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 - Tela inicial do Xcode

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



Figura - 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.**

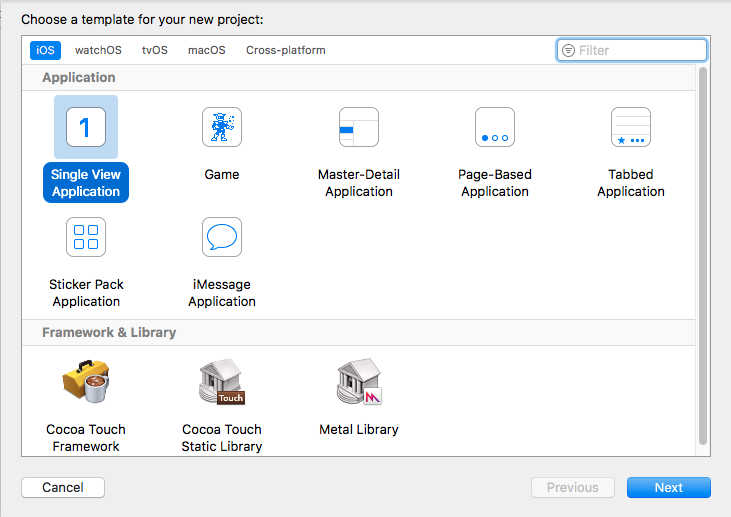


Figura - Escolha **Single View Application**

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

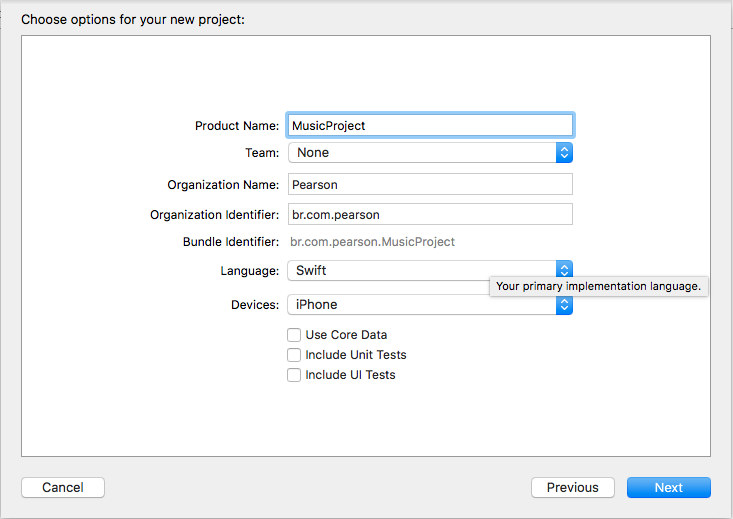


Figura - Configurações iniciais do projeto

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:

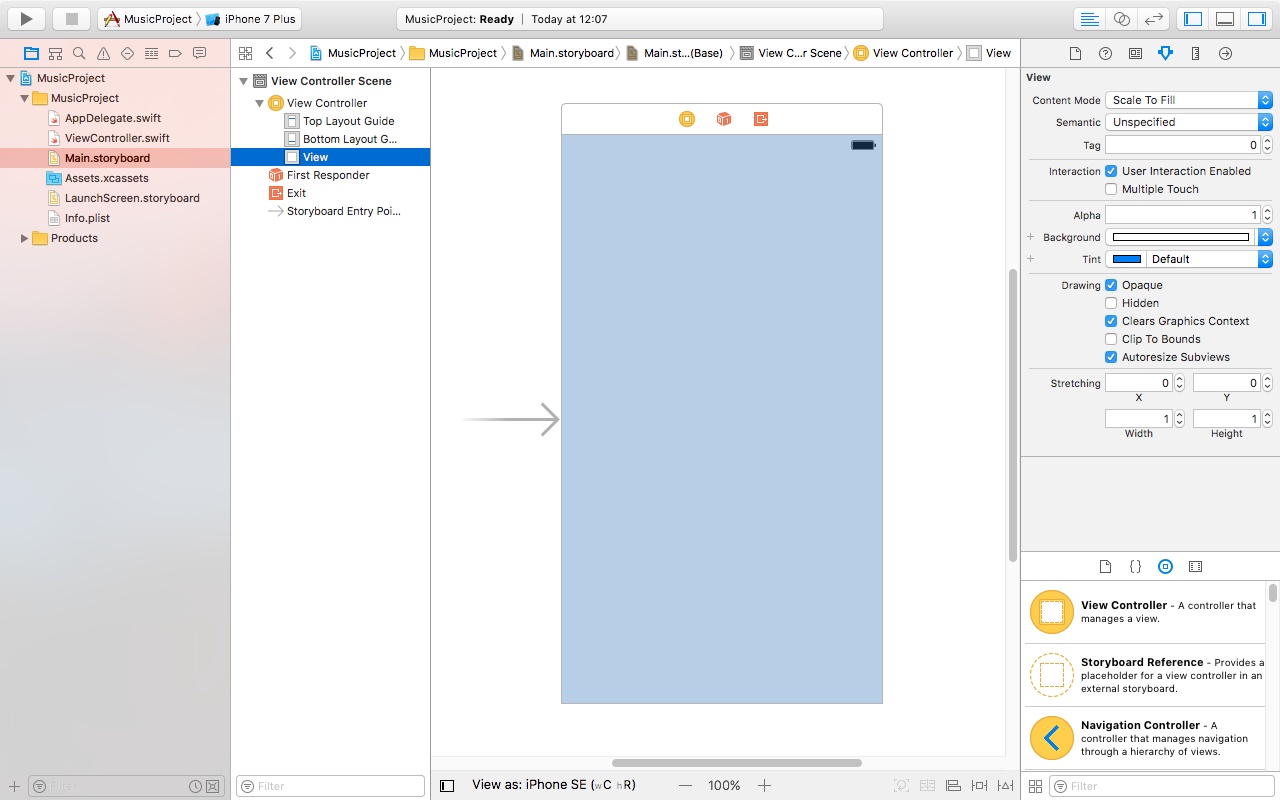


Figura - Main.storyboard

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.

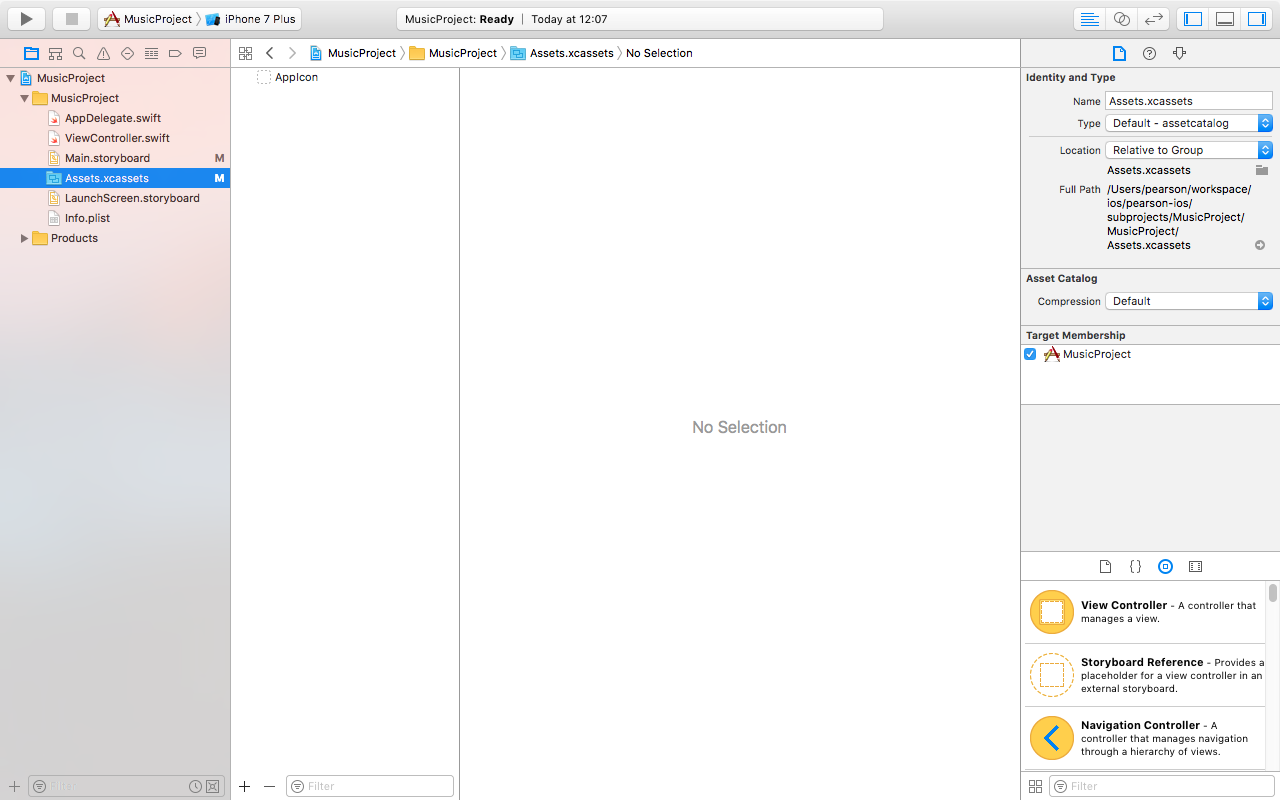


Figura - Visualização da pasta Assets

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.

