# Unidade 4

## Aula 1 - Gerenciando estados das ViewControllers

Para não ficar muito repetitivo e o curso de iOS não se tornar apenas uma **tradução de código Android** mostraremos algumas técnicas diferenciadas para melhorar a arquitetura do app.  
Começamos acima, nas primeiras aulas, introduzindo a ideia de ResponseModel e ViewModels com seus respectivos parsings em **structs**. Agora mostraremos um conceito de usabilidade bastante importante nos aplicativos que é o **feedback ao usuário**.

O usuário precisa saber o que está acontecendo no app. Quando algum erro acontece ele precisa ser informado disso e se possível receber uma instrução sobre como contorná-lo. Quando uma tarefa muito demorada ocorre, como por exemplo uma chamada de serviço, o usuário precisa saber que algo está carregando e não apenas ficar preso em uma tela, dando a impressão que o app travou.

Sabendo disso, todoas as nossas ViewControllers apresentarão os três seguintes estados:

* **Erro**: uma mensagem de erro será apresentada quando algum erro oriundo das chamadas de serviço ocorrer
* **Carregando**: uma Activity Indicator View aparecerá no centro da tela indicando que algo está carregando.
* **Sucesso**: o conteúdo principal da View Controller já está sendo exibido.

### Atividade

Encontre uma maneira reutilizável e minimamente funcional para realizarmos a transição entre estes três estados. Não é necessário testá-los totalmente, mas apenas mostrar que isto é uma boa idéia. Uma boa dica é fazer o uso de UIStackView, mas você pode procurar outras soluções

### Solução

A exibição e organização destes três estados pode ser alcançado através de uma **UIStackView**.

#### UIStackView

Introduzida recentemente no iOS 9 as Stack Views vieram para facilitar a construção de Views que demandam auto-layout.

A maioria das restrições que definimos via auto-layout podem ser feitas utilizando Stack Views, de uma forma mais simples e organizada.

As Stack Views se assemelham ao LinearLayout do Android. Ela basicamente organiza as suas subviews de forma que uma fique abaixo da outra (caso queiramos um empilhamento vertical) ou uma ao lado da outra (caso enfileiramento horizontal).

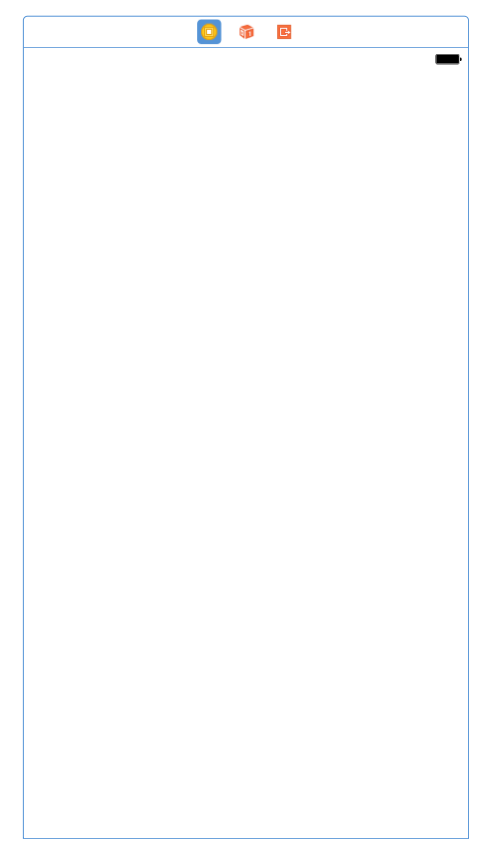
#### Criando nosso View Controller básico

Nossos View Controllers terão o tema Dark, seguindo o padrão utilizado no Android. Então todas as nossas telas terão que obedecer este tema.

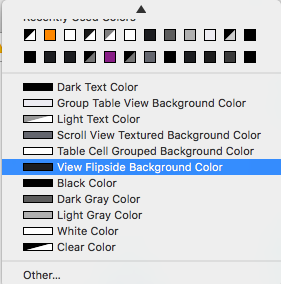
##### Criando o Visual

Vamos construir o funcionamento base que todas as nossos View Controllers terão. Para isto siga os seguintes passos:

1- Adicione uma View Controller no storyboard **Main**:



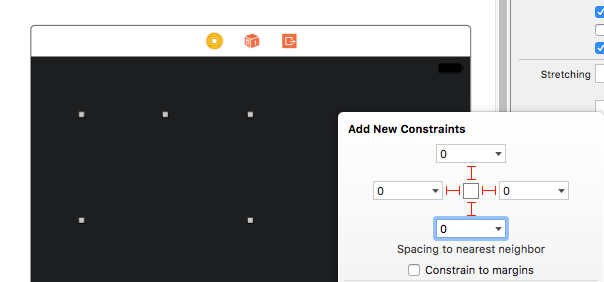
2 - Configure a **Background Color** da view raiz para: **View Flipside Background Color** :



