APP WORLD

INTEGRANDO A SUA APLICAÇÃO

8B

LISTA DE FIGURAS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

1 INTEGRANDO A SUA APLICAÇÃO

Com os conteúdos de gravação de dados, agora precisamos deixar o seu projeto ainda mais funcional, por isso vamos criar acesso a dados que estão hospedados em servidores externos. Além disso, você vai aprender a criar notificações. Dessa forma, seu app estará mais integrado e inserido na realidade das aplicações disponíveis no mercado.

Os principais aplicativos usados hoje em dia fazem uso de recursos em nuvem, sejam serviços, banco de dados etc. Facebook, WhatsApp, Twitter e Instagram são exemplos de apps que não existiriam sem webservices. As aplicações dinâmicas permitem que seu app converse com outros sistemas por meio dos webservices. Vamos integrar nossa aplicação?

1.1 Consumindo APIs com URLSession

Um dos recursos mais utilizados por developers no desenvolvimento de aplicações iOS é o uso de APIs. Consumir serviços efetuando requisições HTTP é essencial em muitos modelos de negócios, que precisam fornecer ao usuário dados dinâmicos e que são atualizados constantemente ou até mesmo atualizados pelos próprios usuários.

1.1.1 URLSession

Quando precisamos consumir uma API REST, fazemos uso da classe **URLSession**. Ela é utilizada juntamente com um conjunto de classes relacionadas, que são **URLSessionConfiguration**, **URLSessionDataTask**, entre outras.

É possível utilizar várias sessões (URLSession) na mesma aplicação. Por exemplo, usar uma sessão com prioridade alta para processos mais críticos, e outra com prioridade baixa para os demais processos.

As principais vantagens de se utilizar o URLSession são:

- Requisições assíncronas.

- Uploads e downloads em background.
- Poder pausar e continuar qualquer tarefa de rede.
- Compartilhar configurações com outras sessões.
- Armazenamento privado por sessão.

A classe URLSession nos fornece um singleton que contém uma sessão compartilhada, configurada com valores padrão, pronta para uso. Esse singleton pode ser obtido por meio de **URLSession.shared** e possui um objeto de configuração (URLSessionConfiguration). Seu uso é aconselhado quando formos fazer operações simples e que não necessitem de configurações mais específicas como, por exemplo, efetuar downloads enquanto o app está em repouso. Em resumo, se seu app precisa alterar configurações de cache, cookies, autenticação, etc., não deve usar o singleton shared.

É possível fazer uso de delegates ao utilizar URLSession, ou podemos utilizar tarefas com seus respectivos **completion handlers**, que nos devolvam todas as informações da requisição efetuada.

1.1.2 URLSessionConfiguration

Utilizamos esta classe para criar um objeto de configuração para a nossa sessão. Esse objeto irá definir todo o comportamento e as regras utilizadas nessa sessão.

Ao criarmos um objeto **URLSessionConfiguration**, é imprescindível que o configuremos antes de atribuí-lo a uma sessão. No momento da atribuição, é feita uma cópia desse objeto e, depois disso, quaisquer modificações não são aplicadas à sessão. Caso seja necessário realizar alterações nesse objeto, deve-se criar uma nova sessão e utilizar nela o objeto modificado.

Durante a criação, devemos escolher que tipo de configuração iremos adotar. Os tipos possíveis são:

- **default**: Retorna uma configuração com os valores mais padronizados (cache global em disco, cookies, credenciais etc.).

- **ephemeral**: Não armazena cache, cookies e credenciais no disco, somente em memória. Assemelha-se ao “modo privado” dos navegadores.
- **background**: Usado quando precisamos realizar tarefas e o app está em background. Ao utilizá-lo, é possível recuperar downloads em andamento ou concluir uma tarefa mesmo que o aplicativo entre em repouso.

É possível configurar diversos aspectos da nossa sessão por meio do URLSessionConfiguration. Dentre as possibilidades, destacam-se:

- **allowsCellularAccess**: Definimos se será permitido ou não o uso de conexões 3G, 4G etc., para efetuar as requisições nessa sessão (true ou false).
- **httpAdditionalHeaders**: Neste dicionário, configuramos os Headers que serão utilizados em todas as requisições realizadas nessa sessão, por exemplo, definir o UserAgent ou o Content-Type.
- **timeoutIntervalForRequest**: Aqui, podemos definir um tempo que a requisição irá aguardar uma resposta. O valor padrão é 60 segundos.
- **httpMaximumConnectionsPerHost**: Por meio desse parâmetro, podemos limitar a quantidade de conexões feitas no nosso host pelas tasks dessa sessão, lembrando que esse valor é definido por sessão. O padrão é 4.

1.1.3 URLSessionDataTask

A classe **URLSessionDataTask** é responsável pela execução da requisição à nossa API. Ela retorna os dados recebidos diretamente em memória.

