CMPS 312

Firebase Cloud Services

Firestore Database

Authentication

Storage

**Dr. Abdelkarim Erradi**

CSE@QU

Outline

1. Firestore Data Model

2. Firestore CRUD Operations

3. Firebase Storage

4. Access Image Gallery and Camera

5. Firebase Authentication

Firebase Cloud Services

- Firebase is a **cloud platform** offering many **services** that work together as a backend server infrastructure for mobile/web apps

- We will focus on introducing:

  Firestore: store/query documents in collections

  Storage: store and retrieve files

  Firebase Authentication: secure user authentication using various identity provides (e.g., email/password, Google Auth)

Firestore Data Model
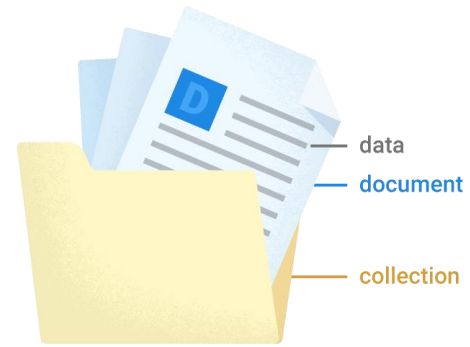

Data

Collection
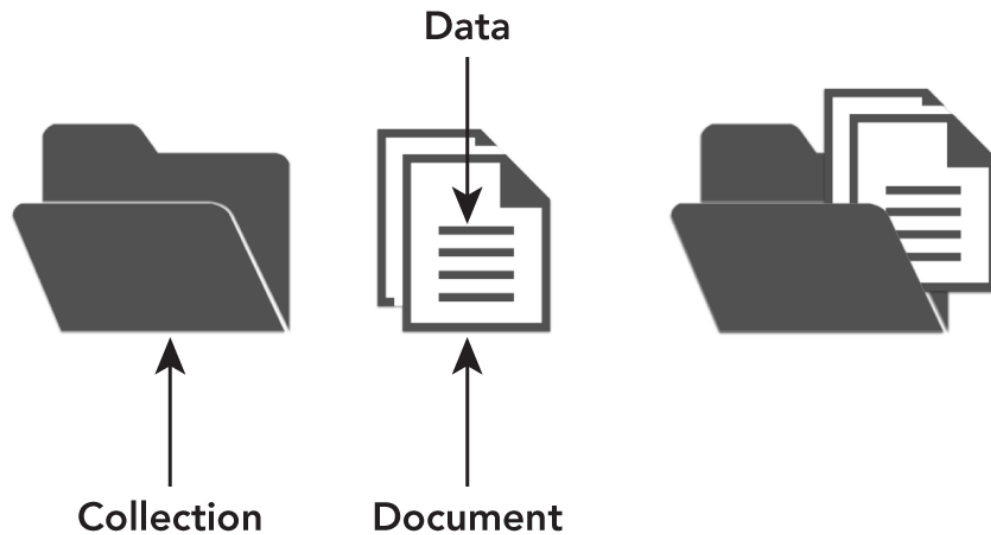
Document

Firestore Database

- Cloud-hosted **scalable** database to manage app data
  - No need to set up or maintain backend servers
- Provides real-time updates and offline support
- Uses a **document-oriented** data model
  - Data is organized in **collections**
  - Each collection contains documents, which can further include subcollections
  - Allowing you to build flexible and hierarchical data structures
- **NoSQL** (does not use SQL as a query language)
- Access controlled with **security rules**
- Includes a [free tier](#) (1 GB data, 50K reads/day and 20K writes/day) then pay as you use

Data Model

- Firestore is **Document Oriented Database**

    o Stores data as **documents** that utilizes a flexible, JSON-like data model

        • instead of rows and columns as done in a relational database

    o Documents are grouped into **collections**

    o **API to query** and manage documents

- Better alternative data management solution for Mobile/Web applications (compared to using a Relational DB)

    o Real-time synchronization capabilities

    o Scalable data management

Document

```
{
  "isbn" : 123,
  "title": "Mr Bean and the Forty Thieves",        property : value
  "category": "Fun",
  "pages": 250
  "authors": ["Mr Bean", "Juha Dahak"],             property : array
  "publisher": {
                "name": "MrBeanCo",                 property : map
                "country": "UK"
               }
}
```

