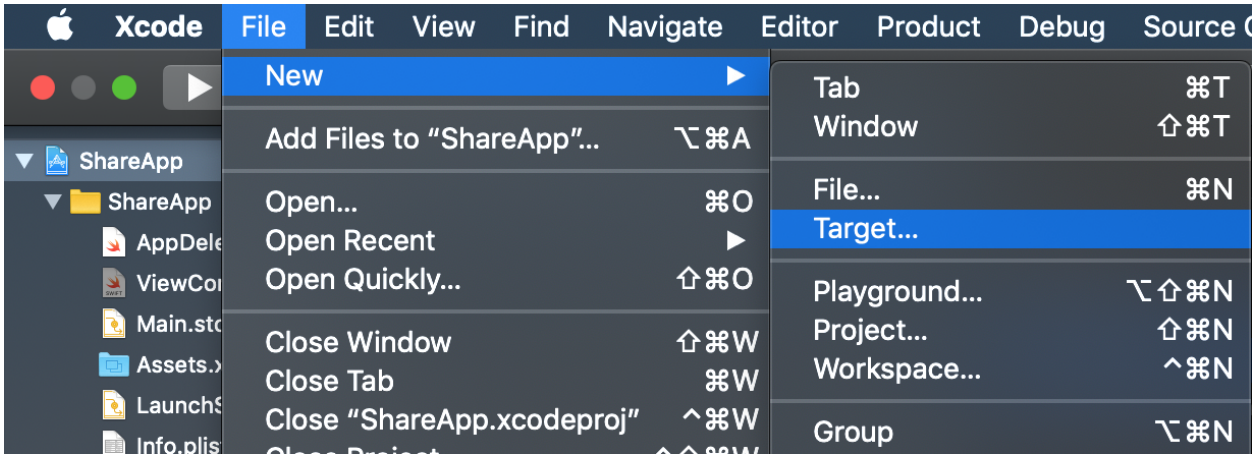
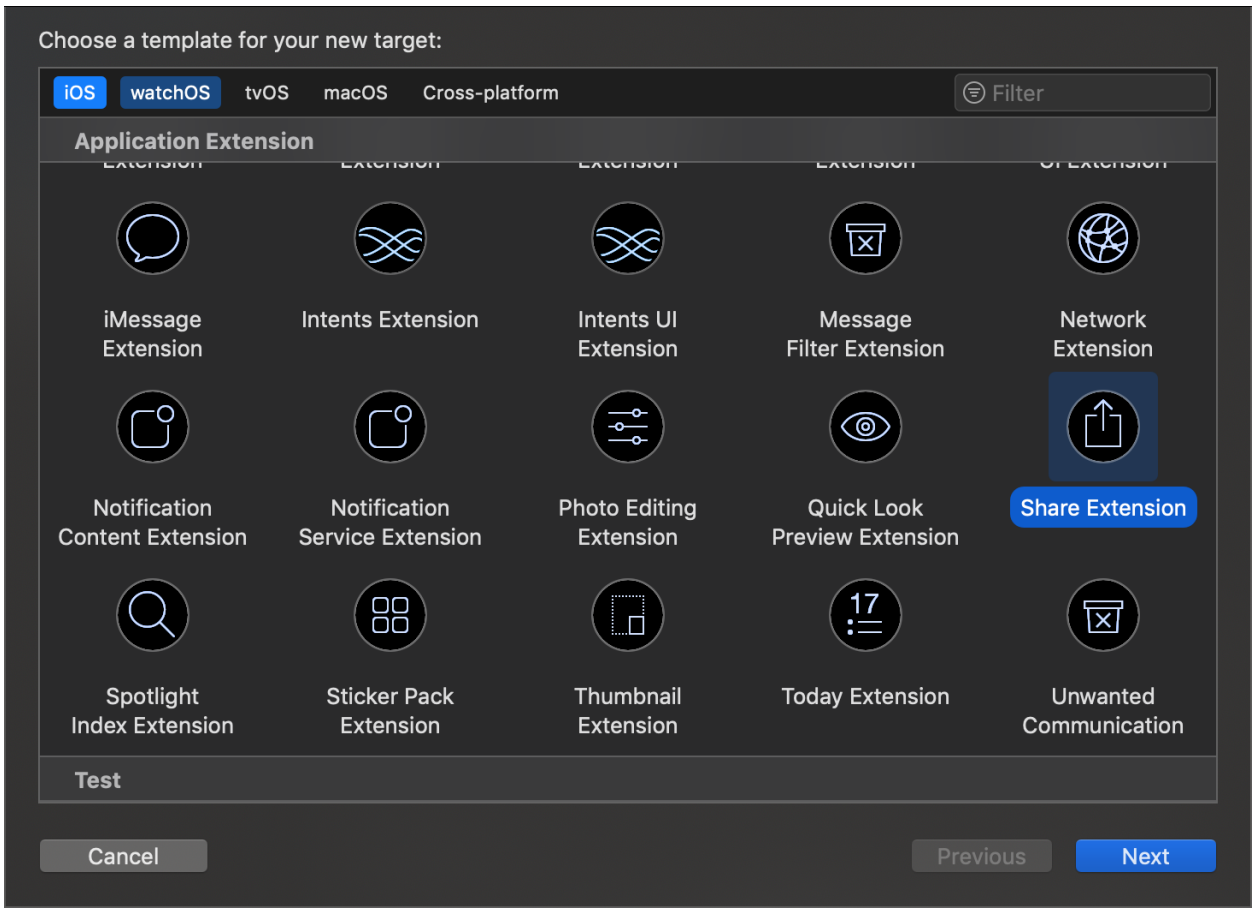
How to create a Share Extension for iOS 11 and above

