CMPS 312 Mobile App Development
Lab 11 – Cloud FireStore

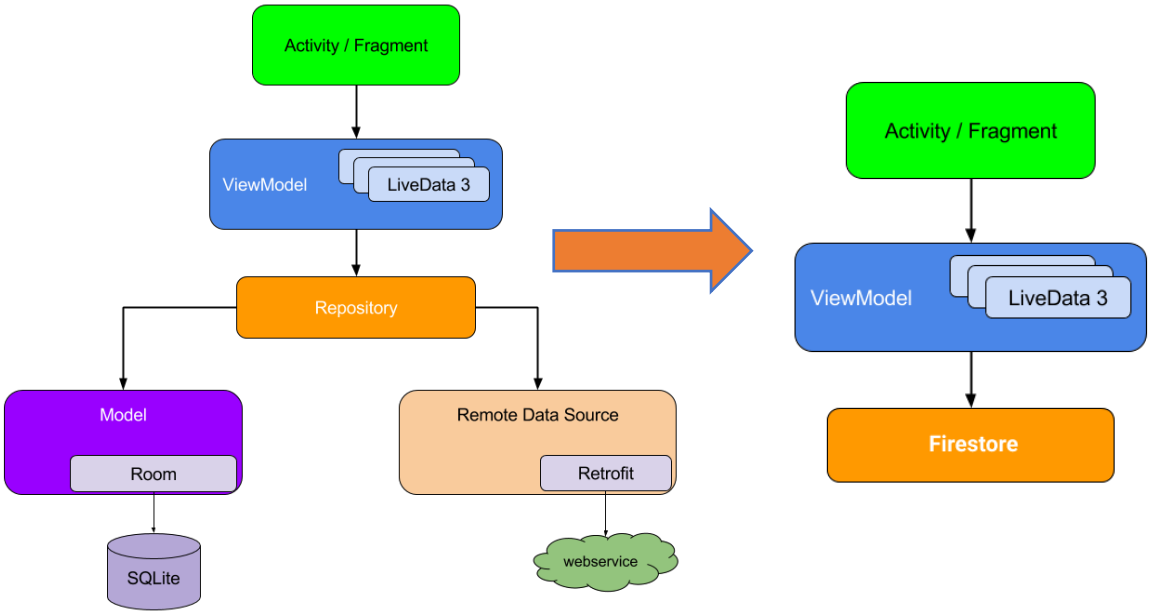
Objective

In this Lab, you will continue with the **Todo app** that you created in Lab 10 and replace the local offline SQLite/room database with **Cloud Firestore; which** is a NoSQL document database that lets you easily store, sync, and query data for your mobile apps - at global scale.

In this Lab you will practice:

- Read and write data to Firestore from an Android app
- Listen to changes in Firestore data in realtime
- Use Firebase Authentication and security rules to secure Firestore data
- Write complex Firestore queries

The image below shows how we replace the Room database in the recommended MVVM architecture with Cloud **FireStore**

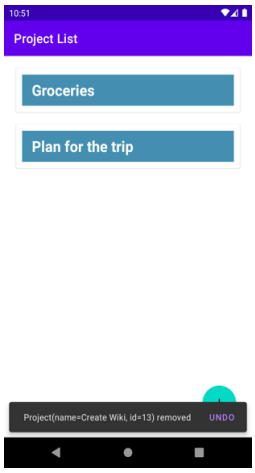


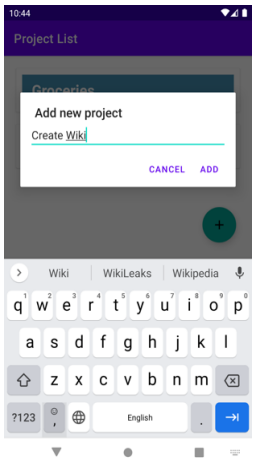
Preparation

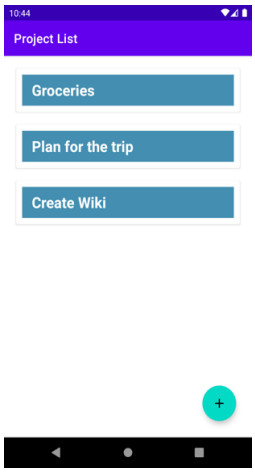
1. Sync the Lab GitHub repo and copy the **Lab 11-Cloud FireStore** folder into your repository.
2. Import the TodoList project into Android Studio. You will probably see some compilation errors or maybe a warning messages. We'll correct this in the next sections.

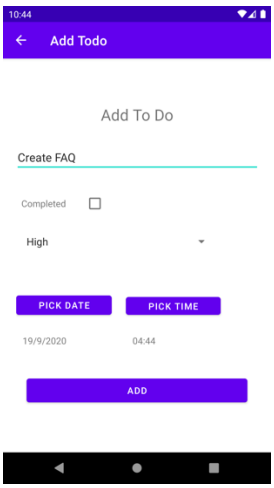
PART A: Implementing the Todo App

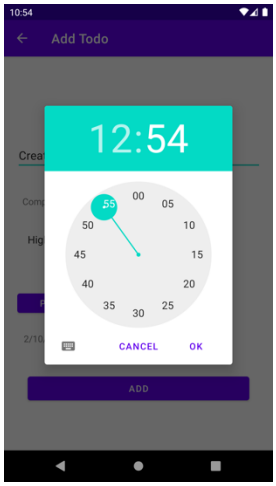
In this lab you will re-implement the Todo app you created in Lab 10 and replace the database with Cloud Fire store. The functionality of the application will still be the same get, add, update delete projects and to-dos.

