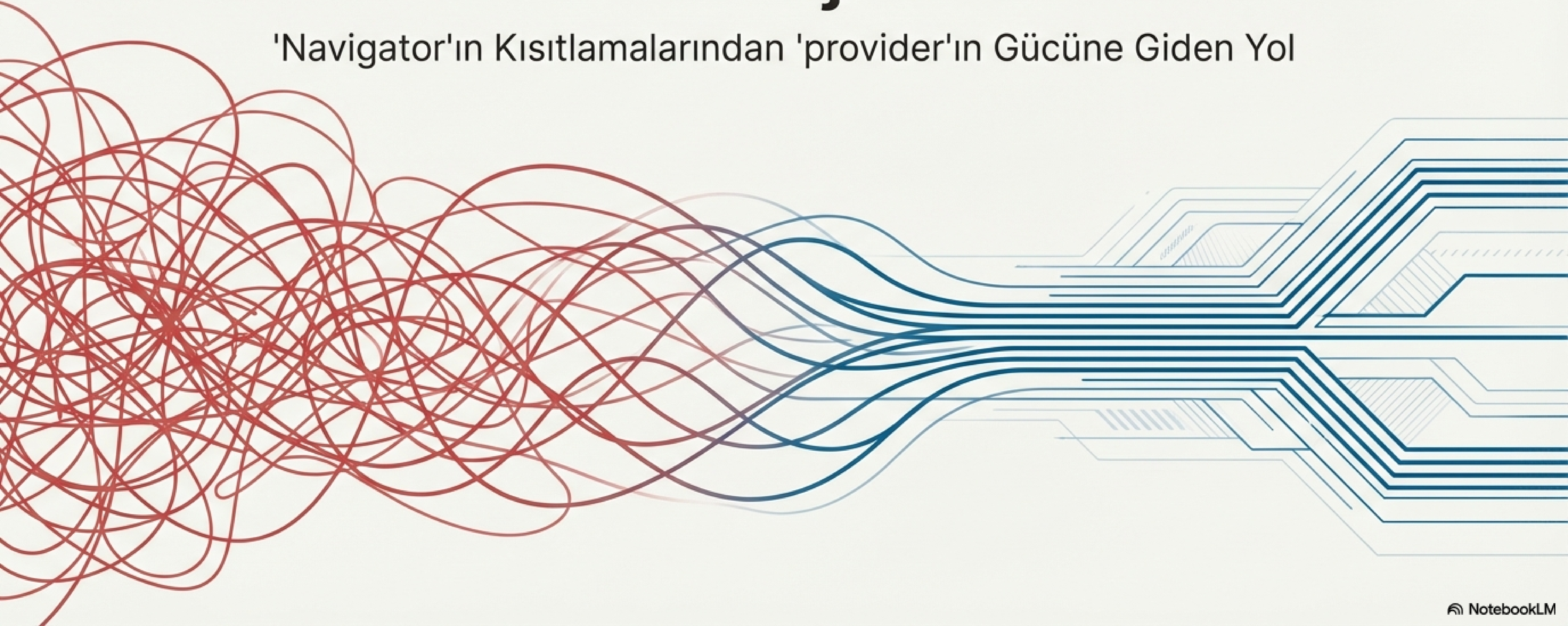
Flutter'da Veri Aktarımında Ustalaşmak

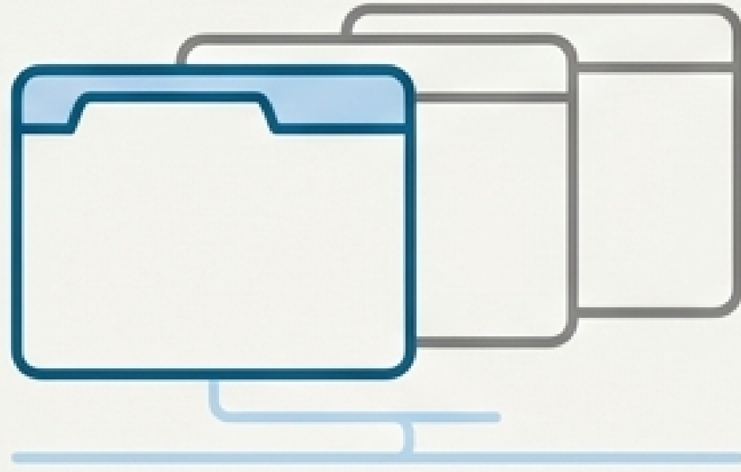
'Navigator'ın Kısıtlamalarından 'provider'ın Gücüne Giden Yol



