# Unidade 7 - Toques finais

Nesta unidade iremos implementar os últimos passos para nossa aplicação básica. Iremos construir Views reusáveis, fazer nosso app consumir um serviço do Firebase e executar músicas.

# Aula 1

Nesta aula iremos aprender maneiras especiais de como melhorar o desenvolvimento de Views no iOS. Em comparação com o Android, onde a melhor forma para compor Views é via código, a Apple volta seus esforços para que possamos criar views sem programar uma sequer linha de código, utilizando 100% os recursos dos interface builder. Nesta aula iremos aprender como desenhar formas geométricas em views e visualizá-las em tempo real no Interface Builder.

## Melhorando o Interface Builder

A criação de Views pode ser massiva e repetitiva. Criar views utilizando código, apesar de nos proporcionar mais controle, é uma tarefa árdua e improdutiva, então o que nos resta é utilizar os recursos do interface builder.

O interface builder é uma ferramenta preciosa para agilizar o processo de criação das views. Com simples “clicar e arrastar” podemos compor desde views mais simples até views mais complexas. Porém ainda temos um problema. O retrabalho.

Apesar de termos o poder de criar views sem ao menos escrever uma linha de código, ainda teremos a necessidade de reusar views em diferentes storyboards. Isto impacta em reconfigurarmos toda a interface da view, tal como sua aparência ou suas restrições (auto-layout).

Para evitar este retrabalho utilizamos, logicamente, a sagrada herança.

### Reutilizando Views

Quando surge a necessidade de reutilizar a mesma view em diferentes storyboards, podemos criar uma classe personalizada para estas views.

No primeiro momento, subclassear uma view para fazer reuso da mesma impacta na composição do layout via código.

Configurando uma subclasse de view via código permite que reusemos a view em outras circunstâncias, e garante que a view terá o comportamento esperado em qualquer lugar que ela será utilizada. Mas lembra que falamos que esta é uma tarefa árdua e improdutiva? Sim, além disso, poderemos ver o resultado apenas ao executar o aplicativo, e isto acarreta em tempo, pois o aplicativo demora consideravelmente para ser compilado.

Tendo isto em vista a Apple criou o live rendering (ou rederização em tempo real) que permite que vejamos o resultado da nossa codificação no interface builder instantaneamente a medida que modificamos o código.

### Renderizando Views no Interface Builder

Para renderizar as modificações via código no interface builder em tempo real, temos que utilizar algumas técnicas disponíveis:

* **Draw Rect:** Somente as alterações da view feitas dentro do método draw(\_ :) serão aplicadas no interface builder, além disso elas serão mostradas apenas se a classe for Designable.
* **IBDesignable:** Tornar uma view Designable significa que ela será desenhável pelo interface builder. Isto faz com que o código contido no método draw(\_ :) seja mostrado no interface builder.
* **IBInspectable:** Como auxílio adicional, poderemos adicionar propriedades ao painel de propriedades do interface builder, assim poderemos configurar cores, tamanho e outras propriedades que não são padrões da view diretamente do painel de propriedades do interface Builder.

Vejamos a seguir com mais detalhes o que cada um dos items acima representam na prátican

#### O drawRect()

Segundo uma definição básica dada pela Apple, o draw(\_ :) é um método contido na UIView que desenha a imagem do receptor dentro de um dado retângulo. Exemplo da definição do método:

override func drawRect(rect: CGRect) {

}

Dentro deste método podemos desenhar qualquer forma ou imagem na UIView. Basta termos um pouco de conhecimento em geometria analítica.

Toda a construção gráfica e visual padrão que temos no iOS é desenhada pelo framework CoreGraphics. Ele usa dados e cálculos geométricos para desenhar qualquer coisa na tela. Os componentes que personalizamos (View, Button, TextField, Label) são meras abstrações se comparados com o meio que o CoreGraphics da vida ao visual do seu app.

Estes cálculos de geometria analítica podem parecer complicados no início, mas lembra da brincadeira de ligar os pontos para formar um desenho que brincávamos na nossa infância? A base do CoreGraphics é isto: Definir e ligar pontos para criar uma forma geométrica, e o conjuto de várias formas resulta em um desenho. Mas não vamos entrar muito em detalhes no CoreGraphics e veremos apenas o que realmente é bastante utilizado.

Imagine que queiramos desenhar a seguinte imagem (um botão circular de “Adicionar”):



Figura - Botão adicionar

Bastaria apenas criar uma subclasse de UIView e inserir implementar o draw(\_ :) como no código abaixo:

import UIKit

class PushButtonView: UIButton {

var fillColor: UIColor = UIColor.green

override func draw(\_ rect: CGRect) {

let path = UIBezierPath(ovalIn: rect)

fillColor.setFill()

path.fill() // Preenche a forma geométrica

// Configurando as variáveis de altura e largura

// ... para a linha horizontal

let plusWidth: CGFloat = min(bounds.width, bounds.height) \* 0.6

// Inicializa um objeto responsável por desenhar um "caminho"

let plusPath = UIBezierPath()

// Define que a largura do caminho será igual a altura da linha horizontal

// Move para o ponto inicial do caminho para o ponto inicial da linha

plusPath.move(to: CGPoint(

x:bounds.width/2 - plusWidth/2 + 0.5,

y:bounds.height/2 + 0.5))

// Adiciona um ponto ao caminho no final da linha

plusPath.addLine(to: CGPoint(

x:bounds.width/2 + plusWidth/2 + 0.5,

y:bounds.height/2 + 0.5))

// ... para linha vertical

// Move para o ponto inicial da linha vertical

plusPath.move(to: CGPoint(

x:bounds.width/2 + 0.5,

y:bounds.height/2 - plusWidth/2 + 0.5))

plusPath.addLine(to: CGPoint(

x:bounds.width/2 + 0.5,

y:bounds.height/2 + plusWidth/2 + 0.5))

// Define a cor das linhas

UIColor.white.setStroke()

// Aplica o desenho das linhas definidos acima

plusPath.stroke()

}

}

Parece longo? Sim, mas lembre-se que temos que construir ponto a ponto da nossa imagem.

Não se preocupe com o código agora. No final desta aula iremos construir passo-a-passo um botão personalizado para nosso Player musical.

Antes de irmos para o próximo assunto, repare que o desenho gerado por este método passa a ser exibido nos interface builders:

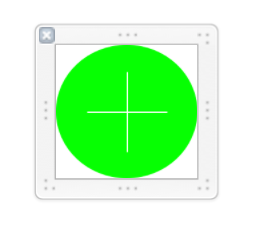


Figura - Botão sendo renderizado no IB (Interface Builder)

#### IBDesignables

Escrever um código para desenhar algo na tela e rodar o aplicativo para então ver o resultado final pararece um pouco desmotivador e improdutivo, mas temos opções para isto.

Live rendering ou renderização em tempo real é uma feature presente desde o Xcode 6, e permite que você dê o atributo @IBDesignable as suas views. Com este atributo a view se atualizará imediatamente no interfacebuilder à medida que você altera o método draw(\_ :).

O IBDesignable é uma anotação do Objective-C que colocamos em nossas classes de view que indica ao compilador que esta classe (subclasse de UIView) será Designable pelo interface builder, ou seja, tudo que desenharmos nela será exibido pelo interface builder nos lugares que a usarem. Vamos ser mais específicos.

No exemplo acima, do botão adicionar, precisaríamos apenas adicionar a anotação @IBDesignable no início da classe:

@IBDesignable

class PushButtonView: UIButton {

/...

}

Com isto, tudo que desenharmos no método draw(\_ :) será exibido instantaneamente no interface builder.

#### IBInspectables

O @IBInspectable é um atributo que você pode adicionar às propriedades que fazem com que elas sejam legíveis pelo Interface Builder. Isto significa que você poderá configurar a cor de preenchimento do botão dos exemplos acima através do interface builder ao invés de fazer isto via código.

Ao adicionarmos a anotação @IBInspectable na propriedade fillColor, como no seguinte trecho de código:

@IBInspectable var fillColor: UIColor = UIColor.green

Teríamos o seguinte resultado no nosso interface builder:

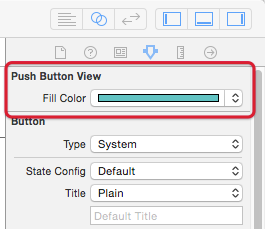


Figura - Trocando a cor de preenchimento do botão utilizando o IB

## Melhorando nosso player de música

Para aplicar os conhecimentos adquiridos nesta aula, iremos criar um simples ícone. Este ícone irá acompanhar a view do mini-player, que é a view que aparece quando selecionamos uma música na tela Navegar. Então iremos criar a seguinte forma geométrica:



Figura - Seta apontada para cima

Esta é uma seta que poderá estar apontando para baixo:



Figura - Seta apontada para baixo

Além disso poderá assumir diversos tamanhos e cores.

A definição do tamanho será feita através dos controles padrões da UIView e a orientação e cores serão feitos através dos IBInspectables que criaremos. Assim toda a personalização da seta ficará por conta do interface builder.

DICA: Estas setas tem um significado que pode ser entendido por qualquer um que as veem. A seta apontada para baixo indica que existe um conteúdo expansível que pode ser acessado ao tocá-la ou remete a ação de jogar a View para baixo (nosso caso). A seta apontada para cima, tem o significado inverso, pois ela indica que existe um conteúdo que pode ser recolhido/ocultado ao tocá-la ou remete a ação de jogar a View para cima (nosso caso). Você entenderá melhor na execução do exemplo.

