

B5 Cover Page

(182x257mm or 516x728pt)

0円で、モバイルアプリを作つ てみるお～。Vol.1

— Flutter、Firebase、GetX でステップ毎に解説 —

[著] 今北産業

技術書典 13（2022年夏）新刊
2022年8月29日 ver 1.0

■免責

本書は情報の提供のみを目的としています。

本書の内容を実行・適用・運用したことで何が起きようとも、それは実行・適用・運用した人自身の責任であり、著者や関係者はいかなる責任も負いません。

■商標

本書に登場するシステム名や製品名は、関係各社の商標または登録商標です。

また本書では、™、®、©などのマークは省略しています。

はじめに

このたびは、「0円で、モバイルアプリ作るお～！」を手にしていただき、誠にありがとうございます。

今回も、tkg 様の開発された「AA スレ作成アプリ AA ストーリーボード」を使わせていただいます。tkg 様はじめたくさんの AA を投稿されている皆様へ感謝しています。



今回紹介させていただいている「Flutter」ですが、驚くほど簡単にモバイルアプリを作成することができます。「Flutter」がどんなものかをサクッと知りたい方は、以下の [Youtube 動画^{*1}](#) がおすすめです。10分ほどの動画ですが「Flutter」の概要が分かります。



実は、私は「React、Redux、Electron」を使い Mac、Windows 用のデスクトップアプリを開発していました。その縁で「ReactNative」に乗り換えようとしたのですが、開発環境の作成で挫折しました。

いえ、別に「ReactNative」をディスるつもりはありません。

誰でもが簡単にモバイルアプリを作成できるようになり、面白いアプリが世の中にあふれるとい

^{*1} https://youtu.be/5fH6__PAi6A

いですね。

本書は、

- Flutter -- 開発環境
- Firebase -- 認証、データベース、クラウドストレージ、メッセージング
- GetX -- 状態管理 (React での Redux、Recoil、Jotai のようなもの)

を使いオンライン・フリーマーケットを作成します。

E-mail、Twitter アカウント、Google アカウント、AppleID での自分のオンラインショップを作成し、商品の登録、管理、販売ができます。サンプルアプリですので決済までは含んでいません。

構成は、

1. (本書 Vol.1) アカウント作成と認証。ストア情報の管理。
2. (次号 Vol.2) 商品登録・管理
3. (Vol.3) 商品管理、販売カート
4. (Vol.4) 販売・在庫切れなどの通知

と、なります。

とても難しそうですが、Flutter は Google がオープンソースで公開している開発フレームワークです。そのため世界中の開発者が

- GitHub などでアプリのソース公開
- Youtube での解説
- Qiita.com、zenn.dev などの情報

を発信していますので、助けになるものが多くあります。日本語情報も増えていますが、英語でも動画や GitHub でのコードであれば何とかなります。ちょっとだけ頑張れば、あなたもモバイルアプリ作れます。

本書で扱うオンライン・フリーマーケットもゼロからの開発ではなく、GitHub などで公開されているアプリを魔改造していきます。出典元や情報サイトなども提示していますので参考にしてください。

本書のソースコードは GitHub にあります。章毎にブランチを分けていますので、どの章からでも始めていただけます。出来る限り図解していますのでページ数も膨大になりますが、ご容赦ください。

さい。

また、Flutter 開発環境の作成については本書では扱っていません。別途、無償配布しています「Flutter 開発環境の作成 Mac 編」、「Flutter 開発環境の作成 Windows 編」を参照してください。

最後になりますが、本書での「間違い」、「認識不足」、「誤字脱字」、「参照元のリンク切れ」などありましたら、[本書サポートサイト^{*2}](#)にて issue を発行していただけませんでしょうか？

よろしくお願ひいたします。最後まで楽しんでいただけることを願っています。

^{*2} https://github.com/risingforce9zz/book04_t-shirts-freemarket/blob/main/README.md

目次

はじめに	i
第1章 状態管理とは？	1
1.1 状態管理とは？	2
1.2 状態管理ライブラリ	3
1.3 GetX を使っての状態管理の解説	4
1.3.1 スタートアッププロジェクト	4
1.3.2 GetX を導入	5
第2章 Firebase とは？	14
2.1 Firebase を使ってみる	15
2.1.1 Firebase コンソールへ	16
2.1.2 Firebase 認証とは？	22
第3章 アプリケーション認証	23
3.1 認証フロー	24
3.1.1 スプラッシュスクリーンからログイン選択	24
3.1.2 新規登録	25
3.1.3 メール/パスワード	26
3.1.4 SNS 認証	26
3.1.5 プロフィール画面	27
第4章 プロジェクトスタート	29
4.1 新規プロジェクトの作成	30

第 1 章

状態管理とは？

今回、状態管理で使用するのは GetX(状態管理だけではないですが・・・) です。
では、「状態管理」とは一体何を指していて、なぜ必要なのでしょうか？

【この章の内容】

1.1	状態管理とは？	2
1.2	状態管理ライブラリ	3
1.3	GetX を使っての状態管理の解説	4

1.1

状態管理とは？

IT業界での状態管理（英語では-State Management-）の「状態（State）」は、アプリケーションの一部であり、たとえば、

- データベースで管理しているデータ
- ブラウザで発生するイベント、表示している色、形

です。それらは、バックエンド、データベース、フロントエンドのコンポーネントなど、アプリケーションのどこにでもあります。

たとえば、スマートフォンで表示しているボタンの色や形状などです。

それでは、状態を管理する「状態管理」とは何なのでしょうか？

「State-Management」とは、デザインパターンによる実装であり、アプリケーションのあちこちに散らばる状態をすべてのコンポーネントで状態を同期させ、「サービスの実装」や「データベースからのデータの取扱」を楽にすること。

だ、そうです。

上記の例ですと、ボタンの形を変えるためにAモジュールにアクセスし、色を変えるためにBモジュールにアクセスするような実装ではなくボタンの状態を管理するモジュールがあれば、そこにアクセスするだけで良いとする実装を指します。

1.2 状態管理ライブラリ

React でも同様ですが状態を管理するライブラリは、たいていの場合、戦国時代の武将のように湧いてきて、戦いの後は、「ひとりの絶大な勝者」と「2~3人ほどの生き残り」となります。その後時間が経過すると、この状況に不満をもつ新興勢力が台頭してきます。

Flutter の世界も同様で、現在は戦いの終盤戦に近づいているのではないかでしょうか？

Flutter の公式サイトでも状態管理のライブラリ^{*1} を公開しています。しかし、公式サイトですので、どれが良いなどとは口が裂けても言えない状況にあります。

コードを書く人は、たいていの場合どのライブラリが良いかを議論しません。過去の経験から「宗教戦争」になると心得ているからです。しかし、内心では、「状態管理と言えば、XXX一択！」と思っています。

