Unidade 4

# Aula 1

# View Controllers

Nas aulas anteriores aprendemos a compor a interface de usuário nos *storyboards* sem ao menos ter que escrever uma linha de código, mas nem sempre isto será possível. Para criar aplicações dinâmicas e mais complexas precisaremos entender como funcionam as *views* e View Controllers. Nesta aula iremos entender estes conceitos e aprender a interceptar os eventos de entrada dos nossos botões.

DICA: Quando o usuário faz alguma ação em algum componente, como pressionar um botão na tela do celular, por exemplo, um evento é acionado (disparado) automaticamente. Se quisermos fazer algo quando o usuário tocar esse componente na tela do celular, precisamos intercepta-lo.

## Entendendo a hierarquia das Views

Objetivamente a View é representada pela classe UIView e define áreas retangulares que podemos posicionar na tela do nosso aplicativo. Nela podemos colocar outras UIView, criando uma estrutura de camadas de Views dentro da área de cada View.

A UIView é a classe mais primitiva das Views, mas podemos encontrar suas subclasses, como UIButton, UIImageView, UIControl que possuem funcionalidades e características específicas para cada uso. No decorrer do seu aprendizado você pode consultar a documentação oficial do iOS para entender o que cada subclasse faz (há uma variedade enorme de subclasses da UIView).

### Visão Geral

Em tempo de execução um objeto *view* lida com a apresentação de qualquer conteúdo em sua área e também lida com todas as interações com esse conteúdo. Views com conteúdo mais sofisticado podem ser apresentadas por subclasses da UIView e implementam o desenho (apresentação) e códigos de manipulações dos eventos necessários para cada caso. A UIView faz parte do *framework* (ou grupo de classes) UIKit, que também inclui um conjunto de subclasses padrões que variam de simples botões até tabelas complexas. Por exemplo, um objeto UILabel, um texto (*string*) na tela e um UIImageView desenham uma imagem.

Devido ao fato de que a *view* é a principal maneira com que seu aplicativo interage com o usuário, ela tem uma série de responsabilidades. Aqui estão apenas alguns:

* **Desenho e animação**
  + Desenhar o conteúdo em sua área retangular usando tecnologias como UIKit, Core Graphics e OpenGL ES;
  + Algumas propriedades da *view* podem ser utilizadas para definirem animações desta *view*.
* **Gerenciamento de layout e subview**
  + Uma *view* pode conter zero ou mais *subviews*;
  + Cada v*iew* define o seu próprio comportamento de redimensionamento padrão em relação à sua *superview*;
  + Uma v*iew* pode definir o tamanho e posição de suas *subviews* conforme necessário.
* **Manipulação de eventos**
  + A View atende a um Responder e isto significa que pode lidar com (ou responde a) eventos de toque e outros eventos definidos pela classe UIResponder;
  + Views podem usar o método addGestureRecognizer(\_:) para instalar reconhecedores de gesto para lidar com gestos comuns no smartphone.

Com as *views* é possível ainda incorporar outras v*iews*, criando hierarquias visuais sofisticadas. Isto cria uma relação “mãe-filha” entre uma v*iew* que está sendo incorporada à outra *view* (conhecida como *subview*) e a “*view* mãe”, que é a incorporadora (conhecida como *superview*). Normalmente a área visível de uma *subview* não é limitada dentro de sua *superview* dentro dos limites de sua *superview*, mas em iOS podemos utilizar a propriedade clipsToBounds para reverter este comportamento. Toda View Controller possui uma *view* principal que pode conter um número indefinido de *subviews,* mas não possui uma *superview*, pois ela está no ponto mais alto da hierarquia visual. Já que mencionamos a View Controller, vamos compreender seu funcionamento a seguir.

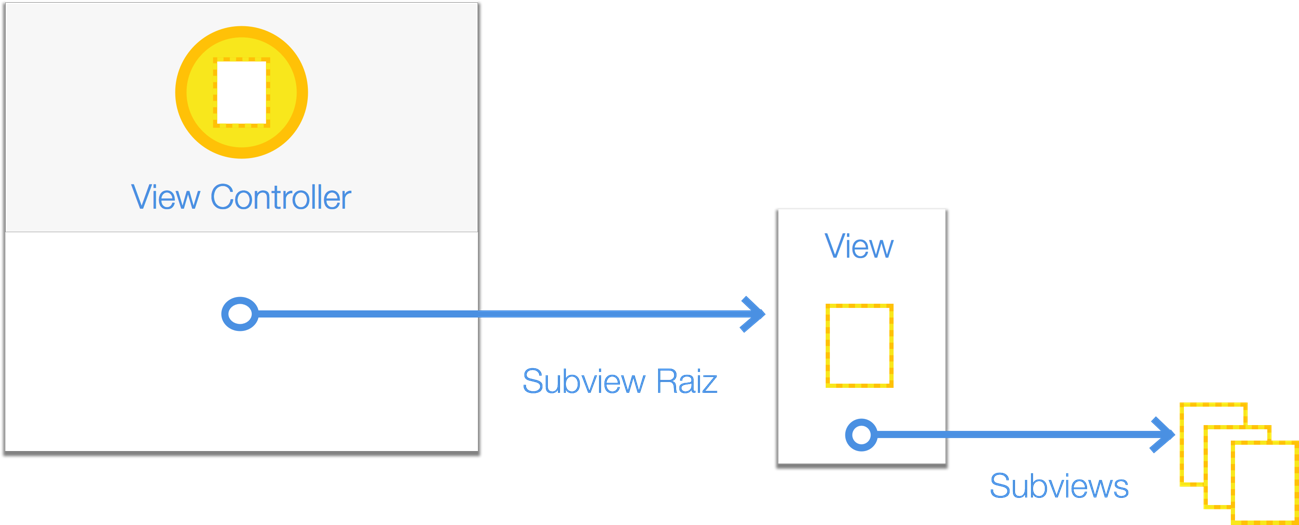


Figura 1 - Representação simplificada da hieraquia de ViewController e suas views

## Compreendendo a View Controller

A classe UIViewController fornece a infraestrutura apropriada para a gestão das *views* dos nossos aplicativos iOS. É responsável por: carregar e eliminar tais *views*, gerenciar interações entre elas, e mediar a interação entre quaisquer modelos de dados. Resumidamente, a View Controller controla e gerencia as *views* delegadas à ela. Considerando o Android, podemos compará-la com a Activity.

### O ciclo de vida de uma View Controller

Um objeto UIViewController (e suas subclasses) vêm com um conjunto de métodos que gerem a sua hierarquia de *views*. O iOS aciona automaticamente estes métodos em momentos adequados, como em transições e mudanças de estados das View Controllers.

Quando criamos uma subclasse de UIViewController (sempre trabalharemos em alguma subclasse de UIViewController), ela herda os métodos definidos na superclasse e permite que adicionemos o comportamento que desejamos para cada um destes métodos. Isto porque esta estrutura de métodos está organizada no padrão de projeto Template Method, visto no curso de Android, e que é o mesmo padrão aplicado na Activity e seu ciclo de vida.

É importante entender que para que você possa configurar certos comportamentos de *views* ou eliminá-las, é necessário estar no método correto, ou seja, existe um método da View Controller apropriado para cada configuração de views a ser feita. A seguir, temos uma ilustração vinda da documentação oficial do iOS que exemplifica o ciclo de vida de uma View Controller e os métodos aqui em questão.

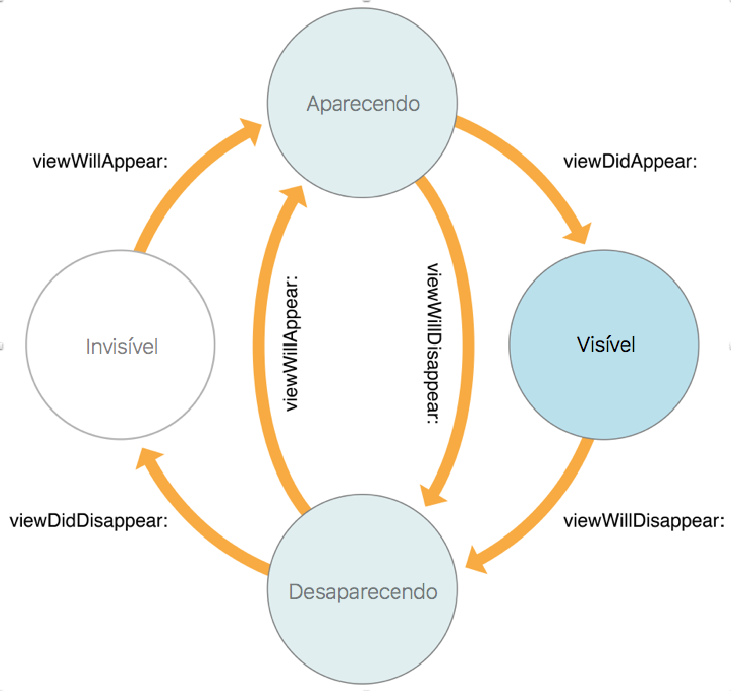


Figura 1.1 – Ciclo de vida de uma View Controller

A UIViewController chama os métodos na sequência como se segue:

* **viewDidLoad()** – É chamado quando o conteúdo da *view* principal do View Controller (a que está no topo da hierarquia de visualização) é criado e carregado pelo *storyboard*. Este método destina-se a configurações iniciais. Entretanto, devido ao fato de que o *smartphone* possui recursos limitados, esta *view* pode ser destruída e reconstruída dependendo do sistema operacional, então não é garantido que este método será chamado apenas uma vez no ciclo, mas o fato é que ele é o ponto inicial da criação de uma View Controller;

**viewWillAppear()** – É destinado para realizar quaisquer operações quando a *view* principal torna-se visível. Devido a possibilidade da visibilidade da *view* ser alterada por outras *views*, este método é chamado diversas vezes. Em uma explicação com nível de abstração mais alto, sempre que **a tela se tornar visível**, este método é invocado, seja porque esta tela esta sendo criada, ou

* seja porque estamos retornando de outra tela;
* **viewDidAppear()** – Este é destinado para qualquer operação que você deseja que ocorra logo que a *view* se torna visível, como busca de dados ou mostrar animações. Pelo mesmo motivo retratado no **viewWillAppear(),** este método pode ser invocado diversas vezes pela View Controller. Em uma explicação com nível de abstração mais alto, sempre que **a tela já está visível** este método é invocado, logo após o método **viewWillAppear()**.

