

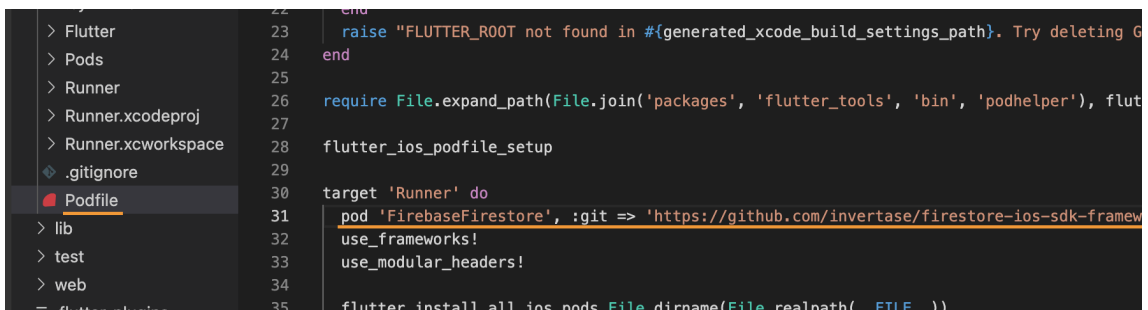
4. (MacOS 쓰시는 분들만) **Podfile** 에 30번째 줄 맨 뒤에 아래 코드스니펫을 붙여 넣어주세요.



firebase의 경우 iOS에서 처음 실행 시간이 오래 걸리는 문제점이 있습니다. 이 문제를 해결하기 위해 아래와 같이 코드를 추가해 줍니다.

▼ [코드스니펫] ios / Podfile firestore

```
pod 'FirebaseFirestore', :git => 'https://github.com/in
```





만약 실행시 아래와 같은 애러가 발생하는 경우

```
DEBUG CONSOLE  ...  Filter (e.g. text, lexclude)

git switch -
Turn off this advice by setting config variable advice.detachedHead to
false
Error: CocoaPods's specs repository is too out-of-date to satisfy dependencies.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 14 Pro Max.
Exited
> Please start a debug session to evaluate expressions
```

Terminal에서 다음 명령어를 실행해 주세요.

```
cd ios && pod repo update && cd ..
```



Firebase 버전이 업데이트 됨에 따라 해당 버전에 맞는 firestore-ios-sdk 버전을 추가해야하며 24.03.11 기준 10.22.0 버전(코드 스니펫 맨 끝에 있습니다)으로 진행하시면 됩니다.

만약 버전이 맞지 않는 경우, 실행시 다음과 같이 에러가 나옵니다.

```
PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE  TERMINAL  Filter (e.g. text, lexclude)  Dart (iPhone 15 Pro Max)

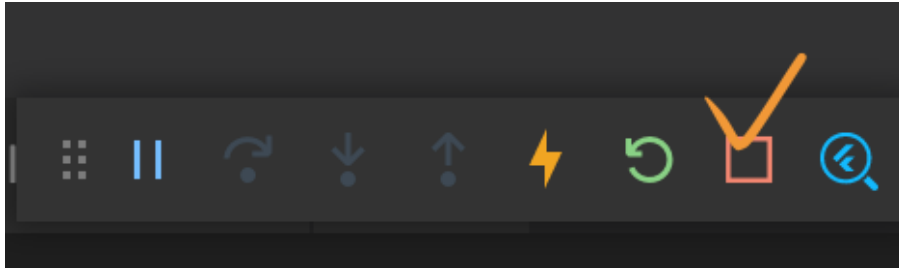
CDN: trunk Relative path downloaded: Specs/6/3/6/FirebaseAuth/10.20.0/FirebaseAuth.podspec.json, save ETag: W/"F50-nFXADSwwVdiu0gKE1EyG+g
CDN: trunk Relative path downloaded: Specs/6/3/6/FirebaseAuth/10.18.0/FirebaseAuth.podspec.json, save ETag: W/"f45-EjFLQ4+nYEHJDI+4bWz8ng
CDN: trunk Relative path downloaded: Specs/6/3/6/FirebaseAuth/10.19.0/FirebaseAuth.podspec.json, save ETag: W/"F50-JJKs3NEkXYQFq99YoFDWag
CDN: trunk Relative path: Specs/6/3/6/FirebaseAuth/10.22.0/FirebaseAuth.podspec.json modified during this run! Returning local
[!] CocoaPods could not find compatible versions for pod "FirebaseFirestore":
  In Podfile:
    FirebaseFirestore (from 'https://github.com/invertase/firestore-ios-sdk-frameworks.git', tag '10.20.0')

    cloud_firestore (from '.symlinks/plugins/cloud_firestore/ios') was resolved to 4.15.8, which depends on
      Firebase/Firestore (= 10.22.0) was resolved to 10.22.0, which depends on
        FirebaseFirestore (-> 10.22.0)

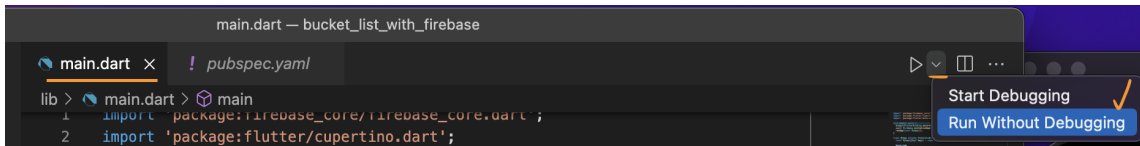
/opt/homebrew/Cellar/cocoapods/1.14.3/libexec/gems/molinillo-0.8.0/lib/molinillo/resolution.rb:317:in 'raise_error_unless_state'
/opt/homebrew/Cellar/cocoapods/1.14.3/libexec/gems/molinillo-0.8.0/lib/molinillo/resolution.rb:299:in 'block in unwind_for_conflict'
<internal:kernel>:90:in 'tap'
```

DEBUG CONSOLE에서 확인하실 수 있으며, 위에 보시면 24.03.11일 기준으로는 10.22.0 버전을 사용하라는 메시지를 보실 수 있습니다. (해당 버전은 실행 시점에 따라 다르게 출력될 수 있으며, 해당 버전으로 코드 스니펫 맨 끝에 숫자를 변경해 주시면 됩니다. 버전 목록은 [링크](#)에서 확인할 수 있습니다.)

5. `pubspec.yaml` 파일을 수정하였으니, 우측 상단에 `Stop` 버튼을 눌러 에뮬레이터를 종료한 뒤 다시 시작해 주세요.



macOS에서는 `main.dart` 파일을 열어야 아래 시작 버튼이 보입니다.



▼ 6) Firebase 사용 준비

💡 Flutter에서 Firebase를 사용하는 방법은 아래 공식 문서를 참고해 주세요.

▼ [코드스니펫] Flutter & Firebase 공식문서

<https://firebase.flutter.dev/docs/overview/>

💡 Flutter에서 Firebase를 사용하려면 **firebase_core** 패키지의 `Firebase.initializeApp();` 를 맨 처음 실행해줘야 합니다.

1. `main.dart` 에 4 ~ 6라인에 `main()` 함수를 삭제한 뒤 코드스니펫을 4번째 줄에 붙여 넣어 주세요.

▼ [코드스니펫] main 함수

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized(); // main 함수0
  await Firebase.initializeApp(); // firebase 앱 시작
  runApp(const MyApp());
}
```

```

1  import 'package:flutter/cupertino.dart';
2  import 'package:flutter/material.dart';
3
4  Run | Debug | Profile
5  void main() async {
6    WidgetsFlutterBinding.ensureInitialized(); // main
7    await Firebase.initializeApp(); // firebase 앱 시작
8    runApp(const MyApp());
9  }

```

2. 6번째 줄 `Firebase` 부분에 `firebase_core` 패키지가 Import가 안되어 발생하는 에러가 있습니다. `Firebase` 를 선택한 뒤 Quick Fix(`Ctrl/Cmd + .`)를 누르고 `Import library ~` 를 선택한 뒤 저장해 주세요.



만약 `Import library ~` 가 안보이신다면, `pubspec.yaml` 파일을 열고 저장 단축 키를 누르신 뒤 다시 시도해주세요.

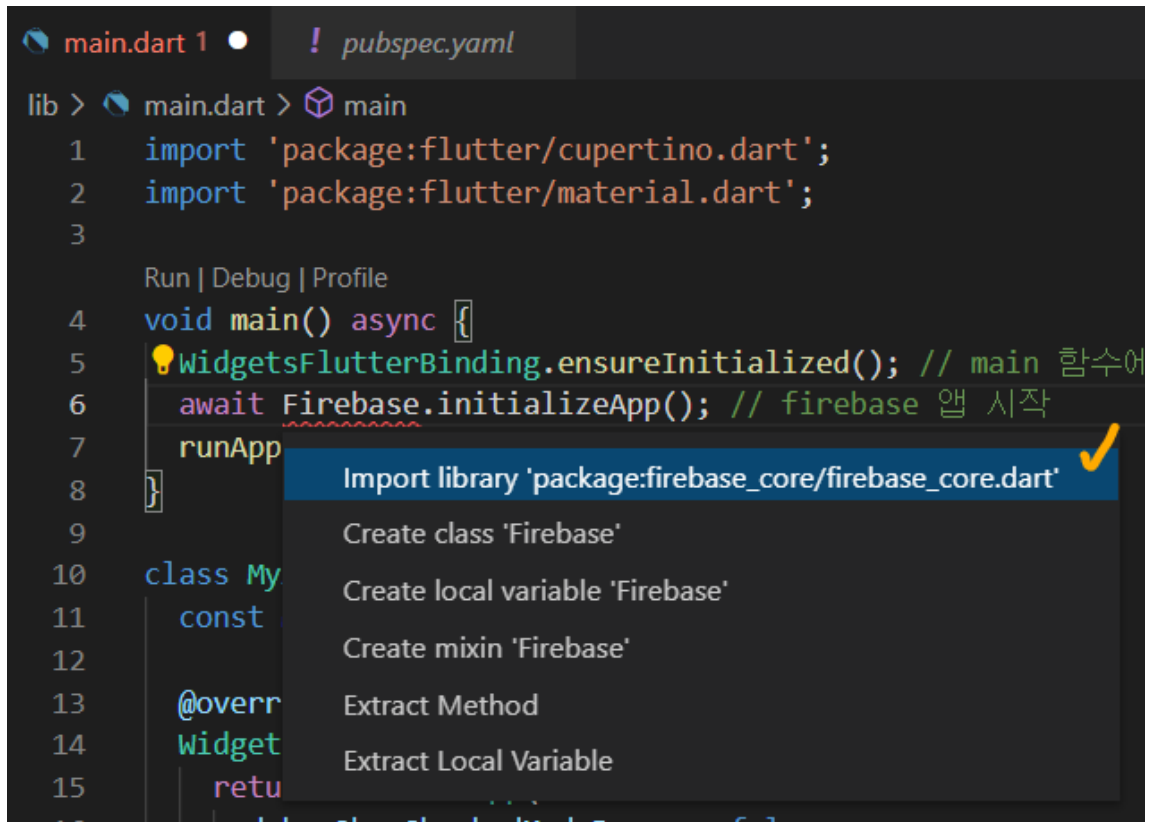
저장 단축키

- window :

`Ctrl + S`

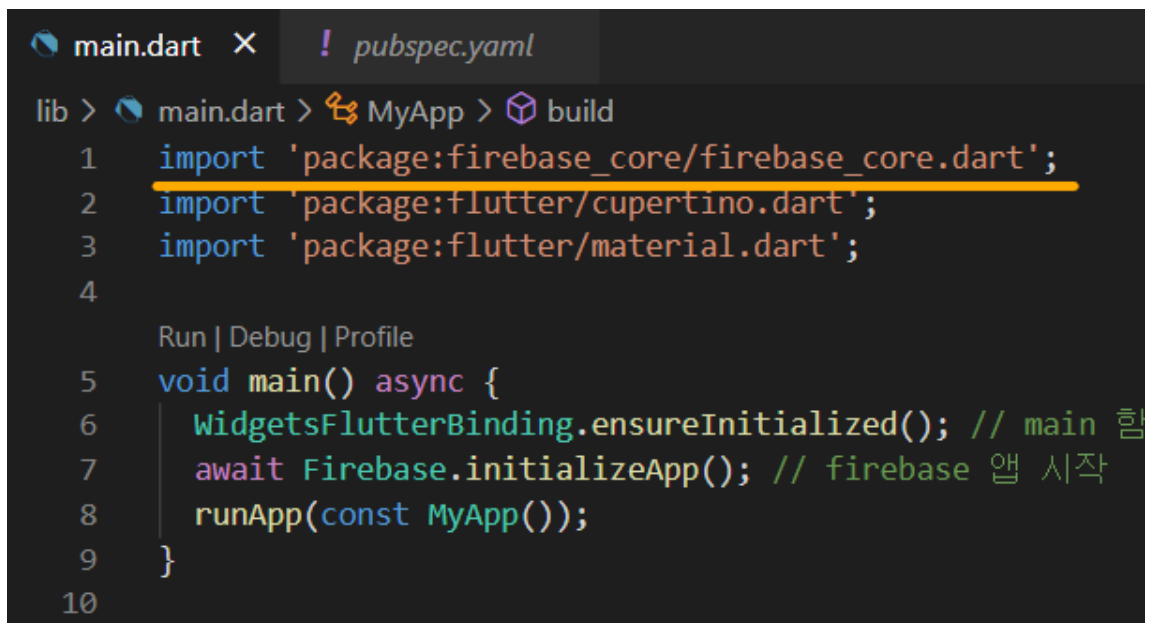
- macOS :

`Cmd + S`



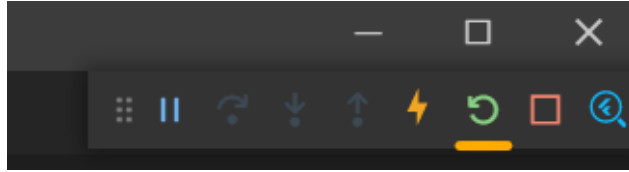
```
main.dart 1 • ! pubspec.yaml
lib > main.dart > main
1 import 'package:flutter/cupertino.dart';
2 import 'package:flutter/material.dart';
3
4 void main() async {
5   WidgetsFlutterBinding.ensureInitialized(); // main 함수여
6   await Firebase.initializeApp(); // firebase 앱 시작
7   runApp
8 }
9
10 class MyApp
11   const
12
13   @override
14   Widget
15   return
```

첫 번째 줄에 firebase_core가 Import 되었고 문제가 해결되었습니다.

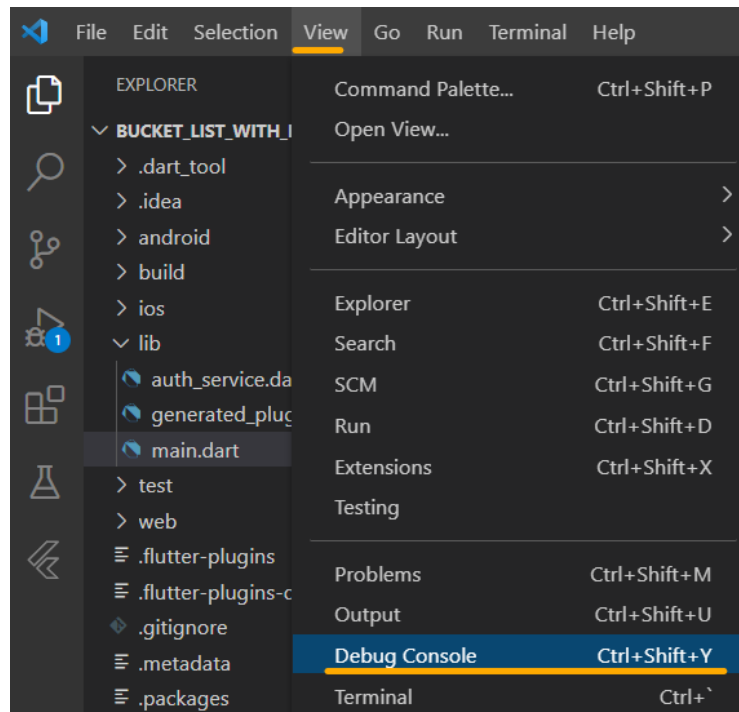


