UNIVERSIDADE SÃO FRANCISCO
Engenharia de Computação

IVES SADAGI MURAKAMI
JOÃO JUNIOR MARQUES DE LIMA
WILIAN CARLOS DE LIMA

ANALISE DE BIBLIOTECAS PARA WEBSERVICES NO DESENVOLVIMENTO EM SMARTPHONES BASEADO NO SISTEMA OPERACIONA MICROSOFT® WINDOWS PHONE™.

Itatiba
2013

IVES SADAGI MURAKAMI - R.A. 002200800120
JOÃO JUNIOR MARQUES DE LIMA - R.A. 002200700535
WILIAN CARLOS DE LIMA - R.A. 002200800435

ANALISE DE BIBLIOTECAS PARA WEBSERVICES NO DESENVOLVIMENTO EM SMARTPHONES BASEADO NO SISTEMA OPERACIONA MICROSOFT® WINDOWS PHONE™.

Monografia apresentada ao Curso de Engenharia de Computação da Universidade São Francisco, como requisito parcial para a obtenção do título de Engenheiro de Computação.

Orientador: Prof. Ms. Fábio Andrijauskas

Itatiba
2013

Eu, Ives, dedico este trabalho aos meus pais, que me apoiaram durante todo o tempo da graduação, a minha namorada Nadia Fornaro que ficou ao meu lado dando o suporte necessário. Agradeço também aos meus companheiros de grupo, pois sem eles este trabalho jamais teria sido concluído.

Eu, João Junior, dedico este trabalho à minha mãe Rosa, ao meu pai João, aos meus irmãos Evandro (*in memoriam*), que nos deixou em 2008 e Evanilton, e à minha namorada Sayuri, que sempre foram e serão meus maiores incentivadores e sem eles eu não teria conseguido chegar a essa etapa.

Eu, Wilian, dedico este trabalho à minha família, e em especial ao meu maior orgulho, meu filho Henrique.

AGRADECIMENTOS

Primeiramente a Deus, por ter nos dado a oportunidade de ingressar nessa instituição de ensino e nos concedido saúde, paciência, sabedoria e serenidade para frequentar as aulas, concluir as disciplinas e finalizar com esse trabalho de conclusão de curso.

Ao PROUNI – Programa do Governo Federal que possibilitou a muitos jovens que não teriam condições de frequentar o curso de ensino superior de sua preferência, incluindo dois integrantes desse grupo.

À Universidade São Francisco, por nos ter disponibilizado ótimos professores e a estrutura necessária para que pudéssemos nos empenhar durante todo o curso.

Aos professores da Universidade São Francisco, que contribuíram e muito para nossa formação acadêmica e humana, e principalmente ao nosso professor orientador Fábio Andrijauskas, que aceitou nos orientar quando já estávamos no meio do ano, sabendo que havia pouco tempo para dar andamento ao trabalho, sendo fundamental ao nos nortear no desenvolvimento e finalização do projeto.

Aos inúmeros amigos que fizemos durante o curso, nas aulas ou no transporte de nossas cidades até o campus, e também os amigos de fora do ambiente acadêmico, os quais, cada um do seu jeito, uns menos, outros mais, contribuíram para essa formação que iremos carregar na lembrança pelo resto de nossas vidas.

Às nossas companheiras, que sempre dedicaram seu tempo a nos ajudar, dar forças e entenderam quando nossa atenção teria que ser voltada mais aos estudos.

E finalmente aos nossos familiares, que sem sombra de dúvidas foram nossos maiores incentivadores e sem o imenso esforço deles nenhum de nós conseguiria concluir esse sonho.

“A persistência é o menor caminho do êxito”
Charles Chaplin

RESUMO

Nos últimos anos, grandes empresas de tecnologia investiram fortemente no mercado de dispositivos móveis. Desde 2010, a *Microsoft* atua neste setor com o sistema operacional móvel *Windows Phone* disponibilizando, para a comunidade de desenvolvedores, ferramentas e ambientes de programação para a criação de aplicativos para esta plataforma. O objetivo deste trabalho é a implementação, execução e análise de resultados de um aplicativo capaz de se comunicar assincronamente com o serviço *REST* (*Representational State Transfer*) através de duas formas distintas de conexão, a biblioteca nativa *System.NET.WebClient* e o *RestSharp,* de código aberto. A codificação dos métodos de consulta demonstra a facilidade de uso deste tipo de serviço. A média dos resultados dos tempos de execução mostra que além da *framework* da *Microsoft*, existem outras opções viáveis para o consumo de *web services* no *Windows Phone*.

Palavras-chave: *Smartphone*, *Windows Phone*, Aplicativos, Requisições *Web*, *Web Services*, *WebClient, RestSharp*.

ABSTRACT

In recent years, large technology companies have invested heavily in the mobile device market. Since 2010, Microsoft operates in this sector with the Windows Phone mobile operating system available to the developer community, tools and programming environments to create applications for the platform. The objective of this work is the implementation, execution and analysis of an application able to communicate asynchronously with a REST service (Representational State Transfer) through two distinct forms of connection, the native library System.Net.WebClient and open source RestSharp. The methods demonstrates the ease of use of this service. The average of the results of the execution times show that besides the Microsoft framework, there are other viable options for consuming web services on Windows Phone.

Keywords: *Smartphone, Windows Phone, Apps, HTTP Requests, Web Services, WebClient, RestSharp.*

LISTA DE FIGURAS E ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*	Interface de Programação de Aplicação
APP	*Application*	Aplicação
CEO	*Chief Executive Officer*	Executivo Principal da Empresa
GB	*Gigabyte*	(Unidade de medida de tamanho de dados)
GPU	*Graphics Processing Unit*	Unidade Gráfica de Processamento
IDE	*Integrated Development Environment*	Ambiente de Desenvolvimento Integrado
iOS	*iPhone Operating System*	Sistema Operacional iPhone
LED	*light-emitting diode*	Diodo Emissor de Luz
MB	*Megabyte*	(Unidade de medida de tamanho de dados)
NFC	*Near Field Communication*	Comunicação em Espaço Próximo
PC	*Personal Computer*	Computador Pessoal
PDA	*Personal Digital Assistant*	Assistente Digital Pessoal
RAM	*Random Access Memory*	Memória de Acesso Aleatório
SDK	*Software Development Kit*	Conjunto de Desenvolvimento de Software
SO	*System Operation*	Sistema Operacional
UI	*User Interface*	Interface do Usuário
USB	*Universal Serial Bus*	Barramento Serial Universal
VGA	*Video Graphics Array*	Padrão de Disposição Gráfica
VS	*Visual Studio*	Visual Studio
WP	*Windows Phone*	Windows Phone
WP8	*Windows Phone 8*	Windows Phone 8
WVGA	*Wide Video Graphic Array*	Padrão de Disposição Gráfica Amplo
XVGA	*Extended Video Graphic Array*	Padrao de Disposição Grafica Extendido
DLL	*Dynamic-Link Library*	Biblioteca de Vínculo Dinâmico
WCF	*Windows Communication Foundation*	Fundação para Cumunicação *Windows*

SUMÁRIO

1 INTRODUCÃO

Nos últimos anos, grandes empresas de tecnologia investiram fortemente no mercado de dispositivos móveis. Segundo relatório da *Pew Internet & American Life Project* (2013), 53% dos americanos possuem um *smartphone*. A agência de Telecomunicações da ONU afirma que em 2014 o número de celulares superará a quantidade de habitantes do planeta. A partir dai conclui-se que estes dispositivos se tornarão tão comuns quanto os computadores pessoais foram na ultima década.

Uma das grandes responsáveis pelo crescimento do uso e a evolução dos *smartphones* foi a *Apple* e seu CEO, Steve Jobs, conforme Olivério (2011), no artigo a importância de Steve Jobs para o mercado e o mundo tecnológico. Em 2007, sob o comando do seu fundador a empresa lançou no mercado o iPhone. Antes do seu lançamento, já havia vários modelos de smartphones, porém a maioria era voltada para o publico corporativo, destacando-se as fabricantes BlackBerry, Palm, Motorola, Dell entre outras.

A *Apple* seguiu na contramão e lançou um aparelho voltado ao uso pessoal, incorporando recursos multimídias do *iPod*, (tocador de arquivos de áudio) navegação via rede celular 3G, câmera fotográfica, sensor de movimento conhecido como acelerômetro entre outros recursos. Para controlar este *hardware* foi usado o *iPhone OS*, mais tarde renomeado para *iOS*, um sistema operacional baseado no *MacOS X*, desenvolvido pela própria *Apple* para uso em seus desktops. O *software* se destacava pelo visual e *design* inovadores, tempos de resposta e interfaces superiores aos concorrentes além da possibilidade da instalação de novos aplicativos, disponibilizados através de um portal web dedicado, a Apple Store (PEREIRA, 2011).

No ano seguinte, em 2008, a *Google Inc*. lançou seu Sistema Operacional (SO) para *smartphones*, o *Android*. Ao contrário do *iOS*, que foi concebido para uso exclusivo do *iPhone*, a ideia da Google foi criar um SO capaz de funcionar em qualquer dispositivo, independente das configurações de *hardware*. Com o passar dos anos, esta interoperabilidade tornou o Android líder de mercado dos smartphones de acordo com a Association (2012).

A *Microsoft*, empresa mundialmente conhecida pelo domínio na área de *softwares* para computadores pessoais e corporativos, já investia desde o ano 2000 no desenvolvimento de sistemas operacionais para dispositivos móveis como *smartphones* e *PDAs*, através do *Windows Mobile*. Contudo, o foco da empresa estava voltado à criação de

um sistema que trouxesse os recursos e a experiência de usuário do SO Windows usado nos Computadores Pessoais – *Personal Computers* (PCs) para estes outros dispositivos, além de atuar com ênfase maior no mercado corporativo. Esta estratégia foi ficando cada vez mais obsoleta e não rendeu a *Microsoft* os mesmos resultados obtidos na área de computadores pessoais (GRALLA, 2012).

A partir de 2010 com a consolidação do *iOS* e do *Android* no mercado, a *Microsoft* decidiu apostar em um novo sistema, sucessor ao *Windows Mobile* para voltar a concorrer em igualdade com as companhias rivais. Em outubro daquele ano foi lançado o *Windows Phone 7* Revista Galileu (2013).

A *Microsoft* desde então, tem estimulado a criação de aplicativos para a plataforma *Windows Phone*, disponibilizando grande quantidade de materiais, treinamentos e tutoriais voltados aos programadores, além de uma SDK (*Software Development Kit*) de desenvolvimento.

Segundo dados da *Microsoft* divulgados numa convenção realizada em junho de 2013, a *Windows Phone Store*, ou Loja Virtual do *Windows Phone*, que é o portal para disponibilização de aplicativos para o sistema, ultrapassou a marca de 160 mil *softwares* publicados. O *Windows Phone* já corresponde ao 2º sistema operacional móvel com maior número de usuários na América Latina. Diversos fabricantes estão apostando no sistema, como *Samsung, Huawei, HTC* e de acordo com Dela Valle (2013), principalmente a *Nokia*, companhia finlandesa que há tempos já era parceira de negócios da *Microsoft* e em setembro de 2013 anunciou a venda de sua unidade de celulares e *smartphones* para a gigante americana.

1.1 Objetivos

Este trabalho demonstra a arquitetura do sistema operacional e alguns recursos básicos oferecidos pela plataforma *Microsoft Windows Phone* para o desenvolvimento de aplicativos simples. O objetivo é a implementação, execução e análise de resultados de um aplicativo capaz de se comunicar assincronamente com um serviço REST através de duas formas distintas de conexão, a biblioteca nativa *System.NET.WebClient* e o *RestSharp,* de código aberto.

1.2 Metodologia

Inicialmente para o desenvolvimento deste trabalho foram realizadas e expostas informações de diversas pesquisas, sobre temas como surgimento e popularização dos smartphones. Partindo destas pesquisas, foram demonstradas informações básicas a respeito da arquitetura, recursos e aplicativos do sistema operacional móvel da Microsoft, o *Windows Phone.* O conteúdo seguinte demonstra a criação de aplicativos simples, com o exemplo clássico "*Hello World".* O trabalho se encerra com o desenvolvimento de um *web service*, um aplicativo cliente Windows Phone, medição e exposição dos resultados de consumo deste.

Os aplicativos mostrados neste trabalho foram desenvolvidos no Microsoft Visual Studio versões Professional 2010 e 2012, instalados em um computador comum com sistema operacional Windows 8. Também foram instaladas as versões 7.1 e 8.0 do *Software Development Kit* (SDK) para Windows Phone. O *web service* foi implementado na linguagem de programação C# e ASP.NET. Os aplicativos foram desenvolvidos em C# e a interface de usuário recebeu códigos no padrão XAML.

Os testes dos aplicativos foram realizados no aparelho Nokia Lumia 510, com a versão 8 do Windows Phone e desbloqueado através da ferramenta *Windows Phone Developer Registration.* Já o *web service* foi publicado no *Internet Information Services versão Express.*

Para repositório e controle de versão dos projetos foi utilizada a ferramenta online Microsoft Team Foundation Server (TFS). Já os arquivos de documentos foram armazenados através da ferramenta online Google Drive. Mediante ao exposto acima, o próximo capitulo apresentará algumas características do *Windows Phone*, o desenvolvimento e testes com a utilização dos aplicativos propostos.

2 FUNDAMENTAÇÃO TEÓRICA

São apresentados neste capítulo os conceitos da Programação Orientada a Objetos, das Linguagens de programação C e C#, da plataforma de desenvolvimento .NET, das linguagens de marcação XML e XAML, do modelo de desenvolvimento web ASP.NET, da solução *Web Services,* chamada assíncrona ao servidor, s*martphones* e pacote de desenvolvimento *Microsoft Visual Studio*.

2.1 Linguagens de Programação

De acordo com Fischer e Grodzinsky (1993), linguagem de programação é um conjunto de regras sintáticas e semânticas usadas para definir um programa de computador que permitem ao programador especificar precisamente sobre quais dados um computador vai atuar, como estes dados serão armazenados ou transmitidos e quais ações devem ser tomadas sobre esses dados, conforme varias circunstancias distintas.

Em outras palavras, linguagem de programação são linhas de código que definem uma rotina lógica para que o computador atue, de forma que o objetivo seja alcançado. Possuem vários paradigmas de estruturação e neste trabalho serão abordadas duas linguagens, as quais utilizam dois desses principais conceitos, que são a linguagem de programação estruturada e a linguagem de programação orientada a objetos.

Linguagens de Programação Estruturada, de acordo com Dahl, Dijkstra e Hoare (1972), “é uma forma de programação de computadores que preconiza que todos os programas possíveis podem ser reduzidos a apenas três estruturas: sequência, decisão e repetição”. Ainda, de acordo com Tucker e Noonan (2002), a programação estruturada “foi a forma dominante na criação de software entre a programação linear e a programação orientada por objetos”.

É uma forma de programação onde os códigos são passados em uma sequencia, com base em estruturas simples de blocos que são interligados pelas sub-rotinas de repetição e decisão e fazem as chamadas aos códigos das funções do programa.

Uma das principais linguagens que utilizam esse conceito de estruturação é a linguagem C, sendo ela de grande importância para a evolução das linguagens de programação de computadores.

A linguagem de programação C foi desenvolvida por Dennis Ritchie e implementada inicialmente em 1973. Trouxe algumas novidades para a época, como a introdução de tipos de dados, não existentes nas linguagens existentes anteriormente (Deitel, 2007, p.6).