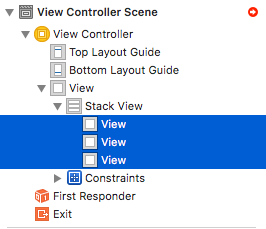
3 - Nesta View Controller adicione uma **UIStackView** vertical:



4 - Adicione as restrições de borda de modo que as bordas da Stack View sejam aderidas as bordas da view raiz (valor 0 em todas as bordas da stack):

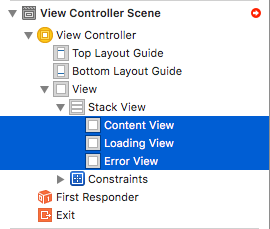


5 - Agora adicione três Views dentro da Stack View:



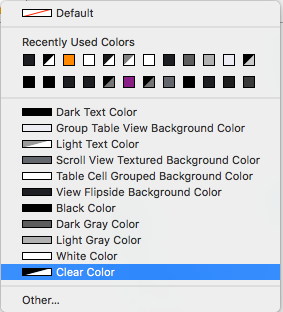
6 - Nomeie-as com os seguintes nomes:

* Content View
* Loading View
* Error View

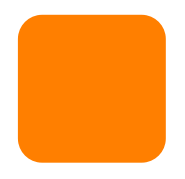


Obs: A ordem das subviews dentro de uma Stack View influencia na sua posição. No nosso caso, iremos exibir apenas uma subview desta Stack por vez, então a ordem que as subviews estiverem não influenciará. Para nível de organização, mantenha a ordem listada acima.

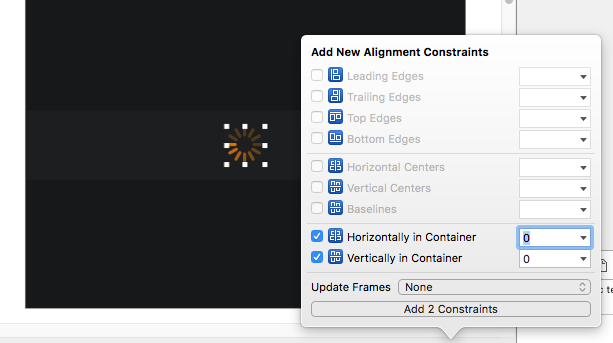
7- Selecione as três Views e defina seu **Background Color** para **Clear Color**:



8 - Vamos começar a preencher as Views com os componentes que serão comuns em todas as Views Controller. Para isto selecione a Loading View e adicione um **Activity Indicator View** como subview.  
9 - Altere seu estilo para **Large White** e defina a seguinte **Color** para o indicator: #FF7F00.

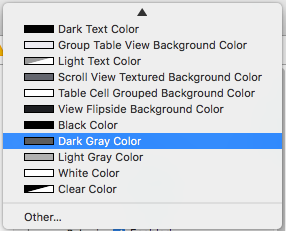
  
Obs.: Esta cor é um laranja amarelado, bem parecido ao amarelo que utilizávamos no app de Android.

10 - Adicione as seguintes restrições para o Activity Indicator View se centralizar na Loading View:

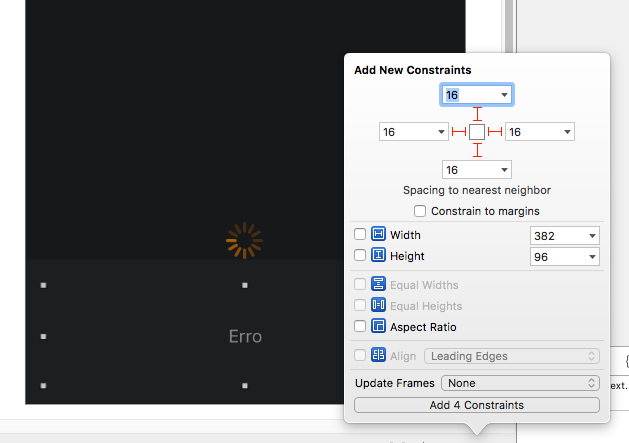


11 - Pronto, vamos adicionar a mensagem de erro na Error View. Adicione uma UILabel como subview da Error View.

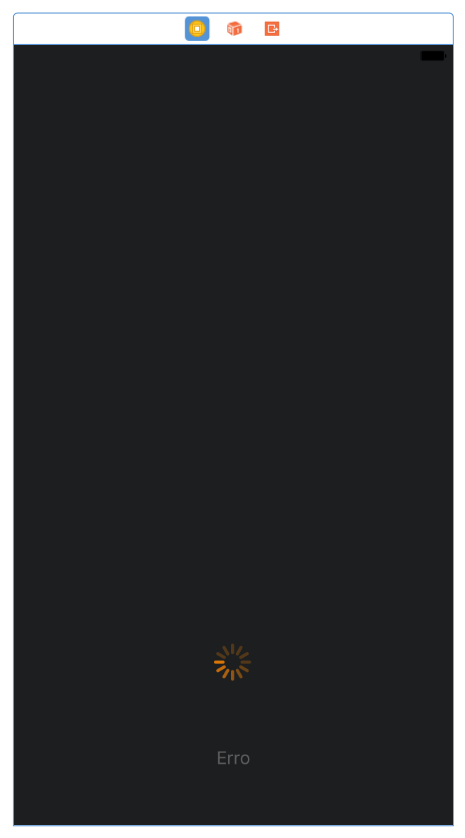
12 - Configure a **Color** da UILabel para **Dark Gray Color**:



13 - Altere o **Text** da label para **Error**  
14 - Adicione as seguintes restrições de bordas na Label:



Nossa View Controller base ficará da seguinte forma:



Terminamos o Layout da nossa View Controller Base.

