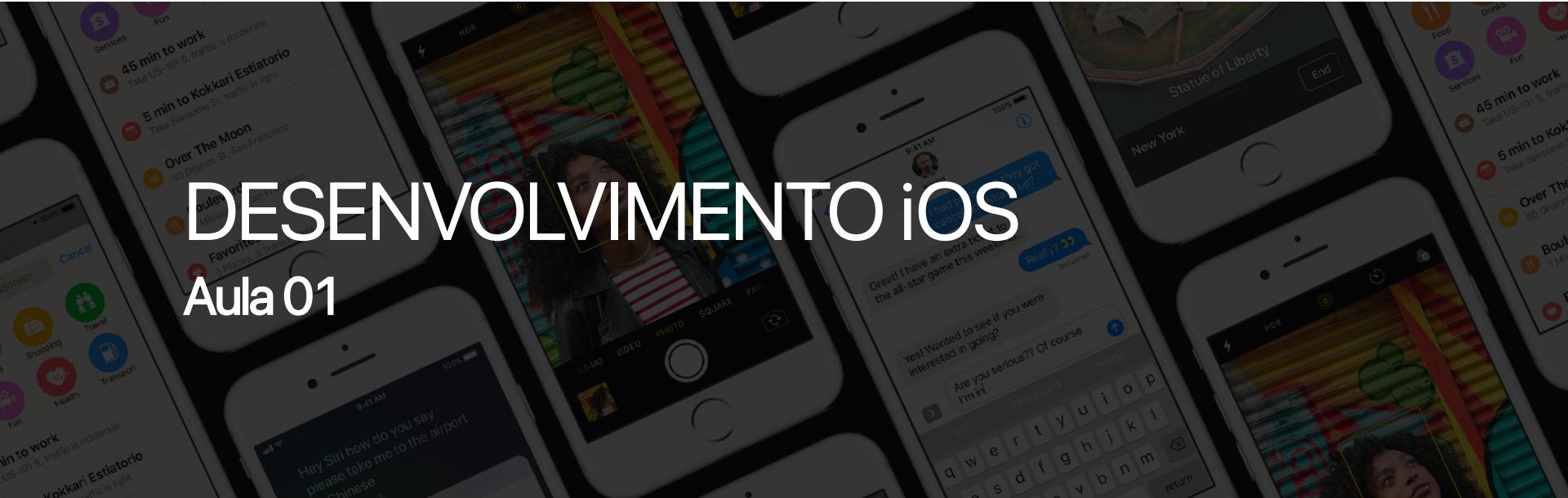


DESENVOLVIMENTO iOS

Aula 01



1. A linguagem Swift e suas particularidades

A Linguagem Swift e suas particularidades

1. Comentários

- Comentários de linha simples:

```
//var str = "Hello, Swift!"
```

- Comentários de múltiplas linhas:

Em Swift podemos ter comentários de múltiplas linhas dentro de outro comentário de múltipla linhas

```
/*
var result: Int = 0

/*
while result < 100 {
    result += 1
}
*/

for i in 1..<100 {
    result += 1
}
*/
```

A Linguagem Swift e suas particularidades

2. Variáveis e constantes

- Utilizamos a palavra reservada `var` para definirmos variáveis
- Constantes são criadas usando `let`
- Podemos definir o tipo de uma variáveis explicitamente ou por inferência
- É possível declarar múltiplas variáveis em uma só linha
- Não usamos ; no fim de um código

```
var accountValue = 250.50      //Variável
let name = "João Felipe"       //Constante
var age: Int = 27              //Variável com tipo explícito
var x = 22, y = 12, z = -8     //Múltiplas declarações
```

A Linguagem Swift e suas particularidades

3. Tipos

- **Int, UInt, Double, Float, String, Character, Bool**
- Variações de Int (signed): **Int8, Int16, Int32, Int64**
- Versões unsigned de Int: **UInt8, UInt16, UInt32, UInt64**
- **Int** pode ser 64bit ou 32bit, dependendo do S.O.
- Números **sem** ponto flutuante é inferido o tipo **Int**
- Números **com** ponto flutuante, o tipo inferido é **Double**
- Usando conteúdo entre aspas, a inferência é **String**

A Linguagem Swift e suas particularidades

3. Tipos

```
let dolar: Float = 3.25
var lastName = "Souza"          //String por inferência
var month = 7                  //Int por inferência
var myAge: UInt8 = 25
var gender: Character = "M"
var latitude: Double = -23.5641226
var longitude = -46.6546489   //Double por inferência

//Concatenando Strings
var firstName = "Fernando"
var lastName = "Oliveira"
var fullName = firstName + " " + lastName

//Interpolação de Strings
var name = "\(firstName) \\(lastName)"
```

A Linguagem Swift e suas particularidades

3. Tipos

Tipo composto: Tupla

- Tipo que pode conter outros tipos, inclusive outras tuplas
- Seus elementos podem ser nomeados

```
var address: (String, Int, String) = ("Av. Paulista", 1000, "01311-000")

//Tupla com elementos nomeados
var student: (name: String, age: Int, grade: Double) = ("Paulo Filho", 33, 10.0)

//Acessando uma tupla pela localização do elemento
print("O CEP da FIAP Paulista é \(address.2)")

//Acessando uma tupla pelo nome do elemento
print("A nota do \(student.name) foi \(student.grade)")

//Decompondo uma tupla em variáveis
let (street, number, zipCode) = address
print("FIAP: \(street) número \(number), CEP: \(zipCode)")

//Se não desejarmos algum dos elementos, usamos _
let (name, _, grade) = student
```

A Linguagem Swift e suas particularidades

4. Optional

- Servem para tratar a ausência de valor em uma variável
- Utilizamos **?** Para definir um tipo como Optional
- Recuperamos/Desembrulhamos (unwrapping) o valor do optional usando **!**
- Podemos definir um tipo como Optional já desembrulado usando **!** Na definição do tipo. Dessa forma, estamos criando um **Implicitly Unwrapped Optional**

Optional Chaining

- Técnica segura para acessar um Optional
- Utilizamos **?** após o objeto no qual desejamos acessar uma propriedade ou chamar um método

A Linguagem Swift e suas particularidades

4. Optional

```
//Optional de String com valor nulo
var address: String?

//Atribuindo valor a um Optional
address = "Meu endereço"

//Recuperando valor de um Optional (forma não é segura)
var totalBooks: Int? = 14
var newTotal = totalBooks! + 1 //Usamos ! ao lado da variável

//Optional Binding: Testando o valor de um Optional
//e atribuindo a uma variável de maneira segura
if let totalBooksValue = totalBooks {
    print("Eu tenho \(totalBooksValue)")
} else {
    print("totalBook não contém valor")
}

//Optional com abertura implícita (Implicitly Unwrapped Optional)
var euro: Double! = 3.95
var ticket: Double = 800.0
let ticketInReal = ticket * euro
```

A Linguagem Swift e suas particularidades

5. Operadores

- **Atribuição:** `=`
- **Aritméticos:** `+`, `-`, `*`, `/`, `%` (módulo)
- **Compostos:** `+ =`, `- =`, `* =`, `/ =`, `% =`
- **Lógicos:** `&&`, `||`, `!` (negação)
- **Comparação:** `>`, `<`, `>=`, `<=`, `==`, `!=`
- **Identidade** (usado em classes): `====`, `!====`
- **Ternário**: (booleano) `?` “resultado true” `:` “resultado false”
- **Coalescência nula**: `valor1 ?? Valor2`
- **Closed Range**: `1...100`
- **Half-Open Range**: `1..100`

A Linguagem Swift e suas particularidades

6. Operadores personalizados

```
//Prefix: Operador unário usado antes do operando
prefix operator >-
prefix func >- (right: Int) -> Int {
    return right * right
}
>-4      //Resultado: 16

//Postfix: Operador unário usado depois do operando
postfix operator -<
postfix func -< (right: Int) -> Int {
    return right / 2
}
4-<      //Resultado: 2

//Infix: Operador binário
infix operator >-<
func >-< (left: Int, right: Int) -> Int {
    return 2 * left * right
}
4>-<5    //Resultado: 40
```

A Linguagem Swift e suas particularidades

7. Estruturas condicionais e controle de fluxo

Estruturas de condição

- If - else - else if

```
let grade = 7.0
if grade >= 7.0 {
    print("Aprovado")
} else {
    print("Reprovado")
}

var temperature = 19, climate = ""
if temperature <= 0 {
    climate = "Muito frio"
} else if temperature < 14 {
    climate = "Frio"
} else if temperature < 21 {
    climate = "Agradável"
} else if temperature < 30 {
    climate = "Quente"
} else {
    climate = "Muito quente"
}
```

7. Estruturas condicionais e controle de fluxo

Estruturas de condição

- **switch:** Em Swift, o switch precisa ser exaurido (ter todos os cenários possíveis cobertos)

```
var letter = "O", letterType = ""
switch letter.lowercased() {
    case "a", "e", "i", "o", "u":
        letterType = "vogal"
    default:
        letterType = "consoante"
}

let speed: Double = 95.0
switch speed {
    case 0.0..<20.0:
        print("Primeira marcha")
    case 20.0..<40.0:
        print("Segunda marcha")
    case 40.0..<50.0:
        print("Terceira marcha")
    case 50.0..<90.0:
        print("Quarta marcha")
    default:
        print("Quinta marcha")
}
```

A Linguagem Swift e suas particularidades

7. Estruturas condicionais e controle de fluxo

Controle de fluxo

- while / repeat while

```
//While
var count = 1
while count <= 15 {
    print("count is \(count)")
    count += 1
}

//Repeat-While
var maxNumbers = 15, numbers: [Int] = [], number: Int = 0
repeat {
    number = Int(arc4random_uniform(25) + 1)
    if numbers.index(of: number) == nil {
        numbers.append(number)
    }
} while numbers.count < maxNumbers
numbers.sort()
```

A Linguagem Swift e suas particularidades

7. Estruturas condicionais e controle de fluxo

Controle de fluxo

- `for in`

```
let names = ["Paulo", "Maria", "João", "Carla", "Felipe"]

//Arrays
for name in names { print("Nome:", name) }

//Usando range
for index in 1...5 {
    print("\(index) multiplicado por 5 é \(index * 5)")
}

//Dicionários
let people = ["Paulo": 25, "Renata": 18, "Pedro": 34, "Carol": 35]
for (name, age) in people {
    print("\(name) tem \(age) anos")
}

//Usando Stride: Definimos o início, o fim e o incremento
for i in stride(from: 3, to: 100, by: 3) { print(i) }
//3, 6, 9, 12, 15, 18, 21, 24, 27, 30, ...
```

A Linguagem Swift e suas particularidades

8. Coleções

Arrays

- Lista ordenada de valores do mesmo tipo

```
//Inicialização
var emptyArray: [String] = []
var emptyArray2 = [String]()
var shoppingList: [String] = ["Açúcar", "Leite", "Café"]
var inferredShoppingList = ["Açúcar", "Leite", "Café"] //Tipo inferido

//Arrays vazios
var emptyIntArray: [Int] = []

//Testa se o array está vazio
if shoppingList.isEmpty {
    print("A lista de compras está vazia.")
} else {
    print("A lista de compras NÃO está vazia.")
}

//Recuperando elementos
var firstItem = shoppingList[0]
print("A lista tem \(shoppingList.count) items.") //A lista tem 3 items.
```

A Linguagem Swift e suas particularidades

8. Coleções

```
//Adição/Alteração de elementos
shoppingList.append("Sabão")
shoppingList += ["Queijo"]
shoppingList += ["Manteiga", "Pão"]
//["Açúcar", "Leite", "Café", "Sabão", "Queijo", "Manteiga", "Pão"]

//Usando índices e ranges
shoppingList[1] = "Leite em pó"
shoppingList[4...6] = ["Maçã", "Pera"]
shoppingList.insert("Banana", at: 3)
//["Açúcar", "Leite em pó", "Café", "Banana", "Sabão", "Maçã", "Pera"]

//Remoção de elementos
let sugar = shoppingList.remove(at: 0) //Remove o primeiro
let milk = shoppingList.removeFirst() //Remove o primeiro
let cheese = shoppingList.removeLast() //Remove o último
shoppingList.removeLast(2) //Remove os 2 últimos

//Pesquisando elementos em um Array
if let bananaIndex = shoppingList.index(of: "Banana") {
    print("Banana se encontra no índice \(bananaIndex)")
} else {
    print("Não tem banana na lista de compras")
}

if shoppingList.contains("Arroz") { //Outra forma de pesquisar. Retorna Bool
    print("Compraram arroz!")
}
```

A Linguagem Swift e suas particularidades

8. Coleções

Dicionários

- Lista não-ordenada de valores do mesmo tipo, associados a chaves únicas

```
//Inicialização
var states: [String: String] = ["BA": "Bahia", "SP": "São Paulo",
                                 "RJ": "Rio de Janeiro", "MG" : "Minas Gerais"]

var emptyStates: [String: String] = [:] //Iniciando um dicionário vazio

//verifica se um dicionário está vazio
if states.isEmpty {
    print("Dicionário de estados está vazio.")
}

//Verificando a presença de um elemento em um dicionário
if let stateName = states["RJ"] {
    print("O nome do estado é \(stateName).")
}
if states["AC"] != nil {
    print("O nome do estado é \(states["AC"]!).")
}

//Total de elementos de itens em um dicionário
print("O dicionario contem \(states.count) items.")
```

A Linguagem Swift e suas particularidades

8. Coleções

```
//Inserindo/atualizando um elementos
states["MG"] = "Minas Gerais"
states.updateValue("S. Paulo", forKey: "SP")
//["SP": "S. Paulo", "BA": "Bahia", "RJ": "Rio de Janeiro", "MG": "Minas Gerais"]

//Remoção de item
states["RJ"] = nil
states.removeValue(forKey: "SP")

//Iteração
for state in states {
    print("\(state.key): \(state.value)")
}

//Iteração por decomposição
for (uf, name) in states {
    print("\(uf): \(name)")
}

//Iteração somente nas chaves
for uf in states.keys {
    print("Sigla do estado: \(uf)")
}

//Iteração somente nos valores
for name in states.values {
    print("Nome do estado: \(name)")
}
```

A Linguagem Swift e suas particularidades

9. Enumeradores

- Define um tipo comum para um grupo de valores, permitindo que se trabalhe com tais valores de uma maneira segura

```
enum Compass {  
    case north  
    case south  
    case east  
    case west  
}  
var heading:Compass = Compass.north  
heading = .south  
switch heading {  
    case .north:  
        print("Indo para o Norte")  
    case .south:  
        print("Indo para o Sul")  
    case .east:  
        print("Indo para o leste")  
    case .west:  
        print("Indo para o oeste")  
}  
  
//Única linha  
enum Month {  
    case january, february, march, april, may, june, july, august,  
        september, october, november, december  
}
```

A Linguagem Swift e suas particularidades

9. Enumeradores

```
//Enums com valores padrões
//Caso o valor não seja definido, é associado o valor do nome
enum WeekDay: String {
    case sunday = "domingo"
    case monday = "segunda-feira"
    case tuesday = "terça-feira"
    case wednesday = "quarta-feira"
    case thursday = "quinta-feira"
    case friday = "sexta-feira"
    case saturday = "sSábado"
}

print(WeekDay.sunday.rawValue) //domingo

enum MonthIndex: Int {
    case january = 1, february, march, april, may, june, july,
        august, september, october, november, december
}

//Inicializando
if let sunday = WeekDay(rawValue: "domingo") {
    print(sunday.hashValue, sunday.rawValue) //0 domingo
}
```

A Linguagem Swift e suas particularidades

9. Enumeradores

```
//Valores Associados (Associated Values)
//Podemos associar valores, de diferentes tipos, a enumeradores

enum Measure {
    case weight(Double)
    case age(Int)
    case size(width: Double, height: Double)
}

let measure: Measure = .size(width: 0.6, height: 1.71) // .weight(82) // .age(39)
switch measure {
    case .weight(let weight):
        print("O seu peso é \(weight)")
    case .age(let age):
        print("A sua idade é \(age)")
    case .size(let size):
        print("Você tem \(size.width)m de largura")
        print("e \(size.height)m de altura")
/*
//Também poderíamos associar os valores a variáveis de uma vez
case let .size(width, height):
    print("Você tem \(width)m de largura e \(height)m de altura")
*/
}
```

10. Estruturas (Structs)

- Definem propriedades e métodos que armazenam valores e executam operações
- Já possuem inicializadores pré-definidos que servem para alimentar suas variáveis
- Podem implementar protocolos que fornecem padrões de funcionalidade
- São passadas por valor, não por referência

A Linguagem Swift e suas particularidades

10. Estruturas (Structs)

```
struct Driver {  
    var name: String  
    var registration: String  
    var age: Int  
  
    //Usamos mutating para que um método interno possa modificar  
    //o valor das propriedades de uma struct  
    mutating func changeAge(newAge: Int) {  
        age = newAge  
    }  
}  
  
//Inicializador pré-definido  
let paula = Driver(name: "Paula Lima", registration: "23409-4", age: 22)  
  
//Os valores são passados por cópia  
var leonardo = paula  
leonardo.name = "Leonardo Ferreira"  
leonardo.registration = "34299-0"  
leonardo.changeAge(newAge: 27)  
  
print(paula.name, paula.age)    //Paula Lima 22
```

A Linguagem Swift e suas particularidades

11. Funções

- Permite organizar trechos de código fonte com diferentes finalidades
- Cada função possui um tipo, que consiste dos tipos dos parâmetros da função e seu tipo de retorno
- Podem aceitar quaisquer tipos de parâmetros e de retorno, inclusive outras funções ou tuplas por exemplo
- Em Swift, funções são “**first-class citizens**” (objetos de primeira classe), ou seja, elas podem ser atribuídas a uma variável e também passadas/retornadas como parâmetro de outras funções.

