



PROF. RENÊ XAVIER

DESENVOLVIMENTO PARA IOS 11 COM SWIFT 4

ONDE ENCONTRAR O MATERIAL?

[HTTPS://GITHUB.COM/RENEFX/IOS-2018-01](https://github.com/RENEFX/IOS-2018-01)

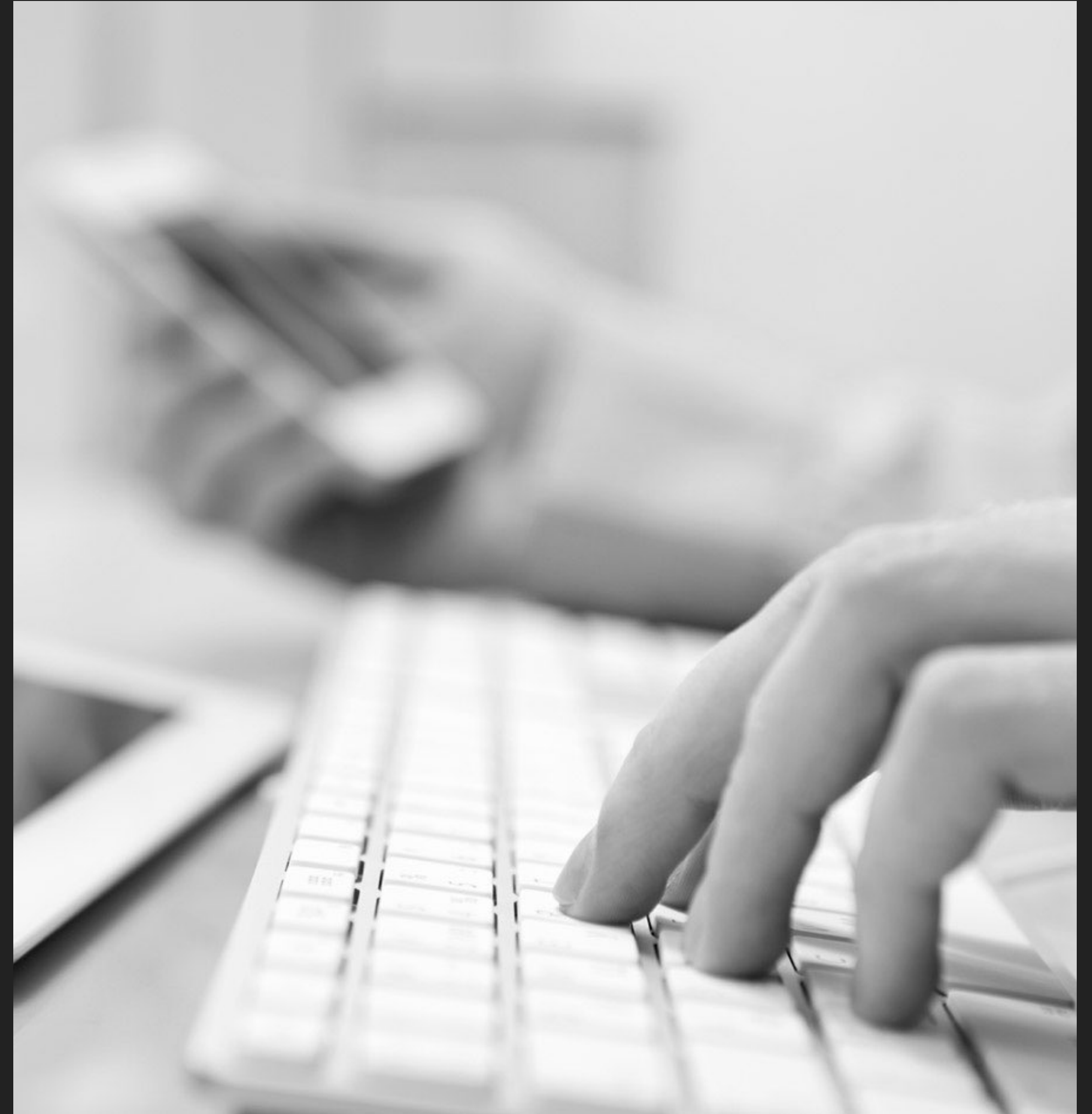
GITHUB

ALÉM DO TRADICIONAL BLACKBOARD DO IESB

O QUE VAMOS FAZER HOJE?

AGENDA

- ▶ Componentes Visuais interessantes





COMPONENTES VISUAIS

COMPONENTES VISUAIS

- ▶ Muito usados para criar tutoriais
- ▶ Page Control e Page View Controller funcionam na mesma idéia que a TableView e a TableViewController



Page View Controller - Presents a sequence of view controllers as pages.



Page Control - Displays a dot for each open page in an application and supports sequential navigation throu...

COMPONENTES VISUAIS

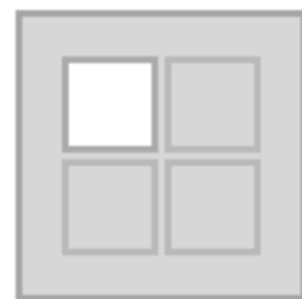
- ▶ Muito comum quando se tem "cards"
- ▶ CollectionView e CollectionViewController funcionam na mesma idéia que a TableView e a TableViewController



Collection View Controller - A controller that manages a collection view.



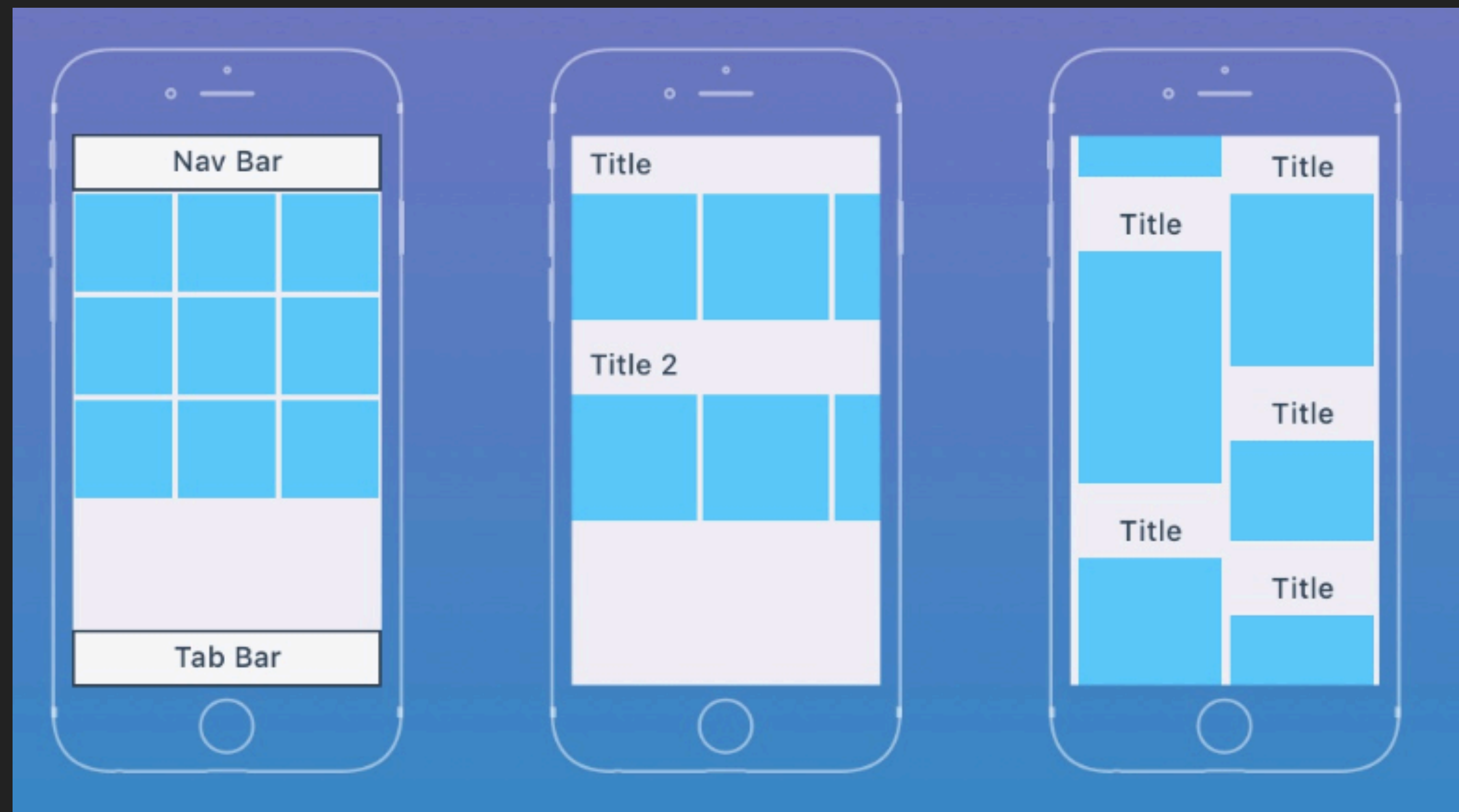
Collection View - Displays data in a collection of cells.



Collection View Cell - Defines the attributes and behavior of cells in a collection view.



Collection Reusable View - Defines the attributes and behavior of reusable views in a collection view, s...