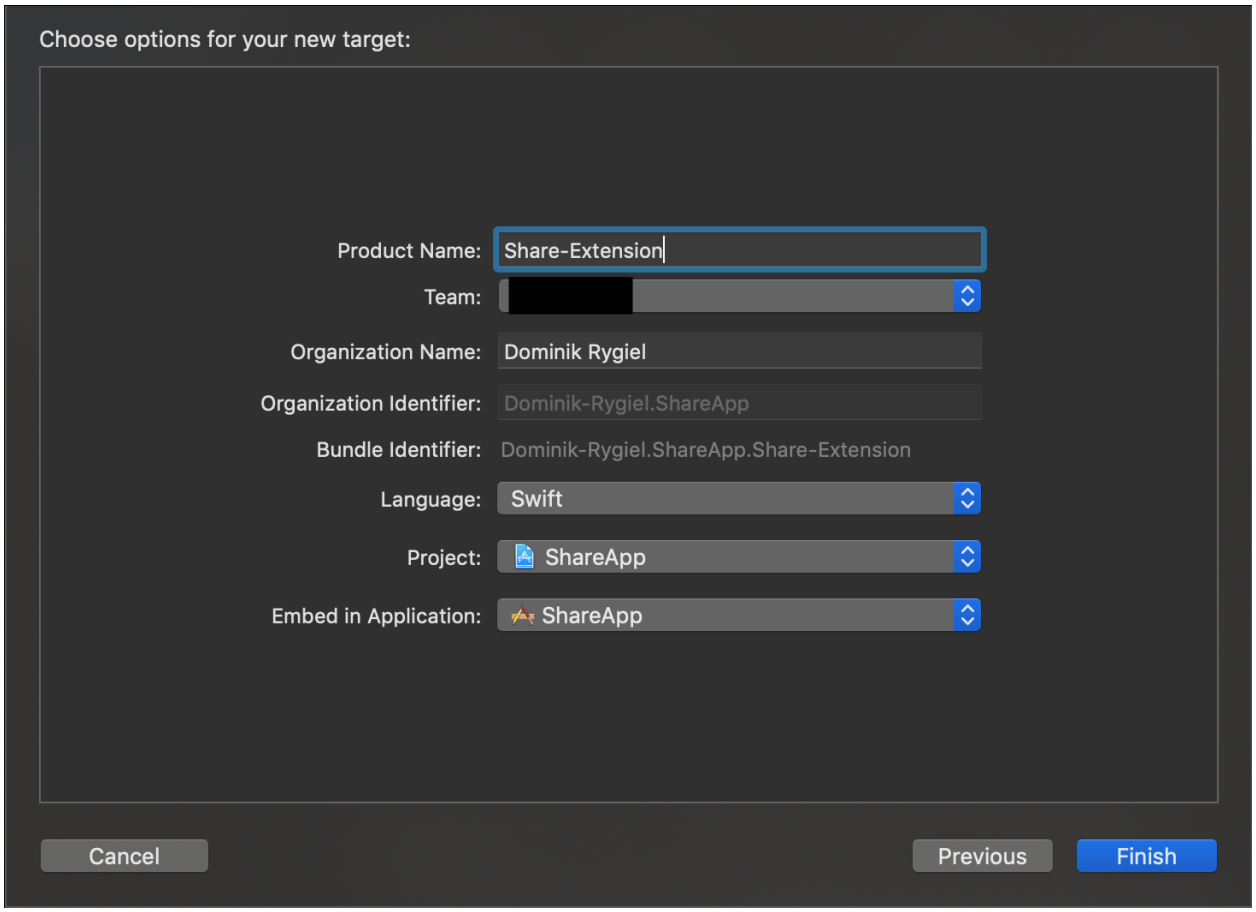
In 2014 along with iOS 8, Apple added the capability to share data between apps. There are some incredibly good tutorials that explain how to add share to your app, but most of them don't use the new feature that was presented in iOS 11 together with *Files,* don't show how to open your app after sharing files nor how to interact with *Post* dialogue.
So today, I want to share with you my knowledge about this topic.

1. Basic setup

First, we create a basic project. In addition to our project called *ShareApp* we need to add share extension. Go to *File* => *New* => *Target* and select *Share Extension*:

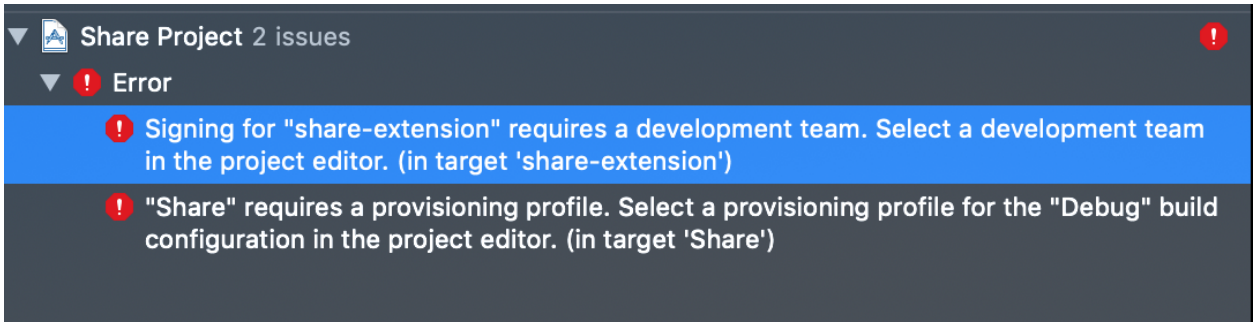

Xcode
File
Edit
View
Find
Navigate
Editor
Product
Debug
Source
New
Add Files to “ShareApp”...
Open...
Open Recent
Open Quickly...
Close Window
Close Tab
Close “ShareApp.xcodeproj”
Tab
Window
File...
Target...
Playground...
Project...
Workspace...
Group
ShareApp
ShareApp


Choose a template for your new target:
iOS
watchOS
tvOS
macOS
Cross-platform
Filter
Application Extension
iMessage Extension
Intents Extension
Intents UI Extension
Message Filter Extension
Network Extension
Notification Content Extension
Notification Service Extension
Photo Editing Extension
Quick Look Preview Extension
Share Extension
Spotlight Index Extension
Sticker Pack Extension
Thumbnail Extension
Today Extension
Unwanted Communication
Test
Cancel
Previous
Next



Now we can see a new folder in our project:

Let's try to run our project:



Whoops! There's 1 warning (we'll get to that later) and a second error that informs us that we don't have a provisioning profile for *ShareApp* (host app) **and *ShareExtension.***
Wait, what? We need two provisioning profiles, why?
Yes, we need two of them because we should treat app and extension as two different apps.
So let's create them. Go to the Apple Developer account and register our App ids.

Registering an App ID

The App ID string contains two parts separated by a period (.) — an App ID Prefix that is defined as your Team ID by default and an App ID Suffix that is defined as a Bundle ID search string. Each part of an App ID has different and important uses for your app. Learn More

App ID Description

Name: ShareApp

You cannot use special characters such as @, &, *, ', "

App ID Prefix

Value: [redacted] (Team ID)

App ID Suffix

Explicit App ID

If you plan to incorporate app services such as Game Center, In-App Purchase, Data Protection, and iCloud, or want a provisioning profile unique to a single app, you must register an explicit App ID for your app.

To create an explicit App ID, enter a unique string in the Bundle ID field. This string should match the Bundle ID of your app.

Bundle ID: Dominik-Rygiel.ShareApp

We recommend using a reverse-domain name style string (i.e., com.domainname.appname). It cannot contain an asterisk (*).

Wildcard App ID

This allows you to use a single App ID to match multiple apps. To create a wildcard App ID, enter an asterisk (*) as the last digit in the Bundle ID field.

