Module 4: Navigation and Routing

Theory Assignments

Name: Chandvaniya Abhay

1. Explain how the Navigator widget works in Flutter.

1. Navigator as a Stack

- push() → Adds a new route to the top of the stack
- pop() → Removes the top route and returns to the previous one

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

2. Routes and BuildContext

- Navigator is accessed using a **BuildContext**
- Flutter searches up the widget tree to find the nearest Navigator
- Usually provided by MaterialApp or CupertinoApp

```
MaterialApp(
  home: HomePage(),
);
```

MaterialApp automatically creates a Navigator.

3. Passing Data Between Screens

Passing data when navigating forward

```
Navigator.push(
  context,
  MaterialPageRoute(
    builder: (context) => DetailsPage(id: 10),
  ),
);
```

Receiving data when popping

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

print(result);
Navigator.pop(context, "Success");
```

4. Named Routes

Instead of creating routes inline, you can define them by name.

```
MaterialApp(
  initialRoute: '/',
  routes: {
    '/': (context) => HomePage(),
    '/details': (context) => DetailsPage(),
  },
);
Navigator.pushNamed(context, '/details');
Navigator.pop(context);
```

Benefits:

- Cleaner code
- Centralized route management
- Easier navigation in large apps

5. Removing Routes

- **pushReplacement()** → Replaces current route
- **pushAndRemoveUntil()** → Clears routes until a condition is met

```
Navigator.pushReplacement(
  context,
  MaterialPageRoute(builder: (_) => LoginPage()),
);
Navigator.pushAndRemoveUntil(
  context,
  MaterialPageRoute(builder: (_) => HomePage()),
  (route) => false,
);
```

6. Navigator 1.0 vs Navigator 2.0 (Overview)

Navigator 1.0	Navigator 2.0
Imperative	Declarative
push() / pop()	Route state driven
Easier to learn	Better for deep linking

Navigator 2.0 uses the **Pages API** and is ideal for complex navigation scenarios like web URLs and deep linking.

2. Describe the concept of named routes and their advantages over direct route navigation.

Named Routes in Flutter

Named routes are a navigation approach where each screen (route) in an app is assigned a **string identifier**, and navigation is done using these names instead of directly creating route objects.

What Are Named Routes?

In named routing, routes are registered in one central place
(usually MaterialApp), and navigation happens using a route name.

Defining Named Routes

```
MaterialApp(
  initialRoute: '/',
  routes: {
    '/': (context) => HomePage(),
    '/details': (context) => DetailsPage(),
    '/profile': (context) => ProfilePage(),
  },
);
```

Navigating Using a Named Route

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

Going Back

```
Navigator.pop(context);
```

Direct Route Navigation (For Comparison)

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

Here, the route is created **inline**, tightly coupling navigation logic with UI code.

Advantages of Named Routes Over Direct Navigation

1. Centralized Route Management

- All routes are defined in one place
- Easier to update or modify navigation flow
- Improves maintainability in large applications

2. Cleaner and More Readable Code

Navigator.pushNamed(context, '/profile');

is simpler and more expressive than creating a MaterialPageRoute every time.

3. Better Scalability

- Ideal for apps with many screens
- Prevents repeated boilerplate route creation
- Makes navigation logic consistent across the app

4. Easier Navigation from Anywhere

- You can navigate without directly referencing widget classes
- Helpful for navigation from services, dialogs, or state management layers

5. Built-in Support for Navigation Features

Named routes integrate well with:

- **Deep linking**
- **Web URL navigation**
- **Navigator 2.0**
- **Route guards (onGenerateRoute)**

6. Route Arguments Support

You can still pass data while using named routes:

```
Navigator.pushNamed(
  context,
  '/details',
  arguments: 42,
);
final id = ModalRoute.of(context)!.settings.arguments as int;
```

When Direct Route Navigation Is Better

- Small apps or prototypes
- One-off screens
- When complex route arguments or custom transitions are needed

3. Explain how data can be passed between screens using route arguments.

Passing Data Using Route Arguments (Named Routes)

Step 1: Navigate and Pass Arguments

You pass data using the arguments parameter of Navigator.pushNamed().

```
Navigator.pushNamed(
  context,
  '/details',
  arguments: {
    'id': 101,
    'title': 'Flutter Basics',
  },
);
```

The argument can be **any object** (primitive, Map, or custom model).

Step 2: Receive Arguments in the Destination Screen

Retrieve the arguments using ModalRoute.

```
class DetailsPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final args =
        ModalRoute.of(context)!.settings.arguments as Map<String, dynamic>;

    return Scaffold(
      appBar: AppBar(title: Text(args['title'])),
      body: Text('ID: ${args['id']}'),
    );
  }
}
```

Passing a Custom Object

Model Example

```
class Product {
  final int id;
  final String name;

  Product(this.id, this.name);
}
```

Pass the Object

```
Navigator.pushNamed(
  context,
  '/product',
  arguments: Product(1, 'Laptop'),
);
```

Receive the Object

```
final product =
    ModalRoute.of(context)!.settings.arguments as Product;
```

Passing Data with Direct Route Navigation

Instead of route arguments, data is passed via the widget constructor.

```
Navigator.push(
  context,
  MaterialPageRoute(
    builder: (context) => DetailsPage(id: 10),
  ),
);
class DetailsPage extends StatelessWidget {
  final int id;
  DetailsPage({required this.id});
```

```
}
```

Returning Data Back to the Previous Screen

You can also **send data back** when popping a route.

```
final result = await Navigator.pushNamed(context, '/details');
print(result);
Navigator.pop(context, 'Success');
```

Advantages of Using Route Arguments

- Keeps navigation **decoupled** from UI widgets
- Works well with **named routes**
- Supports **deep linking**
- Allows passing **complex objects**