

iOS Swift - Corporate

Parte 2



iOS Swift - Corporate Parte 2



Copyright © Quaddro Treinamentos Empresariais LTDA

Todos os direitos autorais reservados. Este manual não pode ser copiado, fotocopiado, reproduzido, traduzido ou convertido em qualquer forma eletrônica, ou legível por qualquer meio, em parte ou no todo, sem a aprovação prévia, por escrito, da Quaddro Treinamentos Empresariais Ltda, estado o contrafator sujeito a responder por crime de Violação do Direito Autoral, conforme o art.184 do Código Penal Brasileiro, além de responder por Perdas e Danos. Todos o logotipos e marcas utilizados neste material pertencem às suas respectivas empresas.

Autoria:

Tiago Santos de Souza
Roberto Rodrigues Junior

Revisão:

Roberto Rodrigues Junior
Tiago Santos de Souza

Design, edição e produção:

Tiago Santos de Souza

“As marcas registradas e os nomes comerciais citados nesta obra, mesmo que não sejam assim identificados, pertencem aos seus respectivos proprietários nos termos das leis, convenções e diretrizes nacionais e internacionais.”

Edição nº1
Fevereiro de 2018



Apresentação.....3

Capítulo 1 - Testes Unitários.....5

Seção 1-1 - Por que criar testes?.....7

Seção 1-2 - Novo projeto com Testes Unitários.....8

Seção 1-3 - Framework XCTest.....9

Seção 1-4 - Classes XCTest e XCTestCase.....10

Seção 1-5 - Afirmar condições de teste.....12

Seção 1-6 - Executando testes.....14

Seção 1-7 - Code Coverage.....17

Capítulo 2 - Testes de Interface.....19

Seção 2-1 - Testes de Interface.....21

Seção 2-2 - Novo projeto com Testes de UI.....22

Seção 2-3 - Gravando testes.....23

Seção 2-4 - Utilizando o Accessibility.....26

Capítulo 3 - Xcode com Git.....28

Seção 3-1 - Introdução.....30

Seção 3-2 - Criação de repositórios.....32

Seção 3-3 - Criando projeto no Xcode.....34

Seção 3-4 - Trabalhando com o Git.....36

Seção 3-5 - Trabalhando de forma colaborativa....39

Capítulo 4 - Publicando na App Store.....44

Seção 4-1 - Entendendo o processo.....46

Seção 4-2 - Programa de Desenvolvedor.....48

Seção 4-3 - Configurações para Desenv.....50

Seção 4-4 - Desenv. e Conf. do App.....59

Seção 4-5 - Submetendo seu aplicativo.....61



Todos nascemos para a programação com um **Hello World!** Independente da linguagem, das estruturas, das sintaxes. Quando falamos de **Swift**, não é diferente.

Evoluímos nas sintaxes, nos aprofundamos no Core da linguagem, desvendamos os segredos dos elementos de interface gráfica. Foundation, UIKit, e depois outros Frameworks nativos que muitas vezes não podem faltar em nossas aplicações para iOS.

Requisições para Webservices, modelagem de bancos de dados, e assim por diante!

Porém quando falamos do mundo corporativo, muitas ferramentas são citadas com frequência e percebemos que talvez tenhamos andado menos da metade do nosso caminho.

Nomes como **Testes Unitários**, **Testes de Interface**, ou Frameworks de terceiros como **Alamofire**, **Google Maps** ou **Realm** são assuntos recorrentes nas rodas de bate papo entre desenvolvedores.

Esse curso tem o intuito de nos transportar para esses novos planetas, de nos deixar por dentro de tudo de mais atual e usual no mundo corporativo de desenvolvimento para iOS.

Vamos nos preparar para utilizar as ferramentas mais poderosas do mercado que vão desde a criação de bancos de dados mais rápidos e inteligentes, prontos para integração com Webservices, até o trabalho de versionamento de projetos para desenvolvimento em grupo.

Talvez o melhor código para descrever o momento seja:

```
let novoObjetivo = "Corporate"  
print("Hello \(novoObjetivo) World!")
```




Testes Unitários

Seção 1-1

Por que criar testes?



As vezes passamos horas a frente de um app, desenvolvendo seus códigos, debugando erros, etc. Rodamos a aplicação, e mais uma vez de forma inesperada, navegando pela interface, simplesmente descobrimos mais um erro.

Enquanto esses erros ocorrem conosco, está tudo certo. Mas esse tipo de problema pode vazar e acabar acontecendo nas mãos do seu usuário final.

Quando criamos testes, minimizamos esse tipo de situação. Com testes podemos fazer uma varredura por tudo o código em busca de possíveis erros, principalmente aqueles que podem ocorrer de forma inesperada, e não são identificados pelo debug da IDE.

O nome **Teste Unitário** já nos dá uma ideia do que podemos fazer com o recurso. Podemos testar cada unidade do nosso código, como as propriedades e métodos nas nossas classes.

Em um primeiro momento, podemos ter a sensação de que desenvolver testes para cada unidade de código pode gerar um trabalho duplicado, porém a garantia de um código testado, bem escrito e funcional nos provê mais agilidade do que reescrever vários trechos em busca da solução de um erro, as vezes simples de ser resolvido.

No mundo corporativo, em algumas empresas, como procedimento adota-se uma porcentagem de códigos a serem cobertos por testes unitários, garantindo assim, o mínimo de confiabilidade necessária para a publicação de um projeto.

Mesmo trabalhando de forma independente é importante criar testes. Garantir a qualidade das unidades de código, proporciona um desenvolvimento mais ágil e as possibilidades de manutenções mais tranquilas.

Os tipos de testes

Além dos **Testes Unitários**, que validam as unidades de código, temos a disposição os **Testes de Interface**. Com os Testes de Interface, podemos gerar códigos que automatizam a operação de uma aplicação.

A função principal dos Testes de Interface é validar o comportamento da interface da aplicação e verificar se os elementos respondem como o esperado.

O Xcode dispõe de recursos nativos tanto para criação de Testes unitários quanto Testes de Interface.

Seção 1-2

Novo projeto com Testes Unitários



Podemos implementar a utilização de Testes Unitários logo na criação de um projeto, os passos a seguir descrevem o procedimento.

1. Crie um novo projeto **Single View Application**;
- 2, Digite o nome desejado para o projeto em **Product Name**;
3. Na tela de opções, selecione **Include Unit Tests**.

Choose options for your new project:

Product Name: testes-unitarios

Team: None

Organization Name: Quaddro

Organization Identifier: br.com.quaddro

Bundle Identifier: br.com.quaddro.testes-unitarios

Language: Swift

Devices: Universal

☐ Use Core Data

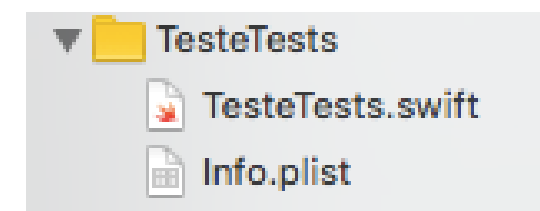
☒ Include Unit Tests

☐ Include UI Tests

Cancel Previous Next

Estrutura básica da classe de teste

Observe que, ao selecionar a opção de inclusão de testes unitários, o Xcode cria um novo grupo para **nomeProjetoTests**.



Dentro desse grupo, temos um arquivo de classe, com o mesmo nome do seu projeto, mais um prefixo **Tests.swift**. Ao abrir essa classe, podemos notar o seguinte:

- Temos importado o **framework XCTest**, que fornece a estrutura necessária para realização de testes
- A instrução **@testable import nomeProjeto**, que indica o projeto foco dos testes unitários;
- Dentro da classe, temos os métodos **setUp()** e **tearDown()**, que atuam antes e depois de todos os testes;
- Outros métodos de exemplo: **testExample()** e **testPerformanceExample()**, que podem ser retirados e substituídos por métodos relacionados ao projeto.

Seção 1-3

Framework XCTest



Utilizamos o framework **XCTest** para integrar comportamentos de testes ao nosso aplicativo. Com o mesmo framework podemos executar tanto testes unitários como testes de interface.

Temos que ter em mente, que dentro do arquivo de classe de teste, podemos adotar os mesmos comandos utilizados no desenvolvimento normal. Estamos falando de comandos do framework Foundation. Podemos condicionar, trabalhar com loops ou criar novas funções na nossa classe de testes, assim como em uma outra classe qualquer.

Alinhado ao conjunto de comandos mais usuais, podemos utilizar as classes e métodos do framework XCTest para fazer com que os testes sejam realizados.

Por ser um framework, o XCTest deverá ser incluído ao nosso projeto na, portanto, quando for necessário, devemos importar o framework logo no início do nosso código de cada uma das classes que realizarão testes.

```
//Início do Projeto
```

```
//Importando o Safari Services
```

```
import XCTest
```

Seção 1-4

Classes XCTest e XCTestCase



Herança: NSObject > XCTest > XCTestCase

Todos os testes basicamente são baseados em uma classe de testes. Essa base encontra-se na classe **XCTest**.

XCTest é uma classe base abstrata que fornece funcionalidades compartilhadas e utilizadas por **XCTestCase** para criar, gerenciar e executar testes. Como vimos ao inicializar com projeto com testes, o arquivo de classe utilizado para gerenciar os testes é uma subclasse de **XCTestCase**.

Principais métodos

A classe **XCTest** possui dois métodos principais, que são de suma importância na execução de testes, estamos falando dos testes **setUp()** e **tearDown()**

```
class func setUp()  
//Método disparado antes de um teste ser executado. Podemos  
método para personalizar o estado inicial para cada teste.
```

```
class func tearDown()  
//Método disparado depois de um teste ser executado. Podemos  
método para limpar definições que um teste tenha imposto o  
estado final para cada teste.
```

Definindo casos de teste e métodos de teste

Adicionamos testes ao projeto escrevendo um ou mais métodos de teste.

Cada um dos métodos de teste verifica um aspecto específico do seu código. Para adicionar testes ao projeto devemos:

- Importar o framework **XCTest** de testes, criar um alvo de teste **@testable import** e uma nova subclasse do **XCTestCase**, caso não existam:

- Adicione uma propriedade de acesso a classe de View Controller que deseja testar:

- Aproveite os métodos **setUp()** e **tearDown()** sempre que necessário;

- Adicione um ou mais métodos de teste ao caso;

- Adicione uma ou mais métodos de afirmações de teste dentro de cada método do teste.

Confira abaixo a estrutura base de uma classe de testes:

```
import XCTest
@testable import Teste

class TesteTests: XCTestCase {

    //MARK:- Propriedades
    // Necessário para instanciar a View Controller
    let acessoVC = ViewController()

    //MARK:- Métodos de entrada e saída de testes
    override func setUp() {
        super.setUp()
    }

    override func tearDown() {
        super.tearDown()
    }

    //MARK:- Métodos próprios de teste
    func testeExemplo() {

        let condicao = true

        XCTAssertTrue(condicao, "Mensagem de Falha")
    }
}
```

Seção 1-5

XCTAssert - Afirmar condições de teste



Podemos verificar (ou afirmar) as condições dentro dos métodos de teste para certificar que o código está se comportando conforme o esperado.

Utilizamos a família de funções **XCTAssert** para verificar condições booleanas, valores nulos ou não nulos, valores esperados e erros lançados. Esse conjunto de funções basicamente define a estrutura de testes unitários, nos dando amplo espectro dentro do código testado para verificar diferentes situações.

Afirmações Booleanas

Testa uma condição que gere um resultado verdadeiro ou falso.

```
//Teste que valida o verdadeiro
XCTAssertTrue(expressão, "Mensagem de Falha")

//Teste que valida o falso
XCTAssertFalse(expressão, "Mensagem de Falha")
```

Afirmações Nil e Non-Nil

Verifica se uma condição de teste é nula ou não nula.

```
//Teste que valida nil
XCTAssertNil(expressão, "Mensagem de Falha")

//Teste que valida o Non-nil
XCTAssertNotNil(expressão, "Mensagem de Falha")
```

Afirmações Igualdade de Desigualdade

Verifica se dois valores são iguais ou desiguais.

```
//Teste que valida a igualdade
XCTAssertEqual(expressão1, expressão2, "Mensagem de Falha")

//Teste que valida a falta de igualdade
XCTAssertNotEqual(expressão1, expressão2, "Mensagem de Falha")
```

Afirmações comparação de valores

Comparação de dois valores para determinar se um é maior ou menor que o outro.

```
//Teste que valida o maior ou igual
XCTAssertGreaterThanOrEqual(expressão1, expressão2, "Mensagem de Falha")

//Teste que valida o menor ou igual
XCTAssertLessThanOrEqual(expressão1, expressão2, "Mensagem de Falha")

//Teste que valida o maior que
XCTAssertGreaterThan(expressão1, expressão2, "Mensagem de Falha")

//Teste que valida o menor que
XCTAssertLessThan(expressão1, expressão2, "Mensagem de Falha")
```

Afirmações de erros

Verifica se uma chamada de função lança ou não um erro.

```
//Método de valida o Erro  
XCTAssertThrowsError()
```

```
//Método de valida a falta de Erro  
XCTAssertNoThrow(expressão)
```

Seção 1-6

Executando testes



Executamos os testes diretamente na janela de códigos do Xcode. Podemos executar um teste de cada vez, ou todos os testes que compõem uma classe. Tudo depende da necessidade atual da tarefa que se executa.

Quando não executamos nenhum teste na classe, ao lado de sua declaração aparece um ícone no formato de um losango:

```
◇ class TestesUnitariosBasicoTests: XCTestCase {
```

O mesmo ícone aparece em algum método de teste que também não foi testado:

```
◇ func testeVerdadeiroFalso(){
```

Quando posicionamos o cursor do mouse sobre esses ícones, temos a possibilidade de executar todos os testes, quando indicado ao lado da classe, ou apenas os testes de um método específico, quando indicado ao lado de uma função:

```
▶ class TestesUnitariosBasicoTests: XCTestCase
```

```
▶ func testeVerdadeiroFalso(){
```

Quando executamos apenas alguns testes com sucesso dentro da classe, os seguintes ícones aparecem para a classe e métodos:

```
◀ class TestesUnitariosBasicoTests: XCTestCase {
```

```
✔ func testeVerdadeiroFalso(){
```

Quando executamos todos os testes de uma classe com sucesso, todos os ícones aparecerão verdes com um tick branco:

```
✔ class TestesUnitariosBasicoTests: XCTestCase {
```

```
✔ func testeVerdadeiroFalso(){
```

#Dica!

Podemos repetir os testes quantas vezes forem necessárias. Testes inválidos não impedirão a compilação e execução da aplicação, mesmo quando alertas de debug aparecerem.

Provocando erros nos testes

Podemos **provocar** erros em nossos testes com intuito de verificar se o teste identificará determinada situação. O erro pode ser provocado de duas formas:

- Invertendo o retorno da função de origem;

- **Modificando o parâmetro da função de teste:** colocando um valor false onde se esperava um true por exemplo.

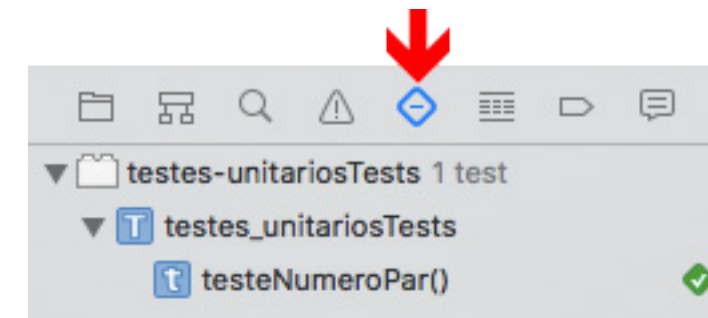
Quando acontecem erros, o teste não é validado e ícones indicarão a região a ser analisada:

```
❌ func testeVerdadeiroFalso(){
40
41     let propriedadeVerificada = false
42
43     //Teste que valida o verdadeiro
44     XCTAssertTrue(propriedadeVerificada, "Teste Verdadeiro Inválido")
45 }
```

Test Navigator

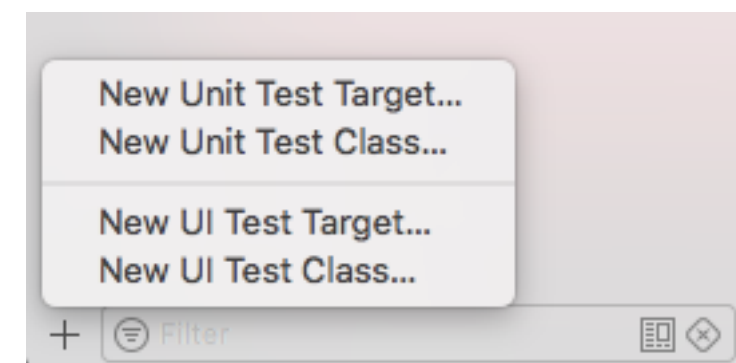
Além dos ícones que nos proporciona a verificação visual dos resultados dos testes na própria classe, o Xcode provê outras formas de visualização que nos auxilia, especialmente em projetos maiores, que podem conter testes distribuídos em diferentes classes.

Inicialmente temos o painel **Test Navigator**, localizado na mesma barra de tarefas de navegação do projeto:



Através do **Test Navigator**, podemos navegar pelos diferentes métodos de testes contidos na respectiva classe, e também dispomos dos ícones de execução e verificação das condições dos testes.

Nesse mesmo painel, dispomos de comandos para criação de novas classes de teste, para integração com outras classes da nossa aplicação, ou ainda a criação de testes para projetos que não foram criados com a opção Unit Tests ativada:



Report Navigator

Outra alternativa onde pode-se verificar com detalhes quais foram as falhas ocorridas, é painel **Report Navigator**.

Nele temos a disposição um **log** com histórico das últimas operações realizadas, incluindo os testes.



Seção 1-7 Code Coverage



Code Coverage é uma ferramenta do Xcode que permite visualizar e medir qual é a porcentagem de seu código que está sendo testada. Com a Code Coverage, podemos determinar se os testes estão fazendo o trabalho desejado em termos de cobertura com testes.

Por ser uma ferramenta visual, fica mais fácil entender quais são as áreas que estão sem cobertura de teste.

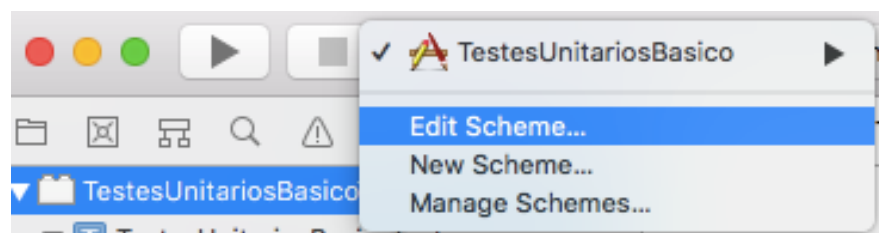
#Dica!