Conforme explicitado por Deitel et al, (2007), C teve grande reconhecimento por ter sido a linguagem pela qual foi desenvolvido o Sistema Operacional UNIX, e hoje por ser uma linguagem de programação independente de *hardware*, é disponível para a maioria dos computadores e muitos dos Sistemas Operacionais existentes são desenvolvidos em C ou nas suas versões evoluídas, a linguagem C++ e C#.

Esta nova linguagem abriu caminho para diferentes tipos de paradigmas de programação, sendo a mais difundida, a programação orientada a objetos. Robinson et al, (2004), diz que uma linguagem de programação orientada a objetos deve atender 4 pilares básicos: Abstração, Encapsulamento, Herança e Polimorfismo e um dos seus principais objetivos é permitir a melhor organização e escrever trechos de código de fácil manutenção e, assim podendo realizar coletivamente tarefas bem complexas.

Uma característica importante das linguagens de programação orientada a objetos é o fato de trechos de código poderem ser reutilizados por outros desenvolvedores em outros programas. A *Microsoft* emprega este paradigma de orientação a objetos na concepção da linguagem de programação C#, conforme será explicado no tópico seguinte.

2.2 Linguagem de Programação C#

Conforme exposto por Deitel et al. (2007), o C# (lê-se *Ce-Sharp*) é uma linguagem de programação que foi projetada especificamente para ser utilizada na plataforma .NET, e como é baseada totalmente nas linguagens C, C++ e Java, torna a migração dos programadores mais acostumados com essas linguagens para a plataforma .NET muito mais fácil e rápida, pois foram desenvolvidos recursos adaptáveis a cada uma dessas linguagens e acrescentando características e capacidades próprias.

Pode-se explicar assim o sucesso de C#, por ser uma linguagem de programação visual dirigida por eventos e totalmente orientada a objetos, na qual os programas são criados utilizando uma IDE (*Integrated Development Enviroment* – ambiente de desenvolvimento integrado) e dessa forma, o tempo de criação, execução e testes dos

programas desenvolvidos é reduzido significativamente e de modo muito mais cômodo para o desenvolvedor.

Segundo Robinson et al. (2004) a *Microsoft*, desenvolvedora da linguagem, descreve C# como uma linguagem de programação “simples, moderna, orientada a objetos do tipo seguro e derivada de C e C++”. O autor ainda afirma que no momento da criação da linguagem, a *Microsoft* foi capaz de absorver as experiências de todas as outras linguagens semelhantes que estiveram disponíveis nos últimos vinte anos aproximadamente, quando os princípios da orientação a objetos começaram a se destacar.

Para Liberty e Xie (2009), o C# foi escolhido para ser a linguagem de programação a ser utilizada na plataforma .NET pois foi desenvolvida com as melhores características de algumas linguagens já existentes com o alto desempenho de C, a estruturação orientada a objetos de C++, a segurança e otimização na coleta de lixo do Java e o método de desenvolvimento rápido do *Visual Basic*, ficando assim preparada para desenvolvimento de aplicações web e *Windows Client* baseadas em componentes e varias camadas. Assim, a linguagem C# é componente principal de desenvolvimento na plataforma de aplicativos .NET, tanto em aplicativos *web* como para *desktop*, atendendo clientes de pequeno e grande porte, aplicativos ricos ou *Web Services*.

Outro autor, Sharp (2007), compartilha do mesmo ponto de vista, afirmando que C# é uma poderosa linguagem da *Microsoft* orientada para componentes que desempenha um papel importante na plataforma .NET, comparando com o mesmo papel que a linguagem C desempenhou no desenvolvimento do UNIX. A Linguagem C# é compatível com diversos outros formatos, tais como *eXtensible Makup Language*, o XML e *eXtensible Application Markup Language*, o XAML, que serão abordados na próxima seção.

2.3 XML e XAML

A XML*, eXtensible Makup Language*, ou Linguagem de Marcação Extensível, proporciona um método padrão para codificação de informação para que seja facilmente compreendido por diferentes aplicações de softwares. De acordo com Liberty e Xie (2009), XML contem dados e é a descrição desses dados que possibilitam às aplicações dos softwares interpretar e processar os dados.

XML é uma linguagem de marcação – tipo de linguagem utilizada para definir formatos, como um conteúdo será exibido na tela e padrões dentro de um documento

qualquer – assim como o *Hiper Text Markup Language* (HTML), com a diferença que o XML é extensível, ou seja, o usuário do XML tem a possibilidade de criar novos elementos e propriedades.

O formato XML serve como modelo para outros subtipos de linguagens de marcação, tais como XAML.

O XAML, ou *eXtensible Application Markup Language*, é uma linguagem de marcação baseada na linguagem XML, porém, consideravelmente mais ampla. XAML é uma linguagem declarativa, que pode iniciar objetos e definir propriedades utilizando uma estrutura de linguagem, o qual há relações hierárquicas entre diversos objetos e convenção de tipo de suporte compatível com a extensão de tipos (LABRIOLA; TAPPER; BOLES, 2012).

De acordo com a Microsoft (2013) através da linguagem XAML, é possível o intercâmbio de fontes entre ferramentas e diversas funções no processo de desenvolvimento sem perda de informações e utilizando XAML como formato de intercâmbio, as funções de designer e desenvolvedor podem ser separadas, e designers e desenvolvedores podem iterar na produção de um aplicativo.

Tendo em vista que o foco principal do presente trabalho está baseado no Sistema Operacional *Windows Phone*, convém citar Mônaco e Carmo (2012, p. 45), quando definem que “a facilidade de trabalho e clareza do XAML dá uma vantagem enorme no desenvolvimento de aplicativos para Windows Phone”. Além disso, a ferramenta de desenvolvimento aplicada *Microsoft Visual Studio,* da qual serão mostrados conceitos fundamentais na próxima seção, é compatível com todas as linguagens citadas.

Outra forma de fazer a representação de dados é a utilização do modelo JSON, que será demonstrado na próxima seção.

2.4 JSON

JSON – *JavaScript Object Notatione* – é um modelo que armazena e transmite informações em formato texto. Trata-se de um modelo simples, que é usado por aplicações *Web,* que comparado ao modelo XML, apresenta-se em uma estrutura mais compacta e rápida com relação ao “*parsing*” das informações (GONÇALVES, 2012). O autor ainda apresenta outras características que diferem o JSON de XML, como a de que JSON não é

linguagem de marcação, pois não possui *tags* e não permite executar instruções de processamento como no XML, entre outras.

Segundo a página *www.json.org*, JSON é completamente independente de linguagem, pois utiliza convenções familiares às linguagens C e derivados, como C++, C#, *Java*, *JavaScript*, *Perl*, *Python*, etc. Dessa forma, faz com que JSON seja um formato ideal de troca de dados.

2.5 A Plataforma de Desenvolvimento .NET e a ferramenta Microsoft Visual Studio

Conforme Robinson et al. (2004), a plataforma *.NET* é uma nova Interface de Programação de Aplicativos, API (*Aplication Programming Interface*) para programar em *Windows*. E a linguagem C# é uma linguagem que foi projetada desde o inicio especificamente para funcionar com *.NET*, tirar proveito de todo o progresso nos ambientes de desenvolvimento e da programação orientada a objetos desenvolvidos nas ultimas décadas.

Demonstra-se com essas afirmações que a arquitetura .NET pode se utilizar de varias plataformas para o desenvolvimento, ampliando ainda mais a portabilidade dos programas .NET. Além disso, a estratégia do ambiente .NET envolve uma nova maneira de desenvolvimento de programas e de como são escritos e executados, obtendo-se com isso uma maior produtividade.

Conforme citado pela Microsoft (2013), a criação de soluções em varias linguagens, como *Visual Basic, Visual C++ e Visual C#* é facilitada na plataforma de desenvolvimento .NET pelo compartilhamento de ferramentas no Microsoft Visual Studio, pois usam do mesmo Ambiente de Desenvolvimento Integrado (IDE) - *Integrated Development Environment*. O *Microsoft Visual Studio*, um dos principais pacote de programas de desenvolvimento da plataforma .NET, consiste em um conjunto de ferramentas utilizadas para desenvolver aplicações para *desktop*, aplicações móveis, serviços *Web* XML e aplicações *web* ASP.NET, cujas definições serão passadas na próxima seção.

2.6 ASP.NET

O modelo ASP.NET foi criado pela Microsoft logo após o desenvolvimento da plataforma .NET. Foi baseado no modelo ASP – *Active Server Pages* – também desenvolvido pela Microsoft, que veio para solucionar problemas de compilação e execução de aplicativos Web, dar suporte ao HTTP padrão, fornecer ferramentas que pudessem facilitar o desenvolvimento de aplicativos Web e gerar aplicativos que pudessem ser acessados e executados de qualquer navegador que de suporte ao HTML (SANTANA FILHO; ZARA, 2002).

Mas, mesmo o ASP trazendo essas soluções, alguns problemas ainda permaneciam no desenvolvimento *web*, como complexidade na codificação e a mistura do código HTML e do código do aplicativo. Sharp (2011) afirma que com a chegada da plataforma .NET esses obstáculos começaram a ser transpassados, visto que o modelo de desenvolvimento utilizado passou a separar o código escrito nas linguagens .NET do código HTML entre muitas outras novas facilidades agregadas com o ASP.NET .

Através do ASP.NET, é possível codificar aplicativos em qualquer linguagem compatível com o CLR – *Common Language Runtime*, onde estão inclusas: *Microsoft Visual Basic*, C#, JScript.NET e J#. O ASP.NET conta com uma estrutura de página e controles, compilador, infraestrutura de segurança, facilidades no gerenciamento de estado, arquivo de configuração da aplicação, recursos para monitoramento da integridade e desempenho, suporte a depuração, estrutura de serviços da Web XML, ambiente de hospedagem extensível, gerenciamento de ciclo de vida da aplicação e ambiente de designer extensível (ACKER; MCGOVERN; PATEL, 2004).

Além do desenvolvimento de aplicações *web*, o ASP.NET também permite a criação dos mais variados tipos de *Web Services* ou como são simplesmente chamados serviços *web*.

2.7 *Web Services*

Um componente importante da arquitetura .NET são os serviços da *web* (*Web services*), que são aplicativos que podem ser usados na internet. Os clientes e outros aplicativos podem usar esses serviços da *web* como blocos de construção reutilizáveis (DEITEL et al., 2007).

A visão do cliente de um *Web Service* é a de uma interface que expõe vários métodos bem definidos. Tudo o que o cliente precisa fazer é chamar esses métodos usando

os protocolos padrão da Internet, passando os parâmetros em um formato XML e recebendo respostas em um formato XML (SHARP, Jonh, p.466).

Web Service é definido como uma classe escrita em uma linguagem suportada pela plataforma .NET, o qual pode ser acessada via protocolo HTTP. Assim, é possível ter acesso a qualquer *Web Service* disponível e usar suas funcionalidades (MICROSOFT, 2013).

Como *Web Services* utilizam como protocolo padrão o HTTP para transmitir dados e o formato de dados baseado no XML, podem ser utilizado em conjunto com aplicativos clientes escritos em ambientes fora da plataforma .NET.

Conforme define BORGES JUNIOR (2005), o serviço é acessado sempre via HTTP, contando com uma string XML que é empacotada em um protocolo SOAP (*Simple Object Acess Protocol*) – padrão aberto criado pela *Microsoft, Ariba* e IBM para que a transferência de dados nas variadas aplicações fossem padronizadas, e por esse motivo é utilizado o XML.

Uma característica importante que se dá com a utilização de W*eb Services*, conforme Microsoft (2013) é que não há necessidade da empresa que desenvolve a DLL distribuir para todos os clientes, pois já estará armazenada e disponível em um HTTP.

O *Web Service* também apresenta outra vantagem que é a transparência para o Firewall da empresa, pois trata-se de uma string XML, um arquivo texto, não sendo necessário autorização do *Firewall*. Conclui-se, de acordo com Sheperd (2007) que W*eb Service* trata-se de uma conexão universal de computadores conectados por meio do protocolo HTTP e utilizando o formato de arquivos XML.

Existem diversos formatos de serviços *web*, sendo os mais comuns SOAP e RESTful, ou REST.

2.7.1 SOAP

SOAP - *Single Object Access Protocol* – é um protocolo padrão usado para enviar solicitações e receber respostas dos *Web Services*. Ele define como formatar as mensagens de chamada e respostas e como associa-las no HTTP (SHEPERD, George, 2007).

Conforme definido por Sharp (2007), SOAP é um protocolo leve, construído sobre o HTTP, que define uma gramática XML para especificar nomes de métodos que o cliente quer chamar em um *Web Service* e define parâmetros e valores de retorno.

Ele padroniza a codificação de dados para um formato portátil e bem leve, diminuindo o trabalho do emissor e do receptor para a comunicação e não faz distinção de como os dados serão transportados (BASIURA, Russ et al, 2003). Em outras palavras, SOAP define como o formato da mensagem e os dados devem ser codificados, como enviar mensagens e como processar as respostas.

É muito utilizado, e acabou se tornando o fundamento dos *Web Services* por ser esse protocolo leve, muito mais simples de implementar que outras tecnologias como CORBA e DCOM, e por utilizar tecnologias padrão como HTTP e XML para realizar a comunicação.

2.7.2 REST

Abreviação de *Representational State Transfer*, REST é um estilo de arquitetura, criado por Roy Thomas Fielding, um dos criadores do protocolo HTTP, tendo sido definido em sua tese de doutorado. Surgiu quando a internet apresentou uma nova necessidade alem das paginas *web*, que já era um sucesso, para uma demanda de serviços *web.* (WILDE; PAUTASSO, 2011).

A intenção da criação do REST é fazer com que os serviços possam ser simples e abertos e qualquer pessoa consiga utiliza-los, utilizando qualquer plataforma disponível, facilitando também o trabalho de desenvolvedores, por ser fácil de entender e desenvolver, com possibilidade de implantação em qualquer cliente ou servidor com suporte a HTTP ou HTTPS (ROZLOG, Mike, 2013).

O REST é uma forma de implementar um estilo de arquitetura de alto nível para compilar *software* em que os clientes podem fazer solicitações de serviço. Sua finalidade é que, ao invés de utilizar meios mais complexos, como por exemplo SOAP para comunicação entre cliente e servidor, seja utilizado o protocolo HTTP, que é muito mais simples para realizar essas chamadas. A requisição de um serviço REST pode ser realizada de forma síncrona ou assíncrona.

2.8 Chamada Síncrona e Chamada Assíncrona

Nos processos entre um aplicativo que faz a requisição e um *Web Service* que retorna um resultado para essa requisição, existem dois tipos de comunicação entre o requisitante e o servidor. São eles a chamada síncrona e a chama assíncrona.

De acordo com Basiura et al. (2003), o padrão de operação de uma chamada síncrona bloqueia um processo até que operação se complete, ou seja, fica aguardando até que se obtenha uma resposta da requisição enviada. Só depois da resposta, é chamada uma nova linha de execução.

Já a chamada com comunicação assíncrona, Basiura et al. (2003) afirmam ser bem mais complexa de ser implementada, pois o processo requisitante não espera pela resposta e não se sabe se a mensagem de requisição foi ou será entregue ao servidor. Também o processo que fez a chamada não sabe a resposta, se foi concluída perfeitamente ou não. Nesses casos tanto o SO tem de possuir meios para relatar esses erros quanto o programador terá de preparar o aplicativo para que o mesmo realize notificações sobre a conclusão da operação. O que compensa esses problemas e dificuldades na implementação de execução assíncrona, é que, se tomando todos os cuidados para se implementar corretamente, o uso da comunicação assíncrona melhora muito o desempenho do sistema e evita atrasos na espera pelo cliente dos resultados do *Web Service* .

