

```

        decoration: InputDecoration(hintText: "이메일"),
    ),

    /// 비밀번호
    TextField(
        controller: passwordController,
        obscureText: false, // 비밀번호 안보이게
        decoration: InputDecoration(hintText: "비밀번호"),
    ),
    SizedBox(height: 32),

    /// 로그인 버튼
    ElevatedButton(
        child: Text("로그인", style: TextStyle(fontWeight: FontWeight.bold)),
        onPressed: () {
            // 로그인
            authService.signIn(
                email: emailController.text,
                password: passwordController.text,
                onSuccess: () {
                    // 로그인 성공
                    ScaffoldMessenger.of(context).showSnackBar(
                        content: Text("로그인 성공"),
                    );

                    // HomePage로 이동
                    Navigator.pushReplacement(
                        context,
                        MaterialPageRoute(builder: (context) => HomePage()),
                    );
                },
            ),
            onError: (err) {
                // 에러 발생
                ScaffoldMessenger.of(context).showSnackBar(
                    content: Text(err),
                );
            },
        ),
    ),

```

```

        );
      },
    ),

    /// 회원가입 버튼
    ElevatedButton(
      child: Text("회원가입", style: TextStyle(fontWeight: FontWeight.bold)),
      onPressed: () {
        /// 회원가입
        authService.signUp(
          email: emailController.text,
          password: passwordController.text,
          onSuccess: () {
            /// 회원가입 성공
            ScaffoldMessenger.of(context).showSnackBar(
              SnackBar(content: Text("회원가입 성공")),
            );
          },
          onError: (err) {
            /// 에러 발생
            ScaffoldMessenger.of(context).showSnackBar(
              SnackBar(content: Text(err)),
            );
          },
        );
      },
    ),
  ],
),
);
},
);
}
}

/// 홈페이지

```

```

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

  @override
  State<HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  TextEditingController jobController = TextEditingControlle

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("버킷 리스트"),
        actions: [
          TextButton(
            child: Text(
              "로그아웃",
              style: TextStyle(
                color: Colors.white,
              ),
            ),
            onPressed: () {
              // 로그아웃
              context.read<AuthService>().signOut();

              // 로그인 페이지로 이동
              Navigator.pushReplacement(
                context,
                MaterialPageRoute(builder: (context) => Logi
              );
            },
          ),
        ],
      ),
      body: Column(

```

```

children: [
  /// 입력창
  Padding(
    padding: const EdgeInsets.all(8),
    child: Row(
      children: [
        /// 텍스트 입력창
        Expanded(
          child: TextField(
            controller: jobController,
            decoration: InputDecoration(
              hintText: "하고 싶은 일을 입력해주세요.",
            ),
          ),
        ),
      ],
    ),

    /// 추가 버튼
    ElevatedButton(
      child: Icon(Icons.add),
      onPressed: () {
        // create bucket
        if (jobController.text.isNotEmpty) {
          print("create bucket");
        }
      },
    ),
  ],
),
Divider(height: 1),

/// 버킷 리스트
Expanded(
  child: ListView.builder(
    itemCount: 5,
    itemBuilder: (context, index) {
      String job = "$index";

```

```

bool isDone = false;
return ListTile(
  title: Text(
    job,
    style: TextStyle(
      fontSize: 24,
      color: isDone ? Colors.grey : Colors.b
      decoration: isDone
        ? TextDecoration.lineThrough
        : TextDecoration.none,
    ),
  ),
  // 삭제 아이콘 버튼
  trailing: IconButton(
    icon: Icon(CupertinoIcons.delete),
    onPressed: () {
      // 삭제 버튼 클릭시
    },
  ),
  onTap: () {
    // 아이템 클릭하여 isDone 업데이트
  },
);
},
),
),
],
),
);
}
}

```

▼ `auth_service.dart`

```

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)
    return FirebaseAuth.instance.currentUser;
  }

  void signUp({
    required String email, // 이메일
    required String password, // 비밀번호
    required Function() onSuccess, // 가입 성공시 호출되는 함수
    required Function(String err) onError, // 에러 발생시 호출도
  }) async {
    // 회원가입
    // 이메일 및 비밀번호 입력 여부 확인
    if (email.isEmpty) {
      onError("이메일을 입력해 주세요.");
      return;
    } else if (password.isEmpty) {
      onError("비밀번호를 입력해 주세요.");
      return;
    }

    // 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());
    }
  }
}

```

```

    }
  }

void signIn({
  required String email, // 이메일
  required String password, // 비밀번호
  required Function() onSuccess, // 로그인 성공시 호출되는 함수
  required Function(String err) onError, // 에러 발생시 호출도
}) async {
  // 로그인
  if (email.isEmpty) {
    onError('이메일을 입력해주세요. ');
    return;
  } else if (password.isEmpty) {
    onError('비밀번호를 입력해주세요. ');
    return;
  }

  // 로그인 시도
  try {
    await FirebaseAuth.instance.signInWithEmailAndPassword(
      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());
  }
}

void signOut() async {

```

```

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

```

▼ pubspec.yaml

```

name: bucket_list_with_firebase
description: A new Flutter project.

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

# The following defines the version and build number for you
# A version number is three numbers separated by dots, like
# followed by an optional build number separated by a +.
# Both the version and the builder number may be overridden
# build by specifying --build-name and --build-number, respe
# In Android, build-name is used as versionName while build-
# Read more about Android versioning at https://developer.an
# In iOS, build-name is used as CFBundleShortVersionString w
# Read more about iOS versioning at
# https://developer.apple.com/library/archive/documentation/
version: 1.0.0+1

environment:
  sdk: ">=2.17.6 <3.0.0"

# Dependencies specify other packages that your package need
# To automatically upgrade your package dependencies to the
# consider running `flutter pub upgrade --major-versions`. A
# dependencies can be manually updated by changing the versi
# the latest version available on pub.dev. To see which depe
# versions available, run `flutter pub outdated`.
dependencies:

```



```

flutter:
  sdk: flutter

# The following adds the Cupertino Icons font to your appl
# Use with the CupertinoIcons class for iOS style icons.
cupertino_icons: ^1.0.2
firebase_core: 1.21.0
firebase_auth: 3.6.4
cloud_firestore: 3.4.5
provider: ^6.0.3

dev_dependencies:
  flutter_test:
    sdk: flutter

# The "flutter_lints" package below contains a set of reco
# encourage good coding practices. The lint set provided b
# activated in the `analysis_options.yaml` file located at
# package. See that file for information about deactivatin
# rules and activating additional ones.
flutter_lints: ^2.0.0

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

# The following section is specific to Flutter packages.
flutter:

  # The following line ensures that the Material Icons font
  # included with your application, so that you can use the
  # the material Icons class.
  uses-material-design: true

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

```

```
# - images/a_dot_ham.jpeg

# An image asset can refer to one or more resolution-speci
# https://flutter.dev/assets-and-images/#resolution-aware

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

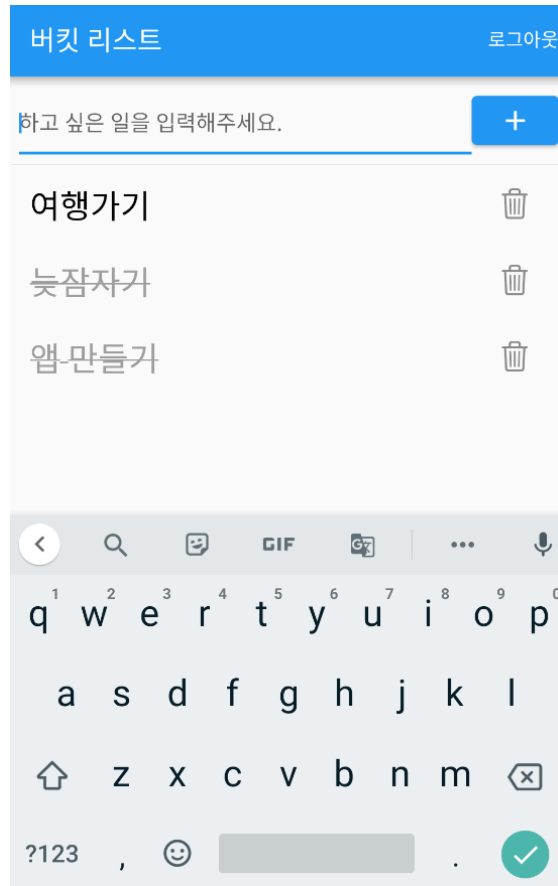
# To add custom fonts to your application, add a fonts sec
# in this "flutter" section. Each entry in this list shoul
# "family" key with the font family name, and a "fonts" ke
# list giving the asset and other descriptors for the font
# example:
# fonts:
#   - family: Schyler
#     fonts:
#       - 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
```

05. 데이터베이스 연동하기

▼ 완성본



Cloud Firestore에 데이터 저장하고 불러오는 방법을 배워보도록 하겠습니다.



▼ 1) Cloud Firestore 이해하기

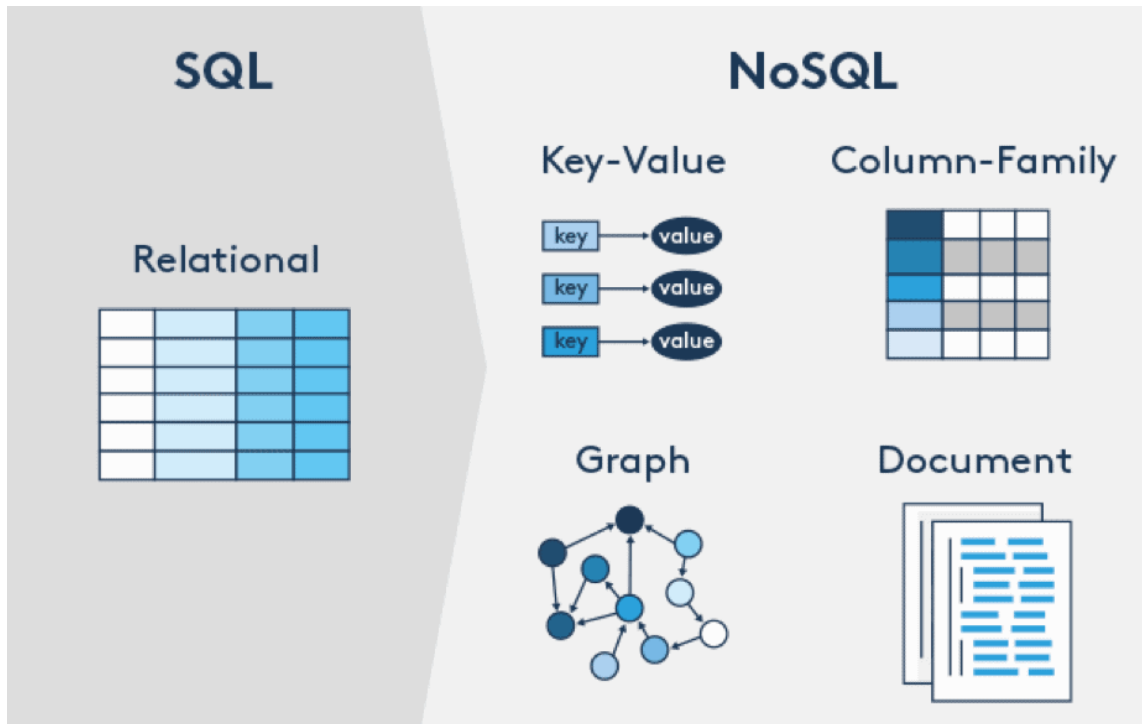
▼ 데이터 저장 위치에 따른 장단점

저장 위치	장점	단점
기기에 저장	서버 운영 비용이 들지 않음 오프라인 서비스 가능	다른 기기에서 사용 불가능 다른 유저간 소통 불가능
서버에 저장 (기기 외부에 저장)	여러 기기에서 사용 가능 다른 유저간 소통 가능	서버 비용이 들음 인터넷 연결 필수

▼ SQL vs NoSQL



데이터 저장에 특화된 데이터베이스는 크게 SQL 진영과 NoSQL 진영으로 제품군을 나눌 수 있습니다.