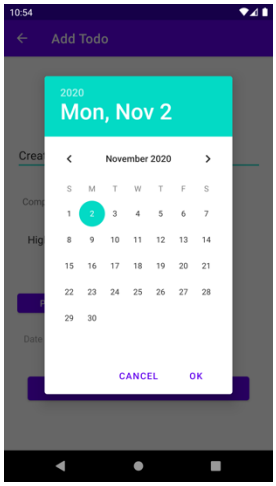


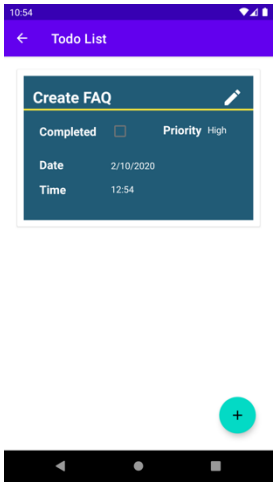








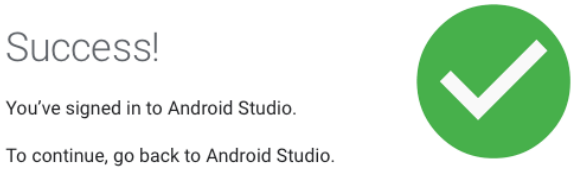




Task 1: Creating Firebase Project

The Firebase Assistant registers your app with a Firebase project and adds the necessary Firebase files, plugins, and dependencies to your Android project — all from within Android Studio!

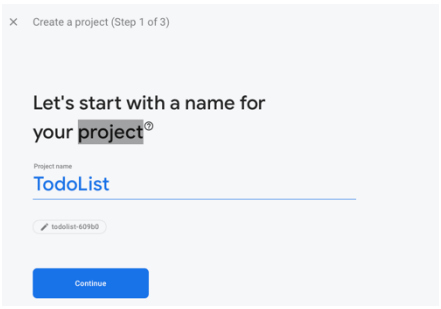
1. Sign in to your google account inside Android Studio

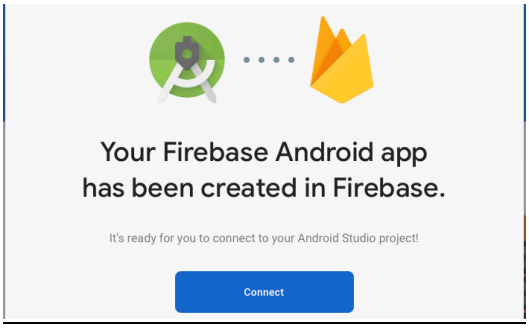


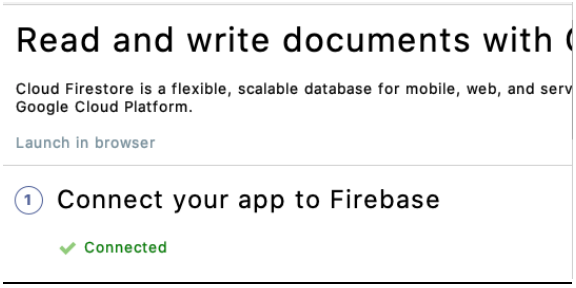
2. Open your Android project in Android Studio and access the Firebase Assistant:
    a. Go to File > Check for updates to make sure that you're using the latest versions of Android Studio and the Firebase Assistant.
    b. Go to Tools > Firebase to open the Assistant pane.
3. Choose a Firesore product to add to your app. Expand its section, then click the tutorial link (for example, Analytics > Log an Analytics event).
    a. Click Connect to Firebase to connect your Android project with Firebase.

    What does this workflow do?
    b. Click the button to add a desired Firebase product (for example, Add Analytics to your app).

    As shown in the screen capture below, enter a name for your FireStore project (for example, "TodoList App"), and click Continue.







