Unidade 3

Aula 1

# Storyboards

Nas aulas anteriores você aprendeu um pouco da programação em Swift, agora é chegada a hora de entender como funciona a construção dos layouts em iOS. No curso de POO mencionamos Storyboards como sendo uma espécie de história em quadrinhos que reproduz o problema que o software irá resolver, e também como uma ferramenta de prototipagem. Em iOS, Storyboards não é necessariamente isto mas vamos ver muita semelhança com este conceito.

## O que é um Storyboard

Você está familiarizado com o Xcode e com o Swift, então está pronto para trabalhar em um Storyboard. Um storyboard é uma representação visual da interface de usuário do aplicativo, mostrando telas de conteúdos e as transições entre eles. Você usa storyboards para esquematizar o fluxo ou história que impulsiona a sua aplicação. Nele você vê exatamente o que você está será construído enquanto você o constrói (realtime), e obtém um feedback imediato sobre como sua UI (Interface de Usuário) está ficando.

Comparando com o Android, assemelha-se ao Canvas, com a diferença que no Storyboard você pode inserir mais de uma tela do seu app e definir a navegação/fluxo entre elas, ligando pontos.

Quando você cria um novo projeto, já existem dois Storyboards criados: o Main.storyboard e o LaunchScreen.storyboard. A LaunchScreen.storyboard é a primeira tela que seu aplicativo abre, e ela permanesce em foco enquanto o app faz os preparativos iniciais (veremos mais nas aulas seguintes). A Main.storyboard é uma storyboard comum criada por padrão em cada projeto. Você pode removê-la, renomeá-la ou ignorá-la pois não é obrigatória, mas por enquanto deixe-a como ela está.

## Criando um novo projeto

Ainda não aprendemos a criar um novo projeto! Então siga os passos abaixo.

1. Abra o Xcode e você encontrará esta tela:



Figura 1 - Tela inicial do Xcode

1. Para criar um novo projeto clique na opção **Create a new Xcode Project.**



Figura 2 - Escolha **Create a new Xcode Application**

1. Como em Android, temos vários modelos iniciais para nosso projeto. No nosso exemplo escolha a **Single View Application.**

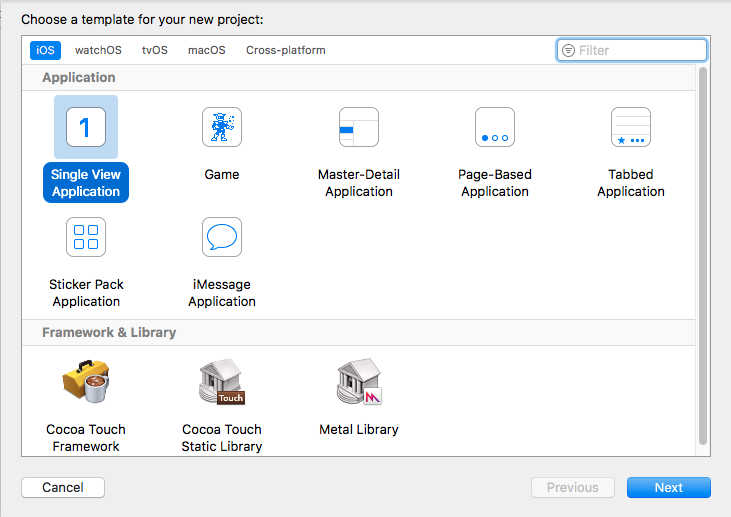


Figura 3 - Escolha **Single View Application**

1. Agora precisamos configurar o projeto. Insira as configurações conforme na imagem abaixo.

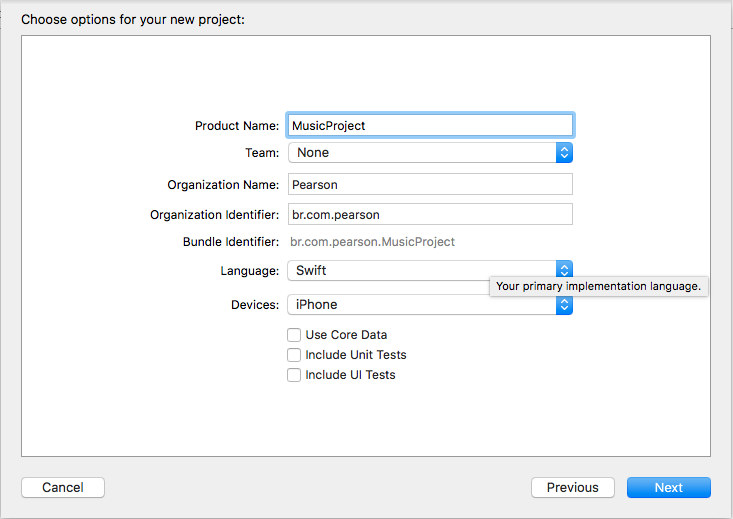


Figura 4 - Configurações iniciais do projeto

1. Como passo final, escolha o local onde será salvo o seu projeto.

## Conhecendo a Storyboard

Após criar um novo projeto, localize a Main.storyboard no painel de navegação e abra-a. Você terá a seguinte tela:



Figura 5 - Main.storyboard

Neste ponto, o storyboard em seu aplicativo contém uma cena, o que representa uma tela de conteúdo no seu aplicativo. A seta que aponta para o lado esquerdo da cena na tela é o ponto de entrada storyboard, significando que esta cena é a primeira cena a ser carregada ao iniciar este storyboard (pois podemos ter vários storyboards que se relacionam uns com os outros). Neste momento, a cena que você vê na tela contém uma única View que é gerenciada por um controlador de Views (View Controller). Você vai aprender mais sobre as funções das Views e View Controllers nas aulas seguintes.

## Prototipando com o Storyboard

Um ponto forte do Storyboard é que podemos já no início do projeto, realizar um protótipo utilizando as imagens provisórias do projeto. Neste curso iremos criar um aplicativo de músicas, onde você poderá listar, tocar e criar playlists de músicas, então para que você possa entender como será este aplicativo, vamos criar um protótipo rápido de parte dele.

Para criar este protótipo você precisará de algumas imagens inseridas no projeto.

### A pasta Assets

Localize uma pasta chamada **Assets.xcassets** no seu painel de navegação. Nesta pasta é onde iremos inserir todas as imagens do nosso projeto. Para adicionar imagens é um processo bem fácil. Basta arrastar as imagens para cá.

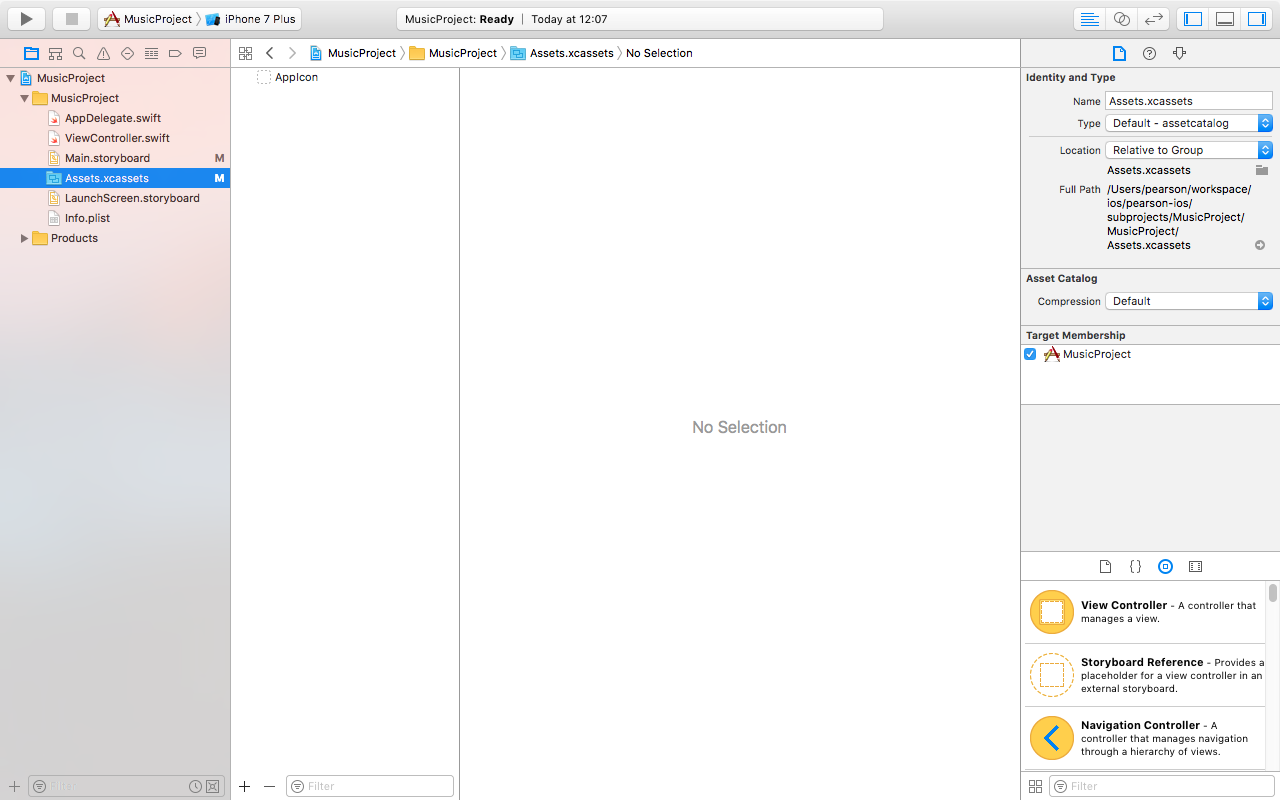


Figura 6 - Visualização da pasta Assets

Vamos colocar as imagens

**Material de apoio:**

As imagens desta aula se encontram no arquivo **assets\_unidade-3\_Aula-1.zip** da pasta de Material de apoio.

1. No Xcode deixe a pasta Asset.xcassets aberta
2. Entre na pasta onde estão contidas as imagens do material de apoio, selecione todas as imagens e arraste para o Xcode.
3. Você terá as seguintes imagens no Xcode.