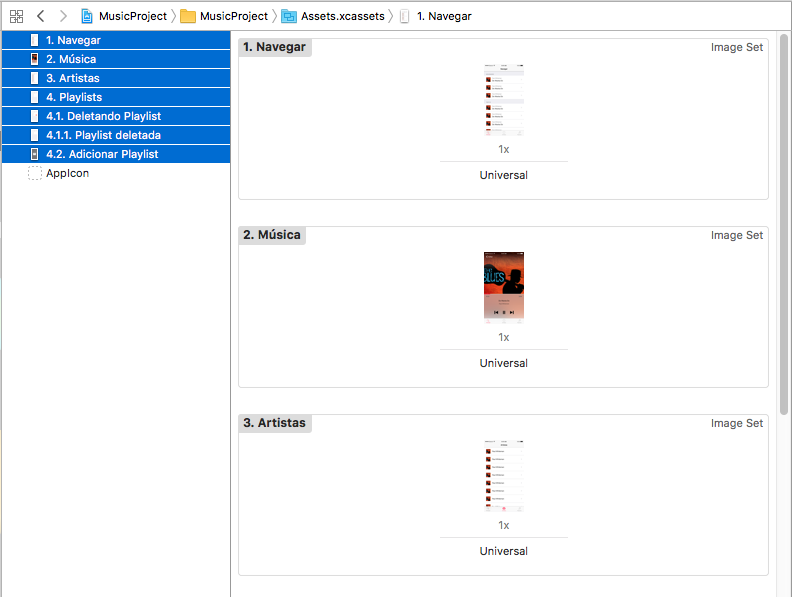


Figura - Imagens adicionadas na pasta Assets

### 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.

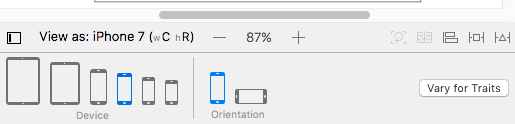


Figura - Painel de Modos de Visualização do Xcode

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

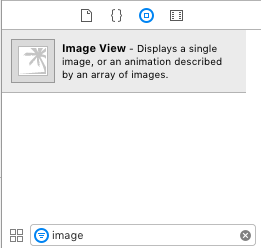


Figura - Procurando Image View na paleta de componentes

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

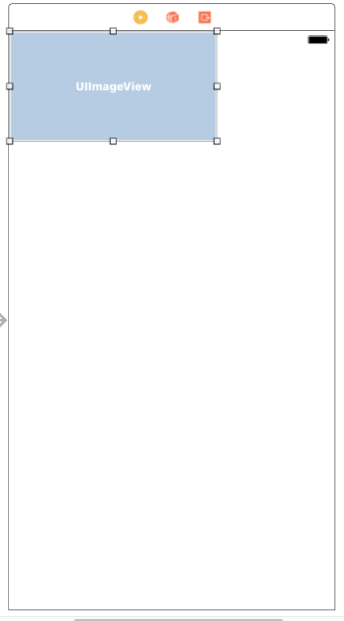


Figura - Posicionando a Image View na cena

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

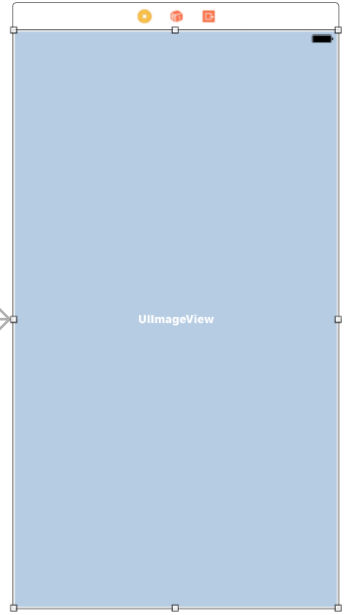


Figura - Redimensionando a Image View na cena

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.

