Flutter State Management Yaklaşımları

Betül Kanmaz

İçerik Bilgisi

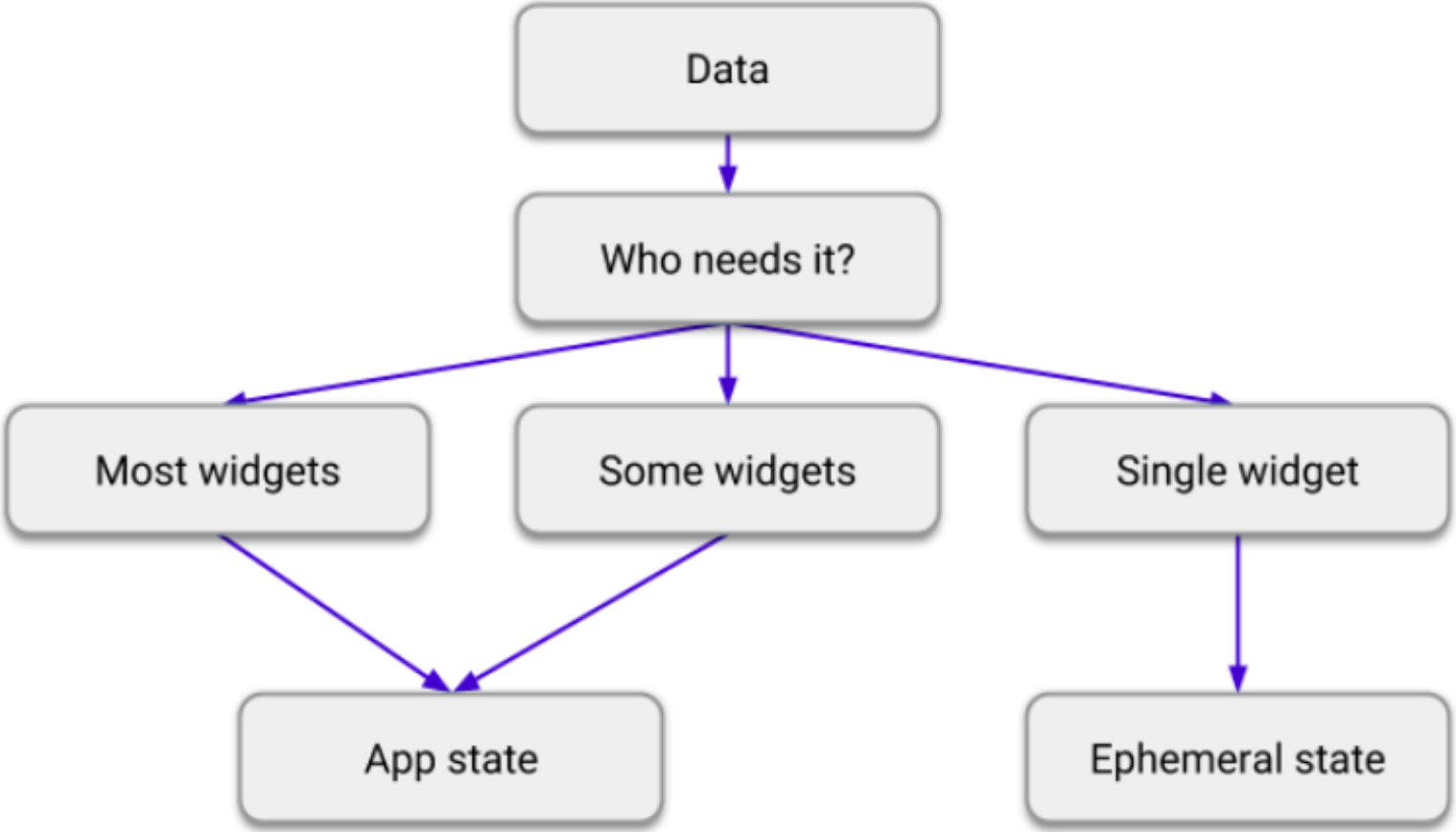
- State Management Nedir ?
- State Management Neden Kullanılır?
- Flutterda State Management Yaklaşımları Nelerdir?
  - Provider Paketi Nedir? Nasıl Kullanılır?