COMPONENTES VISUAIS



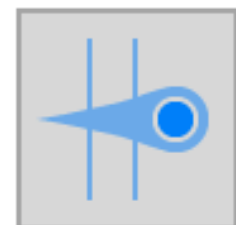
Tap Gesture Recognizer - Recognizes tap gestures, including double-tap or multiple-touch.



Pinch Gesture Recognizer - Recognizes pinch gestures.



Rotation Gesture Recognizer - Recognizes rotation gestures.



Swipe Gesture Recognizer - Recognizes swipe gestures.



Pan Gesture Recognizer - Recognizes pan (dragging) gestures.



Screen Edge Pan Gesture Recognizer - Recognizes pan (dragging) gestures that start near a...



Long Press Gesture Recognizer - Recognizes long press gestures, based on the number and duration of...



Custom Gesture Recognizer - Recognizes custom gestures. Set a custom subclass in the Identity inspe...

- ▶ Se lembra quando usávamos um botão em cima de uma View para clicar?
- ▶ A melhor forma é usar um Gesture Recognizer na View
- ▶ Eles identificam quando um gesto é feito e podemos criar uma função para responder a isso

```
override fun viewDidLoad() {  
    super.viewDidLoad()  
  
    let gestureRecognizer = UITapGestureRecognizer(target: self, action: "handleTap:")  
    self.view.addGestureRecognizer(gestureRecognizer)  
}  
  
func handleTap(gestureRecognizer: UIGestureRecognizer) {
```

COMPONENTES VISUAIS

- ▶ Para empilhar Views de uma forma fácil sem ter que se preocupar muito com as constraints de dentro, somente com as constraints da Stack



Horizontal Stack View - Arranges views linearly.



Vertical Stack View - Arranges views linearly.

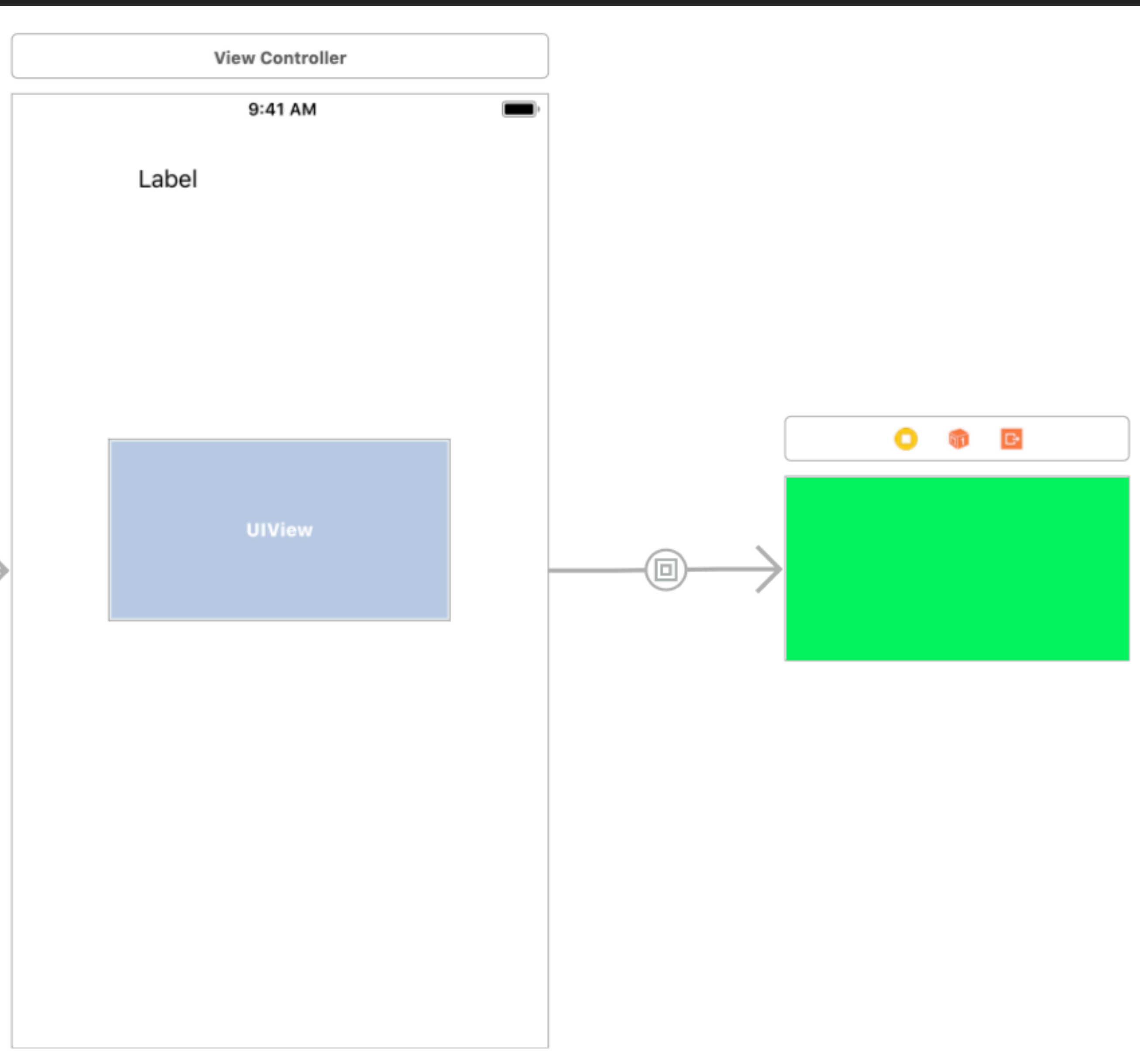
COMPONENTES VISUAIS

- ▶ Quando temos muitas coisas na tela que poderiam ser componentizadas, usamos a `ContainerView`
- ▶ Ela é uma forma de colocar uma `ViewController` (de qualquer tipo que for) dentro de uma `ViewController` e as duas aparecerem ao mesmo tempo

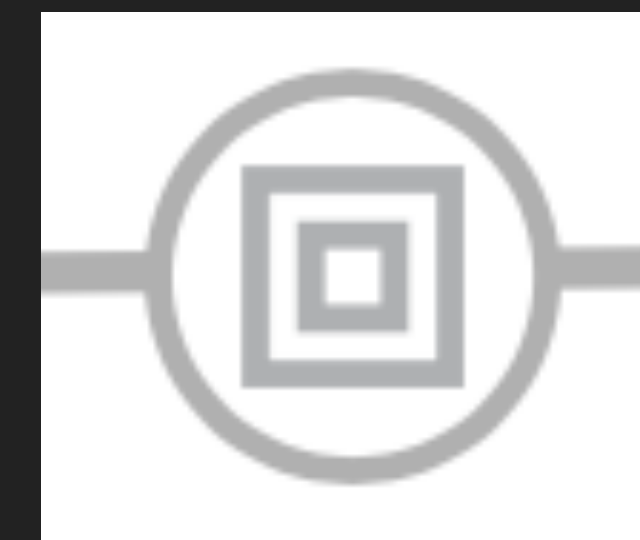


Container View - Defines a region of a view controller that can include a child view controller.

