Level 2: Intermediate Concepts

1. State Management

- **Provider**:
  - Provider is a popular state management library in Flutter. It allows you to manage state and dependency injection in a simple and efficient manner.
  - **Example**:

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';

void main() {
  runApp(
    ChangeNotifierProvider(
      create: (context) => Counter(),
      child: MyApp(),
    ),
  );
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: Center(child: CounterText()),
        floatingActionButton: FloatingActionButton(
          onPressed: () => context.read<Counter>().increment(),
          child: Icon(Icons.add),
        ),
      ),
    );
  }
}

class Counter extends ChangeNotifier {
  int _count = 0;

  int get count => _count;

  void increment() {
    _count++;
```

```
    notifyListeners();
  }
}

class CounterText extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Text('Counter: ${context.watch<Counter>().count}');
  }
}
```

- **Bloc**:
  - Bloc is a state management library that separates business logic from the UI using Streams.
- **Riverpod**:
  - Riverpod is an improvement over the Provider package. It provides a robust solution for state management, allowing more control and flexibility.
- **Getx**:
  - Getx is an all-in-one Flutter package that provides state management, dependency injection, and route management.

2. Networking

- **Serverless**:
  - Serverless refers to a cloud computing model where developers focus on writing code without managing infrastructure. Firebase is a popular serverless solution.
  - **Example with Firebase**:

```
import 'package:firebase_core/firebase_core.dart';
import 'package:firebase_database/firebase_database.dart';
import 'package:flutter/material.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('Firebase Example')),
        body: Center(child: FirebaseData()),
      ),
```

```
    );
  }
}

class FirebaseData extends StatefulWidget {
  @override
  _FirebaseDataState createState() => _FirebaseDataState();
}

class _FirebaseDataState extends State<FirebaseData> {
  final DatabaseReference _dbRef =
FirebaseDatabase.instance.reference().child('data');
  String _data = 'Loading...';

  @override
  void initState() {
    super.initState();
    _dbRef.once().then((DataSnapshot snapshot) {
      setState(() {
        _data = snapshot.value.toString();
      });
    });
  }

  @override
  Widget build(BuildContext context) {
    return Text(_data);
  }
}
```

- **Servers**:
  - Managing backend servers, building APIs, and interacting with them from the Flutter app.
- **Restful API**:
  - RESTful APIs are used to perform CRUD (Create, Read, Update, Delete) operations over HTTP. In Flutter, you can use the `http` package to interact with REST APIs.
  - **Example**:

```
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';

void main() => runApp(MyApp());

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

```
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('REST API Example')),
        body: Center(child: RestApiData()),
      ),
    );
  }
}

class RestApiData extends StatefulWidget {
  @override
  _RestApiDataState createState() => _RestApiDataState();
}

class _RestApiDataState extends State<RestApiData> {
  String _data = 'Loading...';

  @override
  void initState() {
    super.initState();
    fetchData();
  }

  Future<void> fetchData() async {
    final response = await
http.get(Uri.parse('https://jsonplaceholder.typicode.com/posts/1'));

    if (response.statusCode == 200) {
      setState(() {
        _data = jsonDecode(response.body)['title'];
      });
    } else {
      setState(() {
        _data = 'Failed to load data';
      });
    }
  }

  @override
  Widget build(BuildContext context) {
    return Text(_data);
  }
}
```

- **GraphQL**:
  - GraphQL is an alternative to REST, providing more flexibility by allowing clients to request only the data they need. The `graphql_flutter` package can be used to integrate GraphQL in Flutter apps.
  - **Example**:

```dart
import 'package:flutter/material.dart';
import 'package:graphql_flutter/graphql_flutter.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('GraphQL Example')),
        body: Center(child: GraphQLData()),
      ),
    );
  }
}

class GraphQLData extends StatefulWidget {
  @override
  _GraphQLDataState createState() => _GraphQLDataState();
}

class _GraphQLDataState extends State<GraphQLData> {
  final HttpLink _httpLink = HttpLink(
    'https://api.spacex.land/graphql/',
  );

  ValueNotifier<GraphQLClient> _client;

  @override
  void initState() {
    super.initState();
    _client = ValueNotifier(
      GraphQLClient(
        cache: GraphQLCache(),
        link: _httpLink,
      ),
    );
  }

  @override
  Widget build(BuildContext context) {
    return GraphQLProvider(
      client: _client,
      child: Query(
        options: QueryOptions(
          document: gql(
            r'''
            query Launches {
              launchesPast(limit: 5) {
                mission_name
                launch_date_utc
                launch_site {
                  site_name_long
```

```
                }
              }
            }
            ''',
          ),
        ),
        builder: (QueryResult result, {refetch, fetchMore}) {
          if (result.hasException) {
            return Text(result.exception.toString());
          }

          if (result.isLoading) {
            return Text('Loading...');
          }

          final launches = result.data['launchesPast'];

          return ListView.builder(
            itemCount: launches.length,
            itemBuilder: (context, index) {
              final launch = launches[index];
              return ListTile(
                title: Text(launch['mission_name']),
                subtitle: Text(launch['launch_site']['site_name_long']),
              );
            },
          );
        },
      ),
    );
  }
}
```