Para mostrar o reuso da view, utilizaremos a seta apontada para cima para a seguinte view:

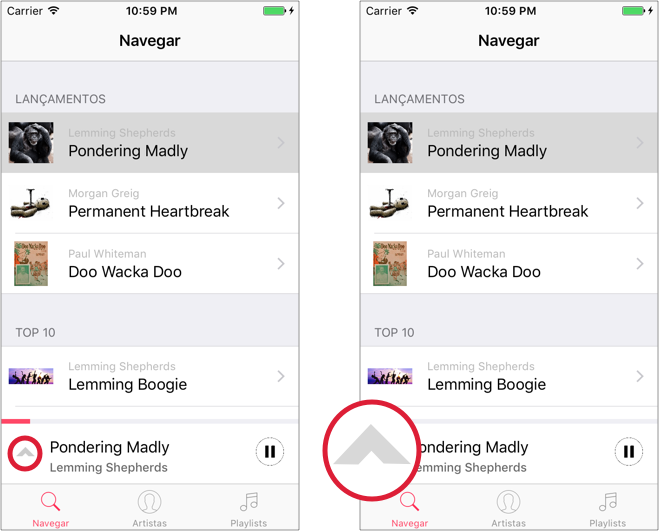


Figura - Seta apontada para cima em evidência na cena Navegar

OBS: As imagens e nomes finais das músicas serão definidos na próxima aula.

E a seta para baixo na seguinte View:



Figura - Seta apontada para baixo em evidência na cena Música

Agora que temos a direção, vamos começar a desenvolver nossa seta. Mas antes de começar, preparamos um ponto de partida para você.

Material de apoio:   
 O ponto de partida para esta atividade se encontra no arquivo Unidade\_7\_-Aula*\_*1-\_Exemplo\_1.zip, na pasta do material de apoio. Extraia e execute o projeto.

### Criando um arquivo de layout (.xib)

Existem duas opções aqui. Criar uma subclasse de UIButton e personalizar sua aparência 100% via código ou utilizando o auxílio de um interface builder. Iremos utilizar a segunda opção para vermos a atualização da view em tempo real. Então siga os seguintes passos:

1. Selecione o group **Views** que já criamos para você:

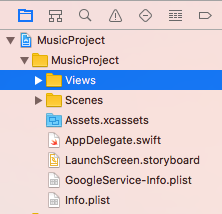


Figura - Selecionando o group View

1. Crie um novo arquivo pelo menu **File > New > File**:

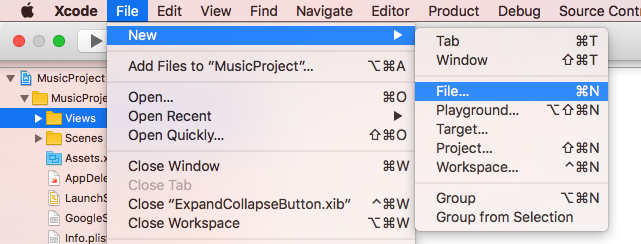


Figura - Caminho para criação de novo arquivo

1. Na nova tela que se abriu selecione a opção **View** e clique em **Next**:

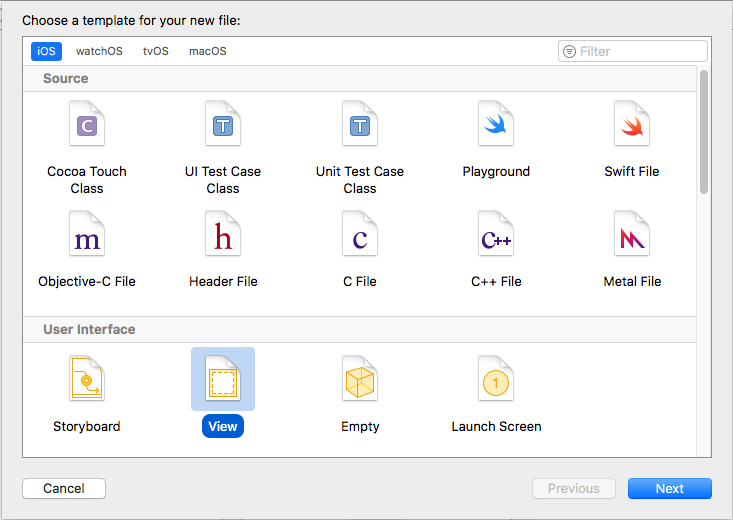


Figura - Escolhendo um arquivo de View

1. Neste passo escolha o nome **ExpandCollapseButton** para a view e clique em **Create**.
2. Você terá um novo **.xib**:

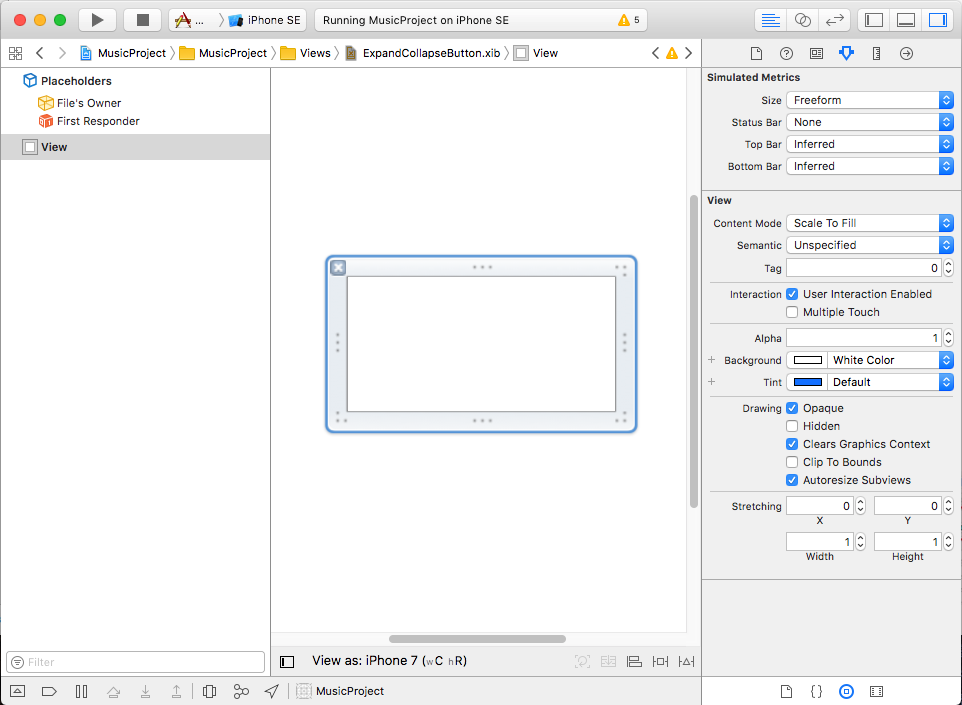


Figura - Arquivo criado

1. Agora precisamos fazer alguns ajustes iniciais. Selecione a **View**:

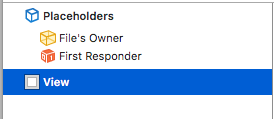


Figura - Selecionando a VIew

1. Na aba **Attributes inspector**, vá até a propriedade **Size** e escolha a opção **Freeform**:

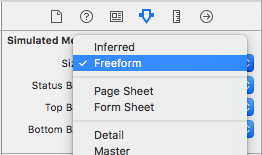


Figura - Mudando a forma de dimensionamento da View

1. Na aba **Size inspector** defina a largura e altura da view para **200x100**:

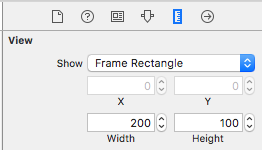


Figura - Configurando o tamanho da View

As configurações iniciais do nosso .xib estão feitas. Agora precisamos criar o File Owner da view.

### Criando uma subclasse de UIButton

Um .xib é bastante limitado quando não temos uma subclasse que o rege. Então vamos criar uma subclasse de botão para ele.

1. Selecione o group **Views**, como feito nos passos anteriores.
2. Crie um novo arquivo pelo menu **File > New > File.**
3. Na nova tela que se abriu selecione a opção **Cocoa Touch Class** e clique em **Next**:

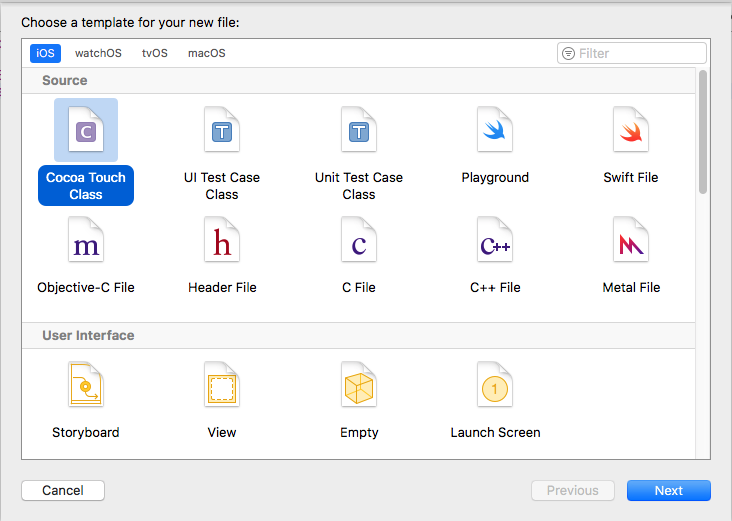


Figura - Escolhendo arquivo do tipo Cocoa Touch Class

1. No campo **Class** utilize o nome **ExpandCollapseButton** , no campo **Subclass of** escolha **UIButton** e clique em **Next**:

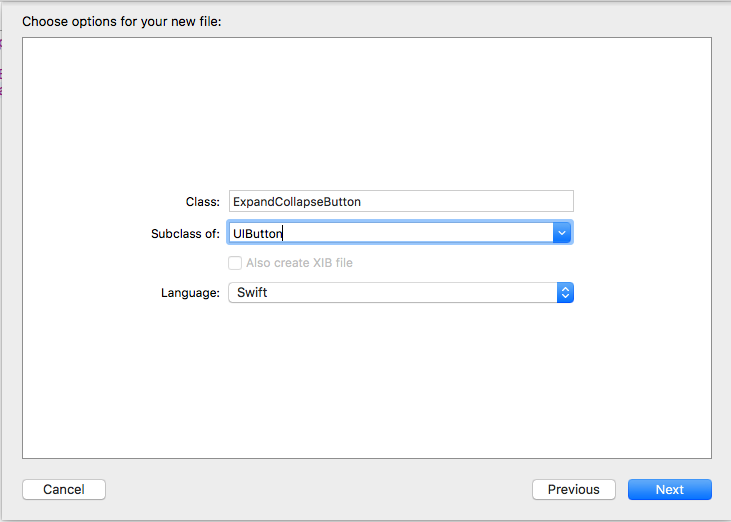


Figura - Definindo nome e subtítulo do novo arquivo

Neste passo estamos criando o arquivo que será o dono do xib criado anteriormente. No caso nossa classe será um Button.

