

A Framework Driven Development

여러분의 프로젝트는 안녕하신가요?

안정민

한국카카오은행

소개

- 現 카카오뱅크 iOS 개발자 (16.10 ~)
- 블로그 운영 : minsone.github.io

목차

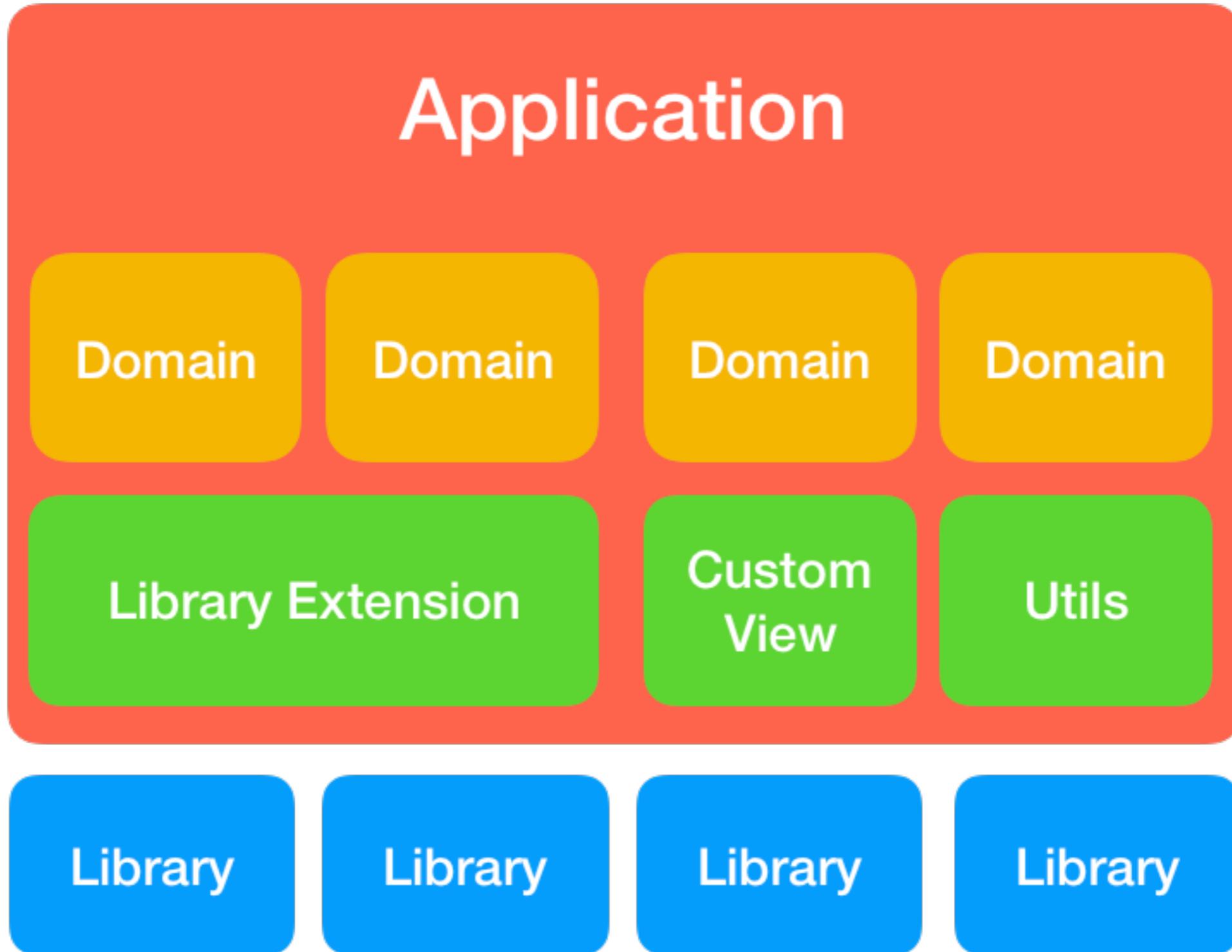
- 기존 개발 방법과 문제점
- 프레임워크 주도 개발
- 프로젝트 재구성
- 프로젝트 리팩토링
- 참고자료
- QnA

기존 개발 방법과 문제점

기존 개발 방법

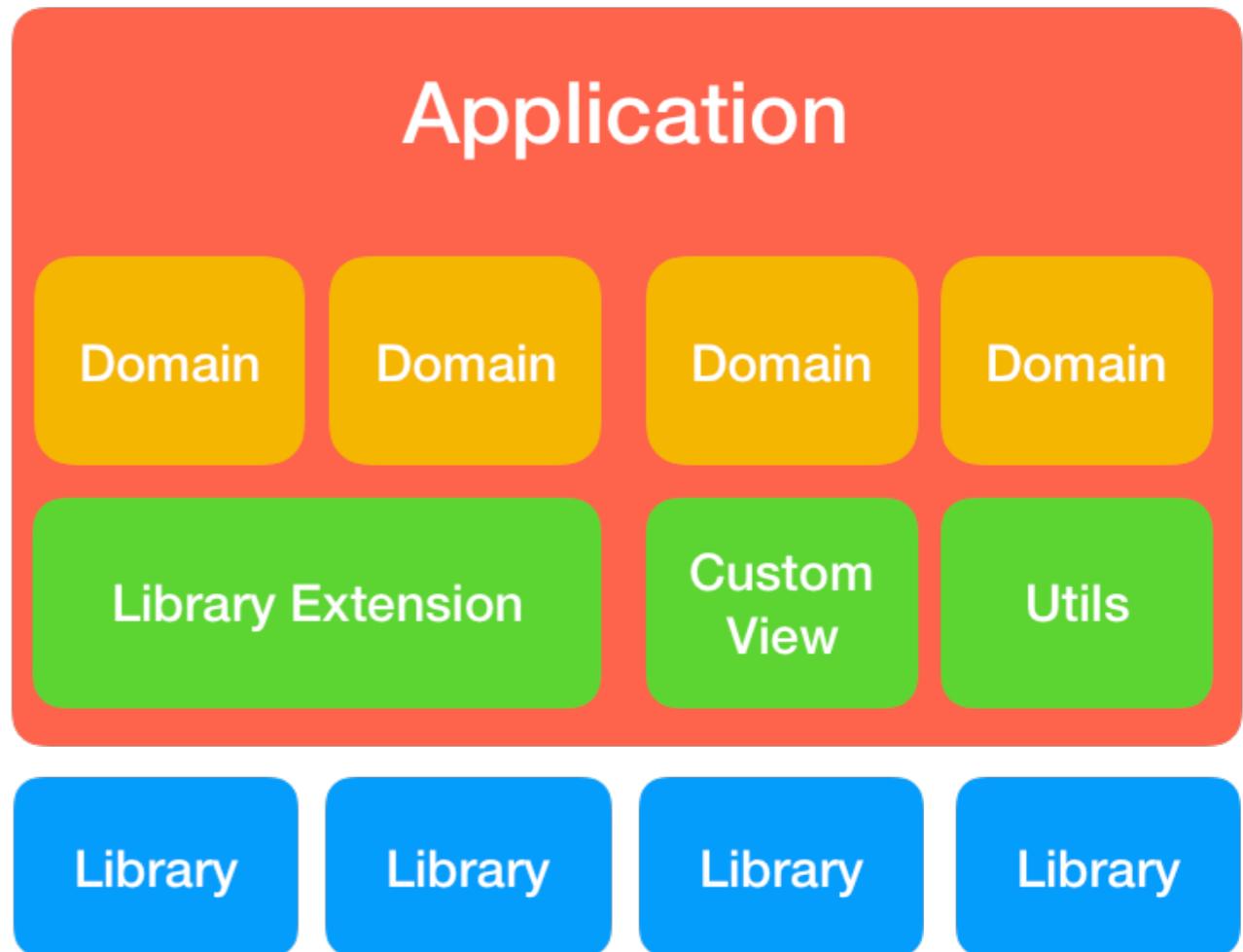
- 기존 개발 방법
- 메인 프로젝트 + 다수의 오픈소스 이용
- Carthage, Cocoapods으로 관리

기존 개발 방법



기존 개발 방법

- 파일 단위로 레이어를 분리
- 소스 코드의 집중화
- 라이브러리를 직접 관리
- 오픈소스, 외부 SDK를 직접 사용.



기존 개발 방법

- 기존 개발 방법의 이유
 - 빌드 속도가 비교적 빠른 Objective-C
 - 의존성 관리도구를 지원하지 않는 Xcode
 - 프레임워크 작성 및 관리의 귀찮음
 - 소규모 프로젝트가 대부분
 - 그냥 그렇게 배워서

기존 개발 방법의 문제점

- 기존 개발 방법의 문제점
 - 코드, 파일 개수가 많아질수록 느려지는 Swift

기존 개발 방법의 문제점

- 기존 개발 방법의 문제점
 - 코드, 파일 개수가 많아질수록 느려지는 Swift
 - 소스 코드의 집중화

A.swift

protocol 카드BaseAPI {}

B.swift

class 대출API {}

A.swift

```
protocol 카드BaseAPI {}
```

B.swift

```
class 대출API:카드BaseAPI {}
```

기존 개발 방법의 문제점

- 기존 개발 방법의 문제점
 - 코드, 파일 개수가 많아질수록 느려지는 Swift
 - 소스 코드의 집중화
 - 응집도 감소, 결합도 증가, 부수효과 증가

기존 개발 방법의 문제점

- 기존 개발 방법의 문제점
 - 코드, 파일 개수가 많아질수록 느려지는 Swift
 - 소스 코드의 집중화
 - 응집도 감소, 결합도 증가, 부수효과 증가
 - 외부 SDK 및 오픈소스 적용 후 관리 비용 높음.
 - 관리지점 - Swift 버전, iOS 버전, 테스트 등

