UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

APLICATIVO DE FORÇA DE VENDAS PARA DISPOSITIVOS MÓVEIS BASEADOS EM WINDOWS PHONE

RAFAEL KULKAMP DA SILVA

BLUMENAU
2015

2015/1-25

RAFAEL KULKAMP DA SILVA

APLICATIVO DE FORÇA DE VENDAS PARA DISPOSITIVOS MÓVEIS BASEADOS EM WINDOWS PHONE

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Prof. Francisco Adell Péricas, Mestre - Orientador

BLUMENAU
2015

2015/1-25

APLICATIVO DE FORÇA DE VENDAS PARA DISPOSITIVOS MÓVEIS BASEADOS EM WINDOWS PHONE

Por

RAFAEL KULKAMP DA SILVA

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Prof. Francisco Adell Péricas, Mestre – Orientador, FURB
Membro:	Prof. Wilson Pedro Carli, Mestre – FURB
Membro:	Prof. Alexander Roberto Valdameri, Mestre – FURB

Blumenau, 6 de Julho de 2015

AGRADECIMENTOS

À meus pais, Felício José da Silva e Edna Kulkamp da Silva, e meu irmão, Ricardo Kulkamp da Silva, pelo apoio durante a realização deste trabalho.

Ao meu orientador, Francisco Adell Péricas, pela confiança na conclusão deste trabalho.

RESUMO

Este trabalho apresenta um aplicativo de força de vendas que foi desenvolvido visando atender o que acontece na empresa Proagro Agropecuária, que está localizada na cidade de Luiz Alves, no estado de Santa Catarina. O aplicativo feito para Windows Phone tem como objetivo facilitar o acesso da equipe de força de vendas a informações importantes para seu trabalho diário. Além de agilizar o cadastro de novos pedidos e clientes, permite que sejam feitos *offline* e enviados quando houver conexão, através de um *web service* também desenvolvido neste trabalho. No desenvolvimento foi utilizado o ambiente Visual Studio com a linguagem de programação C# e o banco de dados SQLite. Estas ferramentas, mesmo apresentando alguns problemas com o banco de dados, permitiram a construção de um aplicativo que faz a sincronização sob demanda.

Palavras-chave: Força de vendas. Windows Phone. *Offline*. *Web service*.

ABSTRACT

This paper presents a sales force application that was developed to meet what happens in Proagro Agropecuária company, that is localized in Luiz Alves city, in Santa Catarina state. The application made for Windows Phone aims to facilitate the sales force team to access important information for their daily work. Also, speed up the registration of new orders and customers, allows them to be made offline and sent when there is connection through a web service also developed in this work. In the development it was used the Visual Studio environment with the programming language C# and SQLite database. These tools, even with some problems with the database, allowed the construction of an application that synchronizes on demand.

Key-words: Sales force. Windows Phone. Offline. Web service.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API – *Application Programming Interface*

BTS – *Background Transfer Service*

CPF – Cadastro de Pessoas Física

HTTP – *HyperText Transfer Protocol*

MER – Modelo de Entidade Relacionamento

NFC – *Near Field Communication*

PDF – *Portable Document Format*

REST – *Representational State Transfer*

SOAP – *Simple Object Access Protocol*

SQL – *Structured Query Language*

URI – *Uniform Resource Identifier*

WCF – *Windows Communication Foundation*

XAML – *eXtensible Application Markup Language*

XML – *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

São cada vez maiores os benefícios que empresas de diversos setores podem ter com tecnologias de mobilidade para profissionais que atuam em campo, prestando serviços direto para os clientes. Sistemas de mobilidade com coletores de dados/*smartphones*, aliados a pacotes de dados de operadoras de telefonia celular e softwares específicos, têm tido uma aceitabilidade crescente em companhias focadas no gerenciamento logístico, transporte e abastecimento de produtos no varejo e prestadores de serviços. Esses equipamentos proporcionam maior eficiência nos processos de venda, aumento da produtividade e redução de custos (BERNARDES, 2009).

Segundo Lecheta (2013, p. 22), "a plataforma da Microsoft é atualmente uma das melhores e vem ganhando a cada dia mais usuários e espaço no mercado. O ambiente de desenvolvimento também é muito simples e produtivo". Portanto o Windows Phone demonstra ser uma ótima opção para o desenvolvimento de um sistema móvel.

Hoje a equipe de força de vendas da Proagro Agropecuária, empresa que está localizada na cidade de Luiz Alves, estado de Santa Catarina, tem dificuldades em ter acesso às informações. Para ter, por exemplo, listas de cobrança e estoque atualizadas e entregar seus novos pedidos para a empresa, ela deve se deslocar até a sede, ou se comunicar por *e-mail*. Assim, constantemente ocorrem problemas com entregas ou cobranças, devido às informações incorretas geradas por falhas de comunicação.

Diante do exposto, foi desenvolvido um aplicativo que permite vendedores consultarem informações da empresa e cadastrarem clientes, pedidos e anotações. Além disso, também foi desenvolvido um *web service* que mantém os dados sincronizados entre o aplicativo e o servidor, que não está integrado com o sistema da empresa, portanto, usa informações que foram copiadas manualmente da base de dados da empresa.

1.1 OBJETIVOS

O objetivo deste trabalho é apresentar o desenvolvimento de um aplicativo para Windows Phone para apoio à equipe de força de vendas de vendedores externos sincronizado com um servidor da empresa Proagro Agropecuária através de um *web service*.

Os objetivos específicos do trabalho são:

a) disponibilizar um *web service* para sincronização entre os dispositivos móveis e o servidor;
b) disponibilizar interface *mobile* para consulta de estoque;
c) disponibilizar interface *mobile* para inserção de novos clientes, novos pedidos e

anotações em pedidos existentes;

d) permitir visualização de dados do servidor na forma de relatórios, como contas à receber, situação de entrega dos pedidos, situação do estoque e tabela de preços.

1.2 ESTRUTURA

No primeiro capítulo é apresentado uma introdução ao tema do trabalho, juntamente com os objetivos.

O segundo capítulo contém a fundamentação teórica pesquisada sobre *web services*, Windows Phone e sobre trabalhos correlatos.

O capítulo três demonstra o desenvolvimento do aplicativo, desde os requisitos, passando pela especificação e implementação e terminando nos resultados.

No quarto capítulo são apresentadas as conclusões obtidas e são apresentadas as sugestões para extensões.

2 FUNDAMENTAÇÃO TEÓRICA

A seção 2.1 expõe sobre *web services*, em seguida a seção 2.2 apresenta o sistema móvel Windows Phone e por fim na seção 2.3 são mostrados dois trabalhos correlatos.

2.1 WEB SERVICES

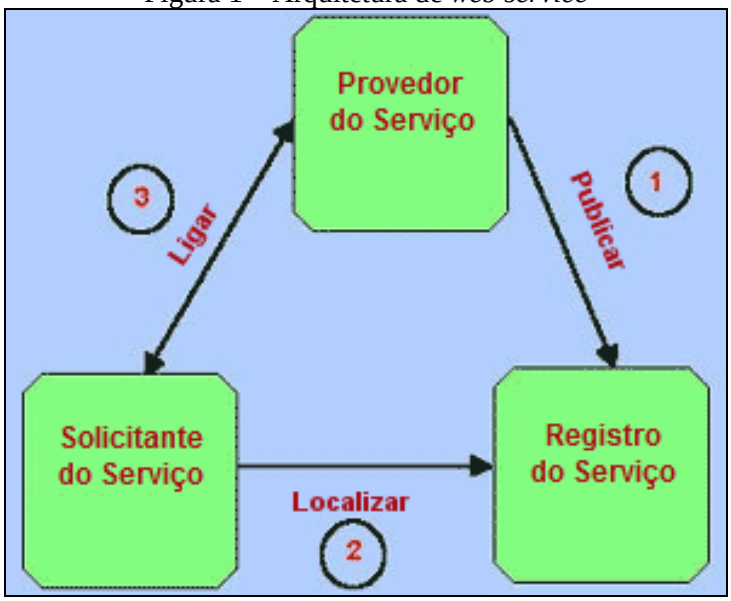
Web services são descritos pela *World Wide Web Consortium* (W3C) como aplicativos cliente e servidor que se comunicam por *HyperText Transfer Protocol* (HTTP), que criam um meio padrão de comunicação entre aplicações de software em execução em diferentes plataformas e *frameworks*. São caracterizados pela sua grande interoperabilidade e extensibilidade, bem como as suas descrições processáveis por máquina, graças à utilização de *eXtensible Markup Language* (XML). Portanto, utilizando *web services* é possível produzir serviços sofisticados por meio de vários programas que prestam serviços simples (ORACLE, 2013, p. 363, tradução nossa).

> *Web services* foram criados para construir aplicações [...] que são serviços na internet. Porém não faz parte do conceito de *web service* a criação de interfaces gráficas para os usuários, deixando esta parte para outras empresas ou pessoas desenvolverem. É comum encontrar textos afirmando que *web services* disponibilizam serviços somente para desenvolvedores, ou que *web services* nada mais são do que chamada de métodos usando XML. (PAMPLONA, 2010).

Para que um computador ou um programa utilize um *web service*, ele precisa ser capaz de encontrar a descrição do serviço, para então ligar-se a ele. Para isso, existem três papéis fundamentais na arquitetura de *web services*: o provedor do serviço, o registro do serviço e o solicitante do serviço. Juntos eles executam três operações no *web service*: publicar, localizar e ligar (GRALLA, 2002, tradução nossa).

Na Figura 1 é possível observar como todas as operações se encaixam. A operação publicar torna as informações sobre o serviço acessíveis para que o *web service* possa ser encontrado e usado, ou seja, faz a descrição do serviço ficar disponível ao público. A operação localizar descobre o *web service*, é a maneira que o computador ou programa procura e entende o que o *web service* é, onde está localizado, e como conectar-se a ele. Já a operação ligar permite que uma pessoa ou programa faça solicitações ao serviço (GRALLA, 2002, tradução nossa).

Figura 1 – Arquitetura de *web service*



Fonte: Gralla (2002).

Segundo Attorre (2015), “Entre as abordagens existentes para a implementação de *web services*, o protocolo SOAP e o REST são as opções de maior destaque nos dias de hoje, estando presentes em grande parte das discussões relacionadas a arquiteturas orientadas a serviços na *web*.”

2.1.1 SOAP

O *Simple Object Access Protocol* (SOAP) é um protocolo simples e leve para troca de informação em um ambiente distribuído e descentralizado, como é o ambiente da Internet. Em outras palavras, SOAP habilita dois processos a se comunicarem desconsiderando o hardware e a plataforma que eles estão rodando (LEOPOLDO, 2003).

Conforme Leopoldo (2003), "ele é um protocolo que define uma gramática XML especializada, porém flexível, que padroniza o formato das estruturas das mensagens. As mensagens são, por outro lado, o método fundamental de troca de informações entre os *web services* e os seus consumidores".

Segundo Leopoldo (2003), a especificação do protocolo SOAP está dividida em quatro partes:

a) SOAP *Envelope*: define como a mensagem será descrita e como será processada. Tem os dados da mensagem e é a única parte do protocolo que é obrigatória;
b) SOAP *Encoding Rules*: aponta um mecanismo de serialização que pode ser usado para troca de instâncias de tipos definidos pela aplicação;
c) SOAP *Remote Procedure Call Style*: determina uma convenção que pode ser usada para representar as chamadas e respostas remotas aos procedimentos;

d) *Link* SOAP HTTP: designa um protocolo que liga o SOAP e o HTTP. Ele descreve como as mensagens SOAP são transmitidas via HTTP.

2.1.2 REST

Representational State Transfer (REST) é uma arquitetura que especifica restrições se aplicada a um *web service* e induz a desejáveis melhorias, como performance, escalabilidade, modificabilidade, que permitem os serviços funcionarem melhor. Na arquitetura REST, dados e funcionalidades são considerados recursos e são acessados usando o *Uniform Resource Identifier* (URI), *links* comuns da web. Os recursos são acionados usando um conjunto simples de operações bem definidas. A arquitetura REST é cliente/servidor e é projetada para usar protocolo de comunicação sem estado, normalmente HTTP (ORACLE, 2013, p. 381, tradução nossa).

Segundo Oracle (2013, p. 382, tradução nossa), o REST é baseado em quatro princípios:

a) recursos identificados através do URI: o REST expõe um conjunto de recursos que identificam os alvos de interação com os clientes. Os recursos são identificados por URIs, quem proporcionam um espaço de endereçamento global de recursos e descoberta de serviços;
b) interface uniforme: recursos são manipulados usando um conjunto fixo de quatro operações: *Put*, *Get*, *Post* e *Delete*. *Put* cria um novo recurso, que pode ser deletado usando *Delete*. *Get* busca o estado atual de um recurso em alguma representação. *Post* transfere um novo estado para um recurso;
c) mensagens auto descritivas: os recursos são desacoplados de suas representações, então o seu conteúdo pode ser acessado em um variedade de formatos, como HTML, XML, texto simples, *Portable Document Format* (PDF), *Joint Photographic Experts Group* (JPEG), JavaScript *Object Notation* (JSON), e outros. Metadados sobre o recurso estão disponíveis e são usados para, por exemplo, controlar *cache*, detectar erros de transmissão, negocionar o formato de representação apropriado, e fazer autenticação ou controle de acesso;
d) interações com estado através de *hyperlinks*: cada interação com um recurso é sem estado, ou seja, mensagens de solicitação são independentes. Interações com estado são baseadas no conceito transferência explícita de estado. Existem algumas técnicas para mudar o estado, como a reescrita da URI, *cookies*, e campos com forma oculta. Estado pode ser incorporado em mensagens de resposta para apontar

para os estados futuros válidos da interação.

2.2 WINDOWS PHONE

Windows Phone é o sucessor da linha Windows Mobile da Microsoft. É baseado no Windows CE 6, enquanto que as versões do Windows Mobile são baseadas no Windows CE 5 (ZIEGLER, 2010, tradução nossa).

Um *reboot* é o que a Microsoft tem iniciado com sua nova abordagem para o mercado de celulares. Com sua aparência limpa, fontes marcantes, e novo paradigma de organização, o Microsoft Windows Phone não só representa uma ruptura com o passado do Windows Mobile mas também o diferencia dos outros *smartphones* que estão no mercado (PETZOLD, 2010, p. 2, tradução nossa).

