



Firebase

Firestore

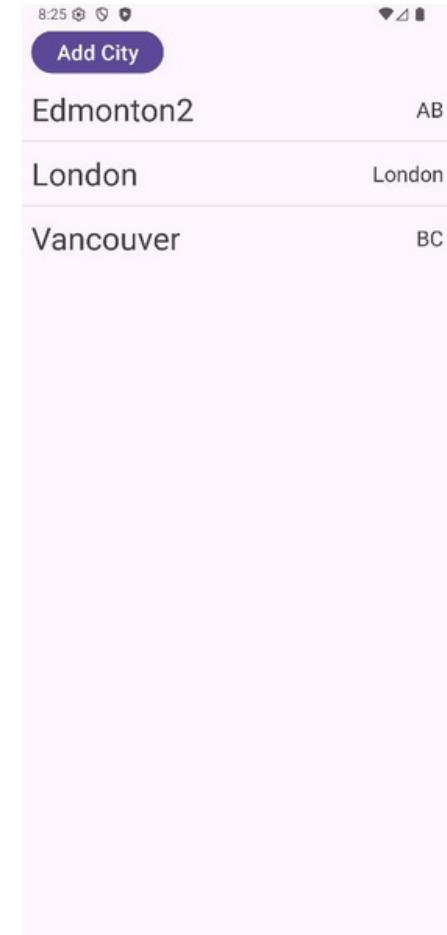
CS360 SPRING 2026

Preparations

1. Fork the starter code repo from github
2. Clone the repo
3. Open Android Studios and initialize the project

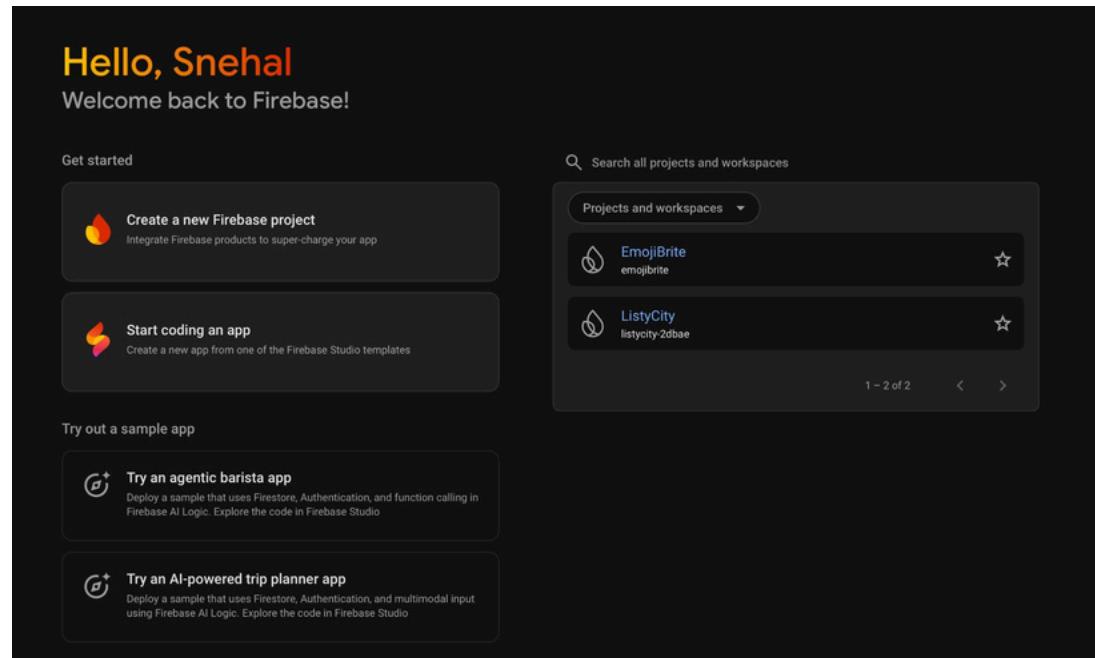
The First Look

- The project continues where we left off
- A list of cities and province abbreviations with the ability to
 - Add city
 - Edit an existing city
- What we are going to do:
 - Send this data to the cloud (FireStore)
 - Synchronize after each action
 - TODO: introduce Delete action