Para criarmos uma dataTask, utilizamos algum dos diversos métodos presentes em URLSession. Temos opções para criar uma task passando apenas a URL que irá chamar (que é o que iremos usar no curso), bem como podemos passar a URL, o objeto de requisição (**URLRequest**) e definir a closure (**completionHandler**) que será executada ao término da requisição. Nesta closure, teremos acesso aos dados fornecidos pela API (por meio de um objeto do tipo Data), bem como o objeto de resposta (**URLResponse**) do servidor com os metadados da

requisição (tais como cabeçalho HTTP, status code, etc.). Também é possível obter essas informações fazendo uso de delegates.

Para que uma tarefa seja executada, é necessário chamar o método **.resume().**

Vamos, agora, colocar em prática o consumo de APIs REST em nosso aplicativo. A ideia é tentar recuperar o trailer do filme fazendo uso da API do iTunes. Essa API nos fornece diversas informações sobre um filme pesquisado e, dentre elas, o trailer do filme, que é do que iremos precisar para o nosso app.

Abra o app MyMovies, e crie um novo Swift file com o nome **API.swift**. Nesse arquivo, vamos implementar uma classe chamada API, que será responsável por fazer a requisição à API do iTunes, pesquisando as informações do filme selecionado.

O Código-fonte "Exemplo de retorno da API do iTunes" mostra o exemplo do retorno que essa API nos traz quando, por exemplo, pesquisamos o filme **Vingadores: Guerra Infinita**. A url da API é **https://itunes.apple.com/search?media=movie&entity=movie&term=,** em que será inserido no final o nome do filme que iremos pesquisar, lembrando que precisaremos formatar este nome para que possa ser usado em uma URL (tirando espaços e acentos, por exemplo).

```
{
    "resultCount": 1,
    "results": [{
                "wrapperType": "track",
                "kind": "feature-movie",
                "trackId": 1370224078,
                "artistName": "Anthony Russo & Joe Russo",
                "trackName": "Avengers: Infinity War",
                "trackCensoredName": "Avengers: Infinity War",
                "trackViewUrl":
"https://itunes.apple.com/us/movie/avengers-infinity-
war/id1370224078?uo=4",
                "previewUrl":
"http://video.itunes.apple.com/apple-assets-us-std-
000001/Video118/v4/fe/70/73/fe707321-a30b-b7a9-83fb-
173e6fcde4fa/mzvf_6871512460174111986.640x354.h264lc.U.p.m4v",
                "artworkUrl30": "https://is3-
ssl.mzstatic.com/image/thumb/Video118/v4/fd/0f/19/fd0f19ae-
7db9-29e5-5da5-c5574d397b07/source/30x30bb.jpg",
                "artworkUrl60": "https://is3-
```

```
ssl.mzstatic.com/image/thumb/Video118/v4/fd/0f/19/fd0f19ae-
7db9-29e5-5da5-c5574d397b07/source/60x60bb.jpg",
                "artworkUrl100": "https://is3-
ssl.mzstatic.com/image/thumb/Video118/v4/fd/0f/19/fd0f19ae-
7db9-29e5-5da5-c5574d397b07/source/100x100bb.jpg",
                "releaseDate": "2018-04-27T07:00:00Z",
                "collectionExplicitness": "notExplicit",
                "trackExplicitness": "notExplicit",
                "trackTimeMillis": 8981638,
                "country": "USA",
                "currency": "USD",
                "primaryGenreName": "Action & Adventure",
                "contentAdvisoryRating": "PG-13",
                "shortDescription": "An unprecedented
cinematic journey ten years in the making and spanning the
entire Marvel Cinematic",
                "longDescription": "An unprecedented cinematic
journey ten years in the making and spanning the entire Marvel
Cinematic Universe, Marvel Studios' Avengers: Infinity War
brings to the screen the ultimate, deadliest showdown of all
time. The Avengers and their Super Hero allies must be willing
to sacrifice all in an attempt to defeat the powerful Thanos
before his blitz of devastation and ruin puts an end to the
universe."
                }]
}
```

Código-fonte 1 – Exemplo de retorno da API do iTunes
Fonte: Elaborado pelo autor (2019)

Note que essa API nos retorna um objeto que possui uma propriedade chamada **results**, que é um Array de resultados. Cada resultado nos traz várias informações do filme pesquisado, como artistName (nome dos diretores), releaseDate (lançamento), entre outros. Para nosso app, a informação que importa é previewUrl, que nos informa a URL do trailer do filme. Para transformar esse JSON em objetos que possamos trabalhar, vamos criar duas structs: uma para representar o JSON principal (chamaremos de **APIResult**) e outra para representar o resultado (chamaremos de **Trailer**). Deixe o seu arquivo API.swift conforme o código-fonte "Implementando APIResult e Trailer"

```
import Foundation

struct APIResult: Codable {
    let results: [Trailer]
}
// Implementando APIResult e Trailer
```

```
struct Trailer: Codable {
    let previewUrl: String
}
```

