

Widget Lifecycle এবং BuildContext

ChatGPT said:

ChatGPT

Flutter Widget Lifecycle

Flutter এর **Widget Lifecycle** মূলত StatefulWidget এর সাথে সম্পর্কিত, কারণ StatelessWidget এর lifecycle খুব সরল। StatefulWidget এর দুইটি প্রধান অংশ আছে:

1. **StatefulWidget Class**: যেটা Immutable এবং State তৈরিতে কাজ করে।
 2. **State Class**: যেটা Mutable এবং Widget এর Dynamic Behavior পরিচালনা করে।
-

StatefulWidget এর Lifecycle ধাপসমূহ:

1. **createState()**
 - এই মেথডের মাধ্যমে State তৈরি হয়।
 - একবারই কল হয় যখন StatefulWidget প্রথমবার ইনস্ট্যান্স হয়।
 2. **initState()**
 - State এর Lifecycle শুরু হয় এই মেথড থেকে।
 - একবারই কল হয় এবং মূলত Widget এর initial configuration সেটআপে ব্যবহৃত হয়।
 - **Note**: এই মেথডে **super.initState()** অবশ্যই কল করতে হয়।
 3. **didChangeDependencies()**
 - যখন Widget এর InheritedWidget বা Context পরিবর্তন হয়, তখন এটা কল হয়।
 - এটা Lifecycle এর মাঝামাঝি পর্যায়ে বারবার কল হতে পারে।
 4. **build()**
 - UI তৈরির জন্য মূল মেথড।
 - Widget এর layout এবং structure এই মেথডে তৈরি হয়।
 5. **setState()**
 - State পরিবর্তনের সময় ব্যবহার করা হয়।
 - এটি build মেথডকে পুনরায় কল করে UI আপডেট করে।
 6. **didUpdateWidget()**
 - যদি StatefulWidget এর Configuration পরিবর্তন হয়, তখন এটি কল হয়।
 - পূর্ববর্তী Widget এর State ধরে রাখতে সাহায্য করে।
 7. **deactivate()**
 - Widget Tree থেকে একটি Widget সরানোর আগে এটি কল হয়।
 8. **dispose()**
 - Widget destroy করার সময় শেষবারের মতো এই মেথড কল হয়।
 - এখানে Resource release এবং Cleanup কাজ করা হয়।
 - **Note**: **super.dispose()** অবশ্যই কল করতে হয়।
-

BuildContext

BuildContext হলো একটি **Immutable Handle** যা Widget এর অবস্থান এবং সম্পর্কিত তথ্য সংরক্ষণ করে। এটি Widget Tree এর সাথে যোগাযোগ স্থাপন করে এবং Parent-Child সম্পর্ক বোঝাতে ব্যবহৃত হয়।

BuildContext এর কাজ:

1. **Widget Tree Navigating:**
 - `Navigator.of(context)` ব্যবহার করে স্ক্রিন Navigate করা যায়।
 2. **InheritedWidget Access:**
 - `context.dependOnInheritedWidgetOfExactType<>()` দিয়ে Parent Widget থেকে ডেটা Access করা যায়।
 3. **Theme এবং Media Query:**
 - `Theme.of(context)` দিয়ে Theme এর Data পাওয়া যায়।
 - `MediaQuery.of(context)` দিয়ে স্ক্রিনের সাইজ এবং Orientation জানা যায়।
-

উদাহরণ

StatefulWidget Lifecycle:

dart

Copy code

```
class MyStatefulWidget extends StatefulWidget {
  @override
  _MyStatefulWidgetState createState() => _MyStatefulWidgetState();
}

class _MyStatefulWidgetState extends State<MyStatefulWidget> {
  @override
  void initState() {
    super.initState();
    print("initState Called");
  }

  @override
  Widget build(BuildContext context) {
    print("build Called");
    return Scaffold(
      appBar: AppBar(title: Text("Lifecycle Example")),
      body: Center(child: Text("Hello, World!")),
    );
  }

  @override
  void dispose() {
```

```

        print("dispose Called");
        super.dispose();
    }
}

```

BuildContext উদাহরণ:

dart

Copy code

```

class MyWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final screenWidth = MediaQuery.of(context).size.width;
    return Container(
      width: screenWidth * 0.8,
      color: Colors.blue,
      child: Text(
        'Screen width: $screenWidth',
        style: Theme.of(context).textTheme.headline6,
      ),
    );
  }
}