Na plataforma .NET existem diversas ferramentas para executar chamadas assíncronas em *web services*, como por exemplo a Biblioteca *WebClient*, da *Microsoft* e a de código livre, *RestSharp.*

2.9 Bibliotecas *WebClient e RestSharp*

A biblioteca *WebClient* é uma classe pertencente a *.Net Framework*, desde a versão 2.0, cuja função é fornecer métodos comuns para enviar e receber dados de recursos web identificados por uma *URI (Uniform Resource Identifier).* Ela faz parte de um conjunto maior de classes, a System.NET, que é responsável pela disponibilização de interfaces para as mais diversas implementações de aplicativos com integração com dispositivos web (Microsoft 2012).

Já o *RestSharp* de acordo *com Netherland (2013) é uma* biblioteca de código livre, criada em 2010 por John Sheehan exclusivamente para o uso em integrações com *web services REST*. Sua DLL esta disponibilizada na web, assim como os códigos fontes. Dentre os principais recursos do *RestShap* estão o suporte para métodos *get, post, put,*

head, options, delete além de serialização e desserialização e capacidade de requisições síncronas e assíncronas. O RestSharp pode ser usado em diversas plataformas com suporte ao *.NET*, tais como aplicações *Desktop, Web* e em *smartphones* com sistema operacional *Windows Phone.*

2.10 *Smartphones*

A palavra provém do inglês, em sua tradução quer dizer “telefone inteligente”. Algumas características exclusivas dos *smartphones* são quanto a duração da bateria, que é menor que as dos celulares comuns devido às inúmeras funcionalidades que os *smartphones* possuem, possuem teclado QWERTY (físico ou virtual), acesso a internet sem fio ou por rede 3G, *e-mail*, câmera, filmadora, GPS, capacidade de armazenamento, *touch screen* (tela sensível ao toque) na maioria dos modelos, leitor e editor de documentos, jogos com qualidade cada vez melhor, integração com redes sociais, como *Facebook, Twitter* e demais.

Trata-se basicamente de um computador de mão (OLHAR DIGITAL, 2010).

> Como resultado da popularidade dos dispositivos eletrônicos móveis, os desenvolvedores de softwares perceberam que seus clientes não estavam mais restritos aos computadores de mesa. Os desenvolvedores reconhecerem a necessidade de software que fosse acessível para qualquer um e disponível por meio de praticamente qualquer tipo de dispositivo (DEITEL et al., 2007).

Com a afirmação de Deitel (2007) acima, observamos que foi necessário ser desenvolvido pelas grandes empresas de software, Sistemas Operacionais que pudessem atender às demandas que antes eram apenas dos computadores de mesa, agora totalmente disponíveis aos dispositivos móveis.

Para execução das funcionalidades, o *smartphone* necessita de um sistema operacional. Atualmente, três plataformas competem no mercado e merecem destaque. São elas: *Android, iPhone e Windows Phone*, respectivamente produtos das empresas *Google, Apple e Microsoft.* Para o desenvolvimento do presente trabalho, a plataforma *Windows Phone* é explorada conforme os próximos capítulos.

2.11 *Windows Phone*

Conforme Whitechapel e McKenna (2012), o telefone (*smartphone*) deve evitar confusões e facilitar a capacidade do usuário para se concentrar em concluir suas principais tarefas rapidamente. Pode-se notar que esta filosofia foi seguida na criação da tela principal do *Windows Phone*. A FIGURA 1 mostra ao usuário várias informações agrupadas de forma que uma simples visualização já é suficiente. A interface é limpa, sem elementos visuais carregados. Entender esta concepção é importante para a elaboração de futuros aplicativos. Ainda segundo os autores, um bom aplicativo deve seguir a identidade visual do Sistema Operacional.

Fonte: WHITECHAPEL e McKENNA (2013).

FIGURA 1 - Tela principal do *Windows Phone.*

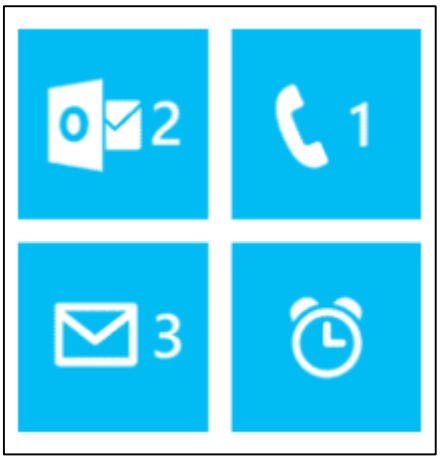
Todos os textos padrões do *Windows Phone* utilizam um modelo de fonte estilizada criada exclusivamente pela *Microsoft* para este projeto, a *Segoe WP*. Esta fonte é disponibilizada como componente nas ferramentas de criação de interfaces dos aplicativos, como será mostrado mais adiante. Contudo é possível utilizar outros padrões de texto, de acordo com a necessidade do desenvolvedor.

De acordo com Whitechapel e McKenna (2013), existe um consenso entre *designers* de interface que visam à criação de elementos gráficos que busquem representar fielmente e fisicamente os objetos da nossa realidade para dentro dos dispositivos móveis e computadores em geral. Nota-se este conceito no formato, geralmente em figuras em 3ª dimensão de alguns objetos e ícones exibidos graficamente em sistemas operacionais,

conforme a FIGURA 02. Ao criar a nova interface para o *Windows Phone*, a *Microsoft* aboliu este conceito e adotou a criação de elementos gráficos tipados, sem efeitos de sombreamento ou dimensões, conforme FIGURA 03. Esta mudança, segundo a empresa foi pensada buscando uma melhor experiência para o usuário.

Fonte: <http://cmsresources.windowsphone.com/devcenter/en-us/design/Principles/Info_2.png>

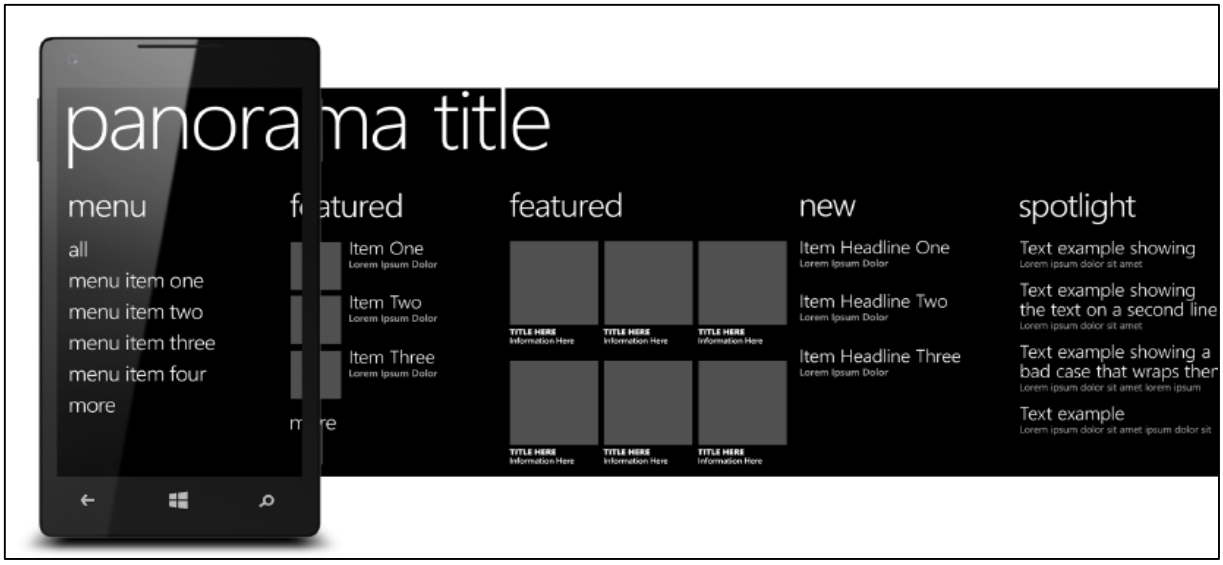
FIGURA 2 - Ícones padrões com inspiração em objetos reais.



Fonte: <http://cmsresources.windowsphone.com/devcenter/en-us/design/Principles/Info_1.png>

FIGURA 3 - Modelo de ícones do *Windows Phone*.

Outro ponto comum entre os aplicativos do *Windows Phone* é o componente Panorama. Este item permite que o desenvolvedor agrupe o conteúdo do aplicativo numa única tela, com deslocamento lateral, conforme mostra a FIGURA 4. A *Microsoft* recomenda que este modelo seja seguido para manter a identidade visual do sistema e oferecer uma visualização de conteúdo mais dinâmica.

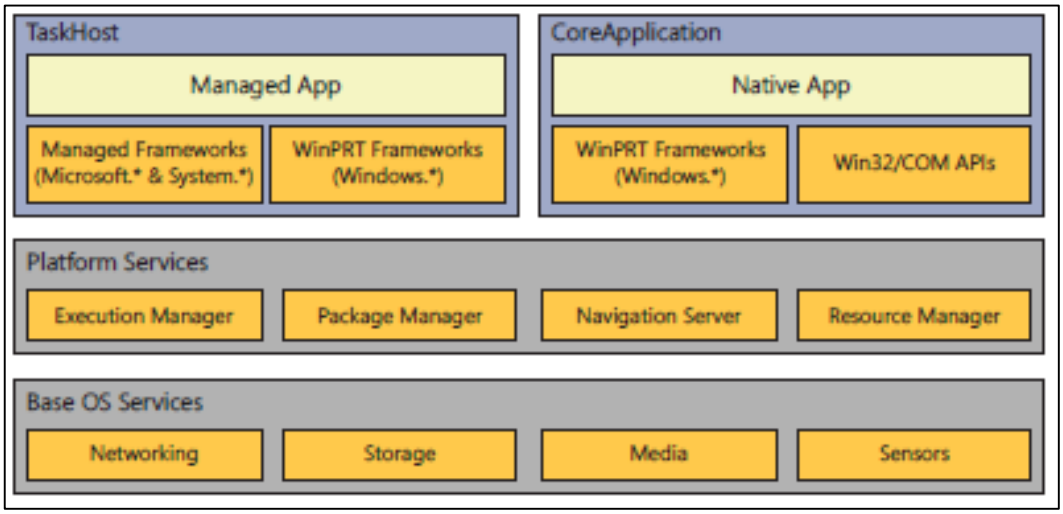


Fonte: <http://cmsresources.windowsphone.com/devcenter/en-us/design/Principles>

FIGURA 4 - Componente Panorama

2.11.1 Arquitetura

Para entendermos o modelo da arquitetura do sistema *Windows Phone*, é necessário conhecer as camadas e os componentes principais do seu *core* (núcleo). Esta compreensão é importante para os desenvolvedores, pois esta ligada diretamente a escolha do tipo de aplicativo a ser desenvolvido, conforme será explicado posteriormente neste trabalho.



Fonte: WHITECHAPEL e McKENNA (2013).

FIGURA 5 - Descrição dos componentes do núcleo do sistema.

No topo do diagrama da FIGURA 5, temos dois tipos de aplicativos, *TaskHost* e *CoreApplication*. O primeiro representa os aplicativos que são gerenciados pelo sistema operacional, desenvolvidos basicamente com XAML. Este é o principal modelo de programação desde a versão 7.x, conforme informado por Shawn Wildermuth (2011).

O segundo tipo, *CoreApplication* se refere aos que acessam os recursos nativos de hardware do *smartphone* e não são gerenciados pelo sistema operacional. Este modelo também é conhecido como *Direct3D*. Uma informação relevante, de acordo com Whitechapel e McKenna (2012), é que estes aplicativos podem ser compartilhados com o *Microsoft Windows 8*, sistema operacional de *desktops* e *tablets* da *Microsoft*.

Conforme especificado pela *Microsoft* (2012), os dois tipos de aplicativos compartilham um conjunto de serviços primários no *core* (núcleo) do Sistema. Abaixo uma breve descrição de cada um:

- ***Package Manager:*** Responsável pela instalação e desinstalação de aplicativos e por manter todos os seus metadados por todo o ciclo de vida do *software*.

- ***Execution Manager:*** Controla toda a lógica associada à execução do aplicativo. Ele cria o processo de hospedagem para o aplicativo ser executado e organiza os eventos de inicialização, desligamento e desativação. Além disso, também é responsável pelo gerenciamento das tarefas de segundo plano.

- ***Navigation Server***: Gerencia toda a navegação entre aplicativos do telefone. Seu trabalho é feito em conjunto com o *Execution Manager*. Um exemplo: Quando estamos na tela inicial e tocamos num determinado aplicativo, o *Navigation Server* direciona para o local de destino e armazena o caminho. Quando tocamos no botão voltar, ele busca o caminho armazenado e retorna para a tela inicial.

- ***Resource Manager***: Responsável por acompanhar a utilização dos recursos do sistema (gerenciamento de recursos de processamento CPU e especialmente memória) por todos os processos ativos e aplicar um conjunto de restrições sobre eles. Se um aplicativo ou processo de segundo plano excede o seu *pool* (conjunto) de recursos alocados, o usuário é alertado, a fim de garantir o funcionamento correto do aparelho.

2.11.2 Especificações de *Hardware*

Ao criar um aplicativo, é essencial para o desenvolvedor conhecer a plataforma de *hardware* onde o software será executado. Estas informações permitem que o aplicativo seja mais bem escalonado para evitar futuros problemas no uso dos recursos, como memória e armazenamento. Em dispositivos móveis, cujos recursos geralmente são mais limitados, a necessidade do conhecimento do hardware se torna vital.

Conforme publicado pela *Microsoft* em 2012, a versão atual do WP 8.0 exige alguns requisitos mínimos de *hardware*. Qualquer fabricante interessado em criar dispositivos com WP deve equipá-los com estes itens:

- Processador *dual-core Qualcomm Snapdragon* S4;
- Mínimo de 512MB de RAM para equipamentos com resolução WVGA;
- Mínimo de 1GB de RAM para equipamentos com resolução WXGA;
- Mínimo de 4GB de armazenamento interno;
- Suporte para micro-USB 2.0;
- Entrada para h*eadphone jack* de 3.5mm;
- Câmara traseira com LED ou *Xenon flash*, câmara frontal opcional, mas ambas têm de no mínimo, suportar resoluções iguais ou superiores a VGA;
- Botão de câmera dedicado;
- Acelerômetro, sensor de proximidade e sensor de luz ambiente;
- *Bluetooth* e normas de WiFi 802.11b/g (802.11n é opcional);
- GPU com suporte para DirectX;
- Tela *multi-touch* capacitivo com um mínimo de 04 pontos simultâneos.

Em relação às especificações de memória, a *Microsoft* alerta que apesar da sua recomendação mínima ser de 512MB, aplicativos terão uma experiência mais satisfatória de uso, em dispositivos equipados com 1 GB ou mais.

2.11.3 Tipos de Aplicativos

A versão mais recente do WP permite o desenvolvimento de 03 tipos básicos de aplicativos. Na TABELA 01 tem-se a descrição de cada um deles:

TABELA 1 - Descrição de tipos de aplicativos suportados no WP 8.

