

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-
GRANDENSE - IFSUL, CÂMPUS PASSO FUNDO
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET**

MARINA DEZORDI LOPES

**UMA ABORDAGEM SOBRE A LINGUAGEM DE PROGRAMAÇÃO
OBJECTIVE-C ATRAVÉS DO DESENVOLVIMENTO DE UM ESTUDO
DE CASO**

José Antônio Oliveira de Figueiredo

PASSO FUNDO, 2014

MARINA DEZORDI LOPES

**UMA ABORDAGEM SOBRE A LINGUAGEM DE PROGRAMAÇÃO
OBJECTIVE-C ATRAVÉS DO DESENVOLVIMENTO DE UM ESTUDO
DE CASO**

Monografia apresentada ao Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-Rio-Grandense, Câmpus Passo Fundo, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador: José Antônio Oliveira de Figueiredo

PASSO FUNDO, 2014

MARINA DEZORDI LOPES

**UMA ABORDAGEM SOBRE A LINGUAGEM DE PROGRAMAÇÃO OBJECTIVE-
C ATRAVÉS DO DESENVOLVIMENTO DE UM ESTUDO DE CASO**

Trabalho de Conclusão de Curso aprovado em ____/____/____ como requisito parcial para a
obtenção do título de Tecnólogo em Sistemas para Internet

Banca Examinadora:

Esp. José Antônio Oliveira de Figueiredo

Msc. Élder Francisco Fontana Bernardi

Msc. Fernando Abrahão Afonso

Dr. Alexandre Tagliari Lazzaretti

PASSO FUNDO, 2014

*À minha família,
pelo incentivo e apoio
que sempre recebi.*

AGRADECIMENTOS

Agradeço a Deus, a fé sempre me ajudou a ser persistente e enfrentar os obstáculos.

Aos meus pais, pelos valores de vida que me foram passados, pelos cuidados, estímulos e por estarem ao meu lado sempre, fatores essenciais durante minha jornada.

Ao meu irmão, por me passar bons exemplos, estar presente durante os momentos bons e difíceis e ser essencial na minha vida.

Ao meu orientador, Prof. José Figueiredo, pela excelente orientação, sinceridade em suas observações, pela motivação transmitida e por passar seus conhecimentos de forma objetiva e clara durante o semestre.

À Profa. Anúbis, pela orientação no Projeto de Conclusão I, por ter sido sempre prestativa, atenciosa e pelos aprendizados que me passou.

Ao Instituto Federal Sul-Rio-Grandense, Câmpus Passo Fundo, pelo ensino de qualidade que oferece e pela abertura de espaço para que esse trabalho pudesse ser realizado.

Aos meus colegas Alisson, Daniel e Luana, os quais me acompanharam desde o início do curso, por termos passados por momentos de concentração, descontração, angústias, alívios, desânimos e alegrias sempre juntos, apoiando uns aos outros.

E a todas as pessoas que de alguma forma contribuíram para que esse trabalho fosse realizado, meus sinceros agradecimentos.

“Algumas pessoas acham que foco significa dizer sim para o que você irá se focar. Mas não é nada disso. Significa dizer não às centenas de outras boas ideias que existem. Você precisa selecionar cuidadosamente.”

Steve Jobs

RESUMO

O presente trabalho expõe um estudo sobre desenvolvimento de aplicativos para iOS, utilizando a linguagem de programação Objective-C. Através de um estudo de caso, implementado para iPad (tablet com sistema operacional iOS), demonstra-se uma das perspectivas em que o conhecimento nessa área pode ser conveniente. O estudo de caso também apresenta aspectos essenciais e outros específicos da linguagem de programação Objective-C.

Palavras-chave: Objective-C; iOS; SQLite.

ABSTRACT

This paper presents a study on developing applications for iOS using the Objective-C language programming. Through a case study, implemented for iPad, tablet with iOS operating system, it brings forward a perspective that knowledge in this theme may be appropriate. The case study also presents essential and specific aspects of the Objective-C programming language.

Key words: Objective-C; iOS; SQLite.

LISTA DE QUADROS

Quadro 1: Arquivo header	20
Quadro 2: Arquivo de implementação.....	21
Quadro 3: Estrutura de um método no Objective-C	22
Quadro 4: Método com parâmetro.....	22
Quadro 5: Código para importar biblioteca do SQLite	23
Quadro 6: Implementação da <i>UINavigationController</i>	30
Quadro 7: Implementação de <i>UIBarButtonItem</i>	30
Quadro 8: Implementação de <i>UITableViewDelegate</i> e <i>UITableViewDataSource</i>	31
Quadro 9: Método para toque em célula na tabela	32
Quadro 10: Método para reaproveitar células	33
Quadro 11: Método para retornar o número de linhas da tabela	33
Quadro 12: Método para retornar caminho do arquivo	34
Quadro 13: Método para abrir banco de dados.....	35
Quadro 14: Método para criação de tabela no banco de dados	35
Quadro 15: Trecho de código com o SQL de criação de tabela.....	36
Quadro 16: Trecho de código com o SQL de inserção na tabela	36
Quadro 17: Variável <i>stmt</i>	36
Quadro 18: Informa o valor do cliente a ser inserido/editado	37
Quadro 19: Leitura de coluna do banco de dados	38

LISTA DE FIGURAS

Figura 1: Camadas do iOS.....	15
Figura 2: XCode	17
Figura 3: MVC no Objective-C.....	19
Figura 4: Diagrama de casos de uso	28
Figura 5: Diagrama de classes.....	29
Figura 6: Tela com UINavigationController	31
Figura 7: <i>Swipe</i> na segunda célula.....	37
Figura 8: Tela de login.....	39
Figura 9: Tela de início.....	40
Figura 10: Tela de listagem de clientes	41
Figura 11: Tela de descrição de cliente	42
Figura 12: Tela de listagem de visitas	43
Figura 13: Tela de descrição de visitas.....	44
Figura 14: Tela de formulário do cliente.....	45
Figura 15: Tela de formulário de visitas.....	46

SUMÁRIO

1	INTRODUÇÃO.....	12
1.1	MOTIVAÇÃO.....	12
1.2	OBJETIVOS.....	12
1.2.1	Objetivo Geral.....	12
1.2.2	Objetivos específicos.....	13
2	REFERENCIAL TEÓRICO.....	14
2.1	IOS.....	14
2.1.1	Evolução do iOS.....	14
2.1.2	Arquitetura do iOS.....	15
2.2	OBJECTIVE-C.....	16
2.2.1	Plataforma de desenvolvimento XCode.....	16
2.2.2	Conceitos fundamentais para programação em Objective-C.....	18
2.2.3	Características de uso recorrente no Objective-C.....	20
2.3	PERSISTÊNCIA DE DADOS.....	23
2.4	TECNOLOGIAS RELACIONADAS.....	24
3	ESTUDO DE CASO.....	25
3.1	CONTEXTO EMPRESARIAL.....	25
3.2	MODELAGEM DO SISTEMA.....	28
3.2.1	Diagrama de casos de uso.....	28
3.2.2	Diagrama de classes.....	29
3.3	DESENVOLVIMENTO DO SISTEMA.....	30
3.3.1	Navegação entre telas.....	30
3.3.2	Tabelas.....	31
3.3.3	Persistência de dados.....	34
3.3.4	Telas do sistema.....	38
4	CONSIDERAÇÕES FINAIS.....	47
4.1	TRABALHOS FUTUROS.....	47
	REFERÊNCIAS.....	49
	ANEXOS.....	51

1 INTRODUÇÃO

O uso de dispositivos móveis tem aumentado com o passar do tempo devido às facilidades que eles podem proporcionar à vida pessoal e também profissional das pessoas. Os celulares que antes tinham a conveniência de conectar pessoas via ligações ou mensagens de textos, estão dando lugar aos smartphones que aumentam inúmeras vezes as possibilidades de utilizações, principalmente por se tratar de dispositivos habilitados a conectar-se à internet.

O sistema operacional iOS, da empresa Apple, é um dos mais utilizados nesse contexto, muitos usuários, empresas, desenvolvedores o preferem por gostarem da interface intuitiva e confortável de usar ou ainda por ele prover diversos recursos úteis e satisfatórios. A forte dinâmica do mercado de dispositivos móveis com iOS gera uma consequente demanda de softwares para esse mercado.

Surge, então, a necessidade de pessoas capacitadas ao desenvolvimento de aplicativos para iOS, a partir do uso da linguagem de programação Objective-C. Há ainda muitas soluções a serem implementadas, porém poucas pessoas com o conhecimento necessário que possam colocar as ideias em prática, fato que incentiva a busca por conhecimentos nessa área.

Na realização do presente trabalho mostrou-se necessário o estudo do sistema operacional iOS, os principais aspectos da linguagem de programação Objective-C e ainda uma abordagem sobre persistência de dados. Para que se pudesse evidenciar como aplicações para iOS podem ser úteis e também demonstrar as particularidades desse tipo de programação foi desenvolvido um estudo de caso baseado nas problemáticas de uma empresa em crescimento.

1.1 MOTIVAÇÃO

Expandir o conhecimento em programação para dispositivos móveis, visto que é uma área em ascensão no mercado, a qual possibilita a criação de soluções para diversas áreas da vida cotidiana das pessoas.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Apresentar um estudo sobre a linguagem de programação Objective-C para o sistema operacional móvel iOS implementando um simples estudo de caso.

1.2.2 Objetivos específicos

- Fundamentar o sistema operacional iOS.
- Identificar os conceitos cruciais da linguagem Objective-C;
- Apresentar algumas particularidades da linguagem Objective-C em um estudo de caso;
- Demonstrar as mais importantes características da ferramenta de desenvolvimento para iOS em um estudo de caso.

2 REFERENCIAL TEÓRICO

As seções a seguir visam dar embasamento teórico ao presente trabalho. Serão abordados os assuntos iOS, Objective-C e persistência de dados.

2.1 IOS

O iOS é um sistema operacional direcionado aos equipamentos iPhone, iPad e iPod Touch, que são smartphone, tablet e player de áudio digital, respectivamente desenvolvidos pela empresa Apple.

2.1.1 Evolução do iOS

Lançado juntamente com o iPhone, em nove de janeiro de 2007, o iOS passou por evoluções desde então. Na primeira versão, a 1.0, haviam três funções básicas: a de celular, iPod e a capacidade de acessar a internet. Ainda não era possível fazer download de aplicativos, apenas tinha-se acesso àqueles que vinham nativos no dispositivo. Dessa forma, na versão 2.0, a Apple Store foi a maior mudança anunciada, a partir dela faziam-se download de aplicativos, já que a Apple havia lançado o SDK (Kit de Desenvolvimento de Software) e desenvolvedores podiam programar para iOS (FERNANDES, 2013).

A versão 3.0, de 2009, trouxe algumas novidades que estavam sendo requisitadas pelos usuários: a funcionalidade de copiar e colar, mensagem MMS, Spotlight (caixa de pesquisa), Push (notificações de atualização) e teclado horizontal. (SILVA, 2010). Foi na versão 3.2 que surgiu a compatibilidade com o iPad (FERNANDES, 2013), tablet desenvolvido pela empresa Apple e anunciado em 2010. Também no ano de 2010, é anunciado o iOS 4.0, além da melhoria de aplicativos nativos, como o chat com vídeo no FaceTime, surgiu o iBooks, aplicativo que permite leitura e compra de livros digitais. Além disso, a multitarefa foi aplicada ao iOS sem diminuir a performance ou drenar a bateria (FERNANDES, 2013).

Mais de 200 novas funções foram adicionadas ao iOS 5.0 (FERNANDES, 2013), entre as mais comentadas estavam a Siri, software capaz de reconhecer e executar comandos por voz, e a possibilidade de integração com a iCloud, aplicativo que armazena arquivos na nuvem que podem ser acessados de qualquer dispositivo. Entre outras alterações, estão o aplicativo de lembretes, o histórico de notificações, entre outros. O iPad, em específico, passou a contar com gestos multitarefas, a divisão de teclado, espelhamento de tela, iMessage (envio e recebimento de mensagens via internet entre dispositivos Apple) e muitas outras melhorias.

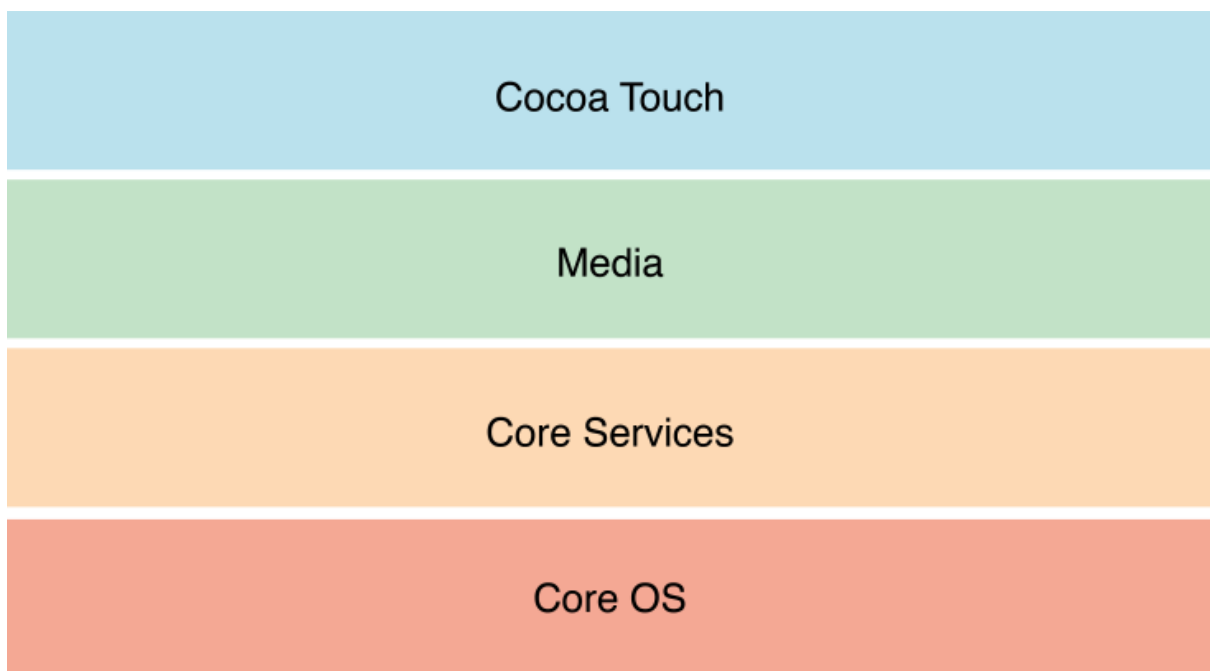
A versão 6.0 do iOS, de 2013, não ofereceu muitas mudanças e a empresa Apple não alcançou as expectativas de muitos de seus clientes. Houve a tentativa de retirada do Google Maps, substituído por um Mapa da Apple, mas a maioria dos usuários não ficaram satisfeitos, foram realizadas melhorias em alguns aplicativos como GameCenter, FaceTime e câmera.

