Advanced Mobile Application Development

Recipe App: Firebase Integration

During this demo, we'll be integrating Google Firebase to persist our users' Favorite recipes to a cloud database. We'll pretty much destroy our local data persistence, but that's not to say that cloud is better than local or vice versa. Typically, you'll see a hybrid of the two, but it will be easiest for us just to start from scratch.

I'll be working from the 08-firebase-START branch throughout the demo so be sure to check that out.

Changes

- Annoy bug fix on the loading indicator. Required a major rework of the logic for showing/hiding the indicator in the RecipeDetailFragment. Check my git logs or reach out if you want to learn more!

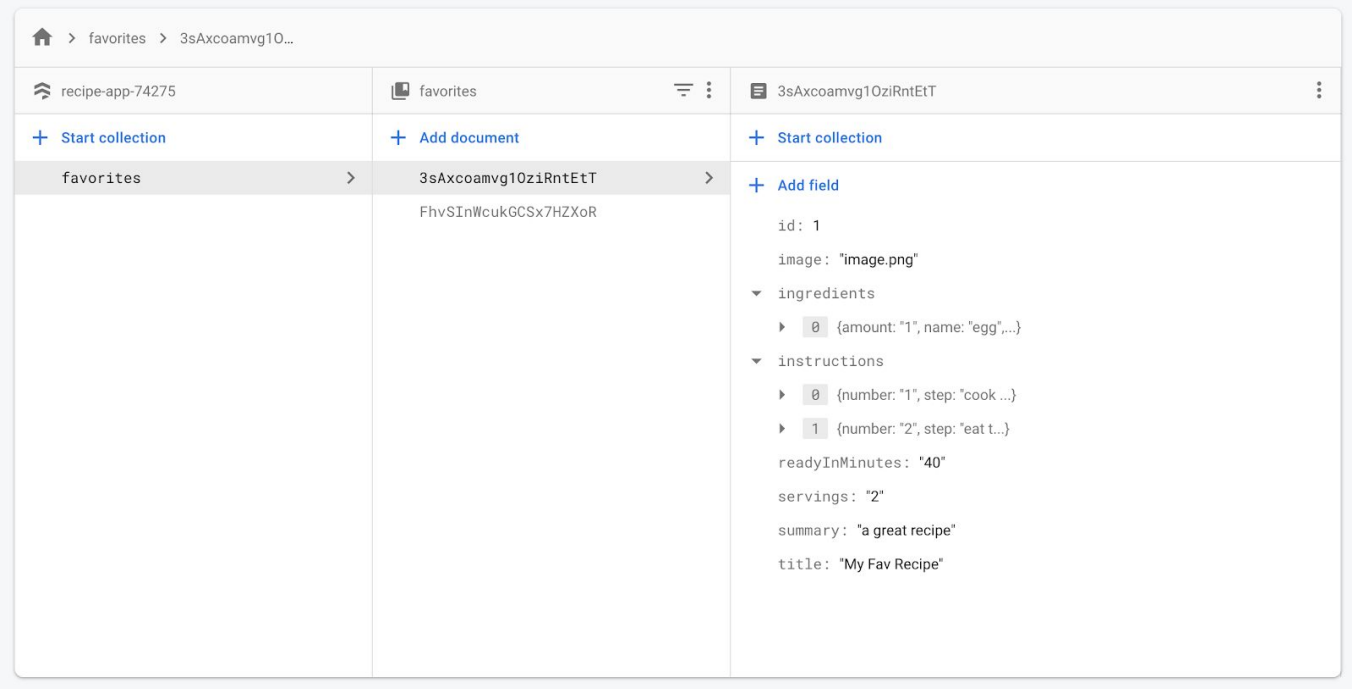
Setup Firebase

First, we need to set up our Firebase project and then add it to the project.

