Firebase in Android Authentication & Firestore

Firebase

Topics

- Setting Up Android App in Firebase
- Integration of Firebase Authentication in ViewModel
- Integration of Firebase Firestore in ViewModel

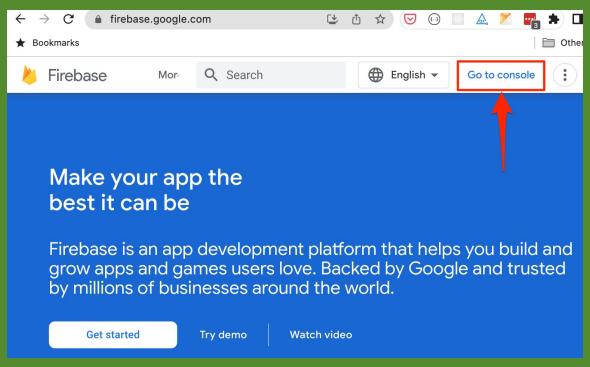
Prerequisites

Know how to

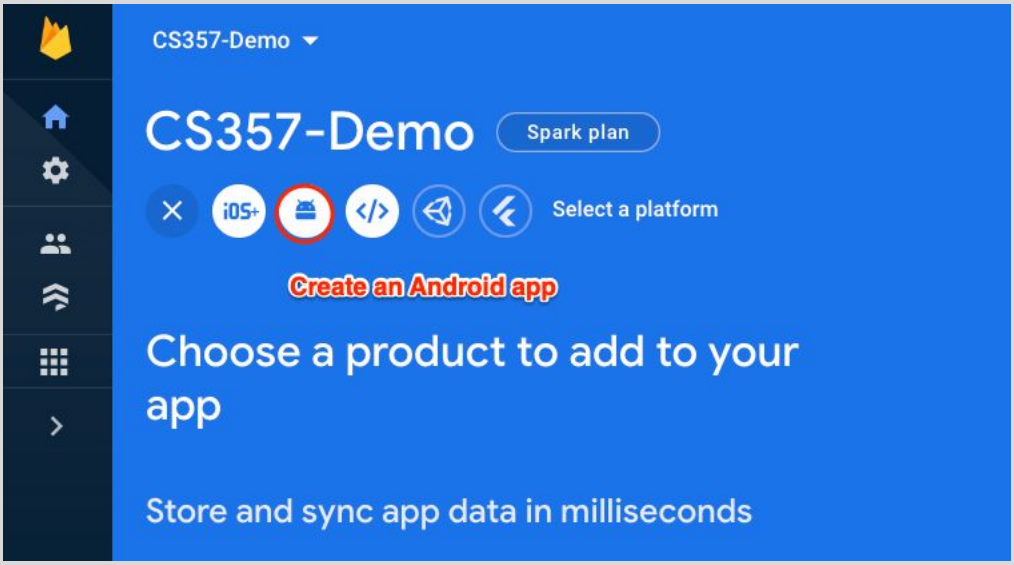
- Define ViewModel classes
- Use `MutableLiveData<>` and `LiveData<>` in ViewModel
- Use `.postValue` or `.setValue` to update MutableLiveData
- Wireup LiveData to UI
  - Use LiveData observers in View-Based
  - Use observeAsState() in Compose
- Working with Kotlin coroutines

Step 1: Create A Firebase Project

- Login to https://firebase.google.com
  - Use your GMail account
    - Try GVSU GMail (xxxxxx@mail.gvsu.edu)
    - If that does not work, use your personal GMail
- Go to Firebase Console
- Create a new project
  - Enter project name
  - Disable (or enable) Google Analytics

