Apollo iOS GraphQL Integration with SwiftUI – Step-by-Step Guide

**Prerequisites**

macOS with Xcode 15 or above

Swift 5.7+

Swift Package Manager (SPM) – Apollo iOS

**Step 1: Setup SwiftUI Project**

Create a new SwiftUI project in Xcode:

→ Open Xcode → File → New → Project → App → SwiftUI

**Step 2: Install Apollo iOS Framework via Swift Package Manager**

1. Open Xcode → File → Add Packages

2. Enter URL: <https://github.com/apollographql/apollo-ios>

3. Select the latest stable version – 1.21.0 – next minor version

4. Add the package to your target

**Step 3: Setup Apollo config json**

3.1 Execute below mentioned command from project root directory to create apollo-codegen-config.json. it will provide default structure to map the graphql and respective schema, Verify the path are properly configured once config final initialized

**./apollo-ios-cli init --schema-namespace AppNameAPI --module-type swiftPackageManager**

Sample config file with schema details which will be downloaded when schema fetch command is executed  
  
{

  "schemaNamespace" : "MLAppAPI",

  "input" : {

    "operationSearchPaths" : [

      "\*\*/\*.graphql"

    ],

    "schemaSearchPaths" : [

      "\*\*/\*.graphqls"

    ]

  },

  "output" : {

    "testMocks" : {

      "none" : {

      }

    },

    "schemaTypes" : {

      "path" : "./MLAppAPI",

      "moduleType" : {

        "swiftPackageManager" : {

        }

      }

    },

    "operations" : {

      "inSchemaModule" : {

      }

    }

  },

  "schemaDownloadConfiguration": {

    "downloadMethod": {

        "introspection": {

            "endpointURL": "https://apollo-fullstack-tutorial.herokuapp.com/graphql",

            "httpMethod": {

                "POST": {}

            },

            "includeDeprecatedInputValues": **false**,

            "outputFormat": "SDL"

        }

    },

    "downloadTimeout": 60,

    "headers": [],

    "outputPath": "./graphql/schema.graphqls"

  }

}

**Step 4: Download GraphQL Schema**

To fetch schema from server execute below mentioned command in project root directory via terminal  
  
**./apollo-ios-cli fetch-schema**

**Step 5: Create .graphql Files**

Create .graphql files like LaunchListQuery.graphql:

**Sample query**

query LaunchList {

  launches {

    hasMore

    launches {

      id

      site

      mission {

        name

        missionPatch(size: SMALL)

      }

      rocket {

        name

      }

    }

  }

}

**Step 6: Generate GraphQL Code**

After adding queries/mutations in .graphql files, run:

**./apollo-ios-cli generate**    
  
Check Sources/GraphQL/ MLAppAPI for generated code  
  
Then map the generated code as local package and tag it with project target. Once its done you can invoke the respective query by creating apollo client manager , launch view model and render data with launch listview.

**Step 7: Apollo Client Manager**

Create a singleton Apollo client instance and replace GraphQL Url or use the existing sample url

**import** Apollo

**import** Foundation

**final** **class** ApolloClientManager {

**static** **let** shared = ApolloClientManager()

**private**(**set**) **lazy** **var** client: ApolloClient = {

**let** url = URL(string: "https://apollo-fullstack-tutorial.herokuapp.com/graphql")!

**return** ApolloClient(url: url)

    }()

}

**Step 8: Swift UI MVVM Integration**

Create a ViewModel to fetch data:

**import** Foundation

**import** Apollo

**import** MLAppAPI

@MainActor

**class** LaunchListViewModel: ObservableObject {

    @Published **var** launches: [MLAppAPI.LaunchListQuery.Data.Launches.Launch] = []

    @Published **var** isLoading = **false**

    @Published **var** errorMessage: String?

**private** **var** cursor: String?

**func** fetchLaunches() {

        isLoading = **true**

        errorMessage = **nil**

        print("➡️ Fetching launches...")

        ApolloClientManager.shared.client.fetch(query: MLAppAPI.LaunchListQuery()) { [**weak** **self**] result **in**

**guard** **let** self = **self** **else** { **return** }

            DispatchQueue.main.async {

                self.isLoading = **false**

**switch** result {

**case** .success(**let** graphQLResult):

                    print("GraphQL success")

**if** **let** data = graphQLResult.data?.launches {

**let** newLaunches = data.launches.compactMap { $0 }

                        print("Launches count: \(newLaunches.count)")

                        self.launches = newLaunches

                    } **else** **if** **let** errors = graphQLResult.errors {

                        self.errorMessage = errors.map { $0.localizedDescription }.joined(separator: "\n")

                        print("GraphQL errors: \(self.errorMessage ?? "")")

                    } **else** {

                        print("GraphQL success but data and errors are nil")

                        self.errorMessage = "No data returned"

                    }

**case** .failure(**let** error):

                    print("GraphQL network failure: \(error.localizedDescription)")

                    self.errorMessage = error.localizedDescription

                }

            }

        }

    }

**}**

**Step 9: Swift UI View**

**import** SwiftUI

**import** MLAppAPI

**struct** LaunchListView: View {

    @StateObject **private** **var** viewModel = LaunchListViewModel()

    @State **private** **var** selectedLaunch: MLAppAPI.LaunchListQuery.Data.Launches.Launch?