2.1.2 Arquitetura do iOS

Segundo Marzullo (2012, p. 80), a arquitetura do sistema operacional iOS é baseada no kernel OSX, mas com algumas alterações, já que ele é direcionado para dispositivos móveis, enquanto o OSX é para computadores desktops e notebooks. A menor capacidade de processamento e de memória e a restrição do uso da bateria foram alguns dos cuidados que precisaram ser observados durante a adaptação do iOS.

A plataforma dos dispositivos móveis da Apple funciona como um intermediário entre hardware e aplicativos. Um conjunto de sistemas bem definidos permite abstrair o hardware, protegendo os aplicativos das mudanças dele. O iOS também pode ser pensado como camadas, mostradas na Figura 1 (APPLE INC., 2012).

Figura 1: Camadas do iOS



Fonte: APPLE INC.(2012)

A camada CocoaTouch é a de mais alto nível no iOS, ela define a infraestrutura básica da aplicação e o suporte para tecnologias fundamentais como multitarefa, notificações (APPLE INC., 2012), multitoque, componentes gráficos, localização, mapas e câmera. O

CocoaTouch oferece um conjunto de recursos responsáveis por responder ao toque do usuário, definindo como as aplicações devem ser desenvolvidas (MARZULLO, 2012).

Logo abaixo tem-se a camada Media, que contém as tecnologias de gráfico, áudio e vídeo que têm a finalidade de facilitar o desenvolvimento de aplicativos com boa qualidade visual e sonora (APPLE INC., 2012). Algumas das funcionalidades fornecidas são: animações, suporte a documentos PDF e a imagens JPEG, PNG, TIFF, etc., mixagem e gravação de áudio, entre outros. (MARZULLO, 2012).

Os serviços de sistema utilizados pelas aplicações estão na camada Core Services, grande parte do sistema é construída sobre ela (APPLE INC., 2012). Segundo Marzullo (2012), algumas funcionalidades apresentadas por essa camada são os utilitários para acesso à internet, caderno de endereços, banco de dados (CoreData, SQLite), acesso a arquivos, serviços de redes, entre outros.

Por fim, tem-se a camada de mais baixo nível, a Core OS, normalmente elas são acessadas por outros frameworks, que não as tecnologias utilizadas pelo desenvolvedor, porém quando há a necessidade de controlar explicitamente questões de segurança ou fazer comunicação com hardware externo, utilizam-se os frameworks dessa camada (APPLE INC., 2012). Como o iOS é uma variação do OSX, ele possui sua base também no BSD Linux, suportando serviço de sockets, processamento paralelo, gerenciamento de energia, segurança, sistemas de arquivos, etc. (MARZULLO, 2012).

2.2 OBJECTIVE-C

A linguagem de programação utilizada para desenvolvimento iOS é a Objective-C, lançada nos anos 80 por Brad J. Cox. A base dessa linguagem está na Small Talk-80 e ainda é uma extensão adicionada à linguagem C, que permite a manipulação e extensão de objetos (KOCHAN, s.d.). Após o lançamento do SDK, desenvolvedores puderam disponibilizar seus aplicativos na Apple Store, para que, então, a partir do iOS 2.0, distribuído no ano de 2008, os usuários conseguissem baixar apps. No primeiro semestre de 2013, foi alcançado o número de 50 bilhões de aplicativos baixados.

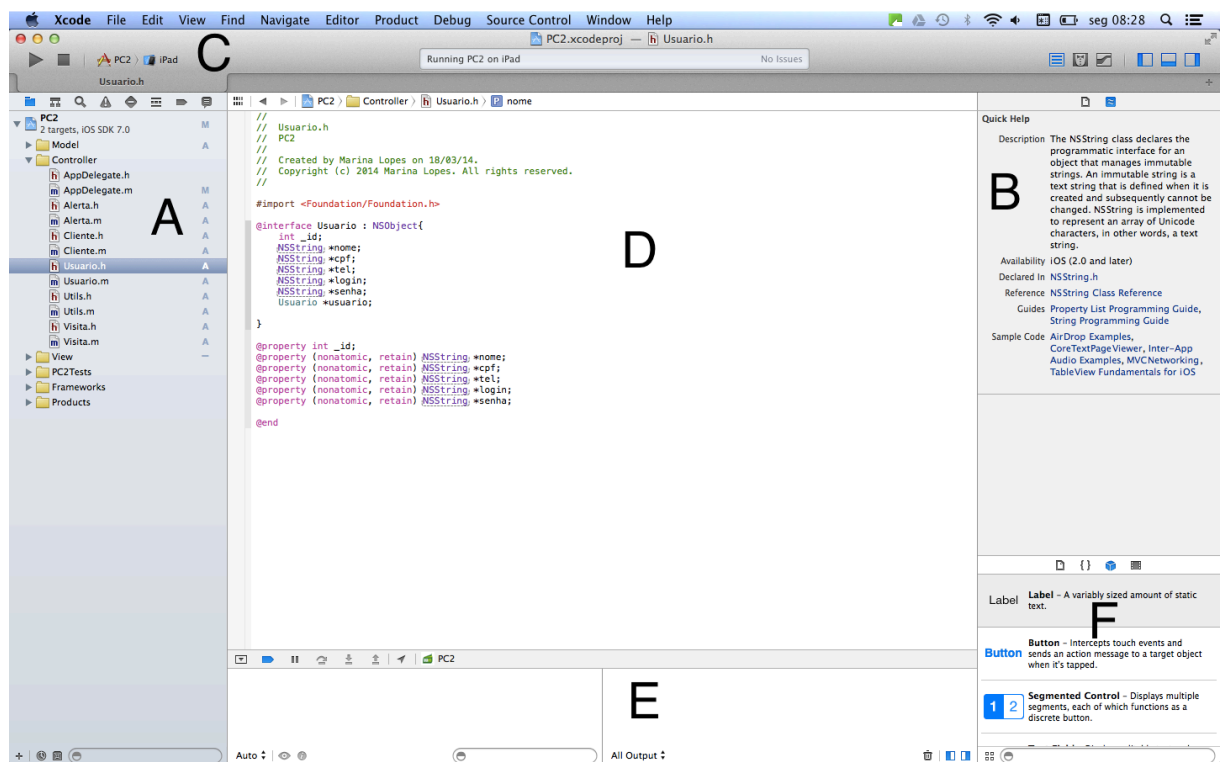
2.2.1 Plataforma de desenvolvimento XCode

A ferramenta de desenvolvimento de softwares para iOS é o XCode, ela pode ser instalada a partir da página oficial: [//developer.apple.com/XCode](http://developer.apple.com/XCode), apenas computadores com o sistema operacional OS X podem fazer a instalação. (LECHETA, 2012). A IDE (ambiente de desenvolvimento) XCode é um sistema abrangente, que fornece as alternativas necessárias

para que se desenvolva desde softwares simples até outros mais complexos. (MARZULLO, 2012).

O XCode possui um sistema de gerenciamento de grupos e arquivos, essa área fica ao lado esquerdo da IDE e permite a navegação entre os arquivos e a separação deles em grupos definidos pelo usuário, na Figura 2(A). Esse mecanismo oferece ao programador um meio de organização e de melhor visualização do projeto. (MARZULLO, 2012).

Figura 2: XCode



Fonte: do autor

A área de propriedades (B), localizada no canto direito da ferramenta, dispõe de informações sobre os arquivos como identificação, parâmetros de configuração, tipo, atributos, ajudas rápidas, entre outros. Além disso, apresenta diferentes tipos de configurações, entre elas a de componentes de tela muito utilizada.

A barra de ferramentas na parte superior da IDE, área (C), tem a função de escolher em qual dispositivo o aplicativo será executado, entre eles pode estar um dispositivo real ou um simulador de iPhone ou iPad em diversas versões. Para testes no próprio dispositivo é necessário que se tenha uma conta individual de desenvolvedor Apple que custa US\$99,00 ou a empresarial de US\$299,00. Ainda na barra de ferramentas pode-se optar por diferentes maneiras de visualizar o XCode e ainda de começar ou parar a execução do projeto em desenvolvimento.

O editor de texto, local onde se realiza a escrita do código dos softwares se localiza na parte central do XCode, Figura 2(D). Ele apresenta funcionalidades que proporcionam maior praticidade na codificação de aplicações, como atalhos, sugestões para completar o código, ocultação de partes do código, apontamento de erros e alertas, etc. (MARZULLO, 2012).

O XCode ainda oferece funcionalidades como depurador para ajudar a encontrar falhas, console, Figura 2(E), que relaciona todas as mensagens provenientes do código executado e também seus erros. Além disso, no canto inferior direito (F), encontram-se os componentes gráficos necessários para construir a interface com o usuário do sistema.

A cada atualização o XCode apresenta algum tipo de melhoria e está evoluindo junto ao sistema operacional iOS. Suas funcionalidades garantem a capacidade de ajudar e oferecer soluções aos programadores que precisam passar por um processo de adaptação a IDE até interagirem com maior fluidez entre os recursos.

2.2.2 Conceitos fundamentais para programação em Objective-C

Para iniciar a trabalhar com a programação para Objective-C é importante que já se tenha conhecimento sobre alguns conceitos e também se saiba como é a estrutura em que a linguagem funciona, esses assuntos serão abordados a seguir.

Os conhecimentos de programação orientada a objetos são indispensáveis para programadores de Objective-C, já que os conceitos de classe, objeto, encapsulamento, interface, herança, polimorfismo e tanto outros que envolvem a orientação a objetos estão presentes nessa programação. Trata-se da abstração de entidades do mundo real, com seus conjuntos de características e comportamentos dentro do mundo virtual, com as especificações das regras ditadas pelo contexto. (MARZULLO, 2012).

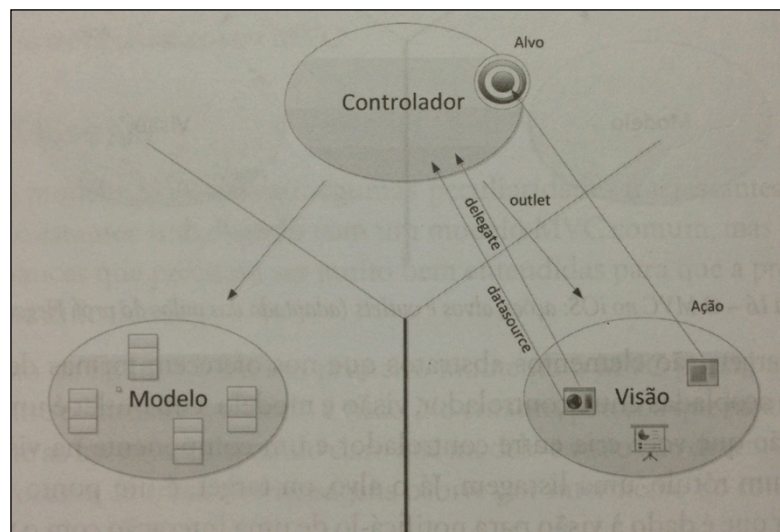
No desenvolvimento em Objective-C, Marzullo(2012) destaca que tudo que é desenvolvido no Objective-C utiliza-se o estilo arquitetural MVC (Model-View-Controller) que, segundo o autor, é um recurso que organiza os componentes e separa as responsabilidades. No MVC há uma separação semântica de componentes: modelo, controle e visão. No modelo, isola-se o domínio do problema, ou seja, é feita a representação das entidades; no controle, programa-se o fluxo de informações que circulam na aplicação; e, por fim, na visão, é feita a interface com o usuário.

O MVC na programação para iOS possui particularidades que intensificam o conceito de que modelo e visão não devem se conhecer. Para que isso ocorra há dois elementos abstratos essenciais, que são:

- **outlet**, responsável por ser um canal de comunicação entre controlador e visão, o qual pode ser um botão, uma listagem;
- **target**, também conhecido como alvo, que “é um ponto fixado no controlador que é dado à visão para notificá-lo de uma interação com o usuário. Neste mecanismo é como se o controlador criasse um alvo em si próprio e dissesse para a visão mirá-lo quando quisesse enviar uma mensagem.” (MARZULLO, 2012, p.37).

A partir desses dois elementos, *outlet* e *target*, é feito o sincronismo entre a visão e o controlador, pode-se também ter o auxílio do padrão de projeto *delegate*, baseado em delegar a outros objetos as suas tarefas, há, então, a chamada inversão de responsabilidade, que garante transparência entre o controlador e a visão. O controlador deve ser o *delegate* da visão e sua implementação é feita a partir de um protocolo que exige a codificação de alguns métodos, a Figura 3 ajuda a visualizar esta sistemática

Figura 3: MVC no Objective-C



Fonte: MARZULLO, 2012

Quando o caminho inverso é realizado, ou seja, as informações do modelo precisam ser repassadas para a visão, o protocolo usado é o *data source*. O controlador será usado para implementar os métodos estabelecidos pelo protocolo *data source*, mesmo que as informações sejam provenientes do modelo.

2.2.3 Características de uso recorrente no Objective-C

A linguagem de programação Objective-C tem uma sintaxe que causa estranheza a quem começa a trabalhar com ela, o código parece ser muito extenso, porém o que se tem é uma tentativa de fazer com que o código seja mais fácil de ser lido e compreendido. A intenção é ser intuitivo, e a partir do momento em que já se tem uma familiaridade com a linguagem, ela apresenta essa característica.

Estrutura das Classes

Quando qualquer projeto é criado, nota-se a presença de dois tipos diferentes de arquivos, um com a extensão *.h* denominado header, o cabeçalho da classe, onde encontram-se declarações de variáveis, métodos e outros recursos, já o outro tem a extensão *.m* onde são feitas as implementações do que foi declarado no header (GARREFA, s.d.).