Código-fonte 2 – Implementando APIResult e Trailer
Fonte: Elaborado pelo autor (2019)

Agora, vamos criar a nossa classe API que irá conter o método que fará a requisição. Para isso, precisamos criar nosso objeto URLSession, configurá-lo por meio da classe URLSessionConfiguration e, no método de chamada da API, vamos criar nossa URLDataTask que fará a chamada e retornará o JSON. No final, a nossa classe API ficará conforme o código-fonte “Implementando APIResult e Trailer”. Implemente o código e atente para as explicações contidas nele.

```
import Foundation

struct APIResult: Codable {
    let results: [Trailer]
}

struct Trailer: Codable {
    let previewUrl: String
}

class API {
    //Este objeto irá conter o endereço base para nossa API.
    // Ao final dessa URL iremos adicionar o nome do filme
    static let basePath =
"https://itunes.apple.com/search?media=movie&entity=movie&term
="

    //Aqui estamos criando nosso objeto
URLSessionConfiguration,
    //utilizando o modo default
    static let configuration: URLSessionConfiguration = {
        let config = URLSessionConfiguration.default

        //Vamos permitir o uso de conexões via redes celulares
        config.allowsCellularAccess = true

        //Abaixo, atribuímos o header de Content-Type à nossa
config.
        //Com isso, estamos dizendo que nossa requisição
trabalha com
        //conteúdo em formato JSON
        config.httpAdditionalHeaders = ["Content-Type":
"application/json"]
```

```
        //Limitamos o tempo de espera de resposta para 45
segundos
        config.timeoutIntervalForRequest = 45.0

        //No máximo, 5 conexões poderão ser feitas
        config.httpMaximumConnectionsPerHost = 5

        return config
    }()

    //Criamos o objeto URLSession, passando o arquivo de
configuração como parâmetro.
    //Note que não usamos nenhum dos singletons (.default,
.efhemeral, .background)
    static let session = URLSession(configuration:
configuration)

    //Se quiséssemos usar uma session mais simples, com
    //configurações padrões, poderíamos usar a opção abaixo
    //static let session = URLSession.shared

    //Este método fará a leitura de todas as informações
    //referentes ao título do filme passado. Note que estamos
trabalhando
    //com métodos de classe (métodos estáticos)
    static func loadTrailers(title: String, onComplete:
@escaping (APIResult?) -> Void) {

        //A linha abaixo formata o título para que possa ser
usado na URL de chamada,
        //pois sabemos que URLs não podem conter espaços ou
caracteres especiais.
        guard let encodedTitle =
(title).addingPercentEncoding(withAllowedCharacters:
.urlHostAllowed),

        //Combinando a url base com o título do filme
        let url = URL(string: basePath+encodedTitle) else {
            onComplete(nil)
            return
        }




        //Estamos utilizando o método dataTask que aceita como
parâmetros a url com a rota
        //que será chamada e a closure que será executada nos
```

```
trazendo todas as informações da requisição,
        //como o objeto Data, a resposta (URLResponse) e o
erro caso tenha ocorrido
        let task = session.dataTask(with: url) { (data: Data?,
response: URLResponse?, error: Error?) in
            if error != nil {
                print(error!.localizedDescription)
                onComplete(nil) //Caso tenha ocorrido algum
erro, devolvemos nil para o onComplete
            } else {
                //Fazemos o cast de URLResponse para
HTTPURLResponse pois esperamos uma resposta HTTP
                guard let response = response as?
HTTPURLResponse else {
                    onComplete(nil) //Não obtivemos um objeto
válido de resposta do servidor
                    return
                }
                //Caso o código seja 200, o servidor respondeu
com sucesso
                if response.statusCode == 200 {
                    guard let data = data else {
                        onComplete(nil) //Não trouxe dados
                        return
                    }
                    //Criando nosso objeto json com os dados
obtidos
                    do {
                        let apiResult = try
JSONDecoder().decode(APIResult.self, from: data)
                        onComplete(apiResult)  //Devolvemos o
array de resultados
                    } catch {
                        onComplete(nil)
                    }
                } else {
                    onComplete(nil)
                }
            }
        }

        //Esta chamada é fundamental pois sem ela a task não é
executada
        task.resume()
    }
}
```

Código-fonte 3 – Implementando APIResult e Trailer (2)
Fonte: Elaborado pelo autor (2019)

Uma coisa importante para a qual se atentar no código apresentado é que nosso método loadTrailers recebe dois parâmetros, o **title**, que contém o título do filme que será pesquisado, e o **onComplete**, que pede uma closure que será executada após a requisição ser finalizada. Isso é muito importante, pois uma requisição é um processo assíncrono, ou seja, só retorna o resultado após certo tempo e, quando esse resultado chegar, nós precisaremos repassá-lo para algum lugar, e isso será feito chamando a closure que virá no parâmetro onComplete. A palavra reservada **@escaping**, que vemos ao lado do parâmetro, é fundamental, pois ela garante que aquele parâmetro (no caso, aquela closure) continuará existindo mesmo depois de o método ser encerrado. Lembre-se: o retorno da chamada ocorrerá depois de o método loadTrailers já ter sido finalizado.

