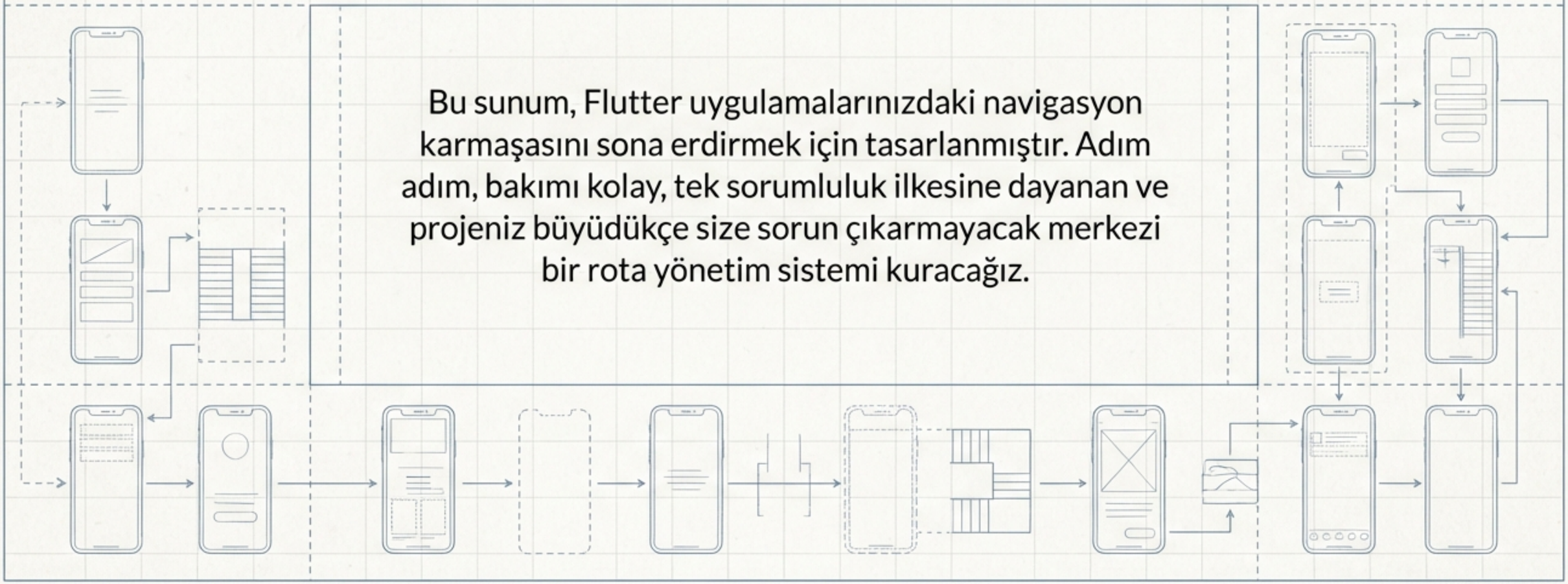
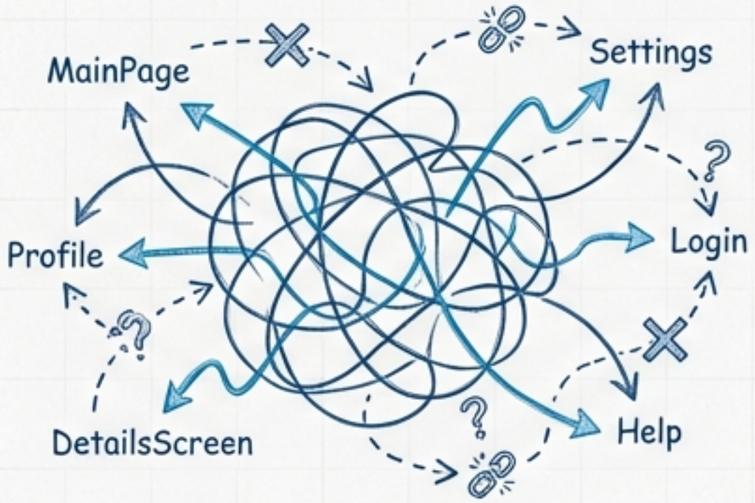
Flutter'da Usta İşi Rota Yönetimi