```

40

You said:

Widget Lifecycle এবং BuildContext।

ChatGPT said:

ChatGPT

Widget Lifecycle

Flutter-এ **Widget Lifecycle** মূলত StatefulWidget এবং তার State-এর সঙ্গে সম্পর্কিত। StatelessWidget-এর কোনো lifecycle নেই কারণ এটি Immutable। StatefulWidget-এর ক্ষেত্রে, Widget-এর বিভিন্ন অবস্থায় কাজ করার জন্য lifecycle মেথডগুলো গুরুত্বপূর্ণ।

StatefulWidget-এর Lifecycle ধাপসমূহ

1. **createState()**
 - StatefulWidget প্রথমবার ইনস্ট্যান্স হওয়ার সময় createState() কল হয়।
 - এটি State অবজেক্ট তৈরি করে।
2. **initState()**

- State অবজেক্টের Lifecycle শুরু হয় `initState()` থেকে।
- একবারই কল হয় এবং সাধারণত Initializations (যেমন AnimationController সেটআপ) করতে ব্যবহৃত হয়।

dart

Copy code

```
@override
void initState() {
  super.initState();
  // Initialization code here
}
```

3.

4. `didChangeDependencies()`

- যখন Widget-এর Dependencies (InheritedWidget-এর Data) পরিবর্তিত হয়, তখন এটি কল হয়।
- `initState()`-এর পরে স্বয়ংক্রিয়ভাবে একবার কল হয়।

5. `build()`

- Widget-এর UI তৈরি করার জন্য এটি কল হয়।
- এই মেথড StatefulWidget-এর Core মেথড।
- Widget Tree প্রতিবার পুনর্নির্মাণ করার সময় `build()` কল হয়।

dart

Copy code

```
@override
Widget build(BuildContext context) {
  return Text("Hello, World!");
}
```

6.

7. `setState()`

- State পরিবর্তন করার জন্য এটি ব্যবহৃত হয়।
- এটি UI-কে Rebuild করে।

dart

Copy code

```
setState(() {
  // Update state variables
});
```

8.

9. `didUpdateWidget()`

- যদি Parent Widget এর Configurations পরিবর্তিত হয়, তখন এটি কল হয়।
- এর মাধ্যমে নতুন Widget-এর সঙ্গে পুরনো State মানিয়ে নেওয়া যায়।

10. `deactivate()`

- Widget Tree থেকে Widget সরানোর সময় এটি কল হয়।

11. `dispose()`

- Widget Destroy করার আগে Resource Cleanup-এর জন্য এটি কল হয়।

dart

Copy code

@override

```
void dispose() {  
  // Cleanup code here  
  super.dispose();  
}
```

12.

BuildContext

BuildContext হলো একটি Immutable অবজেক্ট যা Widget-এর Tree-এর মধ্যে তার অবস্থান এবং সম্পর্ক প্রকাশ করে। এটি Parent এবং Child Widget-এর মধ্যে যোগাযোগ স্থাপন করে।

BuildContext-এর কাজ

1. Widget Tree Navigating

- `Navigator.of(context)` ব্যবহার করে এক Widget থেকে অন্য Widget-এ Navigate করা যায়।

2. InheritedWidget Access

- `context.dependOnInheritedWidgetOfExactType<T>()` ব্যবহার করে Parent Widget-এর Data Access করা যায়।

3. Theme এবং MediaQuery Access

- `Theme.of(context)` দিয়ে Theme Data Access করা যায়।
- `MediaQuery.of(context)` দিয়ে স্ক্রিন সাইজ বা Orientation পাওয়া যায়।

উদাহরণ

StatefulWidget Lifecycle উদাহরণ

dart

Copy code

```
class MyStatefulWidget extends StatefulWidget {  
  @override  
  _MyStatefulWidgetState createState() => _MyStatefulWidgetState();  
}
```

```
class _MyStatefulWidgetState extends State<MyStatefulWidget> {  
  @override
```

```

void initState() {
  super.initState();
  print("initState Called");
}

@override
Widget build(BuildContext context) {
  print("build Called");
  return Scaffold(
    appBar: AppBar(title: Text("Lifecycle Example")),
    body: Center(child: Text("Hello, Lifecycle!")),
  );
}

@override
void dispose() {
  print("dispose Called");
  super.dispose();
}
}

