

SwiftUI + Firebase: The Complete Step-by-Step Guide to Building Mobile Apps

5 min read · Dec 9, 2024



octavia

Follow



Listen



Share

SwiftUI provides an elegant way to build modern, reactive UIs for Apple platforms, while Firebase is a powerful backend platform for developing mobile and web applications. Combining these tools enables developers to create apps with authentication, real-time databases, cloud functions, and more. This article walks you through integrating Firebase into a SwiftUI project.



Why Use Firebase with SwiftUI?

Firebase simplifies backend management with features like:

- **Authentication:** Supports multiple providers (Google, Apple, Email/Password).
- **Firestore:** A scalable, real-time database.
- **Cloud Storage:** Store and retrieve user files like images.
- **Analytics:** Monitor app usage and performance.
- **Crashlytics:** Track and fix issues in production.

Setting Up Your Firebase Project

Before diving into SwiftUI, configure Firebase:

Step 1: Create a Firebase Project

1. Go to the Firebase Console (<https://console.firebase.google.com/>).
2. Click **Add Project** and follow the setup wizard.
3. Register your app:
 - Select **iOS** as the platform.
 - Enter your app's **Bundle Identifier** (found in Xcode under **General > Identity**).

Step 2: Download GoogleService-Info.plist

1. After registering your app, download the configuration file **GoogleService-Info.plist**.
2. Drag and drop this file into your Xcode project, ensuring it's added to all targets.

Adding Firebase to Your SwiftUI Project

Firebase can be integrated using Swift Package Manager:

Step 1: Add Firebase via SPM

1. In Xcode, go to **File > Add Packages**.
2. Paste the Firebase SPM URL:
<https://github.com/firebase/firebase-ios-sdk>
3. Select the Firebase modules you need:
 - For authentication: **FirebaseAuth**.
 - For database: **Firestore**.

Initialize Firebase in Your App

Modify your **App** file to initialize Firebase:

```
import SwiftUI
import SwiftData
import Firebase
```

```
@main
struct YourAppName: App {

    init() {
        FirebaseApp.configure()

        #if DEBUG
        let providerFactory = AppCheckDebugProviderFactory()
        AppCheck.setAppCheckProviderFactory(providerFactory)
        #endif
    }

    var body: some Scene {
        WindowGroup {
            ContentView()
        }
    }
}
```

Adding Firebase Authentication

Let's implement Firebase Authentication with **Email and Password**:

Step 1: Create a ViewModel for Authentication

```
import SwiftUI
import FirebaseAuth

class AuthViewModel: ObservableObject {
    @Published var user: User? = nil
    @Published var isSignedIn: Bool = false

    init() {
        self.user = Auth.auth().currentUser
        self.isSignedIn = user != nil
    }

    func signUp(email: String, password: String) {
        Auth.auth().createUser(withEmail: email, password: password) { result,
            if let error = error {
                print("Sign Up Error: \(error.localizedDescription)")
                return
            }
            self.user = result?.user
            self.isSignedIn = true
        }
    }
}
```

Medium

🔍 Search



```
        return
    }
    self.user = result?.user
    self.isSignedIn = true
}
}

func signOut() {
    do {
        try Auth.auth().signOut()
        self.user = nil
        self.isSignedIn = false
    } catch {
        print("Sign Out Error: \(error.localizedDescription)")
    }
}
}
```

Step 2: Use ViewModel in SwiftUI Views

Create a simple UI for sign-in and sign-up:

```
import SwiftUI

struct AuthView: View {
    @StateObject private var viewModel = AuthViewModel()

    @State private var email = ""
    @State private var password = ""

    var body: some View {
        VStack {
            if viewModel.isSignedIn {
                FContentView()
            } else {
                TextField("Email", text: $email)
                    .textFieldStyle(RoundedBorderTextFieldStyle())
                    .padding()
                SecureField("Password", text: $password)
                    .textFieldStyle(RoundedBorderTextFieldStyle())
                    .padding()
                HStack {
                    Button("Sign In") {
                        viewModel.signIn(email: email, password: password)
                    }
                }
            }
        }
    }
}
```

```

        Button("Sign Up") {
            viewModel.signUp(email: email, password: password)
        }
    }
}
.padding()
}
}

```

Integrating Firestore for Real-Time Data

Let's add Firestore to store and display user data.