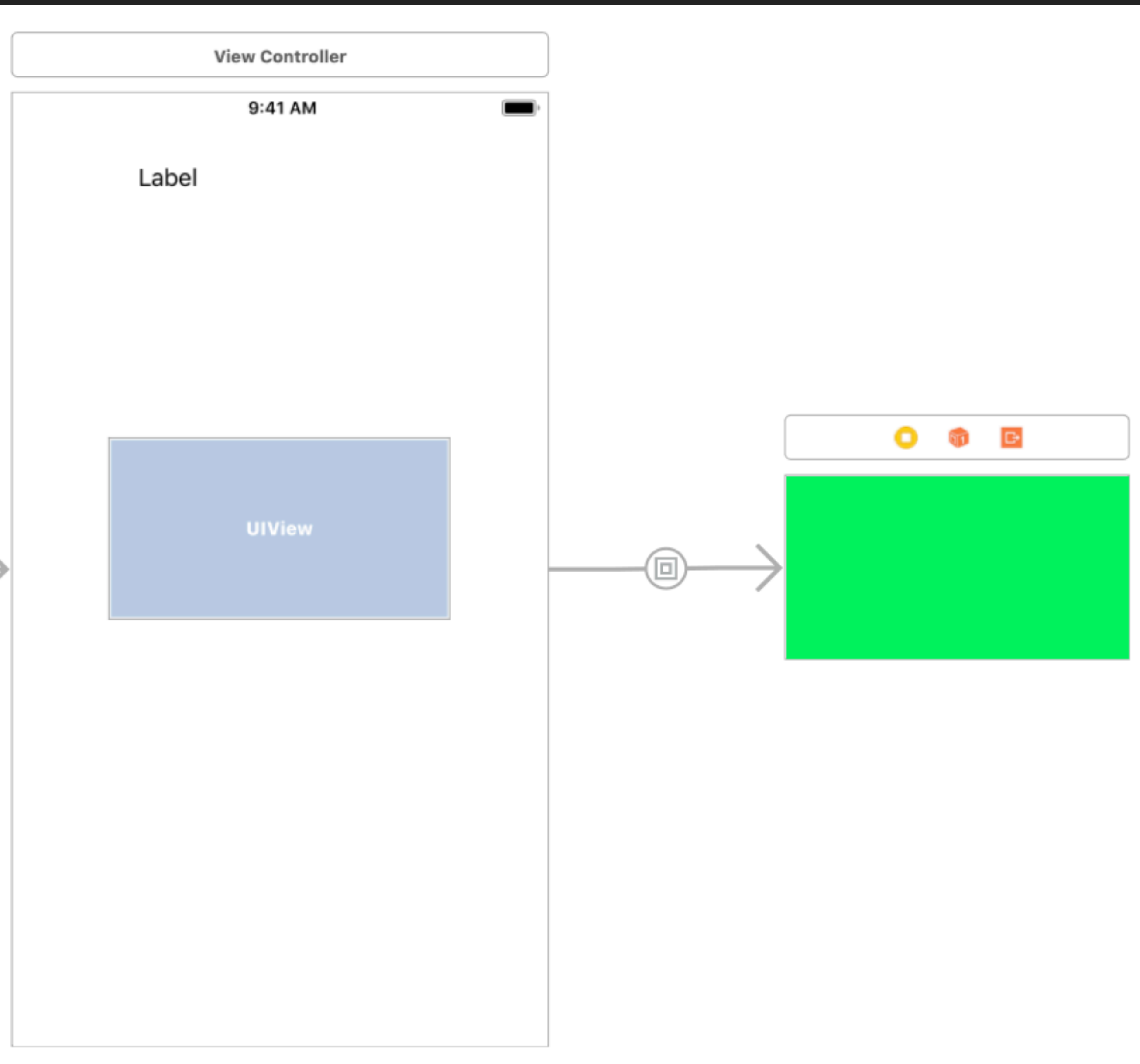
COMPONENTES VISUAIS



- ▶ Nesse caso, a tela não faz uma Segue para a ViewController verde para a ViewController verde
- ▶ A ViewController verde está dentro da outra ViewController
- ▶ Note que a ViewController verde tem o mesmo tamanho que a View
- ▶ Além disso, o ícone da Segue é diferente. Ele representa uma ContainerView

