

# 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<sup>\\*1</sup>](#) 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

<sup>\*1</sup> [https://youtu.be/5fH6\\_\\_PAi6A](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<sup>\\*2</sup>](#)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。

---

<sup>\*2</sup> [https://github.com/risingforce9zz/book04\\_t-shirts-freemarket/blob/main/README.md](https://github.com/risingforce9zz/book04_t-shirts-freemarket/blob/main/README.md)

# 目次

はじめに	i
<b>第1章 状態管理とは？</b>	<b>1</b>
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
<b>第2章 Firebase とは？</b>	<b>14</b>
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
<b>第3章 アプリケーション認証</b>	<b>23</b>
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
<b>第4章 プロジェクトスタート</b>	<b>29</b>
4.1 新規プロジェクトの作成	30
<b>第5章 認証関連画面の作成</b>	<b>33</b>
5.1 UIについて	34
5.2 ログインプログラム	35
5.2.1 元プロジェクトの確認	35
5.2.2 プロジェクトへ適用	36
<b>第6章 アプリで認証するユーザのデータ型を作成する。</b>	<b>39</b>
6.1 ユーザモデルの作成	40
<b>第7章 Flutter 側で Firebase が使えるようにする。</b>	<b>42</b>
7.1 クライアント側（Flutter）で Firebase の設定	43

7.2	Firebase サービス用 Flutter プラグインのインストール . . . . .	47
7.3	テスト . . . . .	49
<b>第 8 章</b>	<b>状態管理コントローラーを作る</b>	<b>50</b>
8.1	GetX コントローラー . . . . .	51
8.2	認証コントローラーの作成 . . . . .	53
8.2.1	GetXController について . . . . .	53
8.2.2	AuthController の作成 . . . . .	54
<b>第 9 章</b>	<b>新規登録</b>	<b>58</b>
9.1	新規登録ができるようにする . . . . .	59
9.2	新規登録画面の修正 . . . . .	60
9.2.1	新規登録画面の修正 . . . . .	60
9.2.2	スプラッシュスクリーンで認証コントローラーを初期化 . . . . .	68
9.2.3	新規登録メソッドの実装 . . . . .	71
9.2.4	ページ推移のための Route 作成 . . . . .	72
9.2.5	動作確認 . . . . .	73

---

# 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<sup>\*1</sup> 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。

<sup>\*1</sup> <https://docs.flutter.dev/development/data-and-backend/state-mgmt/options>

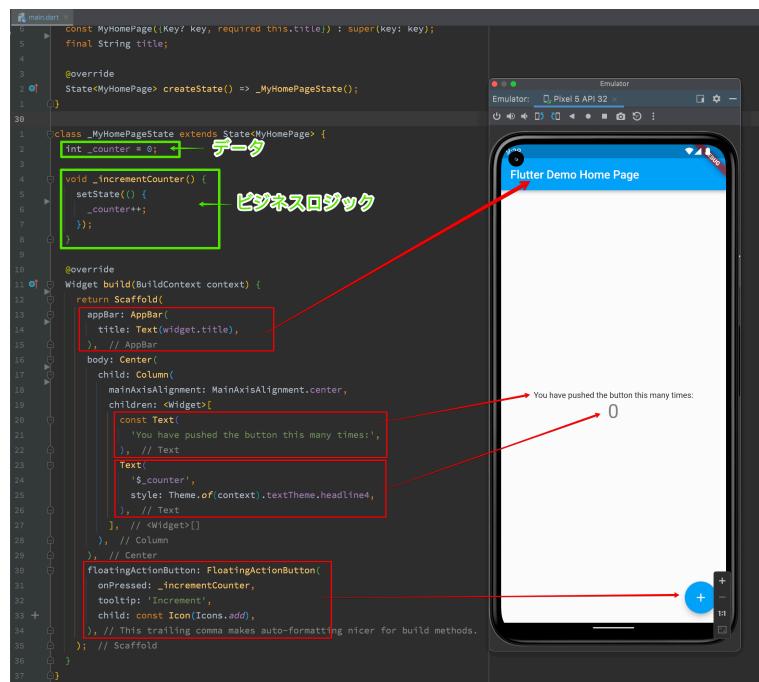
## 1.3

## 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、

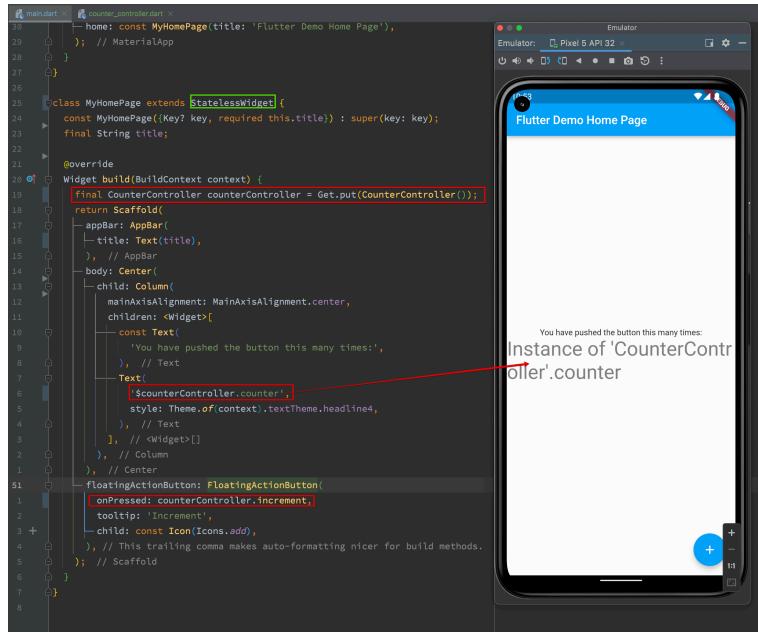
- 状態を保持する `State` を継承した状態管理 `setState` を使用
- データである `_counter` を持っている
- データを操作する `_incrementCounter` ビジネスロジックを持っている。
- データを表示する `Text Widget` を持っている。

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

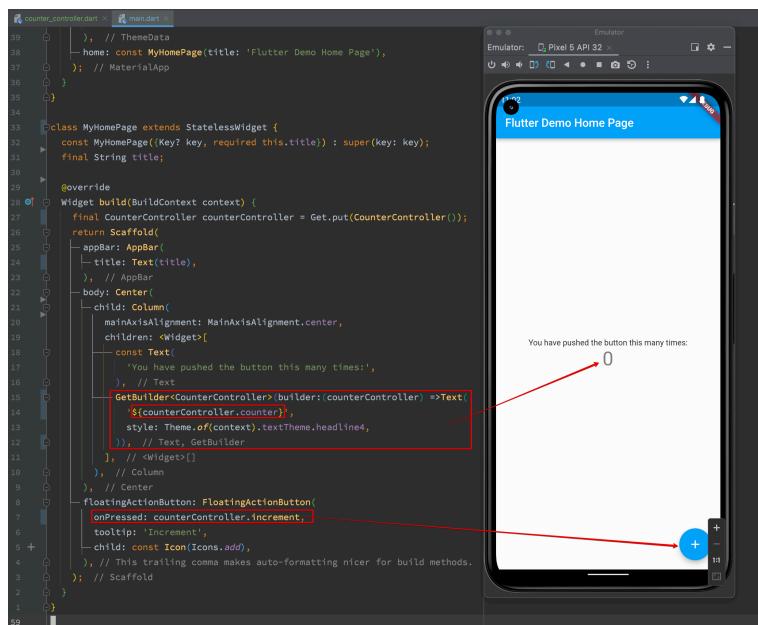
GetX（あくまで本書では）を導入し状態管理を行います。

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 two files: `main.dart` and `counter_controller.dart`. In `main.dart`, the `MyHomePage` class extends `StatelessWidget` and contains a `build` method that creates a `Scaffold` with an `AppBar` and a `body` containing a `Center` widget with a `Column`. The `Column` has 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. In the `build` method, a `final CounterController counterController = Get.put(CounterController());` line is used to inject the controller. On the right, the emulator shows the app running on a Pixel 5 device, displaying the text "You have pushed the button this many times:" followed by "Instance of 'CounterController'.counter". The code editor highlights the `counterController.counter` reference in red, indicating a potential error or warning.

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 the code has been modified. In `main.dart`, the `Text` widget that previously contained the controller reference is now wrapped in a `GetBuilder` widget. The `GetBuilder` widget takes a `builder` parameter that returns a `Text` widget with the controller reference. This change addresses the error shown in the previous screenshot. The emulator on the right shows the updated app running on a Pixel 5 device, displaying the text "You have pushed the button this many times:" followed by "0". The code editor highlights the `builder` parameter of the `GetBuilder` widget in red, likely indicating a warning about the deprecated `builder` parameter.

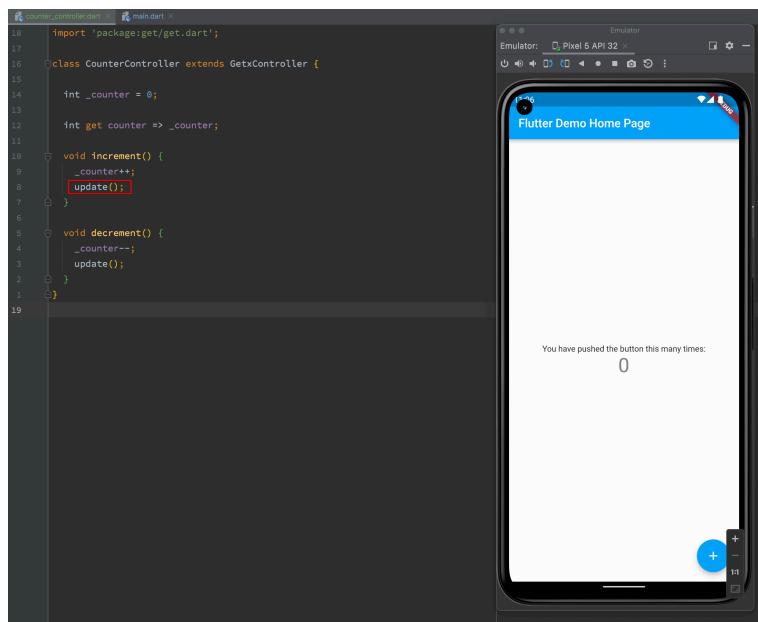
表示部に状態管理「GetBuilder」を適用するとデータが表示されています。実は、GetBuilder自体は、 StatefulWidget を継承していて GetBuilder 自体が状態を保持しています。

### ▼ 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ます。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？

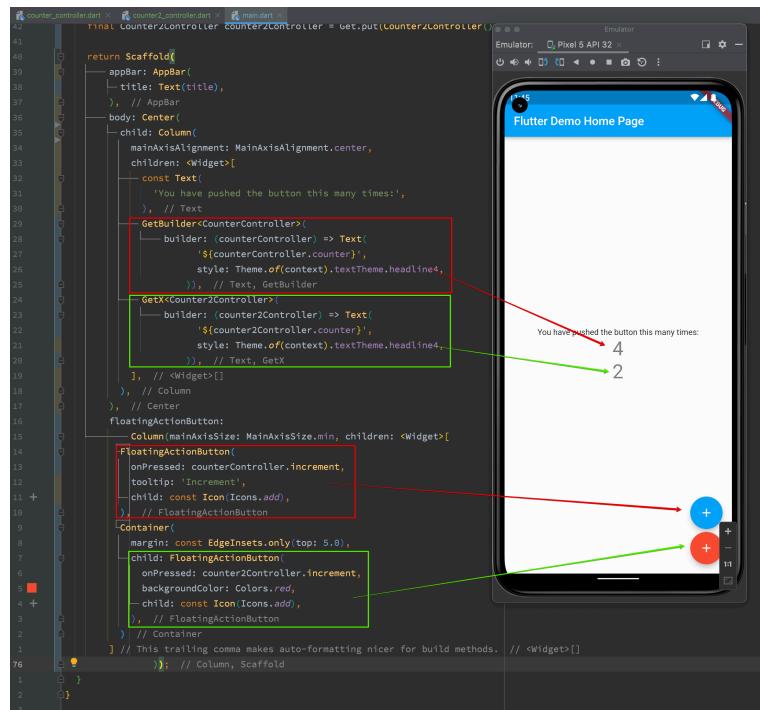
1. データとビジネスロジックを一ヵ所にまとめたので、変更があっても一ヵ所。
  2. 状態管理コントローラーを増やしても、表示部のコード変更は最小です。
  3. 別なページにデータを簡単に渡せる。
  4. データと表示ページには関連がないので、ページを再表示してもデータ再取得がない。

