App-final (flutter)

중간중간마다 코드를 탑재하면 너무 분량이 많아질 것 같아 앱 실행 화면과, firebaseconsole위주로 탑재하고 코드는 나중에 한번에 탑재하겠습니다.

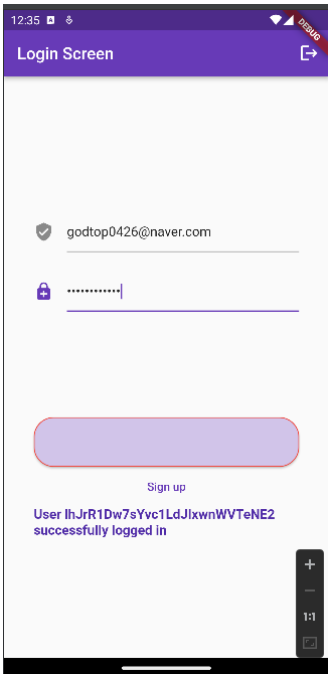
Configuring a Firebase app

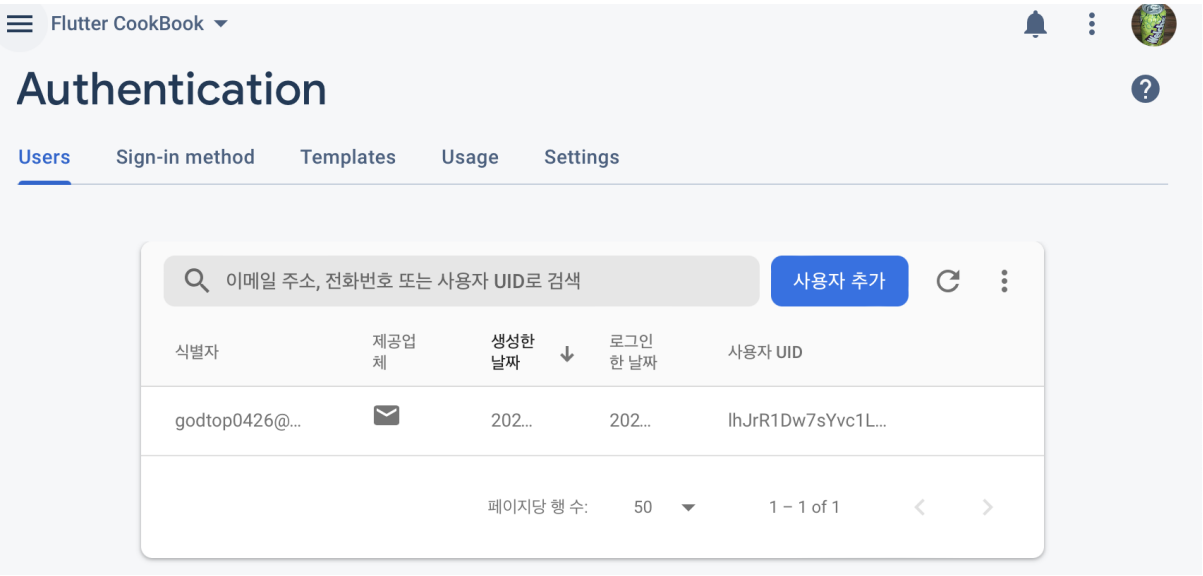
- google-service.json
- defaultConfig → applicationId → `it.flutter.firebase`
- google-services.json → android/app directory
- android → app → build.gradle → `apply plugin: 'com.google.gms.google-services'`
- android → build.gradle → dependencies → `classpath 'com.google.gms:google-services:4.3.14'`
- pubspec.yaml → dependencies → add it

```
firebase_core: ^2.4.0
firebase_auth: ^4.2.0
cloud_firestore: ^4.2.0
```

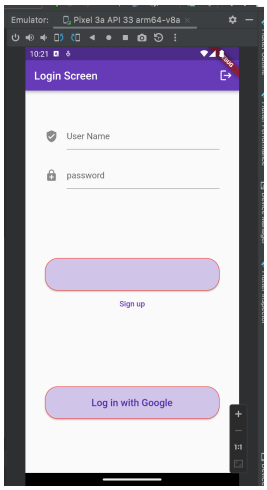
- android → app → build.gradle → `minSdkVersion 21`

