



PROF. RENÊ XAVIER

DESENVOLVIMENTO PARA IOS 11 COM SWIFT 4

ONDE ENCONTRAR O MATERIAL?

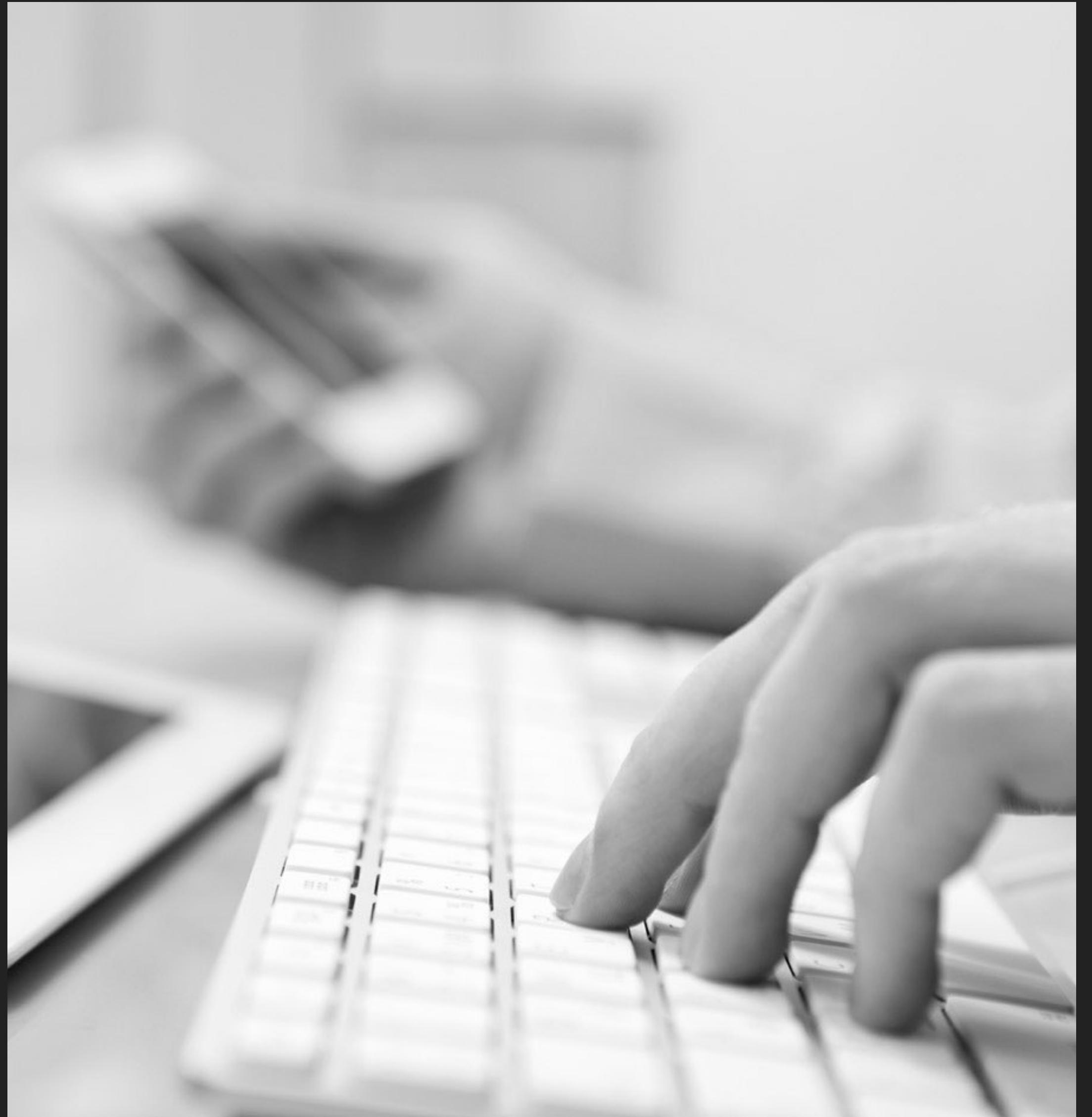
[HTTPS://GITHUB.COM/RENEFX/IOS-2018-01](https://github.com/renefx/ios-2018-01)

GITHUB
ALÉM DO TRADICIONAL BLACKBOARD DO IESB

O QUE VAMOS FAZER HOJE?

AGENDA

- ▶ Compartilhar - UIActivityViewController
- ▶ Alertas
- ▶ Modais
- ▶ Componente

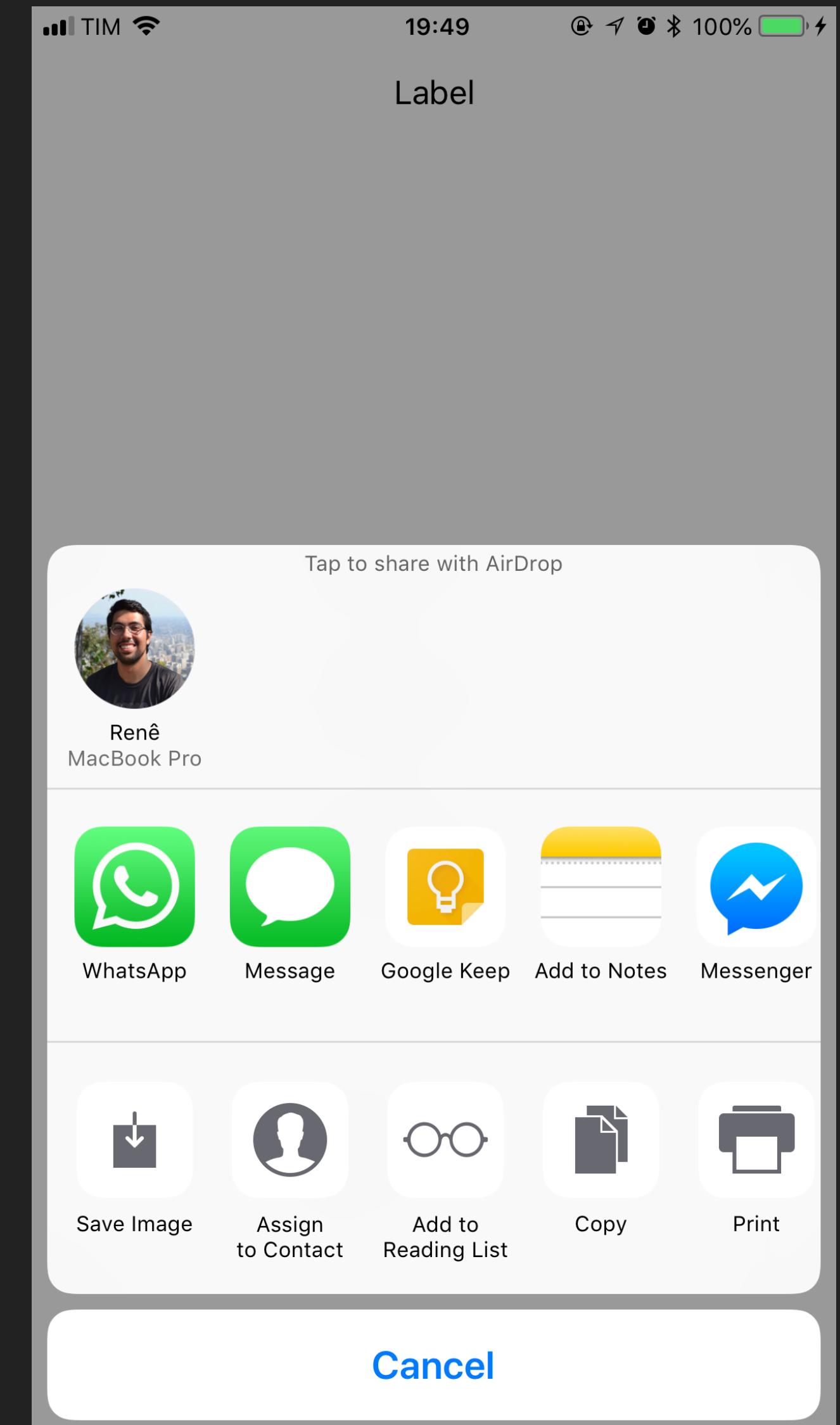




UIACTIVITY VIEWCONTROLLER

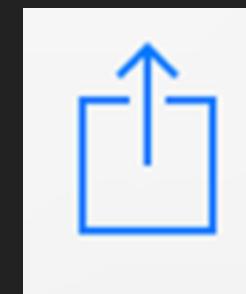
UIACTIVITYVIEWCONTROLLER

- ▶ Não tem segredo
- ▶ Você pode compartilhar um texto, imagem, link ou os três
- ▶ Geralmente usado ao clicar no botão 



UIACTIVITYVIEWCONTROLLER

- ▶ Não tem segredo
- ▶ Você pode compartilhar um texto, imagem, link ou os três
- ▶ Geralmente usado ao clicar no botão



```
let texto = "This is the text...."
let imagem = UIImage(named: "liked")
let meuLink = URL(string:"https://www.google.com")

if let meuLink = meuLink, let imagem = imagem {
    let arrayCompartilhar = [texto, imagem, meuLink] as [Any]

    let activityViewController = UIActivityViewController(activityItems: arrayCompartilhar, applicationActivities: nil)
    activityViewController.popoverPresentationController?.sourceView = self.view

    self.present(activityViewController, animated: true, completion: nil)
}
```

UIACTIVITYVIEWCONTROLLER

- ▶ Não tem segredo
- ▶ Você pode compartilhar um texto, imagem, link ou os três
- ▶ Geralmente usado ao clicar no botão 
- ▶ Podemos determinar quais ítems não iremos exibir

```
let texto = "This is the text...."
let imagem = UIImage(named: "liked")
let meuLink = URL(string:"https://www.google.com")

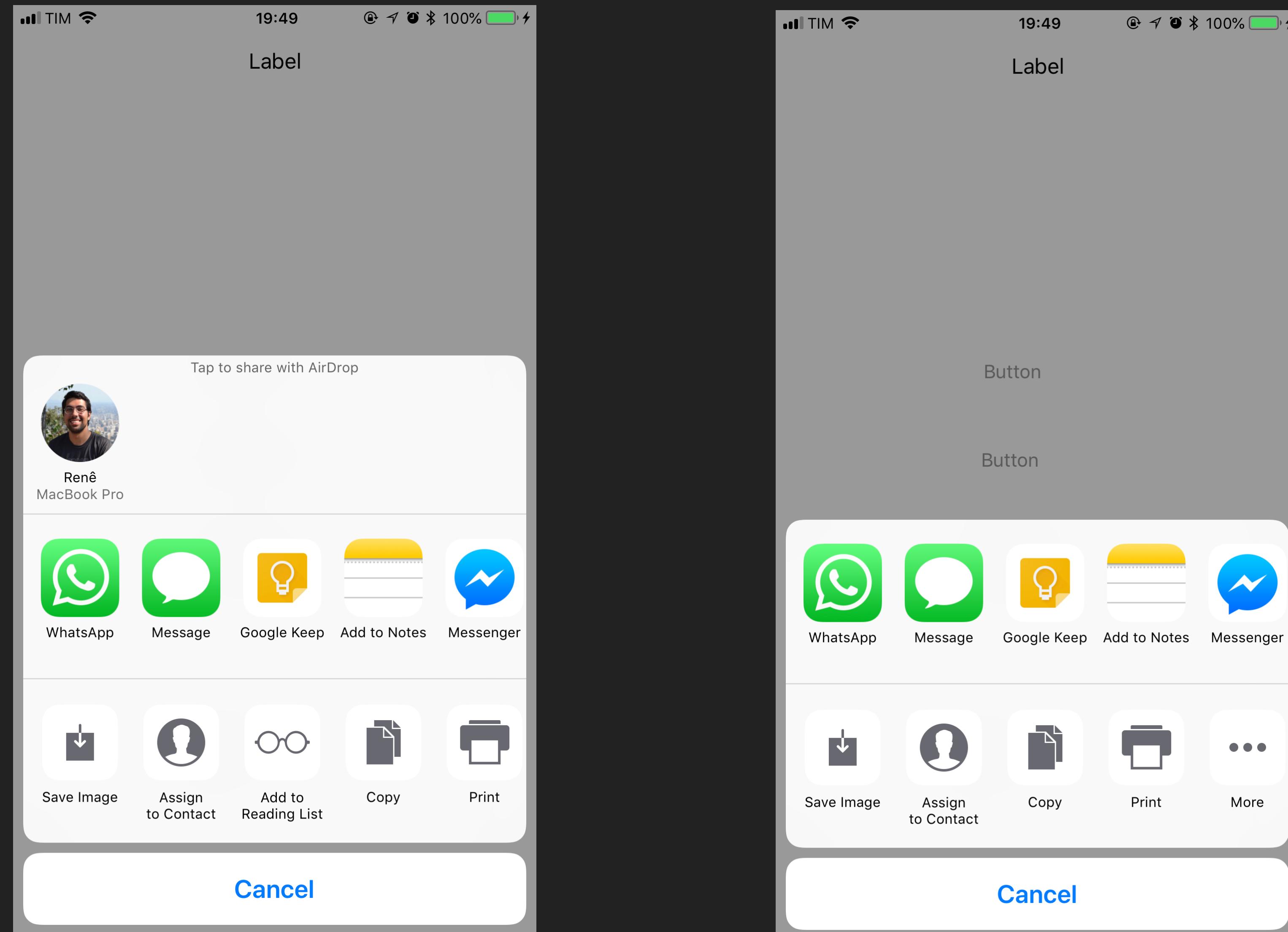
if let meuLink = meuLink, let imagem = imagem {
    let arrayCompartilhar = [texto, imagem, meuLink] as [Any]

    let activityViewController = UIActivityViewController(activityItems: arrayCompartilhar, applicationActivities: nil)
    activityViewController.popoverPresentationController?.sourceView = self.view
    activityViewController.excludedActivityTypes = [.airDrop, .addToReadingList]
}

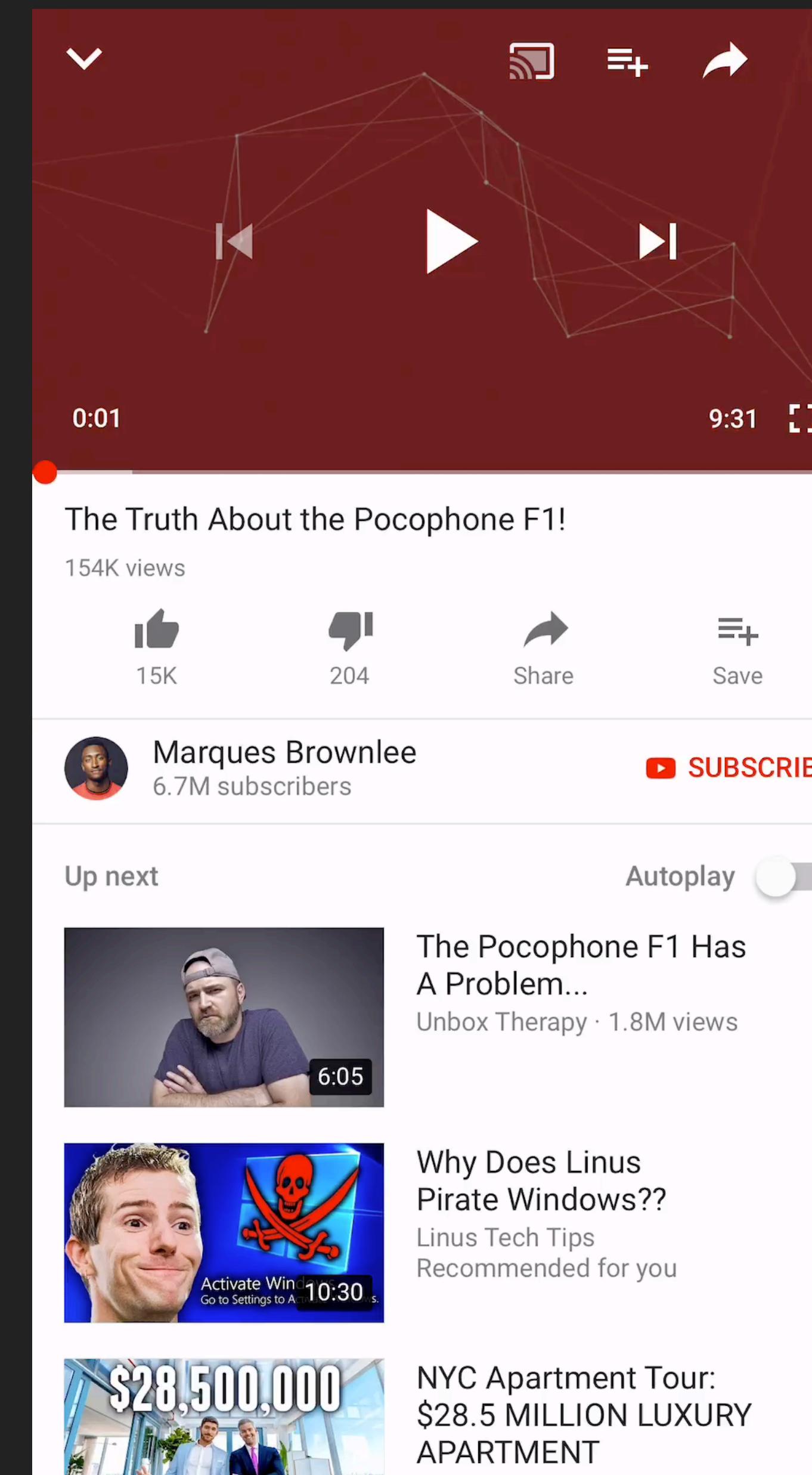
self.present(activityViewController, animated: true, completion: nil)
```