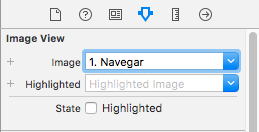


Figura - Selecionando a imagem que aparecerá na Image View

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 - Localizando o Simulador configurado

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



Figura - 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 - Botão Run (executar)

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

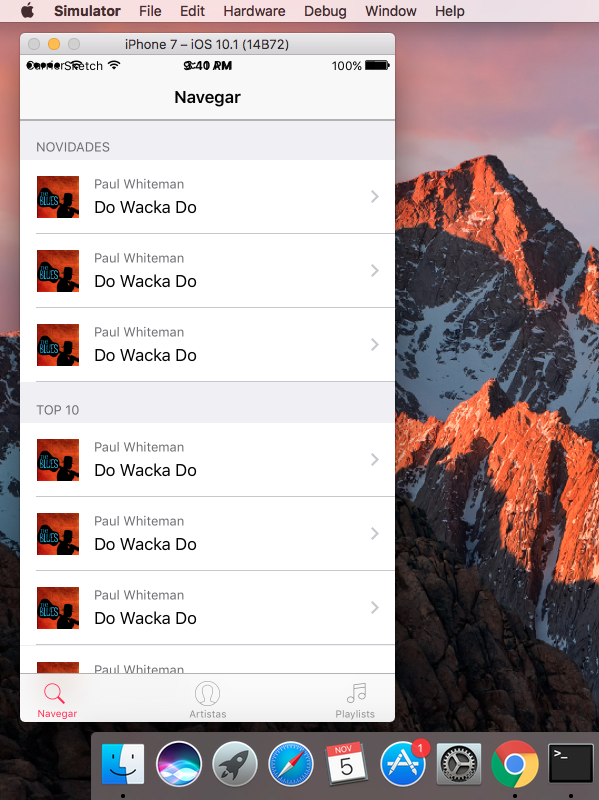


Figura - Aplicativo sendo executado no simulador do iPhone 7

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

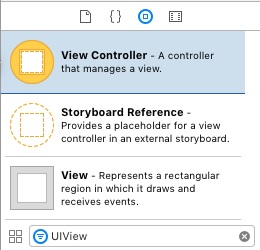


Figura - Localizando a View Controller na paleta de componentes

1. Insira a View Controller no Storyboard.

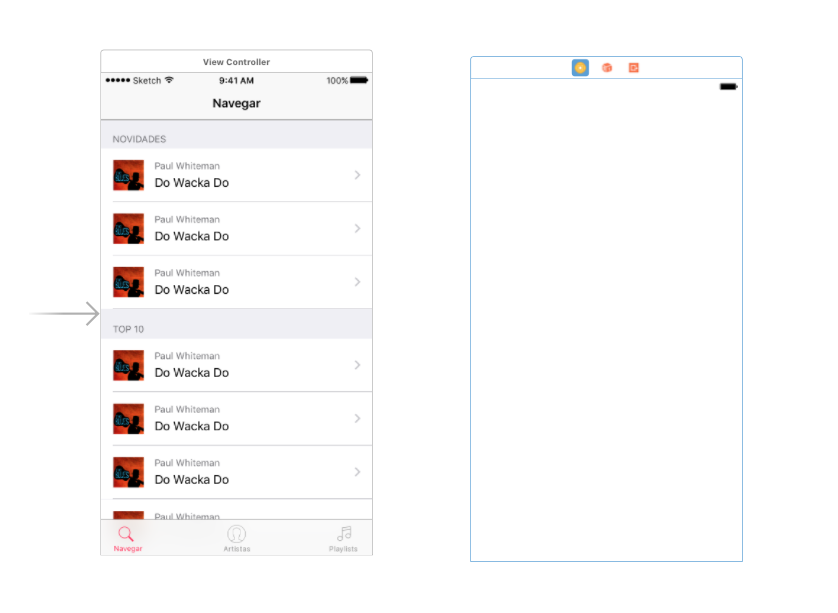


Figura - Posicionando a View Controller no Sotryboard

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:

