# Livro do Educador

## Desenvolvimento de Aplicativos Móveis

### iOS

# Unidade 7

## Aula 1 – Toques Finais

Bem-vindos a última unidade de nosso curso, isso é motivo de parabenizar você e o aluno, agora ele concluirá a primeira etapa da carreira de programador. Até agora aprendemos o essencial para estruturar o aplicativo de música. 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.

### CARGA HORÁRIA

Conforme o plano de aula, esta aula terá duração de 1h30 e deverá ser conduzida de acordo com as orientações pedagógicas.

### OBJETIVO DA AULA

Ao final da aula, você deverá garantir que o aluno tenha subsídios para

* Criar via código, uma view;
* Ter noção de criação de formas simples;
* Fazer uso de utilitários para que a programação via código seja mais produtiva.

### ORIENTAÇÕES PEDAGÓGICAS

Para atender os objetivos de aprendizagem, você deverá conduzir o processo de ensino considerando a organização didática apresentada a seguir:

* 40 minutos de aula expositiva;
* 10 minutos para tirar as dúvidas dos alunos;
* 40 minutos para desenvolver as atividades propostas para a turma e apresentação da TDP.

### TÓPICOS DE ESTUDO

Todos os tópicos a seguir, conforme livro do aluno, devem ser trabalhados de forma dinâmica, criativa, com embasamento teórico e prático voltado ao mercado de trabalho.

* Melhorando o Interface Builder;
* Melhorando nosso player de música;
* Firebase Authentication;
* Criando um Projeto Firebase;

### PONTOS IMPORTANTES

Nesta aula o aluno perceberá, por definidivo, como é mais fácil desenvolver Views para iOS do que para Android, através do Interface Builder. O aluno aprenderá a reutilizar as interfaces das views apenas utilizando recursos de herança. Ainda irá adicionar o botão cuja função é sobrepor a View da música sendo tocada sobre a View de Navegação.

### Melhorando o Interface Builder

Educador, neste item o aluno aprenderá a utilizar recursos do Interface Builder para evitar ao máximo a manipulação de códigos e tornar o trabalho mais produtivo. É muito simples a maneira de trabalhar com o Interface Builder, mas muitas vezes precisaremos utilizar views idênticas em storyboards diferentes e para evitar o retrabalho de criação de uma mesma view em cada storyboard que ela for utilizada, poderemos reutilizar estas views com seus recursos de **herança**.

#### Reutilizando Views

Educador, para utilizar os recursos de herança na reutilização de Views, explique ao aluno que é necessário criarmos uma classe personalizada para elas para que ela seja “subclasseada”, mas isso deveríamos fazer via código. O aluno irá se perguntar o por quê de estarmos codificando nossas views se acabamos de dizer que este é um método muito trabalhoso. Pois bem, ao codificarmos nossas views, podemos reutilizá-las em outras circunstancias, garantindo que esta apresente um comportamento esperado em qualquer situação em que for utilizada.

As desvantagens de manipular estes códigos são claras e ainda só poderíamos ver o após a compilação. Ainda sim, você pode convencer o aluno que, mesmo tendo que implementar o código de nossas views, poderíamos economizar bastante tempo graças ao recurso de **renderização em tempo real** que o Interface Builder nos oferece, mostrando assim, em tempo real, nossas views no momento imediato em que estamos digitando nosso código, sem necessitar de compilação.

#### Renderizando Views no Interface Builder

Educador, reforce um pouco mais sobre a renderização em tempo real e mostre que há trê técnicas disponíveis para isso: o Draw Rect, o IBDesignable e o IBInspectable. Em seguida, iremos explicar cada um deles à parte.

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

#### O drawRect()

Com ele, 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.

O método poderia ser definido dessa forma:

override func drawRect(rect: CGRect) {

}

Dentro dele podemos definir formas através de nossos conhecimentos sobre geometria, a pesar de não parecer, o framework CoreGraphics torna essa tarefa bem fácil, mas isto não será nosso foco principal.

Educador, no exemplo dado temos um botão “Adicionar” da Figura 1, onde podemos exercitar com o aluno, antes de mais nada, como poderiamos desmembrar este botão nas formais mais simples possíveis para implementar isso em código logo após. A maneira mais fácil de representar o botão é por um círculo com preenchimento na cor verde e dois retângulos que se interceptam em seus centros geométricos e estão posicionados no centro geométrico do círculo.

