**­­CMPS 312 Mobile App Development**

**Lab 12 – Cloud FireStore & Background Work with WorkManager**

**Objective**

In this Lab you will practice how to:

* Use Firebase Authentication and security rules to secure Firestore data and application
* Create Cloud Storage bucket to upload and download files
* Create Background Work with WorkManager and support both asynchronous one-off and periodic tasks
* Create notifications for your app

Graphical user interface, text, application

Description automatically generated**A screenshot of a video game

Description automatically generated**

**Preparation**

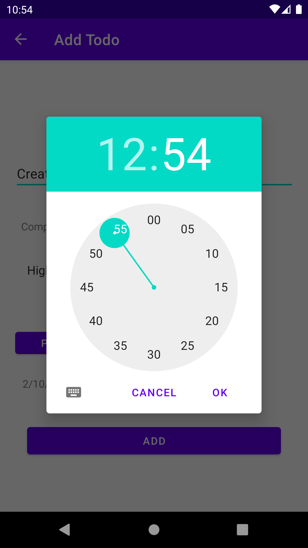
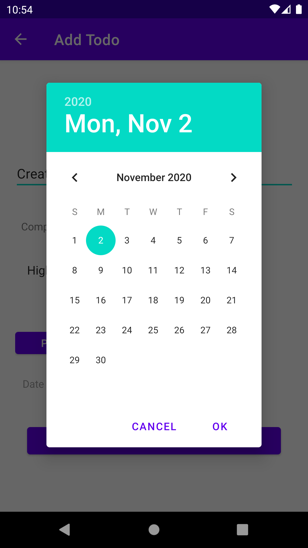
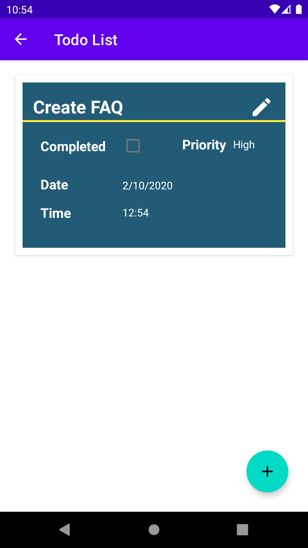
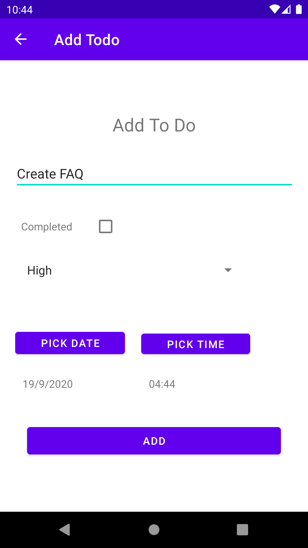
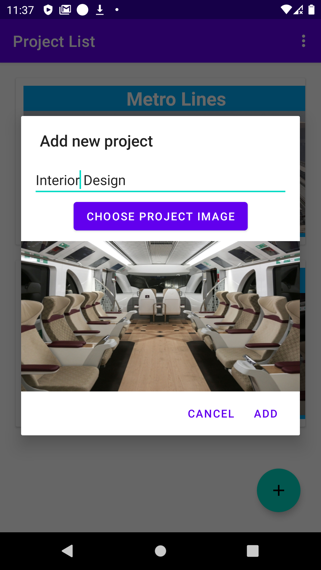
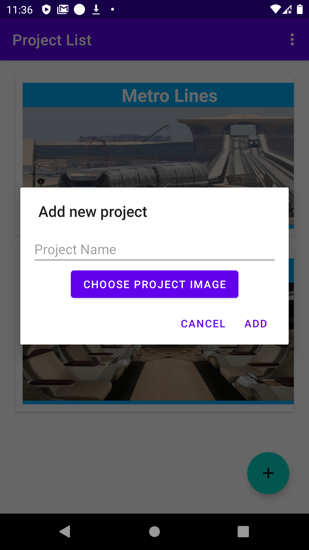
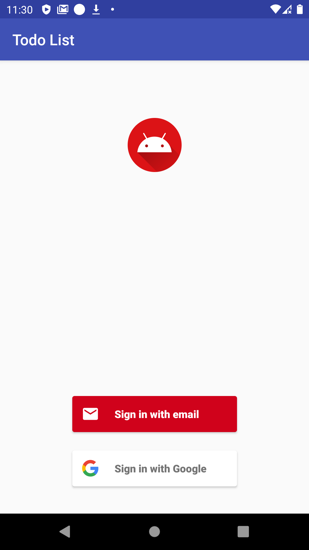
1. Sync the Lab GitHub repo and copy the **Lab 12-****Cloud Firestore and Background Processing** folder into your repository.
2. Import the **TodoList** project into Android Studio. You will probably see some compilation errors or maybe a warning messages. We'll correct this in the next sections.



**PART A: Firebase Authentication**

**­­­**

In this lab you will complete the Todo app implementation you created in Lab 11 and add the authentication part. Also, you will add a way to upload and download files from firebase storage. The rest of the functionality of the application will still be the same get, add, update delete projects and to-dos.



**Task 1: Configure Firebase Auth to use Email and Google Auth**

The [Firebase Assistant](https://firebase.google.com/docs/android/learn-more#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

Graphical user interface, text, application

Description automatically generated

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

What does this workflow do?

* 1. 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.

Graphical user interface, text, application, chat or text message

Description automatically generatedGraphical user interface, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

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

A picture containing text

Description automatically generated

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

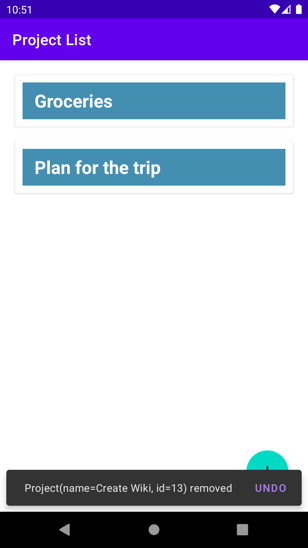
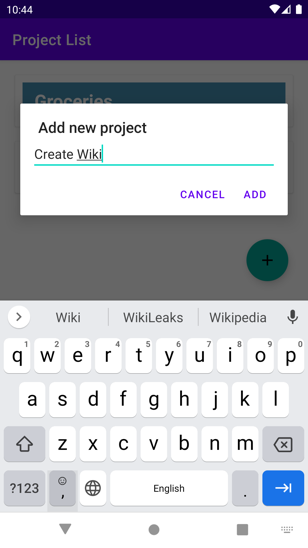
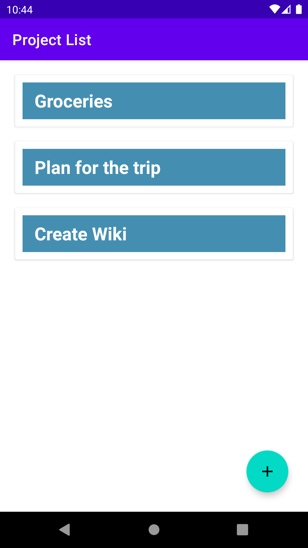
implementation platform('com.google.firebase:firebase-bom:26.0.0')  
  
implementation 'com.google.firebase:firebase-firestore-ktx'  
implementation 'com.google.firebase:firebase-auth-ktx'  
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-play-services'  
  
*// FirebaseUI (for authentication)*implementation '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](https://console.firebase.google.com),



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

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

1. 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") **}**

1. Enable offline caching for the database

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

1. 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")  
 }  
 **}**}