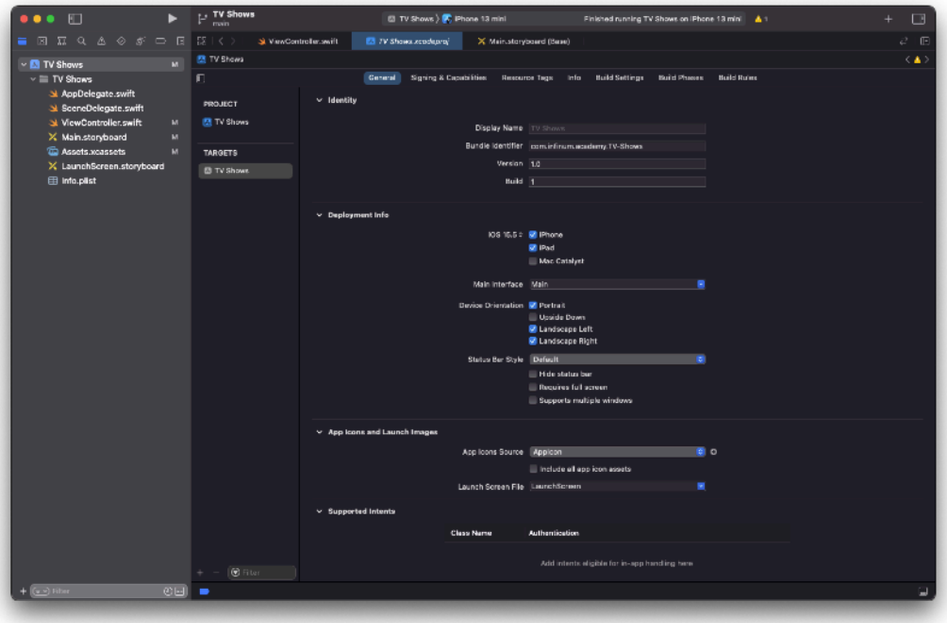
04. iOS development

neverstop

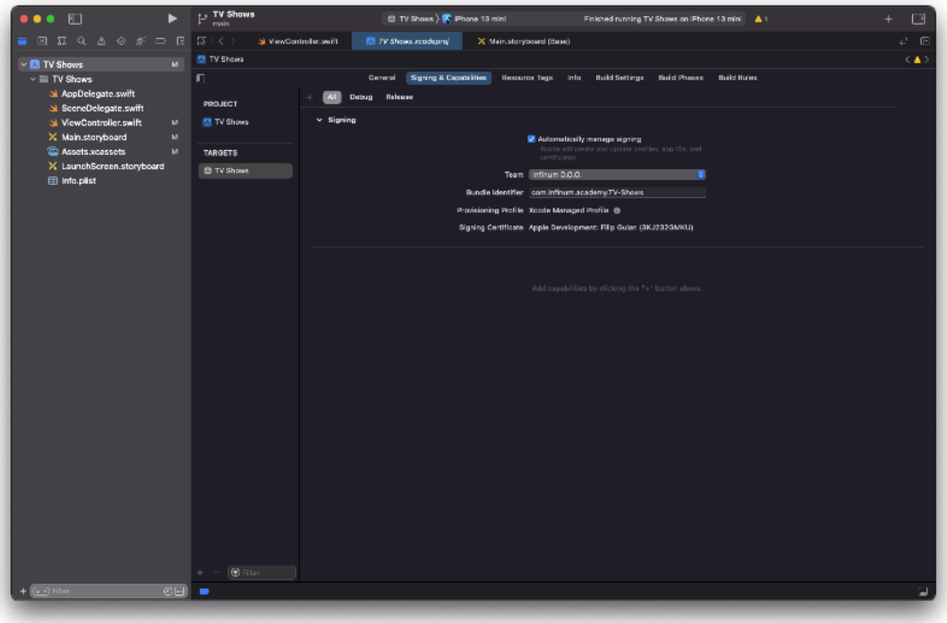
INFINUM

01

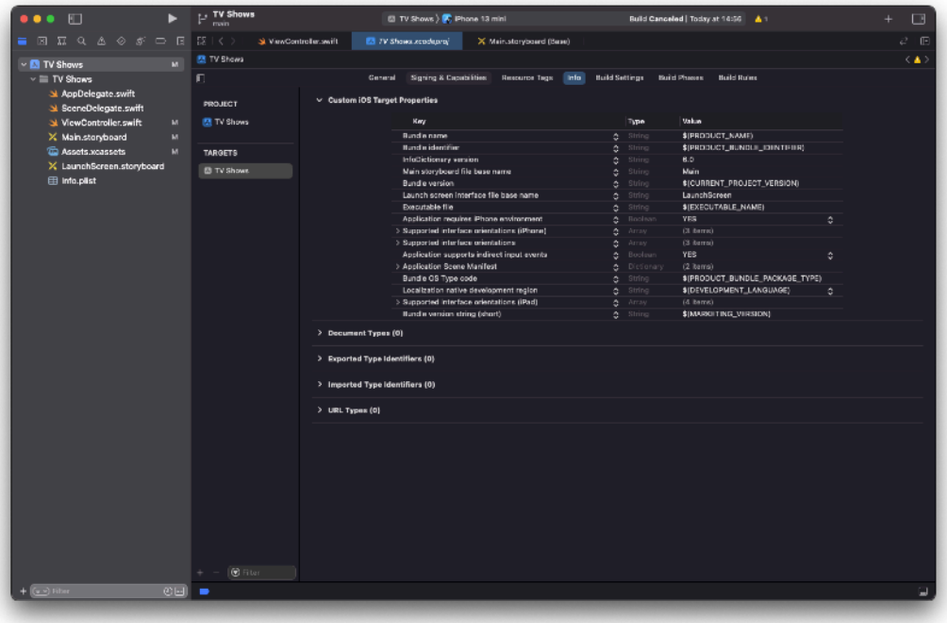
Xcode overview



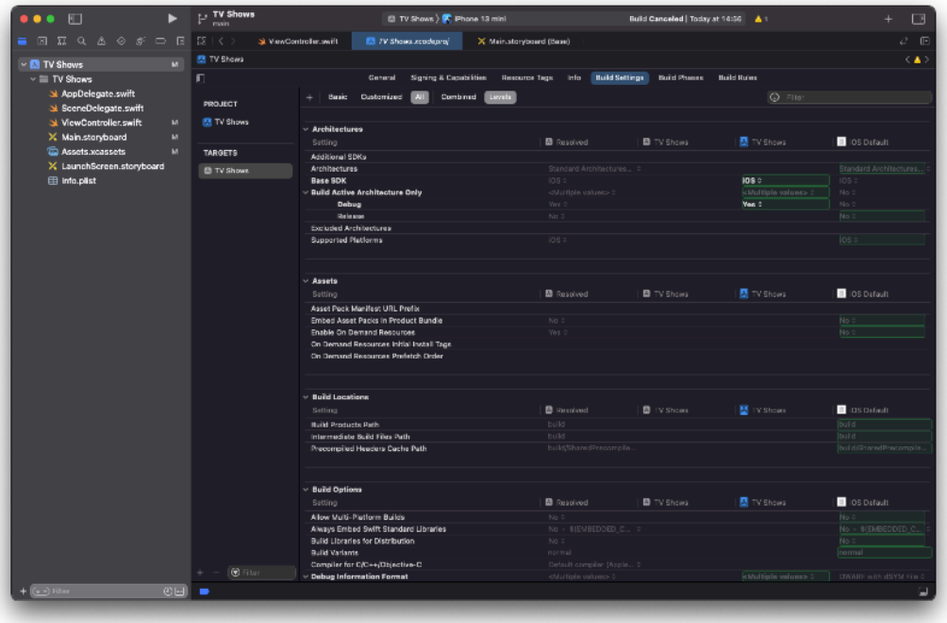
- unique identifier, every app has its own ID
- version - front facing
- build - similar to version but for developers to track current build
- supported ios version
- iPadOS, just scaled version of iOS
- mostly portrait
- launch screen, main screen
- icon



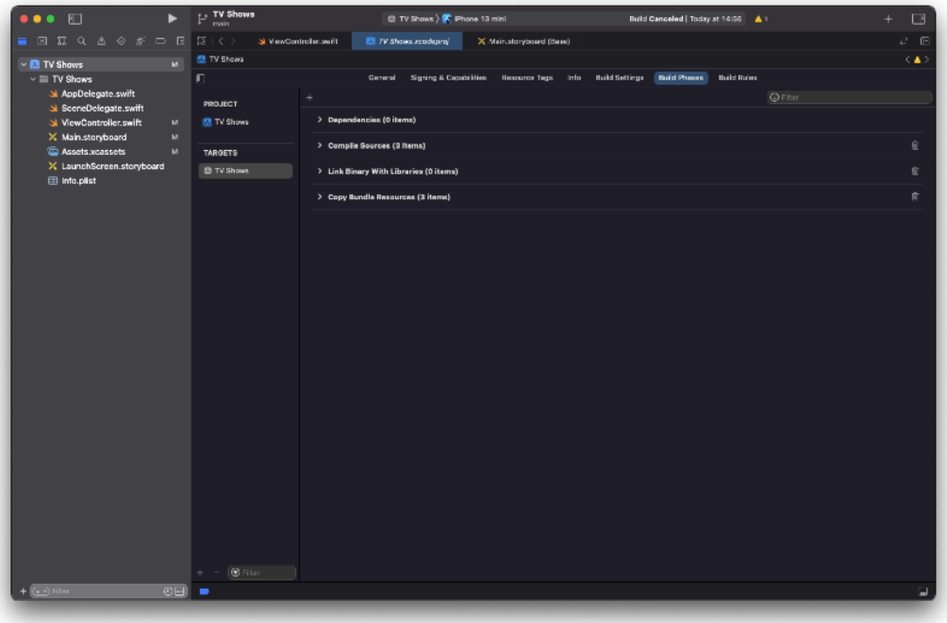
- this is added security to your app
- before sending to the store, your app is signed with certificate
- it guarantees that app you created is really your app
- if you want to run it on your device ping us because we'll help you with the setup
