

CENTRO UNIVERSITÁRIO ESTÁCIO DE BRASILIA

THIAGO ANDERSON MARTINS

CALCULADORA SWIFT

**TAGUATINGA, DF
2016**

Thiago Anderson Martins

CALCULADORA SWIFT

Trabalho de Conclusão de Curso apresentado à
banca examinadora do Centro Universitário Estácio
de Brasília, como requisito parcial à obtenção do
Título de Bacharel em Sistemas de Informação.

Orientador: Prof. João Paulo Pimentel.

TAGUATINGA, DF
2016

Thiago Anderson Martins

CALCULADORA SWIFT

Trabalho de Conclusão de Curso aprovado Centro Universitário Estácio de Brasília como requisito à obtenção do Título de Bacharel em Sistemas de Informação.

Taguatinga, DF, _____ de _____ de 2016

Prof. Esp. João Paulo Pimentel
Orientador

Prof. MSc. Welder Maurício de Souza

Prof. Esp. Raphael Alves Bruce

RESUMO

A linguagem de programação *Swift* facilita a criação de *apps* para *iOS*, e torna mais fácil e entrada de novos programadores ao crescente campo do desenvolvimento móvel. Este estudo introduz os conceitos básicos da linguagem *Swift*, explica extensivamente o funcionamento de um aplicativo de calculadora para *iOS*, e demonstra, passo a passo, a criação de um *app* “Olá Mundo” interativo.

Palavras Chave: Desenvolvimento *iOS*, Linguagem de programação *Swift*, *Xcode*, calculadora, *app*.

ABSTRACT

The Swift programming language facilitates iOS app creation, and makes it easier for new programmers to enter the burgeoning field of mobile development. This study introduces the basic concepts of Swift, goes through an extensive explanation of an iOS calculator app, and demonstrates, step by step, the creation of an interactive “hello world” app.

Key Words: iOS Development, Swift programming language, Xcode, calculator, app.

LISTA DE ILUSTRAÇÕES

Imagen 1.0 - Logotipo da linguagem Swift	19
Imagen 1.1 - Declaração de variável e constante em Swift	19
Imagen 1.2 - Exemplo de um loop do tipo for em Swift.....	20
Imagen 1.3 - Tratamento de erros em Swift	20
Imagen 1.4 - Função com número de argumentos variáveis	20
Imagen 1.5 - Escrevendo cabeçalhos em Swift	21
Imagen 1.6 - Sugestão automática do Xcode	21
Imagen 1.7 - Ajuda rápida do Xcode	22
Imagen 1.8 - if e let para valores opcionais (1/2)	22
Imagen 1.9 - if e let para valores opcionais (2/2)	23
Imagen 1.10 - Exemplo de um dicionário do tipo [String:String] em Swift	23
Imagen 1.11 - Exemplo de um dicionário do tipo [String:AnyObject] em Swift	23
Imagen 1.12 - Uso de caracteres Unicode em Swift.....	24
Imagen 1.13 - Exemplo do loop for em Swift	24
Imagen 2.0 - Ícone do app Calculadora Swift	26
Imagen 2.1 - Tela Principal (Vertical e Horizontal)	28
Imagen 2.2 - Tela Principal (Multi-Tarefas).....	28
Imagen 2.3 - Tela de Histórico.....	29
Imagen 2.4 - Menu de Opções.....	30
Imagen 2.5 - Tela de Cores	30
Imagen 2.6 - Tela de Ajuda.....	31
Imagen 2.7 - Tela de Mais Opções.....	32
Imagen 2.8 - Ciclo de Funcionamento da Calculadora Swift	33
Imagen 2.9 - Código do Método <code>recebeEntradasDoUsuário</code>	34

Imagen 2.10 - Parte do código do Método filtraExpressãoMatemáticaInválida	35
Imagen 2.11 - Código do Método mostraResultadoFinal	35
Imagen 2.12 - Código do método preparaEResolveExpressãoMatemática	36
Imagen 2.13 - Código do método tentaResolverExpressãoMatemáticaVálida	36
Imagen 2.14 - Código do Método resolveExpressaoMatematicaValidaECompleta	37
Imagen 2.15 - Código da extensão NSNumber e do método calculaCossenoGrau	38
Imagen 2.16 - Exemplo de sintaxe da função calculaCossenoGrau.....	38
Imagen 2.17 - Exemplo de sintaxe da função calculaRaizEnésima.....	39
Imagen 2.18 - Código da função calculaRaizEnésima.....	39
Imagen 2.19 - Estrutura de Chaves para Armazenamento Local	40
Imagen 2.21 - Código do método carregaERetornaTemaDaCalculadora.....	42
Imagen 2.22 - Criação de uma notificação de orientação de tela	42
Imagen 2.23 - Sintaxe de um seletor	43
Imagen 2.24 - Configurações iniciais	43
Imagen 2.25 - Código da função detectaIdiomaDoSistema.....	44
Imagen 2.26 - Vetor de idiomas e regiões disponíveis.....	44
Imagen 2.27 - Convertendo códigos de idioma/região em nomes de idiomas	45
Imagen 2.28 - Variável Global de Idioma	45
Imagen 2.29 - Ligações outlet do menu de opções	46
Imagen 2.30 - Menu de opções em português.....	46
Imagen 2.31 - Dicionários do menu de opções	46
Imagen 2.32 - Exemplo de uso do dicionário do botão de ajuda	47
Imagen 2.33 - Aplicando os dicionário ao texto dos botões	47
Imagen 2.34 - Opção de alterar idioma do menu de mais opções.....	47
Imagen 2.35 - Código do método traitCollectionDidChange	48
Imagen 2.36 - Parte do código da função detectaTipoDeTela	49

Imagen 2.37 - Ligação entre o contêiner pai e o código	49
Imagen 2.38 - Acessando as dimensões de um contêiner.....	50
Imagen 2.39 - Obtendo o tipo de tela.....	50
Imagen 2.40 - Top charts, aba de gratuitos da App Store (12/05/2016).....	51
Imagen 2.41 - Logotipo do GitHub	55
Imagen 3.0 - Tela de boas vindas do Xcode	56
Imagen 3.1 - Tela de seleção de modelos de projeto do Xcode	57
Imagen 3.2 - Opções do projeto “Olá Mundo”	57
Imagen 3.3 - Seleção de Diretório do Projeto “Olá Mundo”	58
Imagen 3.4 - Tela de Projeto Inicial do Xcode	58
Imagen 3.5 - Acessando o Main.storyboard através do navegador do Xcode.....	59
Imagen 3.6 - Escondendo o Navegador do Xcode	59
Imagen 3.7 - Pesquisando por “text” na Biblioteca de Objetos	60
Imagen 3.8 - Arrastando o objeto Text Field para a Interface	61
Imagen 3.9 - Pesquisando por “label” na Biblioteca de Objetos	61
Imagen 3.10 - Arrastando o Objeto Label para a Interface	62
Imagen 3.11 - Alterando o texto do objeto Label.....	62
Imagen 3.12 - Alinhando a etiqueta ao centro da interface	63
Imagen 3.13 - Pesquisando por “button” na Biblioteca de Objetos	63
Imagen 3.14 - Arrastando o Objeto Button para a Interface.....	64
Imagen 3.15 - Arrastando outro Objeto Button para a interface.....	64
Image 3.16 - Renomeando os botões para “Ok” e “Limpar”	65
Imagen 3.17 - Arrastando a etiqueta final para a interface	65
Imagen 3.18 - Apontando sobre a interface de redimensionamento do objeto	66
Imagen 3.19 - Redimensionando a etiqueta final	66
Imagen 3.20 - Centralizando o texto da etiqueta	67

Imagen 3.21 - Alterando o texto “Label” para “Olá, Mundo!”.....	67
Imagen 3.22 - Selecionando a palavra “Mundo”	68
Imagen 3.23 - Encontrando a interface de Emojis e Símbolos	68
Imagen 3.24 - Selecionando o Emoji de globo.....	69
Imagen 3.25 - Verificando a interface do app.....	69
Imagen 3.26 - Alterando o tamanho da fonte da etiqueta “Olá, Mundo”	70
Imagen 3.27 - Conclusão da construção da interface do app.....	70
Imagen 3.28 - Executando o Simulador pela primeira vez.....	71
Imagen 3.29 - Verificando a interface do app no Simulador iOS	71
Imagen 3.30 - Selecionando todos os objetos adicionados ao app	72
Imagen 3.31 - Clicando no botão Align	72
Imagen 3.32 - Adicionando Constraints Horizontais	73
Imagen 3.33 - Verificando os Erros de Constraints	73
Imagen 3.34 - Clicando no botão Pin	74
Imagen 3.35 - Adicionando constraints verticais, de altura e largura.....	74
Imagen 3.36 - Linhas azuis indicando ausência de erros de interface	75
Imagen 3.37 - Simulador iOS executando o app “Olá Mundo”	76
Imagen 3.38 - Mudando a orientação do Simulador iOS	76
Imagen 3.39 - App “Olá Mundo” em modo Multi-Tarefas para iPad	77
Imagen 3.40 - Escondendo o menu de utilidades do Xcode.....	77
Imagen 3.41 - Revelando o código fonte do app	78
Imagen 3.42 - Interface dividida do Xcode.....	78
Imagen 3.43 - Criando o outlet do campo de texto (1/3).....	79
Imagen 3.44 - Criando o outlet do campo de texto (2/3).....	79
Imagen 3.45 - Criando o outlet do campo de texto (3/3).....	80
Imagen 3.46 - Variável outletCampoDeTexto	80

Imagen 3.47 - Outlets Criados para o App	80
Imagen 3.48 - Criando a ação do botão “Ok” (1/2)	81
Imagen 3.49 - Criando a ação do botão “Ok” (2/2)	81
Imagen 3.50 - Código do método açãoBotãoOk.....	81
Imagen 3.51 - Código fonte dos outlets e ações criados até então	82
Imagen 3.52 - Linha de código do método açãoBotãoOk	82
Imagen 3.53 - Linhas de código do método açãoBotãoLimpar	83
Imagen 3.54 - Executando a versão final do app “Olá Mundo”	83

LISTA DE TABELAS

Tabela 1 - Relação de Dados Armazenados Localmente	40
Tabela 2 - Relação das calculadoras mais populares e suas avaliações	51
Tabela 3 - Relação do Espaço em Disco das Calculadoras.....	52
Tabela 4 - Relação de Funcionalidades das Calculadores	52
Tabela 5 - Relação de Tempo de Carregamento das Calculadoras	53
Tabela 6 - Relação de Características Rentáveis das Calculadoras.....	54

SUMÁRIO

1. Introdução	15
1.1 Apresentação.....	15
1.2 Formulação do Problema	15
1.3 Justificativa sobre o desenvolvimento da calculadora.....	16
1.4 Objetivos	16
1.4.1 Objetivo Geral.....	16
1.4.2 Objetivos Específicos	16
1.5 Delimitação do Escopo.....	16
1.6 O mercado.....	17
1.7 Recursos Utilizados	17
1.7.1 Hardware.....	17
1.7.2 Software	17
2. Referencial Teórico	18
3. A Linguagem de Programação Swift	19
3.1 Sintaxe (Swift 2.2).....	19
3.2 Novidades na Linguagem Swift	25
3.2.1 Remoção dos operadores ++ e --.....	25
3.2.2 Portabilidade.....	25
3.2.3 Estabilização da Interface Binária de Aplicação (IBA).....	25
4. A Calculadora Swift.....	26
4.1 Análise de Requisitos	26
4.1.1 Funcionais	26
4.1.2 Não Funcionais.....	27
4.2 Interface da Calculadora.....	27

4.2.1 Tela Principal	27
4.2.2 Tela de Histórico.....	29
4.2.3 Menu de Opções.....	29
4.2.4 Tela de Cores	30
4.2.5 Tela de Ajuda	31
4.2.6 Tela de Mais Opções	31
4.3 Funcionamento da Calculadora.....	33
4.3.1 Entrada da Expressão Matemática.....	34
4.3.2 Filtro de Expressão Inválida.....	34
4.3.3 Filtro de Expressão Incompleta	36
4.3.4 Motor de Cálculo	36
4.3.5 Apresentação do Resultado Final.....	37
4.4 Funções Personalizadas na Classe NSExpression	37
4.5 Funções Personalizadas com Parâmetros Extras	38
4.6 Armazenamento Local	40
4.6.1 Definindo Uma Estrutura de Armazenamento.....	40
4.6.2 Salvando Dados	41
4.6.3 Recuperando um Dado Salvo	41
4.7 Detecção de Orientação de Tela e Multi-Tarefas	42
4.8 Detectando idiomas	43
4.9 Alterando o Idioma do App.....	45
4.10 Implementando o Multi-Tarefas para iPad	48
4.11 Comparando a Calculadora Swift com Outros Apps	50
4.11.1 Tamanho em Disco.....	52
4.11.2 Funcionalidade	52
4.11.3 Velocidade de Carregamento	53

4.11.4 Rentabilidade	54
4.12 Código Fonte da Calculadora Swift.....	55
5. Criando o app “Olá Mundo” Interativo	56
5.1 Construindo a Interface do App.....	56
5.2 Adicionando adaptabilidade à interface.....	71
5.3 Adicionando Funcionalidade ao App	77
REFERÊNCIAS.....	87

1. Introdução

Swift é uma nova linguagem de programação criada pela Apple em 2014, sua sintaxe simples está tornando mais fácil o desenvolvimento de *apps* para *iOS*. Este trabalho irá explicar algumas das principais características desta linguagem, demonstrar o funcionamento de um *app* de calculadora, e explicar, passo a passo, o processo de criação de um aplicativo interativo através da *IDE Xcode*.

A Calculadora a ser apresentada foi completamente escrita em *Swift* e é de código aberto. Ela possui suporte às novas funções do sistema *iOS*, como multi-tarefas. Sua interface é altamente adaptável, tornando o *app* compatível com *iPhone*, *iPod* e *iPad*.

Também serão demonstrados neste trabalho todas as etapas para a criação de um *app* básico do tipo “Olá, Mundo” para *iOS*, abordando temas essenciais com relação ao desenvolvimento de aplicativos através do *Xcode*.

1.1 Apresentação

O sistema proposto é uma calculadora científica moderna com o nome “Calculadora *Swift*”, que é compatível com qualquer dispositivo que suporte a versão 9.0 do sistema operacional *iOS*. É um aplicativo de código aberto completamente escrito em *Swift*. Suas principais características são o suporte à nova função de multi-tarefas do *iOS*, realização de cálculos em tempo real, alta capacidade de personalização e suporte à múltiplos idiomas.

1.2 Formulação do Problema

Com o passar dos anos, ficou evidente o gigante crescimento de vendas de smartphones e tablets, que na verdade são pequenos computadores pessoais, com sistemas operacionais complexos, processadores avançados e eficientes e constante suporte e incentivo das maiores empresas do mundo, como a *Apple*. Como desenvolver para a plataforma *iOS*?

Este trabalho irá responder os seguintes perguntas com relação ao desenvolvimento de aplicativos para *iOS*:

- Quais são as principais características da linguagem de programação *Swift*?
- Como o *app* Calculadora *Swift* funciona?

- Como desenvolver um aplicativo para *iOS*?
- Como desenvolver uma interface compatível com *smartphones* e *tablets*?

1.3 Justificativa sobre o desenvolvimento da calculadora

Existem diversas calculadoras para *iOS* na *App Store*, desde as mais básicas até super complexas, com funções que vão muito além das de uma calculadora convencional, porém os aplicativos gratuitos normalmente possuem anúncios e são, na maioria dos casos, de baixa qualidade. Os melhores e mais complexos *apps* certamente são pagos.

Ao contrário de *iPhones*, os *iPads* não possuem uma calculadora nativa, o que de certa forma incentiva o desenvolvimento delas para o dispositivo, já que todos os usuários que necessitem de uma calculadora, mesmo que básica, irão recorrer à aplicativos da *App Store*.

1.4 Objetivos

1.4.1 Objetivo Geral

Apresentar a linguagem de programação *Swift*, explicar o funcionamento de um *app* de calculadora e demonstrar o processo de criação de *apps* através da *IDE Xcode*.

1.4.2 Objetivos Específicos

- Demonstrar algumas características de sintaxe da linguagem de programação *Swift*;
- Explicar diversas funcionalidades da calculadora;
- Comparar o *app* Calculadora *Swift* com as principais calculadoras da *App Store*;
- Demonstrar, passo a passo, a criação de um aplicativo de interface adaptável para *iOS*.

1.5 Delimitação do Escopo

A primeira versão do aplicativo consiste em uma calculadora com todas as funções científicas essenciais. O usuário pode alterar diversas configurações, como precisão de casas decimais, idiomas e temas de cor. Seu sistema de armazenamento local guarda todas as configurações do usuário, histórico de cálculos e último estado da calculadora. Nesta versão do *app*, não serão tratados:

- Modo RPN;
- Modo programador;
- Gráficos.

1.6 O mercado

O app Calculadora *Swift* não possuí fins lucrativos e é licenciado sob a Licença Pública Geral *GNU*. Seu código fonte e documentação está disponível no *GitHub*.

Seu público alvo são usuários de *iOS*, em especial de *iPads*, que necessitem de uma calculadora básica ou científica para qualquer fim.

1.7 Recursos Utilizados

1.7.1 Hardware

- MacBook;
- iPad;

1.7.2 Software

- Sistema operacional *OS X El Capitan* (11.11);
- Linguagem de programação *Swift*;
- *Xcode* 7.3;
- *Simulator* 9.3;
- *Pages*;
- *Keynote*;
- *Adobe Photoshop CS6*;

2. Referencial Teórico

A nova linguagem de programação *Swift*, criada pela Apple, torna a criação de aplicativos mais simples e divertida. Sua sintaxe mínima, auto-descritiva e segura foi criada com o intuito de substituir linguagens mais antigas, como C e *Objective-C*.

Swift é uma nova linguagem de programação para aplicativos *iOS*, *OS X*, *watchOS* e *tvOS*. Foi construída sobre o melhor das linguagens C e *Objective-C*, sem as limitações de compatibilidade da linguagem C. *Swift* adota padrões de programação segura e adiciona funções modernas para tornar a programação fácil, mais flexível, e mais divertida. “Texto Traduzido” (APPLE INC, 2016, p. 2)

Swift é uma linguagem de código aberto, onde qualquer pessoa pode enviar idéias e contribuições ao seu projeto.

Estamos animados com este novo capítulo na história de *Swift*. Depois da Apple revelar a linguagem de programação *Swift*, seu crescimento se tornou um dos mais rápidos da história das linguagens de programação. O design de *Swift* torna simples a escrita de software rápido e seguro. Agora que *Swift* é de código aberto, você pode ajudar a criar a melhor linguagem de propósito geral disponível em todos os lugares. “Texto Traduzido” (THE SWIFT TEAM, 2016, p. 1)

Xcode é um ambiente de desenvolvimento integrado (*IDE*) disponível gratuitamente para o sistema operacional *OS X*. Nele é possível criar *apps* para os sistemas operacionais *OS X*, *iOS*, *watchOS* e *tvOS*.

A *IDE Xcode* é o centro da experiência de desenvolvimento Apple. Precisamente integrada com os frameworks *Cocoa* e *Cocoa Touch*, *Xcode* é um incrível ambiente de produção para construir *apps* para *Mac*, *iPhone*, *iPad*, *Apple Watch* e *Apple TV*. “Texto Traduzido” (APPLE, 2016, p. 1)

3. A Linguagem de Programação *Swift*

Swift é uma linguagem de programação introduzida pela *Apple* em junho de 2014. Sua sintaxe é minimalista e moderna. Ela foi criada com o intuito de substituir a linguagem de programação *Objective-C*. Segundo a *Apple*, códigos em *Swift* podem ser até 2,6 vezes mais rápidos que *Objective-C* e até 8,4 vezes mais rápido que *Python 2.7*.

Imagen 1.0 - Logotipo da linguagem Swift



Fonte: Apple (2016)

3.1 Sintaxe (*Swift 2.2*)

Uma das características da linguagem é a ausência da necessidade de utilizar ponto e vírgula ao final de cada linha ou expressão, como mostra a imagem:

Imagen 1.1 - Declaração de variável e constante em *Swift*

```
var variável = "Alguma coisa."
let constante = "Outra coisa."
print(variável + constante) // Imprime "Alguma coisa. Outra coisa."
```

Fonte: *Xcode 7.3 - Playground* (2016)

Note também que a linguagem suporta acentuação e caracteres especiais normalmente, além de suportar a criação de variáveis por inferência, ou seja, não é necessário deixar explícito o tipo na declaração.

Em um *loop* do tipo *for*, é possível percorrer vetores de forma simples através de variáveis temporárias (Variáveis que existem apenas dentro do *loop*):

Imagen 1.2 - Exemplo de um loop do tipo *for* em *Swift*

```
let idades = [22, 30, 18, 13, 44, 62, 17, 29, 54, 9]
var quantidadeDeMenores = 0
for idade in idades {
    if idade < 18 {
        quantidadeDeMenores += 1
    }
}
print(quantidadeDeMenores) // Imprime 3
```

Fonte: Xcode 7.3 - Playground (2016)

A linguagem *Swift* também possui um tratamento de erros avançado. Com uma sintaxe simples e expressiva, é possível capturar praticamente qualquer tipo de erro:

Imagen 1.3 - Tratamento de erros em *Swift*

```
func fazAlgumaCoisa() throws { } // Função que possivelmente lança um erro.
func teste() {
    do {
        try fazAlgumaCoisa()
    } catch {
        print(error) // Linha de código é executada em caso de erro.
    }
}
```

Fonte: Xcode 7.3 - Playground (2016)

Funções e métodos podem receber um número variado de argumentos através da seguinte sintaxe:

Imagen 1.4 - Função com número de argumentos variáveis

```
func somaNúmeros(números: Int...) -> Int {
    var total = 0
    for número in números {
        total += número
    }
    return total
}
somaNúmeros() // Retorna 0
somaNúmeros(20, 50, 30) // Retorna 100
```

Fonte: Xcode 7.3 - Playground (2016)

Caso o programador esteja usando o *Xcode*, os cabeçalhos de funções podem ser bastante úteis pois, se criado com a sintaxe correta, seus metadados podem servir de auxílio futuramente, apresentando informações importantes como a descrição de uma função antes mesmo de escrever ela completamente.

