CareemEngineering

Jun 2023

CareemEngineering

Modular iOS Architecture

Feature Modularization

Osama Gamal

Jun 2023

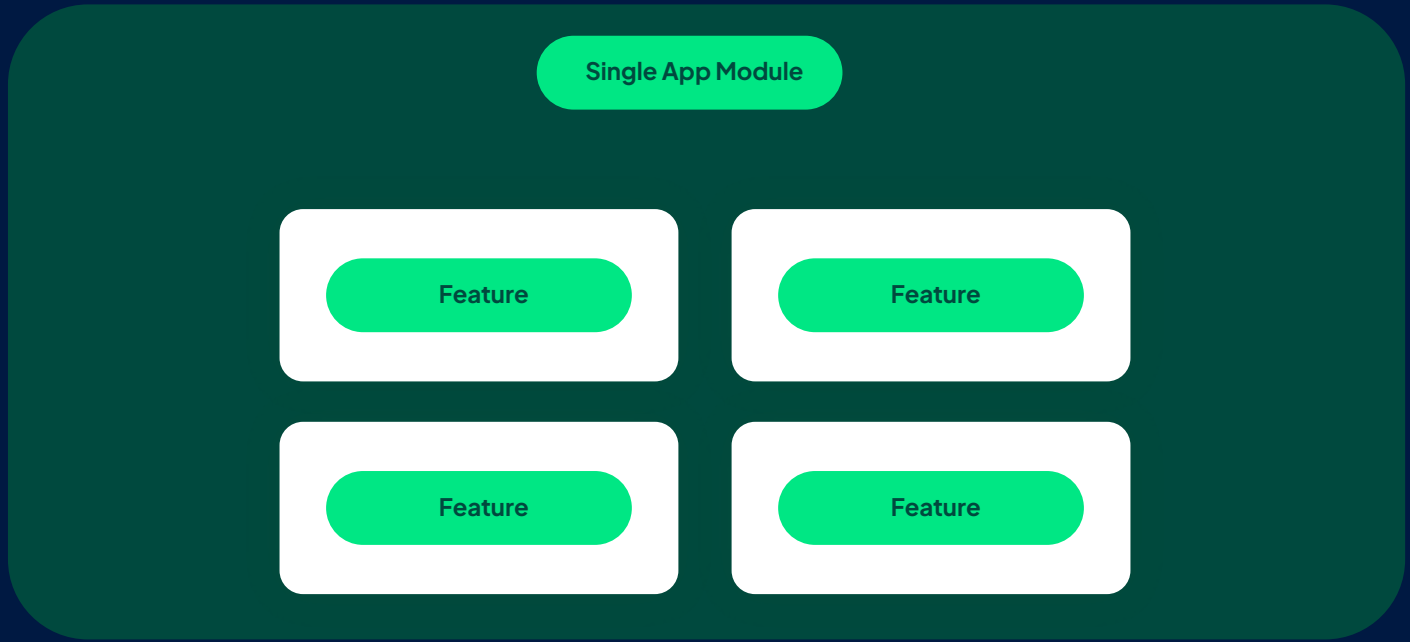
This is Osama Gamal

Senior iOS Engineer @ Careem
Global Experiences Team

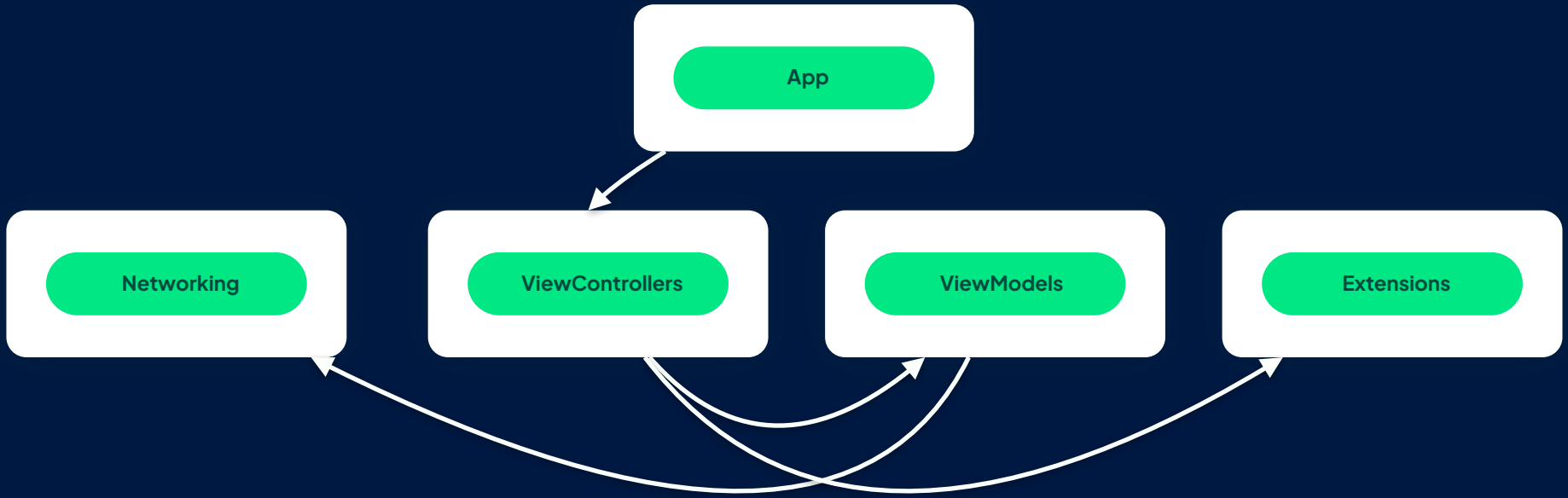
- ex iOS Engineer @ Swvl
- ex iOS Engineer @ Sary
(Both are Careem Mafia 🤓)

- Founder of PhoneCaller Cydia tweak for jailbroken devices
- Founder of SwiftZilla iOS Community

Monolith App



Monolith App



What do you know about **Modularization**?

Modularization (What)

Imagine that we made a small change in a screen, and we want to try to see if this change works, Currently we will need to recompile the entire codebase to see the results.

This process is time consuming.

We should be able to compile and run parts independently, in isolation.

In large scale projects (and maybe any project), the capability of creating, running, and testing parts in isolation is a **necessity**

Why **modularization**?

Why Modularization

01

Faster build times

You can mark modules as binary, which prevents unchanged modules from being built!

02

Demo Apps

Each module is its own project, your team can work separately on his module

03

Reusability

You can easily reuse your modules in other apps, since they are isolated.

04

Testability

Your tests will run faster, for your module only, Imagine integrating CI/CD checks only for your feature!