```
main.dart × ! pubspec.yaml
lib > main.dart > MyApp > build
1 import 'package:firebase_core/firebase_core.dart';
2 import 'package:flutter/cupertino.dart';
3 import 'package:flutter/material.dart';
4
5 void main() async {
6   WidgetsFlutterBinding.ensureInitialized(); // main 함
7   await Firebase.initializeApp(); // firebase 앱 시작
8   runApp(const MyApp());
9 }
10
```

이제 Firebase를 사용할 준비가 완료 되었습니다. **Restart** 를 해주세요.



3. 만약 **Restart** 를 해도 에뮬레이터에서 에러가 발생하는 경우 **View** → **Debug Console** 을 열어 에러 로그를 확인해주세요.





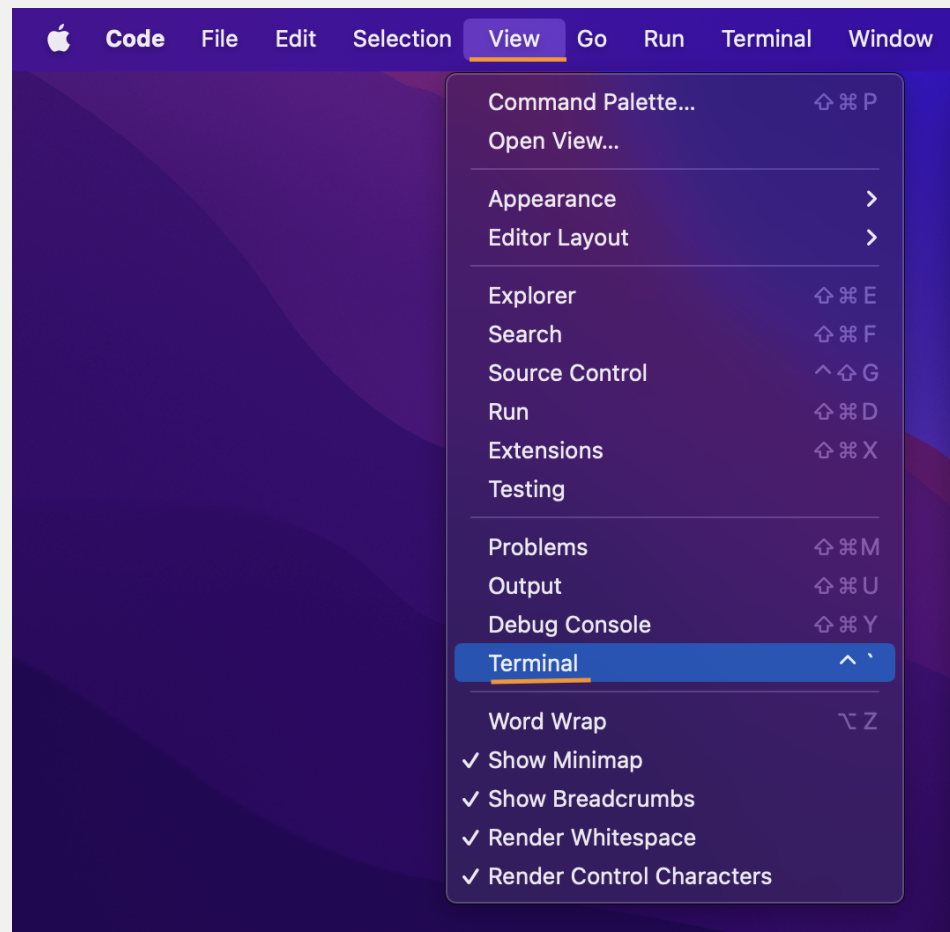
Apple Chip 기기를 사용하는 경우 아래와 같이 `Error running pod install` 에러가 출력될 수 있습니다.

```
PROBLEMS 27 OUTPUT DEBUG CONSOLE TERMINAL Filter (e.g. text, lexclude)
ndencies'
from /Library/Ruby/Gems/2.6.0/gems/cocoapods-1.11.3/lib/cocoapods/user_interface.rb:64:in `section'
from /Library/Ruby/Gems/2.6.0/gems/cocoapods-1.11.3/lib/cocoapods/installer.rb:240:in `resolve_dependencies'
from /Library/Ruby/Gems/2.6.0/gems/cocoapods-1.11.3/lib/cocoapods/installer.rb:161:in `install!'
from /Library/Ruby/Gems/2.6.0/gems/cocoapods-1.11.3/lib/cocoapods/command/install.rb:52:in `run'
from /Library/Ruby/Gems/2.6.0/gems/cocade-1.1.0/lib/cocade/command.rb:334:in `run'
from /Library/Ruby/Gems/2.6.0/gems/cocoapods-1.11.3/lib/cocoapods/command.rb:52:in `run'
from /Library/Ruby/Gems/2.6.0/gems/cocoapods-1.11.3/bin/pod:55:in `<top (required)>'
from /usr/local/bin/pod:23:in `load'
from /usr/local/bin/pod:23:in `<main>'
Error: To set up CocoaPods for ARM macOS, run:
arch -x86_64 sudo gem install ffi

Error running pod install
Error launching application on iPhone 13 Pro Max.
Exited
>
```

그런 경우 다음과 같이 실행해 주세요.

1. `View` → `Terminal` 을 선택해 주세요.



2. 터미널에 아래 명령어를 붙여 넣은 뒤 엔터를 눌러주세요.