프레임워크 주도 개발

프레임워크로 레이어를 나누어 개발

A.swift

```
protocol 카드BaseAPI {}
```

B.swift

```
class 대출API:카드BaseAPI {}
```

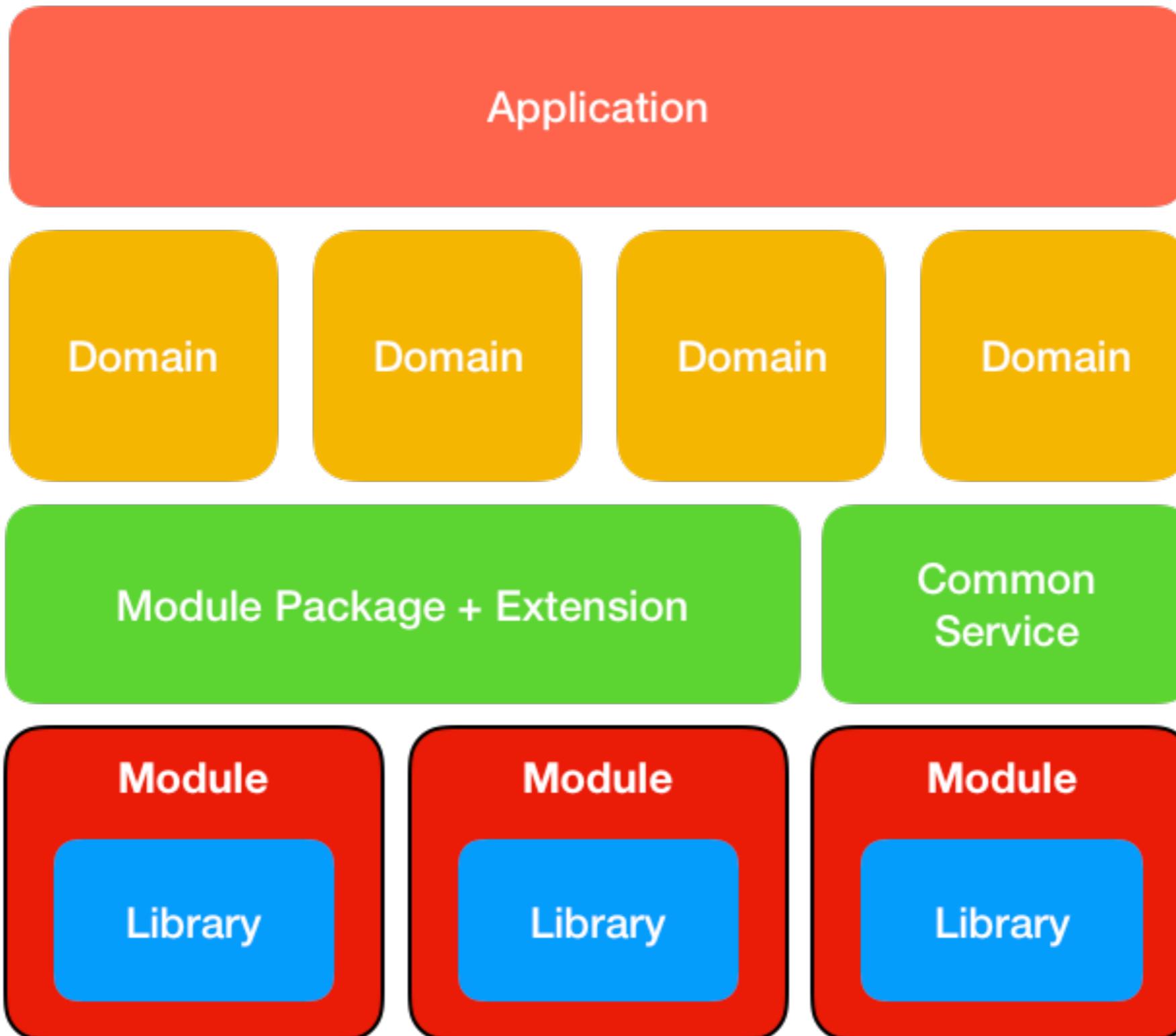
A.framework/A.swift

protocol 카드BaseAPI {}

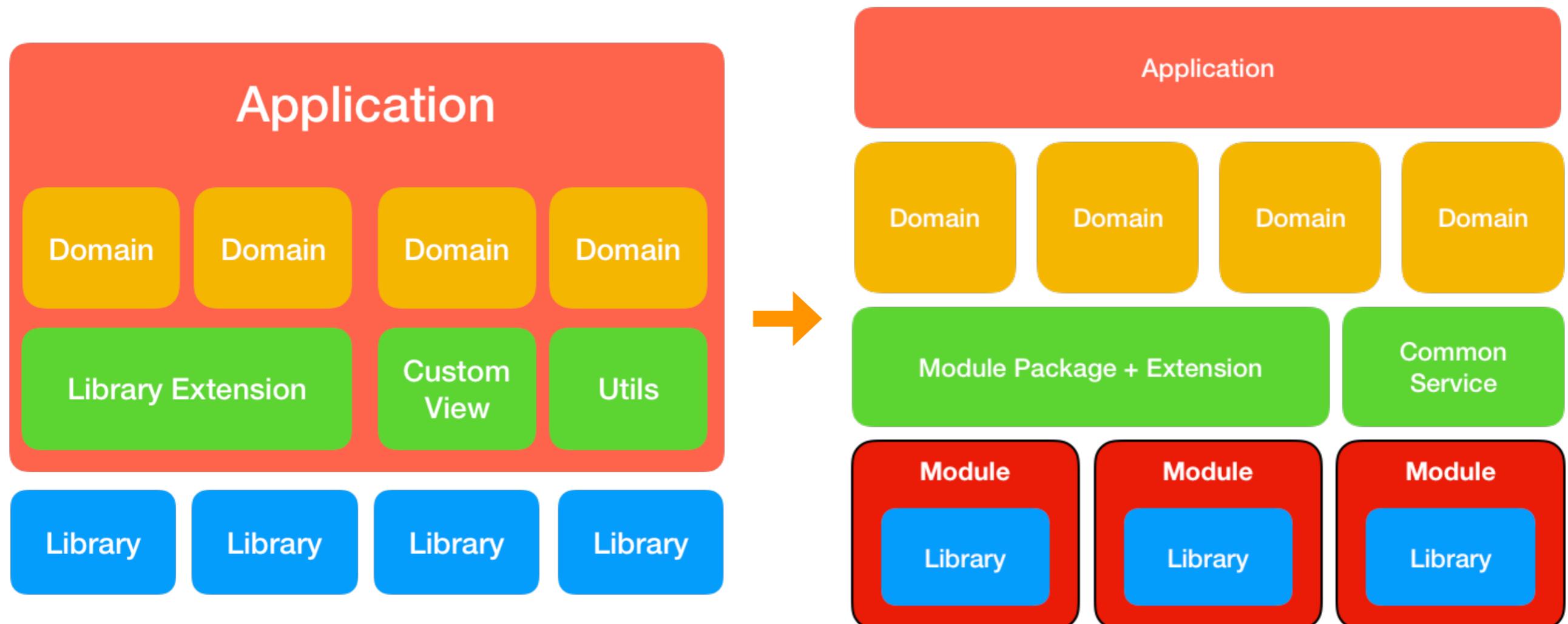
B.framework/B.swift

public class 대출API {}

프레임워크 주도 개발



프레임워크 주도 개발



프레임워크 주도 개발 - 장점

- 외부에 코드를 노출 여부를 결정 가능.
 - 노출 코드의 접근 수준은 **public, open**
 - 미노출 코드의 접근 수준은 **internal**
 - 다른 프레임워크와의 격리화
 - 응집도 증가, 결합도 감소, 부수효과 감소

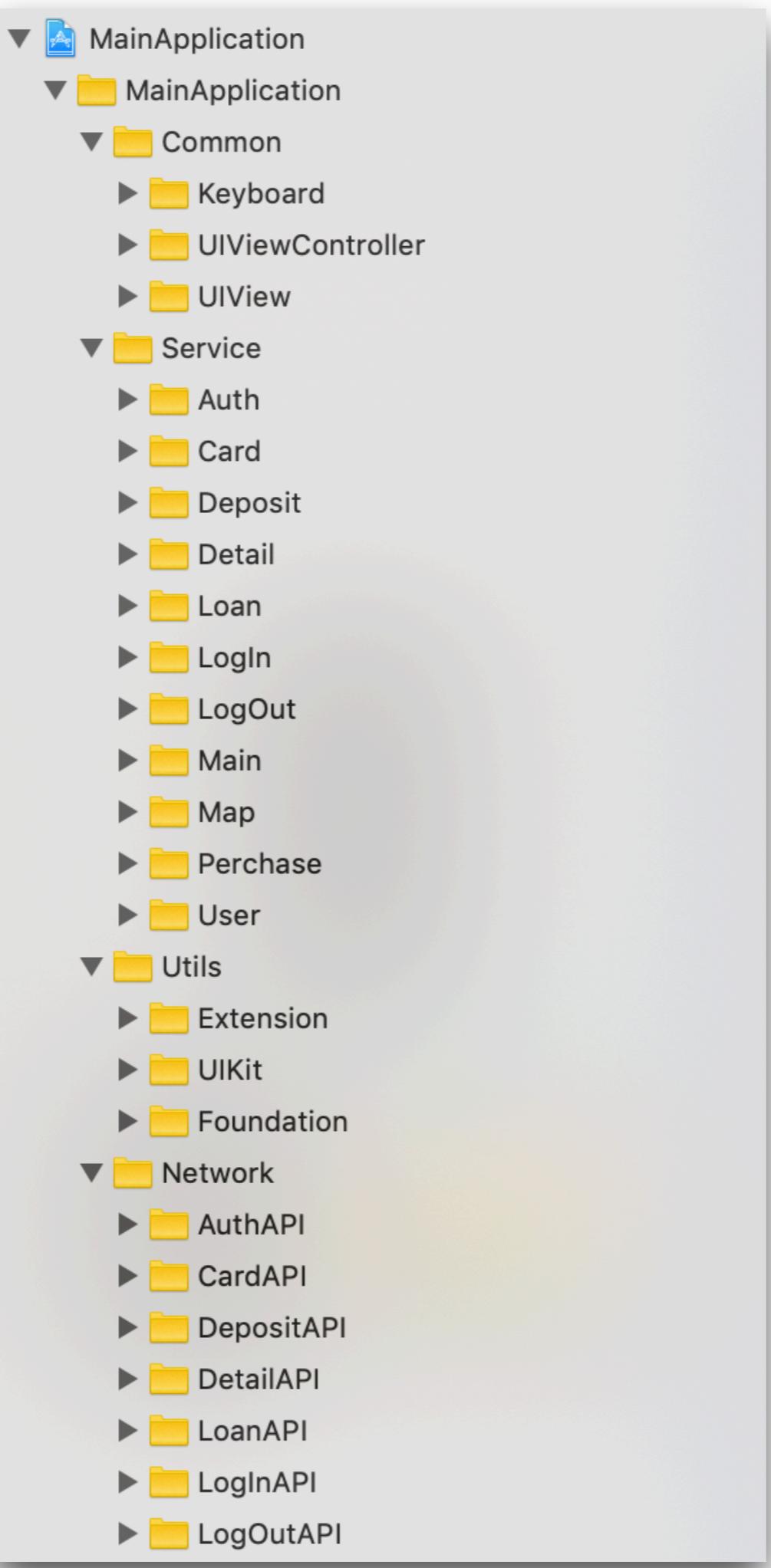
프레임워크 주도 개발 - 장점

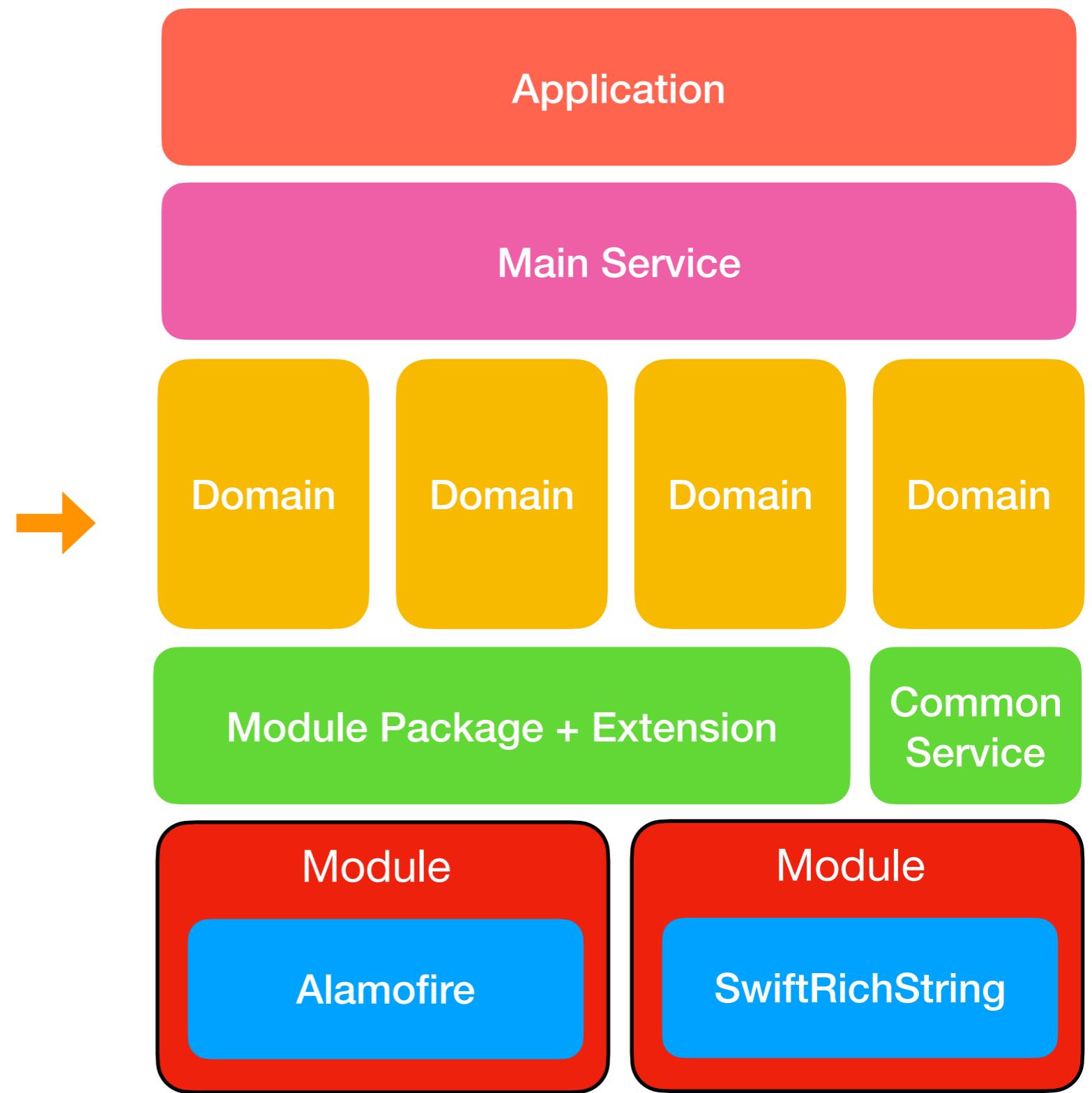
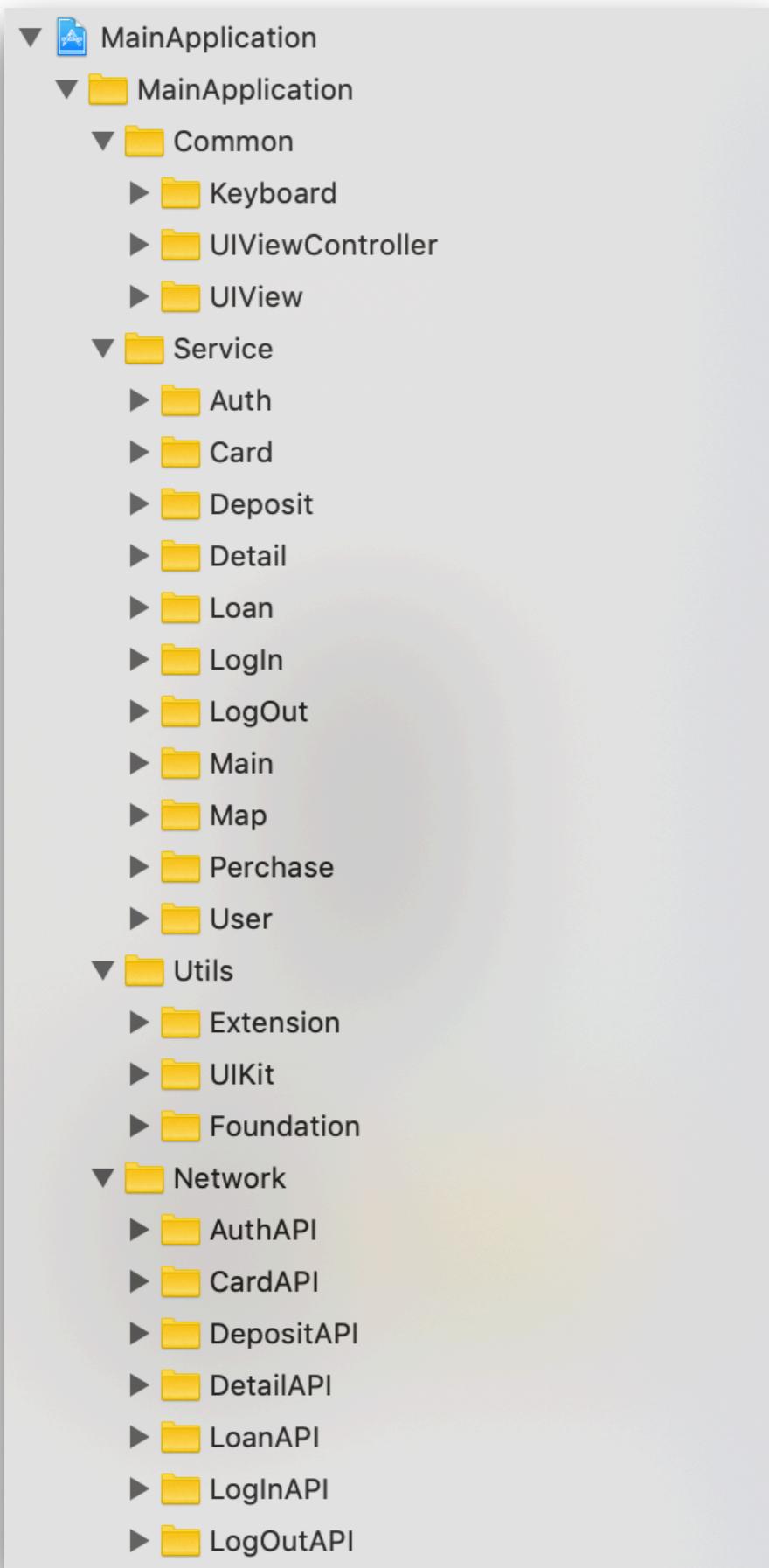
- 각 프레임워크에서 빠르고 많은 테스트를 수행
 - 메인 프로젝트에 좀 더 안정적으로 적용 가능.
- 메인 프로젝트의 빌드 속도가 비교적 빨라짐.
 - 프레임워크 빌드 후 메인 프로젝트를 빌드함.
- 협업시 충돌이 덜 발생함. - ex) 프로젝트 파일
 - **xUnique** 등 대체 방안은 존재.