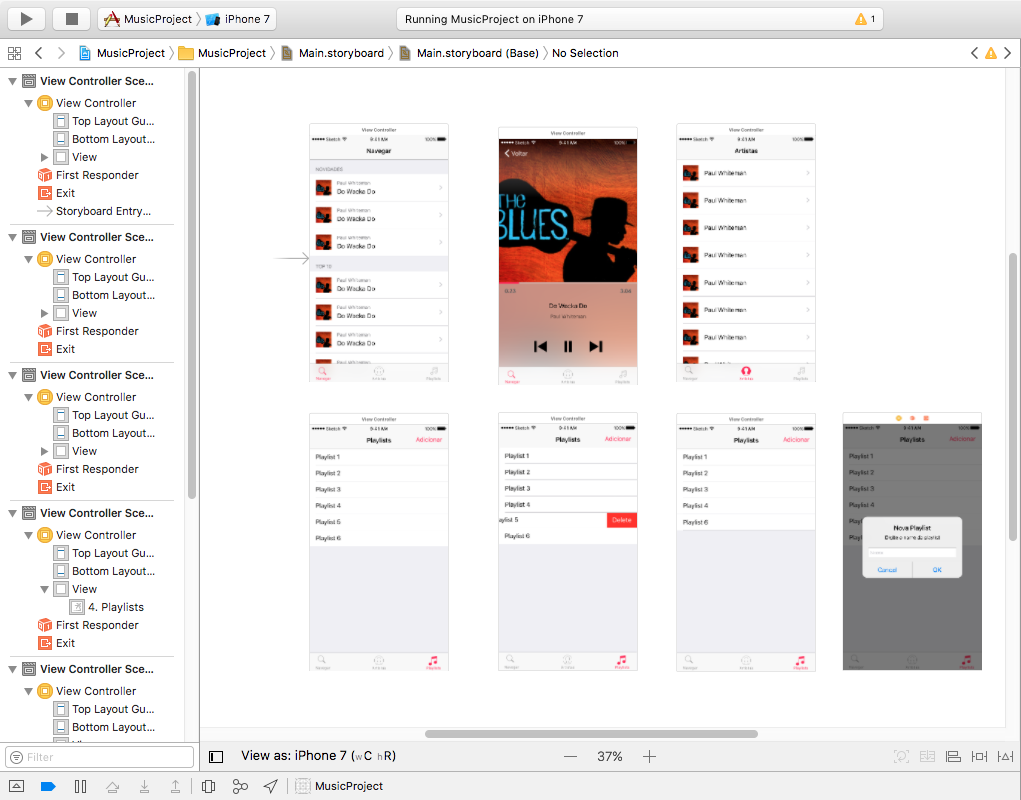


Figura - Todas as View Controllers adicionadas com suas respectivas imagens

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:

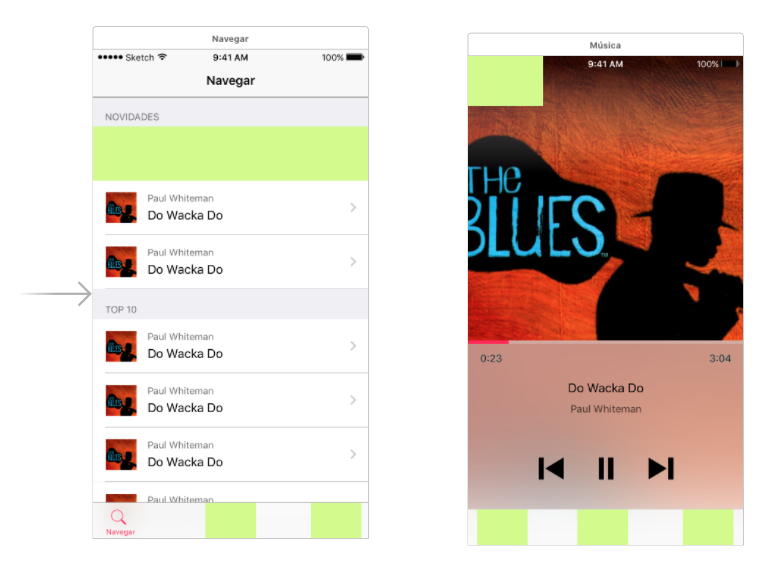


Figura - Cena Navegar (esquerda) e Música (direita)