Imagen 1.5 - Escrevendo cabeçalhos em *Swift*

```
/**  
Diz olá para uma pessoa específica.  
- Parâmetros: Nome. (String)  
- Retorna: Nada.  
  
*/  
func dizOlá(nome: String) {  
    print("Olá, " + nome + "!")  
}  
  
dizOlá("Paula") // Imprime "Olá, Paula!"
```

Fonte: *Xcode 7.3 - Playground* (2016)

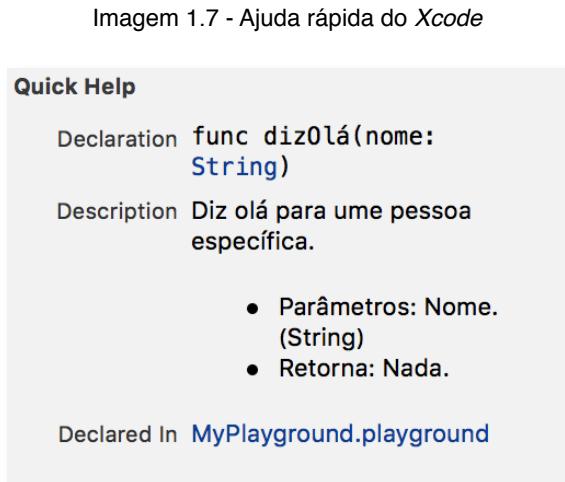
Ao escrever a função futuramente, a sugestão automática do *Xcode* irá exibir a descrição da função presente em seu cabeçalho:

Imagen 1.6 - Sugestão automática do *Xcode*



Fonte: *Xcode 7.3 - Playground* (2016)

No menu “*quick help*” do *Xcode*, é possível ver informações do cabeçalho como parâmetros, retorno e descrição:



Fonte: *Xcode 7.3 - Playground* (2016)

Valores opcionais podem conter um valor válido ou nada (nulo). Valores nulos são chamados de *nil* em *Swift*. É possível utilizar *if* e *let* juntos para trabalhar com este tipo de valor da seguinte forma:

Imagen 1.8 - *if* e *let* para valores opcionais (1/2)

```
var nomeOpcional: String? = "Maurício"

if let nome = nomeOpcional {
    print("Olá, " + nome) // Esta linha é executada!
} else {
    print("Nome opcional é nulo!")
}
```

Fonte: *Xcode 7.3 - Playground* (2016)

No exemplo acima, a variável *nomeOpcional* é declarada como *String?*, onde a interrogação indica que ela é do tipo *String* opcional. Na próxima linha ocorre uma tentativa de atribuir o valor da variável *nomeOpcional* para uma constante local chamada de *nome*, caso haja alguma *String* na variável, a comparação é verdadeira e o código dentro do *if* é executado, caso contrário, se o valor da variável opcional for *nil*, o código dentro do *else* será executado, como mostra a imagem:

Imagen 1.9 - *if* e *let* para valores opcionais (2/2)

```
var nomeOpcional: String? = nil

if let nome = nomeOpcional {
    print("Olá, " + nome)
} else {
    print("Nome opcional é nulo!") // Esta linha é executada!
}
```

Fonte: Xcode 7.3 - Playground (2016)

Outra característica importante da linguagem de programação *Swift* são os dicionários, através deles é possível criar associações entre diferentes elementos:

Imagen 1.10 - Exemplo de um dicionário do tipo [String:String] em *Swift*

```
var dicionarioCachorro : [String:String] = [
    "Português" : "Cachorro",
    "Inglês" : "Dog",
    "Espanhol" : "Perro"
]

print(dicionarioCachorro["Português"]) // Imprime "Cachorro".
print(dicionarioCachorro["Inglês"]) // Imprime "Dog".
print(dicionarioCachorro["Espanhol"]) // Imprime "Perro".
```

Fonte: Xcode 7.3 - Playground (2016)

Dicionários também podem assumir um papel semelhante ao de *Structs*, onde diferentes tipos de dados são armazenados em uma única variável ou constante:

Imagen 1.11 - Exemplo de um dicionário do tipo [String:AnyObject] em *Swift*

```
var dicionarioDeThiago : [String:AnyObject] = [
    "Nome" : "Thiago",
    "Idade" : 21,
    "Altura" : 1.84,
    "Estudando" : true
]

dicionarioDeThiago["Idade"] = 22
print(dicionarioDeThiago) // Imprime "["Altura": 1.84, "Nome": Thiago, "Idade": 22, "Estudando": true]"
```

Fonte: Xcode 7.3 - Playground (2016)

A combinação da simplicidade da linguagem *Swift*, com o amplo suporte a caracteres *Unicode* pode gerar códigos interessantes:

Imagen 1.12 - Uso de caracteres *Unicode* em *Swift*

```
let frutas = ["🍏", "🍉", "🍍", "🍓"]
let ❤️ = "Coração Partido"

let dicionárioDeBandeiras: [String:String] = [
    "Brasil" : "🇧🇷",
    "Estados Unidos" : "🇺🇸",
    "Portugal" : "🇵🇹"
]

print( frutas[3] ) // Imprime "🍓".
print(❤️) // Imprime "Coração Partido".
print( dicionárioDeBandeiras["Brasil"] ) // Imprime "🇧🇷".
```

Fonte: Xcode 7.3 - Playground (2016)

Loops do tipo *for* são um pouco diferentes em *Swift* se comparados com outras linguagens, sua sintaxe é a seguinte:

Imagen 1.13 - Exemplo do loop *for* em *Swift*

```
var total01 = 0

for contador in 0...10 {
    total01 += 1
}

print( total01 ) // Imprime 11


var total02 = 0

for contador in 0...10 {
    total02 += contador
}

print( total02 ) // Imprime 55
```

Fonte: Xcode 7.3 - Playground (2016)

O valor contador é uma constante que irá percorrer todo o índice “0...10”, apesar de seu valor mudar a cada repetição de acordo com o índice, não é possível alterar seu valor (apenas leitura). A constante é destruída após o *loop*.

3.2 Novidades na Linguagem *Swift*

A linguagem de programação *Swift* é de código aberto. Com contribuições de especialistas do mundo inteiro, seu crescimento está sendo um dos maiores da história das linguagens de programação. Sua primeira versão, 1.0, foi lançada em junho de 2014, atualmente (maio de 2016) a linguagem *Swift* já se encontra na versão 2.2.

Ao final de 2016 sua versão 3.0 será lançada, com diversas novidades importantes, entre elas:

3.2.1 Remoção dos operadores ++ e --

Segundo a Apple, a vantagem na velocidade de escrita de uma expressão “x++” não é muito grande em relação à “x += 1”, além de ser menos expressiva.

3.2.2 Portabilidade

Outras plataformas se tornarão compatíveis com *Swift*.

3.2.3 Estabilização da Interface Binária de Aplicação (IBA)

Com a estabilização da IBA da linguagem *Swift*, aplicativos poderão compartilhar mais bibliotecas e componentes já utilizados pelo sistema operacional, o que pode reduzir o tamanho de *apps* escritos em *Swift*.

4. A Calculadora Swift

Calculadora *Swift* é um aplicativo para iOS compatível com iPhone, iPod e iPad. Seu código fonte é aberto e completamente escrito em Swift.

Imagen 2.0 - Ícone do app Calculadora *Swift*



Fonte: Elaborado pelo autor (2016)

4.1 Análise de Requisitos

4.1.1 Funcionais

O aplicativo deve:

- Suportar múltiplas orientações de tela.
- Suportar Multi-Tarefas para *iPad*.
- Possuir todas as principais funções de uma calculadora científica padrão.
- Realizar cálculos em tempo real, sem a necessidade de um botão de resultado.
- Impedir que o usuário digite operações inválidas.
- Ter precisão ajustável de até 15 casas decimais.
- Possuir uma tela de ajuda.
- Possuir um botão de último resultado.
- Possuir todas as funções clássicas de memória.
- Permitir que o usuário seja capaz de copiar o resultado final.
- Salvar seu estado quando for encerrado.
- Possuir 20 temas de cores.
- Permitir que o usuário seja capaz de ligar ou desligar o som das teclas.
- Armazenar todas as configurações e histórico de cálculos localmente.
- Permitir que o usuário seja capaz de apagar o histórico de cálculos.
- Possuir suporte à quatro idiomas: português, inglês, espanhol e mandarim.

4.1.2 Não Funcionais

- O aplicativo deverá rodar em qualquer sistema que suporte o sistema *iOS 9*.
- O código fonte deverá ser único para todos os dispositivos.
- A calculadora deve ocupar espaço em disco de no máximo 30MB.
- O aplicativo deverá ser licenciado sob a Licença Pública Geral *GNU*, sendo de código aberto e sem fins lucrativos.
- O código fonte e toda a documentação deverá estar disponível no *GitHub*.
- O código fonte e documentação da versão 1.0 do *app* deverá ser completamente escrito e comentado em português brasileiro.
- Qualquer pessoa poderá contribuir e fazer parte do desenvolvimento da Calculadora *Swift*.

4.2 Interface da Calculadora

A Calculadora *Swift* é um *app* compatível com múltiplos dispositivos, por questões de praticidade as interfaces demonstradas neste capítulo serão todas relativas ao *iPad* com resolução de 2048x1536 pixels.

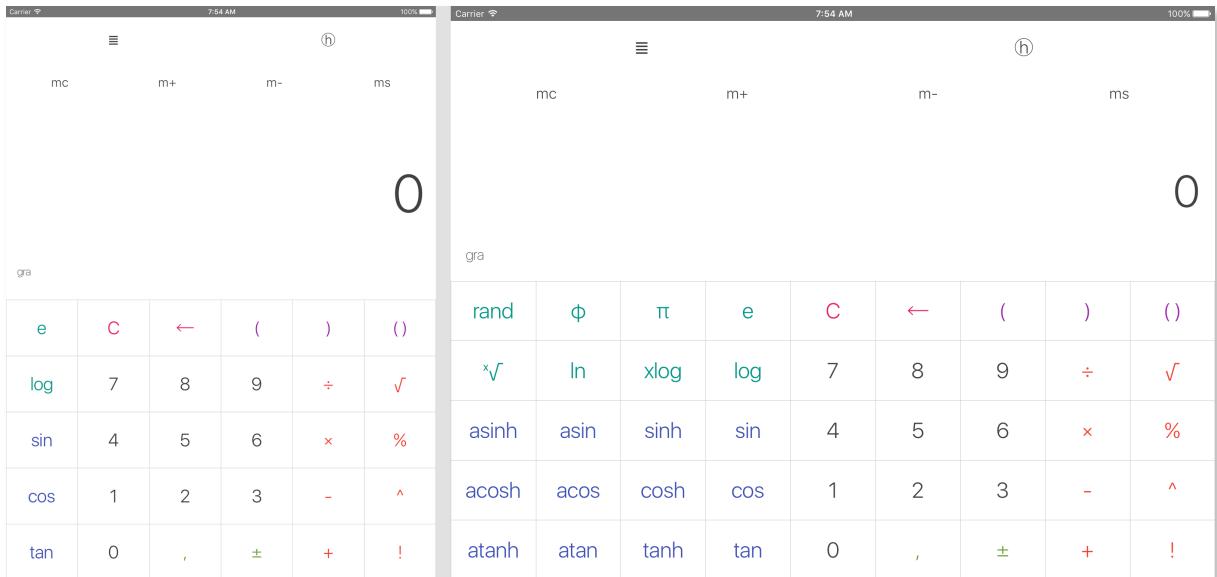
A estrutura de interface do usuário é relativamente simples, consistindo de uma tela de carregamento e tela principal além dos menus de histórico, configurações, temas de cores, ajuda e menu de mais opções.

4.2.1 Tela Principal

A tela principal do aplicativo é a calculadora em si. É nela que todos os botões e menus são acessíveis ao usuário.

Sua interface é altamente adaptável, se ajustando automaticamente a diferentes tamanhos e posições de tela:

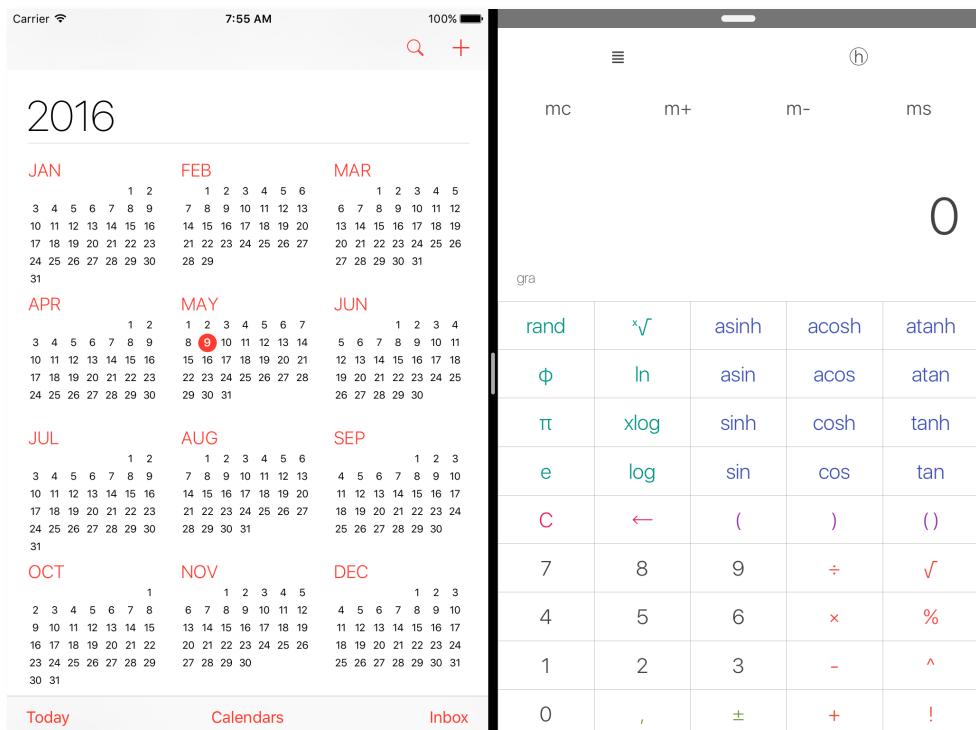
Imagen 2.1 - Tela Principal (Vertical e Horizontal)



Fonte: Simulator (iOS 9.3) - Calculadora Swift (2016)

Em iPads com suporte a função multi-tarefas do iOS, a calculadora adapta sua interface de forma especial, mantendo sua funcionalidade total em um espaço de tela menor.

Imagen 2.2 - Tela Principal (Multi-Tarefas)

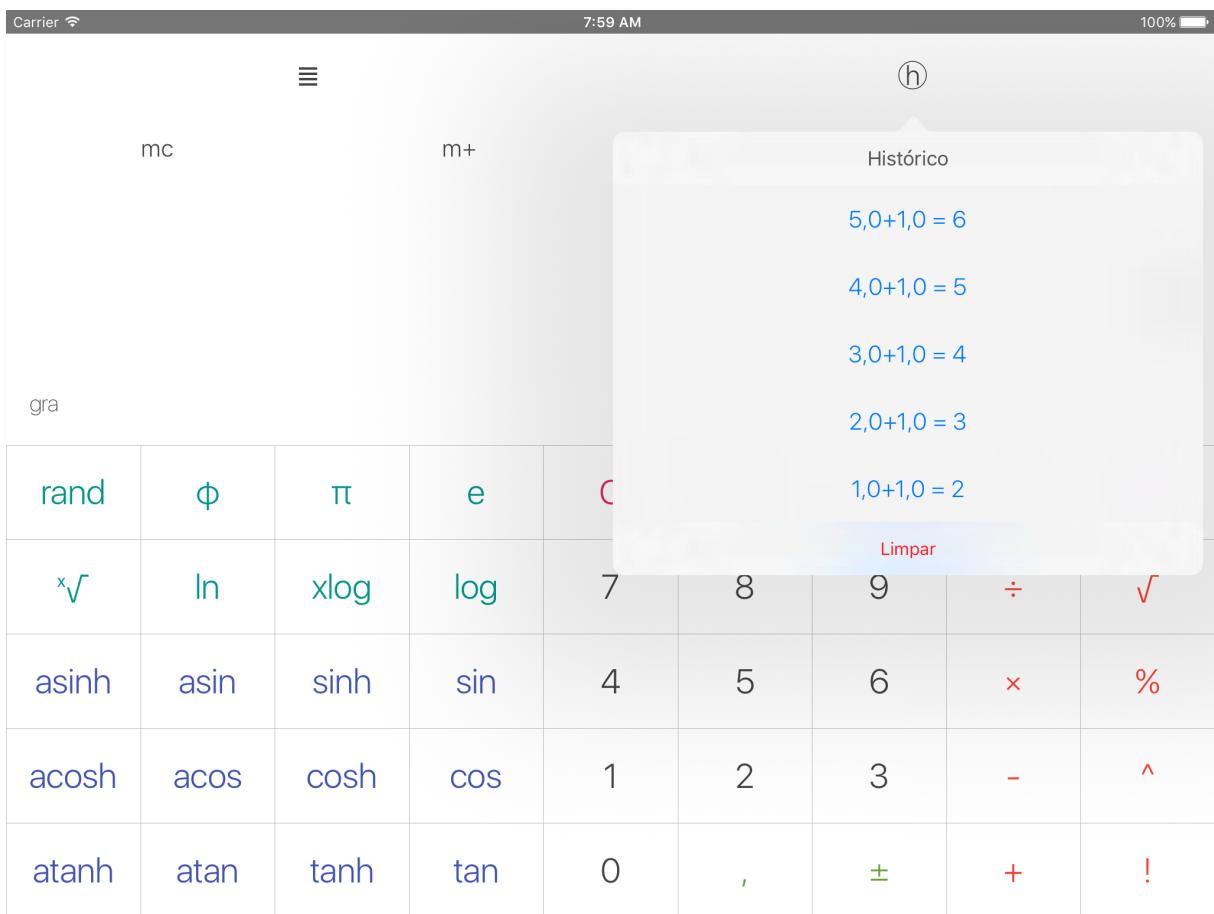


Fonte: Simulator (iOS 9.3) - Calendário e Calculadora Swift (2016)

4.2.2 Tela de Histórico

A tela de histórico é um menu do estilo *pop-over*, quando o usuário toca no botão de histórico, uma pequena interface sobrepõe a tela principal. Em iPads, seu tamanho é de apenas 500x375 pixels, porém ela é baseada em *scroll-view*, ou seja, o usuário pode deslizar o dedo para navegar na lista de históricos da calculadora:

Imagen 2.3 - Tela de Histórico

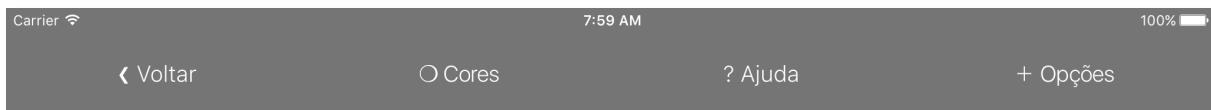


Fonte: *Simulator (iOS 9.3) - Calculadora Swift (2016)*

4.2.3 Menu de Opções

O menu de opções é basicamente uma pequena interface que disponibiliza os botões para os outros menus da calculadora:

Imagen 2.4 - Menu de Opções



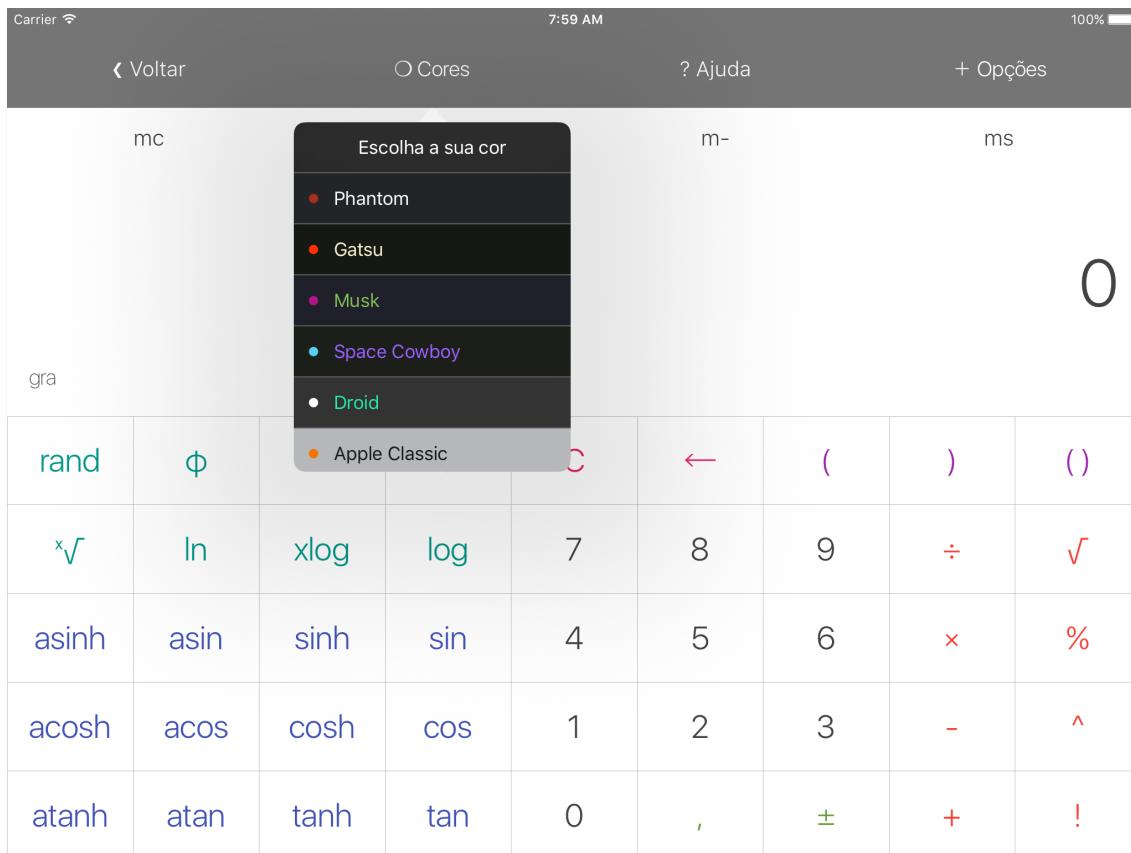
Fonte: *Simulator (iOS 9.3) - Calculadora Swift (2016)*

Quatro botões são apresentados ao usuário: Voltar, Cores, Ajuda e Opções Extras. A distância entre os itens é relativa e adaptável.

4.2.4 Tela de Cores

A tela de cores, assim como a tela de histórico, é um menu do estilo *pop-over* com *scroll-view*. Sua função é disponibilizar diversos temas de cores para a interface da calculadora.

Imagen 2.5 - Tela de Cores

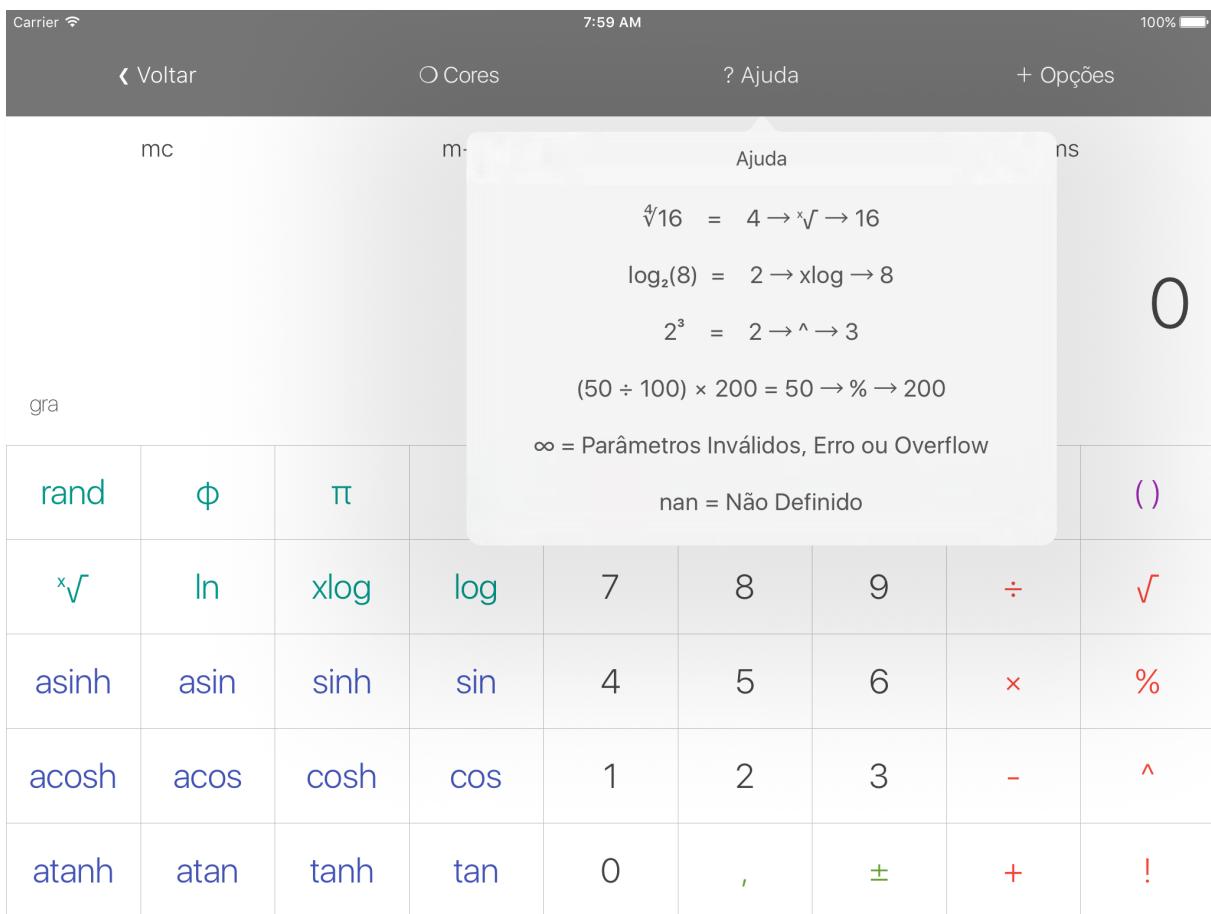


Fonte: *Simulator (iOS 9.3) - Calculadora Swift (2016)*

4.2.5 Tela de Ajuda

A tela de ajuda é uma interface simples com o objetivo de ajudar um usuário que possivelmente tenha alguma dúvida sobre o funcionamento da calculadora. Ela explica o funcionamento de alguns botões e o significado de algumas mensagens, entre outras características do *app*.