프레임워크 주도 개발 - 단점

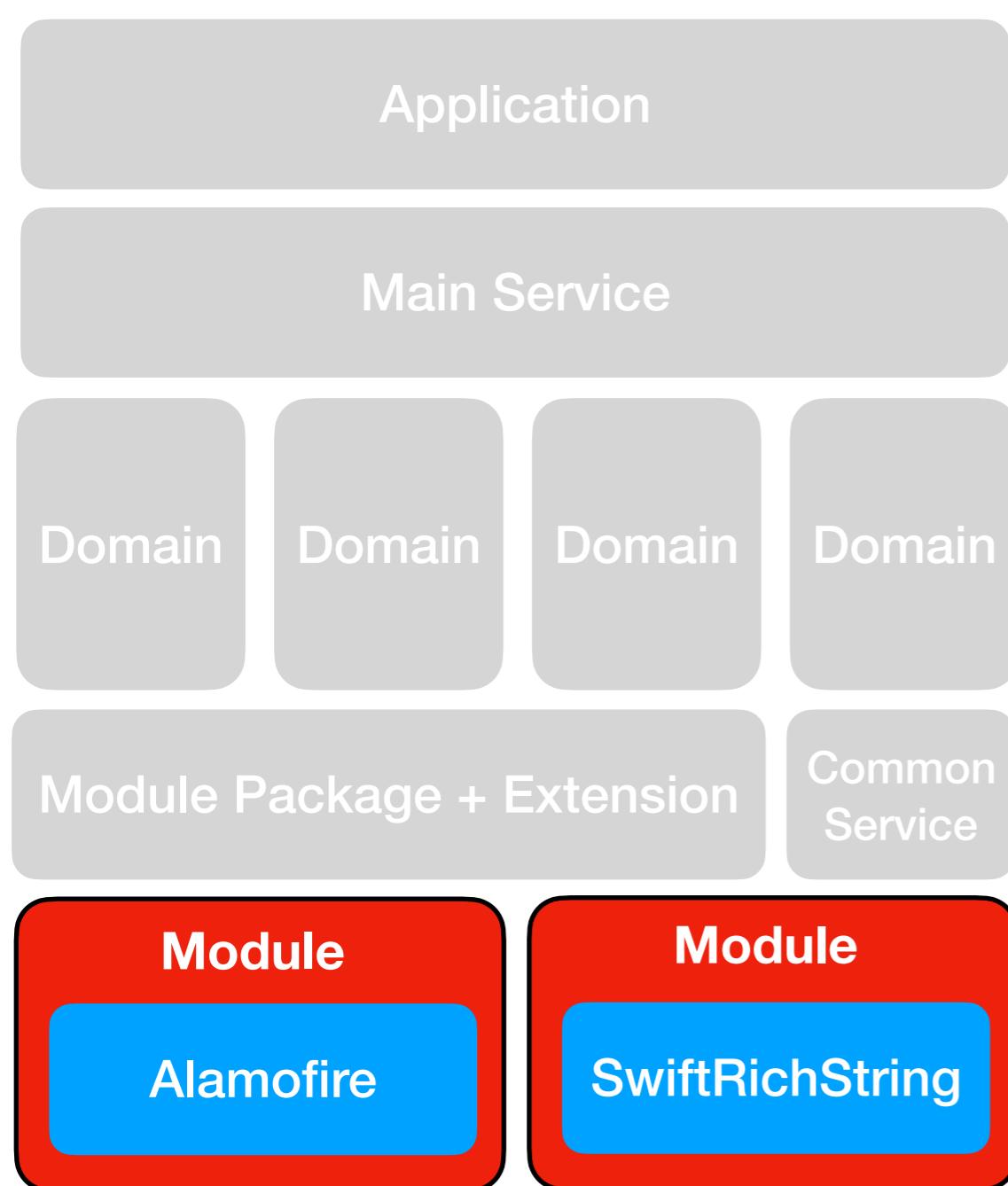
- 프레임워크를 만들어야 함.
- 기존 코드의 커플링을 제거하면서 리팩토링 해야함.
- 수많은 프레임워크가 생길 수 있음.
- 귀찮음. 귀찮음. 귀찮음. 귀찮음. 귀찮음.

프로젝트 재구성





Module



- 라이브러리를 가진 프로젝트
- 특정 역할(네트워크, 이미지 다운로드, 커스텀 UI 등)을 수행
- 외부에는 정의된 인터페이스를 통한 호출
- 다른 라이브러리로 교체가 가능.