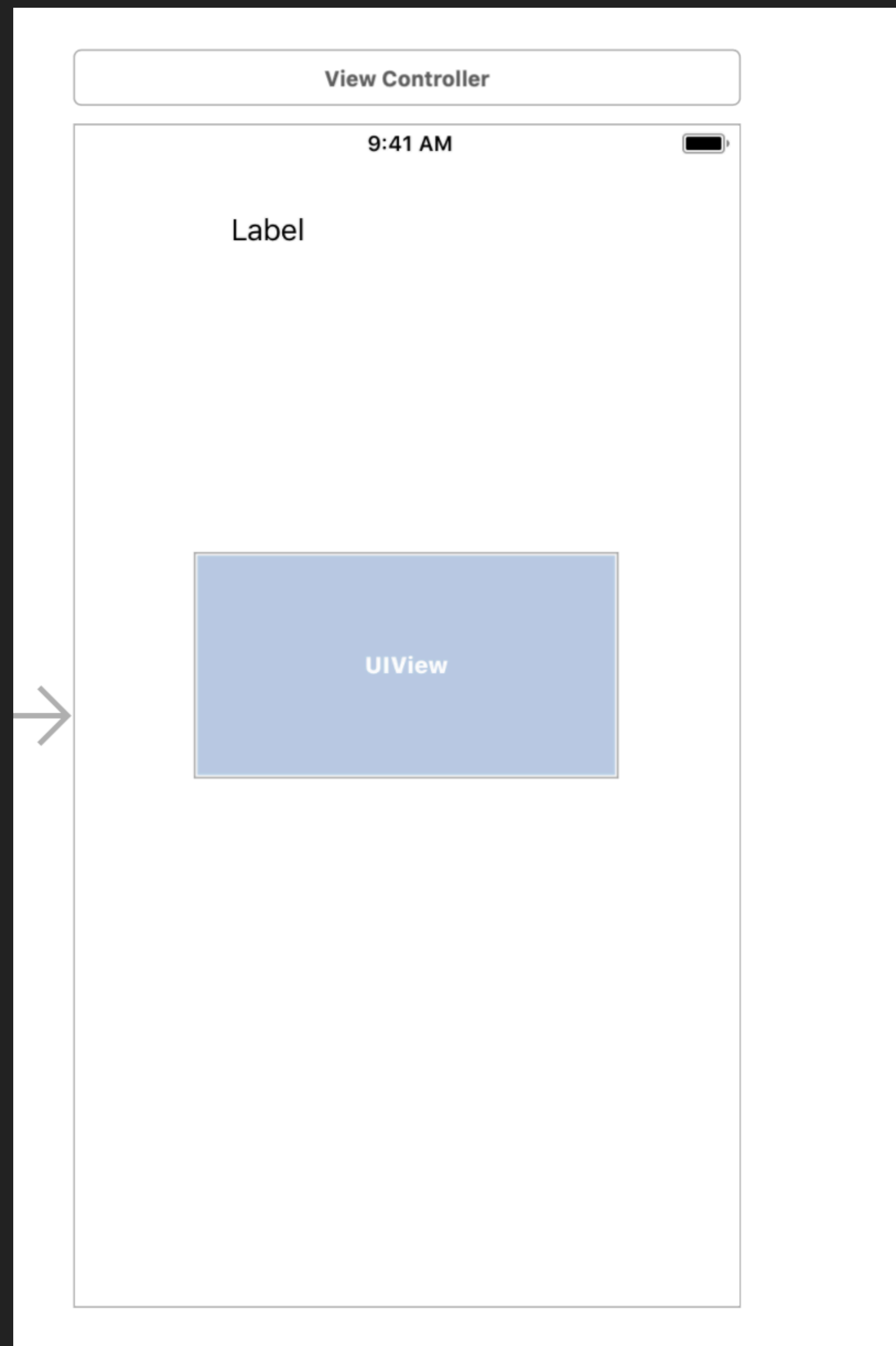


COMPONENTES VISUAIS



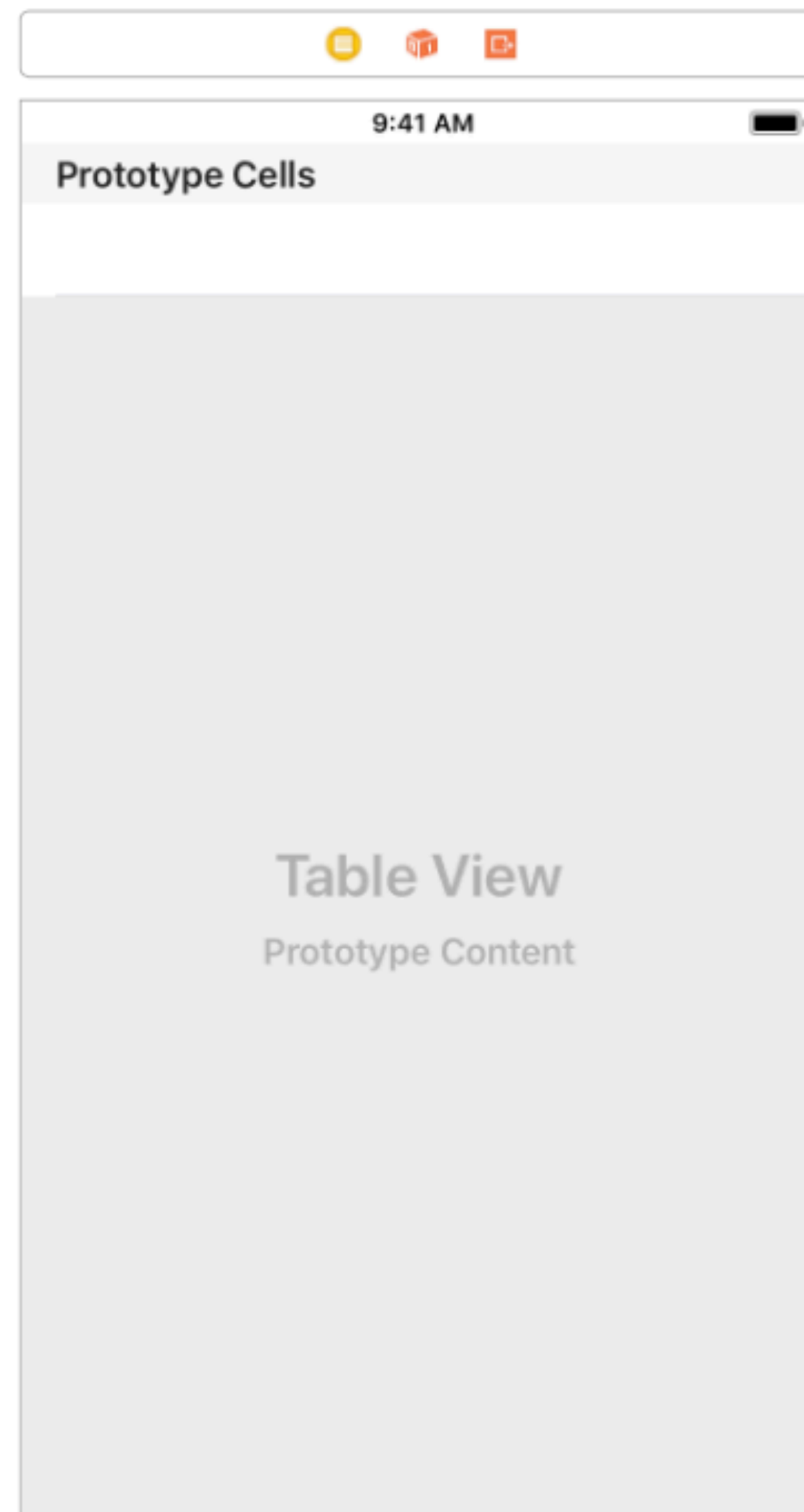
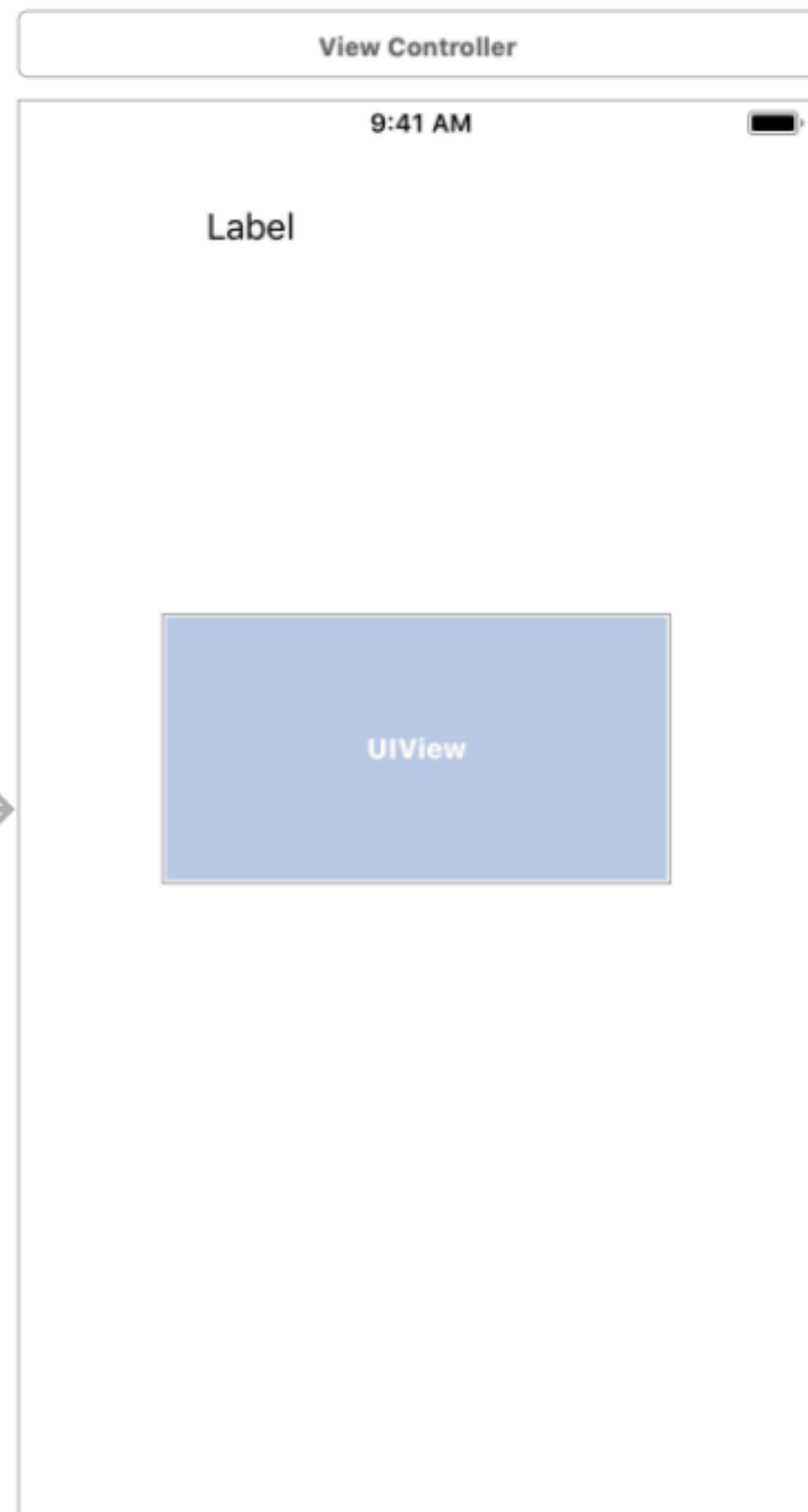
- ▶ Caso você queira colocar uma TableViewController (ou qualquer uma que não seja uma UIViewController)
- ▶ Você pode selecionar a Segue e clicar em delete
- ▶ Apagar a ViewController verde, só mantendo a UIView

COMPONENTES VISUAIS



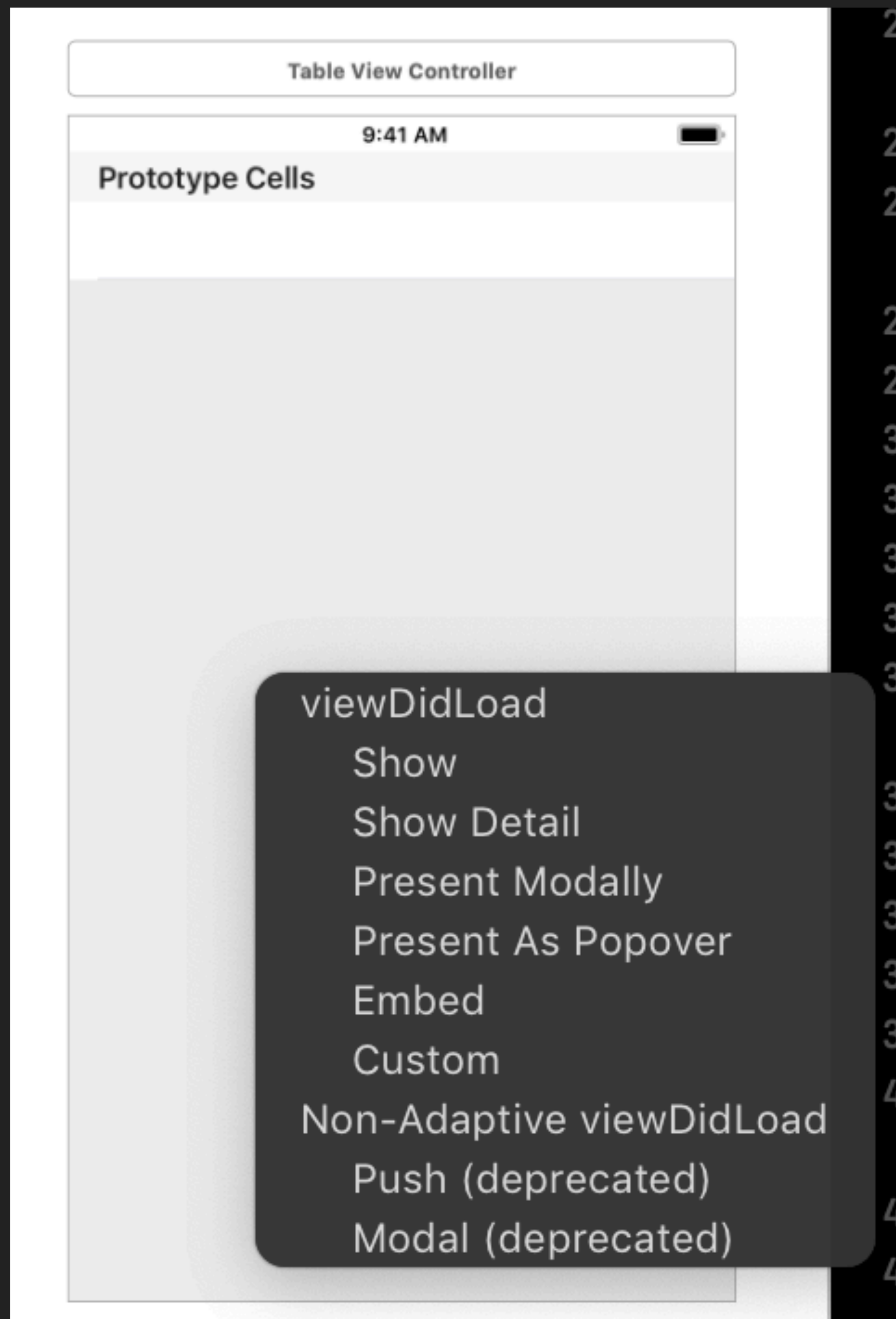
- ▶ Caso você queira colocar uma UITableViewController (ou qualquer uma que não seja uma UIViewController)
- ▶ Você pode selecionar a Segue e clicar em delete
- ▶ Apagar a ViewController verde, só mantendo a UIView
- ▶ Coloque a UITableViewController