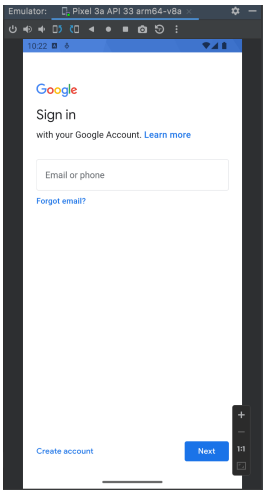
Creating a login form

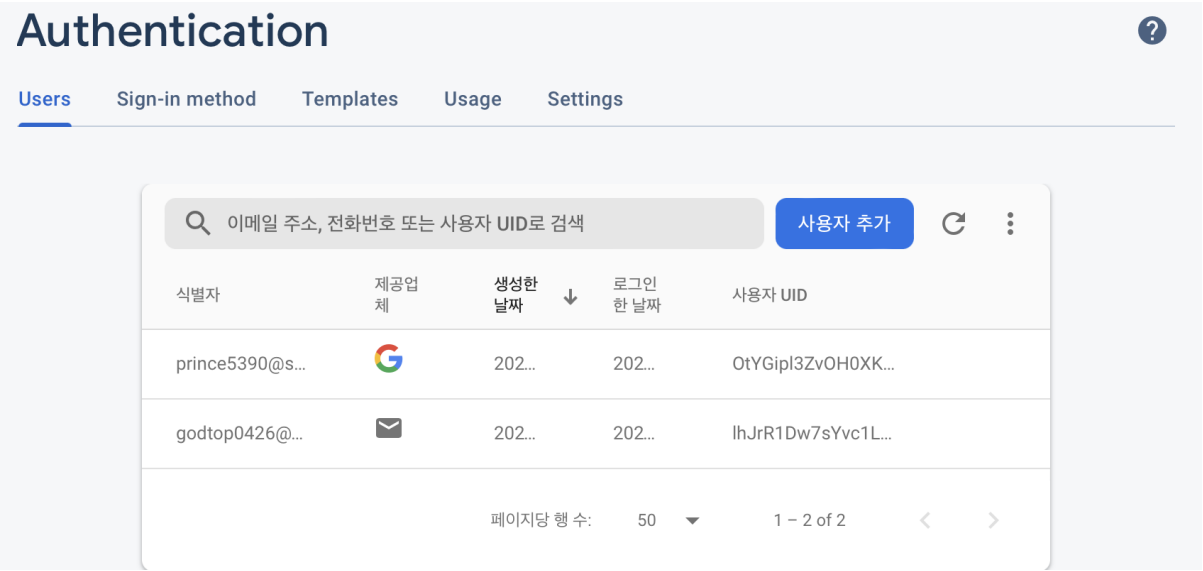




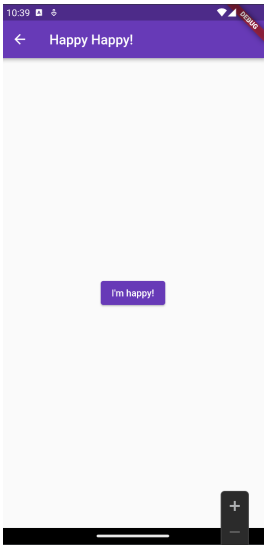
Adding Google Sign-in

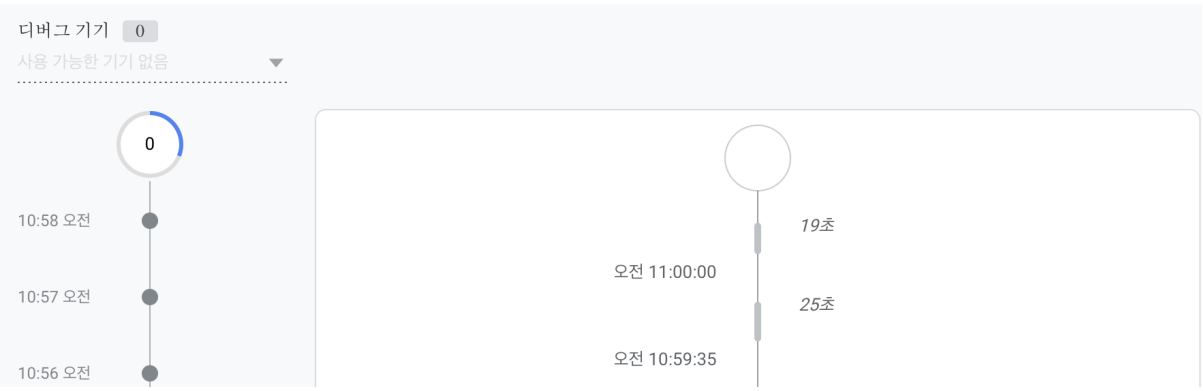






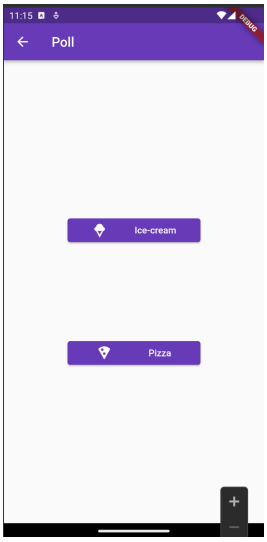
Integrating Firebase Analytics

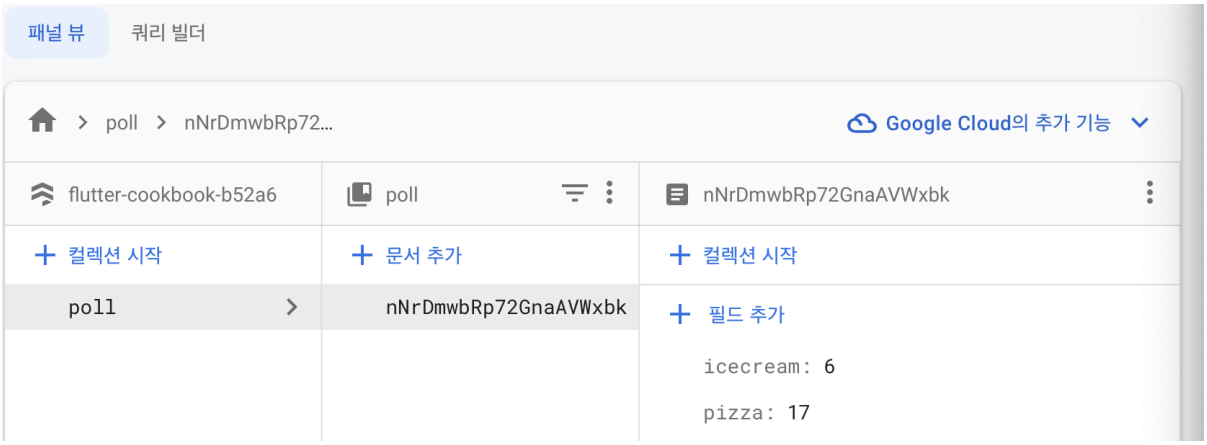




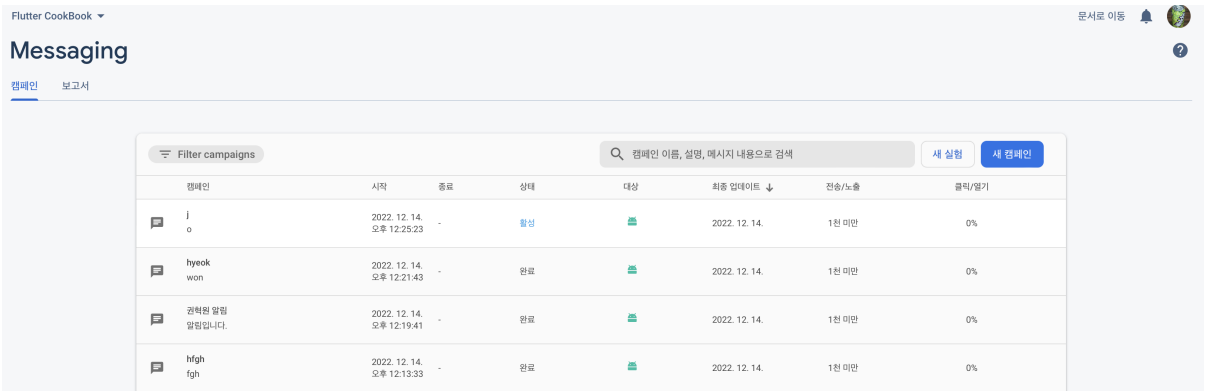
- `adb shell setprop debug.firebase.analytics.app it.flutter.firebase`
