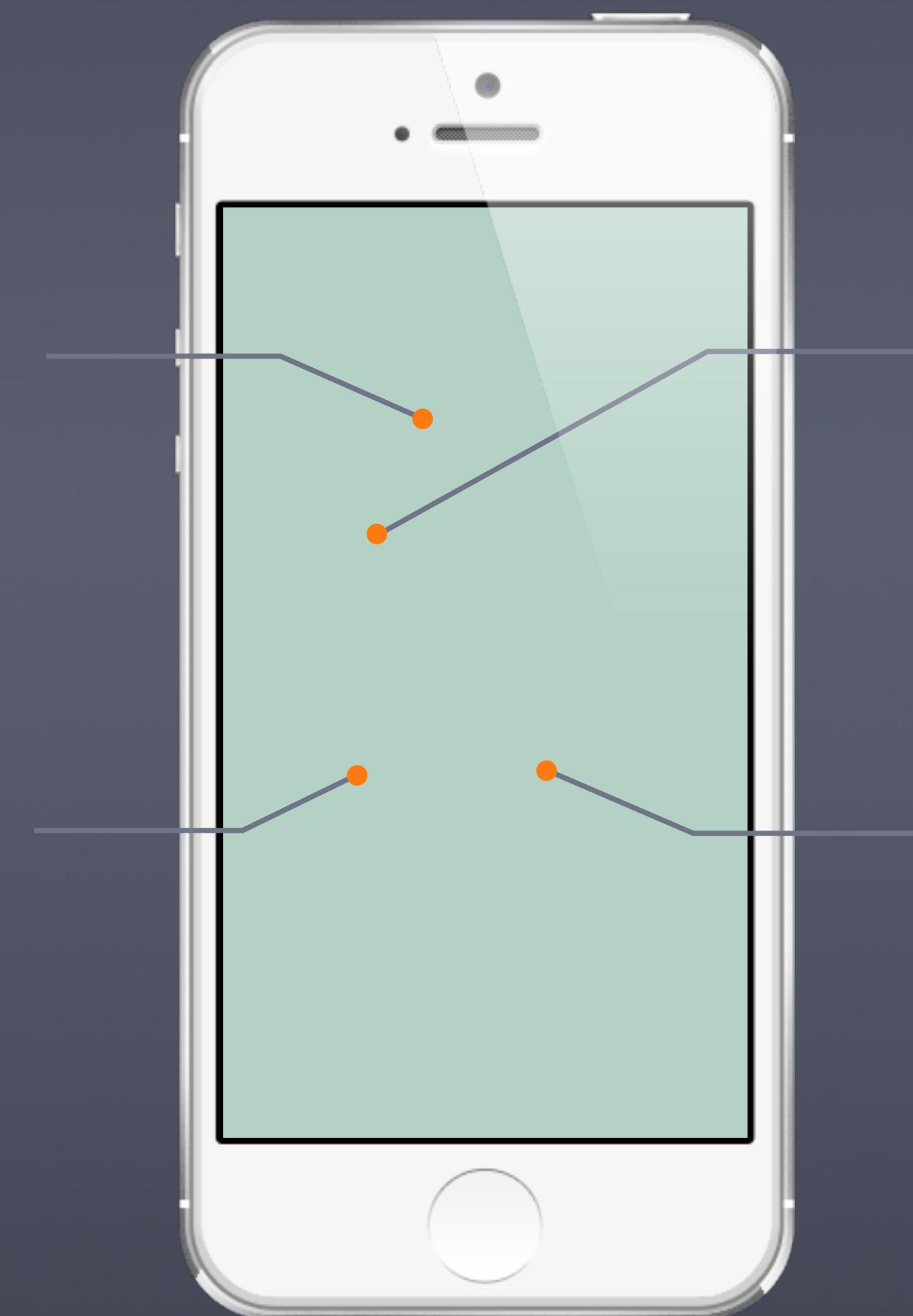


IOS Overview

History, IOS version, IOS Architecture
SDK Overview.

Swift, Objective-C

UIViewController, Navigation Controller, life cycle,
pass data



App Development Life Cycle

Hello World App, Structure of project, Deploying
to an Emulator, Debug app.

IOS User Interface

Layout/Story board, Using resource in app,
Component.

/ iOS Overview



/ iPhone OS 1

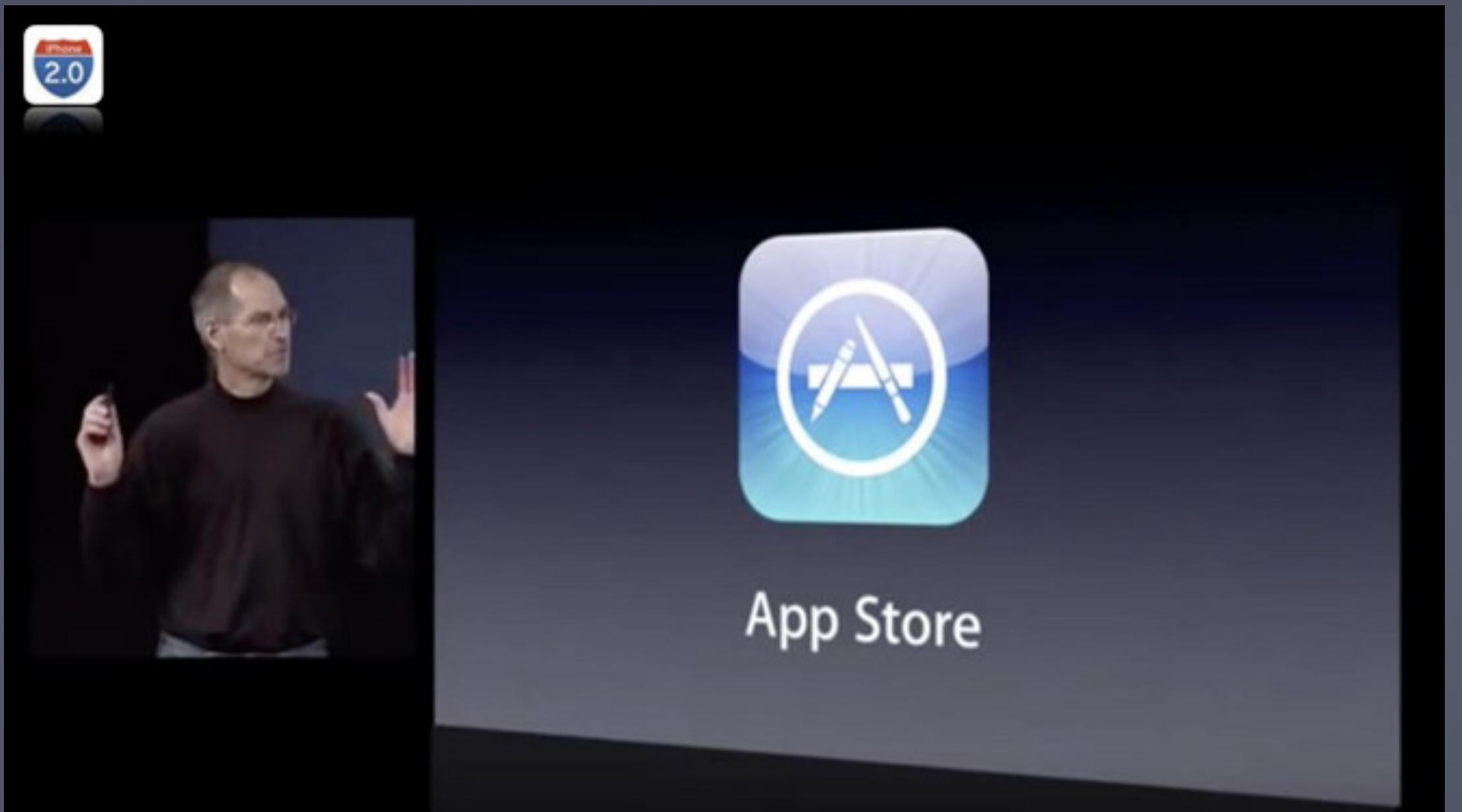
Ngày công bố: 9.1.2007

Ngày phát hành: 29.6.2007



/ iPhone OS 2

- App Store
- Khi ra mắt, có khoảng 500 ứng dụng trên Store
- Ứng dụng iTunes cũng cho phép kết nối thông qua dữ liệu di động, thay vì chỉ nhận Wi-Fi như trước.