と、たくさんのメリットがあります。

表示を増やしてみた

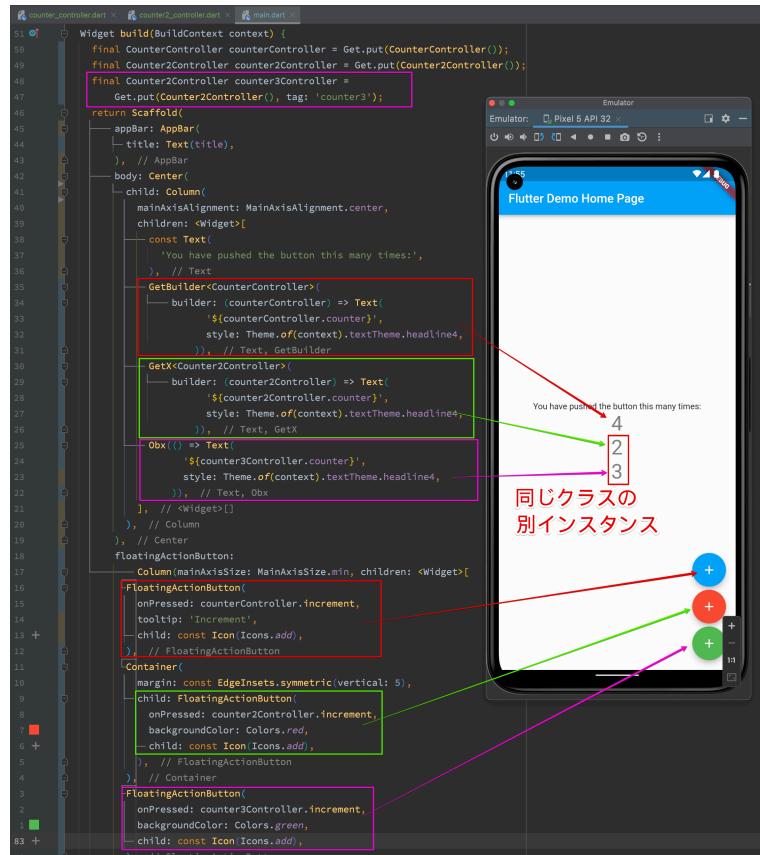
表示部を2つ、それぞれを操作するボタンを2つにしましたが、状態管理コントローラーをひとつ増やしただけです。こちらには、優秀な「GetX<コントローラー型>」を使って表示している

ます。



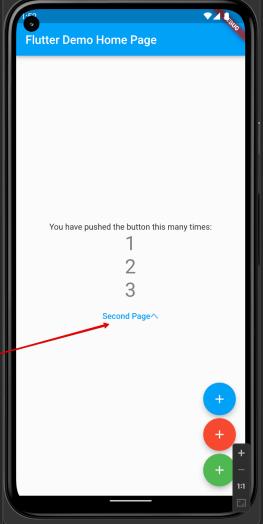
## 状態管理コントローラーの別インスタンス

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。



Flutter Demo Home Page

You have pushed the button this many times:

1  
2  
3

Second Page^

```

1 counter_controller.dart
2 import 'package:flutter/material.dart';
3 import 'package:get/get.dart';
4
5 import '../controllers/counter2_controller.dart';
6
7 class CounterController extends GetxController {
8   int counter = 0;
9
10   void increment() => counter++;
11
12   void reset() => counter = 0;
13 }
14
15 final CounterController counterController = Get.put(CounterController());
16 final Counter2Controller counter2Controller =
17   Get.put(Counter2Controller(), tag: 'counter2');
18 final Counter3Controller counter3Controller =
19   Get.put(Counter3Controller(), tag: 'counter3');
20
21 return Scaffold(
22   appBar: AppBar(
23     title: Text('title'),
24   ),
25   body: Center(
26     child: Column(
27       mainAxisAlignment: MainAxisAlignment.center,
28       children: <Widget>[
29         const Text(
30           'You have pushed the button this many times:',
31         ),
32         Text(
33           '${counterController.counter}',
34           style: Theme.of(context).textTheme.headline4,
35         ),
36         GetX<Counter2Controller>(
37           builder: (counter2Controller) => Text(
38             '${counter2Controller.counter}',
39             style: Theme.of(context).textTheme.headline4,
40           ),
41         ),
42         Obx(() => Text(
43           '${counter3Controller.counter}',
44           style: Theme.of(context).textTheme.headline4,
45         )),
46         TextButton(
47           child: const Text('Second Page^'),
48           onPressed: () => Get.to(() => const SecondPage()),
49         ),
50       ],
51     ),
52   ),
53   floatingActionButton:
54     Column(mainAxisSize: MainAxisSize.min, children: <Widget>[
55       FloatingActionButton(
56         heroTag: 'counter1',
57         onPressed: counterController.increment,
58         tooltip: 'Increment',
59         child: const Icon(Icons.add),
60       ),
61       Container(
62         margin: const EdgeInsets.symmetric(vertical: 5),
63         child: FloatingActionButton(
64           heroTag: 'counter2',
65           onPressed: counter2Controller.increment,
66           backgroundColor: Colors.red,
67           child: const Icon(Icons.add),
68         ),
69       ),
70     ]),
71   floatingActionButtonCenter:
72     FloatingActionButton(
73       heroTag: 'counter3',
74     ),
75 )
76
77 
```



second page

counter2Controllerの値

2  
3

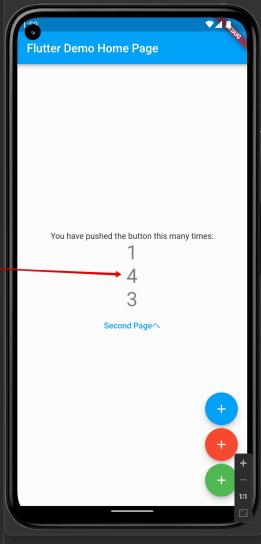
```

1 counter_controller.dart
2 import 'package:flutter/material.dart';
3 import 'package:get/get.dart';
4
5 import '../controllers/counter2_controller.dart';
6
7 class SecondPage extends StatelessWidget {
8   const SecondPage({Key? key}) : super(key: key);
9
10 @override
11 Widget build(BuildContext context) {
12   final Counter2Controller counter2Controller = Get.find();
13   final CounterController counter3Controller = Get.find(tag: 'counter3');
14
15   return Scaffold(
16     appBar: AppBar(
17       leading: IconButton(
18         icon: const Icon(Icons.arrow_back),
19         onPressed: () => Get.back(),
20       ),
21       title: const Text('second page'),
22     ),
23     body: Center(
24       child: Column(
25         mainAxisAlignment: MainAxisAlignment.center,
26         children: <Widget>[
27           const Text('counter2Controllerの値'),
28           GetX<Counter2Controller>(
29             builder: (counter2Controller) => Text(
30               '${counter2Controller.counter}',
31               style: Theme.of(context).textTheme.headline4,
32             ),
33           ),
34           Obx(() => Text(
35             '${counter3Controller.counter}',
36             style: Theme.of(context).textTheme.headline4,
37           )),
38         ],
39       ),
40     ),
41     floatingActionButton: FloatingActionButton(
42       onPressed: counter2Controller.increment,
43       backgroundColor: Colors.red,
44       child: const Icon(Icons.add),
45     ),
46   );
47 }
48 
```

ここでデータを操作し変化させます。

```
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                 '$counter2Controller.counter',
71                 style: Theme.of(context).textTheme.headline4,
72               ), // Text, GetX
73               Obx(() => Text(
74                 '$counter3Controller.counter',
75                 style: Theme.of(context).textTheme.headline4,
76               )), // Text, Obx
77             ], // <Widget>[]
78           ), // Column, Center
79           floatingActionButton: FloatingActionButton(
80             onPressed: counter2Controller.increment,
81             backgroundColor: Colors.red,
82             child: const Icon(Icons.add),
83           ), // FloatingActionButton
84         ); // Scaffold
85       }
86     );
87   }
88 }
```

