### Tutorial: Criando uma Aplicação de Lista de Tarefas em Flutter

#### Objetivo

Neste tutorial, você aprenderá a criar uma aplicação simples de lista de tarefas onde o usuário pode adicionar, remover e marcar tarefas como concluídas.

#### Pré-requisitos

* Flutter instalado em sua máquina.
* Conhecimentos básicos de Dart e Flutter.

### Passo 1: Criar um Novo Projeto Flutter

1. Abra seu terminal ou prompt de comando.
2. Crie um novo projeto Flutter com o seguinte comando:

flutter create todo\_app

1. Navegue até a pasta do projeto:

cd todo\_app

### Passo 2: Adicionar Dependência do Provider

1. Abra o arquivo pubspec.yaml.
2. Adicione a dependência provider na seção de dependências:

dependencies:

flutter:

sdk: flutter

provider: ^6.0.0

1. Salve o arquivo e execute o comando para instalar as dependências:

flutter pub get

### Passo 3: Criar o Modelo de Tarefa

1. Crie uma nova pasta chamada model dentro da pasta lib.
2. Dentro da pasta model, crie um arquivo chamado task\_model.dart.
3. Adicione o seguinte código ao task\_model.dart:
4. import 'package:flutter/material.dart'; *// Importa o pacote de Material Design do Flutter.*
5. class Task {
6. final String name; *// Nome da tarefa, que não pode ser alterado após a criação.*
7. bool isDone; *// Estado da tarefa: se está concluída ou não.*
8. *// Construtor da classe Task. 'required' indica que o parâmetro 'name' é obrigatório.*
9. Task({required this.name, this.isDone = false});
10. }
11. *// Classe que gerencia a lista de tarefas, estendendo ChangeNotifier para notificar mudanças.*
12. class TaskModel with ChangeNotifier {
13. List<Task> \_tasks = []; *// Lista privada que armazena as tarefas.*
14. *// Getter para acessar a lista de tarefas.*
15. List<Task> get tasks => \_tasks;
16. *// Método para adicionar uma nova tarefa à lista.*
17. void addTask(String *taskName*) {
18. \_tasks.add(Task(*name*: *taskName*)); *// Cria uma nova tarefa e a adiciona à lista.*
19. notifyListeners(); *// Notifica ouvintes de que o estado mudou.*
20. }
21. *// Método para alternar o estado de uma tarefa (concluída ou não).*
22. void toggleTask(int *index*) {
23. \_tasks[*index*].isDone = !\_tasks[*index*].isDone; *// Inverte o estado da tarefa.*
24. notifyListeners(); *// Notifica ouvintes da mudança de estado.*
25. }
26. *// Método para remover uma tarefa da lista.*
27. void removeTask(int *index*) {
28. \_tasks.removeAt(*index*); *// Remove a tarefa no índice especificado.*
29. notifyListeners(); *// Notifica ouvintes de que o estado mudou.*
30. }
31. }

### Passo 4: Criar a Tela Principal

1. Crie um novo arquivo chamado todo\_screen.dart dentro da pasta lib.
2. Adicione o seguinte código ao todo\_screen.dart:
3. import 'package:flutter/material.dart'; *// Importa o pacote de Material Design do Flutter.*
4. import 'package:provider/provider.dart'; *// Importa o pacote Provider para gerenciamento de estado.*
5. import 'model/task\_model.dart'; *// Importa o modelo de tarefas que contém a lógica de gerenciamento.*
6. class TodoScreen extends StatelessWidget {
7. *// Controlador para o campo de texto onde o usuário insere novas tarefas.*
8. final TextEditingController \_controller = TextEditingController();
9. @override
10. Widget build(BuildContext *context*) {
11. *// Obtém a instância do TaskModel usando o Provider.*
12. final taskModel = Provider.of<TaskModel>(*context*);
13. return Scaffold(
14. *appBar*: AppBar(
15. *title*: Text('Lista de Tarefas'), *// Título da barra de navegação.*
16. ),
17. *body*: Column(
18. *children*: [
19. *// Campo de entrada para novas tarefas.*
20. Padding(
21. *padding*: const EdgeInsets.all(16.0), *// Adiciona espaçamento ao redor do campo.*
22. *child*: TextField(
23. *controller*: \_controller, *// Associa o controlador ao campo de texto.*
24. *decoration*: InputDecoration(
25. *labelText*: 'Nova Tarefa', *// Texto que aparece acima do campo.*
26. *suffixIcon*: IconButton(
27. *icon*: Icon(Icons.add), *// Ícone de adicionar.*
28. *onPressed*: () {
29. *// Ao pressionar o botão, verifica se o campo não está vazio.*
30. if (\_controller.text.isNotEmpty) {
31. taskModel.addTask(\_controller.text); *// Adiciona a nova tarefa ao modelo.*
32. \_controller.clear(); *// Limpa o campo de texto.*
33. }
34. },
35. ),
36. ),
37. ),
38. ),
39. Expanded(
40. *child*: ListView.builder(
41. *// Constrói uma lista de tarefas dinamicamente.*
42. *itemCount*: taskModel.tasks.length, *// Número de tarefas na lista.*
43. *itemBuilder*: (*context*, *index*) {
44. final task = taskModel.tasks[*index*]; *// Obtém a tarefa atual.*
45. return ListTile(
46. *title*: Text(
47. task.name, *// Nome da tarefa a ser exibido.*
48. *style*: TextStyle(
49. *decoration*: task.isDone ? TextDecoration.lineThrough : null, *// Aplica um riscado se a tarefa estiver concluída.*
50. ),
51. ),
52. *trailing*: IconButton(
53. *icon*: Icon(Icons.delete), *// Ícone de deletar.*
54. *onPressed*: () => taskModel.removeTask(*index*), *// Remove a tarefa ao pressionar o botão.*
55. ),
56. *onTap*: () => taskModel.toggleTask(*index*), *// Alterna o estado da tarefa ao clicar nela.*
57. );
58. },
59. ),
60. ),
61. ],
62. ),
63. );
64. }
65. }

### Passo 5: Configurar o Widget Principal

1. Abra o arquivo main.dart.
2. Modifique o código para configurar o Provider e exibir a tela principal:
3. import 'package:flutter/material.dart'; *// Importa o pacote de Material Design do Flutter.*
4. import 'package:provider/provider.dart'; *// Importa o pacote Provider para gerenciamento de estado.*
5. import 'model/task\_model.dart'; *// Importa o modelo de tarefas.*
6. import 'todo\_screen.dart'; *// Importa a tela principal da lista de tarefas.*
7. void main() {
8. runApp(
9. ChangeNotifierProvider(
10. *create*: (*context*) => TaskModel(), *// Cria uma instância de TaskModel como um ChangeNotifier.*
11. *child*: MyApp(), *// O widget MyApp é o filho do ChangeNotifierProvider.*
12. ),
13. );
14. }
15. *// Classe principal da aplicação.*
16. class MyApp extends StatelessWidget {
17. @override
18. Widget build(BuildContext *context*) {
19. return MaterialApp(
20. *title*: 'To-Do List App', *// Define o título da aplicação.*
21. *home*: TodoScreen(), *// Define a tela inicial como TodoScreen.*
22. );
23. }
24. }

### Passo 6: Executar a Aplicação

1. Conecte um dispositivo ou inicie um emulador.
2. No terminal, execute o comando:

flutter run

1. A aplicação deve abrir, permitindo que você adicione, remova e marque tarefas como concluídas.