



INSTITUTO DE GESTÃO E  
TECNOLOGIA DA INFORMAÇÃO

---

## Arquitetura de Aplicações para iOS

---

Rafael Lobato

2019

## Arquitetura de Aplicações para iOS

Rafael Lobato

© Copyright do Instituto de Gestão e Tecnologia da Informação.

Todos os direitos reservados.

## Sumário

---

Capítulo 1. Introdução ao iOS .....	5
1.1. Sistema Operacional Mac OS X .....	5
1.2. Sistema Operacional iOS .....	5
1.3. Dispositivos iOS .....	6
1.4. Apple Developer Program .....	7
1.5. App Store.....	8
Capítulo 2. IDE de Desenvolvimento XCode .....	11
2.1. Versão atual .....	11
2.2. Templates de projetos .....	13
2.3. Interface Builder .....	20
2.4. Storyboards .....	21
2.5. Extensões, bibliotecas e frameworks .....	22
Capítulo 3. Linguagens de Programação .....	29
3.1. Objective-C .....	29
3.2. Swift.....	33
Capítulo 4. Ciclo de vida de desenvolvimento na plataforma .....	36
4.1. Desenvolvimento .....	36
4.2. Versionamento .....	45
4.3. Testes.....	48
4.4. Deployment .....	49
Capítulo 5. Persistência de dados .....	55
5.1. Arquivo .....	55
5.2. SQLite .....	58

5.3. Core Data .....	61
5.4. iCloud .....	66
Capítulo 6. Comunicação Remota .....	69
6.1. Requisições HTTP.....	69
6.2. Serviços REST .....	75
Capítulo 7. Processos e Threads.....	79
Capítulo 8. Sensores .....	91
Capítulo 9. Tópicos avançados .....	101
9.1. MLKit .....	101
9.2. ARKit .....	103
Referências.....	107

## Capítulo 1. Introdução ao iOS

---

Neste capítulo é apresentada uma visão geral do que é o sistema operacional iOS, presente em todos os dispositivos móveis da Apple. É importante conhecer o que ele oferece em termos de funcionalidade antes de iniciar o desenvolvimento de aplicativos na plataforma.

### 1.1. Sistema Operacional Mac OS X

---

Primeiramente falando do predecessor do iOS, o sistema operacional Mac OS X foi desenvolvido pela Apple, baseado no sistema operacional Unix, para ser o sistema operacional de suas máquinas de grande porte. Foi especialmente projetado para ser executado em computadores Mac, uma vez que a mesma empresa fabrica o hardware e desenvolve um software totalmente especializado.

Atualmente, outros hardwares de grande porte utilizam o Mac OS X como sistema operacional, são eles o Mac Mini, iMac, MacBook Pro e Air.

### 1.2. Sistema Operacional iOS

---

A partir do sistema Mac OS X, a Apple adaptou o seu sistema operacional de grande porte para atender dispositivos mobile criando o iOS, que nada mais é que uma versão reduzida do Mac OS X.

Foi originalmente projetado para o iPhone de 2007 e apresentou conceitos como o Multitouch e controle por gestos, abstraídos para os desenvolvedores no framework Cocoa Touch.

Assim como o Mac OS, se destaca principalmente pela maior integração com outros dispositivos Apple.

### 1.3. Dispositivos iOS

---

Dispositivos iPhone ainda comumente encontrados em uso:

- Iphone 4S, 5, 5S, 5C, 6, 6 PLUS, 7, 7 Plus, 8, 8 Plus e iPhone X.

iPod Touch – Praticamente o mesmo layout do iPhone sem a opção de efetuar ligações.

iPad – Tablets:

- Ipad 2, 3, Air, mini, Retina e Pro.

Apple TV – Dispositivo que possui conectividade Ethernet, Wi Fi e saída HDMI, adicionando características de smart tv a uma televisão. Possui aplicativos nativos e uma variação do iOS, o TV OS, como sistema operacional. Possui loja App Store própria.

Apple Watch – Novo conceito de smart watch da Apple. Utiliza uma variação do iOS chamada Watch OS.

Características de hardware dos dispositivos:

- iPhone 8:
  - Processador A11 Apple 64 Bits, ARMv8-A six-core CPU, 2 high-performance cores at 2.39 GHz, 2GB RAM e 4.7 Polegadas
- iPhone X:
  - A11 com GPU e 3GB RAM.
- iPad Pro:
  - Processador A10X Fusion Apple 64 Bits, 2.38 GHz ARMv8-A six-core CPU, with three high-performance Hurricane cores, integrado a 12-core graphics processing unit (GPU), 4GB RAM e 12.9 Polegadas.

Segundo informações disponibilizadas pela própria Apple, mais de 800 milhões de dispositivos mobile foram vendidos, sendo eles 500 milhões iPhones, 200 milhões iPads e 100 milhões iPod Touch. Foram 130 milhões de novos usuários somente no último ano.

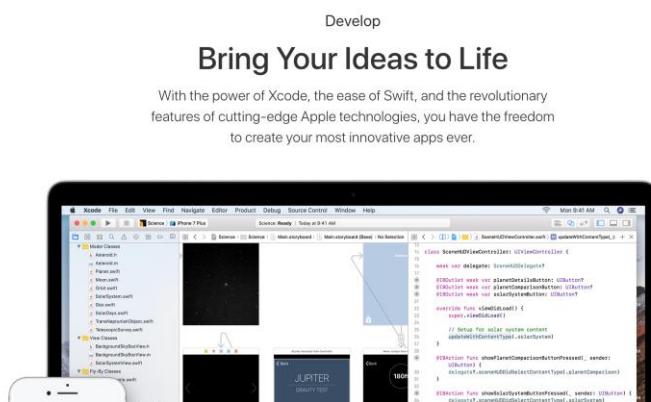
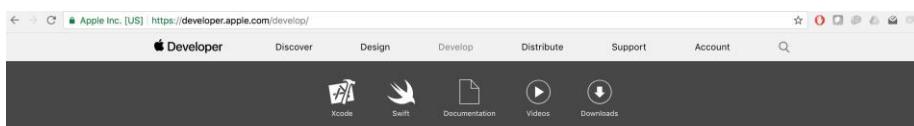
## 1.4. Apple Developer Program

O Apple developer program é um programa de cadastro único de desenvolvedores que os credenciam a ter acesso aos principais recursos e fóruns de desenvolvimento.

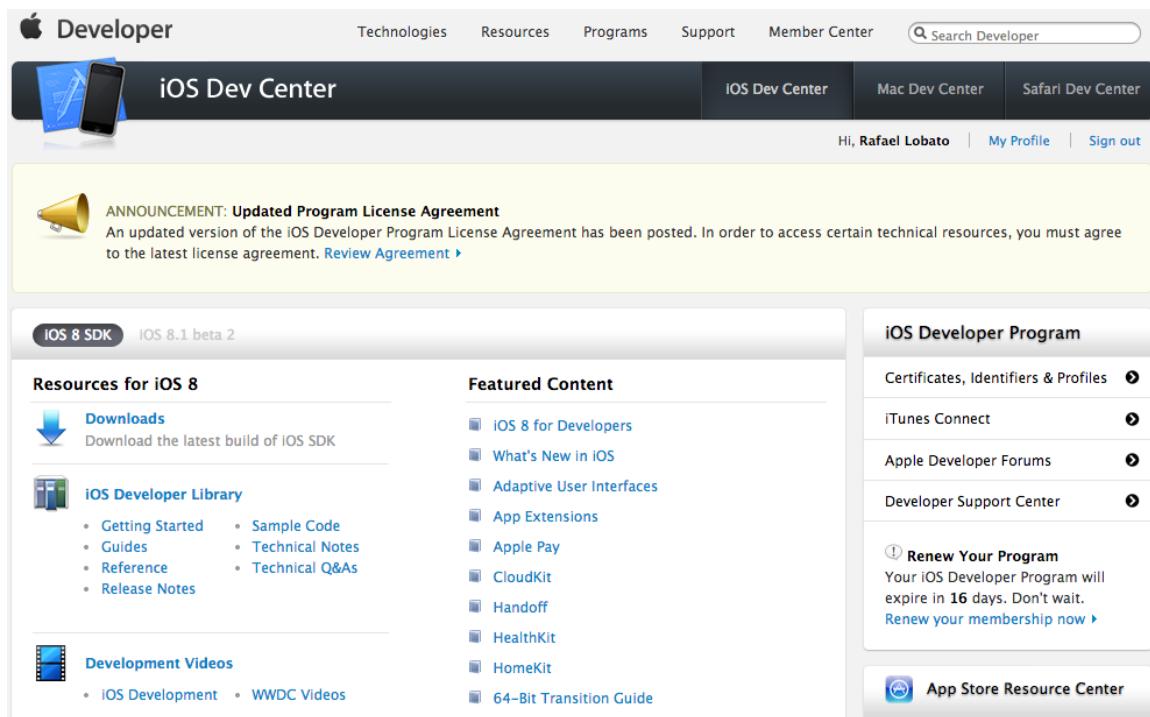
Benefícios:

- Acesso a versões beta do iOS.
- Acesso a versões beta IDE Desenvolvimento e SDK.
- Acesso a fóruns de desenvolvimento restritos.
- Testes On Device.
- Suporte técnico.
- Distribuição de aplicativos App Store.

**Figura 1 – Página inicial do site do programa de desenvolvedores.**



**Figura 2 – Página inicial do site do programa de desenvolvedores.**



The screenshot shows the Apple Developer website's homepage. At the top, there's a navigation bar with links for Technologies, Resources, Programs, Support, Member Center, and a search bar. Below the navigation is a banner for the iOS Dev Center, featuring icons for an iPhone and a Mac. The banner also includes links for iOS Dev Center, Mac Dev Center, and Safari Dev Center. A user profile for "Rafael Lobato" is shown with links for "My Profile" and "Sign out". A yellow announcement box at the top right says "ANNOUNCEMENT: Updated Program License Agreement" and encourages users to review the latest license agreement. The main content area has sections for "Resources for iOS 8" (with links for Downloads, iOS Developer Library, and Development Videos), "Featured Content" (listing various developer resources like iOS 8 for Developers, What's New in iOS, Adaptive User Interfaces, etc.), and a sidebar for the "iOS Developer Program" (with links for Certificates, Identifiers & Profiles, iTunes Connect, Apple Developer Forums, and Developer Support Center). It also includes a reminder about an expiring membership and a link to renew it, along with an "App Store Resource Center" button.

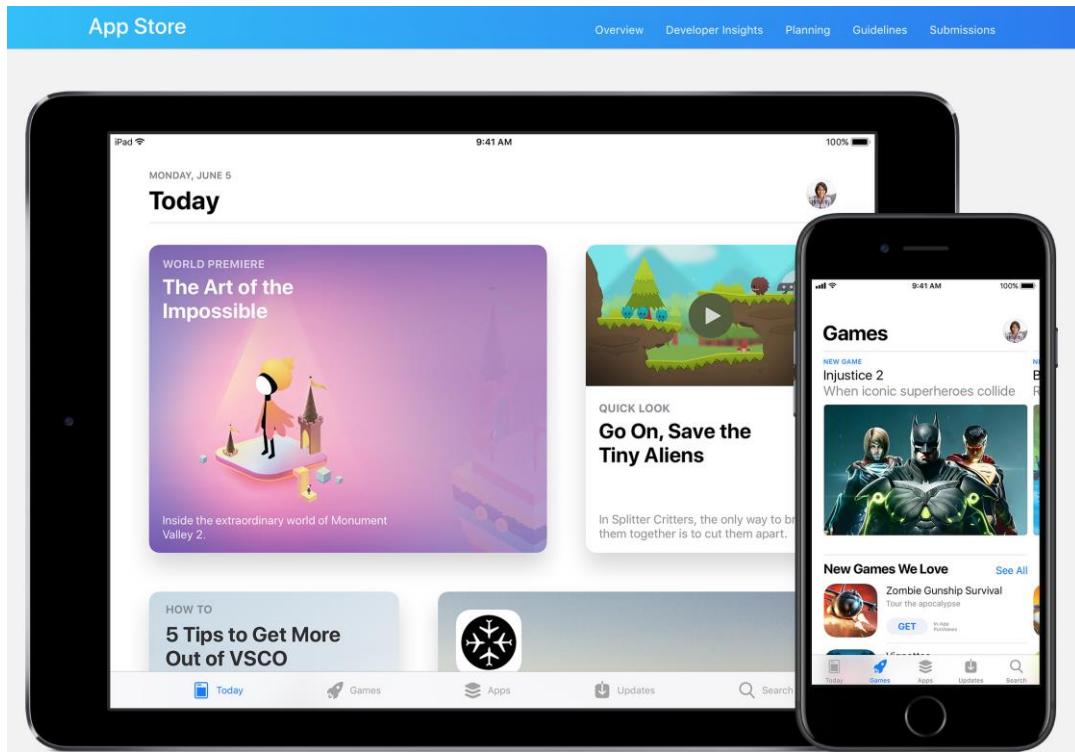
## 1.5. App Store

Uma grande criação da Apple foi a App Store. Conceito de vendas e distribuição online de aplicativos. Posteriormente, foi expandido inclusive para a plataforma Mac OS X. Na App Store estão presentes para download mais de 1.2 milhões de aplicações para iOS, sendo destes 500 mil otimizadas para iPad. Juntas, totalizam 60 bilhões de downloads.

É possível efetuar venda de apps avulsos ou em volume, para empresas, por exemplo, distribuírem internamente conforme a necessidade.

Tudo o que é vendido na App Store é faturado mensalmente e pago para os desenvolvedores, sejam pessoas físicas ou jurídicas. 30% de todo o valor adquirido fica com a Apple.

**Figura 3 – App Store em um iPad e iPhone.**



Outra forma de se obter lucro é disponibilizar conteúdo para venda dentro de aplicativos. Em diversos softwares ou jogos encontramos conteúdo opcional para compra. Estes são tributados da mesma forma por parte da Apple, mas muda a estratégia de lucro por parte do dono do software. Muitos software são gratuitos e todo o lucro vem com a venda de conteúdo adicional, conhecida como In App Purchases.

**Figura 4 – In app purchases.**



A Apple também oferece uma plataforma de propagandas a serem disponibilizadas em qualquer tipo de aplicação. Aos desenvolvedores basta reservar um espaço na interface das aplicações que a Apple divulga propagandas de seus parceiros comerciais. O pagamento é feito por visualização de propagandas por parte dos usuários.

**Figura 5 – iAd.**



## Capítulo 2. IDE de Desenvolvimento XCode

---

A IDE de desenvolvimento de aplicativos e softwares na plataforma Apple é chamada de XCode. Possui todo o ferramental necessário para desenvolvimento, testes e publicação de softwares profissionais.

### 2.1. Versão atual

---

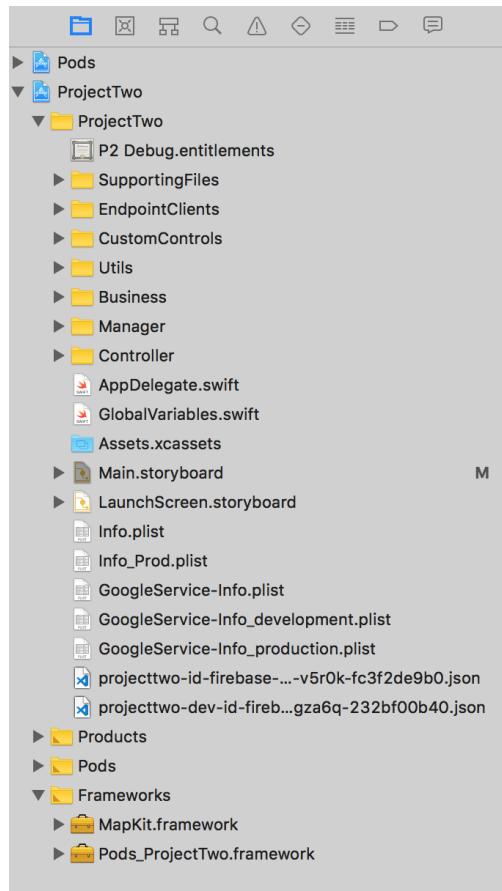
A versão mais atual do XCode trouxe uma série de inovações que facilitaram ainda mais o desenvolvimento. Uma nova linguagem de desenvolvimento, alternativa ao Objective C, chamada Swift, muito mais parecida com as linguagens modernas de desenvolvimento.

Na parte de desenvolvimento de interfaces gráficas temos agora o Live rendering e o View Debugger, que atuam na renderização automática de conteúdo gráfico e na depuração da parte gráfica, exibindo todas as camadas da interface, facilitando na investigação de problemas.

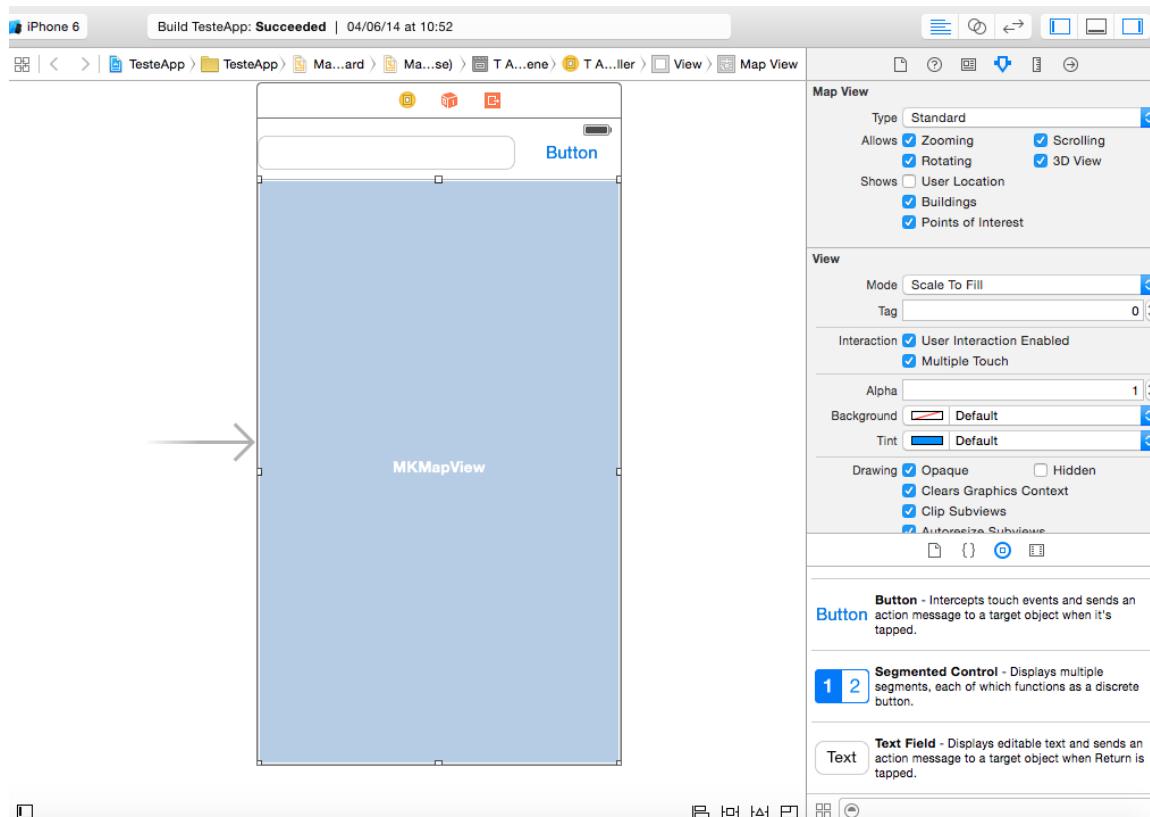
Outra facilidade foi a geração automática de arquivos de localização a partir de um código com strings fixas.

A seguir, as principais áreas encontradas no XCode para tratar os diversos pontos de um projeto criado nesta IDE.

**Project Navigator** – Área que exibe todos os arquivos que compõem uma solução. Código fonte e arquivos de configuração.

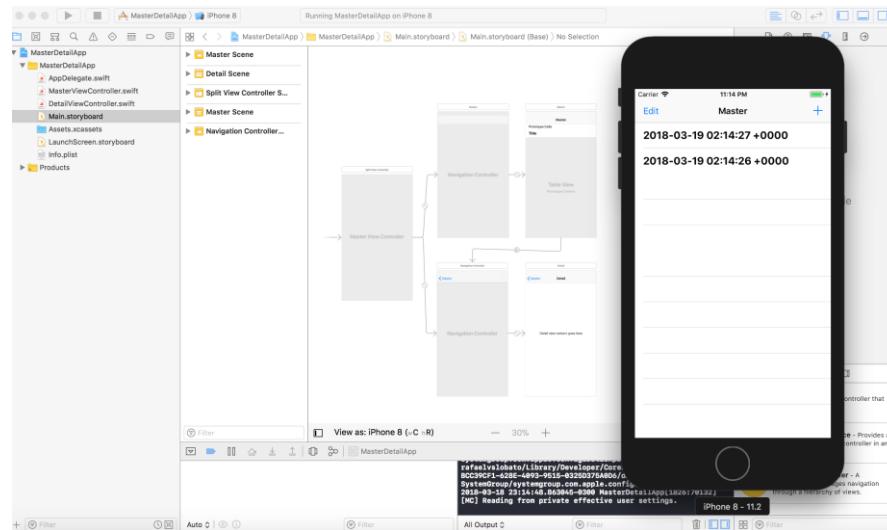
**Figura 6 - Project Navigator.**

**Interface Builder, Attribute Inspector e Object Library** – Atuam respectivamente no desenvolvimento gráfico de interfaces, edição de cada atributo presente na interface sendo desenvolvida e biblioteca de componentes disponíveis para adição visual nos projetos.

**Figura 7 – Detalhe do Interface Builder, Attribute inspector e Object Library.**

## 2.2. Templates de projetos

Existem vários templates de projetos predefinidos no XCode, tanto para projetos iOS quanto OSX, que já fornecem uma configuração inicial para diversos tipos de projeto. O primeiro deles é o Master Detail, que oferece um layout dividido em uma lista de itens que, ao selecionados, exibem detalhes complementares de visualização.

**Figura 8 – Master Detail Template.**

Este template tem comportamento um pouco diferente para aplicações iPhone ou iPad, respondendo ao tamanho da tela. Desta forma ele apresenta o detalhe em uma interface distinta, ou na mesma tela dependendo do espaço disponível.

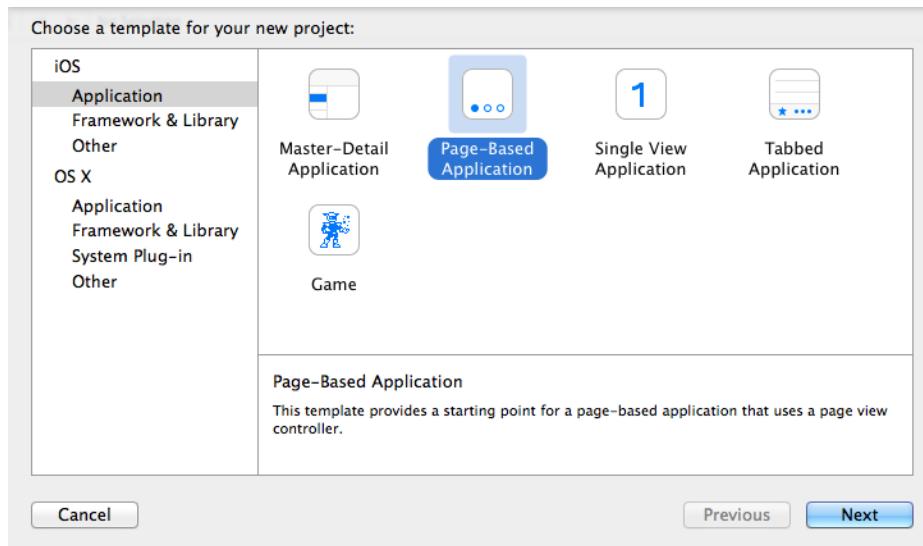
**Figura 9 – Detalhe de um app utilizando este template.**

**Figura 10 – Template master detail em apps para iPad.**



O page-based application é ideal para aplicações que possuem várias telas ou views que não necessariamente tem relação entre sí. As páginas são navegadas utilizando o gesto de swipe.