TIPO	DESCRIÇÃO	LINGUAGENS SUPORTADAS	UI FRAMEWORK	APIs SUPORTADAS
XAML	O tipo mais comum de aplicativo. Desenvolvido com XAML e código gerenciado.	C# e Visual Basic	XAML	*Microsoft* .NET *Windows Phone* API e WinPRT API
Mixed Mode	Este tipo de aplicativo basicamente segue o modelo XAML, porém permite a inclusão de código nativo através do componente WinPRT. Aplicativos nativos que por algum motivo precisem de interfaces providas pelo XAML também se enquadram nesta categoria.	C#, Visual Basic e C/C++	XAML, Direct3D (via *DrawingSurface*)	*Microsoft* .NET *Windows Phone* API, WinPRT API e Win32/COM API (com componentes WinPRT)
Direct3D	Aplicativos que usam apenas código nativo. Esta característica o torna a melhor opção para o desenvolvimento de jogos.	C/C++	Direct3D	WinPRT API e Win32/COM API

Fonte: WHITECHAPEL e McKENNA (2013)

3 DESENVOLVIMENTO

Para desenvolver aplicativos para o WP é necessário utilizar o Kit de Desenvolvimento de *Software Windows Phone* SDK 8.0, disponibilizado gratuitamente pela *Microsoft*. Esta SDK permite criar softwares compatíveis com a versão 8.0 e 7.8 do WP. O pacote de instalação oferece os seguintes itens:

- *Microsoft Visual Studio 2012 Express for Windows Phone.*
- *Microsoft Blend 2012 Express for Windows Phone.*
- Dispositivo Emulador do *Windows Phone.*

Abaixo são apresentados os requisitos necessários para a instalação da SDK:

Sistemas operacionais suportados:

- *Windows 8, Windows 8 Pro.*

Tipo de sistema operacional:

- *Windows* 8 64-bit (x64) versões para cliente.

Hardware:

- 6.5 GB de espaço livre no disco;
- 4 GB memória RAM;
- 64-bit (x64) CPU.

O emulador do WP 8 exige alguns requisitos específicos:

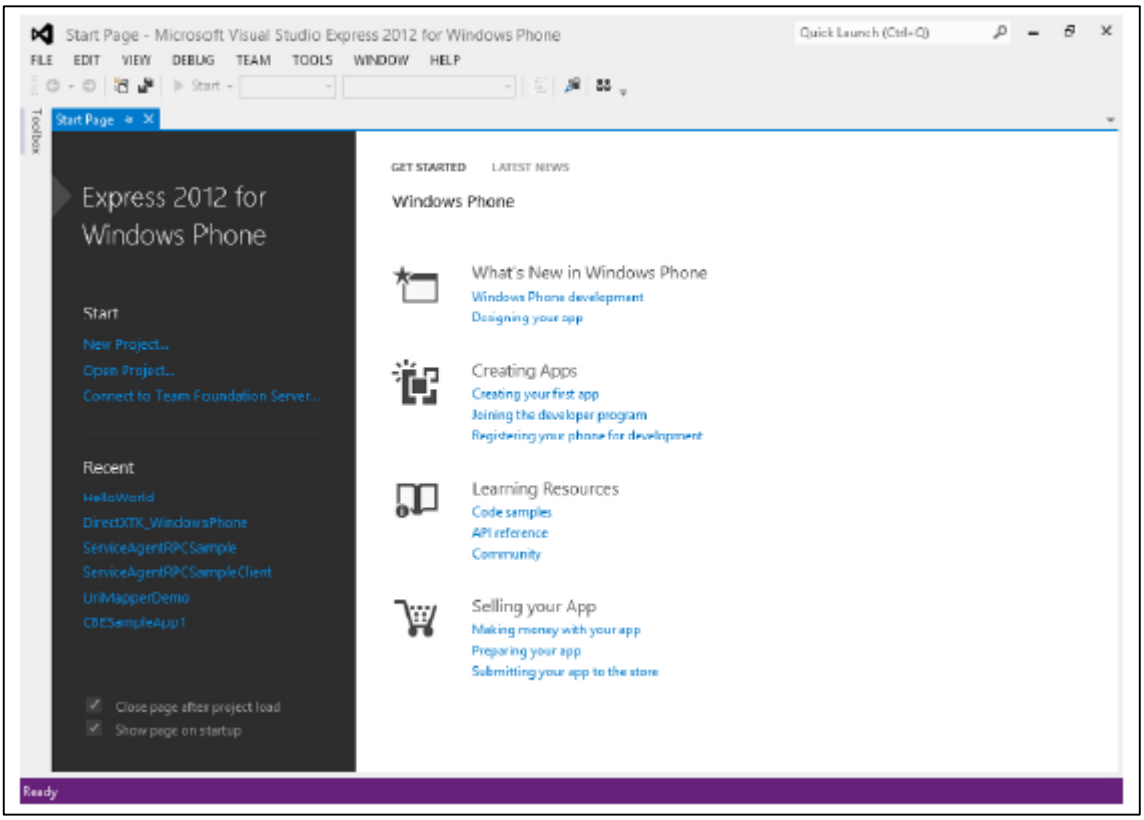
- *Windows 8 Pro* ou superior;
- Processador com suporte a *Second Level Address Translation* (SLAT).

Segundo a *Microsoft* (2012) é possível instalar a SDK caso a máquina de destino atenda apenas os primeiros requisitos listados acima, contudo o emulador não estará disponível para depuração.

3.1 Criando um aplicativo *"Hello World!"*

Existem muitas maneiras de compreender a lógica e arquitetura de um ambiente de desenvolvimento. Evidentemente que um exemplo prático agrega muito mais conhecimento aos envolvidos do que simplesmente a leitura e análise de termos teóricos. Dentro do campo da programação, é comum utilizar-se como modelo para elucidar ou mesmo ensinar o uso de uma linguagem ou ambiente de desenvolvido, o famoso "*Hello world*". Baseado nisso, o primeiro exemplo de aplicativo para *Windows Phone* que será implementado nesse trabalho de conclusão de curso será um projeto deste tipo.

Inicialmente deve-se abrir o *Microsoft Visual Studio 2012*. Ao ser iniciado, a IDE irá exibir uma página inicial ou *Start Page* conforme a FIGURA 6 mostrada abaixo. Dentre as opções disponíveis, deve-se escolher *Start -> New Project*.



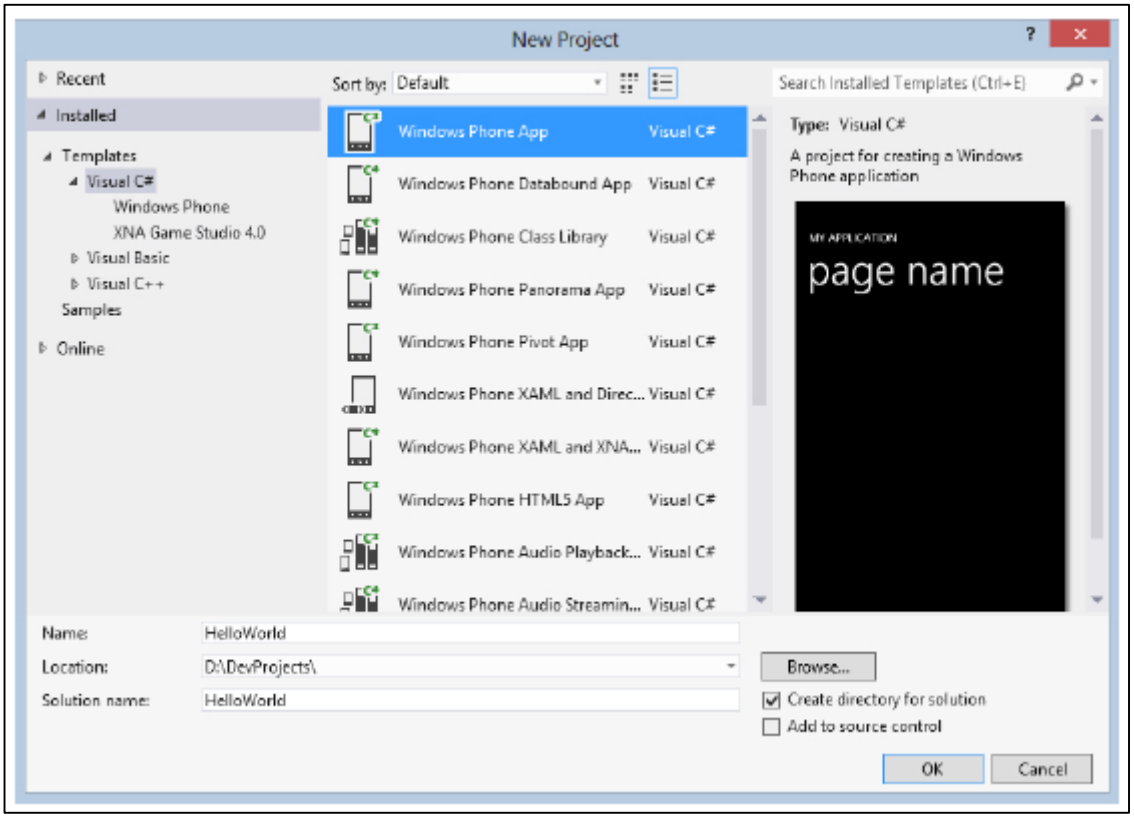
Fonte: <http://versatilias.wordpress.com/2013/01/20/windows-phone-8-creating-hello-world-app/>.

FIGURA 6 - Start Page do Visual Studio Express 2012 for Windows Phone.

O Visual Studio irá abrir a caixa de diálogo que permite a escolha da linguagem de programação (C#, *Visual Basic* ou C++) e também a escolha de um *template*, que nada mais é do que um modelo base para o aplicativo a ser desenvolvido. Nesse caso, a

escolha será um aplicativo do tipo C# -> *Windows Phone* APP. Conforme explicado anteriormente, aplicativos escritos em C#, que é uma linguagem gerenciada, são do tipo XAML.

Após a escolha do aplicativo, define-se um nome para o mesmo, nesse caso foi escolhido "*Hello World*". Também é definido o diretório para armazenar o projeto e o nome da *Solution* (FIGURA 7). No *Visual Studio*, uma *solution* é um conjunto de projetos. Pode-se inserir numa mesma *solution* vários aplicativos distintos.



Fonte: <http://versatilias.files.wordpress.com/2013/01/1-create-project.jpg?w=710>.

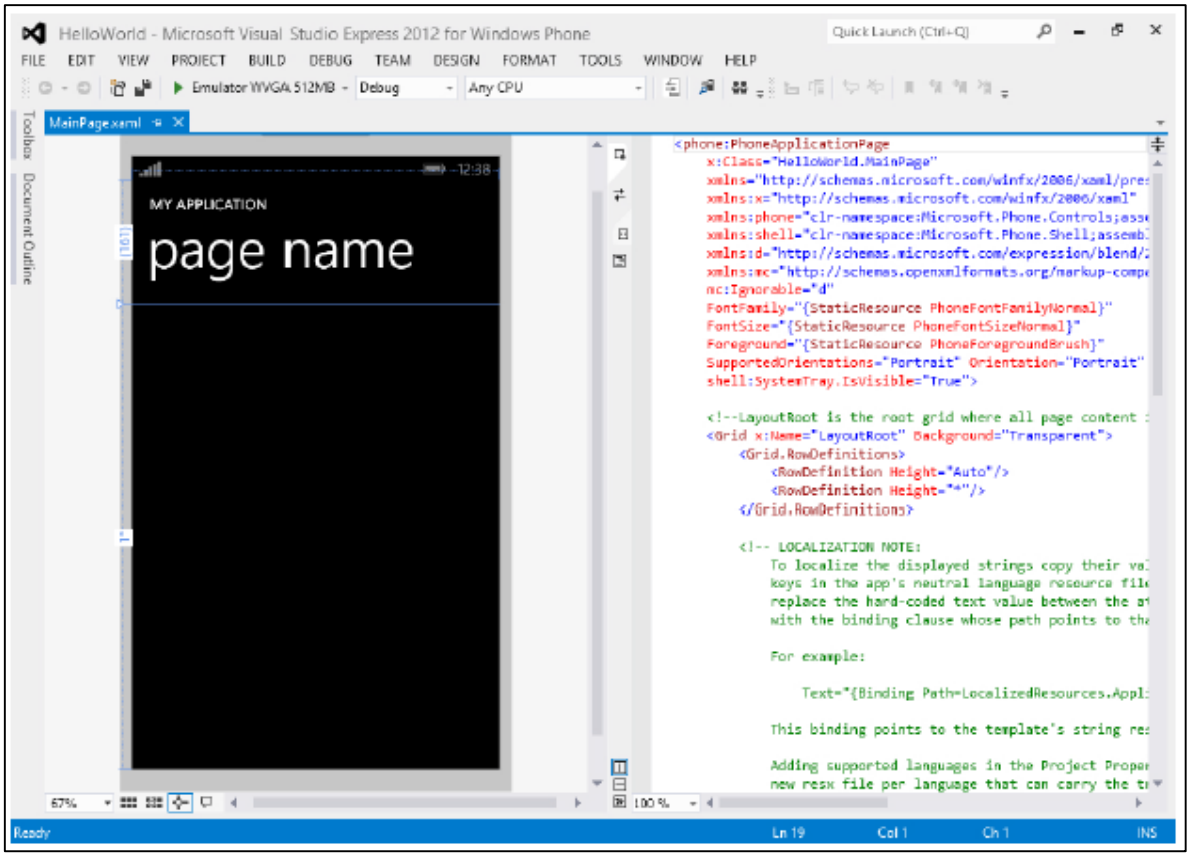
FIGURA 7 - Caixa de dialogo para definição do novo projeto.

Após a confirmação da etapa anterior, o *Visual Studio* solicita que seja informada a plataforma na qual o aplicativo será destinado. Nesse caso a versão *Windows Phone* OS 8.0 foi a escolhida, assim como mostrado na FIGURA 8.

Fonte: <http://versatilias.files.wordpress.com/2013/01/2-select-os-version.jpg?w=507>.

FIGURA 8 - Seleção da plataforma de destino do aplicativo.

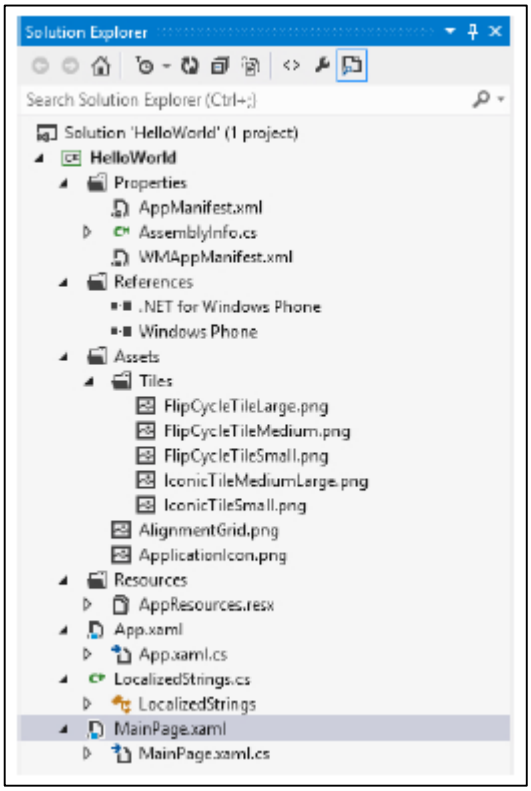
Após a confirmação da escolha, A IDE cria o projeto e abre o arquivo principal, MainPage.xaml. Por padrão, o *Visual Studio* abre o arquivo no modo de edição do código XAML (FIGURA 9).



Fonte: <http://versatilias.files.wordpress.com/2013/01/3-start.jpg?w=710>

FIGURA 9 - Arquivo MainPage.xaml no modo de edição de código.

Ao criar-se um projeto baseado num *template*, o *Visual Studio* disponibiliza uma série de arquivos básicos para o funcionamento do aplicativo. Ao analisar a barra de ferramentas *Solution Explorer*, pode-se visualizar todos estes arquivos, conforme FIGURA 10 abaixo:



Fonte: Captura de tela obtida pelos autores.

FIGURA 10 - Arquivos do projeto exibidos na *Solution Explorer* do VS.