1.2 Threads

Quando efetuamos uma chamada de uma URLDataTask, essa chamada ocorre em background, em um processo diferente do processo em que ocorrem todas as outras instruções e códigos que fizemos no nosso app.

Em iOS, o mecanismo que trata de filas de processos, permitindo, assim, a multitarefa em nossos devices, é chamado de GCD (Grand Central Dispatch). Ele é responsável por gerenciar a execução de tasks (tarefas ou conjunto de códigos), seja síncrona ou assincronamente, de maneira serial (uma após a outra) ou paralela (ao mesmo tempo ou concorrentes).

Para trabalhar com o GCD, fazemos uso da classe DispatchQueue, que foi criada com o intuito de simplificar e centralizar o uso de filas (queues), nos permitindo criar e modificar filas, bem como executar códigos assíncronos e síncronos, dentre outras coisas.

1.2.1 DispatchQueue

É esse mecanismo que gerencia o uso da DispatchQueue e demais classes. Algumas filas que precisamos usar constantemente já são criadas e fornecidas automaticamente para nós pela classe DispatchQueue. Por exemplo, temos a fila

.global que é uma fila padrão que pode ser executada para os mais diversos tipos de tarefas.

Uma das principais e mais usadas filas disponíveis é a main queue. É nela que ocorrem todas as tarefas principais do seu aplicativo e, **obrigatoriamente, qualquer tarefa que modifique elementos visuais do seu app precisa ser executada nessa fila**. Essa é uma fila serial, ou seja, todos os processos que estão rodando nela são executados sequencialmente.

Para utilizá-la, usamos **DispatchQueue.main**, passando na sequência, a forma de execução, síncrona, assíncrona, etc. No nosso aplicativo, ao recebermos o trailer do filme solicitado, vamos armazenar essa informação em uma variável para que, quando o usuário tocar no botão play, ele possa executar o filme. Como a única coisa que vamos fazer ao receber o trailer é armazenar em uma variável, não precisaríamos, necessariamente, colocar a execução desse código dentro da thread main, porém vamos fazer isso para que saiba como fazer, caso precise modificar elementos visuais baseados na resposta de uma requisição.

Abra a classe ViewController. É nesta classe, na qual visualizamos os dados do filme, que faremos a chamada da API e iremos recuperar o trailer do filme.

Nessa classe iremos criar uma nova variável, chamada **trailer**, que é uma String e irá receber a url do trailer. Vamos criar um método chamado **loadTrailers** e chamá-lo na viewDidLoad. Esse método, além de realizar chamada do método loadTrailers da API, passando o título do filme, irá recuperar, por intermédio da closure, o resultado e armazenar o trailer na variável criada anteriormente. Essa parte do código será executada na main thread. O código-fonte “Chamando API na classe ViewController” mostra como **ficará o início da classe** ViewController, juntamente com as devidas explicações.

```
import UIKit

class ViewController: UIViewController {

    var movie: Movie?

    //Variável que irá receber o trailer do filme
    var trailer: String = ""

    @IBOutlet weak var ivMovie: UIImageView!
```

```
    @IBOutlet weak var lbTitle: UILabel!
    @IBOutlet weak var lbCategories: UILabel!
    @IBOutlet weak var lbDuration: UILabel!
    @IBOutlet weak var lbRating: UILabel!
    @IBOutlet weak var tvSummary: UITextView!

    override func viewDidLoad() {
        super.viewDidLoad()

        //Chamando o método loadTrailers passando o título
        //do filme
        if let title = movie?.title {
            loadTrailers(title: title)
        }
    }

    func loadTrailers(title: String) {

        //Chamamos o método loadTrailers da classe API e na
        //closure vamos recuperar o primeiro resultado da
        //lista de resultados.
        API.loadTrailers(title: title) { (apiResult) in
            guard let results = apiResult?.results, let
trailer = results.first else {return}

            //Aqui, pegamos a url do trailer e atribuímos à
            //nossa variável trailer, fazendo na thread main.
            DispatchQueue.main.async {
                self.trailer = trailer.previewUrl
                print("URL do Trailer:", trailer.previewUrl)
            }
        }
    }
```

Código-fonte 4 – Chamando API na classe ViewController
Fonte: Elaborado pelo autor (2019)

1.3 Alertas e Datepicker

É muito comum precisarmos mostrar algum alerta para o usuário, seja para notificá-lo de que algo aconteceu, seja para pedir permissão para executar uma ação ou até mesmo para confirmar algo. Uma coisa bem legal que é possível fazer em iOS é criar alertas com campos de texto, pois assim, podemos, inclusive, pedir que o usuário nos forneça alguma informação para ser trabalhada pelo alerta.

