CMPS 312 Mobile App Development
Lab 11 – Cloud Firestore

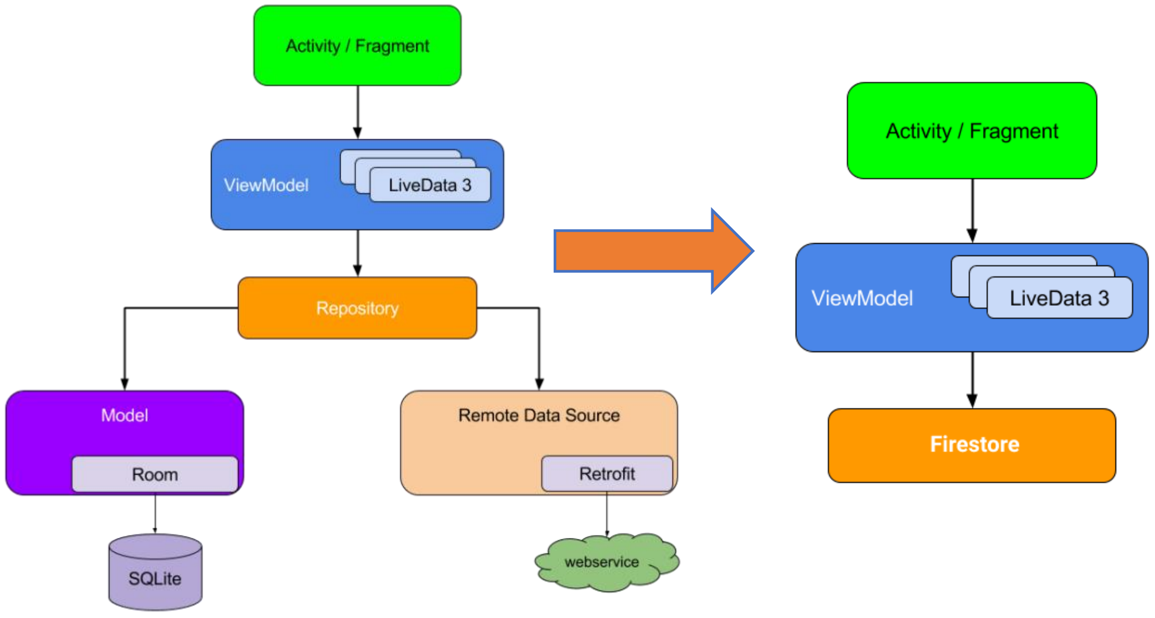
Objective

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.
- Query Firestore collections.
- Listen to changes in Firestore data in realtime.

The image below shows MVVM architecture with Cloud **Firestore** instead of room.

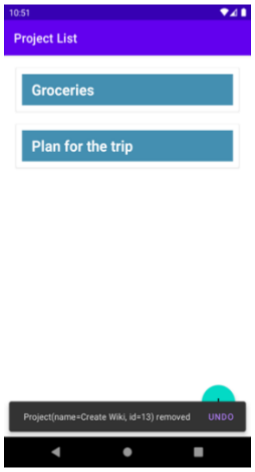


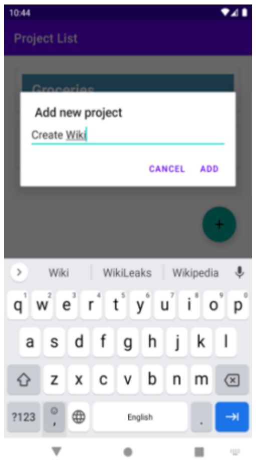
Preparation

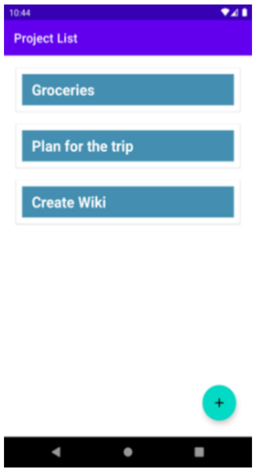
1. Sync the Lab GitHub repo and copy the **Lab 11-Cloud Firestore** folder into your repository.
2. Open the TodoList project in Android Studio. You will see some compilation errors or warning messages. You will correct this in the next sections.

