LabW04 – Cloud Service

Objectives:

1. Learn how to develop mobile app using Google’s Firebase platform
2. Understand how to use Cloud Firestore to store and read data

Tasks:

1. Connect app to Firebase and configure Cloud Firestore
2. Write data to Cloud Firestore
3. Display data from Cloud Firestore
4. Sort and filter data in Cloud Firestore

We have so far learnt how a mobile app persists data locally on the phone (refer Lab Week 03). It does not allow data access from other clients, such as another app or website.

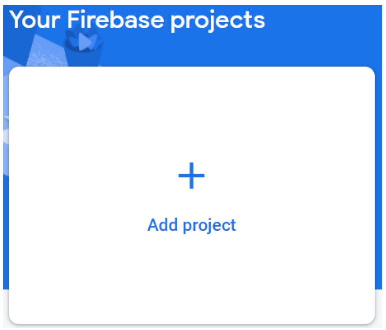
This tutorial is adopted from Cloud Firestore Android Codelab and shows you how to connect a mobile app to Cloud Firestore – Google's Firebase NoSQL Database in the cloud for mobile and web apps – and then query data from it. You will learn how to build a restaurant recommendation app powered by Cloud Firestore called "Friendly Eats". In addition, you will also learn how to:

- Connect your app to Google's Firebase platform and configure Cloud Firestore
- Read and write data to Cloud Firestore from an Android app
- Listen to changes in Firestore data in real time
- Use basic Firebase Authentication

Note: this tutorial uses the free Spark plan, which is sufficient for development purposes for most apps.

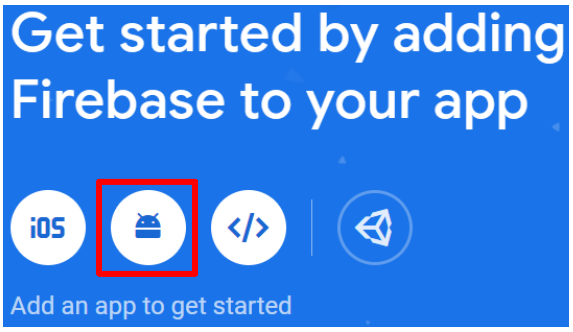
Task 1: Connect app to Firebase and configure Cloud Firestore

1. Sign into the Firebase console with your Google account.
2. On the Firebase console, click Add project.



3. In the "Create a project (Step 1 of 3)", enter a name for your Firebase project, for example, "Friendly Eats". Click Continue.
4. In step 2 of 3 "Google Analytics for your Firebase project", choose "Enable Google Analytics for this project". Click Continue.
5. In step 3 of 3 "Configure Google Analytics", choose "Create a new account", enter "COMP5216" as the new Google Analytics account name. Click Save.
6. Leave the option for "Analytics location" as "United States" and leave the default option for "Use the default settings for sharing Google Analytics data."
7. Accept the other two terms of Google Analytics.
8. Click "Create project".
9. After a minute or so, your Firebase project will be ready. Click Continue.
10. Download from Canvas and unzip the sample base code provided for Fire Eats app: "friendlyeats-android.zip". Once unzipped, a folder "friendlyeats-android" should be created on your machine. You can also download the sample code directly from GitHub.

11. Import the project into Android Studio. You will probably see some compilation errors or maybe a warning about a missing google-services.json file.
12. On the Firebase console, select Project Overview in the left navigation. Get started by adding Firebase to your app by clicking the Android button to select the platform. When prompted for an Android package name use “com.google.firebase.example.fireeats”.