Application

Main Service

Domain

Domain

Domain

Domain

Module Package + Extension

Common
Service

Module

Alamofire

Module

SwiftRichString

Module Package

- 여러 모듈을 관리 및 모듈간의 결합으로
기능 확장

Application

Main Service

Domain

Domain

Domain

Domain

Module Package + Extension

Common
Service

Module

Alamofire

Module

SwiftRichString

Common Service

- 인증, 보안, 네트워크 API 등 다른 서비스에서 공통으로 사용

Application

Main Service

Domain

Domain

Domain

Domain

Module Package + Extension

Common
Service

Module

Alamofire

Module

SwiftRichString

Domain Service

- 어플리케이션의 특정 도메인 역할을 담당
ex) 예적금개설, 카드 신청, 대출 신청 등

Application

Main Service

Domain

Domain

Domain

Domain

Module Package + Extension

Common
Service

Module

Alamofire

Module

SwiftRichString

Main Service

- 각 서비스를 호출 및 연결을 수행

Application

Main Service

Domain

Domain

Domain

Domain

Module Package + Extension

Common
Service

Module

Alamofire

Adapter

SwiftRichString

Application

- UIApplication(AppDelegate)에서 제공하는 기능을 받아서 처리

프로젝트 리팩토링

- Network

리팩토링 - Network

Application

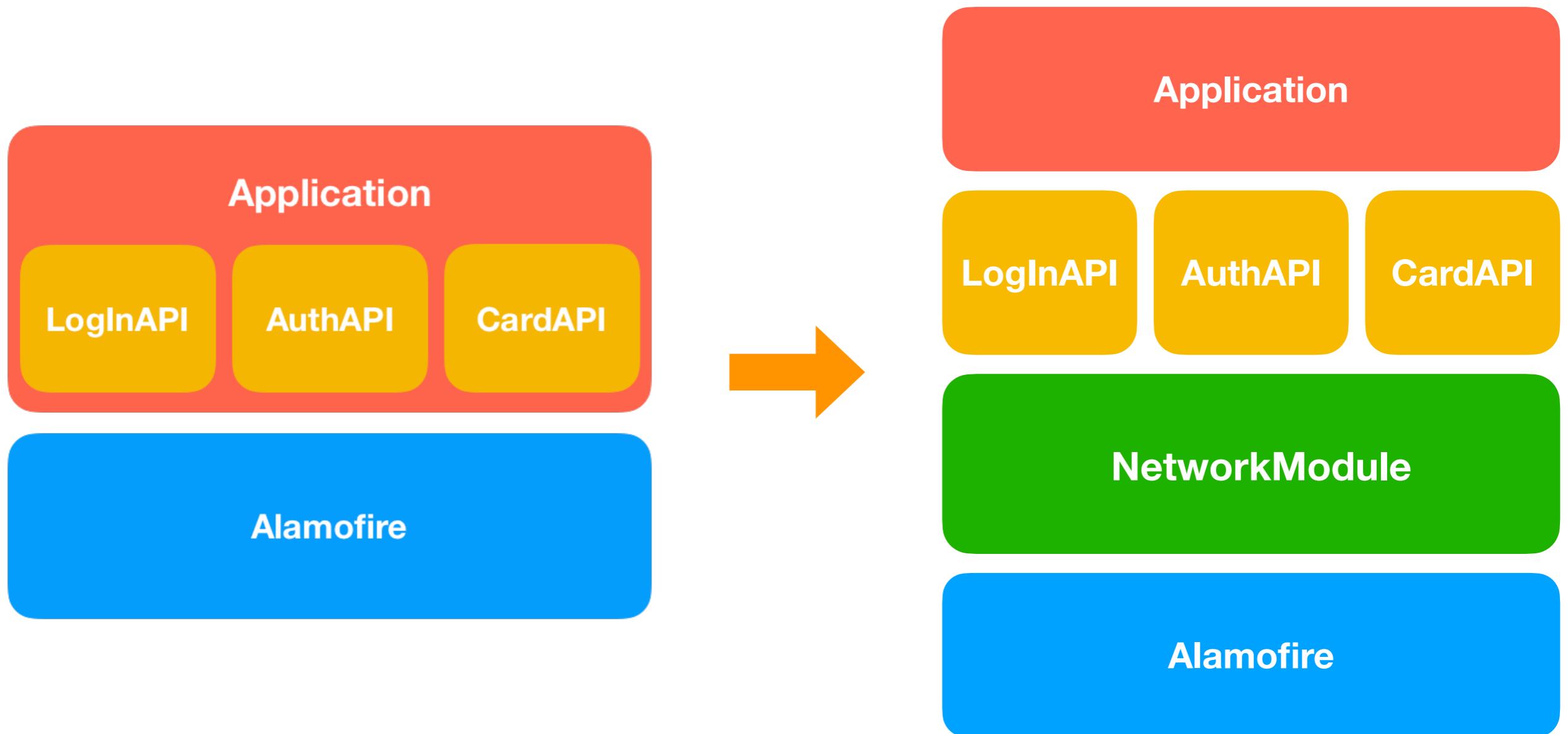
LogInAPI

AuthAPI

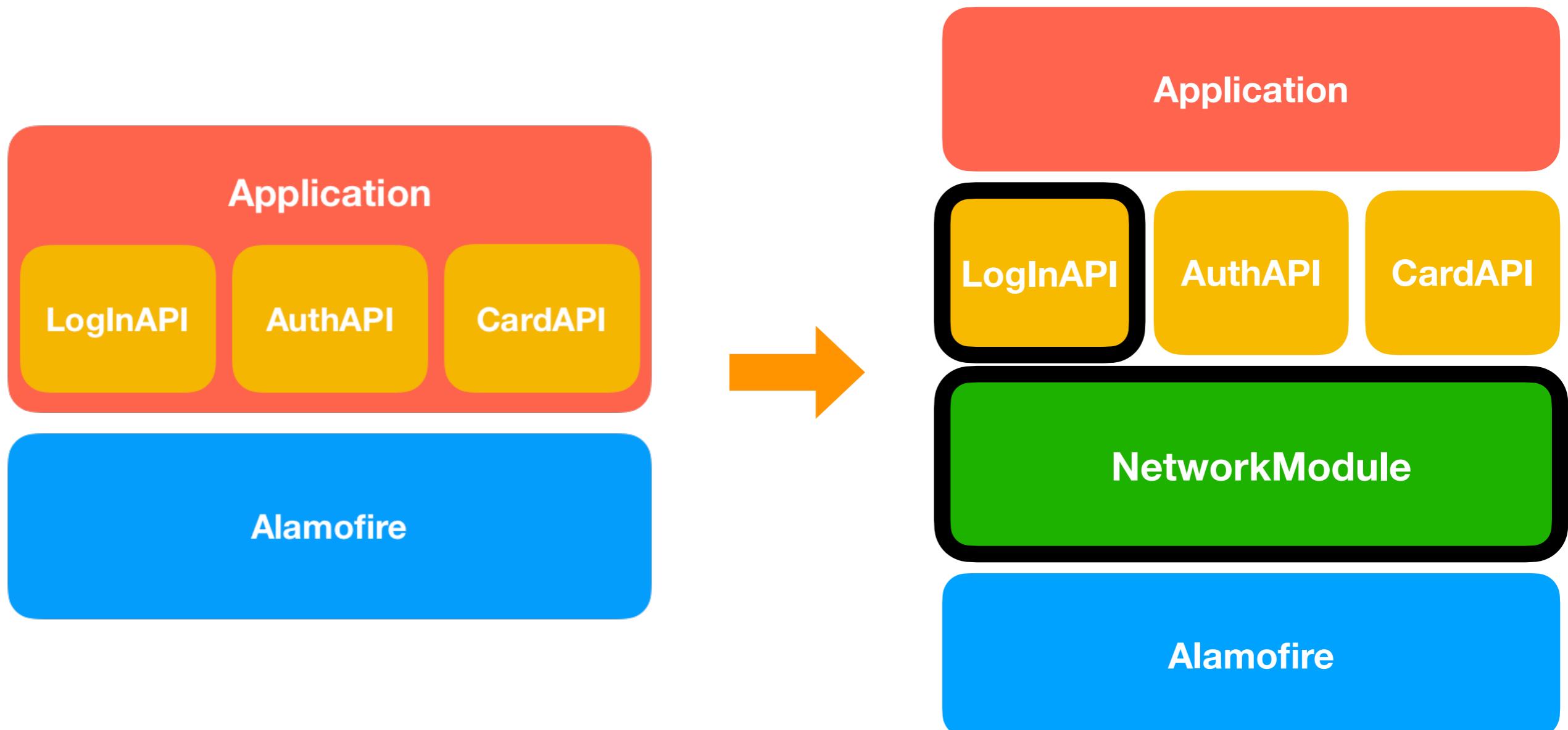
CardAPI

Alamofire

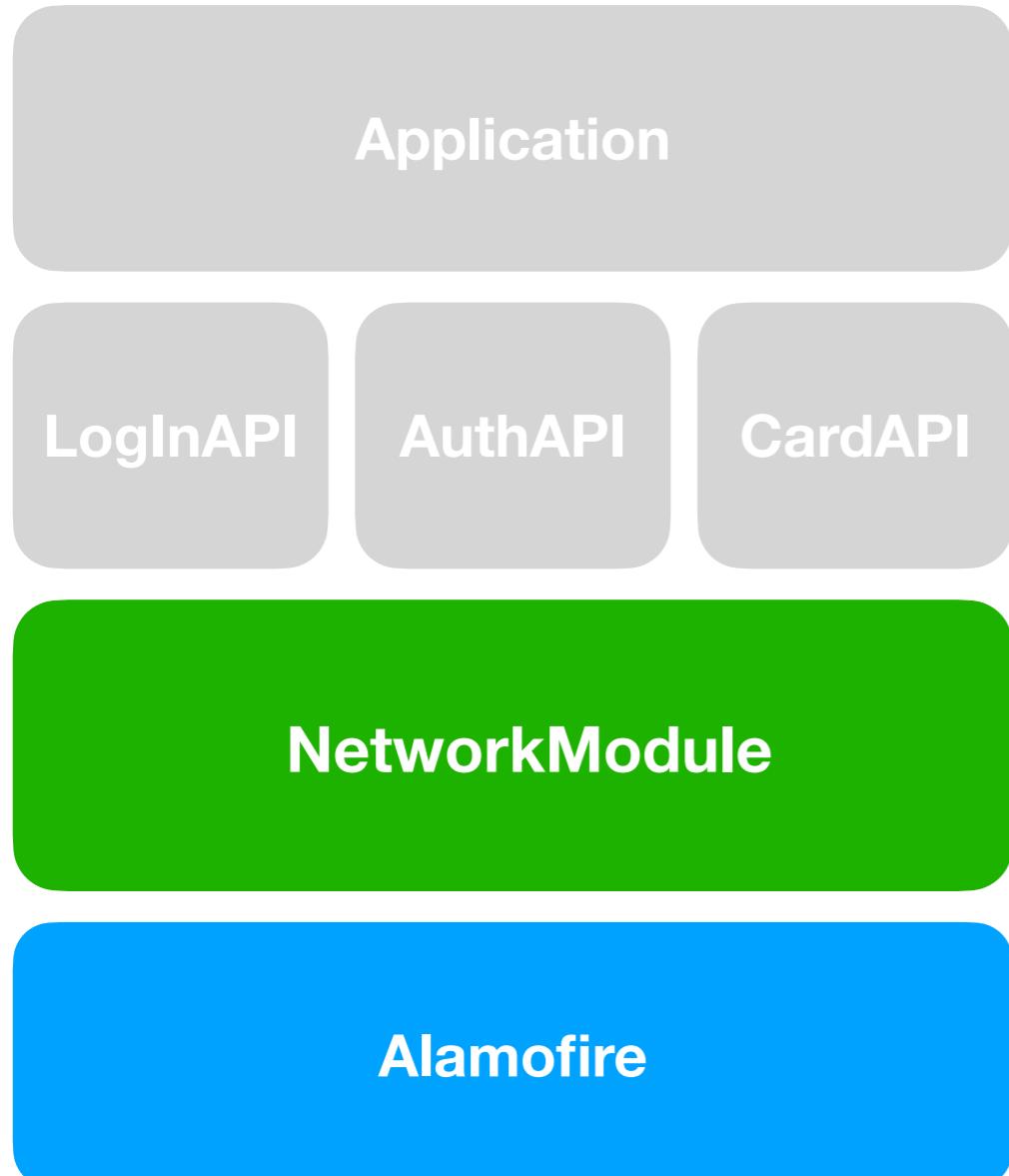
리팩토링 - Network



리팩토링 - Network



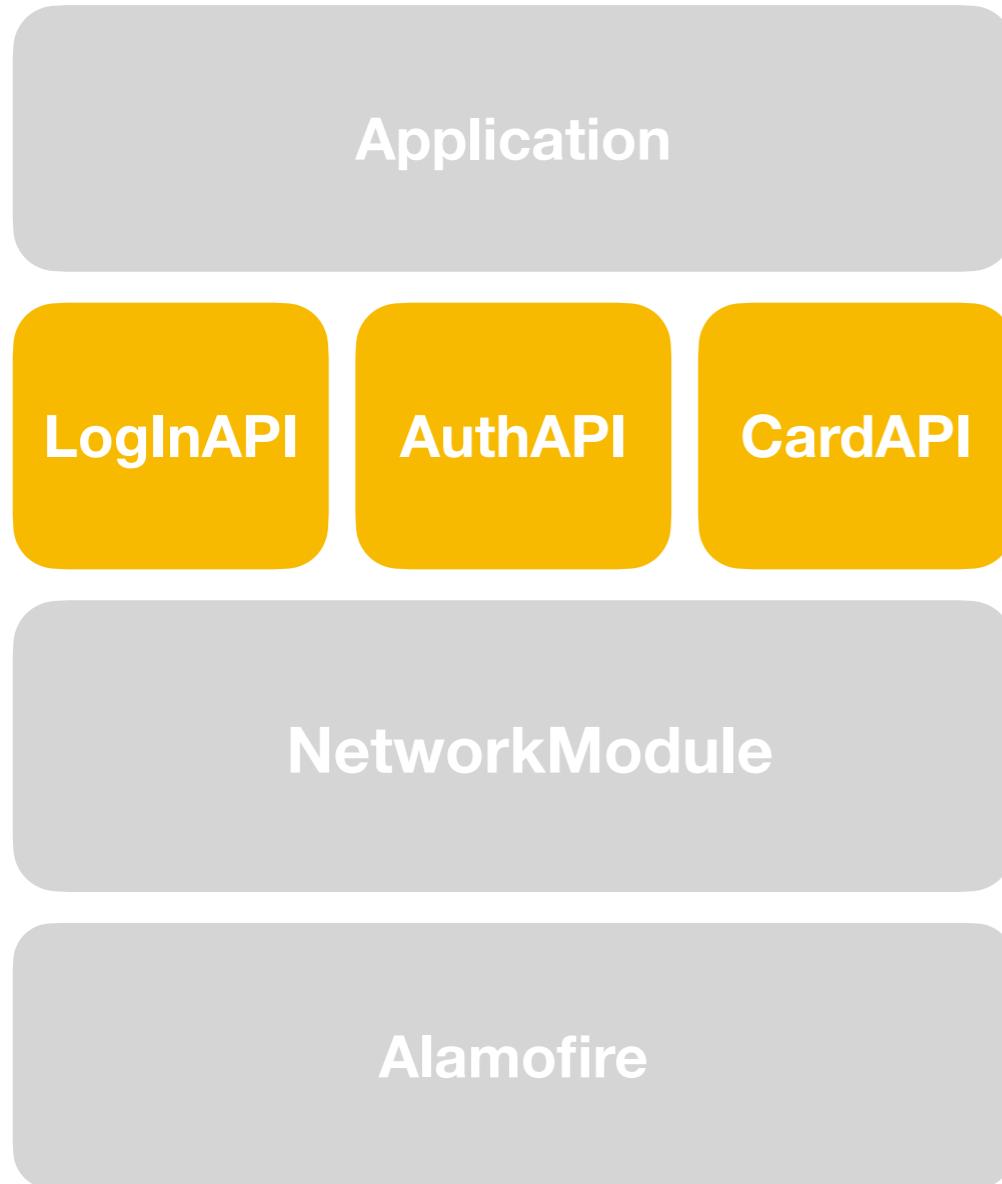
리팩토링 - Network



NetworkModule 프로젝트 생성

- 네트워크 라이브러리를 가진 프로젝트
- 클래스 또는 프로토콜을 이용해서 네트워크 요청을 기본 구현

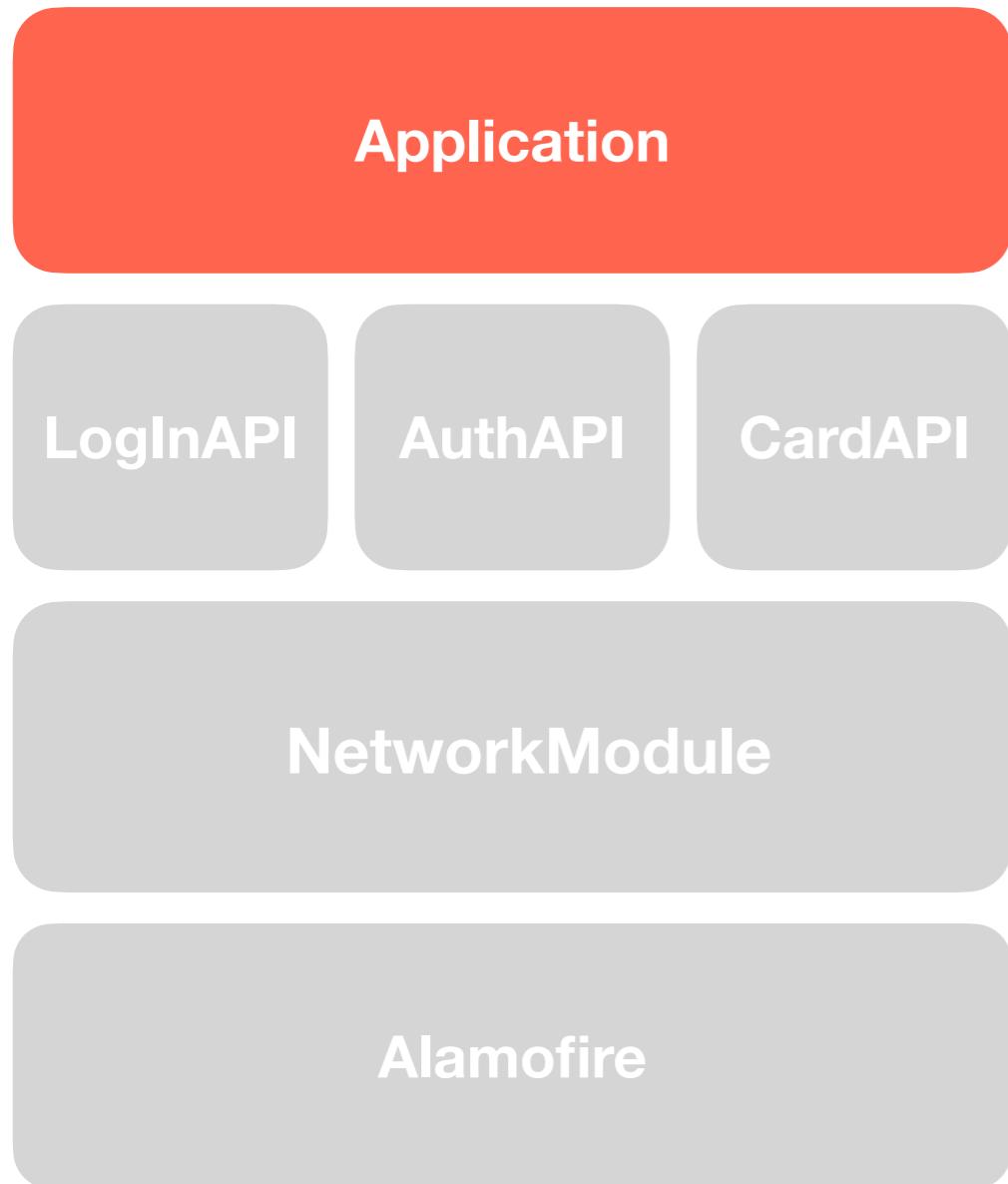
리팩토링 - Network



LogIn Network 프로젝트 생성

- NetworkModule 라이브러리를 가진 프로젝트
- LogIn, Logout API를 구현

리팩토링 - Network

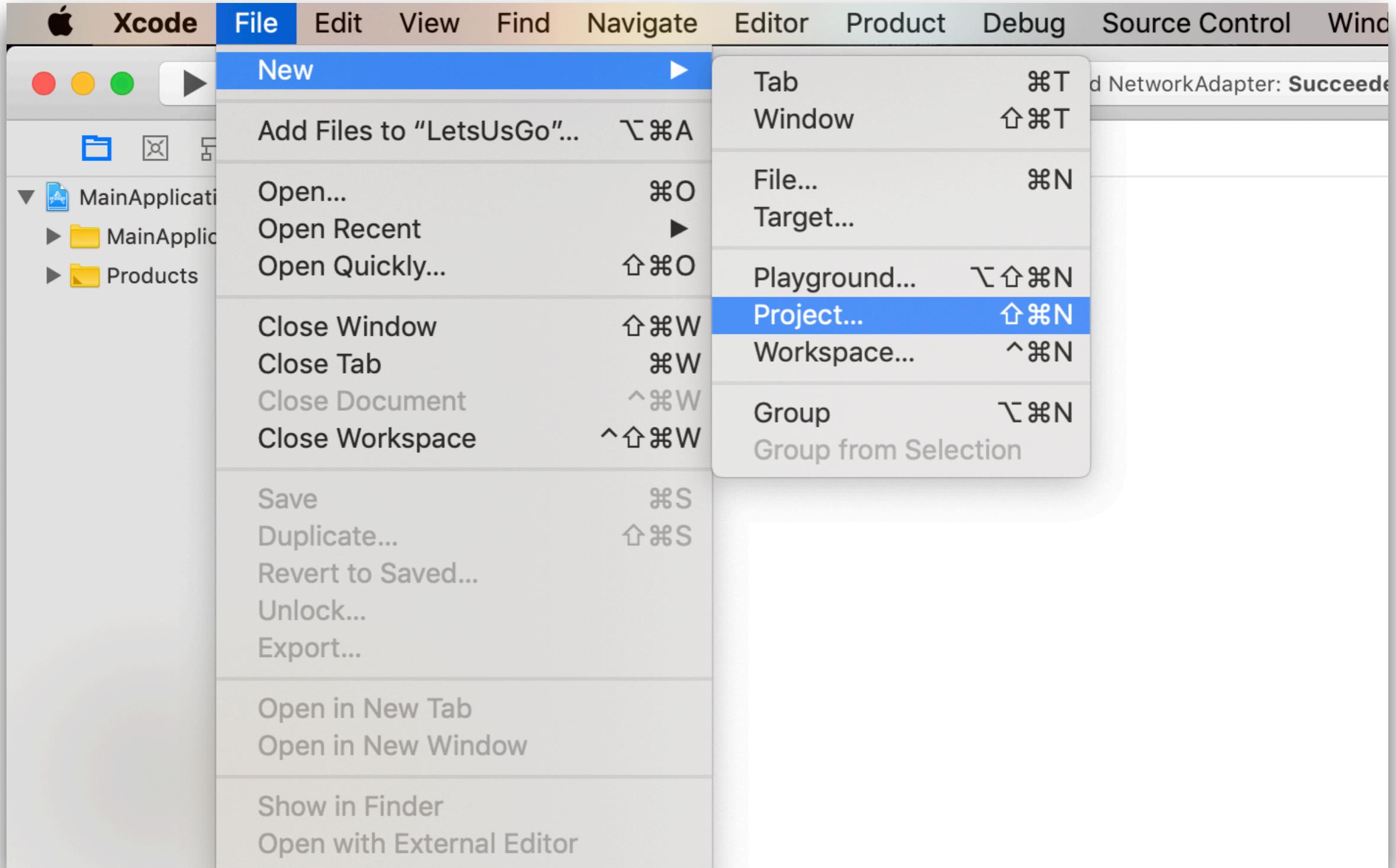


Application에서 LogInAPI 요청

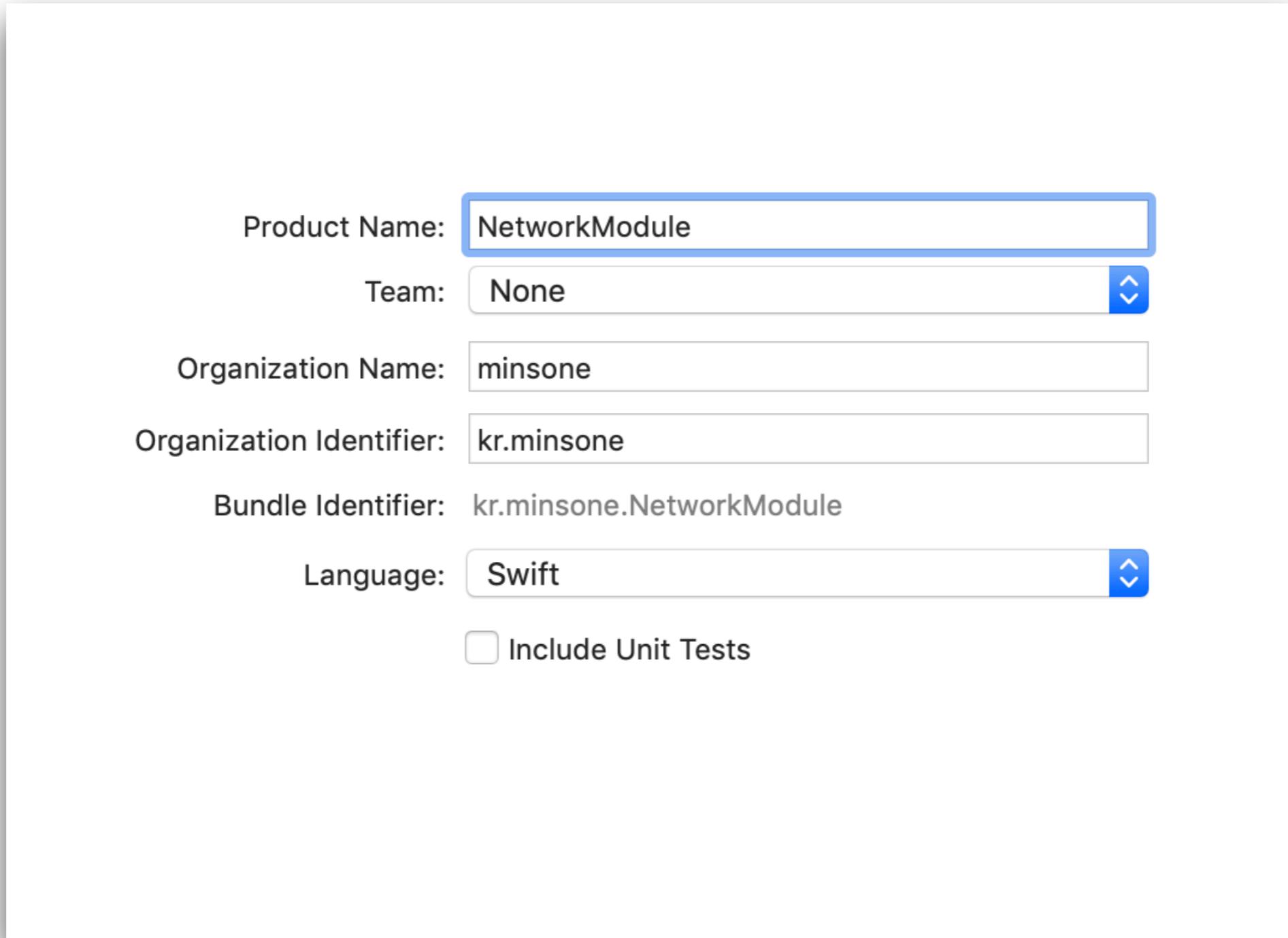
리팩토링 - Network

- NetworkModule 프로젝트 생성
 - 네트워크 라이브러리를 가진 프로젝트
 - 클래스 또는 프로토콜을 이용해서 네트워크 요청을 기본 구현
- LogIn Network 프로젝트 생성
 - NetworkModule 라이브러리를 가진 프로젝트
 - LogIn, Logout API를 구현
- 메인 프로젝트에서 LogIn, Logout API를 요청