No mundo corporativo é comum equipes de desenvolvimento serem cobradas para atingir determinadas porcentagens de coberturas com testes. Essa ferramenta auxilia muito nesses casos.

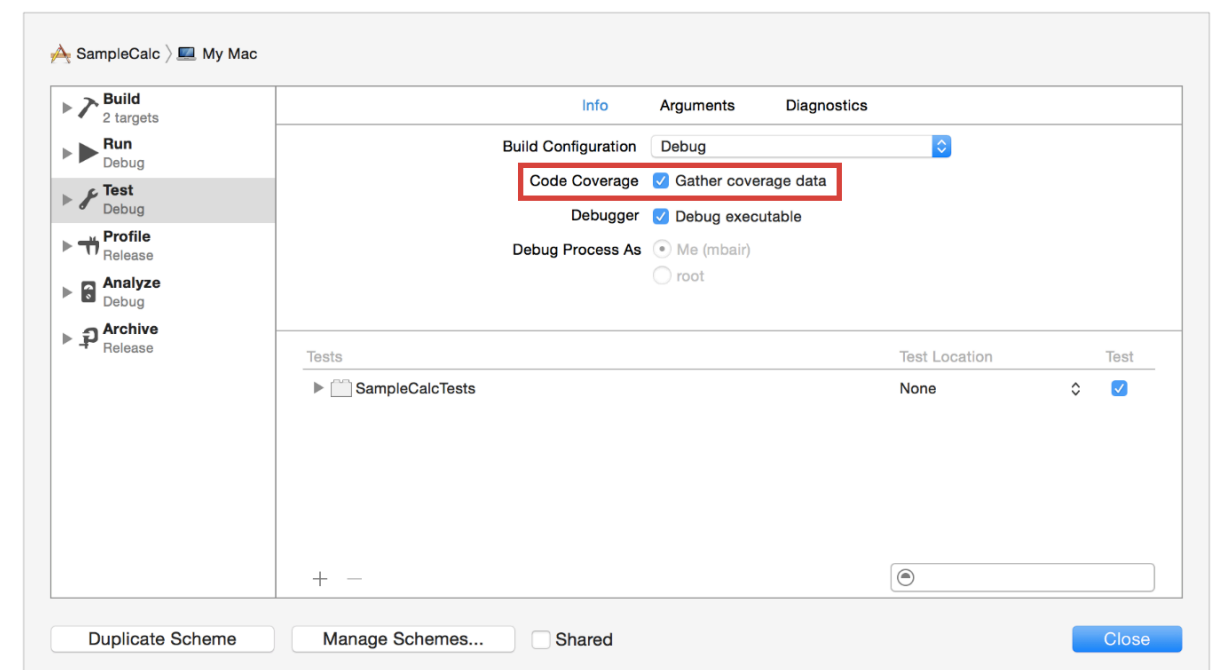
Habilitando o Code Coverage

O Code Coverage não está habilitado no início do projeto. Vamos aos passos para habilitar a ferramenta:

1. Clique sobre o botão **Set active scheme**, ao lado dos botões que rodam a aplicação, e escolha a opção **Edit Scheme**:



2. Na janela exibida, escolha a opção **Test** no menu a esquerda, e habilite a caixa **Gather coverage data**. Finalize com o botão **Close**.



Acessando o Code Coverage

Agora podemos acessar o Code Coverage e verificar qual é a porcentagem de códigos do projeto cobertos por testes.

Para fazer essa verificação, devemos rodar todos os testes de uma classe, em seguida, vamos nos dirigir ao painel **Report Navigator** e escolher o ultimo teste realizado.

Ao escolher o teste, na janela exibida temos a opção **Coverage**. Nela podemos navegar pelos métodos e verificar a porcentagem de códigos coberta por testes:

The screenshot shows the Xcode interface with the 'TestesUnitariosBasico' project selected. The 'Test Hoje 18:14' test is highlighted in the left sidebar. The 'Coverage' tab is active in the right pane, showing a table of code coverage for the selected test.

Name	Coverage
TestesUnitariosBasico.app	72%
ViewController.swift	100%
TestesUnitariosBasico.ViewController.iniciarTeste() -> ()	100%
TestesUnitariosBasico.ViewController.finalizarTeste() -> ()	100%
TestesUnitariosBasico.ViewController.testando(nome: Swift.String	100%
TestesUnitariosBasico.ViewController.logar(nomeUsuario: Swift.St	100%
AppDelegate.swift	33,33%



Testes de Interface

Seção 2-1

Testes de Interface



Os **Testes de Interface** oferecem a capacidade de encontrar e interagir com elementos de UI para validar suas propriedades e estados.

Geramos os testes de interface através da gravação das interações com a aplicação e simultaneamente geramos o código responsável pelos testes. Quando interagimos com os elementos, a captação de gestos como **tap** ou **swipe** é realizada.

Além do já conhecido framework **XCTest**, os testes de UI também podem utilizar dos recursos de **Acessibility** para serem desenvolvidos.

Os recursos de **Acessibility** incluem um rico conjunto de dados semânticos sobre a UI que podemos usar para orientar nossos testes. A acessibilidade é integrada com UIKit e permite ajustes de comportamentos. O teste de UI usa esses dados para executar suas funções.

Enquanto os teste unitários permitem que se trabalhe dentro do escopo da aplicação e permite que se exerça funções e métodos com acesso total às variáveis e ao estado da sua aplicação, os testes de UI acessam a UI do aplicativo da mesma maneira que o usuário faria sem acesso aos métodos, funções e variáveis internas do aplicativo.

Isso permite que seus testes vejam o aplicativo do mesmo modo que um usuário faria, expondo os problemas de interface.

Os testes de UI são baseados na implementação de três classes:

- **XCUApplication:** É um proxy para o aplicativo que será testando. Permite a inicialização e encerramento do aplicativo para a execução de testes;
- **XCUElement:** É um proxy para os elementos de interface que serão testados. Todos os elementos possuem tipos e identificadores que são combinados para localizá-los na aplicação;
- **XCUElementQuery:** Quando precisamos localizar elementos de interface, são utilizadas as **queries**. As queries procuram o elemento de interface até encontrar o resultado exato.

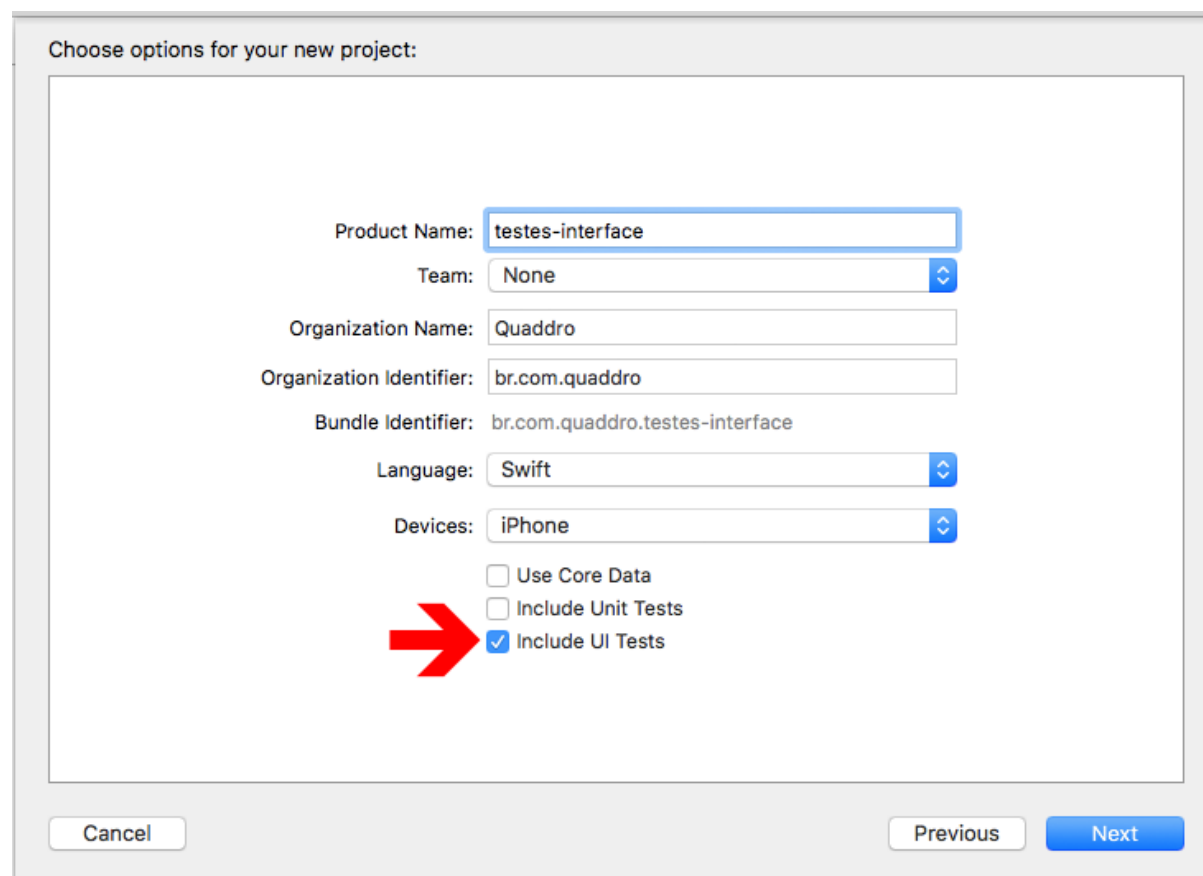
Seção 2-2

Novo projeto com Teste de Interface



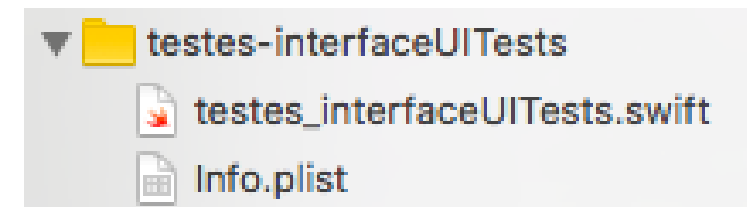
Podemos implementar a utilização de Testes de Interface logo na criação de um projeto, os passos a seguir descrevem o procedimento.

1. Crie um novo projeto **Single View Application**;
- 2, Digite o nome desejado para o projeto em **Product Name**;
3. Na tela de opções, selecione **Include UI Tests**.



Estrutura básica da classe de teste

Assim como ocorreu com testes unitários, ao selecionar a opção de inclusão de testes de UI, o Xcode cria um novo grupo para **nomeProjetoUITests**.



Dentro desse grupo, temos um arquivo de classe, com o mesmo nome do seu projeto, mais um prefixo **UITests.swift**. Ao abrir esse classe, podemos notar uma estrutura semelhante ao arquivo de classe para testes unitários, a diferença fica por conta dos códigos inseridos no método **setUp()**:

```
continueAfterFailure = false
//Propriedade que indica se o teste continua após uma falha

XCUIApplication().launch()
//Instância da classe XCUIApplication que faz o lançamento da aplicação
```


Seção 2-3

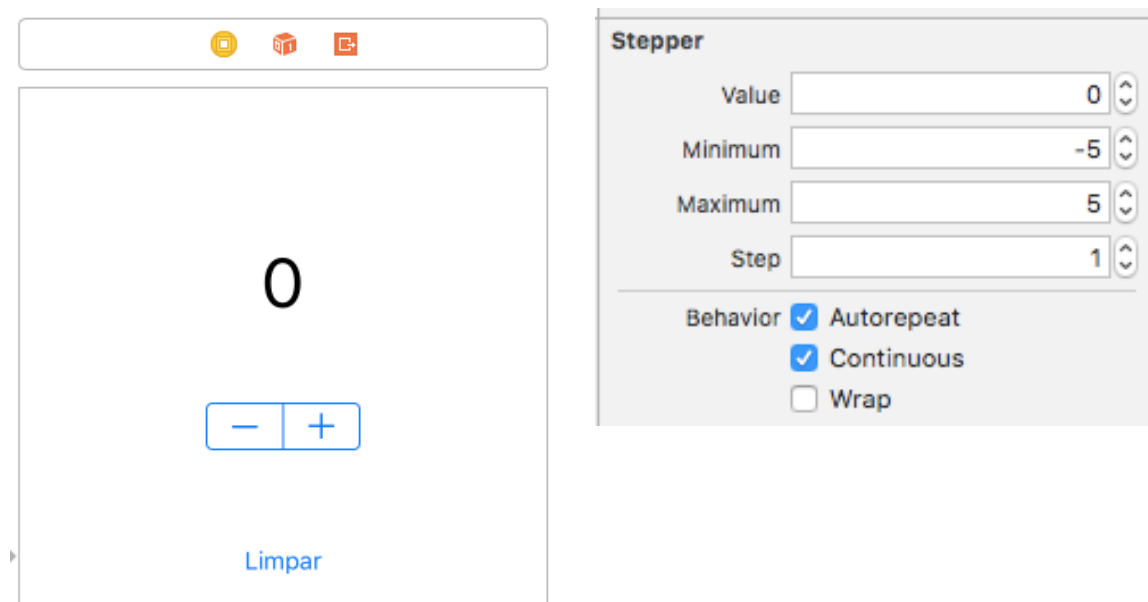
Gravando testes



O recurso de gravação de testes é bem simples e dinâmico. Vamos utilizar um projeto básico para entender como o processo funciona.

Preparando o projeto

Inicialmente, vamos montar um projeto que contemple a utilização de testes de UI. A interface do projeto deve conter um objeto **Label**, um **Stepper** com os atributos relacionados, e um **Button** conforme as imagens:



No arquivo **ViewController.swift** vamos trabalhar nos respectivos **Outlets**, **Actions**, **Métodos Próprios** e a **extension** conforme indicado:

```
class ViewController: UIViewController {

    //MARK:- Outlets
    @IBOutlet weak var labelValor: UILabel!
    @IBOutlet weak var stepperValor: UIStepper!

    override func viewDidLoad() {
        super.viewDidLoad()

        alterarValor(stepper: stepperValor)
    }

    //MARK:- Actions
    @IBAction func interagirStepper(_ sender: UIStepper) {
        alterarValor(stepper: sender)
    }

    @IBAction func limpar(_ sender: UIButton) {
        self.labelValor.text = "0"
    }

    //MARK:- Métodos Próprios
    func alterarValor(stepper : UIStepper) {

        self.labelValor.text = (stepper.value).valorInteiro
    }

    //MARK:- Extensão de String para
    extension Double{

        var valorInteiro : String{
            return String(format: "%.0f", self)
        }
    }
}
```

Preparando a classe de testes

Vamos até o **arquivoDeTest.swift**, e no final da classe de teste vamos inserir os seguintes métodos:

```
func testeStepperIncremento() {  
}  
  
func testeStepperDecremento() {  
}  
  
func testeButtonLimpar() {  
}
```

Em seguida, faça um **Build** para que esses métodos sejam incluídos ao projeto. Ao final do processo, sua classe de testes deverá apresentar as seguintes informações:

```
class ContadorUITests: XCTestCase {  
  
    override func setUp() {  
        super.setUp()  
  
        continueAfterFailure = false  
        XCUIApplication().launch()  
    }  
  
    override func tearDown() {  
        super.tearDown()  
    }  
  
    func testeStepperIncremento() {  
  
    }  
  
    func testeStepperDecremento() {  
  
    }  
  
}
```

Nosso intuito com esses testes é verificar os momentos em que o Stepper incrementa ou decrementa os valores na Label, e quando o botão Limpar zera a Label.

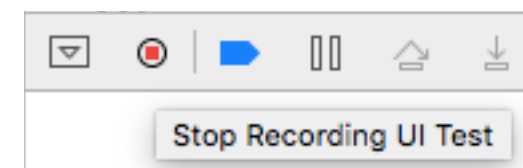
Gravando os testes

Inicialmente vamos gravar o teste de incremento. No **arquivoDeTest.swift**, posicione o cursor de digitação dentro do método **testeStepperIncremento**. Observe que, junto a barra acima do console, temos um conjunto de botões, nesse conjunto, vamos acionar o botão **Record UI Test**:



Perceba que a aplicação será executada. Todas as interações que fizermos com a aplicação serão refletidas no nosso código, dentro da função que destacamos antes de iniciar a gravação. Ou seja, nossas interações estão sendo gravadas no nosso código de teste.

Vamos utilizar o botão **+** do stepper para incrementar os valores, até **5**. Em seguida podemos finalizar a gravação com o mesmo botão que iniciamos, que agora apresenta a opção **Stop Recording UI Test**:



Ao término da gravação, nossa função apresentará os seguintes códigos:

```
func testeStepperIncremento() {  
    let incrementButton = XCUIApplication().steppers.  
        buttons["Increment"]  
  
    incrementButton.tap()  
    incrementButton.tap()  
    incrementButton.tap()  
    incrementButton.tap()  
    incrementButton.tap()  
}
```

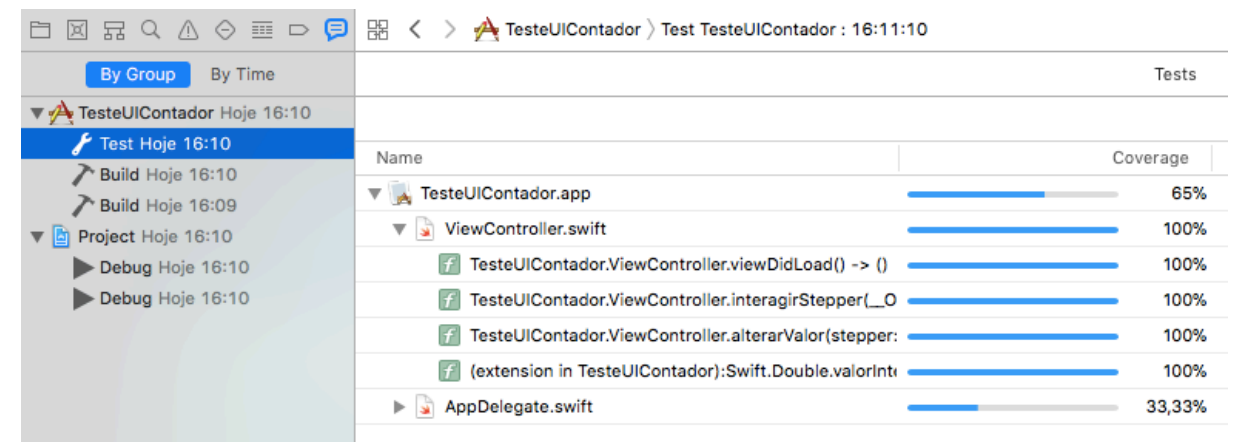
Por enquanto não vamos nos preocupar com as outras funções pois criaremos cada um dos testes com recursos que veremos adiante.