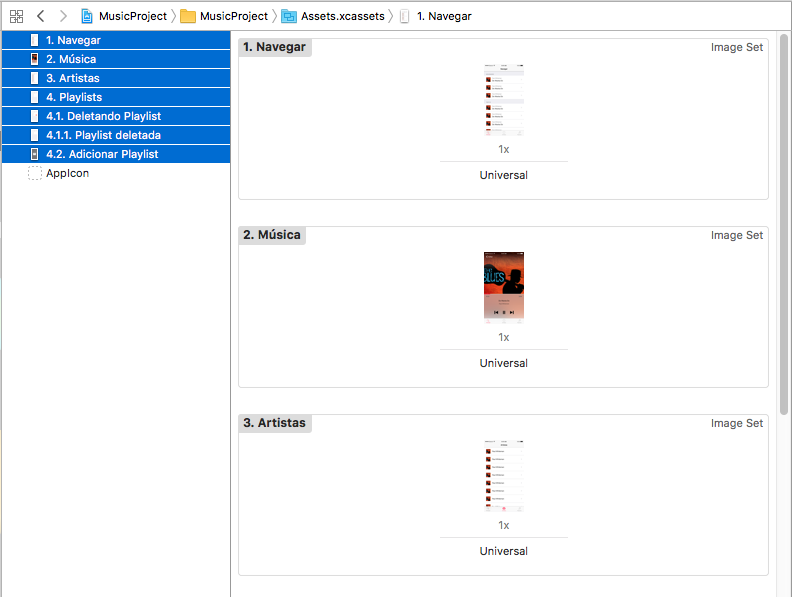


Figura 7 - Imagens adicionadas na pasta Assets

### Usando as imagens para criar os protótipos

Com as imagens adicionadas, vamos criar os protótiposde telas. Sigas os passos:

1. No Main.storyboard, mude o modo de visualização para iPhone 7. Para isto localize **View as** no rodapé de seu storyboard.

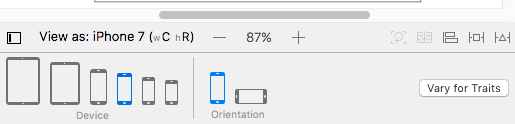


Figura 8 - Painel de Modos de Visualização do Xcode

1. Procure na paleta de componentes a **Image View**

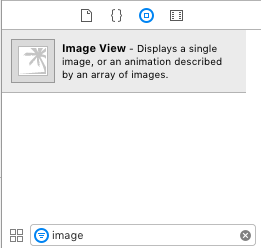


Figura 9 - Procurando Image View na paleta de componentes

1. Selecione a Image View e arraste-a e solte-a na cena.

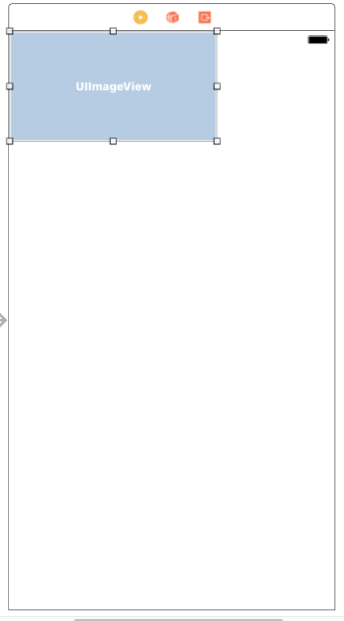


Figura 10 - Posicionando a Image View na cena

1. Agora redimensione esta Image View de modo que ela preencha a cena completamente.

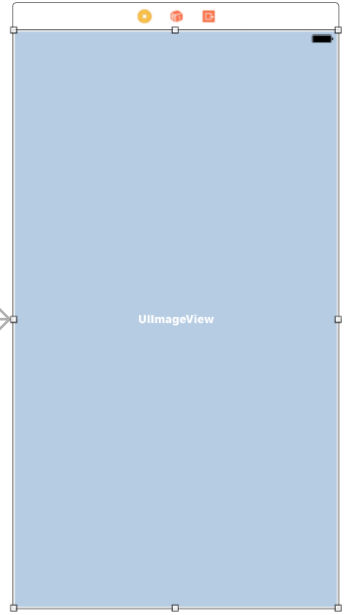


Figura 11 - Redimensionando a Image View na cena

1. Ainda com a Image View selecionada vamos definir a imagem que ela mostrará. Para isto veja no painel de propriedades a propriedade **Image** e insira o nome “1. Navegar” como na figura abaixo.

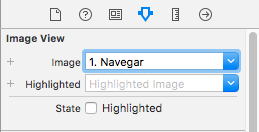


Figura 12 - Selecionando a imagem que aparecerá na Image View

1. Neste ponto você já poderá ver sua imagem sendo exibida na cena do storyboard
2. Para executar primeiro troque de simulador clicando no ícone destacado abaixo.



Figura 13 - Localizando o Simulador configurado

1. Nas opções que se abriram, clique em **iPhone 7.**

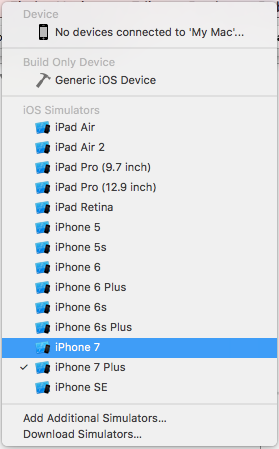


Figura 14 - Trocando para o simulador de iPhone 7

1. Agora basta executar o projeto clicando no ícone **Run** encontrado no canto superior esquerdo do sue Xcode.



Figura 15 - Botão Run (executar)

1. Seu primeiro app (ou quase) está pronto e executando no simulador.

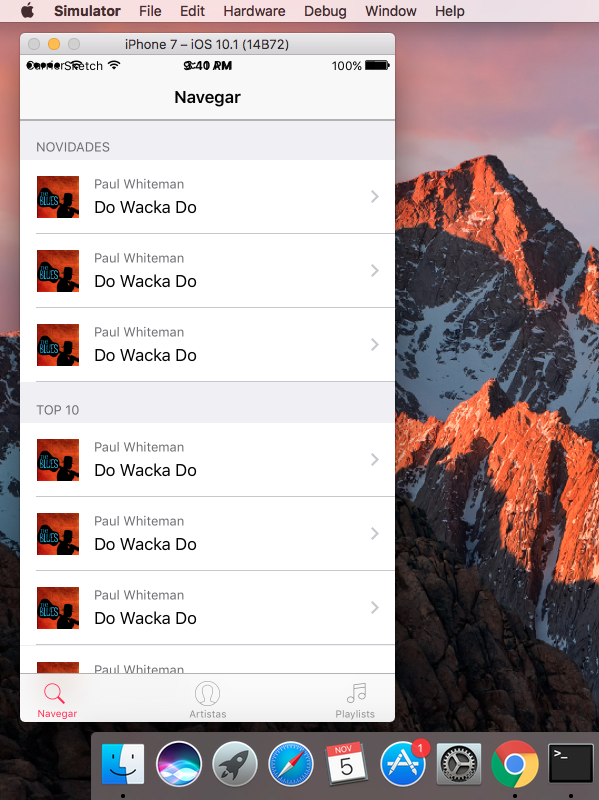


Figura 16 - Aplicativo sendo executado no simulador do iPhone 7

1. Você ainda precisa criar uma cena para cada imagem adicionada nos Assets. Para isto, localize a **View Controller** na paleta de componentes.



Figura 17 - Localizando a View Controller na paleta de componentes

1. Insira a View Controller no Storyboard.

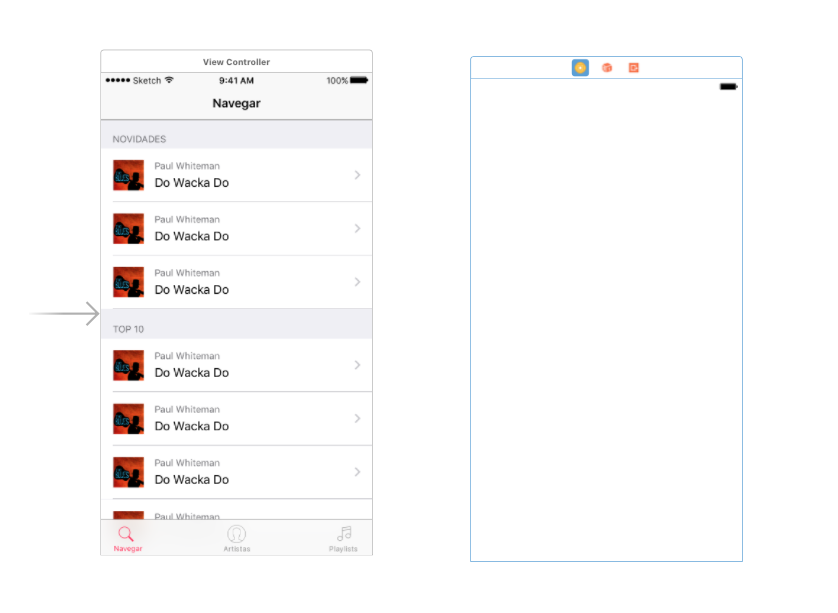


Figura 18 - Posicionando a View Controller no Sotryboard

1. Repita os passos de inserção de imagem (passos 2 à 5) para cada novo View Controller adicionado (total de sete View Controllers)
2. Seu Storyboard ficaá da seguinte maneira:

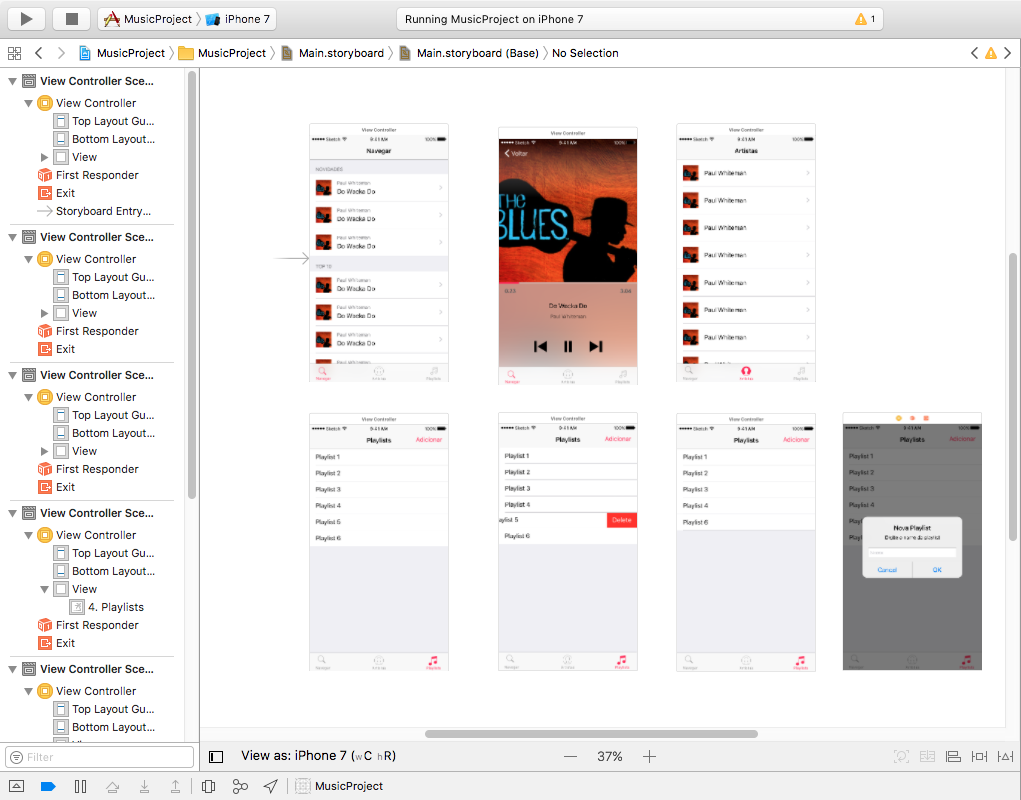


Figura 19 - Todas as View Controllers adicionadas com suas respectivas imagens

Note que todas as imagens são enumeradas. Esta numeração corresponde a ordem de fluxo de navegação destas telas. Antes de continuar a prototipagem, vamos entender alguns componentes utilizados aqui.

