3rd PARTY DEPENDENCY

Swift BCA Training

LESSON 5

Dependency Manager

Dependency managers perform some handy functions, including:

- Simplifying and standardizing the process of fetching third-party code and incorporating it into your project
- Making it easier to update third-party libraries in the future
- Selecting appropriate and compatible versions of each dependency you use

IOS Dependency Manager

- Cocoapods
- Carthage
- Swift Package Manager (SPM)

Cocoapods

- CocoaPods require a Ruby environment
- CocoaPods brings a lot of simplifications into dependencies management:
  - Dependencies are managed from a single place, it is called **Podfile**
  - Some of the dependencies have their own dependencies. CocoaPods take care of figuring that out for you.
  - Updating dependencies is much simpler. Just bump version name in a **Podfile** and run **pod install** on your terminal

How to Use Cocoapods

After you create a project, here are steps to use Cocoapods:

- Open your terminal
- If you haven't installed Cocoapods, you can install Cocoapods via this command

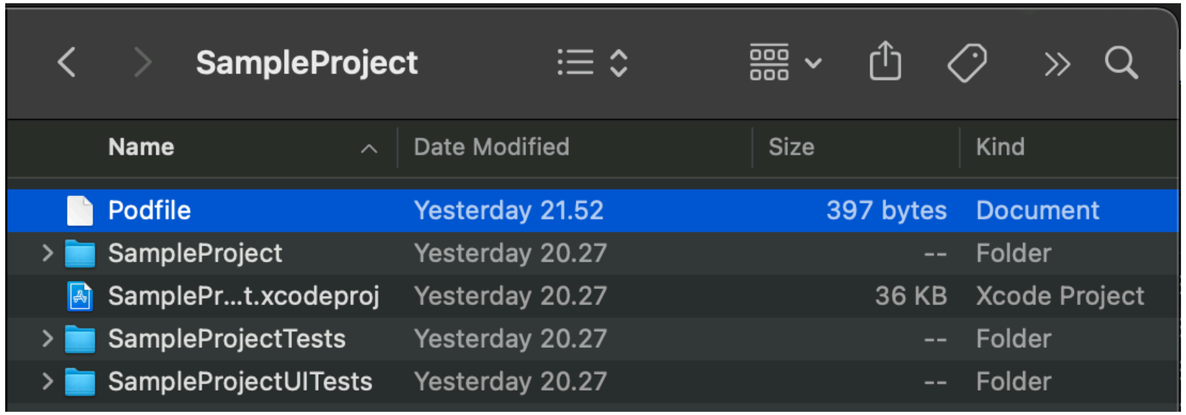
```
sudo gem install cocoapods
```

- Change your location into your project directory

```
cd <#yourProjectFilePath#>
```

- Run command **pod init** to create Podfile

How to Use Cocoapods



Podfile

```
# Uncomment the next line to define a global platform for your project
# platform :ios, '9.0'

target 'SampleProject' do
  # Comment the next line if you don't want to use dynamic frameworks
  use_frameworks!

  # Pods for SampleProject

  target 'SampleProjectTests' do
    inherit! :search_paths
    # Pods for testing
  end

  target 'SampleProjectUITests' do
    # Pods for testing
  end

end
```