A Linguagem Swift e suas particularidades

11. Funções

Algumas funções do framework Foundation

- Matemáticas

```
//pow: Potência  
pow(2, 3)      //8  
pow(64, 0.5)   //8
```

```
//sqrt: Raiz quadrada  
sqrt(64)       //8
```

```
//abs: Valor absoluto (sem sinal)  
abs(-3)        //3
```

```
//round, floor, ceil  
let dolar = 3.20  
round(dolar)    //Arredondamento  
floor(dolar)    //Pra baixo  
ceil(dolar)     //Pra cima
```

```
//min, max: Menor valor, maior valor  
min(5, 9, 12, 0, 22, 14)  //0  
max(5, 9, 12, 0, 22, 14) //22
```

A Linguagem Swift e suas particularidades

11. Funções

- Coleções / Strings

```
var teams = ["Palmeiras", "São Paulo", "Corinthians", "Santos", "Flamengo", "Fluminense",
    "Remo", "Chapecoense"]

//enumerated: decompõe o array em tuplas de chaves / valor
for (index, team) in teams.enumerated() {
    print(index, team)      //index é a chave e team é o valor
}

//min, max: menor valor, maior valor
teams.min()      //Chapecoense
teams.max()      //São Paulo

//joined: retorna uma string com os valores do array separados por um separador
let teamsString = teams.joined(separator: ";")

//hasPrefix, hasSuffix: verifica se a string inicia ou termina com o valor passado
var name = "Maria Fernanda Lima"
name.hasPrefix("A")      //false
name.hasSuffix("o")      //true
```

A Linguagem Swift e suas particularidades

11. Funções

```
//lowercased e uppercased: converte para minúsculas e maiúsculas
name.lowercased()           //maria fernanda lima
name.uppercased()           //MARIA FERNANDA LIMA

//map: modifica cada um dos elementos de um array
let teamsUppercased = teams.map({$0.uppercased()})
teamsUppercased
//[ "PALMEIRAS", "SÃO PAULO", ....]

//reduce: agrupa os valores de um array
let ages = [29, 32, 38, 19, 34, 16, 36]
let agesCount = ages.reduce(0, {$0 + $1})
print(agesCount)           //204

//components: separa os elementos de uma string em arrays
teamsString.components(separatedBy: ";")
//[ "Palmeiras", "São Paulo", ...]

//replacingOccurrences: substitui as ocorrências de um valor por outro em uma String
teamsString.replacingOccurrences(of: ";", with: "-")
```

A Linguagem Swift e suas particularidades

11. Funções

Personalidas

```
//Funções sem parâmetro e retorno
func doNothing() {}
doNothing()

//Funções com retorno
func getNumberMonths() -> Int {
    return 12
}
let x1 = getNumberMonths()      //x1 recebe 12

//Funções com parâmetro
func doubles(a: Int) -> Int {
    return 2 * a
}
let x2 = doubles(a: 10)        //x2 recebe 20

//Funções com mais de um parâmetro
func sum(a:Int, b:Int) -> Int {
    return a + b
}
sum(a: 1, b: 5)               //6

//Funções com mais de um retorno
let student = "Jaqueline Pereira;38"
func getNameAndAge(data: String) -> (name: String, age: Int) {
    let student = data.components(separatedBy: ";")
    guard let name = student.first, let ageStr = student.last, let age = Int(ageStr) else {
        return ("", 0)
    }
    return (name, age)           //(name: "Jaqueline Pereira", age: 38)
}
```

A Linguagem Swift e suas particularidades

11. Funções

```
//Nomes externos e internos de parâmetro
func say(greeting sentence: String, to name: String) {
    print("\(sentence) \(name)!")
}

say(greeting: "Hello", to: "Fernando")      //Hello Fernando!

//Funções sem parâmetro exteno e com valores padrão
func power(_ a: Double, _ b: Double = 2) -> Double {
    return pow(a, b)
}

power(3, 3)      //27

//Criando funções que aceitam N parâmetros
func sumValues(initialValue: Int, withValues values: Int...) -> Int {
    var result = initialValue
    for value in values {
        result += value
    }
    return result
}

sumValues(initialValue: 10, withValues: 5, 8, 9, 7, 6, 5, 1, 2)      //53
```

A Linguagem Swift e suas particularidades

11. Funções

```
//Aceitando funções como parâmetro
func multiply(a:Int, b:Int) -> Int {
    return a * b
}
func divide(a:Int, b:Int) -> Int {
    return a / b
}
func subtract(a:Int, b:Int) -> Int {
    return a - b
}
func calculate(a: Int, b: Int, operation: (Int, Int) -> Int) -> Int {
    return operation(a, b)
}
calculate(a: 12, b: 6, operation: divide) //2

//Devolvendo funções
typealias Operation = (Int, Int) -> Int      //Criando um "apelido" para um tipo
func getOperation(named operation: String) -> Operation {
    switch operation.lowercased() {
        case "adição":
            return sum
        case "subtração":
            return subtract
        case "multiplicação":
            return multiply
        default:
            return divide
    }
}
let op = getOperation(named: "subtração")
op(7, 2) //5
```

A Linguagem Swift e suas particularidades

12. Closures

- São blocos auto-contidos de funcionalidades que podem ser utilizados como parâmetro em métodos
- Semelhantes aos blocos em C / Objective-C e aos lambdas em outras linguagens de programação
- Em essência, são funções que não possuem nome
- Podem ser utilizadas com uma sintaxe mais curta e rápida

```
//Sintaxe de uma closure

/*
 {(parametro: Tipo) -> TipoDoRetorno in
    codigo 1
    codigo 2
    codigo 3
    return TipoDoRetorno
}
*/
```

A Linguagem Swift e suas particularidades

12. Closures

```
func calculate(a: Int, b: Int, operation: (Int, Int) -> Int) -> Int {  
    return operation(a, b)  
}  
  
//Forma completa  
calculate(a: 12, b: 6, operation: {(num1: Int, num2: Int) -> Int in  
    return num1*num2 + num2  
})  
  
//Simplificando a closure  
calculate(a: 12, b: 6, operation: {(num1, num2) -> Int in  
    return num1*num2 + num2  
})  
  
calculate(a: 12, b: 6, operation: {num1, num2 -> Int in  
    return num1*num2 + num2  
})  
  
calculate(a: 12, b: 6, operation: {num1, num2 in  
    return num1*num2 + num2  
})  
  
calculate(a: 12, b: 6, operation: {  
    return $0*$1 + $1  
})  
  
calculate(a: 12, b: 6, operation: {$0*$1 + $1})  
calculate(a: 12, b: 6, operation: +)
```

A Linguagem Swift e suas particularidades

12. Closures

```
func calculate(a: Int, b: Int, operation: (Int, Int) -> Int) -> Int {  
    return operation(a, b)  
}  
  
//Forma completa  
calculate(a: 12, b: 6, operation: {(num1: Int, num2: Int) -> Int in  
    return num1*num2 + num2  
})  
  
//Simplificando a closure  
calculate(a: 12, b: 6, operation: {(num1, num2) -> Int in  
    return num1*num2 + num2  
})  
  
calculate(a: 12, b: 6, operation: {num1, num2 -> Int in  
    return num1*num2 + num2  
})  
  
calculate(a: 12, b: 6, operation: {num1, num2 in  
    return num1*num2 + num2  
})  
  
calculate(a: 12, b: 6, operation: {  
    return $0*$1 + $1  
})  
  
calculate(a: 12, b: 6, operation: {$0*$1 + $1})  
calculate(a: 12, b: 6, operation: +)
```

13. Tratamento de Erros

- Definimos que o método dispara erros com o uso do `throws`
- Os erros devem ser representados por valores de tipos que implementam o protocolo `Error`
- Utiliza a estrutura `do / try / catch`
- Temos acesso ao erro capturado pela propriedade `error`

A Linguagem Swift e suas particularidades

13. Tratamento de Erros

```
enum AccessError: Error {
    case wrongPassword, wrongLogin, wrongData
}

func login(userName: String, password: String) throws -> String {
    let validUser = "alunofiap"
    let validPassword = "abc@123"
    if userName != validUser && password != validPassword {
        throw AccessError.wrongData
    } else if userName != validUser {
        throw AccessError.wrongLogin
    } else if password != validPassword {
        throw AccessError.wrongPassword
    } else {
        return "Usuário logado com sucesso"
    }
}

do {
    let result = try login(userName: "nome-aluno", password: "senha321")
    print(result)
} catch {
    switch error as! AccessError {
        case .wrongData:
            print("Você errou sua senha")
        case .wrongLogin:
            print("Você errou o seu usuário")
        case .wrongPassword:
            print("Você errou o seu usuário e a sua senha!!")
    }
}
```

14. Generics

- Permite que criemos funções flexíveis e reutilizáveis que aceitam vários tipos e sujeitas a certos requerimentos
- Também podemos criar tipos genéricos

A Linguagem Swift e suas particularidades

14. Generics

```
//Função que aceita apenas inteiros
func swapInts(_ a: inout Int, _ b: inout Int) {

    //Obs.: inout faz com que os valores sejam passados por referência
    let temporaryA = a
    a = b
    b = temporaryA
}

var a = 23
var b = 7
swapInts(&a, &b)      //a é 7 e b é 23

//Função que aceita qualquer tipo
func swapValues<T>(_ a: inout T, _ b: inout T) {
    let temporaryA = a
    a = b
    b = temporaryA
}

var x = 25.9
var y = 12.8

swapValues(&x, &y)  //Aceita qualquer tipo (Int, Double, Float, etc..)
```

15. Classes e Objetos

Classes

- Definem propriedades e métodos que armazenam valores e executam operações
- Devem conter inicializadores que alimentem suas propriedades
- Podem herdar características de outras classes e implementar protocolos que fornecem padrões de funcionalidade
- São passadas por referência (**reference type**)
- Podem ter suas funcionalidades ampliadas (**extensão**)

A Linguagem Swift e suas particularidades

15. Classes e Objetos

```
class Vehicle {  
    var _speed: UInt = 0  
    var model: String  
  
    //variáveis computadas  
    var maxSpeed: UInt {  
        return 250  
    }  
    var speed: UInt {  
        get { return _speed }  
        set {  
            let finalValue = min(newValue, maxSpeed)  
            _speed = finalValue  
        }  
    }  
  
    //propriedade de classe (estática)  
    static let speedUnit = "Km/h"  
  
    //inicializador (construtor)  
    init(model: String) {  
        self.model = model  
    }  
  
    //métodos de classe (estáticos)  
    class func alert() -> String { return "Se beber não dirija" }  
  
    //métodos de instância  
    func descript() -> String {  
        return "Veículo: \(model), velocidade atual: \(speed)"  
    }  
}
```

A Linguagem Swift e suas particularidades

15. Classes e Objetos

Inicializadores convenience e designated

- **Designated initializers** são inicializadores responsáveis pela construção de uma nova instância da classe e definição dos valores de suas propriedades
- **Convenience initializers** são inicializadores secundários, construídos para instanciar uma classe em cenários específicos

Regras de uso

- Um designated deve chamar um designated da sua classe mãe imediata
- Um convenience deve chamar outro inicializador de sua própria classe
- O convenience deve, no final da hierarquia de chamadas, chamar um designated

A Linguagem Swift e suas particularidades

15. Classes e Objetos

```
let bike = Vehicle(model: "Honda Civic")      //Instanciando

class Car: Vehicle {    //Herança
    var licensePlate: String
    var driver: String?
    override var maxSpeed: UInt { return 100 } //Sobrescrita
    subscript(index: Int) -> String {        //Subscripts
        get {
            let plateArray = Array(self.licensePlate)
            return String(plateArray[index])
        }
        set {
            var plateArray = Array(self.licensePlate)
            plateArray[index] = Character(newValue)
            self.licensePlate = String(plateArray)
        }
    }
    override func descript() -> String {
        return "\(super.descript()), Placa: \(licensePlate)"
    }
    init(model: String, licensePlate: String) { //Designated Initializer
        self.licensePlate = licensePlate
        super.init(model: model)
    }
    convenience init(driver: String) {          //Convenience Initializer
        self.init(model: "", licensePlate: "")
        self.driver = driver
    }
}
```

A Linguagem Swift e suas particularidades

16. Extensões e protocolos

Extensões

- Permitem que novas funcionalidades sejam implementadas em estruturas classes existentes

```
let carlos = "Carlos Alberto Marques Rezede"
extension String {
    var initials: String {
        return self.components(separatedBy: " ").map({String($0.first!)})joined()
    }
}

print(carlos.initials)          //CAMR
"Fernando Inácio Almeda Peixoto".initials //FIAP
```

A Linguagem Swift e suas particularidades

16. Extensões e protocolos

```
extension Car {  
  
    var rotation: String {  
        switch String(self.licensePlate.last!) {  
            case "1", "2":  
                return "segunda-feira"  
            case "3", "4":  
                return "terça-feira"  
            case "5", "6":  
                return "quarta-feira"  
            case "7", "8":  
                return "quinta-feira"  
            case "9", "0":  
                return "sexta-feira"  
            default:  
                return "dados inválidos"  
        }  
    }  
  
    func showLicenseAndRotation() {  
        print("este carro tem a placa \(licensePlate) e o seu rodízio é dia de \(rotation)")  
    }  
  
}  
  
print(car.rotation)  
car.showLicenseAndRotation()
```

16. Extensões e protocolos

Protocolos

- Definem uma regra (propriedades e métodos) que deve ser obedecida pela estrutura que o implementa
- A classe que adota ou implementa um protocolo deve obrigatoriamente implementar seus métodos e propriedades

A Linguagem Swift e suas particularidades

16. Extensões e protocolos

```
protocol ACCapable {
    var hasAC: Bool {get set}
    func turnAC(on: Bool)
}

class Bus: Vehicle, ACCapable {
    //Precisa implementar a propriedade
    var hasAC: Bool

    init(model: String, hasAC: Bool) {
        self.hasAC = hasAC
        super.init(model: model)
    }

    //Precisa implementar o método
    func turnAC(on: Bool) {
        if on {
            print("Ligando ar-condicionado")
        } else {
            print("Desligando ar-condicionado")
        }
    }
}

let bus = Bus(model: "", hasAC: true)
```

2. Arquitetura de desenvolvimento iOS



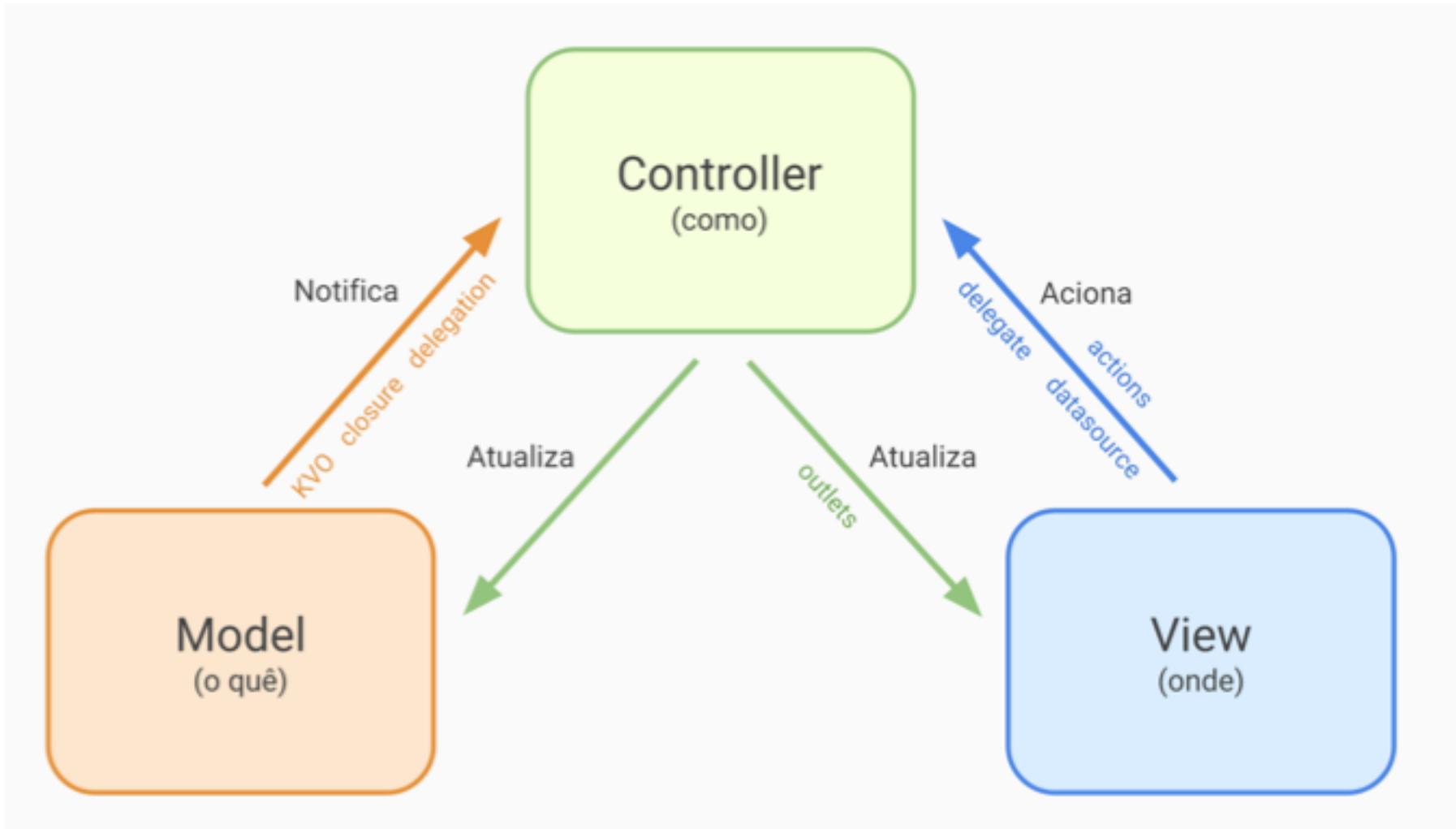
A Linguagem Swift e suas particularidades

16. Model-View-Controller

- MVC é o principal Arquitectural Pattern adotado nos frameworks de desenvolvimento iOS.
- Este Pattern atribui aos objetos 3 papéis principais: **Model**, **View** e **Controller**.
- Além de definir os papéis, define também a forma como esses objetos se comunicam.
 - **Model**: Responsáveis por encapsular os dados que são específicos de uma aplicação e também definir a lógica computacional que manipula e processa essa informação.
 - **Controller**: Define como uma informação deverá ser apresentada ao usuário. Atua como uma ponte entre os models de uma aplicação e suas view. Controla o ciclo de vida das views.
 - **View**: Responsável por apresentar as informações e interagir com o usuário.