Executando os testes

Com o teste já gravado, podemos executá-lo de forma semelhante aos testes unitários. Podemos executar um teste de cada vez, ou todos os testes que compõem a classe de testes.

Quando rodamos os testes, o simulador abre, e executa as tarefas determinadas pela gravação, como se fosse um robô.

Ao término da execução do teste proposto, podemos aferir a taxa de cobertura de testes (Code coverage):



Seção 2-4

Utilizando o Accessibility

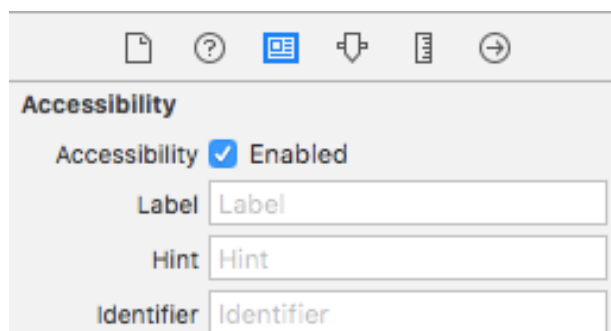


Podemos utilizar objetos **Accessibility** como parte do processo de teste. Eles auxiliam na identificação dos elementos visuais no código, e também no acesso desses elementos na classe de teste.

Inicialmente vamos trabalhar com o objeto **Button**, que tem a função de limpar o contador e voltar o texto da label para 0:



Podemos indicar um objeto accessibility para esse button. Selecione o objeto na view controller, e em **Identity Inspector** procure pela seção **Accessibility**



Verifique se a opção **Enabled** encontra-se ligada, e no campo **Label** entre como texto **Botão Limpar**.

Outro elemento da interface que também necessitará de um objeto accessibility, é a label que exibe a contagem conforme interagimos com o stepper. Selecione essa label, e em **Accessibility**, verifique se a opção **Enabled** encontra-se ligada. No campo **Label** entre como texto **Label Valor**.

A Partir de agora, nossos objetos de interface contam um identificador, que será utilizado no momento da gravação do código para identificar os objetos ou captar os valores conforme a necessidade.

Utilizando o Accessibility para identificar objetos

Vamos gravar um teste dentro da função `testeButtonLimpar`, onde incrementaremos o valor duas vezes, e depois efetuaremos a limpeza. Ao término do processo teremos o seguinte resultado:

```
func testeButtonLimpar() {  
    let app = XCUIApplication()  
  
    let incrementButton = app.steps.buttons["Increment"]  
    incrementButton.tap()  
    incrementButton.tap()  
  
    app.buttons["Botão Limpar"].tap()  
}
```

Observe que o teste que envolve o button fez uso do identificador definido no objeto de acessibilidade: `app.buttons["Botão Limpar"].tap()`

Captando valores de objetos com accessibility

Por ultimo vamos trabalhar com a função `testeStepperDecremento`, que declaramos anteriormente e ainda não implementamos os devidos testes.

Vamos trabalhar de uma forma diferente com essa função. Ao invés de gravarmos um teste, vamos implementar os códigos manualmente. Dessa forma, podemos verificar que os testes podem ser gravados, ou digitados. Tudo depende da necessidade de cada teste:

```
func testeStepperDecremento() {  
    let decrementButton = XCUIApplication().steppers.  
        buttons["Decrement"]  
  
    let valorLabel = XCUIApplication().staticTexts  
        ["Label Valor"]  
  
    for valorAtual in 0...4{  
        decrementButton.tap()  
        XCTAssertEqual(valorLabel.value as! String,  
            "\(-valorAtual - 1)")  
    }  
}
```

Observe que, quando optamos por digitar um teste, podemos dispor de recursos como **loops** ou **condicionais**. No nosso caso, utilizamos a estrutura **for-in**.

Nessa ultima função, temos dois pontos a serem destacados:

1- `let valorLabel = XCUIApplication().staticTexts["Label Valor"]`

Através desse objeto, vamos fazer a captação de uma label de acordo com o seu elemento de accessibility.

2- `XCTAssertEqual(valorLabel.value as! String, "\(-valorAtual - 1)")`

Assim como fizemos com os testes unitários, dispomos das classes de **XCTAssert** para testar comparações.

Para que nosso teste de decremento seja efetuado com sucesso, devemos incluir uma linha de código na classe `ViewController`.

Vamos até o arquivo **ViewController.swift**, e no método próprio `alterarValor`, adicione a nova linha de código conforme o exemplo:

```
func alterarValor stepper : UIStepper) {  
    self.labelValor.text = (stepper.value).valorInteiro  
  
    //Indicando o elemento de accessibility  
    self.labelValor.accessibilityValue = labelValor.text  
}
```

Basta executar o teste para ver o resultado.



Integrando o Xcode com o Git



Um projeto pode ser desenvolvido de forma independente ou com um time estruturado corporativamente. Ambas as abordagens devem ter a mesma preocupação com relação às versões do código fonte dos projetos que estão sendo desenvolvidos e implantados em produção.

Vamos considerar alguns cenários do mundo real:

- Um diretório onde estava o código fonte de uma aplicação foi acidentalmente apagado sem que tivesse um backup;
- Uma nova versão de uma aplicação foi implantada está com um problema sério, e o time opta por retornar à versão anterior. Mas, onde encontrar essa versão?
- Surgiu a necessidade de implantar novas funcionalidades no app, mas não queremos correr o risco de gerar impactos naquela versão que já está funcionando.

Para não gerar desespero com essas situações, que podem ocorrer com alguma frequência, é necessário fazer uso de um sistema de controle de versionamento de software, que seja confiável, fácil de se utilizar e que possua integração com o interface de desenvolvimento iOS, no nosso caso o Xcode.

Uma das soluções mais utilizadas no mercado é o **Git**.

O que é o Git?

Git é um sistema de controle de versionamento de arquivos. Através dele podemos desenvolver projetos no qual diversos profissionais podem contribuir simultaneamente, editando e criando novos arquivos e permitindo que os mesmos possam existir sem o risco de suas alterações serem sobrescritas. Adicionalmente, O Git guarda um histórico de alterações em seu código.

Uma das características mais interessantes do Git é que um desenvolvedor, ao iniciar a codificação de uma nova funcionalidade do app, pode criar uma área separada, chamada **branch**, e trabalhar nela de forma independente, até julgar que a tarefa esteja concluída.

Neste momento ele pode integrar em um processo chamado **merge**, o seu trabalho à uma branch compartilhada entre os demais desenvolvedores do time.

Um branch, que podemos traduzir livremente para ramo, é uma forma de manter suas alterações num mesmo repositório. Quando confirmamos essas alterações com um processo chamado de **commit**, sua árvore ficará atualizada com todo o trabalho executado.

A quantidade de comandos do Git é extensa. Durante esse capítulo mostraremos que é possível utilizar os recursos do Git exclusivamente através do Xcode.

Git x GitHub

É comum confundir esses dois serviços, mas podemos, em termos gerais, distingui-los da seguinte forma:

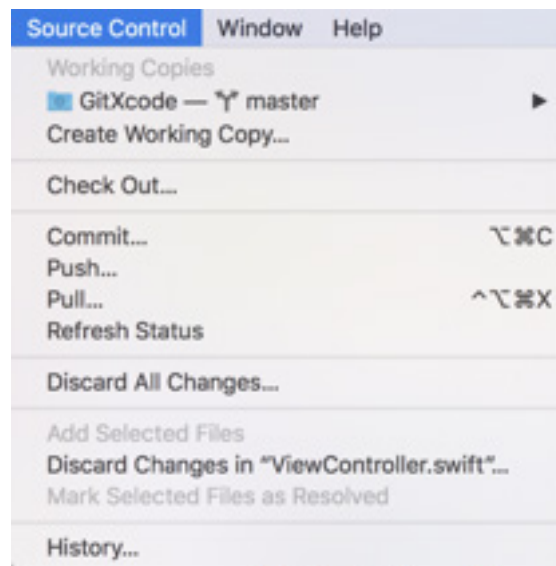
O **Git** é um sistema de controle de versões de projetos. Uma ferramenta para gerenciar o histórico de alterações do código. Já o **GitHub** é um serviço de armazenamento de repositórios Git, na nuvem.

A integração entre o Xcode e o Git

A integração entre o Git e o Xcode existe desde a versão 5 da plataforma de desenvolvimento da Apple. Com o Xcode podemos efetuar todo o controle de versionamento diretamente na interface de desenvolvimento, sem uso de comandos Terminal.

O menu **Source Control** do Xcode fornece acesso a todas as funcionalidades do Git. Exploraremos passo a passo os recursos no desenvolvimento de um exemplo prático.

Caso queira aprofundar os conceitos e uso do Git recomendamos a leitura do material oficial em <https://git-scm.com>.



Seção 3-2

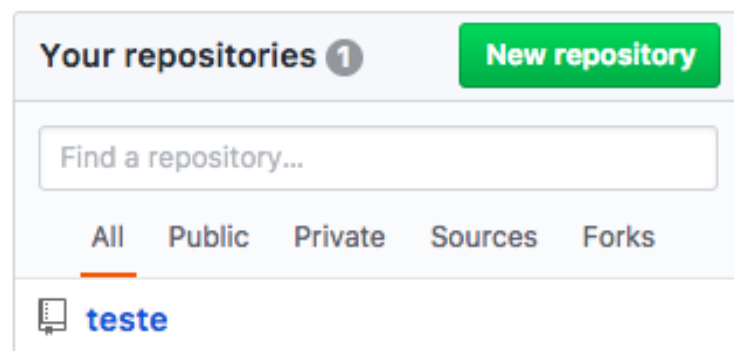
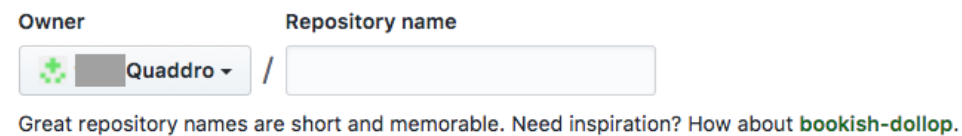
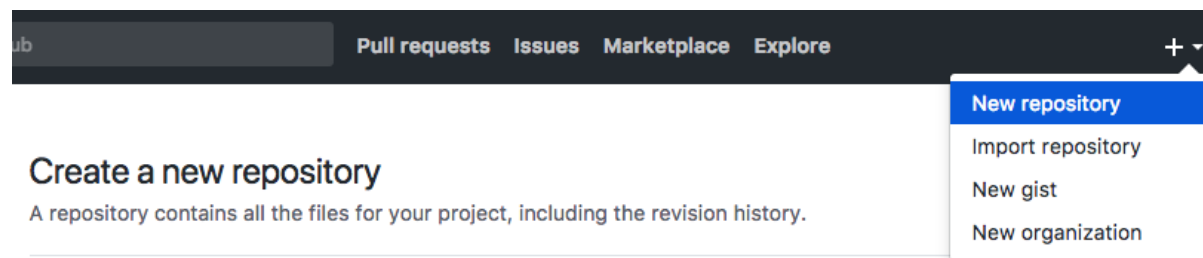
Criação de repositórios



Para se ter acesso aos benefícios do Git, é necessário ter uma conta no Github, e nessa conta ter um repositório. Vamos seguir os passos abaixo:

- Acesse o GitHub em <https://github.com/>, crie uma conta, caso não tenha, e acesse a página principal.

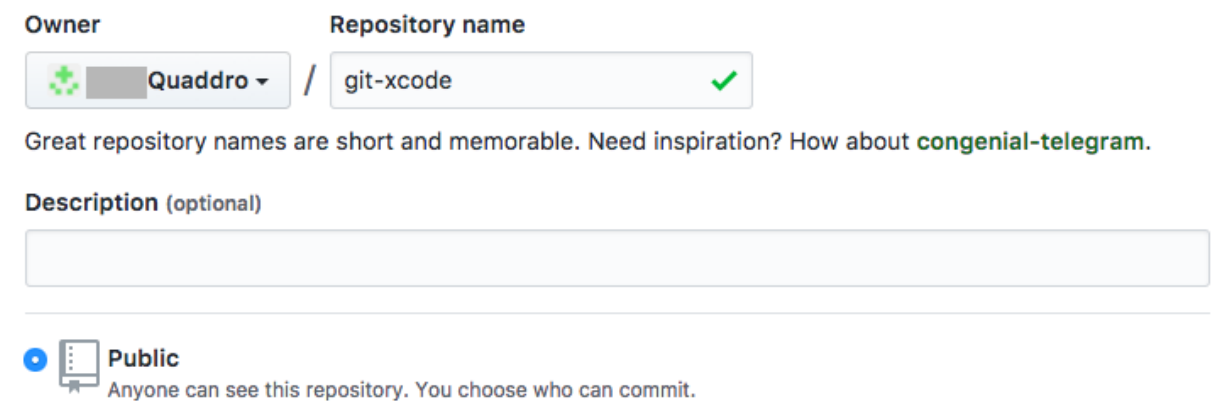
- Localize a seção onde se possa criar um novo repositório, podemos encontrar essa opção em lugares diferentes:



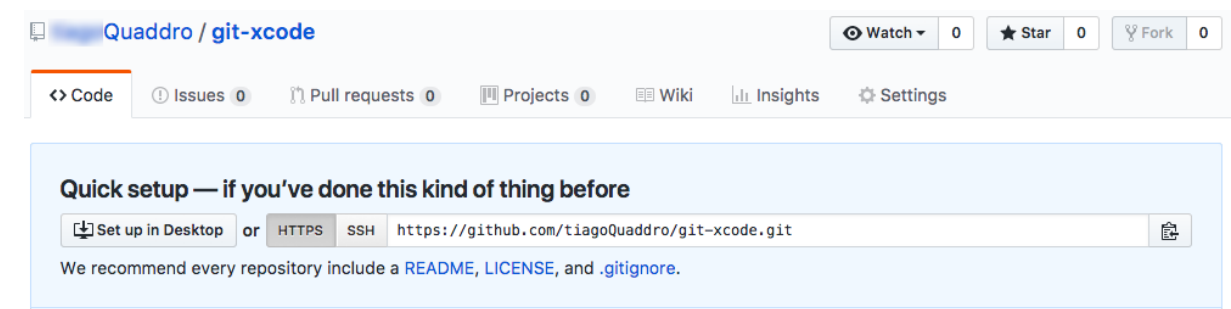
- Crie um novo repositório, com o nome **git-xcode**. Preencha os demais campos como mostrado na figura abaixo:

Create a new repository

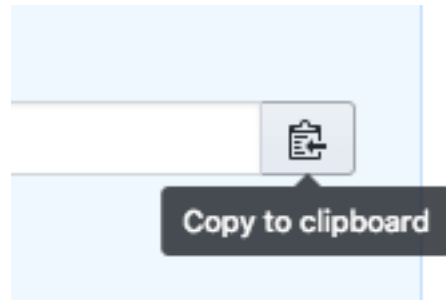
A repository contains all the files for your project, including the revision history.



Se tudo correu como esperado na criação do repositório, será exibida a tela a seguir:



- Antes de prosseguir, vamos guardar a url do projeto, que foi criada pelo github. Clique sobre o botão **Clone or download** e copie a **url** gerada:



Como criamos um repositório público, será através deste mesmo caminho que outras pessoas poderão baixar o projeto para uso próprio.

Seção 3-3

Criando o projeto no Xcode



No Xcode vamos criar um projeto que será preparado para integração com o Git, vamos aos passos:

- Crie um novo projeto Single View Application, com o nome o nome **git-xcode**.

Product Name:

Team:

Organization Name:

Organization Identifier:

Bundle Identifier:

Language:

☐ Use Core Data

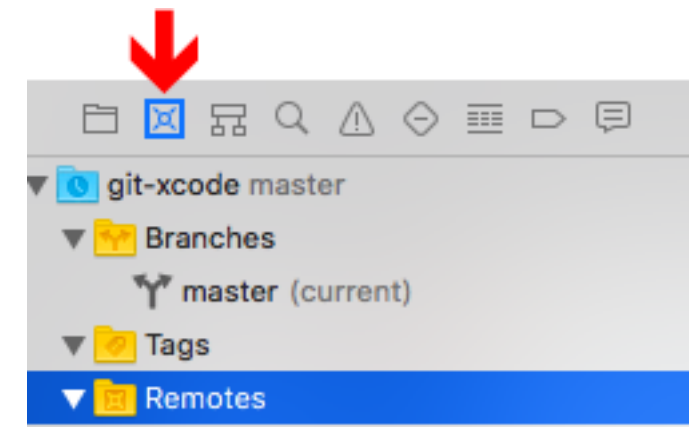
☐ Include Unit Tests

☐ Include UI Tests

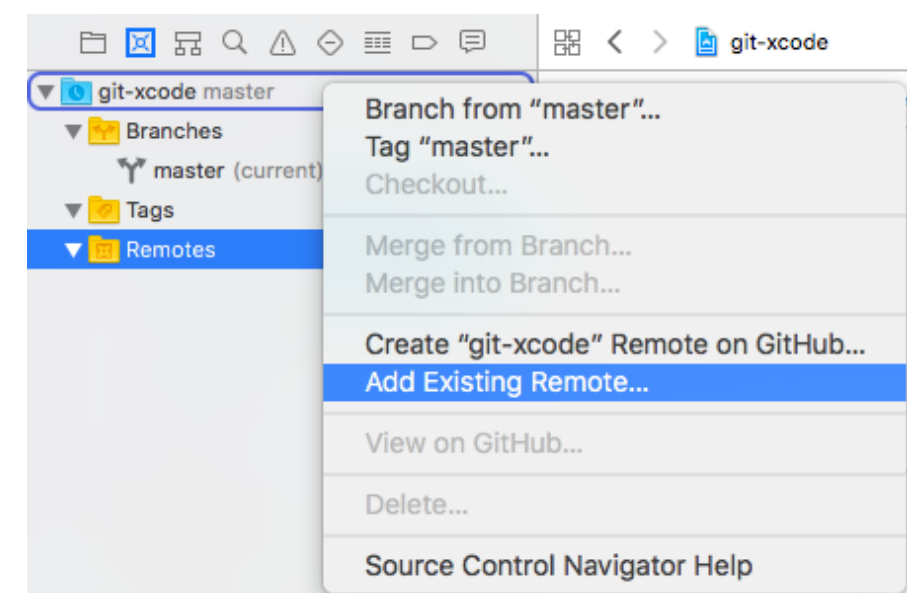
- Na próxima tela, após clicar em **Next**, atenção para ligar o opção **Create Git repository on My Mac**:

Source Control: ☒ **Create Git repository on my Mac**
Xcode will place your project under source control

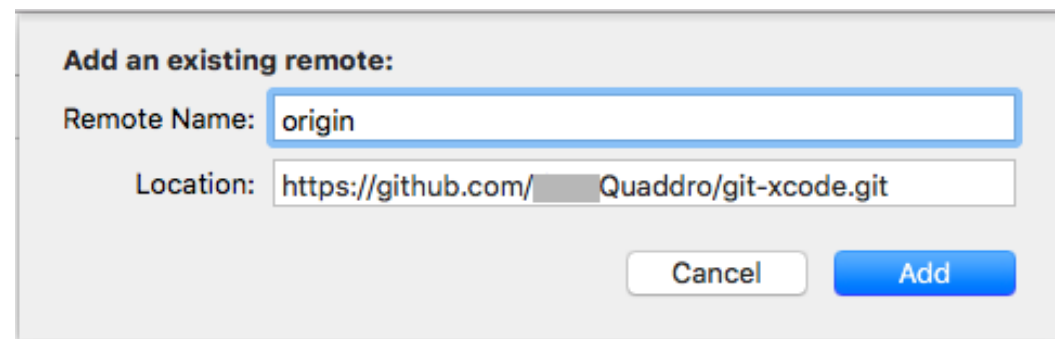
- Na barra de navegadores, procure por **Source Control navigator**;



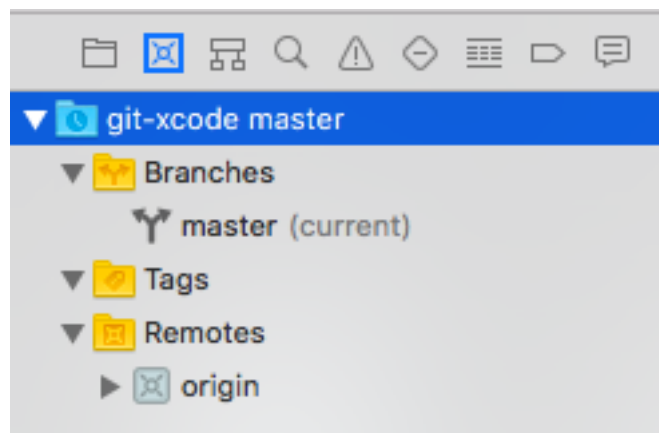
- Agora vamos associar o projeto ao repositório remoto que criamos no Github. Clique com o botão direito sobre o ícone **git-xcode master**, e no menu escolha a opção **Add Existing Remote**;



- Na tela de inclusão, mantenha o nome **origin** no campo **Remote Name**, e copie a **url** gerada pelo Github no campo **Location**, seguido de um clique no botão **Add**.



