2025

Erciyes Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği

Mobile Application Development
Dr. Öğr. Üyesi Fehim KÖYLÜ

Navigasyon ve Routing

VİZE PROJESİ

BETÜL MOLLAMUSA - 1030510334

FLUTTER'DA NAVIGATION VE ROUTİNG

1. Giriş

Mobil uygulama geliştirme süreçlerinde kullanıcı etkileşimini artırmak ve çok ekranlı yapılar oluşturmak için etkili bir yönlendirme (navigasyon) sistemi büyük önem arz etmektedir. Kullanıcıların bir ekran üzerinden diğerine geçmesini sağlayan bu yapıların doğru şekilde kurgulanması, kullanıcı deneyiminin temel belirleyicilerindendir. Flutter, kullanıcı arayüzlerinin hızlı ve etkili bir şekilde geliştirilmesine olanak tanıyan modern bir UI framework'üdür. Bu raporda, Flutter ortamında sayfa geçişlerini sağlayan navigation ve routing kavramları teorik bir yaklaşımla ele alınmış, çeşitli uygulama örnekleri ile desteklenerek açıklanmıştır.

2. Navigation ve Routing Kavramları

Navigation, bir kullanıcının uygulama içerisindeki farklı sayfalar (ekranlar) arasında geçiş yapmasını ifade eder. Bu geçişler; buton tıklamaları, liste elemanı seçimi veya formların gönderimi gibi kullanıcı etkileşimleri sonucunda gerçekleşir.

Routing ise, bu geçişlerin nasıl ve hangi yollarla gerçekleştirileceğini tanımlar. Flutter'da routing kavramı, bir ekranın çağrılmasını sağlayan route tanımları aracılığıyla uygulamaya entegre edilir. Routing, sadece hedef ekranı değil, aynı zamanda geçişte kullanılan animasyonları ve parametreleri de kapsayan daha geniş bir yapıdır.

3. Flutter'da Navigation Yapısı ve Sayfa Yönetimi

Flutter'da sayfalar bir yığın (stack) veri yapısı ile yönetilir. Her yeni ekran, mevcut ekranın üzerine eklenir ve bu yapı, Navigator sınıfı tarafından yönetilir. Bu yapı, klasik LIFO (Last-In, First-Out) prensibiyle çalışır. Kullanıcı geri dönmek istediğinde en üstteki ekran kaldırılır (pop edilir) ve bir önceki ekrana geri dönülür.

Aşağıda bu yapıyı yöneten temel Navigator metotları açıklanmıştır:

- *push():* Yeni bir sayfayı yığının en üstüne ekler.
- *pop():* En üstteki sayfayı kaldırarak bir önceki sayfaya dönüş sağlar.
- *pushReplacement():* Mevcut sayfanın yerine başka bir sayfayı yerleştirir.

- *popUntil():* Belirli bir koşul sağlanana kadar ekranları geri alır.

Bu mekanizma sayesinde, uygulama içindeki geçişlerin kontrolü programatik hale gelir.

4. Route Kavramı ve Türleri

Flutter'da bir sayfa (ya da ekran), teknik anlamda bir Route nesnesiyle temsil edilir. Flutter, farklı platformlara özel geçiş efektlerini sağlayabilmek için farklı Route sınıfları sunar:

- *MaterialPageRoute*: Android platformuna özgü geçiş animasyonları içerir.
- *CupertinoPageRoute*: iOS platformuna özel animasyonlar sağlar.
- *PageRouteBuilder*: Geçiş animasyonlarının geliştirici tarafından özelleştirilmesini mümkün kılar.

Örnek:

```
Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => DetailPage()),
);
```

Bu örnekte, *DetailPage* adlı ekran çağrılmış ve yığının en üstüne eklenmiştir. Kullanıcı bu ekrandan geri dönmek isterse *Navigator.pop(context)* çağrısıyla önceki sayfaya dönebilir.

5. Named Routes Kullanımı

Yönlendirme işlemlerinin daha yönetilebilir olması için *"named route"* yapısını Kullanılmaktadır. Bu yöntemle her sayfaya bir isim atanır ve geçişler bu isimler üzerinden yapılır. Böylece yönlendirme süreçleri merkezi şekilde kontrol edilebilir.

Örnek:

```
MaterialApp(
  initialRoute: '/',
  routes: { '/': (context) => HomePage(),
            '/settings': (context) => SettingsPage(), },);
```

Kullanım:

```
Navigator.pushNamed(context, '/settings');
```

Bu yapı sayesinde sabit yollarla sayfalar arası yönlendirme sağlanmış olur. Ayrıca *Navigator.pop(context)* çağrısı ile tekrar önceki sayfaya dönülebilir.