State?

★ Durum, widget oluşturulduğunda okunabilen ve uygulamanın ömrü boyunca değişebilen veya değiştirilebilen bilgilerdir.

★ Bilgisayar bilimlerinde, bir programın durumu, depolanan girdilerle ilgili durumu olarak tanımlanır.

"whatever data you need in order to rebuild your UI at any moment in time"


Data
Who needs it?
Most widgets
Some widgets
Single widget
App state
Ephemeral state

State Management Nedir?

★ Durum yönetimi, bir grafik kullanıcı arabiriminde metin alanları, OK düğmeleri, radyo düğmeleri vb. gibi bir veya daha fazla kullanıcı arabirimi denetiminin durumunun yönetimini ifade eder.

★ Bu kullanıcı arayüzü programlama tekniğinde, bir UI kontrolünün durumu, diğer UI kontrollerinin durumuna bağlıdır.

Durum Yönetim Kullanma Sebepleri

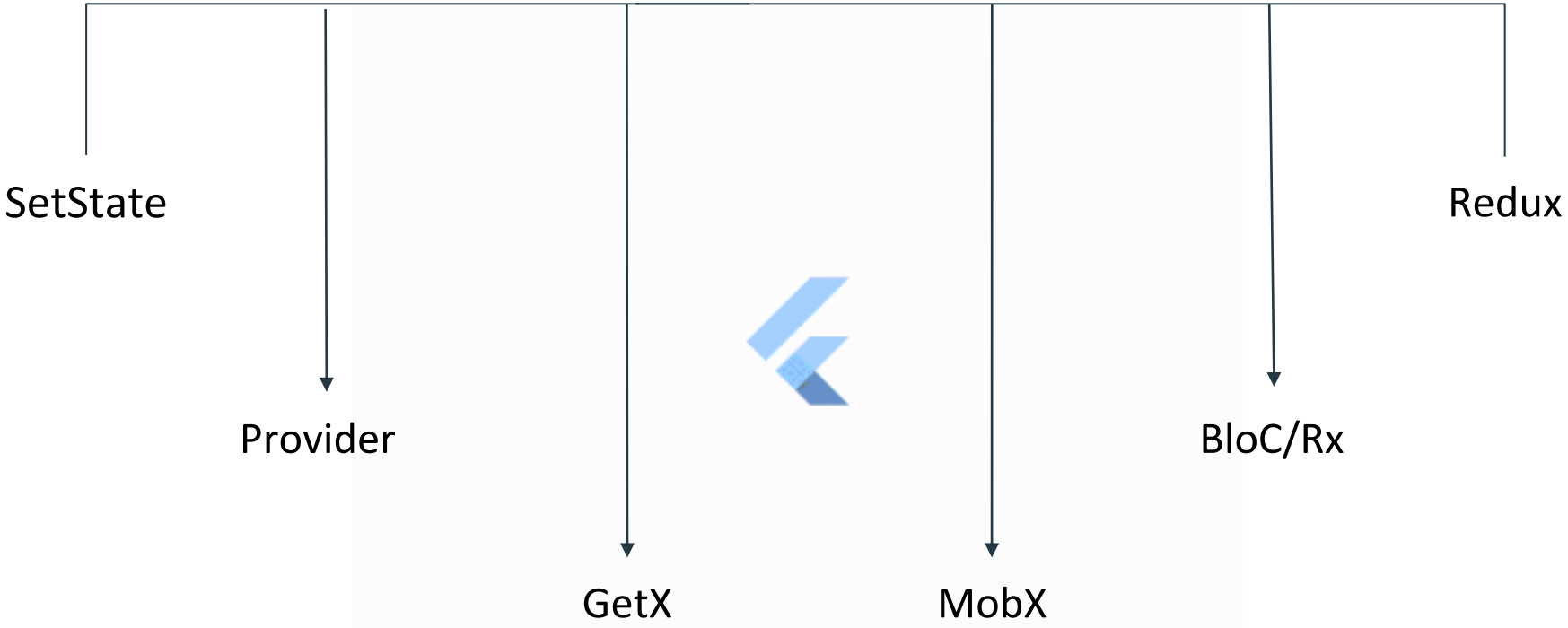
Neden Durum Yönetimleri?