1.3.1 UIAlertController

Para criarmos alertas em iOS, utilizamos a classe UIAlertController. Existem dois tipos, **Alert** e **Action Sheet**. Alert é o tipo de alerta que aparece no centro da tela, e é o mais comumente utilizado. Action Sheet aparece na parte de baixo da tela e costuma ser usado quando queremos que o usuário escolha uma ação de continuação, dentre várias.

Por meio da classe UIAlertController, configuramos nossos alertas e action sheets com o título, a mensagem que queremos mostrar ao usuário e os botões, ou actions, que ele irá escolher. Após a configuração, apresentamos o método **present(_:animated:completion:)**. O UIKit apresenta alertas e action sheets modalmente, por cima da sua tela.

Além de mostrar a mensagem com suas respectivas actions, poderá associar closures a essas actions, que serão executadas quando o usuário tocar nos botões. Cada um desses botões é adicionado no alerta pelo método **addAction(_:)**.

Além do título, mensagem e actions, também podemos incluir na Alert Controller, o Text Fields para recuperar do usuário alguma informação de que necessitarmos. Isso é feito por intermédio do método **addTextField(configurationHandler)**. Esse método lhe fornece uma closure como parâmetro, em que podemos configurar o seu Text Field (por exemplo, mudar o seu placeholder ou seu tipo de teclado).

1.3.2 UIDatePicker

Existe um componente que é responsável por solicitar ao usuário uma data ou um horário, e esse componente é o **Date Picker**.

Podemos usar um date picker para permitir que o usuário entre com uma data representando um momento no tempo (data no calendário, hora/minuto ou ambos) ou um intervalo de tempo (por exemplo, para um timer). É possível fazer com que o Date Picker reporte, por meio de um método, as interações ocorridas nele.

É possível utilizar o date picker diretamente dentro de uma view ou também como sendo o teclado do usuário, ou seja, podemos vinculá-lo a um text field para

que, ao invés do teclado numérico ou alfanumérico, apareça um date picker quando o usuário tocar no text field.

No nosso aplicativo, iremos permitir que o usuário possa definir um alerta para lembrá-lo de assistir ao filme escolhido. Falaremos de alertas mais para frente, porém, a maneira como faremos isso será por meio de um botão dentro da tela de visualização do filme. Quando o usuário tocar nesse botão, surgirá um alerta solicitando que ele entre com o dia, mês, ano, hora e minuto em que deseja ser lembrado para assistir a um determinado filme.

Para isso, abra a Main.storyboard e adicione um botão logo abaixo do label de duração do filme. Troque a cor da fonte para **laranja**, colocando o texto como **Agendar lembrete**. O frame desse botão será X: **211**, Y: **328**, Width: **167** e Height: **35**. Agora, crie as constraints de topo (**16**) e direita (**16**).

No final, sua tela ficará como mostra a Figura “Tela de filme com botão Agendar lembrete”.

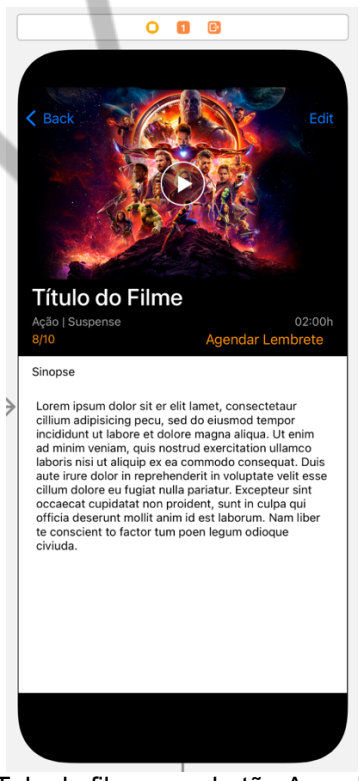

Figura 1 – Tela de filme com botão Agendar lembrete
Fonte: Elaborado pelo autor (2023)

Agora, crie uma action para esse botão e chame o método de **scheduleNotification.** Esse método irá mostrar o alerta para o usuário, perguntando qual a data em que ele deseja ser lembrado.

Antes de implementarmos esse método, precisamos criar um objeto do tipo **UIDatePicker,** que será utilizado para recuperar a data pelo usuário, e também um objeto que será o alerta que iremos mostrar. No início da classe ViewController, implemente os códigos conforme o código-fonte “Criando objetos do Date Picker e do Alert” (logo abaixo do outlet tvSummary).

```
    //Criando datepicker para recuperar a data
    var datePicker = UIDatePicker()

    //Criando objeto de alerta
    var alert: UIAlertController!
```

Código-fonte 5 – Criando objetos do Date Picker e do Alert
Fonte: Elaborado pelo autor (2019)

O objeto alert será instanciado quando o usuário tocar no botão. Outra coisa que precisamos fazer é definir que nosso date picker não aceitará uma data inferior à data atual, para que não seja possível criar um alerta para o passado. Isso é feito por meio da propriedade **minimumDate**. Vamos fazer com que essa propriedade receba a data atual e, para isso, instanciaremos a classe Date. Esta é a classe que usamos para trabalhar com datas em iOS, e ela representa um momento no tempo. Quando instanciamos essa classe, criamos um objeto que representa aquele momento no tempo (o momento atual, contendo dia, mês, ano, hora, minuto, segundo, etc.).