COMPONENTES VISUAIS



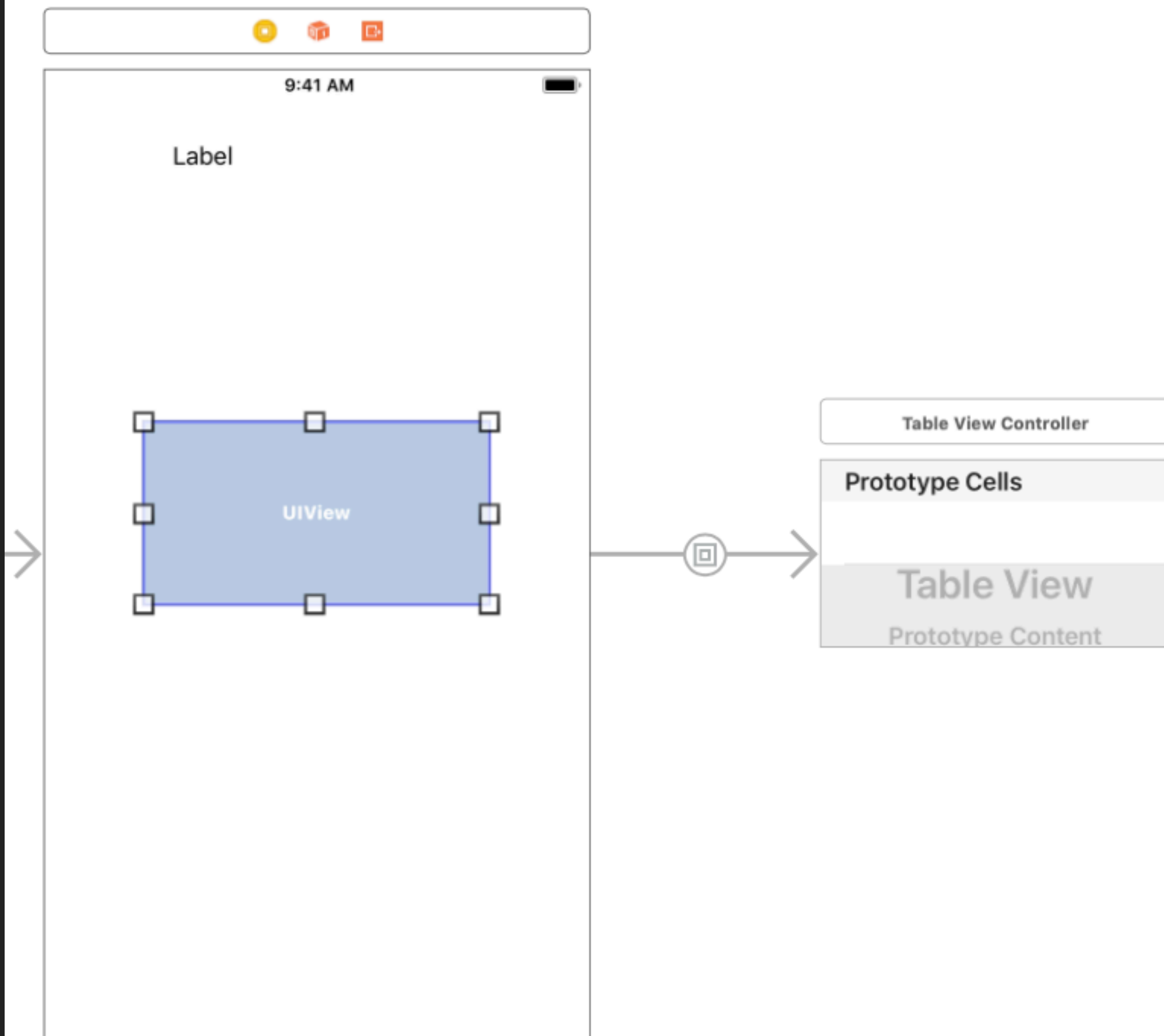
- ▶ Caso você queira colocar uma `TableViewController` (ou qualquer uma que não seja uma `UIViewController`)
- ▶ Você pode selecionar a Segue e clicar em delete
- ▶ Apagar a `ViewController` verde, só mantendo a `UIView`
- ▶ Coloque a `TableViewController`
- ▶ Use o Clica + Control + Arrasta da `UIView` até a `Table View Controller`

COMPONENTES VISUAIS



► Selecione a opção Embed

COMPONENTES VISUAIS



► Selecione a opção Embed

► Pronto!

COMPONENTES VISUAIS

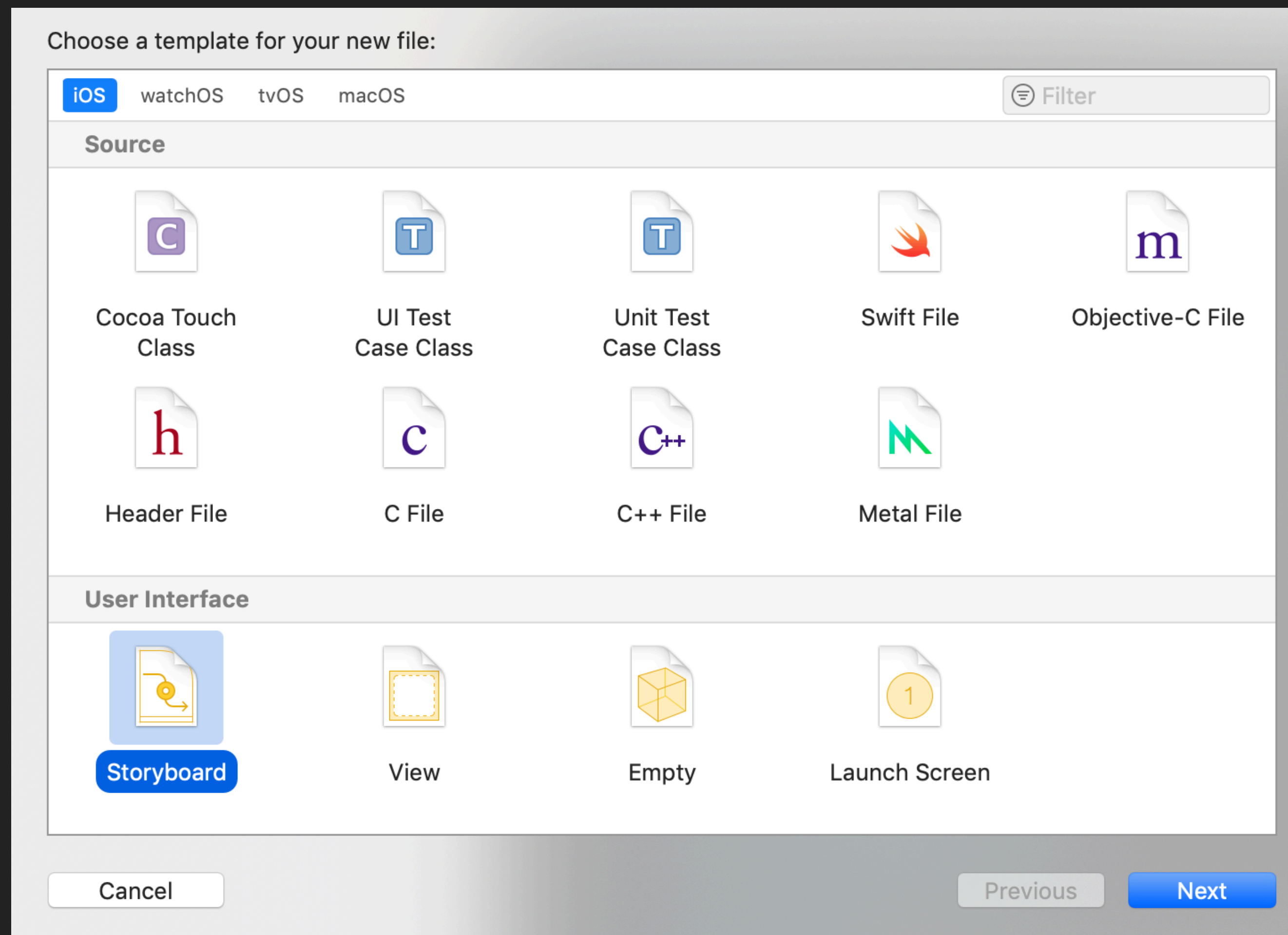
- ▶ Storyboard Reference é basicamente uma referência para outro Storyboard
- ▶ Assim, seu Storyboard fica mais organizado
- ▶ Você pode fazer então uma segue para uma ViewController específica de outro Storyboard



Storyboard Reference - Provides a placeholder for a view controller in an external storyboard.