```
cd ios && arch -x86_64 pod install && cd ..
```

▼ 위 명령어가 궁금하신 분들은 왼쪽 토글을 눌러주세요.

- `cd ios` : `ios` 폴더로 이동합니다.
- `arch -x86_64 pod install` : `pod install` 이라는 명령어를 Apple Chip에서 실행합니다. `pod` 이란 iOS에 앱을 실행하는데 필요한 외부 파일을 받아오는 Cocoapod이라는 프로그램입니다. Cocoapod을 Apple Chip에서 실행하려면 `arch -x86_64` 이라는 명령어를 앞에 붙여야합니다.
- `cd ..` : `ios` 폴더에서 나옵니다.

3. 더 이상 출력되는 결과가 없는 경우, 다시 실행해 보시면 정상적으로 작동합니다.

이제 정상적으로 화면이 출력 됩니다.

로그인

로그인해 주세요 😊

이메일

비밀번호

로그인

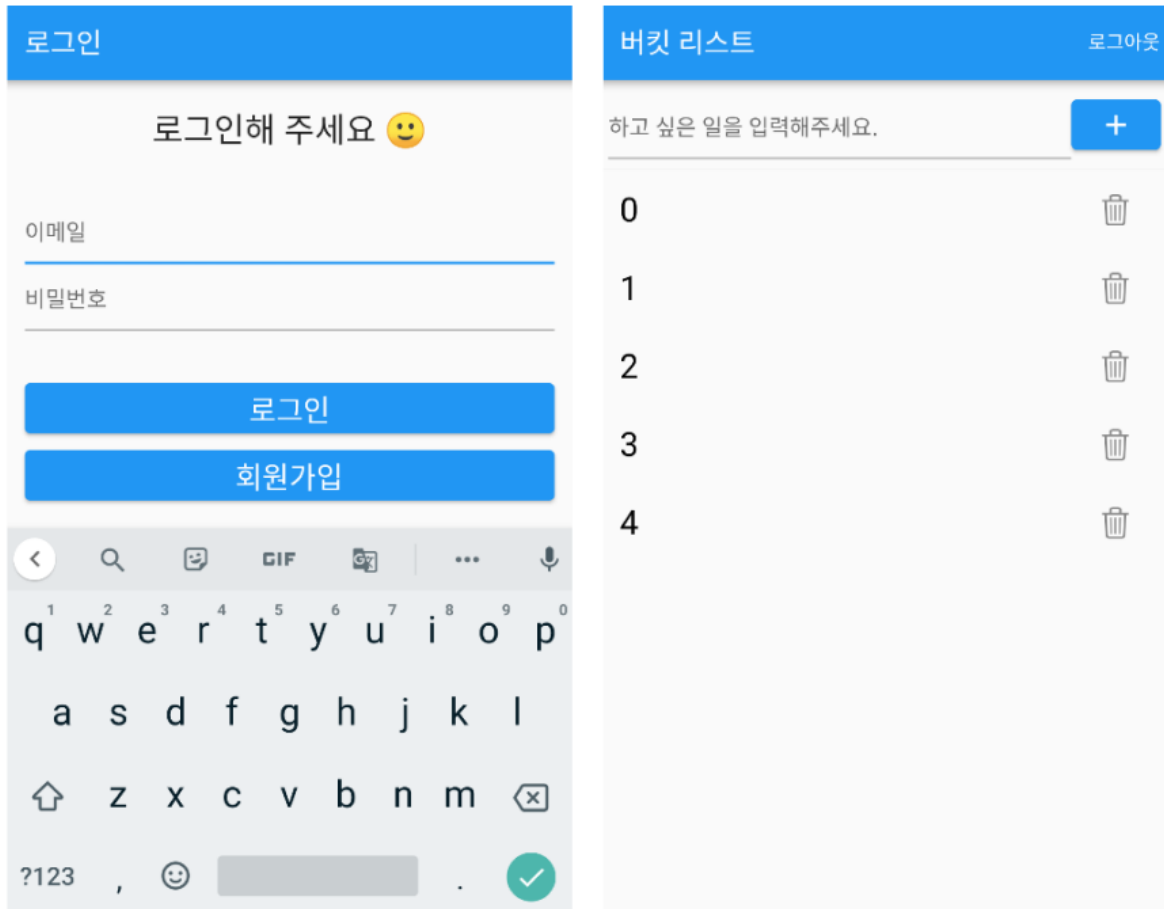
회원가입

04. 로그인 기능 만들기

▼ 완성본



데이터 접근 권한과 유저를 확인하기 위해 **로그인 기능**이 필요합니다.



💡 이메일 비밀번호를 이용한 회원 가입 / 로그인 / 로그아웃 기능을 구현해 보도록 하겠습니다.

💡 로그인은 다양한 방법으로 구현할 수 있습니다.

- 이메일 로그인
- 소셜 로그인 (구글, 카카오톡, 네이버, 애플 등)

참고로 iOS의 경우 소셜 로그인을 지원하려면 애플 로그인도 필수로 구현해야 출시가 가능합니다. (단, 이메일 로그인도 상관 없습니다)

▼ 1) Firebase Auth를 이용하는 이유



이메일 로그인 기능을 직접 구현하려면, 다음과 같은 과정이 필요합니다.

- 유저의 정보를 저장해 둘 데이터베이스 준비
- 유효한 이메일인지 확인하기 위한 이메일 인증번호 발송 기능
- 비밀번호를 잊은 경우를 위한 임시 비밀번호 발급 기능
- 이메일 중복 확인 기능 등등..

직접 구현하려면 다소 복잡한 기능들이 Firebase Authentication에서는 이미 구현되어있고 서버도 직접 구축할 필요가 없습니다. 심지어 무료로 사용할 수 있습니다.



Firebase Authentication의 자세한 사용 설명은 아래 링크를 참고해주세요.

▼ **[코드스니펫] firebase authentication 문서**

<https://firebase.google.com/docs/auth>

▼ **[코드스니펫] flutter & firebase authentication 문서**

<https://firebase.flutter.dev/docs/auth/usage/#emailpas>

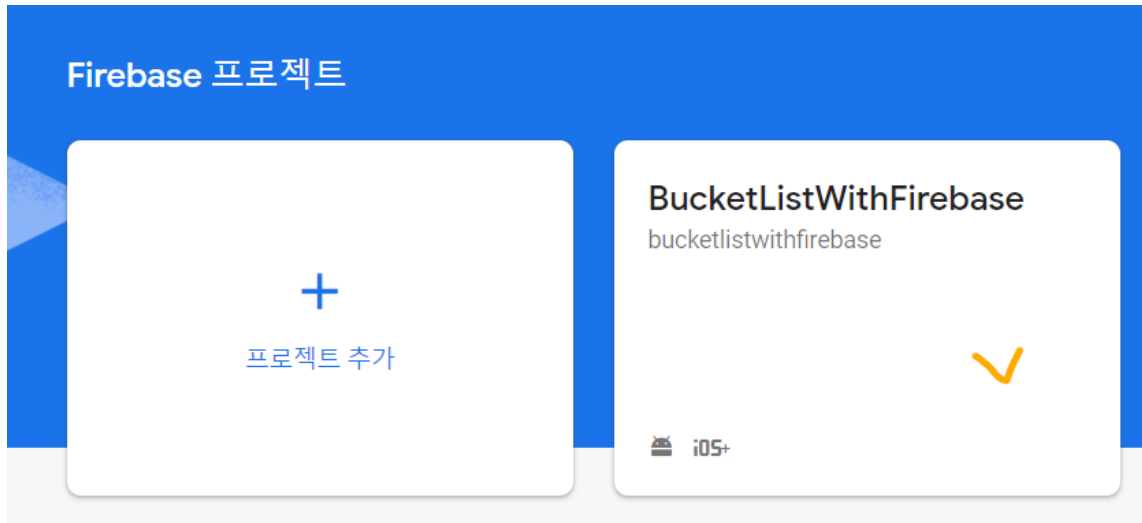
▼ 2) Firebase Auth 사용 준비

1. 코드스니펫을 복사해 Firebase Console에 접속해 주세요.

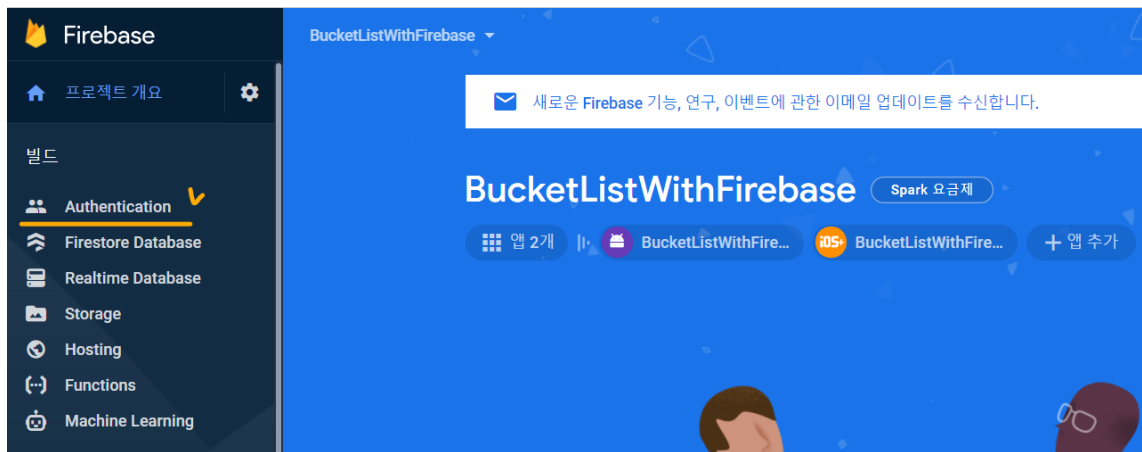
▼ **[코드스니펫] Firebase Console**

<https://console.firebase.google.com/?hl=ko>

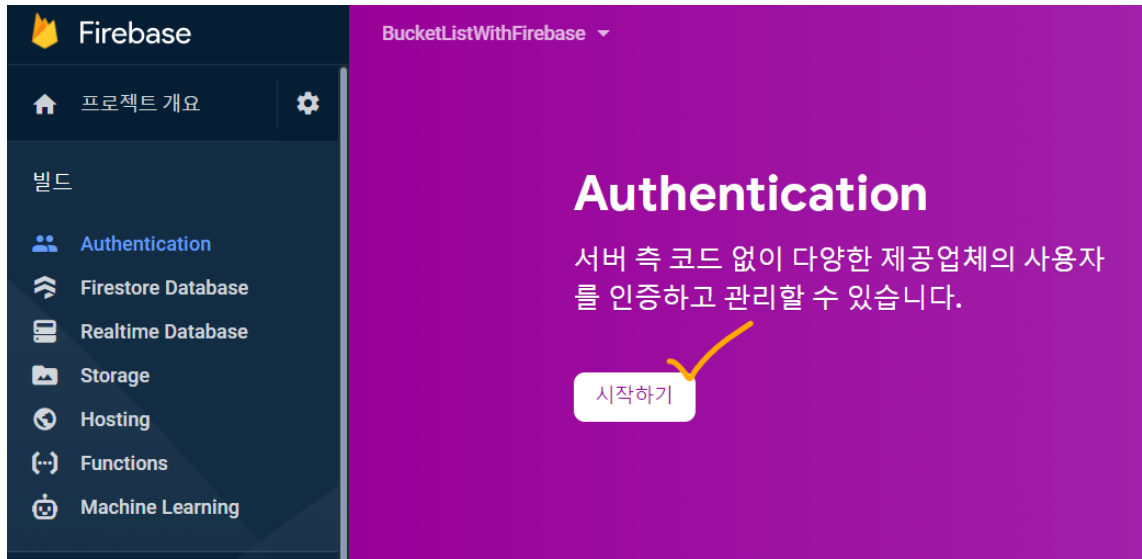
2. `BucketListWithFirebase` 프로젝트를 선택해 주세요.



3. **Authentication** 을 선택해 주세요.



4. **시작하기** 버튼을 선택해 주세요.



5. 이메일/비밀번호 를 이용한 로그인 방법을 선택해주세요.



Firebase에서는 다양한 로그인 방법을 제공합니다.

첫 번째 로그인 방법을 추가하여 Firebase 인증 시작하기

기본 제공업체

이메일/비밀번호

전화

익명

추가 제공업체

Google

Facebook

Play 게임

게임 센터

Apple

GitHub

Microsoft

Twitter

Yahoo

6. 이메일/비밀번호 사용 설정 스위치 버튼을 ON 상태로 변경해주시고 저장 버튼을 눌러주세요.

이메일/비밀번호

사용 설정

사용자가 자신의 이메일 주소와 비밀번호를 사용해 로그인할 수 있도록 허용합니다. Google SDK는 이메일 주소 확인, 비밀번호 복구, 이메일 주소 변경 기본 요소도 제공합니다. [자세히 알아보기](#)

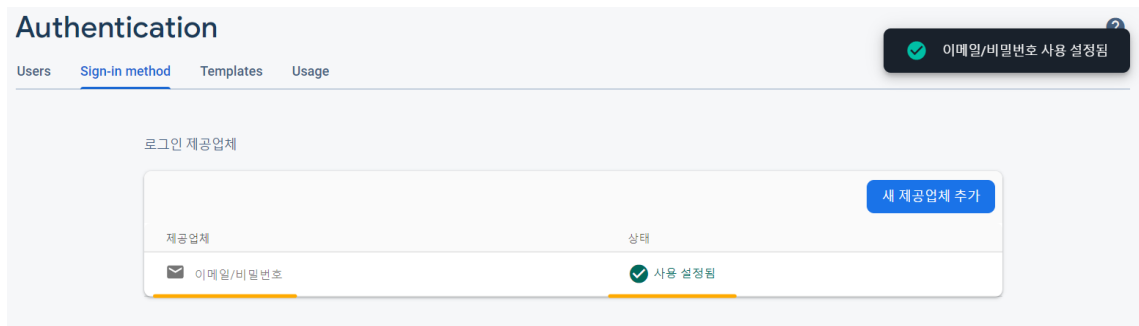
이메일 링크(비밀번호가 없는 로그인)

사용 설정

취소

저장

7. 이메일 비밀번호 로그인 사용 설정이 완료되었습니다.



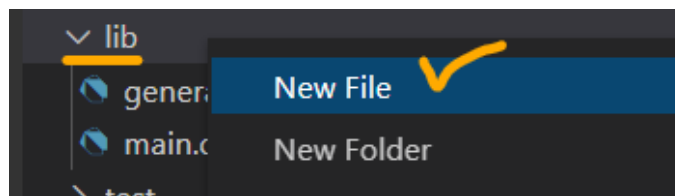
Firebase Auth 사용 준비가 끝났으니 VSCode에서 코드를 작성해 봅시다.

▼ 3) AuthService 만들기

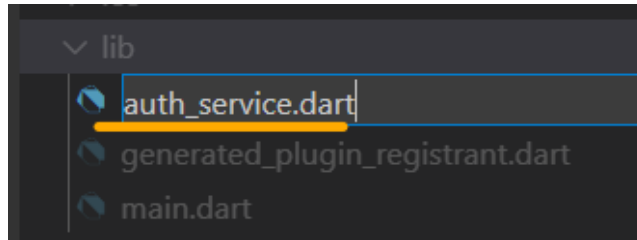


로그인 기능을 담당해주는 `AuthService` 를 만들도록 하겠습니다.

1. `lib` 폴더를 우클릭 한 뒤 `New File` 을 선택해 주세요.



2. `auth_service.dart` 라고 이름을 지어주세요.



3. 코드스니펫을 복사해 `auth_service.dart` 파일에 붙여 넣어주세요.

▼ [코드스니펫] `auth_service.dart` / 시작

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

class AuthService extends ChangeNotifier {
  User? currentUser() {
    // 현재 유저(로그인 되지 않은 경우 null 반환)
  }

  void signUp({
    required String email, // 이메일
    required String password, // 비밀번호
    required Function() onSuccess, // 가입 성공시 호출되는 함수
    required Function(String err) onError, // 에러 발생시 호출되는 함수
  }) async {
    // 회원가입
  }

  void signIn({
    required String email, // 이메일
    required String password, // 비밀번호
    required Function() onSuccess, // 로그인 성공시 호출되는 함수
    required Function(String err) onError, // 에러 발생시 호출되는 함수
  }) async {
    // 로그인
  }

  void signOut() async {
    // 로그아웃
  }
}
```

```
}
}
```