How to Use Cocoapods

- Add your dependency on selected target, for example

```
pod 'Alamofire'
```

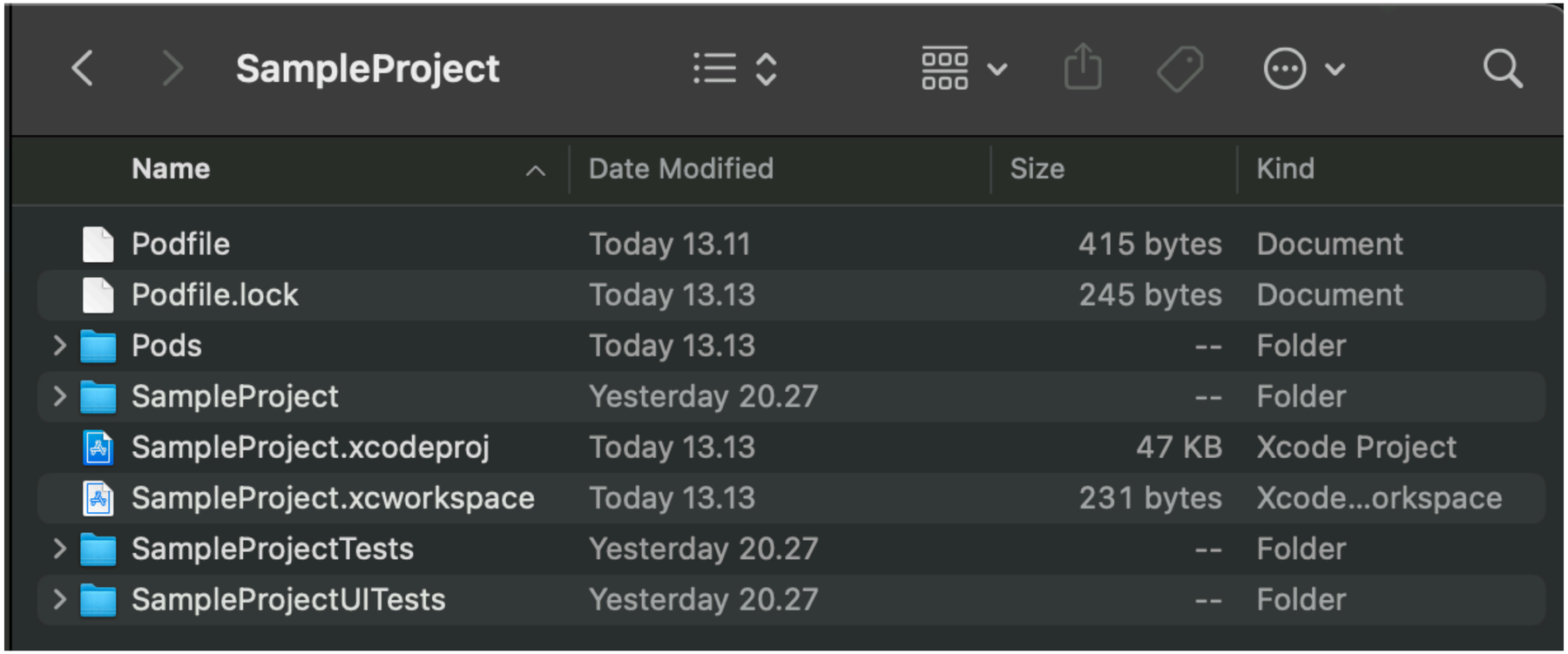
→ Get latest version

```
pod 'Alamofire', '5.0.0'
```

→ Get specific version

- When you done add dependency, run command **pod install** and your downloaded dependency is ready to use
- After it success, open **.xcworkspace** file to use your downloaded dependency on your project

How to Use Cocoapods



How to Use Cocoapods

SampleProject — -bash — 77×22

```
NeosKnight:SampleProject hashfi$ pod install
Analyzing dependencies
Downloading dependencies
Installing Alamofire (5.7.1)
Generating Pods project
Integrating client project

[!] Please close any current Xcode sessions and use `SampleProject.xcworkspac
e` for this project from now on.
Pod installation complete! There is 1 dependency from the Podfile and 1 total
 pod installed.
```

Let’s Practice!

Swift Package Manager

- SPM has been part of Swift since it was open sourced, but it took until Xcode 11 for SPM to get integrated in the Xcode IDE.
- The things which SPM makes better in the context of our project are:
  - Has better support for the internal frameworks
  - Because of Apple's official support, hopefully it have no more issues when moving to a newer version of Xcode
  - Don't require any additional installation
  - Fewer steps to perform in Terminal to resolve all dependencies

Adding Package Dependency

- In Xcode, go to **File → Add Packages...** OR select your project in the **Project Editor**, go to the **Package Dependencies** tab, and press the + **(plus)**
- Enter a Package URL (e.g. a GitHub repository URL) or a search term in the search field in the upper right.
- Select the package you want to add.
- Select a **Dependency Rule**. In most cases, you probably want to set this to **Up to Next Major Version**
- Click **Add Package**.
- **Done!** 🚀 Yes, it's that simple.

