# Unidade 7

## Aula 1 - Integrando Gêneros

Agora que criamos a camada View (com os View Controllers e o Storyboard) e a camada Service, precisamos interligá-las. Precisamos de algo que possibilite que a View Controller **interaja** com a camada Service, e este alguém chamaremos de Interactor.

Recordando o nosso estudo sobre arquiteturas, a utilização do VIP consiste em três principais camadas a View, a Interactor e a Presenter. Como dissemos, a Presenter tem a função de formatar os dados provenientes da camada Interactor, mas como nossos dados já vêm quase prontos para serem exibidos das classes Service podemos eliminar o presenter e fazer tudo no Interactor. Como nosso app é pequeno, isto não influenciará muito, mas você pode criar a camada Presenter se quiser.

### Atividade

1. Implemente o Interactor da cena Gêneros.
2. Integre a ViewController com o Interactor de forma que possamos exibir definitivamente a lista de Gêneros.

### Solução

Começaremos criando o interactor

#### Implementando o Interactor

1. Crie um novo arquivo chamado **GenresInteractor.swift**
2. Neste arquivo crie a classe **GenresInteractor**:

class GenresInteractor {  
}

Lembra do padrão Singleton que sempre estamos falando? Nosso interactor será um Singleton, pois ele precisará guardar uma lista de gêneros.

Como é raro surgirem novos gêneros, podemos confiar que a lista de gêneros obtida na chamada de serviço não mudará com muita frequência e poderemos armazená-la em um array para reutilizá-la e economizar requisições de chamadas de serviço.

1. Para criar um singleton em Swift basta adicionar o seguinte código na classe:

static var shared = GenresInteractor()  
private init() {}

1. E a lista de gêneros será armazenada na seguinte propriedade:

var list: [Genre.ResponseModel]?

1. Por fim adicionamos o método que fará a chamada do método da camada Service:

func getGenres(completion: @escaping (Content<[Genre.ViewModel]>) -> Void) {  
   
 // Se a lista não for nula...  
 if let genres = list {  
   
 // Reutiliza a lista salva  
 let viewModels = genres.map {  
 Genre.ViewModel(name: $0.name ?? "")  
 }  
   
 completion(Content.success(viewModels))  
   
 } else {  
   
 // Realiza uma nova chamada de serviço  
 GenreService.getGenres() { result in  
 switch result {  
 case .success(let genres):  
   
 // Atualiza a lista  
 self.list = genres  
   
 // Converte os Genre.ResponseModels em Genre.ViewModel  
 // Se tivessemos um Presenter, ele faria o trecho abaixo  
 let viewModels = genres.map {  
 Genre.ViewModel(name: $0.name ?? "")  
 }  
   
 completion(Content.success(viewModels))  
 case .failure(let error):  
 completion(Content.error(error))  
 }  
 }  
 }  
 }

Nosso Interactor está implementado. Basta adicionar o restante do código na ViewController

#### Finalizando a View Controller

Por fim precisaremos chamar o método que criamos no Interactor no nosso View Controller

1. Abra o arquivo GenresViewController e localize o método viewWillAppear().
2. No final deste método adicione o seguinte código:

GenresInteractor.shared.getGenres() { content in  
 self.content = content  
 }

E isto bastará para que tudo esteja encaixado e a cena **Gênero** esteja finalizada.

Execute a aplicação e veja a tela de Gêneros. Ela irá obter a lista de gênero utilizando uma chamada Rest. Saia da tela gênero e volte. Você verá que a requisição foi muito mais rápida pois a lista está sendo reutilizada.

## Aula 2 - Integrando Detalhes do Filme

Seguindo os moldes da aula passada, precisamos agora fazer o Interactor da cena Detalhes do Filme.

Na unidade 3 criamos o layout para esta cena, e ainda não havíamos aprendido sobre os estados das Views, então ainda precisamos implementar este View Controller base para a cena.