> Microsoft Windows Phone é uma grande plataforma móvel porque oferece todas as funções de um *smartphone* moderno, incluindo GPS, e-mail, SMS, câmera e tocador de música, e também provém um framework de desenvolvimento que permite que milhares de desenvolvedores .NET aprendam e desenvolvam no Windows Phone rapidamente. O Windows Phone também oferece capacidade de multitoque na tela, uma bela interface que implementa o novo e moderno design chamado Metro. Além disso, a Microsoft também introduziu o conceito de *hub*, como por exemplo o *hub* chamado People, que é um local onde os usuários podem armazenar todos os seus contatos e conexões sociais juntos. (LEE; CHUVYROV, 2012, p. 4, tradução nossa).

"Uma das grandes coisas sobre o Windows Phone é que a Microsoft impõe especificações de hardware sobre os fabricantes, fazendo mais fácil para você desenvolver um aplicativo sem se preocupar com escrita de código específico" (LEE; CHUVYROV, 2012, p. 4, tradução nossa). Segundo Petzold (2010, p. 4, tradução nossa), "muitos outros recursos de hardware são de existência garantida em cada dispositivo". Desta forma, "para todos os futuros lançamentos do Windows Phone, está garantido que o aplicativo de hoje funcionará independentemente da marca do dispositivo móvel, desde que ele execute Microsoft Windows Phone." (LEE; CHUVYROV, 2012, p. 4, tradução nossa).

2.2.1 Visual Studio

O Visual Studio é o ambiente de desenvolvimento padrão utilizado pela Microsoft em que os desenvolvedores podem criar projetos utilizando a plataforma .NET com todas as tecnologias da Microsoft (LECHETA, 2013).

Uma versão específica do Visual Studio é utilizada para criar aplicativos para Windows 8. As aplicações para Windows 8 são chamadas de *Windows Store Apps*, e para criá-las é utilizado o *Microsoft Visual Studio Express 2012 for Windows 8* (LECHETA, 2013).

> As ferramentas incluem um editor de código completo, um depurador poderoso, um criador de perfil preciso, um abrangente suporte de linguagens que você pode usar para criar aplicativos que você escreve em HTML5/JavaScript, C++, C# ou Visual Basic. O Visual Studio Express 2013 para Windows também inclui ferramentas para o desenvolvimento do Windows Phone 8.0, um simulador de dispositivo que você pode usar para testar os aplicativos para Windows Store em vários tipos de dispositivo, e emuladores Windows Phone que você pode usar para testar como seus aplicativos para Windows Phone serão executados em diferentes dispositivos. (VISUAL STUDIO, 2013).

2.2.2 Tipos de aplicativos

O tipo mais comum de aplicativo para o Windows Phone 7.x é o tipo *eXtensible Application Markup Language* (XAML). São exclusivamente escritos em XAML e com a linguagem C# ou Visual Basic, e tem suporte às *Application Programming Interface* (API) Microsoft .NET, Windows Phone API e WinPRT API (WHITECHAPEL; MCKENNA, 2012, p. 8, tradução nossa).

Há também o tipo *Mixed Mode*, que segue a estrutura dos aplicativos XAML, com a adição do suporte a C/C++, mas permite a inclusão de código nativo envolto em um componente WinPRT. Isso serve bem para aplicativos onde se quer reusar uma biblioteca nativa já existente, em vez de reescrever o código. Também é útil para os casos em que se quer escrever a maior parte do aplicativo em código nativo (incluindo gráficos Direct3D) mas também precisa acessar alguns dos recursos que só estão disponíveis para aplicativos XAML. As APIs suportadas são .NET Windows Phone API, WinPRT API e Win32/COM API (WHITECHAPEL; MCKENNA, 2012, p. 8, tradução nossa).

Para o desenvolvimento de jogos existe o tipo Direct3D, que oferece a capacidade de extrair o máximo do hardware do telefone. Também, porque é baseado no modelo de aplicativos do Windows, oferece o maior grau de compartilhamento de código entre Windows e Windows Phone. Este tipo tem suporte para a WinPRT API e Win32/COM API (WHITECHAPEL; MCKENNA, 2012, p. 8, tradução nossa).

2.2.3 Processamento em segundo plano

Quando a execução vai para segundo plano no dispositivo móvel, os usuários frequentemente têm objetivos conflitantes. Por um lado, eles querem que seus aplicativos continuem fornecendo informações mesmo quando eles não estão interagindo diretamente com eles. Querem fazer *streaming* de música da *web*, atualizar seu *Live Tile* com as últimas informações sobre o tempo, ou instruções de navegação *turn-by-turn*. Por outro lado, também querem que seus telefones durem até o fim do dia sem acabar a bateria e que os aplicativos

que estão em primeiro plano não fiquem lentos por causa dos que estão em segundo plano (WHITECHAPEL; MCKENNA, 2012, p. 8 - 9, tradução nossa).

O Windows Phone oferece um conjunto de serviços que podem realizar tarefas comuns em favor dos aplicativos. Como o *Background Transfer Service* (BTS), *Alarms*, *Background Audio Agents*, *Scheduled Tasks* e *Continuous Background Execution for Location Tracking* (WHITECHAPEL; MCKENNA, 2012, p. 9, tradução nossa).

O BTS torna possível para os aplicativos fazerem transferências HTTP usando a mesma infraestrutura que o sistema operacional usa. Este serviço garante que os *downloads* continuem mesmo que o aparelho seja reiniciado e que os *downloads* não impactarão no tráfego de rede do aplicativo que está em primeiro plano (WHITECHAPEL; MCKENNA, 2012, p. 9, tradução nossa).

Com a API *Alarms* é possível criar lembretes para cenários específicos que fornecem *links* para voltar ao aplicativo em uma parte específica. Por exemplo, um aplicativo de receitas pode prover um mecanismo para adicionar um alarme que toca quando é para desligar o forno e que também fornece um *link* que leva o usuário ao próximo passo da receita. O *Alarms* não somente remove a necessidade de o aplicativo ficar em execução em segundo plano, mas também possibilita o uso da interface de notificação do Windows Phone (WHITECHAPEL; MCKENNA, 2012, p. 9, tradução nossa).

Background Audio Agents é uma solução simples para permitir que aplicativos possam executar áudio mesmo estando em segundo plano. E para evitar que cada aplicativo crie sua própria infraestrutura, o que pode tornar a experiência do usuário confusa além de causar impacto na performance. O Windows Phone reutiliza a infraestrutura de áudio existente e invoca o código do aplicativo só para fornecer o gerenciamento de *playlist* ou de *streaming*. Desta forma o áudio pode ser colocado em um pequeno processo que fica rodando em segundo plano para preservar o processo principal e a vida da bateria (WHITECHAPEL; MCKENNA, 2012, p. 9, tradução nossa).

Scheduled Tasks oferece solução mais genérica para processamento em segundo plano, mas dentro dele há dois tipos que o aplicativo pode criar. O primeiro tipo é o *Periodic Tasks*, que executa por um período curto de tempo em um intervalo regular, a configuração atual é 25 segundos a cada 30 minutos, desde que o telefone não esteja em modo de economia de energia. É planejado para pequenas tarefas que se beneficiem de execução frequente. Como por exemplo, um aplicativo de previsão de tempo que pode buscar as últimas informações em um *web service*. O segundo tipo é o *Resource-intensive Agents*, que podem executar por um período longo, mas não podem ter uma agenda prevista. Por poderem causar grande impacto

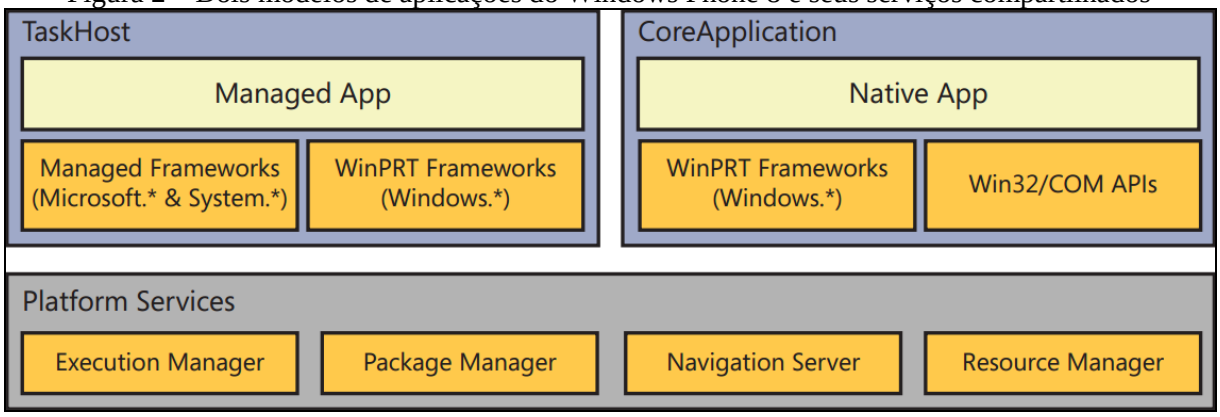
na performance, eles somente executam quando o aparelho está plugado na tomada, quase totalmente carregado, conectado no Wi-Fi, e não em uso. *Resource-intensive Agents* destinam-se para as operações mais exigentes como a sincronização de banco de dados com um servidor (WHITECHAPEL; MCKENNA, 2012, p. 10, tradução nossa).

Os aplicativos de navegação *turn-by-turn* geralmente precisam constantemente receber e processar informação atualizada em períodos de poucos segundos para determinar se o usuário precisa virar para a esquerda ou direita. Eles precisam também oferecer um mapa mostrando a rota completa para o destino, o tempo e distância para chegar. Para este problema o Windows Phone 8 introduziu um conceito conhecido como *Continuous Backgroud Execution* (CBE), que se refere à capacidade do aplicativo atual continuar executando mesmo que o usuário tenha saído para outro aplicativo, mas com um conjunto restrito de APIs (WHITECHAPEL; MCKENNA, 2012, p. 10, tradução nossa).

2.2.4 Arquitetura

O Windows Phone está dividido em dois modelos de aplicações. Como é possível ver na Figura 2, há o TaskHost que representa as aplicações XAML, que tem sido o modelo principal desde o lançamento do Windows Phone 7, e o CoreApplication, um novo modelo de aplicação para Windows Phone, que é um subconjunto do novo modelo de aplicação do Windows 8. Esse modelo somente suporta aplicativos nativos que usam Direct3D para interface (WHITECHAPEL; MCKENNA, 2012, p. 6, tradução nossa).

Figura 2 – Dois modelos de aplicações do Windows Phone 8 e seus serviços compartilhados



Fonte: Whitechapel e McKenna (2012).

Os dois modelos dependem de um conjunto compartilhado de serviços da plataforma. A maior parte dos aplicativos só tem contato com esses serviços de forma indireta, mas eles têm um papel importante em assegurar que os aplicativos funcionem corretamente (WHITECHAPEL; MCKENNA, 2012, p. 7, tradução nossa).

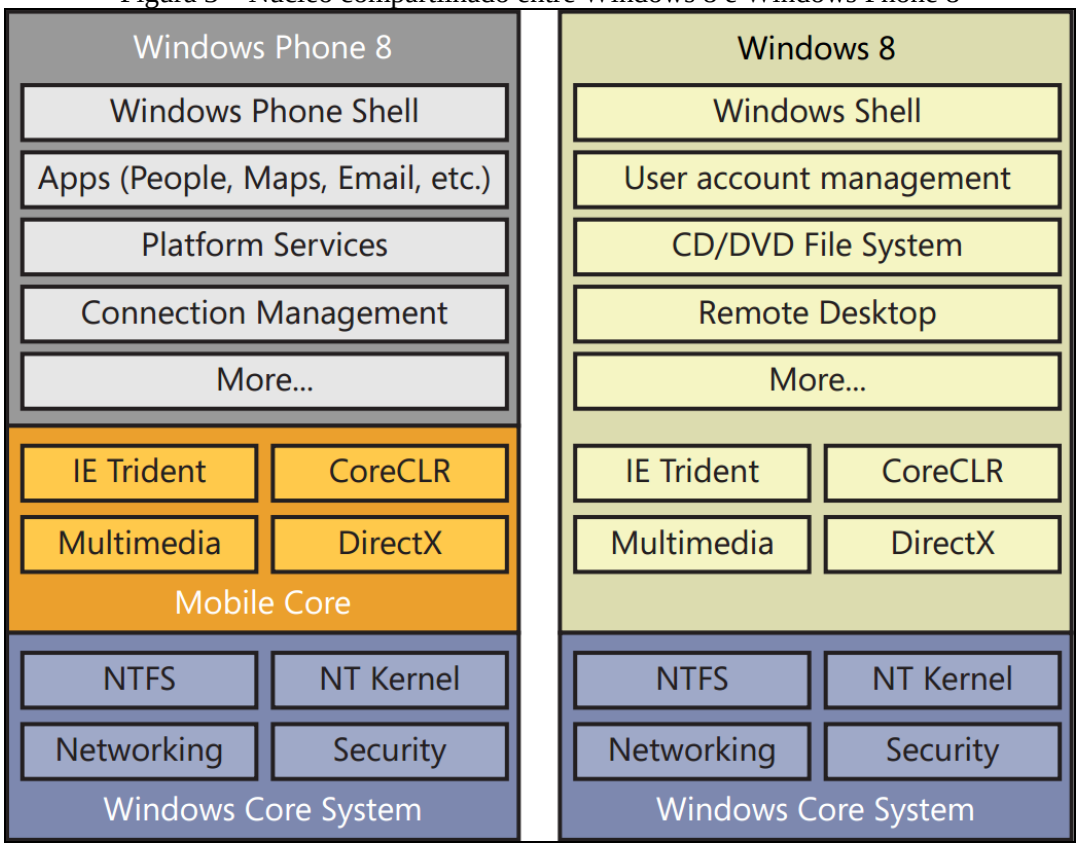
Segundo Whitechapel e McKenna (2012, p. 7) a função de cada um destes serviços é:

a) *Package Manager*: é responsável por instalar, desinstalar aplicativos e manter todos os metadados durante o ciclo de vida do aplicativo. Ele mantém registros de quais aplicativos estão instalados e licenciados, também persiste as informações sobre qualquer aplicativo que o usuário tenha adicionado a tela inicial;
b) *Execution Manager*: controla toda lógica associada com o ciclo de vida de execução de um aplicativo. Ele cria o processo *host* para que o aplicativo execute e chama os eventos associados com a inicialização, finalização e desativação do aplicativo. Também faz isso para os processos que estão em segundo plano, o que também inclui o agendamento destas tarefas;
c) *Navigation Server*: gerencia todo o movimento entre aplicativos que estão em primeiro plano. Quando o usuário inicia um aplicativo que está na tela inicial, o *Navigation Server* é responsável por transmitir a intenção para o *Execution Manager* de mover do aplicativo "Tela Inicial" para o aplicativo escolhido. Da mesma forma quando o usuário pressiona e segura o botão voltar e seleciona um aplicativo iniciado anteriormente, o *Navigation Server* deve informar ao *Execution Manager* qual aplicativo deve ser reativado;
d) *Resource Manager*: é responsável por monitorar o uso dos recursos do sistema, especialmente o processador e memória, por todos os processos ativos e forçar um conjunto de restrições para assegurar que o telefone esteja sempre rápido e responsivo. Caso um processo exceda os recursos disponíveis para ele, o *Resource Manager* o finalizará para manter o desempenho do telefone.