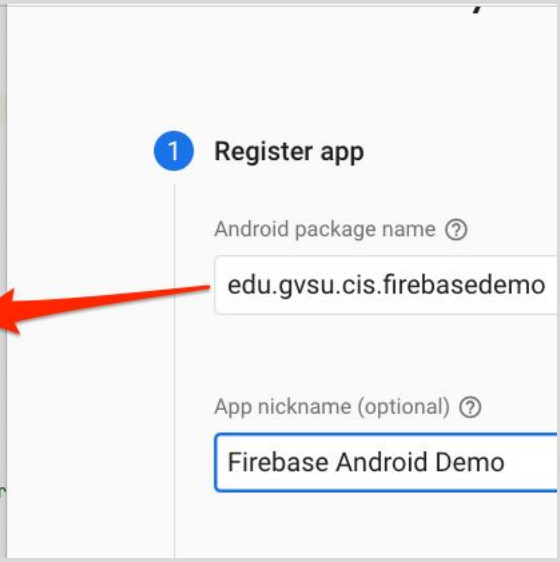


Step 2: Create Android App

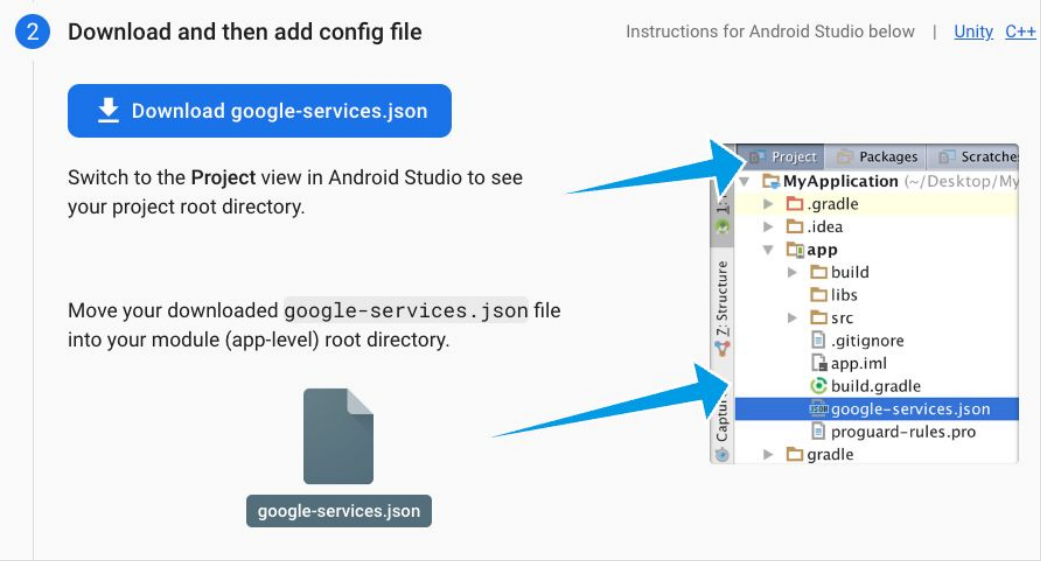


Step 3: Use applicationId (build.gradle) to register

```
plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
}

android {
    namespace 'edu.gvsu.cis.firebasedemo'
    compileSdk 33

    defaultConfig {
        applicationId "edu.gvsu.cis.firebasedemo"
        minSdk 24
        targetSdk 33
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner
    }
```



Step 4: Download JSON To Android Project



Step 5a: Add Library Dependencies

```
// Top-level build.gradle
buildscript {
    repositories {
        google()
        mavenCentral()
    }
    dependencies {
        classpath "com.google.gms:google-services:4.3.15"
    }
}
plugins {
    id 'com.android.application' version '7.4.2' apply false
    id 'com.android.library' version '7.4.2' apply false
    id 'org.jetbrains.kotlin.android' version '1.8.0' apply false
}
allprojects {
    repositories {
        google()
        mavenCentral()
    }
}
```

Exclude both brown text blocks if you are using Gradle 6.8 or newer.
And confirm that your `settings.gradle` *includes dependencyResolutionManagement with similar settings*

