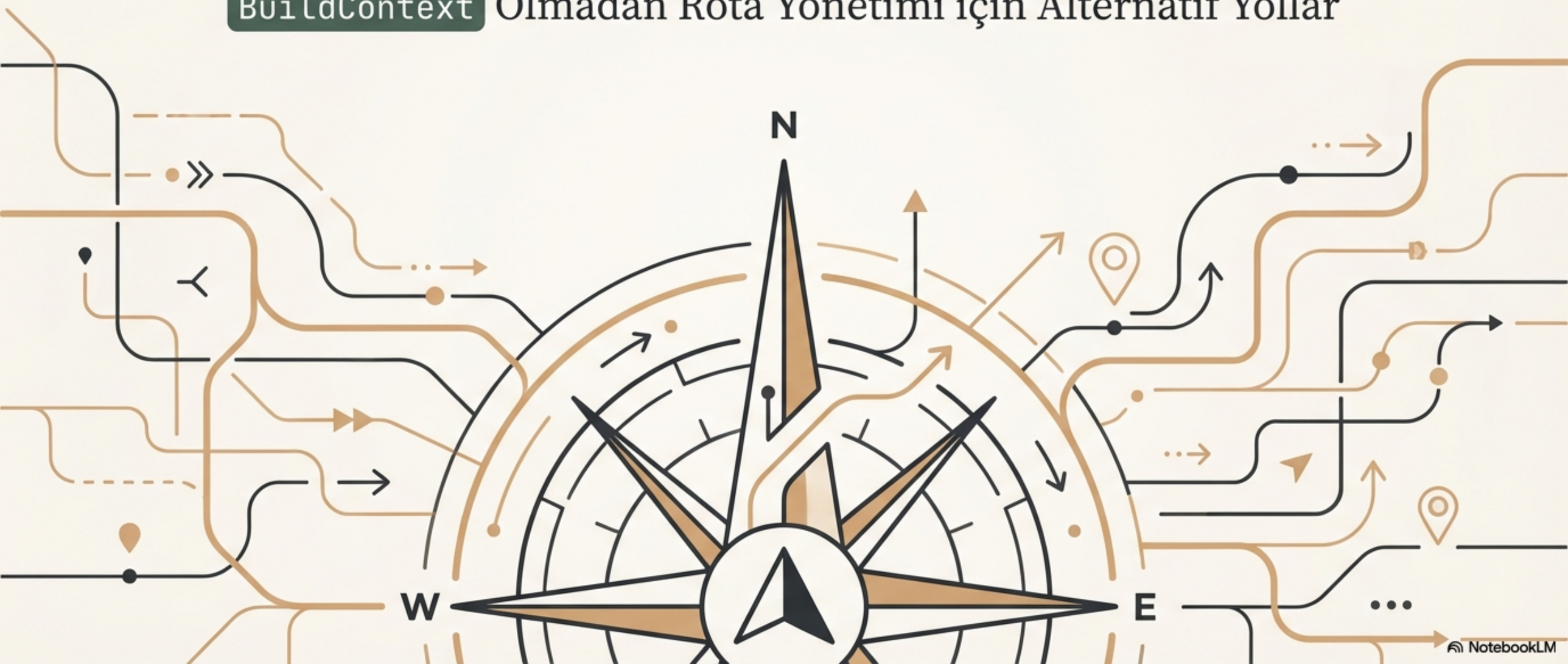
Flutter Navigasyonunun Ötesi

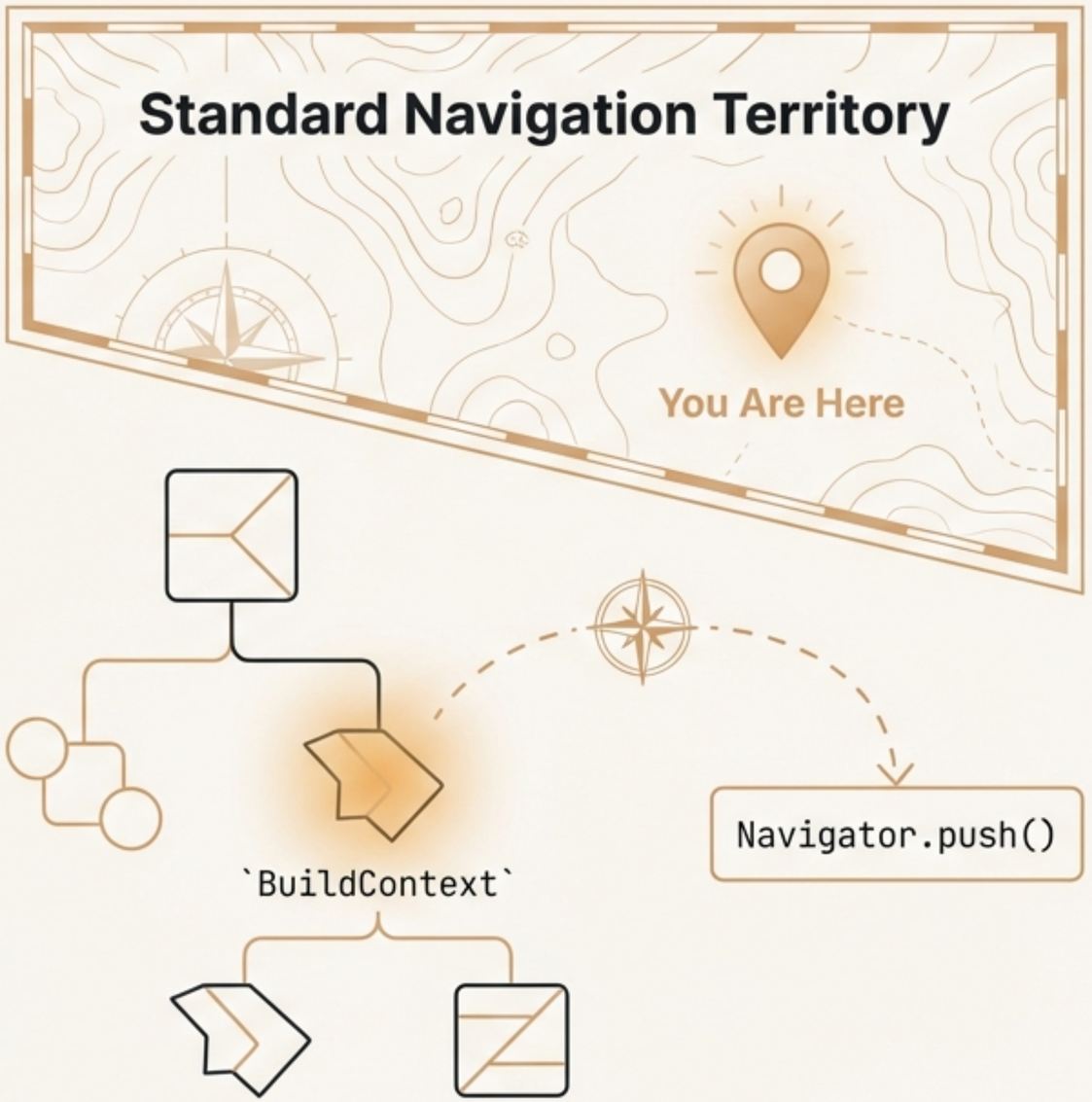
`BuildContext` Olmadan Rota Yönetimi için Alternatif Yollar



Yolculuğun Başlangıcı: Standart Navigasyon

Flutter’da rotalar arasında geçiş yapmak için temel bir gereksinim vardır: `BuildContext`.

Neredeyse tüm standart navigasyon işlemleri, `Navigator.of(context)` çağrısı ile widget ağacındaki mevcut konuma dayanır. Bu, çoğu senaryo için güçlü ve sezgisel bir yaklaşımdır.