- 위 명령어 까지 전부 실행이 되고, I'm happy 스크린으로 변경하는 것 까지 성공 하였으나, 디버그 기기에서 사용 가능한 기기 없음 이 해결되지 않았습니다.

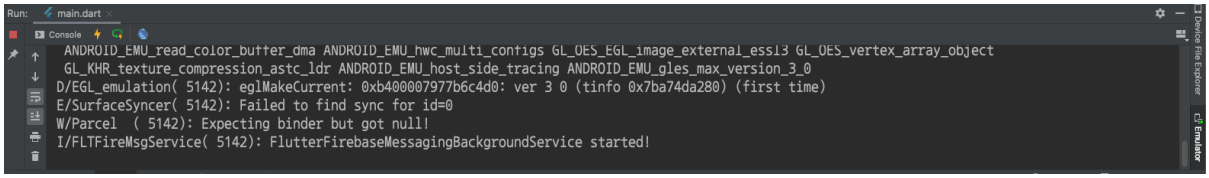
Using Firebase Cloud Firestore





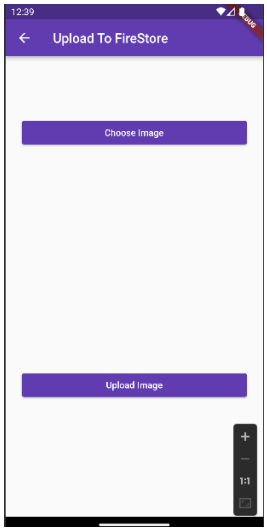
Sending Push Notifications with Firebase Cloud Messaging (FCM)

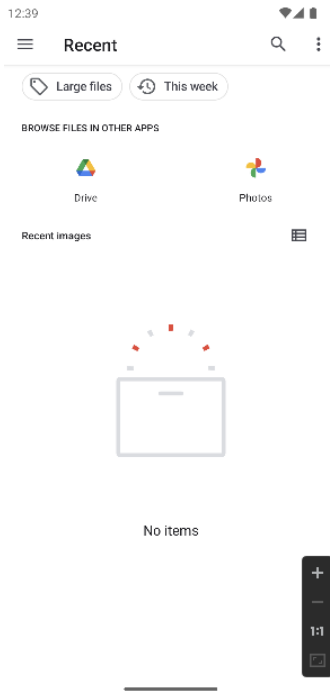


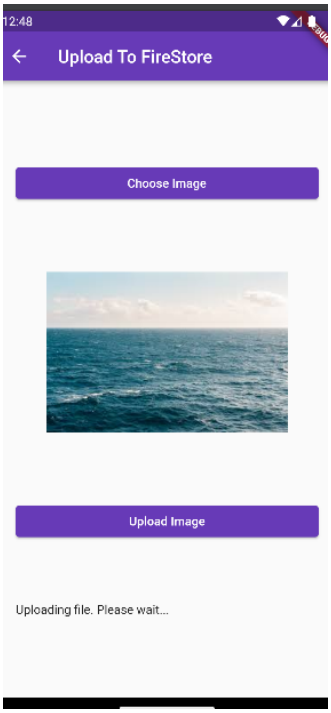


- 위 콘솔 그림에서 FlutterFirebaseMessagingBackgroundService started!가 출력된 것으로 보아 제대로 연결은 되었다고 판단했습니다. 하지만 실제 앱상에서 알림이 뜨는 것 까지는 확인을 못하였습니다.