Estes são os principais métodos que você provavelmente fará uso massivo, para os métodos **viewWillDesapear()** e **viewDidDisapear(),** mostrados no diagrama, vale o análogo dos métodos **viewWillAppear()** e **viewDidAppear()** respectivamente, ou seja, quando os **dois últimos** valem para quando a view irá **aparecer**, os **dois primeiros** valem para quando a *view* irá **desaparecer.**

## UINavigationController

A classe UINavigationController implementa um View Controller que gera a navegação entre conteúdos. Esta interface de navegação torna possível apresentar os seus dados de forma eficiente e torna mais fácil para que o usuário navegue por estes conteúdos. Você pode utilizar esta classe exatamente como ela está, pois já oferece as funcionalidades mínimas para tais objetivos mencionados, mas você pode também criar subclasses para personalizar seu comportamento e aparência livremente.

As telas apresentadas por uma interface de navegação normalmente imitam a maneira como a hierarquia dos seus dados está organizada. A cada nível hierárquico, você fornece a tela apropriada (gerida por uma View Controller personalizada) para que o conteúdo deste nível (na estrutura dos seus dados) seja exibido.

A imagem seguinte mostra um exemplo da interface de navegação do aplicativo **Settings** (ou Ajustes) do simulador do iOS. A primeira tela apresenta ao usuário uma lista de preferências de aplicativos. Quando selecionamos um aplicativo, é revelada as configurações e os grupos de configurações deste aplicativo individualmente. Selecionada uma destas configurações é revelada mais configurações e por aí em diante. Para todas as telas, exceto para a tela raiz, é apresentado um botão de retorno, que permite que o usuário volte para cima na hierarquia.

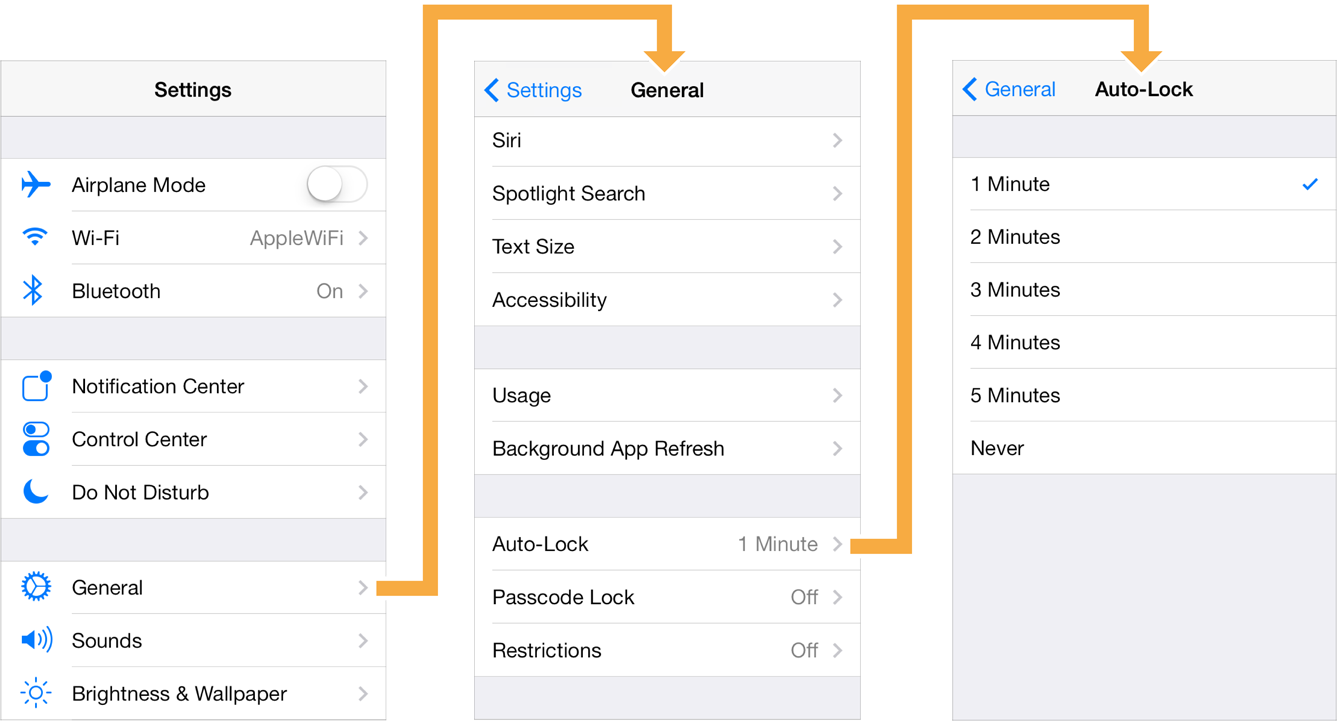


Figura 1.2 – Exemplo de navegação no iOS com o app Ajustes

### Definindo a navegação do app

Vamos agora implementar uma UINavigationBar no nosso projeto. Para isto utilize o projeto fornecido pelo seu Instrutor, localizado no arquivo **Unidade\_4\_-\_Aula\_1\_-\_Exemplo\_1.zip**.

**Material de apoio:**

O projeto de ponto de partida desta aula se encontra no arquivo **Unidade\_4\_-\_Aula\_1\_-\_Exemplo\_1.zip** da pasta de Material de apoio.

Nesta atividade iremos adicionar a navegação entre as cenas **Navegar** e **Música**:

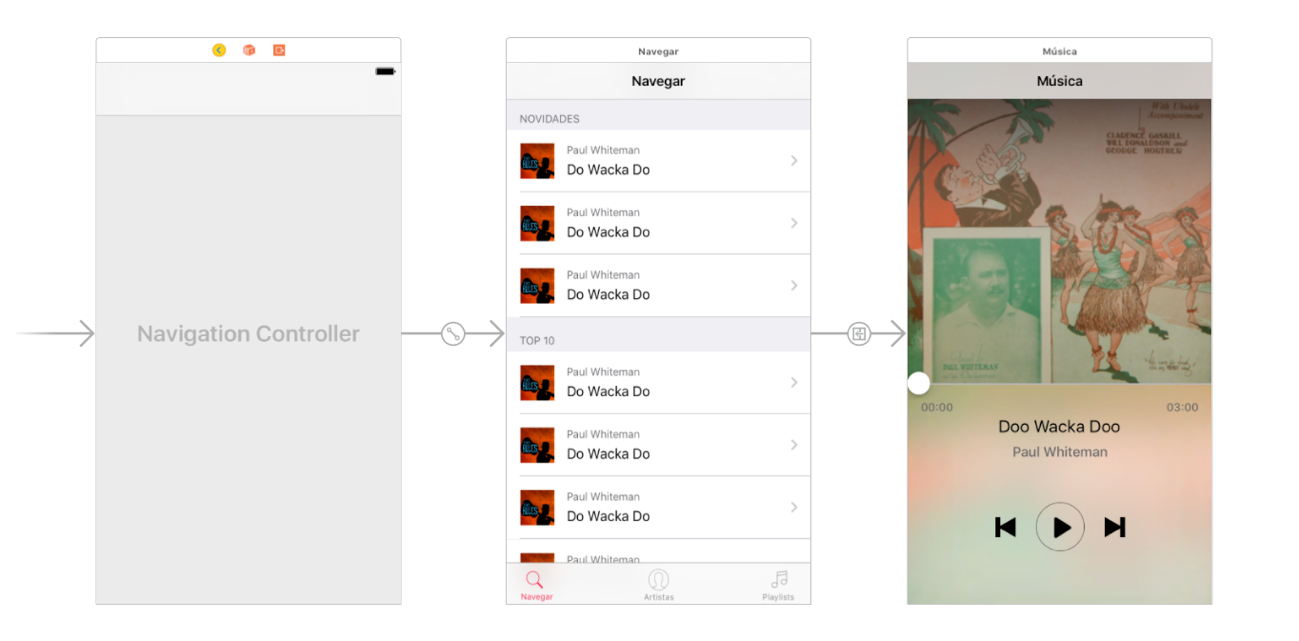


Figura 1.3 – Resultado esperado de navegação entre a cena **Navegar** e **Música** no Interface Builder

Então siga os seguintes passos:

1. Abra o **Main.storyboard**. Nele, teremos somente a seguinte cena:

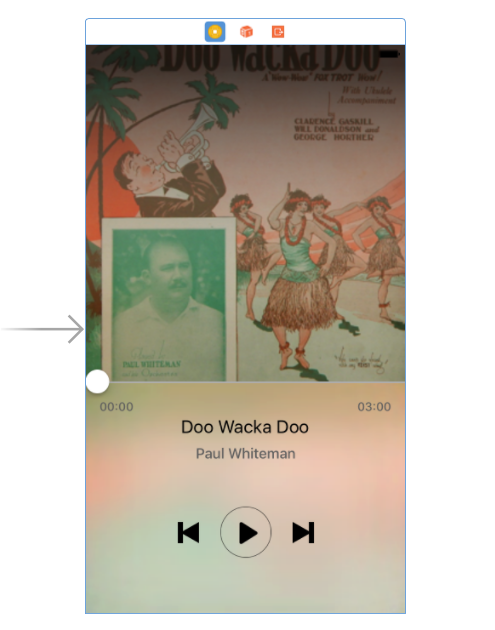


Figura 1.4 – Cena **Música** já criada

1. Vamos criar então uma cena para a tela **Navegar**. Ela irá preceder a cena **Música** e chamará esta tela. Para isso, vá até a paleta de componentes pegue uma UIViewController e arraste-a até o *storyboard*.