Imagen 2.6 - Tela de Ajuda

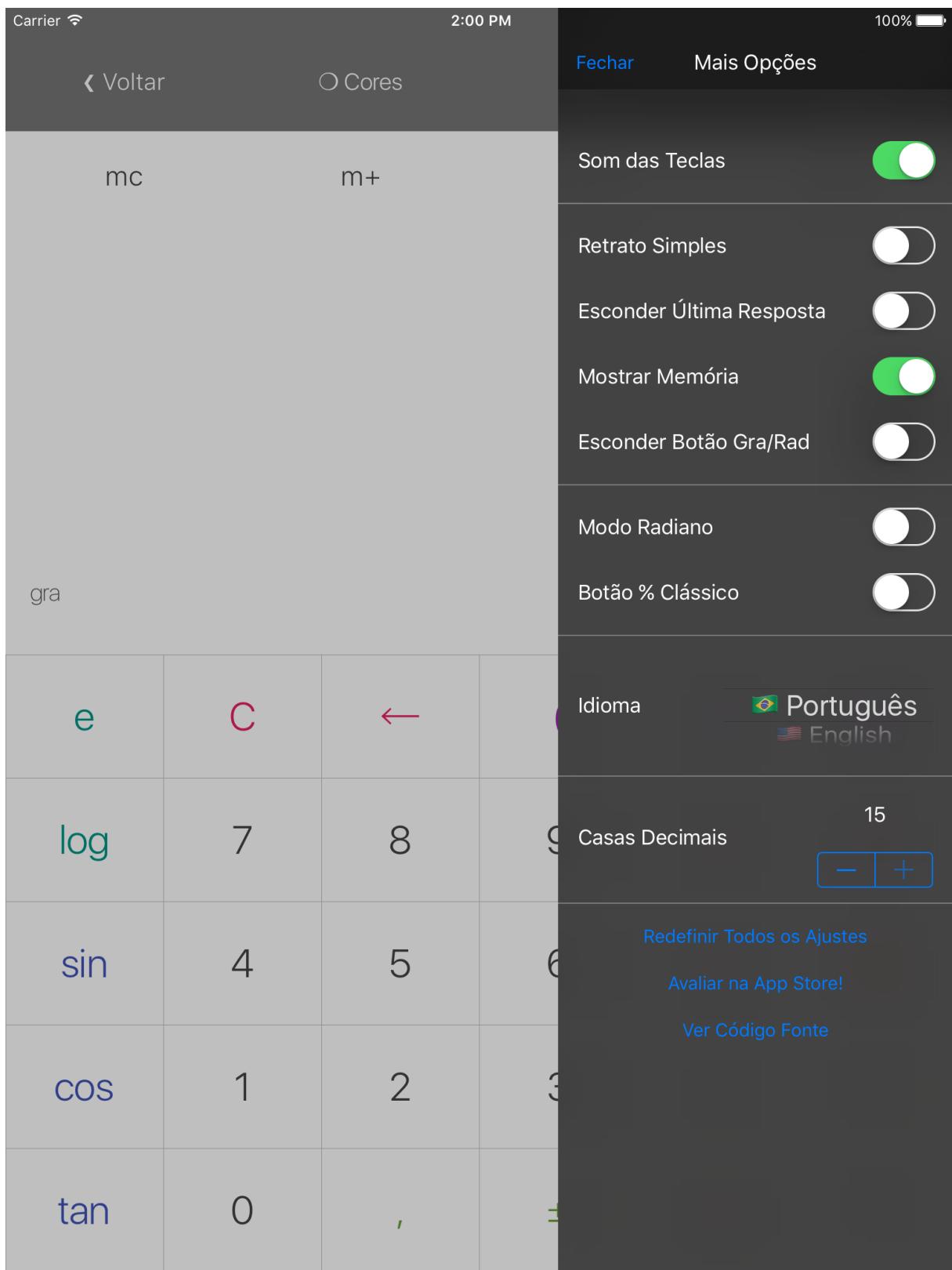


Fonte: *Simulator (iOS 9.3) - Calculadora Swift (2016)*

4.2.6 Tela de Mais Opções

Todas as opções mais avançadas estão no menu de mais opções da calculadora. Nela o usuário pode alterar algumas características da interface ou até mesmo a linguagem do *app*. A partir dela o usuário também tem a opção de redefinir todos os ajustes, avaliar o *app* e visitar a página do *GitHub* que contém seu código fonte.

Imagen 2.7 - Tela de Mais Opções



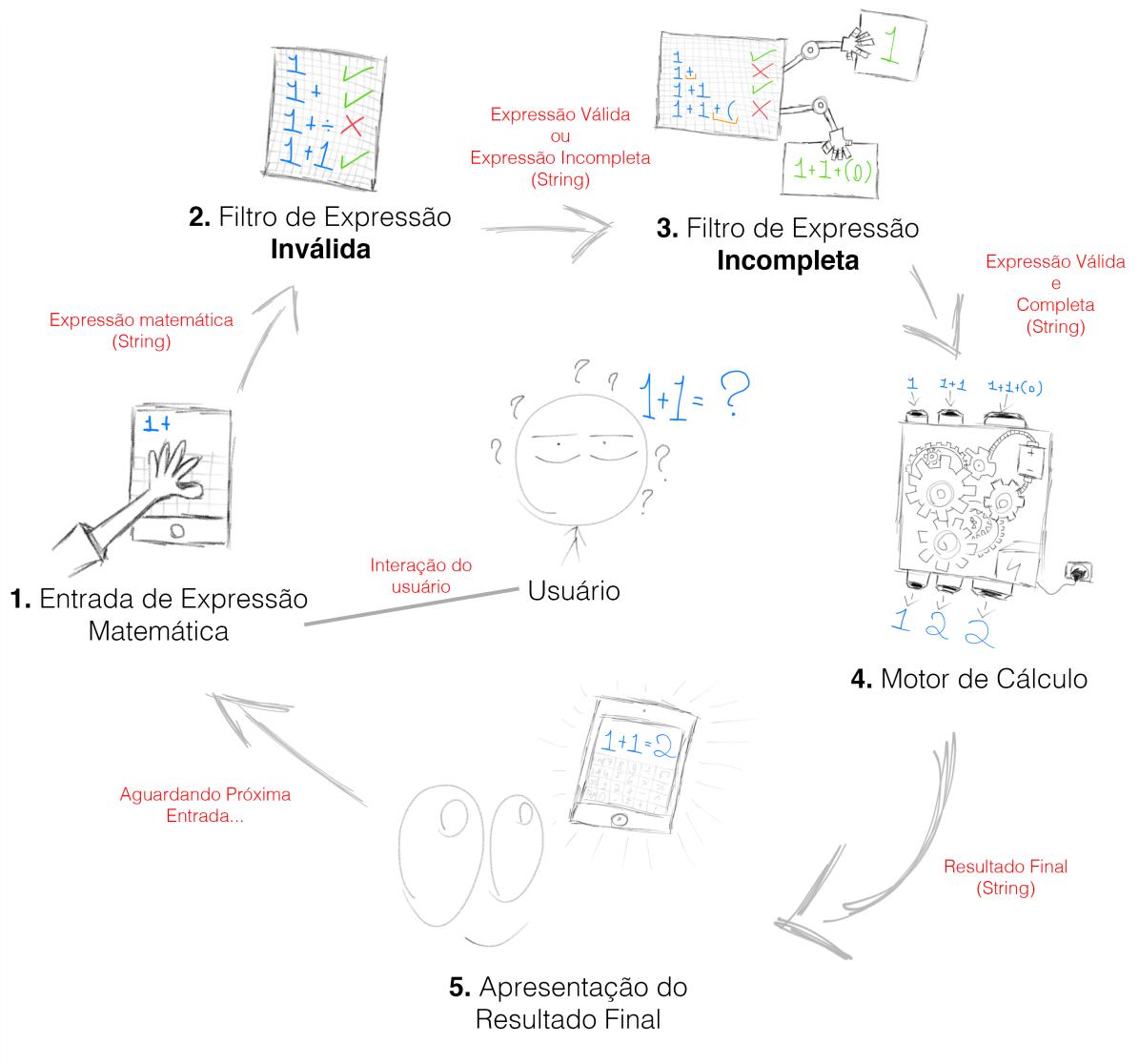
Fonte: *Simulator (iOS 9.3)* - Calculadora *Swift* (2016)

4.3 Funcionamento da Calculadora

O funcionamento interno da Calculadora *Swift* possui muitas diferenças se comparada a uma calculadora convencional. Para atingir o objetivo de resolver operações em tempo real, foi necessário implementar funções preparadas para verificar, corrigir e resolver expressões matemáticas constantemente.

O diagrama a seguir mostra uma visão geral do ciclo de funcionamento da Calculadora *Swift*, que pode ser dividido em 5 etapas:

Imagen 2.8 - Ciclo de Funcionamento da Calculadora *Swift*



Fonte: Elaborado pelo autor (2016)

4.3.1 Entrada da Expressão Matemática

O primeiro passo da calculadora é reagir à interação do usuário, primeiramente é necessário detectar qual botão foi tocado. Com excessão de dois botões (Limpar e Apagar), todos eles quando tocados chamam um único método chamado *recebeEntradasDoUsuário*, ele recebe um objeto do tipo *UIButton* como parâmetro, este objeto contém as informações do botão que foi tocado.

Imagen 2.9 - Código do Método *recebeEntradasDoUsuário*

```
/*
    Recebe e direciona a entrada do usuário.
    - Parâmetros: sender. (UIButton)
    - Retorna: Nada.

*/
@IBAction func recebeEntradasDoUsuario(sender: UIButton) {
    reproduzSomDeTecla()
    filtraExpressaoMatematicaInvalida(sender.titleLabel!.text!)
    mostraResultadoFinal()
}
```

Fonte: Código fonte da Calculadora *Swift / Xcode 7.3* (2016)

Sua primeira ação é chamar o método *reproduzSomDeTecla*, que reproduz o som padrão do teclado *iOS*. Ele é o primeiro método para garantir que o usuário tenha um retorno rápido antes mesmo de iniciar o processamento de uma expressão matemática.

A segunda ação é chamar o método *filtraExpressãoMatemáticaInválida*, enviando uma *String* como parâmetro. A *String* em questão é o texto do botão tocado, pode ser “1”, “+”, “log”, etc. Este segundo método é muito importante para o funcionamento da calculadora, pois ele garante que a expressão matemática a ser resolvida é válida.

4.3.2 Filtro de Expressão Inválida

O método *filtraExpressãoMatemáticaInválida* verifica a entrada que o usuário está tentando incluir na expressão matemática, caso ela seja válida, ela é incluída, caso contrário, a entrada é ignorada (não é adicionada à expressão atual).

Imagen 2.10 - Parte do código do Método *filtreExpressãoMatemáticaInválida*

```
/**
Aceita ou ignora uma entrada do usuário.
- Parâmetros: Entrada mais recente do usuário.
- Retorna: Nada.

*/
func filtraExpressaoMatematicaInvalida(novaEntrada: String) {
    var impedeAtualizacaoDaLinhaDoTempo = false

    // Entrada é número:
    if caractereENumerico(novaEntrada.characters.last) == true {
        if outletLabelExpressaoMatematica.text!.characters.last != ")" && outletLabelExpressaoMatematica.text!.characters.last != "!" {
            outletLabelExpressaoMatematica.text = outletLabelExpressaoMatematica.text! + novaEntrada
        }
    }

    // Entrada não é número:
} else {
    if novaEntrada == "(" {
        if caractereENumerico(outletLabelExpressaoMatematica.text!.characters.last) == false &&
            outletLabelExpressaoMatematica.text!.characters.last != "," &&
            outletLabelExpressaoMatematica.text!.characters.last != ")" &&
            outletLabelExpressaoMatematica.text!.characters.last != "!" {
            outletLabelExpressaoMatematica.text = outletLabelExpressaoMatematica.text! + novaEntrada
        }
    } else if novaEntrada == ")" {
```

Fonte: Código fonte da Calculadora Swift / Xcode 7.3 (2016)

A terceira ação do método *recebeEntradasDoUsuário* é invocar o método *mostraResultadoFinal*, que primeiramente verifica se há alguma expressão matemática a ser resolvida, caso negativo, nenhum cálculo é realizado e o número zero é apresentado ao usuário. Caso haja uma expressão, o método *preparaEResolveExpressãoMatemática* é chamado.

Imagen 2.11 - Código do Método *mostraResultadoFinal*

```
/**
Mostra o Resultado Final.
- Parâmetros: Nenhum.
- Retorna: Nada.

*/
func mostraResultadoFinal() {
    // Caso a expressão matemática esteja vazia, o Resultado Final é Zero:
    if outletLabelExpressaoMatematica.text!.isEmpty == true {
        outletLabelResultadoFinal.text = "0"
    }

    // Caso contrário, a expressão matemática deve ser resolvida:
} else {
    outletLabelResultadoFinal.text = preparaEResolveExpressaoMatematica(outletLabelExpressaoMatematica.text!)
}
```

Fonte: Código fonte da Calculadora Swift / Xcode 7.3 (2016)

4.3.3 Filtro de Expressão Incompleta

A primeira ação do método `preparaEResolveExpressãoMatemática` chamar o método `garanteExpressãoMatemáticaFormatadaECompleta`, é nele que expressões incompletas são convertidas em expressões completas, entre outras preparações.

Imagen 2.12 - Código do método `preparaEResolveExpressãoMatemática`

```
/**
Prepara a operação matemática e obtém o resultado final.

- Parâmetros: Expressão Matemática (String).
- Retorna: Resultado Final (String).

*/
func preparaEResolveExpressaoMatematica(operacao: String) -> String {
    let operacaoValidaECompleta = garanteExpressaoMatematicaFormatadaECompleta(operacao)
    var resultadoFinal = tentaResolverOperacaoMatematicaValida(operacaoValidaECompleta)
    resultadoFinal = formataResultadoFinal(resultadoFinal)
    return resultadoFinal
}
```

Fonte: Código fonte da Calculadora *Swift / Xcode 7.3 (2016)*

4.3.4 Motor de Cálculo

Agora que a expressão matemática finalmente é válida e completa, é possível resolver-la sem problemas. O método `tentaResolverExpressãoMatemáticaValida` é chamado, ele é a primeira parte do motor de cálculo da calculadora:

Imagen 2.13 - Código do método `tentaResolverExpressãoMatemáticaVálida`

```
/**
Tenta resolver uma expressão matemática válida, retornando uma mensagem de erro em caso de falha.

- Parâmetros: Expressão Matemática Válida e Completa (String).
- Retorna: Resultado (String).

*/
func tentaResolverExpressaoMatematicaValida(expressaoValidaECompleta: String) -> String {
    var resultado: Double
    do {
        resultado = try resolveExpressaoMatematicaValidaECompleta(expressaoValidaECompleta)
    } catch {
        return dicionarioMensagemDeErro[idioma]!
    }
    return String(resultado)
}
```

Fonte: Código fonte da Calculadora *Swift / Xcode 7.3 (2016)*

Como o método `resolveExpressãoMatemáticaVálida` pode retornar uma excessão, usa-se a sintaxe de captura de erros padrão da linguagem *Swift*. Depois de criar uma variável do tipo `Double`, tenta-se resolver a expressão matemática. Caso por algum motivo desconhecido não se consiga resolver-la, o escopo do `catch` será executado, retornando uma mensagem de erro ao usuário.

O método `resolveExpressãoMatemáticaValidaECompleta` é responsável por finalmente resolver a expressão matemática, podendo porém, retornar uma exceção:

Imagem 2.14 - Código do Método `resolveExpressãoMatemáticaValidaECompleta`

```
/**  
 * Retorna o resultado final ou um excessão.  
 * - Parâmetros: Expressão Matemática (String).  
 * - Retorna: Resultado Final (Double).  
 */  
func resolveExpressãoMatemáticaValidaECompleta(expressãoValidaECompleta: String) throws -> Double {  
    return NSExpression(format: expressãoValidaECompleta).expressionValueWithObject(nil, context: nil) as! Double  
}
```

Fonte: Código fonte da Calculadora *Swift / Xcode 7.3* (2016)

4.3.5 Apresentação do Resultado Final

Após obter sucesso ao resolver uma expressão, o resultado final volta como um valor `Double` para o método `tentaResolverExpressãoMatemáticaVálida`, que converte ele em uma `String` e envia-o de volta ao método `preparaEResolveExpressãoMatemática`, onde recebe uma formatação final para que possa ser apresentado ao usuário.

É aqui que a função do motor de cálculo termina, e a calculadora apenas aguarda a próxima entrada o usuário. Todo este processo descrito ocorre em poucos milissegundos sempre que o usuário toca em algum botão do teclado.

4.4 Funções Personalizadas na Classe `NSExpression`

A classe `NSExpression`, utilizada para resolver uma expressão matemática, nativamente não suporta todas as funções matemáticas, como funções trigonométricas, por exemplo. Para resolver este problema, é necessário criar extensões do tipo `NSNumber` que contenham as funções que faltam. A função que calcula o cosseno em graus, por exemplo, é declarada globalmente da seguinte forma:

Imagen 2.15 - Código da extensão *NSNumber* e do método *calculaCossenoGrau*

```
public extension NSNumber {
    /**
     Calcula o cosseno em graus de um valor.
     - Parâmetros: Nenhum.
     - Retorna: Cosseno em graus. (NSNumber)

    */
    func calculaCossenoGrau() -> NSNumber {
        return __cospi(self.doubleValue/180.0)
    }
}
```

Fonte: Código fonte da Calculadora *Swift / Xcode 7.3* (2016)

A sintaxe da expressão matemática para calcular o cosseno em graus de 42 utilizando a função personalizada *calculaCossenoGrau* é a seguinte:

Imagen 2.16 - Exemplo de sintaxe da função *calculaCossenoGrau*

```
let exemploDeExpressãoMatemática = "function(42.0, 'calculaCossenoGrau')"
```

Fonte: *Xcode 7.3* (2016)

A *String* “`function(42.0, 'calculaCossenoGrau')`” é reconhecida pela classe *NSExpression* caso a extensão *NSNumber* com a função *cossenoGrau* seja devidamente declarada. Um dos desafios do app Calculadora *Swift* é formatar expressões personalizadas corretamente. Quando, por exemplo, o usuário digita a expressão matemática “`cos(2+3)`”, ela deve ser convertida para “`function(2+3, 'calculaCossenoGrau')`” antes de ser resolvida.

4.5 Funções Personalizadas com Parâmetros Extras

Também é possível criar e utilizar funções que possuam diversos parâmetros. Para isto se cria uma extensão *NSNumber* com a função que possua um ou mais parâmetros. A principal mudança está na sintaxe da expressão matemática, como mostra o exemplo do uso da função de raiz enésima do *app*:

Imagen 2.17 - Exemplo de sintaxe da função *calculaRaizEnésima*

```
let exemploDeExpressãoMatemática = "function(27.0, 'calculaRaizEnésima:', 3.0)"
```

Fonte: Xcode 7.3 (2016)

Imagen 2.18 - Código da função *calculaRaizEnésima*

```
/**  
Calcula a raiz enésima de um radicando.  
– Parâmetros: Índice. (NSNumber)  
– Retorna: Raíz enésima do radicando. (NSNumber)  
*/  
func calculaRaizEnesima(indiceNSNumber: NSNumber) -> NSNumber {  
    let indice = Double(indiceNSNumber)  
    let radicando = Double(self.doubleValue)  
  
    if indice % 2 == 0 {  
        return pow(radicando, 1.0/indice)  
    } else {  
        if radicando.isSignMinus == true {  
            return (pow(radicando * -1, 1.0/indice)) * -1.0  
        } else {  
            return pow(radicando, 1.0/indice)  
        }  
    }  
}
```

Fonte: Código fonte da Calculadora *Swift* / Xcode 7.3 (2016)

Note que a sintaxe é um pouco diferente de uma função sem parâmetros. O uso de dois pontos após o nome da função é obrigatório na expressão. O parâmetro *índiceNSNumber* recebe o valor 3.0 da expressão.

Para utilizar a função de raiz enésima na Calculadora *Swift*, o usuário deve primeiramente digitar o índice, depois deve tocar no botão de raiz enésima e por fim digitar o radicando.

Para calcular a raiz cúbica de 27, por exemplo, o usuário irá digitar o número “3”, depois pressionar o botão de raiz enésima e digitar “27”, formando a *String* “3nth_root(27)” na tela da calculadora. Esta *String* deve ser convertida para “function(27.0, ‘calculaRaizEnésima:’, 3.0)” antes de ser resolvida.

4.6 Armazenamento Local

O Armazenamento local de dados da Calculadora *Swift* consiste em uma *String* e três dicionários, como mostra a tabela a seguir:

Tabela 1 - Relação de Dados Armazenados Localmente

Nome	Tipo	Conteúdo
Tema de Cor	<i>String</i>	Nome do último tema de cor escolhido pelo usuário
Último Estado da Calculadora	[<i>String</i> : <i>AnyObject</i>]	Expressão matemática, linha do tempo da expressão, valor da memória e último resultado
Histórico de Cálculos e Linhas do Tempo	[<i>String</i> : <i>AnyObject</i>]	Histórico de cálculos e suas linhas do tempo
Configurações do Menu de Mais Opções	[<i>String</i> : <i>AnyObject</i>]	Todas as configurações do menu de mais opções.

Fonte: Elaborado pelo autor (2016)

Existem diversas formas de armazenar dados locais no iOS, a técnica utilizada neste aplicativo se baseia no método *NSUserDefaults.standardUserDefaults()*.

4.6.1 Definindo Uma Estrutura de Armazenamento

Primeiro é necessário criar uma estrutura que irá conter as chaves para acessar os dados armazenados localmente:

Imagem 2.19 - Estrutura de Chaves para Armazenamento Local

```
//  
// Chaves para o armazenamento local:  
//  
struct chavesDeArmazenamentoLocal {  
  
    static let chaveCorDaCalculadora = "Tema de Cor"  
    static let chaveUltimoEstadoCalculadora = "Último Estado da Calculadora"  
    static let chaveHistoricoELinhasDoTempo = "Histórico e Linhas do Tempo"  
    static let chaveConfiguracoesUsuario = "Configurações do Menu de Mais Opções"  
}
```

Fonte: Código fonte da Calculadora *Swift* / Xcode 7.3 (2016)

Todas as chaves devem estar associadas com diferente *Strings*, independente o tipo de dado a ser armazenado.

4.6.2 Salvando Dados

Com a *Struct* devidamente declarada, já é possível salvar dados localmente. O método *armazenaCorDaCalculadora* é responsável por armazenar a *String* que contém o tema de cor.

O método recebe como parâmetro o dado a ser armazenado (neste caso, uma *String*). Sua primeira linha de código cria um objeto *standardUserDefaults*, este objeto possui acesso a todos os dados armazenados localmente.

A segunda linha é responsável por salvar o dado, para isso é necessário relacionar ele com uma das chaves já definidas pela estrutura *chavesDeArmazenamentoLocal*, neste caso, a *chaveCorDaCalculadora*. Após esta linha, o dado foi armazenado localmente com sucesso:

Imagen 2.20 - Código do método *armazenaCorDaCalculadora*

```
/**
 * Armazena localmente o nome do tema de cor da calculadora.
 * - Parâmetros: Nome do Tema. (String)
 * - Retorna: Nada.
 */
func armazenaCorDaCalculadora(corDaCalculadora: String) {
    let chaveDeArmazenamento = UserDefaults.standardUserDefaults()
    chaveDeArmazenamento.setValue(corDaCalculadora, forKey: chavesDeArmazenamentoLocal.chaveCorDaCalculadora)
}
```

Fonte: Código fonte da Calculadora *Swift / Xcode 7.3 (2016)*

4.6.3 Recuperando um Dado Salvo

Para carregar um dado já salvo, o processo é muito semelhante. Cria-se o mesmo objeto com acesso aos dados locais, mas agora é necessário tentar recuperar um dado com sua respectiva chave:

Imagen 2.21 - Código do método *carregaERetornaTemaDaCalculadora*

```
/*
Carrega o tema de cor.

- Parâmetros: Nenhum.
- Retorna: Tema de cor. (String)

*/
func carregaERetornaTemaDaCalculadora() -> String {
    let chaveDeArmazenamento = UserDefaults.standardUserDefaults()

    // Caso já exista um tema salvo...
    if let corCalculadora = chaveDeArmazenamento.stringForKey(chavesDeArmazenamentoLocal.chaveCorDaCalculadora) {
        return corCalculadora
    } // ... Caso contrário:
    else {
        return nomeTemaDeCorPadrao
    }
}
```

Fonte: Código fonte da Calculadora *Swift / Xcode 7.3 (2016)*

É importante tratar a possibilidade de não existir dado algum localmente, no caso da função acima, usa-se um *if let* que tenta atribuir o dado de uma chave à constante *corCalculadora*, caso esta operação não obtenha sucesso, o escopo do *else* será executado, retornando o nome do tema de cor padrão.