Go to the Firebase console and take the following steps:

- **Create Firebase project** (you could also reuse an old project and clear out the database if you want)
    - Go to Firebase Console and select Add Project
    - Give it a name (recipe-app)
    - Select Next
    - We won't need analytics for this demo so leave that unchecked
    - Select Create Project
    - Wait for it to load
- **Create Database**
    - Go to Develop->Database
    - Select create database under Cloud Firestore banner
    - Start in test mode
    - Click next
    - Select nam5(us-central) for location
    - Schema - match RecipeDetails structure sort of - many different ways - you may be able to come up with a better design than I could last night
        - Start collection
        - Collection id "favorites"
        - Click next

- Create first document
    - Use recipe id for document id eventually, but use “1” for now
    - Id = string
    - title = string
    - summary = string
    - image = string
    - readyInMinutes = string
    - servings = string
    - Instructions = array of maps
        - map
            - Number = string
            - Step = string
    - Ingredients = array of maps
        - map
            - originalString = string
            - Amount = number
            - Unit = string
            - Name = string
- DB should looks something like this



- **Add Firebase to project**
    - Go to Project Overview
    - Select Add Firebase to your Android App
    - Enter package name (com.isaac.recipes)
    - Nickname

- SHA-1
    - Open up command line and enter the following:

`keytool -list -v -alias androiddebugkey -keystore ~/.android/debug.keystore`

    - If it asks for a password, the default it android
- Download config file - make sure you add filename to .gitignore!
    - Drag into project like it says
- Open project build.gradle and add this line to dependencies:
    - classpath **'com.google.gms:google-services:4.3.3'**
- Add this line to top of app module build.gradle:
    - apply **plugin**: **'com.google.gms.google-services'**
- Add this line to the dependencies
    - implementation **'com.google.firebase:firebase-firestore-ktx:21.4.2'**
- Sync gradle

Get Favorites

We're going to pretty much gut the FavoritesRepository to use the cloud database. In a real world application there is typically a mix of local and cloud data persistence to some extent. Often, there will be a backup local database that is used when there is no network connection that gets synced to the server when the connection becomes available. This type of data persistence is somewhat outside the scope of this class, so we'll just switch between the two and you'll have experience with both if you need to implement something more complicated in the future.

Start by removing all of the code inside our FavoritesRepository class :(

We'll need properties to store our data and the reference to our database so **add those to the top of the empty class**:

```
private val db = Firebase.firestore

val favoriteList = MutableLiveData<List<Recipe>>()
val favoriteDetails = MutableLiveData<RecipeDetails>()
private val allData: MutableMap<Int, RecipeDetails> = mutableMapOf()

val recipeIsFavorite = MutableLiveData<Boolean>()
```

We'll listen to our favorites collection in the init block and use a couple private methods to parse all the data and store it appropriately in our repository:

```
init {
  db.collection("favorites")
     .addSnapshotListener { snapshot, e ->
        if(e != null) {
           Log.e(LOG_TAG, "Listen failed.", e)
           return@addSnapshotListener
```

```
        }
        if (snapshot != null) {
            parseAllData(snapshot)
        } else {
            Log.d(LOG_TAG, "Current data: null")
        }

    }

}

private fun parseAllData(result: QuerySnapshot) {
    val allFavorites = mutableListOf<Recipe>()
    for(doc in result) {
        //get the data from the document
        val id: Int = doc.id.toInt()
        val title: String = doc.getString("title")!!
        val summary: String = doc.getString("summary")!!
        val image: String = doc.getString("image")!!
        val readyInMinutes: Int = (doc.get("readyInMinutes") as String).toInt()
        val servings: Int = doc.getString("servings")!!.toInt()
        val genericInstructions: ArrayList<*> = doc.get("instructions") as ArrayList<*>
        val genericIngredients : ArrayList<*> = doc.get("ingredients") as ArrayList<*>

        //add to all favorites array
        allFavorites.add(Recipe(
            id,
            title,
            readyInMinutes,
            servings,
            image
        ))

        allData[id] = RecipeDetails(
            id,
            title,
            summary,
            image,
            readyInMinutes,
            servings,
            listOf(Steps(getRecipeInstructions(genericInstructions))),
            getRecipeIngredients(genericIngredients)
        )
    }

    Log.i(LOG_TAG, "allData: $allData")

    favoriteList.value = allFavorites
}
```

```
private fun getRecipeInstructions(dbArray: ArrayList<*>): List<Instruction> {
  val instructions = mutableListOf<Instruction>()
  for(obj in dbArray) {
    val map = obj as HashMap<*, *>
    instructions.add(
      Instruction(
        (map["number"] as String).toInt(),
        map["step"] as String
      ))
  }

  return instructions
}

private fun getRecipeIngredients(dbArray: ArrayList<*>): Set<Ingredient> {
  val ingredients = mutableSetOf<Ingredient>()
  for(obj in dbArray) {
    val map = obj as HashMap<*, *>
    ingredients.add(Ingredient(map["originalString"] as String,
      map["name"] as String,
      (map["amount"] as String).toDouble(),
      map["unit"] as String
    ))
  }
  return ingredients
}
```