Ao final do processo, teremos um item no grupo **Remotes** com o nome **origin**, conforme criamos:



Seção 3-4

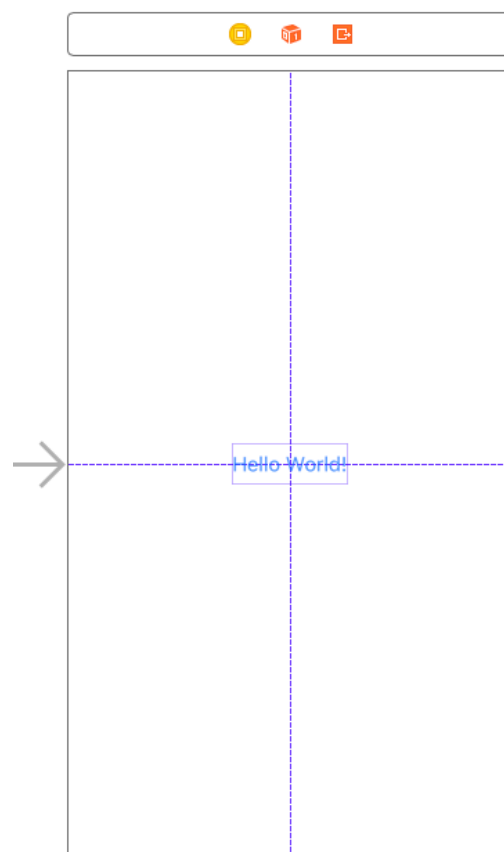
Trabalhando com o Git



Com a criação e configuração do projeto, podemos começar a trabalhar com seus elementos e códigos.

Faremos a primeira implementação para efeito ilustrativo, um botão com um **Hello World!**:

- Na **ViewController** principal do projeto, adicione um objeto **button**, e o posicione no centro da tela, troque o título desse botão para **Hello World!**;



Crie uma conexão **@IBAction** na classe **ViewController.swift**, com o nome **showHello()**;

- Dentro da **func**, crie um **alerta** para exibir uma mensagem, conforme o código abaixo:

```
// MARK - Actions
@IBAction func showHello() {

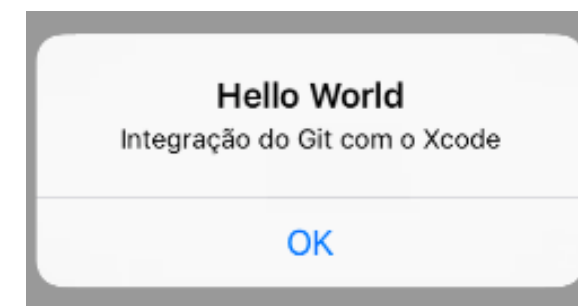
    let alert = UIAlertController(title: "Hello World",
                                message: "Integração do Git com o Xcode",
                                preferredStyle: .alert)

    let action = UIAlertAction(title: "OK", style: .default,
                                handler: nil)

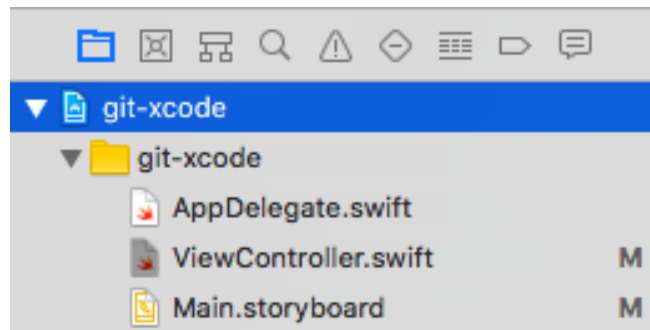
    alert.addAction(action)

    present(alert, animated: true)
}
```

Execute o projeto para certificar-se que tudo está funcionando bem.



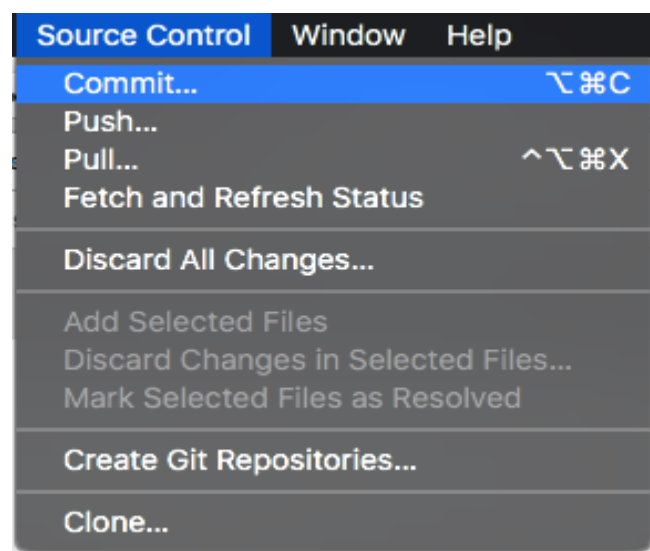
Observe o navegador de projetos, as indicações dos componentes que sofreram alterações (indicados pela letra “M”).



Como fizemos a nossa primeira implementação, podemos pensar em salvar as alterações no nosso repositório remoto.

Para isso, iremos, ao mesmo tempo, gravar as alterações (operação **Commit**) e atualizar o repositório remoto master (operação de **Push**).

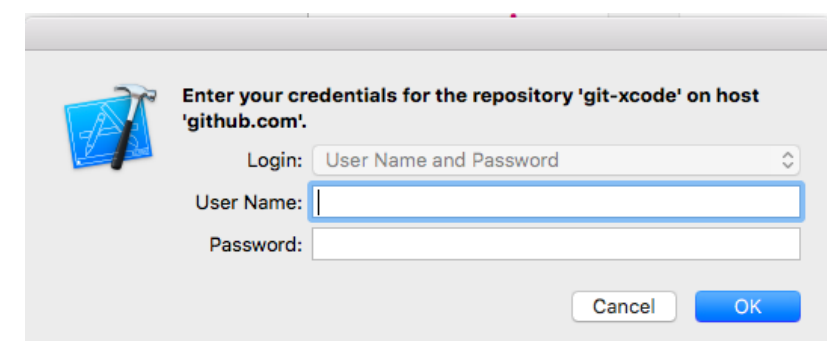
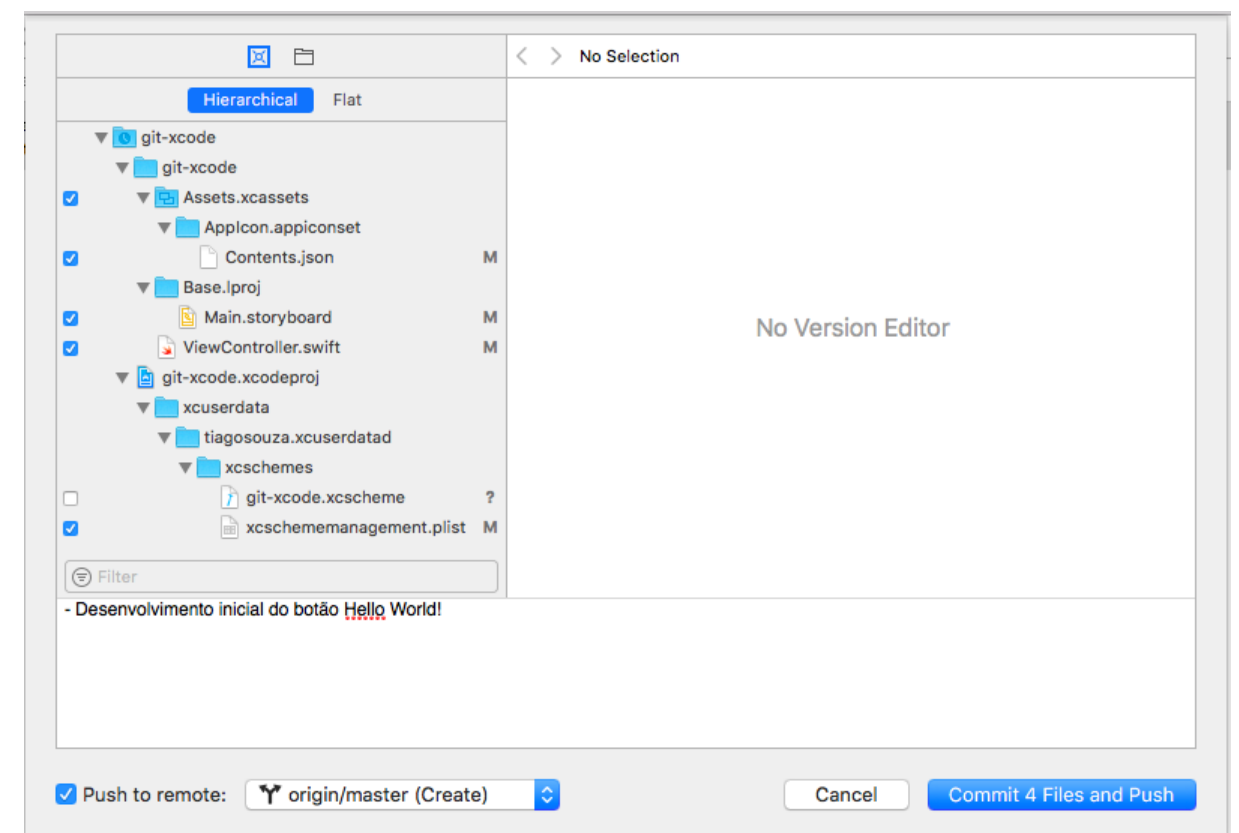
- No menu **Source Control**, execute a opção **Commit**. O Xcode irá exibir a tela de alterações.



- No campo **Enter commit message here**, podemos descrever as operações efetuadas;

- Como queremos salvar as alterações remotamente, devemos ligar a opção **Push to Remote**;

- Após executar a operação através do botão **Commit N Files and Push**, serão solicitadas as credenciais no Github e o processo será concluído:



Podemos, no Github, confirmar o Commit. Basta efetuar um refresh na tela do repositório de nosso projeto.

Observe que o texto que colocamos como Commit no Xcode aparece refletido no GitHub. Quando o repositório foi criado, o próprio Github gerou um Initial Commit, que também é exibido:

The screenshot shows the GitHub interface for a repository named 'git-xcode' under the user 'Quaddro'. At the top, there are buttons for 'Watch' (0), 'Star' (0), and 'Fork' (0). Below this is a navigation bar with links for 'Code', 'Issues' (0), 'Pull requests' (0), 'Projects' (0), 'Wiki', 'Insights', and 'Settings'. The main content area starts with a message 'No description, website, or topics provided.' and an 'Edit' button. Below this is a summary bar showing '5 commits', '1 branch', '0 releases', and '1 contributor'. A secondary bar contains a 'Branch: master' dropdown, a 'New pull request' button, and buttons for 'Create new file', 'Upload files', 'Find file', and a green 'Clone or download' button. The commit history is listed below, showing a commit by 'ThomazQuaddro' with the message 'Desenvolvimento inicial do botão Hello World!' (2 minutes ago) and an 'Initial Commit' (13 minutes ago). At the bottom, there is a light blue banner encouraging the user to 'Add a README'.

Quaddro / git-xcode

Watch 0 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

No description, website, or topics provided. Edit

Add topics

5 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

ThomazQuaddro committed with ThomazQuaddro - Desenvolvimento inicial do botão Hello World! Latest commit b30eaae 2 minutes ago

git-xcode.xcodeproj	Initial Commit	13 minutes ago
git-xcode	- Desenvolvimento inicial do botão Hello World!	2 minutes ago

Help people interested in this repository understand your project by adding a README. Add a README

Seção 3-5

Trabalhando de forma colaborativa



Nos passos anteriores, trabalhamos diretamente na árvore principal do projeto, a **master**.

Sempre que efetuarmos a operação **Push**, o repositório remoto do projeto será atualizado com as nossas últimas alterações.

Essa forma de trabalhar pode atender um desenvolvedor que trabalha sozinho, mas não é a forma ideal de se trabalhar em equipe, ou também quando já temos uma versão do aplicativo na loja, e não queremos correr o risco de gerar algum impacto na versão de produção.

A forma mais comum de trabalho colaborativo é gerenciar as ramificações da seguinte forma:

- A master é deixada como espelho da versão em produção, e é atualizada, através de uma operação **merge**, apenas quando uma nova versão do aplicativo for subir para a loja;

- É criada uma nova árvore, normalmente chamada de **develop**, que é atualizada sempre com alterações já devidamente estáveis e testadas, e que estão aguardando o merge na master;

- Quando os desenvolvedores estão trabalhando individualmente em implementações de novas funcionalidades, são criadas ramificações locais, onde é efetuada toda a implementação, antes de qualquer atualização remota.

Nos próximos passos vamos reproduzir essa última forma de trabalho, simulando o desenvolvimento de uma nova funcionalidade em nosso projeto **git-xcode**.

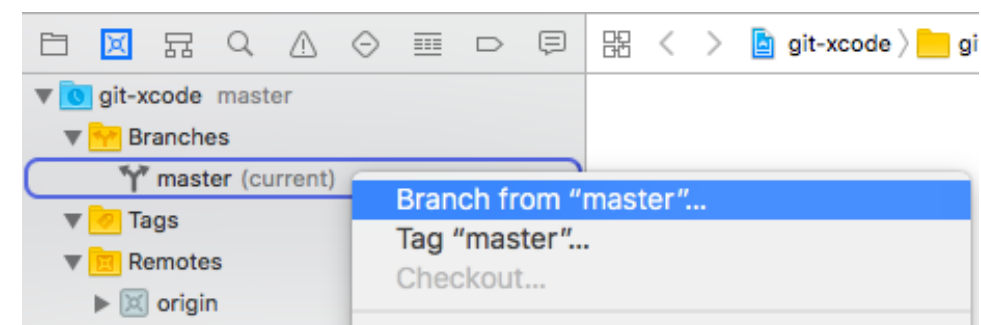
Implementando uma nova funcionalidade

Faremos uma alteração no nosso app, que consiste em criar um campo texto, onde o usuário possa customizar a mensagem produzida pelo botão Hello World!. Ao invés de trazer um conteúdo fixo, a mensagem de alerta deve ser intercalada com o texto digitado pelo usuário.

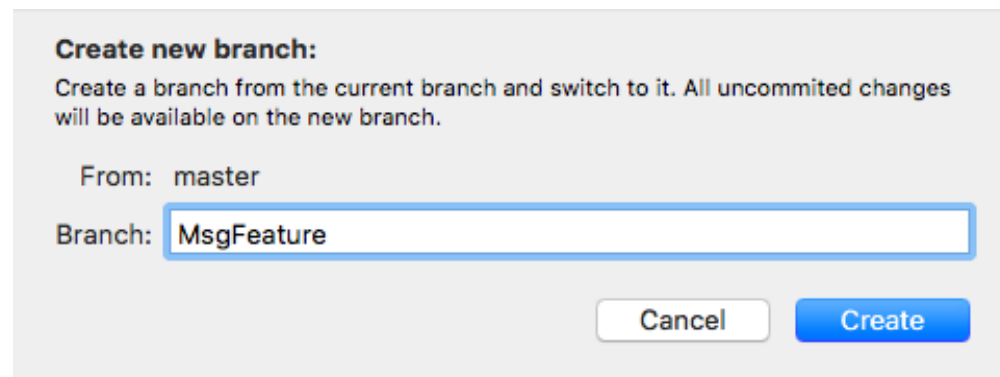
Na interface, será criado um novo elemento visual **TextField**, e a lógica do botão será alterada para utilizar o conteúdo do texto, e não mais uma literal fixa.