Agora o aluno pode transpor essa ideia para o código e fazer acontecer. O código abaixo é uma maneira inteligente e simples de representar esse botão. Não se preocupe em demonstrá-lo agora via compilador. Apenas explique o texto de código escrito no livro do aluno.

No código, após abrirmos o método draw(\_ rect: CGRect) a primeira coisa que implementamos é um “caminho” oval dentro de um retângulo (ovalIn: rect). Geometricamente, teríamos que criar um caminho inscrito (no caso de um círculo, significa que ele tangencia as 4 paredes internas de um retângulo) em um retângulo de lados iguais (um quadrado).

Primeiro preenchemos a forma oval com uma cor e em seguida implementamos os limites (bounds) da largura e altura da linha horizontal do “+”, e definimos que a largura do caminho será igual à altura da linha horizontal.

Para definir um ponto inicial para o nosso caminho sem desenhar um seguimento, usamos o método move(to: ) com as coordenadas x e y descritas no exemplo, e para definirmos o ponto final do seguimento horizontal utilizamos o método addLine(to: ) com as coordenadas no exemplo. Fazemos o mesmo com a linha vertical. Por fim definimos as cores das linhas e aplicamos o desenho das linhas que definimos com o método stroke(). O resultado deverá ser como na Figura 2 e poderá ser renderizado na hora, no IB.

#### IBDesignables

Através deste método podemos fazer com que uma view se torne Designable, o que significa que ela será desenhável pelo interface builder. Isto faz com que o código contido no método draw(\_ :) seja mostrado no interface builder.

Educador, aqui explique ao aluno que para que a nossa view seja atualizada imediatamente no IB, conforme alteramos o método draw(\_ :), devemos utilizar o atributo @IBDesignable no inicio da classe, como foi apresentado no exemplo do método anterior.

#### IBInspectables

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.

Esta facilidade é possível, adicionando a anotação @IBInspectable na propriedade que queremos alterar via IB, como por exemplo a propriedade fillColor, onde poderemos controlar a cor de preenchimento de nossas formas através do IB.

### Melhorando nosso player de música

Educador, visto a teoria, vamos agora praticá-la criando os ícones que representarão o botão para esconder e mostrar a view no mini-player na frente das nossas storyboards. Serão criados ícones como mostrados nas Figuras 4 e 5, uma seta para cima e outra para baixo.

Educador, mostre o escopo do projeto, com as Figuras 6 e 7, explicando que a seta para cima terá a função de puxar para cima a View de música sendo tocada, e a seta para baixo é para puxar para baixo a view de música sendo tocada.

Por fim, abra o arquivo **Unidade\_7\_-Aula*\_*1-\_Exemplo\_1.zip** e daremos início ao nosso passo-a-passo.

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

Educador, instrua o aluno a:

* Selecionar o group **Views** que já foi criado previamente e criar um novo arquivo pelo menu **File > New > File**, selecionando a opção **View** na janela que se abriu e, clicando em **Next**, selecione a opção **ExpandCollapseButton** para a view. Finalizar a criação clicando em **Create**.
* Um novo arquivo .xib foi criado, onde deverá ser feito alguns ajustes. Para isso, selecionar a **View** no painel de navegação e na aba **Attributes inspector**, escolher a opção **Freedom** na propriedade **Size**. Em seguida, definir a largura e altura da view para **200x100**, na aba **Size Inspector.**

#### Criando uma subclasse de UIButton

Temos que criar uma subclasse para reger o .xib criado, que originalmente é bastante limitado. Instrua o aluno a:

Selecionar o group **Views** e criar um novo arquivo, como feito nos passos anteriores. Escolher a opção **Cocoa Touch Class,** e ao clicar em **Next**, preencha o campo **Class** com o nome **ExpandCollapseButton** e o campo **Subclass of** com a opção UIButton. Clicando em **Next**, o resultado será semelhante ao apresentado nos passos 5, 6 e 7.

