Firebase Storage

Firestore vs. Storage

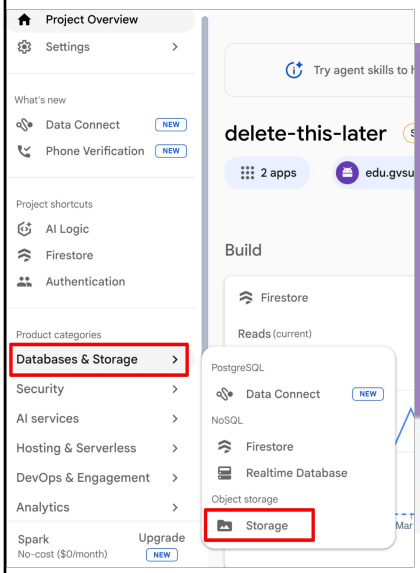
- Documents (identified by unique DocID)
- Collections (a placeholder of child documents)
- Fields in a document are limited to
  - String
  - Number (integer or floating-point)
  - Timestamp (microsecond accuracy)
  - Boolean
  - Map/Dictionary
  - GeoPoint
  - **Blob (binary up to 1 MB)**

- Remote filesystems similar to those on Linux, OSX, Windows
- Data are organized into files and folder
- Any file type
- No size limit (if you a pay customer)
- Limits for free-tier users
  - 20000 writes/day
  - 50000 reads/day
  - Download: 500 GB/day
  - Upload: 500 operations/day
- **Each file is associated with a unique URL**

Library Dependencies Setup

```
// In the module build.gradle.kts
dependencies {
    implementation("com.google.firebase:firebase-bom:34.11.0")
    implementation("com.google.firebase:firebase-ai")
    implementation(“com.google.firebase:firebase-common-ktx”)
}
```

Enable Firebase Storage



Initialize Storage Reference

```
class AppViewModel : ViewModel() {
   val appStorage = Firebase.storage
   val rootRef: StorageReference = appStorage.root
   // val rootRef: StorageReference = appStorage.reference
   fun yourUploadToStorageOperationHere() {
       val fileRef = appStorage.root.child("name/your/file/path/here")
       viewModelScope.launch {
           val upload = fileRef.putXXX(____).await()
       }
   }

   fun yourDownloadToStorageOperationHere() {
       val fileRef = appStorage.root.child("name/your/file/path/here")
       viewModelScope.launch {
           val upload = fileRef.gettXXX(____).await()
       }
   }
}
```

Summary of Storage Operations

Upload	Download
.putBytes(your_byte_array).await()	.getBytes(number_of_max_bytes).await()
.putFile(your_file_object).await()	.getFile(file_object).await()
.putStream(your_stream_object).await()	.getStream(stream_object).await()

Upload to Storage

Upload from Bytes

```
fun uploadFromBytes() {        // A function inside your ViewModel
    val fileRef = appStorage.root.child("name/your/file/path/here")
    val myData = ByteArray(40_000)  // Allocate as needed
    // populate myData with actual payload
    viewModelScope.launch(Dispatchers.IO) {
        val upload = fileRef.putBytes(myData).await()
        if (upload.task.isSuccessful) {
            val remoteUrl = fileRef.downloadUrl.await()
            println("Storage url is $remoteUrl)
        }
    }
}
```

Typically used with other utility that generates in-memory contents

Upload From Local File (Uri)

```
fun uploadFromFile(fileName: String) {   // A function inside your
ViewModel
    val fileRef = appStorage.root.child("name/your/file/path/$fileName")
    val myLocalFile = File("/path/to/LOCAL/file/$fileName")
    val fileUri = Uri.fromFile(myLocalFile)
    viewModelScope.launch(Dispatchers.IO) {
        val upload = fileRef.putFile(fileUri).await()
        if (upload.task.isSuccessful) {
            val remoteUrl = fileRef.downloadUrl.await()
            println("Storage url is $remoteUrl)
        }
    }
}
```

Upload From (Input) Stream

```
fun uploadFromStream() {       // A function inside your ViewModel
    val fileRef = appStorage.root.child("name/your/file/path/here")
    val myStream = FileInputStream(File("/local/file/path"))
    viewModelScope.launch(Dispatchers.IO) {
        val upload = fileRef.putStream(myStream).await()
        if (upload.task.isSuccessful) {
            val remoteUrl = fileRef.downloadUrl.await()
            println("Storage url is $remoteUrl)
        }
    }
}
```

Upload + Metadata

This may be necessary for correct interpretation of contents by browsers

```
fun uploadOperation() {
    val metaData = storageMetadata {
        contentType = "application/pdf"        // Any standard MIME type
        contentLanguage = "en-US"
    }
    viewmodelScope.launch(Dispatchers.IO) {
        appStorage.root._____.putFile(_____, metaData).await()
    }
}
```

Download From Storage

Download to ByteArray

```
fun downloadBytes() {          // A function inside your ViewModel
    val fileRef = appStorage.root.child("name/your/file/path/here")
    viewModelScope.launch(Dispatchers.IO) {
        try {
            val remoteData: ByteArray = fileRef.getBytes(40_000).await()
            // val remoteData: ByteArray = fileRef.getBytes(Long.MAX_VALUE).await()
        } catch(e: Exception) {
            println("Unable to download ${e.message}")
        }
    }
}
```

Download to File

```
fun downloadBytes() {          // A function inside your ViewModel
    val fileRef = appStorage.root.child("name/your/file/path/here")
    viewModelScope.launch(Dispatchers.IO) {
        val destinationFile = File("/your/local/path/here")
        try {
            val task = fileRef.getFile(destinationFile).await()
            println("Bytes transferred: ${task.bytesTransferred}")
            println("Total bytes in file: ${task.totalByteCount}")
        } catch(e: Exception) {
            println("Unable to download ${e.message}")
        }
    }
}
```

Download As Stream

Typical use case: apply processing logic as bytes are downloaded without having the file saved locally first.

```
fun downloadIntoStream() {          // A function inside your ViewModel
    val fileRef = appStorage.root.child("name/your/file/path/here")
    viewModelScope.launch(Dispatchers.IO) {
        fileRef.getStream { snapshot, stream ->
           println("${snapshot.bytesTransferred} of ${snapshot.totalByteCount}")
           val partialData = stream.read(____)
        }
    }
}
```