Antes de começar os trabalhos, vamos tomar o cuidado de criar uma nova ramificação, chamada de **MsgFeature**. Para isso, serão executadas as seguintes ações:

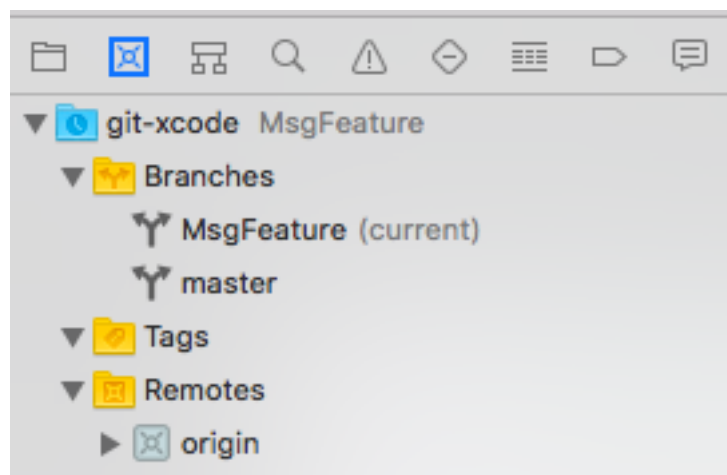
- No navegador **Source Control**, clique com o botão direito em **master**, e no menu, escolha a opção **Branch from “master”**:



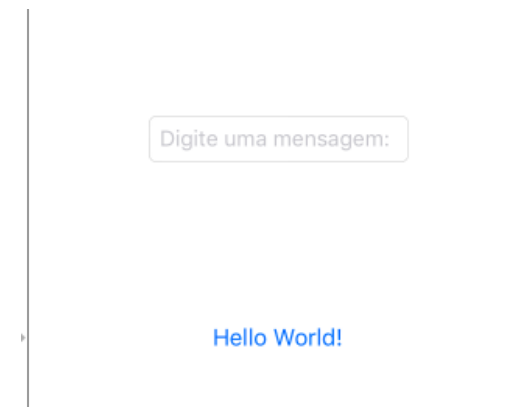
- Na caixa exibida, defina o nome da nova árvore como MsgFeature;



Observe que o Xcode avisa que a nova árvore será criada, e passará a ser padrão para os futuros commits e pushes. Podemos conferir a nova árvore corrente no navegador Source Control:



- Vamos agora implementar as alterações no projeto com um novo campo na ViewController principal, um objeto **TextField**, conforme a imagem:



- Crie uma conexão **@IBOutlet** desse textField na classe **ViewController.swift**, com o nome **textFieldMsg**;

```
//MARK: Outlets
@IBOutlet weak var textFieldMsg: UITextField!
```

- Dentro da **func**, atualize a propriedade **message** do alerta para exibir uma mensagem de acordo com o texto digitado no textField:

```
// MARK - Actions
@IBAction func showHello() {

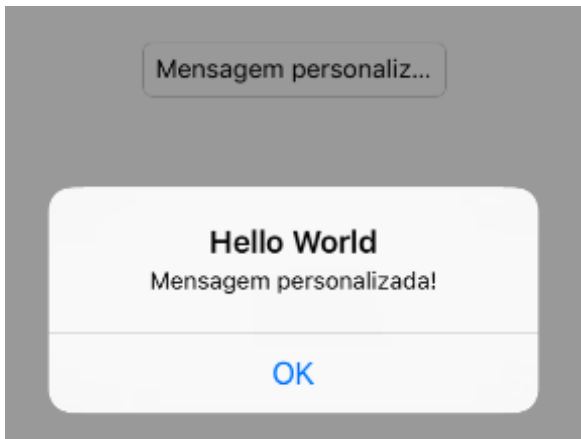
    let alert = UIAlertController(title: "Hello World",
    message: textFieldMsg.text,
    preferredStyle: .alert)

    let action = UIAlertAction(title: "OK", style: .default,
    handler: nil)

    alert.addAction(action)

    present(alert, animated: true)
}
```

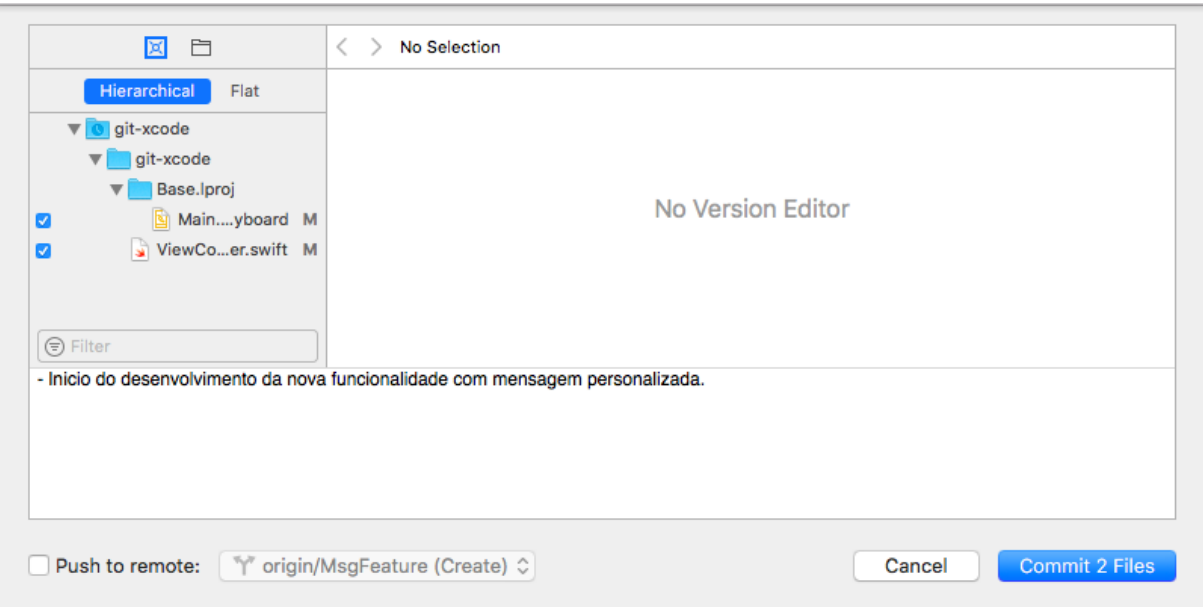
Execute o projeto para certificar-se que tudo está funcionando bem.



Salvando as alterações

Após trabalhar durante algum tempo, podemos optar por salvar as alterações de forma local, ainda sem atualizar nosso repositório remoto:

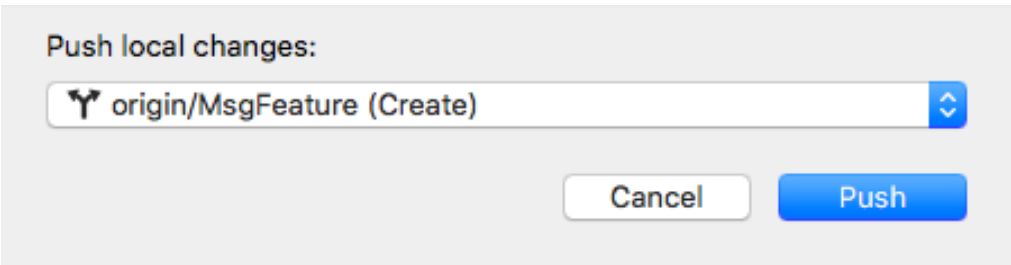
Através do menu **Source Control** podemos fazer um **Commit** das alterações:



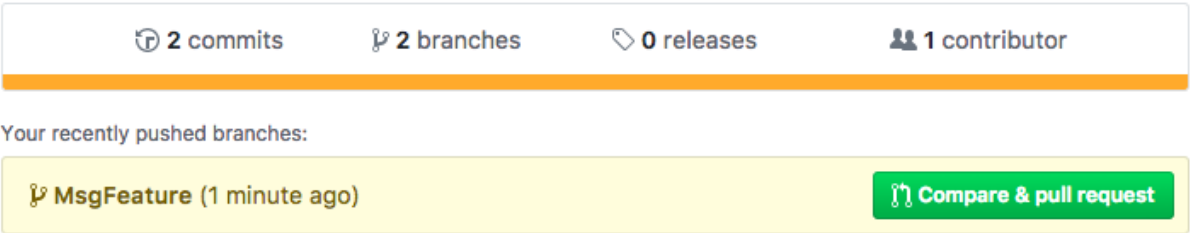
Nesse caso, como não optamos por efetuar o Push para a ramificação remota, essas alterações ainda não foram salvas remotamente.

Se consultarmos o Github, veremos que alterações não foram refletidas na ferramenta.

Vamos supor agora que, ao final de dia, decidimos salvar as alterações remotamente, prevenindo qualquer problema que possa ter com tudo o que foi implementado. Para tal, basta executar a opção **Push** do menu **Source Control**:



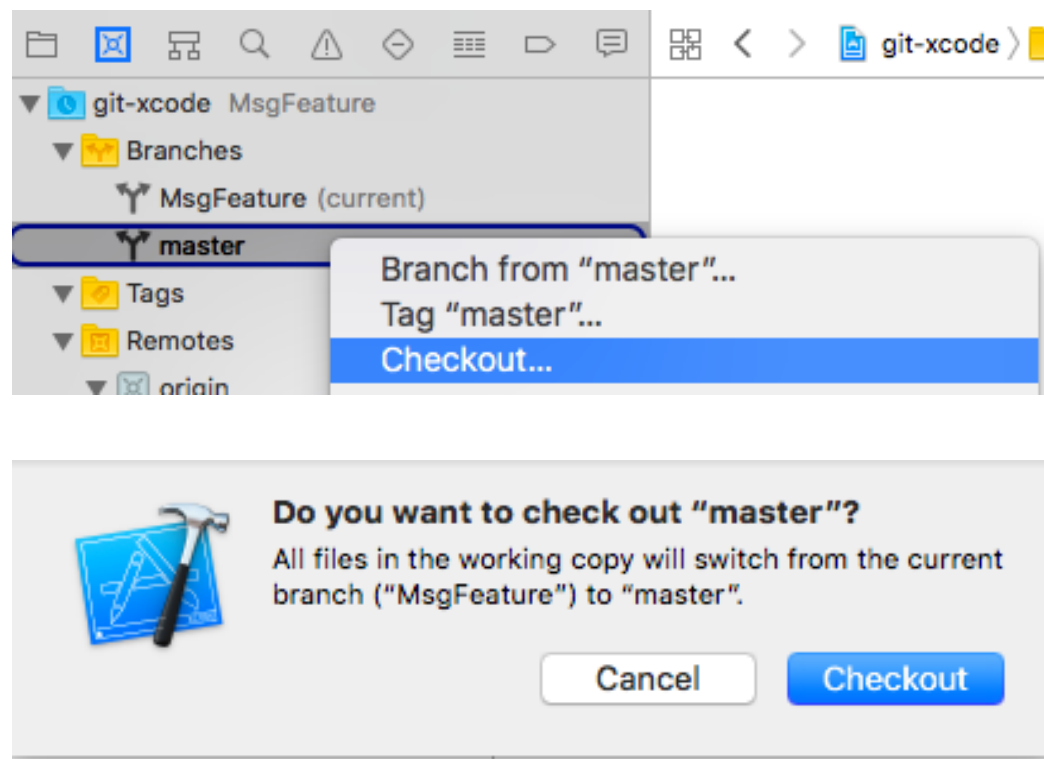
Ao consultarmos o Github, veremos que estamos com 2 branches agora:



Atualizando a árvore master

Chegou a hora de atualizar a árvore master, pois ela ainda não contém esta última implementação das mensagens personalizadas.

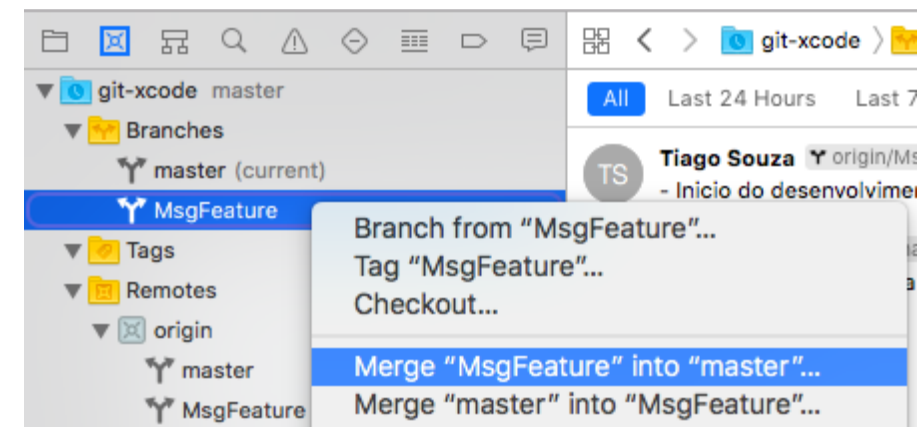
Um forma de visualizar essa situação é chavear o projeto para a árvore master local. Para tal, basta clicar com o botão direito sobre a árvore que deseja tornar corrente e escolher a opção **Checkout**:



Podemos observar que temos duas versões do nosso projeto, uma com o textField para mensagens personalizadas na árvore MsgFeature, e outra sem o textField na árvore master.

Vamos efetuar o **Merge** das árvores, para atualizar local e remotamente as últimas implementações.

- Efetue um **Checkout** na árvore **master** para deixá-la corrente;
- No navegador Source Control, clique com o botão direito sobre a árvore MsgFeature e no menu escolha a opção **Merge "MsgFeature" into "master"**;



A árvore **master** ficará atualizada mas somente de forma local, para subir a atualização para o GitHub será necessário efetuar um **Push**. O processo também pode ser conferido no seu repositório.



Publicando na App Store

Seção 4-1

Entendendo o processo



O processo de desenvolvimento de um aplicativo para a plataforma iOS passa por três grandes fases, que são elas:

Programa de Desenvolvedor (Enroll)

O cadastramento no programa de desenvolvedores da Apple, constituindo uma conta paga de desenvolvedor, que deve ser renovada anualmente. Os procedimentos de cadastramento e manutenção da conta de desenvolvedor são efetuados através do portal Apple Member Center: <https://developer.apple.com/account/#/welcome>

Desenvolvimento e Configuração do App

É a fase de criação, preparação e upload do aplicativo. O Xcode é a ferramenta utilizada nesta fase, sendo a versão de produção pode ser baixada diretamente da App Store.

Nesta fase, já temos uma atividade que envolve o portal de gerenciamento e distribuição dos aplicativos, que é o iTunes Connect: <https://itunesconnect.apple.com>

Os desenvolvedores cadastrados no programa de desenvolvimento ainda tem acesso às versões beta tanto do Xcode quanto dos sistemas operacionais, que podem ser baixadas diretamente do Apple Member Center.

Distribuição do App

É o último passo para colocar o app no mercado. Não é um processo complicado, porém exige questões cadastrais e burocráticas a serem resolvidas.

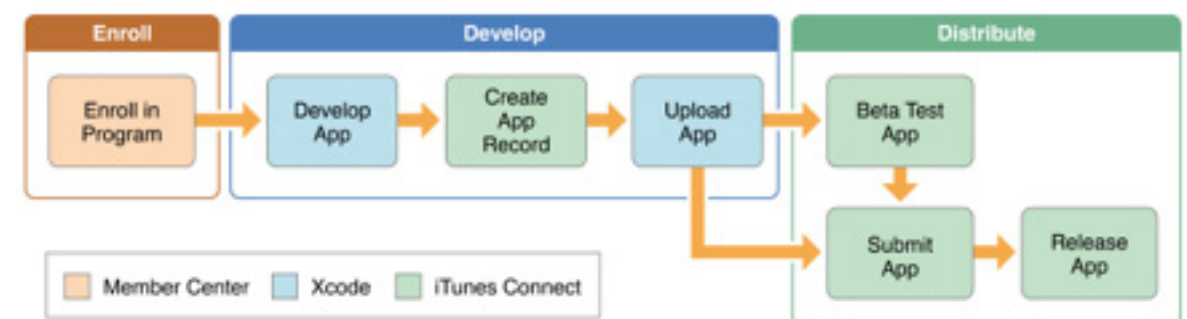
Adicionalmente, aspectos de qualidade e padronização devem ser levados em consideração, para que o app não seja rejeitado pela Apple no review do aplicativo. É extremamente recomendável o estudo dos guidelines da Apple, para que não ocorram surpresas no processo de review.

Podemos acessar os guidelines através do link <https://developer.apple.com/app-store/guidelines/>

O guideline mais importante para um desenvolvedor iOS, é o iOS Human Interface Guideline: <https://developer.apple.com/ios/human-interface-guidelines/overview/design-principles/>

Na WWDC - World Wide Developers Conference, de 2016, a Apple disponibilizou um documento muito interessante, em forma de HQ, com todas as fases do processo de review de aplicativos: <https://devimages.apple.com.edgekey.net/app-store/review/guidelines/App-Review-Guidelines-The-Comic-Book.pdf>

Veja, na figura a seguir, uma visualização gráfica de todas as fases:



No decorrer desse capítulo, entraremos nos detalhes de cada fase, focando no processo de distribuição do aplicativo. Ao final, teremos um roteiro passo a passo, para utilizar quando estiver pronto para publicar seu primeiro app.

Projeto

O projeto utilizado como exemplo nesse capítulo não é um projeto real ou passível de ser publicado na App Store.

Existem algumas fases do processo de publicação que requerem o uso de screenshots e ícones. Nestes casos, para que fosse possível simular o procedimento real no portal iTunes Connect, foram utilizadas imagens de um aplicativo utilizado na Quaddro para treinamentos internos, de cálculo de IMC - Índice de Massa Corpórea.

Seção 4-2

Programa de Desenvolvedor



Inicialmente vamos detalhar os passos para tornar-se um desenvolvedor da plataforma iOS, habilitado para publicar seus aplicativos na App Store.

Podemos resumir essa fase em quatro etapas:

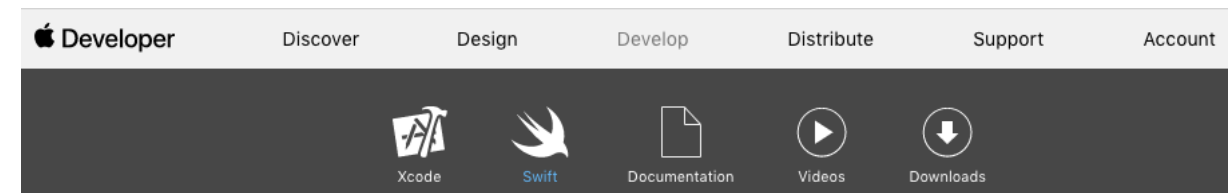
1. Registrar-se como desenvolvedor Apple;
2. Criar uma conta paga de desenvolvedor;
3. Configurar seu Mac para desenvolvimento, que inclui:
 - Criar um certificado;-
 - Registrar seus dispositivos físicos para testes;
 - Criar um App ID;
 - Criar Provisioning Profiles.