**Figura 11 – Page-Based Template.**



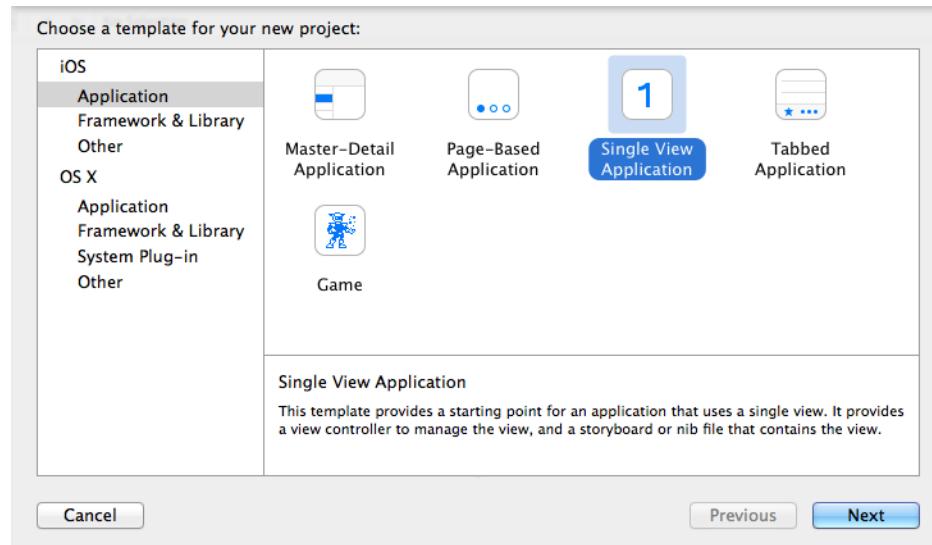
Um exemplo de aplicação nativa que utiliza este template é a aplicação de tempo do iPhone.

**Figura 12 – App utilizando o template Page-Based.**



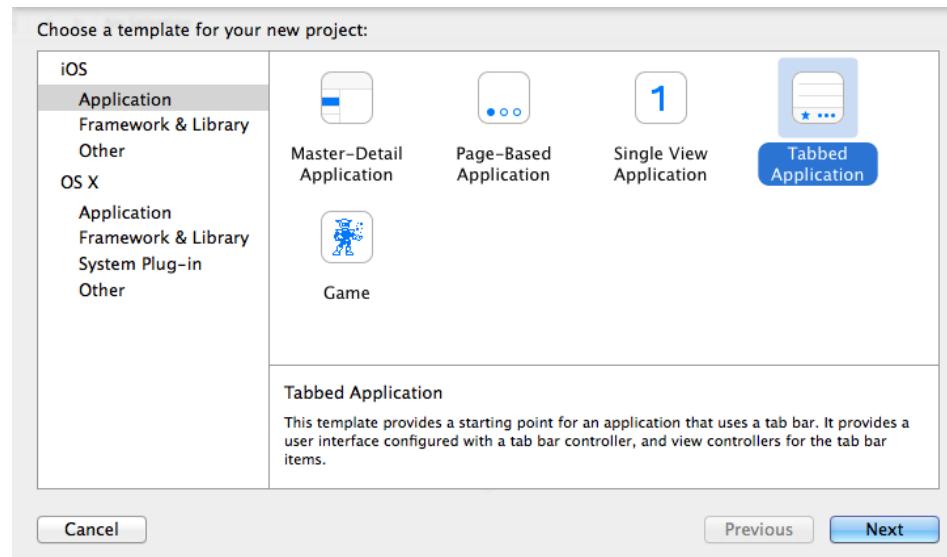
Apesar de parecer simples, aplicações do tipo Single View application ganharam muito espaço atualmente. Este tipo de aplicação apresenta todas as funcionalidades em uma única interface, que tende a resolver todos os problemas do usuário. O exemplo mais conhecido de aplicação web single view application é o gmail.

**Figura 13 – Single View Template.**



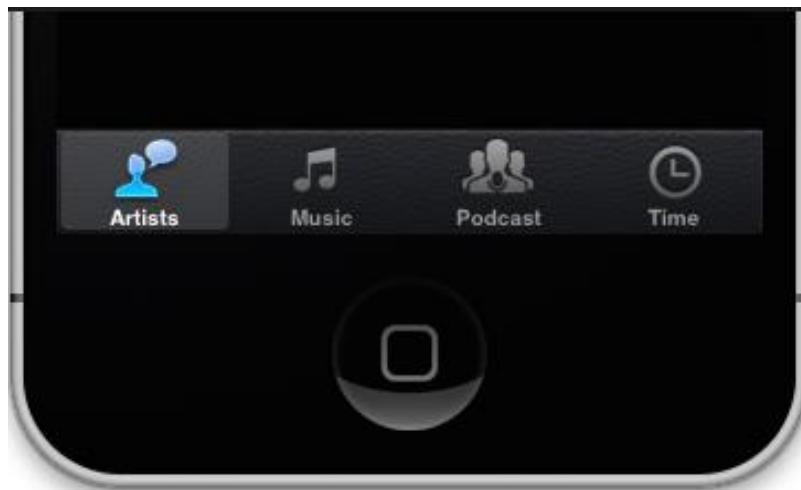
Tabbed Application é mais um tipo de template que apresenta várias views, que não necessariamente estão inter relacionadas, mas que não são navegadas através de gestos swipe, mas apresentando um ícone para cada página existente na aplicação.

**Figura 14 – Tabbed Template.**



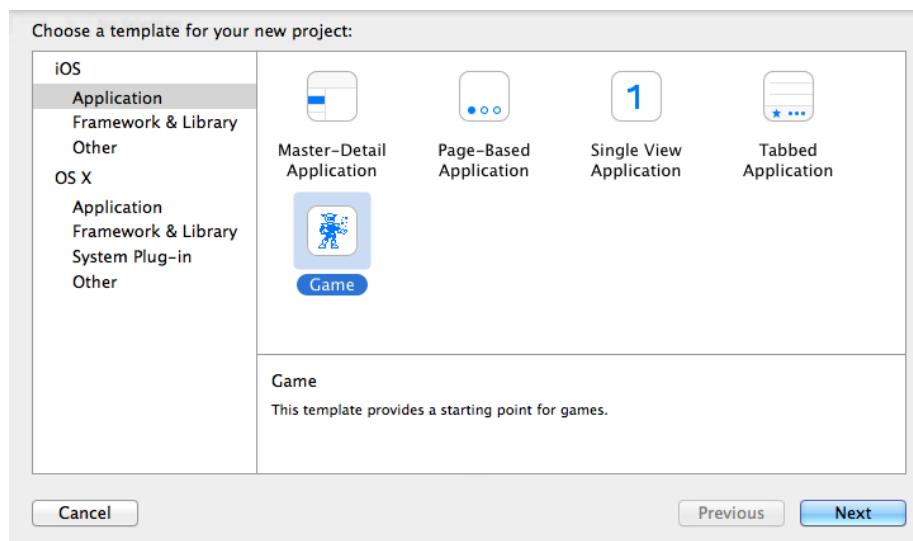
Abaixo o detalhe de como as views são acessíveis por ícones no componente tabbed.

**Figura 15 – App utilizando o componente tabbed.**



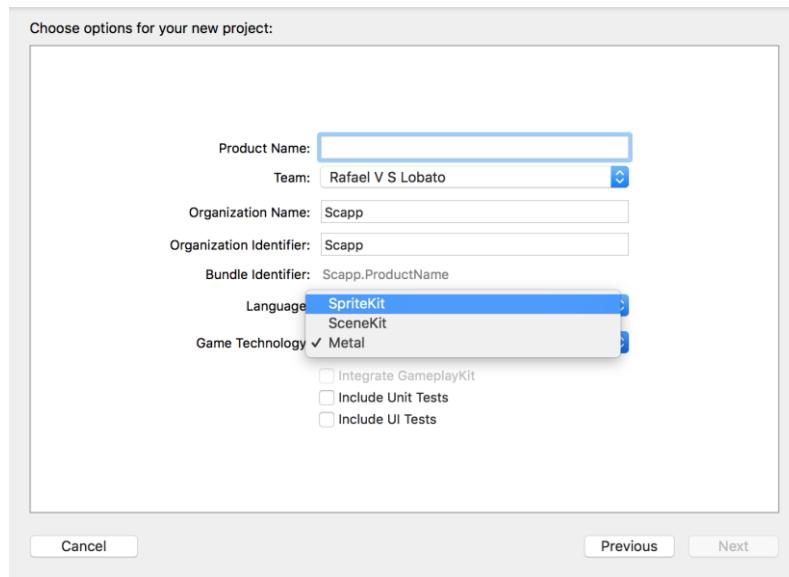
Como o desenvolvimento de jogos é um assunto a parte e envolve uma complexidade maior, na maioria das vezes um dos templates mais completos do XCode é justamente o de jogos. Vem com exemplos prontos, que mostram na prática como trabalhar com som, imagens, gráficos 2D e 3D e física.

**Figura 16 – Template de jogos.**



Possui quatro opções de frameworks predefinidas, apesar de existirem inúmeras no mercado, para cada tipo ou estratégia de desenvolvimento de jogo.

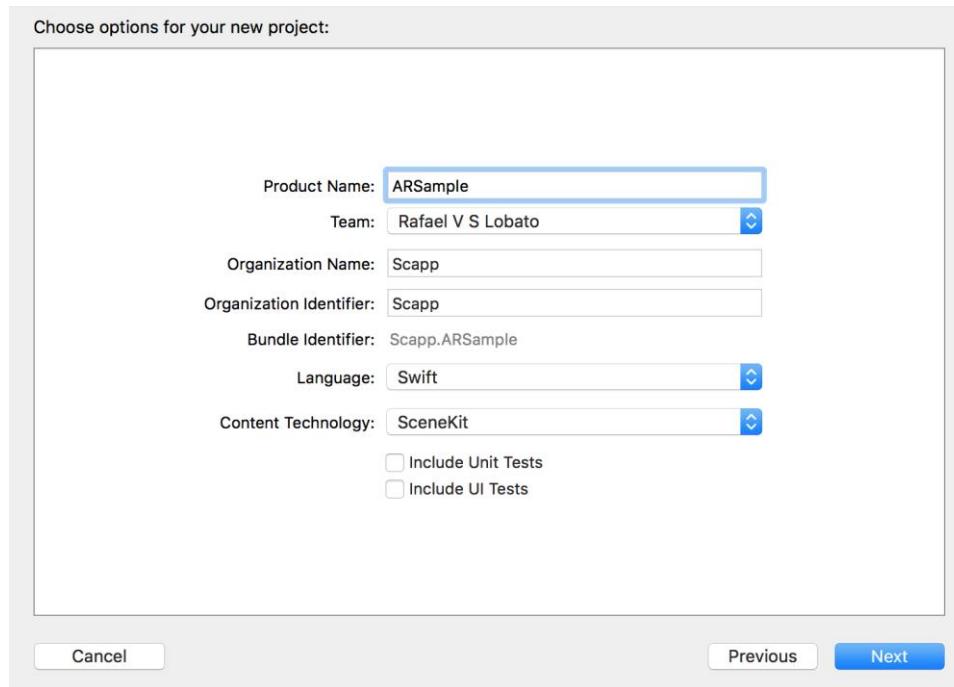
**Figura 17 – Frameworks para desenvolvimento de jogos.**



#### Frameworks para desenvolvimento de jogos:

- SceneKit – Gráficos 3D engine de renderização de alta performance.
- SpriteKit – Usa um loop tradicional de renderização, onde o conteúdo de cada frame é processado antes de ser exibido.
- OpenGL ES – OpenGL for Embedded Systems.
- Metal – Surgiu com o iOS 8, disponível para dispositivos com processador A7 ou superior.

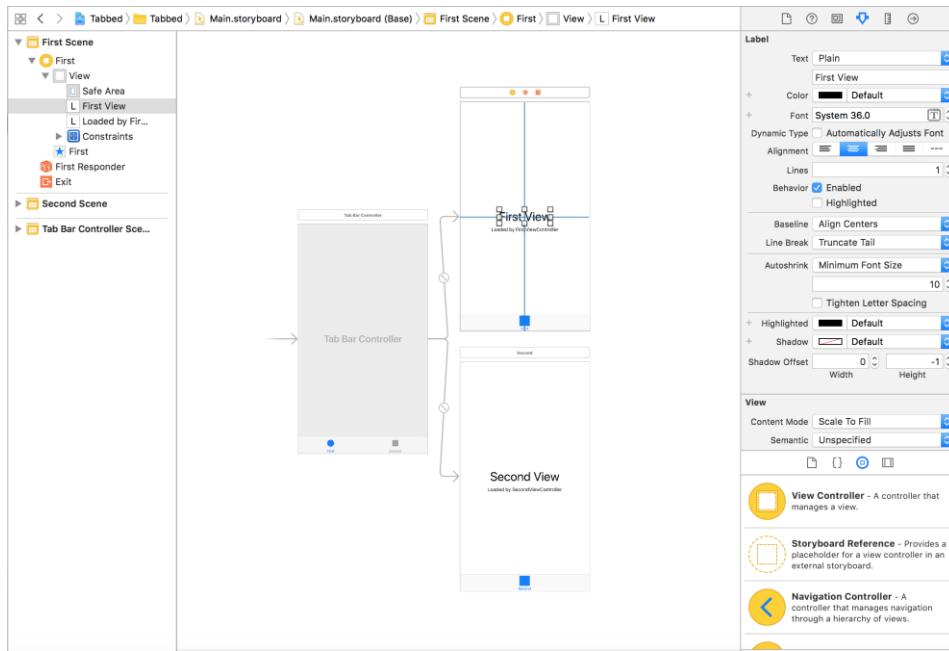
Um último template que vale mencionar no contexto desta disciplina é o template que facilita o desenvolvimento de conteúdo com realidade aumentada.

**Figura 18 – Template para apps com realidade aumentada.**

### 2.3. Interface Builder

O desenvolvimento de interfaces gráficas é um assunto a parte. Chegar no layout ideal seria muito difícil sem o apoio de ferramental específico. No XCode não é diferente. O Interface builder é uma parte da IDE que auxilia na composição de layout e interfaces elaboradas.

Com o Interface builder é possível editar views. Um sistema pode conter várias views que, por sua vez, podem ser desenvolvidas em arquivos XIB ou Storyboard. Além disso, o Interface builder facilita a associação e navegação entre views e controllers, que possuem o código de controle das aplicações.

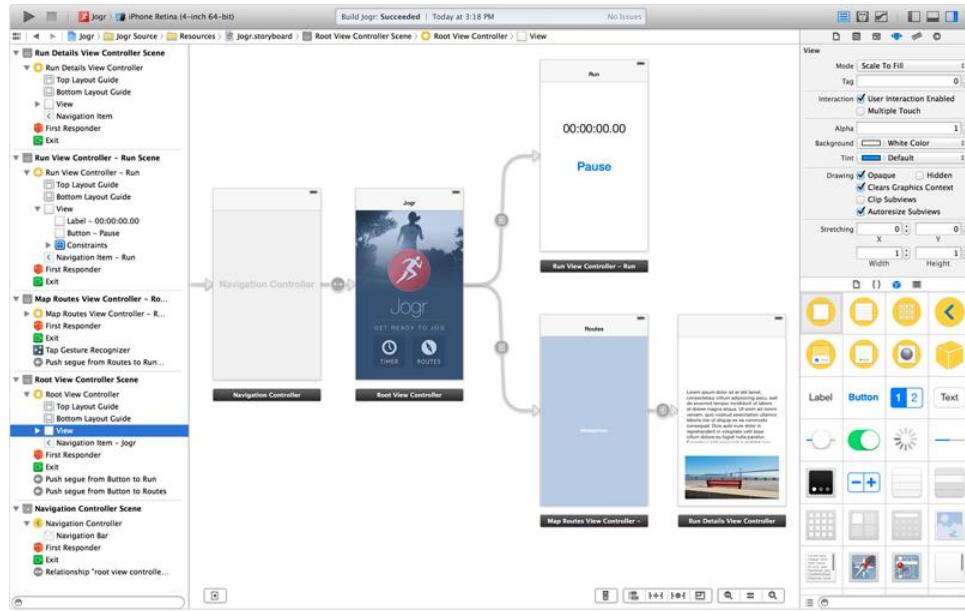
**Figura 19 – Interface Builder.**

## 2.4. Storyboards

Em várias situações precisamos desenvolver aplicações que possuem várias view ou interfaces que são correlacionadas. Da maneira tradicional, utilizando views independentes, podemos ter em um projeto do XCode um número muito elevado de arquivos XIB. Além disso, a complexidade de fazer a transição entre cada uma das views fica toda a cargo do desenvolvedor.

Storyboards são uma solução para este problema, pois permitem desenvolver e visualizar, em um único local, todas as views que participam de um único contexto. O arquivo para este tipo de abordagem tem a extensão storyborad.

Mesmo sendo uma boa alternativa, não deve ser utilizado em todos os casos. É interessante utilizar storyboards, ou até mais de um em uma aplicação, se faz sentido agrupar todas as views em um único contexto. Não é interessante fazer todo o fluxo de uma aplicação em storyboards se não houver uma coerência. Em termos de performance, pode ser interessante ter views separadas se no caso em questão queremos ter um controle maior do que e quando será renderizado.

**Figura 20 – Storyboards.**

## 2.5. Extensões, bibliotecas e frameworks

O principal conceito em qualquer tipo de desenvolvimento é não reinventar a roda, reutilizando sempre que necessário. Para isso agrupamos um conjunto de classes, métodos e funcionalidades em bibliotecas e frameworks que possam ser referenciados em qualquer projeto. A seguir, vamos mostrar alguns exemplos destes frameworks nativos do XCode, que podem ser utilizados em qualquer projeto livremente.

- **Foundation Framework** – Framework inicial com funcionalidades, objetos e coleções básicas:
  - Framework com utilidades básicas.
  - Classe base para todas as classes NSObject.
- **NSString** – Classe utilizada para manipulação de strings:
  - `NSString *testeString = @”Testando...”;`

- Funções:
    - initWithContentsOfFile – Inicializa uma variável string com o conteúdo de um arquivo.
    - initWithContentsOfUrl – Faz uma requisição HTTP a um determinado endereço e inicializa uma string com o conteúdo retornado.
    - rangeOfString – Retorna uma substring a partir de um determinado range.
    - Etc.
  - NSDate:
    - NSDate \*hoje= [NSDate date];
  - NSArray:
    - NSArray \*nomes = @[@"Joao", @"Maria", @"Daniel", @"Felipe"];
  - NSDictionary:
- UIKit:**
- Agrupa a arquitetura necessária para construção e gerenciamento de janelas e views em aplicações iOS.
  - Gerencia eventos para responder a interações com o usuário.
  - Suporte a itens específicos de alguns dispositivos como:
    - Câmera.
    - Photo Library.
    - Sensor de proximidade.
    - Etc.
  - É semelhante ao AppKit para desenvolvimento Mac OS X.

Figura 21 – Lista de frameworks em um projeto.

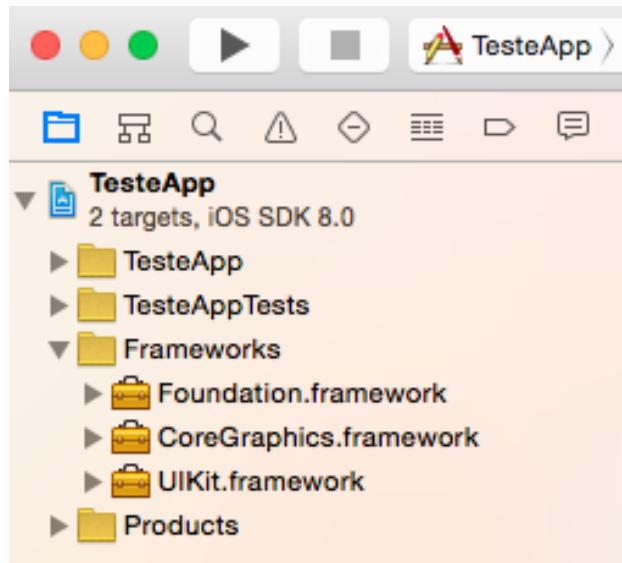
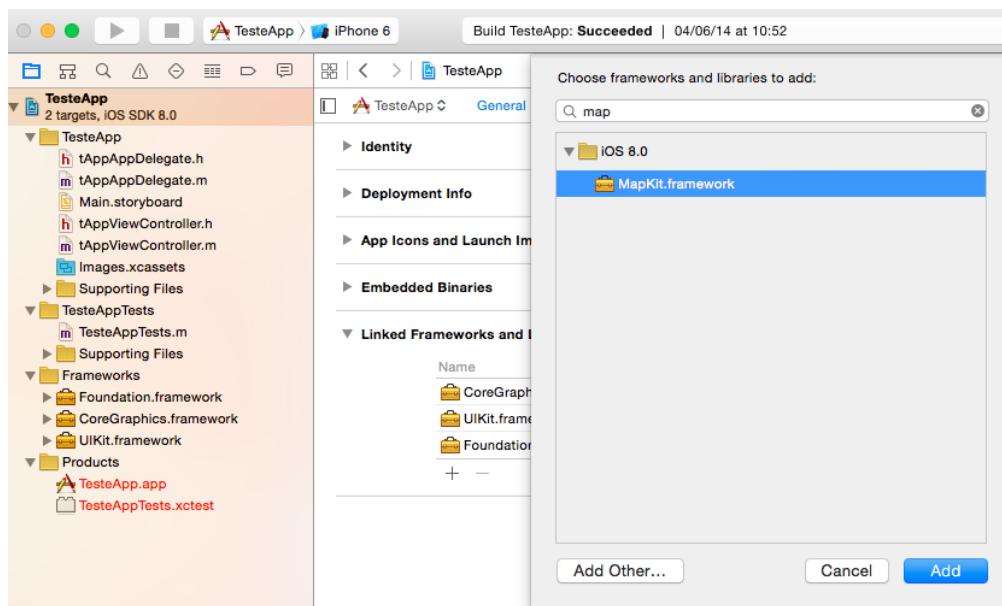
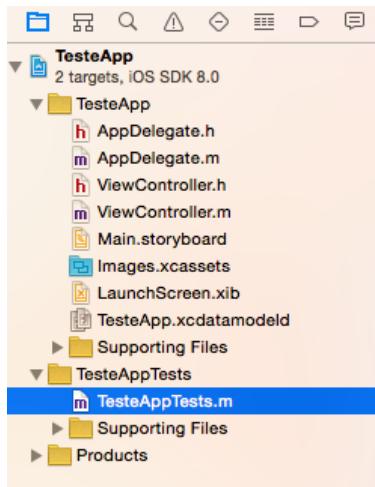


Figura 22 – Adicionando um novo framework.



### XCTest Framework:

- Framework para testes automatizados.
- Adicionado por padrão em novos projetos.
- Classe base para testes: XCTestCase.