### UIImageView

A Image View, é uma view que exibe imagens no nosso app, como a do Android. Em iOS a sua classe corresponde a UIImageView (este é o nome da classe). A UIImageView View é uma subclasse de UIView, que entenderemos melhor na aula seguinte. Para que uma UIImageView exiba uma imagem,precisamos fornecer uma UIImage, que é a classe responsável por manipular arquivos de imagem do nosso projeto. No exemplo acima, para fornecer a UIImage precisamos apenas especificar o nome do arquivo de imagem.

### UIViewController

No Storyboard a UIViewController (ou View Controller) representa uma cena ou simplesmente uma tela do nosso app. Ela se assemelha as Activities do Android e veremos mais detalhes na aula seguinte.

### Continuando os protótipos

Você criou as cenas, mas agora precisamos liga-las entre si para construir a navegação do app. Vamos fazer isto provisoriamente usando botões (UIButtons). Então abra a sua Main.storyboard e entre na primeira cena que criamos.

1. Para inserir os botões, localize-o na paleta de componentes e araste-o para a cena. Coloque redimensione os botões para que correspondam com as áreas em verde nas imagens abaixo:

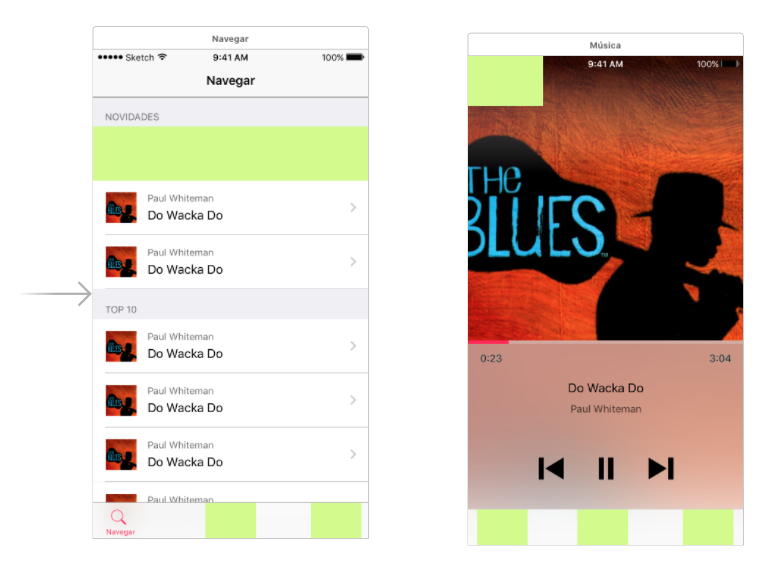


Figura 20 - Cena Navegar (esquerda) e Música (direita)

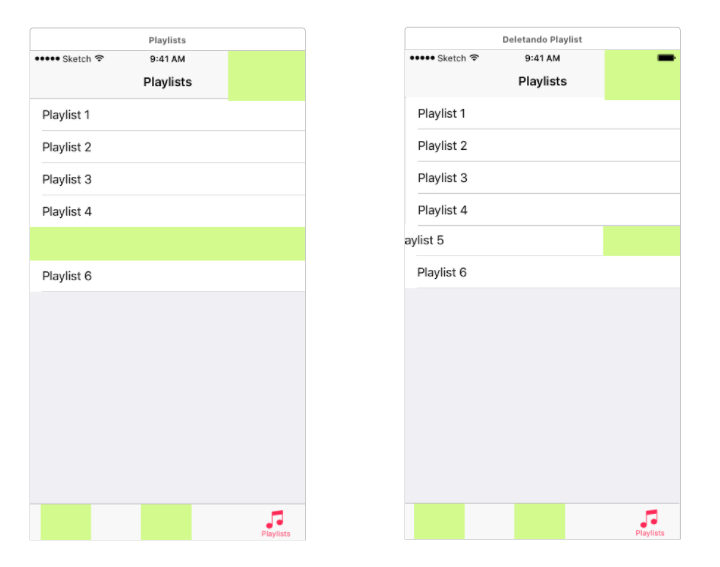


Figura 21 - Cena Playlists (esquerda) e Deletando Playlist (direita)

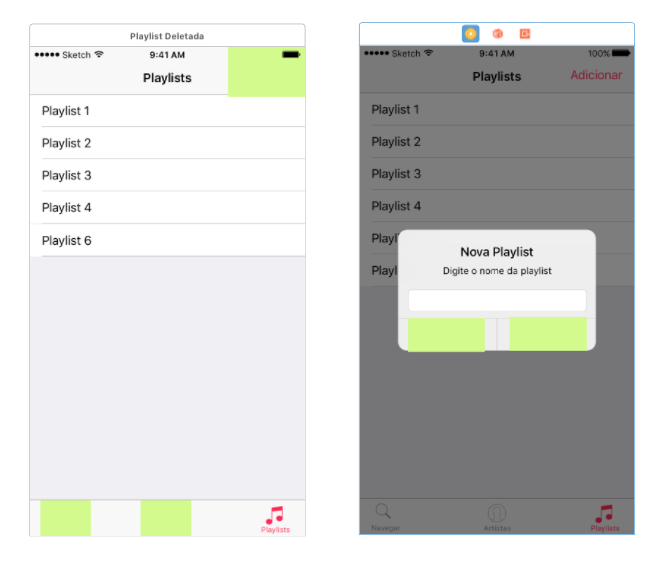


Figura 22 - Cena Playlist Deletada (esquerda) e Adicionar Playlist (direita)

1. No painel de propriedades de cada botão, remova o título do botão como na imagem abaixo:

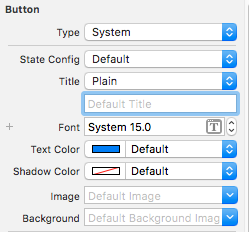


Figura 23 - Removendo o texto do botão

Desta forma você terá botões transparentes na tela. Se você preferi, pode mudar a cor do botão modificando a propriedade **Background** do botão para manter o controle dos mesmo, pois depois teremos que utilizá-los.

### UIButton

O UIButton é um botão da nossa interface gráfica. Semelhante ao Button do Android, o UIButton é um controle de entrada que recebe interação de toques do usuário. Com ele podemos capturar diverso tipos de toque, como **tocar e segurar, duplo toque, deslizar para alguma direção, tocar e arrastar.** No nosso exemplo iremos capturar os eventos de toques simples. O próximo passo para prototipagem é relacionar os botões e as telas de destino, isto é, devemos definir que tela nosso botão irá abrir ao ser tocado.

No iOS existem alguns eventos de entrada que não necessitam de uma sequer linha d código. A ação de abrir uma nova tela ao tocar em um botão é um destes eventos. Isto é feito graças as Segues.

### UISegue

A UIsegue ou simplesmente Segue é um link que estabelecemos entre uma cena e outra. Este link possui uma ação relacionada a ele, como por exemplo abrir uma tela. Vamos utilizar a Segue para você entender como funciona. Siga o passo-a-passo abaixo.

1. Vamos ligar o botão que foi colocado na primeira célula da cena **Navegar.**

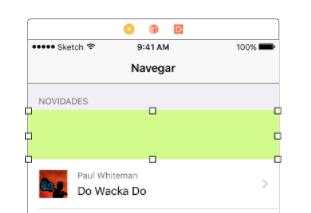


Figura 24 - Primeira célula da cena Navegar

1. Segure a tecla **Control** esegureo clique no botão acima, arrastando até a cena **Música**.



Figura 25 - Efeito ao clicar e arrastar um componente quando a tecla Control está pressionada

1. Ao soltar o clique na cena Música você terá as seguintes opções para escolher.

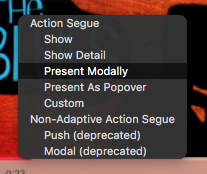


Figura 26 - Ações para a Segue

1. Escolha a opção **Present Modally**. Esta opção fará com que a Segue criada mostre a tela com uma animação que dá a impressão que uma tela sobe sobre a outra.
2. Ao realizar estas operações você poderá ver a Segue.

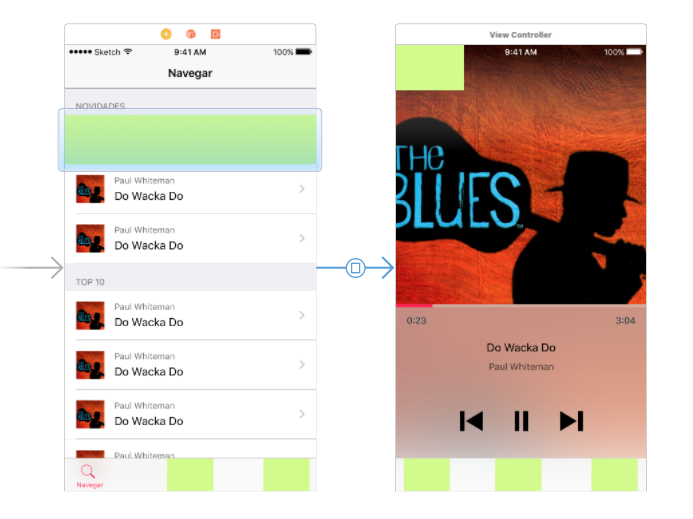


Figura 27 - Cenas vinculadas por uma Segue

1. Selecione a segue e no painel de propriedades desabilite a animação (em **Animates**) pois por enquanto ela não nos interessa. Você poderá executar o aplicativo para testar esta transição.