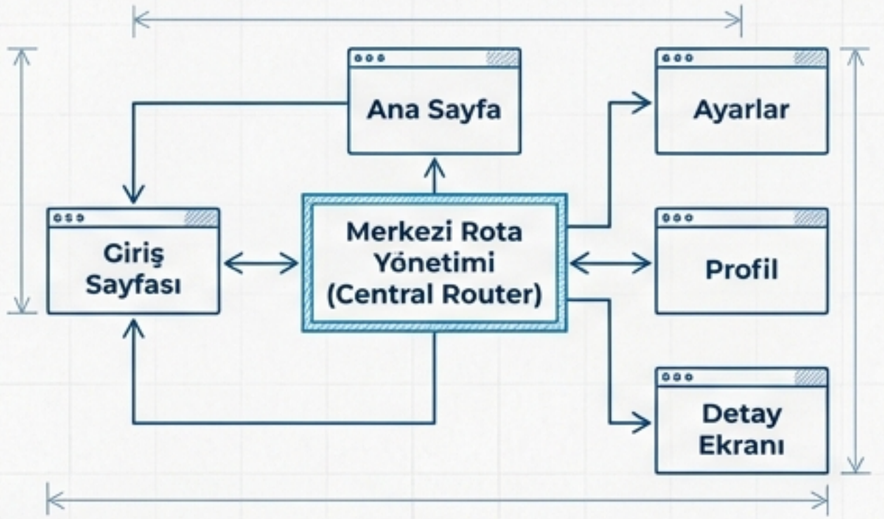
Temiz, Ölçeklenebilir ve Merkezi Bir Navigasyon Mimarisi İnşa Etmek



Bu sunum, Flutter uygulamalarınızdaki navigasyon karmaşasını sona erdirmek için tasarlanmıştır. Adım adım, bakımı kolay, tek sorumluluk ilkesine dayanan ve projeniz büyüdükçe size sorun çıkarmayacak merkezi bir rota yönetim sistemi kuracağız.

Neden Merkezi Bir Rota Mimarisine İhtiyacımız Var?





Sorun (The Mess)