Sistemin verimliliğini ve geliştiricilerin bu sistemi oluşturmaya özen göstermesini yansıtır, böylece her işlevsellik ve özellik sorunsuz ve hassas bir şekilde çalışır.

- ➔ Uygulama içindeki temel iş mantığını sunucular ve veritabanlarıyla hizalamaya ve bütünleştirmeye yardımcı olur. Uygun durum yönetimi olmadan kullanıcılar üzerindeki yük artacak ve bu kesinlikle bir uygulamanın etkinliğini azaltacaktır.

- ➔ Bir kod tabanını korumak daha kolay olacak ve kod kalitesini ve okunabilirliğini artıracaktır.

State Management Yaklaşımları



SetState

setState, yerel, widget'a özgü durum yönetimi için çok kullanışlıdır.

Redux

İş mantığını ve sunum mantığını ayıran tek yönlü bir veri akışı mimarisidir. Senkron durumlar için daha uygundur.

GetX

Flutter için ekstra hafif ve güçlü bir durum yönetimi çözümüdür. Rota yönetimi, durum yönetimi, navigasyon ve akıllı bağımlılık enjeksiyonunu daha pratik ve daha hızlı bir şekilde birleştirir.

MobX

Geliştiricilerin uygulama durumunu herhangi bir UI çerçevesi dışında yönetmelerini sağlar. Tüm değişiklikleri reaktif olarak algılayarak ve bunları gerekli kullanıcı arayüzüne yayarak durum yönetimini süper kolay hale getirmek için tasarlanmıştır.

BlocRx

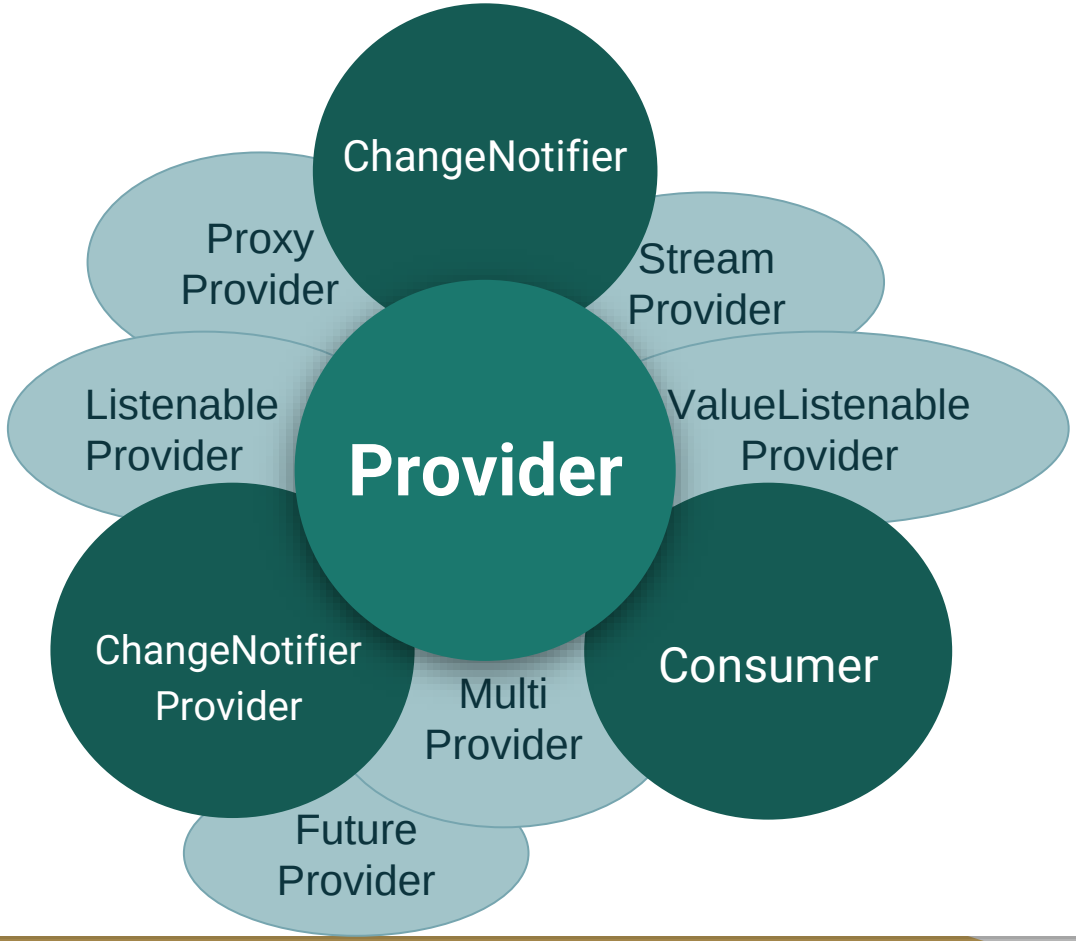
Bir iş mantığı bileşenidir. Geliştiricilerin verilere merkezi bir yerden erişmelerini sağlar. Blok / Rx, Blok ve indirgenmiş desenin birleşimidir. Redüktörler, mevcut durumu ve eylemleri argüman olarak alan ve durum biçiminde yeni bir sonuç döndüren işlevlerdir.

