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.**

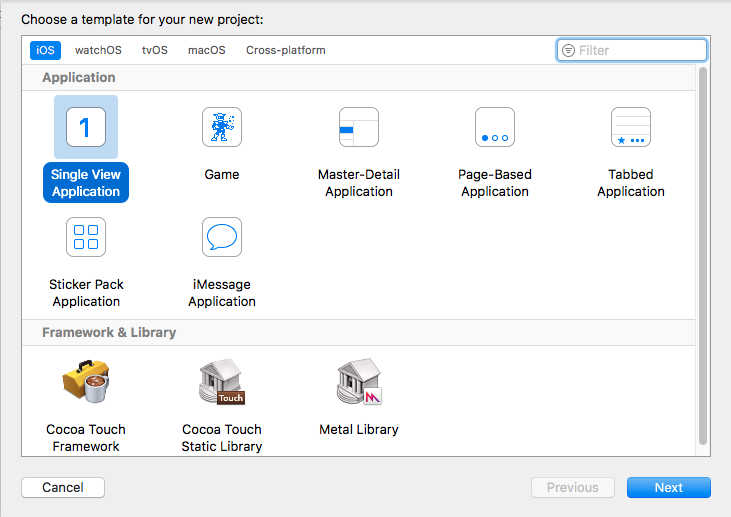


Figura 3 - Escolha **Single View Application**

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

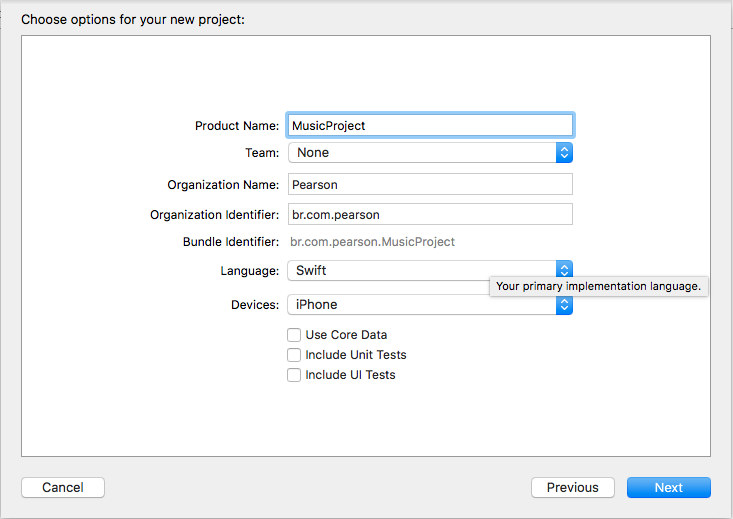


Figura 4 - Configurações iniciais do projeto

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:

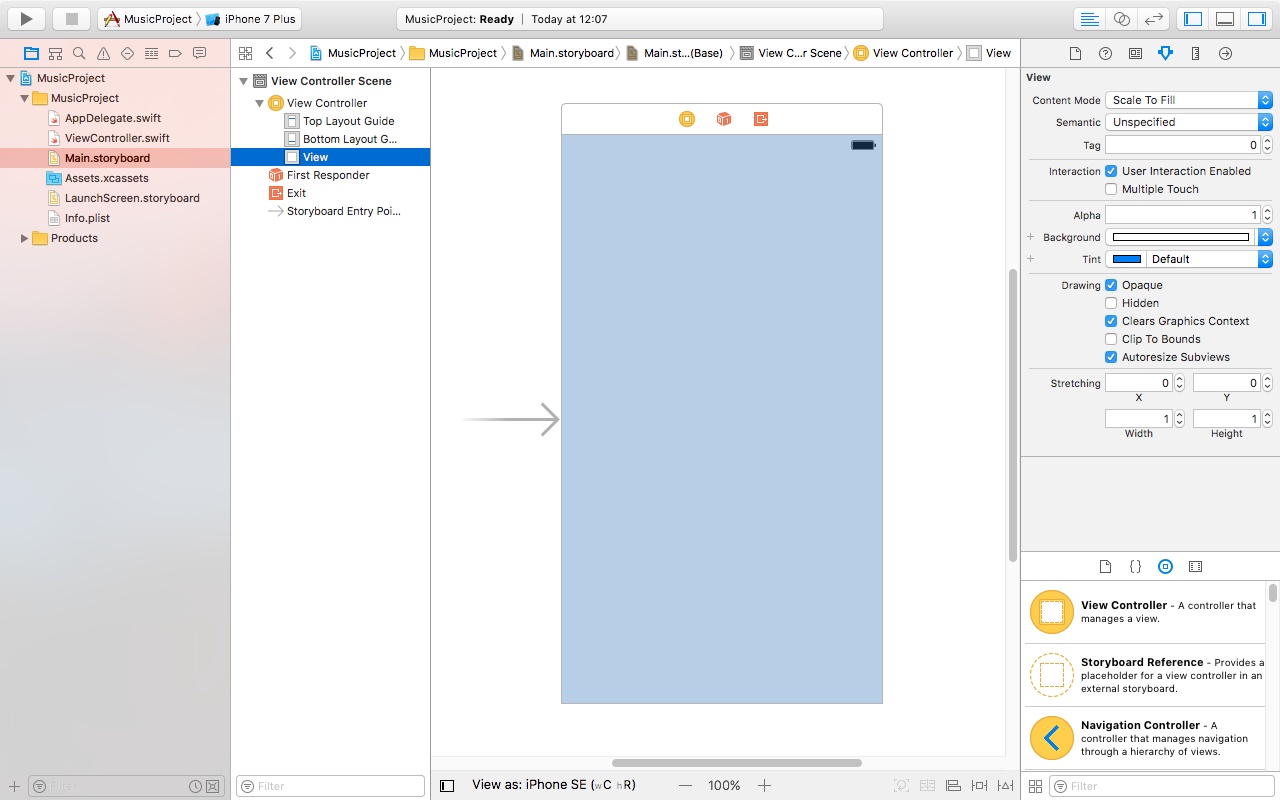


Figura 5 - Main.storyboard

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.