Recently Used 1

Collections

Apple Swift Packages 9

Source Control Accounts

GitHub 122

Recently Used
Found 1 result

https://github.com/gonzalezreal/Markdown...

MarkdownUI

MarkdownUI github.com/gonzalezreal/MarkdownUI

Dependency Rule Up to Next Major Version 1.0.0 < 2.0.0

Add to Project Playground

MarkdownUI

CI passing | Swift Compatibility 5.5 | 5.4

Platform Compatibility iOS | macOS | tvOS | watchOS

contact @gonzalezreal

MarkdownUI is a Swift package for rendering Markdown in SwiftUI, fully compliant with the CommonMark Spec.

Supported Platforms

You can use MarkdownUI in the following platforms:

- macOS 11.0+
- iOS 14.0+
- tvOS 14.0+

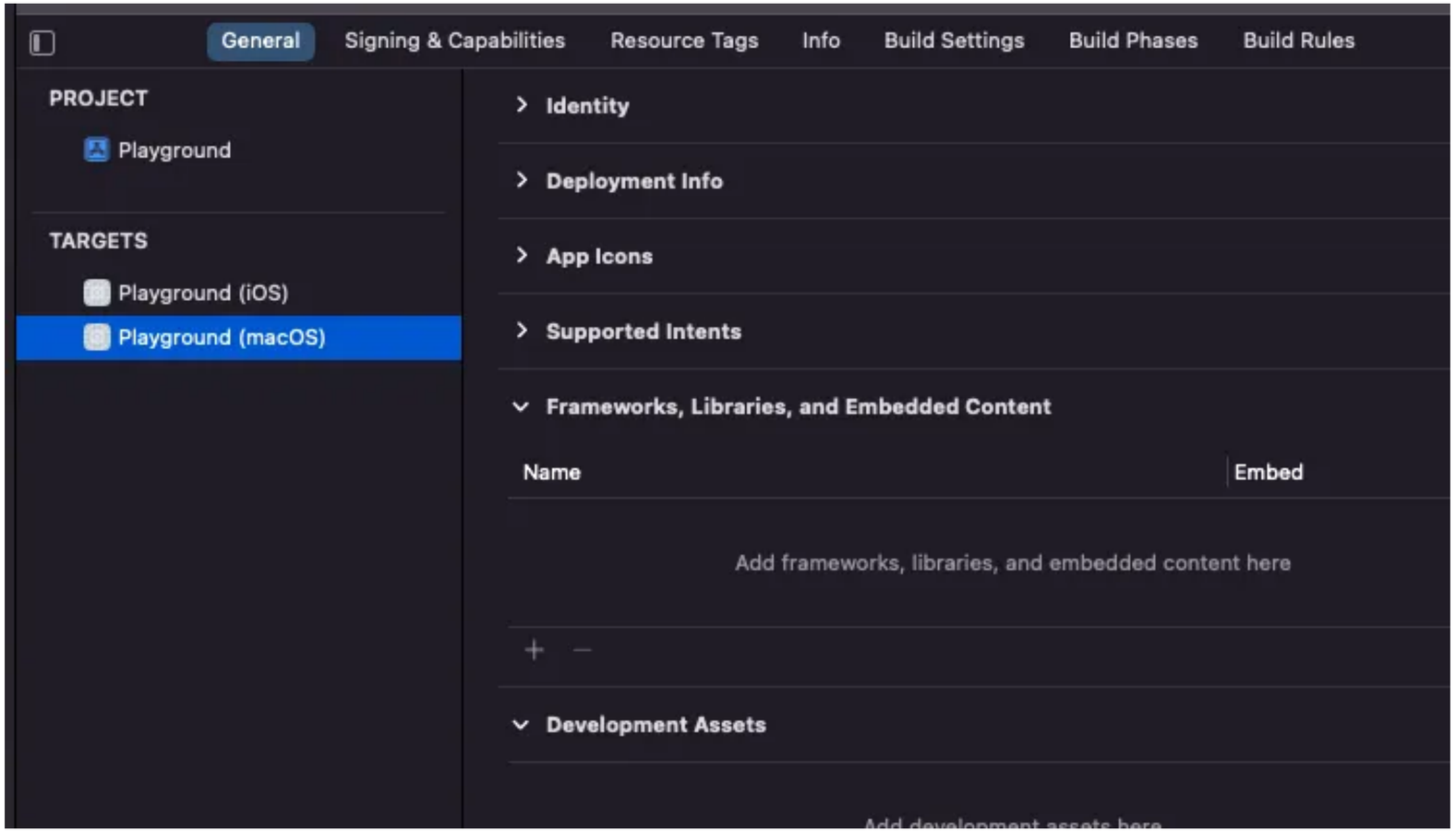
Add Local... Cancel Add Package

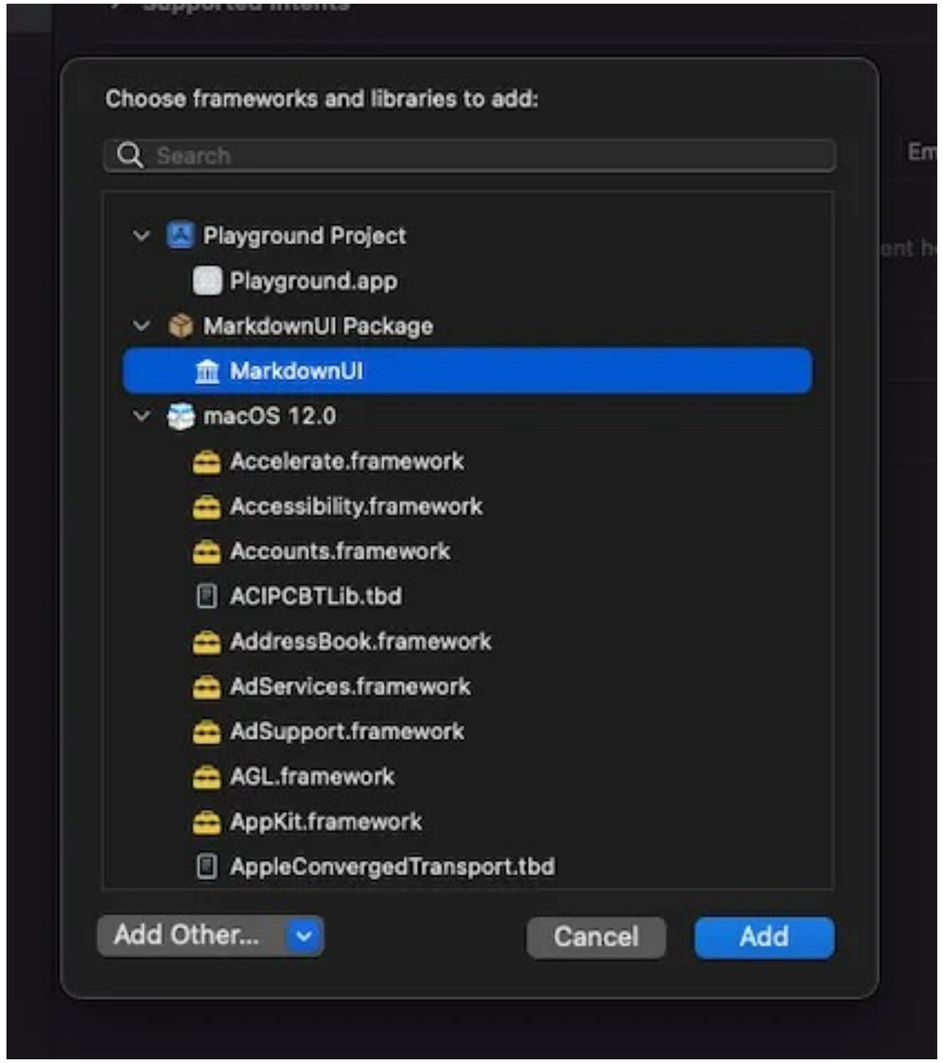
Adding Package Dependency To Multiple Targets

- Select the additional target you want to add your package to in the **Project Editor**
- In the **General** tab, scroll down to **Frameworks, Libraries, and Embedded Content**
- Click the + **(plus)**, select your package in the list, and click **Add**.