Yol Ayrımı: `BuildContext` Olmadan Navigasyon İhtiyacı

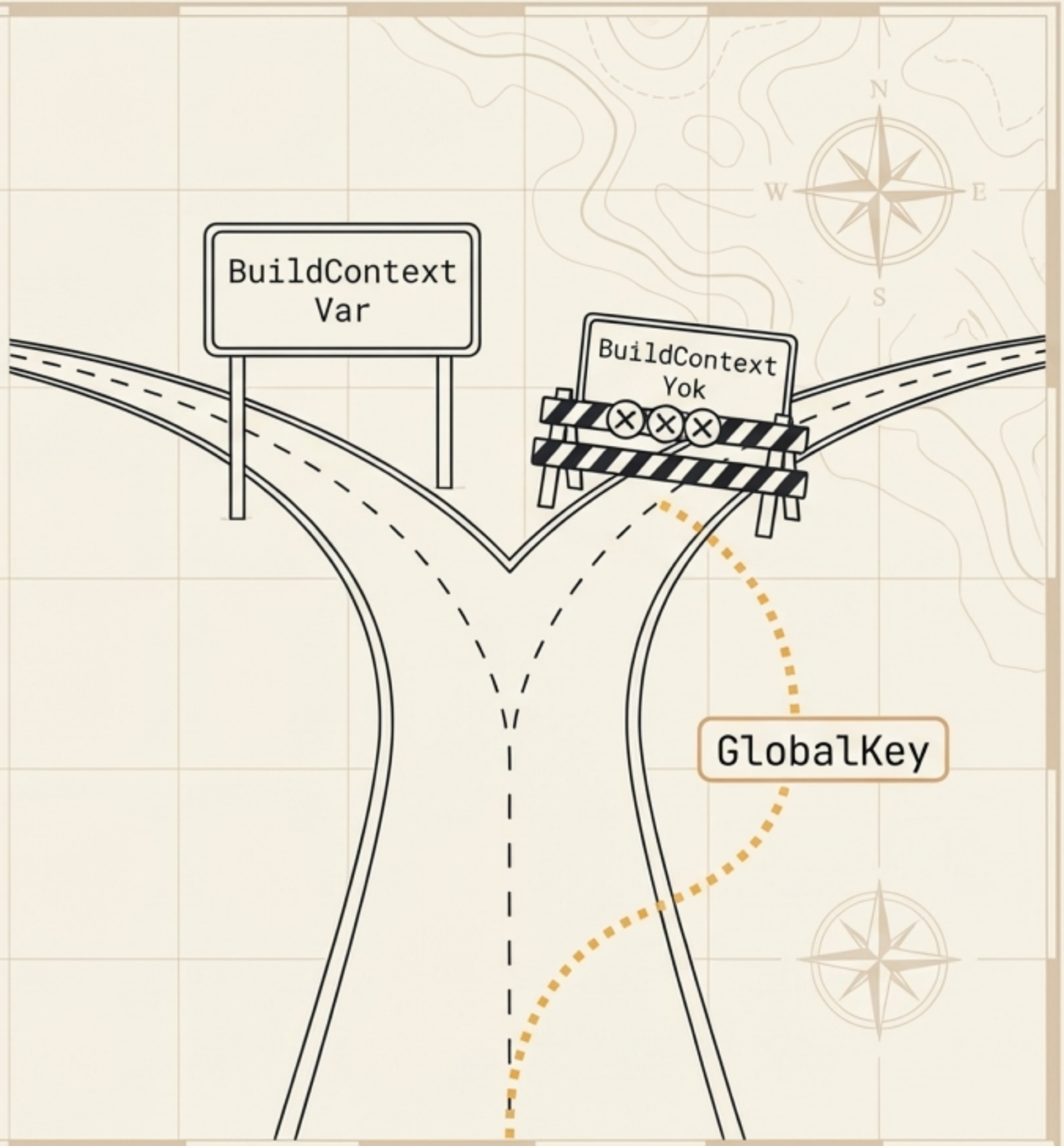
Bazen, elinizde bir `BuildContext` olmadan yeni bir sayfaya geçmeniz gerekebilir. Bu durum genellikle UI katmanı dışındaki kodlardan (örrneğin, bir business logic component veya bir servis sınıfı) tetiklenen navigasyon işlemlerinde ortaya çıkar.

