

Firestore

Authentication & Firestore



Firestore

Topics

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

On GitHub: [Android Firebase Compose](https://github.com/dulimarta/cs357-code-example)

```
git clone https://github.com/dulimarta/cs357-code-example  
cd androidfirebasecompose
```

Prerequisites

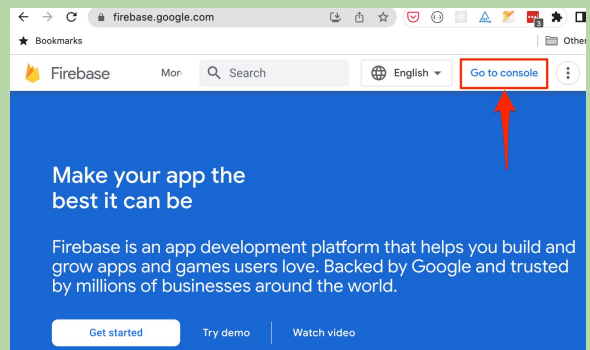
Know how to

- Define ViewModel classes
- Use MutableStateFlow<>
- Wireup StateFlow to UI
 - Use collectAsState() in Compose
- Working with Kotlin coroutines

3

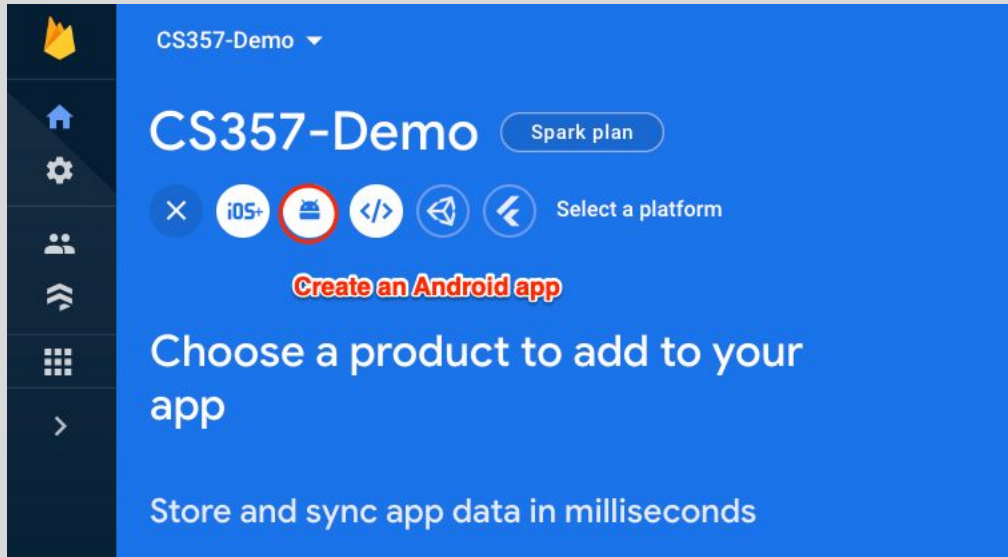
Step 1: Create A Firebase Project

- 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



4

Step 2: Create Android App



5

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

```
plugins {  
    id 'com.android.application'  
    id 'org.jetbrains.kotlin.android'  
}  
  
android {  
    namespace 'edu.gvsu.cis.firbasedemo'  
    compileSdk 33  
  
    defaultConfig {  
        applicationId "edu.gvsu.cis.firbasedemo"  
        minSdk 24  
        targetSdk 33  
        versionCode 1  
        versionName "1.0"  
  
        testInstrumentationRunner "androidx.test.runner"    }  
}
```

1 Register app

Android package name ?

edu.gvsu.cis.firbasedemo

App nickname (optional) ?

Firebase Android Demo

6

Step 4: Download JSON To Android Project

2 Download and then add config file

Instructions for Android Studio below | [Unity](#) [C++](#)

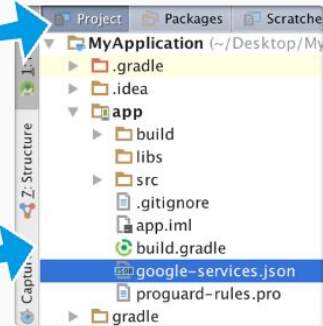
Download google-services.json

Switch to the Project view in Android Studio to see your project root directory.