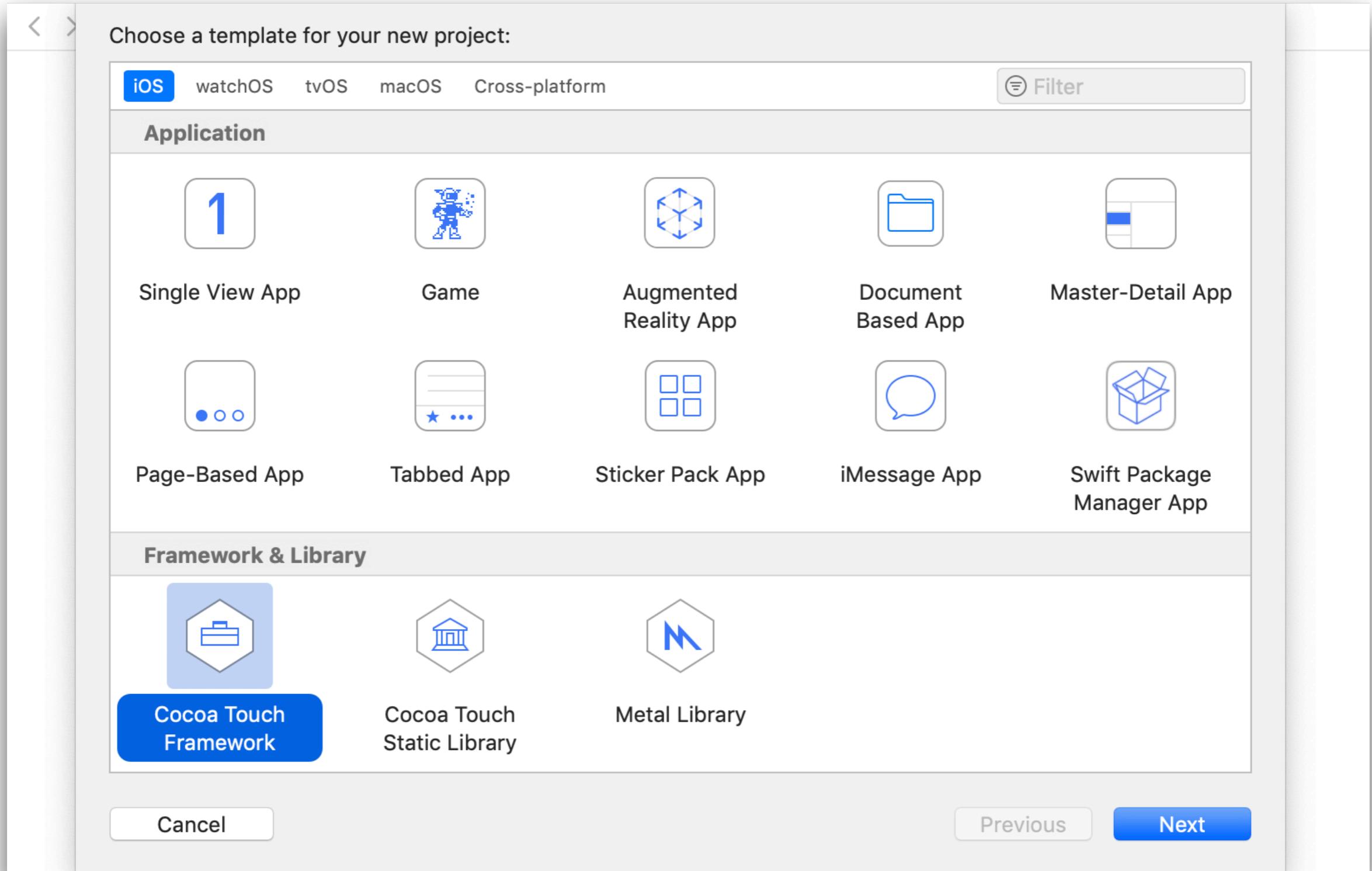
리팩토링 - Network



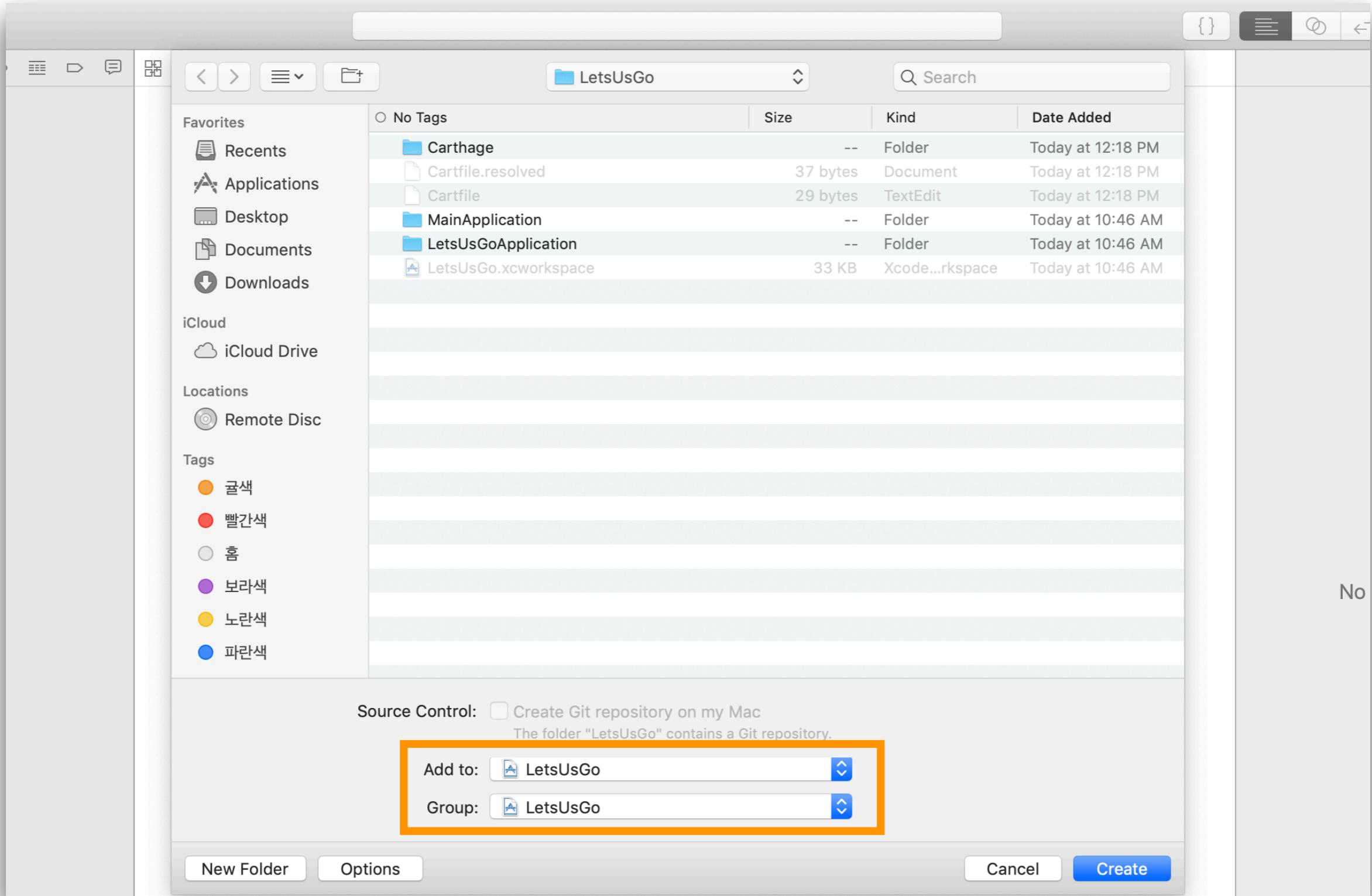
리팩토링 - Network



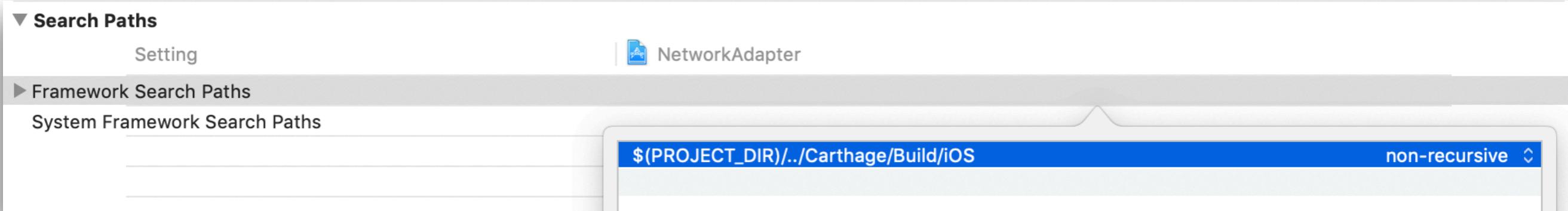
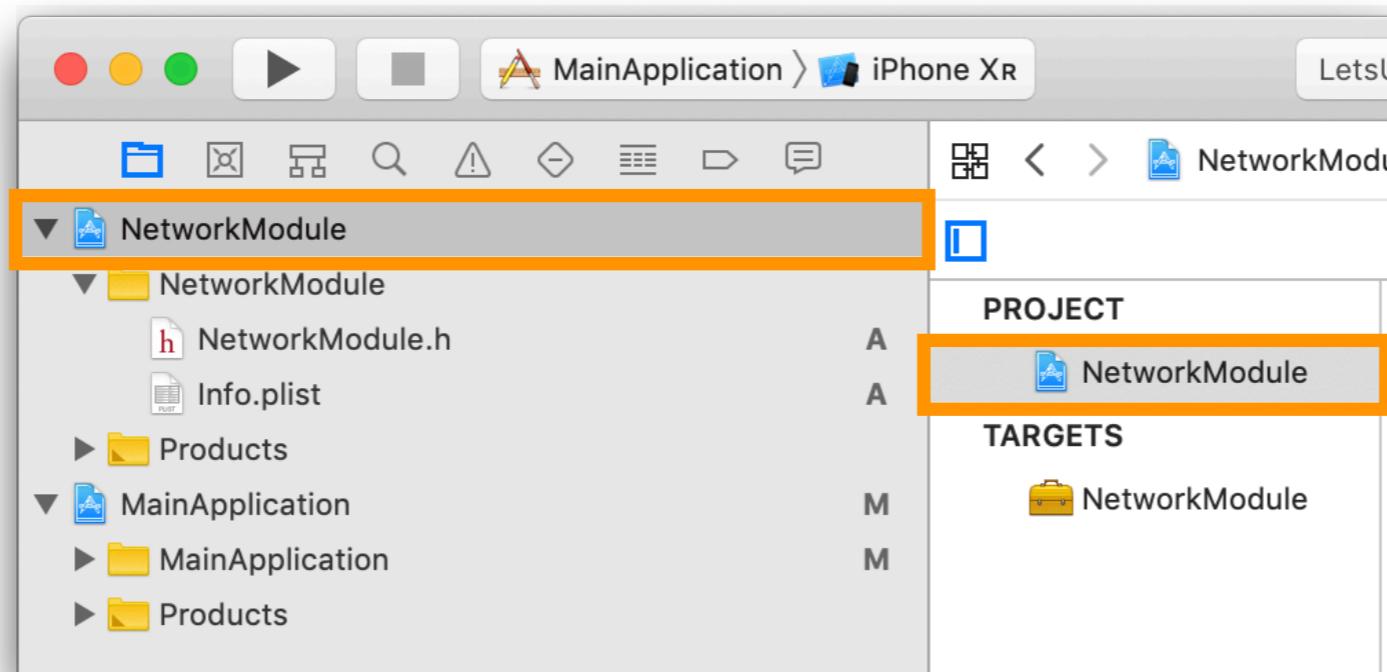
리팩토링 - Network