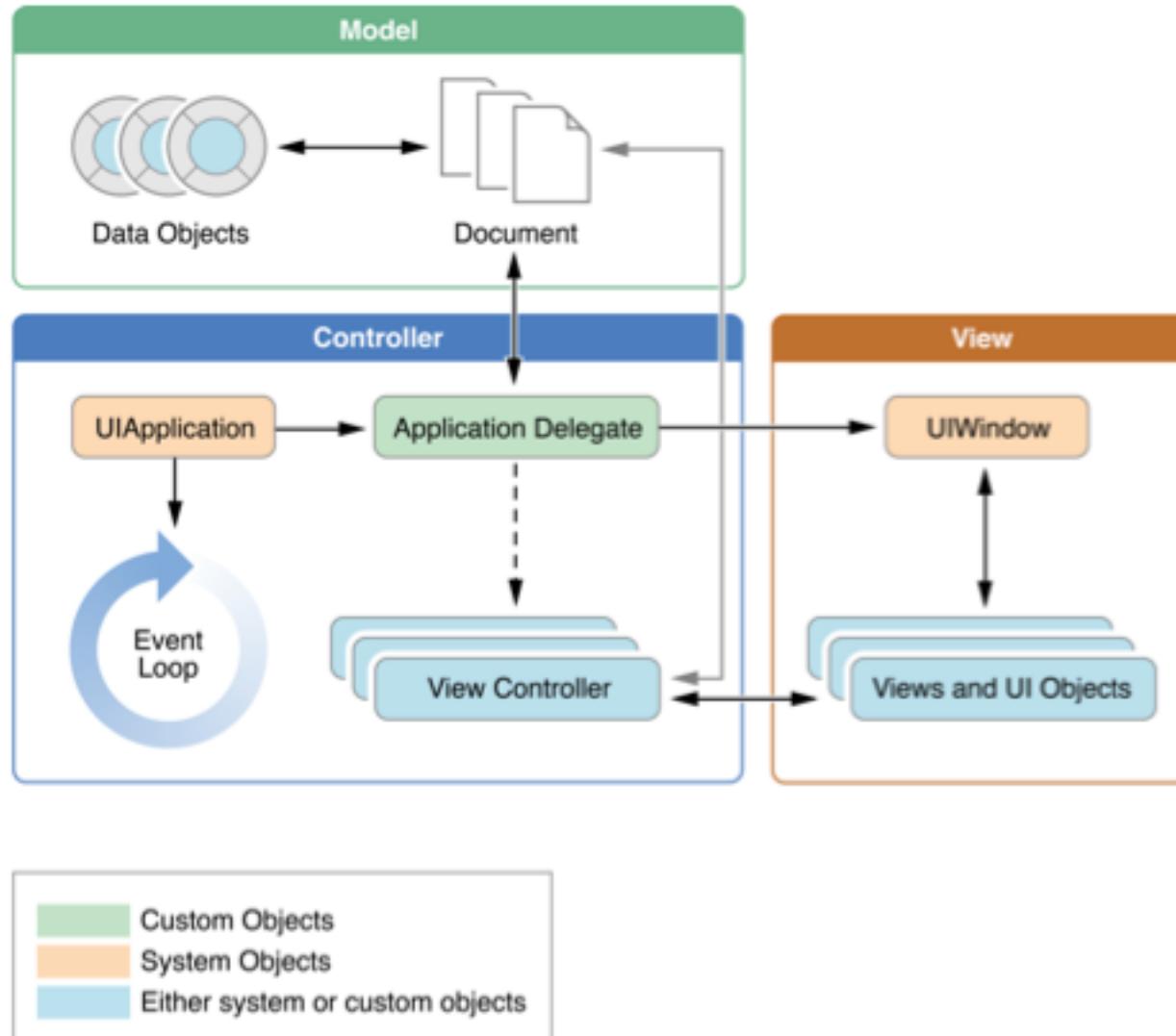
A Linguagem Swift e suas particularidades

16. Model-View-Controller



A Linguagem Swift e suas particularidades

16. Model-View-Controller



A Linguagem Swift e suas particularidades

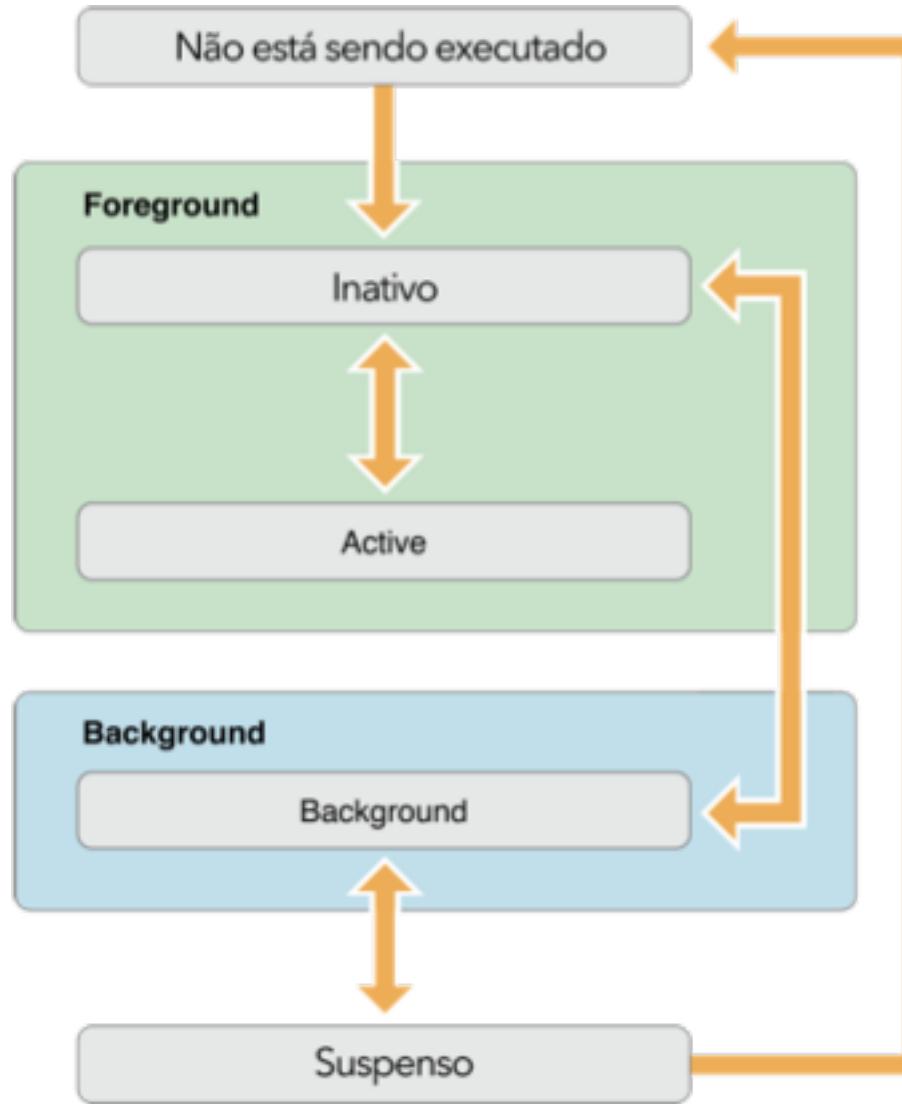
17. Ciclo de Vida - App

Estados de um aplicativo

- **Não está sendo executado:** O aplicativo não foi aberto ou foi fechado pelo Sistema
- **Inativo:** Está rodando mas ainda não está recebendo eventos. Fica nesse estado por um curto período de tempo.
- **Ativo:** Está sendo executado em primeiro plano e está recebendo eventos. É o estado natural quando o usuário está usando um app.
- **Background:** Está em background e está executando tarefas. Geralmente ficam por pouco tempo nesse estado até entrar no estado Suspenso.
- **Suspensão:** Está em background sem executar nenhuma tarefa. Fica nesse estado até o sistema notar falta de memória e decidir encerrá-lo.

A Linguagem Swift e suas particularidades

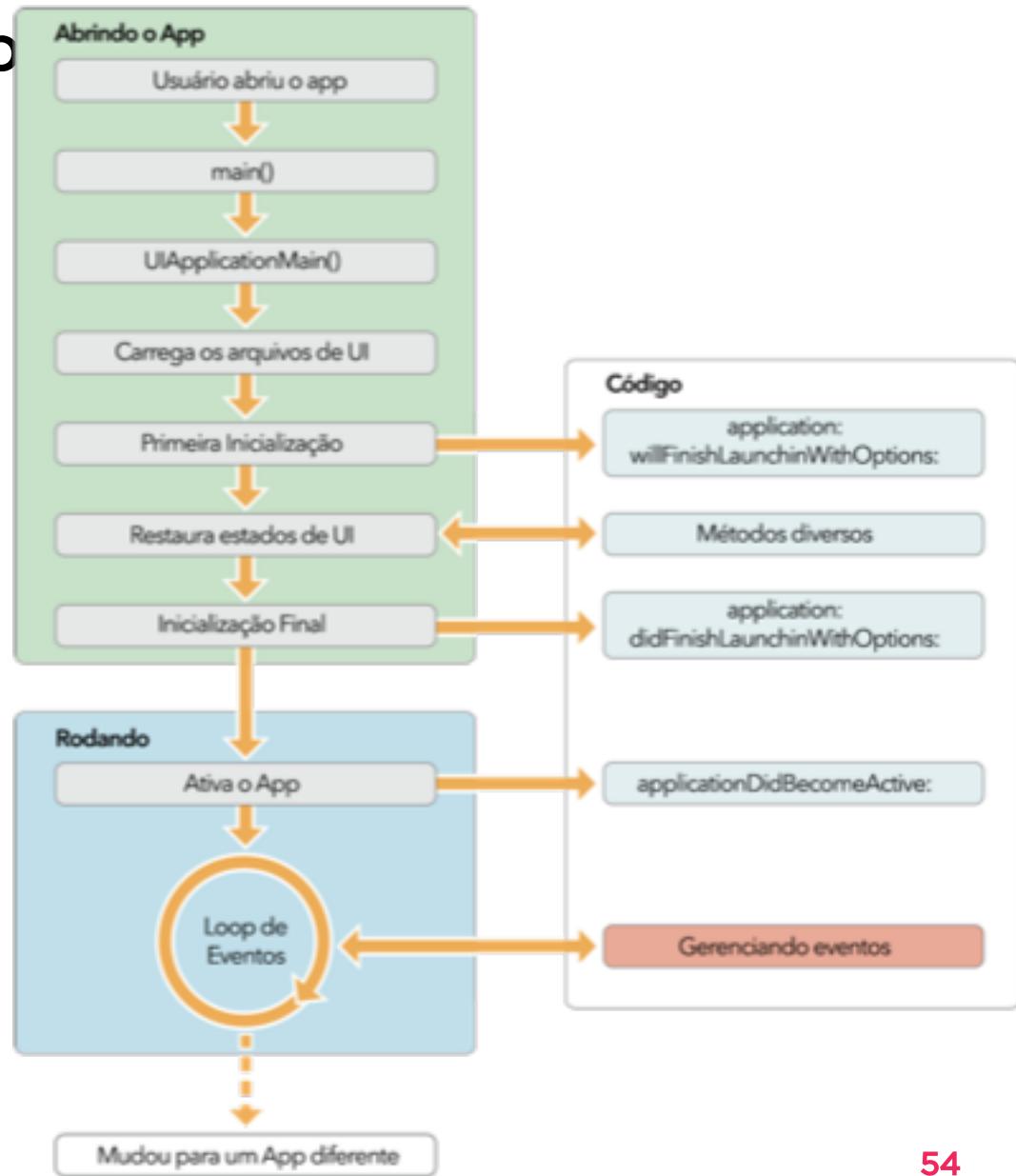
17. Ciclo de Vida - App



A Linguagem Swift e suas particularidades

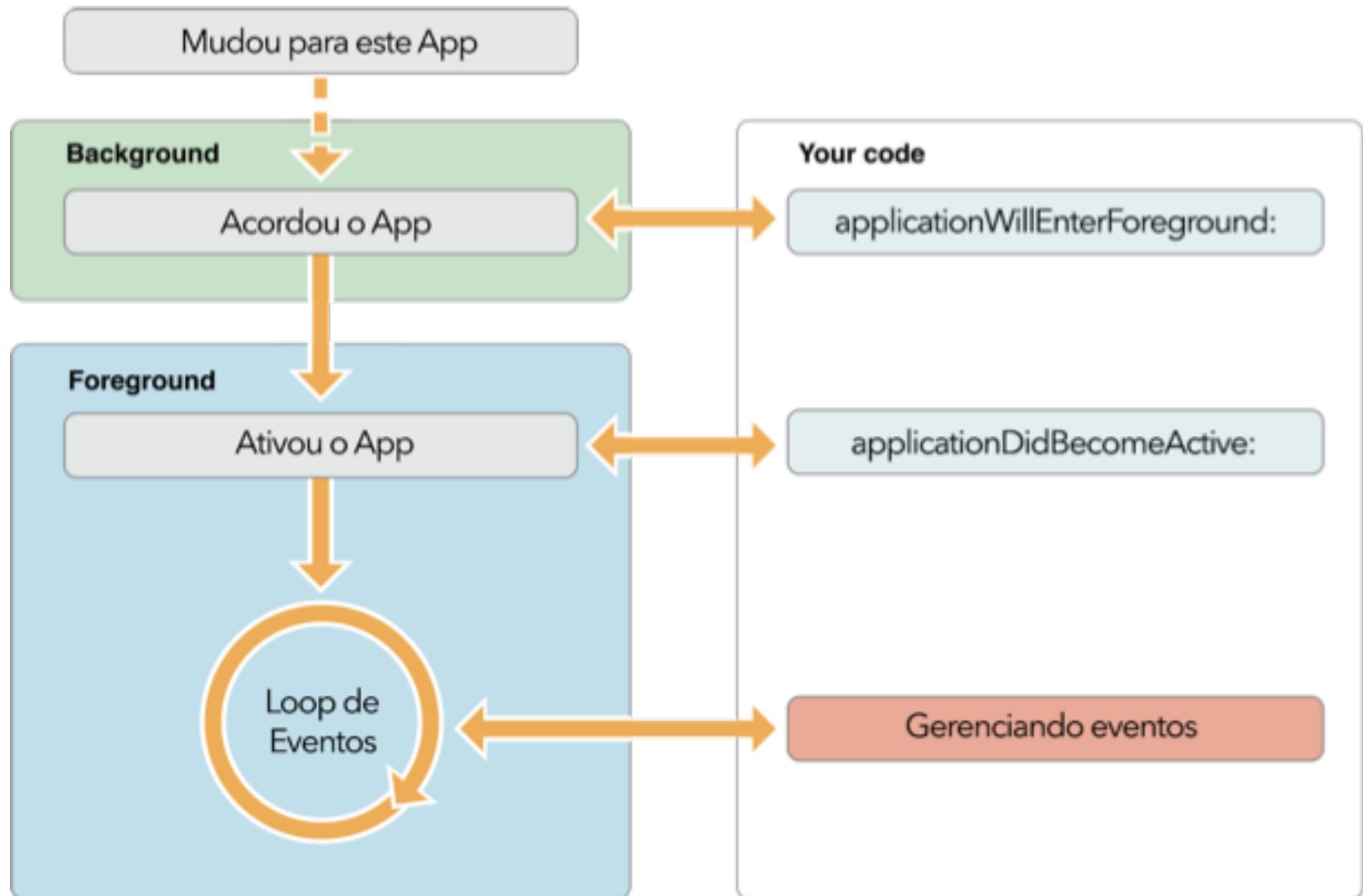
17. Ciclo de Vida - App

- É de extrema importância conhecermos todas as etapas que levam ao funcionamento de um App, desde o momento que o usuário toca no ícone do aplicativo até o seu encerramento.
- Através de métodos específicos, podemos tomar decisões com relação ao funcionamento do nosso App em diversos cenários distintos.



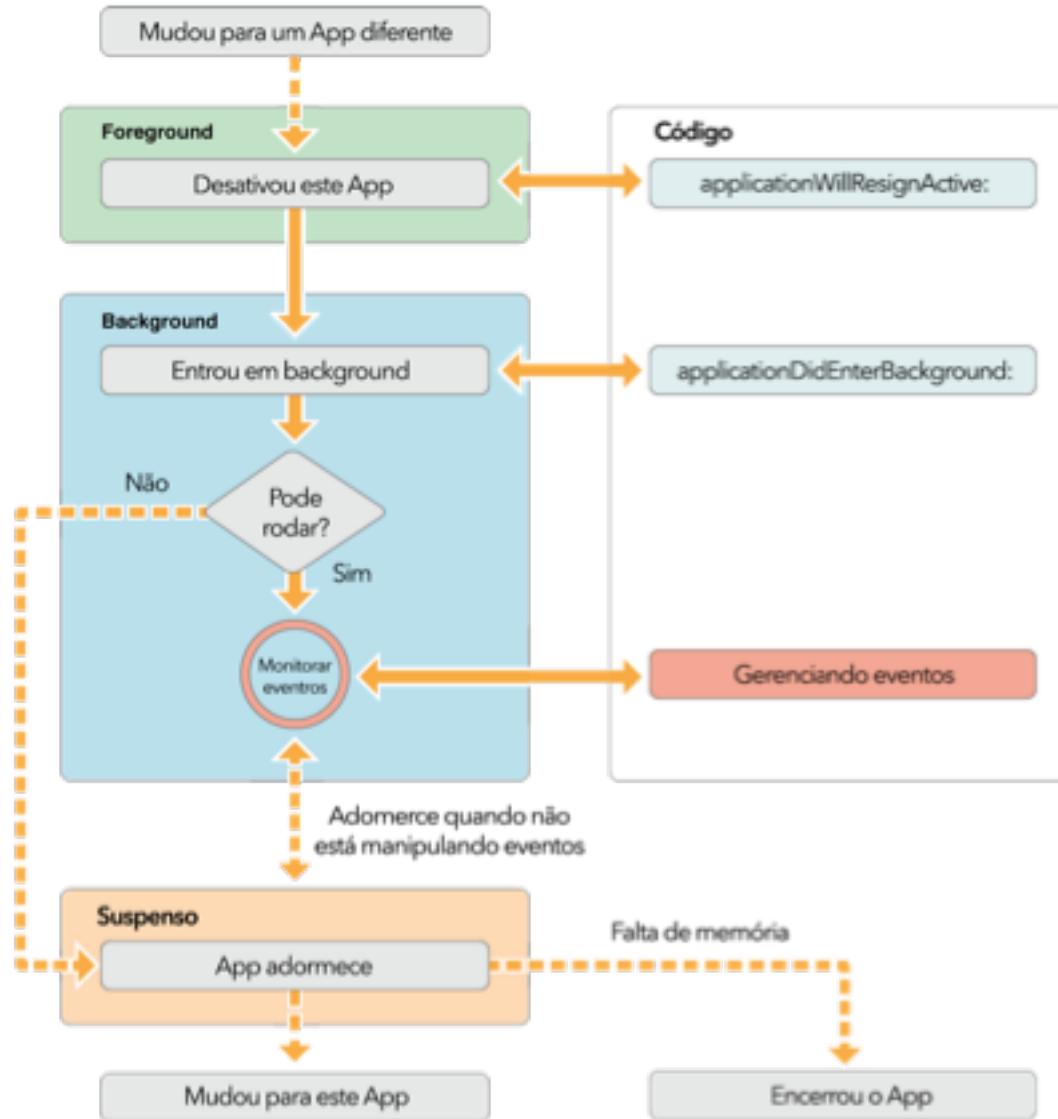
A Linguagem Swift e suas particularidades

17. Ciclo de Vida - App



A Linguagem Swift e suas particularidades

17. Ciclo de Vida - App



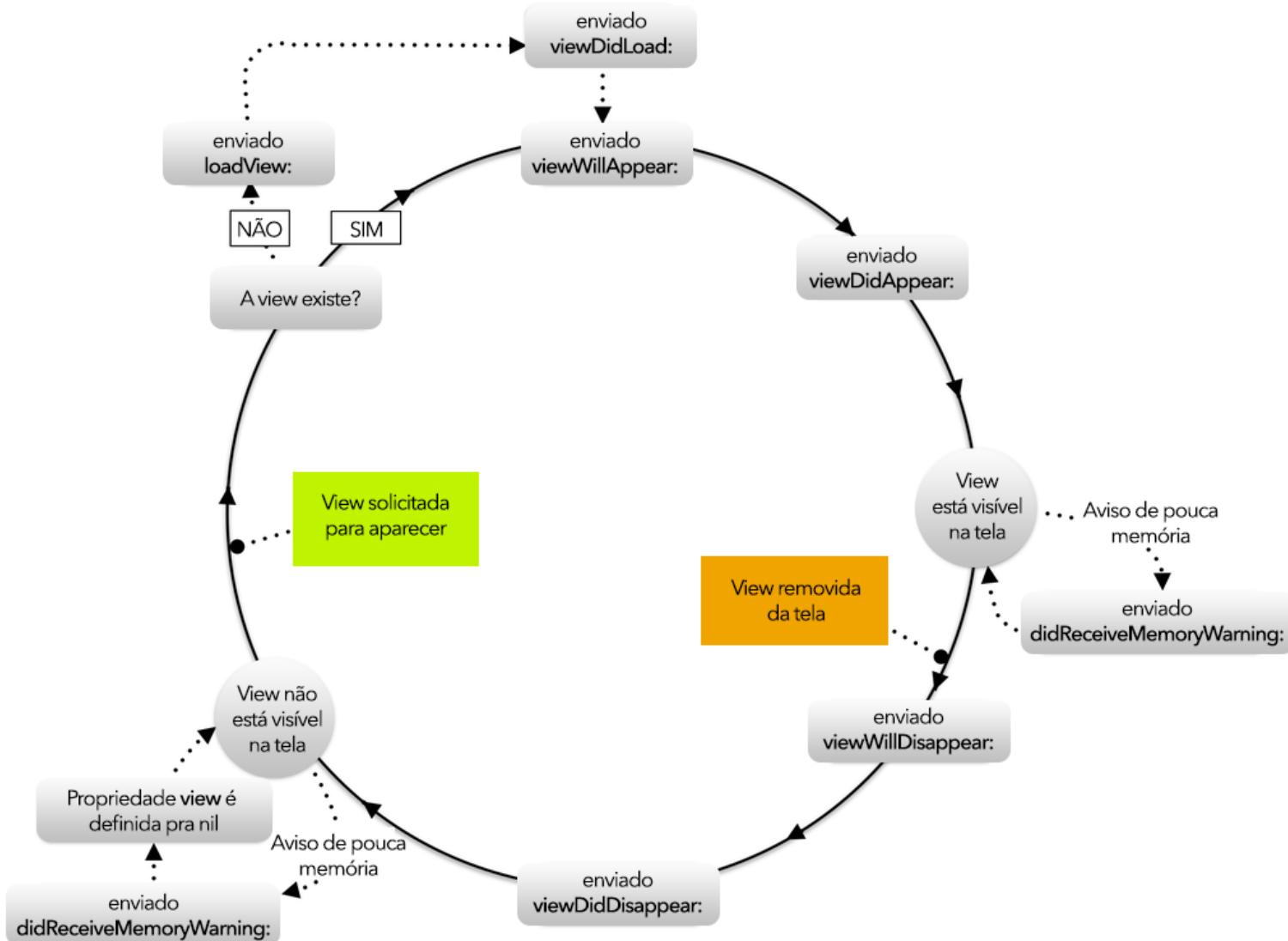
A Linguagem Swift e suas particularidades