Widget ağacının dışından `Navigator'a nasıl erişebiliriz?

Gizli Bir Kestirme Yol: `GlobalKey` Kullanımı

Flutter, bu sorunu çözmek için `GlobalKey` adında güçlü bir mekanizma sunar. Bu anahtar, `BuildContext`'e ihtiyaç duymadan `Navigator` durumuna erişmemizi sağlar. sağlar.

Bu yöntem, tüm uygulama için tek bir erişim noktası oluşturarak, kodun herhangi bir yerinden navigasyon işlemlerini tetiklememize olanak tanır.



`GlobalKey` ile Navigasyon: Adım Adım Kod

Adım 1: Anahtarı Tanımlayın

Bir yardımcı sınıfta (örneğin `RouteGenerator`) statik bir `GlobalKey` oluşturun.

```
// In RouteGenerator.dart
class RouteGenerator {
  RouteGenerator._();
  // Use a key to use a navigator
  without a context
  static final key = GlobalKey<
  NavigatorState>();
  // ...
}
```

Adım 2: Kök Widget'a Bağlayın

Bu anahtarı, `MaterialApp` veya `CupertinoApp` widget'ınızın `navigatorKey` özelliğine atayın.

```
// In main.dart
MaterialApp(
  navigatorKey: RouteGenerator.key,
  // ...
)
```

Adım 3: Navigasyonu Tetikleyin

Artık `BuildContext` olmadan rotalar arasında gezinebilirsiniz.

```
RouteGenerator.key.currentState?
.pushNamed('/details');
```

Kestirme Yolda Dikkat Edilmesi Gerekenler

`GlobalKey` güçlü bir çözüm olsa da, akılda tutulması gereken bazı önemli noktalar vardır:

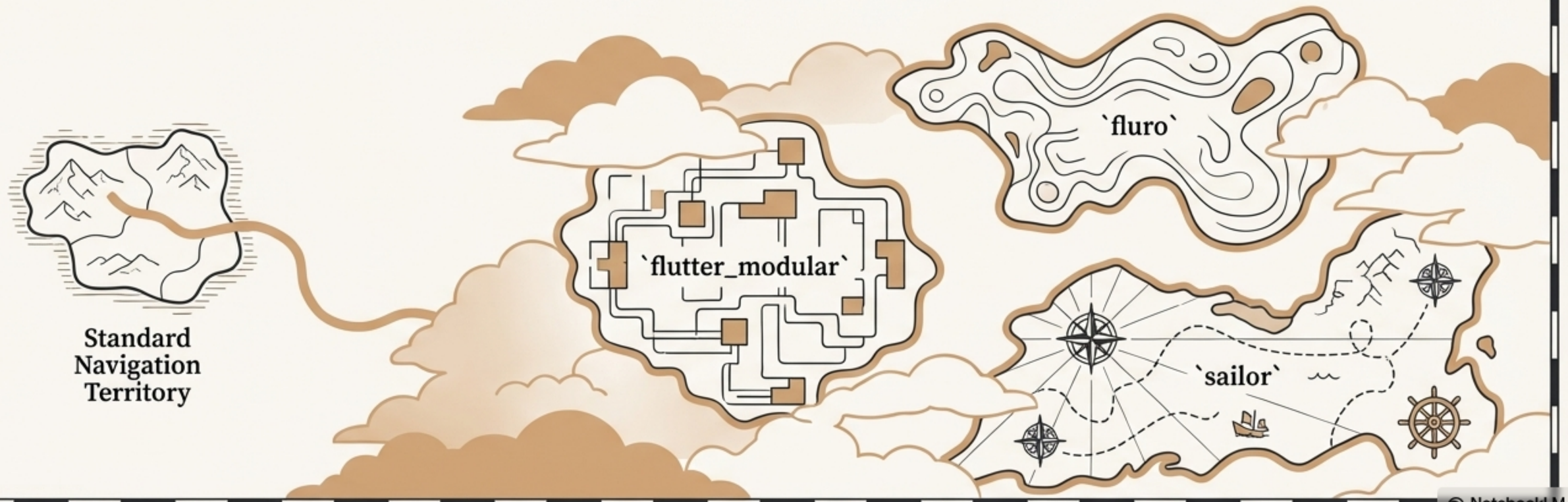
- **Varsayılan Tercih:** Mümkün olduğunca `Navigator.of(context)` kullanmaya devam edin. Bu, varsayılan tercihiniz olmalıdır.
- **Performans Maliyeti:** Global anahtarların kullanımı göreceli olarak maliyetlidir.
- **İhtiyaç Odaklı Kullanım:** `GlobalKey'i yalnızca gerçekten ihtiyaç duyulduğunda kullanın. Bu, nadir durumlar için bir kaçış yoludur, genel bir mimari deseni değildir.