#### Implementando a troca de estados das View Controllers

Todas as View Controllers que implementarmos daqui em diante, precisaremos utilizar o mesmo layout da View Controller base que utilizamos. Agora vamos implementar a troca de estados para as nossas View Controllers.

Devemos criar os outlets para as três subviews da stack e relacioná-los. Para este exemplo iremos utilizar a tela de detalhes de um Filme, que pode ser vista na imagem a seguir:



1. Crie uma nova classe que estenda de UIViewController e dê o nome **MovieDetailsViewController** para ela.
2. Insira os seguintes outlets:

@IBOutlet weak var contentView: UIView!  
 @IBOutlet weak var loadingView: UIView!  
 @IBOutlet weak var errorView: UIView!  
 @IBOutlet weak var errorLabel: UILabel!

1. Relacione as três views e a label de erro com os outlets criados
2. Vamos criar um método geral para atualizar as Views:

func updateViews() {  
  
}

1. Crie uma propriedade Content<Any> como variável computada inicializando-o com o valor .loading

var content: Content<Any> = .loading {  
  
}

Obs.: Iremos mudar o tipo de objeto de retorno de sucesso do Content futuramente.

1. Insira a clausula didSet que invoca o método updateViews():

var content: Content<Any> = .loading {  
 didSet {  
 updateViews()  
 }  
}

1. No método updateViews() faça o gerenciamento dos estados da View Controller:

func updateViews() {  
 switch content {  
 case .success(let data):  
 errorView.isHidden = true  
 loadingView.isHidden = true  
 contentView.isHidden = false  
 case .loading:  
 errorView.isHidden = true  
 loadingView.isHidden = false  
 contentView.isHidden = true  
 case .error(let error):  
 errorView.isHidden = false  
 loadingView.isHidden = true  
 contentView.isHidden = true  
   
 if case .userMessage(let message) = error {  
 errorLabel.text = message  
 }  
 }  
 }

Este é o comportamento básico para gerenciar os estados das telas.

## Aula 2 - Enum Content

Voltando ao assunto sobre os estados das ViewControllers, precisamos de uma maneira de controlá-los. Vimos na aula passada que podemos dividir nossas camadas MVC e fomos apresentados a camada Interactor.

De certa forma o Interactor precisará mandar dados para a camada View, mas além de precisarmos os dados principais da aplicação (como a lista de gêneros, filmes e etc), precisaremos informar qual estado da requisição para que a ViewController apresente um feedback consistente ao usuário.

Os três possíveis casos são:

* Loading (carregando): a chamada de serviço ainda está sendo processada e por enquanto não retornou nenhum resultado.
* Error (erro): a chamada de serviço já foi processada mas algum erro ocorreu.
* Success (sucesso): a chamada de serviço já foi processada e os dados que estávamos esperando foram retornados no resultado da requisição.

### Atividade

Crie um modelo/enum baseado no enum Result que criamos para controlarmos os estados das views. Este enum deverá conter os três casos mencionados acima.

### Solução

Com o Swift podemos criar enums bastante elaborados e úteis para diversas atividades. Um enum que criaremos se chamará Content.

Este enum carregará os ViewModels das telas e também será utilizado para gerenciar o estado da View Controller (Sucesso, erro e carregando).

Com o Swift podemos colocar parâmetros e injetar objetos nos cases de nossos enums, então teriamos uma estrutura assim para o enum Content.

enum Content<T> {  
 case success(T)  
 case error(ReturnError)  
 case loading  
}

1. Copie o código acima e cole em um arquivo separado com o nome **Content.swift**

Como você pode ver, estamos utilizando um **Generic** indicado pelo simbolo <T>. Isto significa que podemos inserir qualquer dado no **case success** pois este suporta um dado Genérico.

No **case error** estamos passando via parâmetro um ReturnError para caso queiramos exibir um erro na tela.