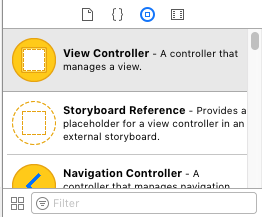


Figura 1.5 – Localização do **UIViewController** na Paleta de Componentes

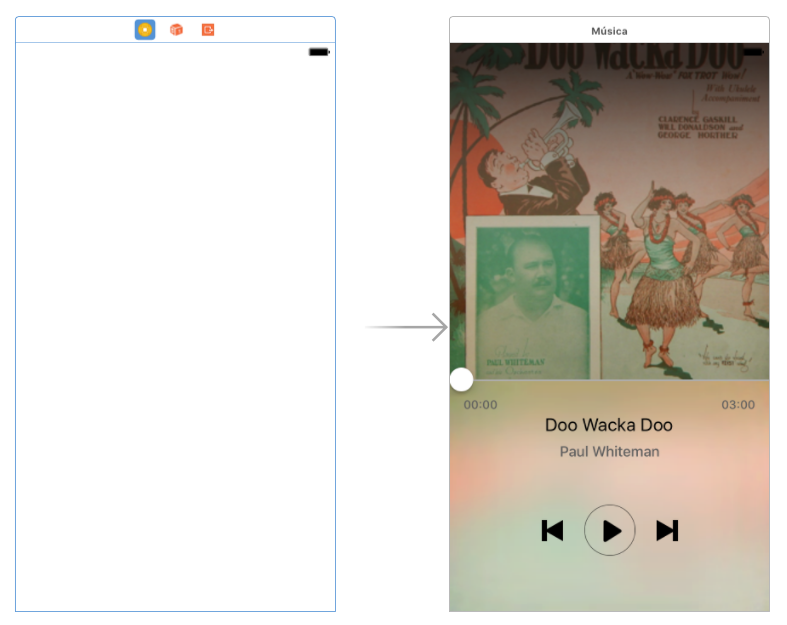
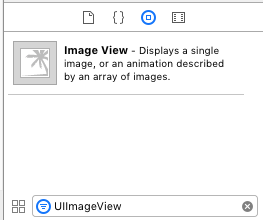


Figura 1.6 – Posicionamento da nova View Controller no Storyboard

1. Para que não tenhamos uma tela vazia, vamos inserir a imagem de protótipo na nova ViewController. Então pegue uma UIImageView e insira na nova cena.



Localização da **UIImageView** na Paleta de Componentes

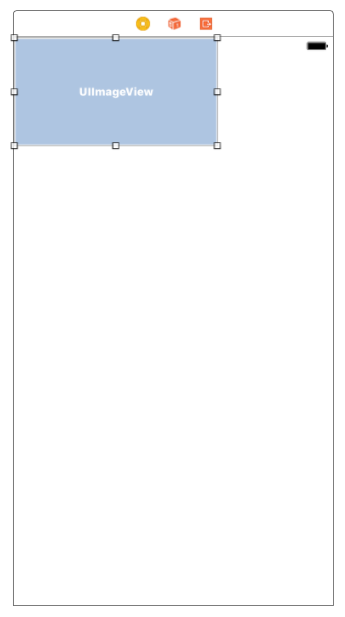


Figura 1.8 – Posicionamento da Image View na nova View Controller

1. Escolha a imagem **1. Navegar** para a Image View.

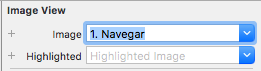


Figura 1.9 – Definindo uma imagem a ser exibida na Image View

1. Agora ajuste as restrições da Image View para que ela fique aderida nas bordas de sua *superview*:

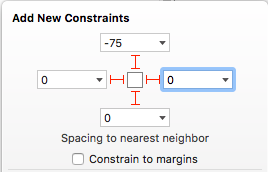


Figura 1.10 – Definindo as Restrições da Image View

DICA: Temos uma restrição de topo com o valor **-75.** Como nossa imagem aparece uma Navigation Bar, e nosso objetivo é justamente inserir uma Navigation Bar real, este valor da restrição faz com que a Navigation Bar da imagem se esconda.

1. Feito isto temos o seguinte resultado:

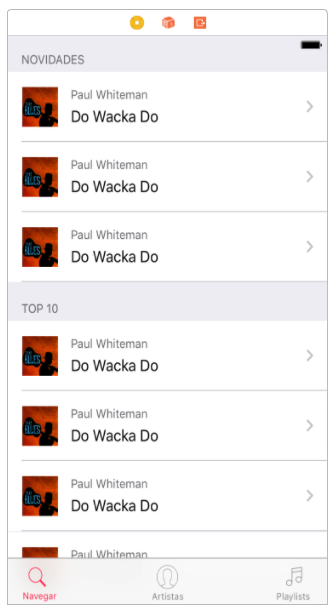


Figura 1.11 – Imagem representativa da cena Navegar sendo apresentada pela Image View

1. O próximo passo agora é inserir um Navigation Controller nos nossos *controllers*. Tenha em mente que a View Controller raiz deste fluxo, ou seja, a primeira tela que irá aparecer é a cena **Navegar**, então selecione a View Controller destinada a esta cena, clicando no ícone amarelo no topo dela:

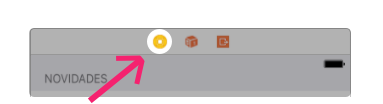


Figura 1.12 – Ícone de seleção do View Controller

1. Agora um atalho para embarcar esta View Controller em um Navigation Controller é acessar o menu **Editor > Embed In > Navigation Controller:**

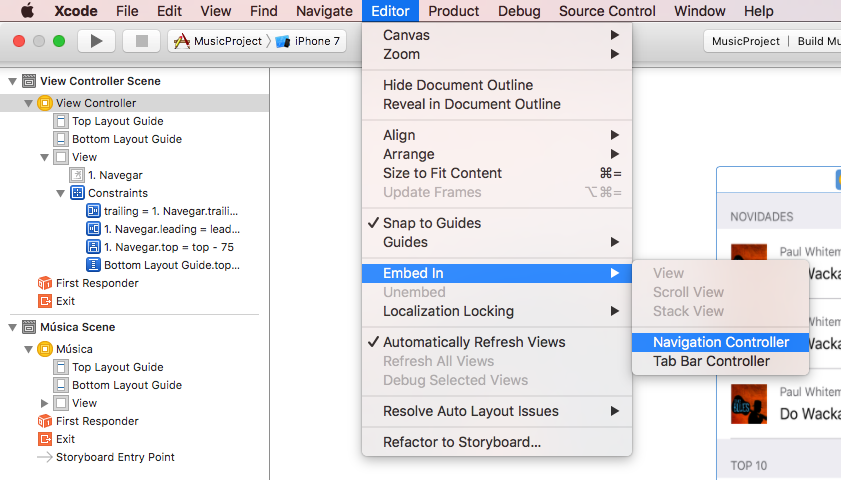


Figura 1.13 – Embarcando o View Controller em um Navigation Controller

1. Você terá o seguinte resultado:

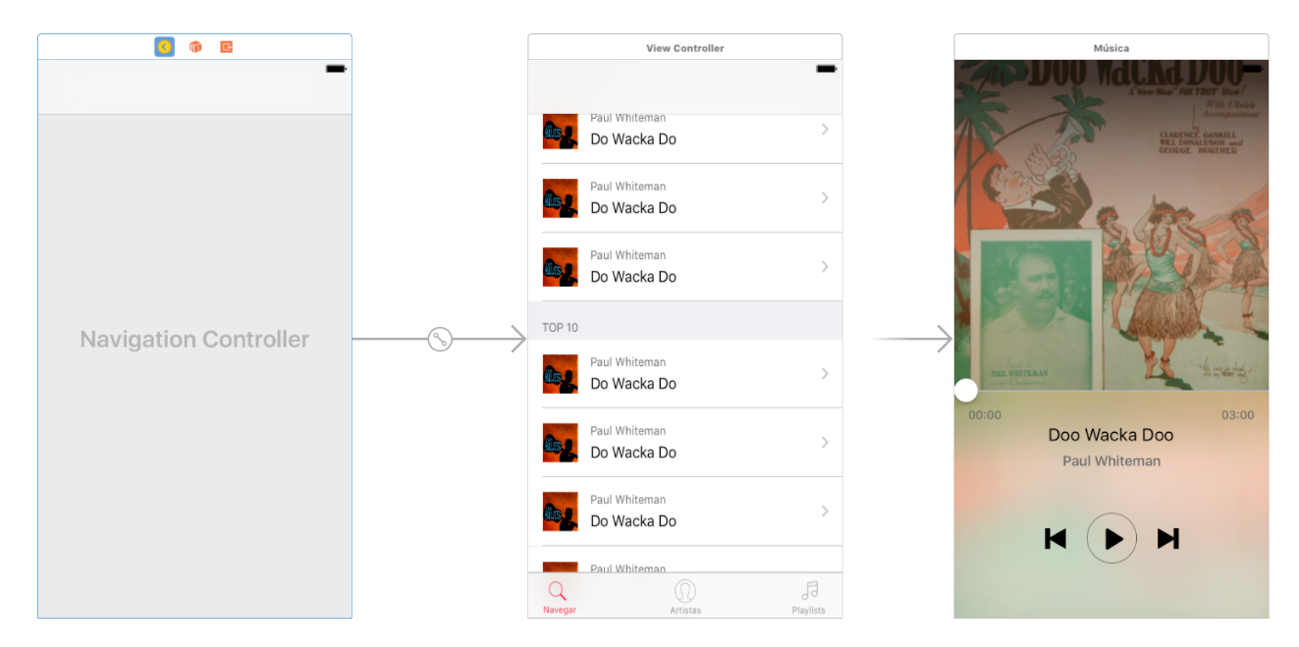


Figura 1.14 – Resultado da cena com a Navigation Controller

1. Com a Navigation Controller selecionada, vamos agora personaliza-la. Marque a opção **Is Initial View Controller:**

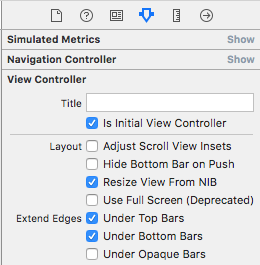


Figura 1.15 – Definindo a Navigation Controller como View Controller inicial

1. Selecione a sua Navigation Bar:

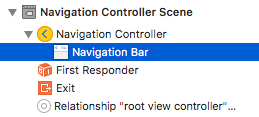


Figura 1.16 – Localização da Navigation Bar no painel de hierarquia das Views

1. Mude a cor da propriedade **Tint** para a cor #FF2D55 (../../../../Desktop/Captura%20de%20Tela%202016-11-22%20às%2022.33).