Sınırların Ötesine Yolculuk: Alternatif Rota Kütüphaneleri

Flutter'ın yerleşik navigasyon araçları kullanışlıdır, ancak özellikle büyük uygulamalarda rotalar arası veri paylaşımı ve karmaşık akışların yönetimi zorlaşabilir.

"Bunlar Flutter'ın Navigator yaklaşımından daha iyi veya daha kötü olmak zorunda değil; sadece ilginç bulabileceğiniz bir dizi alternatif."

Şimdi, rota yönetimine tamamen farklı felsefelerle yaklaşan en popüler paketlerden bazılarını keşfedeceğiz.



Bölge: `flutter_modular`

Felsefesi

Modüler mimari ve dahili bağımlılık enjeksiyonu (Dependency Injection) üzerine kuruludur. Rota yönetimini, uygulamanın genel mimarisinin bir parçası olarak ele alır.

Öne Çıkan Özellikler

- **Modüler Yapı:** Rotalar ve bağımlılıklar modüller halinde düzenlenir.
- **binds:** Bağımlılıkların (servisler, BLoC'lar) enjekte edildiği alan.
- **routers:** Merkezi ve bildirimsel (declarative) rota tanımlama listesi.
- **bootstrap:** Uygulamanın başlangıç widget'ını belirler.

`flutter_modular`: Pratikte Bir Bakış

Rotalar ve bağımlılıklar, `MainModule`'dan türetilen bir sınıfta merkezi olarak yönetilir.

```dart
class AppModule extends MainModule {
  // Bağımlılıkların enjekte edildiği yer (provider, bloc vb.)
  @override
  List<Bind> get binds => [];

  // Uygulamanızın rotaları
  @override
  List<ModularRouter> get routers => [
    Router("/", child: (_, __) => HomePage()),
    Router("/info", child: (_, __) => InfoPage()),
  ];

  // MaterialApp veya CupertinoApp içeren widget
  @override
  Widget get bootstrap => MyRootWidget();
}
```

Bağımlılıkların enjekte edildiği yer (provider, bloc vb.)

Uygulamanızın rotaları

MaterialApp veya CupertinoApp içeren widget

Bölge: `Fluro`

Felsefesi

Esnek ve güçlü bir rota tanımlama sistemi sunar. Rota geçişleri ve parametre yönetimi için detaylı kontrol sağlar.