### Atividade

1. Implemente a View Controller para Detalhes do Filme.
2. Implemente o Interactor da cena Detalhes de filmes seguindo os models do Interactor da cena Gêneros

#### Solução

Para a a criação da View Controller de Detalhes de Filmes siga os seguintes passos:

1. Crie um novo arquivo chamado **MovieDetailsViewController.swift**.
2. Neste arquivo crie nova classe com o nome **MovieDetailsViewController** que estenda a classe **UIViewController**.

class MovieDetailsViewController: UIViewController {  
}

1. Insira o conteúdo básico da View Controller:

class MovieDetailsViewController: UIViewController {  
   
 var content: Content<MovieDetails.ViewModel> = .loading {  
 didSet {  
 updateViews()  
 }  
 }  
   
 @IBOutlet weak var contentView: UIView!  
 @IBOutlet weak var loadingView: UIView!  
 @IBOutlet weak var errorView: UIView!  
 @IBOutlet weak var errorLabel: UILabel!  
   
 override func viewWillAppear(\_ animated: Bool) {  
 updateViews()  
 }  
   
 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  
 }  
 }  
 }  
}

1. Não se esqueça de elencar todos os @IBOutlets com as respectivas Views como fizemos nas demais cenas.
2. Esta cena possui muitas propriedades a serem exibidas na tela. Adicione-as e elenque com as respectivas Views:

@IBOutlet weak var backDropImageView: UIImageView!  
 @IBOutlet weak var titleLabel: UILabel!  
 @IBOutlet weak var yearLabel: UILabel!  
 @IBOutlet weak var genreLabel: UILabel!  
 @IBOutlet weak var runtimeLabel: UILabel!  
 @IBOutlet weak var overviewLabel: UILabel!  
 @IBOutlet weak var originalTitleLabel: UILabel!  
 @IBOutlet weak var budgetLabel: UILabel!  
 @IBOutlet weak var revenueLabel: UILabel!  
   
 @IBOutlet weak var star1ImageView: UIImageView!  
 @IBOutlet weak var star2ImageView: UIImageView!  
 @IBOutlet weak var star3ImageView: UIImageView!  
 @IBOutlet weak var star4ImageView: UIImageView!  
 @IBOutlet weak var star5ImageView: UIImageView!

1. Crie alguns métodos auxiliares para atualizar os dados contidos nas labels e as imagens:

func updateLabels(viewModel: MovieDetails.ViewModel) {  
 titleLabel.text = viewModel.title  
 yearLabel.text = viewModel.releaseDate  
 genreLabel.text = viewModel.genre  
 runtimeLabel.text = viewModel.runtime  
 overviewLabel.text = viewModel.overview  
 originalTitleLabel.text = viewModel.originalTitle  
 revenueLabel.text = viewModel.revenue  
 budgetLabel.text = viewModel.budget  
 }  
   
 func updateImage(path: String) {  
 let url = "https://image.tmdb.org/t/p/w500/" + path  
   
 backDropImageView.kf.indicatorType = .activity  
   
 backDropImageView.kf.setImage(with: URL(string: url))  
 }  
   
 func updateStars(voteAverage: Int) {  
   
 star1ImageView.image = #imageLiteral(resourceName: "ic\_star\_off")  
 star2ImageView.image = #imageLiteral(resourceName: "ic\_star\_off")  
 star3ImageView.image = #imageLiteral(resourceName: "ic\_star\_off")  
 star4ImageView.image = #imageLiteral(resourceName: "ic\_star\_off")  
 star5ImageView.image = #imageLiteral(resourceName: "ic\_star\_off")  
   
 switch voteAverage {  
 case 1:  
 star1ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 case 2:  
 star1ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 star2ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 case 3:  
 star1ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 star2ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 star3ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 case 4:  
 star1ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 star2ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 star3ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 star4ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 case 5:  
 star1ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 star2ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 star3ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 star4ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 star5ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 default: break  
 }  
 }  
}

1. Por fim chame estes métodos no updateViews():

func updateViews() {  
 switch content {  
 case .success(let data):  
 errorView.isHidden = true  
 loadingView.isHidden = true  
 contentView.isHidden = false  
   
 updateLabels(viewModel: data)  
 updateStars(voteAverage: data.voteAverage)  
 updateImage(path: data.backDropPath)  
   
 ...  
   