Storing files in the cloud







아래 부터는 위 모든 과정을 진행한 최종 코드입니다.

```
#pubspec.yaml
name: realfirebase
description: A new Flutter project.

# The following line prevents the package from being accidentally published to
# pub.dev using `flutter pub publish`. This is preferred for private packages.
publish_to: 'none' # Remove this line if you wish to publish to pub.dev

# The following defines the version and build number for your application.
# A version number is three numbers separated by dots, like 1.2.43
# followed by an optional build number separated by a +.
# Both the version and the builder number may be overridden in flutter
# build by specifying --build-name and --build-number, respectively.
# In Android, build-name is used as versionName while build-number used as versionCode.
# Read more about Android versioning at https://developer.android.com/studio/publish/versioning
# In iOS, build-name is used as CFBundleShortVersionString while build-number is used as CFBundleVersion.
# Read more about iOS versioning at
# https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/CoreFoundationKeys.html
# In Windows, build-name is used as the major, minor, and patch parts
# of the product and file versions while build-number is used as the build suffix.
version: 1.0.0+1

environment:
  sdk: '>=2.18.0 <3.0.0'

# Dependencies specify other packages that your package needs in order to work.
# To automatically upgrade your package dependencies to the latest versions
# consider running `flutter pub upgrade --major-versions`. Alternatively,
# dependencies can be manually updated by changing the version numbers below to
# the latest version available on pub.dev. To see which dependencies have newer
# versions available, run `flutter pub outdated`.
dependencies:
  flutter:
    sdk: flutter
  firebase_core: ^2.4.0
  firebase_auth: ^4.2.0
  cloud_firestore: ^4.2.0
```

```
  google_sign_in: ^5.4.2
  firebase_analytics: ^10.0.7
  firebase_messaging: ^14.1.4
  image_picker: ^0.8.6
  firebase_storage: ^11.0.7


  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.2

dev_dependencies:
  flutter_test:
    sdk: flutter

  # The "flutter_lints" package below contains a set of recommended lints to
  # encourage good coding practices. The lint set provided by the package is
  # activated in the `analysis_options.yaml` file located at the root of your
  # package. See that file for information about deactivating specific lint
  # rules and activating additional ones.
  flutter_lints: ^2.0.0

# For information on the generic Dart part of this file, see the
# following page: https://dart.dev/tools/pub/pubspec

# The following section is specific to Flutter packages.
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

  # To add assets to your application, add an assets section, like this:
  # assets:
  #   - images/a_dot_burr.jpeg
  #   - images/a_dot_ham.jpeg

  # An image asset can refer to one or more resolution-specific "variants", see
  # https://flutter.dev/assets-and-images/#resolution-aware

  # For details regarding adding assets from package dependencies, see
  # https://flutter.dev/assets-and-images/#from-packages

  # To add custom fonts to your application, add a fonts section here,
  # in this "flutter" section. Each entry in this list should have a
  # "family" key with the font family name, and a "fonts" key with a
  # list giving the asset and other descriptors for the font. For
  # example:
  # fonts:
  #   - family: Schyler
  #     fonts:
  #       - asset: fonts/Schyler-Regular.ttf
  #       - asset: fonts/Schyler-Italic.ttf
  #         style: italic
  #   - family: Trajan Pro
  #     fonts:
  #       - asset: fonts/TrajanPro.ttf
  #       - asset: fonts/TrajanPro_Bold.ttf
  #         weight: 700
  #
  # For details regarding fonts from package dependencies,
  # see https://flutter.dev/custom-fonts/#from-packages
```

```
//main.dart
import 'package:firebase_core/firebase_core.dart';
import 'login_screen.dart';
import 'package:flutter/material.dart';

void main() async{
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.deepPurple,
      ),
```

```
      home: LoginScreen(),
    );
  }
}
```

```
//login_screen.dart
import 'package:firebase_core/firebase_core.dart';
import 'package:firebase_messaging/firebase_messaging.dart';
import 'package:flutter/material.dart';
import 'package:realfirebase/poll.dart';
import 'happy_screen.dart';
import 'shared/firebase_authentication.dart';
import 'package:realfirebase/upload_file.dart';

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

  @override
  State<LoginScreen> createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
  String _message = '';
  bool _isLogin = true;
  final TextEditingController txtUserName = TextEditingController();
  final TextEditingController txtPassword = TextEditingController();
  late FirebaseAuthentication auth;
  final FirebaseMessaging messaging = FirebaseMessaging.instance;

  @override
  void initState() {
    Firebase.initializeApp().whenComplete(() {
      auth = FirebaseAuthentication();
      FirebaseMessaging.onBackgroundMessage(_firebaseBackgroundMessageReceived);
      setState(() {});
    });
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
          title: Text('Login Screen'),
          actions: [
            IconButton(
              icon: Icon(Icons.logout),
              onPressed: () {
                auth.logout().then((value) {
                  if (value) {
                    setState(() {
                      _message = 'User Logged Out';
                    });
                  } else {
                    _message = 'Unable to Log Out';
                  }
                });
              },
            ),
          ],
        ),
        body: Container(
