Of course. Here is a complete guide to building a to-do list app in Flutter using the Riverpod package for state management and shared\_preferences for data persistence.

This app will allow you to:

* Add new tasks.
* Mark tasks as complete.
* Delete tasks.
* Filter tasks by "All," "Active," and "Completed."
* Save tasks to the device's local storage, so they persist between app launches.

**1. Project Setup & Dependencies**

First, create a new Flutter project and add the necessary packages to your pubspec.yaml file. We'll use riverpod\_generator for modern, boilerplate-free provider definitions.

Bash

flutter create todo\_app

cd todo\_app

Now, add the following to your pubspec.yaml:

YAML

dependencies:

flutter:

sdk: flutter

flutter\_riverpod: ^2.5.1

shared\_preferences: ^2.2.3

uuid: ^4.4.0

dev\_dependencies:

flutter\_test:

sdk: flutter

build\_runner: ^2.4.9

riverpod\_generator: ^2.4.0

riverpod\_lint: ^2.3.10

custom\_lint: ^0.6.4

After adding the dependencies, install them by running: flutter pub get

**2. Data Model (Todo)**

Let's define the model for a single to-do item. This class will include methods to convert it to and from JSON for persistence.

Create a file lib/todo.dart:

Dart

import 'package:uuid/uuid.dart';

const \_uuid = Uuid();

class Todo {

final String id;

final String description;

final bool completed;

Todo({

required this.description,

this.completed = false,

String? id,

}) : id = id ?? \_uuid.v4();

// Method to create a new instance with updated values (immutability)

Todo copyWith({String? description, bool? completed}) {

return Todo(

id: id,

description: description ?? this.description,

completed: completed ?? this.completed,

);

}

// Serialization: Convert a Todo object into a Map

Map<String, dynamic> toJson() {

return {

'id': id,

'description': description,

'completed': completed,

};

}

// Deserialization: Create a Todo object from a Map

factory Todo.fromJson(Map<String, dynamic> map) {

return Todo(

id: map['id'] as String,

description: map['description'] as String,

completed: map['completed'] as bool,

);

}

}

**3. State Management with Riverpod (AsyncNotifier)**

This is the core of our app. We will create an AsyncNotifier which handles loading, managing, and saving the list of todos. Using riverpod\_generator, this becomes incredibly clean.

Create a file lib/todos\_provider.dart:

Dart

import 'dart:convert';

import 'package:riverpod\_annotation/riverpod\_annotation.dart';

import 'package:shared\_preferences/shared\_preferences.dart';

import 'todo.dart';

part 'todos\_provider.g.dart'; // This file will be generated by build\_runner

const \_kTodosKey = 'todos'; // Key for shared\_preferences

@riverpod

class TodosNotifier extends \_$TodosNotifier {

// The build method is where we initialize the state.

// It's async, so we can perform async operations like reading from storage.

@override

Future<List<Todo>> build() async {

return \_loadTodos();

}

// --- Persistence Logic ---

Future<List<Todo>> \_loadTodos() async {

final prefs = await SharedPreferences.getInstance();

final String? todosJson = prefs.getString(\_kTodosKey);

if (todosJson == null) {

return []; // Return an empty list if no data is stored

}

final List<dynamic> decodedJson = jsonDecode(todosJson);

return decodedJson.map((json) => Todo.fromJson(json)).toList();

}

Future<void> \_saveTodos(List<Todo> todos) async {

final prefs = await SharedPreferences.getInstance();

final String encodedJson = jsonEncode(todos.map((todo) => todo.toJson()).toList());

await prefs.setString(\_kTodosKey, encodedJson);

}

// --- Public Methods to Modify State ---

Future<void> addTodo(String description) async {

// Get the current state (list of todos)

final previousState = await future;

final newTodo = Todo(description: description);

// Update the state with the new list. Riverpod will handle UI updates.

state = AsyncData([...previousState, newTodo]);

// Save the new state to persistence

await \_saveTodos(state.value!);

}

Future<void> removeTodo(String todoId) async {

final previousState = await future;

state = AsyncData(previousState.where((todo) => todo.id != todoId).toList());

await \_saveTodos(state.value!);

}

Future<void> toggle(String todoId) async {

final previousState = await future;

state = AsyncData([

for (final todo in previousState)

if (todo.id == todoId)

todo.copyWith(completed: !todo.completed)

else

todo,

]);

await \_saveTodos(state.value!);

}

}

Now, run the build runner to generate the todos\_provider.g.dart file. In your terminal, run:

flutter pub run build\_runner watch --delete-conflicting-outputs

**4. UI - The Home Screen**

Let's build the UI. We'll have a main screen that consumes our providers to display and interact with the data.