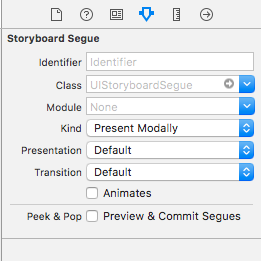
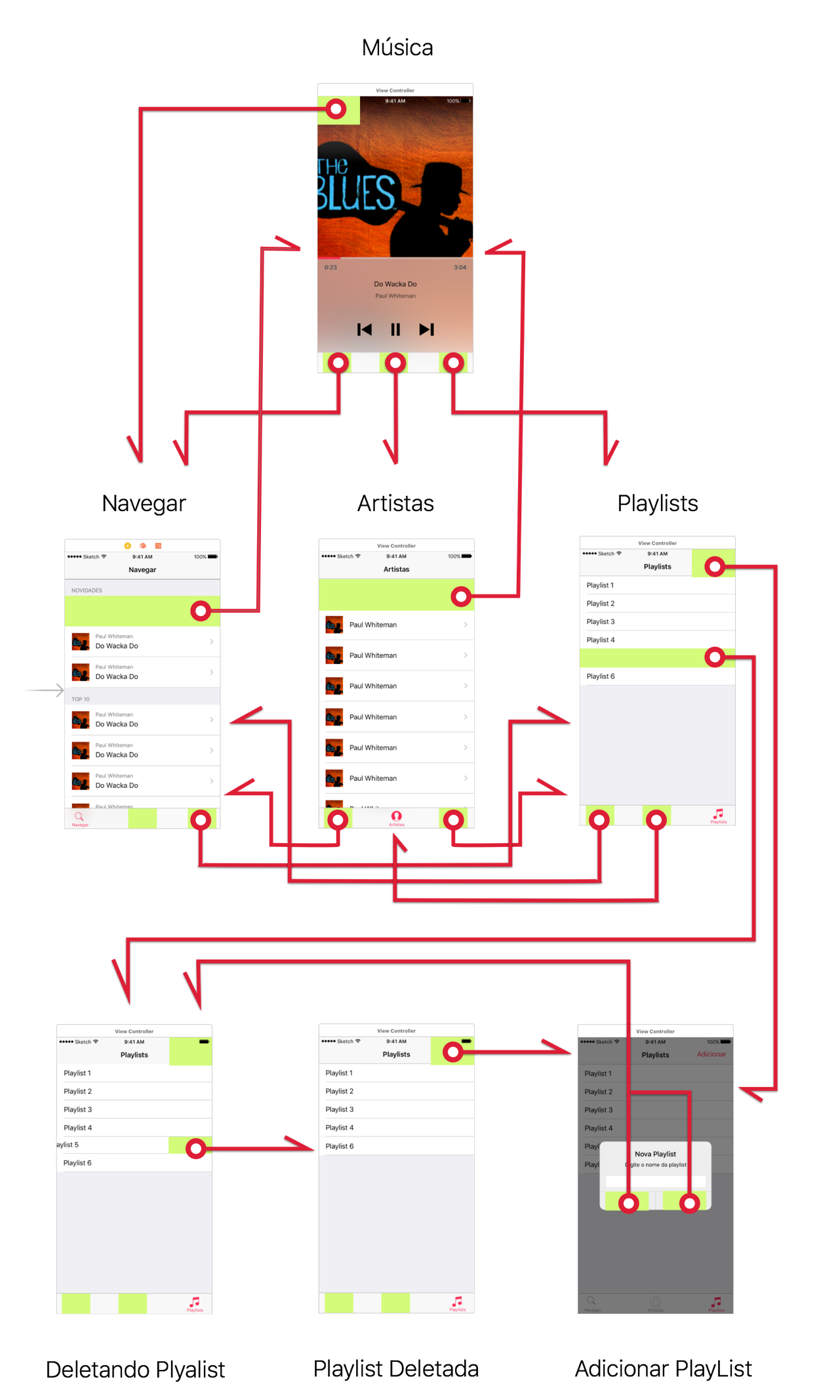


Figura 28 - Desabilitando animações

Este é o processo para ligar uma cena a outra. Para ligar todas as cenas use o mapa abaixo:



Agora temos nosso protótipo pronto. Veja que com ele nem precisamos explicar como o app funcionará e já conseguimos ter uma ideia. No decorrer das aulas iremos explicando as partes deste app.

## Resumo

Nesta aula vimos como é fácil prototipar e criar telas rápidas utilizando o Storyboard. Você conheceu três novos componentes que iremos utilizar muito nas aulas: a UIImageView, a UIViewController e a UISegue. Vimos também como é fácil transitar de uma tela para outra sem ao menos ter que escrever uma linha de código.

Aula 2

# Views e Auto Layout

O Auto Layout calcula dinamicamente o tamanho e a posição de todas as Views em uma View Controller, com base nas restrições (constraints) que colocamos nestas views. Por exemplo, podemos restringir que um botão seja centralizado horizontalmente em uma view e que uma Image View se posicione a 8 pontos abaixo da parte inferior do botão. Se o tamanho e posição do botão mudar, o posicionamento da imagem irá mudar também.

## Mudanças externas

Entendemos como mudanças externas, as mudanças que ocorrem quando o tamanho ou forma da superview são alteradas por algum evento fora do app. A cada mudança, devemos atualizar a hierarquia de layout a fim de reajustar os espaços disponíveis. Aqui estão alguns exemplos de mudança externa:

* O usuário entra ou sai da visualização de Split View em um iPad
* O dispositivo é rotacionado
* O usuário recebe chamadas telefônicas (isto altera o design dos apps)
* Quando o app apoia diferentes tamanhos de tela

A maioria destas mudanças podem ocorrer em tempo de execução, e exigem uma resposta dinâmica programada previamente do seu aplicativo. O tamanho da tela não irá mudar em tempo de execução, mas devemos criar um layout adaptativo para que nossos apps funcionem igualmente em um iPhone 4S, um iPhone 7 Plus, ou até mesmo um iPad.

## Mudanças internas

As mudanças internas ocorrem quando uma view que possui restrições e que afeta outra view é alterada, ou quando executamos animações visuais. Aqui estão algumas fontes comuns de mudança interna:

* O conteúdo exibido pelas mudanças de aplicativos.
* O aplicativo suporta internacionalização.
* O aplicativo suporta Fonte Dinâmica.

Quando ocorrem mudanças no conteúdo do seu aplicativo, este novo conteúdo pode exigir um layout diferente do antigo. Isto normalmente ocorre em aplicativos que exibem texto ou imagens dinâmicos. Por exemplo, uma aplicação de notícias necessita de ajustar a disposição do seu layout com base nos artigos a serem exibidos. Da mesma forma, a exibição de uma imagem deve ser exibida sem distorções ou perda de resolução, e com o auto layout tratamos isto.

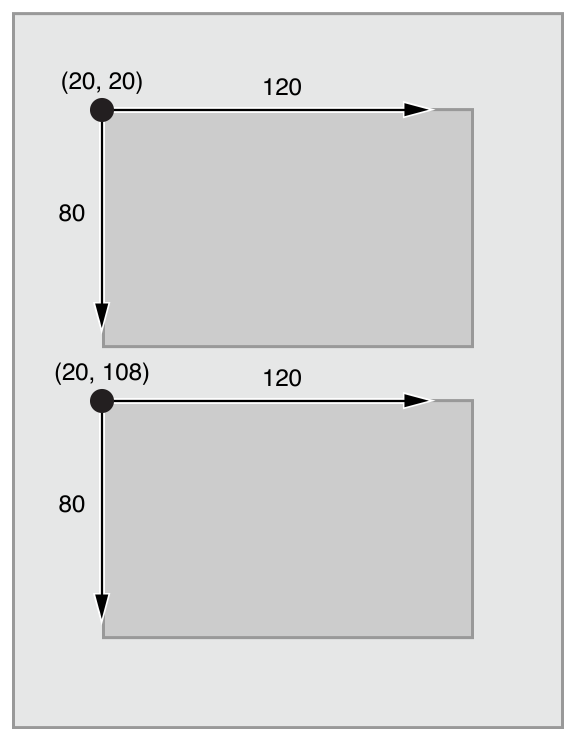
A internacionalização é o processo de fazer seu aplicativo se adaptar a diferentes linguagens, regiões e alfabetos. O layout de um aplicativo internacionalizado de levar estas diferenças em conta para que seja exibido corretamente para todas as línguas e regiões que o aplicativo suporta.

Finalmente, se o seu aplicativo iOS suporta fonte dinâmica, o usuário pode alterar o tamanho da fonte usada em seu aplicativo. Isso pode mudar a altura e a largura de quaisquer elementos textuais em sua interface de usuário. Se o usuário alterar o tamanho da fonte enquanto o aplicativo está sendo executado, ambas as fontes e o layout devem se adaptar.

## Disposição Auto Layout Versus baseada em Frame

Há três abordagens principais para conceber uma interface de usuário. Você pode programaticamente, ou seja, via código, implementar uma interface do usuário, você pode usar máscaras de redimensionamento automático para automatizar algumas das respostas à mudanças externas, ou você pode usar Auto Layout.

Tradicionalmente, os aplicativos desenhavam sua interface de usuário, definindo programaticamente uma moldura (frame) para cada view em uma hierarquia de views. Um frame define a origem, altura e largura da view no sistema de coordenadas da superview.



### Layout baseado em Frame

Em versões mais antigas do iOS, quando iríamos definir uma interface do usuário, tínhamos que calcular o tamanho e a posição de cada view na nossa hierarquia de views. Então, se uma mudança ocorresse, tínhamos que recalcular o frame para todas as views afetadas.

Definir os frames das views via código nos fornece o máximo de flexibilidade e poder, porém é uma tarefa árdua e muito suscetível a erros. Quando ocorre uma alteração interna ou externa, você pode literalmente fazer a mudança que desejar nas views. No entanto, devido ao fato de que nós mesmos temos que gerenciar todas estas mudanças, mesmo que esta defina uma interface de usuário simples, esta tarefa acaba requerendo uma quantidade de esforço consideravelmente alta na hora de projetar, depurar e manter seu código. Criar uma UI rica e adaptativa aumenta a dificuldade em uma ordem de magnitude.

### Layout baseado em máscaras de redimensionamento automático

Você pode usar máscaras de redimensionamento automático (autoresizing mask) para ajudar a aliviar um pouco deste esforço. Uma máscara de redimensionamento automático define a maneira como o frame de uma view se altera quando o frame de sua superview é alterado. Isto simplifica a criação de layouts que se adaptam a mudanças externas.

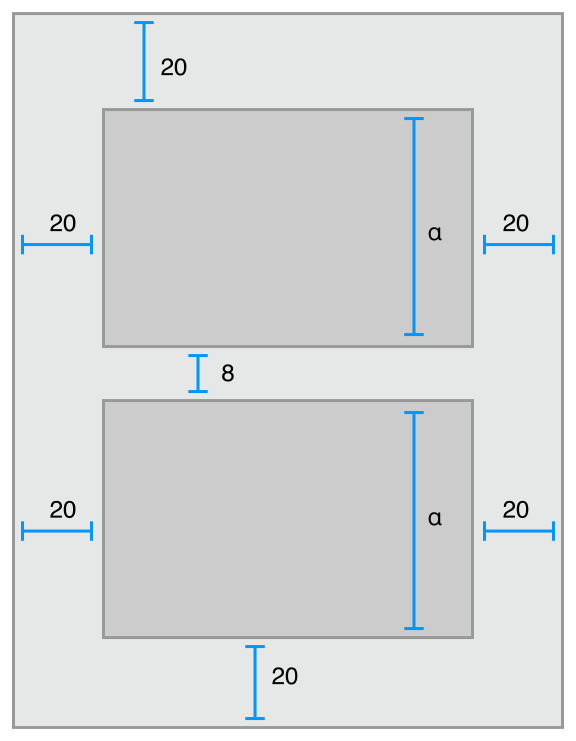
DICA: Você pode encontrar mais detalhes sobre autoresizing mask em: <https://developer.apple.com/reference/uikit/uiview/1622559-autoresizingmask>

No entanto, as máscaras de redimensionamento automático apoiam um subconjunto relativamente pequeno de possíveis layouts. Para interfaces complexas, normalmente precisamos de aumentar nossas máscaras com nossas próprias mudanças programáticas. Além disso, as máscaras de redimensionamento automático se adaptam apenas à mudanças externas. Elas não suportam mudanças internas.

