

Flutter Counter App - State Management Report

Konsep Utama yang Diimplementasikan

Aplikasi ini dibuat untuk memahami dua pendekatan state management di Flutter:

1. Local State Management - untuk data yang cuma diperlukan dalam satu widget
2. Global State Management - untuk data yang perlu diakses dari berbagai widget

Local State Implementation

Cara Kerja Local State



Kenapa pakai ini?

- Sederhana dan cepat
- Data tidak perlu dibagi ke widget lain
- Cocok untuk form input, toggle button, dll

Kelebihan:

- Performa bagus karena cuma rebuild widget ini aja
- Kode mudah dipahami

- Tidak butuh package tambahan

🌐 Global State Implementation

Setup Provider Pattern



```
1 MultiProvider(
2   providers: [
3     ChangeNotifierProvider(create: (_) => GlobalState()),
4     ChangeNotifierProvider(create: (_) => AppState()),
5   ],
6   child: MaterialApp(...),
7 )
```

Fungsinya: Bikin GlobalState bisa diakses dari semua widget di aplikasi.

GlobalState Class

```
1 class GlobalState extends ChangeNotifier {  
2     List<CounterModel> _counters = [];  
3  
4     void addCounter() {  
5         _counters.add(newCounter);  
6         notifyListeners(); // Kasih tau semua widget ada perubahan  
7     }  
8  
9     void incrementCounter(String id) {  
10        // Cari counter berdasarkan ID  
11        final index = _counters.indexWhere((counter) => counter.id == id);  
12        if (index != -1) {  
13            _counters[index] = _counters[index].copyWith(  
14                value: _counters[index].value + 1,  
15            );  
16            notifyListeners(); // Update semua widget  
17        }  
18    }  
19 }
```

Kenapa pakai ChangeNotifier?

- Bisa notify semua widget yang listen
- Pattern yang direkomendasikan Flutter team
- Mudah dipahami dan diimplementasi

Cara Akses Data Global

```
1 Consumer<GlobalState>(  
2     builder: (context, globalState, child) {  
3         // Setiap kali notifyListeners() dipanggil, ini akan rebuild  
4         return Text('Total Counters: ${globalState.counterCount}');  
5     },  
6 )
```

- Consumer vs Provider.of:
 - Consumer lebih efisien karena cuma rebuild bagian yang perlu