Além disso, precisamos fazer com que o date picker execute um método sempre que seu valor for alterado pelo usuário. Precisamos disso, pois queremos que o Text Field reflita o que o usuário acabou de escolher no date picker. Fazemos isso chamando o método addTarget(_ target: Any?, action: Selector, for controlEvents: UIControlEvents). Esse método pede o **target**, ou seja, a classe na qual se encontra o método que será chamado, a ação (método) que será disparada e qual o evento que irá disparar tal ação. No nosso caso, criaremos mais tarde um método chamado **changeDate**, que será chamado pelo date picker quando o seu evento **valueChanged**, ou seja, valor mudado, for disparado. Implementaremos

todos esses códigos dentro do método viewDidLoad. No final, seu método viewDidLoad ficará conforme o código-fonte “Definindo data mínima do Date Picker”.

```
override func viewDidLoad() {
    super.viewDidLoad()
    if let title = movie?.title {
        loadTrailers(title: title)
    }

    //Definindo a data mínima do datePicker para hoje
    datePicker.minimumDate = Date()

    //Adicionando método a ser chamado quando o DatePicker
    //tiver seu valor alterado
    datePicker.addTarget(self, action: #selector(changeDate),
for: .valueChanged)
}
```

Código-fonte 6 – Definindo data mínima do Date Picker
Fonte: Elaborado pelo autor (2019)

Note que temos que usar **#selector** para definir o método que será chamado. Perceba também que passamos apenas o seu nome, sem ().

Agora, vamos criar o método changeDate. Esse método irá pegar a data associada ao date picker e irá formatar essa data para que apareça dentro do text field. A data é recuperada por meio da propriedade **date** e, para formatá-la em um texto que o usuário entenda, usamos a classe DateFormatter. Essa classe define como será a formatação de uma data por meio de sua propriedade **dateFormat**. Passamos uma string para essa propriedade, definindo como é o formato que queremos e, no nosso caso, usaremos "dd/MM/yyyy HH:mm'h", ou seja, orientamos o preenchimento do item dia com dois algarismos (dd), mês com dois algarismos (MM) e ano com quatro (yyyy). Além disso, vamos mostrar a hora (HH) e os minutos (mm), seguindo da letra h ('h).

Usamos o método **string(from date: Date) -> String**, presente no date formatter, para recuperar o texto a partir de uma data e associar esse texto ao text field presente no alerta. O método changeDate ficará como mostra o código-fonte “Definindo data mínima do Date Picker”.

```
@objc func changeDate() {

        //Criamos o dateFormatter para formatar a data
        let dateformatter = DateFormatter()

        //Aqui, definimos o estilo que a data será
        //apresentada
        dateformatter.dateFormat = "dd/MM/yyyy HH:mm'h"

        //Recuperamos a data do datePicker para repassar ao
        //TextField
        alert.textFields?.first?.text =
dateformatter.string(from: datePicker.date)
    }
```

Código-fonte 7 – Definindo data mínima do Date Picker
Fonte: Elaborado pelo autor (2019)

Um detalhe: é obrigatório o uso de **@objc** antes de qualquer método que sirva como target de um componente. Isso é necessário, pois o componente foi escrito em **Objective-C** e nossa classe é em **Swift**.

Agora, vamos implementar o método scheduleNotification. É implementado o código-fonte “Definindo data mínima do Date Picker”, atentando para as explicações.

```
@IBAction func scheduleNotification(_ sender: UIButton) {

    //Criando Alert Controller como .alert
    alert = UIAlertController(title: "Lembrete", message:
"Quando deseja ser lembrado de assistir o filme?",
preferredStyle: .alert)

    //Adicionando um Text Field ao alerta
    alert.addTextField { (textField) in
        textField.placeholder = "Data do lembrete"
        self.datePicker.date = Date()
        textField.inputView = self.datePicker
    }

    //Adicionando botões de Agendar e Cancelar. Para criar uma
    //Action, definimos o seu titulo e seu estilo.
    //O estilo .default é o padrão.
    let okAction = UIAlertAction(title: "Agendar", style:
.default) { (action) in
        self.schedule()
    }
```

```
    //Em uma action que serve para cancelar uma ação, o ideal
    //é utilizar o estilo .cancel
    let cancelAction = UIAlertAction(title: "Cancelar", style:
.cancel, handler: nil)

    //Aqui, adicionamos as duas actions ao alert
    alert.addAction(okAction)
    alert.addAction(cancelAction)

    //Mostrando o alerta para o usuário
    present(alert, animated: true, completion: nil)
}
```

Código-fonte 8 – Definindo data mínima do Date Picker
Fonte: Elaborado pelo autor (2019)

Você deve ter notado que na closure da action “Agendar”, nós chamamos um método schedule. Esse método será construído daqui a pouco e será o responsável por agendar a notificação de lembrete.