 }

Sua classe ficará assim, por enquanto:

class MovieDetailsViewController: UIViewController {  
   
 var content: Content<MovieDetails.ViewModel> = .loading {  
 didSet {  
 updateViews()  
 }  
 }  
   
 @IBOutlet weak var contentView: UIView!  
 @IBOutlet weak var loadingView: UIView!  
 @IBOutlet weak var errorView: UIView!  
 @IBOutlet weak var errorLabel: UILabel!  
   
 @IBOutlet weak var backDropImageView: UIImageView!  
 @IBOutlet weak var titleLabel: UILabel!  
 @IBOutlet weak var yearLabel: UILabel!  
 @IBOutlet weak var genreLabel: UILabel!  
 @IBOutlet weak var runtimeLabel: UILabel!  
 @IBOutlet weak var overviewLabel: UILabel!  
 @IBOutlet weak var originalTitleLabel: UILabel!  
 @IBOutlet weak var budgetLabel: UILabel!  
 @IBOutlet weak var revenueLabel: UILabel!  
   
 @IBOutlet weak var star1ImageView: UIImageView!  
 @IBOutlet weak var star2ImageView: UIImageView!  
 @IBOutlet weak var star3ImageView: UIImageView!  
 @IBOutlet weak var star4ImageView: UIImageView!  
 @IBOutlet weak var star5ImageView: UIImageView!  
   
 override func viewWillAppear(\_ animated: Bool) {  
 updateViews()  
 }  
   
 func updateViews() {  
 switch content {  
 case .success(let data):  
 errorView.isHidden = true  
 loadingView.isHidden = true  
 contentView.isHidden = false  
   
 updateLabels(viewModel: data)  
 updateStars(voteAverage: data.voteAverage)  
 updateImage(path: data.backDropPath)  
 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  
 }  
 }  
 }  
   
 func updateLabels(viewModel: MovieDetails.ViewModel) {  
 titleLabel.text = viewModel.title  
 yearLabel.text = viewModel.releaseDate  
 genreLabel.text = viewModel.genre  
 runtimeLabel.text = viewModel.runtime  
 overviewLabel.text = viewModel.overview  
 originalTitleLabel.text = viewModel.originalTitle  
 revenueLabel.text = viewModel.revenue  
 budgetLabel.text = viewModel.budget  
 }  
   
 func updateImage(path: String) {  
 let url = "https://image.tmdb.org/t/p/w500/" + path  
   
 backDropImageView.kf.indicatorType = .activity  
   
 backDropImageView.kf.setImage(with: URL(string: url))  
 }  
   
 func updateStars(voteAverage: Int) {  
   
 star1ImageView.image = #imageLiteral(resourceName: "ic\_star\_off")  
 star2ImageView.image = #imageLiteral(resourceName: "ic\_star\_off")  
 star3ImageView.image = #imageLiteral(resourceName: "ic\_star\_off")  
 star4ImageView.image = #imageLiteral(resourceName: "ic\_star\_off")  
 star5ImageView.image = #imageLiteral(resourceName: "ic\_star\_off")  
   
 switch voteAverage {  
 case 1:  
 star1ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 case 2:  
 star1ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 star2ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 case 3:  
 star1ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 star2ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 star3ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 case 4:  
 star1ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 star2ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 star3ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 star4ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 case 5:  
 star1ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 star2ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 star3ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 star4ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 star5ImageView.image = #imageLiteral(resourceName: "ic\_star\_on")  
 default: break  
 }  
 }  
}

Agora precisaremos criar o Interactor. Este é fácil e seguirá a mesma lógica do **GenresInteractor**.