Para criação de uma classe, por exemplo, serão sempre criados dois arquivos, o de extensão *.h* apresenta-se como uma interface, na situação a seguir mostra-se, como exemplo, o caso de um arquivo chamado *Cliente.h*,

Quadro 1: Arquivo header

```
1. #import<Foundation/Foundation.h>
2. @interfaceCliente : NSObject
3. {
4.     int _id;
5.     NSString *nome;
6.     NSString *cpf;
7. }
8. @propertyint _id;
9. @property (nonatomic) NSString
   *nome;
10. @property (nonatomic) NSString
   *cpf;
11. @end
```

Fonte: do autor

e o de extensão *.m* apresenta-se como implementação. Abaixo o arquivo *Cliente.m*, implementação da classe *Cliente.h*:

Quadro 2: Arquivo de implementação

```

1. #import "Cliente.h"
2. @implementationCliente
3. @synthesize _id,nome, cpf;
4. @end

```

Fonte: do autor

Analisando o código do arquivo *Cliente.h* verifica-se o início e fim determinados pelas diretivas `@interface` e `@end`, respectivamente e entre as chaves são definidos os atributos da classe com seus tipos. É possível notar também a existência do asterisco antes de cada variável, que identifica um ponteiro, toda declaração de objeto no Objective-C é feita dessa forma.

Depois das chaves a diretiva `@property` declara aqueles atributos que deverão ser expostos por métodos de acesso. O atributo `nonatomic`, da diretiva `@property`, disposto entre parênteses é opcional e significa que a propriedade pode ser acessada sem nenhum controle formal. Existem ainda outras propriedades como a `strong` que indica uma propriedade com forte ligação ao seu objeto ou a `weak` que caracteriza um relacionamento fraco, dentre outras. (MARZULLO, 2012). No mesmo local onde estão as diretivas `@property` poderiam estar também as declarações dos métodos caso se enquadrasse no contexto da classe.

No arquivo *Cliente.m* tem-se definições criadas pela codificação do arquivo *.h*, quando a diretiva `@property` é usada, obriga que se implemente no arquivo *.m* o método de acesso por meio da diretiva `@synthesize`. É importante notar que caso existir métodos declarados no arquivo cabeçalho eles devem estar com suas lógicas codificadas no arquivo de implementação.

Quando deseja-se criar uma nova tela é preciso gerar uma classe que tenha o tipo `UIViewController`, essa classe se configura como controladora no padrão MVC, e como a tela que origina a interface gráfica e trata os eventos correspondentes às ações dos usuários. (MARZULLO, 2012). Nessa situação são criados 3 arquivos, o *.h* e o *.m* já referidos e ainda o *.xib*, esse adicional contém o Interface Builder (IB), construtor da interface gráfica, e o local onde os componentes gráficos serão utilizados.

Instâncias e métodos

Na orientação a objetos toda vez que é feita a instanciação de um objeto se cria na memória uma cópia estrutural da classe que será completada com as informações de seu

contexto em cada um de seus atributos, é dessa forma que ocorre na linguagem de programação Objective-C. (MARZULLO, 2012).

A mudança está apenas na sintaxe que se difere do que é visto normalmente. No trecho de código abaixo se tem um método chamado `getClientes` que exige o retorno de um `NSMutableArray` (uma lista encadeada de objetos), para criar a nova instância `db` de `ClienteDB` foi utilizado o `alloc`, e logo após o `init` que indica o término da inicialização do objeto. (LECHETA, 2012).

Abaixo da inicialização é demonstrada uma chamada de método, entre colchetes o objeto e o método que devem ser chamados separados por um espaço, e depois os dois pontos que indicam uma passagem de parâmetro. Por fim o retorno do `NSMutableArray`.

Quadro 3: Estrutura de um método no Objective-C

```

1. + (NSMutableArray *) getClientes {
2.     [...]
3.     ClienteDB *db = [[ClienteDB alloc] init];
4.     [db abrir:@"clientes"];
5.     [...]
6.     return clientes;
7. }
```

Fonte: do autor

Um método com parâmetro tem a sintaxe demonstrada abaixo, o nome do método é acompanhado por dois pontos, o tipo do parâmetro é colocado entre parênteses seguido do seu nome:

Quadro 4: Método com parâmetro

```

1. -(void) abrir: (NSString *)nomeBanco{
2.     [...]
3. }
```

Fonte: do autor

Controle de exceções

Para tratar exceções o Objective-C tem soluções simples e se assemelha ao Java. Possui as seguintes diretivas (MARZULLO, 2012):

- `@try`: utilizado para agrupar o trecho de código que pode lançar a exceção.
- `@catch`: trata a exceção quando ela ocorre no trecho englobado pelo `@try`.
- `@throw`: lança uma exceção, que se torna também um objeto.

- `@finally`: os códigos escritos nessa diretiva são sempre executados ocorrendo ou não uma exceção e seu tratamento.

Diretiva *import*

A diretiva `import` é utilizada toda vez que se deseja utilizar outra classe no projeto, ela deve referenciar o arquivo de cabeçalho da classe pretendida. A diretiva `import` pode ser colocada tanto no arquivo *header* quanto no arquivo de implementação.

2.3 PERSISTÊNCIA DE DADOS

A persistência de dados no iOS pode ser feita a partir de três mecanismos: arquivos, banco de dados SQLite e ainda o framework Core Data. Para a persistência por meio de arquivos, deve-se dominar a manipulação de arquivos por meio da linguagem Objective-C, a criação de arquivos para armazenamento de dados é feita em “Diretório de Documentos”, área reservada que não permite interferências da aplicação no sistema operacional. (MARZULLO, 2012).

Persistir dados por meio do Framework Core Data garante as funcionalidades principais encontradas em um SGBD. As entidades ou objetos criados pelo programador serão gerenciadas por um objeto controlador, o contexto, responsável por controlar ações de inserção, exclusão, alteração e consulta por meio de ferramentas de controle. Além disso, o contexto possui regras capazes de proteger os objetos gerenciados, evitando inconsistência enquanto as informações não são salvas. (MARZULLO, 2012).

O banco de dados SQLite, biblioteca desenvolvida em C e que possui um banco de dados SQL embutido tem suporte nativo no iOS. Ele conta com os seguintes tipos de dados: para números inteiros – INTEGER; valores em ponto flutuante – REAL; texto – TEXT; dados binários – BLOB; e para valores nulo – NULL. Não existe um tipo de dado específico para data. O SQLite não controla acesso concorrente e não trabalha com o armazenamento de imagem, áudio e vídeo. (SAMPAIO, 2012).

Por padrão o SQLite vem instalado no sistema operacional MAC OS X, e a sua biblioteca vem disponível no XCode, para usá-la é preciso adicioná-la ao projeto na seção *Link Binary with Libraries*, a biblioteca chama-se *libsqlite3.dylib*, dessa forma é possível importar a biblioteca toda vez que for preciso utilizar de suas funções, assim:

Quadro5: Código para importar biblioteca do SQLite

```
1. #import "/usr/include/sqlite3.h"
```

Fonte: do autor

2.4 TECNOLOGIAS RELACIONADAS

Além do sistema operacional iOS existem outros sistemas operacionais para dispositivos móveis, entre as mais notáveis hoje estão: Android e Windows Phone. As três tecnologias passam por mudanças em curtos intervalos de tempo sempre objetivando melhoria de seus sistemas para manter e conquistar clientes.

O Android é o sistema operacional móvel da empresa Google e atualmente a mais utilizada entre os dispositivos móveis no mundo. O sistema é licenciável, dessa forma qualquer fabricante que desejar produzir o hardware para o Android, e cumprir os pré-requisitos determinados pela Google, pode produzir os aparelhos. (KIRCOVE, 2013). O desenvolvimento de aplicativos para Android é feito em Java com a utilização de bibliotecas desenvolvidas pela Google. (SILVA, F., s.d.).

Lançado em 2010, o Windows Phone, da Microsoft, surgiu como forte concorrente do Android e iOS, com foco na personalização de cada dispositivo a partir da opção do usuário. O desenvolvimento para essa linguagem é feita em C# ou Visual Basic, e ainda, para criação da interface, é utilizado o XAML (Extensible Application Markup Language), baseada em XML e criada pela Microsoft. (SAFATLI, s.d.).

Segundo dados divulgados pelo IDC, o Android é o sistema operacional móvel mais utilizada no Brasil. Nos três últimos meses de 2013 ele estava instalado em 88,73% dos smartphones do país, logo em seguida o Windows Phone, com participação de 6% no mercado, e por fim o iOS com 4,7%. (OLHAR DIGITAL, 2014).

3 ESTUDO DE CASO

O trabalho proposto tem como foco abordar a linguagem de programação Objective-C, a partir de um estudo de caso feito sobre a venda de um insumo. Esta seção apresenta o produto, a empresa e uma descrição dos processos realizados atualmente, além disso, descreve o estudo de caso desenvolvido com o objetivo de resolver parcialmente os problemas enfrentados pela empresa atualmente.

3.1 CONTEXTO EMPRESARIAL

O produto Gigamix é um estimulante vegetal comercializado pela empresa Basalto São Cristóvão (BSC), localizada em Paraí - RS. A BSC existe desde o ano de 1979 e tem como foco a industrialização de rochas para a composição de pisos ornamentais e a comercialização de vidro prensado. Além disso, o Gigamix tornou-se um ramo da empresa que vem se destacando no setor agrícola, já que está investindo num produto que oferece aumento de produtividade no campo, com sustentabilidade.

Desde o ano de 2002, foram realizados estudos no Vale do Caí e na Serra Gaúcha, conjuntamente a análises laboratoriais e ensaios de campo. O objetivo era alcançar um resultado satisfatório para a produção rural. Em 2005, foi lançada a primeira versão do Gigamix, destinada à cultura de pequeno cunho comercial, e, no ano seguinte, o produto foi produzido em escala industrial com registro no INPI (Instituto Nacional de Produto Industrial).

No ano de 2008, investiu-se em tecnologias para produção do insumo, qualificando o processo de produção. Dessa forma tornou-se possível expandir a comercialização do Gigamix, conquistando clientes responsáveis por grandes produções agrícolas, em cidades do Sul, Nordeste e Centro Oeste. Seguidamente, em 2010, a empresa conquistou a certificação da ECOCERT, uma garantia aos clientes de que o produto possui credibilidade.

Após a certificação, com o crescimento das vendas, foi necessário ampliar o número de colaboradores. Atualmente, o Grupo BSC-Gigamix conta com uma estrutura de quatro funcionários, dez vendedores, cinco revendedores e dois engenheiros agrônomos. Muitos moram na região dos pontos de vendas, outros fazem viagens para as localidades onde o Gigamix está sendo comercializado.

Os clientes do insumo são trabalhadores desde a agricultura familiar até grandes produtores da área agrícola e encontram no produto muita versatilidade, já que se adapta em múltiplas culturas, tipos de solo e climas. A fácil adaptação do Gigamix permite que ele esteja

espalhado pelo Brasil, podendo ser encontrado nos estados do Rio Grande do Sul, de Santa Catarina, do Mato Grosso, de Goiás, do Distrito Federal, da Bahia, do Piauí e do Maranhão.

O estimulante vegetal pode ser utilizado em todos os tipos de cultura, como frutíferas, hortaliças, olerícolas, flores, leguminosas e grãos em geral; agregado a isso, realiza também tratamento de sementes, entre outros. Cada cultura tem suas especificações de utilização, como consta na arte da embalagem do produto, que pode ser visualizada no anexo A.

O benefício destaque do produto é o aumento de produtividade com sustentabilidade e segurança alimentar, complementar a isso, há a diminuição do uso de agroquímicos, maior vida pós-colheita aos produtos, resistência a eventos climáticos e pragas, etc. Ele age no fortalecimento das plantas, promovendo nutrição equilibrada, sanidade, desenvolvimento do sistema radicular, entre outros, podendo alcançar diferentes objetivos, dependendo da cultura em que é utilizado.

Inicialmente, quando o Gigamix ainda estava entrando no mercado, eram fornecidas gratuitamente quantidades do produto para testes, assim os clientes podiam conhecer o produto, acompanhar a ação dele com suporte dos especialistas do insumo. Com os bons resultados alcançados e com o passar do tempo, o Gigamix passou a se tornar conhecido a partir do marketing boca-a-boca.

Nos dias de hoje, aqueles produtores que desejam testar o produto devem comprá-lo e possuem assistência dos especialistas da mesma forma. O marketing boca-a-boca ainda é a principal forma de promover o insumo, e os possíveis clientes que entram em contato com a empresa são conquistados a partir da força de vendas. No anexo B, visualiza-se uma lista de contatos dos clientes ou possíveis clientes, escritos à mão, para o trabalho de conquista e manutenção das vendas.

As vendas são realizadas por vendedores ou revendedores e, ambos os casos, acontecem da mesma forma. Quando uma venda é concretizada, o vendedor envia um e-mail à administração da empresa com uma planilha de pedidos, como mostra o anexo C ou apenas a descrição do pedido no corpo do e-mail, sem detalhes.

As informações que constam na planilha de pedido são os dados do cliente, como nome, contato e endereço; a forma de pagamento do pedido e o prazo de entrega, além do endereço de cobrança e de entrega, que podem ou não diferirem um do outro. Deve constar também a quantidade de Gigamix que está sendo vendida, referenciando a unidade de medida: sacos de 1kg ou potes de 175g e o preço unitário e total.

Após o processamento do pedido, é feita uma nota fiscal eletrônica (NF-e), o estoque é conferido a partir de uma ficha de controle e, se necessário, solicita-se que se produza mais.

Com o produto pronto, ele é encaminhado junto à NF ao destino do comprador, via veículos próprios ou transportadora.

Um trabalho importante realizado pelo Grupo BSC-Gigamix refere-se ao acompanhamento da aplicação do insumo nas lavouras do cliente. Existem alguns tipos de visitas diferentes pontuadas a seguir.

A primeira refere-se ao apoio dos clientes no momento em que ele está testando o produto. Apesar de a empresa ter convicção do progresso que o estimulante vegetal fornece às plantas, alguns produtores ainda precisam ser convencidos, visualizando em sua própria terra a ação do insumo.