- you need a team/AppleID account to run on a device



- basic settings for the app



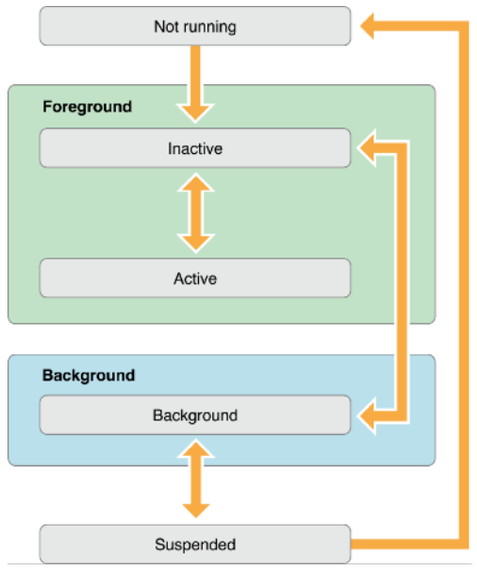
- complex settings for the app



- some build related mumbo jumbo, not important for the course

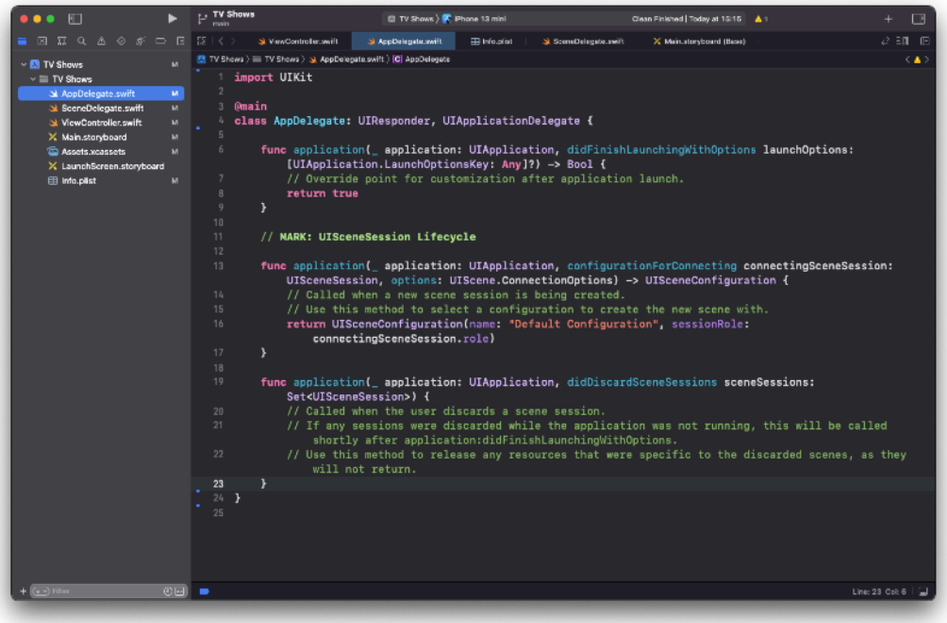
02

App lifecycle



neverstop

- pretty important stuff to understand and also pretty simple
- your applications will go trough various states during its lifecycle
- we can define the lifecycle from the moment you start the app up until you kill it
- one example of how you can leverage this:
    - background fetch of some user data, so that each time user brings the app into foreground he will have all of the latest data
    - maybe when app is going to background, you want to save some stuff
    - etc.