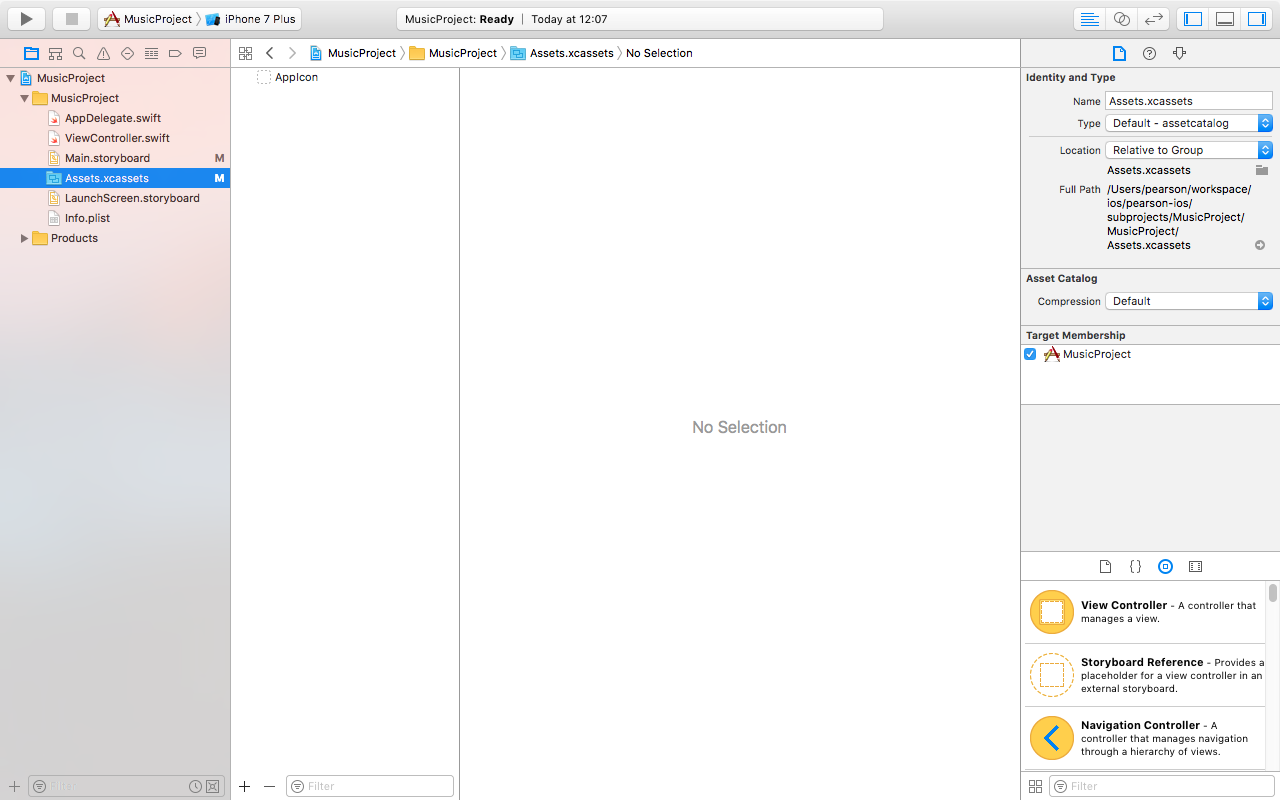


Figura 6 - Visualização da pasta Assets

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.