1.4 Utilizando notificações locais

Precisamos agora, que nosso app mostre ao usuário uma notificação indicando que chegou a hora de assistir ao filme desejado. Notificações são recursos utilizados em aplicativos, pois é uma forma de lembrar ao usuário que algo relativo ao seu app está acontecendo. Por intermédio delas, podemos mostrar uma espécie de alerta para o usuário, que irá abrir o app em questão quando for tocada.

1.4.1 Criando notificações

Para criarmos uma notificação, utilizamos a classe **UNMutableNotificationContent**, definindo seu título, seu corpo (mensagem principal) dentre outras coisas.

A data em que a notificação será disparada é definida pela classe **UNCalendarNotificationTrigger**. É possível definir um momento futuro (com dia, hora, minutos, etc.) para que aquela notificação apareça no device do usuário. O

agendamento dessa notificação é realizado pela classe **UNNotificationRequest**, que extrai o conteúdo, o trigger e cria uma requisição de notificação.

Toda e qualquer notificação é agendada pelo Centro de Notificações do Usuário, que é acessado pela classe **UNUserNotificationCenter**. É por meio dela que, efetivamente, colocamos aquela notificação na fila para ser disparada quando chegar o momento certo.

Todas essas classes estão presentes no framework **UserNotifications**, ou seja, é necessário importá-lo em qualquer classe que for fazer uso de notificações.

Vamos aprender na prática como criar e disparar notificações. Na classe ViewController, certifique-se de importar o framework UserNotifications colocando o código **import UserNotifications** logo abaixo de **import UIKit**.

Agora, crie o método schedule conforme o código-fonte “Criando e agendando notificações”. Preste atenção à explicação de cada linha de código.

```
//No método abaixo iremos agendar uma notificação
func schedule() {
    //Criamos o objeto que define o conteúdo da notificação
    let content = UNMutableNotificationContent()

    //Definimos seu título, pela propriedade title
    content.title = "Lembrete"

    //E sua mensagem, através da propriedade body
    let movieTitle = movie?.title ?? ""
    content.body = "Assistir filme \"\(movieTitle)\""

    //Criamos um objeto DateComponents que contém os
    //elementos que formam uma data
    let dateComponents =
Calendar.current.dateComponents([.year, .month, .day, .hour,
.minute], from: datePicker.date)

    //Objeto que define a condição para o disparo da
    //notificação
    let trigger = UNCalendarNotificationTrigger(dateMatching:
dateComponents, repeats: false)

    let request = UNNotificationRequest(identifier:
"Lembrete", content: content, trigger: trigger)
    UNUserNotificationCenter.current().add(request,
withCompletionHandler: nil)
}
```

Código-fonte 9 – Criando e agendando notificações
Fonte: Elaborado pelo autor (2019)

Uma coisa que não foi falada foi sobre a classe **DateComponents**. Sempre que precisamos separar uma data em componentes, ou seja, em dia, hora, minutos, segundo, etc., fazemos uso dessa classe. Um objeto do tipo DateComponents é um objeto que possui uma data com todos os seus componentes separados em propriedades, ou seja, existe uma propriedade para o dia, outra para a hora, e por aí vai.

Criamos um objeto DateComponents utilizando a classe **Calendar**, e seu singleton **current**. Essa classe possui um método chamado **func dateComponents(_ components: Set<Calendar.Component>, from date: Date) -> DateComponents**, que devolve um DateComponents baseado em uma data e também em quais componentes dessa data nós desejamos utilizar.

No nosso exemplo, nós criamos um DateComponents extraindo a data do datePicker e definindo que, dessa data, nós queríamos apenas o ano, mês, dia, hora e minuto, ou seja, não queremos os segundos. Isso é importante porque, quando criar um trigger, se usar os segundos, ele só será disparado naquele dia, mês, ano, hora, minuto e segundo, e não queremos isso. Queremos que seja disparado assim que o minuto chegar, ou seja, desconsiderando o segundo em que aquele trigger foi criado.

1.4.2 Apresentando notificações ao usuário

Quando uma notificação é criada e agendada, isso não significa que ela será realmente apresentada ao usuário, isso ocorre porque nós precisamos solicitar ao usuário que ele permita que as notificações sejam disparadas pelo aplicativo. **Esse controle é feito por aplicativo**, e o usuário precisa aceitar o uso de notificações para que elas possam funcionar.

Quem solicita a permissão é o próprio Centro de Notificações (UNUserNotificationCenter). É ele que possui o método para pedir permissão ao usuário, bem como agir quando uma notificação for tocada.