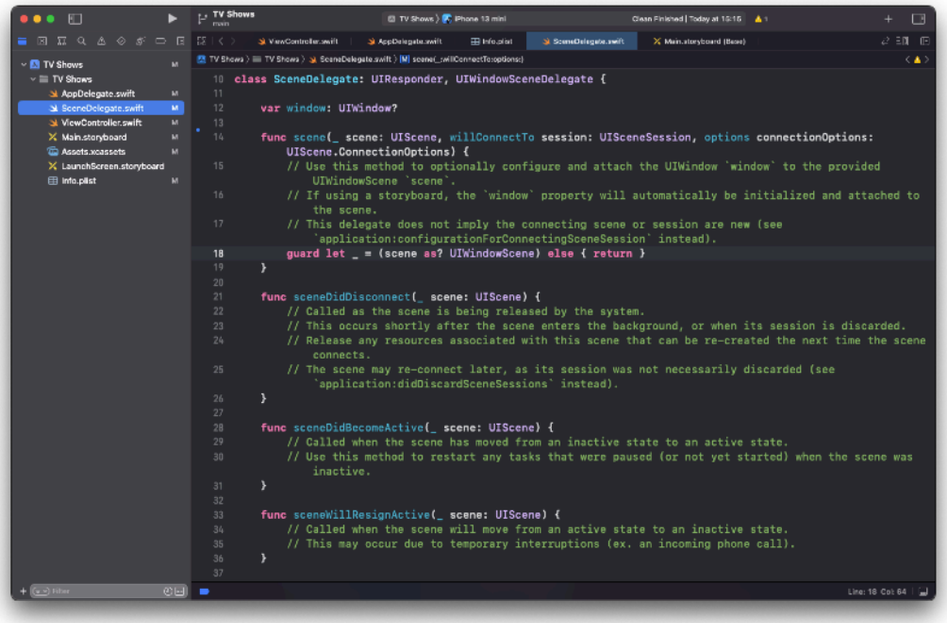



```swift
import UIKit

@main
class AppDelegate: UIResponder, UIApplicationDelegate {

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
        [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        // Override point for customization after application launch.
        return true
    }

    // MARK: UISceneSession Lifecycle

    func application(_ application: UIApplication, configurationForConnecting connectingSceneSession:
        UISceneSession, options: UIScene.ConnectionOptions) -> UISceneConfiguration {
        // Called when a new scene session is being created.
        // Use this method to select a configuration to create the new scene with.
        return UISceneConfiguration(name: "Default Configuration", sessionRole:
            connectingSceneSession.role)
    }

    func application(_ application: UIApplication, didDiscardSceneSessions sceneSessions:
        Set<UISceneSession>) {
        // Called when the user discards a scene session.
        // If any sessions were discarded while the application was not running, this will be called
            shortly after application:didFinishLaunchingWithOptions.
        // Use this method to release any resources that were specific to the discarded scenes, as they
            will not return.
    }
}
```

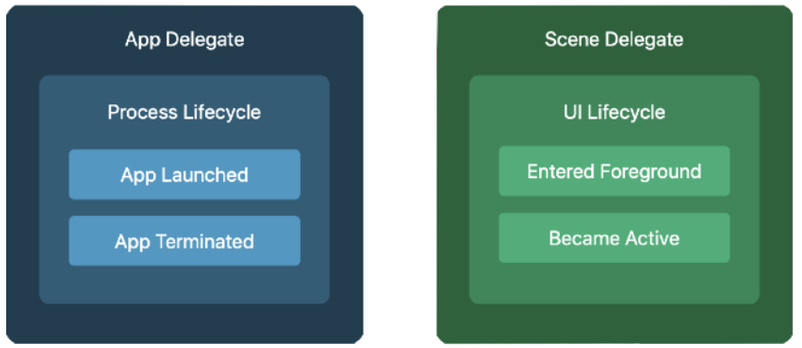

- AppDelegate.swift
- You can look at this file like an entry point in your application (think main.c)

- About the name:
  - more specific *Delegate* part, just to note that it follows the delegate pattern
  - delegate pattern is used by a lot of iOS frameworks
  - most new components will use closure based pattern

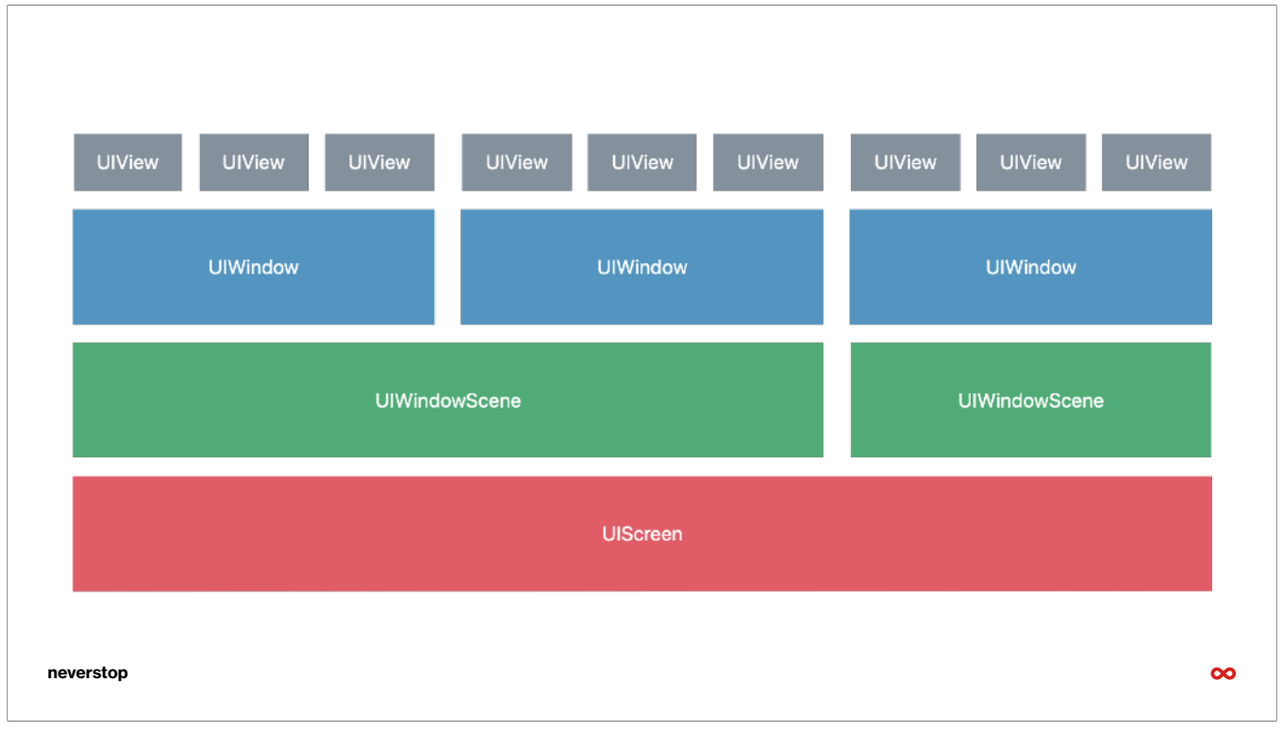
- Delegate pattern:
  - in our case of AppDelegate, means that the class AppDelegate will be the class that the system will delegate some event to
  - system will offload the work to our class, it will say, hey, you will be going to background, do what you like
    - in more technical terms, system (OS) will declare some sort of protocol/interface which will then define certain events
    - if we want to be notified of these events, we need to conform to that protocol/interface
    - and we need to tell the system that we will be its delegate