/ iOS 3

- Ra mắt máy tính bảng Ipad
- Đổi tên hệ điều hành từ iPhone OS thành iOS

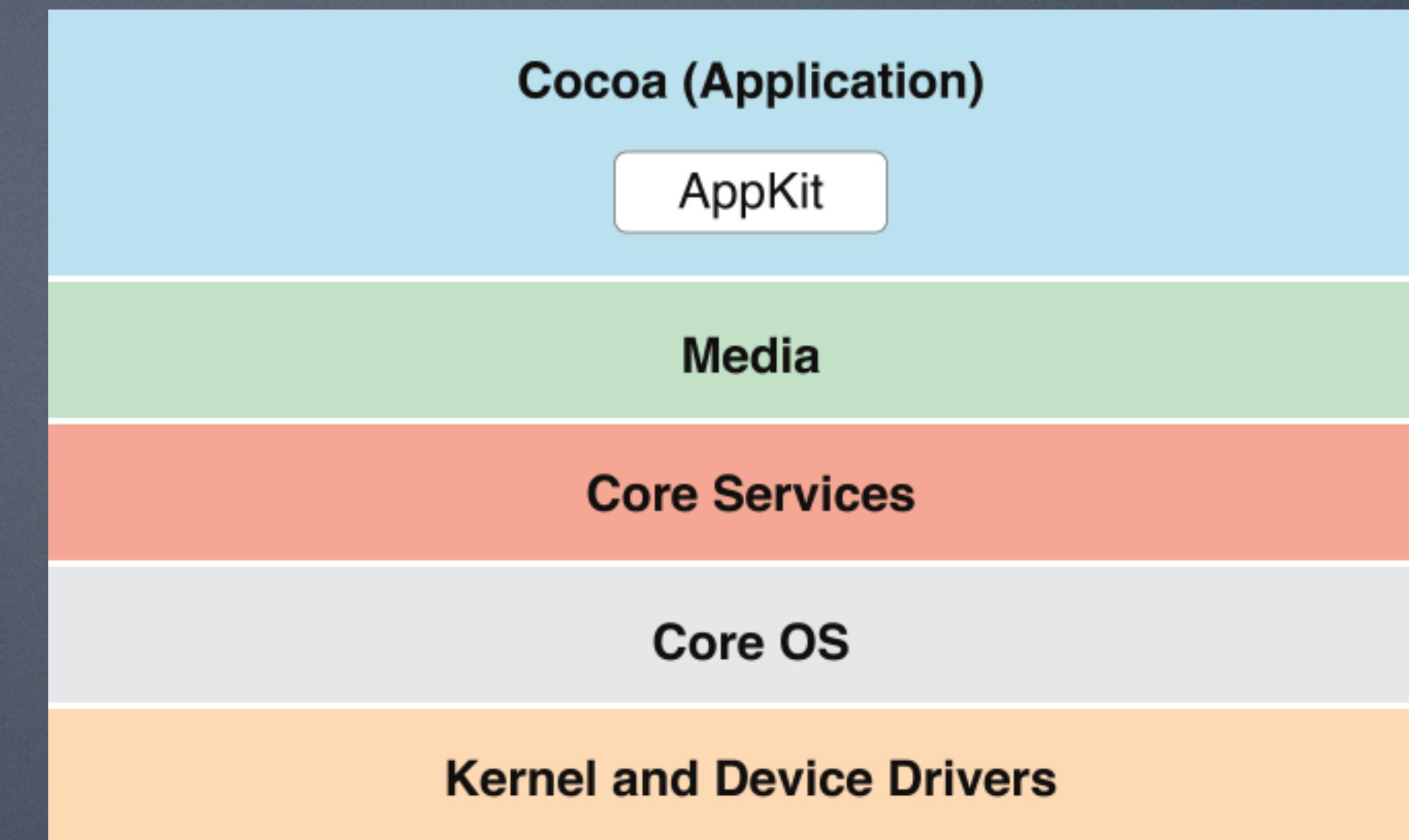


/ iOS 14



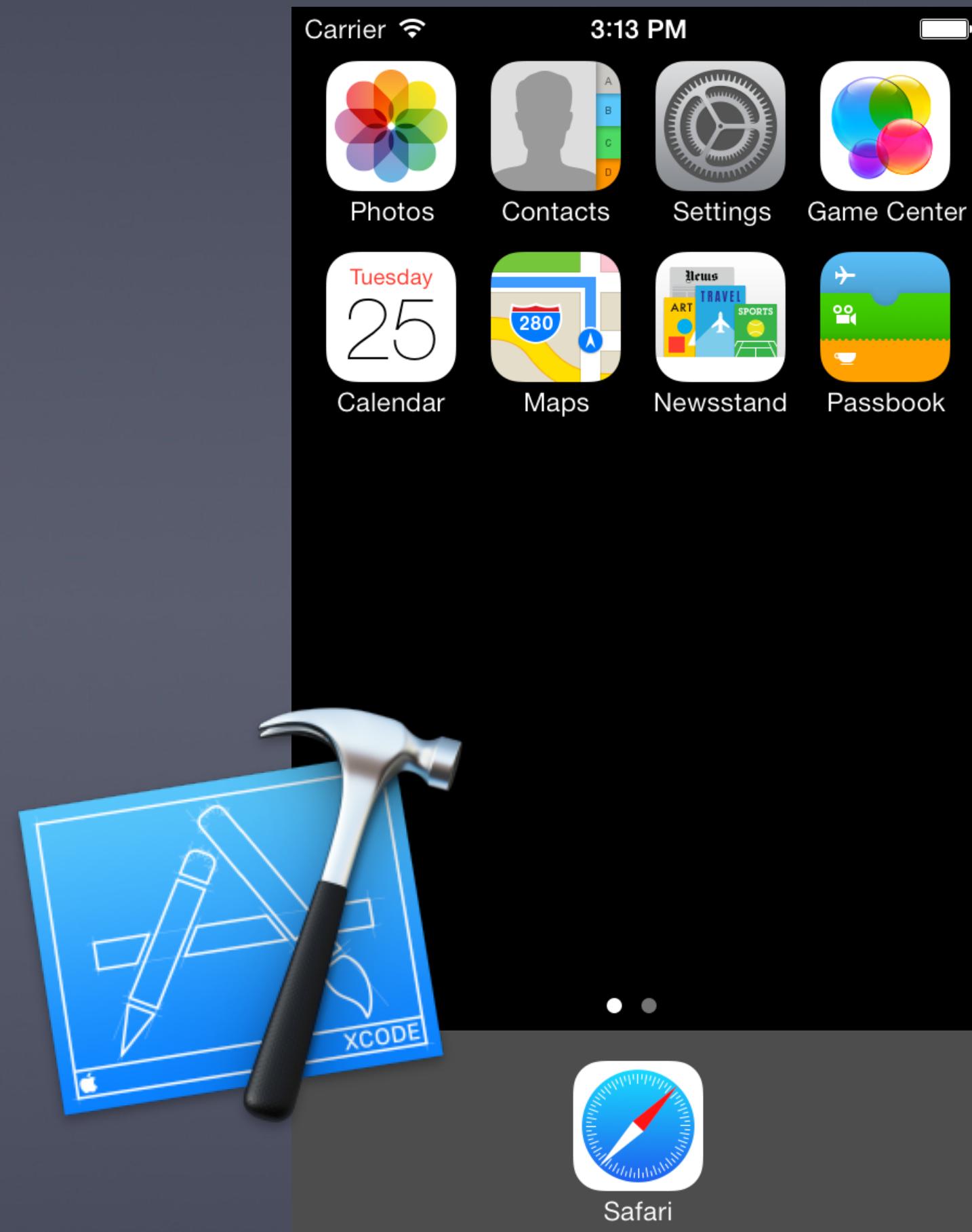


/ iOS Architecture