```

BuildContext উদাহরণ

dart

Copy code

```

class MyWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final screenSize = MediaQuery.of(context).size;
    final theme = Theme.of(context);

    return Container(
      width: screenSize.width * 0.8,
      color: theme.primaryColor,
      child: Text(
        "Screen Width: ${screenSize.width}",
        style: theme.textTheme.headline6,
      ),
    );
  }
}

```

সংক্ষেপে

1. **Widget Lifecycle:**

- StatefulWidget-এর বিভিন্ন পর্যায়ে বিভিন্ন মেথড ব্যবহার করা হয়।
- `initState()`, `build()`, এবং `dispose()` প্রধান মেথড।

2. **BuildContext:**

- এটি Widget-এর Parent, Child এবং Tree সম্পর্কে তথ্য প্রদান করে।
- MediaQuery এবং Theme-এর মতো ফাংশনালিটি Access করতে ব্যবহৃত হয়।

40

You said:

Flutter Layouts (Row, Column, Stack, Container)।

ChatGPT said:

ChatGPT

Flutter Layouts: Row, Column, Stack, এবং Container

Flutter-এ **Layouts** হলো UI-র মৌলিক কাঠামো নির্মাণের উপায়। **Row**, **Column**, **Stack**, এবং **Container** হলো সবচেয়ে সাধারণ Layout Widgets। এদের সাহায্যে Widgets সাজানো যায় এবং তাদের সঠিক অবস্থান নির্ধারণ করা যায়।

1. Row

Row Widget ব্যবহার করে Widgets গুলি **Horizontally** সাজানো হয়।

প্রোপার্টিস:

- **mainAxisAlignment**: Horizontal alignment (e.g., center, start, end, spaceAround)।
- **crossAxisAlignment**: Vertical alignment (e.g., start, center, stretch)।
- **children**: একটি List যেখানে Row-এর ভিতর Widgets থাকে।

উদাহরণ:

dart

Copy code

```
Row(  
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
  crossAxisAlignment: CrossAxisAlignment.center,  
  children: [  
    Icon(Icons.star, color: Colors.red),  
    Icon(Icons.star, color: Colors.green),  
    Icon(Icons.star, color: Colors.blue),  
  ],  
)
```

2. Column

Column Widget ব্যবহার করে Widgets গুলি **Vertically** সাজানো হয়।

প্রোপারটিস:

- **mainAxisAlignment**: Vertical alignment।
- **crossAxisAlignment**: Horizontal alignment।
- **children**: একটি List যেখানে Column-এর ভিতর Widgets থাকে।

উদাহরণ:

dart

Copy code

```
Column(  
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
  crossAxisAlignment: CrossAxisAlignment.center,  
  children: [  
    Text("Hello, World!", style: TextStyle(fontSize: 20)),  
    ElevatedButton(onPressed: () {}, child: Text("Click Me")),  
    Icon(Icons.thumb_up, size: 50),  
  ],  
)
```

3. Stack

Stack Widget ব্যবহার করে Widgets একটির উপর আরেকটি **Overlay** করা হয়। এটি Z-axis-এ Widgets সাজাতে ব্যবহৃত হয়।

প্রোপারটিস:

- **alignment**: Stack-এর মধ্যে Widgets-এর অবস্থান নির্ধারণ করে।
- **children**: একাধিক Widgets ধারণ করে।

উদাহরণ:

dart

Copy code

```
Stack(  
  alignment: Alignment.center,  
  children: [  
    Container(width: 200, height: 200, color: Colors.blue),  
    Container(width: 150, height: 150, color: Colors.green),  
  ],  
)
```



```
    Container(width: 100, height: 100, color: Colors.red),  
  ],  
)
```

4. Container

Container হলো একটি মাল্টি-পারপাস Widget যা Padding, Margins, Borders, এবং Background Color সেটআপে ব্যবহৃত হয়।

প্রোপারটিস:

- **color**: Background color।
- **margin**: বাহিরের ফাঁকা জায়গা।
- **padding**: ভিতরের ফাঁকা জায়গা।
- **alignment**: Child Widget-এর অবস্থান নির্ধারণ করে।
- **width** এবং **height**: Container-এর সাইজ সেট করতে ব্যবহৃত হয়।