Para vincular nossos dois objetos criados, selecionar a View no arquivo **ExpandCollapseButton.xib** e escrever **ExpandCollapseButton** no campo **Class** que se encontra na aba **Identity inspector.** Isto fará com que a view selecionada obedeça a classe **ExpandCollapseButton** definida no arquivo **ExpandCollapseButton.swift.**.

#### Desenhando no botão

Para desenhar a seta, abra o arquivo **ExpandCollapseButton.swift** e siga os seguintes passos:

* Criar a propriedade que definirá o sentido da seta, como no passo 1 e a propriedade de cor, como no passo 2, na classe que foi aberta. A classe deverá estar como mostrada no passo 3.
* Inserir o código, contido no passo 4, dentro do método **drawRect(\_ :)**. Neste código, há bastante conceitos matemáticos de localização em coordenadas geométricas, instigue ao aluno procurar entender como estas linhas são desenhadas teoricamente, em um plano cartesiano, visitando também a documentação da Apple.
* Verificar se sua classe está como a mostrada no passo 5. Ainda não podemos ver a seta instantaneamente, então a próxima etapa é adicionar o atributo IBDesignable.

#### Exibindo atualizações em tempo real

Educador, agora iremos tornar possível a visualização da view em tempo real. Siga os passos.

* Adicionar a anotação **@IBDesignable** no início da definição da classe **ExpandCollapseButton** como no passo 1, e em seguida abrir o arquivo **ExpandCollapseButton.xib** e selecionar a view **Expand Collapse Button**. Agora é possível visualizar na aba **Identity inspector** um estado dos Designables, quando o estado for Up to date será possível visualizar o desenho no xib.

#### Tornando propriedades Inspectables

Educador, iremos agora adicionar propriedades no xib utilizando o IBInspectable. Para isso, realizaremos algumas mudanças no código, como no passo-a-passo a seguir:

* Na classe **ExpandCollapseButton**, modificar a propriedade de sentido e cor da seta que criamos anteriormente adicionando o atributo **@IBInspectable**, antes da declaração de cada propriedade, como mostrado no passo 1.
* Realizar as modificações como mostrado no passo 2 para garantir que o visual irá se atualizar quando modificarmos os Inspectables. Explique como o método setNeedsDisplay() é capaz de forçar essa atualização e em seguida verifique se a classe **ExpandCollapseButton**  está como o exemplo mostrado no passo 3.
* Abrir o arquivo **ExpandCollapseButton.xib**, selecionar a view **Expand Collapse Button** e verificarna aba **Attribute inspector**, que agora aparecem as duas propriedades que transformamos em Inspectable, sendo assim possível modifica-la a seu gosto.

#### Utilizando nosso botão

Educador, agora iremos incluir nosso botão criado ao nosso mini-player. Siga os passos.

* Abrir a Main.storyboard e selecionar o botão **Expand Button**, na cena **Navegar**. Em seguida, definir no campo Class o nome ExpandCollapseButton, na aba Identity inspector e verificar que a seta apareceu no lugar do botão selecionado.
* Ainda em Main.storyboard, localizar o botão que nomeamos **Collapse Button**  na cena **Música** e **n**a aba Identity inspector, definir no campo Class o nome ExpandCollapseButton, selecionar **On** na propriedade **Is Pointing To Down** e escolher a cor branca na propriedade **Arrow Color**. Verificar o resultado da view. Deve ser parecido com o resultado mostrado no passo 8.

Assim finalizamos a primeira aula desta Unidade.

### Resumo

Nesta aula, o aluno aprendeu a implementar uma view através de código para que seja possível ele reutilizá-la e ter controle de seu comportamento. Descobriu que a implementação por código no Xcode possui suas facilidades com os atributos @IBDesignable, que lhe permite mostrar em tempo real o desenho que se está sendo implementado via código, e também o atributo @IBInspectable, que permite que as propriedades associadas À ele sejam controladas via Interface Buider.

### 1.4. Exercícios

### 1.5. TDP

## Aula 2

### 2. Manipulando dados obtidos pela rede

Nesta aula iremos rever o Firebase Database, utilizá-lo juntamente com a estrutura JSON para obter informações de músicas da web tela Navegar de nosso app e também atribuir imagens da web como capa de nossas músicas.

