T.C.

ERCİYES ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

Bilgisayar Mühendisliği Bölümü

MOBILE APPLICATION DEVELOPMENT PROJE ÖDEVİ

Navigasyon ve Routing

Hazırlayan

MERVE TARHAN 1030510276

Öğretim Üyesi

Dr. Öğr. Üyesi FEHİM KÖYLÜ

Kayseri,2025

FLUTTER'DA NAVİGASYON VE ROUTING

1. Navigasyon ve Routing Nedir?

Flutter uygulamalarında navigasyon, bir ekrandan (sayfadan) başka bir ekrana geçişi ifade eder. Bu, kullanıcıların uygulamayı kullanırken bir işlem yaptıktan sonra farklı sayfalara yönlendirilmesini sağlar. Örneğin, giriş ekranından ana sayfaya geçmek bir navigasyon işlemidir.

Routing ise bu geçişlerin nasıl yapılacağını belirler. Hangi sayfa hangi rota (route) ile gösterilecek, nasıl çağrılacak gibi konular routing kapsamında ele alınır. Flutter, bu işlemler için Navigator adında özel bir widget ve route sistemleri sunar.

2. Temel Navigasyon Konseptleri

Stack-Based Navigasyon

Flutter'da navigasyon sistemi "stack (yığın)" yapısı ile çalışır. Yeni bir sayfa açıldığında, önceki sayfanın üstüne eklenir (push). Geri dönülmek istendiğinde en üstteki sayfa çıkarılır (pop).

Routes ve Pages

- **Route:** Flutter'da bir sayfaya verilen isimdir. Route'lar sayesinde uygulamanın hangi kısmına gidileceği belirlenir.
- **Page:** Uygulamanın kullanıcıya gösterdiği ekranlardır.

o MaterialPageRoute Kullanımı

MaterialPageRoute, Android tarzı geçiş animasyonları sağlar. Genellikle Material Design kullanılan uygulamalarda tercih edilir.

o CupertinoPageRoute Kullanımı

CupertinoPageRoute, iOS tarzı animasyonları ve geçiş davranışlarını taklit eder. Cupertino tasarım sistemini kullanan uygulamalar için uygundur.

Navigator Widget'ı

Navigator, route'ların yönetildiği yapıdır. Push ve pop işlemleri Navigator üzerinden yapılır. Uygulama içinde context ile erişilir.

- **Push**: Bir ekran açmak veya geçiş yapmak için kullanılır.
- **Pop**: Mevcut sayfayı kapatır ve bir önceki sayfaya geri döner.

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

Navigator.pop(context);
```

Şekil1: Navigator.push ve Navigator.pop Örneği

Bu kod parçasındaki ilk örnekte *SecondPage()*adlı ekrana gidebilmek için gerekli olan Navigator yöntemi gösterilmiştir. Bu kod örneğinde MetarialPageRoute geçiş efektini kullanarak sayfalar arası geçiş yapılmaktadır.
İkinci örnekte ise ekrandan çıkış yapılıyor.

3. Navigasyon Türleri

Named Routes (Adlandırılmış Rotalar)

Named routes, uygulama içerisinde her sayfanın bir isimle tanımlanarak çağrılmasını sağlar. Bu yöntem özellikle çok sayfalı uygulamalarda sayfa yönetimini kolaylaştırır.

