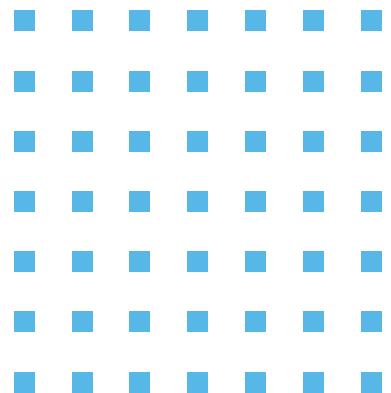
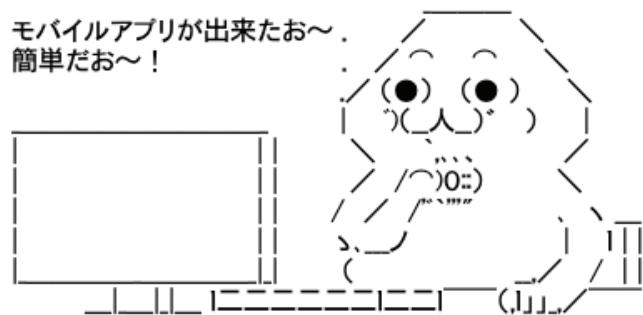


-0円で作る モバイルアプリ)-Vol.1

Firebase認証編



モバイルアプリが出来たお～.
簡単だお～！



Flutter



Built with
Firebase

 **GetX** State Manager | Navigation Manager
Dependencies Manager
Fast, Stable, Extra-light and Powerful Flutter Framework



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。

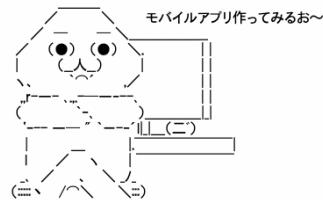
■商標

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」を使わせていただけます。tkg 様始めたたくさんの 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}](#)がお勧めです。10分ほどの動画ですが「Flutter」の概要が分かります。



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

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、おもしろいアプリが世の中に溢れるといいですね。

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。
2. (次号 Vol.2) ストア情報の管理、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 でのコードだと何とか理解できます。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。

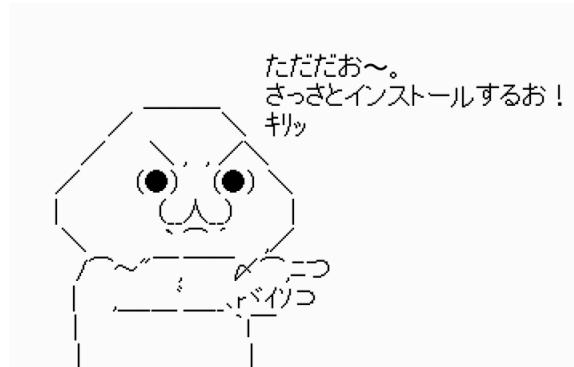
本書のソースコードは GitHub にあります。章毎にブランチを分けていますので、どの章からでも始めていただけます。できる限り図解していますのでページ数も膨大になりますが、ご容赦ください。

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？	15
2.1 Firebase を使ってみる	16
2.1.1 Firebase コンソールへ	19
2.1.2 Firebase 認証とは？	25
第3章 アプリケーション認証	26
3.1 認証フロー	27
3.1.1 スプラッシュスクリーンからサインイン選択	28
3.1.2 新規登録	29
3.1.3 メール/パスワード	30
3.1.4 SNS 認証	31
3.1.5 プロフィール画面	32
第4章 プロジェクトスタート	33
4.1 新規プロジェクトの作成	34
第5章 認証関連画面の作成	38
5.1 UIについて	39
5.2 ログインプログラム	40
5.2.1 元プロジェクトの確認	41
5.2.2 プロジェクトへ適用	42
第6章 アプリで認証するユーザーのデータ型を作成する。	45
6.1 ユーザーモデルの作成	46
第7章 Flutter で Firebase を使う	48
7.1 クライアント側（Flutter）で Firebase の設定	49

7.2	Firebase サービス用 Flutter プラグインのインストール	54
7.3	テスト	56
第 8 章	状態管理コントローラを作る	57
8.1	GetX コントローラ	58
8.2	認証コントローラの作成	60
8.2.1	GetXController について	61
8.2.2	AuthController の作成	62
第 9 章	新規アカウント登録	65
9.1	新規登録ができるようにする	66
9.2	新規登録画面の修正	67
9.2.1	新規登録画面の修正	67
9.2.2	スプラッシュスクリーンで認証コントローラを初期化	78
9.2.3	新規登録メソッドの実装	81
9.2.4	ページ推移のための Route 作成	82
9.2.5	動作確認	85
第 10 章	メール/パスワードでのサインイン	87
10.1	メール/パスワードでサインインできるようになるまで	88
10.2	サインイン選択画面を作成	89
10.3	LoginPage の修正	94
10.3.1	StatelessWidget への変更	94
10.3.2	入力フィールドと認証コントローラの結び付け	95
10.3.3	パスワード・リセットページへの推移	97
10.4	ForgotPasswordPage の修正	98
10.4.1	StatelessWidget へ変更	98
10.4.2	認証コントローラのバンドル	99
10.4.3	LoginPage へ戻るナビゲーション	101
10.4.4	送信ボタンへパスワードリセットメソッドをバインド	101
10.5	サインアウトの実装	103
10.6	パスワード・リセット要求完了ページ	104
10.7	認証コントローラにメソッドの実装	107
10.7.1	メール/パスワードサインイン・メソッド	107
10.7.2	サインアウト・メソッド	107
10.7.3	パスワードリセット・メソッド	108
10.8	動作確認	109
10.8.1	Route	109
10.8.2	メール/パスワードでのサインイン	110
10.8.3	サインアウト	112

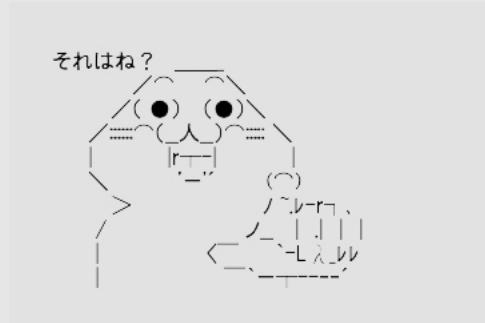
10.8.4 パスワード・リセット	113
第 11 章 Twitter アカウントでサインイン	115
11.1 Twitter アカウントで Firebase 認証	116
11.2 Twitter Developer サイトでキー作成	117
11.3 Firebase 認証で Twitter API キーの登録	125
11.4 アプリケーションへメソッド追加	128
11.4.1 flutter_dotenv のインストール	128
11.5 認証コントローラへメソッドの追加	130
11.5.1 twitter_login プラグインのインストール・設定	130
11.5.2 Twitter サインイン・メソッドの追加	134
11.6 動作確認	136
11.6.1 iOS	136
11.6.2 Android	136
11.6.3 Firebase コンソール	137
第 12 章 Google サインイン	138
12.1 Google アカウントで Firebase 認証	139
12.2 Firebase にキーを登録	140
12.2.1 表示された SHA1 フィンガープリントを Firebase に登録	140
12.3 google_sign_in プラグインのインストール	145
12.4 iOS 用の設定	146
12.5 アプリに google サインイン・メソッドの実装	147
12.6 動作確認	148
12.6.1 iPhone	148
12.6.2 Android	151
第 13 章 AppleID でサインイン	152
13.1 Firebase コンソールでログインプロバイダを追加	153
13.2 Apple Developer で「Apple サインイン」を有効に	155
13.2.1 アプリケーションの ID を取得します。	155
13.2.2 Xcode で Apple でサインインを有効にする	158
13.2.3 Apple の匿名 E-mail サービスの設定	159
13.2.4 Android 用のサービス ID、キー ID を取得する	163
13.2.5 Firebase へキーを登録	171
13.3 アプリケーションに Apple サインイン・メソッドの実装	173
13.4 動作確認	175
第 14 章 次号予告	178

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」とはいったい何を指していて、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、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。

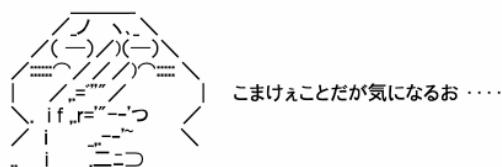
たとえば、スマートフォンで表示しているボタンの色や形状などです。



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、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。

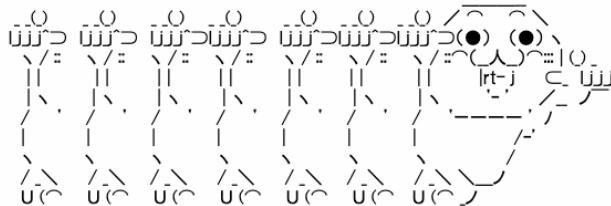


上記の例ですと、ボタンの形を変えるために A モジュールへアクセスし、色を変えるため B モジュールへアクセスするような実装ではなくボタンの状態を管理するモジュールがあれば、そこへアクセスするだけで良いとする実装を指します。

1.2

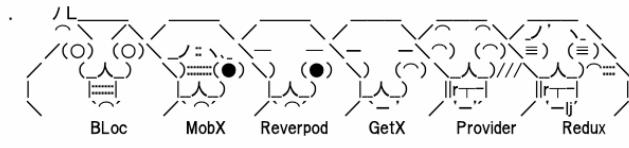
状態管理ライブラリ

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付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いえない状況にあります。



状態管理を紹介しているだけで、表情とは無関係です。です。

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、「状態管理といえば、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>

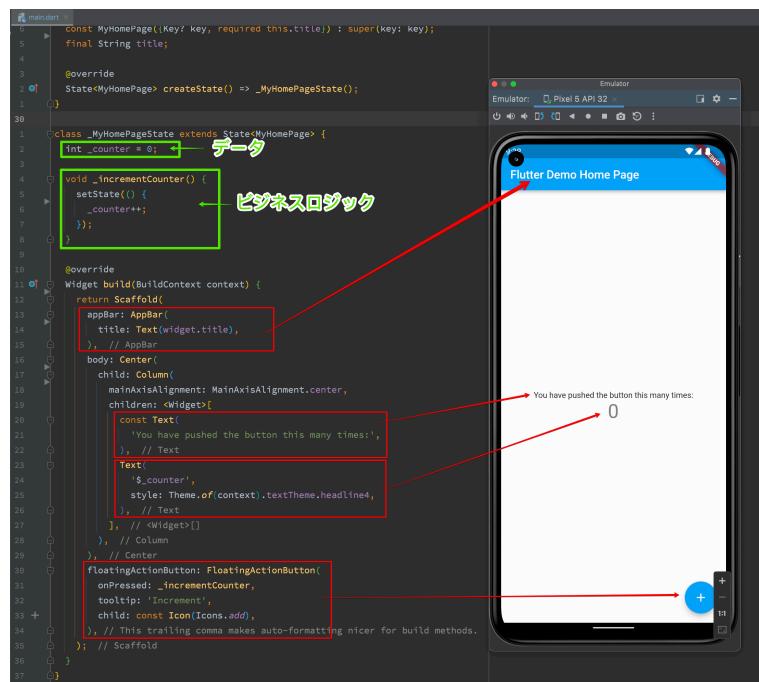
1.3

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。



お馴染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、

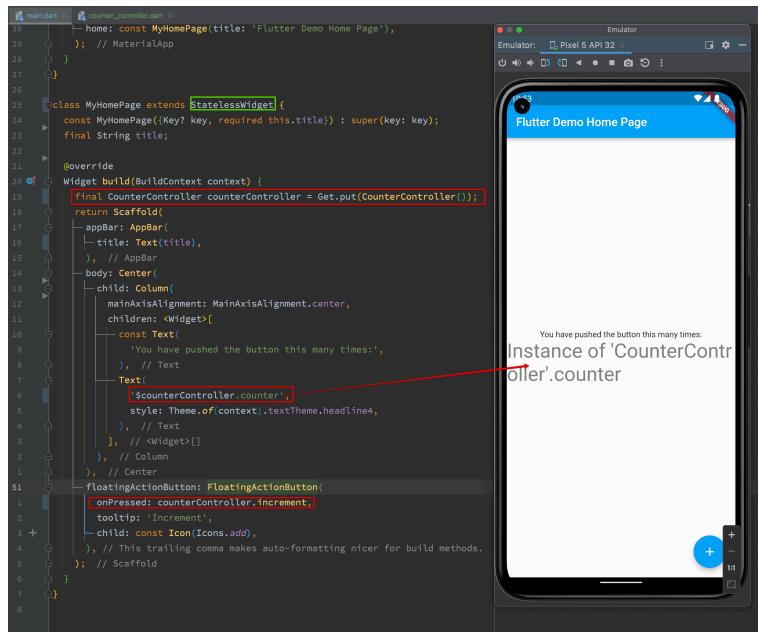
- 状態を保持する `State` を継承した状態管理 `setState` を使用
- データ保持用変数「`_counter`」を持っている
- データを操作する「`_incrementCounter`」ビジネスロジックを持っている。
- データを表示する「`Text Widget`」を持っている。

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

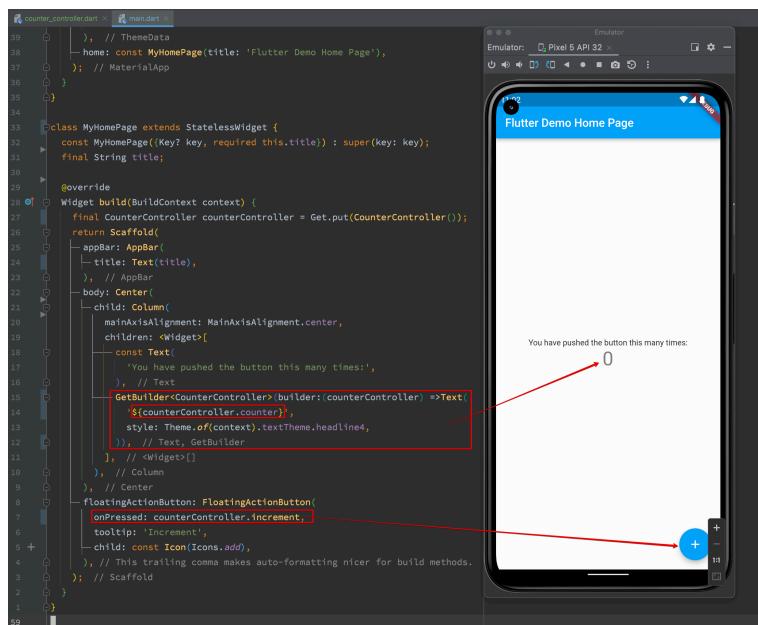
GetX（あくまで本書では）を導入し状態管理を。

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 `Get.put(CounterController())` to inject the controller. 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` with an increment icon 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 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 text on the 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。



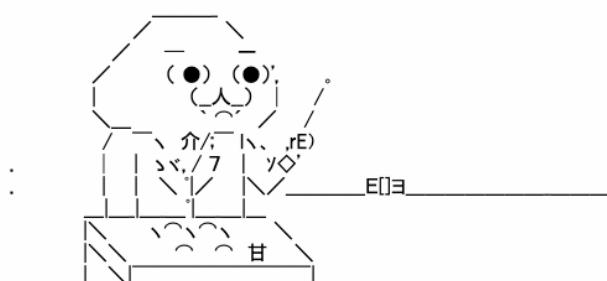
This screenshot shows the same setup as the previous one, but with modifications to the code. In `main.dart`, the `Text` widget displaying the counter value now includes a `GetBuilder` widget, which takes a builder function that receives the `counterController` as a parameter. This allows the UI to rebuild whenever the `counter` value changes. The `counter_controller.dart` file remains the same as in the previous screenshot. The emulator shows the same application running on a Pixel 5 device, but now the text "You have pushed the button this many times: 0" is displayed correctly without any errors. A red arrow points from the `GetBuilder` code in the `main.dart` code to the text on the 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

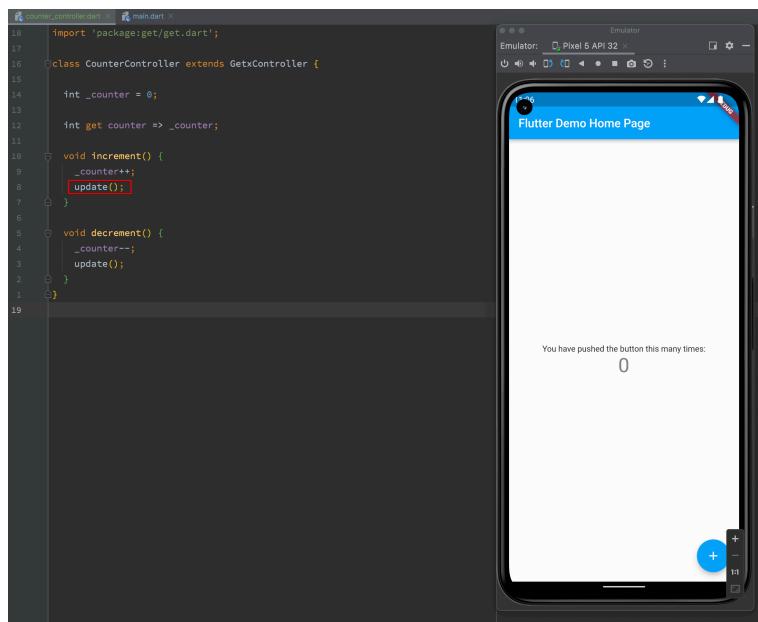
```
class GetBuilder<T extends GetxController> extends StatefulWidget {
final GetControllerBuilder<T> builder;
final bool global;
final Object? id;
final String? tag;
final bool autoRemove;
final bool assignId;
final Object Function(T value)? filter;
final void Function(GetBuilderState<T> state)? initState,
    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

状態管理をしているコントローラは、こんな風になっています。



コントローラは、「GetX コントローラ」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。お願ひっ！」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？

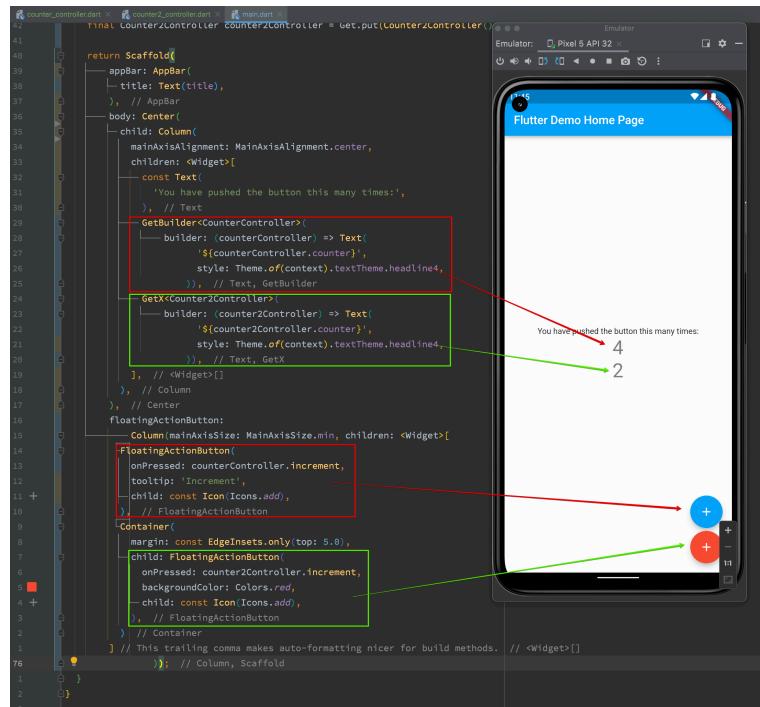
では、状態管理コントローラを導入したメリットは何でしょうか？

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

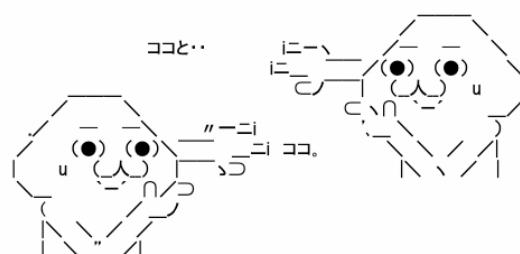
表示部を2つ、それぞれを操作するボタンを2つにしましたが、状態管理コントローラをひとつ増やしただけです。こちらには、「GetX<コントローラ型>」を使って表示しています。



```

42 final CounterController counterController = Get.put(CounterController());
43
44 return Scaffold(
45   appBar: AppBar(
46     title: Text(title),
47   ), // AppBar
48   body: Center(
49     child: Column(
50       mainAxisAlignment: MainAxisAlignment.center,
51       children: <Widget>[
52         const Text(
53           'You have pushed the button this many times:',
54         ), // Text
55         GetBuilder<CounterController>(
56           builder: (counterController) => Text(
57             '$(counterController.counter)',
58             style: Theme.of(context).textTheme.headline4,
59           ), // Text, GetBuilder
60         ),
61         GetX<Counter2Controller>(
62           builder: (counter2Controller) => Text(
63             '$(counter2Controller.counter)',
64             style: Theme.of(context).textTheme.headline4,
65           ), // Text, GetX
66         ), // <Widget>[]
67       ], // Column
68     ), // Center
69     floatingActionButton:
70       Column(mainAxisSize: MainAxisSize.min, children: <Widget>[
71         FloatingActionButton(
72           onPressed: counterController.increment,
73           tooltip: 'Increment',
74           child: const Icon(Icons.add),
75         ), // FloatingActionButton
76       Container(
77         margin: const EdgeInsets.only(top: 5.0),
78         child: FloatingActionButton(
79           onPressed: counter2Controller.increment,
80           backgroundColor: Colors.red,
81           child: const Icon(Icons.add),
82         ), // FloatingActionButton
83       ), // Container
84     ], // This trailing comma makes auto-formatting nicer for build methods. // <Widget>[]
85   ); // Column, Scaffold
86 }

```



状態管理コントローラの別インスタンス

同じデータ型、同じビジネスロジックをもつのなら、状態管理コントローラ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。

The screenshot shows the Android Studio interface. On the left is the code editor with three files: counter_controller.dart, counter2_controller.dart, and main.dart. The main.dart file contains the following code:

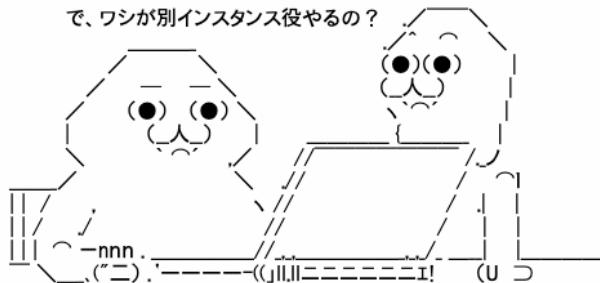
```

51:   Widget build(BuildContext context) {
52:     final CounterController counterController = Get.put(CounterController());
53:     final Counter2Controller counter2Controller = Get.put(Counter2Controller());
54:     final Counter2Controller counter3Controller =
55:       Get.put(Counter2Controller(), tag: 'counter3');
56:
57:     return Scaffold(
58:       appBar: AppBar(
59:         title: Text(title),
60:       ), // AppBar
61:       body: Center(
62:         child: Column(
63:           mainAxisAlignment: MainAxisAlignment.center,
64:           children: <Widget>[
65:             const Text(
66:               'You have pushed the button this many times:',
67:             ), // Text
68:             GetBuilder<CounterController>(
69:               builder: (counterController) => Text(
70:                 '$counterController.counter',
71:                 style: Theme.of(context).textTheme.headline4,
72:               ), // Text, GetBuilder
73:             GetX<Counter2Controller>(
74:               builder: (counterController) => Text(
75:                 '$counter2Controller.counter',
76:                 style: Theme.of(context).textTheme.headline4,
77:               ), // Text, GetX
78:               Obx(() => Text(
79:                 '${counter3Controller.counter}',
80:                 style: Theme.of(context).textTheme.headline4,
81:               )), // Text, Obx
82:           ], // <Widget>[]
83:         ), // Column
84:         floatingActionButton:
85:           Column(mainAxisSize: MainAxisSize.min, children: <Widget>[
86:             FloatingActionButton(
87:               onPressed: counterController.increment,
88:               tooltip: 'Increment',
89:               child: const Icon(Icons.add),
90:             ), // FloatingActionButton
91:             Container(
92:               margin: const EdgeInsets.symmetric(vertical: 5),
93:               child: FloatingActionButton(
94:                 onPressed: counter2Controller.increment,
95:                 backgroundColor: Colors.red,
96:                 child: const Icon(Icons.add),
97:               ), // FloatingActionButton
98:             ), // Container
99:             FloatingActionButton(
100:               onPressed: counter3Controller.increment,
101:               backgroundColor: Colors.green,
102:               child: const Icon(Icons.add),
103:             ), // FloatingActionButton
104:           ],
105:         ), // Column
106:       ),
107:     );
108:   }

```

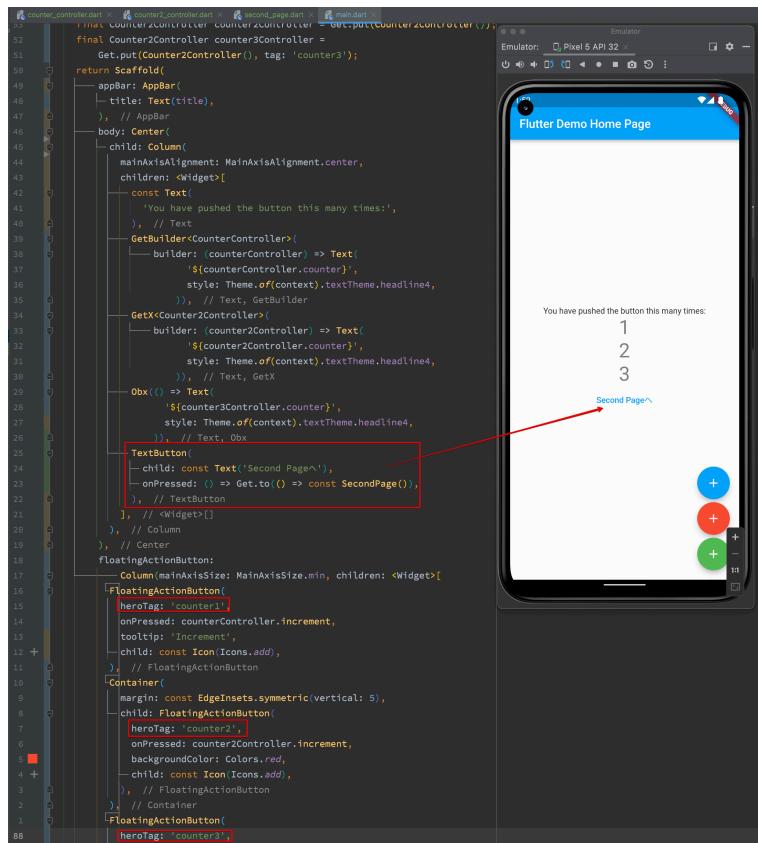
The right side shows an emulator running the app. The text "You have pushed the button this many times:" is followed by three numbers: 4, 2, and 3. A red box highlights the first number (4), a green box highlights the second (2), and a pink box highlights the third (3). A red arrow points from the first number to the first FloatingActionButton in the code. A green arrow points from the second number to the second FloatingActionButton. A pink arrow points from the third number to the third FloatingActionButton. The text "同じクラスの別インスタンス" (Different instances of the same class) is written in pink at the bottom right.

で、ワシが別インスタンス役やるの？



別ページでも同じ状態管理コントローラへアクセス

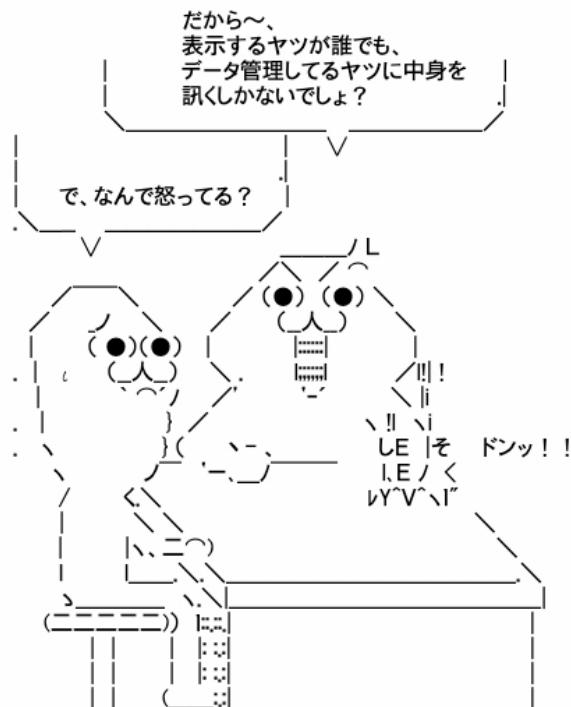
ページ移動しても、アプリケーション全体で状態管理コントローラは動作していますので、状態管理コントローラへアクセスすれば、先ほどまでのデータを取得できます。



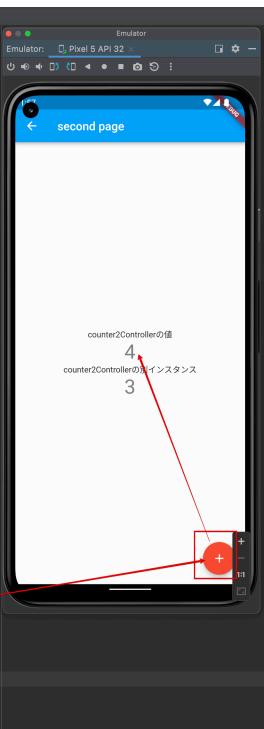
```

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 CounterController 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           ],
80         ), // Column, Center
81       ),
82       floatingActionButton: FloatingActionButton(
83         onPressed: counter2Controller.increment,
84         backgroundColor: Colors.red,
85         child: const Icon(Icons.add),
86       ), // FloatingActionButton
87     ); // Scaffold
88   }
89 }

```



ここでデータを操作し変化させます。

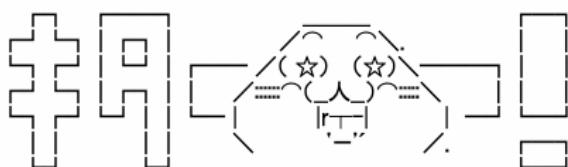


```

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("counter2Controller의 값"),
68             GetX<Counter2Controller>(
69               builder: (counter2Controller) => Text(
70                 '$(counter2Controller.counter)',
71                 style: Theme.of(context).textTheme.headline4,
72               ), // Text, GetX
73             ),
74             const Text("counter2Controller의別インスタンス"),
75             Obx(() => Text(
76               '$(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 }

```

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のデータを取得しますので、データは最新です。



The screenshot shows the Android Studio interface with the Flutter code for Counter 3. The code uses three CounterController instances to manage state for three separate counters. The UI consists of three floating action buttons (FABs) at the bottom, each with an increment icon and a hero tag. Tapping one of these FABs increments the corresponding counter. The main screen displays the current value of each counter. A 'Second Page' button is also present.