本書では、GetX を使いますが、Riverpod 推しや Provider 推しの方はお許しください。

^{*1} <https://docs.flutter.dev/development/data-and-backend/state-mgmt/options>

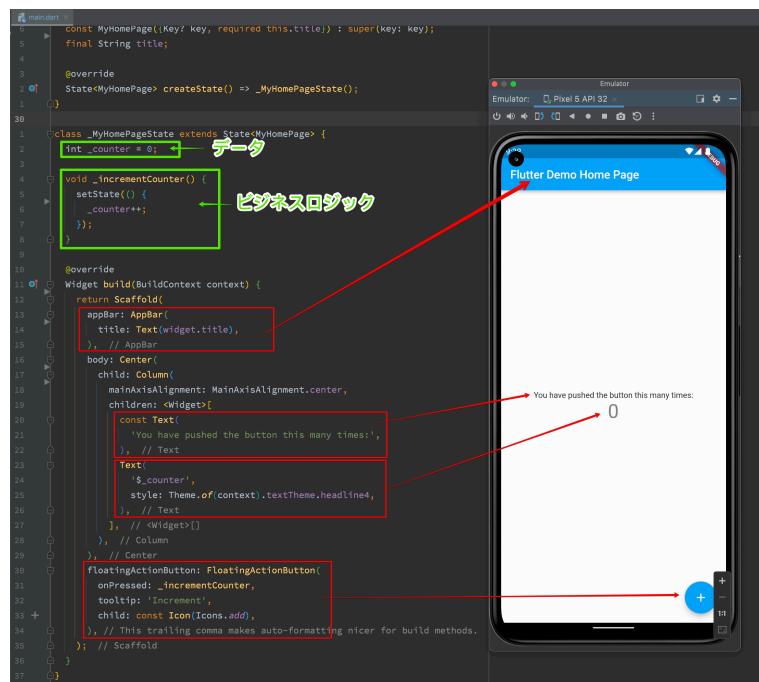
1.3

GetX を使っての状態管理の解説

それでは、Flutter のスタートアッププロジェクトを使って説明します。

♣ 1.3.1 スタートアッププロジェクト

下図は、Android Studio で新規プロジェクトを作成しエミュレータでデバッグしているところです。



おなじみのスタートアッププロジェクトが作成されており、FAB（フローティング・アクション・ボタン）をクリックすると表示されているカウンターが増えていきます。

コードを見ると、

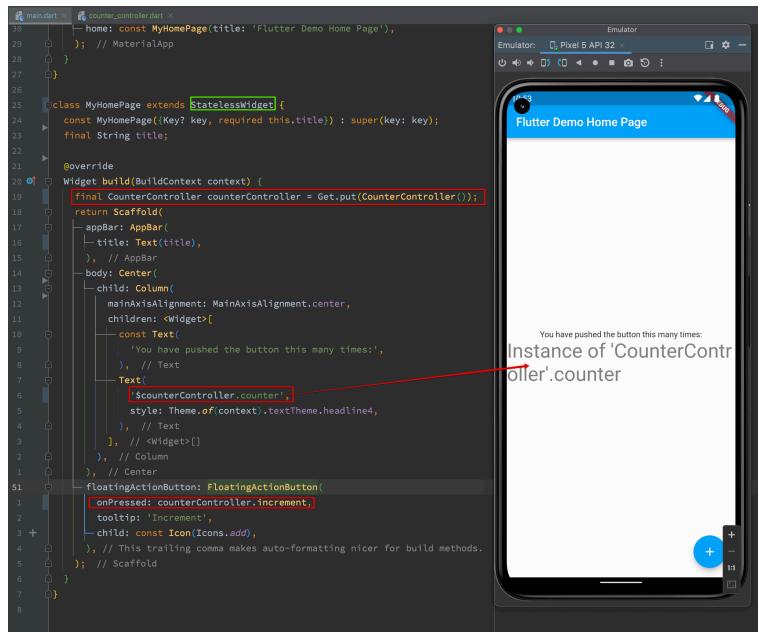
- 状態を保持する `State` を継承した状態管理 `setState` を使用
- データである `_counter` を持っている
- データを操作する `_incrementCounter` ビジネスロジックを持っている。
- データを表示する `Text Widget` を持っている。

つまり、表示用コード、データ、ビジネスロジックが同居しています。別なページから、このデータにアクセスするためにはどのようにするのでしょうか？

♣ 1.3.2 GetX を導入

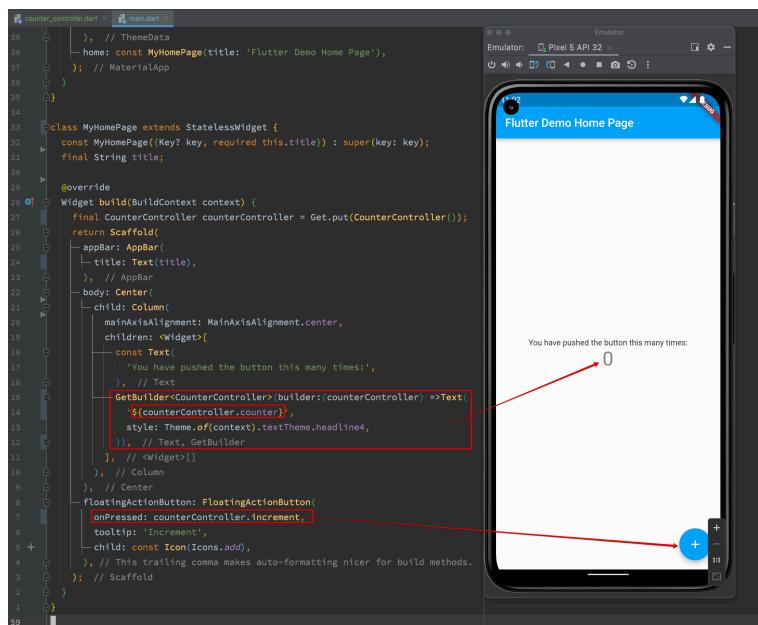
GetX（あくまで本書では）を導入し状態管理を行います。

ご覧のように、データ、ビジネスロジックとも外部に移動しています。



The screenshot shows the Flutter development environment. On the left, the code editor displays `main.dart` and `counter_controller.dart`. The `main.dart` file contains the application's entry point, defining a `MaterialApp` with a `MyHomePage` widget. The `MyHomePage` class extends `StatelessWidget` and uses `GetX` to inject a `CounterController` instance. The `build` method creates a `Scaffold` with an `AppBar` and a `Center` body. The body contains a `Column` with a `Text` widget displaying the message "You have pushed the button this many times:" and another `Text` widget displaying the value of `counterController.counter`. A `FloatingActionButton` is also present. The `counter_controller.dart` file defines the `CounterController` class, which implements the `GetxController` interface and contains a `counter` variable. On the right, the emulator shows the application running on a Pixel 5 API 32 device. The screen displays the title "Flutter Demo Home Page" and the text "You have pushed the button this many times: Instance of 'CounterController'.counter". A red arrow points from the error in the `main.dart` code to the `counterController.counter` text on the emulator screen.

データ表示部にエラーが出ているのは、表示部分に状態管理を適用していないためです。



This screenshot shows the same setup as the previous one, but with a fix applied to the code. In `main.dart`, the `Text` widget that was previously showing an error now uses `GetBuilder` to build its content. The `GetBuilder` widget takes a `builder` function that receives the `counterController` instance as a parameter. Inside this function, the `Text` widget is created with the value of `counterController.counter`. The rest of the code remains the same. The emulator on the right shows the updated state: the text now correctly displays "0" instead of the previous error message. A red arrow points from the `GetBuilder` code in `main.dart` to the "0" value on the emulator screen.