18. UIViewController

- É a classe que fornece a estrutura para o gerenciamento das views de um aplicativo
- Responsável pela atualização dos conteúdos das views, geralmente em resposta às mudanças de dados que ela controla.
- Responde às interações do usuário com as views (toques, gestos, etc)
- Redimensiona as views e gerencia o layout geral da interface, respondendo às mudanças de orientação (portrait, landscape, etc) do device

A Linguagem Swift e suas particularidades

19. Ciclo de Vida - View



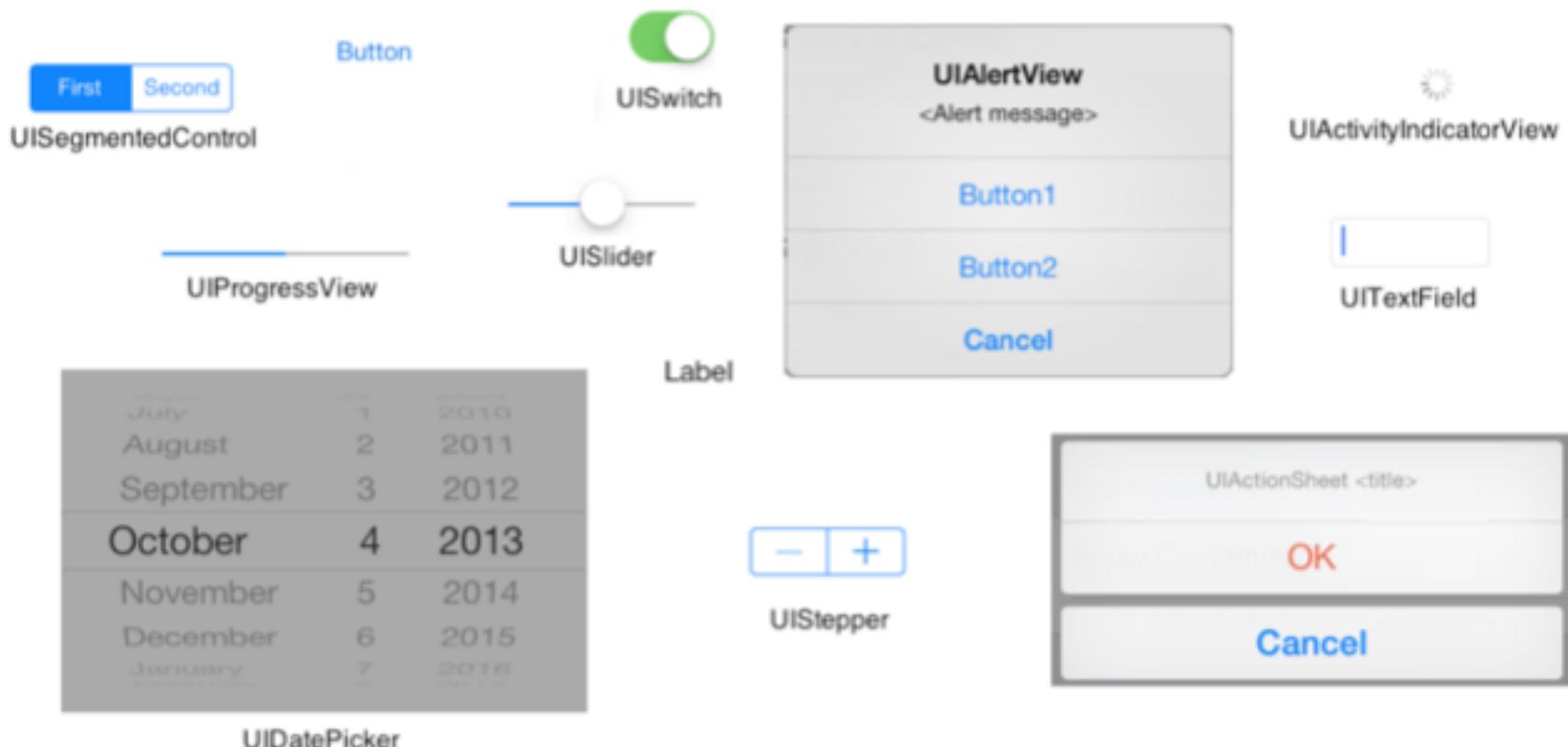
A Linguagem Swift e suas particularidades

19. UIKit

- Framework responsável por construir e gerenciar a interface do usuário em Apps para iOS, watchOS e tvOS
- É responsável por toda a arquitetura de Window e Views
- Coordena todo o gerenciamento de eventos e interação do usuário com o sistema
- Responsável por todos elementos visuais (imagens, textos, botões, views, etc), animações, bem como acesso aos sensores e integração do App com o sistema.

A Linguagem Swift e suas particularidades

19. UIKit

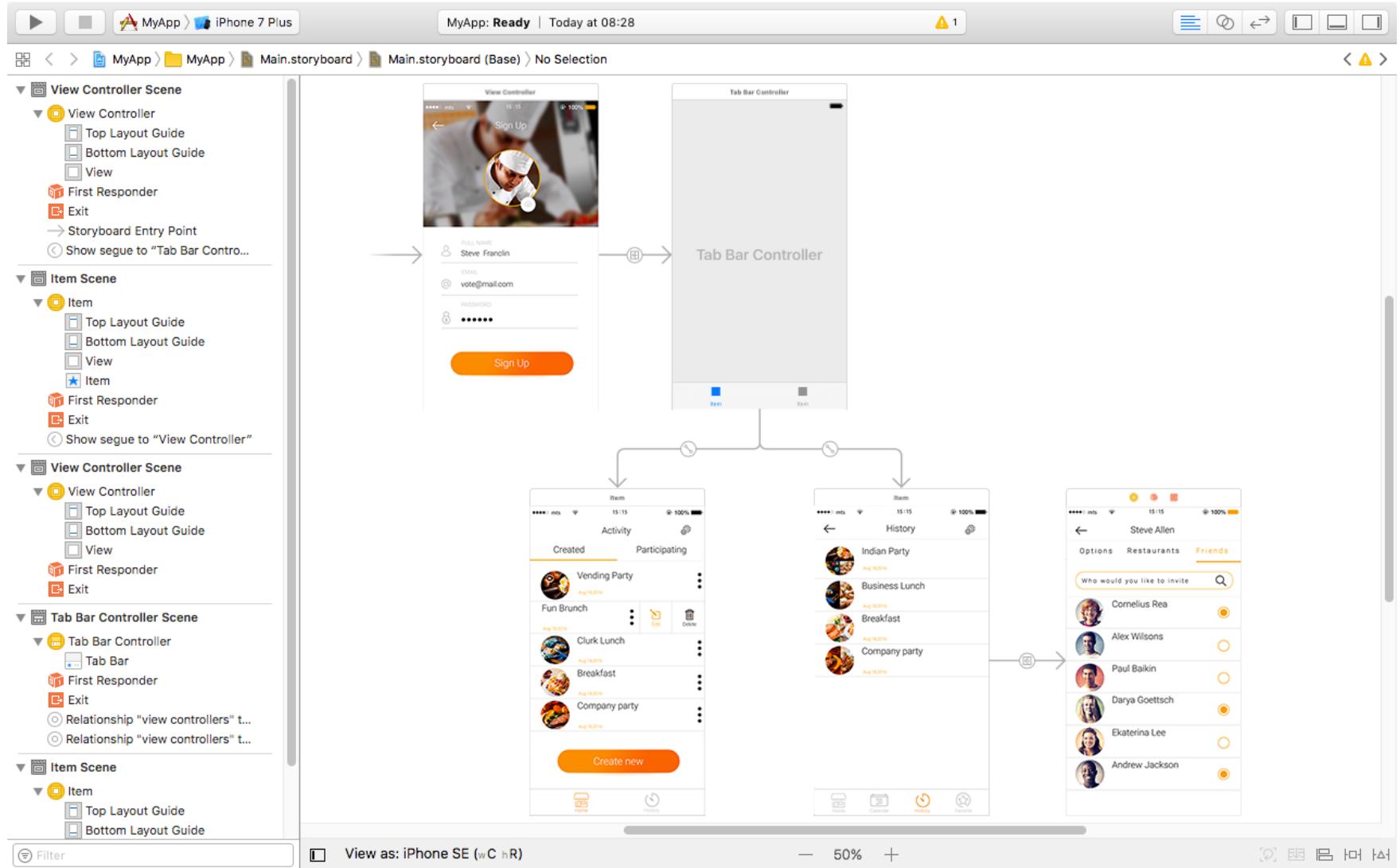


19. Storyboard

- Arquivo responsável por entregar uma representação visual de um aplicativo, mostrando seu conteúdo.
- Permite que você tenha uma visualização não só da aparência como do fluxo de sua aplicação
- É composto por cenas, que representam as ViewControllers e suas respectivas Views
- Conectamos as cenas usando segues

A Linguagem Swift e suas particularidades

19. Storyboard



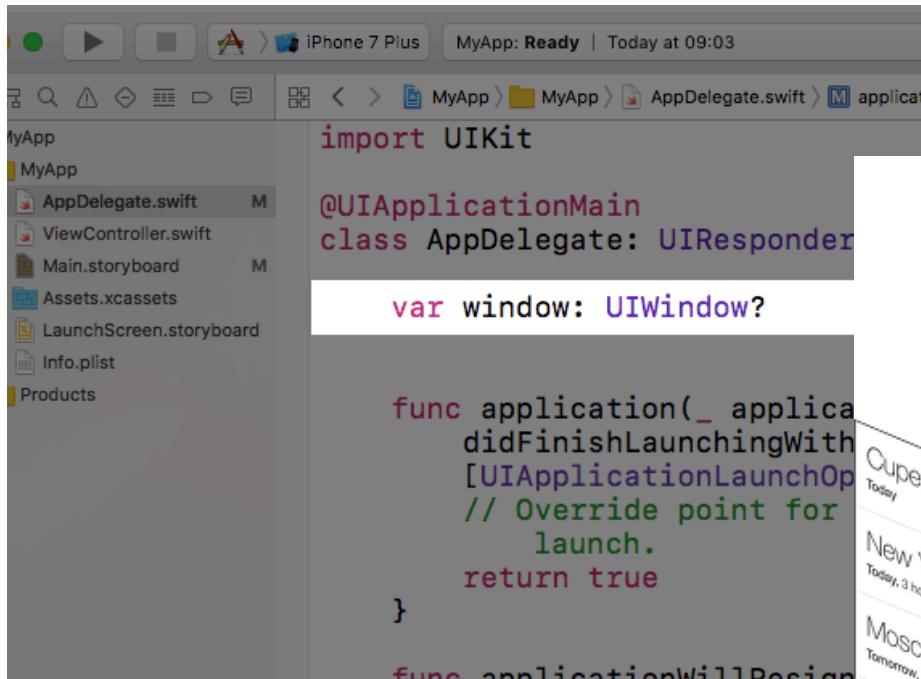
20. UIWindow, UIView

UIWindow

- Container que gerencia toda a interface (views) do aplicativo.
- É criada no AppDelegate (propriedade window)
- Cada App possui apenas uma Window, mas é possível usar mais de um Window caso seja necessário dividir a tela do App em outros displays
- É usada para gerenciar a ViewController que está sendo apresentada no App.

A Linguagem Swift e suas particularidades

20. UIWindow, UIView



```
import UIKit

@UIApplicationMain
class AppDelegate: UIResponder {

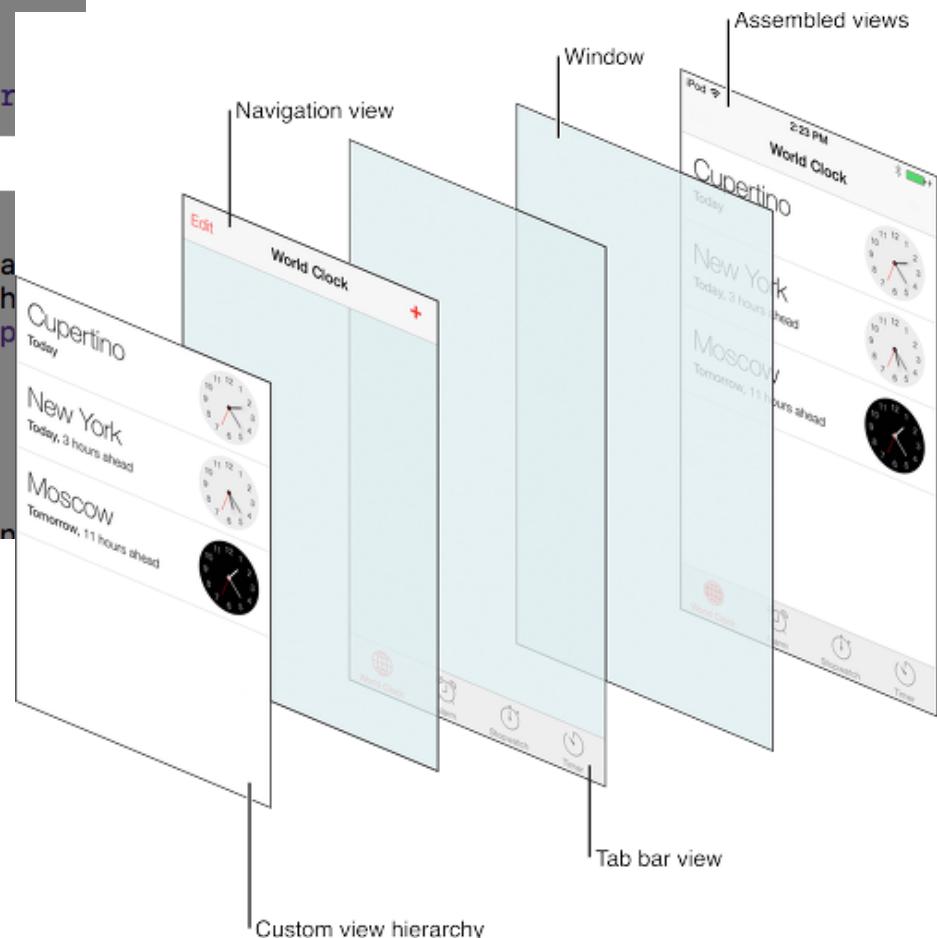
    var window: UIWindow?

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

    func applicationWillResignActive(application: UIApplication) {
        // Called when the application is about to move from active to inactive state. This can occur for certain types of temporary interruptions (such as an incoming phone call or SMS message) or when the user quits the application and it begins the transition to the background state.
        // Use this method to pause ongoing tasks, disable timers, and throttle down OpenGL ES frame rates. Games should use this method to pause the game.
    }

    func applicationWillEnterForeground(application: UIApplication) {
        // Called as part of the transition from the background to the foreground, during which, if applicable, you should reset any information modified by enterBackground:.
    }

    func applicationWillTerminate(application: UIApplication) {
        // Called when the application is about to terminate. Save data as appropriate. If your application supports background execution, this method is called before the user quits.
    }
}
```



A Linguagem Swift e suas particularidades

20. UIWindow, UIView

UIView

- Classe que define uma área retangular na tela e toda a interface para gerenciamento do conteúdo dessa área
- É a base para a maioria dos elementos gráficos presentes no UIKit
- É responsável pela não só pela renderização de qualquer elementos contido em sua área como também a interação com esses elementos
- É onde inserimos os elementos visuais do App (imagens, botões, textos, formulários, etc)
- Dentre suas responsabilidades, destacam-se
 - Renderização e animação
 - Layout e gerenciamento de subviews
 - Gerenciamento de eventos