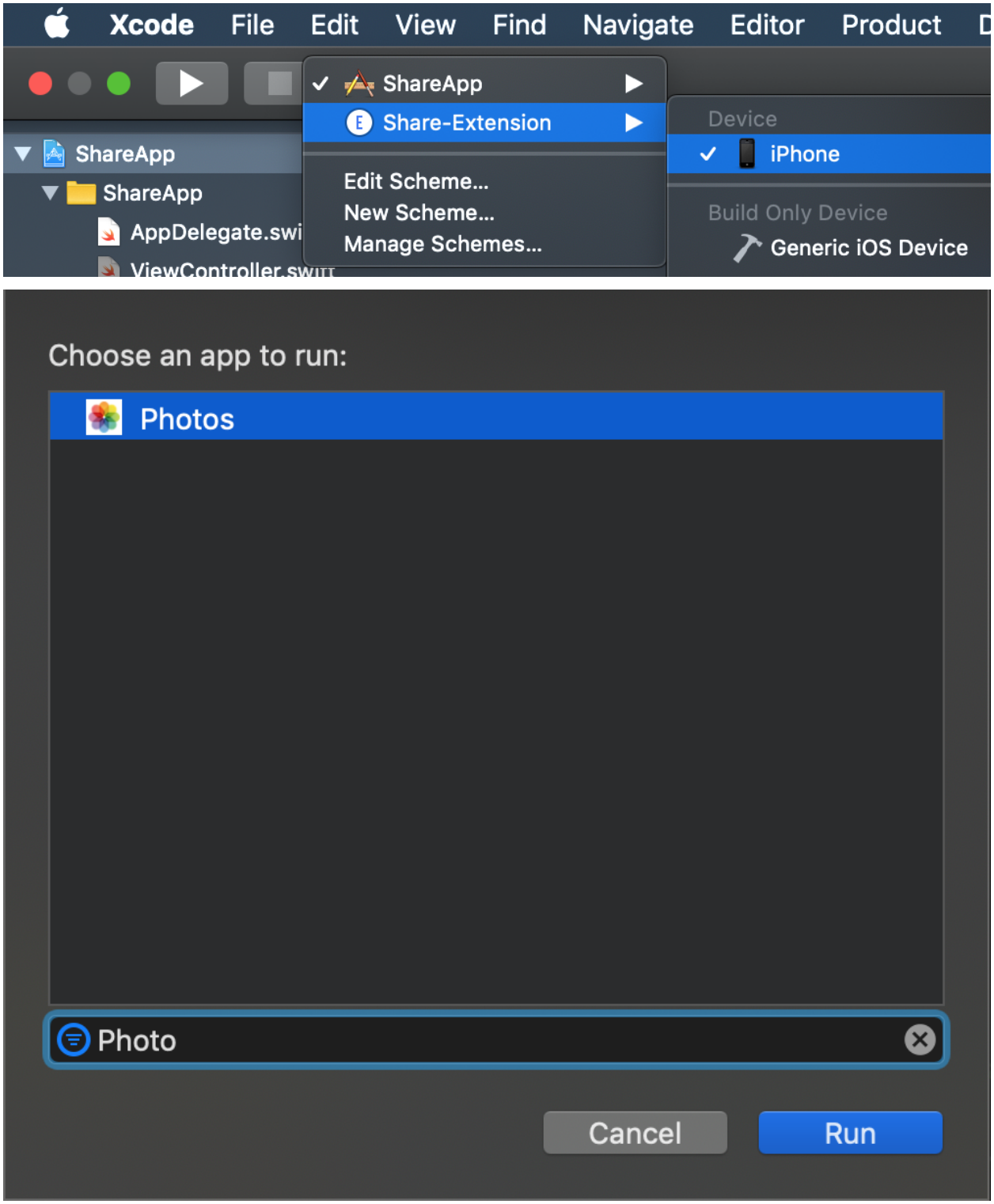
Bundle ID:

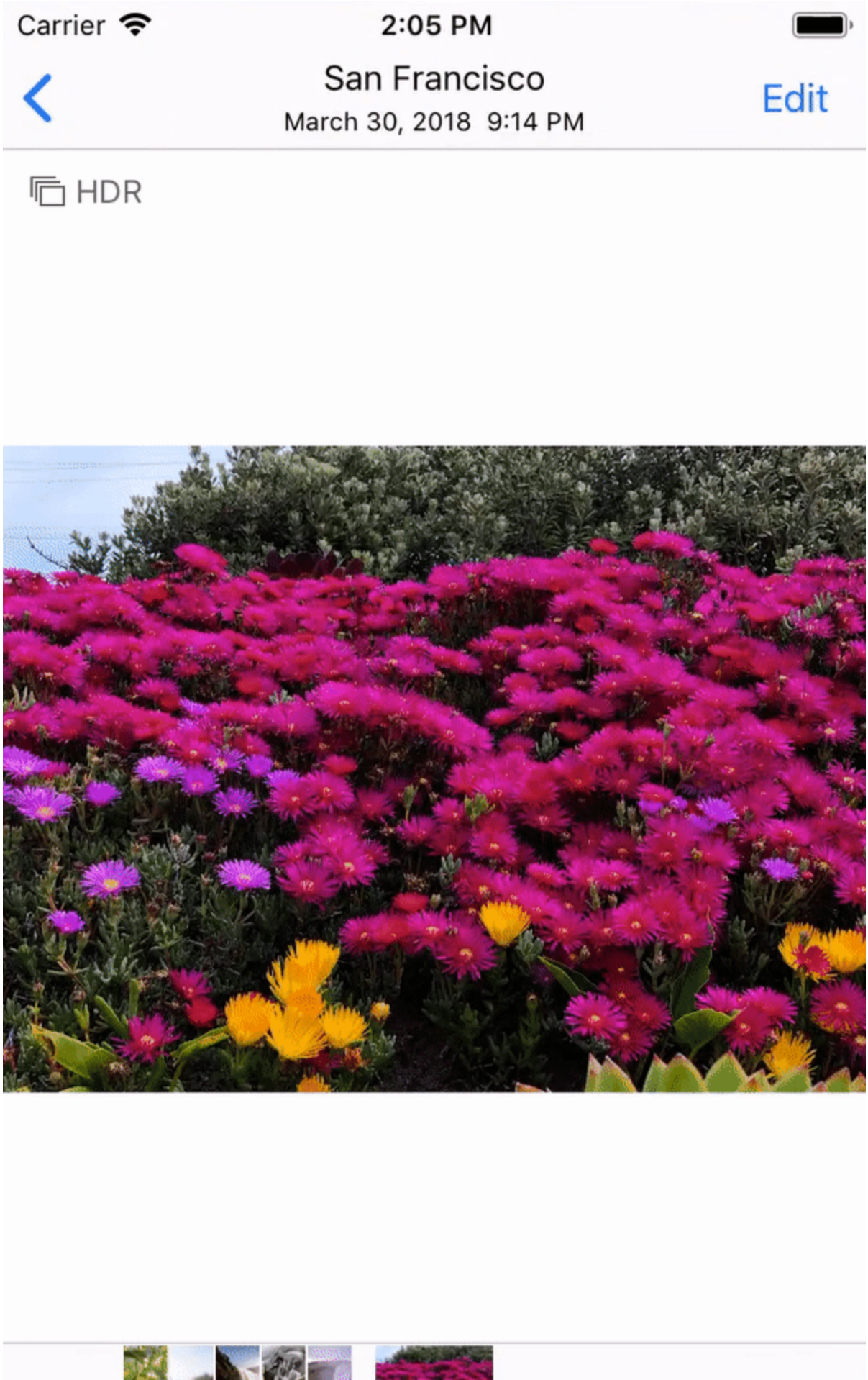
Example: com.domainname.*

Remember to enable *App Groups* service but if you forget to do that, don't worry we can edit it later. Repeat this process for extension (obviously change the name and bundle ID).
Now we're all set with the provisioning profiles, so let's dive a little into the code.
When you open *ShareViewController.swift* you already have some auto-generated code:

1. *IsContentValid,* function which basically enables or disable the *Post* Button. You can do any validation here (for example, check if the text that user wants to add isn't too long)
2. *DidSelectPost,* a function which is called when the user taps on the "Post" button
3. In *configurationItems* you can add the object of class *SLComposeSheetConfigurationItem,* for example:

In order to run the app, change target to *Share-Extension* and then pick an app you want to use your extension with:


Xcode
File
Edit
View
Find
Navigate
Editor
Product
ShareApp
Share-Extension
Edit Scheme...
New Scheme...
Manage Schemes...
Device
iPhone
Build Only Device
Generic iOS Device
ShareApp
ShareApp
AppDelegate.swi
Choose an app to run:
Photos
Photo
Cancel
Run