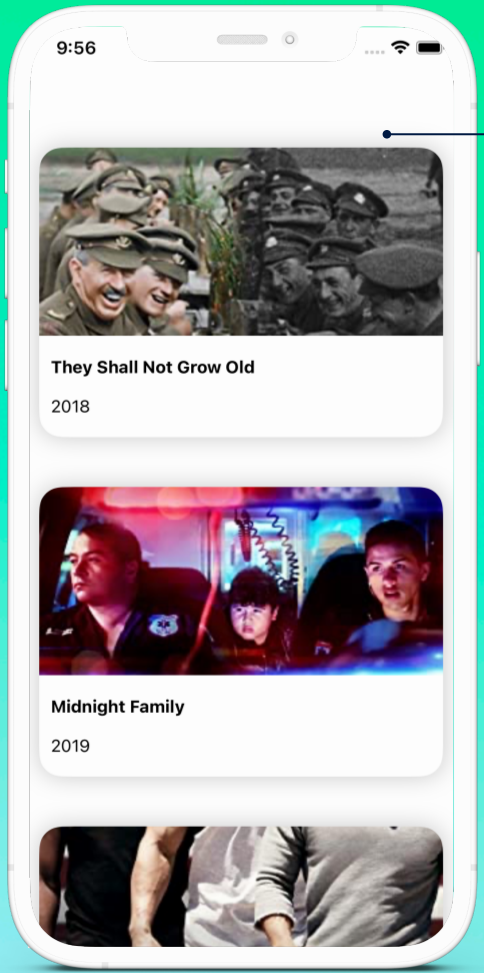
Why Modularization

05

Decoupling

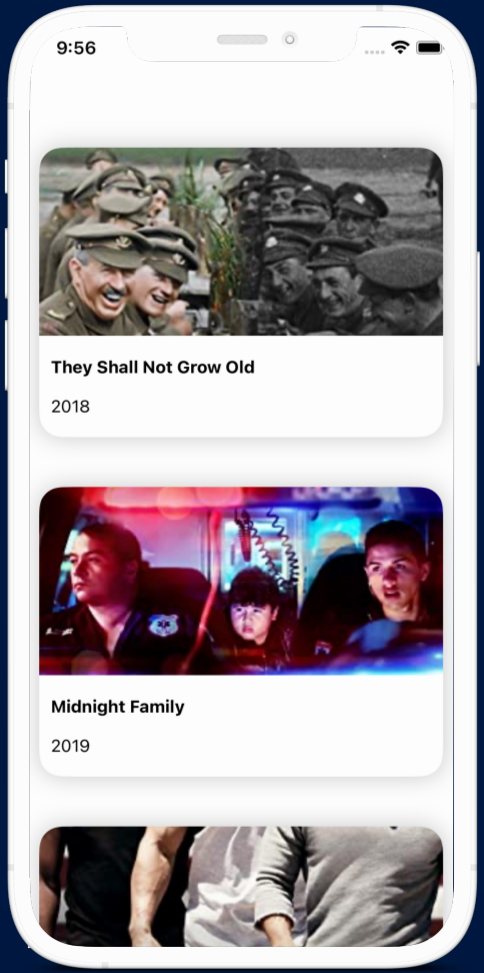
Each module is isolated, and can be located in a separate repository (its own CI/CD workflows, PRs, commits, changelogs)

We have a movie list app
It has two features
MoviesList
&
MovieDetails



- MoviesListViewController.swift
  + (Interactor/Presenter)
- NetworkService.swift
- Movie.swift (Entity)
- CachingService

- *Kingfisher*



Modularized

SwiftCairoMovies

Application

ConsumerNetworking

Networking module

SwiftCairoCache

Exposes storing/fetching functionality
Requires consumer id for initialization

SwiftCairoCommon

Common domain layer that shares common interfaces between modules

MoviesListFeature

Feature module for movies list

SwiftCairo DesignSystem

Design System module, provides UI Components for the app/modules to consume

MovieDetailsFeature

Feature module for movie details

Setting up module

Using:

- Cocoapods
- Swift Package Manager
- Nested Xcode Projects

Its up to you to choose..

In this talk I'll focus on Cocoapods since mostly everyone here knows about it, so I can familiarize you quickly on the setup, and not to confuse you..

However, if you're getting started, I recommend SPM :)

Create our first module

using Cocoapods

SwiftCairoCommon.podspec

```
Pod::Spec.new do |spec|
    spec.name                  = 'SwiftCairoCommon'
    spec.version               = '1.0.0'
    spec.summary               = 'Domain Module for SwiftCairoMovies Project'
    spec.homepage              = 'https://swiftcairo.com'
    spec.source                = { :git => '.', :tag => spec.version.to_s }
    spec.author                = { 'Osama Gamal' => 'me@i0sa.com' }

    spec.ios.deployment_target = '13.0'

    spec.source_files          = 'SwiftCairoCommon/**/*.{swift}'
end
```