O enum Content será utilizado nas View Controllers, mas será gerado pelos nossos interactors, então o pacote de comunicação Interactor -> View Controller é um Content. Sendo assim, nossa arquitetura fica assim:

Result Content   
Service <-----------------------> Interactor <-----------------------> ViewController  
 ResponseModel ViewModel

A forma de usar o Content no View Controller é simples:

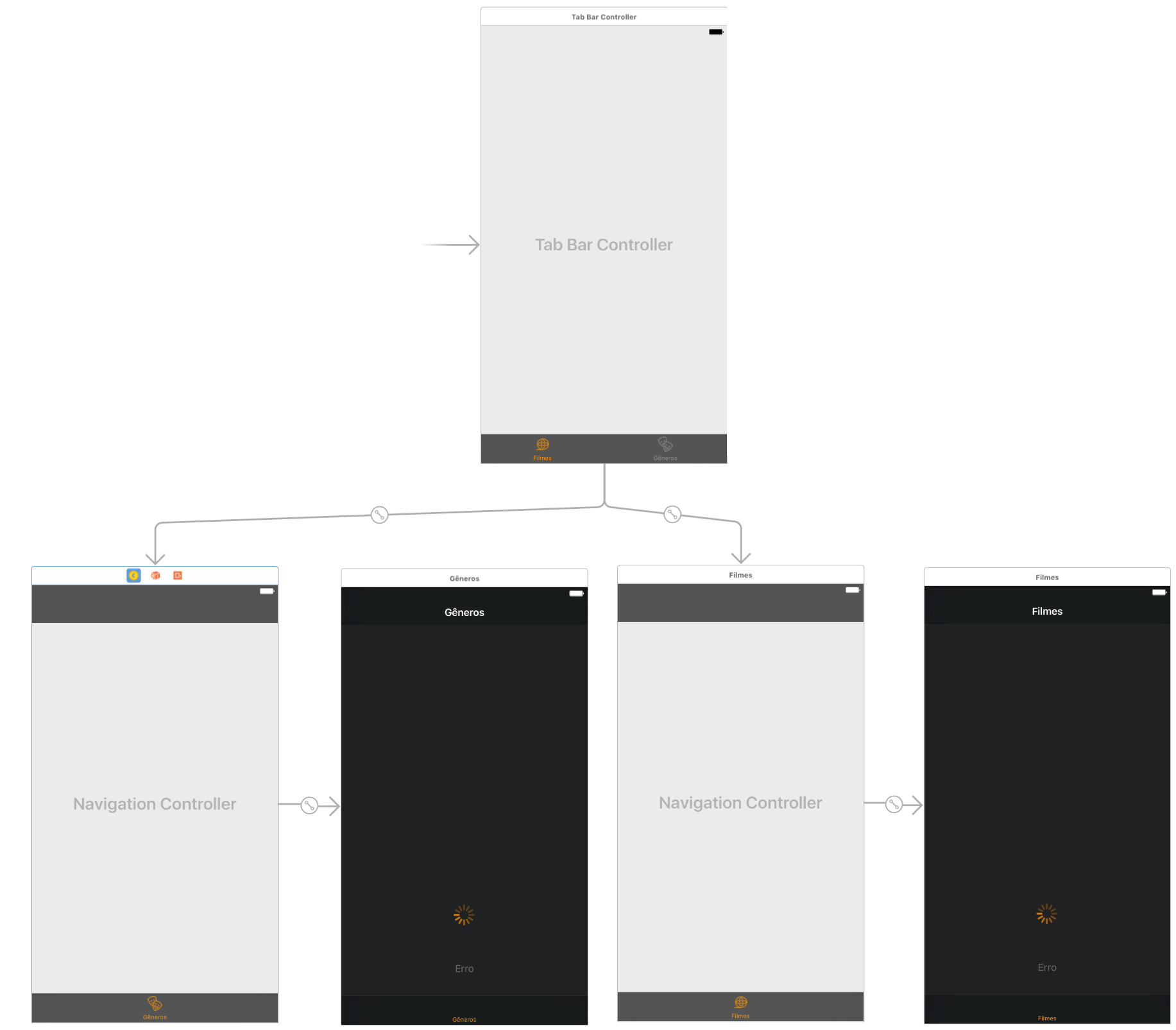
Primeiro definimos uma variável do tipo desejado. No exemplo queremos um objeto Any como retorno de sucesso, então teríamos que escrever o seguinte:

var content: Content<Any> = .loading {  
 didSet {  
 // atualiza as views  
 }  
}

Como o intuito deste enum é gerenciar os estados das Views, começamos com o estado loading. Iremos utilizar isto em todos os nossos View Controllers

## Aula 3 - Navegação (TabViewController)

Nesta atividade vamos implementar a navegação geral do aplicativo. Ela será composta por uma Tab Bar Controller. As View Controllers desta Tab Bar Controller serão Navigation Controllers. Veja abaixo como deverá ficar o fluxo no storyboard quando finalizarmos esta atividade:



Esta Tab Bar Controller deverá conter duas abas, uma com a cena Filmes e a outra Gêneros. Utilize os assets fornecidos na unidade 1 para inserir o ícone de cada aba.