2.2.4.1 Windows e Windows Phone

O Windows 8 e Windows Phone 8 adotaram um núcleo compartilhado que contém dois componentes distintos. Como é possível ver na Figura 3, a parte mais baixa é o Windows Core System, que tem as funções mais básicas do sistema operacional. Essas funções são a base comum para computadores convencionais e *smartphones*. Acima do Windows Core System está o Mobile Core, que inclui componentes como Multimedia, CoreCLR, e Trident, que é a *engine* de renderização do Internet Explorer. O Windows também tem estes componentes, porém faz parte de um conjunto maior de funções (WHITECHAPEL; MCKENNA, 2012, p. 14, tradução nossa).

Figura 3 – Núcleo compartilhado entre Windows 8 e Windows Phone 8



Fonte: Whitechapel e McKenna (2012).

Core System e Mobile Core só estão representando o alinhamento do Windows e Windows Phone, pois os dois sistemas operacionais estão rodando exatamente o mesmo código. Há inúmeras outras áreas onde as APIs e comportamentos são compartilhados, mas com pequenas diferenças de implementação devido aos diferentes tipos de ambientes. Por exemplo, a API de localização no Windows Phone automaticamente incorpora dados dos usuários sobre a posição de torres de celular e pontos de acesso a redes sem fio para melhorar as leituras de localização. Essa é uma otimização que não faz parte do *framework* de localização do Windows 8 (WHITECHAPEL; MCKENNA, 2012, p. 14, tradução nossa).

2.2.5 Hardware

Segundo Thurrott (2014, p. 16 - 19), os principais componentes de hardware do Windows Phone são os seguintes:

a) processador: todos os *smartphones* da linha vêm com um processador baseado em ARM, o número de núcleos e *clock* de velocidade do processador variam entre dois núcleos e 1 GHz nos modelos de nível de entrada até quatro núcleos e 1.5 GHz no nível médio e alto de telefones. Em teoria, mais núcleos e maiores *clocks* são melhores, mas no uso normal não são notáveis as diferenças. Porém, alguns

usuários como os *gamers* se beneficiarão dos processadores de alto nível. Portanto, o processador é raramente um diferencial para o comprador normal;

b) gráficos: todos os processadores usados no Windows Phone tem suporte ao DirectX com aceleração de hardware por Direct3D;

c) tamanhos de tela: os aparelhos vêm em uma variedade de tamanhos como é possível ver na Figura 4. No nível de entrada o tamanho da tela varia normalmente entre 3.8 a 4.3 polegadas, no nível médio o tamanho é somente 4.3 polegadas, e no alto nível normalmente são oferecidas as maiores telas de 4.5 até 6 polegadas. Telas grandes requerem grandes aparelhos, então o pagamento por isso normalmente é um aparelho maior, mais pesado e mais grosso;

Figura 4 – Diferentes tamanhos de telefones

Fonte: Thurrott (2014).

d) resoluções de tela: Windows Phone suporta quatro resoluções de tela, WVGA (480 x 800), 720p (720 x 1280), WXGA (768 x 1280) e 1080p (1080 x 1920). Como uma regra, as maiores resoluções tendem a estar disponíveis nas maiores telas, o que de alguma forma atenua a presumida vantagem dos pixels extras em aparelhos de alto nível. Mas ao contrário da sabedoria convencional, mesmo a supostamente baixa resolução WVGA tem ótima qualidade no Windows Phone 8, com texto nítido e gráficos limpos, graças ao design independente de resolução do sistema operacional;

e) multitoque: as telas têm multitoque com pelo menos quatro toques ao mesmo tempo;

f) tecnologias de tela: os fabricantes são livres para escolher a tecnologia usada na

tela. As telas podem variar do *Liquid-Crystal Display* (LCD) comum até o *Active-Matrix Organic Light-Emitting Diode* (AMOLED), que oferecem cores mais vivas e pretos profundos. Muitas também utilizam alguma das tecnologias *Gorilla Glass*, que ajudam a proteger a tela de arranhões e, muitas vezes, de quebrar depois de uma queda. E certas telas podem ainda ser tocadas e usadas enquanto você está usando luvas, um bom adicional para aqueles que estão em climas frios;

g) memória: aparelhos Windows Phone vêm com um mínimo de 512 MB de memória para os *devices* com tela WVGA (800 x 480), 1 GB de memória para as telas com 720p (1280 x 720) e WXGA (1280 x 768), e 2 GB para telas 1080p. Como uma regra geral, são 512 MB de memória em telefones de baixo nível, 1 GB em telefones de nível médio, e 2 GB em telefones de alto nível;

h) armazenamento interno: o sistema operacional requer que o *smartphone* inclua um mínimo de 4 GB de memória flash, ou armazenamento interno. De forma prática, são 8 GB de armazenamento interno em aparelhos de baixo nível, 16 GB em aparelhos de nível médio e 32 GB em aparelhos de alto nível. Na verdade 8 GB é suficiente para uso leve, especialmente se o *device* é expansível via micro-SD;

i) entrada micro-SD: *Smartphones* Windows Phone opcionalmente podem prover expansão do armazenamento interno de até 64 GB através de entradas que suportam cartões de memória micro-SD ou micro-SDXC;

j) rede: todos os aparelhos Windows Phone suportam rede sem fio 802.11b/g e Bluetooth. Enquanto que redes 802.11n e 802.11ac são opcionais;

k) sensores: o Windows Phone vem com acelerômetro, sensor de proximidade, magnetômetro e sensor de luz ambiente, assim como GPS com *Assisted Global Navigation Satellite System* (A-GNSS). Todos, mas só os mais recentes no caso dos aparelhos de nível baixo, tem bússola. E os fabricantes também tem a opção de incluir giroscópio, que permite obter a posição do aparelho de forma mais precisa e é muito útil para jogos e outros aplicativos;

l) NFC: *Near Field Communication* (NFC) também pode estar disponível em alguns aparelhos, normalmente nos de alto nível. Pode ser usado para fazer pagamentos seguros, compartilhar itens como fotos, contatos e documentos, e também para parear dispositivos Bluetooth mais facilmente;

m) conexões: todos os Windows Phone têm uma porta micro-USB 2.0 para carregar e conectar ao computador, tem um conector para fones de ouvido e alto falantes, e também suportam fones com três botões de controle de mídia;

n) botões: há seis botões: ligar, aumentar volume, diminuir volume, voltar, iniciar e pesquisa, sendo que os últimos três podem ser capacitivos ou implementados via *software*;
o) câmera: os *smartphones* têm uma câmera principal atrás do aparelho, e os fabricantes podem adicionar uma segunda câmera na frente. Ambas devem ser ao menos de qualidade VGA (640 x 480), mas na maioria dos aparelhos de baixo nível a câmera traseira tem 5 *megapixels* ou mais e nos de nível médio e alto 8 megapixels ou mais. As câmeras frontais tem resolução entre VGA e HD. A maioria das câmeras usadas em Windows Phone são consideradas de boa qualidade, enquanto que a maioria da linha Lumia são consideradas de alta qualidade em relação a todos os tipos de *smartphones* no mercado;
p) carregamento sem fio: alguns dos aparelhos de alto nível suportam o padrão de carregamento sem fio Qi, enquanto que alguns aparelhos de nível médio tem capas que adicionam esta capacidade;
q) cartões SIM: existem aparelhos com suporte para dois cartões *Subscriber Identity Module* (SIM).

2.2.6 Lumia

Há fabricantes que fazem aparelhos excelentes, mas a maior parte de seu lucro e vendas é com aparelhos Android e consideram o Windows Phone um projeto menor. Neste ambiente, somente a Microsoft com a linha Lumia está totalmente empenhada no Windows Phone (THURROTT, 2014, p. 19, tradução nossa).

Aparelhos Lumia vêm com uma variedade de aplicativos de localização da marca HERE, incluindo o HERE Maps, HERE Drive e Drive+, HERE City Lens e HERE Transit. Esta tecnologia é tão boa que a Microsoft fez o HERE a plataforma de localização para todos os aparelhos Windows Phone. Então esta tecnologia é usada no Bing Maps em aparelhos que não são Lumia (THURROTT, 2014, p. 20, tradução nossa).

Enquanto que alguns aparelhos de outras marcas oferecem câmeras decentes, as dos Lumia são de alto nível, mesmo quando comparado ao Android e iPhone, especialmente em condições de pouca luz (THURROTT, 2014, p. 20, tradução nossa).

Além do HERE Maps, Lumia também vem com uma seleção de aplicativos exclusivos e jogos, provendo para os usuários uma grande variedade de conteúdo que não está disponível para os Windows Phone fabricados por outras empresas (THURROTT, 2014, p. 20, tradução nossa).

A Microsoft é a única fabricante a dar suporte a toda sua plataforma com uma grande variedade de acessórios. Isso inclui alto falantes com carregador sem fio interno, *headsets* e fones, carregadores sem fio de vários tipos, carregadores portáteis *Universal Serial Bus* (USB), carregadores para carro, capas especiais, e muito mais (THURROTT, 2014, p. 20, tradução nossa).

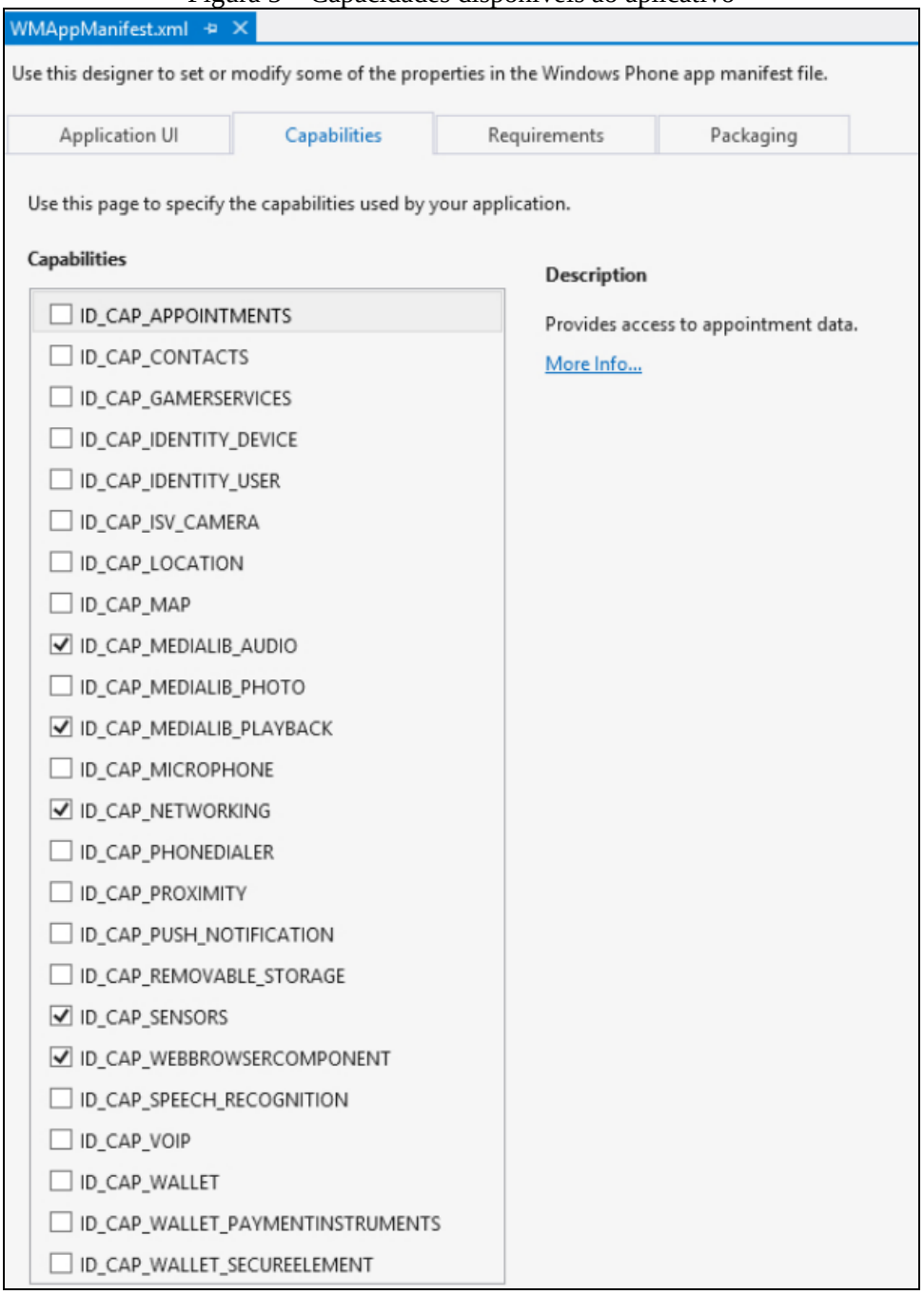
2.2.7 Câmara de segurança

A segurança do Windows Phone é baseada na noção de câmara de segurança, que são contêineres isolados em que os processos são criados e executados. O sistema garante os direitos a uma câmara baseado no princípio de segurança de longa data do menor privilégio, que garante que um aplicativo não deve receber direitos além do estritamente necessário para fazer suas funções declaradas. Por exemplo, o aplicativo de e-mail não deve ter a habilidade de iniciar a câmera e tirar uma foto (WHITECHAPEL; MCKENNA, 2012, p. 10 - 11, tradução nossa).