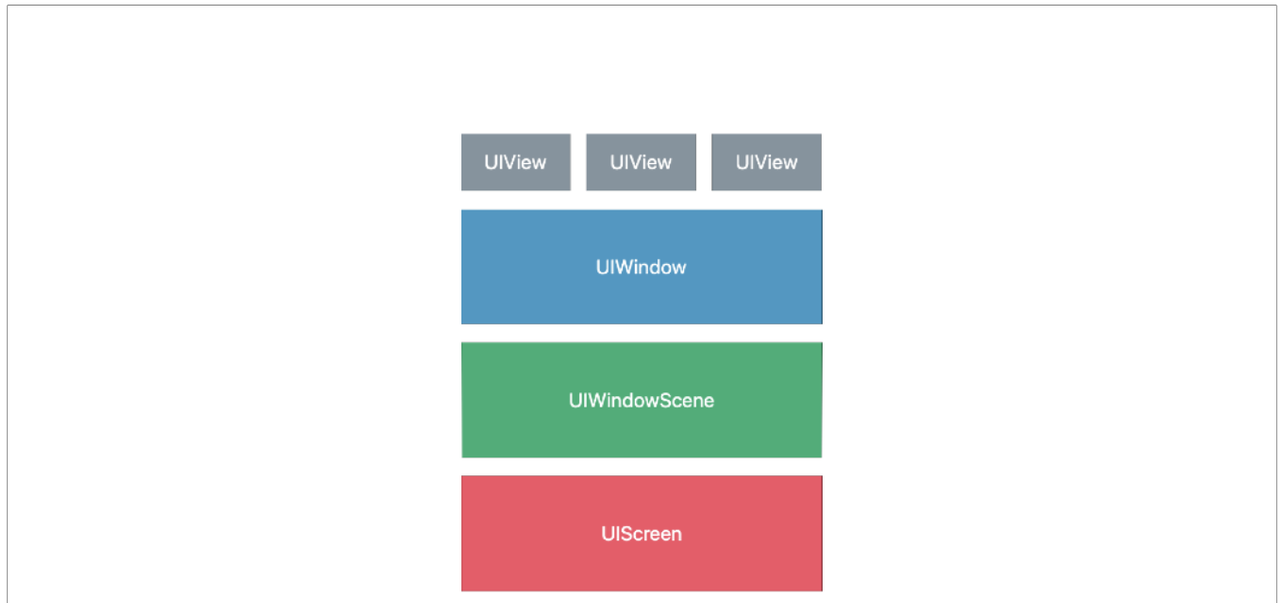
- SceneDelegate.swift
- You can look at this file like an UI entry point in your application (think main.c but for UI setup)
- willResign - save some state, for example game state



- App Delegate is responsible for process lifecycle
- Scene delegate is responsible for UI lifecycle
- But to understand why, next slide...



- iOS and iPadOS support multiple app scenes
- they share the same app process, and app delegate
- but...



- for this course we won't use more than one UIWindowScene, nor UIWindow for that matter
- but it is good to know what's behind

03

MVC

A guideline for app architecture.

neverstop

- Follows MVC pattern (Model View Controller)
    - A bit weird implementation on Apples part, because they glued View + Controller
    - View -> Show UI
    - Controller -> Reacts on UI events

- Can lead to an issue know as MassiveViewController, because your first instinct is to put everything in it

- During this course we will put everything in it :D

- In Storyboards, before you can put any UI element on the canvas, you need to add UIViewController element

Model View Controller

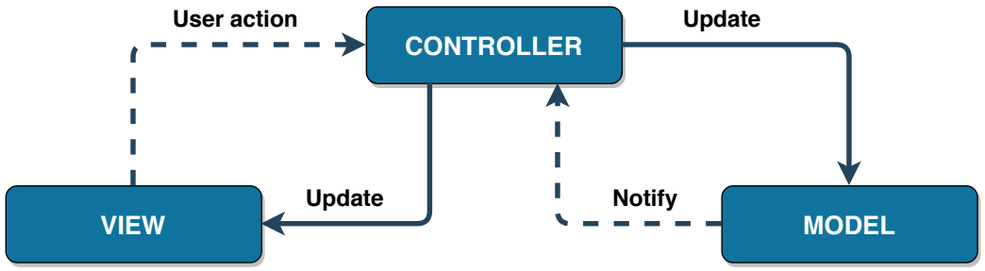
- **Model**
  - encapsulates app's data
  - contains logic to manipulate that data
  - networking data, persisted data, constants...
- **View**
  - shows information to the user and allows interaction
  - Should not contain any business logic
  - UIView and subclasses
- **Controller**
  - bridge between the other two layers
  - least reusable
  - ***UIViewController***

- Follows MVC pattern (Model View Controller)
  - A bit weird implementation on Apples part, because they glued View + Controller
  - View -> Show UI
  - Controller -> Reacts on UI events

- Can lead to an issue know as MassiveViewController, because your first instinct is to put everything in it

- During this course we will put everything in it :D

- In Storyboards, before you can put any UI element on the canvas, you need to add UIViewController element



neverstop

- Follows MVC pattern (Model View Controller)
  - A bit weird implementation on Apples part, because they glued View + Controller
  - View -> Show UI
  - Controller -> Reacts on UI events

- Can lead to an issue know as MassiveViewController, because your first instinct is to put everything in it