1. Você terá o seguinte arquivo:

import UIKit

class ExpandCollapseButton: UIButton {

/\*

// Only override draw() if you perform custom drawing.

// An empty implementation adversely affects performance during animation.

override func draw(\_ rect: CGRect) {

// Drawing code

}

\*/

}

1. E seu group **Views** estará assim:
2. Descomente o método **draw(\_ :)**:

import UIKit

class ExpandCollapseButton: UIButton {

override func draw(\_ rect: CGRect) {

}

}

1. Temos nossos dois arquivos necessários criados. Agora precisamos vinculá-los. Entre no arquivo **ExpandCollapseButton.xib** e selecione a **View**:

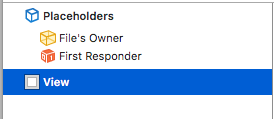


Figura - Selecionando a View

1. Na aba Identity inspector escreva ExpandCollapseButton no campo Class:

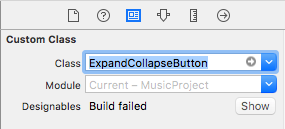


Figura - Definindo uma classe para a View

Isto fará com que a view selecionada obedeça a classe **ExpandCollapseButton** definida no arquivo **ExpandCollapseButton.swift**.

Agora o que precisamos é finalmente desenhar a seta.

### Desenhando no botão

Vamos codificar nossa classe ExpandCollapseButton. Abra o arquivo **ExpandCollapseButton.swift** e siga os seguintes passos:

1. Para decidir se a seta será voltada para cima ou para baixo precisamos de uma propriedade que indique isto, então crie a seguinte propriedade:

// O estado padrão da seta é apontado para cima

var isPointingToDown : Bool = false

DICA: Lembre-se que propriedades se equivale aos atributos em Java, portanto elas devem ser declaradas dentro de uma classe.

1. Agora precisamos criar a propriedade para a cor da seta:

// A cor padrão da seta será cinza claro

var arrowColor : UIColor = UIColor.lightGray

1. Sua classe por enquanto estará da seguinte maneira:

import UIKit

class ExpandCollapseButton: UIButton {

var isPointingToDown : Bool = false

var arrowColor : UIColor = UIColor.lightGray

override func draw(\_ rect: CGRect) {

}

}

1. Agora basta desenharmos a seta. Insira o seguinte código no método **drawRect(\_ :):**

let shape = CAShapeLayer()

layer.addSublayer(shape)

shape.lineJoin = kCALineJoinMiter

shape.fillColor = arrowColor.cgColor

let path = UIBezierPath()

if isPointingToDown {

// Apontando para baixo

path.move(to: CGPoint(x:bounds.width/2, y:bounds.height))

path.addLine(to: CGPoint(x:bounds.width, y:0))

path.addLine(to: CGPoint(x:(bounds.width/3) \* 2, y:0))

path.addLine(to: CGPoint(x:bounds.width/2, y:bounds.width/6))

path.addLine(to: CGPoint(x:bounds.width/3, y:0))

path.addLine(to: CGPoint(x:0, y:0))

} else {

// Apontando para cima

path.move(to: CGPoint(x:bounds.width/2, y:0))

path.addLine(to: CGPoint(x:bounds.width, y:bounds.height))

path.addLine(to: CGPoint(x:(bounds.width/3) \* 2, y:bounds.height))

path.addLine(to: CGPoint(x:bounds.width/2, y:bounds.height - (bounds.width/6)))

path.addLine(to: CGPoint(x:bounds.width/3, y:bounds.height))

path.addLine(to: CGPoint(x:0, y:bounds.height))

}

path.close()

shape.path = path.cgPath

OBS: Os conceitos do CoreGraphics são extensos e com bastante teor matemático, então se você desejar mais detalhes sobre o uso deste Framework, visite sua documentação. Leia com atenção o código, mas não se preocupe em entender. O objetivo desta aula é apenas lhe apresentar a ferramenta.

1. Sua classe por completo ficará assim:

import UIKit

class ExpandCollapseButton: UIButton {

var isPointingToDown : Bool = false

var arrowColor : UIColor = UIColor.lightGray

override func draw(\_ rect: CGRect) {

let shape = CAShapeLayer()

layer.addSublayer(shape)

shape.lineJoin = kCALineJoinMiter

shape.fillColor = arrowColor.cgColor

let path = UIBezierPath()

if isPointingToDown {

// Apontando para baixo

path.move(to: CGPoint(x:bounds.width/2, y:bounds.height))

path.addLine(to: CGPoint(x:bounds.width, y:0))

path.addLine(to: CGPoint(x:(bounds.width/3) \* 2, y:0))

path.addLine(to: CGPoint(x:bounds.width/2, y:bounds.width/6))

path.addLine(to: CGPoint(x:bounds.width/3, y:0))

path.addLine(to: CGPoint(x:0, y:0))

} else {

// Apontando para cima

path.move(to: CGPoint(x:bounds.width/2, y:0))

path.addLine(to: CGPoint(x:bounds.width, y:bounds.height))

path.addLine(to: CGPoint(x:(bounds.width/3) \* 2, y:bounds.height))

path.addLine(to: CGPoint(x:bounds.width/2, y:bounds.height - (bounds.width/6)))

path.addLine(to: CGPoint(x:bounds.width/3, y:bounds.height))

path.addLine(to: CGPoint(x:0, y:bounds.height))

}

path.close()

shape.path = path.cgPath

}

}

Se você colocar este botão em algum ViewController, poderá ver o desenho da seta somente quando executar o app. Para ver o resultado instantâneamente, vamos utilizar o atributo IBDesignable.

### Exibindo atualizações em tempo real

Para que você possa ver o resultado do desenho em tempo real no xib que criamos, basta realizar os seguintes passos:

1. No início da definição da classe **ExpandCollapseButton** adicione a anotação **@IBDesignable** como a seguir:

@IBDesignable class ExpandCollapseButton: UIButton {

// ...

}

ou pode ser inserido acima da definição da classe:

@IBDesignable

class ExpandCollapseButton: UIButton {

// ...

}

1. Abra o arquivo ExpandCollapseButton.xib e selecione a view Expand Collapse Button:

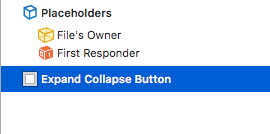


Figura - Selecionando o Expand Collapse Button

1. Veja na aba **Identity inspector** que aparece um estado dos Designables, que passará de **Updating** para **Up to date** ou **Error**. Quando estiver no estado **Up to date** você conseguirá ver o desenho no xib:

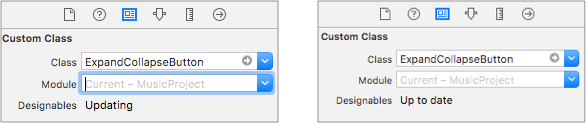


Figura - Mudança de estado dos Designables

A atualização em tempo real está pronta. Agora iremos adicionar propriedades no xib utilizando os IBInspectable.

### Tornando propriedades Inspectables

Para utilizar IBInspectable precisaremos de poucas modificações no código. Siga os seguintes passos:

1. Volte para nossa classe **ExpandCollapseButton** modifique as duas propriedades que criamos, utilizando o atributo **@IBInspectable**:

@IBInspectable

var isPointingToDown : Bool = false

@IBInspectable

var arrowColor : UIColor = UIColor.lightGray

1. Para garantir que o visual irá se atualizar quando modificarmos os Inspectables, precisamos trasformar nossas propriedades em variáveis/propriedades computadas. Faça a seguinte modificação:

@IBInspectable

var isPointingToDown : Bool = false {

didSet {

setNeedsDisplay()

}

}

@IBInspectable

var arrowColor : UIColor = UIColor.lightGray {

didSet {

setNeedsDisplay()

}

}

DICA: Como não podemos chamar diretamente o método draw(\_ :), por questões arquiteturais do UIKit, chamamos método setNeedsDisplay que força a atualização da View. Quando forçamos a atualização da View com este método, todo o ciclo de vida dela será executado, e o método draw(\_ :) será invocado neste ciclo.