          padding: EdgeInsets.all(36),
          child: ListView(
            children: [
              userInput(),
              passwordInput(),
              btnMain(),
              btnSecondary(),
              btnGoogle(),
              txtMessage(),
            ], ),
        ));
  }

  Widget userInput() {
    return Padding(
        padding: EdgeInsets.only(top: 24),
        child: TextFormField(
          controller: txtUserName,
          keyboardType: TextInputType.emailAddress,
          decoration: InputDecoration(
              hintText: 'User Name', icon: Icon(Icons.verified_user)),
```

```
        validator: (text) => text!.isEmpty ? 'User Name is required'
            : '', ));
}
Widget passwordInput() {
  return Padding(
      padding: EdgeInsets.only(top: 24),
      child: TextFormField(
        controller: txtPassword,
        keyboardType: TextInputType.emailAddress,
        obscureText: true,
        decoration: InputDecoration(
            hintText: 'password', icon:
        Icon(Icons.enhanced_encryption)),
        validator: (text) => text!.isEmpty ? 'Password is required'
            : '',
      )); }

Widget btnMain() {
  String btnText = _isLogin ? 'Log in' : 'Sign up';
  return Padding(
      padding: EdgeInsets.only(top: 128),
      child: Container(
height: 60,
child: ElevatedButton(
style: ButtonStyle(
backgroundColor: MaterialStateProperty.all
(Theme.of(context).primaryColorLight),
shape: MaterialStateProperty.all
<RoundedRectangleBorder>(
RoundedRectangleBorder(
borderRadius: BorderRadius.circular(24.0),
side: BorderSide(color: Colors.red)),
), ),
child: Text(
btnText,
style: TextStyle(
fontSize: 18, color:
Theme.of(context).primaryColorLight),
),
onPressed: () {
  String userId = '';
  if (_isLogin) {
  auth.login(txtUserName.text, txtPassword.text).then((value) {
  if (value == null) {
  setState(() {
  _message = 'Login Error';
  });
  } else {
  userId = value;
  setState(() {
  _message = 'User $userId successfully logged in';
  });
  FirebaseMessaging.onBackgroundMessage(_firebaseBackgroundMessageReceived);
  FirebaseMessaging.onMessage;
  changeScreen();
  }});
  } else {
  auth.createUser(txtUserName.text,
  txtPassword.text).then((value) {
  if (value == null) {
  setState(() {
  _message = 'Registration Error';
  });
  } else {
  userId = value;
  setState(() {
  _message = 'User $userId successfully signed in';
  });}});}
},
  )));
}

Widget btnSecondary() {
  String buttonText = _isLogin ? 'Sign up' : 'Log In';
  return TextButton(
    child: Text(buttonText),
    onPressed: () {
      setState(() {
        _isLogin = !_isLogin;
      });},);
}

Widget txtMessage() {
  return Text(
    _message,
    style: TextStyle(
```

```
          fontSize: 16, color: Theme.of(context).primaryColorDark,
          fontWeight: FontWeight.bold),
    );
  }

  Widget btnGoogle() {
    return Padding(
        padding: EdgeInsets.only(top: 128),
    child: Container(
    height: 60,
    child: ElevatedButton(
    style: ButtonStyle(
    backgroundColor: MaterialStateProperty.all(
    Theme.of(context).primaryColorLight),
    shape: MaterialStateProperty.all
    <RoundedRectangleBorder>(
    RoundedRectangleBorder(
    borderRadius: BorderRadius.circular(24.0),
    side: BorderSide(color: Colors.red)),
    ), ),
    onPressed: () {
    auth.loginWithGoogle().then((value) {
    if (value == null) {
    setState(() {
    _message = 'Google Login Error';
    });
    } else {
    setState(() {
    _message =
    'User $value successfully logged in with Google';
    });
    changeScreen();
    } });
    },
    child: Text(
    'Log in with Google',
    style: TextStyle(
        fontSize: 18, color:
    Theme.of(context).primaryColorDark),
    ), ),
    ));
  }

  void changeScreen() {
    Navigator.push(
        context, MaterialPageRoute(builder: (context) =>
        UploadFileScreen()));
  }


}

Future _firebaseBackgroundMessageReceived(RemoteMessage message) async {
  print("Notification: ${message.notification?.title} - ${message.notification?.body}");
}
```

```
//firebase_authentication.dart
import 'dart:async';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:google_sign_in/google_sign_in.dart';

class FirebaseAuthentication {
  final FirebaseAuth _firebaseAuth = FirebaseAuth.instance;
  final GoogleSignIn googleSignIn = GoogleSignIn();

  Future<String?> createUser(String email, String password) async {
    try {
      UserCredential credential = await _firebaseAuth
          .createUserWithEmailAndPassword(email: email, password:
      password);
      return credential.user?.uid;
    } on FirebaseAuthException {
      return null;
    }
  }

  Future<String?> login(String email, String password) async {
    try {
      UserCredential credential = await _firebaseAuth
          .signInWithEmailAndPassword(email: email, password:
      password);
      return credential.user?.uid;
    } on FirebaseAuthException {
```

```
      return null;
    }
  }

  Future<bool> logout() async {
    try {
      _firebaseAuth.signOut();
      return true;
    } on FirebaseAuthException {
      return false;
    }
  }

  Future<String?> loginWithGoogle() async {
    final GoogleSignInAccount? googleSignInAccount = await
    googleSignIn.signIn();
    final GoogleSignInAuthentication? googleSignInAuthentication =
    await googleSignInAccount?.authentication;
    final AuthCredential authCredential =
    GoogleAuthProvider.credential(
      accessToken: googleSignInAuthentication?.accessToken,
      idToken: googleSignInAuthentication?.idToken,
    );
    final UserCredential authResult =
    await _firebaseAuth.signInWithCredential(authCredential);
    final User? user = authResult.user;
    if (user != null) {
      return '$user';
    }
    return null;
  }

}
```