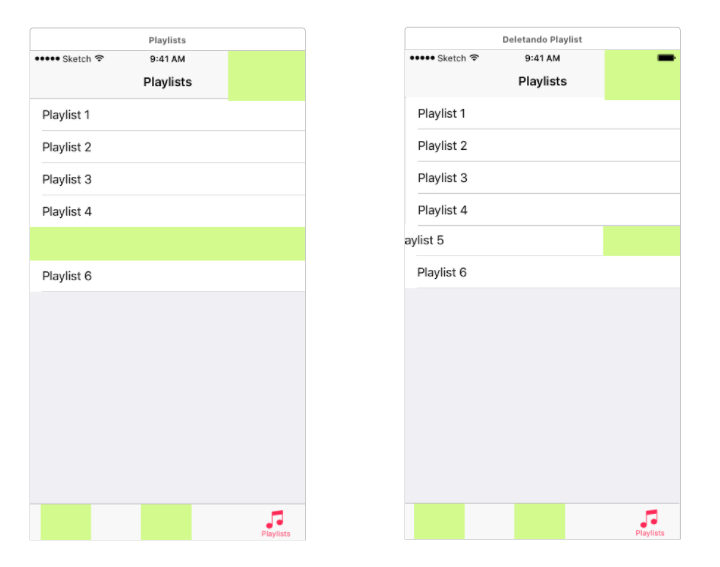


Figura - Cena Playlists (esquerda) e Deletando Playlist (direita)

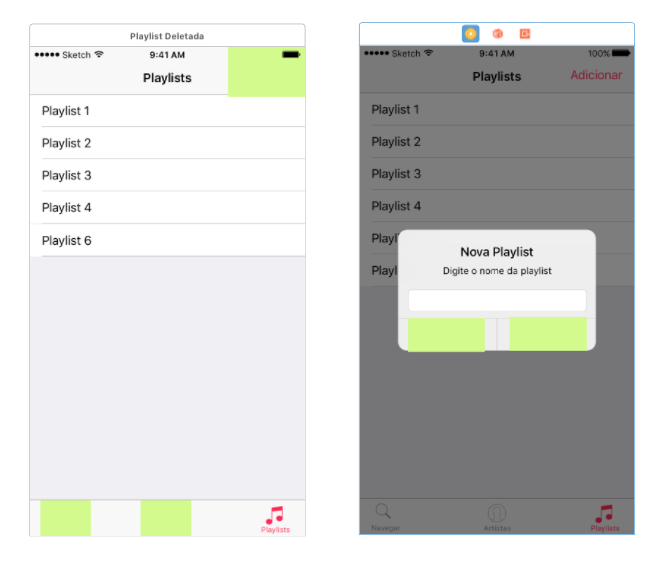


Figura - Cena Playlist Deletada (esquerda) e Adicionar Playlist (direita)

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

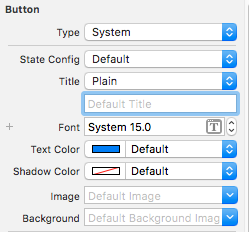


Figura - Removendo o texto do botão

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.**

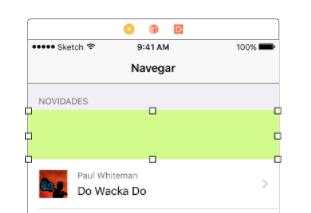


Figura - Primeira célula da cena Navegar

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

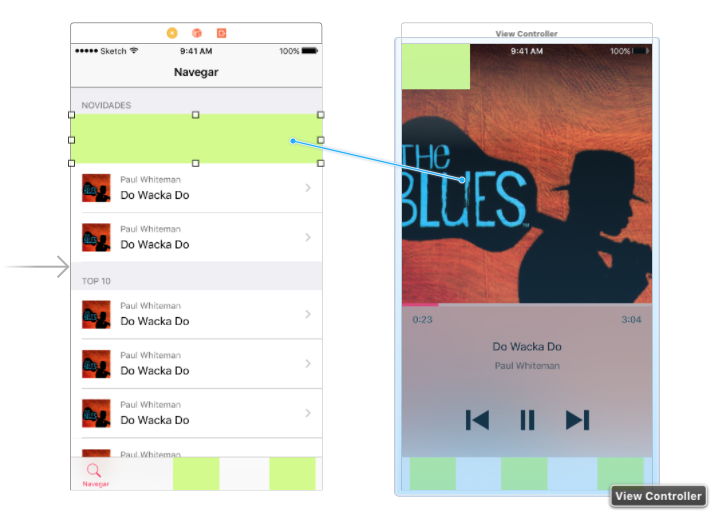


Figura - Efeito ao clicar e arrastar um componente quando a tecla Control está pressionada

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

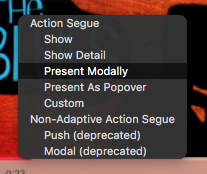


Figura - Ações para a Segue

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.