Embora estas máscaras são apenas uma melhoria iterativa em layouts programáticos, o auto layout representa paradigma inteiramente novo. Em vez de pensarmos em como criar a estrutura das views, pensamos sobre em elas irão se relacionar entre si.

### Auto Layout

O Auto Layout define a sua interface de usuário usando restrições. Restrições normalmente representam uma relação entre duas views. O auto layout, em seguida, calcula o tamanho e a localização de cada view com base nessas restrições. Isso produz layouts que respondem dinamicamente às mudanças internas e externas.



A lógica usada para projetar um conjunto de restrições para criar comportamentos específicos é muito diferente da lógica usada para escrever código procedural ou orientada a objeto. Felizmente, dominar auto layout não é diferente de dominar qualquer outra tarefa de programação. Há dois passos básicos: Primeiro você precisa entender a lógica por trás dos layouts baseados em restrição, e então você precisa aprender a API.

É possível definir as restrições para auto layout via código, mas neste curso aprenderemos apenas a definir restrições via Interface Builder, que cobre quase 100% dos casos que você irá se deparar na construção de layouts.

Então para entender melhor, vamos a prática.

## Aplicando o Auto Layout

Para você entender o real motivo de utilizarmos o auto layout basta abrir nosso projeto **MusicProject** e executá-lo com outro simulador diferente do iPhone 7. Veja o resultado da tela Música no emulador do iPhone 7 Plus:



Figura 29 - Cena Música executada no iPhone 7 Plus em modo retrato

No simulador tecle a combinação **Control + .** O simulador irá entrar em modo paisagem (landscape) e o resultado será este:

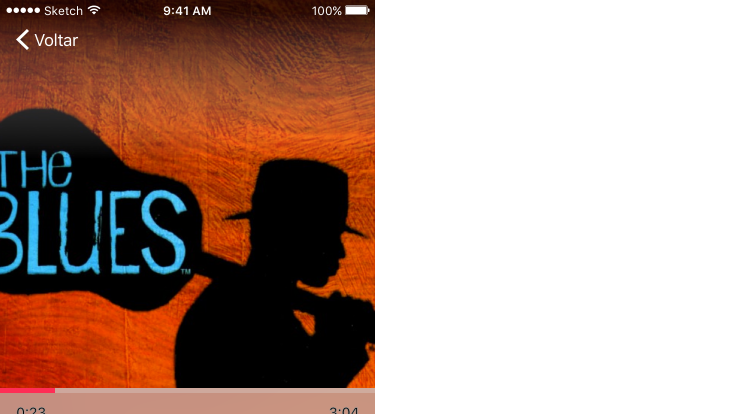


Figura 30 - Cena Música executada no iPhone 7 Plus no modo landscape

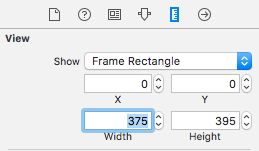
Obviamente este protótipo não será adaptável pois o proposito dele é apenas mostrar a navegação do app, então vamos começar a criar a cena Música, definitivamente.

**Material de apoio:**

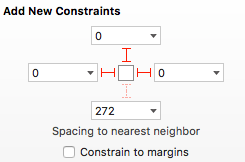
Para dar início a esta aula prática utilize o projeto contido em **Unidade\_3\_-\_Aula\_1\_-\_Exemplo\_1.zip**.

1. No storyboard Main, adicione uma UIImageView à com as seguintes dimensões:
   1. X: 0
   2. Y: 0
   3. Width: 375
   4. Height: 395

DICA: Para alterar as dimensões da view, selecione-a e no painel de propriedades selecione a aba **Size Inspector** (). Nesta aba você pode localizar as propriedades da imagem abaixo e alterar as dimensões da view.

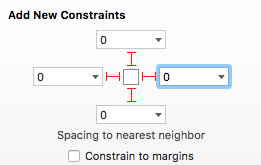


1. Agora vamos inserir restrições nesta Image View. No interface builder localize no canto inferior direito, a aba **Add New Constraints** (). Uma tela irá saltar no interface builder:
2. Para definir as restrições, precisamos primeiro pensar no objetivo que queremos alcançar. Para esta Image View precisamos que ela possua zero de espaçamento entre o topo, lado direito e lado esquerdo da superview. Então selecione as restrições conforme na imagem abaixo:



DICA: Veja na imagem acima que as linhas em vermelhos representam as restrições ativas e em vermelho tracejado, a restrição não ativa. Para ativar a restrição, basta clicar no tracejado. Neste caso ativamos a Top, Left e Right. Veja que possuem valores nas restrições. Estes valores representam o espaçamento em si que a restrição irá impor.

1. Agora procure uma UIVisualEffectView na paleta de componentes e adicione ao interface builder. Suas dimensões deverão corresponder com as informações abaixo:
   1. X: 0
   2. Y: 396
   3. Width: 375
   4. Height: 271
2. As restrições desta nova view devem corresponder com a imagem abaixo:



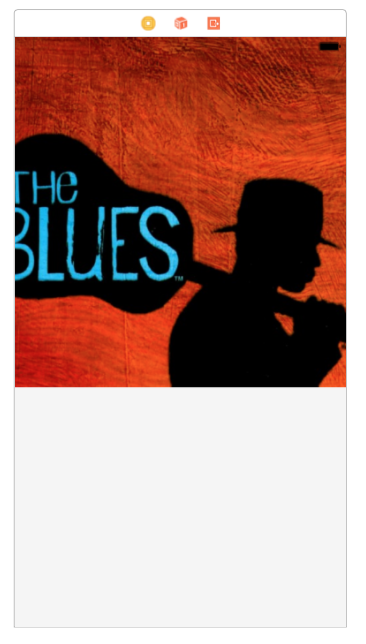
1. Ative as restrições clicando em:



1. Esta view irá conter uma altura fixa também. Para isto vá em Add New Constraints e ative a seguinte restrição:



1. Ainda nesta view, mude a cor de fundo na propriedade **Background** para a cor **EBEBEB**
2. O resultado até então é o seguinte:

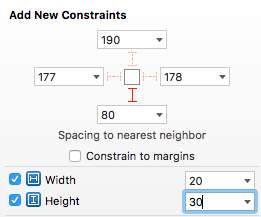


1. Execute o app em vários simuladores em diferentes orientações e veja o resultado. As views irão se adaptar.



DICA: Este é realmente o resultado esperado. A imagem se manteve aderia as bordas e a view inferior também, além disso esta view manteve sua altura fixa.

1. Agora iremos inserir os controles do Player. Adicione uma UIImageView como subview da nossa View inferior.
2. Nesta Image View escolha a imagem **ic\_pause.png**
3. Agora iremos definir as restrições desta imagem. Selecione a Image View e vá em Add New Constraints.
4. Selecione apenas a restrição inferior com valor 80.
5. Ainda na tela de definição de restrições, ative **Width** com valor 20 e **Height** com valor 30, segundo a imagem abaixo.

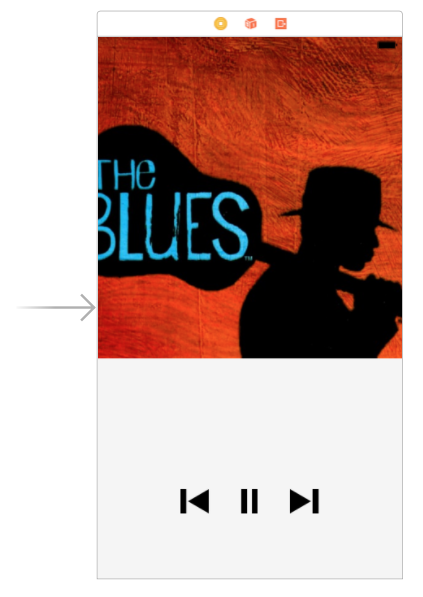


1. Ao aplicar estas restrições, veja que algumas linhas vermelhas, amarelas e azuis aparecem na Image View:



DICA: As linhas vermelhas indicam que as restrições não foram definidas corretamente e certamente irão falhar na execução do app. As linhas amarelas mostram que a view ficará diferente do interface builder em tempo de execução. Para corrigir as linhas amarelas, basta selecionar a view que apresenta estas linhas e clicar no botão **Update Frames** () no canto inferior direito do nosso interface builder.

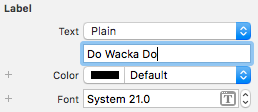
1. Quando estas linhas aparecem, precisamos corrigi-las. No caso, está faltando restrição de posição X. Queremos que nossa Image View fique centralizada verticalmente na View, então localize a aba **Add New Alignment Constraints** selecione a opção **Horizontally in Container.**
2. Ao aplicar você verá as linhas amarelas, então atualize os frames. Adicione outra Image View, ao lado direito da Image View com ícone de pause.
3. Configure esta Image View com a imagem **ic\_foward.png.**
4. Aplique as seguintes restrições.
5. Agora precisamos alinhar a imagem **forward** com a imagem **pause,** horizontalmente. Selecione a imagem **foward** clique e arraste com a tecla **Control** pressionada, e solte no botão pause.
6. Algumas opções irão se revelar. Escolha a opção **Center Vertically.**
7. Faça o mesmo para a imagem **ic\_preview**, posicionando-o ao lado esquerdo da imagem **pause.**
8. Você terá o seguinte resultado:



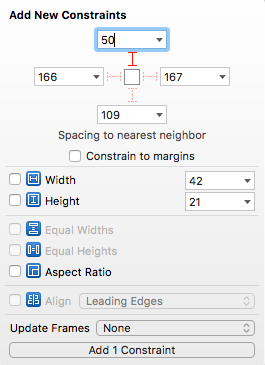
1. Precisamos ainda inserir os textos. Para isto, procure na paleta de componentes o a UILabel e arraste para a cena, como subview da view inferior (a mesma que colocamos as imagens de pause, forward, preview). Insira duas UILabels.

DICA: Uma UILabel é uma view que exibe texto. É similar ao TextView do Android.

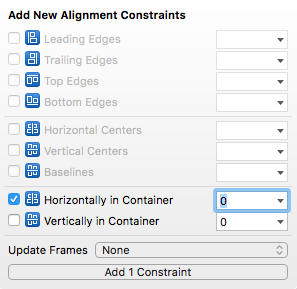
1. Na primeira Label iremos colocar o título da música que será tocada nesta cena (sim, ao final deste curso esta tela irá tocar músicas), então no painel de propriedades escreva um nome de uma música de sua preferência. Neste exemplo iremos utilizar uma música de domínio público chamada **Do Wacka Do** de Paul Whiteman. Deixe a fonte com tamanho 21. Veja na imagem abaixo.