1. A classe **ExpandCollapseButton** estará da seguinte maneira:

import UIKit

@IBDesignable

class ExpandCollapseButton: UIButton {

@IBInspectable

var isPointingToDown : Bool = false {

didSet {

setNeedsDisplay()

}

}

@IBInspectable

var arrowColor : UIColor = UIColor.lightGray {

didSet {

setNeedsDisplay()

}

}

override func draw(\_ rect: CGRect) {

let shape = CAShapeLayer()

layer.addSublayer(shape)

shape.lineJoin = kCALineJoinMiter

shape.fillColor = arrowColor.cgColor

let path = UIBezierPath()

if isPointingToDown {

// Apontando para baixo

path.move(to: CGPoint(x:bounds.width/2, y:bounds.height))

path.addLine(to: CGPoint(x:bounds.width, y:0))

path.addLine(to: CGPoint(x:(bounds.width/3) \* 2, y:0))

path.addLine(to: CGPoint(x:bounds.width/2, y:bounds.width/6))

path.addLine(to: CGPoint(x:bounds.width/3, y:0))

path.addLine(to: CGPoint(x:0, y:0))

}else{

// Apontando para cima

path.move(to: CGPoint(x:bounds.width/2, y:0))

path.addLine(to: CGPoint(x:bounds.width, y:bounds.height))

path.addLine(to: CGPoint(x:(bounds.width/3) \* 2, y:bounds.height))

path.addLine(to: CGPoint(x:bounds.width/2, y:bounds.height - (bounds.width/6)))

path.addLine(to: CGPoint(x:bounds.width/3, y:bounds.height))

path.addLine(to: CGPoint(x:0, y:bounds.height))

}

path.close()

shape.path = path.cgPath

}

}

1. Abra o arquivo ExpandCollapseButton.xib e selecione a view Expand Collapse Button.
2. Na aba **Attribute inspector** veja que apareceram as duas propriedades que transformamos em Inspectable:

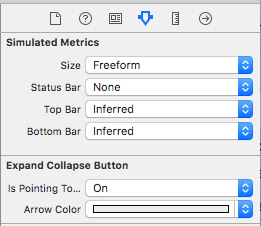


Figura - Propriedades Inspectable sendo exibidas no IB

1. Modifique-as como quiser.

### Utilizando nosso botão

Agora iremos aplicar nossa seta no mini player. Siga os passos:

1. Abra a Main.storyboard:

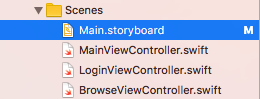


Figura - Selecionando o arquivo Main.storyboard

1. Na cena **Navegar**, selecione o botão que nomeamos de **Expand Button**:

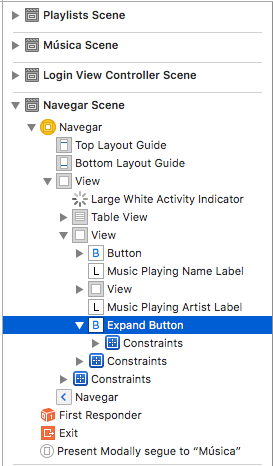


Figura - Selecionando Expand Button

1. Na aba Identity inspector defina no campo Class o nome ExpandCollapseButton:

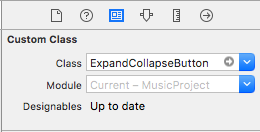


Figura - Definindo a classe da View

1. Veja que a seta irá aparecer no lugar do botão:

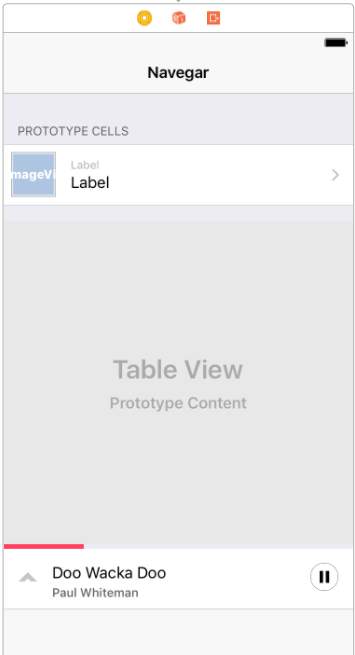


Figura - Designable sendo exibido na cena Navegar

Nós já definimos a ação para este botão. Quando você o tocar, a cena **Música** será aberta.

A última coisa que falta é inserir este mesmo botão na cena **Música**, mas desta vez a seta apontará para baixo e será da cor branca. Siga os passos:

1. No mesmo storyboard (Main.storyboard), desta vez na cena **Música**, localize o botão que nomeamos **Collapse Button**:

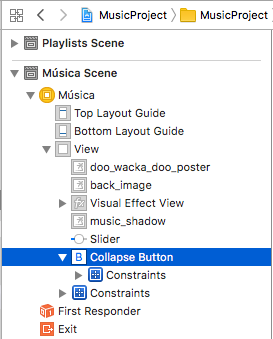


Figura - Selecionando o Collapse Button

1. Repita o passo 3 para este botão:

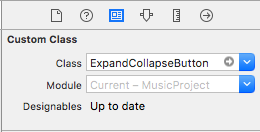


Figura - Definindo a classe do Collapse Button

1. Aguarde até os inspectables aparecer e selecione **On** na propriedade **Is Pointing To Down** e escolha a cor branca na propriedade **Arrow Color**:

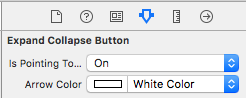


Figura - Definindo a cor do Collapse Button através do nosso Inspectable

1. Espere a view atualizar, e teremos uma nova seta:



Figura - Resultado final do Collapse Button

Com este passo-a-passo vimos uma maneira incrível de reaproveitar Views e fazer desenhos complexos nas Views.

## Resumo

## Exercícios

## TDP

# Aula 2

# Manipulando dados obtidos pela rede

Continuando o ensinamento sobre o Firebase que aprendemos na unidade passada, agora iremos fazer uso do Firebase Database no nosso app. Todo o conteúdo dinâmico estará lá, basta apenas consumí-lo.

## Firebase Database

Relembrando a sua funcionalidade, o Firebase Realtime Database é um banco de dados hospedado na nuvem. Os dados são armazenados como JSON e sincronizados em tempo real com todos os clientes conectados. Quando você cria aplicativos multiplataforma com os SDKs para iOS, Android e JavaScript, todos os seus clientes compartilharão uma instância de Realtime Database e automaticamente receberão atualizações com os dados mais recentes.

Nesta aula iremos utilizar o Firebase Database para obter as músicas da tela Navegar e teremos um resultado parecido com este:

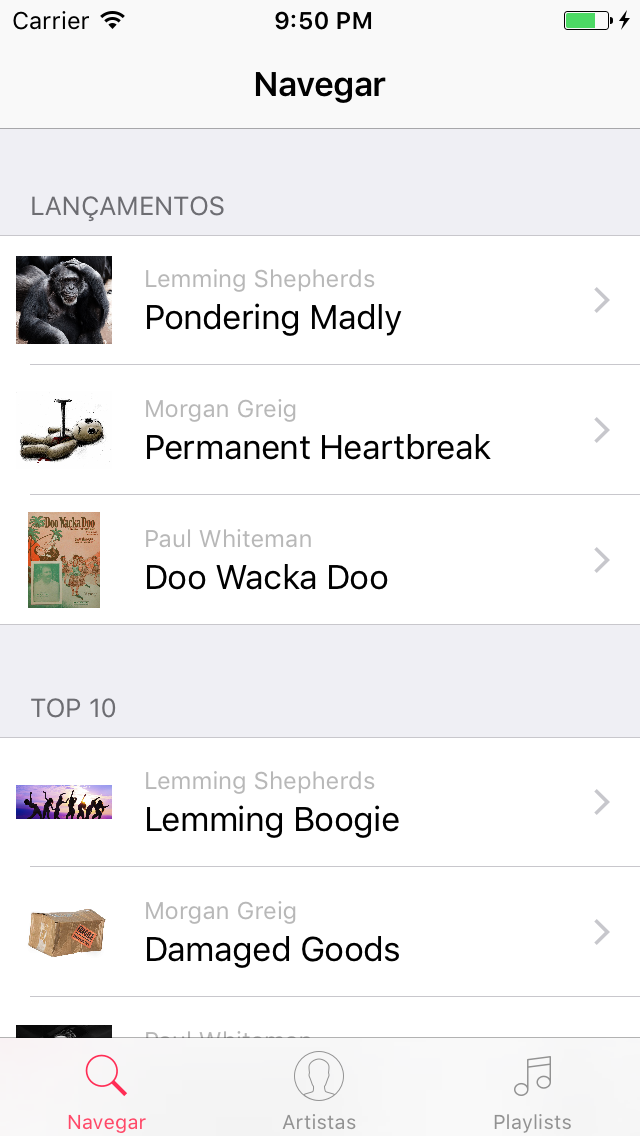


Figura - Resultado esperado para o final desta aula

Antes de implementarmos vamos entender a estrutura de dados.

## Estrutura do JSON

Lembra do projeto Android, onde requisitávamos um JSON na tela Inbox que continha informações sobre os e-mails da caixa de entrada do usuário? Pois bem, iremos fazer algo parecido aqui.

Quando a tela Navegar aparecer na tela será feita uma requisição ao Firebase pedindo as músicas em “Lançamento” e “Top 10”. Estas músicas terão a informação de seu nome, nome do autor e URL da capa (capa do álbum). Para isto iremos estruturar um JSON.

Esclarecimento: Como estamos lidando com um aplicativo de música teremos que utilizar como exemplo músicas em domínio público para não infligir possíveis direitos autorais. Para nível de estudo você pode utilizar como exemplo as músicas que você desejar, mas não poderá publicar o aplicativo na App Store. Na verdade poderá, mas você terá sérios problemas com os direitos autorais dos autores, ok?

