Architecture globale iOS

Par Vincent Ducastel 15 1

- Description
  - Application en Multi Modules
  - Clean Architecture
  - MVVM-C
- Structure
  - Dossier LocalPackages
    - Dossier Features
    - Dossier Core
  - Dossier Coyote
    - Dossier App
  - Package FeaturePresentation
  - Package FeatureDomain
  - Package FeatureDomainApi
  - Package FeatureData
  - Package Core
- Ressources
  - Clean Architecture
  - Factory modular dependency injection

Description

Le projet sera structuré en Multi-Modules et en Clean Architecture.

Voici le projet Coyote V12 : https://git.coyotesystems.com/coyote-ios-apps/apps/alerting/

On utilise 3 grands principes :

- Application en Multi Modules
- Clean Architecture

- MVVM-C

Mise à jour : sept. 09, 2024 Modifier

Application en Multi Modules

On utilise SPM pour découper le projet en package et en module. Le projet est composé :

- Des packages **Feature**, code métier en lien avec l'application
- Des packages **Core**, code utiles sans lien avec le code métier de l'application

Les modules sont ici : Modules | Modules Features

Clean Architecture

Une Feature sera décomposé en plusieurs couches en utilisant l'architecture "Clean Architecture"

- Découpage en 3 couches :
  - **Presentation** :
    - un package **FeaturePresentation** : portion de UI de la feature
  - **Domain** :
    - un package **FeatureDomain** : le code métier de la feature
    - un package **FeatureDomainApi** : code métier de la feature exposé à d'autres features
  - **Data** :
    - un package **FeatureData** : les données de la feature

Plus d'information ici : https://git.coyotesystems.com/poc-sample/ios/poc-clean-archi#clean-architecture-overview

MVVM-C

-

☐ @Vincent Ducastel MVVM-C trouvé une doc confluence déjà écrite

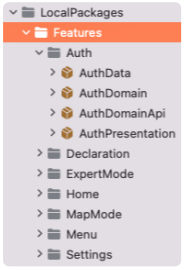
Structure

Mise à jour : sept. 09, 2024 Modifier

Contient 2 dossiers : **Features** et **Core**

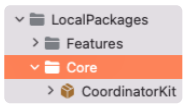
Dossier Features

Contient les différentes packages **Features** de l'application



Dossier Core

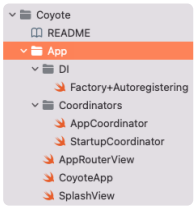
Contient les différentes packages **Core**



Dossier Coyote

Dossier App

Contient le code principal qui est le point d'entrée de l'application



Package FeaturePresentation

- Contient un module FeaturePresentation
  - Contient des View, ViewModel, Coordinator, Route, ViewBuilder
  - Peut contenir d'autres vues
- Un package FeaturePresentation
  - ✅ peut dépendre de 1 ou plusieurs packages FeaturePresentation
  - ✅ peut dépendre du package FeatureDomain de la même feature
  - ❌ ne doit pas dépendre des packages FeatureDomain d'autres features

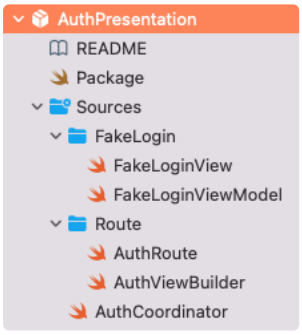
- ✅ peut dépendre du package FeatureDomainApi de la même feature
- ✅ peut dépendre de 1 ou plusieurs packages FeatureDomainApi d'autres

Mise à jour : sept. 09, 2024 Modifier

- ❌ ne doit pas dépendre des packages FeatureData
- Core
  - ✅ peut dépendre de 1 ou plusieurs packages Core utiles, DesignSystem, Utils, Tracking, CoordinatorKit, Translations, PreviewKit, Sound, MapProvider, Configuration, Location, Position
  - ❌ ne doit pas dépendre des packages CoreLibrary, Purchase, WebServices, Repository, XmlRpcServices, PushNotif, CoreSecurity

