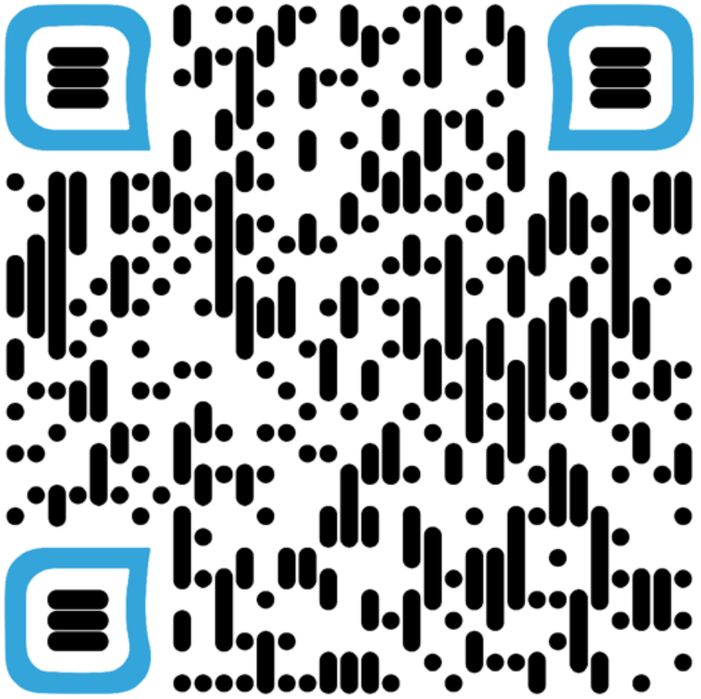
PROF. PEDRO HENRIQUE

DESENVOLVIMENTO PARA IOS 11 COM SWIFT 4

GITHUB DA TURMA



HTTPS://GIT.IO/VF50W

ONDE ENCONTRAR O MATERIAL?

LEIA O QRCODE

ALÉM DO TRADICIONAL BLACKBOARD DO IESB

AGENDA

- Multithreading
  - Mantendo a interface ágil
  - Usando o componente UIActivityIndicator
- Fazendo o SplitView abrir na master
- O componente UITextField

MULTITHREADING NO IOS - FILAS!

- Quando falamos de multithreading no iOS, estamos falando majoritariamente de filas de execução;
- As funções (ou closures) são simplesmente enfileiradas numa das diversas filas de execução existentes;
- Então, as funções são “consumidas" da fila e executadas numa thread associada àquela fila;
- As filas podem ser do tipo serial, quando uma função é executada por vez. As filas também podem ser do tipo concorrente, quando várias threads consomem a fila ao mesmo tempo.

A FILA PRINCIPAL

- Existe uma fila muito especial, do tipo serial, que é chamada **main queue**;
- Toda e qualquer atividade envolvendo a interface gráfica, deve obrigatoriamente acontecer nesta fila. Apenas nela;
- E, como é fácil deduzir, atividades não envolvidas com a interface gráfica e que consomem tempo/recursos não devem acontecer na fila principal;
- Faz-se assim porque deseja-se que a interface gráfica tenha o tanto responsiva quanto for possível às entradas do usuário;
- Faz-se assim também porque deseja-se que o acontece na UI aconteça de forma previsível (serial, como a natureza da fila);
- Por padrão, as funções assíncronas são executadas na **main queue** apenas quando ela está “quieta”.

FILAS GLOBAIS DO SISTEMA

- Para trabalhos não adequados à **main queue**, você normalmente vai usar uma fila global, que é compartilhada e do tipo concorrente;
- Para obter a fila principal, onde toda e qualquer atividade da UI deve acontecer, usar:
  let mainQueue = DispatchQueue.main
- Para obter uma fila global compartilhada e concorrente, onde você vai, na grande maioria das vezes, colocar o trabalho pesado:
  let backgroundQueue = DispatchQueue.global(qos: DispatchQoS) //onde:
  DispatchQoS.userInteractive //alta prioridade, só coisas pequenas e rápidas
  DispatchQoS.userInitiated //alta prioridade, mas pode demorar um pouco mais
  DispatchQoS.background //não iniciado diretamente pelo usuário, pode rodar lentamente
  DispatchQoS.utility //tarefa de plano de fundo de longa duração, baixa prioridade