출처 - [k21academy](https://k21academy.com)

- SQL
 - 데이터를 엑셀과 같이 정해진 틀에 저장합니다.
 - 제품 : MySQL, Oracle 등
- NoSQL
 - SQL 진영보다 데이터를 자유롭게 다양한 형태로 저장합니다.
 - 제품 : Firestore, MongoDB 등



이번에 사용해볼 Cloud Firestore는 NoSQL에 속해있습니다.

▼ Cloud Firestore 구조



Firestore에 대한 공식문서는 아래 링크를 참고해 주세요.

▼ [코드스니펫] Cloud Firestore 공식문서

<https://firebase.google.com/docs/firestore/data-model>

Flutter에서 Firestore를 사용하는 방법은 아래 링크를 참고해주세요.

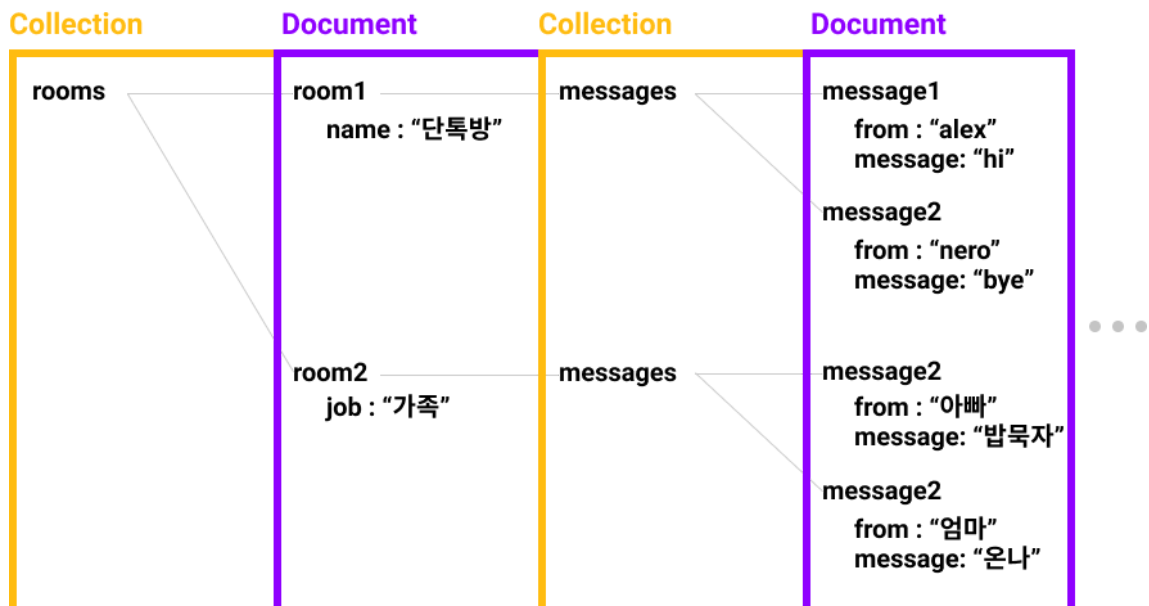
▼ [코드스니펫] Flutter cloud firestore 공식문서

<https://firebase.flutter.dev/docs/firestore/usage>



Firestore는 아래와 같이 컬렉션(Collection)과 문서(Document)를 반복하는 형태로 데이터를 저장합니다. API 응답으로 넘어오는 JSON 형태와 비슷하게 생겼습니다.

- Collection : 여러 Document를 가진 폴더 역할을 합니다.
- Document : 고유 ID와 Map과 같이 **key : value** 형태로 구성된 문서입니다. 문서는 또 다른 Collection을 가질 수 있습니다.

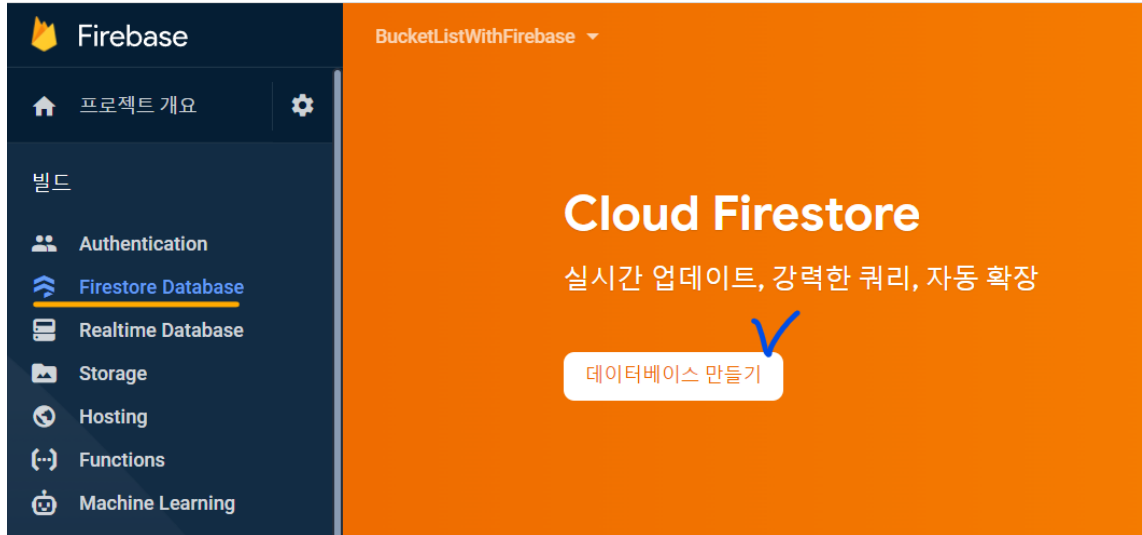




Firestore 사용 준비를 해보도록 하겠습니다.

▼ 2) Cloud Firestore 사용 준비

1. Firebase 콘솔에서 **Firestore Database** 를 선택한 뒤 **데이터베이스 만들기** 버튼을 클릭해 주세요.



2. **테스트 모드에서 시작** 을 선택한 뒤 **다음** 버튼을 눌러주세요.



데이터베이스 CRUD 권한을 설정하는 내용인데, 지금은 테스트 모드로 진행하도록 하겠습니다.

데이터베이스 만들기

1 Cloud Firestore의 보안 규칙

2 Cloud Firestore 위치 설정

데이터 구조를 정의한 후 데이터에 보안을 적용하는 규칙을 작성해야 합니다.

[자세히 알아보기](#)

프로덕션 모드에서 시작

데이터는 기본적으로 비공개됩니다. 클라이언트 읽기/쓰기 액세스 권한은 보안 규칙에서 지정한 대로만 부여됩니다.