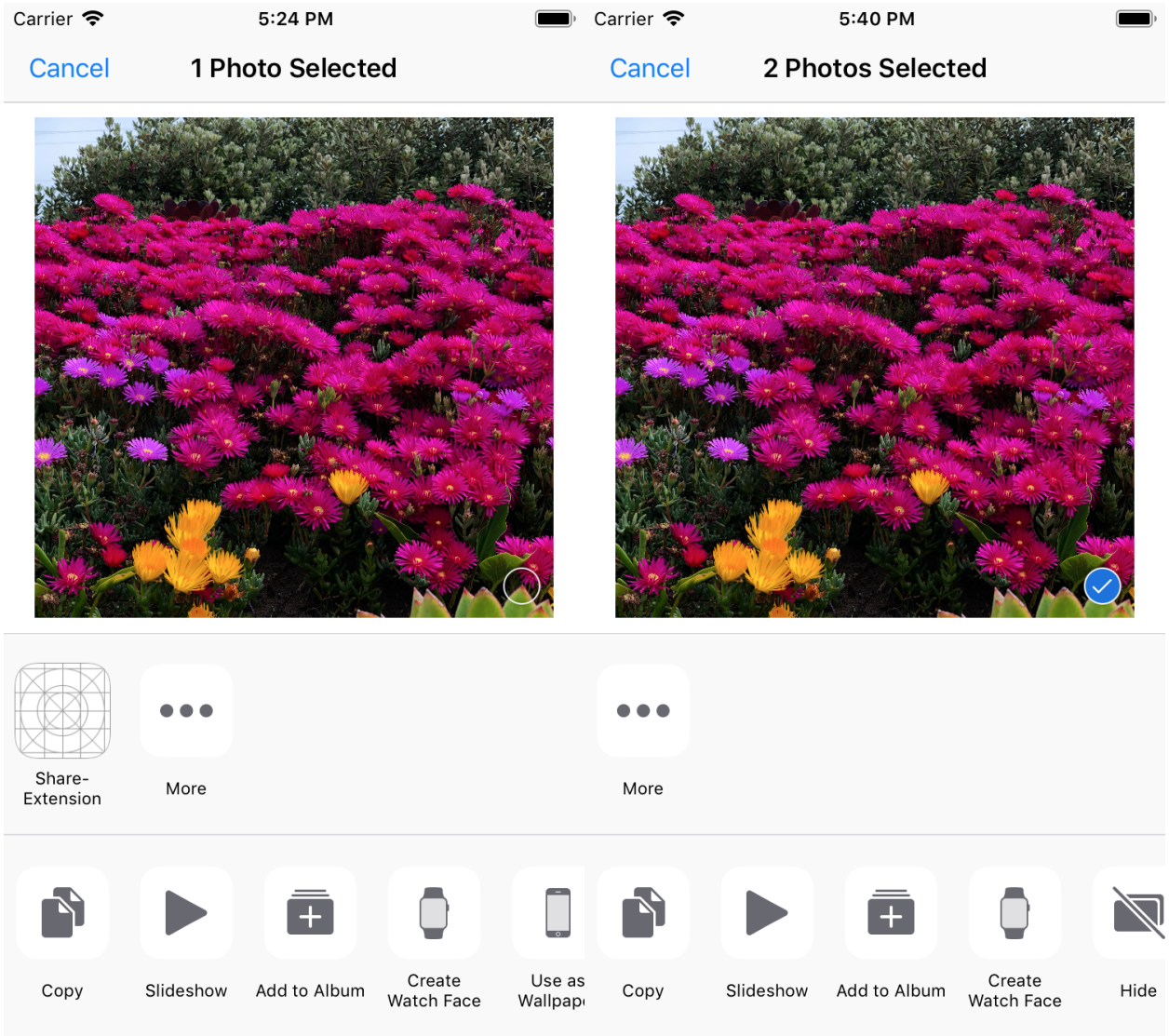

Carrier
2:05 PM
San Francisco
March 30, 2018 9:14 PM
Edit
HDR

2. Add rules to Info.plist

At the beginning, I told you that we have got a warning.

> ⚠ Embedded binary's NSExtensionActivationRule is TRUEPREDICATE. Before you submit your containing app to the App Store, be sure to replace all uses of TRUEPREDICATE with specific predicate statements or NSExtensionActivationRule keys. If any extensions in your containing app include TRUEPREDICATE, the app will be rejected.

In order to get the app through the AppStore review and avoid warnings, we need to add to *Info.plist* information about the maximum count of images.
We need to remove the *TRUEPREDICATE* value and add the key *NSExtensionActivationSupportsImageWithMaxCount* with value 1.
Now users will be able to share only a single photo. If he or she wants to share more than one, our share button won't be visible:

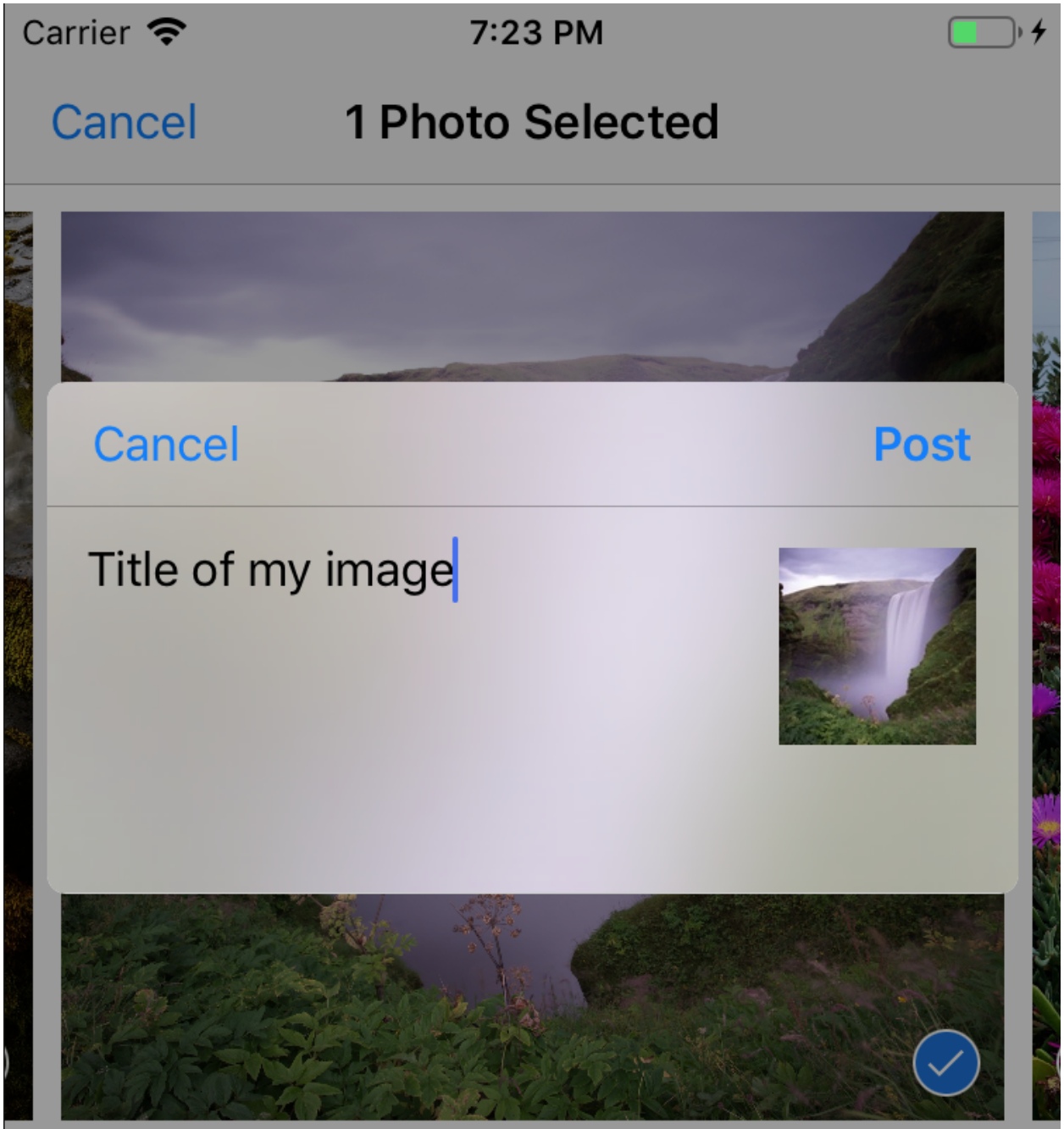


3. Sharing Data

Let’s say that we want to share an image and display it in our main app. The first thing we need to do is load data