**Figura 23 – Código de testes.**

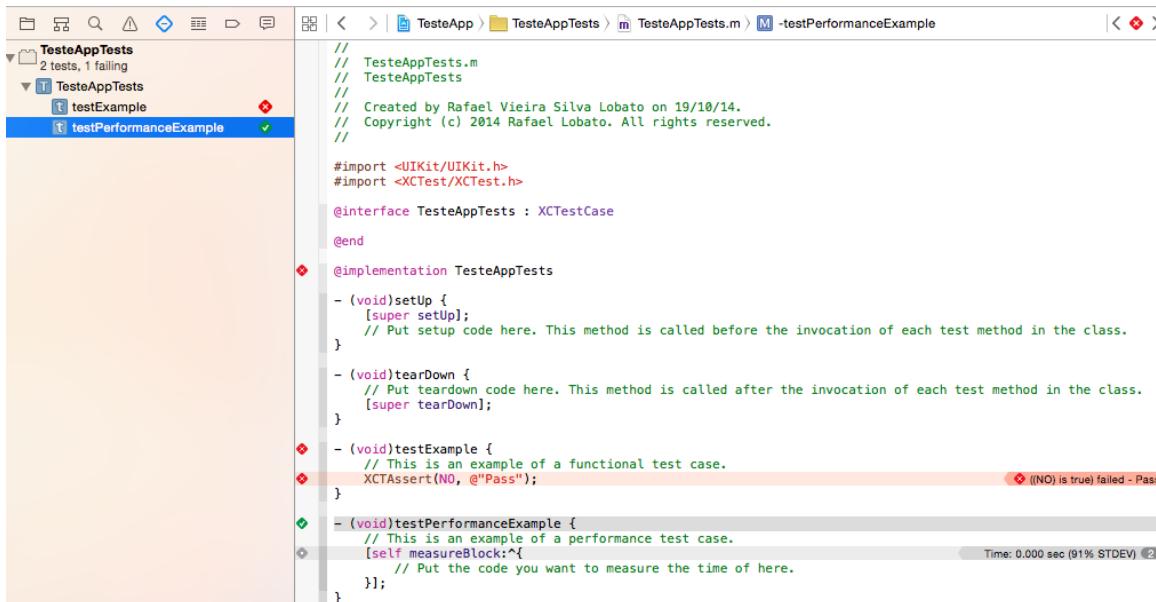
Abaixo o exemplo de teste unitário e de performance criado automaticamente em um novo projeto de testes. Importante notar a herança para a classe XCTestCase e que todos os métodos iniciam com o prefixo test. Além disso, verificar como efetuar as assertivas, com XCTAssert, para que um método passe ou falhe dependendo de um resultado. E como medir o tempo de execução de um bloco de código em testes de performance.

**Figura 24 – Exemplo de teste unitário e de performance.**

```
@interface TesteAppTests : XCTestCase
@end
@implementation TesteAppTests
- (void)setUp {
    [super setUp];
    // Put setup code here. This method is called before the invocation of each test method in the class.
}
- (void)tearDown {
    // Put teardown code here. This method is called after the invocation of each test method in the class.
    [super tearDown];
}
- (void)testExample {
    // This is an example of a functional test case.
    XCTAssert(YES, @"Pass");
}
- (void)testPerformanceExample {
    // This is an example of a performance test case.
    [self measureBlock:^{
        // Put the code you want to measure the time of here.
    }];
}
```

Na execução dos testes, ou da chamada suíte de testes, detalhe para o painel que indica todos os testes presentes e o resultado de cada um.

**Figura 25 – Detalhe da execução de testes.**



```

// TesteAppTests.m
// TesteAppTests
//
// Created by Rafael Vieira Silva Lobato on 19/10/14.
// Copyright (c) 2014 Rafael Lobato. All rights reserved.
//

#import <UIKit/UIKit.h>
#import <XCTest/XCTest.h>

@interface TesteAppTests : XCTestCase
@end

@implementation TesteAppTests
- (void)setUp {
    [super setUp];
    // Put setup code here. This method is called before the invocation of each test method in the class.
}

- (void)tearDown {
    // Put teardown code here. This method is called after the invocation of each test method in the class.
    [super tearDown];
}

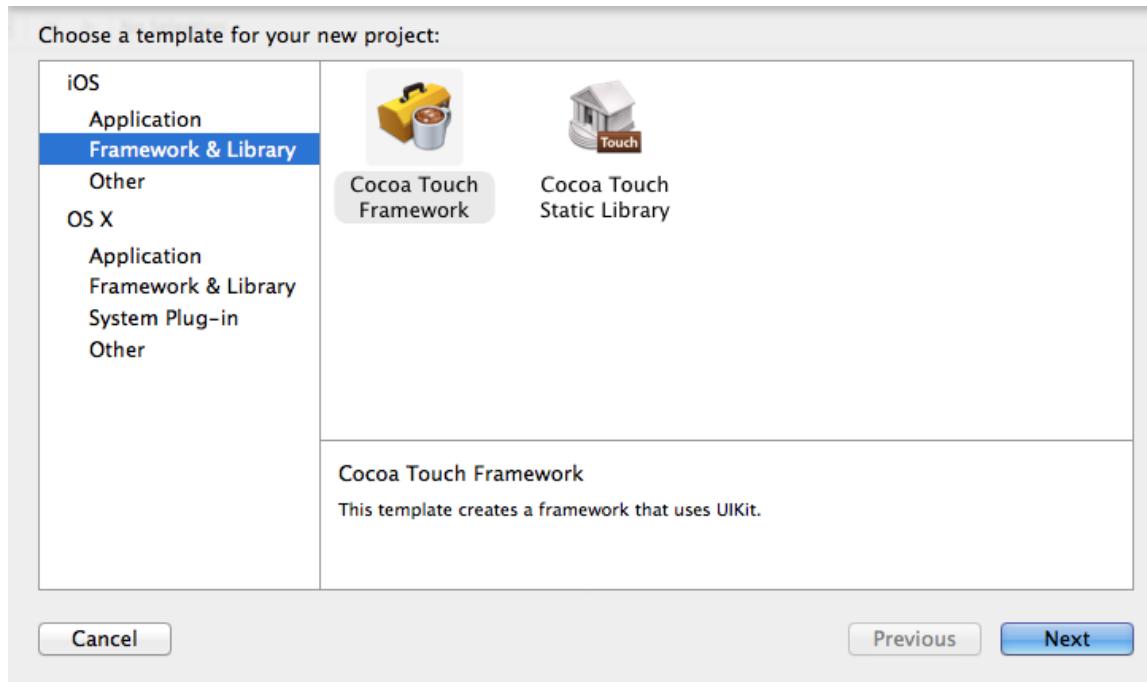
- (void)testExample {
    // This is an example of a functional test case.
    XCTAssertNil(nil, @"Pass");
    //((NO) is true) failed - Pass
}

- (void)testPerformanceExample {
    // This is an example of a performance test case.
    [self measureBlock:^{
        // Put the code you want to measure the time of here.
    }];
}

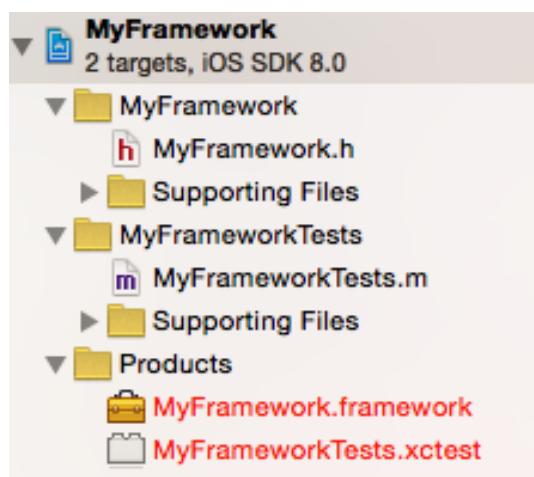
```

- Outros frameworks:
  - MapKit.
  - CoreData.
  - CoreLocation.
  - iAd.
  - NotificationCenter.
  - QuartzCore.
  - WebKit.
  - Etc.

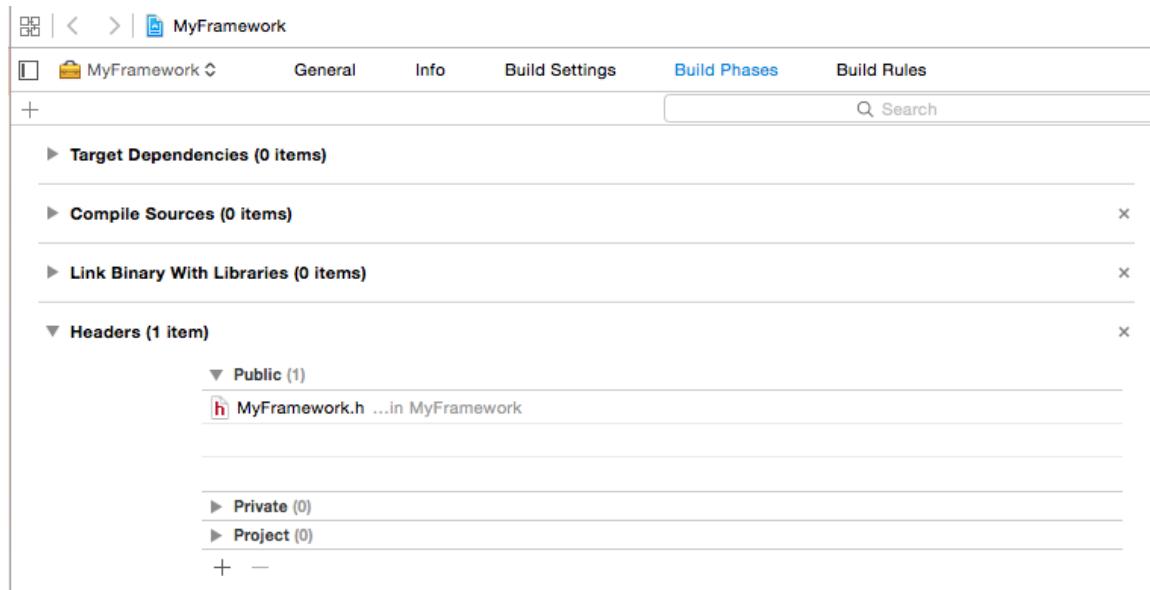
Além destes frameworks existentes, é possível criar novos utilizando o template abaixo:

**Figura 26 – Template de frameworks.**

Toda funcionalidade genérica e reutilizável deve ser encapsulada em um framework que facilita a sua distribuição e utilização em outros projetos.

**Figura 27 – Detalhe do Framework compilado.**

Em um framework é importante deixar explícito quais funcionalidades vão ser expostas ou não para o consumidor. Para isso é necessário indicar todos os arquivos de header que vão ser públicos através da interface abaixo, presente nas configurações do projeto:

**Figura 28 – Interfaces públicas de um framework.**

## Capítulo 3. Linguagens de Programação

---

Por muito tempo o desenvolvimento para iOS e Mac OS X era todo feito utilizando a linguagem Objective-C. A partir do final de 2014, uma nova alternativa foi apresentada: a linguagem Swift, trazendo conceitos mais modernos de linguagem de programação, uma vez que o objective-C é uma linguagem com muito tempo de mercado.

### 3.1. Objective-C

---

O Objective-C é uma linguagem que complementa a linguagem C e é orientada a objetos. Como dito anteriormente, é utilizada no desenvolvimento de aplicações OS X e iOS. Tem bastante tempo de mercado sendo originada dos anos 80.

Para quem não é familiarizado com a linguagem C, é necessário antes conhecer alguns conceitos de organização e extensões de arquivos:

- Extensões C:
  - .h
  - .c
- Extensões C++
  - .h
  - .cpp
- Extensões Objective C
  - .h
  - .m

Nas três linguagens as assinaturas dos métodos ficam separados em arquivos de header, .h.

Abaixo as principais características de sintaxe da linguagem:

- Definindo classes – Interface Section:

*@interface Fracao: NSObject*

{

*int numerador;*

*int denominador;*

}

*-(void) print;*

*@end*

- Definindo classes – Implementation Section:

*@implementation Fracao*

*-(void) print*

{

*NSLog (@”%i/%i”, numerador, denominador);*

}

*@end*

- Envio de mensagens / Chamadas a métodos é um conceito bem diferente da maioria das linguagens. Como chamadas a métodos são realizadas:

*java/c#/c++*

*obj.metodo(argumentos);*

### Objective C

[*obj metodo:argumentos*];

- Métodos com múltiplos parâmetros:

Java/C#/C++

*Definição:*

*void InserirObjetoEm(Object obj, Int indice){}*

*Utilização:*

*obj.InserirObjetoEm(obj, 2);*

- Métodos com múltiplos parâmetros:

Objective C

*Definição:*

- (*void*) *InserirObjeto: (NSObject)obj Em: (int) indice {}*

*Utilização:*

[*obj InserirObjeto:obj Em:2*];

```
#import <Foundation/Foundation.h>
```

```
int main (int argc, const char *argv[]) {
```

```
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
```

```
    NSLog (@"Testing... \n..1\n..2\n....3");
```

```
[pool drain];
```

```
    return 0;  
}
```

- Utilizando classe Fracao:

```
Fracao *f;  
  
f = [[Fracao alloc] init];  
  
[f print];
```

- Condicionais e Loops seguindo o mesmo formato da linguagem C, também semelhante a Java a C#:
  - If Else
  - For
  - While
  - Do while
  - Etc.
- O mesmo ocorre para comparações e expressões aritméticas:
  - ==
  - >=
  - <=
  - !=
  - &&
  - ||
- Herança – Sintaxe semelhante a linguagem C#:
  - @interface ClasseB : ClasseA  
Classe B herdando Classe A.

- Todos os conceitos de orientação a objeto se aplicam da mesma forma em Objective C:
  - Atributos herdados de classes pai.
  - Polimorfismo.
  - Referenciar classes filhas com referências de classes base.
  - Etc.

### 3.2. Swift

---

A linguagem Swift foi introduzida no WWDC 2014, conferência mundial de desenvolvedores, e está disponível para utilização no XCode 6. Para novos projetos é necessário escolher entre as duas linguagens, porém é compatível com Objective C, ou seja, frameworks desenvolvidos podem ser acessados independente da linguagem desenvolvida. É possível ainda que um mesmo programa contenha C, C++, Objective-C e Swift.

O material de estudo mais indicado é o guia de referência oficial da Apple, disponível de graça na App Store.

**Figura 29 – Livro de programação na linguagem Swift.**



O principal objetivo de Swift: “Objective C sem o C”, remover a complexidade e a insegurança do desenvolvimento de aplicações. Diminui overheads como criação e alocação de ponteiros. Utilizou características positivas das linguagens Ruby, Python, C#, dentre outras.

Diferença ao se manipular strings:

```
NSMutableString *str = @”hello, “;
```

```
str = [str stringByAppendingString: @”world”];
```

```
var str = “hello, “;
```

```
str += “hello”;
```

Características em comum com o Objective C:

- Tipo numéricos.
- Operadores.
- Chaves para agrupar código.
- Associar variáveis e valor com = e comparar com ==.
- Loops e condicionais.
- self (this do java e c#).

Diferenças para o Objective C:

- Arquivos header ou .h.
- Sobrescrita de operadores.
- Ponteiros explícitos.
- Associações não retornam valor, evitando:
  - if(a=0) => causa erro de compilação (if(a==0) é o correto)
- Programação genérica.
- Etc.

ARC – Contagem automática de referências e simula uma coleta automática de lixo (memória não referenciada), apesar de não ser na prática coleta de lixo, como existe em .NET e Java:

- Automatic Reference Counting.
- Melhorada para o Swift.
- Diminui o custo de gerenciamento de memória.
- Não existe Garbage Collection para iOS.
- Elimina a necessidade de alocar e desalocar explicitamente variáveis.

Exemplo de classe em Swift:

```
class Counter {  
    var count = 0  
    func increment() {  
        count++  
    }  
    func incrementBy(amount: Int) {  
        count += amount  
    }  
    func reset() {  
        count = 0  
    }  
}
```

Quando usar Objective C?

- Se já você ou sua equipe são experts em Objective C.
- Se vai dar manutenção em sistema Objective C.

Quando usar Swift?

- Se é novo em desenvolvimento para iOS.
- Se tem condições de se capacitar e migrar para esta nova linguagem.

## Capítulo 4. Ciclo de vida de desenvolvimento na plataforma

---

O ciclo de vida de desenvolvimento de software pode ter várias etapas. Estas etapas podem variar ainda, dependendo do processo de desenvolvimento de software adotado. Neste capítulo, vamos falar de quatro fases que estão presentes na maioria dos processos.

### 4.1. Desenvolvimento

---

Para configurar um ambiente de desenvolvimento iOS é necessário obrigatoriamente de um computador Mac executando OS X, XCode e do iOS SDK (Software development kit). Existem maneiras alternativas em máquina virtuais, mas para distribuir a aplicação de maneira profissional e formal na App Store, somente com hardware Apple.

- Processo básico de desenvolvimento na plataforma:
  - Utilizar o XCode para criar e gerenciar projetos.
  - Identificar os pontos chave do projeto.
  - Criar uma interface com o usuário.
  - Codificar.
  - Compilar.
  - Executar.
  - Depurar.
  - Testes de desenvolvimento.

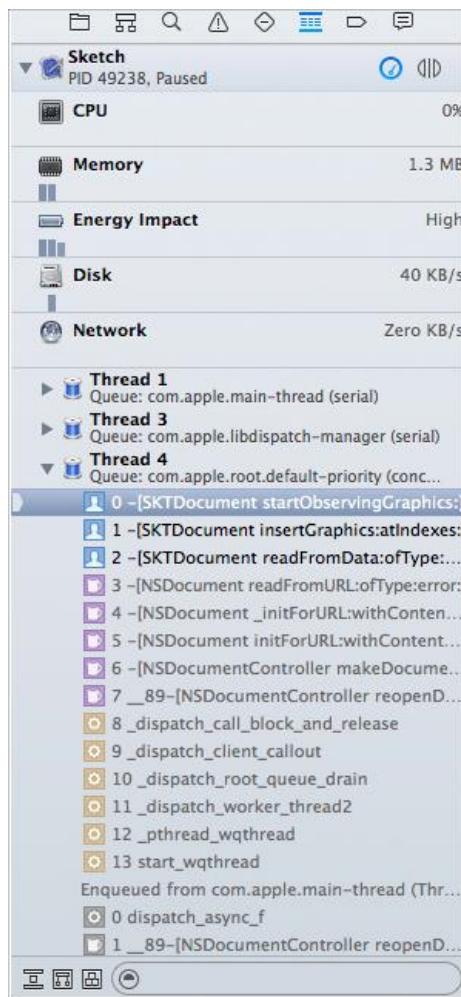
Tendo em mente estes passos no processo básico de desenvolvimento na plataforma, é interessante conhecer o que o XCode nos oferece para otimizar cada um destes passos. O fato é que manualmente quase nada mais precisa ser feito.

- Debug Navigator – Auxilia na parte de depuração de código:
  - Aberto automaticamente quando um breakpoint é atingido ou quando clicamos em pause durante a execução de um app.

- Visão de consumo de recursos durante a execução do programa.
- Detalhe de consumo de cada recurso.
- Threads, ou processos paralelos, em execução.
- Trace de execução, ou seja, pilha de chamadas de métodos.

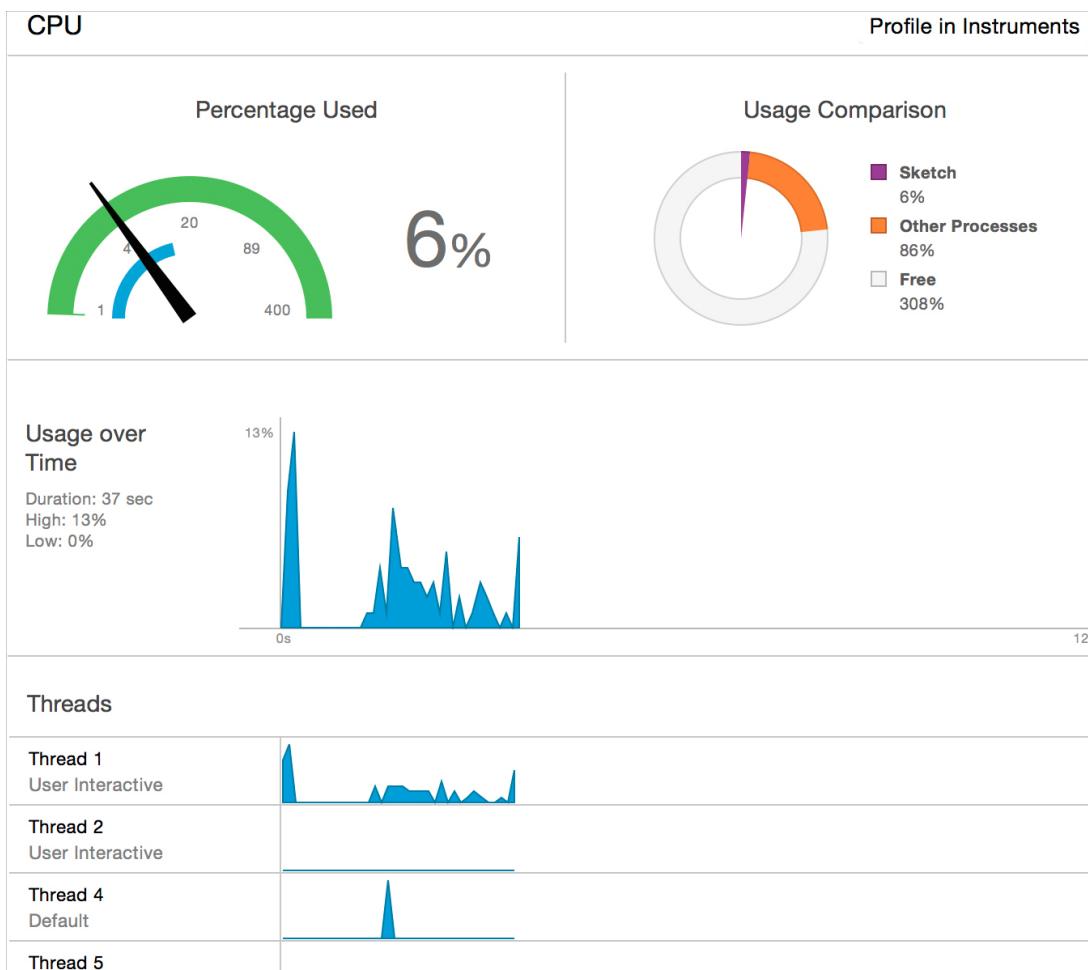
Abaixo o detalhe de como é o debug navigator. Como visualizar cada recurso consumido e a pilha de chamadas para cada processo paralelo em execução:

**Figura 30 – Debug navigator.**

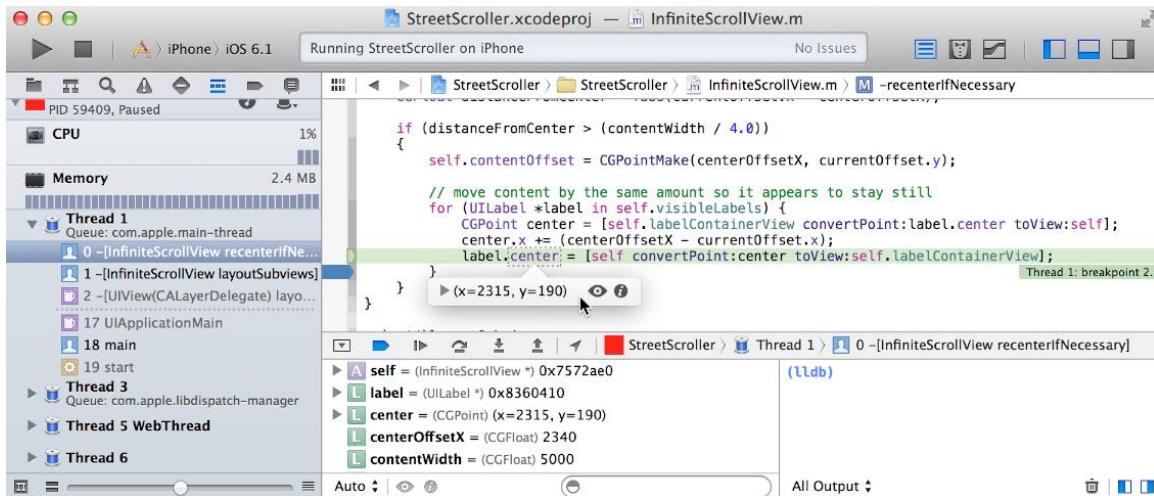


Cada um dos pontos exibidos acima no debug navigator, apresenta detalhes que podem ser acessados com informações mais precisas sobre cada consumo.