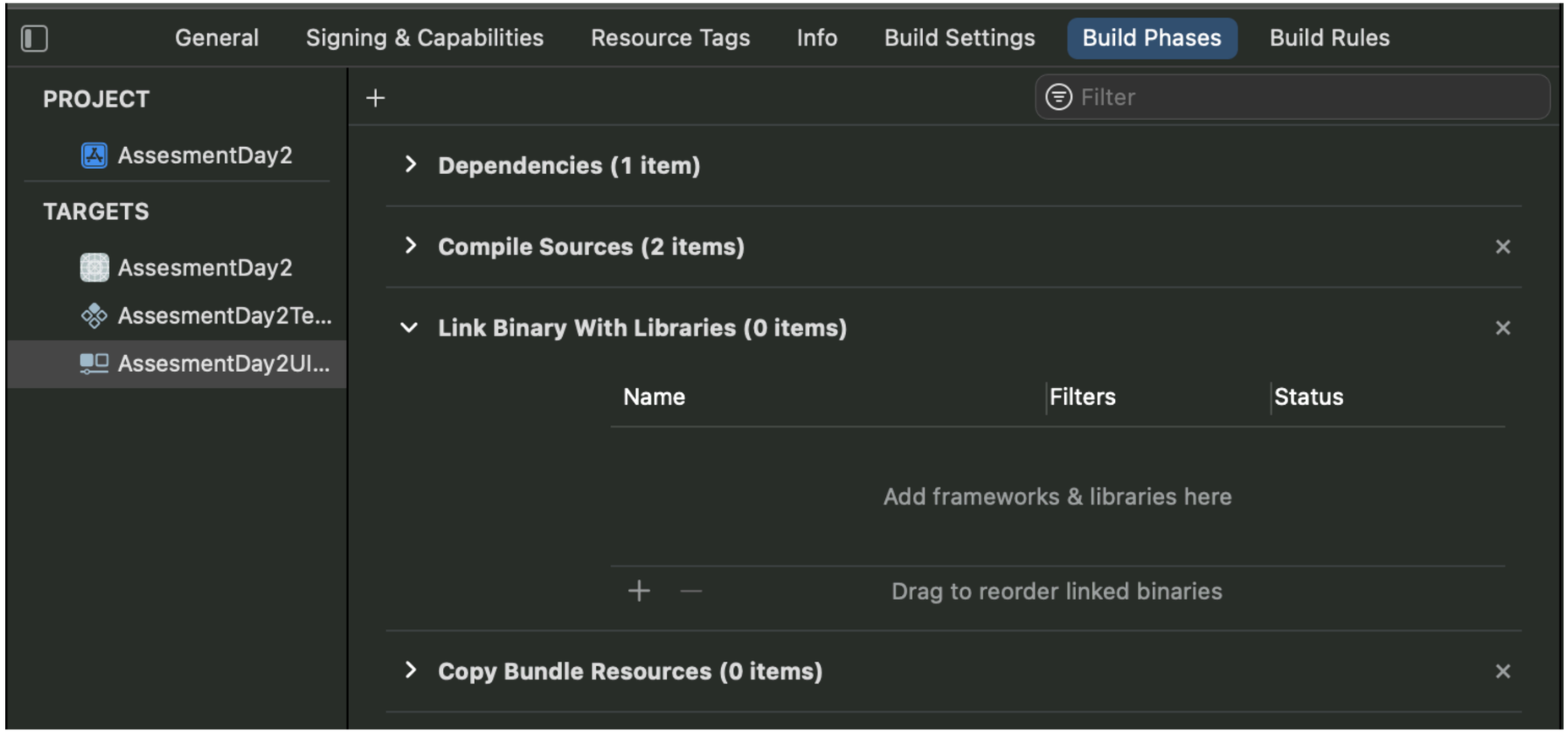
Unit Test & UI Test targets

- For Test targets, actually if you already added package on main project, Test Bundle automatically can uses the package also.
- But if you want to add the dependency only on Test Targets, you can add it in the **Build Phases** tab, put it on **Link Binary With Libraries** submenu
- Click the + **(plus)**, select your package in the list, and click **Add**.





