Firestore Integration

Cloud Persistence for Android Applications

CMPUT 301

Preparations

1. Fork the starter code repository
2. Clone your fork to your computer
3. Open the project in Android Studio
4. Wait for Gradle to sync to make sure the project builds and runs on your device

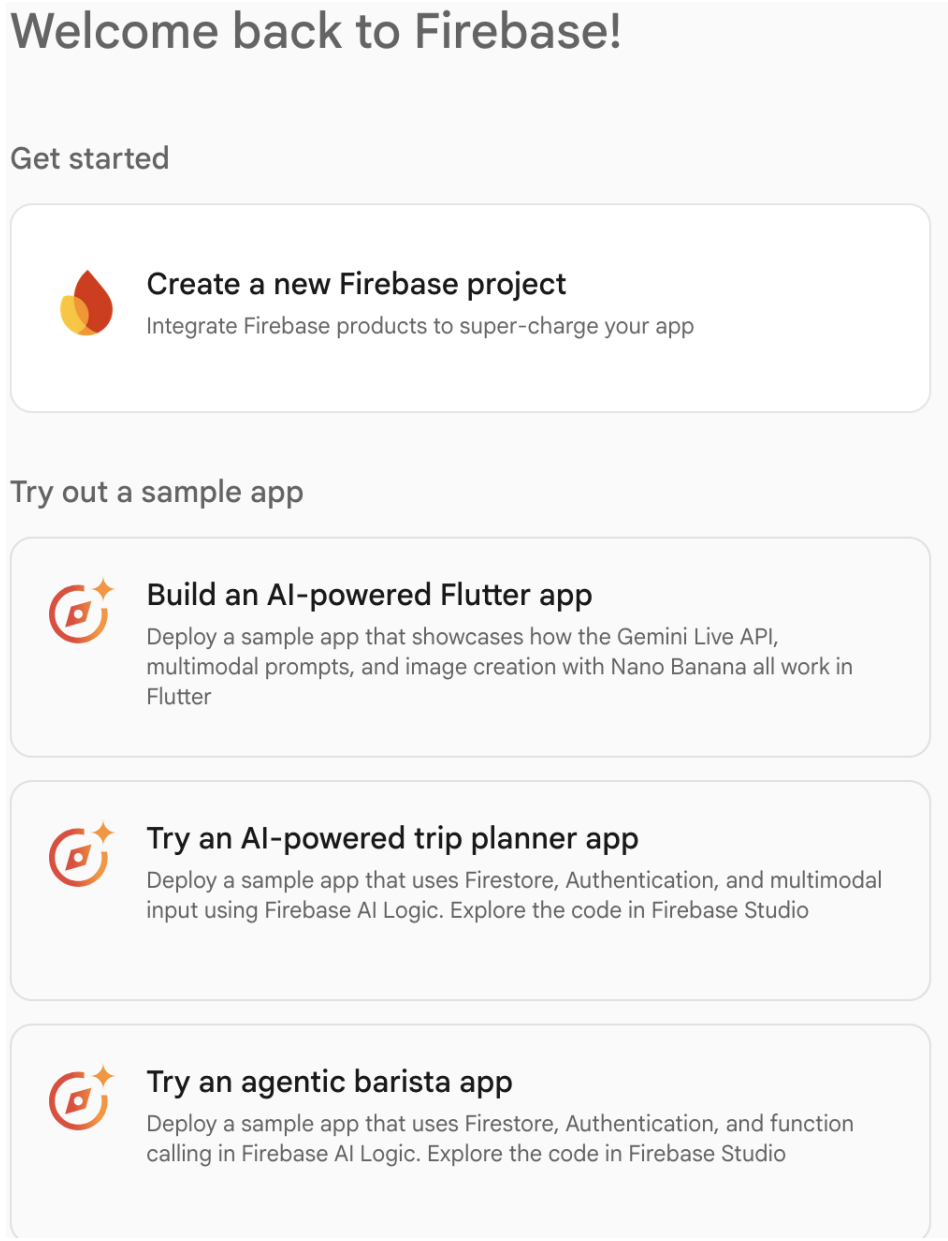
What We Are Building

- This lab continues the ListyCity app
- The app currently stores a list of cities locally
- Users can:
  - add a city
  - edit an existing city

- In this lab, we will connect the city list to Firestore
  - **Goal**: save city data in the cloud so additions, edits, and *deletions* persist after restarting the app