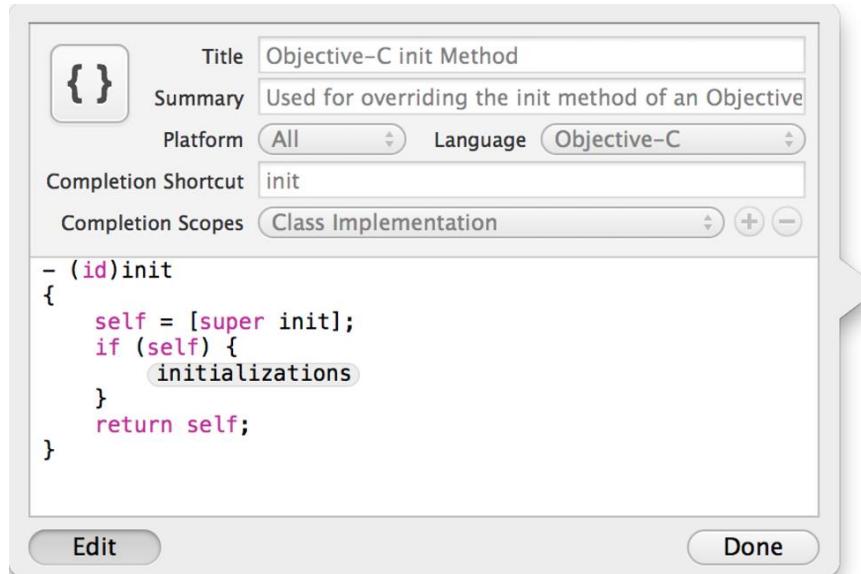
**Figura 31 – Detalhe de cada item do debug navigator.**



Uma das ferramentas mais utilizadas é o property inspector. Ele permite que durante a depuração de um programa, seja verificado em tempo de execução os valores de qualquer variável do código. Estes valores podem ser inclusive alterados manualmente para simular comportamentos diferentes, sem precisar reiniciar a execução.

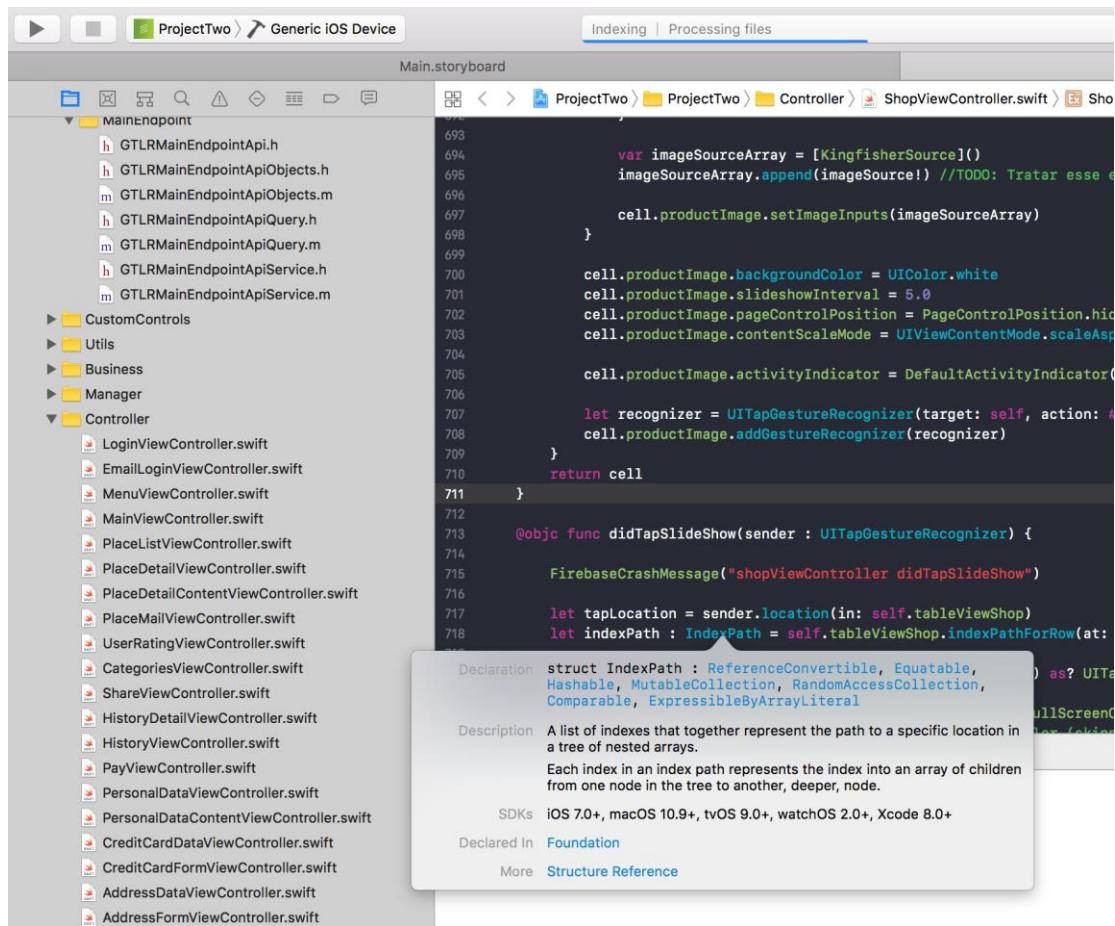
**Figura 32 – Property inspector.**

Visando agora a produtividade, temos um outro conceito também presente na maioria das IDEs, que são os code snippets. Podem ser acessados no menu Code Snippets (View -> Utilities -> Code Snippets Library). Todo trecho de código que se repete muito, como criação de variáveis, loops, etc. podem ser convertidos em code snippets. A partir daí, ao iniciar a digitação do código, ele se autocompleta.

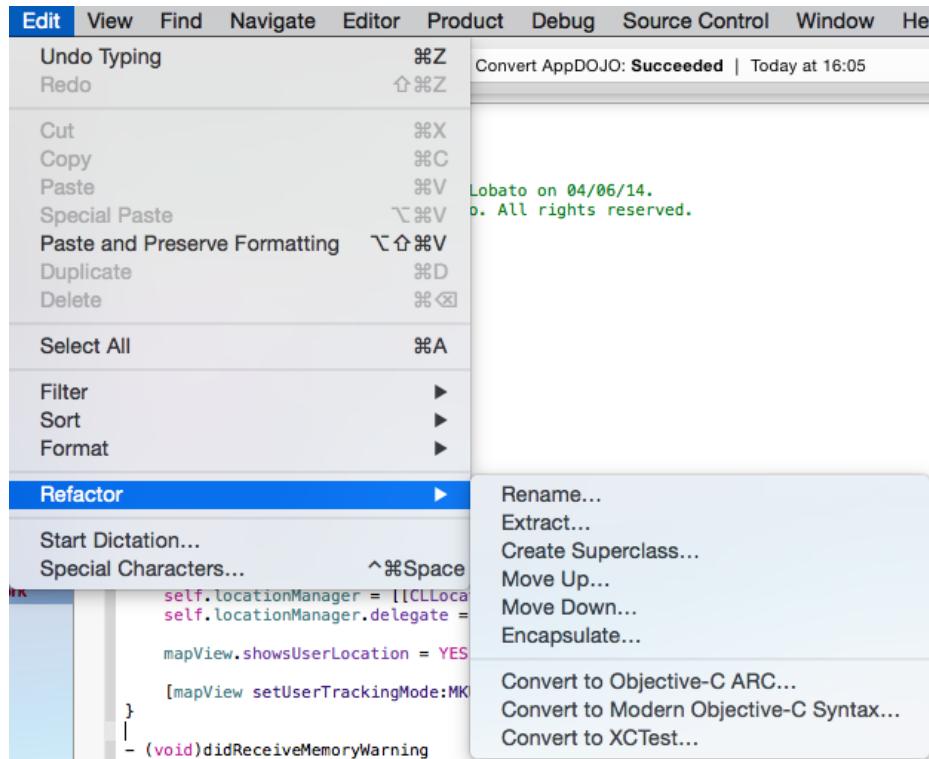
**Figura 33 – Edição de code snipets.**

Ainda falando de produtividade, existe um help de acesso rápido dentro do próprio XCode. Mostra a descrição de objetos e métodos possíveis, dentre outras informações.

**Figura 34 – Quick help.**

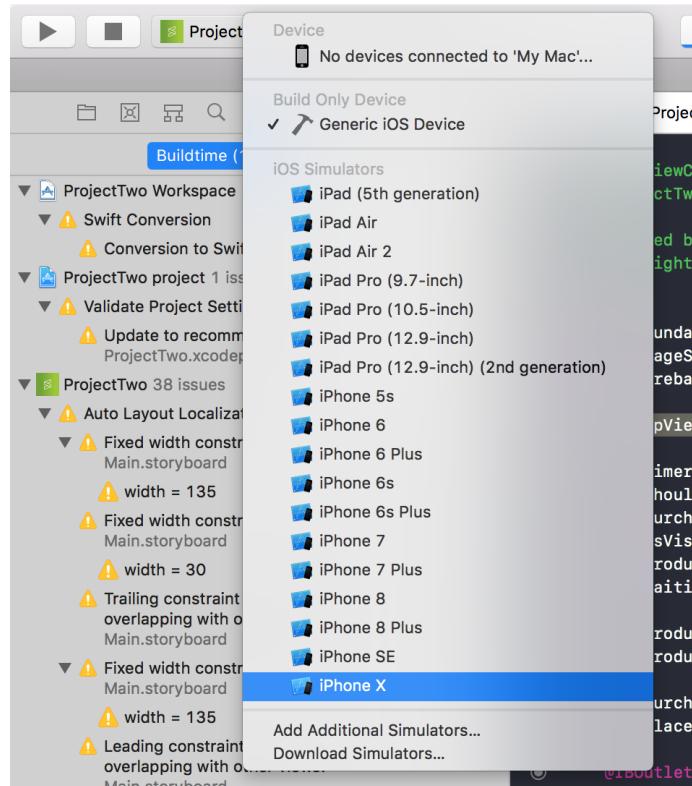


Refactoring é o ato de se reorganizar ou reestruturar um código já pronto. Muitas vezes, desenvolvemos rapidamente visando entregas, e posteriormente identificamos códigos duplicados ou funções que poderiam ser reaproveitadas, por exemplo. Como o refactoring é um esforço previsto na maioria dos projetos, o XCode também fornece ferramental para esta prática.

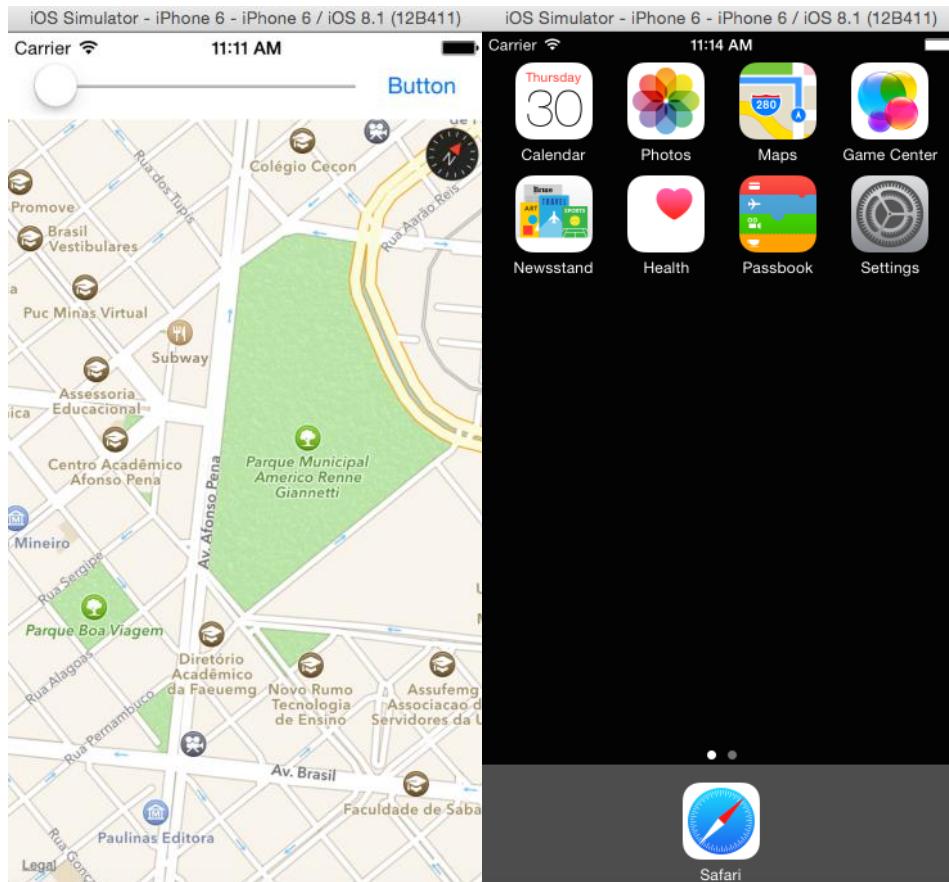
**Figura 35 – Opções de refactoring.**

É possível renomear uma variável e automaticamente alterar todos os pontos que ela é utilizada. Transformar um trecho de código em método, encapsular propriedades, dentre outras funcionalidades.

- Static Code Analysis (Product -> Analyse) – Análise estática de código.  
Praticamente uma revisão do código de maneira automatizada e inteligente.
  - Identifica falhas de lógica como:
    - Acessar variáveis não inicializadas.
    - Desreferenciar ponteiros nulos.
  - Falhas de gerenciamento de memória.
  - Variáveis não utilizadas.
  - Falha na utilização de APIs.
- iOS Simulator – Simulador iOS que simula diferentes tipos de dispositivo que utilizam iOS.

**Figura 36 – Simuladores iOS.**

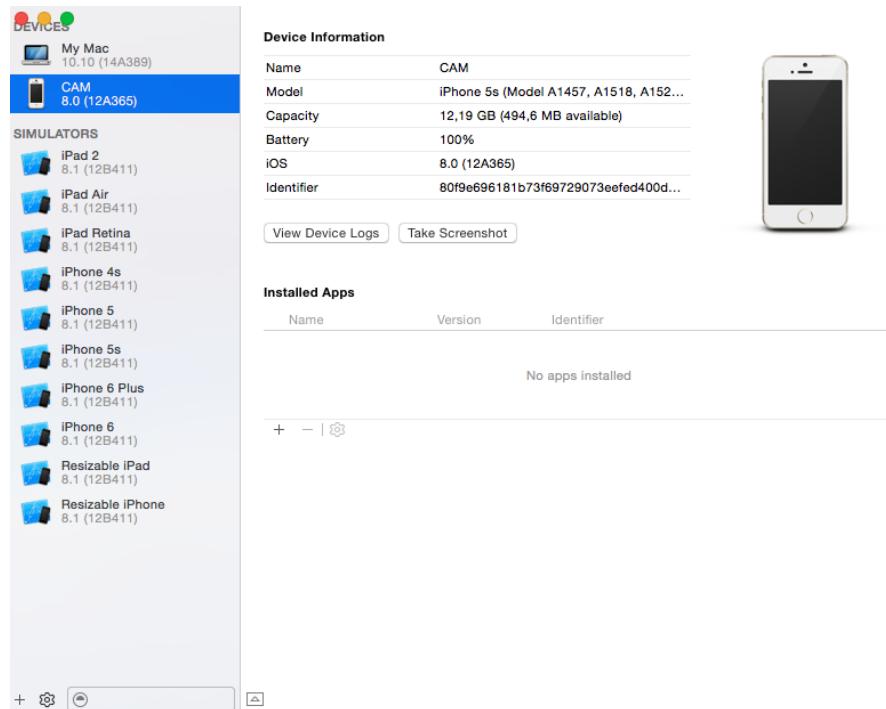
- Funcionalidades principais:
  - Simula vários devices iOS.
  - Simula várias versões iOS.
  - Locations.
  - Displays Externos.
  - Memory Warnings.
  - Pode ser usado para testar Web Apps pelo Safari.

**Figura 37 – Simulador iPhone 6.**

- iOS Simulator - Restrições:
  - Não possui Câmera.
  - Não possui Microfone.
  - Não possui sensor proximidade.
  - Não possui Acelerômetro e Giroscópio.
  - Nem todos os bugs de performance podem ser testados no simulador.
  - Deve ser utilizado para testes primários e nunca para testes de release.
  - OpenGL ES Performance – A performance de aplicações Open GL pode não ser a mesma de dispositivos reais.

A principal e mais confiável forma de se validar aplicações é utilizando dispositivos reais. Os mesmos são gerenciados praticamente da mesma forma que simuladores conforme a ilustração abaixo:

**Figura 38 – Lista de dispositivos.**



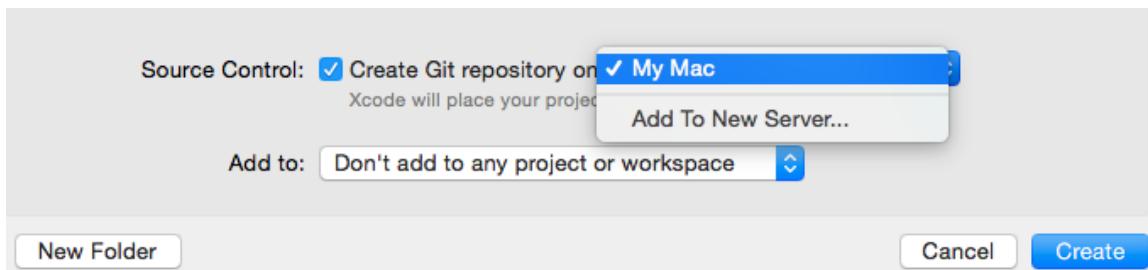
Cada aplicação em um dispositivo possui um container próprio, onde todos os arquivos e dados locais ficam armazenados. Com o gerenciador de dispositivos é possível:

- Gerenciar containers:
  - Visualizar container ajuda na investigação de problemas.
  - Permite analisar o conteúdo de arquivos gerados.
  - Substituir arquivos permite definir um estado particular para a aplicação.
- Outras funcionalidades:
  - Screenshots.
  - Instalar versões beta do iOS.

## 4.2. Versionamento

Versionamento de código é algo indispensável em desenvolvimento em equipe e em projetos de mercado. O XCode oferece integração padrão com o Git, Source Control Open Source e Free. Deve ser utilizado mesmo em desenvolvimento independente.

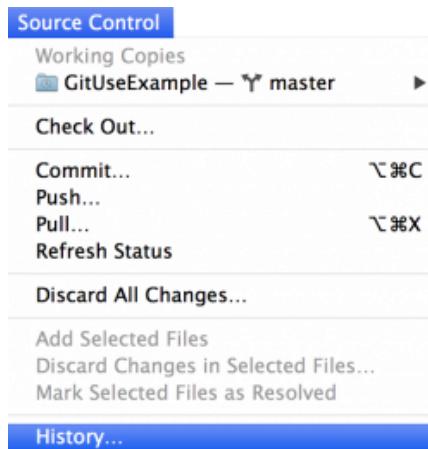
**Figura 39 – Criação repositório GIT.**



Mesmo que não se possua um servidor estruturado para esta finalidade, é possível criar um repositório local em sistema de arquivos como exibido na imagem acima.

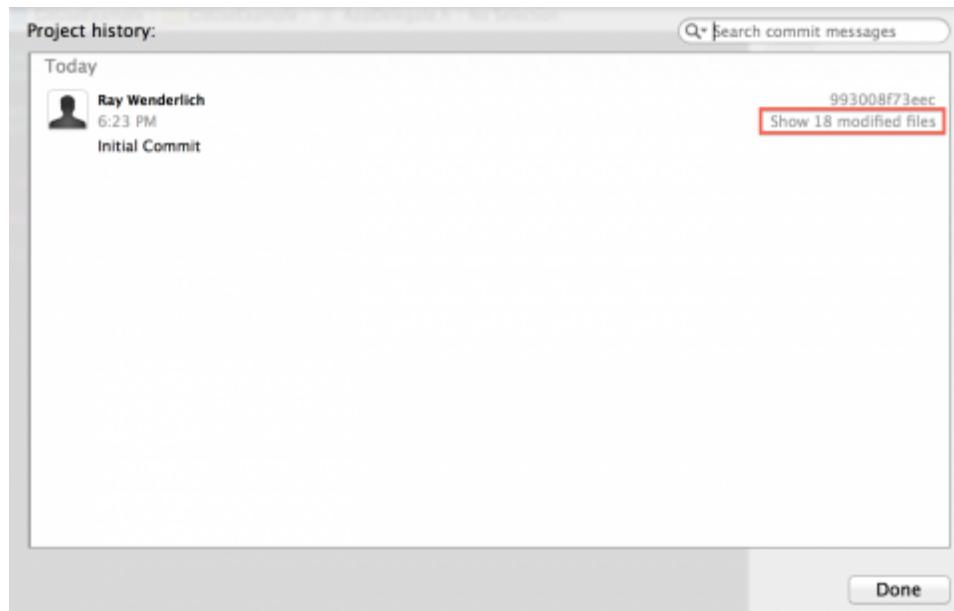
Assim que um projeto é adicionado ao controle do GIT, é possível executar as ações abaixo sobre o mesmo:

**Figura 40 – Opções do gerenciador de fontes.**



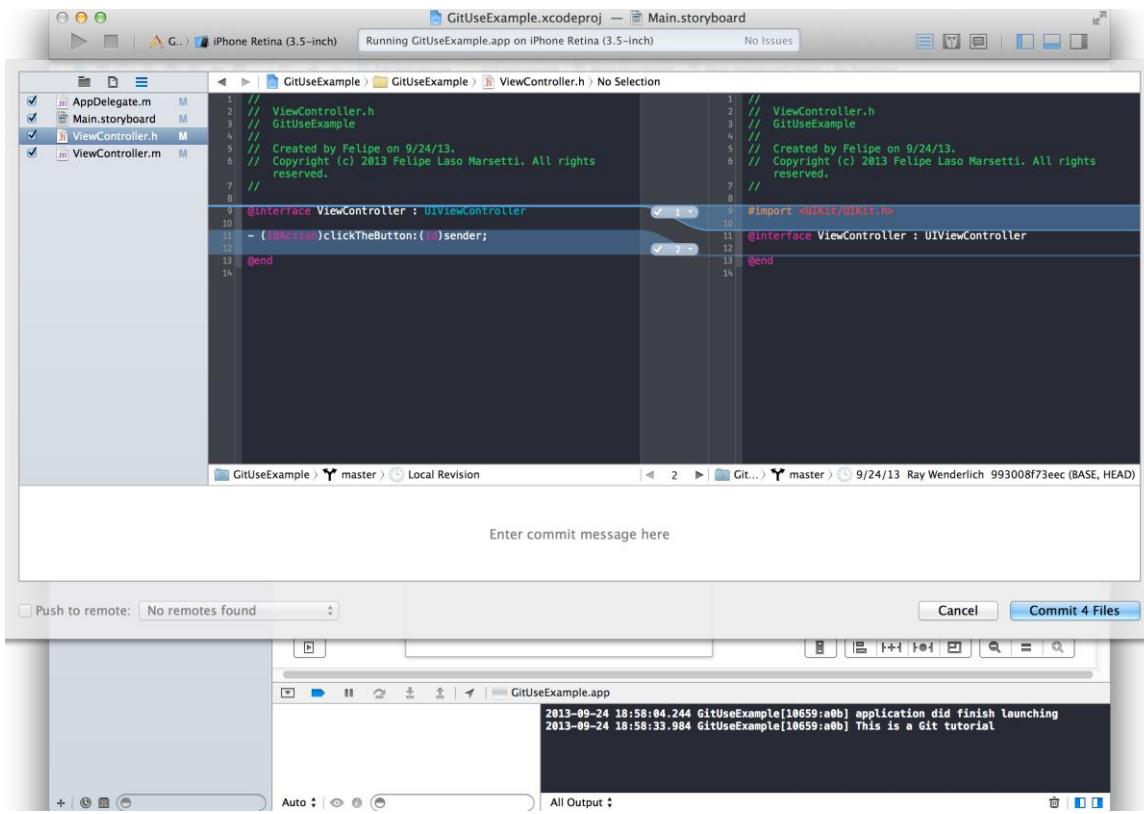
Falando especificamente da opção de histórico é possível não só ver quem efetuou uma determinada alteração, seja a nível de arquivo ou a nível de projeto, e ainda o que realmente foi alterado. Existe uma ferramenta para evidenciar as alterações e que também é utilizada para fazer merge de alterações com o código já existente no gerenciador de fontes.