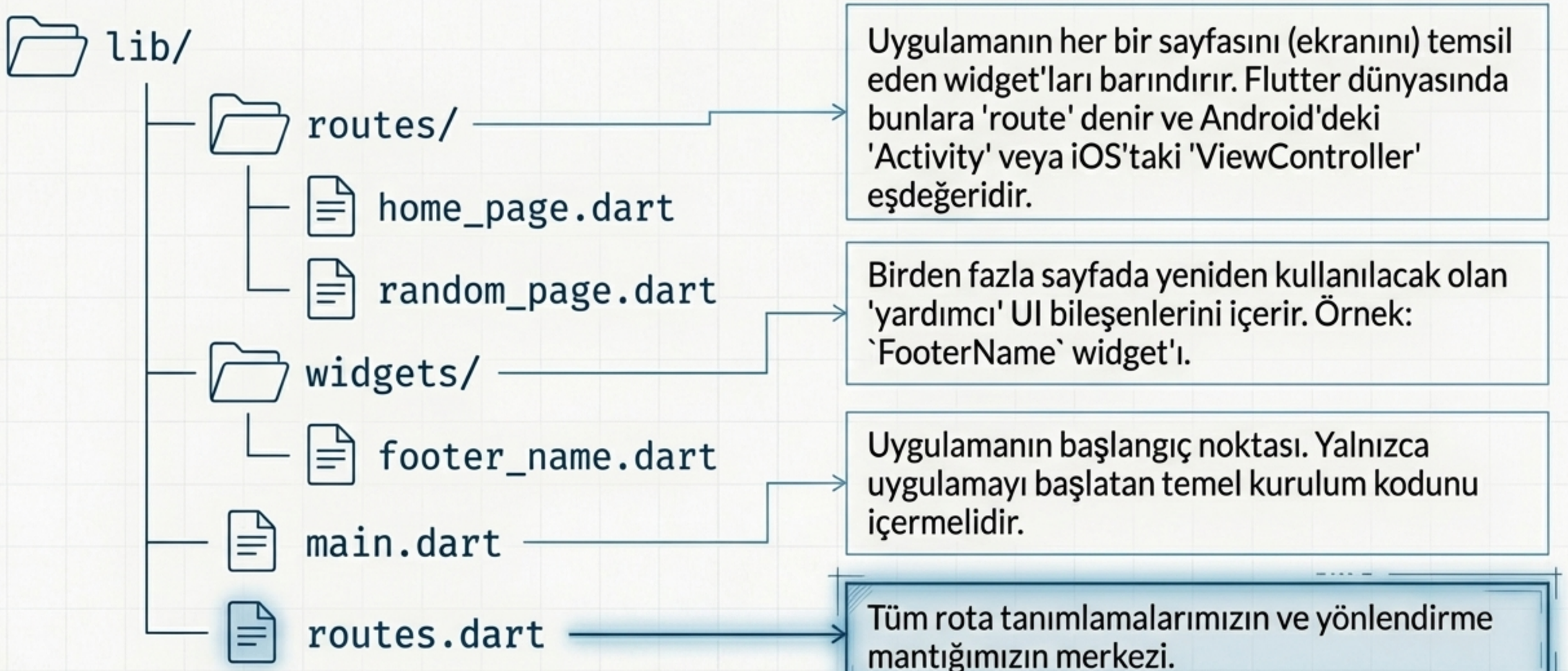
- Büyüyen projelerde dağınık ve takip etmesi zorlaşan navigasyon kodları.
- Kodun farklı yerlerine serpiştirilmiş MaterialPageRoute çağrıları.
- Rota mantığında yapılacak bir değişikliğin birden fazla dosyayı etkilemesi.
- Bakım ve test edilebilirliğin azalması.

Çözüm (The Architecture)

- ☑ Tüm rota mantığını tek bir yerde toplamak (Single Responsibility Principle).
- ☑ Rotaları isimlendirilmiş (named routes) sabitlerle yöneterek yazım hatalarını önlemek.
- ⚙ Uygulama genelinde temiz, okunabilir ve öngörülebilir bir navigasyon akışı sağlamak.
- ⚙ Yeni sayfalar eklemeyi veya mevcutları değiştirmeyi basitleştirmek.

Amacımız, navigasyon mantığını uygulamanın kalbinde merkezi bir kontrol kulesi gibi yönetmektir.

Mimari Planımız: Önerilen Proje Yapısı



1. Adım: Rotalarımızı (Sayfalarımızı) İnşa Edelim

`routes/home_page.dart`

```dart
// Located in routes/home_page.dart
class HomePage extends StatelessWidget {
  const HomePage();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: RaisedButton(
          onPressed: () { /* Henüz boş */ },
          child: const Text("Random"),
        ),
      ),
    );
  }
}
```

`routes/random_page.dart`

```dart
// Located in routes/random_page.dart
class RandomPage extends StatelessWidget {
  const RandomPage();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
    return Scaffold(
      body: Center(
        child: Text("${Random().nextInt(20)}"),
      ),
    );
  }
}
```

Uygulamamız için iki basit sayfa oluşturuyoruz: HomePage ve RandomPage'. Her ikisi de bir Scaffold içeren temel StatelessWidget'lardır ve önerilen routes/ klasöründe yer alırlar.