ConsumerNetworking.podspec

```
Pod::Spec.new do |spec|
  spec.name                  = 'ConsumerNetworking'
  spec.version               = '1.0.0'
  spec.summary               = 'Consumer networking'
  spec.homepage              = 'https://swiftcairo.com'
  spec.source                = { :git => '.', :tag => spec.version.to_s }
  spec.author                = { 'Osama Gamal' => 'me@i0sa.com' }

  spec.ios.deployment_target  = '13.0'

  spec.source_files          = 'ConsumerNetworking/ConsumerNetworking/**/*.{swift}'
  spec.dependency              'SwiftCairoCommon'

  spec.test_spec  'UnitTests' do |test_spec|
    test_spec.source_files  = 'ConsumerNetworking/Tests/**/*'
    test_spec.requires_app_host = true
  end
end
```

```ruby
Pod::Spec.new do |spec|
    spec.name                   = 'MoviesListFeature'
    spec.version                = '1.0.0'
    spec.summary                = 'Movies List Feature Module for SwiftCairoMovies Project'
    spec.homepage               = 'https://swiftcairo.com'
    spec.source                 = { :git => '.', :tag => spec.version.to_s }
    spec.author                 = { 'Osama Gamal' => 'me@i0sa.com' }

    spec.ios.deployment_target  = '13.0'

    spec.source_files           = 'MoviesListFeature/MoviesListFeature/**/*.{swift}'
    spec.resource_bundle = {
      'MoviesListFeature' => [
        'MoviesListFeature/MoviesListFeature/**/*.{xib,xcassets}'
      ]
    }

    spec.dependency               'SwiftCairoCommon'
    spec.dependency               'SwiftCairoDesignSystem'

    spec.test_spec  'UnitTests' do |test_spec|
        test_spec.source_files  = 'MoviesListFeature/Tests/**/*'
        test_spec.requires_app_host = true
    end
end
```

podfile?

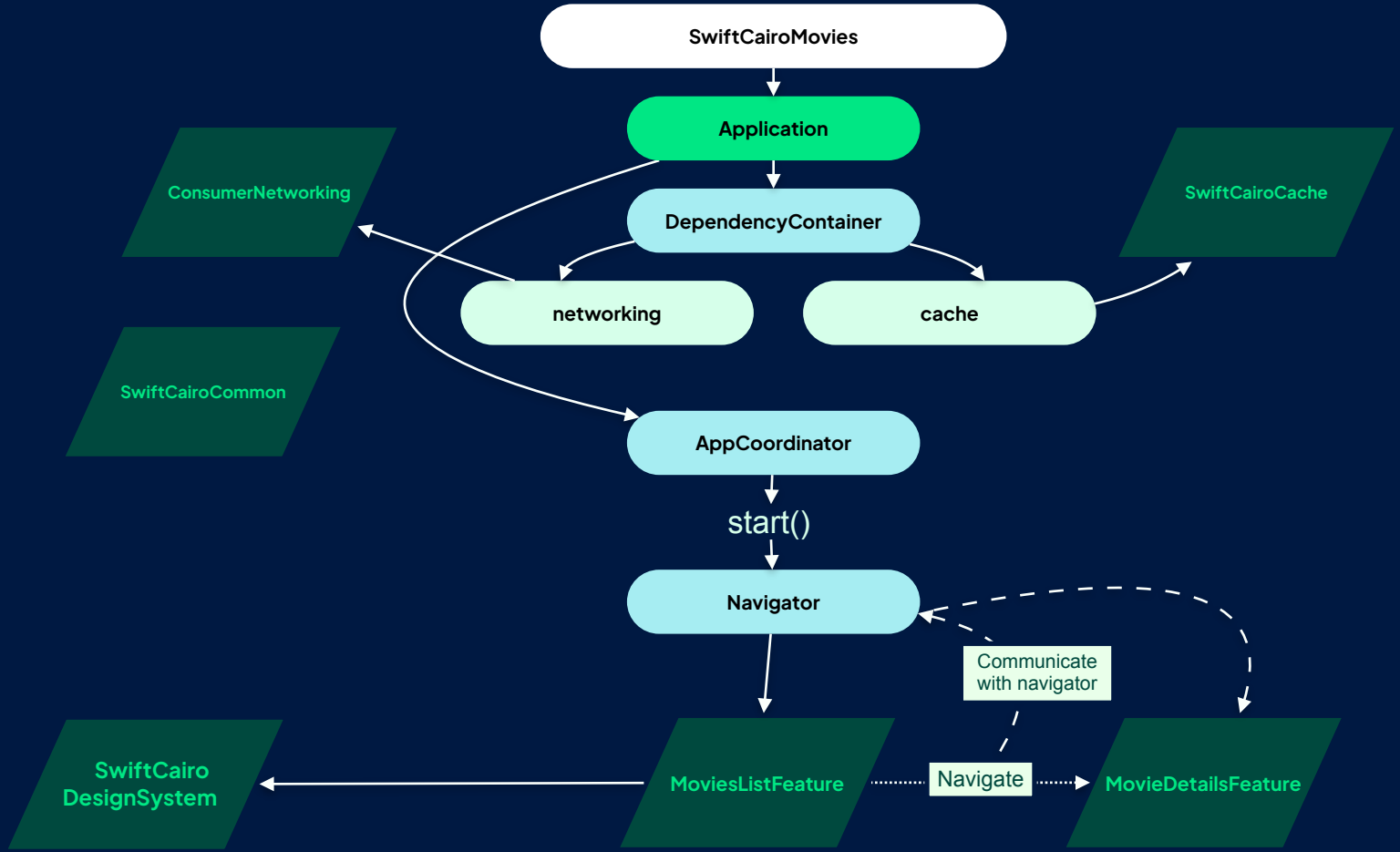
```
use_frameworks!
inhibit_all_warnings!
platform :ios, '13.0'

def features
  pod 'MoviesListFeature', :path => '.'
  pod 'MovieDetailsFeature', :path => '.'
end


target 'iOS Test' do
  pod 'ConsumerNetworking', :path => '.'
  pod 'SwiftCairoCommon', :path => '.'
  pod 'SwiftCairoDesignSystem', :path => '.'
  pod 'SwiftCairoCache', :path => '.'
  features
end
```