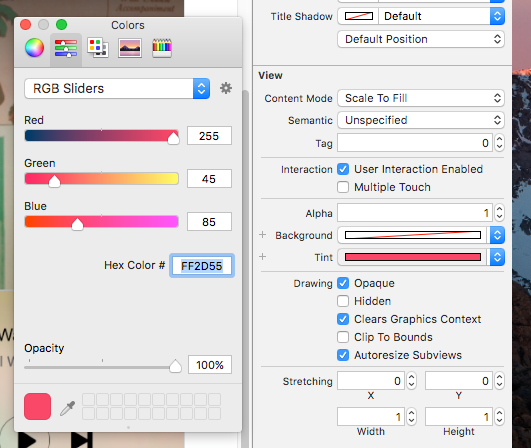


Figura 1.17 – Definindo a cor de **Tint** da Navigation Bar

DICA: Com esta propriedade o botão **voltar** da nossa barra de navegação mudará de cor.

1. Agora dê um duplo clique na Navigation Bar visível na cena **Navegar** e escreva “Navegar”**:**

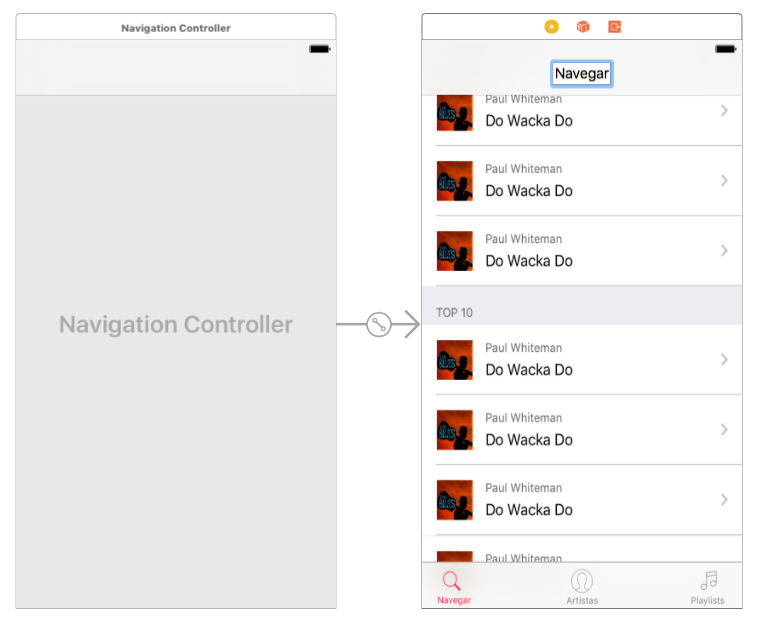


Figura 1.18 – Definindo o título da cena na Navigation Bar

1. O próximo passo é adicionar um botão invisível na cena para que possamos tocá-lo e exibir a cena **Música**. Então adicione um botão sem título conforme a imagem abaixo, como fizemos na Unidade 3:

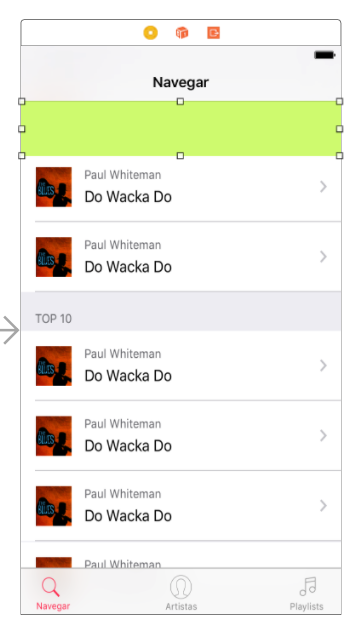


Figura 1.19 – Adicionando um botão provisório na cena

DICA: A cor do botão está verde para destacar para você o local que ele deve ser posicionado. Você pode deixa-lo com a cor transparente para aproximar o layout do que se espera.

1. Adicione as restrições para que ele se fixe nesta posição:

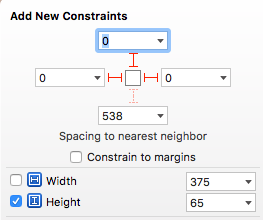


Figura 1.20 – Definindo as restrições do botão

1. Pronto. O que precisamos agora é fazer uma ligação do botão até a cena **Música**:

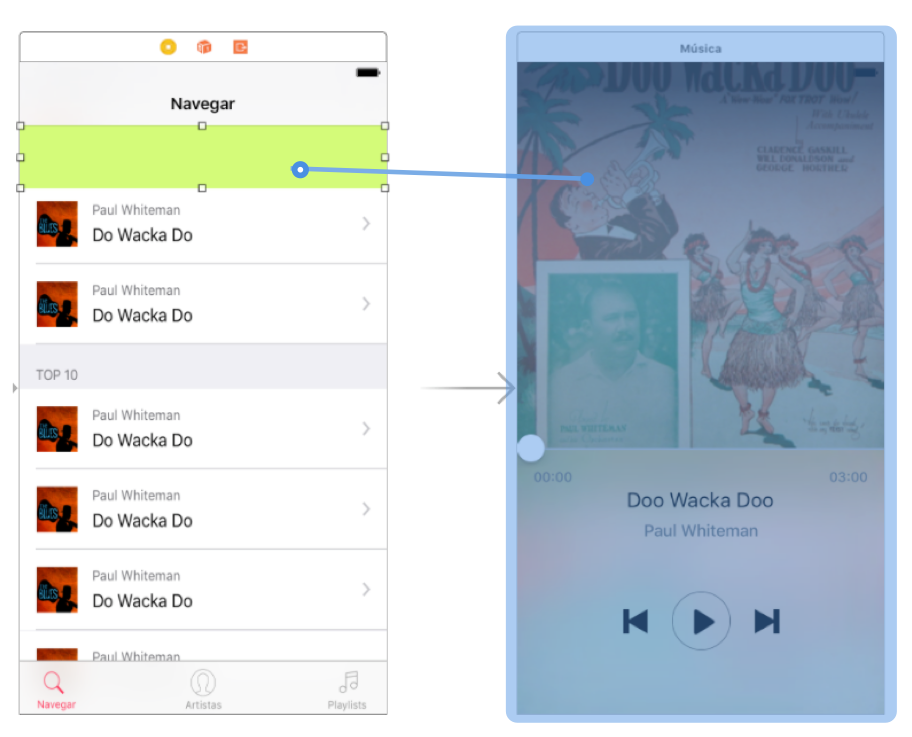


Figura 1.21 – Criando uma ligação entre duas cenas

1. Escolha a opção **Show**.

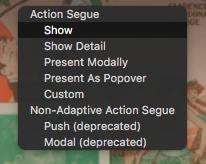


Figura 1.22 – Menu de opções em uma ligação

1. Você terá o seguinte resultado:

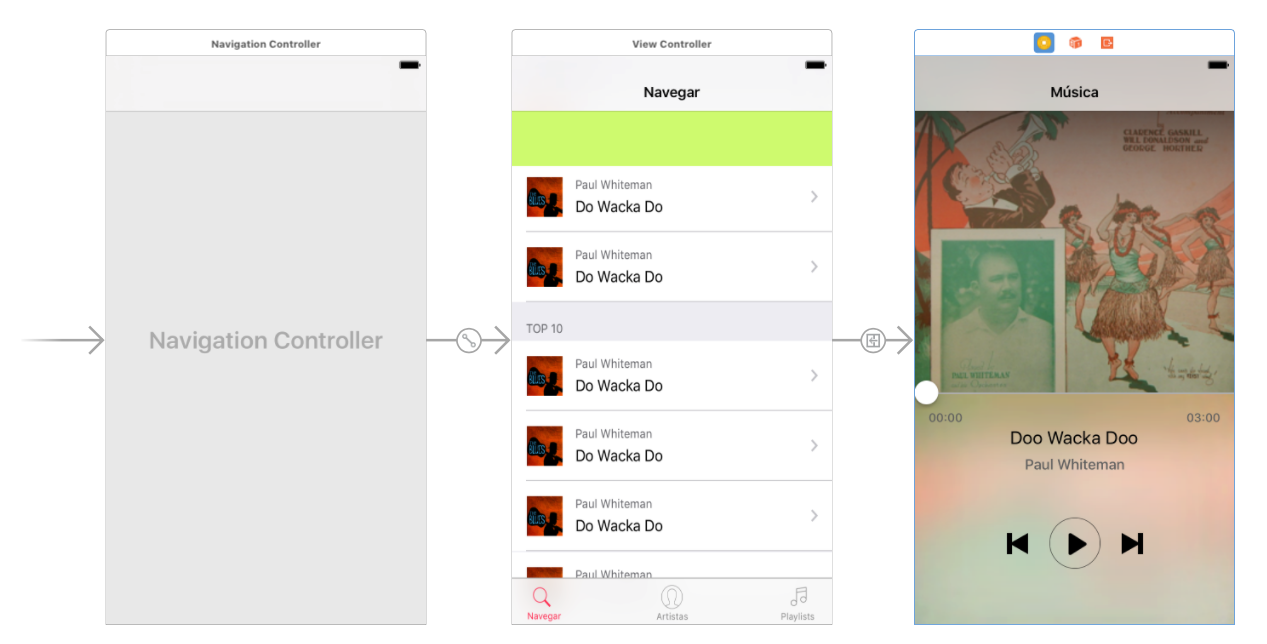


Figura 1.23 – Resultado da navegação no Storyboard

Agora você pode executar o *app* e testar o resultado final.

Veja que sem uma linha de código podemos definir a navegação do fluxo do *app*, inclusive o fluxo de volta.

Veja o resultado da cena **Música** com uma Navigation Controller:



Figura 1.24 – Navigation Bar sendo mostrada na cena Música

## UITabBarController

A classe UITabBarController implementa um View Controller que gere uma interface de seleção no estilo rádio.

DICA: Uma seleção rádio é uma seleção onde podemos escolher exclusivamente um item de uma lista.