Nosso JSON de exemplo é o seguinte:

{  
 "navegar": {  
 "top\_10": [  
 {  
 "nome": "Lemming Boogie",  
 "artista": "Lemming Shepherds",  
 "imagem": "https://firebasestorage.googleapis.com/v0/b/musicproject-63abd.appspot.com/o/Lemming%20Shepherds%2Fmusicas%2FLemming%20Boogie.jpg?alt=media&token=f50b54d5-162f-49d6-99c6-e4fa6e0615bd",  
 "mp3" : "https://firebasestorage.googleapis.com/v0/b/musicproject-63abd.appspot.com/o/Lemming%20Shepherds%2Fmusicas%2FLemming%20Boogie.mp3?alt=media&token=1ab50129-3e16-49f8-8050-ac0fa58ba8df"  
 },  
 {  
 "nome": "Damaged Goods",  
 "artista": "Morgan Greig",  
 "imagem": "https://firebasestorage.googleapis.com/v0/b/musicproject-63abd.appspot.com/o/Morgan%20Greig%2Fmusicas%2FDamaged%20Goods.jpg?alt=media&token=3fd3070e-046e-4b48-b418-5b3108630bfd",  
 "mp3" : "https://firebasestorage.googleapis.com/v0/b/musicproject-63abd.appspot.com/o/Morgan%20Greig%2Fmusicas%2FDamaged%20Goods.mp3?alt=media&token=35885090-d54e-4622-aead-224e9e1435b5"  
 },  
 {  
 "nome": "All of Me, ft. Mildred Bailey",  
 "artista": "Paul Whiteman",  
 "imagem": "https://firebasestorage.googleapis.com/v0/b/musicproject-63abd.appspot.com/o/Paul%20Whiteman%2Fmusicas%2FAll%20of%20Me%2C%20ft.%20Mildred%20Bailey.jpg?alt=media&token=7d0d02a4-f56b-42cd-97e1-9392bca36e66",  
 "mp3" : "https://firebasestorage.googleapis.com/v0/b/musicproject-63abd.appspot.com/o/Paul%20Whiteman%2Fmusicas%2FAll%20of%20Me%2C%20ft.%20Mildred%20Bailey.mp3?alt=media&token=1431622c-7532-44d1-934d-a9332ab5df31"  
 }  
 ],  
 "lancamentos": [  
 {  
 "nome": "Pondering Madly",  
 "artista": "Lemming Shepherds",  
 "imagem": "https://firebasestorage.googleapis.com/v0/b/musicproject-63abd.appspot.com/o/Lemming%20Shepherds%2Fmusicas%2FPondering%20Madly.jpg?alt=media&token=1e371568-33c0-4a2b-80eb-24f54923f823",  
 "mp3" : "https://firebasestorage.googleapis.com/v0/b/musicproject-63abd.appspot.com/o/Lemming%20Shepherds%2Fmusicas%2FPondering%20Madly.mp3?alt=media&token=33732288-af31-45bc-8439-2ff969f5f6bf"  
 },  
 {  
 "nome": "Permanent Heartbreak",  
 "artista": "Morgan Greig",  
 "imagem": "https://firebasestorage.googleapis.com/v0/b/musicproject-63abd.appspot.com/o/Morgan%20Greig%2Fmusicas%2FPermanent%20Heartbreak.jpg?alt=media&token=0774db0b-5eee-4fe9-b94c-093df44d8d43",  
 "mp3" : "https://firebasestorage.googleapis.com/v0/b/musicproject-63abd.appspot.com/o/Morgan%20Greig%2Fmusicas%2FPermanent%20Heartbreak.mp3?alt=media&token=a32de2f5-189f-460e-9f44-e8e146a7f68a"  
 },  
 {  
 "nome": "Doo Wacka Doo",  
 "artista": "Paul Whiteman",  
 "imagem": "https://firebasestorage.googleapis.com/v0/b/musicproject-63abd.appspot.com/o/Paul%20Whiteman%2Fmusicas%2FDoo%20Wacka%20Doo.png?alt=media&token=c8d29064-a50f-4016-94ae-91ea31e26c92",  
 "mp3" : "https://firebasestorage.googleapis.com/v0/b/musicproject-63abd.appspot.com/o/Paul%20Whiteman%2Fmusicas%2FDoo%20Wacka%20Doo.mp3?alt=media&token=8f184ca5-cdc4-4b69-b023-f8f0cc0eeff9"  
 }  
 ]  
 }  
}

Veja que a raiz do JSON é o objeto "navegar":

Caso você queira adicionar informações para a aba “Artistas” e “Playlists”, basta adicionar objetos no mesmo nível do "navegar".

Dentro do objeto "navegar" temos mais dois objetos:

* **"top\_10":** Neste objeto está contido uma lista de objetos que representam as músicas que entrarão na seção “Top 10”.
* **"lancamentos":** Neste objeto está contido uma lista de objetos que representam as músicas que entrarão na seção “Lançamentos”.

Os dois objetos acima possuem listas de músicas que por sua vez é representada pelo seguinte objeto:

{  
 "nome": "...",  
 "artista": "...",  
 "imagem": "https://...",  
 "mp3" : "https://..."  
}

Cada um destes campos representam uma informação de música:

* **"nome":** Indicará o nome da música
* **"artista":** Indicará o nome do autor (ou cantor) da música
* **"imagem":** Indicará o endereço na web (URL) da foto do album da música
* **"mp3":** Indicará o endereço na web (URL) da música em si, no formato mp3.

DICA: Na aula seguinte veremos como executar um arquivo mp3 na nossa aplicação.

No nosso exemplo, as imagens e os mp3s estão armazenados no serviço Storage do Firebase. Este serviço permite que guardemos arquivos diversos na plataforma. Estes arquivos podem ser inseridos tanto manualmente, utilizando o Firebase console, quanto via aplicação, ou seja, o usuário do seu aplicativo poderá enviar arquivos via app e salvar no Firebase Storage.

Agora que entendemos como será a estrutura do JSON precisamos adicioná-lo ao Firebase.

## Adicionando um JSON ao Firebase

O primeiro passo para consumirmos algum conteúdo no Firebase, é adicionar este conteúdo propriamente dito em formato JSON, utilizando o Firebase Console. Para isto execute os seguintes passos:

* Abra o Firebase Console no seu browser: <https://console.firebase.google.com/?hl=pt-BR>
* Entre no projeto **MusicProject**:



Figura - Localização do projeto no Console Firebase

1. Entre no menu **Database**:

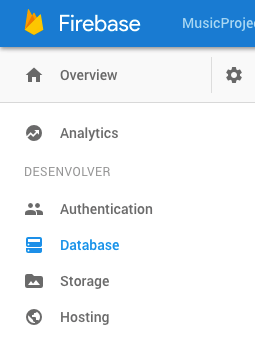


Figura - Localização da opção Database

1. Você será apresentado à seguinte tela:

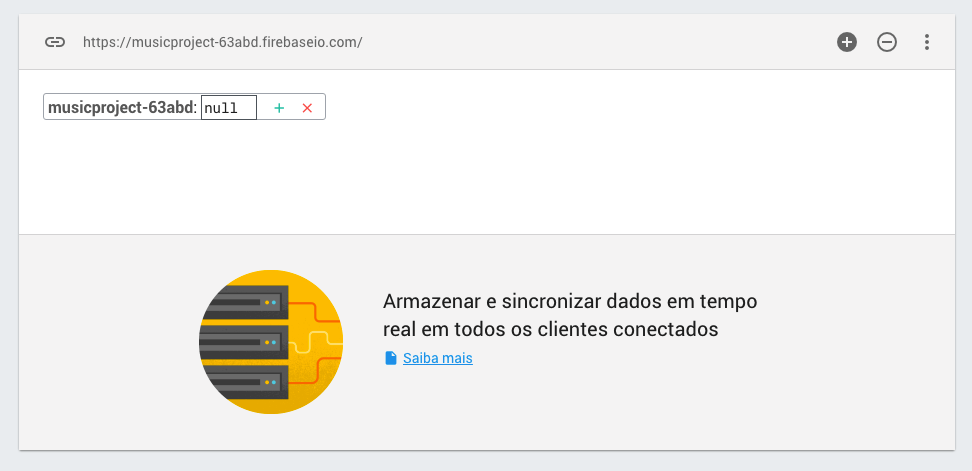


Figura - Firebase Database em seu estado inicial

* Agora baixe o JSON do exemplo acima no seguinte link: <https://firebasestorage.googleapis.com/v0/b/musicproject-63abd.appspot.com/o/conteudo.json?alt=media&token=69bb4fd4-6529-46bc-bb2d-a821ba8048aa>  
  O nome do arquivo baixado será **conteudo.json**
* Com o JSON já baixado, clique no botão “…” e o menu overflow irá aparecer:

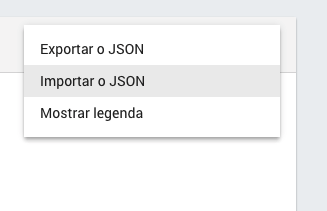


Figura - Localização da opção Importar JSON

* Clique no botão **Importar JSON** e a seguinte tela aparecerá:

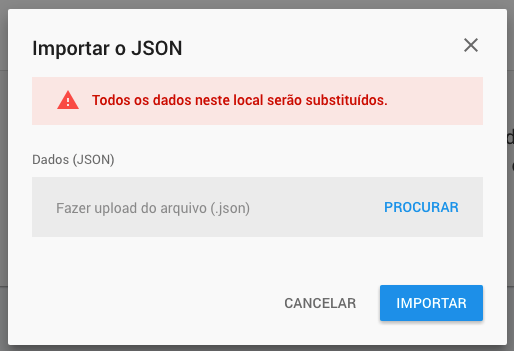


Figura - Tela Importar JSON

1. Clique em procurar e escolha o arquivo **conteudo.json**, baixado anteriormente.



Figura - JSON adicionado

* Clique em **Importar** e o JSON terá sido adicionado ao Firebase, como pode ser visto na seguinte tela:

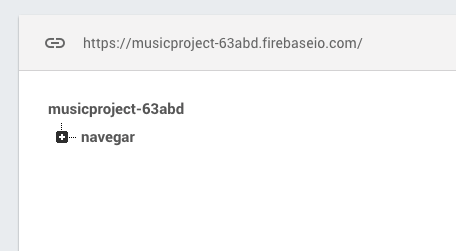


Figura - Representação do JSON adicionado no Firebase Database

* Você poderá expandir os objetos para ver a representação dos dados contidos no JSON:



Figura - JSON representado em forma de árvore

## Consumindo o serviço

Para executar este passo-a-passo você deverá baixar o ponto de partida que preparamos para você:

Material de apoio  
 O ponto de partida para esta atividade se encontra no arquivo Unidade\_7\_-Aula\_2-\_Exemplo\_1.zip, na pasta do material de apoio.

Neste ponto de partida temos uma configuração do Firebase que pode ser utilizada para seguir os próximos passos. Se você quiser utilizar sua própria configuração do Firebase, com suas músicas, JSONs e usuários, basta substituir o arquivo **GoogleService-Info.plist** contido no projeto Xcode pelo seu próprio **GoogleService-Info**, que pode ser baixado do Firebase console, como visto na unidade 6.