- During this course we will put everything in it :D

- In Storyboards, before you can put any UI element on the canvas, you need to add UIViewController element

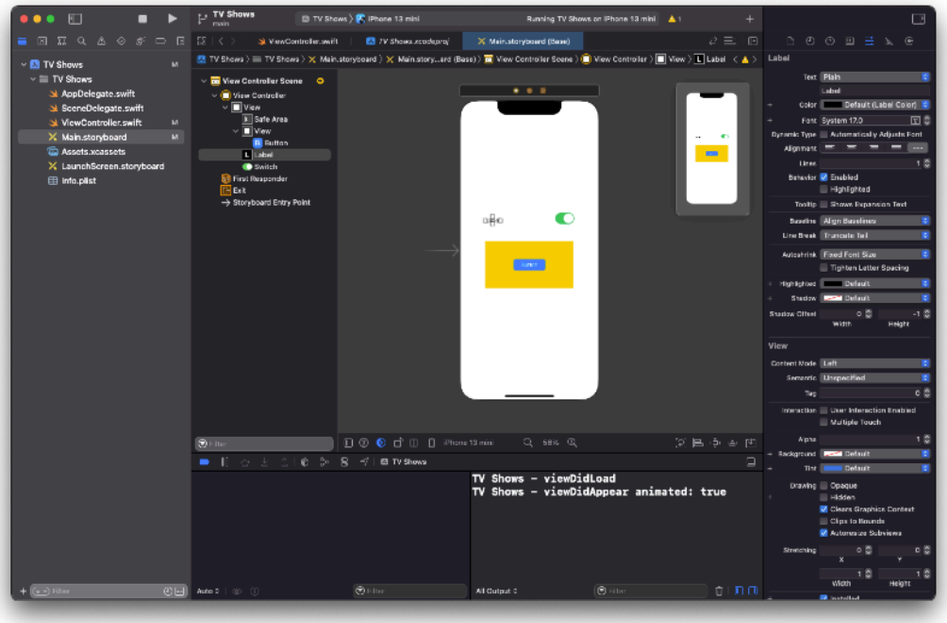
04

UIViewController

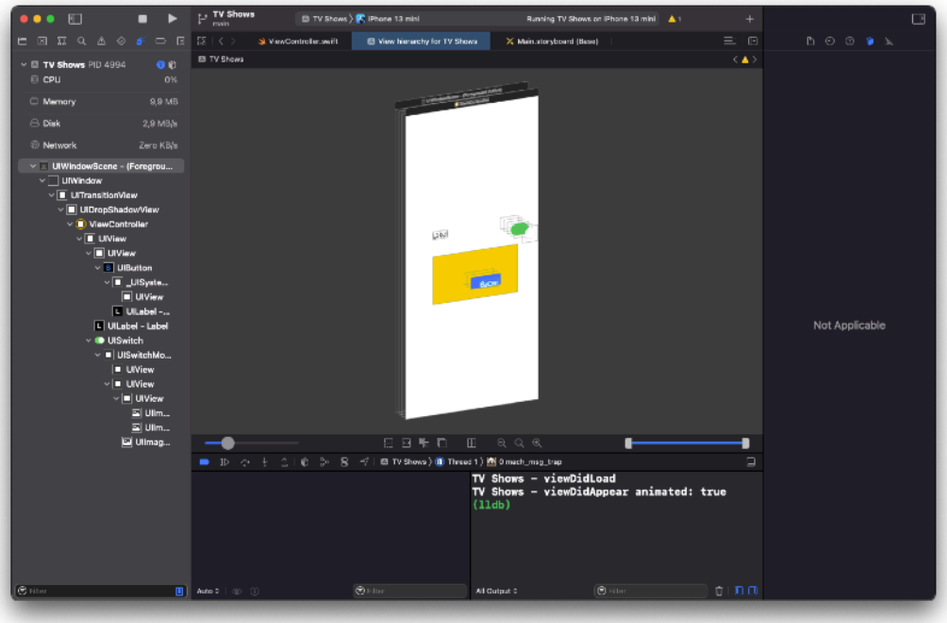
- name of the class that you want to subclass

The brick

- the building block of your apps
- usually represents one screen
- most of your work will be in here
- usually used in conjunction with a storyboard or a xib
- storyboards need ***UIViewController*** as its first element