Abra o arquivo AppDelegate.swift. No AppDelegate iremos perguntar ao usuário se ele aceita notificações, e isso será feito assim que o app for aberto.

Primeiro importe, nessa classe, o framework (**import UserNotifications**). Agora, implemente no método **application(_ application:didFinishLaunchingWithOptions launchOptions:)** que já existe nesta classe (note que o return true já existe e é necessário ter apenas 1), os códigos que farão a solicitação ao usuário. Seu método ficará como no código-fonte “Solicitando permissão ao usuário para exibir notificações”.

```
func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
[UIApplicationLaunchOptionsKey: Any]?) -> Bool {

    //Definimos o delegate do UNUserNotificationCenter como
    //sendo esta própria classe.
    //Com isso, esta classe irá responder pelo
    //UNUserNotificationCenter.
    UNUserNotificationCenter.current().delegate = self

    //Primeiro solicitamos que nos seja fornecido o ajuste
    //atual, ou seja, o que o usuário respondeu quando foi
    //solicitada a permissão.
    UNUserNotificationCenter.current().getNotificationSettings
{ (settings) in

        //Se o status de autorização for notDetermined,
        //significa que o usuário nunca foi pedido para
        //aceitar (ou seja, é a primeira vez que o app é
        //aberto. Nesse caso, iremos solicitar a permissão.
        if settings.authorizationStatus == .notDetermined {

            //Primeiro, criasse um arquivo de opções
            //indicando o que nossa notificação irá fazer.
            //No nosso caso, iremos mostrar o alerta da
            //notificação e tocar um som.
            let options: UNAuthorizationOptions = [.alert,
.sound]

            //Aqui, através do método requestAuthorization,
            //solicitamos a autorização

UNUserNotificationCenter.current().requestAuthorization(option
s: options, completionHandler: { (success, error) in

                //A propriedade success indica se o usuário
                //autorizou ou não.
                if error == nil {
```

```
                    print(success)
                } else {
                    print(error!.localizedDescription)
                }

            })

        //Caso o usuário tenha negado, sempre imprimimos a
        //mensagem abaixo.
        } else if settings.authorizationStatus == .denied {
            print("Usuário negou a Notificação")
        }
    }
    return true
}
```

Código-fonte 10 – Solicitando permissão ao usuário para exibir notificações
Fonte: Elaborado pelo autor (2019)

Uma coisa importante: caso o usuário tenha negado a notificação, ele nunca mais verá a tela de solicitação, pois o iOS só exibe uma vez, ou seja, se o usuário não aceitou, essa decisão será respeitada e não será mostrada novamente na tela, mesmo que você force isso no código.

Agora, para finalizar, precisamos implementar o protocolo UNUserNotificationCenterDelegate, para que nossa classe possa ser a delegate do Notification Center. Nesse protocolo existem dois métodos, um que deverá ser implementado para que as notificações apareçam no device quando o app estiver em background ou fechado, e outra para que elas apareçam quando o app estiver aberto.

Basicamente, se quisermos esses comportamentos, precisamos implementar esses métodos e, dentro deles, executar seus completionHandlers.

Implemente o código-fonte "Implementando UNUserNotificationCenterDelegate" após o final da classe AppDelegate (após a última chave - **}**).

```
extension AppDelegate: UNUserNotificationCenterDelegate {

    //Método chamado quando a notificação for aparecer com o
    //app aberto
    func userNotificationCenter(_ center:
UNUserNotificationCenter, willPresent notification:
```

```
UNNotification, withCompletionHandler completionHandler:
@escaping (UNNotificationPresentationOptions) -> Void) {
        completionHandler([.banner, .sound])
    }

    //Método chamado quando a notificação for aparecer com o
    //app fechado ou em background
    func userNotificationCenter(_ center:
UNUserNotificationCenter, didReceive response:
UNNotificationResponse, withCompletionHandler
completionHandler: @escaping () -> Void) {
        completionHandler()
    }
}
```

Código-fonte 11 – Implementando UNUserNotificationCenterDelegate
Fonte: Elaborado pelo autor (2019)

Parabéns, agora seu app, além de consumir uma API REST para recuperar o trailer de um filme, ainda permite que o usuário possa agendar uma data para assisti-lo.

Figura 2 – Mostrando notificação
Fonte: Elaborado pelo autor (2023)

CONCLUSÃO

Sua aplicação deixou de ser estática e se tornou dinâmica. Lembra-se do webservice que construiu no padrão REST? Ele está pronto para fornecer os dados em JSON para consumir no aplicativo. O projeto está quase no fim, e para encerrarmos esta trilha com chave de ouro, entraremos no ecossistema de sensores e multimídia.

REFERÊNCIAS

APPLE. **Framework UIKit** 2023. Disponível em: <https://developer.apple.com/documentation/uikit>. Acesso em: 27 jun. 2023.

APPLE. **Framework UserNotification** 2023. Disponível em: <https://developer.apple.com/documentation/usernotifications>. Acesso em: 27 jun. 2023.