### CARGA HORÁRIA

Conforme o plano de aula, esta aula terá duração de 1h30 e deverá ser conduzida de acordo com as orientações pedagógicas.

### OBJETIVO DA AULA

Ao final da aula, você deverá garantir que o aluno tenha subsídios para

* Inserir um JSON no Firebase Database;
* Instalar e utilizar um JSON em nossa aplicação situado num Database criado;
* Instalar e utilizar imagens a partir da web como capa de nossas músicas;
* Concluir um visual completo para um app player de música ou semelhantes.

### ORIENTAÇÕES PEDAGÓGICAS

Para atender os objetivos de aprendizagem, você deverá conduzir o processo de ensino considerando a organização didática apresentada a seguir:

* 40 minutos de aula expositiva;
* 10 minutos para tirar as dúvidas dos alunos;
* 40 minutos para desenvolver as atividades propostas para a turma e apresentação da TDP.

### TÓPICOS DE ESTUDO

Todos os tópicos a seguir, conforme livro do aluno, devem ser trabalhados de forma dinâmica, criativa, com embasamento teórico e prático voltado ao mercado de trabalho.

* Firebase Database;
* Estrutura do JSON;
* Adicionar um JSON ao Firebase;
* Consumindo o serviço;
* Exibindo imagens com a Kingfisher.

### PONTOS IMPORTANTES

Esta é a última que trataremos da questão visual de nosso app. Se houver tempo, podemos personalizar nosso app e torna-lo mais bonito, definitivamente.

### 2.1. Firebase Database

Educador, relembre os conceitos de Firebase Database, a estrutura em que esses dados são armazenados na núvem (JSON) e também a forma como eles são sincronizados em tempo real com todos os clientes conectados, ideal 0para aplicativos multiplataforma.

Para obtermos músicas da tela Navegar através do Firebase Database, teremos que compreender os conceitos seguintes.

### 2.2. Estrutura do JSON

Educador, iremos utilizar um JSON para que nossa lista de músicas apresente informações sobre as mesmas assim como fizemos na nossa lista de e-mails em Android.

No exemplo listado temos um JSON, com informações de nome da música, artista, imagem de capa e o arquivo de música para a nossa tela Navegar. Estes dois últimos podem ser requisitados de nosso Firebase. Caso queira adicionar informações para outras abas, basta adicionar objetos no mesmo nível que “navegar”. Dentro de “navegar” ainda temos mais dois objetos “top\_10” e “lançamentos”, para que os objetos que estejam dentro destes sejam catalogados em seções do mesmo nome na nossa tela Navegar.

Os arquivos de música e imagem podem ser armazenados no Storage do Firebase tanto manualmente, através do Firebase console, como pelo usuário, via app.

### 2.3. Adicionando um JSON ao Firebase

Tendo os conceitos de JSON em mente, podemos agora utilizar este formato para adicionar um conteúdo no Firebase, para que possamos consumi-lo posteriormente. Faça o seguinte:

* Abrir o Firebase Console pelo endereço: <https://console.firebase.google.com/?hl=pt-BR> , entrar no projeto MusicProject e em seguida, acessar o menu Database. Uma nova tela será apresentada.
* Em uma nova janela do seu navegador, colar o link:
* <https://firebasestorage.googleapis.com/v0/b/musicproject-63abd.appspot.com/o/conteudo.json?alt=media&token=69bb4fd4-6529-46bc-bb2d-a821ba8048aa> e o arquivo conteúdo.json será baixado em seu computador.
* Retornando na janela em que foi aberto o Firebase Databe, clicar no munu Overflow no canto superior direito e clicar no botão Importar o JSON. Clicar em PROCURAR e adicionar o arquivo conteúdo.json recentemente baixado. Clicar em IMPORTAR e o JSON será adicionado ao Firebase Database, e poderá ser expandido para que a estrutura deste possa ser visualizada.

### 2.4. Consumindo o serviço

Educador, agora o aluno deverá abrir o arquivo **Unidade\_7\_-Aula\_2-\_Exemplo\_1.zip** onde a configuração do Firebase foi aprimorada para que possamos dar continuidade.

#### 2.4.1. Instanlando o Firebase Realtime Database

Educador, agora você ensinará o aluno a instalar o Database que acabamos de criar através do CocoaPods, processo semelhante às outras bibliotecas que instalamos. Siga os passos.