NotebookLM

Her Flutter Geliştiricisinin Karşılaştığı Zorluk: Veri Paylaşımı

Uygulamaların farklı bölümlerindeki widget'ların veri paylaşımı çok yaygın bir ihtiyaçtır. En sık karşılaşılan senaryolar şunlardır:

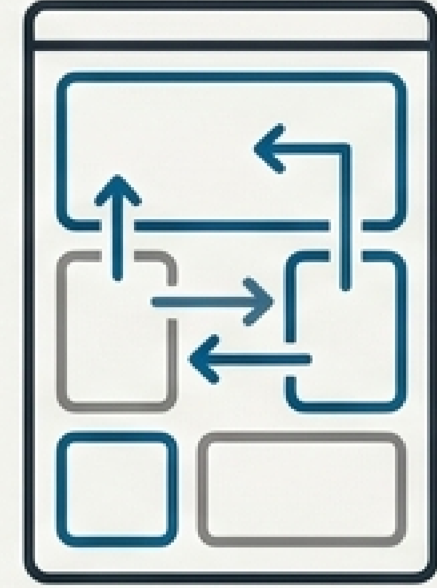


[ICON_TABS]

1. Tab'lar Arasında Veri Aktarımı: Bir sekmede yapılan bir işlemin sonucunu diğer bir sekmede göstermek.

[ICON_PAGES]

2. Sayfalar Arasında Veri Gönderme: Bir listeden seçilen bir öğenin detaylarını yeni bir sayfada göstermek.



[ICON_WIDGETS]

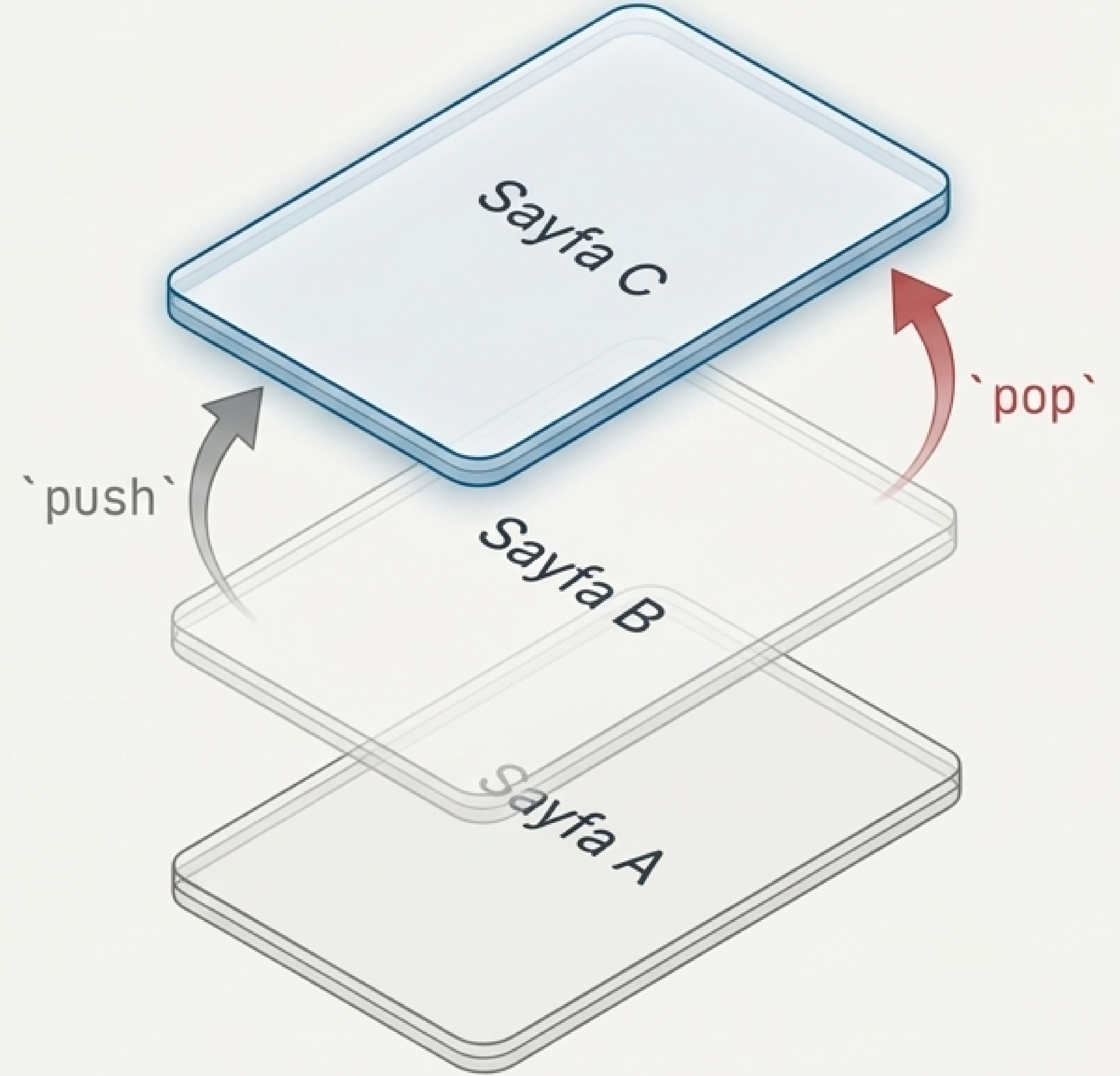
3. Aynı Sayfadaki Widget'lar Arası İletişim: Bir sayfadaki iki veya daha fazla widget'ın birbiriyle veri alışverişi yapması.