De acordo com Whitechapel e McKenna (2012), o primeiro arquivo importante é WMAppManifest.xml, o manifesto do aplicativo. O manifesto do aplicativo contém todas as informações que o sistema operacional precisa saber sobre o aplicativo, a fim de executa-lo e submete-lo corretamente ao *smartphone*. Alguns elementos do manifesto (por exemplo, requisitos de hardware) também são utilizados durante processo de submissão.

O manifesto inclui entre outras coisas o seguinte:

- O nome do aplicativo;
- Uma referência ao ícone padrão para exibição na tela inicial;
- As resoluções suportadas;
- A lista de recursos de segurança, que exige para seu funcionamento, como localização e fotos, por exemplo, e qualquer requisito de *hardware*, tais como NFC ou uma câmera frontal.

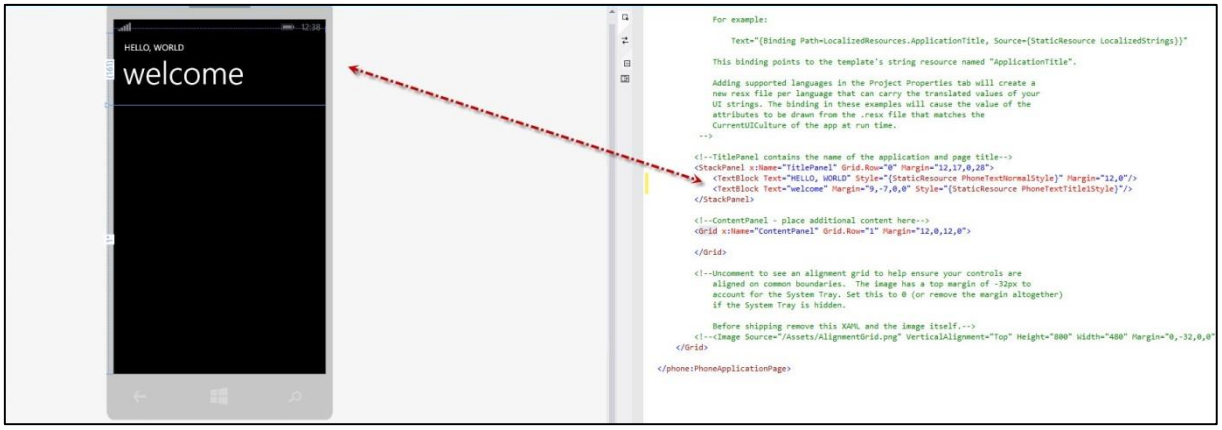
A pasta *Assets* corresponde ao diretório específico para incluir as imagens centrais que o aplicativo irá utilizar. Na raiz da pasta *Assets* existe um arquivo chamado ApplicationIcon.png. Este é um ícone padrão que é utilizado pelo SO na lista de aplicativos. A subpasta *Tiles* contém várias imagens utilizadas como ícones para os modelos do tipo *FlipCycle* e *Iconic*. O *Windows Phone* permite que o usuário escolha o tamanho das *Tiles* (telhas na tradução literal, mas que para o SO representam os ícones de atalho para acesso ao aplicativo) e estas podem possuir estes tipos, de acordo com o tamanho. Estes arquivos devem ser customizados pelo desenvolvedor antes de submeter o projeto a *Windows Store*, pois faz parte da identidade visual do aplicativo.

Os arquivos da pasta ***Resources*** como o AppResources.resx e da pasta *LocalizedStrings.cs* são utilizados para criar aplicativos com o conceito de globalização. Estes arquivos permitem que o aplicativo possa utilizar uma série de palavras específicas para o idioma de localização do *smartphone*, de forma dinâmica.

O arquivo **App.xaml** fornece um local adequado para armazenar recursos que serão utilizados por toda a aplicação, tais como os estilos de interface do usuário. Em contrapartida o arquivo de código, App.xaml.cs, contém os métodos de inicialização fundamentais e alguns manipuladores vazios para o ciclo de vida do aplicativo, tais como eventos de inicialização, ativação, desativação, e encerramento. Se o aplicativo desenvolvido precisar executar alguma função customizada durante estes eventos, o **app.xaml** deve ser editado para estes fins.

MainPage.xaml é o ponto de partida padrão para o aplicativo. Conforme explicado por Whitechapel e McKenna (2012), pode-se fazer uma analogia deste arquivo como sendo o equivalente eficaz para páginas da *web*. Eles contêm tanto a definição da interface que será exibido para o usuário, bem como o código de ligação entre *IU* e que o resto da funcionalidade da aplicação.

O *Visual Studio* oferece uma interface gráfica para a criação dos aplicativos, chamado de modo *Designer*. Em paralelo a este modo é apresentado o XAML *code*. Qualquer modificação em ambos reflete em mudanças no lado inverso. A FIGURA 11 demonstra esta característica:



Fonte: <http://versatilias.files.wordpress.com/2013/01/6-reflected-changes.jpg>

FIGURA 11 - Modo Designer e XAML Code.

No lado esquerdo é mostrado o modo designer, onde se visualiza a tela do *smartphone*. Já à direita, o código XAML responsável pela geração dos itens exibidos na tela do dispositivo. Para este projeto a propriedade dos textos foi alterada para *Hello, World* e *welcome*.

Seguindo com o desenvolvimento do aplicativo *Hello World*, no arquivo MainPage.xaml, será incluído um componente do tipo do *StackPanel* com outro componente do tipo *TextBlock* como filho, ou seja dentro do primeiro. A propriedade *Name* do componente *TextBlock* deve ser alterada para "*hellotextBlock*" e a propriedade *Text* para "*Hello from Windows Phone 8!*".

Adiante é criado um componente *Button*, que deve receber na propriedade *Content* o valor "*Say goodbye*" e "*goodbyebutton*" para o *Name*.

O Código do arquivo ficará como mostrado na FIGURA 12 abaixo:

```
<StackPanel x:Name="ContentPanel_1" Grid.Row="1" Margin="12,0,12,0">
    <TextBlock
            x:Name="helloTextBlock"
            Text="Hello from Windows Phone 8!"
            Foreground="{StaticResource PhoneAccentBrush}"
            Grid.Row="0"
            HorizontalAlignment="Center"/>
            <Button x:Name="goodbyeButton"
            Content="Say goodbye!"
            Grid.Row="1" Click="Button_Click_1"/>
</StackPanel>
```

FIGURA 12 - Trecho do código fonte do arquivo XAML.

O componente *StackPanel* oferece uma maneira simples e eficiente de configurar a exibição de elementos na tela do aplicativo. O objeto *TextBlock* representa um texto

simples e o *Button* oferece um botão que executa uma ação ao ser clicado. Esta ação é configurada no evento "*Button_Click_1*", incluindo o código abaixo (FIGURA 13) no arquivo **MainPage.xaml.cs:**

```
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    helloTextBlock.Visibility = System.Windows.Visibility.Collapsed;
    goodbyeButton.IsEnabled = false;
}
```

FIGURA 13 - Código do método Button_Click_1

Após esta alteração, deve-se executar o aplicativo para testar os resultados. Para isso é necessário selecionar no *Visual Studio* qual a versão do emulador depurará o projeto. Neste caso foi escolhida a versão *Emulator* WVGA 512 MB (FIGURA 14).

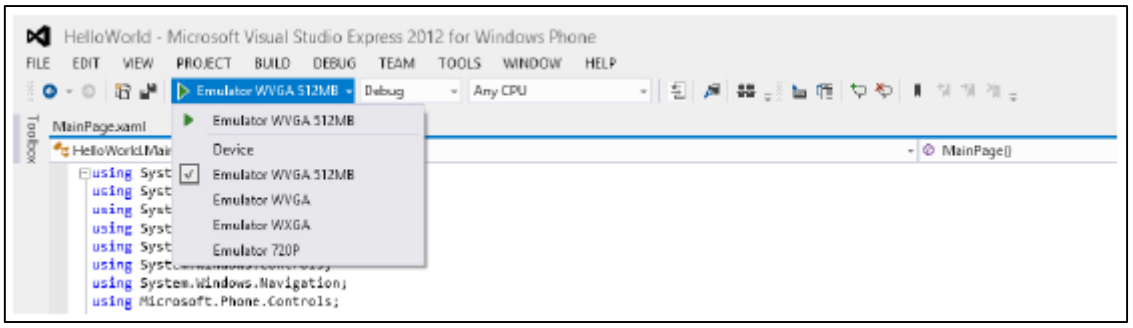


FIGURA 14 - Escolha do Emulador padrão para execução de aplicativo.

Após a escolha e execução, o VS irá compilar o projeto e abrir o emulador do *Windows Phone* 8. O resultado esperado pode ser observado abaixo (FIGURA 15):

Fonte: <http://versatilias.files.wordpress.com/2013/01/9-unclicked.jpg?w=322>

FIGURA 15 - Emulador exibindo o aplicativo *Hello World*.

Ao iniciar, o aplicativo exibe a mensagem "*Hello from Windows Phone 8*". Ao clicarmos no botão abaixo da mensagem, o texto é ocultado e o mesmo botão fica desabilitado, conforme foi definido no evento quando alteramos a propriedade *Visibility* do *TextBlock* e a propriedade *IsEnabled* do *Button*.

3.2 Análise da Comunicação do Aplicativo

Para analisar o desempenho do SO ao executar tarefas onde se exigem a interoperabilidade através de recursos *web*, neste trabalho será feita uma comparação entre duas formas distintas de comunicação empregadas entre aplicações *Windows Phone* e serviços *web*. Para isso, nos próximos capítulos será demonstrado o

desenvolvimento de um serviço *web* básico, com o uso de tecnologia REST. Para consumir este serviço será criado um aplicativo *Windows Phone*, com 2 métodos distintos de conexão, um com uso da biblioteca *Microsoft WebClient*, pertencente a DDL *System.NET* e outro aplicativo com a alternativa de código aberto, *RestSharp*, biblioteca esta desenvolvida pela comunidade de programadores.

Ao fim do desenvolvimento serão feitos testes de consumo do serviço usando os métodos distintos. O tempo de chamada e resposta dos métodos será calculado e anotado para posteriores analises e conclusões. O web service se chamará WCFAlunos e o aplicativo WSClient. O modelo foi representado conforme FIGURA 16.

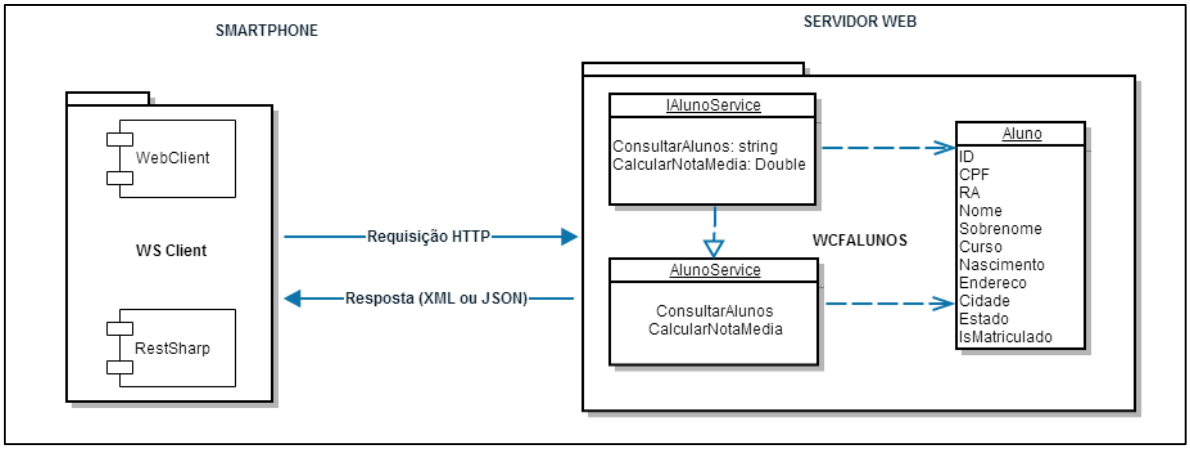


FIGURA 16 - Representação geral do modelo a ser desenvolvido

3.3 Desenvolvimento de Serviço *Web REST*

Existem diversas maneiras de se criar um serviço do tipo REST. Para este trabalho foi escolhida a IDE *Microsoft Visual Studio 2012*, linguagem C#, tecnologia ASP.NET com *.NET Framework* 4.5, e a opção de um novo projeto do tipo WCF *Service Application*. Este projeto recebeu o nome de WcfAlunos. A próxima etapa foi a inclusão de uma nova classe para representar um objeto Aluno, conforme código na FIGURA 17:

```
using System;

namespace WcfAlunos
{
    public class Aluno
    {
        public int ID { get; set; }
        public string CPF { get; set; }
        public string RA { get; set; }
        public string Nome { get; set; }
        public string SobreNome { get; set; }
        public string Curso { get; set; }
        public DateTime Nascimento { get; set; }
        public string Endereco { get; set; }
        public string Cidade { get; set; }
        public string Estado { get; set; }
        public bool IsMatriculado { get; set; }
    }
}
```

FIGURA 17 - Classe Aluno

Após a criação da entidade Aluno, foi adicionado ao projeto uma interface, posteriormente nomeada como *IAlunoService*, com dois métodos: ConsultaAlunos e CalcularNotaMedia, ambos declarados com o atributo *WebGet* e *ServiceContract*, que permitem a exposição destes métodos como serviços REST. A propriedade *UriTemplate* define o caminho usado para identificar a requisição de cada método. O código é representado conforme demonstrado na FIGURA 18 a seguir:

```
using System;
using System.Collections.Generic;
using System.ServiceModel;
using System.ServiceModel.Web;

namespace WcfAlunos
{
    [ServiceContract]
    public interface IAlunoService
    {
        [OperationContract]
        [WebGet(UriTemplate = "ConsultarAlunos/{quantidade}")]
        List<Aluno> ConsultarAlunos(string quantidade);

        [OperationContract]
        [WebGet(UriTemplate = "CalcularNotaMedia?N1={N1}&N2={N2}")]
        Double CalcularNotaMedia(string N1, string N2);
    }
}
```

FIGURA 18 - Interface IAlunoService

A implementação destes métodos foi feita na classe AlunoService, conforme o código a seguir (FIGURAS 19, 20 e 21):

```
using System;
using System.Collections.Generic;
using System.ServiceModel.Activation;

namespace WcfAlunos
{
    [AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Required)]
    public class AlunoService : IAlunoService
    {
        public List<Aluno> ConsultarAlunos(string quantidade)...

        public Double CalcularNotaMedia(string N1, string N2)...
    }
}
```

FIGURA 19 - Implementação dos métodos da Interface.

```
public List<Aluno> ConsultarAlunos(string quantidade)
{
    int count = Convert.ToInt32(quantidade);

    Aluno a;
    List<Aluno> alunos = new List<Aluno>(count);
    for (int i = 1; i <= count; i++)
    {
        a = new Aluno();
        a.ID = i;
        a.CPF = "111.111.111-11";
        a.RA = "002201400999";
        a.Nome = "José";
        a.SobreNome = "Silva";
        a.Nascimento = System.DateTime.Now;
        a.Curso = "Engenharia de Computação";
        a.Endereco = "Avenida Paulista, 1907";
        a.Cidade = "São Paulo";
        a.Estado = "SP";
        a.IsMatriculado = true;

        alunos.Add(a);
    }

    return alunos;
}
```

FIGURA 20 - Código fonte do método ConsultarAlunos

```
public Double CalcularNotaMedia(string N1, string N2)
{
    return (Convert.ToDouble(N1) + (Convert.ToDouble(N2))) / 2;
}
```

FIGURA 21 - Código fonte do método CalcularNotaMedia

O primeiro método, ConsultarAlunos, recebe a quantidade de registros que devem ser gerados e através de um laço condicional FOR cria uma lista de objetos do tipo Alunos. Ao final esta lista é retornada. Já o segundo método retorna um valor do tipo *double* com o calculo da média das notas recebidas via parâmetro.