### Instanlando o Firebase Realtime Database

Iremos instalar o Firebase Database via CocoaPods então o procedimento é o mesmo das outras bibliotecas que instalamos:

1. Abra o arquivo Podfile
2. Insira a seguinte linha:

pod 'Firebase/Database'

1. Seu Podfile estará da seguinte maneira:

#Uncomment the next line to define a global platform for your project  
#platform :ios, '9.0'  
  
target 'MusicProject' do  
 # Comment the next line if you're not using Swift and don't want to use dynamic frameworks  
 use\_frameworks!  
  
 pod 'Firebase/Core'  
 pod 'Firebase/Auth'  
 pod 'Firebase/Database'  
end

1. Abra o terminal do seu Mac
2. Navegue a até o diretório do projeto:

cd caminho/para/seu/projeto

1. Execute o comando de instalação do pod:

**pod** install

O Firebase Database estará instalado.

### Utilizando o Firebase Database

Para utilizar o Firebase Database precisaremos executar alguns pequenos passos:

1. Entre no arquivo BrowseViewController.
2. No topo da classe (na região dos imports) importe a Firebase Database:

import FirebaseDatabase

1. Localize o método viewWillAppear(\_ :).
2. Neste método iremos fazer a chamada do JSON que criamos anteriormente. O efeito de utilizar o viewWillAppear(\_ :) é que sempre q a tela for exibida novamente o serviço será chamado novamente, assim mantendo os dados atualizados, caso você altere o JSON no Firebase. Insira o seguinte código:

let ref = FIRDatabase.database().reference()

self.tableView.isHidden = true

ref.child("navegar").observeSingleEvent(of: .value, with: { (snapshot) in

// Get user value

let value = snapshot.value as? JSONObject

self.browseData = Browse(from: value!)

self.tableView.reloadData()

self.tableView.isHidden = false

}) { (error) in

print(error.localizedDescription)

}

1. Vamos entender o código acima.

* Primeiro criamos uma referencia de um objeto FIRDatabase. É ele que manipulará todas as requisições de serviço:

let ref = FIRDatabase.database().reference()

* Logo após, ocultamos a Table View, para que uma Activity Indicator que colocamos atras da Table View seja exibida. Esta Activity Indicator indica ao usuário que o aplicativo está trabalhando, enquanto o Firebase não retorna uma resposta. Este tipo de feedback é otimo em casos de conexão lenta com a internet.

self.tableView.isHidden = true

* O próximo comando realiza a chamada do JSON, efetivamente. Ao especificarmos o atributo "navegar" , o Firebase irá entender qual objeto estamos requisitando e o retornará no objeto snapshot ou um error .

ref.child("navegar").observeSingleEvent(of: .value, with: { (snapshot) in

// Get user value

}) { (error) in

print(error.localizedDescription)

}

* O Snapshot pode ser convertido para JSONObject :

let value = snapshot.value as? JSONObject

DICA: JSONObject é um typealias para o tipo [String: Any] que é um Dictionary. Este typealias está definido no arquivo Models.swift

* Seguindo os próximos comandos, fazemos a conversão do JSONObject para o objeto Browse:

self.browseData = Browse(from: value!)

* Veja que preparamos a camada de model para você. Está contida no arquivo **Models.swift**. Esta camada faz a conversão do JSON para o objeto Browse e subobjetos Music:

import Foundation

/// MARK: Models para a cena Navegar

typealias JSONObject = [String:Any]

struct Browse {

let releases: [Music]

let top10: [Music]

init(from json: JSONObject) {

if let releasesArray = json["lancamentos"] as? [JSONObject] {

releases = releasesArray.map { Music(from: $0) }

}else{

releases = [Music]()

}

if let top10Array = json["top\_10"] as? [JSONObject] {

top10 = top10Array.map { Music(from: $0) }

}else{

top10 = [Music]()

}

}

}

struct Music {

let artistName: String?

let imageURL: String?

let mp3URL: String?

let name: String?

init(from json: JSONObject) {

artistName = json["artista"] as? String

imageURL = json["imagem"] as? String

mp3URL = json["mp3"] as? String

name = json["nome"] as? String

}

}

* Os próximos dois comandos atualizam os dados da tabela e em seguida deixa a table view visível novamente:

self.tableView.reloadData()

self.tableView.isHidden = false

## Exibindo imagens com a Kingfisher

Tendo nosso Firebase funcionando corretamente é hora de baixarmos as imagens especificadas pelas URLs no JSON. Para isto iremos adicionar mais uma dependência ao projeto, o Kingfisher.

### Kingfisher

A Kingfisher é uma biblioteca leve, escrita puramente em Swift que possibilita o download e cache de imagens da web no nosso aplicativo.



Figura - Logo da Kingfisher

**Features:**

* Download e cache de imagens assincronamente.
* Networking baseado no componente nativo do iOS, URLSession.
* Método de cache em disco ou em memória estão disponíveis.
* Download cancelável para melhorar a performance do app e economia no consumo de dados.
* Componentes separados. Você pode usar o sistema de download ou de cache separadamente se precisar.
* Pré-busca e exibição da imagem no cache antes, quando for necessário
* Extensions para UIImageView, NSImage e UIButton para definir a imagem diretamente de uma URL
* E muito mais

Para se ter uma ideia de como é fácil exibir imagens da web, utilizando o Kingfisher com e extension da UIImageView, precisamos apenas das linhas de código abaixo:

let url = URL(string: "url\_da\_sua\_imagem")

imageView.kf.setImage(with: url)

O Kingfisher irá fazer o download da imagem especificada pela url, enviá-la tanto para memória quanto para o disco como cache, e mostrá-la na imageView.

### Instalando o Kingfisher

Iremos instalar o Kingfisher via CocoaPods então o procedimento é o mesmo das outras bibliotecas que instalamos:

1. Abra o arquivo **Podfile**
2. Insira a seguinte linha:

pod 'Kingfisher', '~> 3.0'

1. Seu Podfile estará da seguinte maneira:

#Uncomment the next line to define a global platform for your project  
#platform :ios, '9.0'  
  
target 'MusicProject' do  
 # Comment the next line if you're not using Swift and don't want to use dynamic frameworks  
 use\_frameworks!  
  
 pod 'Firebase/Core'  
 pod 'Firebase/Auth'  
 pod 'Firebase/Database'  
 pod 'Kingfisher', '~> 3.0'  
  
end

1. Abra o terminal do seu Mac
2. Navegue a até o diretório do projeto:

cd caminho/para/seu/projeto

1. Execute o comando de instalação do pod:

pod install

Aguarde até o processo de instalação terminar e estará pronto. Vamos utilizá-la.

### Utilizando o Kingfisher

A utilização da Kingfisher é mais fácil do que a própria instalação, então execute os seguintes passos:

1. Entre no arquivo BrowseViewController.
2. No topo da classe (na região dos imports) importe a Kingfisher:

import Kingfisher

1. Localize o método tableView(\_ :, cellForRowAt:) .
2. Insira as linhas em destaque de modo que o método fique assim:

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

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

if indexPath.section == 0 {

cell.artistLabel?.text = browseData?.releases[indexPath.row].artistName ?? ""

cell.musicNameLabel?.text = browseData?.releases[indexPath.row].name ?? ""

//-----------------------------

// Insira as duas linhas de código abaixo

//-----------------------------

let url = URL(string: (browseData?.releases[indexPath.row].imageURL)!)\*\*

cell.musicImageView?.kf.setImage(with: url)

}else{

cell.artistLabel?.text = browseData?.top10[indexPath.row].artistName ?? ""

cell.musicNameLabel?.text = browseData?.top10[indexPath.row].name ?? ""

//-----------------------------

// Insira as duas linhas de código abaixo

//-----------------------------

let url = URL(string: (browseData?.top10[indexPath.row].imageURL)!)

cell.musicImageView?.kf.setImage(with: url)

}

return cell

}

Como na demonstração do tópico 2.5.1. precisamos apenas de duas linhas para configurar a imagem, uma para instanciar a URL e outra para definir a imagem da ImageView a partir da url.

Execute o projeto e veja o resultado.

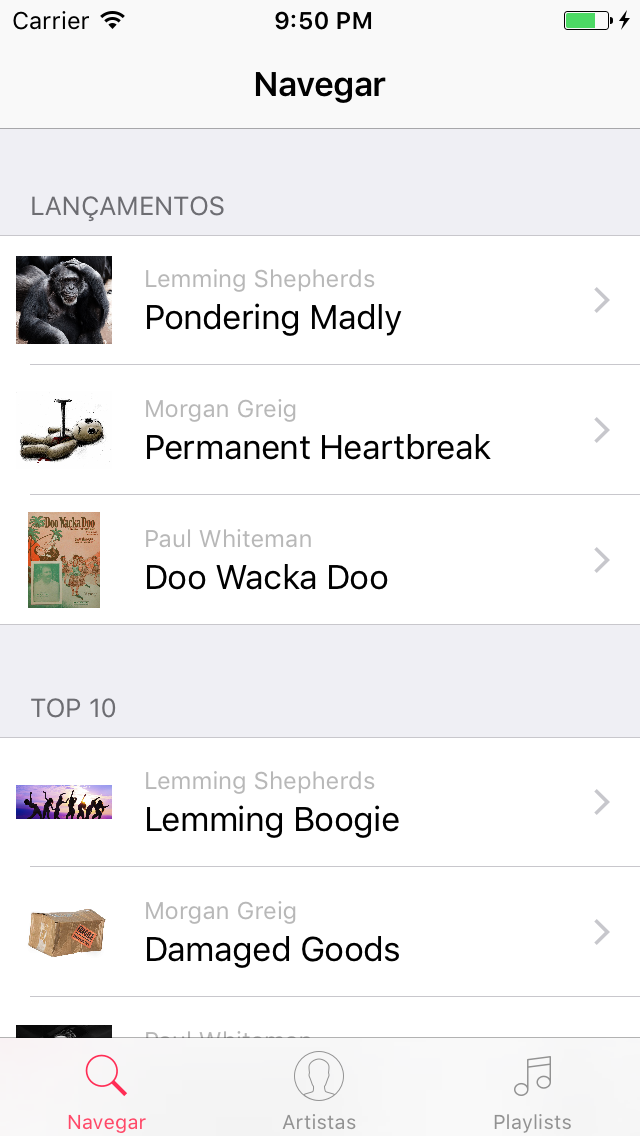


Figura - Resultado final

A parte visual do nosso aplicativo está finalizada. Na aula que vem aprenderemos a executar um som em mp3 no nosso app.