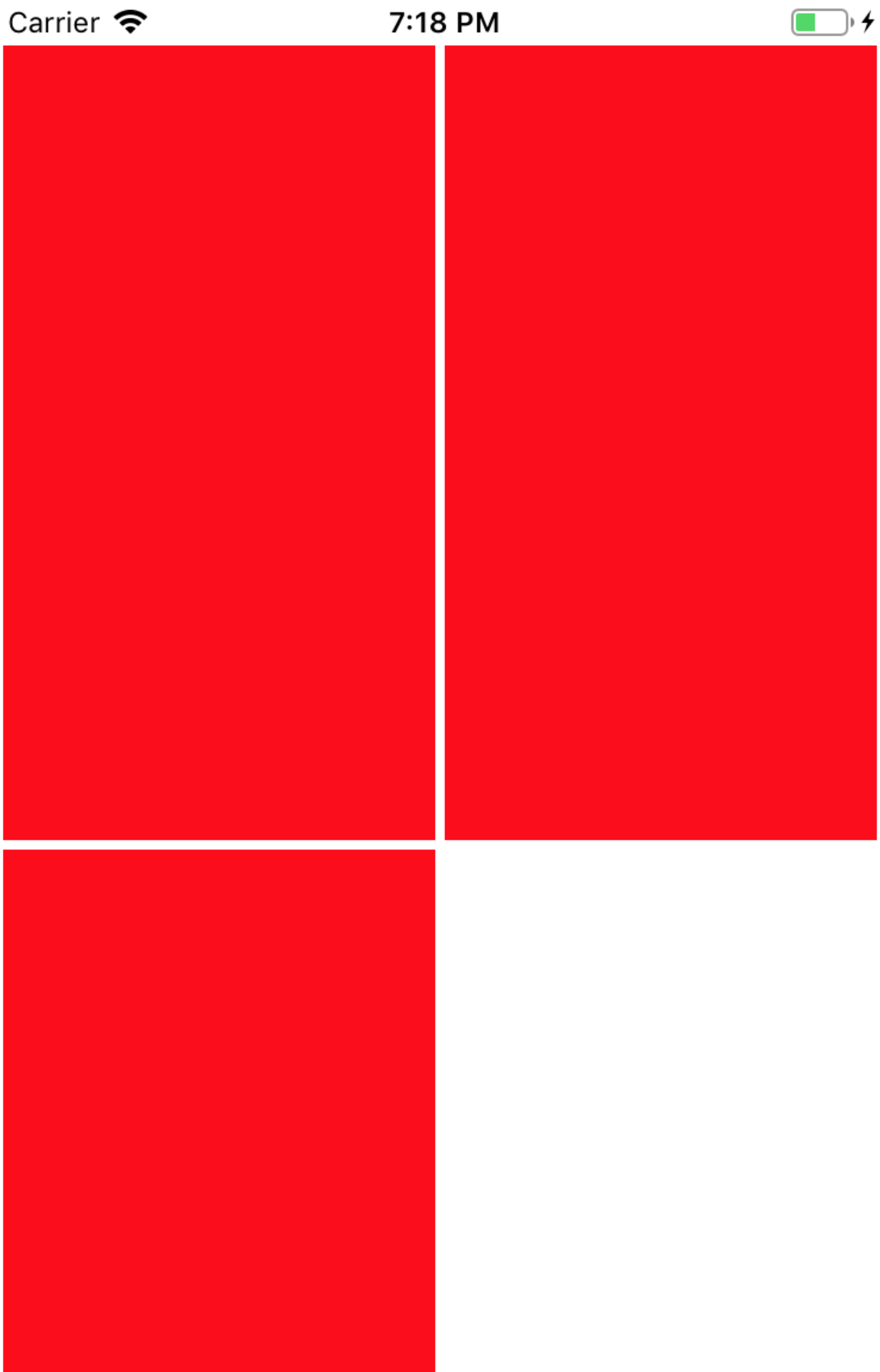
1. *ShareViewController* has textView by default. If you want to get text provided by the user then that is the correct way.
2. In *item.attachments* we have all files that the user shared. For our purpose, we want to allow the user to add only a single image.
3. We need to load our attachment as Data, but to do that we need to determine what is the object type. In this case, it will be an image.
4. Function *completeRequest* basically tells the host app to dismiss the share extension.

Let’s try it.