- **Document = JSON**-like object
- **Document = set of key-value pairs**
- **Document = basic unit of data** in Firestore
  - You can only fetch a document not part of it
- Analogous to **row** in a relational database
- Size limit to **1 MB** per document
- A document can optionally point to subcollections
- A Document **cannot** point to another document

Data Types

- Cloud Firestore supports a variety of data types for values:

  ○ boolean, number, string,

  ○ geo point, binary blob, and timestamp

  ○ arrays, nested objects (called maps) to structure a complex object (e.g., address) within a document
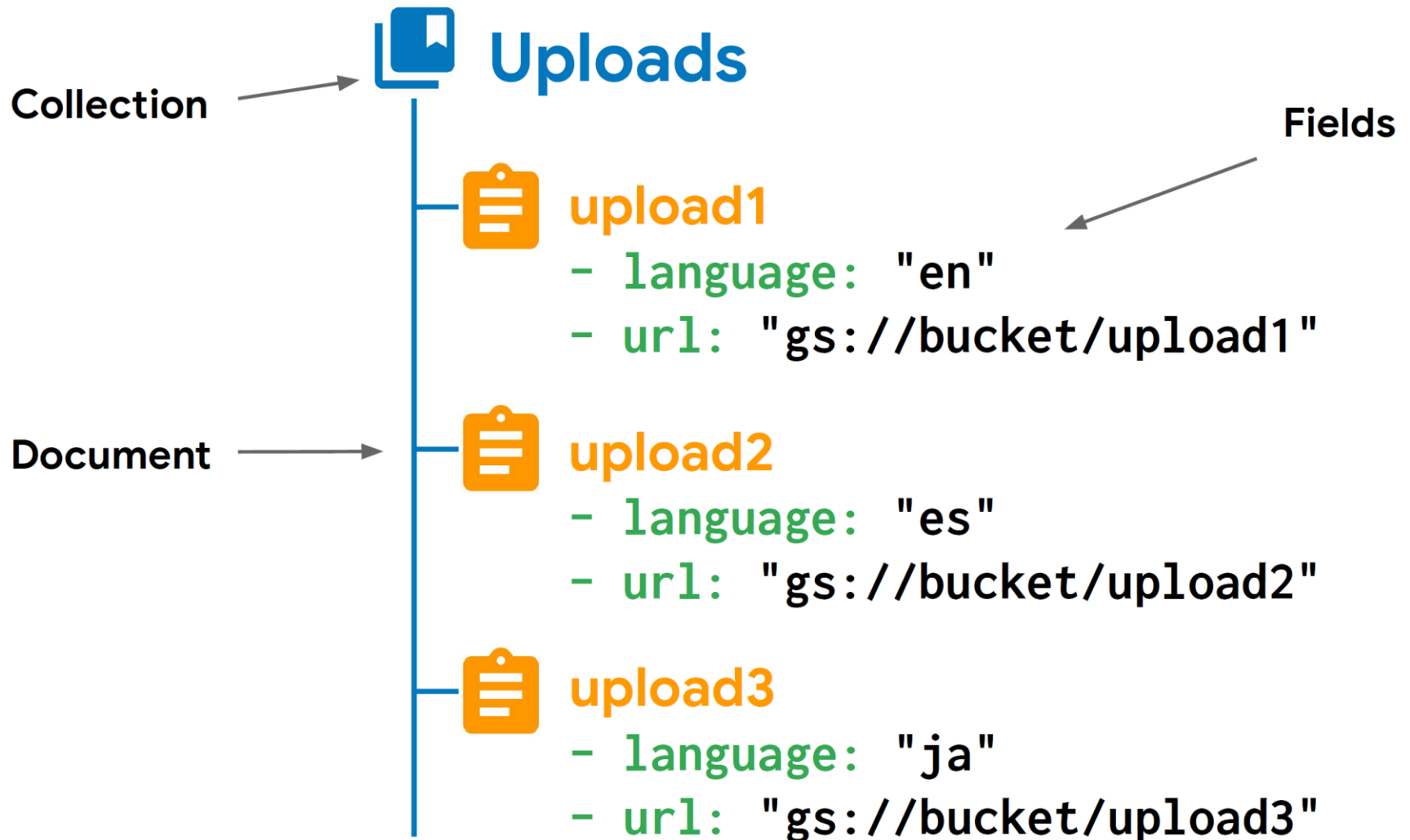
Document

bird_type: "swallow"
airspeed: 42.733
coconut_capacity: 0.62
isNative: false
icon: <binary data>
vector:
        {x: 36.4255,
        y: 25.1442,
        z: 18.8816}
distances_traveled:
        [42, 39, 12, 42]