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。



```

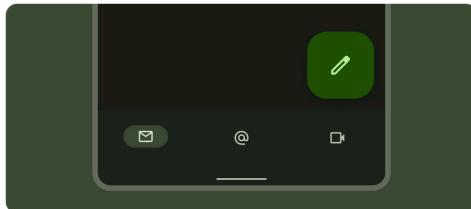
52   final Counter2Controller counter2Controller =
53     Get.put(Counter2Controller(), tag: "counter2");
54
55   final Counter3Controller counter3Controller =
56     Get.put(Counter3Controller(), tag: "counter3");
57
58   return Scaffold(
59     appBar: AppBar(
60       title: Text(title),
61     ),
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。

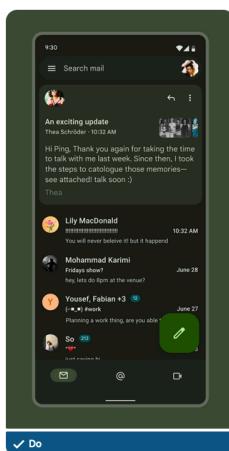
Flutter で推奨されている Material Design では、FAB が複数あることを推奨していません。今回は、状態管理の解説のため簡易的にです。

## FAB

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



A FAB should present a screen's primary action.



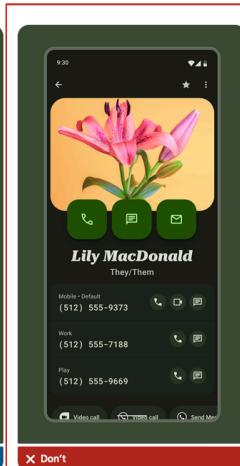
✓ Do

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
------------------------------	----

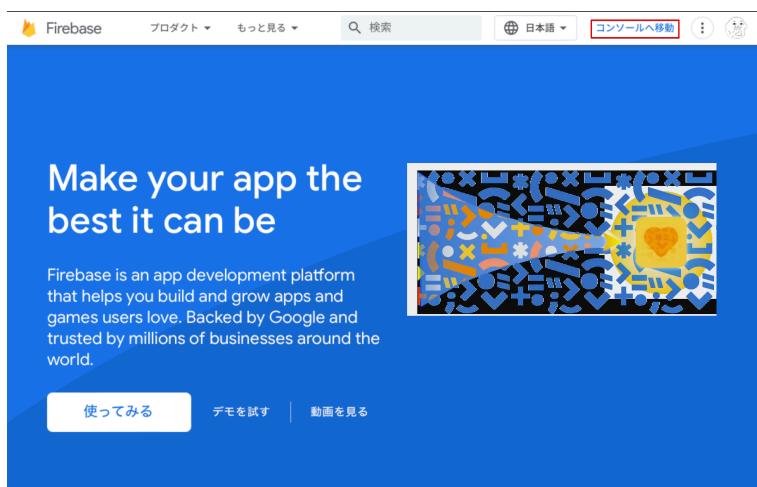
## 2.1 Firebase を使ってみる

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	
<b>Monthly active users</b>	<b>50k/month</b>
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:

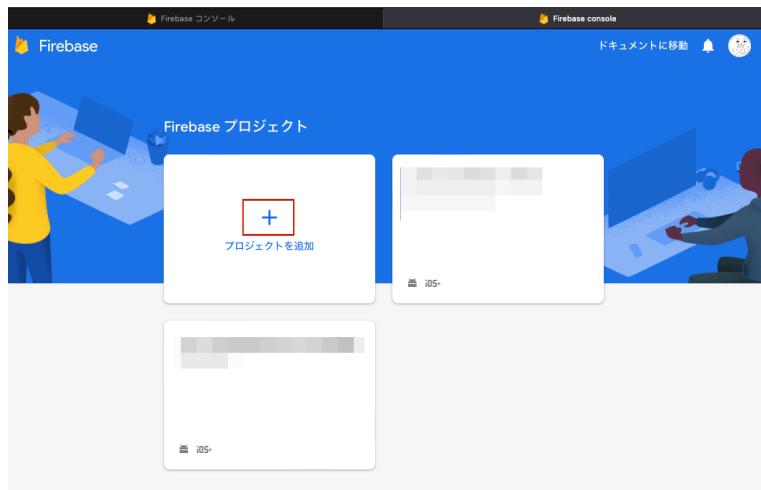
サービス	上限
データ保存	1GB
データ転送	10GB
データ検索	10TB
データ削除	1TB
データ同期	1TB
<b>Monthly active users</b>	<b>50k/month</b>
Monthly active users - SAML/OIDC	50/month

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。

× プロジェクトの作成（手順 2/3）

## Google アナリティクス (Firebase プロジェクト向け)

Google アナリティクスは無料かつ無制限のアナリティクス ソリューションです。これにより、Firebase Crashlytics、Cloud Messaging、アプリ内メッセージング、Remote Config、A/B Testing、Cloud Functions で、ターゲティングやレポートなどが可能になります。

Google アナリティクスによって以下の機能が有効になります。

- A/B テスト ②
  - イベントベースの Cloud Functions ト リガー ②
  - Firebase プロダクト全体でのユーザー ②
  - セグメンテーションとターゲティング
  - 無料で無制限のレポート ②
  - クラッシュに遭遇していないユーザー ②
  - 数
- このプロジェクトで Google アナリティクスを有効にする  
おすすめ

前へ

次へ

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。

× プロジェクトの作成（手順 3/3）

## Google アナリティクスの構成

Google アナリティクス アカウントを選択または作成します ②

このアカウントに新しいプロパティを自動的に作成します

プロジェクトを作成すると、選択した Google アナリティクス アカウントに新しい Google アナリティクス プロパティが作成され、Firebase プロジェクトにリンクされます。このリンクにより、両サービスの間でデータをやり取りできるようになります。  
Google アナリティクスのプロパティから Firebase にエクスポートされるデータには Firebase の利用規約が適用され、Google アナリティクスにインポートされる Firebase のデータには Google アナリティクスの利用規約が適用されます。[詳細](#)

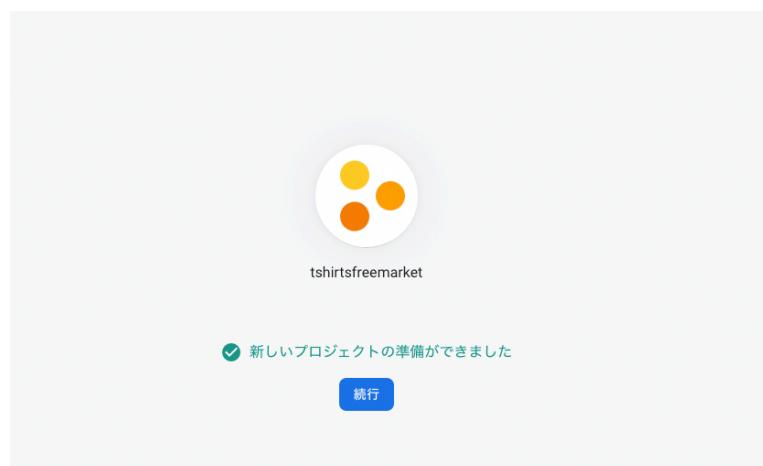
前へ

プロジェクトを作成

作成中です。



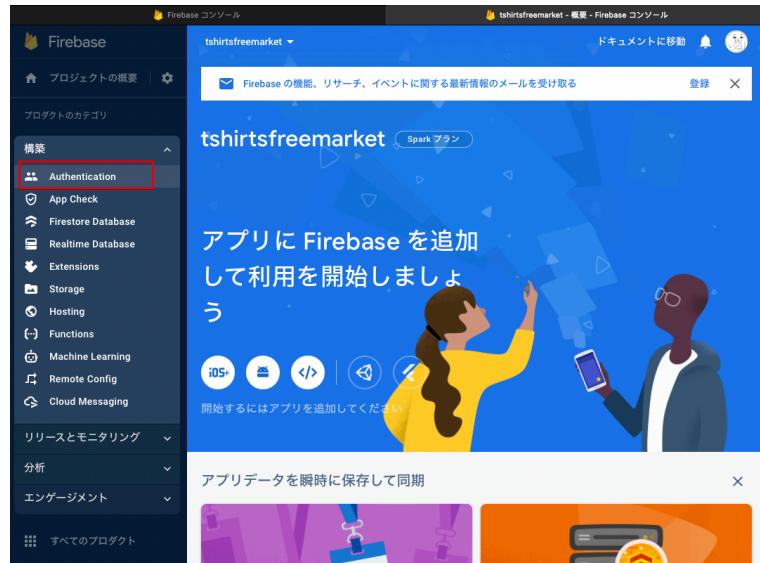
プロジェクトが出来上りました。



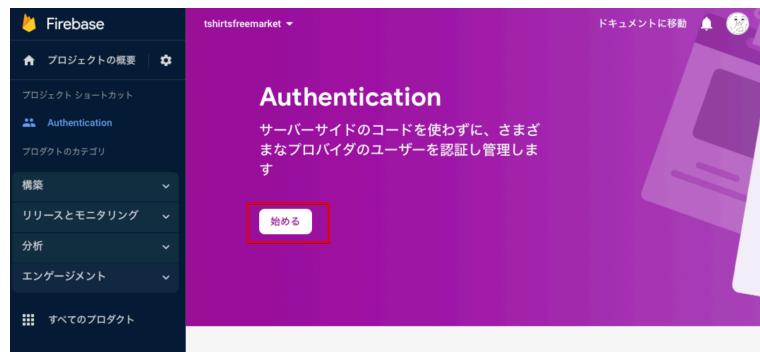
サイドバーにある [構築] ドロップダウンを開くと、追加できるサービスが表示されます。



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" sign-in method. The "メール / パスワード" provider now has a blue toggle switch labeled "有効にする" (Enable) with a red border. Below it, another toggle switch for "メールリンク (パスワードなしでログイン)" (Email Link (Login without password)) is shown. At the bottom right of the modal is a blue "保存" (Save) button 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。

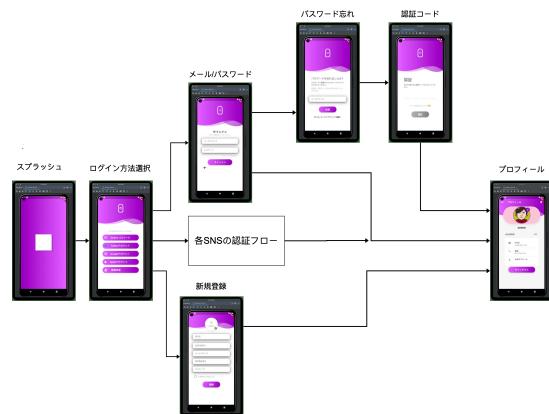
認証フローにある画面の作成と認証コードを作成します。

### 【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

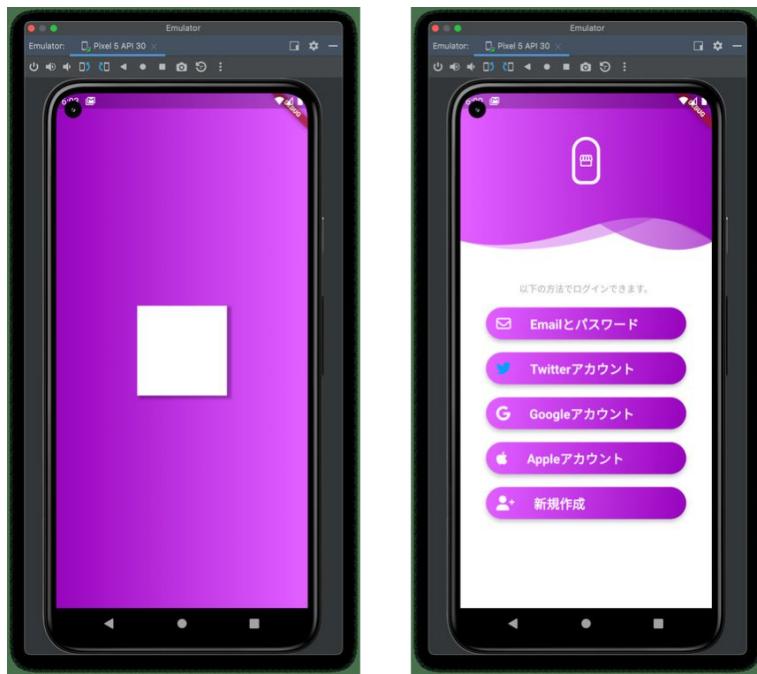
アプリを起動後初期化し、ログイン選択画面を表示します。

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

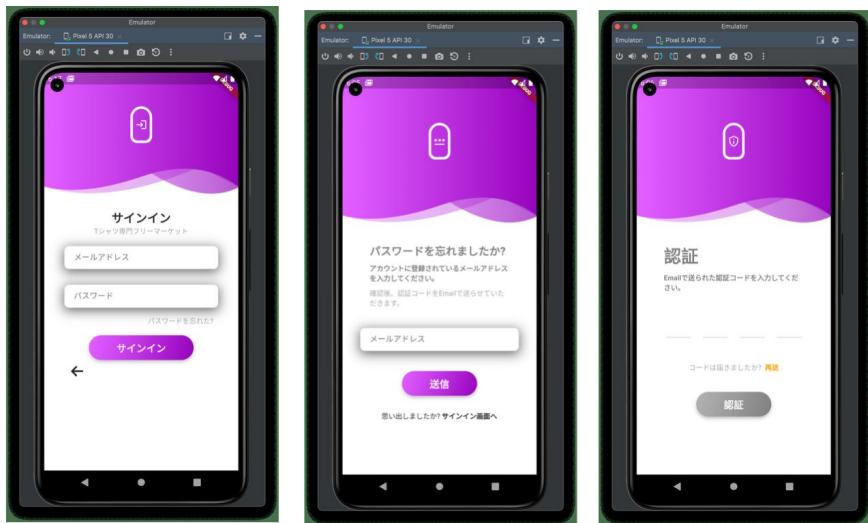
メールアドレス、パスワードなど必要な情報を入力しアカウントを作成します。作成後は、プロフィール画面が表示されます。

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/パスワードデ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、送られて来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。

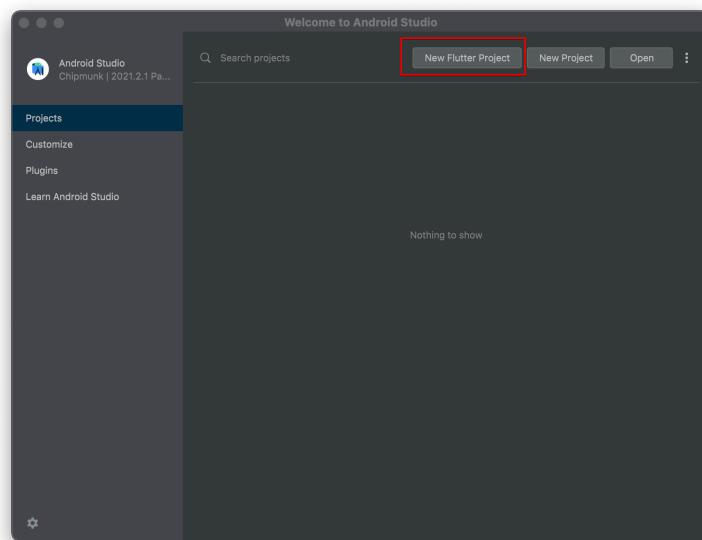
### 【この章の内容】

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

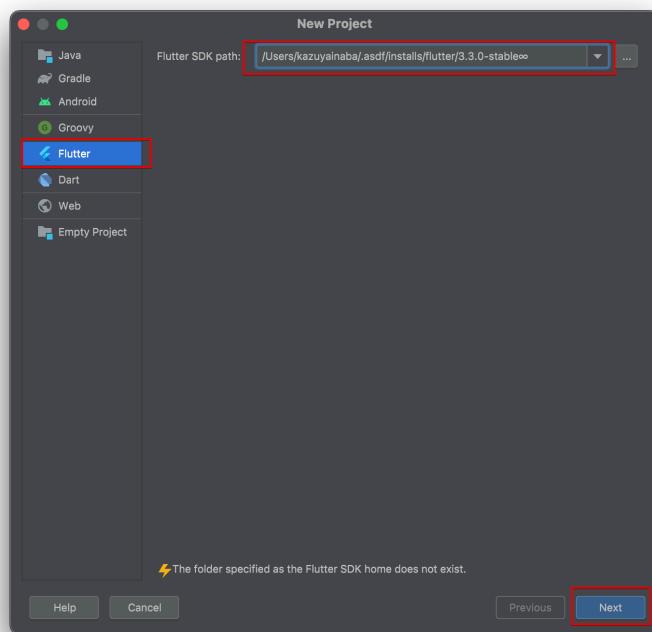
## 4.1

## 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。



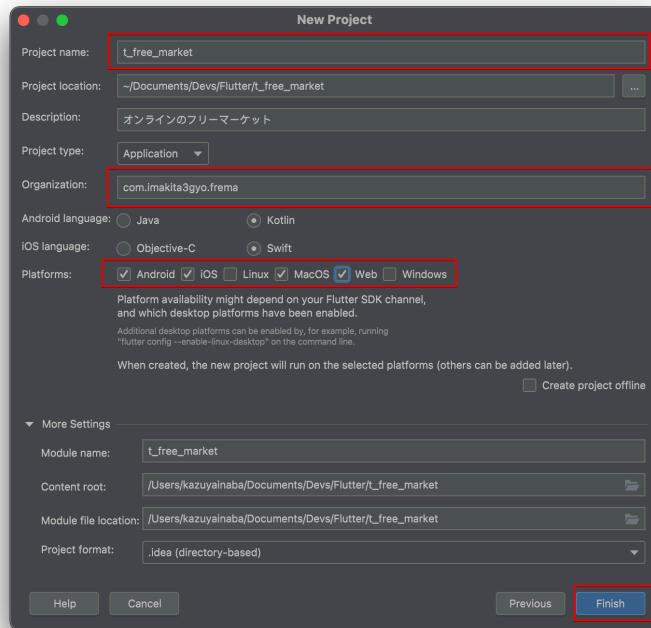
Flutter SDK の場所を訊かれますので、指定するか選択します。



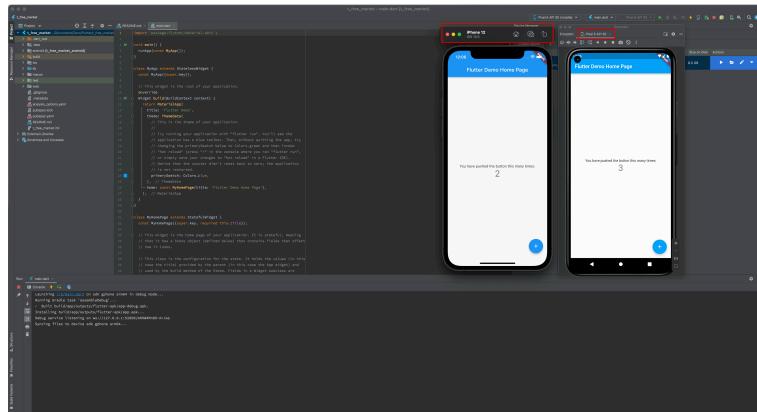
プロジェクトの名前などを設定します。

- 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？

---

# 第 5 章

## 認証関連画面の作成

---

Flutter で作成する UI 部分は、すべて Widget と呼ばれる部品を組み合わせて作成します。

HTML では、<div>タグの中に複数の HTML タグを入れて UI を組み立てます。入れ子になった HTML タグもあります。

Flutter も同じように Widget を組合せて UI を構築します。

### 【この章の内容】

5.1	UI について . . . . .	34
5.2	ログインプログラム . . . . .	35

## 5.1 UIについて

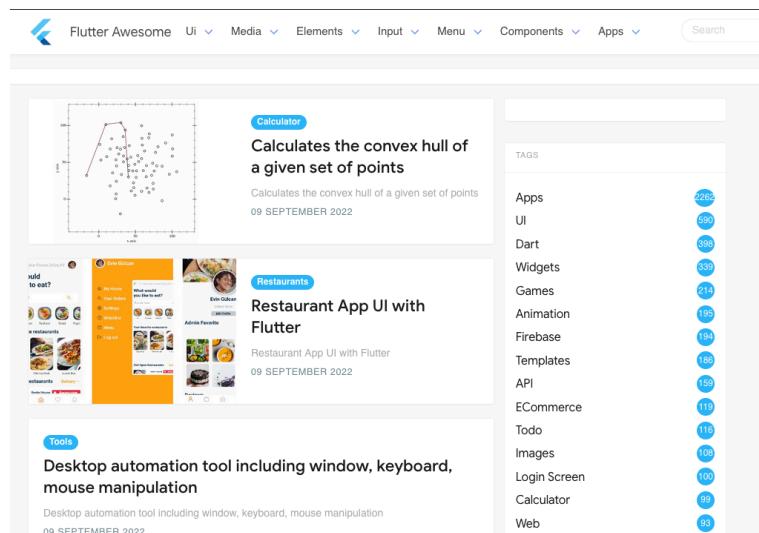
本書では、一画面に表示されるもの全体を指してページと呼びます。

概要でも少し説明していますが、ページは Widget と呼ぶ部品を組み合わせて作成します。Flutter の公式（SDK に含まれている）Widget もありますし、世界中の開発者が作成したものが集めてあるサイトもあります。

もちろん自分で作成することもできます。

色、形、ドロップシャドウなどのデザインに関するものは、ThemeData（HTML で言えば、全体に適用される CSS）が適用され、個別の Widget でもスタイルを適用できます。

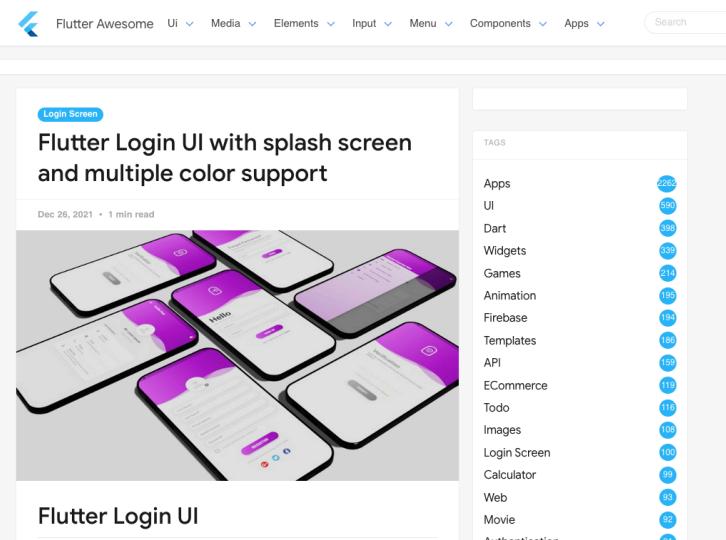
下記のサイト「[Flutter Awesome<sup>\\*1</sup>](#)」では、UI やアプリが沢山公開されており、そのほとんどは GitHub などでソースコードを入手できます。実際に手元で動かし、変更を加えてみれば Widget の使い方も理解が進みます。



\*1 <https://flutterawesome.com/>

## 5.2 ログインプログラム

今回使用するのは、上記 FlutterAwesome で見つけたログイン関連のアプリ **Flutter Login UI**\*<sup>2</sup> です。



このサイトは、Video チュートリアルのリンクもあり、クリックすると Youtube で作成過程を見るすることができます。Youtube の Video チュートリアルの詳細には、同じデザインですが別の GitHub へのリンクがありました。

今回は、[こちら](#)\*<sup>3</sup>を使うことにします。

### ♣ 5.2.1 元プロジェクトの確認

クローンして魔改造しても良いのですが、勉強も兼ねて写経することにします。Android Studio でコードを書くと、linter がエラー（インストールされていないパッケージの Widget など）がでます。そのため、元プロジェクトの「pubspec.yaml」を開きインストールされているパッケージを確認します。

#### ▼ pubspec.yaml