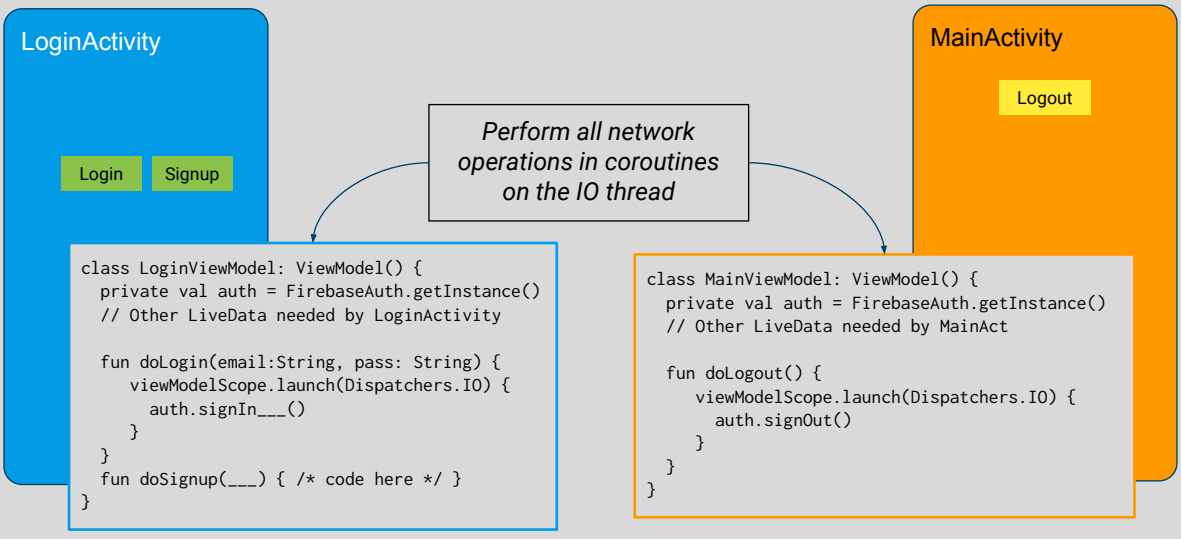
```
dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
        google()
        mavenCentral()
    }
}
```

Step 5b: Add Library Dependencies

```
plugins {
    id(“com.android.application”)
    id(“org.jetbrains.kotlin.android”)
    id(“com.google.gms.google-services”)
}

dependencies {
    // Add these two to the current dependencies
    implementation platform('com.google.firebase:firebase-bom:31.4.0')
    // Firebase Authentication and Firestore
    implementation(“com.google.firebase:firebase-auth-ktx”)
    implementation(“com.google.firebase:firebase-firestore-ktx”)

    // Lifecycle & coroutines
    implementation(“androidx.lifecycle:lifecycle-livedata-ktx:2.7.0”)
    implementation("androidx.lifecycle:lifecycle-viewmodel-ktx:2.7.0")
}
```


Firebase Authentication

Overall Design



SignInWithEmailAndPassword

```
// In LoginViewModel.kt
class LoginViewModel: ViewModel() {
  private val _uid: MutableLiveData<String?> = MutableLiveData(null)
  val uid: LiveData<String?> get() = _uid
  private val auth = Firebase.auth

  fun signIn(e:String, p:String) {
    viewModelScope.launch {
      auth.signInWithEmailAndPassword(e,p)
      // More code (next slide)
    }
  }
}
```

Handling signUp and signOut follows the same pattern

Handling Authentication Result / Errors

Option #1

```
fun signIn(e:String, p:String) {
    viewModelScope.launch {
        try {
           val result = auth.signInWithEmailAndPassword(e,p).await()
           println ("Your UID is ${result.user!!.uid}")
        } catch(e: Exception) {
           println ("Unable to login: ${e.message}")
        }
    }
}
```