Side note

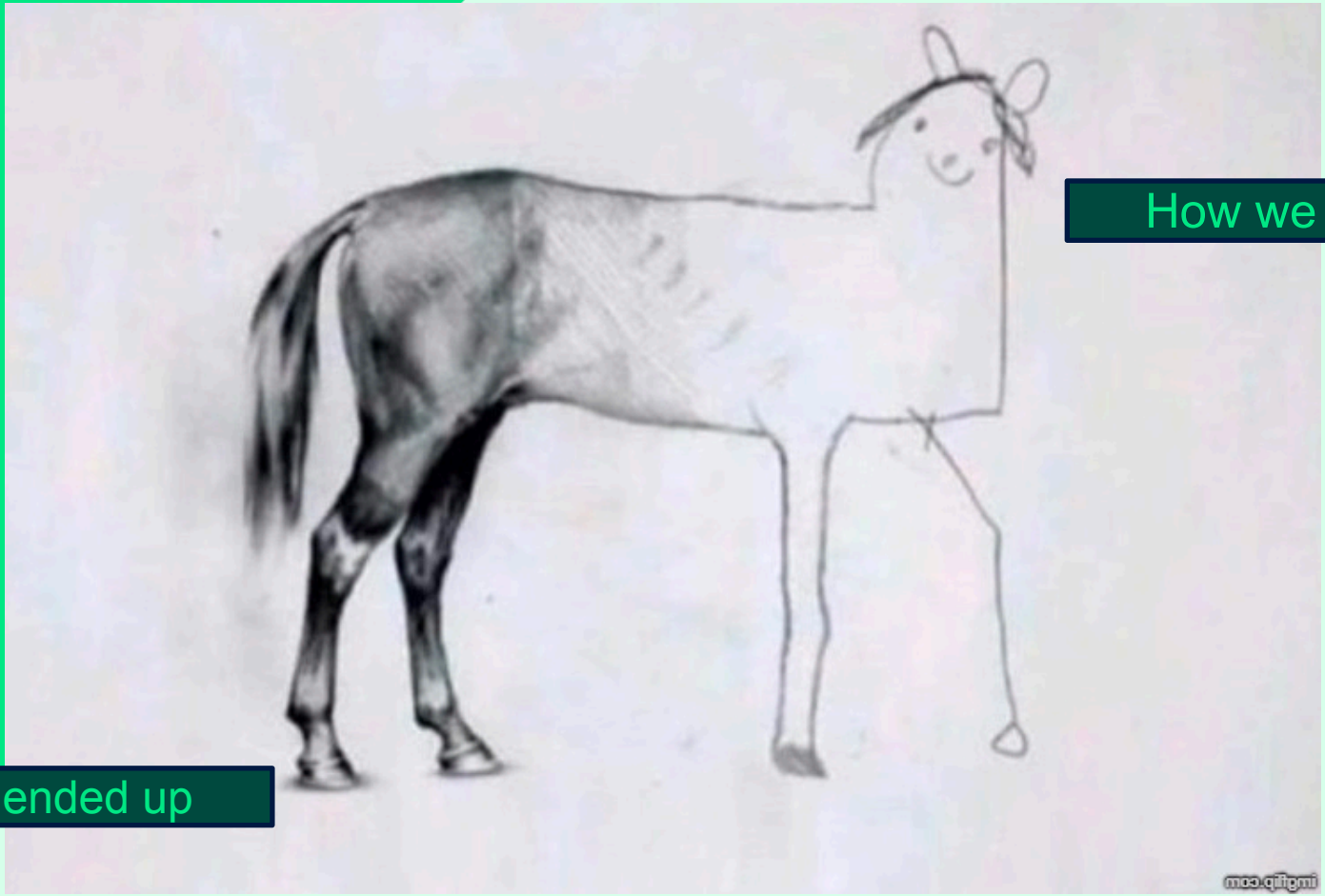
Those pods are defined as local pods for development purposes

