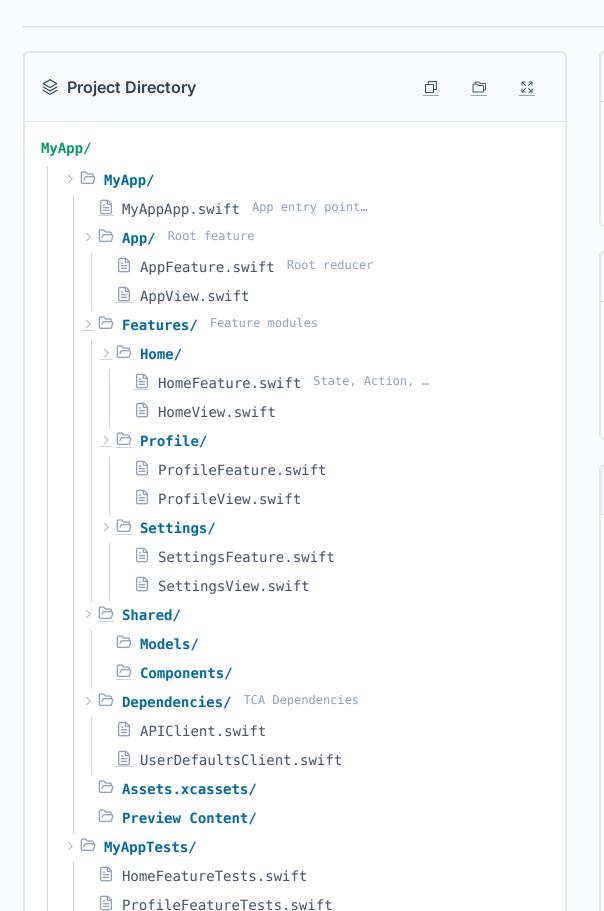
SwiftUI TCA Project Structure

The Composable Architecture with reducers, effects, and unidirectional data flow.

#swiftui #swift #ios #tca #composable #redux

MyApp.xcodeproj

.gitignore



O Why This Structure?

TCA enforces unidirectional data flow: State → View → Action → Reducer → State. Every feature is a self-contained module with its own State, Action, and Reducer. Features compose together. Side effects are explicit and testable via Dependencies.

☑ PNG

PDF PDF

🗇 Сору

</> Prompt

Features/ - Each feature has its own State, Action, Reducer, and View **App**/ - Root feature that composes child features

Dependencies/ - TCA dependency clients for APIs, storage, etc.

Shared/ - Models and components used across features

</> Feature Reducer

```
// Features/Home/HomeFeature.swift
@Reducer
struct HomeFeature {
    @ObservableState
    struct State: Equatable {
        var posts: [Post] = []
        var isLoading = false
    }
    enum Action {
        case onAppear
        case postsLoaded([Post])
    @Dependency(\.apiClient) var apiClient
    var body: some ReducerOf {
        Reduce { state, action in
            switch action {
            case .onAppear:
                state.isLoading = true
                return .run { send in
                    let posts = try await apiClient.fetchP
                    await send(.postsLoaded(posts))
            case .postsLoaded(let posts):
                state.isLoading = false
                state.posts = posts
                return .none
       }
   }
}
```

>_ Getting Started

- 1. File → Add Package → github.com/pointfreeco/swift
 - composable-architecture
- Create root AppFeature with Store
 Add features in Features/ folder
- 4. Define dependencies in Dependencies/

☑ When To Use This

- Large apps with complex state interactions
- Need exhaustive unit testing of logic
- Apps with many side effects to coordinate
- Teams that want strict architectural patterns
- Apps requiring time-travel debugging

া Trade-offs

Learning curve - TCA concepts take time to internalize

Verbose - State, Action, Reducer for every feature

Compile times - Heavy macro usage can slow builds

Aa Naming Conventions

Features - {Name}Feature.swift contains State, Action, Reducer
Views - {Name}View.swift paired with its Feature
Dependencies - {Name}Client.swift (APIClient, StorageClient)

☑ Best Practices

- Use @Reducer macro for less boilerplate
- Keep reducers pure—side effects go in .run
- Mock dependencies in tests via withDependencies
- Compose features using Scope reducer
- Use @ObservableState for iOS 17+ observation