Vamos ver agora os detalhes de cada etapa.

É importante observar que várias das informações geradas são únicas e exclusivas de sua conta, seus devices e aplicativos, portanto são confidenciais e não devem ser compartilhadas. Nesta apostila, essas informações estão ofuscadas sempre que aparecerem nas figuras.

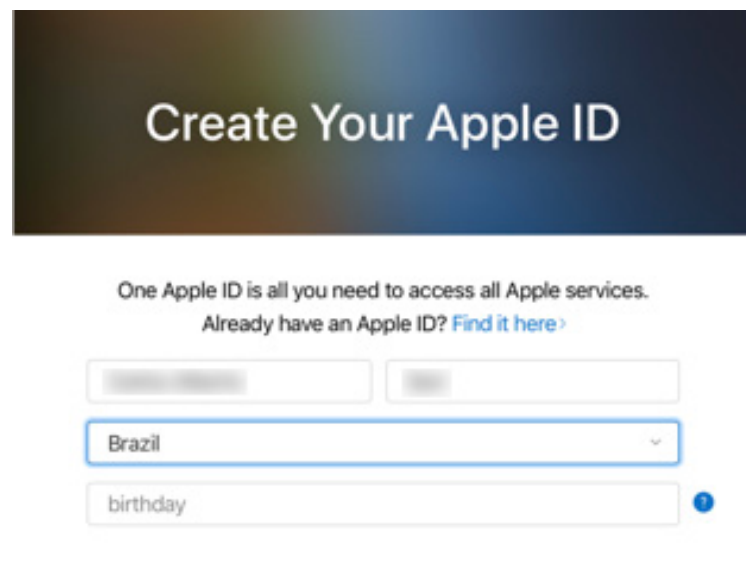
Registrar-se como desenvolvedor Apple

O primeiro passo é registrar-se como desenvolvedor Apple, através do acesso em <https://developer.apple.com/develop/>, selecionar **account**, no menu superior da página:



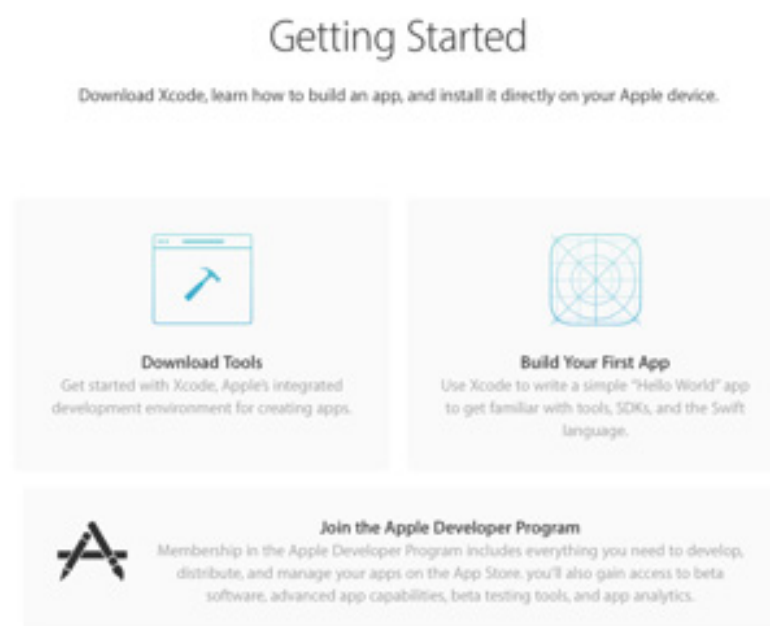
É possível utilizar um Apple ID existente para configurar um perfil de desenvolvedor. Caso seja necessário, podemos criar um novo Apple ID, para assim manter separados os assuntos pessoais dos profissionais.

A partir deste ponto, será apresentado um cadastro para preenchimento e criação da conta:



Com essa conta, mesmo gratuita, já é possível utilizar o Xcode para testar seu app em dispositivos físicos. Portanto, essa conta é suficiente enquanto estiver na fase de estudos e testes, pois ela já dará acesso ao portal de desenvolvedor, após o login no endereço <https://developer.apple.com/account/#/welcome>:

Observe, na figura a seguir, que o acesso da conta gratuita é restrito à documentação e download das ferramentas básicas de desenvolvimento:



Ainda observando a figura, vemos que a seção mais abaixo (Join the Apple Developer Program) indica que, para distribuir e gerenciar seu app na App Store, assim como ter acesso às versões beta e ferramentas de análise (analytics), é necessário aderir ao programa de desenvolvedor, migrando para a conta paga. É o que veremos a seguir.

Criar uma conta paga de desenvolvedor

A página para aderir à conta paga de desenvolver está no endereço <https://developer.apple.com/programs/>.

Ao abrir essa tela, será exibido, no lado superior esquerdo, o botão **Enroll**, através do qual se inicia o processo de cadastramento da conta paga:



A conta paga tem um custo anual de US\$ 99 ao ano, e durante o processo de cadastro, você terá a opção de escolher se a renovação será automática, ou não.

É necessário também efetuar o pagamento durante o cadastro, utilizando um cartão de crédito internacional.

Após a conclusão do cadastro, a Apple enviará dois emails de confirmação; um para confirmar a adesão ao programa de desenvolvedor e outro liberando acesso ao iTunes Connect, que terá um papel importante no processo de publicação.

Com a conta paga regularizada, você está pronto para continuar os passos necessários e publicar seu aplicativo na Apple Store.

Seção 4-3

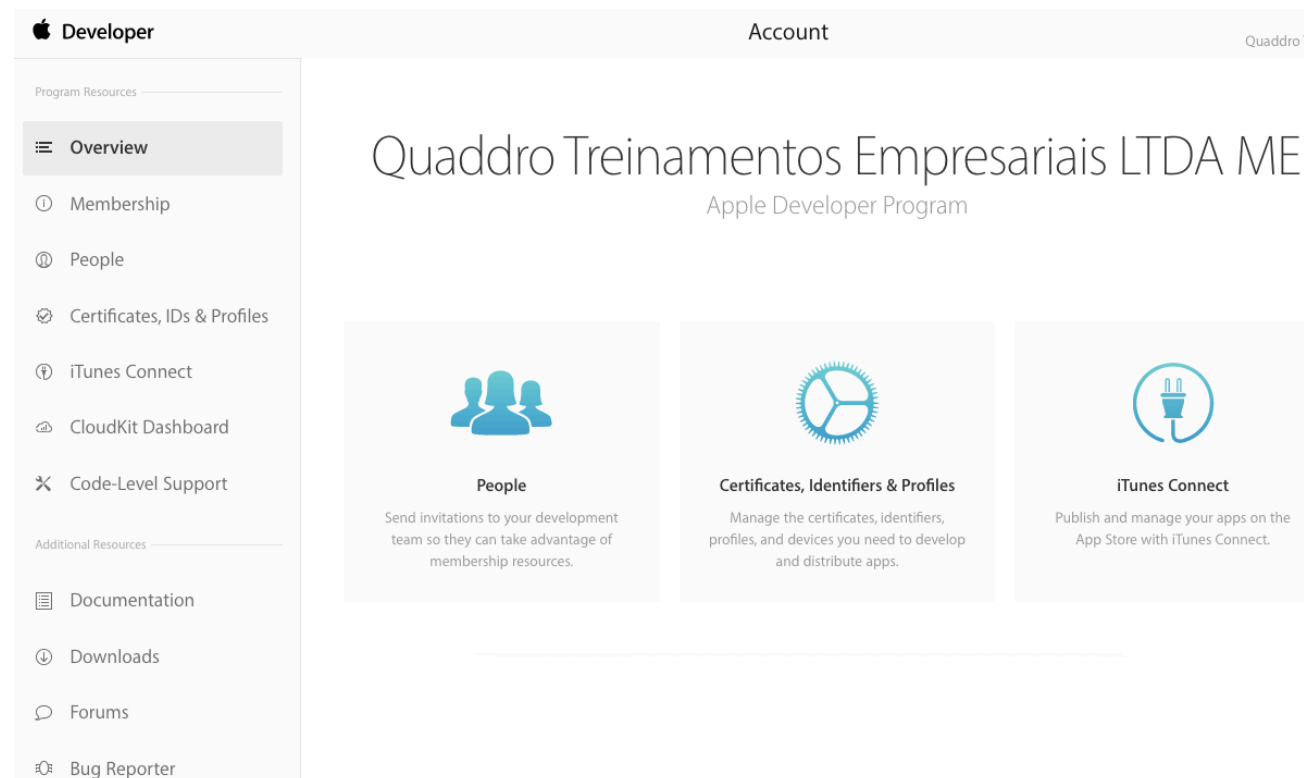
Configurações para desenvolvimento



Nesta fase do processo, é esperado que você já tenha instalado o Xcode em seu Mac. A configuração de um Mac para desenvolvimento envolve quatro passos, que veremos em detalhes nesta seção:

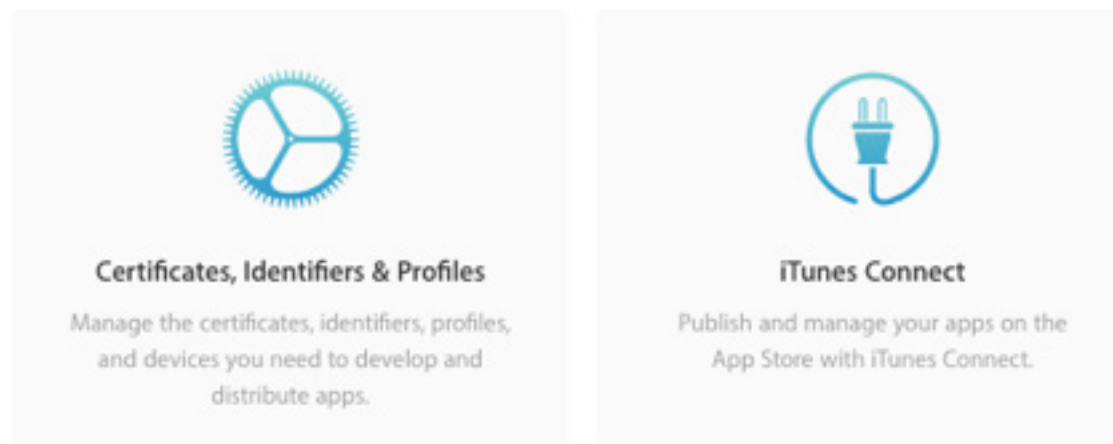
1. Criação de um certificado para o aplicativo;
2. Registrar seus dispositivos físicos para testes;
3. Criação de um identificador, também para o aplicativo;
4. Criação de um perfil para o desenvolvedor.

De posse de sua conta paga, ao logar novamente na página principal de desenvolvedor - <https://developer.apple.com/account>, você perceberá que vários outros conteúdos e acessos foram disponibilizados:



Na barra lateral, você encontrará guias de programação, códigos de exemplo, documentação geral, vídeos, acesso a fóruns de desenvolvimento e centro de suporte.

Mas é a área central da tela contém os itens que serão muito acessados durante o processo de desenvolvimento:



Inicialmente, vamos entender o significado de cada um deles:

Certificates: Sabemos que os devices iOS executam apenas aplicativos aprovados pela Apple, e que só podem ser instalados através da loja oficial - App Store. Para isso, todos os aplicativos devem possuir um certificado assinado pela Apple. A cada execução, o sistema verifica se o app possui um certificado válido. Adicionalmente, para que o desenvolvedor possa rodar seus aplicativos em um device físico, é necessário criar seu próprio certificado, como veremos a seguir, passo a passo.

App Identifiers: Cada aplicativo da App Store deve ter um identificador único. Esse identificador, ou App ID, é formado pela seguinte combinação:

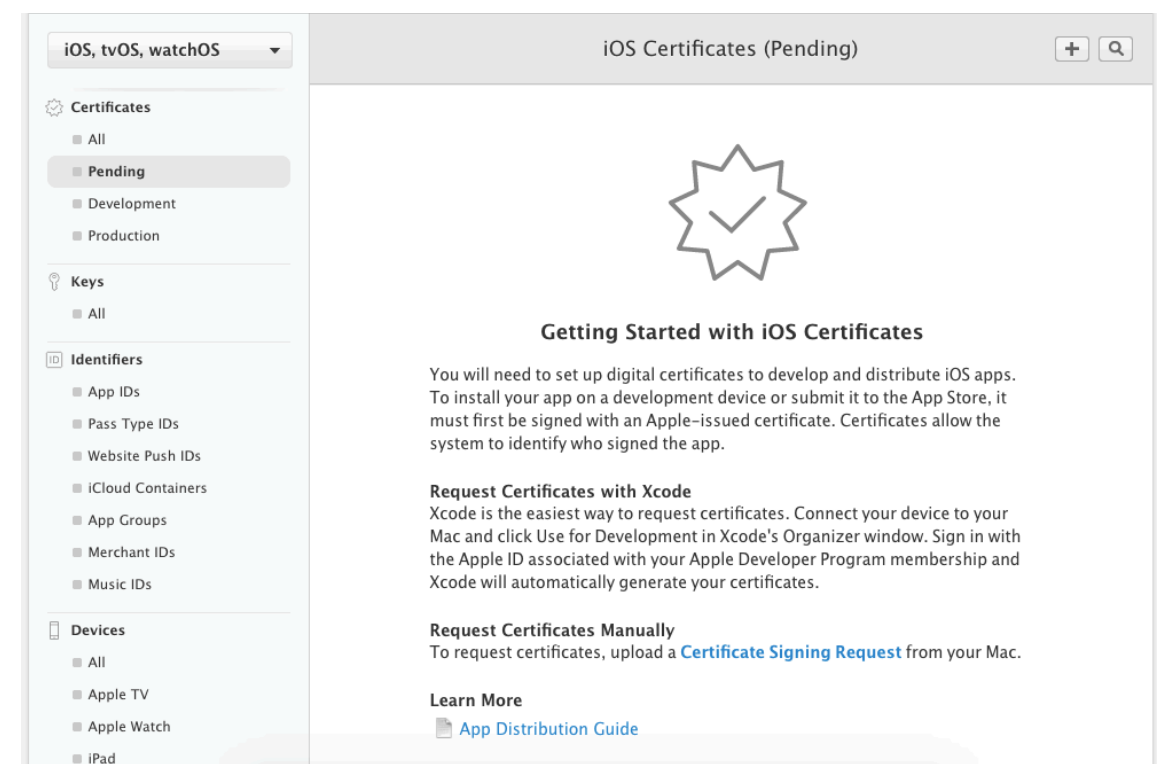
- Um prefixo gerado pela Apple
- Um sufixo gerado pelo desenvolvedor, através de uma string denominada Bundle ID do aplicativo. Veremos mais abaixo como gerar esse identificador.

Provisioning Profiles: A finalidade dos provisioning profiles é juntar tudo o que foi feito nos passos anteriores, incluindo os certificados, os identificadores dos dispositivos e os App IDs. Veremos adiante as formas de gerar esses arquivos.

Criando um certificado

É importante ressaltar que o Xcode gera automaticamente o certificado de seu aplicativo, mas de qualquer forma é interessante conhecer o processo manual.

Na página principal de desenvolvedor, selecione a opção **Certificates, IDs & Profiles**. Será exibida uma seguinte tela, onde os certificados podem ser criados:



Para criar um certificado, selecione o botão **+**. Inicialmente iremos criar um certificado de desenvolvimento:


Add iOS Certificate

Select Type

Request

Generate

Download

**What type of certificate do you need?**

Development

☒ **iOS App Development**
Sign development versions of your iOS app.

☐ **Apple Push Notification service SSL (Sandbox)**
Establish connectivity between your notification server and the Apple Push Notification service sandbox environment to deliver remote notifications to your app. A separate certificate is required for each app you develop.

Vá até o final da página e selecione o botão **Continue**. A página exibirá instruções de geração do certificado. A geração manual de um certificado é efetuada por um utilitário do macOS chamado **Keychain Access**.

Localize a abra o Keychain Access e efetue a geração do certificado, seguindo as instruções exibidas na tela. O certificado deverá ser salvo em disco.

Quando completar o processo paralelo de geração do certificado pelo Keychain Access, retorne á página de desenvolvedor e clique em **Continue**. Iremos agora efetuar o upload do certificado gerado para a página de desenvolvedor:



Generate your certificate.

When your CSR file is created, a public and private key pair is automatically generated. Your private key is stored on your computer. On a Mac, it is stored in the login Keychain by default and can be viewed in the Keychain Access app under the "Keys" category. Your requested certificate is the public half of your key pair.

Upload CSR file.

Select .certSigningRequest file saved on your Mac.

Choose File...

Selecione **Choose File**, localize no disco o certificado salvo pelo Keychain Access, e efetue o upload.

O certificado gerado está pronto agora para download:

Download, Install and Backup

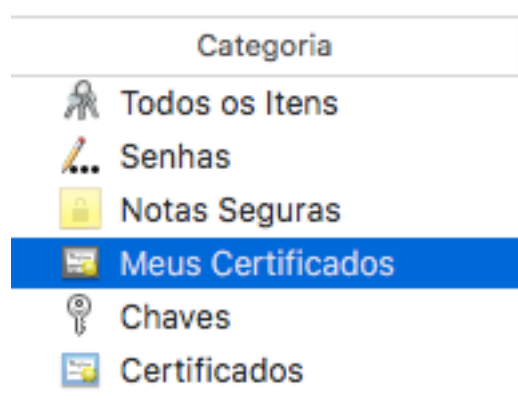
Download your certificate to your Mac, then double click the .cer file to install in Keychain Access. Make sure to save a backup copy of your private and public keys somewhere secure.



Name: iOS Development: Tiago Souza
Type: iOS Development
Expires: Feb 15, 2019

Download

Efetue o download do arquivo, localize-o pelo **Finder**, e, através de um duplo clique, efetue a instalação do certificado nas chaves do Mac. Pelo **Keychain Access**, é possível verificar os certificados instalados, através da opção **Meus Certificados**:



Agora vamos gerar um certificado para o ambiente de produção, o que nos permitirá subir o aplicativo para a loja. Na página de desenvolvedor, vá até o final da tela e escolha a opção **Add Another**. Em seguida vamos escolher um certificado de produção:

Production

- ☐ **App Store and Ad Hoc**
Sign your iOS app for submission to the App Store or for Ad Hoc distribution.

A partir deste ponto, repita os mesmos passos para a geração do certificado de desenvolvimento, e conclua o processo.

Registrar seus dispositivos para testes

Neste passo iremos registrar os dispositivos físicos que serão utilizados para testes do aplicativo, durante o desenvolvimento.