COMPONENTES VISUAIS

- ▶ Para criar um novo Storyboard é só clicar em Cmd+N e selecionar Storyboard



COMPONENTES VISUAIS

- ▶ Você pode fazer então uma segue para uma ViewController específica de outro Storyboard



The screenshot shows the 'Storyboard Reference' configuration panel in Xcode. The panel has a title bar with icons for file, help, storyboard, segue, and navigation. The main area is titled 'Storyboard Reference' and contains three fields:

Property	Value
Storyboard	Main
Referenced ID	Initial View Controller
Bundle	com.iesb.TesteRealm1

COMPONENTES VISUAIS

- ▶ Como colocar uma imagem atrás de uma tableView
- ▶ Lembrar de colocar o background de cada célula como clear color

```
let imageBackground = UIImageView(image: UIImage(named: "background"))
imageBackground.contentMode = .scaleAspectFill
imageBackground.frame = self.tableView.frame

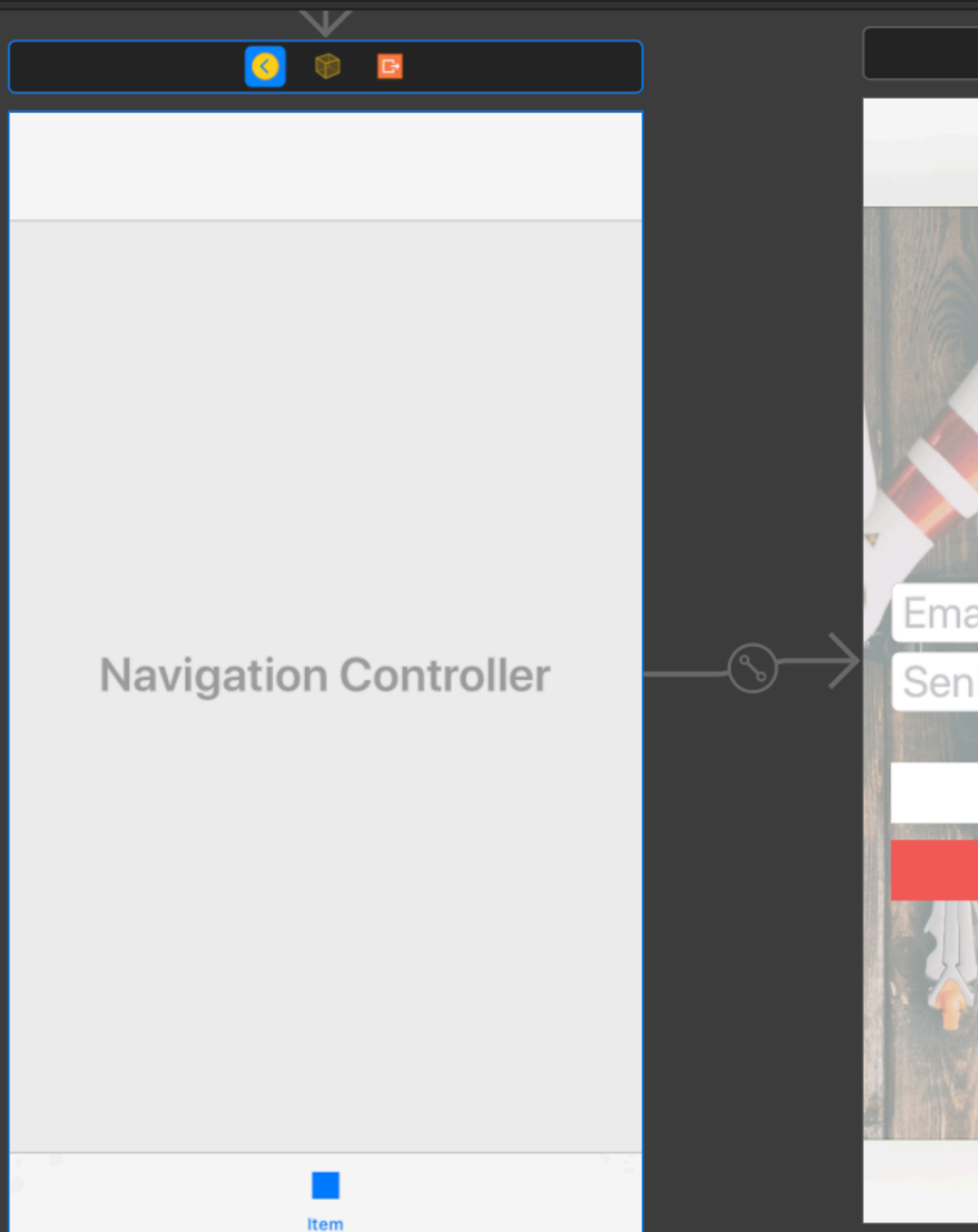
self.tableView.backgroundColor = imageBackground
```

COMPONENTES VISUAIS

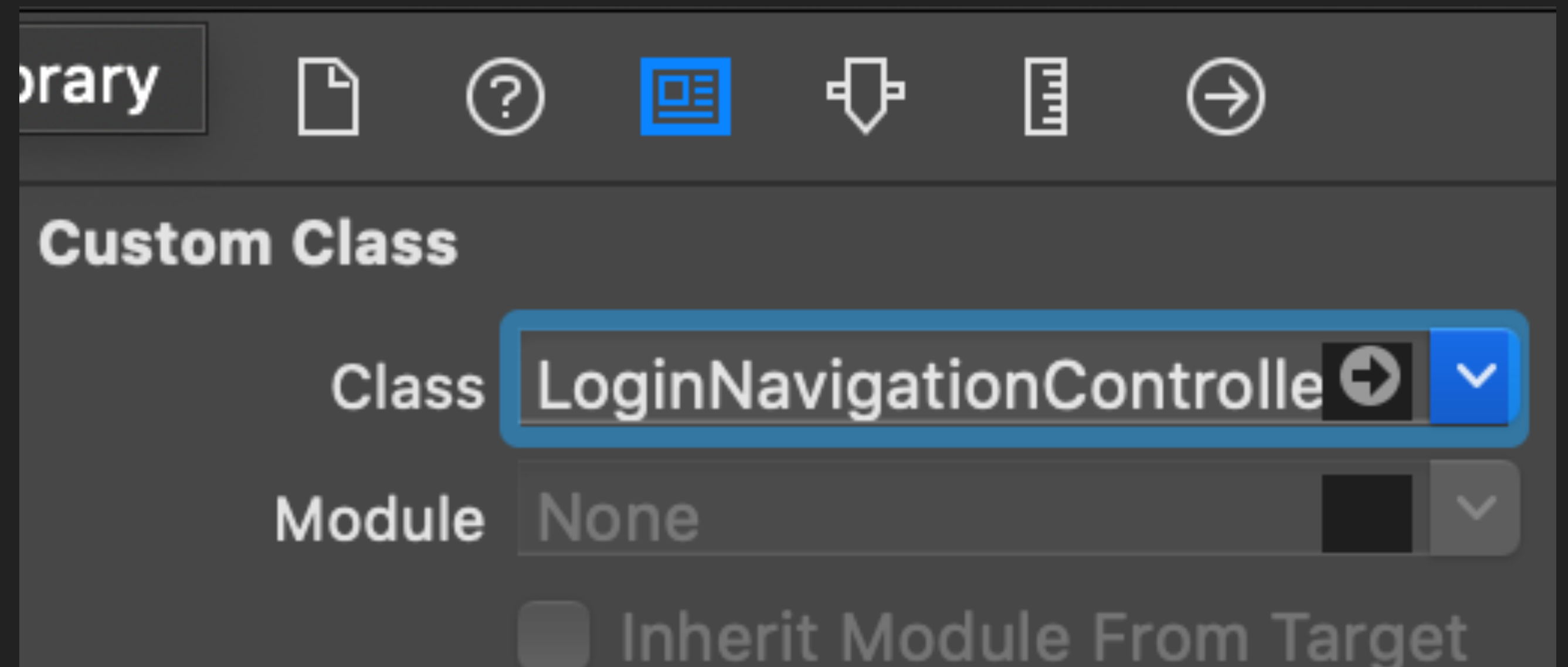
- ▶ Como determinar qual viewcontroller uma navigation controller irá carregar
- ▶ É possível criar uma classe para qualquer componente visual do Storyboard, então criamos um que controle a Navigation e determine qual ViewController será exibida quando a Navigation for aparecer

```
class LoginNavigationViewController: UINavigationController {  
  
    override func viewWillAppear(_ animated: Bool) {  
        // buscar isLogged no BD  
        var isLogged = false  
        let identifier = isLogged ? "telaVerdeId" : "tableviewControllerId"  
        if let vc = storyboard?.instantiateViewController(withIdentifier: identifier) {  
            viewControllers = [vc]  
        }  
    }  
}
```


COMPONENTES VISUAIS



- ▶ Selecionamos então a NavigationController que queremos e então colocamos nossa classe para controlar ela