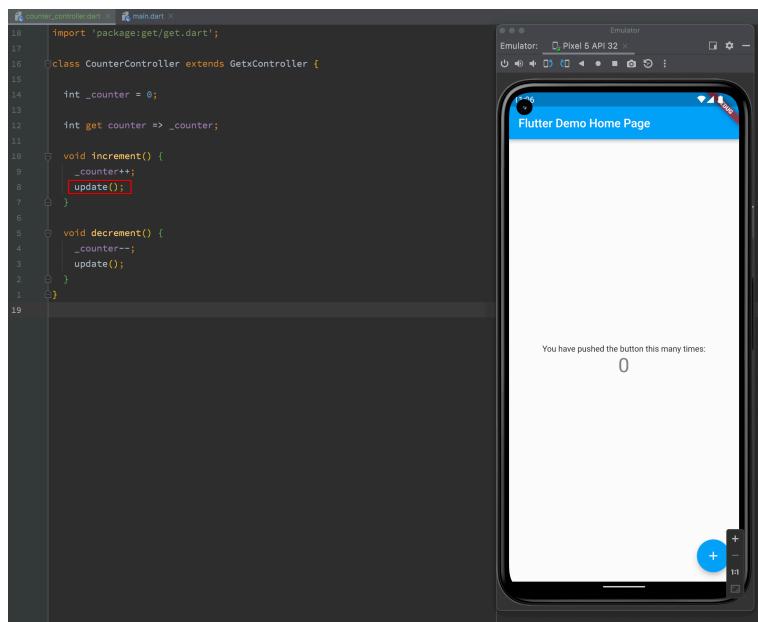
表示部に状態管理「GetBuilder」を適用するとデータが表示されています。実は、GetBuilder自体は、 StatefulWidget を継承していて GetBuilder 自体が状態を保持しています。

▼ GetBuilder の正体

```
class GetBuilder<T extends GetxController> extends StatefulWidget {  
    final GetControllerBuilder<T> builder;  
    final bool global;  
    final Object? id;  
    final String? tag;  
    final bool autoRemove;  
    final bool assignId;  
    final Object Function(T value)? filter;  
    final void Function(GetBuilderState<T> state)? initState,  
        dispose,  
        didChangeDependencies;  
    final void Function(GetBuilder oldWidget, GetBuilderState<T> state)?  
        didUpdateWidget;  
    final T? init;  
  
    const GetBuilder({  
        Key? key,  
        this.init,  
        this.global = true,  
        required this.builder,  
        this.autoRemove = true,  
        this.assignId = false,  
        this.initState,  
        this.filter,  
        this.tag,  
        this.dispose,  
        this.id,  
        this.didChangeDependencies,  
        this.didUpdateWidget,  
    }) : super(key: key);
```

状態管理をしているコントローラー

状態管理をしているコントローラーは、こんな風になっています。



コントローラーは、「GetX コントローラー」を継承し、データとビジネスロジックを持っていて、ビジネスロジック「increment」に「update()」が含まれているのは、「GetBuilder」に、

状態が変化したので表示を書き換えてね。お願いつ

と通知するためです。

状態が変化すると、自動で再描画する「GetX<>」、「Obx()」のような優秀なのもいます。しかし、優秀なのは、それなりにメモリを使う・CPUを使うなどリソースを使います。

何がうれしいのか？

では、状態管理コントローラーを導入したメリットは何でしょうか？

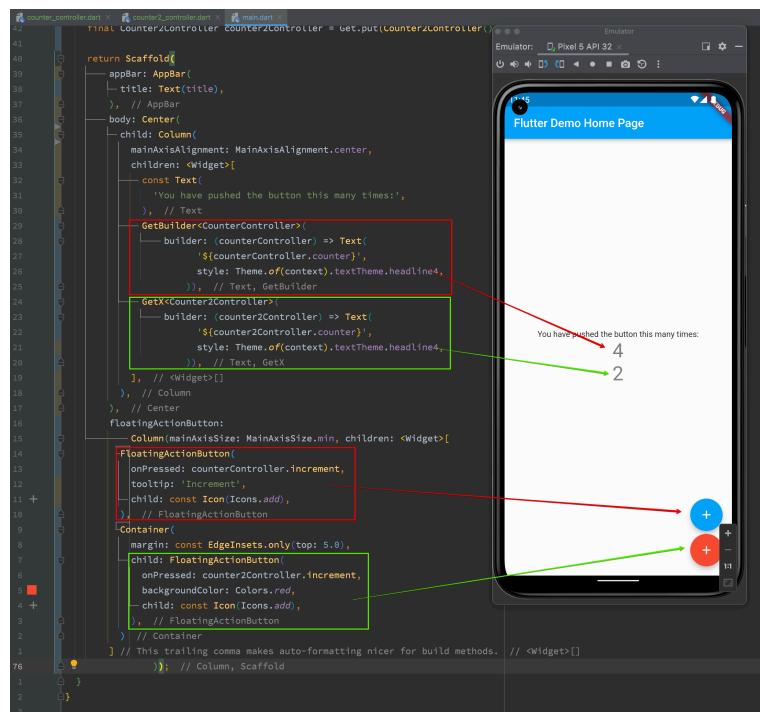
1. データとビジネスロジックを一ヵ所にまとめたので、変更があっても一ヵ所。
 2. 状態管理コントローラーを増やしても、表示部のコード変更は最小です。
 3. 別なページにデータを簡単に渡せる。
 4. データと表示ページには関連がないので、ページを再表示してもデータ再取得がない。

と、たくさんのメリットがあります。

表示を増やしてみた

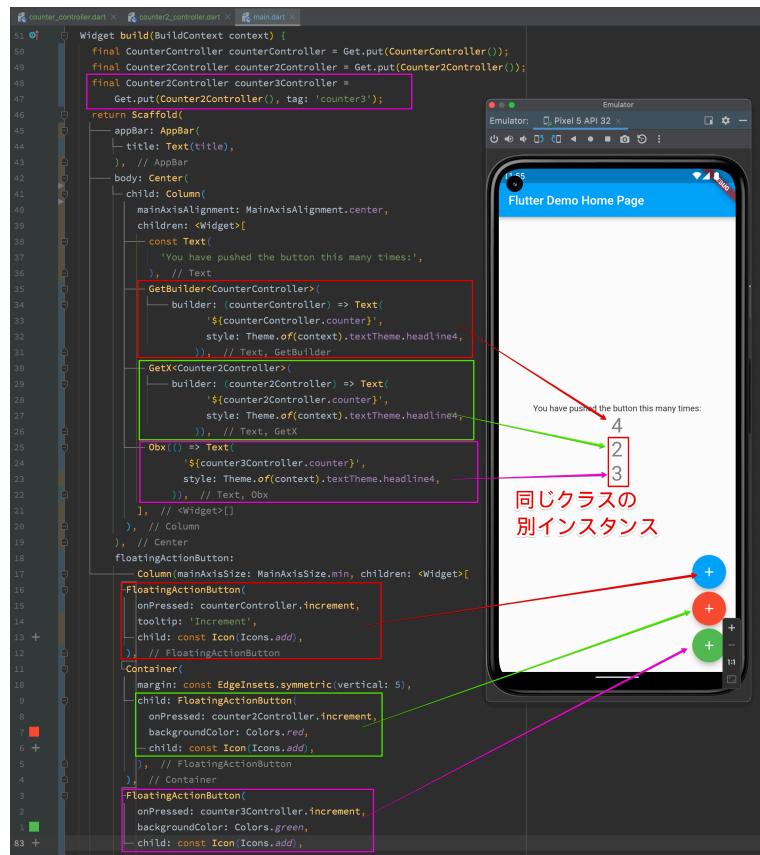
表示部を2つ、それぞれを操作するボタンを2つにしましたが、状態管理コントローラーをひとつ増やしただけです。こちらには、優秀な「GetX<コントローラー型>」を使って表示している