NotebookLM

İlk Akla Gelen Araç: `Navigator` Sınıfı

`Navigator` sınıfı, sayfalar (route'lar) arasında ileri ve geri hareket etmenizi sağlar. Gidilen sayfalar bir yığın (stack) mantığıyla üst üste eklenir.

- `push' metodu yığının en üstüne yeni bir rota ekler. Ekranda gördüğünüz sayfa, yığının en üstündeki sayfadır.
- `pop' metodu yığının en üstündeki rotayı kaldırır, böylece alttaki sayfa görünür hale gelir.



NotebookLM

Uygulamada `Navigator`: Bir "To-Do" Örneği

Bir yapılacaklar listesindeki öğeye tıklandığında, seçilen öğenin açıklamasını gösteren yeni bir sayfa açılır.

`TodosPage`

```
// 'TodosPage' içindeki _itemPressed metodu
void _itemPressed(BuildContext context, Todo item) =>
  Navigator.of(context)?.push(
    MaterialPageRoute(
      builder: (context) => InfoPage(todo: item),
    ),
  );
```

Navigasyon ve UI mantığı iç içe. Rota oluşturma (`MaterialPageRoute`) doğrudan widget içinde yapılıyor. Bu, 'plain route' (isimsiz rota) kullanımına bir örnektir.

`InfoPage`

```
// 'InfoPage' widget'ı veriyi constructor üzerinden alır
class InfoPage extends StatelessWidget {
  final Todo item;
  const InfoPage(this.item);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
       child: Text("${item.description}")
      ),
    );
```

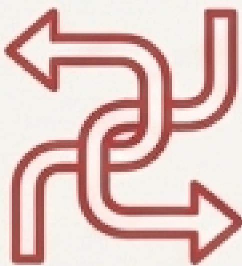
NotebookLM

Bu Yaklaşımın Bedeli: Karışıklık ve Kırılganlık

`Navigator` ile doğrudan veri aktarımı ilk başta kolay görünse de ciddi mimari sorunlara yol açar:

Tek Sorumluluk Prensibi (SRP) İhlali: Widget'lar artık navigasyon yönetimini de üstlenmek zorunda kalır. Navigasyon mantığı birden fazla dosyaya dağılır ve merkezden yönetilemez hale gelir.

Mantıksal Karmaşa: Navigasyon, iş ve UI mantığı birbirine karışır. Kodun okunabilirliği ve bakımı zorlaşır.



Zorlu Veri Akışı: Bir sayfadan geriye veri döndürmek için `push` metodunun döndürdüğü `Future`ı `async/await` ile beklemek gerekir. Çok sayıda `push` ve `pop` çağrısı arasındaki veri akışını takip etmek hızla kafa karıştırıcı hale gelir.