But ideally, you can host them remotely on GitHub, so you can version control, etc

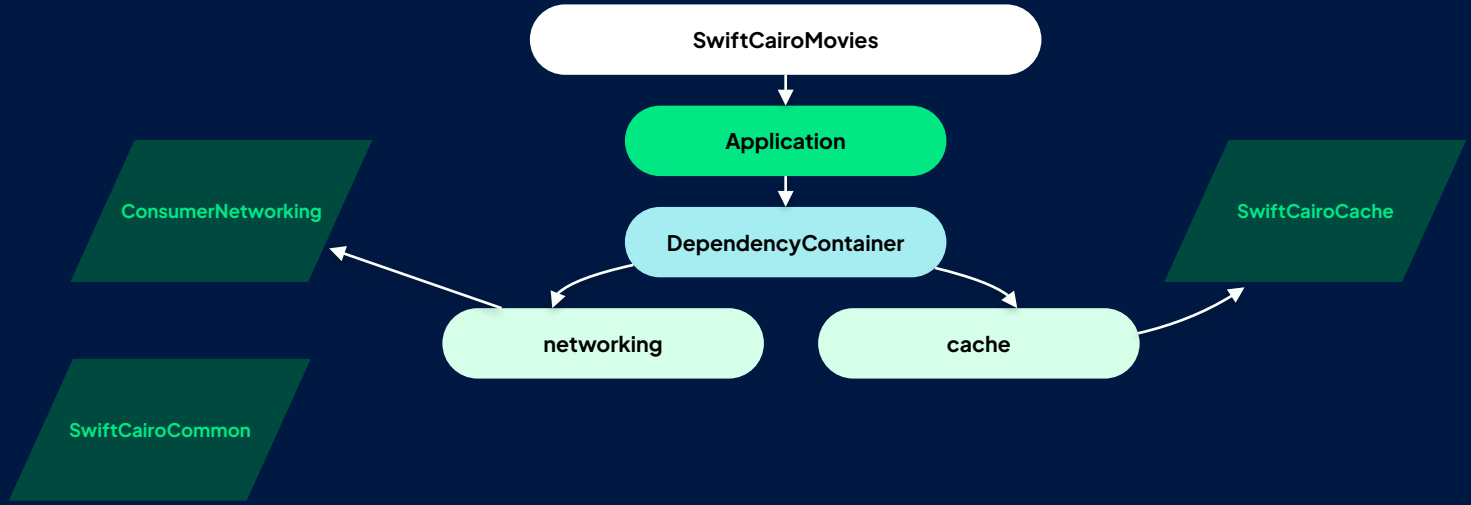
Now to the tricky part..
How will they **connect**?