/ iOS Development

1. OSX
2. Xcode
3. Simulator
4. Apple Dev Account



/ MAC OS X



/Hackintosh



/Virtual machine

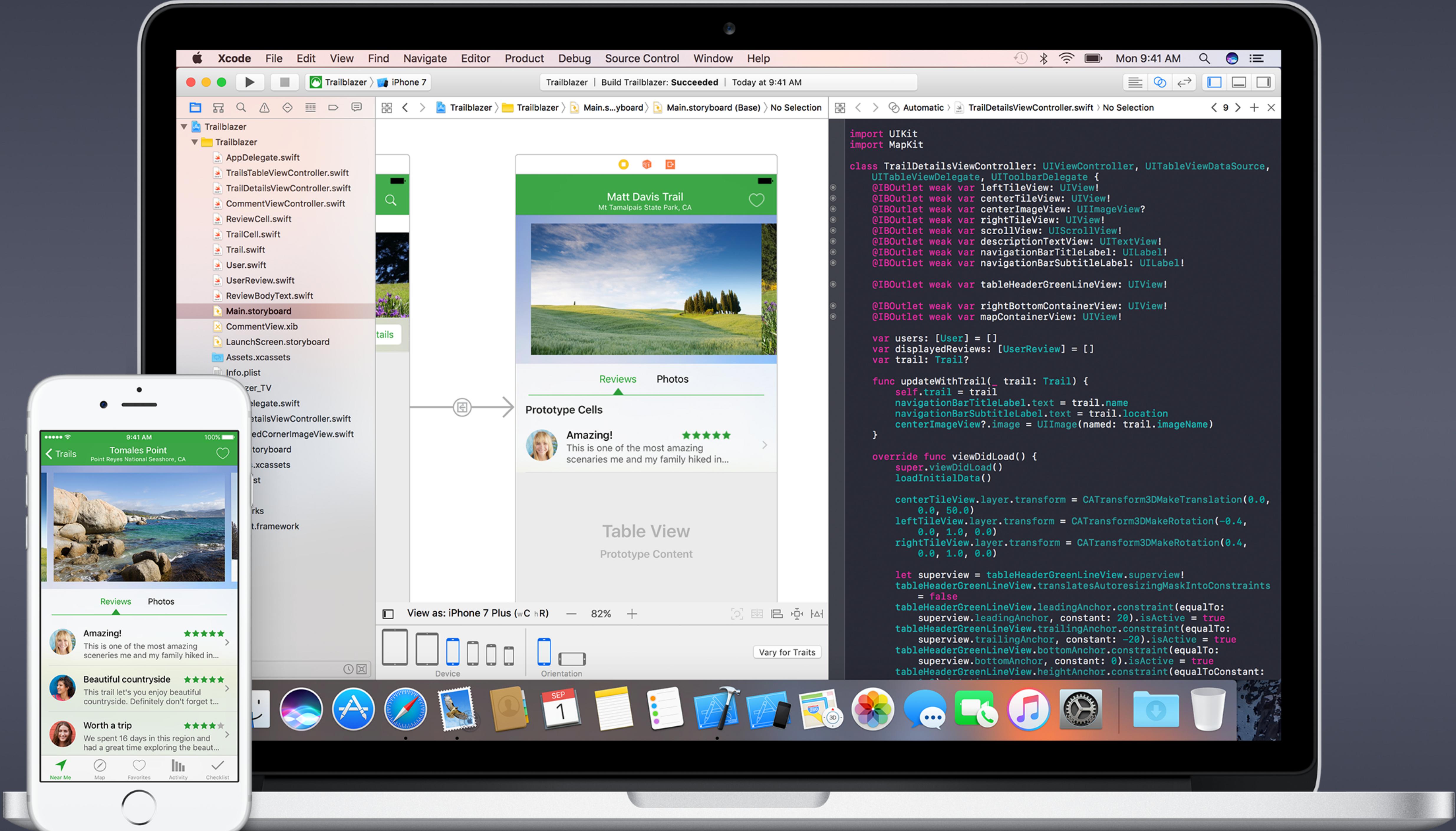
VMware

Virtualbox