Move your downloaded google-services.json file into your module (app-level) root directory.



google-services.json



7

Step 5a: Add Library Dependencies

```
// Top-level build.gradle
buildscript {
    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
}
```

8

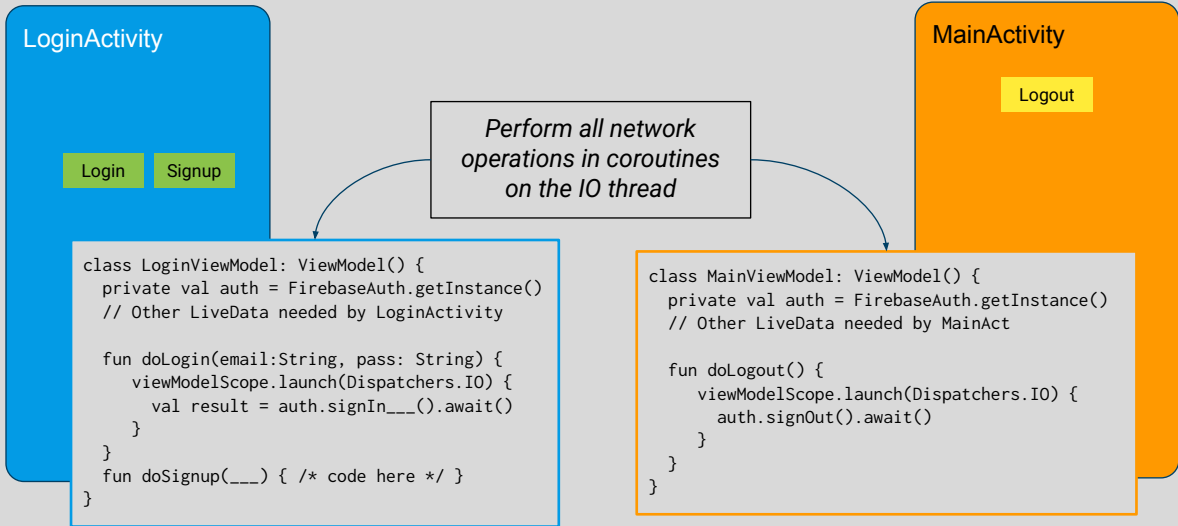
Step 5b: Add Library Dependencies

```
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:33.4.0")  
    // Firebase Authentication and Firestore  
    implementation("com.google.firebase:firebase-auth-ktx")  
    implementation("com.google.firebase:firebase-firestore-ktx")  
    // Lifecycle & coroutines  
    implementation("androidx.lifecycle:lifecycle-viewmodel-ktx")  
}
```

Bill of Material (BOM) collectively manages compatible libraries

Firebase Authentication

Overall Design



11

SignInWithEmailAndPassword

```
// In LoginViewModel.kt
class LoginViewModel: ViewModel() {
    private val _uid: MutableStateFlow<String?>(null)
    val uid = _uid.asStateFlow()
    private val auth = FirebaseAuth

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

Handling signUp and signOut follows the same pattern

12

Handling Authentication Result / Errors

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

Option #1