Para realizar o teste, o profissional deve dar orientações precisas de como utilizar o produto e é decidida, junto ao cliente, qual a parcela da lavoura será testada. A partir do momento que o estimulante vegetal começa a mostrar seus efeitos, antes mesmo da colheita, que se pode ter a visão quantitativa da melhora, é de extrema importância a visita de um profissional ao cliente que aponte o desenvolvimento que o Gigamix está oferecendo.

O segundo tipo de visita acontece apenas para acompanhamento da produção daqueles que já estão utilizando o insumo em suas culturas. Nessas ocasiões, o colaborador Gigamix vai poder perceber se o produtor está ou não satisfeito, conhecer as suas colocações, e poderá, então, tomar decisões em relação a isso.

Por fim, existem as visitas técnicas, as quais acontecem quando o produtor tem algum problema relacionado ao Gigamix e precisa do auxílio de um profissional. Nesses casos, é preciso ir até a propriedade, analisar o problema e definir uma solução passível para ele. Em alguns casos, resolve-se facilmente, em outros, são necessários estudos específicos, como experimentos, análise de como foi feita a aplicação, entre outros.

Para todos os tipos de visita é importante que os superiores da empresa tenham um parecer sobre o que aconteceu e quais os procedimentos que foram realizados. Para isso, os colaboradores fazem relatórios, não há um padrão nem uma organização que permita controlar o histórico de visitas àquela determinada propriedade. Ou ainda é realizada uma reunião, muitas vezes longa, que serve para relatar as visitas que foram feitas e concluir ações futuras.

O processo atual em que o Grupo BSC-Gigamix opera tem encontrado algumas dificuldades citadas a seguir:

- Inexistência de um cadastro de clientes;
- Falta de um histórico sobre vendas, pedidos e visitas vinculados a determinado cliente;
- Relatórios sem padronização;

- Necessidade de longas reuniões;
- Dificuldade de controle sobre o que foi realizado durante o dia pelos colaboradores;
- Pedidos feitos em planilhas ou e-mails de forma pouco prática e pouco organizada;
- Necessidade de uso de vários dispositivos, como GPS, câmera digital, notebook, etc;
- Utilização de documentos impressos para consulta;
- Alto número de viagens.

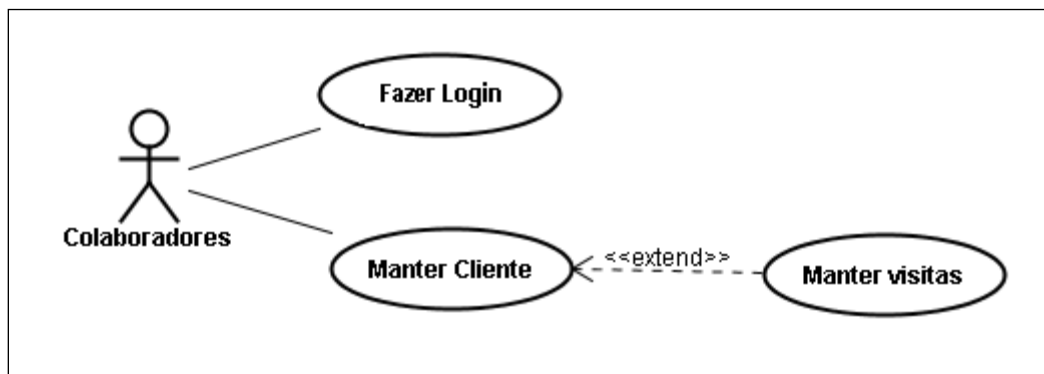
3.2 MODELAGEM DO SISTEMA

O sistema realizado como estudo de caso foi desenvolvido para iPad, tablet com o sistema operacional iOS. Ele faz a manutenção dos dados de clientes e de suas visitas a partir de acesso via login e senha, com o objetivo de resolver parcialmente os problemas da empresa BSC-Gigamix.

3.2.1 Diagrama de casos de uso

O diagrama de casos de uso tem como objetivo demonstrar as funcionalidades (casos de uso) que o sistema proposto vai possuir e quem irá interagir (atores) com ele. A partir da análise dos requisitos funcionais tem-se o diagrama de casos de uso a seguir (Figura 4):

Figura 4: Diagrama de casos de uso



Fonte: do autor

O sistema possui o acesso de colaboradores da equipe Gigamix, o acesso é feito por login e senha previamente cadastrados no sistema. A partir disso o colaborador tem acesso a duas funções do sistema, a principal delas é o acesso a manutenção de clientes, ou seja, é possível acessar a lista de clientes e suas informações, além de poder criar, modificar, ou deletar um cliente.

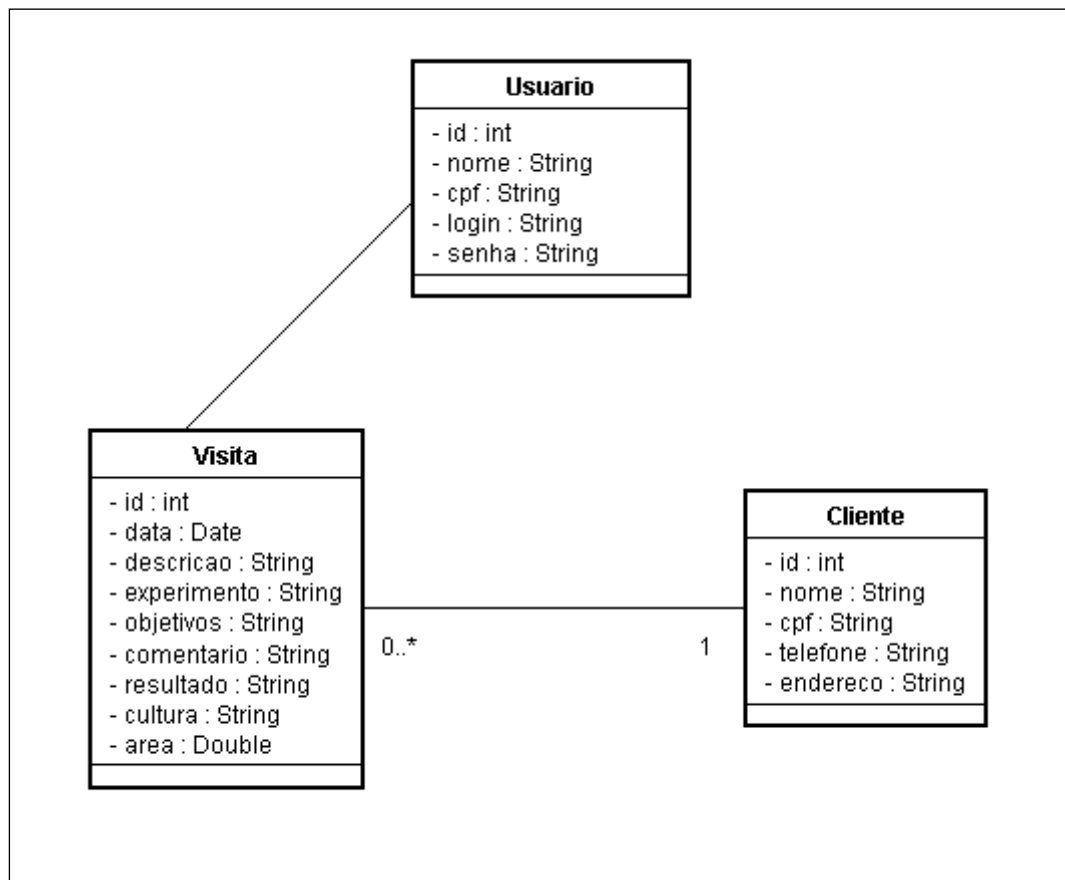
A manutenção também acontece no caso das visitas que são acessadas a partir do momento em que um cliente é selecionado. Toda vez que um colaborador vai até a

propriedade do cliente, ele descreve a visita, para que se possa manter um histórico do que já foi visto, constatado e executado na propriedade.

3.2.2 Diagrama de classes

O diagrama de classes visa demonstrar a organização das classes do sistema. Nesse caso (Figura 5) tem-se um diagrama com foco na persistência de dados, ou seja, apresentam-se as classes com seus atributos, relacionamentos e multiplicidades.

Figura 5: Diagrama de classes



Fonte: do autor

A classe “Cliente” possui os atributos referentes aos clientes da empresa, como nome, telefone e endereço. A classe “Usuario” possui atributos referentes aos dados dos usuários, login e senha para acesso ao sistema. A classe **Visita** possui obrigatoriamente um atributo cliente, entre outros atributos, que servem para descrever todas as visitas técnicas realizadas na propriedade como data, objetivos e resultados.

3.3 DESENVOLVIMENTO DO SISTEMA

As seções a seguir abordam os aspectos relevantes da programação realizada no estudo de caso concretizado.

3.3.1 Navegação entre telas

A ação de navegar entre telas diferentes em aplicativos para dispositivos móveis é algo muito recorrente. Devido a esse fato existem algumas formas diferentes na linguagem de programação Objective-C que podem ser escolhidas pelo programador para que a aplicação tenha a característica de navegação aplicada de forma coerente e adequada.

No estudo de caso elaborado foi utilizada a `UINavigationController`, nessa forma de navegação uma barra na parte superior da tela, apresenta, normalmente, um título e botões. Ela funciona como uma pilha de telas e facilita a navegação de uma tela para outra. (LECHETA, 2012). O botão comumente usado é o voltar que pode ser utilizado com o `UINavigationController`, e se necessário passar por algumas configurações.

O trecho de código abaixo foi colocado no arquivo `AppDelegate.m`, ele estabelece o sistema de `UINavigationController` e seta o controlador `Inicio` como raiz, ou seja, como a primeira tela a ser aberta quando o aplicativo é iniciado. Ainda, na última linha, uma configuração adicional que muda a cor de fundo da barra de navegação a deixando verde.

Quadro 6: Implementação da UINavigationController

```
1. Inicio *i = [[Inicio alloc] init];
2. UINavigationController *nav = [[UINavigationController alloc]
   initWithRootViewController: i];
3. self.window.rootViewController = nav;
```

Fonte: do autor

Entre as muitas configurações que a `UINavigationController` pode passar o `UIBarButtonItem` é uma delas, um botão utilizado para ser colocado na barra de navegação, abaixo uma utilização dele no estudo de caso:

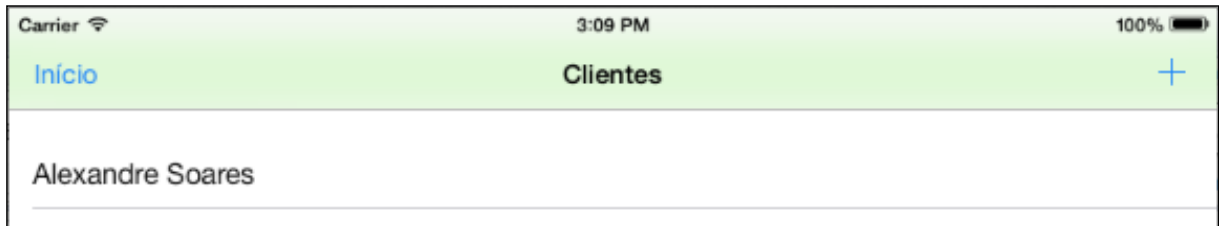
Quadro 7: Implementação de UIBarButtonItem

```
1. UIBarButtonItem *addButton = [[UIBarButtonItem alloc]
   initWithBarButtonSystemItem:UIBarButtonSystemItemAddtarget:self
   action:@selector(abrirForm:)];
2. self.navigationItem.rightBarButtonItem = addButton;
```

Fonte: do autor

Nessa situação foi feita a inicialização do `UIBarButtonItem` e atribuído a ele o método `abrirForm` que será acionado quando houver um toque no botão. Na última linha a propriedade faz com que o botão fique localizado na direita da barra de navegação, como é possível notar na figura abaixo:

Figura 6: Tela com UINavigationController



Fonte: do autor

3.3.2 Tabelas

Para utilização de listagens em tabelas, no Objective-C, é necessário o uso da classe `TableView`, que possui protocolos de *delegate* e *datasource*, `UITableViewDelegate` e `UITableViewDataSource` respectivamente. No estudo de caso implementado esta classe foi empregada duas vezes para a listagem de clientes e também de visitas.

O primeiro passo é notar que na biblioteca de objetos no Xcode, onde ficam os componentes gráficos, encontra-se a `TableView` que deve ser arrastada à tela, no arquivo `ClientesController.xib`, onde se deseja estruturar a tabela. No arquivo `ClienteController.h` deverá aparecer a implementação do `UITableViewDelegate` e do `UITableViewDataSource`, que determinarão ações que poderão ser implementadas, além disso ainda é necessário a declaração da `TableView`, que pode ser observado abaixo:

Quadro 8: Implementação de `UITableViewDelegate` e `UITableViewDataSource`

```

1. @interface ClientesController : UIViewController<UITableViewDataSource,
   UITableViewDelegate>{
2. }
3. @property (nonatomic, retain) IBOutlet UITableView *tableView;
4. [...]

```

Fonte: do autor

Com o protocolo `UITableViewDelegate` é possível trabalhar com os métodos seguintes:

- `willSelectRowAtIndexPath`: a ação codificada nesse método é executada quando uma célula da tabela está para ser selecionada, ela ocorre antes da seleção.
- `didSelectRowAtIndexPath`: a ação codificada nesse método é executada no momento em que uma célula da tabela é selecionada.
- `willDeselectRowAtIndexPath`: a ação codificada nesse método é executada quando uma célula da tabela está para ser desmarcada, ela ocorre antes da desmarcação.
- `didDeselectRowAtIndexPath`: a ação codificada nesse método é executada no momento em que uma célula da tabela é desmarcada.

No estudo de caso realizado foi implementado o método `didSelectRowAtIndexPath` no arquivo *ClientesController.m*, acionado no clique de uma célula, o qual pode ser observado a seguir:

Quadro 9: Método para toque em célula da tabela

```

1. -(void)tableView:(UITableView
   *)tableViewdidSelectRowAtIndexPath:(NSIndexPath *)indexPath{
2.     NSInteger linha = indexPath.row;
3.     Cliente *c = [_clientesobjectAtIndex:linha];
4.     DetalhesClienteController *detalhes =
       [[DetalhesClienteController alloc]init];
5.     detalhes.cliente = c;
6.     [self.navigationController pushViewController:detalhes
       animated:YES];
7. }

```

Fonte: do autor

No momento em que o usuário selecionar uma célula da tabela, outra tela é aberta para mostrar as informações do cliente selecionado, o número da linha selecionado é recuperado a partir do código: `indexPath.row`, o objetivo é recuperar o objeto relacionado ao cliente que foi selecionado, dessa forma: `objectAtIndex:linha`.

