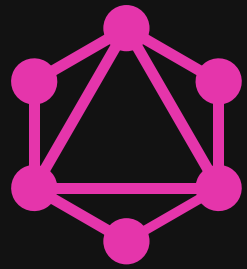
Flutter × GraphQLでアプリを作ってみる



Press Space for next page →

自己紹介

- 📝 飯野陽平（wheatandcat）
- 🏢 会社: 合同会社UNICORN 代表社員
- 📚 Blog: https://www.wheatandcat.me/
- 🛠️ 今までに作ったもの
  - memoir
  - OOMAKA
  - MarkyLinky

Flutterとは？

- Google開発のオープンソースのマルチプラットフォームの開発フレームワーク
- 一つのコードベースから、**iOS、Android、Web、Windows、Mac、Linuxアプリ**など複数のプラットフォームの作成が可能
- 言語は**Dart**を使用
- Google独自のUI Widgetを使用（Material Designベース）
    - なので、プラットフォーム毎でUIに差分が発生しづらい

今回の開発物

- まだ開発中だが、家の在庫を管理するアプリを作成中
- 上記の画像のような画面構成にする予定

現在の開発構成

- Frontend
  - Flutter
- Backend
  - NestJS
  - Prisma
  - PlanetScale
  - GraphQL Code Generator
  - Cloud Run

開発構成のモチベーション

- 普段はReact Nativeでアプリ開発をしているので、よく比較対象に挙がる **Flutter** で本格的な開発をしてみたかった
- NestJSの採用の一番大きい理由は、**Prisma**を使って開発をしたかったから
  - 前に試したT3 Stackでの開発体験が良かったので、他の構成でも使ってみたかった
  - tRPCが、もっとも望ましかったが、Flutterではサポートされていないので、**GraphQL**を採用
- NestJSは**BFF**の構成にした場合に、採用されるけーすが多いので、触ってみたかった
- Flutterでもgraphql_codegenで型安全で実装はできるので、それを使って開発
- 各比較についてはスライドの最後に記載

Flutterの開発

基本的なコードの書き方は以下のような感じ

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

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Hello Flutter'),
        ),
        body: Center(
          child: Text('Welcome to Flutter'),
        ),
      ),
    );
  }
}
```

Flutterの開発

基本的なコードの書き方は以下のような感じ

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

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Hello Flutter'),
        ),
        body: Center(
          child: Text('Welcome to Flutter'),
        ),
      ),
    );
  }
}
```

Flutterの開発

基本的なコードの書き方は以下のような感じ

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

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Hello Flutter'),
        ),
        body: Center(
          child: Text('Welcome to Flutter'),
        ),
      ),
    );
  }
}
```

Flutterの開発

基本的なコードの書き方は以下のような感じ

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

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Hello Flutter'),
        ),
        body: Center(
          child: Text('Welcome to Flutter'),
        ),
      ),
    );
  }
}
```

Flutterの開発

基本的なコードの書き方は以下のような感じ

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

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Hello Flutter'),
        ),
        body: Center(
          child: Text('Welcome to Flutter'),
        ),
      ),
    );
  }
}
```

Flutterの開発

基本的なコードの書き方は以下のような感じ

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

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Hello Flutter'),
        ),
        body: Center(
          child: Text('Welcome to Flutter'),
        ),
      ),
    );
  }
}
```

Widgetとは？

- 先程のコードで説明した通り、Flutterでは**Widget**を使用してUIを構築していく
- FlutterではWidgetを公式から提供されており、それらを組み合わせて画面を作成していく
- 提供されているWidgetの一覧は以下のページから確認可能
  - Widget catalog | Flutter

FlutterとGraphQLの連携①

Flutterでもgraphql_codegenを導入すれば型安全でGraphQLのクエリを実行できるので手順を紹介

まずは、backendからGraphQLのスキーマファイルを取得

```
type Query {
  categories: [Category]
  category(id: Int!): Category
}
type Category {
  "カテゴリーID"
  id: ID!
  "カテゴリー名"
  name: String!
  "順番"
  order: Int!
}
```

FlutterとGraphQLの連携②

次にFlutterからアクセスするためのクエリを作成

```
query Category($id: Int!) {
  category(id: $id) {
    id
    name
    order
  }
}
```