```
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}")  
            }  
    }  
}
```

Option #2

DON'T USE THIS

13

Online Documentation

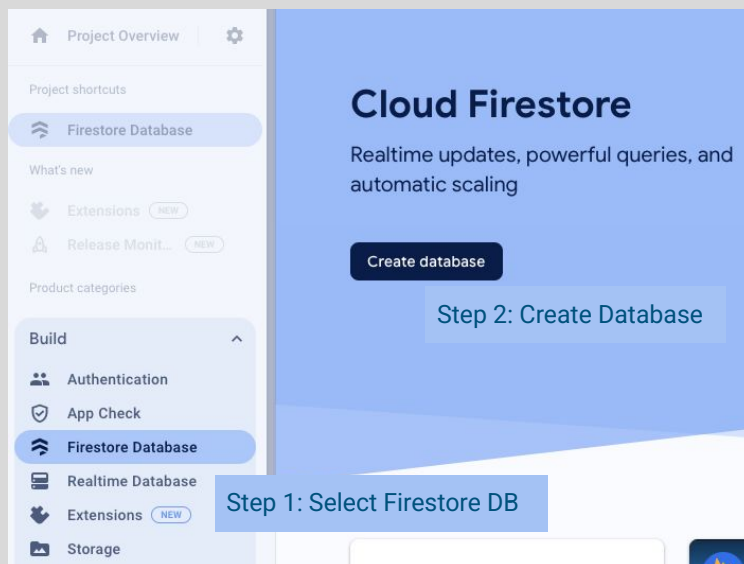
- General Firebase: <https://firebase.google.com/docs>
- Firebase Authentication: <https://firebase.google.com/docs/auth>
- Firebase Auth for Android:
<https://firebase.google.com/docs/auth/android/start>

14

Firestore

20

Prerequisite: Create New Firestore instance



21

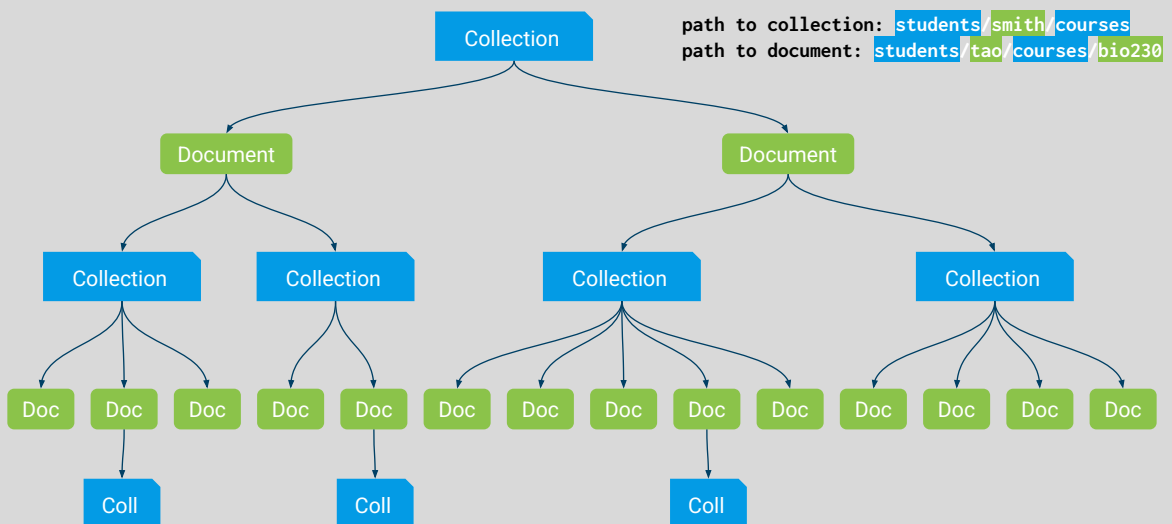
Firestore

- 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

22

Firestore Data Organization



23

Summary of CRUD Operations

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

	Collection	Document
Reference	<code>val collRef = db.collection("collName")</code>	<code>val docRef = db.collection("") .document("xx")</code>
Create	No API provided, collections are automatically created (when creating the first document in it)	<code>docRef.set(__).await() collRef.add(__).await()</code>
Read	<code>collRef.get().await()</code> // All docs in collection	<code>docRef.get().await()</code>
Update	N/A	<code>docRef.set(__).await()</code>
Delete	No API provided	<code>docRef.delete().await()</code>

24

Path to ...

path to collection: `students/smith/courses`

path to document: `students/tao/courses/bio230`

```
val collRef = db.collection("students/smith/courses")  
val cRef    = db.collection("students").document("smith").collection("courses")  
  
val docRef  = db.document("students/tao/courses/bio230")  
val dRef    = db.collection("students").document("tao")  
            .collection("courses").document("bio230")
```

25

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
- 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

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

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

26

Kotlin Data Class (with Firestore)

```
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

27

Using Coroutine in ViewModel

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

    fun doSomeWorkOnCollection() {
        viewModelScope.launch(Dispatchers.IO) {
            try {
                // make it into a suspendable call
                val result = db.collection("_____").someFirestoreOperationHere().await()
                // OR val result = db.document("_____").someFirestoreOperationHere().await()
                // Code here to handle the result
            } catch (e: Error) {
                // Code here to handle error
            }
        }
    }
}
```

28

CRUD: Read All Documents in a Collection

```
try {
    val colRef = db.collection("courses").get().await()
    for (doc in colRef.documents) {
        val cc: Course = doc.toObject<Course>()
        println("${cc.id} ${cc.name} ${cc.credits}")
    }
} catch (e: Exception) {
    println("Failed to fetch documents ${e.message}")
}
```

SELECT * FROM courses

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

29

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")
            }
    }
}
```

DONT USE THIS

```
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")
        }
    }
}
```

USE THIS

30

ALWAYS

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

31

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("BI0100", 3, true, ["John", "Amy"])
```

```
/* Document In Firestore */  
name:      "BI0100"  
credits:   3,  
isRequired: true  
instructors:  
  0:      "John"  
  1:      "Amy"
```

32

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}")
```

33

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)

try {
    val newDoc = db.collection("courses").document("w2020cs16204") // your own doc ID
        .set(newCourse).await()
    println("A new course created with ID w2020cs16204")
}
catch (e: Exception) {
    println("Failed to add document ${it.message}")
}
```

34

CRUD: Read A Specific Document (known DocID)