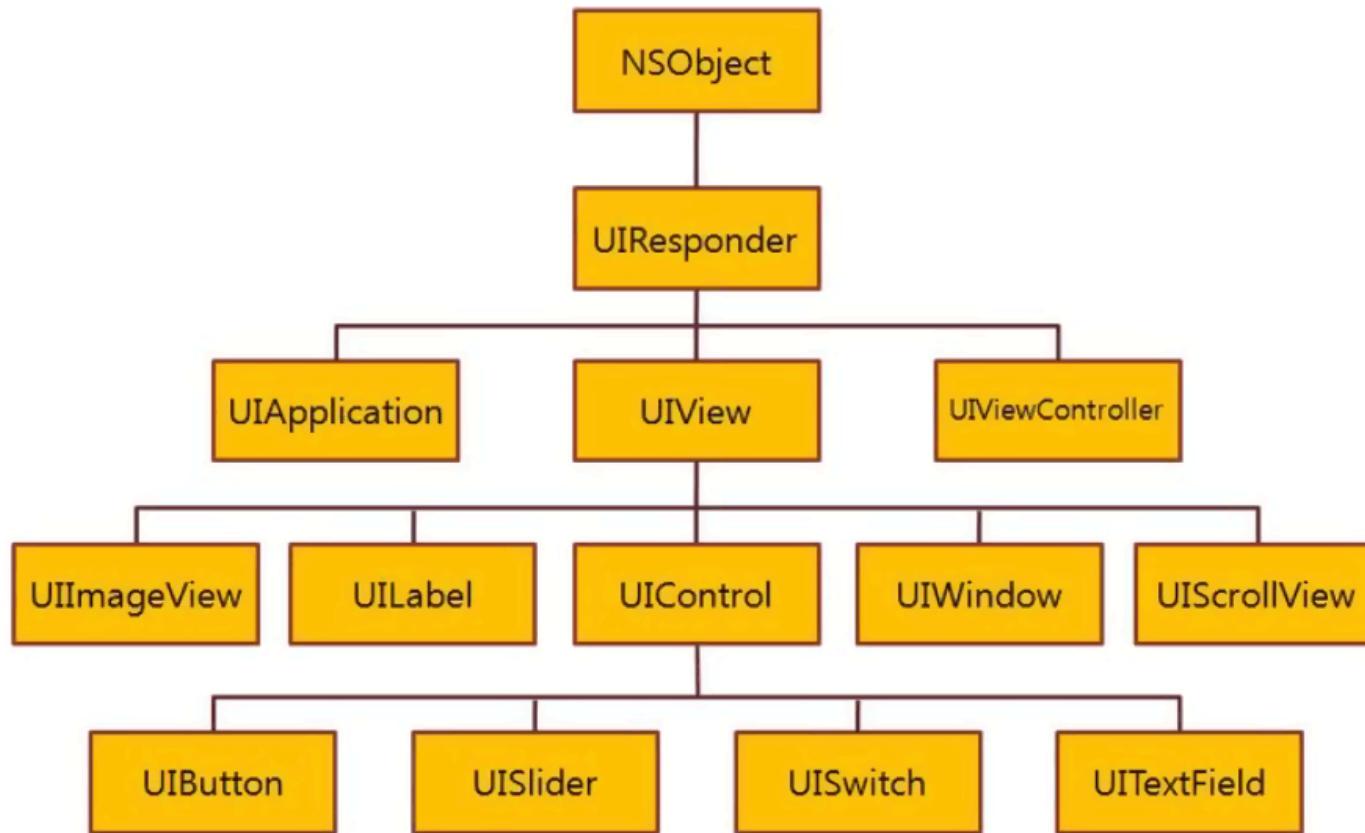
20. UIWindow, UIView

UIView

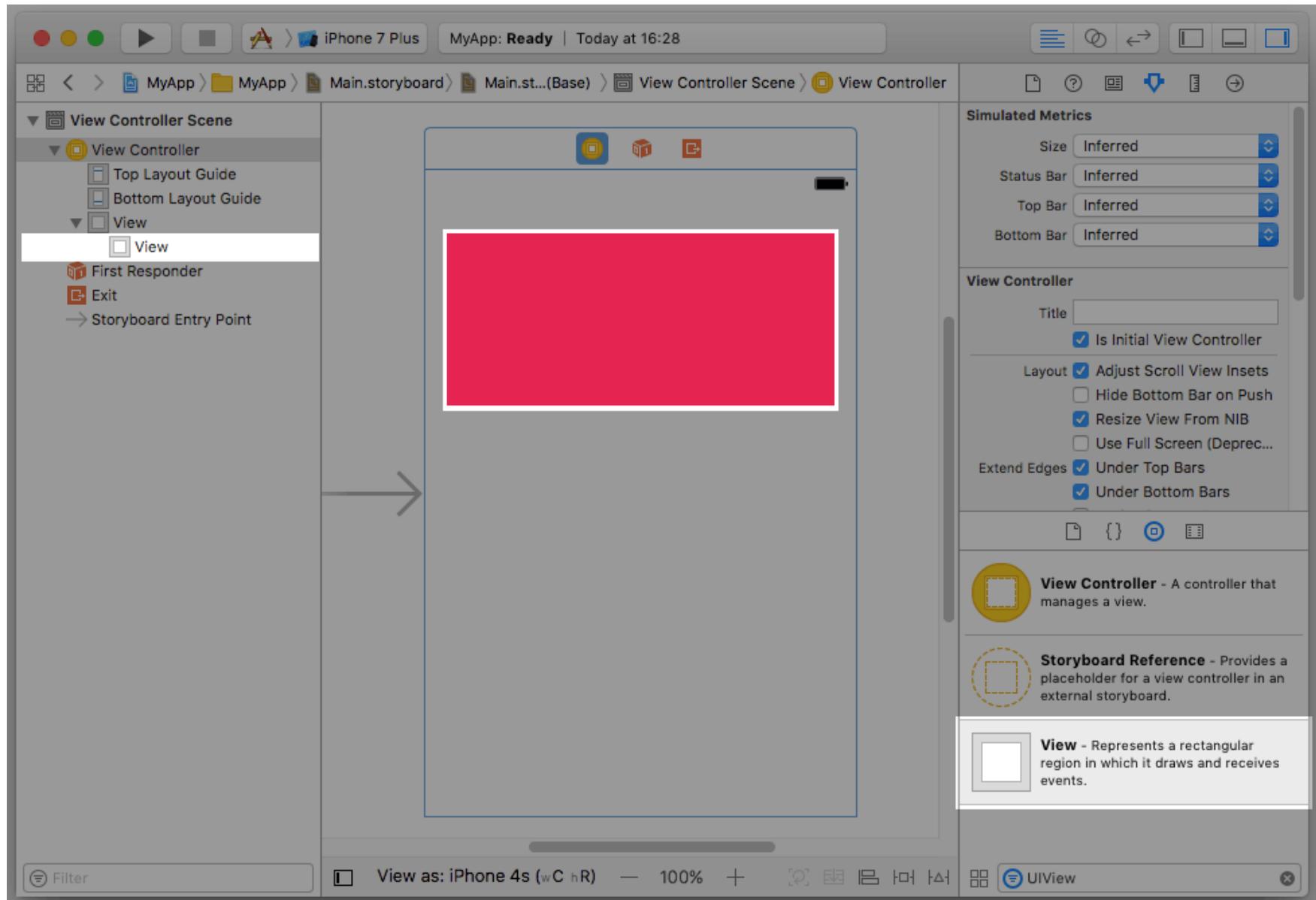
- O tamanho e a localização são definidos através das propriedades
 - **Frame**: retângulo que especifica o tamanho e localização da view de acordo com o sistema de coordenadas da sua superview
 - **Bounds**: especifica o tamanho de uma view de acordo com seu próprio sistema de coordenadas

A Linguagem Swift e suas particularidades

20. UIWindow, UIView



A Linguagem Swift e suas particularidades



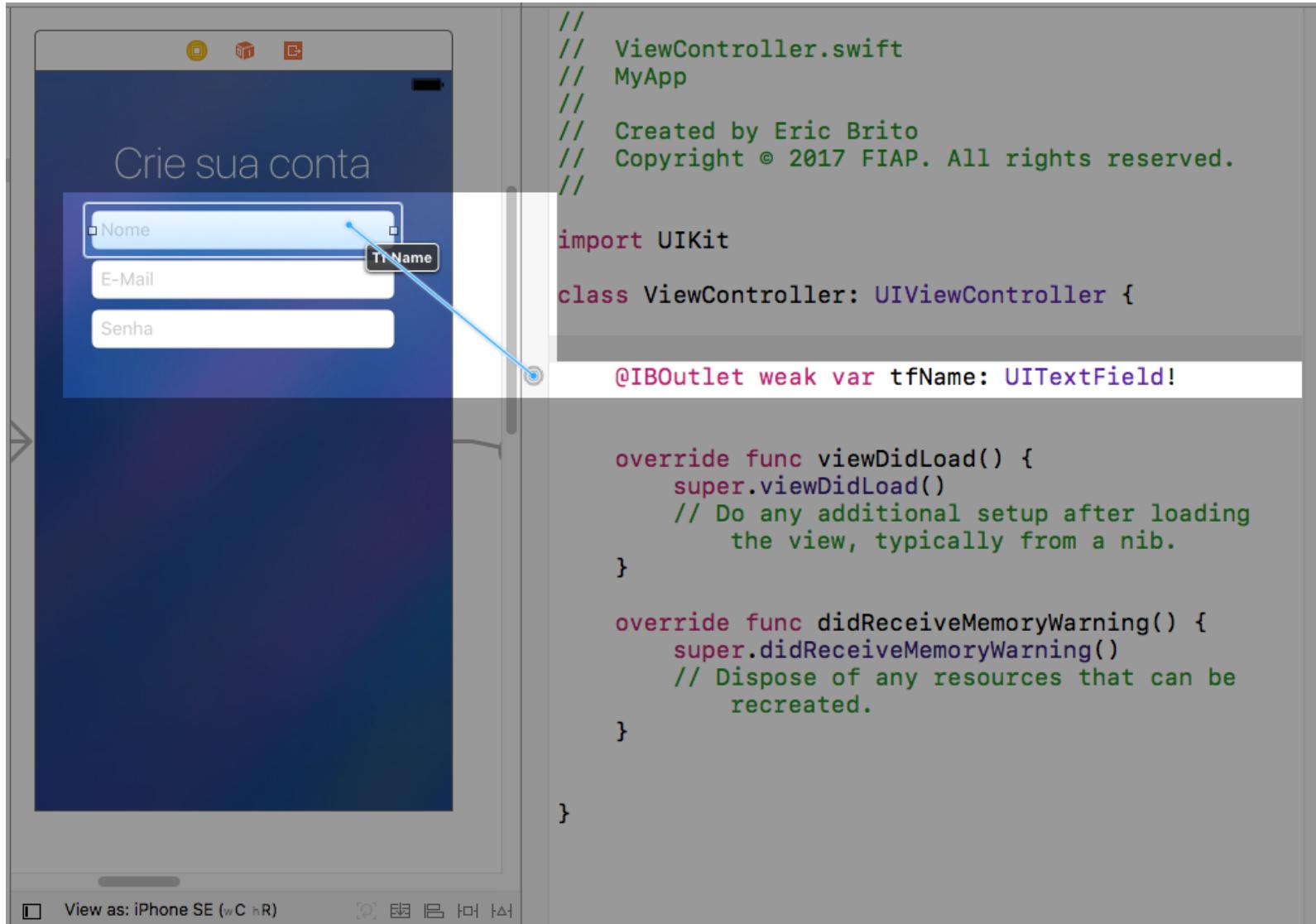
21. IBAction, IBOutlet

IBOutlet

- Qualificador de tipo responsável por referenciar um objeto em código a um elemento no Interface Builder (Área onde é apresentado o Storyboard e arquivos XIB)
- A criação/conexão do objeto IBOutlet com seu elemento correspondente é feita definindo o Xcode no modo **Assistant Editor**.

A Linguagem Swift e suas particularidades

21. IBAction, IBOutlet



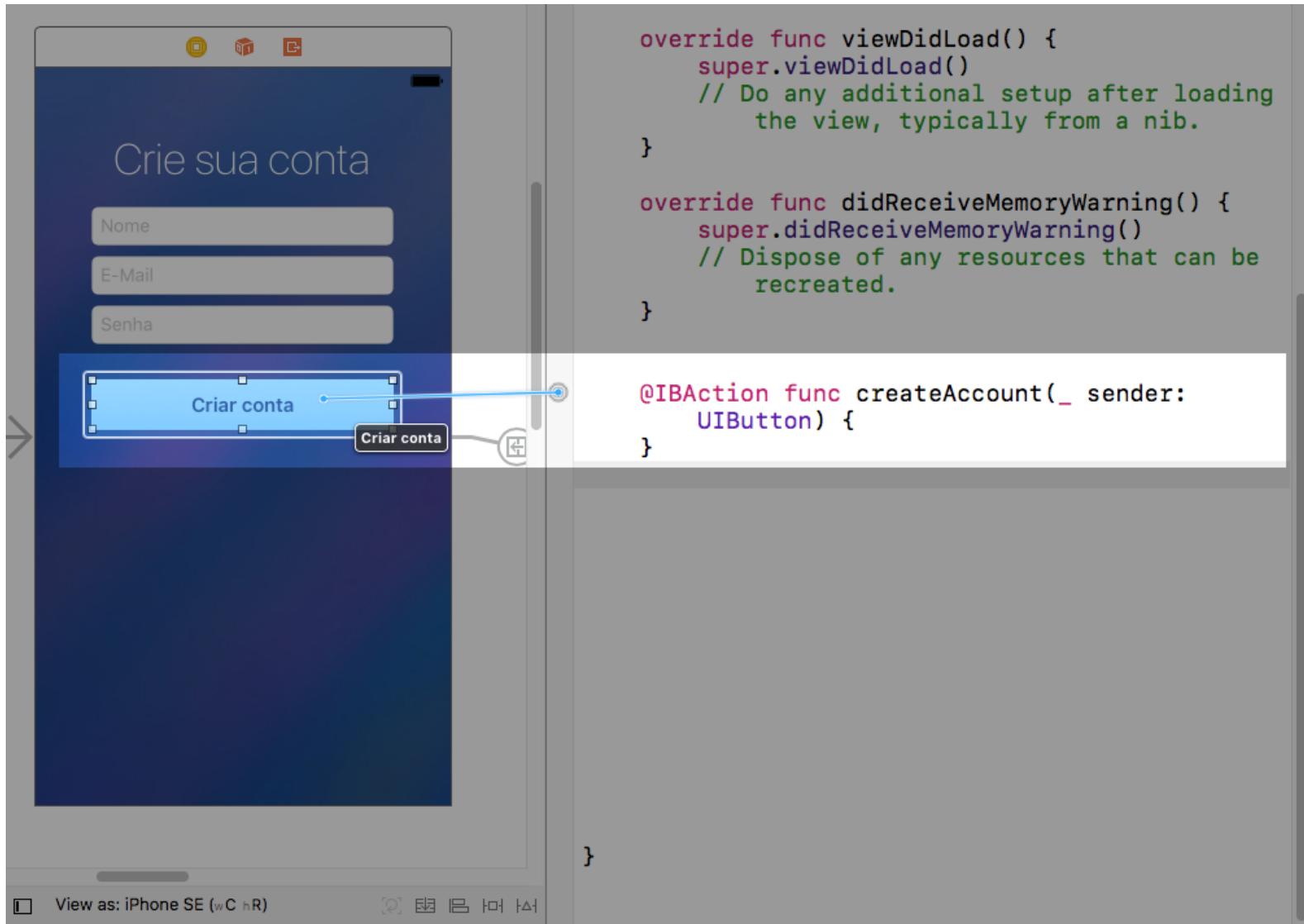
21. IBAction, IBOutlet

IBAction

- Qualificador de tipo responsável por expor um método como uma conexão entre um elemento visual e o código do aplicativo
- A criação/conexão do objeto IBOutlet com seu elemento correspondente é feita definindo o Xcode no modo **Assistant Editor**.
- Pode ser usado em elementos que disparam eventos (botões, etc).
- Podemos escolher a qual evento o método será vinculado e que parâmetros o método aceitará (sender, sender e evento)
- Uma mesma IBAction pode ser atribuída a vários elementos (respeitando o tipo do sender)
- O mesmo elemento pode disparar várias IBActions

A Linguagem Swift e suas particularidades

21. IBAction, IBOutlet



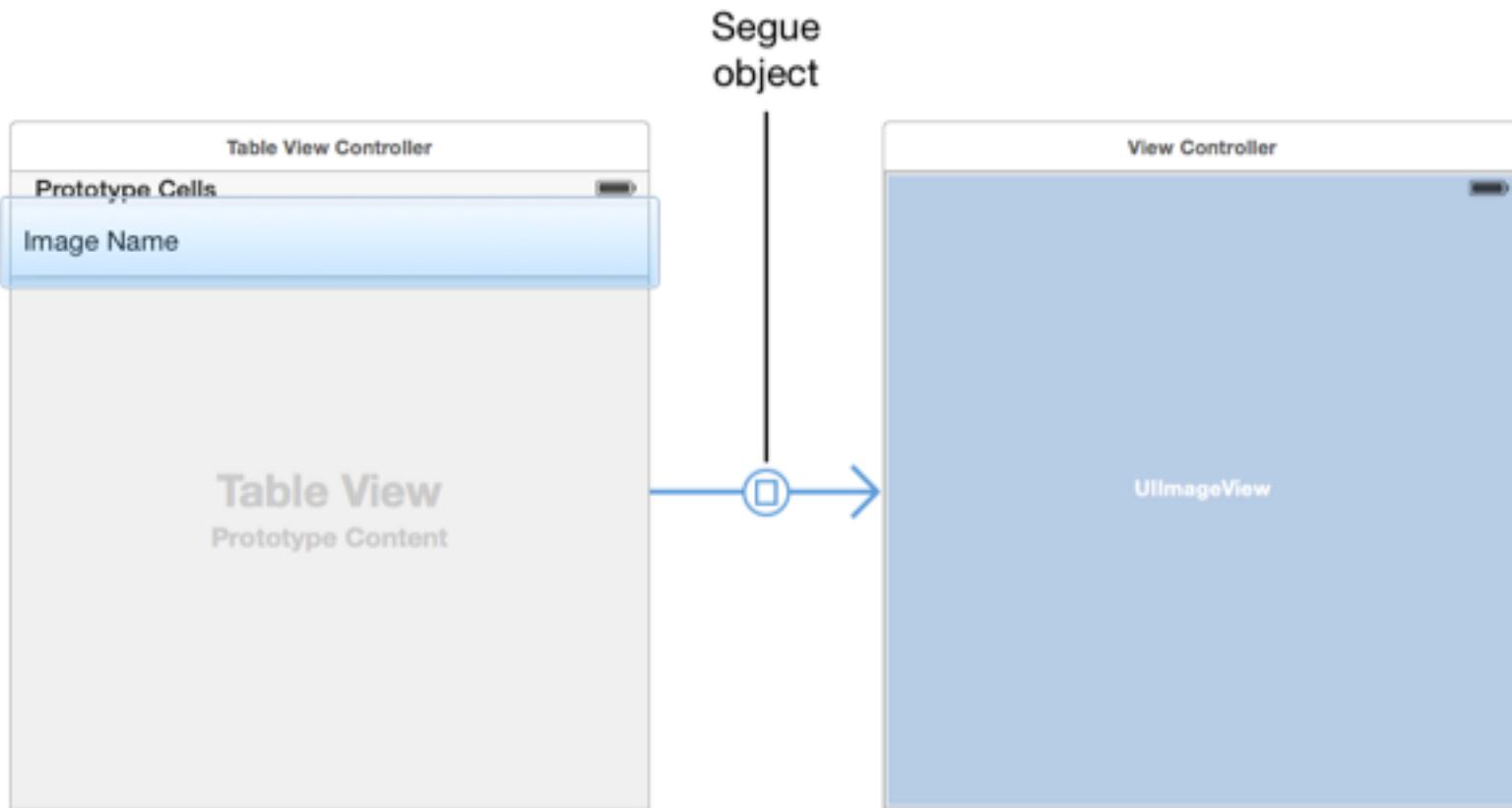
22. Navegação, Controllers, XIBs

Segue

- Definem o fluxo de navegação do Aplicativo, criando uma transição entre duas ViewControllers no Storyboard
- A conexão pode ser iniciada por um botão, uma célula em uma tabela, um gesto ou pode partir de uma própria ViewController e ser chamada via código
- Sempre são usadas para apresentar uma nova ViewController