**Figura 41 – Histórico de alterações.**

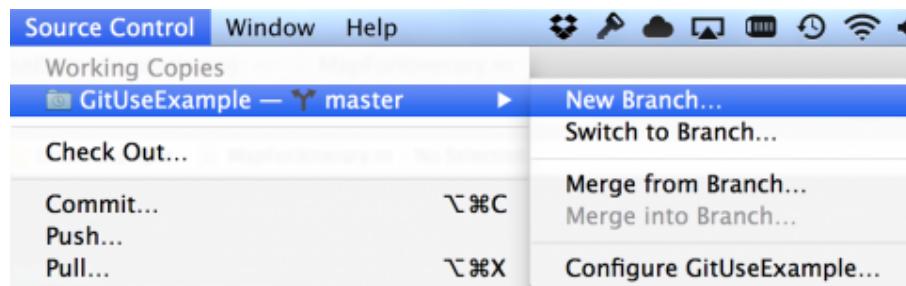


Ferramenta de merge de alterações.

**Figura 42 – Commit e alterações.**



Criação de Branchs para desenvolvimento paralelo. Funcionalidade fundamental para desenvolvimento paralelo de várias alterações independentes, que têm datas de subida para ambiente de produção distintas.



Alternativas existentes ao gerenciamento de fontes e versionamento com Git:

- Team Foundation Server – Microsoft.
- Repositório local do Git no TFS usando Git-TF.
- Permite utilizar o TFS como ALM.

- Acesso total às funcionalidades, como histórico.
- Gittf.codeplex.com.

#### 4.3. Testes

Quando o assunto são testes de uma aplicação, sempre temos que ter em mente criar um conjunto de testes unitários que tenha a maior cobertura de código possível dentro do projeto. Para isso, existe um tipo de código específico que implementam este conceito de testes. O conjunto de testes é chamado de suíte de testes que pode ser executada em vários contextos. Diretamente pelos desenvolvedores, após cada atualização de código, junto do processo de build, em um contexto de integração contínua, dentre outros. A suite de testes na plataforma geralmente utiliza o XCTest Framework.

**Figura 43 – Execução suíte de testes.**

```
XCTAssertEqualObjects([calcViewController.displayField stringValue], @"10",
                     @"Part 2 failed.");
}
/* testSubtraction performs a simple subtraction test.
 * Check: 6 - 2 = 4.
 */
- (void) testSubtraction {
    [calcViewController press:[calcView viewWithTag: 6]]; // 6
    [calcViewController press:[calcView viewWithTag:14]]; // -
    [calcViewController press:[calcView viewWithTag: 1]]; // 2
    [calcViewController press:[calcView viewWithTag:12]]; // =
    XCTAssertEqualObjects([calcViewController.displayField stringValue], @"4", @"");
}
/* testDivision performs a simple division test.
 * Check: 25 / 4 = 6.25.
 */
- (void) testDivision {
    [calcViewController press:[calcView viewWithTag: 2]]; // 2
    [calcViewController press:[calcView viewWithTag: 5]]; // 5
}
```

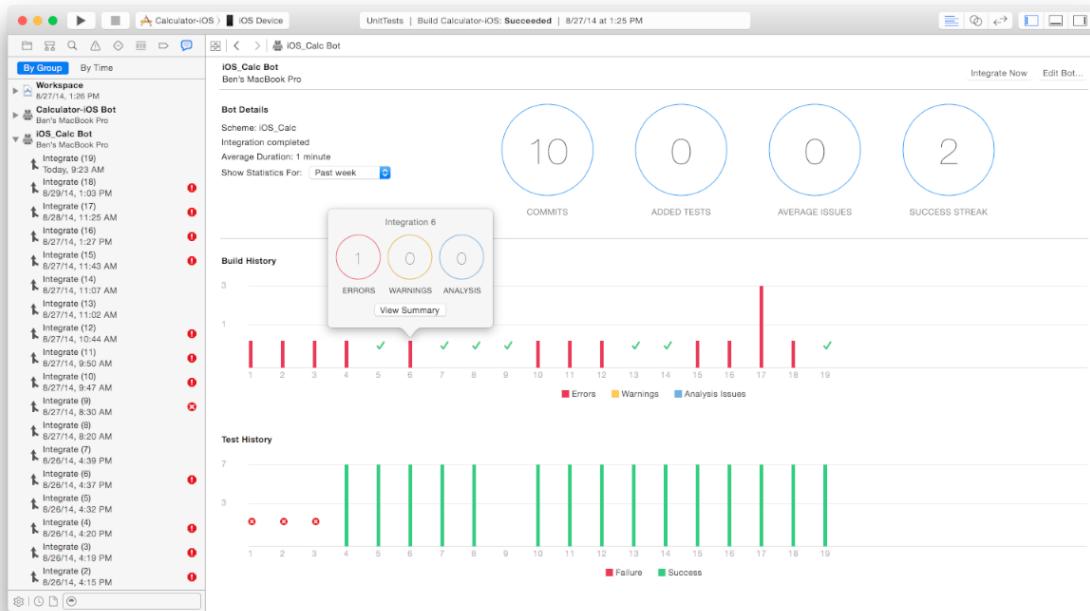
O XCode também tem suporte a integração contínua, porém precisamos de um novo item que é o XCode Server.

Principais benefícios:

- Ajuda a encontrar problemas rapidamente e mais facilmente.
- Melhora a distribuição de atividades e a colaboração entre o time.
- O Xcode server automatiza o processo de compilação, análise, testes e arquivamento das aplicações.

- Objetivo principal: ter sempre versões funcionais e estáveis após qualquer alteração
- Pode ser agendado, executado após commits ou manualmente.

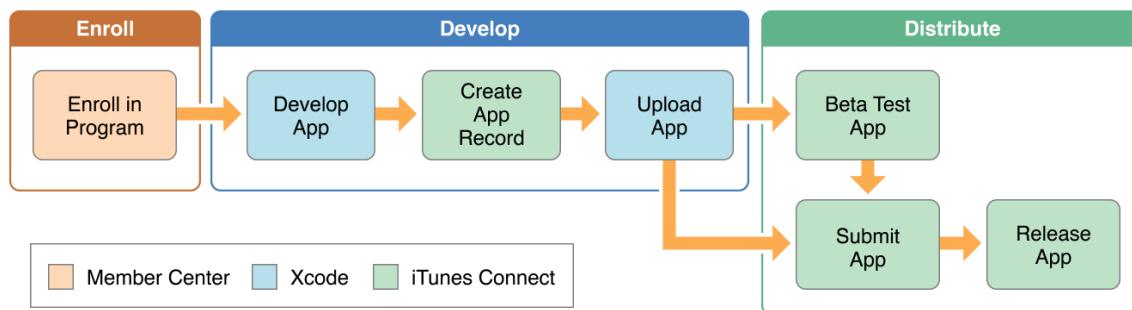
**Figura 44 – Relatórios acompanhamento integração contínua.**



#### 4.4. Deployment

Todas as fases de desenvolvimento anteriores visão mitigar problemas para a última fase, que é a de deployment, ou seja, publicar um sistema em produção. No caso de aplicativos iOS, geralmente publicar em produção significa subir um app para a App Store para que esteja disponível para download para todos os usuários de iOS.

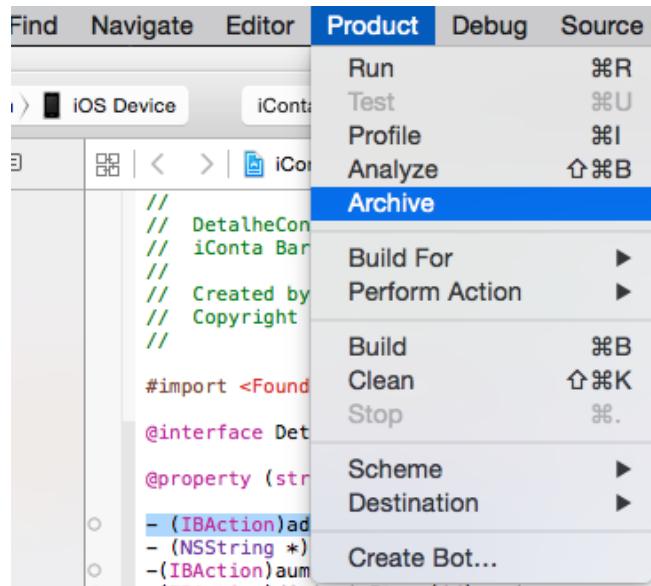
Existem algumas etapas que precisam ser cumpridas para a publicação de apps na App Store. Cada fase possui uma ferramenta que auxilia neste processo. Estes passos estão ilustrados da figura abaixo:

**Figura 45 – Ferramentas utilizadas em cada ciclo.**

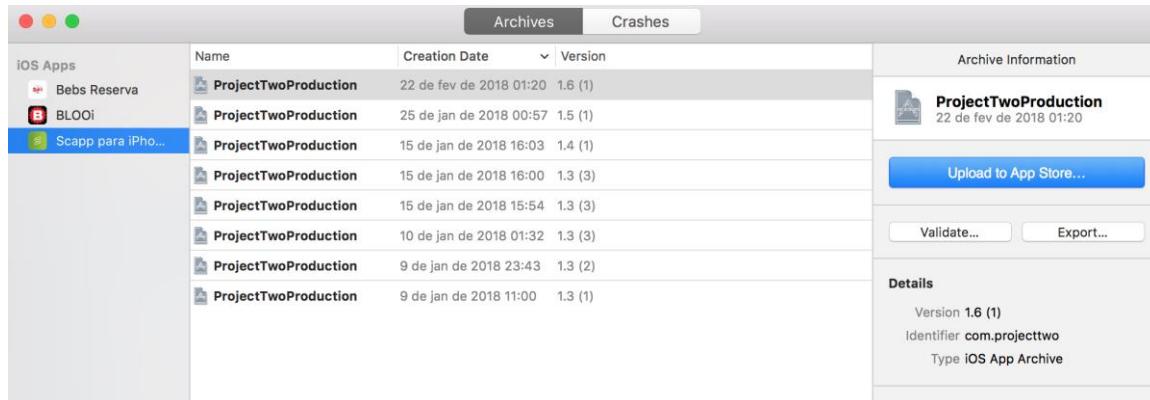
Para se tornar um desenvolvedor apple, é necessário acessar o Member Center no site developer.apple.com, e se cadastrar, pagando a anuidade. Nas fases de desenvolvimento utilizamos o XCode. Nele, além de desenvolver, criamos o registro do aplicativo, que seria o seu compilado, e realizamos também o upload deste compilado para a Apple. Depois disso, todo o acompanhamento passa a ser feito em outra ferramenta web, que é o Itunes Connect, disponível em itunesconnect.apple.com. Nele, é possível acompanhar todo o processo de aprovação e revisão da aplicação, e ainda controlar todas as datas de lançamento e disponibilização para o público, além de acompanhar as vendas em um dashboard específico para isso.

Falando especificamente da publicação de apps na App Store, existem três fases principais:

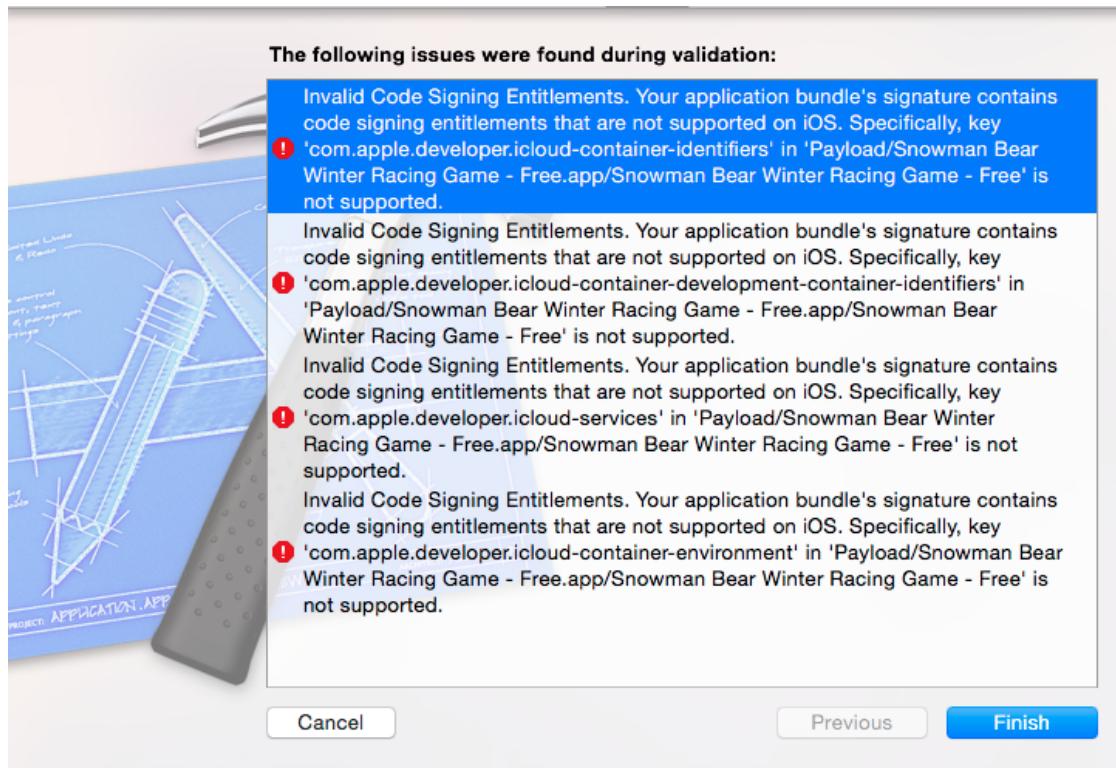
- Archive – Arquivar.
- Validation Tests – Testes de validação.
- Uploading to Apple – Upload da aplicação para revisão da Apple.

**Figura 46 – Arquivamento de aplicações.**

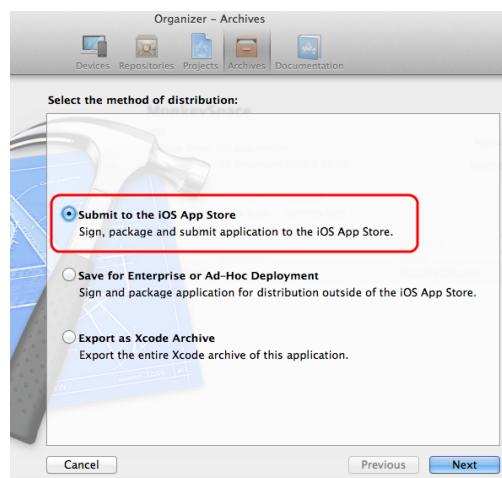
Após arquivar o App, existe uma parte específica do XCode, que é o Organizer. Nele são executadas validações iniciais antes de realizar o Upload para Apple.

**Figura 47 – Organizer.**

Todas as compilações realizadas, ficam registradas para avaliação:

**Figura 48 – Validação da aplicação para distribuição.**

Qualquer eventual erro de padrão que inviabilize a submissão da aplicação é apontado neste momento.

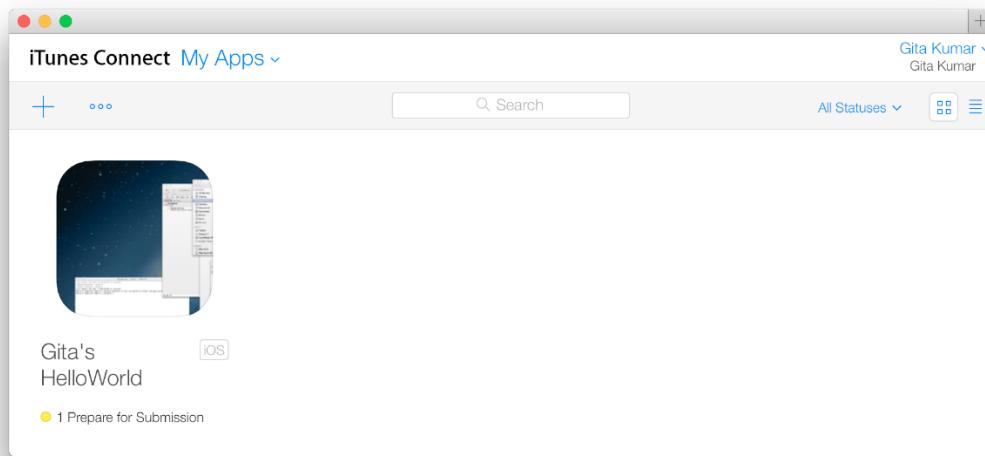
**Figura 49 – Submetendo aplicações.**

O último passo realizado no XCode é o de submeter a aplicação. Todo o acompanhamento passa agora a ser feito na web via Itunes Connect.

Funcionalidades principais do iTunes Connect – [itunesconnect.apple.com](http://itunesconnect.apple.com):

- Gerenciamento e status do app.
- Acompanha o processo de validação por parte da apple.
- Controla o data de disponibilidade.
- Preço.
- Etc.

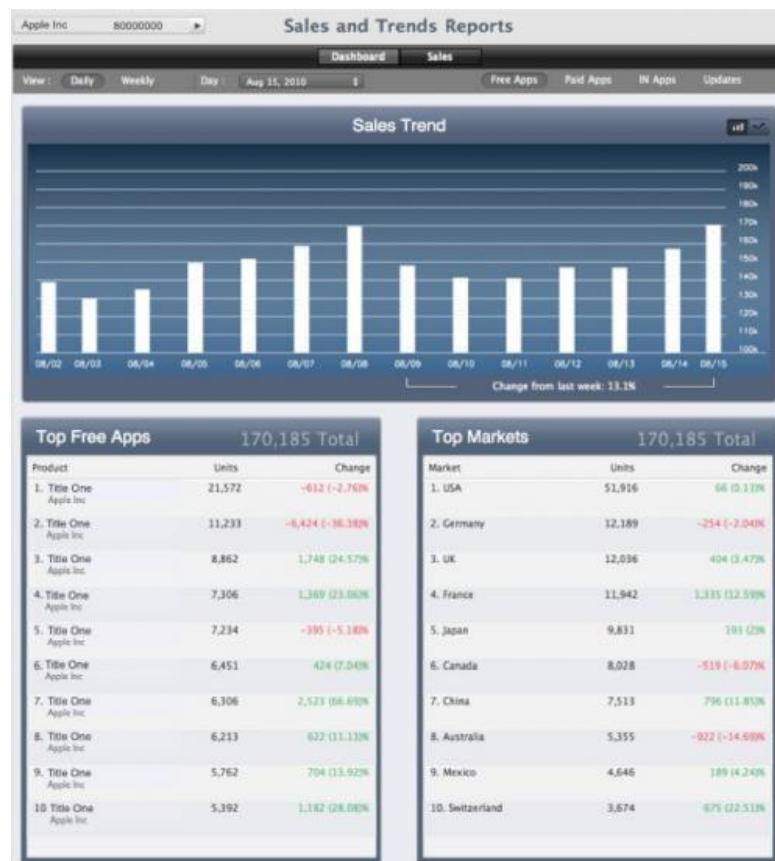
**Figura 50 – iTunes Connect.**



Acompanhamento de vendas no iTunes Connect feito pelo Cash Reports:

- Dashboards e gráficos.

**Figura 51 – Dashboard iTunes Connect: venda de Apps.**



## Capítulo 5. Persistência de dados

---

Armazenar dados e persistir o estado de uma aplicação é um item presente em praticamente todos os sistemas desenvolvidos em qualquer plataforma existente. Em iOS e dispositivos móveis não é diferente, vamos apresentar diversas formas de se persistir dados localmente no dispositivo ou na nuvem.

### 5.1. Arquivo

---

Um conceito importante para entender o sistema de arquivos de dispositivos iOS é o conceito de App SandBox. É uma espécie de container que isola cada aplicação, impede que interfiram entre si e que interfiram até no sistema de arquivos do sistema operacional. Com isso, só é possível gravar arquivos e ler arquivos de maneira direta somente dentro do contexto da própria aplicação.

Todo app possui diretórios /Documents e /tmp. Abaixo o código necessário para acessar cada um destes diretórios.

```
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,  
NSUserDomainMask, YES);
```

```
NSString *documentsDirectory = [paths objectAtIndex:0];
```

```
NSString *tempPath = NSTemporaryDirectory();
```

```
NSString *tempFile = [tempPath  
stringByAppendingPathComponent:@"tempFile.txt"];
```

Para armazenamento em arquivo é importante definir uma estratégia:

- Single File – Gravar todas as informações em um único arquivo:
  - Se baseia em um objeto raiz serializado em um único arquivo.
  - Todos os dados da aplicação precisam ser carregados e persistidos juntos.
  - Muito simples!
- Multiple File – Múltiplos arquivos que podem ser gravados e lidos de maneira independente:
  - Carregar e persistir dados independentes.

Outro tipo de arquivo são os Property Lists. São arquivos de listas chave/valor com a extensão plist. São utilizados para vários fins pelo XCode e configurações de projeto, e podem ser utilizados dentro do desenvolvimento de software pra iOS.

#### Características e benefícios:

- Podem ser editados manualmente no XCode.
- NSDictionary e NSArray podem ser carregados ou serializados em um Property List:
  - Os itens da coleção devem ser serializáveis!
- Outros objetos compatíveis: NSData, NSString, NSNumber, etc.
- Se conseguimos descrever nossos dados utilizando estes objetos, conseguimos utilizar property lists como base da nossa persistência de dados.

## Property Lists – Serializando:

```
[myArray writeToFile:@"/some/file/location/output.plist" atomically:YES];
```

### ▪ Property Lists – Desvantagens:

- Somente os objetos descritos anteriormente podem ser serializados em plists.
  - Classes como NSURL, NSImage e UIColor não podem ser utilizadas diretamente.
  - Como objetos não são armazenados diretamente, propriedades derivadas não são simples de ser implementadas. Lógicas deste tipo precisam ser movidas para controllers.

### ▪ Property Lists – Vantagens:

- Simples e indicadas para armazenar dados estáticos da aplicação.
- Se incluída na pasta Resources é compilada junto da aplicação.

Quando possuímos objetos mais complexos, informações mais difíceis de serem descritas e propriedades serializáveis, temos uma alternativa mais elaborada que são os Model Objects. Com Model Objects é possível serializar qualquer tipo de objeto. Objetos complexos podem ser gravados ou lidos de arquivos. Para isso, o objeto a ser serializado precisa implementar dois contratos: NSCoder e NSCopying.

### ▪ Model Objects – NSCoder:

- Implementar um método para serializar e outro para deserializar.
- `(void)encodeWithCoder:(NSCoder *)encoder`
- `(id)initWithCoder:(NSCoder *)decoder`

### ▪ Model Objects – NSCopying:

- Implementar um método que retorne uma instância de um novo objeto com as mesmas informações do atual.
- `(id)copyWithZone:(NSZone *)zone`

## 5.2. SQLite

---

Para um armazenamento de dados mais elaborado, confiável e estruturado, é sempre conveniente utilizar um sistema gerenciador de banco de dados. Todo dispositivo iOS possui um SGBD completo embutido, o SQLite. O SQLite é um database relacional como outros de mercado (SQLServer e Oracle, por exemplo) porém adequado para dispositivos com menos processamento, somente com recursos necessários para aplicações do porte de dispositivos móveis.

Fisicamente, toda base de dados criada com suas tabelas fica em um único arquivo gerenciado pelo SQLite. Cada aplicação pode ter um ou mais arquivos sqlite em seu sistema de arquivos.

Vantagens:

- Mais eficiente para armazenar e recuperar grandes quantidades de informação.
- Possui recursos que SGBDs maiores possuem, como criação de índices para acelerar performance de queries.
- <http://www.sqlite.org>.

Abrindo uma conexão com o SQLite e criando um novo arquivo caso não exista:

```
sqlite3 *database;  
int result = sqlite3_open("/path/to/database/file", &database);
```

O resultado deve ser igual a: SQLITE\_OK

Fechando conexão:

```
sqlite3_close(database);
```

Observações importantes:

- É boa prática abrir e fechar a conexão o mais rápido possível.
- Para manipular e criar dados, utilizamos comandos SQL padrão, como em qualquer SGBD (ex: SQL Server, Oracle, DB2, etc.)