Collection

```
{
  "isbn" : "123",
  "title": "Mr Bean and the Forty Thieves",
  "authors": ["Mr Bean", "Juha Dahak"],
  "publisher": {"name": "MrBeanCo", "country": "UK"},
  "category": "Fun",
  "pages": 250
}
```

- **Collection = container** for documents

- Analogous to **table** in a relational database

- **Does not enforce** a schema

- Documents in a collection usually **have similar purpose** but they may have slightly different schema

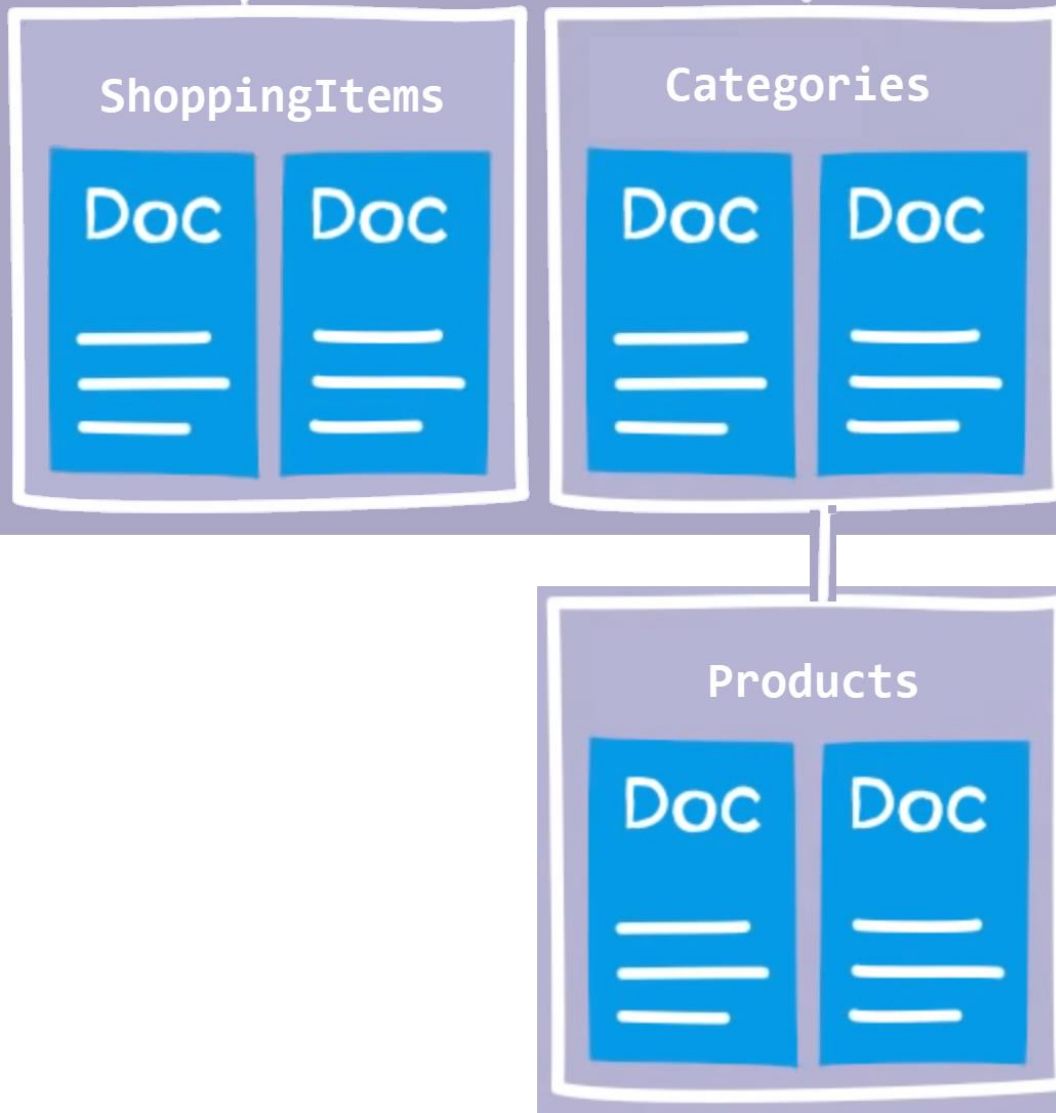- A collection <u>cannot</u> contain other collections

