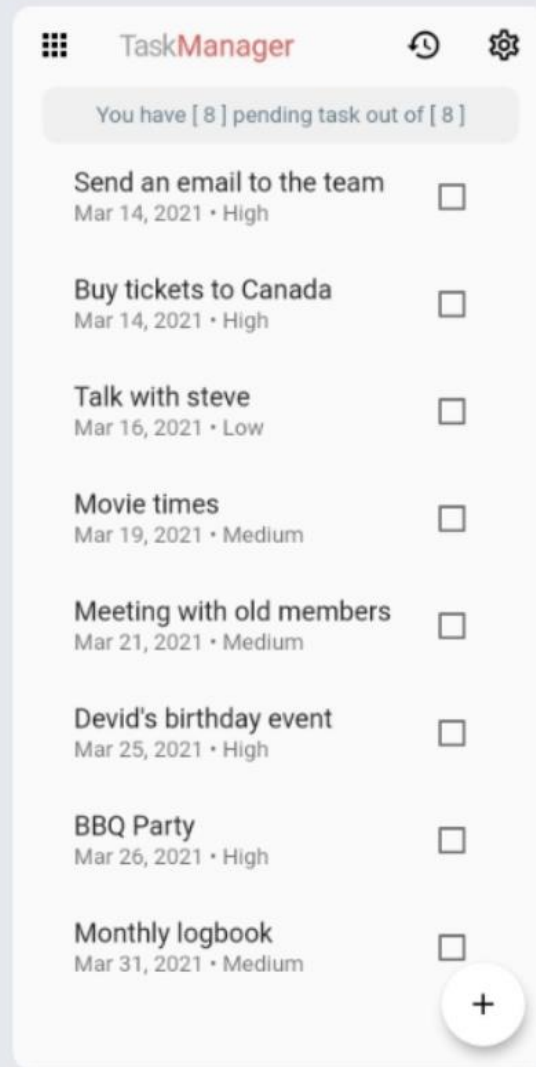
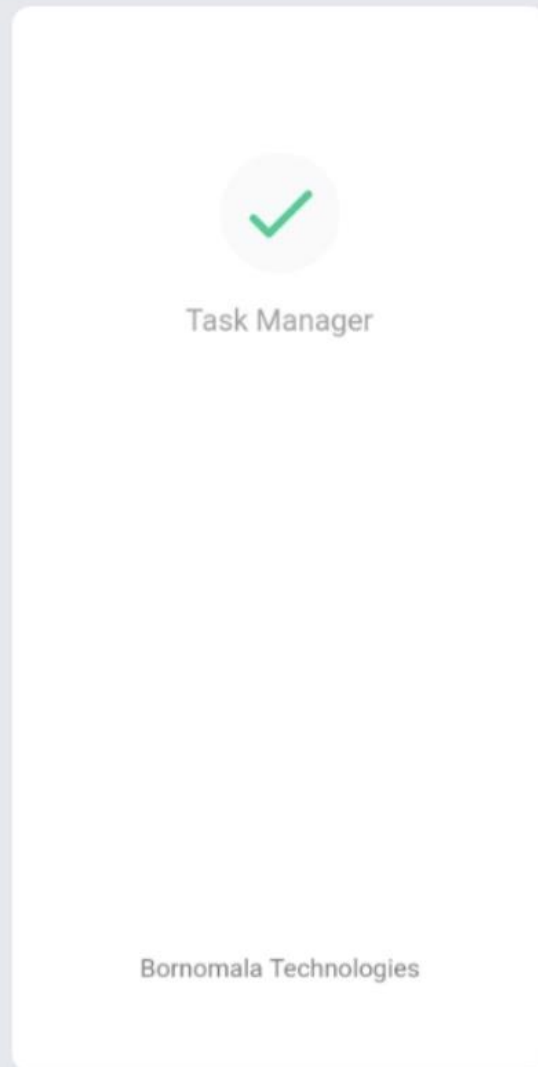
### **Use Case:**

- Free up memory when leaving a screen
- Stop unnecessary streams or timers
- Avoid stale data in unused screens

```
final timerProvider = StreamProvider.autoDispose<int>((ref) {  
  return Stream.periodic(Duration(seconds: 1), (count) => count);  
});
```

```
class TimerWidget extends ConsumerWidget {  
  @override  
  Widget build(BuildContext context, WidgetRef ref) {  
    final countAsync = ref.watch(timerProvider);  
  
    return countAsync.when(  
      data: (count) => Text('Seconds: $count'),  
      loading: () => CircularProgressIndicator(),  
      error: (e, _) => Text('Error'),  
    );  
  }  
}
```