테스트 모드에서 시작

빠른 설정을 위해 기본적으로 데이터가 공개됩니다. 하지만 30일 내에 보안 규칙을 업데이트하여 장기적 클라이언트 읽기/쓰기 액세스 권한을 사용 설정해야 합니다.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if
        request.time < timestamp.date(2022, 1, 2);
    }
  }
}
```

! 테스트 모드의 기본 보안 규칙에서는 데이터베이스 참조 사용자는 누구나 향후 30일 동안 데이터베이스의 모든 데이터를 보고 수정하며 삭제할 수 있습니다.

Cloud Firestore를 사용 설정하면 이 프로젝트, 특히 관련 App Engine 앱에서 Cloud Datastore를 사용할 수 없게 됩니다.

취소

다음

3. Cloud Firestore 위치를 지정하는 화면이 나옵니다. 데이터베이스의 물리적 위치라고 보시면 되는데, 서비스하려는 장소와 가까울수록 네트워크 속도가 빠릅니다.



서울에 설치하고 싶은 경우 `asia-northeast3`를 선택하시고, 그 외의 지역이라면 아래 코드스니펫을 복사해 새 탭에서 열어 전체 지역을 확인하실 수 있습니다.

▼ [코드스니펫] firestore region 목록 URL

`https://cloud.google.com/datastore/docs/locations`

원하는 위치를 선택한 뒤 **사용 설정** 버튼을 눌러주세요.

데이터베이스 만들기

✓ Cloud Firestore의 보안 규칙

2 Cloud Firestore 위치 설정

위치 설정은 Cloud Firestore 데이터가 저장되는 위치를 결정합니다.

⚠ 이 위치를 설정한 후에는 나중에 변경할 수 없습니다. 또한 설정한 위치가 기본 **Cloud Storage** 버킷의 위치가 됩니다.

[자세히 알아보기](#)

Cloud Firestore 위치

asia-northeast3

Cloud Firestore를 사용 설정하면 이 프로젝트, 특히 관련 App Engine 앱에서 Cloud Datastore를 사용할 수 없게 됩니다.

취소

사용 설정

4. 다음과 같은 화면이 나오면 데이터베이스 사용 준비가 완료 되었습니다.

BucketListWithFirebase


Cloud Firestore

데이터 규칙 색인 사용량

로컬 에뮬레이터 도구 모음을 사용한 프로토타입 제작 및 엔드 투 엔드 테스트에 **Firebase** 인증 지원 도입 [시작하기](#)

bucketlistwithfirebase

+ 컬렉션 시작

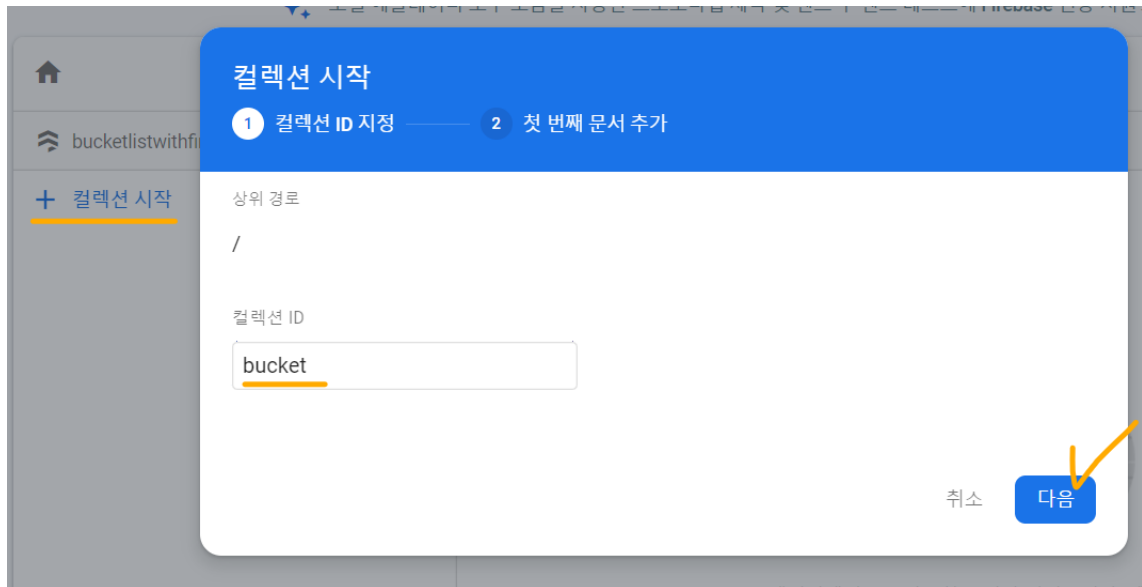


데이터베이스를 사용할 준비가 되었습니다. 데이터만 추가하면 됩니다.



간단하게 Firesbase 콘솔에서 데이터를 만들고 지우는 방법을 알아보겠습니다.

5. **+ 컬렉션 시작** 을 누르면 아래와 같이 팝업이 뜹니다. **컬렉션 ID** 에 **bucket** 이라고 입력한 뒤 **다음** 버튼을 눌러주세요.



6. **자동 ID** 를 선택해 주세요.



컬렉션 하위에 문서 ID가 중복되지 않도록 ID를 자동으로 만들어주는 기능입니다.

컬렉션 시작

✓ 컬렉션 ID 지정 — 2 첫 번째 문서 추가

문서 상위 경로

/bucket

문서 ID

자동 ID

필드

유형

값

= string

+

취소

저장

7. 아래와 같이 문서(doc)를 작성한 뒤 **저장** 버튼을 눌러주세요.

문서 ID

rTUweZ1HqK0f5FHAVKeb

필드

유형

값

uid = string user-id

필드

유형

값

job = string 여행가기

필드

유형

값

isDone = boolean false

+

취소

저장



잠시 후 Flutter로 실제 위와 같이 데이터를 저장할 예정입니다.

uid : 어떤 유저가 작성한 버킷인지 알 수 있는 유저의 고유 식별자

job : 하고싶은 일

isDone : 완료 여부

위 필드 이름들은 정해져 있지 않고, 원하는 이름을 변수명 규칙을 따라 만들면 됩니다. 어떤 값이 담길지 알 수 있도록 직관적인 이름으로 지으면 좋습니다.

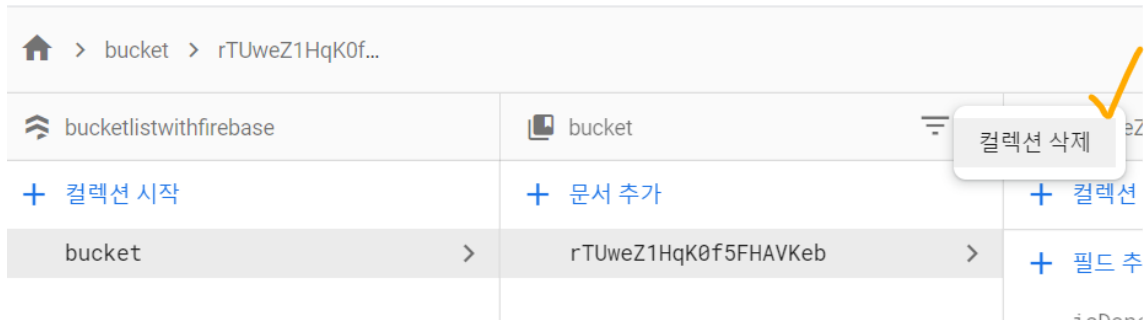
8. 다음과 같이 데이터가 생성 됩니다.

<div> <div>home > bucket > rTUweZ1HqK0f...</div> <div> <div>bucketlistwithfirebase</div> <div>bucket</div> <div>rTUweZ1HqK0f5FHAVKeb</div> </div> </div>		
<div>+ 컬렉션 시작</div> <div>bucket ></div>	<div>+ 문서 추가</div> <div>rTUweZ1HqK0f5FHAVKeb ></div>	<div>+ 컬렉션 시작</div> <div>+ 필드 추가</div> <div> <div>isDone: false</div> <div>job: "여행가기"</div> <div>uid: "user-id"</div> </div>

9. 데이터를 삭제해 보도록 하겠습니다. 아래 이미지에서 **컬렉션 삭제** 라고 적힌 문구 밑의 버튼을 눌러주세요

<div> <div>home > bucket > rTUweZ1HqK0f...</div> <div> <div>컬렉션 삭제</div> <div>문서 삭제</div> </div> </div>		
<div>bucketlistwithfirebase</div> <div>bucket</div> <div>rTUweZ1HqK0f5FHAVKeb</div>	<div>+ 컬렉션 시작</div> <div>+ 문서 추가</div> <div>rTUweZ1HqK0f5FHAVKeb ></div>	<div>+ 컬렉션 시작</div> <div>+ 필드 추가</div> <div> <div>isDone: false</div> <div>job: "여행가기"</div> <div>uid: "user-id"</div> </div>

컬렉션 삭제 를 선택해 주세요.



아래와 같이 팝업이 뜨면 컬렉션 이름인 `bucket` 을 입력한 뒤 `삭제` 버튼을 눌러주세요.

⚠ 컬렉션을 삭제하시겠어요?

이 작업으로 중첩된 모든 문서 및 컬렉션을 포함하여 컬렉션 경로의 데이터가 영구적으로 삭제됩니다.

컬렉션 경로

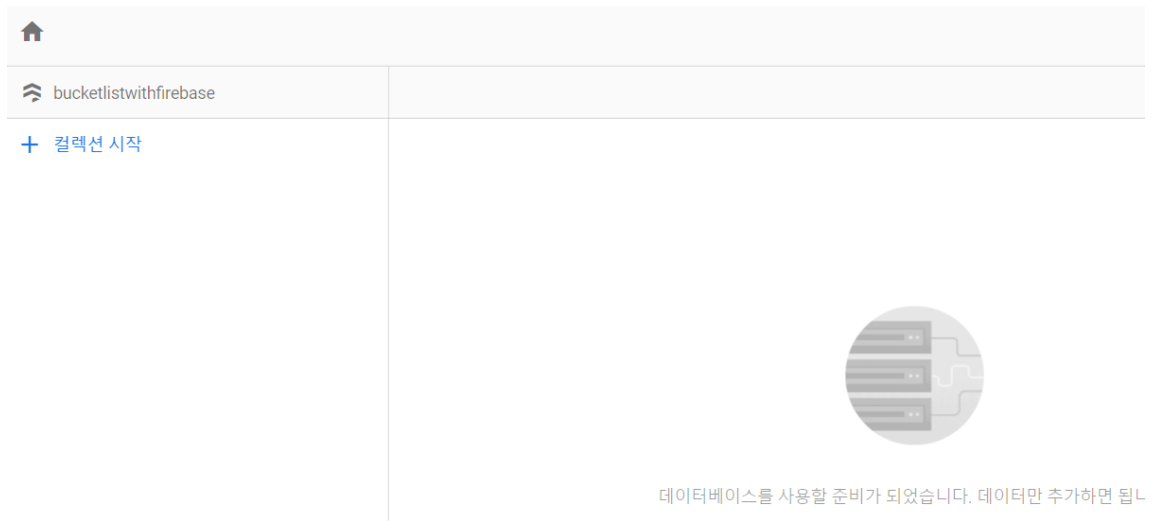
`/bucket`

컬렉션을 삭제하려면 컬렉션 ID를 입력하세요. `bucket`

취소

삭제

10. 모든 데이터가 삭제 되었습니다.

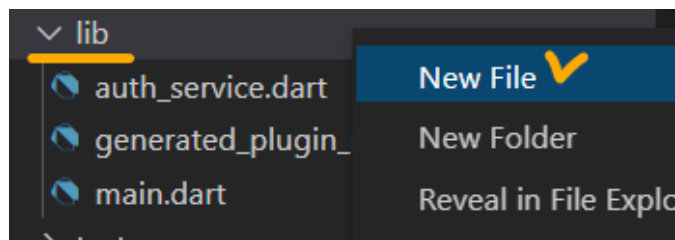


💡 Flutter 프로젝트에서 Firestore CRUD를 구현해 보도록 하겠습니다.

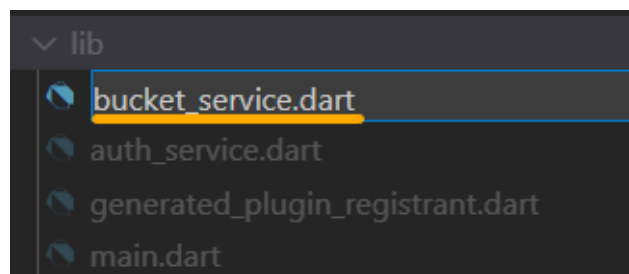
▼ 3) BucketService 만들기

💡 Bucket CRUD를 담당하는 BucketService를 만들어봅시다.

1. VSCode에 `lib` 폴더를 우클릭하여 `New File` 을 선택해주세요.



2. `bucket_service.dart` 라고 파일 이름을 입력해주세요.



3. `BucketService` 로 구현할 기능은 다음과 같습니다.



read : 내가 작성한 버킷 리스트 불러오기
create : 버킷 생성하기
update : 버킷 isDone 업데이트
delete : 버킷 삭제하기

코드스니펫을 복사해 `bucket_service.dart` 파일에 붙여 넣어주세요.

▼ [코드스니펫] `bucket_service.dart` / 시작

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

class BucketService extends ChangeNotifier {
  final bucketCollection = FirebaseFirestore.instance.col

  Future<QuerySnapshot> read(String uid) async {
    // 내 bucketList 가져오기
    throw UnimplementedError(); // return 값 미구현 에러
  }

  void create(String job, String uid) async {
    // bucket 만들기
  }

  void update(String docId, bool isDone) async {
    // bucket isDone 업데이트
  }

  void delete(String docId) async {
    // bucket 삭제
  }
}
```



컬렉션 이름도 변수명 규칙을 따라 원하는 이름을 지으면 되는데 여기에선 모든 데이터를 `bucket` 이라는 이름의 Collection에 저장하도록 하겠습니다.