- Provider.of rebuild seluruh widget

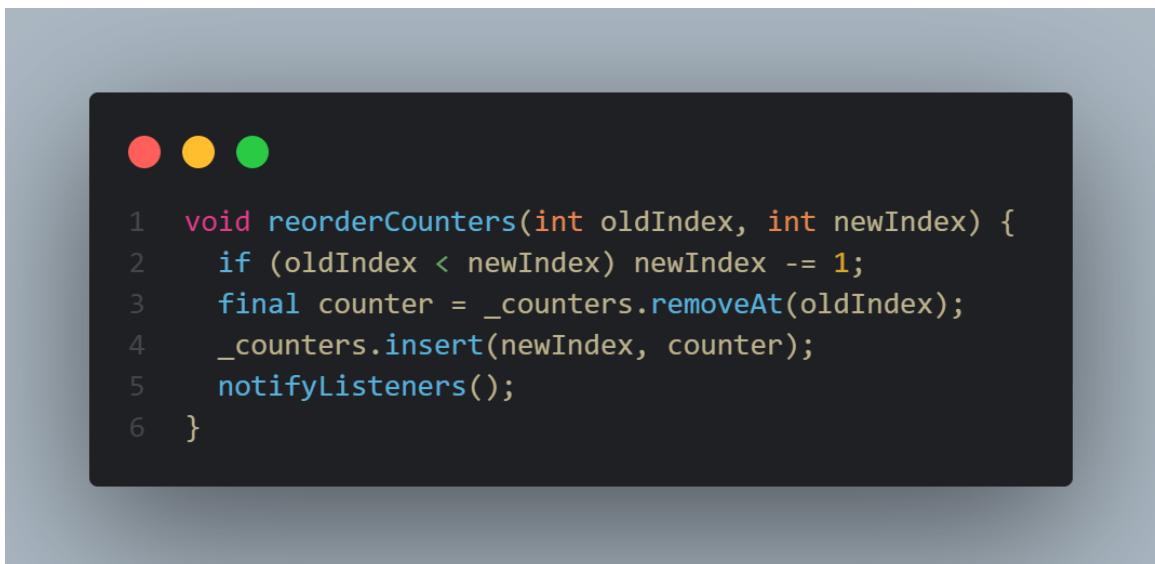
🌐 Advanced Features yang Diimplementasi

1. Drag and Drop Functionality



```
1  ReorderableListView.builder(
2    itemCount: globalState.counters.length,
3    onReorder: (oldIndex, newIndex) {
4      globalState.reorderCounters(oldIndex, newIndex);
5    },
6    itemBuilder: (context, index) {
7      return GlobalCounterItem(key: ValueKey(counter.id), counter: counter);
8    },
9  )
```

Implementasi di GlobalState:



```
1  void reorderCounters(int oldIndex, int newIndex) {
2    if (oldIndex < newIndex) newIndex -= 1;
3    final counter = _counters.removeAt(oldIndex);
4    _counters.insert(newIndex, counter);
5    notifyListeners();
6  }
```

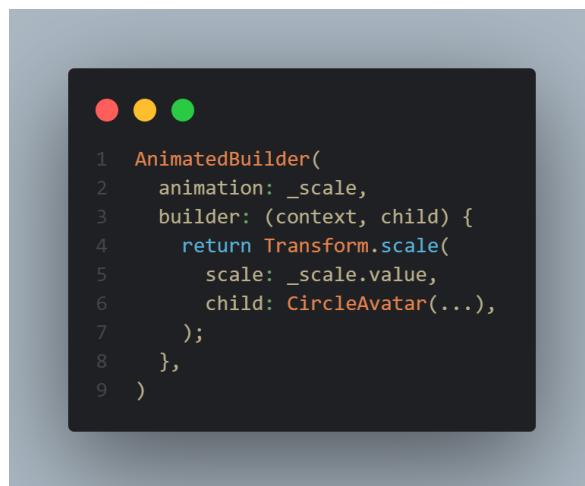
Kenapa pakai ini?

- User experience lebih baik
- Flutter udah nyediain widget ReorderableListView
- Cuma perlu handle callback onReorder

2. Smooth Animations



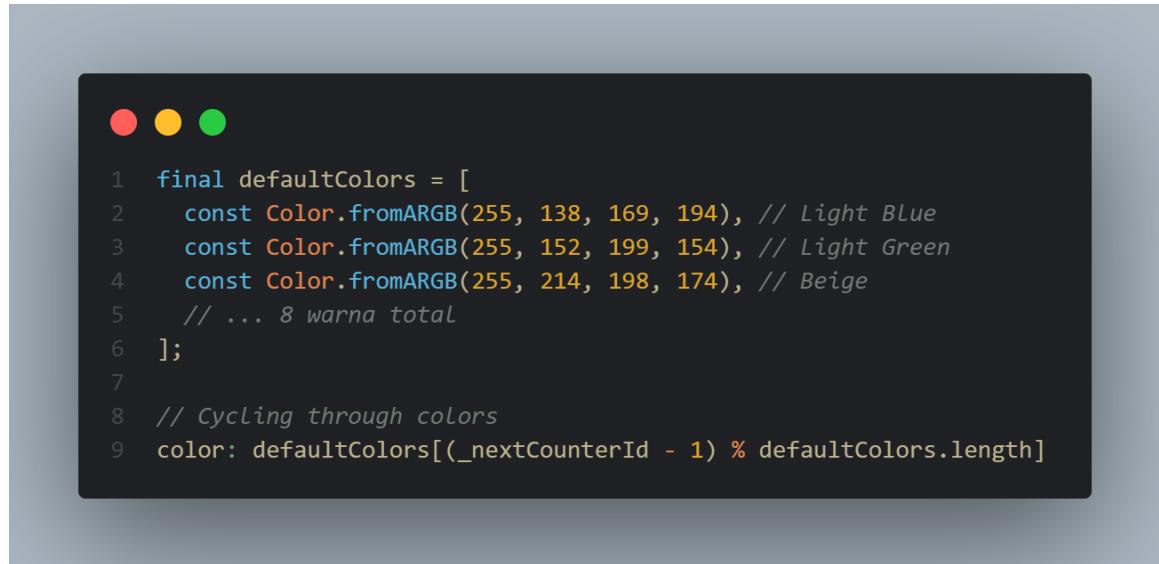
Penggunaan di UI:



