Firebase with Dart

Cloud NoSQL Database & Backend-as-a-Service

What is Firebase?

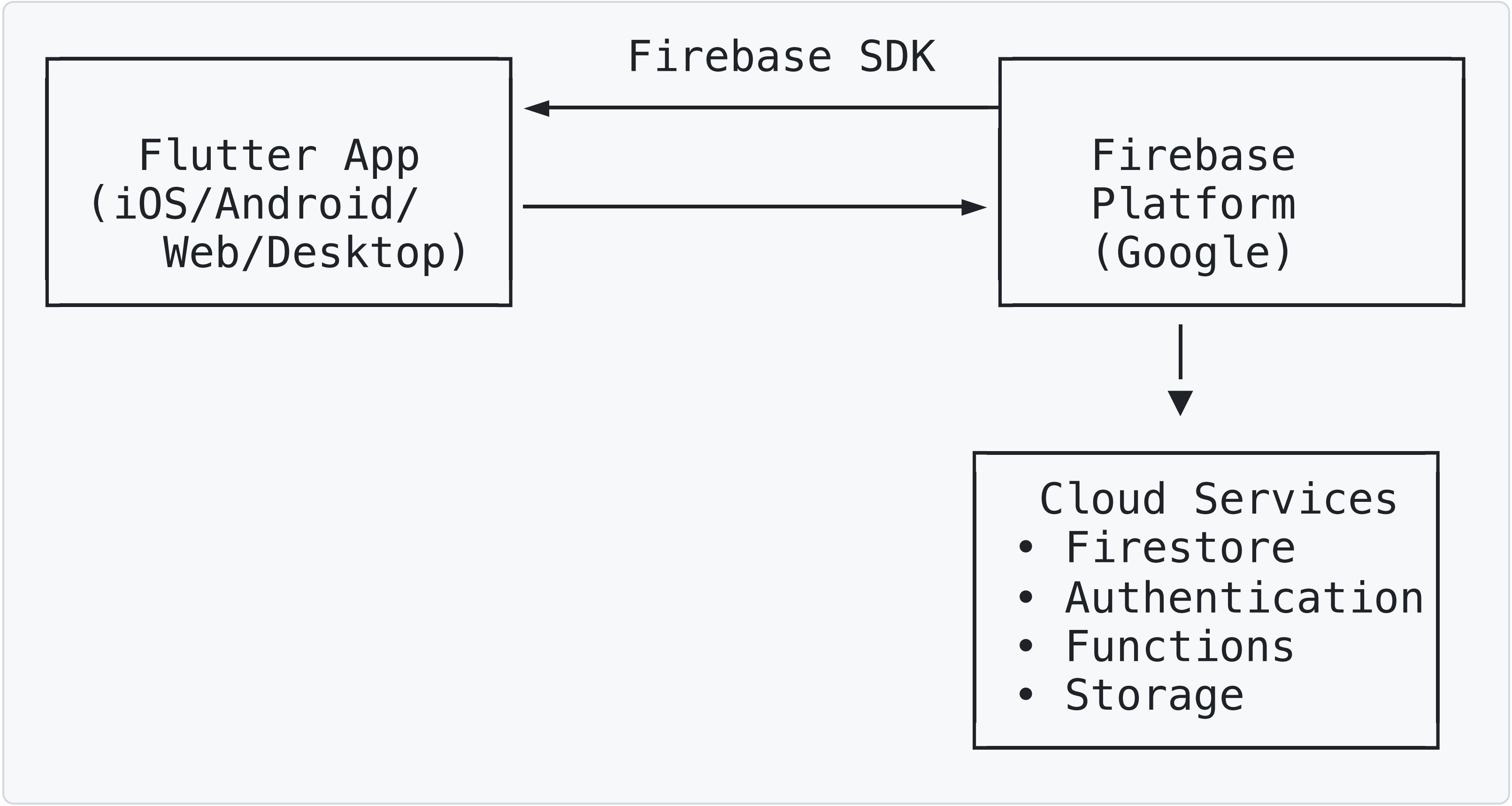
- **Google's Backend-as-a-Service** platform
- **Cloud Firestore** - NoSQL document database
- **Real-time synchronization** across devices
- **Built-in authentication** and security
- **Cross-platform** - iOS, Android, Web, Desktop

Firebase Services:

- Firestore (Database)
- Authentication
- Cloud Functions
- Cloud Storage
- Analytics
- Hosting
- Push Notifications

Used by: WhatsApp, Spotify, Airbnb, The New York Times

Architecture Overview



Benefits:

- No server management required
- Automatic scaling and backup
- Real-time data synchronization
- Global edge network

foo project

- In this project, we use Firebase.
- Use "Firebase_Quick_Start_Guide.md" for registering and making Firebase/firestore project.
    - We assume that users already made the foobar project.