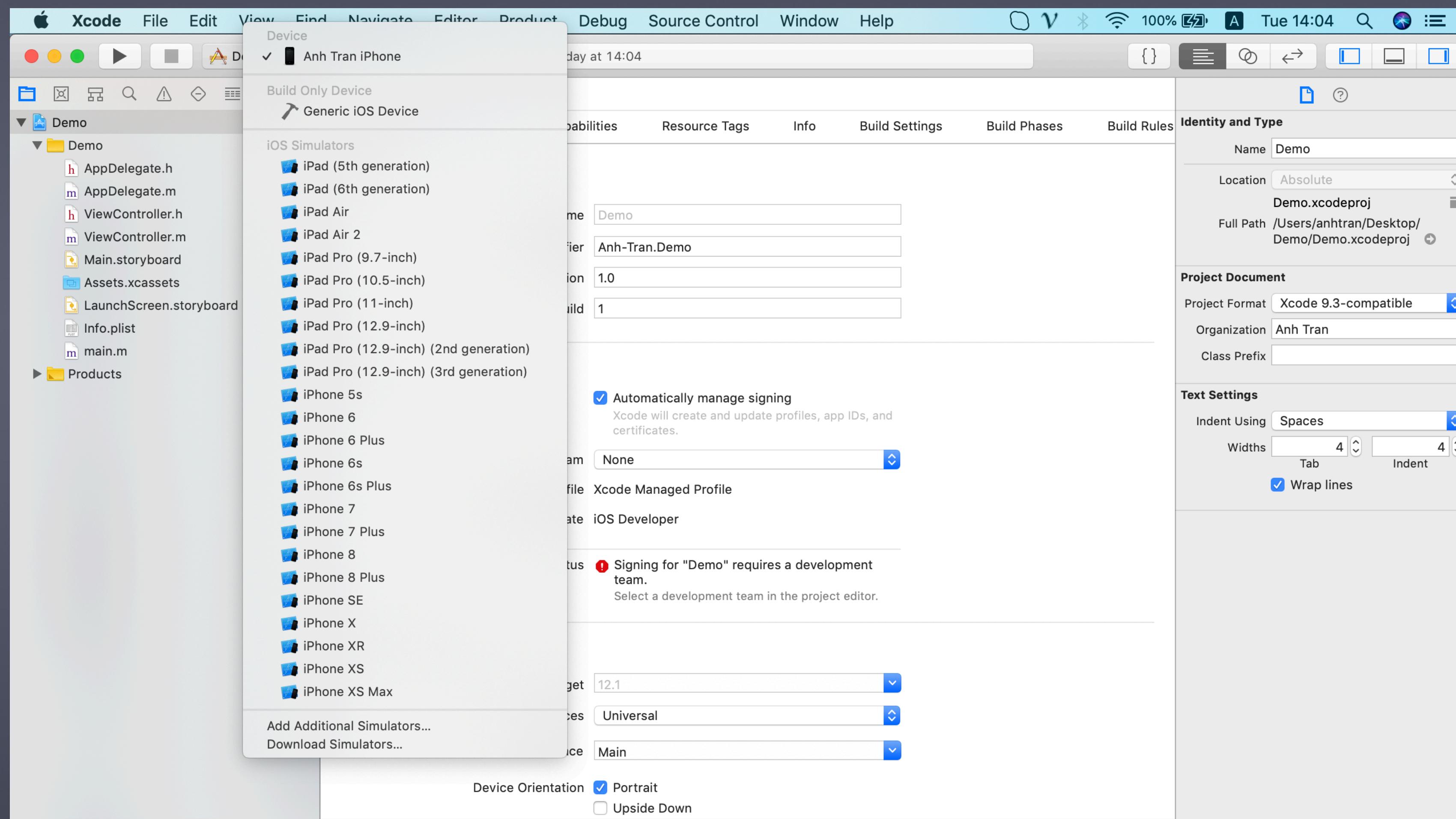


/ Xcode

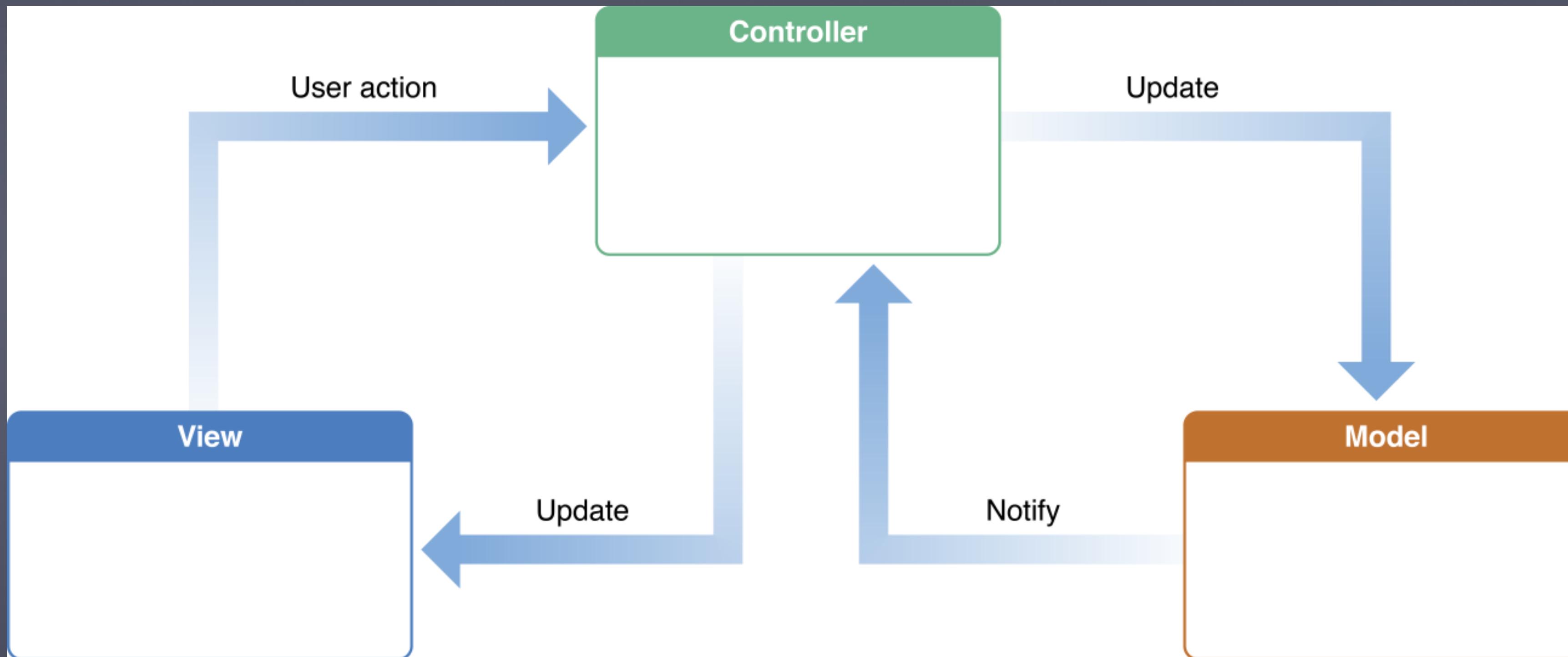


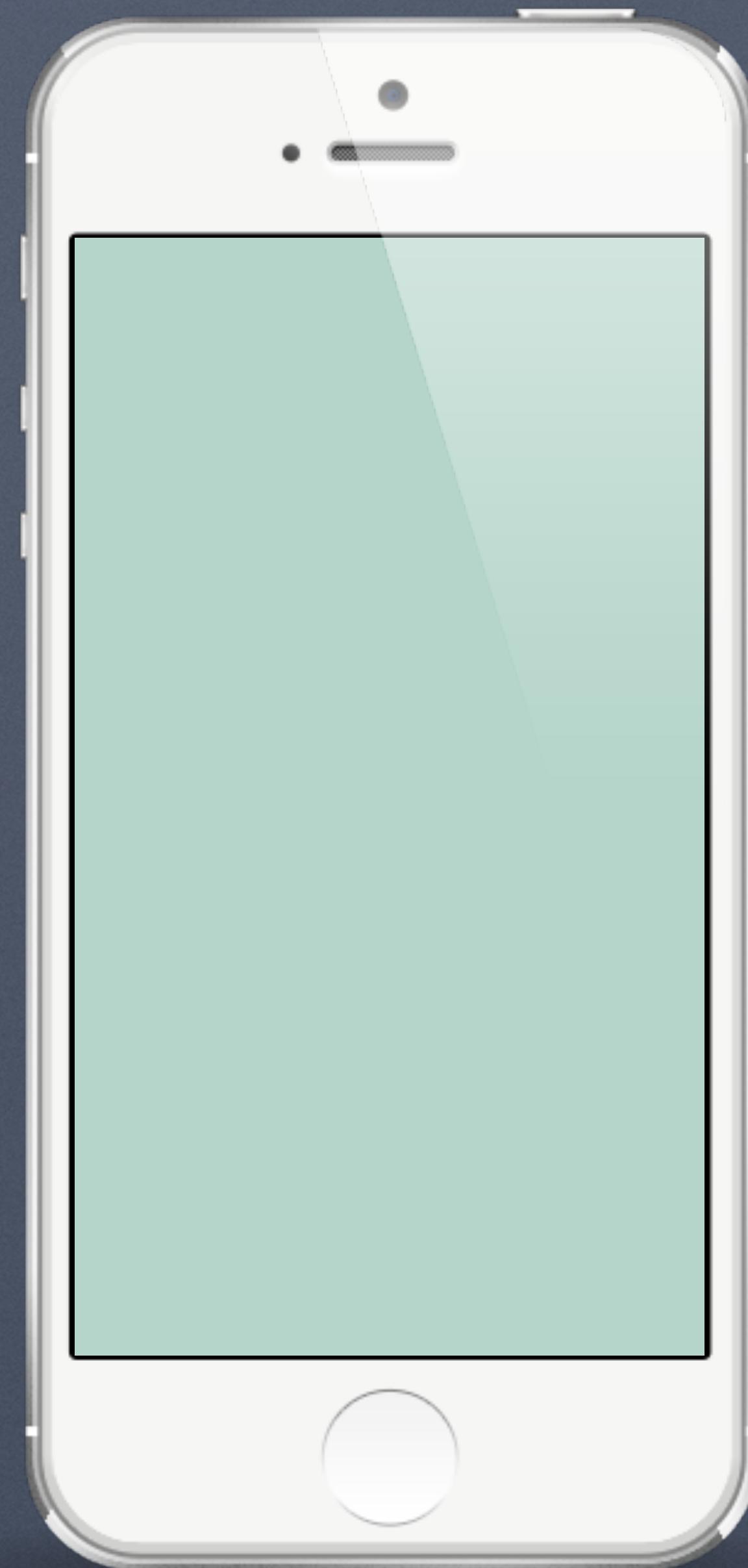


/ iOS simulator

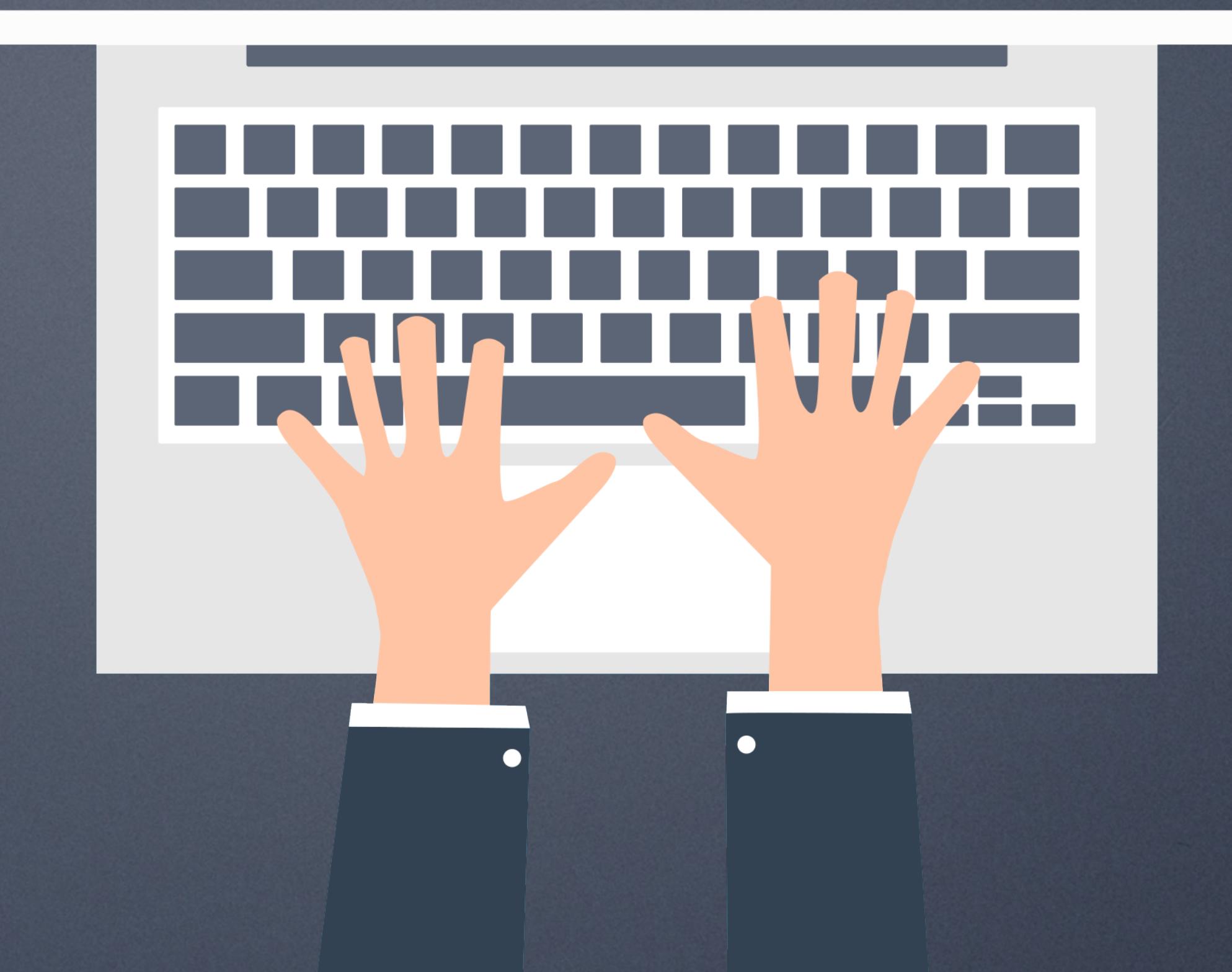


/ MVC