O funcionamento deste princípio ocorre da seguinte forma: cada câmara de segurança inicia com um conjunto limitado de privilégios, o suficiente para fazer, por exemplo, uma calculadora ou um jogo simples. Se um aplicativo quiser ter acesso a recursos que estão fora da sua câmara, como enviar tráfego pela rede ou ler os contatos do usuário, esses direitos têm de ser explicitamente concedidos através das *capabilities*. As *capabilities* agem como um conjunto de mecanismos de controle que cuidam do acesso ao uso de recursos sensíveis. O sistema tem que conceder explicitamente as capacidades para a câmara (WHITECHAPEL; MCKENNA, 2012, p. 11, tradução nossa).

Os desenvolvedores encontram essas capacidades diretamente quando estão construindo seus aplicativos, porque para acessar qualquer recurso que precisa de privilégio é necessário incluir a capacidade no arquivo *manifest* do aplicativo (WHITECHAPEL; MCKENNA, 2012, p. 11, tradução nossa). Na Figura 5 é possível ver a página gráfica que mostra o arquivo *manifest* e as capacidades disponíveis.

Figura 5 – Capacidades disponíveis ao aplicativo



Fonte: Whitechapel e McKenna (2012).

As capacidades listadas no arquivo *manifest* são traduzidas de forma que usuário consiga ler cada item quando ele estiver visualizando a página detalhada do aplicativo na loja. Então o usuário é que decide se instala um aplicativo que requer certas capacidades. Por exemplo, o usuário provavelmente espera que um aplicativo que o ajuda a encontrar cafeterias vai precisar acessar a sua localização, mas provavelmente vai achar estranho que uma calculadora precise do mesmo recurso (WHITECHAPEL; MCKENNA, 2012, p. 12, tradução nossa). A Figura 6 mostra a página do aplicativo onde é possível ver as capacidades de forma que o usuário entenda.

Figura 6 – Página de aplicativo na loja

Fonte: Whitechapel e McKenna (2012).

2.3 TRABALHOS CORRELATOS

Nesta seção são apresentados dois trabalhos correlatos na área de força de vendas em dispositivos móveis. Um protótipo desenvolvido em um Trabalho de Conclusão de Curso (TCC) e um aplicativo comercial.

2.3.1 Protótipo de aplicativo de força de vendas para dispositivos móveis baseados na plataforma Android

O objetivo deste aplicativo é facilitar as operações de vendas, controle de estoque, demonstração de produtos e cadastros da empresa Sold Out (OLIVEIRA, 2014), portanto permite visualizar, cadastrar e editar produtos, clientes e estoque e executar vendas. A Figura 7 mostra as telas de menu, cadastro de produtos, cadastro de clientes e vendas do aplicativo.

No desenvolvimento do servidor, foi utilizado um *framework* para persistência dos dados, chamado de *Java Persistence API* (JPA). E para realizar a integração entre o servidor e o aplicativo, foi utilizado o modelo de *web services* REST. Enquanto que o desenvolvimento

do aplicativo foi realizado com a linguagem Java no Eclipse com o *plugin* ADT (OLIVEIRA, 2014).

Figura 7 – Algumas telas do protótipo

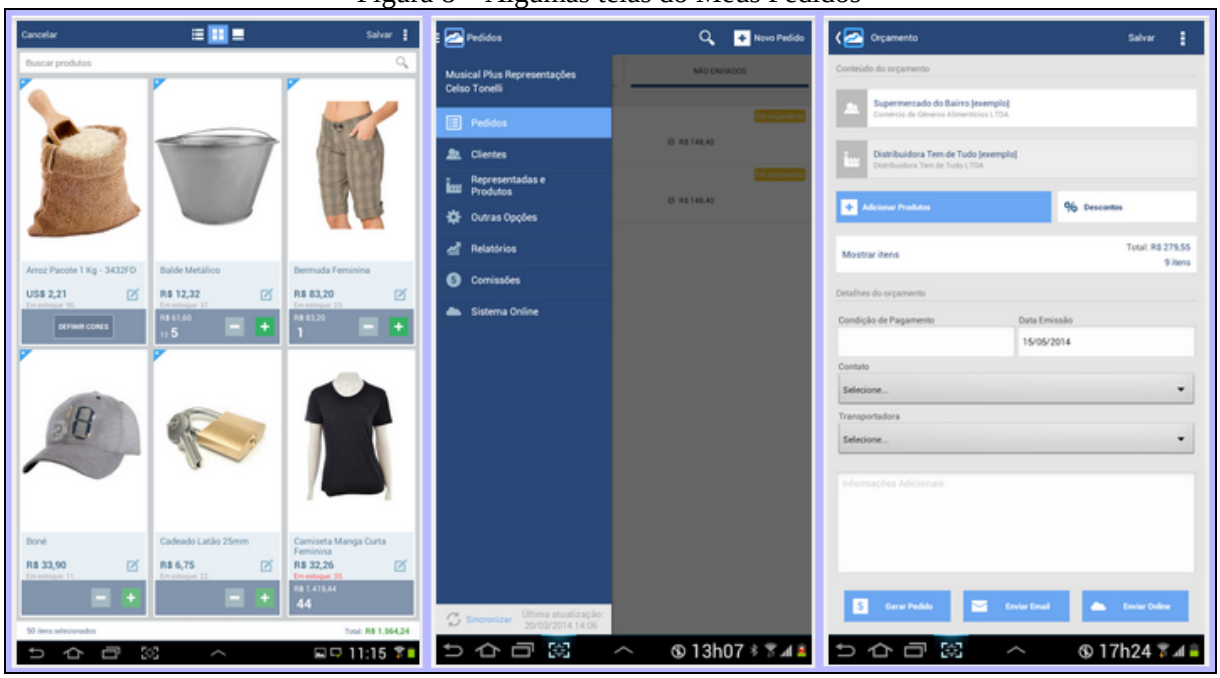
Fonte: Oliveira (2014).

2.3.2 Meus Pedidos

Meus Pedidos é um sistema Android feito para o representante comercial e empresas que vendem através de representantes. Permite a emissão de pedidos no *tablet* e *smartphone* através de catálogos de produtos com fotos, sem a necessidade de conexão com a internet, além de consultar clientes, produtos, preços e o histórico completo de pedidos de cada cliente (GOOGLE PLAY, 2014). A Figura 8 mostra a tela de seleção de produtos do pedido, o menu e a tela de cadastro de pedido do aplicativo.

O aplicativo é conectado com um sistema *on-line* que possui relatórios de vendas, comissões, clientes inativos, e é um complemento para gerenciar a equipe de vendedores, tabelas de preços e controlar os pedidos (GOOGLE PLAY, 2014).

Figura 8 – Algumas telas do Meus Pedidos



Fonte: Google Play (2014).

3 DESENVOLVIMENTO

Neste capítulo são detalhados os requisitos trabalhados, a especificação, a implementação e os resultados obtidos.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

A partir da observação das formas com que os vendedores da equipe de força de vendas interagiam com a empresa Proagro Agropecuária, foram elencados alguns objetivos para este aplicativo. Portanto, para atender aos objetivos específicos deste trabalho, o aplicativo proposto deve:

a) permitir inserir e alterar clientes (Requisito Funcional - RF);
b) permitir inserir pedidos (RF);
c) permitir inserir e alterar anotações em pedidos existentes (RF);
d) permitir visualizar informações completas dos pedidos (RF);
e) permitir visualizar relatórios de contas a receber, situação de estoque, tabela de preços e situação de entrega de pedidos (RF);
f) permitir realizar sincronização tanto automaticamente quanto sob demanda (RF);
g) executar em sistema Windows Phone (Requisito Não Funcional – RNF);
h) ser implementado utilizando o ambiente Visual Studio e linguagem C# (RNF);
i) manter sincronização com o servidor através de um *web service* (RNF).

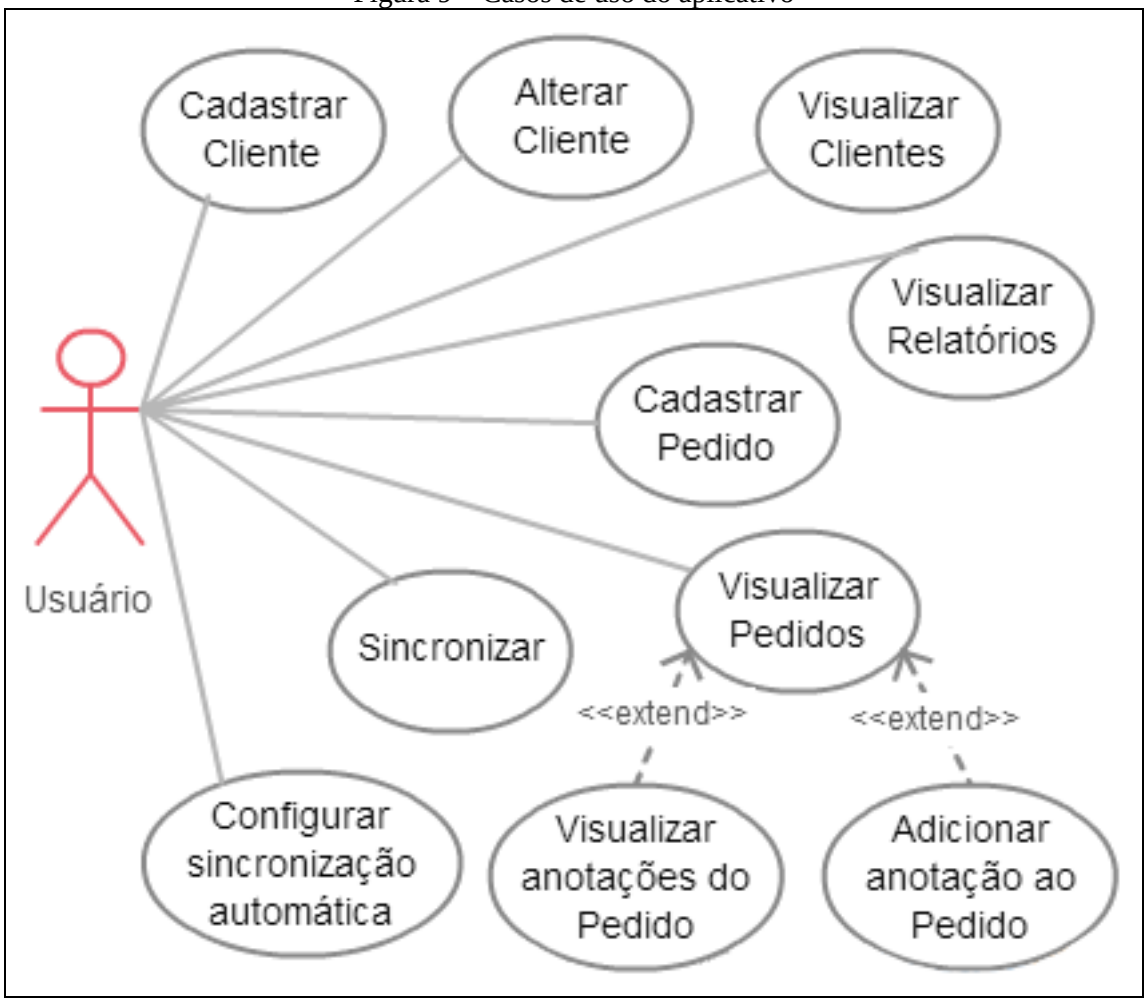
3.2 ESPECIFICAÇÃO

A seguir são apresentados o diagrama de casos de uso, o diagrama de atividades e os modelos de entidade relacionamento (MER).

3.2.1 Diagrama de casos de uso

A Figura 9 ilustra o diagrama de casos de uso do aplicativo. Nele verificam-se as funcionalidades que o ator `Usuário` tem acesso.

Figura 9 – Casos de uso do aplicativo



A seguir é apresentada uma breve descrição de cada caso de uso:

a) `Cadastrar cliente`: permite ao usuário cadastrar clientes;

b) `Alterar cliente`: permite ao usuário alterar clientes já cadastrados;

c) `Visualizar clientes`: permite ao usuário visualizar o cadastro de clientes que já estão cadastrados;

d) `Cadastrar pedido`: permite ao usuário cadastrar pedidos;

e) `Visualizar pedidos`: permite ao usuário visualizar pedidos já cadastrados;

f) `Adicionar anotação ao pedido`: dentro da visualização de pedido o usuário pode cadastrar uma anotação a um pedido;

g) `Visualizar anotações do pedido`: dentro da visualização de pedido o usuário pode visualizar as anotações já cadastradas;

h) `Visualizar relatórios`: permite ao usuário gerar relatórios;

i) `Sincronizar`: permite ao usuário acionar a sincronização dos dados com o servidor;

j) `Configurar sincronização automática`: permite ao usuário ativar a

sincronização automática, que acontece a cada trinta minutos quando o aplicativo não está ativo.