Firebase Project

- Make sure you installed firebase CLI tool.
- Get your Firebase "Project ID" for your Dart project.

```
> firebase projects:list

Preparing the list of your Firebase projects
┌──────────────────────┬──────────────────────┬────────────────┬──────────────────────┐
│ Project Display Name │ Project ID           │ Project Number │ Resource Location ID │
├──────────────────────┼──────────────────────┼────────────────┼──────────────────────┤
│ foobar               │ foobar-YOUR_ID       │ 827133271343   │ [Not specified]      │
└──────────────────────┴──────────────────────┴────────────────┴──────────────────────┘
```

Project Setup

```
# pubspec.yaml
dependencies:
  firedart: ^0.9.8
```

Then download the dependencies.

```
dart pub get
```

Import the package.

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

Firestore

- Using Firebase in Dart is working with **Cloud Firestore** - NoSQL document database
- Firebase works with **collections** and **documents** (not tables and rows)
- Firebase is **cloud-based** - works on web, mobile, and desktop
- Firebase provides **real-time updates** and **offline support**

Initialization

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

Future<void> main() async {
  // Initialize Firestore
  // with your project ID (not Firebase app)
  Firestore.initialize("foobar-YOUR_ID");

  // Get Firestore instance
  final firestore = Firestore.instance;
  ...
```

CRUD

- Use `add()` to create new documents with auto-generated IDs
- Use `set()` to create/update documents with specific IDs
- Use `get()` to retrieve documents once
- Use `snapshots()` to listen for real-time changes

Create: *`collection().add()`*

- Creates document with auto-generated ID
- Safe: No ID conflicts

```
Document addedDoc = await firestore
    .collection('foo')
    .add(generateRandomData());
```

Create/Update: `collection().document().set()`

- Uses specific document ID
- Creates new or overwrites existing
- Risk: Can overwrite existing data

```
await FirebaseFirestore.instance
  .collection('foo')
  .document(foo.id)
  .set(foobar.toMap());
```

Read Once: `collection().document().get()`

- Fetches document data one time
- Good for displaying current state
- Efficient: Single network request

```
Document retrievedDoc = await firestore
    .collection('foo')
    .document(foo.id)
    .get();
```

Read Live: `collection().snapshots()`

- Creates real-time stream of changes
- Updates automatically when data changes
- Cost: Continuous connection and billing

```
Stream<QuerySnapshot> stream = firestore
  .collection('foo')
  .snapshots();
```

Update Fields: `doc().update()`

- Updates specific fields only
- Preserves other existing fields
- Efficient: Only changes specified data

```
await firestore
  .collection('foo')
  .document(foo.id)
  .update({'bar': 21, 'foo': 'Data Science'});
```

Delete Document: `doc().delete()`

- Completely removes document
- Cannot be undone
- Risk: Permanent data loss

```
await FirebaseFirestore.instance
  .collection('foo')
  .document(foo.id)
  .delete();
```

foobar project

- In this project, we use foobar data model to make Dart firebase application.
- Compared to the foo project that aims to understand firebase operation in Dart, foobar project is well designed with OOP.

- Foobar data model

```
class FooBar {
  // Document ID from Firebase
  // (nullable for new documents)
  final String? id;
  final String foo;      // String field
  final int bar;         // Integer field

  /// Constructor with required fields
  FooBar({
    this.id,
    required this.foo,
    required this.bar,
  });
```

```
Map<String, dynamic> toMap() {
  return {
    'foo': foo,
    'bar': bar,
  };
}
static FooBar fromMap(Map<String, dynamic> map,
[String? documentId]) {
  return FooBar(
    id: documentId,
    foo: map['foo'] ?? '',
    bar: map['bar'] ?? 0,
  );
}
```

- String? documentId is an optional positional parameter.