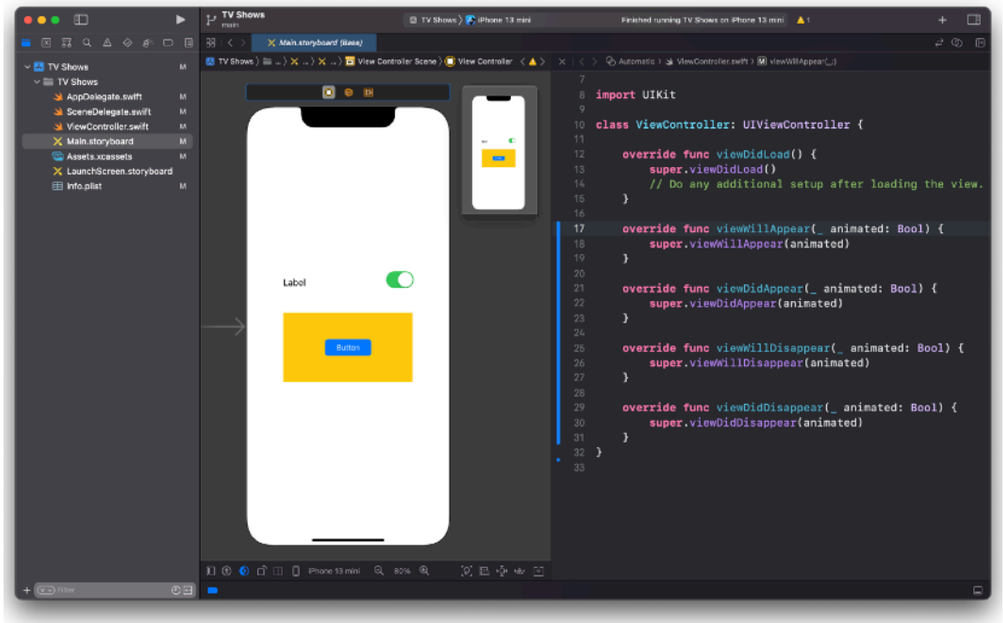

TV Shows - viewDidLoad
TV Shows - viewDidAppear animated: true

neverstop



neverstop

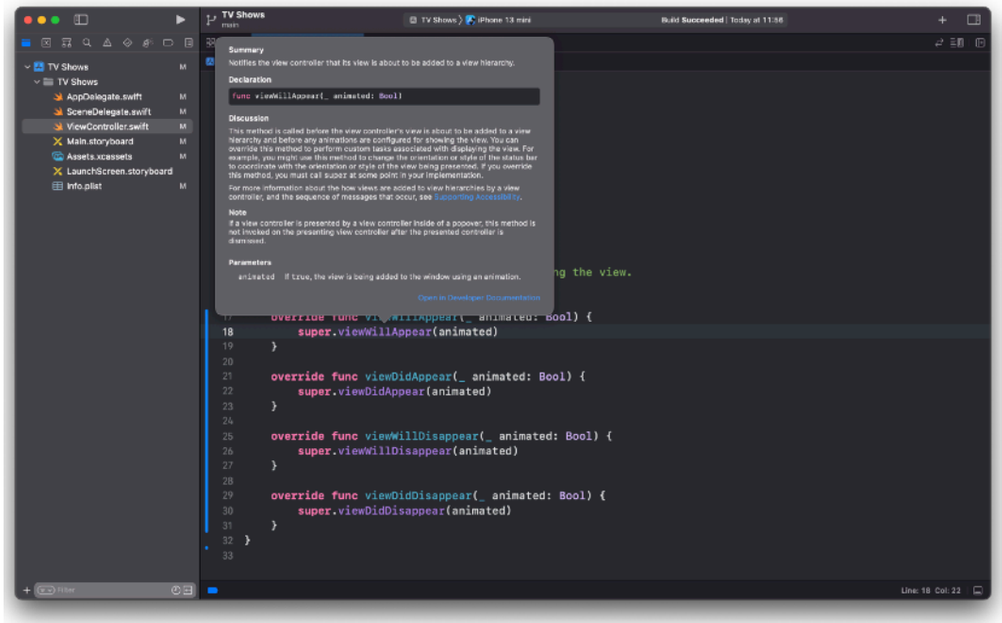
- As you can see here, all of our elements are inside the view controller
  - look at the view hierarchy on the left ;)



neverstop

- It is really important for you to understand this:
  - If you build your UI in code, you will only have your UIViewController subclass
  - If you build your UI using IB, you will have your UIViewController subclass and your UIViewController element
    - and you need to hook them up

- Storyboards are represented as XML



neverstop

- Super shortcut: OPTION + LEFT CLICK
  - you will see little question mark (?) when hoovering with mouse over functions names
  - use this to verify you assumptions

- Here you can see that the UIViewController also has lifecycle, similar to application lifecycle, but more contained

05

Memory management

Memory management

- **MMR** - manual memory management
  - malloc/free
  - C style
- **GC** - garbage collector
  - Java, C#, Python...
- **ARC** - automatic reference counting
  - retain/release
  - ObjC, Swift
  - similar to C++ with smart pointers (shared, unique, weak, auto)

- MMR
  - Acquire memory with malloc, release with free
  - Used in C
  - Error prone
    - very easy to forget free
    - sometimes is not clear when to call free

- GC
  - Acquire memory with new and constructor – Java like
  - Very safe, memory is handled automagically
  - Everything can be detected for garbage collection, including reference cycles
  - Processing is done in the background (mostly)
  - Disadvantages:
    - Non-deterministic finalization (deconstructor execution is not predictable)
    - Under low memory conditions, garbage collection may halt thread execution and slow down the app

- ARC
  - Deterministic finalization (deconstructor/dealloc is predictably run)
  - Objects are deallocated immediately when they are no longer needed
  - Disadvantages:
    - Can not deal with reference cycles without developer intervention.

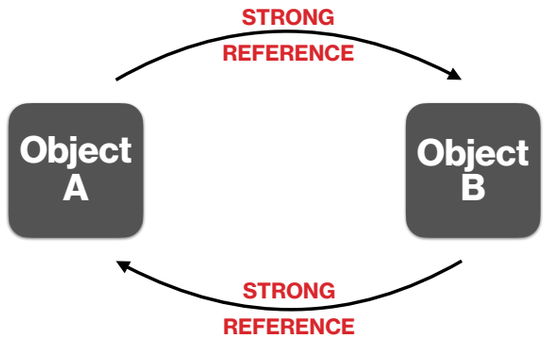
Automatic Reference Counting

- +
  - not part of the runtime
  - deterministic reclamation of memory
  - no background or runtime processing
- -
  - cannot cope with retain cycles or memory leaks

- WHAT:
  - compiler injects reference tracking and clean-up code, similar to what a developer would do
  - deterministic reclamation of memory, at the time when the object goes out of scope
  - since there is no background or runtime processing like in garbage collected languages (Java, C#…) - it requires less CPU and RAM, making it efficient on mobile devices
  - cannot cope with retain cycles/object graphs – gets stuck with an object graph, whereas a GC would look for an external reference to an object graph, and if not found would clear up the entire object graph
  - more prone to memory leaks based on the quality of code written

HOW:
  - this means that it only frees up memory for objects when there are zero strong references to them
  - every strong reference will increment count by 1, when dealloc decrement count by 1
  - it is fast
  - it is predictable

Retain cycle



- WHAT:
  - compiler injects reference tracking and clean-up code, similar to what a developer would do
  - deterministic reclamation of memory, at the time when the object goes out of scope
  - since there is no background or runtime processing like in garbage collected languages (Java, C#…) - it requires less CPU and RAM, making it efficient on mobile devices
  - cannot cope with retain cycles/object graphs – gets stuck with an object graph, whereas a GC would look for an external reference to an object graph, and if not found would clear up the entire object graph
  - more prone to memory leaks based on the quality of code written

HOW:
  - this means that it only frees up memory for objects when there are zero strong references to them
  - every strong reference will increment count by 1, when dealloc decrement count by 1
  - it is fast
  - it is predictable

06

Memory management in Swift

neverstop

ARC modifiers

- **strong**
  - will increment retain count
  - says “I use it and I own it”
- **weak**
  - will NOT increment retain count
  - says "I use it but I don’t own it"
  - must be Optional (e.g. Int?)
- **unowned**
  - will NOT increment retain count
  - says "I use it and I expect it to be there but I don’t care about the ownership"
  - source of runtime crashes (~NULL pointer exception)

- READ THIS:
  - https://www.raywenderlich.com/966538-arc-and-memory-management-in-swift

- SWIFT:
  - we have three ARC modifiers, weak, strong and unowned
  - only to be used with reference types

  - strong is implicit and it will increment retain count by 1

  - weak should be specified, and it will NOT increment retain count by 1
    - when used on property it will require from you to wrap it in using Optional type and then later on to unpack it
    - it must be marked as *var* because it will be set to nil during dealloc

  - unowned should be specified, and it will NOT increment retain count by 1
    - when used on property it will NOT require from you to wrap it in using Optional type, but if *nil* when accessed it will cause runtime crash

- During this course, I would suggest you don’t use *unowned*

- APPLE DOCS:

“Use a weak reference whenever it is valid for that reference to become nil at some point during its lifetime. Conversely, use an unowned reference when you know that the reference will never be nil once it has been set during initialisation.”