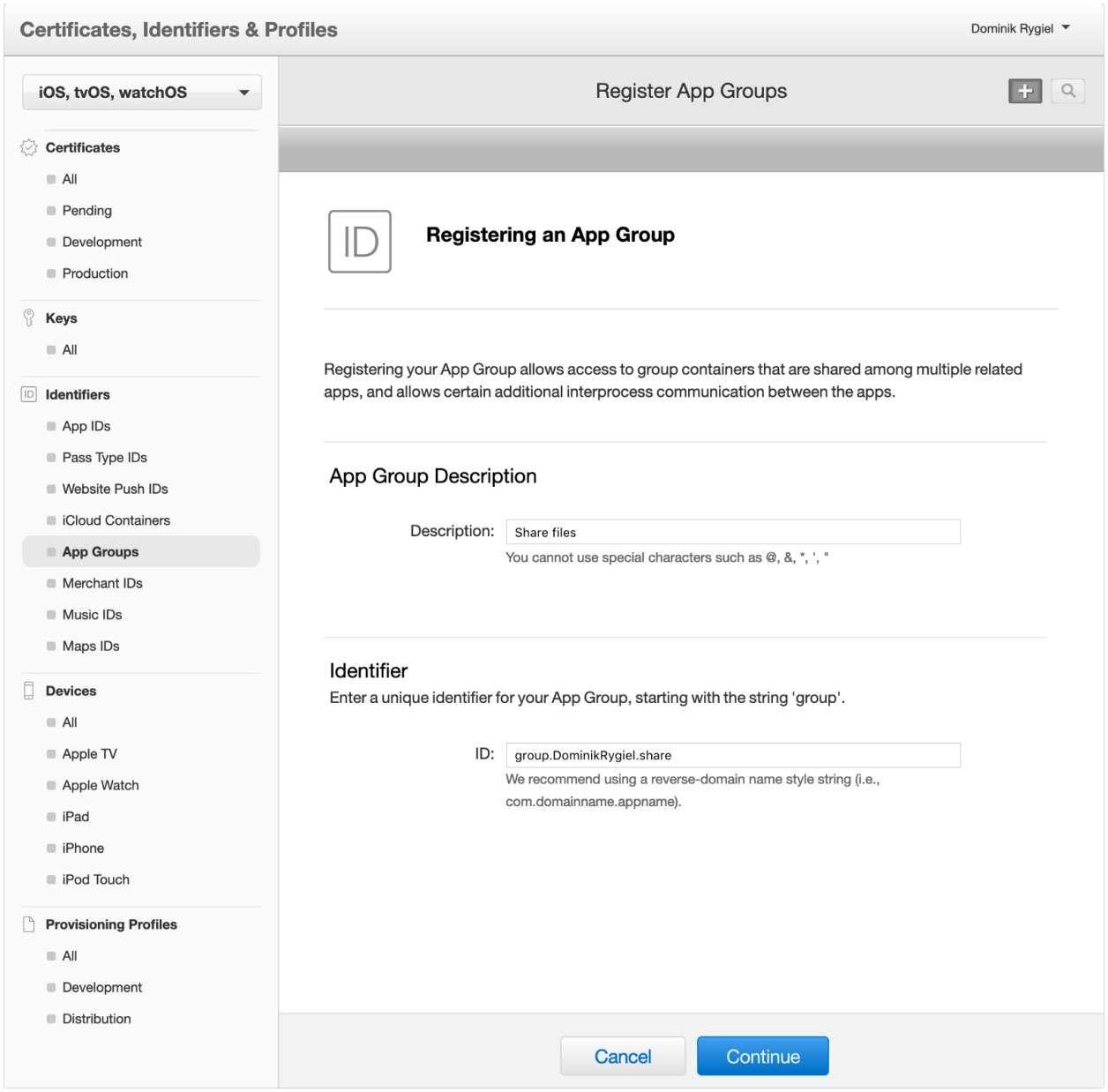

Carrier
7:23 PM
Cancel
1 Photo Selected
Cancel
Post
Title of my image
Share-
Extension
More

```
Optional("Title of my image")
<UIImage: 0x6000009d4d20>, {1668, 2500}
```

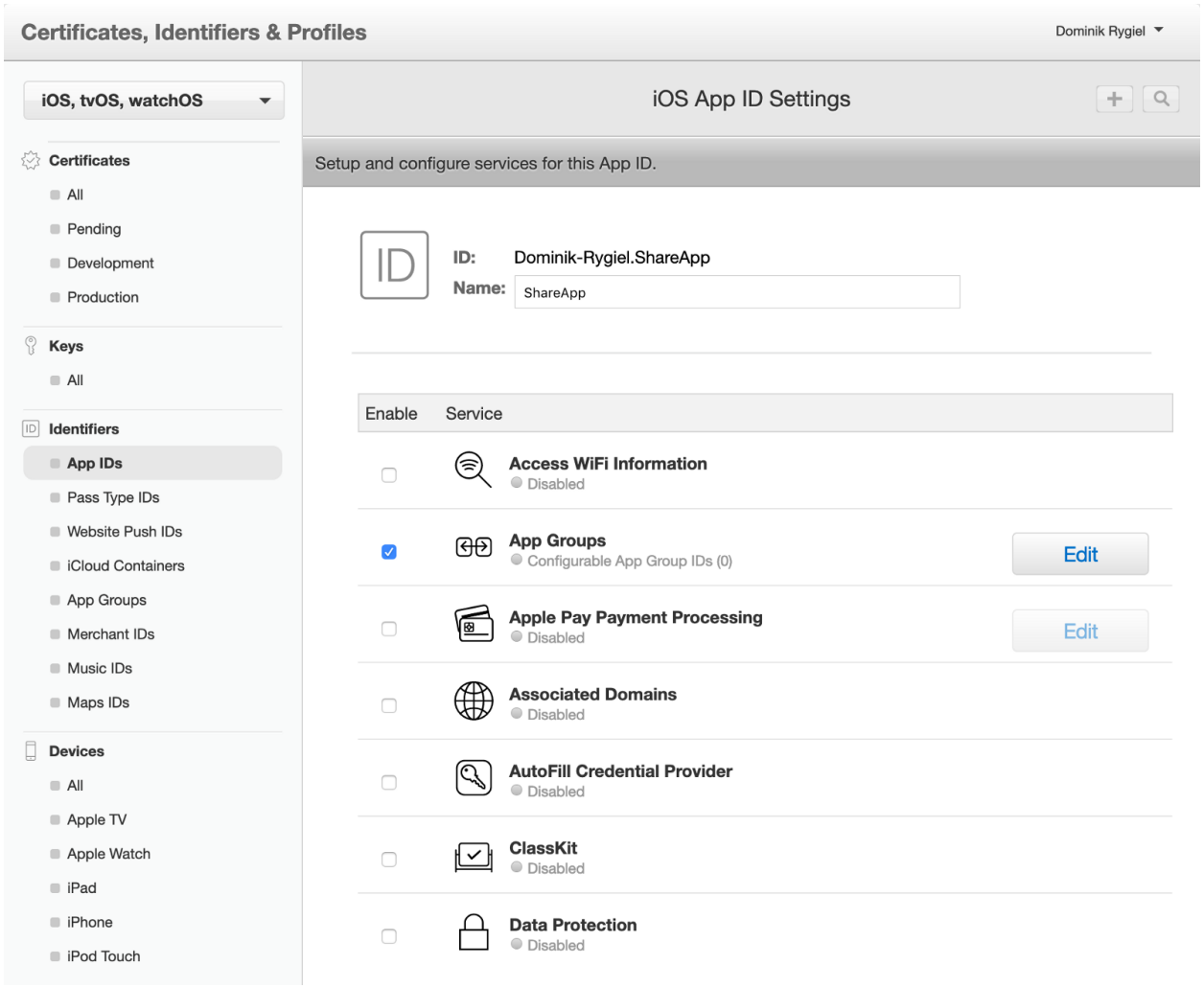
So the result is like we expected, we have an image and a title.
Now we need to create a basic VC in our host app.
Let’s create a Collection View that displays cells with the image
Run our code to determine if everything works well. The result should look like this:

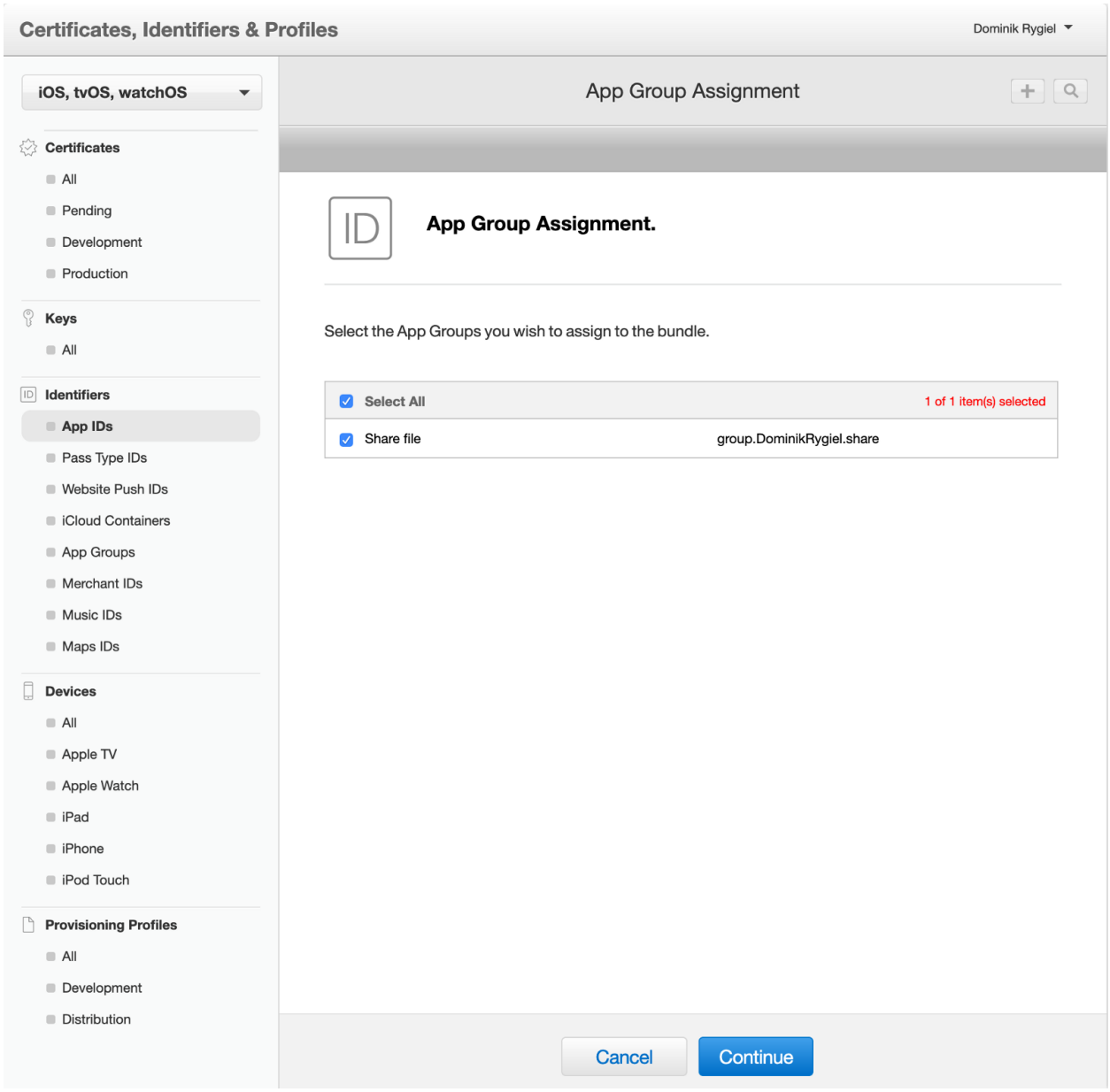

Carrier
7:18 PM

So we've got code that takes the image from photos, we have code that displays the image but we don't have a logic that shares our image from extension to our host app.
To do this, we need to create an *AppGroup*:

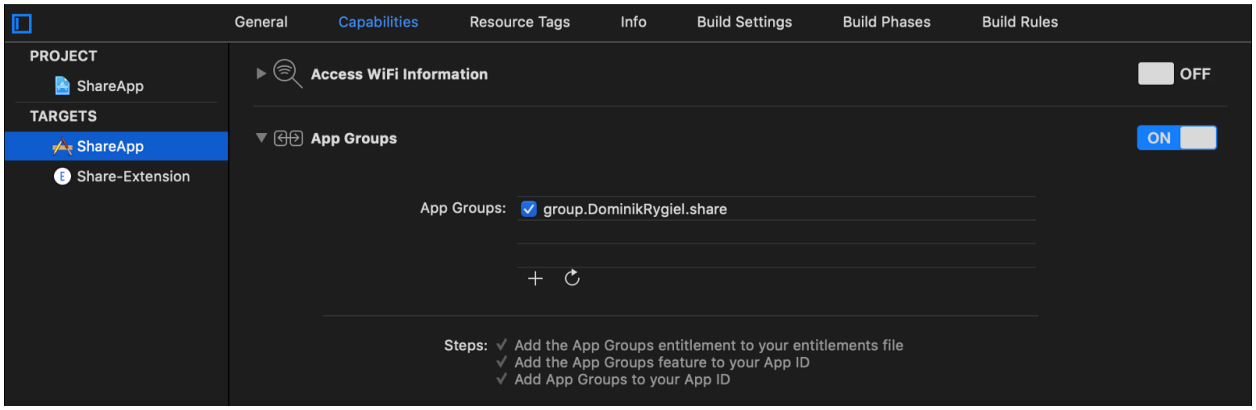


And add it to both of our App Ids


Certificates, Identifiers & Profiles
Dominik Rygiel
iOS, tvOS, watchOS
Certificates
All
Pending
Development
Production
Keys
All
Identifiers
App IDs
Pass Type IDs
Website Push IDs
iCloud Containers
App Groups
Merchant IDs
Music IDs
Maps IDs
Devices
All
Apple TV
Apple Watch
iPad
iPhone
iPod Touch
iOS App ID Settings
Setup and configure services for this App ID.
ID
ID: Dominik-Rygiel.ShareApp
Name: ShareApp
Enable
Service
Access WiFi Information
Disabled
App Groups
Configurable App Group IDs (0)
Edit
Apple Pay Payment Processing
Disabled
Edit
Associated Domains
Disabled
AutoFill Credential Provider
Disabled
ClassKit
Disabled
Data Protection
Disabled



Unfortunately, this action will invalidate our provisioning profiles, so we need to refresh them. Go to *Provisioning Profiles => Development,* find our profiles => *Edit => Generate.*
Now go to Xcode and add app group capability. **Remember that you need to do that in both App and Extension.**



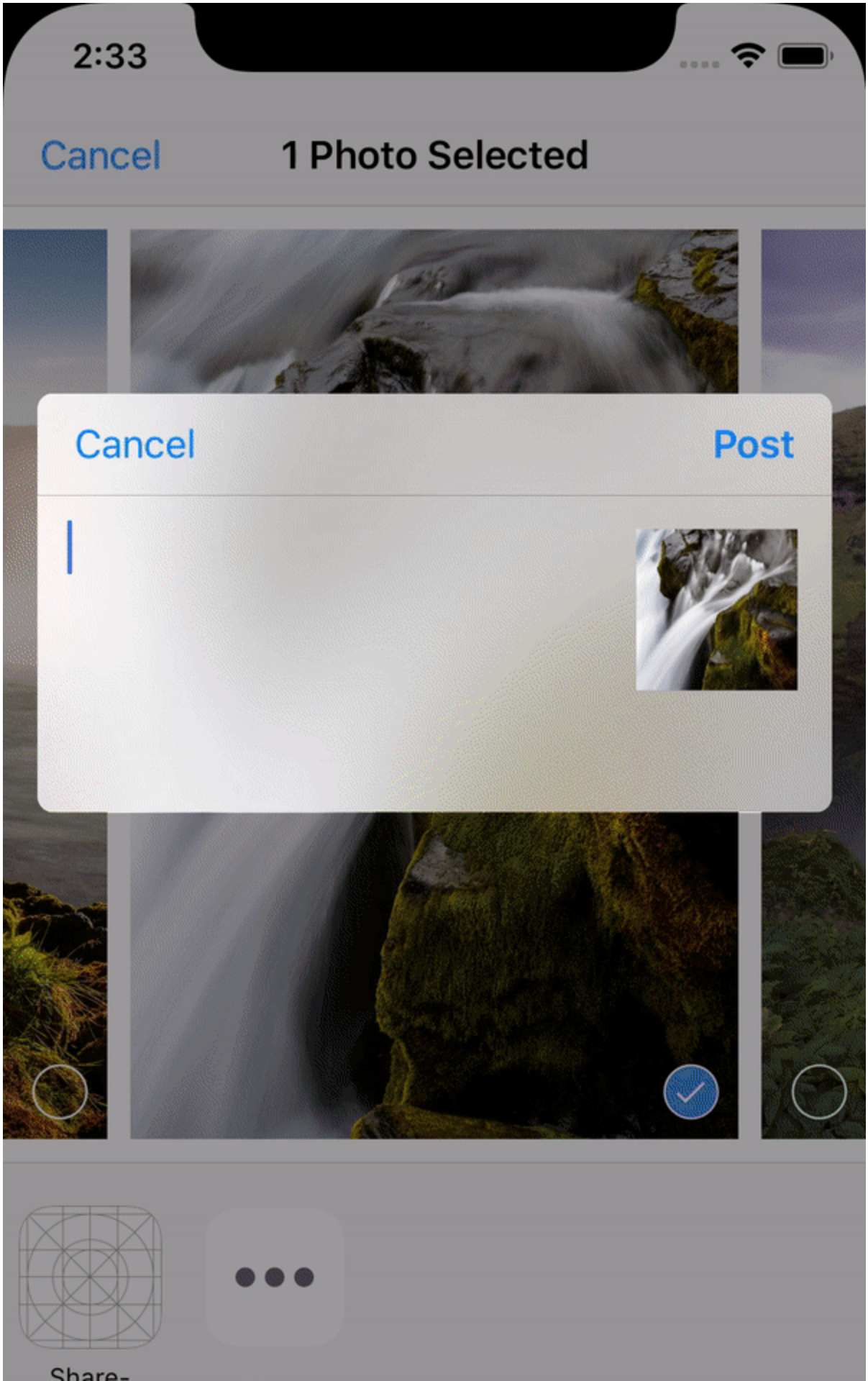
Awesome, we have app group in both apps. But what is it and why we did that? Put simply, app groups allow our apps to share data with each other. Without it our *Share-Extension* won't be able to send a shared image to our host app.
So we're all set to share data. But how can we achieve that? We simply use *UserDefaults*:

1. We create user defaults with a suit name that is **the same as our app group name.**
2. We want to add only a single image but we could have there an image that we added earlier so we must remove it and add the new image.

And then we need to update the code in our VC to display images:

1. Like in *ShareViewController,* we need to add user defaults.
2. We want to reload data also when the app becomes active.
3. We need to get data from user defaults and reload *CollectionView*.

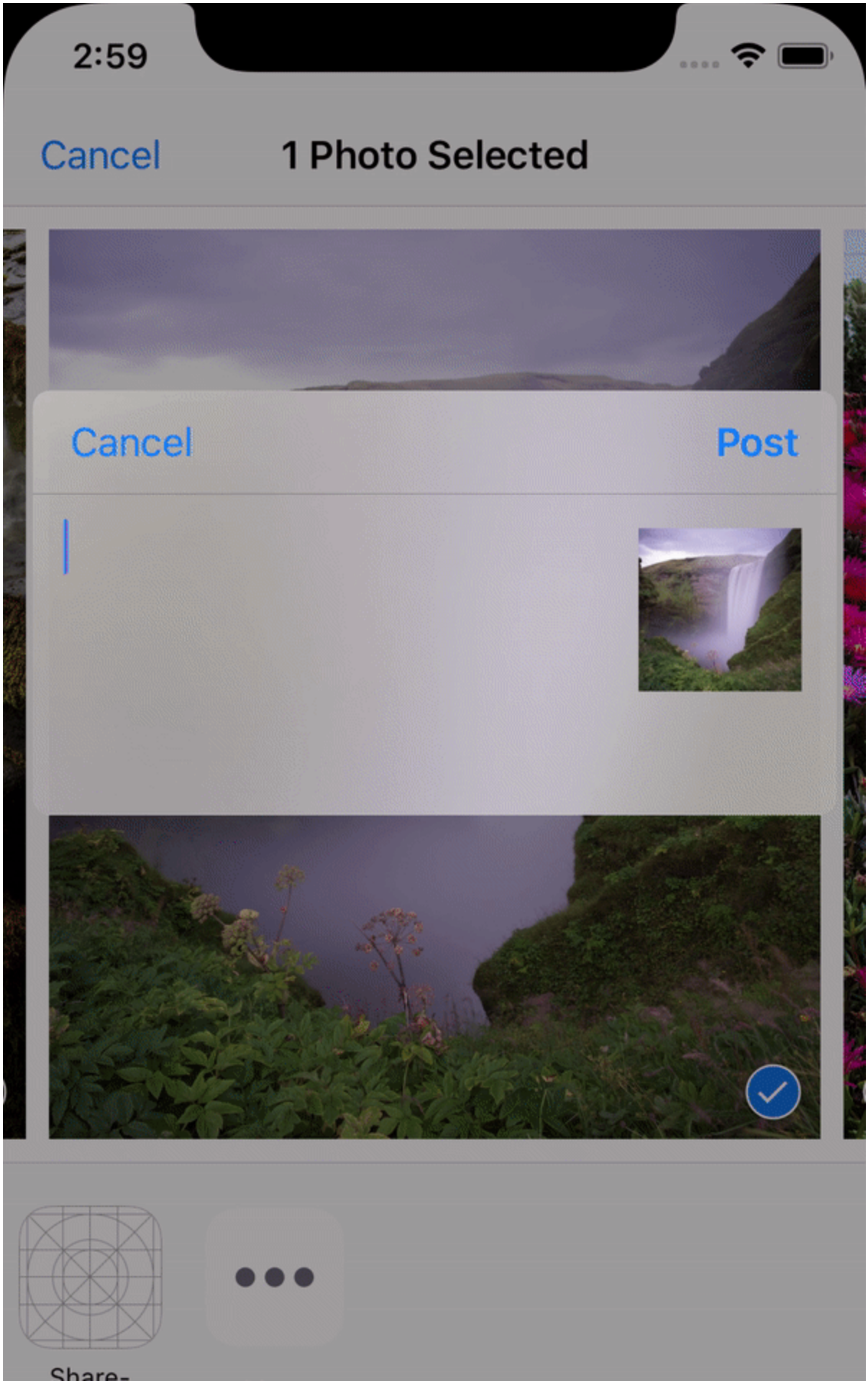
Everything looks great so run our code and try to share an image.
Does it work…? No, and that's a common mistake that I wanted to show you.
As I told you earlier, "Function *completeRequest* (…) dismiss the share extension." and *loadDataRepresentation* are an asynchronous function. **That basically means our ShareViewController will be released from memory and in closure our *self* will be *nil.***
How are we going to fix it? In an easy way, just move *completeRequest* at the end of the *loadDataRepresentation* closure.
Let's try again:


2:33
Cancel
1 Photo Selected
Cancel
Post

Everything works great but there are some things that we can do better.

4. Open the application after sharing

Let’s say we want to automatically open the app after we shared a file.
First, we need to add a URL scheme for our **ShareApp**. To do that, open Info.plist and define the **CFBundleURLTypes** and **CFBundleURLScheme** keys.
Then add code in ShareViewController that allows us to open our app.
Let’s test it:


2:59
Cancel
1 Photo Selected
Cancel
Post

Awesome! But there's one thing that seems redundant. Why do we need to open popup and press *"Post"* if our app only takes images and we don't want to allow users to add any text?

5. Hide Post dialogue

Firstly we want to make share photo fully automatic (without needing to press "*Post"* by the user). How can we achieve that? In *configurationItems* just call the *didSelectPost* method.


ROBERT DOWNEY JR.
CHRIS EVANS
MARK RUFFALO
CHRIS HEMSWORTH
SCARLETT JOHANSSON
JEREMY RENNER
DON CHEADLE
PAUL RUDD
BRIE LARSON
KAREN GILLAN
DANAI GURIRA
BRADLEY COOPER
JOSH BROLIN
MARVEL STUDIOS
AVENGERS
ENDGAME

Great! But if we look a little closer, for a half second you can see a popup and the keyboard. We don't want that because we want the user to have the best experience without weird things popping up.
So in the initialization of our *shareViewController* let's add code that hides the keyboard before it is shown.
Also, we need to hide our view. How to do that?
Now our app works perfectly well!

Bonus: Support iOS 10 and below

This tutorial works for iOS 11 and above. But if you want to support iOS 10 or below we only need to change the called method from the attachment object.
The difference is that now we load the object as a *URL* to the file that we're sharing:

I hope that this tutorial helped you with share extension!
You can download the complete project here.