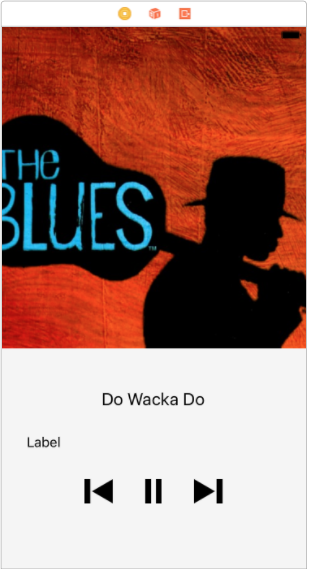
1. Vamos definir as restrições desta label. Vá em **Add New Constraints** e defina as restrições baseadas na imagem abaixo:



1. Em seguida, defina as restrições em **Add New Alignment Constraints** conforme a imagem abaixo:



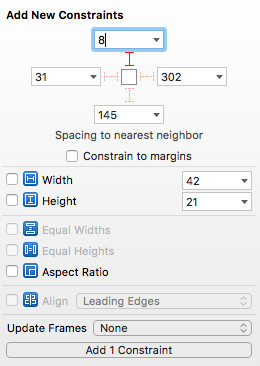
1. Atualize os frames para ter o seguinte resultado:



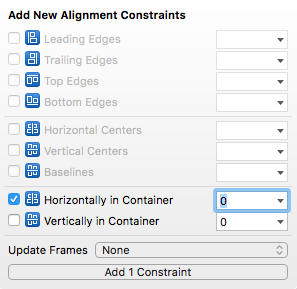
1. Na segunda label iremos colocar o nome do artista. Neste caso colocamos **Paul Whiteman.** Deixe a fonte com tamanho 17. Veja na imagem abaixo.



1. Vamos definir as restrições desta label tambem. Vá em **Add New Constraints** e defina as restrições baseadas na imagem abaixo:



1. Em seguida, defina as restrições em **Add New Alignment Constraints** conforme a imagem abaixo:



1. Atualize o frame e o resultado final da nossa aula será este:



1. Quando executarmos ele no simulador e em modo paisagem teremos o seguinte resultado:



## Resumo

Ainda há muitas possibilidades com auto layout. Para poupar tempo, nós terminaremos esta tela para você, mas não deixe de explorar e brincar com as possibilidades do auto-layout.

Nesta aula aprendemos o fundamental do auto layout e fizemos nossa primeira tela adaptativa utilizando o auto layout.

Aula 3

# Controles e eventos de entrada

Até então estivemos focados ao visual do aplicativo, compondo telas e evitando o máximo escrever qualquer linha de código. Você conseguiu dar a ação simples de abrir telas a alguns botões, mas o mundo dos apps é muito mais do que isto. Nesta aula você aprenderá a dar ações personalizadas aos botões. Iremos também conhecer os diversos controles e eventos de entrada que irão enriquecer a experiência de uso no app.

## Controles de entrada

As metodologias e arquitetura de desenvolvimento Android e iOS são bem similares. O conceito básico de desenvolvimento mobile inevitavelmente (e felizmente) é o mesmo para as grandes plataformas (iOS, Android e Windows Phone). Os controles de entrada são comuns nestas plataformas.

Como em Android, os controles de entrada são componentes visuais que estão prontos para receber interação do usuário. Estes componentes podem receber toques, texto ou voz.

Vejamos alguns exemplos.

### UIControl

O UIControl é um componente chave no iOS. Ele é uma subclasse da UIView que implementa diversos eventos de entrada. Todos os outros controles de entrada listados nesta aula são subclasses do Control. Com o Control podemos manipular eventos estipulados pelas constantes definidas pela enumeração UIControlEvents. Veja alguns exemplos de Control Events:

* **Touch-Down**: O touch-down é o momento que o usuário encosta o dedo na tela.
* **Touch Drag Inside:** É um evento de quando o dedo é deslizado dentro dos limites do Control.
* **Touch Drag Outside:** É um evento de quando o dedo é deslizado para fora dos limites do Control.
* **Touch Drag Enter:** É um evento de quando o dedo é deslizado para dentro dos limites do Control.
* **Touch-Up Inside:** Evento de tocar e tirar o dedo de cima do Control. Este é o evento mais comum que usaremos
* **Value Changed:** Evento disponível para alguns componentes como o **UISwitch** que é disparado quando seu valor é alterado. No UISwitch, que é uma view de **habilitar/desabilitar** algo, o evento value changed quando o estado desta view é alterada para habilitado ou desabilitado.

Exitem muitos outros eventos que podem ser conferidos na documentação oficial do UIControlEvent.

O UIControl não é um componente acessível pela paleta de componentes pois visualmente não apresenta nenhuma diferença de uma UIView simples, mas podemos utilizá-lo estendendo-o para subclasses. Também é possuivel atribuir esta classe para UIViews no Interface Builder.

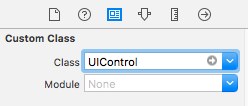


Figura 31 - Método para definir um UIControl através do Interface Builder

### UIButton

O UIButton, ou simplesmente botão, é o principal controle de entrada em um aplicativo iOS. Este é um botão que recebe o toque do usuário e mediante a isto podemos executar uma atividade no app.

Os eventos de entrada de um UIButton estão disponíveis graças a classe UIControl, que é a sua superclasse.



Figura 32 - UIButton sendo mostrado na paleta de componentes do Xcode

A diferença principal entre um UIButton e um UIControl é que o botão já possui componentes pré-definidos como um **título** e uma **imagem,**  e sendo assim não aceita subviews.

### UITextField e UITextView

Com o UITextField, ou simplesmente Text Field, o usuário pode inserir textos e outros tipos de dados. Em comparação com o Android, o UITextView se assemelha ao EditText.

Em sua forma padrão, o Text Field apresenta um teclado, mas podemos implementar nosso próprio método de entrada, como uma caixa de combinação, um seletor de data e etc.



Figura 33 - UITextField sendo mostrado na paleta de componentes do Xcode

O UITextView é utilizado para inserção de textos grandes em múltiplas linhas. Estes dois elementos possuem protocolos (iremos aprender mais sobre isto nas aulas seguintes), ou interfaces a serem implementadas que definem o que acontece quando um texto é inserido, quando um texto é apagado, quando o campo entra em foco, ou quando o campo perde seu foco. Isto possibilita criarmos máscaras de entrada, controlar a ordem de tabulação dos campos e etc.

DICA: Uma máscara de entrada é uma expressão regular que formata a entrada de acordo com seu objetivo. Por exemplo, uma máscara de telefone impede que o usuário insira letras ou mais caracteres do que o necessário para um número telefônico.



Figura 34 - UITextView sendo mostrado na paleta de componentes do Xcode

### Outros

O Xcode disponibiliza uma série de outros componentes de entrada prontos para serem utilizados. Veja um resumo de outros componentes bastante utilizados

|  |  |
| --- | --- |
|  | **Swich –** Apresenta um componente com estado booleano. Quando ele é tocado o estado é trocado. |
|  | **Segmented Control –** Exibe múltiplos segmentos, cada um com sua função como botões discretos. No iOS utiliza-se o Segmented Control como abas, semelhante ao Tabbar do Android. |
|  | **Slider –** Apresenta uma quantidade continua de valores e permite a seleção de um destes valores pela ação de deslizar o dedo neste controle |
|  | **Page Control –** É um controle utilizado para alternar entre View Controllers com um gesto de deslizar para direita ou esquerda |
|  | **Stepper –** É uma interface de usuário utilizada para incrementar ou decrementar um valor. |
|  | **Scroll View –** Providencia um mecanismo para mostrar um conteúdo que é maior de que se pode caber na tela da aplicação. Se o conteúdo exceder o tamanho da tela, o Scroll View permite que o usuário deslize o conteúdo. |

## Eventos de entrada

Além dos eventos de entrada controlados pelo UIControl (UIControlEvents), podemos definir nossos próprios eventos personalizados, como gestos, movimento do smartphone e toques com pressão.

### Reconhecimento de gestos

No iOS, podemos implementar reconhecimento de gestos. Em algumas situações é necessário pegar o evento de toque em elementos que não são UIControl, por exemplo, imagine que queremos realizar uma ação ao segurar o dedo sobre o título da tela; a UINavigationBar (que veremos nas aulas seguintes) é uma view que dentre outras funcionalidades, apresenta o título da tela, e ela não é uma extensão de UIControl, então precisaríamos implementar e inserir o reconhecimento de toques nela.

Para adicionar reconhecimento de gestos a uma view, basta instanciar um tipo de reconhecimento de gesto e atribuí-lo à view utilizando o método **addGestureRecognizer().**

let recognizer =

UITapGestureRecognizer(target: self, action: #selector(viewDidTouched))

view.addGestureRecognizer(recognizer)

No código acima estamos instanciando um reconhecimento de toques e adicionando a uma UIView simples.

DICA: Na instanciação do reconhecedor de toques é necessário fornecer uma ação (action) via argumento, e para isto utilizamos o #selector. Com o selector podemos selecionar qualquer método que esteja dentro do contexto do objeto que foi passado via argumento target. Neste exemplo, quando a view for tocada o método viewDidTouched definido na própria classe (target: self) será executado.

Existe uma grande variedade de reconhecedores de gestos que podem ser utilizadas. A UITapGestureRecognizer é classe mais utilizada para estes objetivos. Veja mais reconhecedores de gestos na documentação oficial do iOS.

### 3D Touch

O 3D Touch deixa o app apto a medir a pressão exercida sobre sua tela e oferecer diferentes opções de interação de acordo com a intensidade do toque.

É importante ressaltar que o 3D Touch é um recurso de hardware, e só está disponível nas versões mais novas do iPhone (iPhone 6s e superiores).

Na prática, podemos mostrar opções ou realizar ações de acordo com a força que o usuário toca na tela.

### Reconhecimento de movimento

Usuários estão constantemente gerando eventos de movimento quando se movem, agitam ou inclinam o dispositivo. Estes eventos de movimento são detectados pelo dispositivo de hardware, especificamente, pelo acelerómetro e o giroscópio.

O acelerômetro é na verdade composta de três acelerômetros, um para cada eixo-x, y, e z. Cada um mede mudanças de velocidade ao longo do tempo ao longo de um caminho linear. Combinando todas os três acelerômetros permite detectar o movimento do dispositivo em qualquer direção e obter orientação atual do dispositivo. O giroscópio mede a velocidade de rotação em torno dos três eixos.

Todos os eventos de movimento se originam o mesmo hardware. Existem várias maneiras diferentes que você pode acessar os dados de hardware, dependendo das necessidades do seu aplicativo:

* Se você precisa para detectar a orientação geral de um dispositivo, mas você não precisa conhecer o vetor de orientação, use a classe UIDevice.
* Se você quer que sua aplicação responda quando um usuário sacode o dispositivo, você pode usar o os métodos de manipulação de eventos de movimento do pacote UIKit para obter informações a partir do objeto UIEvent recebido.
* Se nem o UIDevice e nem as UIEvents são suficientes, é provável que você queira utilizar a Core Event Framework para acessar o giroscópio e o acelerômetro.