Example Collection & Documents

Uploads

Collection →

Fields

upload1
- language: "en"
- url: "gs://bucket/upload1"

Document →

upload2
- language: "es"
- url: "gs://bucket/upload2"

upload3
- language: "ja"
- url: "gs://bucket/upload3"

Firestore Root

ShoppingItems

Doc   Doc

Categories

Doc   Doc

Products

Doc   Doc

Shopping List App

- Database with 2 **top-level** collections: ShoppingItems and Categories

- Each category document has a Products sub-collection

Document Identifiers

- Documents within a collection have **unique identifiers**
  - You can provide your own keys, such as using the **email** as a unique identifier for users
  - You can let Cloud Firestore assign a random IDs

- You do not need to "create" or "delete" collections
  - A collection gets created after you create the first document in a collection
  - A collection is deleted when you delete all the documents in a collection

- Access a document using its **collection** and its doc **Id**

```
final FirebaseFirestore db = FirebaseFirestore.instance;
final u1DocRef = await db.collection("users").doc("u1@test.com").get();
```

Subcollections

- A subcollection is a collection associated with a specific document
  - E.g., A subcollection called messages for every room document in the rooms collection

📚 rooms

  📖 roomA

`name : "my chat room"`

    📚 messages

      📖 message1

`from : "alex"`
`msg : "Hello World!"`
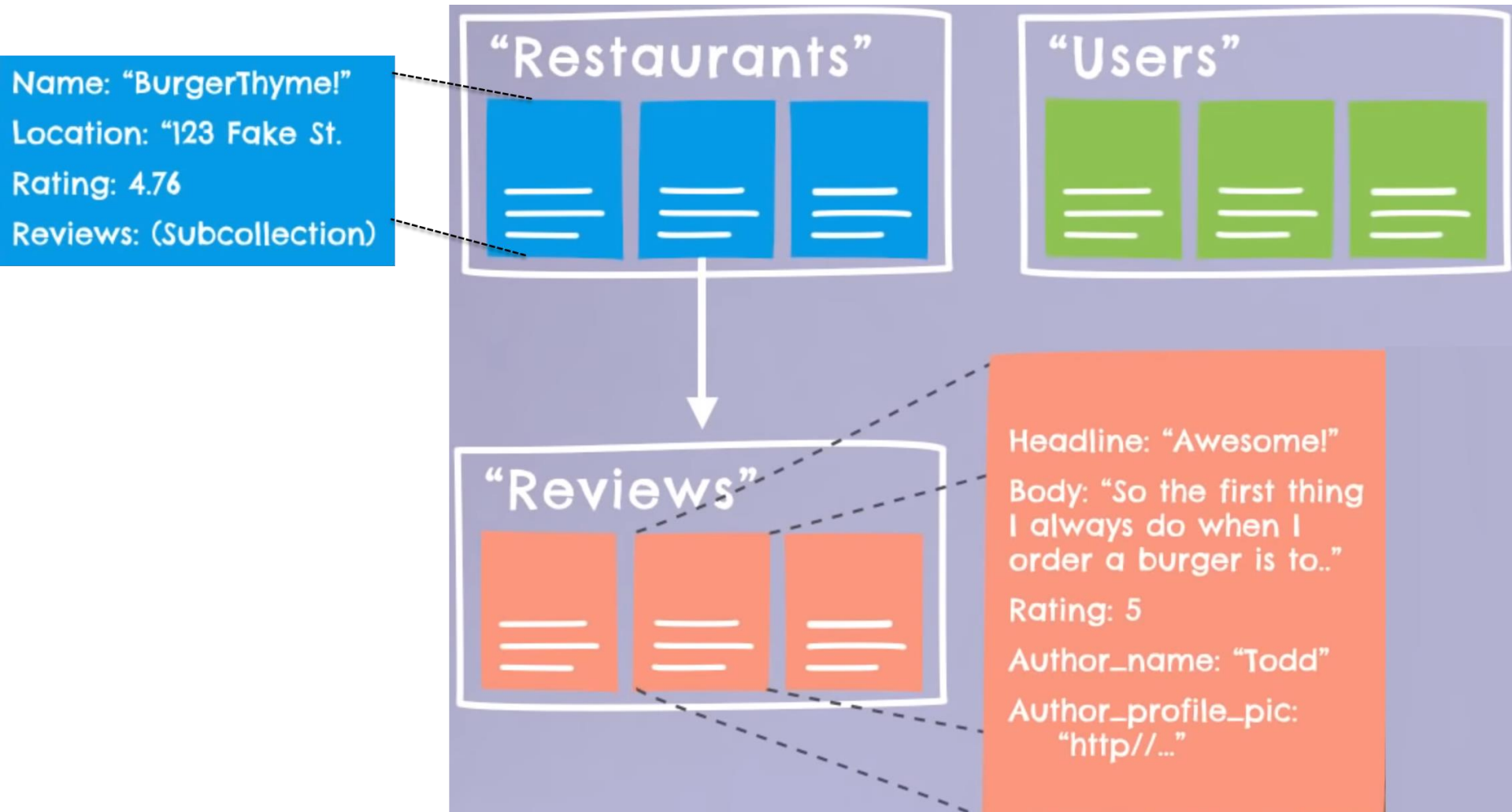
      📖 message2