FlutterとGraphQLの連携③

graphql_codegenの初期設定をして以下のコマンドを実行

```
dart run build_runner build
```

以下のファイルが生成される

- lib/graphql/category.gql.dart

FlutterとGraphQLの連携④

③で生成したファイルを参照して、以下のようにクエリを実行して値を取得

```
class Items extends HookWidget {
  final int id;

  const Items({super.key, required this.id});

  @override
  Widget build(BuildContext context) {
    final queryResult = useQuery$Category(
        Options$Query$Category(variables: Variables$Query$Category(id: id)));

    final result = queryResult.result;

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

    final Query$Category$category category = Query$Category$category.fromJson(
        result.data!['category'] as Map<String, dynamic>);

    return Text(category.name);
  }
}
```

FlutterとGraphQLの連携④

③で生成したファイルを参照して、以下のようにクエリを実行して値を取得

```
class Items extends HookWidget {
  final int id;

  const Items({super.key, required this.id});

  @override
  Widget build(BuildContext context) {
    final queryResult = useQuery$Category(
        Options$Query$Category(variables: Variables$Query$Category(id: id)));

    final result = queryResult.result;

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

    final Query$Category$category category = Query$Category$category.fromJson(
        result.data!['category'] as Map<String, dynamic>);

    return Text(category.name);
  }
}
```

FlutterとGraphQLの連携④

③で生成したファイルを参照して、以下のようにクエリを実行して値を取得

```
class Items extends HookWidget {
  final int id;

  const Items({super.key, required this.id});

  @override
  Widget build(BuildContext context) {
    final queryResult = useQuery$Category(
        Options$Query$Category(variables: Variables$Query$Category(id: id)));

    final result = queryResult.result;

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

    final Query$Category$category category = Query$Category$category.fromJson(
        result.data!['category'] as Map<String, dynamic>);

    return Text(category.name);
  }
}
```

FlutterとGraphQLの連携④

③で生成したファイルを参照して、以下のようにクエリを実行して値を取得

```
class Items extends HookWidget {
  final int id;

  const Items({super.key, required this.id});

  @override
  Widget build(BuildContext context) {
    final queryResult = useQuery$Category(
        Options$Query$Category(variables: Variables$Query$Category(id: id)));

    final result = queryResult.result;

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

    final Query$Category$category category = Query$Category$category.fromJson(
        result.data!['category'] as Map<String, dynamic>);

    return Text(category.name);
  }
}
```

FlutterとGraphQLの連携④

③で生成したファイルを参照して、以下のようにクエリを実行して値を取得

```
class Items extends HookWidget {
  final int id;

  const Items({super.key, required this.id});

  @override
  Widget build(BuildContext context) {
    final queryResult = useQuery$Category(
        Options$Query$Category(variables: Variables$Query$Category(id: id)));

    final result = queryResult.result;

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

    final Query$Category$category category = Query$Category$category.fromJson(
        result.data!['category'] as Map<String, dynamic>);

    return Text(category.name);
  }
}
```

今回作成したリポジトリの紹介

- Repository
  - wheatandcat/stock-keeper
  - wheatandcat/stock-keeper-backend

React Nativeとの比較

まだ、Flutterの方はざっくりな開発しかしていないが、現状の所感での比較を記載

- パフォーマンスチューニングのやりやすさ
  - Flutter >> React Native
- 運用/保守コスト
  - Flutter >> React Native
- UI/システムの柔軟性
  - React Native > Flutter
- 開発のとっつきやすさ
  - Expo >>> Flutter > React Native
- エコシステムの充実度/熟練度
  - React Native >>> Flutter
- backendとの連携
  - React Native > Flutter

まとめ

- よくFlutterとReact Nativeの比較がされる記事を見かけていたが、Flutterは触ってなかったので、あまりピンと来てなかったが、今回の開発でその辺が把握できた
- dartは簡単なので、たぶんReact経験者なら、すぐに慣れると思う
- Flutterの魅力はGoogle独自のWidgetを使用することで、プラットフォーム毎でUIに差分が発生しづらく保守コストが少なく済むところ
- React Nativeはパフォーマンスチューニングのコーディング難易度が非常に高いので、その辺はFlutterの方が強そう
- エコシステムの充実度はnpmの圧倒的な数 & 完成度の高いパッケージが多いので、この辺はReact Nativeの方に軍配が上がる
- GraphQLの連携周りも React Nativeの方がサポートが充実している
- 初期開発の充実度はExpoが最も優れており、アプリリリースまではコストは一番低そう
- 今後増えてくるtRPCのサポートに関してはFultterだと言語の壁があり、もしサポートされてもファースト扱いにはならないので、この辺はReact Nativeの方が有利そう
- 思った以上に良い勝負をしているので、今後の動向が楽しみ

ご清聴ありがとうございました🎉