Esta interface exibe guias ou abas em uma barra na parte inferior da tela. Cada aba está relacionada a um View Controller e ao tocá-la esta View Controller é exibida. Como a UINavigationController, podemos utilizar a UITabBarController exatamente como ela está e é possível fazer personalizações.

Como dito anteriormente, cada aba de um Tab Bar Controller está associada a uma View Controller personalizada. Quando o usuário seleciona uma aba específica, a Tab Bar Controller coloca a View Controller na raiz da estrutura de navegação, substituindo as View Controllers que estavam lá anteriormente. As Tab Bar Controllers são utilizadas comumente para apresentar tipos diferentes de informações ou apresentar a mesma informação utilizando estilos completamente diferentes. A figura abaixo mostra com o exemplo do nosso projeto um Tab Bar Controller. Cada aba representa um fluxo diferente do *app*, com informações não relacionadas entre si.

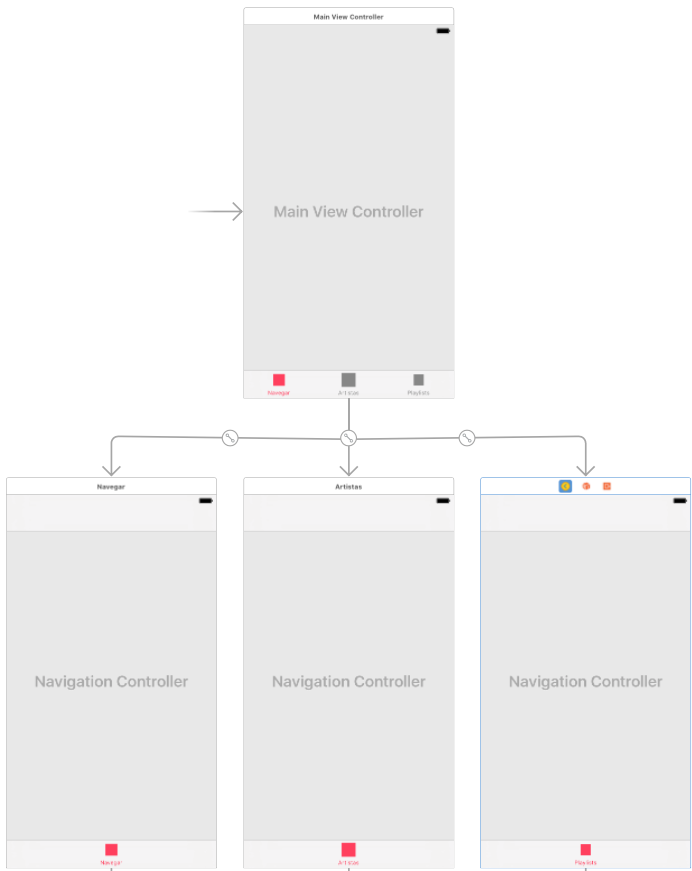


Figura 1.25 – Estrutura de Views Controllers em uma Tab Bar Controller

Você nunca deve selecionar uma aba na Tab Bar Controller diretamente, via código. Isto é algo que o próprio iOS deve cuidar mediante a interação do usuário. Para configurar as abas da Tab Bar Controller basta associarmos uma View Controller. A ordem que as abas aparecerão na tela será definida de acordo com a ordem que as View Controllers são associadas a Tab Bar Controller. Ao definir as abas, também podemos definir qual será a aba inicialmente selecionada e, por padrão, a primeira aba (da esquerda para a direita) estará selecionada.

Outro ponto importante é que podemos associar o Navigation Controllers, já que também são View Controllers. Sendo assim, a Tab Bar Controller gerencia um tipo de fluxo em abas e a Navigation Controller gerencia outro tipo de fluxo (sequencial) de View Controllers; então podemos fundir estes fluxos. Tendo esta possibilidade, é interessante saber que quando alternamos entre abas que possuem Navigation Controllers, o estado destas são mantidos.

### Criando uma navegação em abas

Vamos agora implementar uma UITabBarController no nosso projeto. Para isto utilize o projeto fornecido pelo seu Instrutor, presente no arquivo **Unidade\_4\_-\_Aula\_1\_-\_Exemplo\_2.zip**.

**Material de apoio:**

O projeto de ponto de partida desta aula se encontra no arquivo **Unidade\_4\_-\_Aula\_1\_-\_Exemplo\_2.zip** da pasta de Material de apoio.

A nossa Tab Bar Controller será baseada na imagem acima. Para isto inserimos todas as cenas e *navigation controllers* necessários, seguindo os passos do exemplo anterior. Veja como está nosso **Main.storyboard**:

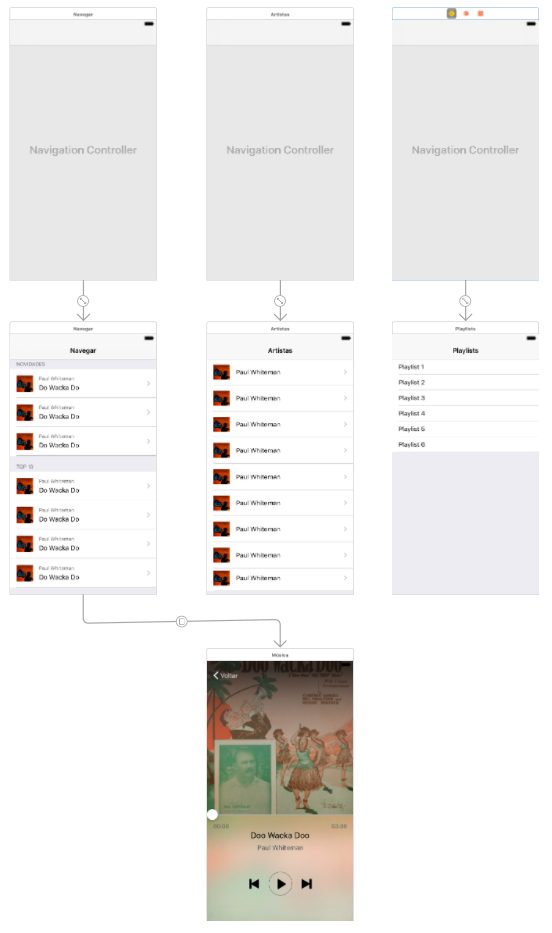


Figura 1.26 – Todas as cenas embarcas em um fluxo com Navigation Controllers

Criamos também uma subclasse de UITabBarController, chamada de MainViewController. É nela que trabalharemos.

Então siga os seguintes passos:

1. Primeiramente procure na paleta de componentes uma UITabBarController e adicione-a no topo do Storyboard (acima das *navigation controllers*):

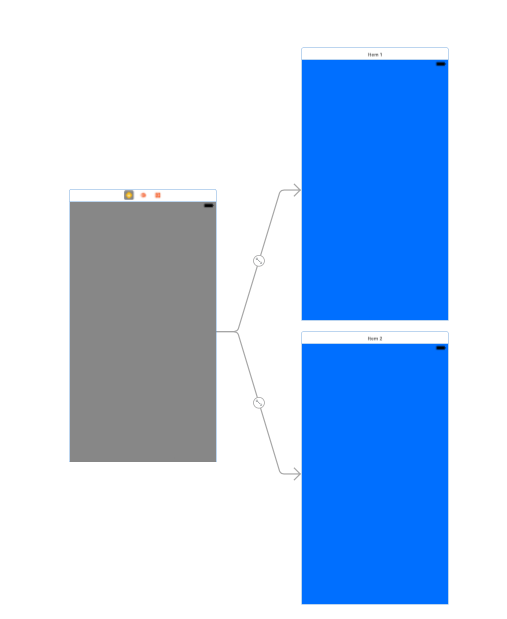


Figura 1.27 – Definindo uma Tab Bar Controller no Storyboard

1. Remova as duas View Controllers que vieram atreladas na UITabBarController.

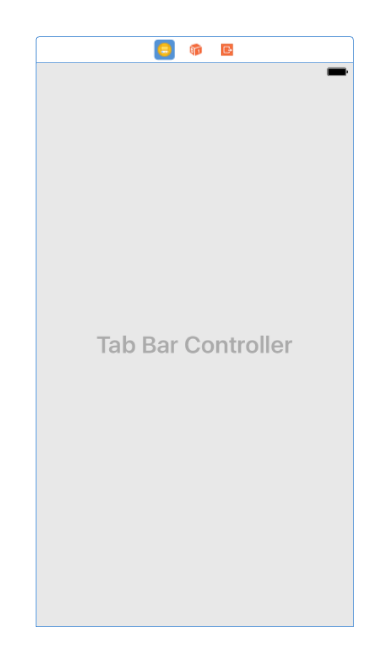


Figura 1.28 – Tab Bar Controller no Storyboard

1. Puxe uma conexão da Tab Bar Controller para cada *navigation controller*. Escolha a opção **View Controller.** O resultado será o seguinte:

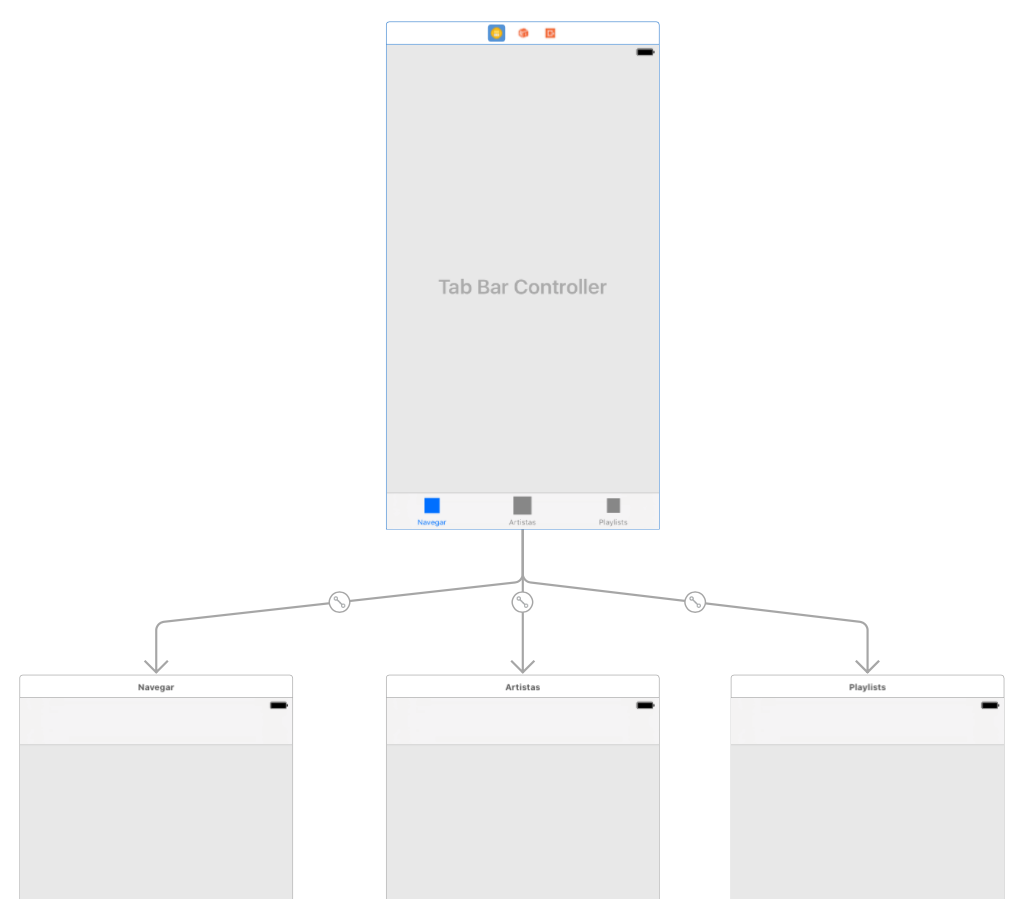


Figura 1.29 – Tab Bar Controller ligada com suas View Controllers

1. Configure a Tab Bar Controller como sendo a view controller inicial (**Is Initial View Controller**)

Neste momento você já pode executar o aplicativo, mas você verá que não temos ícones das *tabs* e que a cor é a padrão do iOS:



Figura 1.30 – Tab Bar Controller funcionando

Para resolver isto execute os seguintes passos:

1. Entre no arquivo **MainViewController.swift** e adicione o seguinte código dentro do método **viewDidLoad():**

UITabBar.appearance().tintColor = #colorLiteral(red: 1, green: 0.2842617035, blue: 0.4058894515, alpha: 1)

((tabBar.items?[0])! as UITabBarItem).image = #imageLiteral(resourceName: "ic\_search").withRenderingMode(.alwaysOriginal)

((tabBar.items?[1])! as UITabBarItem).image = #imageLiteral(resourceName: "ic\_artist").withRenderingMode(.alwaysOriginal)

((tabBar.items?[2])! as UITabBarItem).image = #imageLiteral(resourceName: "ic\_music").withRenderingMode(.alwaysOriginal)

((tabBar.items?[0])! as UITabBarItem).selectedImage = #imageLiteral(resourceName: "ic\_search\_selected").withRenderingMode(.alwaysOriginal)

((tabBar.items?[1])! as UITabBarItem).selectedImage = #imageLiteral(resourceName: "ic\_artist\_selected").withRenderingMode(.alwaysOriginal)

((tabBar.items?[2])! as UITabBarItem).selectedImage = #imageLiteral(resourceName: "ic\_music\_selected").withRenderingMode(.alwaysOriginal)

DICA: Neste código, primeiramente mudamos a propriedade **tint** da tabBar para a cor rosa do nosso aplicativo. Após isto indicamos duas imagens para cada item da Tab Bar Controller, uma para o estado **ativo** e outra para o **inativo**, ou seja, uma será apresentada quando a respectiva aba estiver selecionada e a outra, quando não estiver selecionada.

1. Agora volte ao *storyboard*, selecione a TabBarController e mude a classe dela para **MainViewController**, no painel de propriedades:

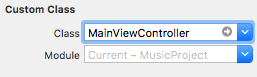


Figura 1.31 – Definindo uma classe **Custom** para a Tab Bar Controller

Neste momento você pode executar o aplicativo e verá todas as imagens corretas nas abas da Tab Bar.

## Resumo

Nesta aula aprendemos alguns dos conceitos mais fundamentais do iOS: a View Controller e suas navegações. Aprendemos a definir o fluxo de navegação com Navigation Controllers e navegação em abas com Tab Bar Controllers.

Com estes *controllers* abrangemos a maioria das possibilidades em um *app*, mas existem muitos outros. Explore a paleta de componente e descubra novos *controllers*.

Aula 2

# Listas com Table Views

Já sabemos que os aplicativos, predominantemente, fazem o uso excessivo de listas para apresentar dados dinâmicos na tela que podem variar, ou, simplesmente, para organizar o conteúdo em um *layout*. A TableView é a classe responsável por colocar uma lista de células em nossa tela e, apesar da sua manipulação ser complicada no começo, ela é um componente poderoso para cuidar desta atividade.

## Table View Controllers

A Table View Controller, representada pela classe **UITableViewController**, é uma classe convenientemente utilizada para gerenciar uma **Table View**. Ela possui algumas limitações, mas para uma introdução à exibição das listas, a Table View Controller é um bom começo. Além disso, ela ajuda na compreensão de alguns dos conceitos mais utilizados no iOS, como os **delegates** e **data sources.**

### Anatomia de uma Table View Controller

Como mencionado, a Table View Controller é uma classe de conveniência, isto é, ela não é fundamental no kit de desenvolvimento iOS, mas ela é feita de componentes existentes e cuidadosamente configurados para facilitar o uso. Uma representação da organização estrutural dos componentes de uma Table View Controller pode ser vista na ilustração seguinte:

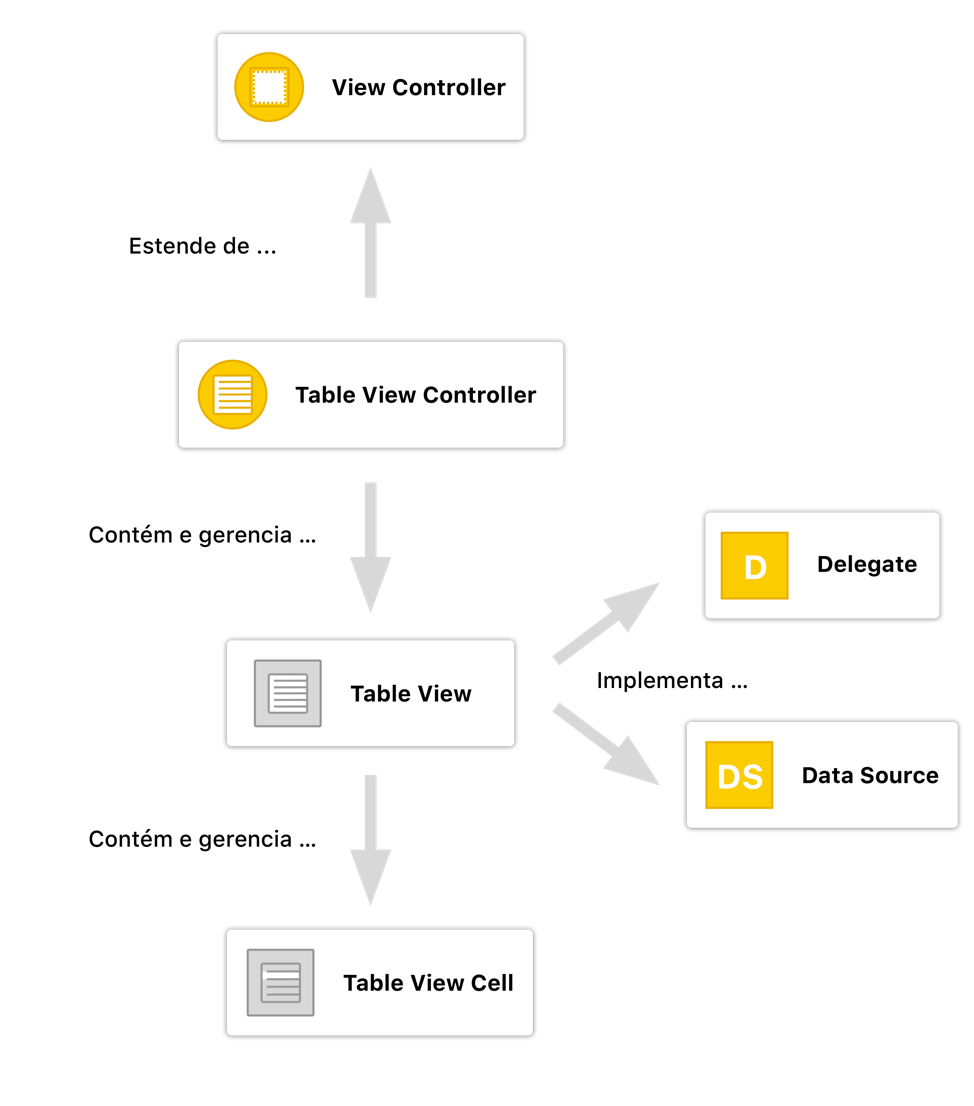


Figura 2. - Esquema da estrutura de uma UITableViewController

## Table View

A Table View é o componente mais importante da UITableViewController. É nela que o gerenciamento de listas acontece realmente.