Tips animasi:

- SingleTickerProviderStateMixin wajib untuk AnimationController
- Jangan lupa dispose controller di dispose()
- 200ms durasi yang enak untuk micro-interaction

3. Dynamic Color System



Edit Color Dialog:



4. Dark Mode Support



A screenshot of a mobile application interface. At the top, there are three colored dots (red, yellow, green) used as a navigation bar. Below this, the code for a class named AppState is displayed in a monospaced font. The code defines a private variable _isDarkMode, a public getter for isDarkMode, and a method toggleDarkMode that toggles the value of _isDarkMode and notifies listeners. It also shows how this state is used in the MaterialApp's theme configuration.

```
1  class AppState extends ChangeNotifier {
2      bool _isDarkMode = false;
3
4      bool get isDarkMode => _isDarkMode;
5
6      void toggleDarkMode() {
7          _isDarkMode = !_isDarkMode;
8          notifyListeners(); // Update theme di seluruh app
9      }
10 }
11
12 // Di MaterialApp
13 Consumer<AppState>(
14     builder: (context, appState, child) {
15         return MaterialApp(
16             theme: appState.isDarkMode ? ThemeData.dark() : ThemeData.light(),
17             home: CounterListScreen(),
18         );
19     },
20 )
```

Perbandingan Pendekatan

Feature	Local State	Global State
Scope	Satu widget	Seluruh aplikasi
Kompleksitas	Sederhana	Menengah
Data Sharing	Tidak bisa	Bisa
Performance	Sangat cepat	Cepat dengan Consumer
Maintenance	Mudah	Perlu struktur yang baik
Use Case	Form, toggle button	User data, app settings

💡 Lessons Learned

- Yang Mudah Dipahami:
 - Local state dengan setState() straightforward
 - Provider.of vs Consumer usage
 - Basic animation dengan AnimationController
- Yang Challenging:
 - Provider pattern awalnya bingung - butuh waktu untuk paham flow notifyListeners()
 - Animation timing - cari durasi yang pas buat user experience
 - State structure - gimana organize data biar maintainable
 - Performance optimization - kapan pakai Consumer vs Provider.of
- Best Practices yang Dipelajari:
 - Selalu dispose AnimationController
 - Pakai copyWith() untuk immutable updates
 - Consumer cuma untuk bagian yang perlu rebuild
 - Separasi concern antara UI dan business logic

Kesimpulan

Tugas ini memberikan pemahaman praktis tentang state management di Flutter. Local state cocok untuk data sederhana, sementara global state dengan Provider pattern sangat powerful untuk aplikasi yang lebih kompleks.

Provider pattern memang butuh learning curve, tapi setelah paham konsepnya, jadi lebih mudah untuk scale aplikasi. Advanced features seperti animations dan drag-drop membuat user experience jadi lebih engaging.

Key takeaway: Pilih state management sesuai kebutuhan. Jangan over-engineering untuk hal sederhana, tapi juga jangan under-engineering untuk aplikasi yang complex.