O controlador da tela *DetalhesClienteController* é instanciado para que o objeto cliente dessa classe seja inicializado. Finalmente a tela é chamada e aberta com as informações do cliente selecionado pelo usuário.

No protocolo *UITableViewDataSource* os métodos são:

- `cellForRowAtIndexPath`: obrigatoriamente esse método deve ser implementado, é nele que é feita a renderização das células da tabela.
- `numberOfRowsInSection`: método obrigatório e que deve retornar o número de linhas que a tabela deve ter.

- `numberOfSectionsInTableView`: apenas para quando a tabela possui diversas seções, esse método deve retornar o número de seções da tabela.

Os dois primeiros métodos citados acima foram implementados no estudo de caso, o primeiro, relativo a renderização das células é mais complexo. As linhas dele, codificadas abaixo, funcionam como procedimento padrão, elas servem para que as referências de uma célula sejam recuperadas e possam ser reaproveitadas enquanto existirem registros no banco de dados.

Quadro 10: Método para reaproveitar células

```

1. -(UITableViewCell *)tableView:(UITableView
2.   *)tableViewcellForRowAtIndexPath:(NSIndexPath *)indexPath{
3.     static NSString *CellIdentifier = @"Cell";
4.     UITableViewCell *cell =
5.       [self.tableViewdequeueReusableCellWithIdentifier:CellIdentifier];
6.     if (cell == nil){
7.       cell=[[UITableViewCellalloc]
8.         initWithStyle:UITableViewCellStyleSubtitlereuseIdentifier:CellIdentifier];
9.     }
10.    [...]
11. }

```

Fonte: do autor

O método que retorna o número de linhas que a tabela deve ter é bem mais simples no estudo de caso implementado, basta que se retorne a quantidade de clientes existentes na lista `NSMutableArray` chamada de `clientes`.

Quadro 11: Método para retorno do número de linhas da tabela

```

1. -(NSInteger) tableView:(UITableView
2.   *)tableViewnumberOfRowsInSection:(NSInteger)section{
3.     return [self.clientes count];
4. }

```

Fonte: do autor

Para montar uma tabela de forma adequada é indispensável que se tenham informações de como ela funciona e quais métodos devem ser implementados, para isso deve-se ter o domínio dos objetos, do `NSMutableArray` e ainda dos protocolos implementados pela classe em questão.

3.3.3 Persistência de dados

A criação do banco de dados SQLite pode ser feita via programação. Para manter o projeto organizado é interessante haver uma classe responsável por abrir e fechar a base de dados. No estudo de caso a classe *SQLiteHelper* tem essa funcionalidade e serve de apoio para as outras classes que precisam manipular os dados.

Para a criação da base de dados pode-se começar implementando um método para descobrir o caminho do arquivo onde vão ser salvas as informações. O método recebe uma *string* com o nome do arquivo e, então, retorna o caminho do mesmo, como é mostrado a seguir.

Quadro 12: Método para retornar caminho do arquivo

```

1. - (NSString *)caminhoDoArquivo:(NSString *)nomeBanco {
2.     NSString *db = [NSString stringWithFormat:@"%%.sqlite3",
3.         nomeBanco];
4.     NSArray *paths =
5.         NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
6.         NSUserDomainMask, YES);
7.     NSString *documentsDirectory = [paths objectAtIndex:0];
8.     NSString *caminhoArquivo =
9.         [documentsDirectorystringByAppendingPathComponent:db];
10.    return caminhoArquivo;
11. }

```

Fonte: do autor

No estudo de caso o banco de dados é chamado de *clientes*, dessa forma o arquivo terá o nome de *clientes.sqlite3*, a string com esse nome é armazenada na variável *db*, operação feita na primeira linha do método. Já a variável *paths* é responsável por guardar o caminho do arquivo, utilizando a função *NSSearchPathForDirectoriesInDomains*. Por fim, tem-se o retorno da variável que possui o caminho completo de onde o banco de dados é armazenado, chamada de *caminhoArquivo*.

Criar o banco de dados com o nome desejado é uma parte importante do projeto. No Quadro 11 apresenta-se a situação do estudo de caso, o método se comporta de forma que se o banco de dados não existe ele é criado, senão ele é aberto para estar aguardando as operações que podem ser realizadas, isso é feito a partir da função *sqlite3_open*, seus parâmetros são: nome do arquivo e ponteiro para o objeto *sqlite3*, chamado *bancoDeDados*.

Quadro 13: Método para abrir banco de dados

```

1. -(void) abrir:(NSString *)nomeBanco{
2.     int result = sqlite3_open([[self caminhoDoArquivo:nomeBanco]
   UTF8String], &bancoDeDados);
3.     if(result == SQLITE_OK){
4.         [...]
5.     }else{
6.         [...]
7.     }

```

Fonte: do autor

Para otimizar o método de abertura do banco de dados, é interessante que se chame outro método responsável por criar as tabelas. A classe `SQLiteHelper` funciona como uma superclasse que executa o SQL retornado por suas subclasses. A organização dos arquivos foi feita de tal forma que cada tabela tem uma classe específica, que cuidará não só da criação das tabelas, mas também do fluxo de dados, caso tratado com detalhes mais adiante.

Na prática a tabela é criada quando o método consegue abrir/criar o banco de dados, dessa forma o código dessa ação é colocado dentro da condicional `if`, mostrada no Quadro 11, e executa o código demonstrado a seguir:

Quadro 14: Método para criação de tabela no banco de dados

```

1. If (result == SQLITE_OK) {
2.     NSString *sql = [self getSQLCreate];
3.     char *erroMsg;
4.     int resultado = sqlite3_exec (bancoDeDados, [sql UTF8String], NULL,
   NULL, &erroMsg);
5. }

```

Fonte: do autor

A função `getSQLCreate` deve ser implementada nas subclasses, ela é responsável pelo recebimento do SQL de criação da tabela desejada, a função `sqlite3_exec` cria a tabela, caso ela ainda não exista, a partir da execução desse SQL. A superclasse `SQLiteHelper` serve de apoio às suas subclasses, nesse caso foram criadas basicamente três subclasses para que cada uma representasse uma das tabelas do banco de dados: `ClienteDB`, `VisitaDB` e `UsuarioDB`.

O arquivo de cabeçalho de cada uma das subclasses deve importar o arquivo `SQLiteHelper.h` e também herdar a classe `SQLiteHelper`. Cada uma delas deve ter os métodos responsáveis pelas manutenções de fluxo de dados desejáveis, essas manutenções são

entendidas como adicionar, deletar, editar ou ainda consultar dados. A subclasse `ClienteDB` que possui todas as operações de banco de dados será detalhada a seguir.

O método que precisa ser implementado primeiro é aquele que retorna uma `NSString` que contém o SQL de criação da tabela no banco de dados, chamado de `getSQLCreate`. Tal SQL só será executado caso a tabela que se tem intenção de criar ainda não exista, condição imposta pelo próprio código SQL, como pode ser visto no Quadro 13. Nesse caso a tabela cliente é criada, com cinco atributos.

Quadro 15: Trecho de código com o SQL de criação de tabela

```
1. NSString *sql = @"create table if not exists cliente (_id integer primary
   key autoincrement, nome text, cpf text, tel text, desc text, endereco
   text);";
```

Fonte: do autor

Para os métodos que seguem são necessárias duas telas, uma é a que mostra a lista de clientes existentes no banco de dados, e outra é referente ao formulário que deve ser usado no momento de inclusão e também de edição de dados relativos à tabela de clientes. A inclusão e edição são feitas no mesmo método, isso é possível através do código SQL, que é escrito dessa forma:

Quadro 16: Trecho de código com o SQL de inserção na tabela

```
1. char *sql = "insert or replace into cliente (_id, nome, cpf, tel, desc,
   endereco) VALUES (?, ?, ?, ?, ?, ?);";
```

Fonte: do autor

Os pontos de interrogação aguardam pelos valores correspondentes de cada atributo, que são passados após o uso da função `sqlite3_prepare_v2` responsável por criar o *statement* que será executado. É interessante notar que uma variável do tipo `sqlite3_stmt` é criada para guardar todas as informações produzidas na construção e execução do SQL, pode ser observado a seguir:

Quadro 17: Variável stmt

```
1. sqlite3_stmt *stmt;
2. int resultado = sqlite3_prepare_v2(bancoDeDados, sql, -
   1, &stmt, nil);
```

Fonte: do autor

Outro ponto chave, para que se possa fazer tanto a edição quanto a adição no mesmo método é um teste feito para saber se o campo `_id` recebido é nulo ou não, já que ele é auto incrementado e não informado pelo usuário. Caso ele seja maior que zero significa que já existe no banco, ou seja, será feita a edição e existe a necessidade de informar qual o `_id` do registro em questão, porém se ele não for maior que zero significa que ainda não existe e nesse caso trata-se de uma inserção.

No Quadro 15 está uma linha de código mostrando como é feita a recuperação de dados escritos no formulário pelo usuário para que seja feita a inserção ou edição, essa linha de código deve ser repetida para todos os campos existentes na tabela.

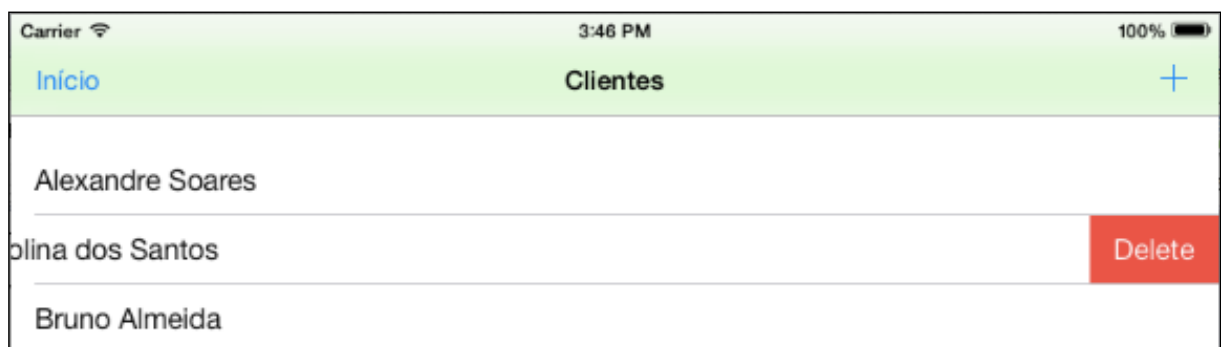
Quadro 18: informa o nome do cliente a ser inserido/editado

```
1. sqlite3_bind_text (stmt, 2, [cliente.nome UTF8String], -1, nil);
```

Fonte: do autor

O método que realiza a exclusão de um registro no banco de dados é mais simples, o importante é que se receba o objeto que deve ser deletado para apenas criar o código SQL de deleção com a cláusula `where` indicando o `_id` daquele que será excluído. É interessante observar que para o usuário realizar essa operação foi implementado o *swipe*, que é o ato de deslizar o dedo na célula, que faz com que apareçam novas opções, nesse caso o “Delete”:

Figura 7: *Swipe* na segunda célula



Fonte: do autor

Por fim, a operação faltante é referente a consulta dos dados, que permite, por exemplo uma listagem de clientes, como a que se pode ver na Figura 7. No estudo de caso o SQL é codificado para que se recupere todos os campos da tabela cliente. Para esse método o que difere do que já foi visto nos outros métodos é a forma como os dados são lidos, para cada campo tem-se linhas de código como as seguintes:

Quadro 19: Leitura de coluna do banco de dados

```
1. c._id = sqlite3_column_int(stmt, 0);
```

Fonte: do autor

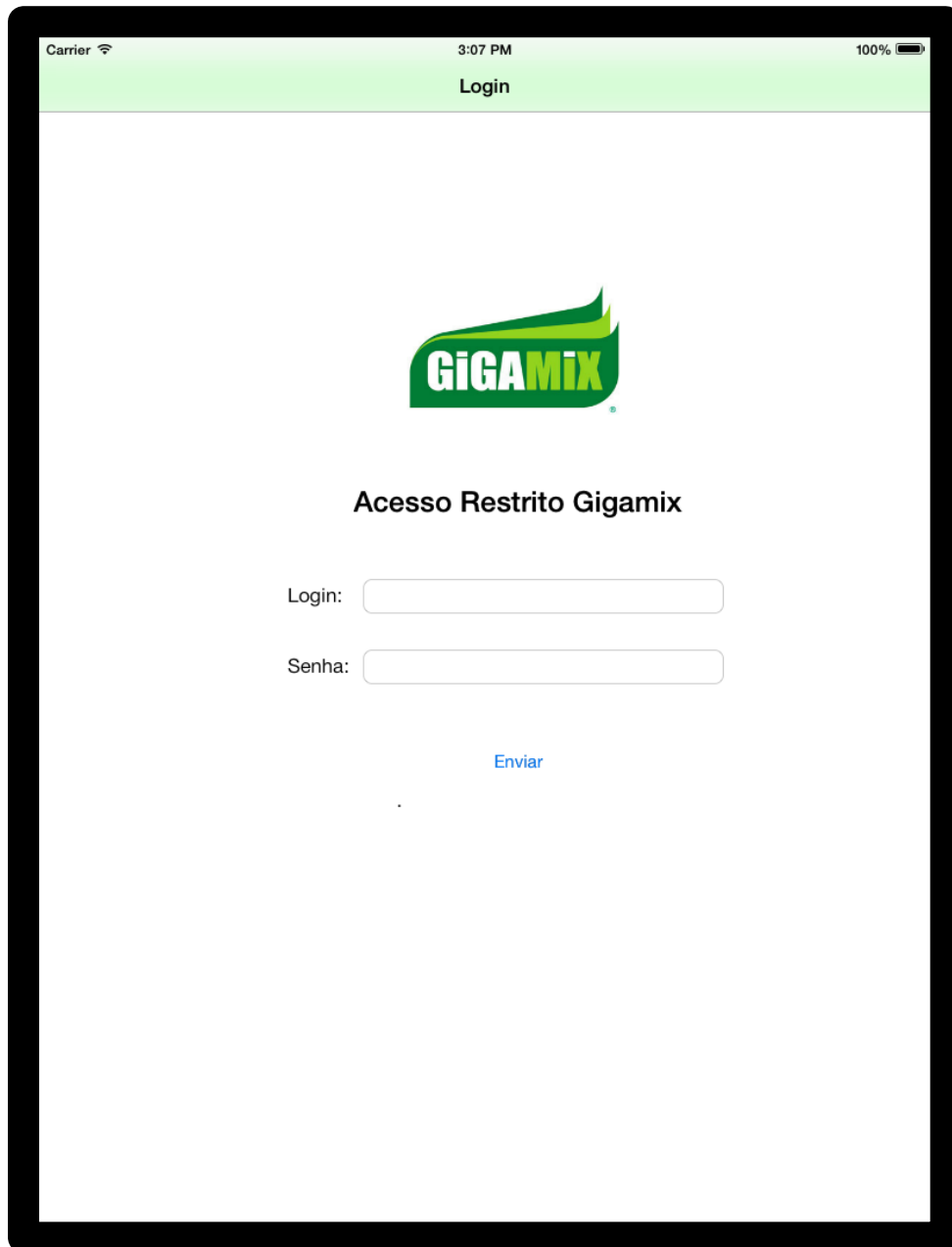
A função `sqlite3_column_int` faz a leitura de dados, que podem ser de tipos diferentes, em caso de *string*, por exemplo, `sqlite3_column_text` é usado. A leitura é feita dentro de um laço `while`, que compara o resultado obtido da criação do *statement* com `SQLITE_ROW`, sendo possível percorrer todos os registros de forma adequada. Para todos os métodos de manutenção a finalização do *statement* é feita com a função `sqlite3_finalize`.