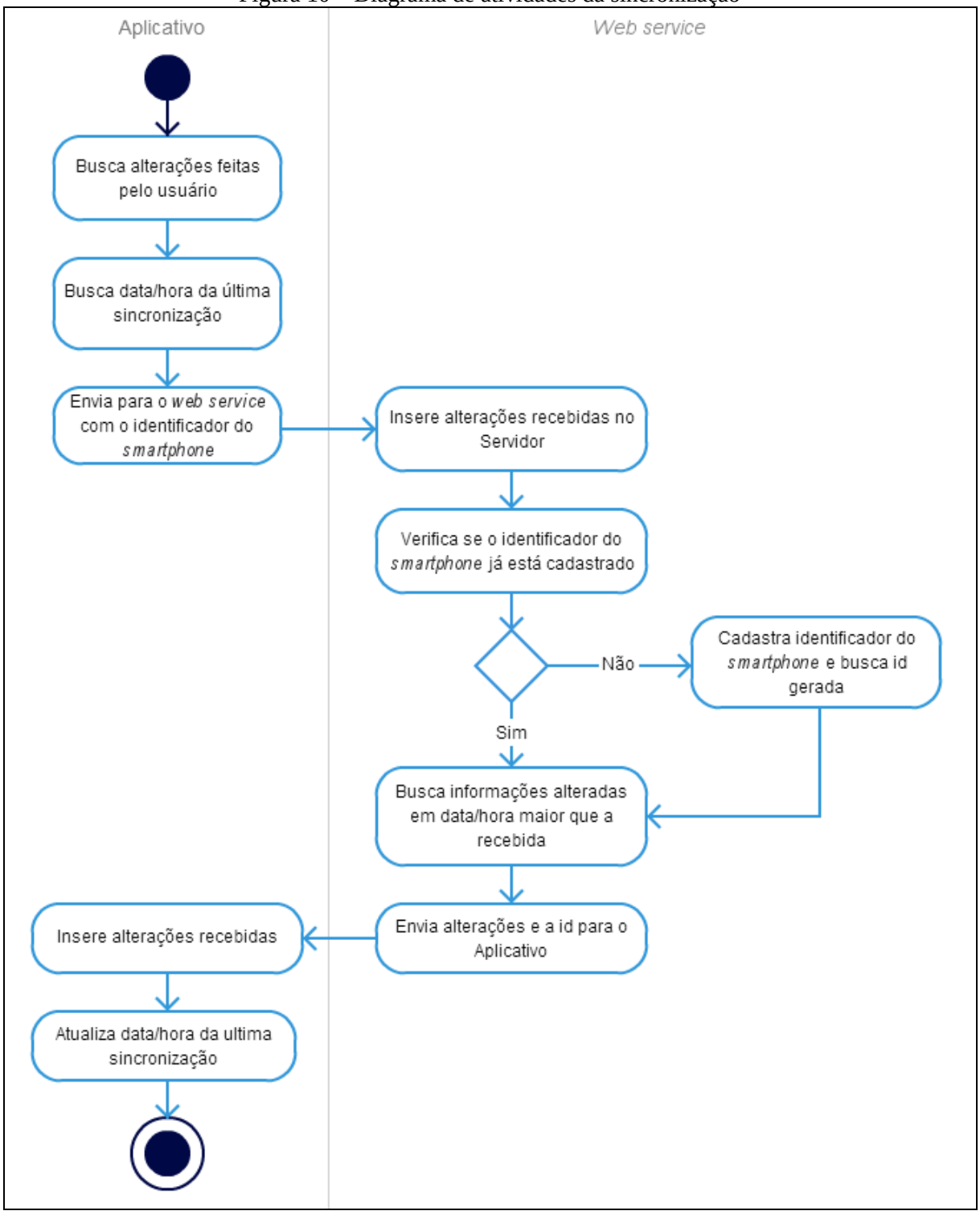
3.2.2 Diagrama de atividades

Na Figura 10 é apresentado o diagrama de atividades que exibe a sequência da sincronização dos dados do aplicativo com o servidor. Ao iniciar o processo, o aplicativo busca as alterações feitas nos dados pelo usuário desde a última sincronização e busca a data e hora da última sincronização. Em seguida envia ao *web service* os dados buscados junto com o identificador único do *smartphone*, que é um código gerado pelo próprio sistema operacional e é diferente para cada aplicativo instalado e que nunca se repete mesmo entre *smartphones*, e aguarda a resposta.

Ao receber os dados enviados pelo aplicativo, o *web service* executa as alterações do usuário que já estão no formato de comandos *Structured Query Language* (SQL). Após isso verifica se o identificador único já existe, e se não existir o cadastra. Por último, busca os dados alterados no servidor em data e hora maior que a última sincronização do aplicativo e os envia como resposta à chamada do *web service* junto com o identificador cadastrado para o identificador único.

Ao receber a resposta o aplicativo atualiza os dados com as informações recebidas e também atualiza a data e hora da última sincronização.

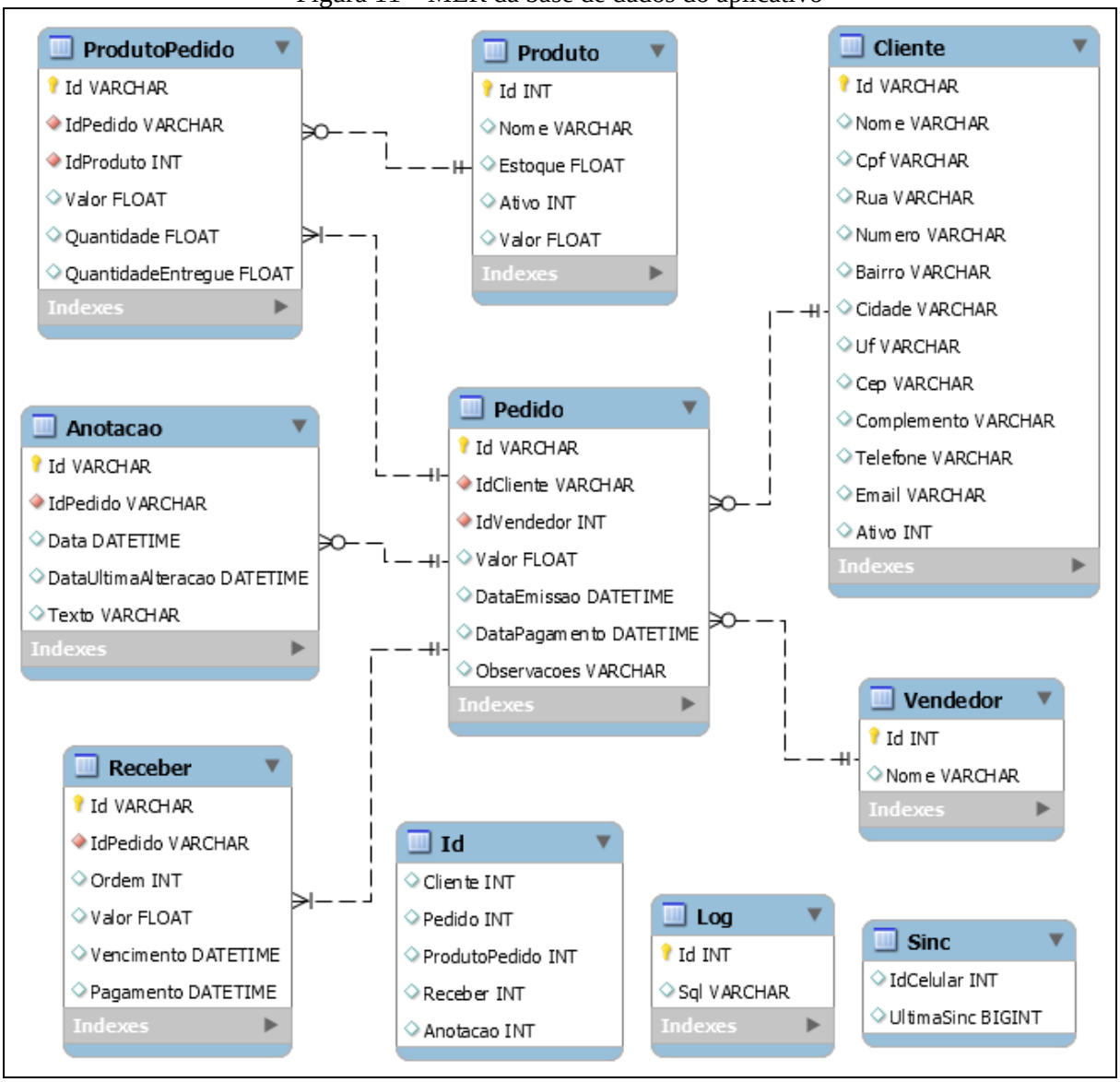
Figura 10 – Diagrama de atividades da sincronização



3.2.3 Modelo de Entidade Relacionamento

A modelagem da base de dados utilizada para o aplicativo está representada na Figura 11.

Figura 11 – MER da base de dados do aplicativo



A seguir é apresentada uma descrição das tabelas da base de dados do aplicativo:

a) `Cliente`: é responsável por armazenar os clientes cadastrados, com seu nome, Cadastro de Pessoas Física (CPF), endereço, informações de contato e se é um cadastro ativo;

b) `Pedido`: é responsável por armazenar os pedidos cadastrados, com seu valor, observações, data de emissão e data de pagamento, que é a data que o cliente terminou de pagar todas as parcelas do pedido. Além de fazer o vínculo com o cliente e o vendedor;

c) `Receber`: é responsável por armazenar as parcelas dos pedidos, com o valor, data de vencimento, data de pagamento e também qual é a numeração da parcela dentro do pedido através do campo `Ordem`. Além de fazer o vínculo com o pedido;

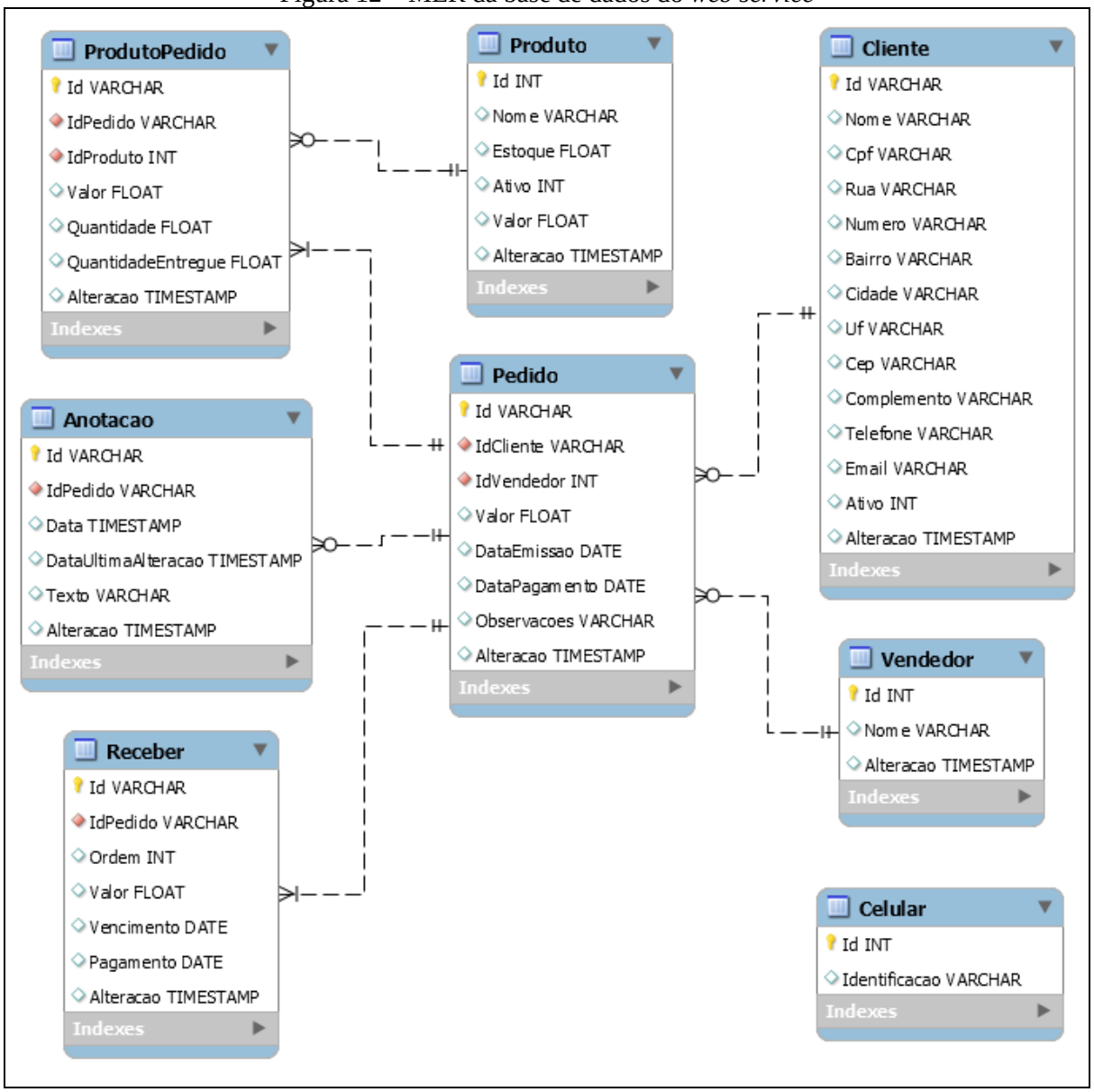
d) `Anotacao`: é responsável por armazenar as anotações feitas nos pedidos, com a

data de criação, data de alteração e o texto. Além de fazer o vínculo com o pedido;

e) `ProdutoPedido`: é responsável por armazenar os produtos que foram vendidos nos pedidos, com o valor, quantidade e quantidade entregue. Além de fazer o vínculo com o pedido e o produto;
f) `Produto`: é responsável por armazenar os produtos cadastrados, com seu nome, quantidade em estoque, valor atual e se é um cadastro ativo;
g) `Vendedor`: é responsável por armazenar os vendedores cadastrados;
h) `Id`: é responsável por manter as chaves primárias para as tabelas `Cliente`, `Pedido`, `ProdutoPedido`, `Receber` e `Anotacao`;
i) `Sinc`: é responsável por armazenar o identificador do *smartphone* gerado pelo servidor. Este identificador é concatenado com a chave primária das tabelas `Cliente`, `Pedido`, `ProdutoPedido`, `Receber` e `Anotacao`, com o objetivo de que as chaves nunca se repitam entre as informações cadastradas por *smartphones* diferentes. A tabela `Sinc` também é responsável por guardar a data e hora da última sincronização, esta informação é utilizada durante a sincronização para que o servidor saiba quais informações precisam ser atualizadas no aplicativo;
j) `Log`: é responsável por armazenar os comandos SQL correspondentes aos cadastros e alterações efetuadas no aplicativo. Durante a sincronização estes comandos são enviados ao servidor, que os executa em seu banco de dados.

A modelagem da base de dados utilizada para o *web service* está representada na Figura 12.

Figura 12 – MER da base de dados do *web service*



As tabelas `Cliente`, `Pedido`, `Receber`, `Anotacao`, `ProdutoPedido`, `Produto` e `Vendedor` do *web service* tem a mesma função que as tabelas de mesmo nome na base de dados do aplicativo. Entretanto, estas tabelas tem a adição do campo `Alteração` que guarda uma data e hora, este campo é atualizado sempre que alguma informação do registro é alterada. O campo `Alteração` é utilizado durante a sincronização para que o *web service* consiga comparar a data e hora da última sincronização do aplicativo e assim identificar quais são as informações que o aplicativo precisa receber. Por fim, a tabela `Celular` é responsável por armazenar o identificador único de cada *smartphone* e guardar os identificadores gerados para os *smartphones* no servidor.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

A implementação foi dividida em aplicativo e *web service*. Em ambos foi utilizado o ambiente Visual Studio 2013 e a linguagem C#. No aplicativo foi utilizado o banco de dados SQLite e no *web service* o PostgreSQL. Para o desenvolvimento do *web service* foi utilizado o *framework* Windows Communication Foundation (WCF).