```
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Named Routes Demo',
      initialRoute: '/',
      routes: {
        '/': (context) => const HomePage(),
        '/second': (context) => const SecondPage(),
      },
    ); // MaterialApp
  }
}
```

Şekil2: Named route örnek tanımlama

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

Şekil3: Named route örnek kullanımı

Anonymous Routes (Anonim Rotalar)

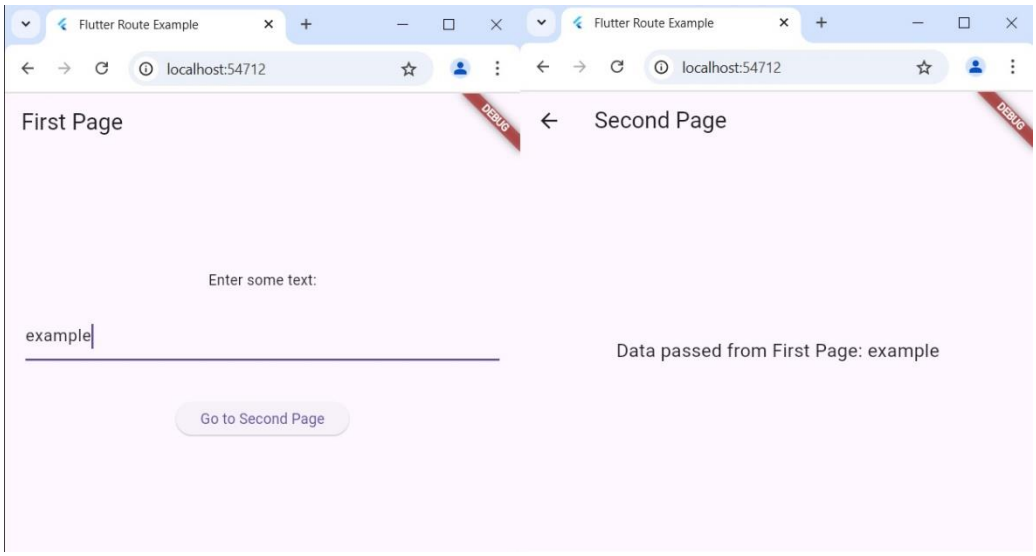
Anonymous routes, bir route ismi tanımlanmadan doğrudan widget üzerinden sayfa geçişi yapılmasını sağlar. Daha küçük uygulamalarda veya sayfa geçişlerinin basit olduğu durumlarda tercih edilir.

Dynamic Routes (Dinamik Rotalar)

Dinamik route yapısı, route'ların çalışma zamanında parametrelerle oluşturulmasını sağlar. Bu yöntem, detay sayfaları gibi parametreye göre değişen içeriklerde kullanılır.

4. Route Arguments (Geçiş Argümanları)

Route arguments, Flutter'da bir sayfadan başka bir sayfaya veri geçmek için kullanılan bir tekniktir. Yani bir sayfada belirli bir veriyi, başka bir sayfada kullanmak üzere taşımak istediğinizde route arguments kullanılır. Bu özellik genellikle uygulamalarda bir kullanıcıyı bir detay sayfasına yönlendirmek veya bir sayfada yapılacak işlemler için gerekli bilgileri diğer sayfalara aktarmak için kullanılır.



Şekil4: Route Arguments Kullanılarak Oluşturulan Basit Uygulama

Bu örnekte route arguments yöntemiyle kullanıcıdan alınan veri bir butona tıklanarak başka bir sayfaya taşınması gösterilmektedir. İlk sayfada bir TextField aracılığıyla kullanıcıdan alınan

veri, Navigator.pushNamed() yöntemi kullanılarak arguments parametresi ile bir sonraki sayfaya iletilmektedir. Böylece sayfalar arası veri iletimi gerçekleştirilmektedir.

```
      child: TextField(
        controller: _controller,
      ), // TextField
    ), // Padding
    SizedBox(height: 20),
    ElevatedButton(
      onPressed: () {
        Navigator.of(context).pushNamed(
          Routes.secondPage,
          arguments: _controller.text,
        );
```

Şekil5: Route Arguments Kullanımı

5. PageRouteBuilder (Geçiş Animasyonlarını Özelleştirme)

Sayfa geçişlerini ve animasyonları özelleştirmek amacıyla kullanılan yapılardan biri de PageRouteBuilder sınıfıdır. Bu yapı, uygulamada kullanılan varsayılan geçiş animasyonlarını değiştirmeye, geçiş süresini ayarlamaya veya tamamen özgün efektler oluşturmaya olanak tanır. PageRouteBuilder, ekranlar(sayfalar) arasında geçiş yapılırken bu geçişin nasıl gerçekleşeceğini tanımlamak için kullanılan bir yapıdır. Bu yapı, iki temel parametre üzerinden çağılır:

- ✓ pageBuilder: Yeni sayfanın hangi içerikle ve nasıl oluşturulacağını belirler.
- ✓ transitionsBuilder: Sayfalar arası geçiş sırasında kullanılacak animasyonu tanımlar.

PageRouteBuilder kullanımı, özellikle kullanıcı deneyimini geliştirmek ve uygulamaya daha akıcı bir his kazandırmak için tercih edilir.