`bucket` Collection을 가리키는 변수를 아래와 같이 만들 수 있습니다.

```
final bucketCollection = FirebaseFirestore.instance.co
```

```
1 import 'package:cloud_firestore/cloud_firestore.dart';
2 import 'package:flutter/material.dart';
3
4 class BucketService extends ChangeNotifier {
5   final bucketCollection = FirebaseFirestore.instance.collection('bucket');
6
7   Future<QuerySnapshot> read(String uid) async {
8     // 내 bucketList 가져오기
9     throw UnimplementedError(); // return 값 미구현 에러
10  }
11
12  void create(String job, String uid) async {
13    // bucket 만들기
14  }
15
16  void update(String docId, bool isDone) async {
17    // bucket isDone 업데이트
18  }
19
20  void delete(String docId) async {
21    // bucket 삭제
22  }
23 }
24
```



cloud_firestore 패키지의 상세한 사용 방법은 아래 링크를 참고해 주세요.

▼ **[코드스니펫] cloud firestore 패키지 사용법 문서**

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

4. BucketService를 provider를 이용하여 위젯 트리 최상단에 추가하도록 하겠습니다.

코드스니펫을 복사해 `main.dart` 파일 14번째 줄 맨 뒤에 붙여 넣어주세요.

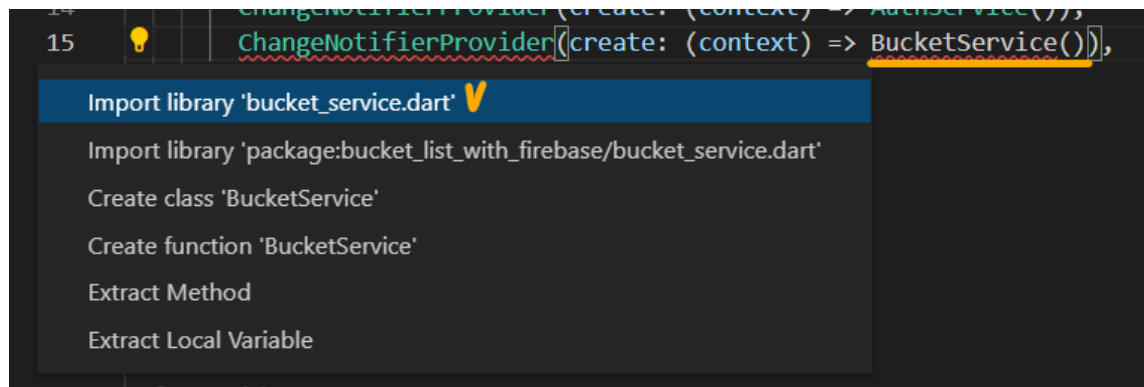
▼ [코드스니펫] `main.dart` / BucketService Provider

```
ChangeNotifierProvider(create: (context)
```

```
8 void main() async {
9   WidgetsFlutterBinding.ensureInitialized(); // main 함수에서 async 사용하기 위
10  await Firebase.initializeApp(); // firebase 앱 시작
11  runApp(
12    MultiProvider(
13      providers: [
14        ChangeNotifierProvider(create: (context) => AuthService()),
15        ChangeNotifierProvider(create: (context) => BucketService()),
16      ],
17      child: const MyApp(),
18    ), // MultiProvider
19  );
20 }
21
```

5. 15번째 줄에서 `BucketService` 가 아직 Import 되지 않아 에러가 발생합니다.

`BucketService` 를 클릭한 뒤 Quick Fix(`Ctrl/Cmd + .`)를 누른 뒤 `Import library`
'`bucket_service.dart`' 를 선택해 주세요.



그리고 저장해주시면 아래와 같이 7번째 줄에 `bucket_service.dart` 파일이 import 되고 문제가 해결 됩니다.


```

5
6 import 'auth_service.dart';
7 import 'bucket_service.dart';
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         ChangeNotifierProvider(create: (context) => BucketService()),
17       ],
18       child: const MyApp(),
19     ), // MultiProvider
20   );
21 }
22

```

6. BucketService에서 CRUD를 하는 경우 HomePage 화면을 갱신해 줄 수 있도록 Consumer로 HomePage를 감싸주겠습니다.

160번째 줄에서 `Scaffold` 를 선택한 뒤 우클릭하여 `Refactor` 를 선택해 주세요.

```

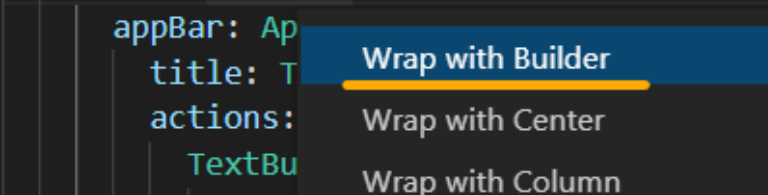
154
155 class _HomePageState extends State<HomePage> {
156   TextEditingController jobController = TextEditingContro
157
158   @override
159   Widget build(Buil
160   return Scaffold
161     appBar: AppBa
162     title: Text
163     actions: [
164       TextButto
165       child:
166       "로그
167       style
168       col
169     ), //
170   ), //
171   onPressed
172     // 로
173     conte
174
175     //

```

Go to Definition	F12
Go to Implementations	Ctrl+F12
Go to References	Shift+F12
Peek	>
Find All References	Shift+Alt+F12
Find All Implementations	
Rename Symbol	F2
Change All Occurrences	Ctrl+F2
Format Document	Shift+Alt+F
Format Document With...	
Refactor...	Ctrl+Shift+R
Source Action...	

`Wrap with Builder` 를 선택한 뒤 저장해주세요.

```
154
155 class _HomePageState extends State<HomePage>
156     TextEditingController jobController = Text
157
158     @override
159     Widget build(BuildContext context) {
160         return Scaffold(
161             appBar: App
162             title: T
163             actions:
164             TextBu
```



그러면 아래와 같이 `Scaffold` 를 Builder 위젯이 감싸게 됩니다.

```
158     @override
159     Widget build(BuildContext context) {
160         return Builder(builder: (context) {
161             return Scaffold(
162                 appBar: AppBar(
163                     title: Text("버킷 리스트"),
164                     actions: [
```

아래 코드스니펫을 복사해 160번째 줄을 지우고 붙여 넣어 주세요.

▼ [코드스니펫] main.dart / Consumer<BucketService>

```
return Consumer<BucketService>(builder: (context, buckets
```



이제 `BucketService` 에서 `notifyListeners()` 를 호출하면 HomePage의 Consumer 위젯의 builder 함수가 실행되면서 화면이 갱신되게 됩니다.

```

158   @override
159   Widget build(BuildContext context) {
160     return Consumer<BucketService>(builder: (context, bucketService, child) {
161       return Scaffold(
162         appBar: AppBar(

```

줄 정렬을 위해 아래 사진과 같이 252번째 줄의 중괄호(`}`)와 소괄호(`(`)) 사이에 콤마(`,`)를 추가한 뒤 저장해주세요.

```

250     ), // Column
251   ); // Scaffold
252   },); // Consumer
253 }
254 }
255 |

```

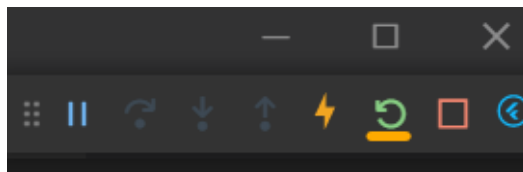
그러면 아래와 같이 Consumer와 builder가 정렬됩니다.

```

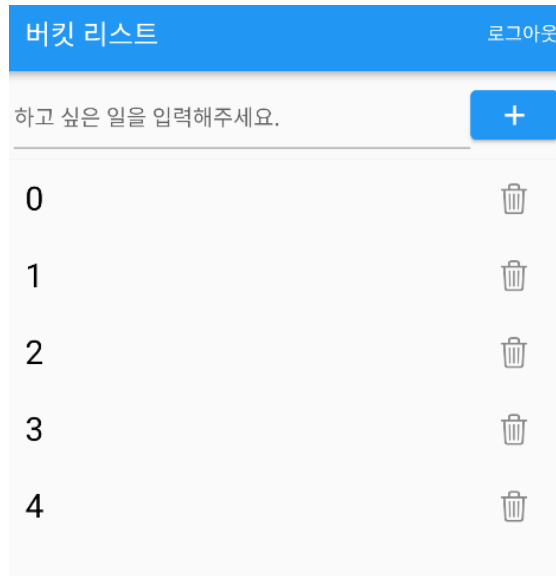
155   class _HomePageState extends State<HomePage> {
156     TextEditingController jobController = TextEditingController();
157
158     @override
159     Widget build(BuildContext context) {
160       return Consumer<BucketService>(
161         builder: (context, bucketService, child) {
162           return Scaffold(
163             appBar: AppBar(
164               title: Text("버킷 리스트"),
165               actions: [

```

7. 에뮬레이터에서 에러가 발생하면 VSCode 우측 상단에 **Refresh** 을 눌러주세요.



그러면 정상적으로 HomePage가 출력됩니다.



8. HomePage에서 현재 로그인 된 user를 가져올 수 있도록 해주겠습니다. 코드스니펫을 복사해 159번째 라인 맨 뒤에 붙여 넣어주세요.

▼ [코드스니펫] main.dart / HomePage user

```
final authService = context.read<AuthService>();  
final user = authService.currentUser()!;
```



HomePage는 로그인한 유저만 접속할 수 있으니 `currentUser()` 를 호출했을 때 null을 응답받을 일이 없습니다. 따라서 161번째 라인에서 `currentUser()` 라고 뒤에 느낌표(!)를 붙여서 타입을 `User?` 가 아닌 `User` 로 강제로 nullable을 제거해줍니다.

만약 null을 강제로 제거했는데

`currentUser()` 가 null을 반환한다면 에러가 발생합니다.

```

155  class _HomePageState extends State<HomePage> {
156      TextEditingController jobController = TextEditingController();
157
158      @override
159      Widget build(BuildContext context) {
160          final authService = context.read<AuthService>();
161          final user = authService.currentUser!;
162          return Consumer<BucketService>(
163              builder: (context, bucketService, child) {
164                  return Scaffold(
165                      appBar: AppBar(

```

이제 HomePage에서는 바로 `user` 에 접근이 가능합니다.