PART A: Implementing the Todo App

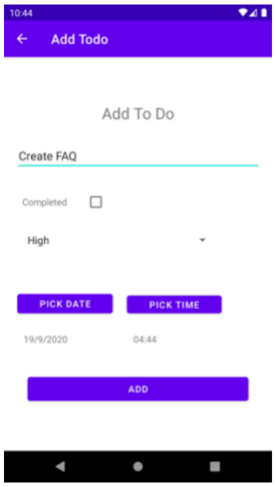
In this lab you will re-implement the Todo app you created in Lab 10 and replace the local database with Firestore. The functionality of the application will still be the same including get, add, update delete projects and to-dos.

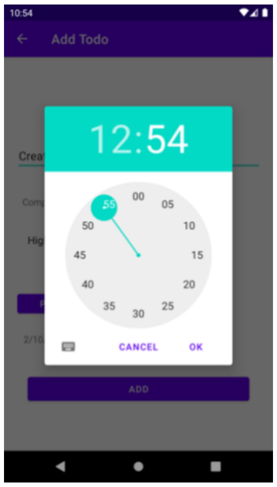


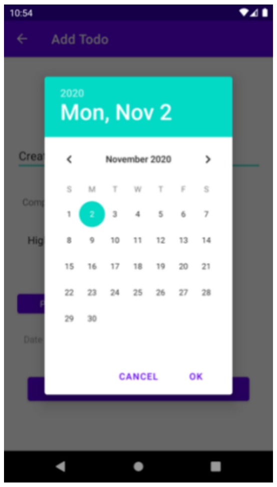


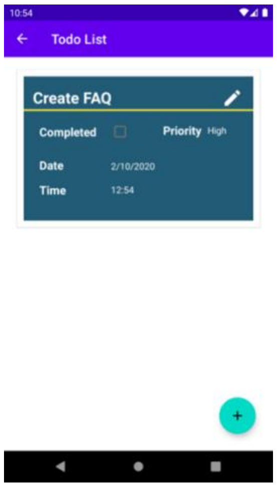


10:53
Todo List





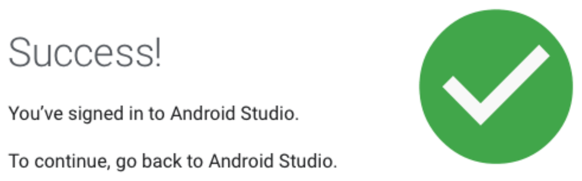




Task 1: Creating Firebase Project

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

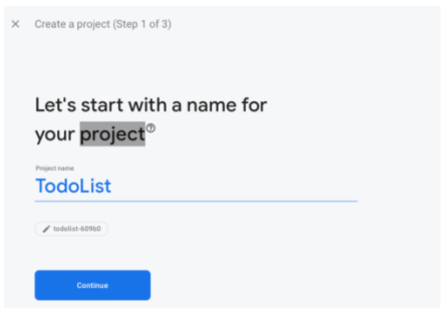

Success!

You've signed in to Android Studio.

To continue, go back to 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.

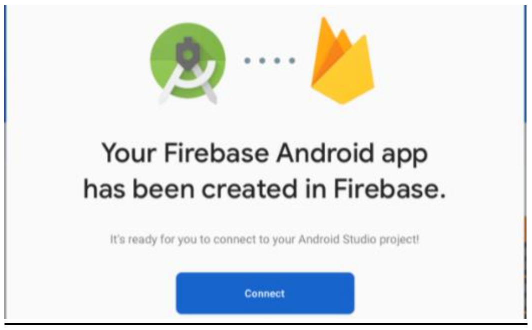
   Choose a Firestore product to add to your app. Then click the button to add a desired Firebase product (for example, Add Firestore to your app).
   As shown in the screen capture below, enter a name for your Firestore project (for example, "TodoList App"), and click Continue.