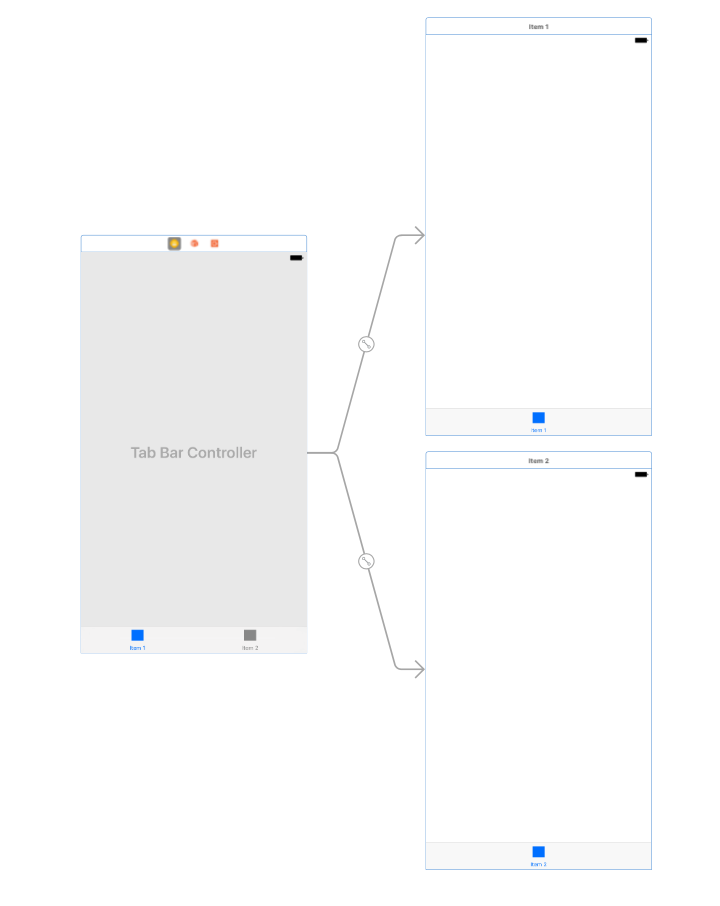
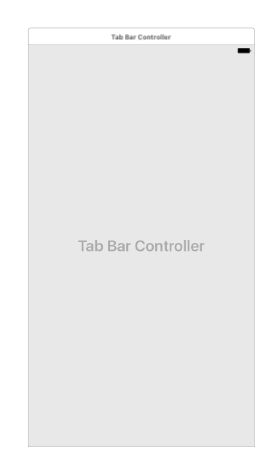
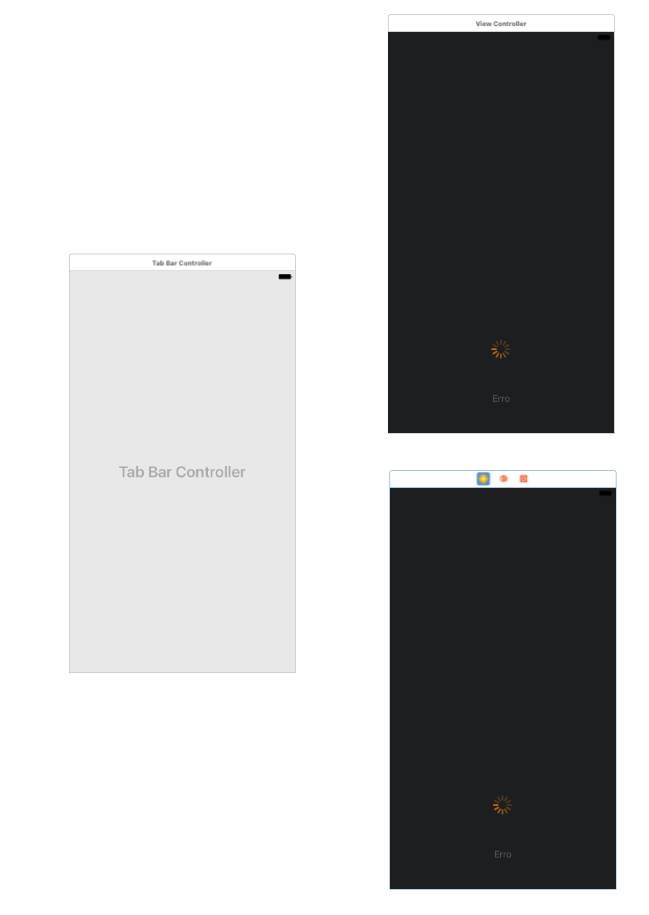
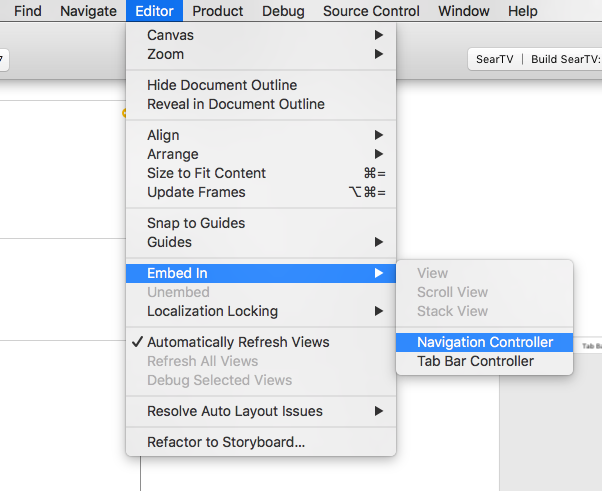
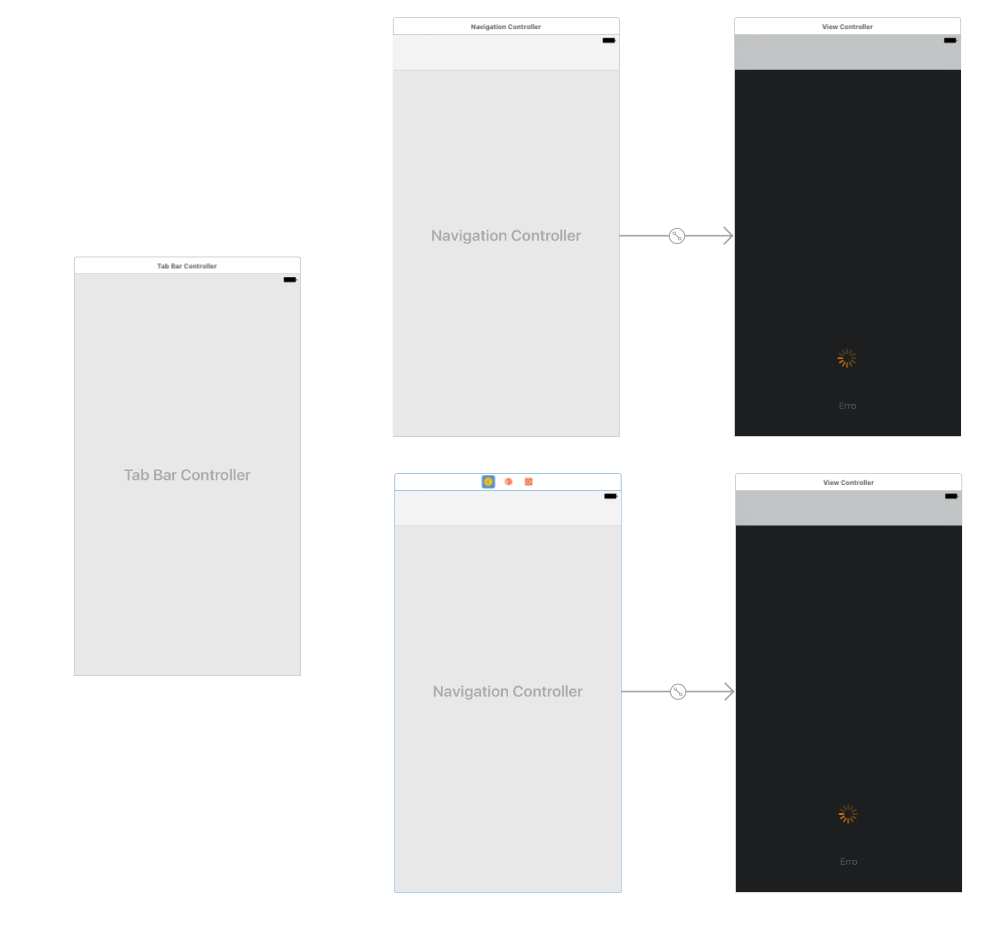