# Project



## < Update Task

Title

BBQ Party with friends

Date

Mar 26, 2021

Priority

High



Update

Delete

## < History

You have completed [ 1 ] tasks

Movie times

Mar 19, 2021 • Medium



Task reassigned

```
final todoProvider = StateNotifierProvider<TodoProvider, List<Task>>(  
    (ref) => TodoProvider(),  
); // StateNotifierProvider  
  
class TodoProvider extends StateNotifier<List<Task>> {  
    TodoProvider() : super([]);  
  
    final List<Task> _allTasks = [];  
  
    final titleController = TextEditingController();  
    final dateController = TextEditingController();  
    TaskPriority? priority;  
  
    void initTask(String? id) {  
        if (id != null) {  
            getTaskById(id);  
        } else {  
            clearForm();  
        }  
    }  
  
    void addTask() {  
        final newTask = Task(  
            id: DateTime.now().microsecondsSinceEpoch.toString(),  
            title: titleController.text,  
            date: dateController.text,  
            priority: priority ?? TaskPriority.low,  
        ); // Task  
        _allTasks.add(newTask);  
        state = [..._allTasks];  
    }  
}
```

```
void updateTask(String id, Task updatedTask) {
    final index = _allTasks.indexWhere((task) => task.id == id);
    if (index != -1) {
        _allTasks[index] = updatedTask;
        state = [..._allTasks];
    }
}

void deleteTask(String id) {
    _allTasks.removeWhere((task) => task.id == id);
    state = [..._allTasks];
}

List<Task> get tasks => state;

Task? getTaskById(String? id) {
    if (id == null) return null;
    final task = _allTasks.firstWhere((task) => task.id == id);
    if (task.id.isNotEmpty) {
        titleController.text = task.title;
        dateController.text = task.date;
        priority = task.priority;
    }
    return task;
}

void searchByTitle(String title) {
    if (title.trim().isEmpty) {
        state = [..._allTasks];
        return;
    }

    state = _allTasks
        .where(
            (task) => task.title.toLowerCase().contains(title.toLowerCase()),
        )
        .toList();
}
```

```
void setPriority(TaskPriority newPriority) {  
    priority = newPriority;  
}  
  
void clearList() {  
    _allTasks.clear();  
    state = [];  
}  
  
void clearForm() {  
    titleController.clear();  
    dateController.clear();  
    priority = null;  
}  
}
```

```
class Task {
    const Task({
        required this.id,
        required this.title,
        required this.date,
        required this.priority,
    });

    final String id;
    final String title;
    final String date;
    final TaskPriority priority;

    Task copyWith({
        String? title,
        String? date,
        TaskPriority? priority,
    }) {
        return Task(
            id: id,
            title: title ?? this.title,
            date: date ?? this.date,
            priority: priority ?? this.priority,
        );
    }

    @override
    bool operator ==(Object other) =>
        identical(this, other) ||
        other is Task && runtimeType == other.runtimeType && id == other.id &&
            title == other.title && date == other.date &&
            priority == other.priority;

    @override
    int get hashCode =>
        id.hashCode ^ title.hashCode ^ date.hashCode ^ priority.hashCode;
}
```

```
final routerProvider = Provider((ref) {  
  return GoRouter(  
    initialLocation: Routes.home.path,  
    routes: [  
      GoRoute(  
        path: Routes.home.path,  
        builder: (context, state) {  
          return const HomeScreen();  
        },  
      ), // GoRoute  
      GoRoute(  
        path: Routes.addTask.path,  
        builder: (context, state) {  
          final id = state.extra as String?;  
          return AddTaskScreen(id: id);  
        },  
      ), // GoRoute  
    ],  
  ); // GoRouter  
}); // Provider
```

```
enum TaskPriority {  
    low('Low'),  
    medium('Medium'),  
    high('High');  
  
    const TaskPriority(this.name);  
  
    static TaskPriority fromString(String name) {  
        return TaskPriority.values.firstWhere(  
            (priority) => priority.name.toLowerCase() == name.toLowerCase(),  
            orElse: () => TaskPriority.low,  
        );  
    }  
  
    final String name;  
}💡
```

```
class AddTaskScreen extends ConsumerWidget {  
  const AddTaskScreen({super.key, this.id});  
  
  final String? id;  
  
  @override  
  Widget build(BuildContext context, WidgetRef ref) {  
    ref.read(todoProvider.notifier).initTask(id);  
  
    final controller = ref.read(todoProvider.notifier);  
    final isEditMode = id != null;  
    final items =  
      TaskPriority.values.map((e) {  
        return DropdownMenuItem(value: e.name, child: Text(e.name));  
      }).toList();  
    return Scaffold(  
      appBar: AppBar(  
        leading: IconButton(  
          icon: const Icon(Icons.arrow_back, color: Colors.black),  
          onPressed: () {  
            context.pop();  
          },  
        ), // IconButton  
        centerTitle: false,  
        title: Text(  
          isEditMode ? 'Edit Task' : 'Add Task',  