1. Crie um novo arquivo chamado **MovieDetailsInteractor.swift** e nele crie uma classe de mesmo nome:

class MovieDetailsInteractor {  
}

1. Faça com que ela siga o padrão Singleton:

static var shared = MovieDetailsInteractor()  
private init() {}

1. Adicione uma propriedade para armazenarmos o filme selecionado. Este filme será selecionado na cena de **Lista de filmes**, e passado para este interactor.

var selectedMovie: Movie.ResponseModel.Result?

1. Adicione o método que se comunicará com a classe da camada de serviço:

func getMovie(completion: @escaping (Content<MovieDetails.ViewModel>) -> Void) {  
 guard let selected = selectedMovie else {  
 completion(Content.error(ReturnError.userMessage("Não foi possível obter os detalhes do filme. Por favor tente novamente mais tarde")))  
 return  
 }  
   
 MovieService.getMovie(withId: selected.id) { result in  
 switch result {  
 case .success(let data):  
   
 let viewModel = MovieDetails.ViewModel(with: data)  
   
 completion(Content.success(viewModel))  
 case .failure(let error):  
 completion(Content.error(error))  
 }  
 }  
 }

Terminamos o interactor. Agora precisamos usá-lo na View Controller

1. Abra a classe **MovieDetailsViewController** e localize o método viewWillAppear()
2. Coloque o seguinte código no final deste método:

MovieDetailsInteractor.shared.getMovie { content in  
 self.content = content  
}

Nossa cena está pronta mas infelizmente não conseguiremos testá-la ainda. Precisamos finalizar a cena **Lista de Filmes** e faremos isto na aula seguinte.

## Aula 3 - Integrando Filmes

Chegamos a nossa aula final. Já temos pronto a camada Service para a cena Filmes e o que precisamos fazer agora é criar a camada Interactor e também finalizar o View Controller.

A cena de filmes precisária chamar três serviços:

* Filmes Populares
* Melhores filmes
* Lançamentos em breve

Estes serviços correspondem aos filtros que inserimos e que podem ser controlados através do Segmented Control.

Além disso, cada um destes serviços apresenta paginação, como visto no aplicativo Android. Como a lista de filmes é muito grande, uma técnica dos desenvolvedores de backend é a utilização de paginação. Vamos supor que existam duzentos mil filmes classificados como Populares, cada um com seu nível de popularidade. Podemos dizer intuitivamente que realizar uma requisição que retornará 200 mil objetos demorará para ser completada e consumirá uma banda de dados muito grande da sua internet. Isto sem contar que existe uma remota possibilidade que o usuário veja os 200 mil filmes retornados em um único uso do app.

Em resumo, é desnecessário retornar um volume de dados tão grande, então cria-se a paginação, onde é determinado um número máximo de resultados que serão retornados na requisição. Por exemplo, imagine que temos uma página que retornará apenas 100 filmes; se dividirmos por 200 mil, temos um total de 2 mil páginas. Para que você possa ver os 200 mil resultados, 2 mil requisições deverão ser feitas, mas o usuário mal verá os 100 primeiros resultados.

Tendo conhecimento disso você terá que fornecer o numero de páginas de cada serviço.

### Atividade

1. Crie o Interactor da cena Filmes. Ele deverá realizar a chamada dos três serviços filtro listados acima. Além disso deverá ter paginação automática, ou seja, quando o usuário rolar até o final da Collection View de filmes, a página seguinte de filmes deverá ser obtida e assim em diante.
2. Complemente o View Controller já criado para a cena Filmes. Adicione o recurso de paginação, as requisições dos filtros, eventos de toque no Segmented Control, seleção de filmes e navegação para a cena Detalhes do Filme ao selecionar um filme da Collection.

### Solução

Vamos começar pelo Interactor.