Criando uma tabela:

```
char *errorMsg;
```

```
const char *createSQL = "CREATE TABLE IF NOT EXISTS USUARIO
```

```
(ID INTEGER PRIMARY KEY AUTOINCREMENT, DADOS TEXT);
```

```
int result = sqlite3_exec(database, createSQL, NULL, NULL, &errorMsg);
```

\*Como anteriormente devemos checar o resultado da operação.

Selecionando dados:

- O método sqlite3\_exec é utilizado para comandos que não retornam dados (criação de tabelas, inserts, updates, etc.)
- Para selecionar dados precisamos de mais alguns passos.

Preparar um comando com o código SQL de select:

```
NSString *query = @"SELECT ID, DADOS FROM USUARIO ORDER BY ROW";  
sqlite3_stmt *statement;
```

```
int result = (sqlite3_prepare_v2(database, [query UTF8String], -1, &statement, nil);
```

*Iterar sobre o statement (cursor), retornado:*

```

while (sqlite3_step(statement) == SQLITE_ROW)

{
    int      rowNum      =      sqlite3_column_int(statement, 0);
    char    *rowData     =      (char *)sqlite3_column_text(statement, 1);
    NSString *fieldValue = [[NSString alloc] initWithUTF8String:rowData];

    //Utilizar os dados retornados

    [fieldValue release];
}

sqlite3_finalize(statement);

```

Um boa prática ao se passar parâmetros em comandos SQL é a utilização de Bind variables. Nunca devemos utilizar concatenação de string para passagem de parâmetros, pois isso vai incidir diretamente na performance da execução destes comandos. Cada comando SQL é precompilado antes de ser executado pelo SGBD. Se concatenamos as variáveis sempre que enviamos o mesmo comando com variáveis diferentes, o SGBD vai recompilar, pensando ser um novo comando. Utilizando Bind variables somente uma vez o comando é pre compilado pois fica detectado que se trata de um mesmo comando com parâmetros diferentes.

```

char    *sql      =      "insert      into      tabela      values      (?,      ?);";
sqlite3_stmt
*stmt;
if (sqlite3_prepare_v2(database, sql, -1, &stmt, nil) == SQLITE_OK) {

    sqlite3_bind_int(stmt, 1, 235);

    sqlite3_bind_text(stmt, 2, "Bar", -1, NULL);

```

```
    }  
  
    if (sqlite3_step(stmt) != SQLITE_DONE) NSLog(@"This should be real error  
checking!");  
  
    sqlite3_finalize(stmt);
```

### 5.3. Core Data

---

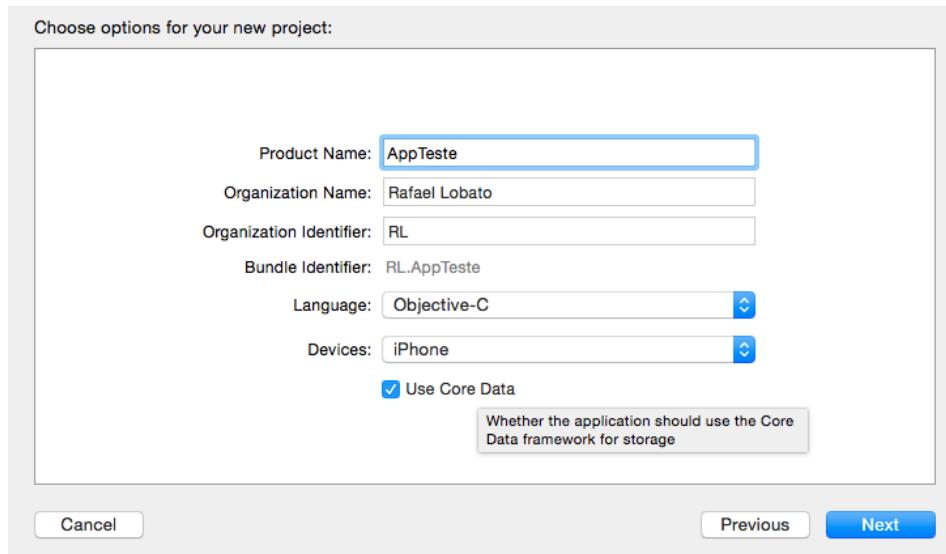
Praticamente toda plataforma de desenvolvimento hoje implementa o conceito ORM – Object Relational Mapping. Este conceito visa abstrair o acesso ao banco de dados, utilizando uma camada orientada a objetos. Objetos são mapeados a um modelo de dados de forma que um desenvolvedor, só precisa se preocupar em criar alterar e deletar objetos, e a camada ORM automaticamente converte estas ações sobre objetos em comando SQL a serem aplicados em um modelo de dados.

Desta forma, desenvolvedores que nem sempre são especialistas em banco de dados, não precisam se preocupar com boas práticas de banco, que são abstraídas pelo ORM. Em .NET, o ORM mais conhecido é o EntityFramework, em Java existem várias, dentre eles o Hibernate. Em desenvolvimento iOS e OS X existe o Core Data, implementação ORM na plataforma Apple.

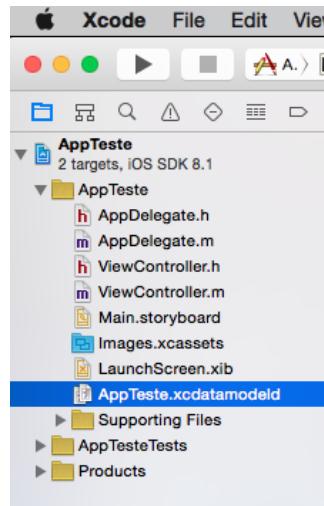
Pode ser trabalhado em duas abordagens:

- Database First – Gerar o modelo de objetos a partir de um modelo de dados existente.
- Code First – Gerar o modelo de dados a partir de um conjunto de classes definido.

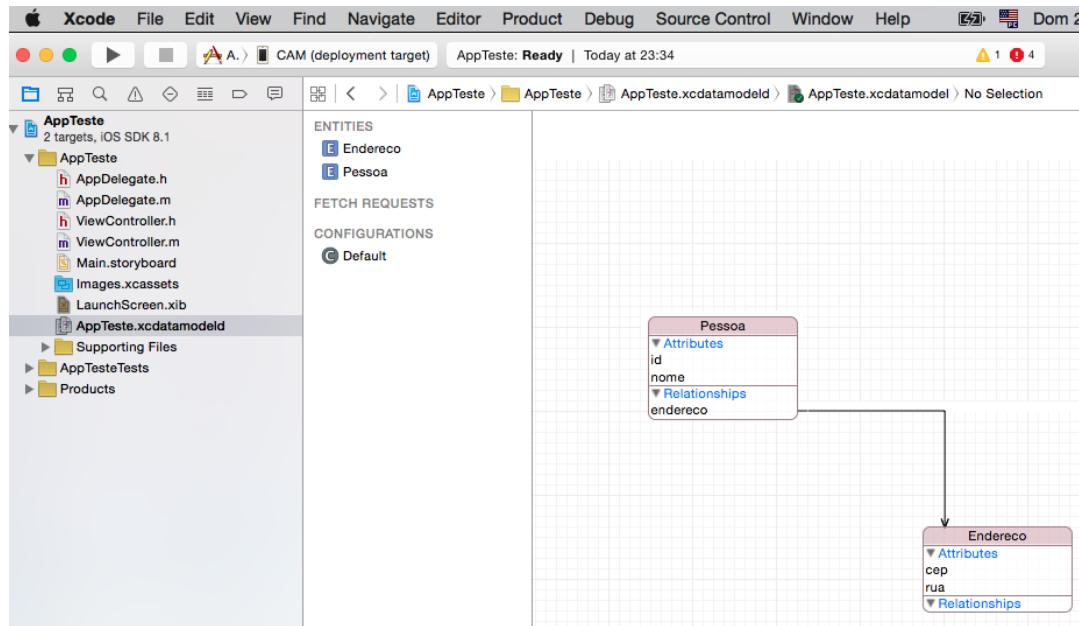
Ao criarmos um novo projeto no XCode, é apresentada a opção de já criar este projeto com suporte a Core Data.

**Figura 52 – Criando novos projetos com Core Data.**

Com isso, automaticamente são criadas as configurações básicas. Na lista de arquivos do projeto é possível verificar o arquivo que contém todo o modelo de objetos, com a extensão xcdatamodeld.

**Figura 53 – Arquivo com o modelo de dados.**

O Data Model pode ser editado visualmente.

**Figura 54 – Editor gráfico Core Data.**

As entidades são compostas por propriedades. No core data existem três tipos:

- Atributos.
- Relacionamentos.
- Fetched Properties – Lazy Loading – Estas são semelhantes a relacionamentos, porém não carregam a entidade relacionada até que seja utilizada.

Com Core Data podemos escolher a maneira que ele utiliza para persistir os dados por trás dos panos. A mais tradicional é como uma camada sobre o SQLite e, desta forma, automaticamente é criado um modelo no SQLite com tabelas e registros automaticamente, sem que o desenvolvedor precise entrar nos detalhes disso. Além disso, é possível utilizar como repositório arquivos binários, memória e iCloud.

Consumindo o Core Data:

- A interface principal com o código é o objeto de contexto – NSManagedObjectContext.
- O contexto é criado no AppDelegate do projeto.

Inserindo objeto:

```
Pessoa *p = (Pessoa *)[NSEntityDescription
insertNewObjectForEntityForName:@"Pessoa"
inManagedObjectContext:
managedObjectContext];

p.id = [[NSNumber alloc] initWithDouble: 1];

p.nome = @"Rafael";

NSError *error;

if (![managedObjectContext save: &error]) {

    return error;

}
```

Selecionando todos os objetos de um tipo:

```
NSEntityDescription *entity = [NSEntityDescription entityForName:@"Pessoa"
inManagedObjectContext:managedObjectContext];

NSFetchRequest *request = [[NSFetchRequest alloc] init];

request.entity = entity;

NSError *error;

NSMutableArray *mutableFetchResults = [[managedObjectContext
executeFetchRequest:request error: &error] mutableCopy];
```

Selecionando com “Where”:

```
NSPredicate *pred = [NSPredicate predicateWithFormat:@"(nome = %@",  
nameString];
```

```
[request setPredicate: pred];
```

Deletando entidade:

```
NSError *error;  
  
[managedObjectContext deleteObject:p];  
  
if (![managedObjectContext save: &error])  
  
{  
  
    return error;  
  
}
```

Vantagens:

- Com um único framework podemos implementar o mecanismo de armazenamento de dados e de cache de informações.
- Abstração do acesso a banco de dados – NoSQL.
- Redução da complexidade do código.
- Redução da quantidade de código necessário para persistência de dados.
- Mais intuitivo para desenvolvedores.

## 5.4. iCloud

A Apple disponibiliza para todos os seus usuários acesso ao serviço de armazenamento de dados na nuvem chamado iCloud. Todos possuem 5GB grátis de armazenamento e podem aumentar a capacidade com planos mensais.

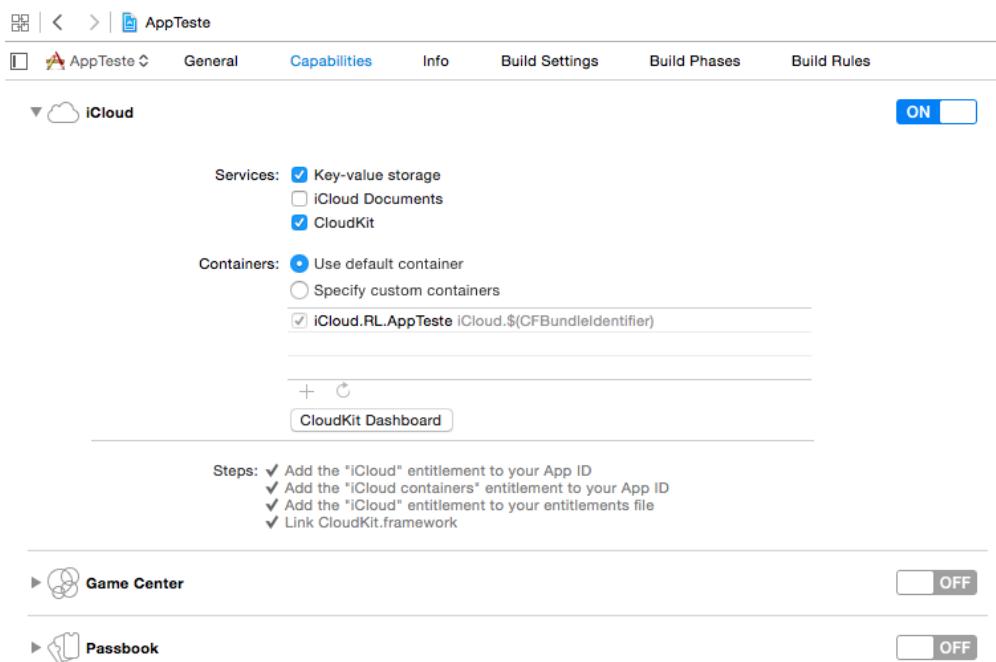
O Armazenamento pode ser acessado de qualquer dispositivo utilizado pela mesma conta e é usado para backups do dispositivo e para armazenamento de Contatos, Notas, E-mails, etc.

Em termos de desenvolvimento de software:

- Pode ser utilizado como armazenamento de dados de aplicações.
- Possui um SDK de desenvolvimento – CloudKit Framework.
- Se integra totalmente com Core Data.

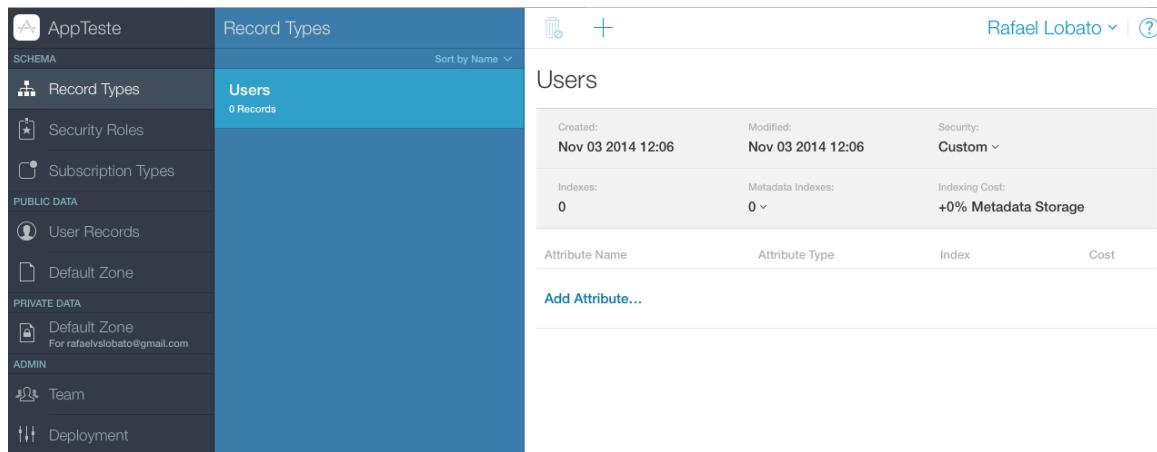
Habilitando iCloud para app nas propriedades do projeto.

**Figura 55 – Habilitando iCloud.**



Toda a administração do modelo de dados utilizado e das informações armazenadas pode ser feita no CloudKit Dashboard – [icloud.developer.apple.com/dashboard](https://icloud.developer.apple.com/dashboard).

**Figura 56 – Editor de dados iCloud.**



The screenshot shows the CloudKit Dashboard interface. On the left, there's a sidebar with a dark background containing various options like 'Record Types', 'Security Roles', 'Subscription Types', 'User Records', 'Default Zone', 'Default Zone (For rafaelvslobato@gmail.com)', 'Team', and 'Deployment'. The main area is titled 'Record Types' and shows a list for 'Users' with '0 Records'. To the right, there's a detailed view for a single 'User' record. It shows the creation date as 'Nov 03 2014 12:06', the modification date as 'Nov 03 2014 12:06', and the security level as 'Custom'. It also shows 'Indexes: 0', 'Metadata Indexes: 0', and 'Indexing Cost: +0% Metadata Storage'. There's a table at the bottom with columns 'Attribute Name', 'Attribute Type', 'Index', and 'Cost', which is currently empty. A button 'Add Attribute...' is visible.

### Containers e databases:

- Containers são a representação de um storage no iCloud.
- Representados pela classe CKContainer.
- Cada container é dividido em databases públicos e privados.
- Representados pela classe CKDatabase.

### Acessando o container e os databases via código:

```
let container : CKContainer
let publicDB : CKDatabase
let privateDB : CKDatabase

init() {
    // 1
    container = CKContainer.defaultContainer()
    // 2
    publicDB = container.publicCloudDatabase
    // 3
    privateDB = container.privateCloudDatabase
}
```

- Consumindo informação:
  - Dados são organizados em registros de chave-valor – CKRecord.
  - Objetos NSOperation permitem que registros com IDs conhecidos sejam recuperados.
  - Quando não se conhece o ID, utilizamos CKQuery e CKQueryOperation para efetuar consultas na nuvem.

Existem ambientes de desenvolvimento e de produção distintos na nuvem.  
Os simuladores iOS só têm acesso ao ambiente de desenvolvimento.

## Capítulo 6. Comunicação Remota

---

Aplicativos de grande porte muitas vezes não podem ser desenvolvidos utilizando apenas uma linguagem, infraestrutura ou plataforma. Principalmente para aplicativos móveis, a necessidade de se distribuir processamento é enorme. Neste capítulo, vamos ver como estabelecer a comunicação com serviços remotos permitindo desenvolver softwares muito mais robustos e confiáveis.

### 6.1. Requisições HTTP

---

Existem várias formas de se estabelecer uma comunicação remota. Porém existe um protocolo que pode ser considerado o principal para consumir informações remotas, que é o HTTP.

- **HTTP – Hypertext Transfer Protocol:**
  - Protocolo para requisições e resposta cliente-servidor.
  - O Web Browser pode ser um cliente e o servidor uma máquina hospedando um web site.
  - Um cliente sempre submete requisições e espera a resposta de um servidor.
  - Padrão de mercado aberto!

Exemplo de requisição HTTP:

*GET /index.html http/1.1*

*Host: [www.exemplo.com](http://www.exemplo.com)*

Exemplo de resposta HTTP:

*HTTP/1.1 200 OK*

*Date: Mon, 23 May 2005 22:38:34 GMT*

*Server: Apache/1.3.27 (Unix) (Red-Hat/Linux)*

*Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT*

*Etag: "3f80f-1b6-3e1cb03b"*

*Accept-Ranges: bytes*

*Content-Length: 438*

*Connection: close*

*Content-Type: text/html; charset=UTF-8*

O HTTP está em constante expansão e é possível trafegar com ele vários formatos de conteúdo. Com isso é necessário indicar na troca de mensagens o tipo de conteúdo trafegado, ou os MIME Types.

- **MIME Types:**

- *text/plain*
- *text/html*
- *Image/gif*
- *Image/jpeg*
- *Application/zip*

Com o HTTP é possível enviar e receber informação de várias formas. Estas formas são indicadas por verbos ou métodos. Principais métodos HTTP:

- **GET.**
- **POST.**

- PUT.
- DELETE.

URLs – Universal Resource Location são endereços que nos ajudam na localização do conteúdo que está distribuído na nuvem. Formato:

*esquema://domínio:porta/caminho/recurso?queryString##fragmento*

Como as mensagens são trafegadas em texto livre, existe um protocolo derivado ao HTTP, porém mais seguro por ser criptografado. O HTTPS, ou HTTP Seguro, ou HTTP com SSL.

Objetivos:

- Evitar a visualização de mensagens por terceiros.
- Geralmente o servidor é autenticado com um certificado digital.
- Segurança baseada em autoridades certificadoras:
  - O usuário confia que utiliza um browser seguro que implementa o HTTPS.
  - O usuário confia em quem emitiu o certificado do servidor, ou seja, na sua autoridade certificadora – Ex: Certisign.
  - As mensagens trocadas são criptografadas.

Ao digitar [www.google.com](http://www.google.com) no browser e pressionar enter, uma requisição GET HTTP é feita ao servidor do Google, que retorna o conteúdo de sua página inicial mais o header.

Como efetuar uma requisição HTTP programaticamente via Objective-C?

- Objective-C HTTP Request – Modelo Simplista:

```
NSString * urlString = @ "http://www.google.com/";
```

```
NSError * error;
```

```
NSString *locationString = [NSString stringWithContentsOfURL:[NSURL
URLWithString:urlString] encoding:NSUTF8StringEncoding error:&error];
```

- Objective-C HTTP Request :
  - Total controle da requisição:

1. Criando string com parametros do post:

```
NSString *post = [NSString stringWithFormat:@"Param1=%@&Param2=%@", @"teste1", @"teste2"];
```

2. Utilizando o encoding adequado e armazenando o tamanho da requisição:

```
NSData *postData = [post dataUsingEncoding:NSUTF8StringEncoding
allowLossyConversion:YES];
```

```
NSString *postLength = [NSString stringWithFormat:@"%d",[postData length]];
```

3. Criando o objeto principal da requisição e setando suas propriedades:

```
NSMutableURLRequest *request = [[[NSMutableURLRequest alloc] init]
autorelease];
```

```
[request setURL:[NSURL
stringWithFormat:@"http://www.abcde.com/xyz/teste.aspx"]];
```

```
[request setHTTPMethod:@"POST"];
```

```
[request setValue:postLength forHTTPHeaderField:@"Content-Length"];
```

```
[request setValue:@"application/x-www-form-urlencoded"
forHTTPHeaderField:@"Content-Type"];
```

```
[request setHTTPBody:postData];
```

#### 4. Criando e controlando a conexão HTTP:

```
NSURLConnection *conn = [[NSURLConnection alloc] initWithRequest:request  
delegate:self];  
  
if(conn) {  
  
    NSLog(@"Connection Successful");  
  
} else {  
  
    NSLog(@"Connection could not be made");  
  
}
```

#### 5. Métodos que controlam as respostas à requisição:

- *(void)connection:(NSURLConnection \*)connection didReceiveData:(NSData \*)data*
- *(void)connection:(NSURLConnection \*)connection didFailWithError:(NSError \*)error*
- *(void)connectionDidFinishLoading:(NSURLConnection \*)connection*

Com a expansão do HTTP, este passou a ser utilizado também para prover acesso aos métodos de serviços remotos. Com isso, surgiram os hoje famosos Web Services. Este tipo de serviço web utiliza toda a infraestrutura HTTP para implementação de serviços. Utiliza o padrão SOAP e XML para troca de mensagens e é ideal para integração de sistemas em plataformas diferentes. Possui um descriptor chamado de WSDL que descreve a assinatura de todos os seus métodos com parâmetros de entrada e retorno de informações.

- Consumindo Web Services:

- Utilizando somente código nativo Objective-C sem utilizar bibliotecas de terceiros.