`...`

  📖 roomB

`...`

- Get a reference to a message in the subcollection

```
final FirebaseFirestore db = FirebaseFirestore.instance;

final message1DocRef =
      db.collection("rooms").document("roomA")
        .collection("messages").document("message1");
```
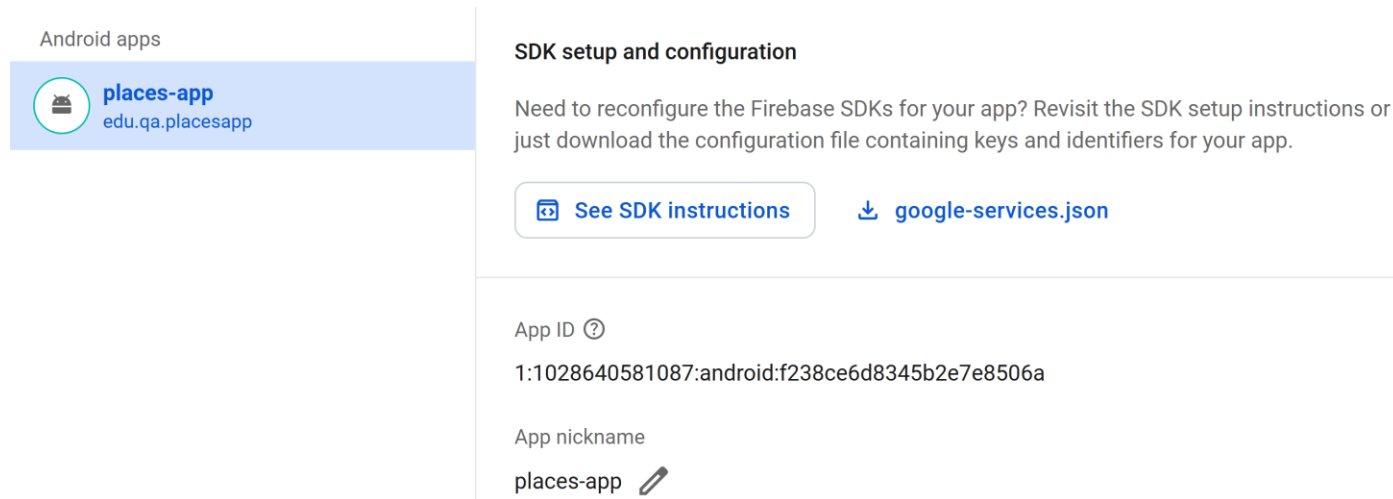
Example Restaurant Review App

Add Firebase to your Flutter project

- Login to https://console.firebase.google.com/

- Create a **project** (give it a meaningful name, see the steps at this link)
  - ○ to keep it simple disable Google Analytics for the project

- Create Firestore database (see the steps at this link)

- Select the **Project settings** and add an Android app

Android apps

🤖 **places-app**
   edu.qa.placesapp

SDK setup and configuration

Need to reconfigure the Firebase SDKs for your app? Revisit the SDK setup instructions or just download the configuration file containing keys and identifiers for your app.