`ChangeNotifier`의 기능을 상속 받은 클래스에서는 `notifyListeners()` 함수를 호출하여 화면을 새로고침 할 수 있습니다.

```
1  import 'package:firebase_auth/firebase_auth.dart';
2  import 'package:flutter/material.dart';
3                                     인증 상태 변경시, 화면 새로 고침을 위해 상속
4  class AuthService extends ChangeNotifier {
5    User? currentUser() {
6      // 현재 유저(로그인 되지 않은 경우 null 반환)
7    }
8
9    void signUp({
10      required String email, // 이메일
11      required String password, // 비밀번호
12      required Function onSuccess, // 가입 성공시 호출되는 함수
13      required Function(String err) onError, // 에러 발생시 호출되는 함수
14    }) async {
15      // 회원가입
16    }
17
18    void signIn({
19      required String email, // 이메일
20      required String password, // 비밀번호
21      required Function onSuccess, // 로그인 성공시 호출되는 함수
22      required Function(String err) onError, // 에러 발생시 호출되는 함수
23    }) async {
24      // 로그인
25    }
26
27    void signOut() async {
28      // 로그아웃
29    }
30  }
31
```




구현할 함수 목록

`currentUser` : 현재 유저 조회(로그인을 하지 않은 경우 null을 반환)

`signUp` : 회원가입

`signIn` : 로그인

`signOut` : 로그아웃



각 함수들은 모두 Firebase Auth 서버와 통신을 해야하기 때문에 실행하는데 시간이 소요되는 비동기 코드로 `async` 키워드를 미리 넣어 두었습니다.

`signUp` 과 `signIn` 함수는 이름 지정 매개 변수(named parameter)로 정의하였습니다. 자세한 사용 방법은 코드스니펫을 복사한 뒤 주소창에 붙여 넣어, DartPad로 접속한 뒤 확인해 보시다.

▼ [코드스니펫] DartPad SignUp 함수 샘플

<https://dartpad.dev/?id=913c464db29d6be5d5b0586dcb2b79ea>

4. AuthService 파일을 Provider를 이용하여 위젯 트리의 최상단에 넣어주도록 합시다.

코드스니펫을 복사해 `main.dart` 파일에 8번째 줄을 지우고 붙여 넣어주세요.

▼ [코드스니펫] MultiProvider 설정

```

runApp(
  MultiProvider(
    providers: [
      ChangeNotifierProvider(create: (context) => AuthS
    ],
    child: const MyApp(),
  ),
);

```

```

4
Run | Debug | Profile
5 void main() async {
6   WidgetsFlutterBinding.ensureInitialized(); // main 함수에서 async 사용하
7   await Firebase.initializeApp(); // firebase 앱 시작
8   runApp(
9     MultiProvider(
10      providers: [
11        ChangeNotifierProvider(create: (context) => AuthService()),
12      ],
13      child: const MyApp(),
14    ),
15  );
16 }
17

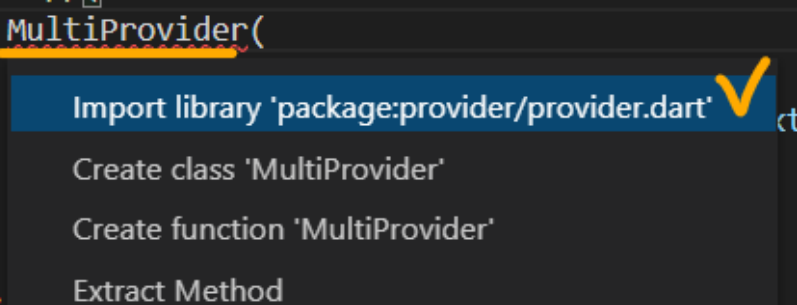
```

5. 9번째 줄에 `MultiProvider` 를 클릭한 뒤 Quick Fix(`Ctrl/Cmd + .`)를 눌러 provider 패키지를 Import 해주세요.

```

7   await Firebase.initializeApp(); // firebase
8   runApp(
9     MultiProvider(
10
11
12
13
14
15

```



아래 사진과 같이 provider가 Import 되고, 10번째 줄에 에러가 사라졌습니다.

```

1 import 'package:firebase_core/firebase_core.dart';
2 import 'package:flutter/cupertino.dart';
3 import 'package:flutter/material.dart';
4 import 'package:provider/provider.dart';
5
6 void main() async {
7   WidgetsFlutterBinding.ensureInitialized(); // main 함수에서 async 사용
8   await Firebase.initializeApp(); // firebase 앱 시작
9   runApp(
10     MultiProvider(
11       providers: [
12         ChangeNotifierProvider(create: (context) => AuthService()),
13       ],
14       child: const MyApp(),
15     ), // MultiProvider
16   );
17 }
18

```

6. 12번째 줄에 `AuthService`를 클릭한 뒤 Quick Fix(`Ctrl/Cmd + .`)를 누른 뒤 `auth_service`를 Import 해줍니다.

```

11 providers: [
12   ChangeNotifierProvider(create: (context) => AuthService()),
13 ]
14
15 Import library 'auth_service.dart' ✓
16 Import library 'package:bucket_list_with_firebase/auth_service.dart'
17 Create class 'AuthService'
18 Create function 'AuthService'
19 Extract Method
20 Extract Local Variable
21
22 @override

```

아래 사진과 같이 `auth_service`가 Import 되었고 에러가 해결되었습니다.

```

1  import 'package:firebase_core/firebase_core.dart';
2  import 'package:flutter/cupertino.dart';
3  import 'package:flutter/material.dart';
4  import 'package:provider/provider.dart';
5
6  import 'auth_service.dart';
7
8  Run | Debug | Profile
9  void main() async {
10     WidgetsFlutterBinding.ensureInitialized(); // main 함수에서 async 사용
11     await Firebase.initializeApp(); // firebase 앱 시작
12     runApp(
13       MultiProvider(
14         providers: [
15           ChangeNotifierProvider(create: (context) => AuthService()),
16         ],
17         child: const MyApp(),
18       ), // MultiProvider
19     );
20 }

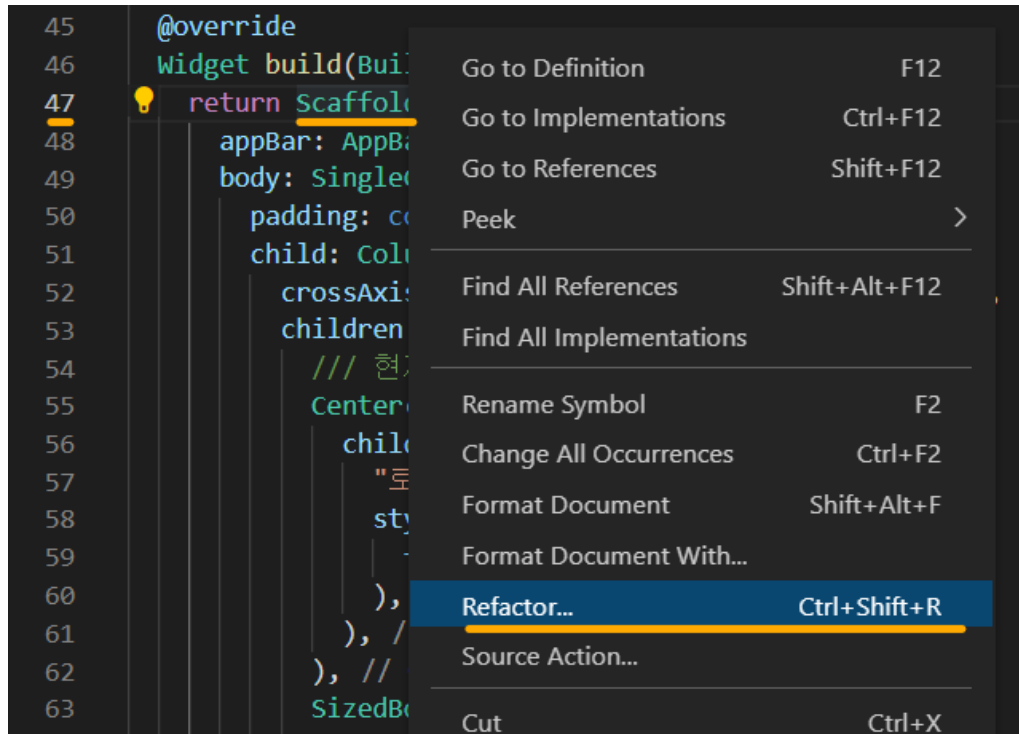
```

이제 AuthService를 어디서든 접근할 수 있습니다.

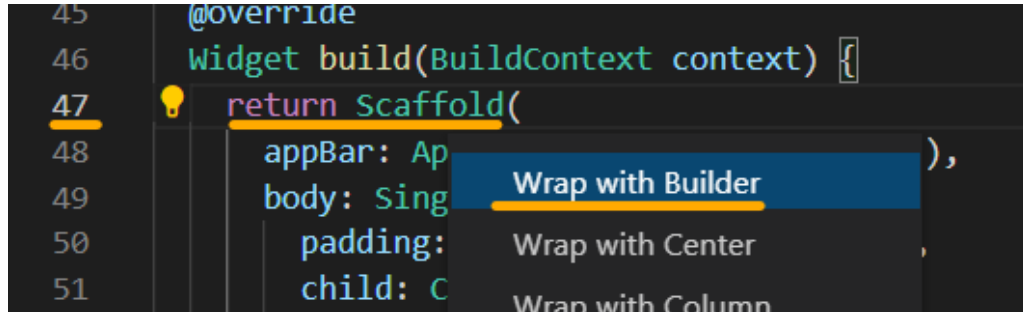
▼ 4) LoginPage에서 AuthService 접근하기

1. AuthService의 인증 상태에 따라 LoginPage가 갱신되도록 **Consumer** 로 감싸 주겠습니다.

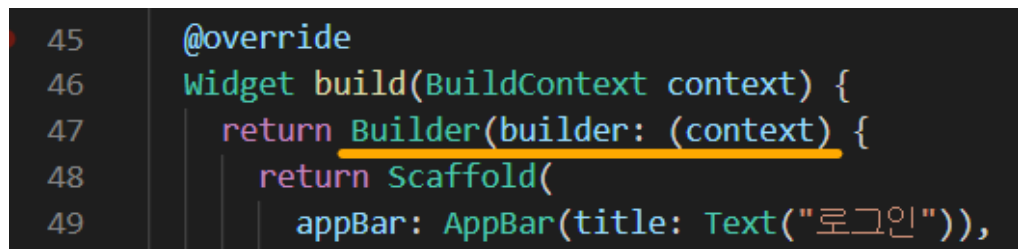
47번째 줄에 Scaffold를 선택한 뒤 마우스 우클릭을 눌러 **Refactor** 를 선택해 주세요.