As referências de todas estas classes citadas podem ser encontradas na documentação oficial do iOS. Vamos logo a prática.

## Adicionando ações aos nossos botões

**Material de apoio:**

Para dar início a esta aula prática utilize o projeto contido em **Unidade\_3\_-\_Aula\_3\_-\_Exemplo\_1.zip**.

Primeiramente vamos ver as mudanças que temos. No painel de navegação temos um novo arquivo. O arquivo ViewController.swift foi substituído por MusicViewController.swift:

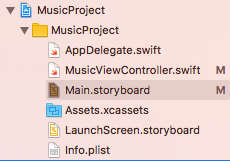
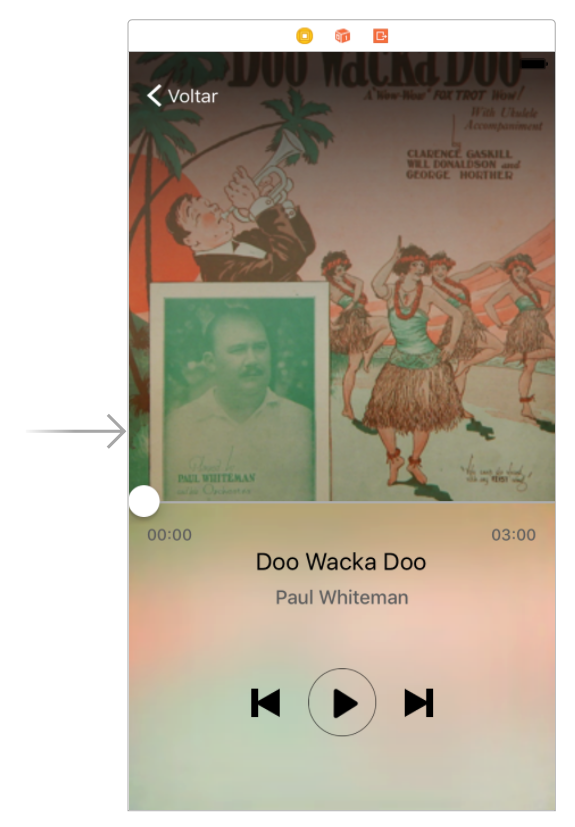


Figura 35 - Arquivos do projeto

Agora visite o arquivo **Main.storyboard.** Fizemos pequenas alterações na tela de música:



Apenas adicionamos novas imagens, um UISlider e duas labels de tempo. Uma mostrará o tempo corrido da música (total tocado) e a outra mostrará o restante a ser tocado.

Veja que temos um controle de música:



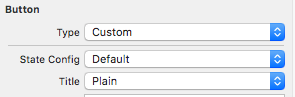
Nesta aula iremos utilizar apenas o botão Play.

Os componentes do tipo UIButton possui estados, e entre eles está o estado **isSelected**. Iremos utilizar este estado para mostrar a imagem pause:

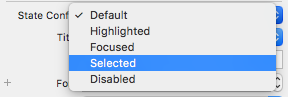


Então esta será nossa primeira atividade.

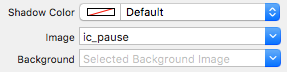
1. É possível definir imagens para os botões em seus diferentes estados. Para isto clique no botão play e no painel de propriedades localize a propriedade **State Config**:



1. Clique na caixa de combinação e selecione a opção **Selected.**



1. Agora localize a propriedade **Image.**
2. Nós já adicionamos a image para o botão pause para você então, no campo Image coloque o nome **ic\_pause.**



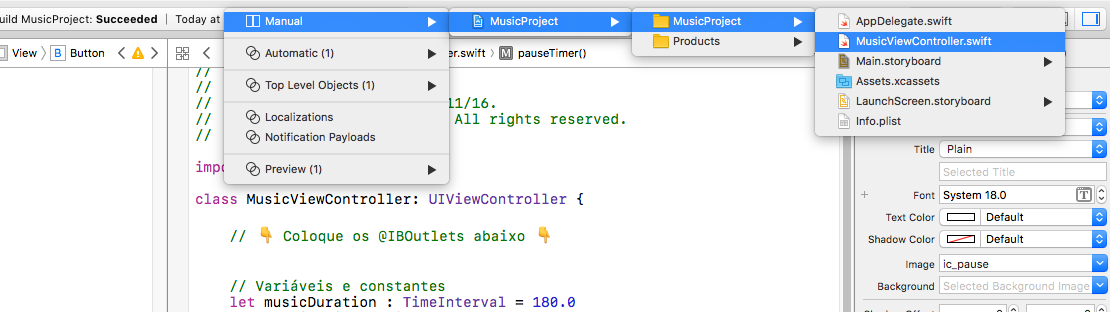
Agora quando dissermos que o botão está selecionado, a imagem será trocada automaticamente. Vale também para o inverso.

O próximo passo é referenciar os componentes no seu código. Para isto siga os seguintes passos:

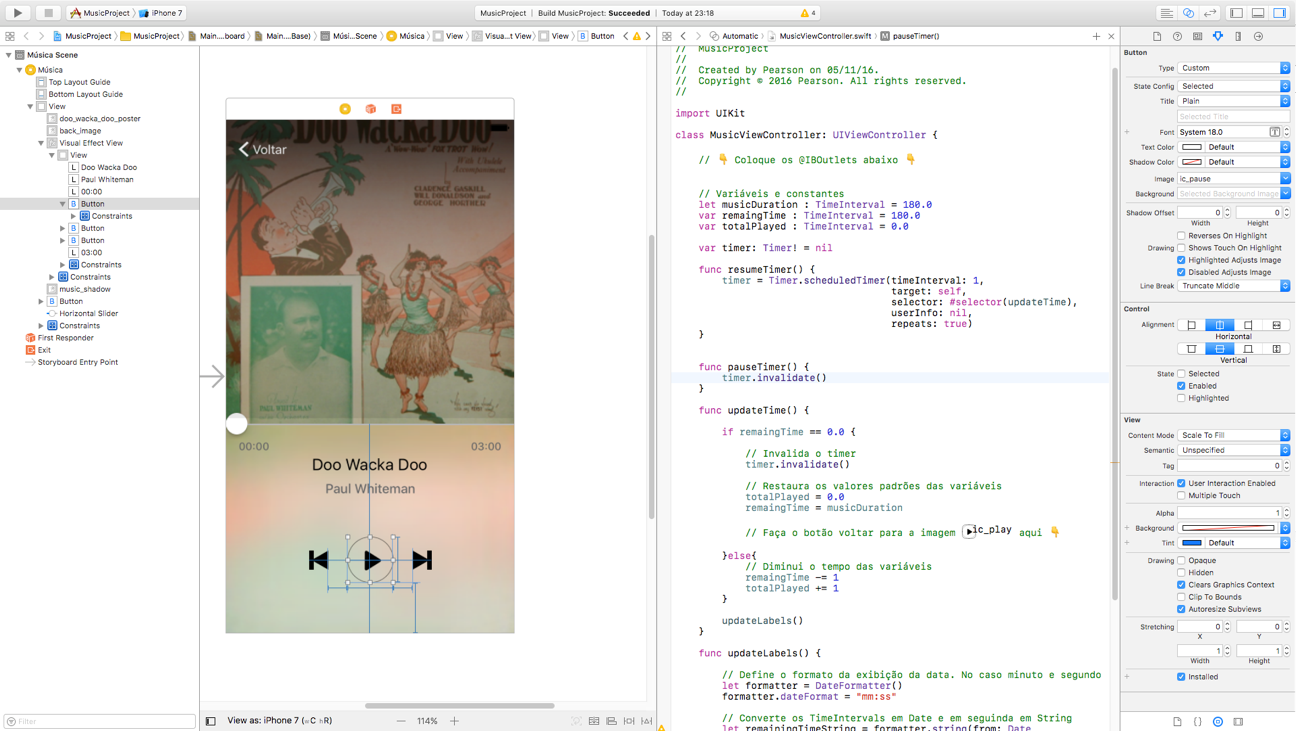
1. Abra o Assistant Editor de modo que do lado esquerdo seja exibido o storyboard e do lado direito seja exibido o arquivo MusicViewController.swift. Para isto clique no botão do meio nas abas, como na imagem abaixo:



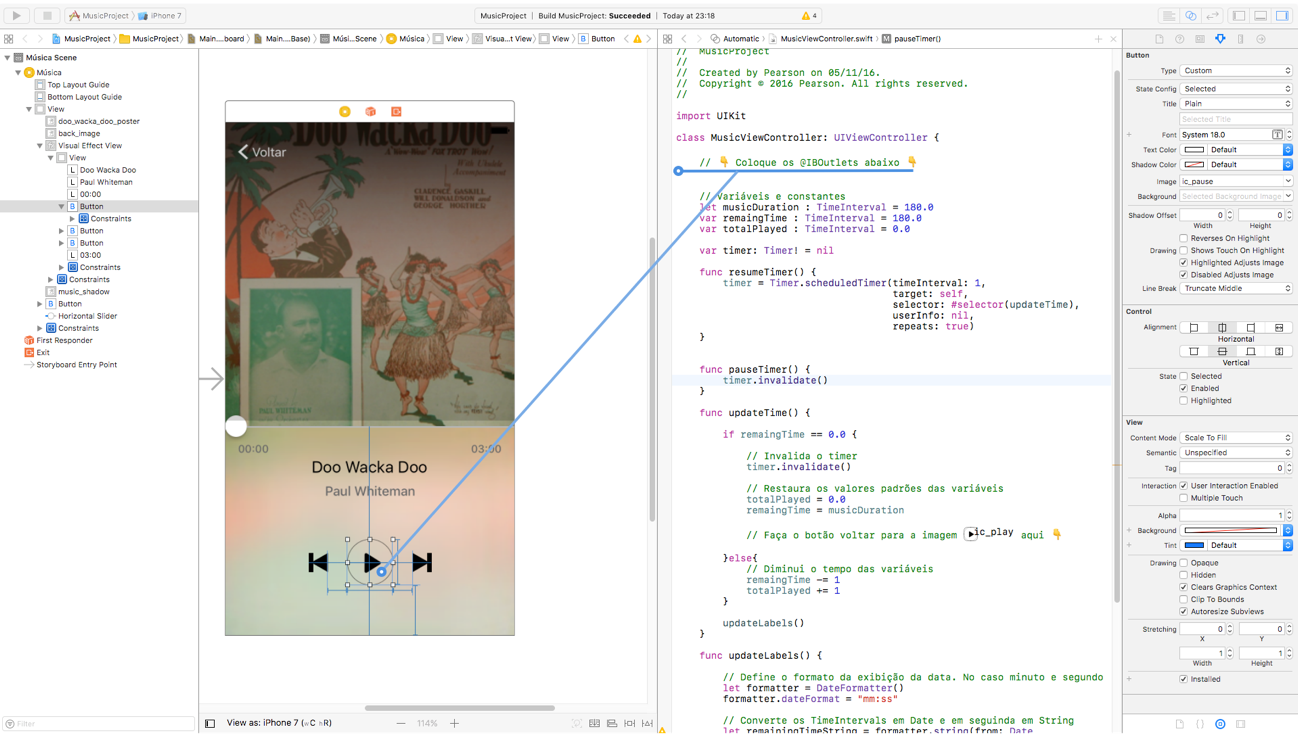
1. Após clicar neste botão, um segundo editor será aberto à direita. Por vezes ele não abre na ViewController correta, então selecione o arquivo manualmente como na seguinte imagem:



1. Você terá a seguinte tela:

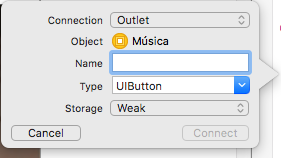


1. Para referenciar um componente do storyboard no arquivo Swift, basta clicar no componente com a tecla Control pressionada e arrastar a linha que se aparece para qualquer local dentro da classe (exeto dentro dos métodos), como na imagem abaixo:



DICA: No arquivo Swift tem uma marcação do local que você deverá soltar o elemento. Veja que temos emojis no meio do código. Esta é mais uma feature do Swift.