NotebookLM

Daha Temiz, Daha Güçlü Bir Yol: `provider` Paketi

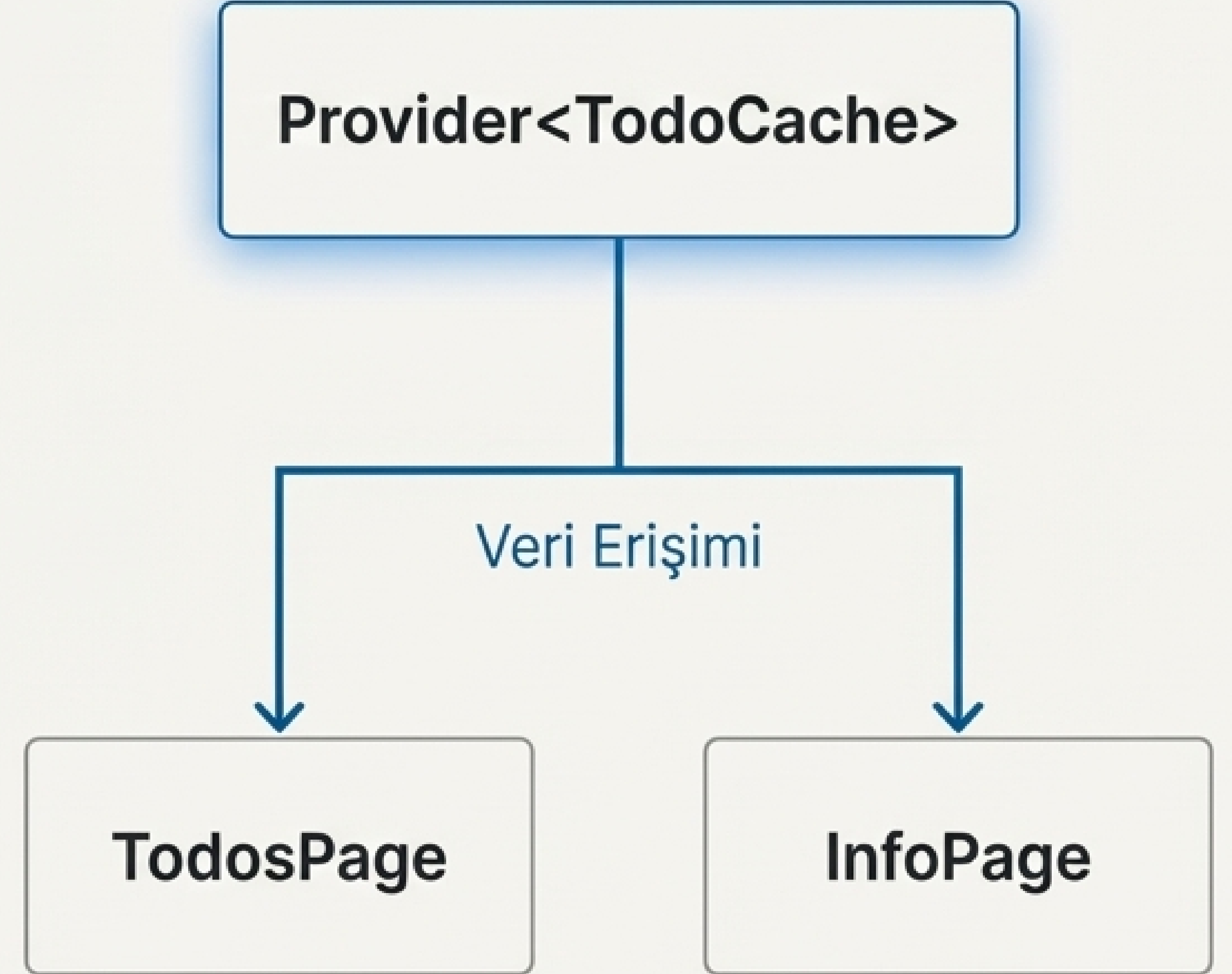
`provider`, bu veri paylaşım sorunlarını Dependency Injection (Bağımlılık Enjeksiyonu) prensibini kullanarak çözen bir durum yönetimi (state management) çözümüdür. `provider` kullanarak, navigasyonla ilgilenmek zorunda kalmadan widget'lar arasında nesneleri kolayca geçirebilirsiniz.