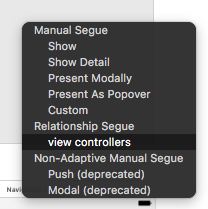
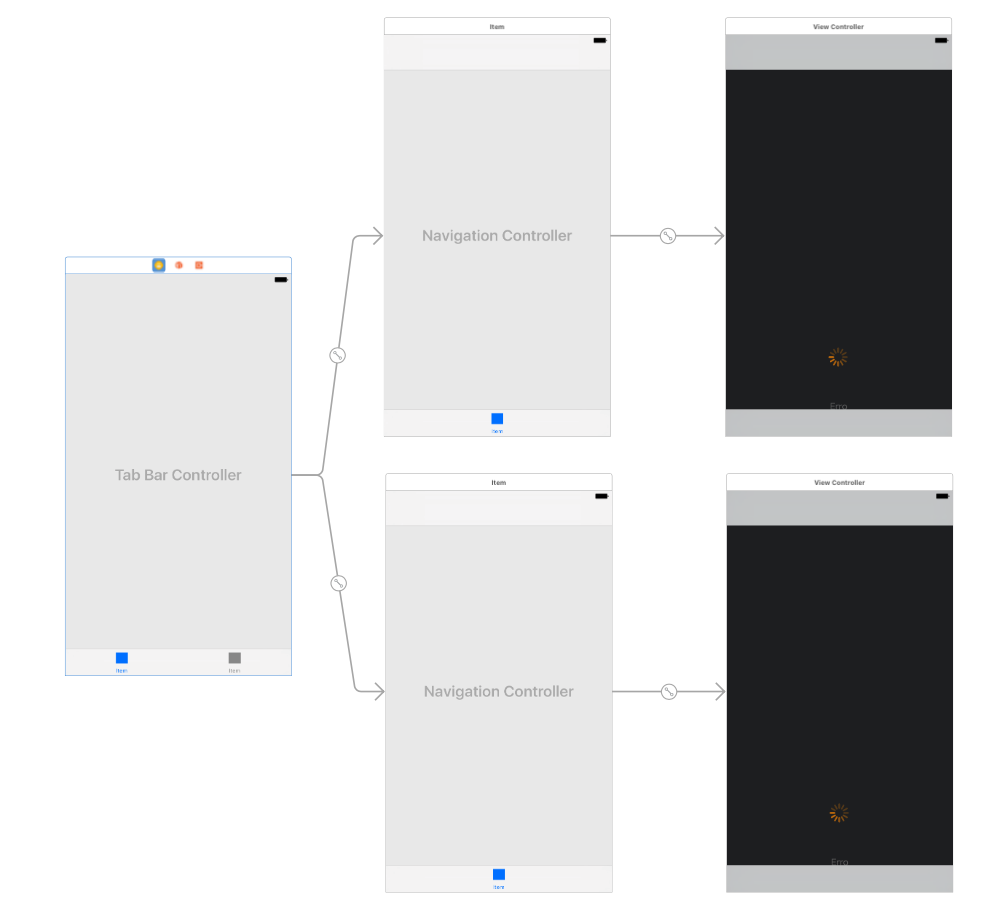
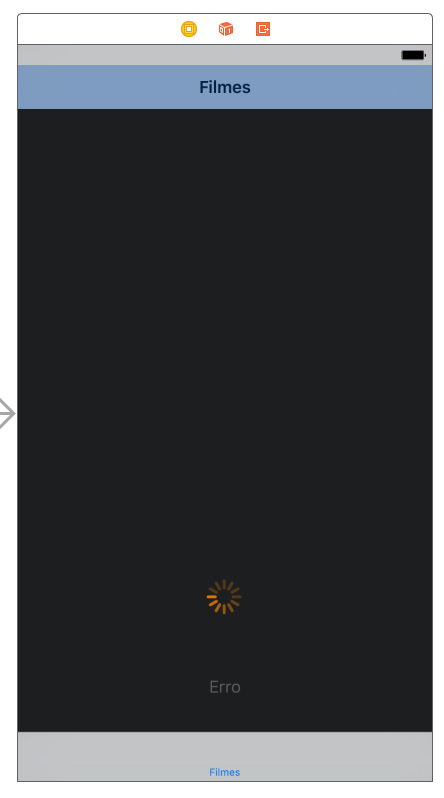
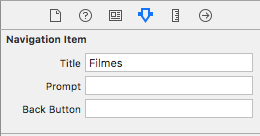