* Abrir o arquivo Podfile e inserir a linha de código como no passo 2 e em seguida verificar se o Podfile se assemelha com o apresentado no passo 3.
* Agora, com o terminal do Mac aberto, é preciso navegar até o diretório do seu projeto, que você acabou de obter do material de apoio e executar o comando **pod** install. Assim estará instalado o Database.

#### 2.4.2. Utilizando o Firebase Database

Agora, você mostrará ao aluno como utilizar o Database instalado, seguindo os seguintes passos.

* Entrar no arquivo BrowseViewController e importar o Firebase Database, digitando o código, que se encontra no passo 2, no topo da classe.
* Em seguida localize o método viewWillAppear(\_ :) e insira o código como no passo 4. Assim o JSON será chamado e atualizado a cada vez que a tela for exibida.
* Agora, é importante que o código seja explicado, no passo 5 temos uma explicação básica.
* Verifique o arquivo Models.swift e veja que esta camada model converte o JSON para o objeto Browse e subobjetos Music.

### 2.5. Exibindo imagens com a Kingfisher

Neste ítem, o aluno aprenderá a baixar imagens especificadas por URLs no JSON, através da biblioteca Kingfisher.

#### 2.5.1. Kingfisher

Educador, defina ao aluno a biblioteca Kingfisher como sendo uma biblioteca que possibilita o download do cache de imagens da web em nosso app. Esse cache pode ser armazenado em disco ou em memória e é possível escolher apenas um sistema de download ou somente de cache, sendo estes canceláveis, fazendo com que haja economia no consumo de dados, entre outras características.

Com apenas as linhas de código abaixo é possível exibir facilmente imagens da web, utilizando o Kingfisher e com extension da UIImageView.

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.

#### 2.5.2. Instalando o Kingfisher

Nesta etapa iremos instalar o Kingfisher via CocoaPods, como fizemos nas outras bibliotecas que instalamos

* Abrir o arquivo Podfile e inserir a linha de código como no passo 2 e em seguida verificar se o Podfile se assemelha com o apresentado no passo 3.
* Agora, com o terminal do Mac aberto, é preciso navegar até o diretório do seu projeto, que você acabou de obter do material de apoio e executar o comando **pod** install. Assim estará instalado a biblioteca Kingfisher.

#### 2.5.3. Utilizando o Kingfisher

Para utilizá-la, execute os seguintes passos:

* Entrar o arquivo BrowseViewController e no topo da classe, importar a Kingfisher como no passo 2. Em seguida, localizar o o método tableView(\_ :, cellForRowAt:) e inserir as linhas de comando como mostrado no passo 4.
* Faça com que o aluno note que foi necessário apenas inserir 2 linhas para configurar uma imagem. A linha responsável por instanciar uma URL e outra para definir a imagem da ImageView a partir da URL. Veja o resultado final.

Assim, finaliza-se a parte visual de nosso app. Na próxima aula será apresentado um modo para que nosso aplicativo toque um som em .mp3.

### 2.6. Resumo

Nesta aula o aluno aprimorou seus conhecimentos sobre o recurso Firebase Database e o utilizou para definir um JSON para ser apresentadas as informações das músicas em nossa tela Navegar, e ainda o aluno aprendeu a obter imagens apartir de uma URL com a biblioteca Kingfisher.

### 2.7. Exercícios

### 2.8. TDP

## Aula 3

### 3. Mídia (Sons)

Aula final de nosso curso básico, finalmente faremos com que nosso player de música toque músicas, nesta aula utilizaremos o AVAudioPlayer, que é um Controller simples disponibilizado pela Apple e é uma das formas mais simples das inúmeras formas de se tocar música em nosso app.

### CARGA HORÁRIA

Conforme o plano de aula, esta aula terá duração de 1h30 e deverá ser conduzida de acordo com as orientações pedagógicas.

### OBJETIVO DA AULA

Ao final da aula, você deverá garantir que o aluno tenha subsídios para

* Buscar cada funcionalidade do AVAudioPlayer;
* Atribuir controllers, delegates e protocolos para que possamos tocar músicas;
* Obter músicas do Firebase Storage;
* Atribuir músicas baixadas à um closure;
* Dar função pause e resume ao botão Play;
* Atualizar em tempo real a barra de progresso de tempo de reprodução da música.