Provider

Sağlayıcı paketi, Devralınan Widget'ları daha yeniden kullanılabilir ve kullanımı kolay hale getirmek için bir sarıcıdır. Çok fazla kod gerektirmez ve sağlayıcının en temel şeklidir. Bir değer alır ve onu temsil eder, ancak sunduğu değerdeki değişikliklere dikkat etmez.

Sağlayıcı, kullanıcı oturum açma ayrıntılarımız gibi bazı değerleri altındaki widget'lar için kullanılabilir kılan bir widget'tır. Durumu widget ağacından geçirmek ve değişiklikler olduğunda kullanıcı arayüzünü yeniden oluşturmak için kullanışlı bir araç olarak düşünün.

Devralınan Widget(Inherited Widget), tüm verileri ağaçtan yukarıdan aşağıya doğru etkili bir şekilde geçiren temel sınıftır. Ağaçtaki herhangi bir Widget'ın Devralınan Widget'ın özelliklerine erişmesine izin verir.


ChangeNotifier
Proxy Provider
Stream Provider
Listenable Provider
Provider
ValueListenable Provider
ChangeNotifier Provider
Multi Provider
Consumer
Future Provider

ChangeNotifier

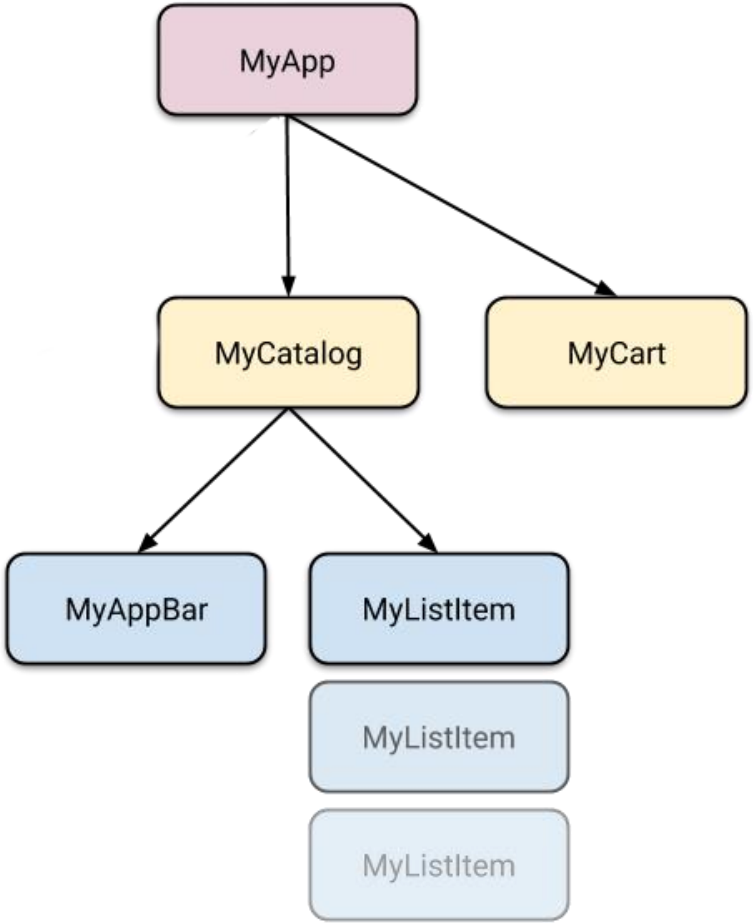
Dinleyicilerine değişiklik bildirimi sağlayan basit bir sınıftır. Az sayıda dinleyici için anlaşılması, uygulanması ve optimize edilmesi kolaydır. Dinleyicinin değişiklikler için bir model gözlemlemesi için kullanılır. Bu durumda, yalnızca dinleyicileri bilgilendirmek için notifylisteners() yöntemini kullanırız.