উদাহরণ:

dart

Copy code

```
Container(  
  width: 200,  
  height: 200,  
  alignment: Alignment.center,  
  margin: EdgeInsets.all(20),  
  padding: EdgeInsets.all(10),  
  decoration: BoxDecoration(  
    color: Colors.blue,  
    borderRadius: BorderRadius.circular(15),  
    boxShadow: [  
      BoxShadow(  
        color: Colors.black.withOpacity(0.2),  
        offset: Offset(5, 5),  
        blurRadius: 10,  
      ),  
    ],  
  ),  
  child: Text(  
    "Hello, Container!",  
    style: TextStyle(color: Colors.white, fontSize: 18),  
  ),  
)
```

Layouts এর মধ্যে পার্থক্য

Widget	Alignment Axis	Main Use Case
Row	Horizontal	একসারি Widgets সাজাতে।
Column	Vertical	একসারি Widgets উপরে-নিচে সাজাতে।
Stack	Z-axis	একটির উপর আরেকটি Widget রাখতে।
Container	Flexible	Styling বা একক Widget-এর জন্য।

উদাহরণ: Row, Column, এবং Stack একসঙ্গে

dart

Copy code

```
class MyLayoutExample extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Flutter Layouts Example")),
      body: Column(
        children: [
          Row(
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            children: [
              Icon(Icons.star, color: Colors.red),
              Icon(Icons.star, color: Colors.green),
              Icon(Icons.star, color: Colors.blue),
            ],
          ),
          Stack(
            alignment: Alignment.center,
            children: [
              Container(width: 200, height: 200, color:
Colors.yellow),
              Container(width: 150, height: 150, color:
Colors.orange),
              Text("Stack Example", style: TextStyle(fontSize: 18)),
            ],
          ),
        ],
      ),
    );
  }
}
```

```

    ),
    Container(
      width: double.infinity,
      height: 50,
      color: Colors.purple,
      alignment: Alignment.center,
      child: Text("Footer", style: TextStyle(color:
Colors.white))),
    ),
  ],
),
);
}
}

```

এটি চালালে আপনি Row, Stack এবং Container-এর একটি মিলিত উদাহরণ দেখতে পারবেন।

40

You said:

State Management (setState, Provider, Riverpod, Bloc)।

ChatGPT said:

ChatGPT

Flutter State Management: setState, Provider, Riverpod, এবং Bloc

Flutter অ্যাপ্লিকেশনে **State Management** হলো UI এবং Data-এর মধ্যে সম্পর্ক সঠিকভাবে বজায় রাখা। বিভিন্ন State Management টুলস ও পদ্ধতির মাধ্যমে আমরা Data পরিবর্তনের উপর ভিত্তি করে UI আপডেট করি। এখানে **setState**, **Provider**, **Riverpod**, এবং **Bloc** এর বিস্তারিত আলোচনা করা হলো।

1. setState

setState হলো Flutter-এর Built-in State Management পদ্ধতি, যা ছোট এবং সহজ অ্যাপের জন্য উপযুক্ত।

কিভাবে কাজ করে:

- **setState()** ফাংশন State আপডেট করে এবং **build()** মেথডকে পুনরায় কল করে।
- এটি শুধুমাত্র StatefulWidget-এর জন্য কার্যকর।

সুবিধা:

- সহজ এবং দ্রুত।
- ছোট প্রোজেক্টে ভালো কাজ করে।

অসুবিধা:

- বড় স্কেলে জটিল হয়ে যায়।
- State গুলো Centralized রাখা কঠিন।

উদাহরণ:

dart

Copy code

```
class CounterApp extends StatefulWidget {
  @override
  _CounterAppState createState() => _CounterAppState();
}

class _CounterAppState extends State<CounterApp> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("setState Example")),
      body: Center(
        child: Text("Counter: $_counter", style: TextStyle(fontSize:
20)),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        child: Icon(Icons.add),
      ),
    );
  }
}
```

2. Provider

Provider হলো Google দ্বারা সুপারিশকৃত State Management টুল, যা InheritedWidget-এর উপর ভিত্তি করে কাজ করে। এটি Dependency Injection-এর মাধ্যমে State Management সহজ করে।

কিভাবে কাজ করে:

- `ChangeNotifier` এবং `ChangeNotifierProvider` ব্যবহার করে State Management করা হয়।
- UI এবং State এর মধ্যে Listener হিসেবে কাজ করে।