### ORIENTAÇÕES PEDAGÓGICAS

Para atender os objetivos de aprendizagem, você deverá conduzir o processo de ensino considerando a organização didática apresentada a seguir:

* 40 minutos de aula expositiva;
* 10 minutos para tirar as dúvidas dos alunos;
* 40 minutos para desenvolver as atividades propostas para a turma e apresentação da TDP.

### TÓPICOS DE ESTUDO

Todos os tópicos a seguir, conforme livro do aluno, devem ser trabalhados de forma dinâmica, criativa, com embasamento teórico e prático voltado ao mercado de trabalho.

* AVAudioPlayer;
* Prática;
* Considerações finais.

### PONTOS IMPORTANTES

A compreensão desta aula é essencial para o funcionamento do nosso aplicativo, pois agora o faremos finalmente tocar.

### 3.1. AVAudioPlayer

O AVAudioPlayer é um controller disponibilizado e recomendado pela Apple capaz de satisfazer a nossa necessidade, a reprodução de dados de áudio a partir de um arquivo ou memória.

Educador, você poderá descrever alguns dos recursos que obtemos ao utilizar o AVAudioPlayer , e para que serve os delegates implementados para realizar interrupções e atualização da tela quando a música terminar.

Em seguida serão apresentados alguns dos recursos, apresentados na documentação da Apple para utilização de algumas funções de reprodução.

#### 3.1.1. API - Métodos e Propriedades

Educador, na documentação da Apple temos diversos métodos e propriedades que podemos utilizar nessa classe de reprodução de áudio. A seguir, dê uma breve explicação sobre cada grupo de funções, mas discorra sobre cada método e propriedade de maneira individual, fazendo uso do texto do livro do aluno.

**3.1.1.1 Inicializando um objeto AVAudioPlayer**

São opções de inicialização que podem ser úteis, como inicialização de um player a partir de uma URL remota ou de uma memória local, um arquivo entre outros tipos de inicialização.

**3.1.1.2. Configurando e controlando a reprodução**

Esse conjunto de métodos são responsáveis por comandos básicos como função play de algum ponto específico de tempo ou não, pause, stop, retornar um valor se o player está tocando (essencial para definir o estado do botão Play), volume, balanço, delegate, e diversas outras opções de controle.

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

Neste grupo podemos obter informações como o tempo total de duração do áudio, o tempo de reprodução atual, os dados contendo o som associado o player, entre outros.

**3.1.1.4. Usando medição de nível de áudio**

Este grupo, são opções mais avançadas de medição de nível de áudio, potência, picos de determinadas frequência, efeito de fade in e fade out, e outros.Também existem opções mais avançadas para alcançarmos.

### 3.2. Prática

Dado as devidas explicações, entraremos na parte prática, aplicando alguns conceitos ministrados no curso.

#### 3.2.1. Padrão de Projeto

Com todo este conceito de reusabilidade aprendido em sala, a questão agora é fazer com que um componente manipule o AVAudioPlayer de forma democrática entre as duas cenas que tocará música, a cena Música, e o mini-player na cena Navegar, reusando o código de maneira 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**.

O objtivo aqui é permitir que a música seja tocada de maneira assíncrona, sem que ela seja interrompida durante a navegação pelo app, para isso deve-se criar uma instancia única compartilhada, para que não haja outras instancias tentando ser executadas simultaneamente, causando conflitos e para que a classe MusicPlayer não seja perdida ao destruirmos um View Controller.

O **Singleton** é o padrão perfeito para isso.

#### 3.2.2. Requisitos

A principio iremos fazer tocar apenas o mini-player, mas precisamos listar alguns requisitos para que tenhamos êxito. Apresente tais requisitos aos alunos, segundo o mostrado no livro do aluno, exercitando a capacidade do aluno identificar possíveis erros, ou complicações, previamente para que ele possa ir diretamente ao ponto para resolvê-los, logo de cara.