Öne Çıkan Özellikler

- Handler Tabanlı: Her rota, bir `Handler` fonksiyonu ile tanımlanır. Bu, rota oluşturma mantığını merkezi bir yerde toplar.
- Açık Tanımlama: Rotalar, bir `Router` nesnesi üzerinde `define` metodu ile açıkça tanımlanır.
- Gelişmiş Geçişler: `fadeIn` gibi hazır animasyonlu geçişleri destekler.

Fluro: Pratikte Bir Bakış

Fluro'da rota yönetimi üç temel adımdan oluşur: router oluşturma, rota tanımlama ve navigasyon.

```
// 1. Router nesnesini oluştur
final router = Router();

// 2. Bir rota için Handler tanımla
var loginRoute = Handler(
  handlerFunc: (context, params) {
    return LoginScreen();
  }
);

// 3. Handler'ı bir yola ata
void defineRoutes(Router router) {
  router.define("/login", handler: loginRoute);
}

// 4. Navigasyonu gerçekleştir
router.navigateTo(context, "/login",
  transition: TransitionType.fadeIn
);
```

Bölge: `Sailor`

Felsefesi

Minimum kod tekrarı (boilerplate) ile basit, güvenli ve `BuildContext`'ten bağımsız navigasyon sağlamak.

Öne Çıkan Özellikler

- **Statik Erişim:** Tüm navigasyon işlemleri statik bir `Sailor` nesnesi üzerinden yapılır.
- **Merkezi Rota Kaydı:** Rotalar `sailor.addRoute` metodu ile tek bir yerde oluşturulur.
- **Gerçek `Context`-Bağımsızlığı:** `navigate` metodu `BuildContext` gerektirmez.

Sailor: Pratikte Bir Bakış

Sailor'ın kurulumu, rotaların merkezi olarak tanımlanmasını ve `MaterialApp'e entegre edilmesini içerir.

1. Rota sınıfını ve Sailor nesnesini oluştur

```
class Routes {
  static final sailor = Sailor();
  static void createRoutes() {
    sailor.addRoute(SailorRoute(
      name: "/loginPage",
      builder: (context, args, params) {
        return LoginPage();
      },
    ));
  }
}
```

2. Uygulama başlangıcında rotaları oluştur

```
void main() async {
  Routes.createRoutes();
  runApp(const MyApp());
}
```

3. MaterialApp'e entegre et

```
MaterialApp(
  navigatorKey: Routes.sailor.navigatorKey,
  onGenerateRoute: Routes.sailor.generator(),
);
```

4. Context olmadan navigasyonu tetikle

```
Routes.sailor.navigate("/loginPage");
```

Kendi Rotanızı Çizmek: Yaklaşımların Karşılaştırması

Her yaklaşımın kendine özgü avantajları ve kullanım senaryoları vardır. Projenizin ihtiyaçlarına en uygun olanı seçin.

Yaklaşım	Basitlik	Ölçeklenebilirlik	`Context` Bağımlılığı	Anahtar Özellik
Standart Navigator	Yüksek	Düşük-Orta	Gerekli	Flutter'ın yerleşik ve temel çözümü
GlobalKey	Yüksek (tek amaç için)	Düşük	Gerekli Değil	Hızlı, `context`-dışı çözüm
flutter_modular	Orta	Yüksek	Gerekli Değil	DI ile entegre modüler mimari
Fluro	Orta	Orta-Yüksek	Gerekli (bazı durumlarda)	Esnek `Handler` tabanlı tanımlama
Sailor	Yüksek	Orta	Gerekli Değil	`Context`-bağımsız, basit API

Yolculuk Size Ait

Keşfettiğimiz bu yollar, Flutter'ın standart navigasyon sistemine güçlü alternatifler sunar. Projenizin mimarisi, ekibinizin alışkanlıkları ve uygulamanızın karmaşıklığı, hangi patikanın sizin için en doğrusu olduğunu belirleyecektir.

En iyi rota yönetimi, sadece bir hedefe ulaşmak değil, aynı zamanda yolculuğu sürdürülebilir ve keyifli kılmaktır.