Exemple avec la feature Auth

- Feature : Auth
- Package : AuthPresentation
- Module : AuthPresentation




```
let package = Package(
    name: "AuthPresentation",
    platforms: [.iOS(.v16)],
    products: [
        .library(
            name: "AuthPresentation",
            targets: ["AuthPresentation"]
        )
    ],
    dependencies: [
        .package(name: "CoordinatorKit", path: "../../../Core/CoordinatorKit"),
        .package(name: "AuthDomain", path: "../AuthDomain")
    ],
    targets: [
        .target(
            name: "AuthPresentation",
            dependencies: [
                "AuthDomain",
                "CoordinatorKit"
            ],
            path: "./Sources"
        )
    ]
)
```

Package FeatureDomain

- Contient un module FeatureDomain
  - Contient des UseCase et des UseCaseProtocol **interne** à la feature
  - Contient des RepositoryProtocol qui seront implémentés par le module Dat
- Un package FeatureDomain
  - ❌ ne doit pas dépendre des packages FeaturePresentation
  - ❌ ne doit pas dépendre d'autres packages FeatureDomain
  - ✅ peut dépendre de 1 ou plusieurs packages FeatureDomainApi

- ❌ ne doit pas dépendre des packages FeatureData
- Core

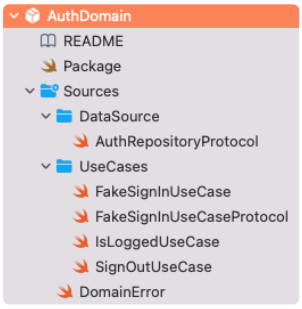
Mise à jour : sept. 09, 2024 Modifier

SharedEntities

- ❌ ne doit pas dépendre des autres packages Core

Exemple avec la feature Auth

- Feature : Auth
- Package : AuthDomain
- Module : AuthDomain




```
let package = Package(
    name: "AuthDomain",
    platforms: [.iOS(.v16)],
    products: [
        .library(
            name: "AuthDomain",
            targets: ["AuthDomain"]
        )
    ],
    dependencies: [
        .package(name: "AuthDomainApi", path: "../AuthDomainApi")
    ],
    targets: [
        .target(
            name: "AuthDomain",
            dependencies: ["AuthDomainApi"],
            path: "./Sources"
        )
    ]
)
```

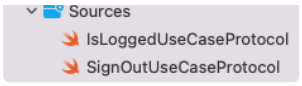
Package FeatureDomainApi

- Contient un module FeatureDomainApi
  - Contient les UseCaseProtocol **externes** qui seront exposés à d'autres features
  - Contient les EntityProtocol **externes** qui seront exposés à d'autres features
- Un package FeatureDomainApi
  - ✅ doit dépendre de Factory
  - ❌ ne doit avoir aucune autre dépendance

Exemple avec la feature Auth

- Feature : Auth
- Package : AuthDomainApi
- Module : AuthDomainApi

Mise à jour : sept. 09, 2024 Modifier




```
            name: "AuthDomainApi",
            targets: ["AuthDomainApi"])
    ],
    dependencies: [
        .package(url: "https://github.com/hmlongco/Factory", from: "2.2.0")
    ],
    targets: [
        .target(
            name: "AuthDomainApi",
            dependencies: ["Factory"],
            path: "./Sources"
        )
    ]
)
```