Para cada uma das outras tabelas existe uma classe, a qual possui os métodos de manutenção que necessita. Todas essas classes juntas formam o modelo do MVC desejável na programação para Objective-C. As outras classes que necessitam utilizar de qualquer manutenção têm a seu favor métodos especializados nisso, basta que se importe a classe correspondente.

3.3.4 Telas do sistema

Com o estudo de caso finalizado, totalizaram-se oito telas com propósitos distintos. Primeiramente a tela de login, onde o usuário deverá preencher com seu login e senha para ter acesso à tela de início. A segunda tela apresenta o botão “Clientes” responsável por acessar outra parte do sistema, os outros botões estão desabilitados para implementação de futuras funcionalidades. As telas estão dispostas na Figura 8 e Figura 9.

Figura 8: Tela de login



Carrier 3:07 PM 100%

Login



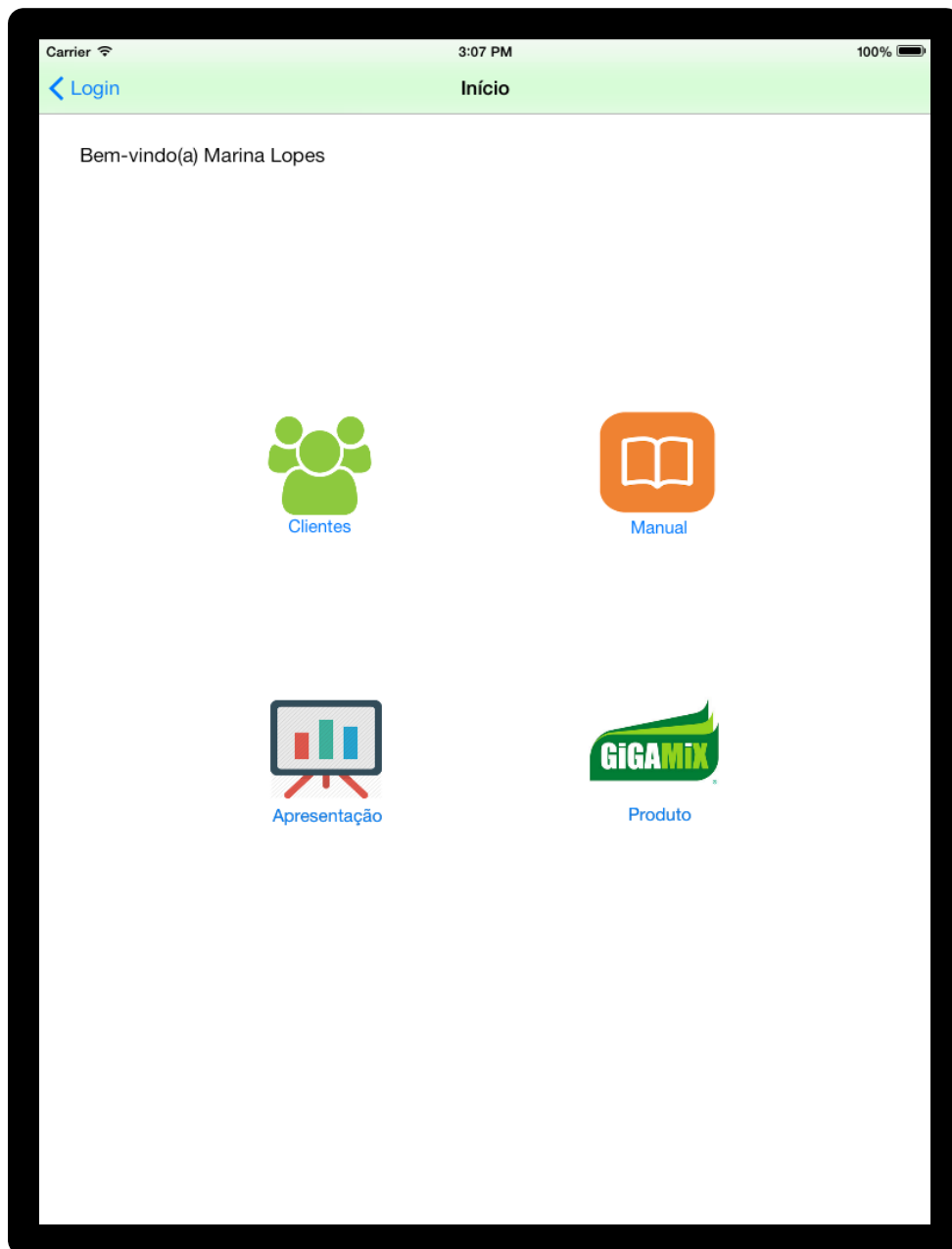
Acesso Restrito Gigamix

Login:

Senha:

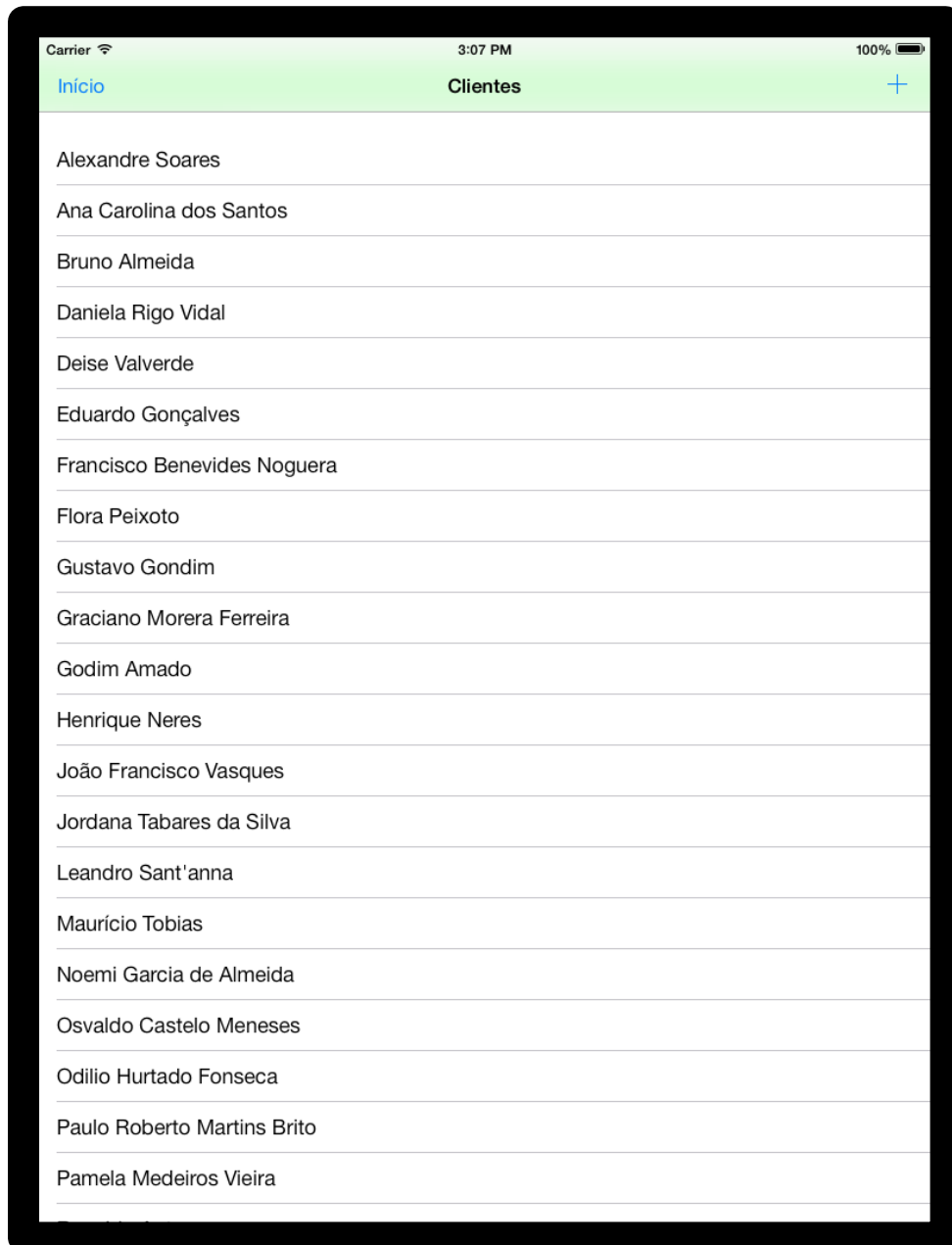
[Enviar](#)

Fonte: do autor

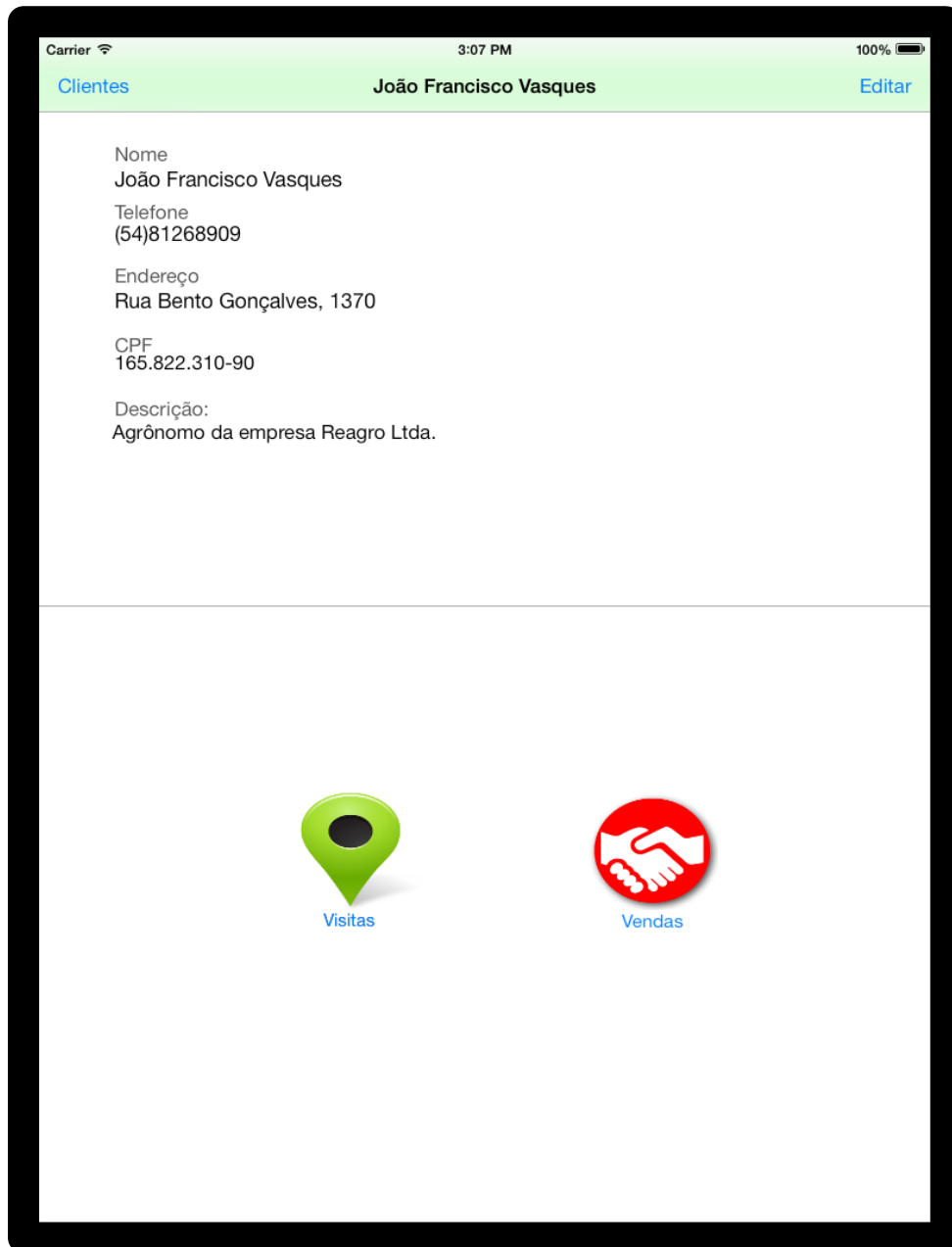
Figura 9: Tela de início

Fonte: do autor

Ao tocar no botão “Clientes”, na tela de início, é aberta uma lista de todos os clientes (Figura 10) cadastrados no banco de dados. Quando uma das células com os nomes dos clientes é acessada tem-se a descrição dos mesmos com todos os dados que foram salvos (Figura 11), além disso, há o acesso ao botão “Visitas”, o botão “Vendas” será desenvolvido futuramente.