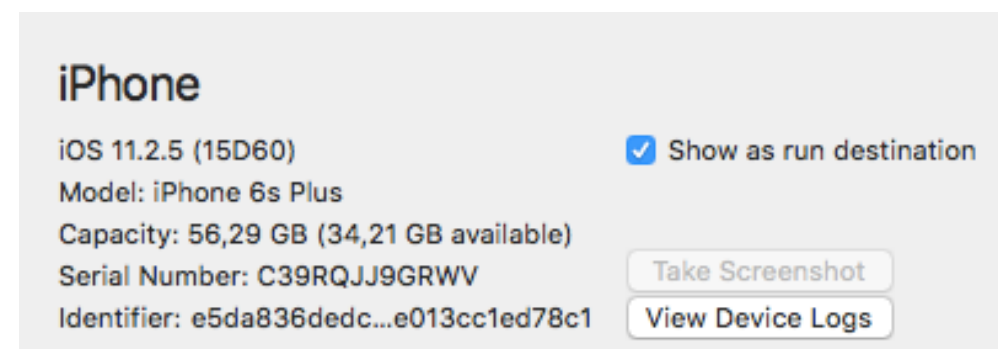
Para isso, retornaremos mais uma vez à página de desenvolvedor e faremos acesso à seção **Devices**, na barra lateral esquerda. Ao clicar no botão **+**, teremos acesso à tela de registro de novos devices:



Nesta seção, dois campos devem ser preenchidos:

1. O nome do device;
2. A identificação única de um device (UDID).

A identificação única pode ser obtida de várias formas. Uma delas é através do próprio Xcode, na opção de menu **Window > Devices and Simulators**. Será exibida uma tela com todos os devices e simuladores instalados, onde pode-se selecionar e localizar as informações do device que está conectado naquele momento:



De posse do **UDID**, retorne agora ao portal e preencha os campos solicitados:



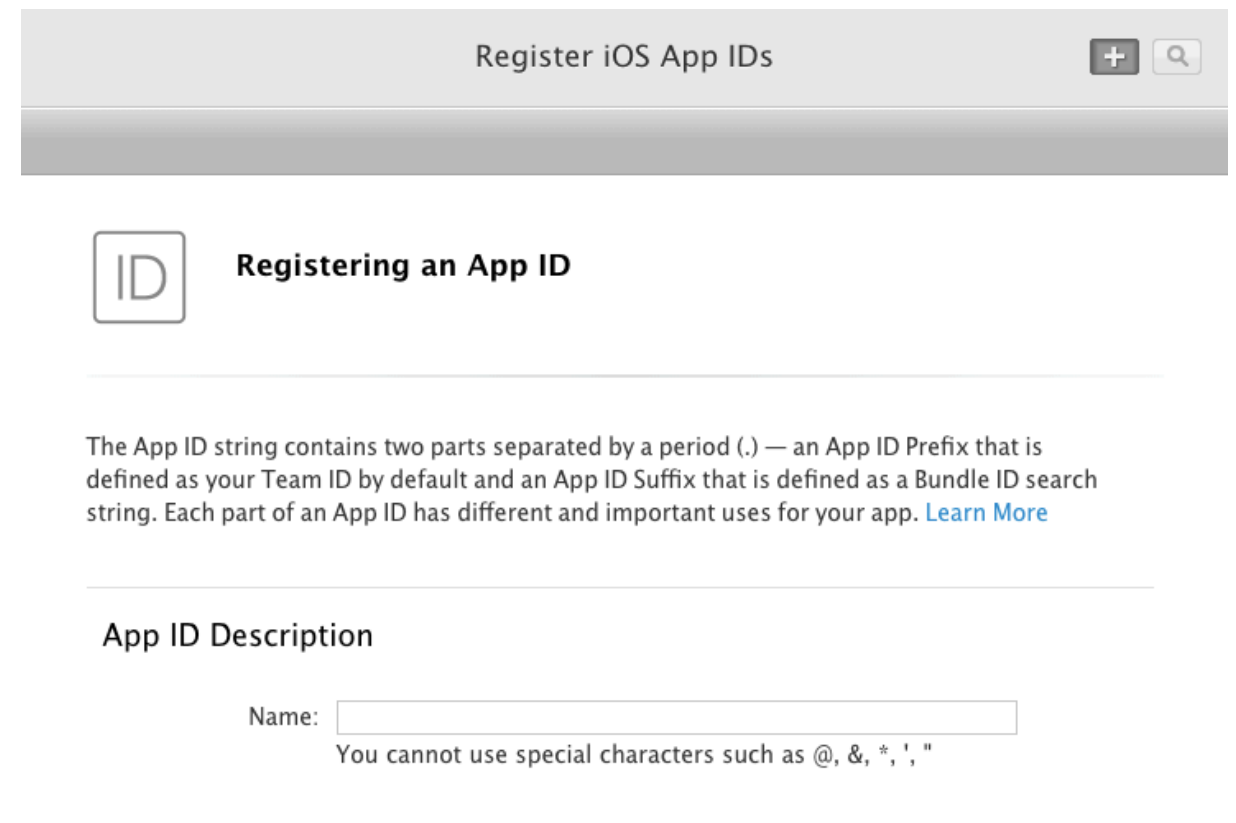
Ao clicar em **Register**, o processo estará finalizado, e o seu dispositivo passará a aparecer na relação de Devices autorizados.

Criar um App ID

Cada aplicativo desenvolvido deverá ter uma identificação única, chamada pela Apple de **App ID**.

Como colocamos anteriormente, um App ID é uma combinação de um prefixo gerado pela Apple com um sufixo criado por você, através de um campo de texto denominado **Bundle ID**. Essa combinação irá criar uma identificação única para o seu aplicativo.

A criação de um App ID é efetuada no portal do desenvolvedor. Na página principal, selecione a opção **Certificates, IDs & Profiles**. Na próxima tela, escolha **Identifiers**, e, em seguida, selecione a opção **App IDs**. E, assim como em procedimentos anteriores, clique no botão **+**:



Vamos agora preencher os seguintes campos:

- **App ID Description:** é uma descrição livre para o seu app. Exemplo: App Exemplo de Distribuicao. Não é permitida acentuação;

- **Bundle ID:** É o domínio e sua organização, escrita em modo reverso, incluindo o nome do app ao final. É o mesmo utilizado no Xcode, sempre que você cria um novo projeto. Exemplo: br.com.quaddro.PublicandoSeuApp. Observe que o prefixo do app foi automaticamente gerado pela Apple

Ao clicar em **Continue** e **Done**, seu App ID estará criado.

Criar Provisioning Profiles

O objetivo dos provisioning profiles é permitir que seu aplicativo seja instalado em um dispositivo físico. Um provisioning profile agrupa os certificados dos aplicativos, os identificadores dos devices e o App ID.

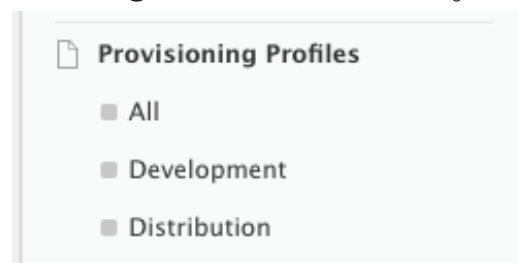
Temos dois tipos de provisioning profiles:

- **Development Provisioning Profiles:** permitem a compilação e testes do seu aplicativo durante o processo de desenvolvimento.

- **Distribution Provisioning Profiles:** permitem que o aplicativo seja submetido à App Store e também distribuído para os beta testers, através da plataforma Test Flight.

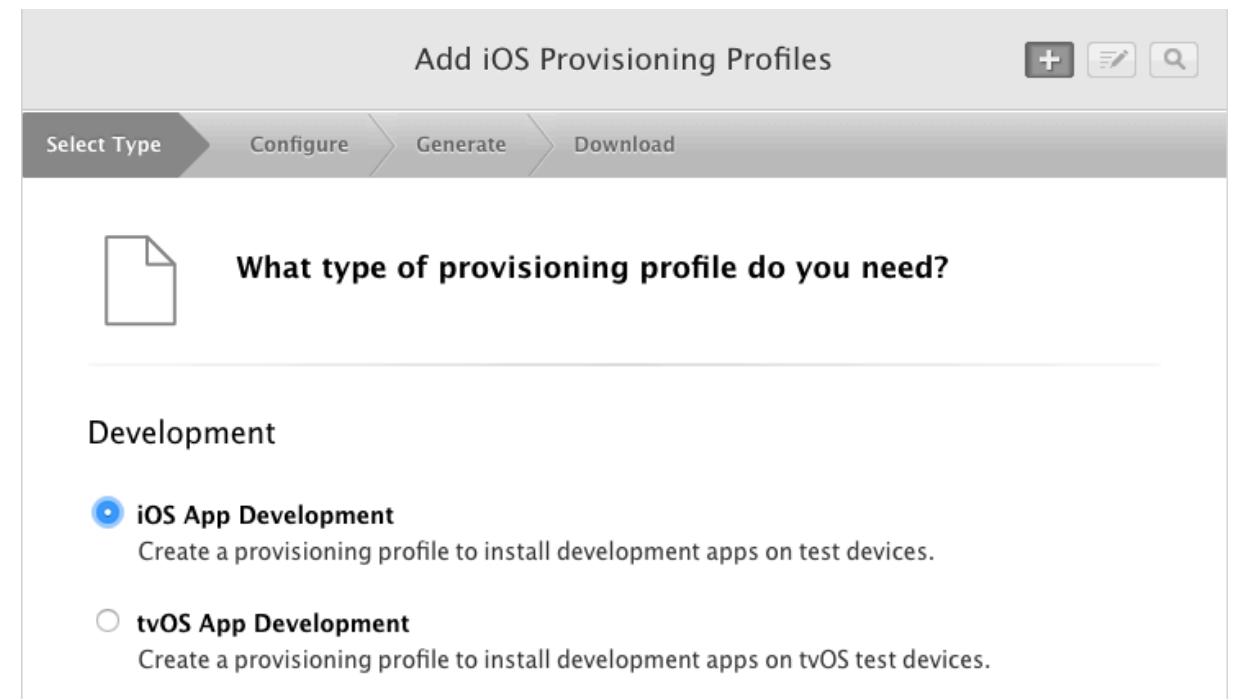
O Xcode pode gerar os provisioning profiles na própria ferramenta, mas como temos feito até agora, iremos conhecer o processo manual, através do portal do desenvolvedor.

A seção de **Provisioning Profiles** fica na seção mais inferior da tela:

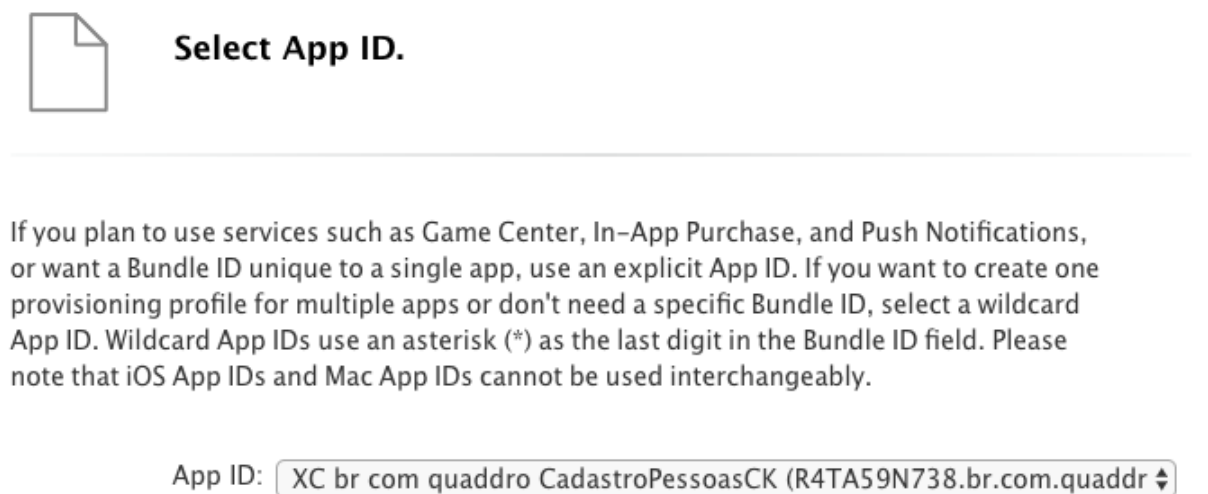


Selecione **All** e em seguida o botão **+** para criarmos um novo provisioning profile.

Inicialmente iremos criar um profile para desenvolvimento:



Após clicar em **Continue**, selecione o App ID desejado na próxima tela e clique em **Continue**:



Na próxima tela, selecione os certificados que deseja incluir no profile. Caso tenha vários desenvolvedores no time, inclua os que participam deste projeto:



Select certificates.

Select the certificates you wish to include in this provisioning profile. To use this profile to install an app, the certificate the app was signed with must be included.

☐ Select All 0 of 20 item(s) selected

☐ Tiago Souza (iOS Development)

☐ Tiago Souza (iOS Development)

Por último, selecione os dispositivos que pretende utilizar para instalar seu app:



Select devices.

Select the devices you wish to include in this provisioning profile. To install an app signed with this profile on a device, the device must be included.

☐ Select All 0 of 11 item(s) selected

☐ iPad

☐ iPad Quaddro

Após clicar em [Continue], na próxima tela, você deve dar um nome ao provisioning profile:



Name this profile and generate.

The name you provide will be used to identify the profile in the portal.

Profile Name:

Type: **iOS Development**

App ID: **XC br com quaddro CadastroPessoasCK (R4TA59N738.br.com.quaddro.CadastroPessoasCK)**

Certificates: **1 Included**

Devices: **1 Included**

Na próxima tela, após clicar em [Continue], efetue o download de seu profile:



Your provisioning profile is ready.

Download and Install

Download and double click the following file to install your Provisioning Profile.



Name: Perfil
Type: iOS Development
App ID: R4TA59N738.br.com.quaddro.CadastroPessoasCK
Expires: Feb 16, 2019

Download

Neste momento, você também pode gerar o provisioning profile de produção, para a distribuição ao aplicativo na App Store. Para isso, ao final da tela de download, selecione o botão **Add Another** e, na tela de inclusão, escolha App Store, na seção **Distribution**:

Distribution

☐ App Store

Create a distribution provisioning profile to submit your app to the App Store.

Validando o trabalho

Como saber se tudo deu certo? É através do próprio Xcode que podemos verificar se o provisioning profile foi gerado com sucesso.

Abra o projeto no qual você está trabalhando para distribuir. Aqui iremos abrir o projeto exemplo PublicandoSeuApp.

Antes de mais nada, é preciso adicionar a sua conta de desenvolvedor ao Xcode. A inclusão de uma conta de desenvolvedor é essencial para testar seu app em um dispositivo físico.

É importante lembrar que, para rodar o app no dispositivo físico, pode-se utilizar uma conta de simples, mesmo que não esteja associado ao programa de desenvolver, com conta paga.

Agora, com o projeto aberto, selecione o target principal e observe, na seção General, que existe uma seção **Signing**:

Como efetuamos todo o processo manualmente, vamos deixar **desmarcada** a opção **Automatically manage signing**.

▼ Signing

☐ Automatically manage signing

Xcode will create and update profiles, app IDs, and certificates.

No campo logo abaixo é que iremos especificar o provisioning que criamos para o desenvolvimento.

Observe que o Xcode está apontando um erro, indicando que o projeto requer um provisioning profile. Ao abrir o campo, ele oferece a opção de Import e Download.

Como no processo de geração do provisioning já efetuamos o download em nosso Mac, iremos agora importar o arquivo gerado anteriormente. Após localizar seu arquivo pelo Finder e importar, o Xcode irá indicar que o provisioning foi importado com sucesso!

▼ Signing (Debug)


Provisioning Profile

Import Profile...

Download Profile...

Team

Signing Certificate None

Status  "CRUDCoreData" requires a provisioning profile.

Select a provisioning profile for the "Debug" build configuration in the project editor.

ATENÇÃO: O mesmo procedimento deverá ser repetido para carregar o certificado de produção, no campo Signing (Release).

Ao final de todo o processo, você deverá ter ambos os provisionings regularizados.

Seu app está pronto para testes nos dispositivos físicos e distribuição na App Store!

Vale observar que todo o processo pode ser efetuado pelo Xcode, automaticamente, porém nem sempre dá certo. Além disso, você também não terá o domínio dos passos necessários assim como o conhecimento de tudo o que foi feito, o que sem dúvida é o cenário ideal para sua formação de desenvolvedor.

Veja também, no campo Team, que uma conta de desenvolvedor já está associada ao projeto.

Seção 4-4

Desenvolvimento e configuração do app



Essa fase envolve o desenvolvimento do app em si, a configuração no iTunes Connect, e upload do binário para o review da Apple. Essa última fase pode ser considerada, para os desenvolvedores mais experientes e que vieram de outras plataformas, como a geração do executável do aplicativo.

Iremos direto às configurações do iTunes Connect, visto que o estudo da linguagem Swift, os frameworks do iOS, as técnicas de programação e boas práticas são aprendidas nos cursos regulares de desenvolvimento da Quaddro.

O processo de upload, apesar de ser efetuado pelo Xcode, será detalhado na próxima fase, pois depende de configurações prévias no iTunes Connect.

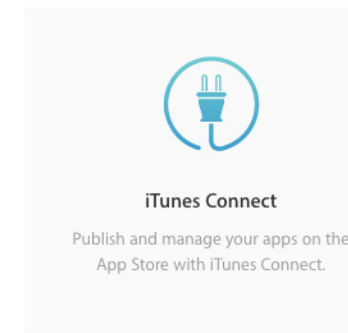
Configurando e utilizando o iTunes Connect

O iTunes Connect, como o próprio nome já indica, é o portal de conexão de seus aplicativos com a App Store.