UIACTIVITYVIEWCONTROLLER

► Podemos determinar quais ítems não iremos exibir



UIACTIVITYVIEWCONTROLLER

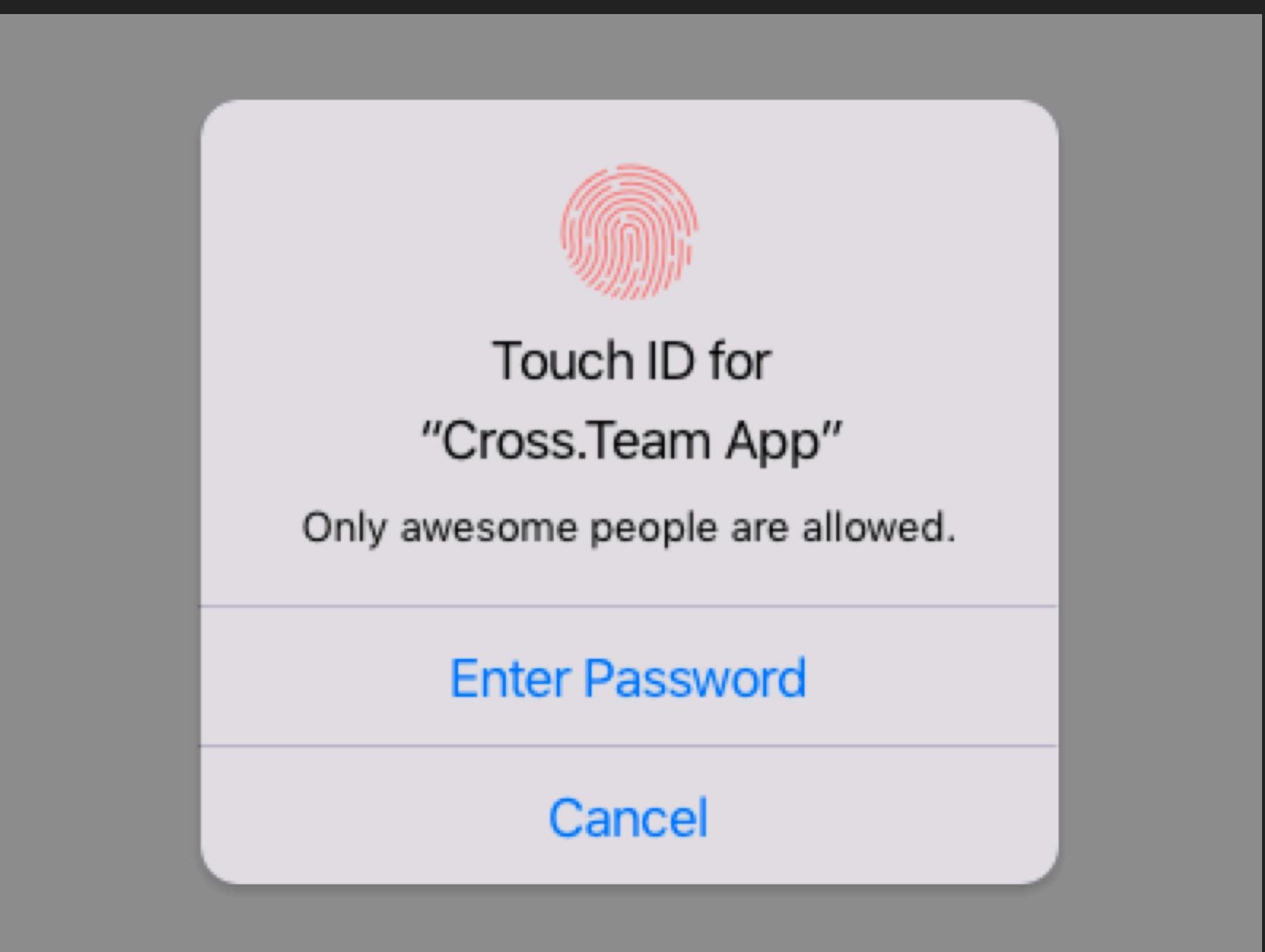
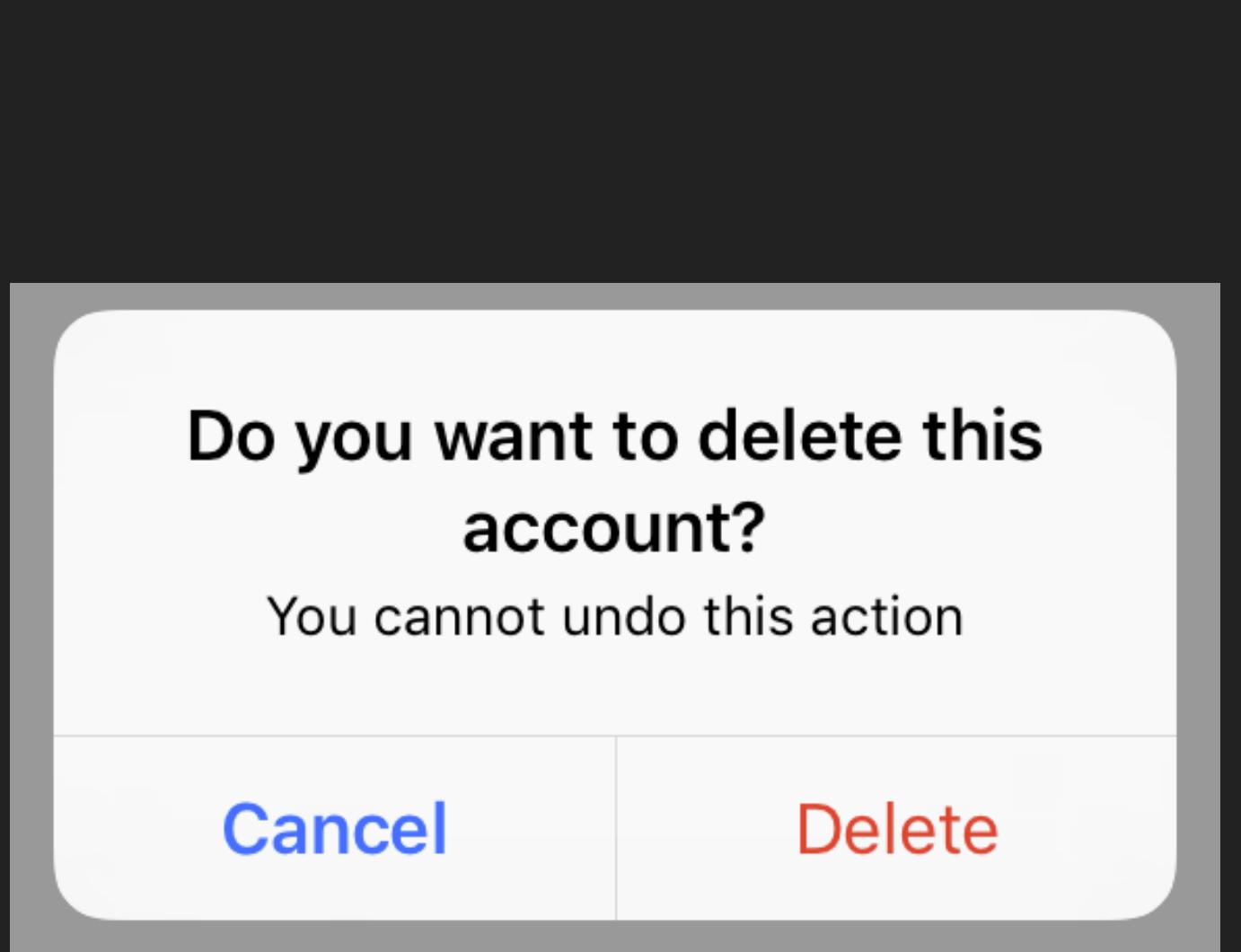
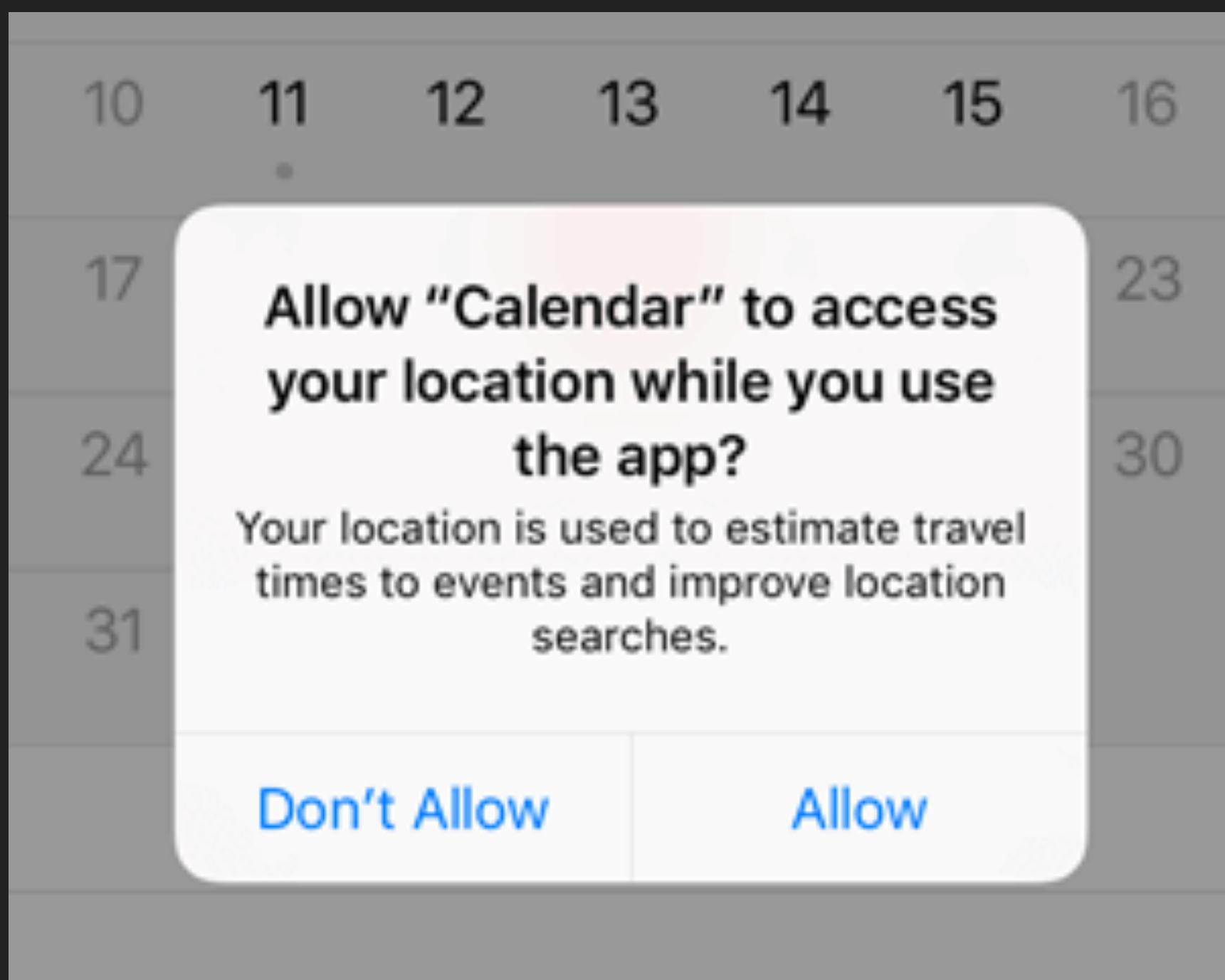




ALERTAS

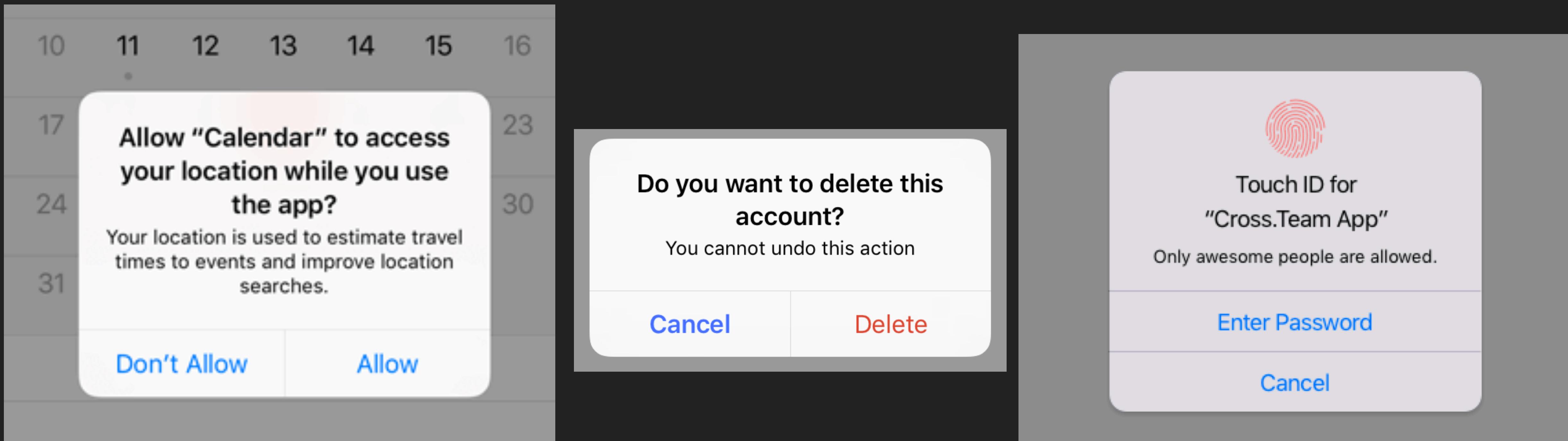
ALERTS

- ▶ Muito usado para confirmação do usuário
- ▶ Perguntas de sim ou não
- ▶ Confirmação/Permissão



ALERTS

- ▶ Preenche a tela toda
- ▶ O usuário tem que tomar uma ação, ainda que seja a ação de cancelar
- ▶ É outro tipo especial de ViewController, feito especialmente para alertas



ALERTS

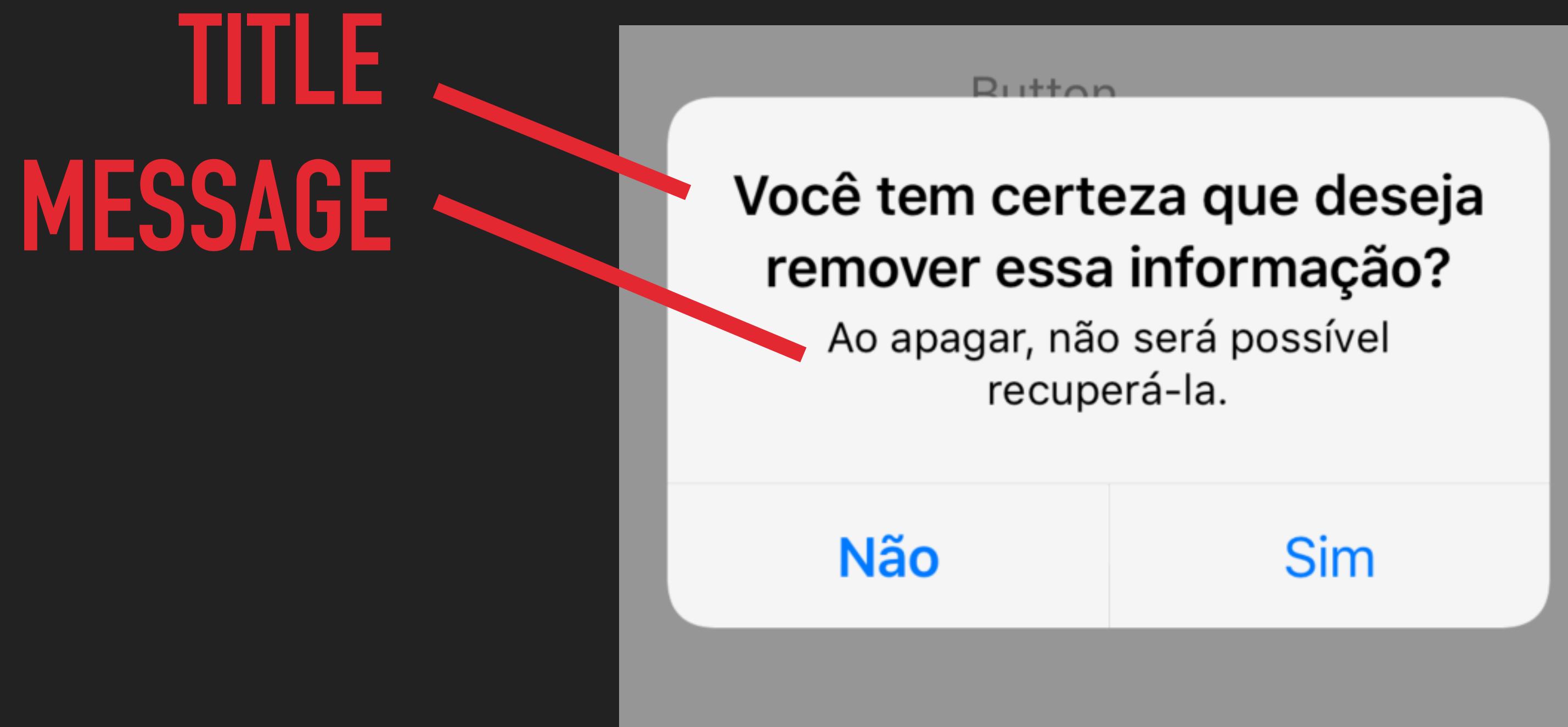
- ▶ Para criar um alerta:
- ▶ Primeiro, crie uma variável iniciando nossa UIAlertController

```
let alert = UIAlertController(title: "Você tem certeza que deseja remover essa informação?",  
message: "Ao apagar, não será possível recuperá-la.", preferredStyle: .alert)
```