ます。



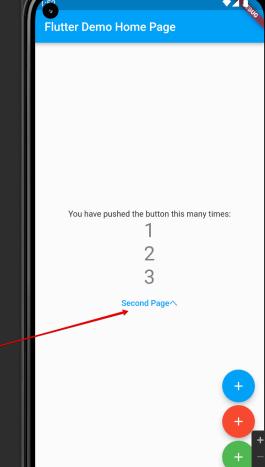
状態管理コントローラーの別インスタンス

同じデータ型、同じビジネスロジックを持つのであれば、状態管理コントローラーの別インスタンスで管理できます。インスタンスが別ですので、それぞれの操作で、それぞれのデータを管理できます。



別ページでも同じ状態管理コントローラーへアクセス

ページ移動を行っても、アプリケーション全体で状態管理コントローラーは動作していますので、状態管理コントローラーへアクセスすれば、先ほどまでのデータを取得できます。



```
class CounterController extends ChangeNotifier {
    int counter = 0;
    void increment() => counter++;
    void decrement() => counter--;
}

class Counter2Controller extends ChangeNotifier {
    int counter = 0;
    void increment() => counter++;
    void decrement() => counter--;
}

class Counter3Controller extends ChangeNotifier {
    int counter = 0;
    void increment() => counter++;
    void decrement() => counter--;
}

class CounterControllerCounter extends StatelessWidget {
    final CounterController counter3Controller =
        Get.put(CounterController(), tag: 'counter3');
    final Counter2Controller counter2Controller =
        Get.put(Counter2Controller(), tag: 'counter2');

    return Scaffold(
        appBar: AppBar(
            title: Text('title'),
        ), // AppBar
        body: Center(
            child: Column(
                mainAxisAlignment: MainAxisAlignment.center,
                children: <Widget>[
                    const Text(
                        'You have pushed the button this many times:',
                    ), // Text
                    GetBuilder<CounterController>(
                        builder: (counterController) => Text(
                            '${counterController.counter}',
                            style: Theme.of(context).textTheme.headline4,
                        ), // Text, GetBuilder
                    ),
                    GetX<Counter2Controller>(
                        builder: (counter2Controller) => Text(
                            '${counter2Controller.counter}',
                            style: Theme.of(context).textTheme.headline4,
                        ), // Text, GetX
                    ),
                    Obx(() => Text(
                        '${counter3Controller.counter}',
                        style: Theme.of(context).textTheme.headline4,
                    )), // Text, Obx
                    TextButton(
                        child: const Text('Second Page^'),
                        onPressed: () => Get.to(() => const SecondPage()),
                    ), // TextButton
                ],
            ), // Column
        ), // Center
        floatingActionButton:
            Column(mainAxisSize: MainAxisSize.min, children: <Widget>[
                FloatingActionButton(
                    heroTag: 'counter1',
                    onPressed: counterController.increment,
                    tooltip: 'Increment',
                    child: const Icon(Icons.add),
                ), // FloatingActionButton
                Container(
                    margin: const EdgeInsets.symmetric(vertical: 5),
                    child: FloatingActionButton(
                        heroTag: 'counter2',
                        onPressed: counter2Controller.increment,
                        backgroundColor: Colors.red,
                        child: const Icon(Icons.add),
                    ), // FloatingActionButton
                ), // Container
                FloatingActionButton(
                    heroTag: 'counter3',
                ), // FloatingActionButton
            ]),
    );
}
```

The screenshot shows the Android Studio interface with two main parts: the code editor and the emulator.

Code Editor (counter_controller.dart):

```
42 import 'package:flutter/material.dart';
43 import 'package:get/get.dart';
44
45 import '../controllers/counter2_controller.dart';
46
47 class SecondPage extends StatelessWidget {
48   const SecondPage({Key? key}) : super(key: key);
49
50   @override
51   Widget build(BuildContext context) {
52     final Counter2Controller counter2Controller = Get.find();
53     final Counter3Controller counter3Controller = Get.find(tag: "counter3");
54
55     return Scaffold(
56       appBar: AppBar(
57         leading: IconButton(
58           icon: const Icon(Icons.arrow_back),
59           onPressed: () => Get.back(),
60         ), // IconButton
61         title: const Text('second page'),
62       ), // AppBar
63       body: Center(
64         child: Column(
65           mainAxisAlignment: MainAxisAlignment.center,
66           children: [
67             const Text('counter2Controllerの値'),
68             Getx<Counter2Controller>(
69               builder: (counter2Controller) => Text(
70                 '$counter2Controller.counter',
71                 style: Theme.of(context).textTheme.headline4,
72               ), // Text, GetX
73             ),
74             const Text('counter2Controllerの別インスタンス'),
75             Obx(() => Text(
76               '$counter3Controller.counter',
77               style: Theme.of(context).textTheme.headline4,
78             ), // Text, Obx
79           ], // <Widget>[]
80         ), // Column, Center
81         floatingActionButton: FloatingActionButton(
82           onPressed: counter2Controller.increment,
83           backgroundColor: Colors.red,
84           child: const Icon(Icons.add),
85         ), // FloatingActionButton
86       ), // scaffold
87     );
88   }
89 }
```

Emulator: The emulator shows the application running on a Pixel 5 API 32 device. The title bar says "second page". The screen displays the value "2" and the text "counter2Controllerの別インスタンス". A floating action button (FAB) is visible at the bottom right.

Annotations:

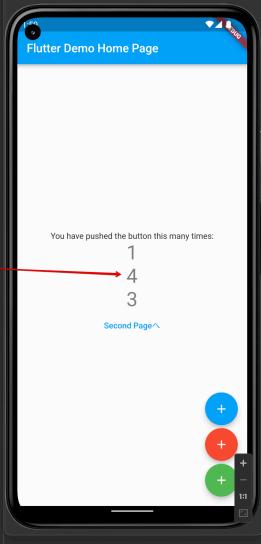
- A red box highlights the line `final Counter2Controller counter2Controller = Get.find();`.
- A yellow box highlights the line `final Counter3Controller counter3Controller = Get.find(tag: "counter3");`.
- A green box highlights the line `Getx<Counter2Controller>(...)`.
- An orange arrow points from the code editor to the "counter2Controller" value "2" in the emulator.
- A blue arrow points from the code editor to the "counter3Controller" value "3" in the emulator.
- A red circle with a plus sign is located in the bottom right corner of the emulator screen.

