Flutter State Management: Provider for Beginners

Subtitle: Learn how to manage app state efficiently using the Provider package in Flutter.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Managing state is crucial for **dynamic apps**. Provider is a simple and powerful **state management solution** in Flutter that separates UI from logic. This guide covers the basics and practical usage.

Step 1: Add Provider Dependency

```
dependencies:
  provider: ^6.1.6
```

Run:

```
flutter pub get
```

Import in Dart:

```
import 'package:provider/provider.dart';
```

Step 2: Create a State Class

```
class Counter with ChangeNotifier {
  int _count = 0;

  int get count => _count;

  void increment() {
    _count++;
    notifyListeners();
  }
}
```

Exercise: Create a `Counter` class with increment functionality.

Step 3: Wrap App with ChangeNotifierProvider

```
void main() {
  runApp(
    ChangeNotifierProvider(
      create: (_) => Counter(),
      child: MyApp(),
    ),
  );
}
```

Exercise: Wrap your app to provide state globally.

Step 4: Access State in Widgets

```
Consumer<Counter>(
  builder: (context, counter, child) {
    return Text('Count: ${counter.count}');
  },
)
```

Exercise: Display the current count on screen using `Consumer`.

Step 5: Update State from UI

```
ElevatedButton(
  onPressed: () {
    context.read<Counter>().increment();
  },
  child: Text('Increment'),
)
```

Exercise: Add a button to increment the counter and update the UI.

Step 6: Mini Project

Build a **Flutter Counter App with Provider**:

- Global counter state using Provider

- Global counter state using Provider
- Display current value with `Consumer`
- Increment button updates state instantly
- Bonus: Add multiple widgets consuming the same state

Key Takeaways

- Provider separates **logic from UI**, making apps clean
- `ChangeNotifier` notifies listeners when state changes
- `Consumer` and `context.read()` access and update state
- Proper state management ensures **scalable and maintainable apps**

Skipping state management leads to **messy and unmaintainable code**, especially for complex apps.

Visit **haas.dev** for more Flutter guides, tutorials, and complete project resources.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app