`create` 기능부터 하나씩 만들어 보도록 하겠습니다.

▼ 4) Create 만들기



`main.dart` 파일에 HomePage를 보면 아래와 같이 Bucket CRUD를 한 페이지에서 할 수 있도록 구성하였습니다.

버킷 리스트

로그아웃

하고 싶은 일을 입력해주세요. +

Create

0

1

클릭 👉 Update

2

3

4

Read

Delete



Create 기능은 + 버튼을 누르는 것으로 시작됩니다.

1. + 버튼을 눌렀을 때 현재 입력된 텍스트를 가져와 BucketService의 create 함수를 호출하도록 만들어 줍니다.



참고로 HomePage에 `jobController` 를 TextField에 controller로 등록해 두었으므로 `jobController.text` 로 현재 입력된 값을 가져올 수 있습니다.

```
155 class _HomePageState extends State<HomePage> {
156   TextEditingController jobController = TextEditingController();
```

```
195 // 텍스트 입력창
196 Expanded(
197   child: TextField(
198     controller: jobController,
199     decoration: InputDecoration(
200       hintText: "하고 싶은 일을 입력해주세요.",
201     ), // InputDecoration
202   ), // TextField
203 ), // Expanded
204
```

코드스니펫을 복사해 211번째 라인을 지우고 붙여 넣어주세요.

▼ [코드스니펫] main.dart / bucketService create

```
bucketService.create(jobController.text, user.uid);
```



`user.uid` 로 로그인된 유저의 고유 식별자 uid를 가져올 수 있습니다.

```
205 // 추가 버튼
206 ElevatedButton(
+ 207   child: Icon(Icons.add),
208   onPressed: () {
209     // create bucket
210     if (jobController.text.isNotEmpty) {
211       bucketService.create(jobController.text, user.uid);
212     }
213   },
214 ), // ElevatedButton
```

2. `bucket_service.dart` 파일에 create 함수를 구현하도록 하겠습니다.

코드스니펫을 복사해 `bucket_service.dart` 파일의 13번째 라인 맨 뒤에 붙여 넣어주세요.

▼ [코드스니펫] bucket_service.dart / create

```

        await bucketCollection.add({
          'uid': uid, // 유저 식별자
          'job': job, // 하고싶은 일
          'isDone': false, // 완료 여부
        });
        notifyListeners(); // 화면 갱신

```



`bucket_service.dart` 파일의 5번째 줄에 미리 만들어 둔 `bucketCollection` 에 문서 (document)를 추가합니다. 아래와 같이 collection 뒤에 바로 `add` 를 호출하여 문서를 입력하면 문서 ID를 랜덤으로 부여하게 됩니다.

만약 문서의 ID를 지정하고 싶은 경우 아래와 같이 작성하시면 됩니다.

```

        await bucketCollection.doc("원하는 ID").set({
          'uid': uid, // 유저 식별자
          'job': job, // 하고싶은 일
          'isDone': false, // 완료 여부
        });

```

좀 더 자세한 사항은 아래 링크를 참고해 주세요.


▼ **[코드스니펫] flutter firestore / adding documents 공식 문서**

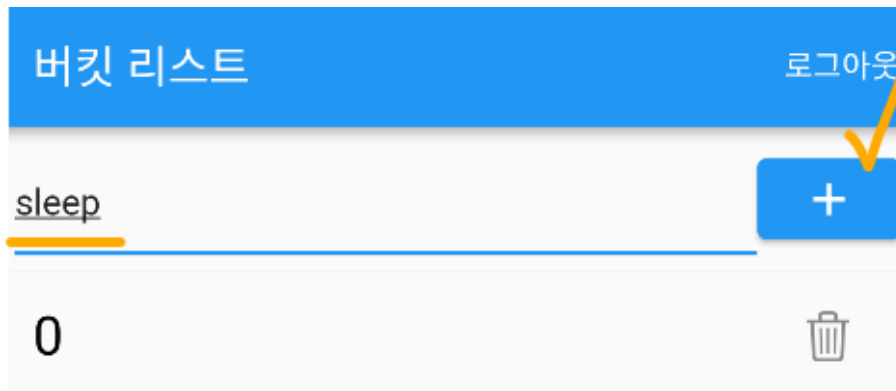
<https://firebase.flutter.dev/docs/firestore/usage/#>


```

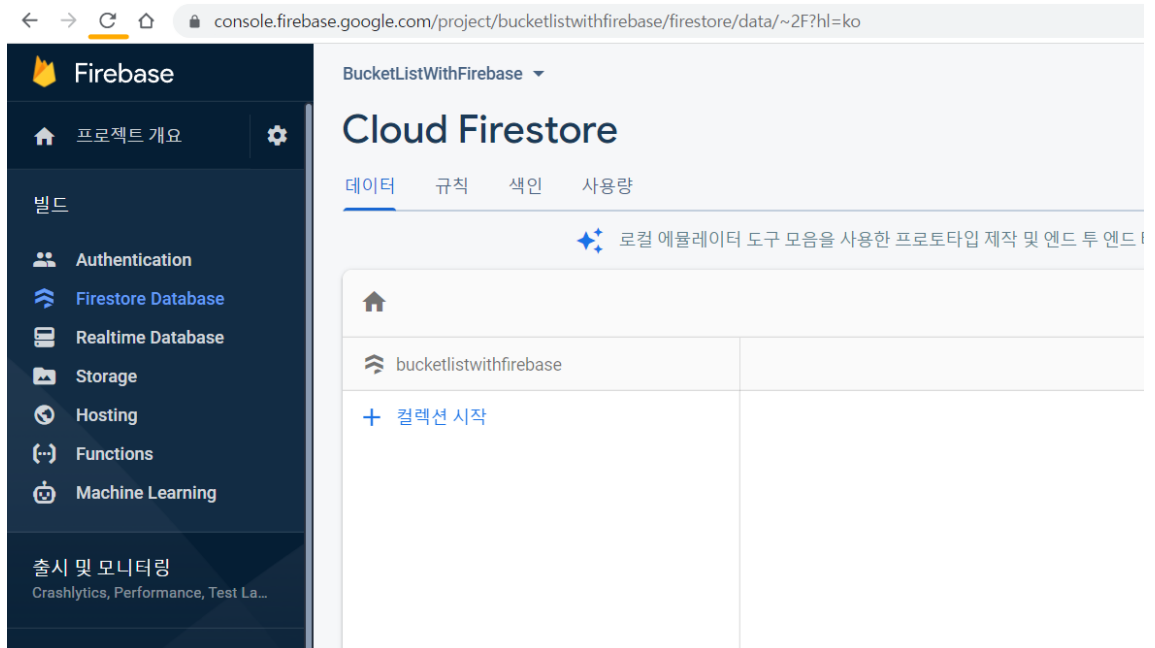
12  void create(String job, String uid) async {
13      // bucket 만들기
14      await bucketCollection.add({
15          'uid': uid, // 유저 식별자
16          'job': job, // 하고싶은 일
17          'isDone': false, // 완료 여부
18      });
19      notifyListeners(); // 화면 갱신
20  }
21

```

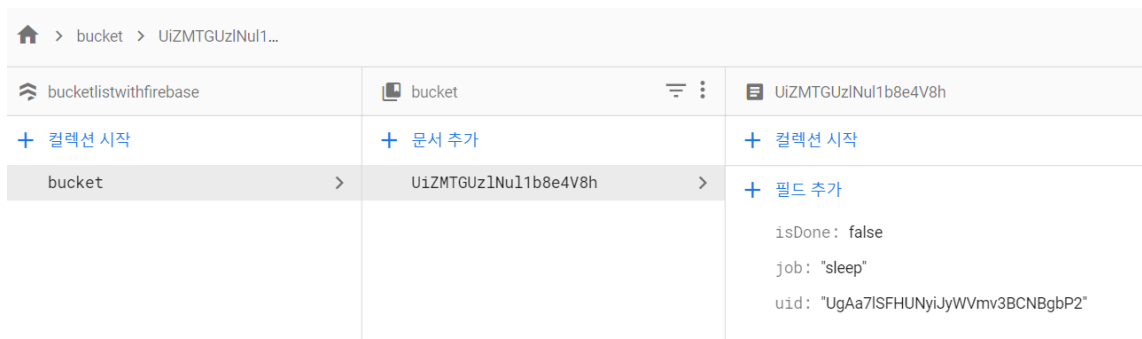
3. 저장한 뒤 bucket을 추가해 보도록 하겠습니다. 아래와 같이 하고 싶은 일을 작성하고 버튼을 눌러주세요. 



Firebase console을 새로고침해 주세요.



그러면 아래와 같이 bucket collection과 문서가 추가된 것을 확인할 수 있습니다.



💡 다음으로 Read 기능을 구현해 보도록 하겠습니다.

▼ 5) Read 만들기

💡 Read 기능은 bucket 컬렉션 밑의 문서들을 가져오는데, uid가 현재 로그인된 유저의 uid와 일치하는 문서만 가져와야 합니다.

1. 코드스니펫을 복사해 `bucket_service.dart` 파일의 9번째 줄을 지우고 붙여 넣어주세요.

▼ [코드스니펫] bucket_service.dart / read

```
return bucketCollection.where('uid', isEqualTo: uid).get(
```



조건에 맞는 문서를 가져오기 위해 `where` 를 사용하였습니다.

`where('uid', isEqualTo: uid)` : 전달 받은 uid와 일치하는 문서만 가져오기

좀 더 자세한 사항은 아래 링크를 참고해 주세요.