#### Criando o Interactor

Crie um arquivo chamado MoviesInteractor e adicione o seguinte conteúdo:

class MoviesInteractor {  
 static var shared = MoviesInteractor()  
 private init() {}  
   
 // Modelo de lista  
 struct MovieList {  
 var list: [Movie.ResponseModel.Result]  
 var page: Int // page indica a próxima página da respectiva lista  
 }  
   
 // Inicializamos listas vazias na página 1  
 var popularMovies = MovieList(list: [], page: 1)  
 var topRatedMovies = MovieList(list: [], page: 1)  
 var upcomingMovies = MovieList(list: [], page: 1)  
   
   
   
 func getPopularMovies(completion: @escaping (Content<[Movie.ViewModel]>) -> Void) {  
 MovieService.getPopularMovies(page: popularMovies.page) { result in  
 switch result {  
 case .success(let movies):  
   
 // Concatena a nova página na lista completa  
 self.popularMovies.list.append(contentsOf: movies)  
   
 // Formata a lista completa  
 let viewModels = self.popularMovies.list.map {  
 Movie.ViewModel(id: $0.id, posterPath: $0.poster\_path ?? "")  
 }  
   
 // Incrementa o número da página na respectiva lista  
 self.popularMovies.page += 1  
   
 completion(Content.success(viewModels))  
 case .failure(let error):  
 completion(Content.error(error))  
 }  
 }  
 }  
   
 func getTopRatedMovies(completion: @escaping (Content<[Movie.ViewModel]>) -> Void) {  
 MovieService.getTopRatedMovies(page: topRatedMovies.page) { result in  
 switch result {  
 case .success(let movies):  
   
 self.topRatedMovies.list.append(contentsOf: movies)  
   
 let viewModels = self.topRatedMovies.list.map {  
 Movie.ViewModel(id: $0.id, posterPath: $0.poster\_path ?? "")  
 }  
   
 self.topRatedMovies.page += 1  
   
 completion(Content.success(viewModels))  
 case .failure(let error):  
 completion(Content.error(error))  
 }  
 }  
 }  
   
 func getUpcomingMovies(completion: @escaping (Content<[Movie.ViewModel]>) -> Void) {  
 MovieService.getUpcomingMovies(page: upcomingMovies.page) { result in  
 switch result {  
 case .success(let movies):  
   
 self.upcomingMovies.list.append(contentsOf: movies)  
   
 let viewModels = self.upcomingMovies.list.map {  
 Movie.ViewModel(id: $0.id, posterPath: $0.poster\_path ?? "")  
 }  
   
 self.upcomingMovies.page += 1  
   
 completion(Content.success(viewModels))  
 case .failure(let error):  
 completion(Content.error(error))  
 }  
 }  
 }  
   
 func setSelectedMovie(with viewModel: Movie.ViewModel) {  
   
 // Junta todas as listas que temos até o momento  
 var allMovies = popularMovies.list  
 allMovies.append(contentsOf: topRatedMovies.list)  
 allMovies.append(contentsOf: upcomingMovies.list)  
   
 // Tenta obter o objeto selecionado comparando o id  
 let selected = allMovies.first {  
 $0.id == viewModel.id  
 }  
   
 // Atualiza o filme selecionado  
 MovieDetailsInteractor.shared.selectedMovie = selected  
 }  
}

Leia com atenção a classe e seus comentários. Vamos entender o que está acontecendo.

A estrutura como um todo da classe é um Singleton como os outros interactors. Mas atente-se para a seguinte struct:

// Modelo de lista  
 struct MovieList {  
 var list: [Movie.ResponseModel.Result]  
 var page: Int // page indica a próxima página da respectiva lista  
 }

Criamos uma struct que armazenará uma lista de ResponseModels e a próxima página que deveremos requisitar ao utilizarmos o filtro da respectiva lista. Mais abaixo, criamos três propriedades MovieList, uma para cada filtro e serviço.