Nos testes o *smartphone* Nokia Lumia 530 foi utilizado para o aplicativo e o IIS Express para o *web service*.

3.3.1.1 Sincronismo

O processo de sincronia do aplicativo com o servidor é feito em três etapas. A seguir estão descritas cada uma delas.

3.3.1.1.1 Primeira etapa

A primeira etapa acontece no aplicativo. No Quadro 1 é possível observar o trecho principal do método `Sincronizar` da classe `Sincronizacao`.

Quadro 1 – Trecho do método `Sincronizar`

```
1  List<Sinc> ls = BancoDeDados.Query<Sinc>("select * from Sinc");
2  Sinc ultSinc = ls.Count > 0 ? ls[0] : new Sinc();
3  atualizacoes = BancoDeDados.Query<Log>("select * from Log order by Id");
4  List<string> lista = new List<string>();
5  foreach(Log log in atualizacoes)
6  {
7      lista.Add(log.Sql);
8  }
9
10 TCCWSClient client = new TCCWSClient();
11 client.SincronizarCompleted += SincronizarCompleted;
12 client.SincronizarAsync(lista, ultSinc.getUltimaSinc(),
13         Windows.Phone.System.Analytics.HostInformation.PublisherHostId);
```

Nas linhas 1 e 2 é feita a busca da última sincronização. Em seguida, nas linhas 3 a 8 são buscadas todas as alterações feitas no aplicativo, que são comandos SQL. Após isso nas linhas 10 a 13 é informado qual é o método que receberá a resposta do *web service* e é feita a chamada informando a lista de alterações, a última sincronização e o identificador único do *smartphone*. Por fim, o aplicativo fica aguardando a resposta do *web service*.

3.3.1.1.2 Segunda etapa

A segunda etapa é o processamento dos dados informados pelo aplicativo dentro do *web service*. O processamento acontece no método `Sincronizar` da classe `Sincronizacao`. O Quadro 2 mostra o trecho de código onde o banco de dados do servidor é atualizado e onde é verificada a identificação do *smartphone*.

Quadro 2 – Trecho do método `Sincronizar`

```
1  foreach (string sql in atualizacoes)
2  {
3      NpgsqlCommand comando = new NpgsqlCommand(sql);
4      bd.NonQuery(comando);
5  }
6
7  Atualizacao atualizacao = new Atualizacao();
8  bool buscarIds = false;
9
10 #region Identificacao
11 NpgsqlCommand command = new NpgsqlCommand
12     ("SELECT id FROM celular WHERE identificacao = '" + identificacao + "'");
13 DataSet ds = bd.Query(command);
14 if (ds == null) return null;
15 DataTableReader dtr = ds.CreateDataReader();
16
17 if (dtr.Read())
18 {
19     atualizacao.idCelular = dtr.GetInt32(0);
20     if (ultimaSincronizacao.Ticks == 0)
21         buscarIds = true;
22 }
23 else
24 {
25     command = new NpgsqlCommand
26         ("INSERT INTO celular(identificacao) VALUES ('" + identificacao + "')");
27     bd.NonQuery(command);
28     command = new NpgsqlCommand
29         ("SELECT id FROM celular WHERE identificacao = '" + identificacao + "'");
30     ds = bd.Query(command);
31     if (ds == null) return null;
32     dtr = ds.CreateDataReader();
33     dtr.Read();
34     atualizacao.idCelular = dtr.GetInt32(0);
35 }
36 #endregion
```

Nas linhas de 1 a 5 é executado no banco de dados do servidor os comandos SQL enviados pelo aplicativo. Na linha 7 é criado um objeto `Atualizacao`, que será enviado como resposta a chamada do *web service*. Nas linhas de 11 a 15 é verificado se o *smartphone* que está solicitando a sincronização já está cadastrado.

No caso de já estar cadastrado, entra na linha 19 e informa o número de identificação do *smartphone* para o objeto `Atualizacao`. Em seguida nas linhas de 20 e 21 é identificado o caso de o aplicativo ter sido reinstalado, pois nesta situação é necessário verificar quais foram

os últimos sequenciais cadastrados pelo aplicativo como chave primária nas tabelas que ele pode alterar. Desta forma após uma reinstalação o aplicativo continuará a cadastrar as chaves a partir da sequência já existente.

Se o *smartphone* não estiver cadastrado, nas linhas de 25 a 34 o cadastrará e informará o número de identificação cadastrado ao objeto `Atualizacao`.

O Quadro 3 mostra um trecho do código que é executado após a verificação da identificação do *smartphone*. Nesta parte é feito a busca de dados que foram alterados no servidor após a última sincronização do aplicativo. No caso de nunca ter feito sincronização carregará todos os dados das tabelas do banco de dados.

Quadro 3 – Trecho do método `Sincronizar`

```
1  #region Pedido
2  command = new NpgsqlCommand
3      (@"SELECT id, numero, id_cliente, id_vendedor, valor,
4                data_emissao, data_pagamento, observacoes, alteracao
5         FROM pedido WHERE alteracao > @alt ORDER BY Id");
6  command.Parameters.Add("alt", NpgsqlDbType.Timestamp).Value = ultimaSincronizacao;
7  ds = bd.Query(command);
8  if (ds == null) return null;
9  dtr = ds.CreateDataReader();
10 List<PedidoWS> pedidos = new List<PedidoWS>();
11
12 while (dtr.Read())
13 {
14     pedidos.Add(new PedidoWS()
15     {
16         Id = dtr.GetString(0),
17         Numero = dtr.GetString(1),
18         IdCliente = dtr.GetString(2),
19         IdVendedor = dtr.GetInt32(3),
20         Valor = dtr.GetDecimal(4),
21         DataEmissao = dtr.GetDateTime(5),
22         DataPagamento = dtr.GetDateTime(6),
23         Observacoes = dtr.GetString(7)
24     });
25     if (atualizacao.dtAtualizado == null || atualizacao.dtAtualizado < dtr.GetDateTime(8))
26     {
27         atualizacao.dtAtualizado = dtr.GetDateTime(8);
28     }
29     if (buscarIds)
30     {
31         string[] aux = dtr.GetString(0).Split('/');
32         int id = Convert.ToInt32(aux[1]);
33         if (atualizacao.idCelular == Convert.ToInt32(aux[0])
34                 && (atualizacao.maxIdPedido == null || atualizacao.maxIdPedido < id))
35             atualizacao.maxIdPedido = id;
36     }
37 }
38 atualizacao.pedidos = pedidos;
39 #endregion
```

Nas linhas de 2 a 9 é feita a busca dos dados alterados ou inseridos na tabela `Pedido` após a última sincronização. Nas linhas de 14 a 24 é criado um objeto `PedidoWS` para cada linha buscada no banco de dados. Nas linhas de 25 a 28 é verificada a maior data e hora das

linhas carregadas, para ser utilizado como a data e hora da sincronização em execução. Caso seja necessário buscar os últimos sequenciais cadastrados, nas linhas de 31 a 35 verificará se o registro atual foi inserido pelo *smartphone* que está sincronizando. Se sim, guardará o sequencial se ele for maior que o atual. Por fim, coloca a lista de `PedidoWS` no objeto `Atualização`.

O processo feito no Quadro 3 para os pedidos também é feito de forma semelhante para os clientes, produtos, produtos do pedido, itens a receber dos pedidos, anotações e vendedores. Em seguida, o objeto `Atualizacao` é enviado ao aplicativo.

3.3.1.1.3 Terceira etapa

A terceira parte acontece quando o aplicativo recebe a resposta do *web service* e chama o método `SincronizarCompleted` da classe `Sincronizar`. No Quadro 4 é possível observar o tratamento dos comandos SQL enviados ao *web service* e do objeto `Atualizacao` recebido.

Quadro 4 – Trecho do método `SincronizarCompleted`

```
1  Atualizacao atualizacao = e.Result;
2  BancoDeDados.BeginTransaction();
3  BancoDeDados.DeleteAll<Log>();
4
5  #region Pedido
6  List<Pedido> pedidos = new List<Pedido>(atualizacao.pedidos.Count);
7  foreach (PedidoWS item in atualizacao.pedidos)
8  {
9      pedidos.Add(new Pedido()
10     {
11         Id = item.Id,
12         Numero = item.Numero,
13         IdCliente = item.IdCliente,
14         IdVendedor = item.IdVendedor,
15         Valor = item.Valor,
16         DataEmissao = item.DataEmissao,
17         DataPagamento = item.DataPagamento,
18         Observacoes = item.Observacoes
19     });
20 }
21
22 BancoDeDados.Atualiza<Pedido>(pedidos);
23 #endregion
```

Nas linhas de 1 a 3 é recebido o objeto `Atualizacao` do *web service* e é deletado todos os comandos SQL, pois já foram enviados ao *web service*. Em seguida, nas linhas de 6 a 20 é percorrida a lista de pedidos recebida e criado outra lista com os objetos correspondentes utilizados no aplicativo. Na linha 22 os pedidos são inseridos no banco de dados do aplicativo.

Os comandos executados nas linhas de 6 a 22 também são feitos de forma semelhante para os clientes, produtos, produtos do pedido, itens a receber dos pedidos, anotações e vendedores.

O Quadro 5 mostra o código executado após a atualização das tabelas principais. Dentro do mesmo método é feita a atualização dos sequenciais no caso de ser a primeira sincronização após uma reinstalação do aplicativo, além de gravar a data e hora da atualização e o identificador do *smartphone* gerado pelo *web service*.

Quadro 5 – Trecho do método `SincronizarCompleted`

```
1  if (atualizacao.maxIdAnotacao != null || atualizacao.maxIdCliente != null
2          || atualizacao.maxIdPedido != null
3          || atualizacao.maxIdProdutoPedido != null
4          || atualizacao.maxIdReceber != null)
5  {
6      Id id = new Id()
7      {
8          Anotacao = atualizacao.maxIdAnotacao ?? 0,
9          Cliente = atualizacao.maxIdCliente ?? 0,
10         Pedido = atualizacao.maxIdPedido ?? 0,
11         ProdutoPedido = atualizacao.maxIdProdutoPedido ?? 0,
12         Receber = atualizacao.maxIdReceber ?? 0
13     };
14     BancoDeDados.DeleteAll<Id>();
15     BancoDeDados.Insert<Id>(id);
16 }
17
18 Sinc sinc = new Sinc();
19 List<Sinc> ls = BancoDeDados.Query<Sinc>("select * from Sinc");
20 if (ls.Count > 0)
21 {
22     sinc.UltimaSinc = atualizacao.dtAtualizado.Ticks > ls[0].UltimaSinc
23             ? atualizacao.dtAtualizado.Ticks : ls[0].UltimaSinc;
24 }
25 else
26 {
27     sinc.UltimaSinc = atualizacao.dtAtualizado.Ticks;
28 }
29 sinc.IdCelular = atualizacao.idCelular;
30
31 BancoDeDados.DeleteAll<Sinc>();
32 BancoDeDados.Insert<Sinc>(sinc);
33 BancoDeDados.CommitTransaction();
```

As linhas de 1 a 16 são executadas se for a primeira sincronização após uma reinstalação, pois o *web service* terá informado dentro do objeto `Atualizacao` o maior sequencial cadastrado para cada tabela. Tendo informado, será gravado na tabela `Id`. Nas linhas de 18 a 33 é verificado se a data e hora da sincronia recebida é maior que a já existente no aplicativo. Pois no caso de o *web service* não ter nenhuma informação atualizada para

enviar ao aplicativo a data e hora serão nulas. Por fim, o identificador gerado pelo *web service* e a maior data e hora são gravadas.

3.3.2 Operacionalidade da implementação

Ao iniciar o aplicativo é exibido o menu principal (Figura 13). Esta tela tem as seguintes opções:

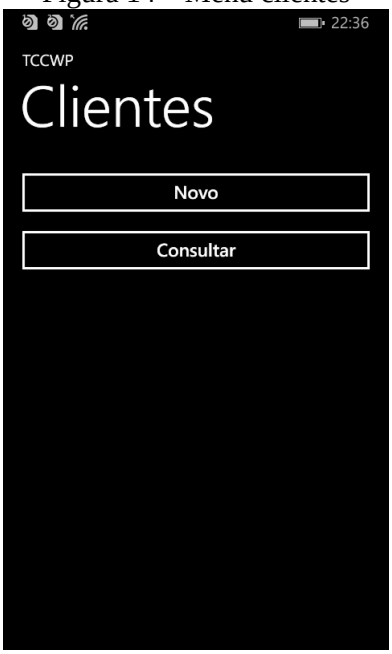
a) `Clientes`: nesta opção o usuário poderá cadastrar um novo cliente, visualizar cadastros já existentes e altera-los;
b) `Pedidos`: nesta opção o usuário poderá cadastrar um novo pedido, visualizar pedidos já existentes, e adicionar ou alterar anotações dos pedidos;
c) `Estoque`: nesta opção o usuário poderá consultar a quantidade em estoque e o preço dos produtos;
d) `Relatórios`: nesta opção o usuário poderá gerar relatórios de contas a receber, estoque, tabela de preços e situação de entrega;
e) `Sincronizar`: nesta opção o usuário poderá enviar as informações cadastradas ou alteradas por ele e receber as atualizações disponíveis no servidor;
f) `Configurar`: nesta opção o usuário poderá ativar a sincronia automática que ocorrerá a cada trinta minutos, desde que o aplicativo não esteja em uso.

Figura 13 – Menu principal

3.3.2.1 Cadastrar cliente

Para cadastrar um cliente o usuário deve selecionar o botão `Clientes` no menu principal (Figura 13), então será exibido o menu clientes (Figura 14).

Figura 14 – Menu clientes



Em seguida o usuário deve selecionar o botão `Novo` para exibir a tela de cadastro de cliente (Figura 15).