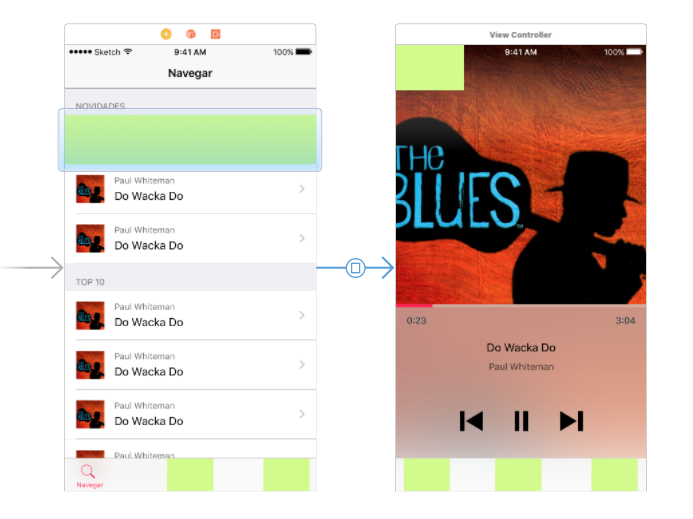


Figura - Cenas vinculadas por uma Segue

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.

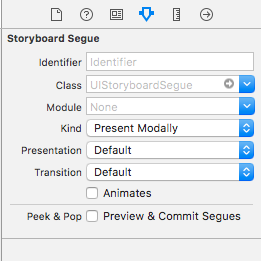
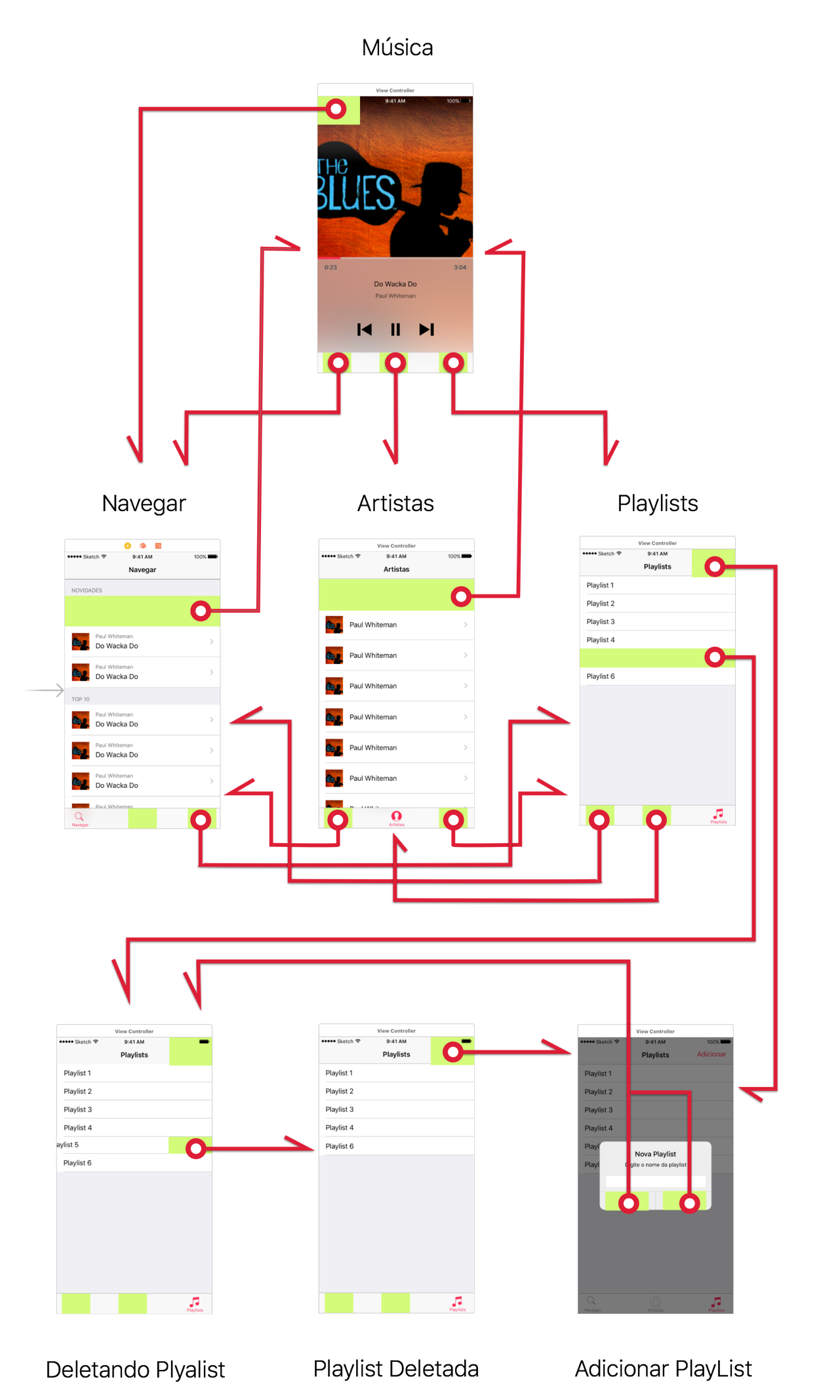