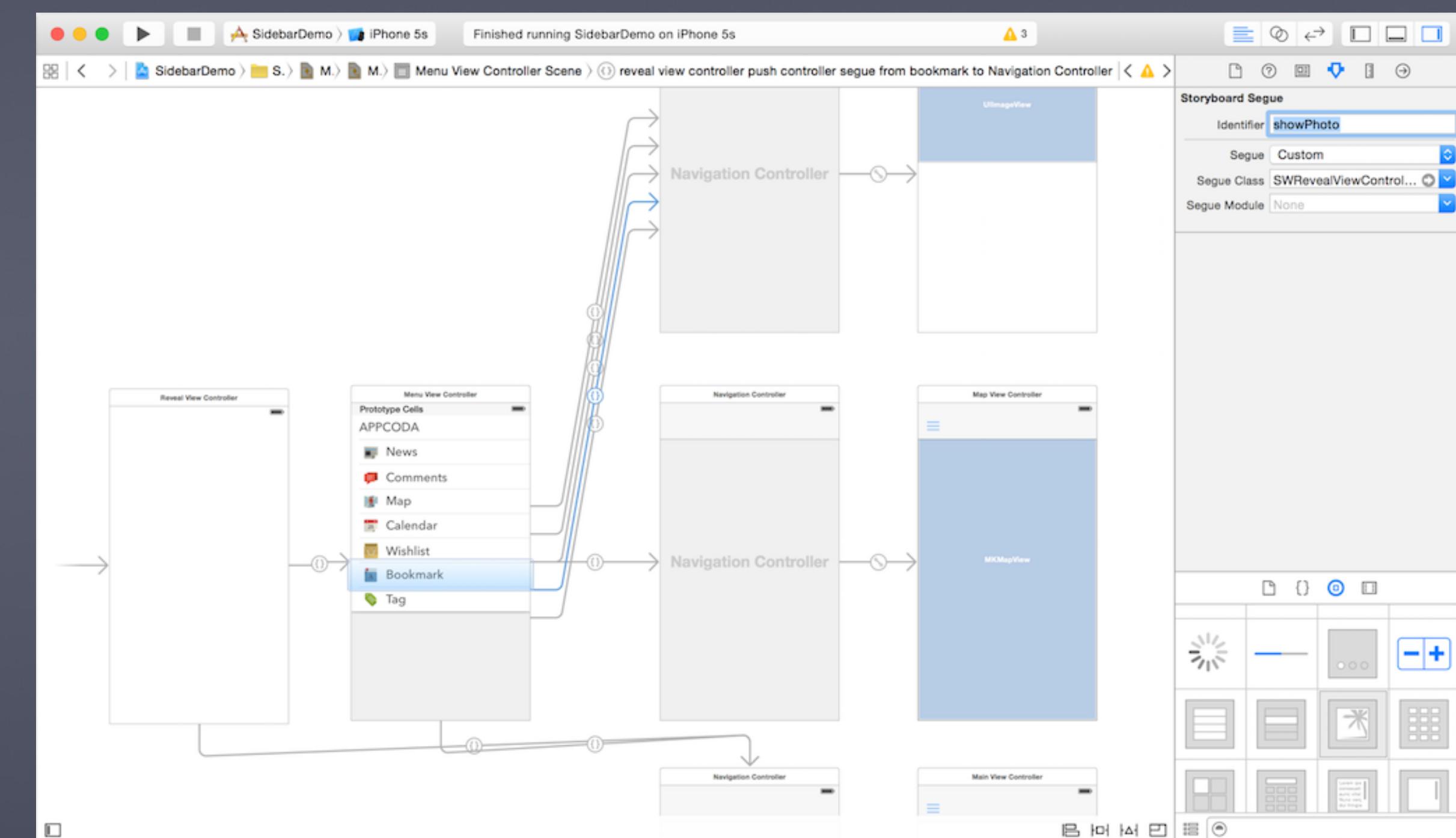


Demo



/ StoryBoard

- Xuất hiện cùng với iOS 5
- Một cách thức tổ chức – quản lý các file giao diện, có cái nhìn tổng quát về toàn bộ màn hình của ứng dụng.