4.7 Detecção de Orientação de Tela e Multi-Tarefas

É sempre importante saber quando o usuário muda a orientação da tela ou entra em modo multi-tarefas, afinal, é necessário adaptar a interface.

Para detectar mudanças de orientação, é necessário pedir ao sistema operacional para que ele mande uma notificação ao aplicativo sempre que o usuário girar o dispositivo. A classe responsável por estas notificações é a *NSNotificationCenter*.

É possível chamar um método sempre que uma notificação de orientação for recebida. Este método deverá ser responsável pelas mudanças na interface.

No app Calculadora *Swift*, o objeto detector de orientação é criado logo em sua inicialização, ou seja, no método *viewDidLoad*. Sua declaração tem a seguinte sintaxe:

Imagen 2.22 - Criação de uma notificação de orientação de tela

```
// Cria uma notificação de orientação de tela:
NSNotificationCenter.defaultCenter().addObserver(self, selector: #selector(ViewController.
acaoMudancaDeOrientacao), name: UIDeviceOrientationDidChangeNotification, object: nil)
```

Fonte: Código fonte da Calculadora *Swift / Xcode 7.3 (2016)*

A função a ser ativada sempre que uma notificação for recebida é definida no parâmetro *selector*, e tem a seguinte sintaxe:

Imagen 2.23 - Sintaxe de um seletor

```
selector: #selector(ViewController.acaoMudancaDeOrientacao)
```

Fonte: Código fonte da Calculadora *Swift / Xcode 7.3* (2016)

ViewController se refere à classe, e *açãoMudançaDeOrientação* ao método a ser ativado sempre que o usuário mudar a orientação de seu dispositivo. Este método realiza os tratamentos de interface necessários, como esconder ou mostrar alguns botões.

4.8 Detectando idiomas

Na versão 1.0 da Calculadora *Swift*, o app já conta com suporte total à quatro idiomas, que são: Português Brasileiro, Inglês Americano, Espanhol e Mandarim. Em versões posteriores pretende-se adicionar mais idiomas, já que o processo é muito simples e basicamente envolve adicionar valores aos dicionários já existentes no aplicativo.

Na primeira vez que o aplicativo é executado em um dispositivo, as configurações do usuário iniciam-se com seus valores padrões:

Imagen 2.24 - Configurações iniciais

```
let dicionarioOpcoesPadraoDoUsuario: [String:AnyObject] = [
    "Som das Teclas Ativado" : true,
    "Retrato Simples Ativado" : false,
    "Esconder Última Resposta Ativado" : false,
    "Mostrar Memória Ativado" : true,
    "Esconder Botão Gra/Rad Ativado" : false,
    "Modo Radiano Ativado" : false,
    "Botão % Clássico Ativado" : false,
    "Idioma" : detectaIdiomaDoSistema(),
    "Casas Decimais" : 15
]
```

Fonte: Código fonte da Calculadora *Swift / Xcode 7.3* (2016)

Note que em “Idioma”, o valor padrão definido é a função *detectaIdiomaDoSistema*, como seu próprio nome diz, ela tem a função de detectar o idioma do dispositivo *iOS*, desta forma o idioma sempre estará correto na primeira vez que o *app* é instalado.

Imagen 2.25 - Código da função *detectaIdiomaDoSistema*

```
/**
Detecta o idioma do sistema operacional.
- Parâmetros: Nenhum.
- Retorna: Idioma. (String)

*/
func detectaIdiomaDoSistema() -> String {
    if let melhorIdioma = NSBundle.preferredLocalizationsFromArray(idiomasERegioesDisponiveis).first {
        return dicionarioCodigoIdiomaParaNomeIdioma[melhorIdioma]!
    } else {
        return "Inglês"
    }
}
```

Fonte: Código fonte da Calculadora *Swift / Xcode 7.3* (2016)

A primeira ação da função *detectaIdiomaDoSistema* é tentar atribuir o melhor idioma para uma constante através do método *NSBundle.preferredLocalizationsFromArray*. Este método recebe como parâmetro um vetor que deve conter todas os idiomas que são suportados pelo *app*, no caso da Calculadora *Swift*, este é o vetor:

Imagen 2.26 - Vetor de idiomas e regiões disponíveis

```
let idiomasERegioesDisponiveis = ["pt-BR", "en-US", "es", "zh"]
```

Fonte: Código fonte da Calculadora *Swift / Xcode 7.3* (2016)

O que o método *NSBundle.preferredLocalizationsFromArray* faz é detectar a linguagem do sistema operacional, e retornar a linguagem mais próxima à ela que esteja no vetor.

Por exemplo, se o sistema operacional do usuário estiver em “pt-PT” (Português de Portugal), este método irá analisar o vetor *idiomasERegiõesDisponíveis*, e retornar o idioma/região mais próximo existente, ou seja, “pt-BR” (Português Brasileiro).

O próximo passo da função *detectaIdiomaDoSistema* é converter o idioma/região no nome completo do idioma, ou seja, a *String* “pt-BR” deve se tornar “Português”, usa-se um dicionário para isso:

Imagen 2.27 - Convertendo códigos de idioma/região em nomes de idiomas

```
let idiomasERegioesDisponiveis = ["pt-BR", "en-US", "es", "zh"]

let dicionarioCodigoIdiomaParaNomeIdioma: [String:String] = [
    idiomasERegioesDisponiveis[0] : "Português",
    idiomasERegioesDisponiveis[1] : "Inglês",
    idiomasERegioesDisponiveis[2] : "Espanhol",
    idiomasERegioesDisponiveis[3] : "Mandarim"
]
```

Fonte: Código fonte da Calculadora *Swift / Xcode 7.3* (2016)

Após a conversão, o nome do idioma é retornado, caso ocorra algum problema na atribuição do idioma mais próximo, o idioma “Inglês” será retornado, garantindo que a função sempre retorne um idioma.

4.9 Alterando o Idioma do App

O idioma do aplicativo, além de salvo localmente nas configurações do usuário, também sempre será atribuído a uma variável global chamada *idioma*:

Imagen 2.28 - Variável Global de Idioma

```
var idioma = ""
```

Fonte: Código fonte da Calculadora *Swift / Xcode 7.3* (2016)

A razão para ela ser global é a praticidade geral, já que todas as classes precisam ter acesso à ela para alterar o idioma de suas interfaces em tempo real.

Para alterar o idioma dos elementos de interface do *app*, é necessário criar ligações de todos eles com o código. Ligações de objetos com o código fonte são chamadas de *outlets*. Vejamos as ligações dos botões do menu de opções, por exemplo:

Imagen 2.29 - Ligações *outlet* do menu de opções

```
@IBOutlet var outletBotaoVoltarMenuOpcoes: UIButton!
@IBOutlet var outletBotaoCoresMenuOpcoes: UIButton!
@IBOutlet var outletBotaoAjudaMenuOpcoes: UIButton!
@IBOutlet var outletBotaoMaisOpcoesMenuOpcoes: UIButton!
```

Fonte: Código fonte da Calculadora *Swift / Xcode 7.3* (2016)

Estas quatro ligações são referentes aos botões da interface do menu de opções:

Imagen 2.30 - Menu de opções em português

Fonte: *Simulator (iOS 9.3) - Calculadora Swift (2016)*

Para que a tradução do conteúdo dos botões seja possível, foram criados dicionários referentes a cada um deles:

Imagen 2.31 - Dicionários do menu de opções

```
let dicionarioBotaoVoltar: [String:String] = [
    "Português" : "< Voltar",
    "Inglês" : "< Back",
    "Espanhol" : "< Atrás",
    "Mandarin" : "< 回报"
]

let dicionarioBotaoCores: [String:String] = [
    "Português" : "○ Cores",
    "Inglês" : "○ Colors",
    "Espanhol" : "○ Colores",
    "Mandarin" : "○ 颜色"
]

let dicionarioBotaoAjuda: [String:String] = [
    "Português" : "? Ajuda",
    "Inglês" : "? Help",
    "Espanhol" : "? Ayuda",
    "Mandarin" : "? 帮帮我"
]

let dicionarioBotaoMaisOpcoes: [String:String] = [
    "Português" : "+ Opções",
    "Inglês" : "+ Options",
    "Espanhol" : "+ Opciones",
    "Mandarin" : "+ 选项"
]
```

Fonte: Código fonte da Calculadora *Swift / Xcode 7.3* (2016)

O texto presente nos botões é associado à estes dicionários, onde cada idioma possui um texto diferente. O uso destes dicionários é muito simples, basta enviar o nome do idioma como atributo, e ele irá retornar a tradução correta, como mostra o exemplo:

Imagen 2.32 - Exemplo de uso do dicionário do botão de ajuda

```
let dicionarioBotaoAjuda: [String:String] = [
    "Português" : "? Ajuda",
    "Inglês" : "? Help",
    "Espanhol" : "? Ayuda",
    "Mandarin" : "? 帮帮我"
]

print( dicionarioBotaoAjuda["Português"] ) // Imprime "? Ajuda"
print( dicionarioBotaoAjuda["Espanhol"] ) // Imprime "? Ayuda"
```

Fonte: Xcode 7.3 - Playground (2016)

Sempre que um idioma é carregado, usa-se a variável global *idioma* para atribuir o texto traduzido em todos os elementos de interface do *app*. A atribuição ocorre da seguinte maneira:

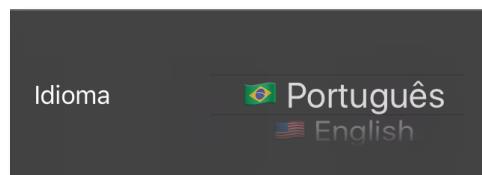
Imagen 2.33 - Aplicando os dicionário ao texto dos botões

```
func aplicaIdiomaViewController(idiomaSendoUsado: String) {
    // Menu de Configurações:
    outletBotaoVoltarMenuOpcoes.setTitle(dicionarioBotaoVoltar[idiomaSendoUsado], forState: UIControlState.Normal)
    outletBotaoCoresMenuOpcoes.setTitle(dicionarioBotaoCores[idiomaSendoUsado], forState: UIControlState.Normal)
    outletBotaoAjudaMenuOpcoes.setTitle(dicionarioBotaoAjuda[idiomaSendoUsado], forState: UIControlState.Normal)
    outletBotaoMaisOpcoesMenuOpcoes.setTitle(dicionarioBotaoMaisOpcoes[idiomaSendoUsado], forState: UIControlState.Normal)
    // ...
}
```

Fonte: Código fonte da Calculadora Swift / Xcode 7.3 (2016)

O usuário também tem a opção de alterar o idioma do *app* a qualquer momento, independente do idioma do sistema operacional, através do menu de mais opções:

Imagen 2.34 - Opção de alterar idioma do menu de mais opções



Fonte: Simulator (iOS 9.3) - Calculadora Swift (2016)

Após a alteração manual de idioma, toda a interface do *app* é traduzida em tempo real, sem a necessidade de reiniciar a aplicação.

4.10 Implementando o Multi-Tarefas para *iPad*

Na versão 9.0 do sistema *iOS*, foi introduzida a funcionalidade de multi-tarefas para os novos *iPads*. Com isso é possível executar até dois *apps* lado a lado. A calculadora *Swift* já possui suporte total a esta funcionalidade.

O processo para detectar quando o *app* entra em modo multi-tarefas é bastante semelhante ao método de detecção de orientação. O que se busca é, sempre que o usuário ativar o multi-tarefas, executar uma função no *app* que terá a finalidade de realizar o tratamento da interface.

Diferentemente da detecção de orientação de tela, não há como criar uma notificação para o modo multi-tarefas, mas é possível fazer uso do método *traitCollectionDidChange*, sobrepondo ele da seguinte forma:

Imagen 2.35 - Código do método *traitCollectionDidChange*

```
extension ViewController {
    /**
     Chama a função 'acaoAoAtivarMultiTarefas' quando modo multi-tarefas over fica ativo.
     - Parâmetros: previousTraitCollection. (UITraitCollection?)
     - Retorna: Nada.
    */
    override func traitCollectionDidChange(previousTraitCollection: UITraitCollection?) {
        acaoAoAtivarMultiTarefas()
    }
}
```

Fonte: Código fonte da Calculadora *Swift* / Xcode 7.3 (2016)

Sempre que o usuário ativa qualquer um dos modos de multi-tarefas, o método *traitCollectionDidChange* será automaticamente executado, só resta aplicar todas as mudanças de interface necessárias nele, que neste caso, são realizadas pelo método *acaoAoAtivarMultiTarefas*. O funcionamento acaba sendo o mesmo de uma notificação de orientação de tela, e os dois podem funcionar ao mesmo tempo.

No tratamento da interface, é necessário detectar a resolução atual da tela para determinar a orientação, o tipo de dispositivo e se o modo multi-tarefas realmente está ativado. Esta funcionalidade está reservada para a função *detectaTipoDeTela*:

Imagen 2.36 - Parte do código da função *detectaTipoDeTela*

```
/*
Detecta o tipo e orientação de tela.
- Parâmetros: Tamanho da tela. (CGRect)
- Retorna: Tipo e orientação de tela. (String)

*/
func detectaTipoDeTela(tamanhoDaTela: CGRect) -> String {
    let largura = tamanhoDaTela.width
    let altura = tamanhoDaTela.height
    var tipoDaTela: String

    // -----
    // iPad:
    // -----

    if altura == 1024 && largura == 768 { // Retrato
        tipoDaTela = "ipadPortrait"
    } else if (altura == 1024 && largura == 320) || (altura == 1024 && largura == 438) { // Retrato (Multi-Tarefas)
        tipoDaTela = "ipadPortraitSlideOver"
    } else if (altura == 768 && largura == 1024) || (altura == 768 && largura == 694) { // Horizontal
        tipoDaTela = "ipadLandscape"
    } else if (altura == 768 && largura == 320) || (altura == 768 && largura == 507) { // Horizontal (Multi-Tarefas)
        tipoDaTela = "ipadLandscapeSlideOver"
    }
    // -----
    // iPad Pro:
    // -----

    // ...
}
```

Fonte: Código fonte da Calculadora *Swift / Xcode 7.3 (2016)*

A função *detectaTipoDeTela* recebe como parâmetro um valor do tipo *CGRect*, em *Swift* este valor representa as dimensões em pixels de um retângulo. Para utilizar esta função, é necessário enviar a ela o tamanho da tela em que o *app* ocupa, ou seja, a região retangular da tela em que ele ocupa no formato *CGRect*.

A melhor forma pra obter um tamanho de tela preciso é medir as dimensões do contêiner pai de todos os elementos de interface, no caso da Calculadora *Swift*, ele se chama *viewPrincipal*.

É importante este contêiner possuir *constraints* com ligações de valor zero em todos os quatro lados da interface do *app*, garantindo que ele sempre ocupe o tamanho máximo disponível. Para acessar suas informações, foi criado uma ligação entre ele e o código fonte:

Imagen 2.37 - Ligação entre o contêiner pai e o código

@IBOutlet var viewPrincipal: UIView!

Fonte: Código fonte da Calculadora *Swift / Xcode 7.3 (2016)*

Após a criação da ligação do contêiner, é possível acessar as dimensões da tela através da seguinte sintaxe:

Imagen 2.38 - Acessando as dimensões de um contêiner

```
viewPrincipal.bounds
```

Fonte: Código fonte da Calculadora *Swift / Xcode 7.3 (2016)*

O comando acima retorna um valor do tipo *CGRect*, basta enviar este valor à função *detectaTipoDeTela* para obter a *String* que contém as informações necessárias para o tratamento da interface:

Imagen 2.39 - Obtendo o tipo de tela

```
let tipoDeTela = detectaTipoDeTela(viewPrincipal.bounds)
```

Fonte: Código fonte da Calculadora *Swift / Xcode 7.3 (2016)*

Caso o comando acima seja executado enquanto o usuário esteja usando um *iPad* no modo horizontal, o valor atribuído à constante *tipoDeTela* será a String “*ipadLandscape*”, se estiver em modo retrato e com o multi-tarefas ativo, o valor será “*ipadPortraitSlideOver*”.

Com esta informação é possível realizar o tratamento ideal para todos os modos e orientações que uma tela pode assumir.

4.11 Comparando a Calculadora *Swift* com Outros Apps

Neste capítulo serão realizadas diversas comparações da Calculadora *Swift* com algumas calculadoras gratuitas da *App Store*. Entre as calculadoras selecionadas, está a calculadora mais popular para *iPad*, que é líder tanto na categoria “gratuitos” quanto “pagos”.

As comparações abordarão quatro temas principais: tamanho em disco, velocidade de carregamento, funcionalidade e preço. Todos os testes foram realizados em um *iPad Air* de primeira geração.

Abaixo, a lista de *apps* selecionados para a comparação:

Tabela 2 - Relação das calculadoras mais populares e suas avaliações

Nome do App	Avaliação dos usuários (Max de 5 estrelas)	Ranking na Loja (Gratuitos)	Data de verificação das informações
Calculadora Pro para iPad Grátis	4,5 Estrelas	25º	12/05/2016
A Calculadora - grátils	4 Estrelas	87º	12/05/2016
Calculadora•	4 Estrelas	-	12/05/2016

Fonte: Elaborado pelo autor (2016)

O primeiro *app* da lista, *Calculadora Pro para iPad Grátis*, é um dos aplicativos mais populares da *App Store*, e de longe a calculadora mais popular da *App Store*. Para se ter uma idéia da popularidade desta calculadora, na data de verificação destas informações, o aplicativo oficial do *Facebook* estava duas posições abaixo da calculadora, como mostra a imagem:

Imagen 2.40 - *Top charts*, aba de gratuitos da *App Store* (12/05/2016)Fonte: *App Store (iPad)* (2016)

4.11.1 Tamanho em Disco

Nesta comparação, todos os aplicativos terão seus tamanhos em disco analisados. Tanto o tamanho da aplicação quanto de seus dados locais serão considerados.

Tabela 3 - Relação do Espaço em Disco das Calculadoras

Nome do App	Espaço em disco do app	Espaço em disco de arquivos locais	Espaço em disco total
Calculadora Swift	18,7 MB	660 KB	19,3 MB
Calculadora Pro para iPad Grátis	83,6 MB	15,2 MB	98,7 MB
A Calculadora - grátis	51,5 MB	10,3 MB	61,8 MB
Calculadora•	13,2 MB	2,9 MB	16,1 MB

Fonte: Elaborado pelo autor (2016)

Vale notar que o *app* de calculadora mais popular da *App Store* ocupa quase 100 MB de disco, sendo possível questionar sua qualidade.

Apenas um *app* ocupa menos espaço que a Calculadora *Swift*, porém o mesmo é apenas uma calculadora básica, sem funções científicas.

4.11.2 Funcionalidade

Na tabela abaixo, algumas funcionalidades foram relacionadas:

Tabela 4 - Relação de Funcionalidades das Calculadoras

Nome do App	Calculadora Científica	Multi-Tarefas	Compatível com iPhone, iPod e iPad	Cálculos em Tempo Real	Temas de Cores
Calculadora Swift	Sim	Sim	Sim	Sim	Sim
Calculadora Pro para iPad Grátis	Sim	Sim	Não	Não	Sim
A Calculadora - grátis	Sim	Sim	Sim	Não	Sim

Nome do App	Calculadora Científica	Multi-Tarefas	Compatível com iPhone, iPod e iPad	Cálculos em Tempo Real	Temas de Cores
Calculadora•	Não	Não	Sim	Não	Não

Fonte: Elaborado pelo autor (2016)

Nota-se que o *app calculadora•*, apesar de ocupar menos espaço em disco, possui pouquíssimas funcionalidades em relação às outras calculadoras. Também é importante mencionar que nenhum *app*, com exceção da Calculadora *Swift*, possui a função de cálculos em tempo real.

4.11.3 Velocidade de Carregamento

Para realizar o teste de velocidade de carregamento dos *apps*, foram realizadas dez medições de tempo por aplicativo, onde o tempo final é uma média simples dos resultados. Um *app* só será considerado completamente carregado quando sua tela de carregamento desaparecer e der lugar à sua interface principal. Após cada medição, o aplicativo será fechado e re-aberto, garantindo que ele jamais seja carregado da memória RAM do dispositivo.

Tabela 5 - Relação de Tempo de Carregamento das Calculadoras

Nome do App	Média do Tempo de Carregamento (Segundos)
Calculadora <i>Swift</i>	1,1
Calculadora Pro para iPad Grátis	1,44
A Calculadora grátis	1,76
Calculadora•	1,19

Fonte: Elaborado pelo autor (2016)

Apesar das diferenças não serem muito significativas no dispositivo testado, o tempo de carregamento é um bom indicador de boas práticas de programação e eficiência do código fonte. A Calculadora *Swift* conseguiu obter um menor tempo de carregamento mesmo se comparada à última calculadora da lista *Calculadora•*, que possui pouquíssimas funcionalidades.

4.11.4 Rentabilidade

Muitos aplicativos são considerados “grátis”, porém possuem inúmeros anúncios e funções bloqueadas. A tabela a seguir possui uma relação de fatores relacionados à rentabilidade deles:

Tabela 6 - Relação de Características Rentáveis das Calculadoras

Nome do App	Grátis	Código Aberto	Anúncios	Compra de Funções
Calculadora Swift	Sim	Sim	Não	Não
Calculadora Pro para iPad Grátis	Sim	Não	Sim	Sim
A Calculadora grátis	Sim	Não	Sim	Sim
Calculadora•	Sim	Não	Sim	Não

Fonte: Elaborado pelo autor (2016)

Todos os *apps* são “grátis”, porém nenhum deles, com exceção da Calculadora *Swift*, é de código aberto. Também é importante ressaltar que a Calculadora *Swift* é o único aplicativo da lista que não possui anúncios.

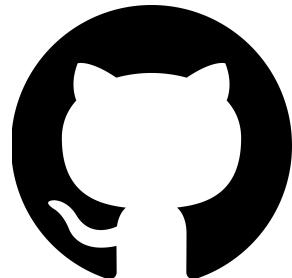
Seria injusto criticar aplicativos pelo simples fato deles buscarem lucros, afinal, o desenvolvimento consome tempo e conhecimento do desenvolvedor. Na verdade, há também um custo para colocar e manter aplicativos na *App Store*. É natural que aplicativos grátis tentem obter uma certa rentabilidade.

A crítica tem relação com a qualidade destes *apps*, que não justificam seus excessivos anúncios, venda de funções básicas e falta de evolução, onde não recebem atualizações importantes por anos.

Dito isto, não foi descartada a possibilidade de, em versões futuras, vender temas de cores exclusivos para a Calculadora *Swift* pelo preço mínimo (\$0,99) e pagamento único, mantendo a promessa de jamais incluir anúncios ou venda de funções. O código fonte da Calculadora *Swift* continuará a ser aberto em todas as versões.

4.12 Código Fonte da Calculadora *Swift*

Imagen 2.41 - Logotipo do GitHub



Fonte: GitHub (2016)

O código fonte da versão 1.0 do app Calculadora *Swift* está disponível através do link: github.com/ThiagoAM/calculadora-swift-1-0.

5. Criando o app “Olá Mundo” Interativo

Neste capítulo, será demonstrado, passo a passo, o processo de criação de um *app* interativo para *iOS*. As funcionalidades do aplicativo são simples, porém apresentam alguns conceitos fundamentais do desenvolvimento de interfaces através da *IDE Xcode*.