Figura 15 – Cadastro de cliente

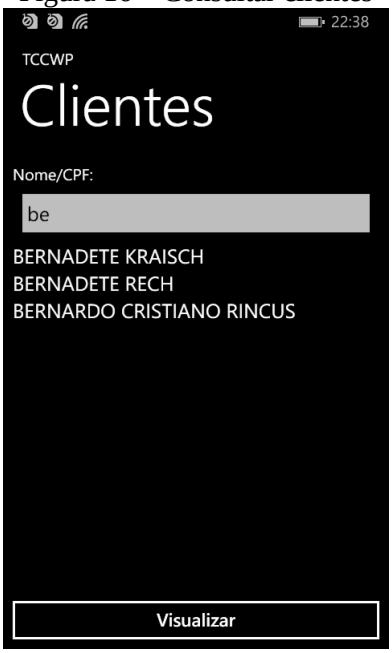
Dentro do cadastro de cliente o usuário pode deslizar o dedo para navegar entre as telas do cadastro de cliente representadas na Figura 15 e iniciar o cadastro na ordem que preferir.

Para fazer o cadastro o usuário deve preencher os campos, sendo que `Telefone`, `Email` e `Complemento` não são obrigatórios, e selecionar `Gravar`.

3.3.2.2 Visualizar e alterar cliente

Para visualizar os clientes cadastrados o usuário deve selecionar `Consultar` no menu clientes (Figura 14) para exibir a tela consultar clientes (Figura 16).

Figura 16 – Consultar clientes

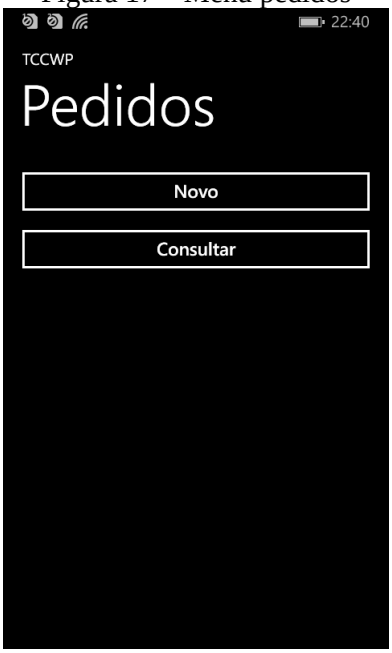


Para buscar um cliente, o usuário deve escrever o nome ou CPF do cliente desejado. A cada caractere digitado a lista é recarregada com os clientes com nome ou CPF correspondentes. Para visualizar o cadastro completo o usuário deve selecionar o nome e selecionar o botão `Visualizar`. Então será exibida a tela de cadastro de cliente (Figura 15), porém com os campos preenchidos com os dados do cliente. Caso desejar, o usuário poderá alterar as informações e selecionar o botão `Gravar` para alterar o cadastro.

3.3.2.3 Cadastrar pedido

Para cadastrar um pedido o usuário deve selecionar a opção `Pedidos` no menu principal (Figura 13), então será exibido o menu pedidos (Figura 17).

Figura 17 – Menu pedidos



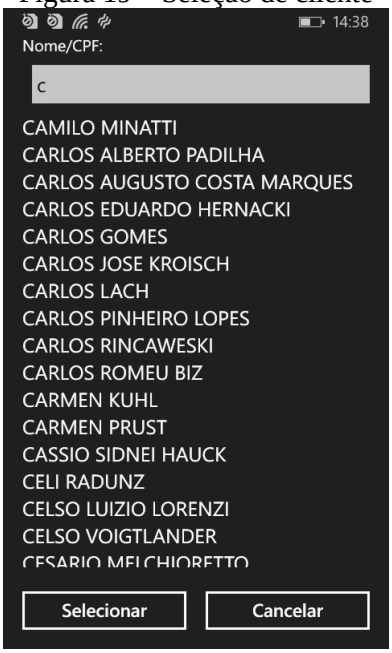
Em seguida o usuário deve selecionar o botão `Novo` para exibir a tela de cadastro de pedido (Figura 18).

Figura 18 – Cadastro de pedido

Dentro do cadastro de pedido o usuário pode deslizar o dedo para navegar entre as telas do cadastro de pedido representadas na Figura 18 e iniciar o cadastro na ordem que preferir. Para fazer o cadastro o usuário obrigatoriamente deve preencher os campos `Data de Emissão` e `Vendedor`, deve escolher um cliente, adicionar pelo menos um produto e um vencimento.

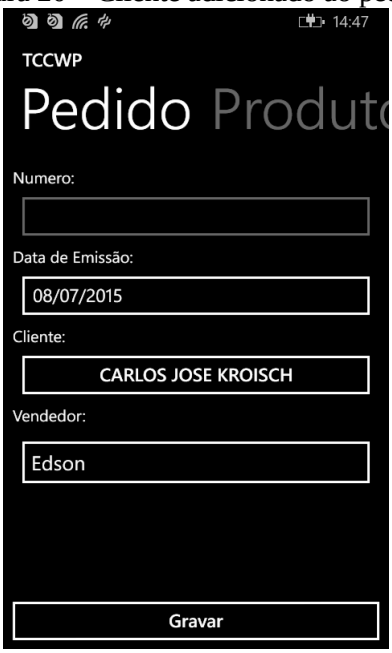
Para escolher o cliente o usuário deve selecionar o botão `Selecionar` na tela `Pedido` do cadastro de pedido, então será exibida a tela de seleção de cliente (Figura 19).

Figura 19 – Seleção de cliente



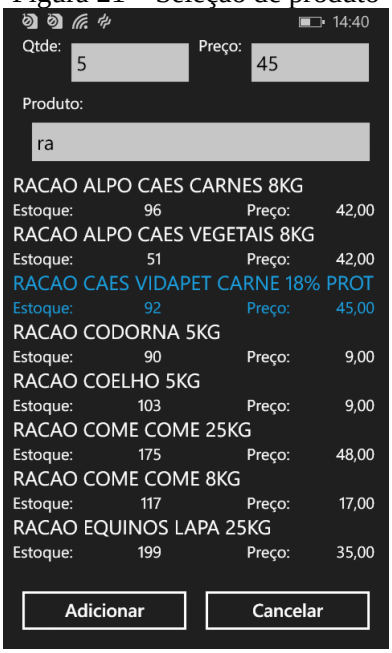
Para buscar um cliente, o usuário deve escrever o nome ou o CPF do cliente desejado. A cada caractere digitado a lista é recarregada com os clientes com nome ou CPF correspondentes. Em seguida o usuário deve selecionar o cliente desejado e selecionar o botão `Selecionar`. Então o cliente será adicionado ao pedido, como é possível ver na Figura 20.

Figura 20 – Cliente adicionado ao pedido



Para adicionar um produto ao pedido o usuário deve selecionar o botão `Adicionar` na tela de `Produtos` do cadastro de pedido, então será exibida a tela de seleção de produto (Figura 21).

Figura 21 – Seleção de produto



Para buscar um produto, o usuário deve escrever o nome do produto desejado. A cada caractere digitado a lista é recarregada com os produtos com nome correspondentes. Em seguida o usuário deve selecionar o produto desejado, informar a quantidade e o preço, e então selecionar o botão `Adicionar`. Então o produto será adicionado ao pedido, como é possível ver na Figura 22.

Figura 22 – Produto adicionado ao pedido

Para adicionar um vencimento ao pedido o usuário deve selecionar o botão `Adicionar` na tela de `Vencimentos` do cadastro de pedido, então será exibida a tela de seleção de data (Figura 23).

Figura 23 – Seleção de data

Após escolher a data o usuário deve selecionar o botão de confirmação, então será exibida a tela para informar o valor (Figura 24).

Figura 24 – Informando valor do vencimento

O usuário deve informar o valor do vencimento e selecionar o botão `Adicionar`. Então o vencimento será adicionado ao pedido, como é possível ver na Figura 25.

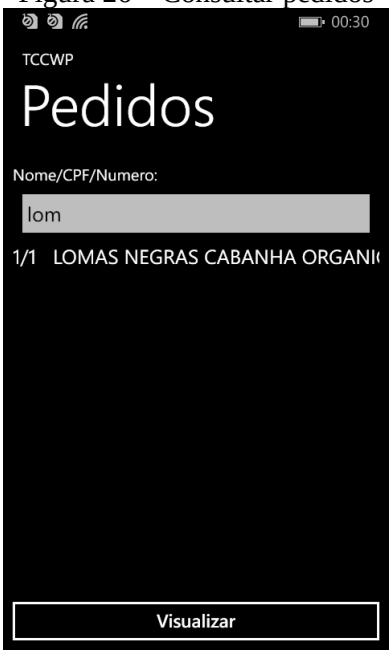
Figura 25 – Vencimento adicionado ao pedido

Após preencher todos os campos obrigatórios, o usuário deve selecionar o botão `Gravar` para gravar o pedido.

3.3.2.4 Visualizar pedido

Para visualizar os pedidos cadastrados o usuário deve selecionar `Consultar` no menu pedidos (Figura 17) para exibir a tela consultar pedidos (Figura 26).

Figura 26 – Consultar pedidos



Para buscar um pedido, o usuário deve escrever o nome ou CPF do cliente ou o número do pedido desejado. A cada caractere digitado a lista é recarregada com os pedidos com número correspondente ou clientes com nome ou CPF correspondentes. Para visualizar o

cadastro completo o usuário deve selecionar o pedido e selecionar o botão `Visualizar`. Então será exibida a tela de cadastro de pedido (Figura 18), porém com os campos sem permissão para alteração e preenchidos com os dados do pedido.

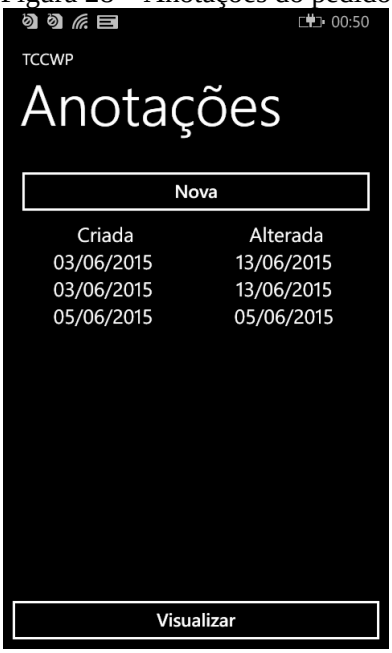
3.3.2.5 Adicionar, visualizar e alterar anotação

Quando o usuário estiver visualizando um pedido é exibido o botão `Anotações`, como é possível ver na Figura 27.

Figura 27 – Visualizar pedido

Para visualizar as anotações o usuário deve selecionar a opção `Anotações`, então será exibida a tela com as anotações do respectivo pedido (Figura 28).

Figura 28 – Anotações do pedido



Para criar uma nova anotação o usuário deve selecionar a opção `Nova`, então será exibida a tela de cadastro de anotação (Figura 29). Dentro da tela de anotações do pedido também é possível visualizar selecionando a anotação e selecionando `Visualizar`, então será exibida a tela de cadastro de anotação preenchida com o texto. Para alterar a anotação o usuário deve visualizar uma anotação, alterar o texto e selecionar o botão `Gravar`.

Figura 29 – Cadastro de anotação

3.3.2.6 Consultar estoque e preço

Para consultar estoque e preço de produtos o usuário deve selecionar a opção `Estoque` no menu principal (Figura 13), então será exibida a tela de consulta de estoque (Figura 30).

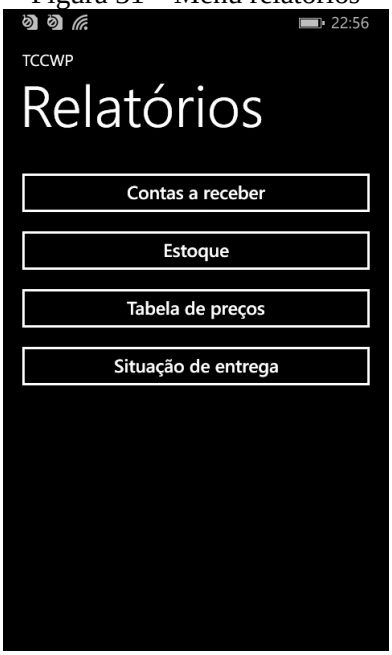
Nesta tela o usuário deve escrever o nome do produto desejado. A cada caractere digitado a lista é recarregada com os produtos com nome correspondente.

Figura 30 – Consulta de estoque e preço

3.3.2.7 Visualizar relatórios

Para visualizar os relatórios o usuário deve selecionar a opção `Relatórios` no menu principal (Figura 13), então será exibido o menu relatórios (Figura 31).

Figura 31 – Menu relatórios



Em seguida o usuário deve selecionar a opção desejada para gerar o relatório. Por exemplo, selecionando `Tabela de preços` será gerado o relatório em PDF, como é possível ver na Figura 32.