// Inicializamos listas vazias na página 1  
 var popularMovies = MovieList(list: [], page: 1)  
 var topRatedMovies = MovieList(list: [], page: 1)  
 var upcomingMovies = MovieList(list: [], page: 1)

O método setSelectedMovie() é responsável por atualizar o ResponseModel referente ao filme atualizado dado um ViewModel. É por isso que nosso Movie.ViewModel armazena um id que não é exibido na tela, pois precisamos dele para obter o respectivo ResponseModel.

func setSelectedMovie(with viewModel: Movie.ViewModel) {  
   
 // Junta todas as listas que temos até o momento  
 var allMovies = popularMovies.list  
 allMovies.append(contentsOf: topRatedMovies.list)  
 allMovies.append(contentsOf: upcomingMovies.list)  
   
 // Tenta obter o objeto selecionado comparando o id  
 let selected = allMovies.first {  
 $0.id == viewModel.id  
 }  
   
 // Atualiza o filme selecionado  
 MovieDetailsInteractor.shared.selectedMovie = selected  
 }  
}

Os demais métodos fazem a. comunicação com a camada Rest em si. São parecidos com os demais interactors que criamos. Leia os comentários do código para entender melhor.

#### Criando a View Controller.

Primeiro de tudo precisamos criar uma célula personalizada para a Collection View. Sem ela não conseguiremos mudar a imagem do filme de cada célula.

1. Crie um arquivo com o nome **MovieCollectionViewCell.swift** que estenda a classe **UICollectionViewCell**:

class MovieCollectionViewCell: UICollectionViewCell {  
}

1. Adicione o conteúdo básico dela:

@IBOutlet weak var imageView: UIImageView!  
   
 func update(with data: Movie.ViewModel) {  
 let url = "https://image.tmdb.org/t/p/w150/" + data.posterPath  
   
 imageView.kf.indicatorType = .activity  
  
 imageView.kf.setImage(with: URL(string: url))  
 }

1. Neste mesmo arquivo crie uma outra classe que estende de UICollectionViewCell mas com o nome **LoadingCollectionViewCell**:

class LoadingCollectionViewCell: UICollectionViewCell {  
}

1. Adicione seu conteúdo:

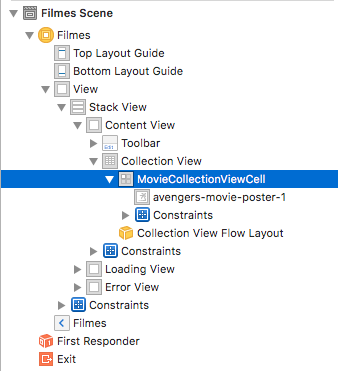
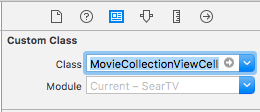
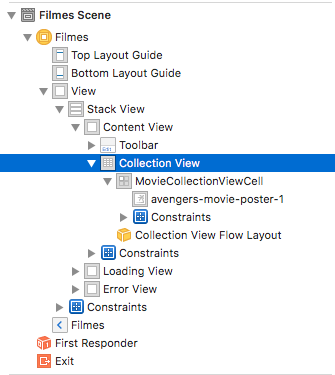
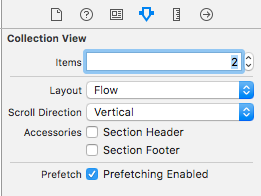
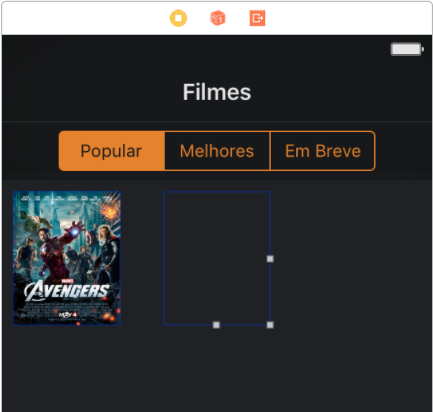
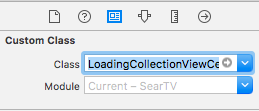
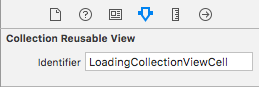
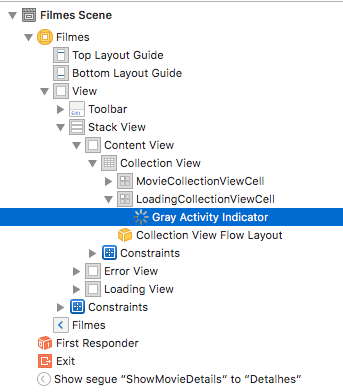
@IBOutlet weak var activityIndicatorView: UIActivityIndicatorView!