▼ [코드스니펫] flutter firestore / querying 공식 문서

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

```
6
7   Future<QuerySnapshot> read(String uid) async {
8     // 내 bucketList 가져오기
9     return bucketCollection.where('uid', isEqualTo: uid).get();
10  }
11
```

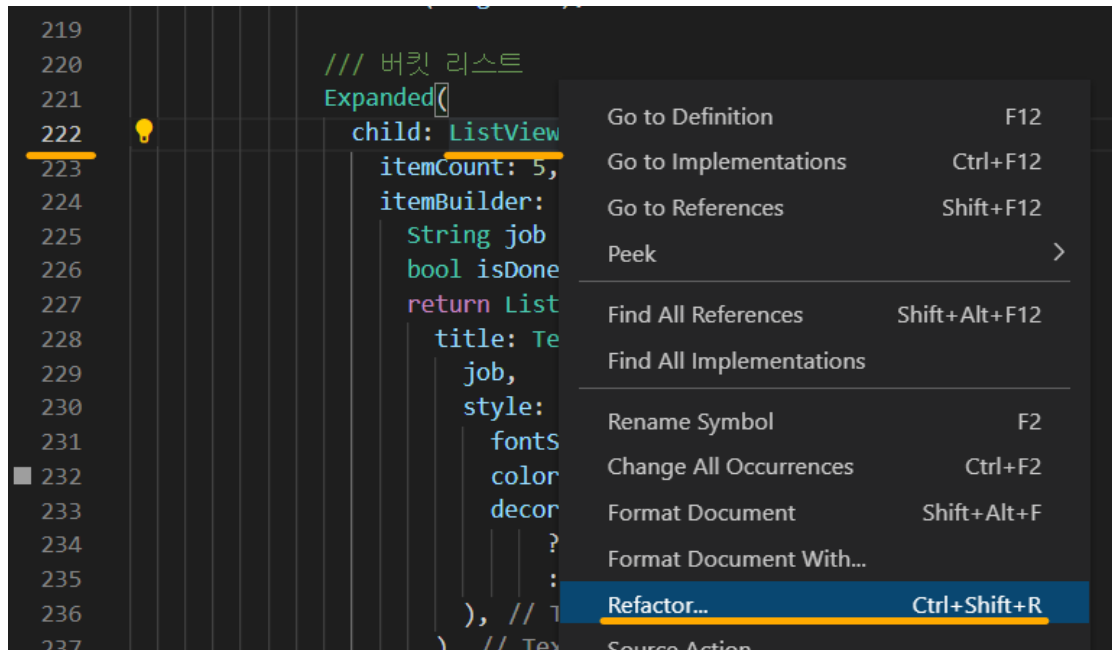


최종적으로 `컬렉션.get()` 을 하는 경우 반환하는 값이 `Future<QuerySnapshot>` 입니다. 보다 상세한 반환 값은 9번째 줄에 `.get()` 함수에 마우스를 올려두면 확인할 수 있습니다. 최종적으로 key와 value로 된 문서를 반환하기 때문에 마지막은 `Map<String, dynamic>` 형태입니다.

```
lib > bucket_service.dart > BucketService > create
3
4 class BucketService extends ChangeNotifier {
5   final bucketCollection = FirebaseFirestore.instance.collection('bucketList');
6
7   Future<QuerySnapshot<Map<String, dynamic>>> read(String uid) async {
8     // 내 bucketList 가져오기
9     return bucketCollection.where('uid', isEqualTo: uid).get();
10  }
11
12 void create(String job, String uid) async {
13   // bucket 만들기
14 }
```

`Future<QuerySnapshot<Map<String, dynamic>>> get([GetOptions? options])`
package:cloud_firestore/cloud_firestore.dart
Fetch the documents for this query.
To modify how the query is fetched, the [options] parameter provided with a [GetOptions] instance.

2. 구현한 read 함수를 `main.dart` 파일에서 호출하여 화면에 데이터를 보여주도록 하겠습니다. 222번째 줄에 `ListView` 를 선택한 뒤 우클릭하여 `Refactor` 를 선택해주세요.



`Wrap with StreamBuilder` 를 선택해 주세요.



그러면 아래와 같이 `StreamBuilder` 가 `ListView` 를 감싸게 됩니다.

```

220      /// 버킷 리스트
221      Expanded(
222        child: StreamBuilder<Object>(  

223          stream: null,  

224          builder: (context, snapshot) {  

225            return ListView.builder(  

226              itemCount: 5,  

227              itemBuilder: (context, index) {  

228                String job = "$index";

```

3. 코드스니펫을 복사해 222 ~ 223번째 줄을 지우고 붙여 넣어주세요.

▼ [코드스니펫] main.dart / FutureBuilder

```

child: FutureBuilder<QuerySnapshot>(
  future: bucketService.read(user.uid),

```



BucketService의 read 기능은 반환하는 값이 시간이 걸리는 `Future` 이기 때문에 화면에 바로 보여줄 수 없습니다. 하지만 `FutureBuilder` 라는 위젯을 활용하면 통신하여 데이터를 받아온 뒤에 builder 부분이 갱신되면서 화면에 보여줄 수 있습니다.

FutureBuilder의 상세 내용은 코드스니펫을 복사해 새 탭에서 열어 확인해주세요.

▼ [코드스니펫] Youtube FutureBuilder URL

<https://www.youtube.com/watch?v=ek8ZPdWj4Qo>

```

220      /// 버킷 리스트
221      Expanded(
222        child: FutureBuilder<QuerySnapshot>(
223          future: bucketService.read(user.uid),
224          builder: (context, snapshot) {
225            return ListView.builder(
226              itemCount: 5,
227              itemBuilder: (context, index) {

```

4. 222번째 줄에 `QuerySnapshot` 이 Import 되어있지 않아 에러가 발생합니다. 빨간 줄이 있는 부분을 클릭한 뒤 Quick Fix(`Ctrl + .`)를 누르고 `Import library` `'package:cloud_firestore/cloud_firestore.dart'` 를 선택해 주세요.

```
220 // 버킷 리스트
221 Expanded(
222   child: FutureBuilder<QuerySnapshot>(
223     future: bucketService.read(),
224     builder: (context, snapshot) {
225       return ListView.builder(
226         itemCount: 5,
```

그러면 맨 위에 `cloud_firestore`가 Import 되면서 에러가 해결됩니다.

```
lib > main.dart > main
1 import 'package:cloud_firestore/cloud_firestore.dart';
2 import 'package:firebase_core/firebase_core.dart';
3 import 'package:flutter/cupertino.dart';
4 import 'package:flutter/material.dart';
5 import 'package:provider/provider.dart';
6
```

5. 이제 문서들을 가져와 보도록 하겠습니다.

코드스니펫을 복사해 225번째 줄 맨 뒤에 붙여 넣어주세요.

▼ [코드스니펫] main.dart / documents 가져오기

```
final document = FirebaseFirestore.instance.collection('documents').doc('1').get();
```



`FutureBuilder`는 `future`로 실행하는 값의 결과를 `builder` 함수에서 `snapshot.data` 를 통해 접근할 수 있습니다.

아래에서는

`snapshot.data` 로 접근하는 경우 `bucketService.read()` 의 응답 값인 `Future<QuerySnapshot>` 에서 `Future`가 벗겨진 `QuerySnapshot?` 를 가져올 수 있습니다. 아직 응답이 안 온 경우 `null`일 수 있으므로 `nullable`(물음표)가 추가됩니다.

```

221     /// 버킷 리스트
222     Expanded(
223       child: FutureBuilder<QuerySnapshot>(
224         future: bucketService.read(user.uid),
225         builder: (context, snapshot) {
226           final documents = snapshot.data?.docs ?? []; // 문서들 가져오기
227           return ListView.builder(
228             itemCount: 5,

```



`final documents = snapshot.data?.docs ?? [];` : 모든 docs를 가져오는데 만약 null인 경우 빈 배열을 반환하기

- docs를 이용하여 화면에 문서들을 보여주도록 하겠습니다. 코드스니펫을 복사해 228(`itemCount`) ~ 231(`isDone`)까지 라인을 지운 뒤 붙여 넣어주세요.

▼ [코드스니펫] main.dart / ListView에 문서 보여주기

```

itemCount: docs.length,

    itemBuilder: (context, index) {
      final doc = docs[index];
      String job = doc.get('job');
      bool isDone = doc.get('isDone')

```



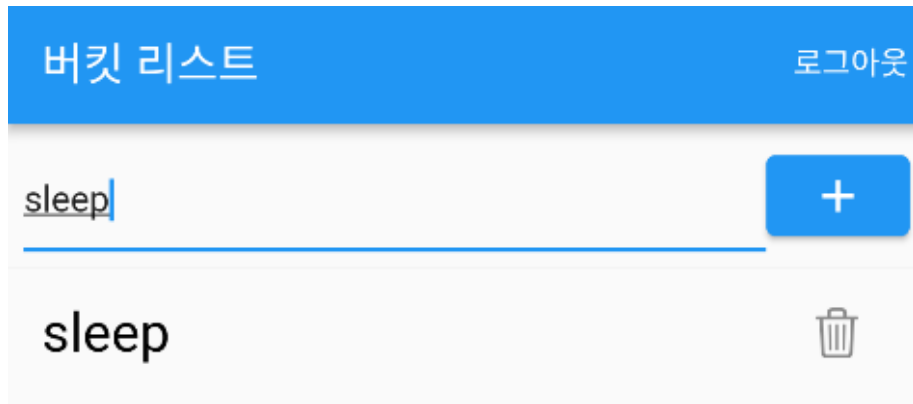
버킷리스트가 저장되어 있는 doc은 `doc.get("key")` 와 같은 방식으로 값을 가져올 수 있습니다.

```

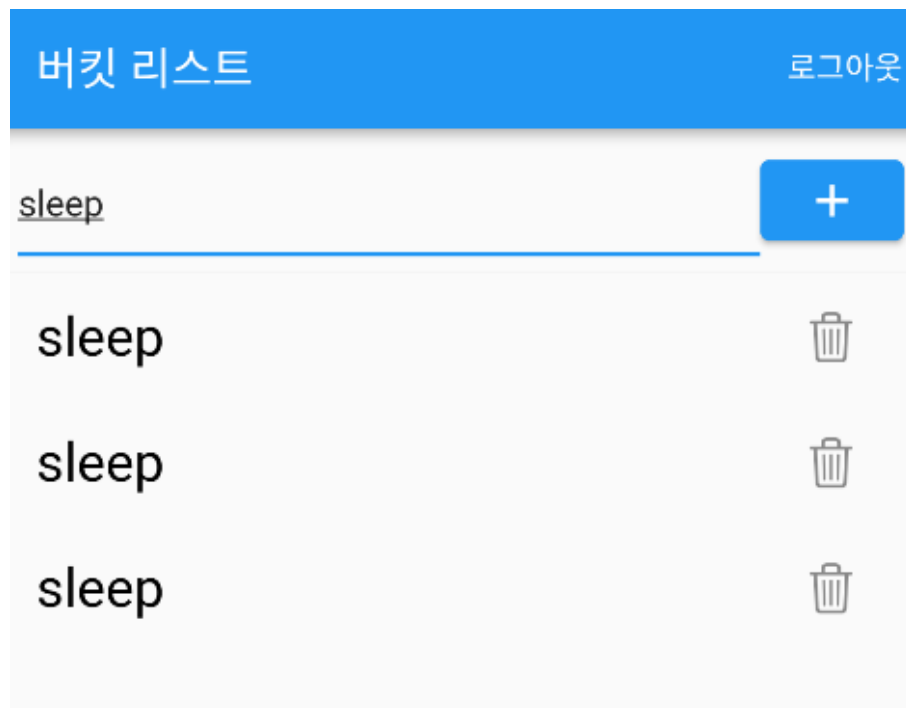
225     builder: (context, snapshot) {
226       final documents = snapshot.data?.docs ?? []; // 문서들 가져오기
227       return ListView.builder(
228         itemCount: documents.length,
229         itemBuilder: (context, index) {
230           final doc = documents[index];
231           String job = doc.get('job');
232           bool isDone = doc.get('isDone');
233           return ListTile(
234             title: Text(
235               job

```

- 그리고 저장을 해주면 에뮬레이터에 아까 작성한 버킷리스트가 출력되는 것을 볼 수 있습니다.