⊡ See SDK instructions   ⬇ google-services.json

App ID ⑦

1:1028640581087:android:f238ce6d8345b2e7e8506a

App nickname

places-app ✏

- Download `google-services.json` and place it under `\android\app` subfolder

Dependencies

- Add to **pubspec.yaml**:

```
dependencies:
  firebase_core: ^3.6.0
  cloud_firestore: ^5.4.4
```

- Install [Firebase CLI](#)

npm install -g firebase-tools

- Install FlutterFire CLI

dart pub global activate flutterfire_cli
([add the folder](#) in the Warning message to Windows/MacOS System's environment path)

- Generate **firebase.json** and **firebase_options.dart** config files to connect to Firebase

firebase login
flutterfire configure

Easier way to Flutter app to use Firebase

- Add to `<project>/build.gradle`

```
plugins {
  // Add the dependency for the Google services Gradle plugin
  id 'com.google.gms.google-services' version '4.4.2' apply false
}
```

- Add to `<project>/app/build.gradle`

```
plugins {
    ...
    // Add the Google services Gradle plugin
    id 'com.google.gms.google-services'
}

  defaultConfig { ...
      minSdk = 23
```

Set the min SDK version to 23

Initialize Firebase in Your Flutter App

- Open the main function initialize Firebase before running the app

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(
    const ProviderScope(
      child: MyApp(),
    ),
  );
}
```

Firestore CRUD Operations

Create Data Classes Mapped to Firestore Docs

- Normal **classes** having the same structure as Firebase docs
  - Map fields to/from Firestore using a fromJson and toJson method

```dart
class Category {
  String id;
  String name;
  // Default constructor
  Category({this.id = '', this.name = ''});
  factory Category.fromJson(Map<String, dynamic> data) {
    return Category(
      id: data['id'],
      name: data['name'] ?? '',
    );
  }
  Map<String, dynamic> toJson() {
    return {
      'name': name,
    };
  }
}
```

Query – return all documents

- Using **collection reference** use the **.get** method to return the collection documents
  - You can sort the results using **.orderBy**
  - Use the same technique to get documents from a subcollection associated with a particular document

```
Future<List<Product>> fun getProducts(categoryId: String) async {
  final queryResult = await categoryCollectionRef.doc(categoryId).collection("products")
                      .orderBy("name", Query.Direction.DESCENDING).get();
  List<Product> products = queryResult.docs.map((doc) =>
          Product.fromJson(doc.data() as Map<String, dynamic>)
        ).toList();
  return products;
}
```

Query – filer using .where

- Use **.where** 🔗 to filter the documents to return from a collection

```
final citiesRef = db.collection("cities");
final stateQuery = citiesRef.where("state", isEqualTo: "CA");
final populationQuery = citiesRef.where("population",
               isLessThan: 100000);
final nameQuery = citiesRef.where("name", isEqualTo: "Doha");
final notCapitals = citiesRef.where("capital", isNotEqualTo: true);
final cities = citiesRef.where("country", whereIn: ["USA", "Japan"]);


Future<Category?> getCategory(categoryName: String) async {
  final queryResult = await categoryRef.where("name", isEqualTo:categoryName).get();
  if (queryResult.docs.isNotEmpty) {
    return Category.fromJson(queryResult.docs.first.data());
  }
  return null;
}
```

and / or filter condition

- Filter condition connected with **and**

```
citiesRef.where("state", isEqualTo: "CA")
        .where("population", isLessThan: 1000000);
```

- Filter condition connected with **or**

```
var query = db.collection("cities").where(
    Filter.or(
      Filter("capital", isEqualTo: true),
      Filter("population", isGreaterThan: 100000),
    ),
  );
```

Add a document to a Collection

- Get a collection reference

```
var collectionRef = db.collection("colName")
```

- Call **.add** method and pass the object to add the collection
  - Firebase adds the object to the collection and returns the auto-assigned **docId**

⌃ cmp312-fall2020	📖 categories ⧩ ⋮	📄 9bbraJMpuCt7eFWpbvA6
categories ❯	9bbraJMpuCt7eFWpbvA6 ❯	name: "Fruits"