Para inserir dados em uma lista é preciso primeiro definir um *layout* para a Table View Cell, que na prática é uma linha (*row*) da tabela (lista). No UIKit já temos alguns *layouts* pré-definidos de Table View Cell que você pode utilizar, caso não esteja esperando nenhum visual mais complexo. Veja alguns exemplos de estilo de células disponíveis por padrão:

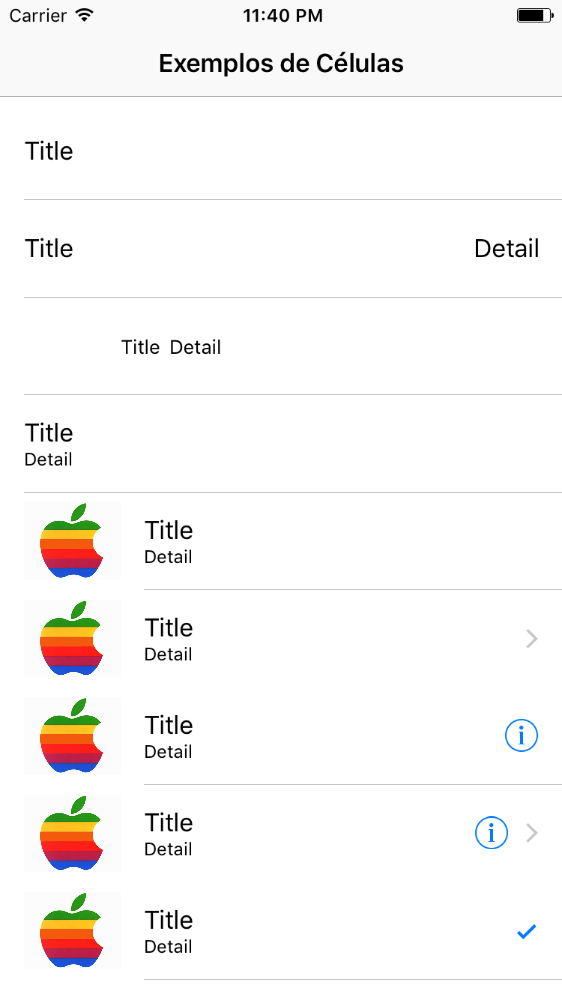


Figura 2. - Exemplos de células padrões

Antes de continuar a explicar o funcionamento das Table Views vamos entender o que é uma Table View Cell.

### Table View Cell

A UITableViewCell é uma *view* suportada pela Table View. Para exibir qualquer coisa em uma Table View devemos exibir mediante a uma Table View Cell. Você pode optar por escolher células com estilos padrões ou personalizar a sua própria. Para isso existe a opção de prototipar o *layout* da célula direto no *storyboard*, onde a tabela está inserida. Feito isto basta criarmos uma subclasse de UITableViewCell e informar a UITableView que é esta célula que deverá ser renderizada.

Aprenderemos a fazer tudo isto na prática.

Voltando as Table Views, para definir a célula que será renderizada, o número de células o tamanho das células, a quantidade de seções da tabela e outras coisas da configuração da tabela, é necessário implementar suas interfaces, ou melhor, protocolos, como são chamados no Swift.

Nesta aula iremos implementar alguns protocolos que levam o padrão de projeto Delegate, mas não entraremos em detalhes, pois ele será explicado com mais profundidade na Unidade 5.

## Implementando uma Table View Controller

**Material de apoio:**

O projeto de ponto de partida desta aula se encontra no arquivo **Unidade\_4\_-\_Aula\_2\_-\_Exemplo\_1.zip** da pasta de Material de apoio.

Nesta aula iremos implementar a tela **Navegar**. Ela é o tipo de tela que é gerenciada por uma lista e será perfeita para estudarmos. Para isso, abra o arquivo **Unidade\_4\_-\_Aula\_2\_-\_Exemplo\_1.zip**, fornecido por seu Instrutor. Ao final desta aula teremos o seguinte resultado:

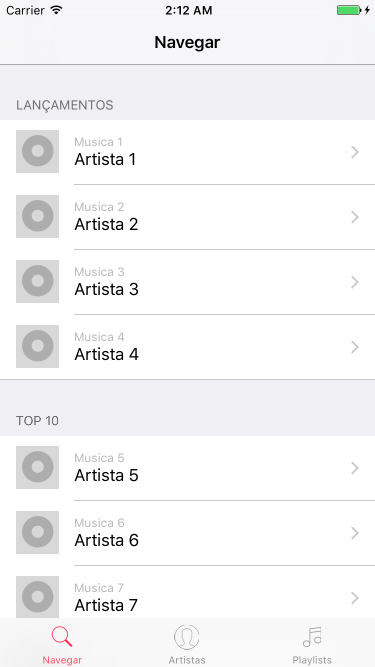


Figura 2.4 – Resultado esperado para o final da aula

1. Primeiramente veja que mudamos a cena **Navegar** e deixamos o *layout* da célula pronto para você:

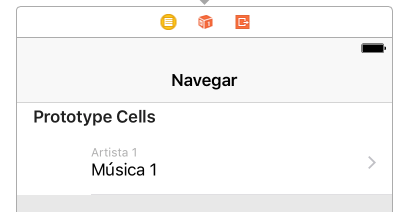


Figura 2.4 – Protótipo da célula

1. Se você preferir, pode formatar as Labels e o estilo da célula como desejar. Para facilitar, siga o estilo que preparamos.
2. Para nossa lista ficar dividida em seções com este visual, que é muito comum no iOS, devemos configurar seu estilo para **Grouped**. Para isto selecione a **Table View** e mude a propriedade **Style** para **Grouped:**

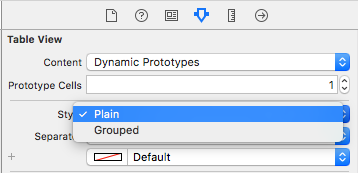


Figura 2.5 – Modificando o estilo de agrupamento da Table View

1. O *layout* da tabela irá mudar:

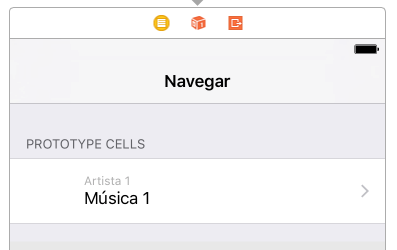


Figura 2.6 – Layout da Table View após aplicar o estilo **Grouped**

1. Outra coisa que precisamos aqui é definir um identificador da célula. Isto será útil para conseguirmos recuperar esta célula e manipulá-la no código. Então insira o valor **UITableViewCellIdentifier** no campo **Indentifier** da **TableViewCell:**

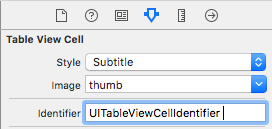


Figura 2.7 – Definindo um Identifier para a célula

1. Finalizamos a parte do Storyboard. Agora entre no arquivo **BrowseTableViewController.**
2. Neste arquivo você encontrará vários métodos. Todos eles pertencem ao UITableViewController e devem ser preenchidos para que a tabela possa exibir dados. Estes métodos seguem o padrão **Template Method.** Vamos implementar método á método. Primeiro veja que criamos um *array* com alguns dados que serão exibidos na tabela:

let items : [Section] = [

Section(name: "LANÇAMENTOS", musics: [

Music(name: "Musica 1", artist: "Artista 1"),

Music(name: "Musica 2", artist: "Artista 2"),

Music(name: "Musica 3", artist: "Artista 3"),

Music(name: "Musica 4", artist: "Artista 4")

]),

Section(name: "TOP 10", musics: [

Music(name: "Musica 5", artist: "Artista 5"),

Music(name: "Musica 6", artist: "Artista 6"),

Music(name: "Musica 7", artist: "Artista 7"),

Music(name: "Musica 8", artist: "Artista 8"),

Music(name: "Musica 9", artist: "Artista 9")

])

]

Este *array* é dividido em **Section** e **Music**, que são *structs* definidos neste mesmo arquivo. Cada Section tem um nome e um conjunto de objetos Music (que representa uma música) e cada Music possui um nome e um artista.

1. O primeiro método a ser implementado é o **numberOfSections(in:).** Nele definiremos o número de seções que a tabela conterá. Faremos isto oferecendo o número de elementos do *array* **items**, pois ele dita a estrutura de dados da tabela. Neste caso teremos o total de duas seções. (Lançamentos e Top 10). O método deverá corresponder com o seguinte código:

override func numberOfSections(in tableView: UITableView) -> Int {

return items.count

}

1. Agora vamos definir o número de linhas que cada seção terá com o método **tableView(numberOfRowInSection:).** Neste caso precisaremos pegar a quantidade de Musics que tem dentro de cada seção, então deixe o método da seguinte maneira:

override func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {

return items[section].musics.count

}

1. Agora localize o método **tableView(heigthForRowAt:).** Aqui definimos a altura de cada célula. Apenas retorne 65.

override func tableView(\_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {

return 65.0

}

1. Para definir o texto de cada seção, implemente o seguinte método:

override func tableView(\_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {

return items[section].name

}

1. Sobraram dois métodos. Vamos ao primeiro deles. O método **tableView(cellForRowAt:)** é onde a mágica acontece. Faça com que ele corresponda com o seguinte código:

override func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {

let cell = tableView.dequeueReusableCell(withIdentifier: "UITableViewCellIdentifier", for: indexPath)

let section = items[indexPath.section]

let music = section.musics[indexPath.row]

cell.textLabel?.text = music.name

cell.detailTextLabel?.text = music.artist

return cell

}

A primeira coisa que fazemos aqui é obter a célula através do identificador que definimos no *storyboard*. Em seguida desmembramos o *array* de itens para obter as músicas de cada célula, atentando-se a seção respectiva. Nos últimos passos preenchemos as *labels* da célula.

1. O último método é o **tableView(didSelectRowAt:), ele** é disparado quando tocamos (selecionamos) uma célula, então o que faremos aqui é abrir nossa cena de Música:

override func tableView(\_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {

performSegue(withIdentifier: "MusicViewControllerIdentifier", sender: self)

}

Você pode executar o *app* e ver o resultado. Você terá uma tela correspondente a imagem mostrada no início da atividade.

## Resumo

Nesta aula aprendemos finalmente a utilizar listas com as Table Views e Table View Controllers. Com este conhecimento poderemos finalizar todas as demais telas do nosso *app*, já que elas são baseadas em lista. Tente implementar todas as demais telas, pois isto será um bom treinamento. Vimos o termo **delegate** ou **delegação**. Este se trata de mais um padrão de projeto que iremos aprender na Unidade 5.

Aula 3

# Animações básicas

Inserir animações nos nossos *apps* garante que as transições entre os diferentes estados que nossas views se submetem fiquem mais fluídos e naturais. No iOS, animações são usadas exaustivamente para reposicionamento de *views*, mudança de seus tamanhos, inserção e remoção de *views* de uma hierarquia visual e ocultá-las.