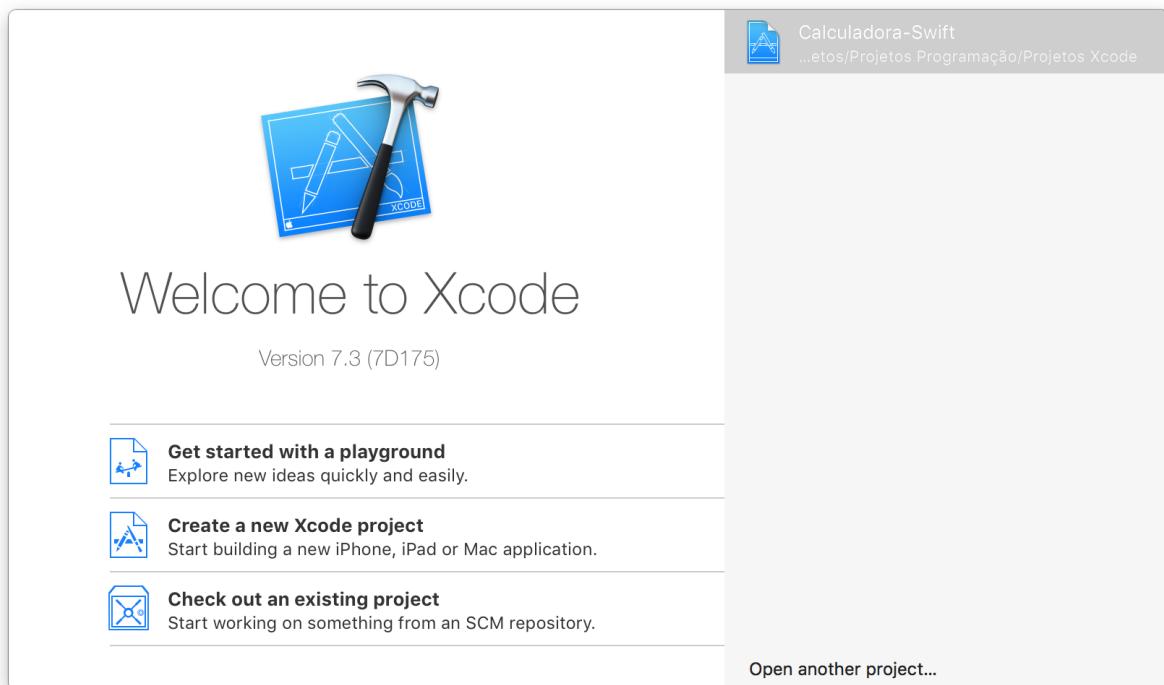
Infelizmente o *Xcode* é um software exclusivo do sistema operacional *OS X*, logo para reproduzir este passo a passo, o acesso à este sistema é requerido.

Realize o download gratuito do *Xcode* através da *App Store* para *Mac*.

5.1 Construindo a Interface do App

Após a instalação do *Xcode*, execute a aplicação. A seguinte tela será exibida:

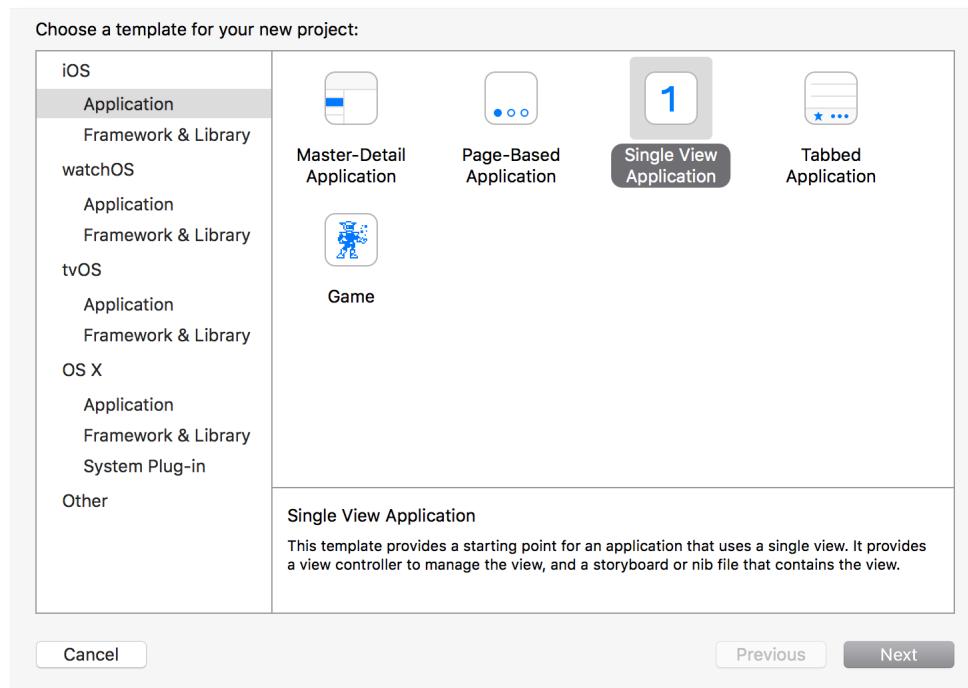
Imagen 3.0 - Tela de boas vindas do *Xcode*



Fonte: *Xcode 7.3 (2016)*

Clique no botão *Create a new Xcode project*, a próxima tela de será exibida:

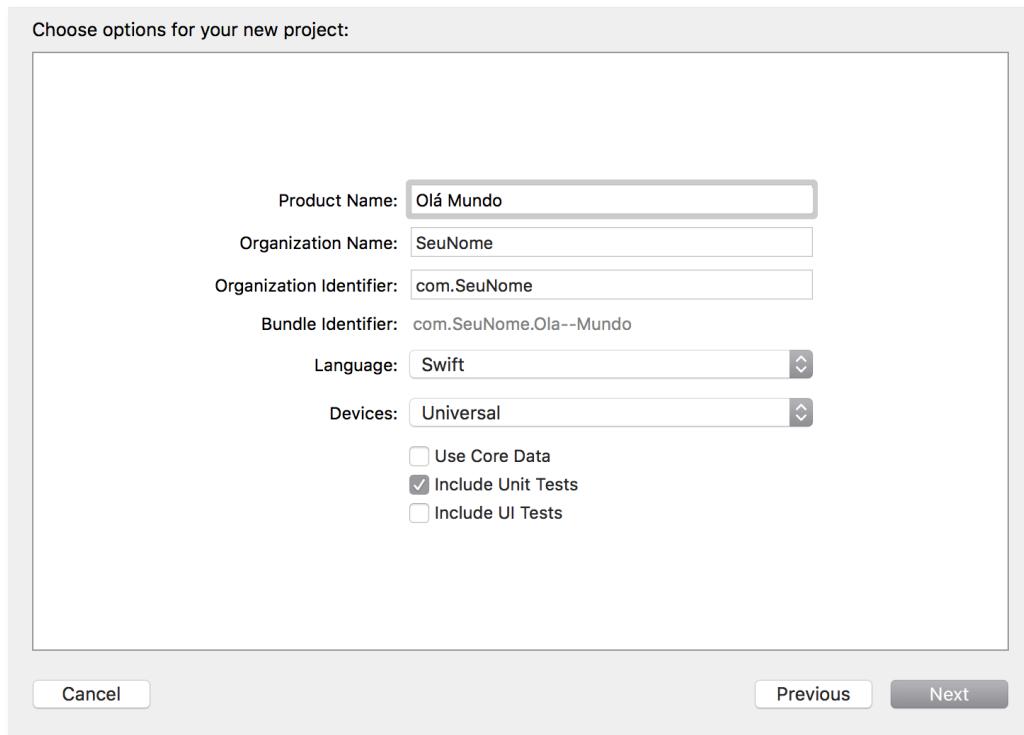
Imagen 3.1 - Tela de seleção de modelos de projeto do Xcode



Fonte: Xcode 7.3 (2016)

Selecione a opção *Single View Application* e clique em *Next*:

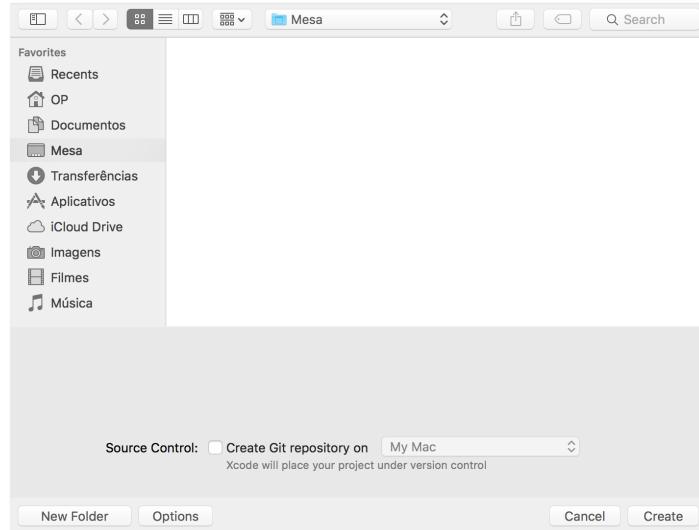
Imagen 3.2 - Opções do projeto “Olá Mundo”



Fonte: Xcode 7.3 (2016)

No campo *Product Name*, digite o nome do app, que será “Olá Mundo”. Em *Organization Name* digite seu nome. Tenha certeza que o campo *Language* está como “Swift” e *Devices* como “Universal”. Clique em *Next*:

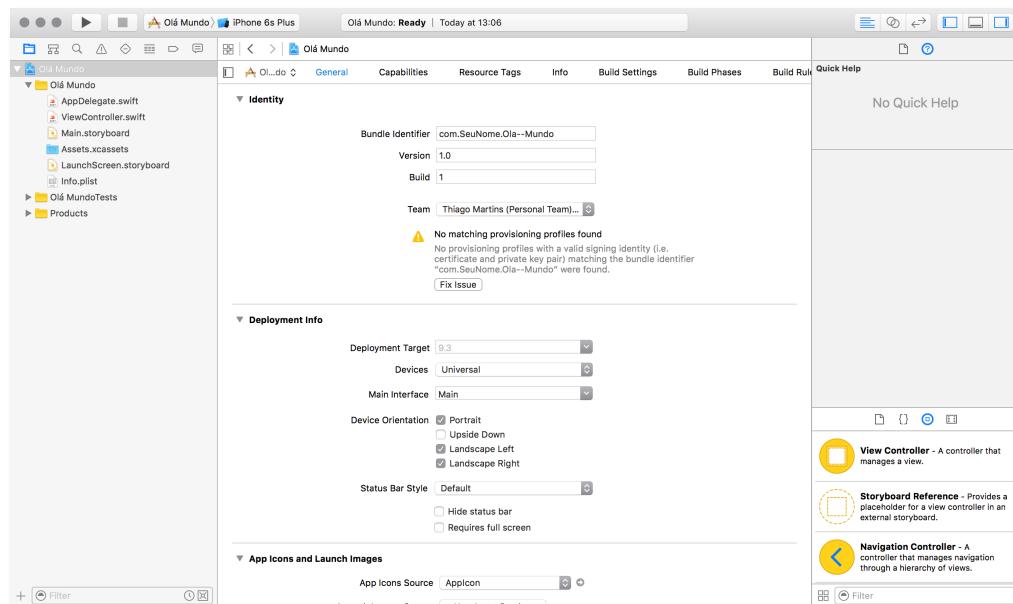
Imagen 3.3 - Seleção de Diretório do Projeto “Olá Mundo”



Fonte: Xcode 7.3 (2016)

Selecione como diretório a Mesa (ou seu diretório de preferência), e clique em *Create*. A seguinte tela será exibida:

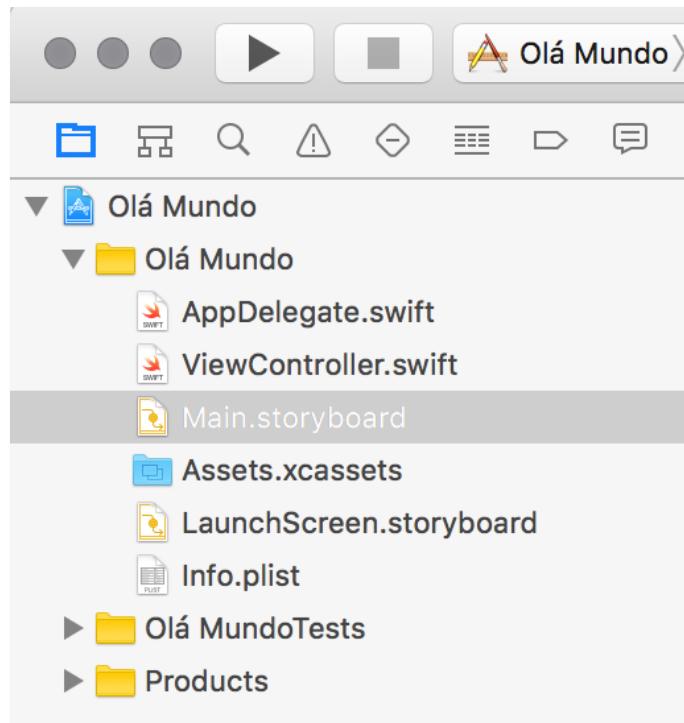
Imagen 3.4 - Tela de Projeto Inicial do Xcode



Fonte: Xcode 7.3 (2016)

Primeiramente será criada a interface do *app*, no canto esquerdo da interface, clique em *Main.storyboard*, como mostra a imagem:

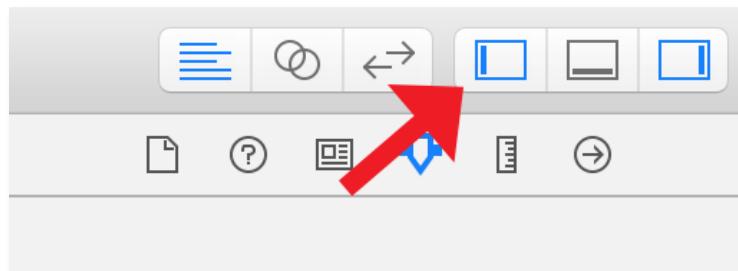
Imagen 3.5 - Acessando o *Main.storyboard* através do navegador do Xcode



Fonte: Xcode 7.3 (2016)

Como o navegador não será mais utilizado neste projeto, esconda ele para liberar mais espaço às outras interfaces do Xcode. Clique no botão *Hide or Show the Navigator*, localizado no canto superior direito da interface, como a imagem abaixo indica:

Imagen 3.6 - Escondendo o Navegador do Xcode

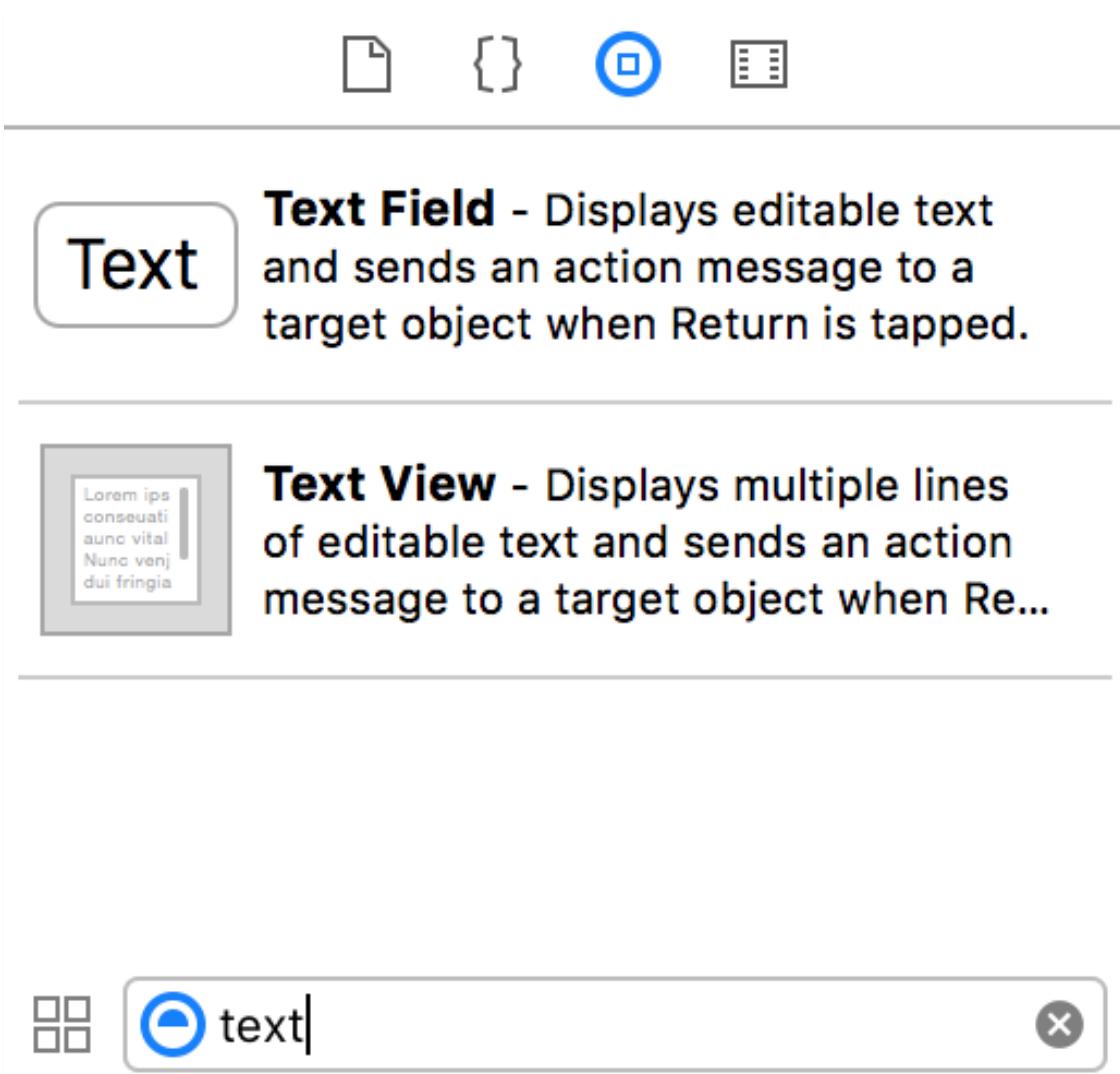


Fonte: Xcode 7.3 (2016)

O primeiro elemento de interface a ser criado é um campo de texto, nele o usuário poderá digitar qualquer coisa.

No campo de pesquisa da biblioteca de objetos do *Xcode*, localizada no canto inferior direito da interface, digite “text”, como mostra a imagem:

Imagen 3.7 - Pesquisando por “text” na Biblioteca de Objetos

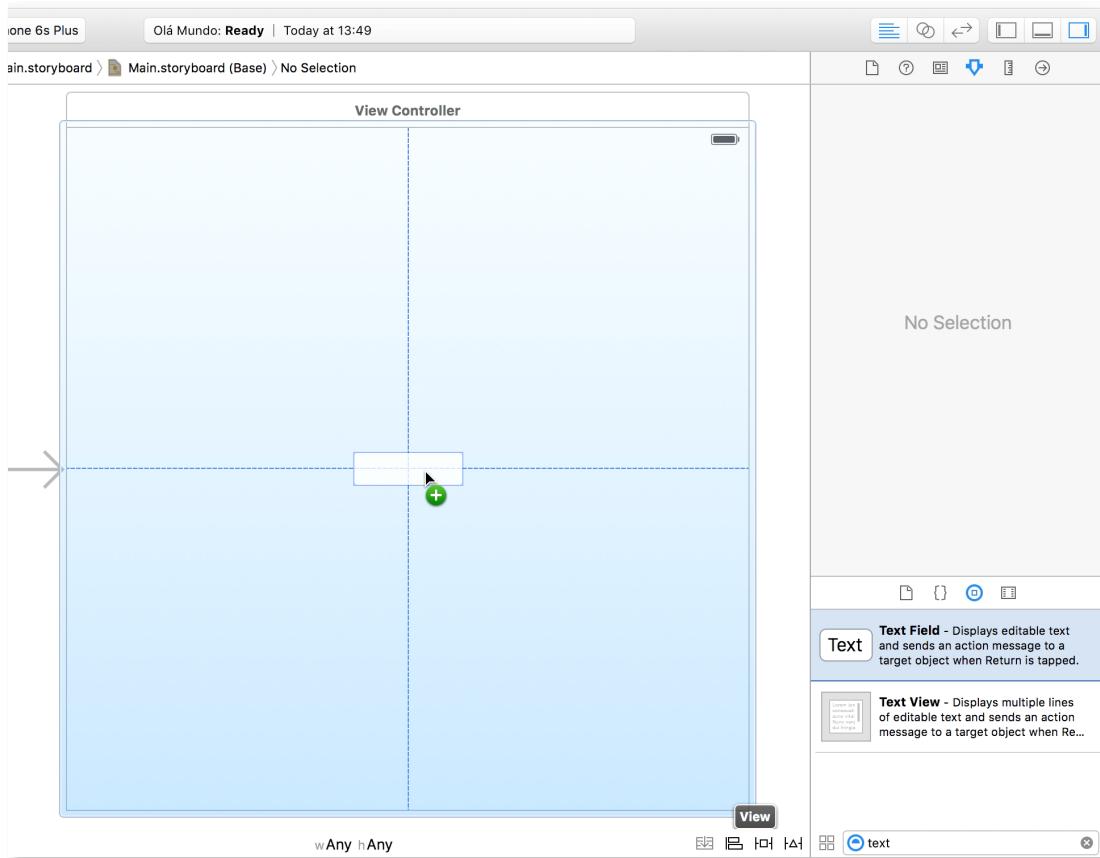


Fonte: *Xcode 7.3 (2016)*

Note que dois resultados foram exibidos, porém apenas um objeto será utilizado, que é o *Text Field*. Ele consiste em um campo de texto, com toda sua funcionalidade já pré-programada pela Apple.

Para inserir este objeto no *app*, basta arrastar-lo para dentro da interface. Clique e segure no objeto *Text Field*, arraste e solte ele no meio da interface *View Controller*, como mostra a imagem a seguir:

Imagen 3.8 - Arrastando o objeto *Text Field* para a Interface

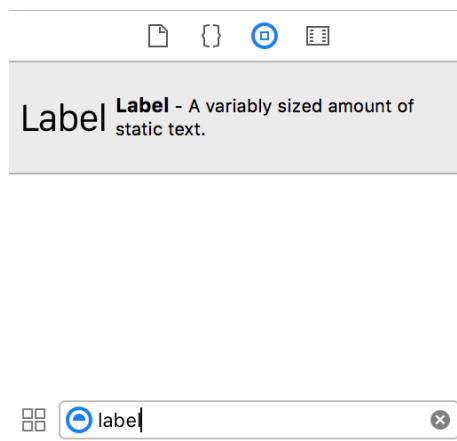


Fonte: Xcode 7.3 (2016)

Agora é necessário dizer ao usuário o que fazer com este campo de texto. Uma das formas de fazer isso é inserir uma etiqueta acima do campo com um mensagem.