NotebookLM

Yeni Strateji: Sorumlulukları Ayırmak

`provider` yaklaşımı, endişeleri üç ana bileşene ayırır:

1. **Model** (`Todo`): Veri yapısını temsil eden saf bir sınıf. (Öncekiyle aynı)
2. **Veri Kaynağı** (`TodoCache`): Tüm iş mantığını ve 'to-do' listesini barındıran özel bir sınıf. Sayfalar bu 'cache'e `provider` aracılığıyla erişir. Veri taşıma sorumluluğu widget'lardan alınır.
3. **Sağlayıcı** (Provider): `TodoCache` nesnesini, ihtiyaç duyan tüm alt widget'ların erişebilmesi için widget ağacının üst katmanlarına 'sağlar' (enjekte eder).



NotebookLM

Kodun Evrimi: `provider` ile Yeniden Yazım

Kod, sorumlulukların ayrılması prensibine göre yeniden düzenlendi.

TodosPage

```dart
// Navigasyon artık temiz ve isme dayalı
void _open(BuildContext context) =>
  Navigator.of(context)?.pushNamed(RouteGenerator.infoPage);

// ListTile içindeki onTap metodu
onTap: () {
  cache.index = index; // Seçili öğenin index'ini cache'e yaz
  _open(context);
}
```

Temiz, isimlendirilmiş rota (`named route`). Navigasyon mantığı artık UI'dan bağımsız.

dart: JetBrains Mono Regular

InfoPage

```dart
// InfoPage artık constructor'a bağımlı değil
class InfoPage extends StatelessWidget {
  const InfoPage();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Consumer<TodoCache>( // Veriye Consumer ile erişim
          builder: (context, cache, _) {
            final item = cache.list[cache.index];
            return Text("${item.description}");
          }
        ),
      ),
    );
  }
}
```

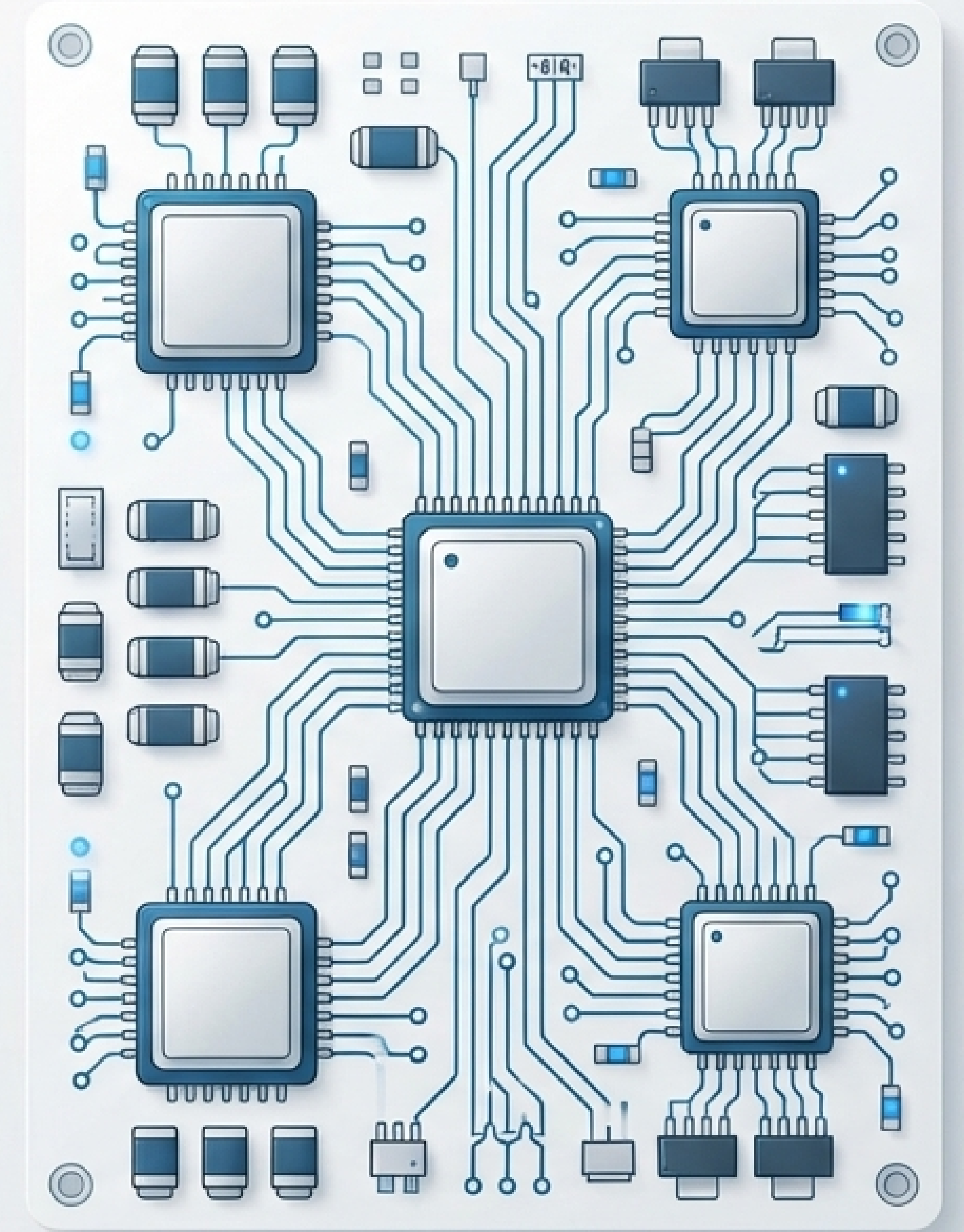
Veriye doğrudan ve reaktif erişim. `InfoPage` artık verinin nereden geldiğini bilmiyor, sadece istiyor.