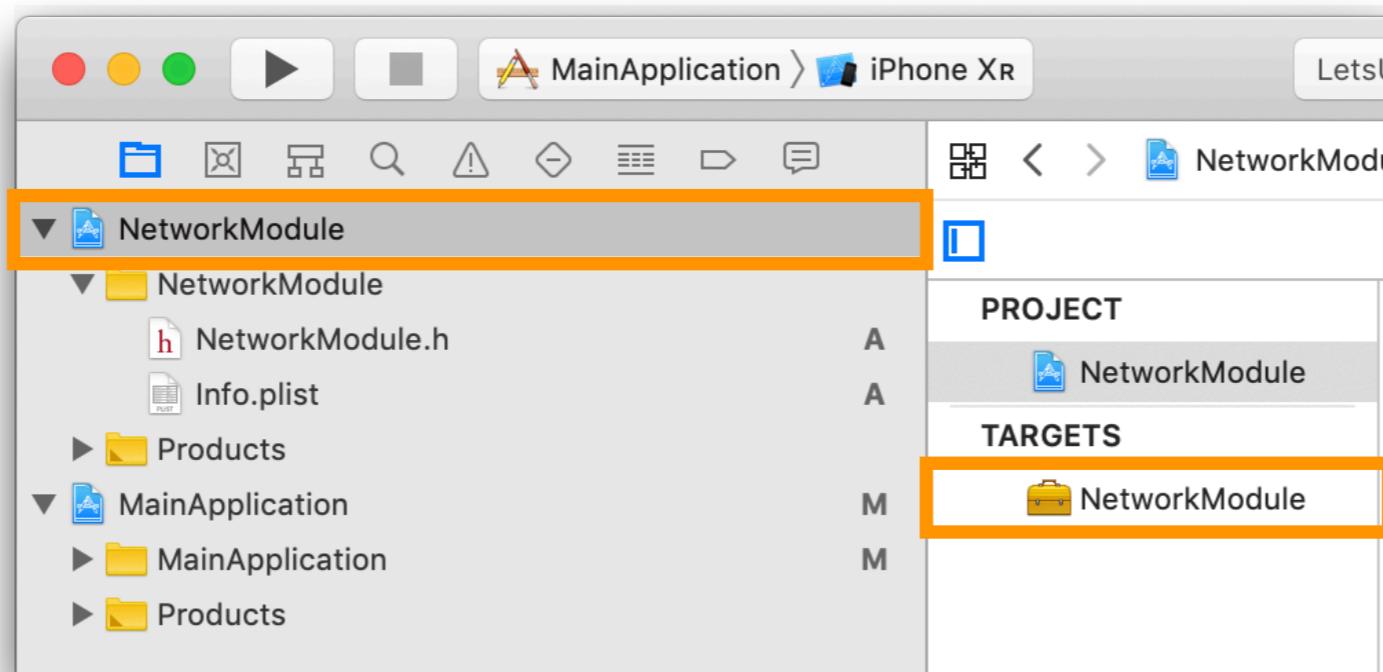
리팩토링 - Network



리팩토링 - Network



리팩토링 - Network

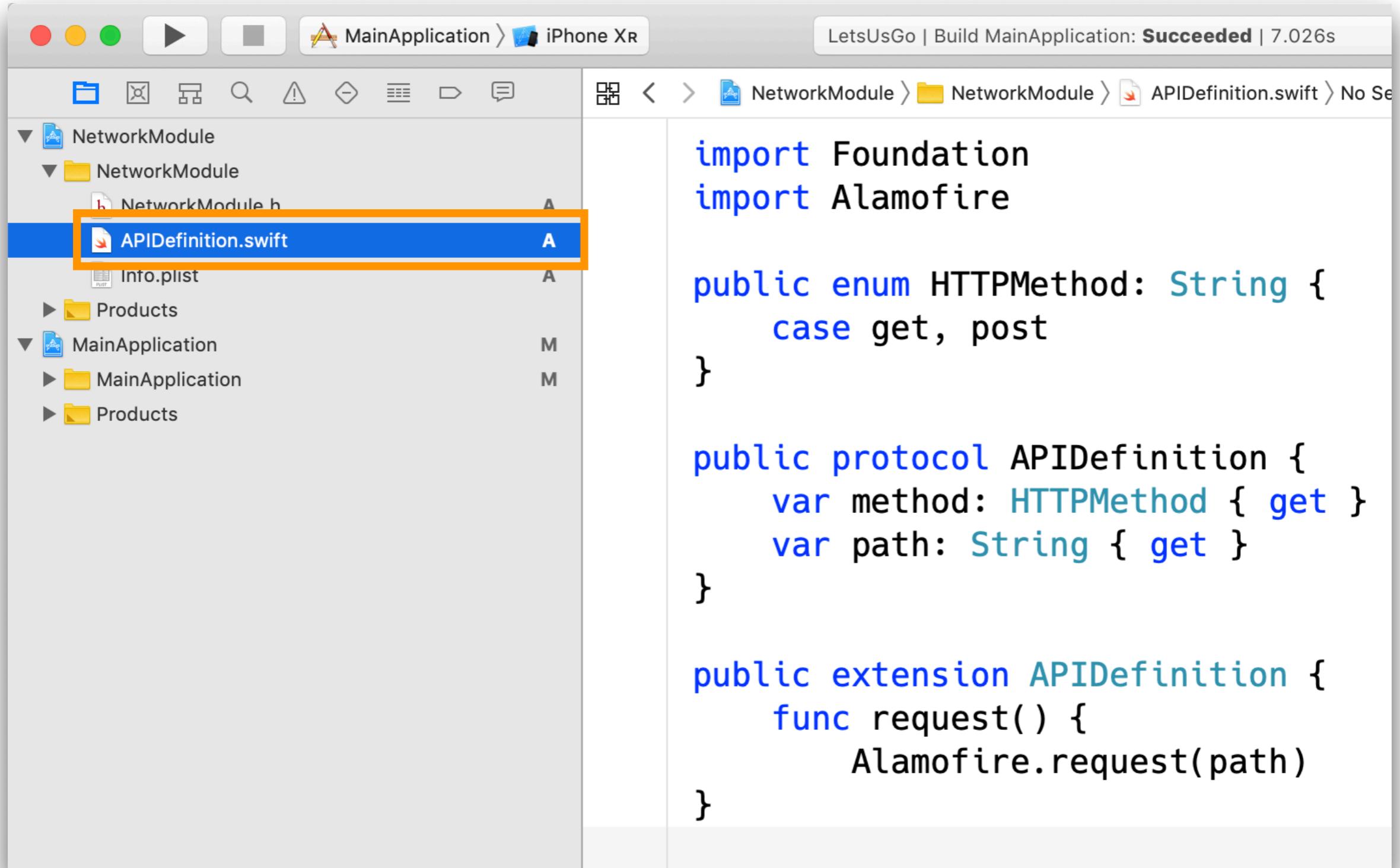


Linked Frameworks and Libraries

| Name | Status |
|---------------------|----------|
| Alamofire.framework | Required |

+ -

리팩토링 - Network



The screenshot shows the Xcode interface with the project navigation bar at the top. The main area displays the code for `APIDefinition.swift` in the `NetworkModule` folder. The file is highlighted with a blue selection bar. The code defines an enum `HTTPMethod`, a protocol `APIDefinition`, and an extension `APIDefinition` for `Alamofire.request`.

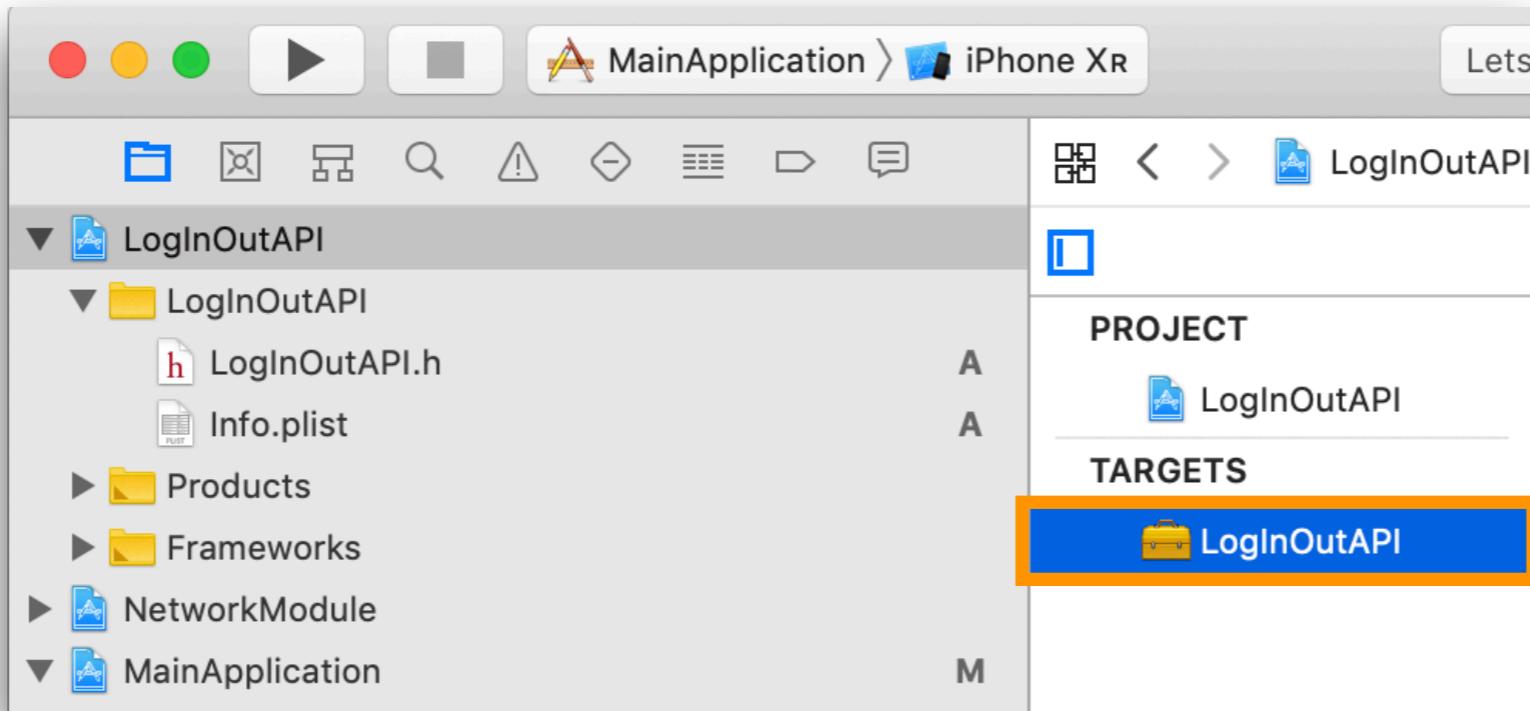
```
import Foundation
import Alamofire

public enum HTTPMethod: String {
    case get, post
}

public protocol APIDefinition {
    var method: HTTPMethod { get }
    var path: String { get }
}

public extension APIDefinition {
    func request() {
        Alamofire.request(path)
    }
}
```

리팩토링 - Network

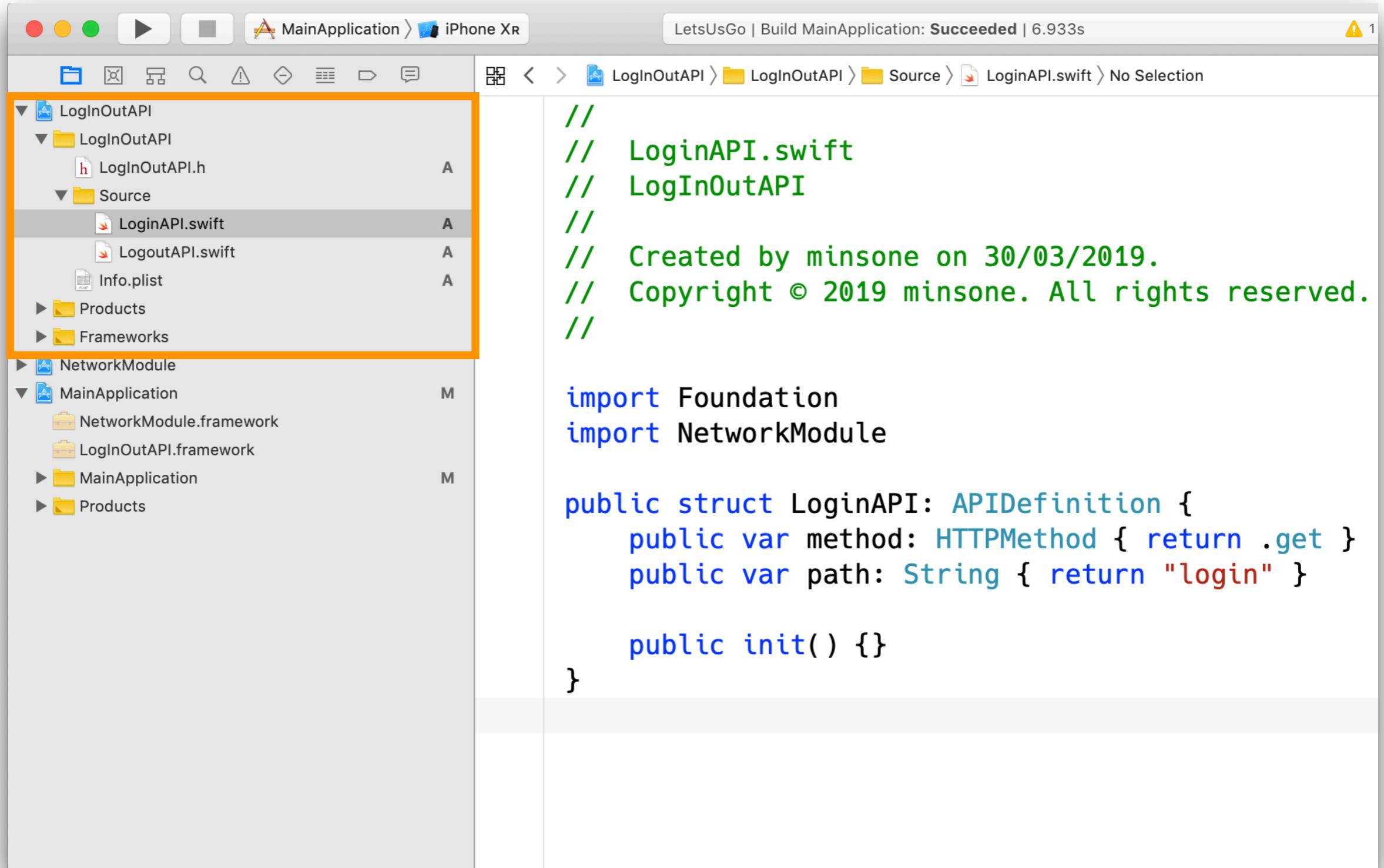


The screenshot shows the 'Linked Frameworks and Libraries' section in the Xcode settings. A table lists the linked frameworks:

| Name | Status |
|-------------------------|----------|
| NetworkModule.framework | Required |

Below the table are '+' and '-' buttons for managing framework links.

리팩토링 - Network



The screenshot shows the Xcode interface with the project navigation bar at the top. The project name is "MainApplication" and the target is "iPhone XR". The status bar indicates "LetsUsGo | Build MainApplication: Succeeded | 6.933s" and has a warning icon.