## Resumo

Nesta aula aprendemos a utilizar mais um recurso do Firebase. O Firebase Realtime Database. Vimos que com ele podemos definir JSONs para nossa aplicação consumir e não precisamos de um backend personalizado para isto. Uma coisa que não foi mencionada na aula é que o aplicativo tem poder de modificar o JSON no Firebase, então temos uma aplicação totalmente dinâmica e na nuvem. Além disso aprendemos a fazer download de imagens da forma mais eficiente possível utilizando a biblioteca Kingfisher. Finalizamos então o estudo sobre Firebase e Networking.

## Exercícios

## TDP

# Aula 3 - Mídia (Sons)

Finalizando nosso app e consequentemente nosso curso, iremos aprender como executar música no no nosso app, afinal é um app de música. Existem inúmeras maneiras para isto, como stream, baixar no arquivo ou executar em memória. Iremos ensinar como baixar as músicas para uma pasta temporária do seu iPhone via Firebase e então executá-la. Ao final desta aula você poderá incrementar o projeto e deixá-lo mais profissional.

## AVAudioPlayer

Para tocar áudios no nosso aplicativo é mais simples do que você imagina. Nós já criamos as Views e a Apple nos disponibilizou os controllers, o AVAudioPlayer. Basta utilizarmos.

Uma instância da classe AVAudioPlayer, proporciona a reprodução de dados de áudio a partir de um arquivo ou memória.

A Apple recomenda que você use essa classe para reprodução de áudio, a menos que você esteja reproduzindo áudio capturado de um fluxo de rede ou exigindo latência de E/S muito baixa.

Usando um reprodutor de áudio você pode:

* Reproduzir sons de qualquer duração
* Reproduzir sons de arquivos ou em buffer de memória
* Reprodução de músicas em loop
* Reproduzir vários sons simultaneamente, um por leitor de áudio, com sincronização precisa
* Controle do nível relativo de reprodução, posicionamento estéreo e taxa de reprodução para cada som que você está tocando
* Procura um ponto específico em um arquivo de som, que suporta tais recursos de aplicativo como avanço rápido e retrocesso
* Obtém dados que você pode usar para a medição de nível de reprodução

A AVAudioPlayer permite reproduzir som em qualquer formato de áudio disponível no iOS e MacOS. Você implementa um delegate para lidar com interrupções (como uma chamada telefônica recebida no iOS) e para atualizar a interface do usuário quando um som terminar de ser reproduzido. Os métodos do delegate são descritos em AVAudioPlayerDelegate.

Para reproduzir, pausar ou parar um leitor de áudio, ligue para um dos seus métodos de controle de reprodução, descritas em Configurando e Controlar a reprodução .

Essa classe usa o recurso de propriedades declaradas Objective-C para gerenciar informações sobre um som, como o ponto de reprodução dentro da linha de tempo do som e para acessar opções de reprodução, como volume e loop.

### API - Métodos e Propriedades

Esta classe é bem fácil de ser utilizada. Possui uma quantidade razoável de métodos, porém são todos muito úteis, caso queira criar um player de audio completo.

#### 3.1.1.1 Inicializando um objeto AVAudioPlayer

A AVAudioPlayer possui algumas opções de inicialização que podem ser úteis, como inicialização a partir de uma URL remota (ou local), ou a partir de um objeto Data, que é a opção que escolheremos no nosso player.

* init(contentsOf: URL): Inicializa e retorna um player de áudio para reproduzir um arquivo de som.
* init(data: Data): Inicializa e retorna um player de áudio para reproduzir um buffer de memória.
* init(contentsOf: URL, fileTypeHint: String?): Inicializa e retorna um player de áudio usando a URL especificada e a dica ou expectativa do tipo de arquivo.
* init(data: Data, fileTypeHint: String?): Inicializa e retorna um player de áudio usando os dados especificados e dica/expectativa de tipo de arquivo.

#### 3.1.1.2. Configurando e controlando a reprodução

Além da inicialização, temos também controles de reprodução que vão desde os básicos **play, pause, stop** até controles mais específicos como tocar de um certo ponto da música ou controle de loops, volumes e etc.

* func play(): Reproduz um som de forma assíncrona.
* func play(atTime: TimeInterval): Reproduz um som de forma assíncrona, começando em um ponto especificado na linha de tempo do dispositivo de saída de áudio.
* func pause(): Pausa a reprodução; O som permanece pronto para retomar a reprodução de onde parou.
* func stop(): Interrompe a reprodução e desfaz a configuração necessária para a reprodução.
* func prepareToPlay(): Prepara o leitor de áudio para reprodução pré-carregada de seus buffers.
* var isPlaying: Bool: Um valor booleano que indica se o player de áudio está tocando (**true**) ou não (**false**).
* var volume: Float: O volume de reprodução para o player de áudio, variando de 0.0 a 1.0 em uma escala linear.
* var pan: Float: A posição da caixa estéreo do player de áudio.
* var rate: Float: A taxa de reprodução do reprodutor de áudio.
* var enableRate: Bool: Um valor booleano que especifica se o ajuste da taxa de reprodução está ativado para um player de áudio.
* var numberOfLoops: Int: O número de vezes que um som retornará ao início, ao chegar ao final, para repetir a reprodução.
* var delegate: AVAudioPlayerDelegate?: O objeto delegate para o player de áudio.
* var settings: [String : Any]: O dicionário de configurações do player de áudio, contendo informações sobre o som associado ao player.

#### 3.1.1.3. Gerenciamento de informações sobre um som

Podemos também obter os metadados do som, ou seja, as informações adicionais sobre o som.

* var numberOfChannels: Int: O número de canais de áudio no som associado ao leitor de áudio.
* var channelAssignments: [AVAudioSessionChannelDescription]?: Uma matriz de AVAudioSessionChannelDescriptionobjetos associados com o player de áudio
* var duration: TimeInterval: Retorna a duração total, em segundos, do som associado ao player de áudio.
* var currentTime: TimeInterval: O ponto de reprodução, em segundos, dentro da linha de tempo do som associado ao player de áudio.
* var deviceCurrentTime: TimeInterval: O valor de tempo, em segundos, do dispositivo de saída de áudio.
* var url: URL?: URL do som associado ao player de áudio.
* var data: Data?: O objeto de dados contendo o som associado ao player de áudio.

#### 3.1.1.4. Usando medição de nível de áudio

Também existem opções mais avançadas para alcançarmos.

* var isMeteringEnabled: Bool: Um valor booleano que especifica o estado de ativação / desativação da medição de nível de áudio para o reprodutor de áudio.
* func averagePower(forChannel: Int): Retorna a potência média para um determinado canal, em decibéis, para o som que está sendo reproduzido.
* func peakPower(forChannel: Int): Retorna a potência de pico para um determinado canal, em decibéis, para o som que está sendo reproduzido.
* func updateMeters(): Refresca os valores de potência média e de pico para todos os canais de um reprodutor de áudio.
* var format: AVAudioFormat: Obtém propriedades da instância
* func setVolume(Float, fadeDuration: TimeInterval): Ajusta o volume do som com efeito fade (ajuste gradual).

## Prática

Vamos agora aplicar os conhecimentos adquiridos nesta aula e relembrar alguns conhecimentos obtidos ao decorrer do curso de desenvolvimento mobile.

### Padrão de Projeto

Você deve estar cansado de ouvir o termo **Reusabilidade** nestas aulas, mas não há o que fazer. O princípio fundamental de POO é a reusabilidade.

Veja que no nosso projeto **MusicProject** temos dois possíveis lugares destinados a tocar música. O mini player, criado na cena **Navegar** e o player completo localizado na cena **Música**. Tendo isto em vista, temos que desenvolver um componente que manipule o AVAudioPlayer de forma democrática entre estas cenas, ou seja, reusando código de forma genérica.

Para isto iremos criar uma classe chamada **MusicPlayer** que estará desacoplada de qualquer ViewController/Cena e sua instância poderá ser utilizada de forma **compartilhada**.

Repare no comportamento dos demais players de outros aplicativos, como o Spotfy, que também possui um miniplayer. A música é tocada de forma assíncrona como se estivesse de fundo. Uma vez que você a coloca para tocar, você pode continuar navegando no app sem interromper sua execução. Então termos que ter uma **instância única compartilhada** para que não haja instâncias conflitantes tentando executar mais de uma música simultaneamente e para que a MusicPlayer não seja perdida ao destruirmos um View Controller.

O requisito **instância única compartilhada** te lembra de algum padrão de projeto? Sim, teremos que utilizar novamente um **Singleton**.

### Requisitos

Nesta aula colocaremos para funcionar apenas o mini player. Ele é mais prático e objetivo para aprendermos. Além disso na classe MusicViewController existem algumas propriedades que precisarão ser substituídas, pois o AVAudioPlayer já possui propriedades similares. Após terminarmos esta aula você poderá implementar sozinho o player na MusicViewController e completar seu app.