Strong

```
class Show {
    let name = "Fringe"
}

class TVShowsViewController: UIViewController {
    let show: Show
}
```

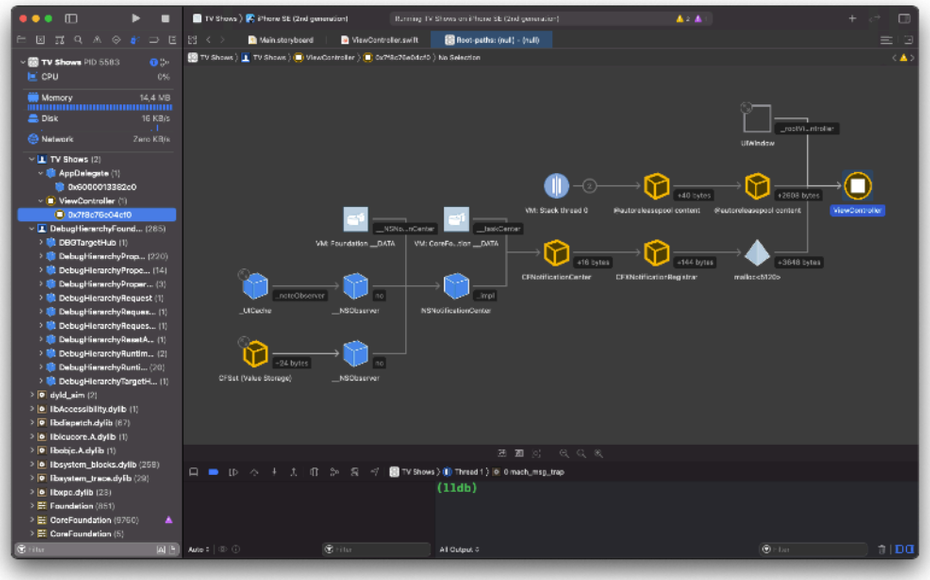
Weak

```
class Show {
    let name = "Fringe"
}

class TVShowsViewController: UIViewController {
    weak var show: Show?
    // weak var show: Optional<Show>
}
```

Unowned

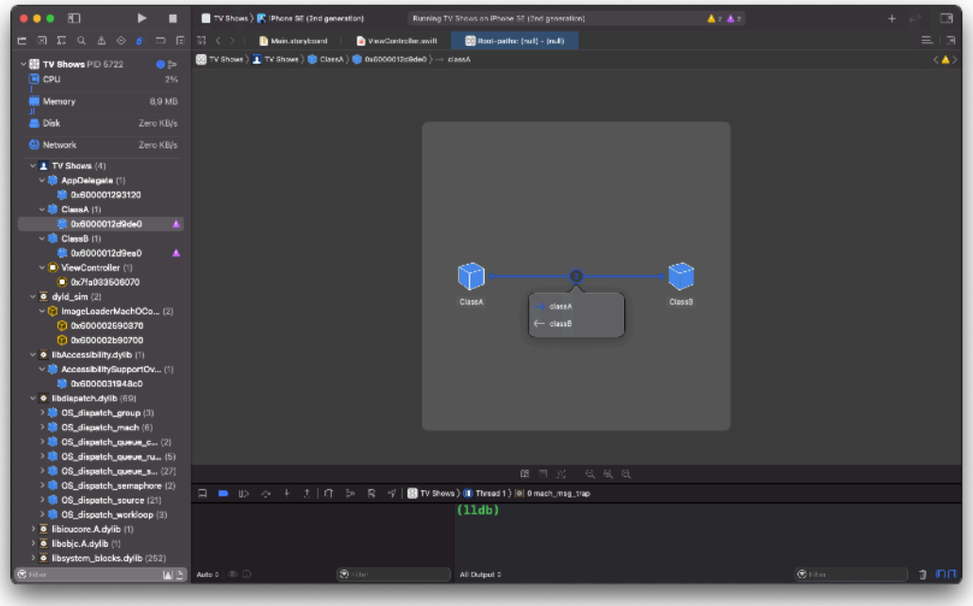
```
class Show {
    let name = "Fringe"
}

class TVShowsViewController: UIViewController {
    unowned let show: Show
}
```

Memory debugger



neverstop

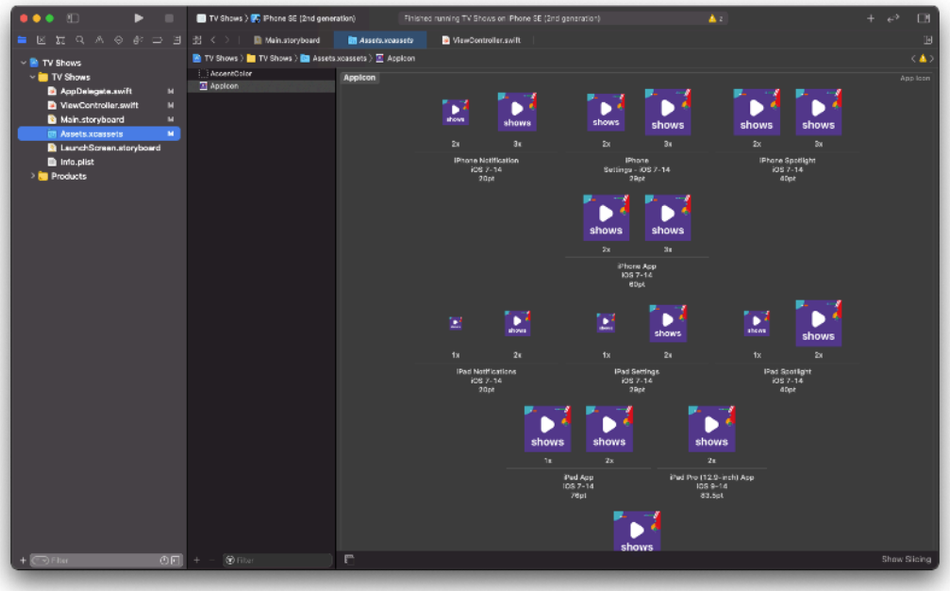
- if you suspect a memory leak you can always use memory debugger to verify your claim, really useful stuff!
- first thing you can do is to look at the left pane, and search for you class name, and look at the number of active instances
    - rapidly growing number can indicate that something is being retained
    - you should start debugger, check numbers, resume app, reproduce steps and then start debugger again ;)
    -