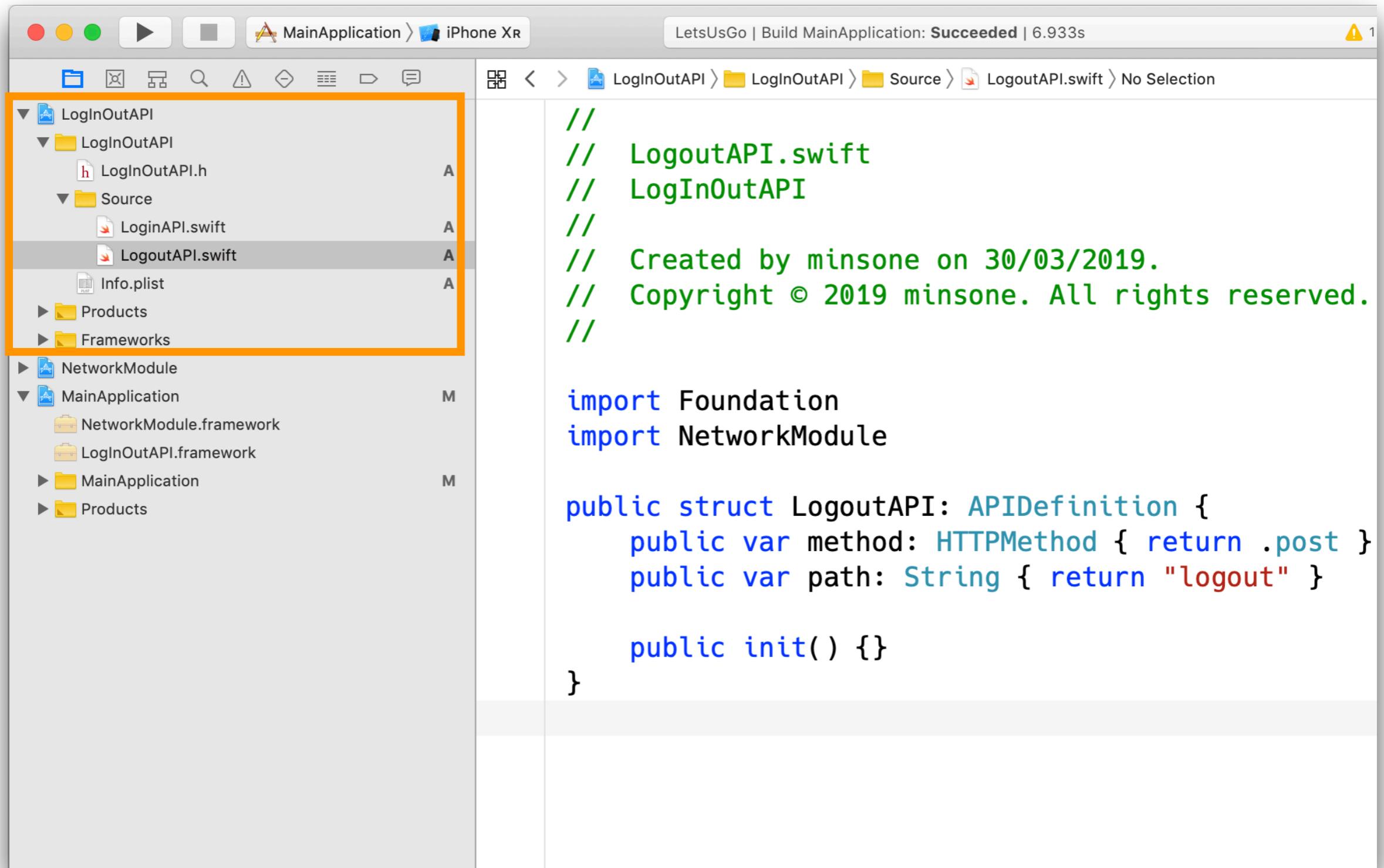
The left sidebar displays the project structure:

- LogInOutAPI (selected)
- ↳ LogInOutAPI (group)
- ↳ LogInOutAPI.h
- ↳ Source (group)
- ↳ LoginAPI.swift (selected)
- ↳ LogoutAPI.swift
- ↳ Info.plist
- ↳ Products
- ↳ Frameworks

The right pane shows the code editor with the file "LoginAPI.swift" open. The code is as follows:

```
//  
//  LoginAPI.swift  
//  LogInOutAPI  
//  
//  Created by minsone on 30/03/2019.  
//  Copyright © 2019 minsone. All rights reserved.  
  
import Foundation  
import NetworkModule  
  
public struct LoginAPI: APIDefinition {  
    public var method: HTTPMethod { return .get }  
    public var path: String { return "login" }  
  
    public init() {}  
}
```

리팩토링 - Network

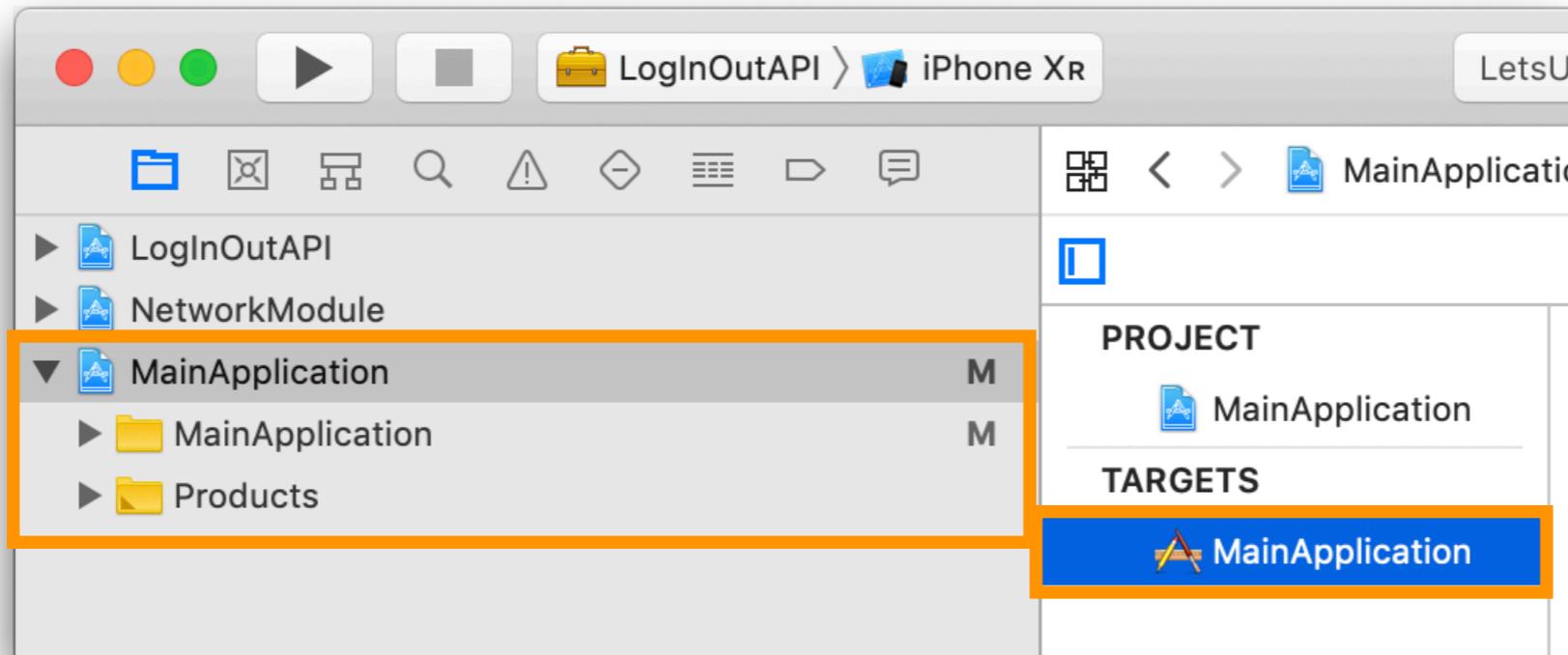


The screenshot shows the Xcode interface with the project navigation bar at the top. The main area displays the file structure of the 'LogoutAPI' module and its contents. A specific file, 'LogoutAPI.swift', is selected and shown in the editor. The code within this file is as follows:

```
//  
// LogoutAPI.swift  
// LogInOutAPI  
//  
// Created by minsone on 30/03/2019.  
// Copyright © 2019 minsone. All rights reserved.  
  
import Foundation  
import NetworkModule  
  
public struct LogoutAPI: APIDefinition {  
    public var method: HTTPMethod { return .post }  
    public var path: String { return "logout" }  
  
    public init() {}  
}
```

The file structure on the left side of the interface is highlighted with an orange box, specifically around the 'LogoutAPI' module and its 'Source' folder.

리팩토링 - Network



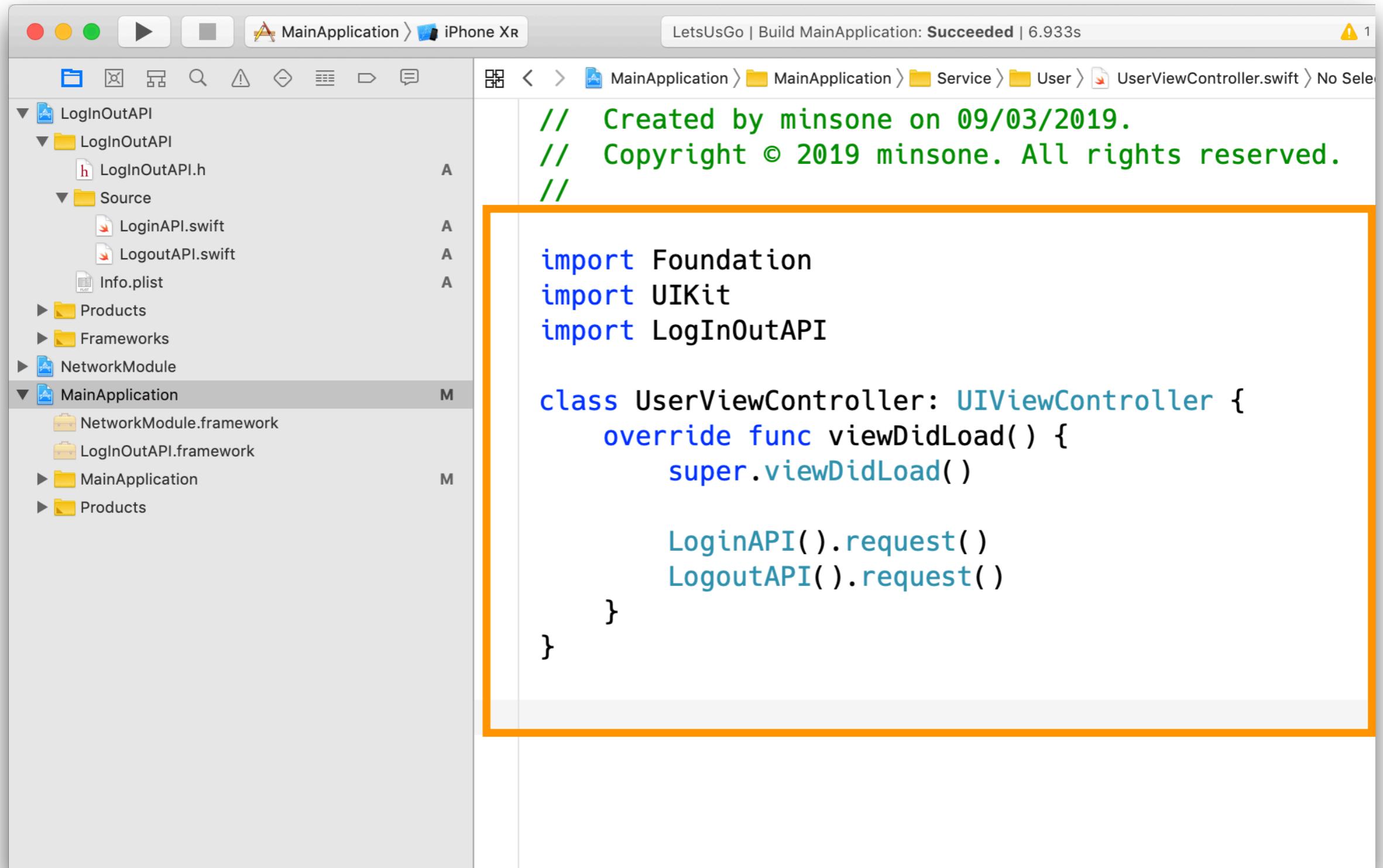
▼ Embedded Binaries

LogInOutAPI.framework ...in build/Debug-iphoneos

NetworkModule.framework ...in build/Debug-iphoneos

+

리팩토링 - Network



LetsUsGo | Build MainApplication: **Succeeded** | 6.933s

1

```
// Created by minsone on 09/03/2019.  
// Copyright © 2019 minsone. All rights reserved.  
  
import Foundation  
import UIKit  
import LogInOutAPI  
  
class UserViewController: UIViewController {  
    override func viewDidLoad() {  
        super.viewDidLoad()  
  
        LoginAPI().request()  
        LogoutAPI().request()  
    }  
}
```

리팩토링

- 리팩토링으로 얻은 것

리팩토링

- 리팩토링으로 얻은 것
 - 메인 프로젝트는 Alamofire를 알 필요가 없어짐.
 - 언제든 Alamofire를 다른 라이브러리로 교체 가능.

리팩토링

- 리팩토링으로 얻은 것
 - 메인 프로젝트는 Alamofire를 알 필요가 없어짐.
 - 각각의 네트워크 API를 프레임워크로 만듦.
 - 다른 API를 알 이유가 없어짐. 도메인 기반으로 만들어 응집도 증가.

리팩토링

- 리팩토링으로 얻은 것
 - 메인 프로젝트는 Alamofire를 알 필요가 없어짐.
 - 각각의 네트워크 API를 프레임워크로 만듦.
 - WatchOS 등 앱 확장시 필요한 프레임워크만 사용.
 - 중복코드가 상대적으로 줄어듬.

참고자료

참고자료

- Cocoa Layered Architecture
- Paper - A Framework for iOS Application Development
- 쿠팡 개발 블로그
 - 안드로이드 애플리케이션 모듈화
 - 리팩토징을 통한 의존성 제거

QnA

카카오뱅크 iOS 개발자 구인중!