```
final category = Category("Fruits");
final categoryRef = db.collection("categories");
final queryResult = await categoryRef.add(category.toJson());
final categoryId = queryResult.id;
```

Add a document and set DocId

- First specify the desired **docId** to be assigned to the new doc
  `collectionRef`**.doc**(**docId**)

- Call **.set** method and pass the object to add to the collection
  - Firebase adds the object to the collection and the id of the new doc is **docId** passed to **.doc** method

```
Future<void> addUser(User user) async {
  var userCollectionRef = FirebaseFirestore.instance.collection("users");

  await userCollectionRef.doc(user.email).set(user.toJson());
}
```


cmp312-fall2020	users	erradi@live.com
users >	erradi@live.com >	email: "erradi@live.com"
categories		firstName: "Abdelkarim"
restaurants		lastName: "Erradi"
shoppingItems		role: "Admin"
		▼ address
		city: "Doha"
		houseNumber: "345"
		street: "Test St"

Update a document

- Use **.update** and pass the fields to update and their new values as a Map

  - Or use **.set** to replace the whole document

```
Future<void> updateQuantity(String itemId,
      int quantity) async {
  await shoppingItemRef.doc(itemId).update(
      {'quantity': quantity});
}


Future<void> updateItem(ShoppingItem item) async {
  await shoppingItemRef.doc(item.id).set(item.toJson());
}
```

Delete a document

- Use **.delete** method to delete a document

```
Future<void> deleteItem(ShoppingItem item) async {
  await shoppingItemRef.doc(item.id).delete();
}
```

Subscribing to collection/document Realtime Updates

- Use **.snapshots()** to observe the changes of a collection/document and get near **real-time updates**


```
Stream<List<ShoppingItem>> observeShoppingListItems() {
  final uid = FirebaseAuth.instance.currentUser?.uid;
  if (uid == null) return Stream.value([]);
  return FirebaseFirestore.instance
      .collection('shoppingItems')
      .where('uid', isEqualTo: uid)
      .snapshots()
      .map((snapshot) => snapshot.docs
          .map((doc) =>
              ShoppingItem.fromJson(doc.data() as Map<String, dynamic>))
      )
    .toList());
}
```

Securing Data

- Cloud Firestore **Security Rules** consist of:
  - o **match statements**, which identify documents in the database, and
  - o **allow expressions**, which control access to those documents

```
// Allow read/write access on all documents to any user signed in to the app
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if request.auth.uid != null;
    }
  }
}
```

Firebase Storage

Firebase Storage

- Firebase Cloud Storage

  - Store and serve files

  - Robust

  - Secure

  - Access controlled with security rules

- To access to the Firebase Storage service:

```
final storageRef = FirebaseStorage.instance.ref();
```

.ref() creates a reference to the root path. You can append **.child()** to this reference to navigate to specific paths.

Firebase Storage File Operations

▼ ◇ Upload Operations

ⓜ 🔓 putBytes(byte[]): UploadTask

ⓜ 🔓 putFile(Uri): UploadTask

▼ ◇ Download Operations

ⓜ 🔓 getBytes(long): Task<byte[]>

ⓜ 🔓 getFile(Uri): FileDownloadTask

ⓜ 🔓 getFile(File): FileDownloadTask

▼ ◇ Delete

ⓜ 🔓 delete(): Task<Void>

▼ ◇ List

ⓜ 🔓 list(int): Task<ListResult>

ⓜ 🔓 list(int, String): Task<ListResult>

ⓜ 🔓 listAll(): Task<ListResult>

List

- Get URLs of files in particular subfolder

```dart
Future<List<String>> getImageUrls() async {
  List<String> imageUrls = [];
  try {
    // Reference to the 'images/' directory in Firebase Storage
    final storageRef = FirebaseStorage.instance.ref().child("images/");

    // List all items in the 'images/' directory
    final result = await storageRef.listAll();

    // Loop through the items and get the download URL for each image
    for (var image in result.items) {
      String url = await image.getDownloadURL();
      imageUrls.add(url);
    }
  } catch (e) {
    print('Error fetching image URLs: $e');
  }
  return imageUrls;
}
```

Add file

```
final storageRef = FirebaseStorage.instance.ref();
// Upload file
await storageRef.child("images/$filename")
                .putFile(filePath);
final imageUrl = await storageRef.getDownloadURL();
```

Delete file