- Seguir os mesmos passos de um post http com algumas diferenças.
- Consumindo Web Services:
  - O conteúdo da requisição precisa ser um XML no formato SOAP.

```
NSString *soapMessage = [NSString stringWithFormat:
@"><?xml version=\"1.0\" encoding=\"UTF-8\"?>\n"
"<SOAP-ENV:Envelope \n"
"xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" \n"
"xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" \n"
"xmlns:SOAP-ENC=\"http://schemas.xmlsoap.org/soap/encoding/\" \n"
"SOAP-ENV:encodingStyle=\"http://schemas.xmlsoap.org/soap/encoding/\" \n"
"xmlns:SOAP-ENV=\"http://schemas.xmlsoap.org/soap/envelope/\"> \n"
"<SOAP-ENV:Body \n"
"<Login xmlns=\"http://tempuri.org/\"><username>JACKSON</username><password>PASSWORD</p
"</Login> \n"
"</SOAP-ENV:Body> \n"
"</SOAP-ENV:Envelope>"];
```

- Consumindo Web Services:
  - Algumas diferenças nas propriedades da requisição.

```
NSURL *url = [NSURL URLWithString:@"http://172.16.0.142:8731/Service1/"];
NSMutableURLRequest *theRequest = [NSMutableURLRequest requestWithURL:url];
NSString *msgLength = [NSString stringWithFormat:@"%d", [soapMessage length]];
[theRequest addValue: @"text/xml; charset=utf-8" forHTTPHeaderField:@"Content-Type"];
[theRequest addValue: @"http://tempuri.org/IService1/Login" forHTTPHeaderField:@"Soapaction"];
[theRequest addValue: msgLength forHTTPHeaderField:@"Content-Length"];
[theRequest setHTTPMethod:@"POST"];
[theRequest setHTTPBody: [soapMessage dataUsingEncoding:NSUTF8StringEncoding]];
NSURLConnection *theConnection = [[NSURLConnection alloc] initWithRequest:theRequest de
if(theConnection) {
    webData = [[NSMutableData data] retain];
}
else {
    NSLog(@"theConnection is NULL");
}
```

- Consumindo Web Services:
  - Para facilitar o tratamento de resposta no formato XML, utilizar NSXMLParser.

```
- (void)parser:(NSXMLParser *)parser didStartElement:(NSString *)elementName  
namespaceURI:(NSString *)namespaceURI  
qualifiedName:(NSString *)qName  
attributes:(NSDictionary *)attributeDict  
  
-(void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string  
  
-(void)parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName  
namespaceURI:(NSString *)namespaceURI  
qualifiedName:(NSString *)qName
```

## 6.2. Serviços REST

Mesmo com a facilidade e padronização devido ao Web Services, ainda era necessário a criação de muitos mecanismos para se consumir uma informação. Ainda era possível simplificar mais e diminuir camadas na pilha de chamadas de métodos remotos. Com browsers poderosos, esta necessidade ficou ainda maior, pois estas informações agora são cada vez mais consumidas diretamente do frontend, utilizando JavaScript puro ou frameworks como jquery, Angular ou até HTML 5 puro. Esta simplificação veio através do padrão REST.

Para se adotar o padrão REST em serviços é necessário seguir algumas regras.

- REST – Representational State Transfer:
  - Procura abstrair protocolos de comunicação.
  - Descreve como o HTTP e URLs devem ser utilizadas.
  - Serviços REST tendem a funcionar melhor na Web.
  - Se bem implementados, tendem a ser performáticos, escaláveis e ainda mais interoperáveis do que Web Services.
  - Utiliza os métodos ou verbos padrão HTTP como principal forma de comunicação.
    - POST.
    - GET.
    - PUT.
    - DELETE.

- Princípios REST:

- De todas as coisas tem um identificador.
  - Vincule as coisas.
  - Utilize métodos padronizados.
  - Recursos com múltiplas representações.
  - Comunique sem estado.

- URLs que identificam itens:

- [www.exemplo.com/clientes/12](http://www.exemplo.com/clientes/12)
  - [www.exemplo.com/produtos/123](http://www.exemplo.com/produtos/123)

- Retorno de itens vinculados:

```
<order self="http://example.com/customers/1234">  
  
  <amount>23</amount>  
  
  <product ref="http://example.com/products/4554"></product>  
  
  <customer ref="http://example.com/customers/1234"></customer>  
  
</order>
```

Este mesmo retorno poderia ser representado em JSON.

- JSON – Javascript Object Notation: é um padrão muito utilizado para formatação de conteúdo, uma vez que é muito mais otimizado que XML. Por exemplo:

- Padrão aberto.
  - De fácil entendimento ao olho humano.
  - Alternativo e mais reduzido que o XML.
  - Muito utilizado na comunicação REST.
  - Possui APIs para iOS – NSJSONSerialization.

- Tecnologias Server Side para implementação de serviços:
  - WCF - .NET.
  - Web API - .NET.
  - JAX-WS – Java.
  - Spring – Java.
- Exemplo de Serviço REST Geolocation Google retornando conteúdo em XML:



The screenshot shows a browser window with the URL [maps.googleapis.com/maps/api/geocode/xml?address=av.contornobh](https://maps.googleapis.com/maps/api/geocode/xml?address=av.contornobh). Below the address bar, a message reads: "This XML file does not appear to have any style information associated with it. The document tree is shown below."

```
<?xml version="1.0" encoding="UTF-8"?>
<GeocodeResponse>
  <status>OK</status>
  <result>
    <type>route</type>
    <formatted_address>
      Avenida Contorno, Minas Gerais, República Federativa do Brasil
    </formatted_address>
    <address_component>
      <long_name>Avenida Contorno</long_name>
      <short_name>Av. Contorno</short_name>
      <type>route</type>
    </address_component>
    <address_component>
      <long_name>Betim</long_name>
      <short_name>Betim</short_name>
      <type>administrative_area_level_2</type>
      <type>political</type>
    </address_component>
    <address_component>
      <long_name>Minas Gerais</long_name>
      <short_name>MG</short_name>
      <type>administrative_area_level_1</type>
      <type>political</type>
    </address_component>
    <address_component>
      <long_name>República Federativa do Brasil</long_name>
      <short_name>BR</short_name>
      <type>country</type>
      <type>political</type>
    </address_component>
    <geometry>
      <location>
        <lat>-19.9640672</lat>
        <lng>-44.0964683</lng>
      </location>
    </geometry>
  </result>
</GeocodeResponse>
```

- Exemplo de Serviço REST Geolocation Google retornando conteúdo em JSON:



```
{
  "results" : [
    {
      "address_components" : [
        {
          "long_name" : "Avenida Contorno",
          "short_name" : "Av. Contorno",
          "types" : [ "route" ]
        },
        {
          "long_name" : "Betim",
          "short_name" : "Betim",
          "types" : [ "administrative_area_level_2", "political" ]
        },
        {
          "long_name" : "Minas Gerais",
          "short_name" : "MG",
          "types" : [ "administrative_area_level_1", "political" ]
        },
        {
          "long_name" : "República Federativa do Brasil",
          "short_name" : "BR",
          "types" : [ "country", "political" ]
        }
      ],
      "formatted_address" : "Avenida Contorno, Minas Gerais, República",
      "geometry" : {
        "bounds" : {
          "northeast" : {
            "lat" : -19.9583897,
            "lng" : -44.0930426
          },
          "southwest" : {
            "lat" : -19.9703616,
            "lng" : -44.0995235
          }
        }
      }
    }
  ],
  "status" : "OK"
}
```

- Objective-C HTTP Request Google REST Service:

- GET:

```
NSString *urlString = [NSString stringWithFormat:@"http://maps.googleapis.com/maps/api/geocode/xml?address=%@&sensor=true",
[addressSearch.text stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding]];
```

```
NSError *error;
```

```
NSString *locationString = [NSString stringWithContentsOfURL:[NSURL URLWithString:urlString] encoding:NSUTF8StringEncoding error:&error];
```

## Capítulo 7. Processos e Threads

---

Para se chegar em um nível muito mais elevado de experiência e responsividade, é indispensável conhecer e saber implementar conceitos de computação paralela. Criar vários processos que executados em paralelo deixam um sistema totalmente utilizável enquanto processa outras informações em background.

Atualmente até dispositivos móveis como iPhone e iPads já possuem processadores com mais de um núcleo, fazendo com que desenvolvedores tenham mais alternativas quando o assunto é computação paralela.

Computação Paralela, ou em inglês MultiTasking, pode ser definida grosseiramente como a execução de mais de um processo, ou thread, simultaneamente. Sem Threads, sistemas ficariam sem resposta durante o processamento de um trecho de código.

Paralelizando o processamento, viabilizamos uma melhoria de performance, porém não é uma bala de prata. Criar um processo paralelo tem um overhead considerável de processamento e deve ser feito com cautela apenas quando realmente é necessário. Além disso, existe um aumento considerável da complexidade do código.

- Dicas de design:
  - Evitar criar Threads explicitamente. Existem frameworks que abstraem a complexidade em se criar Threads.
  - Manter Threads ocupadas. Criar Threads que o pouco que executam pode piorar a performance de um sistema.
  - Evitar memória compartilhada. Sincronização de Threads para acesso aos recursos compartilhados trás overhead de processamento e complexidade de código.

- Projetar Threads para finalizarem naturalmente, sem execução de código específico para finalização das mesmas.
  - Ao projetar bibliotecas, prever multiprocessamento.
- 
- Criando Thread explicitamente – NSThread:  

```
NSThread* myThread = [[NSThread alloc] initWithTarget:self  
    selector:@selector(myThreadMainMethod:)  
    object:nil];  
  
[myThread start];
```
  - Criando Thread usando NSObject – Todo objeto herda funções para criação de threads ao herdar automaticamente de um NSObject:  

```
[myObj performSelectorInBackground:@selector(doSomething) withObject:nil];
```
  - Sincronização de Threads. Como definir blocos que só podem ser acessados de maneira sequencial por todas as Threads de um sistema:

```
NSLock* arrayLock = GetArrayLock();  
  
NSMutableArray* myArray = GetSharedArray();  
  
id anObject;  
  
[arrayLock lock];  
  
anObject = [myArray objectAtIndex:0];  
  
[arrayLock unlock];
```

*[anObject doSomething];*

- Sincronização de Threads com @synchronized:

```
- (void)myMethod:(id)anObj  
{  
    @synchronized(anObj)  
    {  
        // Trecho de código sincronizado  
    }  
}
```

Uma das maneiras de se criar processos paralelos com menos complexidade é utilizando o Grand Central Dispatch. O GCD é a implementação da Apple para abstrair desenvolvimento paralelo. Implementa o pattern Thread Pool, o que diminuir o overhead de processamento reutilizando Threads que foram previamente criadas e estão inutilizadas. Disponível para Mac OS X e iOS a partir do iOS4. Conceito chave: pilha (queue) de Threads e gerenciamento automático de Threads em background. O GCD reutiliza Threads criadas em um pool, para executar as atividades empilhadas.

Para explicar como o GCD funciona, vamos mostrar o passo a passo da criação de uma aplicação que simula o processamento demorado e como otimizá-lo utilizando Threads gerenciadas pelo GCD.

- Grand Central Dispatch – Como funciona:

- Aplicação de exemplo chamada SlowWorker.
- Ao clicar no botão Start Working:
  - Início de processo de longa duração.

- Após o fim deste processamento, exibe texto.
- Processamento totalmente síncrono.

**Figura 57 – App Slow Worker.**



```
#import <UIKit/UIKit.h>

@interface SlowWorkerViewController : UIViewController {

    UIButton *startButton;
    UITextView *resultsTextView;
}

@property (retain, nonatomic) IBOutlet UIButton *startButton;
```

```

@property (retain, nonatomic) IBOutlet UITextView *resultsTextView;

- (IBAction)doWork:(id)sender;

- (NSString *)fetchSomethingFromServer {

    [NSThread sleepForTimeInterval:1]; return @"Hi there";


}

- (NSString *)processData:(NSString *)data {

    [NSThread sleepForTimeInterval:2];

    return [data uppercaseString];


}

- (NSString *)calculateFirstResult:(NSString *)data {

    [NSThread sleepForTimeInterval:3];

    return [NSString stringWithFormat:@"Number of chars: %d", [data length]];


}

- (NSString *)calculateSecondResult:(NSString *)data {

    [NSThread sleepForTimeInterval:4];

    return [data stringByReplacingOccurrencesOfString:@"E" withString:@"e"];


}

- (IBAction)doWork:(id)sender {
    NSDate *startTime = [NSDate date];
    NSString *fetchedData = [self fetchSomethingFromServer];
}

```

```
NSString *processedData = [self processData:fetchedData];
NSString *firstResult = [self calculateFirstResult:processedData];

NSString *secondResult = [self calculateSecondResult:processedData];
NSString *resultsSummary = [NSString stringWithFormat:

    @"First: %@\nSecond: %@", firstResult,
    secondResult];

resultsTextView.text = resultsSummary;

NSDate *endTime = [NSDate date]; NSLog(@"Completed in %f seconds",
[endTime timeIntervalSinceDate:startTime]);

}
```

- Melhorando o app SlowWorker:

```
- (IBAction)doWork:(id)sender {
NSDate *startTime = [NSDate date];

dispatch_async(dispatch_get_global_queue(0, 0), ^{
    NSString *fetchedData = [self fetchSomethingFromServer];
    NSString *processedData = [self processData:fetchedData];
    NSString *firstResult = [self calculateFirstResult:processedData];

    NSString *secondResult = [self calculateSecondResult:processedData];

    NSString *resultsSummary = [NSString stringWithFormat:

        @"First: %@\nSecond: %@", firstResult,
        secondResult];

    resultsTextView.text = resultsSummary;

    NSDate *endTime = [NSDate date]; NSLog(@"Completed in %f seconds",
    [endTime timeIntervalSinceDate:startTime]);
});
```

});

}

- Problema – UIKit não é Thread safe. Não é possível chamar métodos da interface em background. Precisamos delegar a Thread principal a associação de texto aos labels. O GCD tem um bloco que facilita isso sem precisarmos refatorar o código.

```

- (IBAction)doWork:(id)sender {
    NSDate *startTime = [NSDate date];

    dispatch_async(dispatch_get_global_queue(0, 0), ^{
        .....
        dispatch_async(dispatch_get_main_queue(), ^{
            resultsTextView.text = resultsSummary;
        });
    });

    NSDate *endTime = [NSDate date];
    NSLog(@"%@", @"Completed in %f seconds",
    [endTime timeIntervalSinceDate:startTime]);
}
}; }
```

- O que mudou?
  - O botão não fica travado durante o processamento.
  - Sem um controle, o usuário consegue acionar esta ação várias vezes.
  - O resto da interface continua responsiva.
    - É possível, por exemplo, interagir com a área de texto.
- Mais melhorias:
  - O algoritmo continua sequencial, mesmo após as últimas mudanças.

- Os métodos calculateFirstResult e calculateSecondResult podem ser executados em paralelo.
- Utilizar conceito de execução em grupos com dispatch group.
- Todos os blocos de código disparados assincronamente no contexto de um grupo, vão ser executados o mais rápido possível em múltiplas Threads.
- Existe também um processo de notificação para executar trechos de código que só podem ser executados quando todos os processos de um grupo finalizarem.

```
- (IBAction)doWork:(id)sender {  
  
NSDate *startTime = [NSDate date];  
  
dispatch_async(dispatch_get_global_queue(0, 0), ^{  
  
    NSString *fetchedData = [self fetchSomethingFromServer];  
    NSString *processedData = [self processData:fetchedData];  
    __block NSString *firstResult;  
    __block NSString *secondResult;  
    dispatch_group_t group = dispatch_group_create();  
  
    dispatch_group_async(group, dispatch_get_global_queue(0, 0), ^{  
  
        firstResult = [[self calculateFirstResult:processedData] retain];  
    });  
  
    dispatch_group_async(group, dispatch_get_global_queue(0, 0), ^{  
  
        secondResult = [[self calculateSecondResult:processedData] retain];  
    });  
  
    dispatch_group_notify(group, dispatch_get_global_queue(0, 0), ^{  
  
        NSString *resultsSummary = [NSString stringWithFormat:  
            @"First: [% @]\nSecond: [% @]", firstResult, secondResult];  
    });  
});  
});
```

```
dispatch_async(dispatch_get_main_queue(), ^{
    resultsTextView.text = resultsSummary;
});

NSDate *endTime = [NSDate date];

[firstResult release];

[secondResult release];

});
}
```

- Observações:

- Tempo de execução total passou a ser de sete segundos.
- Em aplicações reais, o ganho depende do tipo de processamento e dos recursos acessados.
- Ganhos com esta técnica dependem da disponibilidade de mais de um núcleo no processador para processamento intensamente de CPU.
- Para consumo de recursos como acesso à rede, por exemplo, podem ter ganhos mesmo com um processador com um único núcleo.

Falando em um nível mais alto de processamento paralelo, existe a execução de vários aplicativos em paralelo por parte do sistema operacional. Isso é muito comum e natural em sistemas operacionais de grande porte como OS X, Windows e Linux.

Em dispositivos móveis é necessário economizar bateria, além de não possuirmos o mesmo poder de processamento de máquinas de grande porte. Para simular este comportamento de sistemas operacionais de grande porte, foi elaborado o Background Processing:

- Permite a execução de apps em segundo plano.
- Ainda não é multitask.

- Por padrão, apps saem da memória RAM ao pressionarmos HOME.
- Apps que precisam continuar executando, como apps de áudio ou que efetuam ligações, podem requisitar ao iOS este tipo de comportamento.
- Este tipo de requisito é declarado no arquivo Info.plist.

Nem todo tipo de processamento é permitido em background.

- Tipos de processamento em background:

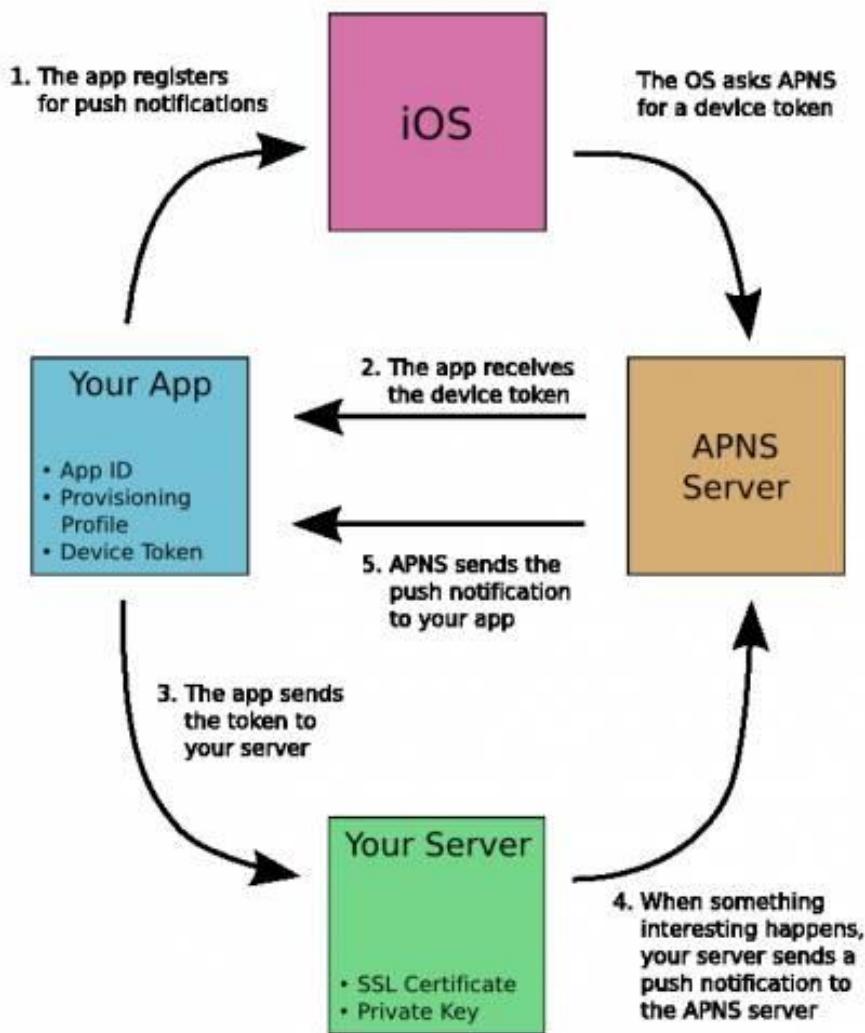
- Áudio.
- Location.
- VoIP.
- Newsstand-Content.
- External-Acessory.

Uma vez que não é possível manter em sua plenitude uma aplicação executando em segundo plano, um mecanismo diferente teve que ser criado para o envio de notificações ao usuário por parte das aplicações. O Push Notifications tem o objetivo de notificar usuários mesmo quando o app não está executando.

- Notificações por push podem:

- Exibir uma mensagem curta.
- Reproduzir um som.
- Setar um número no ícone do app.

Para implementar este tipo de recurso, para não ser necessário o app ficar em execução continuamente, um componente server-side é escrito para enviar este tipo de notificação. Entra em cena um serviço da Apple para comunicar com o serviço implementado e entregar efetivamente a mensagem. O serviço se chama Apple Push Notification Service.

**Figura 58 – Ciclo de vida push notifications.**

Para entender o que ocorre com a aplicação quando saímos da mesma, alternamos para outro aplicativo ou quando recebemos uma ligação, é necessário entender o seu ciclo de vida.

- Ciclo de vida da aplicação:
  - Not Running.
  - Active.
  - Background.
  - Suspended.
  - Inactive.

Um evento é chamado no código de cada app sempre que ocorre uma transição de estado do seu ciclo de vida.

- Notificações de mudança de estado:
  - application:didFinishLaunchingWithOptions:
  - applicationWillResignActive:
  - applicationWillBecomeActive:
  - applicationWillEnterBackground:
  - applicationWillEnterForeground:
  - applicationWillTerminate:
  
- Notificações de mudança de estado:
  - Active -> Inactive: applicationWillResignActive
  - Inactive -> Background: applicationWillEnterBackground
  - Background -> Inactive: applicationWillEnterForeground
  - Inactive -> Active: applicationWillBecomeActive

## Capítulo 8. Sensores

Sem dúvida, quando o primeiro iPhone foi lançado já com vários dos sensores encontrados nos dispositivos atuais, não se imaginava as infinitas alternativas de desenvolvimento de software que estavam nas mãos dos desenvolvedores. Conhecendo cada um dos sensores presentes e como utilizá-los no desenvolvimento de aplicativos, faz com que possamos, a qualquer momento, descobrir novas formas de interação com o usuário e novas formas de entrega de conteúdo.

- Acelerômetro:

- Sensor que indica a orientação e se há movimento no dispositivo.
  - O autorotation, quando viramos um iPhone, por exemplo, e toda a interface apresentada acompanha a sua nova orientação, utiliza este sensor.

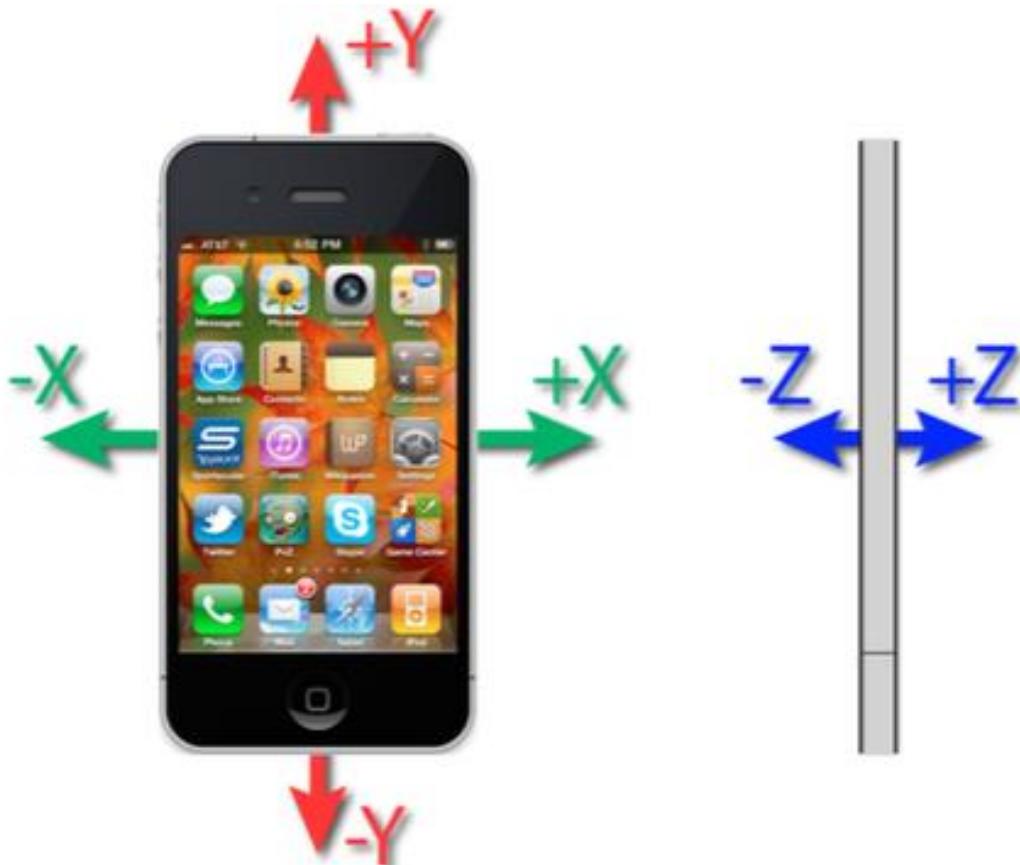
- Giroscópio:

- Determina o ângulo que o dispositivo está posicionado em relação a cada eixo.
  - Somente com o acelerômetro não conseguimos saber exatamente qual o ângulo que este se encontra. Geralmente, o giroscópio e o acelerômetro são utilizados em conjunto e por isso são abstraídos pelo mesmo framework de desenvolvimento.