NotebookLM

Sonuç: Bakımı Kolay ve Ölçeklenebilir Kod

`provider` yaklaşımının getirdiği temel kazanımlar:

- ✓ **Net Sorumluluk Ayrımı:** UI, iş mantığı ve navigasyon mantığı artık birbirinden tamamen ayrılmıştır. Her bileşenin tek bir görevi vardır.
- ✓ **Bağımsız Navigasyon:** Rota mantığı, `RouteGenerator` gibi merkezi bir sınıfta izole edilebilir. Widget'lar hangi sayfanın nasıl açılacağını bilmek zorunda değildir.
- ✓ **Basit ve Merkezi Veri Erişimi:** Karmaşık `push/`pop` akışlarına ve `Future` beklemeye gerek kalmaz. Veri, ihtiyaç duyulan yerde doğrudan `provider`'dan okunur.



NotebookLM

`Navigator` vs. `provider`: Net Bir Karşılaştırma

Kriter	`Navigator` ile Doğrudan Aktarım	`provider` ile Aktarım
Mantık Bağlılığı	**Yüksek** (UI ve Navigasyon iç içe) JetBrains Mono Regular	**Düşük** (Sorumluluklar ayrılmış)
Rota Tipi	Plain routes, isimsiz ve dağınık	İsimlendirilmiş rotalar (`named routes`), merkezi
Veri Akışı	Karmaşık ve zor takip edilir (`push`/`pop`)	Merkezi, basit ve reaktif
Ölçeklenebilirlik	**Düşük**. Proje büyüdükçe yönetilemez hale gelir.	**Yüksek**. Proje büyüse de düzenli kalır.