    @State **private** **var** hasLoaded = **false**

**var** body: **some** View {

        NavigationView {

            Group {

**if** viewModel.isLoading {

                    ProgressView("Loading Launches...")

                } **else** **if** **let** error = viewModel.errorMessage {

                    Text("Error: \(error)")

                        .foregroundColor(.red)

                } **else** {

                    List(viewModel.launches, id: \.id) { launch **in**

                        Button {

                            selectedLaunch = launch

                        } label: {

                            HStack(alignment: .top, spacing: 12) {

                                AsyncImage(url: URL(string: launch.mission?.missionPatch ?? "")) { phase **in**

**switch** phase {

**case** .empty:

                                        ProgressView()

                                            .frame(width: 100, height: 100)

**case** .success(**let** image):

                                        image

                                            .resizable()

                                            .aspectRatio(contentMode: .fit)

                                            .frame(width: 100, height: 100)

                                            .cornerRadius(8)

**case** .failure:

                                        Image(systemName: "photo")

                                            .resizable()

                                            .aspectRatio(contentMode: .fit)

                                            .frame(width: 100, height: 100)

                                            .foregroundColor(.gray)

**@unknown** **default**:

                                        EmptyView()

                                    }

                                }

                                VStack(alignment: .leading, spacing: 5) {

                                    Text("Mission: \(launch.mission?.name ?? "N/A")")

                                        .font(.headline)

                                    Text("Rocket: \(launch.rocket?.name ?? "N/A")")

                                        .font(.subheadline)

                                    Text("Site: \(launch.site ?? "N/A")")

                                        .font(.footnote)

                                        .foregroundColor(.gray)

                                }

                            }

                            .padding(.vertical, 6)

                        }

                    }

                    .listStyle(PlainListStyle())

                }

            }

            .navigationTitle("Launches")

            .onAppear {

**if** !hasLoaded {

                    hasLoaded = **true**

                    viewModel.fetchLaunches()

                }

            }

            .background(

                NavigationLink(

                    destination: LaunchDetailView(launch: selectedLaunch),

                    isActive: Binding(

                        get: { selectedLaunch != **nil** },

                        set: { **if** !$0 { selectedLaunch = **nil** } }

                    ),

                    label: { EmptyView() }

                )

            )

        }

    }

**}**

Sample screen based on above config and implementation

**Reference link**[**https://www.apollographql.com/**](https://www.apollographql.com/)[**https://www.apollographql.com/docs/ios**](https://apc01.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwww.apollographql.com%2Fdocs%2Fios&data=05%7C02%7CMaheshKumar.Bajaj%40cognizant.com%7C245bbd3520634be0df7a08ddb3fc270f%7Cde08c40719b9427d9fe8edf254300ca7%7C0%7C0%7C638864617760390392%7CUnknown%7CTWFpbGZsb3d8eyJFbXB0eU1hcGkiOnRydWUsIlYiOiIwLjAuMDAwMCIsIlAiOiJXaW4zMiIsIkFOIjoiTWFpbCIsIldUIjoyfQ%3D%3D%7C0%7C%7C%7C&sdata=qPaFwqPBXwcJtja%2FyARtDgZmD2gD1UXD3B8wiBXdH5Y%3D&reserved=0)[**https://studio.apollographql.com/sandbox/explorer**](https://studio.apollographql.com/sandbox/explorer)[**https://www.apollographql.com/tutorials/apollo-ios-swift-part1/01-intro-and-setup**](https://apc01.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwww.apollographql.com%2Ftutorials%2Fapollo-ios-swift-part1%2F01-intro-and-setup&data=05%7C02%7CMaheshKumar.Bajaj%40cognizant.com%7C245bbd3520634be0df7a08ddb3fc270f%7Cde08c40719b9427d9fe8edf254300ca7%7C0%7C0%7C638864617760410494%7CUnknown%7CTWFpbGZsb3d8eyJFbXB0eU1hcGkiOnRydWUsIlYiOiIwLjAuMDAwMCIsIlAiOiJXaW4zMiIsIkFOIjoiTWFpbCIsIldUIjoyfQ%3D%3D%7C0%7C%7C%7C&sdata=2SUlkg5Vi8o9Oxf0ebMPYczT%2FODNBl09o8vDsLbyA3Y%3D&reserved=0)

**Final Checks**

1. Apollo iOS SPM and Apollo CLI installed (apollo --version)
2. schema.graphqls is downloaded
3. .graphql queries written
4. apollo.config.json correctly set
5. Swift types generated – codegen
6. Apollo Client used in SwiftUI ViewModel and View
7. SwiftUI views displaying data