Package FeatureData

- Contient un module FeatureData
  - Contient les Repository (implémentations des RepositoryProtocol défini dans le package FeatureDomain)
  - Contient une classe <Feature>Dependencies qui autoRegister les implémentations pour l'injection de dépendance Factory (class Container)
    - Exemple avec AuthDependencies :
    - ```
      public enum AuthDependencies {

          public static func autoRegister(_ container: Container) {
              container.authRepository.register {
                  AuthRepository()
              }.scope(.singleton)

              container.fakeSignInUseCase.register {
                  FakeSignInUseCase()
              }

              container.isLoggedUseCase.register {
                  IsLoggedUseCase()
              }

              container.signOutUseCase.register {
                  SignOutUseCase()
              }
          }
      }
      ```
- Un package FeatureData
  - ❌ ne doit pas dépendre des packages FeaturePresentation
  - ✅ peut dépendre du package FeatureDomain de la même feature
      ```

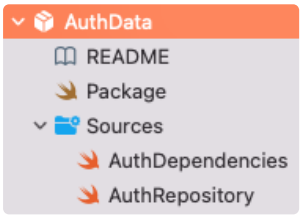
- ✅ peut dépendre du package FeatureDomainApi de la même feature
- ❌ ne doit pas dépendre des packages FeatureDomainApi d'une autre feature

Mise à jour : sept. 09, 2024 Modifier

- ✅ peut dépendre de 1 ou plusieurs packages Core

Exemple avec la feature Auth

- Feature : Auth
- Package : AuthData
- Module : AuthData



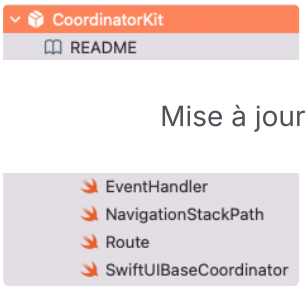

```
let package = Package(
    name: "AuthData",
    platforms: [.iOS(.v16)],
    products: [
        .library(
            name: "AuthData",
            targets: ["AuthData"]
        )
    ],
    dependencies: [
        .package(name: "AuthDomain", path: "../AuthDomain")
    ],
    targets: [
        .target(
            name: "AuthData",
            dependencies: ["AuthDomain"],
            path: "./Sources"
        )
    ]
)
```

Package Core

- Contient un module Core
- Un package Core
  - ✅ peut dépendre de 1 ou plusieurs packages Core
  - ❌ ne doit pas dépendre du package FeaturePresentation
  - ❌ ne doit pas dépendre du package FeatureDomain
  - ❌ ne doit pas dépendre du package FeatureDomainApi
  - ❌ ne doit pas dépendre du package FeatureData

Exemple avec CoordinatorKit

- Package : CoordinatorKit
- Module : CoordinatorKit




```
let package = Package(
    name: "CoordinatorKit",
            targets: ["CoordinatorKit"])
    ],
    dependencies: [
    ],
    targets: [
        .target(
            name: "CoordinatorKit",
            dependencies: []),
        .testTarget(
            name: "CoordinatorTests",
            dependencies: ["CoordinatorKit"])
    ]
)
```

Mise à jour : sept. 09, 2024 Modifier

Ressources

Clean Architecture

- The “Real” Clean Architecture in Android: Modularization
- Clean Architecture and MVVM on iOS Clean Architecture and MVVM on iOS
- Clean Architecture for MassiveToBe Mobile Apps Clean Architecture for MassiveToBe Mobile Apps
- Clean Architecture: iOS App
- The Clean Code Blog Clean Coder Blog

Factory modular dependency injection

- Factory Modular Development
  https://hmlongco.github.io/Factory/documentation/factory/modules
- Factory Multiple Module Registration Factory: Multiple Module Registration

Contenu connexe

App Architecture
Coyote iOS

Plus de contenu similaire

Samples iOS
DSI

Plus de contenu similaire

Mise à jour : sept. 09, 2024 Modifier

Plus de contenu similaire

Proposition d'organisation des modules refontes UI
Coyote Android

Plus de contenu similaire

Guide d'utilisation de Firebase
DSI

Lue avec ce contenu

Architecture CarPlay Coyote V12
Coyote iOS

Lue avec ce contenu

Ajouter un commentaire V12 1