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

class HappyScreen extends StatefulWidget {
  HappyScreen({Key? key,}) : super(key: key);
  FirebaseAnalytics? analytics;

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

class _HappyScreenState extends State<HappyScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Happy Happy!'),
      ),
      body: Center(
        child: ElevatedButton(
          child: Text('I\'m happy!'),
          onPressed: () {
            _testAnalytics();
          },
        ),
      ),
    );
  }
  Future<void> _testAnalytics() async {
    await widget.analytics?.logEvent(name: 'Happy', parameters: null);
    //setMessage('setUserId succeeded');
  }
}
```

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

class PollScreen extends StatefulWidget {
  @override
  _PollScreenState createState() => _PollScreenState();
}
class _PollScreenState extends State<PollScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
```

```
        appBar: AppBar(
          title: Text('Poll'),
        ),
        body: Padding(
          padding: const EdgeInsets.all(96.0),
          child: Column(
              mainAxisAlignment: MainAxisAlignment.spaceEvenly,
              children: [
                ElevatedButton(
                  child: Row(
                      mainAxisAlignment: MainAxisAlignment.spaceAround,
                      children: [Icon(Icons.icecream), Text('Ice-cream')]),
                  onPressed: () {
                    vote(false);
                  },
                ),
                ElevatedButton(
                  child: Row(
                      mainAxisAlignment: MainAxisAlignment.spaceAround,
                      children: [Icon(Icons.local_pizza), Text('Pizza')]),
                  onPressed: () {
                    vote(true);
                  },
                ),
              ]
          ),
        ) );
  }

  Future vote(bool voteForPizza) async {
    FirebaseFirestore db = FirebaseFirestore.instance;
    CollectionReference collection = db.collection('poll');
    QuerySnapshot snapshot = await collection.get();
    List<QueryDocumentSnapshot> list = snapshot.docs;
    DocumentSnapshot document = list[0];
    final id = document.id;

    if (voteForPizza) {
      int pizzaVotes = document.get('pizza');
      collection.doc(id).update({'pizza':++pizzaVotes});
    } else {
      int icecreamVotes = document.get('icecream');
      collection.doc(id).update({'icecream':++icecreamVotes});
    }
  }


}
```

```
//upload_file.dart
import 'package:path/path.dart';
import 'package:image_picker/image_picker.dart';
import 'package:firebase_storage/firebase_storage.dart';
import 'package:flutter/material.dart';
import 'dart:io';


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

  @override
  State<UploadFileScreen> createState() => _UploadFileScreenState();
}

class _UploadFileScreenState extends State<UploadFileScreen> {
  String _message = '';
  File? _image;
  final picker = ImagePicker();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Upload To FireStore')),
      body: Container(
        padding: EdgeInsets.all(24),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: [
            ElevatedButton(
              child: Text('Choose Image'),
              onPressed: () {
                getImage();
```

```
              },
            ),
            (_image == null) ? Container(height: 200) : Container(height: 200,
                child: Image.file(_image!)),
            ElevatedButton(
              child: Text('Upload Image'),
              onPressed: () {
                uploadImage();
              },
            ),
            Text(_message),
          ],
        ),),);
  }

  Future getImage() async {
    final pickedFile = await picker.pickImage(source: ImageSource.gallery);
    setState(() {
      if (pickedFile != null) {
        _image = File(pickedFile.path);
      } else {
        print('No image selected.');
      }});}

  Future uploadImage() async {
    if (_image != null) {
      String fileName = basename(_image!.path);
      FirebaseStorage storage = FirebaseStorage.instance;
      Reference ref = storage.ref(fileName);
      setState(() {
        _message = 'Uploading file. Please wait...';
      });
      ref.putFile(_image!).then((TaskSnapshot result) {
        if (result.state == TaskState.success) {
          setState(() {
            _message = 'File Uploaded Successfully';
          });
        } else {
          setState(() {
            _message = 'Error Uploading File';
          });
        }});}}

}
```

13장 - 머신러닝

- 텐서플로우 : 구글이 2011년에 개발을 시작하여 2015년에 오픈 소스로 공개한 기계학습 라이브러리.
- **Microsoft Cognitive Toolkit**
- 머신러닝 : 한국어로 직역해보면 **“기계 학습”**이 되겠죠? 머신러닝은 **인공지능을 만들기 위해 기계를 학습시키는 다양한 방법**에 대한 학문으로 ‘로봇공학’, ‘제어계측공학’과 같이 **하나의 학문**이랍니다.
- 딥러닝 : 딥러닝(Deep Learning)이란 머신러닝보다 더 작은 개념으로 ‘신경망’을 통해 인공지능을 만드는 머신러닝의 한 종류입니다.
- 인공지능 : 인공 지능(AI)은 학습, 문제 해결, 패턴 인식 등과 같이 주로 인간 지능과 연결된 인지 문제를 해결하는 데 주력하는 컴퓨터 공학 분야입니다.
- **인공지능 > 머신러닝 > 딥러닝**
- 바코드 인식 : 인터넷 연결 필요치 않음
- 얼굴인식
  - 얼굴 특징 인식 및 위치 찾기 감지된 모든 얼굴의 눈, 귀, 볼, 코, 입의 좌표를 확인합니다.
  - 얼굴 특징의 윤곽 가져오기 감지된 얼굴과 눈, 눈썹, 입술, 코의 윤곽을 가져옵니다.
  - 표정 인식: 사람이 웃고 있는지 아니면 눈을 감고 있는지 판단합니다.
  - 동영상 프레임에서 얼굴 추적 감지된 고유한 얼굴의 식별자를 가져옵니다. 식별자는 호출 전체 에서 일관되므로 동영상 스트림의 특정인에 대해 이미지 조작을 할 수 있습니다.
  - 동영상 프레임을 실시간으로 처리 얼굴 인식은 기기에서 실행되며, 동영상 조작과 같은 실시간 애플리케이션에서 사용하기에 충분히 빠릅니다.