Option #2

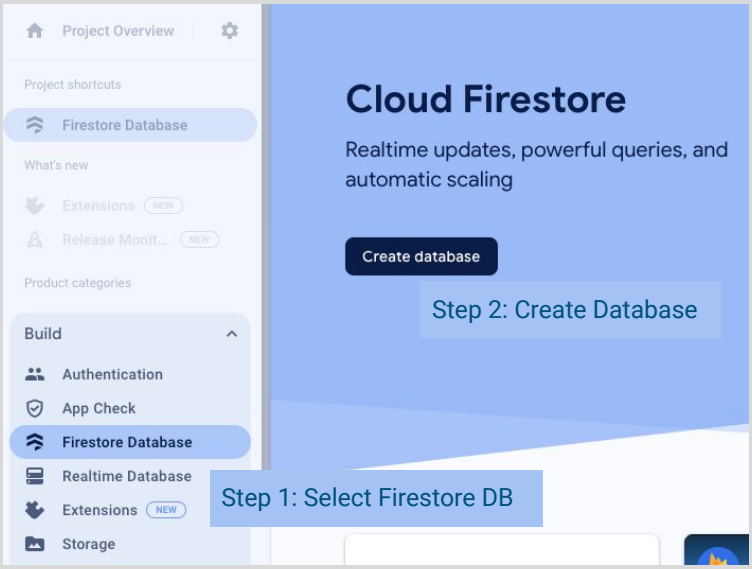
```
fun signIn(e:String, p:String) {
    viewModelScope.launch {
        auth.signInWithEmailAndPassword(e,p)
            .addOnSuccessListener { result ->
               println ("Your UID is ${result.user!!.uid}")
            }
            .addOnFailureListener { e ->
               println ("Unable to login: ${e.message}")
            }
        }
    }
}
```

Online Documentation

- General Firebase: https://firebase.google.com/docs
- Firebase Authentication: https://firebase.google.com/docs/auth
- Firebase Auth for Android: https://firebase.google.com/docs/auth/android/start

Firebase Firestore

Prerequisite: Create New Firestore instance



Firebase Firestore

- Cloud Database
- No SQL no DB Schema required to create tables
- Data are organized into Collections and Documents

SQL	Cloud Firestore
Tables	Collections
Rows	Documents
Primary Key	Document ID
Fields	key-value pairs

Summary of CRUD Operations

```
// Do this first
val db = Firebase.firestore
```

	Collection	Document
Reference	`val collRef = db.collection("collName")`	`val docRef = db.collection("")` `.document("xx")`
Create	**No API provided**	`docRef.set(___)` `collRef.add(___)`
Read	`collRef.get() // All documents`	`docRef.get()`
Update	N/A	`docRef.set(___)`
Delete	**No API provided**	`docRef.delete()`

All these functions takes a *completion* closure

No API to Create/Delete Collections???

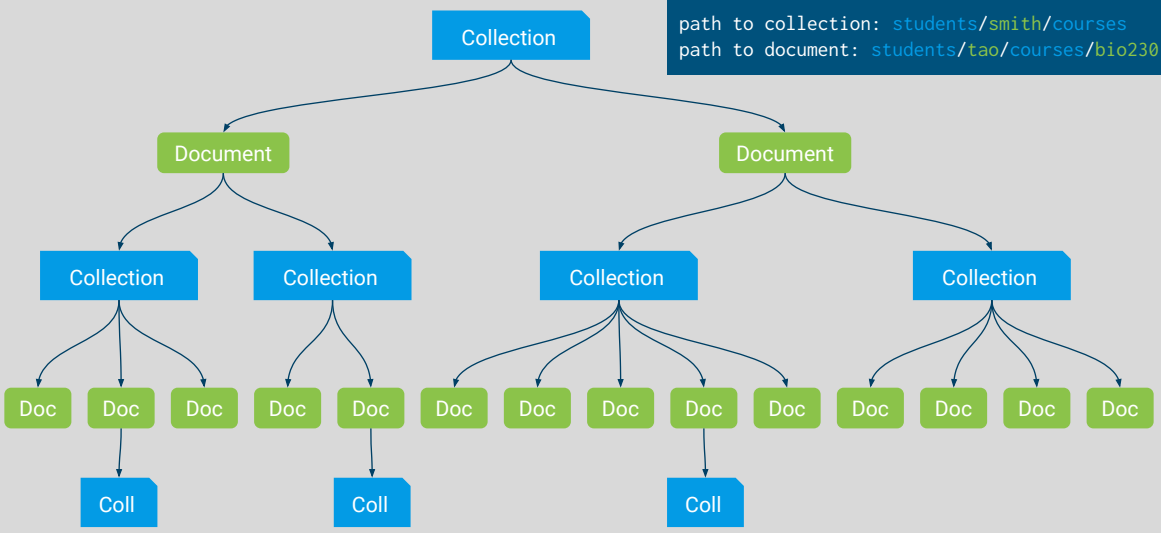
- You are **not allowed** to create empty collections
- A collection is automatically created when you create the first document in the collection

```
val db = Firebase.firestore
// This will create BOTH the
// "students" collection and the
// "G00123" document
db.document("students/G00123")
   .set(/* student_info_obj_here */)
```