/ Story Board

Ưu điểm

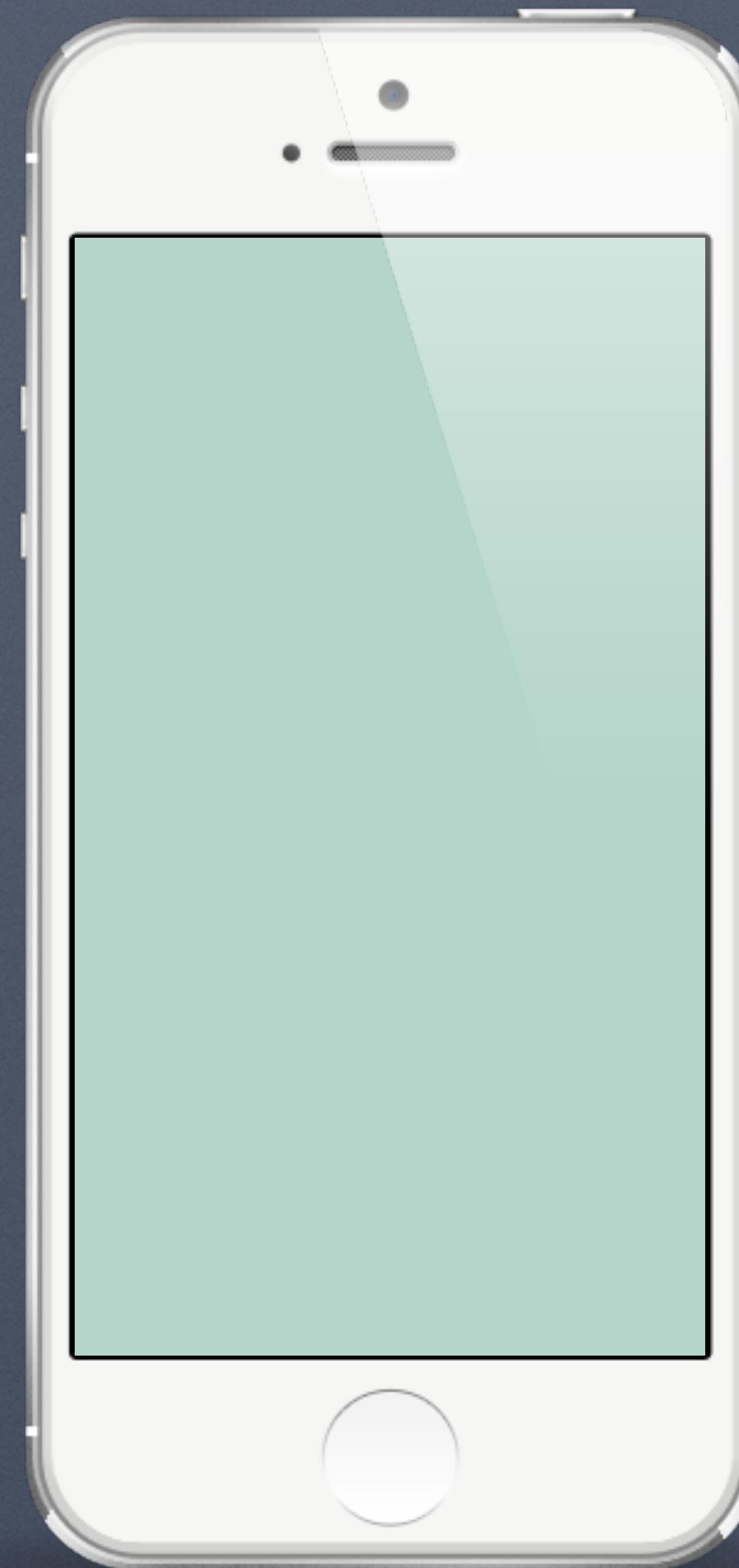
- Cho phép lập trình viên có cái nhìn tổng quát về chương trình, dễ dàng quan sát luồng UI.
- Dễ dàng kiểm soát các chuyển động của màn hình.Tiết kiệm lượng code → giảm thời gian → tiết kiệm chi phí.

Nhược điểm

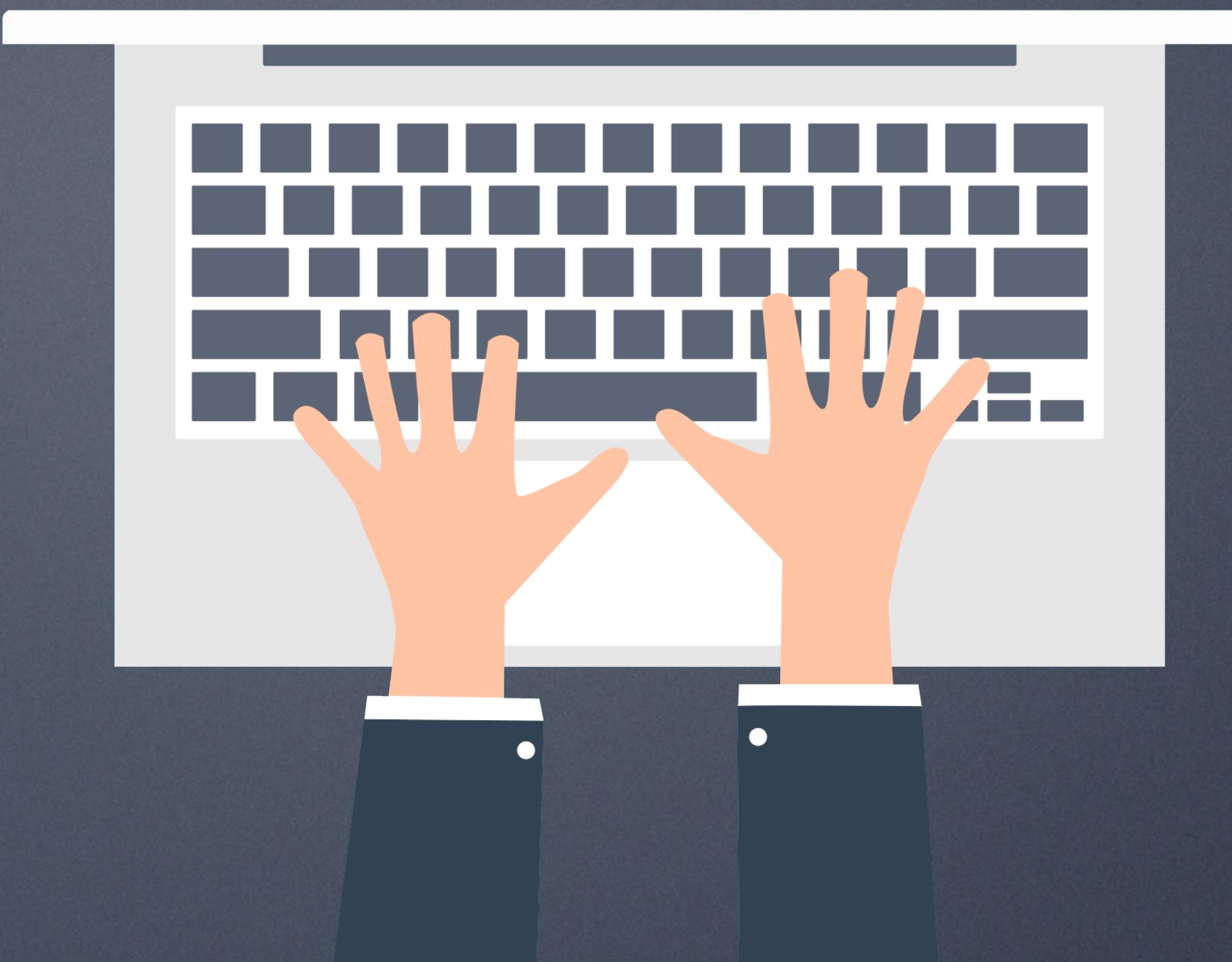
- Trong quá trình làm nhóm, dễ bị conflict.
- Nếu bố trí không tốt, đưa quá nhiều view controller vào một story board sẽ làm rối, load chậm.

Giải pháp

- Chia nhỏ thành nhiều các storyboard.