## Por que utilizar animações?

Usar animações é opcional, mas ela garante uma experiência de uso por parte do usuário muito mais amigável do que um aplicativo que simplesmente troca os estados em um piscar de olhos.

Em computação, principalmente móvel, existe uma gigante área que estuda a fundo o comportamento e interação do homem com o computador, chamada Interação Humano-Computador (HCI). Ela vai tão a fundo, que se preocupa em saber como o subconsciente do usuário reage a uma interface gráfica. A HCI define como será a Experiência de Uso (UX) de um aplicativo. Com ela descobrimos que animações são de extrema importância.

Em um resumo, a utilização de animações garante um *feedback* visual mais natural e fluído, pois se comparado com a vida real, nada que sofre mudança, muda instantaneamente. Sempre há uma transição entre um estado e outro. Implementar esta semelhança com a vida real mexe com o poder cognitivo do usuário, e quanto menos ele precisar se adaptar, mas rápido ele vai aprender a utilizar seu *app* e o usará por mais tempo, pois não precisará lidar com frustrações e estranhezas que só o subconsciente dele percebe.

## O que pode ser animado

Como visto em aulas passadas, o kit de desenvolvimento do iOS possui um *framework* (ou biblioteca, ou ainda SDK) chamado **UIKit**. Este *framework* possui todas as classes e tratamentos para lidar com a interface gráfica. É nele que encontramos os componentes UIButton, UIView, UIControl, UIViewController e todas as classes que possuem o prefixo **UI.** É daí que vem estes prefixos.

DICA: A linguagem utilizada para o desenvolvimento iOS era o Objective-C antes do Swift tomar o lugar. Nela não era possível definir **namespaces**, ou pacotes, então adotava-se o costume de adicionar um prefixo de duas letras maiúsculas no nome da classe para identificar o contexto que ela estava inserida. A cada atualização do Swift estes prefixos estão sendo deixados para trás. As classes String e Dictionary, por exemplo, eram chamadas antigamente de NSString e NSDictionary pois faziam parte do “pacote” (entre aspas pois não existia pacotes mas separava-se as classes via prefixo) **NeXSTEP,** empresa criadora do Objective-C, fundada por Steve Jobs após ter sido demitido da Apple. A NeXSTEP estava ganhando espaço no mercado, enquanto a Apple estava cada vez mais na pior, até que a NeXSTEP foi agregada à Apple e Steve ganhou de volta o seu lugar de CEO.

No kit de desenvolvimento do iOS também temos o *framework* **Core Animation** que possui os tratamentos específicos e de mais baixo nível para manipulação de animações.

Ambos, UIKit e Core Animation, suportam animações mas o nível de suporte varia entre os dois. No UIKit, as animações são executadas por objetos de UIView. *Views* suportam um conjunto básico de animações, que felizmente cobrem a maioria das tarefas comuns que iremos nos deparar. Podemos animar as seguintes propriedades das *views*:

* **frame:** Modifique esta propriedade para alterar o tamanho e posição da *view* relativamente a ao sistema de coordenadas de sua *superview*;
* **bounds:** Modifique esta propriedade para alterar o tamanho da *view*;
* **center:** Modifique esta propriedade para mudar a posição de uma *view*, considerando o ponto de seu centro, relativamente ao sistema de coordenadas de sua *superview;*
* **transform:** Modifique esta propriedade para escalar, rotacionar ou transladar uma *view* relativamente ao seu próprio centro;
* **alpha:** Modifique esta propriedade para mudar gradualmente a transparência de uma *view*;
* **backgroundColor:** Modifique esta propriedade para mudar a cor de fundo de uma *view*;
* **contentStretch:** Modifique esta propriedade para mudar a maneira que uma *view* preenche os espaços disponíveis.

## Como animar as Views

Existem duas maneiras de animar *views*. A primeira é utilizando os métodos da UIView, que são utilizados para animações mais simples e a segunda, para animações mais complexas, utilizando as classes disponíveis no iOS 10: UIViewAnimating e UIViewPropertyAnimator.

Vejamos um exemplo simples de animação.

No desenvolvimento de apps iOS teremos que trabalhar principalmente com animações em ocultar ou mostrar uma View mediante a uma situação.

Para ocultar uma View basta atribuir o valor true à propriedade isHidden.

view.isHidden = true

Para tornar a view visível novamente basta atribuir o valor false à mesma propriedade.

Mas existe um problema. Quando fazemos isto a view é ocultada/revelada instantaneamente, causando uma estranhesa a quem vê. A solução que temos é deixar esta transição da visibilidade de uma view mais suave utilizando animação.

Para isto precisaremos de uma propriedade adicional: alpha.

O alpha define a transparência da view mas a diferenças entre o alpha e o isHidden são:

* O alpha recebe um valor Float entre 0 e 1 enquanto o isHidden é um booleano. O efeito disso é que o alpha tem níveis de transparência mas o isHidden possui apenas dois níveis 0% visível e 100% visível.
* O isHidden cancela a interação do usuário com a view e o alpha não faz nada a respeito disso, e a view poderá continuar recebendo eventos mesmo não estando visível na tela.

Então o procedimento padrão para ocultar uma view com animação é primeiro definir seu alpha para zero e em seguida ocultar a view. Vejamos como fazer isso:

1. Acesse o método de classe **animate()** da UIView e defina alguns argumentos mandatórios que irão configurar nossa animação:

UIView.animate(withDuration: 0.3, animations: {

}, completion: { (finished) in

})

Neste passo estamos definindo uma animação com 0,3 segundos de duração (é pouco mas faz muita diferença). Nesta animação também poderemos executar alguma instrução ao seu término.

1. Agora, dentro do bloco **animations** insira a instrução que deixará a view transparente:

UIView.animate(withDuration: 0.3, animations: {

self.view.alpha = 0

}, completion: { (finished) in

})

1. O último passo é ocultar a view no término da animação:

UIView.animate(withDuration: 0.3, animations: {

self.view.alpha = 0

}, completion: { (finished) in

if finished {

self.view.isHidden = true

}

})

1. Para tornar a view visível novamente basta utilizar a seguinte lógica

view.isHidden = true

UIView.animate(withDuration: 0.3) {

self.view.alpha = 1

}

### UIViewAnimating

A UIViewAnimating é um novo protocolo introduzido no iOS 10 para nos dar a possibilidade de animarmos as *views* de uma forma muito mais interativa. Ela define alguns métodos a implementar por parte dos animadores. Os animadores são definidos pela classe UIViewPropertyAnimator. Estes métodos implementam o controle de fluxo da animação, dando a opção de **iniciar, pausar** e **cancelar** animações.

Quando criamos um animador UIViewPropertyAnimator, este começa no estado **inativo** e para conseguirmos coloca-lo em execução, utilizamos os métodos **startAnimation()** e **pauseAnimation()** que vão leva-lo para o estado **ativo.** Com o método **stopAnimation()** cancelamos a animação em execução e as propriedades do objeto animado conservam o exato estado em que a animação o deixou. Veja abaixo o ciclo de vida de uma animação.

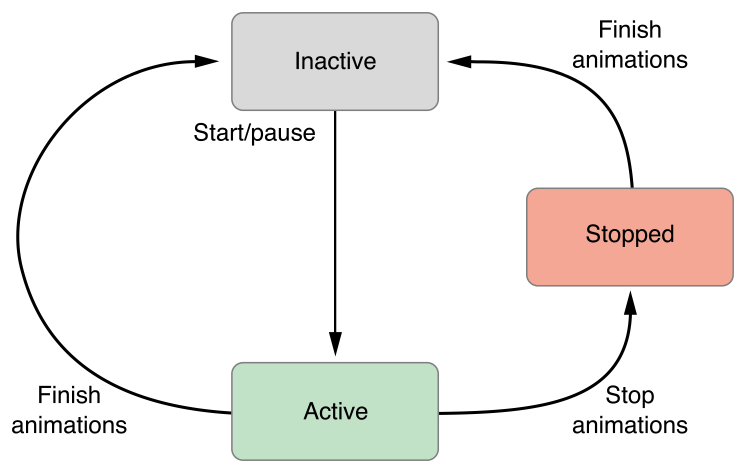


Figura 3.1 – Ciclo de vida de uma animação

A animação possui algumas propriedades importantes:

* **fractionComplete:** Mostra a porcentagem do avanço da animação (em um valor entre 0.0 e 1.0) em um determinado instante de tempo;
* **isReversed:** Indica se a animação está sendo executada ao contrário;
* **state:** Nos indica o estado UIViewAnimatingState atual da animação (ativo, inativo ou cancelado);
* **isRunning:** Nos permite identificar se a animação está em execução ou não em um instante de tempo.

Possui também alguns métodos importantes:

* **addAnimations:** permite incluir um bloco (ou *closure*, que iremos explicar em detalhes na Unidade 5) com as animações que desejamos realizar e uma duração desta animação indicada na propriedade **duration** do animador. Esta é uma chamada assíncrona que não retorna nenhum valor, ou seja, não precisamos esperar a animação terminar para realizar outras tarefas no *app*, pois ela irá executar em paralelo;
* **addAnimations:delayfactor:** Este é semelhante ao método anterior, com a diferença que podemos incluir via parâmetro um *delay*, ou **atraso** entre as animações. Este atraso será inserido quando a animação que está sendo adicionada começar a sua execução. Se introduzirmos um valor entre 0.0 e 1.0, ele se multiplicará pelo valor da propriedade **duration** do animador.

## Resumo

Nesta aula você foi apresentado as ferramentas de animações básicas do iOS. No inicio da aula conhecemos um pouco sobre a HCI que é a área de estudo da computação que se preocupa com a usabilidade e experiência de uso dos sistemas computacionais. Ela é muito importante, e quanto mais dominarmos esta área nossos aplicativos serão mais ricos, intuitivos e bem sucedidos. Como animações básicas, você aprendeu a implementar um bloco de animação. Este método é o mais convencional e é o que mais iremos encontrar nos aplicativos de até então. Em paralelo temos o UIViewAnimating que é uma ferramenta nova, do iOS 10, que acrescenta uma série de métodos e propriedades voltadas a manipulação de animações que irão facilitar a criação das mesmas.

## Exercícios

## TDP