```
Navigator.push(
  context,
  PageRouteBuilder(
    pageBuilder: (context, animation, secondaryAnimation) {
      return const SecondPage();
    },
    transitionsBuilder: (context, animation, secondaryAnimation, child) {
      return SlideTransition(
        position: Tween<Offset>(
          begin: const Offset(1.0, 0.0), // Sağdan başla
          end: Offset.zero,              // Ortada bitir
        ).animate(animation), // Tween
        child: child,
```

Şekil6: PageRouteBuilder Örneği

Bu örnekte, özelleştirilmiş bir sayfa geçişi oluşturmak için PageRouteBuilder kullanılmıştır. PageBuilder fonksiyonu, yeni açılacak olan sayfayı (SecondPage) belirlerken, tansitionBuilder ise geçiş animasyonunu tanımlar.

Bu örnekte, yeni sayfanın ekranın sağından kayarak gelmesi şeklindedir. Bu hareket SlideTransition ile sağlanmıştır. Bir Tween<Offset> kullanılarak, sayfanın başlangıçta ekranın sağ köşesinden (Offset(1.0, 0.0)) başlayıp, animasyonla ekranın merkezine (Offset.zero) gelmesi sağlanmıştır. Bu sayede sayfa geçişi, kullanıcıya daha akıcı ve etkileyici bir şekilde sunulmuştur.

6. Navigator 2.0

Flutter Navigator 2.0, daha karmaşık uygulamalarda (web gibi) kullanılmak için geliştirilmiştir. Bu yapı, URL tabanlı yönlendirme ve sayfa yönetimini daha detaylı kontrol etmeye olanak sağlar.

Navigator 2.0, "Declarative" yaklaşımla çalışır ve Router, RouteInformationParser, RouterDelegate gibi yapılar kullanır. Bu yapı Flutter web geliştirirken daha fazla kontrol sunar ancak basit uygulamalar için Navigator 1.0 yeterlidir.

BottomNavigationBar

BottomNavigationBar, Flutter'da ekranlar arasında geçiş yapmak için kullanılan bir widget'tır ve genellikle uygulamanın alt kısmında görünür. Bu widget, kullanıcının farklı sekmeler veya ekranlar arasında geçiş yapmasını sağlar. BottomNavigationBar, her bir sekme için bir route tanımlar ve kullanıcı sekmeye tıkladığında o sekme ile ilişkilendirilmiş sayfaya yönlendirir.

BottomNavigationBar kullanarak yaptığım projeden bahsedeceğim. Öncelikle pubspec.yaml dosyasının içerisine gerekli bağımlılıkların eklenmesi gereklidir.

```yaml
dependencies:
  flutter:
    sdk: flutter

  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.8
  curved_navigation_bar: ^1.0.6

dev_dependencies:
  flutter_test:
    sdk: flutter
```

Şekil7: pubspec.yaml Dosyasındaki Eklentiler

curved_navigation_bar

Flutter için geliştirilmiş, alt kısmında yer alan ve yuvarlatılmış bir tasarıma sahip estetik bir navigasyon çubuğudur. Klasik BottomNavigationBar'a alternatif olarak kullanılır ve daha modern, dinamik bir görünüm sunar.