ALERTS

- ▶ Para criar um alerta:
- ▶ Primeiro, crie uma variável iniciando nossa UIAlertController

```
let alert = UIAlertController(title: "Você tem certeza que deseja remover essa informação?",  
message: "Ao apagar, não será possível recuperá-la.", preferredStyle: .alert)
```



ALERTS

- ▶ Para criar um alerta:
- ▶ Primeiro, crie uma variável iniciando nossa UIAlertController

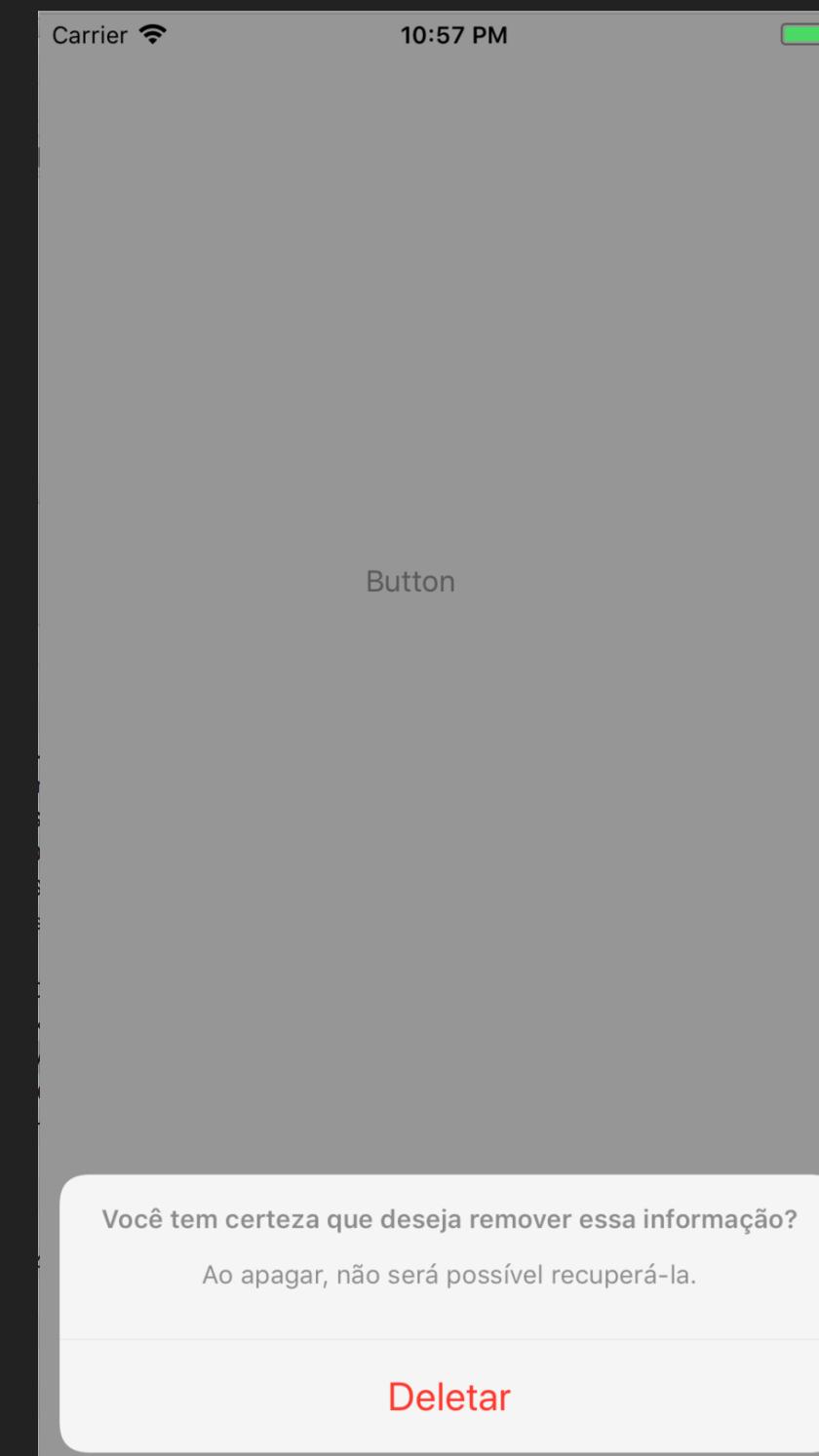
```
let alert = UIAlertController(title: "Você tem certeza que deseja remover essa informação?",  
message: "Ao apagar, não será possível recuperá-la.", preferredStyle: .alert)
```



preferredStyle:

.ALERT

.ACTIONSHEET

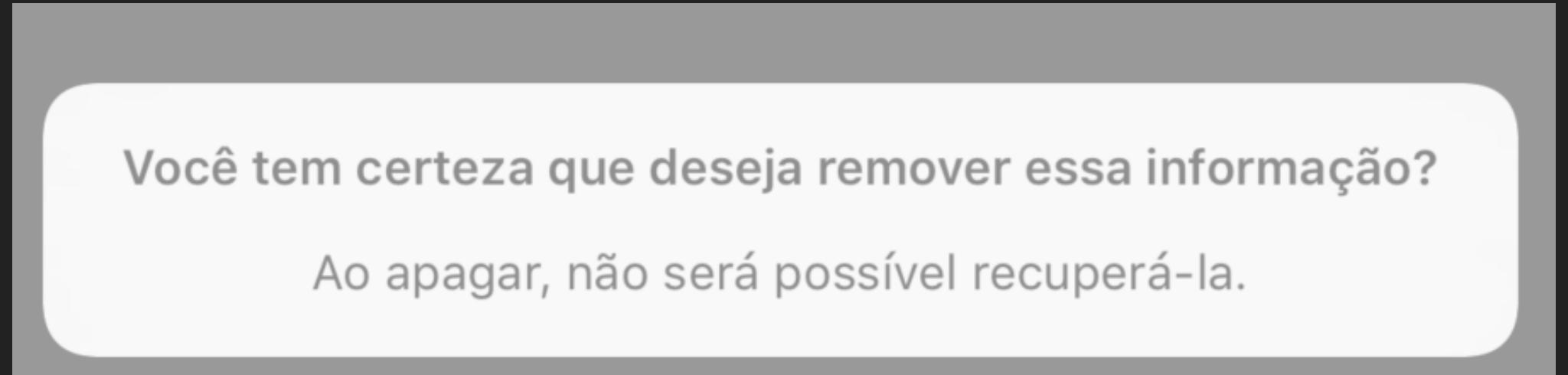


ALERTS

- ▶ Vamos apresentar esse alerta, usando o present
- ▶ Vamos lembrar que o present faz parte da navegação. Funciona em uma UINavigationController, mas não em uma UITableViewCell

```
let alert = UIAlertController(title: "Você tem certeza que deseja remover essa informação?",  
    message: "Ao apagar, não será possível recuperá-la.", preferredStyle: .actionSheet)  
  
self.present(alert, animated: true)
```

- ▶ Ainda falta colocar os botões (actions), para o usuário selecionar:



ALERTS

- ▶ Um Alert pode ser somente informativo. Somente um botão, mas ao clicar Ok não será tomada nenhuma ação. O alerta só irá deixar de ser exibido.

```
let alert = UIAlertController(title: "Sem conexão", message: "Você está sem conexão com a Internet. Conecte-se para prosseguir.", preferredStyle: .alert)

alert.addAction(UIAlertAction(title: "OK", style: .default, handler: nil))

self.present(alert, animated: true)
```



ALERTS

- ▶ Um Alert pode ter a função de tomada de decisão. Seus botões podem executar uma ação quando clicados.
- ▶ Por uma questão de padrão do iOS, sempre devemos colocar um botão cancelar.
- ▶ O style alert só permite adicionar duas ações
- ▶ O style actionSheet não tem limites de ações

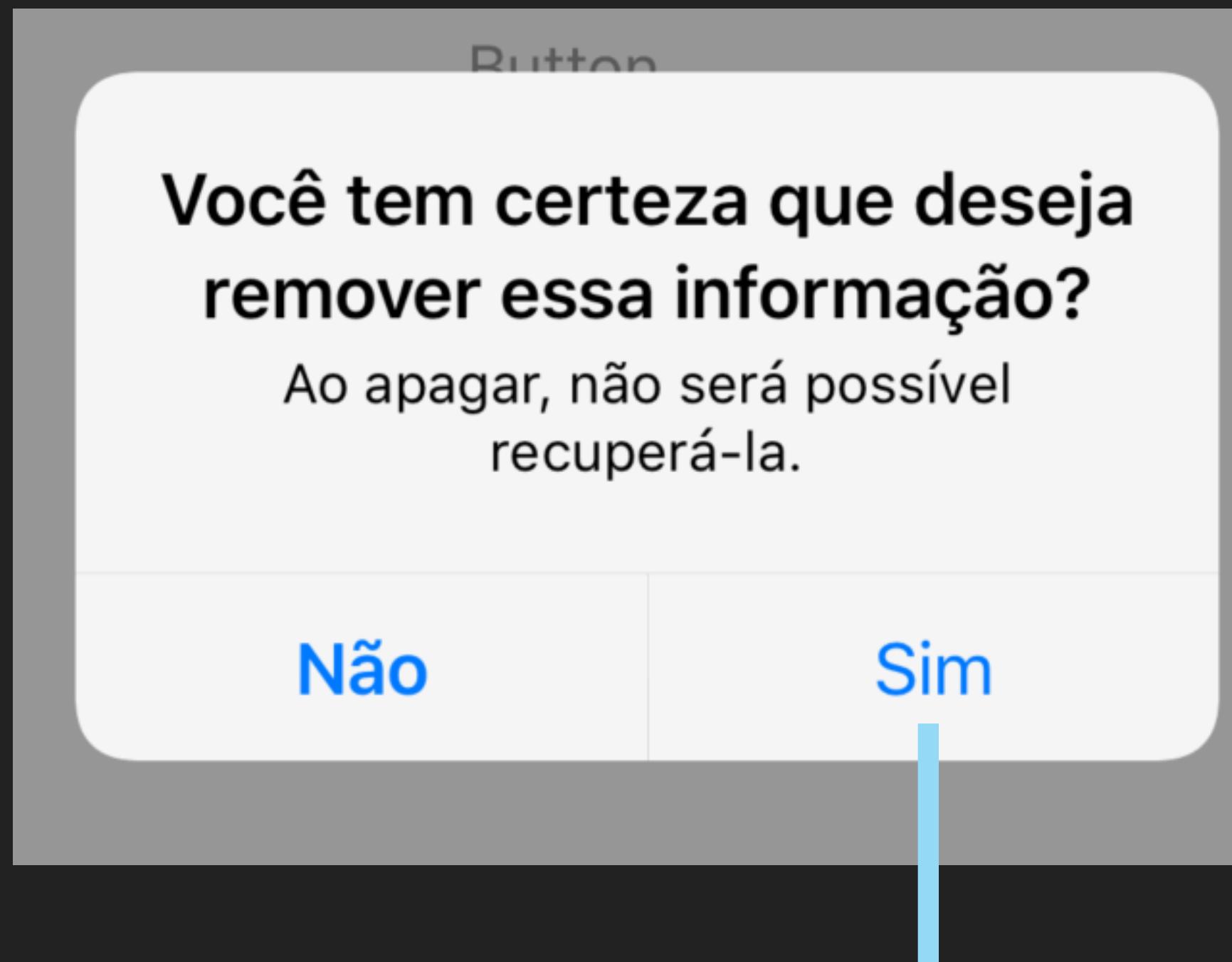
ALERTS

```
let alert = UIAlertController(title: "Você tem certeza que deseja remover essa informação?",  
    message: "Ao apagar, não será possível recuperá-la.", preferredStyle: .actionSheet)  
  
alert.addAction(UIAlertAction(title: "Apagar", style: .destructive, handler: nil))  
  
self.present(alert, animated: true)
```

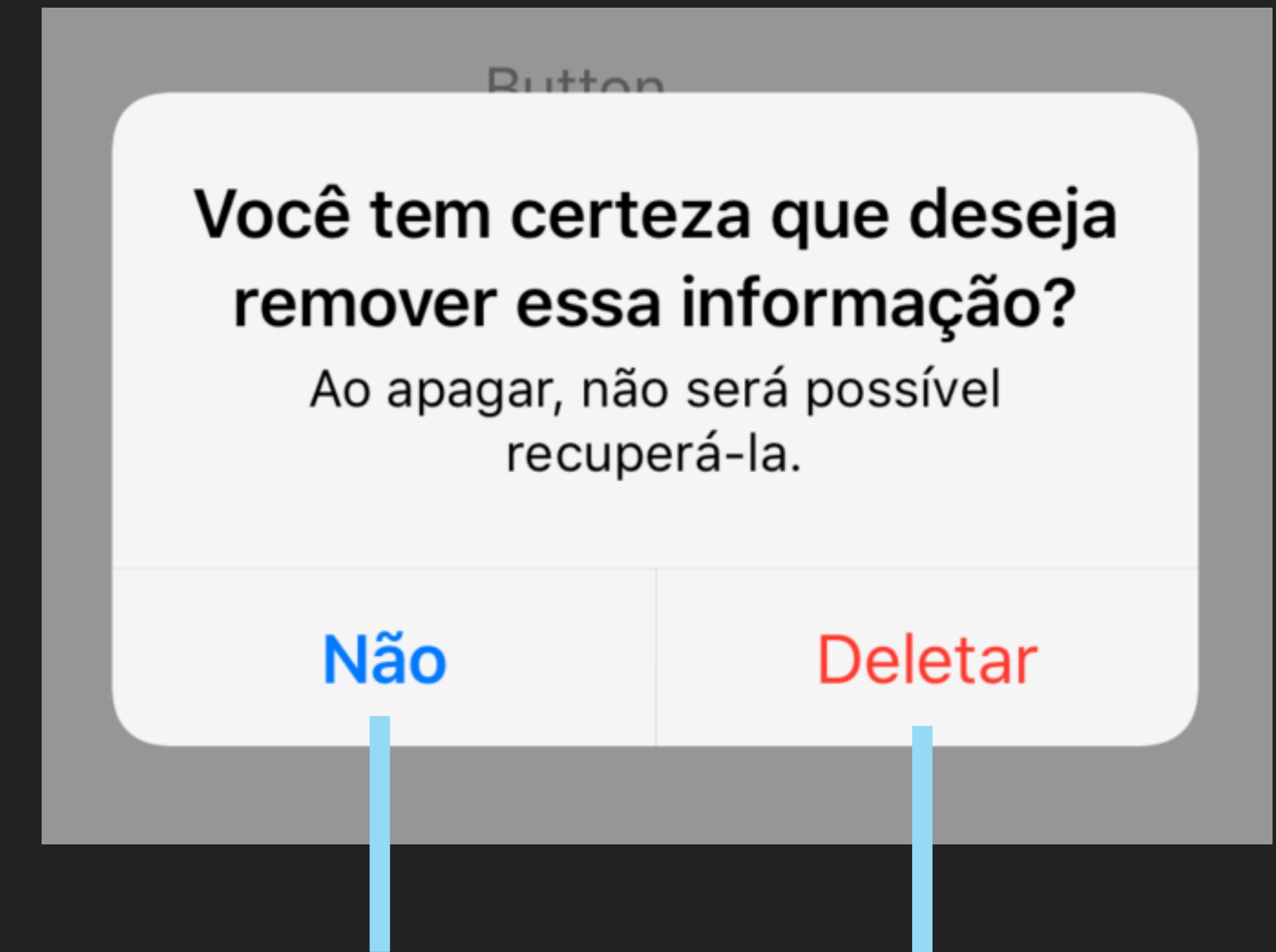
- ▶ title - texto que irá aparecer no botão
- ▶ style - estilo do texto
- ▶ handler - uma closure para lidar com a ação daquele botão em específico. No caso de nil, o alerta só irá fechar

ALERTS

```
let alert = UIAlertController(title: "Você tem certeza que deseja remover essa informação?",  
    message: "Ao apagar, não será possível recuperá-la.", preferredStyle: .actionSheet)  
  
alert.addAction(UIAlertAction(title: "Apagar", style: .destructive, handler: nil))  
  
self.present(alert, animated: true)
```



.DEFAULT



.CANCEL

.DESTRUCTIVE

ALERTS

```
alert.addAction(UIAlertAction(title: "Apagar", style: .default, handler: {
```

```
}))
```

! Cannot convert value of type '() -> ()' to expected argument type '((UIAlertAction) -> Void)?'

x

- ▶ Ao abrir as chaves {} indicamos que estamos criando uma closure do tipo () -> ()
- ▶ O erro ocorre, pois o handler espera de parâmetro uma closure do tipo (UIAlertAction) -> Void
- ▶ Nossa closure irá receber de retorno uma UIAlertAction e não deve retornar nada