TV Shows
CPU
Memory
Disk
Network
ClassA
ClassB
classA
classB
(lldb)

07

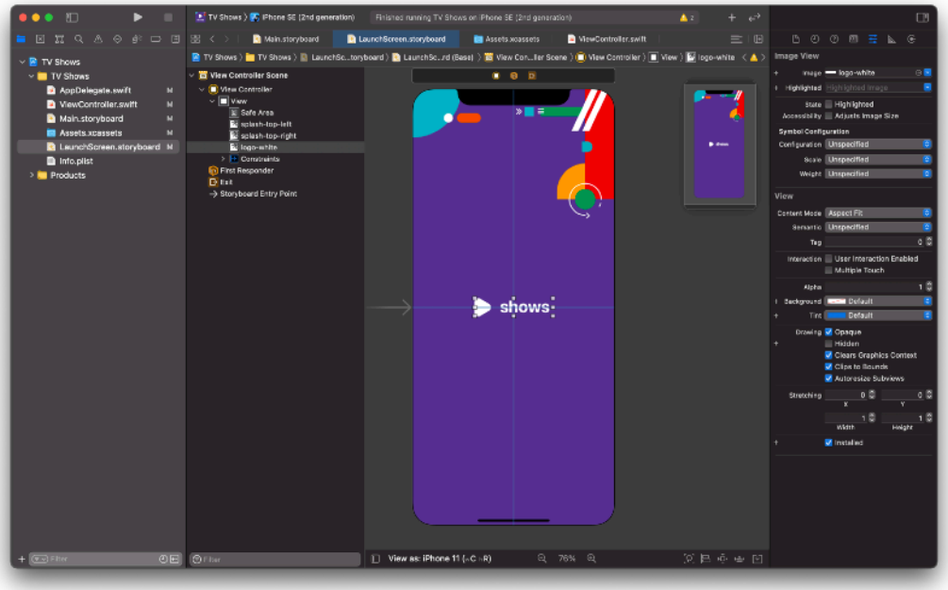
Resources & assets

App Icon



- App Icon needs to be exported for every device and resolution
- but in Xcode 14 that will change, finally!

Launch Screen

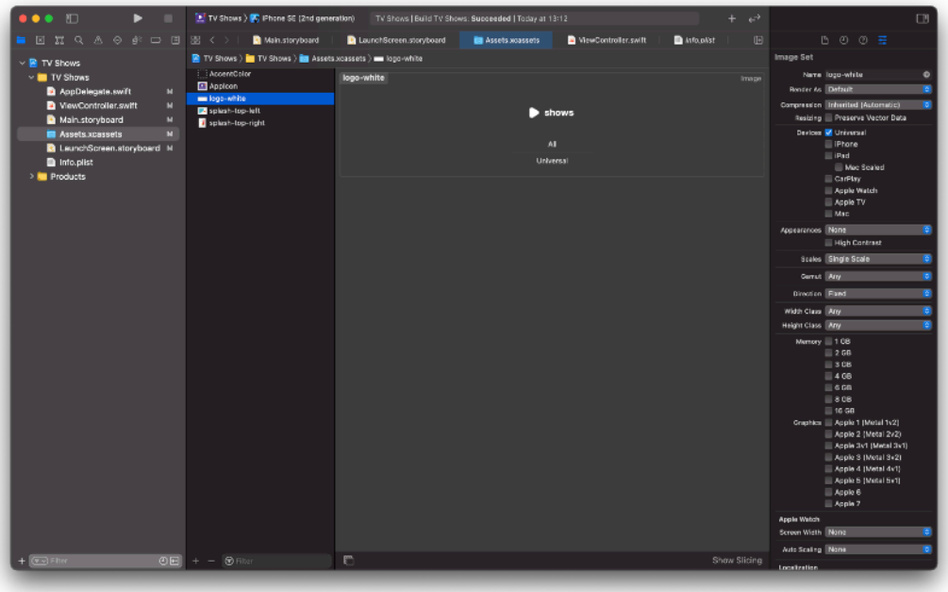


- Launch image can be dynamic (partially)

Asset Catalogs

- collections of images to be used in the app
- can be vector (.pdf) or multiple size images (.jpg, .png)
- images have a suffix of @1x, @2x, @3x
  - depending on the screen size scale
  - @1x is to be deprecated
- use these to group images logically

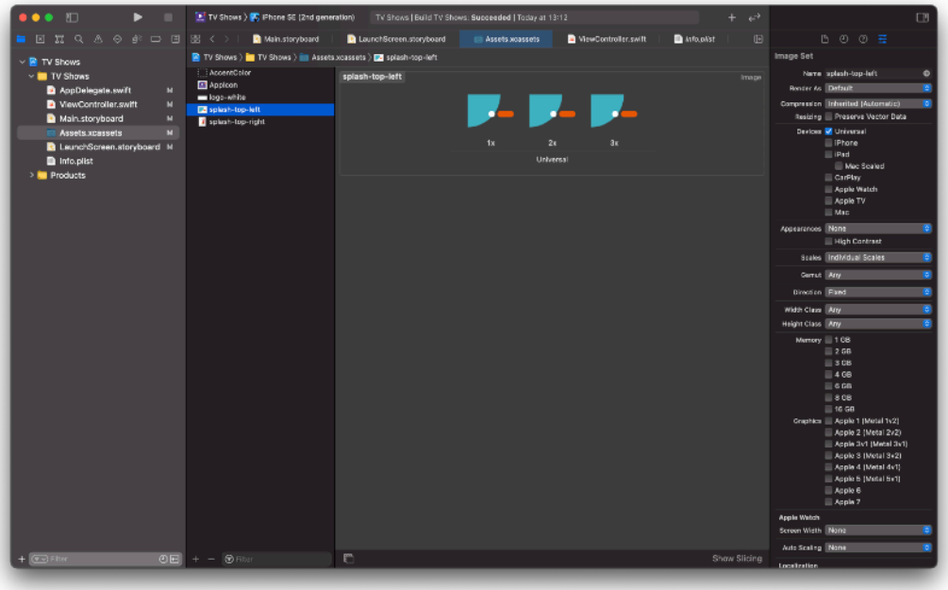
Vector asset - single scale



neverstop

- if you have assets that have shadow on them, there is very good chance that you will not be able to add it as PDF
- PDF is vector based
- you should try to use PDF as much as possible

Multiple scale image



Thank you!

neverstop

Linkedin Instagram Twitter Facebook

∞ INFINUM