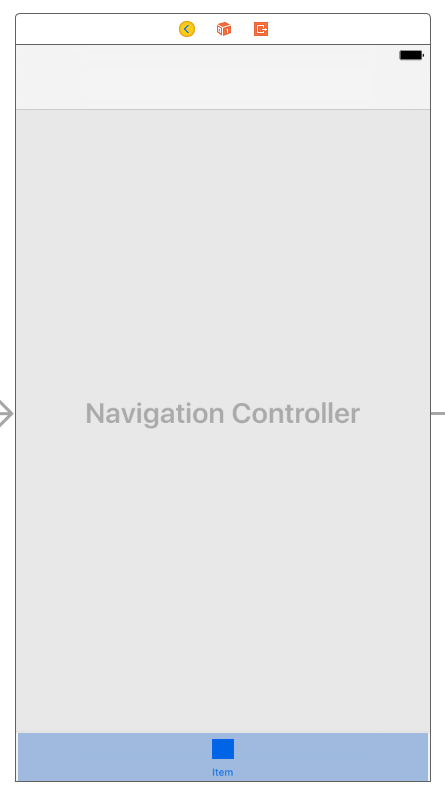
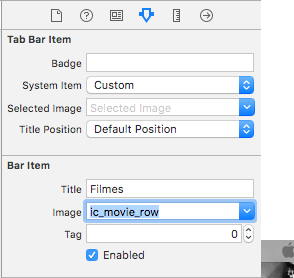
Você também poderá utilizar a View Controller base que criamos na aula 1 desta unidade para representar a cena.