Get started by adding Firebase to your app
iOS
</>
Add an app to get started

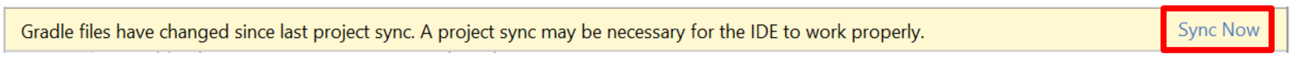
13. Click “Register app” and follow the instructions to download the “google-services.json” config file, and move it into the app/ folder of your Android app code. Click Next.
14. Follow the instructions to add the Firebase SDK dependencies to your gradle files by modifying your build.gradle files to use the Google services plugins:
    - Project-level build.gradle (<project>/build.gradle)

```
buildscript {
  dependencies {
    // Add this line
    classpath 'com.google.gms:google-services:4.3.3'
  }
}
```

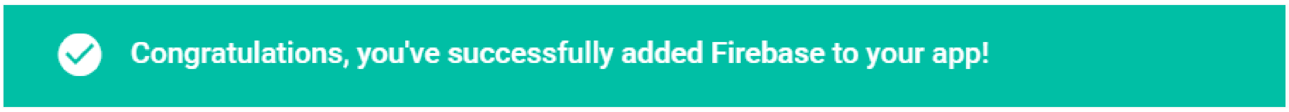
    - App-level build.gradle (<project>/<app-module>/build.gradle):

```
dependencies {
  // Add this line
  implementation 'com.google.firebase:firebase-analytics:17.5.0'
}
...
// Add to the bottom of the file
apply plugin: 'com.google.gms.google-services'
```

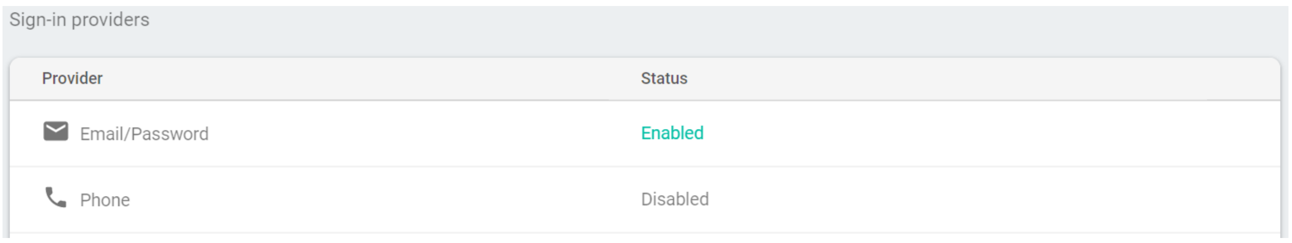
15. Finally, press "Sync now" in the bar that appears in Android Studio. Click Next.


Gradle files have changed since last project sync. A project sync may be necessary for the IDE to work properly.

16. Run the app to verify installation. You should see the below message on the Firebase console if Firebase is successfully added to your app.


Congratulations, you've successfully added Firebase to your app!

17. Next, we should set some basic rules to restrict data access to users who are signed in, so we can prevent unauthenticated users from reading or writing. From the console’s navigation pane, select “Authentication” -> “Sign-in method” -> enable “Email/Password” authentication:



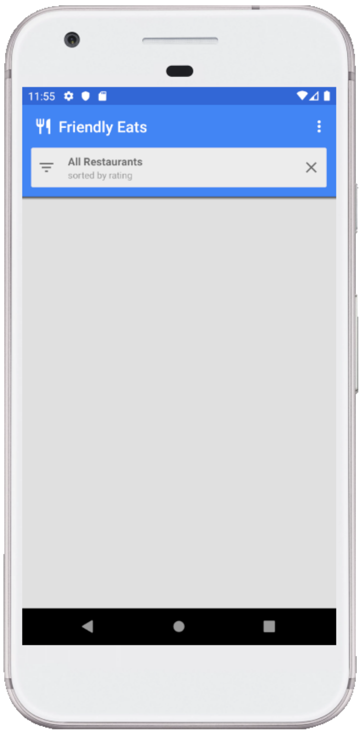
Sign-in providers

Provider	Status
Email/Password	Enabled
Phone	Disabled

18. Enable Cloud Firestore for your project. In the Firebase console's navigation pane, select "Cloud Firestore" -> "Create database" to provision Cloud Firestore database.
19. Select a starting mode for your Cloud Firestore Security Rules. Choose "Start in test mode" to get set up quickly by allowing all reads and writes to your database. Click Next.
20. Set a location for your Cloud Firestore as "nam5 (us-central)". Once the location is set, you cannot change it later. To learn more, you can read Select locations for your project.
21. Click Enable to provision Cloud Firestore.
22. Access to data in Cloud Firestore is controlled by Security Rules. First we need to set some basic rules on our data to restrict access only to users who are signed in. In the console, navigate to "Cloud Firestore" -> "Rules" tab, add these rules and click Publish:

```
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if request.auth != null;
    }
  }
}
```

23. If you have set up your app correctly, the project should now compile. In Android Studio click Build -> Rebuild Project. Run the app on your Android device. At first you will be presented with a "Sign in" screen. You can use an email and password to sign up into the app. Once you have completed the sign in process you should see the home screen:



Task 2: Write data to Cloud Firestore

In this task, we will write some data to Cloud Firestore so that we can populate the home screen. You can enter data manually in the Firebase console, but we'll do it in the app itself to demonstrate how to write data to Firestore using the Android SDK.

The main model object in our app is a restaurant (refer model/Restaurant.java). Firestore data is split into documents, collections, and subcollections. We will store each restaurant as a document in a top-level collection called "restaurants". To learn more about the Firestore data model, read about documents and collections.

1. First, let's get an instance of FirebaseFirestore to work with. Edit the initFirestore() method in MainActivity:

```
private void initFirestore() {
    mFirestore = FirebaseFirestore.getInstance();
}
```

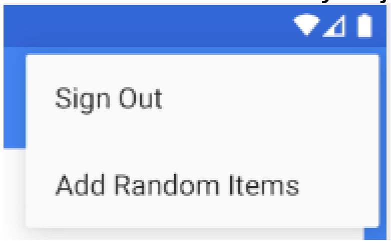
2. Add functionality in the app to create 10 random restaurants when we click the "Add Random Items" button in the overflow menu. Fill in the onAddItemsClicked():

```
private void onAddItemsClicked() {
    // Get a reference to the restaurants collection
    CollectionReference restaurants = mFirestore.collection("restaurants");

    for (int i = 0; i < 10; i++) {
        // Get a random Restaurant POJO
        Restaurant restaurant = RestaurantUtil.getRandom(this);

        // Add a new document to the restaurants collection
        restaurants.add(restaurant);
    }
}
```