Create a file lib/home\_screen.dart:

Dart

import 'package:flutter/material.dart';

import 'package:flutter\_riverpod/flutter\_riverpod.dart';

import 'todos\_provider.dart';

// Enum for filtering options

enum TodoFilter { all, active, completed }

// A provider to hold the current filter state

final todoFilterProvider = StateProvider<TodoFilter>((ref) => TodoFilter.all);

// A provider that computes the filtered list of todos

final filteredTodosProvider = Provider.autoDispose<AsyncValue<List<Todo>>>((ref) {

final filter = ref.watch(todoFilterProvider);

final todos = ref.watch(todosNotifierProvider);

return todos.whenData((todoList) {

switch (filter) {

case TodoFilter.completed:

return todoList.where((todo) => todo.completed).toList();

case TodoFilter.active:

return todoList.where((todo) => !todo.completed).toList();

case TodoFilter.all:

default:

return todoList;

}

});

});

class HomeScreen extends ConsumerWidget {

const HomeScreen({super.key});

@override

Widget build(BuildContext context, WidgetRef ref) {

// Watch our filtered list provider

final filteredTodos = ref.watch(filteredTodosProvider);

return Scaffold(

appBar: AppBar(

title: const Text('Riverpod Todos'),

actions: [

// Filter Dropdown

PopupMenuButton<TodoFilter>(

initialValue: ref.watch(todoFilterProvider),

onSelected: (filter) => ref.read(todoFilterProvider.notifier).state = filter,

itemBuilder: (BuildContext context) => <PopupMenuEntry<TodoFilter>>[

const PopupMenuItem(value: TodoFilter.all, child: Text('All')),

const PopupMenuItem(value: TodoFilter.active, child: Text('Active')),

const PopupMenuItem(value: TodoFilter.completed, child: Text('Completed')),

],

icon: const Icon(Icons.filter\_list),

),

],

),

body: filteredTodos.when(

data: (todos) => ListView.builder(

itemCount: todos.length,

itemBuilder: (context, index) {

final todo = todos[index];

return Dismissible(

key: ValueKey(todo.id),

direction: DismissDirection.endToStart,

onDismissed: (\_) {

ref.read(todosNotifierProvider.notifier).removeTodo(todo.id);

},

background: Container(

color: Colors.red,

alignment: Alignment.centerRight,

padding: const EdgeInsets.symmetric(horizontal: 20),

child: const Icon(Icons.delete, color: Colors.white),

),

child: ListTile(

title: Text(

todo.description,

style: TextStyle(

decoration: todo.completed ? TextDecoration.lineThrough : TextDecoration.none,

),

),

leading: Checkbox(

value: todo.completed,

onChanged: (bool? value) {

ref.read(todosNotifierProvider.notifier).toggle(todo.id);

},

),

),

);

},

),

loading: () => const Center(child: CircularProgressIndicator()),

error: (err, stack) => Center(child: Text('Error: $err')),

),

floatingActionButton: FloatingActionButton(

onPressed: () => \_showAddTaskDialog(context, ref),

child: const Icon(Icons.add),

),

);

}

void \_showAddTaskDialog(BuildContext context, WidgetRef ref) {

final TextEditingController controller = TextEditingController();

showDialog(

context: context,

builder: (context) {

return AlertDialog(

title: const Text('Add a new task'),

content: TextField(

controller: controller,

autofocus: true,

decoration: const InputDecoration(hintText: 'Task description'),

),

actions: [

TextButton(

onPressed: () => Navigator.of(context).pop(),

child: const Text('Cancel'),

),

TextButton(

onPressed: () {

if (controller.text.isNotEmpty) {

ref.read(todosNotifierProvider.notifier).addTodo(controller.text);

Navigator.of(context).pop();

}

},

child: const Text('Add'),

),

],

);

},

);

}

}

**5. Main Application File**

Finally, let's tie everything together in lib/main.dart. The only change here is wrapping our app in a ProviderScope to make the providers available throughout the widget tree.

Dart

import 'package:flutter/material.dart';

import 'package:flutter\_riverpod/flutter\_riverpod.dart';

import 'home\_screen.dart';

void main() {

runApp(

// ProviderScope stores the state of our providers.

const ProviderScope(

child: MyApp(),

),

);

}

class MyApp extends StatelessWidget {

const MyApp({super.key});

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Todo App',

theme: ThemeData(

primarySwatch: Colors.blue,

useMaterial3: true,

),

home: const HomeScreen(),

);

}

}

You can now run your application! You have a fully functional and persistent to-do list app built with clean, modern Riverpod state management.