- Likewise, you are **not allowed** to delete a (non-empty) collection
  - You must use a loop to delete each document in the collection
  - If it ends up in an empty collection, the collection is automatically removed

Google wants you to pay for every DELETE operation!!!

Firestore Data Organization



Using Kotlin Data Class

```
data class Course (
    val name:String = “”
    val credits: Int = 0
    @field:JvmField // required when the field name begins with “is”
    val isRequired: Boolean = false
)
```

Each field in the data class must have a default initializer

Using Coroutine in ViewModel classes

```
class YourViewModel: ViewModel() {
  private val db = Firebase.firestore

  fun doSomeWorkOnCollection() {
      viewModelScope.launch(Dispatchers.IO) {
          db.collection("_____").someFirebaseOperationHere()
            .addSuccessListener { /* handle result here */ }
            .addFailureListener { /* handle error here */ }
      }
  }

  fun doSomeWorkOnDocument() {
      viewModelScope.launch(Dispatchers.IO) {
          db.document("_____").someFirebaseOperationHere()
            .addSuccessListener { /* code here */ }
            .addFailureListener { /* code here */ }
      }
  }
}
```


addXXXListener() vs. await()

```
fun doSomeWorkOnCollection() {
    viewModelScope.launch(Dispatchers.IO) {

        db.collection("_____")
          .someFirebaseOperationHere()
          .addSuccessListener { res ->
              // Handle result here
              print("OK $res")
           }
          .addFailureListener { err ->
              /* handle error here */
              print("Nope $e")
          }
      }
  }
}
```

```
fun doSomeWorkOnCollection() {
    viewModelScope.launch(Dispatchers.IO) {
      try {
          val res = db.collection("_____")
            .someFirebaseOperationHere()
            .await()
          /* handle result here */
          print("OK $res")
       }
       catch(e: Exception) {
          /* handle error here */
          print ("Nope $e")
       }
    }
  }
}
```

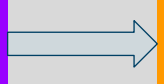
ALWAYS

Call the Firebase Auth/Firebase Firestore functions inside a *Coroutine scope*

Kotlin Data Class ⇒ Firestore Document

```
data class Course (
    val name:String = “”
    val credits: Int = 0
    @field:JvmField
    val isRequired: Boolean = false
    val instructors: List<String> = []
)
```

```
// Object In Kotlin
Course(“BIO100”, 3, true, [“John”, “Amy”])
```



```
/* Document In Firestore */
name:           “BIO100”
credits:        3,
isRequired:     true
instructors:
     0:    “John”
     1:    “Amy”
```

CRUD: Create Documents (with auto ID)

```
INSERT INTO courses(name, credits, isrequired) VALUES(“CS162”, 4, TRUE)
```

```
val newCourse = Course(name = “CS162”, credits = 4, isRequired = true)

db.collection(“courses”)                  // autogenerated document ID
  .add(newCourse)
  .addSuccessListener {
     println(“A new course created with ID ${it.id}”)
   }
   .addFailureListener {
     println("Failed to add document ${it.message}")
   }
```

```
// Using await()
val newCourse = Course(name = “CS162”, credits = 4, isRequired = true)

val newRef = db.collection(“courses”).add(newCourse).await()
println(“A new course created with ID ${newRef.id}”)
```


CRUD: Create Documents (with your own ID)

```
INSERT INTO courses(name, credits, isrequired) VALUES(“CS162”, 4, TRUE)
```

```
val newCourse = Course(name = “CS162”, credits = 4, isRequired = true)

db.collection(“courses”).document(“w2020cs16204”) // your own doc ID
  .set(newCourse)
  .addSuccessListener {
     println(“A new course created with ID w2020cs16204”)
   }
   .addFailureListener {
     println("Failed to add document ${it.message}")
   }
```