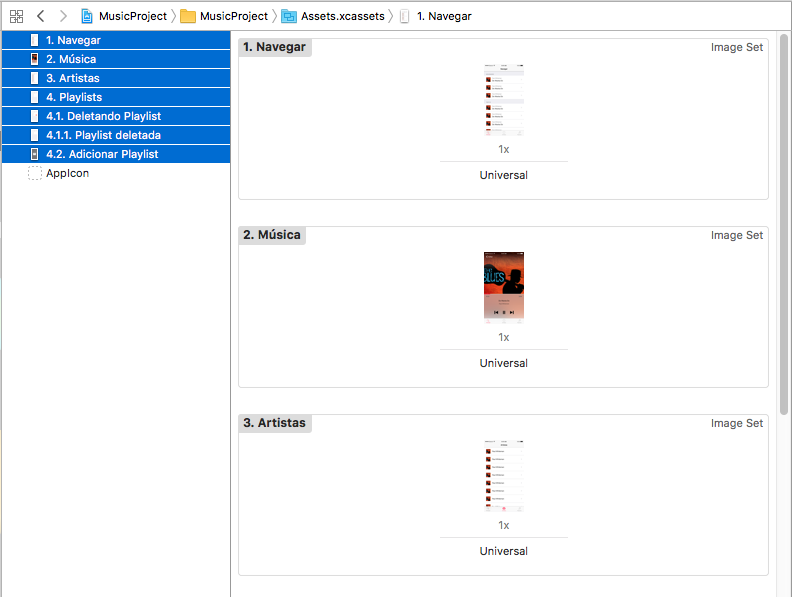


Figura 7 - Imagens adicionadas na pasta Assets

### 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.

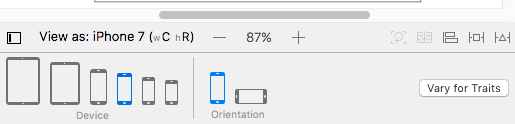


Figura 8 - Painel de Modos de Visualização do Xcode

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

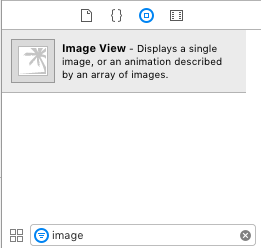


Figura 9 - Procurando Image View na paleta de componentes

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

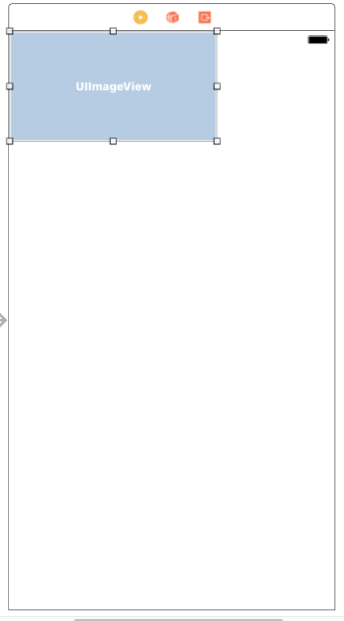


Figura 10 - Posicionando a Image View na cena

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

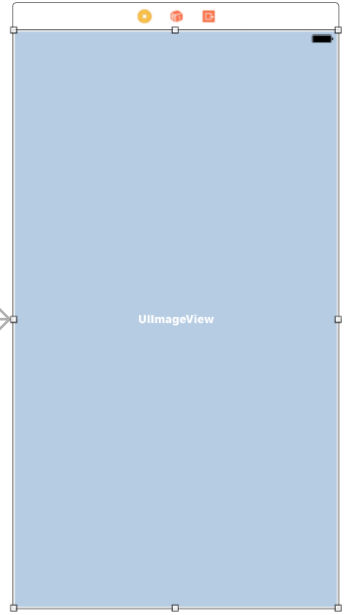


Figura 11 - Redimensionando a Image View na cena

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.

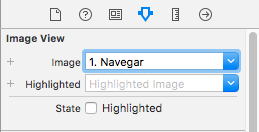


Figura 12 - Selecionando a imagem que aparecerá na Image View

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.