ALERTS

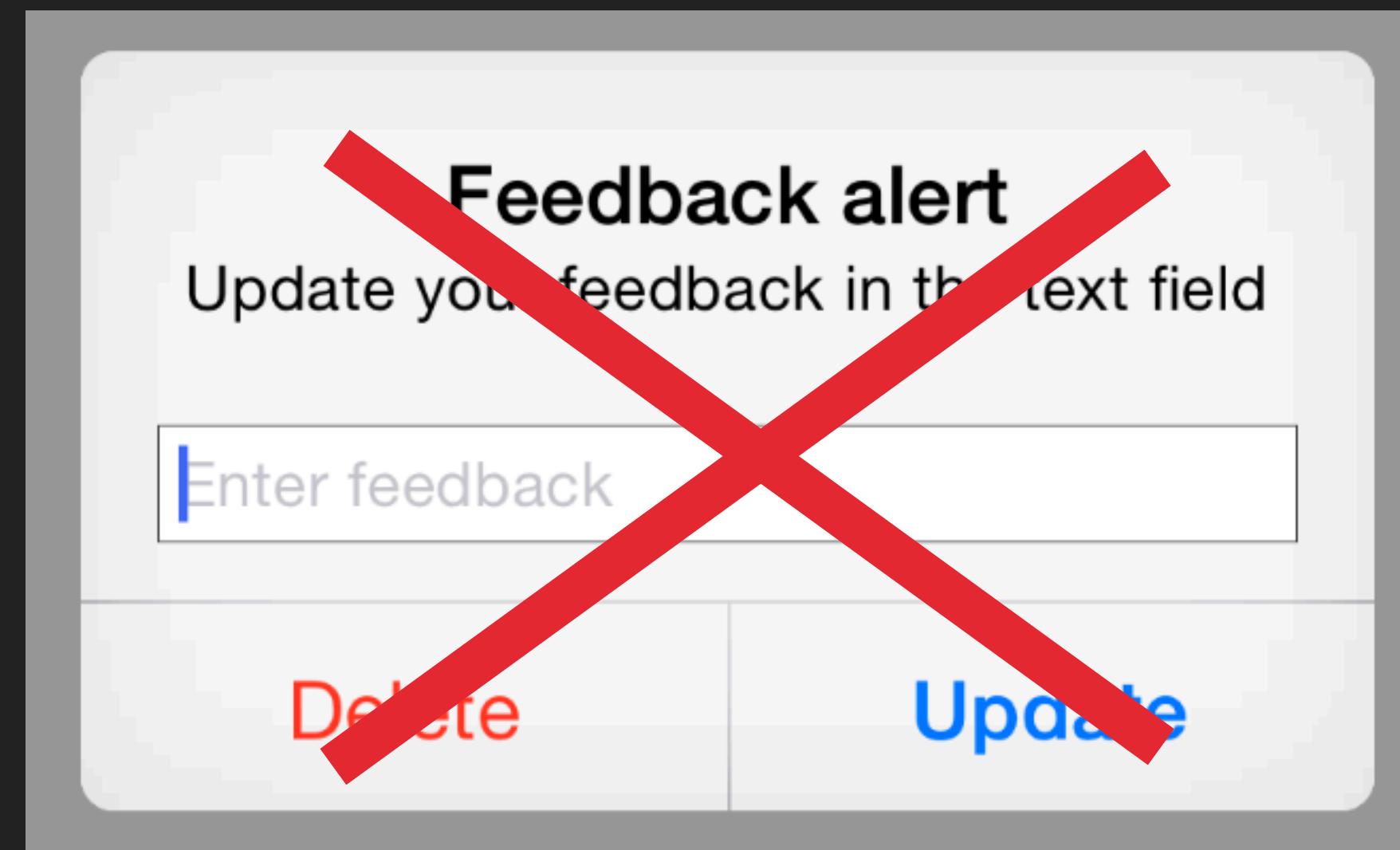
```
alert.addAction(UIAlertAction(title: "Apagar", style: .default, handler: { action in
}))
```

- ▶ Ao colocar `action in` indicamos que a variável `action` receberá a `UIAlertAction`
- ▶ O nome da variável colocada é `action`, mas poderia ser qualquer outro. A parte que retorna a closure pode escolher o nome como irá tratar a variável.
- ▶ Ali dentro podemos executar a ação que desejamos quando o botão é clicado.

ALERTS

[HTTPS://DEVELOPER.APPLE.COM/DOCUMENTATION/UIKIT/
WINDOWS AND SCREENS/
GETTING THE USER S ATTENTION WITH ALERTS AND ACTION
SHEETS](https://developer.apple.com/documentation/uikit/windows_and_screens/getting_the_user_s_attention_with_alerts_and_action_sheets)

MAS TEM COMO CUSTOMIZAR?





MODAIS

MODAIS

~
SÃO
EXTREMAMENTE
RARAS

MODAIS

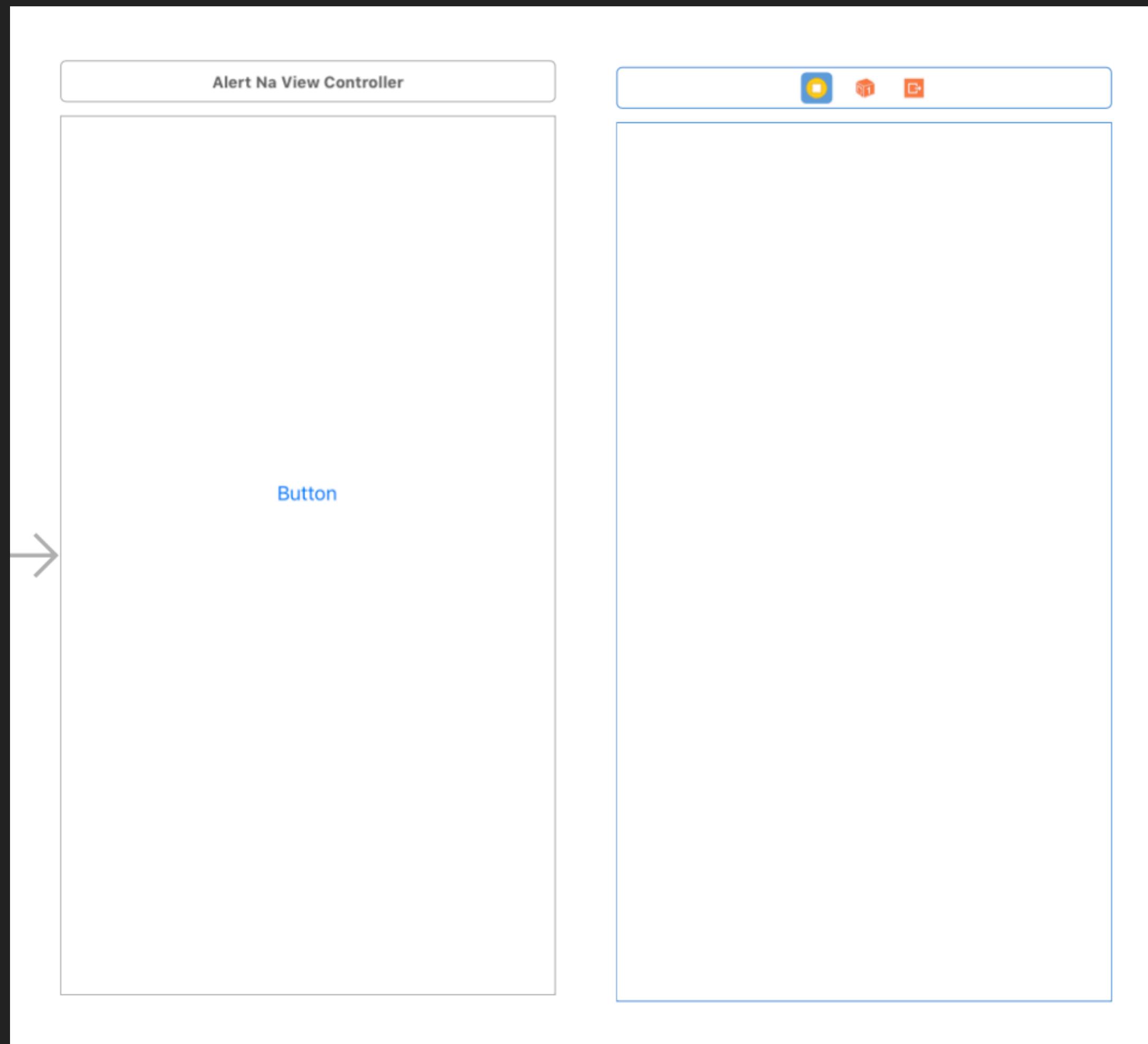
GERALMENTE SÃO
SUBSTITUÍDAS POR
VIEW CONTROLLER
PRESENT MODALLY

MODAIS

- ▶ Vamos utilizar uma ViewController para exibir nossa modal
- ▶ A tela de origem executará uma Segue do tipo Present Modally
- ▶ Seleccionaremos a Segue e informaremos que ela será apresentada sobre o contexto atual
- ▶ Informaremos que a tela que irá iniciar a exibição fornecerá o contexto

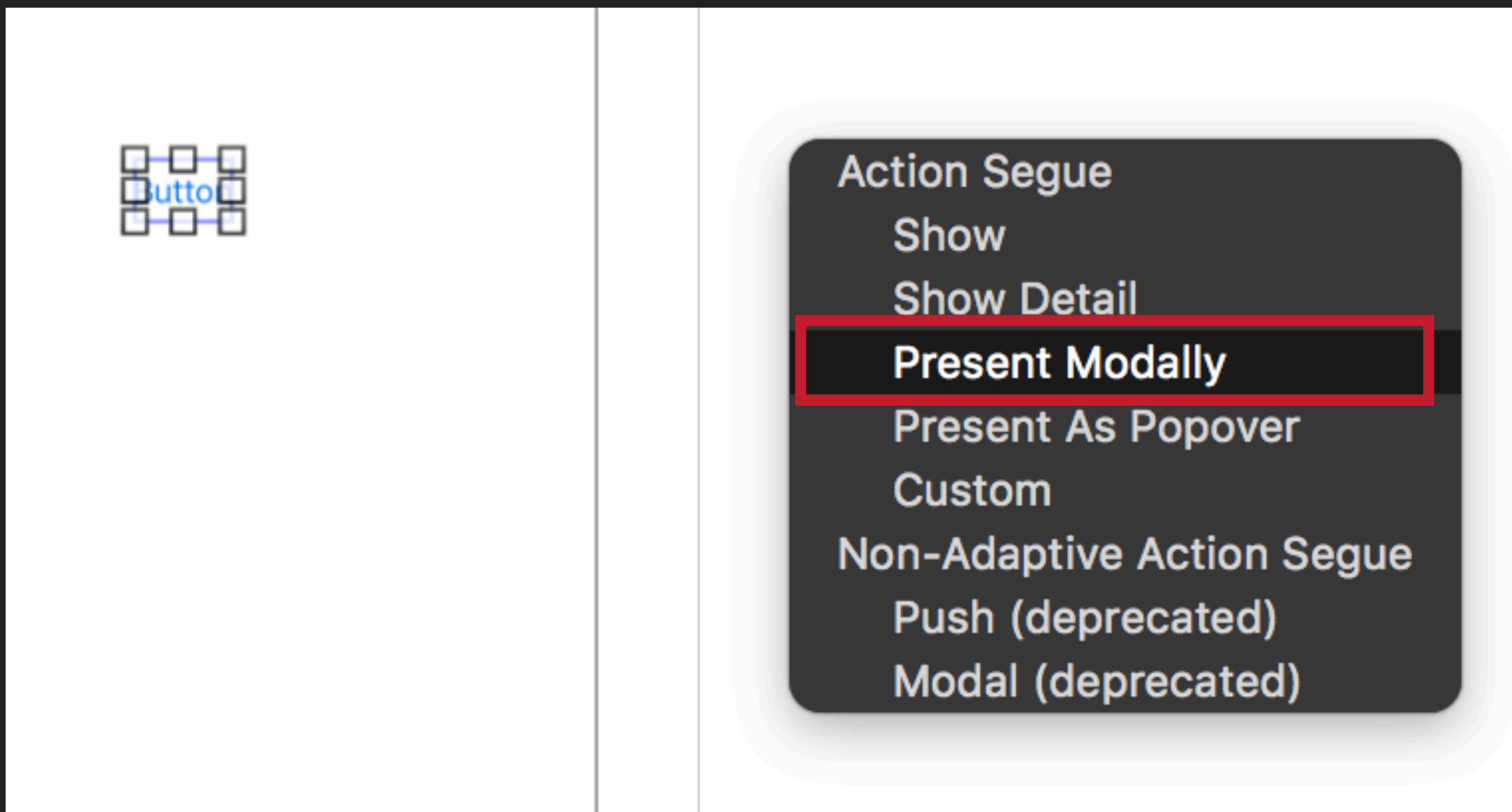
MODAIS

- ▶ Crie uma ViewController com o Button que irá abrir a Modal
- ▶ Crie outra ViewController que será a Modal



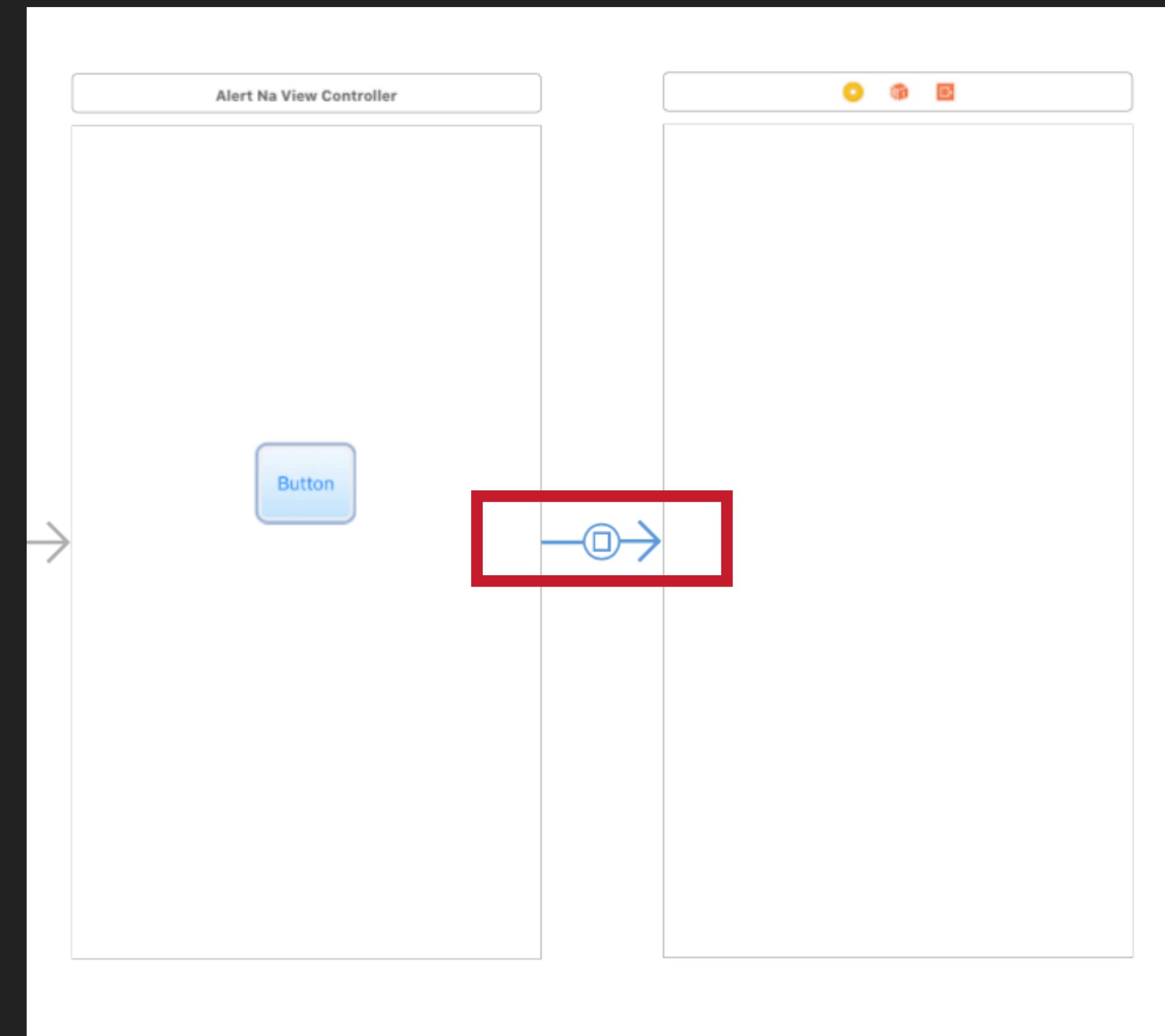
MODAIS

- ▶ Crie uma Segue do Button para a outra ViewController, mas selecione Present Modally