Create a project (Step 1 of 3)

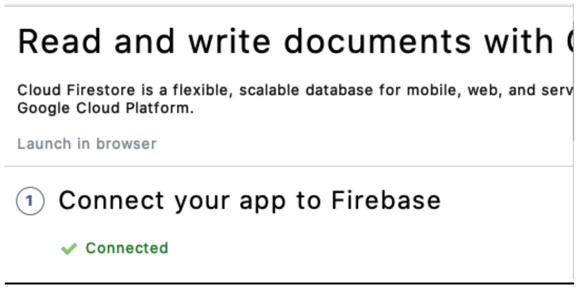
Let's start with a name for your project

Project name: TodoList

todolist-609b0


Your Firebase Android app has been created in Firebase.

It's ready for you to connect to your Android Studio project!


Read and write documents with

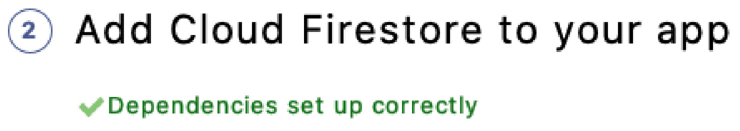
Cloud Firestore is a flexible, scalable database for mobile, web, and serv
Google Cloud Platform.

Launch in browser

1 Connect your app to Firebase

✓ Connected

3. Sync app to download all dependencies.


2 Add Cloud Firestore to your app

✓ Dependencies set up correctly

4. Replace the auto added dependencies in app build.graidle with by the following:

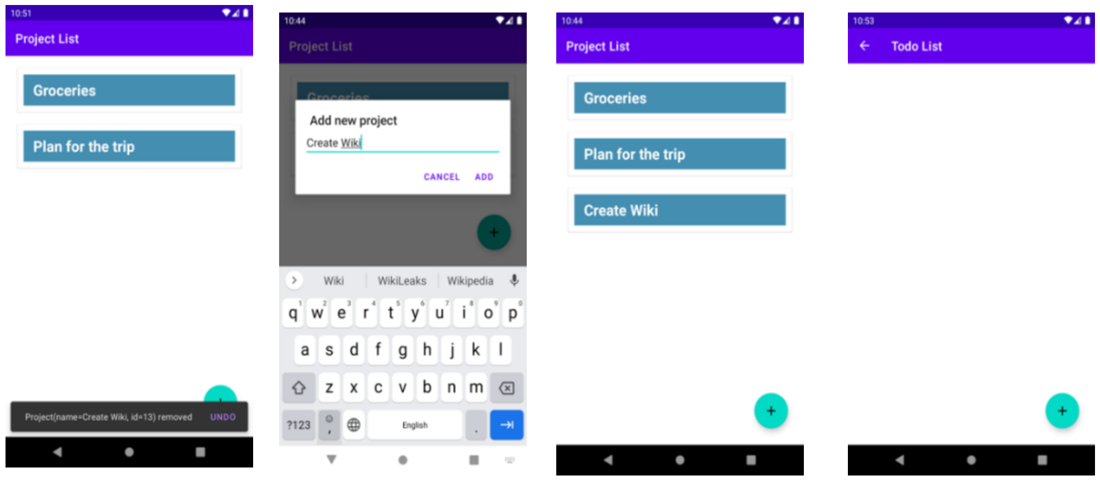
```
implementation platform('com.google.firebase:firebase-bom:26.0.0')
implementation 'com.google.firebase:firebase-firestore-ktx'
implementation 'com.google.firebase:firebase-auth-ktx'
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-play-services'

// 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 implement add/update/delete/query todo list data using **Firestore**.
Note : It is possible to enter data manually in the Firebase console.



Project(name=Create Wiki, id=13) removed

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

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

2. Create two more instance variables inside the TodoListRepo to hold references to project and todo collections.

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

3. 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 changes of query results with the **addSnapshotListener()** method. First, you will get immediately a snapshot having the current document(s) returned by the query.
Then, each time the content change, the app gets notify and it will receive updated documents snapshot. In this part we will replace the manual update of the getting the project and todos with a dynamic update.

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")
        }
      }
    }
```