```dart
body: _pages[_selectedIndex],
bottomNavigationBar: CurvedNavigationBar(
  backgroundColor: Colors.transparent,
  color: Colors.white,
  buttonBackgroundColor: Colors.white,
  height: 60,
  index: _selectedIndex,
  items: _navigationItems,
  animationDuration: const Duration(milliseconds: 300),
  onTap: (index) {
    setState(() {
      _selectedIndex = index;
    });
  },
```

Şekil8: Sayfa Geçişlerinin Kontolü

Oluşturulan uygulamada, MyApp sınıfı uygulamanın ana çatısını oluşturur, MyHomePage ise navigasyonun kontrolünü sağlar. Bu yapıyla temel çok sayfalı Flutter uygulaması kolayca oluşturulabilir.

Sayfalar arasında geçiş, alt kısımda yer alan CurvedNavigationBar aracılığıyla yapılır. Kullanıcı, ikonlara tıkladıkça ilgili sayfa ekranda görüntülenir. Bu yapı, sade ve kullanıcı dostu bir gezinme deneyimi sunar.

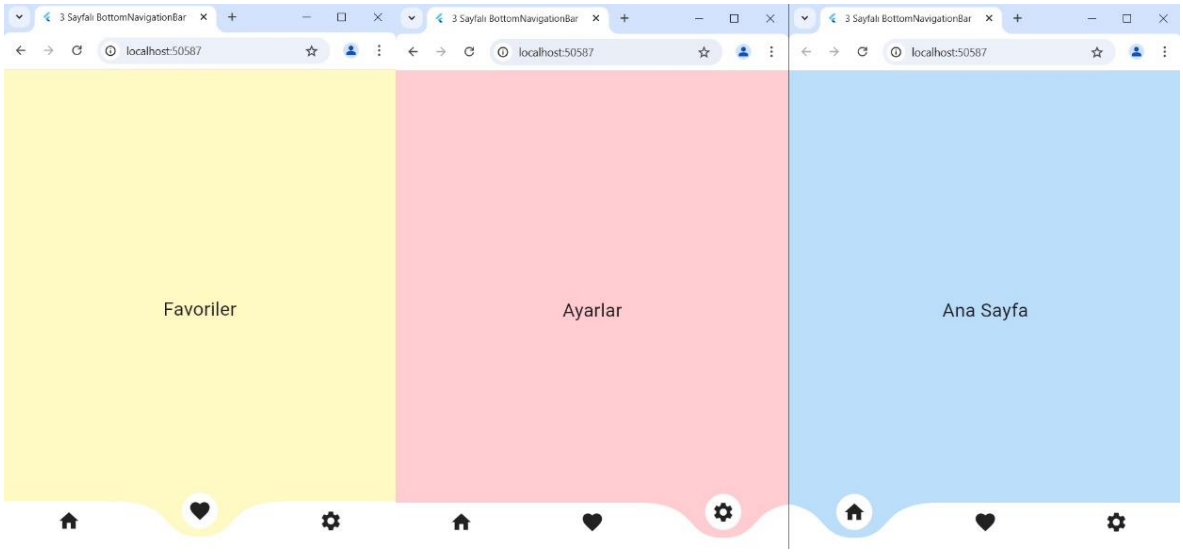
```dart
class FavoritePage extends StatelessWidget {
  const FavoritePage({super.key});

  @override
  Widget build(BuildContext context) {
    return Container(
      color: Colors.yellow[100],
      width: double.infinity,
      height: double.infinity,
      child: const Center(
        child: Text(
          'Favoriler',
          style: TextStyle(fontSize: 24),
```

Şekil9: Favoriler Sayfası

Bu Flutter projesi, üç sayfalı bir mobil uygulamayı temsil eder: Ana Sayfa, Favoriler ve Ayarlar. Sayfalar HomePage, FavoritePage ve SettingsPage olarak ayrı ayrı tanımlanmıştır. Her sayfa farklı bir renkte arka plana sahiptir ve ortasında büyük puntolu bir başlık yer alır.

Uygulamanın son hali aşağıdadır:



Şekil10: Projenin Gösterimi

Şekilde de görüldüğü üzere BottomNavigationBar, Flutter uygulamalarında kullanıcıların farklı sayfalar arasında hızlıca geçiş yapmasını sağlayan alt menü bileşenidir. Kullanım kolaylığı ve erişim hızı sağladığı için özellikle çok sayfalı uygulamalarda tercih edilir. İhtiyaca göre standart ya da özelleştirilmiş görünümlerle kullanılabilir.