```
final Counter2Controller counter2Controller = Get.put(Counter2Controller(), tag: 'counter2');
final Counter2Controller counter3Controller = Get.put(Counter2Controller(), tag: 'counter3');

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(
            '${counterController.counter}',
            style: Theme.of(context).textTheme.headline4,
          ), // Text, GetBuilder
        ),
        GetBuilder<CounterController>(
          builder: (counterController) => Text(
            '${counterController.counter}',
            style: Theme.of(context).textTheme.headline4,
          ), // Text, GetBuilder
        ),
        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',
        |
```

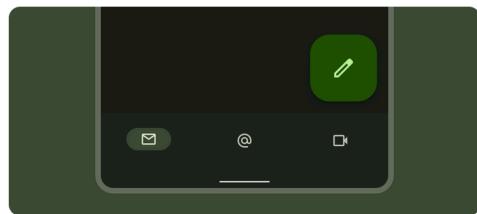


FABはひとつ。

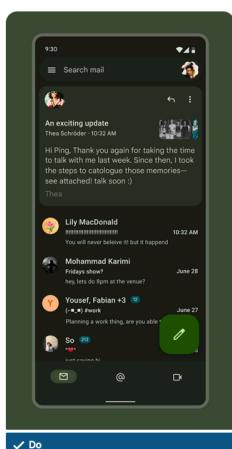
Flutterで推奨されているMaterial Designでは、FABが複数あることを推奨していません。今回は、状態管理の解説のため簡易的にです。

FAB

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



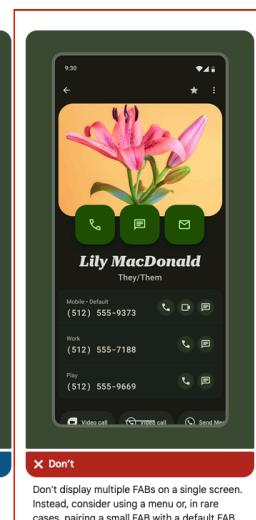
A FAB should present a screen's primary action.



✓ 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、ユーザー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みて、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 択ですので、結局 Google に買収されてしまう。

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

オタクの会社、買い取ります！



【この章の内容】

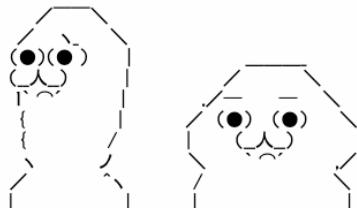
2.1 Firebase を使ってみる	16
-----------------------------------	----

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。

- ・アプリケーションへのログイン Authentication
- ・NoSQLドキュメントデータベース Cloud Firestore
- ・リアルタイム・データベース Realtime Database
- ・携帯へのメッセージ Cloud Messaging
- ・クラウド・ストレージ Cloud Storage

は、使いそうだな～。



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。

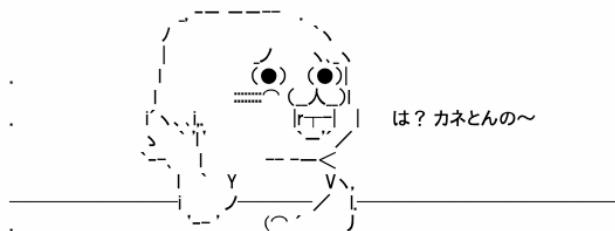
A screenshot of the Firebase homepage. At the top, there is a navigation bar with the Firebase logo, a 'Products' dropdown, a 'More' dropdown, a search bar, and language selection for 'Japanese'. A red box highlights the 'Console' button. Below the navigation, there is a large blue banner with the text 'Make your app the best it can be'. To the right of the text is a decorative graphic of a heart surrounded by colorful geometric shapes like triangles and circles. Below the banner, there is a paragraph of text: 'Firebase is an app development platform that helps you build and grow apps and games users love. Backed by Google and trusted by millions of businesses around the world.' At the bottom of the banner, there are three buttons: 'Try it now' (highlighted in red), 'Try a demo', and 'Watch a video'.

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ーが月間 5 万人まで無料です。

The screenshot shows the Firebase Pricing page for the Spark plan. The main title is "Spark プラン". Below it, a note says "初めての方向けに柔軟な上限を設定". A large blue button labeled "無料" (Free) is prominently displayed. A red box highlights the "Authentication" section. The table below lists the included services and their monthly limits:

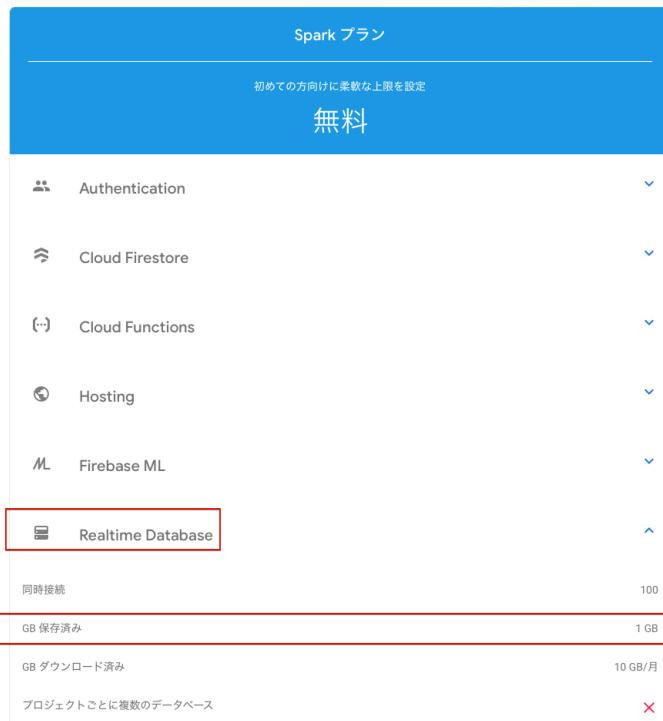
サービス	月間料金
電話認証 - 米国、カナダ、インド	1万/月
電話認証 - 他のすべての国	1万/月
他の認証サービス	✓
With Identity Platform	

Below the table, another red box highlights the "Monthly active users" row, which is listed as "50k/month".

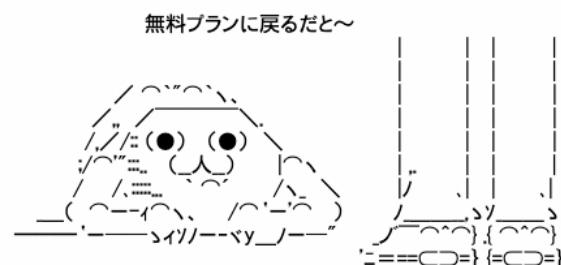


Realtime Databaseでは、保存するデータが1GBまで無料です。ただし、ダウンロードされるデータは10GBまでです。

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。

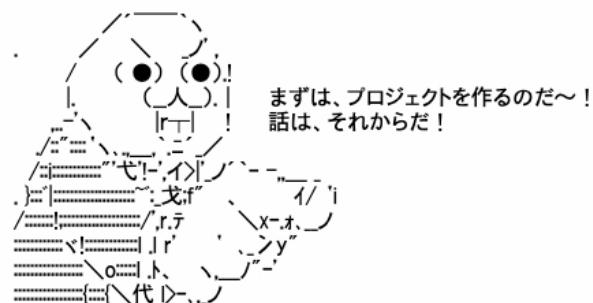
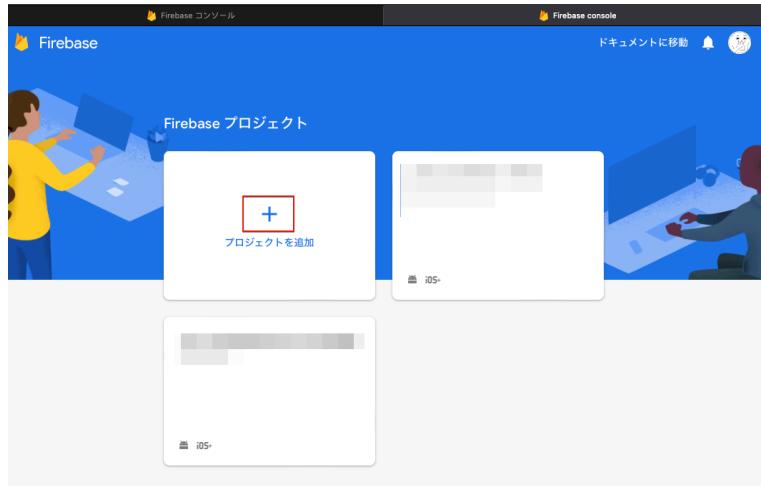


無料から有料へは、いつでも切り替えることができますし、使用量が減れば無料プランへ戻ることもできます。



♣ 2.1.1 Firebase コンソールへ

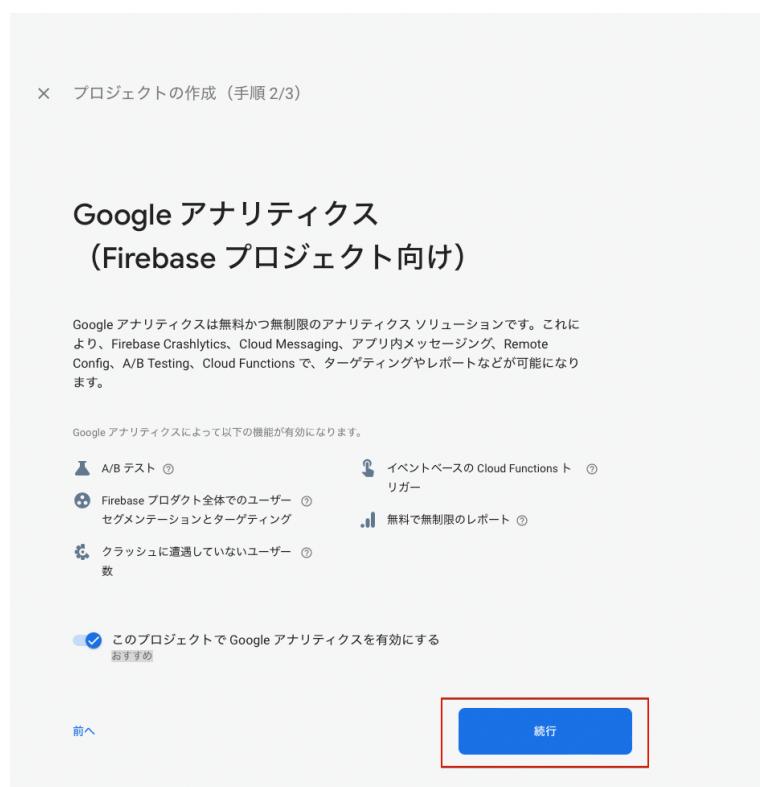
「コンソールへ移動」をクリックすると、Firebase コンソールトップへ移動します。ここで「+ プロジェクトを追加」をクリックして新規プロジェクトを作成します。



最初にプロジェクトの名前を付けます。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。



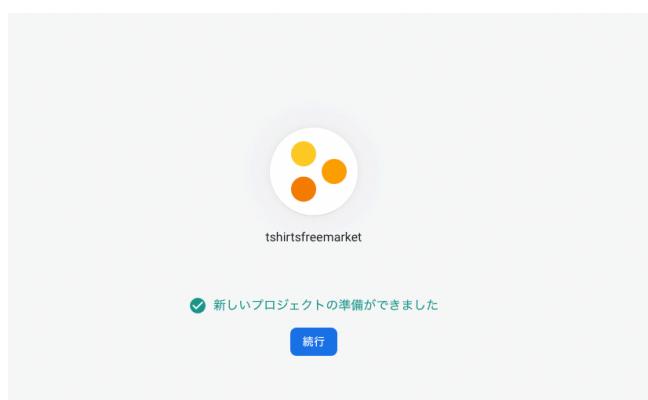
アナリティクスのアカウントを選択するか、作成します。[プロジェクトを作成] ボタンをクリックすると作成開始します。



作成中です。



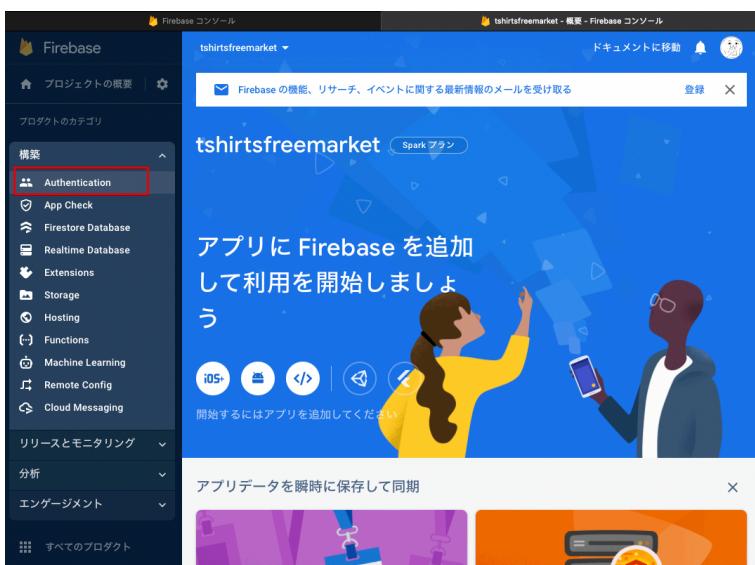
プロジェクトができあがりました。



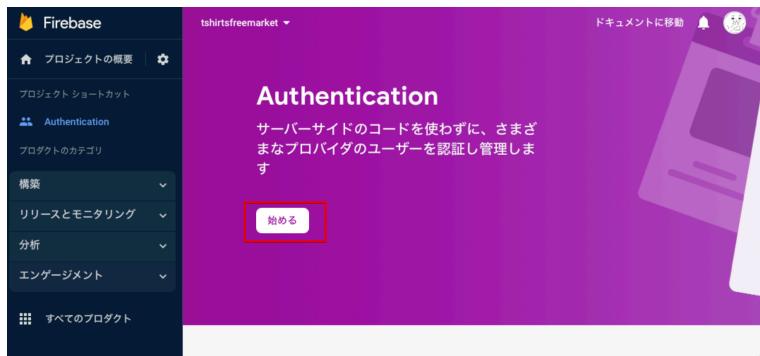
サイドバーにある [構築] ドロップダウンを開くと、追加できるサービスが表示されます。



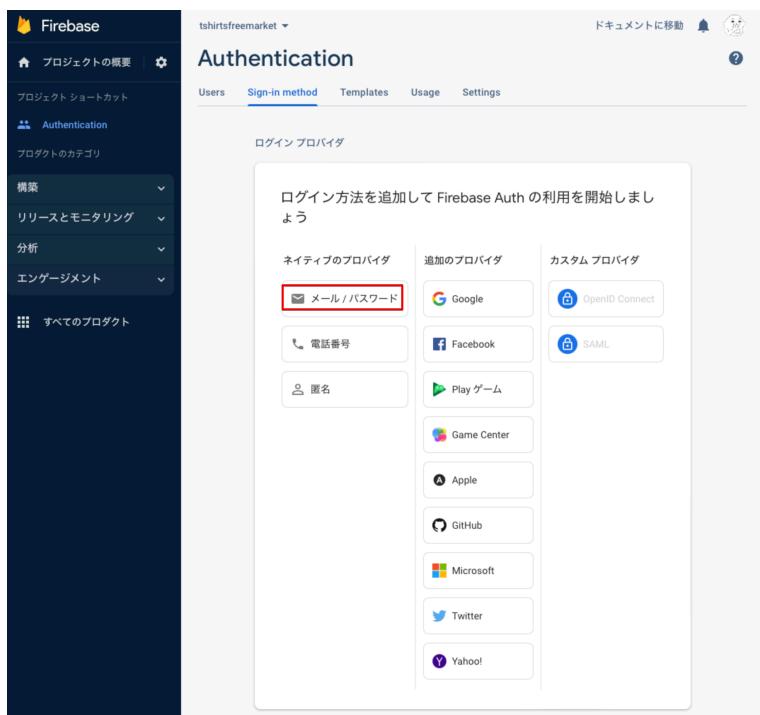
最初に、「Authentication」をクリックし認証サービスを追加します。



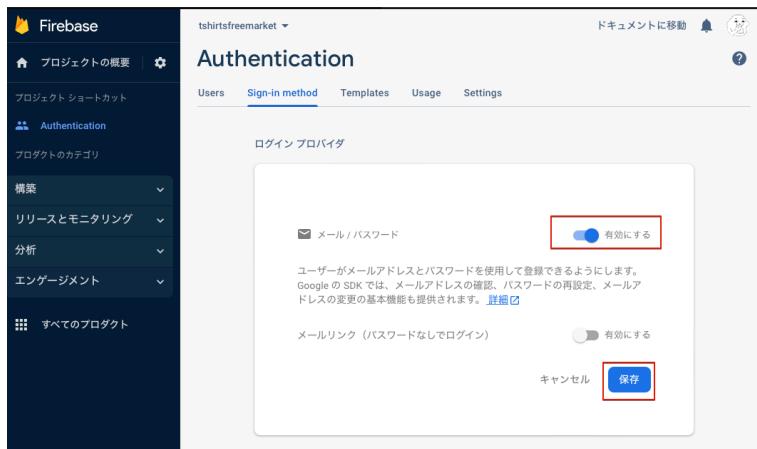
認証サービスのトップ画面に移動します。[始める] ボタンをクリックします。



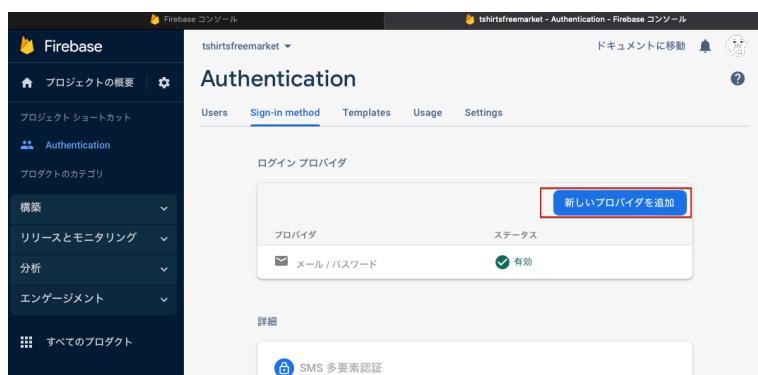
ログイン方法（ログインプロバイダ）の選択画面になります。最初は、「メール/パスワード」を選択します。



「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。



「新しいプロバイダ追加」ボタンをクリックします。



以上で「メール/パスワード」での認証が可能になりました。

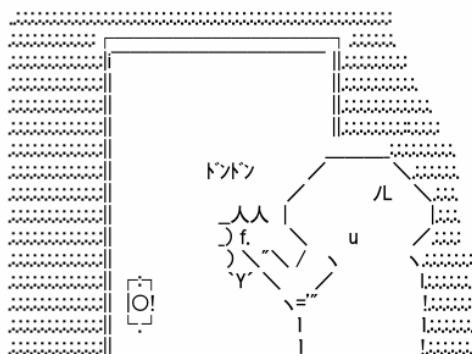
♣ 2.1.2 Firebase 認証とは？

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

ができます。

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。

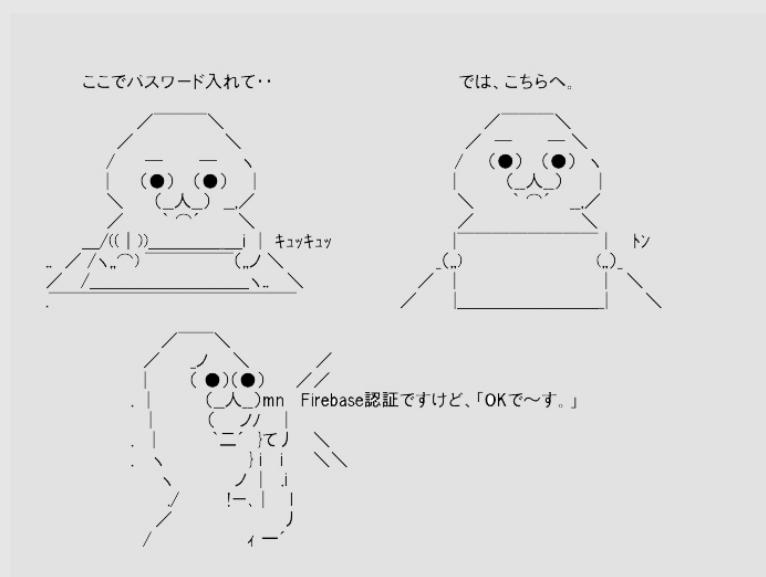
次の章からは、オンライン・フリーマーケットの認証部分を作成していきます。

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。



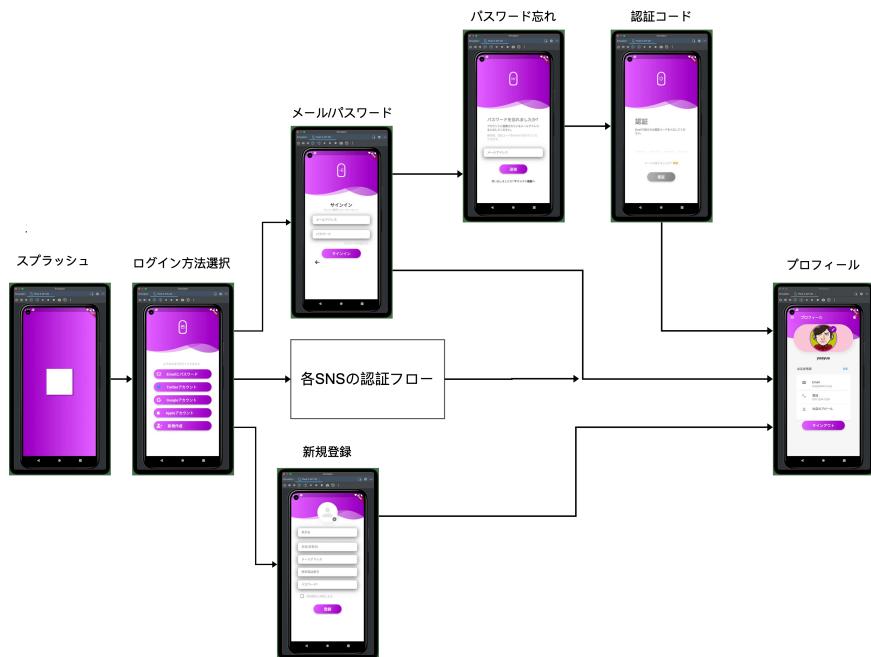
【この章の内容】

3.1 認証フロー	27
---------------------	----

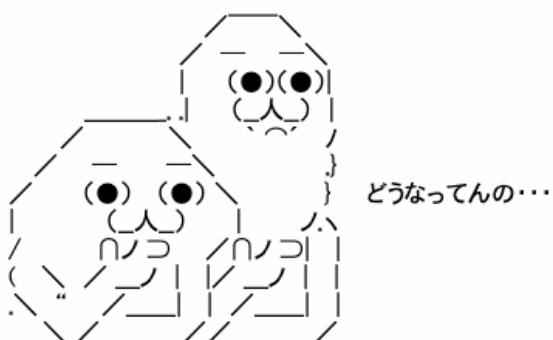
3.1

認証フロー

下図のように画面移動します。



では、順を追って画面の流れを追いかけます。



♣ 3.1.1 スプラッシュスクリーンからサインイン選択

アプリを起動すると、スプラッシュスクリーンを表示します。表示するとともに、コントローラなどの初期化を行います。

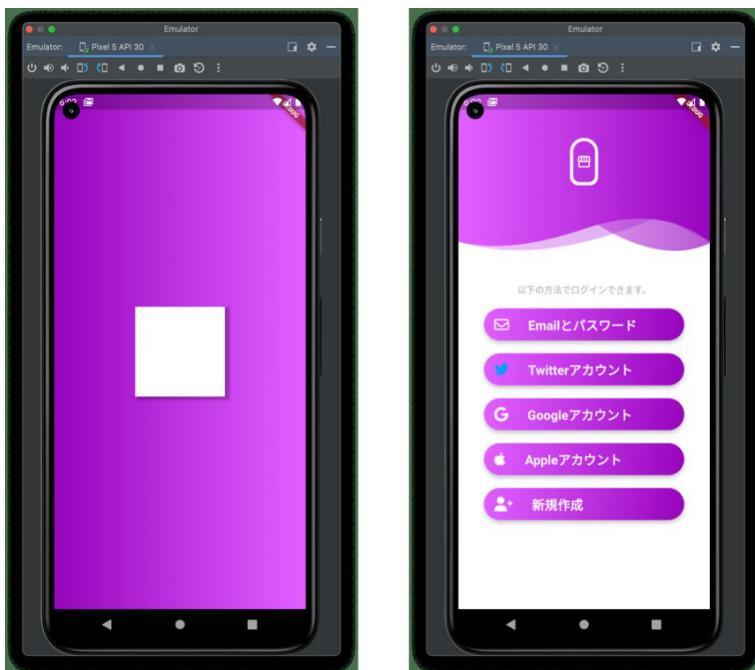
スプラッシュスクリーンでサインイン状態を判断し、サインしていない場合にサインイン方法選択画面を表示します。

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、新規登録をする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。

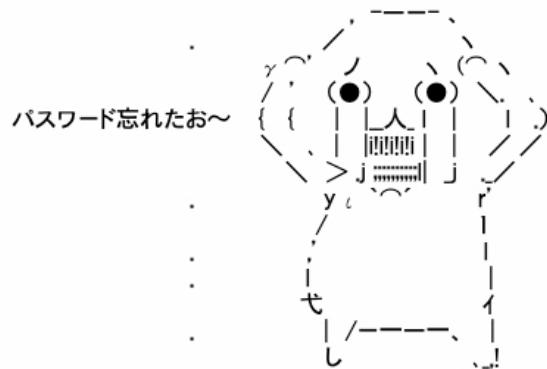
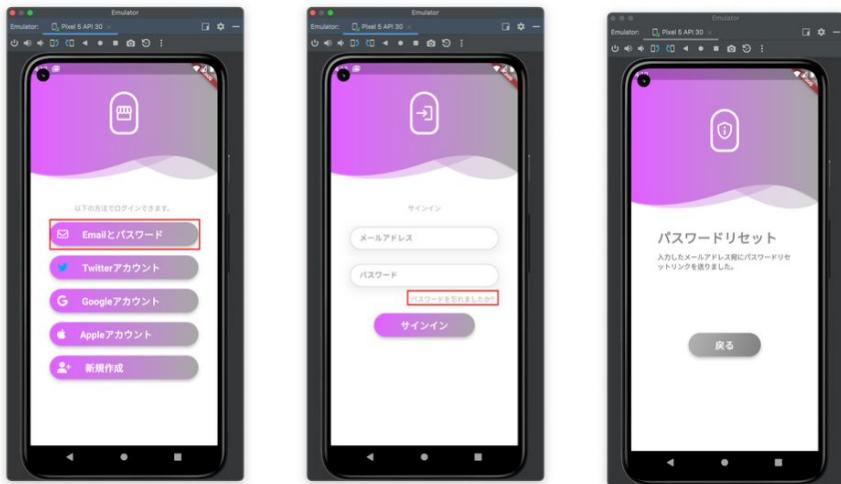
ここで入力されたものは、ストア情報（プロフィール）として使用します。アバター画像は、初回はランダムに選択しています。ストア情報管理が完成すれば、プロフィール画面で編集できます。



		履歴書 平成19年05月19日	
		<p>氏名: やる夫 生年月日 照和52年 2月10日 現住所: 東京都新宿区 連絡先: 050-xxx-xxx (IP電話)</p>	
年	月	学歴・職歴	
		学歴	
平成3	03	○○市立○○中学校卒業	
		職歴	
平成3	04	自宅警備員	
		以上	

♣ 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、登録メールアドレスへパスワードリセット用のリンクが送られて来ます。



♣ 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。

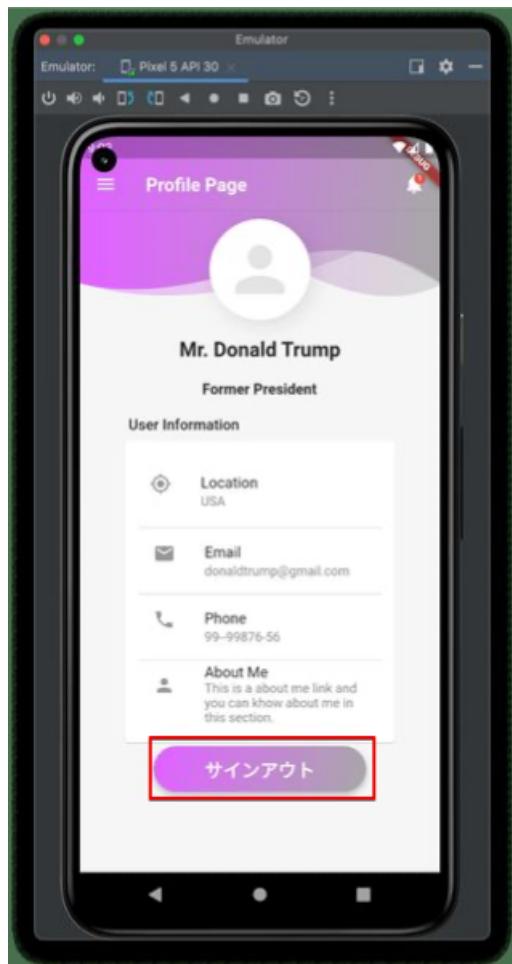
どちらも Developer アカウントでログインし、キーの取得などが必要ですがすべて図解しています。



♣ 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。

ストア管理モジュールが完成するまでは、ハードコードしてある情報が表示されます。

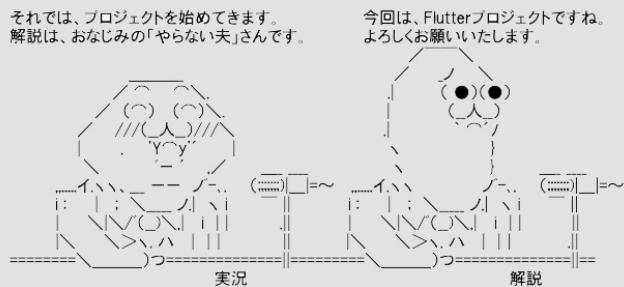


それでは、実装していきましょう。

第4章

プロジェクトスタート

Android 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。



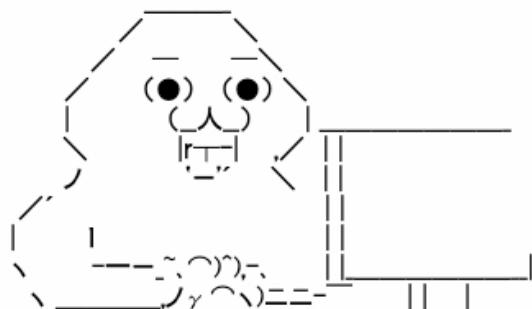
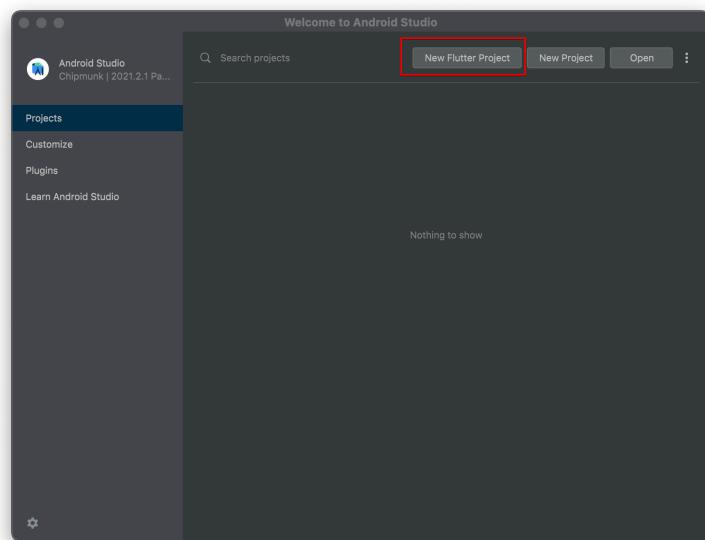
【この章の内容】

4.1 新規プロジェクトの作成	34
---------------------------	----

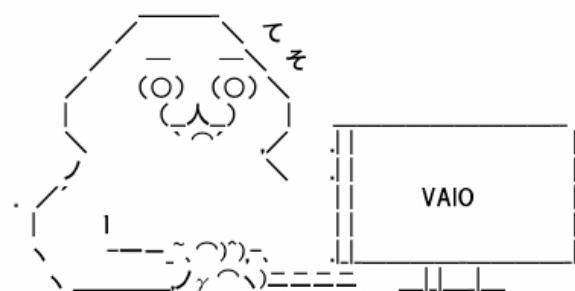
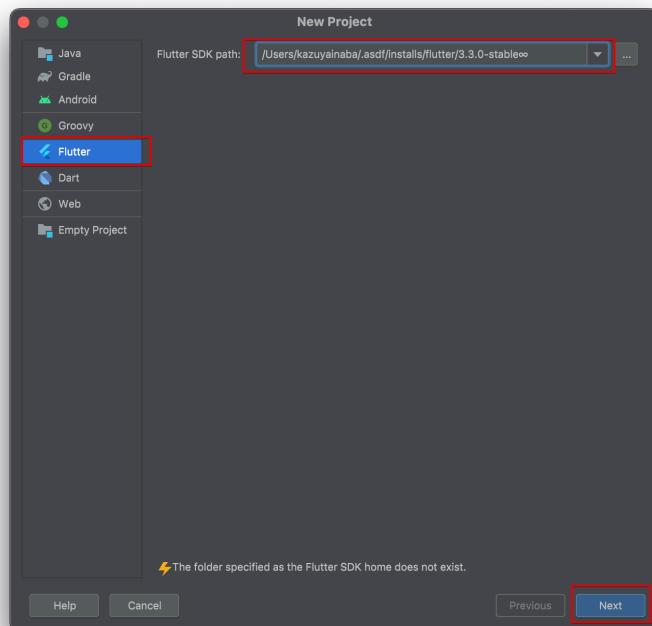
4.1

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。



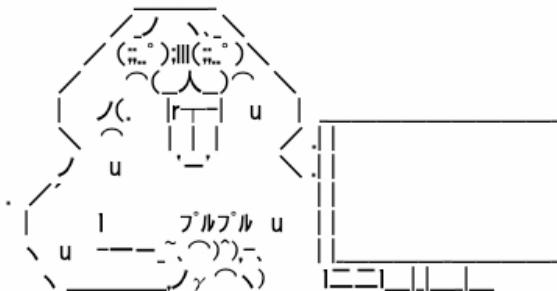
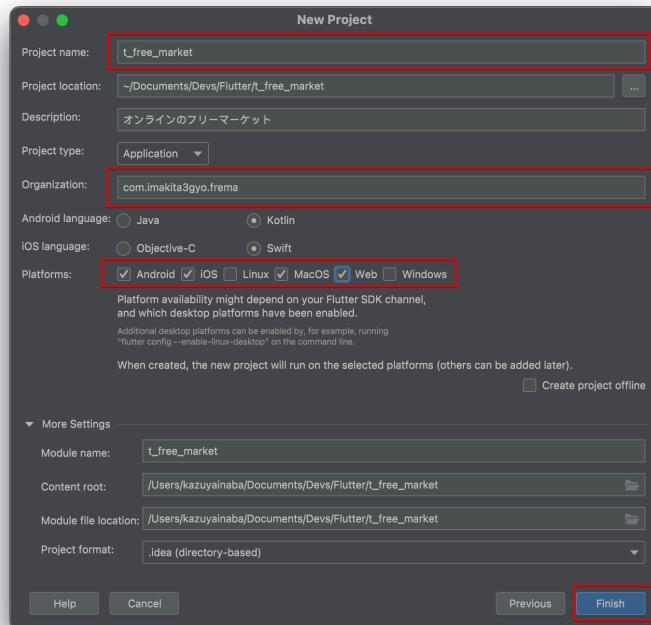
Flutter SDK の場所を訊かれますので、指定するか選択します。



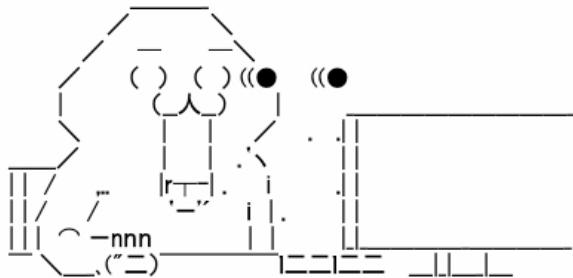
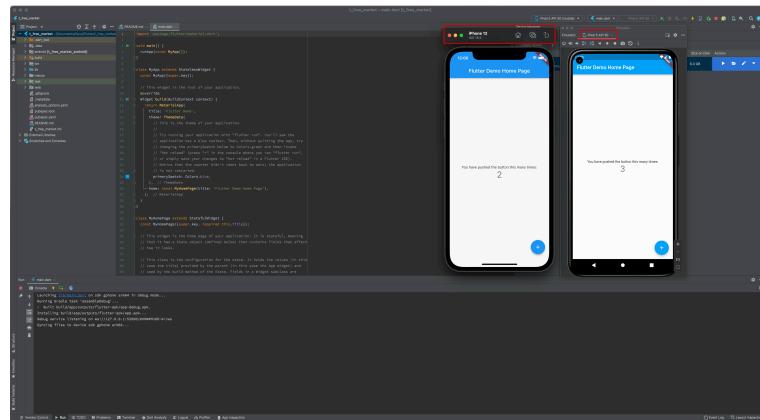
プロジェクトの名前などを設定します。

- 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、できましたでしょうか？

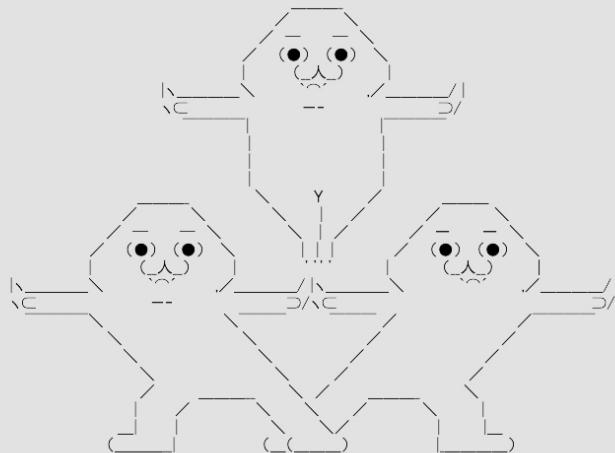
第 5 章

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 を組み合わせて UI を構築します。



【この章の内容】

5.1	UI について	39
5.2	ログインプログラム	40

5.1 UIについて

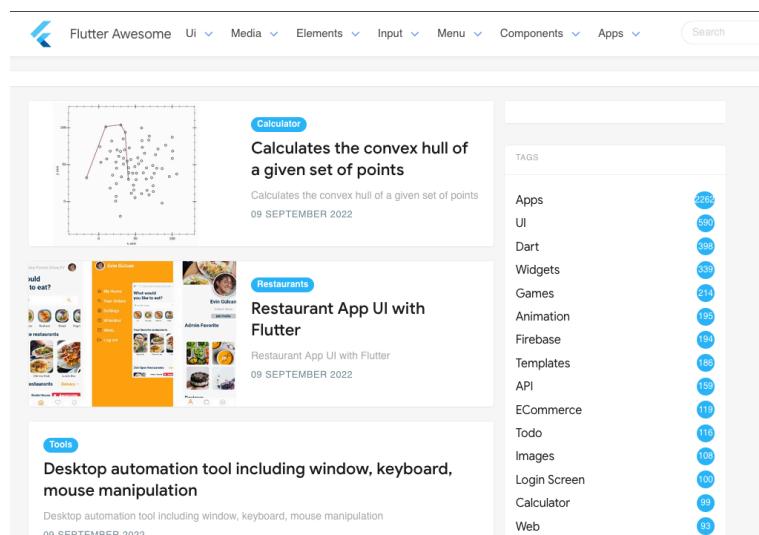
本書では、一画面に表示されるもの全体を指してページと呼びます。

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し世界中へ公開できます。

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 でいえば、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。

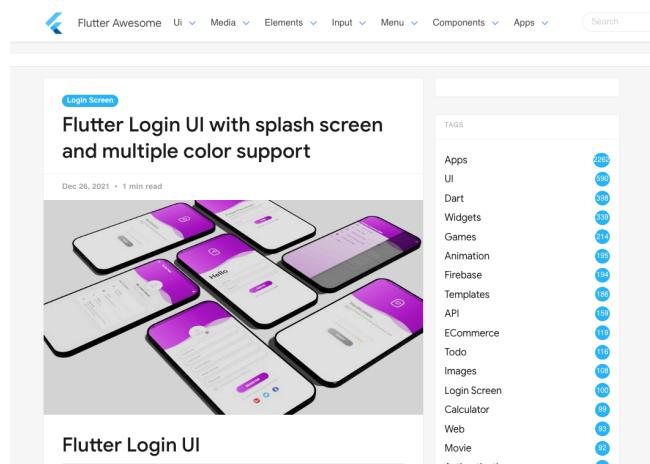
下記のサイト「Flutter Awesome^{*1}」では、UI やアプリがたくさん公開されており、そのほとんどは GitHub などでソースコードを入手できます。実際に手元で動かし、変更を加えてみれば Widget の使い方も理解が進みます。



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

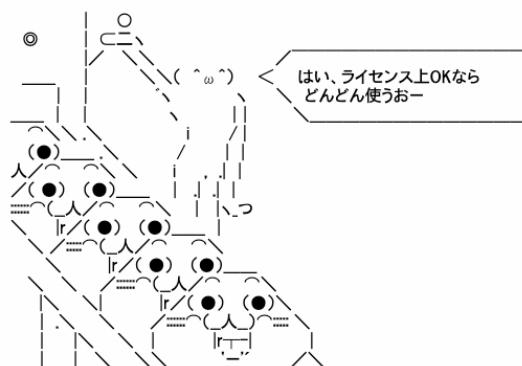
5.2 ログインプログラム

今回使用するのは、上記 FlutterAwesome で見付けたログイン関連のアプリ Flutter Login UI^{*2}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 で作成過程を見る事ができます。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^{*3}を使うことにします。



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

*3 https://github.com/FlutterTutorialNet/flutter_login_ui

♣ 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で写経すると、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）やWarningを出します。。少しでもエラーや警告を減らすため、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

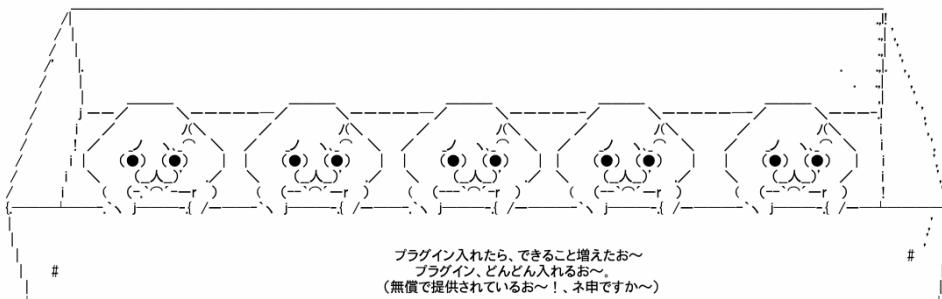
```
dependencies:
  flutter:
    sdk: flutter