```
dependencies:
  flutter:
    sdk: flutter
```

\*<sup>2</sup> <https://flutterawesome.com/flutter-login-ui-with-splash-screen-and-multiple-color-support/>

\*<sup>3</sup> [https://github.com/FlutterTutorialNet/flutter\\_login\\_ui](https://github.com/FlutterTutorialNet/flutter_login_ui)

```
# The following adds the Cupertino Icons font to your application.
# Use with the CupertinoIcons class for iOS style icons.
cupertino_icons: ^1.0.2
hexcolor: ^2.0.4
font_awesome_flutter: ^9.1.0
otp_text_field: ^1.1.1
```

使用されているプラグインは、

- hexcolor -- <https://pub.dev/packages/hexcolor>
- font\_awesome\_flutter -- [https://pub.dev/packages/font\\_awesome\\_flutter](https://pub.dev/packages/font_awesome_flutter)
- otp\_text\_field -- [https://pub.dev/packages/otp\\_text\\_field](https://pub.dev/packages/otp_text_field)

の3つです。

## ♣ 5.2.2 プロジェクトへ適用

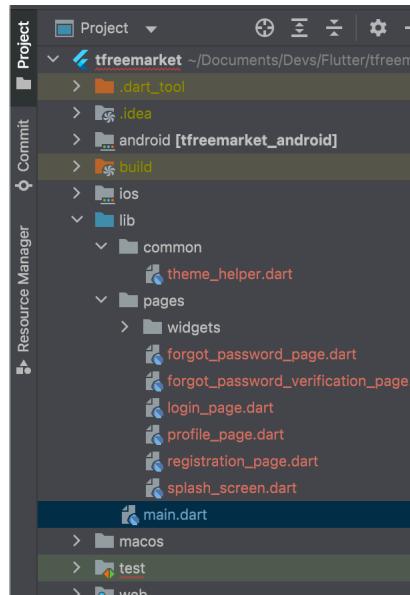
それでは、元プロジェクトをパクっていきましょう。

最初にプラグインをインストールします。

### ▼ プラグインのインストール

```
> flutter pub add hexcolor font_awesome_flutter otp_text_field
Resolving dependencies...
+ flutter_web_plugins 0.0.0 from sdk flutter
+ font_awesome_flutter 10.2.1
+ hexcolor 2.0.7
+ js 0.6.4
  material_color_utilities 0.1.5 (0.2.0 available)
+ otp_text_field 1.1.3
  source_span 1.9.0 (1.9.1 available)
  test_api 0.4.12 (0.4.13 available)
  vector_math 2.1.2 (2.1.3 available)
Downloading font_awesome_flutter 10.2.1...
Changed 5 dependencies!
```

後は、写経するか、コピペするかでファイルを作成します。フォルダ、ファイルの位置は後々変更することとし、今は元プロジェクトと同じにしておきます。

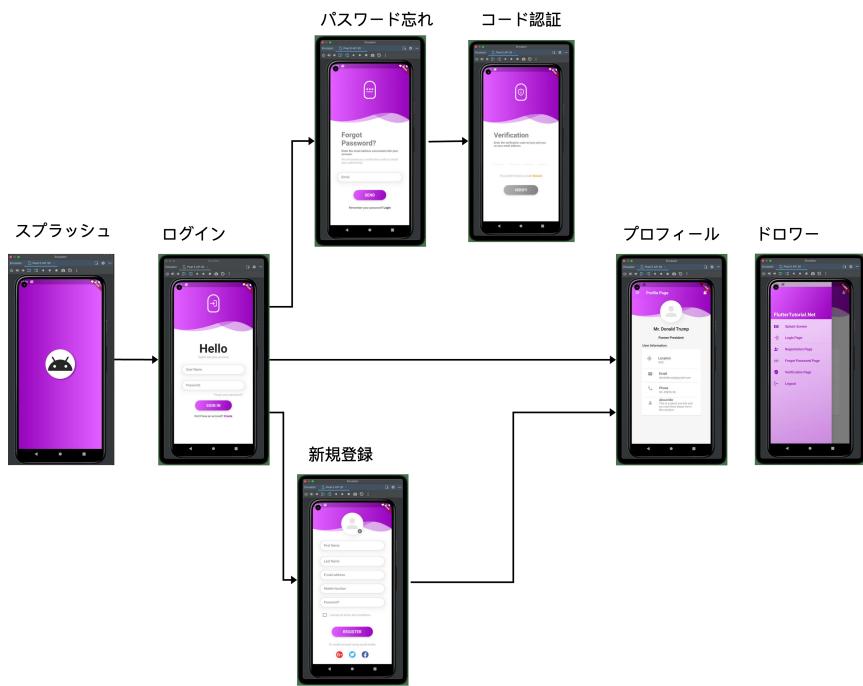


lint で Warning が沢山出ていますが、コントローラーを追加した後すべてのページを、

- StatefulWidget から StatelessWidget へ変更
- タイトル、入力フィールドなどの文字を日本語化

しましので、そのときに修正します。

エミュレータで確認すると、以下のように取り入れることが出来ました。



▲図5.1: desc

ここまでまでのコードは、以下から取得できます。

#### ここまでまでのソースコード

##### ▼ GitHub

```
>git clone -b 01_login_screens_import https://github.com/risingforce9zz/tf>
>reemarket.git
```

---

## 第 6 章

# アプリで認証するユーザのデータ型を作成する。

---

アプリで認証するユーザー (Firebase を使用するユーザとは別) のデータ型を作成します。

#### 【この章の内容】

6.1 ユーザモデルの作成	40
---------------	----

## 6.1 ユーザモデルの作成

アプリで認証するユーザ（今回は、オンライン・フリーマーケットの出店者兼購入者）の必要な情報を検討しデータ型を作成します。

lib フォルダ内に「models」フォルダを作成し、「user\_model.dart」ファイルを作成します。内容は以下となります。

### ▼ ユーザモデル型

```
class UserModel {  
  
    String id = '';  
    String name = '';  
    String displayName = '';  
    String email = '';  
    String phoneNumber = '';  
    String profilePic = '';  
    String storeDescription = '';  
  
    UserModel({  
        required this.id,  
        required this.name,  
        required this.displayName,  
        required this.email,  
        required this.phoneNumber,  
        required this.profilePic,  
        required this.storeDescription,  
    });  
  
    Map<String, dynamic> toMap() {  
        return {  
            'id': id,  
            'name': name,  
            'displayName': displayName,  
            'email': email,  
            'phoneNumber': phoneNumber,  
            'profilePic': profilePic,  
            'storeDescription': storeDescription,  
        };  
    }  
  
    UserModel.fromMap(Map<dynamic, dynamic>? map) {  
        if (map == null) {  
            return;  
        }  
        id = map["id"];  
        name = map['name'];  
        displayName = map['displayName'];  
        email = map['email'];  
        phoneNumber = map['phoneNumber'];  
        profilePic = map['profilePic'];  
    }  
}
```

```
    storeDescription = map['storeDescription'];
}

UserModel copyWith({
  String? id,
  String? name,
  String? displayName,
  String? email,
  String? phoneNumber,
  String? profilePic,
  String? storeDescription,
}) {
  return UserModel(
    id: id ?? this.id,
    name: name ?? this.name,
    displayName: displayName ?? this.displayName,
    email: email ?? this.email,
    phoneNumber: phoneNumber ?? this.phoneNumber,
    profilePic: profilePic ?? this.profilePic,
    storeDescription: storeDescription ?? this.storeDescription,
  );
}
}
```

.....

### ここまで

までのソースコード

▼ GitHub