2. Adım: Uygulamanın Giriş Noktası: `main.dart`

```dart
// main.dart

import 'package:flutter/material.dart';

void main() => runApp(const RandomApp());

class RandomApp extends StatelessWidget { ①
  const RandomApp();

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      onGenerateTitle: (context) => "Random App",
      initialRoute: RouteGenerator.homePage, // ②
      onGenerateRoute: RouteGenerator.generateRoute, // ③
      debugShowCheckedModeBanner: false,
    );
  }
}
```

① **RandomApp**: Ağacın kökünde yer alan ve uygulama için rota ayarlarını yapan widget.

② **initialRoute**: Uygulama Uygulama açıldığında yüklenecek olan ilk rotanın yolunu (`'/'`) belirten bir string.

③ **onGenerateRoute**: Bir rota ismi (`/random` gibi) istendiğinde hangi widget'ın oluşturulacağını belirleyen metot referansı. Navigasyonun kalbi burada atmaya başlar.

Mimarinin Kalbi: `routes.dart` ve `RouteGenerator` Sınıfı

- Tüm rota yönetimini tek bir sınıfta toplamak, 'tek sorumluluk ilkesi'ni (single responsibility principle) mükemmel bir şekilde karşılar.
- Bu yaklaşım kodu temiz tutar. Rotalarla ilgili bir değişiklik yapmanız gerektiğinde, bakmanız gereken tek bir yer vardır: `routes.dart`.
- Bu sınıfa istediğiniz ismi verebilirsiniz, ancak içinde 'route' kelimesinin geçmesi, amacını anında belli etmesi açısından iyi bir pratiktir.

// routes.dart

```
// routes.dart
class RouteGenerator {
  // 1. Rota isimleri için sabitler (const
  strings)


  // 2. Sınıfın örneğinin oluşturulmasını
  engelleyen özel kurucu


  // 3. Rota isimlerini widget'lara eşleyen
  statik metot

}
```

Bu sınıf, global bir fonksiyon tanımlamak yerine statik bir metot için sadece bir **'wrapper' (sarmalayıcı)** görevi görür. Bu, daha temiz bir koddur.

`RouteGenerator`'ın İç İşleyişi

```dart
// routes.dart

class RouteGenerator {
  static const String homePage = '/'; ①
  static const String randomPage = '/random'; ②

  RouteGenerator._(); // Private constructor

  static Route<dynamic> generateRoute(RouteSettings settings) {

    switch (settings.name) { ③
      case homePage:
        return MaterialPageRoute(builder: (_) => const HomePage());

      case randomPage:
        return MaterialPageRoute(builder: (_) => const RandomPage());

      default:
        // Hata durumunda gösterilecek bir sayfa da döndürülebilir.
        throw FormatException("Route not found");
    }
  }
}
```

Rota isimlerini sabitler olarak tanımlamak, **`pushNamed`** çağrılarında yazım hatası yapma riskini ortadan kaldırır.

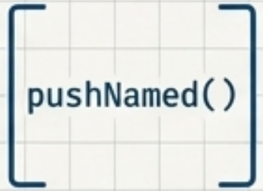
Uygulama başladığında gösterilecek ilk sayfanın yolu '/' olmalıdır. Bu bir zorunluluktur.

Navigator tarafından iletilen rota ismini içerir. **`switch`** ifadesi bu değere göre karar verir.

Android stili sayfa geçiş animasyonu sağlar. iOS için **`CupertinoPageRoute`** kullanılabilir.

Navigasyon Akışı: Perde Arkasında Neler Oluyor?

Kullanıcı `HomePage'deki 'Random' butonuna tıklar.

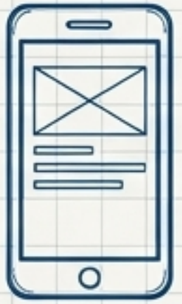


`Navigator.of(context).pushNamed('/random')` çağrılır.

`MaterialApp` `onGenerateRoute` özelliği, isteği `RouteGenerator.generateRoute` metoduna yönlendirir.

`generateRoute` içindeki `switch` ifadesi `'/random'` ismini `RandomPage` widget'ı ile eşleştirir.