#### 3.2.3. Criando um Singleton

A partir do nosso arquivo da aula passada vamos criar nossa classe Singleton **MusicPlayer**, com os seguintes passos

* Criar um novo arquivo do tipo **Swift File** e nomeá-lo para **MusicPlayer,** escolhendo seu local de armazenamento e finalizando a criação. Assim teremos um arquivo com uma linha e então temos que definir a classe **MusicPlayer**, como mostrado no passo 3.
* Adicionar a propriedade no passo 4 para criarmos uma **instância compartilhada** e assim teremos nosso **Singleton** definido.

#### 3.2.4. Download de músicas utilizando Firebase Storage

Precisamos agora instalar e utilizar nosso Storage do Firebase para obtermos todos os nossos arquivos de música que estarão nele armazenado. Podemos fazê-los da seguinte forma:

* Instalar o **Firebase Storage** utilizando o mesmo procedimento utilizando nos demais módulos do **Firebase**, via **Cocoa Pods** e inserir no **Podfile**, uma linha como mostrado no passo 1. Por fim, instale as dependências através do **terminal** do Mac.
* Para criar um método de download, é necessário entrar na classe **MusicPlayer** e importar o **Firebase Storage** da forma mostrada no passo 1, em seguida, crie uma **closure** para receber o arquivo mp3, utilizando o cabeçalho apresentado no passo 2. Para finalizar, adicione o código como no passo 3, que será o código responsável para download e de as explicações do código como mostra os seus comentários.

#### 3.2.5. Manipulando um AVAudioPlayer

Para criar uma instância do AVAudioPlayer e criar alguns controles básicos, faça o seguinte:.

* Importe o **AVFoundation** como o indicado no passo 1 e crie as propriedades mostradas no passo 2. Insira alguns métodos mostrados no passo 3, estes serão responsáveis por algumas ações básicas de controle.
* Para inserir o método, **play**() e o método **startTime**(), cole os códigos do passo 4 e explique como eles funcionam.

#### 3.2.6. Executando músicas

Agora faremos com que as músicas possam ser executadas. Basta seguir os próximos passos:

* Na classe **BrowseViewController** adicionar o método stopMusic da nossa instância compartilhada de MusicPlayer, dentro de hideMiniPlayer(), ficará como no passo 1. Em seguida, é necessário incrementar o conteúdo do método pauseButtonTouchUpInside(\_ :) mostrado no passo 2, para que possamos utilizar os controles de pause e resume da nossa instancia compartilhada, pressionando o botão.
* Por fim, precisamos atualizar o método tableView(\_ :, didSelectRowAt:), inserindo o código mostrado no passo 3 e executar o aplicativo, que já poderá tocar música. Teste os botões de pause e resume.

#### 3.2.7. Obtendo o progresso da música

Agora podemos atualizar o tempo de reprodução de nossa música, chamando um delegate. Para isto execute os seguintes passos:

* No arquivo **MusicPlayer,** declarar o protocolo (fora da classe) mostrado no passo 1, para definir o delegate, que enviará o tempo atual da música e sua duração total para a classe que o conforma.
* Ainda na classe MusicPlayer, criar uma propriedade delegate para este protocolo, como mostrada no passo 2 e no método startTimer() , executar o método deste delegate como mostra o passo 3.
* No método BrowserViewController, localize o método viewDidLoad() e deixe a view em conformidade com o delegate como mostra o passo 4. Para finalizar, utilize um extension, mostrado no passo 5 para melhorar a organização do método delegate.

Assim se finda nosso projeto.

## Considerações finais

Como consideração final, diga ao aluno que ele pode aperfeiçoar ainda mais o app. Garanta a ele, que apesar de termos feito uma aplicação simples, acabamos por utilizar as principais ferramentas do iOS e que também há outras formas do aluno realizar este mesmo projeto de aplicativo.

Sendo assim, o aluno agora possui subsídio básico para programar aplicações para aparelhos mobile, e é altamente recomendável, que a partir deste final, o aluno continue buscando crescer ainda mais no campo de aprendizagem, pois a tecnologia está sempre em evolução, e com certeza, quanto mais o aluno buscar por conhecimento, maior será o reconhecimento dele no mercado de trabalho.

Sucesso à você, Educador, e nos veremos novamente, com a próxima turma.

## Bye Bye World