ここでデータを操作し変化させます。

The screenshot shows the Android Studio interface. On the left is the code editor with the file `second_page.dart` open. The code defines a `SecondPage` widget that extends `StatelessWidget`. It imports `package:flutter/material.dart`, `package:get/get.dart`, and `../controllers/counter2_controller.dart`. The `build` method creates a `Scaffold` with an `AppBar` containing a back button and the title "second page". The `body` contains a `Column` with a main axis alignment of center. Inside the `Column`, there is a `Text` widget with the placeholder "counter2Controllerの値" and a `GetX<Counter2Controller>` builder. This builder takes a `counter2Controller` parameter and returns a `Text` widget with the value of `counter2Controller.counter`. Below this is another `Text` placeholder "counter3Controllerの値". A red box highlights the `GetX` builder and the `Text` widget it generates. To the right of the code editor is the Android emulator showing the app running on a Pixel 5 API 32 device. The screen displays the title "second page" and two `Text` widgets. The top `Text` has the value "counter2Controllerの値" followed by "4" and "3", with a red arrow pointing from the "4" to the floating action button. The bottom `Text` has the placeholder "counter3Controllerの値". A red box highlights the floating action button, which has a plus sign (+) icon.

```
42 import 'package:flutter/material.dart';
43 import 'package:get/get.dart';
44
45 import '../controllers/counter2_controller.dart';
46
47 class SecondPage extends StatelessWidget {
48   const SecondPage({Key? key}) : super(key: key);
49
50   @override
51   Widget build(BuildContext context) {
52     final Counter2Controller counter2Controller = Get.find();
53     final Counter3Controller counter3Controller = Get.find(tag: 'counter3');
54
55     return Scaffold(
56       appBar: AppBar(
57         leading: IconButton(
58           icon: const Icon(Icons.arrow_back),
59           onPressed: () => Get.back(),
60         ), // IconButton
61         title: const Text("second page"),
62       ), // AppBar
63       body: Center(
64         child: Column(
65           mainAxisAlignment: MainAxisAlignment.center,
66           children: <Widget>[
67             const Text("counter2Controllerの値"),
68             GetX<Counter2Controller>(
69               builder: (counter2Controller) => Text(
70                 '$counter2Controller.counter',
71                 style: Theme.of(context).textTheme.headline4,
72               ), // Text, GetX
73               const Text("counter2Controllerの別インスタンス"),
74               Obx(() => Text(
75                 '$counter3Controller.counter',
76                 style: Theme.of(context).textTheme.headline4,
77               )), // Text, Obx
78             ], // <Widget>[]
79           ), // Column, Center
80           floatingActionButton: FloatingActionButton(
81             onPressed: counter2Controller.increment,
82             backgroundColor: Colors.red,
83             child: const Icon(Icons.add),
84           ), // FloatingActionButton
85         ); // Scaffold
86       },
87     );
88   }
89 }
```

元ページに戻っても、アクセスしている状態管理コントローラーのデータを取得しますので、データは最新です。



```

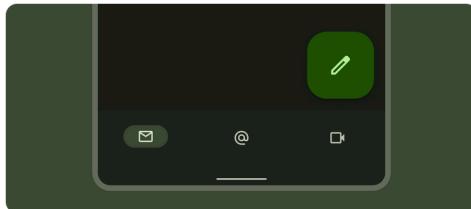
52   final Counter2Controller counter2Controller =
53     Get.put(Counter2Controller(), tag: "counter2");
54
55   final Counter3Controller counter3Controller =
56     Get.put(Counter3Controller(), tag: "counter3");
57
58   return Scaffold(
59     appBar: AppBar(
60       title: Text(title),
61     ),
62     body: Center(
63       child: Column(
64         mainAxisAlignment: MainAxisAlignment.center,
65         children: <Widget>[
66           const Text(
67             'You have pushed the button this many times:',
68           ),
69           // Text
70           GetBuilder<CounterController>(
71             builder: (counterController) => Text(
72               '$(counterController.counter)',
73               style: Theme.of(context).textTheme.headline4,
74             ),
75             // Text, GetX
76             GetX<Counter2Controller>(
77               builder: (counter2Controller) => Text(
78                 '${counter2Controller.counter}',
79                 style: Theme.of(context).textTheme.headline4,
80               ),
81             ), // Text, GetX
82             Obx(() => Text(
83               '${counter3Controller.counter}',
84               style: Theme.of(context).textTheme.headline4,
85             )),
86             // Text, Obx
87             TextButton(
88               child: const Text("Second Page^"),
89               onPressed: () => Get.to(() => const SecondPage()),
90             ),
91           ],
92         ),
93       ),
94     ),
95   );
96 }
97
98 
```

FAB はひとつ。

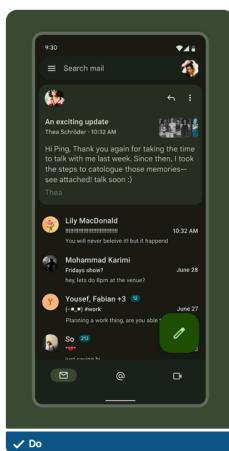
Flutter で推奨されている Material Design では、FAB が複数あることを推奨していません。今回は、状態管理の解説のため簡易的にです。

FAB

Use a FAB to represent the screen's primary action.



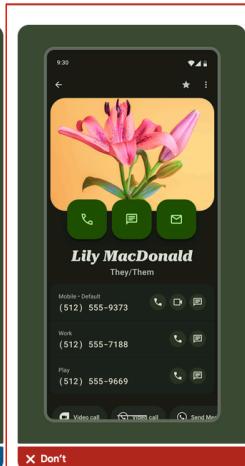
A FAB should present a screen's primary action.



✓ Do
Represent the most common primary action with a FAB, such as drafting a new email.



✓ Do
FABs are not needed on every screen, such as when images represent primary actions.



✗ Don't
Don't display multiple FABs on a single screen. Instead, consider using a menu or, in rare cases, pairing a small FAB with a default FAB.

第 2 章

Firebase とは？

もともとは、オンラインチャット機能を Web サイトに統合する API を提供していたスタートアップ Envolve が、ユーザがチャットではなくゲームデータをリアルタイムに交換しているのを見て、別会社 Firebase を設立し iOS、Android、Web でアプリケーションデータを同期する「Firebase Realtime Database」を発表する。

この製品を中心とし、「Firebase Hosting」、「Firebase Authentication」を合わせモバイル・バックエンドサービスを提供していた。

ネットの世界では、スタートアップはツブされるか買収されるかの 2 択なので、結局 Google に買収されてしまう。

Google は、更にサービスを追加しモバイル・バックエンド・サービス、サーバーレス・アーキテクチャの勝者となります。

【この章の内容】

2.1 Firebae を使ってみる	15
------------------------------	----

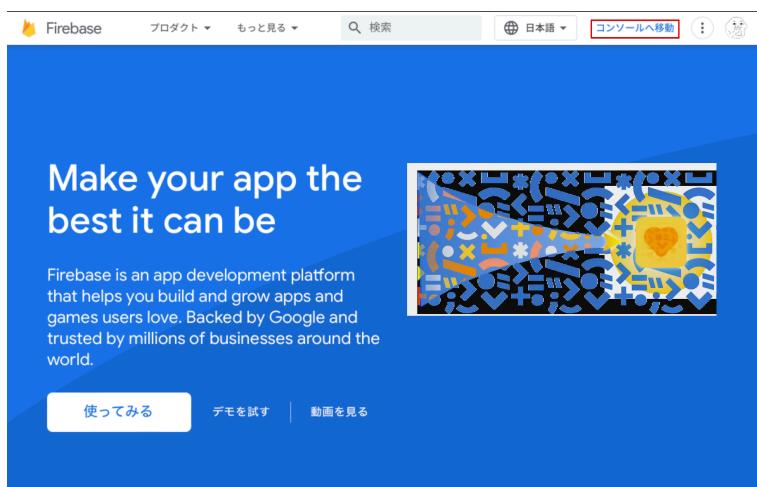
2.1 Firebase を使ってみる

Firebase は、個別にたくさんのサービスを提供しています。いきなり Firebase を使うと言っても、どのサービスを使ってよいものかもわかりません。

使い方は、

- プロジェクトを開始し、
 - その都度、必要なサービスを追加
- します。

Google アカウントがあれば、スグに使い始めることができますし、ある程度の使用量は無償です。



料金については、

Firebase Authentication（認証）では、以下のようにアクティブユーザが月間 5 万人まで無料です。

The screenshot shows the Firebase Pricing page for the 'Spark' plan. The plan is described as '初めての方向けに柔軟な上限を設定 無料'. The 'Authentication' section is highlighted with a red border. It includes the following details:

サービス	上限
電話認証 - 米国、カナダ、インド	1万/月
電話認証 - 他のすべての国	1万/月
他の認証サービス	✓
With Identity Platform	
Monthly active users	50k/month
Monthly active users - SAML/OIDC	50/month