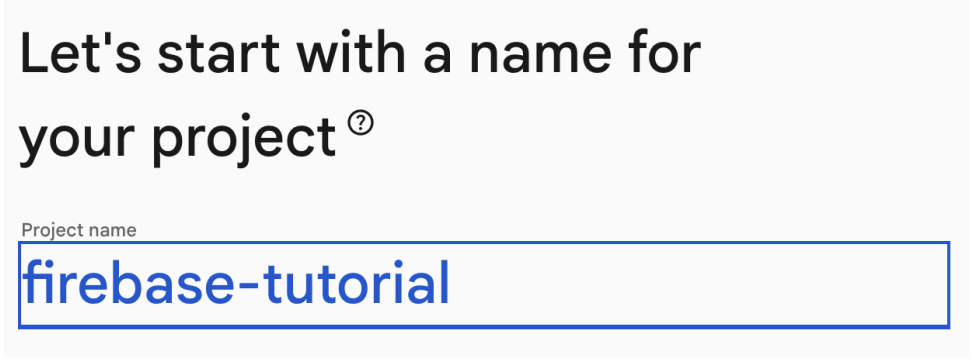
Create a Firebase Project

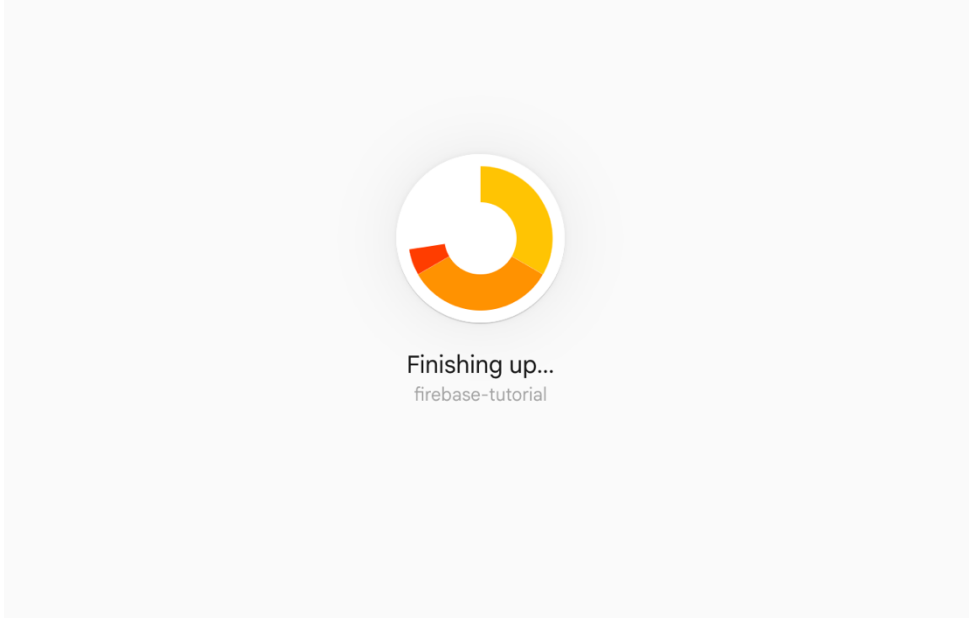
- We create a Firebase project first, then enable Firestore
- Open the **Firebase Console**
  - https://console.firebase.google.com
- Sign in with your Google account
  - Use your personal email for Firebase, **not** your @ualberta.ca email!
- Click **Create a project**
- Give the project a recognizable name