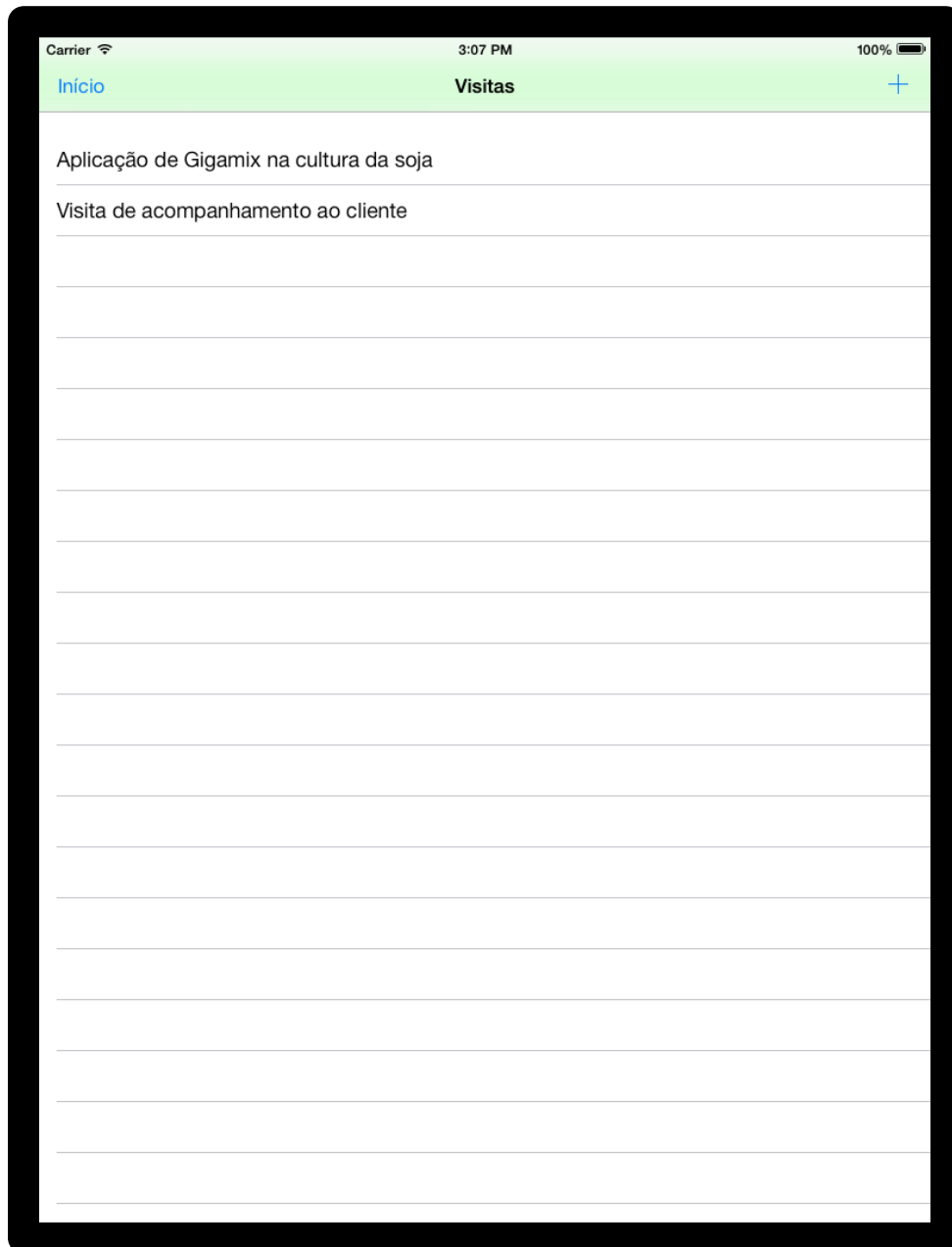
Figura 10: Tela de listagem de clientes

Fonte: do autor

Figura 11: Tela de descrição de cliente

Fonte: do autor

O botão “Visitas” dá acesso a uma listagem de visitas realizadas no cliente correspondente (Figura 12). Da mesma forma que acontece na lista de clientes, aqui o usuário também tem acesso a uma tela de descrição com todas as informações referentes à visita realizada, como pode ser visto na Figura 13:

Figura 12: Tela de listagem de visitas

Fonte: do autor

Figura 13: Tela de descrição de visitas

Carrier 3:10 PM 100%

Visitas

Descrição:
Aplicação de Gigamix na cultura da soja

Data:
20/02/2014

Experimento:

1- Severidade de Ferrugem Asiática em - % de área foliar ocupada por pústulas de Ferrugem Asiática em 5 trifólios do 1/2 inferior e 5 do 1/2 superior.

2-Peso de mil grãos

Objetivos:

Avaliar os benefícios para a cultura de soja na lavoura de João Francisco.

Comentários:

Data das aplicações: 24/11/2013 e 09/12/2013

Resultado:

Nas avaliações percentagem de veridade de ferrugem, pode-se observar que os tratamentos que receberam Aplicação de Gigamix em V4 e V4 + 15DAA, obtiveram menores médias se comparado com o tratamento testemunha, respondendo no incremento produtividade de 8%.
Os tratamentos que receberam aplicações com o produto Gigamix obtiveram maiores médias de peso de mil grãos e com mais de uma Aplicação pode-se observar que o

Cultura:
Soja

Área:
5.000000

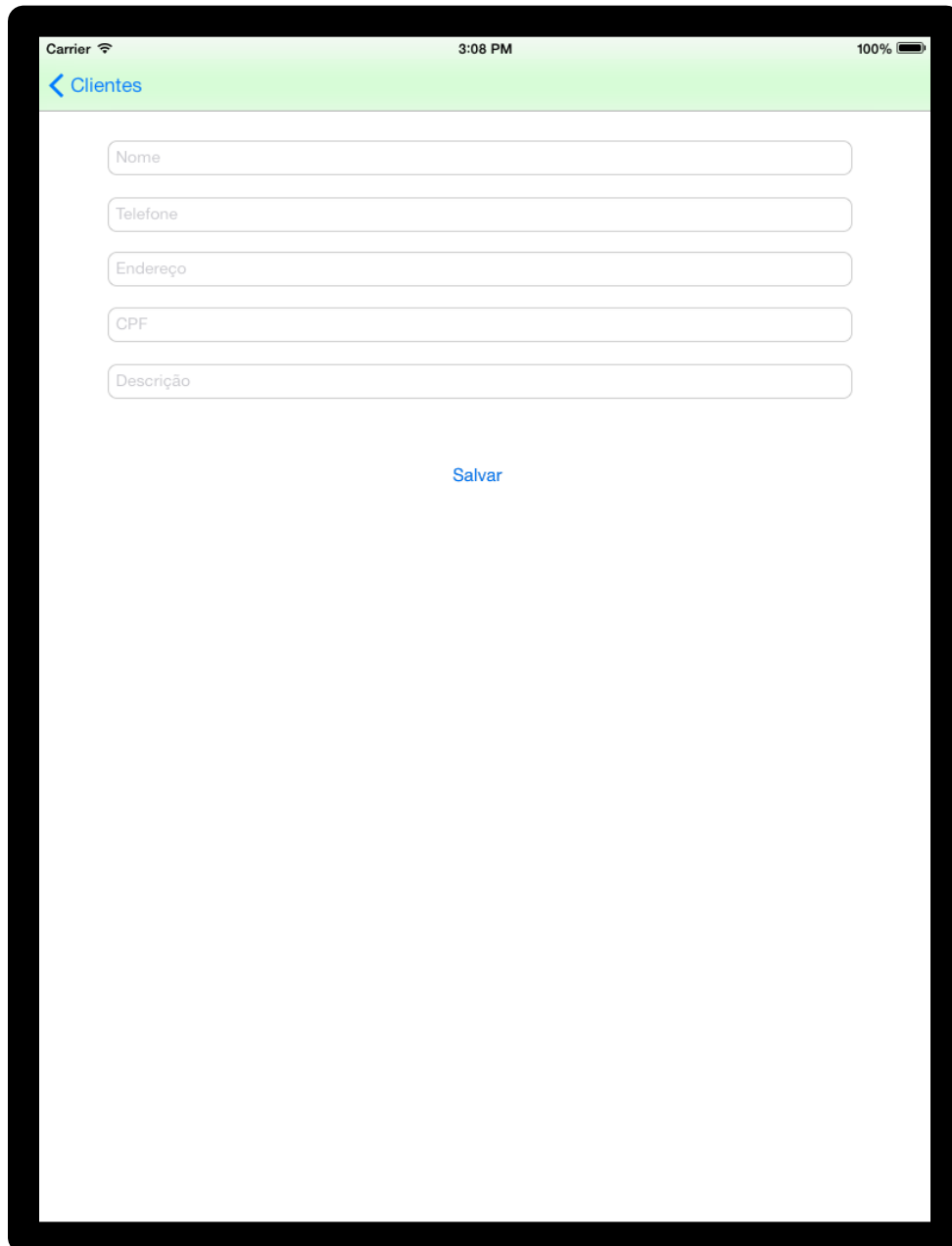
Fonte: do autor

Finalmente, têm-se os dois formulários necessários (Figura 14 e Figura 15), para acrescentar e editar um cliente e para adição de visitas. A adição de um novo cliente ou uma nova visita é feita nas listagens correspondentes de cada um, o botão responsável por abrir o formulário mostra-se como o sinal de adição “+”, na parte superior da tela onde fica o sistema de navegação entre telas.

Para abrir o formulário de edição do cliente deve-se tocar no botão “Editar” que pode ser encontrado na barra superior de navegação à direita, (vide Figura 11). Nesse caso é aberto

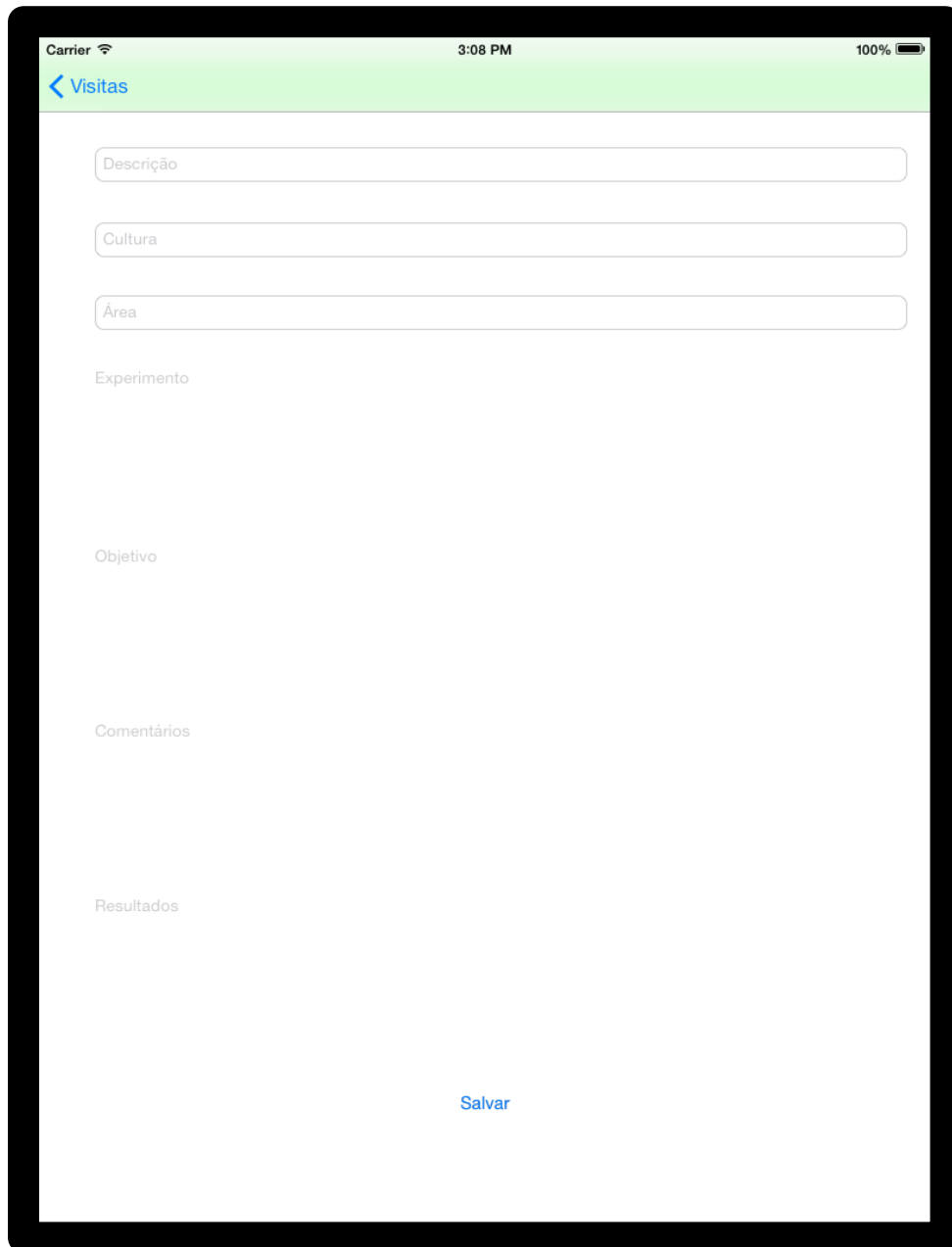
o mesmo formulário da adição de dados, porém com os dados que já foram salvos recuperados do banco de dados, para que seja feita a atualização.

Figura 14: Tela de formulário do cliente



The image shows a mobile application interface for updating a client's information. At the top, there is a status bar with "Carrier", signal strength, "3:08 PM", and "100%" battery. Below this is a green header bar with a back arrow and the text "Clientes". The main content area contains five text input fields stacked vertically, labeled "Nome", "Telefone", "Endereço", "CPF", and "Descrição". Below the input fields is a blue button labeled "Salvar".

Fonte: do autor

Figura 15: Tela de formulário de visitas

The image shows a mobile application interface for a 'Visitas' (Visits) form. The screen is framed by a thick black border. At the top, there is a status bar with 'Carrier', signal strength, '3:08 PM', and '100%' battery. Below the status bar is a green header with a back arrow and the text 'Visitas'. The form consists of several input fields: 'Descrição', 'Cultura', and 'Área', each with a light blue border. Below these are labels for 'Experimento', 'Objetivo', 'Comentários', and 'Resultados'. At the bottom center, there is a blue button labeled 'Salvar'.