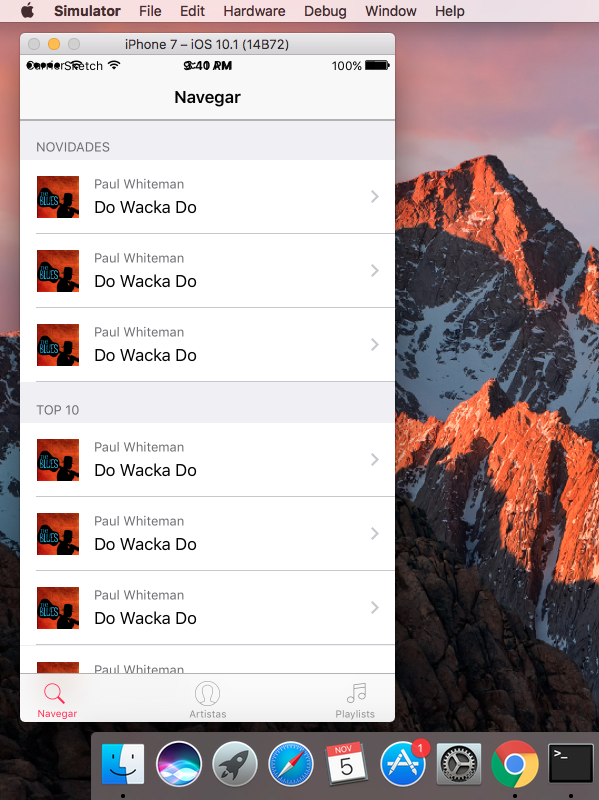


Figura 16 - Aplicativo sendo executado no simulador do iPhone 7

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

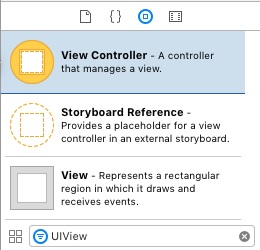


Figura 17 - Localizando a View Controller na paleta de componentes

1. Insira a View Controller no Storyboard.

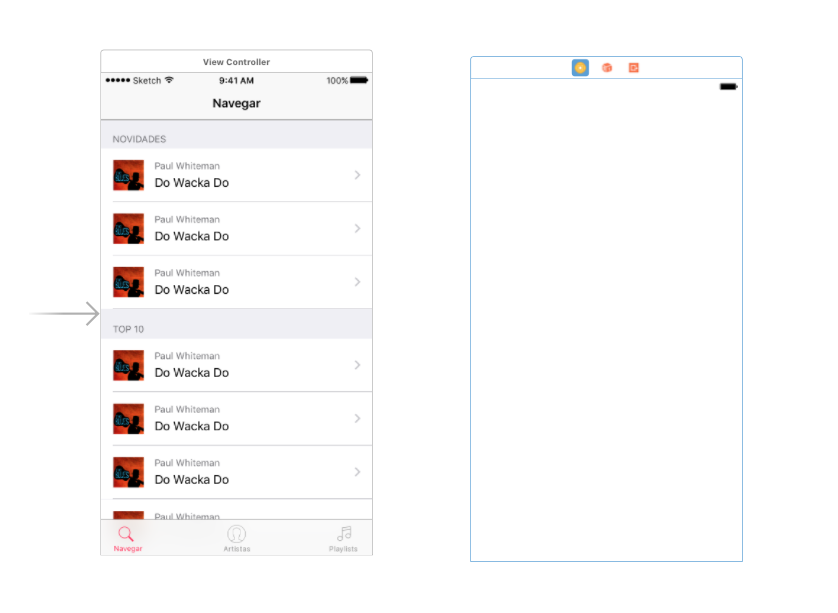


Figura 18 - Posicionando a View Controller no Sotryboard

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:

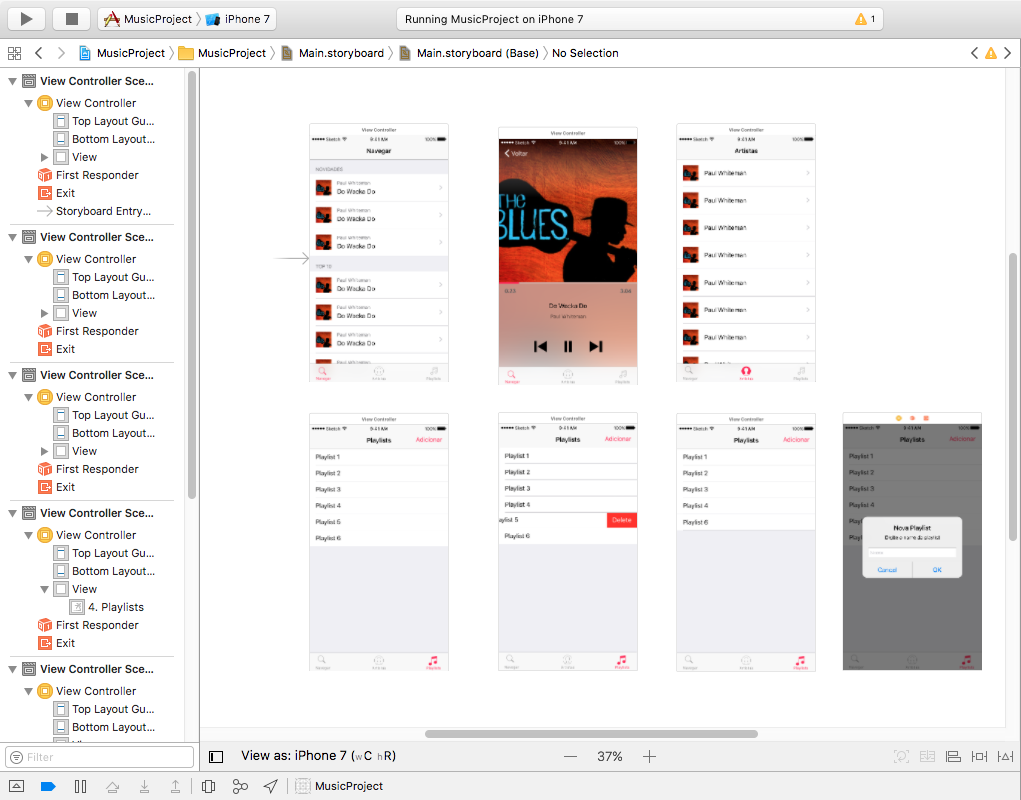


Figura 19 - Todas as View Controllers adicionadas com suas respectivas imagens

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:

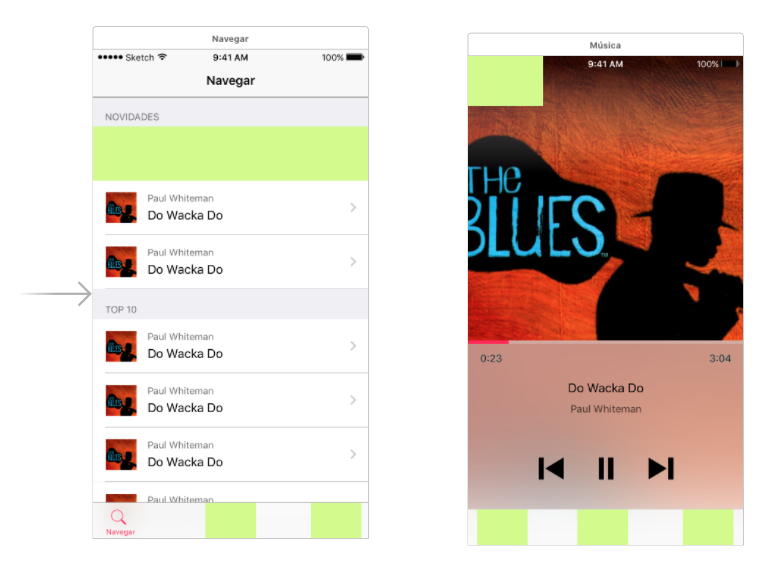


Figura 20 - Cena Navegar (esquerda) e Música (direita)