`Wrap with Builder` 를 선택해 주세요.



그리고 저장을 해주면 아래와 같이 Scaffold 위젯이 Builder로 감싸집니다.



코드스니펫을 복사해 47번째 라인을 지우고 붙여 넣어주세요.

▼ [코드스니펫] AuthService Consumer

```
return Consumer<AuthService>(builder: (context, authServi
```

```
45  @override
46  ✓  Widget build(BuildContext context) {
47  ✓    return Consumer<AuthService>(builder: (context, authService, child) {
48  ✓      return Scaffold(
49        appBar: AppBar(title: Text("로그인")),
50  ✓      body: SingleChildScrollView(
```

줄 정렬을 예쁘게 하기 위해 아래 사진과 같이 104번째 줄에 중괄호(`}`)와 소괄호(`)`) 사이에 콤마(`,`)를 넣어주세요.

```
101      ), // Column
102    ), // SingleChildScro
103  ); // Scaffold
104  },); // Consumer
105  }
106  }
107
```

그리고 저장해주면 47번째 라인의 builder 함수가 아래와 같이 28번째 줄로 정렬됩니다.

```
45  @override
46  ✓  Widget build(BuildContext context) {
47  ✓  ( return Consumer<AuthService>(
48  ✓    builder: (context, authService, child) {
49  ✓      return Scaffold(
50        appBar: AppBar(title: Text("로그인")),
51  ✓      body: SingleChildScrollView(
52        padding: const EdgeInsets.all(16),
```

에뮬레이터에서 에러가 발생하는 경우 `Restart` 버튼을 누르면 해결됩니다.



이제 AuthService에서 `notifyListeners()` 를 호출하면 Consumer 의 builder 함수가 실행되며 화면이 갱신 됩니다.

▼ 5) 회원 가입 만들기



기능을 구현할 때, 기능의 시작 지점인 트리거(trigger) 부터 찾으면 좋습니다.

회원 가입 기능은,

`회원 가입` 버튼을 누르는 순간 시작되므로 96번째 라인의 `onPressed` 함수부터 시작됩니다.



회원이 가입 버튼을 클릭하는 경우 구현해야할 로직 순서는 다음과 같습니다.

1. 회원가입 버튼 클릭
2. 사용자가 입력한 이메일과 비밀번호 가져오기
3. AuthService에 signUp 함수로 전달



TextField에 아래 사진과 같이 `TextEditingController` 가 연결 되어 있습니다.

```
41 class _LoginPageState extends State<LoginPage> {  
42   TextEditingController emailController = TextEditingController();  
43   TextEditingController passwordController = TextEditingController();  
44 }
```

```
65  
66   /// 이메일  
67   TextField(  
68     controller: emailController,  
69     decoration: InputDecoration(hintText: "이메일"),  
70   ), // TextField  
71  
72   /// 비밀번호  
73   TextField(  
74     controller: passwordController,  
75     obscureText: false, // 비밀번호 안보이게  
76     decoration: InputDecoration(hintText: "비밀번호"),  
77   ), // TextField
```



TextEditingController를 이용해서 텍스트 값을 아래와 같이 가져올 수 있습니다.

```
String email = emailController.text; // 이메일 가져오기
String password = passwordController.text; // 비밀번호 가져오기
```

1. 회원가입 버튼을 누르면 AuthService의 SignUp 함수를 호출해 보도록 하겠습니다.

코드스니펫을 복사해 `main.dart` 파일 98번째 줄(`print("sign up")`)를 지우고 붙여 넣어주세요.

▼ [코드스니펫] `main.dart` / `signIn onPressed`

```
authService.signUp(
  email: emailController.text,
  password: passwordController.text,
  onSuccess: () {
    // 회원가입 성공
    print("회원가입 성공");
  },
  onError: (err) {
    // 에러 발생
    print("회원가입 실패 : $err");
  },
);
```



```

93      /// 회원가입 버튼
94      ElevatedButton(
95        child: Text("회원가입", style: TextStyle(fontSize: 21)),
96        onPressed: () {
97          // 회원가입
98          authService.signUp(
99            email: emailController.text,
100            password: passwordController.text,
101            onSuccess: () {
102              // 회원가입 성공
103              print("회원가입 성공");
104            },
105            onError: (err) {
106              // 에러 발생
107              print("회원가입 실패 : $err");
108            },
109          );
110        },
111      ), // ElevatedButton

```

`auth_service.dart` 파일에 `signUp` 함수를 구현해 보도록 하겠습니다.

2. 사용자에게 어떤 값을 입력 받을 때에는 항상 값을 올바르게 입력했는지 확인하는 **유효성 검사**를 해야 합니다. 코드스니펫을 복사해 `auth_service.dart` 파일 15번째 줄 맨 뒤에 붙여 넣어주세요.

▼ [코드스니펫] `auth_service.dart` / `signUp` 유효성 검사

```

      // 이메일 및 비밀번호 입력 여부 확인
      if (email.isEmpty) {
        onError("이메일을 입력해 주세요.");
        return;
      } else if (password.isEmpty) {
        onError("비밀번호를 입력해 주세요.");
        return;
      }

```



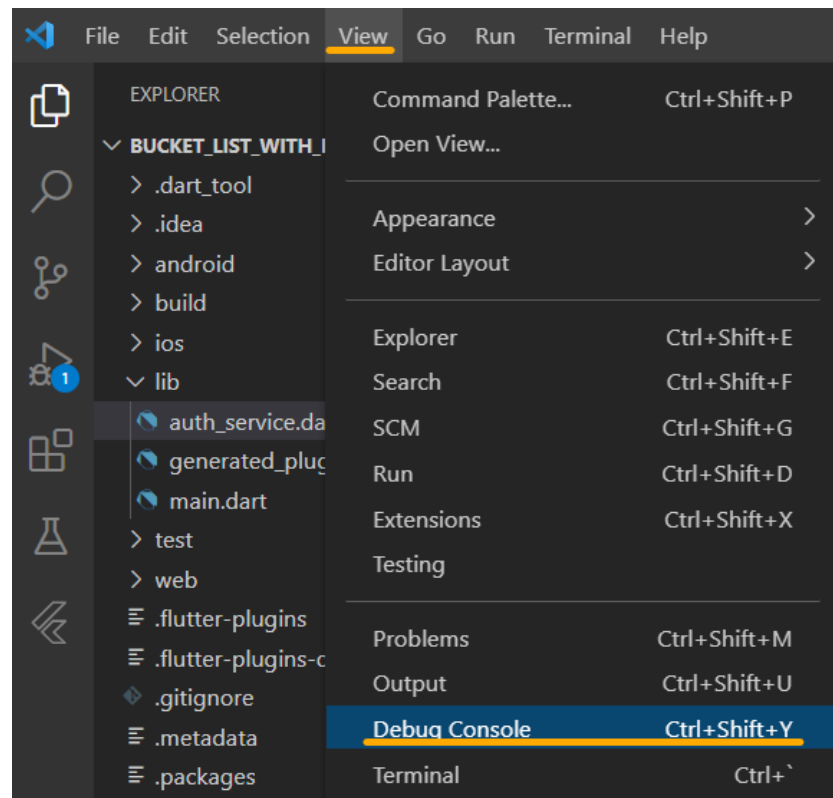
이메일 또는 비밀번호를 입력하지 않은 경우, `onError` 함수를 호출하고 `signUp` 함수를 종료하도록 `return;` 을 호출하였습니다.

```

9  void signUp({
10     required String email, // 이메일
11     required String password, // 비밀번호
12     required Function onSuccess, // 가입 성공시 호출되는 함수
13     required Function(String err) onError, // 에러 발생시 호출되는 함수
14 }) async {
15     // 회원가입
16     // 이메일 및 비밀번호 입력 여부 확인
17     if (email.isEmpty) {
18         onError("이메일을 입력해 주세요.");
19         return;
20     } else if (password.isEmpty) {
21         onError("비밀번호를 입력해 주세요.");
22         return;
23     }
24 }

```

저장한 뒤, 올바르게 작동하는지 확인해 봅시다. **View** → **Debug Console** 을 선택해주세요.



에뮬레이터에서 이메일 또는 비밀번호를 입력하지 않은 상태에서 회원가입 버튼을 눌러주세요. 아래와 같이 **main.dart** 의 105번째 줄 **onError** 함수가 잘 호출되고, 콘솔창에 에러 메시지가 잘 뜨는 것을 확인할 수 있습니다.

```

93      /// 회원가입 버튼
94      ElevatedButton(
95        child: Text("회원가입", style: TextStyle(fontSize: 21)),
96        onPressed: () {
97          /// 회원가입
98          authService.signUp(
99            email: emailController.text,
100            password: passwordController.text,
101            onSuccess: () {
102              /// 회원가입 성공
103              print("회원가입 성공");
104            },
105            onError: (err) {
106              /// 에러 발생
107              print("회원가입 실패 : $err");
108            },
109          );
110        },

```

PROBLEMS 6 OUTPUT DEBUG CONSOLE TERMINAL Filter (e.g. text, !exclude)

```

I/flutter ( 7230): 회원가입 실패 : 이메일을 입력해 주세요.
I/flutter ( 7230): 회원가입 실패 : 비밀번호를 입력해 주세요.

```



사용자는 콘솔창을 볼 수 없으니, Snackbar 위젯을 사용하여 화면에 보여주도록 하겠습니다.

코드스니펫을 복사해 `main.dart` 파일 107번째 줄을 지우고 붙여 넣어주세요.

▼ [코드스니펫] main.dart / signUp onError

```

ScaffoldMessenger.of(context).showSnackBar(SnackBar(
  content: Text(err),
));

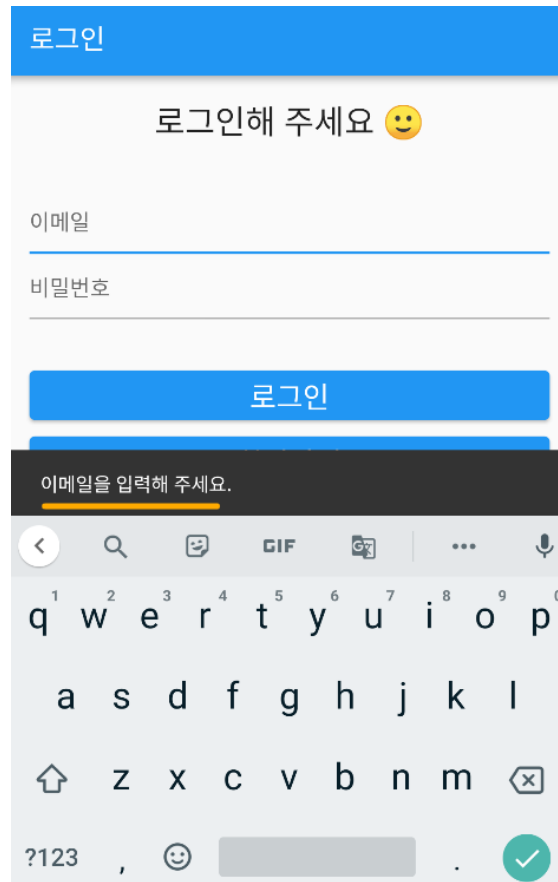
```

```

105      onError: (err) {
106        /// 에러 발생
107        ScaffoldMessenger.of(context).showSnackBar(SnackBar(
108          content: Text(err),
109        )); // Snackbar
110      },
111    );
112  },
113 ), // ElevatedButton

```

그리고 다시 테스트를 해보면 아래 이미지와 같이 화면 하단에 SnackBar 위젯이 뜨는 것을 확인할 수 있습니다.



회원가입을 성공한 경우에도 SnackBar 위젯을 보여주도록 합시다. 코드스니펫을 복사해 `main.dart` 103번째 줄을 지우고 붙여 넣어주세요.