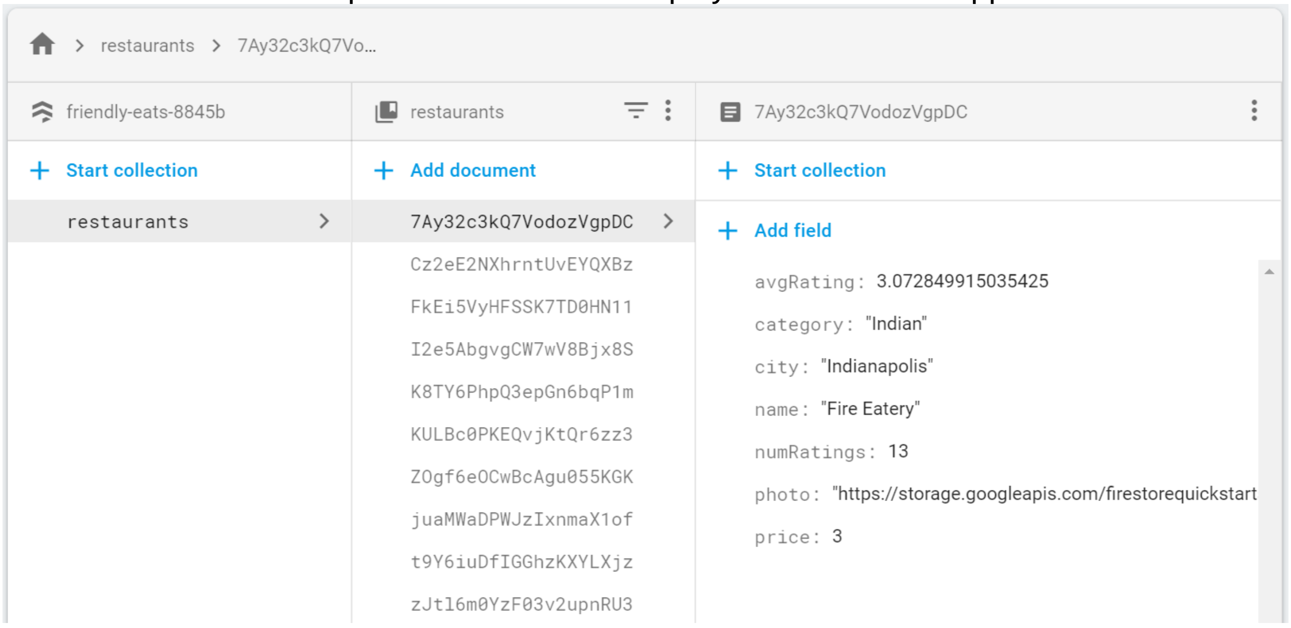
There are a few important things to note about the code above:

- We started by getting a reference to the "restaurants" collection. Collections are created implicitly when documents are added, so there was no need to create the collection before writing data.
- Documents can be created using Plain Old Java Objects (POJOs), which we use to create each Restaurant doc.
- The add() method adds a document to a collection with an auto-generated ID, so we did not need to specify a unique ID for each Restaurant.

3. Now run the app again and click the "Add Random Items" button in the overflow menu to invoke the code you just wrote:



If you navigate to the Firebase console -> "Cloud Firestore" -> "Data" tab, you should see the newly added data as per below. Congratulations, you just wrote data to Cloud Firestore! In the next step we'll learn how to display this data in the app.



```
7Ay32c3kQ7VodozVgpDC
Cz2eE2NXhrntUvEYQXBz
FkEi5VyHFSSK7TD0HN11
I2e5AbgvgCW7wV8Bjx8S
K8TY6PhpQ3epGn6bqP1m
KULBc0PKEQvjKtQr6zz3
Z0gf6eOCwBcAgu055KGK
juaMWaDPWJzIxnmaX1of
t9Y6iuDfIGGhzKXYLXjz
zJtl6m0YzF03v2upnRU3
```

```
avgRating: 3.072849915035425
category: "Indian"
city: "Indianapolis"
name: "Fire Eatery"
numRatings: 13
photo: "https://storage.googleapis.com/firestorequickstart
price: 3
```

Task 3: Display data from Cloud Firestore

In this task we will learn how to retrieve data from Cloud Firestore and display it in our app.

1. The first step to reading data from Cloud Firestore is to create a Query. Modify the initFirestore() method:

```java
private void initFirestore() {
    mFirestore = FirebaseFirestore.getInstance();

    // Get the 50 highest rated restaurants
    mQuery = mFirestore.collection("restaurants")
            .orderBy("avgRating", Query.Direction.DESCENDING)
            .limit(LIMIT);
}
```

2. Now we want to listen to the query, so that we get all matching documents and are notified of future updates in real time. Because our eventual goal is to bind this data to a RecyclerView, we need to create a RecyclerView.Adapter class to listen to the data. Whilst this would demonstrate the real-time capabilities of Cloud Firestore, but it is also simple to fetch data without a listener. You can call get() on any query or reference to fetch a data snapshot.