সুবিধা:

- সহজ এবং জটিল অ্যাপের জন্য উপযুক্ত।
- Boilerplate কোড কম।

অসুবিধা:

- বড় অ্যাপ্লিকেশনে প্রচুর `ChangeNotifier` প্রয়োজন হতে পারে।

উদাহরণ:

dart

Copy code

```
// State Class
class CounterNotifier extends ChangeNotifier {
  int _counter = 0;

  int get counter => _counter;

  void increment() {
    _counter++;
    notifyListeners();
  }
}

// Main App
void main() {
  runApp(
    ChangeNotifierProvider(
      create: (_) => CounterNotifier(),
      child: MyApp(),
    ),
  );
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
```

```

        home: CounterScreen(),
    );
}
}

class CounterScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final counter = Provider.of<CounterNotifier>(context);

    return Scaffold(
      appBar: AppBar(title: Text("Provider Example")),
      body: Center(
        child: Text("Counter: ${counter.counter}", style:
TextStyle(fontSize: 20)),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: counter.increment,
        child: Icon(Icons.add),
      ),
    );
  }
}

```

3. Riverpod

Riverpod হলো Provider-এর একটি উন্নত সংস্করণ। এটি Compile-Time Safety, Scalability, এবং Simplicity নিয়ে আসে। এটি Functional Approach ব্যবহার করে।

কিভাবে কাজ করে:

- State তৈরি এবং ব্যবহারের জন্য **StateProvider** এবং **ConsumerWidget** ব্যবহার করা হয়।

সুবিধা:

- Compile-Time Errors ধরতে পারে।
- সহজ Unit Testing।
- Scalable এবং Flexible।

অসুবিধা:

- শিখতে কিছুটা সময় লাগে।
- নতুন Package হওয়ায় Community Support তুলনামূলক কম।

উদাহরণ:

dart

Copy code

```
final counterProvider = StateProvider<int>((ref) => 0);

class CounterApp extends ConsumerWidget {
  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final counter = ref.watch(counterProvider);

    return Scaffold(
      appBar: AppBar(title: Text("Riverpod Example")),
      body: Center(
        child: Text("Counter: $counter", style: TextStyle(fontSize:
20)),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () => ref.read(counterProvider.notifier).state++,
        child: Icon(Icons.add),
      ),
    );
  }
}

void main() {
  runApp(ProviderScope(child: MaterialApp(home: CounterApp())));
}
```

4. Bloc (Business Logic Component)

Bloc হলো Event-Driven State Management প্যাটার্ন, যা Reactive Programming ব্যবহার করে। এটি বড় স্কেলের অ্যাপ্লিকেশনের জন্য আদর্শ।

কিভাবে কাজ করে:

- Events এবং States-এর মাধ্যমে Data এবং UI পরিচালনা করা হয়।
- **bloc** এবং **flutter_bloc** প্যাকেজ ব্যবহার করে কাজ করা হয়।

সুবিধা:

- জটিল অ্যাপ্লিকেশনের জন্য উপযুক্ত।
- Clear Separation of Concerns।
- Reactive Architecture।

অসুবিধা:

- শিখতে সময় লাগে।
- কোড কিছুটা জটিল হতে পারে।

উদাহরণ:

dart

Copy code

```
// Bloc Class
class CounterBloc extends Bloc<CounterEvent, int> {
  CounterBloc() : super(0);

  @override
  Stream<int> mapEventToState(CounterEvent event) async* {
    if (event is IncrementEvent) {
      yield state + 1;
    }
  }
}

// Events
abstract class CounterEvent {}
class IncrementEvent extends CounterEvent {}

// UI
class CounterApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return BlocProvider(
      create: (context) => CounterBloc(),
      child: Scaffold(
        appBar: AppBar(title: Text("Bloc Example")),
        body: BlocBuilder<CounterBloc, int>(
          builder: (context, count) {
            return Center(
              child: Text("Counter: $count", style:
TextStyle(fontSize: 20)),
            );
          },
        ),
        floatingActionButton: FloatingActionButton(
          onPressed: () =>
            context.read<CounterBloc>().add(IncrementEvent()),
        ),
      ),
    );
  }
}
```



```

        child: Icon(Icons.add),
      ),
    ),
  );
}

void main() {
  runApp(MaterialApp(home: CounterApp()));
}