Realtime Database では、保存するデータが 1GB まで無料です。ただし、ダウンロードされるデータは 10GB までです。

実際に作成してデータをみていただくと、Realtime Database に保存されるデータは、巨大な JSON ファイルですので文字列で 1GB となると、相当量です。

The screenshot shows the Firebase Pricing page for the 'Spark' plan. The plan is described as '初めての方向けに柔軟な上限を設定 無料'. The 'Realtime Database' section is highlighted with a red border. It includes the following details:

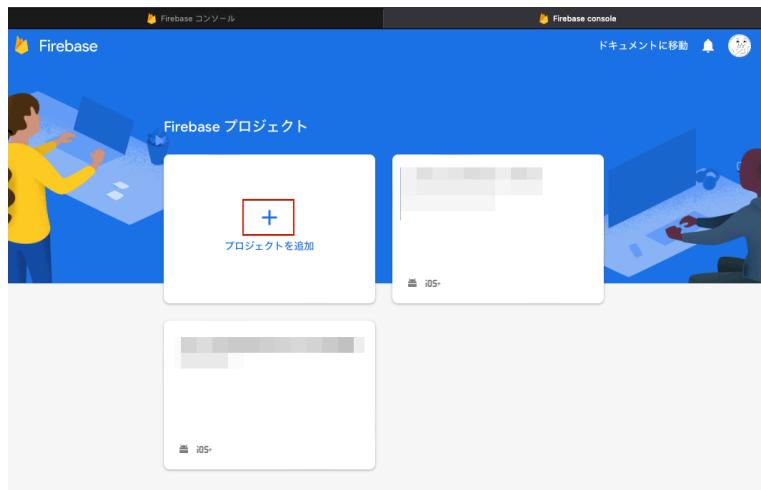
サービス	上限
データ保存	1GB
データ転送	10GB
データ検索	10TB
データ削除	1TB
データ同期	1TB
Monthly active users	50k/month
Monthly active users - SAML/OIDC	50/month

無料から有料へは、いつでも切り替えることが出来ますし、使用量が減れば無料プランへ戻ることもできます。

♣ 2.1.1 Firebase コンソールへ

「コンソールへ移動」をクリックすると、Firebase コンソールトップへ移動します。ここで「+」

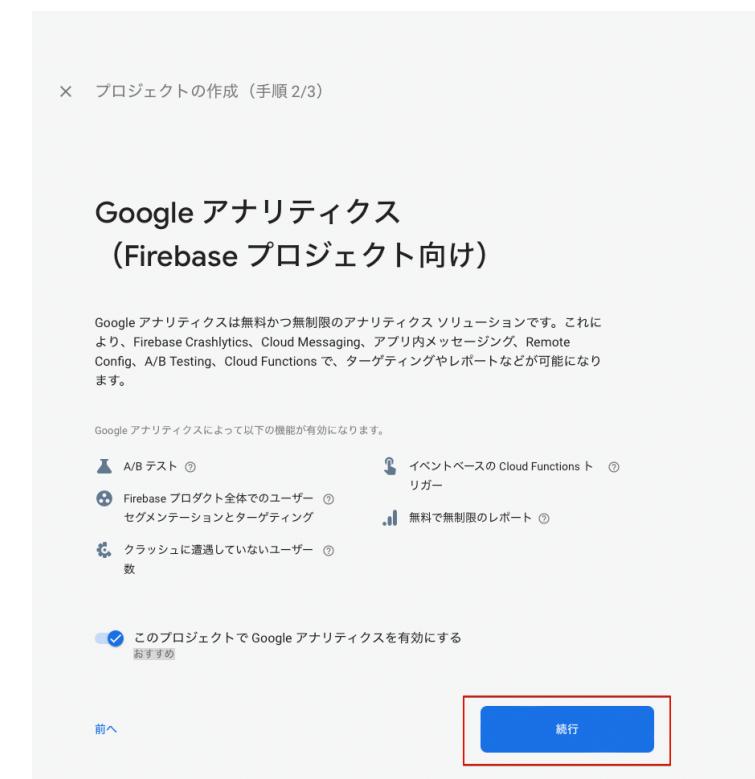
「プロジェクトを追加」をクリックして新規プロジェクトを作成します。



最初にプロジェクトに名前を付けます。使える文字は半角英数、スペース、-!'"の記号 4 つです。
[続行] ボタンをクリックします。



Google アナリティクスの画面になります。使う場合には Andorid 向け SDK のバージョンが 19 となります。



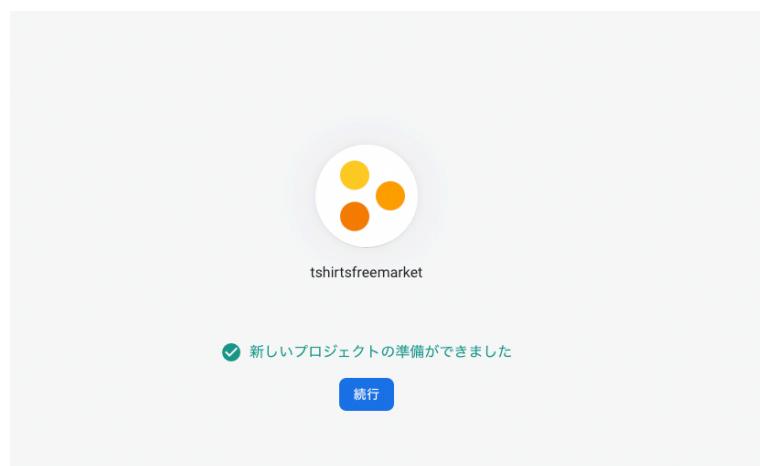
アナリティクスのアカウントを選択するか、作成します。[プロジェクトを作成] ボタンをクリックすると作成開始します。



作成中です。



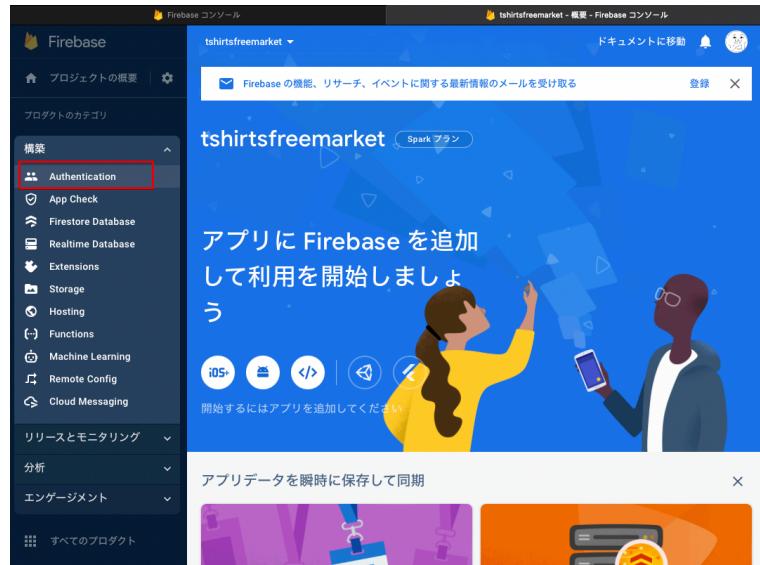
プロジェクトが出来上りました。



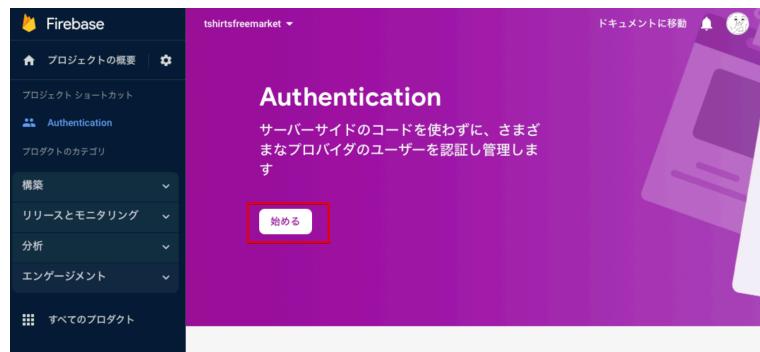
サイドバーにある [構築] ドロップダウンを開くと、追加できるサービスが表示されます。