   Open the FirestoreAdapter class, which has already been partially implemented. First, let's make the adapter implement EventListener and define the onEvent function so that it can receive updates to a Firestore query:

```
public abstract class FirestoreAdapter<VH extends RecyclerView.ViewHolder>
        extends RecyclerView.Adapter<VH>
        implements EventListener<QuerySnapshot> {

        // ...
        @Override
        public void onEvent(QuerySnapshot documentSnapshots,
                            FirebaseFirestoreException e) {
            // Handle errors
            if (e != null) {
                Log.w(TAG, "onEvent:error", e);
                return;
            }

            // Dispatch the event
            for (DocumentChange change : documentSnapshots.getDocumentChanges()) {
                // Snapshot of the changed document
                DocumentSnapshot snapshot = change.getDocument();

                switch (change.getType()) {
                    case ADDED:
                        // TODO: handle document added
                        break;
                    case MODIFIED:
                        // TODO: handle document modified
                        break;
                    case REMOVED:
                        // TODO: handle document removed
                        break;
                }
            }
            onDataChanged();
        }
}
```

3. On initial load the listener will receive one ADDED event for each new document. As the result set of the query changes over time the listener will receive more events containing the changes. Now let's finish implementing the listener. First add three new methods: onDocumentAdded, onDocumentModified, and onDocumentRemoved:

```
protected void onDocumentAdded(DocumentChange change) {
    mSnapshots.add(change.getNewIndex(), change.getDocument());
    notifyItemInserted(change.getNewIndex());
}

protected void onDocumentModified(DocumentChange change) {
    if (change.getOldIndex() == change.getNewIndex()) {
        // Item changed but remained in same position
        mSnapshots.set(change.getOldIndex(), change.getDocument());
        notifyItemChanged(change.getOldIndex());
    } else {
        // Item changed and changed position
        mSnapshots.remove(change.getOldIndex());
        mSnapshots.add(change.getNewIndex(), change.getDocument());
        notifyItemMoved(change.getOldIndex(), change.getNewIndex());
    }
}

protected void onDocumentRemoved(DocumentChange change) {
    mSnapshots.remove(change.getOldIndex());
    notifyItemRemoved(change.getOldIndex());
}
```

4. Call the three new methods from onEvent:

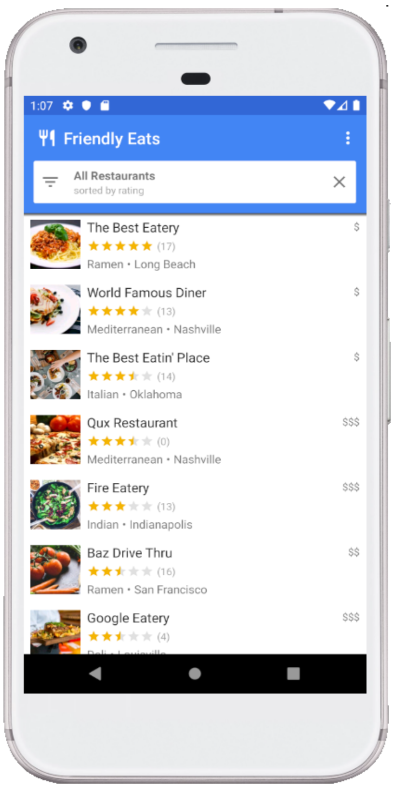
```
@Override
public void onEvent(QuerySnapshot documentSnapshots,
                    FirebaseFirestoreException e) {

    // Handle errors
    if (e != null) {
        Log.w(TAG, "onEvent:error", e);
        return;
    }

    // Dispatch the event
    for (DocumentChange change : documentSnapshots.getDocumentChanges()) {
        // Snapshot of the changed document
        DocumentSnapshot snapshot = change.getDocument();

        switch (change.getType()) {
            case ADDED:
                onDocumentAdded(change);
                break;
            case MODIFIED:
                onDocumentModified(change);
                break;
            case REMOVED:
                onDocumentRemoved(change);
                break;
        }
    }

    onDataChanged();
}
```

5. Finally implement the startListening() method to attach the listener:

```
public void startListening() {
    if (mQuery != null && mRegistration == null) {
        mRegistration = mQuery.addSnapshotListener(this);
    }
}
```

6. Now the app is fully configured to read data from Cloud Firestore. Run the app again and you should see the restaurants you added in the previous step.