```
SELECT FROM courses WHERE id = "JFHGDFS3656"
```

```
try {
    val docRef = db.document("courses/JFHGDFS3656").get().await()
    val crs:Course = docRef.toObject<Course>() // Type-Safe
    print("DocID ${it.id } Course ${crs.name}")
} catch (/* arg */) {
}
```

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

```
try {
    val docRef = db.document("courses/JFHGDFS3656").get().await()
    val crsName = docRef.get("name") // DON'T USE THIS (Not Type-Safe)
} catch (/* arg */) {
}
```

35

Firestore Document \Rightarrow Kotlin Data Object

```
/* Document In Firestore */  
name:          "BI0100"  
credits:       3,  
isRequired:    true  
instructors:  
  0:  "John"  
  1:  "Amy"
```

toObject<Course>()

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

```
// Object In Kotlin  
Course("BI0100", 3, true, ["John", "Amy"])
```

36

toObject(): unmatched names

```
/* Document In Firestore  
*/  
name:          "BI0100"  
credits:       3,  
isRequired:    true  
instructors:  
  0:  "John"  
  1:  "Amy"
```

toObject<Kourse>()

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

```
// Object In Kotlin  
Kourse("BI0100", 0, true)
```

37

CRUD: Update a Document (known DocID)

```
UPDATE courses SET isrequired = false WHERE id = "MTH100"
```

```
val newMath100 = Course(name = "____",
                        credits = 5,
                        isRequired = false)
try {
    val docRef = db.document("courses/MTH100").set(newMath100).await()
    println("Document updated ${docRef.id}")
}
catch(e: Exception) {
    println("Failed to update document ${e.message}")
}
```

38

CRUD: Delete a Document (known DocID)

```
DELETE FROM courses WHERE id = "MTH100"
```

```
try {
    db.document("courses/MTH100").delete().await()
    println("Document deleted")
}
catch(e: Exception) {
    println("Failed to delete document ${e.message}")
}
```

39

CRUD: Delete All Documents in a Collection

```
DELETE * FROM courses
```

```
try {
    val collRef = db.collection("courses").get().await()
    for (d in collRef.documents) {
        // Delete each individual document
        await d.reference.delete().await()
    }
    print("Add docs under courses are deleted")
}
catch(/* arg */) {
    println("Failed to delete documents")
}
```

40

CRUD: Query Documents in a Collection

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

```
val colRef = db.collection("courses")
    .whereGreaterThan("credits", 3)
    .get().await()
for (doc in colRef.documents) {
    val cc: Course = doc.toObject<Course>()
}
```

41

Other where___() functions

Operator	Example
<, <=, ==, >=, >	<pre>whereEqualTo("credits", 3) whereNotEqualTo("credits", 3) whereLessThan("credits", 3) whereLessThanOrEqualTo("credits", 3) whereGreaterThan("credits", 3) whereGreaterThanOrEqualTo("credits", 3)</pre>

Operator	Example (prereqs must be an ARRAY in the course document)
array-contains	<pre>// Is MTH200 a prereq? whereArrayContains("prereqs", "MTH200")</pre>
array-contains-any	<pre>// Is either MTH200 or STA215 a prereq? whereArrayContainsAny("prereqs", ["MTH200", "STA215"])</pre>

42

Advanced Topic

43

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 { qSnap, err ->
                if (qSnap != null) {
                    for (chg in qSnap.documentChanges) {
                        val updatedDoc = chg.document.toObject<Student>()
                        val modType = chg.type.name // ADDED, MODIFIED, REMOVED
                        println("Doc ${chg.document.id} ${updatedDoc} ${modType}")
                    }
                }
            }
    }
}
```

44

Connecting to Multiple Firebase Projects

45

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

46

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")
}
```

47

Step 2: Get Details of Firebase Project Settings

Your project

Project name: CS357-Demo

Project ID: cs357-demo

Project number: 520110768371

Default GCP resource location: Not yet selected

Web API Key: Alz8yMfu59b-gcsbap08AD0trzH5Y95OR5g

Environment

Environment type: Unspecified

Public settings

Public-facing name: project-520110768371

Support email: hans.dulimarta@gmail.com

Your apps

Android apps

edu.gvsu.cis.android_multi_firebase...

SDK setup and configuration

App ID: 1:520110768371:android:1b333ce4f01489fb8f7786

From your Firebase Project

- Project ID
- Web API Key

From your Firebase App

- App ID

48

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()

49

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