6. Dinamik Yönlendirme: onGenerateRoute

Bazı uygulamalarda, sayfa geçişleri sırasında dinamik veriler taşınması gerekebilir. Örneğin bir kullanıcı profili sayfasına geçerken kullanıcı ID'si gibi parametrelerin aktarılması gerekir. Bu durumlarda onGenerateRoute fonksiyonu kullanılmaktadır.

Örnek:

```
MaterialApp(
  onGenerateRoute: (RouteSettings settings) {
    if (settings.name == '/userProfile') {
      final userId = settings.arguments as int;
      return MaterialPageRoute(
        builder: (context) => UserProfilePage(userId: userId),
      );
    }
    return null;
  },
);
```

Bu yapı sayesinde, uygulama içerisindeki tüm yönlendirmeler bir fonksiyon içinde kontrol edilerek daha esnek hale getirilebilir.

7. Sayfalar Arası Veri Alışverişi

Flutter'da sadece sayfa geçişleri değil, aynı zamanda sayfalar arasında veri alışverişi de mümkündür. Bu sayede, örneğin bir form ekranından dönen veri ana ekranda kullanılabilir.

Örnek:

```
final result = await Navigator.push(
  context,
  MaterialPageRoute(builder: (context) => InputScreen()),
);

// Dönen verinin kullanımı
print("Kullanıcının girdisi: $result");
```

Veri göndermek için ise pop() metoduna parametre ile dönüş yapılabilir:

```
Navigator.pop(context, "Tamamlandı");
```

8. Yeni Nesil Router API (Navigation 2.0)

Flutter 2.0 ile tanıtılan Router API'si, geleneksel Navigator yapısının ötesine geçerek web uygulamaları ve ileri seviye durum yönetimi için daha kapsamlı bir yönlendirme yapısı sunmaktadır.

Bu yeni yapıda Router, RouteInformationParser ve RouterDelegate gibi bileşenler kullanılarak sayfa geçişleri, uygulama durumları ile senkronize biçimde yönetilebilmektedir. Ancak bu yapı geleneksel navigasyona göre daha karmaşıktır ve genellikle büyük ölçekli ya da web tabanlı uygulamalarda tercih edilmektedir.

9. Uygulama Örnekleri ile İnceleme

9.1. Temel Navigasyon Örneği

Bu örnekte, kullanıcı bir butona tıkladığında SecondPage sayfası açılmakta ve önceki sayfa yığının altında kalmaktadır.

```
ElevatedButton(
  onPressed: () {
    Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => SecondPage()),
    );
  },
  child: Text("İleri"),
);
```

9.2. Named Route Kullanımı

Bu kullanım, önceden tanımlanmış bir route'a yönlendirme yapar. Geri dönüş pop(context) ile sağlanır.

```
Navigator.pushNamed(context, '/settings');
```

9.3. onGenerateRoute ile Veri Aktarımı

Bu çağrı sonucunda, kullanıcı profiline ait ekran *userId: 1234* ile açılacaktır. Bu da dinamik içerik sunumu açısından önemlidir.

```
Navigator.pushNamed(
  context,
  '/userProfile',
  arguments: 1234,
);
```

10. Sonuç ve Değerlendirme

Bu raporda, Flutter framework'ü üzerinde navigation ve routing mekanizmaları teorik ve uygulamalı biçimde incelenmiştir. Sayfa geçişlerinin yığın mantığına dayalı olarak

yönetilmesi, Navigator sınıfı ve Route yapıları ile sağlanmakta; geçişler farklı şekillerde yapılandırılabilmektedir.

Geliştiricilere esneklik sağlayan named routes ve onGenerateRoute fonksiyonu gibi özellikler, büyük ölçekli uygulamalarda navigasyonun merkezi bir şekilde kontrolünü mümkün kılmaktadır. Öte yandan, Flutter 2.0 ile tanıtılan Router API, modern web uygulamaları için daha gelişmiş bir yönlendirme altyapısı sunarak framework'ün yetkinliklerini artırmıştır.

Sonuç olarak, Flutter'da navigation yapısının doğru anlaşılması ve uygulanması, uygulamaların kullanıcı dostu, okunabilir ve sürdürülebilir olmasını sağlayan temel bileşenlerden biridir.