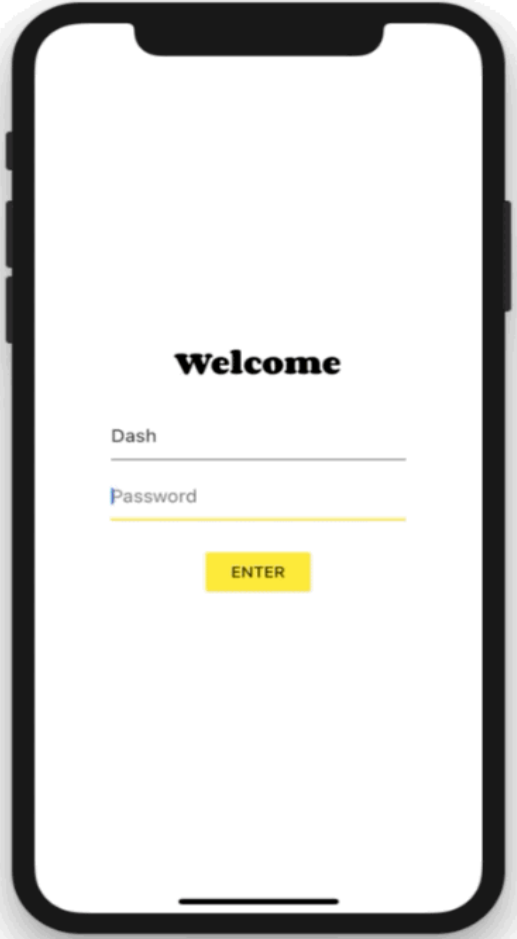
ChangeNotifierProvider

Change Notifier Provider, alt öğelerine bir Change Notifier örneği sağlayan widget'tır.

Consumer

Sağlayıcıyı yeni bir widget'ta çağırır ve yapı uygulamasını oluşturucuya devreder.


MyApp
MyCatalog
MyCart
MyAppBar
MyListItem
MyListItem
MyListItem


Welcome
Dash
Password
ENTER

```dart
class CartModel extends ChangeNotifier {
  /// Internal, private state of the cart.
  final List<Item> _items = [];

  /// An unmodifiable view of the items in the cart.
  UnmodifiableListView<Item> get items => UnmodifiableListView(_items);

  /// The current total price of all items (assuming all items cost $42).
  int get totalPrice => _items.length * 42;

  /// Adds [item] to cart. This and [removeAll] are the only ways to modify the
  /// cart from the outside.
  void add(Item item) {
    _items.add(item);
    // This call tells the widgets that are listening to this model to rebuild.
    notifyListeners();
  }

  /// Removes all items from the cart.
  void removeAll() {
    _items.clear();
    // This call tells the widgets that are listening to this model to rebuild.
    notifyListeners();
  }
}
```

```dart
void main() {
  runApp(
    ChangeNotifierProvider(
      create: (context) => CartModel(),
      child: const MyApp(),
    ),
  );
}
```

```dart
return Consumer<CartModel>(
  builder: (context, cart, child) {
    return Text('Total price: ${cart.totalPrice}');
  },
);
```