WCF Service Applications por padrão geram APIs com *self-hosting*, isto é, não necessitam do IIS e são hospedadas como serviços locais no servidor de aplicação. Nesse caso, como é preciso que o serviço seja acessado na nuvem via HTTP, foram alteradas as propriedades do arquivo *web.config* além da inclusão do arquivo global.asax para definição de uma rota para o acionamento do serviço.

O serviço, ao ser finalizado foi testado no navegador *Google Chrome*. O acesso é feito pela informação da URL seguida do método de consulta e seus parâmetros. O resultado é exibido em formato XML na FIGURA 22.

FIGURA 22 - Resultado do Teste de Consumo.

3.4 Criação da Interface do Aplicativo

Para a criação do aplicativo *Windows Phone* que se comunica com o serviço *web* desenvolvido anteriormente foi usado o *Microsoft Visual Studio 2012 Professional*.

Foi criado um novo projeto do tipo *Windows Phone Application*, em linguagem Visual C#. O projeto foi nomeado como WPAppTester. Foi escolhida a versão 4.0 da *Microsoft .NET Framework*. A versão de SO foi definida como 8 e o emulador escolhido para testes, o *Emulator WVGA 512MB.*

Foram adicionados ao projeto 04 novos arquivos XAML para representar as telas de navegação do aplicativo: Sobre.XAML, Configuracao.XAML, Alunos.XAML e Media.XAML. Estes arquivos foram alterados de forma a exibir os elementos gráficos necessários para a execução dos testes. O arquivo sobre exibe um texto descritivo do aplicativo. Já o arquivo de configurações recebeu dois combos para escolha da biblioteca de conexão e uma caixa de texto para informar o endereço do *web service*. Os arquivos Media e Alunos receberam caixas de texto para a entrada dos parâmetros de consulta de média de nota e alunos, respectivamente.

Além disso, foi incluída nas paginas o componente *ApplicationBar*, responsável pela exibição de uma barra inferir com os botões de ação de cada página. Este objeto é muito comum em aplicativos *Windows Phone*, conforme foi verificado na *Windows Phone Store*. A FIGURA 23 abaixo exibe o código para criação em tempo de execução da barra na página que consulta a média de notas.

```
31          private void CriarBarraInferior()
32          {
33              ApplicationBar = new ApplicationBar();
34              ApplicationBar.Opacity = 0.5;
35
36              appBarButtonSearch = new ApplicationBarIconButton();
37              appBarButtonSearch.Text = "calcular";
38              appBarButtonSearch.Click += appBarButtonCalcular_Click;
39              appBarButtonSearch.IconUri = new Uri("/Images/check.png", UriKind.Relative);
40
41              appBarButtonSettings = new ApplicationBarIconButton();
42              appBarButtonSettings.Text = "configurar";
43              appBarButtonSettings.Click += appBarButtonSettings_Click;
44              appBarButtonSettings.IconUri = new Uri("/Images/feature.settings.png", UriKind.Relative);
45
46              ApplicationBar.Buttons.Add(appBarButtonSearch);
47              ApplicationBar.Buttons.Add(appBarButtonSettings);
48          }
```

FIGURA 23 - Código fonte do método CriarBarraInferior().

Na linha 33 e 34 é criada uma barra de atalhos para o aplicativo através do uso de uma instancia da classe *ApplicationBar,* que recebe a opacidade 0.5. Da linha 36 a 44, são criados dois botões com o uso da classe *ApplicationBarIconButton*, o calcular e configurar. Cada um recebe um método no respectivo evento *click*. Nas linhas 46 e 47 os botões são adicionados na barra criada anteriormente.

O resultado obtido ao executar o emulador foi o mostrado na FIGURA 24 que segue abaixo:

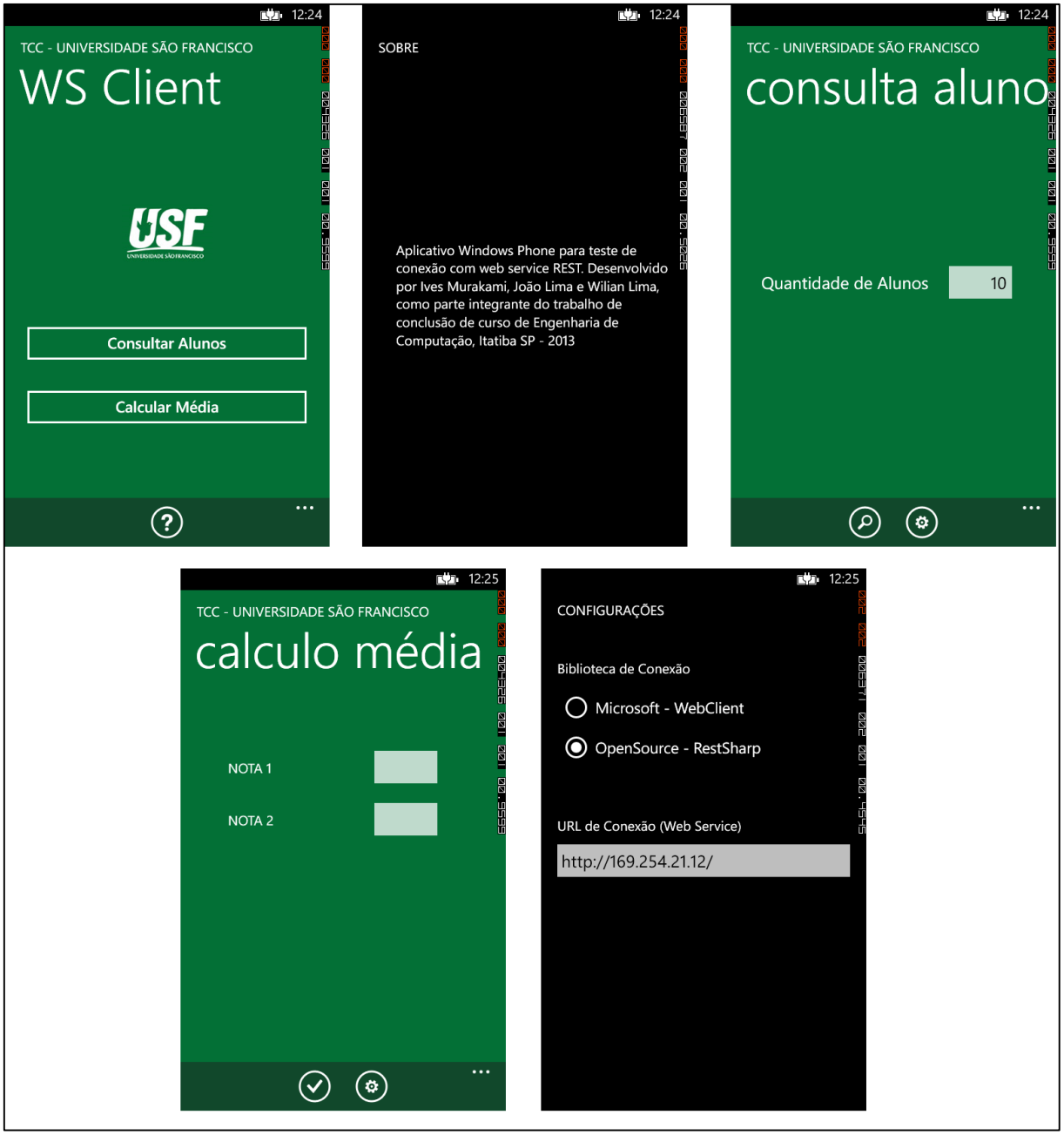


FIGURA 24 - Interface do aplicativo WPAppTester.

O arquivo configuracoes.xaml.cs trabalha com a classe *IsolatedStorageSettings*, que é capaz de armazenar informações localmente no dispositivo. Foi criada uma chave para isolar a configuração da biblioteca de conexão ao *web service*, ISO_KEY_CONEXAO e outra para guardar o endereço do serviço, ISO_KEY_URL.

Os arquivos *media.xaml.cs e alunos.xaml.cs* receberam dois métodos (FIGURA 25), disparados pelos eventos dos botões calcular e consultar respectivamente. A sintaxe de

cada um é muito semelhante, pois tem a mesma função, controlar a exibição de componentes visuais, verificar as configurações escolhidas na configuração e redirecionar para as funções de consulta específicas.

```
56          private void appBarButtonCalcular_Click(object sender, EventArgs e)
57          {
58              txtResultado.Text = "";
59              appBarButtonSearch.IsEnabled = false;
60              appBarButtonSettings.IsEnabled = false;
61              txtN1.IsEnabled = false;
62              txtN2.IsEnabled = false;
63
64              _progressIndicator = new ProgressIndicator
65              {
66                  IsIndeterminate = true,
67                  Text = "Consultando...",
68                  IsVisible = true,
69              };
70
71              SystemTray.SetIsVisible(this, true);
72              SystemTray.SetProgressIndicator(this, _progressIndicator);
73              SystemTray.SetOpacity(this, 1);
74              _isoSettings = IsolatedStorageSettings.ApplicationSettings;
75
76              if (_isoSettings["ISO_KEY_CONEXAO"].ToString() == "RestSharp")
77                  CalcularMediaOpenSourceRestSharp(txtN1.Text, txtN2.Text);
78              else
79                  CalcularMediaMicrosoftWebCliente(txtN1.Text, txtN2.Text);
80          }
```

FIGURA 25 - Método appBarButtonCalcular_Click.

Os códigos representados da linha 58 até 62 são responsáveis por desabilitar elementos visuais da tela durante o tempo da consulta, como o botão consultar e configurar além da caixa de texto de inserção das notas. Já os códigos da linha 64 até 69 criam uma barra de progressão, com uma instancia da classe *ProgressIndicator* na tela, para sinalizar ao usuário que o aplicativo esta realizando a consulta.

Na linha 71, 72 e 73, é usada a classe *System.Tray* em conjunto com a *ProgressIndicator*, para configurar a maneira que a barra de progresso será exibida ao usuário.

As informações do registro da aplicação, como as configurações definidas para a consulta são recuperadas na linha 74 através do uso da classe *IsolatedStorageSettings*. Da linha 76 à 79 o método verifica o conteúdo da configuração recuperada e redireciona a execução para a consulta definida, *WebClient ou RestSharp.*

Foi criada uma função simples para formatar o tempo obtido e exibir novamente os controles visuais (FIGURA 26):

```
120         private void ExibirResultados()
121         {
122             _progressIndicator.IsVisible = false;
123             SystemTray.SetIsVisible(this, false);
124 
125             TimeSpan ts = horaTermino - horaInicio;
126             txtResultado.Text = String.Format("Tempo total da consulta: {0:00}h:{1:00}m:{2:00}s.{3:000}ms",
127             ts.Hours, ts.Minutes, ts.Seconds,
128             ts.Milliseconds);
129 
130             appBarButtonSearch.IsEnabled = true;
131             appBarButtonSettings.IsEnabled = true;
132             txtN1.IsEnabled = true;
133             txtN2.IsEnabled = true;
134         }
```

FIGURA 26 - Método *ExibirResultado*.

Neste método existem rotinas simples, para remover a barra de progressão (linhas 122 e 123), calcular, formatar e exibir o tempo total calculado (linhas 125 a 128) e habilitar novamente os controles da tela (linhas 130 a 133).

3.5 Desenvolvimento dos Métodos de Consumo de Serviço

Para analisar o consumo do serviço web usando a biblioteca *Microsoft WebClient* na consulta de alunos foi necessário criar dois métodos, sendo o primeiro o principal acionado pelo método descrito no capitulo anterior e o segundo um auxiliar que foi vinculado através de um *delegate* ao evento *DownloadStringCompleted* do objeto instanciado na classe *WebClient*. O código pode ser verificado na FIGURA 27 abaixo:

```
101         private void ConsultaAlunosMicrosoftWebCliente(string Quantidade)
102         {
103             _isoSettings = IsolatedStorageSettings.ApplicationSettings;
104 
105             string ServiceUri = _isoSettings["ISO_KEY_URL"].ToString() + ConsultaAlunos + Quantidade;
106             WebClient proxy = new WebClient();
107             proxy.DownloadStringCompleted +=
108                 new DownloadStringCompletedEventHandler
109                     (proxy_DownloadAlunosCompleted);
110 
111             horaInicio = System.DateTime.Now;
112             proxy.DownloadStringAsync(new Uri(ServiceUri));
113         }
114 
115         private void proxy_DownloadAlunosCompleted(object sender, DownloadStringCompletedEventArgs e)
116         {
117             horaTermino = DateTime.Now;
118             MessageBox.Show(e.Result);
119             ExibirResultados();
120         }
```

FIGURA 27 - Método *ConsultaAlunosMicrosoftWebCliente*

O método *ConsultaAlunosMicrosoftWebCliente* recupera as configurações armazenadas localmente na linha 103 e cria a URI de conexão na linha 105. Na linha 106, é criado um objeto chamado *proxy*, instanciado da biblioteca *WebClient.* O atributo *DownloadStringCompleted* recebe um ponteiro para o método *prox_DownloadAlunosCompleted* conforme o código da linha 107 à 109. Na linha 111 o calculo de tempo é iniciado e finalmente na linha 112 é feita a chamada assíncrona do *web service.*

O método atribuído a propriedade do objeto *proxy_DownloadAlunosCompleted é implementado da linha 115 a 120.* Dentro deste método, o tempo de termino é armazenado e o resultado da consulta mostrado na tela através de uma *MessageBox.*

O método de consulta usando a biblioteca *RestSharp* tem uma concepção mais simplificada que o primeiro, pois o componente abstrai a necessidade de um método auxiliar para receber o retorno da chamada (FIGURA 28).

```
84          private void ConsultaAlunosOpenSourceRestSharp(string Quantidade)
85          {
86              _isoSettings = IsolatedStorageSettings.ApplicationSettings;
87
88              var client = new RestClient(_isoSettings["ISO_KEY_URL"].ToString());
89              var request = new RestRequest(ConsultaAlunos + Quantidade, Method.GET);
90
91              horaInicio = System.DateTime.Now;
92              client.ExecuteAsync(request, response =>
93              {
94                  horaTermino = System.DateTime.Now;
95                  MessageBox.Show(response.Content);
96                  ExibirResultados();
97              });
98
99          }
```

FIGURA 28 - Código do método ConsultaAlunosOpenSourceRestSharp

O método acima também recupera informações locais no *smartphone*, na linha 86. Para realizar a consulta são criados dois objetos da classe *RestClient*. O *client* (linha 88) representa o endereço absoluto do web service e o *request* (linha89) a URI da consulta com os parâmetros adicionados. Na linha 91, é armazenado o tempo inicial da consulta. A partir da linha 92 até a 97, a requisição é iniciada, o tempo final armazenado e o resultado exibido através da *MessageBox*.

Os métodos para a consulta de nota média basicamente seguem a mesma logica dos explicados acima. A única diferença ocorre na conexão com o *web service*. Nesse caso, A *Uri* usada é a CalcularNotaMedia.

3.6 Testes e Medições

Todos os testes de consumo do serviço foram realizados no aparelho *Nokia Lumia* 520, com *Windows Phone* 8. O dispositivo possui 512MB de memória, processador *Qualcomm SnapDragon* com dois núcleos de 1000 MHz. O Serviço *Web* foi disponibilizado no *Microsoft* IIS Express de um computador *Microsoft Windows 8*, conectado a uma rede local privada. O *smartphone* foi conectado nesta mesma rede através do uso da conexão WIFI. O endereço do serviço *web* foi apontado no aplicativo para o IP local do computador, no caso 169.254.21.12 na porta 80.