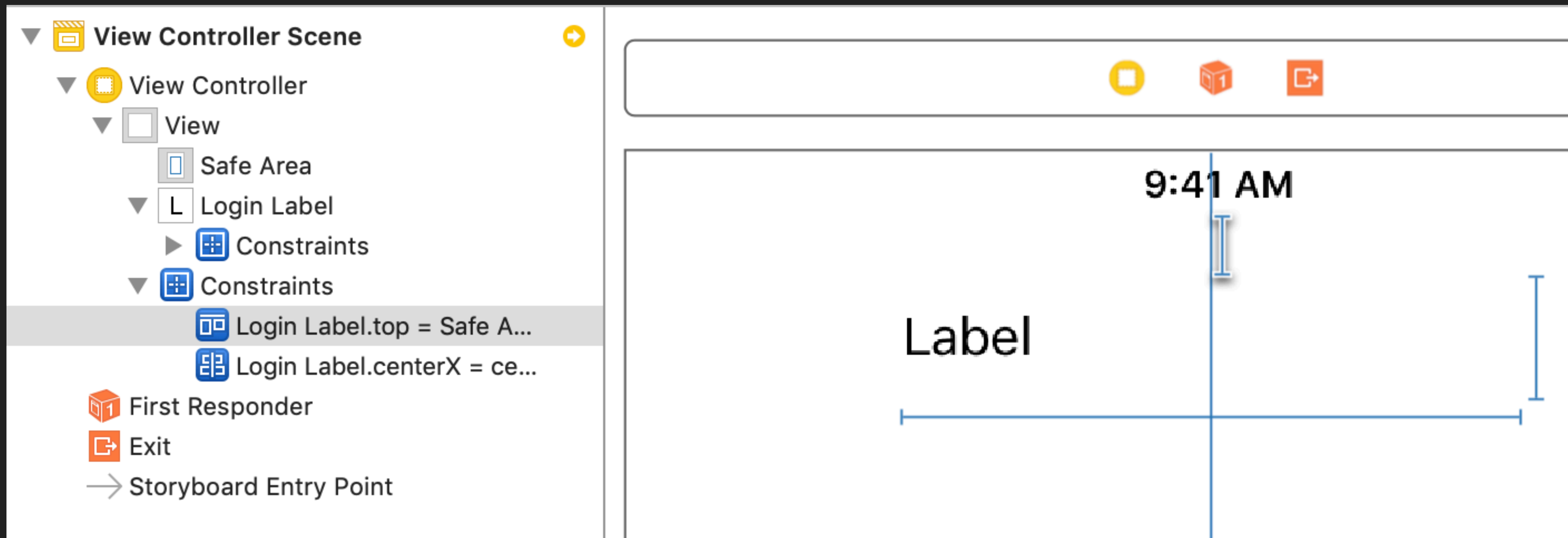




ANIMAÇÕES

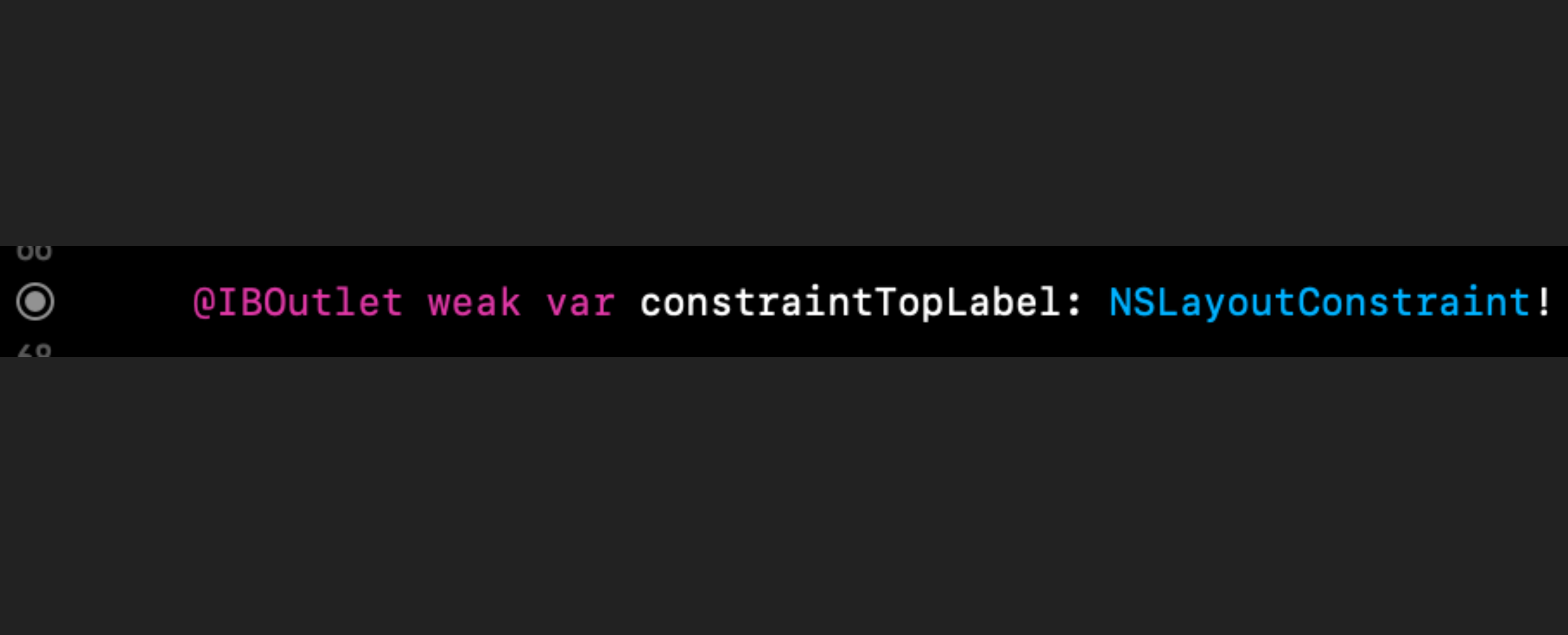
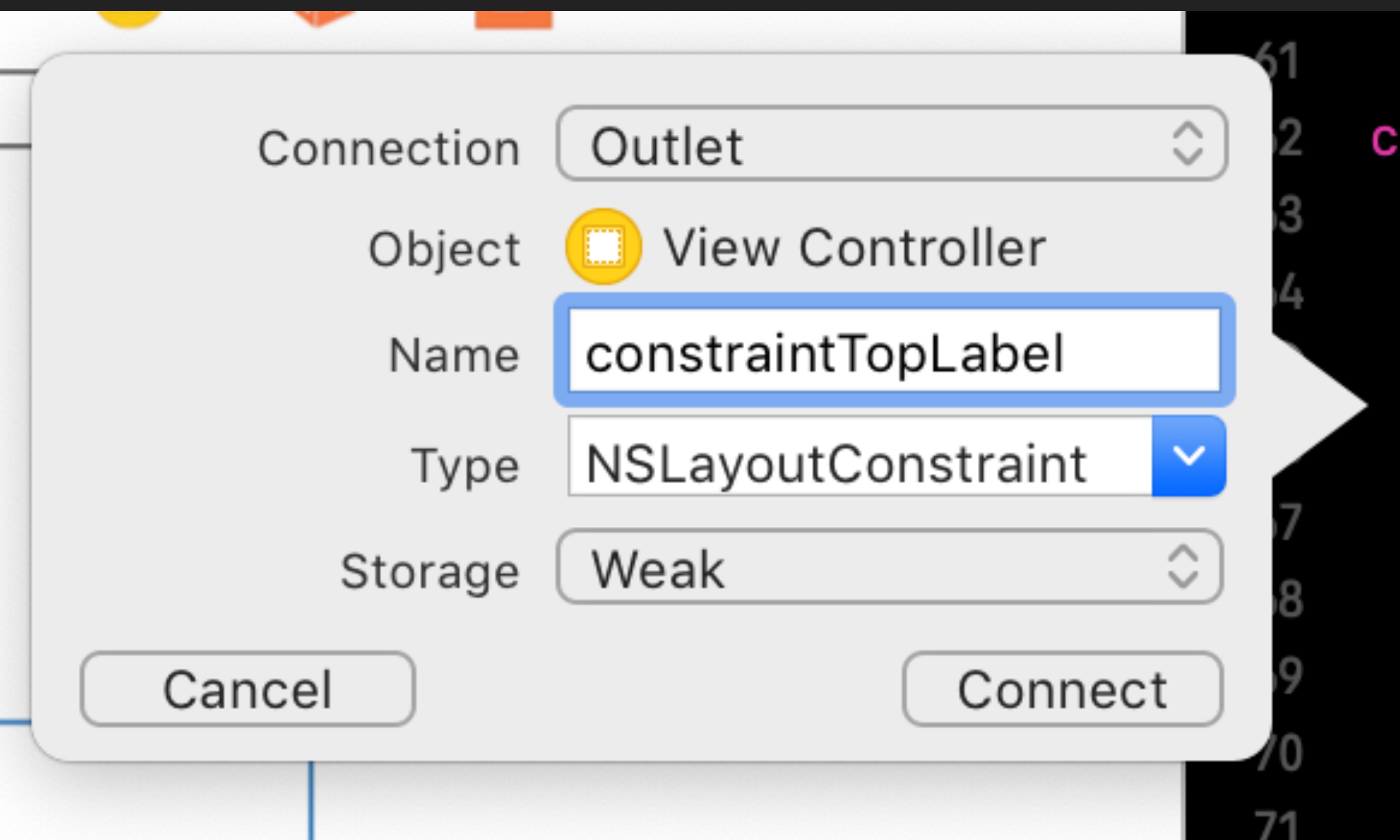
ANIMAÇÕES

- ▶ Para entender animações, temos que entender muito bem de constraints
- ▶ Temos que entender também que assim como fazemos IBOutlets com UIViews também podemos fazer referências de Constraints no nosso código
- ▶ Isso é feito clicando na hierarquia de Views, encontrando a Constraint e usando o Clica Control e arrasta



ANIMAÇÕES

- ▶ Isso é feito clicando na hierarquia de Views, encontrando a Constraint e usando o Clica Control e arrasta
- ▶ Com isso temos um IBOutlet da nossa Constraint



ANIMAÇÕES

- ▶ Isso é feito clicando na hierarquia de Views, encontrando a Constraint e usando o Clica Control e arrasta
- ▶ Com isso temos um IBOutlet da nossa Constraint
- ▶ Conseguimos alterar o valor da nossa Constraint atribuindo um novo valor a property constant
- ▶ Porém isso não está animado

```
override func viewDidLoad() {  
    super.viewDidLoad()  
    self.constraintTopLabel.constant = 300  
}
```