+ 버튼을 여러번 눌러서 버킷을 추가해보면 아래와 같이 생성된 버킷 리스트를 볼 수 있습니다.



💡 다음은 Update 기능을 구현해 보도록 하겠습니다.

▼ 6) Update 만들기

💡 버킷 아이템을 클릭하는 경우, 버킷 완료여부를 변경해 보도록 하겠습니다.

1. 버킷 아이템을 클릭하는 경우, BucketService의 update 함수를 호출하도록 만들어 봅시다. 코드스니펫을 복사해 `main.dart` 파일에 252번째 줄 맨 뒤에 붙여주세요.

▼ [코드스니펫] `main.dart` / `update`

```
bucketService.update(doc.id
```



`doc.id` 로 현재 문서의 고유 식별자를 가져올 수 있습니다.

```
251  ✓
252
253  onTap: () {
254      // 아이템 클릭하여 isDone 업데이트
      bucketService.update(doc.id, !isDone);
  },
```

2. `bucket_service.dart` 파일의 update 로직을 구현해 보도록 하겠습니다. 코드스니펫을 복사해 23번째 라인 맨 뒤에 붙여 넣어주세요.

▼ [코드스니펫] `bucket_service.dart` / `update`

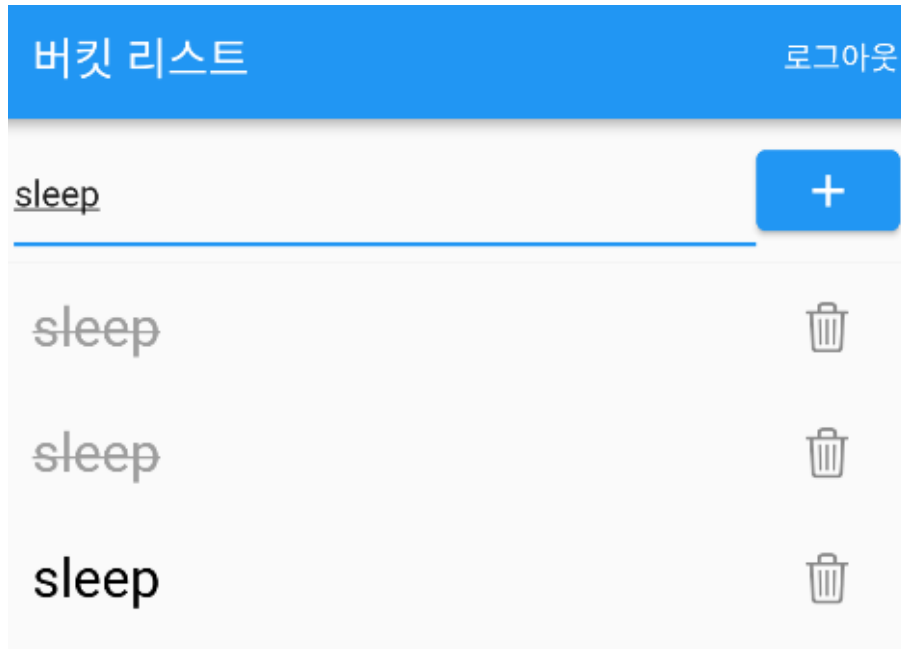
```
await bucketCollection.doc(docId).update({'isDone': isDone});
notifyListeners(); // 화면 갱신
```



24번째 라인과 같이 문서의 id를 전달하여 특정 문서를 업데이트할 수 있습니다.

```
22  ✓ void update(String docId, bool isDone) async {
23      // bucket isDone 업데이트
24      await bucketCollection.doc(docId).update({'isDone': isDone});
25      notifyListeners(); // 화면 갱신
26  }
```

3. `isDone` 상태에 따라 화면에 다르게 보여주는 부분은 이미 구현 되어있습니다. 저장한 뒤 에뮬레이터에서 버킷 리스트를 클릭하여 업데이트가 잘 되는 확인해 주세요.



위와 같이 선택된 항목의 `isDone` 값이 토글 형태로 업데이트 되는 것을 볼 수 있습니다.



다음으로 Delete 기능을 구현해 보도록 하겠습니다.

▼ 7) Delete 만들기

1. `bucket_service.dart` 파일에서 `delete` 함수를 바로 구현하겠습니다. 코드스니펫을 복사해 29번째 라인 맨 뒤에 붙여 넣고 저장해 주세요.

▼ [코드스니펫] `bucket_service.dart` / `delete`

```
await bucketCollection.doc(docId).delete();  
notifyListeners(); // 화면 갱신
```



30번째 라인과 같이 문서의 id를 이용하여 문서를 삭제할 수 있습니다.

```

28 void delete(String docId) async {
29   // bucket 삭제
30   await bucketCollection.doc(docId).delete();
31   notifyListeners(); // 화면 갱신
32 }

```

- 휴지통 아이콘을 누르면 BucketService의 delete 함수를 호출하도록 코드스니펫을 복사해 `main.dart` 의 248번째 라인 맨 뒤에 붙여 넣고 저장해 주세요.

▼ [코드스니펫] main.dart / delete

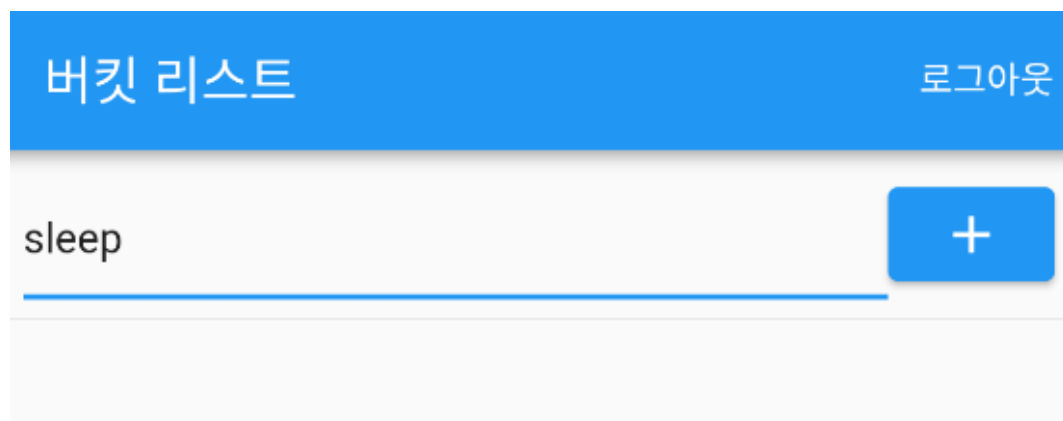
```
bucketService.delete(doc.
```

```

244 // 삭제 아이콘 버튼
245 trailing: IconButton(
246   icon: Icon(CupertinoIcons.delete),
247   onPressed: () {
248     // 삭제 버튼 클릭시
249     bucketService.delete(doc.id);
250   },
251 ), // IconButton

```

- 저장한 뒤 에뮬레이터에서 삭제 아이콘을 눌러보면 삭제가 잘 되는 것을 확인할 수 있습니다.



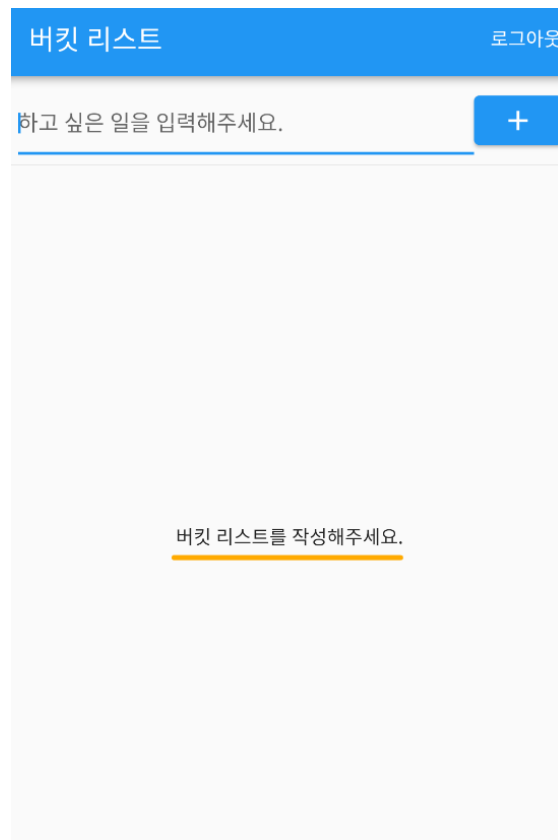
- 마지막으로 모든 버킷 리스트를 삭제하는 경우, 문구를 추가해 보도록 하겠습니다. 코드스니펫을 복사해 `main.dart` 파일의 226번째 맨 뒤에 붙여 넣어주세요.

▼ [코드스니펫] main.dart / empty widget

```
if (documents.isEmpty) {  
  return Center(child: Text("버킷 리  
}
```

```
221 // 버킷 리스트  
222 Expanded(  
223   child: FutureBuilder<QuerySnapshot>(  
224     future: bucketService.read(user.uid),  
225     builder: (context, snapshot) {  
226       final documents = snapshot.data?.docs ?? []; // 문서를 가져오기  
227       if (documents.isEmpty) {  
228         return Center(child: Text("버킷 리스트를 작성해주세요."));  
229       }  
230       return ListView.builder(  
231         itemCount: documents.length,  
232         itemBuilder: (context, index) {
```

모든 버킷 리스트를 삭제해 보면 안내 문구가 잘 나오는 것을 보실 수 있습니다.





여기까지 Firestore에 CRUD하는 기능 구현을 완료하였습니다. Firestore에는 지금 소개해 드린 기본 기능 이외에도 다양한 기능들이 있으니 공식 문서를 한 번 참고해 보세요.

▼ 8) Firestore 보안 규칙 설정



Firestore에는 보안 규칙을 지정하여 특정 데이터에 대한 CRUD 권한을 지정할 수 있습니다.

로그인을 해야만

Create 를 할 수 있고, 그 외에 **Read** , **Update** , **Delete** 는 본인이 작성한 버킷 리스트만 가능하도록 보안 규칙을 수정해 봅시다.