ENTRANDO NA FILA

- No iOS, a arte do multithread pode ser resumida no saber colocar os processos nas filas certas;
- Existem duas formas principais de enfileirar uma closure:
- Colocar uma função/closure numa determinada fila e seguir com o fluxo de execução como se nada tivesse acontecido:
  queue.async { ... }
- Ou você pode bloquear a fila até que a closure termine de executar na outra fila:
  queue.sync { ... }
- Na imensa maioria das vezes, usamos a primeira opção.

OBTENDO UMA FILA NÃO-GLOBAL

- Em raras ocasiões você pode precisar de uma fila diferente das globais e da principal;
- Por exemplo, você pode querer sua própria fila serial em background. Esta situação só faz algum sentido quando você tem múltiplas tarefas que são dependentes da sequencia de execução...
  let serialQueue = DispatchQueue(label: “MinhaFilaSerial”)
- Você pode ainda criar sua própria fila concorrente em background. Mas isso é ainda mais raro, uma vez que você tem as filas globais...
  let concurrentQueue = DispatchQueue(label: “MinhaFila”, attributes: .concurrent)

MULTITHREADING

- Tem muito mais de onde saiu esse! Missão extra para casa: ler a documentação que fala do GCD (Grand Central Dispatch);
- Com ele você pode criar locks, proteger seções críticas, leitores e escritores, fazer enfileiramento síncrono, etc;
- Existe ainda uma terceira API para threads!
- Estamos falando de OperationQueue e Operation;
- Mas geralmente usamos a API DispatchQueue mesmo. Isso porque a legibilidade fica muito boa;
- Contudo, não ignoremos a API Operation. Ela é muito útil, especialmente para multithread mais complexo.

APIS NATIVAS QUE SÃO MULTITHREAD

- Poucos lugares no iOS vão fazer o que tem que ser feito fora da fila principal;
- Eles podem te oferecer a oportunidade de fazer alguma coisa fora da fila principal, perguntado por uma função (geralmente usamos uma closure) que é executada fora da **main queue**;
- Eventualmente o iOS pode te pedir uma closure para executar fora da fila principal;
- **Dica:** nunca se esqueça que quando você quiser fazer uma tarefa de UI, você precisa delegar a tarefa para a fila principal.

EXEMPLO DE API MULTITHREAD DENTRO DO IOS

```
let session = URLSession(configuration: .default)
if let url = URL(string: "http://www.google.com.br") {
    let task = session.dataTask(with: url) { (data: Data?, response, error) in
        print("Recebi os dados: \(data)")
    }
    task.resume()
}
```

EXEMPLO DE API MULTITHREAD DENTRO DO IOS

```
if let url = URL(string: "http://www.google.com.br") {
    let task = session.dataTask(with: url) { (data: Data?, response, error) in
        // Quero fazer coisas de UI aqui,
        // usando o valor de 'data' que foi baixado.
        // Posso?
    }
    task.resume()
```

NÃO! POR QUE O CÓDIGO DENTRO DESTA CLOSURE É EXECUTADO FORA DA FILA PRINCIPAL. COMO PODEMOS LIDAR COM ISSO?

UM JEITO É USAR UMA VARIANTE DESTA API QUE DEIXA VOCÊ ESPECIFICAR A FILA ONDE A CLOSURE IRÁ SER EXECUTADA (NO CASO VOCÊ INDICARIA A MAIN QUEUE). OUTRO JEITO É O QUE VAMOS VER NO PRÓXIMO SLIDE, USANDO O GCD...

MULTITHREADING

```
let session = URLSession(configuration: .default)
if let url = URL(string: "http://www.google.com.br") {
    let task = session.dataTask(with: url) { (data: Data?, response, error) in
        DispatchQueue.main.async {
            // coisas de UI aqui
        }
    }
    task.resume()
}
```

AGORA PODEMOS FAZER COISAS RELACIONADAS COM A INTERFACE. ISSO PORQUE O CÓDIGO RELACIONADO À UI FOI DESPACHADO DE VOLTA PARA A MAIN QUEUE.