```

তুলনামূলক বিশ্লেষণ

State Management Tool	উপযুক্ত ক্ষেত্রে	সুবিধা	অসুবিধা
setState	ছোট এবং সহজ অ্যাপ	সহজ এবং দ্রুত	বড় অ্যাপের জন্য জটিল
Provider	মাঝারি আকারের অ্যাপ	সহজ এবং পরিষ্কার	বড় স্কেলে Maintenance কঠিন
Riverpod	বড় স্কেলের অ্যাপ, Compile-Time Safety প্রয়োজন	Flexible, Testable	শিখতে কিছুটা সময় লাগে
Bloc	জটিল ও বড় আকারের অ্যাপ	Reactive এবং Scalable	শিখতে এবং ইমপ্লিমেন্ট করতে সময় লাগে

কোনটি ব্যবহার করবেন?

- ছোট অ্যাপ: **setState** বা **Provider**।
- মাঝারি অ্যাপ: **Provider** বা **Riverpod**।
- বড় এবং জটিল অ্যাপ: **Riverpod** বা **Bloc**।

40

You said:

Navigation এবং Routing (Named Routes, GoRouter)।

ChatGPT said:

ChatGPT

Navigation এবং Routing: Named Routes এবং GoRouter

Flutter-এ **Navigation** এবং **Routing** হলো একটি Widget থেকে অন্য Widget-এ যাওয়ার প্রক্রিয়া। অ্যাপের বিভিন্ন স্ক্রিনের মধ্যে পরিচালনা করার জন্য আমরা **Navigator** এবং **Routing** পদ্ধতি ব্যবহার করি। **Named Routes** এবং **GoRouter** হলো Flutter-এ Routing-এর দুটি সাধারণ পদ্ধতি।

1. Named Routes

Named Routes ব্যবহার করলে অ্যাপের স্ক্রিনগুলোর জন্য নির্দিষ্ট নাম দেওয়া হয়। এটি বড় অ্যাপের Navigation সহজ করে।

কিভাবে কাজ করে:

- Routes Mapping: একটি Map-এর মাধ্যমে Route এবং তার Widget Define করা হয়।
- Navigator: `Navigator.pushNamed()` এবং `Navigator.pop()` এর মাধ্যমে স্ক্রিন Navigate করা হয়।

উদাহরণ:

dart

Copy code

```
// Main Function
void main() {
  runApp(MaterialApp(
    initialRoute: '/',
    routes: {
      '/': (context) => HomeScreen(),
      '/details': (context) => DetailsScreen(),
    },
  ));
}

// Home Screen
class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Home Screen")),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            Navigator.pushNamed(context, '/details');
          },
          child: Text("Go to Details"),
        ),
      ),
    );
  }
}
```

```

        ),
    );
}
}

// Details Screen
class DetailsScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Details Screen")),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            Navigator.pop(context);
          },
          child: Text("Go Back"),
        ),
      ),
    );
  }
}

```

সুবিধা:

- সহজ এবং সরাসরি।
- স্ক্রিনের নাম মনে রাখা সহজ।

অসুবিধা:

- ডায়নামিক প্যারামিটার হ্যান্ডলিং জটিল।
- বড় অ্যাপ্লিকেশনের জন্য সীমিত।

2. GoRouter

GoRouter হলো Flutter-এর জন্য একটি আধুনিক এবং শক্তিশালী Routing প্যাকেজ। এটি Declarative Routing মডেল এবং URL-based Navigation সমর্থন করে।

কিভাবে কাজ করে:

- Route Define করা হয় **GoRoute** এর মাধ্যমে।
- URL-based Navigation সম্ভব।

প্যাকেজ ইনস্টলেশন:

প্রথমে `pubspec.yaml`-এ **go_router** প্যাকেজ যোগ করুন:

yaml

Copy code

```
dependencies:  
  go_router: ^7.1.0
```

উদাহরণ:

dart

Copy code

```
import 'package:flutter/material.dart';  
import 'package:go_router/go_router.dart';  
  
void main() {  
  runApp(MyApp());  
}  
  
// Define GoRouter  
final GoRouter _router = GoRouter(  
  routes: [  
    GoRoute(  
      path: '/',  
      builder: (context, state) => HomeScreen(),  
    ),  
    GoRoute(  
      path: '/details',  
      builder: (context, state) => DetailsScreen(),  
    ),  
  ],  
);  
  
class MyApp extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp.router(  
      routerConfig: _router,  
    );  
  }  
}
```

```

// Home Screen
class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Home Screen")),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            context.go('/details');
          },
          child: Text("Go to Details"),
        ),
      ),
    );
  }
}

// Details Screen
class DetailsScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Details Screen")),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            context.pop();
          },
          child: Text("Go Back"),
        ),
      ),
    );
  }
}