Os requisitos que localizamos logo de cara são:

* **A música não poderá ser interrompida quando sairmos da cena Navegar**: Isto resolveremos com o Singleton.
* As músicas devem ser baixadas utilizando uma URL: Se você retornar na aula 2, verá que no JSON temos a URL da localização das músicas, então será estas que utilizamos.
* **As músicas devem ser executadas preferencialmente sem interrupções:** Não gostamos muito quando nossa música favorita para de tocar quando nossa internet está com conexão lenta, não é? Então para evitar isto devemos optar pela técnica de buffer ao invés de stream nos aparelhos mobile. Esta técnica pode ser alcançada da forma mais simples: baixando a música completamente para o dispositivo e só assim executá-la.
* **Deveremos poder pausar e retomar a música em execução:** Uma vez em execução, deveremos conseguir pausar e retomar a música utilizando o botão de pause/play que inserimos no mini player.
* **O progresso da execução da música deverá ser exibido:** Iremos utilizar a barra cor violeta para mostrar o progresso da música. Utilizaremos animações e restrições para alterar o tamanho da barra conforme o progresso da música.
* **As músicas não devem ser acumuladas em buffer:** Gerenciamento de memória é algo critico no device. Se deixarmos todas as músicas baixadas no device, logo o usuário terá um problema de espaço em disco, e isto não é legal. Por outro lado é um pouco complexo aplicar uma heurística de cache que nos mantenha entre manter apenas uma música em cache ou manter inteligentemente as músicas mais tocadas de modo a aumentar a performance na execução das músicas e um baixo consumo de memória de armazenamento. Optaremos por apenas manter a música em execução no disco do device.

Vamos colocar em prática.

### Criando um Singleton

De todas as formas que vimos nos outros cursos, criar um Singleton no Swift é a mais fácil. Vamos então criar nossa classe Singleton **MusicPlayer**. Vamos continuar do ponto em que paramos na aula passada.

1. Crie um novo arquivo clicando em **File > New File** do tipo **Swift File**:
2. Nomeie-o para **MusicPlayer**, escolha o local de armazenamento e finalize.
3. Você terá apenas um arquivo com a linha import Foundation. Defina a classe **MusicPlayer**

import Foundation

class MusicPlayer {

}

1. Agora vamos criar uma **instância compartilhada** desta classe. Adicione a seguinte propriedade:

static let shared = MusicPlayer()

Pronto. É só disso que precisamos para definir um Singleton no Swift. Sua classe completa estará assim:

import Foundation

class MusicPlayer {

static let shared = MusicPlayer()

}

### Download de músicas utilizando Firebase Storage

Todas as imagens e arquivos mp3 estão armazenados no Storage do Firebase, então temos mais uma facilidade aqui.

Vamos então instalar e utilizar o Storage.

1. Instale o Firebase Storage utilizando o mesmo procedimento utilizando nos demais módulos do Firebase, via Cocoa Pods. Insira a seguinte linha no **Podfile**:

pod 'Firebase/Storage'

1. Instale as dependências. Abra o **Terminal**, navegue até o diretório do seu projeto e execute o comando:

pod install

Agora vamos criar o método de download. Entre na classe **MusicPlayer** e execute os seguinte passos:

1. Importe o Firebase Storage:

import FirebaseStorage

1. Vamos criar uma closure para receber o arquivo mp3. Então a primeira coisa a se fazer é definir o cabeçalho do método. No nosso caso estamos esperando uma URL especificando o caminho local da música. Então adicione as seguinte linhas na sua classe:

import Foundation

class MusicPlayer {

static let shared = MusicPlayer()

}

DICA: Utilizamos a anotação @escaping para podermos omitir o rótulo completion: na chamada do método.

1. Adicione o seguinte código para download:

// Cria um sistema local de referência para o Firebase

let httpsReference = FIRStorage.storage().reference(forURL: url)

// Define um diretório temporário para a música a ser baixada

let tempDirUrl = URL(fileURLWithPath: NSTemporaryDirectory())

// Define um nome para o arquivo no diretório local. Chamará "current.mp3"

let localURL: URL! = tempDirUrl.appendingPathComponent("current").appendingPathExtension(".mp3")

// Chamamos o método write(toFile:)

\_ = httpsReference.write(toFile: localURL) { (URL, error) -> Void in

// Este método tentará baixar o arquivo especificado em `httpReference` no diretório definido em `localURL`

if (error != nil) {

print("Download error: \(error!)")

} else {

// Se a operação for bem sucedida, enviamos a URL local do arquivo para fora da closure

completion(URL!)

}

}

Este método é o suficiente para fazermos download da mídia.

### Manipulando um AVAudioPlayer

O próximo passo é criar uma instância do AVAudio player e criar alguns controles básicos.

1. Importe o AVFoundation:

import AVFoundation

1. Crie as seguintes propriedades:

var currentMusic: Music? // Armazena um objeto do modelo Music da musica sendo tocada

var audioPlayer: AVAudioPlayer? // Nosso player

var timer: Timer! = nil // Timer para calcular o tempo da música

1. Agora insira os seguintes métodos:

func pauseMusic() {

guard let player = audioPlayer else { return }

if player.isPlaying {

player.pause()

timer.invalidate()

}

}

func resumeMusic() {

guard let player = audioPlayer else { return }

if !player.isPlaying {

player.play()

startTimer()

}

}

func stopMusic() {

guard let player = audioPlayer else { return }

if player.isPlaying {

player.stop()

currentMusic = nil

}

}

Nos métodos acima temos ações de Pause, Retomar e Parar a reprodução da música. Agora vamos adicionar os dois métodos mais complexos:

1. Adicione os métodos para Play da música:

func playMusic(music: Music) {

currentMusic = music

stopMusic()

downloadMusic(at: music.mp3URL ?? "") { (localURL) in

do {

self.audioPlayer = try AVAudioPlayer(contentsOf: localURL)

self.audioPlayer?.prepareToPlay()

self.audioPlayer?.play()

self.startTimer()

} catch {

print("Audio error: \(error)")

}

}

}

func startTimer() {

timer = Timer.scheduledTimer(withTimeInterval: 1, repeats: true) {\_ in

guard let player = self.audioPlayer else { return }

}

}

O método play para a reprodução de uma possível música, faz o download da nova música utilizando a closure que criamos e então instancia um AVAudioPlayer com a URL obtida. Ao instanciar, utilizamos o método play() para reproduzir a música.

Criamos também o método startTimer() que inicia um temporizador que de segundo em segundo obterá o tempo corrente da música. Mas não acontece nele ainda. Faremos um delegate para passar este dado para os Controllers.

### Executando músicas

Agora temos o básico para reproduzir músicas no nosso mini player.

1. Entre na classe **BrowseViewController** e em hideMiniPlayer() adicione o método stopMusic da nossa instância compartilhada de MusicPlayer.

func hideMiniPlayer() {

miniPlayerBottomConstraint.constant = self.bottomSpaceWhenHidden

view.layoutIfNeeded()

// Adicione o código abaixo

MusicPlayer.shared.stopMusic()

}

Quando o mini player se fechar a música que estava tocando irá parar.

1. Incremente o conteúdo do método pauseButtonTouchUpInside(\_ :) com o seguinte código:

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

sender.isSelected = !sender.isSelected

// Adicione o códico abaixo

if sender.isSelected {

MusicPlayer.shared.pauseMusic()

}else {

MusicPlayer.shared.resumeMusic()

}

}

Neste método utilizamos os controles pause e resume da nossa instância compartilhada de MusicPlayer. Quando o botão é tocado a música é pausada ou retomada.

1. A última coisa é atualizar o método tableView(\_ :, didSelectRowAt:) com o código abaixo:

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

if indexPath.section == 0 {

showMiniPlayer(withMusic: (browseData?.releases[indexPath.row])!)

// Adicione a linha abaixo

MusicPlayer.shared.playMusic(music: (browseData?.releases[indexPath.row])!)

} else {

showMiniPlayer(withMusic: (browseData?.top10[indexPath.row])!)

// Adicione a linha abaixo

MusicPlayer.shared.playMusic(music: (browseData?.top10[indexPath.row])!)

}

}

Execute o aplicativo e veja a mágica. Ou melhor, ouça.

Tente pausar e retomar a música para testar.

Agora vamos fazer um extra. Atualizar a rosa do mini player para vermos o progresso da música.

### Obtendo o progresso da música

Iremos implementar nosso método startTimer() com a chamada de um delegate. Para isto execute os seguintes passos:

1. Vamos criar um protocolo que defina este delegate. No arquivo **MusicPlayer** declare o seguinte protocolo (fora da classe):

protocol MusicPlayerDelegate: class {

func progressDidChange(currentTime: TimeInterval, duration: TimeInterval)

}

Este delegate será passado para a classe que o conforma, enviando o tempo atual da música e sua duração total.

1. Crie uma propriedade delegate para este protocolo na classe MusicPlayer:

weak var delegate: MusicPlayerDelegate?

1. Por fim, no método startTimer() execute o método deste delegate:

func startTimer() {

timer = Timer.scheduledTimer(withTimeInterval: 1, repeats: true) {\_ in

guard let player = self.audioPlayer else { return }

self.delegate?.progressDidChange(currentTime: player.currentTime, duration: player.duration)

}

}

A classe MusicPlayer está completa.

1. Volte ao BrowserViewController e localize o método viewDidLoad(). Faça o contrado do delegate com esta ViewController (deixe-a em conformidade):

override func viewDidLoad() {

hideMiniPlayer()

MusicPlayer.shared.delegate = self

}

1. Por fim implemente seu método delegate. Utilize um extension para organizar melhor o código:

extension BrowseViewController: MusicPlayerDelegate {

func progressDidChange(currentTime: TimeInterval, duration: TimeInterval) {

updateBars(currentTime: currentTime, duration: duration)

if currentTime == duration {

hideMiniPlayer()

}

}

}

Execute o projeto e veja a barra se atualizar conforme a música

## Considerações finais

O aplicativo mostrado nesta aula, apesar de lhe ter apresentado as principais ferramentas do iOS é um app relativamente simples. Além disso existem outras (e mais sofisticadas) formas de se desenvolver este projeto. É tudo questão do seu ponto de vista e criatividade.

Como dito nos cursos anteriores, as possibilidades são infinitas. O mercado mobile está com força total e agora você está preparado para ele. Seja criativo, destaque-se entre os desenvolvedores e nunca deixe de estudar pois a cada dia que passa temos uma tecnologia nova.

Nossa jornada se encerra aqui. Esperamos que você tenha aproveitado e aprendido bastante, pois aprendemos muito com você. Agradecemos pelo seu esforço e dedicação.

Desejamos um bom sucesso a você!