Adding Package Dependency To Test Targets



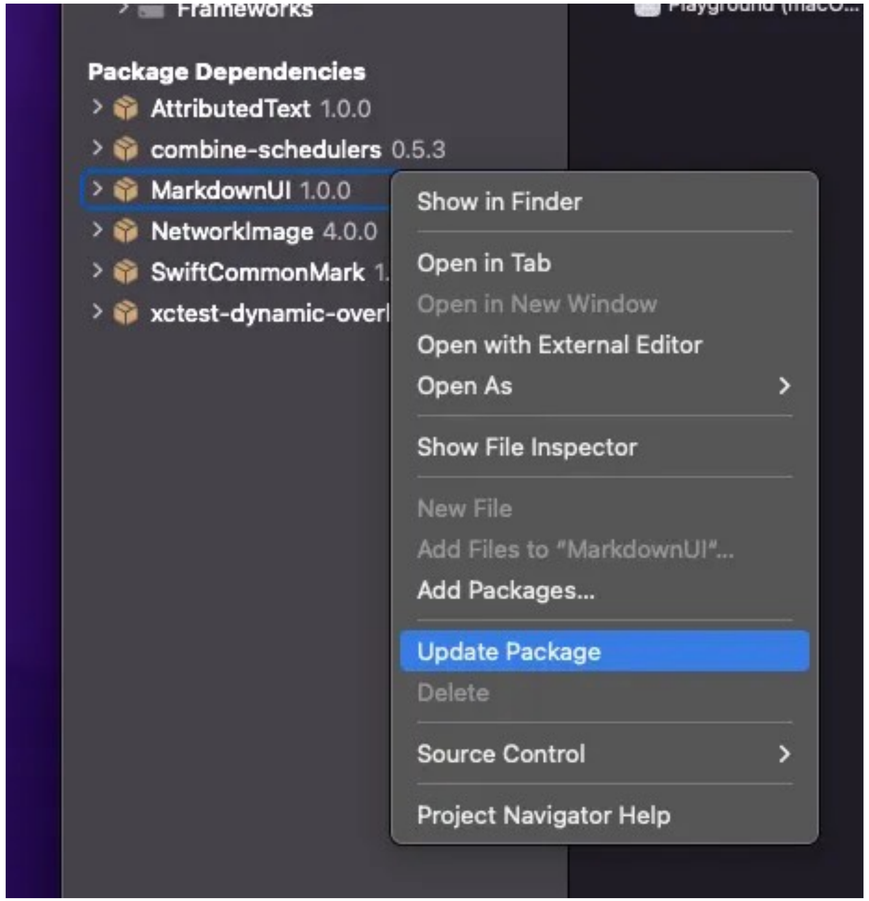
Updating Package Dependency

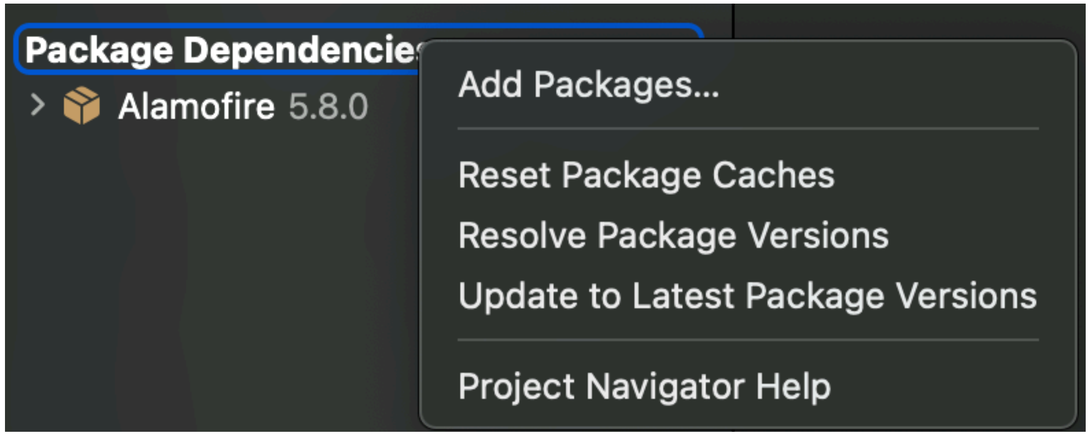
- Find your package under **Package Dependencies** in the **Project Navigator**
- **Right-click** the package and select **Update Package**.