```

সুবিধা:

- URL-based Routing।
- Nested Routes এবং Dynamic Parameters সহজ।
- Declarative এবং Scalable।

অসুবিধা:

- নতুন প্যাকেজ হওয়ায় শিখতে সময় লাগে।
- ছোট অ্যাপের জন্য Overkill হতে পারে।

তুলনামূলক বিশ্লেষণ

পদ্ধতি	উপযুক্ত ক্ষেত্রে	সুবিধা	অসুবিধা
Named Routes	ছোট বা মাঝারি অ্যাপ	সহজ এবং দ্রুত	ডাইনামিক প্যারামিটার হ্যান্ডলিং কঠিন।
GoRouter	বড় এবং জটিল অ্যাপ	URL-based Routing এবং Nested Routes।	শিখতে সময় লাগে এবং Config-heavy।

Dynamic Parameters উদাহরণ

Named Routes:

dart

Copy code

```
// Routes Mapping
routes: {
  '/details/:id': (context) => DetailsScreen(),
},

// Navigating
Navigator.pushNamed(context, '/details/123');
```

GoRouter:

dart

Copy code

```
final GoRouter _router = GoRouter(
  routes: [
    GoRoute(
      path: '/details/:id',
      builder: (context, state) {
        final id = state.params['id'];
        return DetailsScreen(id: id);
      },
    ),
  ],
);
```

```
);
```

```
// Navigating
```

```
context.go('/details/123');
```

কোনটি ব্যবহার করবেন?

1. ছোট বা মাঝারি অ্যাপ:
 - **Named Routes** সহজ এবং কার্যকর।
2. বড় ও জটিল অ্যাপ:
 - **GoRouter** Nested Routes, Dynamic Parameters, এবং URL-based Routing এর জন্য আদর্শ।

Dart-এ Map (মানচিত্র)

Map হলো একটি ডেটা স্ট্রাকচার, যেখানে **Key-Value** আকারে ডেটা সংরক্ষণ করা হয়।

প্রত্যেকটি **Key** ইউনিক হয়, এবং তার সাথে একটি **Value** থাকে। এটি অন্যান্য প্রোগ্রামিং ভাষার Dictionary বা HashMap এর মতো।

Map তৈরির উপায়

1. Literal Syntax ব্যবহার

dart

Copy code

```
void main() {  
  Map<String, String> countries = {  
    'BD': 'Bangladesh',  
    'US': 'United States',  
    'IN': 'India',  
  };  
  print(countries); // Output: {BD: Bangladesh, US: United States,  
IN: India}  
}
```

2. Constructor ব্যবহার

dart

Copy code

```
void main() {
```

```
Map<int, String> numbers = Map();
numbers[1] = 'One';
numbers[2] = 'Two';
print(numbers); // Output: {1: One, 2: Two}
}
```

Map-এর সাধারণ অপারেশন

1. মান যোগ করা

dart

Copy code

```
void main() {
  Map<String, int> ages = {};
  ages['Alice'] = 25;
  ages['Bob'] = 30;
  print(ages); // Output: {Alice: 25, Bob: 30}
}
```

2. মান পড়া

dart

Copy code

```
void main() {
  Map<String, String> fruits = {'A': 'Apple', 'B': 'Banana'};
  print(fruits['A']); // Output: Apple
}
```

3. মান পরিবর্তন করা

dart

Copy code

```
void main() {
  Map<String, String> capitals = {'BD': 'Dhaka', 'IN': 'Delhi'};
  capitals['BD'] = 'Chittagong';
  print(capitals); // Output: {BD: Chittagong, IN: Delhi}
}
```

4. মান মুছে ফেলা

dart

Copy code

```
void main() {
```



```
Map<String, String> fruits = {'A': 'Apple', 'B': 'Banana'};
fruits.remove('A');
print(fruits); // Output: {B: Banana}
}
```