ANIMAÇÕES

- ▶ Para animar a execução usamos o método de classe da UIView
- ▶ Devemos informar o tempo que essa animação irá demorar executando
- ▶ Atribuimos um novo valor e dentro do animate executamos a atualização do layout da nossa View

```
self.constraintTopLabel.constant = 300
UIView.animate(withDuration: 0.5) {
    self.view.layoutIfNeeded()
}
```

ANIMAÇÕES

- ▶ Ainda não está animando, pois o ViewDidLoad não é local para a animação
- ▶ Ao colocar no viewDidLoadAppear, a View é animada

```
override func viewDidLoadAppear(_ animated: Bool) {  
    super.viewDidLoadAppear(animated)  
  
    self.constraintTopLabel.constant = 300  
    UIView.animate(withDuration: 0.5) {  
        self.view.layoutIfNeeded()  
    }  
}
```

ANIMAÇÕES

- ▶ Temos outra opção de animação se quisermos um delay, repetição ou outras opções.

```
UIView.animate(withDuration: 2.5, delay: 2.4, options: [.curveEaseIn, .repeat], animations: {  
    self.view.layoutIfNeeded()  
}, completion: nil)
```


ANIMAÇÕES

- ▶ Essa forma de animação por Constraints é interessante, mas é possível animar a View e a posição dela de outra forma, porém o Auto-Layout não é garantido

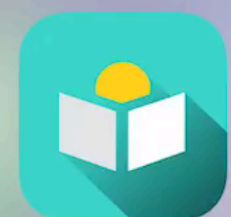
```
UIView.animate(withDuration: 2.5, delay: 2.4, options: [.curveEaseIn, .repeat], animations: {  
    self.loginLabel.layer.anchorPoint.x = 300  
}, completion: nil)
```

**MAIS SOBRE
ISSO AQUI**

ANIMAÇÕES

- ▶ Outra forma de fazer animações mais complexas é usando o pod Lottie
 - ▶ É um pod para animações, a animação é feita no Adobe After Effects e é exportado um JSON
 - ▶ Esse JSON é adicionado ao seu projeto (um bundle) e então a biblioteca lê esse pod e transforma em animação
 - ▶ A animação é executada em uma View que você configura no Storyboard
- 
- The Lottie logo is a teal square containing the word 'Lottie' in a white, stylized, rounded font. The 'L' is particularly large and has a decorative swirl at its base.
- ▶ Possui para Android e iOS e o mesmo JSON pode ser usado para as duas plataformas
 - ▶ É feito pelo AirBnB

Developer



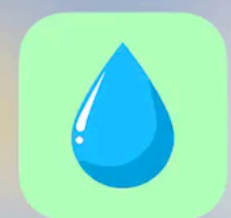
Adote



Cat Run



Investimentos



Racionar DF



BBHost



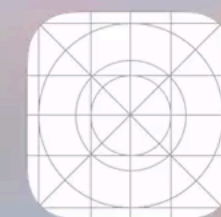
Siscaer



GeoMapa Rural



Beta



GrupoBrasileir...





THREADS

THREADS

- ▶ Já vimos um pouco de Threads na nossa requisição quando pedíamos para atualizar a View após fazer uma request
- ▶ As Threads funcionam por filas de execução
- ▶ Sendo as mais conhecidas a global e a main (mas é possível criar uma própria)
- ▶ A main ocorrem as alterações principais e a global tudo que não é imediato

```
DispatchQueue.global(qos: .userInitiated).async {  
    // Executar processamento  
  
    DispatchQueue.main.async {  
        // Atualizar views  
    }  
}
```


THREADS

- ▶ Conseguimos disparar uma thread global com delay

```
let delay = DispatchTime.now() + .seconds(60)
DispatchQueue.global(qos: .userInitiated).asyncAfter(deadline: delay, execute: {
    // Executar processamento

    DispatchQueue.main.async {
        // Atualizar views
    }
})
```


THREADS

- ▶ Conseguimos disparar uma thread global com delay
- ▶ Uma thread main também
- ▶ Dentro das threads global, temos uma lista de prioridades de execução

```
let delay = DispatchTime.now() + .seconds(60)
DispatchQueue.global(qos: .userInitiated).asyncAfter(deadline: delay, execute: {
    // Executar processamento

    DispatchQueue.main.asyncAfter(deadline: delay) {
        // Atualizar views
    }
})
```

THREADS

- ▶ User-Interactive - Coisas quase instantâneas
- ▶ User-Initiated - Quase instantâneas, que levem no máximo alguns segundos
- ▶ Utility - Operações que demorem alguns segundos ou minutos
- ▶ Background - Algo que leve minutos ou horas
- ▶ Default - Fica entre o User-Initiated e o Utility - Quando não se é informado a classificação, essa é a opção padrão
- ▶ Unspecified - Para certos casos de App com API legada

THREADS

```
override fun viewDidLoad() {  
    super.viewDidLoad()  

```

```
DispatchQueue.global(qos: .|)
```

DispatchQoS.QoSClass background

DispatchQoS.QoSClass default

DispatchQoS.QoSClass unspecified

DispatchQoS.QoSClass userInitiated

DispatchQoS.QoSClass userInteractive

DispatchQoS.QoSClass utility

```
}
```

```
over
```

```
}
```

THREADS

- ▶ Temos falado de threads async, mas é bom lembrar que existem casos que é interessante ter threads Síncronas
- ▶ Para isso existe o sync
- ▶ É usado quando se tem situações que você quer garantir que algo terminou

```
let delay = DispatchTime.now() + .seconds(60)
DispatchQueue.global(qos: .userInitiated).asyncAfter(deadline: delay, execute: {
    // Executar processamento

    DispatchQueue.main.async {
        // Atualizar views
    }
})
```


THREADS

```
func doItAll() {  
    let grupoDeExecucao = DispatchGroup()  
  
    grupoDeExecucao.enter()  
    DispatchQueue.global().async {  
        self.executaAlgoJSON1()  
        grupoDeExecucao.leave()  
    }  
  
    grupoDeExecucao.enter()  
    DispatchQueue.global().async {  
        self.executaAlgoJSON2()  
        grupoDeExecucao.leave()  
    }  
  
    grupoDeExecucao.enter()  
    DispatchQueue.global().async {  
        self.downloadImage()  
        grupoDeExecucao.leave()  
    }  
  
    grupoDeExecucao.notify(queue: .global()) {  
        self.finishFunction()  
    }  
}
```

- ▶ Podemos agregar execuções para serem lançadas juntas
- ▶ Criando grupos de execuções
- ▶ Esses grupos de execuções podem ser síncronos ou assíncronos (como na imagem)



NOTIFICAÇÃO

NOTIFICAÇÃO

- ▶ Existem dois tipos de notificação:

- ▶ Notificação local

- ▶ Não necessitam de internet
 - ▶ Podem ser agendadas
 - ▶ Limite de 64 notificações

- ▶ Notificação push

- ▶ Requer que o usuário esteja conectado à Internet no momento do envio
 - ▶ Não há limite
 - ▶ One Signal - Gratuito





CRASHLYTICS & ANALYTICS

CRASHLYTICS & ANALYTICS

- ▶ Ferramentas que pegam o log do App logo após ele quebrar
- ▶ Ao abrir novamente o App, esses logs são enviados para um servidor e então é exibido a linha de código que o App quebrou
- ▶ Tem para Android, porém não tem para React Native
- ▶ O Fabric também tem uma ferramenta para distribuição de Apps para testes e uma ferramenta para Analytics
- ▶ Gratuito



CRASHLYTICS & ANALYTICS

- ▶ Outra ferramenta com Analytics é o Firebase
- ▶ Ele possui outras aplicações também
- ▶ O Firebase é para funcionar como uma aplicação Serverless
- ▶ É bem interessantes para projetos pessoais, porém pouco usado no mercado de trabalho. Para aplicações pequenas o custo é zero ou baixo, mas para aplicações grande o custo é muito alto.





DICAS

DICAS

- ▶ Se o seu projeto possuir muitos endpoints, considere fazer uma classe de Network com métodos genéricos para GET POST DELETE e você envia para esses métodos somente o endpoint e os parâmetros do Body
 - ▶ O retorno será tratado da mesma forma que a comunicação da View com a Controller, uma closure.
 - ▶ A View chama a Controller que por sua vez chama a Network.
 - ▶ Quando a conexão da Network finalizar, ela irá chamar o handler da Controller e a closure da Controller irá chamar o handler da View
- ▶ Em certas implementações do MVC, ao preencher a Model, a controller não é notificada por uma Closure, mas sim usando o padrão Observer. Dessa forma todas as classes que têm um Observer para aquela classe são notificadas.
- ▶ Senhas devem ser salvas no Keychain

DICAS

- ▶ Centralize as Strings como foi dito
- ▶ Centralize cores como no caso das Strings
- ▶ Caso você passe a usar Extensions para classes mais gerais (Strings, Int..) crie um Swift file Extensions.swift que só irá conter as Extensions
- ▶ Considere sempre criar uma subclasse de UIButton, UIView... que já tenha configurado o padrão de botão do seu projeto. Assim você só determina no Storyboard que aquele botão é da classe ButtonProjetoLojas, assim ele já terá essa configuração (cornerRadius, backgroundColor...)
 - ▶ Tente fazer isso com um IBDesignable
 - ▶ Fique atento, pois o IBDesignable é interessante, porém faz seu Storyboard demorar um pouco mais a carregar