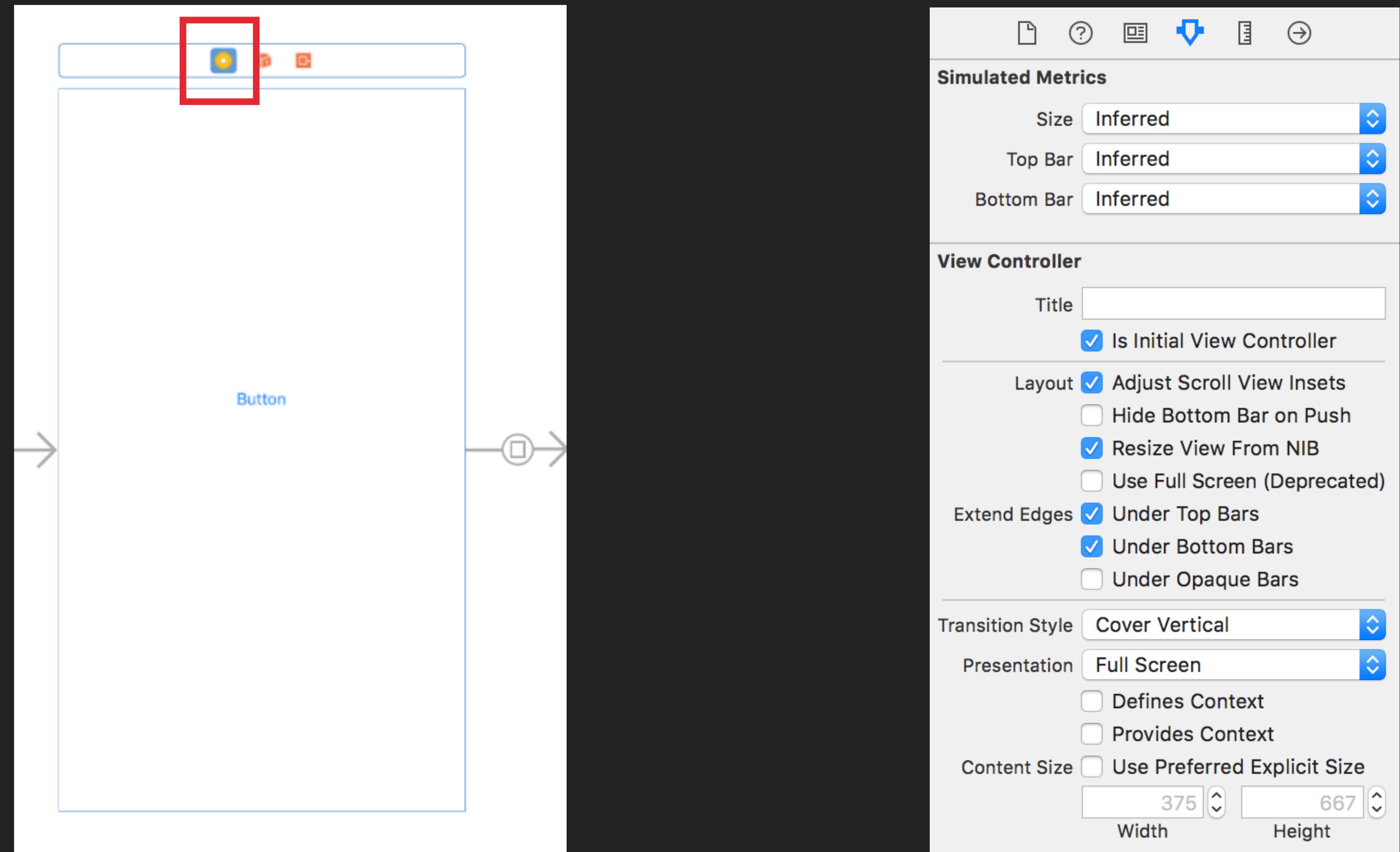
MODAIS

- ▶ A Segue foi criada. Note que ela tem um ícone diferente, um quadrado indicando Modal



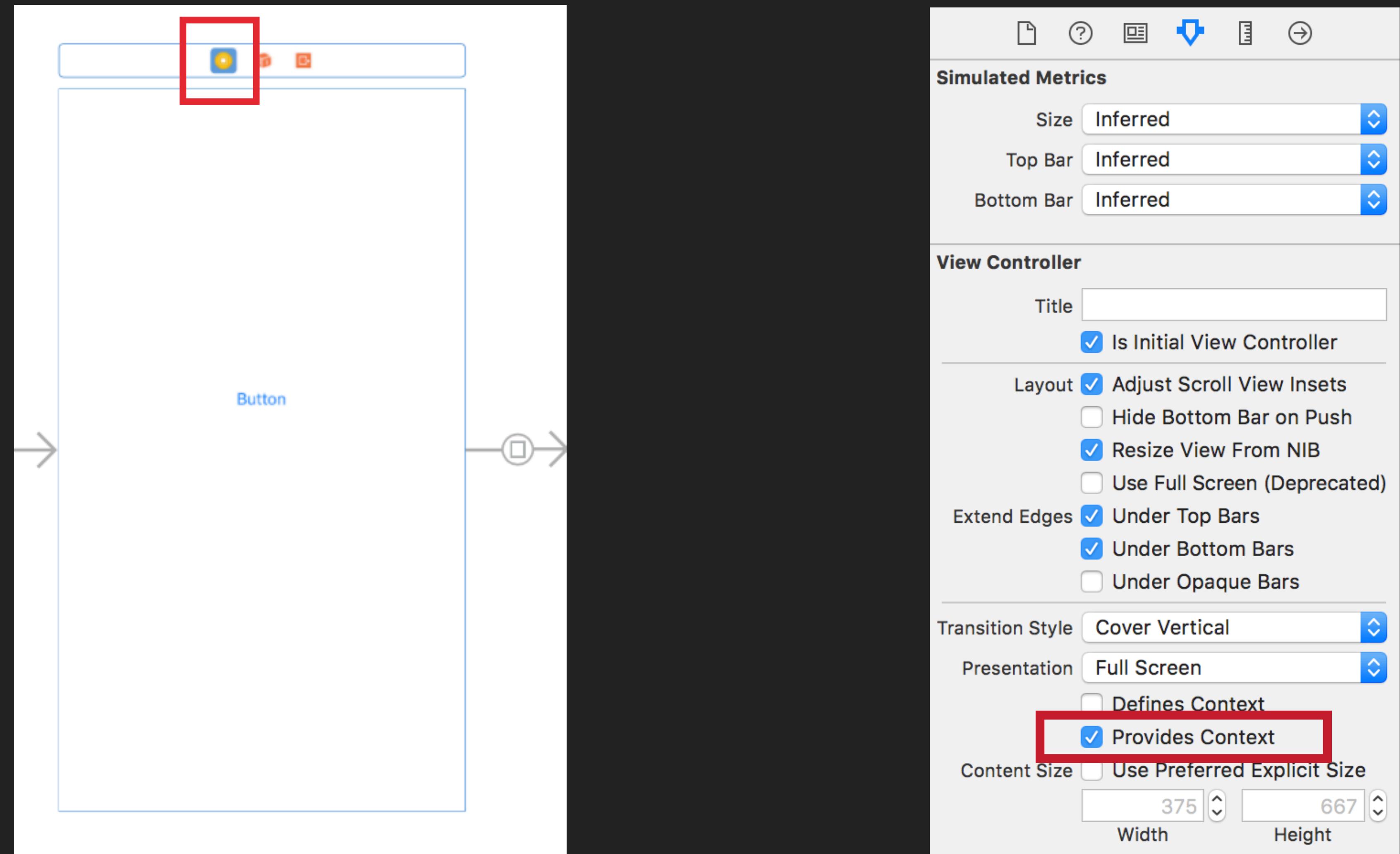
MODAIS

► Na tela com o botão que irá abrir a modal:



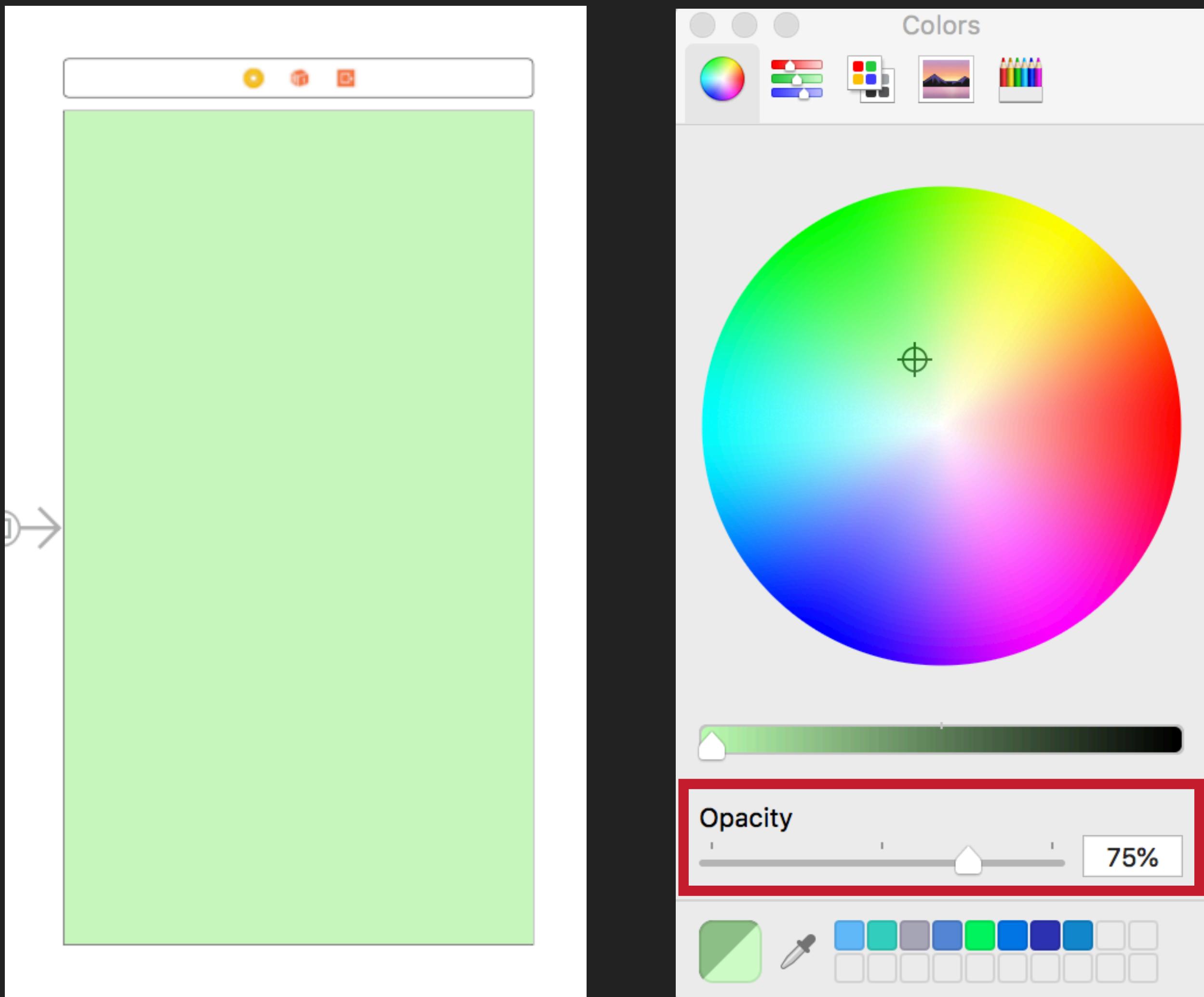
MODAIS

- ▶ Na tela com o botão que irá abrir a modal:
- ▶ Indique que ela irá prover o contexto



MODAIS

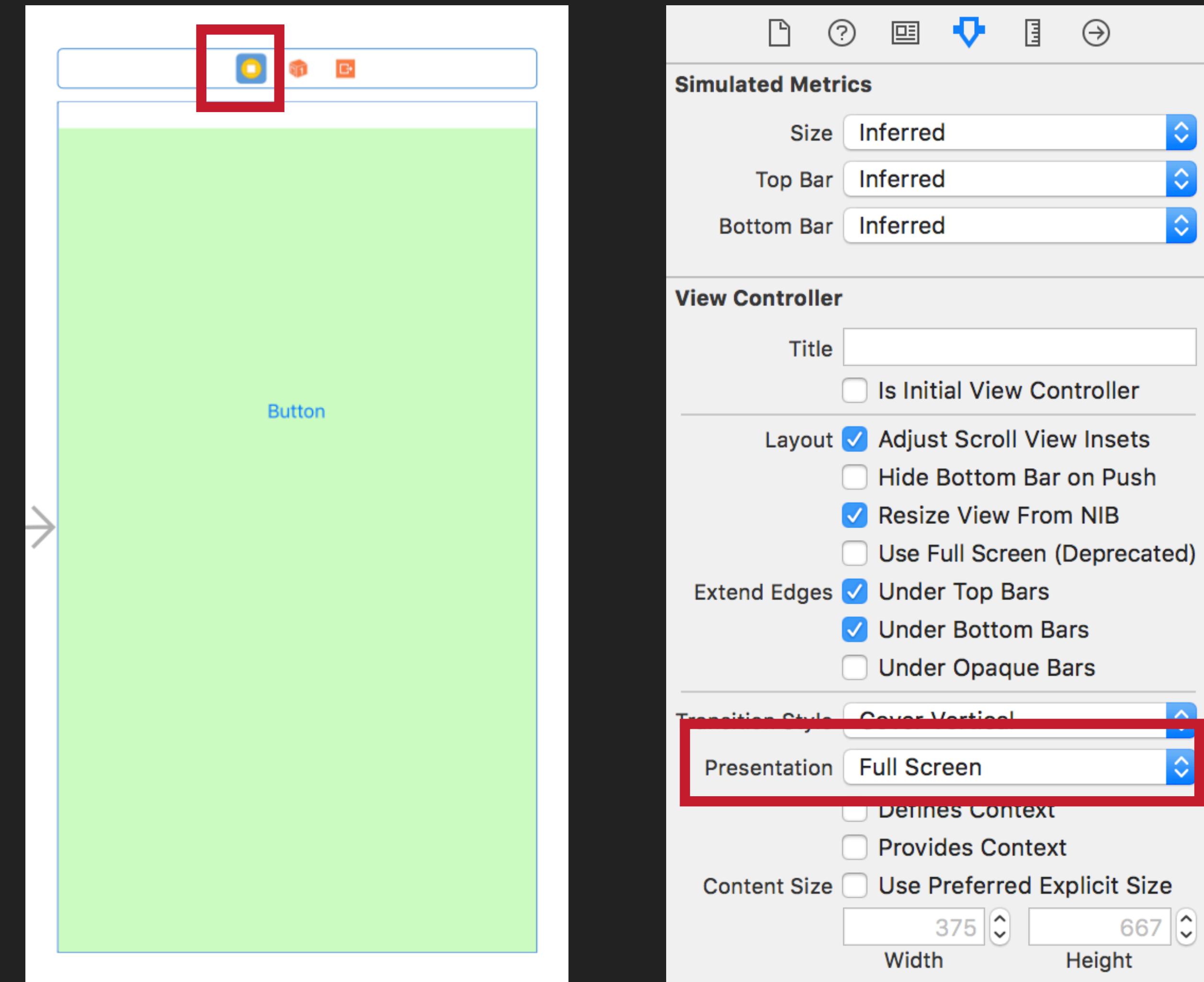
- ▶ Para colocar um fundo na Modal e ainda assim aparecer a tela anterior, coloque um background com opacidade diferente de 100%.
- ▶ Obs: não crie uma View nova, utilize a que já está como background. O padrão da View que já existe é ser branco 100%, então se criar uma nova, essa View que já existe irá impedir de ver o que está atrás.



MODAIS

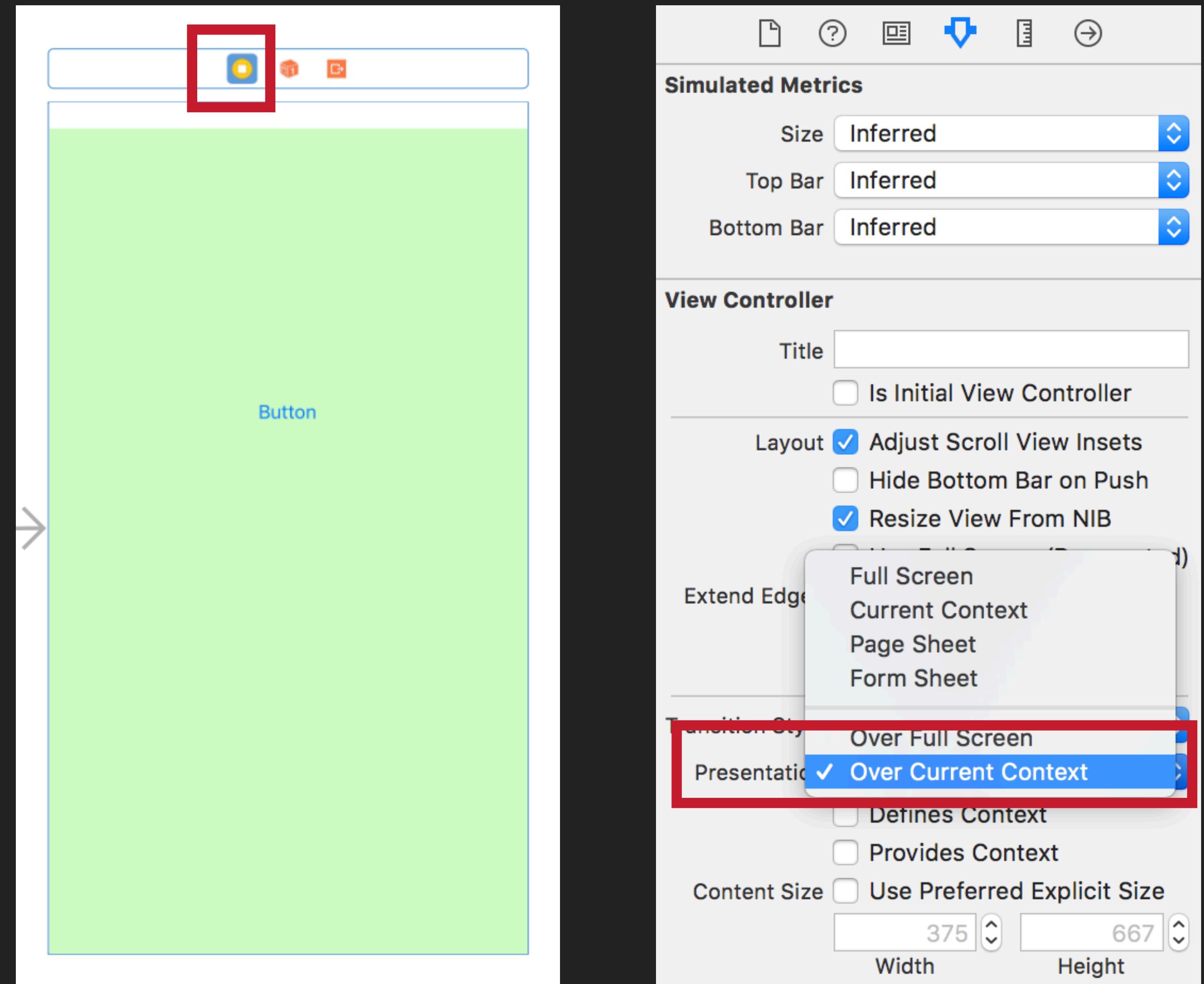
► Selecione a ViewController da nossa Modal.