Step 1: Firestore Model

Create a Model for Firestore data model :

```

import FirebaseFirestore

struct Note: Identifiable, Codable {
    @DocumentID var id: String?
    var title: String
    var content: String
}

```

Step 2: Firestore ViewModel

Create a ViewModel for Firestore interactions:

```

import FirebaseFirestore

class FirestoreManager: ObservableObject {
    private var db = Firestore.firestore()
    @Published var notes = [Note]()

    // Create Note
    func addNote(title: String, content: String) {
        let newNote = Note(title: title, content: content)

        do {
            _ = try db.collection("notes").addDocument(from: newNote)
        } catch {
            print("Error adding document: \(error)")
        }
    }
}

```

```

// Read Notes
func getNotes() {
    db.collection("notes").order(by: "title").addSnapshotListener { snapshot, error in
        if let error = error {
            print("Error getting notes: \(error)")
            return
        }

        self.notes = snapshot?.documents.compactMap { document in
            try? document.data(as: Note.self)
        } ?? []
    }
}

// Update Note
func updateNote(note: Note) {
    guard let noteID = note.id else { return }

    do {
        try db.collection("notes").document(noteID).setData(from: note)
    } catch {
        print("Error updating note: \(error)")
    }
}

// Delete Note
func deleteNote(note: Note) {
    guard let noteID = note.id else { return }

    db.collection("notes").document(noteID).delete { error in
        if let error = error {
            print("Error deleting note: \(error)")
        }
    }
}
}

```

Step 3: Firestore View

Create a view to send and display messages:

```

import SwiftUI

struct FContentView: View {
    @StateObject private var firestoreManager = FirestoreManager()
    @State private var showingAddNote = false
    @EnvironmentObject var authViewModel: AuthViewModel
}

```

```

var body: some View {

    VStack{

        NavigationView {
            List {
                ForEach(firestoreManager.notes) { note in
                    HStack {
                        VStack(alignment: .leading) {
                            Text(note.title).font(.headline)
                            Text(note.content).font(.subheadline)
                        }
                        Spacer()
                        Button("Delete") {
                            firestoreManager.deleteNote(note: note)
                        }
                    }
                    .swipeActions {
                        Button("Edit") {
                            // handle editing note here
                            showingAddNote = true
                        }
                    }
                    .tint(.blue)
                }
            }
            .navigationTitle("Notes")
            .navigationBarItems(trailing: Button(action: {
                showingAddNote = true
            }) {
                Image(systemName: "plus")
            })
            .onAppear {
                firestoreManager.getNotes()
            }
            .sheet(isPresented: $showingAddNote) {
                AddNoteView(firestoreManager: firestoreManager)
            }
        }

        Button(action: {
            authViewModel.signOut()
        }) {
            Text("Sign Out")
                .foregroundColor(.white)
                .frame(maxWidth: .infinity)
                .padding()
                .background(Color.red)
                .cornerRadius(10)
        }
    }
}