Figura 32 – Relatório tabela de preços

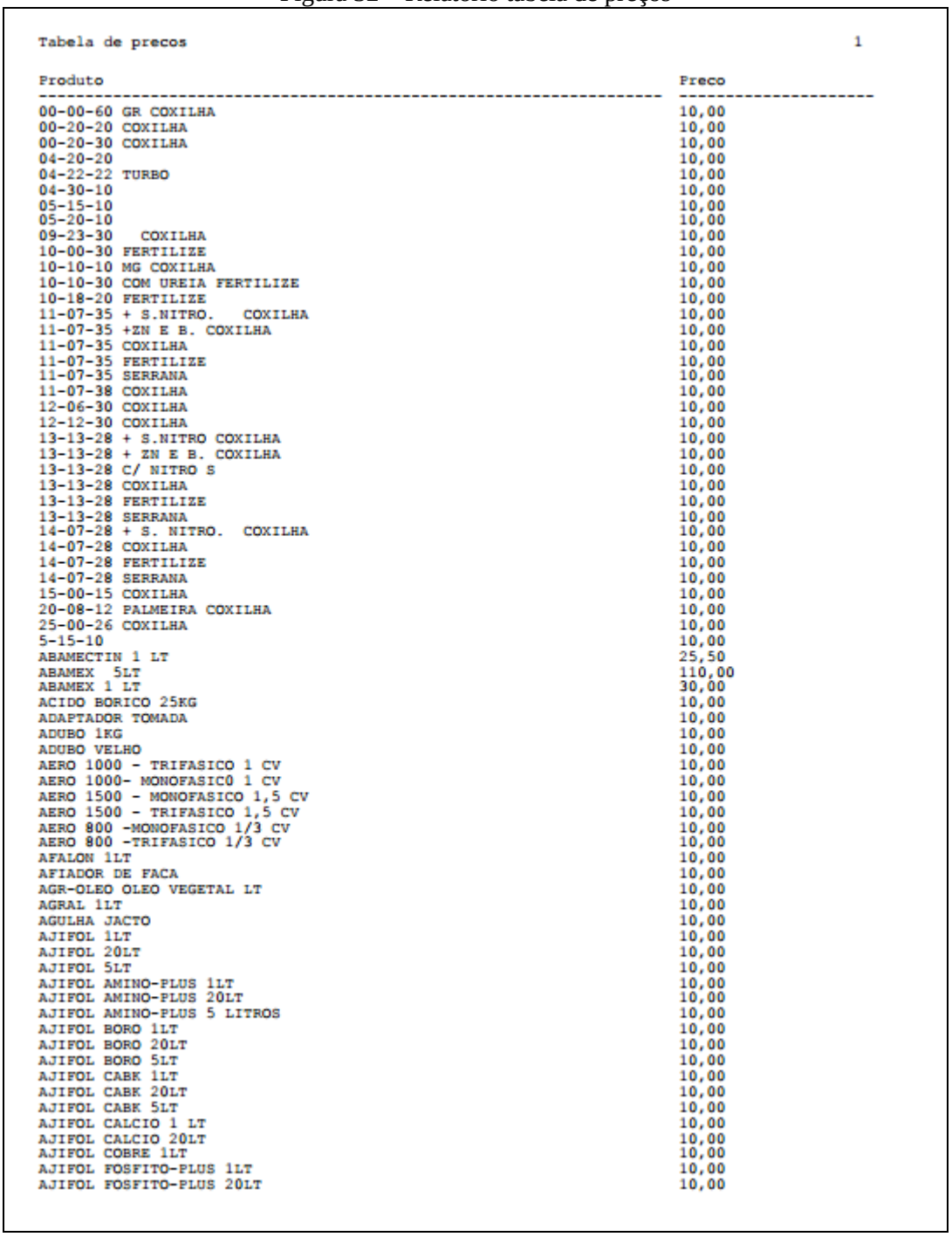


Tabela de precos 1

Produto	Preco
00-00-60 GR COXILHA	10,00
00-20-20 COXILHA	10,00
00-20-30 COXILHA	10,00
04-20-20	10,00
04-22-22 TURBO	10,00
04-30-10	10,00
05-15-10	10,00
05-20-10	10,00
09-23-30 COXILHA	10,00
10-00-30 FERTILIZE	10,00
10-10-10 MG COXILHA	10,00
10-10-30 COM UREIA FERTILIZE	10,00
10-18-20 FERTILIZE	10,00
11-07-35 + S.NITRO. COXILHA	10,00
11-07-35 +ZN E B. COXILHA	10,00
11-07-35 COXILHA	10,00
11-07-35 FERTILIZE	10,00
11-07-35 SERRANA	10,00
11-07-38 COXILHA	10,00
12-06-30 COXILHA	10,00
12-12-30 COXILHA	10,00
13-13-28 + S.NITRO COXILHA	10,00
13-13-28 + ZN E B. COXILHA	10,00
13-13-28 C/ NITRO S	10,00
13-13-28 COXILHA	10,00
13-13-28 FERTILIZE	10,00
13-13-28 SERRANA	10,00
14-07-28 + S. NITRO. COXILHA	10,00
14-07-28 COXILHA	10,00
14-07-28 FERTILIZE	10,00
14-07-28 SERRANA	10,00
15-00-15 COXILHA	10,00
20-08-12 PALMEIRA COXILHA	10,00
25-00-26 COXILHA	10,00
5-15-10	10,00
ABAMECTIN 1 LT	25,50
ABAMEX 5LT	110,00
ABAMEX 1 LT	30,00
ACIDO BORICO 25KG	10,00
ADAPTADOR TOMADA	10,00
ADUBO 1KG	10,00
ADUBO VELHO	10,00
AERO 1000 - TRIFASICO 1 CV	10,00
AERO 1000- MONOFASIC0 1 CV	10,00
AERO 1500 - MONOFASICO 1,5 CV	10,00
AERO 1500 - TRIFASICO 1,5 CV	10,00
AERO 800 -MONOFASICO 1/3 CV	10,00
AERO 800 -TRIFASICO 1/3 CV	10,00
AFALON 1LT	10,00
AFIADOR DE FACA	10,00
AGR-OLEO OLEO VEGETAL LT	10,00
AGRAL 1LT	10,00
AGULHA JACTO	10,00
AJIFOL 1LT	10,00
AJIFOL 20LT	10,00
AJIFOL 5LT	10,00
AJIFOL AMINO-PLUS 1LT	10,00
AJIFOL AMINO-PLUS 20LT	10,00
AJIFOL AMINO-PLUS 5 LITROS	10,00
AJIFOL BORO 1LT	10,00
AJIFOL BORO 20LT	10,00
AJIFOL BORO 5LT	10,00
AJIFOL CABK 1LT	10,00
AJIFOL CABK 20LT	10,00
AJIFOL CABK 5LT	10,00
AJIFOL CALCIO 1 LT	10,00
AJIFOL CALCIO 20LT	10,00
AJIFOL COBRE 1LT	10,00
AJIFOL FOSFITO-PLUS 1LT	10,00
AJIFOL FOSFITO-PLUS 20LT	10,00

3.3.2.8 Sincronizar

Para executar a sincronização, o usuário deve selecionar a opção `Sincronizar` no menu principal (Figura 13), então os botões do menu principal ficarão inativos até o término

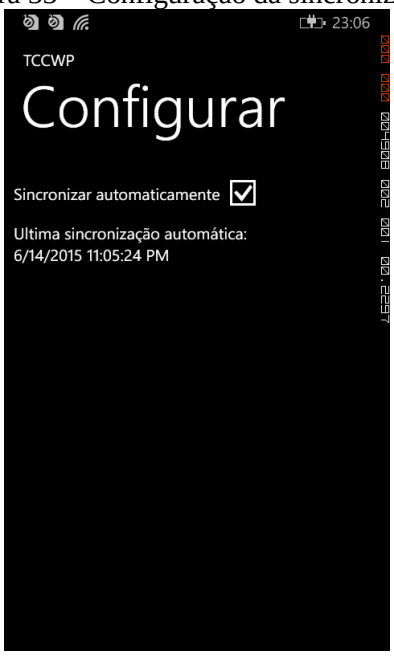
da sincronização. Ao terminar será exibida uma mensagem indicando sucesso ou erro na sincronização.

Mesmo com a sincronização automática que é executada a cada trinta minutos, há situações que o usuário precisa enviar ou receber informações imediatamente. Por exemplo, a situação em que o usuário acabou de cadastrar um pedido importante e sabe que em poucos minutos ficará sem conexão à internet, neste momento ele poderá executar a sincronização e adiantar o envio do pedido.

3.3.2.9 Ativar sincronização automática

O usuário deve selecionar a opção `Configurar` no menu principal (Figura 13). Então será exibida a tela de configuração (Figura 33). Para ativar o usuário deve marcar a opção `Sincronizar automaticamente`.

Figura 33 – Configuração da sincronização



3.4 RESULTADOS E DISCUSSÕES

O presente trabalho atingiu o objetivo proposto que era desenvolver um aplicativo voltado a força de vendas que permitisse o usuário visualizar informações, cadastrar as informações, visualizar relatórios de forma *offline* e sincronizar os dados com servidor através de um *web service*.

Comparando com o aplicativo desenvolvido por Oliveira (2014), pode-se notar que há algumas diferenças. Como é possível ver na Quadro 6, o sistema operacional utilizado pelo aplicativo de Oliveira (2014) é o Android, a linguagem utilizada foi Java, foi desenvolvido no ambiente Eclipse, o *web service* utiliza REST e deve estar sempre *online*. Já o aplicativo deste

trabalho foi feito para o sistema operacional Windows Phone, com a linguagem C# no ambiente Visual Studio, o *web service* utiliza SOAP e pode ser utilizado *offline*.

Quadro 6 – Comparativo com o aplicativo de Oliveira (2014) e Meus Pedidos

Características	**Aplicativo desenvolvido**	**Oliveira (2014)**	**Meus Pedidos**
Sistema operacional	Windows Phone	Android	Android
Linguagem	C#	Java	Desconhecida
Ambiente	Visual Studio	Eclipse	Desconhecido
Web service	SOAP	REST	Desconhecido
Uso de internet	Somente na sincronização	Obrigatório	Somente na sincronização

Em relação ao aplicativo Meus Pedidos (GOOGLE PLAY, 2014) a principal diferença é o sistema operacional Android. Além disso, tem funções para empresas de outros seguimentos, por exemplo a indústria têxtil.

4 CONCLUSÕES

Os objetivos propostos por este trabalho foram alcançados. Foi desenvolvido um aplicativo que sincroniza suas informações sob demanda e que permite à equipe de força de vendas realizar cadastros de clientes e pedidos, além de permitir visualizar informações importantes para o trabalho de vendedor externo.

As ferramentas utilizadas para o desenvolvimento se mostraram adequadas. O Visual Studio juntamente com a linguagem C# mostraram-se como um bom ambiente para o desenvolvimento de aplicativos para Windows Phone. Porém houve uma dificuldade em relação aos relatórios, porque não é possível utilizar a ferramenta de criação de relatórios no Windows Phone. Assim, foi necessário fazer toda a geração dos arquivos PDF. Em relação ao *framework* WCF, nele o desenvolvimento de *web services* é muito fácil e rápido. Em questão ao SQLite, a ferramenta Sqlite-net, que foi escolhida para fazer conexão com o banco de dados, apresentou alguns erros. Assim ela dificultou o desenvolvimento, porém os problemas puderam ser contornados.

Por fim, após uma demonstração realizada à equipe de vendas da empresa Proagro Agropecuária, foi sugerido que seria interessante desenvolver também um sistema de gerenciamento de entregas, para que pudesse ser possível visualizar no aplicativo as entregas agendadas e a previsão de data de entrega. Também sugeriram o desenvolvimento de mais relatórios, como relatórios de metas e vendas de um produto em um período selecionado pelo usuário.

4.1 EXTENSÕES

Sugerem-se as seguintes extensões para trabalhos futuros:

a) desenvolver interface para *tablets*;
b) desenvolver a parte de segurança do *web service*;
c) desenvolver integração com o aplicativo Here Drive.

REFERÊNCIAS

ATTORRE, Brunno Fidel Maciel. **Web Services REST versus SOAP**. [S.l.], 2015. Disponível em: <http://www.devmedia.com.br/web-services-rest-versus-soap/32451>. Acesso em: 07 jul. 2015.

BERNARDES, Wagner. **Mobilidade**: Maior eficiência à cadeia de suprimentos. [S.l.], 2009. Disponível em: <http://www.faithsystem.com.br/artigos_sobre_forca_de_vendas_e_mobilidade_mobilidade_maior_eficiencia_a_cadeia_de_suprimentos.asp>. Acesso em: 26 ago. 2014.

GOOGLE PLAY. **Meus Pedidos - controle vendas**. [S.l.], 2014. Disponível em: <https://play.google.com/store/apps/details?id=br.com.meuspedidos&hl=pt_BR>. Acesso em: 11 set. 2014.

GRALLA, Preston. **Demystifying Web services:** How they really work. [S.l.], 2002. Disponível em: <http://searchsoa.techtarget.com/news/797955/Demystifying-Web-services-How-they-really-work>. Acesso em: 17 abr. 2015.

LECHETA, Ricardo R. **Desenvolvendo para Windows 8**: aprenda a desenvolver aplicativos para Windows Phone 8 e Windows 8. São Paulo: Novatec, 2013.

LEE, Henry; CHUVYROV, Eugene. **Beginning Windows Phone App Development**. New York: Apress, 2012.

LEOPOLDO, Marcus Rommel Barbosa. **Simple Object Access Protocol:** Entendendo o Simple Object Access Protocol (SOAP). [S.l.], 2003. Disponível em: <http://wiki.pge.ce.gov.br/images/0/0b/SOAP.pdf>. Acesso em: 16 abr. 2015.

OLIVEIRA, Fabio. **Protótipo de aplicativo de força de vendas para dispositivos móveis baseados na plataforma Android**. 2014. 78 f. Trabalho de conclusão de curso (Bacharelado Sistemas de Informação) – Universidade Regional de Blumenau, Blumenau. Disponível em: <http://www.inf.furb.br/~pericas/orientacoes/ForcaVenda2014.pdf>. Acesso em: 12 set. 2014.

ORACLE. **The Java EE 6 Tutorial**. Redwood City, 2013. Disponível em: <http://docs.oracle.com/javaee/6/tutorial/doc/javaeetutorial6.pdf>. Acesso em: 6 set. 2014.

PAMPLONA, Vitor F. **Web Services.** Construindo, disponibilizando e acessando Web Services via J2SE e J2ME. [S.l.], 2010. Disponível em: <http://javafree.uol.com.br/artigo/871485/Web-Services-Construindo-disponibilizando-e-acessando-Web-Services-via-J2SE-e-J2ME.html>. Acesso em: 6 set. 2014.

PETZOLD, Charles. **Programming Windows Phone 7**. Redmond: Microsoft Press, 2010.

THURROT, Paul. **Windows Phone 8.1 Field Guide.** [S. l.: s.n.], 2014. Disponível em: <http://static.windows81book.com/Windows Phone 8.1 Field Guide 0.4.pdf>. Acesso em: 16 mar. 2015.

VISUAL STUDIO, **Microsoft Visual Studio Express 2013 com Update 3 para Windows**. [S.l.], [2013?]. Disponível em: <http://www.visualstudio.com/pt-br/downloads#d-express-windows-8>. Acesso em: 6 set. 2014.

WHITECHAPEL, Andrew; MCKENNA, Sean. **Windows Phone 8 Development Internals:** Preview 1. [S. l.]: Microsoft Press, 2012.

ZIEGLER, Chris. **Windows Phone 7**: the complete guide. [S.l.], 2010. Disponível em: <http://www.engadget.com/2010/03/18/windows-phone-7-series-the-complete-guide>. Acesso em: 12 set. 2014.