► Nas propriedades, vamos selecionar a opção Presentation



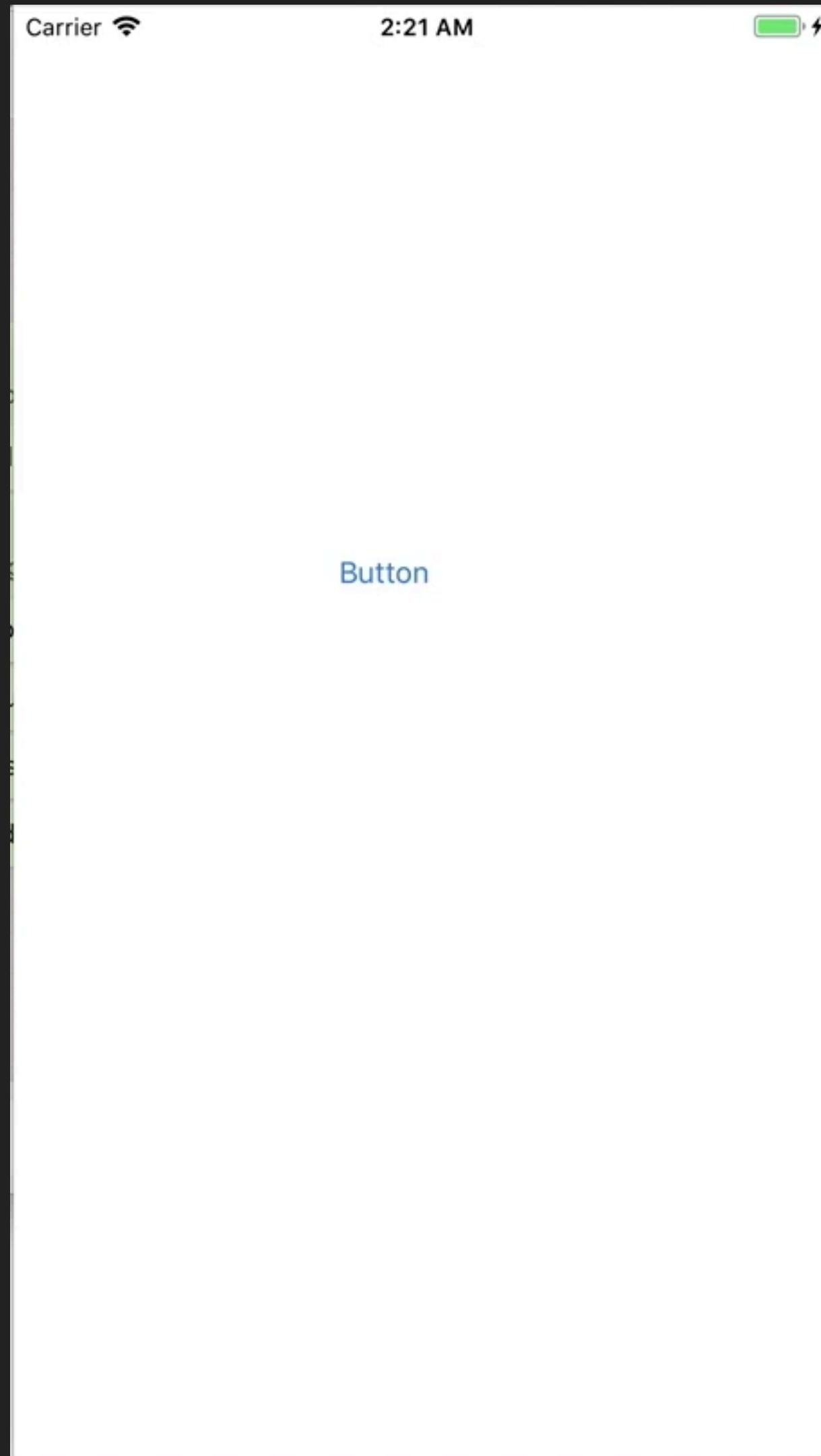
MODAIS

- ▶ Selecione a ViewController da nossa Modal.
- ▶ Nas propriedades, vamos selecionar a opção Presentation
- ▶ E mudar para Over Current Context



MODAIS

- ▶ Com isso, nossa modal está pronta

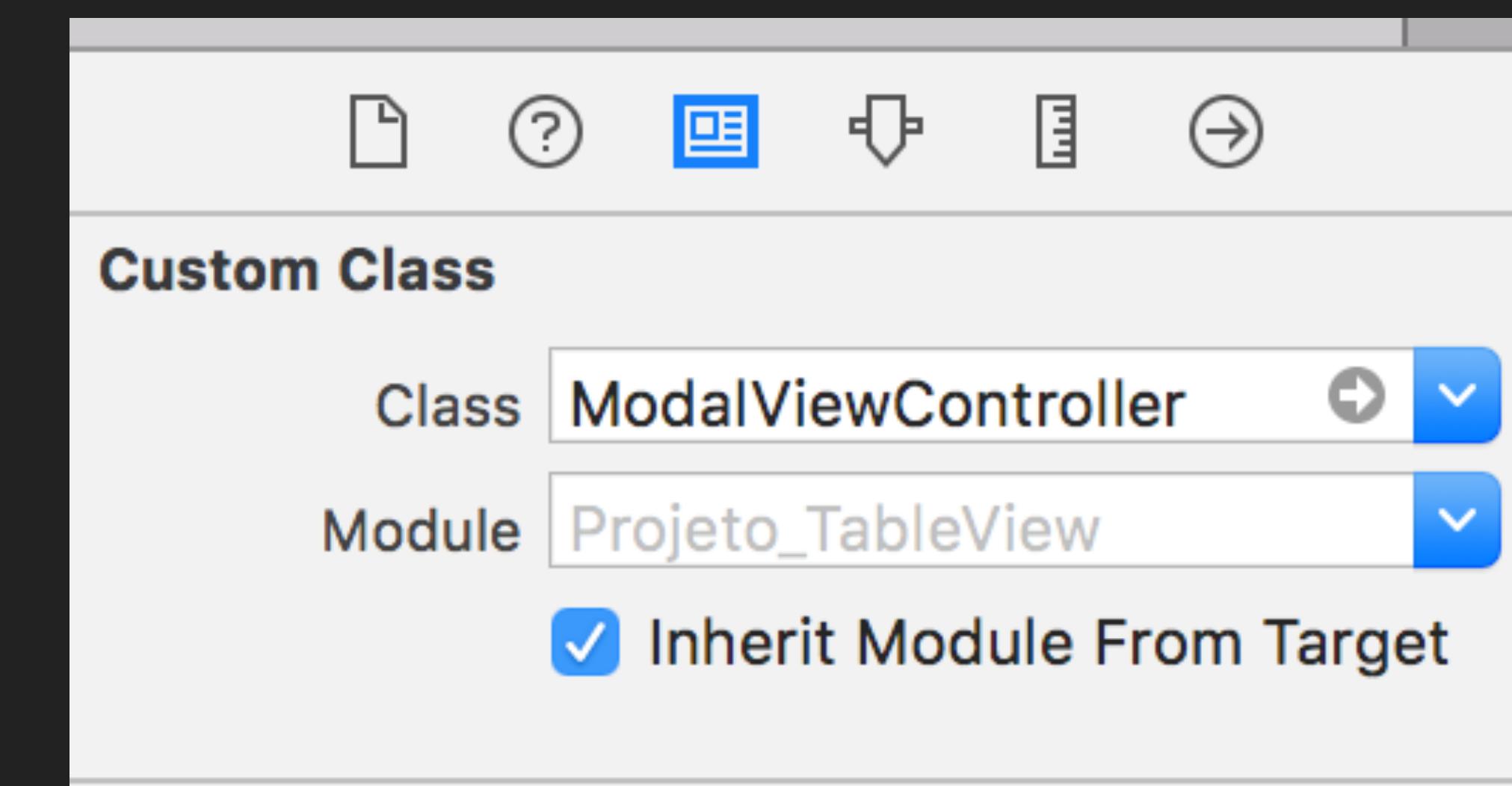


- ▶ Podemos então criar uma tela da forma que queremos para aparecer por cima da anterior.
- ▶ Podemos criar um botão Cancelar que dispensa nossa Modal

MODAIS

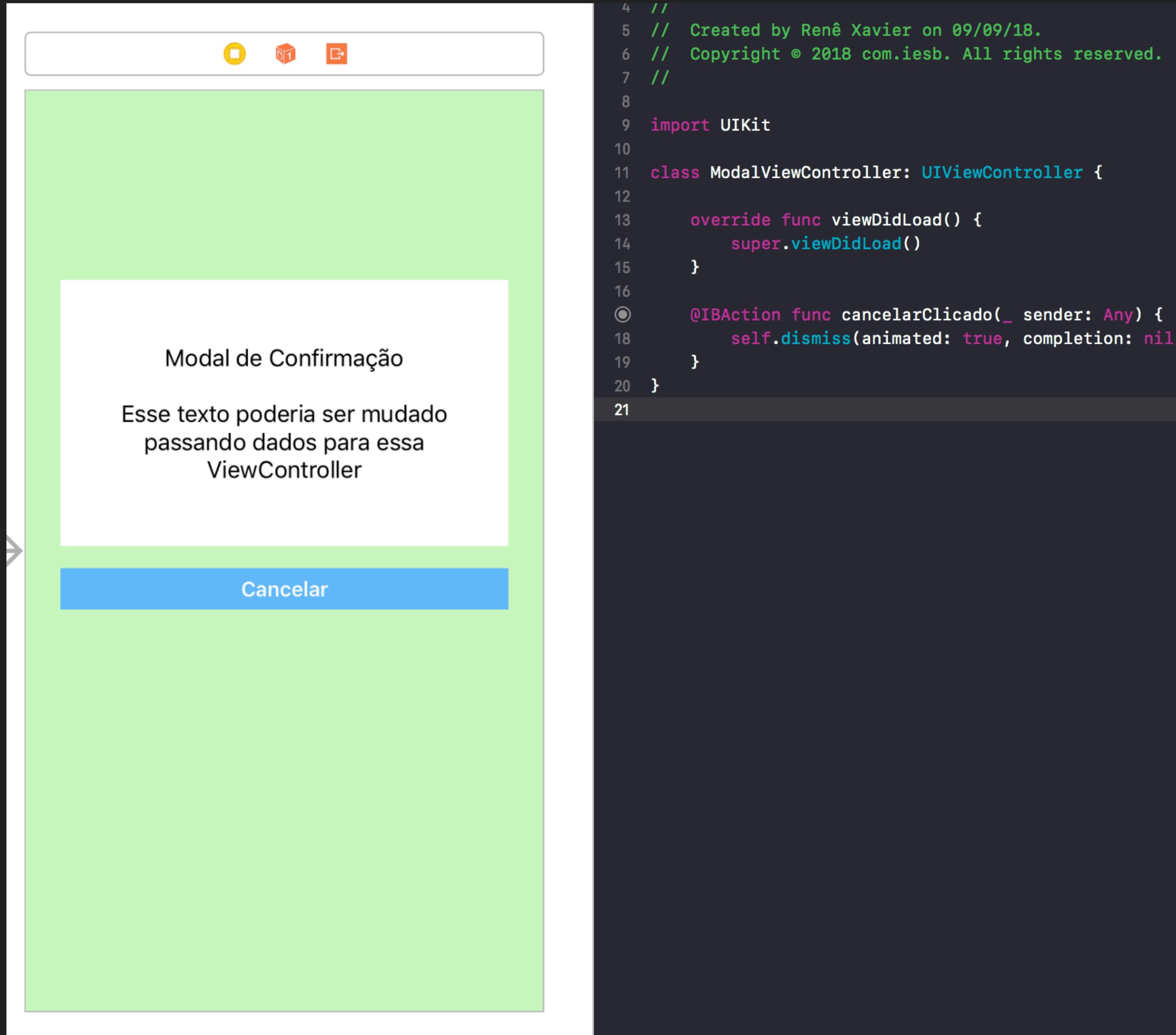
- ▶ Para isso, temos que criar uma classe para nossa Modal (do tipo `UIViewController`)
- ▶ Também temos que indicar que essa classe controla nossa Modal

```
8  
9 import UIKit  
10  
11 class ModalViewController: UIViewController {  
12  
13     override func viewDidLoad() {  
14         super.viewDidLoad()  
15     }  
16 }  
17
```

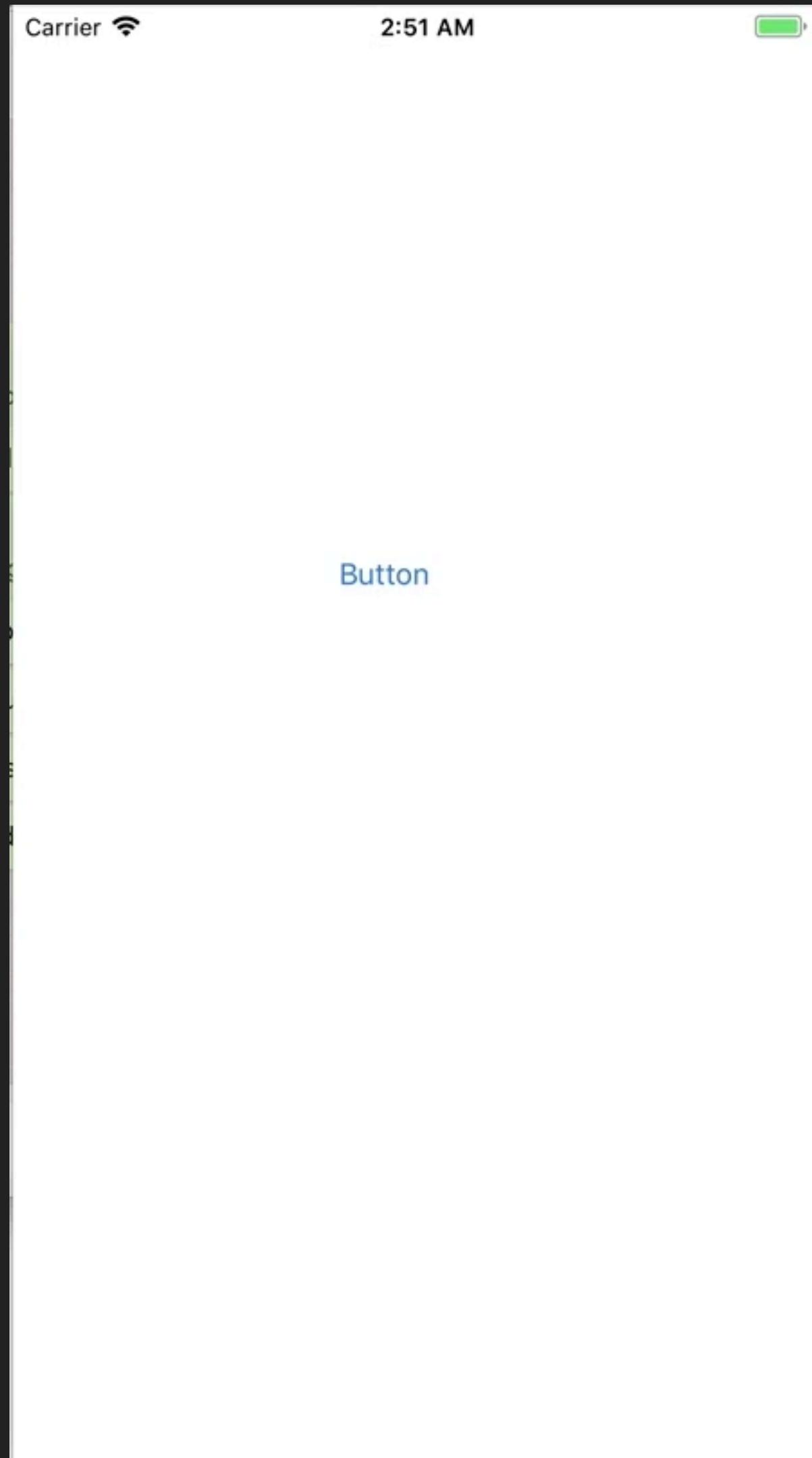


MODAIS

- Feito isso, podemos colocar Views nela pelo Storyboard



- Para retornar para a tela anterior, criamos um botão Cancelar
- Criamos uma IBAction ligando ao nosso código
- Na IBAction, executamos um:
`self.dismiss(animated: true completion: nil)`
- Podemos até mesmo colocar um botão que cobre a tela abaixo da modal e ao tocar no background o dismiss é executado.