A Linguagem Swift e suas particularidades

22. Navegação, Controllers, XIBs



A Linguagem Swift e suas particularidades

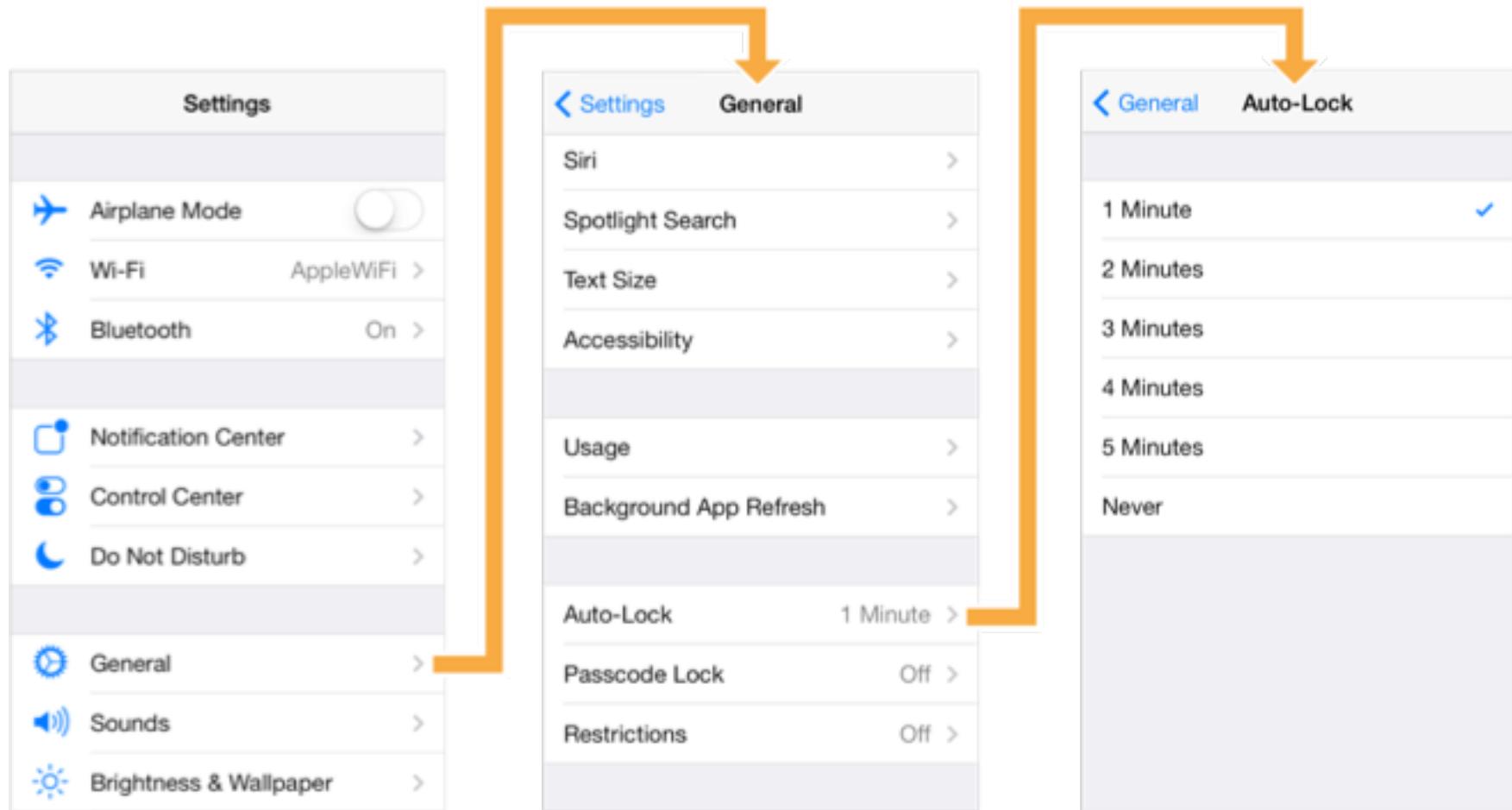
22. Navegação, Controllers, XIBs

UINavigationController

- Controller especializada em gerenciar a navegação hierárquica entre ViewControllers
- Fornece uma forma simples e automatizada de apresentação de novas view, bem como retorno para view previamente apresentadas
- As viewControllers são inseridas em um array (navigation stack). A primeira view que foi apresentada encontra-se no início do array, e a atual encontra-se no final do array

A Linguagem Swift e suas particularidades

22. Navegação, Controllers, XIBs



A Linguagem Swift e suas particularidades

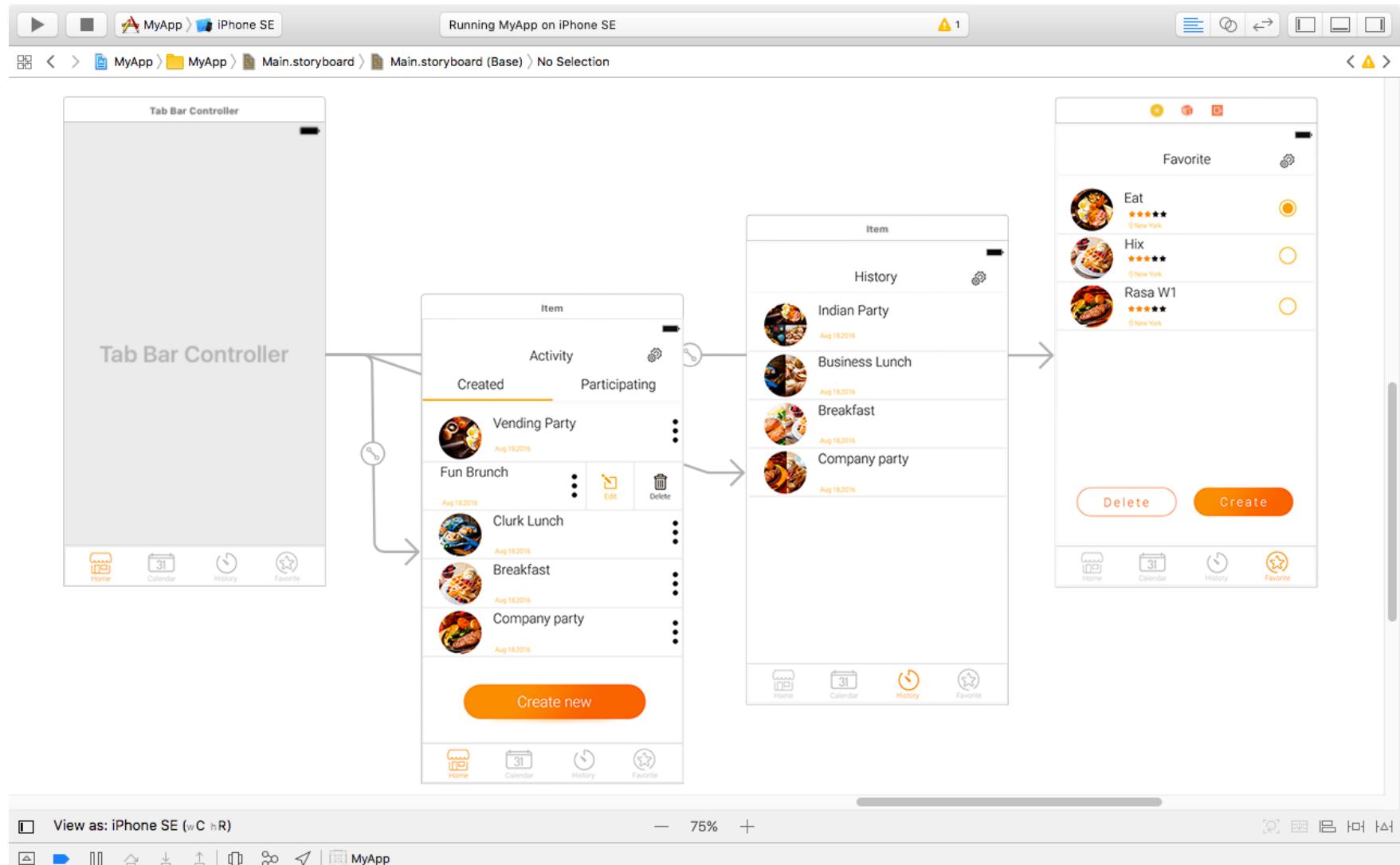
22. Navegação, Controllers, XIBs

UITabBarController

- Controller especializada em gerenciar a navegação entre ViewControllers utilizando “abas”
- Cada ViewController apresentada é associada a uma “aba” da barra
- Apenas uma ViewController é apresentada por vez, e ao ser apresentada ela se mantém em memória, ou seja, seu estado atual é mantido
- As ViewControllers associadas a uma TabBarController são armazenadas na propriedade viewController, que é um array de UIViewController

A Linguagem Swift e suas particularidades

22. Navegação, Controllers, XIBs



22. Navegação, Controllers, XIBs

XIB

- XIB: XML Interface Builder
- Representa uma UIView, um container para representação de uma tela do App
- É instanciada via código juntamente com sua respectiva ViewController

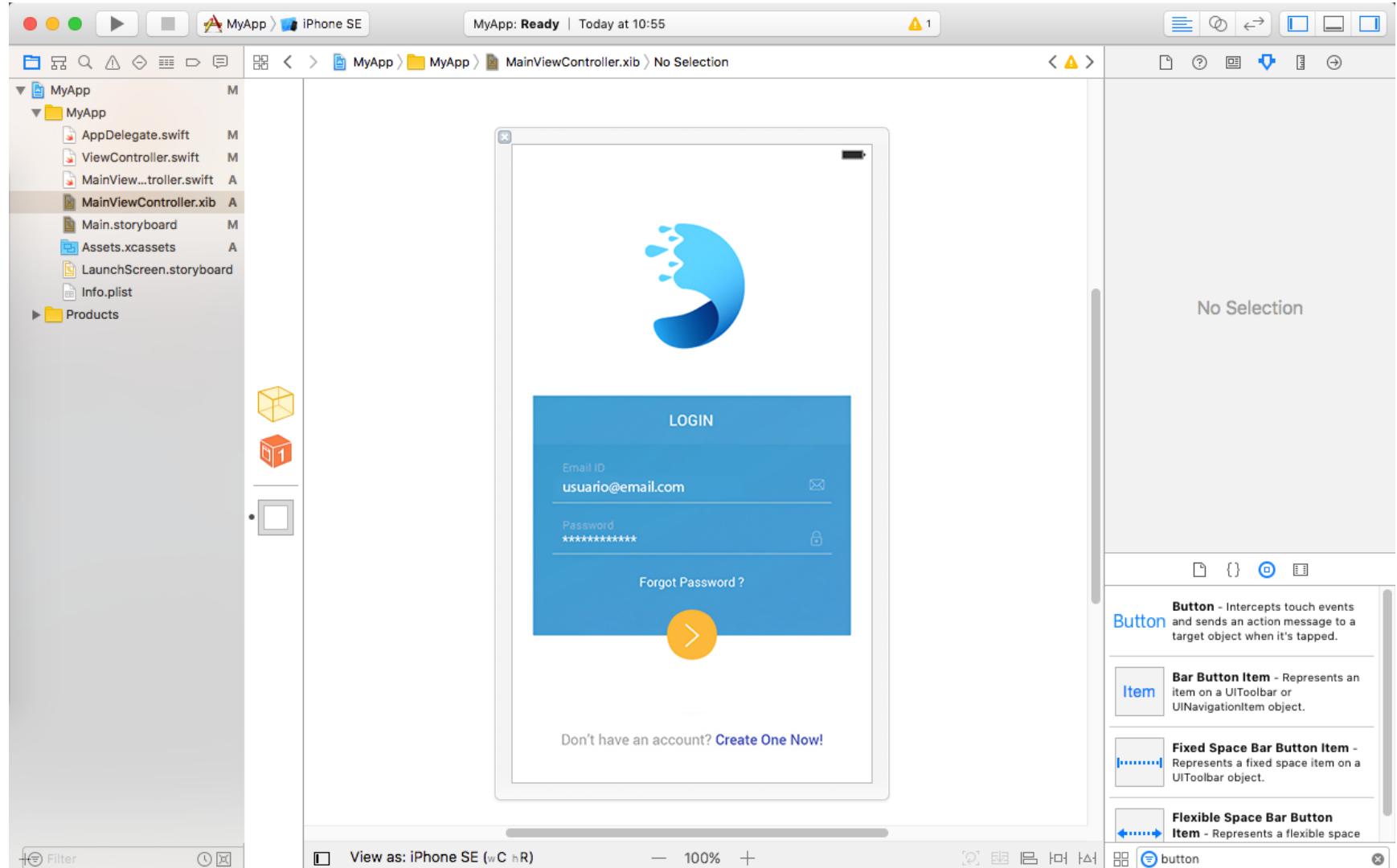
```
//Instanciando a ViewController juntamente com sua XIB
let mainScreen = MainViewController(nibName: "MainViewController", bundle: nil)

//Apresentando via NavigationController
navigationController?.pushViewController(mainScreen, animated: true)

//Apresentando Modalmente
present(mainScreen, animated: true, completion: nil)
```

A Linguagem Swift e suas particularidades

22. Navegação, Controllers, XIBs



A Linguagem Swift e suas particularidades

23. Delegate

Delegation

- Segundo a documentação da Apple:
“Delegation is a simple and powerful pattern in which one object in a program acts on behalf of, or in coordination with, another object”
- Delegação é um padrão simples e poderoso em que um objeto age no lugar de, ou em coordenação com, outro objeto.
- Em outras linguagens, semelhante a Listener e Callback
- A implementação é feita através de protocolos
- No UIKit diversos componentes fornecem protocolos para delegates
- É de suma importância que o objeto delegate seja implementado como weak para que não seja criada uma referência cíclica

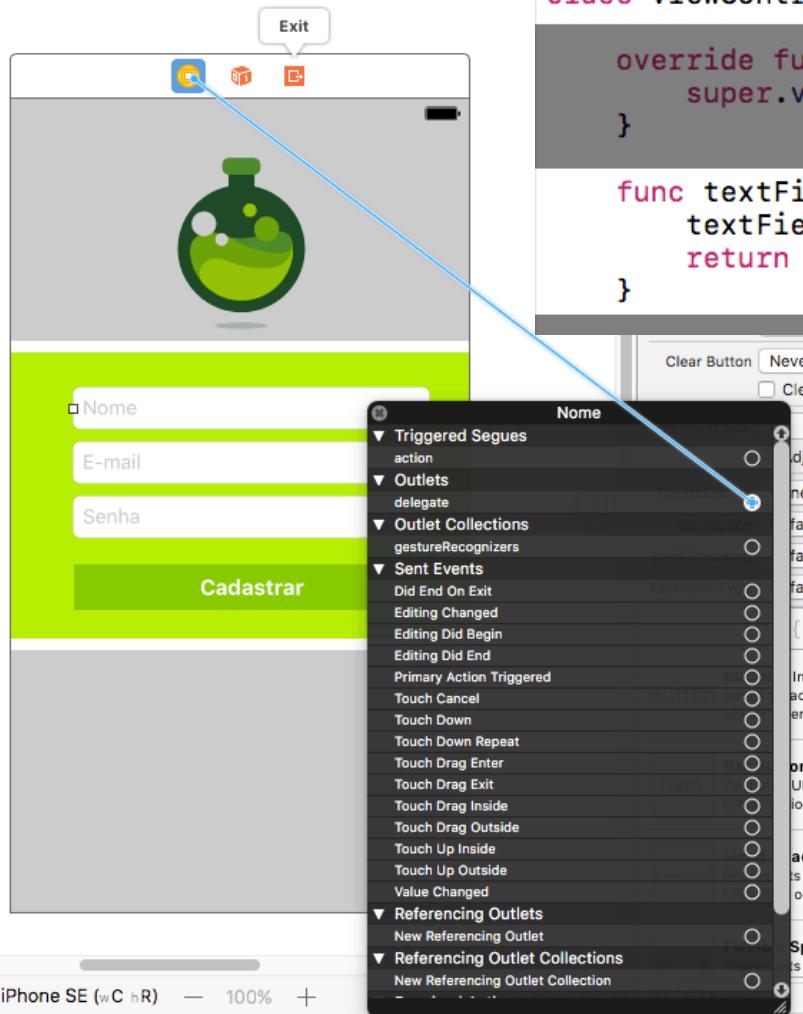
23. Delegate

Exemplo: Delegate do UITextField

- Definimos qual será o delegate do textField, ou seja, quem será responsável pela tomada de decisão quando o textField executar uma tarefa. No nosso exemplo, a própria classe ViewController será o delegate.
- Implementamos o protocolo UITextFieldDelegate na classe, para que ela possa ser delegate do TextField
- Implementamos os métodos que desejamos e/ou precisamos utilizar deste protocolo

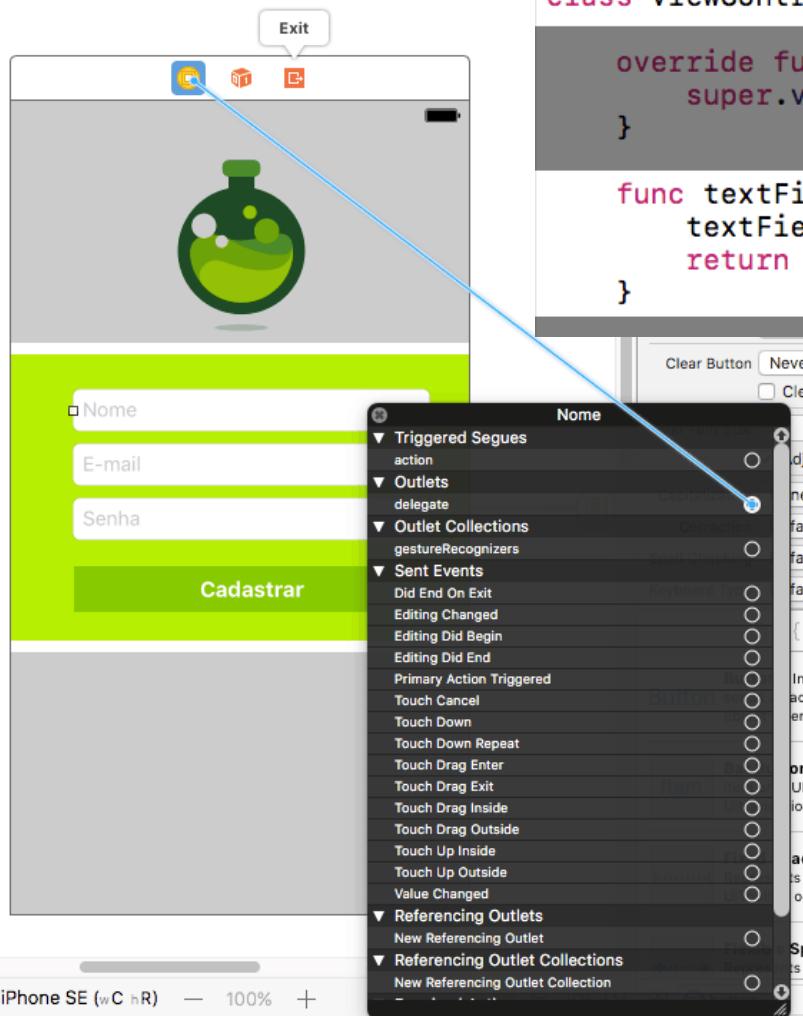
A Linguagem Swift e suas particularidades

23. Delegate



A Linguagem Swift e suas particularidades

23. Delegate



```
import UIKit

class ViewController: UIViewController, UITextFieldDelegate {

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        textField.resignFirstResponder()
        return true
    }
}
```