          style: const TextStyle(  
            color: Colors.redAccent,  
            fontSize: 24,  
            fontWeight: FontWeight.w500,  
          ), // TextStyle  
        ), // Text  
      ), // AppBar
```

```
body: Padding(
  padding: const EdgeInsets.all(16.0),
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      SizedBox(height: 60),
      TextFormField(...), // TextFormField
      const SizedBox(height: 40),
      TextFormField(...), // TextFormField
      const SizedBox(height: 40),
      DropdownButtonFormField<String>(...), // DropdownButtonFormField
      const SizedBox(height: 40),
      ElevatedButton(
        style: ElevatedButton.styleFrom(...),
        onPressed: () {
          if (isEditMode) {
            controller.updateTask(
              id!,
              Task(
                id: id!,
                title: controller.titleController.text,
                date: controller.dateController.text,
                priority: controller.priority ?? TaskPriority.low,
              ), // Task
            );
          } else {
            controller.addTask();
          }
          controller.clearForm();
          context.pop();
        },
      ),
      child: SizedBox(
        width: MediaQuery.sizeOf(context).width,
        child: Center(
          child: Text(isEditMode ? 'Update Task' : 'Add Task'),
        ), // Center
      ), // SizedBox
    ), // ElevatedButton
  ), // ElevatedButton
  const SizedBox(height: 20),
```

```
        child: SizedBox(
          width: MediaQuery.sizeOf(context).width,
          child: Center(
            child: Text(isEditMode ? 'Update Task' : 'Add Task'),
          ), // Center
        ), // SizedBox
      ), // ElevatedButton
    const SizedBox(height: 20),
    if (isEditMode)
      ElevatedButton(
        style: ElevatedButton.styleFrom(
          backgroundColor: Colors.redAccent,
          foregroundColor: Colors.white,
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(8.0),
          ), // RoundedRectangleBorder
        ),
        onPressed: () {
          controller.deleteTask(id!);
          controller.clearForm();
          context.pop();
        },
        child: SizedBox(
          width: MediaQuery.sizeOf(context).width,
          child: const Center(child: Text('Delete Task')),
        ), // SizedBox
      ), // ElevatedButton
  ],
), // Column
), // Padding
); // Scaffold
}
}
```

```
class HomeScreen extends ConsumerWidget {  
  const HomeScreen({super.key});  
  
  @override  
  Widget build(BuildContext context, WidgetRef ref) {  
    final listOfTasks = ref.watch(todoProvider);  
    final controller = ref.read(todoProvider.notifier);  
    return Scaffold(  
      appBar: AppBar(  
        leading: IconButton(...), // IconButton  
        centerTitle: false,  
        title: RichText(...), // RichText  
        actions: [...],  
      ), // AppBar  
      body: Column(  
        children: [  
          Padding(  
            padding: const EdgeInsets.symmetric(horizontal: 16.0),  
            child: TextFormField(  
              onChanged: (value){  
                controller.searchByTitle(value);  
              },  
              decoration: InputDecoration(  
                hintText: 'Search by title',  
                border: const OutlineInputBorder(),  
                prefixIcon: const Icon(Icons.search),  
              ), // InputDecoration  
            ), // TextFormField  
          ), // Padding  
          Expanded(...), // Expanded  
        ],  
      ), // Column  
      floatingActionButton: FloatingActionButton(...), // FloatingActionButton  
    ); // Scaffold  
  }  
}
```

```
Expanded(  
  child: ListView.builder(  
    itemCount: listOfTasks.length,  
    itemBuilder: (context, index) {  
      final task = listOfTasks[index];  
      return ListTile(  
        title: Text(task.title),  
        subtitle: Text('${task.date} - ${task.priority.name}'),  
        onTap: () {  
          context.push(Routes.addTask.path, extra: task.id);  
        },  
      ); // ListTile  
    },  
  ), // ListView.builder  
, // Expanded
```

```
floatingActionButton: FloatingActionButton(  
  backgroundColor: Colors.white,  
  shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(100)),  
  onPressed: () {  
    context.push(Routes.addTask.path);  
  },  
  child: const Icon(Icons.add, color: Colors.black),  
), // FloatingActionButton
```

```
void main() {  
  runApp(ProviderScope(child: const MyApp()));  
}  
  
class MyApp extends ConsumerWidget {  
  const MyApp({super.key});  
  
  @override  
  Widget build(BuildContext context, WidgetRef ref) {  
    final appRouter = ref.watch(routerProvider);  
    return MaterialApp.router(  
      title: 'Flutter Demo',  
      theme: ThemeData(  
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),  
      ), // ThemeData  
      routerConfig: appRouter,  
    ); // MaterialApp.router  
  }  
}
```

```
enum Routes {  
    home('/'),  
    addTask('/addTask');  
  
    const Routes(this.path);  
  
    final String path;  
}💡
```

# **What is Next ?**



**Thank You**