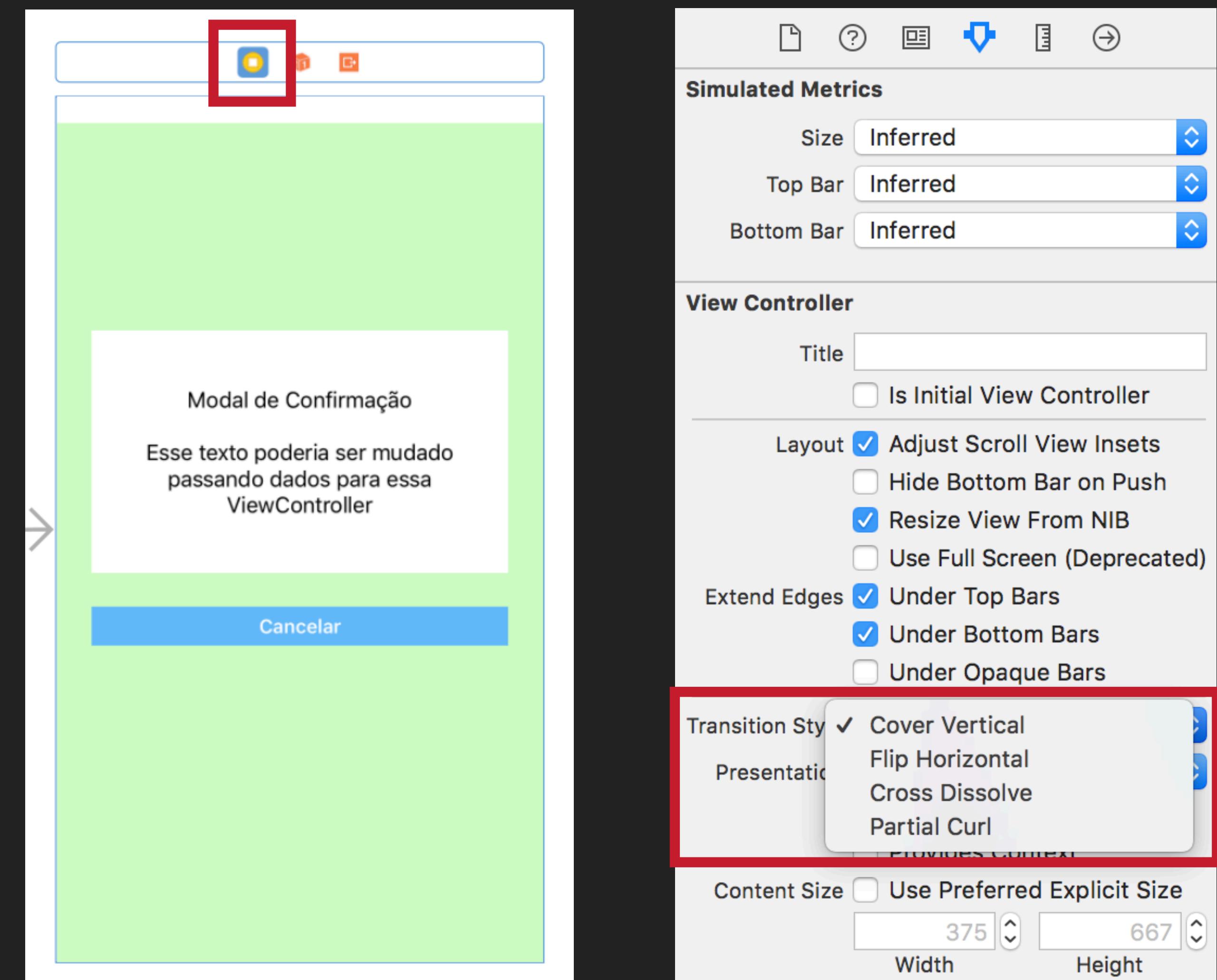
- ▶ Conseguimos também mudar essa animação
(de baixo para cima)

MODAIS

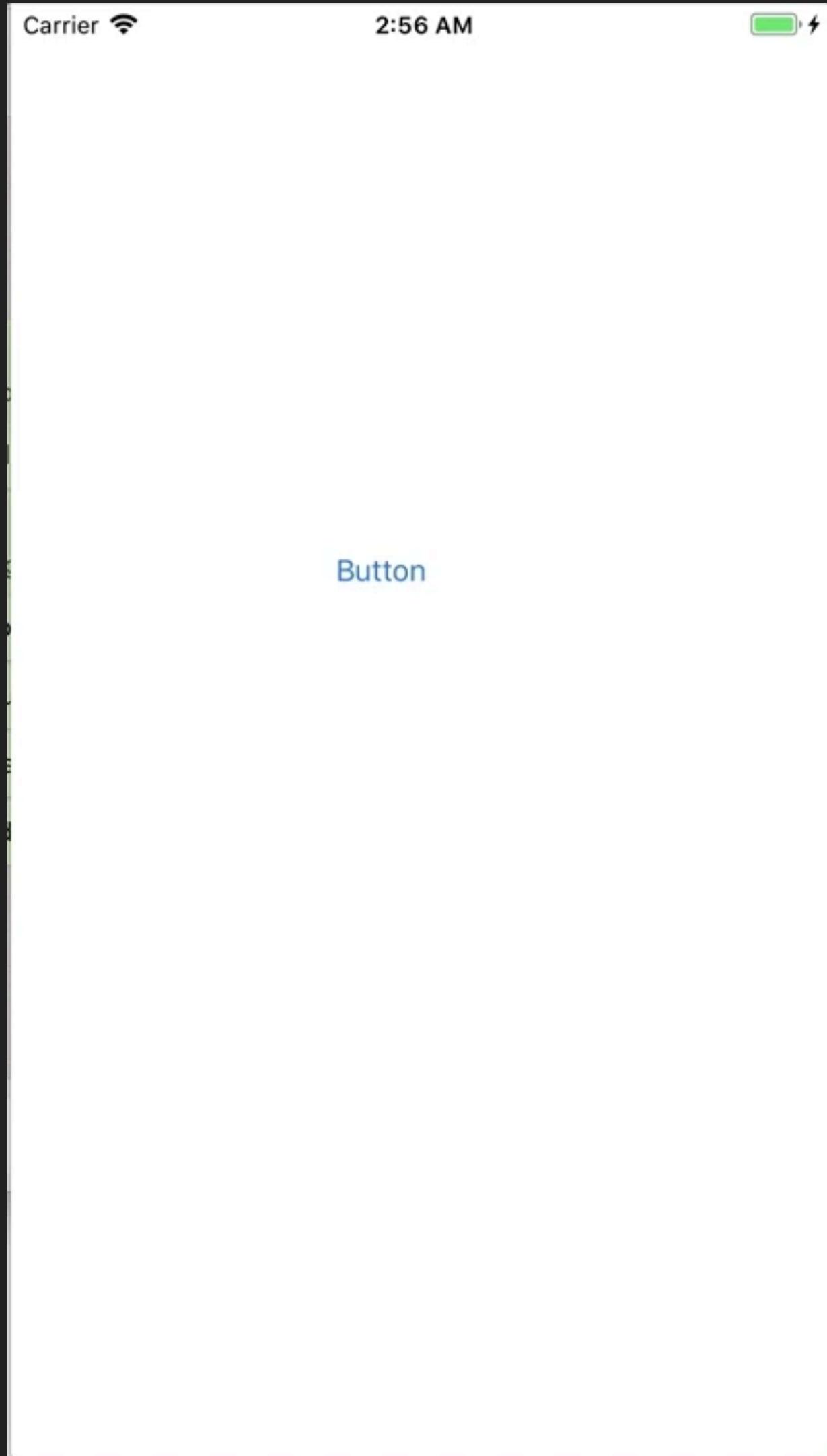
▶ Para mudar a animação é simples.

▶ Selecione a ViewController

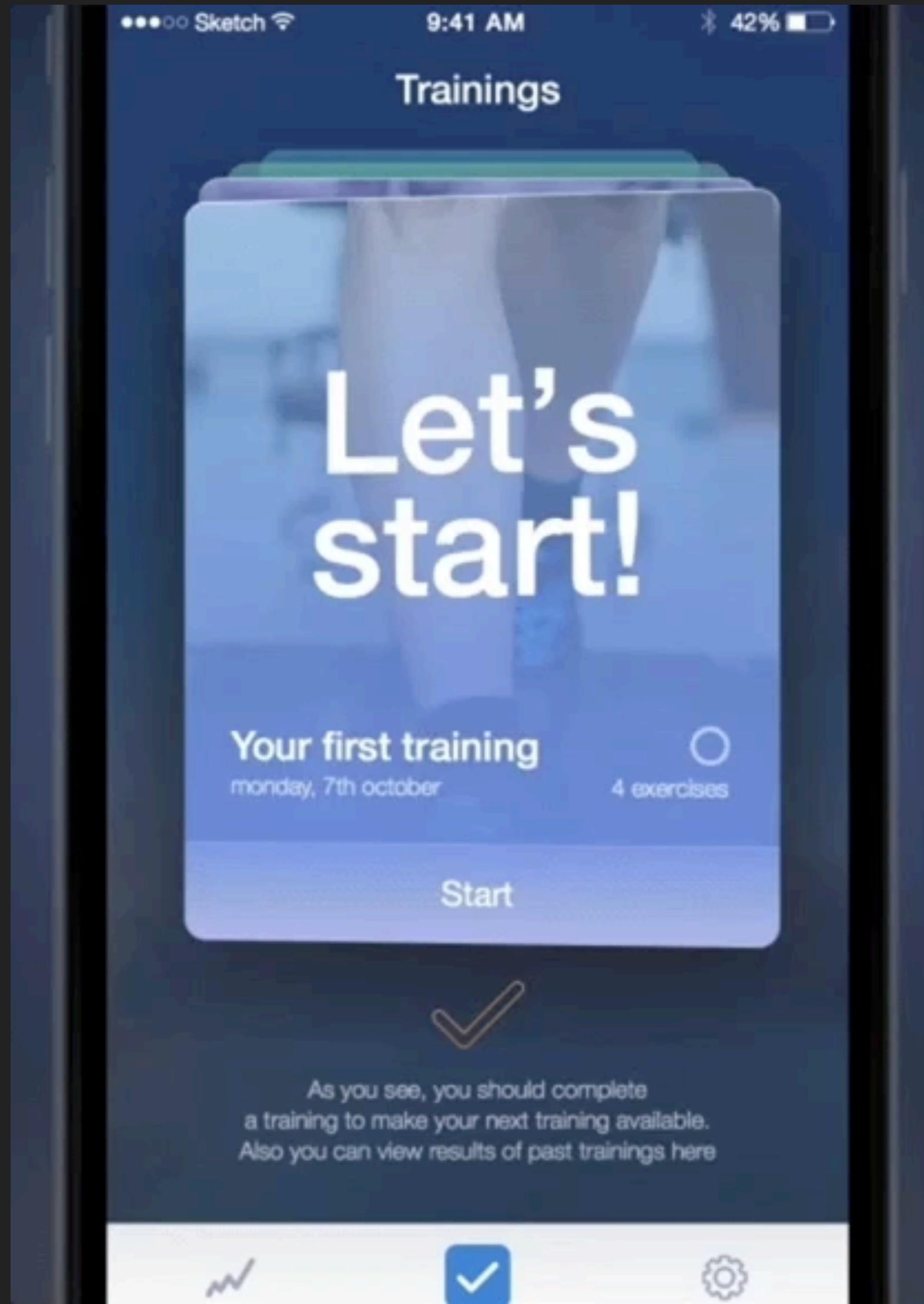
▶ Altere a propriedade Transition para Cross Dissolve



MODAIS



► Agora ela "aparece do nada"



- ▶ Tente adequar da melhor forma para o seu App (Flip Horizontal)



**MODAL COMO
COMPONENTE**

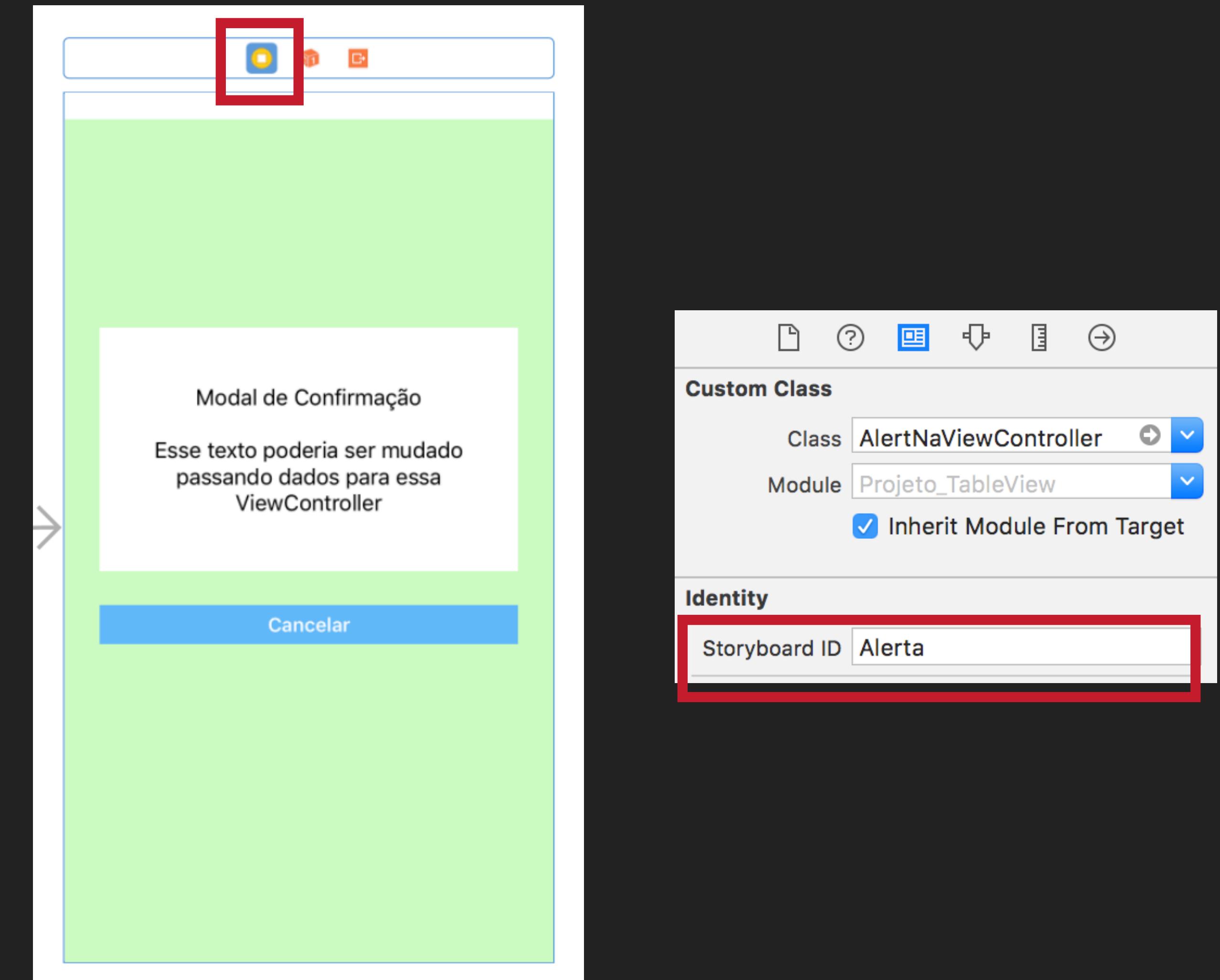
MODAL COMO COMPONENTE

- ▶ Seria bom chamar a Modal de qualquer lugar
 - 1. Podemos chamar de forma programática de qualquer lugar;
 - 2. Podemos passar informações para a próxima ViewController e assim preencher os textos da Modal.
 - 3. Para retornar informações (caso o usuário tenha clicado em outro botão de ação) podemos usar o padrão delegate.

MODAL COMO COMPONENTE

1. Chamar programaticamente de qualquer lugar:

- ▶ Selecione a ViewController da nossa Modal.
- ▶ Crie um Storyboard ID



MODAL COMO COMPONENTE

1. Chamar programaticamente de qualquer lugar:

- ▶ Nas ViewControllers que queremos apresentar, só executamos um **present** com o identifier
- ▶ Assim, já não precisamos mais da Segue

```
@IBAction func programaticoClicado(_ sender: Any) {  
    let storyboard = UIStoryboard(name: "Main", bundle: nil)  
    let alertViewController = storyboard.instantiateViewController(withIdentifier: "Alerta")  
    self.present(alertViewController, animated: true, completion: nil)  
}
```

MODAL COMO COMPONENTE

2. Podemos passar informações para a próxima ViewController e assim preencher os textos da Modal.

- ▶ Na modal, vamos criar uma Outlet para a Label.
- ▶ Criar uma property e preencher a Label com ela.

```
class ModalViewController: UIViewController {  
    @IBOutlet weak var alertLabel: UILabel!  
    var textAlert: String?  
  
    override func viewDidAppear(_ animated: Bool) {  
        super.viewDidAppear(animated)  
        alertLabel.text = textAlert  
    }  
}
```