NotebookLM

Profesyonel Flutter Geliştirme İçin Altın Kural

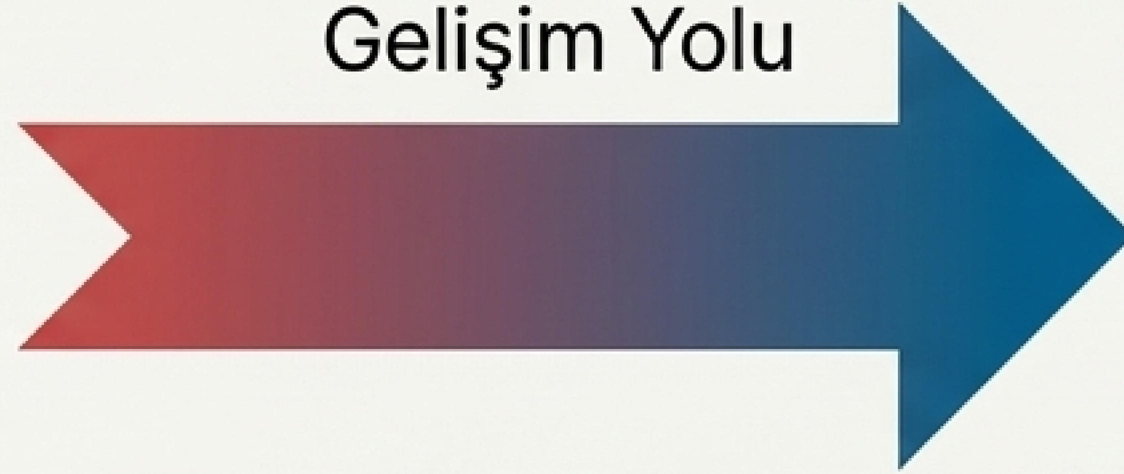
Sayfalar arası veri paylaşımı için `Navigator`'ın `push`/`pop` mekanizmalarını değil, `provider` gibi durum yönetimi (state management) çözümlerini tercih edin.

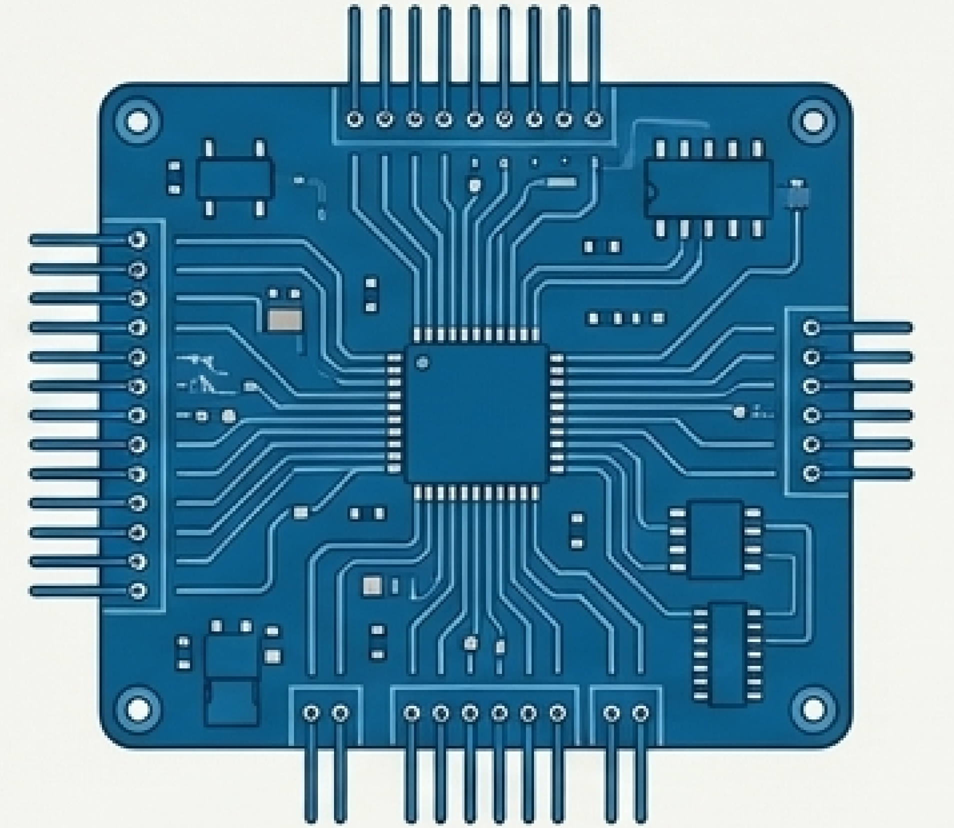
Bu, sadece bir tercih değil, temiz, sürdürülebilir ve ölçeklenebilir uygulamalar yazmak için bir standarttır.

NotebookLM

Karışık Kablolardan Düzenli Devrelere

Kodunuzdaki veri akışını basitleştirmek, sadece anlık bir sorunu çözmek değil, aynı zamanda gelecekteki gelişim için sağlam bir temel atmaktır. Doğru araçları kullanarak uygulamanızın mimarisini en başından itibaren temiz tutun.





NotebookLM