Figura - Desabilitando animações

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.

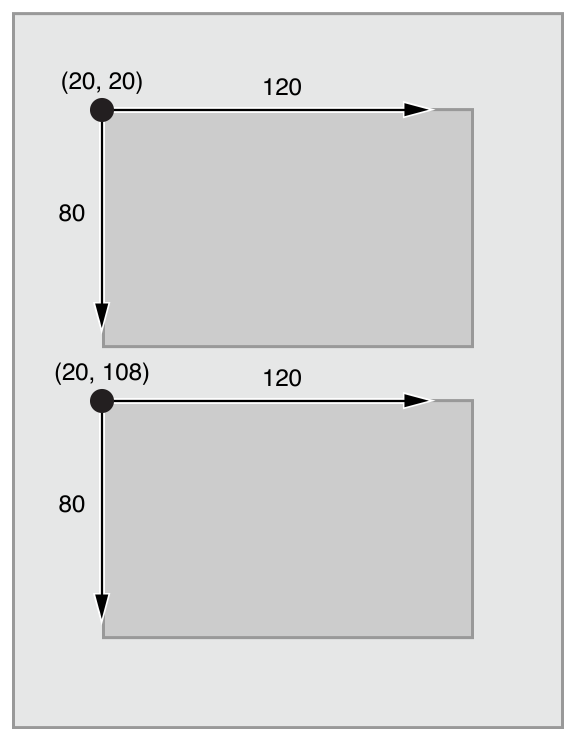
A internacionalização é o processo de fazer seu aplicativo se adaptar a diferentes linguagens, regiões e alfabetos. O layout de um aplicativo internacionalizado de levar estas diferenças em conta para que seja exibido corretamente para todas as línguas e regiões que o aplicativo suporta.

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.

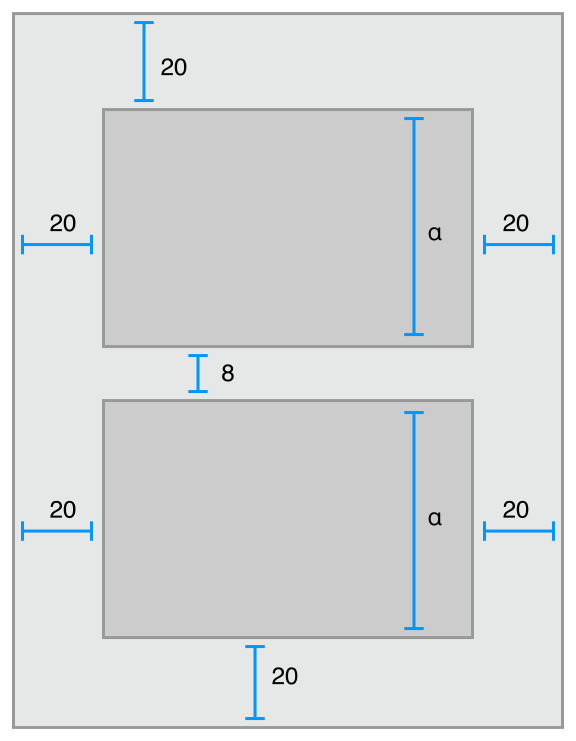
DICA: Você pode encontrar mais detalhes sobre autoresizing mask em: <https://developer.apple.com/reference/uikit/uiview/1622559-autoresizingmask>

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.

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.

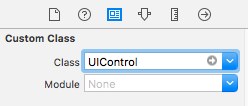


Figura - Método para definir um UIControl através do Interface Builder

### 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 - 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 - 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 - 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

### Reconhecimento de moviemntos