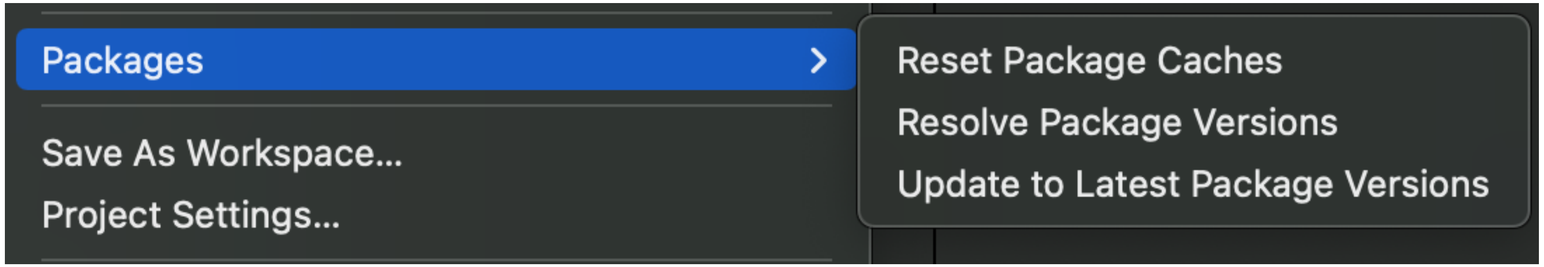
Update All Package Dependency

- **Right-clicking Package Dependencies** and selecting **Update to Latest Package Versions**
- OR by going to **File** → **Packages** → **Update to Latest Package Versions**

Updating Package Dependency

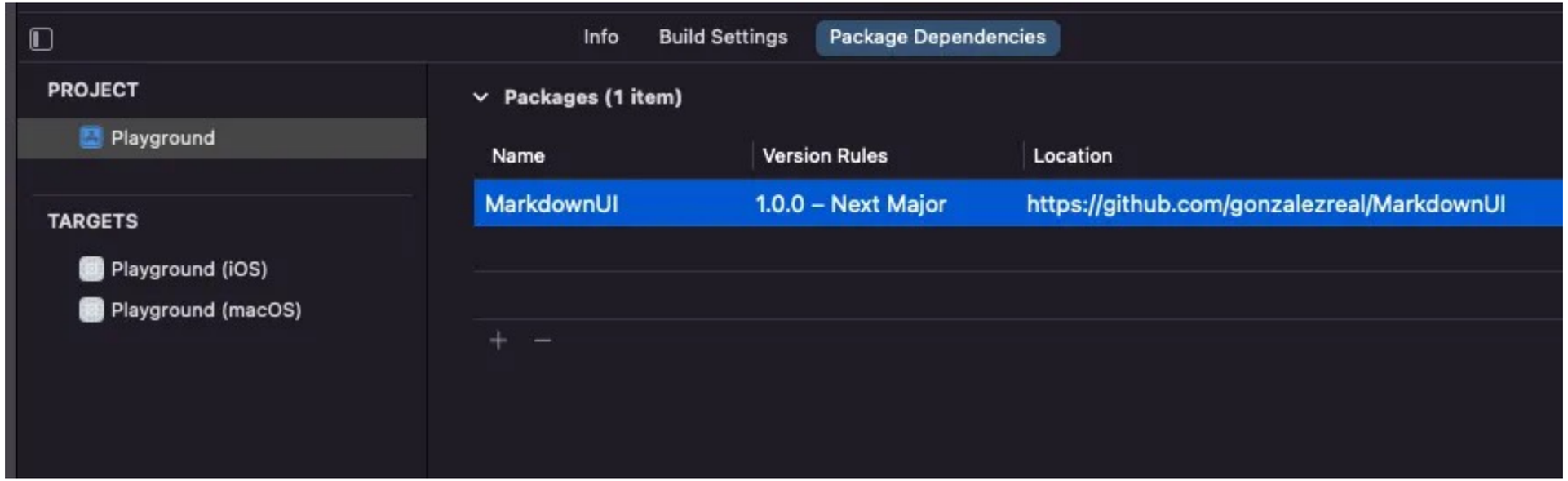






Remove Package Dependency

- Select your project in the **Project Editor**.
- In the **Package Dependencies** tab, select the package you want to remove.
- Click the – **(minus)** and then **Remove**.