Demo Story Board



/Swift

```
var myVariable = 42
```

```
myVariable = 50
```

```
let myConstant = 42
```

/Swift

```
print("Hello, world!")
```

```
// Hello, world!
```

```
var myVariable = 50
```

```
print("Hello \(myVariable)")
```

```
// Hello 50
```

/ Operators

- +
- -
- *
- /
- ++
- --

/ Conditionals and Loops

- if / else if/ else
- for
- while
- break
- continue
- do-while

/ Functions

```
func functionName(param1: type, param2: type, ...) -> return_type
{
    //code of function
}
```

Example

```
func showMeInfo(name: String)
{
    print("My name is \(name) ")
}
```

Example

```
func showMeInfo() -> String
{
    return "MyName"
}
```

/Swift - Array

```
var phones= ["samsung", "htc", "nokia"]  
phones[1]  
//htc  
phones.append("apple")  
// ["samsung", "htc", "nokia", "apple"]  
phones.remove(at: 1)  
// ["samsung", "nokia", "apple"]
```

/Swift - dictionary

```
var dict = ["samsung":"Galaxy S9", "Apple":"iPhone X"]

print(dict["samsung"]!) //Galaxy S9

dict.removeValue(forKey: "samsung")

// ["Apple":"iPhone X"]
```

/Swift - closure

Closure expression syntax has the following general form:

```
{ (parameters) -> return type in  
    //statements  
}
```

/Swift - closure

```
func someFunctionThatTakesAClosure(closure: () -> Void) {  
    // function body goes here  
}
```

// Here's how you call this function without using a trailing closure:

```
someFunctionThatTakesAClosure(closure: {  
    // closure's body goes here  
})
```

// Here's how you call this function with a trailing closure instead:

```
someFunctionThatTakesAClosure() {  
    // trailing closure's body goes here  
}
```

/Swift - protocol

```
protocol SomeProtocol {  
    // protocol definition goes here  
}  
  
struct SomeStructure: FirstProtocol, AnotherProtocol {  
    // structure definition goes here  
}  
  
class SomeClass: SomeSuperclass, FirstProtocol,  
AnotherProtocol {  
    // class definition goes here  
}
```

/Swift - extension

```
extension SomeType: SomeProtocol, AnotherProtocol {  
    // implementation of protocol requirements goes  
    here  
}
```

/Swift – Optional (? !)

- Optional là một tính năng rất mạnh mẽ của Swift để giúp chương trình trở nên an toàn và ít bị crash hơn.
- Kí hiệu của Optional là `Optional<T>` với `T` là kiểu dữ liệu, để ngắn gọn hơn người ta dùng kiểu viết tắt là `T?`

```
var x: Int? // Biến x có kiểu là Optional<Int> - Được phép nil  
var y: Int // Biến y có kiểu là Int, không phải là Optional
```

- Optional cho phép chúng ta gán biến bằng nil: `x = nil`
- Đối với trường hợp không phải Optional, chúng ta không thể gán biến bằng nil: `y = nil` (IDE báo lỗi)
- Biến x có giá trị khởi tạo là nil

/Swift – Optional (? !)

Force Unwrapping (!)

```
var x: Int? = 0  
x = x + 1 //Error
```

Force unwrapping là việc chúng ta cam đoan với trình biên dịch rằng biến optional CHẮC CHẮN CÓ GIÁ TRỊ

x! + 1 // **Force Unwrapping**

/Swift – Optional (? !)

Optional Binding

```
var a: String? = "OK"
```

C1:

```
if let aUnwrap = a {  
    print(aUnwrap)  
}
```

C2:

```
guard let aUnwrap = a else { return }  
print(aUnwrap)
```

C3:

```
print(a ?? "") //unwrap với default value
```

/Swift – Optional (? !)

Implicitly Unwrapped Optional

ImplicitlyUnwrappedOptional<T>

```
var user: User?  
⇒ user?.name
```

```
var user: User!  
⇒ user.name
```

/Swift – class

```
class User {  
  
    var name: String?  
    var age: Int = 0  
  
    init() {  
  
    }  
  
    init(name: String?, age: Int) {  
        self.name = name  
        self.age = age  
    }  
  
    convenience init(name: String?) {  
        self.init(name: name, age: 30)  
    }  
}
```

/Swift – struct

```
struct Product {  
    var name: String?  
    var phone: String?  
    var price: Int = 0  
  
    mutating func changeName(name: String) {  
        self.name = name  
    }  
  
    var p1 = Product(name: "Product 1")  
    var p2 = Product(name: "Product 2", price: 30)  
    var p3 = Product(name: "Product 2", phone: "099999999", price: 30)
```

/Swift - class vs struct

Class: User

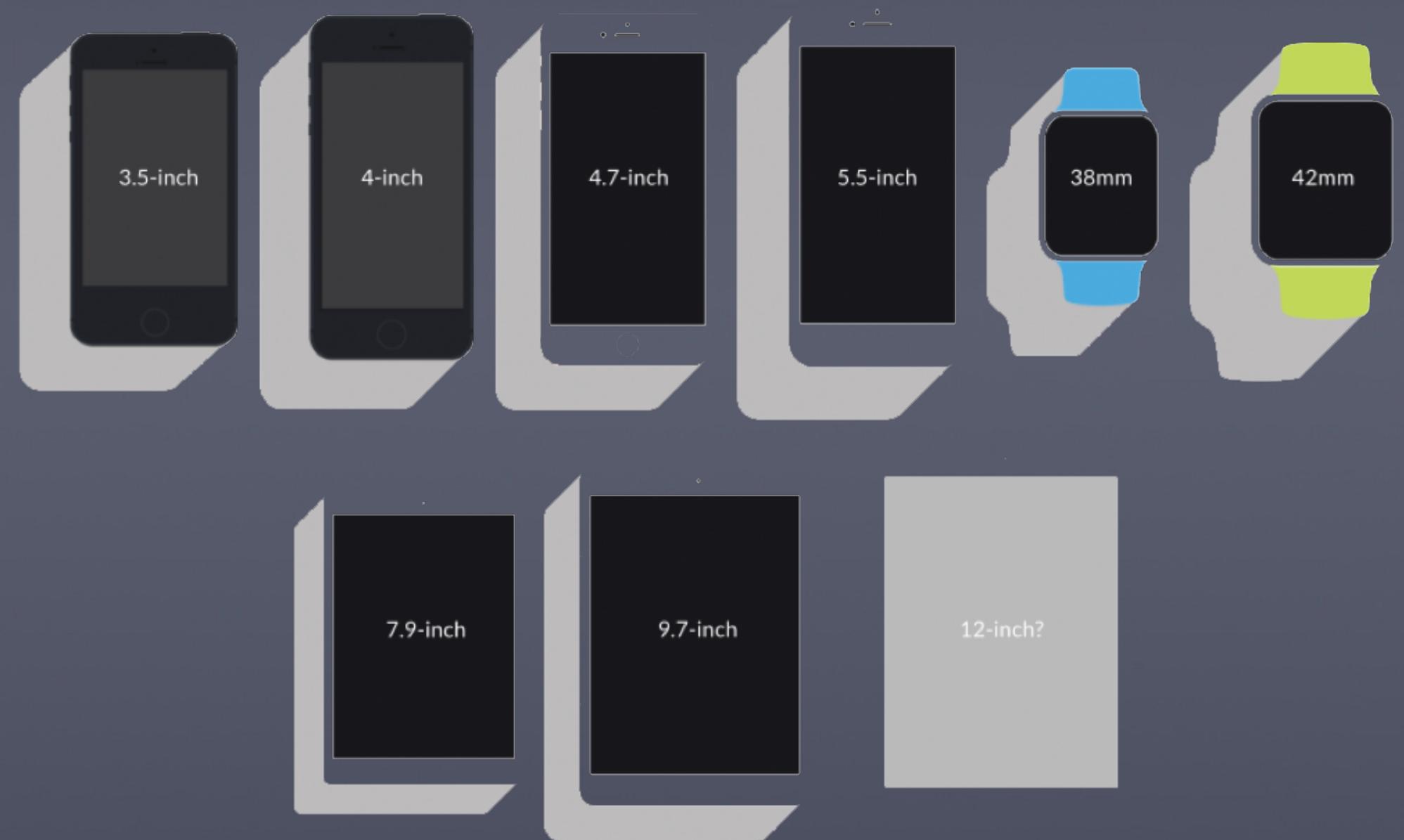
```
var user1 = User(name: "Anhth")
var user2 = user1
user1.name = "Anh Tran"
print(user2.name!) // Anh Tran
```