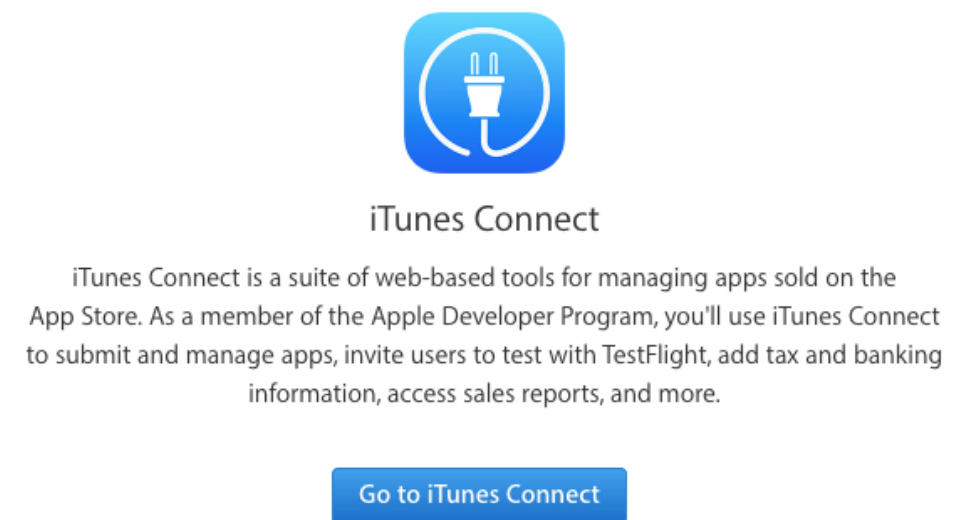
Nessa seção vamos conhecer as funcionalidades do portal, e explicar em detalhes a configuração dos dados contratuais e cadastrais, que são essenciais para que seu app possa ser submetido à App Store.

Configuração do iTunes Connect

O acesso ao portal também se dá pelo portal do desenvolvedor, através dos links da tela central:



Ao selecionar o link, você será levado a uma tela de apresentação:



Ao selecionar o botão **Go to iTunes Connect**, será exibida a página principal do portal, chamada de Dashboard:



Para que você esteja habilitado a submeter seu app para a App Store, nenhuma informação legal ou cadastral pode estar pendente. Portanto o passo essencial a ser seguido é selecionar a opção Contratos, impostos e informações bancárias.

Nesta funcionalidade nenhum item deve estar pendente, caso contrário você terá que regularizar a situação.

Vamos entender a finalidade de cada item:

- **Meus Apps:** submeter e gerenciar seus apps;
- **App Analytics:** verificar como seus apps estão se saindo na loja.
- **Vendas e Tendências:** verificar as vendas e estatísticas de downloads.
- **Pagamentos e relatórios financeiros:** controlar seus pagamentos e visualizar relatórios financeiros.
- **Usuários e Funções:** gerenciar os integrantes do time de desenvolvimento e as funções de cada um.
- **Contratos, impostos e informações bancárias:** visualizar e gerenciar os aspectos legais, assim como as informações financeiras.
- **Recursos e ajuda:** ter acesso aos recursos disponíveis e à ajuda do portal.

Seção 4-5

Submetendo seu aplicativo



Para que o seu aplicativo seja efetivamente publicado na loja ele terá que passar pelo processo de aprovação pela Apple, o qual chamamos de review. Quando o app for aprovado no review, ele então será automaticamente publicado. A seguir veremos cada fase deste processo.

Essa fase envolve também:

- Os requisitos para submeter o aplicativo;

- **Como criar uma nova entrada no portal para o seu app:** Processo efetuado pelo iTunes Connect;

- **Como efetuar o upload e submeter seu app para review:** Processo efetuado pelo Xcode;

- **Como submeter o app para review pela Apple:** Novamente através do iTunes Connect.

Os requisitos para submeter o aplicativo

Vamos a alguns requisitos para que você submeta seu app com mais segurança:

- O aplicativo foi testado em um device físico?

- Todas as funcionalidades foram validadas, inclusive as questões de layout e orientação?

- Caso seja um app universal, foi validado em um iPad também?

Faça uma revisão de seu app, procurando identificar fatores que estejam em desacordo com os guidelines da Apple. Utilize os links abaixo para esta verificação:

- Human Interface Guidelines:

- Store Guidelines:

Uma ótima forma de testar seu novo app é disponibilizá-lo para beta testers, que podem ser seus familiares, amigos, colegas de trabalho ou mesmo parceiros de negócio. O aplicativo é disponibilizado na plataforma **Test Flight**, da própria Apple. Caso não conheça, veja como funciona: <https://developer.apple.com/testflight/>

Tenha em mente que um curador da Apple, que não conhece nada de seu projeto, irá testar o app, e para isso terá que possuir todas as informações necessárias. Por exemplo, se o seu app inicia com um login, você deverá fornecer usuário e senha, para possibilitar o review. Se o curador não puder ter acesso ao app ou mesmo a parte dele, com certeza irá rejeitá-lo.

Você deve ainda fornecer junto ao app algumas informações que são padrão para a Apple:

- O nome do aplicativo;
- A descrição do aplicativo;
- O ícone do aplicativo, em resolução 1024x1024 pixels;
- Screenshots das telas do app.

Como criar uma nova entrada no portal para o seu app

Neste fase iremos utilizar o iTunes Connect: <https://itunesconnect.apple.com/>.

Após logar no portal, selecione **Meus Apps** e, na próxima tela, selecione **+**, seguido de **Novo App**, que é a seleção para um App iOS.

Será exibida a tela inicial de configuração:

Novo app

Plataformas ?

☒ iOS ☐ tvOS

Nome ?

PublicandoSeuApp

Idioma principal ?

Português (Brasil)

ID do pacote ?

App exemplo de distribuicao - com.saviT.Publicar

SKU ?

001

Cancelar

Criar

SKU é um termo utilizado para controle de estoque. É um identificador único para seu app. uma sugestão é utilizar o SKU para atribuir um número sequencial aos seus apps.

Após clicar em **Criar**, uma nova tela será exibida. A informação principal nesta tela é especificar uma categoria principal e, opcionalmente, uma categoria secundária para o seu app.

Informações do app

Informações gerais

Nome do app

App exemplo de distribuicao - com.saviT.PublicandoSeuApp

Idioma principal

Português (Brasil)

Categoria

Utilitários

Subcategoria (opcional)

Id do app

001

ID do app

001

Id do app

001

Informações de preço

Preço

Preço

Preço

Informações de disponibilidade

Disponibilidade

Disponibilidade

Disponibilidade

Utilizando agora o menu lateral, informaremos preço e disponibilidade. O nosso será grátis.

Preços e disponibilidade

Tabela de preços

Preço

0

Data inicial

0

Data final

0

Preço

0

Data inicial

0

Data final

0

Após salvar, iremos, novamente utilizando o menu lateral, para **Preparar para envio**. Nesta outra tela subiremos as screenshots do aplicativo. Em nosso exemplo, subiremos apenas as telas para iPhone.

Utilizando a seção de telas para iPhones de 5,5 polegadas, subimos quatro imagens que, para este dispositivo, em modo **Portrait**, devem possuir a resolução de **1.242x2.208 pixels**. Veja como ficou:



Se tiver algum problema com as dimensões e resolução das imagens, as informações poderão ajudar: <http://help.apple.com/itunes-connect/developer/#/devd274dd925>.

Mais abaixo na mesma tela, vamos colocar uma descrição para o app, assim como palavras-chave para que os clientes encontrem o nosso app, quando efetuarem pesquisa na App Store. Caso essa seção não esteja aparecendo, clique, no menu principal, sobre o título **Apps para iOS 1.0**:

App para iOS 1.0

Preparar para envio

Salvar

Enviar para revisão

Texto promocional

Palavras-chave

IMC, peso, altura, saúde

URL de suporte

http://example.com

URL de marketing

http://example.com (opcional)

Descrição

Esse é um App de exemplo, que busca cobrir todos os passos necessários para geração e publicação de seus próprios aplicativos na App Store. No app de exemplo, estamos utilizando uma ferramenta para cálculo do IMC - Índice de Massa Corporal. O cliente introduz seu peso e sua altura e o app calcula o IMC. O resultado é mostrado de forma gráfica, desenhando uma silhueta de acordo com o índice calculado.

O próximo passo será carregar o ícone do aplicativo, para uso na App Store. O upload do ícone fica ainda mais abaixo na tela, lembrando que o arquivo deve ter a resolução de **1.024x1.024 pixels**:

Informações gerais do app

Ícone do app

Copyright

Copyright © 2017 Quaddio Treinamentos Empresariais LTDA

Informações de contato do representante comercial

☐ Exibir informações de contato do representante comercial na App Store connect.

Carlos Savi

Em seguida, procure pelo atalho **Editar**, que fica ao lado da palavra Classificação. É nesta seção que você deverá informar a classificação indicativa de seu app. Seja honesto, pois o revisor da Apple tem poder de alterar as informações, caso ele não concorde:

Editar classificação

Para cada descrição de conteúdo, selecione o nível de frequência que melhor descreve seu app. A classificação do app que será exibida na App Store é a mesma em todas as suas plataformas. Ela é baseada na plataforma do app com a classificação de idade mais alta. Saiba mais.

Apps não devem conter obscenidades, pornografia, ofensas ou difamações ou materiais de qualquer tipo (texto, gráficos, imagens, fotos etc.), ou outro conteúdo ou materiais que no julgamento da Apple possam ser considerados censuráveis.

Descrição de conteúdo da Apple	Nenhum	Pouco frequente/moderado	Frequente/intenso
Violência em desenhos animados ou fantasia	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Violência realista	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Violência realista sádica ou prolongada	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Conteúdo obsceno ou humor grosseiro	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Temas adultos/sugestivos	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Temas de terror/medo	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Informações médicas/sobre tratamentos	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Uso ou referências a álcool, tabaco ou drogas	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Jogos de azar simulados	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Conteúdo sexual ou nudez	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Conteúdo com imagens sexuais e nudez	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
	Não	Sim	
Acesso irrestrito à web	<input checked="" type="radio"/>	<input type="radio"/>	
Jogos de azar e concursos	<input checked="" type="radio"/>	<input type="radio"/>	

Na sequência, devem ser informados:

- Os dados para contato, caso o revisor tenha alguma dúvida;

- Informações adicionais para que o revisor possa testar o aplicativo, como usuário/senha para login ou algum detalhe específico.

Forneça o máximo de detalhes possível, para que o revisor não tenha nenhuma dúvida ou problema durante o processo de revisão:

Informações para a equipe de revisão dos apps

Informações para iniciar sessão

Forneça um nome de usuário e uma senha para que seja possível iniciar sessão em seu app. Estas informações são necessárias para concluir a revisão do app.

☐ Necessário iniciar sessão

Informações de contato

Carlos Alberto

carlos.alberto@br

Savi

savi.developer@yahoo.com.br

Notas

Por último, indique o timing de liberação do app, após a aprovação. O procedimento padrão é liberar o app automaticamente, mas você tem outras opções, como mostrado a seguir:

Liberação da versão

Depois que o app for aprovado, poderemos liberá-lo imediatamente, ou se você mesmo preferir liberá-lo, escolha uma data ou faça a liberação manualmente a qualquer momento após a aprovação. Enquanto o app estiver com o status "Liberação do desenvolvedor pendente", você poderá distribuir códigos promocionais, continuar testes no TestFlight Beta ou rejeitar a liberação e enviar uma nova compilação. Independentemente da opção escolhida é necessário processarmos seu app antes de disponibilizá-lo na App Store. Enquanto o app estiver com o status "Processando para a App Store", não será possível obter novos códigos promocionais, convidar novos testers ou rejeitar o app.

☐ Liberar manualmente esta versão

☒ Liberar automaticamente esta versão

☐ Liberar automaticamente esta versão após a revisão do app e não antes de

Data e hora local

9 de jul de 2017

00:00

Antes de finalmente submeter o app para revisão, temos que retornar ao Xcode para efetuar o Build, para e carregá-lo no iTunes Connect.

Como efetuar o upload e submeter seu app para review

Nesta fase do processo, o nosso app está totalmente configurado no iTunes Connect, faltando apenas efetuar o Build.

A geração é efetuada pelo Xcode, com o projeto aberto. Teremos a sequência destes passos:

- Trocar o esquema de compilação para **Generic iOS Device**;
- No menu **Product**, executar a opção **Archive**;

Os provisioning profiles devem estar gerados e certificados para que o procedimento de archive seja possível

- Após a geração, será exibida a seguinte tela:

ArchivesCrashes

iOS Apps

Name	Creation Date	Version
PublicandoSeuApp	9 de jul de 2017 19:00	1.0 (1)

Archive Information

PublicandoSeuApp

9 de jul de 2017 19:00

Upload to App Store...

Validate...Export...

Details

Version 1.0 (1)

Identifier com.savi11.PublicandoSeuApp

Type iOS App Archive

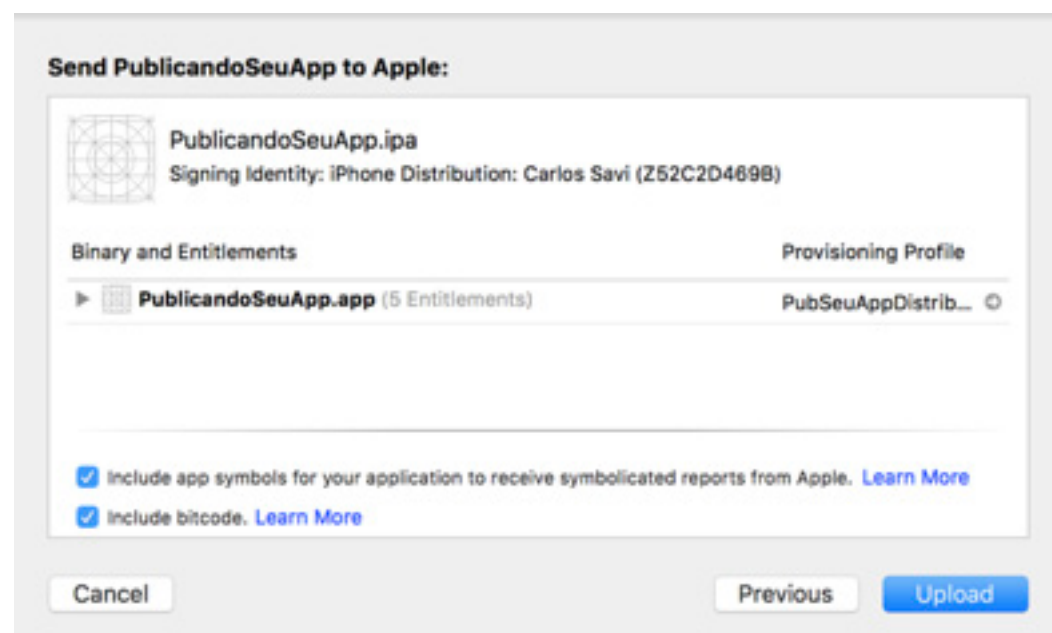
Download dSYM...

Description

No Description

64

- Após clicar em **Upload to App Store** será solicitada a confirmação da conta de desenvolvedor e exibida uma tela final de confirmação:



- Após selecionar **Upload**, e aguardar o tempo de processamento, teremos então o binário disponível para review;

- É possível que ocorram alguns erros durante esse processo. É neste momento que tudo deve se encaixar para a geração do binário - Bundle ID; provisioning profile, developer account, etc. Se houver qualquer inconsistência, o Xcode apontará o erro durante a geração do upload, e você terá que resolver para prosseguir;

- Quando o procedimento for finalizado, teremos a tela de sucesso.



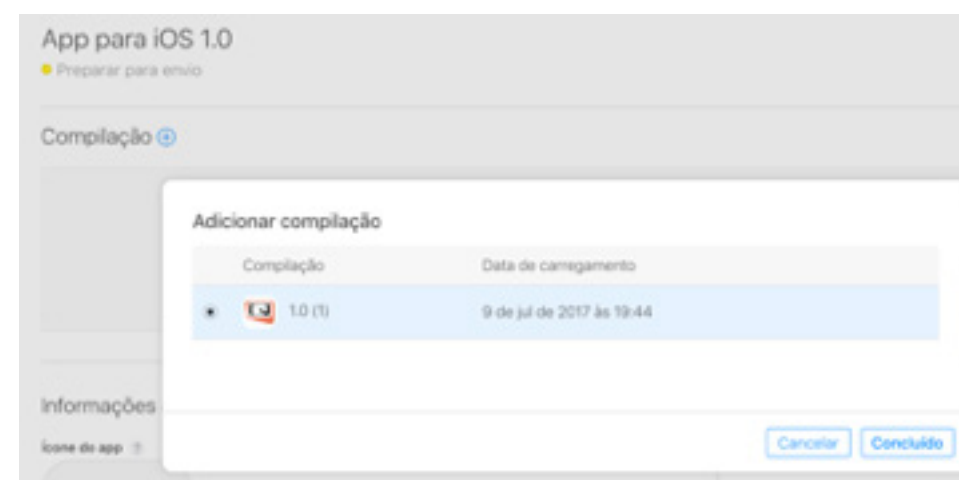
Upload Successful

Thank you for submitting your app to the iOS App Store.

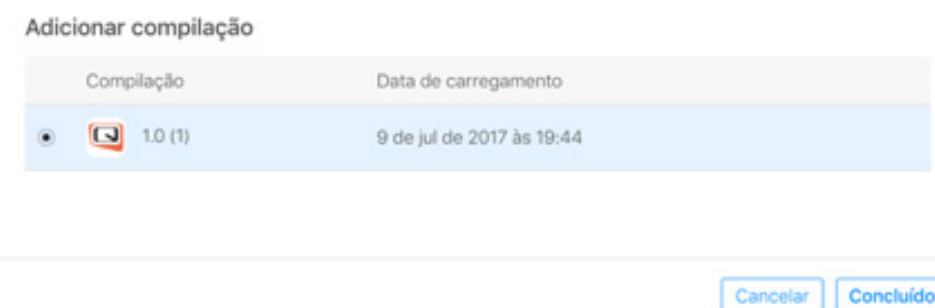
Como submeter o app para review pela Apple

Caso não esteja logado, retorne ao iTunes Connect, localize e selecione seu app em **Meus Apps**, e faça novamente acesso à opção **Preparar para envio**, no menu lateral.

A seção de Compilação deve mostrar um botão **+** ao lado, indicando que há um binário disponível para upload. Caso tenha acabado de executar o passo anterior, é possível que tenha que aguardar um tempo antes de prosseguir:



Ao clicar sobre o botão **+**, o binário recém gerado deve ser exibido:



Após adicionar a compilação e salvar a operação, o botão **Enviar para Revisão** será habilitado.

Agora é clicar, confirmar algumas operações, e aguardar a mágica acontecer!

O status do aplicativo mudará para **Waiting for Review**. Atualmente, aqui no Brasil, a Apple otimizou muito seus processos está levando de 24 a 48 horas para efetuar uma revisão. Após a aprovação, você terá seu app automaticamente publicado, caso tenha configurado dessa forma, ou estará apto para publicação.

Caso seja rejeitado, você será comunicado sobre os motivos da rejeição, terá que atuar na correção, para então reiniciar o processo de submissão do app.

Após a publicação na loja, será iniciada uma nova fase, onde o desenvolvedor passa a acompanhar o desempenho do app, em relação à downloads, vendas, se for o caso, e análise de desempenho e crashes. É uma fase de melhoria contínua, não somente das funcionalidades do aplicativo como de eventuais correções que sejam necessárias.