▼ [코드스니펫] `main.dart` / `signUp onSuccess`

```
ScaffoldMessenger.of(context).showSnackBar(SnackBar(  
    content: Text("회원가입 성공"),  
));
```

```

97 // 회원가입
98 authService.signUp(
99   email: emailController.text,
100   password: passwordController.text,
101   onSuccess: () {
102     // 회원가입 성공
103     ScaffoldMessenger.of(context).showSnackBar(SnackBar(
104       content: Text("회원가입 성공"),
105     )); // SnackBar
106   },
107   onError: (err) {
108     // 에러 발생
109     ScaffoldMessenger.of(context).showSnackBar(SnackBar(
110       content: Text(err),
111     )); // SnackBar
112   },
113 );

```

3. 이메일과 비밀번호 유효성 검사를 끝냈으니 Firebase Auth 패키지를 이용하여 회원 가입을 해 보도록 합시다. 코드스니펫을 복사해 새 탭에서 열면 firebase auth 패키지 사용법을 볼 수 있습니다.

▼ [코드스니펫] Flutter & Firebase Auth 이메일 가입 문서

<https://firebase.flutter.dev/docs/auth/usage/#emailpassword>

Registration

To create a new account on your Firebase project call the `createUserWithEmailAndPassword()` method with the user's email address and password:

```

try {
  UserCredential userCredential = await FirebaseAuth.instance.createUserWithEmailAndPassword(
    email: "barry.allen@example.com",
    password: "SuperSecretPassword!"
  );
} on FirebaseAuthException catch (e) {
  if (e.code == 'weak-password') {
    print('The password provided is too weak.');
```



위 코드를 하나씩 이해해 보도록 하겠습니다. 먼저 전체 로직이 아래와 같이 아직 배우지 않은 Dart 문법인 `try` / `on` / `catch` 로 감싸져 있는 이유부터 알아봅시다.

```
try {  
  
    Firebase Auth 서버로 회원가입 요청  
  
} on FirebaseAuthException catch (e) {  
  
  
  
  
  
  
  
  
  
} catch (e) {  
  
  
  
  
  
  
  
  
  
}
```

`try` 안에서 Firebase Auth 서버에 회원가입을 HTTP 통신을 통해 요청하는데, 요청에 대한 결과를 아래와 같이 세 타입으로 나눌 수 있습니다.

1. 회원가입 성공

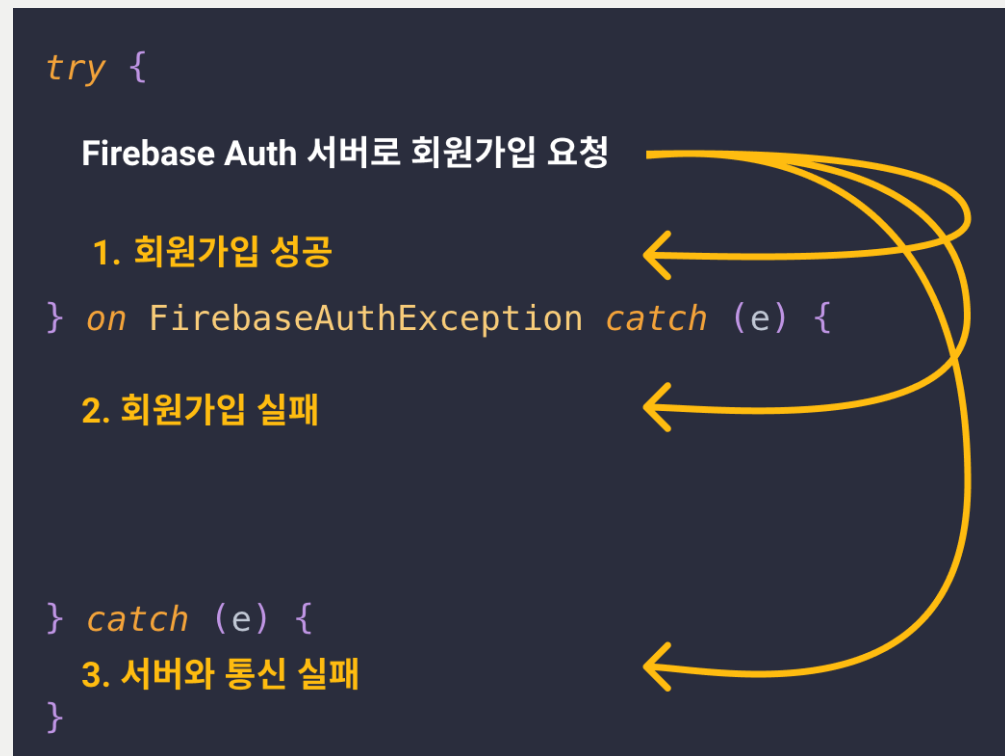
2. 회원가입 실패

- 이메일이 유효하지 않은 경우
- 이메일이 중복된 경우
- 기타 등등..

3. 서버와 통신 실패

- 인터넷이 연결이 안된 경우
- Firebase Auth 서버가 정상 작동중이지 않은 경우
- 기타 등등..

위와 같은 각 상황에 맞춰서 대처하기 위해 사용하는 문법이 바로 `try / on / catch` 문법을 사용합니다.



`try` 내부에서 먼저 요청을 보낸 뒤 아래와 같이 동작합니다.

- 성공하는 경우, 바로 다음 줄 `1. 회원가입 성공` 이 실행되고 로직 종료
- 통신은 성공했지만 이메일 중복 등의 이유로 실패하는 경우, `on` 영역의 `2. 회원가입 실패` 가 실행되고 로직 종료
- 그 이외의 기타 이유로 실패하는 경우, `catch` 영역의 `3. 서버와 통신 실패` 가 실행되고 로직 종료

▼ Try & On & Catch에 대해서 조금 더 자세히 알아보기



DartPad에서 에러를 다루는 방법에 대해서 좀 더 자세히 배워보겠습니다. 코. 스니펫을 복사한 뒤 주소창에 붙여 넣으면, DartPad로 접속합니다.

▼ [코드스니펫] DartPad Dart Exception 학습

<https://dartpad.dev/?id=ccabe730879da20201160653>

```
1 main() {
2   print(5 ~/ 3); // 5를 3으로 나눈 몫을 구함
3
4   int x = 12;
5   int y = 0;
6
7   print("start");
8   int res = x ~/ y; // 12를 0으로 나누면 무한대라서 에러 발생
9   print("finish"); // 윗 줄에서 에러가 발생해 9번째 줄은 실행되지 않고 멈춤
10 }
11
```

▶ Run

Console

```
1
start
Uncaught Error: Uns
```

8번째 줄과 같이 에러가 발생하는 경우 다음 줄이 실행되지 않고 멈춥니다.



에러가 발생할 것이라 예상되는 로직을 `try` / `catch` 구문으로 감싸면 에러가 생하는 경우, 멈추지 않고 다음 로직을 실행할 수 있습니다. 아래 코드스니펫을 복사해서 새 탭에서 열어주세요.

▼ [코드스니펫] DartPad Dart try & catch 학습

<https://dartpad.dev/?id=6f6962fa98bee17f13029891>

```
1 main() {
2   int x = 12;
3   int y = 0;
4
5   try {
6     print("1. start");
7     int res = x ~/ y; // y = 0인 경우 에러 발생
8
9     print("2. finish"); // 에러가 없는 경우 실행
10  } catch (e) {
11    print("3. 에러 발생 : ${e.runtimeType}"); // 에러는 발생시 실행
12  }
13 }
14
```

Run

Console

```
1. start
3. 에러 발생 : UnsupportedError
```

7번째 줄에서 에러가 발생했지만 멈추지 않고 11번째 로직이 실행됩니다. 만약 7번째 줄에 `y` 값을 1로 바꾸면 9번째 줄이 실행됩니다.

```
1 main() {
2   int x = 12;
3   int y = 1;
4
5   try {
6     print("1. start");
7     int res = x ~/ y; // y = 0인 경우 에러 발생
8
9     print("2. finish"); // 에러가 없는 경우 실행
10  } catch (e) {
11    print("3. 에러 발생 : ${e.runtimeType}"); // 에러는 발생시 실행
12  }
13 }
14
```

Run

Console

```
1. start
2. finish
```



`try` / `on` / `catch` 로직을 이용하면 에러의 타입에 따라 실행되는 로직을 나눌 수 있습니다.

▼ [코드스니펫] DartPad Dart try & on & catch 학습

<https://dartpad.dev/?id=fd05bb13f412f5cbb069ee15>

```

1 main() {
2   int x = 12;
3   int y = 0;
4
5   try {
6     print("1. start");
7     int res = x ~/ y; // UnsupportedError 에러 발생
8
9     print("2. finish"); // 에러가 없는 경우 실행
10    } on UnsupportedError catch (e) {
11      print("3. UnsupportedError 에러 : ${e.runtimeType}"); // UnsupportedError 담당
12    }
13    } catch (e) {
14      print("4. 기타 에러 : ${e.runtimeType}"); // 기타 에러
15    }
16  }
17

```

7번째 줄에서 발생하는 에러는 `UnsupportedError` 타입입니다. 해당 타입의 에러만 담당하는 로직을 작성하고 싶은 경우 10번째 줄과 같이 `on UnsupportedError` 라고 적어주면 됩니다.

만약 `UnsupportedError` 타입이 아닌 에러가 발생하는 경우에는 `catch` 로직이 행됩니다. 아래 코드스니펫을 복사해서 새 탭에서 열어주세요.

▼ [코드스니펫] DartPad Dart try & on & catch 학습2

<https://dartpad.dev/?id=1f1e1a16f764ec917aca662c>

```

1 main() {
2   try {
3     print("1. start");
4     assert(false); // AssertionError 발생
5
6     print("2. finish"); // 에러가 없는 경우 실행
7   } on UnsupportedError catch (e) {
8     print("3. UnsupportedError 에러 : ${e.runtimeType}"); // UnsupportedError 담당
9   }
10  } catch (e) {
11    print("4. 기타 에러 : ${e.runtimeType}"); // 기타 에러
12  }
13 }
14

```

4번째 줄에서 발생하는 에러는 `AssertionError` 입니다. 따라서 `on UnsupportedError` 에 잡히지 않고, 11번째 줄의 `catch` 로직이 실행되는 것을 볼 수 있습니다.



이제 회원가입을 구현해 보도록 합시다. 회원가입 문서의 코드를 참고해 아래 코드스니펫을 준비하였습니다. 코드스니펫을 복사한 뒤 `auth_service.dart` 파일 23번째 줄 맨 뒤에 붙여 넣어주세요.

▼ [코드스니펫] `auth_service.dart` / `signUp`

```
        // firebase auth 회원 가입
    try {
        await FirebaseAuth.instance.createUserWithEmailAndPassword(
            email: email,
            password: password,
        );

        // 성공 함수 호출
        onSuccess();
    } on FirebaseAuthException catch (e) {
        // Firebase auth 에러 발생
        onError(e.message!);
    } catch (e) {
        // Firebase auth 이외의 에러 발생
        onError(e.toString());
    }
}
```

```

15 // 회원가입
16 // 이메일 및 비밀번호 입력 여부 확인
17 if (email.isEmpty) {
18   onError("이메일을 입력해 주세요.");
19   return;
20 } else if (password.isEmpty) {
21   onError("비밀번호를 입력해 주세요.");
22   return;
23 }
24
25 // firebase auth 회원 가입
26 try {
27   await FirebaseAuth.instance.createUserWithEmailAndPassword(
28     email: email,
29     password: password,
30   );
31
32   // 성공 함수 호출
33   onSuccess();
34 } on FirebaseAuthException catch (e) {
35   // Firebase auth 에러 발생
36   onError(e.message!);
37 } catch (e) {
38   // Firebase auth 이외의 에러 발생
39   onError(e.toString());
40 }
41 }
42

```



27번째 줄에서 email과 password로 FirebaseAuth를 이용하여 이메일 회원 가입을 시도합니다.

```

await FirebaseAuth.instance.createUserWithEmailAndPassword(
  email: email,
  password: password,
);

```



이메일 중복과 같은 `FirebaseAuthException` 이 발생하면 34번째 줄에서 잡히고 인터넷이 연결되지 않은 경우와 같이 다른 에러는 37번째 줄로 넘어갑니다.