  # 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
- 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

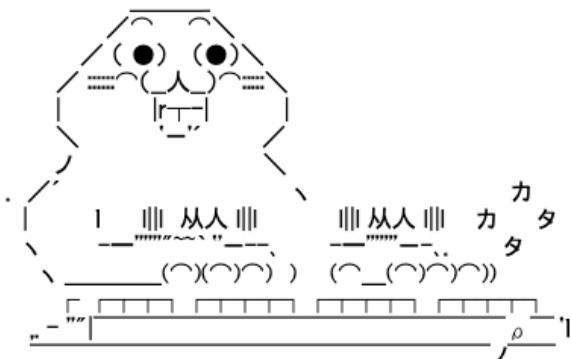
それでは、元プロジェクトをパクっていきましょう。

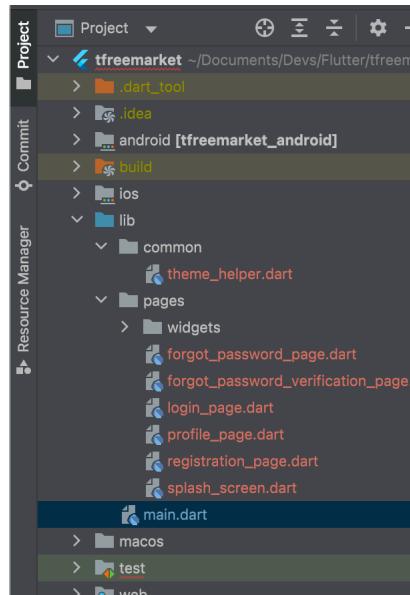
最初にプラグインをインストールします。

▼ 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、ファイルの位置はあとあと変更することとし、今は元プロジェクトと同じにしておきます。

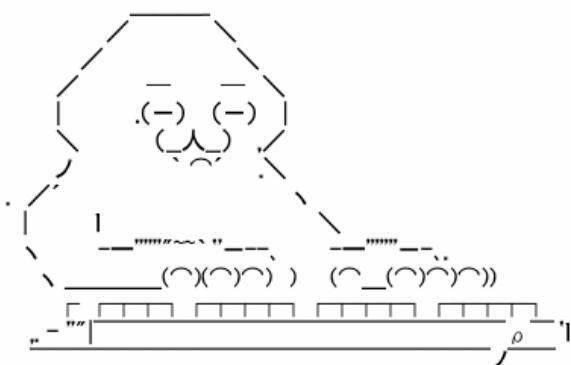




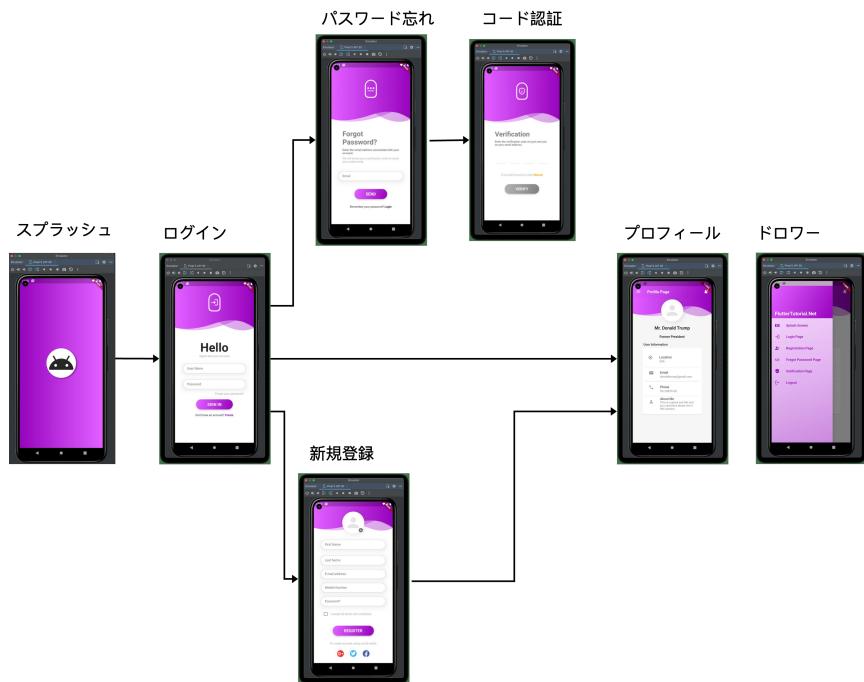
lint で Warning がたくさん出ていますが、コントローラを追加した後すべてのページを、

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

します。その修正時にキレイにします。



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できました。



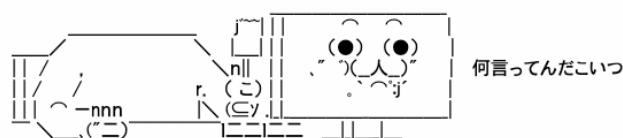
▲図5.1: desc

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ーのデータ型 を作成する。

アプリで使うストア情報（Firebase を使用するユーザーとは別）のデータ型を作成します。

Firebase の認証後、受け取った情報を元に認証コントローラーでフリーマーケット出店者として登録します。



【この章の内容】

6.1 ユーザーモデルの作成	46
----------------	----

6.1 ユーザーモデルの作成

アプリで認証するユーザー（今回は、オンライン・フリーマーケットの出店者で、なおかつ購入者）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。

▼ ユーザーモデル型

```
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

ここまで

▼ GitHub

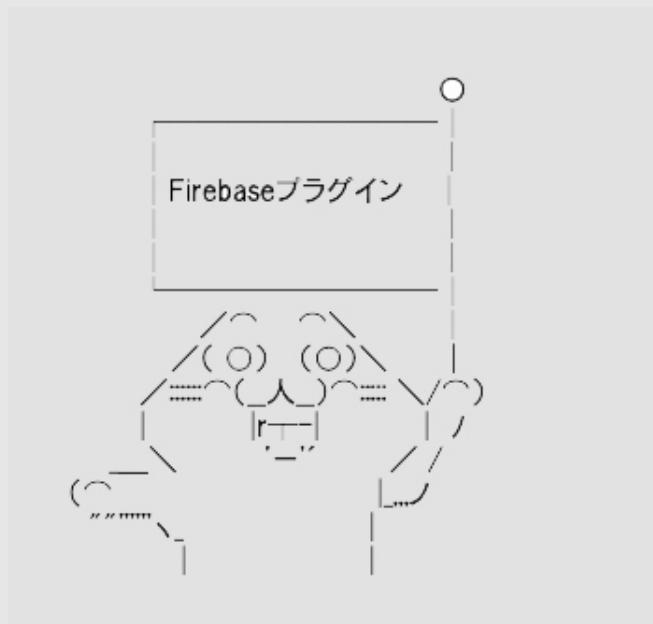
```
>git clone -b 02_login_screens_import https://github.com/risingforce9zz/tf>
>reemarket.git
```

.....

第 7 章

Flutter で Firebase を使う

Firebase 関連のプラグインの導入と設定をします。



【この章の内容】

7.1	クライアント側 (Flutter) で Firebase の設定	49
7.2	Firebase サービス用 Flutter プラグインのインストール	54
7.3	テスト	56

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側が自動生成したファイルを、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、アプリ登録やAndroid、iOS用設定ファイルのダウンロードも自動化されています。

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。

flutterfire コマンドが有効になりましたので、さっそく使ってみます。

▼ 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。

パスの設定が完了しましたら、

▼ 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」で設定します。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

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

```
x 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 Fir  
>ebase configuration and gradle build plugins. Do you want to continue? (y/n) > yes
```

```
Firebase configuration file lib.firebaseio_options.dart generated successfully with the fo  
>llowing 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

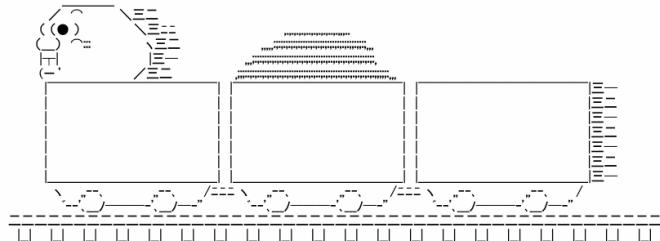
```
Learn more about using this file and next steps from the documentation:
```

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再設定の必要があるので、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の各サービスを使用するために、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 firebase_storage firebase_database  
...  
Changed 22 dependencies!
```

Firebase関連のプラグインをインストールした場合には、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、Android 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 Andoroid 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」にバージョン番号を指定してします。編集は、コメントアウト・新規行の追加を行ます。

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。問題なく起動しましたので次に進みます。

ここまで

ソースコード

▼ GitHub

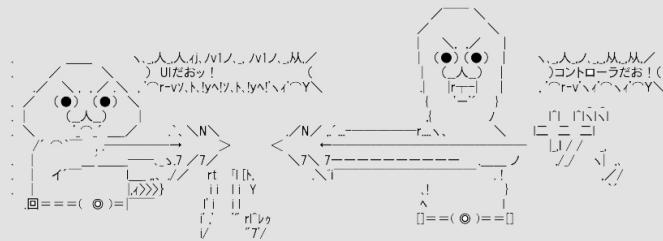
```
>git clone -b 03_flutterfire_setup https://github.com/risingforce9zz/tfree>
>market.git
```



第 8 章

状態管理コントローラを作る

認証状態を管理する状態コントローラを作成します。



コントローラには、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	58
8.2	認証コントローラの作成	60

8.1 GetX コントローラ

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 management 依存管理
- route management ルート管理

が、大きな機能です。

そのほかにも、国際化対応、検証（validation）などがあります。

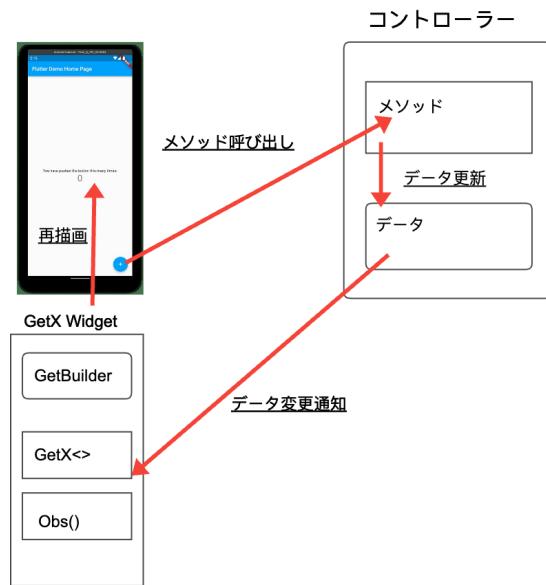
GetX の特徴は、下図のようだ。

- コントローラは、データとデータ更新用メソッドをもつ。
- 表示は、GetX の Widget にコントローラ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。

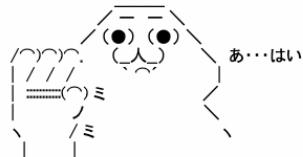
と配置され、

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

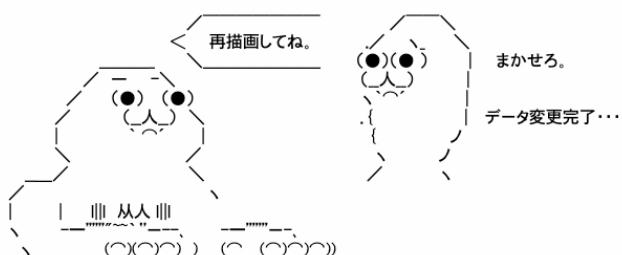
と、流れます。



< 初期画面表示してよ～。



< ボタン押された < コントローラよ、頼んだ！

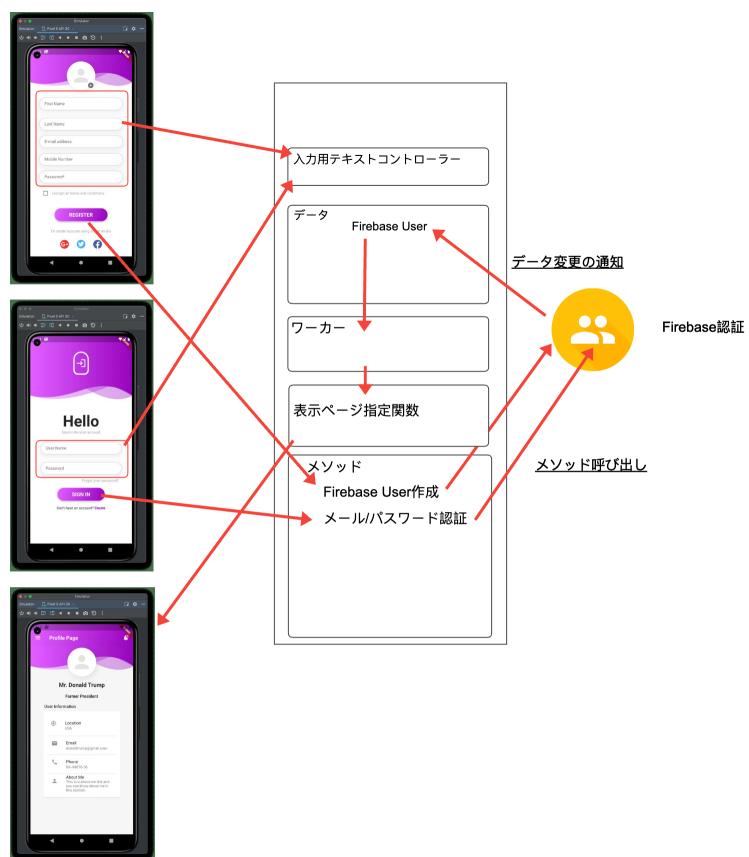


8.2

認証コントローラの作成

認証コントローラ（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、コントローラがもつ。
- サインインなどのメソッドもコントローラがもつ。
- コントローラは、Firebase User データをもつ。
- 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 コントローラはライフサイクルがある。

GetX コントローラには、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 コントローラには、Worker と呼ばれるしくみがあり、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 コントローラがもつデータは、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の作成に入ります。

♣ 8.2.2 AuthController の作成

コントローラ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「handleAuthChanged」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、認証コントローラの根本ができました。次の章からユーザー新規登録ができるようにします。

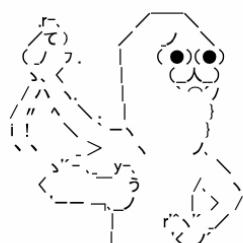
ここまで

までのソースコード

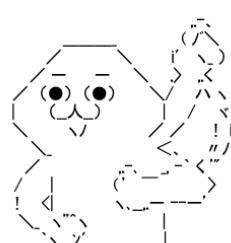
▼ GitHub

```
>git clone -b 04_auth_controller https://github.com/risingforce9zz/tfreema>
rket.git
```

わしは、コントローラーやで。



おいらは、UIやで～。



第9章

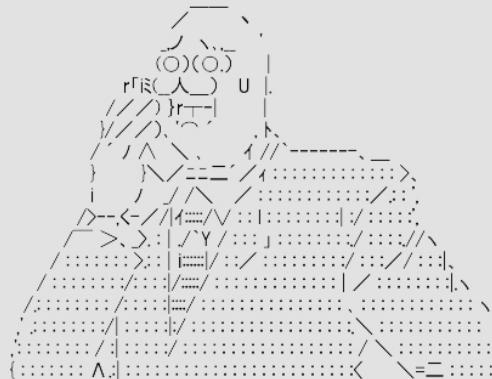
新規アカウント登録

Firebase 認証の新規アカウント作成ができるようになります。

ストア情報コントローラができれば、Firebase 認証に新規登録と同時にストア登録も行います。

なぜ、アカウントを別々に管理するのかと言えば、Firebase Auth のユーザ管理では指定された項目しか管理できないこと Firebase Auth ユーザ情報にアクセスするには、Admin 権限が必要になるためです。

ええっ！新規の登録ですって～



【この章の内容】

9.1	新規登録ができるようにする	66
9.2	新規登録画面の修正	67

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。

本章からは GetX の機能をつかいますので、アプリケーションが GetX アプリケーションとして動作するように変更を加えます。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。

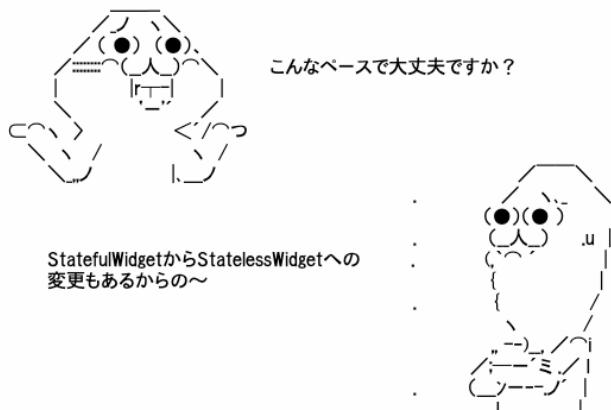
3. 登録ボタンからコントローラのメソッドを実行

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、始めましょう。



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。

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。

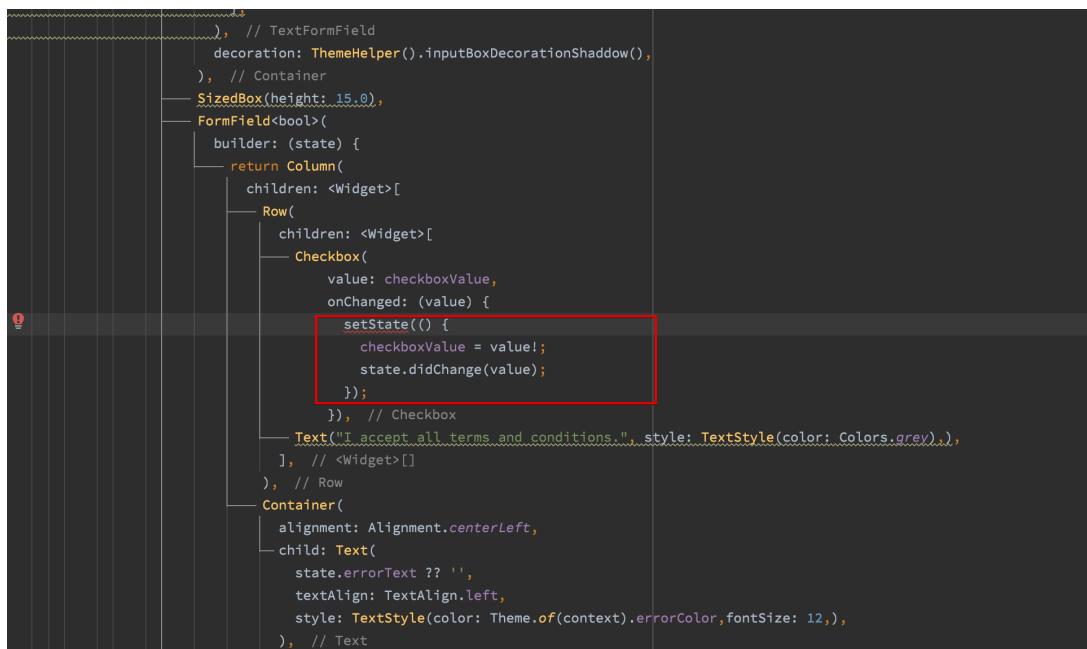
```
import '../../../../../common/theme_helper.dart';
import './widgets/header_widget.dart';

class RegistrationPage extends StatelessWidget { // Red box highlights here
  @override
  State< StatefulWidget > createState() {
    return _RegistrationPageState();
  }
}

class _RegistrationPageState extends State< RegistrationPage > { // Red box highlights here
  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の TextEditingController 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。