```
>git clone -b 02_login_screens_import https://github.com/risingforce9zz/tf>
>reemarket.git
```

.....

---

## 第 7 章

# Flutter 側で Firebase が使えるように する。

---

Firebase 関連のプラグインの導入と設定を行います。

#### 【この章の内容】

7.1	クライアント側（Flutter）で Firebase の設定 . . . . .	43
7.2	Firebase サービス用 Flutter プラグインのインストール . . . . .	47
7.3	テスト . . . . .	49

## 7.1

## クライアント側(Flutter)でFirebaseの設定

以前は、モバイルアプリでFirebaseを使うために、

- Firebaseコンソールにアプリを登録して、
- Firebase側が自動作成したファイルを、Android用、iOS用にダウンロード・配置をし、
- Firebaseのサービス毎にプラグインのインストール

が必要でした(プラグインのインストールは今でも必要)。

しかし、最近は「Firebase CLI」があります。これは、ターミナルからコマンドで設定が完了する優れもので、ファイルのダウンロードやRealtime Databaseへのアクセス設定などが簡単にできます。

公式サイト <https://firebase.google.com/docs/flutter/setup?platform=ios#available-plugins>

インストールは、ターミナルを開き、

### ▼ Firebase CLI のインストール

```
> curl -sL https://firebase.tools | bash
```

と、入力しエンターキーを押します。たった、これだけ。続けて、プロジェクトフォルダへ移動し、

### ▼ Firebase にログインし設定

```
> firebase login  
> firebase projects:list
```

これで、Firebaseに登録しているプロジェクトが表示されればOKです。

プロジェクトフォルダで作業していることを再度確認し、

### ▼ flutterfire の有効化

```
> dart pub global activate flutterfire_cli
```

これで、プロジェクトへFirebase関連の設定のほとんどをコマンドで行うことができます。

flutttrefireコマンドが有効になりましたので、早速使ってみます。

### ▼ flutterfire コマンド

```
> flutterfire configure
 _____>
> _____
 The Dart tool uses Google Analytics to report feature usage statistics
 and to send basic crash reports. This data is used to help improve the
 Dart platform and tools over time.

 To disable reporting of analytics, run:

 dart --disable-analytics


 _____>
> _____
- ansi_styles 0.3.2+1s... (1.2s)
- args 2.3.1
- async 2.9.0
- characters 1.2.1
***** 中略 *****
- uri 1.0.0
- win32 2.7.0 (3.0.0 available)
- xml 5.4.1 (6.1.0 available)
- yaml 3.1.1
Downloading flutterfire_cli 0.2.4...
Downloading pub_updater 0.2.2...
Downloading interact 2.1.1...
Downloading deep_pick 0.10.0...
Downloading cli_util 0.3.5...
***** 中略 *****
Downloading async 2.9.0...
Building package executables... (2.1s)
Built flutterfire_cli:flutterfire.
Installed executable flutterfire.
Warning: Pub installs executables into $HOME/.pub-cache/bin, which is not on your path.
You can fix that by adding this to your shell's config file (.bashrc, .bash_profile, etc>..):
export PATH="$PATH":$HOME/.pub-cache/bin"

Activated flutterfire_cli 0.2.4.
```

最後に、パスを通すように指示されましたので、エディッタで「~/.zshrc」を開き

```
export PATH="$PATH":$HOME/.pub-cache/bin"
```

を追加して保存してください。

### ▼ path の設定

```
> source ~/.zshrc
```

で、設定を読み込みます。

Flutter用のFirebaseコアプラグインをインストールします。ターミナルに「flutter pub add firebase\_core」と入力しエンターキーを押します。

- Firebaseに登録しているプロジェクトが表示されるので、上下矢印キーで選択。
- 対象のプラットフォームを選択する。デフォルトで全選択になっているのでエンターキー。
- Androidのコンパイル設定ファイルの上書きを求められるのでエンターキー。

### ▼ Firebase core のインストール

```
☒ flutter pub add firebase_core
Resolving dependencies...

- firebase_core 1.21.1
- firebase_core_platform_interface 4.5.1
- firebase_core_web 1.7.2
material_color_utilities 0.1.5 (0.2.0 available)
- plugin_platform_interface 2.1.2
source_span 1.9.0 (1.9.1 available)
test_api 0.4.12 (0.4.13 available)
vector_math 2.1.2 (2.1.3 available)
Downloading firebase_core 1.21.1...
Downloading firebase_core_web 1.7.2...
Downloading firebase_core_platform_interface 4.5.1...
Changed 4 dependencies!
```

「flutterfireコマンド」で設定を行います。ターミナルに「flutterfire configure」と入力しエンターキーを押します。

コマンドが実行され、

- Firebaseコンソールに登録されたプロジェクト名が表示されるので、設定先を選択する。
- 

### ▼ flutterfire コマンド

```
☒ flutterfire configure
i Found 3 Firebase projects.
```

←ここにプロジェクトが表示されるので、上下矢印キーで選択しエンターキーで決定

```
☒ Select a Firebase project to configure your Flutter application with · プロジェクト名 >
> (プロジェクト名)
```

```
☒ Which platforms should your configuration support (use arrow keys & space to select)?>
> • ios, macos, web, android
```

←ここに設定するプラットフォームが表示される。チェックマークで選択

```
i Firebase android app com.imakita3gyo.frema.t_free_market is not registered on Firebase
> project tshirtsfreemarket.
i Registered a new Firebase android app on Firebase project tshirtsfreemarket.
i Firebase ios app com.imakita3gyo.frema.tFreeMarket is not registered on Firebase project
> tshirtsfreemarket.
i Registered a new Firebase ios app on Firebase project tshirtsfreemarket.
i Firebase macos app com.imakita3gyo.frema.tFreeMarket registered.
i Firebase web app t_free_market (web) is not registered on Firebase project tshirtsfree
> market.
i Registered a new Firebase web app on Firebase project tshirtsfreemarket.
```

←Androidのコンパイル設定ファイルを上書きしても良いか、yes選択状態なのでエンターキー

```
? The files android/build.gradle & android/app/build.gradle will be updated to apply Firebase
> configuration and gradle build plugins. Do you want to continue? (y/n) > yes
```

```
Firebase configuration file lib.firebaseio_options.dart generated successfully with the following
> Firebase apps:
```

```
Platform  Firebase App Id
web        1:65147047369:web:d8f9e2c3b9faa316a9ec18
android    1:65147047369:android:6e7f4e03f4305567a9ec18
ios         1:65147047369:ios:024142971626c8b6a9ec18
macos      1:65147047369:ios:024142971626c8b6a9ec18
```

```
Learn more about using this file and next steps from the documentation:
```

これで、

- アクセス情報の入った「lib.firebaseio\_options.dart」
- Android用設定ファイル「android/app/google-services.json」
- iOS用設定ファイル「ios/Runner/GoogleService-Info.plist」
- macOS用設定ファイル「macos/firebase\_app\_id\_file.json」

が、ダウンロードされます。

Firebaseサービス用のFlutterプラグインをインストールする度に設定を行う必要があるので、Firebase系のプラグインは一度でインストールします。

## 7.2 Firebase サービス用 Flutter プラグインのインストール

続いて、Firebase を使用するためのコアプラグイン、サービス毎のプラグインをインストールします。

Flutter で 使用 で き る Firebase 関 連 プ ラ グ イ ン 一 覧  
<https://firebase.google.com/docs/flutter/setup?platform=ios#available-plugins>

プロジェクトで使用すると思われるプラグインは、

- アナリティクス firebase\_analytics
- 認証 firebase\_auth
- Cloud Firestore（ドキュメント形式のデータベース） cloud\_firestore
- Cloud Messaging firebase\_messaging
- Cloud Storage firebase\_storage
- Realtime Database firebase\_database

です。

ターミナルに、

```
flutter pub add firebase_analytics firebase_auth cloud_firestore firebase_messaging firebase_storage firebase_database
```

を入力しエンターキーを押します。

### ▼ Firebase サービスプラグインのインストール

```
> flutter pub add firebase_analytics firebase_auth cloud_firestore firebase_messaging fi>
rebase_storage firebase_database
...
Changed 22 dependencies!
```

お約束の「flutterfire configure」を行います。ターミナルには前回と同じ表示が出ます。同じように選択しエンターキーを押します。

以上で、Firebase のサービスを使う準備ができました。

最後に、Firebase アナリティクスを使うためには、Andoroid の SDK の最低バージョンが 19 以

上でないとコンパイルが通りません。ここで最低のSDKバージョンを29に指定します。

また、コンパイルSDKバージョンも最新が要求されます。

2022年8月に「33」が正式リリースされました。古いAndroid OSをサポートすると画面解像度の種類も膨大になるため4世代前を仕様とします。

「`android/app/src/build.gradle`」ファイルの以下の項目の下から4行目にある「`minSdkVersion`」を直接してしまします。編集は、コメントアウト・新規行の追加をおこないます。

2行目の「`compileSdkVersion`」を「33」にします。

#### ▼ `android/app/src/build.gradle`

```
android {  
    compileSdkVersion 33  
    ndkVersion flutter.ndkVersion  
  
    compileOptions {  
        sourceCompatibility JavaVersion.VERSION_1_8  
        targetCompatibility JavaVersion.VERSION_1_8  
    }  
  
    kotlinOptions {  
        jvmTarget = '1.8'  
    }  
  
    sourceSets {  
        main.java.srcDirs += 'src/main/kotlin'  
    }  
  
    defaultConfig {  
        // TODO: Specify your own unique Application ID (https://developer.android.com/studio/build/application-id.html).  
        applicationId "com.imakita3gyo.tfreemarket.tfreemarket"  
        // You can update the following values to match your application needs.  
        // For more information, see: #reviewing-the-build-configuration.  
        // minSdkVersion flutter.minSdkVersion  
        minSdkVersion 29 ←この行を追加、上の行をコメントアウト  
        targetSdkVersion flutter.targetSdkVersion  
        versionCode flutterVersionCode.toInteger()  
        versionName flutterVersionName  
    }  
}
```

## 7.3 テスト

起動時にFirebaseクライアントを初期化し、アプリが起動するかを確認します。

「lib/main.dart」を以下のように変更します。

- firebase\_core をインポート
- firebase\_options をインポート
- main を async キーワード付けて非同期にし、Firebase を初期化

### ▼ lib/main.dart

```
import 'package:firebase_core/firebase_core.dart';

import 'firebase_options.dart';
import 'pages/splash_screen.dart';

void main() async {
    WidgetsFlutterBinding.ensureInitialized();
    await Firebase.initializeApp(
        options: DefaultFirebaseOptions.currentPlatform,
    );
    runApp(LoginUiApp());
}
```

編集が完了すると、エミュレータでデバッグします。問題無く起動しましたので次に進みます。

### ここまで

### までのソースコード

#### ▼ GitHub

```
>git clone -b 03_flutterfire_setup https://github.com/risingforce9zz/tfree>
>market.git
```

---

## 第 8 章

# 状態管理コントローラーを作る

---

認証状態を管理する状態コントローラーを作成します。

コントローラーには、管理するデータとデータを変更するメソッドがあります。

#### 【この章の内容】

8.1	GetX コントローラー	51
8.2	認証コントローラーの作成	53

## 8.1 GetX コントローラー

公式サイト <https://pub.dev/packages/get>

GitHub <https://github.com/jonataslaw/getx>

### インストール

ターミナルに「flutter pub add get」と入力しエンターキーを押します。

#### ▼ GetX のインストール