   Go back to the Firebase console and edit one of the restaurant names. You should see it change in the app almost instantly!



The Best Eatery ★★★★★ (17) $
Ramen • Long Beach

World Famous Diner ★★★★ (13) $
Mediterranean • Nashville

The Best Eatin' Place ★★★½ (14) $
Italian • Oklahoma

Qux Restaurant ★★★½ (0) $$$
Mediterranean • Nashville

Fire Eatery ★★★ (13) $$$
Indian • Indianapolis

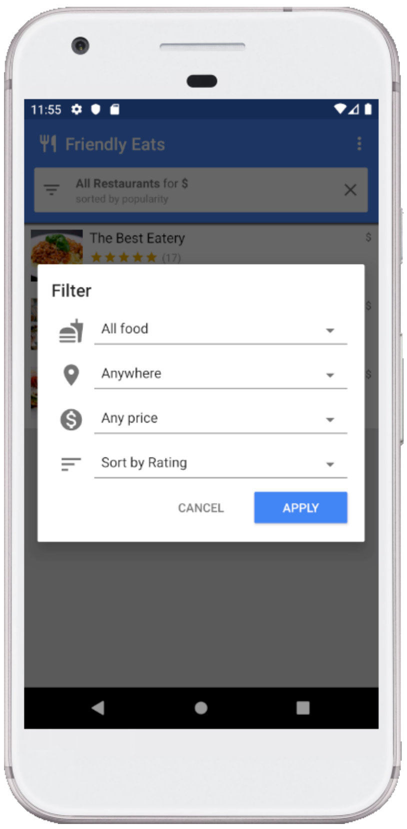
Baz Drive Thru ★★½ (16) $$
Ramen • San Francisco

Google Eatery ★★½ (4) $$$

Task 4: Sort and filter data in Cloud Firestore

The app currently displays the top-rated restaurants across the entire collection, but in a real restaurant app the user would want to sort and filter the data. For example, the app should be able to show "Top seafood restaurants in New York" or "Least expensive pizza".

Clicking white bar at the top of the app brings up a filters dialog. In this section we'll use Firestore queries to make this dialog work:

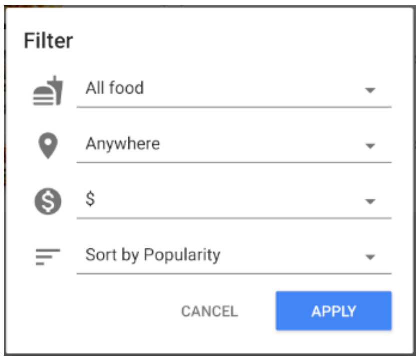


1. Edit the onFilter() method of MainActivity.java. This method accepts a Filters object, a helper object we created to capture the output of the filters dialog. We will change this method to construct a query from the filters.

   In the code snippet below, we build a Query object by attaching where and orderBy clauses to match the given filters.

```
@Override
public void onFilter(Filters filters) {
    // Construct query basic query
    Query query = mFirestore.collection("restaurants");

    // Category (equality filter)
    if (filters.hasCategory()) {
        query = query.whereEqualTo("category", filters.getCategory());
    }

    // City (equality filter)
    if (filters.hasCity()) {
        query = query.whereEqualTo("city", filters.getCity());
    }

    // Price (equality filter)
    if (filters.hasPrice()) {
        query = query.whereEqualTo("price", filters.getPrice());
    }

    // Sort by (orderBy with direction)
    if (filters.hasSortBy()) {
        query = query.orderBy(filters.getSortBy(), filters.getSortDirection());
    }

    // Limit items
    query = query.limit(LIMIT);

    // Update the query
    mQuery = query;
    mAdapter.setQuery(query);

    // Set header
    mCurrentSearchView.setText(Html.fromHtml(filters.getSearchDescription(this)));
    mCurrentSortByView.setText(filters.getOrderDescription(this));

    // Save filters
    mViewModel.setFilters(filters);
}
```