認証コントローラ側の TextEditingController には、上記のフィールド名 + Controller で命名しております。これらコントローラと画面上の TextField を結び付け、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 (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付けます。

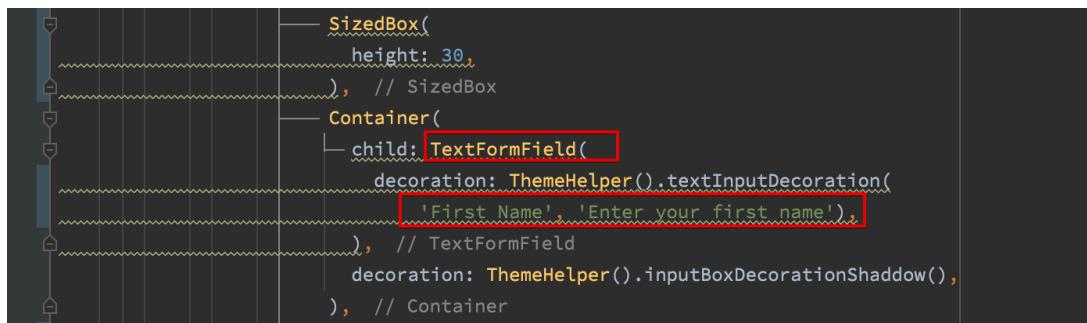
AuthController のインスタンス化は、のちほどスプラッシュスクリーンの起動で行うようにします。

▼ 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します。

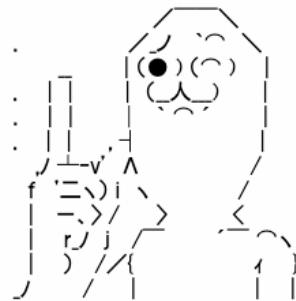
メールアドレス入力欄、電話番号の入力欄、パスワードの入力欄には、入力時のエラーチェック「validator」プロパティがありますが、入力値検証は一ヵ所にまとめるため削除します。



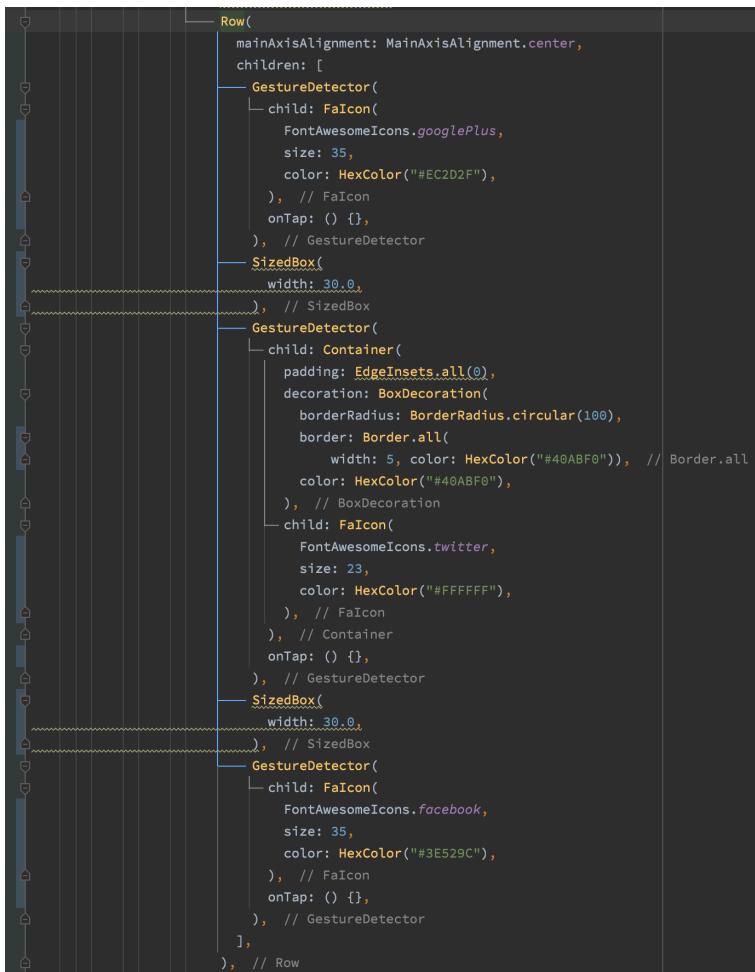
The screenshot shows a portion of a Dart file with code for a Container widget. Inside the container, there is a TextField with various properties like decoration and keyboardType. A red box highlights the validator block, which contains an if statement checking if the value is empty and then using a regular expression to validate the email address. A red '削除' (Delete) annotation is placed over this code block.

```
Container(
  child: TextField(
    decoration: ThemeHelper().textInputDecoration(
      "メールアドレス", "メールアドレスを入力してください。"),
    keyboardType: TextInputType.emailAddress,
    validator: (val) {
      if (!val!.isEmpty) &&
          !RegExp(r"^[a-zA-Z0-9_.!#$%^&*+/=?^`{|}~-]+@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,253}[a-zA-Z0-9])?(?:(?:[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,253}[a-zA-Z0-9])?\.)+[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,253}[a-zA-Z0-9])?)?@([a-zA-Z0-9](?:[a-zA-Z0-9-]{0,253}[a-zA-Z0-9])?\.)+[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,253}[a-zA-Z0-9])?").hasMatch(val)) {
        return "Enter a valid email address";
      }
      return null;
    },
  ), // TextField
  decoration: ThemeHelper().inputBoxDecorationShadow(),
), // Container
```

できる子は、
TextField
TextFormField
の違いを調べてみるとよいぞ！



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、Row Widget ごと削除します。



▲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。

エラー表示用のSnackBarを作成

登録ボタンがクリックされコントローラの新規登録メソッド前に、入力値の検証をしエラーが出た場合は、SnackBarを使って表示します。

GetXは、SnackBarを簡単に出すことができます。

エラー表示用の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）へ以下のバリデーション・コードを追加します。

GetX には、データの Validation 機能もあります。

▼ 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;
}
```

ページ上のデータ変更に伴う再描画

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は、チェックボックの値を持つ変数「checkboxValue」に格納されます。しかし、変数の値が更新されても再描画はされません。

元は、ページが StatefulWidget でしたので変数の値の変更を「setState 関数」で行い、「setState 関数」が StatefulWidget に再描画を指示していました。

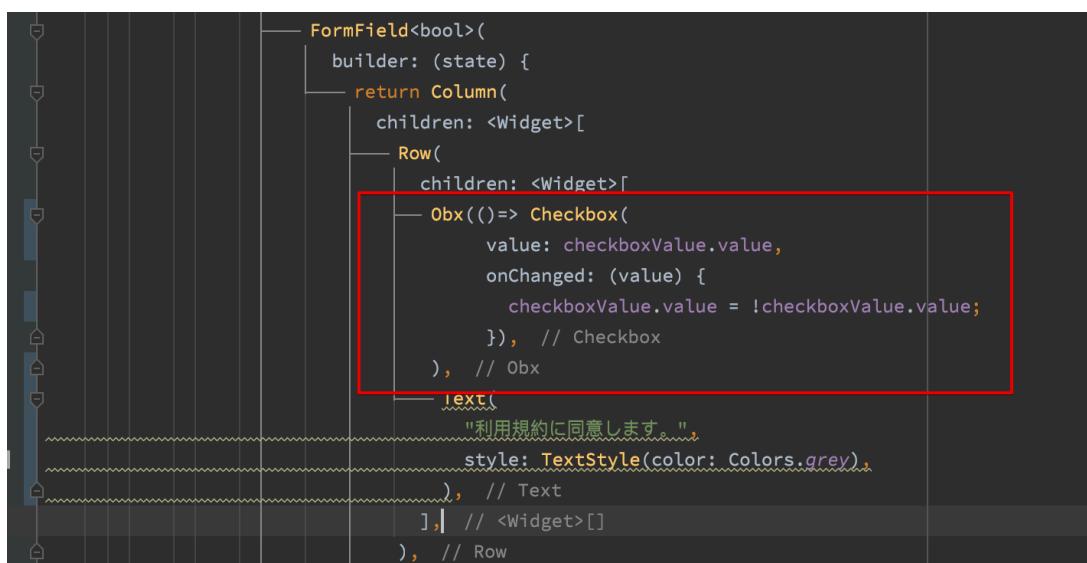
現在は、 StatelessWidget へ変更しましたので、チェックボックスの値の「checkboxValue」を GetX を使って監視対象にします。

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

「obs」は、GetX の提供するオブザーバーです。表示部分を GetX の Widget で囲うことで値の変化があれば、再描画の通知がされます。

「obs」、「Rx<T>」などの GetX のオブザーバーに変更しますと、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
- isNullOrEmpty
- 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^{*1}を見て決めてください。

登録ボタン

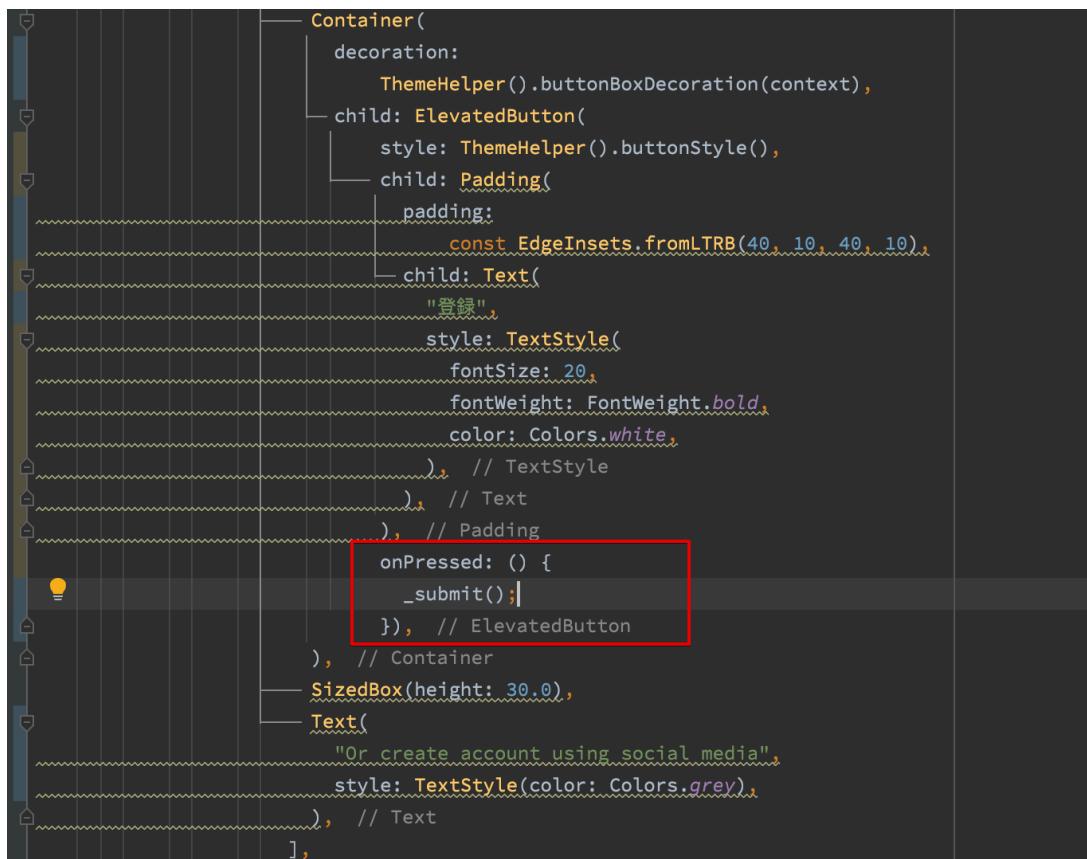
入力値検証後、認証コントローラの新規登録メソッドを呼ぶようにします。認証コントローラの「signup」メソッドは未作成ですので、現時点ではエラーになります。

▼ 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。

^{*1} https://github.com/jonataslaw/getx/blob/master/lib/get_utils/src/get_utils/get_utils.dart



```
Container(
  decoration:
    ThemeHelper().buttonBoxDecoration(context),
  child: ElevatedButton(
    style: ThemeHelper().buttonStyle(),
    child: Padding(
      padding:
        const EdgeInsets.fromLTRB(40, 10, 40, 10),
      child: Text(
        "登録",
        style: TextStyle(
          fontSize: 20,
          fontWeight: FontWeight.bold,
          color: Colors.white,
        ), // TextStyle
      ), // Text
    ), // Padding
    onPressed: () {
      _submit();
    }, // ElevatedButton
  ), // Container
  SizedBox(height: 30.0),
  Text(
    "Or create account using social media",
    style: TextStyle(color: Colors.grey),
  ), // Text
],
```

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を初期化

アプリケーション起動時に表示されるページの「スプラッシュスクリーン」へ認証コントローラ(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かまいません。

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、ほかのページからは「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();
}

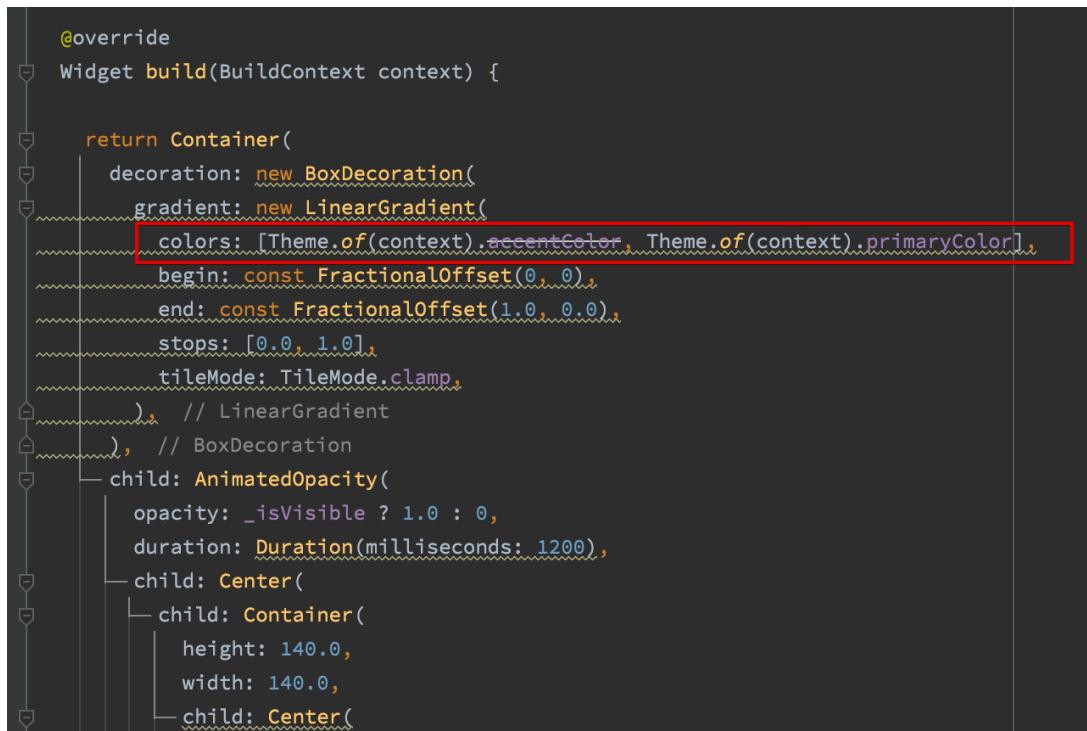
class _SplashScreenState extends State<SplashScreen> {
  bool _isVisible = false;

  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
      });
    });
  }; // Timer
}
```

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^{*2}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,  
,
```

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

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

に変更します。そのほかのファイルにも「`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/パスワードでログインを行うためには、新規ユーザー作成メソッドとして「`createUserWithEmailAndPassword()`」が提供されています。

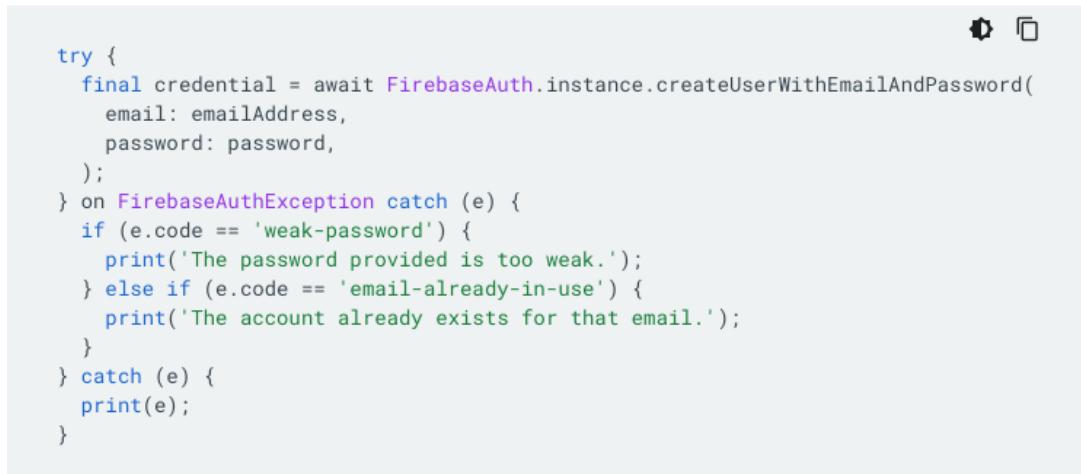
詳細は、[公式サイト](#)^{*3}を確認してください。

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);  
}
```

▲図 9.5: Firebase のドキュメント

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

認証コントローラ（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);
        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は、タイマーでログインページに移動するよう設定されていますが、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

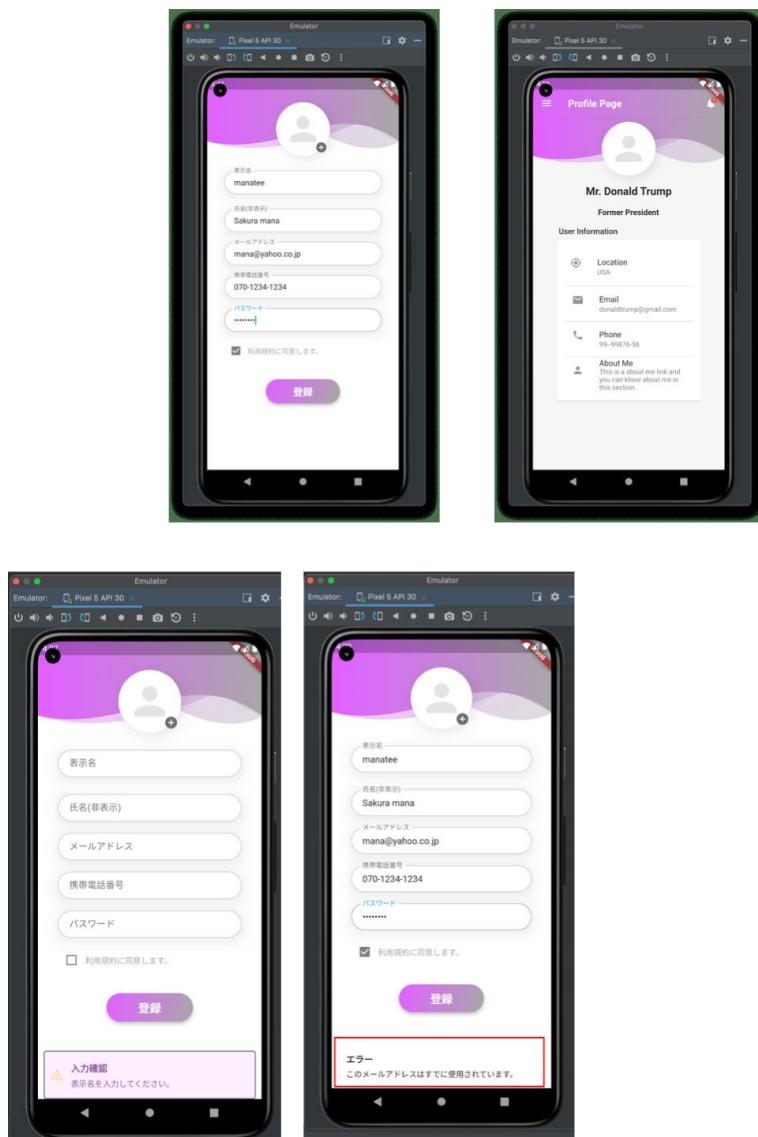
```
class AppRoutes {  
  AppRoutes._();  
  static final routes = [  
    GetPage(name: '/', page: () => SplashScreen(title: "")),  
    GetPage(name: '/login', page: () => const LoginPage()),  
    GetPage(name: '/profile', page: () => ProfilePage()),  
  ];  
}
```



♣ 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、テスト目的ならメールアドレスの書式さえ合っていれば「example@example.com」でもユーザー登録できます。

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 9.6: 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 UI...". To its right is a blue button labeled "ユーザーを追加". Above the table, there are three small icons: a bell, a person, and a gear. Below the table, there are navigation arrows and text indicating "1 - 1 of 1".

ID	プロバ イダ	作成日	ログイ ン日	ユーザー UID
mana@yahoo...	✉	202...	202...	n1YDXX4EBTS24p...

▲図 9.7: Firebase 認証

ここまで

までのソースコード

▼ GitHub

```
>git clone -b 05_signup https://github.com/risingforce9zz/tfreemarket.git
```

第 10 章

メール/パスワードでのサインイン

新規アカウントが作成できるようになったので、メール/パスワードでサインインするメソッドを実装します。

また、前章で新規登録画面から SNS サインインのボタンを削除したため、新たにログイン方法選択画面を作成します。サインイン成功時にページ推移するプロフィール画面には、サインアウトボタンを作成します。



【この章の内容】

10.1	メール/パスワードでサインインできるようになるまで	88
10.2	サインイン選択画面を作成	89
10.3	LoginPage の修正	94
10.4	ForgotPasswordPage の修正	98
10.5	サインアウトの実装	103
10.6	パスワード・リセット要求完了ページ	104
10.7	認証コントローラにメソッドの実装	107
10.8	動作確認	109

10.1 メール/パスワードでサインインできるようになるまで

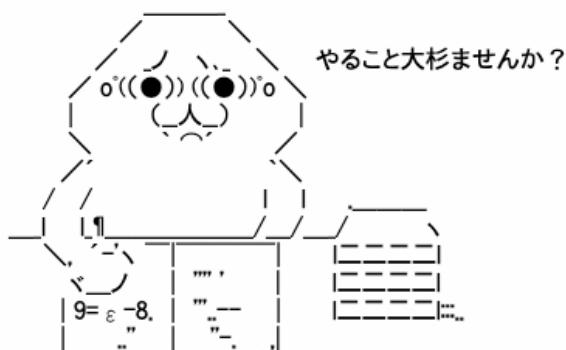
メール/パスワードでサインインするためには、

- ログイン画面
- ログイン成功時に表示するページ
- パスワード・リセットの要求ページ
- パスワード・リセット要求完了ページ

が必要になります。

幸い、元プロジェクトには必要なページが含まれていますので修正して使います。

また、サンプルアプリケーションとしての拡張を考え、SNS認証の種類を増やせるようにログイン方法選択画面を追加します。



10.2 サインイン選択画面を作成

今回のアプリケーションは、SNS 認証を含め 4 つのサインイン方法を提供しています。

メール/パスワード認証は入力画面が必要ですが、SNS 認証はボタンのみ実装できます。仮に、SNS 実装が増えてボタンをひとつ増やし、認証コントローラにログイン・メソッドを実装するだけになります。

Firebase 認証は、

今回実装する、「Twitter」、「Google」、「Apple」だけでなく「Facebook」、「GitHub」、「Microsoft」、「米 Yahoo!」などの SNS ログインに対応しています。

将来的に、ログイン方法を増やすことが容易になるよう新しく「ログイン方法選択画面」を作成します。



▲図 10.1: 作成する画面

「lib/pages」フォルダに「select_signin_page.dart」ファイルを作成します。

▼ select_signin_page.dart

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

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

class SelectSignInPage extends StatelessWidget{
  const LoginPage({Key? key}): super(key:key);

  final double _headerHeight = 250;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.white,
      body: SingleChildScrollView(
        child: Column(
          children: [
            SizedBox(
              height: _headerHeight,
              child: HeaderWidget(_headerHeight, true, Icons.storefront), //let's
            ),
            > create a common header widget
            ),
            SafeArea(
              child: Container(
                padding: const EdgeInsets.fromLTRB(20, 0, 20, 0),
                margin: const EdgeInsets.fromLTRB(20, 0, 20, 0),// This will be
            > the login form
              child: Column(
                children: [
                  const SizedBox(height: 20.0,),
                  const Text(
                    '以下の方法でログインできます。',
                    style: TextStyle(color: Colors.grey),
                  ),
                  const SizedBox(height: 20.0,),
                  Container(
                    decoration: ThemeHelper().buttonBoxDecoration(context),
                    child: ElevatedButton(
                      style: ThemeHelper().buttonStyle(),
                      child: Row(
                        children: const <Widget>[
                          FaIcon(FontAwesomeIcons.envelope, size: 23),
                          Padding(
                            padding: EdgeInsets.fromLTRB(30, 10, 0, 10),
                            child: Text('Emailとパスワード', style: TextStyle(
                              fontSize: 20, fontWeight: FontWeight.bold, color: Colors.white),),
                          ),
                        ],
                    
```

```
        ),
        onPressed: (){
            Get.toNamed('/login');
        },
    ),
),
const SizedBox(height: 20.0,),
Container(
    decoration: ThemeHelper().buttonBoxDecoration(context),
    child: ElevatedButton(
        style: ThemeHelper().buttonStyle(),
        child: Row(
            children: const [
                FaIcon(FontAwesomeIcons.twitter, size: 23, color:>
Colors.blue),
                Padding(
                    padding: EdgeInsets.fromLTRB(30, 10, 0, 10),
                    child: Text('Twitterアカウント', style: TextStyle(
fontSize: 20, fontWeight: FontWeight.bold, color: Colors.white),),
                ),
            ],
        ),
        onPressed: (){
        },
    ),
),
const SizedBox(height: 20.0,),
Container(
    decoration: ThemeHelper().buttonBoxDecoration(context),
    child: ElevatedButton(
        style: ThemeHelper().buttonStyle(),
        child: Row(
            children: const [
                FaIcon(FontAwesomeIcons.google, size: 23),
                Padding(
                    padding: EdgeInsets.fromLTRB(30, 10, 0, 10),
                    child: Text('Googleアカウント', style: TextStyle(
fontSize: 20, fontWeight: FontWeight.bold, color: Colors.white),),
                ),
            ],
        ),
        onPressed: (){
        },
    ),
),
const SizedBox(height: 20.0,),
Container(
    decoration: ThemeHelper().buttonBoxDecoration(context),
    child: ElevatedButton(
        style: ThemeHelper().buttonStyle(),
        child: Row(
            children: const [
                FaIcon(FontAwesomeIcons.apple, size: 23),
            ],
        ),
    ),
)
```

```
        Padding(
            padding: EdgeInsets.fromLTRB(30, 10, 0, 10),
            child: Text('Appleアカウント', style: TextStyle(
>fontSize: 20, fontWeight: FontWeight.bold, color: Colors.white),),
                ),
            ],
            onPressed: () {
            },
        ),
        const SizedBox(height: 20.0),
        Container(
            decoration: ThemeHelper().buttonBoxDecoration(context),
            child: ElevatedButton(
                style: ThemeHelper().buttonStyle(),
                child: Row(
                    children: const [
                        FaIcon(FontAwesomeIcons.userPlus, size: 23),
                        Padding(
                            padding: EdgeInsets.fromLTRB(30, 10, 40, 10),
                            child: Text('新規作成', style: TextStyle(fontSiz>e: 20, fontWeight: FontWeight.bold, color: Colors.white),),
                        ),
                    ],
                ],
                onPressed: () {
                    Get.toNamed('/registration');
                },
            ),
        ),
    ],
),
),
),
);
);

}
}
```

Route に、「SelectSignInPage」を加え、スプラッシュスクリーンから「SelectSignInPage」へ推移するよう変更します。また、「LoginPage」、「ForgotPasswordPage」、「ForgotPasswordVerificationPage」も Route へ追加します。

▼ Route

```
class AppRoutes {  
  AppRoutes._();  
  
  static final routes = [  
    GetPage(name: '/', page: () => SplashScreen(title: "")),  
    GetPage(name: '/select-signin', page: () => SelectSignInPage()),  
    GetPage(name: '/login', page: () => LoginPage()),  
    GetPage(name: '/forgotpassword', page: () => ForgotPasswordPage()),  
    GetPage(  
      name: '/forgotpassword-verify',  
      page: () => const ForgotPasswordVerificationPage()),  
    GetPage(name: '/registration', page: () => RegistrationPage()),  
    GetPage(name: '/profile', page: () => ProfilePage()),  
  ];  
}
```

▼ スプラッシュスクリーンの移動先の変更

```
class _SplashScreenState extends State<SplashScreen> {  
  bool _isVisible = false;  
  
  AuthController authController = Get.put(AuthController());  
  _SplashScreenState(){  
  
    new Timer(const Duration(milliseconds: 2000), (){  
      setState(() {  
        Get.toNamed('/select-signin'); ←ログイン方法選択画面へ  
      });  
    });  
  };
```

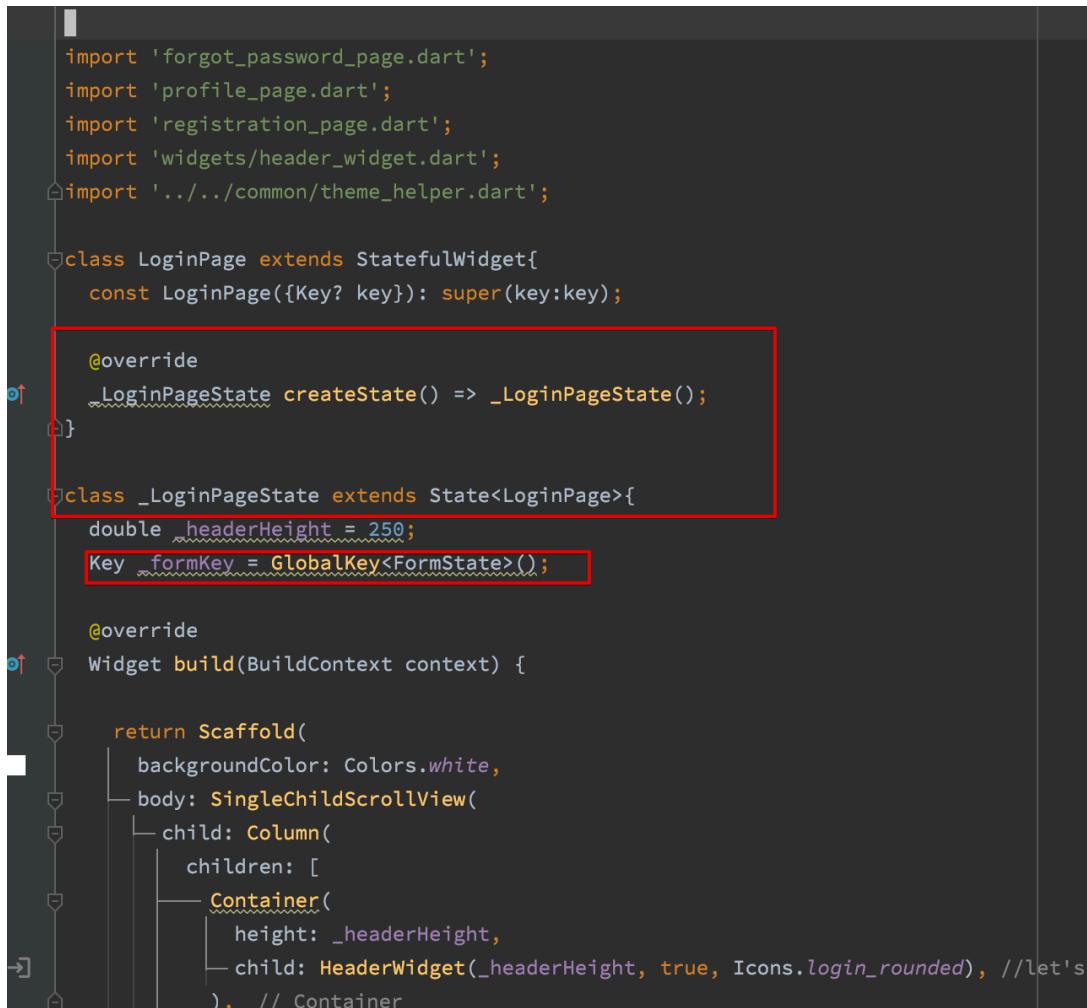
10.3 LoginPage の修正

ログインページを修正します。

- StatelessWidgetへの変更
- 入力フィールドと認証コントローラの TextEditingController の結び付け
- 日本語化
- パスワード忘れのページ推移

♣ 10.3.1 StatelessWidgetへの変更

LoginPage (lib/pages/login_page.dart) を開き、赤マスク部分を削除します。「extends」後の継承元を「 StatefulWidget」から「 StatelessWidget」へ変更します。