SwiftCairoMovies
Application
ConsumerNetworking
DependencyContainer
SwiftCairoCache
networking
cache
SwiftCairoCommon
AppCoordinator
start()
Navigator
Communicate with navigator
SwiftCairo DesignSystem
MoviesListFeature
Navigate
MovieDetailsFeature

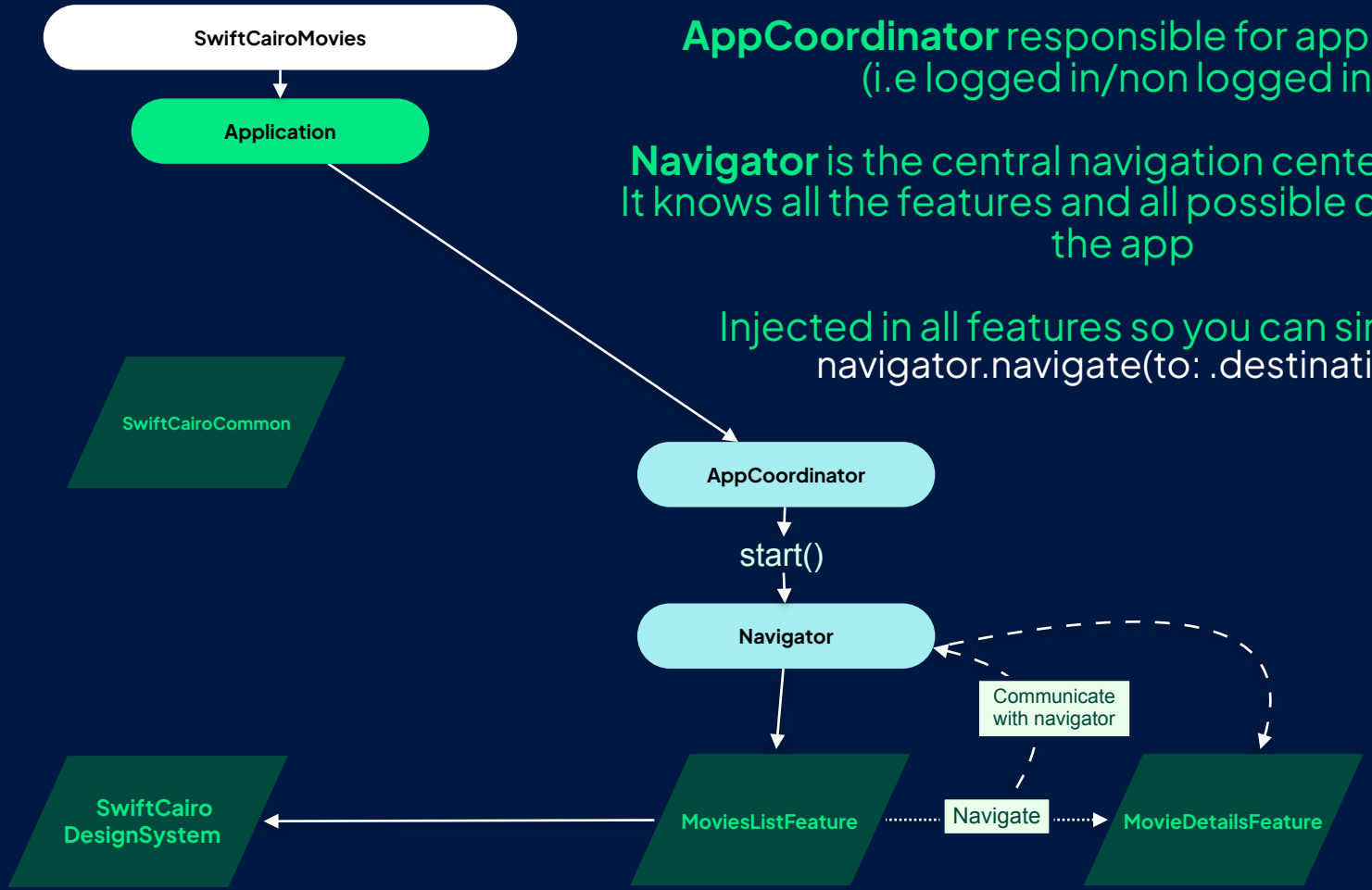
How we started



How it ended up



Dependency container is instantiated from main app (i.e AppDelegate) and its responsible for creating main app dependencies that features will need



AppCoordinator responsible for app start logic (i.e logged in/non logged in)

Navigator is the central navigation center for our app, It knows all the features and all possible destinations in the app

Injected in all features so you can simply call navigator.navigate(to: .destination)

Wait.. Why do we need **SwiftCairoCommon** module ?

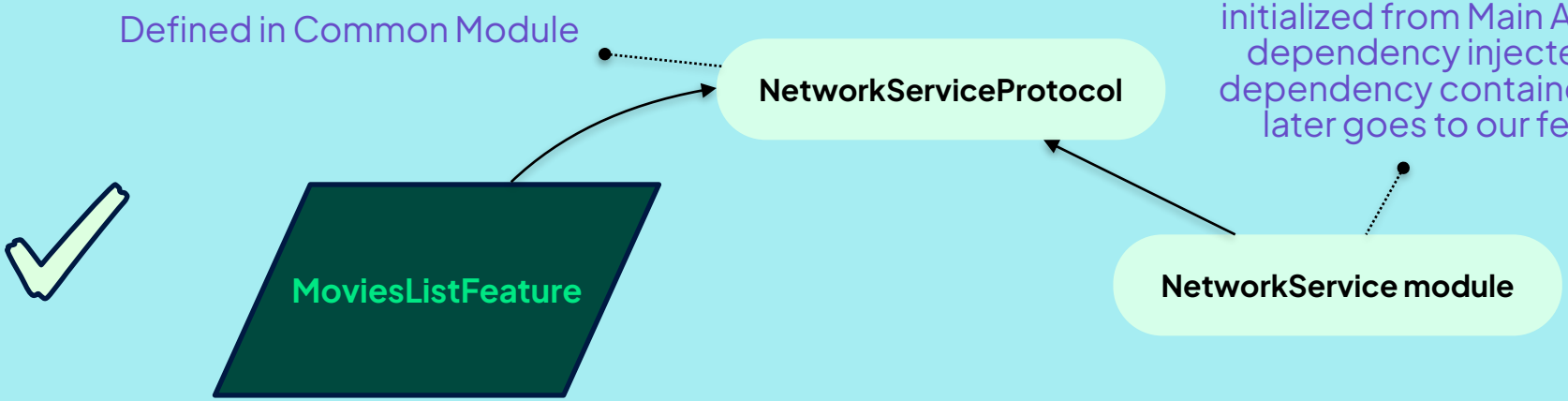
Remember dependency inversion principle ?
“High level modules **should not** depend on low level modules,
both should depend on abstractions.”

Simply, because we want to use networking in a feature, it **DOES NOT** mean i should import networking module in the feature, otherwise handling dependencies will be **hell**

So we have a common module, that defines abstractions (interfaces), which can be shared across modules

Where actual implementation init happens in the main app only

Wait.. Why do we need **SwiftCairoCommon** module ?



Live Demo

Conclusion

01

I’m not large scale app..
It’s totally fine to apply modular architecture in ANY size of app, it will help you eventually with scaling and clean code

02

Why should I care ?
Its **essential** for you nowadays to learn this kind of architecture, world is moving fast..

03

Modular communication
Modular structuring and communication can come in various shapes

Thank you.