最初に、「Authentication」をクリックし認証サービスを追加します。



認証サービスのトップ画面に移動します。[始める] ボタンをクリックします。



ログイン方法（ログインプロバイダ）の選択画面になります。最初は、「メール/パスワード」を選択します。

The screenshot shows the Firebase console's Authentication section for a project named "tshirtsfreemarket". The "Sign-in method" tab is selected. A modal window titled "ログイン方法を追加して Firebase Auth の利用を開始しましょう" (Let's add a login method to start using Firebase Auth) is open. It lists several sign-in providers under three categories: "ネイティブのプロバイダ" (Native providers), "追加のプロバイダ" (Additional providers), and "カスタム プロバイダ" (Custom providers). The "メール / パスワード" (Email / Password) provider is highlighted with a red border.

「メール/パスワード」を有効にし [保存] ボタンをクリックします。

The screenshot shows the same Firebase Authentication screen after enabling the "Email / Password" provider. The "有効にする" (Enable) switch for "メール / パスワード" is now turned on (blue). The "保存" (Save) button at the bottom right of the modal window is highlighted with a red border.

「新しいプロバイダ追加」ボタンをクリックします。

The screenshot shows the Firebase Authentication settings page. The 'Sign-in method' tab is selected. A single provider, 'Email / Password', is listed with a status of 'Enabled' (indicated by a green checkmark). A red box highlights the 'Add new provider' button at the top right of the provider list.

以上で「メール/パスワード」での認証が可能になりました。

♣ 2.1.2 Firebase 認証とは？

作成したプロジェクトに対して、

- メール/パスワードでのアカウント新規作成・ログイン
- Google アカウントでのログイン
- twitter、AppleID、GitHubなどのSNS サービスでのログイン

が出来ます。

ログイン状態になると、そのプロジェクトに追加したサービスの細かなアクセス制限ができます。

通常のアプリケーションで認証サービスを実現しようとすると、セキュリティが問題になりますが Firebase Authentication で解放されます。

また、クライアントプログラムも各言語用のライブラリが提供されています。

次の章からは、オンライン・フリーマーケットの認証部分を作成していきます。

第3章

アプリケーション認証

オンライン・フリーマーケットの認証部分を作成します。

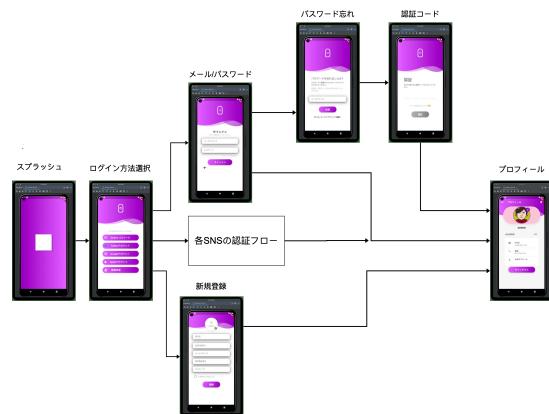
認証フローにある画面の作成と認証コードを作成します。

【この章の内容】

3.1 認証フロー	24
---------------------	----

3.1 認証フロー

下図のように認証を行います。



♣ 3.1.1 スプラッシュスクリーンからログイン選択

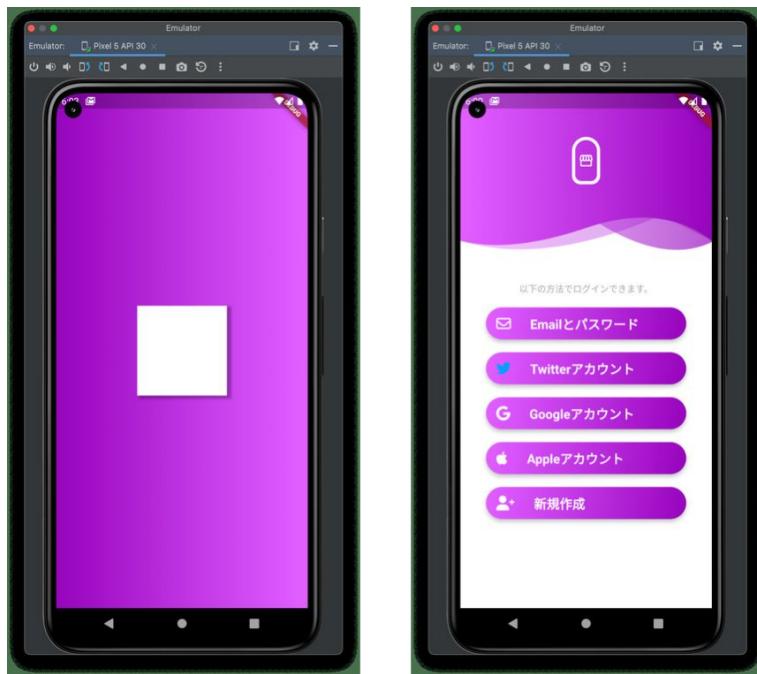
アプリを起動後初期化し、ログイン選択画面を表示します。

サインイン方法は、

- メール/パスワード 新規登録後に利用可能
- Twitter アカウント
- Google アカウント
- AppleID

が選択できます。

「メール/パスワード」でのサインインは、新規登録を行うことで有効になります。



♣ 3.1.2 新規登録

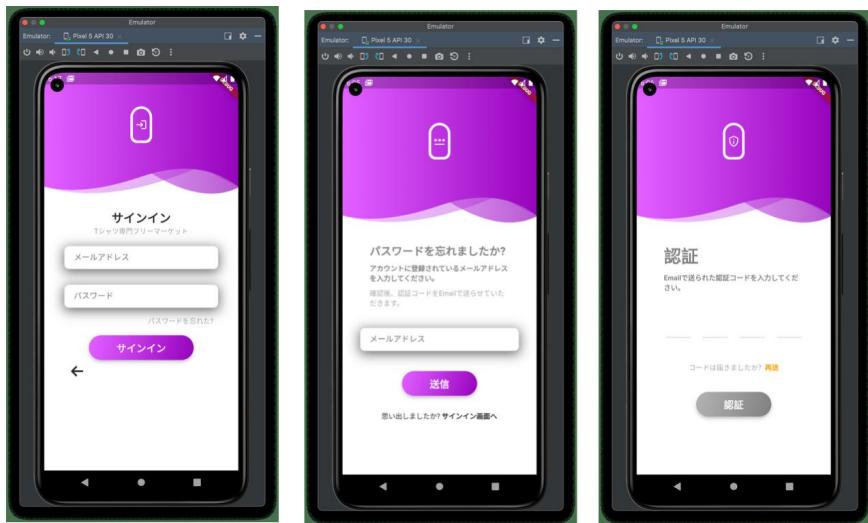
メールアドレス、パスワードなど必要な情報を入力しアカウントを作成します。作成後は、プロフィール画面が表示されます。