Na biblioteca de objetos, pesquise por “*label*”:

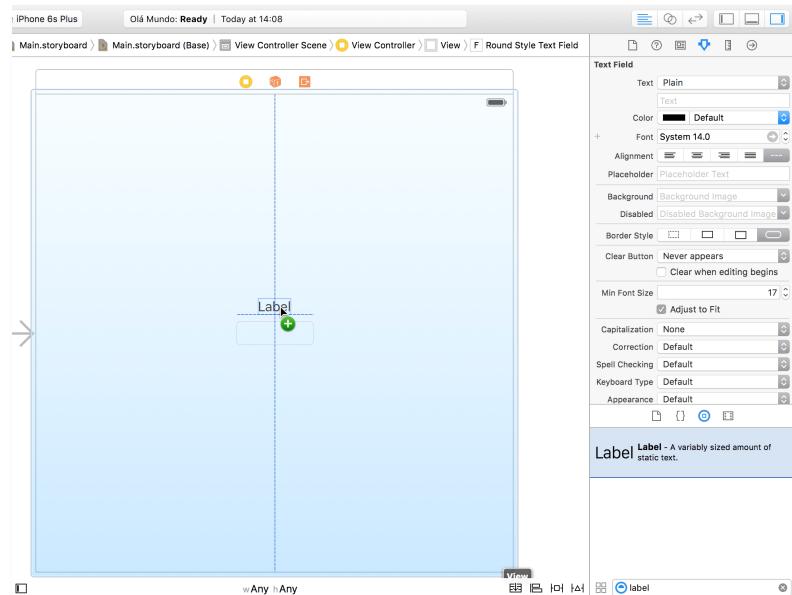
Imagen 3.9 - Pesquisando por “*label*” na Biblioteca de Objetos



Fonte: Xcode 7.3 (2016)

Selecione e arraste o único objeto exibido acima do campo de texto, como mostra a imagem:

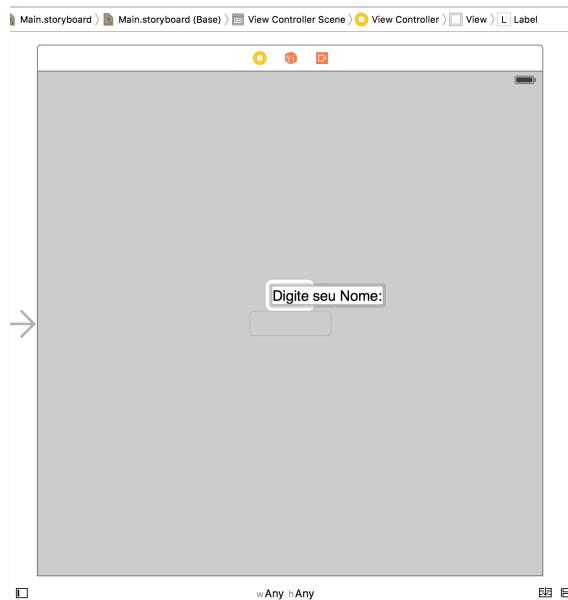
Imagen 3.10 - Arrastando o Objeto *Label* para a Interface



Fonte: *Xcode 7.3 (2016)*

Na interface *View Controller*, clique duas vezes sobre o objeto *label* e altere seu texto para “Digite seu Nome:”, como a imagem abaixo indica:

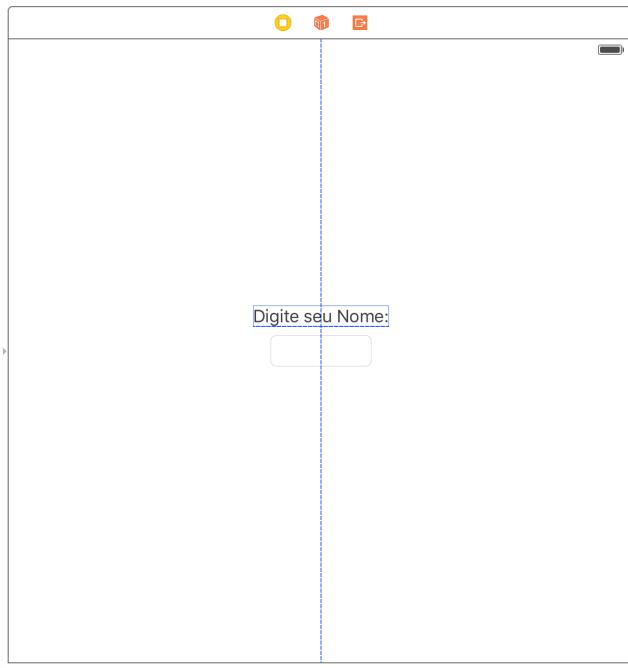
Imagen 3.11 - Alterando o texto do objeto *Label*



Fonte: *Xcode 7.3 (2016)*

Pressione a tecla *Return* para concluir a alteração e alinhe o objeto ao centro da tela novamente (Arrastando com o mouse):

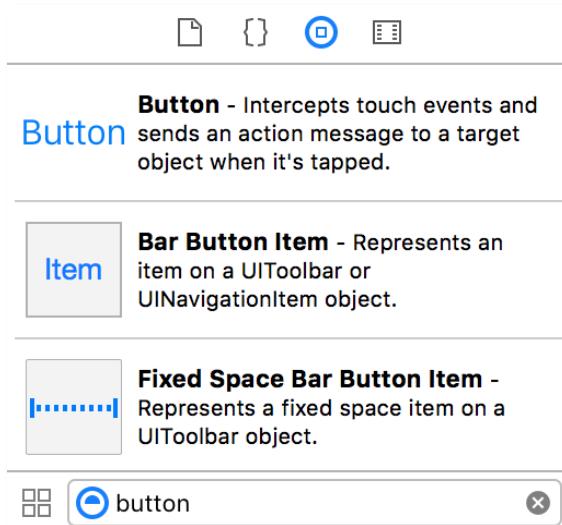
Imagen 3.12 - Alinhando a etiqueta ao centro da interface



Fonte: *Xcode 7.3 (2016)*

Agora iremos criar dois botões, “Ok” e “Limpar”. Na biblioteca de objetos, pesquise por “button”:

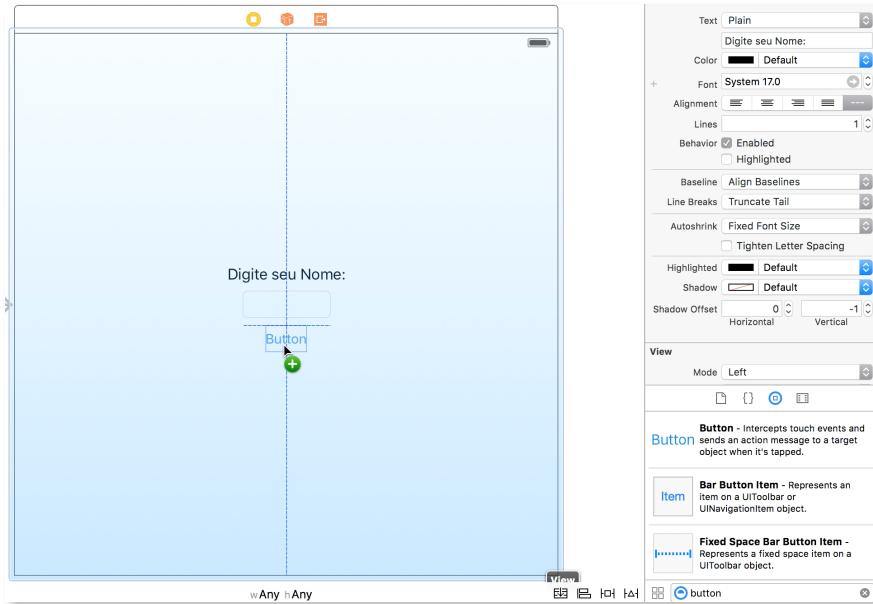
Imagen 3.13 - Pesquisando por “button” na Biblioteca de Objetos



Fonte: *Xcode 7.3 (2016)*

Arraste o objeto *Button* para baixo do campo de texto:

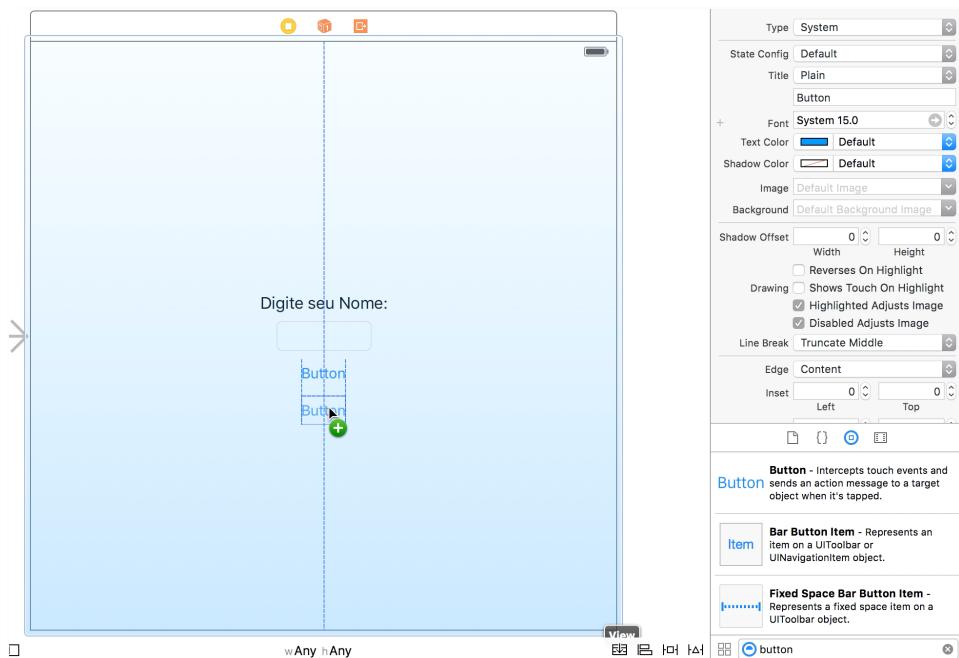
Imagen 3.14 - Arrastando o Objeto *Button* para a Interface



Fonte: Xcode 7.3 (2016)

Arraste, da mesma forma, outro botão para a interface, posicionando o mesmo abaixo do outro botão:

Imagen 3.15 - Arrastando outro Objeto *Button* para a interface



Fonte: Xcode 7.3 (2016)

Utilizando a técnica do duplo clique sob os objetos, renomeie o primeiro botão para “Ok” e o segundo para “Limpar”:

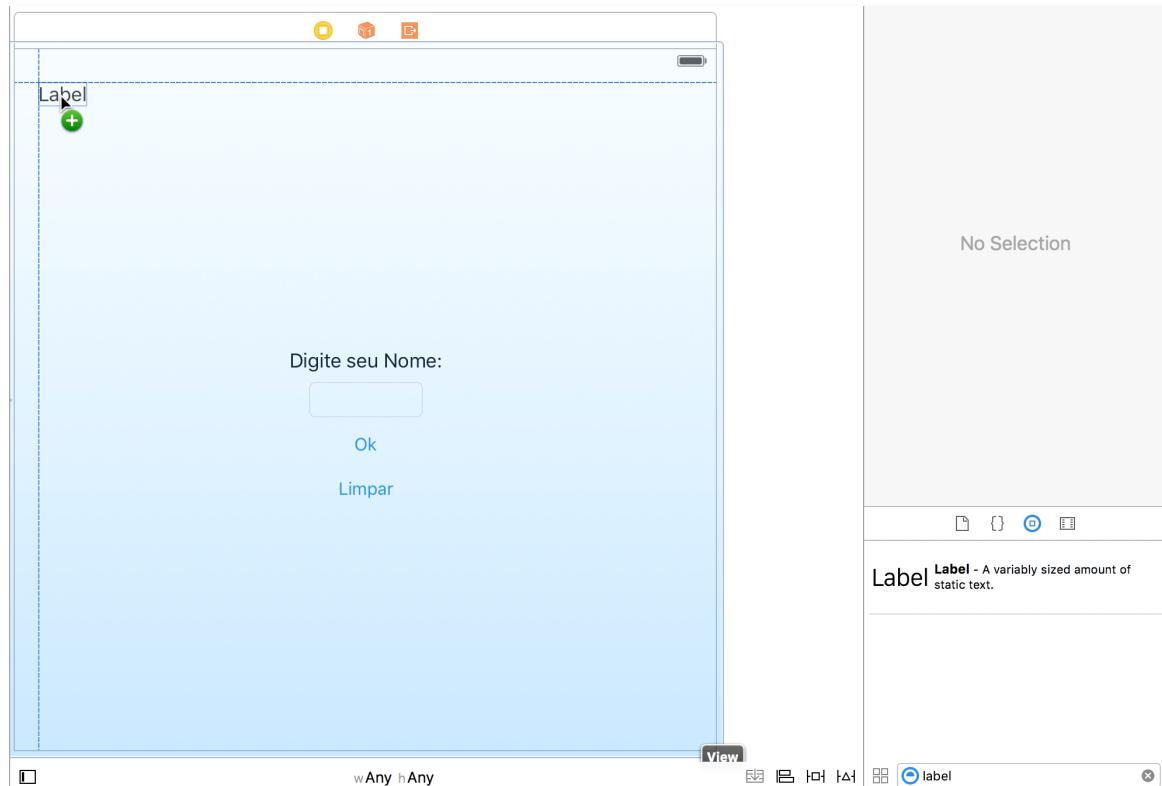
Image 3.16 - Renomeando os botões para “Ok” e “Limpar”



Fonte: Xcode 7.3 (2016)

Por fim, adicionaremos o último elemento da interface, que será a etiqueta “Olá, Mundo”. Na biblioteca de objetos, pesquise por “label”, e arraste ele para as margens do lado esquerdo da interface, como a imagem abaixo demonstra:

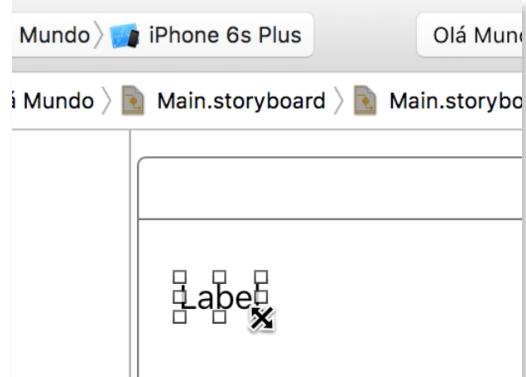
Imagen 3.17 - Arrastando a etiqueta final para a interface



Fonte: Xcode 7.3 (2016)

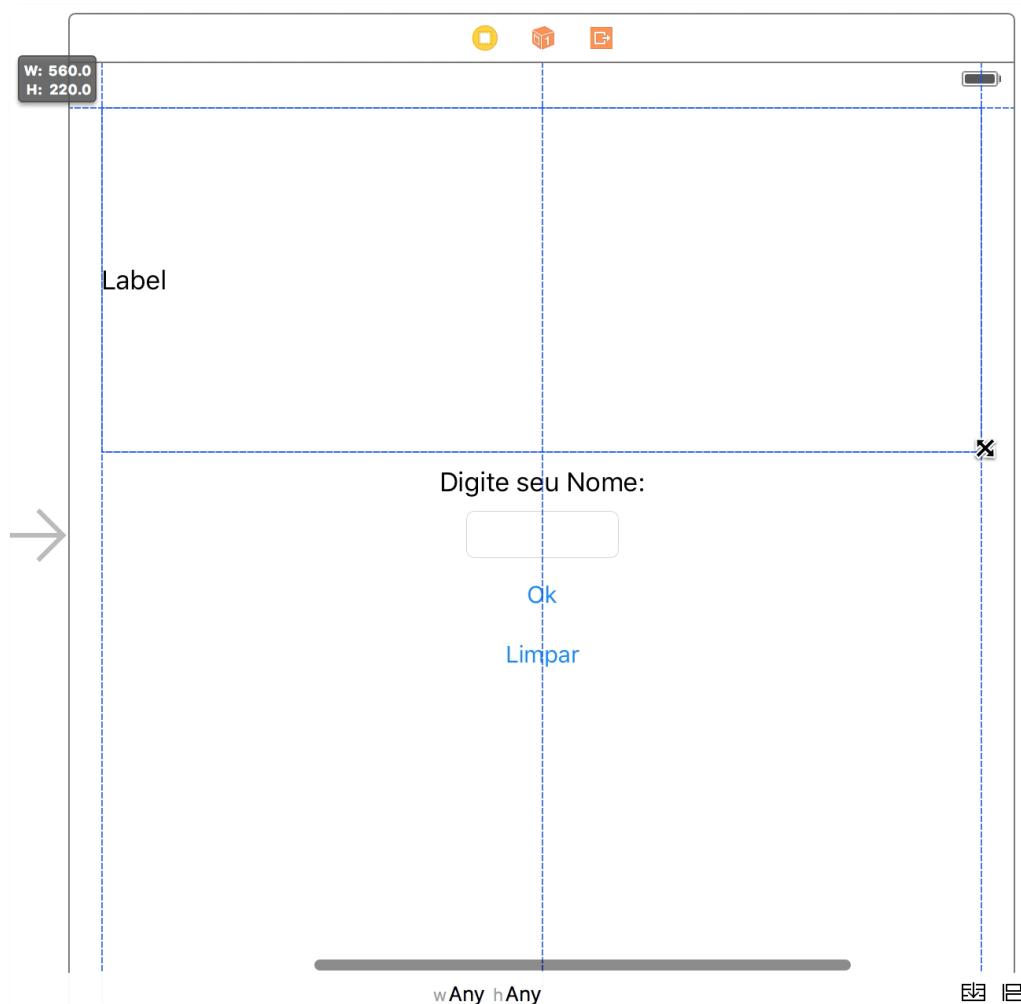
Segure no canto inferior direito do objeto a arraste para redimensionar-lo até as margens do outro lado da interface, como mostram as imagens abaixo:

Imagen 3.18 - Apontando sobre a interface de redimensionamento do objeto



Fonte: Xcode 7.3 (2016)

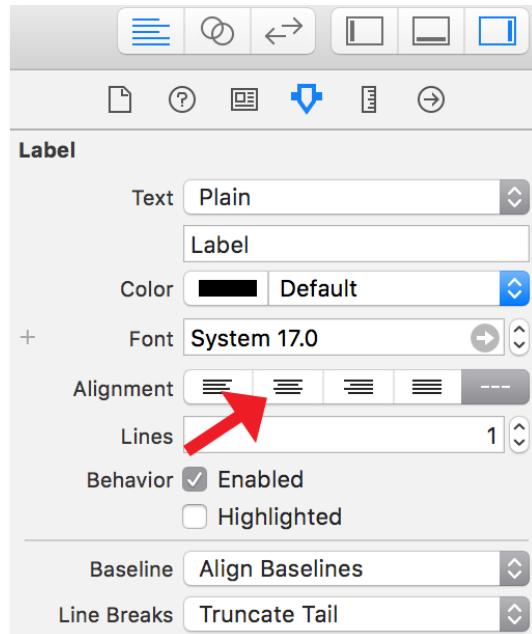
Imagen 3.19 - Redimensionando a etiqueta final



Fonte: Xcode 7.3 (2016)

Na barra de utilidades, ao lado esquerdo da interface, clique no atributo de texto centralizado, como indica a imagem:

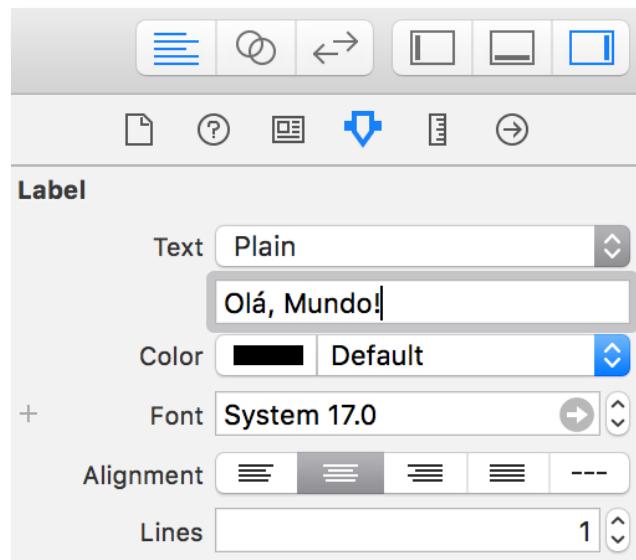
Imagen 3.20 - Centralizando o texto da etiqueta



Fonte: Xcode 7.3 (2016)

Ainda na barra de utilidades, abaixo do campo *Text (Plain)*, digite, no lugar de *"Label"*, "Olá, Mundo!" e pressione a tecla *Return* para concluir a alteração:

Imagen 3.21 - Alterando o texto “Label” para “Olá, Mundo!”

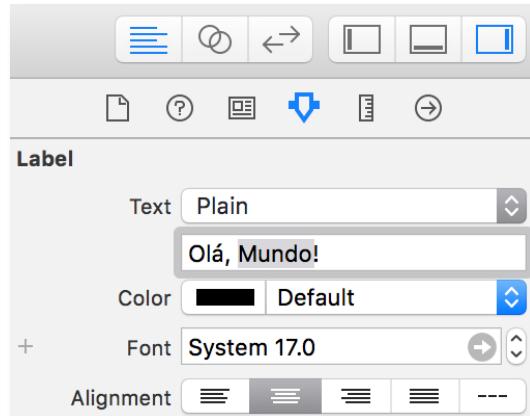


Fonte: Xcode 7.3 (2016)

Também é possível utilizar caracteres *Unicode* em etiquetas, no lugar da palavra “Mundo” pode-se usar um *emoji*, por exemplo.

Para isto, na barra de utilidades, selecione apenas a palavra “Mundo”:

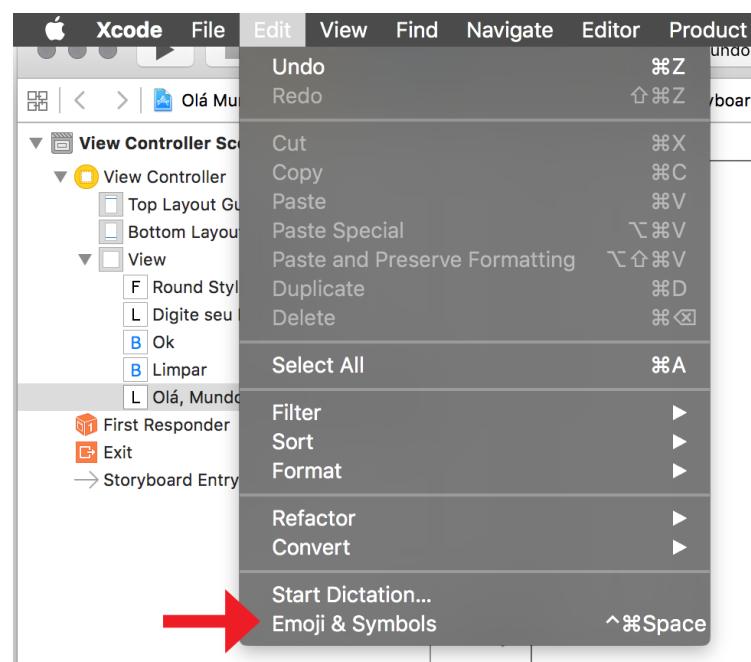
Imagen 3.22 - Selecionando a palavra “Mundo”



Fonte: Xcode 7.3 (2016)

No teclado do *Mac*, pressione as teclas *Control*, *Command* e Espaço ao mesmo tempo para abrir a interface de caracteres. Também é possível acessar esta interface através do menu *Edit → Emoji & Symbols*:

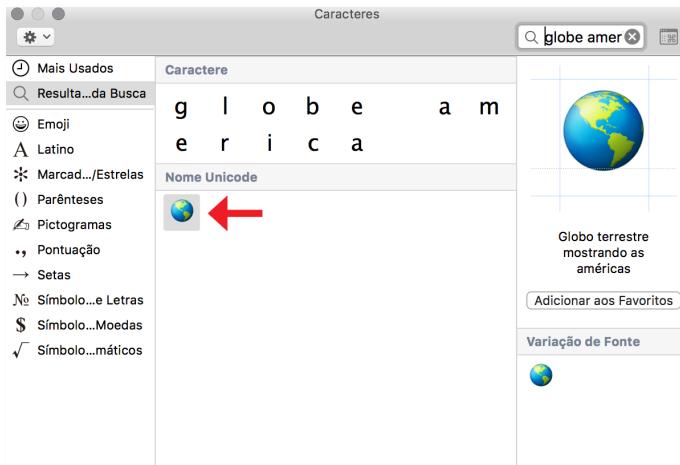
Imagen 3.23 - Encontrando a interface de *Emojis* e Símbolos



Fonte: Xcode 7.3 (2016)

Uma tela com diversos caracteres e *emojis* será exibida, em seu campo de pesquisa digite “*globe america*”, clique duas vezes no único resultado abaixo do campo Nome *Unicode*:

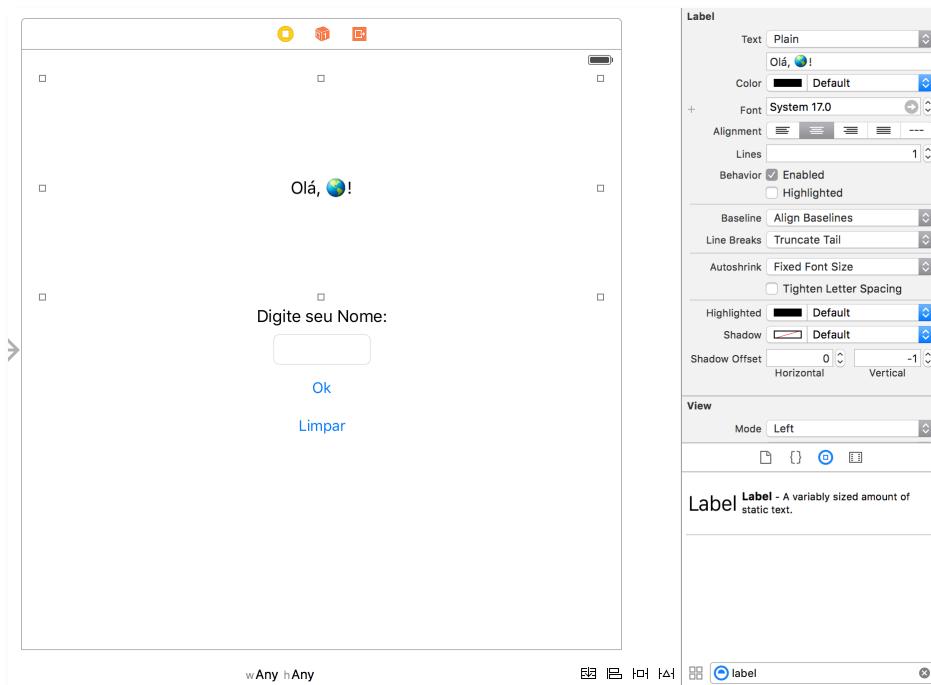
Imagen 3.24 - Selecionando o *Emoji* de globo