```
FooBar copyWith({
  String? id, String? foo, int? bar,
}) {
  return FooBar(
    id: id ?? this.id,
    foo: foo ?? this.foo,
    bar: bar ?? this.bar,
  );
}
```

- Create a copy of this FooBar with some fields updated.
- Useful for update operations

Processing ID

- In our data model, we have the `id`, but we don't set this value, but Firebase automatically assigns the value.
- The retrieved doc from Firebase has id and map components.

```
Document doc = await firestore
    .collection('foo')
    .document(foo.id)
    .get();
```

We create a new Dart object from `doc.map` and `doc.id`.

```
static FooBar fromMap(Map<String, dynamic> map, [String? documentId]) {
  return FooBar(
    id: documentId,
    foo: map['foo'] ?? '',
    bar: map['bar'] ?? 0,
  );
}

FooBar foobar = FooBar.fromMap(doc.map, doc.id);
```

CRUD

Service Class for Firestore Operations

```
class FooBarCrudFirebase {
  late Firestore _firestore;
  final String _collectionName = 'foobars';

  Future<void> initialize({String projectId = 'foobar-PROJECT'})
  async {
    try {
      Firestore.initialize(projectId);
      _firestore = Firestore.instance;
    } catch (e) {
      rethrow;
    }
  }
```

CREATE: Add new student to Firestore

```
Future<FooBar?> create(FooBar foobar) async {
  try {
    Document doc = await _firestore
        .collection(_collectionName)
        .add(foobar.toMap());
    // Return the FooBar with the new ID
    return foobar.copyWith(id: doc.id);
  } catch (e) {
    return null;
  }
}
```

- Returned FooBar object has auto-generated ID.

READ: Get a single FooBar by ID

```dart
Future<FooBar?> read(String id) async {
  try {
    print('📖 Reading FooBar with ID: $id');
    // Get document by ID
    Document doc = await _firestore
        .collection(_collectionName)
        .document(id).get();
    // Convert to FooBar object
    FooBar foobar = FooBar.fromMap(doc.map, doc.id);
    print('✅ FooBar retrieved: $foobar');
    return foobar;
  } catch (e) {
    print('❌ Error reading FooBar: $e');
    return null;
  }
}
```

READ: Get all FooBar documents

```
Future<List<FooBar>> readAll() async {
  try {
    print('📚 Reading all FooBar documents...;');
    // Get all documents from collection
    List<Document> docs = await _firestore
        .collection(_collectionName)
        .get();
    // Convert to FooBar objects
    List<FooBar> foobars = docs
        .map((doc) => FooBar.fromMap(doc.map, c.id))
        .toList();
    print('✅ Retrieved ${foobars.length} oBar documents');
    return foobars;
  } catch (e) {
    print('❌ Error reading all FooBars: $e');
    return [];
  }
}
```

READ: Query FooBar documents where bar value equals the given number

```
Future<List<FooBar>> readByBar(int barValue) aync {
  try {
    print('🔍 Querying FooBars where bar = arValue');
    // Query documents with filter
    List<Document> docs = await _firestore
        .collection(_collectionName)
        .where('bar', isEqualTo: barValue)
        .get();
    // Convert to FooBar objects
    List<FooBar> foobars = docs
        .map((doc) => FooBar.fromMap(doc.map, c.id))
        .toList();
    print('✅ Found ${foobars.length} FooBars th bar = $barValue');
    return foobars;
  } catch (e) {
    print('❌ Error querying FooBars: $e');
    return [];
  }
}
```

UPDATE: Modify an existing FooBar document

```dart
Future<bool> update(String id, FooBar datedFoobar) async {
  try {
    print('✏️ Updating FooBar with ID: $id');
    print('   New data: $updatedFoobar');
    // Update document
    await _firestore
        .collection(_collectionName)
        .document(id)
        .update(updatedFoobar.toMap());
    print('✅ FooBar updated successfully');
    return true;
  } catch (e) {
    print('❌ Error updating FooBar: $e');
    return false;
  }
}
```

UPDATE: Partially update specific fields