- 얼굴인식 개념

얼굴 인식은 디지털 이미지 또는 동영상과 같은 시각적 미디어에서 사람의 얼굴을 찾습니다. 얼굴이 감지되면 연결된 위치, 크기, 방향이 설정되며 눈과 코와 같은 랜드마크를 검색할 수 있습니다.

다음은 ML Kit의 얼굴 인식 기능과 관련하여 사용되는 용어입니다.

- 얼굴 추적은 얼굴 인식을 동영상 시퀀스로 확장합니다. 길이에 관계없이 동영상에 등장하는 모 든 얼굴은 프레임 간에 추적할 수 있습니다. 즉, 연속 동영상 프레임에서 인식된 얼굴이 동일한 사람임을 식별할 수 있습니다. 이는 얼굴 인식의 한 형태가 아닙니다. 얼굴 추적은 동영상 시퀀 스에서 얼굴의 위치와 움직임을 기반으로만 추론합니다.
- 랜드마크는 얼굴 내의 관심 장소입니다. 왼쪽 눈, 오른쪽 눈, 코 밑부분이 모두 랜드마크의 예입 니다. ML Kit는 인식된 얼굴에서 랜드마크를 찾는 기능을 제공합니다.
- 곡선은 얼굴 특징의 형태를 따라 이어지는 점들의 집합입니다. ML Kit는 얼굴 윤곽을 찾는 기 능을 제공합니다.
- 분류는 특정 얼굴 특징이 있는지 확인합니다. 예를 들어 얼굴이 눈을 뜨고 있는지 웃거나 웃고 있는지에 따라 분류할 수 있습니다.

얼굴 방향

다음 용어는 카메라를 기준으로 한 얼굴의 방향 각도를 설명합니다.

- 오일러 X: 오일러 X 각도가 양수인 얼굴이 위로 향합니다.
- 오일러 Y: 오일러 Y각이 양수인 얼굴은 카메라 오른쪽을 바라보거나 음수인 경우 왼쪽을 바라
- 봅니다.
- 오일러 2: 양의 오일러 z각이 있는 얼굴은 카메라를 기준으로 시계 반대 방향으로 회전합니다.

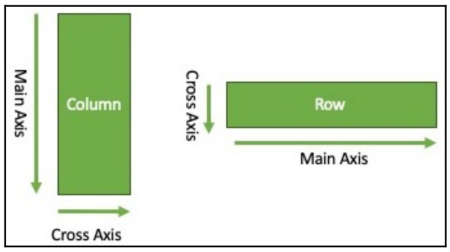
- 위젯 사이의 공간 : padding
- 화면 구조 : scaffold(미리 appBar같은 전형적인 구조를 제공해 주는 것임!)
- 안드로이드 : material
- 애플 : cupertino
- appBar : 앱 최상단 제목 부분
- drawer : 햄버거 아이콘
- 위젯 크기 조정 : aspectRatio
- 텍스트에 여러 속성 부여 : RichText

```
RichText(
             text: TextSpan(
               text: 'Flutter text is ',
               style: TextStyle(fontSize: 22, color: Colors.black),
               children: <TextSpan>[
                 TextSpan(
                   text: 'really ',
                   style: TextStyle(
                     fontWeight: FontWeight.bold,
                     color: Colors.red,
                   ),
                   children: [
                     TextSpan(
                       text: 'powerful.',
                       style: TextStyle(
                         decoration: TextDecoration.underline,
                         decorationStyle: TextDecorationStyle.double,
                         fontSize: 40,
), ),
], ),
], ),
)
//0000000000000000000000000000
Text(
               'Hello, World!',
               style: TextStyle(fontSize: 16),
), Text(
               'Text can wrap without issue',
               style: Theme.of(context).textTheme.headline6,
             ),
```

- 앱 상에서 구글 폰트 쓰려면 아래 코드 쓰고 → pub get

```
dependencies:
  flutter:
    sdk: flutter
  google_fonts: ^2.0.0

# To add assets to your application, add an assets section, like this:
  assets:
    - assets/images/

  # An image asset can refer to one or more resolution-specific "variants", see
  # https://flutter.dev/assets-and-images/#resolution-aware

  # For details regarding adding assets from package dependencies, see
  # https://flutter.dev/assets-and-images/#from-packages

  # To add custom fonts to your application, add a fonts section here,
  # in this "flutter" section. Each entry in this list should have a
  # "family" key with the font family name, and a "fonts" key with a
  # list giving the asset and other descriptors for the font. For
  # example:
  fonts:
    - family: nanum-gothic
      fonts:
        - asset: assets/fonts/nanum-gothic/NanumGothic.otf
#     - family: Trajan Pro
#       fonts:
#         - asset: fonts/TrajanPro.ttf
#         - asset: fonts/TrajanPro_Bold.ttf
#           weight: 700
```