CRUD: Read A Specific Document (known DocID)

```
SELECT FROM courses WHERE id = “JFHGDFS3656”
```

```
db.document(“courses/JFHGDFS3656”)
  .get()
  .addSuccessListener {
     val crs:Course = it.toObject(Course::class.java)
     print("DocID ${it.id } Course ${crs.name}")
  }
  .addFailureListener {
     print("Failed to fetch document ${it.message}")
  }
```

Use `toObject()` to convert Firestore raw dictionary to **Kotlin data class**

Firestore Document ⇒ Kotlin Data Object

```
/* Document In Firestore */
name:              “BIO100”
credits:           3,
isRequired:        true
instructors:
      0:     “John”
      1:     “Amy”
```

toObject(Course::java.class)

```
data class Course (
    val name:String = “”
    val credits: Int = 0
    @field:JvmField
    val isRequired: Boolean = false
    val instructors: List<String> = []
)
```

```
// Object In Kotlin
Course(“BIO100”, 3, true, [“John”, “Amy”])
```

toObject(): unmatched names

```
data class Kourse (
    val name:String = “”
    val credit: Int = 0
    @field:JvmField
    val isRequired: Boolean = false
)
```

```
/* Document In Firestore
*/
name:           “BIO100”
credits:        3,
isRequired:     true
instructors:
     0:    “John”
     1:    “Amy”
```

toObject(Kourse::java.class)

```
// Object In Kotlin
Kourse(“BIO100”, 0, true)
```


CRUD: Update a Document (known DocID)

```
UPDATE courses SET isrequired = false WHERE id = “MTH100”
```

```
val newMath100 = Course(name = “____”,
                        credits = 5,
                        isRequired = false)
db.document(“courses/MTH100”)
  .set(newMath100)
  .addSuccessListener {
    println(“Document updated”)
  }
  .addFailureListener {
     println(“Failed to update document ${it.messsage}”)
  }
```

CRUD: Delete a Document (known DocID)

```
DELETE FROM courses WHERE id = “MTH100”
```

```
db.document(“courses/MTH100”)
  .delete()
  .addSuccessListener {
      println(“Document deleted”)
  }
  .addFailureListener {
      println(“Failed to delete document ${it.messsage}”)
  }
```


CRUD: Delete All Documents in a Collection

```
DELETE * FROM courses
```

```
// Delete each individual document
db.document(“courses”)
  .get()
  .addSuccessListener {
      for (d in it.documents) {
         d.reference.delete()
      }
  }
  .addFailureListener {
      println(“Failed to delete documents”)
  }
```

CRUD: Read All Documents in a Collection

```
SELECT * FROM courses
```

```
db.collection("courses")
  .get()
  .addSuccessListener {
      for (doc in it.documents) {
          doc.toObject(Course::class.java)?.let {
             println("${doc.id} ${it.name} ${it.credits}")
          }
      }
   }
  .addFailureListener {
      println("Failed to delete document ${it.messsage}")
  }
```

Use `toObject()` to convert Firestore raw dictionary to **Kotlin data class**

CRUD: Query Documents in a Collection

```
SELECT * FROM courses WHERE credits > 3
```

```
db.collection("courses")
  .whereGreaterThan("credits", 3)
  .get()
  .addSuccessListener {
      for (doc in it.documents) {
          doc.toObject(Course::class.java)?.let {
             println("${doc.id} ${it.name} ${it.credits}")
          }
      }
   }
  .addFailureListener {
      println("Failed to delete document ${it.messsage}")
  }
```

Other where___() functions

Operator	Example
<, <=, ==, >=, >	whereEqualTo(“credits”, 3) whereNotEqualTo(“credits”, 3) whereLessThan(“credits”, 3) whereLessThanOrEqualTo(“credits”, 3) whereGreaterThan(“credits”, 3) whereGreaterThanOrEqualTo(“credits”, 3)