Fonte: do autor

Ambas as telas de formulário possuem o botão “Salvar” utilizado quando o registro acaba de ser feito, esse botão além de realizar a manutenção no banco de dados direciona o usuário à tela de descrição de cliente ou de visita. A intenção é que o usuário tenha a confirmação de que os dados foram salvos, e facilidade na continuidade de sua navegação pelo sistema.

4 CONSIDERAÇÕES FINAIS

Dispositivos móveis estão, de fato, presentes no cotidiano de um número cada vez maior de pessoas, as quais são atraídas por suas facilidades. Ainda que smartphones e tablets apresentem incontáveis utilidades, há muito que ser aperfeiçoado, criado e desenvolvido para essa área.

Aprender a desenvolver aplicativos na linguagem de programação Objective-C é factível. Evidentemente é necessário que se dedique tempo de estudo para acostumar-se com a sintaxe e com as peculiaridades da linguagem, como, nomes de funções e métodos longos, estrutura de classes diferenciada, plataforma de desenvolvimento específica, utilização de classes próprias com particularidades, entre outros.

Para que o estudo de caso pudesse ser desenvolvido precisou-se de muita leitura e várias tentativas de desenvolver pequenos exemplos encontrados na internet ou em livros. Tudo o que é visto e estudado deve ser adaptado para o contexto desejado, tarefa conquistada apenas quando já há um bom nível de conhecimento sobre o desenvolvimento na linguagem Objective-C.

O estudo de caso apresentado mostra como a empresa BSC-Gigamix necessita de uma solução rápida para tornar seus fluxos de trabalho mais eficazes. O aplicativo criado com a intenção de solucionar parcialmente os problemas do empreendimento mostra-se capaz de garantir um melhor controle das informações de cada cliente e de visitas realizadas a ele.

Quando a fase inicial de aprendizado do Objective-C passa, a implementação do estudo de caso torna-se mais intuitiva e confortável de ser feita. Embora demore a adequar-se a linguagem, a consequência que se tem ao estudá-la é a de saber que existem possibilidades enormes na programação para o iOS e que fazer isso pode também ser confortável e instigante.

4.1 TRABALHOS FUTUROS

Inserir-se nessa dinâmica de mercado como desenvolvedor de aplicativos iOS e não apenas como usuário evidencia-se como uma ótima oportunidade, já que ainda são poucas as pessoas com esse conhecimento. Assim, são muitas as oportunidades que podem ser vistas a partir do momento que há o interesse em continuar a crescer dentro dessa área.

O término do sistema para a empresa BSC-Gigamix utilizar é um dos trabalhos futuros. Poderão ser feitas implementações daquilo que já está planejado, como acesso a um histórico

de vendas, consulta a especificações do produto, apresentações e instruções de uso. Além disso, o sistema poderá ainda contar com a possibilidade de trabalhar tanto online, quanto offline a partir do desenvolvimento de um mecanismo de sincronização de dados.

O estudo da linguagem Objective-C e do sistema operacional iOS poderá ser continuado, abrindo a possibilidade de criar novas soluções, aperfeiçoar conhecimentos e contribuir para o aumento das possibilidades de uso dos dispositivos móveis.

REFERÊNCIAS

APPLE Inc. iOS Technology Overview. Cupertino, 2012. Disponível em: <<http://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/iPhoneOSTechOverview.pdf>>. Acesso em: 22 jun 2013.

FERNANDES, Alexandre. A evolução do iOS: do iPhone ao iOS 7. Apple Tudo, 2013. Disponível em: <<http://www.appletudo.com.br/2013/01/a-evolucao-do-ios-do-iphone-os-ao-ios-7.html>>. Acesso em: 22 jun 2013.

GARREFA, Alexandre. Introdução a programação para iOS. DevMedia. Disponível em: <http://www.devmedia.com.br/introducao-a-programacao-para-plataforma-ios/27032>. Acesso em: 05 mai 2014.

KOCHAN, Stephen G. Programming in Objective-C A complete introduction to the Objective-C language. Indianapolis: Sams Publishing, s.d. Google Books. Disponível em: <http://books.google.com.br/books?id=6lStBOM1DEwC&printsec=frontcover&hl=pt-BR&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false>. Acesso em: 22 jun 2013.

MARZULLO, Fabio. iPhone na prática aprenda passo a passo a desenvolver soluções para iOS. 1. ed. São Paulo: Novatec, 2012.

SAMPAIO Cleuton, RODRIGUES Francisco. Mobile Game Jam Criação de jogos móveis multiplataforma. Rio de Janeiro: Brasport, 2012.

SILVA, Renato. A evolução do sistema operativo móvel. Porto, 2010. Disponível em: <<http://pt.calameo.com/read/00053935644470a885967>>. Acesso em: 22 jun 2013.

LECHETA, Ricardo R. Desenvolvendo para iPhone e iPad Aprenda a desenvolver aplicações utilizando o iOS SDK. 1. ed. São Paulo: Novatec, 2012

KIRCOVE, Bernardo. Android, iOS ou Windows Phone: qual é o melhor sistema para smartphones. Techtudo, 2013. Disponível em: <<http://www.techtudo.com.br/artigos/noticia/2013/04/android-ios-ou-windows-phone-qual-e-o-melhor-sistema-para-smartphones.html>>. Acesso em: 23 mai 2014.

SILVA, Fernando Rodrigues. Iniciando o desenvolvimento com Android. Devmedia, s.d. Disponível em: <<http://www.devmedia.com.br/iniciando-o-desenvolvimento-com-android/21025>>. Acesso em: 23 mai 2014.

SAFATLI, Nabil. Introdução ao Desenvolvimento em Windows Phone.Devmedia, s.d. Disponível em: <<http://www.devmedia.com.br/introducao-ao-desenvolvimento-em-windows-phone/26642>>. Acesso em: 23 mai de 2014

OLHAR DIGITAL. Windows Phone deixa iOS para trás no Brasil. Redação Olhar Digital, 2014. Disponível em: <<http://olhardigital.uol.com.br/pro/noticia/40338/40338>>. Acesso em: 23 mai 2014.

ANEXOS

Anexo A: Panfleto do Gigamix com as especificações da utilização do produto.



GIGAMIX é um insumo mineral complexo
que apresenta em sua composição uma grande variedade
de nutrientes obtidos de forma natural.

Insumo apropriado para uso na produção orgânica. Utilização condicionada aos critérios de cada regulamento orgânico conforme respectivo Atestado emitido. Inspeccionado pela Ecocert.



Como utilizar GIGAMIX	DOSAGEM	FORMA DE APLICAÇÃO
Frutíferas em geral: videira, citros, pêssego, ameixa, maçã, caqui, mamão, melão, maracujá, manga, goiaba, abacate, outras.	250 gramas/100 litros de água ou até 2 kg / hectare	Foliar: pulverizar desde a brotação até a fase de pré-colheita especialmente na floração até a fixação dos frutos
Hortaliças e olerícolas em geral: alface, repolho, brócoli, couve-flor, rúcula, radiche, beterraba, cenoura, rabanete, outras	250 gramas/100 litros	Foliar: pulverizar semanalmente, espaçando os tratamentos conforme a necessidade ou o ciclo da cultura
Flores em geral	250 gramas/100 litros	Foliar: pulverizar semanalmente
Café	250 gramas/100 litros de água ou até 2kg / hectare	Foliar: pulverizar desde a brotação até a fase de pré-colheita especialmente na floração até a fixação dos frutos
Soja, feijão, leguminosas em geral	500 gramas/100 kg 350 gramas/tratamento/HA	TS - Tratamento de Sementes Foliar: pulverizar 3 vezes no ciclo até a formação da vagem
Milho, arroz, girassol, canola	500 gramas/100 kg 350-500 gramas/tratamento/HA	TS - Tratamento de Sementes Foliar: pulverizar 2-3 vezes durante o ciclo até a floração
Algodão	500 gramas/100 kg 350-500 gramas/tratamento/HA	TS - Tratamento de Sementes Foliar: pulverizar 6-8 vezes durante o ciclo até a abertura do primeiro capulho
Cereais de inverno: trigo, aveia, triticale, cevada, centeio	500 gramas/100 kg 350-500 gramas/tratamento/HA	TS - Tratamento de Sementes Foliar: pulverizar 2-3 vezes até o espigamento, preferencialmente junto com defensivos agrícolas
Cebola, alho	250 gramas/100 litros	Foliar: pulverizar quinzenalmente até a formação dos bulbos
Tomate, pimentão	250 gramas/100 litros	Foliar: pulverizar semanalmente até a maturação dos primeiros frutos
Fumo	50 gramas/ regador 70 gramas / 20 litros	Pulverizar semanalmente as bandejas a partir da repicagem Foliar: Pulverizar 3 - 4 vezes durante o ciclo na lavoura
Batata-doce, batata-inglesa	5-10 gramas/litro 350 gramas/100 litros	Pulverizar os tubérculos antes ou no momento do plantio Foliar: pulverizar quinzenalmente
Cana-de-açúcar	10 gramas/litro 350-500 gramas/100 litros	Pulverizar os toletes antes do plantio ou fazer imersão por 24 horas Foliar: pulverizar mensalmente
Pastagens diversas	500 gramas/100 kg 350-500 gramas/100 litros 500 gramas/100 litros	TS - tratamento de sementes Foliar: pulverizar mensalmente No pastoreio rotativo, pulverizar na saída dos animais do piquete
Hidroponia / Fertilirrigação	300 gramas/1000 litros	Semanalmente
Substratos	2 a 3 kg / m ³	Misturar ao substrato antes da sementeira ou transplante
Mudas frutíferas, florestais e ornamentais	2 a 3 kg / m ³ 250 gramas/100 litros	Misturar ao substrato antes da sementeira ou transplante Pulverizar quinzenalmente
Horta & Jardim doméstico	Variável	Em um regador com 10L de água, adicionar duas colheres (sopa) com GIGAMIX. Repetir a cada 15 dias.
Enraizamento de estacas	2,5 / litros	Umedecer as estacas e envolver com GIGAMIX, colocando em seguida no meio do enraizamento (espuma fenólica, areia ou outro substrato) Manter a umidade com solução de 5g/l de GIGAMIX até o transplante

MODO DE PREPARO DO GIGAMIX

- No tratamento de sementes (TS), umedecer as mesmas com água + espalhante adesivo. Respeitar a proporção máxima de 0,5% de produto **GIGAMIX** quantidade de semente. Utilizar sempre espalhante adesivo de qualidade no TS e nas Pulverizações.
- Pulverizações de alta pressão: fazer pré-diluição do produto em um balde (20). Adicionar no tanque/pulverizador preferencialmente com o agitador ligado;
- Pulverizações de baixa pressão. Fazer pré-diluição do produto em um balde (20);

REVENDEDOR AUTORIZADO:

VISITE NOSSO SITE: WWW.GIGAMIXAGRO.COM.BR

Anexo B: Anotações feitas a mão sobre contatos de clientes.

MARCOS

FORMOSA COOPLAN - CARLOS - BRUNO
 CARLOS - 61 3642 ~~1121~~ - 9965 ~~9121~~
 BRUNO " " - 8164 ~~9121~~

FORMOSA LIDERAGRO - MARDEY - CARLOS
 MARDEY 61 3642 ~~1121~~ - 9983 ~~6121~~
 CARLOS " " 9807 ~~9121~~

LUIZ EDUARDO COOPERFARMS
 ODAIR 77 9968 ~~4121~~ - 3628 ~~5538~~
 ENFRENTA AO HOTEL

COLONIA GURGUEIA POSTO COLONIAL
 PAULA NAMORADO MARCOS TEM PIVO
 MARCOS 89 9408 ~~0121~~

BOM JESUS - ELDER PEDRO 89 3562 ~~1121~~
 89 9985 ~~2121~~

- IGOR CAMPARI

NOVA STA ROSA DOUGLAS 89 3544 ~~1121~~
 8116 ~~3121~~

Anexo C: Formulário atual de pedidos e orçamentos Gigamix.

 FERTILIZANTE E ESTIMULADOR VEGETAL	<input type="checkbox"/> PEDIDO <input type="checkbox"/> ORÇAMENTO			
www.gigamixfertilizante.com.br				
INFORMAÇÕES CADASTRAIS DO CLIENTE				
Cliente: _____ Data: _____ Endereço: _____ Estado: _____ Bairro: _____ Cidade: _____ Inscrição Estadual: _____ CEP: _____ CNPJ: _____ Fone/Fax: _____ E-mail: _____ Contato: _____ Prazo de entreg: _____ Cond. De Pagto. _____				
ENDEREÇO DE COBRANÇA (CASO DIFERE DO ACIMA)				
Endereço: _____ Bairro/Complemento: _____ Cidade: _____ Estado: _____ CEP: _____ Contato: _____ Fone/Fax: _____ E-mail: _____				
ENDEREÇO DE ENTREGA (CASO DIFERE DO ACIMA)				
Nome: _____ Bairro: _____ Endereço: _____ Estado: _____ CEP: _____ Cidade: _____ Fone: _____ E-mail: _____ Contato: _____ Fone: _____ CEI: _____				
Qtid.	UN.	Descrição dos Produtos	VALOR UNIT.	TOTAL
				R\$ 0,00
				R\$ 0,00
				R\$ 0,00
				R\$ 0,00
				R\$ 0,00
				R\$ 0,00
				R\$ 0,00
				R\$ 0,00
				R\$ 0,00
				R\$ 0,00
				R\$ 0,00
				R\$ 0,00
				R\$ 0,00
				R\$ 0,00
VALOR TOTAL DOS PRODUTOS:				R\$ 0,00
	FRETE			R\$ 0,00
VALOR TOTAL:				R\$ 0,00
Pedido sujeito a confirmação, cessando nossa responsabilidade de uma vez entregue o produto, não aceitamos reclamações posteriores.				
OBSERVAÇÕES Quantitativos fornecidos pelo comprador, havendo variações prevalece o preço unitário; Material entregue na localidade do cliente; Descarga por conta do comprador/cliente;				
_____ VENDEDOR TELEFONE EMAIL		_____ 0		