Create a Firebase Project

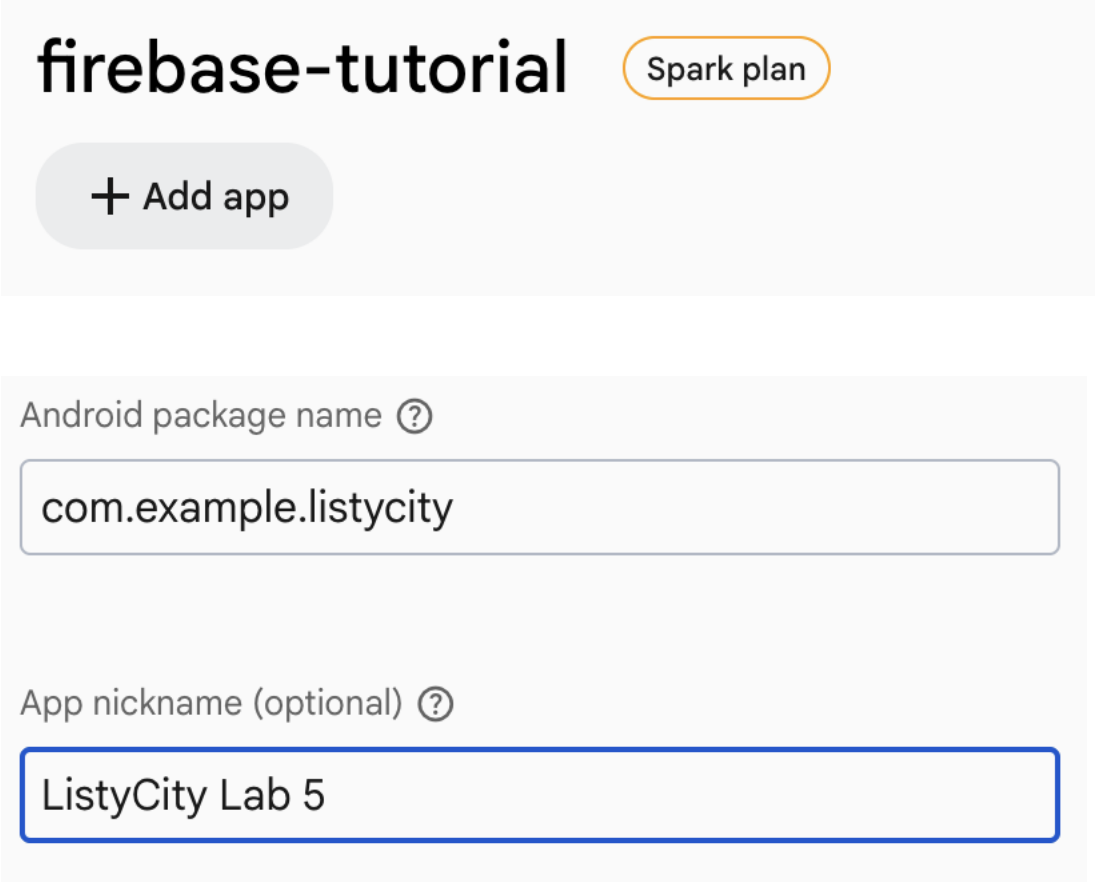
- Continue through the setup prompts
  - Enter any name for your project
- Google Analytics is not needed for this lab
- Click Create Project and wait for Firebase to finish creating the project