A partir das configurações acima, foram realizados 2 baterias de testes iniciais com o método do *web service* ConsultarAlunos, retornando um *array* de *strings* no formato XML. As quantidades de itens retornados foram na primeira bateria 10 (FIGURA 29) e na segunda 100 registros (FIGURA 30). Em cada bateria foram realizados 5 chamadas de consumo para cada biblioteca. Os tempo foram exibidos e anotados, no formato 00:00:00:000 correspondendo a horas, minutos, segundos, e milésimos de segundo. As FIGURAS 29, 30 e 31 abaixo mostram os resultados das medições realizadas:

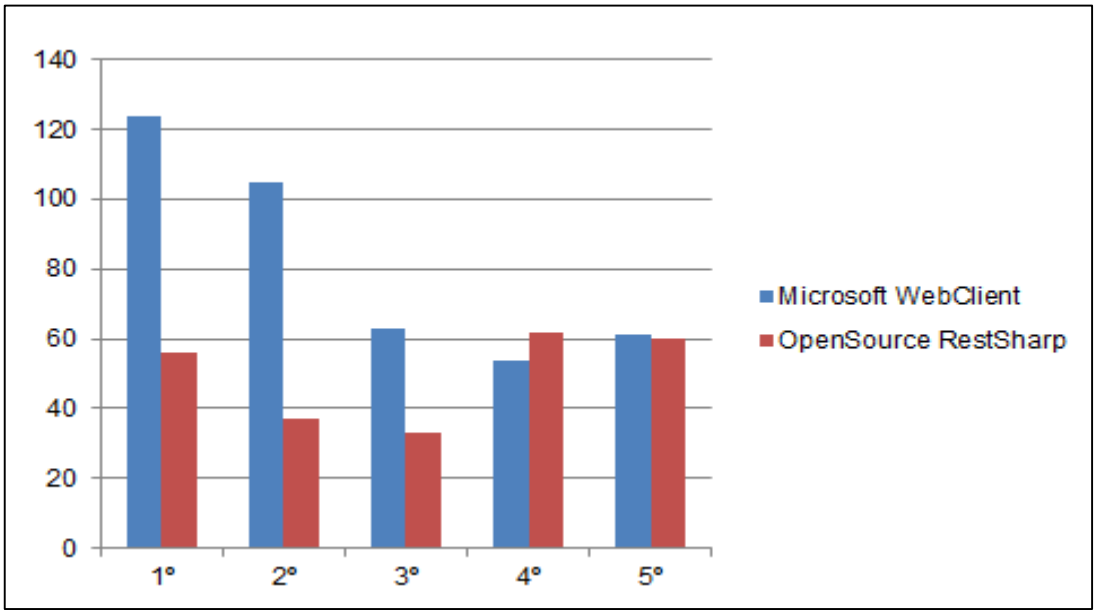


FIGURA 29 - Tempos calculados para consulta de 10 registros com retorno no formato XML.

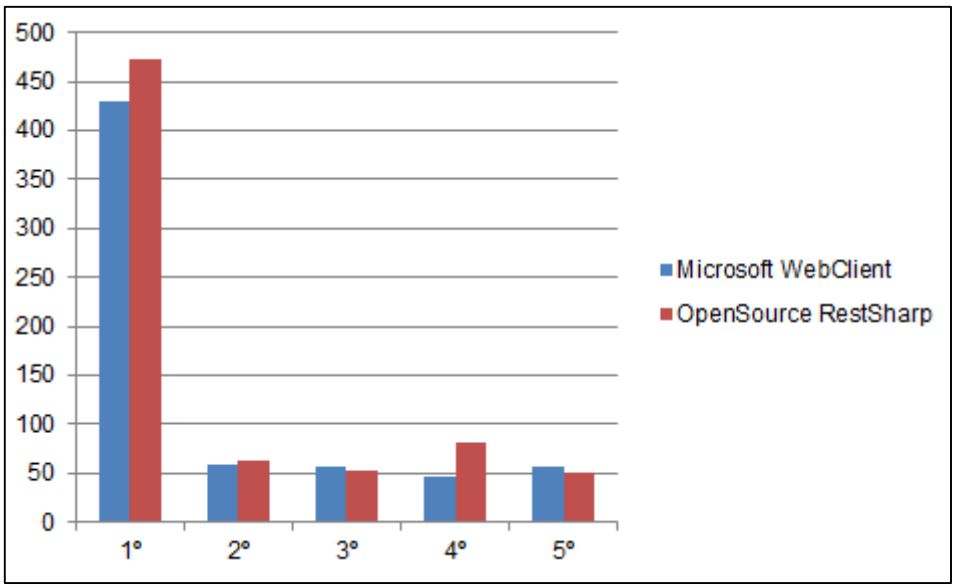


FIGURA 30 - Tempos calculados para consulta de 100 registros com retorno no formato XML

Para a segunda etapa dos testes, o método ConsultarAlunos do web service foi alterado para retornar os resultados no formado JSON, *Java Script Object Notation*. Novamente houve duas baterias de teste, com 5 chamadas, conforme FIGURAS 31 e 32.

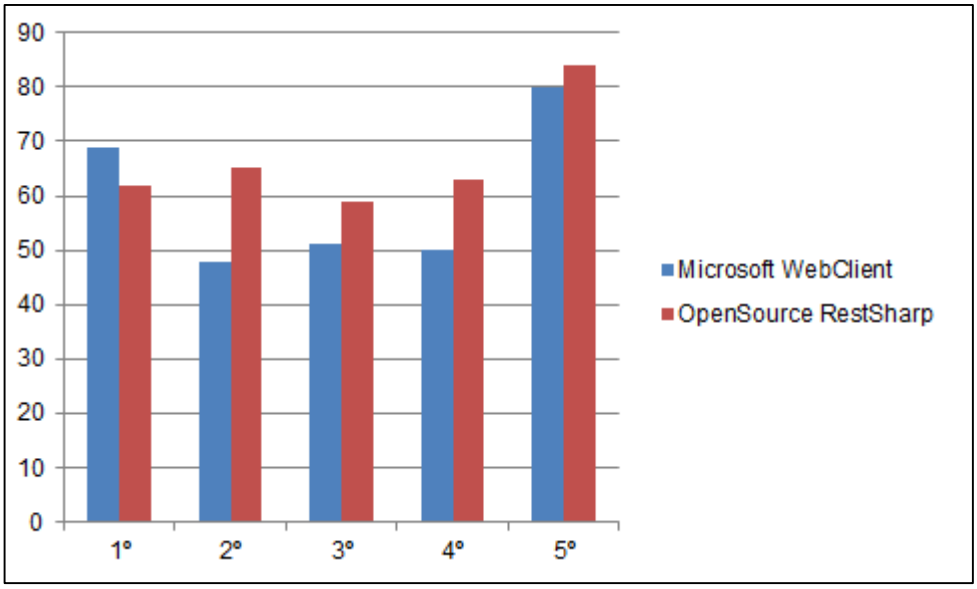


FIGURA 31 - Tempos calculados para consulta de 10 registros com retorno no formato JSON.

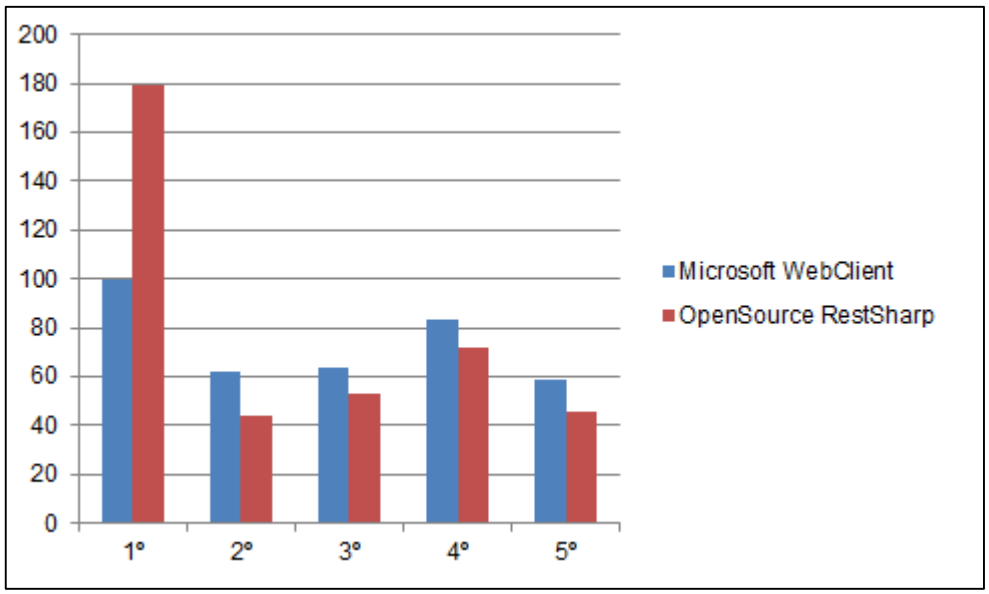


FIGURA 32 - Tempos calculados para consulta de 100 registros com retorno no formato JSON.

O Método do *web service* CalcularNotaMedia foi testado 5 vezes para cada biblioteca com 2 baterias de testes, na primeira retornando XML e na segunda JSON. As FIGURAS 33 e 34 exibem os valores obtidos nestes testes. O calculo executado foi a média das notas 8,7 e 6,5, com resultado igual a 7,6.

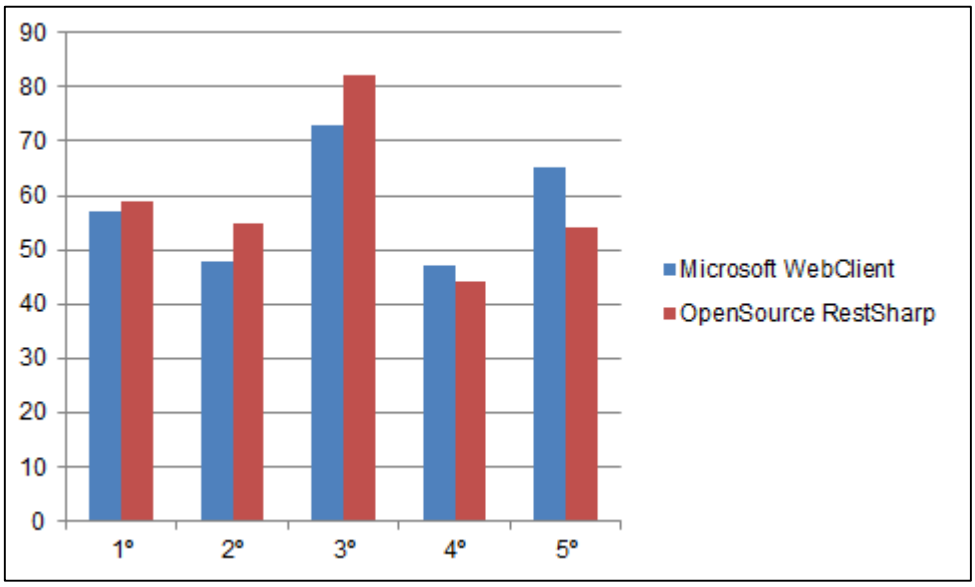


FIGURA 33 - Tempos de consulta para calculo de média com retorno no formato XML.

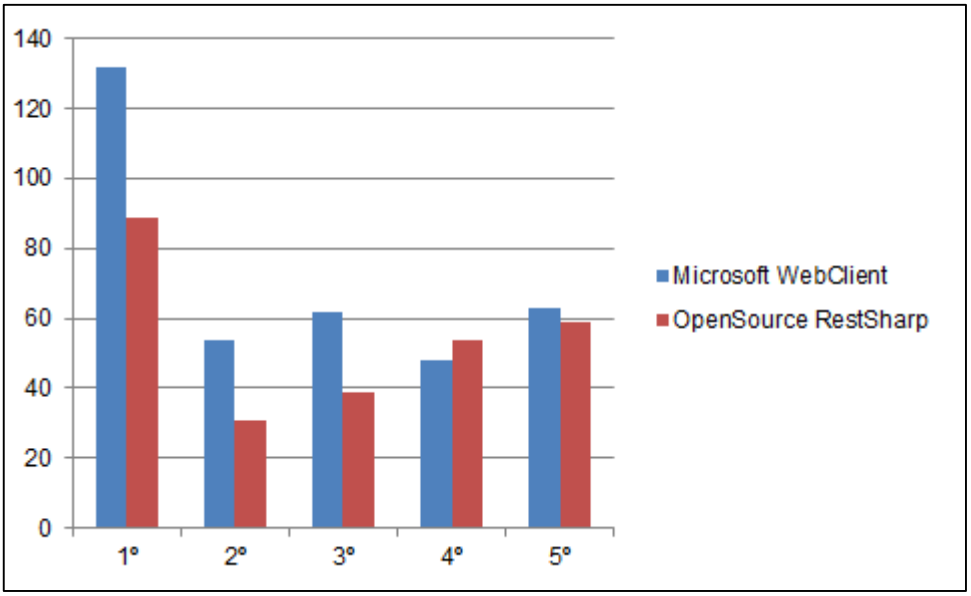


FIGURA 34 - Tempos de consulta para calculo de média com retorno no formato JSON.

3.7 Testes com o serviço indisponível.

Para testar a reação do aplicativo e do sistema operacional mediante uma falha na disponibilidade do serviço ou conexão, foi alterado o endereço IP do *web service* na configuração de forma que este não fosse localizado pelas bibliotecas. A partir dai foi realizada uma consulta simples do método consultarAlunos, com cada biblioteca. Os resultados deste teste estão expostos na tabela 2:

TABELA 2 - Testes do aplicativo com *web service* propositalmente inacessível.

BIBLIOTECA	TEMPO DE RESPOSTA	RESULTADOS
WebClient	8s	O aplicativo foi encerrado pelo SO.
RestSharp	7s173ms	O aplicativo exibe janela de mensagem sem texto.

3.8 Análise dos Resultados

A partir dos valores obtidos nos testes executados anteriormente é possível verificar que a média aproximada de tempo de execução da consulta de alunos, com 10

registros no formato XML foi de 81 milésimos de segundo para a biblioteca *WebClient*, contra apenas 50 da *RestSharp*. Nesse caso a *RestSharp* foi mais eficiente no processo. Já a consulta com 100 registros no mesmo formato XML gerou média de tempo de 128 milésimos de segundo para o *WebClient* contra 143 da *RestSharp*.

As consultas de 10 registros no formato JSON geraram medias de 60 milissegundos para a *WebClient* e 67 para a *RestSharp*. Comparando com os valores obtidos no formato XML, percebe-se que o *WebClient* obteve um ganho considerado de tempo trabalhando com JSON. Já a biblioteca *RestSharp* levou mais tempo para consultar os mesmos 10 registros. Quando a consulta foi realizada com 100 registros em JSON, o *WebClient* obteve média de 74 e o *RestSharp* 79. Apesar da pouca diferença, é evidente que o *WebClient* é mais rápido nos dois cenários (10 e 100 registros em JSON).

O tempo médio do calculo de notas obteve valores próximos no teste em XML em ambas as bibliotecas, 58 e 59 milissegundos. Já o calculo com retorno em JSON foi mais rápido na *RestSharp*, com tempo de 54 milissegundos contra 72 da *WebClient*.

Fazendo-se uma média geral de todas as consultas, observa-se valores próximos, tendo a *WebClient* 78 milissegundos e a *RestSharp* 75, conforme FIGURA 35.