```
import 'forgot_password_page.dart';
import 'profile_page.dart';
import 'registration_page.dart';
import 'widgets/header_widget.dart';
import '../../../../../common/theme_helper.dart';

class LoginPage extends StatefulWidget{
    const LoginPage({Key? key}): super(key:key);

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

class _LoginPageState extends State<LoginPage>{
    double _headerHeight = 250;
    Key _formKey = GlobalKey<FormState>();

    @override
    Widget build(BuildContext context) {

        return Scaffold(
            backgroundColor: Colors.white,
            body: SingleChildScrollView(
                child: Column(
                    children: [
                        Container(
                            height: _headerHeight,
                            child: HeaderWidget(_headerHeight, true, Icons.login_rounded), //let's
                        ), // Container
                    ],
                ),
            ),
        );
    }
}
```

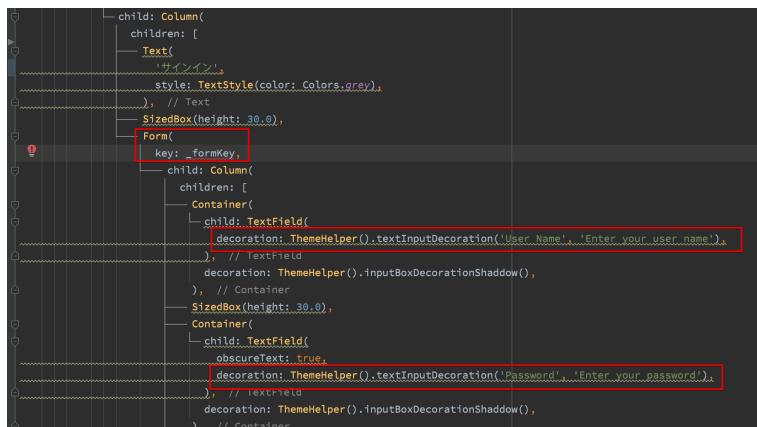
♣ 10.3.2 入力フィールドと認証コントローラの結び付け

認証コントローラを LoginPage にバインドします。

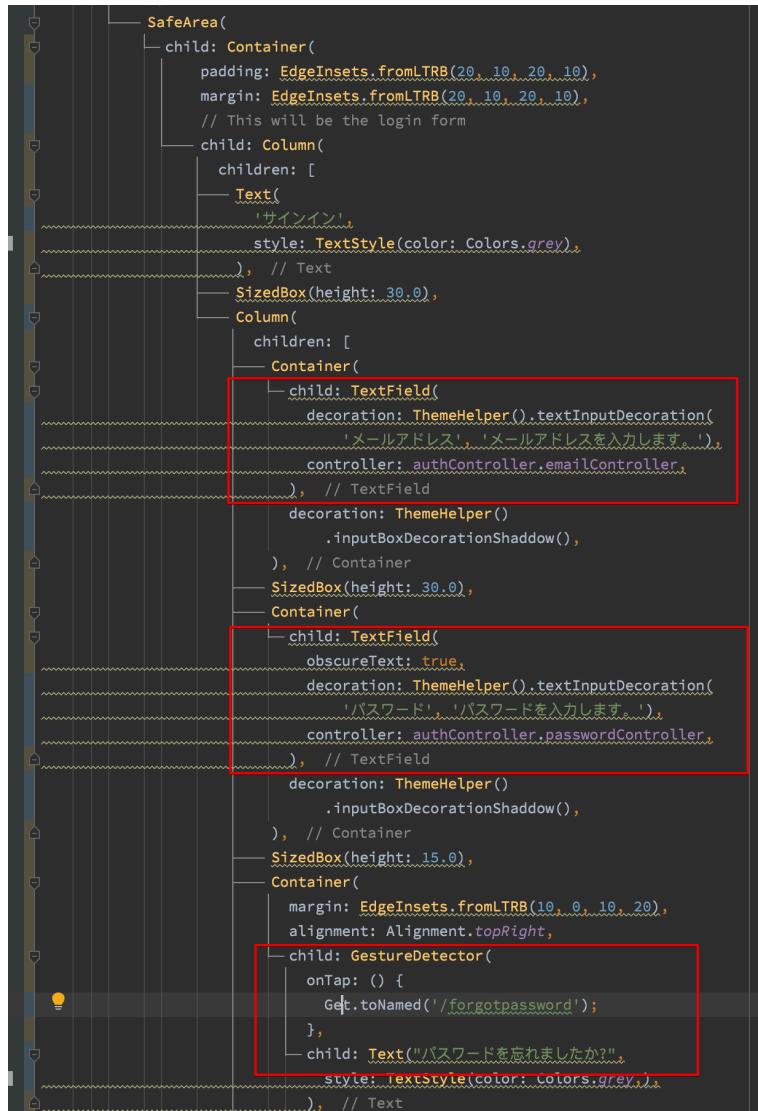
▼ LoginPage へ認証コントローラをバインド

```
class LoginPage extends StatelessWidget {  
  LoginPage({Key? key}) : super(key: key);  
  
  double _headerHeight = 250;  
  
  final AuthController authController = AuthController.to;
```

メールアドレス、パスワード入力欄の「TextField の controller」プロパティに認証コントローラの TextEditingController を指定します。日本語化も同時に行います。



▲ 図 10.2: 変更前



▲図 10.3: 変更後

♣ 10.3.3 パスワード・リセットページへの推移

ページのナビゲーションを GetX に変更します。Route はすでに追加してあります。

▼ ナビゲーションの変更

```
Container(  
    margin: EdgeInsets.fromLTRB(10, 0, 10, 20),  
    alignment: Alignment.topRight,  
    child: GestureDetector(  
        onTap: () {  
            Get.toNamed('/forgotpassword');  
        },  
        child: Text("パスワードを忘れましたか?",  
            style: TextStyle(color: Colors.grey,),  
        ),  
    ),  
,
```

10.4 ForgotPasswordPage の修正

パスワード・リセットページを修正します。

- StatelessWidget へ変更
- 認証コントローラのバインドし、メールアドレス欄にコントローラをセット
- 日本語化
- ログインページ戻るナビゲーションを GetX に
- 送信ボタンに認証コントローラのパスワードリセットをバインド
- メールアドレス欄の入力検証

♣ 10.4.1 StatelessWidget へ変更

赤ワク内を削除し、「StatefulWidget」を「StatelessWidget」へ変更します。

```
class ForgotPasswordPage extends StatelessWidget {  
    const ForgotPasswordPage({Key? key}) : super(key: key);  
  
    @override  
    _ForgotPasswordPageState createState() => _ForgotPasswordPageState();  
}  
  
class _ForgotPasswordPageState extends State<ForgotPasswordPage> {  
    final GlobalKey<FormState> _formKey = GlobalKey<FormState>();  
  
    @override  
    Widget build(BuildContext context) {  
        double _headerHeight = 300;  
        return Scaffold(  
            backgroundColor: Colors.white,  
            body: SingleChildScrollView(  
                child: Column(  
                    children: [  
                        Container(  
                            height: _headerHeight,  
                            child: HeaderWidget(_headerHeight, true, Icons.password_rounded),  
                        ), // Container  
                    ],  
                ),  
            ),  
        );  
    }  
}
```

▲図 10.4: StatelessWidget へ

♣ 10.4.2 認証コントローラのバンドル

▼ 認証コントローラのバンドル

```
class ForgotPasswordPage extends StatelessWidget {  
  ForgotPasswordPage({Key? key}) : super(key: key);  
  
  final AuthController authController = AuthController.to;  
  
  @override  
  Widget build(BuildContext context) {  
    double _headerHeight = 300;  
    return Scaffold(  
      backgroundColor: Colors.white,  
      body: SingleChildScrollView(
```

メールアドレス欄に認証コントローラの TextEditingController をバインド。

```
Container(
  alignment: Alignment.topLeft,
  margin: EdgeInsets.fromLTRB(20, 0, 20, 0),
  child: Column(
    mainAxisAlignment: MainAxisAlignment.start,
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text('パスワードを忘れましたか？',
        style: TextStyle(
          fontSize: 35,
          fontWeight: FontWeight.bold,
          color: Colors.black54
        ), // TextStyle
        // textAlign: TextAlign.center,
      ), // Text
      SizedBox(height: 10),
      Text('登録時のメールアドレスを入力してください。',
        style: TextStyle(
          // fontSize: 20,
          fontWeight: FontWeight.bold,
          color: Colors.black54
        ), // TextStyle
        // textAlign: TextAlign.center,
      ), // Text
      SizedBox(height: 10),
      Text('パスワードリセットリンクを入力されたメールアドレスに送ります。',
        style: TextStyle(
          color: Colors.black38,
          // fontSize: 20,
        ), // TextStyle
        // textAlign: TextAlign.center,
      ), // Text
    ],
  ), // Column
), // Container
SizedBox(height: 40.0),
Column(
  children: <Widget>[
    Container(
      child: TextField(
        decoration: ThemeHelper().textInputDecoration("メールアドレス", "メールアドレスを入力してください。"),
        controller: authController.emailController,
      ), // TextField
      decoration: ThemeHelper().inputBoxDecorationShadow(),
    ), // Container
    SizedBox(height: 40.0),
    Container(
      decoration: ThemeHelper().buttonBoxDecoration(context),
      child: ElevatedButton(
        style: ThemeHelper().buttonStyle(),
        child: Padding(
          padding: const EdgeInsets.fromLTRB(
            40, 10, 40, 10), // EdgeInsets.fromLTRB
          child: Text(
            "送信"
          )
        )
      )
    )
  ]
)
```

♣ 10.4.3 LoginPage へ戻るナビゲーション

戻るリンクのナビゲーションを GetX にします。Route にはすでにあります。

▼ ナビゲーションリンク

```
Text.rich(
    TextSpan(
        children: [
            TextSpan(text: "思い出しましたか? "),
            TextSpan(
                text: 'サインインへ',
                recognizer: TapGestureRecognizer()
                    ..onTap = () {
                        Get.toNamed('/login');
                    },
                style: TextStyle(
                    fontWeight: FontWeight.bold
                ),
            ),
        ],
    ),
),
```

♣ 10.4.4 送信ボタンへパスワードリセットメソッドをバインド

メールアドレス欄に入力された値を検証し、問題なければ認証コントローラの「パスワードリセット」メソッドを実行します。

▼ 入力欄の検証とメソッド実行

```
/// 送信ボタンクリック時にメールアドレス・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() {
    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。', 'warning');
        return false;
    }
    return true;
}

/// 入力値を検証し送信
void _submit() {
    if (_validate()) {
        authController.passwordReset();
    }
}
```

送信ボタンへメソッドの割り当てをします。

▼ パスワード・リセット要求メソッドの割り当て

```
Container(  
  decoration: ThemeHelper().buttonBoxDecoration(context),  
  child: ElevatedButton(  
    style: ThemeHelper().buttonStyle(),  
    child: Padding(  
      padding: const EdgeInsets.fromLTRB(40, 10, 40, 10),  
      child: Text(  
        "送信",  
        style: TextStyle(  
          fontSize: 20,  
          fontWeight: FontWeight.bold,  
          color: Colors.white,  
        ),  
      ),  
    ),  
    onPressed: () {  
      _submit();  
    },  
  ),  
,
```

10.5 サインアウトの実装

サインインができると当然ですがサインアウトの機能も必要です。次の号でドロワーを実装し、ドロワーにサインアウトできる機能を置く予定です。ログイン後にアクセスする全ページにドロワーがありますので、どのページからもサインアウトができます。

それまでは、サインイン後にページ推移するプロフィールページへサインアウトボタンを置きます。

サインアウト・メソッドは、認証コントローラが持っていますので、バインドします。

▼認証コントローラのバインド

```
class _ProfilePageState extends State<ProfilePage>{

    AuthController authController = AuthController.to;
    double _drawerIconSize = 24;
    double _drawerFontSize = 17;
```

プロフィールページ（profile_page.dart）へサインアウトボタンを追加し、サインアウト・メソッドの呼び出します。

▼リスト 10.1: desc

```
//
Container(
  decoration:
    ThemeHelper().buttonBoxDecoration(context),
  child: ElevatedButton(
    style: ThemeHelper().buttonStyle(),
    child: const Padding(
      padding: EdgeInsets.fromLTRB(40, 10, 40, 10),
      child: Text(
        "サインアウト",
        style: TextStyle(
          fontSize: 20,
          fontWeight: FontWeight.bold,
          color: Colors.white,
        ),
      ),
    ),
    onPressed: () {
      authController.signInOut();
    },
  ),
),
```

10.6 パスワード・リセット要求完了ページ

Firebase認証で使うメール/パスワードでのパスワード・リセットは、

1. パスワード・リセットの要求。
2. Firebaseから登録メールアドレスへリセット用リンクをメールで送信。
3. リンクをクリックし表示されるページで新パスワードの登録。

となります。

パスワード・リセット要求ページはできましたので、要求成功時に表示されるページを用意します。

元プロジェクトには、「forgot_password_validation_page.dart」がありますので、これを利用します。

以下が、「forgot_password_validation_page.dart」の変更後になります。

▼パスワード・リセット要求完了ページ

```
import 'package:flutter/gestures.dart';
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:otp_text_field/otp_field.dart';
import 'package:otp_text_field/style.dart';

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

class ForgotPasswordVerificationPage extends StatelessWidget {
  const ForgotPasswordVerificationPage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    double _headerHeight = 300;

    return Scaffold(
      backgroundColor: Colors.white,
      body: SingleChildScrollView(
        child: Column(
          children: [
            Container(
              height: _headerHeight,
              child: HeaderWidget(
                _headerHeight, true, Icons.privacy_tip_outlined),
            ),
            SafeArea(
              child: Container(
```

```
margin: EdgeInsets.fromLTRB(25, 10, 25, 10),
padding: EdgeInsets.fromLTRB(10, 0, 10, 0),
child: Column(
    children: [
        Container(
            alignment: Alignment.topLeft,
            margin: EdgeInsets.fromLTRB(20, 0, 20, 0),
            child: Column(
                mainAxisAlignment: MainAxisAlignment.start,
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                    Text(
                        'パスワードリセット',
                        style: TextStyle(
                            fontSize: 28,
                            fontWeight: FontWeight.bold,
                            color: Colors.black54),
                        // textAlign: TextAlign.center,
                    ),
                    SizedBox(
                        height: 10,
                    ),
                    Text(
                        '入力したメールアドレス宛にパスワードリセットリンクを送りました。',
                        style: TextStyle(
                            // fontSize: 20,
                            fontWeight: FontWeight.bold,
                            color: Colors.black54),
                        // textAlign: TextAlign.center,
                    ),
                ],
            ),
        ),
        SizedBox(height: 40.0),
        Column(
            children: <Widget>[
                SizedBox(height: 50.0),
                SizedBox(height: 40.0),
                Container(
                    decoration: ThemeHelper().buttonBoxDecoration(
                        context, "#AAAAAA", "#757575"),
                    child: ElevatedButton(
                        style: ThemeHelper().buttonStyle(),
                        child: Padding(
                            padding: const EdgeInsets.fromLTRB(40, 10, 40, 10),
                            child: Text(
                                "戻る",
                                style: TextStyle(
                                    fontSize: 20,
                                    fontWeight: FontWeight.bold,
                                    color: Colors.white,
                                ),
                            ),
                        ),
                    ),
                ),
            ],
        ),
    ],
),
```

```
        ),
        ),
        ),
        onPressed: () {
          Get.toNamed('/select-signin');
        },
        ),
        ],
        ],
        ],
        ],
        ],
        ],
        ],
        );
      },
    }
}
```

10.7

認証コントローラにメソッドの実装

上記の修正で必要になりましたメソッドを、認証コントローラに、

- メール/パスワードでのサインイン・メソッド
- サインアウト・メソッド
- パスワードリセット・メソッド

実装します。

♣ 10.7.1 メール/パスワードサインイン・メソッド

認証コントローラに、メール/パスワードでサインインするメソッドを追加します。

▼ サインイン・メソッド

```
/// Email, passwordでサインイン
Future<void> signIn() async {
    try {
        await _auth.signInWithEmailAndPassword(
            email: emailController.text, password: passwordController.text);
        firebaseUser.value = _auth.currentUser;
    } on FirebaseAuthException catch (e) {
        if (e.code == 'firebase_auth/user-not-found') {
            Utility.customSnackBar('エラー', 'このEmailではユーザ登録されていません。');
        } else {
            Utility.customSnackBar('エラー', e.message ?? 'サインイン時に特定できないエラーが発生しました。');
        }
    } catch (e) {
        Utility.customSnackBar('エラー', e.toString());
    }
}
```

♣ 10.7.2 サインアウト・メソッド

認証コントローラに、サインインアウトするメソッドを追加します。

▼ サインアウト・メソッド

```
/// サインアウト
void singOut() async {
    await _auth.signOut();
    firebaseUser.value = _auth.currentUser;
}
```

♣ 10.7.3 パスワードリセット・メソッド

認証コントローラに、パスワードリセットメソッドを追加します。

▼ パスワードリセット・メソッド

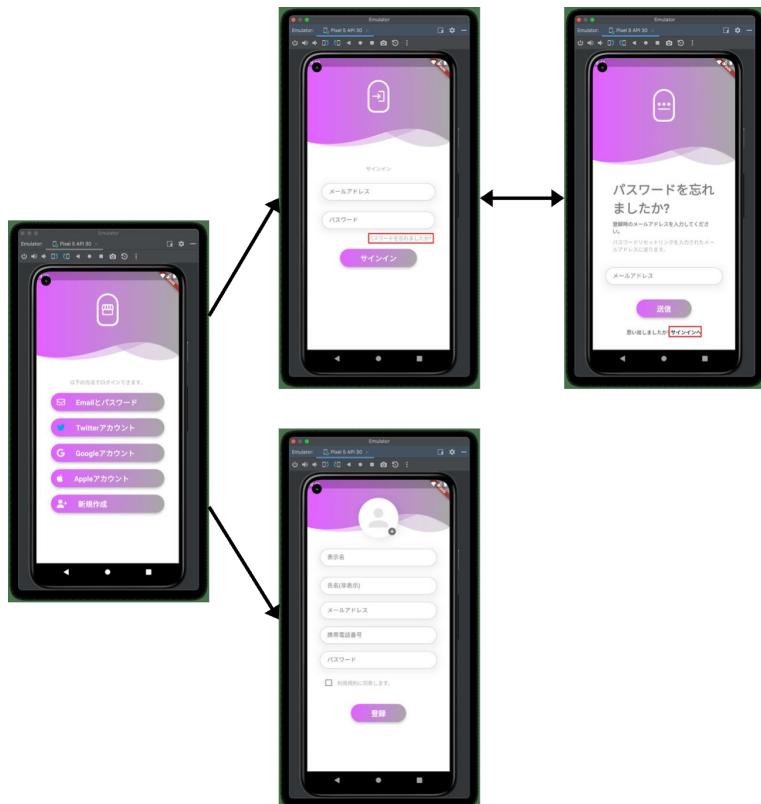
```
///パスワードリセット
Future<void> passwordReset() async {
    try {
        _auth.sendPasswordResetEmail(email: emailController.text);
        Get.offAll(const SelectSignInPage());
    } catch (error) {
        setAuthErrorMessage('パスワードリセットでエラーが発生しました。');
    }
}
```

10.8 動作確認

実装が完了しましたので、動作確認を行います。

♣ 10.8.1 Route

リンクが正常に機能しているか？



▲図 10.5: ナビゲーションテスト

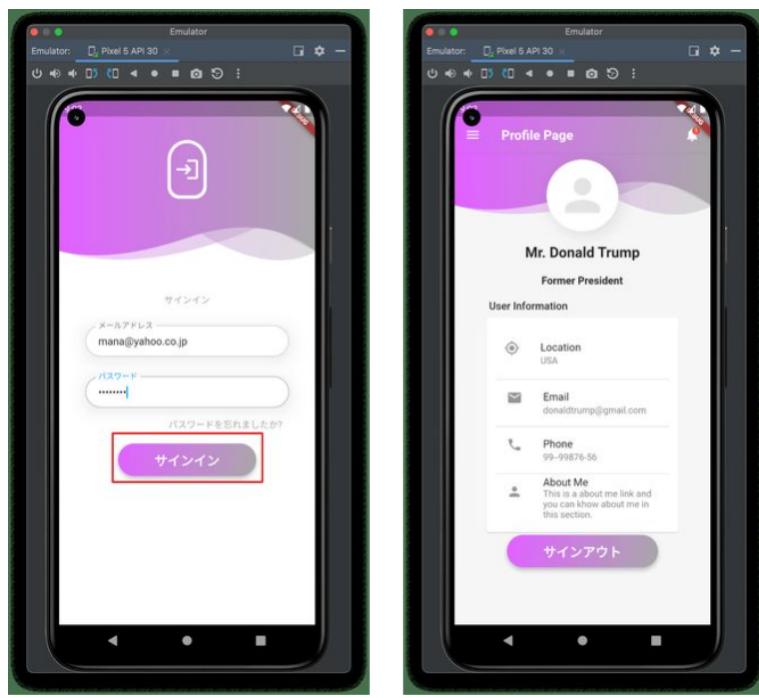
♣ 10.8.2 メール/パスワードでのサインイン

Firebase に登録されていないメールアドレスでエラーは表示されるか？



▲図 10.6: 登録メールアドレスなしテスト

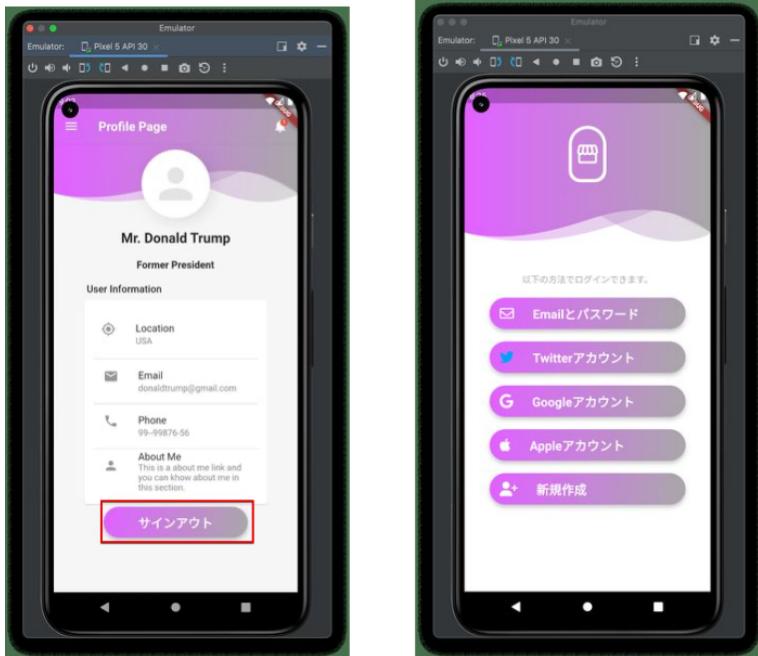
正常にログインできるか？



▲図10.7: ログインテスト

♣ 10.8.3 サインアウト

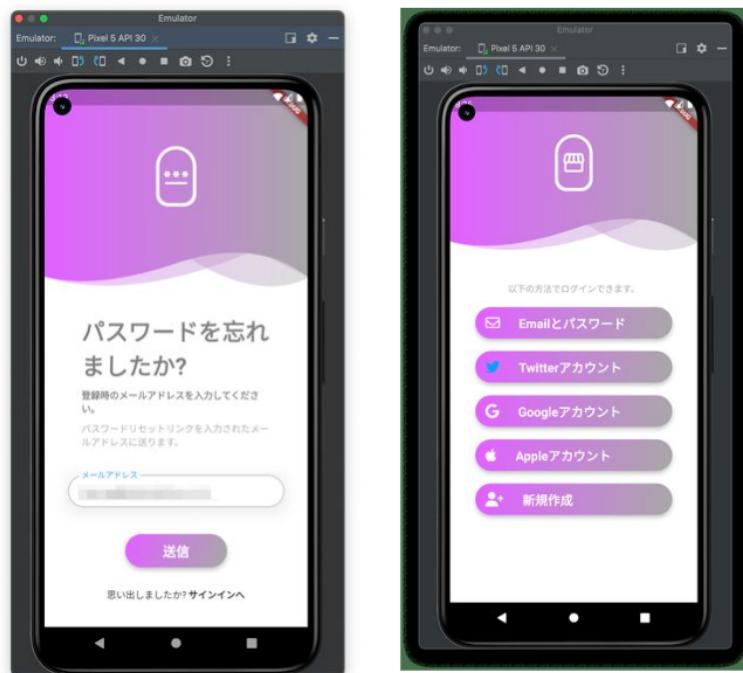
「サインアウト」ボタンで、サインアウトし「ログイン選択画面」へ戻るか？



▲図 10.8: サインアウトテスト

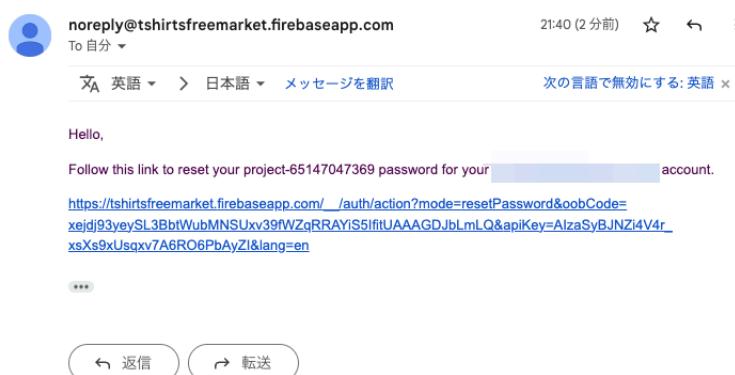
♣ 10.8.4 パスワード・リセット

パスワード・リセットの動作確認です。問題なく動作しています。



▲図10.9: パスワード・リセット

メールも届きます。



▲図10.10: パスワード・リセット用メール

.....
ここまでのソースコード

▼ GitHub