`MaterialPageRoute( (builder: (_) => const RandomPage())` oluşturulur ve `Navigator`'a geri döndürülür.



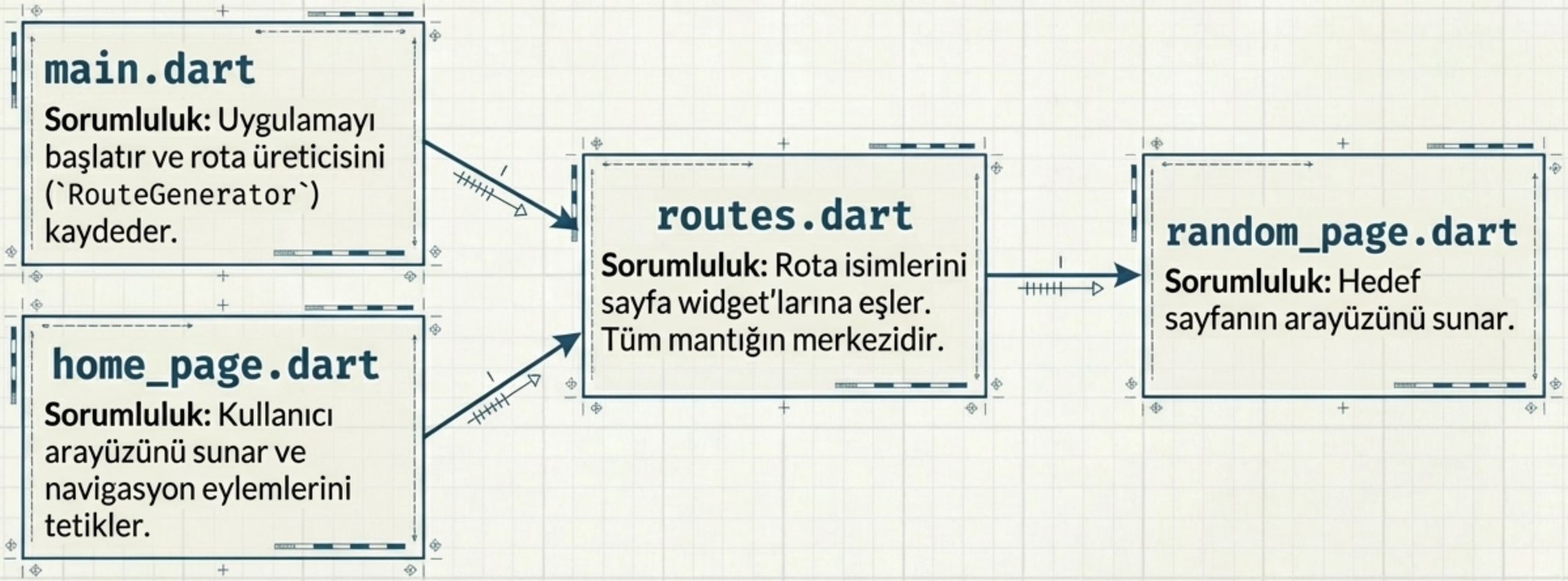
`Navigator`, `RandomPage'i` ekranda gösterir.

3. Adım: Sayfalar Arası Geçişi Tetiklemek

```
// The 'HomePage' widget's build method
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Center(
      child: RaisedButton(
        onPressed: () ⇒
            Navigator.of(context)?.pushNamed(
                RouteGenerator.randomPage
        ),
        child: const Text("Random"),
      ),
    ),
  );
}
```

- Sayfalar arası geçiş yapmak için **Navigator.of(context).pushNamed()** metodunu kullanırız.
- Parametre olarak, **RouteGenerator** sınıfımızda tanımladığımız **statik sabiti** (**RouteGenerator.randomPage**) vererek hem **güvenli** hem de **okunabilir** bir kod yazmış oluruz.
- Bu çağrı, bir önceki slaytta gördüğümüz tüm akışı tetikler.
- Geri dönmek için ise **Navigator.of(context)?.pop()** metodunu çağırmak yeterlidir.