HORA DA PRÁTICA

de Multithreading!

O COMPONENTE UITEXTFIELD

- É como um UILabel, mas é editável
- Digitação no iPhone é algo que se deseja manter no mínimo necessário. No iPad, no entanto, é algo mais usual;
- **Dica do ❤️**: não se deixe enganar pelo simulador! No simulador você pode usar o teclado físico do seu Mac. Use ⌘**K** para mostrar o teclado virtual do iPhone no simulador e veja o que acontece!
- No textField, você pode colocar texto formatado, mudar cores, alinhamento, fonte, etc. Tal qual um label

COMO FUNCIONA O TECLADO?

- O teclado virtual do iPhone aparece automaticamente quando o UITextField se torna o “first responder”;
- O UITextField se tornará o “first responder” quando o usuário interagir com ele;
- Ou, você pode forçar este comportamento chamando o método **becomeFirstResponder**.
- De maneira análoga, você pode fazer o teclado virtual desaparecer chamando o método **resignFirstResponder**.

UITEXTFIELD E O PROTOCOLO DE DELEGAÇÃO

- Para certas coisas, o UITextField faz uso do bom e velho **delegate**, especialmente para coisas relacionadas ao uso da tecla ENTER do teclado virtual

```
extension MyViewController : UITextFieldDelegate {

    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        if desejoProcessamentoNormalDoEnter {
            return true
        }else {
            return false
        }
    }

}
```

EXISTEM VÁRIOS OUTROS MÉTODOS INTERESSANTES NO PROTOCOLO UITEXTFIELDELEGATE! NÃO DEIXE DE CONFERIR!

UITEXTFIELD É UM UICONTROL

- UIControl é uma classe que sabe disparar ações (lembrar do alvo do MVC), tal qual um botão;
- Naturalmente, os eventos que o UITextField emite são diferentes daqueles emitidos por um UIButton;
- Clique com o botão direito em um UITextField no storyboard para ver as opções disponíveis.

CONTROLANDO O TECLADO

- É possível controlar a aparência do teclado que é exibido associado ao UITextField;
- As propriedades que podemos mudar em relação ao teclado são definidas no protocolo UITextInputTraits, que o UITextField implementa, naturalmente;

```
var autocapitalizationType: UITextAutocapitalizationType //words,sentences,etc.
var autocorrectionType: UITextAutocorrectionType //  .yes or .no
var returnKeyType: UIReturnKeyType // Go, Search, Google, Done, etc.
var isSecureTextEntry: Bool // quando for um campo de senha, por exemplo
var keyboardType: UIKeyboardType // ASCII, URL, PhonePad, etc.
```

- Os teclados ainda podem ter funcionalidades acessórias, que você pode incluir através de views extras que aparecem acima do teclado (como uma toolbar customizada). Você diz que view é essa através da propriedade **inputAccessoryView**, do UITextField.

O TECLADO... AH O TECLADO!

- O teclado aparece por cima de outras views, inclusive do UITextField que chamou o teclado!
- Você pode reagir às aparições do teclado ao se subscrever às **notificações** que são enviadas pela UIWindow;
- Certo... esse é um assunto que ainda não falamos e que vai ser tratado com mais profundidade em numa aula futura, mas o que temos a fazer com o UITextField é bem simples...