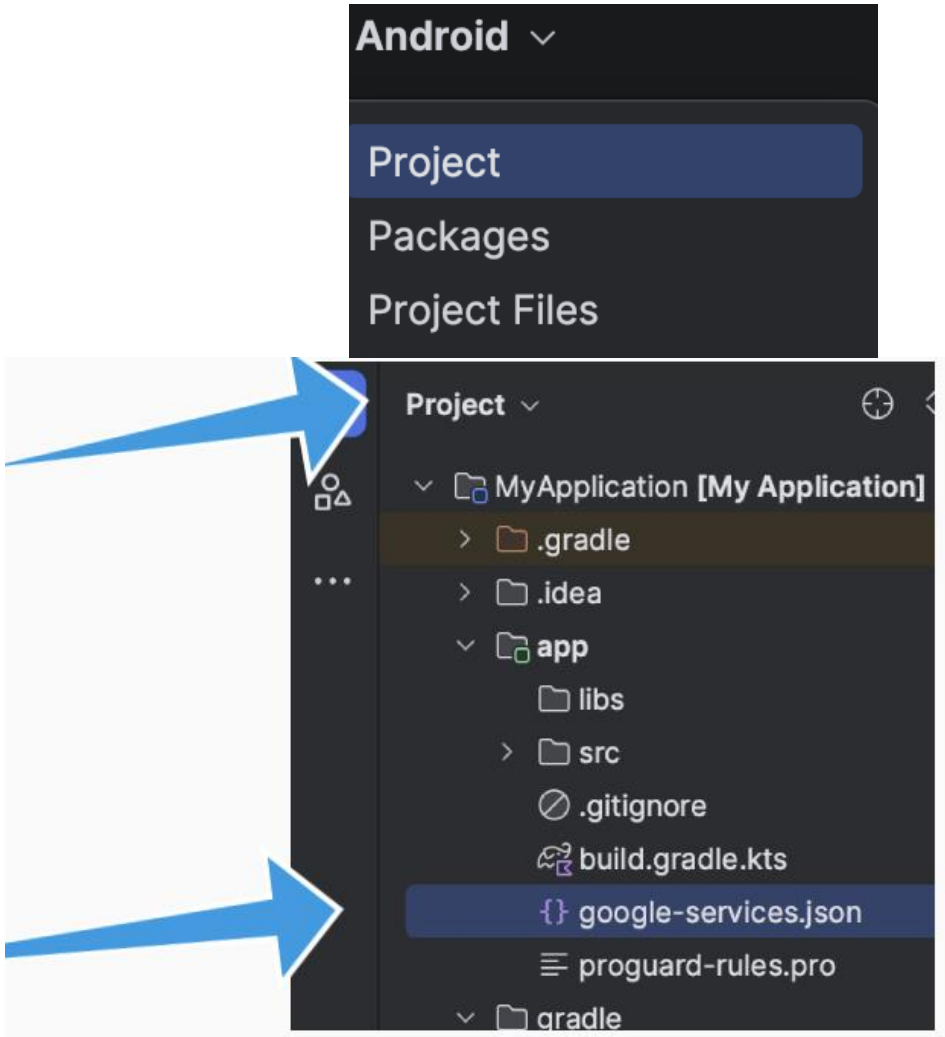
Add Your Android App to Firebase

- In your Firebase project, click **Add app**
- Select the Android icon
- Use the package name from your ListyCity starter code
  - Gradle Scripts -> build.gradle.kts (Module :app) -> Android { namespace = “project name”
- Click **Register app**



Download *google-services.json*

- After registering, Firebase will provide a *google-services.json* file which you must download
- Change the view from Android to Project view
- Follow the direction from Firebase to insert the .json file into the correct folder
  - ListyCity/app/google-services.json
  - Make sure it is inside the ‘app/’ module, **not the project root**



Add Firebase to Gradle

- In Android Studio, return to Android view to update the Gradle files so the app can use Firebase
- In the **Project-level** build.gradle.kts under ‘Plugins’:
  - id("com.google.gms.google-services") version "4.x.x" apply false
- In the **App-level** build.gradle.kts under ‘Plugins’:
  - id("com.google.gms.google-services")
- Note: Firebase dependency versions may change over time
  - If Firebase’s setup page shows newer versions, use the versions recommended by Firebase.

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 "4.4.4" apply false
}
```

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")
  ...
  }
```

Add Firestore Dependencies

- Add Firestore in the **App-level** gradle file under dependencies:
  - dependencies {
  - implementation(platform("com.google.firebase:firebase-bom:3x.x.x"))
  - implementation("com.google.firebase:firebase-firestore") }
- Then click ‘Sync now’

```
dependencies {
  // Import the Firebase BoM
  implementation(platform("com.google.firebase:firebase-bom:34.13.0"))

  // 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
}
```

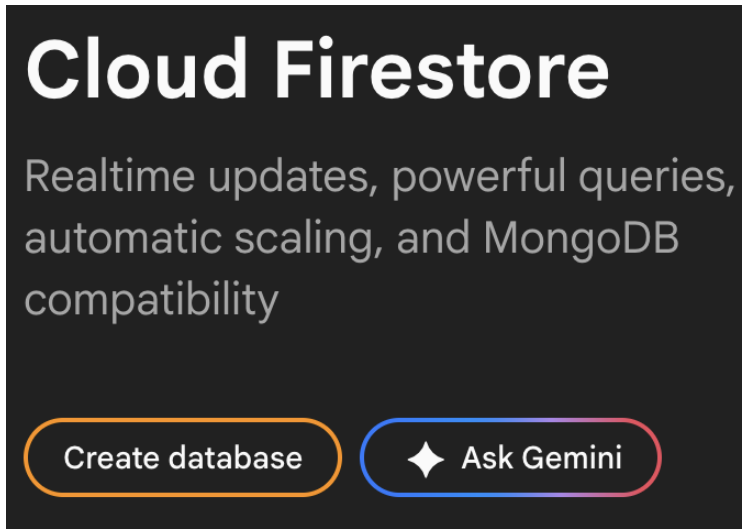
Troubleshooting Firebase Imports

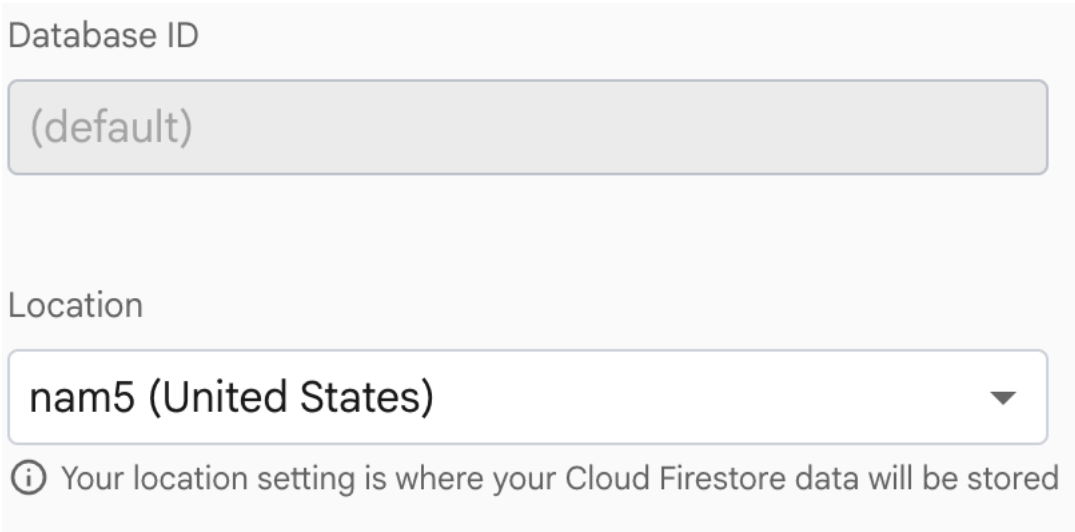
If Firebase imports are red in CityRepository.kt, check these first:

1. Make sure google-services.json is inside the app/ folder, not the project root
2. Make sure the project-level build.gradle.kts includes the google-services plugin
3. Make sure the app-level build.gradle.kts applies the google-services plugin
4. Make sure the app-level dependencies include the Firebase BoM and firebase-firestore
5. Click Sync Now after editing Gradle files
6. If the errors remain, try File > Sync Project with Gradle Files, then Build > Rebuild Project

Create a Firestore Database

- In the Firebase Console home, open your project
- From the left menu, go to **Databases & Storage** -> **Firestore**
- Click ‘Create database’
- Select Standard edition
- Choose the default database ID and location
- Start in **test mode** for this lab
- Click Create





Using Firestore in CityRepository

- Firestore will store the city list in a *cities* collection
- CityRepository will handle the Firestore operations
- The UI should call repository methods instead of working with Firestore directly
- We will replace the local-only city storage with Firestore-backed storage

Use Firestore in CityRepository

- Add Firestore imports at the top of CityRepository
  - import androidx.compose.runtime.mutableStateListOf
  - import com.google.firebase.Firebase
  - import com.google.firebase.firestore.firestore
- Add a Firestore database instance and a reference to the cities collection
  - private val db = Firebase.*firestore*
    private val citiesRef = **db.collection**("**cities**")
- Firestore will store city data in a collection named “cities”

```
import androidx.compose.runtime.mutableStateListOf
import com.google.firebase.Firebase
import com.google.firebase.firestore.firestore
```

```
class CityRepository {  1 Usage
    private val db = Firebase.firestore  1 Usage
    private val citiesRef = db.collection( collectionPath = "cities")
```

Change City.kt for Firestore

- Firestore needs to convert documents into City objects
- We update City.kt so each property has a default value:

```
data class City(
    val name: String = "",
    val province: String = ""
)
```

Listen for City Updates

- Add a Snapshot Listener inside CityRepository.kt in an init block
- The Listener watches the cities collection and updates _cities whenever Firestore changes
- When Firestore changes:
  - Clear the local list
  - Read the city documents
  - Add them to_cities
  - Compose then updates the UI

```
init {
    citiesRef.addSnapshotListener { snapshot, error ->
        if (error != null) {
            return@addSnapshotListener
        }

        _cities.clear()

        snapshot?.documents?.forEach { document ->
            val city = document.toObject( valueType = City::class.java)
            if (city != null) {
                _cities.add(city)
            }
        }
    }
}
```

Save Cities to Firestore

- Before Firestore, addCity only updated the list locally:

```
fun addCity(city: City) {  1 Usage
    _cities.add(city)
}
```

- Now, we write to Firestore, which then triggers the snapshot listener, rebuilding _cities and updating the Compose UI:

```
fun addCity(city: City) {  1 Usage
    citiesRef.document( documentPath = city.name).set(city)
}
```

Update Cities in Firestore

- Now, change updateCity to use Firestore to find the document being updated

```
fun updateCity(oldCity: City, updatedCity: City) {  1 Usage
    citiesRef.document( documentPath = oldCity.name).set(updatedCity)
}
```

Run and Verify in Firebase Console

- Run the ListyCity app
- Add a new city using the app
- Edit an existing city
- Open the Firebase Console
- Check the cities collection in Firestore
- Restart the app and confirm the city list is still saved

Participation Exercise

- **Task**: In this exercise, your task is to add the ability to delete cities and integrate this functionality with Firestore so deletions persist
- After completing today's lab demo, add delete functionality to the ListyCity app
- When a city is deleted, delete the matching document from the Firestore cities collection
- The deleted city should disappear from the app UI
- Restart the app and confirm the deleted city does not reappear