1. O arquivo ficará assim:

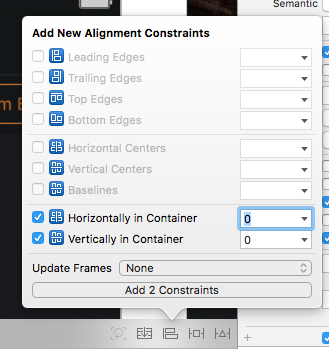
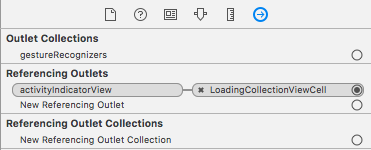
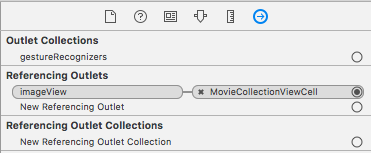
import UIKit  
  
class MovieCollectionViewCell: UICollectionViewCell {  
 @IBOutlet weak var imageView: UIImageView!  
   
 func update(with data: Movie.ViewModel) {  
 let url = "https://image.tmdb.org/t/p/w150/" + data.posterPath  
   
 imageView.kf.indicatorType = .activity  
  
 imageView.kf.setImage(with: URL(string: url))  
 }  
}  
  
class LoadingCollectionViewCell: UICollectionViewCell {  
 @IBOutlet weak var activityIndicatorView: UIActivityIndicatorView!  
}

Esta segunda célula exibirá uma Activity indicator indicando que estamos carregando uma nova página.

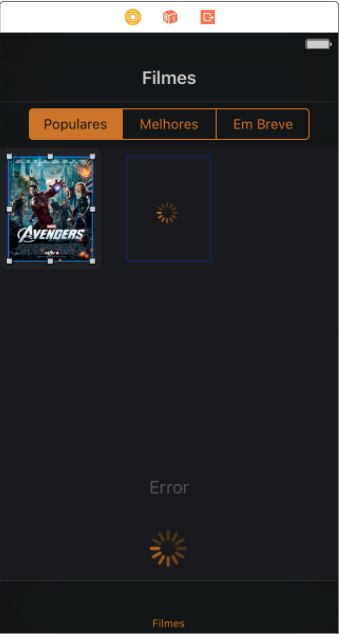
Agora precisamos configurar o View Controller do nosso Storyboard

1. Entre no **Main.storyboard** e selecione a cena **Filmes**.
2. Selecione a **Collecion View Cell** que colocamos uma imagem:  
   
3. Na aba **Identity Inspector** escreva **MovieCollectionViewCell** no campo **Class**.  
   
4. Selecione agora a **Collection View**  
   
5. Adicione mais uma célula escrevendo “2” no campo **Items**:  
   
6. Selecione a nova célula:  
   
7. Especifique no campo **Class** a classe **LoadingCollectionViewCell**  
   
8. E no campo **Identifier** o mesmo nome colocado no campo Class:  
   
9. Insira um **Activity Indicator** na nova célula:  
   
10. Selecione este activity indicator e configure as seguintes propriedades:

* Style: **White**
* Color: **#FF7F00**
* Animating: **true**
* Hides when stops: **true**

1. Adicione as seguintes restrições para ele ficar centralizado na célula:  
   
2. Não se esqueça de elencá-lo ao @IBOutlet criado na classe LoadingCollectionViewCell.  
   
3. Selecione a outra célula e ligue a UIImageView no @IBOutlet da classe MovieCollectionViewCell.  
   

O seu layout deverá ficar assim:



#### View Controller

Agora iremos mudar algumas coisas em nosso **MoviesViewController**.