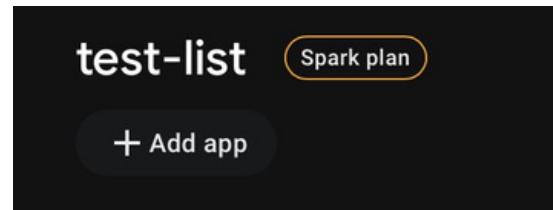


Create a Firestore project

1. Go to <https://console.firebaseio.google.com/> : The firestore website
2. Sign in
3. Create a project

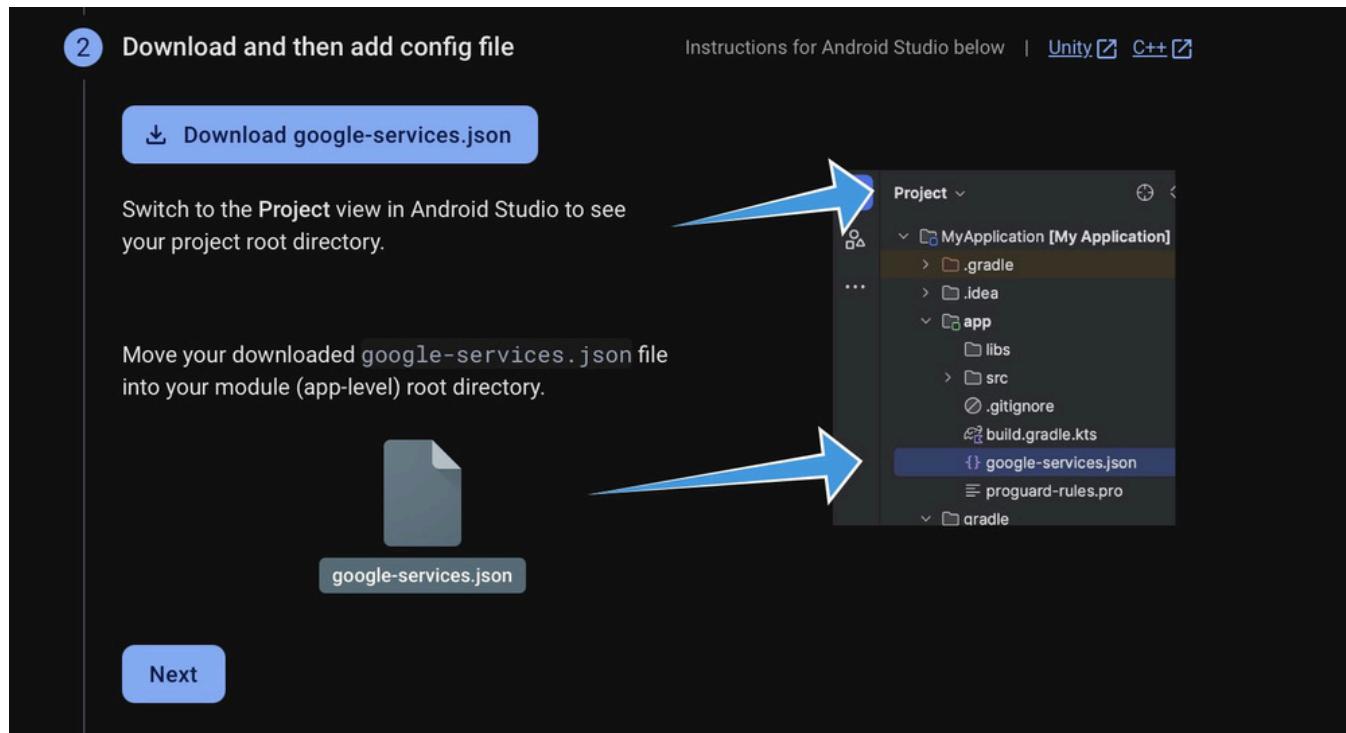


- Input the project name (the name is arbitrary)



- Click on Add App and then click on android

- Android project name: **com.example.lab5_starter**
- You should be able to download a google-services.json file



- Insert the google-services plugin in the project-level Gradle file

1. To make the `google-services.json` config values accessible to Firebase SDKs, you need the Google services Gradle plugin.

Kotlin DSL (`build.gradle.kts`) Groovy (`build.gradle`)

Add the plugin as a dependency to your **project-level** `build.gradle.kts` file:

Root-level (project-level) Gradle file (`<project>/build.gradle.kts`):

```
plugins {  
    // ...  
  
    // Add the dependency for the Google services Gradle plugin  
    id("com.google.gms.google-services") version "4.4.3" apply false  
}
```

- Insert the google-services plugin and firebase dependencies to the app-level Gradle file

2. Then, in your **module (app-level)** `build.gradle.kts` file, add both the `google-services` plugin and any Firebase SDKs that you want to use in your app:

Module (app-level) Gradle file (<project>/<app-module>/build.gradle.kts):

```
plugins {
    id("com.android.application")
    // Add the Google services Gradle plugin
    id("com.google.gms.google-services")
    ...
}

dependencies {
    // Import the Firebase BoM
    implementation(platform("com.google.firebase:firebase-bom:34.3.0"))
    // TODO: Add the dependencies for Firebase products you want to use
    // When using the BoM, don't specify versions in Firebase dependencies
    // https://firebase.google.com/docs/android/setup#available-libraries
}
```