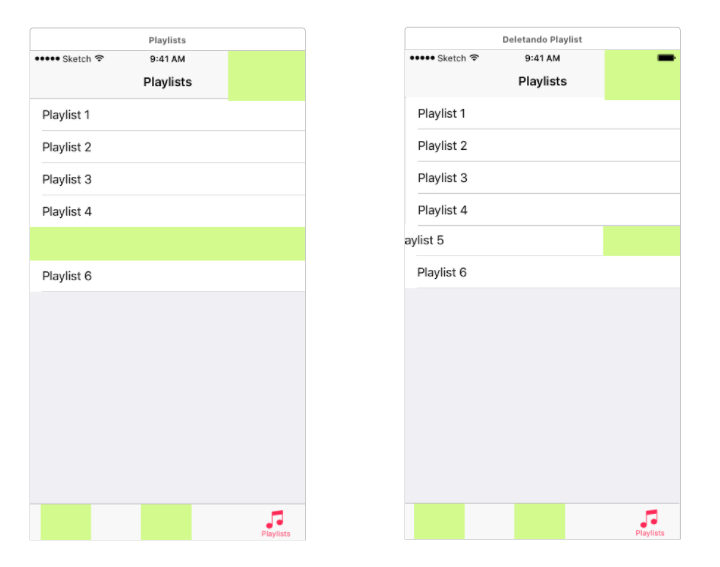


Figura 21 - Cena Playlists (esquerda) e Deletando Playlist (direita)

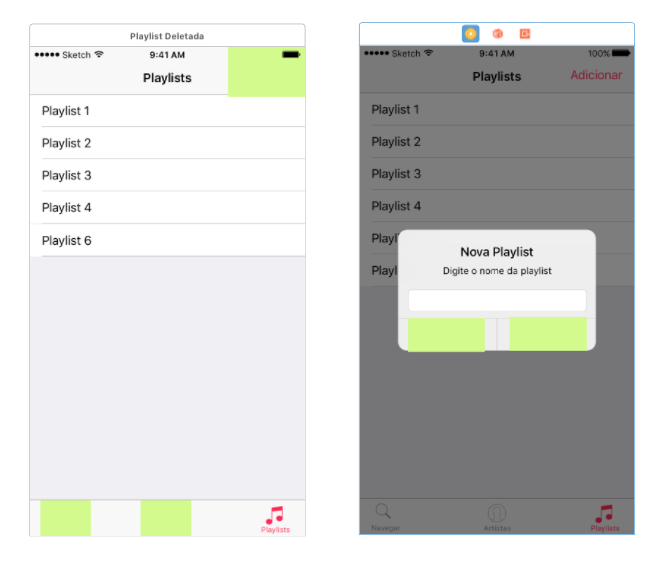


Figura 22 - Cena Playlist Deletada (esquerda) e Adicionar Playlist (direita)

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

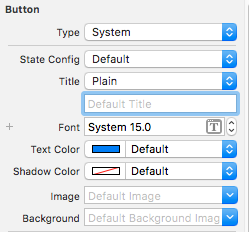


Figura 23 - Removendo o texto do botão

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.**

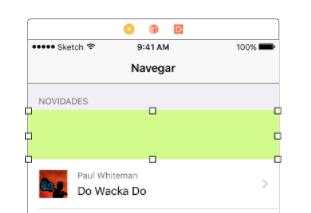


Figura 24 - Primeira célula da cena Navegar

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

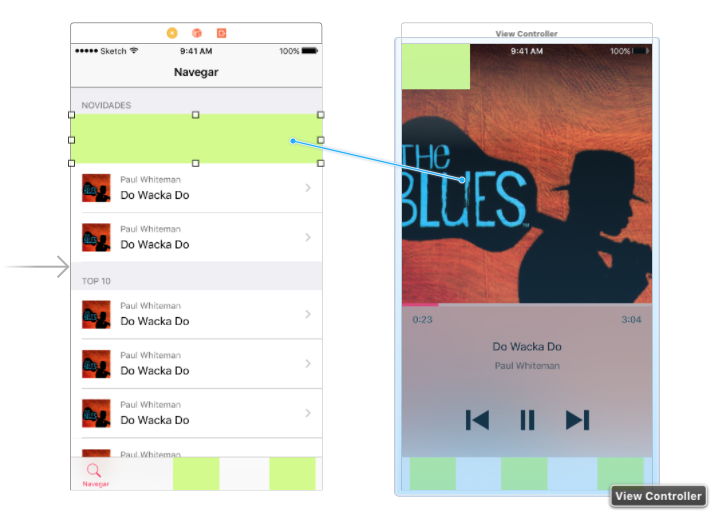


Figura 25 - Efeito ao clicar e arrastar um componente quando a tecla Control está pressionada

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

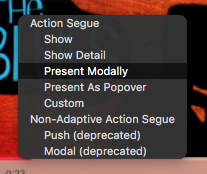


Figura 26 - Ações para a Segue

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.