- CoreMotion:

- Framework utilizado para acessar os valores destes sensores em aplicações iOS.

O acelerômetro presente nos dispositivos iOS possui três eixos, com isso é possível detectar não só que o dispositivo está sendo segurado, mas também se ele está sobre uma mesa, por exemplo, e ainda se está virado para cima ou para baixo.

**Figura 59 – Eixos acelerômetro.**

Entendendo a diferença dos valores do giroscópio e acelerômetro.

Considerando um iPhone deitado em uma mesa, ao rotacionar o dispositivo os valores do acelerômetro não se alteram. Os valores de rotação, detectados pelo giroscópio, alteram, mais precisamente o valor o eixo z. Girando no sentido horário gera um valor negativo no eixo Z; já no sentido anti-horário, um valor positivo. Ao parar de rotacionar, o valor volta para 0.

Agora falando de como interagir com estes sensores via código, precisamos utilizar o CoreMotion Framework.

Características principais:

- Classe principal CMMotionManager.

- Permite duas estratégias de utilização:
  - Executar um código (delegate) sempre que um movimento ocorrer.
  - Atualizar eternamente variáveis com últimos dados recebidos.
- Esta classe deve ser utilizada como singleton, ou seja, utilizar apenas uma instância em todo o aplicativo.
- Para acessar os dados do acelerômetro e giroscópio, utilizamos respectivamente CMAccelerometerData e CMGyroData.

A seguir um exemplo de código para utilizar o framework CoreMotion na modalidade Event Based:

```
#import <UIKit/UIKit.h>

#import <CoreMotion/CoreMotion.h>

@interface MotionMonitorViewController : UIViewController {

    CMMotionManager *motionManager;

    UILabel *accelerometerLabel;

    UILabel *gyroscopeLabel;

}

@property (nonatomic, retain) CMMotionManager *motionManager;

@property (nonatomic, retain) IBOutlet UILabel *accelerometerLabel;

@property (nonatomic, retain) IBOutlet UILabel *gyroscopeLabel;

@end

- (void)viewDidLoad {
```

```

[super viewDidLoad];

self.motionManager = [[[CMMotionManager alloc] init] autorelease];

NSOperationQueue *queue = [[[NSOperationQueue alloc] init] autorelease];

if (motionManager.accelerometerAvailable) {

    motionManager.accelerometerUpdateInterval = 1.0/10.0;

    [motionManager startAccelerometerUpdatesToQueue:queue withHandler:

        ^(CMAccelerometerData *accelerometerData, NSError *error){

            NSString *labelText;

            labelText = [NSString stringWithFormat:@"Accelerometer\n-----\nx: %+.2f\ny: %+.2f\nz: %+.2f",
accelerometerData.acceleration.x, accelerometerData.acceleration.y, accelerometerData.acceleration.z];

            [accelerometerLabel performSelectorOnMainThread:@selector(setText:)
withObject:labelText waitUntilDone:YES];

        }];
}

else {

    accelerometerLabel.text = @"This device has no accelerometer.';

}

if (motionManager.gyroAvailable)

{ motionManager.gyroUpdateInterval = 1.0/10.0;

[motionManager startGyroUpdatesToQueue:queue withHandler: ^(CMGyroData *gyroData, NSError *error) {

    NSString *labelText;
}
}

```

```
if (error) {

    [motionManager stopGyroUpdates]; labelText = [NSString stringWithFormat: @"Gyroscope encountered
error: %@", error];

} else {

    labelText = [NSString stringWithFormat:@"Gyroscope\n-----\nx: %.2f\ny: %.2f\nz: %.2f",
gyroData.rotationRate.x,
gyroData.rotationRate.y,gyroData.rotationRate.z];

}

[gyroscopeLabel performSelectorOnMainThread:@selector(setText© withObject:labelText waitUntilDone:YES];
}];

} else {

    gyroscopeLabel.text = @"This device has no gyroscope";

}

}

Ao executar este código, vemos os seguintes valores impressos na interface gráfica:
```

Accelerometer

X: +0.02

Y: -0.73

Z: -0.83

Gyroscope

X: +0.19

Y: +0.79

Z: +0.22

Agora um exemplo de acesso proativo aos dados. Neste caso:

```
@property (retain) NSTimer *updateTimer;

- (void)viewDidLoad
{
    [super viewDidLoad];
    self.motionManager = [[[CMMotionManager alloc] init] autorelease];

    if (motionManager.accelerometerAvailable) {

        motionManager.accelerometerUpdateInterval = 1.0/10.0;

        [motionManager startAccelerometerUpdates];

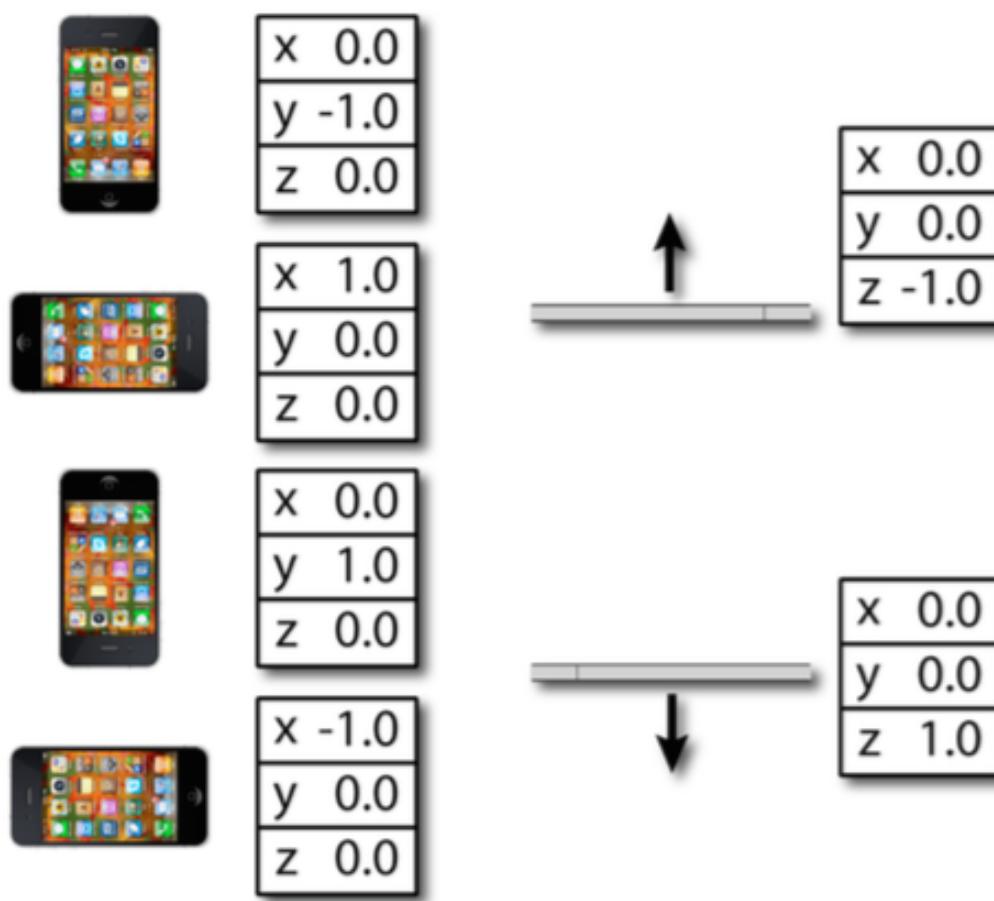
    }

    - (void)viewWillAppear:(BOOL)animated {
        self.updateTimer = [NSTimer scheduledTimerWithTimeInterval:1.0/10.0
            target: self selector:@selector(updateDisplay) userInfo:nil repeats:YES];
    }

    - (void)updateDisplay {
        if (motionManager.accelerometerAvailable) {
            CMAccelerometerData *accelerometerData = motionManager.accelerometerData;
        }
    }
}
```

```
accelerometerLabel.text = [NSString stringWithFormat: @"Accelerometer\n-----\nx: %+.2f\ny: %+.2f\nz: %+.2f", accelerometerData.acceleration.x, accelerometerData.acceleration.y, accelerometerData.acceleration.z];  
}  
}
```

**Figura 60 – Valores dos eixos por orientação.**



Outra empregabilidade de sensores é na implementação de gestos que podem ser utilizados como uma forma de interação com usuários. Um gesto específico a ser utilizado como exemplo é o gesto de “Shake”, quando sacudimos o dispositivo em qualquer direção. Apesar de existir uma maneira automática de se detectar este tipo de gesto, vamos implementar “do zero” esta funcionalidade para entender um pouco mais sobre a leitura dos valores dos sensores envolvidos.

Neste exemplo vamos:

- Monitorar variações bruscas dos valores do acelerômetro.
- Não é comum receber valores maiores que 1.3g.
- Podemos, por exemplo, estabelecer que valores de 1.5, em qualquer direção, correspondem a um shake leve, e maiores que 2.0 a um shake brusco.

Desta maneira podemos considerar que o usuário sacudiu o dispositivo em qualquer direção.

Apesar de nem todos os dispositivos iOS possuírem um GPS de geolocalização, ainda sim podemos desenvolver código que leva em consideração geolocalização, pois o framework que encapsula este sensor possui outras maneiras de obter a posição corrente do dispositivo em relação ao globo.

GPS:

- Conseguimos utilizar este recurso para obter a localização atual do dispositivo.
- O framework Core Location fornece acesso a todos os seus valores e funcionalidades.
- Consome muita bateria: deve ser utilizado somente quando necessário e com a precisão adequada.

Core Location abstrai a verdadeira técnica de localização, que pode ser feita em uma das três maneiras abaixo:

- GPS: utilizando o GPS embutido no aparelho.
- WPS: triangulação por redes Wi-Fi conhecidas.
- Triangulação torres celular 3g ou 4g.

Inicializando CLLocationManager:

- `CLLocationManager *locationManager = [[CLLocationManager alloc] init];`

Precisão:

- `locationManager.delegate = self;`
- `locationManager.desiredAccuracy = kCLLocationAccuracyBest;`

Definindo a precisão com constantes predefinidas:

- `kCLLocationAccuracyBest`
- `kCLLocationAccuracyNearestTenMeters`
- `kCLLocationAccuracyHundredMeters,`
- `kCLLocationAccuracyKilometer`
- `kCLLocationAccuracyThreeKilometers`

Filtro de distância – Para definirmos que só precisamos ser notificados quando uma alteração maior que a distância abaixo ocorrer:

- `locationManager.distanceFilter = 1000.0f;`

Iniciando e parando o Location Manager

- `[locationManager startUpdatingLocation];`
- `[locationManager stopUpdatingLocation];`

CLLocationManagerDelegate – Implementa o método abaixo, que notifica quando a localização foi alterada, passando por parâmetro a localização antiga e a nova:

- `locationManager:didUpdateToLocation:fromLocation:`
  - `CLLocation`

Acessando geolocalização ou posição atual do dispositivo em relação ao globo:

*CLLocationDegrees latitude = theLocation.coordinate.latitude;*

*CLLocationDegrees longitude = theLocation.coordinate.longitude;*

*CLLocationDistance altitude = theLocation.altitude;*

CLLocation – Classe que contém as informações de uma geolocalização:

*CLLocationDistance distance = [fromLocation distanceFromLocation:toLocation];*

Outro sensor que ajuda na geolocalização, mas que pode ser utilizado em outros contextos, é o sensor magnético. Ele ajuda a definir para onde o dispositivo está apontado em relação ao norte da terra. Além disso, é simples transformar um iPhone em um detector de metais com o framework CoreMotion utilizado para encontrar o norte. Da mesma forma que os sensores mencionados anteriormente, precisamos indicar o início da sua utilização, startMagnetometerUpdates, e com isso conseguimos acessar às propriedade magneticField. Esta propriedade mede a força de algum campo magnético sobre o aparelho

## Capítulo 9. Tópicos avançados

---

No capítulo de tópicos avançados serão abordados dois temas recentes, não só no nível de desenvolvimento iOS, que ganharam dois frameworks específicos para facilitar a utilização destes conceitos em qualquer aplicativo.

### 9.1. MLKit

---

Machine Learning é uma área da inteligência artificial, onde ensinamos máquinas a desempenharem algumas funções. Diferente do desenvolvimento de algorítimos convencionais, em Machine Learning um conjunto de dados de entrada são utilizados para treinar um modelo e, baseado nos padrões encontrados, gera as saídas mais conhecidas como previsões.

Existem vários algoritmos utilizados no treinamento de máquina, dentre eles os principais são o de Aprendizagem Supervisionada e Regressão. No primeiro, podemos citar um exemplo de aplicação onde o modelo consegue determinar se um e-mail é ou não um spam, baseado em padrões e massa de dados utilizada no seu treinamento. O Segundo é utilizado em cenários de como determinar preço de uma casa. A diferença principal entre eles é que em Regressão temos resultados mais lineares e mais facilmente representados em gráficos crescentes, e na aprendizagem supervisionada pode representar respostas mais granulares.

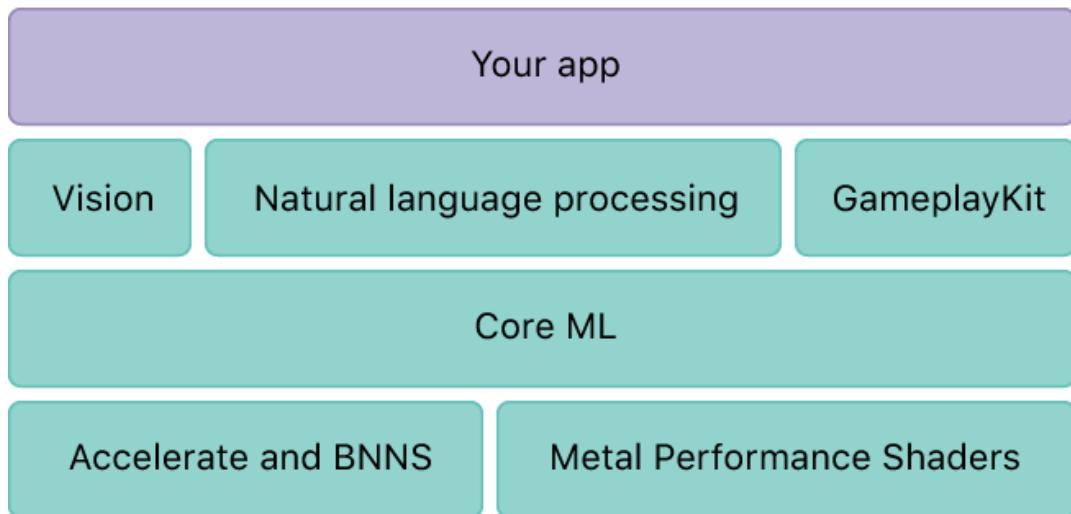
Atualmente, já convivemos com vários recursos que utilizam Machine Learning:

- Processamento de Linguagem Natural:
  - Capacidade de interpretar strings ou áudios não estruturados, que podem variar muito de forma e traduzir isso em tokens ou pedaços de informações processáveis por um sistema tradicional. Atualmente a

pesquisa padrão do Google já consegue dar resultados baseados em pesquisa por linguagem natural. Ex: Pesquisar por Restaurantes próximos do bairro de Lourdes.

- Reconhecimento de objetos em imagens:
  - Reconhecer tipos de objetos em imagens como pessoas, rostos, humor do rosto detectado, veículos, árvores etc. No próprio aplicativo de fotos do iOS, se você pesquisar, por exemplo, por cachorro ou praia ou água, você vai ver que suas fotos que possuem este tipo de objeto vão ser retornadas, pois foram interpretadas por um algoritmo de Machine Learning.
- Análise de fraude:
  - Grandes bancos já fazem análise automática de probabilidade de fraudes em transações financeiras utilizando ML.
- Diagnóstico clínico e Análises Jurídicas já estão sendo feitas sem a presença de um médico ou advogado.
- Carros autônomos.
- Etc.

Para possibilitar que consigamos ter recursos de Machine Learning em nossos apps iOS, a Apple disponibilizou um framework chamado MLKit. É possível ter recursos como os citados acima sem ter nenhum conhecimento prévio em treinamento de modelos e ML.

**Figura 63 – Hierarquia de componentes de ML no iOS.**

Para aqueles que querem utilizar ML sem ter conhecimento, ou sem ter que treinar um modelo específico, é possível efetuar o download de um modelo compatível com MLKit e utilizar com pouco código necessário. Dois exemplos são MobileNet, que detecta mais de 1000 tipos de objetos em images como veículos, árvores, animais, pessoas, etc. e Places205-GoogLeNet, que detecta categorias de imagens como terminais de aeroporto, quartos, floresta, costa, etc.

Para aqueles que possuem conhecimento, é possível criar e treinar o próprio modelo.

## 9.2. ARKit

Realidade aumentada pode ser descrita como a integração de informações virtuais a visualizações do mundo real. Quando falamos de realidade aumentada, lembramos de filmes de ficção onde a coisa ocorre com hologramas e projeções em 3D. Na vida real, várias tentativas estão sendo feitas a respeito, como o Google Glass por exemplo, porém sem muito sucesso. Enquanto esperamos o surgimento de um hardware capaz de possibilitar realidade aumentada em sua plenitude, a melhor opção que temos é fazer isso através das lentes da câmera de um

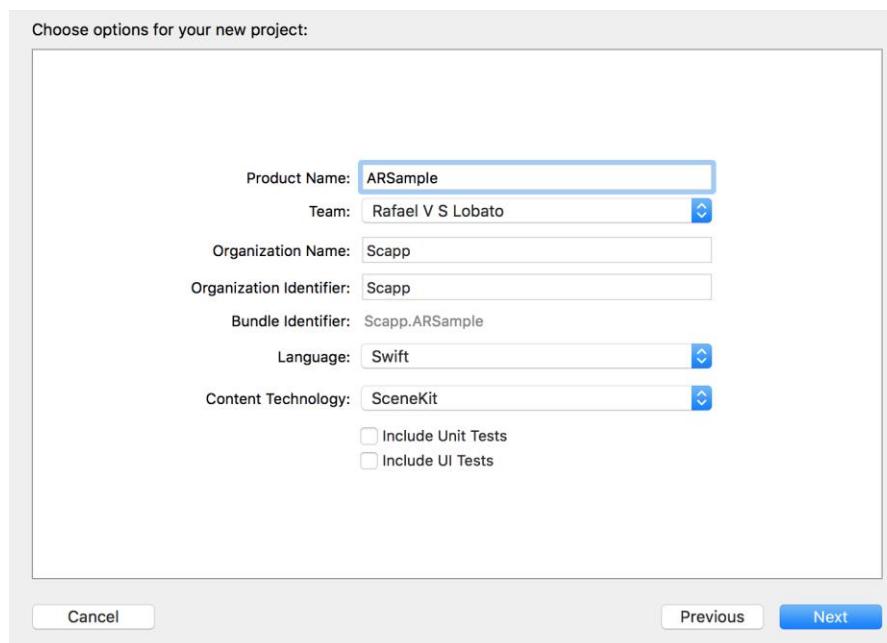
smartphone. Desta forma, conseguimos, vendo o mundo real pela tela do telefone, projetar informações virtuais e dar um grande passo rumo a realidade aumentada.

O ARKit é o framework da Apple para abstrair o desenvolvimento de realidade aumentada. Um dos maiores desafios de se projetar itens virtuais no mundo real é saber rastrear o mundo e garantir que não temos objetos oscilando em posição, tirando do usuário a sensação de estar vendo algo real. O ARKit trás conceitos como:

- World Tracking:
  - Mapear objetos reais e estabelecer uma relação entre estes objetos e os objetos virtuais projetados.
- PlaneDetection:
  - Detecção de superfícies planas para a projeto de objetos.
- Light estimation:
  - Estimativa de luz ambiente e o impacto desta sobre objetos virtuais.

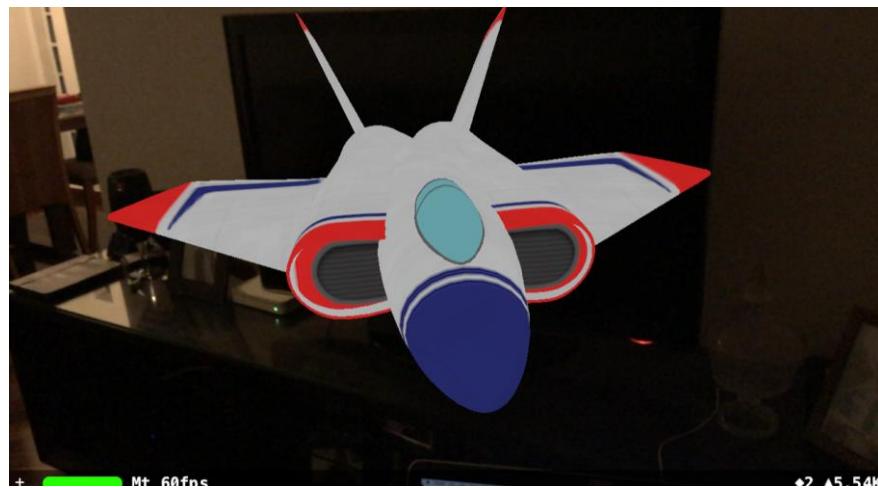
Com estes recursos, o ARKit abstrai grande parte destas dificuldade para o desenvolvedor.

**Figura 64 – Template de projeto para o ARKit.**



Para se criar um projeto com realidade aumentada para iOS, existe um template específico que permite inclusive selecionar qual a tecnologia de processamento gráfico sera utilizada. O ARkit precisa de uma das bibliotecas de processamento gráfico disponíveis para gerar os objetos virtuais a serem projetados no mundo real.

**Figura 65 – Execução do projeto exemplo de ARkit para o XCode.**



Na imagem acima, podemos observer a execução do projeto básico criado utilizando SceneKit como framework para processamento de imagem. Já no exemplo abaixo, é possível observar que todo o código necessário girou em torno de referenciar e instanciar um modelo SceneKit existente e executá-lo com base em um objeto do ARKit chamado ARWorldTrackingConfiguration, que vai abstrair como e onde esse objeto virtual deve ser projetado no mundo real.

**Figura 66 – Código necessário para projetar um objeto virtual.**

```
import UIKit
import SceneKit
import ARKit

class ViewController: UIViewController, ARSCNViewDelegate {

    @IBOutlet var sceneView: ARSCNView!

    override func viewDidLoad() {
        super.viewDidLoad()

        // Set the view's delegate
        sceneView.delegate = self

        // Show statistics such as fps and timing information
        sceneView.showsStatistics = true

        // Create a new scene
        let scene = SCNScene(named: "art.scnassets/ship.scn")!

        // Set the scene to the view
        sceneView.scene = scene
    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)

        // Create a session configuration
        let configuration = ARWorldTrackingConfiguration()

        // Run the view's session
        sceneView.session.run(configuration)
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)

        // Pause the view's session
        sceneView.session.pause()
```

## Referências

---

KOCHAN, Stephen G. *Programming in Objective-C* (Developer's Library). 6. ed. Addison-Wesley Professional, 2013.

MARK, Dave; LAMARCHE, Jeff; NUTTING, Jack. *Beginning Iphone 4 Development: Exploring the iPhone SDK*. 1. ed. Apress, 2011.

NUTTING, J et al. *Beginning iOS 7 Development: Exploring the iOS SDK*. Apress, 2014.