1. Primeiro vamos criar alguns controles para utilizarmos os filtros. Primeiro crie o enum que contenha três cases para os respectivos filtros (adicione dentro da classe):

enum Filter: Int {  
 case popular  
 case topRated  
 case upcoming  
 }

1. Adicione o seguinte método que fará a chamada dos métodos no Interactor.

func getMovies() {  
   
 content = .loading  
   
 let completionHandler: (Content<[Movie.ViewModel]>) -> Void = { content in  
 self.content = content  
 }  
   
 switch currentFilter {  
 case .topRated:  
 MoviesInteractor.shared.getTopRatedMovies(completion: completionHandler)  
 case .popular:  
 MoviesInteractor.shared.getPopularMovies(completion: completionHandler)  
 case .upcoming:  
 MoviesInteractor.shared.getUpcomingMovies(completion: completionHandler)  
 }  
 }

Repare que como o completion handler das três chamadas do interactor serão iguais, armazenamos este closure em uma variável completionHandler para ser reutilizada. Este método chamará o serviço de acordo com a variável que criaremos no próximo passo.

1. Agora precisamos adicionar uma propriedade para especificar que segmento/filtro está selecionado atualmente. Ao selecionarmos um filtro, o método getMovies() deve ser invocado. Então utilize o didSet. Veja:

var currentFilter: Filter = .popular {  
 didSet {  
 getMovies()  
 }  
 }

1. Adicione um evento para mudança de segmento do Segmented Control, e não se esqueça de elencá-lo a view.

@IBAction func segmentedControlValueChanged(\_ sender: UISegmentedControl) {  
 currentFilter = Filter(rawValue: sender.selectedSegmentIndex) ?? .popular  
}

1. Por fim substitua toda a extension que implementamos os delegates para a collection view, por esta:

extension MoviesViewController: UICollectionViewDelegate, UICollectionViewDataSource {  
 func collectionView(\_ collectionView: UICollectionView, didSelectItemAt indexPath: IndexPath) {  
   
 MoviesInteractor.shared.setSelectedMovie(with: movies[indexPath.row])  
 performSegue(withIdentifier: "ShowMovieDetails", sender: nil)  
   
 }  
   
 func collectionView(\_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {  
 let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "MovieCollectionViewCell", for: indexPath) as! MovieCollectionViewCell  
   
 let index = indexPath.row  
 if index == movies.count { // estamos na última célula  
 let loadingcell = collectionView.dequeueReusableCell(withReuseIdentifier: "LoadingCollectionViewCell", for: indexPath) as! LoadingCollectionViewCell  
 loadingcell.activityIndicatorView.startAnimating()  
   
 return loadingcell  
 } else {  
 cell.update(with: movies[indexPath.row])  
 }  
 return cell  
 }  
   
 func numberOfSections(in collectionView: UICollectionView) -> Int {  
 return 1  
 }  
   
 func collectionView(\_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {  
 // O número de itens da collection é incrementado de 1 para exibirmos a célula de Loading (LoadingCollectionViewCell)  
 return movies.count + 1  
 }  
   
 // utilizamos este método para identificarmos rolagem da Collection View  
 func scrollViewDidScroll(\_ scrollView: UIScrollView) {  
 let scrollViewHeight = scrollView.frame.size.height  
 let scrollContentSizeHeight = scrollView.contentSize.height  
 let scrollOffset = scrollView.contentOffset.y  
   
 if (scrollOffset + scrollViewHeight) == scrollContentSizeHeight {  
 // estamos no final da scrollView  
 getMovies()  
 }  
 }   
}