```
> git clone -b 06_email_signin https://github.com/risingforce9zz/tfreemarket.git
```

第 11 章

Twitter アカウントでサインイン

Twitter のアカウントを使用して Firebase にサインインします。

Twiiter の Developer アカウントが必要です。

【この章の内容】

11.1	Twitter アカウントで Firebase 認証	116
11.2	Twitter Developer サイトでキー作成	117
11.3	Firebase 認証で Twitter API キーの登録	125
11.4	アプリケーションへメソッド追加	128
11.5	認証コントローラへメソッドの追加	130
11.6	動作確認	136

11.1 Twitter アカウントで Firebase 認証

Twitter のアカウントを Firebase 認証に使用します。

準備さえできれば簡単に実装できます。ユーザーのメリットは、アプリケーション提供側へパスワードの登録がなく、アプリケーション提供側にセキュリティ上の問題が発生してもパスワード漏洩が起こらないことです。

手順は、

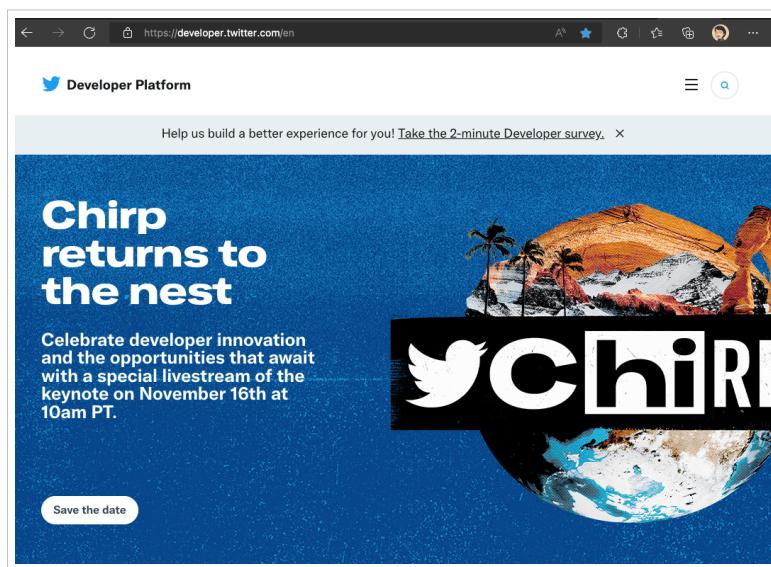
1. Twitter Developer サイトでキーを取得
2. Firebase 認証へ取得したキーを登録
3. アプリケーションでメソッドの追加

と、なります。

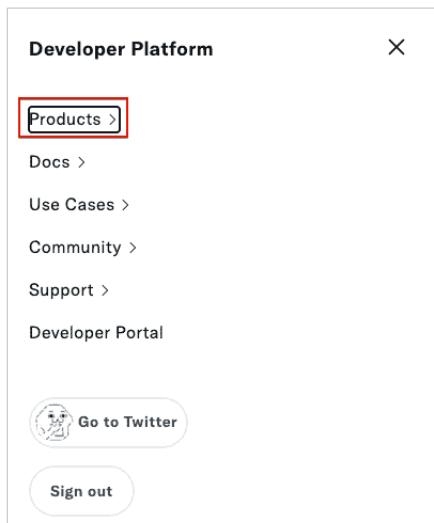
11.2 Twitter Developer サイトでキー作成

最初に、Twitter Developer サイト^{*1}にアクセスしキーを取得します。

Twitter Devloper サイトにログインします。

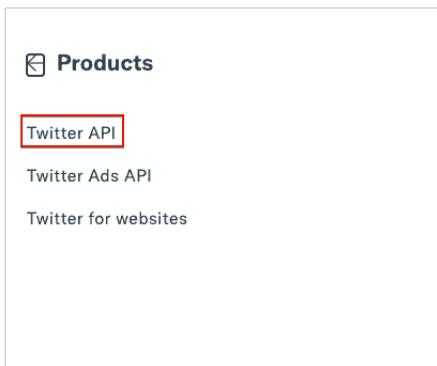


右上のメニューから「Product」を選択します。

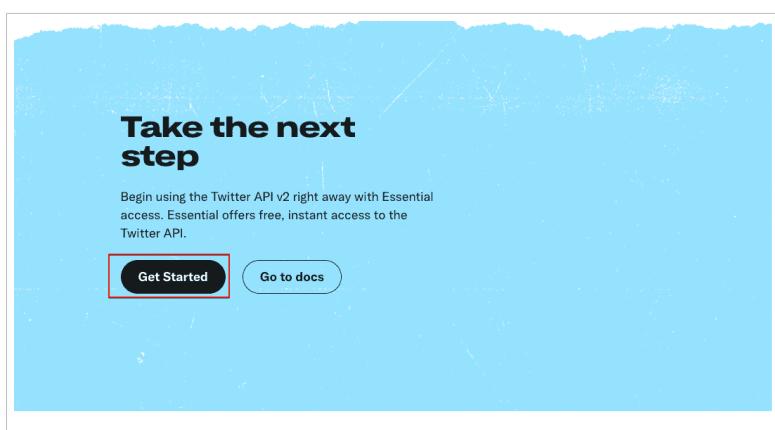


*1 <https://developer.twitter.com/>

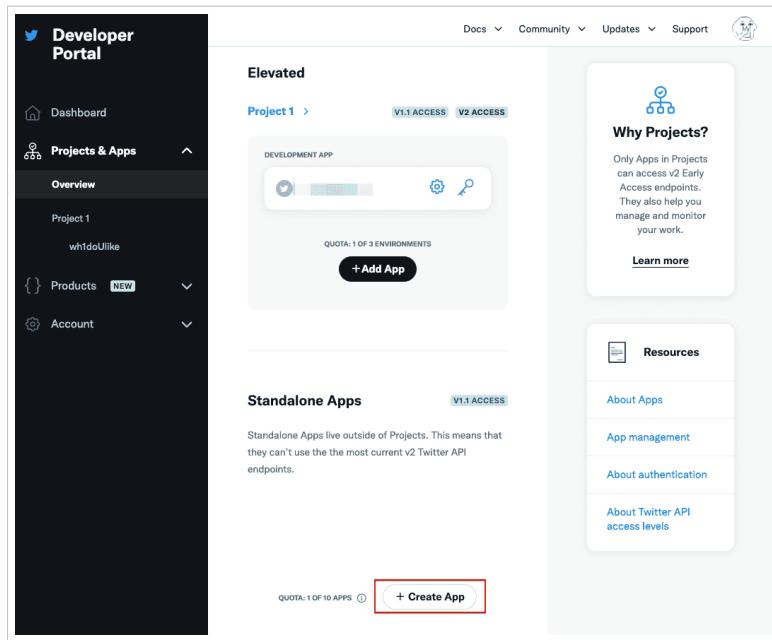
続いて、「Twitter API」を選択します。



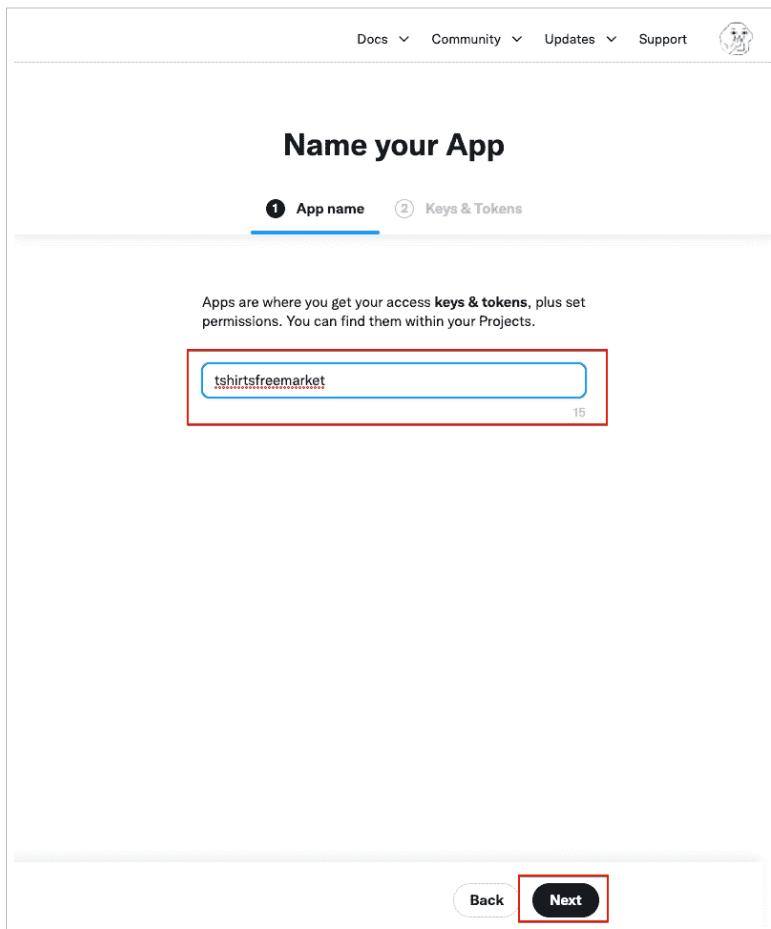
下へスクロールすると、「Get Started」ボタンがあります。クリックすると管理ページにアクセスできます。



「Create App」をクリックします。



アプリケーション名を入力し、「Next」ボタンをクリックします。



各種キーが表示されますのでコピーし、テキストファイルで保存します。アプリ登録が完了すれば、何度でも再作成ができます。

The screenshot shows the Twitter Developer Portal interface. On the left, there's a dark sidebar with navigation links like Dashboard, Projects & Apps (which is expanded to show Overview, Project 1, and STANDALONE APPS), Products, and Account. The main content area has a title "Here are your keys & tokens". It displays three sets of credentials with "Copy" buttons: API Key (value: 2wpuwUo, copied), API Key Secret (value: QihEbx..., copied), and Bearer Token (value: Au..., copied). Below these, a section titled "Setup your App" provides instructions on enabling 3rd party authentication and getting user tokens. At the bottom, there are "Go to dashboard" and "App settings" buttons, with "App settings" being highlighted with a red box.

アプリの登録が完了しましたら、「User authentication settings」の「Setup」をクリックします。

The screenshot shows the "tshirtsfreemarket" app settings page. The sidebar on the left is identical to the previous screenshot. The main content area shows "App details" with fields for NAME (tshirtsfreemarket), APP ID (25), and DESCRIPTION (This information will be visible to people who've authorized your App). To the right, there's a sidebar with "Authentication docs" and "Authentication methods". Under "User authentication settings", it says "User authentication not set up" and provides a description of what authentication allows. A "Set up" button is highlighted with a red box.

ここで、

- アプリケーションのタイプ
- コールバック URL
- アプリケーションの Web サイトの URL

を入力し、「Save」ボタンをクリックします。「新しいプロバイダを追加」ボタンをクリックします。

コールバック URL は、「適当な英数字://」でかまいません。これは、Firebase コンソールで指定する URL がのちほどインストールするプラグインで使えないためです。

User authentication settings

You can change these selections anytime.

App permissions (required)
These permissions enable OAuth 1.0a Authentication. ⓘ

- Read**
Read Tweets and profile information
- Read and write**
Read and Post Tweets and profile information
- Read and write and Direct message**
Read Tweets and profile information, read and post Direct messages
- Request email from users**
To request email from users, you are required to provide URLs to your App's privacy policy and terms of service.

Authentication mapping
Find out which v2 endpoints can be used with which authentication methods.

[View guide](#)

Type of App (required)
The type of App enables OAuth 2.0 Authentication. ⓘ

- Native App ⓘ**
Public client ⓘ
- Web App, Automated App or Bot ⓘ**
Confidential client ⓘ

! Move this App into a Project so that it can access v2 endpoints and OAuth 2.0. [Learn more](#)

App info

Callback URI / Redirect URL (required) ⓘ
yaruoi://

+ Add another URI / URL

Website URL (required)
<https://tshirtsfreemarket.imakita3gyo.com/>

Organization name (optional)
This name will be shown when users authorize your App.
[Input field]

Organization URL (optional)
This link will be shown when users authorize your App.
https://

Terms of service (optional)
A link to your terms of service will be shown when users authorize your App.
https://

Privacy policy (optional)
A link to your privacy policy will be shown when users authorize your App.
https://

[Cancel](#) **Save**

「Client ID」、「Client Secret」が表示されますので、これもコピーして保存します。「Done」で完了です。

The screenshot shows the Twitter Developer Portal with the URL <https://developer.twitter.com/portal/applications/12345678901234567890/client-credentials>. The page title is "Here is your OAuth 2.0 Client ID and Client Secret". It includes a note about security and instructions to save it securely. Below are two input fields: "Client ID" containing "SThWa" and "Client Secret" containing "3ukI". Each field has a "Copy" button to its right. To the right of the fields is a section titled "Authentication mapping" with a "View guide" button. At the bottom right is a "Done" button.

11.3

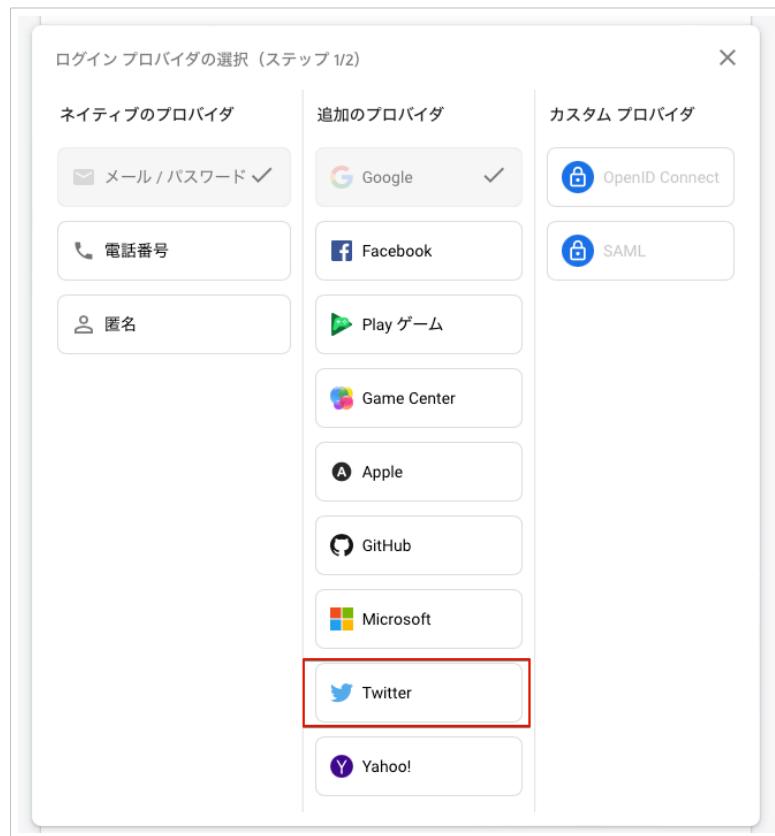
Firebase 認証で Twitter API キーの登録

続いて、Firebase コンソールの Authentication を開きます。

The screenshot shows the Firebase Authentication interface. On the left is a sidebar with project navigation. The main area is titled 'Authentication' under 'Sign-in method'. It lists two providers: 'メール / パスワード' (Email / Password) and 'Google', both marked as '有効' (Enabled). A red box highlights a blue button labeled '新しいプロバイダを追加' (Add new provider).

▲図 11.1: desc

表示されたダイアログから「Twitter」を選択します。



▲図 11.2: desc

表示されたダイアログのスイッチを有効にし、先ほど保存した「API キー」、「API シークレット」を入力し保存します。



▲図 11.3: desc

Authentication へ Twitter が追加されています。

The screenshot shows the 'Authentication' section in the Firebase console under the 'Sign-in method' tab. It lists the available sign-in providers: 'Email / Password', 'Google', and 'Twitter'. Each provider has a status indicator showing it is 'Enabled'. A blue button labeled 'Add new provider' is highlighted with a red box. The top navigation bar includes 'tshirtsfreemarket', 'ドキュメントに移動', and a user profile icon.

▲図 11.4: desc

11.4 アプリケーションへメソッド追加

アプリケーションに Twitter アカウントを使用しての Firebase サインインを実装します。

♣ 11.4.1 flutter_dotenv のインストール

Twitter アカウントでのサインインを実装するときには、Twitter Developper サイトで取得した、

- Twitter API Key
- Twitter API Secret
- コールバック URL

をコードに書き込みます。

しかし、コードに直接上記のキーを書き込むと危険です。ソースコードを GitHub などで管理している場合「公開」になっていますと間違なく悪用されますし、GitHub の場合には警告メールがきます。それを回避するためには、重要な情報は、「.env」ファイルを作成して別途保存し、デバッグ時・リリース時に読み込むようにします。

「.env」ファイルは、ローカル PC に保存し「.gitignore」へプッシュされないよう追加します。

Flutter で「.env」を読み込むために、「flutter_dotenv パッケージ」をインストールします。

The screenshot shows the flutter_dotenv package page on pub.dev. At the top, it displays the package name 'flutter_dotenv 5.0.2' with a copy icon. Below that, it says 'Published 13 months ago' and includes a 'Null safety' badge. It lists supported platforms: 'SDK', 'FLUTTER', 'PLATFORM', 'ANDROID', 'IOS', 'LINUX', 'MACOS', 'WEB', and 'WINDOWS'. To the right, there are 756 likes and a thumbs-up icon. A navigation bar below the platform list includes 'Readme', 'Changelog', 'Example', 'Installing', 'Versions', and 'Scores'. The 'Readme' tab is currently selected.

flutter_dotenv

pub v5.0.2

Load configuration at runtime from a `.env` file which can be used throughout the application.

The [twelve-factor app](#) stores **config** in *environment variables* (often shortened to *env vars* or *env*). Env vars are easy to change between deploys without changing any code... they are a language- and OS-agnostic standard.

▼ flutter_dotenv のインストール

```
> flutter pub add flutter_dotenv  
Resolving dependencies...  
+ flutter_dotenv 5.0.2  
  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!
```

「プロジェクトフォルダ/.env」ファイルを作成します。

▼ プロジェクトフォルダ/.env

```
TWITTER_API_KEY=取得したAPI KEY  
TWITTER_API_SECRET_KEY=取得したAPI Secret Key  
TWITTER_REDIRECT_URI=設定したコールバックURL
```

忘れないように、今、すぐ 「.gitignore」を開き 「*.env」を追加します。

次に、「.env」ファイルが読み込まれるように 「pubspec.yaml」ファイルにパスを書き込みます。

▼ pubspec.yaml

```
flutter:  
  
  # The following line ensures that the Material Icons font is  
  # included with your application, so that you can use the icons in  
  # the material Icons class.  
  uses-material-design: true  
  
  assets:  
    - .env ←ここです。  
  # To add assets to your application, add an assets section, like this:  
  # assets:  
  #   - images/a_dot_burr.jpeg  
  #   - images/a_dot_ham.jpeg
```

main メソッド（lib/main.dart）で初期化します。

▼ main.dart

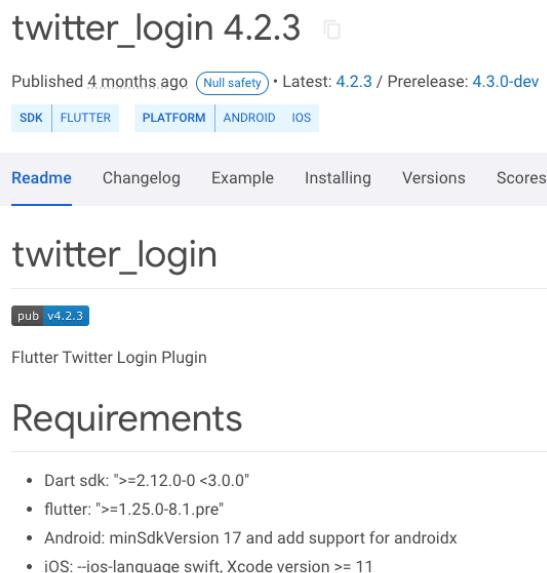
```
void main() async {  
  WidgetsFlutterBinding.ensureInitialized();  
  await Firebase.initializeApp(  
    options: DefaultFirebaseOptions.currentPlatform,  
  );  
  await dotenv.load(fileName: ".env");  
  runApp(LoginUiApp());  
}
```

11.5 認証コントローラヘメソッドの追加

♣ 11.5.1 twitter_login プラグインのインストール・設定

Firebase の Flutter ソーシャル^{*2}を確認します。

「twitter_login」プラグインが必要とありますので、インストールします。



The screenshot shows the official GitHub repository for the `twitter_login` plugin. It's version 4.2.3, published 4 months ago, with null safety support and latest version 4.2.3. The repository has 77 stars. Below the repository summary, there are tabs for `Readme`, `Changelog`, `Example`, `Installing`, `Versions`, and `Scores`. The `Readme` tab is selected. The `Readme` content includes the title "twitter_login", the `pub v4.2.3` badge, and the description "Flutter Twitter Login Plugin". The `Requirements` section lists dependencies: Dart sdk: ">=2.12.0-0 <3.0.0", flutter: ">=1.25.0-8.1.pre", Android: minSdkVersion 17 and add support for androidx, and iOS: -ios-language swift, Xcode version >= 11.

▼ twitter_login のインストール

```
☒ flutter pub add twitter_login
Resolving dependencies...
+ crypto 3.0.2
  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)
+ twitter_login 4.2.3
  vector_math 2.1.2 (2.1.3 available)
Changed 2 dependencies!
```

上記のプラグインサイトの指示にしたがって設定します。

*2 <https://firebase.google.com/docs/auth/flutter/federated-auth#twitter>

Android

プラグインサイトの README に従い、下記のコードを、

プロジェクトフォルダ/Android/app/src/main/AndroidManifest.xml
に追加します。

その際、下から 3 行目の「example」を Twitter へ登録したコールバック URL にします。「://」
は不要です。下から 2 行目の「gizmos」は削除して「host=""」にします。

▼ Android 向け

```
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    <!-- Accepts URIs that begin with "example://gizmos" -->
    <!-- Registered Callback URLs in TwitterApp -->
    <data android:scheme="example"
          android:host="gizmos" /> <!-- host is option -->
</intent-filter>
```

全体は、以下のようになります。

▼ AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.imalita3gyo.tfreemarket.tfreemarket">
    <application
        android:label="tfreemarket"
        android:name="${applicationName}"
        android:icon="@mipmap/ic_launcher">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:launchMode="singleTop"
            android:theme="@style/LaunchTheme"
            android:configChanges="orientation|keyboardHidden|keyboard|screenSize|
|smallestScreenSize|locale|layoutDirection|fontScale|screenLayout|density|uiMode"
            android:hardwareAccelerated="true"
            android:windowSoftInputMode="adjustResize">
            <!-- Specifies an Android theme to apply to this Activity as soon as
                the Android process has started. This theme is visible to the us>
        <er
            while the Flutter UI initializes. After that, this theme continu>
        <es
            to determine the Window background behind the Flutter UI. -->
        <meta-data
            android:name="io.flutter.embedding.android.NormalTheme"
            android:resource="@style/NormalTheme"
        />
```

```
<intent-filter>
    <action android:name="android.intent.action.MAIN"/>
    <category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    <!-- Accepts URIs that begin with "example://gizmos" -->
    <!-- Registered Callback URLs in TwitterApp -->
    <data android:scheme="manatee"
          android:host="" /> <!-- host is option -->
</intent-filter>
</activity>
<!-- Don't delete the meta-data below.
     This is used by the Flutter tool to generate GeneratedPluginRegistrations.java -->
<meta-data
    android:name="flutterEmbedding"
    android:value="2" />
</application>
</manifest>
```

iOS

iOS用は、
プロジェクトフォルダ/ios/Runner/info.plist
に以下のコードを追加します。

下から4行目の「<string>example</string>」の「example」部分を先ほどと同じくコールバックURLに書き換えます。

▼ iOS用設定

```
<key>CFBundleURLTypes</key>
<array>
<dict>
<key>CFBundleTypeRole</key>
<string>Editor</string>
<key>CFBundleURLName</key>
<string></string>
<key>CFBundleURLSchemes</key>
<array>
    <!-- Registered Callback URLs in TwitterApp -->
    <string>example</string>
</array>
</dict>
</array>
```

全体は、以下のようになります。

▼ iOS 設定ファイル info.plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/P>
>ropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>CFBundleDevelopmentRegion</key>
    <string>$(DEVELOPMENT_LANGUAGE)</string>
    <key>CFBundleDisplayName</key>
    <string>Tfreemarket</string>
    <key>CFBundleExecutable</key>
    <string>$(EXECUTABLE_NAME)</string>
    <key>CFBundleIdentifier</key>
    <string>$(PRODUCT_BUNDLE_IDENTIFIER)</string>
    <key>CFBundleInfoDictionaryVersion</key>
    <string>6.0</string>
    <key>CFBundleName</key>
    <string>tfreemarket</string>
    <key>CFBundlePackageType</key>
    <string>APPL</string>
    <key>CFBundleShortVersionString</key>
    <string>$(FLUTTER_BUILD_NAME)</string>
    <key>CFBundleSignature</key>
    <string>????</string>
    <key>CFBundleVersion</key>
    <string>$(FLUTTER_BUILD_NUMBER)</string>
    <key>LSRequiresiPhoneでのTwitterサインインOS</key>
    <true/>
    <key>UILaunchStoryboardName</key>
    <string>LaunchScreen</string>
    <key>UIMainStoryboardFile</key>
    <string>Main</string>
    <key>UISupportedInterfaceOrientations</key>
    <array>
        <string>UIInterfaceOrientationPortrait</string>
        <string>UIInterfaceOrientationLandscapeLeft</string>
        <string>UIInterfaceOrientationLandscapeRight</string>
    </array>
    <key>UISupportedInterfaceOrientations~ipad</key>
    <array>
        <string>UIInterfaceOrientationPortrait</string>
        <string>UIInterfaceOrientationPortraitUpsideDown</string>
        <string>UIInterfaceOrientationLandscapeLeft</string>
        <string>UIInterfaceOrientationLandscapeRight</string>
    </array>
    <key>UIViewControllerBasedStatusBarAppearance</key>
    <false/>
    <key>CADisableMinimumFrameDurationOnPhone</key>
    <true/>
    <key>UIApplicationSupportsIndirectInputEvents</key>
    <true/>
    <key>CFBundleURLTypes</key>
</array>
```

```
<dict>
    <key>CFBundleTypeRole</key>
    <string>Editor</string>
    <key>CFBundleURLName</key>
    <string></string>
    <key>CFBundleURLSchemes</key>
    <array>
        <!-- Registered Callback URLs in TwitterApp -->
        <string>manatee</string>
    </array>
</dict>
</array>
</dict>
</plist>
```

以上で準備完了です。

♣ 11.5.2 Twitter サインイン・メソッドの追加

認証コントローラ (auth_controller.dart) ヘメソッドを追加します。

▼ Twitter サインイン

```
/// Twitterログイン
signInWithTwitter() async {
    // TwitterLoginのインスタンス化
    final twitterLogin = TwitterLogin(
        apiKey: dotenv.env['TWITTER_API_KEY']!,
        apiSecretKey: dotenv.env['TWITTER_API_SECRET_KEY']!,
        redirectURI: dotenv.env['TWITTER_REDIRECT_URI']!);

    // Twitterサインイン フロー開始
    final authResult = await twitterLogin.login();
    String accessToken = authResult.authToken!;
    String accessTokenSecret = authResult.authTokenSecret!;
    String screenName = authResult.user!.screenName;
    String twitterThumbnailImage = authResult.user!.thumbnailImage;
    String name = authResult.user!.name;

    // アクセストークンからcredentialを作成
    final twitterAuthCredential = TwitterAuthProvider.credential(
        accessToken: accessToken, secret: accessTokenSecret);

    // firebaseのcredentialを作成
    await _auth.signInWithCredential(twitterAuthCredential);
}
```

ログイン選択画面に、認証コントローラをバインドします。

▼ select_signin_page.dart

```
class SelectSignInPage extends StatelessWidget{
  SelectSignInPage({Key? key}): super(key:key);

  final AuthController authController = AuthController.to;

  final double _headerHeight = 250;
```

ボタンに認証コントローラのメソッド呼び出しを割り当てます。

▼ Twitter サインイン・メソッド

```
Container(
  decoration: ThemeHelper().buttonBoxDecoration(context),
  child: ElevatedButton(
    style: ThemeHelper().buttonStyle(),
    child: Row(
      children: const [
        FaIcon(FontAwesomeIcons.twitter, size: 23, color: Colors.blue),
        Padding(
          padding: EdgeInsets.fromLTRB(10, 0, 10, 0),
          child: Text('Twitterアカウント', style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold, color: Colors.white),),
        ),
      ],
    ),
    onPressed: (){
      authController.signInWithTwitter();
    },
  ),
),
```

11.6 動作確認

動作確認を行います。

♣ 11.6.1 iOS

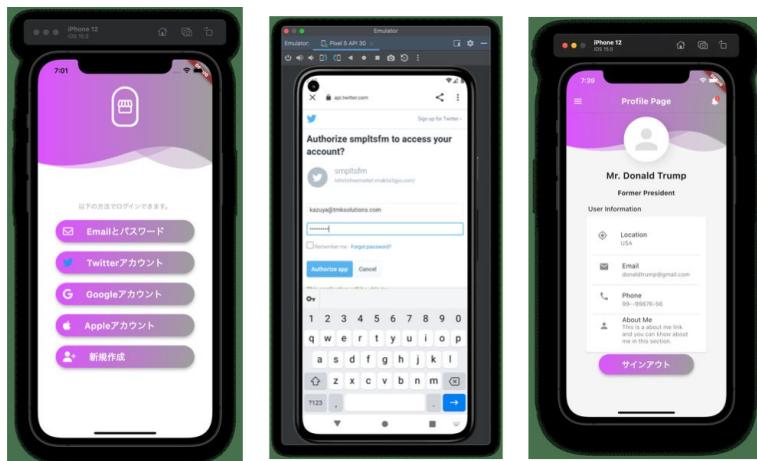
iPhone シミュレータでのサインインです。Twitter アカウントのメール/パスワードでアプリにサインインできました。



▲ 図 11.5: iPhone での Twitter サインイン

♣ 11.6.2 Android

Android エミュレータでのサインです。同じく Twitter アカウントのメール/パスワードでサインインできました。



▲ 図 11.6: Android で Twitter サインイン

♣ 11.6.3 Firebase コンソール

Twitter アカウントがひとつしかないので、iPhone でサイン後アカウント削除し、Android でサインインしています。

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'. The main area displays a table of user information:

ID	プロバイダー	作成日	ログイン日	ユーザー UID
[Redacted]	Twitter	202...	202...	iK1ck3iOE90wMZ...
[Redacted]	Email	202...	202...	8H8kEh9LAIve6oG...
mana@yahoo....	Email	202...	202...	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 - 3 of 3'.

▲図 11.7: Firebase コンソール

ここまで

までのソースコード

▼ GitHub