Mimarinin Tamamlanmış Hali: Büyük Resim



Bu yapı sayesinde her dosyanın net bir sorumluluğu vardır.
Sistem temiz, modüler ve kolayca genişletilebilirdir.

İleri Seviye İpucu: Durum Yönetimi Entegrasyonu

Sayfalarınızın ortak bir **veriye (cache** gibi) veya **duruma (notifier** gibi) erişmesi gerekiyorsa, `MaterialApp` widget'ını bir `MultiProvider` ile sarmak iyi bir pratiktir. Rota mimarimiz bu yapıyla sorunsuz çalışır.

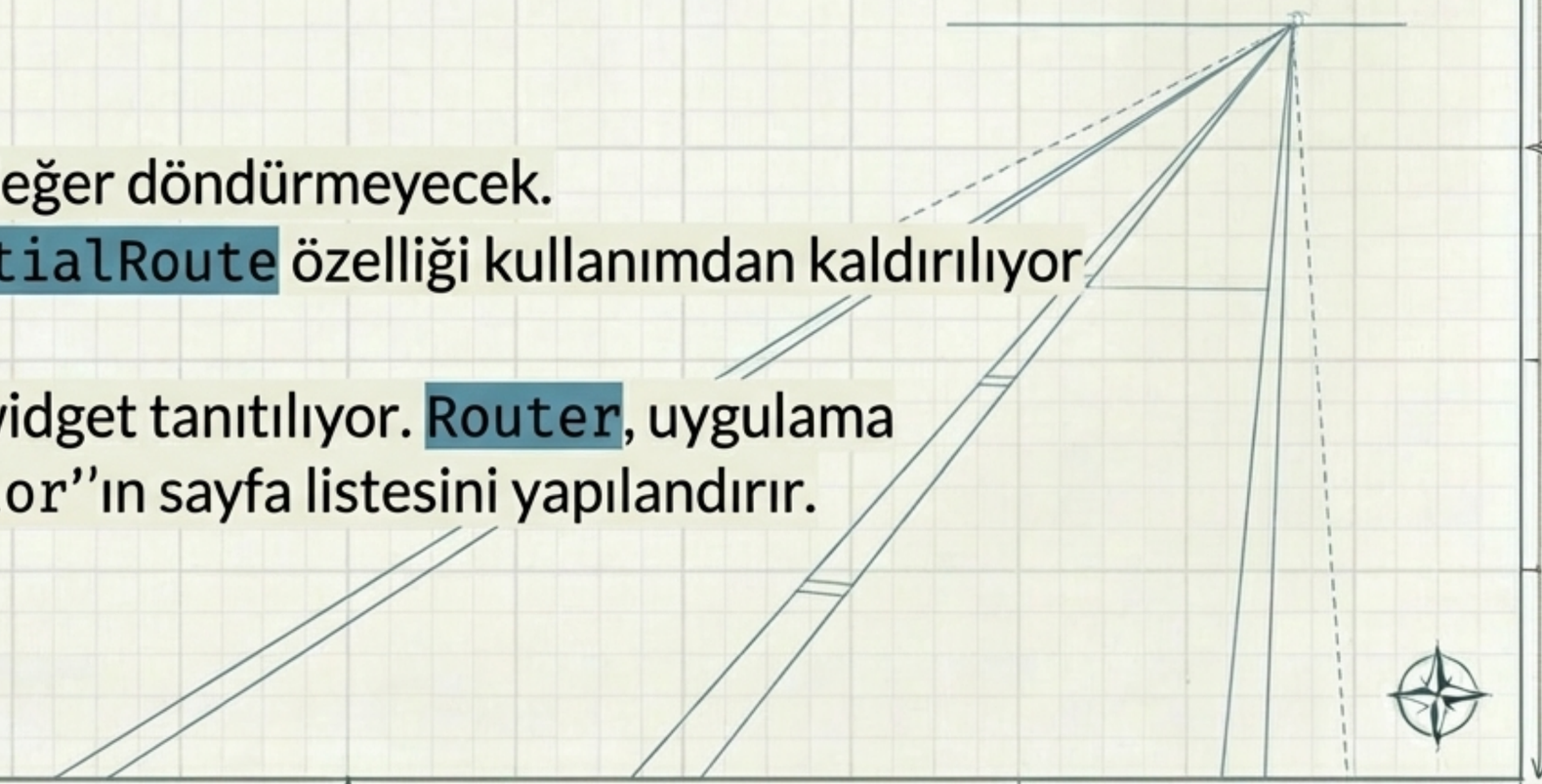
```
// main.dart
@override
Widget build(BuildContext context) {
  return MultiProvider(
    providers: [
      Provider(create: (_) => DataCache()),
      ChangeNotifierProvider(create: (_) => Something()),
    ],
    child: MaterialApp(
      // ... onGenerateRoute ve diğer ayarlar
    ),
  );
}
```