```

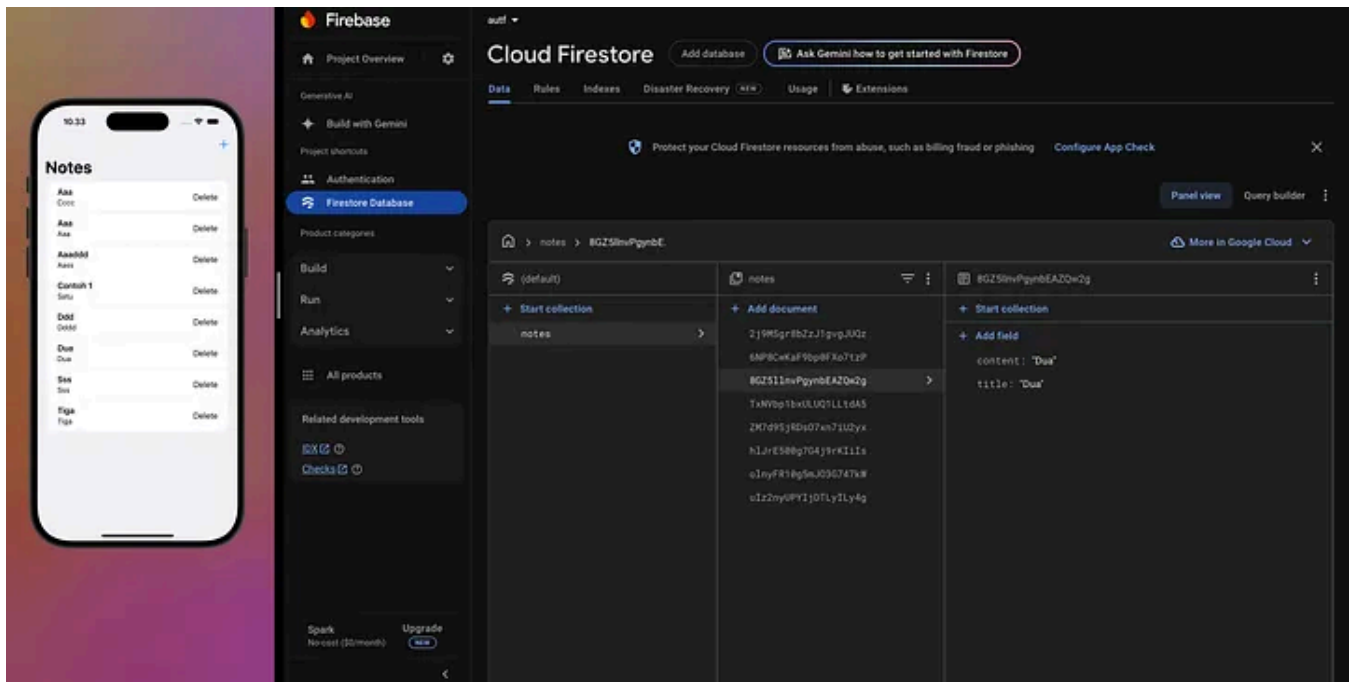
```
}  
}
```

Step 4: Firestore Add Data

Create a view to add data messages:

```
import SwiftUI  
  
struct AddNoteView: View {  
    @Environment(\.presentationMode) var presentationMode  
    @ObservedObject var firestoreManager: FirestoreManager  
    @State private var title = ""  
    @State private var content = ""  
  
    var body: some View {  
        NavigationView {  
            Form {  
                Section(header: Text("Note Details")) {  
                    TextField("Title", text: $title)  
                    TextField("Content", text: $content)  
                }  
  
                Button("Save") {  
                    firestoreManager.addNote(title: title, content: content)  
                    presentationMode.wrappedValue.dismiss()  
                }  
            }  
            .navigationTitle("Add")  
            .navigationBarItems(trailing: Button("Cancel") {  
                presentationMode.wrappedValue.dismiss()  
            })  
        }  
    }  
}
```

Example of Result :



Testing the Application

- Run the app on a **physical device or simulator**.
- Test user authentication and Firestore functionality.

Deploying the App

When ready, deploy your app:

1. Set up **App Check** in Firebase for security.
2. Test thoroughly on physical devices.
3. Submit your app to the App Store.

Conclusion

Integrating Firebase with SwiftUI unlocks powerful backend features with minimal effort. From authentication to real-time databases, Firebase streamlines app development and lets you focus on delivering great user experiences. This guide provides a foundation to explore more Firebase features like Cloud Functions and Analytics.

Get octavia's stories in your inbox

Join Medium for free to get updates from this writer.

Enter your email

Subscribe

Full code: <https://github.com/octavvia/autf> .

Happy coding! 🚀

Swiftui

iOS App Development

iOS

Mobile App Development

Firebase



Follow

Written by octavia

105 followers · 5 following

Mobile dev specializing in Flutter & Swift. Exploring cross-platform with Flutter and native iOS with Swift.