```
> git clone -b 07_twitter_signin https://github.com/risingforce9zz/tfrema>rket.git
```

第 12 章

Google サインイン

Google アカウントを使用してアプリケーションにサインインします。

【この章の内容】

12.1	Google アカウントで Firebase 認証	139
12.2	Firebase にキーを登録	140
12.3	google_sign_in プラグインのインストール	145
12.4	iOS 用の設定	146
12.5	アプリに google サインイン・メソッドの実装	147
12.6	動作確認	148

12.1

Google アカウントで Firebase 認証

Twitter のアカウントを Firebase 認証に使用します。

Firebase、Flutter ともに Google が開発しているため親和性は抜群です。まずは、[公式サイト^{*1}](#)を確認します。

グーグル

Firebase で Google サインインを使用する場合、ほとんどの構成は既にセットアップされていますが、マシンの SHA1 キーが Android で使用できるように構成されていることを確認する必要があります。キーの生成方法については、[インストールドキュメント](#)を参照してください。

[Firebase コンソール](#)で「Google」サインイン プロバイダが有効になっていることを確認します。

ユーザーが Google でサインインした場合、手動でアカウントを登録した後、信頼できるプロバイダーの Firebase Authentication の概念により、認証プロバイダーは自動的に Google に変更されます。詳細については、[こちら](#)をご覧ください。

iOS+ と Android ウェブ

ネイティブ プラットフォームでは、認証フローをトリガーするためにサード パーティ のライブラリが必要です。

公式の [google_sign_in](#) プラグインをインストールします。

インストールしたら、サインイン フローをトリガーし、新しい資格情報を作成します。

手順は、

1. Firebase に登録したアプリに SHA1 フィンガープリントを登録
2. google_sign_in プラグインのインストール
3. iOS 用の設定
4. アプリケーションに google サインイン・メソッドの実装

と、なります。

^{*1} <https://firebase.google.com/docs/auth/flutter/federated-auth#google>

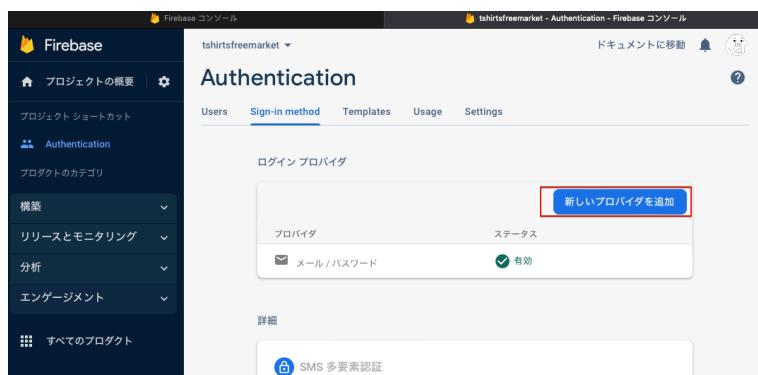
12.2 Firebase にキーを登録

ターミナルに、以下のコマンドを入力してキーを取得します。

▼ SHA1 キーの取得

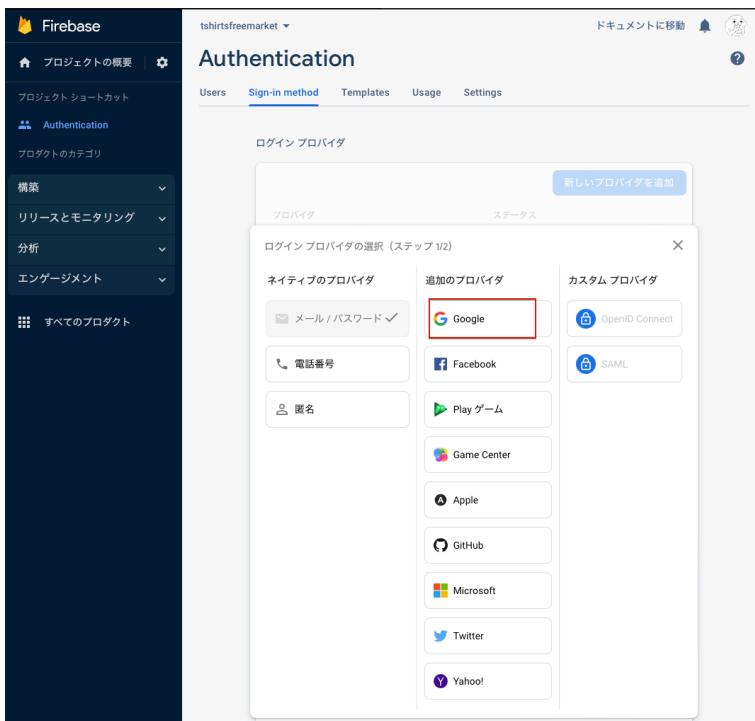
♣ 12.2.1 表示された SHA1 フィンガープリントを Firebase に登録

Firebase コンソールを開き、Authentication を選択します。「Sign-in method」タブを選択し「新しいプロバイダを追加」ボタンをクリックします。



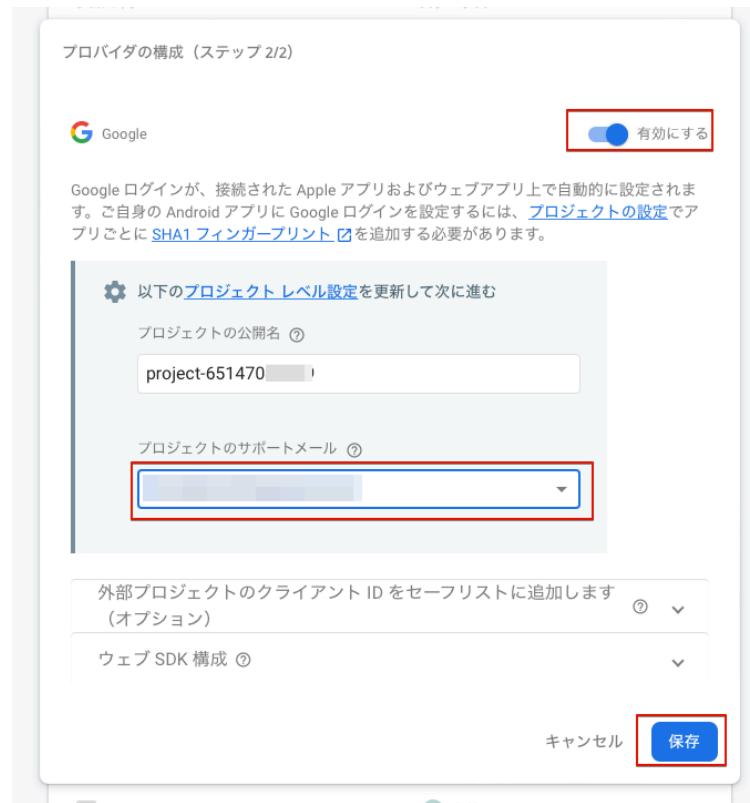
▲図 12.1: ログインプロバイダの登録

表示されたダイアログから、「Google」を選択します。



▲図 12.2: Google を追加

表示されたダイアログのスイッチを有効化します。プロジェクトのサポートメールを選択し、「保存」ボタンをクリックします。



▲図 12.3: 有効化

以上で、Google が追加されます。

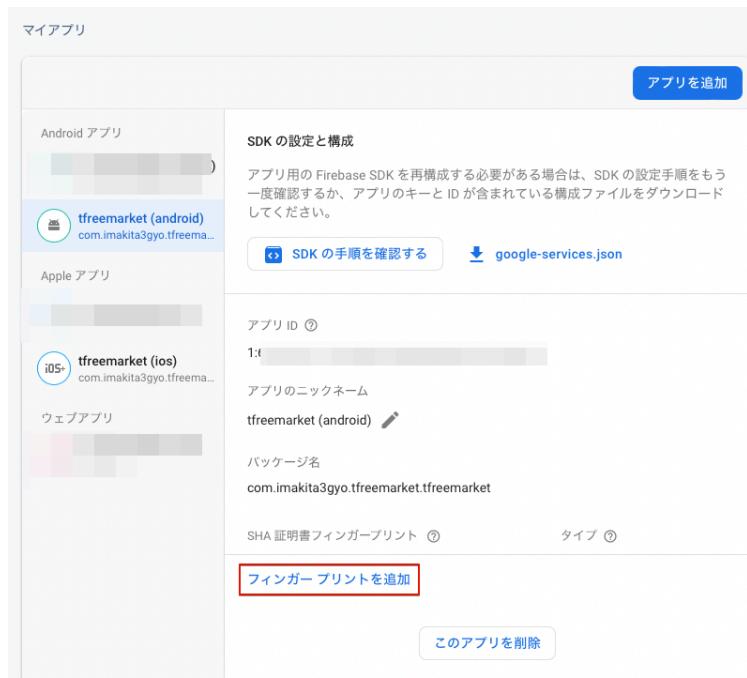
▲図 12.4: ログインプロバイダの登録

Firebase コンソールの「プロジェクトの概要」横の歯車をクリックし、「プロジェクトの設定」を選択します。



▲図 12.5: SHA1 の登録

「flutterfire configure」コマンドを使ったときにアプリケーションが登録されていますので、Android のアプリを選択します。「フィンガープリントを追加」をクリックします。



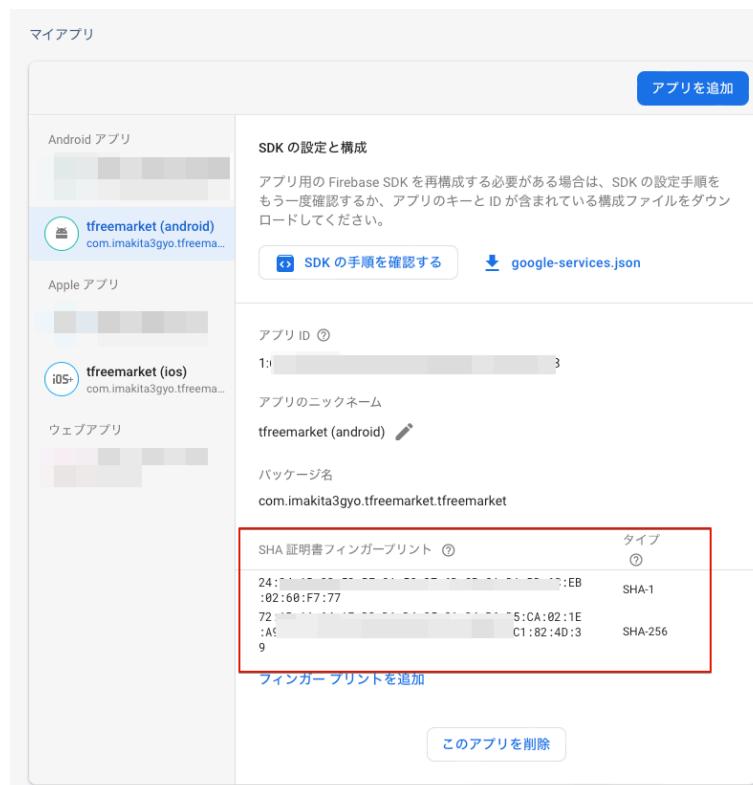
▲図 12.6: SHA1 の登録

フィンガープリントの登録ダイアログが表示されますので、先ほど出力した SHA1 をコピペします。ついでに SHA256 も登録します。



▲図 12.7: SHA1 の登録

「保存」ボタンをクリックすると登録されます。

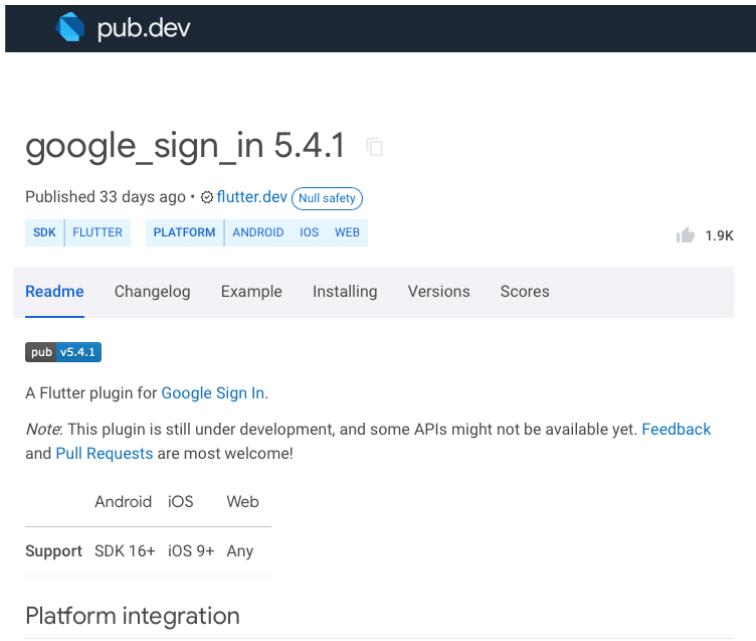


▲図 12.8: SHA1 の登録

以上で Firebase 側の設定は完了です。

12.3 google_sign_in プラグインのインストール

google_sign_in プラグイン^{*2}をインストールします。



The screenshot shows the pub.dev page for the google_sign_in package. At the top, it says "google_sign_in 5.4.1". Below that, it says "Published 33 days ago · ⚡ flutter.dev (Null safety)". It has tabs for "SDK", "FLUTTER", "PLATFORM", "ANDROID", "IOS", and "WEB". There are 1.9K likes. Below the tabs, there's a "Readme" tab which is underlined, and other tabs like "Changelog", "Example", "Installing", "Versions", and "Scores". A "pub v5.4.1" button is visible. The description says "A Flutter plugin for Google Sign In." and notes that it is still under development. It supports "Android", "iOS", and "Web". It requires "Support", "SDK 16+", "iOS 9+", and "Any".

▲図 12.9: google_sign_in プラグイン

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

```
> flutter pub add google_sign_in
Resolving dependencies...
+ google_sign_in 5.4.1
+ google_sign_in_android 6.1.0
+ google_sign_in_ios 5.5.0
+ google_sign_in_platform_interface 2.3.0
+ google_sign_in_web 0.10.2
  material_color_utilities 0.1.5 (0.2.0 available)
+ quiver 3.1.0
  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 6 dependencies!
```

以上でプラグインのインストールは完了です。

^{*2} https://pub.dev/packages/google_sign_in

12.4 iOS用の設定

iOS用のに以下の設定をします。

「プロジェクトフォルダ/ios/Runner/Info.plist」を開き以下を追加します。
「XXXXXXXXXX」の部分は、「プロジェクトフォルダ/ios/Runner/GoogleService-Info.plist」
ファイル内の「REVERSED_CLIENT_ID」に書き換えます。

▼追加するコード

```
<!-- Put me in the [my_project]/ios/Runner/Info.plist file -->
<!-- Google Sign-in Section -->
<key>CFBundleURLTypes</key>
<array>
    <dict>
        <key>CFBundleTypeRole</key>
        <string>Editor</string>
        <key>CFBundleURLSchemes</key>
        <array>
            <!-- TODO Replace this value: -->
            <!-- Copied from GoogleService-Info.plist key REVERSED_CL>
<KEY_ID -->
            <string>XXXXXXXXXX</string>
        </array>
    </dict>
</array>
<!-- End of the Google Sign-in Section -->
```

以上で、iOS用の設定は完了です。

12.5 アプリに google サインイン・メソッドの実装

アプリケーションの認証コントローラに以下のコードを追加します。

▼google サインイン・メソッドの追加

```
/// Googleサインイン
signInWithGoogle() async {
    // 認証フローの開始
    final GoogleSignInAccount? googleUser = await GoogleSignIn(
        clientId: DefaultFirebaseOptions.currentPlatform.iosClientId)
        .signIn();

    // 認証情報詳細を取得
    final GoogleSignInAuthentication? googleAuth =
        await googleUser?.authentication;

    // クレデンシャルの作成
    final credential = GoogleAuthProvider.credential(
        accessToken: googleAuth?.accessToken,
        idToken: googleAuth?.idToken,
    );

    // サインインに成功するとユーザークレデンシャルが返る
    UserCredential userCredential =
        await FirebaseAuth.instance.signInWithCredential(credential);
}
```

ログイン選択画面の「Google アカウント」ボタンにメソッド呼び出しを割り当てます。

▼メソッド呼び出し

```
Container(
    decoration: ThemeHelper().buttonBoxDecoration(context),
    child: ElevatedButton(
        style: ThemeHelper().buttonStyle(),
        child: Row(
            children: const [
                FaIcon(FontAwesomeIcons.google, size: 23),
                Padding(
                    padding: EdgeInsets.fromLTRB(10, 10, 0, 10),
                    child: Text('Googleアカウント', style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold, color: Colors.white),),
                ),
            ],
        ),
        onPressed: (){
            authController.signInWithGoogle();
        },
    ),
),
```

以上で、Google アカウントでアプリケーションへのログインができます。

12.6 動作確認

♣ 12.6.1 iPhone

iPhone シミュレータでデバッグしようとするとエラーになりました。

「プラットフォームが指定されていないので iOS 11 を割り当てた。」
「CocoaPods のリポジトリが期限切れ」

とのことです。

▼ ビルドエラー

Error output from CocoaPods:

```
[!] Automatically assigning platform `iOS` with version `11.0` on target `Runner` because no platform was specified. Please specify a platform for this target in your Podfile. See `https://guides.cocoapods.org/syntax/podfile.html#platform`.
```

Error: CocoaPods's specs repository is too out-of-date to satisfy dependencies.

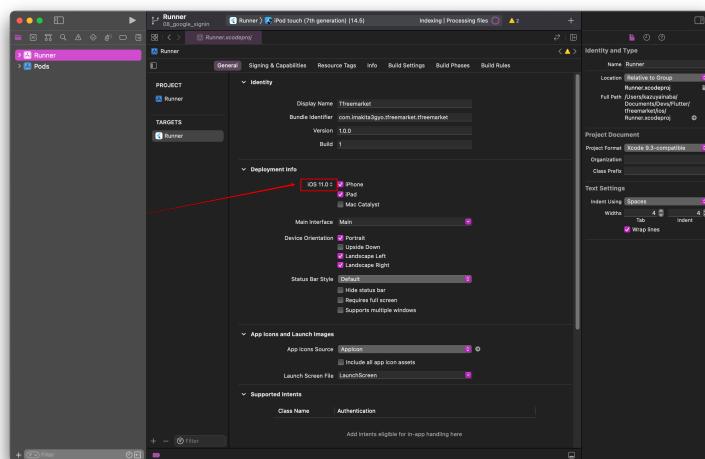
To update the CocoaPods specs, run:

```
pod repo update
```

Error running pod install

Error launching application on iPhone 12.

Xcodeを開き、iOSの対象を指定します。iOS 16 がリリースされるので、2世代前の iOS 14 の最新版を仕様とします。



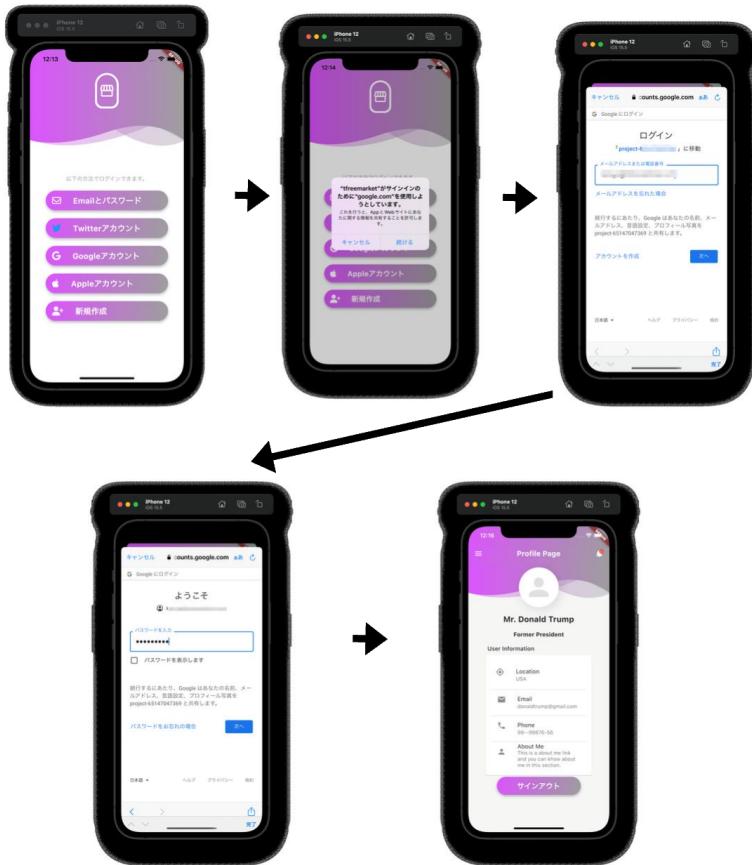
▲図 12.10: Xcode で iOS のバージョン指定

ビルドエラーが発生しましたので、環境を再構築します。

▼ desc

```
> flutter clean                                     4.7s
Cleaning Xcode workspace...                         2,860ms
Cleaning Xcode workspace...                         548ms
Deleting build...                                  3ms
Deleting .dart_tool...                            0ms
Deleting Generated.xcconfig...                   0ms
Deleting flutter_export_environment.sh...        0ms
Deleting Flutter.podspec...                      0ms
Deleting ephemeral...                           0ms
Deleting .flutter-plugins-dependencies...       0ms
Deleting .flutter-plugins...                     0ms
> rm -Rf ios/Pods
> rm -Rf ios/.symlinks
> rm ios/Podfile
> rm ios/Podfile.lock
> cd ios
> pod install
```

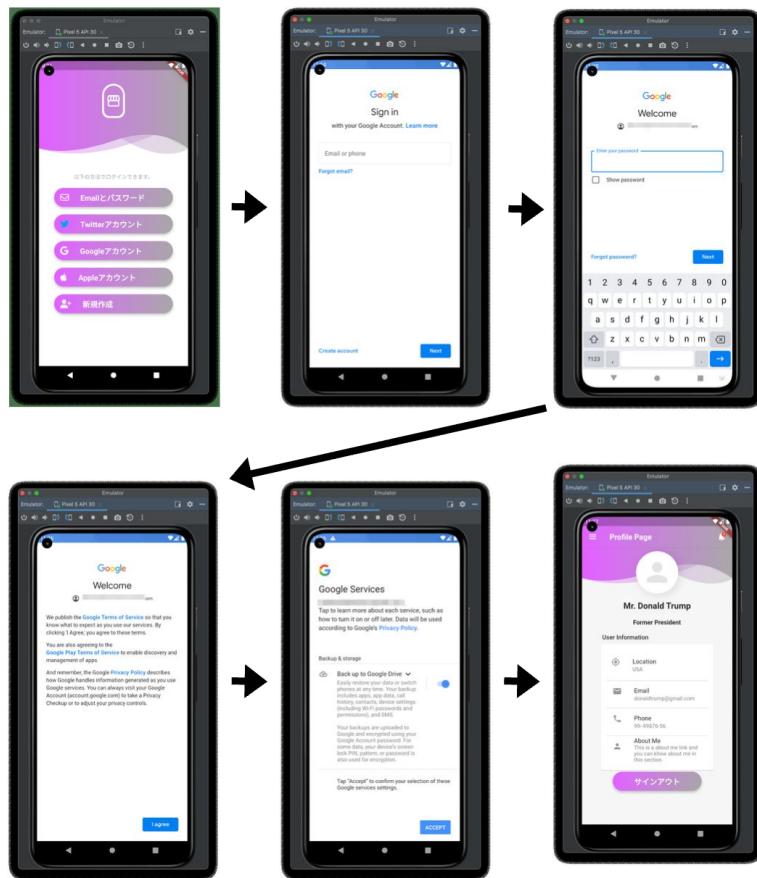
今回は無事にビルドできました。



▲図 12.11: iPhone の動作確認

♣ 12.6.2 Android

さすがに Google がすべて関与しているものだと問題なく動作します。



▲ 図 12.12: Android での動作確認

ここまでまでのソースコード

▼ GitHub