1. Firebase Console에서 Firestore의 **규칙** 탭을 선택해 주세요.

그러면 아래와 같이 처음 Firestore를 설정할 때 만든 누구나 CRUD를 할 수 있도록 설정된 규칙이 보입니다.

```

1 rules_version = '2';
2 service cloud.firestore {
3   match /databases/{database}/documents {
4     match /{document=**} {
5       allow read, write: if
6         request.time < timestamp.date(2022, 1, 4);
7     }
8   }
9 }

```

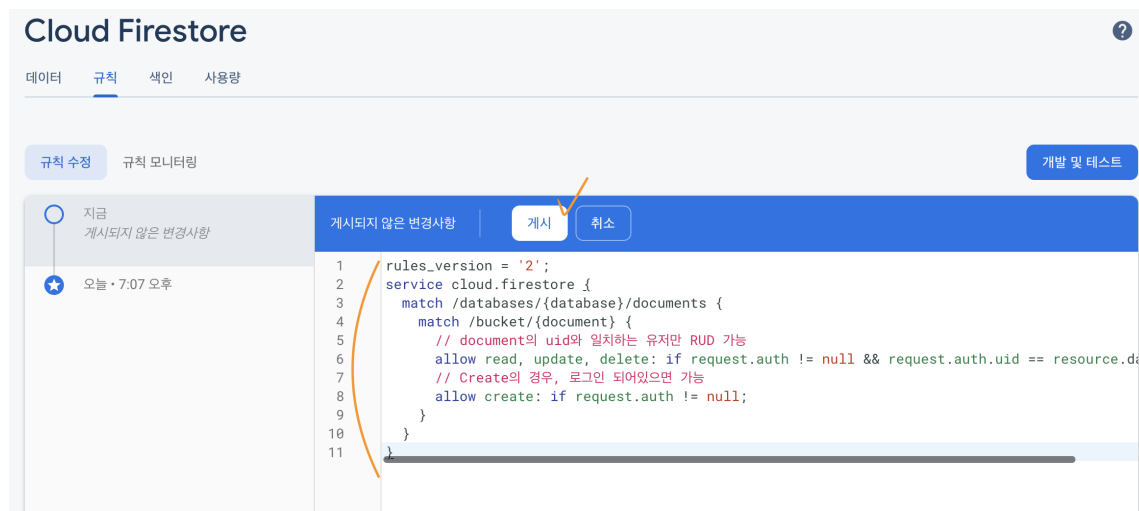
2. 아래 코드스니펫을 복사해서 오른쪽 기존의 모든 규칙을 삭제한 뒤 붙여 넣은 뒤 **게시** 버튼을 눌러주세요.

▼ [코드스니펫] firestore 규칙

```

rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /bucket/{document} {
      // document의 uid와 일치하는 유저만 RUD 가능
      allow read, update, delete: if request.auth != null
      // Create의 경우, 로그인 되어있으면 가능
      allow create: if request.auth != null;
    }
  }
}

```





`/bucket` 밑에 있는 document에 있는 uid 값과 요청자의 uid가 일치하는 경우만 CRUD가 가능

- `request.auth == null` : Firebase Auth 로그인 하지 않음
- `request.auth != null` : Firebase Auth 로그인 됨
- `request.auth.uid` : 로그인한 유저의 uid
- `resource.data.<문서 field 이름>` : 직접 정의한 document 상의 field 이름(여기선 버킷을 작성할 때 추가한 `uid`)

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /bucket/{document} {
      // document의 uid와 일치하는 유저만 RUD 가능
      allow read, update, delete: if request.auth != null;
      // Create의 경우, 로그인 되어있으면 가능
      allow create: if request.auth != null;
    }
  }
}
```



보안 규칙에 대한 상세한 설명은 아래 링크를 참고해주세요.

▼ [코드스니펫] Firestore 보안 규칙 공식 문서

<https://firebase.google.com/docs/rules/basics?hl=ko>

3. 에뮬레이터에서 CRUD를 테스트해보면 잘 작동하는 것을 보실 수 있습니다.

▼ 최종 코드

▼ `main.dart`

```

import 'package:cloud_firestore/cloud_firestore.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';
import 'bucket_service.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized(); // main 함수에서
  await Firebase.initializeApp(); // firebase 앱 시작
  runApp(
    MultiProvider(
      providers: [
        ChangeNotifierProvider(create: (context) => AuthService),
        ChangeNotifierProvider(create: (context) => BucketService),
      ],
      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,
        decoration: InputDecoration(hintText: "이메일"),
    ),

    /// 비밀번호
    TextField(
        controller: passwordController,
        obscureText: false, // 비밀번호 안보이게
        decoration: InputDecoration(hintText: "비밀번호"),
    ),
    SizedBox(height: 32),

    /// 로그인 버튼
    ElevatedButton(
        child: Text("로그인", style: TextStyle(fontWeight: FontWeight.bold)),
        onPressed: () {
            // 로그인
            authService.signIn(
                email: emailController.text,
                password: passwordController.text,
            ).onSuccess: () {
                // 로그인 성공
                ScaffoldMessenger.of(context).showSnackBar(
                    content: Text("로그인 성공"),
                );

                // HomePage로 이동
                Navigator.pushReplacement(
                    context,
                    MaterialPageRoute(builder: (context) => HomePage()),
                );
            },
            onError: (err) {
                // 에러 발생
                ScaffoldMessenger.of(context).showSnackBar(
                    content: Text(err),
                );
            },
        ),
    ),

```

```

        },
      );
    },
  ),

  /// 회원가입 버튼
  ElevatedButton(
    child: Text("회원가입", style: TextStyle(font
onPressed: () {
    // 회원가입
    authService.signUp(
      email: emailController.text,
      password: passwordController.text,
      onSuccess: () {
        // 회원가입 성공
        ScaffoldMessenger.of(context).showSn
          content: Text("회원가입 성공"),
        ));
      },
      onError: (err) {
        // 에러 발생
        ScaffoldMessenger.of(context).showSn
          content: Text(err),
        ));
      },
    );
  },
),
],
),
),
),
);
},
);
}
}

```

```

/// 홈페이지
class HomePage extends StatefulWidget {
  const HomePage({Key? key}) : super(key: key);

  @override
  State<HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  TextEditingController jobController = TextEditingControlle

  @override
  Widget build(BuildContext context) {
    final authService = context.read<AuthService>();
    final user = authService.currentUser()!;
    return Consumer<BucketService>(
      builder: (context, bucketService, child) {
        return Scaffold(
          appBar: AppBar(
            title: Text("버킷 리스트"),
            actions: [
              TextButton(
                child: Text(
                  "로그아웃",
                  style: TextStyle(
                    color: Colors.white,
                  ),
                ),
                onPressed: () {
                  // 로그아웃
                  context.read<AuthService>().signOut();

                  // 로그인 페이지로 이동
                  Navigator.pushReplacement(
                    context,
                    MaterialPageRoute(builder: (context) =>

```

```

        },
      ),
    ],
  ),
  body: Column(
    children: [
      /// 입력창
      Padding(
        padding: const EdgeInsets.all(8),
        child: Row(
          children: [
            /// 텍스트 입력창
            Expanded(
              child: TextField(
                controller: jobController,
                decoration: InputDecoration(
                  hintText: "하고 싶은 일을 입력해주세요.",
                ),
              ),
            ),
          ],
        ),
      ),
      /// 추가 버튼
      ElevatedButton(
        child: Icon(Icons.add),
        onPressed: () {
          // create bucket
          if (jobController.text.isNotEmpty) {
            bucketService.create(jobController
          }
        },
      ),
    ],
  ),
  Divider(height: 1),

  /// 버킷 리스트

```

```

Expanded(
  child: FutureBuilder<QuerySnapshot>(
    future: bucketService.read(user.uid),
    builder: (context, snapshot) {
      final documents = snapshot.data?.docs
      if (documents.isEmpty) {
        return Center(child: Text("버킷 리스트"))
      }
      return ListView.builder(
        itemCount: documents.length,
        itemBuilder: (context, index) {
          final doc = documents[index];
          String job = doc.get('job');
          bool isDone = doc.get('isDone');
          return ListTile(
            title: Text(
              job,
              style: TextStyle(
                fontSize: 24,
                color: isDone ? Colors.grey
                decoration: isDone
                  ? TextDecoration.lineThr
                  : TextDecoration.none,
              ),
            ),
            // 삭제 아이콘 버튼
            trailing: IconButton(
              icon: Icon(CupertinoIcons.delete),
              onPressed: () {
                // 삭제 버튼 클릭시
                bucketService.delete(doc.id)
              },
            ),
            onTap: () {
              // 아이템 클릭하여 isDone 업데이트
              bucketService.update(doc.id, !
            ),

```

```

        );
      },
    );
  )),
),
],
),
);
},
);
}
}

```

▼ `auth_service.dart`

```

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)
    return FirebaseAuth.instance.currentUser;
  }

  void signUp({
    required String email, // 이메일
    required String password, // 비밀번호
    required Function() onSuccess, // 가입 성공시 호출되는 함수
    required Function(String err) onError, // 에러 발생시 호출도
  }) async {
    // 회원가입
    // 이메일 및 비밀번호 입력 여부 확인
    if (email.isEmpty) {
      onError("이메일을 입력해 주세요.");
      return;
    } else if (password.isEmpty) {
      onError("비밀번호를 입력해 주세요.");
    }
  }
}

```

```

        return;
    }

    // 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());
    }
}

void signIn({
    required String email, // 이메일
    required String password, // 비밀번호
    required Function() onSuccess, // 로그인 성공시 호출되는 함수
    required Function(String err) onError, // 에러 발생시 호출도
}) async {
    // 로그인
    if (email.isEmpty) {
        onError('이메일을 입력해주세요. ');
        return;
    } else if (password.isEmpty) {
        onError('비밀번호를 입력해주세요. ');
        return;
    }

    // 로그인 시도

```



```

    try {
      await FirebaseAuth.instance.signInWithEmailAndPassword(
        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());
    }
  }

  void signOut() async {
    // 로그아웃
    await FirebaseAuth.instance.signOut();
    notifyListeners(); // 로그인 상태 변경 알림
  }
}

```

▼ bucket_service.dart

```

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

class BucketService extends ChangeNotifier {
  final bucketCollection = FirebaseFirestore.instance.collec

  Future<QuerySnapshot> read(String uid) async {
    // 내 bucketList 가져오기
    return bucketCollection.where('uid', isEqualTo: uid).get
  }
}

```

```

void create(String job, String uid) async {
  // bucket 만들기
  await bucketCollection.add({
    'uid': uid, // 유저 식별자
    'job': job, // 하고싶은 일
    'isDone': false, // 완료 여부
  });
  notifyListeners(); // 화면 갱신
}

void update(String docId, bool isDone) async {
  // bucket isDone 업데이트
  await bucketCollection.doc(docId).update({'isDone': isDo
  notifyListeners(); // 화면 갱신
}

void delete(String docId) async {
  // bucket 삭제
  await bucketCollection.doc(docId).delete();
  notifyListeners(); // 화면 갱신
}
}

```