MODAL COMO COMPONENTE

2. Podemos passar informações para a próxima ViewController e assim preencher os textos da Modal.

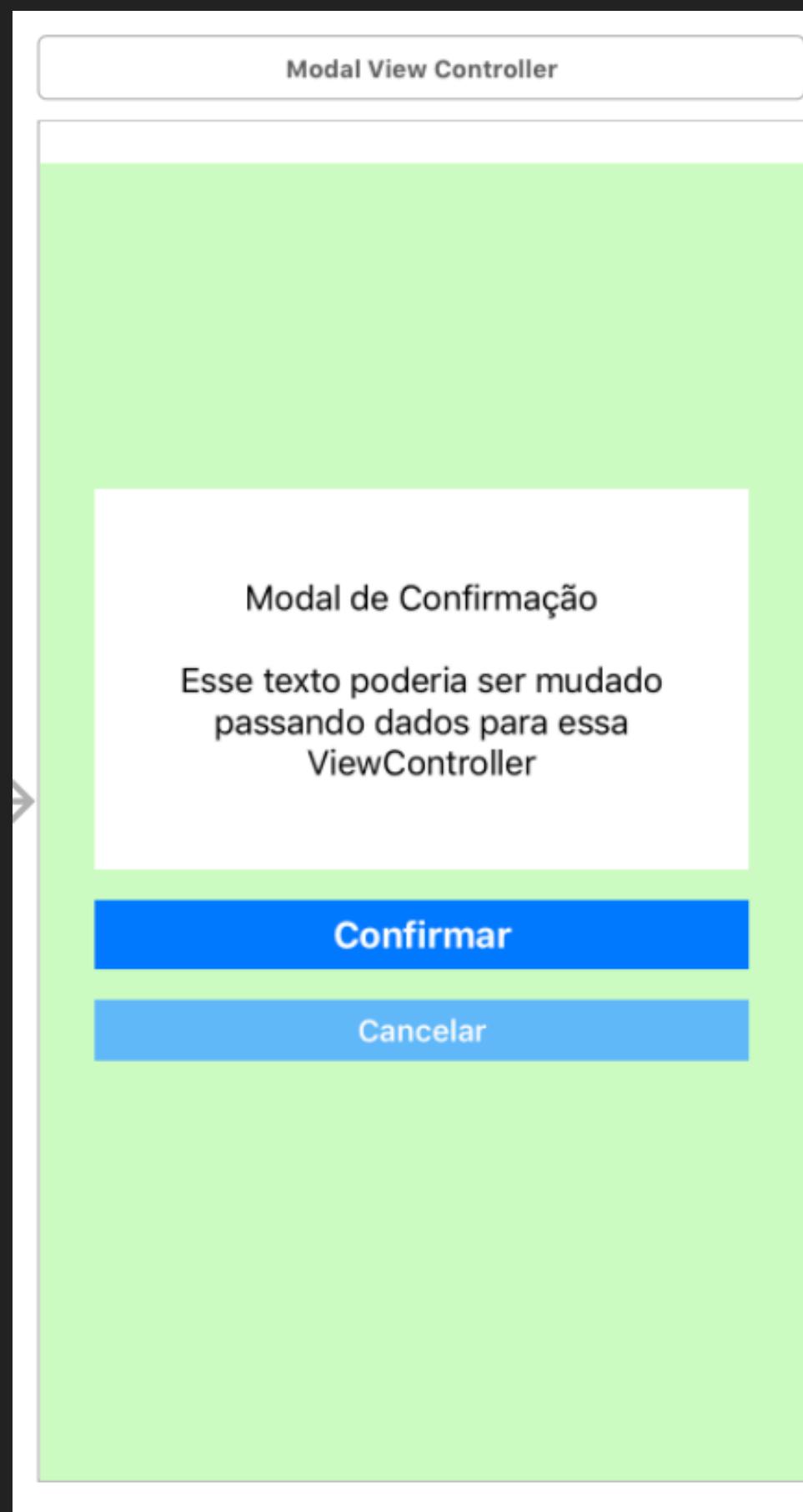
- ▶ Na tela que chama a Modal, vamos preencher a Outlet criada.

```
@IBAction func programaticoClicado(_ sender: Any) {
    let storyboard = UIStoryboard(name: "Main", bundle: nil)
    if let alertViewController = storyboard.instantiateViewController(withIdentifier: "Alerta") as? ModalViewController {
        alertViewController.textAlert = "Esse texto será exibido na Modal"
        self.present(alertViewController, animated: true, completion: nil)
    }
}
```

MODAL COMO COMPONENTE

3. Para retornar informações (caso o usuário tenha clicado em outro botão de ação) podemos usar o padrão delegate.

► Vamos criar um botão de Confirmar e uma IBAction.



```
class ModalViewController: UIViewController {
    @IBOutlet weak var alertLabel: UILabel!
    var textAlert: String?

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        alertLabel.text = textAlert
    }

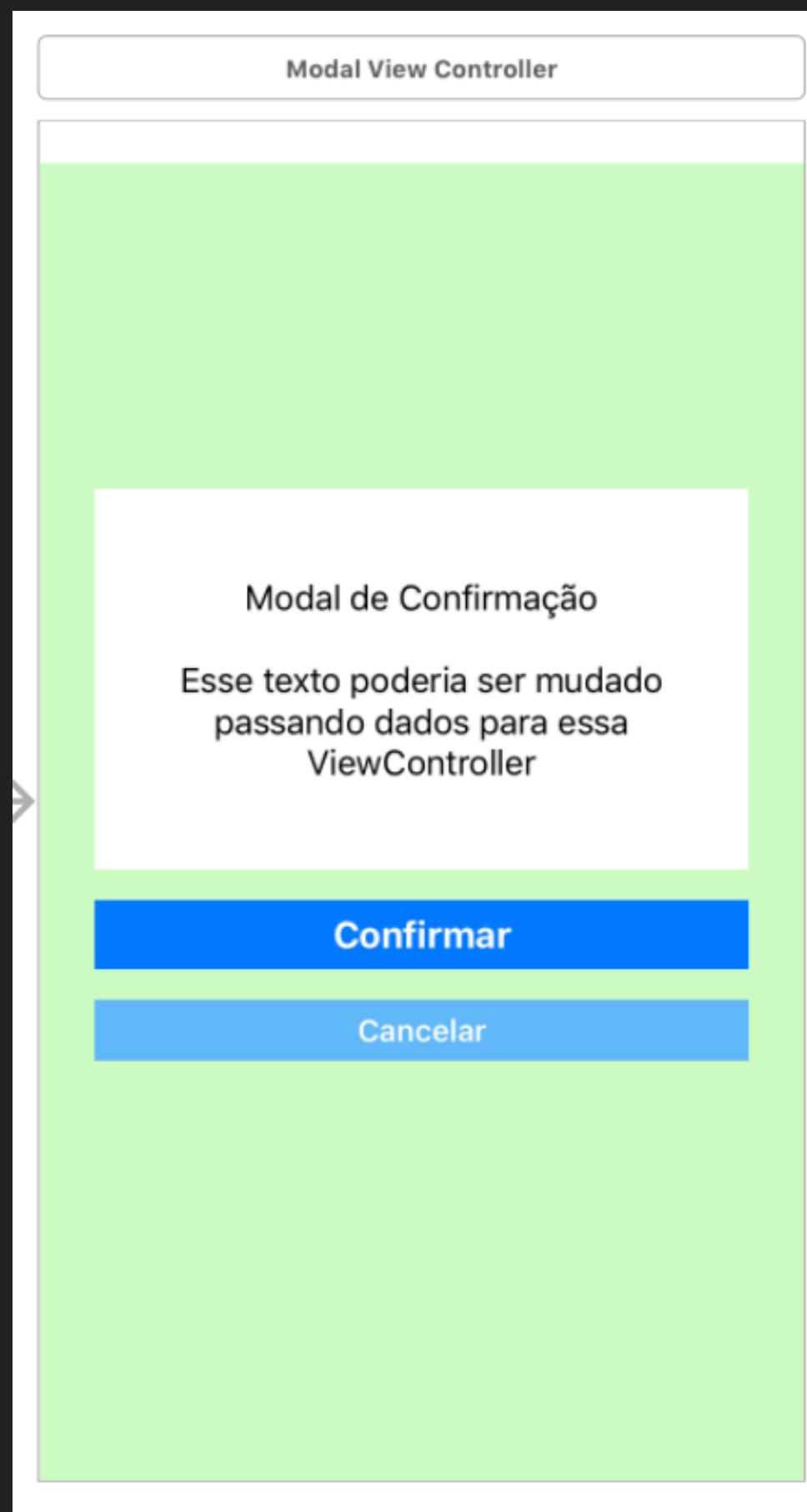
    @IBAction func confirmarClicado(_ sender: Any) {
    }

    @IBAction func cancelarClicado(_ sender: Any) {
        self.dismiss(animated: true, completion: nil)
    }
}
```

MODAL COMO COMPONENTE

3. Para retornar informações (caso o usuário tenha clicado em outro botão de ação) podemos usar o padrão delegate.

► Após isso, vamos implementar o padrão Delegate



```
protocol ModalViewControllerDelegate: AnyObject {
    func botaoConfirmarClicado()
}

class ModalViewController: UIViewController {
    @IBOutlet weak var alertLabel: UILabel!
    var textAlert: String?
    var delegate: ModalViewControllerDelegate?

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        alertLabel.text = textAlert
    }

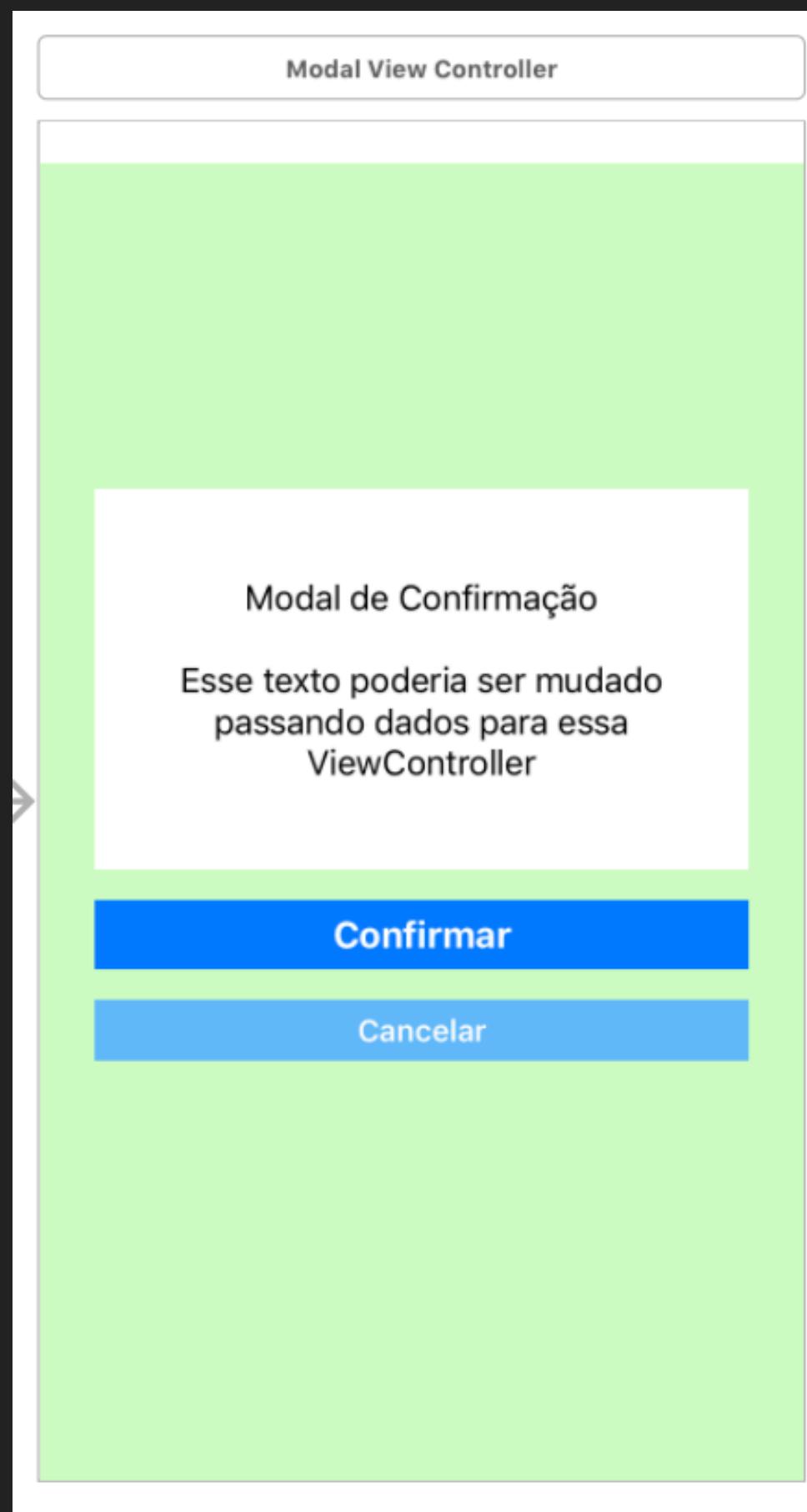
    @IBAction func confirmarClicado(_ sender: Any) {
        delegate?.botaoConfirmarClicado()
    }

    @IBAction func cancelarClicado(_ sender: Any) {
        self.dismiss(animated: true, completion: nil)
    }
}
```

MODAL COMO COMPONENTE

3. Para retornar informações (caso o usuário tenha clicado em outro botão de ação) podemos usar o padrão delegate.

► Antes de chamar o método do delegate, vamos executar um dismiss



```
protocol ModalViewControllerDelegate: AnyObject {
    func botaoConfirmarClicado()
}

class ModalViewController: UIViewController {
    @IBOutlet weak var alertLabel: UILabel!
    var textAlert: String?
    var delegate: ModalViewControllerDelegate?

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        alertLabel.text = textAlert
    }

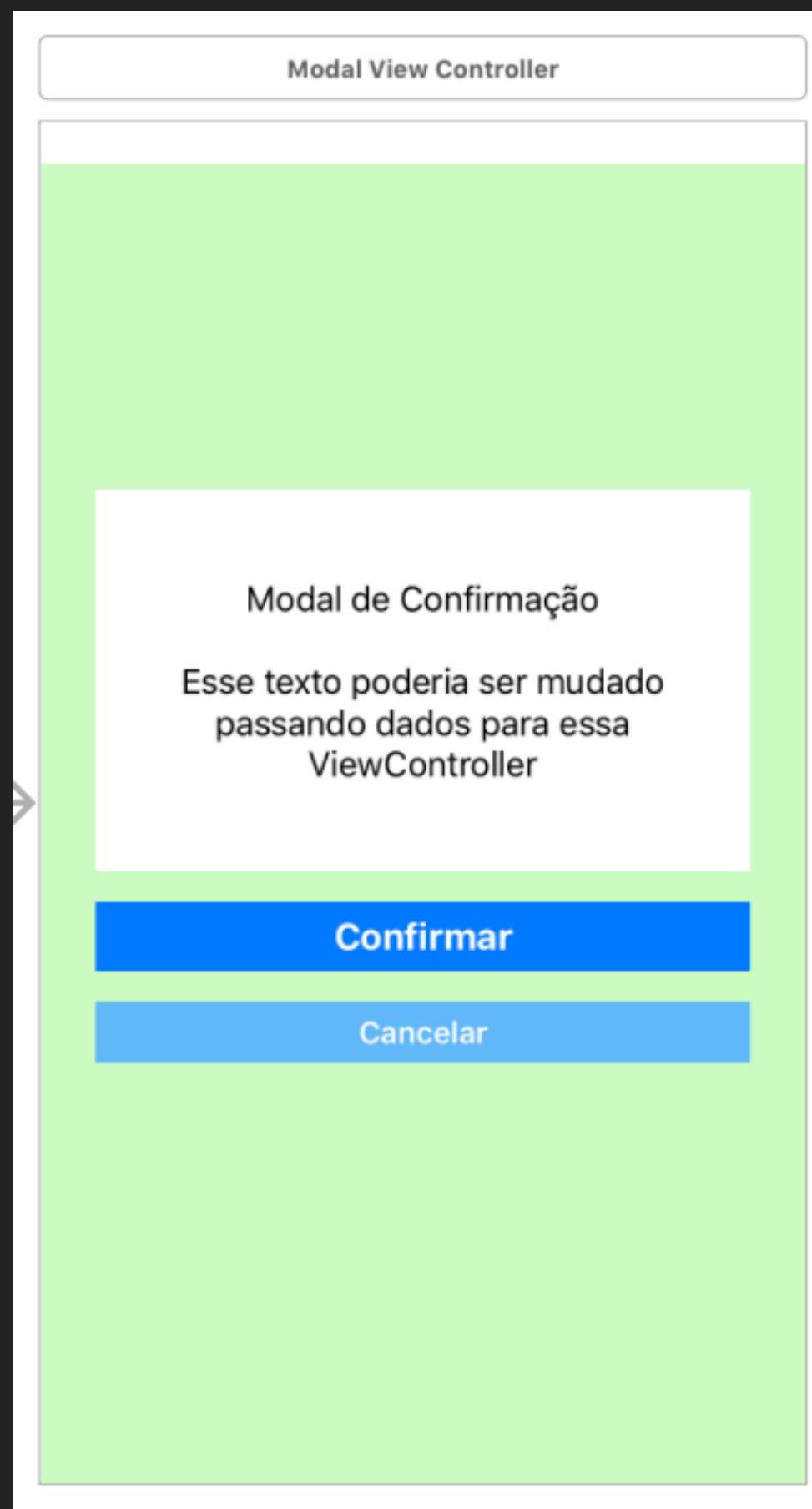
    @IBAction func confirmarClicado(_ sender: Any) {
        self.dismiss(animated: true, completion: nil)
        delegate?.botaoConfirmarClicado()
    }

    @IBAction func cancelarClicado(_ sender: Any) {
        self.dismiss(animated: true, completion: nil)
    }
}
```

MODAL COMO COMPONENTE

3. Para retornar informações (caso o usuário tenha clicado em outro botão de ação) podemos usar o padrão delegate.

- ▶ Com isso, nossa Modal funciona mesmo sem a tela de origem implementar o delegate
- ▶ Apesar de funcionar, a tela de origem não tem uma resposta quando o usuário clicar no Confirmar.
- ▶ Vamos implementar esse protocolo



MODAL COMO COMPONENTE

3. Para retornar informações (caso o usuário tenha clicado em outro botão de ação) podemos usar o padrão delegate.

► Na classe que irá chamar a modal (AlertNaViewController), vamos implementar o protocolo

```
class AlertNaViewController: UIViewController, ModalViewControllerDelegate {
```

```
@IBAction func programaticoClicado(_ sender: Any) {
    let storyboard = UIStoryboard(name: "Main", bundle: nil)
    if let alertViewController = storyboard.instantiateViewController(withIdentifier: "Alerta") as? ModalViewController {
        alertViewController.textAlert = "Esse texto será exibido na Modal"
        alertViewController.delegate = self
        self.present(alertViewController, animated: true, completion: nil)
    }
}
```

```
func botaoConfirmarClicado() {
    print("Ok")
}
```

MODAL COMO COMPONENTE

- COM ISSO, TEMOS UM COMPONENTE DE ALERTA COMPLETAMENTE CUSTOMIZADO.
- ESSE COMPONENTE FUNCIONA MESMO QUANDO O PROTOCOLO NÃO É IMPLEMENTADO.
- PODERÍAMOS ATÉ MESMO VERIFICAR SE O DELEGATE FOR NIL, ENTÃO ESCONDemos O BOTÃO CONFIRMAR