```
> git clone -b 08_google_signin https://github.com/risingforce9zz/tfreemar>
> ket.git
```

第 13 章

AppleID でサインイン

AppleID を使ってアプリケーションにサインインします。

【この章の内容】

13.1	Firebase コンソールでログインプロバイダを追加	153
13.2	Apple Developer で「Apple サインイン」を有効に	155
13.3	アプリケーションに Apple サインイン・メソッドの実装	173
13.4	動作確認	175

13.1

Firebaseコンソールでログインプロバイダを追加

Firebaseコンソールを開きます。Authenticationを選択し、「新しいプロバイダを追加」ボタンをクリックします。

The screenshot shows the Firebase Authentication interface. At the top, there's a navigation bar with tabs: 'Users', 'Sign-in method' (which is underlined, indicating it's selected), 'Templates', 'Usage', and 'Settings'. Below the tabs, the page title is 'Authentication'. Underneath the title, there's a section titled 'ログイン プロバイダ' (Login providers). This section contains a table with three rows:

プロバイダ	ステータス
✉️ メール / パスワード	✓ 有効
Google	✓ 有効

At the bottom right of this table area, there's a blue pencil icon. Above the table, a red box highlights the blue button labeled '新しいプロバイダを追加' (Add new provider).

ダイアログが表示されます。追加のプロバイダから「Apple」を選択します。

The screenshot shows a modal dialog box titled 'ログイン プロバイダの選択 (ステップ 1/2)' (Select login provider (Step 1/2)). The dialog has three columns: 'ネイティブのプロバイダ' (Native providers), '追加のプロバイダ' (Add provider), and 'カスタム プロバイダ' (Custom providers). In the '追加のプロバイダ' column, a red box highlights the 'Apple' provider, which is represented by a small Apple logo icon.

表示されたダイアログで、Appleを有効にします。サービスIDなどは後で登録します。ここでは「保存」ボタンをクリックします。



ログインプロバイダにAppleが追加されているのを確認します。

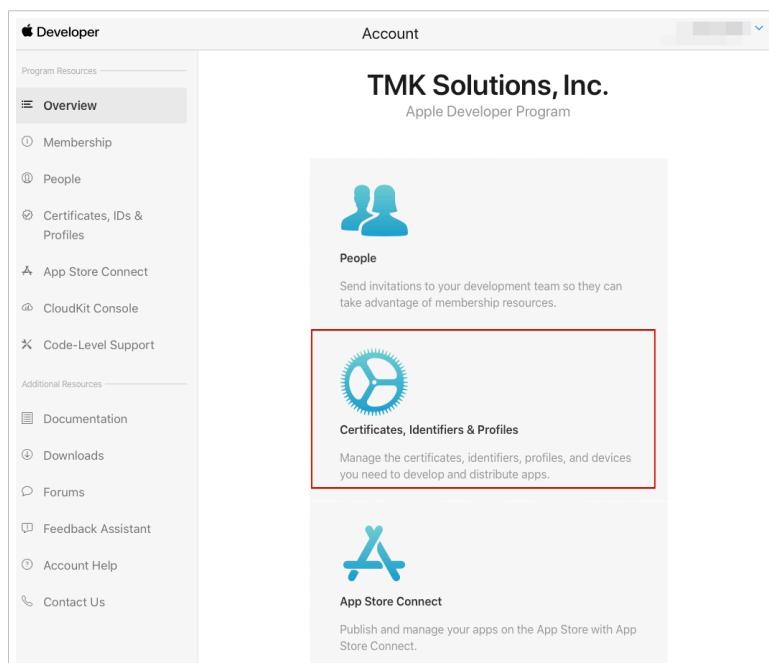


13.2 Apple Developerで「Appleサインイン」を有効に

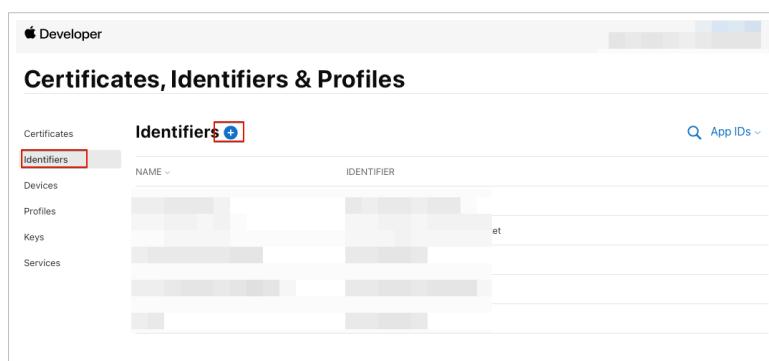
♣ 13.2.1 アプリケーションのIDを取得します。

アプリケーションのIDを取得します。Firebaseプロジェクト作成時に「Organization」欄に入力したものがアプリケーションIDです。すでに登録されている場合は、アプリケーションIDをクリックします。

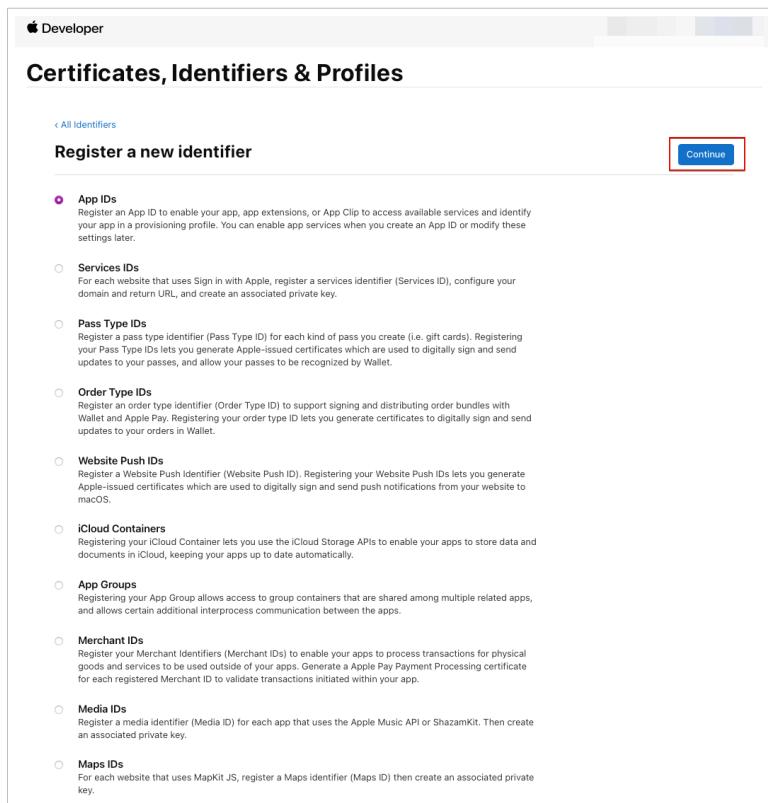
Apple Developerサイトにログインします。「Certificates, Identifiers & Profiles」をクリックします。



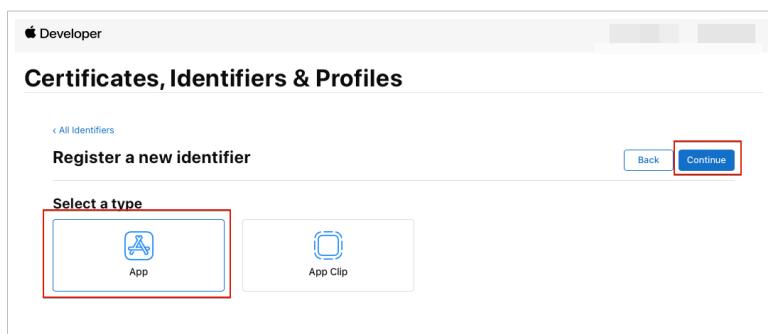
左サイドメニューの「Identifiers」を選択し、「+」をクリックします。



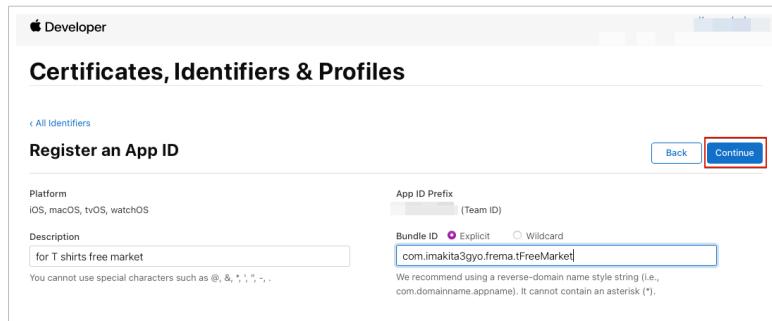
「App IDs」を選択し、「Continue」をクリックします。



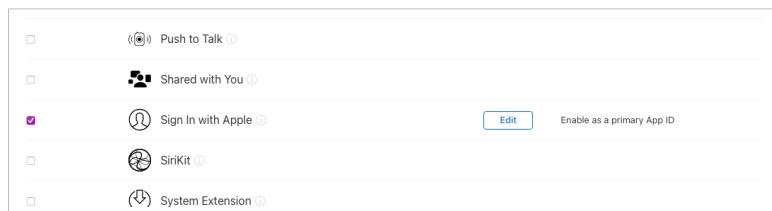
「App」を選択し「Continue」をクリックします。



「Description」欄に備考を入力し、「Bundle ID」欄にアプリケーションIDを入力します。

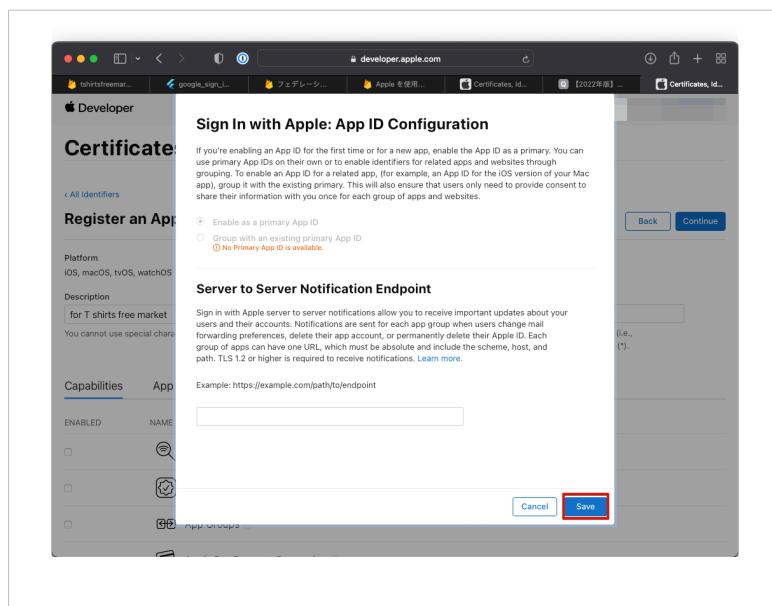


下へスクロールすると、「Sign in with Apple」がありますので、チェックをONにします。



「Edit」をクリックすると、「App ID Configuration」が開きますが、「Save」で閉じます。

上にスクロールするし、「Continue」ボタンをクリックします。

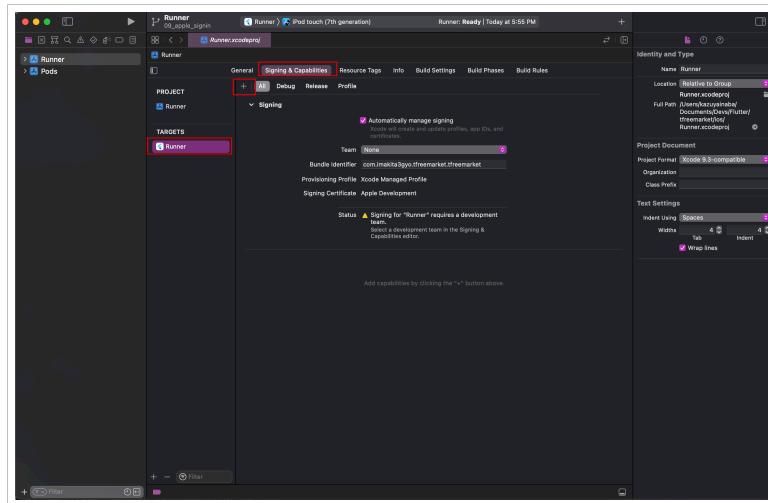


以上でアプリケーションIDの取得が完了しました。

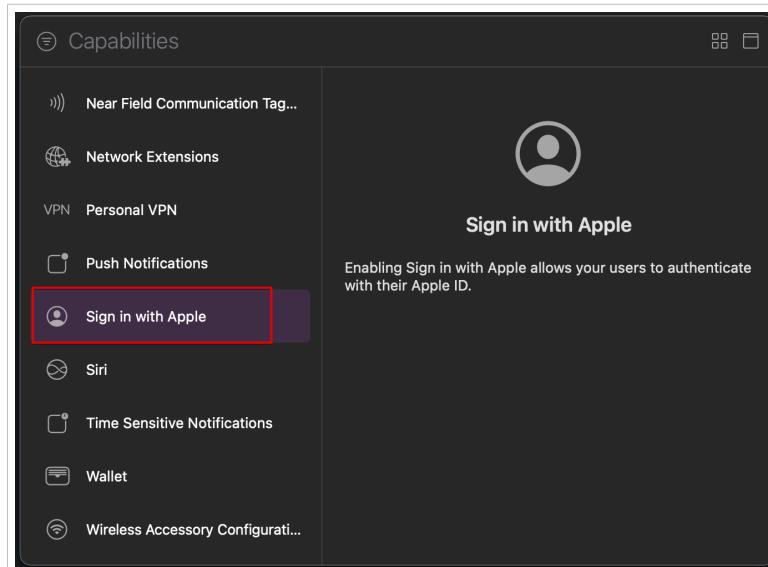
♣ 13.2.2 XcodeでAppleでサインインを有効にする

プロジェクトフォルダ内のiosフォルダをXcodeで開きます。

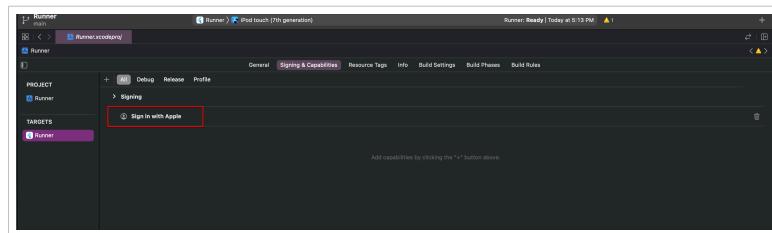
「Target」 - 「Signing & Capabilities」で「+」をクリックします。



「Sign in with Apple」をダブル・クリックします。



アプリケーションに「Sign in with Apple」が追加されました。

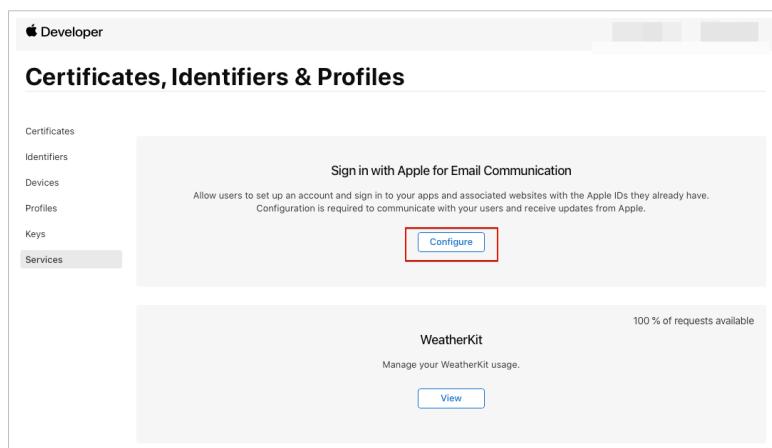


♣ 13.2.3 Appleの匿名E-mailサービスの設定

Apple IDでサインインするときに、「メールを非公開」を選択するとサービス提供者には、「anonymous@privaterelay.appleid.com (anonymous部分がランダムな文字列になる)」のアドレスが渡されます。

サービス提供者がユーザーにメールを送ると、AppleはAppleIDのメールアドレスへ転送してくれます。その場合、登録してある差出人からのメールのみ転送します。ここで、差出人からのメールアドレスを登録します。

再度、Apple Developerサイトへログインします。左サイドメニューからサービスを選択し、「Sign in with Apple for Email Communication」の「configure」ボタンをクリックします。



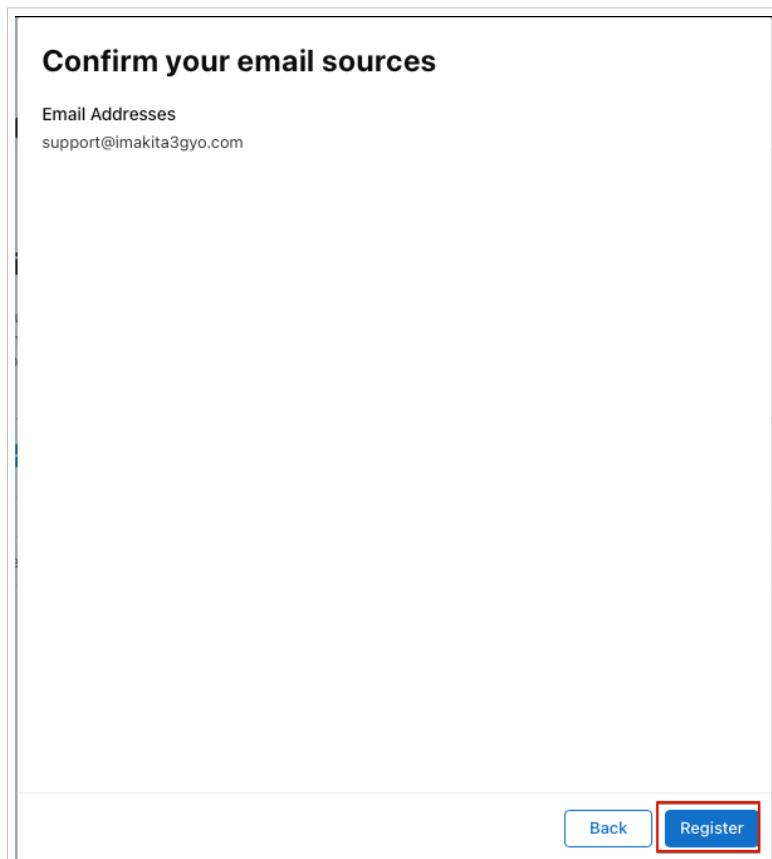
「Email Sources」の「+」をクリックします。

The screenshot shows the 'Certificates, Identifiers & Profiles' section of the Apple Developer portal. Under 'Email Sources', there is a red box around the '+' button. A tooltip 'Add a new email source' is visible above the button. Below the button, it says 'No result found.'

差出人のメールアドレスの「ドメイン」、「サブドメイン」、「メールアドレス」を登録し、「Next」ボタンをクリックします。

The dialog box has a title 'Register your email sources'. It contains two input fields: 'Domains and Subdomains' and 'Email Addresses'. Both fields have placeholder text: 'Enter a comma delimited list of email source domains and subdomains.' and 'Enter a comma delimited list of email addresses.'. At the bottom right are 'Cancel' and 'Next' buttons, with 'Next' being highlighted by a red box.

確認ダイアログが表示されますので、「Register」ボタンをクリックします。



登録完了のダイアログが表示されます。



差出人メールアドレスが登録されました。

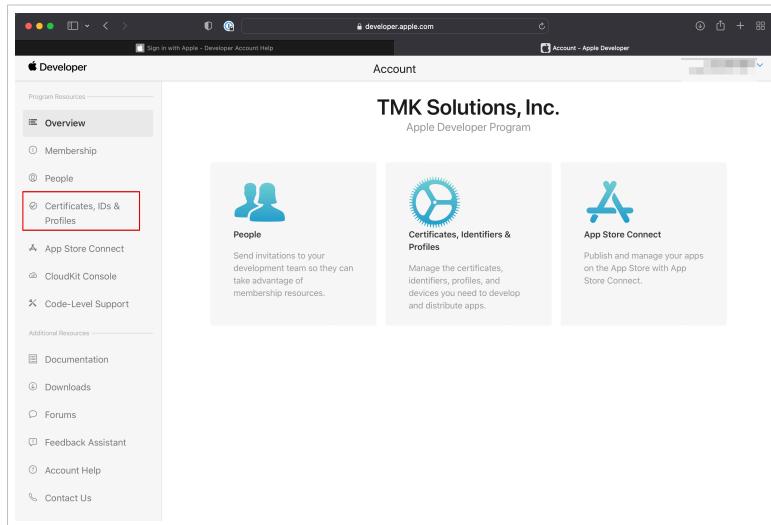
The screenshot shows the "Certificates, Identifiers & Profiles" section. Under "Email Sources", two entries are listed:

	TYPE	STATUS
support@makita3gyo.com	Email address	● SPF
noreply@tshirtsfreemarket.firebaseioapp.com	Email address	● SPF

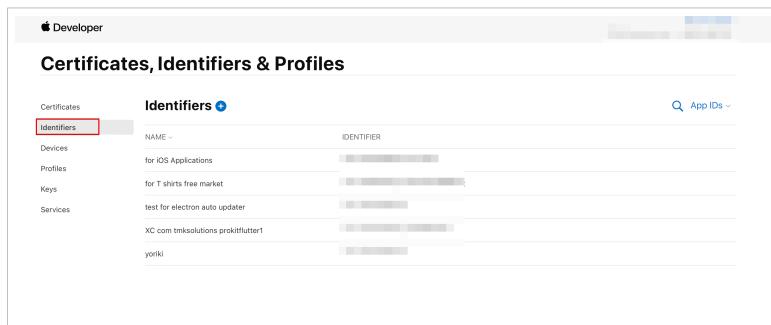
♣ 13.2.4 Android用のサービスID、キーIDを取得する

サービスIDの取得

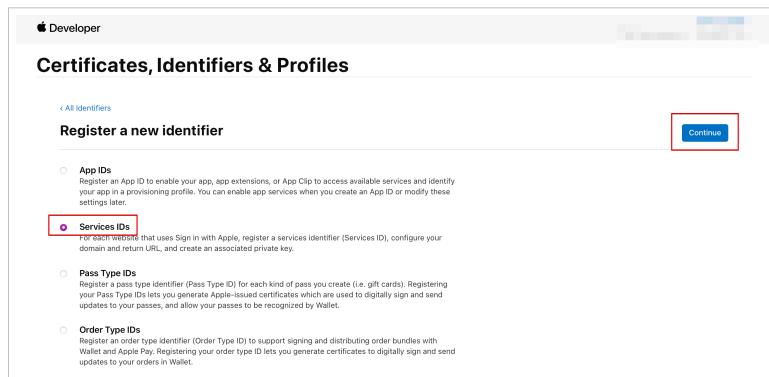
AndroidからAppleでサインインするために、サービスID、キーIDを取得します。Apple Developerサイトにログインし、「Certificates, Identifiers & Profiles」をクリックします。



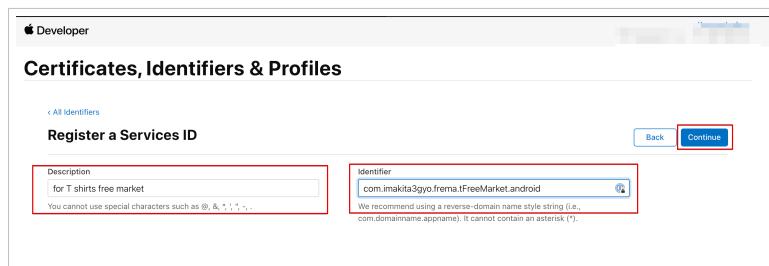
左サイドメニューから「Identifiers」をクリックします。



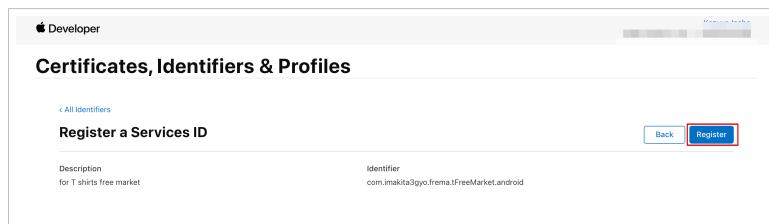
「Service ID s」をクリックし、「Continue」ボタンをクリックします。



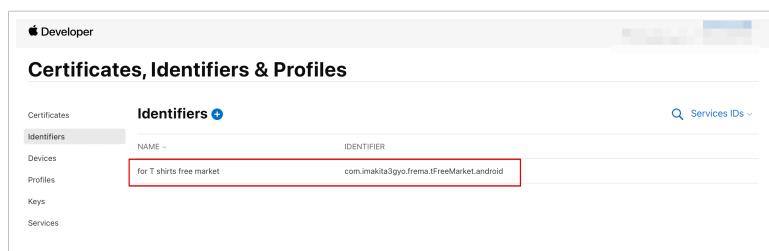
「Description」欄と「Identifier」欄に入力します。「Identifier」欄に入力するものは、ほかで使っていないユニークなものになります。「Continue」ボタンをクリックします。



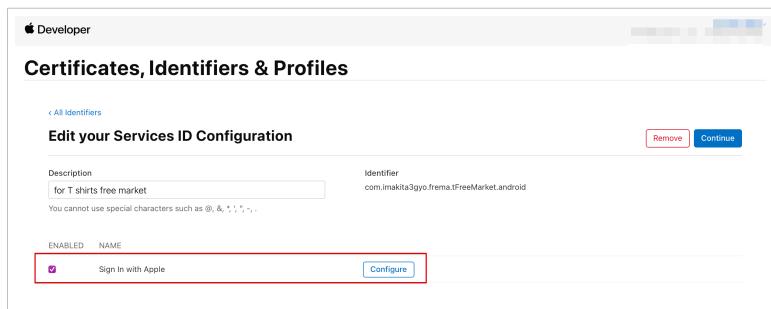
確認ダイアログが表示されますので、「Register」ボタンをクリックします。



ServiceIDが登録されました。登録された「NAME」がリンクになっていますので、クリックします。



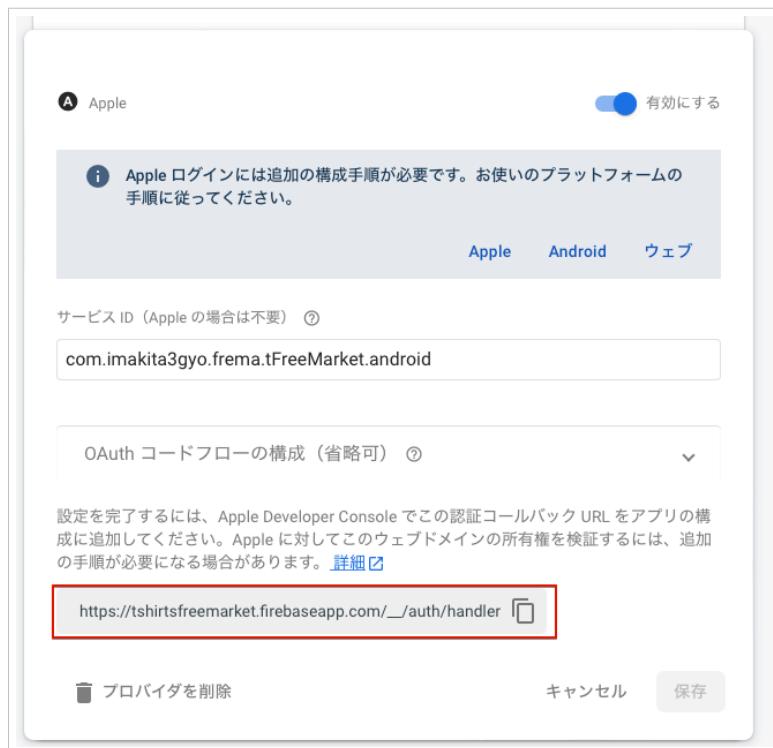
「Sign in with Apple」のチェックボックスをONにし、「Configure」ボタンをクリックします。



ダイアログが表示されますので、登録したアプリケーションIDを選択し、Return URLsにFirebase AuthenticationでAppleを追加したログインプロバイダ登録時の認証コールバックURLを追加します。



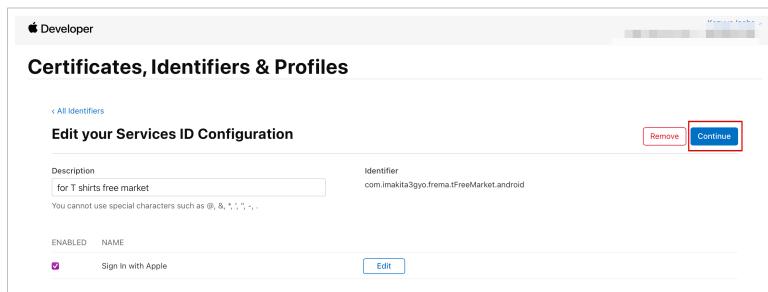
Firebase AuthenticationでApple追加時に表示されています。



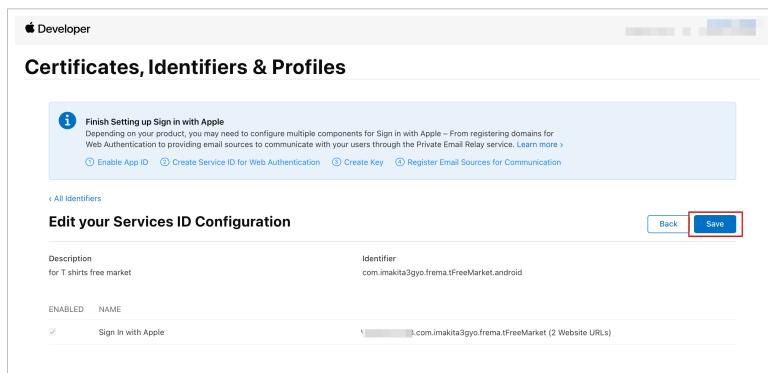
確認ダイアログが表示されますので、「Done」をクリックします。



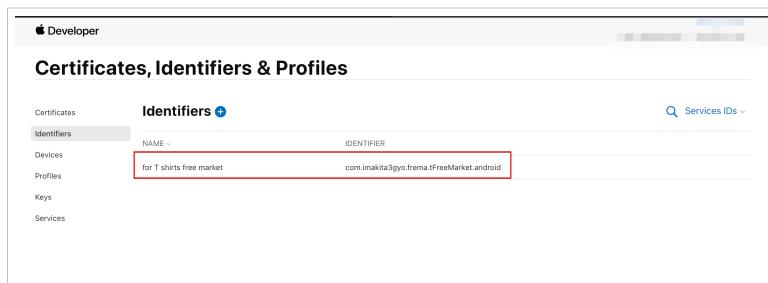
ダイアログが消えますので、「Continue」ボタンをクリックします。



確認が表示されますので、「Save」ボタンをクリックします。



サービスIDの取得が完了しました。



キーの取得

左サイドメニューから「Keys」を選択し、「+」をクリックします。

The screenshot shows the 'Certificates, Identifiers & Profiles' section of the Apple Developer portal. In the sidebar, the 'Keys' button is highlighted with a red box. The main table lists a single key entry:

NAME	SERVICE ENABLED
PushNotificationAuthenticationKeyTwitterClone	1

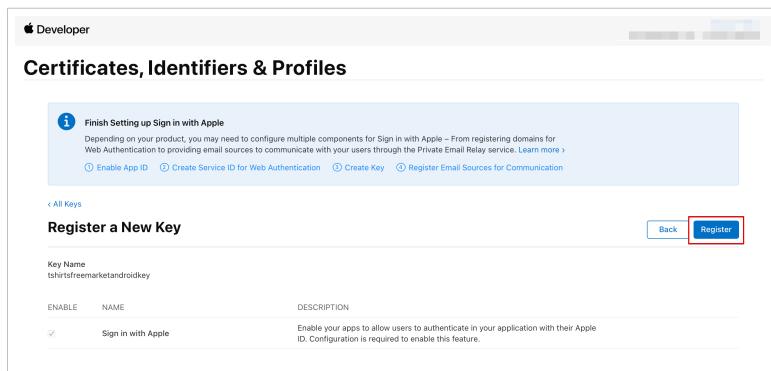
「Register a New Key」の画面で、「Key Name」にユニークなものを入力し、「Sign in with Apple」のチェックボックスをONにします。「Configure」ボタンをクリックします。

The screenshot shows the 'Register a New Key' configuration page. The 'Key Name' field contains 'tshirtfreetmarketandroidkey'. The 'Sign in with Apple' checkbox is checked and highlighted with a red box. The 'Configure' button next to it is also highlighted with a red box.

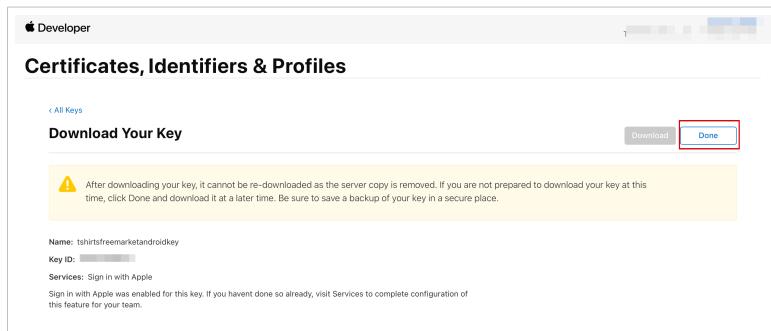
紐付けるアプリケーションIDを選択し、「Save」ボタンをクリックします。

The screenshot shows the 'Configure Key' configuration page. The 'Primary App ID' dropdown is set to 'for T shirts free market (l...com.imakita3gyo.re...)' and highlighted with a red box. The 'Save' button is also highlighted with a red box.

確認画面になりますので、「Register」ボタンをクリックします。



キーIDを控え、「Download」ボタンをクリックします。一度しかダウンロードできません。



♣ 13.2.5 Firebaseへキーを登録

Firebase コンソールの Authentication を選択し、「Sign-in method」で Apple の編集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 active. A table lists the available sign-in providers:

プロバイダ	ステータス
メール / パスワード	有効
Google	有効
Twitter	有効
Apple	有効

A red box highlights the edit icon (pencil) next to the Apple provider's status. There is also a '構成を編集' button at the bottom right of the table.

「サービス ID」、「Apple チーム ID」、「キー ID」、ダウンロードしたファイルをテキストエディタで開き、秘密鍵をコピペします。



以上で準備は完了です。

13.3 アプリケーションにAppleサインイン・メソッドの実装

Firebaseのサイト^{*1}にあるコードそのままです。

アップル

iOS+ アンドロイド ウェブ

開始する前に、Sign In with Appleを構成し、Appleをサインインプロバイダーとして有効にします。

次に、Runnerアプリに「Appleでサインイン」機能があることを確認します。

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

Future<UserCredential> signInWithApple() async {
    final appleProvider = AppleAuthProvider();
    if (kIsWeb) {
        await _auth.signInWithPopup(appleProvider);
    } else {
        await _auth.signInWithAuthProvider(appleProvider);
    }
}
```

▲図13.1: Firebaseドキュメント

認証コントローラへAppleでサインイン・メソッドを割り当てます。

▼認証コントローラへメソッド追加

```
/// AppleIDでサインイン
signInWithApple() async {
    final appleProvider = AppleAuthProvider();
    if (kIsWeb) {
        await _auth.signInWithPopup(appleProvider);
    } else {
        await _auth.signInWithAuthProvider(appleProvider);
    }
}
```

*1 <https://firebase.google.com/docs/auth/flutter/federated-auth?hl=ja&authuser=0#apple>

ログイン選択画面の「Appleアカウント」にメソッドを割り当てます。

▼ ログイン選択画面の Apple アカウント

```
Container(
  decoration: ThemeHelper().buttonBoxDecoration(context),
  child: ElevatedButton(
    style: ThemeHelper().buttonStyle(),
    child: Row(
      children: const [
        FaIcon(FontAwesomeIcons.apple, size: 23),
        Padding(
          padding: EdgeInsets.fromLTRB(10, 0, 10, 0),
          child: Text('Appleアカウント', style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold, color: Colors.white),),
        ),
      ],
    ),
    onPressed: (){
      authController.signInWithApple();
    },
  ),
),
```

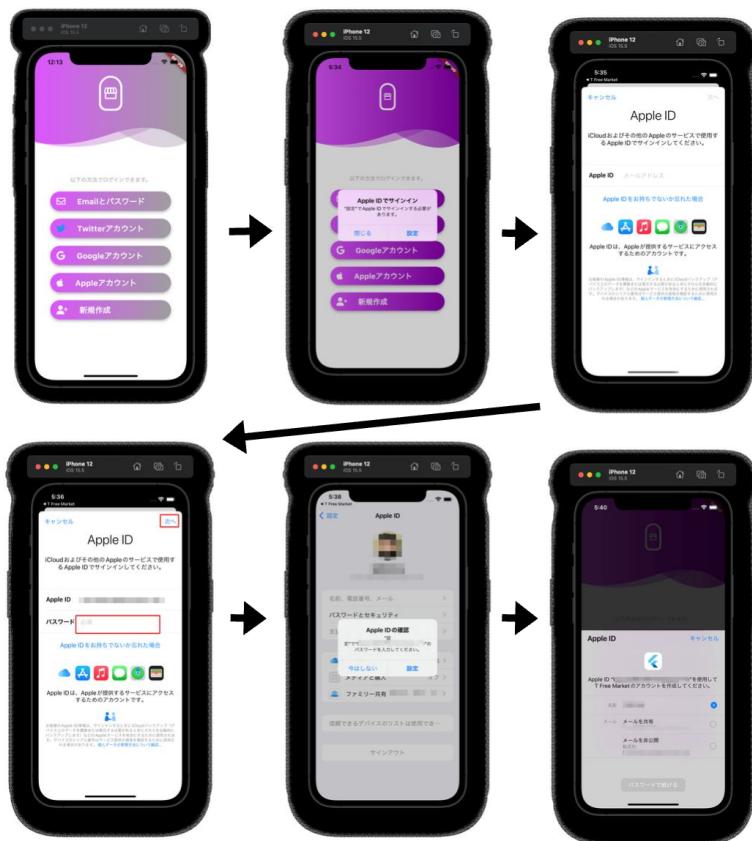
以上で完了です。

13.4

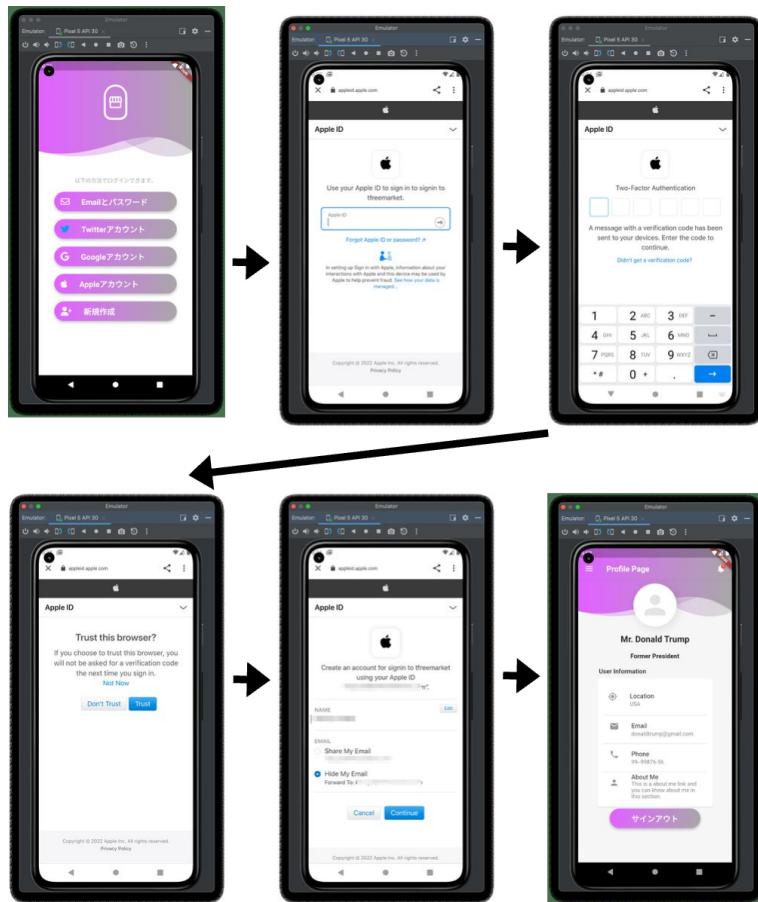
動作確認

iPhoneで確認

iPhoneシミュレータで動作確認しましたが、AppleIDでサインインは実機でないと動作しないとのことです。実機をつないでデバッグしましたところ動作しました。



▲図13.2: iPhoneシミュレータで確認

Androidで確認

▲図 13.3: Android エミュレータで確認

Firebaseで登録確認

Firebaseコンソールでも確認します。

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 are buttons for "ドキュメントに移動" (Move to document), a red notification bell icon, and a help icon. Below the search bar is a blue button labeled "ユーザーを追加" (Add user). The main area displays a table of users:

ID	プロバイダー	作成日	ログイン日	ユーザー UID
[Redacted]	A	202...	202...	mQ4ZH9nj8FfZavp...
[Redacted]	A	202...	202...	Kp2iHjpDNYR20...
[Redacted]	G	202...	202...	L9vgUfUMh4WiG3...
[Redacted]	T	202...	202...	iK1ck3iOE90wMZ...
[Redacted]	G	202...	202...	8H8kEh9LAIv6oG...
mana@yahoo...	E	202...	202...	n1YDXX4EBTS24p...

At the bottom of the table, there are pagination controls: "ページあたりの行数:" (Items per page:), a dropdown menu set to "50", and "1 - 6 of 6".

▲図13.4: Firebaseコンソール

ここまで

までのソースコード

▼ GitHub

```
> git clone -b 09_apple_signin https://github.com/risingforce9zz/tfreemarket.git
```

第 14 章

次号予告

いかがでしたでしょうか？

4つのログイン方法は実装できましたか？

手順をキチンと踏めば、動きます。以前、先輩に言わされたのですが、「コードは書いたとおりにしか動かない。」

さて、次号ですが、

Firebase 認証にアカウントを作る際に（メール、SNS 認証）ストア情報を作ります。また、ストア情報（アバター）を含め編集できるようにします。

ストアが出来上がれば、商品情報も登録・編集ができるようになります。

技術的なことですと、

- ストアコントローラが、Firebase Realtime Database と通信
- ストアコントローラが、保持しているデータを表示
- UI からの操作でコントローラ経由で、Firebase Realtime Database を操作
- 商品画像、アバターは、Cloud Storage へアップロード
- 商品画像は、スマホのカメラなのでカメラへのアクセス。

を予定しています。

状況は、本書のサポートサイト: https://github.com/risingforce9zz/book04_t-shirts-freemarket を参照してください。

ここまで読んでいただき、ありがとうございます。

また、お目にかかりますように・・・

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