Thank You!

REST API (NETWORKING)

Swift BCA Training

LESSON 6

REST API

- REST API is an API that conforms to the design principles of the REST, or *representational state transfer* architectural style
- REST APIs communicate via HTTP requests to perform standard database functions like creating, reading, updating, and deleting records (also known as CRUD) within a resource.
- The information can be delivered to a client in virtually any format including JavaScript Object Notation (JSON), HTML, XLT, Python, PHP, or plain text

Authentication

- API authentication is the process of verifying the identity of a user or other actor - in order to confirm that they have the necessary permissions for whatever they're trying to do via an API
- Authentication allows API owners to do three things:
  - Verify the identity of a client or user
  - Enable authorized clients and users can access the API
  - Prevent unauthorized access

Authorization

- API authorization is how vendors govern which elements of their API clients and users can access.
- Authorization is used for:
  - Granting access and exposure to particular resources or data for different users.
  - Governing what actions different users and clients can take with our API
  - Otherwise enforce defined access control policies

HTTP Protocol

- By default, IOS only accept HTTPS protocol for networking
- There is a setup on **Info.plist** to accept HTTP Protocol also, it is called **Allow Arbitrary Loads** on **App Transport Security Settings**

Key	Type	Value
Information Property List	Dictionary	(2 items)
App Transport Security Settings	Dictionary	(1 item)
Allow Arbitrary Loads	Boolean	NO

Alamofire

- For networking, we can use a popular dependency **Alamofire**
- You can check the implementation on here: https://github.com/Alamofire/Alamofire

```
Alamofire.request(
    <#T##url: URLConvertible##URLConvertible#>,
    method: <#T##HTTPMethod#>,
    parameters: <#T##Parameters?#>,
    encoding: <#T##ParameterEncoding#>,
    headers: <#T##HTTPHeaders?#>
).response { responseData in
    // your code to handle Response Data
}
```

Alamofire GET API

```
guard let url = URL(string: urlString) else { return }

let urlConvertible: URLConvertible = url

Alamofire.request(
    urlConvertible,
    method: .get,
    parameters: parameters, --> [String: Any]?
    encoding: URLEncoding.methodDependent, // type to set param as query string
    headers: headers, --> [String: String]? // put your Bearer Token here
).response { responseData in
    // Response data is DefaultDataResponse struct, that contains data, error,
    and etc
    // your code to handle Response Data
}
```

Alamofire POST API

```
guard let url = URL(string: urlString) else { return }

let urlConvertible: URLConvertible = url

Alamofire.request(
    urlConvertible,
    method: .post,
    parameters: parameters, --> [String: Any]?
    encoding: URLEncoding.httpBody, // type to set param as httpBody
    headers: headers, --> [String: String]? // put your Bearer Token here
).response { responseData in
    // Response data is DefaultDataResponse struct, that contains data, error,
    and etc
    // your code to handle Response Data
}
```

Networking Without Alamofire

- If you don't use Alamofire, you can use **URLSession** to call the API and get the response
- With this method, we can do Unit Test for Networking because it doesn't depend on other dependency

```
do {
    var request = try URLRequest(url: urlConvertible, method: method,
headers: headers)
    request.httpBody = nil
    URLSession.shared.dataTask(
    with: request) { data, Response, error in
      // your code here
   }.resume()
} catch {
      // your code to catch the error
}
```

Mapping Response Data

- Create object with implement **Decodable** protocol as entity for mapping result
- Use **JSONDecoder** to decode the response data and mapping to your object

```
struct Movie: Decodable {
    let title, plot, poster: String

    enum MovieCodingKeys: String, CodingKey {
        case title = "Title"
        case plot = "Plot"
        case poster = "Poster"
    }
}
```

Mapping Response Data

```
guard let data = data else { return }

do {
   let result = try JSONDecoder().decode([Movie].self, from: data)
   onSuccess(result)
} catch let jsonErr {
   print("Error Serialization json:", jsonErr)
}
```

Thank You!