Contact: aoctavia789@gmail.com

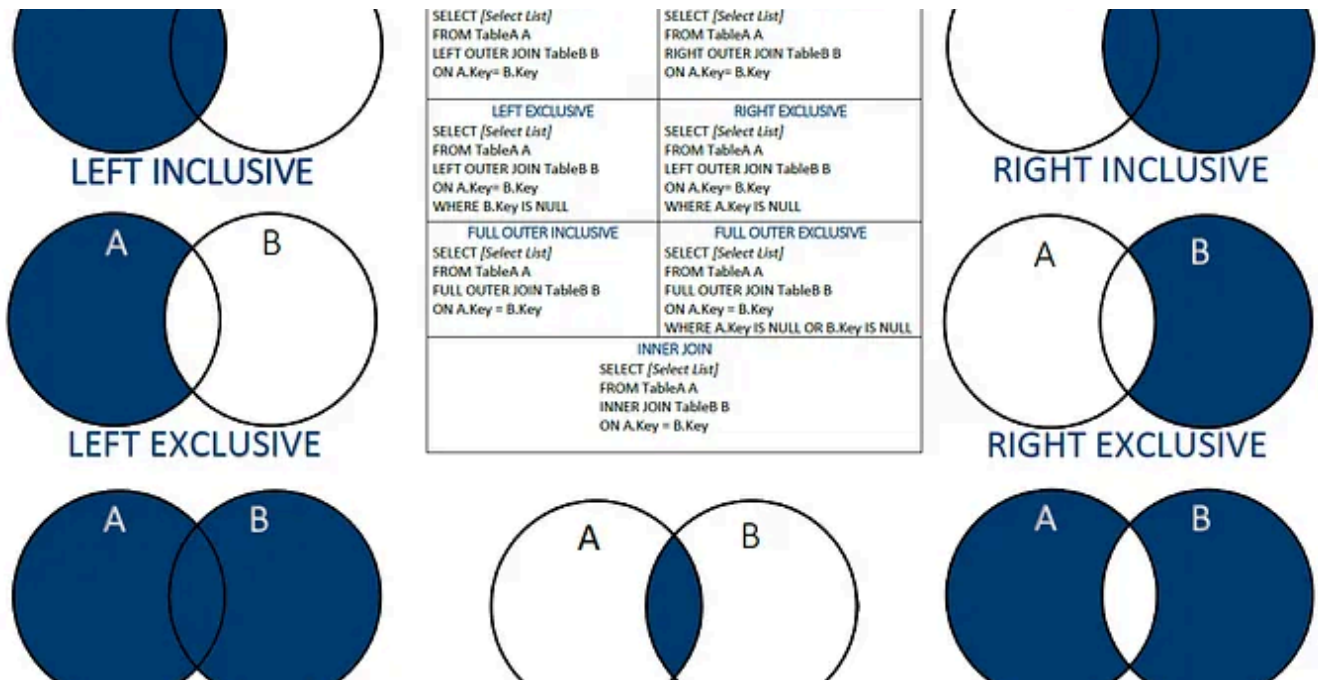
No responses yet



Write a response

What are your thoughts?

More from octavia

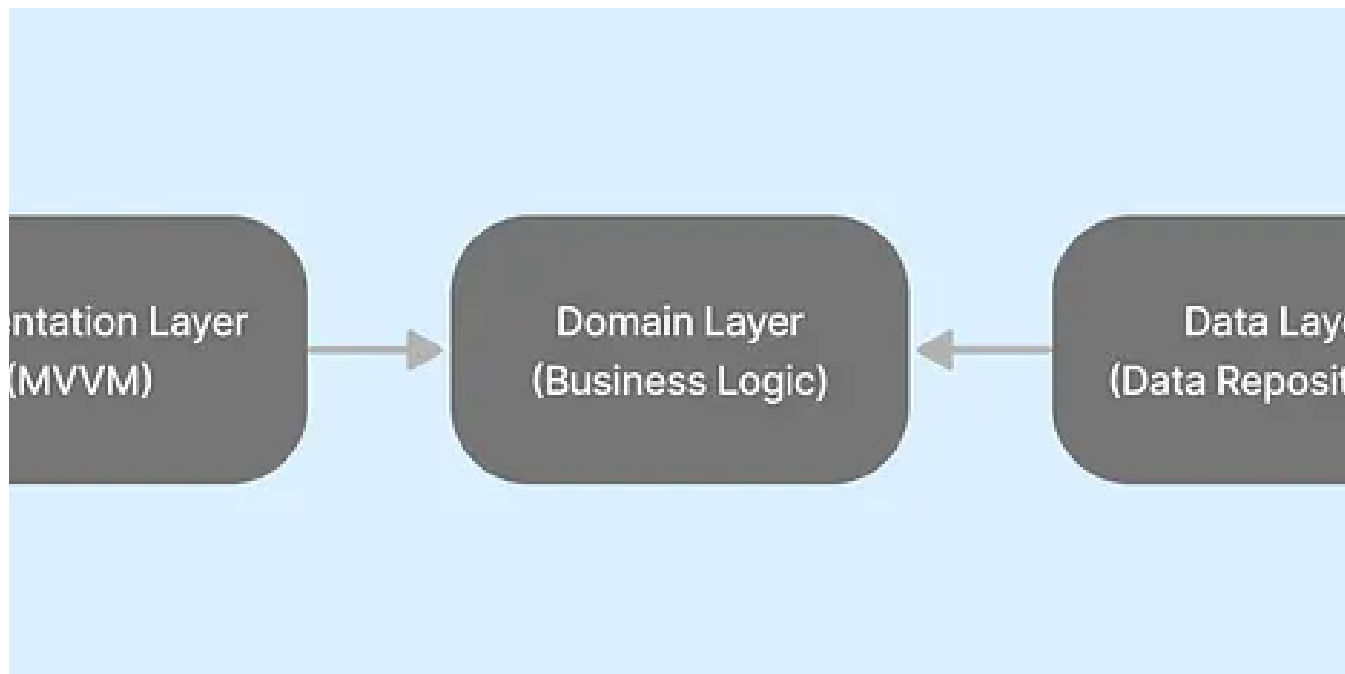


octavia

Seven Join Techniques in SQL

What is join? Join is a way to link data retrieved from tables through a column that links them. For example, the reader may want to link...

May 19, 2022 14



octavia

Step by Step SwiftUI with Clean Architecture

1.Understand Clean Architecture Clean Architecture is a software architecture pattern that separates code based on its responsibilities...