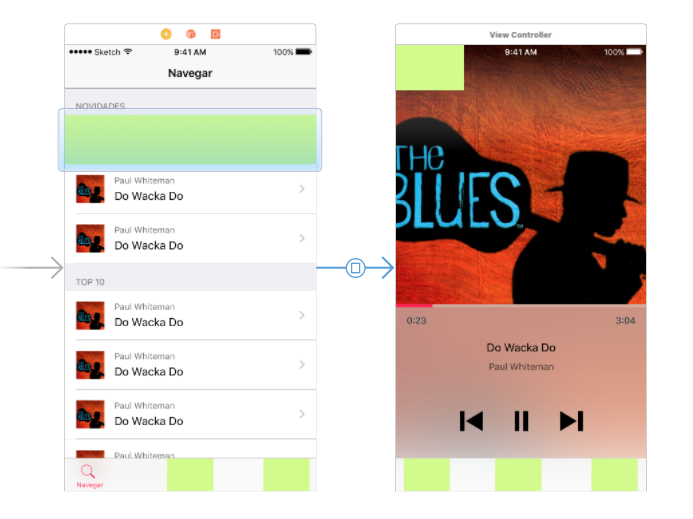


Figura 27 - Cenas vinculadas por uma Segue

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.

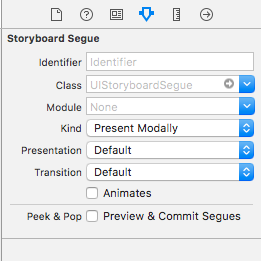
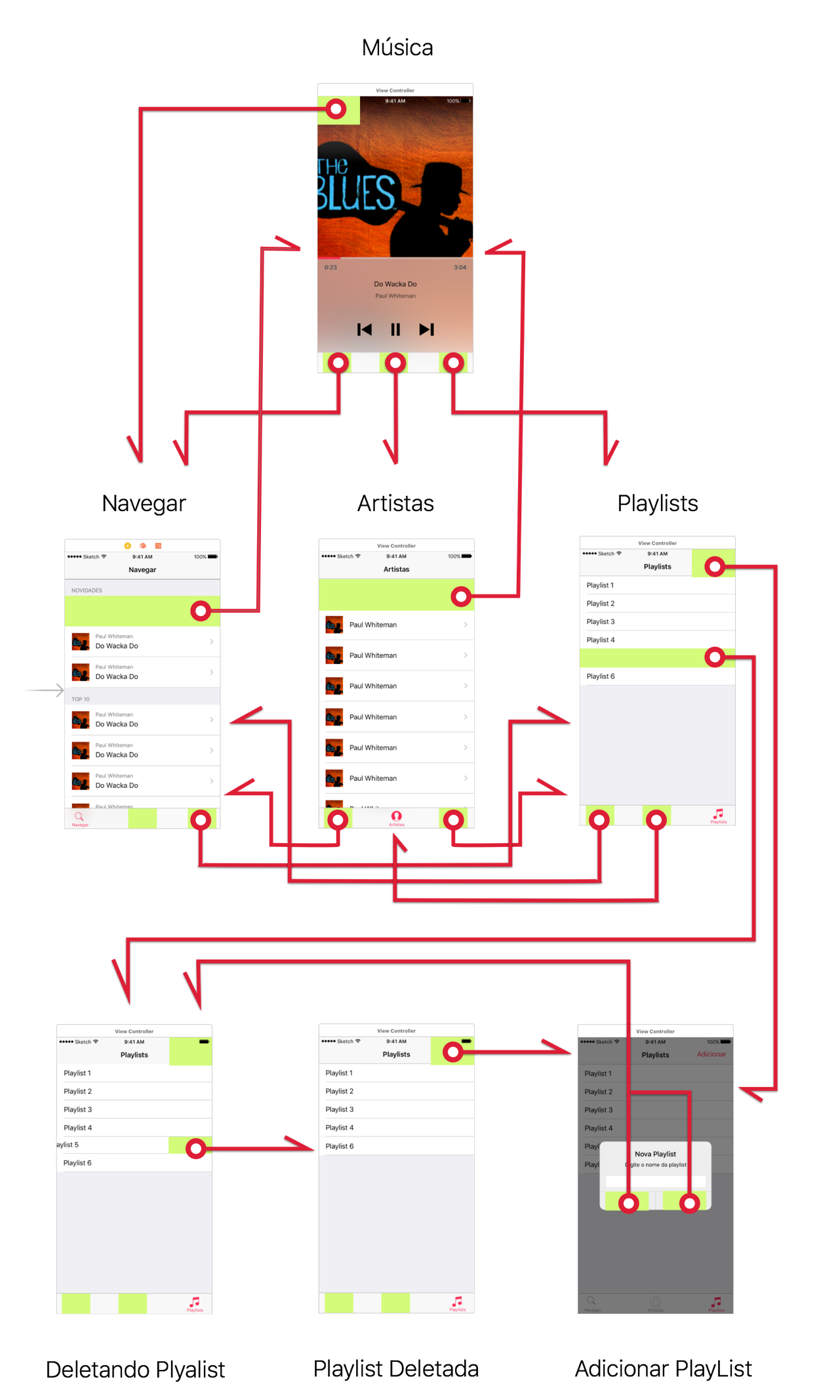


Figura 28 - Desabilitando animações

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.