```
await storageRef.child("images/$filename").delete();
```

```
Future<void> downloadFile(String filename, String destinationPath) async {
  try {
    // Get reference to the file in Firebase Storage
    final ref = FirebaseStorage.instance.ref('files/$filename');

    // Download the file to the destination path
    await ref.writeToFile(File(destinationPath));

    print('File downloaded successfully to $destinationPath');
  } catch (e) {
    print('Error downloading file: $e');
  }
}
```

- Or you can simply get the download URL

```
final ref = FirebaseStorage.instance.ref('files/$filename');
final imageUrl = await ref.getDownloadURL();
```

Firebase Authentication

Firebase Authentication

- **Authentication = Identity verification**:
  - Verify the identity of the user given the credentials received
  - Making sure the user is who he claims to be
- Every user gets a unique ID
- Restrict who can read and write what data

Firebase Authentication

Multiple Identity Providers can be used for Authentication

Sign in

- Sign in using Firebase authentication

```
final authResult = await FirebaseAuth.instance.signInWithEmailAndPassword(
  email: email,
  password: password,
);
print(">> Debug: signIn.authResult : ${authResult.user?.uid}");
```

Sign up

- Sign up and the user details to Firebase authentication

```
Future<User?> signUp(User user) async {
  try {
    // Create a new user with email and password
    User authResult = await FirebaseAuth.instance.createUserWithEmailAndPassword(
      email: user.email,
      password: user.password,
    );

    // If the user was created successfully, update their profile
    if (authResult.user != null) {
      await authResult.user!.updateDisplayName('${user.firstName} ${user.lastName}');
      await authResult.user!.updatePhotoURL('http://test.com/spongebob.png');
      // Refresh the user to apply changes
      await authResult.user!.reload();
    }

    return authResult;
  } catch (e) {
    print('Error during sign up: $e');
    return null;
  }
}
```

Sign out

- Sign out from Firebase auth

  `await FirebaseAuth.instance.`**`signOut`**`();`

- Anywhere in the app you can access the details of current user

```
void getCurrentUser() {
  User? user = FirebaseAuth.instance.currentUser;
  if (user != null) {
    print('User is signed in! Email: ${user.email}');
  } else {
    print('No user is signed in.');
  }
}
```

Access Image Gallery and Camera

Access Image Gallery and Camera

- Using **image_picker** package for picking images from the image gallery or taking new pictures with the camera

```
Future<File?> pickImage(ImageSource source) async {
  final imagePicker = ImagePicker();
  final pickedImage = await imagePicker.pickImage(
    source: source, // camera or gallery
    maxWidth: double.infinity,
  );

  if (pickedImage == null) return null;
  return File(pickedImage.path);
}
```

image_picker methods

```dart
final ImagePicker picker = ImagePicker();
// Pick an image
final XFile? image = await picker.pickImage(source: ImageSource.gallery);
// Capture a photo
final XFile? photo = await picker.pickImage(source: ImageSource.camera);
// Pick a video
final XFile? galleryVideo =
    await picker.pickVideo(source: ImageSource.gallery);
// Capture a video
final XFile? cameraVideo = await picker.pickVideo(source: ImageSource.camera);
// Pick multiple images
final List<XFile> images = await picker.pickMultiImage();
// Pick singe image or video
final XFile? media = await picker.pickMedia();
// Pick multiple images and videos
final List<XFile> medias = await picker.pickMultipleMedia();
```

Summary

- **Cloud Firestore** database store/query app's data

  - Data model consists of collections to store documents that contain data as a key-value pair similar to JSON

- Firebase **Cloud Storage** is used to store and retrieve files

- **Firebase Authentication** provides built-in backend services to ease user authentication

  - email/password authentication allows users to register and log in to the app

  - Secure user's authentication using various identity provides (e.g., email/password, Google Auth)

Resources

- Cloud Firestore

  o https://firebase.google.com/docs/firestore/

- Get to know Cloud Firestore

  o https://www.youtube.com/playlist?list=PLl-K7zZEsYLluG5MCVEzXAQ7ACZBCuZgZ

- Firestore codelab

  o https://firebase.google.com/codelabs/firebase-get-to-know-flutter

- Firebase Auth codelab

  o https://firebase.google.com/codelabs/firebase-auth-in-flutter-apps