```
☒ flutter pub add get
Resolving dependencies...
+ get 4.6.5
  material_color_utilities 0.1.5 (0.2.0 available)
  source_span 1.9.0 (1.9.1 available)
  test_api 0.4.12 (0.4.13 available)
  vector_math 2.1.2 (2.1.3 available)
Changed 1 dependency!
```

GetX は、状態管理以外にもできることがあります。その機能が必要になった時点で個別に説明しますが、

- State management 状態管理
- dependency\_anagement 依存管理
- route management ルート管理

が、大きな機能です。

その他にも、国際化対応、検証（validation）などがあります。

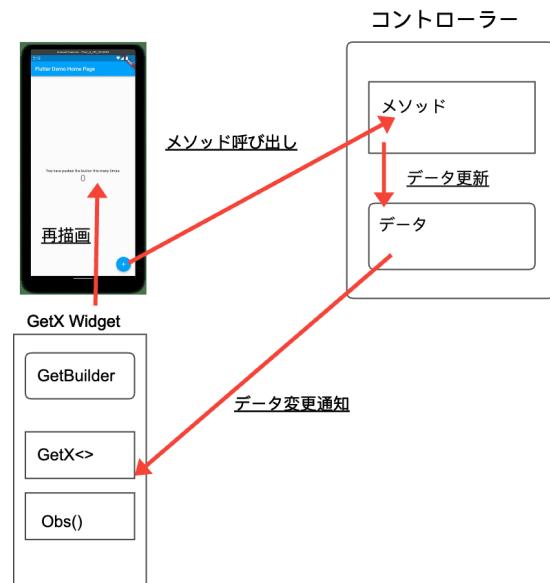
GetX の特徴は、下図のよう、

- コントローラーは、データとデータ更新用メソッドを持つ。
- 表示は、GetX の Widget にコントローラーのデータを表示させる。
- ボタンなど操作部は、コントローラーのメソッドを呼び出す。

と配置され、

1. 操作部からコントローラーのメソッドが呼ばれる。
2. コントローラー内でメソッドが実行され、データ更新。
3. GetX の仕組みで表示用 Widget にデータ変更の通知。
4. 表示用 Widget が再描画。

と、流れます。

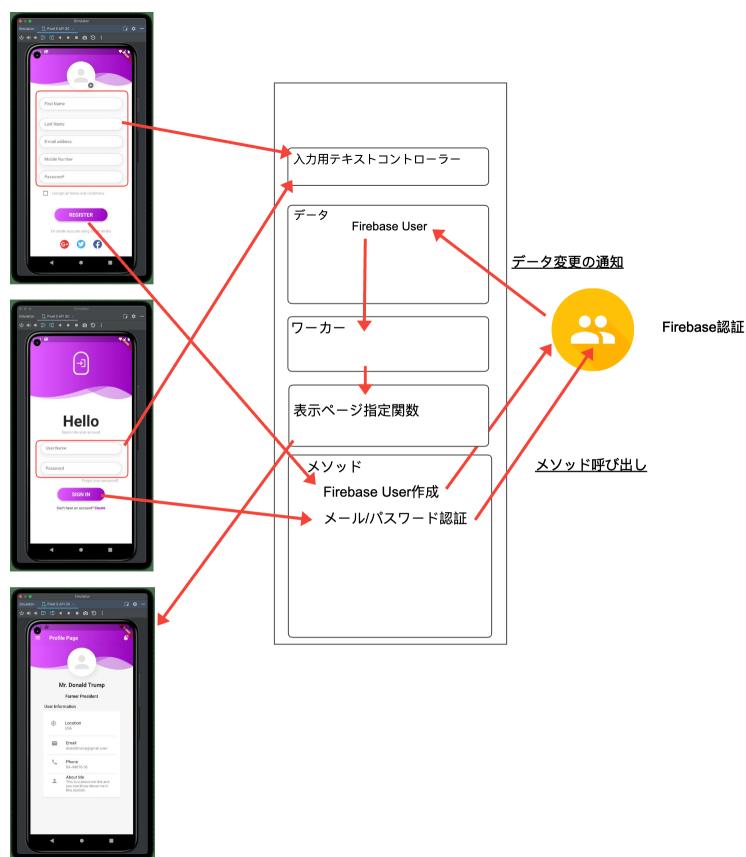


## 8.2

## 認証コントローラーの作成

認証コントローラー (authController) は、以下のようなフローで動作します。

- 画面のテキストフィールドは、コントローラーが持つ。
- サインインなどのメソッドもコントローラーが持つ。
- コントローラーは、Firebase User データを持つ。
- Firebase User データは、認証状態が変わると通知を受ける。
- GetX には、イベント発生時にコールバックを呼び出す Worker がある。
- コールバックで表示ページを切り替える。



### ♣ 8.2.1 GetXController について

GetXController 作成前に、もう少し GetXController について知っておくことがあります。

**GetX コントローラーはライフサイクルがある。**

GetX コントローラーには、ライフサイクルがあり以下のイベント発生時にメソッド呼び出しが

できます。

- `onInit` 初期化されたとき
- `onReady` 準備完了したとき
- `onClose` 終了するとき

GetX コントローラーには、Worker と呼ばれる仕組みがあり、監視対象が変更されるとコールバックを呼び出せる。

- `ever(監視対象, コールバック)` 監視対象が更新されるたびにコールバックされる。
- `once((監視対象, コールバック))` 監視対象が更新と一度だけコールバックされる。
- `debounce(監視対象, コールバック, time: 時間の長さ)` 監視対象が更新され、経過時間後にコールバックされる。
- `interval(監視対象, コールバック, time: 時間の長さ)` 指定した時間内の監視対象が更新は無視する。

GetX コントローラーが持つデータは、監視対象（更新されると通知する）と通常の変数がある。

### ▼ 監視対象

```
// 監視対象のデータに「.obs」を付ける。  
var checkBoxVal = false.obs;  
  
// Rx、Rxnと型で作り初期化する。  
final userName = Rx<String>('');
```

では、いよいよ認証コントローラーの作成に入ります。

## ♣ 8.2.2 AuthController の作成

コントローラーは今後もアプリ内で増えていきますので、lib/controllers フォルダを作成し、「auth\_controller.dart」ファイルを作成します。

コード内容は、

- 新規登録・メール/パスワードサインイン画面のテキストフィールドコントロールを定義
- 監視対象データ「firebaseUser」を定義し、null で初期化
- コントローラーの初期化イベントでテキストフィールドコントロールをインスタンス化
- `onReady` イベントで、`firebaseUser` の状態に更新があれば「`hundleAuthChanged`」呼び出し Worker 作成
- `onReady` イベントで、`firebaseUser` を Firebase から通知を受けるよう設定。
- `onClose` イベントでテキストフィールドコントロールを削除
- `handleAuthChanged` が呼ばれると、`firebaseUser` が「null」であれば「/login」へ

現時点では、アプリを起動しても AuthController はどこからも参照されていないので何も起こりません。

#### ▼ lib/auth\_controller.dart

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

enum TextEditingControllerStatus { init, clear, dispose }

class AuthController extends GetxController {
    static AuthController get to => Get.find();

    /// 新規作成画面、サインイン画面で使用するTextEditingController
    late TextEditingController nameController;
    late TextEditingController displayNameController;
    late TextEditingController emailController;
    late TextEditingController passwordController;
    late TextEditingController phoneNumberController;
    late TextEditingController profilePicController;
    late TextEditingController storeDescriptionController;

    final FirebaseAuth _auth = FirebaseAuth.instance;

    /// firebaseのユーザ firebase_authで定義されている。
    Rxn<User> firebaseUser = Rxn<User>();

    @override
    void onInit() {
        nameController = TextEditingController();
        displayNameController = TextEditingController();
        emailController = TextEditingController();
        passwordController = TextEditingController();
        phoneNumberController = TextEditingController();
        profilePicController = TextEditingController();
        storeDescriptionController = TextEditingController();
        super.onInit();
    }

    @override
    void onReady() {
        /// getXのworker、監視対象(引数1)に変化があった場合、引数2のコールバックを呼び出す。
        /// everは毎回。
        ever(firebaseUser, handleAuthChanged);
        firebaseUser.bindStream(_auth.authStateChanges());
        super.onReady();
    }

    @override
    void onClose() {
```

```
handleTextEditingControllers(TextEditingControllerStatus.dispose);
super.onClose();
}

/// textEditingControllerの初期化、終了処理
void handleTextEditingControllers(TextEditingControllerStatus status) {
    List<TextEditingController> textEditingControllers = [
        nameController,
        displayNameController,
        emailController,
        passwordController,
        phoneNumberController,
        profilePicController,
        storeDescriptionController,
    ];

    switch (status) {
        case TextEditingControllerStatus.clear:
            for (var controller in textEditingControllers) {
                controller.clear();
            }
            break;
        case TextEditingControllerStatus.dispose:
            for (var controller in textEditingControllers) {
                controller.dispose();
            }
            break;
        default:
            break;
    }
}

/// firebaseUserの状態が変化したときに呼ばれる
handleAuthChanged(_firebaseUser) async {
    if (_firebaseUser == null) {
        handleTextEditingControllers(TextEditingControllerStatus.clear);
        if (Get.currentRoute != '/login') {
            Get.toNamed('/login');
        }
    } else {
        if (Get.currentRoute != '/profile') {
            Get.toNamed('/profile');
        }
    }
}
```

以上で、認証コントローラーの根本ができました。次の章からユーザ新規登録ができるようにし

ます。

.....  
**ここまで**のソースコード

▼ GitHub

```
>git clone -b 03_flutterfire_setup https://github.com/risingforce9zz/tfree>
>market.git
```

---

# 第9章

## 新規登録

---

アカウントの新規作成ができるようにします。

新規登録画面から成功すればプロフィール画面へ移動します。

### 【この章の内容】

9.1	新規登録ができるようにする	59
9.2	新規登録画面の修正	60

## 9.1

## 新規登録ができるようにする

手順は、

1. 新規登録画面の修正と入力データの検証部分の作成
2. スプラッシュスクリーンで認証コントローラーを初期化
3. 新規登録メソッドの実装
4. ページ推移のための Route 作成
5. 動作確認

です。

この章からは GetX の機能にアクセスしますので、アプリケーションが GetX アプリケーションになるようになります。具体的には、「main.dart」ファイルにある「`MaterialApp`」を「`GetMaterialApp`」に変更します。

### ▼ 変更前

```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Flutter Login UI',
    theme: ThemeData(
      primaryColor: _primaryColor,
      accentColor: _accentColor,
      scaffoldBackgroundColor: Colors.grey.shade100,
      primarySwatch: Colors.grey,
    ),
    home: SplashScreen(title: 'Flutter Login UI'),
  );
}
```

### ▼ 変更後

```
@override
Widget build(BuildContext context) {
  return GetMaterialApp(
    title: 'Flutter Login UI',
    theme: ThemeData(
      primaryColor: _primaryColor,
      accentColor: _accentColor,
      scaffoldBackgroundColor: Colors.grey.shade100,
      primarySwatch: Colors.grey,
    ),
    home: SplashScreen(title: 'Flutter Login UI'),
  );
}
```

## 9.2

## 新規登録画面の修正

編集対象は、「RegistrationPage (registration\_page.dart)」です。

### ♣ 9.2.1 新規登録画面の修正

新規登録画面の修正部分は、

#### 1. ページを StatefulWidget から StatelessWidget へ変更

コントローラーがデータを保持するため、画面では不要になります。

#### 2. 各フィールドを日本語化

フィールド名、プレースホルダーを日本語化します。

#### 3. 登録ボタンからコントローラーのメソッドを実行

登録ボタンにコントローラーの新規登録メソッドを割り当てます。

#### 4. 入力データの検証

登録ボタンクリック後、コントローラーのメソッドを呼ぶ前に各フィールドの入力値を検証します。

それでは、はじめましょう。

#### 1. StatefulWidget から StatelessWidget へ

まずは、ページ全体に対し、「StatefulWidget」から「 StatelessWidget」へ変更します。

#### StatefulWidget -> StatelessWidget へ

下図のように、

- extends の後ろにある「 StatefulWidget 」を「 StatelessWidget 」へ変更。
- 赤ワク部分を削除。

```
import '../../../../../common/theme_helper.dart';
import './widgets/header_widget.dart';

class RegistrationPage extends StatelessWidget {
  @override
  State<StatefulWidget> createState() {
    return _RegistrationPageState();
  }
}

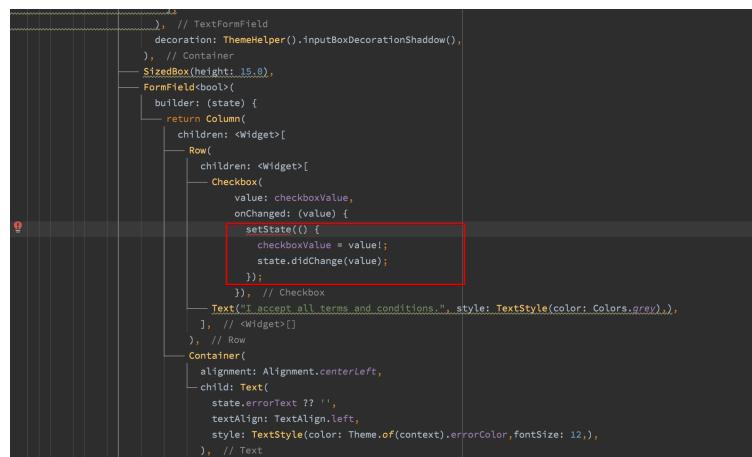
class _RegistrationPageState extends State<RegistrationPage> {

  final _formKey = GlobalKey<FormState>();
  bool checkedValue = false;
  bool checkboxValue = false;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.white,
      body: SingleChildScrollView(
        child: Stack(
          children: [
            Container(
              height: 150,
              child: HeaderWidget(150, false, Icons.person_add_alt_1_rounded),
            ), // Container
            Container(
              margin: EdgeInsets.fromLTRB(25, 50, 25, 10),
              padding: EdgeInsets.fromLTRB(10, 0, 10, 0),

```

これを行うと、`formKey`、`setState((){})` 部分がエラーになるので削除します。



Android Studio 上で lint の出すエラーがなくなれば、動作確認を行います。新規登録画面を開くことができれば問題ありません。

各フィールドとコントローラーの `TextEdittingController` の対応

アカウントの新規登録の際に入力する項目は、

- 表示名 -- displayName

- 氏名 -- name
- メールアドレス -- email
- 携帯電話番号 -- phoneNumber
- パスワード -- password

です。

認証コントローラー側の TextEditingController には、上記のフィールド名 +Controller で命名してあります。これらコントローラーと画面上の TextField を結びつけ、登録ボタンには新規登録メソッドを割り当てます。