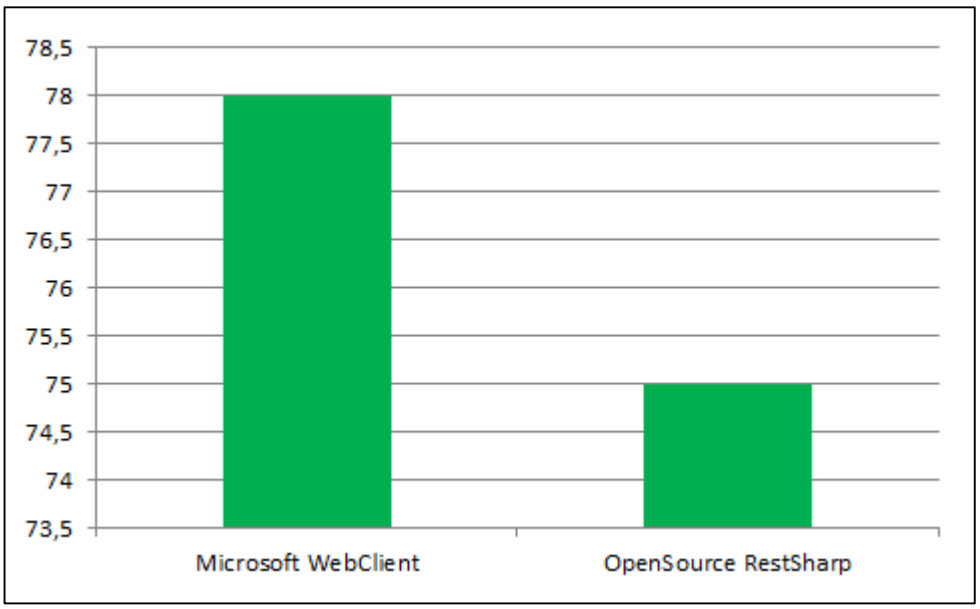


FIGURA 35 - Média Geral dos Resultados

Baseado no teste de consumo com o *web service* indisponível, foi verificado que a biblioteca *Microsoft* encerrou inesperadamente o aplicativo enquanto que o *RestSharp* foi capaz de exibir um alerta e manter a continuidade da aplicação.

4 CONCLUSÕES

Após analise da plataforma e dos recursos oferecidos para o desenvolvimento de aplicativos para *Windows Phone*, conclui-se que a *Microsoft* oferece um conjunto variado de ferramentas para a criação de *softwares*. Por possuírem interfaces gráficas, gerenciadores de código, emuladores nativos e vasta documentação, pode-se dizer que a empresa esta investindo de forma a tornar este desenvolvimento mais fácil e acessível à comunidade de programadores. Evidente que o motivo de tal ação esta ligada ao menor número de aplicativos da *Windows Phone Store*, em comparação aos seus concorrentes diretos. A MS tem interesse que novos aplicativos sejam criados e disponibilizados para seu SO e isto se reflete na sua conduta em relação à disponibilização destas ferramentas.

Neste trabalho foi possível verificar que serviços REST são boas alternativas para a integração e interoperabilidade entre servidores *web* e aplicativos móveis na plataforma *Windows Phone*. Esta conclusão se deve aos tempos de resposta obtidos na consulta dos dados de Alunos, onde foi verificado que mesmo com uma quantidade razoável de informações, o serviço respondeu com rapidez e sem falhas.

Além disso, os resultados dos testes demonstraram que além da própria *Framework .NET*, existem outras boas opções de componentes de terceiros, como o *opensource RestSharp*, que foi capaz de executar as tarefas de conexão, consumo e retorno do serviço de forma tão ou mais satisfatória que quanto o modelo padrão da *Microsoft*.

Pode-se atribuir este desempenho ao esforço da comunidade de desenvolvedores, em otimizar e melhores recursos existentes *da .NET framework e do C#*. Como esta biblioteca foi exclusivamente desenvolvida para o uso em chamadas *HTTP E REST*, é compreensível que consiga trabalhar de forma mais apropriada e ajustada para estas requisições, e consequentemente, obter melhor desempenho de execução do que o *WebClient* que é de uso mais genérico.

4.1 Trabalhos Futuros

Neste trabalho optou-se por uma abordagem mais ampla a respeito dos recursos do *Windows Phone* e a criação de clientes *web services*. Contudo, o tema é muito rico e complexo e existem diversos direcionamentos possíveis para futuros trabalhos. Abaixo seguem algumas sugestões:

- Uso de AJAX em clientes *Windows Phone*;
- Comparativo de desempenho entre clientes de *Web Services SOAP* para *Windows Phone* nas versões 7.8 e 8.0;
- Otimização de desempenho de clientes de *Web Services*;
- Tratamento de exceções em aplicativos *Windows Phone.*

REFERÊNCIAS

ALBINADER NETO, Jorge Abilio; LINS, Rafael Dueire. **Web Services em Java**. Rio de Janeiro: Brasport, 2006.

ASSOCIATION, Online Publishers. **Pesquisa diz que Android lidera mercado nos EUA, mas iPhone vende mais aplicativos.** Disponível em: <http://tecnologia.uol.com.br/noticias/redacao/2012/08/27/pesquisa-diz-que-android-lidera-mercado-nos-eua-mas-iphone-vende-mais-aplicativos.htm>. Acesso em: 27 ago. 2012.

BASIURA, Russ et al. **Professional ASP.NET Web Services:** De programador para programador. São Paulo: Pearson Education, 2003. 712 p.

BORGES JUNIOR, Mauricio Pereira. **Aplicativos Moveis:** Aplicativos para dipositivos móveis, usando o C#.NET com a ferramenta Visual Studio .NETe com Banco de Dados MySQL e SQL Server. Rio de Janeiro: Ciencia Moderna, 2005. 144 p.

BORGES JUNIOR, Mauricio Pereira. **Desenvolvendo WebServices:** Guia rápido usando Visual Studio .NET com Banco de Dados SQL Server. 1. ed. Rio de Janeiro: Editora Ciencia Moderna Ltda., 2005. 144 p.

DAHL, O. J.; DIJKSTRA, E. W.; HOARE, C. A . R. ***Structured Programming***. London: Academic Press, 1972. Capítulo: I-Notes on Structured Programming. , 220 p.

DEITEL, Harvey M. et al. **C# - Como programar**, São Paulo: Pearson Makron Books. 2007. 1098 p.

DELLA VALLE, James. **Compra da Nokia coloca Microsoft na briga com Apple e Samsung.** Disponível em: <http://veja.abril.com.br/noticia/vida-digital/compra-da-nokia-coloca-microsoft-na-briga-com-apple-e-samsung>. Acesso em: 06 set. 2013.

DUNAWAY, Robert B. **The book of Visual Studio .NET:** A guide for developers. Indianapolis: No Starch Press, 2002. 369 p.

FERREIRA, Silvio. **Curso Prático de Windows Server**: Guia completo de configuração, manutenção e otimização de redes e servidores Windows. São Paulo: Digerati Books, 2010. 112 p.

FISCHER, Alice E.; GRODZINSKY, Frances. ***The Anatomy of Programming Languages***. Englewood Cliffs, New Jersey: Prentice Hall, 1993. 557 p.

GARCIA, Marcus. **Visual Studio Team System:** Team Foundation Server. Rio de Janeiro: Brasport, 2007. 232 p.

GONÇALVES, Eduardo Corrêa. **Introdução ao formato JSON.** Disponível em: <http://www.devmedia.com.br/introducao-ao-formato-json/25275>. Acesso em: 25 out. 2013.

GRALLA, Preston. **Microsoft perde market share e passa a deter só 20% do mercado de TI.** Disponível em: <http://computerworld.uol.com.br/tecnologia/2012/12/10/microsoft-perde-market-share-e-passa-a-deter-so-20-do-mercado-de-ti/>. Acesso em: 10 dez. 2012.

JOHNSON, Bruce. **Professional Visual Studio 2012**. Indianapolis: John Wiley & Sons, 2013. 1061 p.

JSON.ORG. **Introdução ao JSON.** Disponível em: <http://www.json.org/json-pt.html>. Acesso em: 25 out. 2013.

LABRIOLA, Michael; TAPPER, Jeff; BOLES, Matthew. **Adobe Flex 4:** Treinamento direto na fonte. Rio de Janeiro: Altabooks, 2012. 466 p.

LIBERTY, Jesse; XIE, Donald. **Programando C# 3.0.** 5. ed. Rio de Janeiro: Alta Books, 2009. 458 p.

LIPPMAN, Stanley B. **C#. Um guia prático:** Tradução Werner Loeffler. 1. ed. Porto Alegre: Boookman, 2003. 316 p.

LOTAR, Alfredo. **Como Programar com ASP.NET e C#.** 2. ed. São Paulo: Novatec, 2010. 656 p.

MICROSOFT. **Microsoft Virtual Academy**: Treinamento gratuito da Microsoft oferecido por especialistas. Disponível em: <http://www.microsoftvirtualacademy.com>. Acesso em: 14 mar. 2012.

MICROSOFT. **Aplicativos da Windows Store:** Visão Geral da XAML. Disponível em: <http://msdn.microsoft.com/pt-br/library/windows/apps/hh700354.aspx>. Acesso em: 05 out. 2013.

MICROSOFT. **Centro de Desenvolvimento:** Desenvolvedores. Designers. Negócios. Disponível em: <http://msdn.microsoft.com/pt-BR/windows/>. Acesso em: 05 out. 2013.

MICROSOFT DEVELOPER NETWORK – MSDN. **Introdução ao Visual Studio**. Disponível em: <http://msdn.microsoft.com/pt-br/library/fx6bk1f4(v=vs.90).aspx>. Acesso em: 15 out. 2013.

MICROSOFT DEVELOPER NETWORK – MSDN. **Plataformas e ferramentas de desenvolvimento**. Disponível em: <http://msdn.microsoft.com/pt-br>. Acesso em: 21 abr. 2012.

MICROSOFT DEVELOPER NETWORK – MSDN. **Tour guiado do Visual Studio**. Disponível em: <http://msdn.microsoft.com/pt-br/library/bb514232(v=vs.90).aspx>. Acesso em: 15 out. 2013.

MICROSOFT DEVELOPER NETWORK – MSDN**. Visão geral do ASP.NET**. Disponível em: <http://msdn.microsoft.com/pt-br/library/4w3ex9c2(v=vs.90).aspx>. Acesso em: 15 out. 2013.

MICROSOFT DEVELOPER NETWORK – MSDN. **Web Services.** Disponível em: <http://msdn.microsoft.com/pt-br/library/cc564893.aspx>. Acesso em: 05 out. 2013.

MICROSOFT WINDOWS PHONE DESIGN. **Como podemos projetar**. Disponível em: <http://dev.windowsphone.com/en-us/design>. Acesso em: 02 ago. 2013.

MICROSOFT WINDOWS PHONE DESIGN – **Principles.** Disponível em: <http://cmsresources.windowsphone.com/devcenter/en-us/design/Principles>. Acesso em: 10 ago. 2013.

MICROSOFT WINDOWS PHONE DESIGN. **Principles**. Disponível em: <http://cmsresources.windowsphone.com/devcenter/en-us/design/Principles/Info_1.png> Acesso em 10 ago. 2013., il. color.

MICROSOFT WINDOWS PHONE DESIGN. **Principles**. Disponível em: <http://cmsresources.windowsphone.com/devcenter/en-us/design/Principles/Info_2.png>. Acesso em: 10 ago 2013., il. color.

NETHERLAND, Wynn. **RestSharp: Simple REST and HTTP API Client for .NET.** Disponível em: <http://thechangelog.com/restsharp-simple-rest-and-http-api-client-for-net/>. Acesso em: 23 nov. 2013.

PEW INTERNET. **Pew Research Center's Internet & American Life Project.** Disponível em: <http://www.pewinternet.org/>. Acesso em: 30 ago. 2013.

MONACO, Thiago; CARMO, Rodolpho Marques. **Desenvolvendo aplicativos para Windows Phone**. 1ª ed. Rio de Janeiro: Brasport. 2012.

OLHAR DIGITAL. **O que diferencia os celulares dos *smartphones*.** Disponível em: <http://olhardigital.uol.com.br/video/o-que-diferencia-os-celulares-dos-*smartphones*/14727>. Acesso em: 15 ago. 2013.

PEREIRA, Halex. **Dica de leitura: a história ilustrada do iOS.** Disponível em: <http://macmagazine.com.br/2011/12/13/dica-de-leitura-a-historia-ilustrada-do-ios/>. Acesso em: 13 fev. 2013.

REVISTA GALILEU. **Microsoft lança Windows Phone 7, sistema operacional para celulares.** Disponível em: <http://revistagalileu.globo.com/Revista/Common/0,,EMI178812-17770,00- MICROSOFT+LANCA+WINDOWS+ PHONE+SISTEMA+OPERACIONAL+ PARA+CELULARES.html>. Acesso em: 20 nov. 2013.

ROBINSON, Simon et al. **Professional C # - Programando:** De Programador para Programador.Tradução de Flavia Barktevicius Cruz, Revisão Técnica de Ulisses Ponticelli Giorgi. São Paulo: Pearson Education do Brasil, 2004. 1124 p.

SANTOS, Luis Carlos dos. Microsoft **Visual C# 2008 Express Edition:** Aprenda na prática. 2. ed. São Paulo: Editora Erica, 2011. 240 p.

SAFATLI, Nabil. **Devmedia:** Introdução ao Desenvolvimento em Windows Phone. Disponível em: <http://www.devmedia.com.br/introducao-ao-desenvolvimento-em-windows-phone/26642>. Acesso em: 01 ago. 2013.

SANTANA FILHO, Ozeas Vieira; ZARA, Pedro Marcelo. **Microsoft.NET:** Uma visão geral para programadores. São Paulo: Editora Senac, 2002. 129 p.

SHARP, John. **Microsoft Visual C# 2010**: Série Passo a Passo. São Paulo: Bookman, 2011. 776 p.

SHEPERD, George. **Microsoft ASP. NET 2.0:** Passo a passo. São Paulo: Bookman, 2007. 408 p.

TADASHI, Alexandre. **Introdução ao Windows Phone 7**. Disponível em: <http://www.linhadecodigo.com.br/artigo/3168/introducao-ao-windows-phone-7.aspx>. Acesso em: 29 ago. 2013.

TUCKER, Allen; NOONAN, Robert**. *Programming Languages*: Principles and paradigms**. Boston: McGraw-Hill, 2002. 411 p.

WHITECHAPEL, Andrew; McKENNA, Sean, **Windows Phone 8 Development Internals**. 1ª Ed. Sebastopol. Microsoft Press: 2013. 1044 p.

WILDE, Eric; PAUTASSO, Cesare. **REST:** From Research to Practice. 2011. ed. New York: Springer, 2011. 528 p.

WILDERMUTH, Shawn. **Essential Windows Phone 7.5**: Application Development with Silverlight, 1ª Ed. Addison-Wesley Professional: 2011. 512 p.

WINDOWS PHONE 8. **Creating Hello World App**. Disponível em: <http://versatilias.wordpress.com/2013/01/20/windows-phone-8-creating-hello-world-app/>. Acesso em 08 jun. 2013., il. color.

WINDOWS PHONE 8. **Creating Hello World App**. Disponível em: <http://versatilias.files.wordpress.com/2013/01/1-create-project.jpg?w=710>. Acesso em 08 jun. 2013., il. color.

WINDOWS PHONE 8. **Creating Hello World**. Disponível em: <http://versatilias.files.wordpress.com/2013/01/2-select-os-version.jpg?w=507>. Acesso em 08 jun. 2013., il. color.

WINDOWS PHONE 8. **Creating Hello World**. Disponível em: <http://versatilias.files.wordpress.com/2013/01/9-unclicked.jpg?w=322>. Acesso em 08 jun. 2013., il. color.