36번째 줄 `onError(e.message!);` 를 아래 코드로 변경하면 한국어로 에러를 보여줄 수도 있습니다.

```
if (e.code == 'weak-password') {
  onError('비밀번호를 6자리 이상 입력해 주세요.');
```

```
} else if (e.code == 'email-already-in-use') {
  onError('이미 가입된 이메일 입니다.');
```

```
} else if (e.code == 'invalid-email') {
  onError('이메일 형식을 확인해주세요.');
```

```
} else if (e.code == 'user-not-found') {
  onError('일치하는 이메일이 없습니다.');
```

```
} else if (e.code == 'wrong-password') {
  onError('비밀번호가 일치하지 않습니다.');
```

```
} else {
  onError(e.message!);
}
```

4. 작동을 잘 하는지 테스트를 진행하도록 하겠습니다.

- 이메일 형식 테스트

The screenshot shows a mobile app login screen. At the top is a blue header with the text '로그인' (Login). Below the header, the text '로그인해 주세요 😊' (Please login 😊) is displayed. There are two input fields: the first one contains the letter 'a', and the second one contains 'a' followed by a cursor. Below the input fields is a blue button labeled '로그인' (Login). At the bottom of the screen, a dark gray banner displays the error message 'The email address is badly formatted.' in white text.

- 비밀번호 형식 테스트

로그인

로그인해 주세요 😊

a@a.com

a

로그인

Password should be at least 6 characters

- 가입 성공

로그인

로그인해 주세요 😊

a@a.com

123456

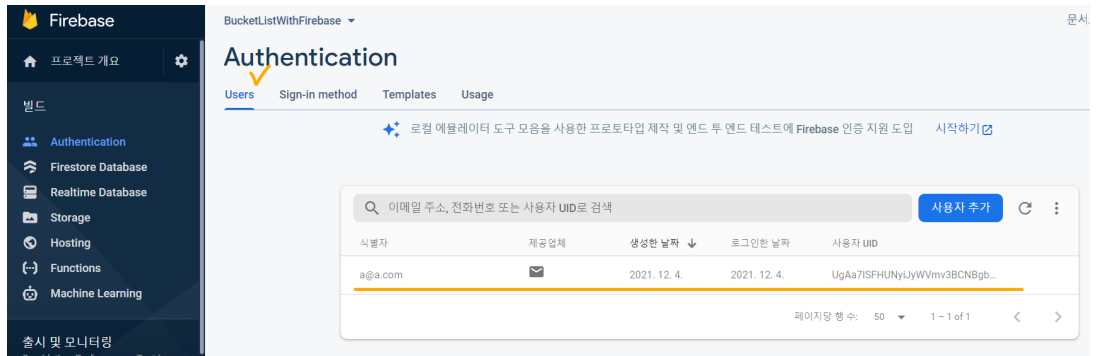
로그인

회원가입 성공

firebase console에서 Authentication의 **Users** 탭을 선택하면 아래와 같이 가입된 완료된 유저가 보입니다.



사용자 uid는 Firebase에서 만들어준 유저의 고유 식별 번호입니다.



💡 회원 가입을 완료하였으니, 이제 로그인 기능을 구현해 보도록 하겠습니다.

▼ 6) 로그인 만들기

💡 로그인 버튼을 누르면 실행되는 84번째 라인을 보면 지금은 바로 HomePage로 이동하는데, 이 부분을 authService의 `signIn` 함수를 호출하도록 변경하겠습니다.

```

80
81
82  ✓   /// 로그인 버튼
83  ElevatedButton(
84  ✓   child: Text("로그인", style: TextStyle(fontSize: 21)),
85  ✓   onPressed: () {
86  ✓       // 로그인 성공시 HomePage로 이동
87       Navigator.pushReplacement(
88         context,
89         MaterialPageRoute(builder: (_) => HomePage()),
90       );
91     }, // ElevatedButton

```

1. 코드스니펫을 복사해 `main.dart` 85 ~ 89 라인을 지우고 붙여 넣어주세요.

▼ [코드스니펫] main.dart / signIn onPressed

```

// 로그인

authService.signIn(
  email: emailController.text,
  password: passwordController.text,
  onSuccess: () {
    // 로그인 성공

```

```

        ScaffoldMessenger.of(context).showSnackBar(
          content: Text("로그인 성공"),
        ));
      },
      onError: (err) {
        // 에러 발생
        ScaffoldMessenger.of(context).showSnackBar(
          content: Text(err),
        ));
      },
    );
  );
}

```



`authService.signIn()` 를 호출하는 부분을 제외하고 회원 가입 만들 때 사용한 코드와 동일합니다.

```

81 // 로그인 버튼
82 ElevatedButton(
83   child: Text("로그인", style: TextStyle(fontSize: 21)),
84   onPressed: () {
85     // 로그인
86     authService.signIn(
87       email: emailController.text,
88       password: passwordController.text,
89       onSuccess: () {
90         // 로그인 성공
91         ScaffoldMessenger.of(context).showSnackBar(SnackBar(
92           content: Text("로그인 성공"),
93         )); // SnackBar
94       },
95       onError: (err) {
96         // 에러 발생
97         ScaffoldMessenger.of(context).showSnackBar(SnackBar(
98           content: Text(err),
99         )); // SnackBar
100      },
101    );
102  },
103 ), // ElevatedButton

```

2. `auth_service.dart` 파일로 이동해 `signIn` 함수를 구현해 보도록 하겠습니다.

먼저 코드스니펫을 복사해 새 탭에서 열어 sign in 관련 공식 문서를 보도록 하겠습니다.

▼ [코드스니펫] Flutter & Firebase Auth / SignIn 문서

<https://firebase.flutter.dev/docs/auth/usage/#sign-in>

Sign-in

To sign-in to an existing account, call the `signInWithEmailAndPassword()` method:

```
try {
  UserCredential userCredential = await FirebaseAuth.instance.signInWithEmailAndPassword(
    email: "barry.allen@example.com",
    password: "SuperSecretPassword!"
  );
} on FirebaseAuthException catch (e) {
  if (e.code == 'user-not-found') {
    print('No user found for that email.');
```

코드스니펫을 복사해 `auth_service.dart` 파일의 49번째 라인 뒤에 붙여 넣어주세요.

▼ [코드스니펫] `auth_service.dart` / `signIn`

```
if (email.isEmpty) {
  onError('이메일을 입력해주세요.');
```

```
// 로그인 시도
```

```
try {
  await FirebaseAuth.instance.signInWithEmailAndPassw
    email: email,
    password: password,
  );
```

```
onSuccess(); // 성공 함수 호출
notifyListeners(); // 로그인 상태 변경 알림
} on FirebaseAuthException catch (e) {
```

```

        // firebase auth 에러 발생
        onError(e.message!);
    } catch (e) {
        // Firebase auth 이외의 에러 발생
        onError(e.toString());
    }
}

```



회원 가입과 마찬가지로 처음에 이메일 및 비밀번호의 유효성 검사를 진행한 뒤 로그인 요청을 보냅니다.

```

43  void signIn({
44      required String email, // 이메일
45      required String password, // 비밀번호
46      required Function onSuccess, // 로그인 성공시 호출되는 함수
47      required Function(String err) onError, // 에러 발생시 호출되는 함수
48  }) async {
49      // 로그인
50      if (email.isEmpty) {
51          onError('이메일을 입력해주세요.');
```

```

52          return;
53      } else if (password.isEmpty) {
54          onError('비밀번호를 입력해주세요.');
```

```

55          return;
56      }
57
58      // 로그인 시도
59      try {
60          await FirebaseAuth.instance.signInWithEmailAndPassword(
61              email: email,
62              password: password,
63          );
64
65          onSuccess(); // 성공 함수 호출
66          notifyListeners(); // 로그인 상태 변경 알림
67      } on FirebaseAuthException catch (e) {
68          // firebase auth 에러 발생
69          onError(e.message!);
70      } catch (e) {
71          // Firebase auth 이외의 에러 발생
72          onError(e.toString());
73      }
74  }
75

```



60번째 줄에서 email과 password로 FirebaseAuth를 이용하여 이메일 회원 가입을 시도합니다.

```
await FirebaseAuth.instance.signInWithEmailAndPassword  
  email: email,  
  password: password,  
);
```



로그인에 성공한 경우 비 로그인 상태에서 로그인 상태로 변경 되었으므로 `notifyListeners()` 를 호출해 모든 Consumer 위젯의 build를 호출해 화면을 갱신해 줍니다.

3. 로그인 기능을 테스트합니다.

- 이메일 형식이 올바르지 않은 경우

- 비밀번호가 일치하지 않는 경우

The login screen has a blue header with the text '로그인'. Below the header, the text '로그인해 주세요 😊' is centered. There are two input fields: the first contains 'a@a.com' and the second contains 'a'. A blue button with the text '로그인' is positioned below the input fields. At the bottom, a dark gray banner displays the error message: 'The password is invalid or the user does not have a password.'

- 로그인 성공

The login screen is identical to the previous one, but the dark gray banner at the bottom now displays the message '로그인 성공' (Login Success).

▼ 7) 유저 정보 가져오기

1. 코드스니펫을 복사해 `auth_service.dart` 파일 6번째 줄 맨 뒤에 붙여 넣어주세요.

▼ [코드스니펫] `auth_service.dart` / `currentUser`

```
return FirebaseAuth.instance.currentUser;
```



`Firebase.instance.currentUser` 는 현재 유저를 조회하는데 로그인 되어있지 않은 경우 null을 반환 합니다.

```
4  class AuthService extends ChangeNotifier {
5    User? currentUser() {
6      // 현재 유저(로그인 되지 않은 경우 null 반환)
7      return FirebaseAuth.instance.currentUser;
8    }
9  }
```

2. 로그인 성공 시 유저 정보를 화면에 보여주도록 하겠습니다. 코드스니펫을 복사해 `main.dart` 파일 48번째 줄 맨 뒤에 붙여 넣어주세요.

▼ [코드스니펫] `main.dart` / `currentUser` 조회

```
final user = authService.currentUser();
```

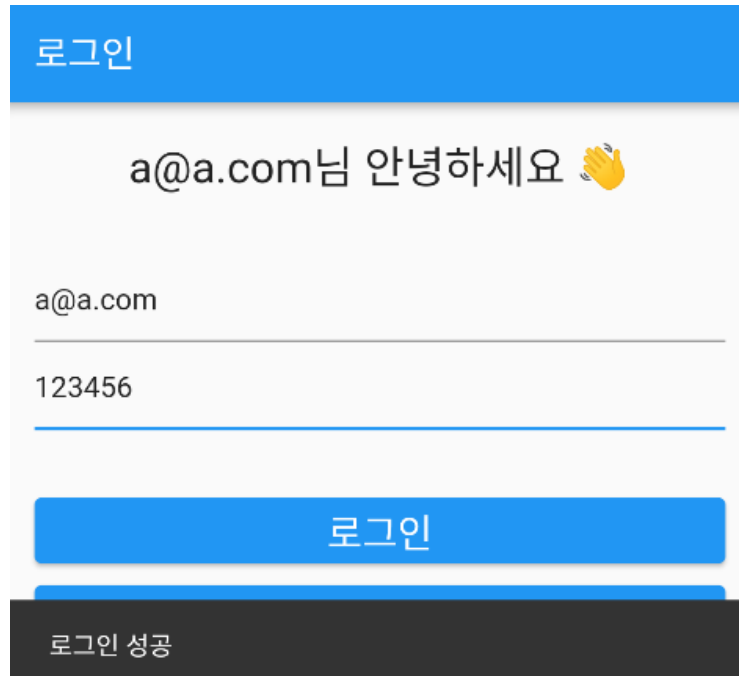
```
45  @override
46  widget build(BuildContext context) {
47    return Consumer<AuthService>(
48      builder: (context, authService, child) {
49        final user = authService.currentUser();
50        return Scaffold(
51          appBar: AppBar(title: Text("로그인")),
52          body: SingleChildScrollView(
53            padding: const EdgeInsets.all(16),
54            child: Column(
```

3. 코드스니펫을 복사해 `main.dart` 파일 60번째 줄을 지우고 붙여 넣어주세요.

▼ [코드스니펫] `main.dart` / 로그인 유저 인사말

```
user == null ? "로그인해 주세요 😊" : "${user.email}님 안녕하세요"
```

아래와 같이 로그인 성공 시 문구가 변경됩니다.