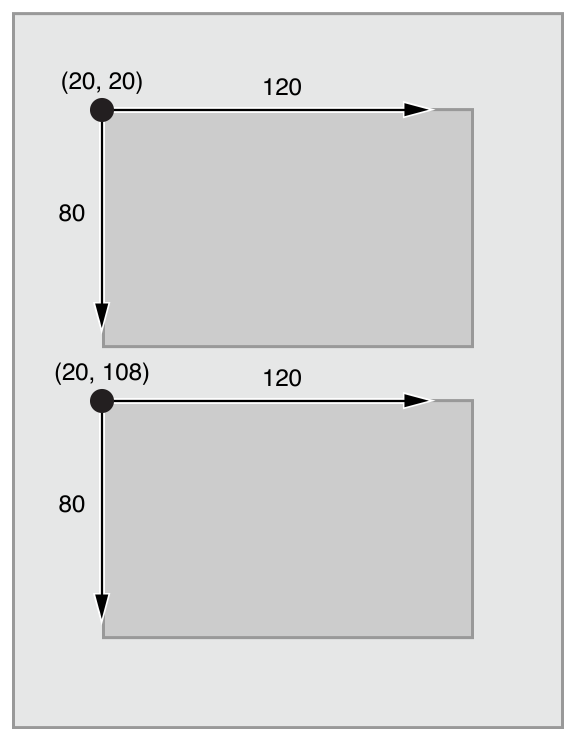
A internacionalização é o processo de fazer seu aplicativo se adaptar a diferentes linguagens, regiões e alfabetos. O layout de um aplicativo internacionalizado de levar estas diferenças em conta para que seja exibido corretamente para todas as línguas e regiões que o aplicativo suporta.

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.

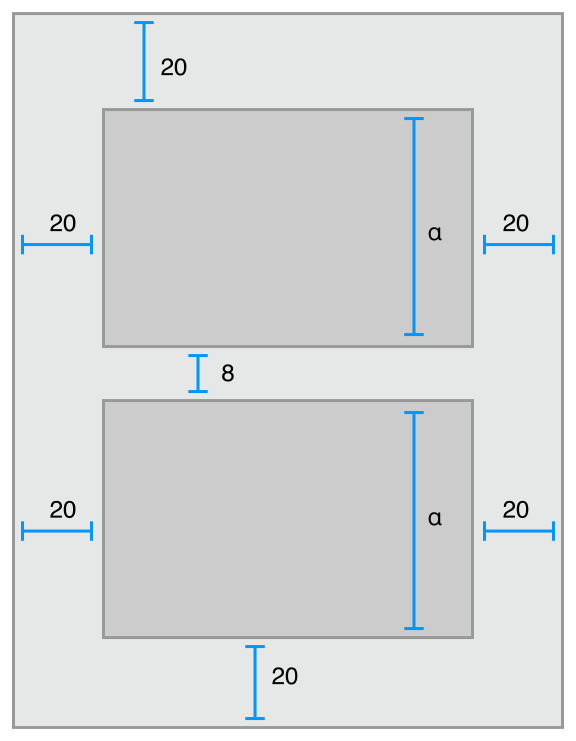
DICA: Você pode encontrar mais detalhes sobre autoresizing mask em: <https://developer.apple.com/reference/uikit/uiview/1622559-autoresizingmask>

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.

Aula 3

# 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.

## Entendendo a hierarquia das Views

Objetivamente a View é representada pela classe UIView e define umas 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 (é uma variedade enorme de subclasses da UIView).

### Visão Geral

Em tempo de execução, um objeto de View lida com a prestação 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 desenha uma imagem.

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

* **Desenho e animação**
  + Visualizações 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 View define o seu próprio comportamento de redimensionamento padrão em relação à sua superview.
  + Uma View pode definir o tamanho e posição de suas subviews conforme necessário.
* **Manipulação de eventos**
  + A View é 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 gesto para lidar com gestos comuns no smartphone.

Com as Views é possível incorporar outras Views, criando hierarquias visuais sofisticadas. Isto cria uma relação mãe-filha entre uma View que está sendo incorporada (conhecida como subview) e a View pai, que é a incorporadora (conhecida como superview). Normalmente, a área visível de uma subview não é podada 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 aView Controller, vamos compreender seu funcionamento a seguir.

## Compreendendo a View Controller

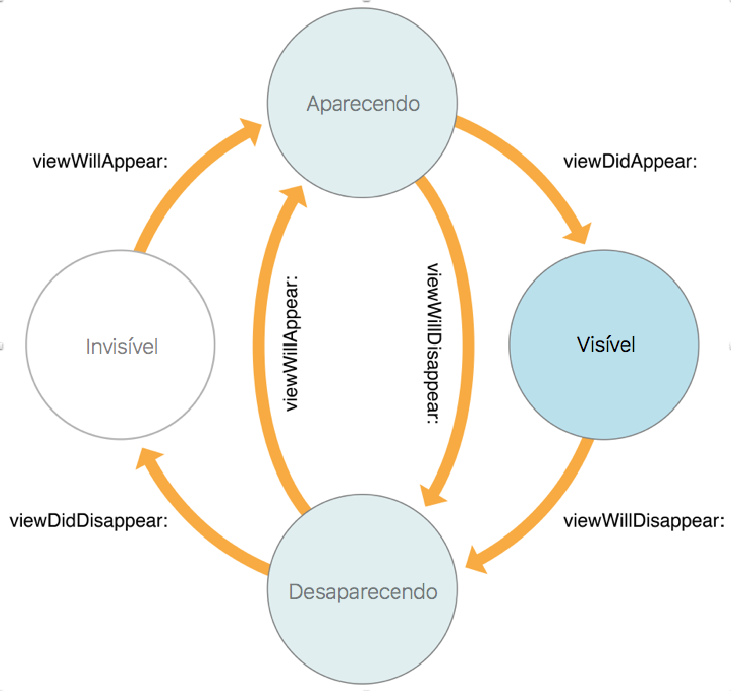
A classe UIViewController fornece a infra-estrutura apropriada para a gestão views dos seus aplicativos iOS. Um View Controller gere um conjunto de views que compõem uma parte da interface do usuário do seu aplicativo. É responsável por carregar e eliminar de tais views, por gerenciar interações entre estas views, e por mediar a interação entre quaisquer modelos de dados e as views. Resumidamente a View Controller controla e gerencia as views delegadas a ela. Comparando com o Android, podemos fazer um paralelo com a Activity.