Tüm uygulama için paylaşılan durumlar burada sağlanır

Bu sayede **Provider** aracılığıyla sağlanan servisler, **RouteGenerator** tarafından oluşturulan tüm sayfalardan erişilebilir hale gelir.

Ufukta Görünenler: Navigator 2.0'a Hazırlık

Flutter ekibi, navigasyon sistemi üzerinde güncellemeler yapmaktadır. 'Navigator 2.0' olarak bilinen bu güncelleme, mevcut `Navigator` sınıfını kaldırmayacak ancak bazı önemli değişiklikler ve yeni bir `Router` widget'ı getirecektir.



Önemli Değişiklikler

- `pop()` metodu artık bir değer döndürmeyecek.
- `RouteSettings.isInitialRoute` özelliği kullanımdan kaldırılıyor (`deprecated`).
- `Router` adında yeni bir widget tanıtılıyor. `Router`, uygulama durumuna göre `Navigator`'ın sayfa listesini yapılandırır.

Navigator 2.0 Geçişi: Pratik Kod Değişiklikleri

Initial Route Yönetimi

Önce

```
// RouteGenerator
static Route generateRoute
   (RouteSettings settings) {
   if (settings.isInitialRoute) {
      return A(); }
   return B();
}
// MaterialApp
MaterialApp(
   onGenerateRoute:
      RouteGenerator.
      generateRoute,
)
```

Sonra

```
// RouteGenerator
static List<Route> generate
   InitialRoutes(String name) {
   return [ A() ];
}

// MaterialApp
MaterialApp(
   onGenerateInitialRoutes:
      RouteGenerator.generate
      InitialRoutes,
   onGenerateRoute:
      RouteGenerator.
      generateRoute,
)
```

`pop` Metodu Kullanımı

Önce

```
if (Navigator.pop(context)) {
 ... }  Artık bir değer
döndürmez
```

Sonra

```
if (Navigator.canPop(con-
   text)) { /* ... */ }
} else { /* ... */ }

Navigator.pop(context);
```

Temel Çıkarımlar ve Sonraki Adımlar

Bu sunumda, Flutter uygulamaları için sağlam, merkezi ve ölçeklenebilir bir navigasyon mimarisinin temel taşlarını döşedik.

Temizlik

Navigasyon mantığı tek bir yerde. Kod tabanınız daha okunabilir ve düzenli.

Ölçeklenebilirlik

Yeni sayfalar eklemek veya mevcut akışları değiştirmek mimariyi bozmadan kolayca yapılabilir.

Bakım Kolaylığı

Tek Sorumluluk İlkesi sayesinde, rotalarla ilgili sorunları bulmak ve düzeltmek çok daha hızlı.

Bu yapıyı bir sonraki projenizde deneyin ve kodunuzun nasıl daha organize hale geldiğini kendiniz görün. **Sağlam bir temel**, başarılı bir uygulamanın anahtarıdır.