Operator	Example (prereqs must be an ARRAY in the course document)

array-contains	// Is MTH200 a prereq?
	whereArrayContains(“prereqs”, “MTH200”)

array-contains-any	// Is either MTH200 or STA215 a prereq?
	whereArrayContainsAny(“prereqs”, [“MTH200”, “STA215”])

Listening for Changes

```
data class Student(...)

class YourViewModel: ViewModel() {
  private val db = Firebase.firestore
  init { // Must be installed in the constructor
    db.collection(“name-of-the-collection”)
      .addSnapshotListener { value, err ->
        value?.let {
          for ((idx,chg) in it.documentChanges.withIndex()) {
            val updatedDoc = chg.document.toObject(Student::class.java)
            val modType = chg.type.name // ADDED, MODIFIED, REMOVED
            println("Doc ${chg.document.id} ${updatedDoc} ${modType}")
          }
        }
      }
    }
  }
}
```

Advanced Topic

Connecting to Multiple Firebase Projects

Preparation Steps

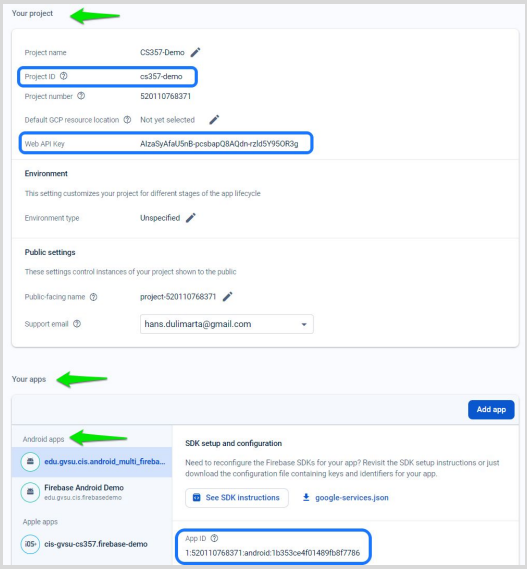
- Remove Google Services (Plugins & .json file)
- Obtain Firebase Project Settings
  - Project ID
  - API Key
  - App ID
- Initialize each Firebase Project in your ViewModel

Step 1: Remove Google Services Plugins

```
// Top-level build.gradle.kts
buildscript {
    dependencies {
        classpath "com.google.gms:google-services:4.3.15"
    }
}
```

```
// App build.gradle.kts
plugins {
    id("com.android.application")
    id("org.jetbrains.kotlin.android")
    id("com.google.gms.google-services")
}
```

Step 2: Get Details of Firebase Project Settings



From your Firebase Project
- Project ID
- Web API Key

From your Firebase App
- App ID

Step 3: Initialize Firebase Instances

```
class MyViewModel(app:Application): AndroidViewModel(app) {
   val option1 = FirebaseOptions.Builder()
       .setProjectId("cs357-demo").setApiKey("____").setApplicationId("____").build()
   val option2 = FirebaseOptions.Builder()
       .setProjectId("__________").setApiKey("____").setApplicationId("____").build()
   var firstDB: FirebaseFirestore
   var secondDB: FirebaseFirestore

   init {
       val ctx = getApplication<Application>().applicationContext
       Firebase.initialize(ctx, option1, "fb_app_1")
       Firebase.initialize(ctx, option2, "my_firebase_app_2")
       firstDB = Firebase.firestore(Firebase.app("fb_app_1"))
       secondDB = Firebase.firestore(Firebase.app("my_firebase_app_2"))
    }
}
```

Parent class is `AndroidViewModel()`, not `ViewModel()`

Step 4: Use the Database(s)

```
class MyViewModel(app:Application): AndroidViewModel(app) {
    val opt1 = ___
    val opt2 = ___
    var firstDB: FirebaseFirestore
    var secondDB: FirebaseFirestore

    init {
    }

    fun fetchSomeData() {
        firstDB.collection("name-of-the-collection").get()
            .addSuccessListener { /* your code here */  }
            .addFailureListener { /* your code here */  }
        }
    }
}
```

From previous slide

Reading Assignment

Engelsma/Dulimarta textbook

- Chapters 8