Now, checking to see if a recipe is a favorite is very simple, so add the method for that:

```
fun isRecipeFavorited(id: Int) {
  recipeIsFavorite.value = allData.containsKey(id)
}
```

Same with getting details for a recipe, so add a method for that:

```
fun isRecipeFavorited(id: Int) {
  recipeIsFavorite.value = allData.containsKey(id)
}
```

Add some stubs for the remove and add methods since we have yet to implement those but we reference from the *SharedFavoritesViewModel*:

```
fun removeRecipeFromFavorites(id: Int) {}
fun addFavorite(recipe: RecipeDetails) {}
```

Change SharedFavoritesViewModel

Go to the *SharedFavoritesViewModel* and **remove** the *init()* and *onCleared()* methods and the favoriteListObserver since we no longer need to transform Room types to Model types.

Change the *favoriteRecipeList* LiveData object to reference the LiveData of Recipes found in the Repository class:

```
val favoriteRecipeList: MutableLiveData<List<Recipe>> = favRepo.favoriteList
```

Finally, we can **run the app** and we should see our dummy Favorite in the list. Should be able to still select it and view the fake details.

Add Favorites

Next, we'll implement the *addFavorites()* method in the *FavoritesRepository* and **include relevant helper functions**:

```
fun addFavorite(recipe: RecipeDetails) {
  val recipeMap = recipeDetailsToHashMap(recipe)

db.collection("favorites").document(recipe.id.toString())
      .set(recipeMap)
      .addOnSuccessListener {
         Log.i(LOG_TAG, "Added favorite success!")
      }
      .addOnFailureListener { exception ->
         Log.w(LOG_TAG, "Error adding document.", exception)
      }
}

private fun recipeDetailsToHashMap(recipe: RecipeDetails): HashMap<String, *> {
  val map = hashMapOf(
      "id" to recipe.id,
      "title" to recipe.title,
      "summary" to recipe.summary,
      "image" to recipe.image,
      "readyInMinutes" to recipe.readyInMinutes.toString(),
      "servings" to recipe.servings.toString(),
      "ingredients" to ingredientsToArrayOfMaps(recipe.extendedIngredients),
      "instructions" to instructionsToArrayOfMaps(recipe.analyzedInstructions[0].steps)
  )
  return map
}

private fun ingredientsToArrayOfMaps(ingredients: Set<Ingredient>): ArrayList<HashMap<String, *>> {
  val ingredientArrayList = ArrayList<HashMap<String, *>>()

  for(ingredient in ingredients) {
     ingredientArrayList.add(hashMapOf(
        "originalString" to ingredient.originalString,
        "name" to ingredient.name,
        "amount" to ingredient.amount.toString(),
        "unit" to ingredient.unit
     ))
  }
```

```
    return ingredientArrayList
}

private fun instructionsToArrayOfMaps(instructions: List<Instruction>): ArrayList<HashMap<String, *>> {
    val instructionArrayList = ArrayList<HashMap<String, *>>()

    for(instruction in instructions) {
        instructionArrayList.add(hashMapOf(
            "number" to instruction.number.toString(),
            "step" to instruction.step
        ))
    }

    return instructionArrayList
}
```

Run the app and verify that you can now add new favorites.

Remove Favorites

Removing favorites is wicked easy:

```
fun removeRecipeFromFavorites(id: Int) {
    db.collection("favorites").document(id.toString()).delete()
}
```

Run it to verify

And that's all for today folks! We'll take a look at authentication next week for out last topic of the semester.