4. 로그인 성공 시 HomePage로 이동하도록 하겠습니다.

코드스니펫을 복사해 `main.dart` 파일 94번째 줄 맨 뒤에 붙여 넣어주세요.

▼ [코드스니펫] `main.dart` / 로그인 성공 시 HomePage로 이동

```
Navigator.pushReplacement(  
  context,  
  MaterialPageRoute(builder: (con  
));
```



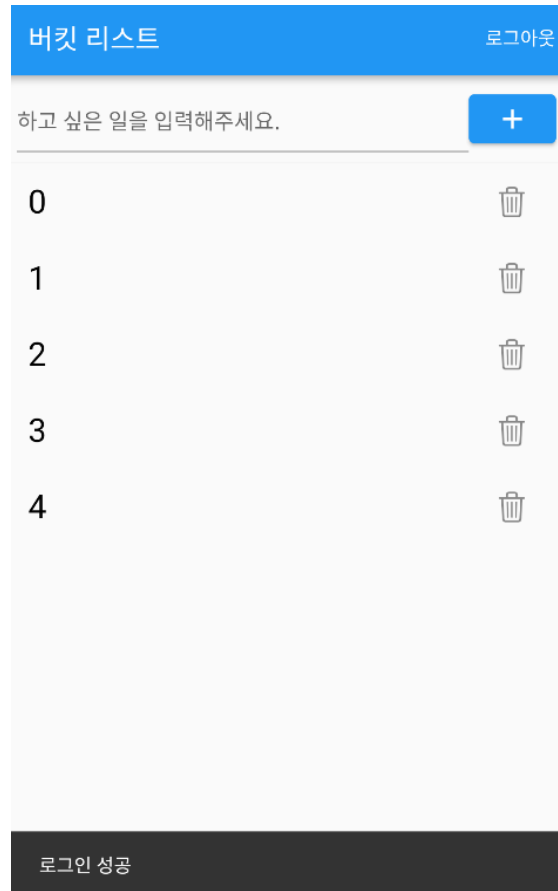
`pushReplacement`로 구현하였으므로 로그인 성공 시 LoginPage를 HomePage로 교체합니다.

```

82      /// 로그인 버튼
83      ElevatedButton(
84        child: Text("로그인", style: TextStyle(fontSize: 21)),
85        onPressed: () {
86          /// 로그인
87          authService.signIn(
88            email: emailController.text,
89            password: passwordController.text,
90            onSuccess: () {
91              /// 로그인 성공
92              ScaffoldMessenger.of(context).showSnackBar(SnackBar(
93                content: Text("로그인 성공"),
94              )); // SnackBar
95
96              /// HomePage로 이동
97              Navigator.pushReplacement(
98                context,
99                MaterialPageRoute(builder: (context) => HomePage()),
100              );
101            },
102            onError: (err) {
103              /// 에러 발생
104              ScaffoldMessenger.of(context).showSnackBar(SnackBar(
105                content: Text(err),

```

로그인을 성공하는 경우 아래와 같이 HomePage로 이동되는 것을 확인할 수 있습니다.



HomePage의 AppBar에 로그아웃 버튼을 만들어 두었습니다.



해당 버튼을 클릭해 보면 LoginPage로 이동하지만 로그인 상태가 그대로 유지되어 있어 유저 email이 그대로 나오는 것을 볼 수 있습니다.

로그인

a@a.com님 안녕하세요 🖐️

이메일

비밀번호

로그인

회원가입

로그아웃 기능을 구현해 문제를 해결해 보도록 하겠습니다.

▼ 8) 로그아웃 만들기

1. 코드스니펫을 복사해 `auth_service.dart` 78번째 라인 맨 뒤에 붙여 넣어주세요.

▼ [코드스니펫] `auth_service.dart` / sign out

```
await FirebaseAuth.instance.signOut();
notifyListeners(); // 로그인 상태 변경 알림
```

💡 로그아웃을 한 뒤, LoginPage 화면을 갱신하기 위해 `notifyListeners()` 를 호출해줍니다.

```
77  void signOut() async {
78      // 로그아웃
79      await FirebaseAuth.instance.signOut();
80      notifyListeners(); // 로그인 상태 변경 알림
81  }
82  }
83
```

2. 로그아웃 버튼 클릭시 AuthService의 signOut 함수를 호출해 주도록 하겠습니다. 코드 스니펫을 복사해 `main.dart` 파일 169번째 줄을 지우고, 붙여 넣어주세요.

▼ [코드스니펫] main.dart / sign out

```
// 로그아웃  
context.read<AuthService>().signOut();
```



HomePage는 `Consumer<AuthService>`를 사용 중이지 않으므로 `context.read<AuthService>()`를 이용하면 위젯 트리 상단에 있는 AuthService에 접근할 수 있습니다.

```
163         "로그아웃",  
164         style: TextStyle(  
165           color: Colors.white,  
166         ), // TextStyle  
167       ), // Text  
168       onPressed: () {  
169         // 로그아웃  
170         context.read<AuthService>().signOut();  
171       },  
172       // 로그인 페이지로 이동  
173       Navigator.pushReplacement(  
174         context,
```

3. 저장한 뒤 다시 로그인을 하여 HomePage로 이동한 뒤 로그아웃 버튼을 눌러 테스트합니다.

이제 로그아웃이 잘 되어서 LoginPage에서 `로그인 해주세요 😊` 라는 문구가 보입니다.

로그인

로그인해 주세요 😊

이메일

비밀번호

로그인

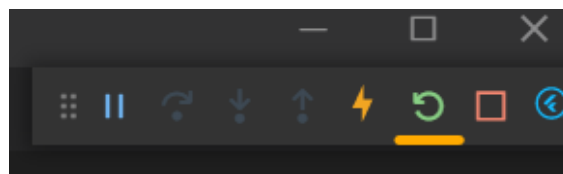
회원가입

▼ 9) 자동 로그인 만들기



앱 시작시 로그인이 되어 있는 경우 LoginPage가 아닌 HomePage로 바로 이동하도록 하겠습니다.

1. 로그인을 한 뒤 HomePage로 이동해주세요.
2. VSCode 우측 상단에 `Restart` 버튼을 눌러주세요.



항상 LoginPage가 뜨는 것을 볼 수 있습니다.

로그인

a@a.com님 안녕하세요 🙌

이메일

비밀번호

로그인

회원가입

3. 코드스니펫을 복사해 `main.dart` 파일에 25번째 라인 맨 뒤에 붙여 넣어주세요.

▼ [코드스니펫] `main.dart` / `MyApp`에서 유저 조회

```
final user = context.read<AuthService>().currentU
```



위젯 트리 상단에 있는 `AuthService`에 접근하여 `currentUser()` 함수를 호출하였습니다.

```

21 ∨ class MyApp extends StatelessWidget {
22     const MyApp({Key? key}) : super(key: key);
23
24     @override
25     ∨ Widget build(BuildContext context) {
26         final user = context.read<AuthService>().currentUser();
27         ∨ return MaterialApp(
28             debugShowCheckedModeBanner: false,
29             home: LoginPage(),
30         ); // MaterialApp
31     }
32 }

```

4. 로그인 상태에 관계없이 29번째 라인에 `home: LoginPage()` 로 고정되어 있어 언제나 LoginPage가 뜨는 것입니다. 코드스니펫을 복사해 29번째 라인을 지우고 붙여 넣어주세요.

▼ [코드스니펫] main.dart / home

```
home: user == null ? LoginPage() : HomePage(),
```



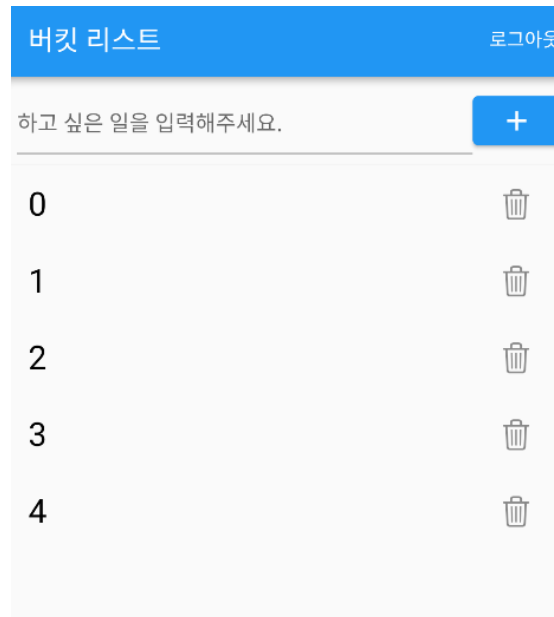
user가 null이라면 로그인 되어있지 않은 상태이므로 `LoginPage` 로 이동하고 그렇지 않은 경우 `HomePage` 로 이동하도록 구현하였습니다.

```

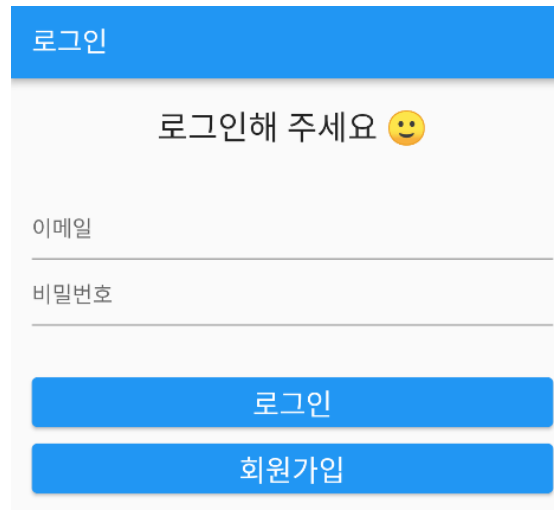
21 ∨ class MyApp extends StatelessWidget {
22     const MyApp({Key? key}) : super(key: key);
23
24     @override
25     ∨ Widget build(BuildContext context) {
26         final user = context.read<AuthService>().currentUser();
27         ∨ return MaterialApp(
28             debugShowCheckedModeBanner: false,
29             home: user == null ? LoginPage() : HomePage(),
30         ); // MaterialApp
31     }
32 }
33

```

5. **Restart** 를 하여 테스트를 해보면 현재 로그인되어 있으므로 바로 HomePage로 이동되는 것을 볼 수 있습니다.



또한 로그아웃한 뒤 다시 **Restart** 를 하면 LoginPage가 뜹니다.



여기까지 Firebase Auth를 사용하여 회원가입 / 로그인 / 로그아웃 등 여러 기능을 구현하였습니다. 사실 이외에도 로그인 기능의 경우 회원 탈퇴, 이메일 인증 등 더 많은 기능들이 남아있는데 이 부분은 공식문서를 참고하시기 바랍니다.

▼ 최종 코드

▼ main.dart

```
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';

import 'auth_service.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized(); // main 함수에서
  await Firebase.initializeApp(); // firebase 앱 시작
  runApp(
    MultiProvider(
      providers: [
        ChangeNotifierProvider(create: (context) => AuthService()),
      ],
      child: const MyApp(),
    ),
  );
}

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

  @override
  Widget build(BuildContext context) {
    final user = context.read<AuthService>().currentUser();
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: user == null ? LoginPage() : HomePage(),
    );
  }
}

/// 로그인 페이지
```

```

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

  @override
  State<LoginPage> createState() => _LoginPageState();
}

class _LoginPageState extends State<LoginPage> {
  TextEditingController emailController = TextEditingControl
  TextEditingController passwordController = TextEditingCont

  @override
  Widget build(BuildContext context) {
    return Consumer<AuthService>(
      builder: (context, authService, child) {
        final user = authService.currentUser();
        return Scaffold(
          appBar: AppBar(title: Text("로그인")),
          body: SingleChildScrollView(
            padding: const EdgeInsets.all(16),
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.stretch
              children: [
                /// 현재 유저 로그인 상태
                Center(
                  child: Text(
                    user == null ? "로그인해 주세요 😊" : "${us
                    style: TextStyle(
                      fontSize: 24,
                    ),
                  ),
                ),
                SizedBox(height: 32),

                /// 이메일
                TextField(
                  controller: emailController,

```