Fonte: Xcode 7.3 (2016)

Feche a janela de caracteres e pressione a tecla *Return* para confirmar a alteração.
A interface agora tem a seguinte aparência:

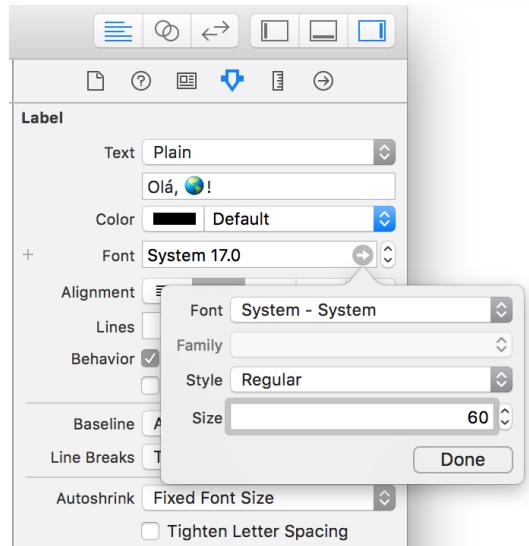
Imagen 3.25 - Verificando a interface do app



Fonte: Xcode 7.3 (2016)

Por fim, aumente o tamanho da fonte da etiqueta “Olá, Mundo” para 60. Para isto, na barra de utilidade, clique na seta ao lado do campo *Font (System 17.0)*, no campo *Size* digite o valor 60 e clique em *Done*:

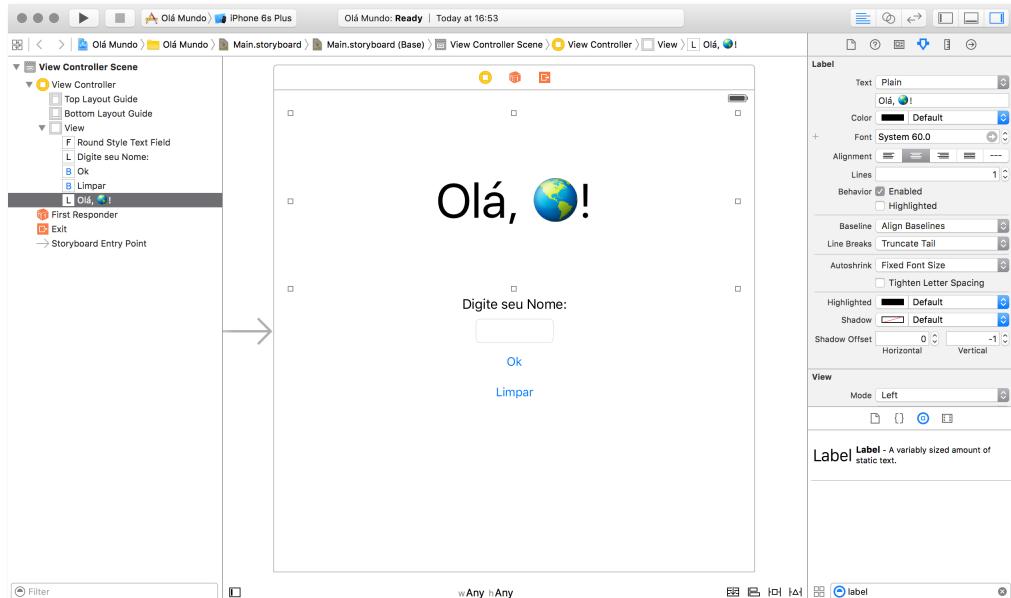
Imagen 3.26 - Alterando o tamanho da fonte da etiqueta “Olá, Mundo”



Fonte: Xcode 7.3 (2016)

Todos os objetos foram inseridos na interface, concluindo a primeira parte deste capítulo.

Imagen 3.27 - Conclusão da construção da interface do app



Fonte: Xcode 7.3 (2016)

5.2 Adicionando adaptabilidade à interface

Mesmo sem funcionalidade alguma, já é possível compilar e executar o app para verificar como a interface irá reagir em um dispositivo *iOS*. No canto superior esquerdo da interface do *Xcode*, clique no botão *Build and Run*, como indica a imagem abaixo:

Imagen 3.28 - Executando o Simulador pela primeira vez

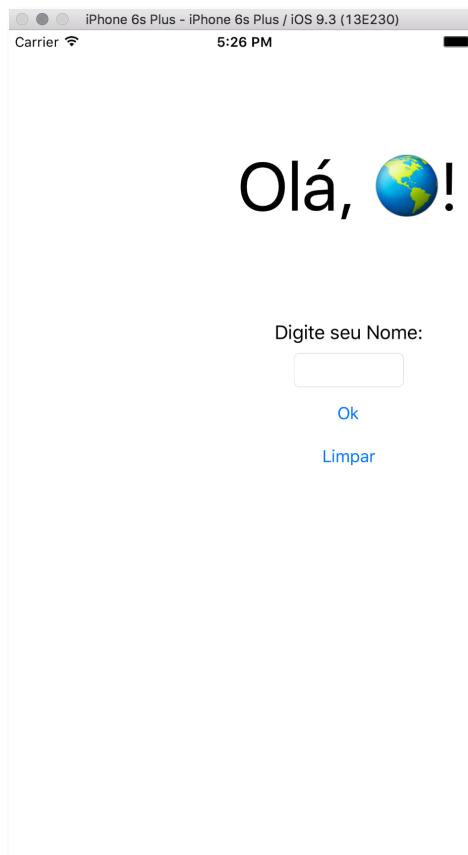


Fonte: *Xcode 7.3 (2016)*

Isso irá executar o programa *Simulator*, que neste caso, irá simular um dispositivo *iPhone 6S Plus*.

O resultado da interface será o seguinte:

Imagen 3.29 - Verificando a interface do app no Simulador *iOS*

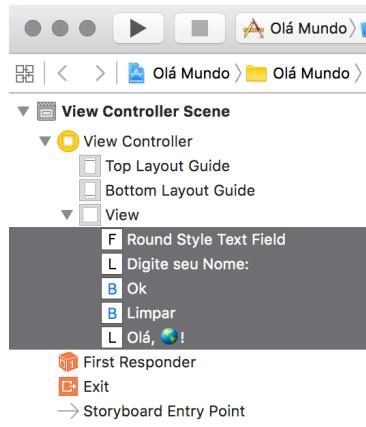


Fonte: *Simulator (iOS 9.3) (2016)*

Note que a interface não está alinhada. Para corrigir isso, é preciso adicionar *constraints* aos objetos de interface. *Constraints* são regras que os elementos de interface devem seguir. As regras sempre se mantêm as mesmas, mesmo se o tamanho ou posição da tela for alterado. Por exemplo, se for aplicado a regra “alinhem-se no centro” à todos os objetos, eles irão permanecer centralizados, independente do estado da tela.

As primeiras *constraints* que serão adicionadas irão centralizar todos os elementos de interface. Primeiramente feche o simulador (*Command + Q*), depois selecione todos os objetos adicionados à interface, como mostra a imagem:

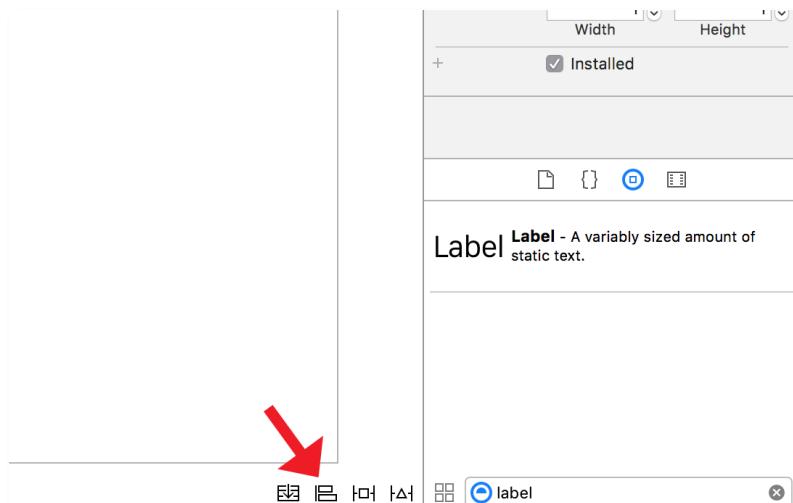
Imagen 3.30 - Selecionando todos os objetos adicionados ao app



Fonte: *Xcode 7.3 (2016)*

Em seguida, clique no botão *Align*, localizado na parte inferior direita do *Xcode*:

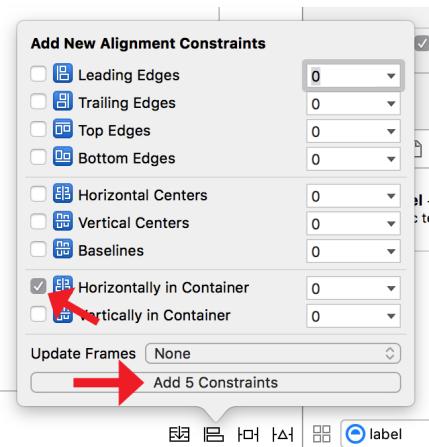
Imagen 3.31 - Clicando no botão *Align*



Fonte: *Xcode 7.3 (2016)*

Uma janela irá se abrir com diversos itens, selecione a opção *Horizontally in Container* e clique em *Add 5 Constraints*:

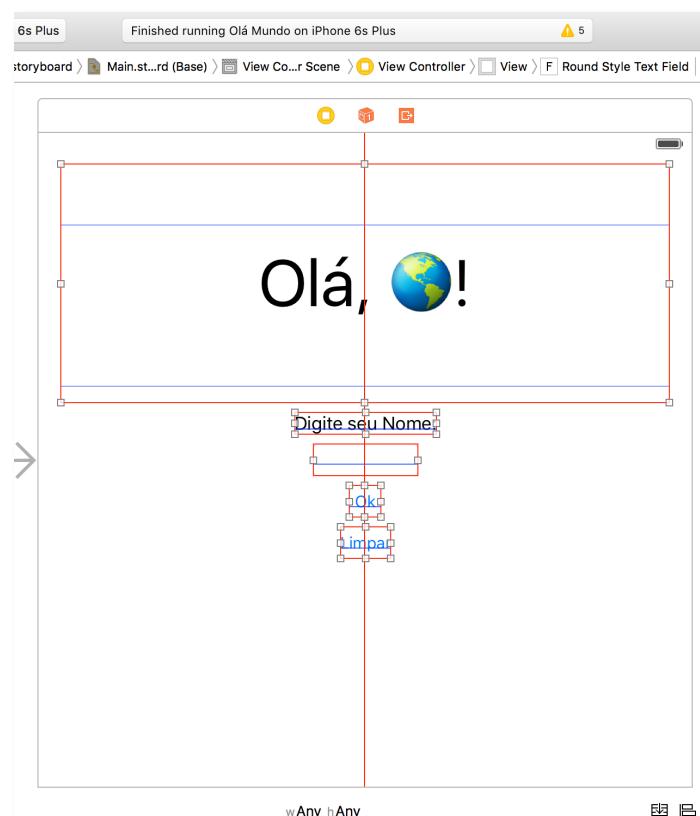
Imagen 3.32 - Adicionando *Constraints* Horizontais



Fonte: Xcode 7.3 (2016)

Note que diversas linhas vermelhas aparecem na interface, indicando erro:

Imagen 3.33 - Verificando os Erros de *Constraints*

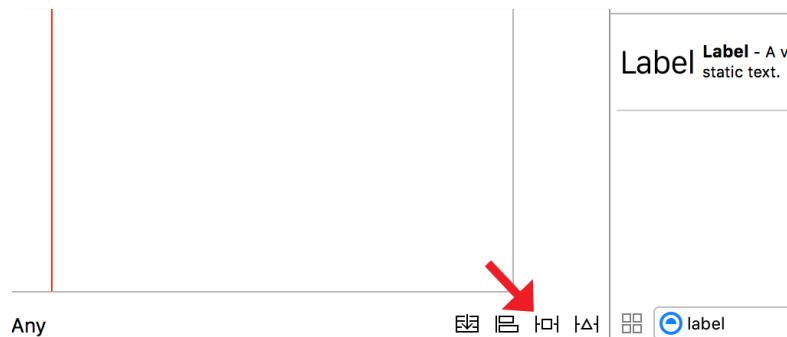


Fonte: Xcode 7.3 (2016)

Isso ocorre pois o *Xcode* detectou que os objetos irão ficar centralizados, porém não há regras quanto a posição vertical, altura e largura dos elementos. É necessário informar todas estas informações para remover os avisos de erro.

Ainda com todos os objetos selecionados, clique no botão *Pin*, localizado ao lado do botão *Align*:

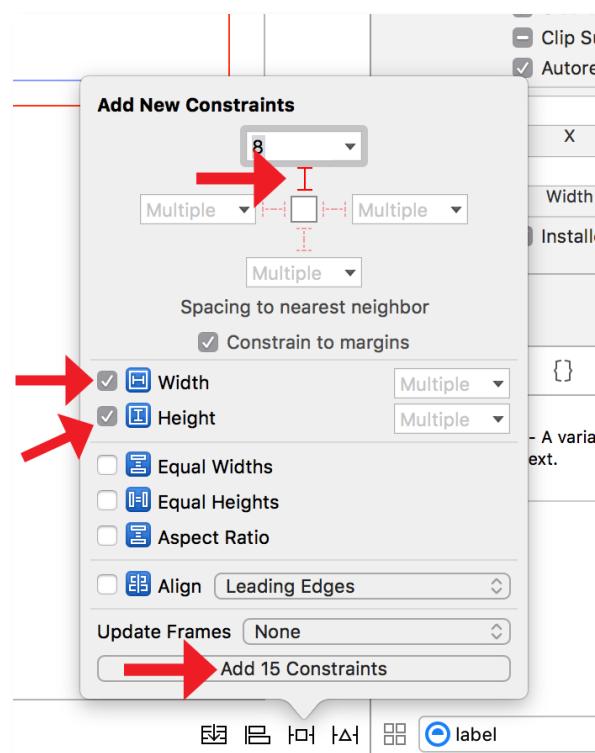
Imagen 3.34 - Clicando no botão *Pin*



Fonte: *Xcode 7.3 (2016)*

Uma janela com diversas opções irá aparecer. Clique nas seguintes opções:

Imagen 3.35 - Adicionando *constraints* verticais, de altura e largura

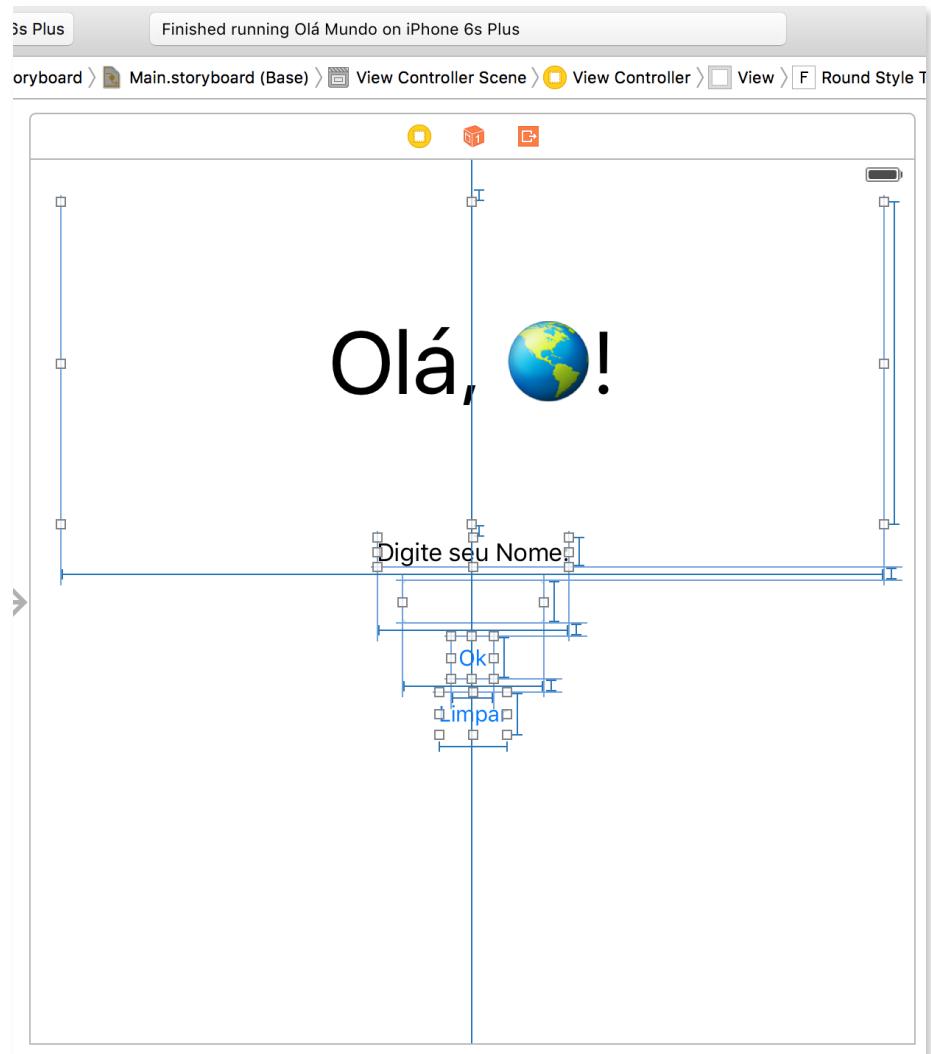


Fonte: *Xcode 7.3 (2016)*

Ao clicar na linha vermelha vertical na parte de cima da janela, as regras de posição vertical serão adicionadas à todos os objetos selecionados. Nos campos *Width* e *Height*, regras de dimensão estão sendo adicionadas, onde todos os elementos terão dimensões fixas.

Clique no botão *Add 15 Constraints* para finalizar as alterações. Note que todas as linhas que antes eram vermelhas, agora são azuis, o que indica que todos os erros foram corrigidos e o *Xcode* tem todas as informações necessárias para uma interface adaptável.

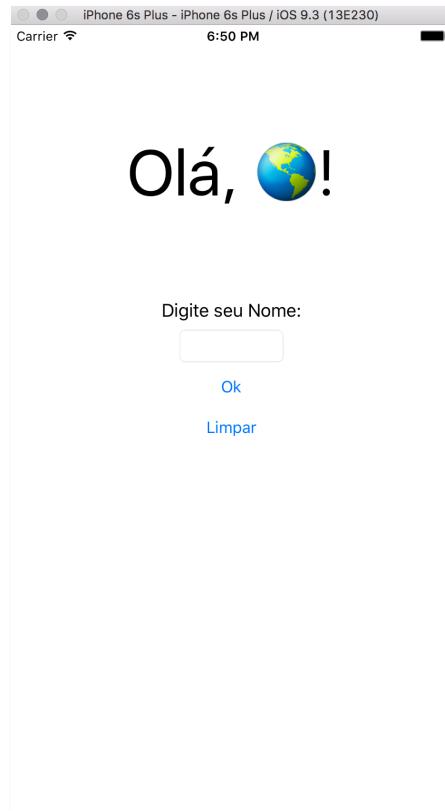
Imagen 3.36 - Linhas azuis indicando ausência de erros de interface



Fonte: *Xcode 7.3 (2016)*

Teste o aplicativo no simulador *iOS* novamente, clicando no botão *Build and Run*, localizado no canto superior esquerdo da interface do *Xcode*.

Imagen 3.37 - Simulador *iOS* executando o app “Olá Mundo”



Fonte: *Simulator* (iOS 9.3) (2016)

Desta vez, a interface está corretamente alinhada, ao contrário da primeira vez que o simulador *iOS* foi executado. Mude a orientação do simulador através do menu *Hardware* → *Rotate Left*:

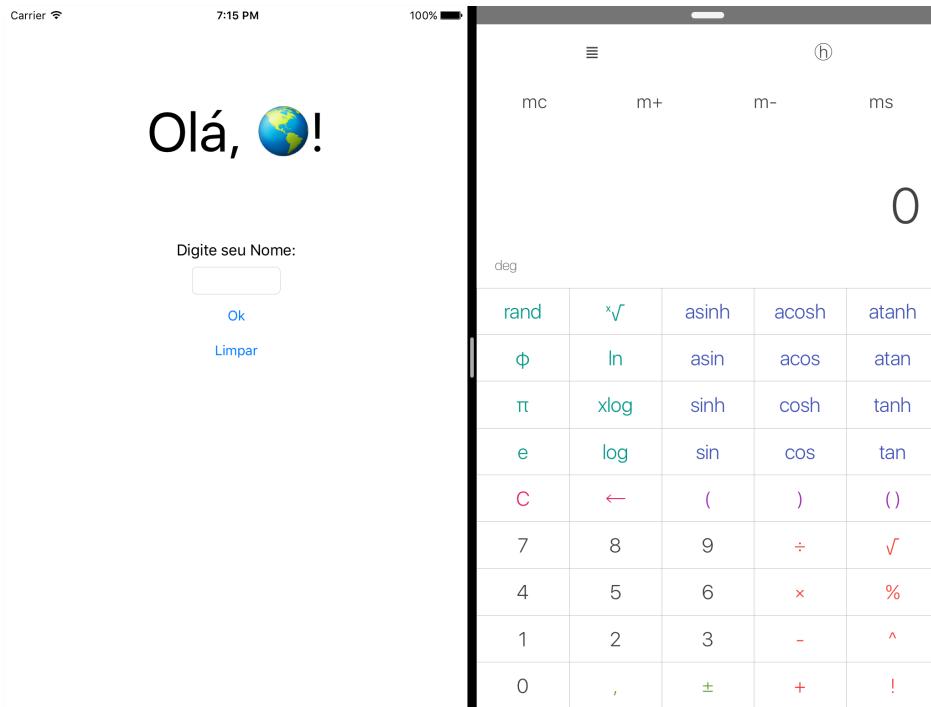
Imagen 3.38 - Mudando a orientação do Simulador *iOS*



Fonte: *Simulator* (iOS 9.3) (2016)

Observe que com a aplicação das *constraints*, a interface já suporta múltiplas orientações de tela. Na verdade, com estas regras, a interface já é compatível até mesmo com o modo multi-tarefas para *iPad*, como mostra a imagem:

Imagen 3.39 - App “Olá Mundo” em modo Multi-Tarefas para *iPad*



Fonte: *Simulator (iOS 9.3) - App “Olá Mundo” e “Calculadora Swift” (2016)*

Com isso já é possível abandonar a interface e partir para o código. O próximo capítulo irá demonstrar, passo a passo, como adicionar funcionalidade ao *app*, fazendo uso da linguagem de programação *Swift*.

