Firebase Firestore

Firebase

A collection of many products

- **Cloud Firestore** (beta since 2017, GA since 2019)
- Authentication
- Cloud Storage
- Realtime DB (beta since 2012, GA since 2014?)
- ML Kit
- Cloud Functions

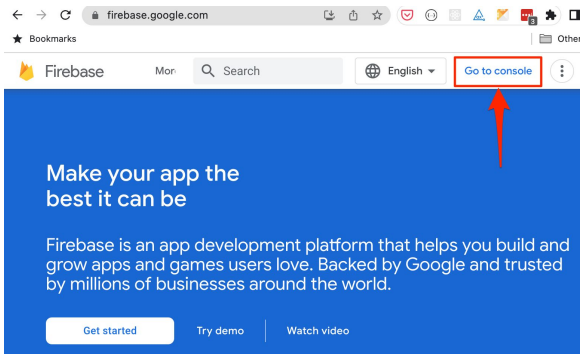
Using Firebase Products in iOS

Prerequisites: Create a Firebase Project

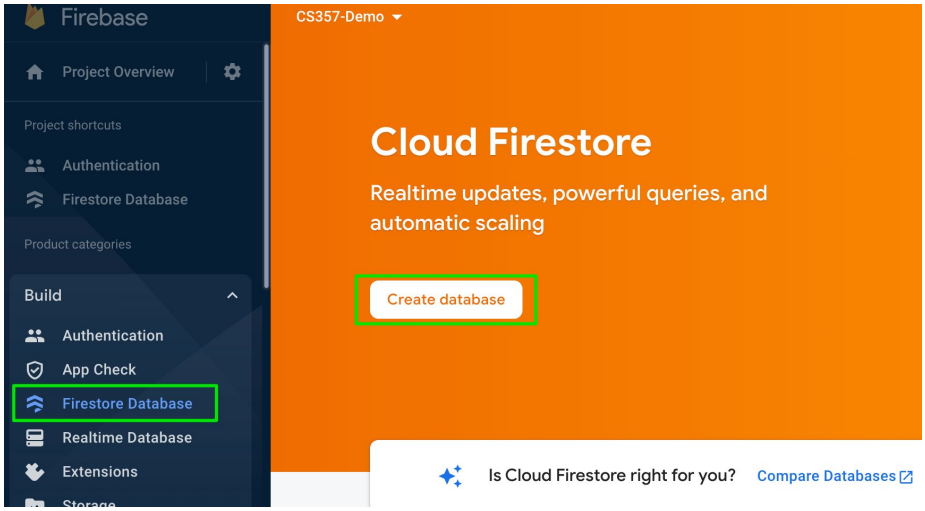
Quick Video Introduction

Create A Firebase Project

- Login to https://firebase.google.com
  - Use your **personal** GMail account
  - Your GVSU GMail (xxxxxx@mail.gvsu.edu) does not have access to Google Developers features
- Go to Firebase Console
- Create a new project
  - Enter project name
  - Disable (or enable) Google Analytics

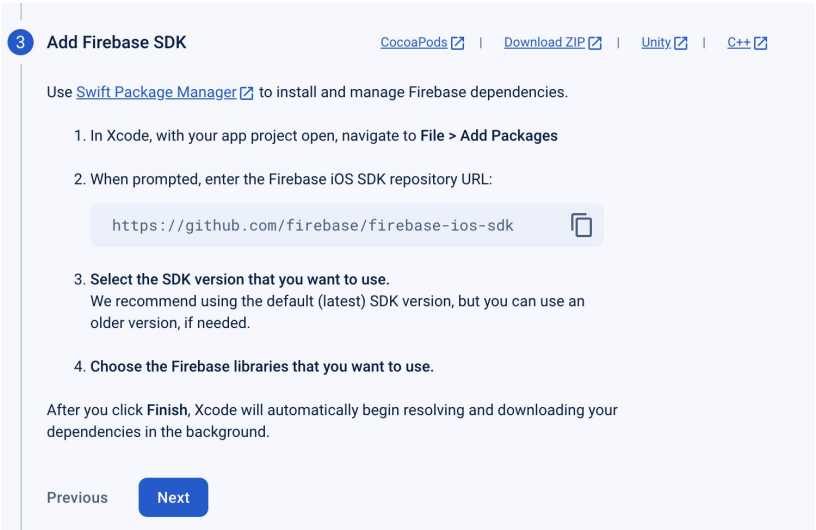


Then .... initialize Firestore



Library Installation
XCode Swift Package Manager

Instructions from Firebase Console



Firebase Firestore

- Cloud Database
- No SQL no DB Schema required to create tables
- Data are organized into Collections and Documents

SQL	Cloud Firestore
Tables	Collections
Rows	Documents
Primary Key	Document ID
Fields	key-value pairs

Typical Setup in a ViewModel

```
import FirebaseFirestore
class MyViewModel: ObservableObject {
  let db = Firebase.firestore()

  func someAsyncFunction() async {
    let myColl = db.collection("xyz")
    let myDoc = db.collection("xyz").doc("def")
    // Firestore work here
  }

  func someNonAsyncFunc() {
    let myColl = db.collection("xyz")
    let myDoc = db.collection("xyz").doc("def")
    Task {
       // Firestore work here
    }
  }
}
```

Option 1: Use async functions

Option 2: Use non-async functions, but perform Firebase work inside a task (similar to Kotlin coroutine)