```
Future<bool> updateFields(String id, p<String, dynamic> updates) async {
  try {
    print('✏️ Updating FooBar fields for ID: d');
    print('   Updates: $updates');
    // Update specific fields
    await _firestore
        .collection(_collectionName)
        .document(id)
        .update(updates);
    print('✅ FooBar fields updated ccessfully');
    return true;
  } catch (e) {
    print('❌ Error updating FooBar fields: ');
    return false;
  }
}
```

DELETE: Remove a FooBar document

```
Future<bool> delete(String id) async {
  try {
    print('🗑️ Deleting FooBar with ID: $id');
    // Delete document
    await _firestore
        .collection(_collectionName)
        .document(id)
        .delete();
    print('✅ FooBar deleted successfully');
    return true;
  } catch (e) {
    print('❌ Error deleting FooBar: $e');
    return false;
  }
}
```

Utility functions

```
  Future<int> count() async {
```

Firebase and Flutter

- To use Firebase with Flutter, we need to add more configuration files.
- Use "Firebase_Quick_Start_Guide.md" for detailed the installation and configuration.

Firebase CLI tools

One time installation.

```
dart pub global activate flutterfire_cli
```

For each Flutter project that uses Firebase, we need to configure to use Firebase.

```
flutterfire configure
```

pubspec.yaml

For flutter applications:

```yaml
dependencies:
  firebase_core: ^2.24.2
  cloud_firestore: ^4.13.6
```

For web applications, we add:

```yaml
  # Add this for web support
  firebase_core_web: ^2.10.0
  cloud_firestore_web: ^3.8.10
```

Developing Flutter Applications + Firebase

1. Make sure "lib/firebase_options.dart" file is generated from `flutterfire configure`.
2. Add dependencies and intialization code.

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

void main() async {
  WidgetsFlutterBinding.ensureInitialized();

  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  runApp(MyApp());
}
```

Web Applications

- Use "database/firebase/foobar_flutter_webapp" as an example.
- Update "web/index" for accessing Firebase from JavaScript.

Developing other platforms + Firebase

- Use "database/firebase/foobar_flutter_app" as an example.
- Make sure you use the correct OS version to support (mac/ios).
- Make sure the app can use network (mac/ios).

Databases

Use Case	IndexedDB	Firebase	SQLite	PocketBase
Browser-only apps	✅ Perfect	⚠️ Overkill	❌ Not available	❌ Not available
Offline-first web	✅ Excellent	✅ Smart sync	❌ Not available	❌ No offline
Large data storage	✅ Good (250MB+)	⚠️ Expensive	✅ Unlimited	⚠️ Server dependent
Complex queries	❌ Limited	✅ Rich NoSQL	✅ Full SQL	✅ REST API
Real-time sync	❌ Manual	✅ Automatic	❌ Manual	✅ Built-in
Multi-device sync	❌ No	✅ Automatic	❌ Manual	✅ Automatic
Learning curve	⚠️ Moderate	✅ Easy	✅ Simple	✅ Easy

Decision Framework

- **Choose PocketBase** for: Self-hosted real-time apps, educational projects, MVPs, data control`
- **Choose IndexedDB** for: Browser-only applications, offline-first web apps, client-side caching

Choose SQLite for: Single-user apps, offline-first, embedded applications

> **Choose Firebase** for: Global scale, automatic scaling, rapid development without hosting

Firebase Limitations

Database Structure Limitations

- No complex queries or JOINs across collections
- Maximum document size: 1 MB
- Limited filtering (max 30 composite indexes)
- Denormalization required → data duplication

Performance Limitations

- Maximum sustained writes: 10,000/second per database
- Single document: 1 write/second sustained
- No server-side aggregations
- Limited offline query capabilities

Cost Limitations

- **Reads:** $0.36 per 100K documents
- **Writes:** $1.08 per 100K documents
- **Storage:** $0.18/GB/month
- Bandwidth charges for large documents

Feature Limitations

- No transactions across multiple collections
- No stored procedures or triggers
- Limited local development tools
- Vendor lock-in with Google ecosystem

Firebase Offline Support

- **Local caching**: Data is stored on device even when offline
- **Automatic sync**: Changes are synced when connection is restored
- **Read operations**: Continue to work using cached data
- **Write operations**: Queued locally and replayed once online
- **Cross-platform**: Works on iOS, Android, and Web