### Atividade

Implemente a navegação do app em abas de acordo com as instruções acima

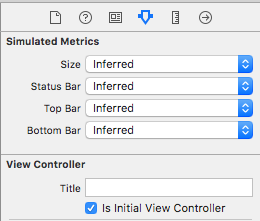
### Solução

Para criar uma navegação em abas, basta consultar o exemplo dado em aula. Vamos ao passo-a-passo:

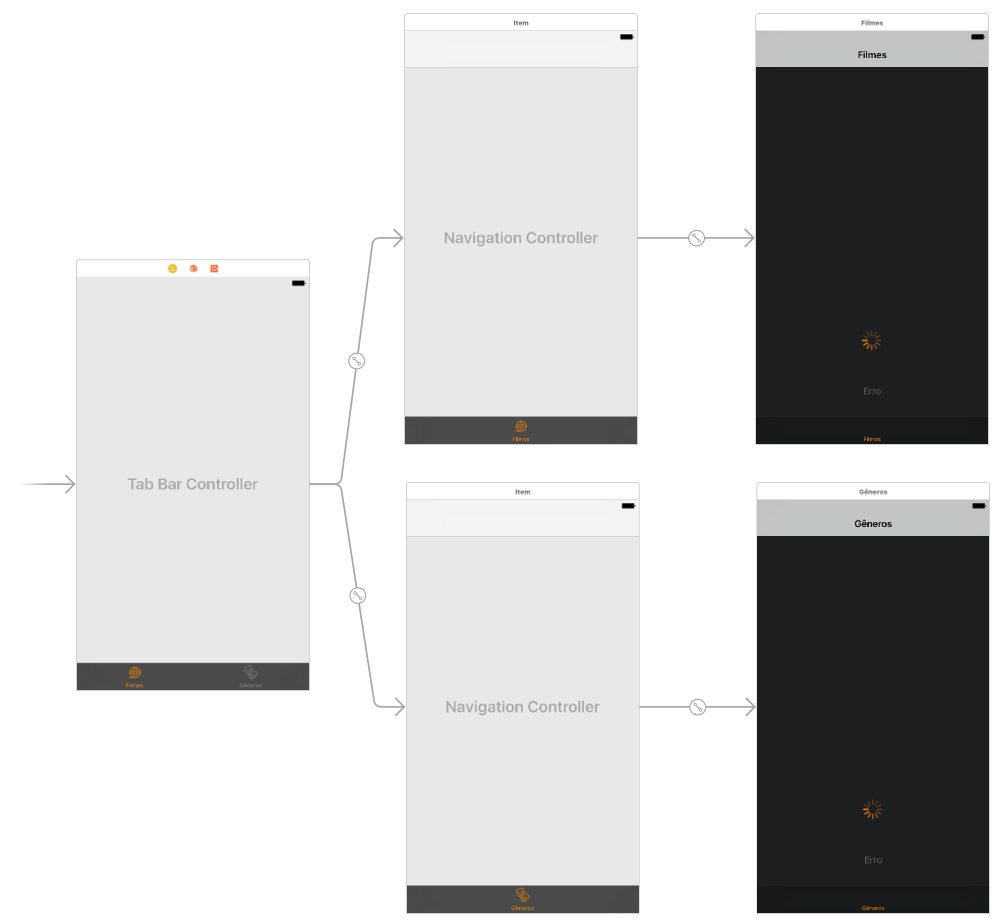
1. Adicione uma UITabBarController no **Main.storyboard**:  
   
2. Delete as duas View Controllers adjacentes  
   
3. Copie e cole duas View Controllers base que criamos na aula 1 desta unidade  
   
4. Selecione uma das View Controllers coladas e no menu superior clique em **Editor > Embed In > Navigation Controller**:  
   
5. Faça o mesmo para a outra View Controller colada:  
   
6. Segure o botão **Ctrl**, então clique no centro da Tab Bar Controller depois arraste e solte em uma das Navigation Controllers
7. Selecione a opção **view controllers**  
   
8. Repita os dois passos anteriores para a outra Navigation Controller  
   
9. Selecione a primeira a View Controller raiz da primeira Navigation Controller que você relacionou com Tab Bar. (No exemplo a primeira Navigation Controller foi a que está em cima)
10. Na aba **Show Atributtes Inspector** escreva **Filmes** no campo **Title**
11. Faça o mesmo para a outra View Controller mas desta vez escreva **Gêneros** ao invés de Filmes.
12. Agora volte na outra View Controller e selecione a barra superior  
    
13. No campo **Title** escreva **Filmes**  
    
14. Selecione a barra inferior  
    
15. No campo **Title** escreva **Filmes** e no campo **Image** escreva **ic\_movie\_row**  
    
16. Repita os passos para a outra View Controller com a diferença que:

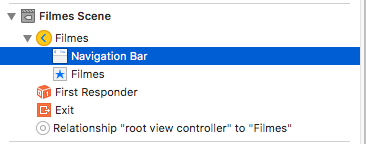
* No campo **Title** da barra superior digite **Gêneros**
* No campo **Title** da barra inferior digite **Gêneros** também
* No campo **Image** da barra inferior digite **ic\_genre\_unselected**

Fizemos os procedimentos mínimos para uma navegação em abas. Agora vamos personalizá-la

1. Selecione a Tab Bar Controller e clique em **Is Initial View Controller**  
   
2. Agora selecione sua Tab Bar
3. Configure as seguintes propriedades da Tab Bar  
   **Tab Bar**  
    - Image Tint: #FF7F00  
    - Style: Black  
   **View**  
    - Background: Clear Color  
    - Tint: #FF7F00

Por enquanto teremos esta aparência:



1. Selecione uma Navigation Controller
2. Selecione sua Navigation Bar  
   
3. Configure as seguintes propriedades das Navigation Controllers  
   **Navigation Bar**  
    - Style: Black  
   **View**  
    Tint: #FF7F00
4. Faça o mesmo para a outra Navigation Controller

Pronto. A navegação base do nosso app está terminada. Você pode executar a aplicação e ver o resultado.