Wiring Up View with ViewModel

```
import FirebaseFirestore
class MyViewModel: ObservableObject {
  let db = Firebase.firestore()

  func yourFirebaseWork() async {
    // Firestore work here
  }
}
```

```
import SwiftUI

struct MyView: View {
  @ObservedObject var vm: MyViewModel

  var body: some View {
     VStack {
        Button("Go") {
           Task {
              vm.yourFirebaseWork()
           }
        }
     }
  }
}
```


Summary of CRUD Operations

```
let db = Firestore.firestore()
```

	Collection	Document

Reference	`let collRef = db.collection("collName")`	`let docRef = db.collection("collName")`
		`.document("xx")`
		`// docRef = db.document("collName/xx")`

Create	**No API provided by Google**	`docRef.setData(from:)` `collRef.addDocument(from:)`
Read	`collRef.getDocuments()`	`docRef.getDocument(as:)`
Update	N/A	`docRef.setData(from:)`
Delete	**No API provided by Google**	`docRef.delete()`

All these functions are either asynchronous or take a *completion* closure

Document Representation as Swift Struct

```
import FirebaseFirestore

struct CourseType: Codable {
    @DocumentID var id: String?
    var name:String
    var credits: Int
    var isRequired: Bool
}
```

- Document ID fetched from Firestore is mapped to the struct member annotated with @DocumentID
  - In Objective-C, "`id`" is reserved keyword for structures

CRUD: Read All Documents in a Collection

```
func getAllCourses() async {
  let myColl = db.collection("courses")
  do {
    let qs = try await myColl.getDocuments()
    for courseDoc in qs.documents {
      let crs = try courseDoc.data(as: CourseType.self)
    }
  } catch {
    print("Failed to read data() \(error)")
  }
}
```

```
SELECT * FROM courses
```

```
class MyViewMode: ObservedObject {
  @Published var courses: Array<CourseType> = []

  func getAllCourses() async {
    let myColl = db.collection("courses")
    do {
      let qs = try await myColl.getDocuments()
      for courseDoc in qs.documents {
        let crs = try courseDoc.data(as: CourseType.self)
        await MainActor.run {
          courses.append(crs)
        }
      }
    } catch {
        print("Failed to read data() \(error)")
    }
  }
}
```

Invoke the Async Function from View

```
import SwiftUI

struct MyView: View {
    @ObservedObject var vm: MyViewModel
    var body: some View {
        // Your UI here
        Button("Go") {
            Task {
                await vm.aFunctionInYourViewModel()
            }
        }
    }
}
```


Invoke the Function when View (dis)appear

```
import SwiftUI

struct MyView: View {
    @ObservedObject var vm: MyViewModel
    var body: some View {
        VStack {
            // Your UI definition goes here
        }
       .onAppear {
           Task {
               await vm.aFunctionInYourViewModel()
           }
        }
        .onDisappear {
          // Your code here
        }
    }
}
```

All the code snippets on the next few slides are shall be invoked from your ViewModel class

CRUD: Create Documents

```
INSERT INTO courses(name, credits, isrequired) VALUES(“CS162”, 4, TRUE)
```

```
let newCourse = CourseType(name: “CS162”, credits: 4, isRequired: true)

let myColl = db.collection(“courses”)
do {
    try await myColl.addDocument(from:newCourse)
} catch {
    print("Failed to add document \(error)")
}
```

CRUD: Read A Specific Document (known DocID)

```
let myDoc = db.document("courses/JFHGDFS3656")
do {
    let crs = try await myDoc.getDocument(as: CourseType.self)
    // crs is a CourseType var
    print("Password is \(crs.name) \(crs.id)")
}
catch {
    print("Failed to fetch document \(error)")
}
```


CRUD: Update a Document (known DocID)

```
UPDATE courses SET isrequired = false WHERE id = "MTH100"
```

```
let myDoc = db.document("courses/MTH100")
// create the updated content
let newMath100 = CourseType(name: "____",
                            credits:5,
                            isRequired:false)
do {
    // change the content in Firestore
    try await myDoc.setData(from: newMath100)
} catch {
     print("Failed to update document \(error)")
}
```

CRUD: Delete a Document (known DocID)

```
DELETE FROM courses WHERE id = “MTH100”
```

```
let myDoc = db.document(“courses/MTH100”)
do {
   try await myDoc.delete()
   print(“MTH is now gone”)
} catch {
   print(“Failed to delete document \(error)”)
}
```


CRUD: Query Documents in a Collection

```
SELECT * FROM courses WHERE credits > 3
```

```
let myColl = db.collection(“courses”)
               .whereField(“credits”, isGreaterThan: 3)
do {
    let qs = try await myColl.getDocuments()
    for courseDoc in qs!.documents {
        let crs = try courseDoc.data(as: CourseType.self)
        print("Course \(crs.credits) [\(crs.isRequired)]")
    }
} catch {
    print("Failed to read data() \(error)")
}
```

Other whereField(_:_)

Operator	Example

`<, <=, ==, >=, >`	`whereField(“credits”, isEqualTo: 3)`
	`whereField(“credits”, isNotEqualTo: 3)`
	`whereField(“credits”, isLessThan: 3)`
	`whereField(“credits”, isLessThanOrEqualTo: 3)`
	`whereField(“credits”, isGreaterThan: 3)`
	`whereField(“credits”, isGreaterThanOrEqualTo: 3)`

Operator	Example (`prereqs` must be an ARRAY in the course document)
`array-contains`	`// Is MTH200 a prereq?` `whereField(“prereqs”, arrayContains: “MTH200”)`
`array-contains-any`	`// Is either MTH200 or STA215 a prereq?` `whereField(“prereqs”, arrayContainsAny: [“MTH200”, “STA215”])`

Live Demo