新規登録ページ (RegistrationPage) に、認証コントローラー (AuthController) をバインドします。ファイル「auth\_controller.dart」をインポートし、インスタンスを結びつけます。

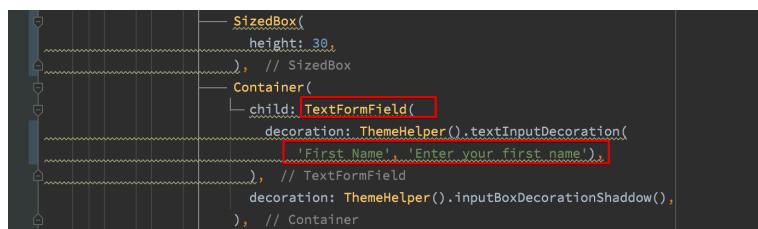
AuthController のインスタンス化は、後ほどスプラッシュスクリーンの起動で行うようにします。

#### ▼ AuthController をバインド

```
import '../controllers/auth_controller.dart';
import 'profile_page.dart';
import '../../common/theme_helper.dart';
import './widgets/header_widget.dart';

class RegistrationPage extends StatelessWidget {
    bool checkboxValue = false;
    final AuthController authController = AuthController.to;
```

テキストフィールドを「TextField」から「TextFormField」へ変更し、「controller:」プロパティへそれぞれのコントローラーをバインドします。フィールド名、ヒントを日本語にします。



▲図 9.1: 変更前



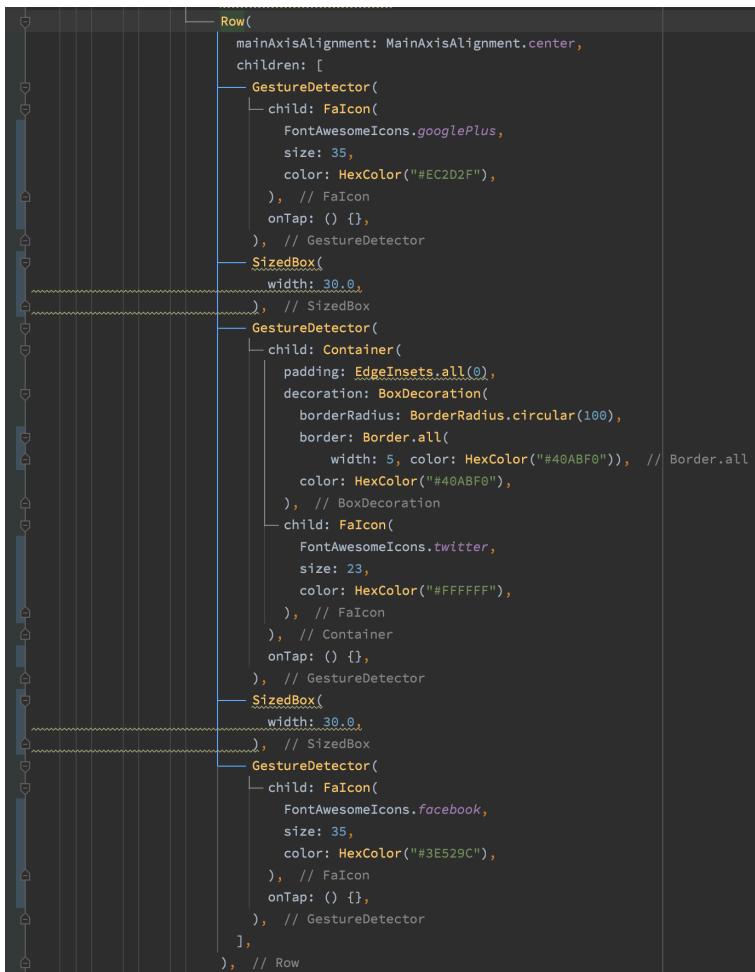
▲ 図 9.2: 変更後

残りのフィールドにも同じ作業を行います。

メールアドレス入力欄、電話番号入力欄、パスワード入力欄には、入力時のエラーチェック「validator」プロパティがありますが、入力値検証は一カ所にまとめるため削除します。



登録ボタンの後ろへソーシャルログインボタンがありますが、これも削除します。



▲図9.3: desc

続いて、新規登録メソッドが実行される前に、入力されたデータが有効なのかをチェックする関数を作成します。

### 入力値検証

登録ボタンがクリックされコントローラーの新規登録メソッド前に、入力値の検証を行います。

エラーは、SnackBar を使って表示します。

GetX は、簡単に Snackbar を出すことができます。

今後増えて行く画面のことを考慮し、SnackBar を出す便利クラスを作成します。「lib/helper」フォルダを作成し「utility.dart」ファイルを作成します。

**▼ lib/helper/utility.dart**

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

class Utility {
    static customSnackBar(String title,
        String message, [
        String? status,
    ]) {
        if (status != null) {
            late Icon icon;
            Color backgroundColor = HexColor('#FEEBFC');
            Color textColor = HexColor('#86468B');
            Color iconColor = HexColor('#FBC425');
            switch (status) {
                case 'success':
                    backgroundColor = HexColor('#28a745');
                    icon = Icon(Icons.task_alt, size: 24.0, color: textColor);
                    break;
                case 'info':
                    backgroundColor = HexColor('#007bff');
                    icon = Icon(Icons.info_outline_rounded,
                        size: 24.0, color: textColor);
                    break;
                case 'warn':
                    icon = Icon(Icons.warning_amber, size: 24.0, color: iconColor);
                    break;
                case 'error':
                    backgroundColor = HexColor('#dc3545');
                    icon = Icon(Icons.error_outline, size: 24.0, color: textColor);
                    break;
                default:
                    break;
            }
            Get.snackbar(title, message,
                backgroundColor: backgroundColor,
                colorText: textColor,
                snackPosition: SnackPosition.BOTTOM,
                icon: icon,
                borderRadius: 5.0,
                borderColor: HexColor('#6c757d'),
                borderWidth: 2
            );
        } else {
            Get.snackbar(title, message,
                backgroundColor: Colors.white, snackPosition: SnackPosition.BOTTOM);
        }
    }
}
```

**入力値検証**

新規登録ページ（registration\_page.dart）へ以下のコードを追加します。

GetX には、データの Validation 機能もあります。ここでは、メールアドレスの検証を GetX の Validation 機能で行っています。

### ▼ 入力値検証

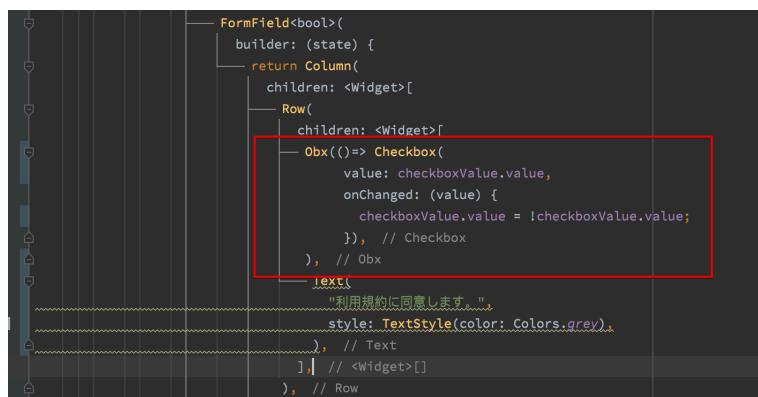
```
/// 送信ボタンクリック時に各入力フィールドをチェック
/// エラーの場合は、SnackBarを表示する。
bool _validate() {
    // 表示名
    if (authController.displayNameController.text.isEmpty) {
        Utility.customSnackBar('入力確認', '表示名を入力してください。', 'warn');
        return false;
    }
    // 氏名
    if (authController.nameController.text.isEmpty) {
        Utility.customSnackBar('入力確認', '氏名を入力してください。', 'warn');
        return false;
    }
    // 表示名
    String val = authController.emailController.text.trim();
    if (val.isEmpty || !GetUtils.isEmail(val)) {
        Utility.customSnackBar('入力確認', '有効なメールアドレスを入力してください。', 'warn');
        return false;
    }
    // 携帯電話番号
    val = authController.phoneNumberController.text;
    if (!RegExp(r"^\d{10}").hasMatch(val)) {
        Utility.customSnackBar('入力確認', 'xxx-xxxx-xxxxで入力してください。。', 'warn');
        return false;
    }
    // パスワード
    RegExp passValid = RegExp(r"^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[\W])");
    val = authController.passwordController.text.trim();
    if (!passValid.hasMatch(val) || val.length < 8) {
        Utility.customSnackBar('入力確認', '8文字で新規登録画面の修正は完了です。、半角で英数大文字・小文字・数字・記号が各1文字以上', 'warn');
        return false;
    }
    // 利用規約同意
    if (!checkboxValue.value) {
        Utility.customSnackBar("入力確認", "利用規約同意が必要です。", 'warn');
        return false;
    }
    return true;
}
```

利用規約同意のチェックボックスの値が更新されたときにチェックボックス Widget が再描画されるように、チェックボックスの値である「checkboxValue」を GetX を使って監視対象にします。

「bool checkboxValue = false;」  
↓  
「final Rx<bool> checkboxValue = false.obs;」  
に変更します。

変更しますと、checkboxValue の値は、「checkboxValue.value」でアクセスします。lint がエラーを出している部分を修正します。

チェックボックスの値が変更されたときに再描画されるように、GetX の Widget で「Checkbox」を囲みます。



## GetX の Validator

GetX の Utility クラスでは、

- isNull
- isNullOrBlank
- isBlank
- isNum
- isNumericOnly
- isAlphabetOnly
- hasCapitalletter
- isBool
- isVideo
- isImage

など、たくさんの Validation がありますが厳密に検査していないものあります。使うか使わないかはソースコード<sup>\*1</sup>を見て決めてください。

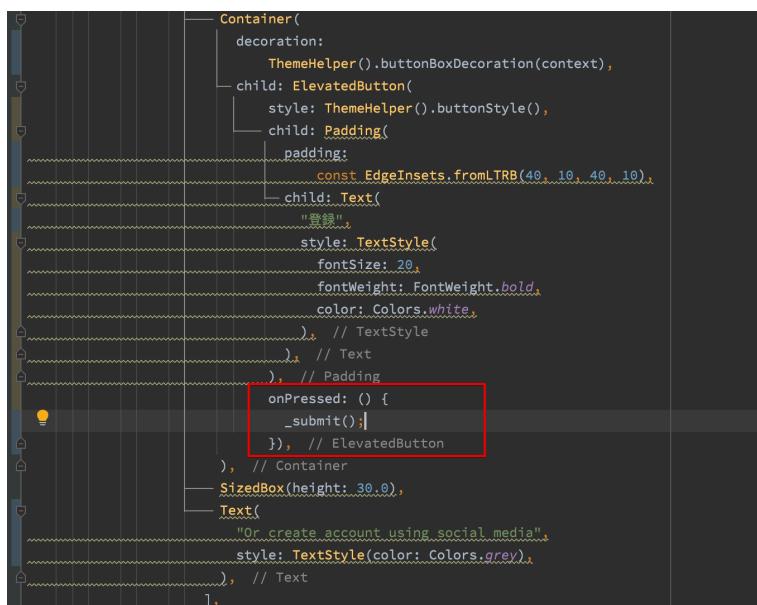
### 登録ボタン

入力値検証後、認証コントローラーの新規登録メソッドを呼ぶようにします。認証コントローラーの「signup」メソッドは未作成ですので、現時点ではエラーになります。

#### ▼認証後、新規登録メソッドを呼ぶ

```
void _submit() {
    if (_validate()) {
        authController.signup();
    }
}
```

最後に、ボタンに上記関数を呼ぶように変更します。



以上で新規登録画面の修正は完了です。

### ♣ 9.2.2 スプラッシュスクリーンで認証コントローラーを初期化

アプリケーション起動時に表示されるページである「スプラッシュスクリーン」へ認証コント

<sup>\*1</sup> [https://github.com/jonataslaw/getx/blob/master/lib/get\\_utils/src/get\\_utils.dart](https://github.com/jonataslaw/getx/blob/master/lib/get_utils/src/get_utils.dart)

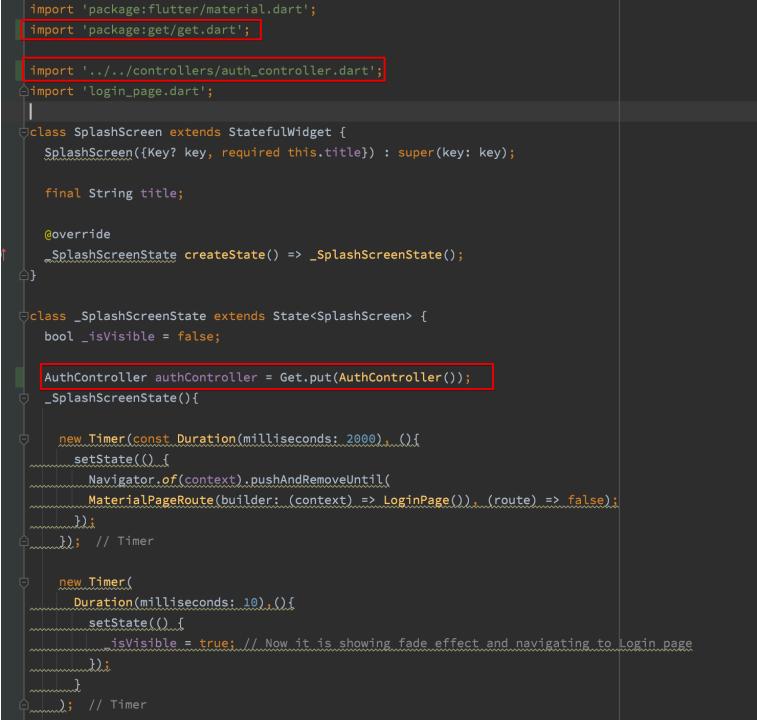
ローラー（AuthController）のインスタンス化を行うようコードを追加します。

認証コントローラーは、アプリケーションが起動した後、どのページへ推移するかを決めるため、Firebase を初期化したようにアプリケーションの最初でも構いません。

GetX のパッケージ、auth\_controller.dart のインポートを行い、