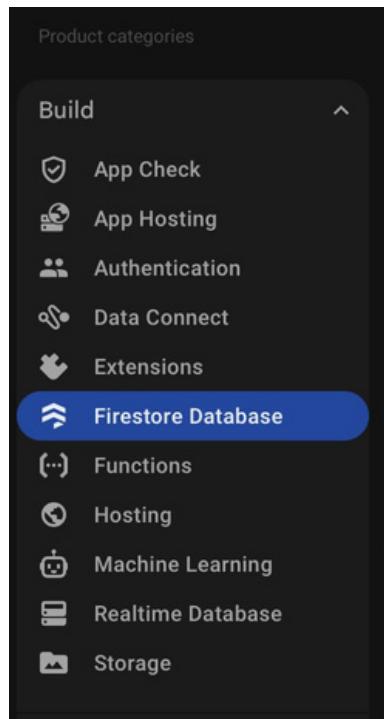
Add Firebase to the Application (summarized)

1. After downloading the google-service.json, move the file to the ListyCity/app/ folder in your Android Studios
2. On Android Studios, open *build.gradle.kts* (*project: ListyCity*) and add
`id("com.google.gms.google-services") version "4.4.3" apply false` **inside**
3. Open *build.gradle.kts* (*Module:app*)
Add `id("com.google.gms.google-services")` **inside** `plugins`
Inside `dependencies`, add these 2 implementations:
`implementation(platform("com.google.firebaseio.firebaseio-bom:34.3.0"))`
`implementation("com.google.firebase:firebase-firebase-storage")`
6. Sync

Create a Firestore Database

1. On the left, under “Product Categories”, Choose “Build -> Firestore Database”
2. Click “Create Database”
3. Select everything as given (Id and location)
4. Choose “Start in test mode”
5. You should see an empty database

Create a Firestore Database



Create a database

1 Select edition
2 Database ID & location
3 Configure

After you define your data structure, you will need to write rules to secure your data. [Learn more](#)

Start in **production mode**
Your data is private by default. Client read/write access will only be granted as specified by your security rules.

Start in **test mode**
Your data is open by default to enable quick setup. However, you must update your security rules within 30 days to enable long-term client read/write access.

```
rules_version = '2';

service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if
        request.time < timestamp.date(2025, 11, 2);
    }
  }
}
```

! The default security rules for test mode allow anyone with your database reference to view, edit and delete all data in your database for the next 30 days

[Create](#)



More in Google Cloud ▾

☁ (default)

+ Start collection



Your database is ready to go. Just add data.

Use Firestore Database in MainActivity

- Remove the hard-coded data in the list (Remove the call to the addCitiesInit method in onCreate method of MainActivity).
- Add Firestore instance in MainActivity.java.

```
1 usage
private FirebaseFirestore db;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Rest of the code

    db = FirebaseFirestore.getInstance();|
```

Adding Collection Reference

- Get a collection reference, in this lab, we store all data in a collection called “*cities*”.

```
2 usages
private FirebaseFirestore db;
1 usage
private CollectionReference citiesRef;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Rest of the code

    db = FirebaseFirestore.getInstance();
    citiesRef = db.collection( collectionPath: "cities" );
```

Adding Snapshot Listener to the Collection

```
db = FirebaseFirestore.getInstance();
citiesRef = db.collection( collectionPath: "cities");

citiesRef.addSnapshotListener(( QuerySnapshot value, FirebaseFirestoreException error) -> {
    if (error != null){
        Log.e( tag: "Firestore", error.toString());
    }
    if (value != null && !value.isEmpty()){
        cityArrayList.clear();
        for (QueryDocumentSnapshot snapshot : value){
            String name = snapshot.getString( field: "name");
            String province = snapshot.getString( field: "province");

            cityArrayList.add(new City(name, province));
        }
        cityArrayAdapter.notifyDataSetChanged();
    }
});
```

Modifying the AddCity method

```
1 usage
@Override
public void addCity(City city){
    cityArrayList.add(city);
    cityArrayAdapter.notifyDataSetChanged();

    DocumentReference docRef = citiesRef.document(city.getName());
    docRef.set(city);
}
```

Optional: Logging

- Optionally, you can add listeners for logging while saving a city. You can find more listeners in the API references ([link](#))

```
citiesRef
    .document(city.getCityName())
    .set(data)
    .addOnSuccessListener(new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            Log.d("Firestore", "DocumentSnapshot successfully written!");
        }
    });
}
```

Run The App

The image displays a composite screenshot of a mobile application and a cloud-based database interface.

Mobile Application (Left):

- Header: Edmonton Alberta
- Search bar: A large search bar at the top.
- Form: A form with fields for "City" and "Province".
- Buttons: An "Add City" button.

Google Cloud Firestore Database (Right):

- Path: cities > Edmonton
- Collection: cities (under default)
- Document: Edmonton
- Fields:
 - + Start collection
 - + Add document
 - + Add field
 - Province: "Alberta"

Lab 5 Participation Exercise

Task: In this exercise it is your task to add the ability to delete Cities and integrate this functionality with the Firestore database, allowing for the persistence of deletions.

- After applying the changes in the lab demo
- Add the ability to delete cities from the ListView and apply these same deletions to your instance of the Firestore database. If you have implemented this functionality correctly, restarting your app should not have an impact on what cities are displayed in your ListView after any addition or deletion actions.