A Linguagem Swift e suas particularidades

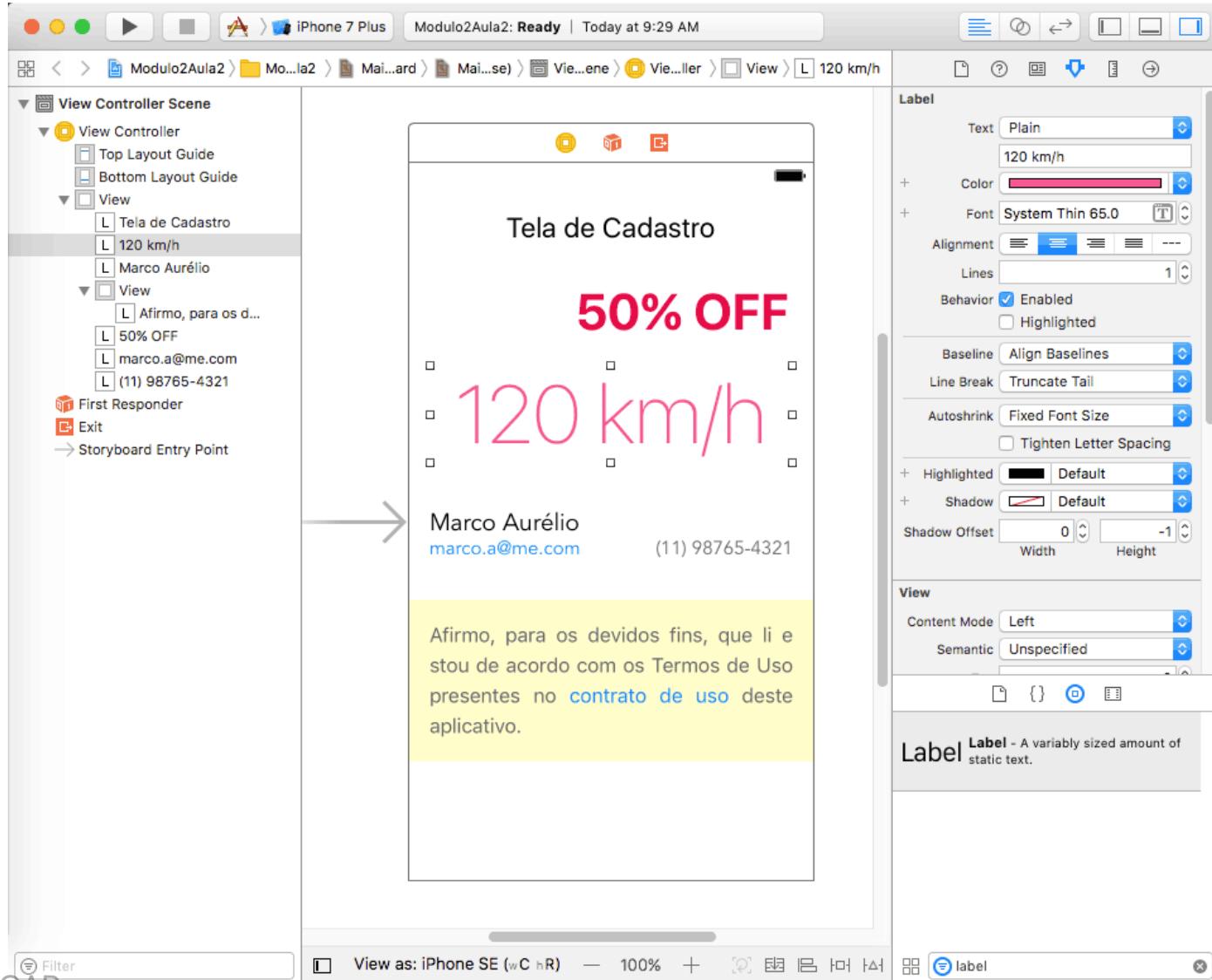
24. UILabel, UIButton

UILabel

- Usada para apresentar textos (somente leitura) ao usuário
- Pode ser apresentado em uma ou múltiplas linhas
- Dentre as propriedades configuráveis estão cor, fonte, alinhamento, etc
- Pode ser simples (**plain**) ou com múltiplas configurações (**attributed**)
- Principais atributos
 - **Text**: Define o conteúdo (texto) e também o tipo do Label. Pode ser Plain (texto simples) ou Attributed (com múltiplas configurações).
 - **Color**: Controla a cor do texto do label
 - **Font**: Define a fonte usada em no texto do label
 - **Alignment**: Controla o alinhamento horizontal do texto. Pode ser left, right,
 - center, justified, ou natural (definido pelos parâmetros de região do App)
 - **Lines**: Define o número máximo de linhas
 - **Line Break**: Define a forma como o texto será tratado quando não couber no seu container. Clip (cortar), Character/Word Wrap

A Linguagem Swift e suas particularidades

24. UILabel, UIButton



A Linguagem Swift e suas particularidades

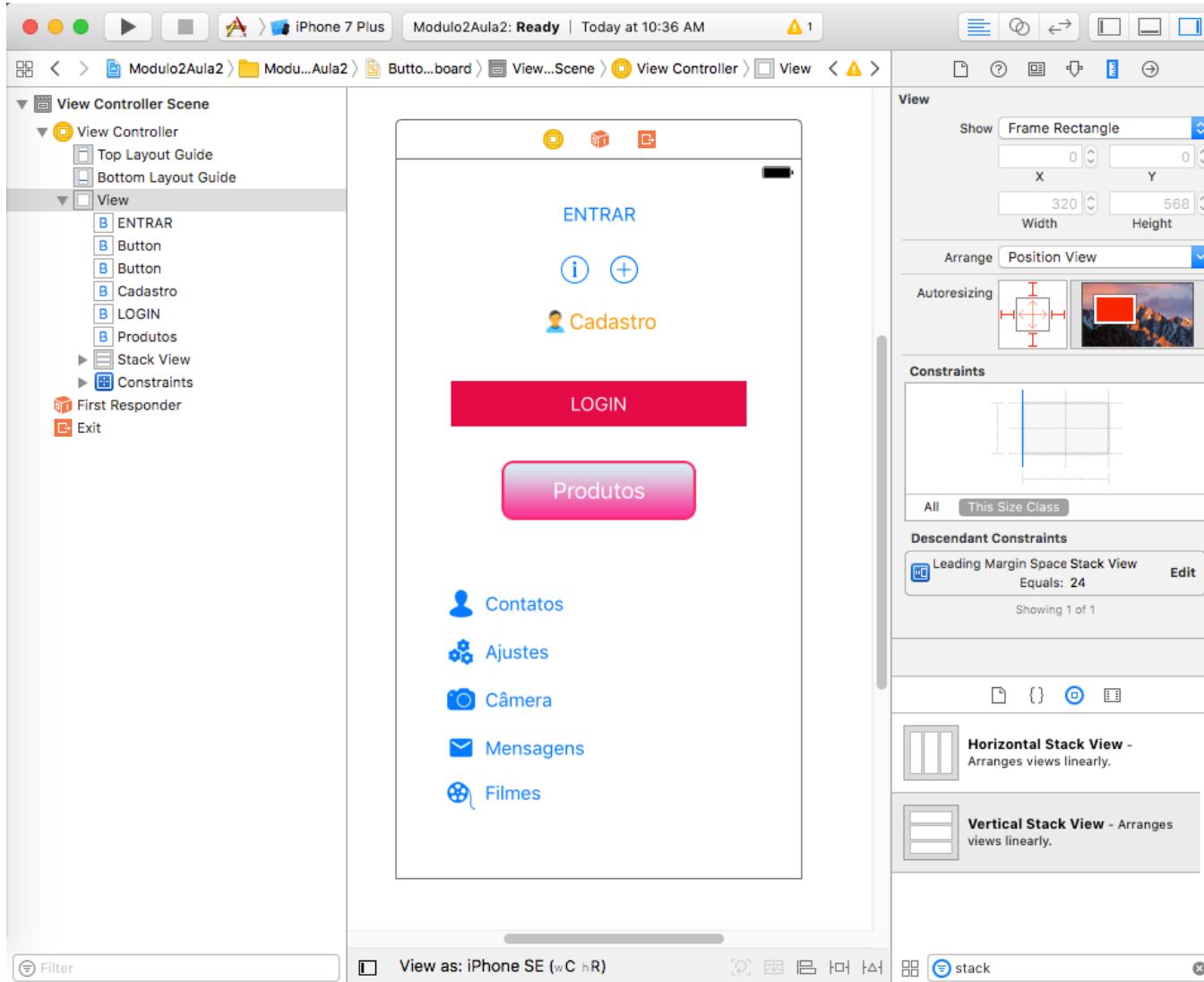
24. UILabel, UIButton

UIButton

- View capaz de executar códigos em resposta à interação do usuário
- Conecta-se ao código usando @IBAction
- Pode ser configurado para usar textos ou imagens (foreground e background)
- Sua aparência pode ser definida através de 5 estados: **default**, **highlighted**, **focused**, **selected** e **disabled**
- Principais atributos
 - **Type**: Define o tipo do botão, o que também define os comportamentos padrões do botão para outros atributos. Não pode ser alterado em tempo de execução. Dentre os valores, estão System (botões padrões do sistema) e Custom (botões com características e comportamentos customizados)
 - **State Config**: Seleciona o estado do botão, dando acesso às suas configurações
 - **Title**: Título (texto) do botão
 - **Image**: Imagem do botão.
 - **Background**: Imagem de fundo do botão. É apresentada atrás do

A Linguagem Swift e suas particularidades

24. UILabel, UIButton



A Linguagem Swift e suas particularidades

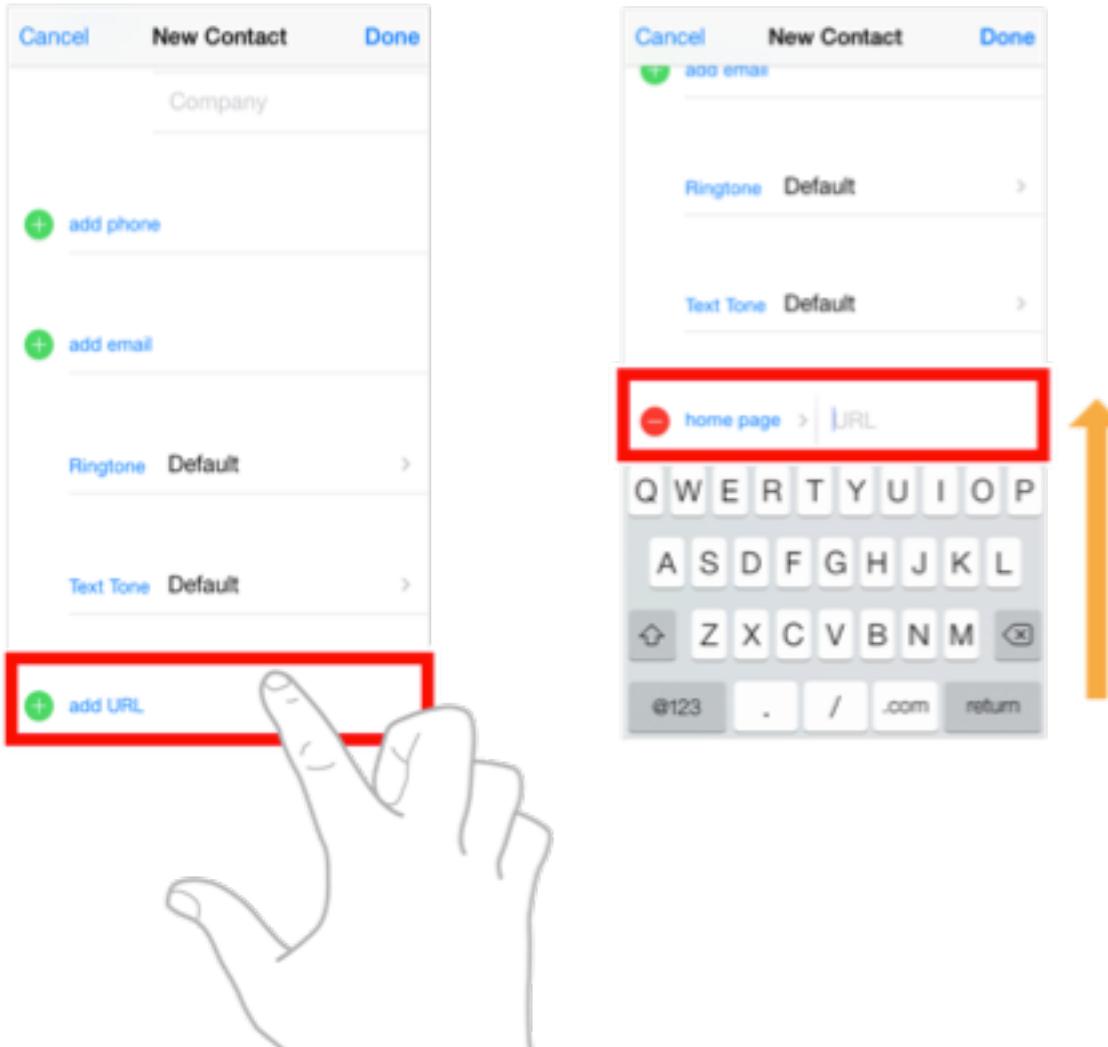
25. UITextField, UITextView

UITextField

- Apresenta ao usuário uma área de texto editável.
- Ideal para quando precisamos que o usuário entre com pequenas quantidades de texto
- Utiliza actions e delegates para reportar mudanças ocorridas durante sua utilização
- O sistema automaticamente apresenta o teclado ao usuário quando editamos um textField
- Principais atributos
 - **Text:** Responsável pelo texto apresentado no campo. É a propriedade que usamos para recuperar o que o usuário digitou.
 - **Color / Font:** Cor e fonte do texto
 - **Alignment:** Alinhamento do texto dentro da área de edição.
 - **Placeholder:** Texto apresentado quando o campo estiver vazio. Usado para indicar ao usuário o tipo de informação que o campo utiliza.
 - **Background:** Imagem de fundo do textField.
 - **Border Style:** Estilo visual do botão (borda).
 - **Clear Button:** Define quando o botão clear será apresentado (nunca, enquanto editando, a menos que esteja editando, sempre)
 - **Keyboard Type:** Tipo do teclado que será apresentado ao usuário

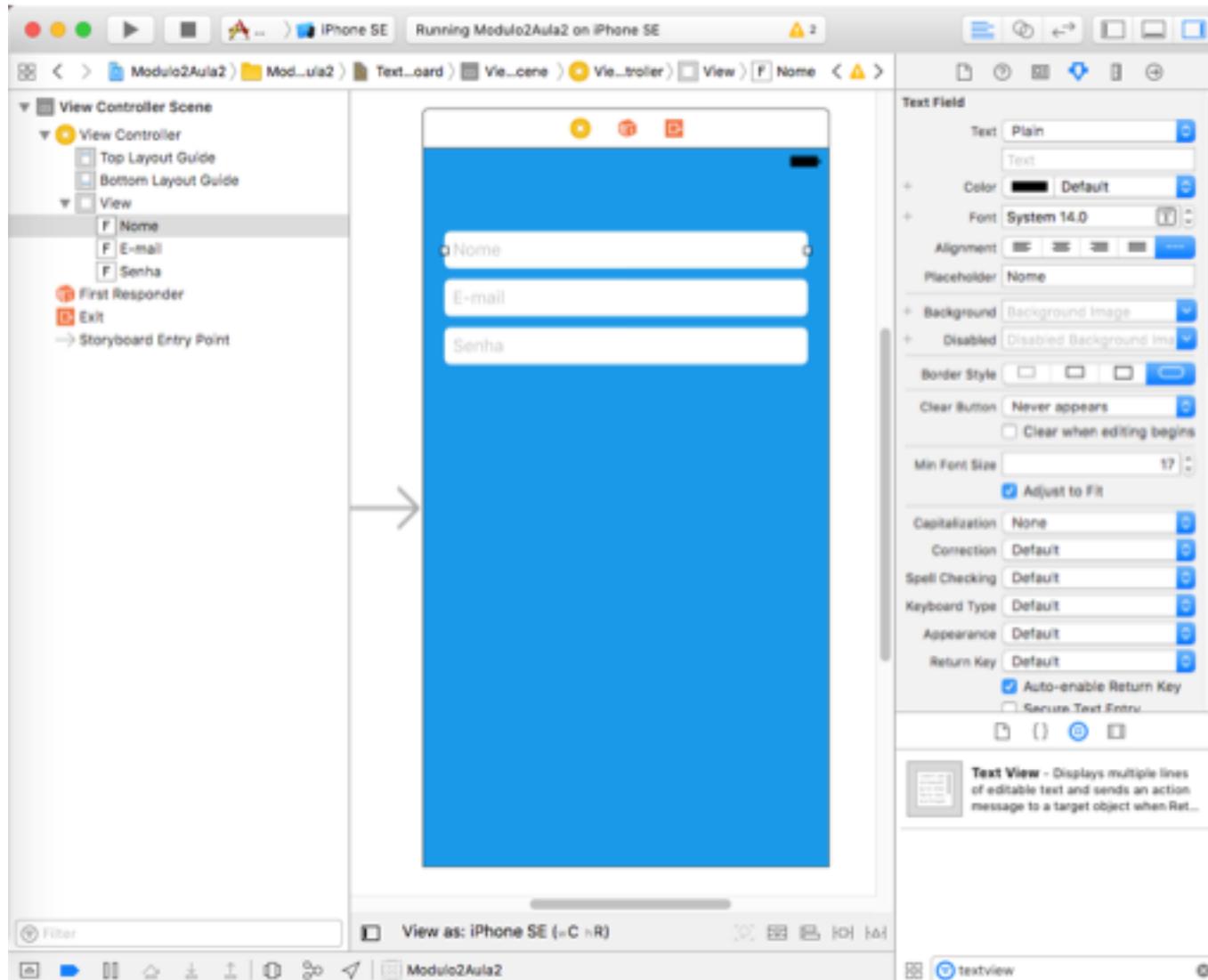
A Linguagem Swift e suas particularidades

25. UITextField, UITextView



A Linguagem Swift e suas particularidades

25. UITextField, UITextView



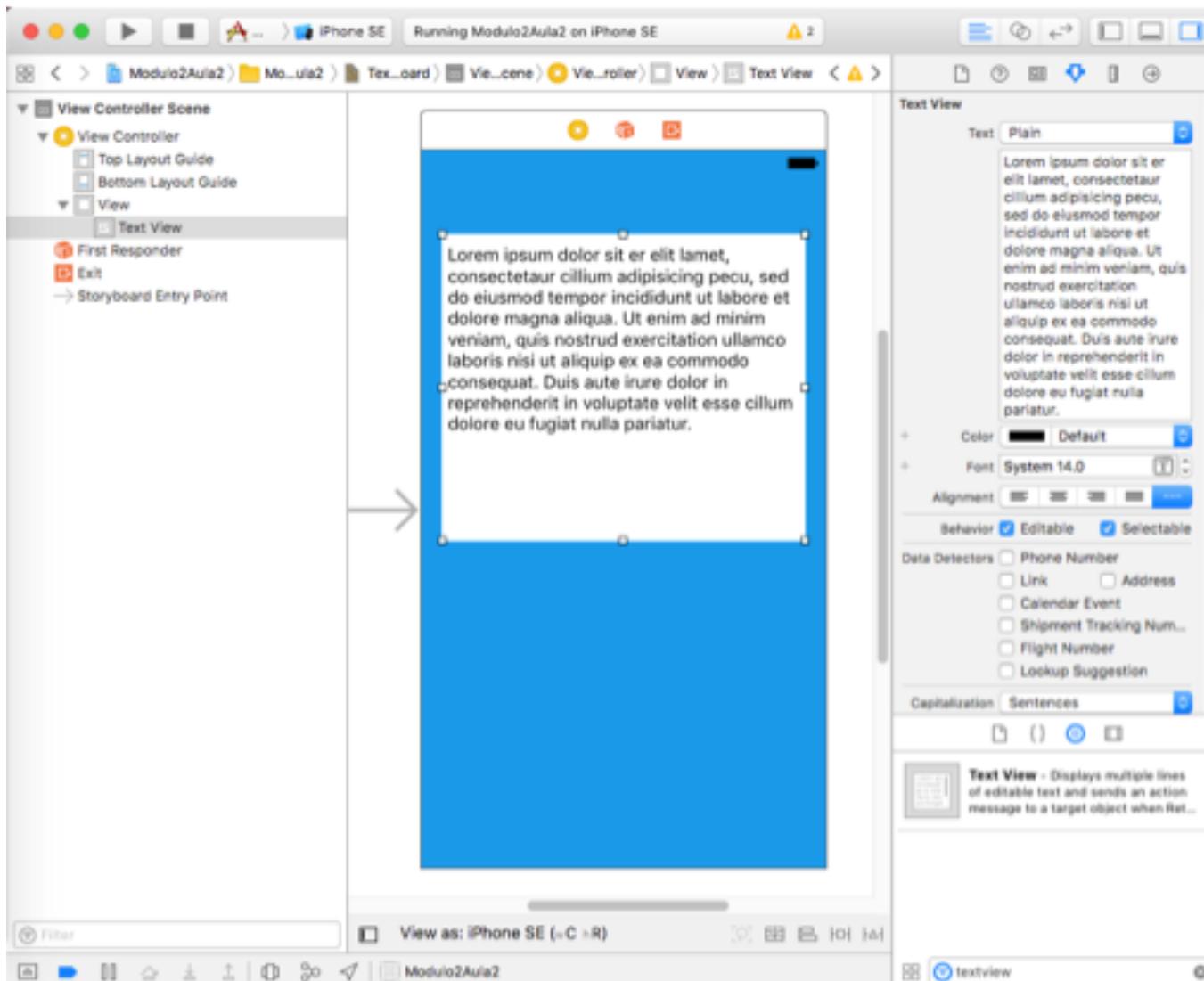
25. UITextField, UITextView

UITextView

- Apresenta ao usuário uma região de texto que aceita múltiplas linhas e pode ser rolável (possui scroll).
- Propriedades semelhantes às do UITextField
- Também pode aceitar textos plain e attributed
- Detecta tipos de dados. Ex.: Números de telefone, links, endereços, etc.