NOTIFICAÇÕES SOBRE O TECLADO

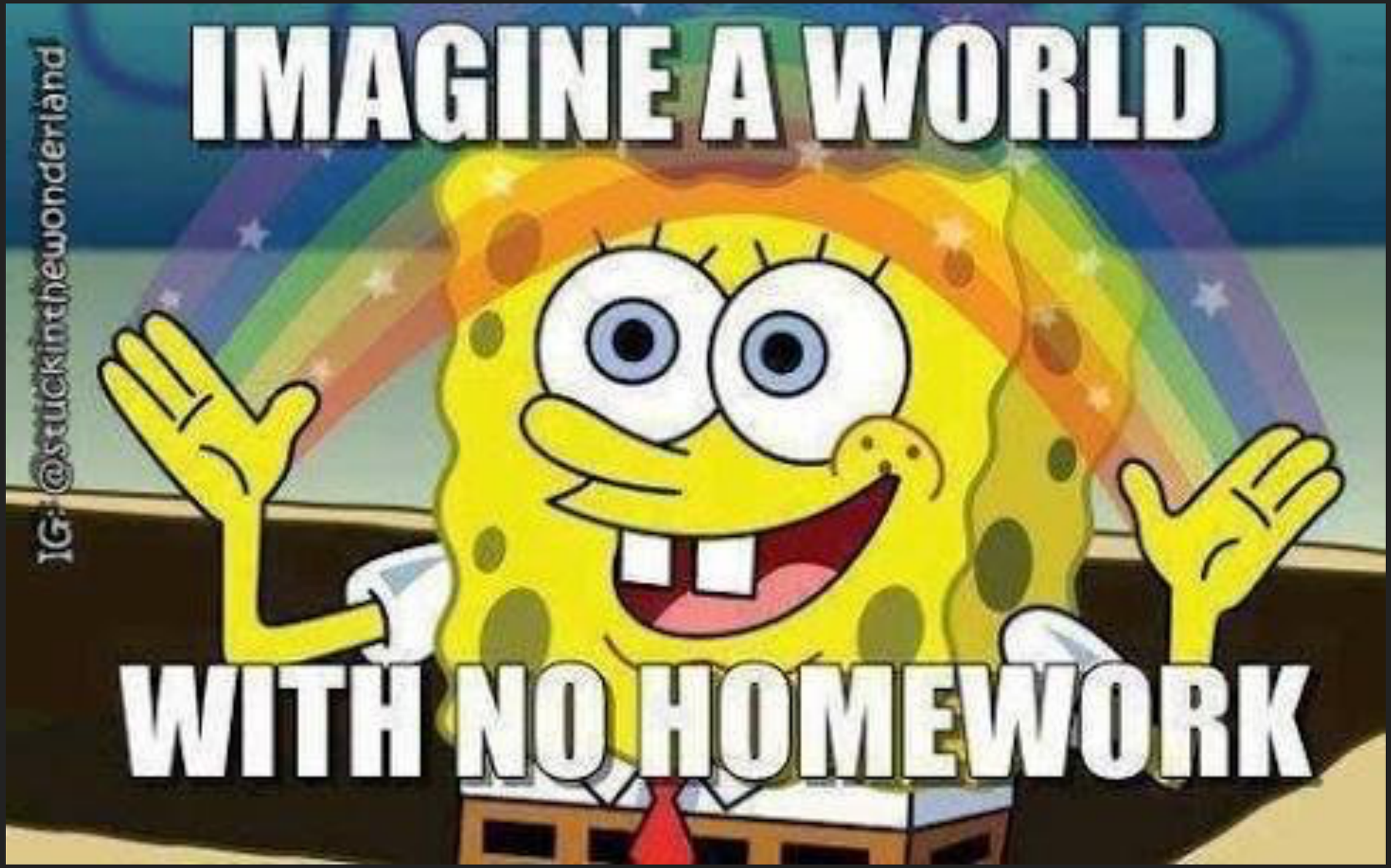
Este é o método que vai ser chamado quando o evento acontecer. O argumento deste método (um objeto do tipo Notification) traz um dicionário que dá detalhes sobre a aparência do teclado. Você sempre deve reagir adequadamente, conforme a aparência do teclado, para evitar que o text view fique obstruído pelo teclado. quando seu UITextField está dentro de uma célula de um UITableViewController, ele mesmo cuida disso e faz scroll automágicamente.

```
NotificationCenter.default.addObserver(self,
                                       selector: #selector(oTecladoApareceu(_:)),
                                       name: Notification.Name.UIKeyboardDidShow,
                                       object: view.window)
```

ESTE É O EVENTO QUE ESTAMOS OBSERVANDO

ESTE É O OBJETO QUE ESTÁ PROVOCANDO O EVENTO. NESTE CASO, É A JANELA DA NOSSA VIEW.

HORA DA PRÁTICA!!!


IMAGINE A WORLD
IG: @stuckinthewonderland
WITH NO HOMEWORK