Flutter 데이터 자산 처리

데이터 처리와 관리 기법

학습목표

- 앱 데이터 리팩토링 및 효율적인 데이터 관리 방법 습득
- JSON 파일의 로컬/원격 데이터 처리 방법 이해
- Future를 활용한 비동기 데이터 처리 기법 습득

데이터 전략 수립

- 데이터 위치에 따른 액세스 전략 수립 필요
- 데이터 양과 업데이트 빈도를 고려한 저장소 선택
- 내장/로컬/원격 데이터의 특성 이해와 활용
- 데이터 파이프라인 구축을 통한 효율적 관리

내장 데이터 리팩토링

- 데이터 클래스를 분리하여 코드 가독성 향상
- 확장성을 고려한 데이터 구조 설계
- 재사용 가능한 데이터 관리 구조 구현
- 유지보수가 용이한 코드 아키텍처 구성

MyData.dart

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

class MyData {
  final List<String> items = [
    'January', 'February', 'March', 'April', 'May',
    'June', 'July', 'August', 'September', 'October',
    'November', 'December'
  ];

  MyData();
}
```

MyApp.dart

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

class MyApp extends StatelessWidget {
  MyApp({Key? key}) : super(key: key);
  final MyData data = MyData();

  @override
  Widget build(BuildContext context) {
    const title = 'MyAwesomeApp';
    List items = data.items;

    return MaterialApp(
      title: title,
      home: Scaffold(
        appBar: AppBar(
          title: const Text(title),
        ),
        body: ListView.builder(
          itemCount: items.length,
          itemBuilder: (context, index) {
            return ListTile(
              title: Text(items[index]),
            );
          },
        ),
      ),
    );
  }
}
```

JSON 데이터 모델

- JSON 데이터를 Dart 클래스로 자동 변환
- 단일/다중 계층 JSON 구조 처리
- fromJson/toJson 메서드를 통한 직렬화
- 데이터 모델의 타입 안정성 확보

DataModel.dart

```dart
class Month {
  List<Data>? data;

  Month({this.data});

  Month.fromJson(Map<String, dynamic> json) {
    if (json['data'] != null) {
      data = <Data>[];
      json['data'].forEach((v) {
        data!.add(new Data.fromJson(v));
      });
    }
  }

  Map<String, dynamic> toJson() {
    final Map<String, dynamic> data = new Map<String, dynamic>();
    if (this.data != null) {
      data['data'] = this.data!.map((v) => v.toJson()).toList();
    }
    return data;
  }
}
```

로컬 JSON 데이터 처리

- Future와 FutureBuilder를 활용한 비동기 데이터 처리
- JSON 디코딩과 모델 변환 작업
- 상태 관리를 통한 데이터 로딩 처리
- 에러 핸들링과 로딩 상태 표시

LocalJsonService.dart

```dart
import 'dart:convert';

class LocalJsonService {
  Future<String> _loadLocalData() async {
    final MyData data = MyData();
    return data.items;
  }

  Future<DataSeries> fetchData() async {
    String jsonString = await _loadLocalData();
    final jsonResponse = json.decode(jsonString);
    return DataSeries.fromJson(jsonResponse);
  }
}
```

JsonDataWidget.dart

```dart
class JsonDataWidget extends StatefulWidget {
  const JsonDataWidget({Key? key}) : super(key: key);

  @override
  State<JsonDataWidget> createState() => _JsonDataWidgetState();
}

class _JsonDataWidgetState extends State<JsonDataWidget> {
  late Future<DataSeries> dataSeries;

  @override
  void initState() {
    super.initState();
    final service = LocalJsonService();
    dataSeries = service.fetchData();
  }

  @override
  Widget build(BuildContext context) {
    return FutureBuilder<DataSeries>(
      future: dataSeries,
      builder: (context, snapshot) {
        if (snapshot.hasData) {
          return ListView.builder(
            itemCount: snapshot.data!.dataModel.length,
            itemBuilder: (context, index) {
              return ListTile(
                title: Text(snapshot.data!.dataModel[index].title),
              );
            },
          );
        } else if (snapshot.hasError) {
          return Text("Error: ${snapshot.error}");
        }
        return const CircularProgressIndicator();
      },
    );
  }
}
```

Assets JSON 데이터 처리

- pubspec.yaml 설정을 통한 assets 등록
- rootBundle을 이용한 assets 파일 접근
- 비동기 데이터 로딩과 상태 관리
- 에러 처리와 예외 상황 대응

AssetLoader.dart

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

class AssetLoader {
  static Future<String> loadJsonData(String path) async {
    try {
      return await rootBundle.loadString(path);
    } catch (e) {
      throw Exception('Failed to load asset: $path');
    }
  }
}
```

AssetJsonWidget.dart

```dart
class AssetJsonWidget extends StatefulWidget {
  const AssetJsonWidget({Key? key}) : super(key: key);

  @override
  State<AssetJsonWidget> createState() => _AssetJsonWidgetState();
}

class _AssetJsonWidgetState extends State<AssetJsonWidget> {
  late Future<DataSeries> dataSeries;

  @override
  void initState() {
    super.initState();
    dataSeries = _loadData();
  }

  Future<DataSeries> _loadData() async {
    final jsonString = await AssetLoader.loadJsonData('assets/data.json');
    final jsonResponse = json.decode(jsonString);
    return DataSeries.fromJson(jsonResponse);
  }

  @override
  Widget build(BuildContext context) {
    return FutureBuilder<DataSeries>(
      future: dataSeries,
      builder: (context, snapshot) {
        // 이전 코드와 동일한 builder 로직
      },
    );
  }
}
```

네트워크 통신 처리

- HTTP 클라이언트를 이용한 원격 데이터 접근
- 네트워크 상태 모니터링과 에러 처리
- 오프라인 대응 전략 구현
- 캐싱과 데이터 동기화 관리

NetworkService.dart

```dart
import 'package:http/http.dart' as http;
import 'package:connectivity_plus/connectivity_plus.dart';

class NetworkService {
  static Future<bool> checkConnectivity() async {
    var connectivityResult = await Connectivity().checkConnectivity();
    return connectivityResult != ConnectivityResult.none;
  }

  static Future<String> fetchData(String url) async {
    try {
      final response = await http.get(Uri.parse(url));
      if (response.statusCode == 200) {
        return response.body;
      } else {
        throw Exception('Failed to load data');
      }
    } catch (e) {
      throw Exception('Network error: $e');
    }
  }
}
```

요약

- Flutter에서 다양한 데이터 소스를 효과적으로 관리하는 방법을 학습
- JSON 데이터의 파싱과 모델 변환을 통한 타입 안정성 확보
- 비동기 처리와 상태 관리를 통한 안정적인 데이터 핸들링 구현

END