- 앱 상에서 이미지 불려오려면 : `Image.asset('assets/beach.jpg')`



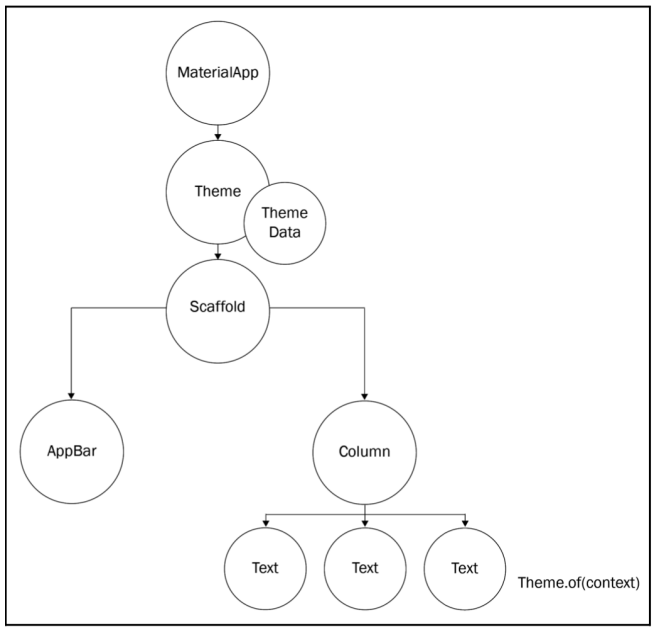
- expanded

```
Widget _buildExpanded(BuildContext context) {
       return SizedBox(
         height: 100,
         child: Row(
           children: <Widget>[
             LabeledContainer(
               width: 100,
               color: Colors.green,
               text: '100',
), Expanded(
               child: LabeledContainer(
                 color: Colors.purple,
                 text: 'The Remainder',
                 textColor: Colors.white,
), ),
             LabeledContainer(
               width: 40,
               color: Colors.green,
               text: '40',
) ],
), );
}
```

- flexible

```
Widget _buildFlexible(BuildContext context) {
             return SizedBox(
               height: 100,
               child: Row(
                 children: <Widget>[
```

```
                    Flexible(
                      flex: 1,
                      child: LabeledContainer(
                        color: Colors.orange,
                        text: '25%',
                      ),
), Flexible(
                      flex: 1,
                      child: LabeledContainer(
                        color: Colors.deepOrange,
                        text: '25%',
                      ),
), Flexible(
                      flex: 2,
                      child: LabeledContainer(
                        color: Colors.blue,
                        text: '50%',
                      ),
) ],
), );
}
```



- 스톱워치
  - `statelesswidget`(스테이트리스 프로세스 또는 애플리케이션은 격리된 것으로 간주됩니다. 과거 트랜잭션에 대한 정보 또는 참조가 저장되지 않기 때문입니다.) → `statefulwidget`(이에 반해, 스테이트풀 애플리케이션과 프로세스는 온라인 뱅킹이나 이메일처럼 여러 번 반환될 수 있습니다. 스테이트풀은 이전 트랜잭션의 컨텍스트에 따라 수행되며, 현재 트랜잭션이 이전 트랜잭션에서 발생한 상황에 영향을 받습니다. 이러한 이유로 스테이트풀 애플리케이션은 사용자에게 받은 요청을 처리할 때마다 같은 서버를 사용합니다.)
- 모든 statefulwidget에는 수명 주기를 유지해야할 새로운 클래스가 있어야 한다. 때문에 private로 두개씩 선언되는 것 이다!
- statefulwidget의 수명 주기를 도와주는 메서드 : initstate, didchangedependencies, build, dispose
- 아래 코드는 stateful, stateless 기본 양식

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

class Test1 extends StatefulWidget {
  const Test1({Key? key}) : super(key: key);

  @override
  State<Test1> createState() => _Test1State();
}

class _Test1State extends State<Test1> {
  @override
  Widget build(BuildContext context) {
    return Container();
```

```
  }
}
//0000000000000000000000000000000000000000000000
class Test2 extends StatelessWidget {
  const Test2({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

- 버튼(elevatedButton, textButton)

```
return Scaffold(
              appBar: AppBar(
                title: Text('Stopwatch'),
              ),
              body: Column(
                mainAxisAlignment: MainAxisAlignment.center,
                children: <Widget>[
                  Text(
                    '$seconds ${_secondsText()}',
                    style: Theme.of(context).textTheme.headline5,
                  ),
                  SizedBox(height: 20),
                  Row(
                    mainAxisAlignment: MainAxisAlignment.center,
                    children: <Widget>[
                      ElevatedButton(
                            style: ButtonStyle(
                              backgroundColor: MaterialStateProperty
                              .all<Color>(Colors.green),
                              foregroundColor: MaterialStateProperty
                              .all<Color>(Colors.white),
                              ),
                            child: Text('Start'),
                            onPressed: null,
                          ),
                      SizedBox(width: 20),
                      TextButton(
                            style: ButtonStyle(
                              backgroundColor: MaterialStateProperty
                              .all<Color>(Colors.red),
                              foregroundColor: MaterialStateProperty
                              .all<Color>(Colors.white),
                              ),
                            child: Text('Stop'),
                            onPressed: null,
), ],
) ],
), );
```

- 아이디, 패스워드 상자 : TextField → TextFormField

```
TextFormField(
        controller: _emailController,
        keyboardType: TextInputType.emailAddress,
        decoration: InputDecoration(labelText: 'Email'),
        validator: (text) {
          if (text.isEmpty) {
            return 'Enter the runner\'s email.';
}
          final regex = RegExp('[^@]+@[^\.]+\..+');
          if (!regex.hasMatch(text)) {
            return 'Enter a valid email';
          }
          return null;
        },
),
```

- 화면변경 : pushreplacement

```
Navigator.of(context).pushReplacement(
      MaterialPageRoute(
```

- model : 앱 구조상 데이터를 처리하는 클래스

- view : 앱 구조상 데이터를 화면에 표시하는 클래스
- 비동기 : future, await, async