5.3 Adicionando Funcionalidade ao *App*

Como a partir daqui a barra de utilidades não será utilizada, esconda ela clicando no botão *Hide or Show the Utilities*, no canto superior direito da interface do *Xcode*:

Imagen 3.40 - Escondendo o menu de utilidades do *Xcode*



Fonte: *Xcode 7.3 (2016)*

Em seguida, clique no botão *Show the Assistant Editor* para revelar o código fonte do app:

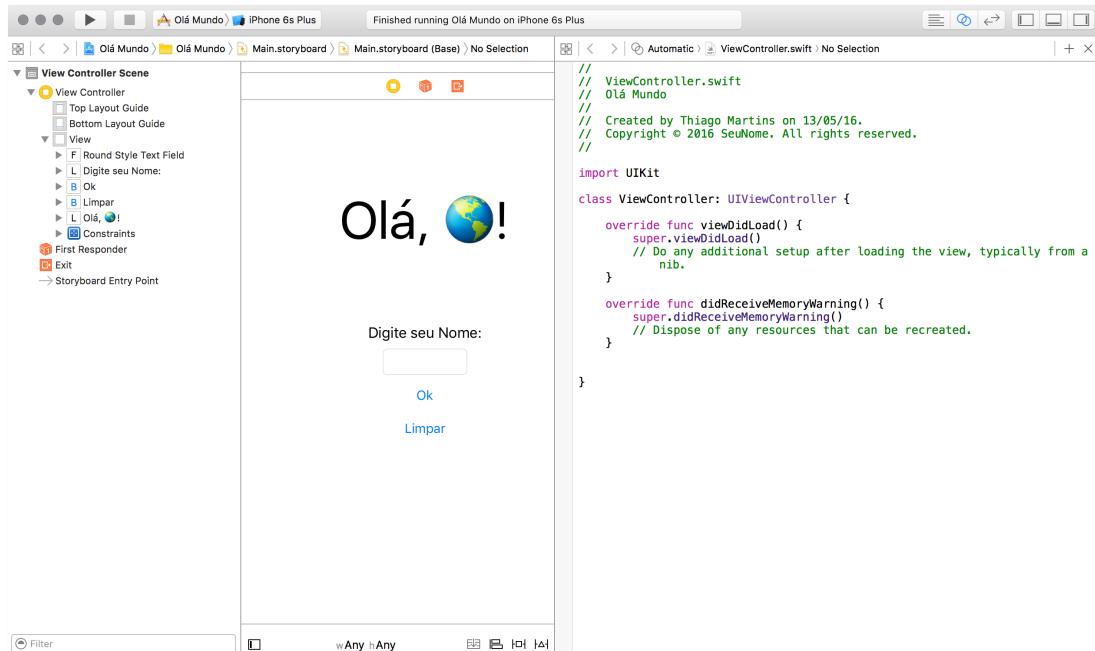
Imagen 3.41 - Revelando o código fonte do app



Fonte: Xcode 7.3 (2016)

Agora o *Xcode* está dividido em duas partes principais, onde à esquerda há a interface e ao lado direito o código fonte:

Imagen 3.42 - Interface dividida do Xcode

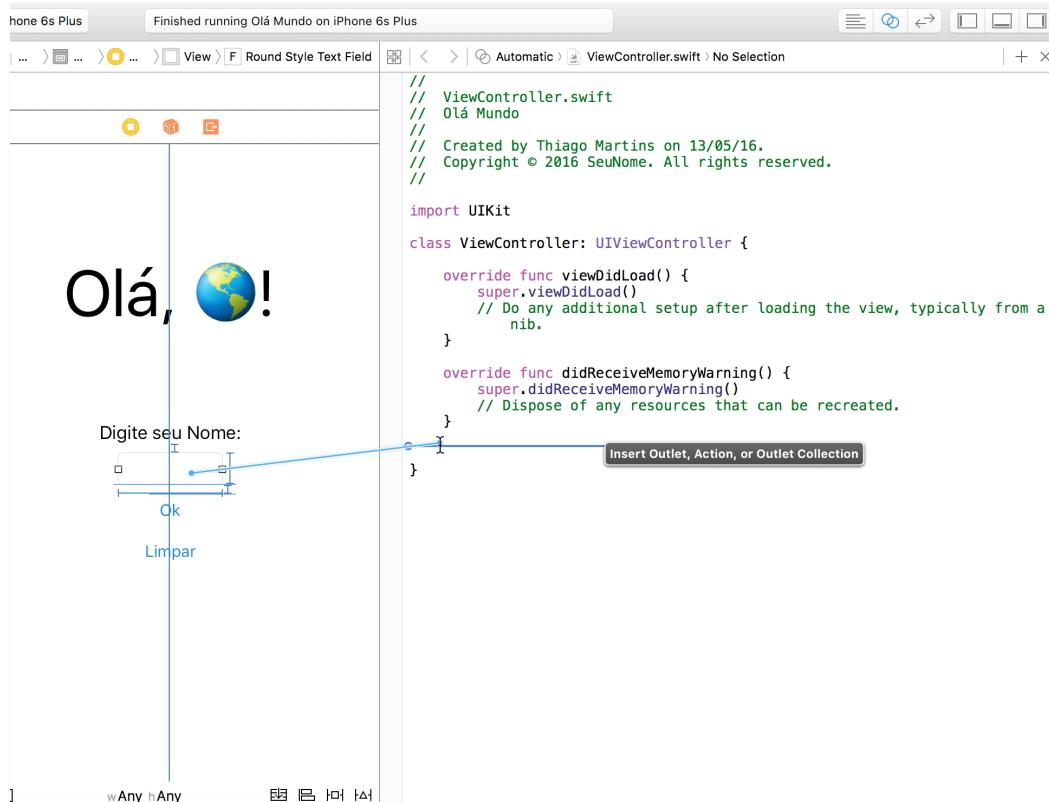


Fonte: Xcode 7.3 (2016)

Primeiramente deseja-se obter acesso à tudo que o usuário digitar no campo de texto, para isso é necessário criar um *outlet*, que é a ligação do objeto com o código. Através deste *outlet*, será possível obter diversas informações sobre o elemento de interface, inclusive o que há escrito nele.

Para criar um *outlet*, selecione o campo de texto com um clique, segure a tecla *Control* e em seguida clique e segure no objeto, uma linha irá aparecer, arraste esta linha até o código, como mostra a imagem abaixo:

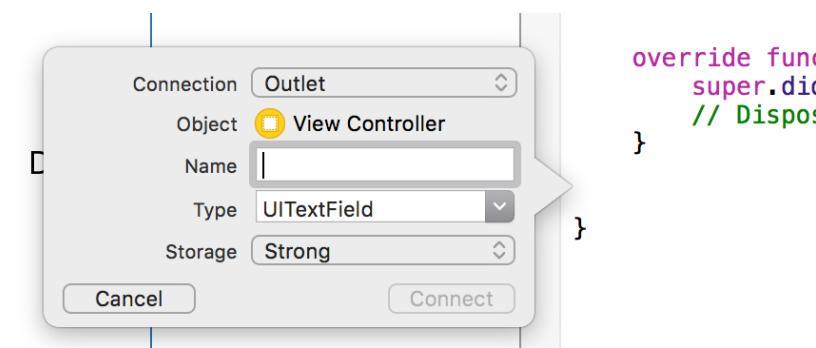
Imagen 3.43 - Criando o *outlet* do campo de texto (1/3)



Fonte: Xcode 7.3 (2016)

Ao alinhar a linha abaixo do método *didReceiveMemoryWarning*, mas ainda dentro da classe *ViewController*, solte o clique. A seguinte janela irá aparecer:

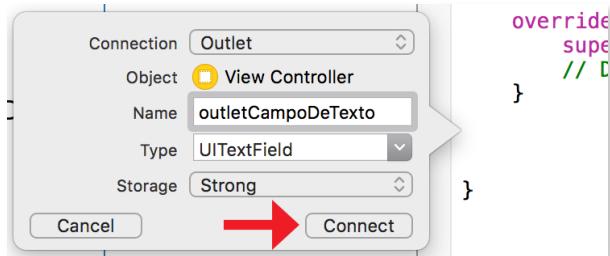
Imagen 3.44 - Criando o *outlet* do campo de texto (2/3)



Fonte: Xcode 7.3 (2016)

No campo name, digite *outletCampoDeTexto* e clique em *Connect*:

Imagen 3.45 - Criando o *outlet* do campo de texto (3/3)



Fonte: Xcode 7.3 (2016)

Após este procedimento, uma variável é adicionada ao código fonte. Esta variável representa o objeto do campo de texto, onde é possível alterar e ler diversos tipos de valores.

Imagen 3.46 - Variável *outletCampoDeTexto*

```
@IBOutlet var outletCampoDeTexto: UITextField!
```

Fonte: Código Fonte do app “Olá Mundo” / Xcode 7.3 (2016)

O próximo outlet a ser criado será o de saudação ao usuário. Selecione a etiqueta “Olá, Mundo!” e arraste ela ao código da mesma forma feita com o campo de texto. Nomeie ela para *outletEtiquetaDeSaudação*. Após este passo, o código fonte deverá se parecer com a imagem abaixo:

Imagen 3.47 - Outlets Criados para o App

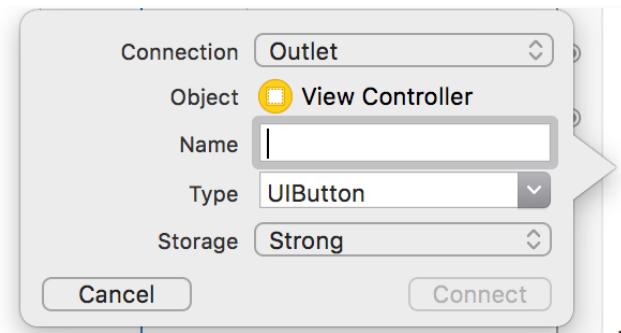
```
@IBOutlet var outletCampoDeTexto: UITextField!
@IBOutlet var outletEtiquetaDeSaudação: UILabel!
```

Fonte: Xcode 7.3 (2016)

Agora é necessário definir ações para os botões “Ok” e “Limpar”. O processo de criação de ações é semelhante ao de *outlets*.

Selecione o botão “Ok”, segure e arraste a linha para o código, a janela de criação de conexões irá novamente aparecer:

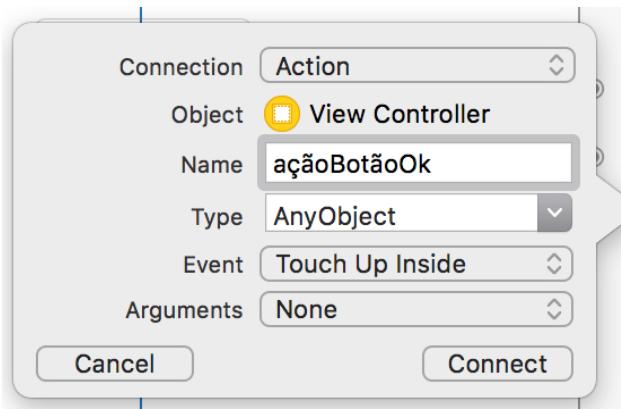
Imagen 3.48 - Criando a ação do botão “OK” (1/2)



Fonte: Xcode 7.3 (2016)

Neste caso não deseja-se um *outlet*, mas sim uma ação. Clique no campo *Connection*, selecione a opção “Action”:

Imagen 3.49 - Criando a ação do botão “OK” (2/2)



Fonte: Xcode 7.3 (2016)

Em “Name”, digite “*açãoBotãoOk*”. Já que o método não necessitará de um parâmetro neste *app*, selecione a opção “*None*” no campo *Arguments*, como mostra a imagem acima.

Clique em *Connect*. Um novo método será adicionado ao código fonte:

Imagen 3.50 - Código do método *açãoBotãoOk*

```
@IBAction func açãoBotãoOk() {  
}
```

Fonte: Xcode 7.3 (2016)

Este método será executado sempre que botão “Ok” for tocado pelo usuário. Realize o mesmo procedimento para o botão “Limpar”, nomeando seu método como *açãoBotãoLimpar*, não esquecendo de remover os argumentos no campo *Arguments*. Após a ligação, o código abaixo do método *didReceiveMemoryWarning* será o seguinte:

Imagen 3.51 - Código fonte dos *outlets* e ações criados até então

```
@IBOutlet var outletCampoDeTexto: UITextField!
@IBOutlet var outletEtiquetaDeSaudação: UILabel!
@IBAction func açãoBotãoOk() {
}
@IBAction func açãoBotãoLimpar() {
}
```

Fonte: Xcode 7.3 (2016)

Todos os *outlets* e ações já foram declarados, restando apenas definir as ações que tornaram o *app* funcional.

Sempre que o usuário tocar no botão “Ok”, o nome que estiver no campo de texto será levado para a etiqueta de saudação, por exemplo: se o usuário digitar o nome “João” e clicar no botão “Ok”, a etiqueta de saudação deverá exibir a mensagem “Olá, João!”.

Dentro do escopo do método *açãoBotãoOk*, adicione a seguinte linha de código:

Imagen 3.52 - Linha de código do método *açãoBotãoOk*

```
outletEtiquetaDeSaudação.text = "Olá, " + outletCampoDeTexto.text! + "!"
```

Fonte: Xcode 7.3 (2016)

O código acima usa os *outlets* criados para atribuir uma *String* ao atributo *outletEtiquetaDeSaudação.text*, que é o texto da etiqueta de saudação.

A atribuição consiste na concatenação de três *Strings*, a primeira sendo “Olá, “, a segunda sendo o conteúdo do campo de texto, e a terceira sendo “!”.

Agora só resta aplicar a ação do botão “Limpar”. Este botão irá apagar qualquer texto presente no campo de texto, e retornar a etiqueta de saudação à seu estado original.

Adicione as seguinte linhas de código para dentro do escopo do método *açãoBotãoLimpar*:

Imagen 3.53 - Linhas de código do método *açãoBotãoLimpar*

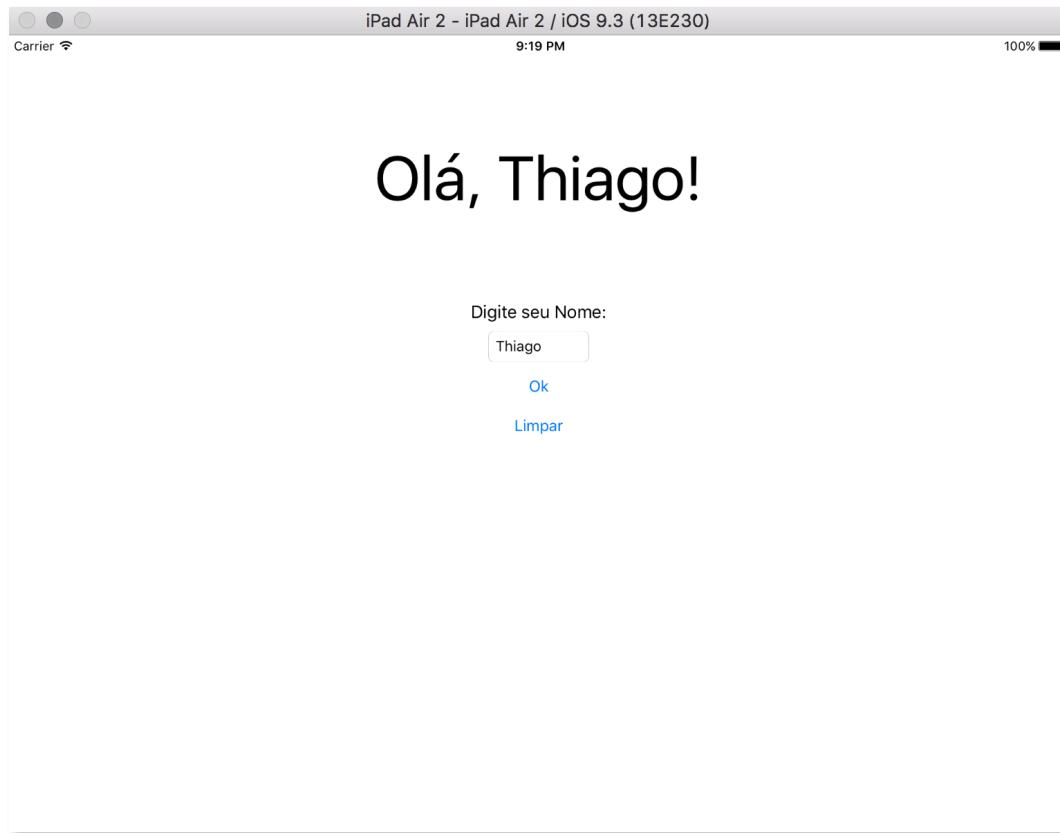
```
outletCampoDeTexto.text = ""
outletEtiquetaDeSaudação.text = "Olá, !"
```

Fonte: Xcode 7.3 (2016)

A primeira linha simplesmente atribui uma *String* vazia para o conteúdo do campo de texto. A segunda linha atribui a *String* “Olá, Mundo!” para o texto da etiqueta de saudação.

Com isso, o *app* “Olá Mundo” está concluído. Execute o simulador para testar suas funcionalidades:

Imagen 3.54 - Executando a versão final do *app* “Olá Mundo”



Fonte: Simulator (iOS 9.3) - App “Olá Mundo” (2016)

6. Conclusão

Nos últimos anos houve uma grande mudança na forma em que computadores pessoais funcionam, com o forte investimento das maiores empresas do mundo em computadores portáteis, hoje é seguro concluir que o mercado *mobile* é o mais rentável para empresas como a Apple. Ferramentas de desenvolvimento de alta qualidade e documentações são disponibilizadas gratuitamente para desenvolvedores com o intuito de investir neste mercado em constante crescimento.

A alta disponibilidade de ferramentas e documentações, somadas com a simplicidade da nova linguagem de programação *Swift*, cria um ambiente extremamente confortável para novos desenvolvedores, onde o aprendizado é rápido e divertido. Muitos problemas clássicos da programação, como acentuação de caracteres e manipulação de *Strings* foram abstraídos pela linguagem *Swift*, permitindo que o desenvolvedor invista mais de seu tempo em criatividade ao invés de sintaxe. *Swift* também é uma linguagem extremamente rápida e eficiente, totalmente inspirada nas linguagens C e *Objective-C*. Por estes fatos sou levado a acreditar que a linguagem de programação *Swift* terá um papel fundamental na formação das novas gerações de programadores, e espero que sua crescente adoção em universidades contribua para a evolução da computação e gere um maior interesse das pessoas de todo o mundo em desenvolvimento de *software*.

Com a disponibilização gratuita de uma gigante biblioteca de objetos e métodos, conclui-se que o desenvolvimento para *iOS* é extremamente orientado a objeto. Este paradigma de programação foi completamente aparente desde o início deste trabalho, já que todos os elementos do aplicativo como botões, campos de texto e contêineres são basicamente objetos pré programados pela Apple, com métodos e documentações prontos. Basta arrastar um objeto para o projeto e ele estará pronto para uso, restando apenas aprender como tratar-los. Ao final deste projeto fui levado a concluir que o entendimento básico do paradigma de orientação a objeto é um requisito essencial para o desenvolvimento *iOS*.

O fato de a Calculadora *Swift* ser um projeto de código aberto, pode incentivar novos desenvolvedores a estudar seu código, ou até mesmo obter contribuições de programadores mais experientes do mundo inteiro. Projetos de código aberto, apesar de a princípio não serem tão rentáveis, são de suma importância para o aumento do padrão de qualidade de todo *software*, afinal, usuários normalmente não irão pagar por um produto inferior a algo disponibilizado gratuitamente. Dito isto, pode-se concluir que o projeto

Calculadora *Swift* apenas começou, possuindo uma grande ambição: se tornar a melhor calculadora da *App Store*.

Mesmo sendo uma calculadora completamente funcional, praticamente todo o seu funcionamento foi baseado em manipulação de *Strings*, com breves conversões *Double* (*Float 64-bits*). Com a existência de incontáveis métodos de alto nível de manipulação de *Strings* já prontos, foi relativamente simples criar um motor de cálculo preciso, onde os maiores desafios do projeto basicamente consistiram em criar funções personalizadas através de extensões do tipo *NSNumber*, e formata-las corretamente. Ao fim do desenvolvimento do motor de cálculo da calculadora, foi possível concluir que o desenvolvimento para *iOS* é de alto nível, e que a leitura das documentações é de extrema importância para tirar máximo proveito de suas poderosas funcionalidades.

A interface da Calculadora *Swift* suporta diversos tamanhos e formatos de tela, inclusive sendo compatível com as novas funcionalidades de multi-tarefas disponível nos novos *iPads*. A implementação de interfaces adaptáveis mostrou-se relativamente simples no desenvolvimento para *iOS*. Concluo que a compreensão dos conceitos de *constraints* são necessárias para alcançar uma interface sólida, onde a Calculadora *Swift* obteve êxito.

Apesar do gigantesco número de *apps* já criados para *iOS* e *Android*, é sempre possível questionar a qualidade da maioria deles. A afirmação “já existe um *app* que faz isso” pode desanimar um desenvolvedor, porém é importante lembrar que sempre há espaço para melhorias. As lojas de *apps* estão cheias de aplicativos mal programados, superar eles não é difícil. Como demonstrado na comparação entre as calculadoras mais populares da *App Store*, é possível concluir que a Calculadora *Swift* superou elas em diversos aspectos.

A *IDE Xcode* é uma ferramenta de desenvolvimento gratuita e poderosa. Como demonstrado na criação do *app* “Olá, Mundo!”, a criação de interfaces adaptáveis e compatíveis com múltiplos dispositivos é relativamente simples. Com os conceitos explicados no passo a passo deste estudo, é possível compreender a essência do funcionamento da *IDE Xcode* e do desenvolvimento para *iOS*.

Por fim, concluo que este trabalho foi de grande importância para minha pessoa, ampliando meu conhecimento em diversas áreas, desde programação até criatividade. No início deste estudo, não possuía experiência alguma com desenvolvimento mobile, porém graças à abundante quantidade de documentos, livros e informações disponíveis gratuitamente, somados com a simplicidade da linguagem de programação *Swift*, obtive

sucesso em finalizar meu primeiro aplicativo. Espero que mais pessoas se interessem no tema, e tirem o máximo proveito deste trabalho acadêmico.

REFERÊNCIAS

APPLE INC. ***The Swift Programming Language (Swift 2.2)***. Cupertino, CA, 2016.

APPLE INC. **About Swift**. Disponível em: <https://swift.org/about/>. Acesso em 15/03/2016.

APPLE INC. **Swift Has Reached 1.0**. Disponível em: <https://developer.apple.com/swift/blog/?id=14>. Acesso em 22/03/2016.

APPLE INC. **Swift. Uma linguagem aberta e poderosa, para todo mundo criar apps incríveis**. Disponível em: <http://www.apple.com/br/swift/>. Acesso em 19/03/2016.

APPLE INC. **Swift Programming Language Evolution**. Disponível em: <https://github.com/apple/swift-evolution>. Acesso em 09/05/2016.

APPLE INC. **Remove the ++ and -- operators**. Disponível em: <https://github.com/apple/swift-evolution/blob/master/proposals/0004-remove-pre-post-inc-decrement.md>. Acesso em 15/05/2016.

HUDSON, PAUL. **What's new in Swift 2.2**. Disponível em: <https://www.hackingwithswift.com/swift2-2>. Acesso em 22/03/2016.

APPLE INC. **Xcode IDE**. disponível em: <https://developer.apple.com/xcode/ide/>. Acesso em 15/03/2016.

APPLE INC. **About Simulator**. Disponível em: https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/iOS_Simulator_Guide/Introduction/Introduction.html. Acesso em 15/03/2016.

HEGARTY, PAUL. **Developing iOS 8 Apps with Swift**. Disponível em: <https://www.youtube.com/playlist?list=PLy7oRd3ashWodnpf8rjfYEkTgwbOEskfU>. Acesso em 15/03/2016.

APPLE INC. **Apple Developer Program - What you Need to Enroll**. Disponível em: <https://developer.apple.com/programs/enroll/>. Acesso em 12/05/2016.

NEDRICH, MATT. **Evaluating Expressions in iOS with Objective-C and Swift.** Disponível em: <https://spin.atomicobject.com/2015/03/24/evaluate-string-expressions-ios-objective-c-swift/>. Acesso em 20/03/2016.

APPLE INC. **UIStackView.** Disponível em: https://developer.apple.com/library/prerelease/ios/documentation/UIKit/Reference/UIStackView_Class_Reference/. Acesso em 15/03/2016.

APPLE INC. **Popovers.** Disponível em: <https://developer.apple.com/library/ios/documentation/WindowsViews/Conceptual/ViewControllerCatalog/Chapters/Popovers.html>. Acesso em 27/03/2016.

APPLE INC. **About Scroll View Programming.** Disponível em: https://developer.apple.com/library/ios/documentation/WindowsViews/Conceptual/UIScrollView_pg/Introduction.html. Acesso em 31/03/2016.

APPLE INC. **Creating and Configuring Scroll Views.** Disponível em: https://developer.apple.com/library/ios/documentation/WindowsViews/Conceptual/UIScrollView_pg/CreatingBasicScrollViews/CreatingBasicScrollViews.html. Acesso em 27/03/2016.

APPLE INC. **Adopting Multitasking Enhancements on iPad.** Disponível em: <https://developer.apple.com/library/prerelease/ios/documentation/WindowsViews/Conceptual/AdoptingMultitaskingOniPad/index.html>. Acesso em 15/03/2016.

APPLE INC. **Technical Note TN2418, Language Identifiers in iOS 9.** Disponível em: https://developer.apple.com/library/ios/technotes/tn2418/_index.html. Acesso em 09/05/2016.

TEX TEXIN, JOHN COWAN. **Using Language Identifiers (RFC 3066).** Disponível em: <http://www.i18nguy.com/unicode/language-identifiers.html>. Acesso em 11/05/2016.

APPLE INC. **O que é o iOS?.** Disponível em: <http://www.apple.com/br/ios/what-is/>. Acesso em 15/03/2016.