Struct: User

```
var user1 = User(name: "Anhth")
var user2 = user1
user1.name = "Anh Tran"
print(user2.name!) // Anhth
```

/ Auto layout

- Xuất hiện cùng với Xcode 5
- Auto Layout là một hệ thống cho phép bạn bố trí giao diện của ứng dụng bằng cách tạo ra các mối quan hệ giữa các yếu tố.
- Người dùng xác định các ràng buộc giữa các view với nhau để bố trí hợp lý, tự động thay đổi linh hoạt kích thước của chúng trên nhiều loại màn hình.
- Thao khảo thêm:
<https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/AutolayoutPG/Introduction/Introduction.html>



Log In

[Use Google Account](#)

EMAIL ADDRESS

PASSWORD

[Forgot your password?](#)

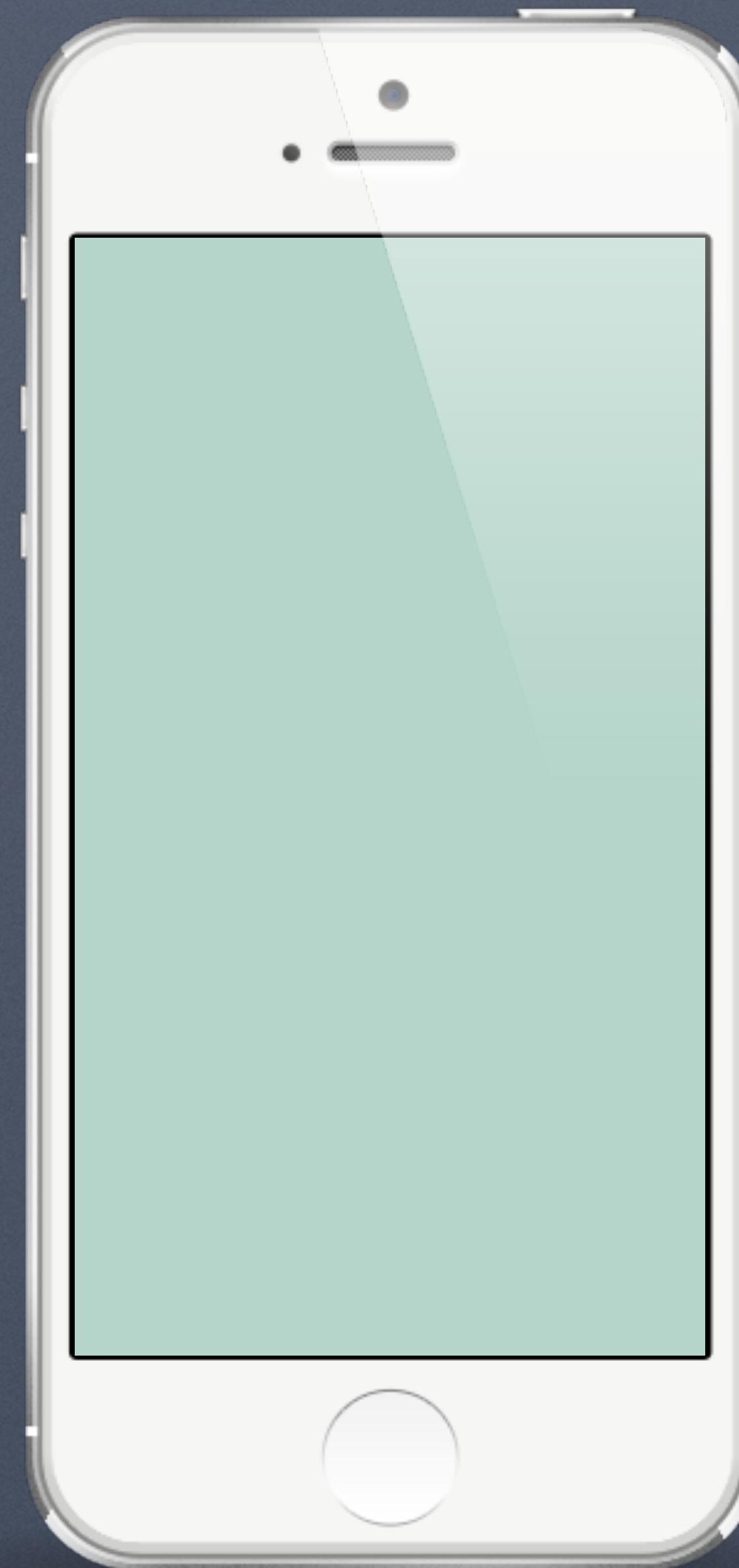
[Log In](#)

Byrdie
10 phut truoc
Pokemon Go thu ve 2 ty do



Doanh thu pokemon go
dantri.com.vn

LIKE COMMENT SHARE

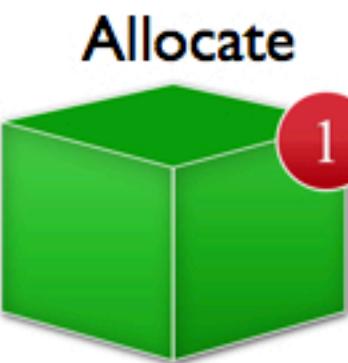


Demo

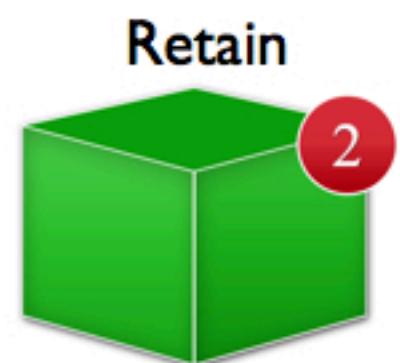


/ Automatic Reference Counting (ARC)

Manual Reference Counting



```
MyClass *obj1 = [[MyClass alloc] init];
```



```
MyClass *obj2 = [obj1 retain];
```



```
[obj2 release];
```



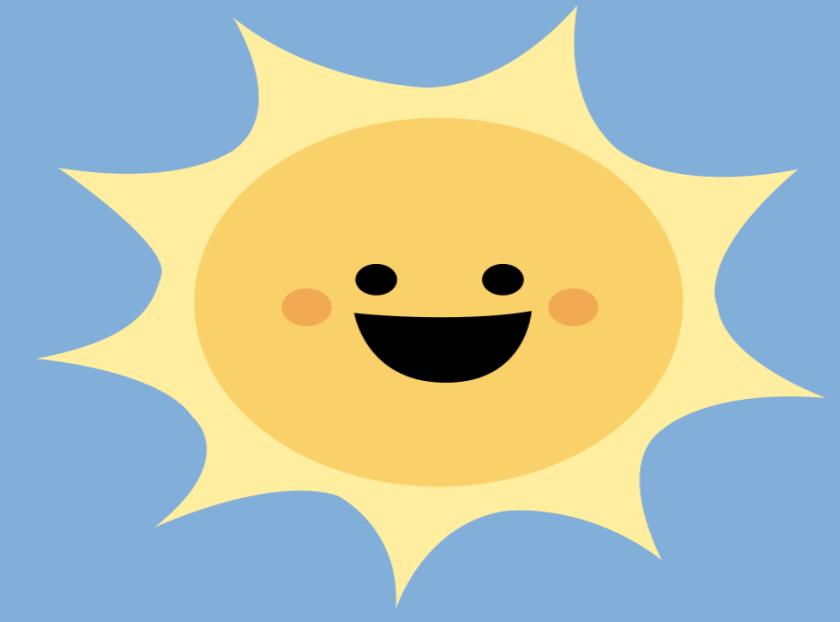
```
[obj1 release];
```

/ Strong, Weak, Unowned

	var	let	optional	non-optional
Strong	✓	✓	✓	✓
Weak	✓	✗	✓	✗
Unowned	✓	✓	✗	✓



~
Ó



THANKS!