2. Run the app and select the following filter to show the most popular low-price restaurants:



Filter

All food

Anywhere

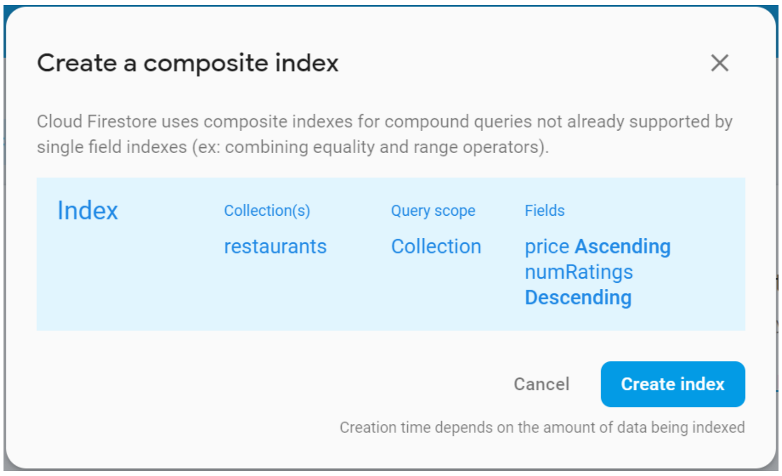
$

Sort by Popularity

3. If you view the app logs using adb logcat or the Logcat panel in Android Studio, you will notice the following warnings:

```
W/Firestore: (20.2.0) [Firestore]: Listen for Query(restaurants where price == 1 order
by -numRatings, -__name__) failed: Status{code=FAILED_PRECONDITION, description=The
query requires an index. You can create it here:
https://console.firebase.google.com/project/friendly-eats-
8845b/database/firestore/indexes?create_composite=CmJwcm9qZWN0cy9mcmllbmRseS1lYXRzLTg4N
DViL2RhdGFiYXNlcy8oZGVmYXVsdCkvY29sbGVjdGlvbkdyb3Vwcy9yZXN0YXVyYW50cy9pbmRleGVzL0NJQ0Fn
T2pYaDRFSxABGgkKBXByaWNlEAEaDgoKbnVtUmF0aW5ncxACGgwKCF9fbmFtZV9fEAI, cause=null}

W/Firestore Adapter: onEvent:error
    com.google.firebase.firestore.FirebaseFirestoreException: FAILED_PRECONDITION: The
query requires an index. You can create it here:
https://console.firebase.google.com/project/friendly-eats-
8845b/database/firestore/indexes?create_composite=CmJwcm9qZWN0cy9mcmllbmRseS1lYXRzLTg4N
DViL2RhdGFiYXNlcy8oZGVmYXVsdCkvY29sbGVjdGlvbkdyb3Vwcy9yZXN0YXVyYW50cy9pbmRleGVzL0NJQ0Fn
T2pYaDRFSxABGgkKBXByaWNlEAEaDgoKbnVtUmF0aW5ncxACGgwKCF9fbmFtZV9fEAI
        at
com.google.firebase.firestore.util.Util.exceptionFromStatus(com.google.firebase:firebas
e-firestore@@20.2.0:121)
```

The query could not be completed on the backend because it requires an index. Most Firestore queries involving multiple fields (i.e. price and rating) require a custom composite index. Clicking the link in the error message will open the Firebase console and automatically prompt you to create the correct index:

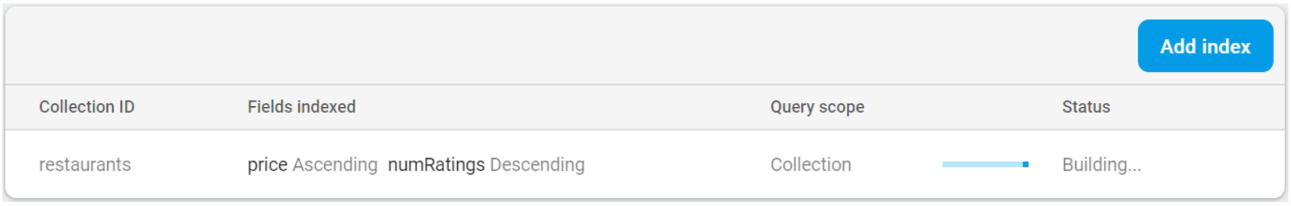

Create a composite index

Cloud Firestore uses composite indexes for compound queries not already supported by single field indexes (ex: combining equality and range operators).

Index	Collection(s)	Query scope	Fields
	restaurants	Collection	price Ascending numRatings Descending

Creation time depends on the amount of data being indexed

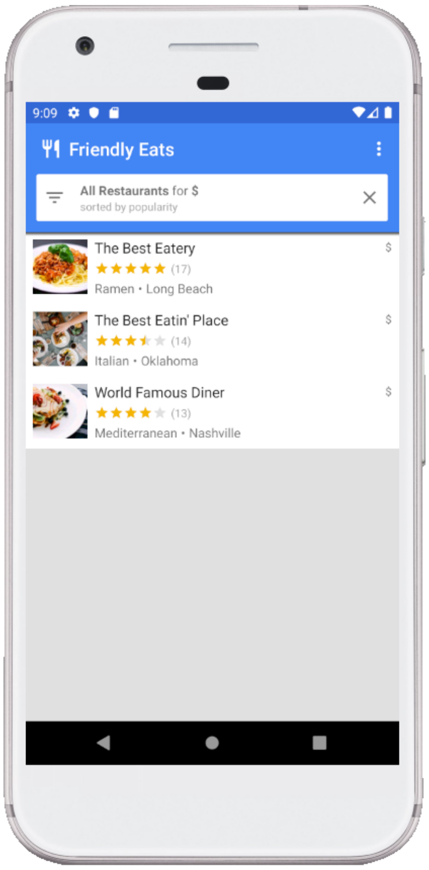
Clicking “Create Index” will begin creating the required index. When the index is complete the status should change from “Building…” to “Enabled”:



Collection ID	Fields indexed	Query scope	Status
restaurants	price Ascending numRatings Descending	Collection	Building...

Note: the Firestore Android SDK caches documents offline and will return those results in the event of a server error. If you execute a query without a matching index you may still see results in your UI but it's important to check the logs to make sure you have all required indexes.

4. Now that the proper index has been created, run the app again and execute the same query. You should now see a filtered list of restaurants containing only low-price options sorted by popularity in descending order:



9:09

Friendly Eats

All Restaurants for $
sorted by popularity

The Best Eatery $
★★★★★ (17)
Ramen • Long Beach

The Best Eatin' Place $
★★★½ (14)
Italian • Oklahoma

World Famous Diner $
★★★★ (13)
Mediterranean • Nashville

You have just built a fully functioning restaurant recommendation viewing app on Cloud Firestore! You can now sort and filter restaurants in real time.

If you have trouble implementing this, you may refer to the reference code provided in the LabW04_files.zip file.

5. [Optional] Add ratings to the app so users can review their favorite (or least favorite) restaurants.

For more information about Firebase and Cloud Firestore, refer to the following links:

- Cloud Firestore Android Codelab - https://codelabs.developers.google.com/codelabs/firestore-android
- Add Firebase to your Android project - https://firebase.google.com/docs/android/setup
- Cloud Firestore - https://firebase.google.com/docs/firestore/
- Friendly Eats - https://github.com/firebase/friendlyeats-android
- Cloud Firestore Quickstart - https://github.com/firebase/quickstart-android/tree/master/firestore