- **JSON**:
  - JSON (JavaScript Object Notation) is a lightweight data-interchange format. In Flutter, you often use JSON to structure data when communicating with APIs. The `dart:convert` library is used to parse JSON.
  - **Example**:

```
import 'dart:convert';

void main() {
  String jsonString = '{"name": "Flutter", "type": "Framework"}';
  Map<String, dynamic> jsonMap = jsonDecode(jsonString);
  print('Name: ${jsonMap['name']}, Type: ${jsonMap['type']}');
}
```

3. Animations

- **Animated Widgets**:
  - Flutter provides built-in widgets like `AnimatedContainer`, `AnimatedOpacity`, and more, which automatically animate changes in their properties.
  - **Example**:

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: Center(child: AnimatedBox()),
      ),
    );
  }
}

class AnimatedBox extends StatefulWidget {
  @override
  _AnimatedBoxState createState() => _AnimatedBoxState();
}

class _AnimatedBoxState extends State<AnimatedBox> {
  double _size = 100;

  void _toggleSize() {
    setState(() {
      _size = _size == 100 ? 200 : 100;
    });
  }

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: _toggleSize,
      child: AnimatedContainer(
        duration: Duration(seconds: 1),
        width: _size,
        height: _size,
        color: Colors.blue,
      ),
    );
  }
}
```

- **Animation Builders**:

- `AnimatedBuilder` is used to create complex animations that require more control. It separates the animation logic from the widget tree, allowing for efficient re-use of animation code.

- **Curved Animations**:
  - Flutter provides various curves (e.g., `Curves.easeIn`, `Curves.bounceOut`) to define non-linear animations. `CurvedAnimation` is often used in conjunction with `Tween` to define the interpolation.
  - **Example**:

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: Center(child: CurvedAnimationExample()),
      ),
    );
  }
}

class CurvedAnimationExample extends StatefulWidget {
  @override
  _CurvedAnimationExampleState createState() =>
_CurvedAnimationExampleState();
}

class _CurvedAnimationExampleState extends State<CurvedAnimationExample>
with SingleTickerProviderStateMixin {
  AnimationController _controller;
  Animation<double> _animation;

  @override
  void initState() {
    super.initState();
    _controller = AnimationController(vsync: this, duration:
Duration(seconds: 2));
    _animation = CurvedAnimation(parent: _controller, curve:
Curves.bounceOut);

    _controller.repeat(reverse: true);
  }

  @override
  void dispose() {
    _controller.dispose();
    super.dispose();
```

```
  }

  @override
  Widget build(BuildContext context) {
    return AnimatedBuilder(
      animation: _animation,
      builder: (context, child) {
        return Transform.scale(
          scale: _animation.value,
          child: child,
        );
      },
      child: Container(width: 100, height: 100, color: Colors.red),
    );
  }
}
```

- **Hero Animations**:
  - Hero animations provide seamless transitions between different screens, making it look like an element is flying from one screen to another.
  - **Example**:

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: FirstScreen(),
    );
  }
}

class FirstScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('First Screen')),
      body: Center(
        child: GestureDetector(
          onTap: () {
            Navigator.push(context, MaterialPageRoute(builder: (_) =>
SecondScreen()));
          },
          child: Hero(
            tag: 'hero-image',
            child: Container(
              width: 100,
              height: 100,
              color: Colors.blue,
```

```
            ),
          ),
        ),
      ),
    );
  }
}

class SecondScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Second Screen')),
      body: Center(
        child: Hero(
          tag: 'hero-image',
          child: Container(
            width: 200,
            height: 200,
            color: Colors.red,
          ),
        ),
      ),
    );
  }
}
```