### O ciclo de vida de uma View Controller

Um objeto de UIViewController (e suas subclasses) vêm com um conjunto de métodos que gerem a sua hierarquia de views. O iOS automaticamente chama 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 adequado no método correto, ou seja, existe um método da View Controller apropriado para cada configuração de views a ser feita. Abaixo 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.



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 a depender 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 que deseja-se que sejam realizadas quando a view principal se torna 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 irá tornar-se 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, e sempre após o método **viewWillAppear()**.

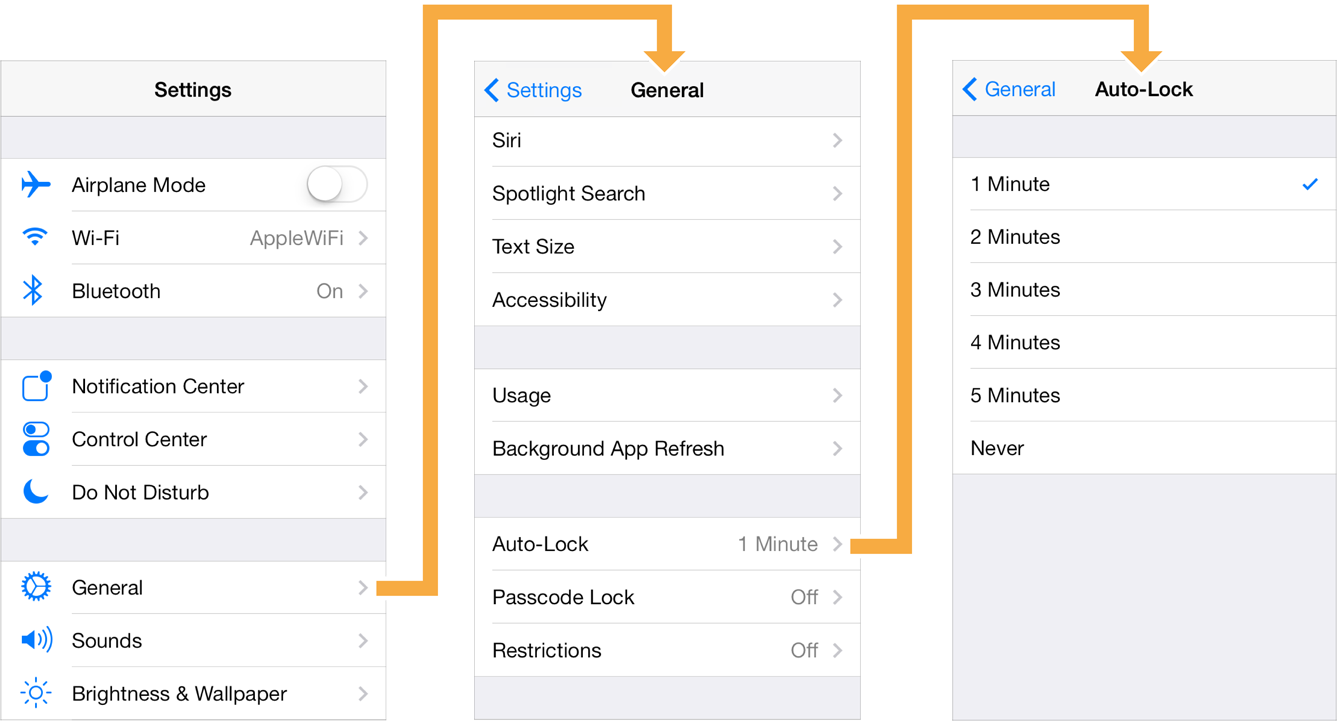
Estes são os métodos principais e que você provavelmente fará uso massivo dele, para os métodos **viewWillDesapear()** e **viewDidDisapear(),** mostrados no diagrama, vale o análogo dos métodos **viewWillAppear()** e **viewDidAppear()** respectivamente, ou seja, quando os dois últimos valem para quando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

## UINavigationController

A classe UINavigationController implementa um View Controller que gere 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 acima, 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 abaixo mostra um exemplo da interface de navegação  do aplicativo Settings do simulador do iOS. A primeira tela apresenta ao usuário uma lista de preferências de aplicativos. Quando selecionamos um aplicativo, é revelado configurações e grupos de configurações deste aplicativo individualmente. Selecionando uma destas configurações é revelado 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.



## UITabBarController

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

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

Esta interface exibe guias ou abas em uma barra na parte inferior da tela. Em cada aba está relacionada 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.

Cada aba de um Tab Bar Controller está associada a uma View Controller personalizada, como dito anteriormente. 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.

<imagem da tabbar>

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 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.