1. Quando você soltar aparecerá a seguinte caixa de diálogo:



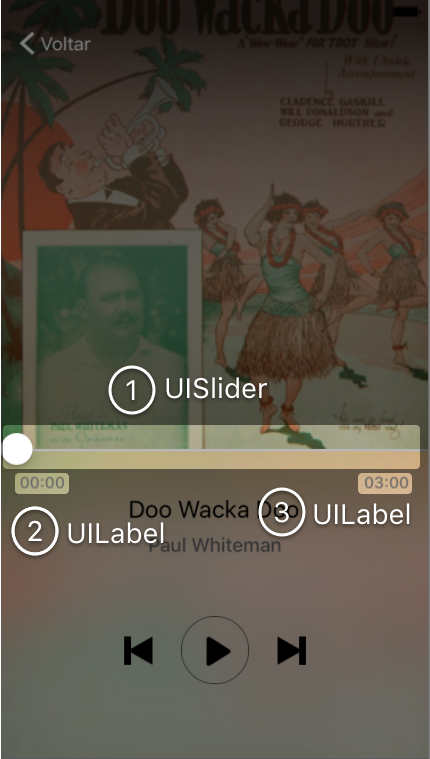
1. Nela é mostrada as seguinte opções:

* **Connection:** É o tipo de conexão que estamos tentando fazer. Podendo ser um Outlet, que é uma referência do componente no código ou uma Action, que atribui uma ação ao componente, sem necessariamente criar uma referência dele.
* **Object:** Indica a cena de origem do componente que está tentando ser conectado no código.
* **Name:** Nome do Outlet (referência ou variável) ou da Action (ação do componente).
* **Type:** Podemos indicar que tipo de componente estamos referenciando ou criando uma ação.
* **Storage:** Indica a maneira que a memória ocupada pelo componente será gerenciada. Com isto você não precisa se preocupar. Deixe sempre como weak.

1. Na opção **Name** insira o nome **playButton** e clique em **Connect.**
2. A seguinte linha de código aparecerá na sua classe:

@IBOutlet weak var playButton: UIButton!

1. O botão está conectado. Repita o processo de conexão para os seguintes elementos:



1. Ao final deste processo temos os seguintes IBOutlets:

// 👇 Coloque os @IBOutlets abaixo 👇

@IBOutlet weak var playButton: UIButton!

@IBOutlet weak var slider: UISlider!

@IBOutlet weak var totalPlayedLabel: UILabel!

@IBOutlet weak var remainingTimeLabel: UILabel!

Agora que temos nossos componentes referenciados, vamos entender um pouco o código que temos pronto.

O objetivo desta aula fazer com que a tela simule que uma música está sendo tocada. Primeiramente quando o botão play for tocado, sua imagem mudará para pause, em seguida ele disparará uma ação que inicirá um temporizador que atualizará as duas labels e a posição do slider como se uma musica estivesse sendo tocada. A label remainingTimeLabel mostrará o tempo restante para o término da música e a label totalPlayedLabel mostrará o tempo já tocado da música. Definimos aqui que esta musica terá por padrão 3 minutos (180 segundos) apenas para teste. Para isto definimos as seguintes variáveis:

let musicDuration : TimeInterval = 180.0

var remaingTime : TimeInterval = 180.0

var totalPlayed : TimeInterval = 0.0

Elas correspondem a duração da música, o tempo restante a ser tocado e o total tocado respectivamente.

Temos também os métodos para o temporizador:

func resumeTimer() {

timer = Timer.scheduledTimer(timeInterval: 1,

target: self,

selector: #selector(updateTime),

userInfo: nil,

repeats: true)

}

Dentro deste método um timer é inicializado e disparado. Ele chamará o método updateTime (que está sendo especificado no parâmetro **selector**) a cada 1 segundo e repetirá este processo até que o timer seja invalidado pelo seguinte método:

func pauseTimer() {

timer.invalidate()

}

Criado os métodos pause e resume, precisamos do método **updateTime()** informado no Timer:

func updateTime() {

if Int(remaingTime) == 0 {

// Invalida o timer

timer.invalidate()

// Restaura os valores padrões das variáveis

totalPlayed = 0.0

remaingTime = musicDuration

}else{

// Diminui o tempo das variáveis

remaingTime -= 1

totalPlayed += 1

}

updateLabels()

}

Ao final deste método vemos a chamada para o método **updateLabels()**:

func updateLabels() {

// Define o formato da exibição da data. No caso minuto e segundo

let formatter = DateFormatter()

formatter.dateFormat = "mm:ss"

// Converte os TimeIntervals em Date e em seguinda em String

let remainingTimeString = formatter.string(from: Date(timeIntervalSinceReferenceDate: remaingTime))

let totalPlayedString = formatter.string(from: Date(timeIntervalSinceReferenceDate: totalPlayed))

}

Este método pega os valores armazenados nas variáveis remainingTime e totalPlayed e converte-os para a exibição em minutos, pronto para serem utilizados nas labels.

Você pode consultar o uso de cada método e classe nova visto acima na documentação do Swift.

Agora vamos colocar o timer para funcionar.

Quando o método resumeTimer() for chamado, o método updateTime() será invocado a cada segundo, e por consequência o método updateLabels() também, então siga os seguintes passos:

1. No método updateTime() temos que fazer com que o botão play troque de ícone ao terminar a música (quando a remainingTime chegar a zero), para isto vá no local indicado e insira a seguinte instrução:

// Faça o botão voltar para a imagem ic\_play aqui 👇

playButton.isSelected = false

Esta instrução irá alterar o estado do botão para **não selecionado** e por causa disso a imagem será trocada automaticamente para a **ic\_play**. Logo criaremos o método para mudar a imagem do botão para **ic\_pause** (estado selecionado).

1. No método updateLabels() insira as seguinte linhas de código no local indicado:

// 👇 Atualize as labels aqui 👇

remainingTimeLabel.text = remainingTimeString

totalPlayedLabel.text = totalPlayedString

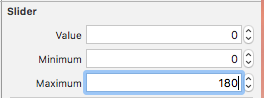
O atributo **text** da UILabel é do tipo String. O texto que for atribuído a ele será exibido na label.

1. Abaixo desta instrução iremos atualizar o Slider. Insira a seguinte instrução:

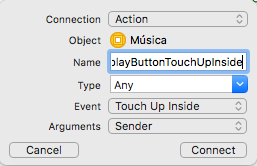
slider.value = Float(totalPlayed)

Este método irá atualizar o slider com o valor contido na variável totalPlayed. Por padrão os sliders suportam valores no intervalo entre 0 e 1, então teremos que mudar.

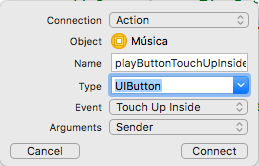
1. No storyboard, selecione o slider e localize a propriedade **Maximum** e atribua o valor 180 para este campo (este valor é o máximo que a variável totalPlayed poderá chegar).



1. Agora precisamos adicionar uma ação para o botão. Faça o mesmo processo indicado para referenciar o botão, mas desta vez solte-o no final da classe (local indicado) e no tipo de conexão (campo **Connection** da caixa de diálogo que se abre) escolha a opção Action.
2. Como nome escolhemos **playButtonTouchUpInside.**



1. No campo Type, escolha a opção **UIButton**:



1. Veja que apareceu um novo campo nesta caixa de diálogo. O campo **Event**. Deixe-o com o valor que está (TouchUpInside).
2. Clique em **Connect**
3. Você terá o seguinte método:

// 👇 Coloque os @IBActions abaixo 👇

@IBAction func playButtonTouchUpInside(\_ sender: UIButton) {

}

1. Agora insira o seguinte código dentro deste método:

if sender.isSelected {

pauseTimer()

}else{

resumeTimer()

}

sender.isSelected = !sender.isSelected

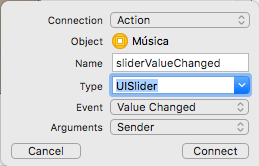
Vamos entender. Caso o estado do botão seja **selecionado,** então estamos “tocando” a música e neste caso ao tocar no botão devemos pausar a música. Caso contrário, iniciamos a música ou continuamos de onde ela parou. Estamos utilizando termos como “tocar música” mas entenda como “iniciar o temporizador”, pois ainda não temos músicas. Ao final deste método trocamos o estado do botão.

Execute e veja!

Toque no botão **play** e veja o tempo remanescente da música está diminuindo, o tempo corrido da música aumentando, o botão play virar pause e o slider progredir sozinho. Mas ainda falta alguma coisa. Mexa no slider.

Ele não irá te obedescer. Para isto devemos dar uma ação para ele. Siga os passos:

1. Como feito no botão, adicione uma ação para o slider.
2. Desta vez o evento deverá ser Value Changed (será o valor padrão):



1. No método criado insira as seguintes instruções:

remaingTime = TimeInterval(sender.value)

totalPlayed = musicDuration – remaingTime

updateLabels()

1. O novo método ficará da seguinte forma:

@IBAction func sliderValueChanged(\_ sender: UISlider) {

totalPlayed = TimeInterval(sender.value)

remaingTime = musicDuration - totalPlayed

updateLabels()

}

Execute o código e teste novamente. O Slider irá te obedecer, pois quando você muda seu valor, ele muda o valor das variáveis de controle de tempo.

## Resumo

Nesta aula você aprendeu como funcionam os controles de entrada e seus eventos. Vimos também como é fácil referenciar um componente do interface builder no código, se comparado com o Android. O uso dos modificadores @IBOutlet e @IBAction nos lembra o uso da biblioteca ButterKnife do Android, que é utilizada justamente para facilitar esta tarefa. Você aprendeu a manipular eventos, estados do botão e utilizar o slider como controlador.