Map এর গুরুত্বপূর্ণ পদ্ধতি

পদ্ধতি	কাজ
addAll()	একাধিক মান যোগ করা।
remove(key)	নির্দিষ্ট Key মুছে ফেলা।
clear()	পুরো Map পরিষ্কার করা।
containsKey(key)	কোনো Key আছে কি না তা যাচাই করা।
containsValue(value))	কোনো Value আছে কি না তা যাচাই করা।
forEach()	প্রত্যেক Key-Value জোড়ায় লুপ চালানো।
keys	সব Keys-এর তালিকা।
values	সব Values-এর তালিকা।

Map-এর উপর লুপ চালানো

1. forEach() ব্যবহার

dart

Copy code

```
void main() {
  Map<String, String> fruits = {'A': 'Apple', 'B': 'Banana'};
  fruits.forEach((key, value) {
    print('$key: $value');
  });
}
// Output:
// A: Apple
// B: Banana
```

2. for-in লুপ ব্যবহার

dart

Copy code

```
void main() {  
  Map<String, int> scores = {'Alice': 90, 'Bob': 80};  
  for (var key in scores.keys) {  
    print('$key: ${scores[key]}');  
  }  
}  
// Output:  
// Alice: 90  
// Bob: 80
```

Spread Operator (...), null-aware Spread Operator (...?)

1. Spread Operator

dart

Copy code

```
void main() {  
  Map<String, int> map1 = {'A': 1, 'B': 2};  
  Map<String, int> map2 = {'C': 3, ...map1};  
  print(map2); // Output: {C: 3, A: 1, B: 2}  
}
```

2. null-aware Spread Operator

dart

Copy code

```
void main() {  
  Map<String, int>? nullableMap;  
  Map<String, int> map = {'X': 10, ...?nullableMap};  
  print(map); // Output: {X: 10}  
}
```

Map-এ ফিল্টারিং এবং ম্যাপিং

1. where() ব্যবহার

dart

Copy code

```
void main() {  
  Map<String, int> scores = {'Alice': 90, 'Bob': 70, 'Charlie': 85};
```

```
    var highScores = scores.entries.where((entry) => entry.value > 80);  
    print(highScores); // Output: (MapEntry(Alice: 90),  
MapEntry(Charlie: 85))  
}
```

2. map() ব্যবহার

dart

Copy code

```
void main() {  
    Map<String, int> scores = {'Alice': 90, 'Bob': 80};  
    var updatedScores = scores.map((key, value) => MapEntry(key, value  
+ 10));  
    print(updatedScores); // Output: {Alice: 100, Bob: 90}  
}
```

Key এবং Value Access

1. keys ব্যবহার

dart

Copy code

```
void main() {  
    Map<String, String> fruits = {'A': 'Apple', 'B': 'Banana'};  
    print(fruits.keys); // Output: (A, B)  
}
```

2. values ব্যবহার

dart

Copy code

```
void main() {  
    Map<String, String> fruits = {'A': 'Apple', 'B': 'Banana'};  
    print(fruits.values); // Output: (Apple, Banana)  
}
```

Map থেকে List তৈরি উদাহরণ

1. Keys এবং Values থেকে List

dart

Copy code

```
void main() {  
    Map<String, int> scores = {'Alice': 90, 'Bob': 80};  
    var keys = scores.keys.toList();  
    var values = scores.values.toList();  
    print(keys); // Output: [Alice, Bob]  
    print(values); // Output: [90, 80]  
}
```

2. Entries থেকে List

dart

Copy code

```
void main() {  
    Map<String, int> scores = {'Alice': 90, 'Bob': 80};  
    var entries = scores.entries.toList();  
    print(entries); // Output: [MapEntry(Alice: 90), MapEntry(Bob:  
80)]  
}
```

উপসংহার:

- **Map** হলো একটি শক্তিশালী ডেটা স্ট্রাকচার, যা Key-Value পদ্ধতিতে ডেটা সংরক্ষণ করে।
- এটি ছোট এবং বড় অ্যাপ্লিকেশনে তথ্য সংরক্ষণ এবং পরিচালনার জন্য অত্যন্ত উপযোগী।
- বিভিন্ন পদ্ধতি ব্যবহার করে Map সহজে Manipulate করা যায়।