アバターの画像は、初回ランダムに選択しています。プロフィール画面で編集することができます。



♣ 3.1.3 メール/パスワード

新規登録が完了すると次回からはメール/パスワードでサインインすることができます。パスワードを忘れた場合の復旧方法は、メールアドレス入力後、送られてきた認証コードを入力することで復旧します。



▲図3.1: desc

♣ 3.1.4 SNS 認証

本書では、

- Twitter
- Google
- Apple

のアカウントを使用してのサインインを実装します。

Twitter、Apple に関しては双方とも Developer アカウントが必要になります。Twitter は無償で Developer アカウント登録できますが、Apple デベロッパーアカウントは有料となります。

どちらも Developer アカウントでログインし、キーの取得などが必要ですがすべて図解しています。



♣ 3.1.5 プロフィール画面

認証完了後は、どのページに飛ばすことも可能ですが、認証フロー確認のためプロフィール画面を表示します。

アバターの画像は、新規登録の場合、事前に保存してある画像からランダムに選択しています。SNS 認証の場合は、SNS に登録してあれば優先されます。



それでは、実装していきましょう。

第 4 章

プロジェクトスタート

Andoroid Studio にて新規 Flutter プロジェクトを作成します。
わずか数クリックで実際に実機でも動作するアプリケーションができることに驚きます。

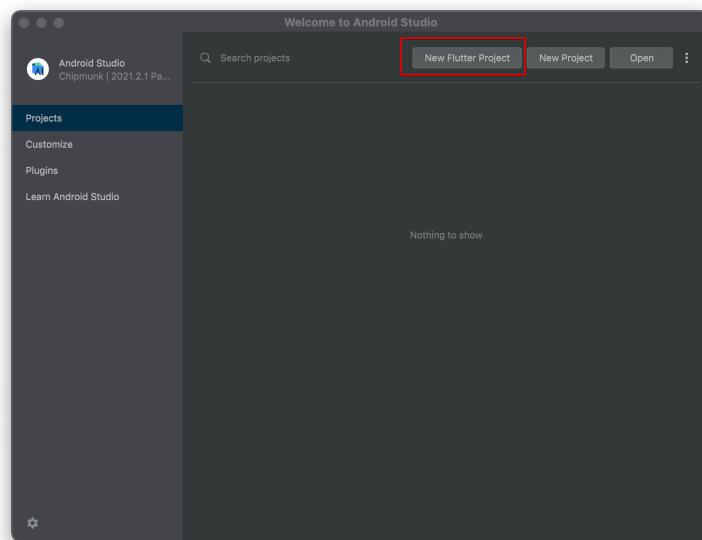
【この章の内容】

4.1 新規プロジェクトの作成	30
---------------------------	----

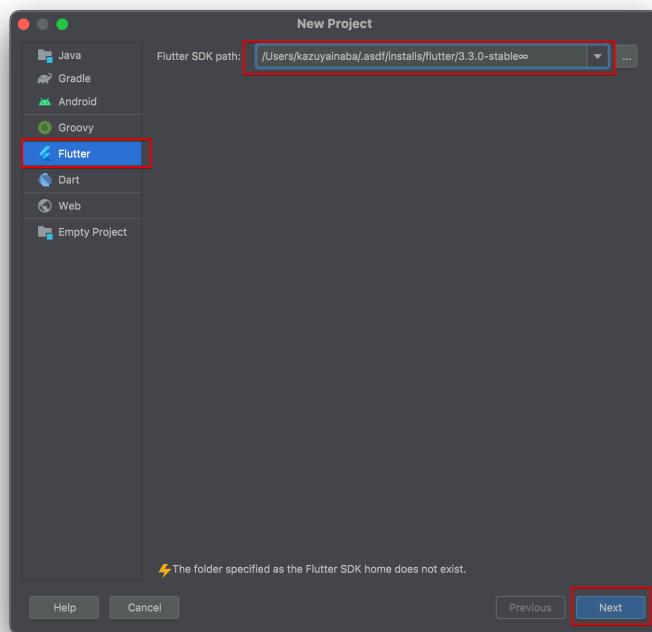
4.1

新規プロジェクトの作成

Android Studio を起動し、[New Flutter Project] ボタンをクリックします。



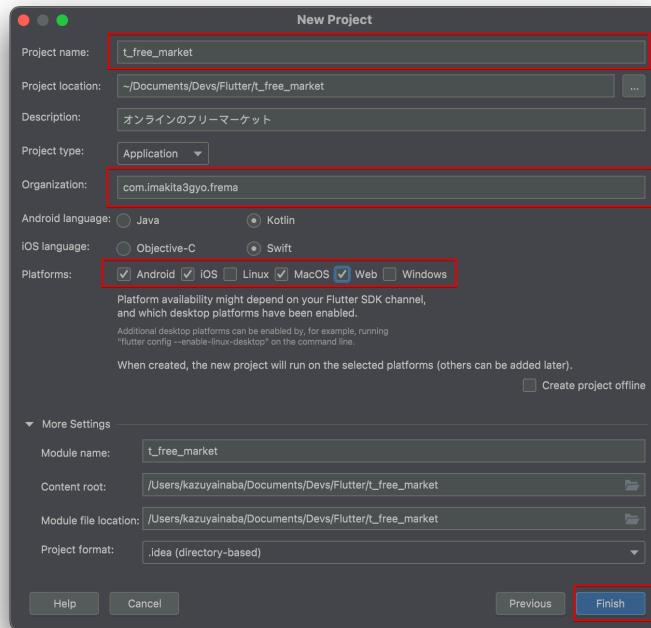
Flutter SDK の場所を訊かれますので、指定するか選択します。



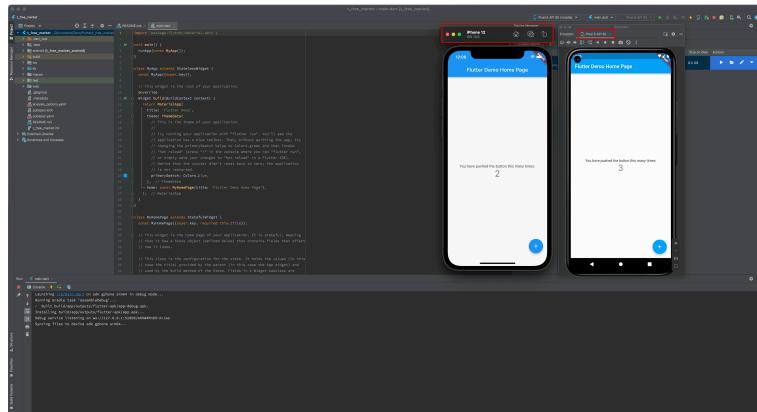
プロジェクトの名前などを設定します。

- プロジェクト名はハイフンを入れるとメンドウになるのでアンダースコアで。
- Organizationですが、Apple デベロッパーに登録しているIDを使いましょう。
- Platformの選択です。

以上が完了しましたら、[Finish] ボタンをクリックします。



プロジェクトが作成され表示されます。iPhone シミュレータを起動しデバッグモードで起動します。また、作成した Android エミュレータでもデバッグしてみます。



ここまででは、出来ましたでしょうか？

0円で、モバイルアプリを作つてみるお～。Vol.1

Flutter、Firebase、GetX でステップ毎に解説

2022年8月29日 ver 1.0 (技術書典 13)

著者 今北産業

© 2022 今北産業