```
AuthController authController = Get.put(AuthController());
```

で、インスタンス化を行います。一度インスタンス化を行うと、他のページからは「to」に指定した「Get.find()」でインスタンスにアクセスできます。



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

import '../../../../../controllers/auth_controller.dart';
import 'login_page.dart';

class SplashScreen extends StatefulWidget {
  SplashScreen({Key? key, required this.title}) : super(key: key);

  final String title;

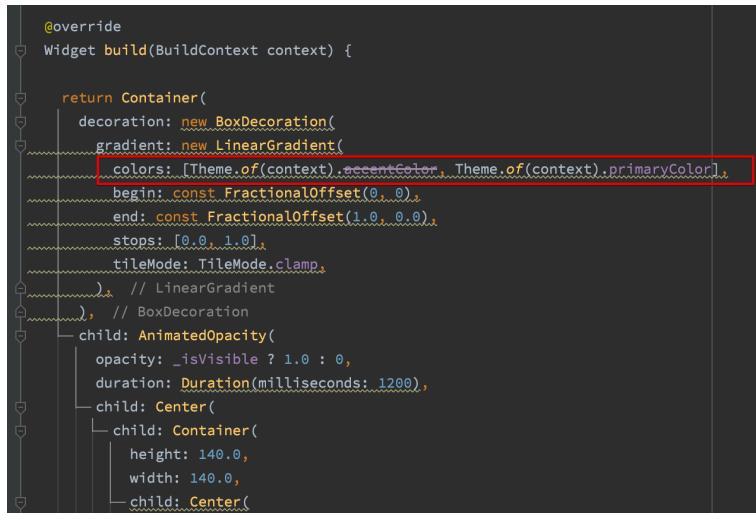
  @override
  _SplashScreenState createState() => _SplashScreenState();
}

class _SplashScreenState extends State<SplashScreen> {
  bool _isVisible = false;

  AuthController authController = Get.put(AuthController());
  _SplashScreenState(){
    new Timer(const Duration(milliseconds: 2000), (){
      setState(() {
        Navigator.of(context).pushAndRemoveUntil(
          MaterialPageRoute(builder: (context) => LoginPage()), (route) => false);
      });
    });
  } // Timer

  new Timer(
    Duration(milliseconds: 10),
    (){
      setState(() {
        _isVisible = true; // Now it is showing fade effect and navigating to Login page
      });
    }
  );
}
```

スプラッシュスクリーンでは、「accentColor」に取り消し線がついています。これは、近い将来のバージョンで消えるプロパティを意味します。



▲図9.4: accentcolorに取り消し線

公式サイト<sup>\*2</sup>に対処法があります。

「main.dart」を以下のように修正します。

### ▼修正前

```

theme: ThemeData(
  primaryColor: _primaryColor,
  accentColor: _accentColor,
  scaffoldBackgroundColor: Colors.grey.shade100,
  primarySwatch: Colors.grey,
),

```

### ▼修正後

```

theme: ThemeData(
  primaryColor: _primaryColor,
  colorScheme: ColorScheme.fromSwatch().copyWith(secondary: _accentColor),
  scaffoldBackgroundColor: Colors.grey.shade100,
  primarySwatch: Colors.grey,
),

```

スプラッシュスクリーン側は、

```

colors: [Theme.of(context).accentColor, Theme.of(context).primaryColor],
↓
colors: [Theme.of(context).colorScheme.secondary,

```

<sup>\*2</sup> <https://docs.flutter.dev/release/breaking-changes/theme-data-accent-properties>

```
Theme.of(context).primaryColor],
```

に変更します。その他のファイルにも「accentColor」が使われていますので同じく修正してください。

### ♣ 9.2.3 新規登録メソッドの実装

最後に、認証コントローラーに新規登録メソッドを作成します。

Firebase 認証でのメール/パスワードでのログインを行うためには、新規ユーザ作成メソッドとして「createUserWithEmailAndPassword()」が提供されています。

詳細は、[公式サイト](#)<sup>\*3</sup>を確認してください。

公式サイトにあるコードをそのままですが、以下のように「singup()」メソッドを実装します。

#### パスワードベースのアカウントを作成する

パスワードを使用して新しいユーザー アカウントを作成するには、  
createUserWithEmailAndPassword() メソッドを呼び出します。

```
try {
    final credential = await FirebaseAuth.instance.createUserWithEmailAndPassword(
        email: emailAddress,
        password: password,
    );
} on FirebaseAuthException catch (e) {
    if (e.code == 'weak-password') {
        print('The password provided is too weak.');
    } else if (e.code == 'email-already-in-use') {
        print('The account already exists for that email.');
    }
} catch (e) {
    print(e);
}
```

認証コントローラー (auth\_controller.dart) へ以下のコードを追加します。

#### ▼ signup メソッド

```
/// Error発生時にログイン状態を変更しSnackBarを表示
void setAuthErrorMessage(String message) {
    Utility.customSnackBar('エラー', message);
}

/// アカウント新規作成
Future<void> signUp() async {
    try {
        await _auth.createUserWithEmailAndPassword(
            email: emailController.text, password: passwordController.text);
```

<sup>\*3</sup> <https://firebase.google.com/docs/auth/flutter/password-auth?hl=ja&authuser=0>

```
    firebaseUser.value = _auth.currentUser;
} on FirebaseAuthException catch(e) {
  if (e.code == 'email-already-in-use') {
    setAuthErrorMessage('このメールアドレスはすでに使用されています。');
  });
} catch (error) {
  setAuthErrorMessage(error.toString());
}
}
```

「singup()」メソッドを実装しましたので、「RegistrationPage」のエラーも解消されています。

#### ♣ 9.2.4 ページ推移のための Route 作成

ページの移動も GetX で管理することができます。

「lib/route」フォルダを作成し「app\_route.dart」ファイルを作成します。

##### ▼ app\_route.dart

```
import 'package:get/get.dart';

import '../pages/profile_page.dart';
import '../pages/splash_screen.dart';

class AppRoutes {
  AppRoutes._();
  static final routes = [
    GetPage(name: '/', page: () => SplashScreen(title: "")),
    GetPage(name: '/profile', page: () => ProfilePage()),
  ];
}
```

ここで作成した Routes を GetMaterialApp に登録します。

##### ▼ main.dart

```
@override
Widget build(BuildContext context) {
  return GetMaterialApp(
    title: 'Flutter Login UI',
    theme: ThemeData(
      primaryColor: _primaryColor,
      colorScheme: ColorScheme.fromSwatch().copyWith(secondary: _accentColor),
      scaffoldBackgroundColor: Colors.grey.shade100,
      primarySwatch: Colors.grey,
    ),
    initialRoute: '/',
    getPages: AppRoutes.routes,
  );
}
```

スプラッシュスクリーンには、タイマーでログインページに移動するようになっていますが、GetX の Route を使うように変更します。

#### ▼ SplashPage 変更前

```
class _SplashScreenState extends State<SplashScreen> {
    bool _isVisible = false;

    AuthController authController = Get.put(AuthController());
    _SplashScreenState(){

        new Timer(const Duration(milliseconds: 2000), (){
            setState(() {
                Navigator.of(context).pushAndRemoveUntil(
                    MaterialPageRoute(builder: (context) => LoginPage()), (route) => false);
            });
        });
    }
}
```

#### ▼ 変更後

```
class _SplashScreenState extends State<SplashScreen> {
    bool _isVisible = false;

    AuthController authController = Get.put(AuthController());
    _SplashScreenState(){

        new Timer(const Duration(milliseconds: 2000), (){
            setState(() {
                Get.toNamed('/login');
            });
        });
    }
}
```

ログインページを Route へ登録します。

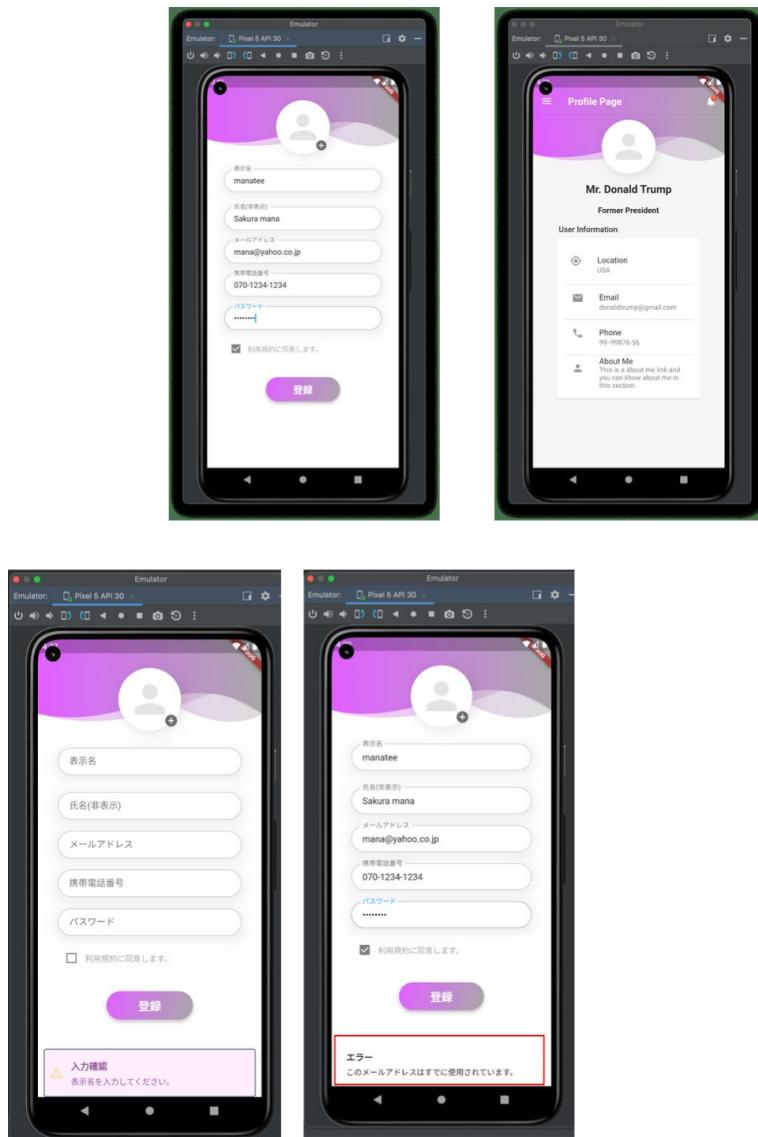
#### ▼ lib/route/app\_route.dart

```
class AppRoutes {
    AppRoutes._();
    static final routes = [
        GetPage(name: '/', page: () => SplashScreen(title: "")),
        GetPage(name: '/login', page: () => const LoginPage()),
        GetPage(name: '/profile', page: () => ProfilePage()),
    ];
}
```

### ♣ 9.2.5 動作確認

実装が完了しましたので、動作確認を行います。メールアドレスは確認されませんので、テスト目的であればメールアドレスの書式さえ合っていれば「example@example.com」でもユーザ登録できます。

以下のように、新規登録ができました。入力エラー、登録エラーも表示されています。



▲図 9.5: 動作確認

Firebase 側も登録されています。



The screenshot shows the Firebase Authentication console for a project named "tshirtsfreemarket". The "Users" tab is selected. A search bar at the top left contains placeholder text "メールアドレス、電話番号、またはユーザー ID...". To its right is a blue button labeled "ユーザーを追加". On the far right are icons for document navigation and help. Below the search bar is a table with four columns: "ID", "プロバイダー", "作成日", and "ログイン日". The table has one row of data, showing a user with the email "mana@yahoo...". The "provider" column shows an envelope icon, "creation date" shows "202...", and "last sign-in date" shows "202...". The "user ID" column shows a long string of characters: "n1YDXX4EBTS24p...". At the bottom of the table, there are pagination controls: "ページあたりの行数:" followed by a dropdown menu set to "50", and "1 - 1 of 1".

▲図 9.6: Firebase 認証

# 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