4. Sync your app to ensure that all dependencies have the necessary versions.

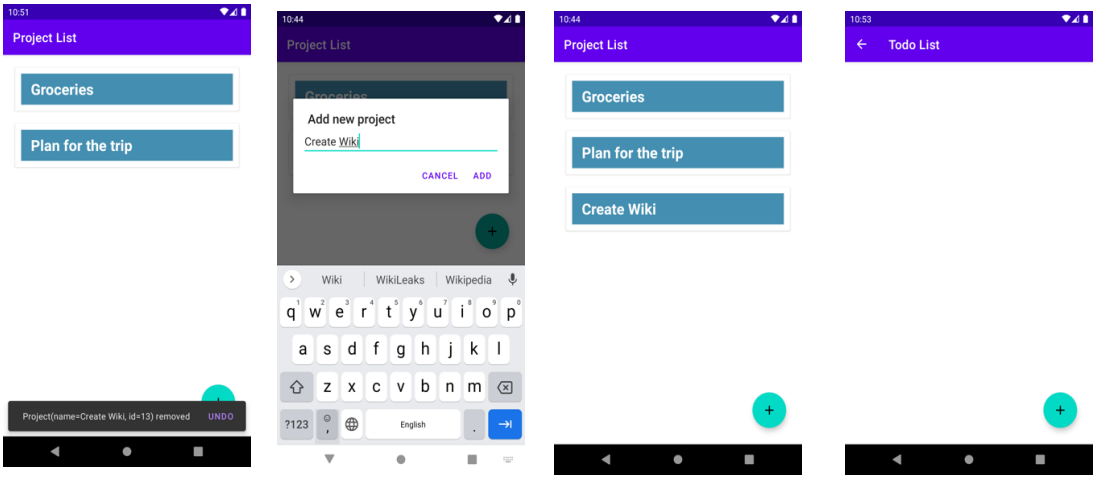
5. Replace the auto generated dependencies in build.graidle app with by the following lines of code.

```
implementation platform('com.google.firebase:firebase-bom:26.0.0')

implementation 'com.google.firebase:firebase-firestore-ktx'
implementation 'com.google.firebase:firebase-auth-ktx'
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-play-services'

// FirebaseUI (for authentication)
implementation 'com.firebaseui:firebase-ui-auth'
implementation 'com.google.android.gms:play-services-auth'
```

➔The Firebase Android BoM (Bill of Materials) enables you to manage all your Firebase library versions by specifying only one version — the BoM's version. This avoid future crushes due to differences in build number between firebase products.

Task 2: Write , Read and Query Data from Cloud Firestore Database

In this section we will write , read and query specific projects and todo list data from **Firestore** so that we can populate the data on the screen.
Note : It is possible to enter data manually in the Firebase console,



1. Open the repository class TodoListRepo
2. Create one object called db that holds the **FirebaseFirestore** interface

```
val db: FirebaseFirestore by lazy {
    FirebaseFirestore.getInstance()
}
```

3. Create two more instance variables inside the TodoListRepo that will hold the two different collections . One for tod list and another one for projects.

```
val projectDocumentsRef by lazy { db.collection("projects") }
val todoDocumentsRef by lazy { db.collection("todos") }
```

4. Enable offline caching for the database

```
init {
    //enable offline caching
    val settings = firestoreSettings { isPersistenceEnabled = true }
    db.firestoreSettings = settings
}
```

5. Implement the following functions using the `projectDocumentsRef and todoDocumentsRef` references.

```
fun getProjects():List<Project>
suspend fun addProject(project: Project)
suspend fun deleteProject(project: Project)

suspend fun getTodoListByProject(pid: String): List<Todo>
suspend fun getTodo(id: String): Todo
suspend fun addTodo(todo: Todo) : Long
suspend fun updateTodo(todo: Todo)
suspend fun deleteTodo(todo: Todo): Int
```

Task 3: Getting realtime updates with Cloud Firestore

You can **listen** to a document with the **addSnapshotListener()** method. An initial call using the callback you provide creates a document snapshot immediately with the current contents of the single document.
Then, each time the contents change, another call updates the document snapshot. Therefore in this part we will try to replace the manual update of the getting the project and todos with a dynamic update. We will read the content immediately as soon as it becomes available.

Open the **ProjectViewModel** and register the following two listeners [**registerProjectlistener**] [**registerTodolistener**]

```
private fun registerProjectlistener() {
        TodoListRepo.projectDocumentsRef
            .addSnapshotListener { snapshot, e ->
                if (e != null) {
                    return@addSnapshotListener
                }
//                val updatedProjects = snapshot!!.toObjects(Project::class.java)

                val updatedProjectDocuments = mutableListOf<Project>()
                snapshot!!.forEach { doc ->
                    Log.d("TAG", doc.id)
                    run {
                        val p = doc.toObject(Project::class.java)
                        p.projectId = doc.id
                        updatedProjectDocuments.add(p)
                    }
                }
                _projects.value = updatedProjectDocuments

            }
    }
```

```
private fun registerTodolistener() {
    TodoListRepo.todoDocumentsRef.addSnapshotListener { snapshot, e ->
        if (e != null) {
            Log.w(TAG, "Listen failed.", e)
            return@addSnapshotListener
        }

        if (selectedProject != null) {
            val updatedTodoList = snapshot!!.toObjects(Todo::class.java)
            _todos.value = updatedTodoList.filter { it.projectId ==
selectedProject?.projectId }
        }

        val source = if (snapshot != null && snapshot.metadata.hasPendingWrites())
            "Local"
        else
            "Server"

        if (snapshot != null && !snapshot.isEmpty) {
            Log.d(TAG, "$source data: ${snapshot.documents}")
        } else {
            Log.d(TAG, "$source data: null")
        }
    }
}
```