A Linguagem Swift e suas particularidades

25. UITextField, UITextView



A Linguagem Swift e suas particularidades

26. Imagens

Asset Catalogs

- Podemos definir todas as imagens que serão utilizadas em nosso app (incluindo o ícone do App) no arquivo Images.xcassets
- Este catálogo de imagens utiliza um padrão de dimensionamento que define qual a imagem será utilizada de acordo com o device em uso
- Podemos definir imagens distintas para iPhone, iPad, Apple Watch, etc, ou usar imagens universais (utilizadas em todos tipos de devices)
- Sufixos são usados para representar em qual device (baseado na sua resolução) a imagem será utilizada.
- 1x (nome-imagem.png): Usado em devices com resolução normal (iPhones até 3GS, iPad 1 e 2)
- 2x (nome-imagem@2x.png): Devices com tela retina (iPhone 4+, iPad 3+)
- 3x (nome-imagem@3x.png): iPhones na versão Plus e iPhone X
- 1x, 2x e 3x também representam o tamanho que a imagem deverá ter

A Linguagem Swift e suas particularidades

26. Imagens



Tamanho 1x: **60x60**



Tamanho 2x: **120x120**



Tamanho 3x: **180x180**

A Linguagem Swift e suas particularidades

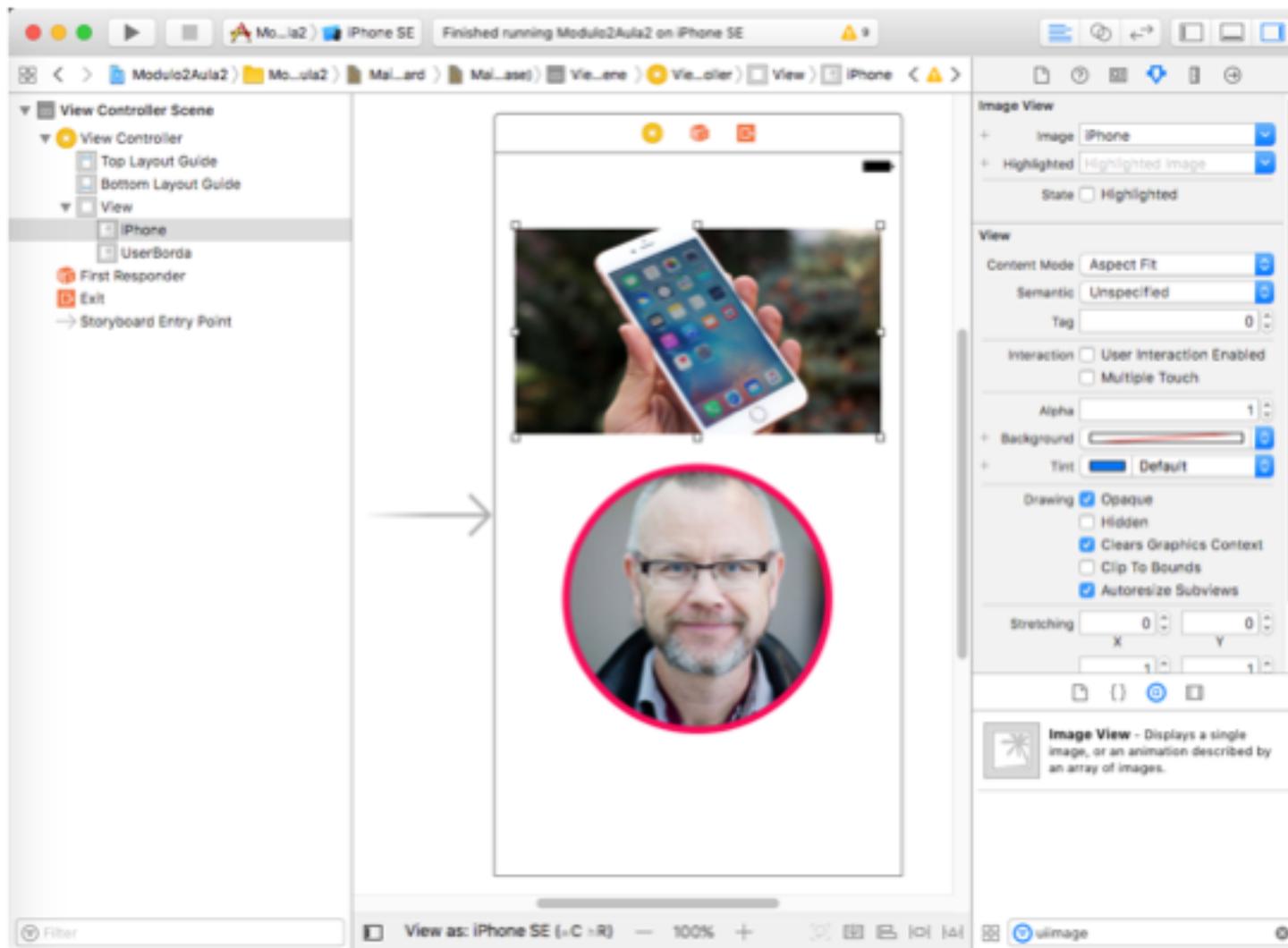
26. Imagens

UIImageView

- UIImageView é a view responsável por apresentar imagens no App (simples ou animadas). A imagem associada a esta view fica armazenada na propriedade `image`, que é do tipo `UIImage`
- Principais atributos:
 - **Image**: Define a imagem que será usada
 - **Content Mode**: Controla a maneira como a imagem se adequa ao container. Destacam-se:
 - **Scale to Fill**: Imagem se adequa ao container, distorcendo se necessário
 - **Aspect Fit**: Mantém o aspecto original da imagem, encaixando-a ao container de modo que apareça toda a imagem
 - **Aspect Fill**: A imagem preenche todo o container, mas mantém seu aspecto

A Linguagem Swift e suas particularidades

26. Imagens



A Linguagem Swift e suas particularidades

26. Imagens

UIImage

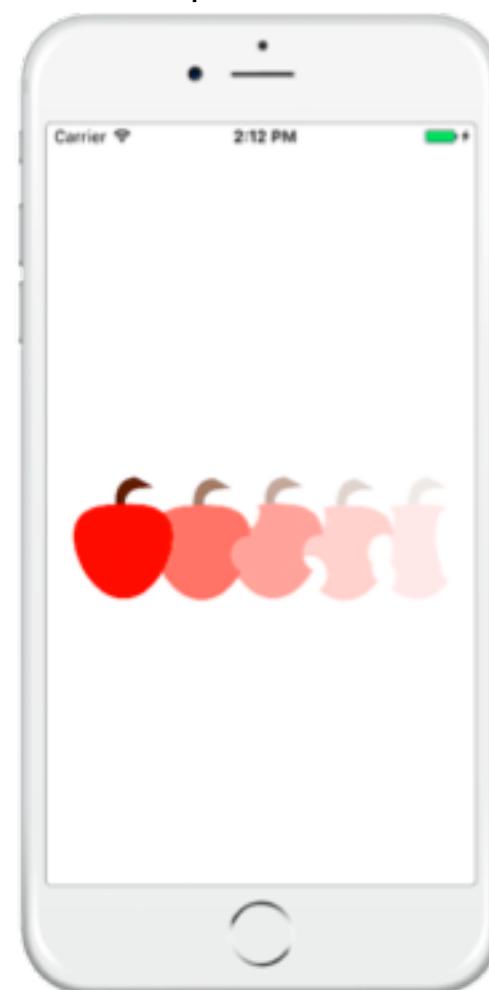
- Classe usada para gerenciar dados de imagem no aplicativo

```
@IBOutlet weak var ivFIAP: UIImageView!
@IBOutlet weak var ivAnimatedFruit: UIImageView!

override func viewDidLoad() {
    super.viewDidLoad()

    //Imagem estática
    let image = UIImage(named: "AnimatedFruit")
    ivFIAP.image = image

    //Imagem animada
    var sequence:[UIImage] = []
    for index in 1...9 {
        if let image = UIImage(named: "\(index)") {
            sequence.append(image)
        }
    }
    //Definindo images
    ivAnimatedFruit.animationImages = sequence
    //Ajustando a repetição (0 = loop)
    ivAnimatedFruit.animationRepeatCount = 0
    //2 segundos de duração
    ivAnimatedFruit.animationDuration = 2
    //Iniciando animação
    ivAnimatedFruit.startAnimating()
```



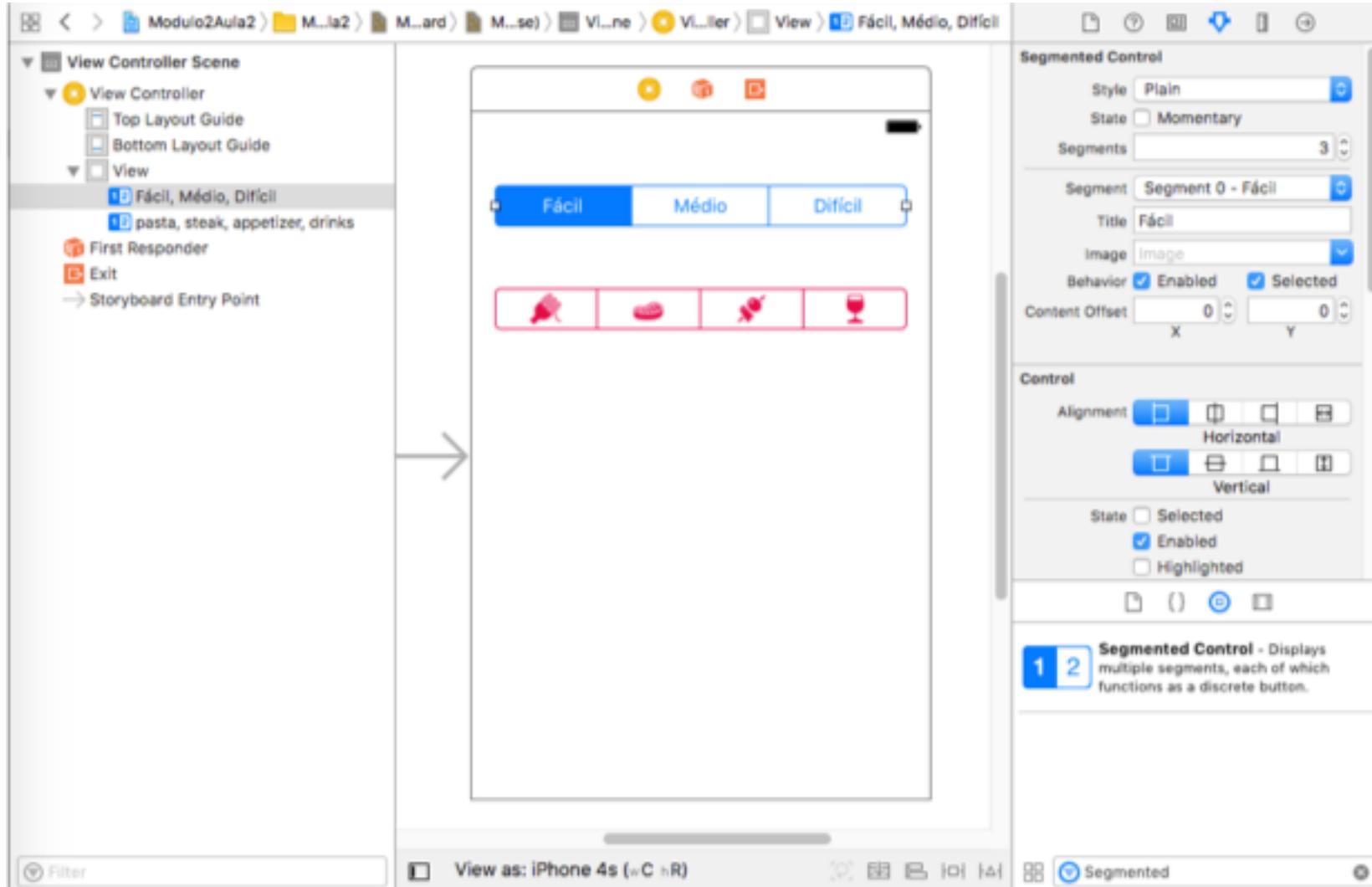
27. UISegmentedControl, UISlider

UISegmentedControl

- Controle horizontal feito de múltiplos segmentos, usado para limitar o usuário a escolher apenas uma opção
- Principais atributos:
 - **State**: Se habilitado (Momentary), a seleção não persiste
 - **Segments**: Define o total de segmentos
 - **Segment**: Controla as propriedades do segmento escolhido:
 - **Title**: Título do segmento
 - **Image**: Imagem do segmento
 - **Enabled / Selected**: Definem se o segmento está habilitado e/ou selecionado

A Linguagem Swift e suas particularidades

27. UISegmentedControl, UISlider



A Linguagem Swift e suas particularidades

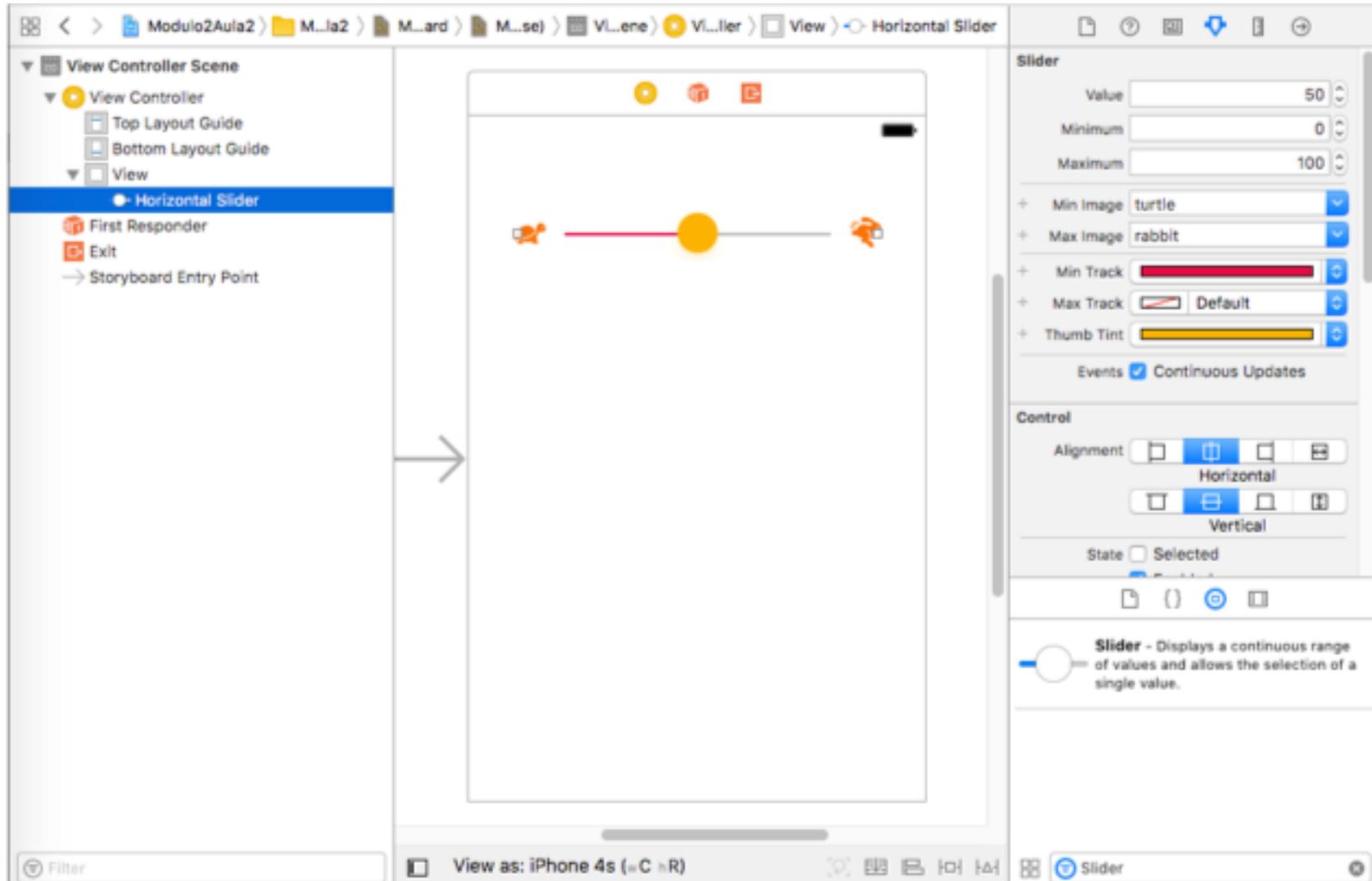
27. UISegmentedControl, UISlider

UISlider

- Controle horizontal que permite ao usuário escolher um valor dentre um range de valores
- Principais atributos:
 - **Value**: Valor atual slider
 - **Minimum/Maximum**: Define os valores mínimos e máximos do slider
 - **Min/Mag Image**: Imagens que serão usadas nas extremidades do slider
 - **Min/Max Track**: Cores usadas na barra do slider
 - **Thumb Tint**: Cor usada no thumb
 - **Events**: Se selecionado (Continuous Updates) dispara o evento do slider continuamente, sempre que o usuário altera o mesmo. Se desabilitado, o evento só é disparado quando o usuário remover o dedo

A Linguagem Swift e suas particularidades

27. UISegmentedControl, UISlider



■ Bibliografia Básica

- Apple. The Swift Programming Language (Swift 4.1). 2018

https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/index.html

Bibliografia Complementar

- MATHIAS, Matthew; GALLAGHE, John. Swift Programming: The Big Nerd Ranch Guide (2nd Edition) (Big Nerd Ranch Guides). Big Nerd Ranch. 2016.